Bölüm 14: Çizgeler

Veri Yapıları

Temel kavramlar | Derece, yol, döngü | Komşuluk listesi ve matrisi | BFS | DFS | Ağırlıksız en kısa yol | Dijkstra | Kruskal ve Prim

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Temel kavramlar

Düğüm, kenar, yön, ağırlık, yol ve bağlantılılık kavramlarını açıklamak.

Gösterimler

Komşuluk matrisi ile komşuluk listesini maliyetleriyle karşılaştırmak.

Gezinme ve sıralama

BFS, DFS ve DAG'lerde topolojik sıralamayı uygulayıp sonuçlarını yorumlamak.

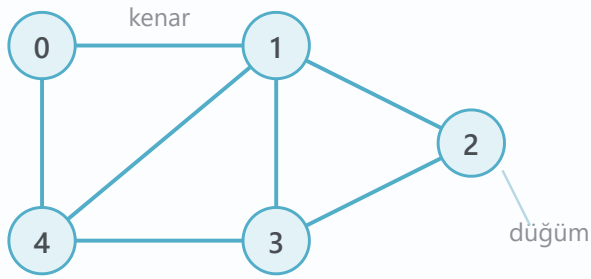
En kısa yol ve MKA

Ağırlıksız BFS, Dijkstra, Kruskal ve Prim algoritmaları için uygun problem türünü seçmek.

TEMEL KAVRAMLAR

Çizge (Graph)

- Düğümlerin (vertex / node) ve bu düğümleri birbirine bağlayan kenarların (edge) bir araya gelmesiyle oluşur.
- Karmaşık ilişkileri, ağları ve yapıları temsil etmek için kullanılır.
- Çizge teorisi, düğümler ve kenarlar arasındaki ilişkileri inceler.



Bir çizge **G = (V, E)** ikilisidir: **V** düğümler kümesi, **E** kenarlar kümesi.
Örnek: V = {0, 1, 2, 3, 4}, E = {(0,1), (0,4), (1,2), (1,3), (1,4), (2,3), (3,4)} · n = |V| = 5, m = |E| = 7

Ağaç ile farkı
Kök yoktur, döngü olabilir, bir düğüme birden çok yoldan ulaşılabilir.

İlişki türü
Çizge "çoğa-çok" ilişkiyi modeller.

Notasyon
n = düğüm sayısı, m = kenar sayısı.

TEMEL KAVRAMLAR

Düğüm (Vertex, Node)

- Çizgenin temel yapı taşıdır.
- Şehirler, kişiler, bilgisayarlar gibi nesneleri temsil eder.
- Düğüm, bir değer (etiket, ad, konum…) saklayabilir.
- **Örnek:** haritadaki şehirler ve kavşaklar.

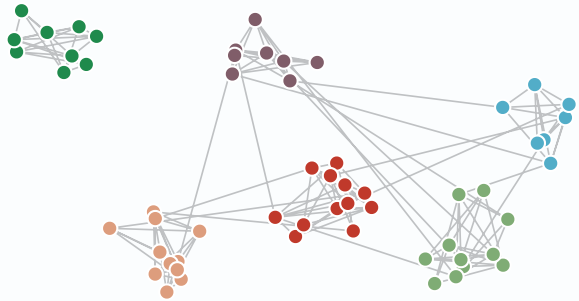


Karayolu haritası: şehirler düğüm, yollar kenar

Harita çizge olarak modellenince "İstanbul'dan Van'a en kısa yol hangisi?" sorusu bir **çizge algoritması** problemine dönüşür (ör. Dijkstra algoritması).

Kenar (Edge)

- Düğümleri birbirine bağlayan bağlantılardır.
- Kenarlar düğümler arasındaki ilişkiyi temsil eder.
- (v, w) bir kenar ise v ve w **bitişik** (adjacent) düğümlerdir; her biri ötekinin komşusudur; kenar bu iki düğüme **bağlıdır** (incident).
- **Örnek:** şehirler arasındaki yollar, bilgisayar ağındaki bağlantılar, sosyal ağdaki arkadaşlıklar.

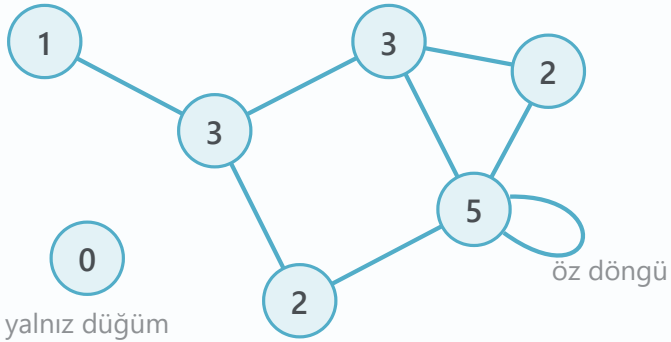


Bir sosyal ağ: kişiler düğüm, ilişkiler kenar

TEMEL KAVRAMLAR

Derece (Degree)

- Bir düğümün derecesi, ona bağlı kenar sayısıdır: `deg(v)`.
- Derecesi 0 olan (yalnız, isolated) düğümler de olabilir.
- Öz döngü (self-loop) dereceye **2** katkı yapar.
- **Örnek:** bir kavşaktaki yolların sayısı.



Her düğümün içinde kendi derecesi yazıyor

El sıkışma lemması

Her kenar iki uca birer derece verir:

`Σ deg(v) = 2m`

Örnekte: 1+3+3+2+5+2+0 = 16 = 2·8 kenar.

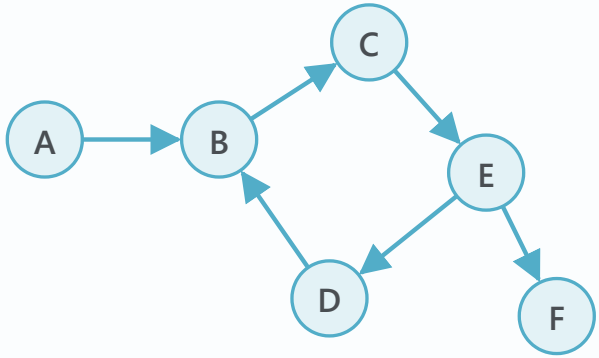
Yönlü çizgede

giriş derecesi (in-degree): gelen ok sayısı, *çıkış derecesi* (out-degree): giden ok sayısı.

`Σ giriş = Σ çıkış = m`

Yönlü Çizge (Digraph)

- Kenarlar bir düğümden diğerine **belirli bir yöndedir**: (u, v) ≠ (v, u).
- Geri dönüş yolu olmayabilir: örnekte B'den A'ya ulaşılamaz.
- Yönlü çizge (directed graph), kısaca *digraph* olarak da adlandırılır.
- **Örnek:** web sayfaları ve bağlantıları, ders ön koşulları, tek yönlü sokaklar.

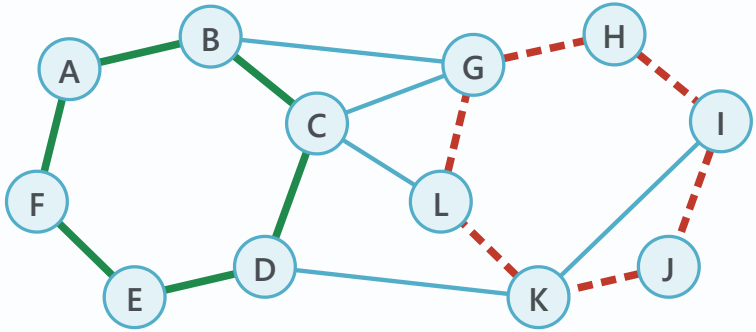


Düğüm	A	B	C	D	E	F
giriş derecesi	0	2	1	1	1	1
çıkış derecesi	1	1	1	1	2	0

TEMEL KAVRAMLAR

Yol (Path) ve Döngü (Cycle)

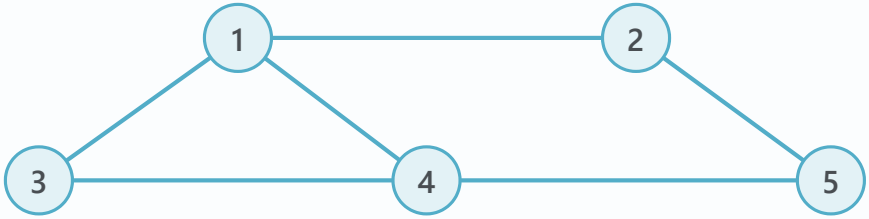
- **Yol (path):** ardışık düğümleri kenarlarla bağlanan düğüm dizisi.
- **Basit yol:** hiçbir düğümü tekrar etmeyen yol.
- **Döngü (cycle):** aynı düğümde başlayıp biten yol.
- **Basit döngü:** başlangıç düğümü dışında hiçbir düğümü tekrar etmez.
- Çizgelerde döngü olabilir; **ağaçta döngü yoktur**.



Yeşil: A-B-C-D-E-F-A · Kırmızı (kesikli): G-H-I-J-K-L-G

Örnek: uçuş rotası. İstanbul → Ankara → İzmir → İstanbul bir döngüdür; hiçbir şehre iki kez uğramadan dönüyorsa basit döngüdür.

Örnek Çizge

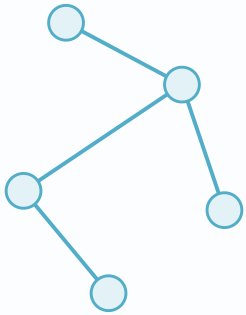


- **V** = {1, 2, 3, 4, 5}
- **E** = {(1,2), (1,3), (1,4), (2,5), (3,4), (4,5)}
- Düğüm 1 ve düğüm 2 **komşu**.
- deg(1) = 3, deg(4) = 3, diğerleri 2 → toplam 12 = 2·6 ✓

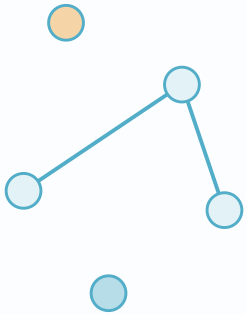
Dizi	Türü
1 → 2 → 5	basit yol
1 → 3 → 4 → 1	basit döngü
1 → 3 → 4 → 1 → 2	basit olmayan yol (1 tekrar)
1 → 3 → 4 → 1 → 4 → 1	basit olmayan döngü

Bağlantılı ve Tam Çizge

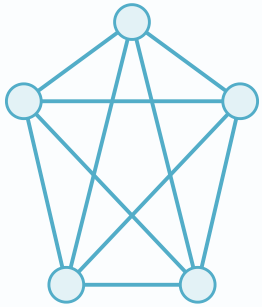
- **Bağlantılı (connected) çizge:** her iki farklı düğüm çifti arasında bir *yol* vardır.
- **Bağlantılı bileşen (connected component):** birbirine yolla ulaşabilen düğümlerin en büyük grubu.
- **Tam (complete) çizge:** her iki farklı düğüm çifti arasında bir *kenar* vardır.



Bağlantılı: 1 bileşen



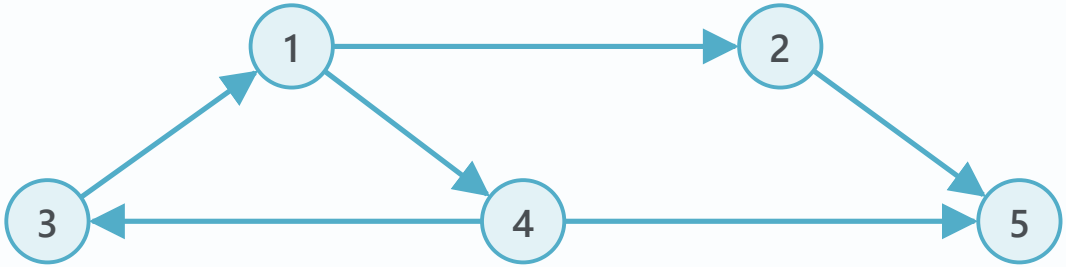
Bağlantısız: 3 bileşen



Tam çizge K_5: 10 kenar

n düğümlü basit yönsüz çizgede $m \le n(n-1)/2$; bu üst sınıra tam çizgede ulaşılır. Bağlantılı bir çizgede $m \ge n - 1$.

Yönlü Çizge Örneği



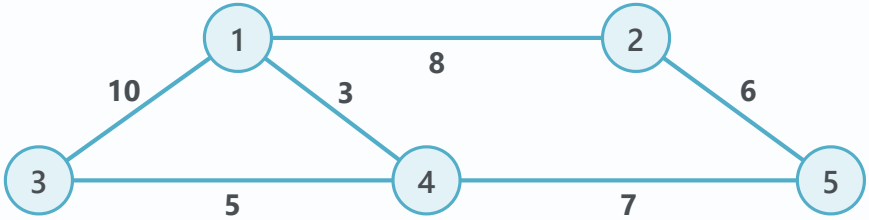
- **V** = {1, 2, 3, 4, 5}
- **E** = {(1,2), (1,4), (2,5), (4,5), (3,1), (4,3)}
- Düğüm 2, düğüm 1'in **komşusu** (1 → 2 kenarı var).
- Düğüm 1, düğüm 2'nin komşusu **değil!**

Dizi	Türü
1 → 2 → 5	yönlü yol
1 → 4 → 3 → 1	yönlü döngü
5 → …	5'ten çıkan kenar yok

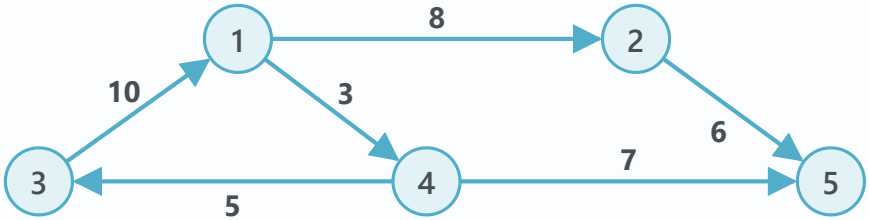
Yönlü kenar (u, v) sıralı bir çifttir: u kuyruk (başlangıç), v baş (hedef) düğümdür.

Ağırlıklı Çizge

- Her kenara bir sayı (**ağırlık**, weight) atanır: uzaklık, süre, maliyet, kapasite…
- Bir yolun uzunluğu = üzerindeki kenar ağırlıklarının toplamı. Yönsüz veya yönlü olabilir.



Ağırlıklı yönsüz: $3 \to 4 \to 1$ yolu $5 + 3 = 8$



Ağırlıklı yönlü: $1 \to 4 \to 3$ yolu $3 + 5 = 8$

Doğrudan kenar her zaman en kısa yol değildir: yönsüz çizgede 1–3 kenarı 10, ama $1 \to 4 \to 3$ yolu yalnızca **8**.

TEMEL KAVRAMLAR

Çizge Soyut Veri Tipi (Graph ADT)

Goodrich'teki `Graph` arayüzünün temel işlemleri (V: düğüm, E: kenar):

İşlem	Açıklama
`numVertices()`, `numEdges()`	Düğüm sayısı n, kenar sayısı m
`vertices()`, `edges()`	Tüm düğümler / tüm kenarlar üzerinde gezinme
`getEdge(u, v)`	u'dan v'ye kenar varsa onu, yoksa null döndürür
`endVertices(e)`, `opposite(v, e)`	Kenarın iki ucu / e kenarının v dışındaki ucu
`outDegree(v)`, `inDegree(v)`	Çıkış / giriş derecesi (yönsüzde ikisi eşit)
`outgoingEdges(v)`, `incomingEdges(v)`	v'den çıkan / v'ye gelen kenarlar
`insertVertex(x)`, `insertEdge(u, v, x)`	Yeni düğüm / yeni kenar ekler
`removeVertex(v)`, `removeEdge(e)`	Düğümü (bağlı kenarlarıyla) / kenarı siler

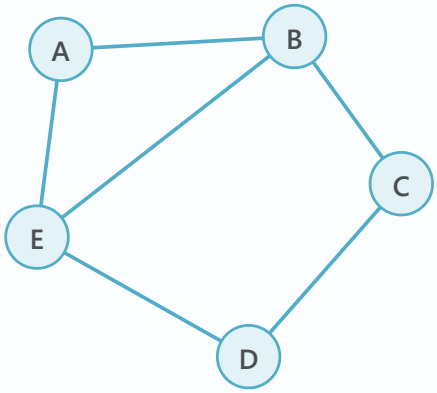
Bu işlemlerin maliyeti, çizgenin bellekte nasıl temsil edildiğine bağlıdır.

Çizge Nasıl Saklanır? Üç Temel Gösterim

Kenar listesi
Tüm kenarlar tek bir listede tutulur.
`[(A,B), (A,E), …]`

Komşuluk listesi
Her düğüm için komşularının listesi tutulur.
`V[i] = {j, k, l}`

Komşuluk matrisi
n×n iki boyutlu dizi; hücre, kenarın varlığını gösterir.
`M[i][j] = 1`



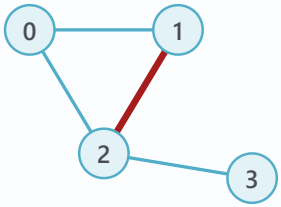
Kenar listesi (edge list) en basit gösterimdir:

`(A, B)` `(A, E)` `(B, C)` `(B, E)` `(C, D)` `(D, E)`

- Alan O(n + m); kenar eklemek O(1).
- Ama "u ile v komşu mu?" ya da "u'nun komşuları kim?" için tüm listeyi taramak gerekir: **O(m)**.

Komşuluk Matrisi (Adjacency Matrix)

- n×n boyutunda bir dizidir: `matris[i][j]` = 1, düğüm i'den j'ye kenar varsa; yoksa 0.
- **Ağırlıklı çizgede** `matris[i][j]` kenarın ağırlığını tutar; kenar yoksa `∞` (veya özel bir değer).
- **Yönsüz çizgede** matris simetriktir: `matris[i][j] == matris[j][i]`.
- Alan karmaşıklığı **O(n^2)**; düğüm sayısı arttıkça bellek hızla büyür.



	0	1	2	3	
0	0	1	1	0	
1	1	0	1	0	← m[1][2]: tek hücre, O(1)
2	1	1	0	1	← 2'nin komşuları: tüm satır, O(n)
3	0	0	1	0	

simetrik: m[i][j] = m[j][i]

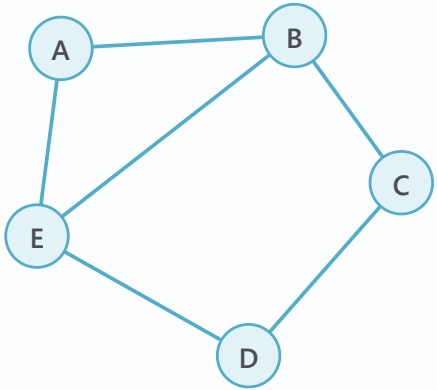
Kolay

+ İki düğüm arasında kenar var mı? **O(1)**
+ Kenar ekleme / silme O(1)

Zor

– Bir düğümün komşularını bulmak: tüm satır, **O(n)**
– Seyrek çizgede hücrelerin çoğu boş (0)

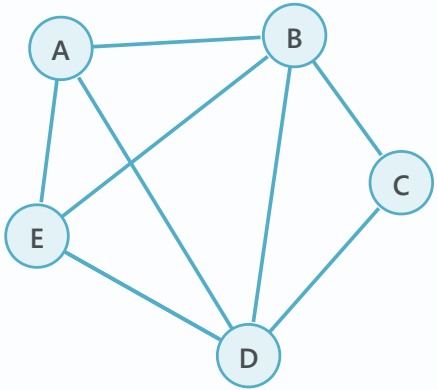
Komşuluk Matrisi: Üç Örnek



	A	B	C	D	E
A	0	**1**	0	0	**1**
B	**1**	0	**1**	0	**1**
C	0	**1**	0	**1**	0
D	0	0	**1**	0	**1**
E	**1**	**1**	0	**1**	0

Yönsüz çizge: her kenar iki hücreye yazılır, matris köşegene göre **simetrik**. Satır toplamı = derece (B: 3).

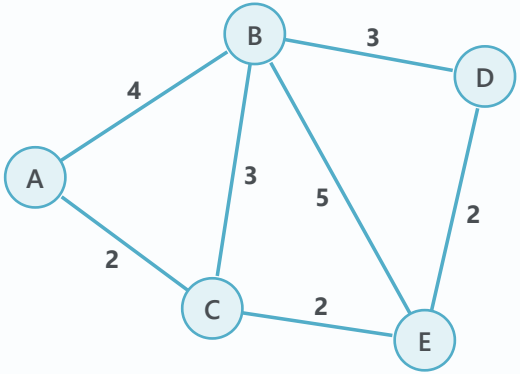
Komşuluk Matrisi: Üç Örnek



	A	B	C	D	E
A	0	**1**	0	0	**1**
B	0	0	**1**	**1**	0
C	0	0	0	0	0
D	**1**	0	**1**	0	**1**
E	0	**1**	0	**1**	0

Yönlü çizge: satır = kaynak, sütun = hedef. Simetrik değil; satır toplamı çıkış, sütun toplamı giriş derecesi.

Komşuluk Matrisi: Üç Örnek

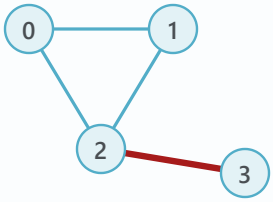


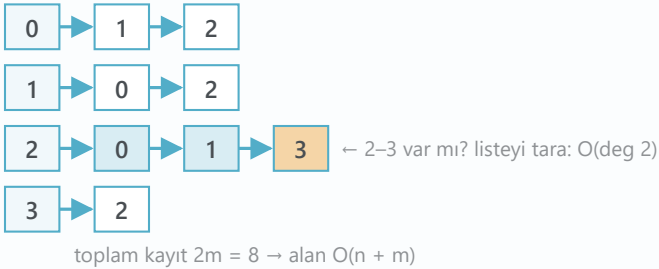
	A	B	C	D	E
A	0	**4**	∞	∞	∞
B	∞	0	**3**	**3**	∞
C	**2**	∞	0	∞	**2**
D	∞	∞	∞	0	∞
E	∞	**5**	∞	**2**	0

Ağırlıklı çizge: hücrede ağırlık, kenar yoksa **∞** (köşegende 0). Boş hücreye 0 yazmak, 0 ağırlıklı bir kenarla karıştırılabilir.

Komşuluk Listesi (Adjacency List)

- n düğüm için n adet liste (çoğunlukla bağlı liste veya dinamik dizi) tutulur.
- `liste[i]`, düğüm i'ye komşu olan düğümleri (ağırlıklıysa ağırlıklarıyla) içerir.
- **Yönsüz çizgede** her kenar (v, w) iki listeye de eklenir: w, v'nin listesine; v, w'nin listesine.
- Alan karmaşıklığı **O(n + m)**; seyrek çizgeler için idealdir.





toplam kayıt 2m = 8 → alan O(n + m)

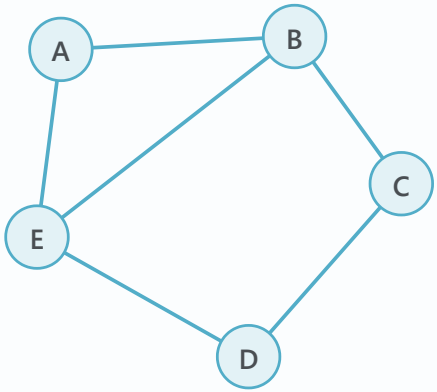
Kolay

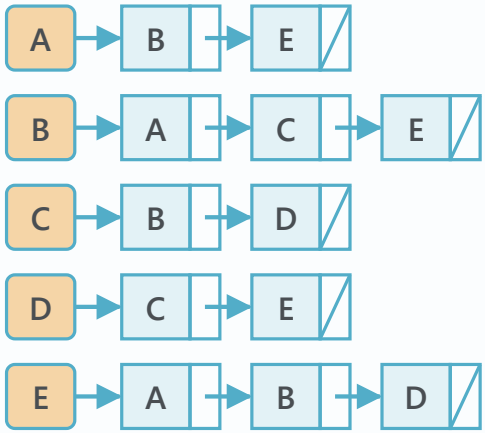
+ Bir düğümün komşularını gezmek: **O(deg(v))**
+ Kenar ekleme O(1)

Zor

– u ile v arasında kenar var mı? Listeyi taramak gerekir: **O(deg(u))**

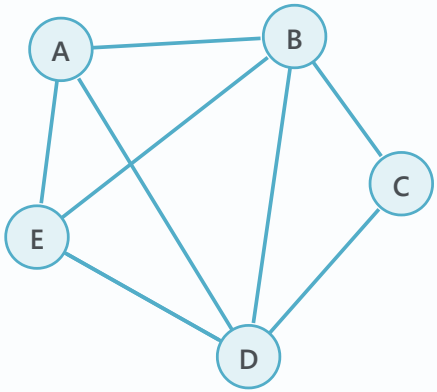
Komşuluk Listesi: Üç Örnek

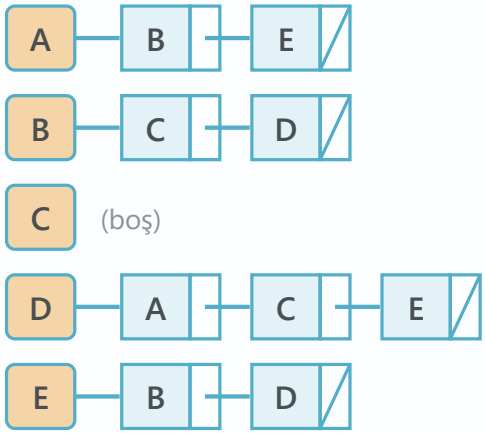




Yönsüz: 6 kenar → listelerde 12 kayıt (her kenar iki kez). Liste uzunluğu = derece.

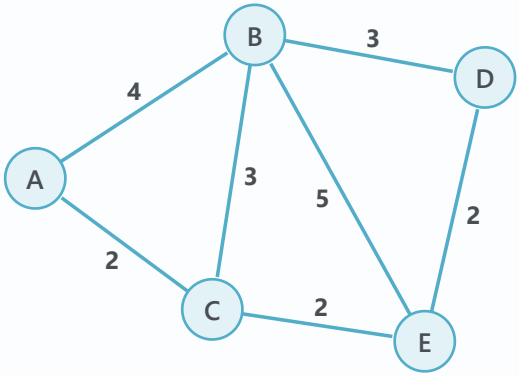
Komşuluk Listesi: Üç Örnek

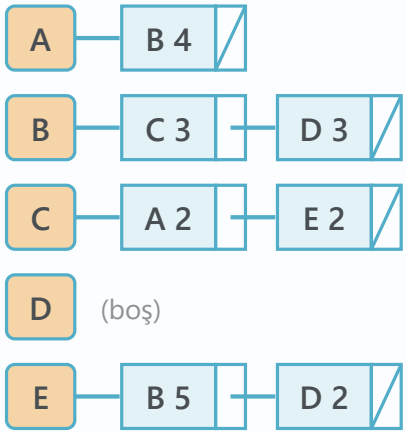




Yönlü: yalnızca çıkan kenarlar listelenir; 9 kenar → 9 kayıt. C'den çıkan kenar yok, listesi boş.

Komşuluk Listesi: Üç Örnek





Ağırlıklı: her kayıt (komşu, ağırlık) çifti tutar; ör. B'nin listesi: (C, 3), (D, 3).

Gösterimlerin Bellek ve İşlem Maliyeti

İşlem (n düğüm, m kenar)	Kenar listesi	Komşuluk listesi	Komşuluk matrisi
Bellek	O(n + m)	O(n + m)	$O(n^2)$
u–v kenarı var mı? `getEdge`	O(m)	yönsüz: O(min(deg u, deg v)) yönlü u→v: O(outDeg u)	**O(1)**
v'nin komşularını gez	O(m)	**O(deg v)**	O(n)

Seyrek çizge (m ≈ n)

Yol ağları, sosyal ağlar → *komşuluk listesi*. n = 10^6 düğümde matris 10^{12} hücre ister!

Yoğun çizge (m ≈ n^2)

Sık "kenar var mı?" sorgusu → *komşuluk matrisi*.

Java ile Komşuluk Listesi

```java
class Cizge {
    private final int n;                                    // düğümler: 0..n-1
    private final List<List<Integer>> komsu = new ArrayList<>();
    Cizge(int n) {
        this.n = n;
        for (int i = 0; i < n; i++)
            komsu.add(new ArrayList<>());
    }
    void kenarEkle(int u, int v) {
        komsu.get(u).add(v);
        komsu.get(v).add(u);                  // yönlü çizgede bu satır yok
    }
    List<Integer> komsular(int u) { return komsu.get(u); }   // O(1)
    int derece(int u) { return komsu.get(u).size(); }
}
```

Matris karşılığı

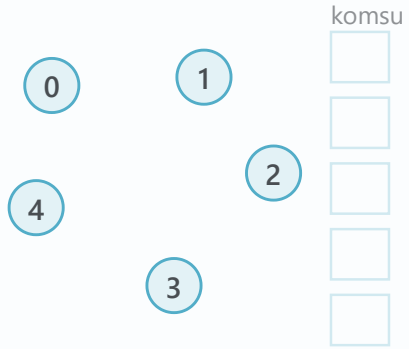
`boolean[][] m = new boolean[n][n];` kenar ekle: `m[u][v] = m[v][u] = true;` sorgu `m[u][v]` O(1).

Ağırlıklı çizge

Liste öğesi `int[]{v, w}` veya küçük bir `Kenar` sınıfı; matriste `int[][]`, kenar yoksa `Integer.MAX_VALUE`.

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
▶    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```



`new Cizge(5)` : yapıcı çağrılır; **n = 5** atanır.
`komsu` henüz boş bir dış liste.

Adım 1 / 19

ÇAĞRI YIĞINI

- `Cizge(5)`
- `main()`

DEĞİŞKENLER

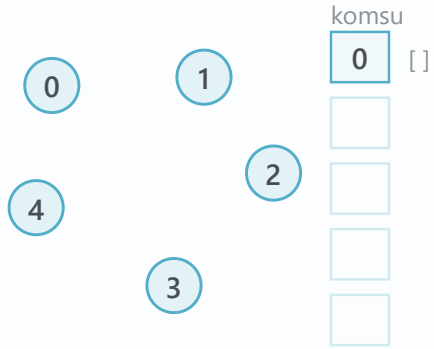
`n = 5`

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
▶    for (int i = 0; i < n; i++)
▶      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- `Cizge(5)`
- `main()`



i = 0: 0 numaralı düğüm için **boş bir iç liste** eklenir.

Adım 2 / 19

DEĞİŞKENLER

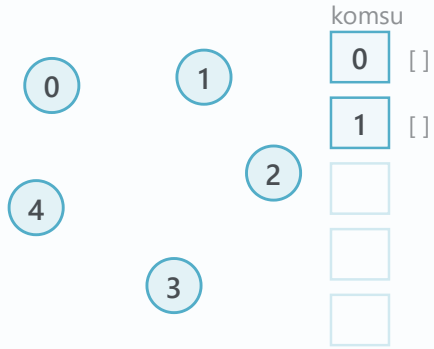
`n = 5` `i = 0`

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
▶    for (int i = 0; i < n; i++)
▶      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- `Cizge(5)`
- `main()`



i = 1: 1 numaralı düğüm için **boş bir iç liste** eklenir.

Adım 3 / 19

DEĞİŞKENLER

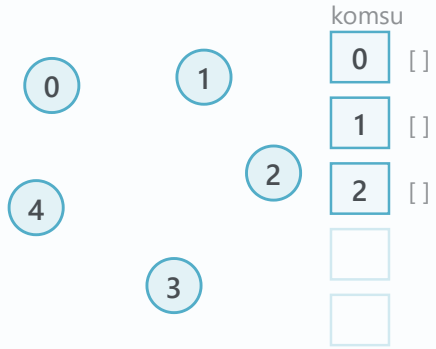
`n = 5` `i = 1`

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
▶    for (int i = 0; i < n; i++)
▶      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- `Cizge(5)`
- `main()`



i = 2: 2 numaralı düğüm için **boş bir iç liste** eklenir.

Adım 4 / 19

DEĞİŞKENLER

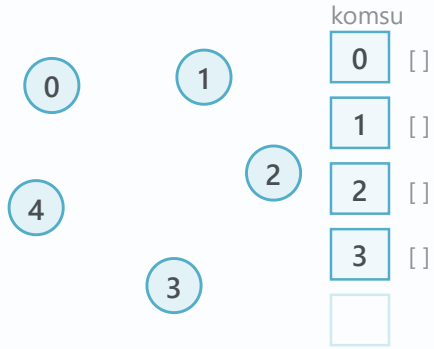
`n = 5` `i = 2`

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
▶    for (int i = 0; i < n; i++)
▶      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- `Cizge(5)`
- `main()`



i = 3: 3 numaralı düğüm için **boş bir iç liste** eklenir.

Adım 5 / 19

DEĞİŞKENLER

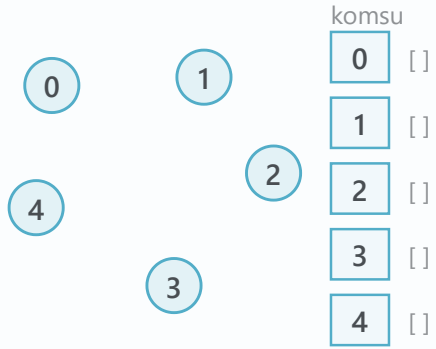
`n = 5` `i = 3`

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
▶    for (int i = 0; i < n; i++)
▶      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13             {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- **Cizge(5)**
- main()



i = 4: 4 numaralı düğüm için **boş bir iç liste** eklenir. Artık her düğümün (boş) komşu listesi var.

DEĞİŞKENLER

n = 5 i = 4

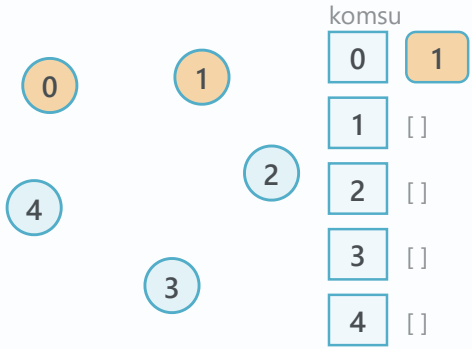
Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
▶    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

kenarEkle(0, 1)

main()



Kenar 0–1: **1**, 0'ın listesinin sonuna eklenir.

Adım 7 / 19

DEĞİŞKENLER

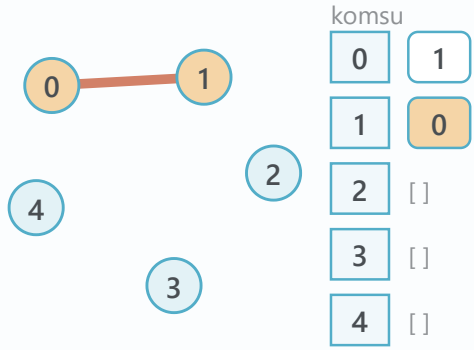
u = 0 v = 1

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
▶    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- **kenarEkle(0, 1)**
- main()



Yönsüz çizge: aynı kenar ters yönde de eklenir → **0**, 1'in listesine.

Adım 8 / 19

DEĞİŞKENLER

u = 0 v = 1

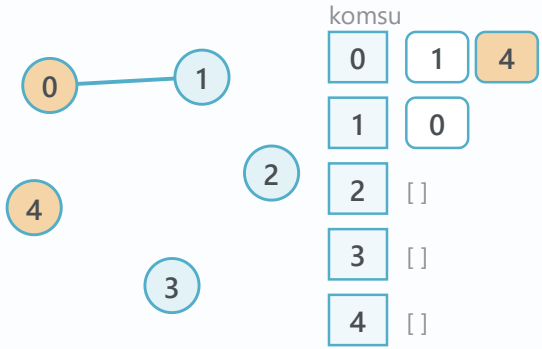
Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
▶    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

kenarEkle(0, 4)

main()



Kenar 0–4: **4**, 0'ın listesinin sonuna eklenir.

Adım 9 / 19

DEĞİŞKENLER

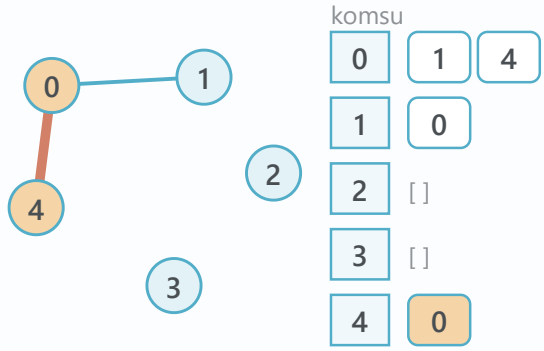
u = 0 v = 4

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
▶    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- `kenarEkle(0, 4)`
- `main()`



Yönsüz çizge: aynı kenar ters yönde de eklenir → **0**, 4'ün listesine.

Adım 10 / 19

DEĞİŞKENLER

`u = 0` `v = 4`

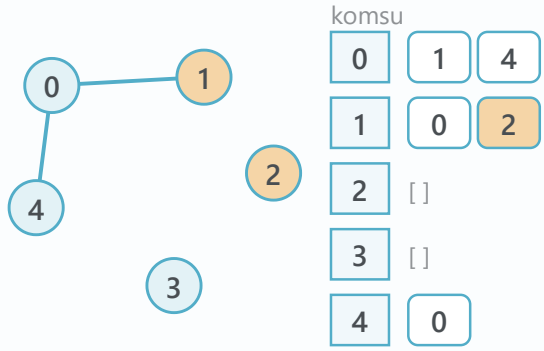
Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
▶    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

kenarEkle(1, 2)

main()



Kenar 1–2: **2**, 1'in listesinin sonuna eklenir.

Adım 11 / 19

DEĞİŞKENLER

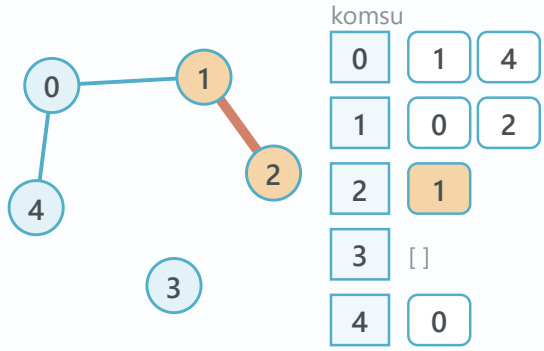
u = 1 v = 2

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
▶    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- `kenarEkle(1, 2)`
- `main()`



Yönsüz çizge: aynı kenar ters yönde de eklenir → **1**, 2'nin listesine.

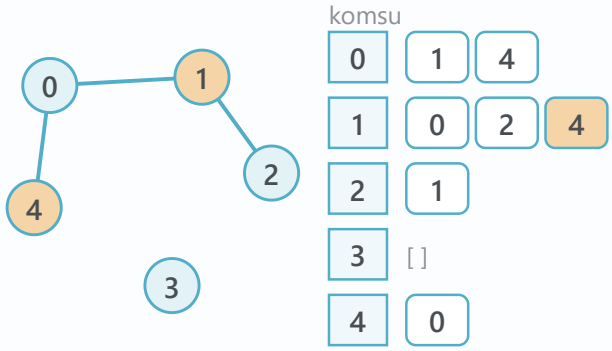
Adım 12 / 19

DEĞİŞKENLER

`u = 1` `v = 2`

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
▶    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```



Kenar 1–4: **4**, 1'in listesinin sonuna eklenir.

Adım 13 / 19

ÇAĞRI YIĞINI

- `kenarEkle(1, 4)`
- `main()`

DEĞİŞKENLER

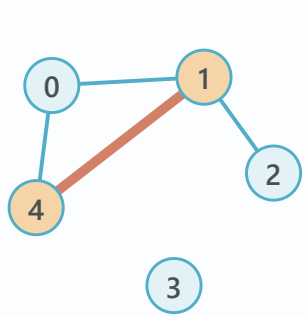
u = **1** v = **4**

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
▶    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- `kenarEkle(1, 4)`
- `main()`



komsu

0	1 4
1	0 2 4
2	1
3	[]
4	0 1

Yönsüz çizge: aynı kenar ters yönde de eklenir → **1**, 4'ün listesine.

Adım 14 / 19

DEĞİŞKENLER

`u = 1` `v = 4`

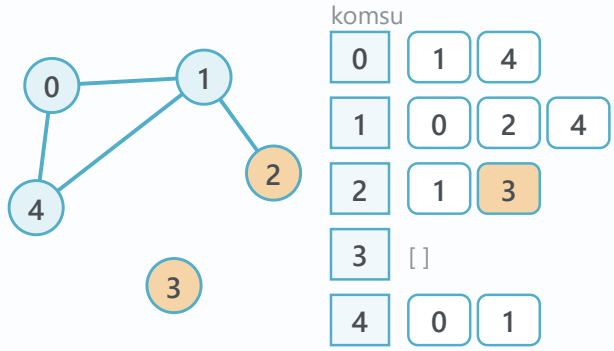
Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
▶    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

kenarEkle(2, 3)

main()



Kenar 2–3: **3**, 2'nin listesinin sonuna eklenir.

