Bölüm 4.1: Çizge Gezinme Algoritmaları

Algoritmalar

Çizge türleri | Gösterim | Derinlik öncelikli arama (DFS) | Genişlik öncelikli arama (BFS) | Labirent ve en kısa yol | Tarjan SCC | Topolojik sıralama | Karmaşıklık

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Çizge kavramları

Yönlü, yönsüz, ağırlıklı, iki parçalı, tam çizge ve ağaç türlerini ayırt etmek; komşuluk listesi ve matrisiyle göstermek.

DFS

Derinlik öncelikli aramayı özyinelemeli ve yığınla yazmak, ziyaret sırasını adım adım çıkarmak, geri dönüşü açıklamak.

BFS

Genişlik öncelikli aramayı kuyrukla yazmak; katman, ebeveyn ve ağırlıksız en kısa yol kavramlarını labirent örneğinde uygulamak.

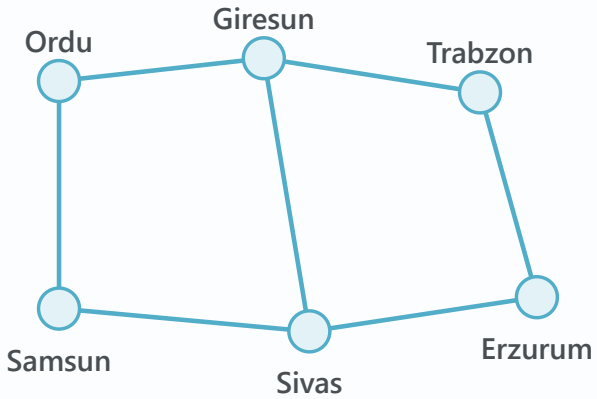
Tarjan ve analiz

Güçlü bağlantılı bileşenleri indeks / enDüşükBağ değerleriyle bulmak; tüm gezinme algoritmalarının O(V + E) zamanını gerekçelendirmek.

GİRİŞ

Çizgeler Her Yerde

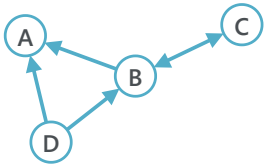
- Dünya bir **ağ** gibidir: şehirler yollarla, insanlar ilişkilerle, bilgisayarlar kablolarla birbirine bağlıdır.
- **Çizge** (graph) G = (V, E): **düğümler** (vertex, V) ve onları birleştiren **kenarlar** (edge, E).
- Çizge algoritmaları bu ağları **inceler ve anlamlandırır**: en kısa yol, gruplama, bağlantılılık, akış…
- Kullanım: sosyal ağlar, harita ve navigasyon, ağ yönlendirme, derleyiciler, oyunlar.



Şehirler düğüm, yollar kenar.

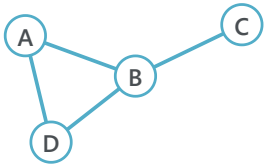
V = düğüm sayısı, E = kenar sayısı. Karmaşıklıklar bu iki sayı cinsinden verilir: $O(V + E)$, $O(V^2)$…

Çizge Türleri



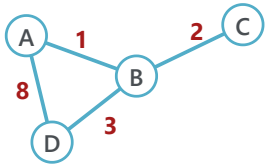
Yönlü (directed)

Kenarlar yalnızca ok yönünde geçilir.



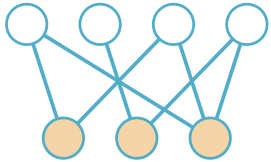
Yönsüz (undirected)

Kenarlar iki yönde de geçilir.



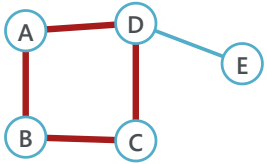
Ağırlıklı (weighted)

Her kenara bir ağırlık (maliyet, uzaklık) atanır.



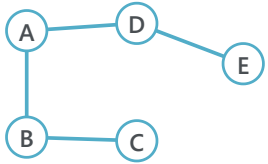
İki parçalı (bipartite)

Kenarlar yalnızca farklı kümelerdeki düğümleri bağlar.



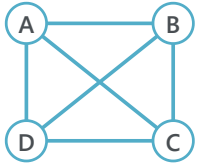
Döngülü (cyclic)

A–D–C–B–A gibi başladığı yere dönen bir yol vardır.



Ağaç (tree)

Bağlı ve döngüsüz; iki düğüm arasında tek yol.



Tam (complete)

Her düğüm çifti bir kenarla bağlı: n(n−1)/2 kenar.



Basit olmayan

Çoklu kenar ve öz döngü içerir; **basit çizge** içermez.

GİRİŞ

Çizge Algoritmalarının Çeşitleri

1 · Gezinme (graph traversal)

DFS bir düğümden başlayıp dallanarak, **BFS** katman katman tüm ağı gezer.

2 · En kısa yol (shortest path)

Dijkstra: başlangıçtan diğer tüm düğümlere en kısa yollar.

3 · Minimum kapsayan ağaç (MST)

Kruskal: ağı minimum toplam maliyetle bağlayan kenarları seçer.

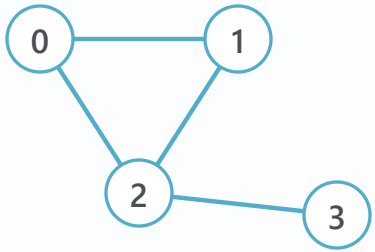
4 · Ağ akışı (network flow)

Kapasiteli bir ağda kaynaktan hedefe gönderilebilecek maksimum akış.

Algoritma	Günlük örnek
DFS	Bir labirentten çıkış yolu ararken bir koridoru sonuna kadar izleyip geri dönmek
BFS	Bir haberin şehirde kişiden kişiye yayılması: önce yakın çevre, sonra onların çevresi
Dijkstra	Bir kuryenin en kısa sürede teslimat yapacağı rotayı bulmak

GİRİŞ

Çizgeyi Bellekte Tutmak



V = 4, E = 4

Komşuluk listesi

```
0: 1, 2
1: 0, 2
2: 0, 1, 3
3: 2
```

Komşuluk matrisi

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

	Komşuluk listesi	Komşuluk matrisi
Bellek	O(V + E)	O(V^2)
u'nun tüm komşuları	O(derece(u))	O(V)
(u, v) kenarı var mı?	O(derece(u))	O(1)
Gezinme (DFS / BFS)	**O(V + E)**	**O(V^2)**

Bu bölümdeki Java kodları komşuluk listesi kullanır: `List<Integer>[] komsu`.

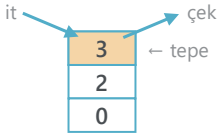
GEZİNME

Çizge Gezinme

- Gezinme (graph traversal), çizgenin yapısını **sistematik** biçimde keşfetmek için her düğümü (ulaşılabiliyorsa) **tam bir kez** ziyaret eder.
- İki ana yaklaşım vardır; fark, sıradaki düğümün **hangi veri yapısından** alındığıdır:

Derinlik Öncelikli Arama (DFS, Depth-First Search)

Yığın (stack, LIFO) kullanır. Bir düğümden başlar, mümkün olduğunca **derine** iner; komşuları bitince **geri döner** (backtracking).



Genişlik Öncelikli Arama (BFS, Breadth-First Search)

Kuyruk (queue, FIFO) kullanır. Başlangıca uzaklığı 1 olanları, sonra 2 olanları… **düzey düzey** ziyaret eder.



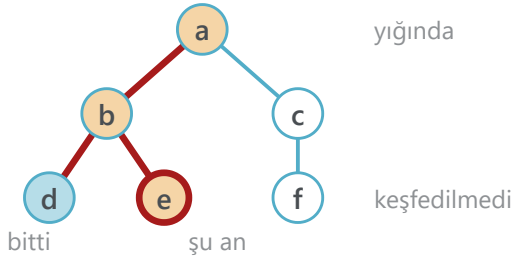
Ziyaret dizisi şart: çizgede döngü olabilir (ağaçtan farkı budur). `ziyaret[]` işaretlenmezse algoritma aynı döngüde sonsuza kadar dolaşır.