Adım 15 / 19

DEĞİŞKENLER

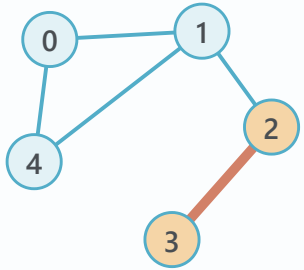
u = 2 v = 3

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
▶    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- **kenarEkle(2, 3)**
- main()



komsu

0	1	4	
1	0	2	4
2	1	3	
3	2		
4	0	1	

Yönsüz çizge: aynı kenar ters yönde de eklenir → **2**, 3'ün listesine.

Adım 16 / 19

DEĞİŞKENLER

u = **2** v = **3**

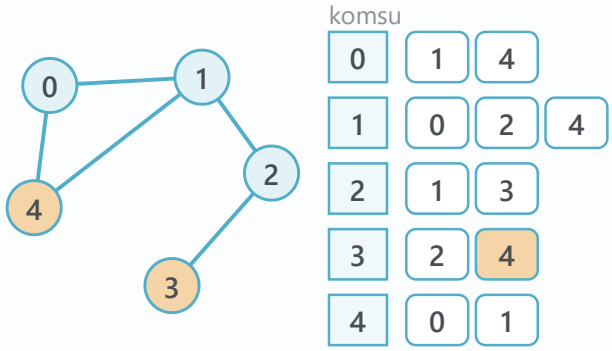
Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
▶    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

kenarEkle(3, 4)

main()



Kenar 3–4: **4**, 3'ün listesinin sonuna eklenir.

Adım 17 / 19

DEĞİŞKENLER

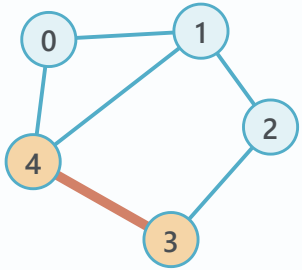
u = 3 v = 4

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
▶    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13              {1,4}, {2,3}, {3,4}};
14 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

ÇAĞRI YIĞINI

- **kenarEkle(3, 4)**
- main()



komsu

0	1	4	
1	0	2	4
2	1	3	
3	2	4	
4	0	1	3

Yönsüz çizge: aynı kenar ters yönde de eklenir → **3**, 4'ün listesine.

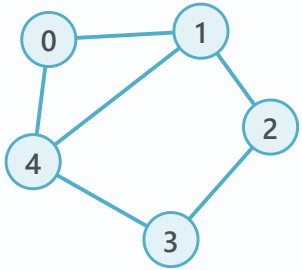
Adım 18 / 19

DEĞİŞKENLER

u = **3** v = **4**

Kod İzleme: Komşuluk Listesi Kurma

```
1  Cizge(int n) {
2    this.n = n;
3    for (int i = 0; i < n; i++)
4      komsu.add(new ArrayList<>());
5  }
6  void kenarEkle(int u, int v) {
7    komsu.get(u).add(v);
8    komsu.get(v).add(u);  // yönlüde yok
9  }
10 // main:
11 Cizge g = new Cizge(5);
12 int[][] E = {{0,1}, {0,4}, {1,2},
13             {1,4}, {2,3}, {3,4}};
▶  for (int[] e : E) g.kenarEkle(e[0], e[1]);
```



komsu

0	1	4	
1	0	2	4
2	1	3	
3	2	4	
4	0	1	3

Bitti: 6 kenar → listelerde **2·6 = 12** kayıt. Her `kenarEkle` O(1); bellek O(n + m). derece(1) = liste uzunluğu = 3.

Adım 19 / 19

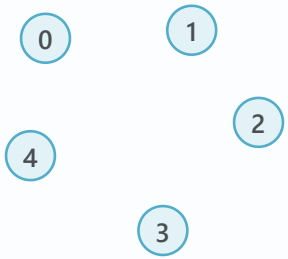
ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

—

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

m[satır][sütun] (1 = true)

5×5 **boolean** matris oluşturulur; Java tüm hücreleri **false** (0) ile başlatır. Bellek $O(n^2) = 25$ hücre.

Adım 1 / 14

```
▶ boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
3   m[u][v] = true;
4   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

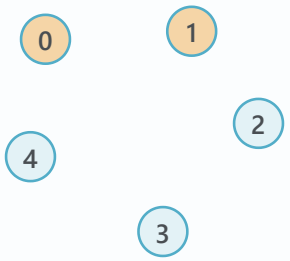
DEĞİŞKENLER

`n = 5`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	**1**	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

m[satır][sütun] (1 = true)

Kenar 0–1: **m[0][1] = true** (satır 0, sütun 1).

Adım 2 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
▶   m[u][v] = true;
4   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

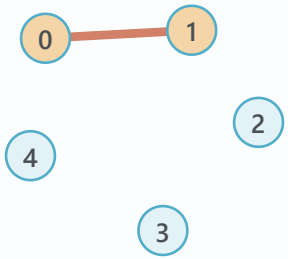
`u = 0` `v = 1`

ÇAĞRI YIĞINI

`kenarEkle(0, 1)`

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	0
1	1	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

m[satır][sütun] (1 = true)

Simetrik hücre: **m[1][0] = true**. Yönsüz çizgenin matrisi köşegene göre simetriktir.

Adım 3 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
3   m[u][v] = true;
▶   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

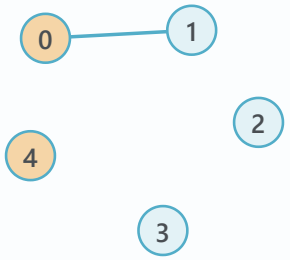
`u = 0` `v = 1`

ÇAĞRI YIĞINI

`kenarEkle(0, 1)`

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

m[satır][sütun] (1 = true)

Kenar 0–4: **m[0][4] = true** (satır 0, sütun 4).

Adım 4 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
▶   m[u][v] = true;
4   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

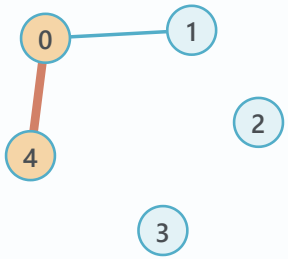
`u = 0` `v = 4`

ÇAĞRI YIĞINI

`kenarEkle(0, 4)`

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	1	0	0	0	0

m[satır][sütun] (1 = true)

Simetrik hücre: **m[4][0] = true**. Yönsüz çizgenin matrisi köşegene göre simetriktir.

Adım 5 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
3   m[u][v] = true;
▶   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

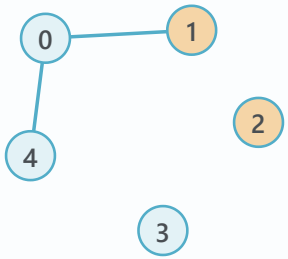
u = 0 v = 4

ÇAĞRI YIĞINI

kenarEkle(0, 4)

main()

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	**1**	0	0	**1**
1	**1**	0	**1**	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	**1**	0	0	0	0

m[satır][sütun] (1 = true)

Kenar 1–2: **m[1][2] = true** (satır 1, sütun 2).

Adım 6 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
▶   m[u][v] = true;
4   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

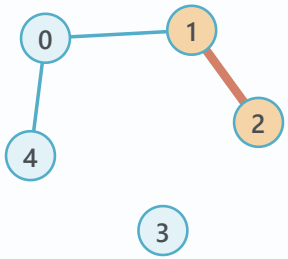
`u = 1` `v = 2`

ÇAĞRI YIĞINI

`kenarEkle(1, 2)`

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	0
2	0	1	0	0	0
3	0	0	0	0	0
4	1	0	0	0	0

m[satır][sütun] (1 = true)

Simetrik hücre: **m[2][1] = true**. Yönsüz çizgenin matrisi köşegene göre simetriktir.

Adım 7 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
3   m[u][v] = true;
▶   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

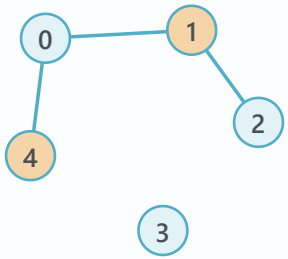
`u = 1` `v = 2`

ÇAĞRI YIĞINI

`kenarEkle(1, 2)`

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	**1**	0	0	**1**
1	**1**	0	**1**	0	**1**
2	0	**1**	0	0	0
3	0	0	0	0	0
4	**1**	0	0	0	0

m[satır][sütun] (1 = true)

Kenar 1–4: **m[1][4] = true** (satır 1, sütun 4).

Adım 8 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
▶   m[u][v] = true;
4   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

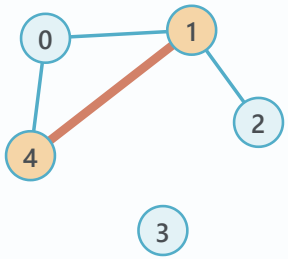
`u = 1` `v = 4`

ÇAĞRI YIĞINI

`kenarEkle(1, 4)`

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	1
2	0	1	0	0	0
3	0	0	0	0	0
4	1	1	0	0	0

m[satır][sütun] (1 = true)

Simetrik hücre: **m[4][1] = true**. Yönsüz çizgenin matrisi köşegene göre simetriktir.

Adım 9 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
3   m[u][v] = true;
▶   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

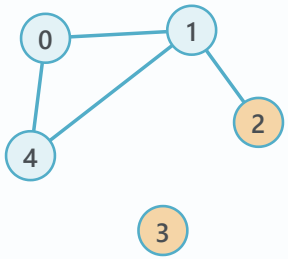
u = 1 v = 4

ÇAĞRI YIĞINI

kenarEkle(1, 4)

main()

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	1
2	0	1	0	1	0
3	0	0	0	0	0
4	1	1	0	0	0

m[satır][sütun] (1 = true)

Kenar 2–3: **m[2][3] = true** (satır 2, sütun 3).

Adım 10 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
▶   m[u][v] = true;
4   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

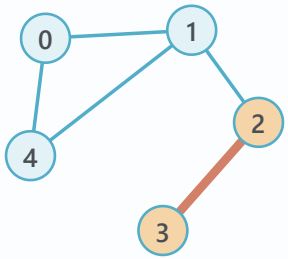
`u = 2` `v = 3`

ÇAĞRI YIĞINI

`kenarEkle(2, 3)`

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	1	1	0	0	0

m[satır][sütun] (1 = true)

Simetrik hücre: **m[3][2] = true**. Yönsüz çizgenin matrisi köşegene göre simetriktir.

Adım 11 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
3   m[u][v] = true;
▶   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

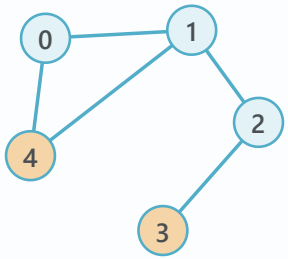
u = 2 v = 3

ÇAĞRI YIĞINI

kenarEkle(2, 3)

main()

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	1
2	0	1	0	1	0
3	0	0	1	0	1
4	1	1	0	0	0

m[satır][sütun] (1 = true)

Kenar 3–4: **m[3][4] = true** (satır 3, sütun 4).

Adım 12 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
▶   m[u][v] = true;
4   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

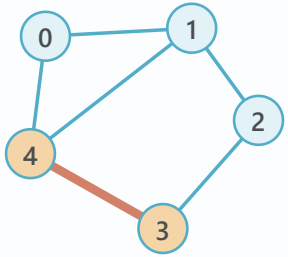
`u = 3` `v = 4`

ÇAĞRI YIĞINI

`kenarEkle(3, 4)`

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	1
2	0	1	0	1	0
3	0	0	1	0	1
4	1	1	0	1	0

m[satır][sütun] (1 = true)

Simetrik hücre: **m[4][3] = true**. Yönsüz çizgenin matrisi köşegene göre simetriktir.

Adım 13 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
3   m[u][v] = true;
▶   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
7 for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

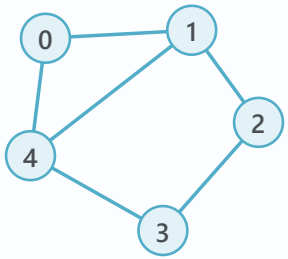
`u = 3` `v = 4`

ÇAĞRI YIĞINI

`kenarEkle(3, 4)`

`main()`

Kod İzleme: Komşuluk Matrisi Kurma



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	1
2	0	1	0	1	0
3	0	0	1	0	1
4	1	1	0	1	0

m[satır][sütun] (1 = true)

Bitti: 12 hücre true, 13'ü boşa false. "2–4 kenarı var mı?" → `m[2][4]` tek erişim, **O(1)**; ama 2'nin komşularını bulmak için tüm satır taranır: O(n).

Adım 14 / 14

```
1 boolean[][] m = new boolean[n][n];
2 void kenarEkle(int u, int v) {
3   m[u][v] = true;
4   m[v][u] = true;  // yönlü çizgede yok
5 }
6 // main: aynı 6 kenar (n = 5)
▶ for (int[] e : E) g.kenarEkle(e[0], e[1]);
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

GEZİNME (BFS / DFS)

Gezinme Algoritmaları

Gezinme (traversal): bir başlangıç düğümünden ulaşılabilen **tüm düğümleri, her birini bir kez** sistematik olarak ziyaret etmek.

Genişlik Öncelikli Arama (BFS)

Breadth-First Search. Önce tüm yakın komşular, sonra onların komşuları… Düzey düzey ilerler.

`yardımcı yapı: kuyruk (FIFO)`

Derinlik Öncelikli Arama (DFS)

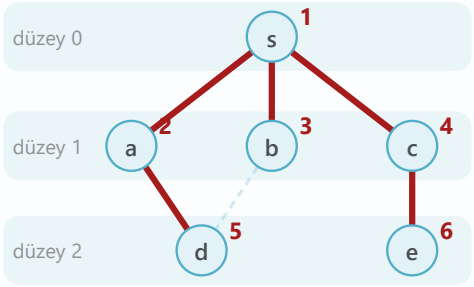
Depth-First Search. Bir yol boyunca gidebildiği kadar derine iner, çıkmazda geri döner.

`yardımcı yapı: yığın (LIFO) / özyineleme`

Ağaçtan farkı: çizgede döngü olabilir, aynı düğüme birden çok yoldan gelinebilir → **ziyaret edildi** işareti (ör. `boolean[] ziyaret`) şarttır; yoksa algoritma sonsuza kadar döner.

Genişlik Öncelikli Arama (BFS)

- Düğümleri **düzey düzey** keşfeder: önce başlangıca 1 kenar uzaklıktakiler, sonra 2 kenar uzaklıktakiler...
- **Kuyruk** yapısını kullanır: ilk keşfedilen ilk işlenir.
- Her düğüm yalnızca bir kez ziyaret edilir.



Numara = ziyaret sırası; kalın = BFS ağacı

Artıları

+ Ağırlıksız çizgede en kısa yolu (en az kenar) bulur.
+ Yakın düğümleri hızlı keşfeder.

Eksisi

– Bellek kullanımı yüksek olabilir: bir düzeyin tamamı kuyrukta bekler.

Örnek: sosyal ağda "arkadaşımın arkadaşları" (2. düzey), bir web tarayıcının (crawler) siteyi katman katman gezmesi.

BFS Algoritması

Sözde kod

```
BFS(başlangıç):
  kuyruk.ekle(başlangıç)
  başlangıç.ziyaretEdildi()
  while kuyruk boş değil:
    w = kuyruk.çıkar()
    for w'nin ziyaret edilmemiş
        her komşusu u:
      u.ziyaretEdildi()
      kuyruk.ekle(u)
```

Düğüm kuyruğa **girerken** işaretlenir; böylece aynı düğüm kuyruğa iki kez girmez.

Java

```java
void bfs(int s) {
  boolean[] ziyaret = new boolean[n];
  Queue<Integer> q = new ArrayDeque<>();
  ziyaret[s] = true;  q.add(s);
  while (!q.isEmpty()) {
    int w = q.poll();
    System.out.print(w + " ");
    for (int u : komsu.get(w))
      if (!ziyaret[u]) {
        ziyaret[u] = true;
        q.add(u);
      }
  }
}
```

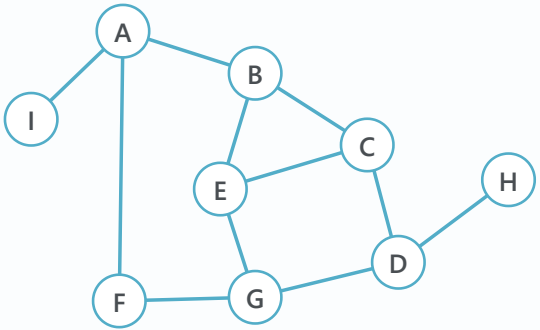
Kod İzleme: `bfs(A)`

```
1 void bfs(int s) {
▶   boolean[] ziyaret = new boolean[n];
▶   Queue<Integer> q = new ArrayDeque<>();
4   ziyaret[s] = true;  q.add(s);
5   while (!q.isEmpty()) {
6     int w = q.poll();
7     System.out.print(w + " ");
8     for (int u : komsu.get(w))
9       if (!ziyaret[u]) {
10        ziyaret[u] = true;
11        q.add(u);
12      }
13  }
14 }
```

DEĞİŞKENLER

`s = A`

ÇIKTI



q (ön → arka)

(boş)							

ziyaret[]

A	B	C	D	E	F	G	H	I
F	F	F	F	F	F	F	F	F

görülmedi · kuyrukta · işleniyor · işlendi

s = A (A=0, B=1, …, I=8; komşular alfabetik). `ziyaret` tümü false, kuyruk boş.

Adım 1 / 15

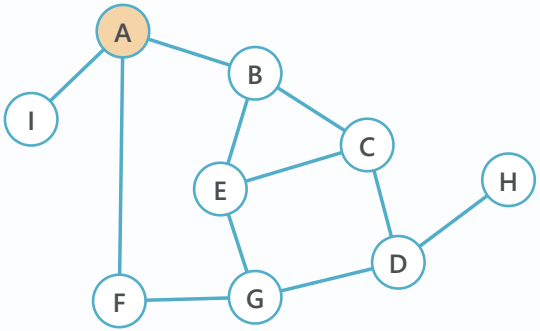
Kod İzleme: `bfs(A)`

```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
▶    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`s = A`

ÇIKTI



q (ön → arka)

A							

ziyaret[]

	A	B	C	D	E	F	G	H	I
T	F	F	F	F	F	F	F	F	F

görülmedi · kuyrukta · işleniyor · işlendi

A kuyruğa girerken işaretlenir: `ziyaret[A] = true`, q = [A].

Kod İzleme: `bfs(A)`

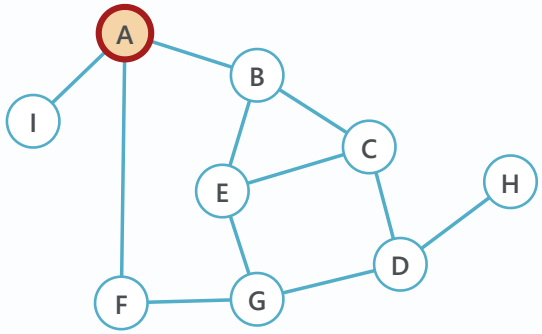
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
▶      int w = q.poll();
▶      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = A`

ÇIKTI

```
A
```



q (ön → arka)

(boş)							

ziyaret[]

	A	B	C	D	E	F	G	H	I
T	F	F	F	F	F	F	F	F	F

görülmedi · kuyrukta · işleniyor · işlendi

Kuyruğun önündeki **A** çıkarılır (poll) ve yazdırılır. Şimdi komşuları B, F, I taranır.

Kod İzleme: `bfs(A)`

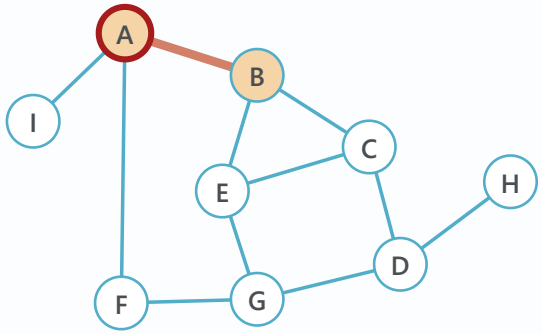
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
▶          ziyaret[u] = true;
▶          q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = A` `u = B`

ÇIKTI

```
A
```



q (ön → arka)

B							

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	F	F	F	F	F	F	F

görülmedi · kuyrukta · işleniyor · işlendi

u = B ziyaret edilmemiş → işaretlenir ve kuyruğun **arkasına** eklenir (düzey 1).

Kod İzleme: `bfs(A)`

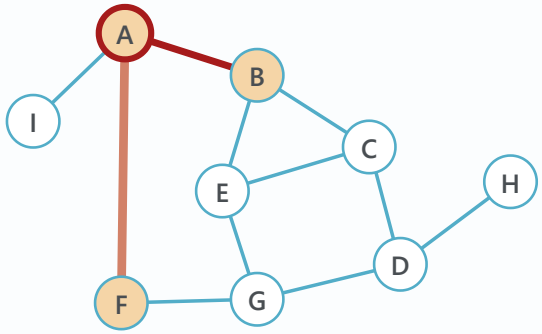
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
▶          ziyaret[u] = true;
▶          q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = A` `u = F`

ÇIKTI

```
A
```



q (ön → arka)

B	F						

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	F	F	F	T	F	F	F

görülmedi · kuyrukta · işleniyor · işlendi

u = F ziyaret edilmemiş → işaretlenir ve kuyruğun **arkasına** eklenir (düzey 1).

Kod İzleme: `bfs(A)`

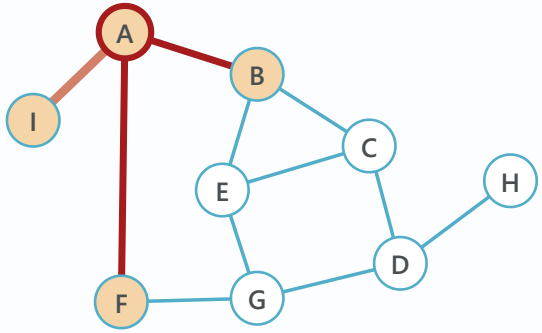
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
▶          ziyaret[u] = true;
▶          q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = A` `u = I`

ÇIKTI

```
A
```



q (ön → arka)

B	F	I					

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	F	F	F	T	F	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = I ziyaret edilmemiş → işaretlenir ve kuyruğun **arkasına** eklenir (düzey 1).

Kod İzleme: `bfs(A)`

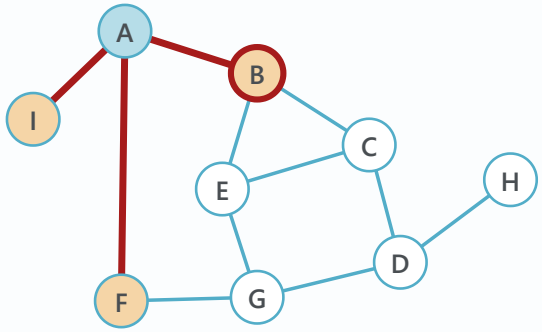
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
▶      int w = q.poll();
▶      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = B`

ÇIKTI

```
A B
```



q (ön → arka)

F	I						

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	F	F	F	T	F	F	T

görülmedi · kuyrukta · işleniyor · işlendi

Kuyruğun önündeki **B** çıkarılır (poll) ve yazdırılır. Şimdi komşuları A, C, E taranır.

Kod İzleme: `bfs(A)`

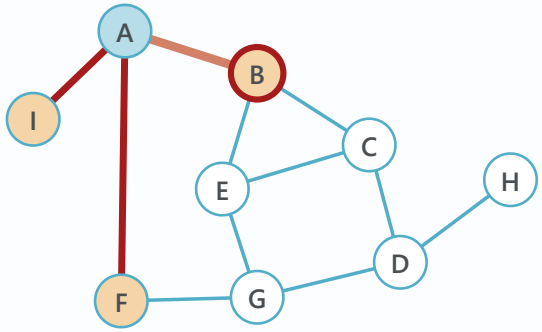
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = B` `u = A`

ÇIKTI

```
A B
```



q (ön → arka)

F	I						

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	F	F	F	T	F	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = A: zaten ziyaret edilmiş → atlanır. Ziyaret işareti olmasa kuyruğa tekrar girerdi.

Kod İzleme: `bfs(A)`

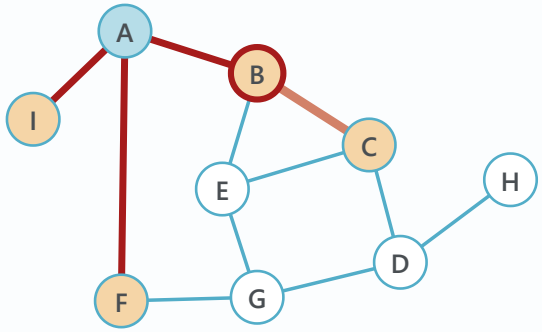
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
▶          ziyaret[u] = true;
▶          q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = B` `u = C`

ÇIKTI

```
A B
```



q (ön → arka)

F	I	C					

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	F	T	F	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = C ziyaret edilmemiş → işaretlenir ve kuyruğun **arkasına** eklenir (düzey 2).

Kod İzleme: `bfs(A)`

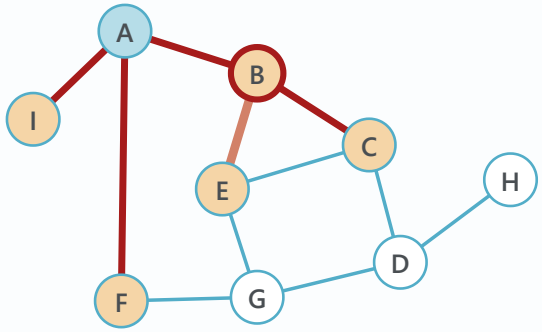
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
▶          ziyaret[u] = true;
▶          q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = B` `u = E`

ÇIKTI

```
A B
```



q (ön → arka)

F	I	C	E				

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	T	T	F	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = E ziyaret edilmemiş → işaretlenir ve kuyruğun **arkasına** eklenir (düzey 2).

Kod İzleme: `bfs(A)`

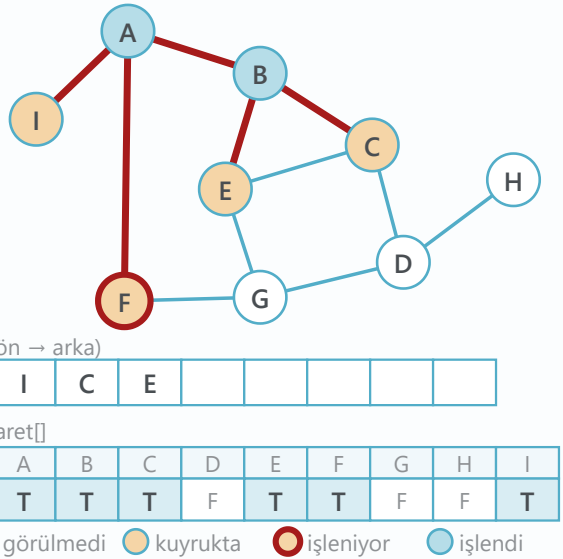
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
▶      int w = q.poll();
▶      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = F`

ÇIKTI

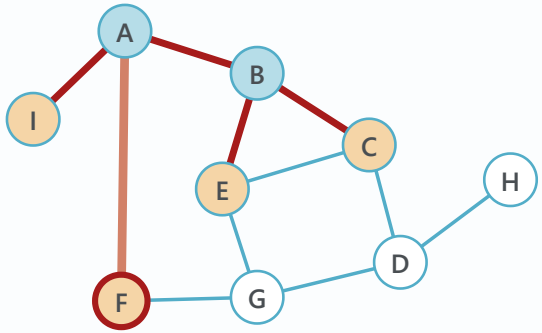
```
A B F
```



Kuyruğun önündeki **F** çıkarılır (poll) ve yazdırılır. Şimdi komşuları A, G taranır.

Kod İzleme: `bfs(A)`

```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```



q (ön → arka)

I	C	E					

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	T	T	F	F	T

○ görülmedi ● kuyrukta ◉ işleniyor ● işlendi

u = A: zaten ziyaret edilmiş → atlanır.

DEĞİŞKENLER

`w = F` `u = A`

ÇIKTI

```
A B F
```

Kod İzleme: `bfs(A)`

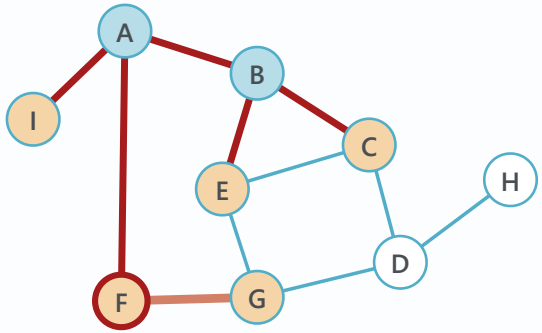
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
▶          ziyaret[u] = true;
▶          q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = F` `u = G`

ÇIKTI

```
A B F
```



q (ön → arka)

I	C	E	G				

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = G ziyaret edilmemiş → işaretlenir ve kuyruğun **arkasına** eklenir (düzey 2).

Kod İzleme: `bfs(A)`

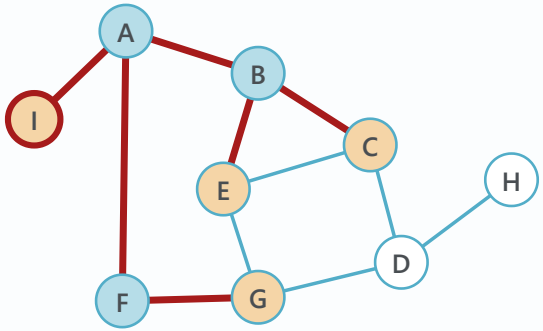
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
▶      int w = q.poll();
▶      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = I`

ÇIKTI

```
A B F I
```



q (ön → arka)

C	E	G					

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

Kuyruğun önündeki **I** çıkarılır (poll) ve yazdırılır. Şimdi komşusu A taranır.

Kod İzleme: `bfs(A)`

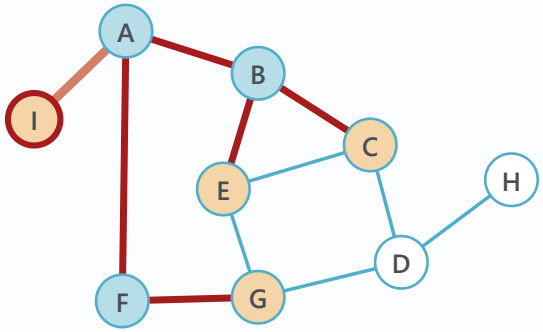
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = I` `u = A`

ÇIKTI

```
A B F I
```



q (ön → arka)

C	E	G					

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = A: zaten ziyaret edilmiş → atlanır.

Kod İzleme: `bfs(A)` (devam)

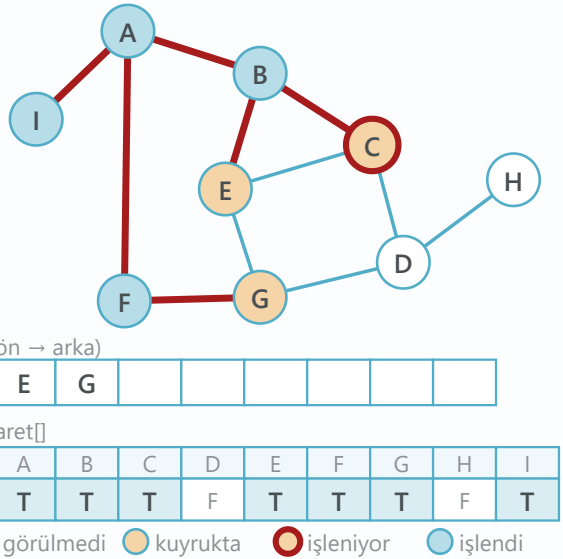
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
▶      int w = q.poll();
▶      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = C`

ÇIKTI

```
A B F I C
```



Kuyruğun önündeki **C** çıkarılır (poll) ve yazdırılır. Şimdi komşuları B, D, E taranır.

Kod İzleme: `bfs(A)` (devam)

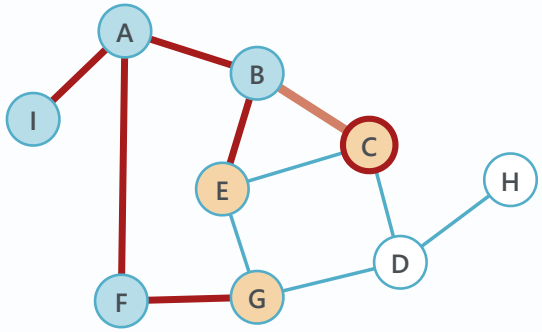
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = C` `u = B`

ÇIKTI

```
A B F I C
```



q (ön → arka)

E	G						

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = B: zaten ziyaret edilmiş → atlanır.

Kod İzleme: `bfs(A)` (devam)

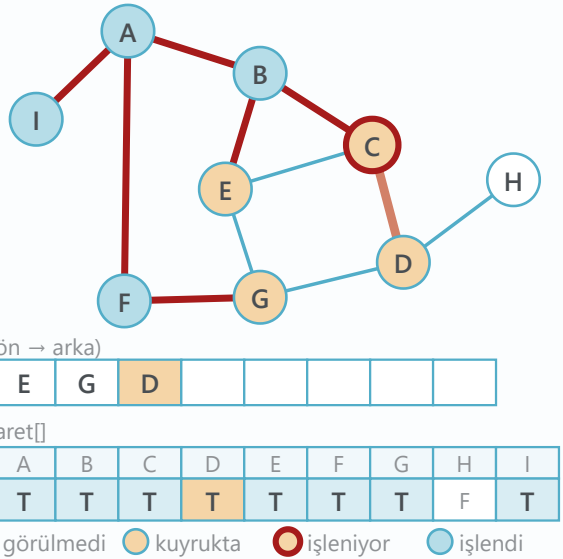
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
▶          ziyaret[u] = true;
▶          q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = C` `u = D`

ÇIKTI

```
A B F I C
```



u = D ziyaret edilmemiş → işaretlenir ve kuyruğun **arkasına** eklenir (düzey 3).

Kod İzleme: `bfs(A)` (devam)

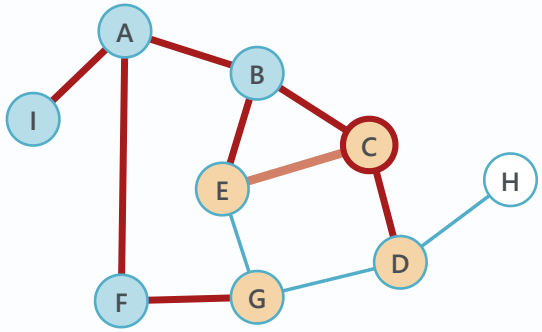
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = C` `u = E`

ÇIKTI

```
A B F I C
```



q (ön → arka)

E	G	D					

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = E: zaten ziyaret edilmiş → atlanır.

Kod İzleme: `bfs(A)` (devam)

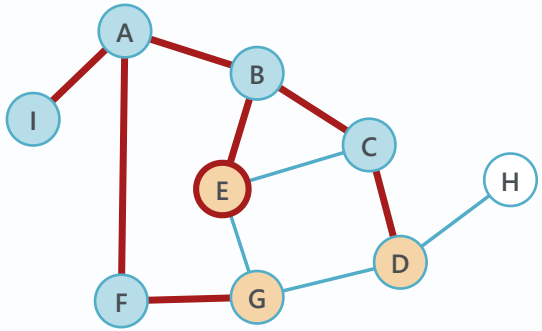
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
▶      int w = q.poll();
▶      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = E`

ÇIKTI

```
A B F I C E
```



q (ön → arka)

G	D						

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

Kuyruğun önündeki **E** çıkarılır (poll) ve yazdırılır. Şimdi komşuları B, C, G taranır.

Kod İzleme: `bfs(A)` (devam)

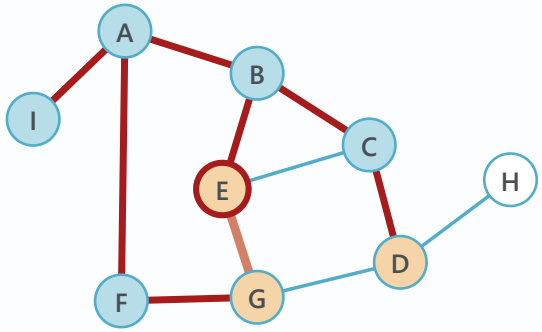
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = E` `u = G`

ÇIKTI

```
A B F I C E
```



q (ön → arka)

G	D						

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = B, C, G: hepsi zaten ziyaret edilmiş → atlanır.

Kod İzleme: `bfs(A)` (devam)

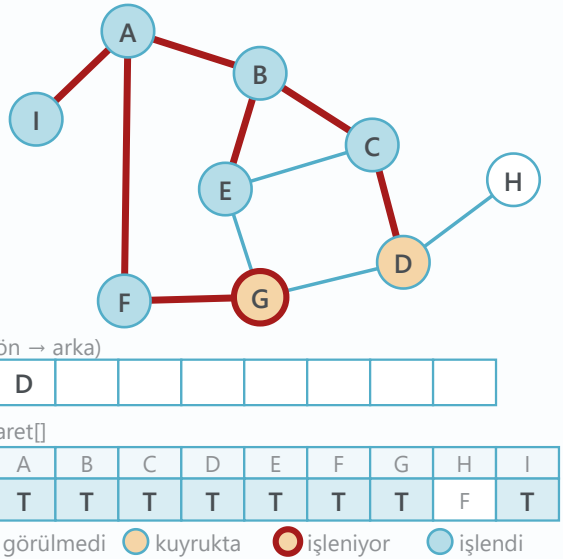
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
▶      int w = q.poll();
▶      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = G`

ÇIKTI

```
A B F I C E G
```



Kuyruğun önündeki **G** çıkarılır (poll) ve yazdırılır. Şimdi komşuları D, E, F taranır.

Kod İzleme: `bfs(A)` (devam)

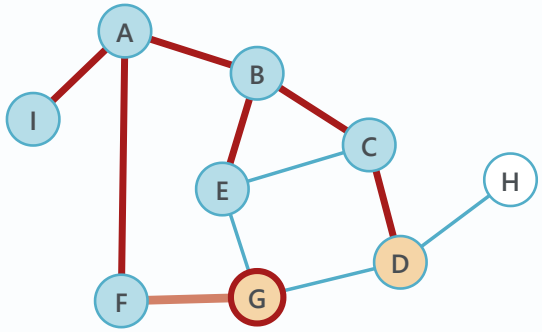
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = G` `u = F`

ÇIKTI

```
A B F I C E G
```



q (ön → arka)

D							

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = D, E, F: hepsi zaten ziyaret edilmiş → atlanır.

Kod İzleme: `bfs(A)` (devam)

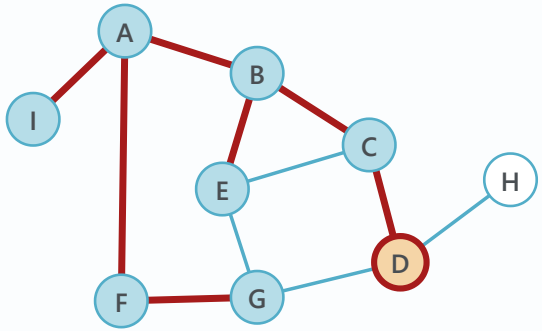
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
▶      int w = q.poll();
▶      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = D`

ÇIKTI

```
A B F I C E G D
```



q (ön → arka)

(boş)							

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

Kuyruğun önündeki **D** çıkarılır (poll) ve yazdırılır. Şimdi komşuları C, G, H taranır.

Kod İzleme: `bfs(A)` (devam)

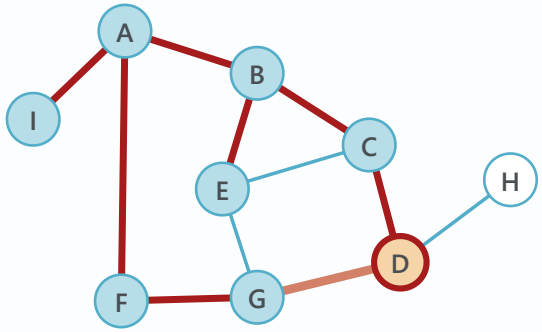
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = D` `u = G`

ÇIKTI

```
A B F I C E G D
```



q (ön → arka)

(boş)							

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	T

görülmedi · kuyrukta · işleniyor · işlendi

u = C, G: hepsi zaten ziyaret edilmiş → atlanır.

Kod İzleme: `bfs(A)` (devam)

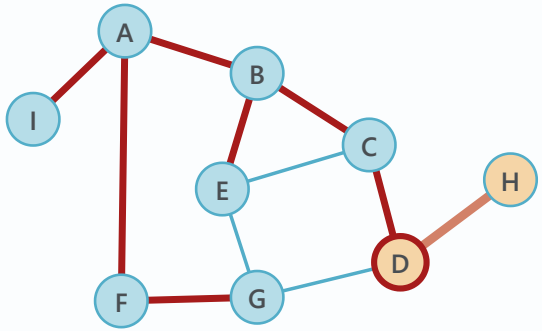
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
▶          ziyaret[u] = true;
▶          q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = D` `u = H`

ÇIKTI

```
A B F I C E G D
```



q (ön → arka)

H							

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · kuyrukta · işleniyor · işlendi

u = H ziyaret edilmemiş → işaretlenir ve kuyruğun **arkasına** eklenir (düzey 4).