DFS: Fikir

- Labirentte bir koridora girilir ve **çıkmaza kadar** ilerlenir; çıkmazda son yol ayrımına **geri dönülür** ve henüz denenmemiş koridor seçilir.
- Özyinelemeli sürümde geri dönüş noktalarını **çağrı yığını** hatırlar; yinelemeli sürümde açık bir **yığın** kullanılır.
- Ziyaret edilen düğümler ve onlara ilk ulaşılan kenarlar bir **DFS ağacı** oluşturur.
- Ziyaret sırası komşuların **sırasına** bağlıdır; aynı çizgede farklı ama geçerli DFS sıraları vardır.

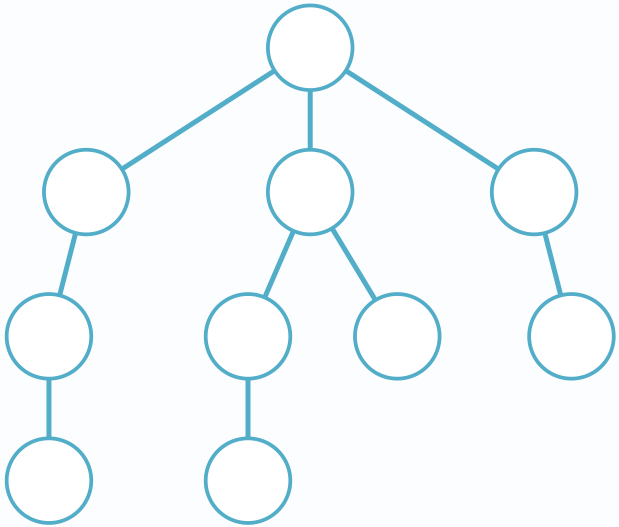
Düğüm durumları

beyaz: henüz keşfedilmedi
sarı: keşfedildi, işi sürüyor (yığında)
koyu: bitti, tüm komşuları incelendi



DFS soruları: "ulaşılabilir mi?", "döngü var mı?", "kaç bileşen var?", "hangi sırayla yapılmalı?" (topolojik sıra), "güçlü bağlantılı bileşenler neler?" (Tarjan).

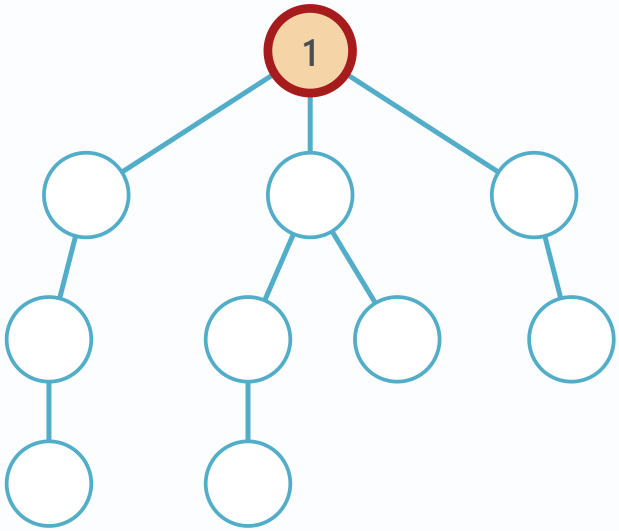
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): boş

Başlangıç: hiçbir düğüm ziyaret edilmedi. DFS kökten başlayacak; düğümlere **ziyaret sırası** numarası yazılacak.

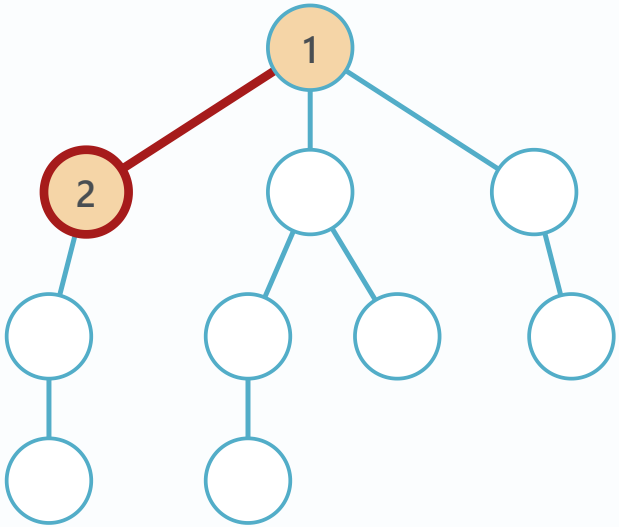
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1

Kök ziyaret edilir (1). Komşulardan ilki seçilip **derine** inilecek.

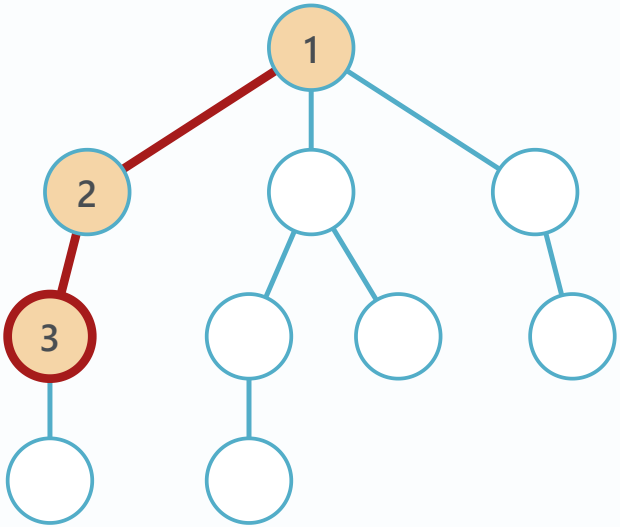
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1 → 2

1 düğümünün ilk keşfedilmemiş komşusuna inilir: **2**. Yığın derinleşir.

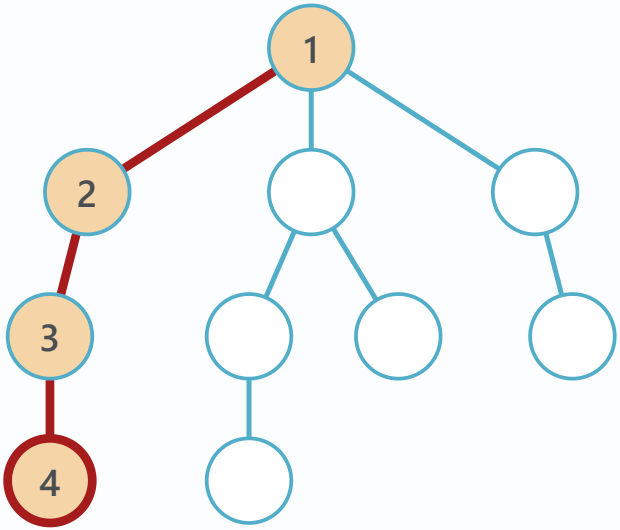
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1 → 2 → 3

2 düğümünün ilk keşfedilmemiş komşusuna inilir: **3**. Yığın derinleşir.

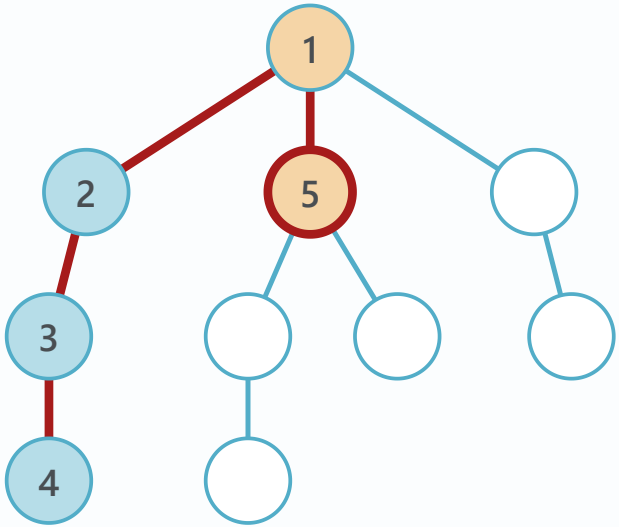
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1 → 2 → 3 → 4

3 düğümünün ilk keşfedilmemiş komşusuna inilir: **4**. Yığın derinleşir.

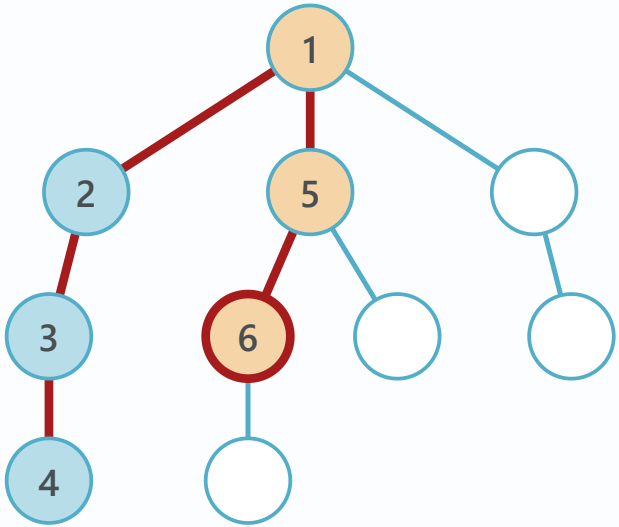
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1 → 5

4, 3, 2 düğümlerinin keşfedilmemiş komşusu kalmadı → **geri dönüldü** (backtrack). 1 düğümünün sıradaki çocuğu ziyaret edilir: **5**.

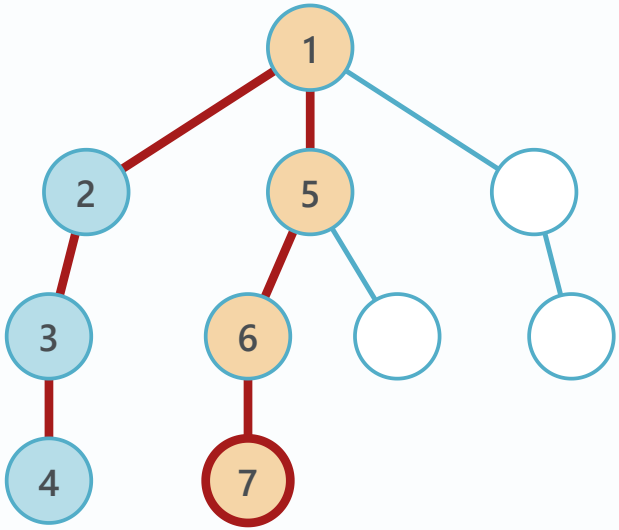
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1 → 5 → 6

5 düğümünün ilk keşfedilmemiş komşusuna inilir: **6**. Yığın derinleşir.

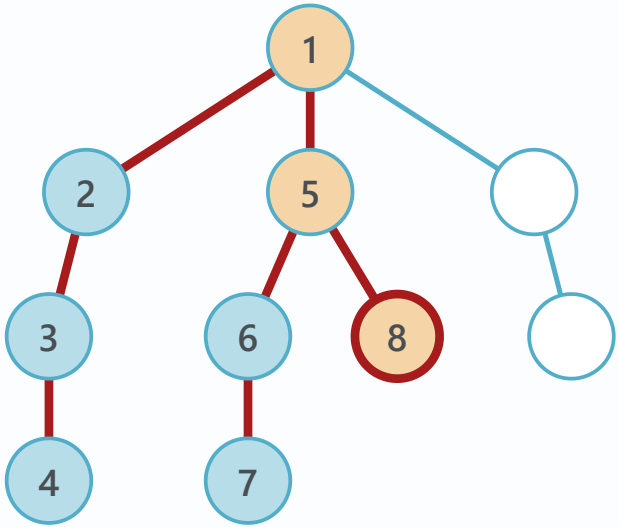
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1 → 5 → 6 → 7

6 düğümünün ilk keşfedilmemiş komşusuna inilir: **7**. Yığın derinleşir.

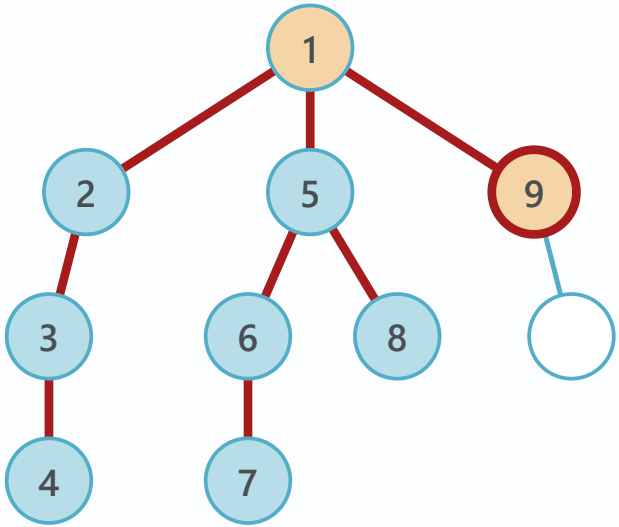
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1 → 5 → 8

7, 6 düğümlerinin keşfedilmemiş komşusu kalmadı → **geri dönüldü** (backtrack). 5 düğümünün sıradaki çocuğu ziyaret edilir: **8**.

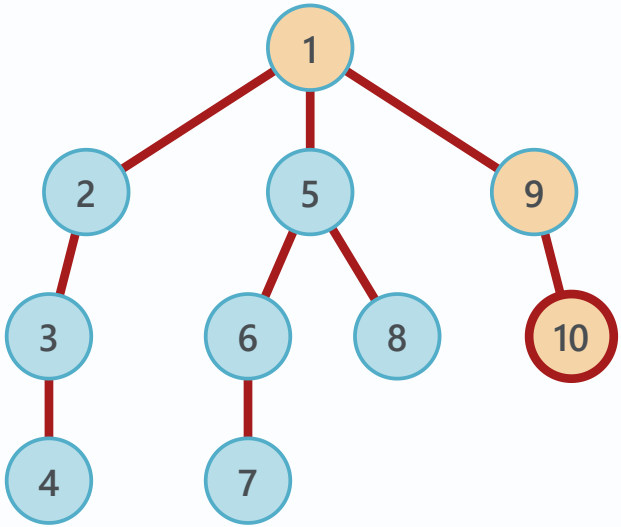
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1 → 9

8, 5 düğümlerinin keşfedilmemiş komşusu kalmadı → **geri dönüldü** (backtrack). 1 düğümünün sıradaki çocuğu ziyaret edilir: **9**.

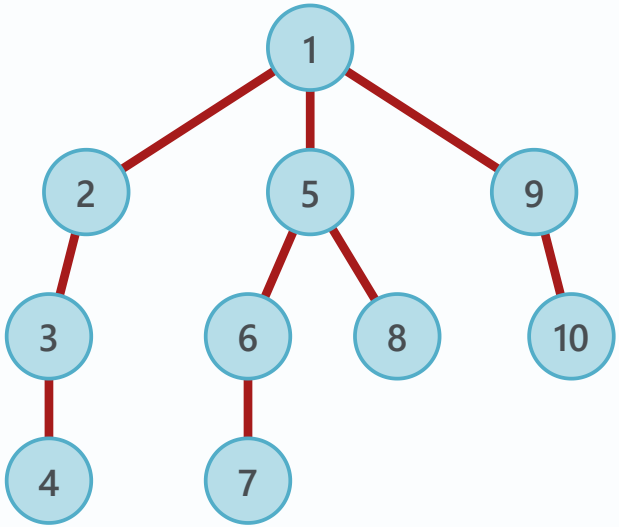
DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): 1 → 9 → 10

9 düğümünün ilk keşfedilmemiş komşusuna inilir: **10**. Yığın derinleşir.