Kod İzleme: `bfs(A)` (devam)

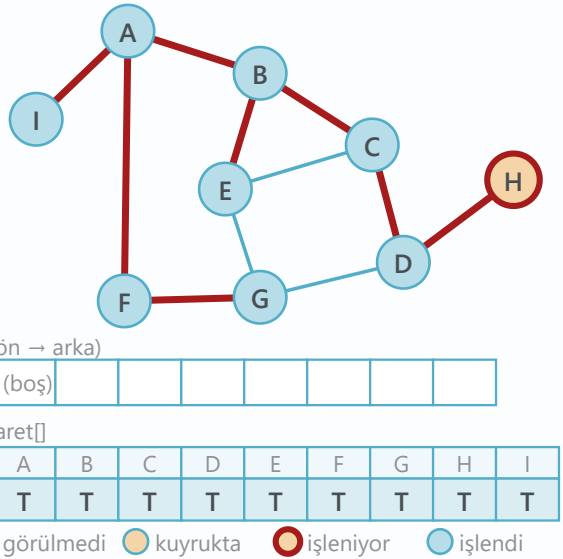
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
▶      int w = q.poll();
▶      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = H`

ÇIKTI

```
A B F I C E G D H
```



Kuyruğun önündeki **H** çıkarılır (poll) ve yazdırılır. Şimdi komşusu D taranır.

Kod İzleme: `bfs(A)` (devam)

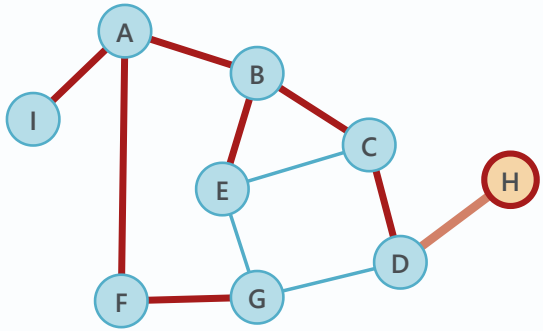
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
5    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
▶        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

`w = H` `u = D`

ÇIKTI

```
A B F I C E G D H
```



q (ön → arka)

(boş)							

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · kuyrukta · işleniyor · işlendi

u = D: zaten ziyaret edilmiş → atlanır.

Kod İzleme: `bfs(A)` (devam)

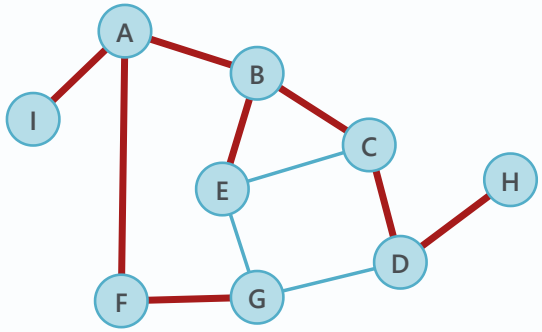
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> q = new ArrayDeque<>();
4    ziyaret[s] = true;  q.add(s);
▶    while (!q.isEmpty()) {
6      int w = q.poll();
7      System.out.print(w + " ");
8      for (int u : komsu.get(w))
9        if (!ziyaret[u]) {
10         ziyaret[u] = true;
11         q.add(u);
12       }
13   }
14 }
```

DEĞİŞKENLER

—

ÇIKTI

```
A B F I C E G D H
```



q (ön → arka)

(boş)							

ziyaret[]

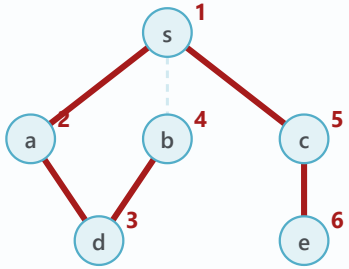
A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · kuyrukta · işleniyor · işlendi

Kuyruk boş → biter. Sıra **A B F I C E G D H**: düzey düzey. Her düğüm bir kez kuyruğa girer, her liste bir kez taranır → **O(n + m)**.

Derinlik Öncelikli Arama (DFS)

- Düğümleri **bir yol boyunca** keşfeder: gidebildiği kadar derine iner.
- Çıkmaza gelince (ziyaret edilmemiş komşu kalmayınca) **geri döner** (backtracking) ve başka dalı dener.
- **Yığın** (stack) yapısı veya özyinelemeli çağrılar kullanılır.



s→a→d→b derine; çıkmazda geri, sonra c→e

Artıları

+ Bellek kullanımı genellikle düşüktür (yalnızca mevcut yol yığında).
+ Bir yolun sonuna kadar gitme yeteneği sağlar.

Eksisi

– En kısa yolu garanti etmez.

Kullanım: labirent çözme, döngü tespiti, bağlantılı bileşenleri bulma ve yönlü döngüsüz çizgelerde topolojik sıralama.

DFS Algoritması

Özyinelemeli

```
void dfs(int v, boolean[] ziyaret) {
  ziyaret[v] = true;
  System.out.print(v + " ");
  for (int u : komsu.get(v))
    if (!ziyaret[u])
      dfs(u, ziyaret);
}
```

Özyineleme, yığını bizim yerimize **çağrı yığını** ile tutar.

Açık yığınla

```
void dfsYigin(int s) {
  boolean[] ziyaret = new boolean[n];
  Deque<Integer> yigin = new ArrayDeque<>();
  yigin.push(s);  ziyaret[s] = true;
  while (!yigin.isEmpty()) {
    int v = yigin.peek();
    Integer yeni = null;
    for (int u : komsu.get(v))
      if (!ziyaret[u]) { yeni = u; break; }
    if (yeni == null) yigin.pop(); // geri
    else { ziyaret[yeni] = true;
           yigin.push(yeni); }
  }
}
```

Kod İzleme: `dfs(A)` (özyinelemeli)

```
1 void dfs(int v, boolean[] ziyaret) {
▶   ziyaret[v] = true;
▶   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

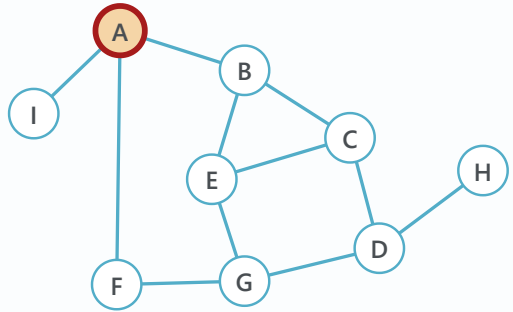
ÇAĞRI YIĞINI

`dfs(A)`

`main()`

ÇIKTI

```
A
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	F	F	F	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

`dfs(A, ziyaret)` çağrıldı (A=0, B=1, …, I=8; komşular alfabetik). A işaretlenir ve yazdırılır.

Adım 1 / 15

DEĞİŞKENLER

v = A

Kod İzleme: `dfs(A)` (özyinelemeli)

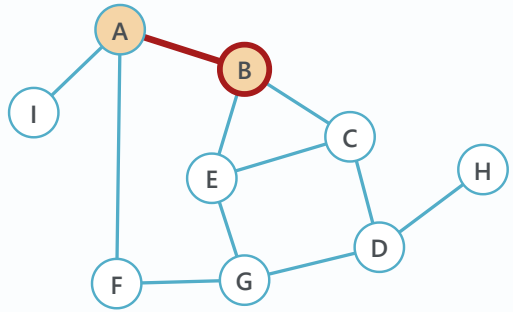
```
1 void dfs(int v, boolean[] ziyaret) {
▶   ziyaret[v] = true;
▶   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	F	F	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

A'nın komşusu B ziyaret edilmemiş → `dfs(B)` çağrılır: yeni çerçeve açılır, B işaretlenip yazdırılır.

Adım 2 / 15

DEĞİŞKENLER

`v = B`

Kod İzleme: `dfs(A)` (özyinelemeli)

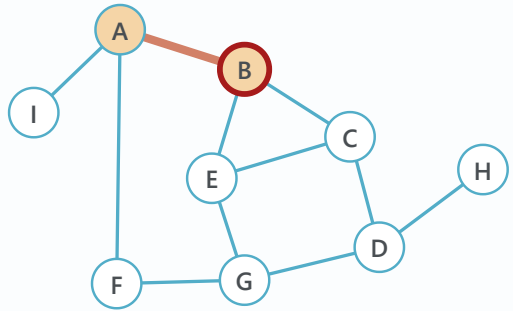
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	F	F	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

B'nin komşusu A zaten ziyaret edilmiş → atla.

Adım 3 / 15

DEĞİŞKENLER

`v = B` `u = A`

Kod İzleme: `dfs(A)` (özyinelemeli)

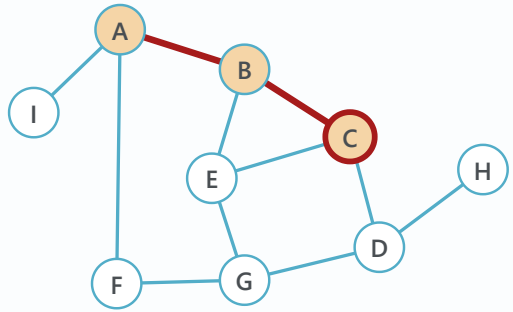
```
1 void dfs(int v, boolean[] ziyaret) {
▶   ziyaret[v] = true;
▶   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

B'nin komşusu C ziyaret edilmemiş → `dfs(C)` çağrılır: yeni çerçeve açılır, C işaretlenip yazdırılır.

Adım 4 / 15

DEĞİŞKENLER

`v = C`

Kod İzleme: `dfs(A)` (özyinelemeli)

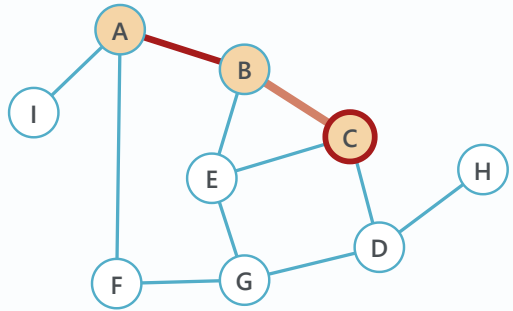
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- **dfs(C)**
- dfs(B)
- dfs(A)
- main()

ÇIKTI

```
A B C
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

C'nin komşusu B zaten ziyaret edilmiş → atla.

Adım 5 / 15

DEĞİŞKENLER

v = C u = B

Kod İzleme: `dfs(A)` (özyinelemeli)

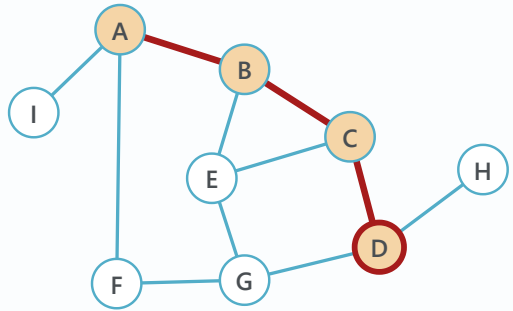
```
1 void dfs(int v, boolean[] ziyaret) {
▶   ziyaret[v] = true;
▶   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(D)`
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

C'nin komşusu D ziyaret edilmemiş → `dfs(D)` çağrılır: yeni çerçeve açılır, D işaretlenip yazdırılır.

DEĞİŞKENLER

`v = D`

Kod İzleme: `dfs(A)` (özyinelemeli)

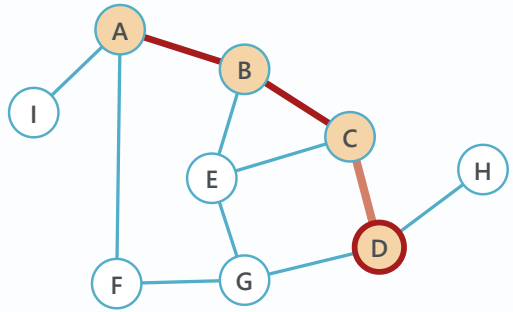
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(D)`
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

D'nin komşusu C zaten ziyaret edilmiş → atla.

DEĞİŞKENLER

`v = D` `u = C`

Kod İzleme: `dfs(A)` (özyinelemeli)

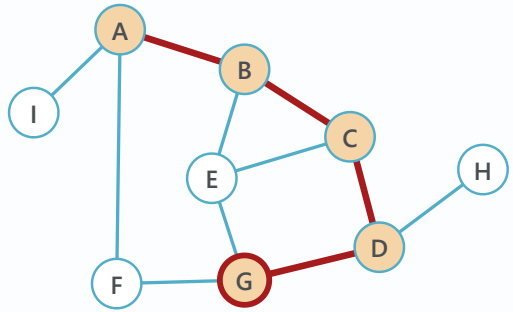
```
1 void dfs(int v, boolean[] ziyaret) {
▶   ziyaret[v] = true;
▶   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(G)`
- `dfs(D)`
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	F	F	T	F	F

görülmedi · yığında · üstte · çıktı

D'nin komşusu G ziyaret edilmemiş → `dfs(G)` çağrılır: yeni çerçeve açılır, G işaretlenip yazdırılır.

Adım 8 / 15

DEĞİŞKENLER

`v = G`

Kod İzleme: `dfs(A)` (özyinelemeli)

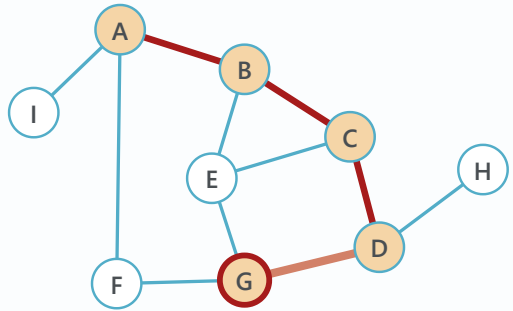
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- **dfs(G)**
- dfs(D)
- dfs(C)
- dfs(B)
- dfs(A)
- main()

ÇIKTI

```
A B C D G
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	F	F	T	F	F

görülmedi · yığında · üstte · çıktı

G'nin komşusu D zaten ziyaret edilmiş → atla.

DEĞİŞKENLER

`v = G` `u = D`

Kod İzleme: `dfs(A)` (özyinelemeli)

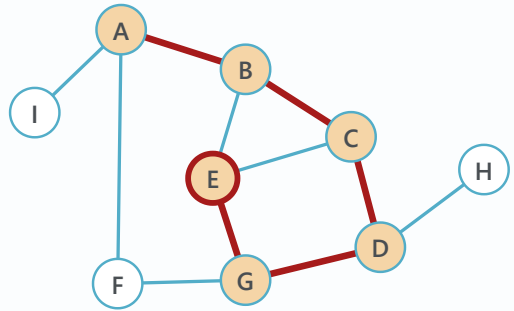
```
1 void dfs(int v, boolean[] ziyaret) {
▶   ziyaret[v] = true;
▶   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(E)`
- `dfs(G)`
- `dfs(D)`
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	F	T	F	F

görülmedi · yığında · üstte · çıktı

G'nin komşusu E ziyaret edilmemiş → `dfs(E)` çağrılır: yeni çerçeve açılır, E işaretlenip yazdırılır.

Adım 10 / 15

DEĞİŞKENLER

`v = E`

Kod İzleme: `dfs(A)` (özyinelemeli)

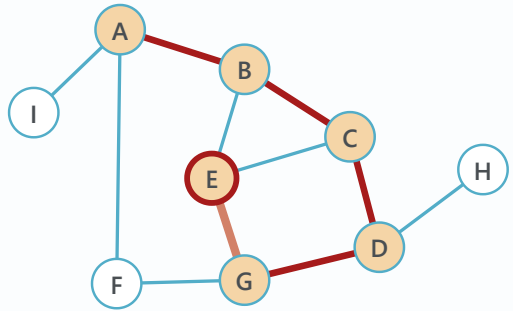
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- **dfs(E)**
- dfs(G)
- dfs(D)
- dfs(C)
- dfs(B)
- dfs(A)
- main()

ÇIKTI

```
A B C D G E
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	F	T	F	F

görülmedi · yığında · üstte · çıktı

E'nin komşuları B, C, G zaten ziyaret edilmiş → atla.

DEĞİŞKENLER

`v = E` `u = G`

Kod İzleme: `dfs(A)` (özyinelemeli)

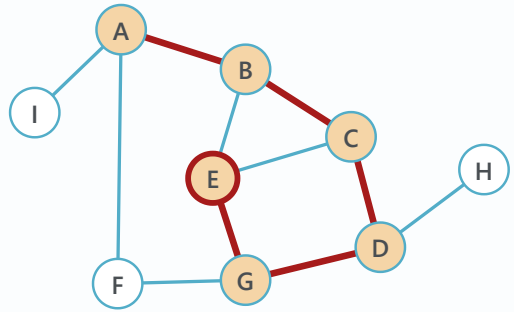
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
  }
```

ÇAĞRI YIĞINI

- **dfs(E)**
- dfs(G)
- dfs(D)
- dfs(C)
- dfs(B)
- dfs(A)
- main()

ÇIKTI

```
A B C D G E
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	F	T	F	F

görülmedi · yığında · üstte · çıktı

E'nin gidilecek komşusu kalmadı → `dfs(E)` biter, çerçeve yığından çıkar (geri dönüş).

Adım 12 / 15

DEĞİŞKENLER

v = **E**

Kod İzleme: `dfs(A)` (özyinelemeli)

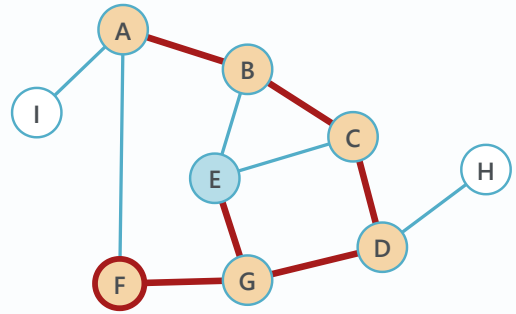
```
1 void dfs(int v, boolean[] ziyaret) {
▶   ziyaret[v] = true;
▶   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(F)`
- `dfs(G)`
- `dfs(D)`
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	F

görülmedi · yığında · üstte · çıktı

G'nin komşusu F ziyaret edilmemiş → `dfs(F)` çağrılır: yeni çerçeve açılır, F işaretlenip yazdırılır.

Adım 13 / 15

DEĞİŞKENLER

v = F

Kod İzleme: `dfs(A)` (özyinelemeli)

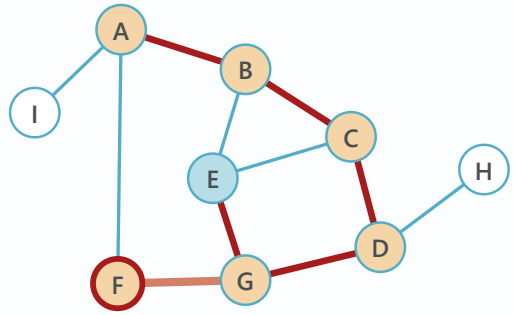
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- **dfs(F)**
- dfs(G)
- dfs(D)
- dfs(C)
- dfs(B)
- dfs(A)
- main()

ÇIKTI

```
A B C D G E F
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	F

görülmedi · yığında · üstte · çıktı

F'nin komşuları A, G zaten ziyaret edilmiş → atla.

DEĞİŞKENLER

v = F u = G

Kod İzleme: `dfs(A)` (özyinelemeli)

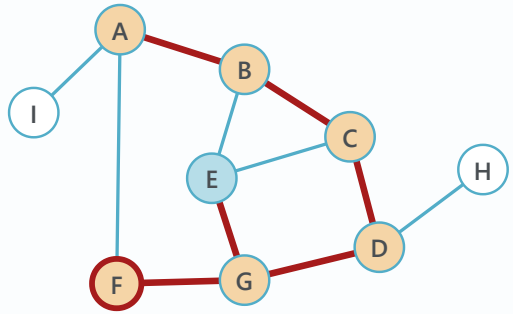
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
  }
```

ÇAĞRI YIĞINI

- `dfs(F)`
- `dfs(G)`
- `dfs(D)`
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	F

görülmedi · yığında · üstte · çıktı

F'nin gidilecek komşusu kalmadı → `dfs(F)` biter, çerçeve yığından çıkar (geri dönüş).

Adım 15 / 15

DEĞİŞKENLER

v = F

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

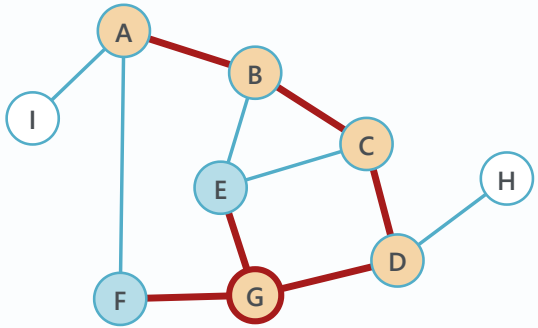
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
▶ }
```

ÇAĞRI YIĞINI

- **`dfs(G)`**
- `dfs(D)`
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	F

görülmedi · yığında · üstte · çıktı

G'nin gidilecek komşusu kalmadı → `dfs(G)` biter, çerçeve yığından çıkar (geri dönüş).

Adım 1 / 14

DEĞİŞKENLER

`v = G`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

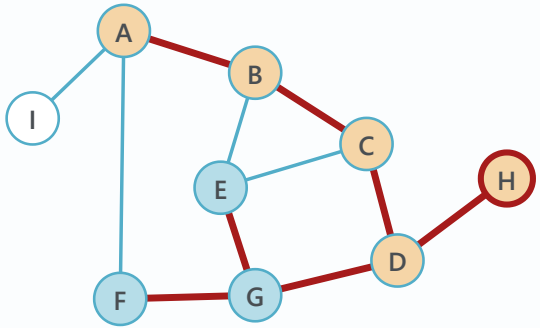
```
1 void dfs(int v, boolean[] ziyaret) {
▶   ziyaret[v] = true;
▶   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(H)`
- `dfs(D)`
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F H
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

D'nin komşusu H ziyaret edilmemiş → `dfs(H)` çağrılır: yeni çerçeve açılır, H işaretlenip yazdırılır.

DEĞİŞKENLER

`v = H`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

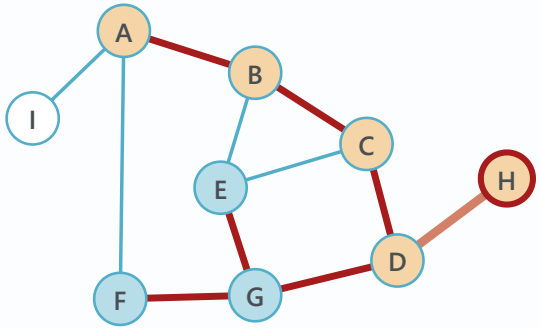
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- **dfs(H)**
- dfs(D)
- dfs(C)
- dfs(B)
- dfs(A)
- main()

ÇIKTI

```
A B C D G E F H
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

H'nin komşusu D zaten ziyaret edilmiş → atla.

DEĞİŞKENLER

`v = H` `u = D`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

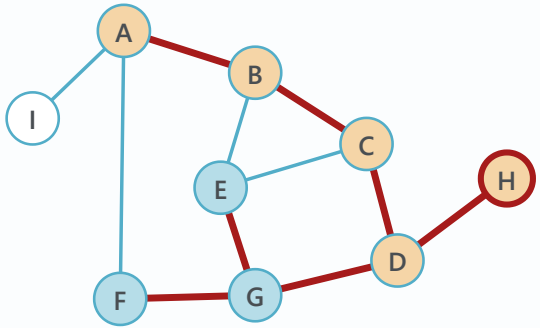
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
▶ }
```

ÇAĞRI YIĞINI

- **`dfs(H)`**
- `dfs(D)`
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F H
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

H'nin gidilecek komşusu kalmadı → `dfs(H)` biter, çerçeve yığından çıkar (geri dönüş).

DEĞİŞKENLER

`v = H`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

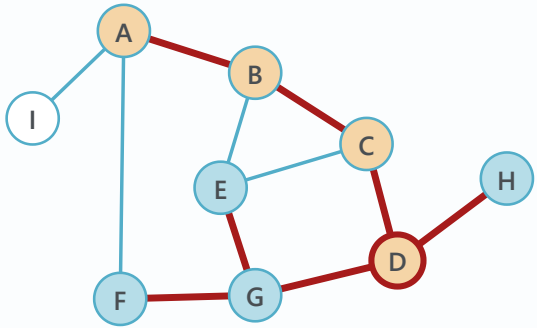
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
▶ }
```

ÇAĞRI YIĞINI

- **`dfs(D)`**
- `dfs(C)`
- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F H
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

D'nin gidilecek komşusu kalmadı → `dfs(D)` biter, çerçeve yığından çıkar (geri dönüş).

DEĞİŞKENLER

`v = D`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

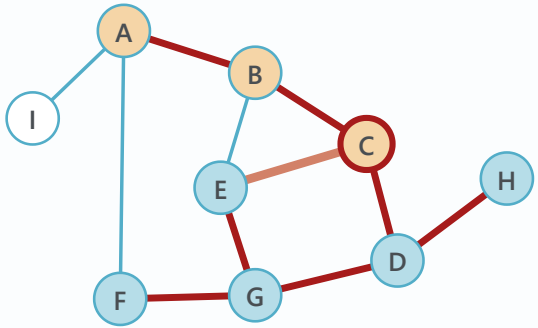
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- **dfs(C)**
- dfs(B)
- dfs(A)
- main()

ÇIKTI

```
A B C D G E F H
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

C'nin komşusu E zaten ziyaret edilmiş → atla.

DEĞİŞKENLER

v = C u = E

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

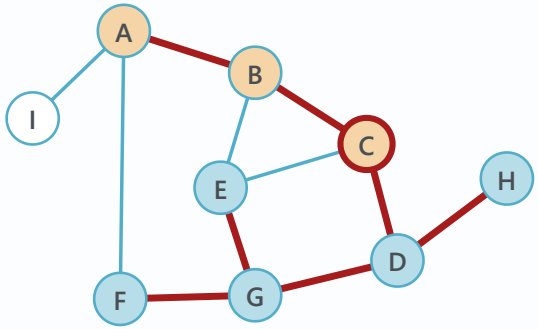
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
  }
```

ÇAĞRI YIĞINI

- **dfs(C)**
- dfs(B)
- dfs(A)
- main()

ÇIKTI

```
A B C D G E F H
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

C'nin gidilecek komşusu kalmadı → `dfs(C)` biter, çerçeve yığından çıkar (geri dönüş).

DEĞİŞKENLER

v = **C**

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

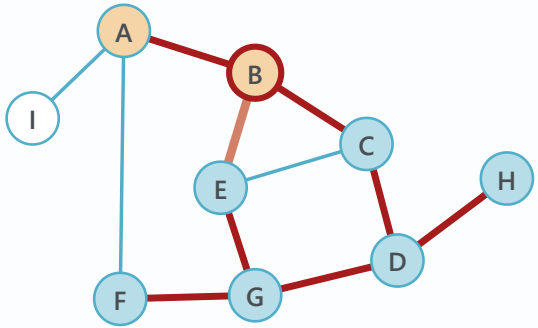
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F H
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

B'nin komşusu E zaten ziyaret edilmiş → atla.

DEĞİŞKENLER

`v = B` `u = E`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

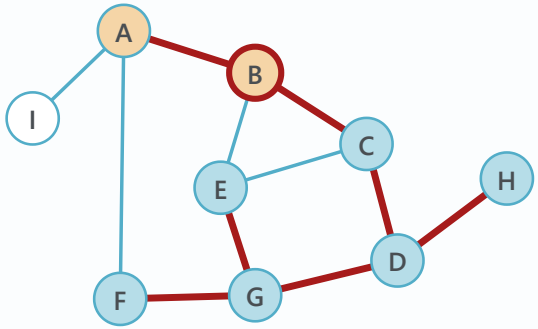
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
▶ }
```

ÇAĞRI YIĞINI

- `dfs(B)`
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F H
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

B'nin gidilecek komşusu kalmadı → `dfs(B)` biter, çerçeve yığından çıkar (geri dönüş).

DEĞİŞKENLER

`v = B`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

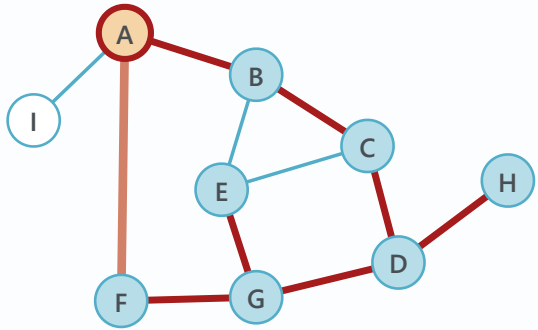
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F H
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

A'nın komşusu F zaten ziyaret edilmiş → atla.

Adım 10 / 14

DEĞİŞKENLER

`v = A` `u = F`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

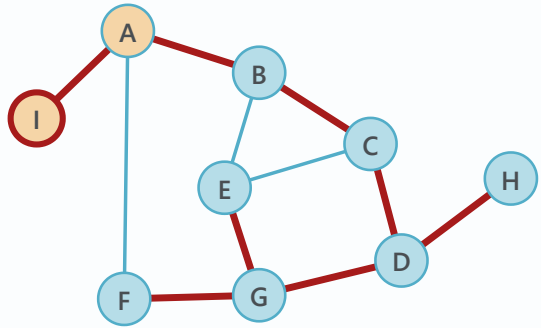
```
1 void dfs(int v, boolean[] ziyaret) {
▶   ziyaret[v] = true;
▶   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- **dfs(I)**
- dfs(A)
- main()

ÇIKTI

```
A B C D G E F H I
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · yığında · üstte · çıktı

A'nın komşusu I ziyaret edilmemiş → `dfs(I)` çağrılır: yeni çerçeve açılır, I işaretlenip yazdırılır.

Adım 11 / 14

DEĞİŞKENLER

`v = I`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

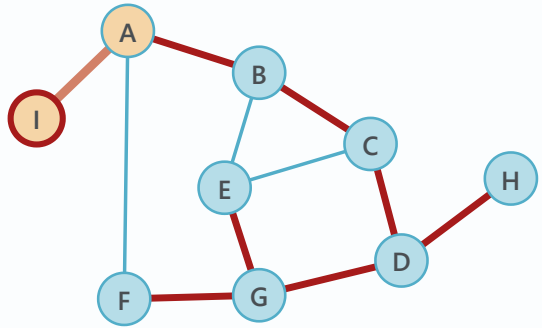
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
▶   for (int u : komsu.get(v))
▶     if (!ziyaret[u])
6       dfs(u, ziyaret);
7 }
```

ÇAĞRI YIĞINI

- **dfs(I)**
- dfs(A)
- main()

ÇIKTI

```
A B C D G E F H I
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · yığında · üstte · çıktı

I'nın komşusu A zaten ziyaret edilmiş → atla.

Adım 12 / 14

DEĞİŞKENLER

`v = I` `u = A`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

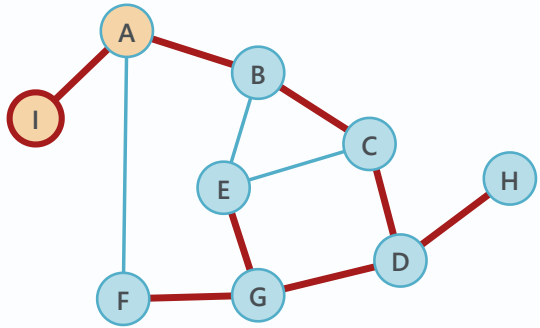
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
▶ }
```

ÇAĞRI YIĞINI

- **`dfs(I)`**
- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F H I
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · yığında · üstte · çıktı

I'nın gidilecek komşusu kalmadı → `dfs(I)` biter, çerçeve yığından çıkar (geri dönüş).

Adım 13 / 14

DEĞİŞKENLER

`v = I`

Kod İzleme: `dfs(A)` (özyinelemeli) (devam)

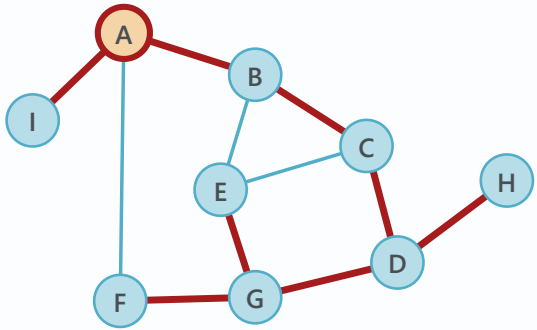
```
1 void dfs(int v, boolean[] ziyaret) {
2   ziyaret[v] = true;
3   System.out.print(v + " ");
4   for (int u : komsu.get(v))
5     if (!ziyaret[u])
6       dfs(u, ziyaret);
▶ }
```

ÇAĞRI YIĞINI

- `dfs(A)`
- `main()`

ÇIKTI

```
A B C D G E F H I
```



ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · yığında · üstte · çıktı

`dfs(A)` biter, yığın boşalır. Sıra: **A B C D G E F H I**.

Adım 14 / 14

DEĞİŞKENLER

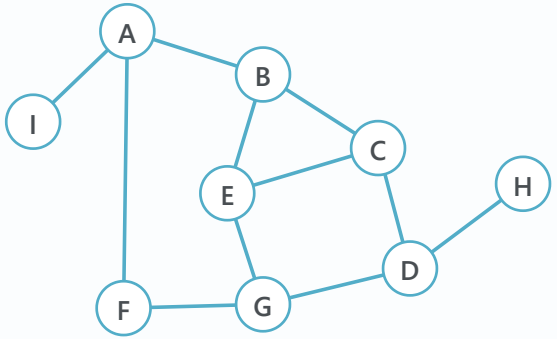
`v = A`

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
▶    boolean[] ziyaret = new boolean[n];
▶    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
6      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`s = A`



yigin (alt → üst)

(boş)							

ziyaret[]

A	B	C	D	E	F	G	H	I
F	F	F	F	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

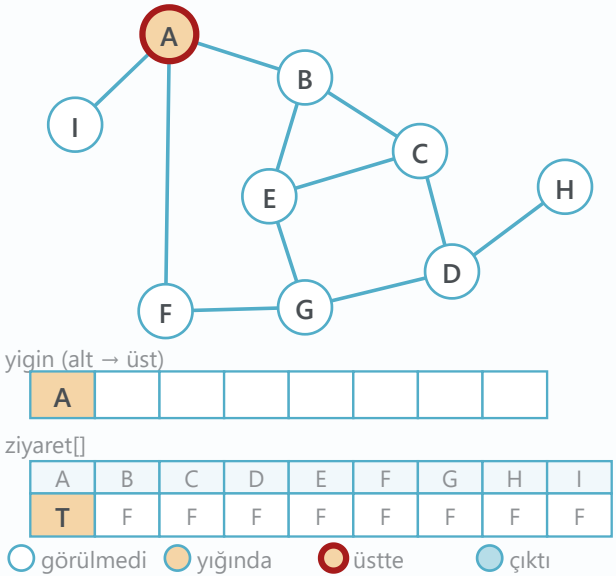
Başlangıç s = A. `ziyaret` tümü false, yığın boş.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
▶    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
6      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`s = A`



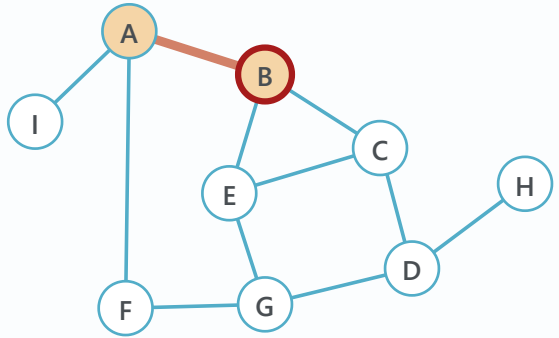
A yığına itilir ve işaretlenir. Yığının üstü = şu an bulunduğumuz düğüm.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
▶        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
▶      else { ziyaret[yeni] = true;
▶             yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = A` `yeni = B`



yigin (alt → üst)

A	B						

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	F	F	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

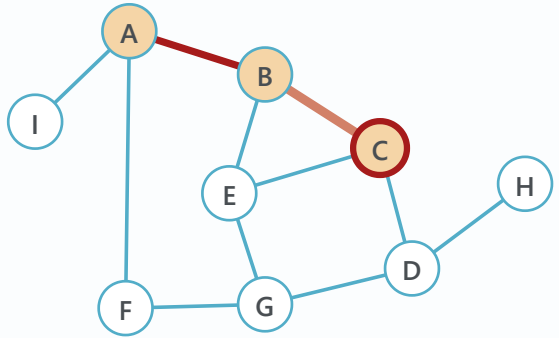
peek = **A**; ilk ziyaret edilmemiş komşusu **B** (break) → işaretlenir ve yığına itilir.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
▶        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
▶      else { ziyaret[yeni] = true;
▶             yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = B` `yeni = C`



yigin (alt → üst)

A	B	C					

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	F	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

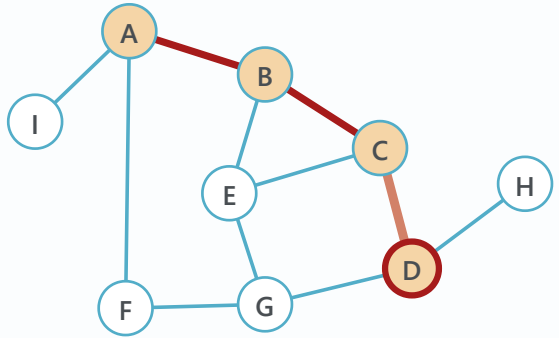
peek = **B**; ilk ziyaret edilmemiş komşusu **C** (break) → işaretlenir ve yığına itilir.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
▶        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
▶      else { ziyaret[yeni] = true;
▶             yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = C` `yeni = D`



yigin (alt → üst)

A	B	C	D				

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	F	F	F	F	F

görülmedi · yığında · üstte · çıktı

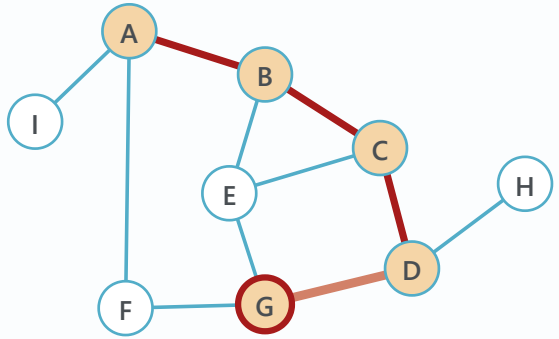
peek = **C**; ilk ziyaret edilmemiş komşusu **D** (break) → işaretlenir ve yığına itilir.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
▶        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
▶      else { ziyaret[yeni] = true;
▶             yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = D` `yeni = G`



yigin (alt → üst)

A	B	C	D	G			

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	F	F	T	F	F

görülmedi · yığında · üstte · çıktı

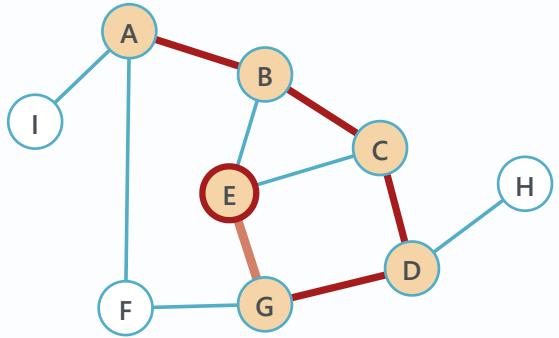
peek = **D**; ilk ziyaret edilmemiş komşusu **G** (break) → işaretlenir ve yığına itilir.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
▶        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
▶      else { ziyaret[yeni] = true;
▶             yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = G` `yeni = E`



yigin (alt → üst)

A	B	C	D	G	E		

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	F	T	F	F

görülmedi · yığında · üstte · çıktı

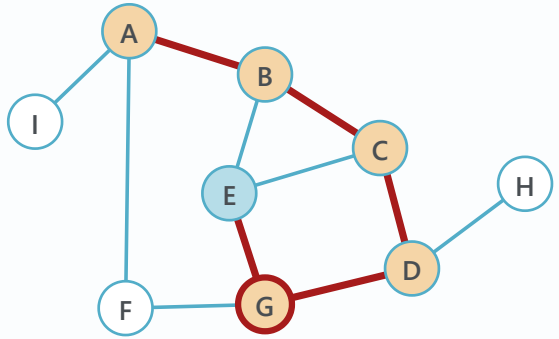
peek = **G**; ilk ziyaret edilmemiş komşusu **E** (break) → işaretlenir ve yığına itilir.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
▶      if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = E` `yeni = null`



yigin (alt → üst)

A	B	C	D	G			

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	F	T	F	F

görülmedi · yığında · üstte · çıktı

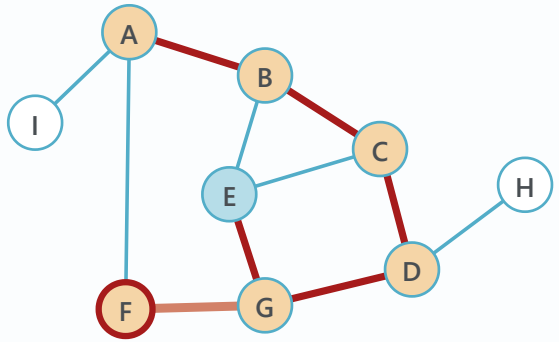
peek = **E**; ziyaret edilmemiş komşusu yok (yeni = null) → **pop**, geri dönülür.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
▶        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
▶      else { ziyaret[yeni] = true;
▶             yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = G` `yeni = F`



yigin (alt → üst)

A	B	C	D	G	F		

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	F

görülmedi · yığında · üstte · çıktı

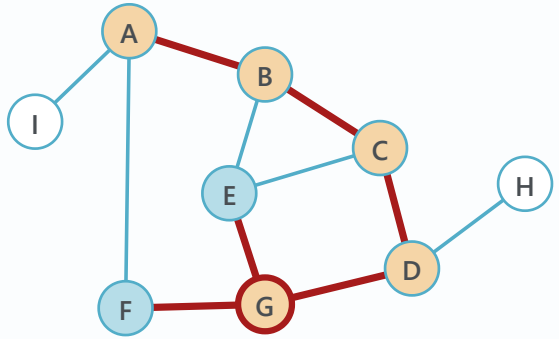
peek = **G**; ilk ziyaret edilmemiş komşusu **F** (break) → işaretlenir ve yığına itilir.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
▶      if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = F` `yeni = null`



yigin (alt → üst)

A	B	C	D	G			

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	F

görülmedi · yığında · üstte · çıktı

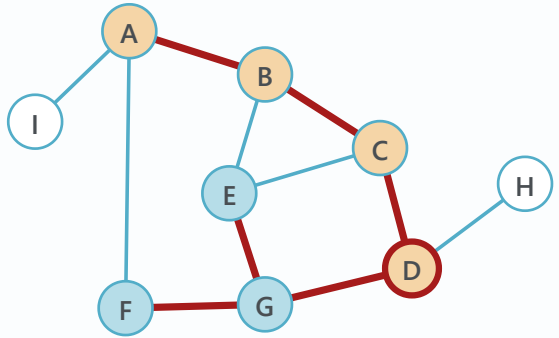
peek = **F**; ziyaret edilmemiş komşusu yok (yeni = null) → **pop**, geri dönülür.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
▶      if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = G` `yeni = null`



yigin (alt → üst)

A	B	C	D				

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	F	F

görülmedi · yığında · üstte · çıktı

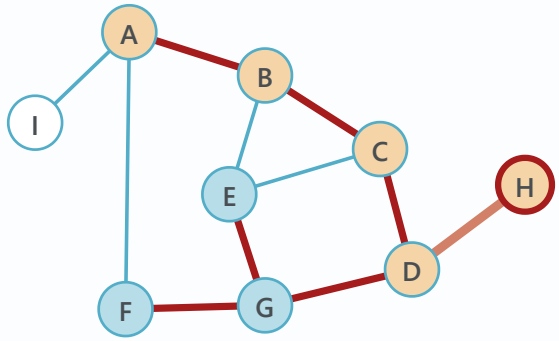
peek = **G**; ziyaret edilmemiş komşusu yok (yeni = null) → **pop**, geri dönülür.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
▶        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
▶      else { ziyaret[yeni] = true;
▶             yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = D` `yeni = H`



yigin (alt → üst)

A	B	C	D	H			

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

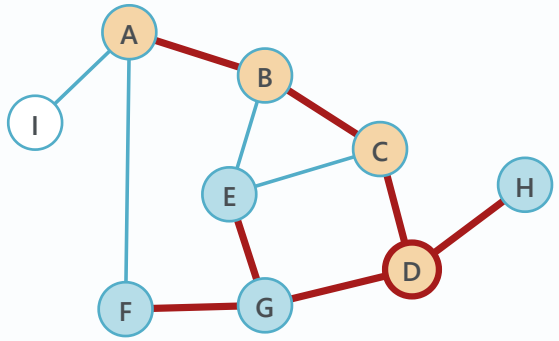
peek = **D**; ilk ziyaret edilmemiş komşusu **H** (break) → işaretlenir ve yığına itilir.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
▶      if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = H` `yeni = null`



yigin (alt → üst)

A	B	C	D				

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

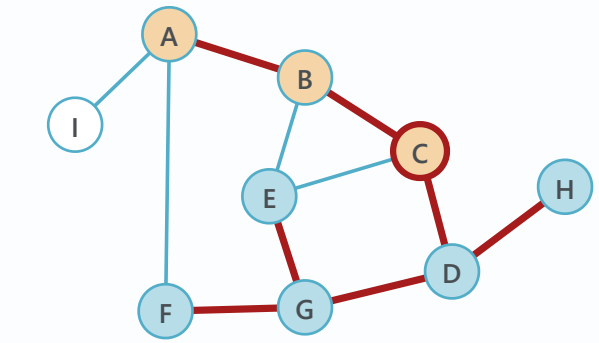
peek = **H**; ziyaret edilmemiş komşusu yok (yeni = null) → **pop**, geri dönülür.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
▶      if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = D` `yeni = null`



yigin (alt → üst)

A	B	C					

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

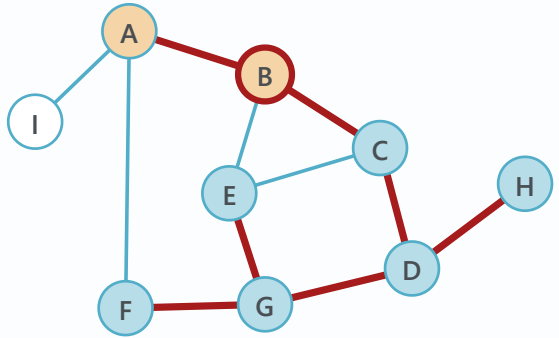
peek = **D**; ziyaret edilmemiş komşusu yok (yeni = null) → **pop**, geri dönülür.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
▶      if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = C` `yeni = null`



yigin (alt → üst)

A	B						

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

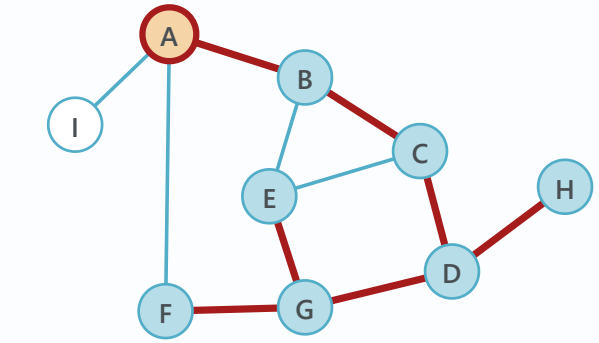
peek = **C**; ziyaret edilmemiş komşusu yok (yeni = null) → **pop**, geri dönülür.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
▶      if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = B` `yeni = null`



yigin (alt → üst)

A							

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	F

görülmedi · yığında · üstte · çıktı

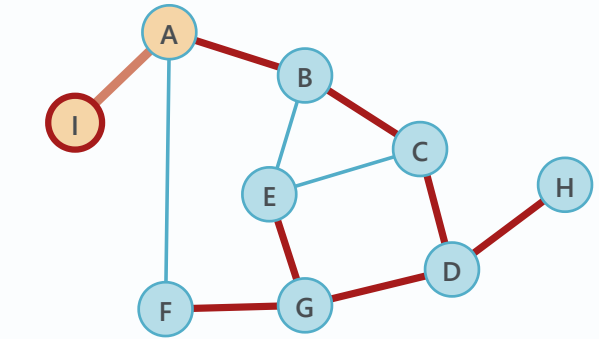
peek = **B**; ziyaret edilmemiş komşusu yok (yeni = null) → **pop**, geri dönülür.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
▶        if (!ziyaret[u]) { yeni = u; break; }
10     if (yeni == null) yigin.pop(); // geri
▶      else { ziyaret[yeni] = true;
▶             yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = A` `yeni = I`



yigin (alt → üst)

A	I						

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · yığında · üstte · çıktı

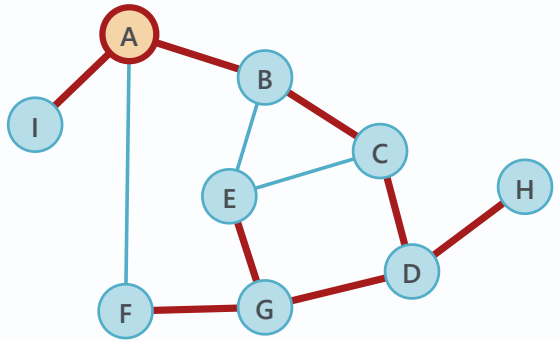
peek = **A**; ilk ziyaret edilmemiş komşusu **I** (break) → işaretlenir ve yığına itilir.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
▶      if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = I` `yeni = null`



yigin (alt → üst)

A							

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · yığında · üstte · çıktı

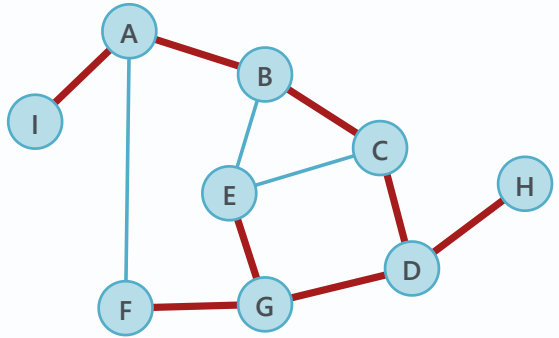
peek = **I**; ziyaret edilmemiş komşusu yok (yeni = null) → **pop**, geri dönülür.

Kod İzleme: `dfsYigin(A)`

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>();
4    yigin.push(s);  ziyaret[s] = true;
5    while (!yigin.isEmpty()) {
▶      int v = yigin.peek();
7      Integer yeni = null;
8      for (int u : komsu.get(v))
9        if (!ziyaret[u]) { yeni = u; break; }
▶      if (yeni == null) yigin.pop(); // geri
11     else { ziyaret[yeni] = true;
12            yigin.push(yeni); }
13   }
14 }
```

DEĞİŞKENLER

`v = A` `yeni = null`



yigin (alt → üst)

(boş)							

ziyaret[]

A	B	C	D	E	F	G	H	I
T	T	T	T	T	T	T	T	T

görülmedi · yığında · üstte · çıktı

peek = **A**; ziyaret edilmemiş komşusu yok (yeni = null) → **pop**, geri dönülür. Yığın boş: bitti. Ziyaret sırası özyinelemeli sürümle aynı.

BFS ve DFS Karşılaştırması

	BFS	DFS
Yardımcı yapı	Kuyruk (FIFO)	Yığın (LIFO) / özyineleme
Zaman (komşuluk listesi)	O(n + m)	O(n + m)
Ek bellek	O(n): en geniş düzey kuyrukta	O(n): en uzun yol yığında
En kısa yol?	Evet (ağırlıksız çizgede)	Hayır
Tipik kullanım	en az adım, düzey bulma	döngü tespiti, bileşenler, labirent

Neden O(n + m)? Her düğüm bir kez işlenir ve tüm komşuluk kayıtları taranır. Yönsüz çizgede her kenar iki listede görünür; 2m yine O(m)'dir.

İzlemelerdeki sıralar, komşuların alfabetik sırayla gezildiği varsayımıyla elde edildi; komşu sırası değişirse ziyaret sırası da değişir. Açık yığınlı `dfsYigin` komşu listesini her `peek`'te baştan taradığı için O(n + m)'den uzun sürebilir; her düğümün kaldığı indis tutulursa O(n + m) olur.

Kod İzleme: DFS ile bileşenleri numaralama



Tüm düğümler **−1** (henüz hiçbir bileşene atanmadı).

Adım 1 / 18

```
1  int bilesenler(int[] bil) {
▶    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`bil = [-1, -1, -1, -1, -1, -1]`

ÇAĞRI YIĞINI

- **bilesenler(bil)**
- main()

Kod İzleme: DFS ile bileşenleri numaralama



Bileşen sayacı **say = 0**.

Adım 2 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
▶    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`say = 0`

`bil = [-1, -1, -1, -1, -1, -1]`

ÇAĞRI YIĞINI

`bilesenler(bil)`

`main()`

Kod İzleme: DFS ile bileşenleri numaralama



bil[0] = −1: **yeni bileşen**. `say++` 0 verir, say = 1 olur; dfs(0, 0) çağrılır.

Adım 3 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
▶    for (int s = 0; s < n; s++)
▶      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 0` `say = 1`

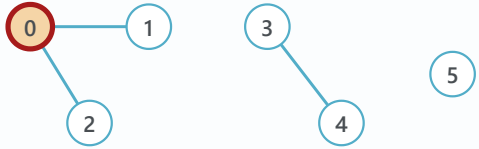
`bil = [-1, -1, -1, -1, -1, -1]`

ÇAĞRI YIĞINI

`bilesenler(bil)`

`main()`

Kod İzleme: DFS ile bileşenleri numaralama



v	0	1	2	3	4	5
bil	0	-1	-1	-1	-1	-1

bil[0] = 0: 0, 0 numaralı bileşene boyandı.

Adım 4 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
▶    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 0` `say = 1` `v = 0` `c = 0`

`bil = [0, -1, -1, -1, -1, -1]`

ÇAĞRI YIĞINI

- `dfs(0, 0)`
- `bilesenler(bil)`
- `main()`

Kod İzleme: DFS ile bileşenleri numaralama



bil[1] = 0: 1, 0 numaralı bileşene boyandı.

Adım 5 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
▶    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 0` `say = 1` `v = 1` `c = 0`

`bil = [0, 0, -1, -1, -1, -1]`

ÇAĞRI YIĞINI

- **dfs(1, 0)**
- dfs(0, 0)
- bilesenler(bil)
- main()

Kod İzleme: DFS ile bileşenleri numaralama



Komşu 0 zaten boyalı (bil = 0) → atla. Her kenara iki kez bakılır: O(n + m).

Adım 6 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
▶    for (int u : komsu.get(v))
▶      if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 0` `say = 1` `v = 1` `c = 0` `u = 0`

`bil = [0, 0, -1, -1, -1, -1]`

ÇAĞRI YIĞINI

- **dfs(1, 0)**
- dfs(0, 0)
- bilesenler(bil)
- main()

Kod İzleme: DFS ile bileşenleri numaralama



bil[2] = 0: 2, 0 numaralı bileşene boyandı.


```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
▶    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 0` `say = 1` `v = 2` `c = 0`

`bil = [0, 0, 0, -1, -1, -1]`

ÇAĞRI YIĞINI

- **dfs(2, 0)**
- dfs(0, 0)
- bilesenler(bil)
- main()

Kod İzleme: DFS ile bileşenleri numaralama



Komşu 0 zaten boyalı (bil = 0) → atla. Her kenara iki kez bakılır: O(n + m).


```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
▶    for (int u : komsu.get(v))
▶      if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

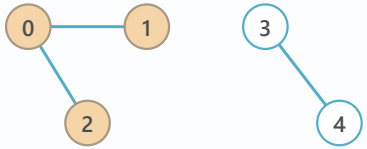
`s = 0` `say = 1` `v = 2` `c = 0`

`u = 0` `bil = [0, 0, 0, -1, -1, -1]`

ÇAĞRI YIĞINI

- `dfs(2, 0)`
- `dfs(0, 0)`
- `bilesenler(bil)`
- `main()`

Kod İzleme: DFS ile bileşenleri numaralama



v	0	1	2	3	4	5
bil	0	0	0	-1	-1	-1

bil[1] = 0: 1 önceki bir DFS'te zaten boyandı → atla.

Adım 9 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
▶    for (int s = 0; s < n; s++)
▶      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 1` `say = 1`

`bil = [0, 0, 0, -1, -1, -1]`

ÇAĞRI YIĞINI

`bilesenler(bil)`

`main()`

Kod İzleme: DFS ile bileşenleri numaralama



bil[2] = 0: 2 önceki bir DFS'te zaten boyandı → atla.


```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
▶    for (int s = 0; s < n; s++)
▶      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 2` `say = 1`

`bil = [0, 0, 0, -1, -1, -1]`

ÇAĞRI YIĞINI

`bilesenler(bil)`

`main()`

Kod İzleme: DFS ile bileşenleri numaralama



bil[3] = −1: **yeni bileşen**. `say++` 1 verir, say = 2 olur; dfs(3, 1) çağrılır.

Adım 11 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
▶    for (int s = 0; s < n; s++)
▶      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 3` `say = 2`

`bil = [0, 0, 0, -1, -1, -1]`

ÇAĞRI YIĞINI

bilesenler(bil)

main()

Kod İzleme: DFS ile bileşenleri numaralama



bil[3] = 1: 3, 1 numaralı bileşene boyandı.

Adım 12 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
▶    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

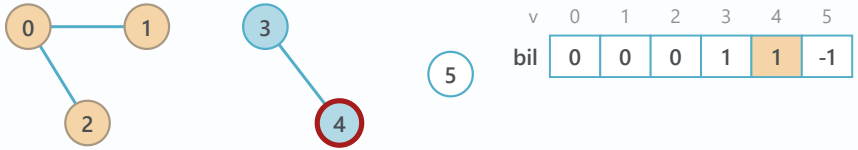
`s = 3` `say = 2` `v = 3` `c = 1`

`bil = [0, 0, 0, 1, -1, -1]`

ÇAĞRI YIĞINI

- **dfs(3, 1)**
- bilesenler(bil)
- main()

Kod İzleme: DFS ile bileşenleri numaralama



bil[4] = 1: 4, 1 numaralı bileşene boyandı.


```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
▶    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

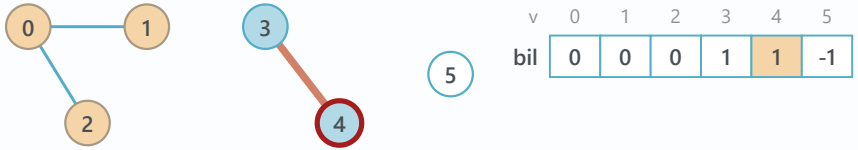
`s = 3` `say = 2` `v = 4` `c = 1`

`bil = [0, 0, 0, 1, 1, -1]`

ÇAĞRI YIĞINI

- **dfs(4, 1)**
- dfs(3, 1)
- bilesenler(bil)
- main()

Kod İzleme: DFS ile bileşenleri numaralama



Komşu 3 zaten boyalı (bil = 1) → atla. Her kenara iki kez bakılır: O(n + m).

Adım 14 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
▶    for (int u : komsu.get(v))
▶      if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

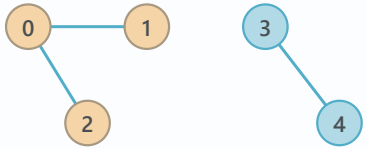
`s = 3` `say = 2` `v = 4` `c = 1`

`u = 3` `bil = [0, 0, 0, 1, 1, -1]`

ÇAĞRI YIĞINI

- `dfs(4, 1)`
- `dfs(3, 1)`
- `bilesenler(bil)`
- `main()`

Kod İzleme: DFS ile bileşenleri numaralama



5

v	0	1	2	3	4	5
bil	0	0	0	1	1	-1

bil[4] = 1: 4 önceki bir DFS'te zaten boyandı → atla.


```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
▶    for (int s = 0; s < n; s++)
▶      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 4` `say = 2`

`bil = [0, 0, 0, 1, 1, -1]`

ÇAĞRI YIĞINI

`bilesenler(bil)`

`main()`

Kod İzleme: DFS ile bileşenleri numaralama



bil[5] = −1: **yeni bileşen**. `say++` 2 verir, say = 3 olur; dfs(5, 2) çağrılır.

Adım 16 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
▶    for (int s = 0; s < n; s++)
▶      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 5` `say = 3`

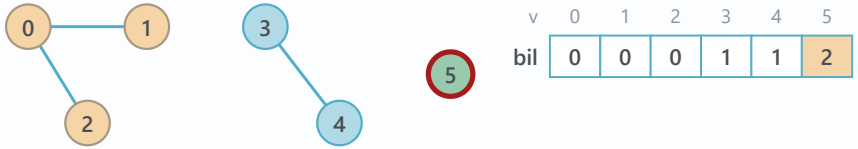
`bil = [0, 0, 0, 1, 1, -1]`

ÇAĞRI YIĞINI

`bilesenler(bil)`

`main()`

Kod İzleme: DFS ile bileşenleri numaralama



bil[5] = 2: 5, 2 numaralı bileşene boyandı.


```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
6    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
▶    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`s = 5` `say = 3` `v = 5` `c = 2`

`bil = [0, 0, 0, 1, 1, 2]`

ÇAĞRI YIĞINI

- **dfs(5, 2)**
- bilesenler(bil)
- main()

Kod İzleme: DFS ile bileşenleri numaralama



3 bileşen: {0, 1, 2}, {3, 4}, {5}. Aynı bil = aynı bileşen; `bil[u] == bil[v]` sorgusu O(1).

Adım 18 / 18

```
1  int bilesenler(int[] bil) {
2    Arrays.fill(bil, -1);
3    int say = 0;
4    for (int s = 0; s < n; s++)
5      if (bil[s] == -1) dfs(s, say++, bil);
▶    return say;
7  }
8  void dfs(int v, int c, int[] bil) {
9    bil[v] = c;
10   for (int u : komsu.get(v))
11     if (bil[u] == -1) dfs(u, c, bil);
12 }
```

DEĞİŞKENLER

`say = 3` `dönüş = 3`

`bil = [0, 0, 0, 1, 1, 2]`

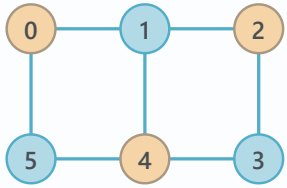
ÇAĞRI YIĞINI

`bilesenler(bil)`

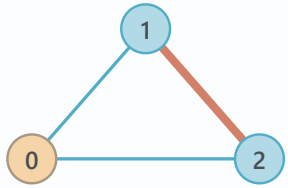
`main()`

İki Parçalı (Bipartite) Çizge Testi

- **İki parçalı** (bipartite): düğümler iki gruba ayrılır, her kenar gruplar *arasında*.
- Test = **BFS ile 2-renkleme**: komşuya zıt renk ver; aynı renkli iki komşu → değil.
- Eşdeğer: **tek uzunlukta döngü yok**.



iki parçalı ✓ (çift döngüler)



üçgen X: 1 ve 2 aynı renk

```
// renk[] = -1 ile başlar; s'den BFS
renk[s] = 0;  q.add(s);
while (!q.isEmpty()) {
  int w = q.poll();
  for (int u : komsu.get(w))
    if (renk[u] == -1) {
      renk[u] = 1 - renk[w];  q.add(u);
    } else if (renk[u] == renk[w]) return false;
}
return true;
```

BFS ile aynı maliyet: O(n + m). Kullanım: öğrenci–proje eşleştirme, iki vardiyalı çizelgeleme, çakışmasız gruplama.

Topolojik Sıralama: Yalnızca DAG

- Bir yönlü çizgenin düğümlerini, her `u → v` kenarında u, v'den önce gelecek biçimde sıralar.
- Yalnızca **yönlü döngüsüz çizgelerde** (DAG) mümkündür.
- Sonuç tek olmak zorunda değildir; aynı anda seçilebilen birden çok giriş derecesi 0 düğüm olabilir.
- Kullanım: ders ön koşulları, görev bağımlılıkları, derleme ve iş akışı planlama.

Çizgede döngü varsa hiçbir topolojik sıra yoktur: döngüdeki her görev başka bir görevin önce yapılmasını bekler.

```
Kahn(G):
  q = girişDerecesi 0 olan düğümler
  sonuc = []
  while q boş değil:
    u = q'dan çıkar
    sonuc'a u ekle
    for u'dan çıkan her (u, v):
      girişDerecesi[v]--
      if girişDerecesi[v] == 0:
        q'ya v ekle
  if sonuc.size < n:
    döngü var
  return sonuc
```

Zaman

O(n + m)

Alternatif

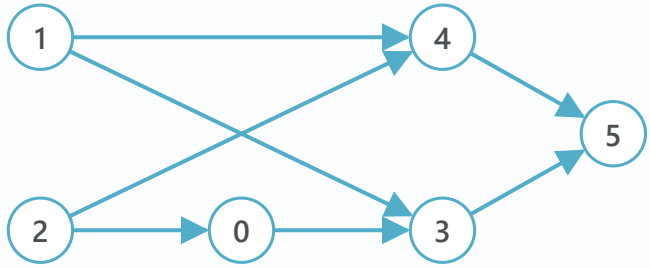
DFS bitiş zamanlarını ters çevirmek; geri kenar bulunursa döngü vardır.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
▶    int n = adj.size();
▶    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`n = 6`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	0	0	0

q	(boş)					
sonuc	[ ]					

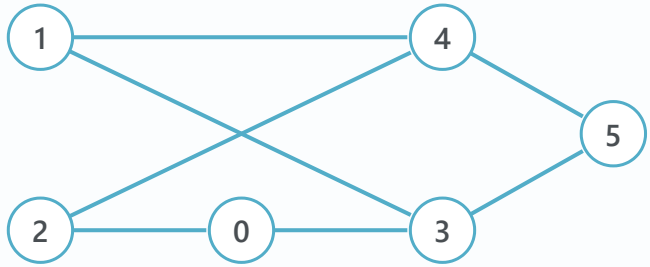
Ders ön koşulu örneği: 6 düğüm, 7 yönlü kenar (u → v: u, v'den önce). `giris[]` sıfırlarla başlar.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
▶    for (List<Integer> l : adj)
▶      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`n = 6` `l = adj[0]`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	1	0	0

q	(boş)					
sonuc	[ ]					

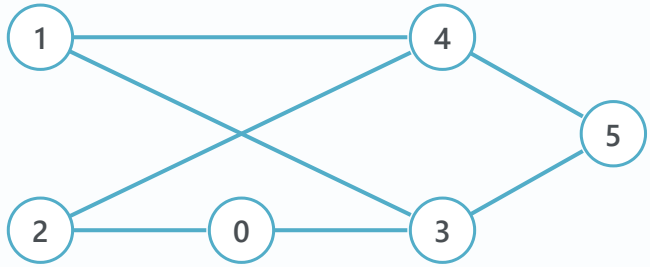
0'dan çıkan kenarlar 0→3: hedeflerin giriş derecesi birer artar.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
▶    for (List<Integer> l : adj)
▶      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`n = 6` `l = adj[1]`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	2	1	0

q	(boş)					
sonuc	[ ]					

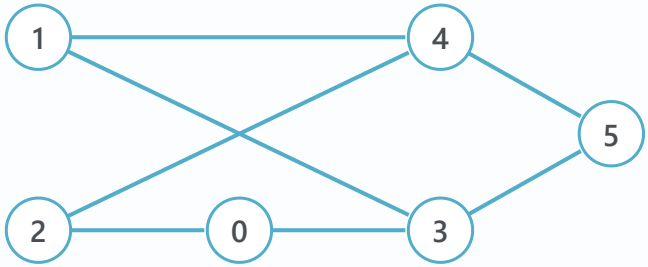
1'den çıkan kenarlar 1→3, 1→4: hedeflerin giriş derecesi birer artar.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
▶    for (List<Integer> l : adj)
▶      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`n = 6` `l = adj[2]`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	1	0	0	2	2	0

q	(boş)					
sonuc	[ ]					

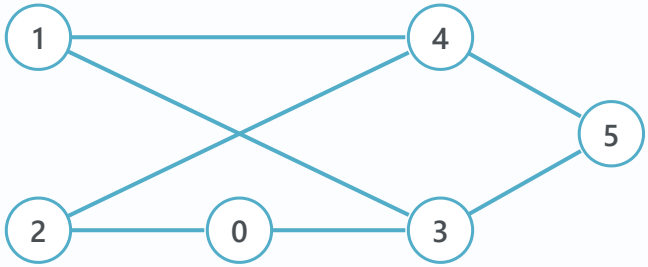
2'den çıkan kenarlar 2→0, 2→4: hedeflerin giriş derecesi birer artar.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
▶    for (List<Integer> l : adj)
▶      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`n = 6` `l = adj[3]`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	1	0	0	2	2	1

q	(boş)					
sonuc	[ ]					

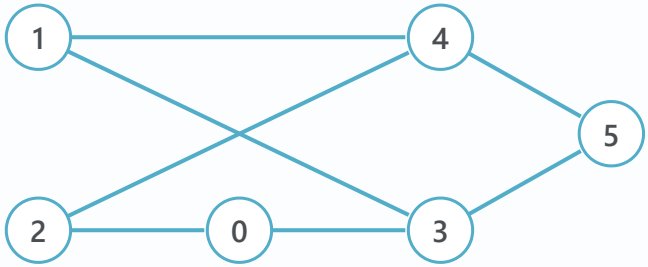
3'ten çıkan kenarlar 3→5: hedeflerin giriş derecesi birer artar.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
▶    for (List<Integer> l : adj)
▶      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`n = 6` `l = adj[4]`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	1	0	0	2	2	2

q	(boş)					
sonuc	[ ]					

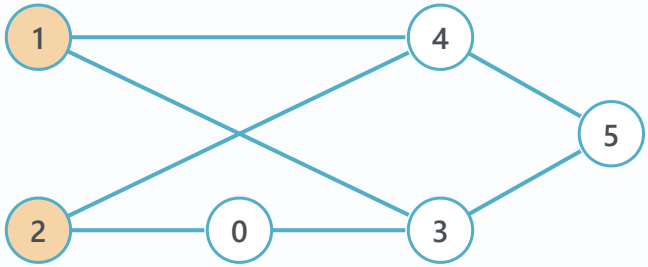
4'ten çıkan kenarlar 4→5: hedeflerin giriş derecesi birer artar.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
▶    Queue<Integer> q = new ArrayDeque<>();
▶    for (int v = 0; v < n; v++)
▶      if (giris[v] == 0) q.add(v);
▶    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`n = 6`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	1	0	0	2	2	2

q	1	2				
sonuc	[ ]					

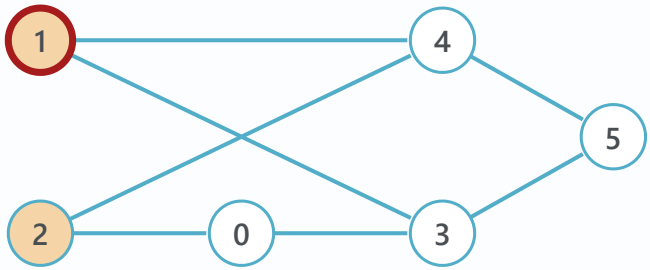
Giriş derecesi 0 olanlar (1, 2) hiçbir şeye bağımlı değil → kuyruğa. (5'in listesi boş olduğundan sayma turu bir şey değiştirmedi.)

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
▶      int u = q.poll();
▶      sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 1`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	1	0	0	2	2	2

q						
	2					

sonuc						
	1					

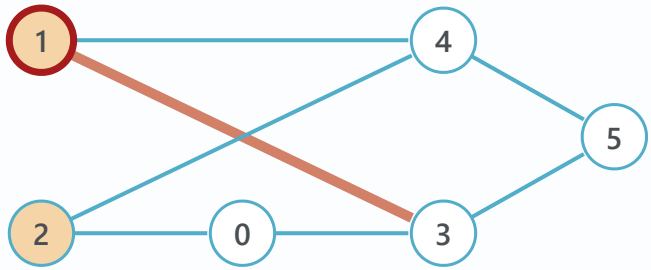
1 kuyruktan çıkar ve sonuca eklenir: ön koşullarının hepsi zaten sonuçta.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
▶        if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 1` `v = 3`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	1	0	0	1	2	2

q						
	2					

sonuc						
	1					

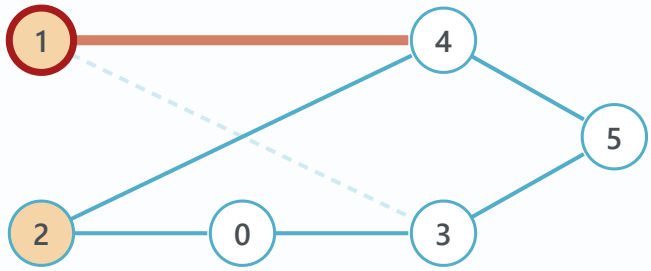
1→3 silinmiş sayılır: giris[3] = 1; 3 hâlâ başka bir ön koşul bekliyor.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
▶        if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 1` `v = 4`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	1	0	0	1	1	2

q						
	2					

sonuc						
	1					

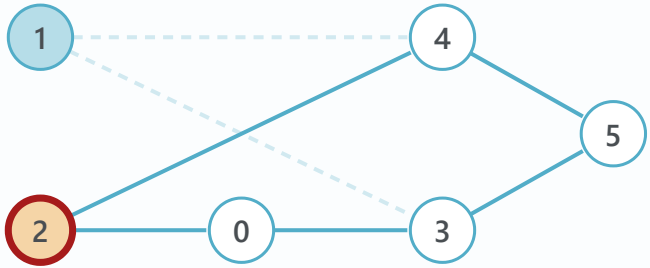
1→4 silinmiş sayılır: giris[4] = 1; 4 hâlâ başka bir ön koşul bekliyor.

Kod İzleme: Topolojik Sıralama (Kahn)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
▶      int u = q.poll();
▶      sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 2`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	1	0	0	1	1	2

q	(boş)					
sonuc	1	2				

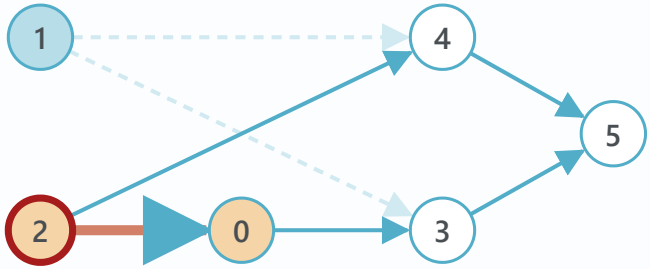
2 kuyruktan çıkar ve sonuca eklenir: ön koşullarının hepsi zaten sonuçta.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
▶        if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 2` `v = 0`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	1	1	2

q	0					
sonuc	1	2				

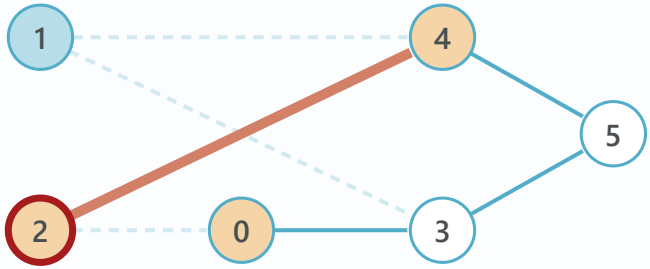
2→0 silinmiş sayılır: giris[0] = **0** → 0 artık hazır, kuyruğa eklenir.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
▶        if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

u = **2** v = **4**



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	1	0	2

q	0	4				
sonuc	1	2				

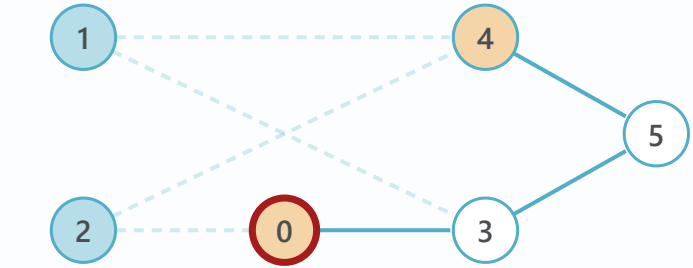
2→4 silinmiş sayılır: giris[4] = **0** → 4 artık hazır, kuyruğa eklenir.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
▶      int u = q.poll();
▶      sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 0`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	1	0	2

q						
	4					

sonuc						
	1	2	0			

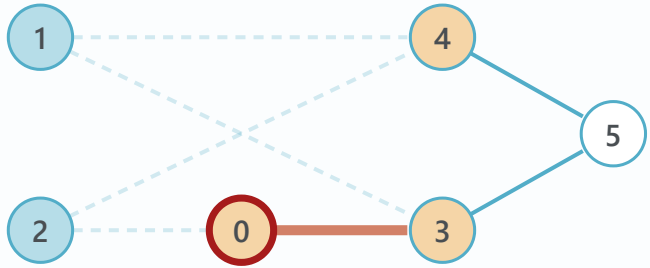
0 kuyruktan çıkar ve sonuca eklenir: ön koşullarının hepsi zaten sonuçta.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
▶        if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 0` `v = 3`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	0	0	2

q	4	3				
sonuc	1	2	0			

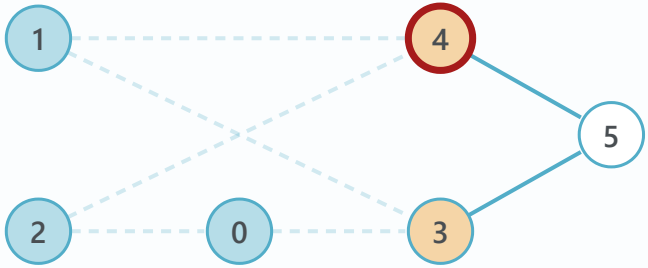
0→3 silinmiş sayılır: giris[3] = **0** → 3 artık hazır, kuyruğa eklenir.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
▶      int u = q.poll();
▶      sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 4`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	0	0	2

q						
	3					

sonuc						
	1	2	0	4		

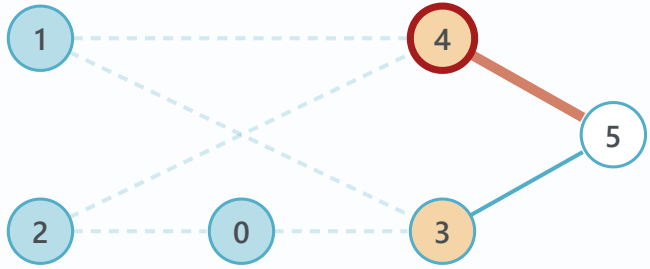
4 kuyruktan çıkar ve sonuca eklenir: ön koşullarının hepsi zaten sonuçta.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
▶        if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 4` `v = 5`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	0	0	1

q						
	3					

sonuc						
	1	2	0	4		

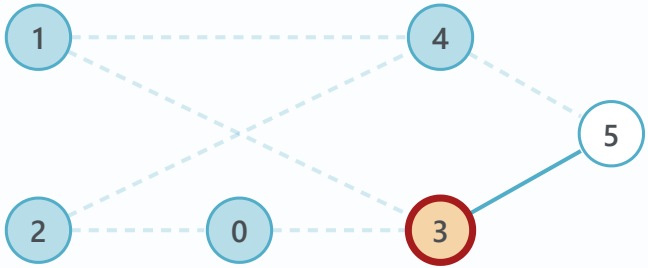
4→5 silinmiş sayılır: giris[5] = 1; 5 hâlâ başka bir ön koşul bekliyor.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
▶      int u = q.poll();
▶      sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 3`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	0	0	1

q	(boş)					

sonuc	1	2	0	4	3	

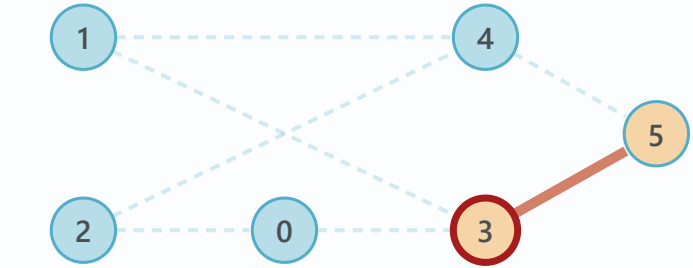
3 kuyruktan çıkar ve sonuca eklenir: ön koşullarının hepsi zaten sonuçta.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
▶        if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 3` `v = 5`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	0	0	0

q						
	5					

sonuc						
	1	2	0	4	3	

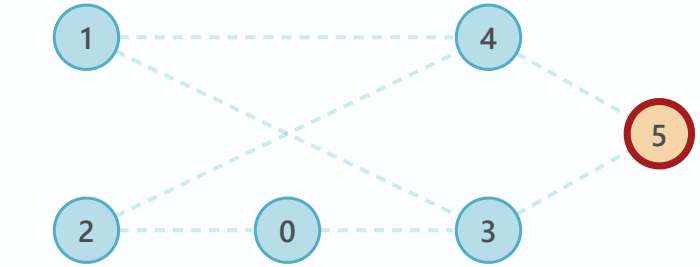
3→5 silinmiş sayılır: giris[5] = **0** → 5 artık hazır, kuyruğa eklenir.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
▶      int u = q.poll();
▶      sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
16   return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

`u = 5`



giris[] (giriş derecesi)

	0	1	2	3	4	5
giris	0	0	0	0	0	0

q: (boş)

sonuc: 1 2 0 4 3 **5**

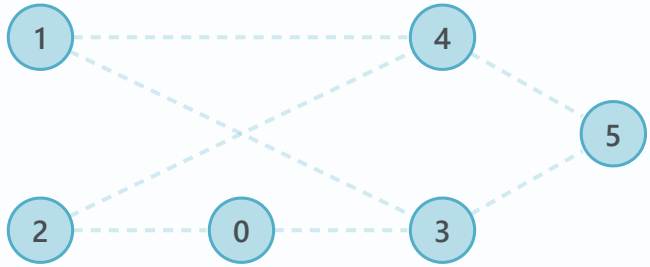
5 kuyruktan çıkar ve sonuca eklenir: ön koşullarının hepsi zaten sonuçta.