DFS Adım Adım: Ziyaret Sırası



Yığın (özyineleme): boş

Tüm düğümler ziyaret edildi; yığın boşaldı. Ziyaret sırası: **1, 2, 3, …, 10**: her dal sonuna kadar izlenip sonra geri dönüldü.

Özyinelemeli DFS: Sözde Kod

İngilizce sözde kod

```
procedure DFS(G, v) is
  label v as discovered
  for all directed edges from v to w
      that are in G.adjacentEdges(v) do
    if vertex w is not labeled
        as discovered then
      call DFS(G, w)
```

Türkçe

```
DFS(düğüm, ziyaretEdilenler):
  eğer düğüm ziyaretEdilenler
        içinde değilse:
    düğüm'ü ziyaretEdilenler'e ekle
    düğüm'ü işle (örneğin, yazdır)
    her bir komşu düğüm için:
      DFS(komşu, ziyaretEdilenler)
```

- Soldaki sürüm komşuyu **çağırmadan önce**, sağdaki çağrının **başında** denetler; ikisi aynı sırayı üretir. Sağdakinde gereksiz çağrılar yapılır ama hemen döner.
- Bağlı olmayan bir çizgede tüm düğümleri gezmek için DFS, ziyaret edilmemiş **her düğümden** yeniden başlatılır (her başlatma bir bağlantılı bileşen).

Yinelemeli DFS: Sözde Kod

İngilizce sözde kod

```
procedure DFS_iterative(G, v) is
  let S be a stack
  S.push(v)
  while S is not empty do
    v = S.pop()
    if v is not labeled as discovered then
      label v as discovered
      for all edges from v to w
          in G.adjacentEdges(v) do
        S.push(w)
```

Türkçe

```
DFS(başlangıç_düğüm):
  boş bir yığın oluştur
  başlangıç_düğüm'ü yığına ekle
  döngü yığın boş değilken:
    mevcut_düğüm = yığından çıkar
    eğer mevcut_düğüm ziyaretEdilenler
          içinde değilse:
      mevcut_düğüm'ü ziyaretEdilenler'e ekle
      her bir komşu_düğüm için:
        komşu_düğüm'ü yığına ekle
```

- Özyineleme yerine **açık bir yığın**: çok derin çizgelerde `StackOverflowError` riski yoktur.
- İşaretleme **çekerken** yapılır; bu yüzden bir düğüm yığına birden çok kez girebilir (yığın boyutu O(E)).
- Komşular itildiği için en son itilen önce çıkar: sıra, özyinelemeli sürümün **tersine** döner. Aynı sıra istenirse komşular ters sırada itilir.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

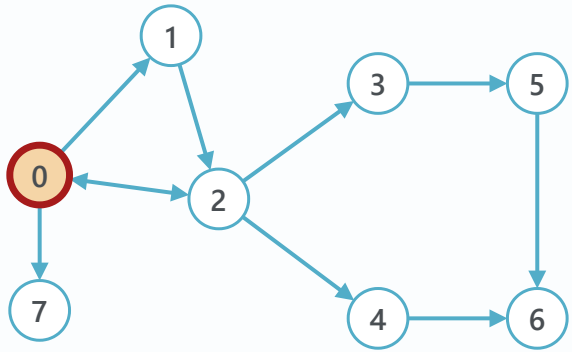
`v = 0` `w = –`

ÇAĞRI YIĞINI

`dfs(0)`

ÇIKTI

```
0
```



Örnek yönlü çizge: 0→1, 0→2, 0→7, 1→2, 2→0, 2→3, 2→4, 3→5, 4→6, 5→6. `dfs(0)` : 0 işaretlenir, yazdırılır.

Adım 1 / 14

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

`v = 1` `w = –`

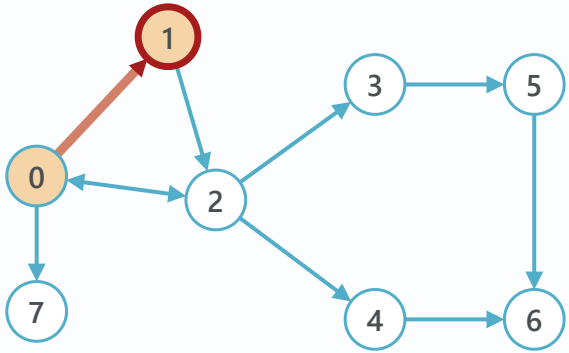
ÇAĞRI YIĞINI

`dfs(1)`

`dfs(0)`

ÇIKTI

```
0 1
```



0'ın komşusu 1 ziyaret edilmemiş → `dfs(1)` : işaretlenir, yazdırılır. Yığına yeni çerçeve eklendi.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

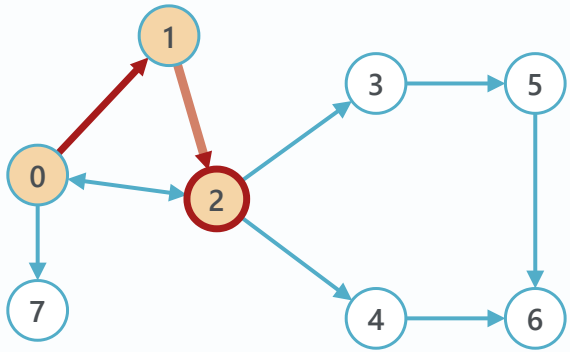
`v = 2` `w = –`

ÇAĞRI YIĞINI

- `dfs(2)`
- `dfs(1)`
- `dfs(0)`

ÇIKTI

```
0 1 2
```



1'in komşusu 2 ziyaret edilmemiş → `dfs(2)`: işaretlenir, yazdırılır. Yığına yeni çerçeve eklendi.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

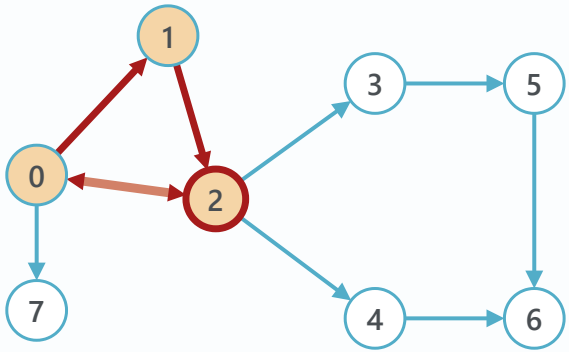
`v = 2` `w = 0`

ÇAĞRI YIĞINI

- `dfs(2)`
- `dfs(1)`
- `dfs(0)`

ÇIKTI

```
0 1 2
```



2'nin komşusu 0 **zaten ziyaret edildi** → atlanır (döngüye girilmez).

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

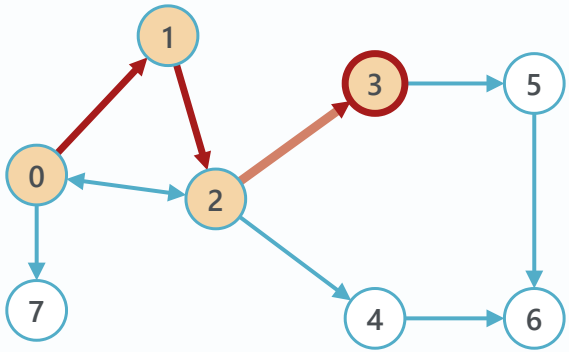
`v = 3` `w = –`

ÇAĞRI YIĞINI

- `dfs(3)`
- `dfs(2)`
- `dfs(1)`
- `dfs(0)`

ÇIKTI

```
0 1 2 3
```



2'nin komşusu 3 ziyaret edilmemiş → `dfs(3)` : işaretlenir, yazdırılır. Yığına yeni çerçeve eklendi.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

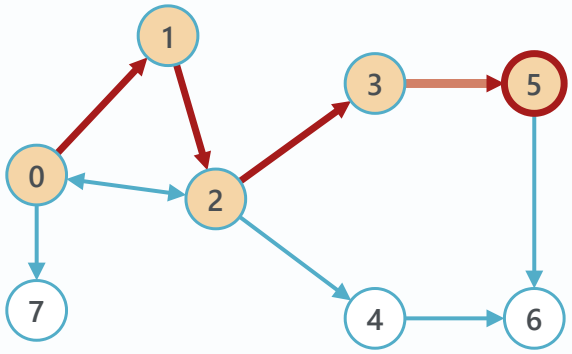
`v = 5` `w = –`

ÇAĞRI YIĞINI

- `dfs(5)`
- `dfs(3)`
- `dfs(2)`
- `dfs(1)`
- `dfs(0)`

ÇIKTI

```
0 1 2 3 5
```



3'ün komşusu 5 ziyaret edilmemiş → `dfs(5)` : işaretlenir, yazdırılır. Yığına yeni çerçeve eklendi.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

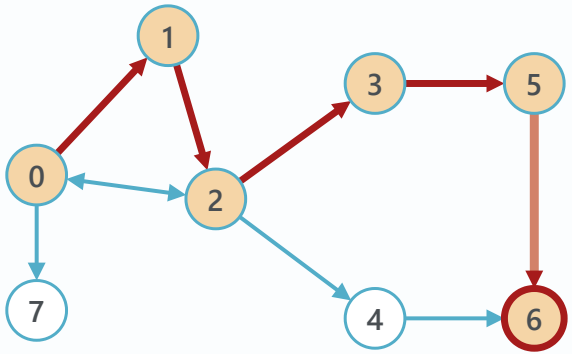
`v = 6` `w = –`

ÇAĞRI YIĞINI

- `dfs(6)`
- `dfs(5)`
- `dfs(3)`
- `dfs(2)`
- `dfs(1)`
- `dfs(0)`

ÇIKTI

```
0 1 2 3 5 6
```



5'in komşusu 6 ziyaret edilmemiş → `dfs(6)` : işaretlenir, yazdırılır. Yığına yeni çerçeve eklendi.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

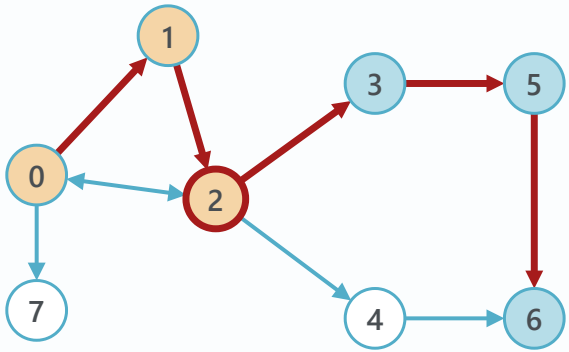
`v = 2` `w = 3`

ÇAĞRI YIĞINI

- `dfs(2)`
- `dfs(1)`
- `dfs(0)`

ÇIKTI

```
0 1 2 3 5 6
```



6, 5, 3 bitti (komşu kalmadı) → çağrı(lar) döner, kontrol **2'ye** geri gelir: geri dönüş.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

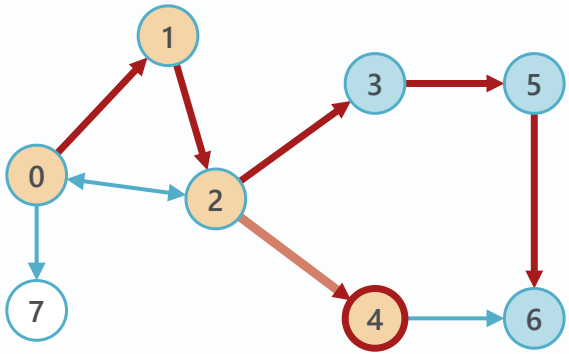
`v = 4` `w = –`

ÇAĞRI YIĞINI

- `dfs(4)`
- `dfs(2)`
- `dfs(1)`
- `dfs(0)`

ÇIKTI

```
0 1 2 3 5 6 4
```



2'nin komşusu 4 ziyaret edilmemiş → `dfs(4)` : işaretlenir, yazdırılır. Yığına yeni çerçeve eklendi.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

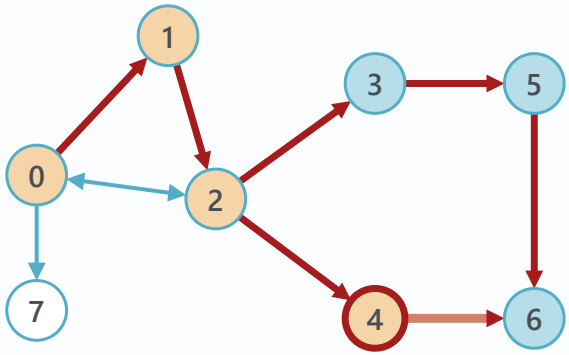
`v = 4` `w = 6`

ÇAĞRI YIĞINI

- `dfs(4)`
- `dfs(2)`
- `dfs(1)`
- `dfs(0)`

ÇIKTI

```
0 1 2 3 5 6 4
```



4'ün komşusu 6 **zaten ziyaret edildi** → atlanır (döngüye girilmez).

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

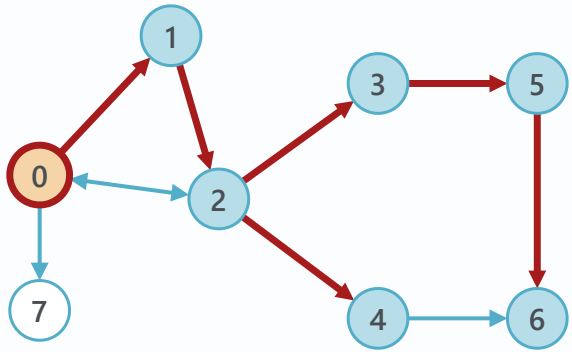
`v = 0` `w = 1`

ÇAĞRI YIĞINI

`dfs(0)`

ÇIKTI

```
0 1 2 3 5 6 4
```



4, 2, 1 bitti (komşu kalmadı) → çağrı(lar) döner, kontrol **0'a** geri gelir: geri dönüş.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

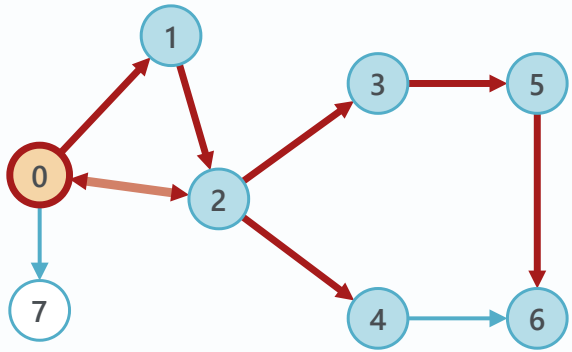
`v = 0` `w = 2`

ÇAĞRI YIĞINI

`dfs(0)`

ÇIKTI

```
0 1 2 3 5 6 4
```



0'ın komşusu 2 **zaten ziyaret edildi** → atlanır (döngüye girilmez).