Kod İzleme: Topolojik Sıralama (Kahn) (devam)

```
1  List<Integer> kahn(List<List<Integer>> adj)
   {
2    int n = adj.size();
3    int[] giris = new int[n];
4    for (List<Integer> l : adj)
5      for (int v : l) giris[v]++;
6    Queue<Integer> q = new ArrayDeque<>();
7    for (int v = 0; v < n; v++)
8      if (giris[v] == 0) q.add(v);
9    List<Integer> sonuc = new ArrayList<>();
10   while (!q.isEmpty()) {
11     int u = q.poll();
12     sonuc.add(u);
13     for (int v : adj.get(u))
14       if (--giris[v] == 0) q.add(v);
15   }
▶    return sonuc;  // boyut < n → döngü
17 }
```

DEĞİŞKENLER

—



giris[] (giriş derecesi)

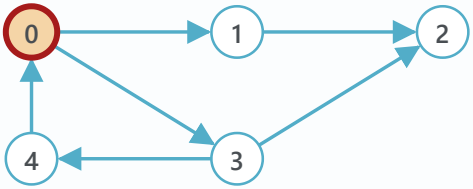
	0	1	2	3	4	5
giris	0	0	0	0	0	0

q	(boş)					

sonuc	1	2	0	4	3	5

Kuyruk boş; sonuç **1 2 0 4 3 5** ve boyutu 6 = n → döngü yok. Her kenar bir kez azaltıldı: **O(n + m)**.

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	0	0	0	0

0 beyaz · 1 gri · 2 siyah

renk[0] = 1 (gri): 0 şu anki DFS yolunda.

Adım 1 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
▶    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

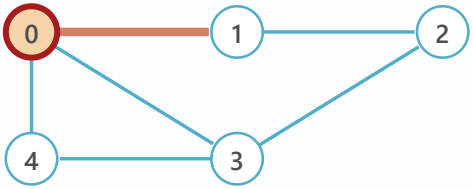
DEĞİŞKENLER

`v = 0` `renk = [1, 0, 0, 0, 0]`

ÇAĞRI YIĞINI

- `donguVar(0)`
- `main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	0	0	0	0

0 beyaz · 1 gri · 2 siyah

Kenar 0 → 1: 1 beyaz → özyinelemeli çağrı.

Adım 2 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
▶      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

DEĞİŞKENLER

`v = 0` `u = 1`

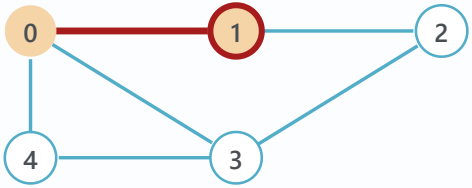
`renk = [1, 0, 0, 0, 0]`

ÇAĞRI YIĞINI

`donguVar(0)`

`main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	1	0	0	0

0 beyaz · 1 gri · 2 siyah

renk[1] = 1 (gri): 1 şu anki DFS yolunda.

Adım 3 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
▶    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

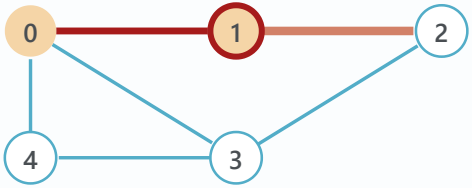
DEĞİŞKENLER

`v = 1` `renk = [1, 1, 0, 0, 0]`

ÇAĞRI YIĞINI

- `donguVar(1)`
- `donguVar(0)`
- `main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	1	0	0	0

0 beyaz · 1 gri · 2 siyah

Kenar 1 → 2: 2 beyaz → özyinelemeli çağrı.

Adım 4 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
▶      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

DEĞİŞKENLER

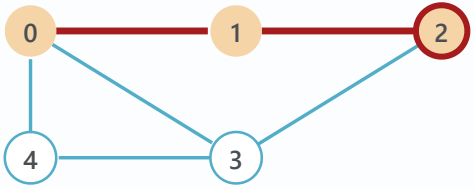
`v = 1` `u = 2`

`renk = [1, 1, 0, 0, 0]`

ÇAĞRI YIĞINI

- **donguVar(1)**
- donguVar(0)
- main()

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	1	1	0	0

0 beyaz · 1 gri · 2 siyah

renk[2] = 1 (gri): 2 şu anki DFS yolunda.

Adım 5 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
▶    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

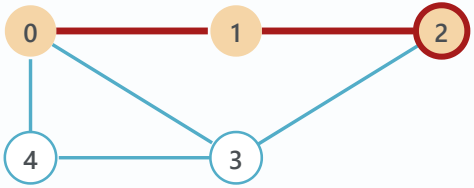
DEĞİŞKENLER

`v = 2` `renk = [1, 1, 1, 0, 0]`

ÇAĞRI YIĞINI

- `donguVar(2)`
- `donguVar(1)`
- `donguVar(0)`
- `main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	1	2	0	0

0 beyaz · 1 gri · 2 siyah

2'nin çıkan kenarı yok: **renk[2] = 2 (siyah)**, false döner; yığından çıkar.

Adım 6 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
▶    renk[v] = 2;
▶    return false;
11 }
```

DEĞİŞKENLER

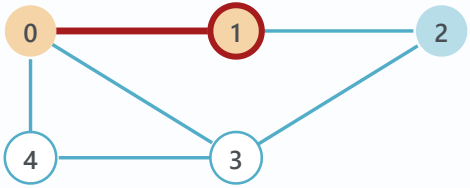
`v = 2` `dönüş = false`

`renk = [1, 1, 2, 0, 0]`

ÇAĞRI YIĞINI

- **donguVar(2)**
- donguVar(1)
- donguVar(0)
- main()

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	2	2	0	0

0 beyaz · 1 gri · 2 siyah

1'in tüm komşuları bitti: **renk[1] = 2 (siyah)**, false döner; yığından çıkar.

Adım 7 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
▶    renk[v] = 2;
▶    return false;
11 }
```

DEĞİŞKENLER

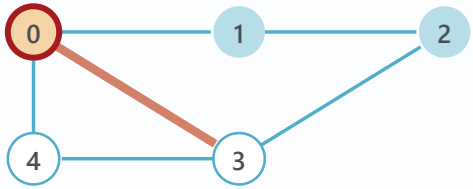
`v = 1` `dönüş = false`

`renk = [1, 2, 2, 0, 0]`

ÇAĞRI YIĞINI

- **donguVar(1)**
- donguVar(0)
- main()

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	2	2	0	0

0 beyaz · 1 gri · 2 siyah

Kenar 0 → 3: 3 beyaz → özyinelemeli çağrı.

Adım 8 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
▶      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

DEĞİŞKENLER

`v = 0` `u = 3`

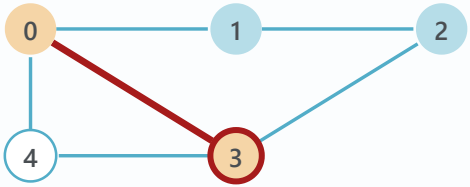
`renk = [1, 2, 2, 0, 0]`

ÇAĞRI YIĞINI

`donguVar(0)`

`main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	2	2	1	0

0 beyaz · 1 gri · 2 siyah

renk[3] = 1 (gri): 3 şu anki DFS yolunda.

Adım 9 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
▶    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

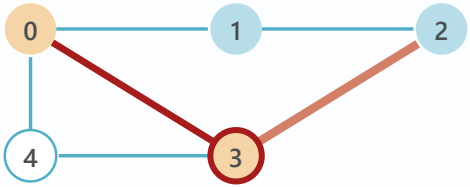
DEĞİŞKENLER

`v = 3` `renk = [1, 2, 2, 1, 0]`

ÇAĞRI YIĞINI

- `donguVar(3)`
- `donguVar(0)`
- `main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	2	2	1	0

0 beyaz · 1 gri · 2 siyah

Kenar 3 → 2: 2 **siyah** (bitmiş). Döngü değil: 2'den geri 3'e yol yok. Sadece gri düğüm döngü demektir.

Adım 10 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
▶      if (renk[u] == 1) return true;  // geri kenar
▶      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

DEĞİŞKENLER

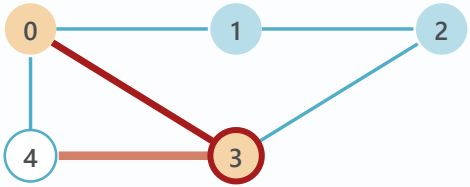
`v = 3` `u = 2`

`renk = [1, 2, 2, 1, 0]`

ÇAĞRI YIĞINI

- `donguVar(3)`
- `donguVar(0)`
- `main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	2	2	1	0

0 beyaz · 1 gri · 2 siyah

Kenar 3 → 4: 4 beyaz → özyinelemeli çağrı.

Adım 11 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
▶      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

DEĞİŞKENLER

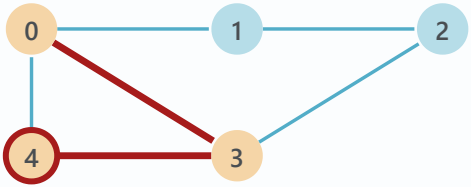
`v = 3` `u = 4`

`renk = [1, 2, 2, 1, 0]`

ÇAĞRI YIĞINI

- `donguVar(3)`
- `donguVar(0)`
- `main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	2	2	1	1

0 beyaz · 1 gri · 2 siyah

renk[4] = 1 (gri): 4 şu anki DFS yolunda.

Adım 12 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
▶    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

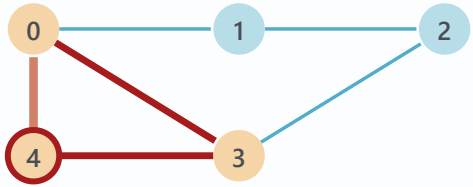
DEĞİŞKENLER

`v = 4` `renk = [1, 2, 2, 1, 1]`

ÇAĞRI YIĞINI

- `donguVar(4)`
- `donguVar(3)`
- `donguVar(0)`
- `main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	2	2	1	1

0 beyaz · 1 gri · 2 siyah

geri kenar → DÖNGÜ

Kenar 4 → 0: 0 **gri**, yani yolda! Bu bir **geri kenar** → döngü 0 → 3 → 4 → 0 bulundu.

Adım 13 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
▶      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
7        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

DEĞİŞKENLER

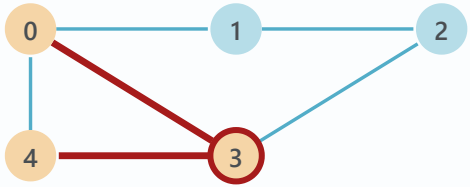
`v = 4` `u = 0`

`renk = [1, 2, 2, 1, 1]`

ÇAĞRI YIĞINI

- **donguVar(4)**
- donguVar(3)
- donguVar(0)
- main()

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	2	2	1	1

0 beyaz · 1 gri · 2 siyah

geri kenar → DÖNGÜ

Alttaki çağrı **true** döndü → 3 de true döndürür.

Adım 14 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
▶        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

DEĞİŞKENLER

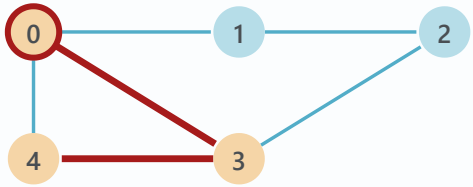
`v = 3` `u = 4` `dönüş = true`

`renk = [1, 2, 2, 1, 1]`

ÇAĞRI YIĞINI

- `donguVar(3)`
- `donguVar(0)`
- `main()`

Kod İzleme: yönlü çizgede döngü (3 renkli DFS)



v	0	1	2	3	4
renk	1	2	2	1	1

0 beyaz · 1 gri · 2 siyah

geri kenar → DÖNGÜ

Alttaki çağrı **true** döndü → 0 da true döndürür.

Adım 15 / 15

```
1  // renk: 0 beyaz, 1 gri (yolda), 2 siyah
2  boolean donguVar(int v, int[] renk) {
3    renk[v] = 1;
4    for (int u : komsu.get(v)) {
5      if (renk[u] == 1) return true;  // geri kenar
6      if (renk[u] == 0 && donguVar(u, renk))
▶        return true;
8    }
9    renk[v] = 2;
10   return false;
11 }
```

DEĞİŞKENLER

`v = 0` `u = 3` `dönüş = true`

`renk = [1, 2, 2, 1, 1]`

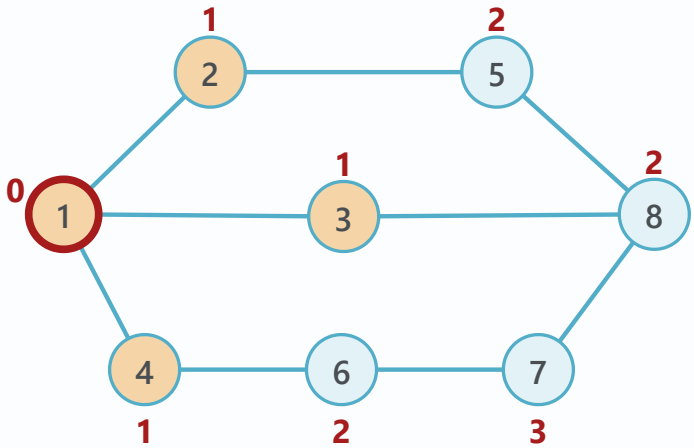
ÇAĞRI YIĞINI

- `donguVar(0)`
- `main()`

EN KISA YOL

Ağırlıksız En Kısa Yol Problemi

- Kaynak düğümden diğer **tüm düğümlere** olan en kısa yolları bulur (tek kaynaklı en kısa yol).
- Her kenarın aynı ağırlığa (1) sahip olduğu varsayılır: yol uzunluğu = kenar sayısı.
- Çözüm: **BFS**. Düzey k'daki düğümün uzaklığı tam olarak k'dır.



Kaynak 1; düğümlerin yanındaki sayı 1'e uzaklık

7'ye iki en kısa yol var: 1 → 4 → 6 → 7 ve 1 → 3 → 8 → 7 (ikisi de 3 kenar). BFS bunlardan birini bulur; uzunluk her durumda 3'tür.

EN KISA YOL

Ağırlıksız En Kısa Yol Algoritması

Sözde kod

```
mesafe[tüm v] = ∞
mesafe[s] = 0
kuyruk.ekle(s)
while kuyruk boş değil:
  w = kuyruk.çıkar()
  for w'nin her komşusu u:
    if mesafe[u] == ∞:
      mesafe[u] = mesafe[w] + 1
      kuyruk.ekle(u)
return mesafe
```

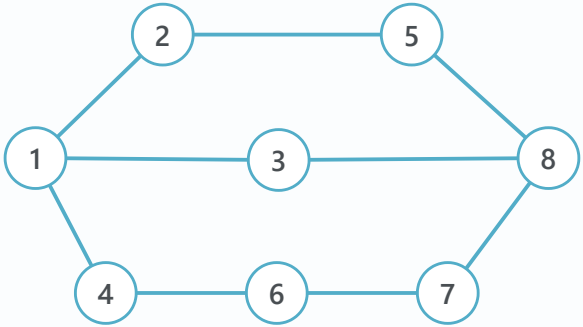
"mesafe = ∞" aynı zamanda "ziyaret edilmedi" demektir; ayrı bir dizi gerekmez. Java'da `cizge` : düğüm → komşu listesi (`Map<Integer, List<Integer>>`), çünkü düğümler 1'den başlayarak numaralanmış.

Java · O(n + m)

```
Map<Integer, Integer> enKisaYol(int s) {
  final int INF = Integer.MAX_VALUE;  // ∞
  Map<Integer, Integer> mesafe =
      new HashMap<>();
  for (int v : cizge.keySet())
    mesafe.put(v, INF);
  mesafe.put(s, 0);
  Queue<Integer> q = new ArrayDeque<>();
  q.offer(s);
  while (!q.isEmpty()) {
    int w = q.poll();
    for (int u : cizge.get(w))
      if (mesafe.get(u) == INF) {
        mesafe.put(u, mesafe.get(w) + 1);
        q.offer(u);
      }
  }
  return mesafe;
}
```

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
▶    for (int v : cizge.keySet())
▶      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	∞	∞	∞	∞	∞	∞	∞	∞
q	(boş)							

Kaynak s = 1. Tüm mesafeler **INF** (∞): "henüz ulaşılmadı" = "ziyaret edilmedi".

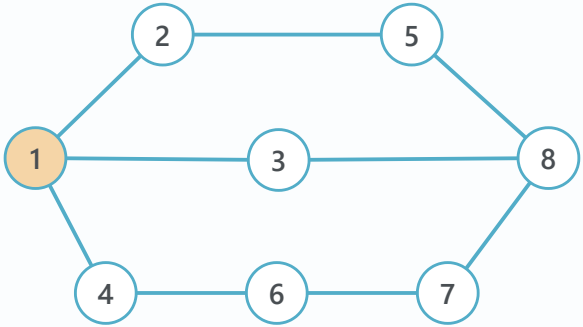
Adım 1 / 13

DEĞİŞKENLER

`s = 1`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
▶    mesafe.put(s, 0);
▶    Queue<Integer> q = new ArrayDeque<>();
▶    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	∞	∞	∞	∞	∞	∞	∞
q	1							

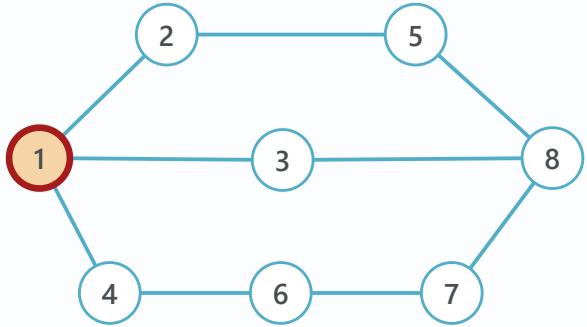
mesafe[1] = 0 ve 1 kuyruğa girer.

DEĞİŞKENLER

`s = 1`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
▶      int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	∞	∞	∞	∞	∞	∞	∞
q	(boş)							

1 çıkarılır; mesafe[1] = 0. Komşuları 2, 3, 4 incelenir.

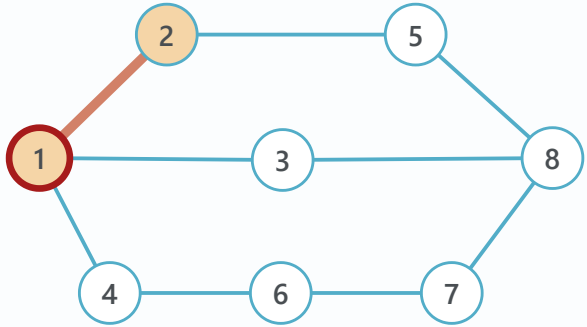
Adım 3 / 13

DEĞİŞKENLER

`s = 1` `w = 1`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
▶          mesafe.put(u, mesafe.get(w) + 1);
▶          q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	∞	∞	∞	∞	∞	∞
q	2							

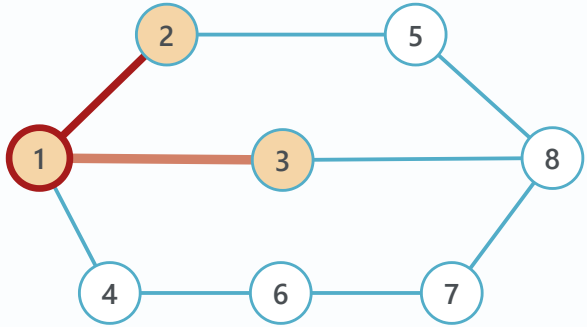
mesafe[2] = INF → **mesafe[2] = 0 + 1 = 1**, 2 kuyruğa eklenir.

DEĞİŞKENLER

`s = 1` `w = 1` `u = 2`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
▶          mesafe.put(u, mesafe.get(w) + 1);
▶          q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	∞	∞	∞	∞	∞
q	2	3						

mesafe[3] = INF → **mesafe[3] = 0 + 1 = 1**, 3 kuyruğa eklenir.

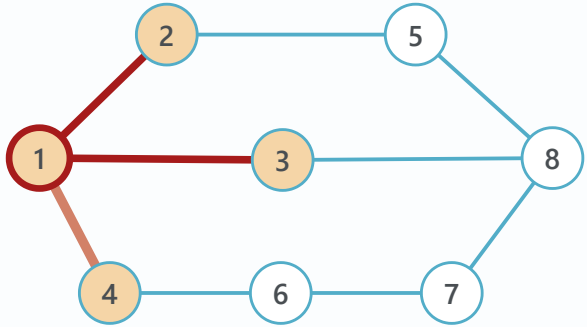
Adım 5 / 13

DEĞİŞKENLER

`s = 1` `w = 1` `u = 3`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
▶          mesafe.put(u, mesafe.get(w) + 1);
▶          q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	∞	∞	∞	∞
q	2	3	4					

mesafe[4] = INF → **mesafe[4] = 0 + 1 = 1**, 4 kuyruğa eklenir.

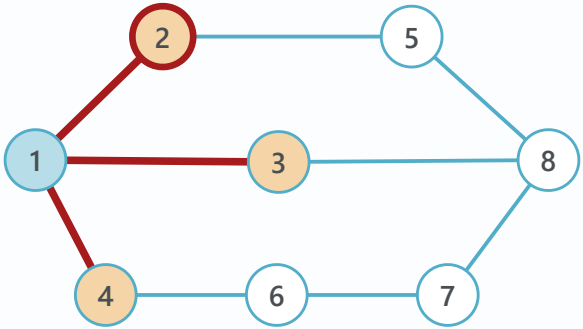
Adım 6 / 13

DEĞİŞKENLER

`s = 1` `w = 1` `u = 4`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
▶      int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	∞	∞	∞	∞
q	3	4						

2 çıkarılır; mesafe[2] = 1. Komşuları 1, 5 incelenir.

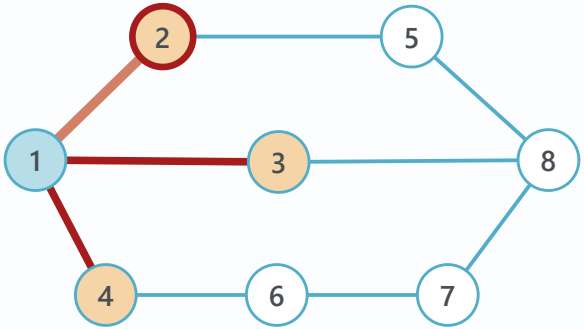
Adım 7 / 13

DEĞİŞKENLER

`s = 1` `w = 2`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	∞	∞	∞	∞
q	3	4						

1: mesafesi zaten belli (0) → değişmez. BFS'de ilk bulunan mesafe en kısadır.

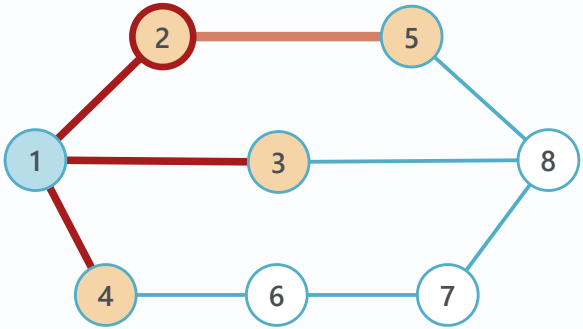
Adım 8 / 13

DEĞİŞKENLER

`s = 1` `w = 2` `u = 1`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
▶          mesafe.put(u, mesafe.get(w) + 1);
▶          q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	∞	∞	∞
q	3	4	5					

mesafe[5] = INF → **mesafe[5] = 1 + 1 = 2**, 5 kuyruğa eklenir.

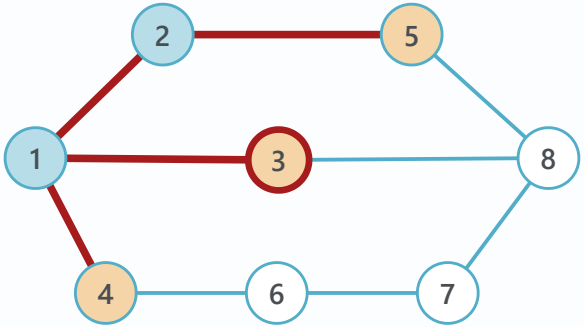
Adım 9 / 13

DEĞİŞKENLER

`s = 1` `w = 2` `u = 5`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
▶      int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	∞	∞	∞
q	4	5						

3 çıkarılır; mesafe[3] = 1. Komşuları 1, 8 incelenir.

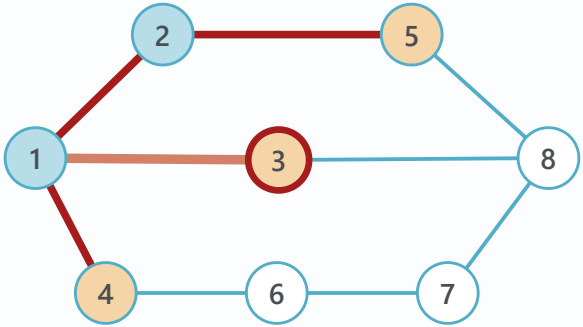
Adım 10 / 13

DEĞİŞKENLER

`s = 1` `w = 3`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	∞	∞	∞
q	4	5						

1: mesafesi zaten belli (0) → değişmez. BFS'de ilk bulunan mesafe en kısadır.

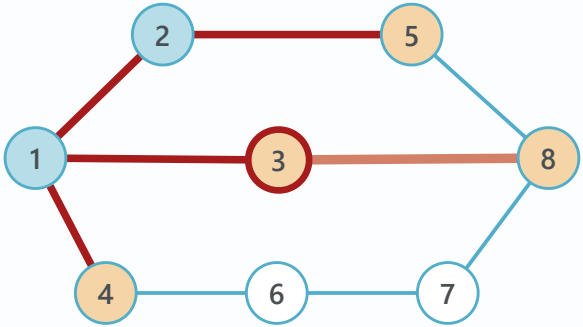
Adım 11 / 13

DEĞİŞKENLER

`s = 1` `w = 3` `u = 1`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
▶          mesafe.put(u, mesafe.get(w) + 1);
▶          q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	∞	∞	2
q	4	5	8					

mesafe[8] = INF → **mesafe[8] = 1 + 1 = 2**, 8 kuyruğa eklenir.

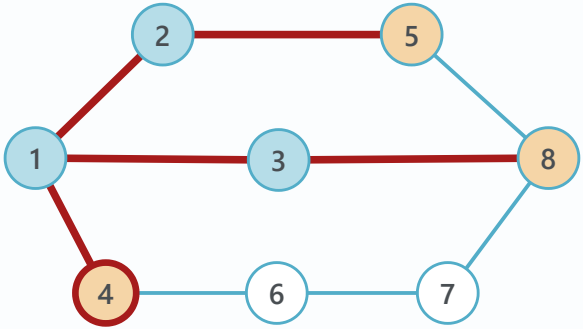
Adım 12 / 13

DEĞİŞKENLER

`s = 1` `w = 3` `u = 8`

Kod İzleme: `enKisaYol(1)`

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
▶      int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	∞	∞	2
q	5	8						

4 çıkarılır; mesafe[4] = 1. Komşuları 1, 6 incelenir.

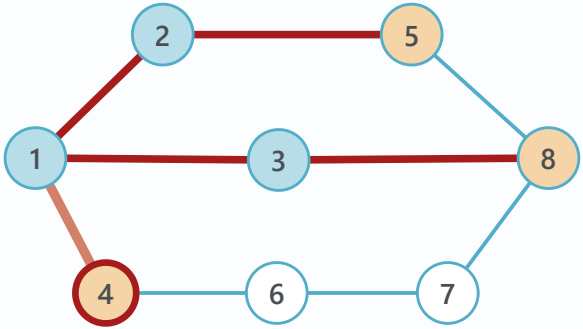
Adım 13 / 13

DEĞİŞKENLER

`s = 1` `w = 4`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	∞	∞	2
q	5	8						

1: mesafesi zaten belli (0) → değişmez. BFS'de ilk bulunan mesafe en kısadır.

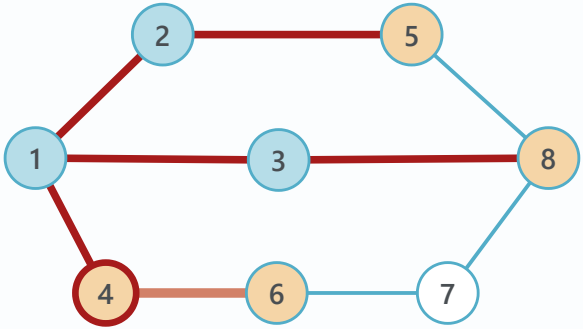
Adım 1 / 12

DEĞİŞKENLER

`s = 1` `w = 4` `u = 1`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
▶          mesafe.put(u, mesafe.get(w) + 1);
▶          q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	∞	2
q	5	8	6					

mesafe[6] = INF → **mesafe[6] = 1 + 1 = 2**, 6 kuyruğa eklenir.

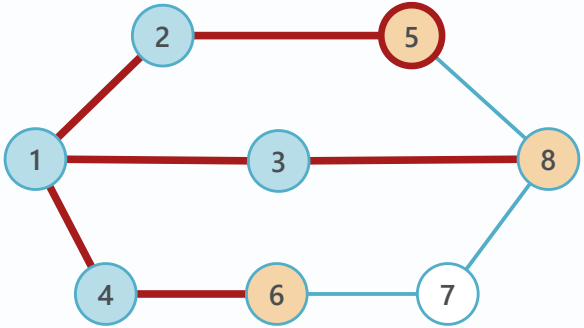
Adım 2 / 12

DEĞİŞKENLER

`s = 1` `w = 4` `u = 6`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
▶      int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	∞	2
q	8	6						

5 çıkarılır; mesafe[5] = 2. Komşuları 2, 8 incelenir.

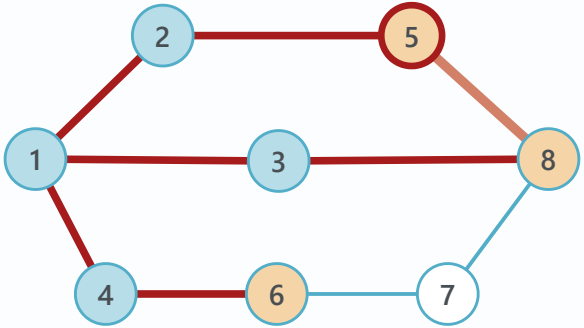
Adım 3 / 12

DEĞİŞKENLER

`s = 1` `w = 5`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	∞	2
q	8	6						

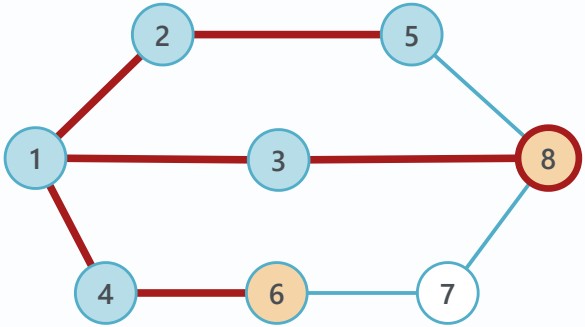
2, 8: mesafesi zaten belli (1, 2) → değişmez. BFS'de ilk bulunan mesafe en kısadır.

DEĞİŞKENLER

`s = 1` `w = 5` `u = 8`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
▶      int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	∞	2
q	6							

8 çıkarılır; mesafe[8] = 2. Komşuları 3, 5, 7 incelenir.

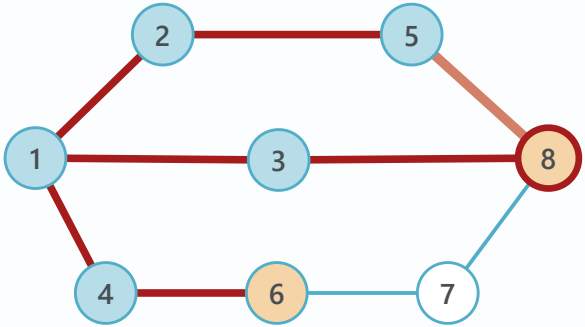
Adım 5 / 12

DEĞİŞKENLER

`s = 1` `w = 8`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	∞	2
q	6							

3, 5: mesafesi zaten belli (1, 2) → değişmez. BFS'de ilk bulunan mesafe en kısadır.

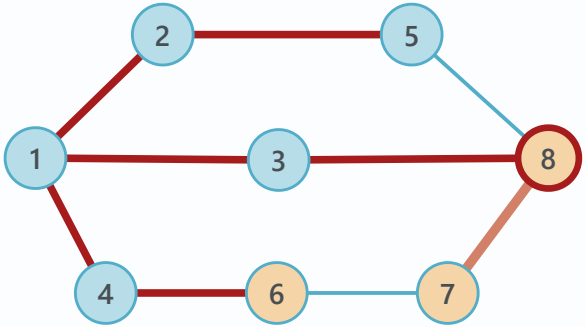
Adım 6 / 12

DEĞİŞKENLER

`s = 1` `w = 8` `u = 5`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
▶          mesafe.put(u, mesafe.get(w) + 1);
▶          q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	3	2

q							
	6	7					

mesafe[7] = INF → **mesafe[7] = 2 + 1 = 3**, 7 kuyruğa eklenir.

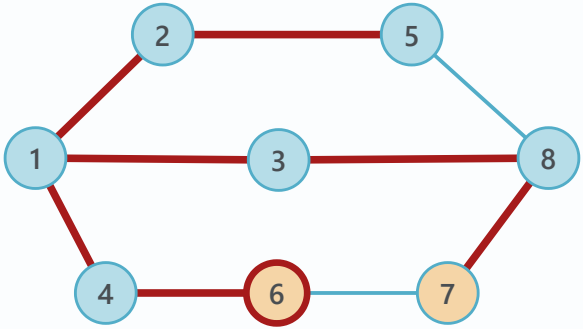
Adım 7 / 12

DEĞİŞKENLER

`s = 1` `w = 8` `u = 7`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
▶      int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	3	2

q	7						

6 çıkarılır; mesafe[6] = 2. Komşuları 4, 7 incelenir.

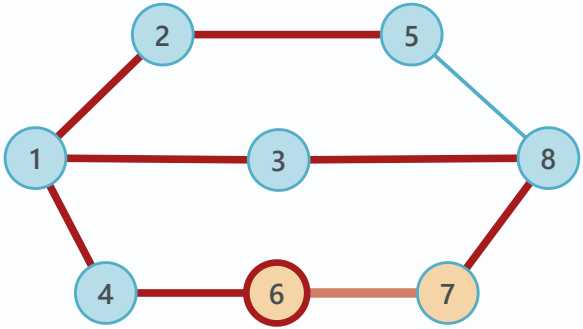
Adım 8 / 12

DEĞİŞKENLER

`s = 1` `w = 6`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	3	2
q	7							

4, 7: mesafesi zaten belli (1, 3) → değişmez. BFS'de ilk bulunan mesafe en kısadır.

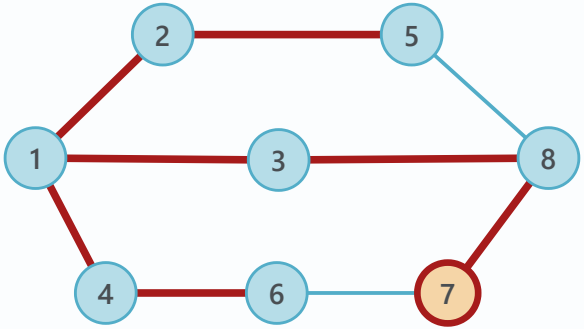
Adım 9 / 12

DEĞİŞKENLER

`s = 1` `w = 6` `u = 7`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
▶      int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	3	2

q	(boş)						

7 çıkarılır; mesafe[7] = 3. Komşuları 6, 8 incelenir.

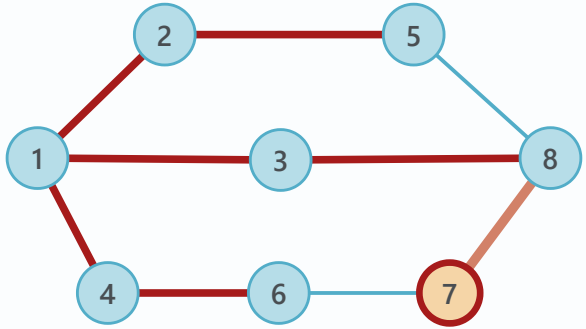
Adım 10 / 12

DEĞİŞKENLER

`s = 1` `w = 7`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
▶        if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
18   return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	3	2

q	(boş)						

6, 8: mesafesi zaten belli (2, 2) → değişmez. BFS'de ilk bulunan mesafe en kısadır.

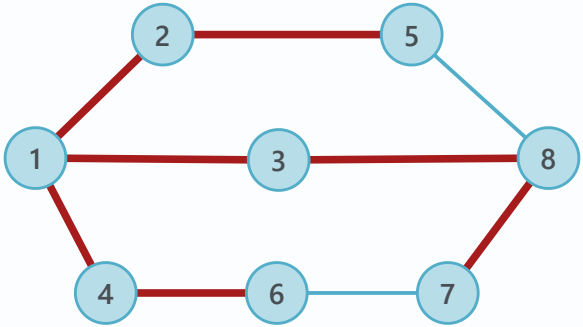
Adım 11 / 12

DEĞİŞKENLER

`s = 1` `w = 7` `u = 8`

Kod İzleme: `enKisaYol(1)` (devam)

```
1  Map<Integer, Integer> enKisaYol(int s) {
2    final int INF = Integer.MAX_VALUE;  // ∞
3    Map<Integer, Integer> mesafe =
4        new HashMap<>();
5    for (int v : cizge.keySet())
6      mesafe.put(v, INF);
7    mesafe.put(s, 0);
8    Queue<Integer> q = new ArrayDeque<>();
9    q.offer(s);
10   while (!q.isEmpty()) {
11     int w = q.poll();
12     for (int u : cizge.get(w))
13       if (mesafe.get(u) == INF) {
14         mesafe.put(u, mesafe.get(w) + 1);
15         q.offer(u);
16       }
17   }
▶    return mesafe;
19 }
```



	1	2	3	4	5	6	7	8
mesafe	0	1	1	1	2	2	3	2

q	(boş)						

Kuyruk boş → `return mesafe`. 7'ye uzaklık 3 (1 → 3 → 8 → 7 bulundu; 1 → 4 → 6 → 7 de 3 kenar). O(n + m).

Adım 12 / 12

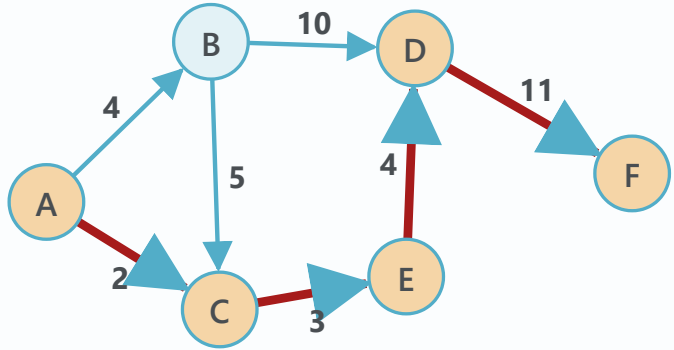
DEĞİŞKENLER

`s = 1`

EN KISA YOL

Ağırlıklı En Kısa Yol Problemi

- Kaynak düğümden diğer tüm düğümlere olan en kısa yolları bulur.
- Her kenarın farklı bir ağırlığı olabilir: uzaklık, süre, maliyet...
- Yol uzunluğu = kenar ağırlıklarının toplamı.



A'dan F'ye en kısa yol: A → C → E → D → F = 2 + 3 + 4 + 11 = **20**

BFS burada yetmez

En az kenarlı yol A → B → D → F (3 kenar) ama uzunluğu 4 + 10 + 11 = **25**

.

Çözüm: Dijkstra algoritması

Edsger W. Dijkstra, 1956. Ağırlıklar **negatif olmamalıdır**

.

EN KISA YOL

Dijkstra Algoritması: Fikir ve Sözde Kod

- Kesinleşmemiş düğümlerden mesafesi en küçük olanı seç.
- Komşuların mesafesini **gevşetme** ile güncelle.
- Seçim için öncelikli kuyruk, yolu kurmak için `önceki[]` kullan.

```
Dijkstra(G, s):
  for her düğüm v:
    mesafe[v] = ∞
    önceki[v] = tanımsız
  mesafe[s] = 0
  Q = tüm düğümler  // öncelikli kuyruk
  while Q boş değil:
    u = Q.çıkar()   // en küçük mesafe[u]
    for u'nun Q'daki her komşusu v:
      alt = mesafe[u] + ağırlık(u, v)
      if alt < mesafe[v]:
        mesafe[v] = alt
        önceki[v] = u
        // Q'da v'nin önceliğini azalt
```

EN KISA YOL

Dijkstra Algoritması: Java

```java
long[] dijkstra(List<List<int[]>> adj, int s, int[] onceki) { // ağırlıklar ≥ 0
    long[] d = new long[adj.size()];
    Arrays.fill(d, Long.MAX_VALUE);  d[s] = 0;  Arrays.fill(onceki, -1);
    PriorityQueue<long[]> pq =                      // {düğüm, mesafe}
        new PriorityQueue<>((a, b) -> Long.compare(a[1], b[1]));
    pq.add(new long[]{s, 0});
    while (!pq.isEmpty()) {
        long[] e = pq.poll();  int u = (int) e[0];
        if (e[1] > d[u]) continue;                  // bayat kayıt: atla
        for (int[] k : adj.get(u)) {
            int v = k[0];  long alt = d[u] + k[1];
            if (alt < d[v]) { d[v] = alt; onceki[v] = u; pq.add(new long[]{v, alt}); }
        }
    }
    return d;
}
```

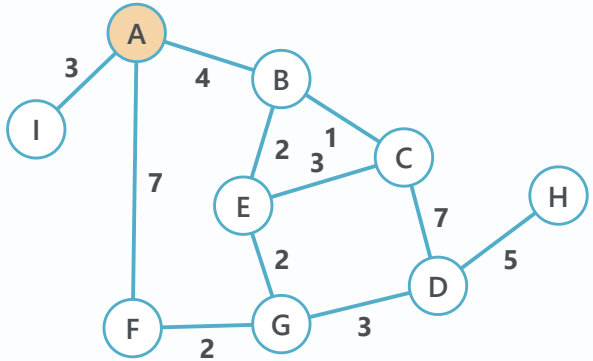
Öncelik = mesafe (düğüm numarası değil). Java'nın `PriorityQueue` 'sunda "önceliği azalt" yok: yeni kayıt eklenir, eskisi çıkınca *bayat* olarak atlanır. `onceki[v] = u` en kısa yol ağacını tutar; yol hedeften geriye izlenir.

Kod İzleme: `dijkstra(adj, A)`

```
// d[] = ∞, d[s] = 0, onceki[] = -1
pq.add(new long[]{s, 0});
while (!pq.isEmpty()) {
  long[] e = pq.poll();
  int u = (int) e[0];
  if (e[1] > d[u]) continue;  // bayat
  for (int[] k : adj.get(u)) {
    int v = k[0];
    long alt = d[u] + k[1];
    if (alt < d[v]) {
      d[v] = alt;
      onceki[v] = u;
      pq.add(new long[]{v, alt});
    }
  }
}
```

DEĞİŞKENLER

`s` = **A**



	A	B	C	D	E	F	G	H	I
d[]	0	∞	∞	∞	∞	∞	∞	∞	∞
onceki	–	–	–	–	–	–	–	–	–

pq (mesafeye göre sıralı gösterim; soluk = bayat)

A,0

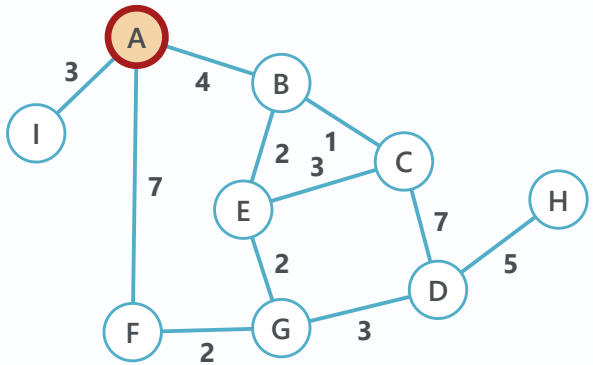
Kaynak A: d[A] = 0, diğerleri ∞. Kuyruğa {A, 0} eklenir. Öncelik = **mesafe**.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {A,0}` `u = A`



	A	B	C	D	E	F	G	H	I
d[]	0	∞	∞	∞	∞	∞	∞	∞	∞
onceki	–	–	–	–	–	–	–	–	–

pq (mesafeye göre sıralı gösterim; soluk = bayat)

(boş)

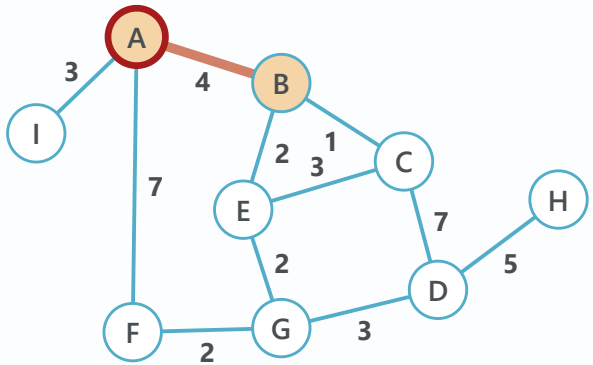
En küçük kayıt {A, 0} çıkar; 0 = d[A] (güncel) → **A kesinleşir**. Komşuları B, F, I gevşetilir.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
       int v = k[0];
       long alt = d[u] + k[1];
       if (alt < d[v]) {
         d[v] = alt;
         onceki[v] = u;
         pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {A,0}` `u = A` `v = B` `alt = 4`



	A	B	C	D	E	F	G	H	I
d[]	0	4	∞	∞	∞	∞	∞	∞	∞
onceki	–	A	–	–	–	–	–	–	–

pq (mesafeye göre sıralı gösterim; soluk = bayat)

B,4

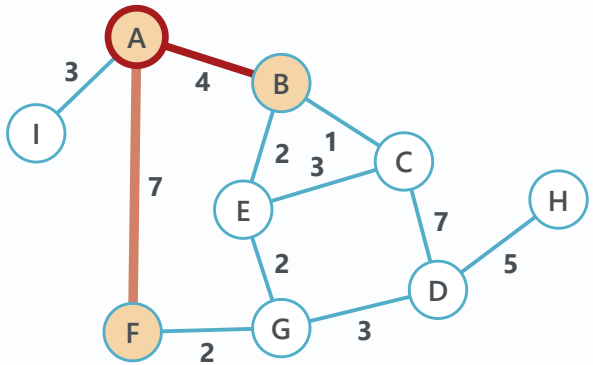
v = B: alt = 0 + 4 = **4** < ∞ → d[B] = 4, onceki[B] = A, {B, 4} kuyruğa.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
▶        d[v] = alt;
▶        onceki[v] = u;
▶        pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {A,0}` `u = A` `v = F` `alt = 7`



	A	B	C	D	E	F	G	H	I
d[]	0	4	∞	∞	∞	7	∞	∞	∞
onceki	–	A	–	–	–	A	–	–	–

pq (mesafeye göre sıralı gösterim; soluk = bayat)

B,4 | F,7

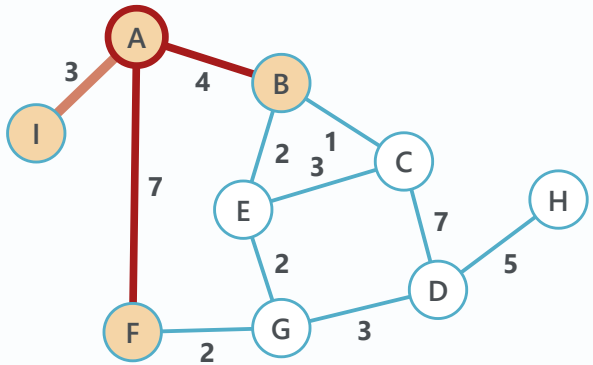
v = F: alt = 0 + 7 = **7** < ∞ → d[F] = 7, onceki[F] = A, {F, 7} kuyruğa.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
▶        d[v] = alt;
▶        onceki[v] = u;
▶        pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {A,0}` `u = A` `v = I` `alt = 3`



	A	B	C	D	E	F	G	H	I
d[]	0	4	∞	∞	∞	7	∞	∞	3
onceki	–	A	–	–	–	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

I,3 | B,4 | F,7

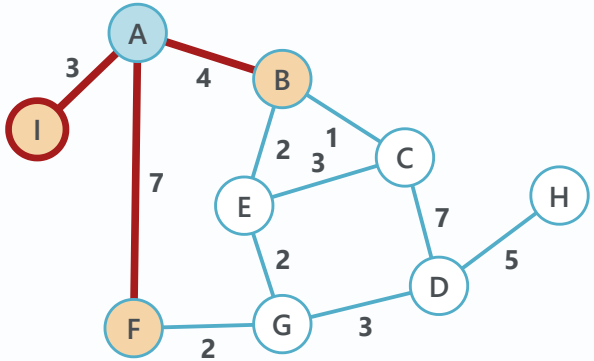
v = I: alt = 0 + 3 = **3** < ∞ → d[I] = 3, onceki[I] = A, {I, 3} kuyruğa.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {I,3}` `u = I`



	A	B	C	D	E	F	G	H	I
d[]	0	4	∞	∞	∞	7	∞	∞	3
onceki	–	A	–	–	–	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

B,4 F,7

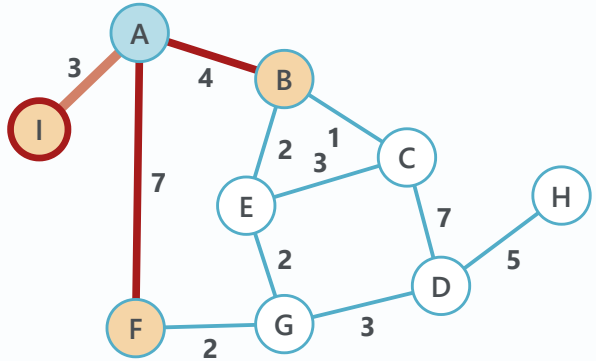
En küçük kayıt {I, 3} çıkar; 3 = d[I] (güncel) → **I kesinleşir**. Komşusu A gevşetilir.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {I,3}` `u = I` `v = A` `alt = 6`



	A	B	C	D	E	F	G	H	I
d[]	0	4	∞	∞	∞	7	∞	∞	3
onceki	–	A	–	–	–	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

B,4 | F,7

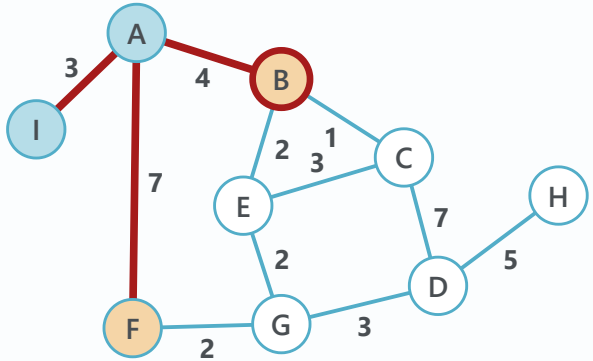
v = A: alt = 6 ≥ d[A] = 0 → güncelleme yok.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {B,4}` `u = B`



	A	B	C	D	E	F	G	H	I
d[]	0	4	∞	∞	∞	7	∞	∞	3
onceki	–	A	–	–	–	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

F,7

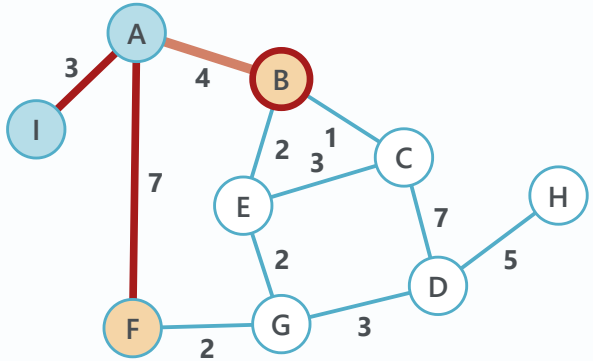
En küçük kayıt {B, 4} çıkar; 4 = d[B] (güncel) → **B kesinleşir**. Komşuları A, C, E gevşetilir.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {B,4}` `u = B` `v = A` `alt = 8`



	A	B	C	D	E	F	G	H	I
d[]	0	4	∞	∞	∞	7	∞	∞	3
onceki	–	A	–	–	–	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

F,7

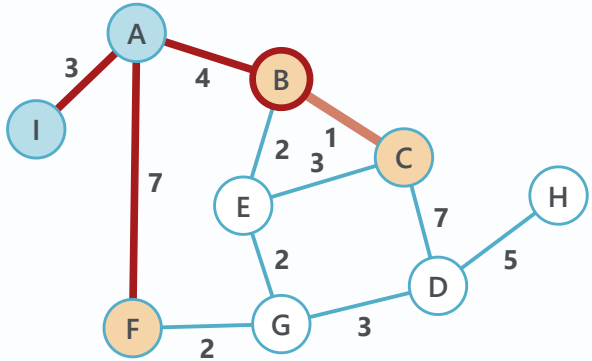
v = A: alt = 8 ≥ d[A] = 0 → güncelleme yok.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
▶        d[v] = alt;
▶        onceki[v] = u;
▶        pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {B,4}` `u = B` `v = C` `alt = 5`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	∞	∞	7	∞	∞	3
onceki	–	A	B	–	–	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

C,5 | F,7

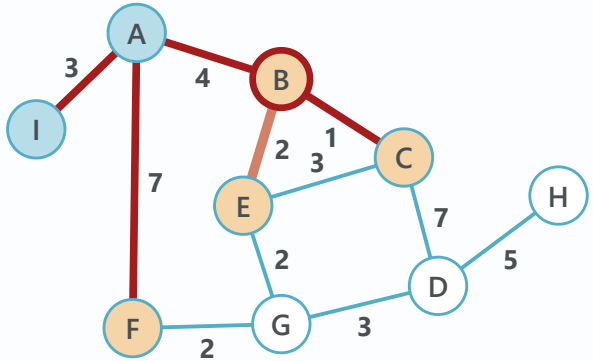
v = C: alt = 4 + 1 = **5** < ∞ → d[C] = 5, onceki[C] = B, {C, 5} kuyruğa.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
▶        d[v] = alt;
▶        onceki[v] = u;
▶        pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {B,4}` `u = B` `v = E` `alt = 6`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	∞	6	7	∞	∞	3
onceki	–	A	B	–	B	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

C,5 | E,6 | F,7

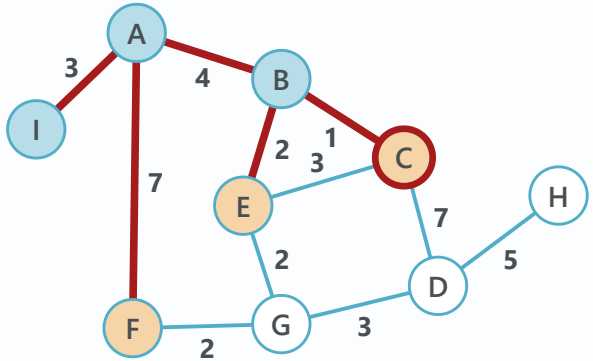
v = E: alt = 4 + 2 = **6** < ∞ → d[E] = 6, onceki[E] = B, {E, 6} kuyruğa.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {C,5}` `u = C`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	∞	6	7	∞	∞	3
onceki	–	A	B	–	B	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

E,6 F,7

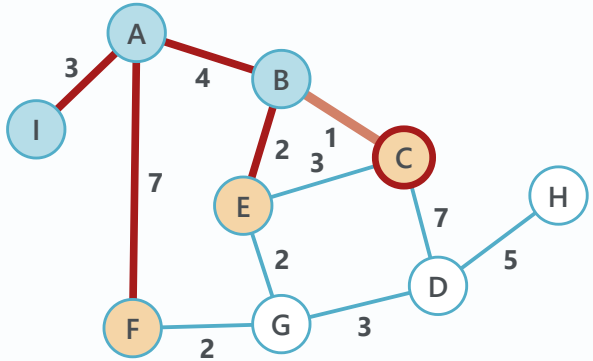
En küçük kayıt {C, 5} çıkar; 5 = d[C] (güncel) → **C kesinleşir**. Komşuları B, D, E gevşetilir.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {C,5}` `u = C` `v = B` `alt = 6`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	∞	6	7	∞	∞	3
onceki	–	A	B	–	B	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

E,6 | F,7

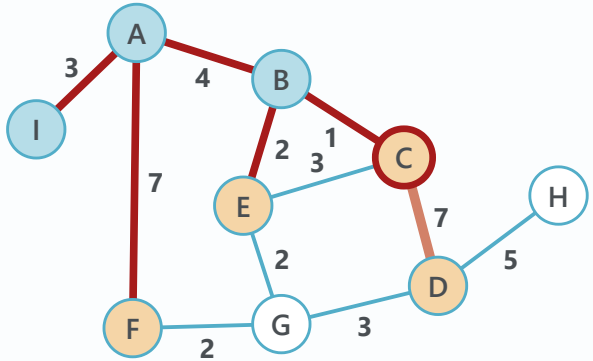
v = B: alt = 6 ≥ d[B] = 4 → güncelleme yok.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
▶        d[v] = alt;
▶        onceki[v] = u;
▶        pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {C,5}` `u = C` `v = D` `alt = 12`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	12	6	7	∞	∞	3
onceki	–	A	B	C	B	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

E,6 | F,7 | D,12

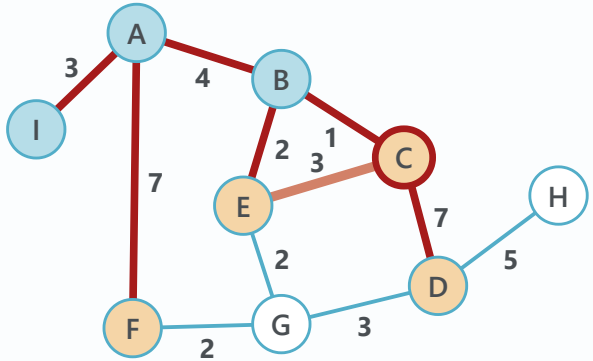
v = D: alt = 5 + 7 = **12** < ∞ → d[D] = 12, onceki[D] = C, {D, 12} kuyruğa.

Kod İzleme: `dijkstra(adj, A)`

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {C,5}` `u = C` `v = E` `alt = 8`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	12	6	7	∞	∞	3
onceki	–	A	B	C	B	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

E,6 | F,7 | D,12

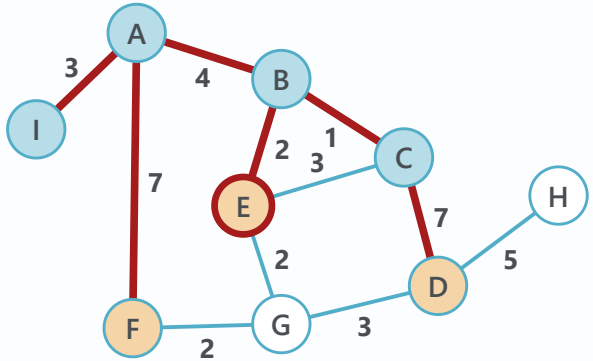
v = E: alt = 8 ≥ d[E] = 6 → güncelleme yok.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {E,6}` `u = E`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	12	6	7	∞	∞	3
onceki	–	A	B	C	B	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

F,7 D,12

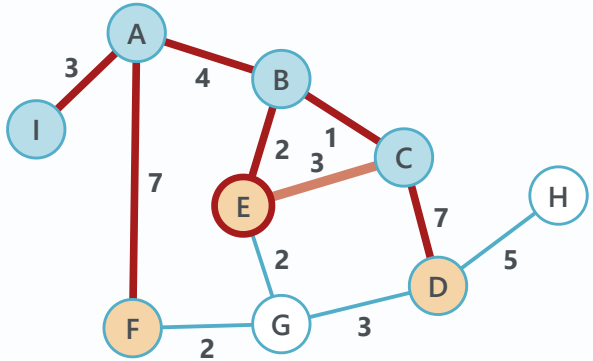
En küçük kayıt {E, 6} çıkar; 6 = d[E] (güncel) → **E kesinleşir**. Komşuları B, C, G gevşetilir.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {E,6}` `u = E` `v = C` `alt = 9`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	12	6	7	∞	∞	3
onceki	–	A	B	C	B	A	–	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

F,7 | D,12

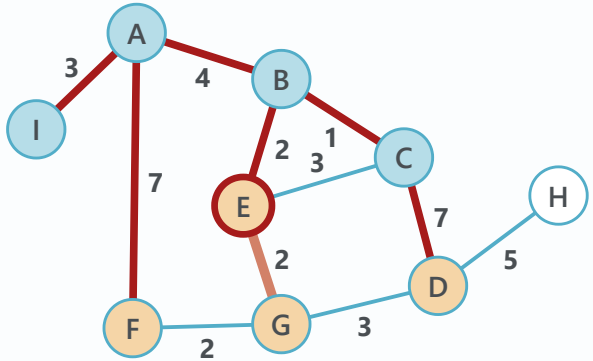
v = B: alt = 8 ≥ d[B] = 4; v = C: alt = 9 ≥ d[C] = 5 → güncelleme yok.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
▶        d[v] = alt;
▶        onceki[v] = u;
▶        pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {E,6}` `u = E` `v = G` `alt = 8`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	12	6	7	8	∞	3
onceki	–	A	B	C	B	A	E	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

F,7 | G,8 | D,12

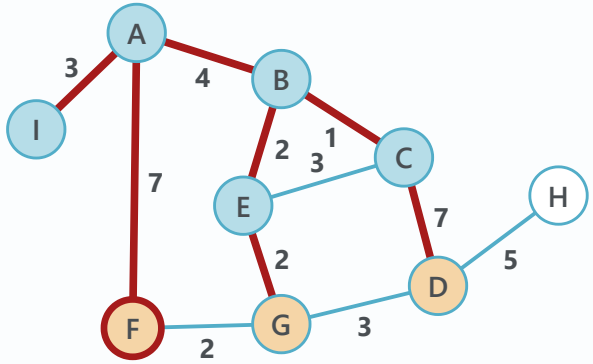
v = G: alt = 6 + 2 = **8** < ∞ → d[G] = 8, onceki[G] = E, {G, 8} kuyruğa.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {F,7}` `u = F`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	12	6	7	8	∞	3
onceki	–	A	B	C	B	A	E	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

`G,8` `D,12`

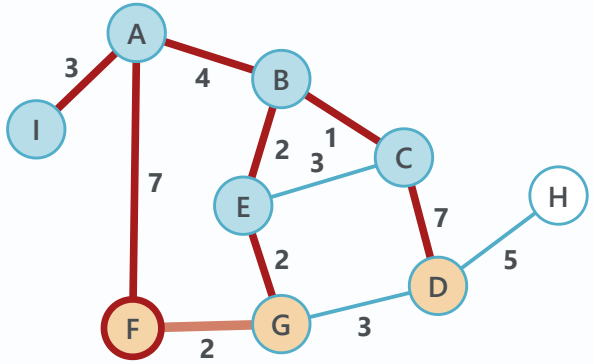
En küçük kayıt {F, 7} çıkar; 7 = d[F] (güncel) → **F kesinleşir**. Komşuları A, G gevşetilir.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {F,7}` `u = F` `v = G` `alt = 9`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	12	6	7	8	∞	3
onceki	–	A	B	C	B	A	E	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

G,8 | D,12

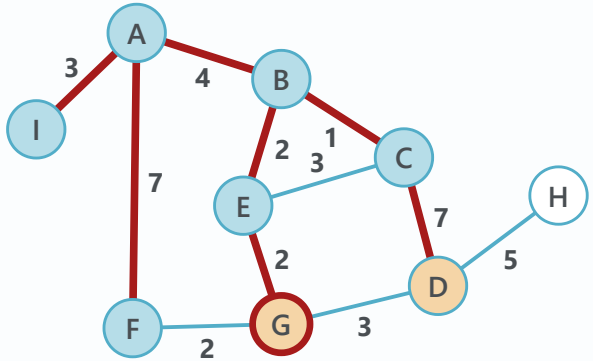
v = A: alt = 14 ≥ d[A] = 0; v = G: alt = 9 ≥ d[G] = 8 → güncelleme yok.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {G,8}` `u = G`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	12	6	7	8	∞	3
onceki	–	A	B	C	B	A	E	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

D,12

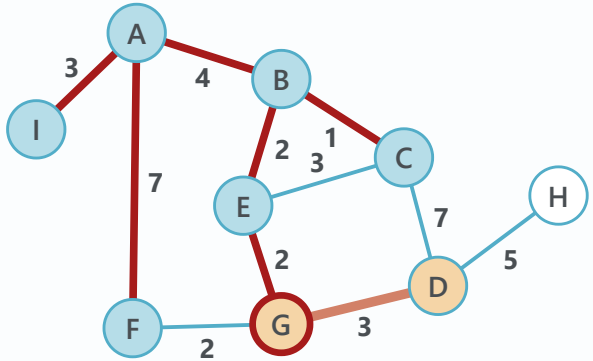
En küçük kayıt {G, 8} çıkar; 8 = d[G] (güncel) → **G kesinleşir**. Komşuları D, E, F gevşetilir.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
       int v = k[0];
       long alt = d[u] + k[1];
       if (alt < d[v]) {
         d[v] = alt;
         onceki[v] = u;
         pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {G,8}` `u = G` `v = D` `alt = 11`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	11	6	7	8	∞	3
onceki	–	A	B	G	B	A	E	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

D,11 ~~D,12~~

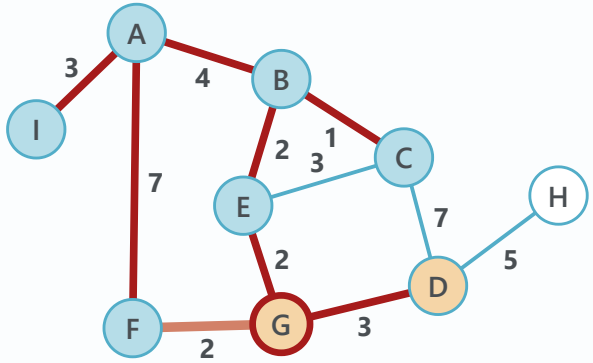
v = D: alt = 8 + 3 = **11** < 12 → d[D] = 11, onceki[D] = G, {D, 11} kuyruğa. Eski {D, 12} kaydı kuyrukta **bayat** kalır.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {G,8}` `u = G` `v = F` `alt = 10`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	11	6	7	8	∞	3
onceki	–	A	B	G	B	A	E	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

D,11 ~~D,12~~

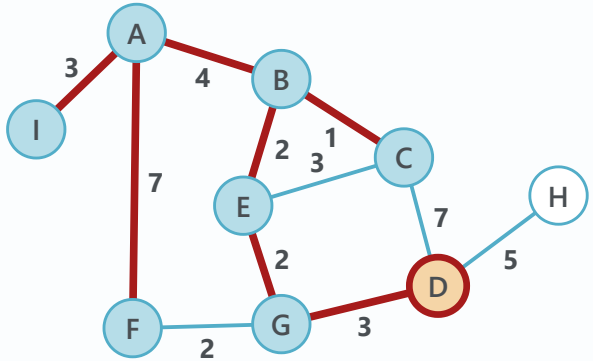
v = E: alt = 10 ≥ d[E] = 6; v = F: alt = 10 ≥ d[F] = 7 → güncelleme yok.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {D,11}` `u = D`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	11	6	7	8	∞	3
onceki	–	A	B	G	B	A	E	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

~~D,12~~

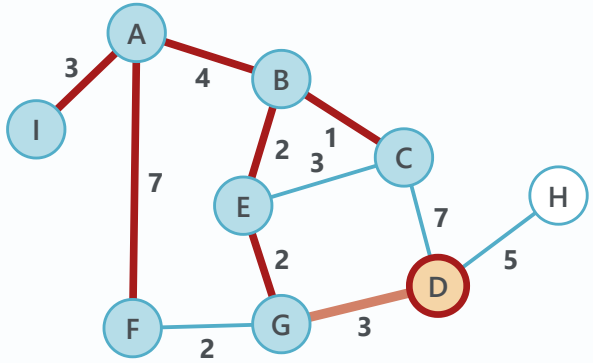
En küçük kayıt {D, 11} çıkar; 11 = d[D] (güncel) → **D kesinleşir**. Komşuları C, G, H gevşetilir.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {D,11}` `u = D` `v = G` `alt = 14`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	11	6	7	8	∞	3
onceki	–	A	B	G	B	A	E	–	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

~~D,12~~

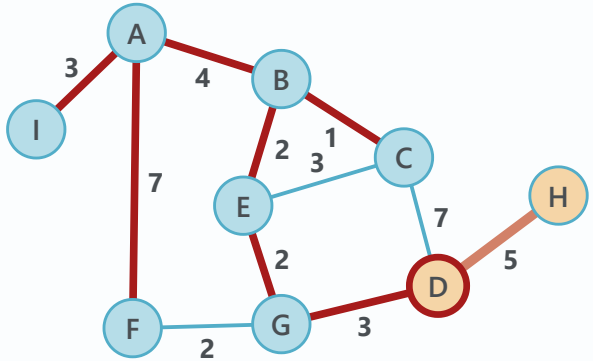
v = C: alt = 18 ≥ d[C] = 5; v = G: alt = 14 ≥ d[G] = 8 → güncelleme yok.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
▶        d[v] = alt;
▶        onceki[v] = u;
▶        pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {D,11}` `u = D` `v = H` `alt = 16`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	11	6	7	8	16	3
onceki	–	A	B	G	B	A	E	D	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

~~D,12~~ H,16

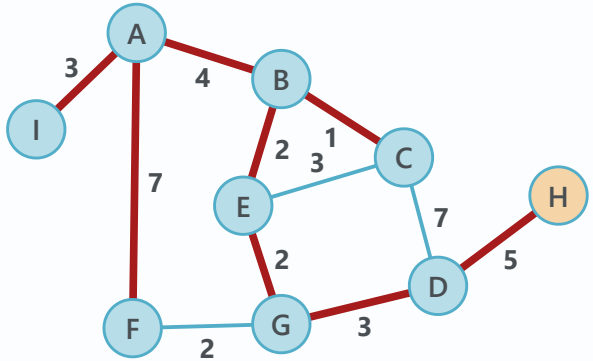
v = H: alt = 11 + 5 = **16** < ∞ → d[H] = 16, onceki[H] = D, {H, 16} kuyruğa.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {D,12}` `u = D`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	11	6	7	8	16	3
onceki	–	A	B	G	B	A	E	D	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

H,16

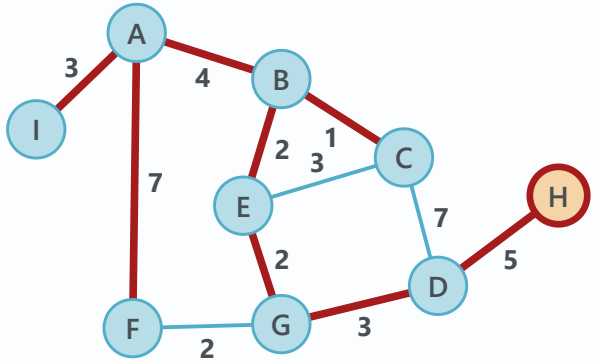
{D, 12} çıktı ama d[D] = 11 < 12: **bayat kayıt** (daha iyi yol sonradan bulundu) → `continue`, atlanır.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
▶    long[] e = pq.poll();
▶    int u = (int) e[0];
▶    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {H,16}` `u = H`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	11	6	7	8	16	3
onceki	–	A	B	G	B	A	E	D	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

(boş)

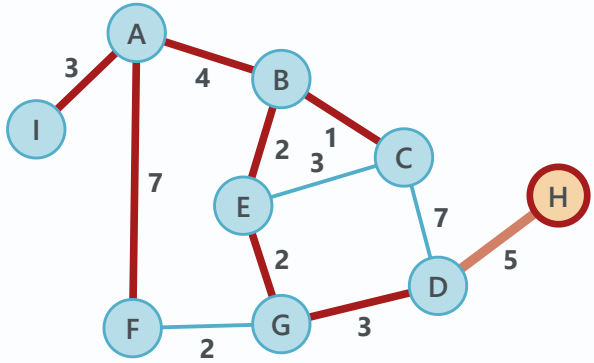
En küçük kayıt {H, 16} çıkar; 16 = d[H] (güncel) → **H kesinleşir**. Komşusu D gevşetilir.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
3  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
▶      int v = k[0];
▶      long alt = d[u] + k[1];
▶      if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

`e = {H,16}` `u = H` `v = D` `alt = 21`



	A	B	C	D	E	F	G	H	I
d[]	0	4	5	11	6	7	8	16	3
onceki	–	A	B	G	B	A	E	D	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

(boş)

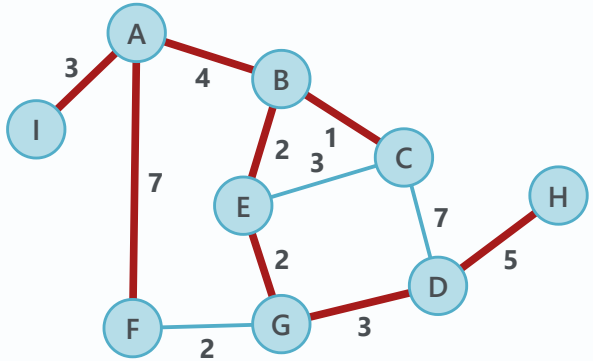
v = D: alt = 21 ≥ d[D] = 11 → güncelleme yok.

Kod İzleme: `dijkstra(adj, A)` (devam)

```
1  // d[] = ∞, d[s] = 0, onceki[] = -1
2  pq.add(new long[]{s, 0});
▶  while (!pq.isEmpty()) {
4    long[] e = pq.poll();
5    int u = (int) e[0];
6    if (e[1] > d[u]) continue;  // bayat
7    for (int[] k : adj.get(u)) {
8      int v = k[0];
9      long alt = d[u] + k[1];
10     if (alt < d[v]) {
11       d[v] = alt;
12       onceki[v] = u;
13       pq.add(new long[]{v, alt});
14     }
15   }
16 }
```

DEĞİŞKENLER

—



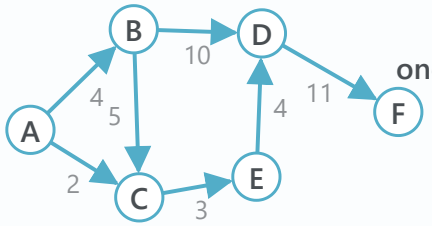
	A	B	C	D	E	F	G	H	I
d[]	0	4	5	11	6	7	8	16	3
onceki	–	A	B	G	B	A	E	D	A

pq (mesafeye göre sıralı gösterim; soluk = bayat)

(boş)

Kuyruk boş → bitti. onceki[] geri izlenir: H ← D ← G ← E ← B ← A. En kısa yol **A → B → E → G → D → H = 16**.

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y						

addFirst → hedeften kaynağa doğru, başa ekle

Dijkstra'nın bıraktığı `onceki[]` : her düğüm için en kısa yolda *kendinden önceki* düğüm. Boş liste y.

Adım 1 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
▶   LinkedList<Integer> y = new LinkedList<>();
3   for (int v = t; v != -1; v = onceki[v])
4     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

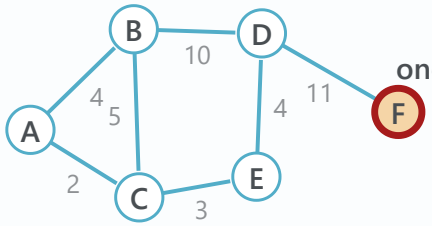
`y = []`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y						

addFirst → hedeften kaynağa doğru, başa ekle

Hedeften başla: **v = F**.

Adım 2 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
▶   for (int v = t; v != -1; v = onceki[v])
4     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

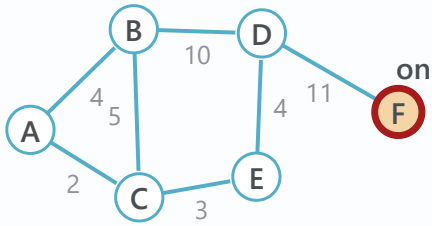
`v = F` `y = []`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y	F					

addFirst → hedeften kaynağa doğru, başa ekle

F listenin **başına** eklenir (sonra gelenler daha önceki düğümler).

Adım 3 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
3   for (int v = t; v != -1; v = onceki[v])
▶     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

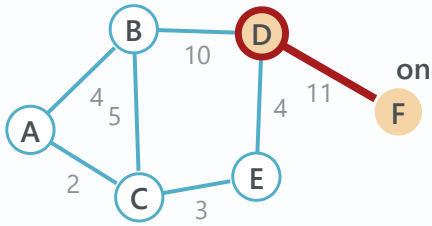
`v = F` `y = [F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y	F					

addFirst → hedeften kaynağa doğru, başa ekle

v = onceki[...] = D: bir adım geri.

Adım 4 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
▶   for (int v = t; v != -1; v = onceki[v])
4     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

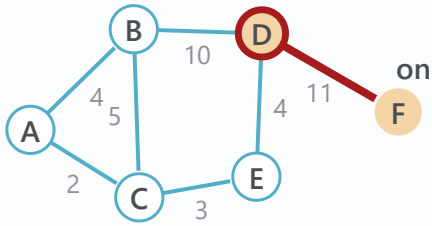
`v = D` `y = [F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y	D	F				

addFirst → hedeften kaynağa doğru, başa ekle

D listenin **başına** eklenir (sonra gelenler daha önceki düğümler).

Adım 5 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
3   for (int v = t; v != -1; v = onceki[v])
▶     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

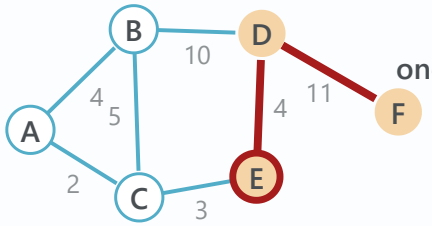
`v = D` `y = [D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y	D	F				

addFirst → hedeften kaynağa doğru, başa ekle

v = onceki[...] = E: bir adım geri.

Adım 6 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
▶   for (int v = t; v != -1; v = onceki[v])
4     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

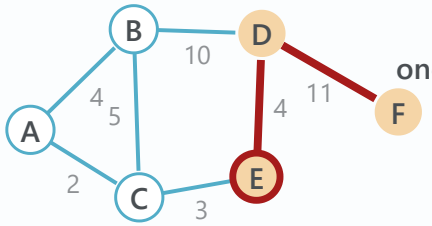
`v = E` `y = [D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y	E	D	F			

addFirst → hedeften kaynağa doğru, başa ekle

E listenin **başına** eklenir (sonra gelenler daha önceki düğümler).

Adım 7 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
3   for (int v = t; v != -1; v = onceki[v])
▶     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

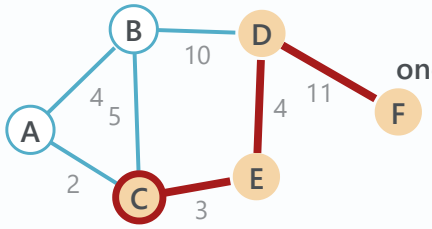
`v = E` `y = [E, D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y	E	D	F			

addFirst → hedeften kaynağa doğru, başa ekle

v = onceki[...] = C: bir adım geri.

Adım 8 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
▶   for (int v = t; v != -1; v = onceki[v])
4     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

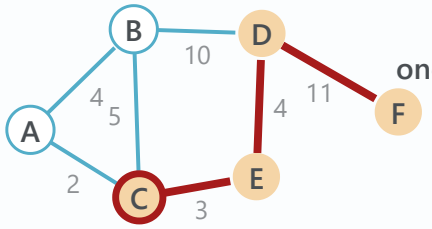
`v = C` `y = [E, D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y	C	E	D	F		

addFirst → hedeften kaynağa doğru, başa ekle

C listenin **başına** eklenir (sonra gelenler daha önceki düğümler).

Adım 9 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
3   for (int v = t; v != -1; v = onceki[v])
▶     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

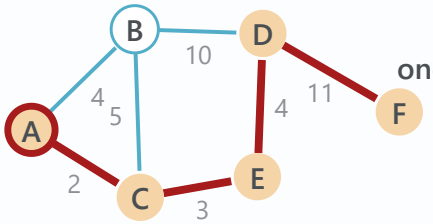
`v = C` `y = [C, E, D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y	C	E	D	F		

addFirst → hedeften kaynağa doğru, başa ekle

v = onceki[...] = A: bir adım geri.

Adım 10 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
▶   for (int v = t; v != -1; v = onceki[v])
4     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

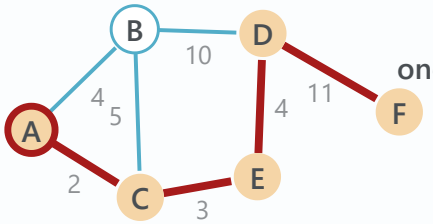
`v = A` `y = [C, E, D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D
y	A	C	E	D	F	

addFirst → hedeften kaynağa doğru, başa ekle

A listenin **başına** eklenir (sonra gelenler daha önceki düğümler).

Adım 11 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
3   for (int v = t; v != -1; v = onceki[v])
▶     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

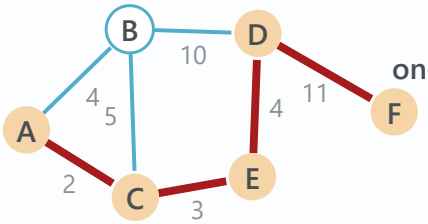
`v = A` `y = [A, C, E, D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D

y	A	C	E	D	F	

addFirst → hedeften kaynağa doğru, başa ekle

onceki[A] = −1: kaynağa varıldı, döngü biter. 5 tur = yoldaki düğüm sayısı → O(yol uzunluğu).

Adım 12 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
▶   for (int v = t; v != -1; v = onceki[v])
4     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

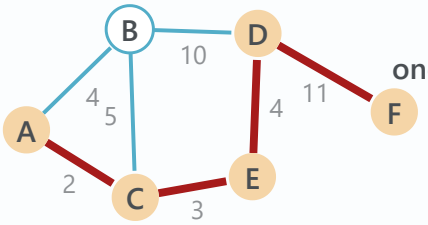
`v = –1` `y = [A, C, E, D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D

y	A	C	E	D	F	

addFirst → hedeften kaynağa doğru, başa ekle

İlk öğe kaynak A mı? Evet → t ulaşılabilir. (Ulaşılamayan t için zincir A'ya varmaz; boş liste döner.)