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

`v = 7` `w = –`

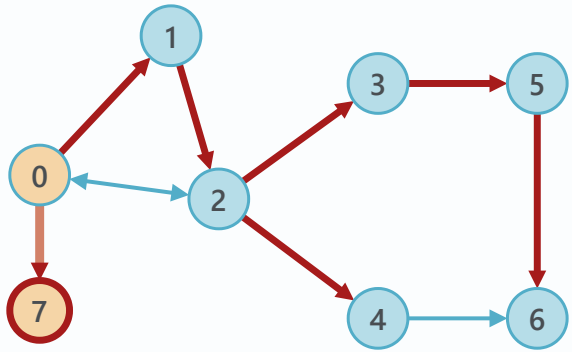
ÇAĞRI YIĞINI

`dfs(7)`

`dfs(0)`

ÇIKTI

```
0 1 2 3 5 6 4 7
```



0'ın komşusu 7 ziyaret edilmemiş → `dfs(7)` : işaretlenir, yazdırılır. Yığına yeni çerçeve eklendi.

Özyinelemeli DFS: `dfs(0)` İzleme

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

DEĞİŞKENLER

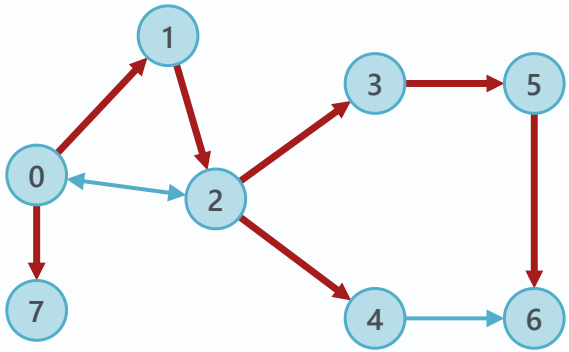
`v = –` `w = –`

ÇAĞRI YIĞINI

boş

ÇIKTI

```
0 1 2 3 5 6 4 7
```



7, 0 bitti; `dfs(0)` da döner. Çıktı: **0 1 2 3 5 6 4 7**. Kalın kenarlar **DFS ağacını** oluşturur.

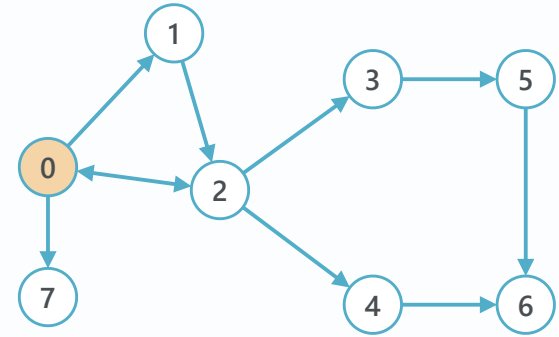
Yığınla DFS: `dfsYigin(0)` İzleme

```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
▶    yigin.push(s);
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
8        ziyaret[v] = true;
9        System.out.print(v + " ");
10       for (int w : komsu[v])
11         yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = –` `yigin = [0]`

ÇIKTI



yığın (alt → üst)

0

Başlangıç düğümü 0 yığına itilir. Henüz hiçbir düğüm işaretli değil.

Adım 1 / 13

Yığınla DFS: `dfsYigin(0)` İzleme

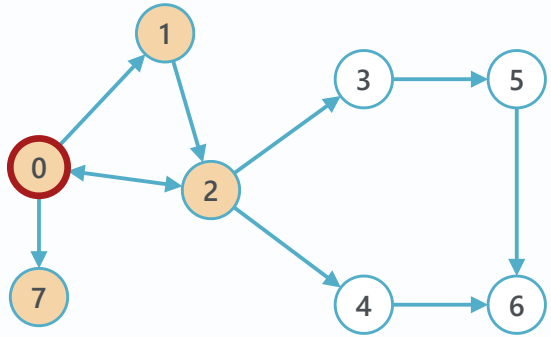
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
▶        ziyaret[v] = true;
▶        System.out.print(v + " ");
▶        for (int w : komsu[v])
▶          yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 0` `yigin = [1, 2, 7]`

ÇIKTI

```
0
```



yığın (alt → üst)

1	2	7

0 çekildi: işaretlenir, yazdırılır; komşuları 1, 2, 7 sırayla itilir → en son itilen **7** en üstte.

Yığınla DFS: `dfsYigin(0)` İzleme

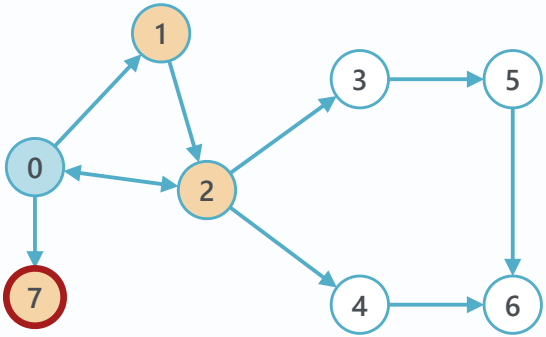
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
▶        ziyaret[v] = true;
▶        System.out.print(v + " ");
10       for (int w : komsu[v])
11         yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 7` `yigin = [1, 2]`

ÇIKTI

```
0 7
```



yığın (alt → üst)



7 çekildi: işaretlenir, yazdırılır; komşusu yok, yığına bir şey eklenmez.

Adım 3 / 13

Yığınla DFS: `dfsYigin(0)` İzleme

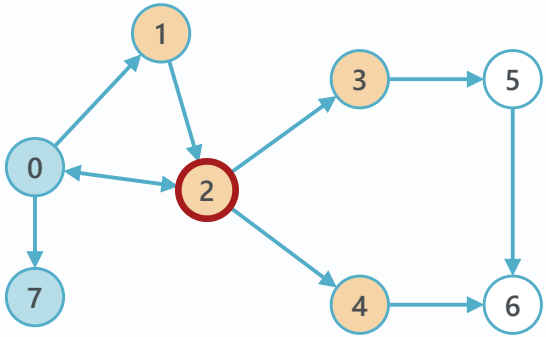
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
▶        ziyaret[v] = true;
▶        System.out.print(v + " ");
▶        for (int w : komsu[v])
▶          yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 2` `yigin = [1, 0, 3, 4]`

ÇIKTI

```
0 7 2
```



yığın (alt → üst)

1	0	3	4

2 çekildi: işaretlenir, yazdırılır; komşuları 0, 3, 4 sırayla itilir → en son itilen **4** en üstte.

Yığınla DFS: `dfsYigin(0)` İzleme

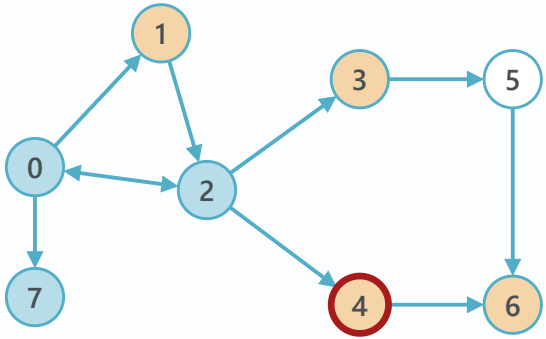
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
▶        ziyaret[v] = true;
▶        System.out.print(v + " ");
▶        for (int w : komsu[v])
▶          yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 4` `yigin = [1, 0, 3, 6]`

ÇIKTI

```
0 7 2 4
```



yığın (alt → üst)

1	0	3	6

4 çekildi: işaretlenir, yazdırılır; komşusu 6 itilir → en son itilen **6** en üstte.

Yığınla DFS: `dfsYigin(0)` İzleme

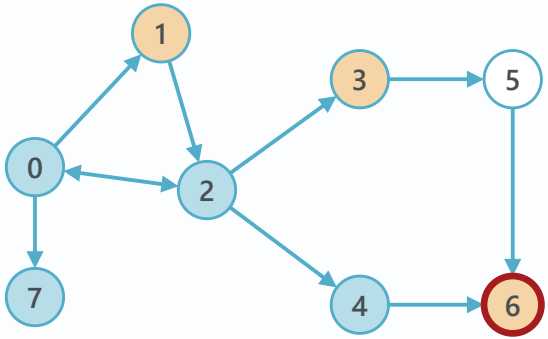
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
▶        ziyaret[v] = true;
▶        System.out.print(v + " ");
10       for (int w : komsu[v])
11         yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 6` `yigin = [1, 0, 3]`

ÇIKTI