Adım 13 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
3   for (int v = t; v != -1; v = onceki[v])
4     y.addFirst(v);
▶   if (y.getFirst() != s) return List.of();
6   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

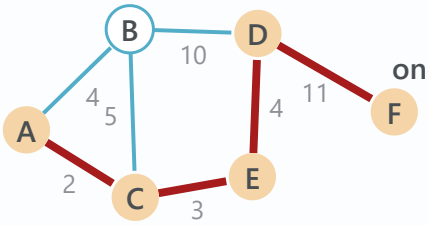
`y = [A, C, E, D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

Kod İzleme: `onceki[]` ile en kısa yolu yazdırma



v	A	B	C	D	E	F
onceki	–	A	A	E	C	D

y	A	C	E	D	F	

addFirst → hedeften kaynağa doğru, başa ekle

Sonuç **A** → **C** → **E** → **D** → **F**, uzunluk d[F] = 2 + 3 + 4 + 11 = **20**.

Adım 14 / 14

```
1 List<Integer> yol(int[] onceki, int s, int t) {
2   LinkedList<Integer> y = new LinkedList<>();
3   for (int v = t; v != -1; v = onceki[v])
4     y.addFirst(v);
5   if (y.getFirst() != s) return List.of();
▶   return y;                    // s ... t
7 }
```

DEĞİŞKENLER

`onceki = [–, A, A, E, C, D]`

`dönüş = [A, C, E, D, F]`

ÇAĞRI YIĞINI

`yol(onceki, A, F)`

`main()`

EN KISA YOL

Dijkstra Algoritmasının Analizi

Gerçekleme	Karmaşıklık	Neden?
Öncelikli kuyruk (ikili heap)	$O((n + m) \log n)$	n kez en küçüğü çıkar, en çok m kez güncelle/ekle; her biri $O(\log n)$
Kuyruksuz (dizi taraması)	$O(n^2 + m) = O(n^2)$	Her adımda en küçük mesafeli düğümü bulmak için n düğüm taranır; komşu kontrolleri toplam m

Hangisi ne zaman?

Seyrek çizgede ($m \approx n$) heap: $O(n \log n)$. Çok yoğun çizgede ($m \approx n^2$) dizi taraması $O(n^2)$ daha iyi olabilir.

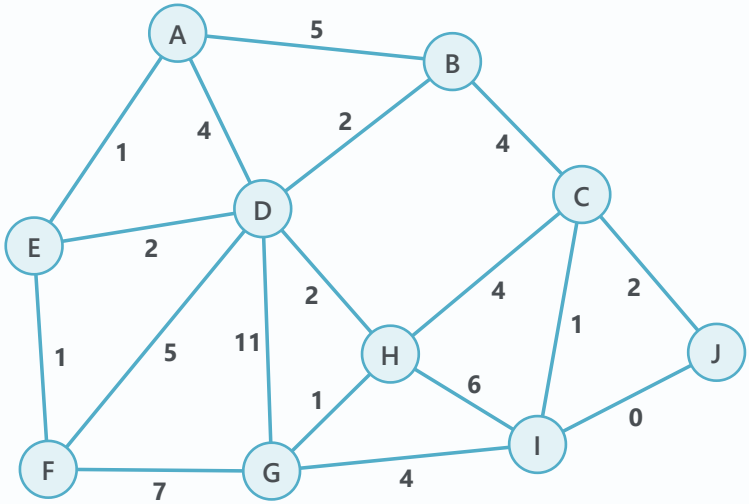
Negatif ağırlık

Kesinleşen bir mesafe sonradan negatif kenarla kısalabilir → sözde koddaki Dijkstra yanlış sonuç verir. Bayat kayıtlı Java sürümü düğümü yeniden işleyip doğruyu bulabilir ama süre garantisi kaybolur, negatif döngüde bitmez. Bu durumda Bellman-Ford kullanılır.

Neden doğru? Ağırlıklar ≥ 0 iken, kuyruktaki en küçük mesafeli düğüme başka bir düğüm üzerinden daha kısa ulaşılamaz: öbür yollar zaten en az bu kadar uzundur.

Minimum Kapsayan Ağaç (Minimum Spanning Tree)

- **Kapsayan ağaç** (spanning tree): bağlantılı, ağırlıklı bir çizgenin **tüm düğümlerini** içeren ve döngü içermeyen alt çizgesi; n düğüm için n − 1 kenar.
- **Minimum kapsayan ağaç** (MKA, MST): toplam kenar ağırlığı **en düşük** olan kapsayan ağaç.
- Uygulama: şehirleri en az kabloyla / boruyla birbirine bağlamak.
- Kruskal algoritması, "bu kenar döngü oluşturur mu?" sorusunu **Union-Find** (ayrık kümeler) ile yanıtlar: `find(x)` x'in kümesinin temsilcisini verir; `union(u, v)` farklı iki kümeyi birleştirip `true`, aynı kümedeyse `false` döndürür.



MİNİMUM KAPSAYAN AĞAÇ

Kruskal Algoritması

```
Kruskal(G):
  T = ∅          // seçilen kenarlar
  for her düğüm v: makeSet(v)
  kenarları ağırlığa göre sırala
  for her (u, v), küçükten büyüğe:
    if find(u) != find(v): // döngü yok
      T.ekle((u, v))
      union(u, v)
    if |T| == n - 1: break
  return T  // MST (bağlı değilse: orman)
```

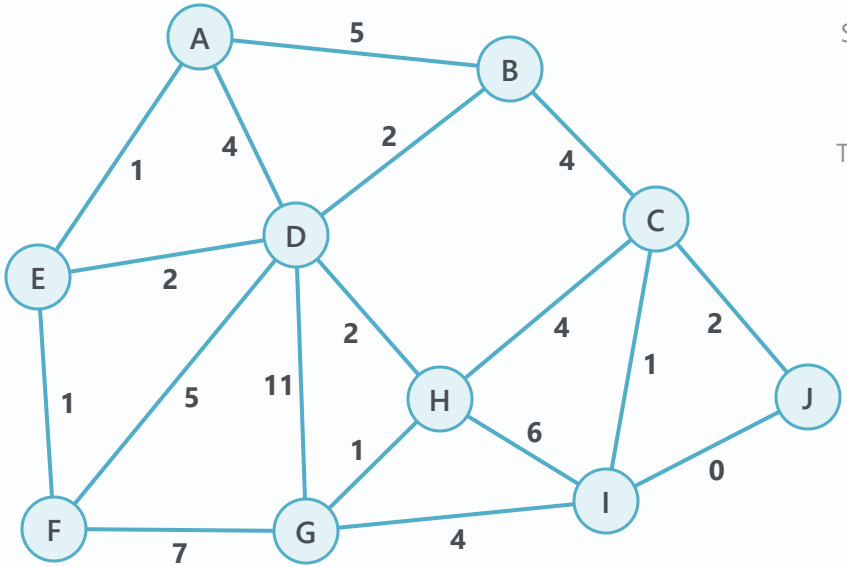
- **Açgözlü**: her zaman döngü oluşturmayan en hafif kenarı seç.
- u ve v zaten aynı kümedeyse, kenar bir **döngü** kapatır → atla.
- Başta n ayrık küme; her kabul edilen kenar iki kümeyi birleştirir.
- Çizge bağlantısızsa n−1 kenara ulaşılamaz; algoritma her bağlantılı bileşen için bir ağaçtan oluşan **minimum kapsayan orman** döndürür.

Sıralama	O(m log m)
2m find + (n−1) union	O(m · α(n))
Toplam	**O(m log n)**

MİNİMUM KAPSAYAN AĞAÇ

Kruskal Algoritması Adım Adım

```
I–J = 0
A–E = 1
C–I = 1
E–F = 1
G–H = 1
B–D = 2
C–J = 2
D–E = 2
D–H = 2
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11
```



Seçilen kenar

0 / 9

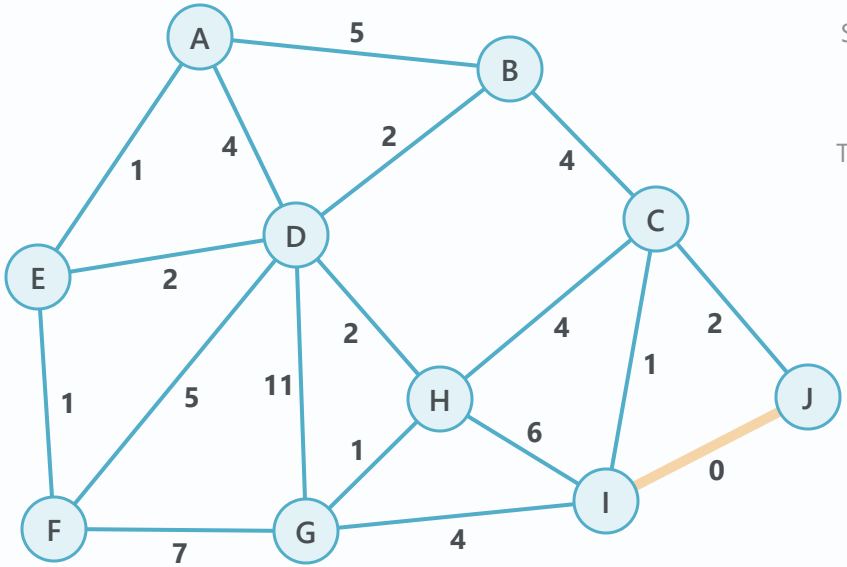
Toplam ağırlık

0

Kenarlar ağırlığa göre sıralandı (solda). Başlangıçta 10 düğüm = 10 ayrı küme, T boş.

Kruskal Algoritması Adım Adım

```
I–J = 0 ✓
A–E = 1
C–I = 1
E–F = 1
G–H = 1
B–D = 2
C–J = 2
D–E = 2
D–H = 2
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11
```



Seçilen kenar

1 / 9

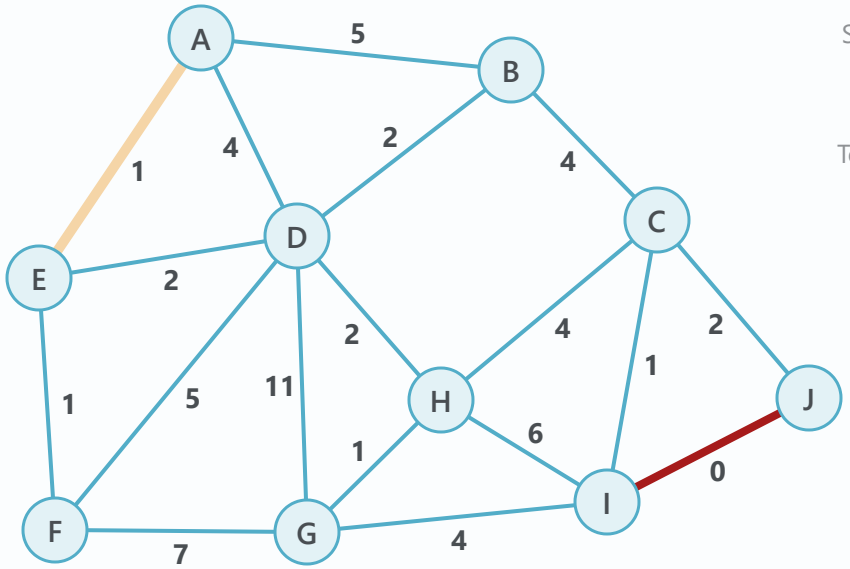
Toplam ağırlık

0

I–J (0): find(I) ≠ find(J) → kabul, kümeler birleşir.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1
E–F = 1
G–H = 1
B–D = 2
C–J = 2
D–E = 2
D–H = 2
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



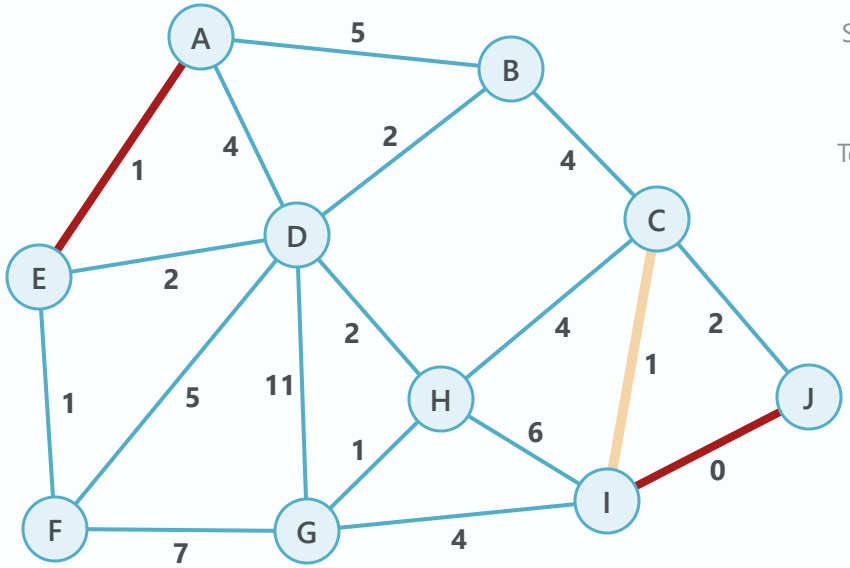
Seçilen kenar
2 / 9

Toplam ağırlık
1

A–E (1): find(A) ≠ find(E) → kabul, kümeler birleşir.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1
G–H = 1
B–D = 2
C–J = 2
D–E = 2
D–H = 2
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



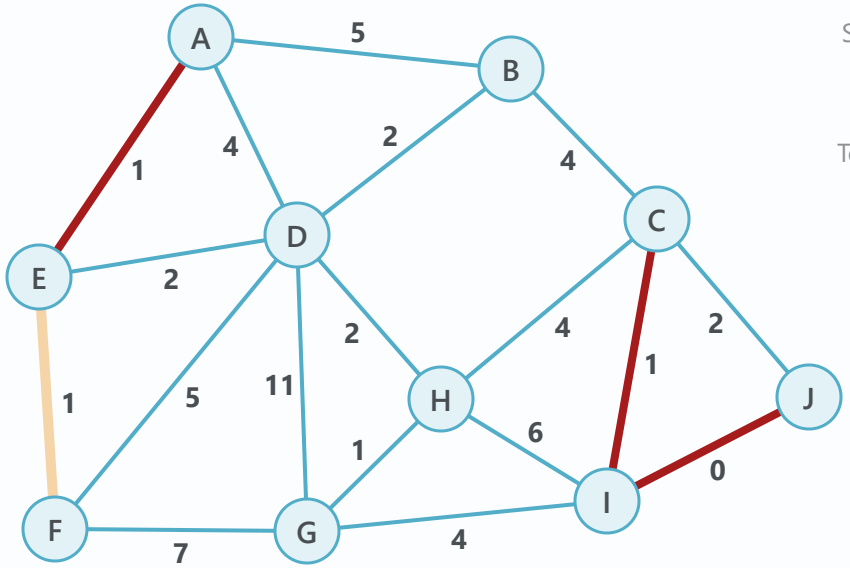
Seçilen kenar
3 / 9

Toplam ağırlık
2

C–I (1): find(C) ≠ find(I) → kabul, kümeler birleşir.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1 ✓
G–H = 1
B–D = 2
C–J = 2
D–E = 2
D–H = 2
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



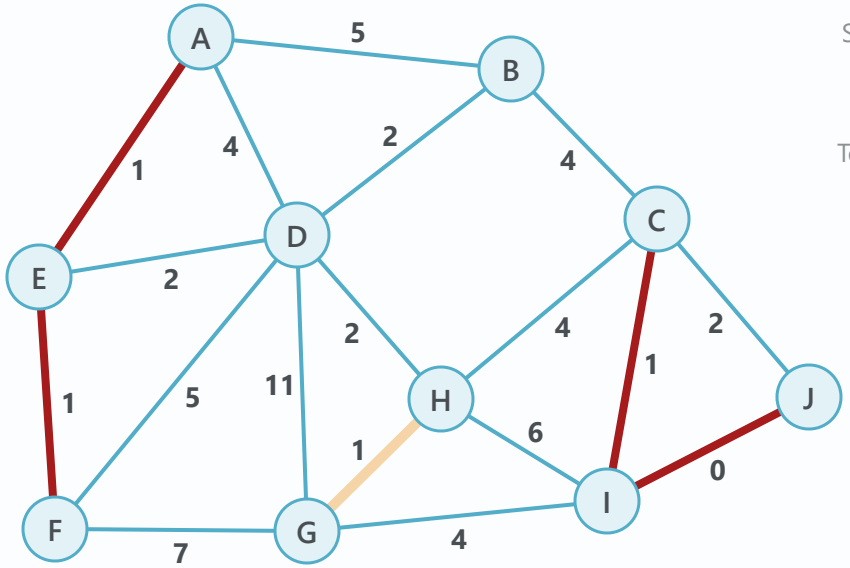
Seçilen kenar
4 / 9

Toplam ağırlık
3

E–F (1): find(E) ≠ find(F) → kabul, kümeler birleşir.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1 ✓
G–H = 1 ✓
B–D = 2
C–J = 2
D–E = 2
D–H = 2
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



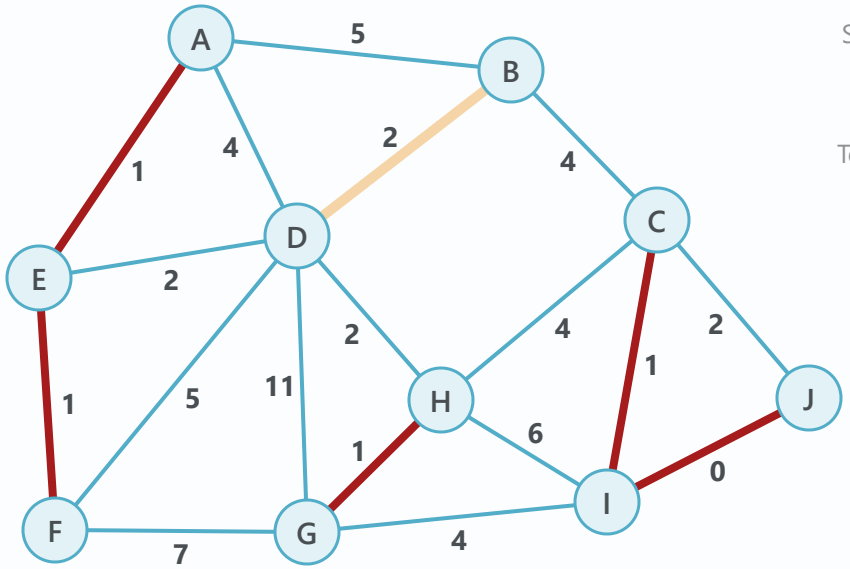
Seçilen kenar
5 / 9

Toplam ağırlık
4

G–H (1): find(G) ≠ find(H) → kabul, kümeler birleşir.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1 ✓
G–H = 1 ✓
B–D = 2 ✓
C–J = 2
D–E = 2
D–H = 2
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



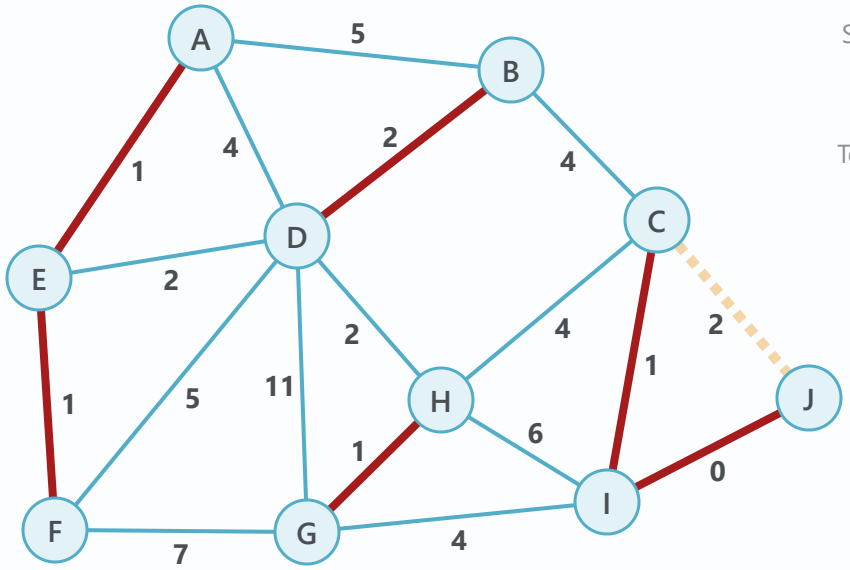
Seçilen kenar
6 / 9

Toplam ağırlık
6

B–D (2): find(B) ≠ find(D) → kabul, kümeler birleşir.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1 ✓
G–H = 1 ✓
B–D = 2 ✓
~~C–J = 2 ✗~~
D–E = 2
D–H = 2
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



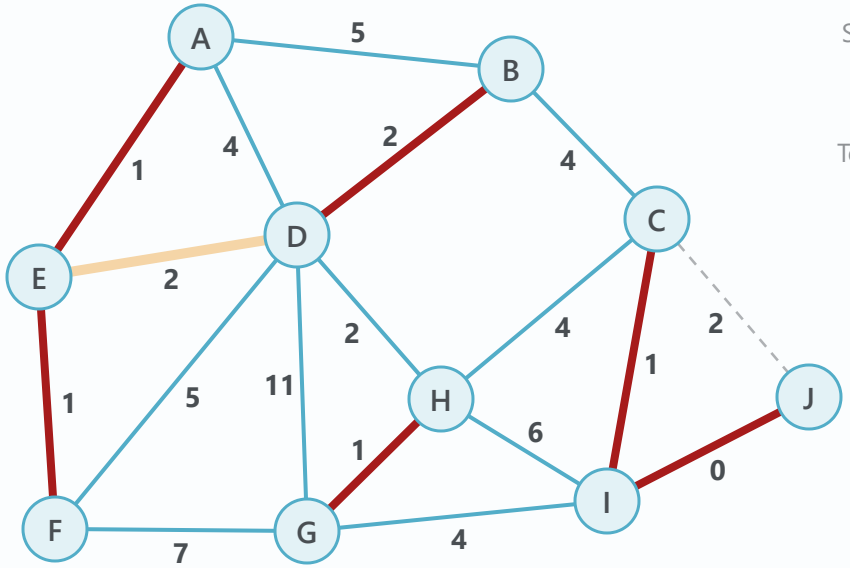
Seçilen kenar
6 / 9

Toplam ağırlık
6

C–J (2): C ve J zaten aynı kümede → döngü oluşturur, **reddedilir**.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1 ✓
G–H = 1 ✓
B–D = 2 ✓
~~C–J = 2 ✗~~
D–E = 2 ✓
D–H = 2
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



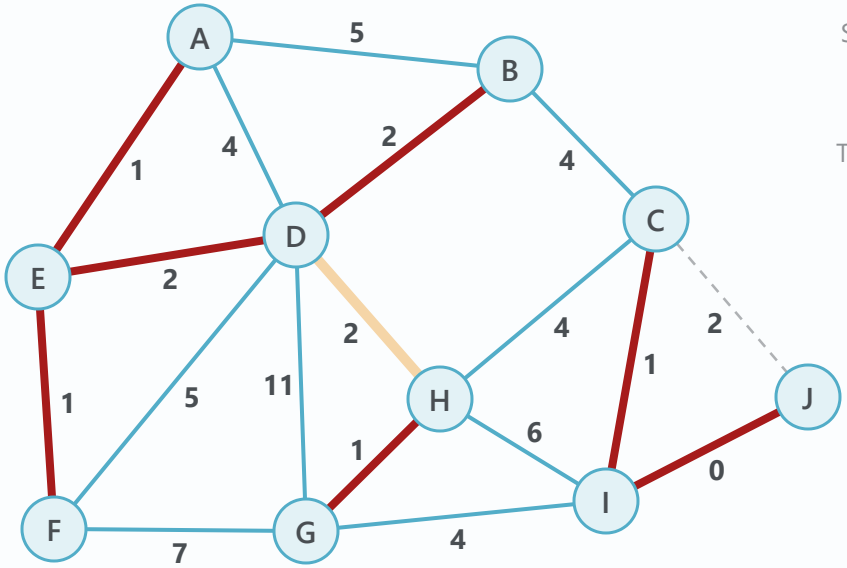
Seçilen kenar
7 / 9

Toplam ağırlık
8

D–E (2): find(D) ≠ find(E) → kabul, kümeler birleşir.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1 ✓
G–H = 1 ✓
B–D = 2 ✓
~~C–J = 2 ✗~~
D–E = 2 ✓
D–H = 2 ✓
A–D = 4
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



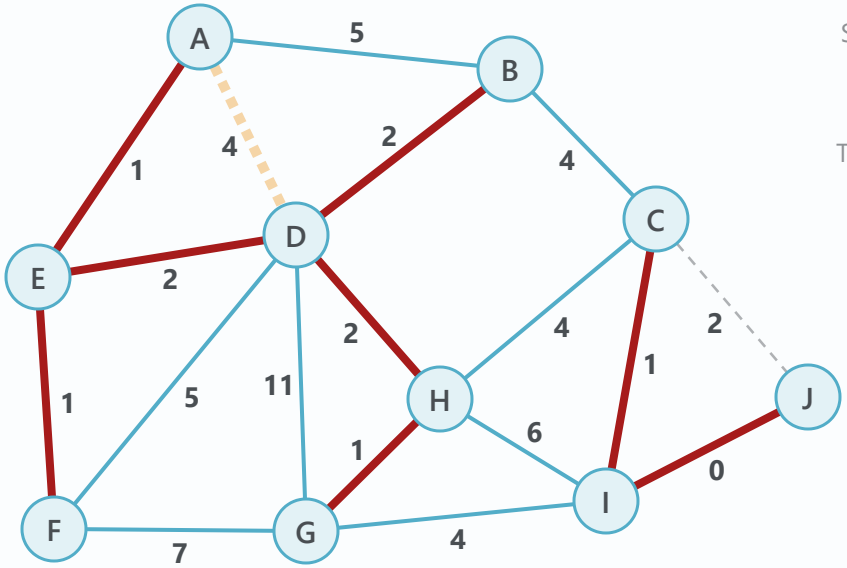
Seçilen kenar
8 / 9

Toplam ağırlık
10

D–H (2): find(D) ≠ find(H) → kabul, kümeler birleşir.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1 ✓
G–H = 1 ✓
B–D = 2 ✓
~~C–J = 2 ✗~~
D–E = 2 ✓
D–H = 2 ✓
~~A–D = 4 ✗~~
B–C = 4
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



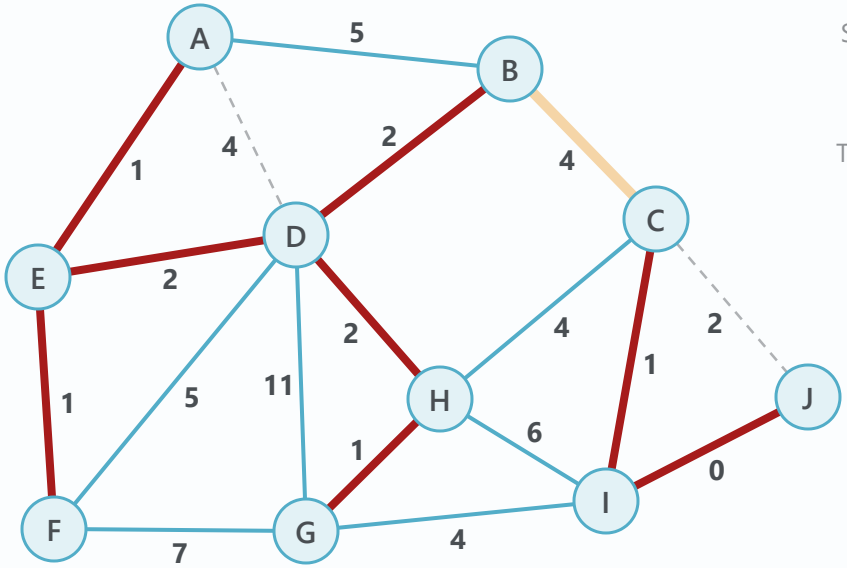
Seçilen kenar
8 / 9

Toplam ağırlık
10

A–D (4): A ve D zaten aynı kümede → döngü oluşturur, **reddedilir**.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1 ✓
G–H = 1 ✓
B–D = 2 ✓
~~C–J = 2 ✗~~
D–E = 2 ✓
D–H = 2 ✓
~~A–D = 4 ✗~~
B–C = 4 ✓
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



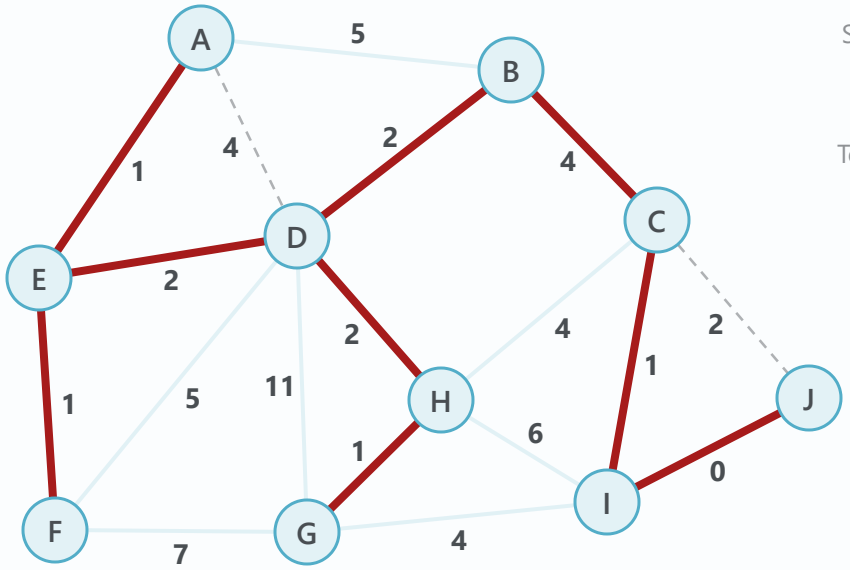
Seçilen kenar
9 / 9

Toplam ağırlık
14

B–C (4): find(B) ≠ find(C) → kabul, kümeler birleşir. 9 = n − 1 kenar seçildi: **bitti**, toplam ağırlık 14.

Kruskal Algoritması Adım Adım

I–J = 0 ✓
A–E = 1 ✓
C–I = 1 ✓
E–F = 1 ✓
G–H = 1 ✓
B–D = 2 ✓
~~C–J = 2 ✗~~
D–E = 2 ✓
D–H = 2 ✓
~~A–D = 4 ✗~~
B–C = 4 ✓
C–H = 4
G–I = 4
A–B = 5
D–F = 5
H–I = 6
F–G = 7
D–G = 11



Seçilen kenar
9 / 9

Toplam ağırlık
14

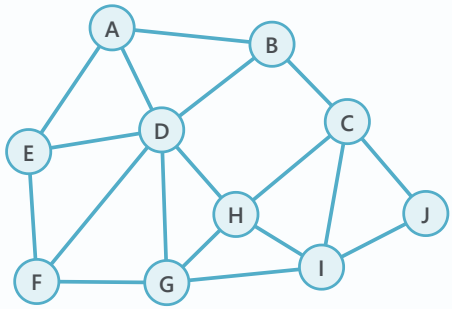
Minimum kapsayan ağaç: A–E, E–F, D–E, B–D, D–H, G–H, B–C, C–I, I–J. Kalan kenarlara bakmaya gerek yok.

Kod İzleme: `kruskal()` + Union-Find

```
1  List<int[]> kruskal(int n, List<int[]> E) {
▶    E.sort((a, b) ->          // {u, v, w}
▶        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
7      if (uf.union(e[0], e[1]))  // döngüsüz
8        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`n = 10`



kenar (sıralı)

```
I–J 0
A–E 1
C–I 1
E–F 1
G–H 1
B–D 2
C–J 2
D–E 2
D–H 2
A–D 4
B–C 4
C–H 4
G–I 4
A–B 5
D–F 5
H–I 6
F–G 7
D–G 11
```

|T|=0, ağırlık=0

uf ormanı (ok → ebeveyn)



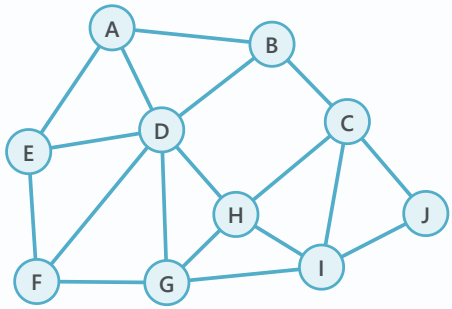
Kenarlar ağırlığa göre sıralanır (A=0 … J=9; `sort` kararlıdır, eşit ağırlıklar girdi sırasında kalır). O(m log m).

Kod İzleme: `kruskal()` + Union-Find

```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->            // {u, v, w}
3        Integer.compare(a[2], b[2]));
   ▶ UnionFind uf = new UnionFind(n);
   ▶ List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
7      if (uf.union(e[0], e[1]))  // döngüsüz
8        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`n = 10`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

```
I–J 0
A–E 1
C–I 1
E–F 1
G–H 1
B–D 2
C–J 2
D–E 2
D–H 2
A–D 4
B–C 4
C–H 4
G–I 4
A–B 5
D–F 5
H–I 6
F–G 7
D–G 11
```

|T|=0, ağırlık=0

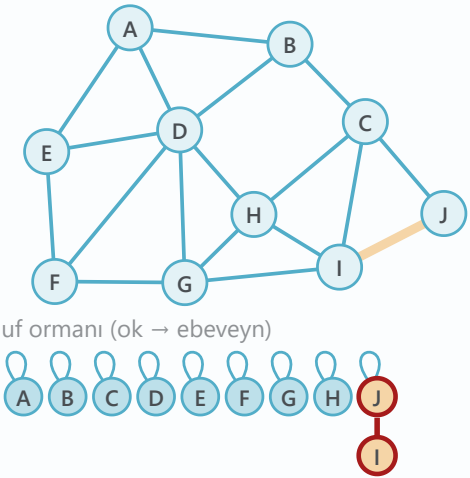
`new UnionFind(10)` : her düğüm kendi kümesi (10 kök). T boş.

Kod İzleme: `kruskal()` + Union-Find

```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->          // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
▶        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {I,J,0}` `|T| = 1`



kenar (sıralı)

- **I–J 0 ✓**
- A–E 1
- C–I 1
- E–F 1
- G–H 1
- B–D 2
- C–J 2
- D–E 2
- D–H 2
- A–D 4
- B–C 4
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=1, ağırlık=0

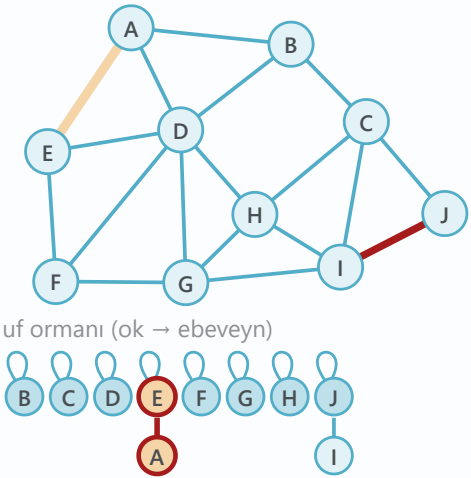
I–J (0): kökler I ≠ J → `union` true: ağaçlar birleşir (I → J), kenar T'ye eklenir.

Kod İzleme: `kruskal()` + Union-Find

```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->            // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
▶        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {A,E,1}` `|T| = 2`



kenar (sıralı)
I–J 0 ✓
A–E 1 ✓
C–I 1
E–F 1
G–H 1
B–D 2
C–J 2
D–E 2
D–H 2
A–D 4
B–C 4
C–H 4
G–I 4
A–B 5
D–F 5
H–I 6
F–G 7
D–G 11
|T|=2, ağırlık=1

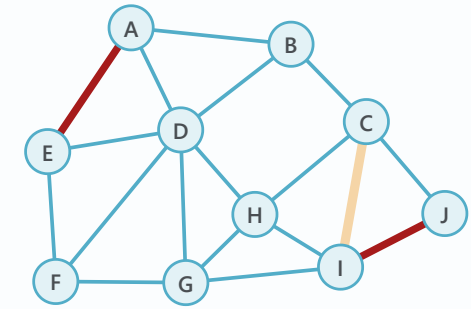
A–E (1): kökler A ≠ E → `union` true: ağaçlar birleşir (A → E), kenar T'ye eklenir.

Kod İzleme: `kruskal()` + Union-Find

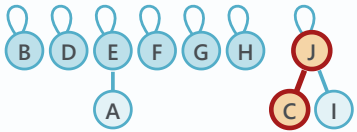
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->            // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
▶        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {C,I,1}` `|T| = 3`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

I–J 0 ✓
A–E 1 ✓
C–I 1 ✓
E–F 1
G–H 1
B–D 2
C–J 2
D–E 2
D–H 2
A–D 4
B–C 4
C–H 4
G–I 4
A–B 5
D–F 5
H–I 6
F–G 7
D–G 11

|T|=3, ağırlık=2

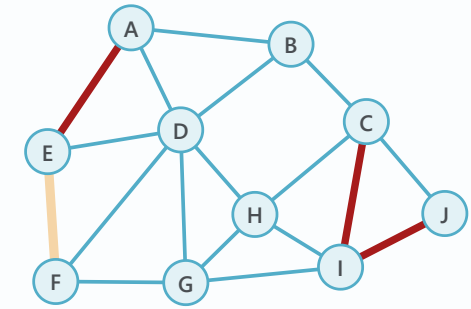
C–I (1): kökler C ≠ J → `union` true: ağaçlar birleşir (C → J), kenar T'ye eklenir.

Kod İzleme: `kruskal()` + Union-Find

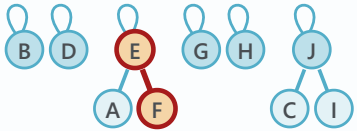
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->            // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
▶        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {E,F,1}` `|T| = 4`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

- **I–J 0** ✓
- **A–E 1** ✓
- **C–I 1** ✓
- **E–F 1** ✓
- G–H 1
- B–D 2
- C–J 2
- D–E 2
- D–H 2
- A–D 4
- B–C 4
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=4, ağırlık=3

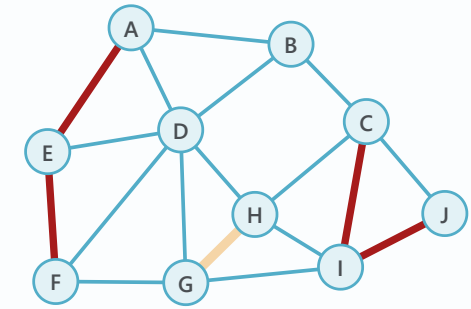
E–F (1): kökler E ≠ F → `union` true: ağaçlar birleşir (F → E), kenar T'ye eklenir.

Kod İzleme: `kruskal()` + Union-Find

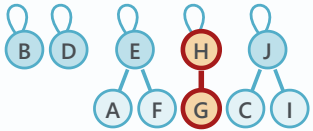
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->            // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
▶        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {G,H,1}` `|T| = 5`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

- **I–J 0 ✓**
- **A–E 1 ✓**
- **C–I 1 ✓**
- **E–F 1 ✓**
- **G–H 1 ✓**
- B–D 2
- C–J 2
- D–E 2
- D–H 2
- A–D 4
- B–C 4
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=5, ağırlık=4

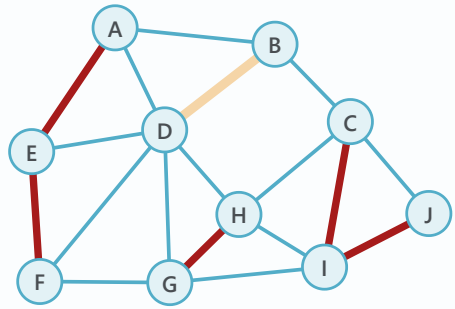
G–H (1): kökler G ≠ H → `union` true: ağaçlar birleşir (G → H), kenar T'ye eklenir.

Kod İzleme: `kruskal()` + Union-Find

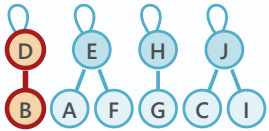
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->            // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
▶        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {B,D,2}` `|T| = 6`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

- **I–J 0** ✓
- **A–E 1** ✓
- **C–I 1** ✓
- **E–F 1** ✓
- **G–H 1** ✓
- **B–D 2** ✓
- C–J 2
- D–E 2
- D–H 2
- A–D 4
- B–C 4
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=6, ağırlık=6

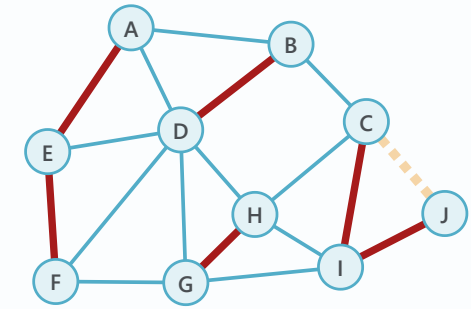
B–D (2): kökler B ≠ D → `union` true: ağaçlar birleşir (B → D), kenar T'ye eklenir.

Kod İzleme: `kruskal()` + Union-Find

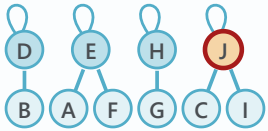
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->            // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
8        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {C,J,2}` `|T| = 6`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

- I–J 0 ✓
- A–E 1 ✓
- C–I 1 ✓
- E–F 1 ✓
- G–H 1 ✓
- B–D 2 ✓
- ~~C–J 2 ✗~~
- D–E 2
- D–H 2
- A–D 4
- B–C 4
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=6, ağırlık=6

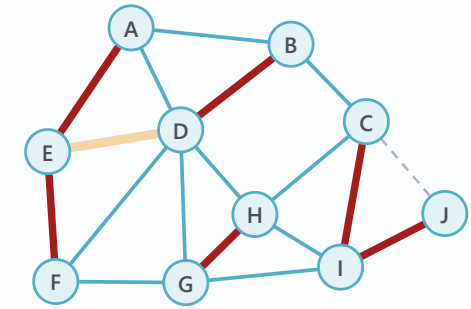
C–J (2): find(C) = find(J) = J → aynı ağaç, `union` false döner: kenar **döngü kurar, atlanır**.

Kod İzleme: `kruskal()` + Union-Find

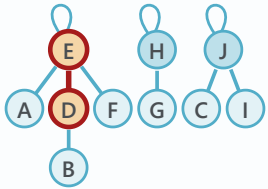
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->            // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
▶        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {D,E,2}` `|T| = 7`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

- I–J 0 ✓
- A–E 1 ✓
- C–I 1 ✓
- E–F 1 ✓
- G–H 1 ✓
- B–D 2 ✓
- ~~C–J 2 ✗~~
- D–E 2 ✓
- D–H 2
- A–D 4
- B–C 4
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=7, ağırlık=8

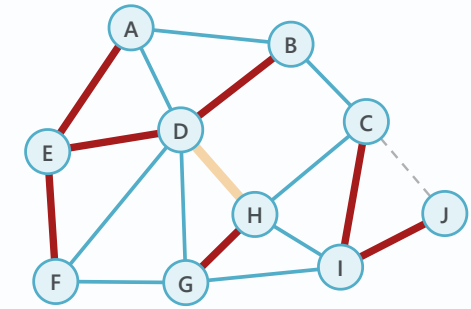
D–E (2): kökler D ≠ E → `union` true: ağaçlar birleşir (D → E), kenar T'ye eklenir.

Kod İzleme: `kruskal()` + Union-Find

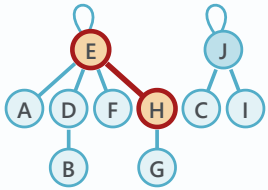
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->          // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
▶        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {D,H,2}` `|T| = 8`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

- I–J 0 ✓
- A–E 1 ✓
- C–I 1 ✓
- E–F 1 ✓
- G–H 1 ✓
- B–D 2 ✓
- ~~C–J 2 ✗~~
- D–E 2 ✓
- D–H 2 ✓
- A–D 4
- B–C 4
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=8, ağırlık=10

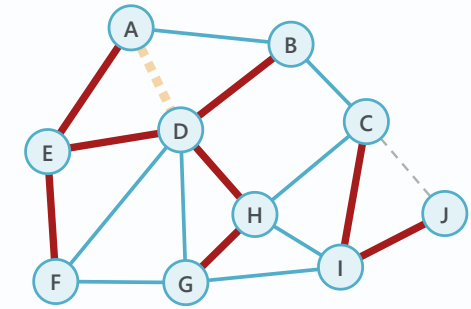
D–H (2): kökler E ≠ H → `union` true: ağaçlar birleşir (H → E), kenar T'ye eklenir.

Kod İzleme: `kruskal()` + Union-Find

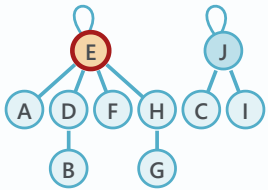
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->          // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
8        T.add(e);
9      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {A,D,4}` `|T| = 8`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

- I–J 0 ✓
- A–E 1 ✓
- C–I 1 ✓
- E–F 1 ✓
- G–H 1 ✓
- B–D 2 ✓
- ~~C–J 2 ✗~~
- D–E 2 ✓
- D–H 2 ✓
- ~~A–D 4 ✗~~
- B–C 4
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=8, ağırlık=10

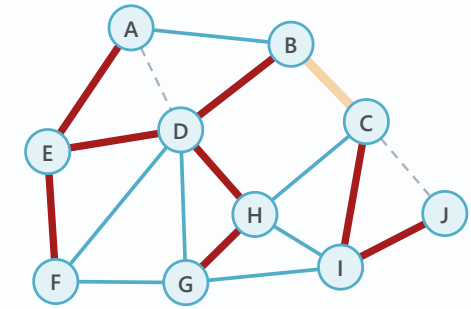
A–D (4): find(A) = find(D) = E → aynı ağaç, `union` false döner: kenar **döngü kurar, atlanır**.

Kod İzleme: `kruskal()` + Union-Find

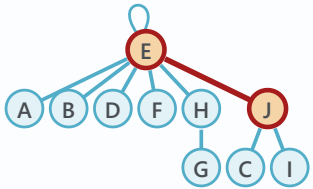
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->            // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
▶      if (uf.union(e[0], e[1]))  // döngüsüz
▶        T.add(e);
▶      if (T.size() == n - 1) break;
10   }
11   return T;
12 }
```

DEĞİŞKENLER

`e = {B,C,4}` `|T| = 9`



uf ormanı (ok → ebeveyn)



kenar (sıralı)

- I–J 0 ✓
- A–E 1 ✓
- C–I 1 ✓
- E–F 1 ✓
- G–H 1 ✓
- B–D 2 ✓
- ~~C–J 2 ✗~~
- D–E 2 ✓
- D–H 2 ✓
- ~~A–D 4 ✗~~
- **B–C 4 ✓**
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=9, ağırlık=14

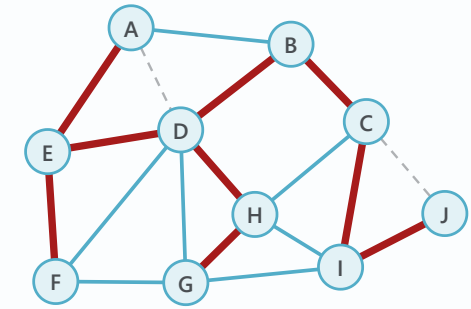
B–C (4): kökler E ≠ J → `union` true: ağaçlar birleşir (J → E), kenar T'ye eklenir. |T| = 9 = n − 1 → **break**.

Kod İzleme: `kruskal()` + Union-Find

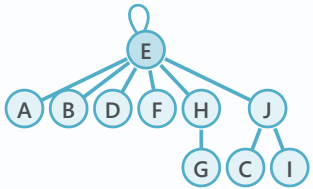
```
1  List<int[]> kruskal(int n, List<int[]> E) {
2    E.sort((a, b) ->          // {u, v, w}
3        Integer.compare(a[2], b[2]));
4    UnionFind uf = new UnionFind(n);
5    List<int[]> T = new ArrayList<>();
6    for (int[] e : E) {
7      if (uf.union(e[0], e[1]))  // döngüsüz
8        T.add(e);
9      if (T.size() == n - 1) break;
10   }
▶    return T;
12 }
```

DEĞİŞKENLER

`|T|` = **9**



uf ormanı (ok → ebeveyn)



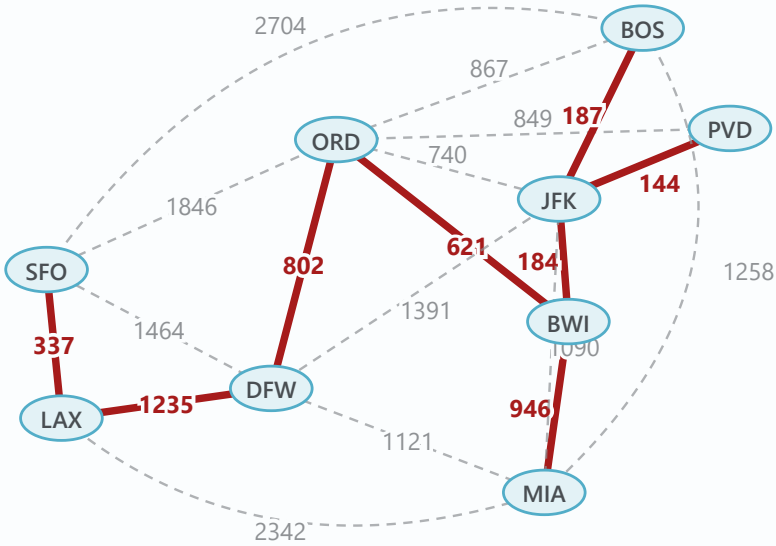
kenar (sıralı)

- **I–J 0** ✓
- **A–E 1** ✓
- **C–I 1** ✓
- **E–F 1** ✓
- **G–H 1** ✓
- **B–D 2** ✓
- ~~C–J 2 ✗~~
- **D–E 2** ✓
- **D–H 2** ✓
- ~~A–D 4 ✗~~
- **B–C 4** ✓
- C–H 4
- G–I 4
- A–B 5
- D–F 5
- H–I 6
- F–G 7
- D–G 11

|T|=9, ağırlık=14

`return T` : 9 kenar, toplam ağırlık **14**. Tek ağaç kaldı. Kalan 7 kenara hiç bakılmadı.

İkinci Örnek: Havalimanları



Goodrich'teki örnek: 9 havalimanı, 19 uçuş hattı (mil). Kruskal sırası:

144, 184, 187, 337, 621	kabul
740 ORD–JFK	ret (döngü)
802	kabul
849, 867	ret
946	kabul
1090, 1121	ret
1235 LAX–DFW	kabul → 8 kenar

Toplam: **4456** mil.

Prim Algoritması

```
Prim(G, s):
  ağaç = {s}, T = ∅
  s'nin kenarlarını kuyruğa ekle
  while |T| < n − 1:
    (u, v) = en hafif kenar  // u ağaçta
    if v ağaçta: atla        // döngü
    v'yi ağaca, (u, v)'yi T'ye ekle
    v'nin dışarı giden kenarlarını
      kuyruğa ekle
  return T
```

- Ağaç **tek bir düğümden** başlar ve her adımda bir düğüm büyür.
- Her adımda, ağaçtaki bir düğümü ağaç dışındaki bir düğüme bağlayan kenarların **en hafifi** seçilir.
- Ağacı dışarıya bağlayan kenarlardan en hafifi her zaman bir minimum kapsayan ağaçta bulunur: ağaç dışarıya bir yolla bağlanmak zorundadır ve bu kenar en ucuz bağlantıdır.
- Dijkstra'ya benzer; fark kuyruğun anahtarıdır: kaynağa uzaklık değil, **tek kenarın ağırlığı**.
- Kruskal'dan farkı: ara durumda hep **tek bir ağaç** vardır, Union-Find gerekmez.

Prim: Java (öncelikli kuyrukla, tembel sürüm)

```java
List<int[]> prim(List<List<int[]>> adj, int s) {
  int n = adj.size();                  // adj.get(u): {v, w}
  boolean[] agacta = new boolean[n];
  PriorityQueue<int[]> pq = new PriorityQueue<>(
      (a, b) -> Integer.compare(a[2], b[2]));  // {u, v, w}
  List<int[]> T = new ArrayList<>();
  agacta[s] = true;
  for (int[] e : adj.get(s)) pq.add(new int[]{s, e[0], e[1]});
  while (!pq.isEmpty() && T.size() < n - 1) {
    int[] e = pq.poll();                // en hafif aday
    if (agacta[e[1]]) continue;         // bayat: döngü kurar
    agacta[e[1]] = true;
    T.add(e);
    for (int[] f : adj.get(e[1]))
      if (!agacta[f[0]]) pq.add(new int[]{e[1], f[0], f[1]});
  }
  return T;
}
```

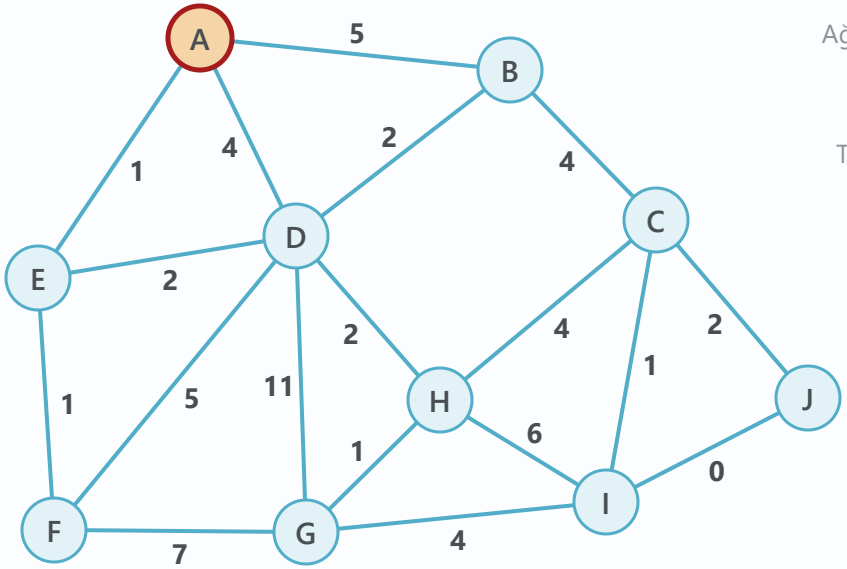
- Kuyrukta {u, v, w} kayıtları: u ağaçta, v aday.
- **Tembel** sürüm: v ağaca girince v'ye giden eski kayıtlar silinmez; kuyruktan çıktıklarında `continue` ile atlanır (bayat kayıt).
- Her kenar kuyruğa en çok bir kez girer (ağaca ilk giren ucundan; öbür ucu girdiğinde karşı uç zaten ağaçtadır): en çok m ekleme ve çıkarma, her biri O(log m).
- Toplam **O(m log m) = O(m log n)**; ek bellek O(m).

MİNİMUM KAPSAYAN AĞAÇ

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

A–E = 1
A–D = 4
A–B = 5



Ağaçtaki düğüm
1 / 10

Toplam ağırlık
0

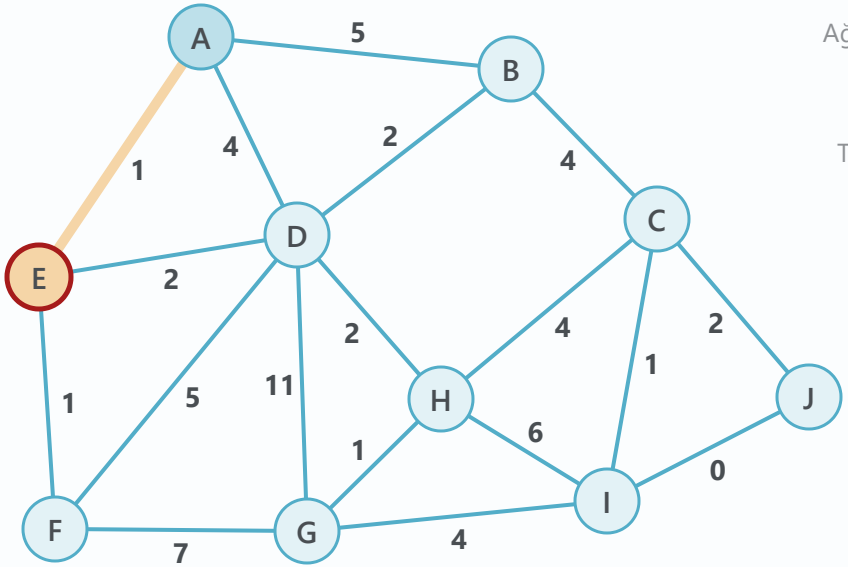
Başlangıç: ağaçta yalnız **A**. A'nın kenarları kuyruğa girer: A–E 1, A–D 4, A–B 5. Eşit ağırlıklarda alfabetik sıra izlenir.

Adım 1 / 11

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

```
E–F = 1
E–D = 2
A–D = 4
A–B = 5
```



Ağaçtaki düğüm

2 / 10

Toplam ağırlık

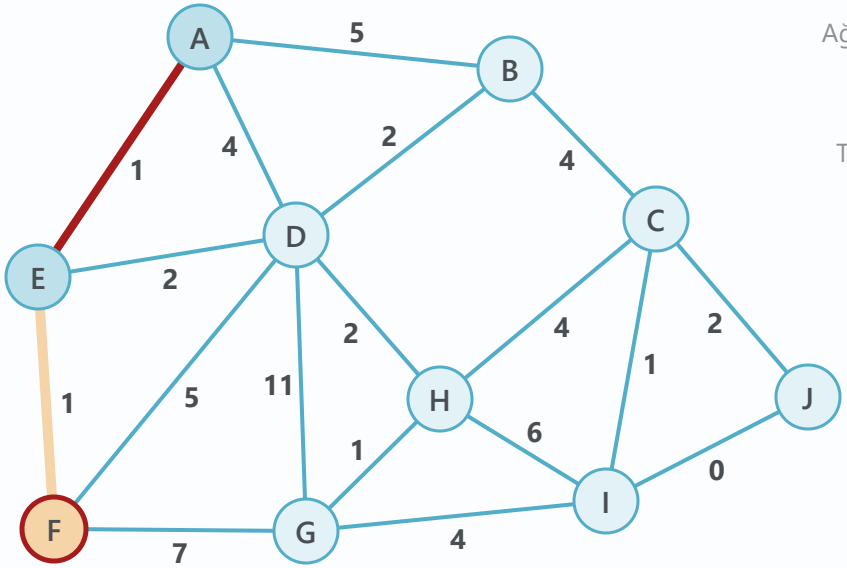
1

A–E (1): ağaçla dışarıyı bağlayan en hafif kenar → **E** ağaca girer. Yeni adaylar: E–F, E–D.

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

- E–D = 2
- A–D = 4
- A–B = 5
- F–D = 5
- F–G = 7



Ağaçtaki düğüm

3 / 10

Toplam ağırlık

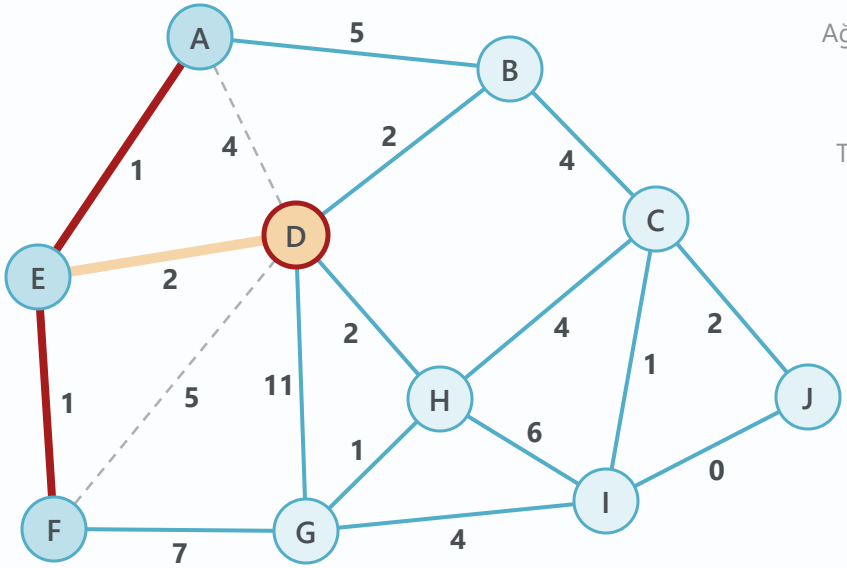
2

E–F (1): ağaçla dışarıyı bağlayan en hafif kenar → **F** ağaca girer. Yeni adaylar: F–D, F–G.

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

- D–B = 2
- D–H = 2
- ~~A–D = 4~~
- A–B = 5
- ~~F–D = 5~~
- F–G = 7
- D–G = 11



Ağaçtaki düğüm

4 / 10

Toplam ağırlık

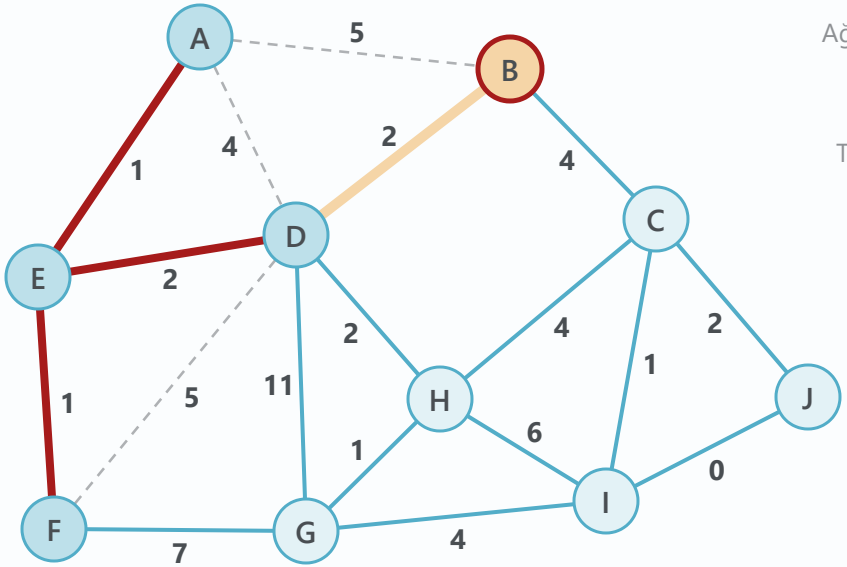
4

E–D (2): ağaçla dışarıyı bağlayan en hafif kenar → **D** ağaca girer. Yeni adaylar: D–B, D–H, D–G. Üstü çizili kayıtlar bayatladı (iki ucu da ağaçta); kuyruktan çıkınca atlanır.

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

- D–H = 2
- B–C = 4
- ~~A–D = 4~~
- ~~A–B = 5~~
- ~~F–D = 5~~
- F–G = 7
- D–G = 11



Ağaçtaki düğüm

5 / 10

Toplam ağırlık

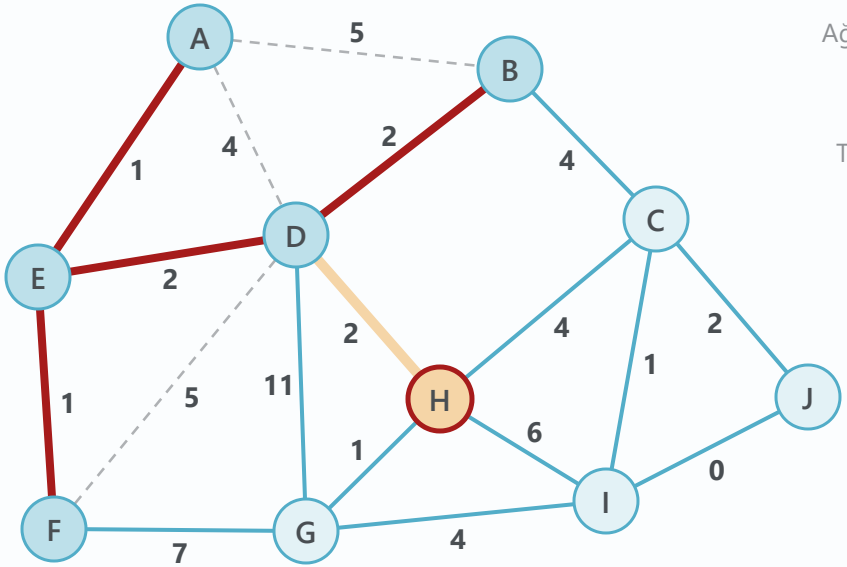
6

D–B (2): ağaçla dışarıyı bağlayan en hafif kenar → **B** ağaca girer. Yeni adaylar: B–C.

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

- H–G = 1
- B–C = 4
- H–C = 4
- ~~A–D = 4~~
- ~~A–B = 5~~
- ~~F–D = 5~~
- H–I = 6
- F–G = 7
- D–G = 11



Ağaçtaki düğüm

6 / 10

Toplam ağırlık

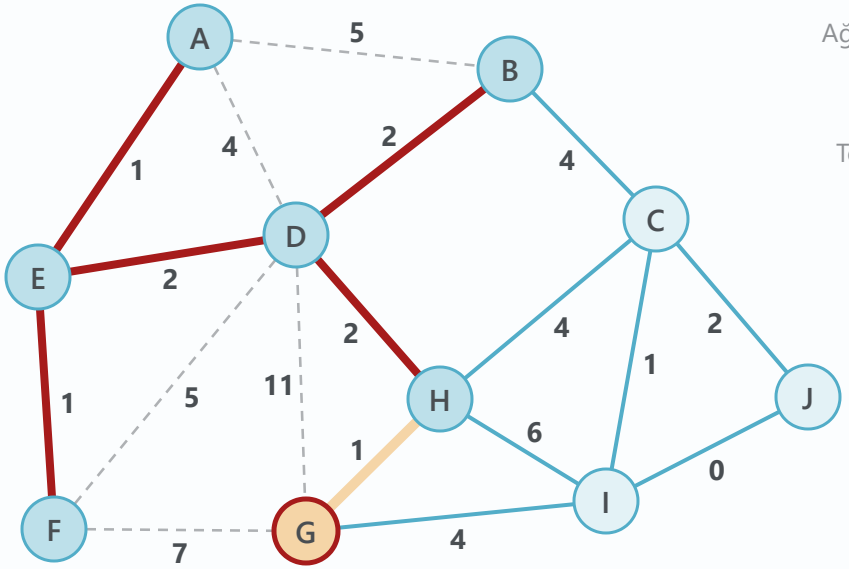
8

D–H (2): ağaçla dışarıyı bağlayan en hafif kenar → **H** ağaca girer. Yeni adaylar: H–G, H–C, H–I.

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

- B–C = 4
- H–C = 4
- ~~A–D = 4~~
- G–I = 4
- ~~A–B = 5~~
- ~~F–D = 5~~
- H–I = 6
- ~~F–G = 7~~
- ~~D–G = 11~~



Ağaçtaki düğüm

7 / 10

Toplam ağırlık

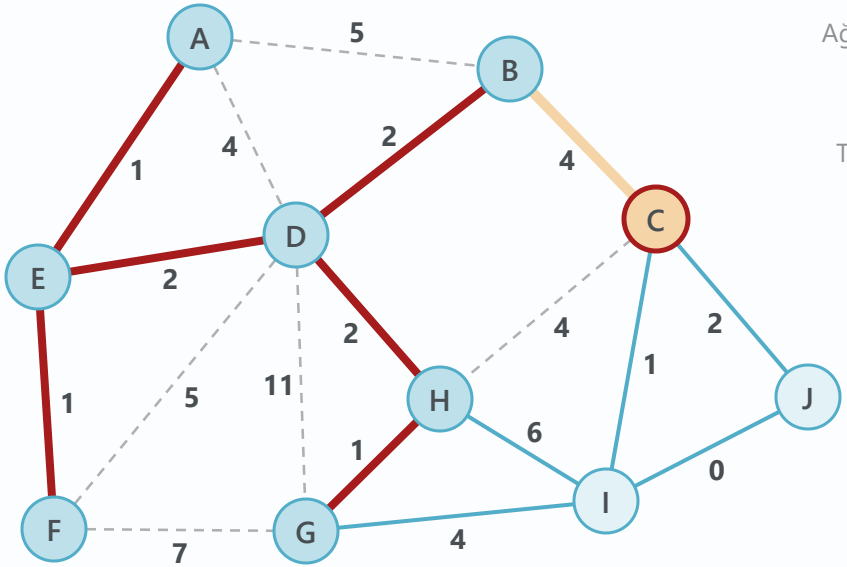
9

H–G (1): ağaçla dışarıyı bağlayan en hafif kenar → **G** ağaca girer. Yeni adaylar: G–I. Üstü çizili kayıtlar bayatladı (iki ucu da ağaçta); kuyruktan çıkınca atlanır.

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

- C–I = 1
- C–J = 2
- ~~H–C = 4~~
- ~~A–D = 4~~
- G–I = 4
- ~~A–B = 5~~
- ~~F–D = 5~~
- H–I = 6
- ~~F–G = 7~~
- ~~D–G = 11~~



Ağaçtaki düğüm

8 / 10

Toplam ağırlık

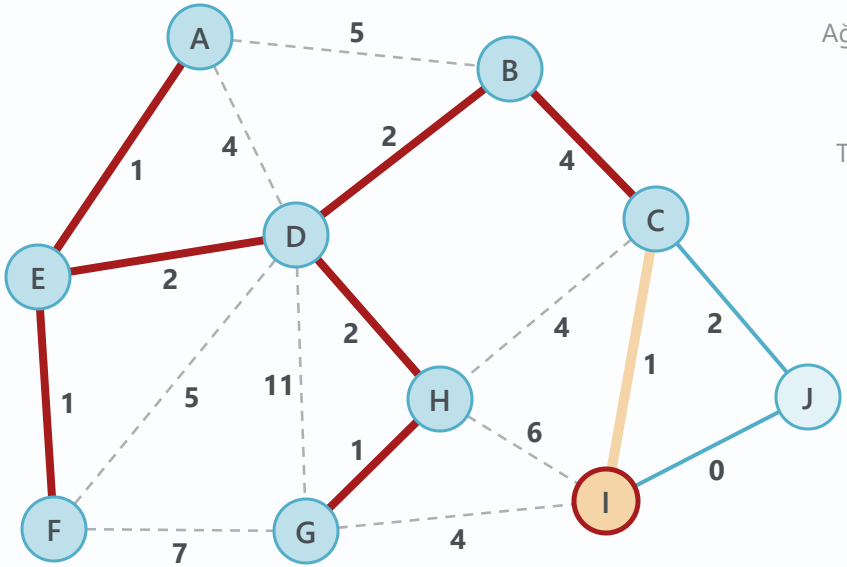
13

B–C (4): ağaçla dışarıyı bağlayan en hafif kenar → **C** ağaca girer. Yeni adaylar: C–I, C–J.

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

- I–J = 0
- C–J = 2
- ~~H–C = 4~~
- ~~A–D = 4~~
- ~~G–I = 4~~
- ~~A–B = 5~~
- ~~F–D = 5~~
- ~~H–I = 6~~
- ~~F–G = 7~~
- ~~D–G = 11~~



Ağaçtaki düğüm

9 / 10

Toplam ağırlık

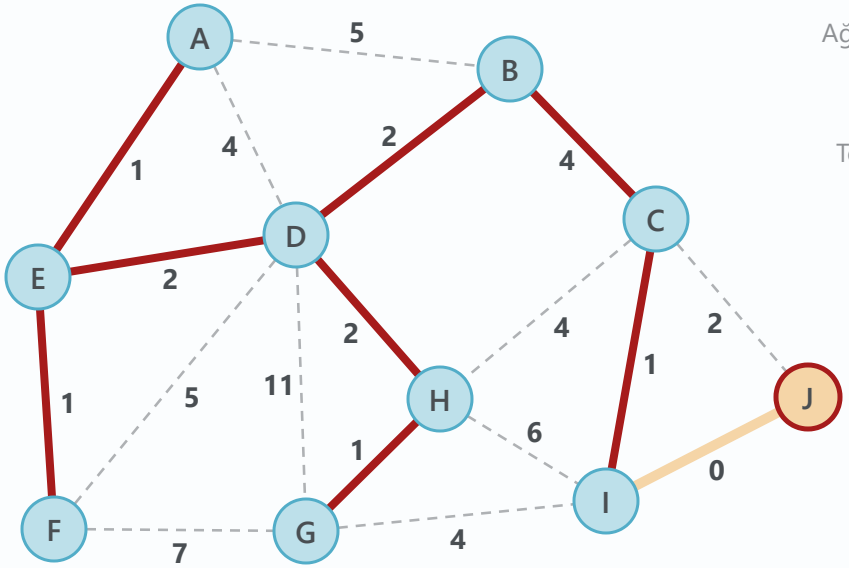
14

C–I (1): ağaçla dışarıyı bağlayan en hafif kenar → **I** ağaca girer. Yeni adaylar: I–J.

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

~~C–J = 2~~
~~H–C = 4~~
~~A–D = 4~~
~~G–I = 4~~
~~A–B = 5~~
~~F–D = 5~~
~~H–I = 6~~
~~F–G = 7~~
~~D–G = 11~~



Ağaçtaki düğüm
10 / 10

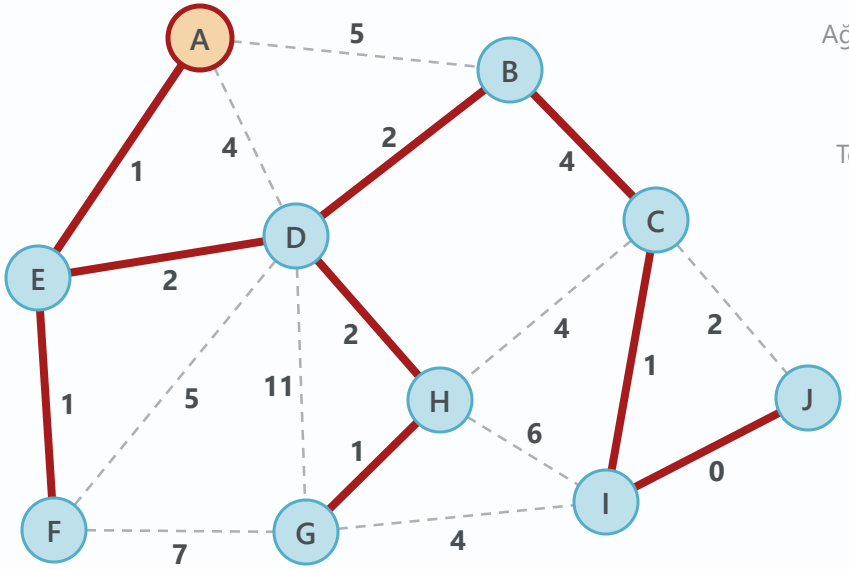
Toplam ağırlık
14

I–J (0): ağaçla dışarıyı bağlayan en hafif kenar → **J** ağaca girer. 9 = n − 1 kenar: **bitti**, toplam ağırlık **14**.

Prim Algoritması Adım Adım (A'dan)

öncelikli kuyruk

~~C–J = 2~~
~~H–C = 4~~
~~A–D = 4~~
~~G–I = 4~~
~~A–B = 5~~
~~F–D = 5~~
~~H–I = 6~~
~~F–G = 7~~
~~D–G = 11~~



Ağaçtaki düğüm
10 / 10

Toplam ağırlık
14

Prim ağacı: A–E, E–F, E–D, D–B, D–H, H–G, B–C, C–I, I–J. Kruskal ile **aynı toplam ağırlık (14)**. Eşit ağırlıklar başka sırayla çıkarsa (ör. B–C yerine H–C) farklı ama yine 14 ağırlıklı bir ağaç bulunabilir.

Kruskal ve Prim Karşılaştırması

	Kruskal	Prim
Fikir	kenarları küçükten büyüğe dener, ağaçları birleştirir	tek ağacı bir düğümden başlayıp büyütür
Ara durum	orman (birden çok ağaç)	tek ağaç
Veri yapısı	sıralı kenar listesi + Union-Find	öncelikli kuyruk + işaret dizisi, komşuluk listesi
Döngü kontrolü	find(u) == find(v)	v zaten ağaçta mı?
Maliyet	O(m log n) (sıralama baskın)	O(m log n) (heap ile); matrisle $O(n^2)$
Ne zaman?	seyrek çizge, kenarlar liste olarak hazır	komşuluk listesi hazır; çok yoğun çizgede $O(n^2)$ sürüm

İkisi de açgözlüdür ve aynı çizgede aynı **toplam ağırlığı** bulur (örnekte 14). Eşit ağırlıklı kenarlar varsa buldukları ağaçlar farklı olabilir.

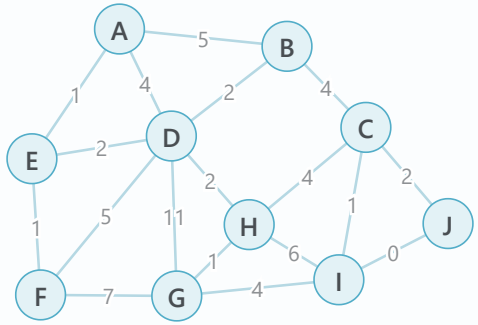
Laboratuvar: Minimum Kapsayan Ağaç

Kruskal: 10 düğümlü örnek çizge

Girdi (kenarlar: A-B:4, ...) `A-B:5, A-D:4, A-E:1, B-C:4, B-D:2, C-H:4, C-I:1, C-J:2, D-E:2, D-F:5, D-`

KOD · **adım 1 / 47**

```
1 List<int[]> kruskal(int n, List<int[]> …
2   E.sort((a, b) ->           // {u, v, w}
3       Integer.compare(a[2], b[2]));
4   UnionFind uf = new UnionFind(n);
5   List<int[]> T = new ArrayList<>();
6   for (int[] e : E) {
7     if (uf.union(e[0], e[1]))  // döngü…
8       T.add(e);
9     if (T.size() == n - 1) break;
10  }
11  return T;
12 }
```



E sıralı boş

`T.size() = 0` `ağırlık = 0`

DÖNÜŞ DEĞERİ –

kruskal(n = 10, E) çağrılır; çizgede 10 düğüm ve 18 kenar var.

ÖZET

Özet

Kavramlar
Düğüm, kenar, derece, yön, ağırlık, yol, döngü ve bağlantılı bileşen.

Gösterimler
Seyrek çizgede komşuluk listesi; hızlı kenar sorgusunda matris.

Algoritma	Yapı	Zaman (liste ile)	Ne verir?
BFS	kuyruk	O(n + m)	düzey sırası, ağırlıksız en kısa yol
DFS	yığın / özyineleme	O(n + m)	derinlik öncelikli gezinme, döngü, bileşenler
Topolojik	kuyruk / DFS	O(n + m)	DAG için bağımlılık sırası
Dijkstra	öncelikli kuyruk	O((n + m) log n)	ağırlıklı en kısa yol (ağırlık ≥ 0)
Kruskal	sıralama + Union-Find	O(m log n)	minimum kapsayan ağaç
Prim	öncelikli kuyruk	O(m log n)	minimum kapsayan ağaç

Veri yapısı algoritmanın davranışını belirler: BFS kuyruk, DFS yığın, Dijkstra ve Prim öncelikli kuyruk kullanır; topolojik sıralama ise yalnızca DAG'lerde tanımlıdır.

ÖZET

Etkileşimli Simülatörler

Yönsüz Çizge Simülatörü

Düğüm/kenar ekle-sil, BFS ve DFS, bağlantılı bileşenler, komşuluk listesi ve matrisi görünümü.

Yönlü Çizge Simülatörü

Oklu kenarlar, giriş/çıkış dereceleri ve yönlü gezinme.

Dijkstra Algoritması Simülatörü

Ağırlıklı çizgede en kısa yolların adım adım bulunması.

Uygulama: Ders Ön Koşul Planlayıcısı → Kahn algoritmasıyla derslerin dönem dönem planlanması ve ön koşul döngüsünün yakalanması.

ÖZET

Örnek Problemler

1. 10 düğümlü basit yönsüz bir çizgede en fazla kaç kenar olabilir? Derecelerin toplamı 30 ise kaç kenar vardır?
2. n = 10.000 düğüm, m = 30.000 kenar olan bir yol ağını hangi gösterimle saklarsınız? Neden?
3. Gezinme örneğindeki çizgede BFS ve DFS'yi **H**'den başlatın (komşular alfabetik).
4. DFS'yi yığın yerine kuyrukla yazarsak hangi algoritmayı elde ederiz?
5. Ağırlıklı en kısa yol örneğinde B → C kenarı 5 yerine 1 olsaydı A'dan D'ye en kısa yol ne olurdu?
6. Kruskal örneğinde C–J kenarı neden reddedildi?
7. Prim'i aynı çizgede A yerine J'den başlatınca ilk üç seçilen kenar ve toplam ağırlık ne olur?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. Yönsüz kenarlar (0,1), (0,2), (1,3), (2,3), (3,4), (4,5): 0'dan BFS sırası ve mesafeler? (komşular küçükten büyüğe)
2. Aynı çizgenin komşuluk matrisinde kaç hücre 1'dir? Komşuluk listesinde toplam kaç kayıt vardır?
3. Yönlü 0→1, 1→2, 2→0, 2→3 için `donguVar(0)` hangi kenarı incelerken true döner?
4. 0→1, 0→2, 1→3, 2→3 DAG'ı için kaç farklı topolojik sıra vardır? Yazın.
5. `onceki = [-1, 0, 0, 1, 3]` ise 0'dan 4'e yol nedir? 2'den 4'e?
6. 5 düğümlü bir döngü (C_5) iki parçalı mıdır? Bağlantılı bileşen sayısı kaçtır?
7. Tembel Prim'de kuyruğa en çok kaç kayıt girer? Kayıt ne zaman bayatlar?

▶ **Cevaplar / ipuçları**

Kaynaklar

- Goodrich, Tamassia ve Goldwasser, *Data Structures and Algorithms in Java*, 6. baskı, 2014.
- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022.
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, 2011.

Son

Bölüm 14: Çizgeler

Çalışma soruları · Simülatörler: Yönsüz Çizge Simülatörü, Yönlü Çizge Simülatörü, Dijkstra Algoritması Simülatörü, Ders Ön Koşul Planlayıcısı · Terim sözlüğü