```
0 7 2 4 6
```



yığın (alt → üst)

1	0	3

6 çekildi: işaretlenir, yazdırılır; komşusu yok, yığına bir şey eklenmez.

Yığınla DFS: `dfsYigin(0)` İzleme

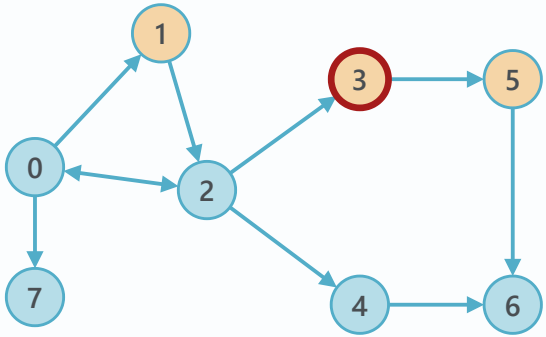
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
▶        ziyaret[v] = true;
▶        System.out.print(v + " ");
▶        for (int w : komsu[v])
▶          yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 3` `yigin = [1, 0, 5]`

ÇIKTI

```
0 7 2 4 6 3
```



yığın (alt → üst)

1	0	5

3 çekildi: işaretlenir, yazdırılır; komşusu 5 itilir → en son itilen **5** en üstte.

Yığınla DFS: `dfsYigin(0)` İzleme

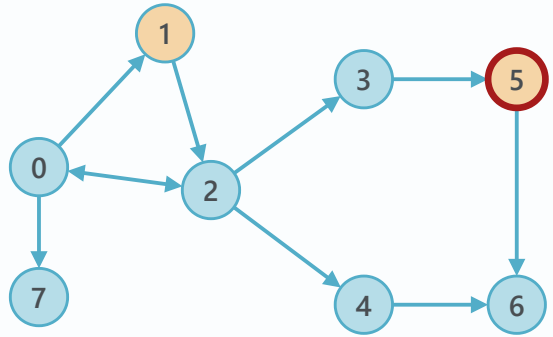
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
▶        ziyaret[v] = true;
▶        System.out.print(v + " ");
▶        for (int w : komsu[v])
▶          yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 5` `yigin = [1, 0, 6]`

ÇIKTI

```
0 7 2 4 6 3 5
```



yığın (alt → üst)

1	0	6

5 çekildi: işaretlenir, yazdırılır; komşusu 6 itilir → en son itilen **6** en üstte.

Yığınla DFS: `dfsYigin(0)` İzleme

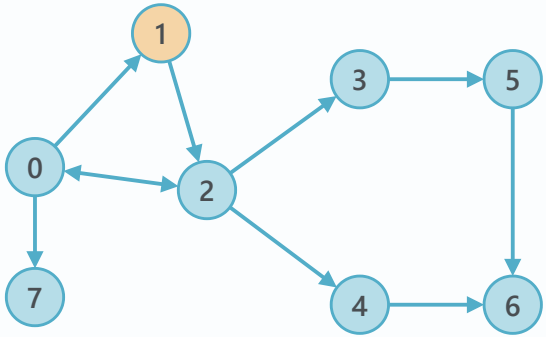
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
▶      int v = yigin.pop();
▶      if (!ziyaret[v]) {
8        ziyaret[v] = true;
9        System.out.print(v + " ");
10       for (int w : komsu[v])
11         yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 6` `yigin = [1, 0]`

ÇIKTI

```
0 7 2 4 6 3 5
```



yığın (alt → üst)

1	0

6 çekildi ama **zaten ziyaret edilmiş** → hiçbir şey yapılmaz. (Bir düğüm yığına birden çok kez girebilir.)

Yığınla DFS: `dfsYigin(0)` İzleme

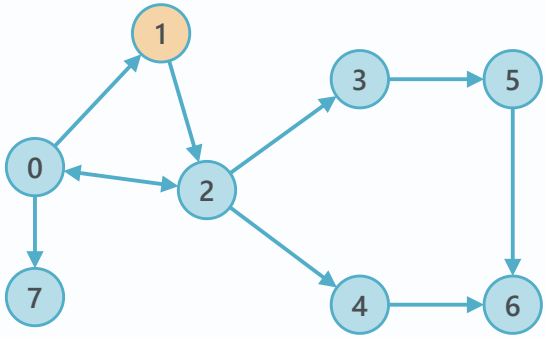
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
▶      int v = yigin.pop();
▶      if (!ziyaret[v]) {
8        ziyaret[v] = true;
9        System.out.print(v + " ");
10       for (int w : komsu[v])
11         yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 0` `yigin = [1]`

ÇIKTI

```
0 7 2 4 6 3 5
```



yığın (alt → üst)

1

0 çekildi ama **zaten ziyaret edilmiş** → hiçbir şey yapılmaz. (Bir düğüm yığına birden çok kez girebilir.)

Yığınla DFS: `dfsYigin(0)` İzleme

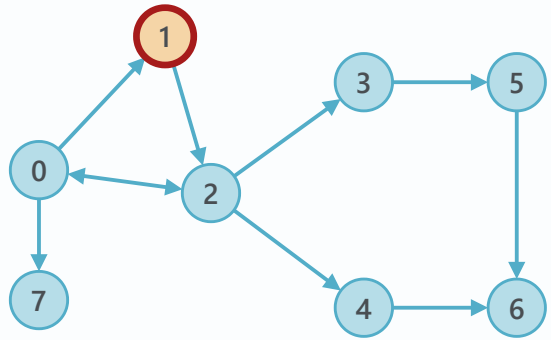
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
▶        ziyaret[v] = true;
▶        System.out.print(v + " ");
▶        for (int w : komsu[v])
▶          yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 1` `yigin = [2]`

ÇIKTI

```
0 7 2 4 6 3 5 1
```



yığın (alt → üst)

2

1 çekildi: işaretlenir, yazdırılır; komşusu 2 itilir → en son itilen **2** en üstte.

Yığınla DFS: `dfsYigin(0)` İzleme

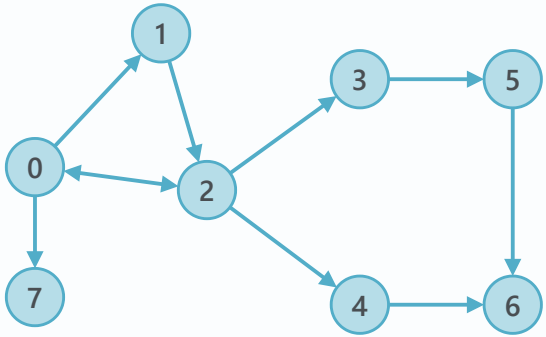
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
5    while (!yigin.isEmpty()) {
▶      int v = yigin.pop();
▶      if (!ziyaret[v]) {
8        ziyaret[v] = true;
9        System.out.print(v + " ");
10       for (int w : komsu[v])
11         yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = 2` `yigin = []`

ÇIKTI

```
0 7 2 4 6 3 5 1
```



yığın (alt → üst)

2 çekildi ama **zaten ziyaret edilmiş** → hiçbir şey yapılmaz. (Bir düğüm yığına birden çok kez girebilir.)

Yığınla DFS: `dfsYigin(0)` İzleme

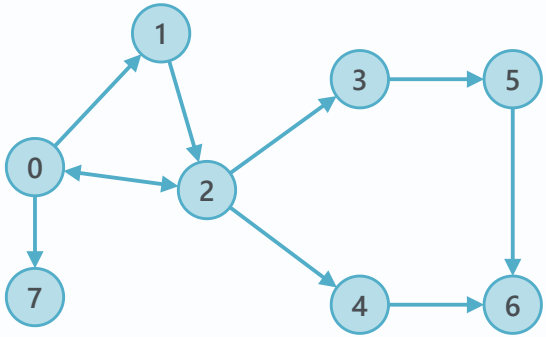
```
1  void dfsYigin(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Deque<Integer> yigin = new ArrayDeque<>
   ();
4    yigin.push(s);
▶    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {
8        ziyaret[v] = true;
9        System.out.print(v + " ");
10       for (int w : komsu[v])
11         yigin.push(w);
12     }
13   }
14 }
```

DEĞİŞKENLER

`v = –` `yigin = []`

ÇIKTI

```
0 7 2 4 6 3 5 1
```



yığın (alt → üst)

Yığın boş, döngü biter. Çıktı: **0 7 2 4 6 3 5 1**: özyinelemeli sürümden (0 1 2 3 5 6 4 7) farklı, çünkü komşular ters sırada çıkar.

Adım 13 / 13

DFS: Analiz ve Kullanım

Karmaşıklık

- Her düğüm **bir kez** işaretlenir: O(V).
- Her düğümün komşu listesi bir kez taranır; listelerin toplam uzunluğu E (yönsüzde 2E): O(E).
- Toplam: **O(V + E)** (komşuluk listesi), **O(V^2)** (matris: her düğüm için V öğeli satır).
- Ek bellek: `ziyaret[]` O(V) + yığın: özyinelemede en uzun yol kadar, O(V); "çekerken işaretle" sürümünde O(E).

Nerede kullanılır?

- Ulaşılabilirlik, **bağlantılı bileşen** sayısı
- **Döngü tespiti**: yığındaki (sarı) bir düğüme giden kenar = geri kenar
- **Topolojik sıralama**: bitiş sırasının tersi
- **Güçlü bağlantılı bileşenler** (Tarjan, Kosaraju)
- Köprü ve eklem noktaları, labirent ve bulmaca çözme, geri izleme (backtracking)

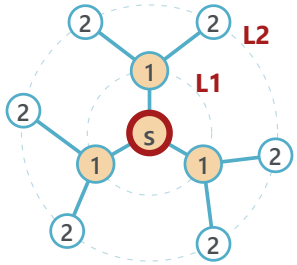
DFS en kısa yolu **bulmaz**: hedefe ilk ulaştığı yol çok uzun bir dolambaç olabilir. En az kenarlı yol için BFS gerekir.

BFS: Fikir

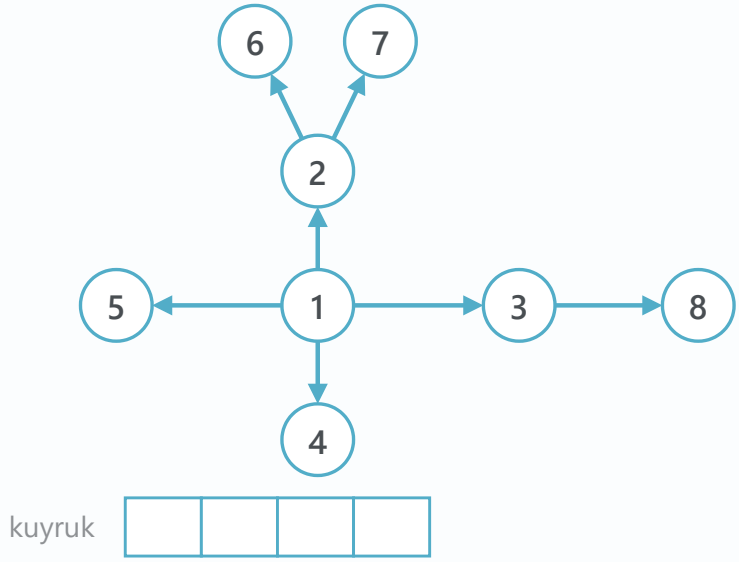
- Suya atılan taşın dalgası gibi: önce başlangıca **1 kenar** uzaktakiler, sonra **2 kenar** uzaktakiler...
- Bir haberin yayılmasını modeller: kişi haberi tüm arkadaşlarına söyler, onlar da kendi arkadaşlarına.
- **Kuyruk** (FIFO) sırayı korur: önce keşfedilen önce işlenir.
- Düğüm **kuyruğa girerken** işaretlenir; böylece her düğüm kuyruğa en fazla bir kez girer.
- Ağırlıksız çizgede başlangıçtan her düğüme **en az kenarlı yolu** bulur.

Katmanlar (levels)

L0 = {s}
L1 = s'nin komşuları
L2 = L1'in yeni komşuları
...
Lk'deki her düğümün s'ye uzaklığı tam **k**'dir.



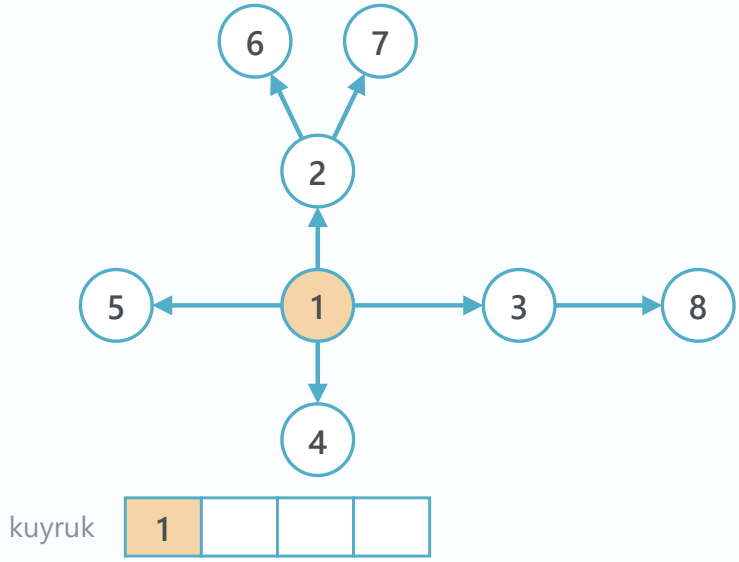
BFS Adım Adım: Katman Katman



işlenen sıra: —

Başlangıç: tüm düğümler **beyaz** (keşfedilmedi). Sarı = keşfedildi, kuyrukta bekliyor · koyu = işlendi.

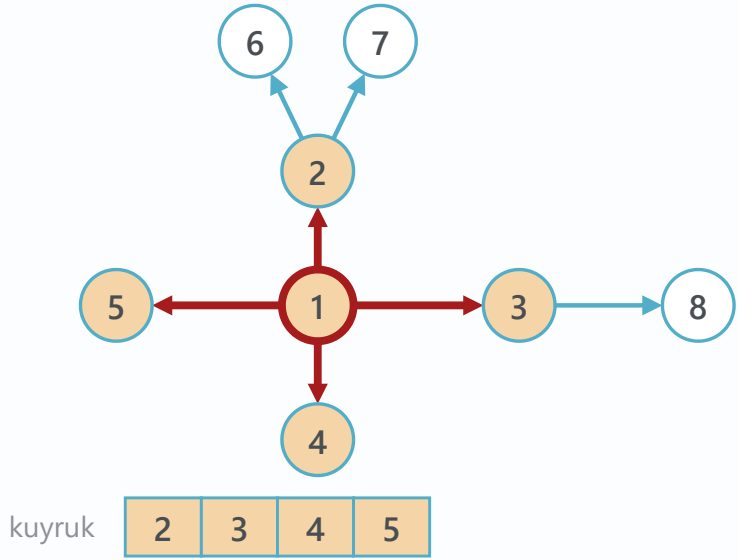
BFS Adım Adım: Katman Katman



işlenen sıra: —

Başlangıç düğümü 1 **keşfedildi** olarak işaretlenir ve kuyruğa konur.

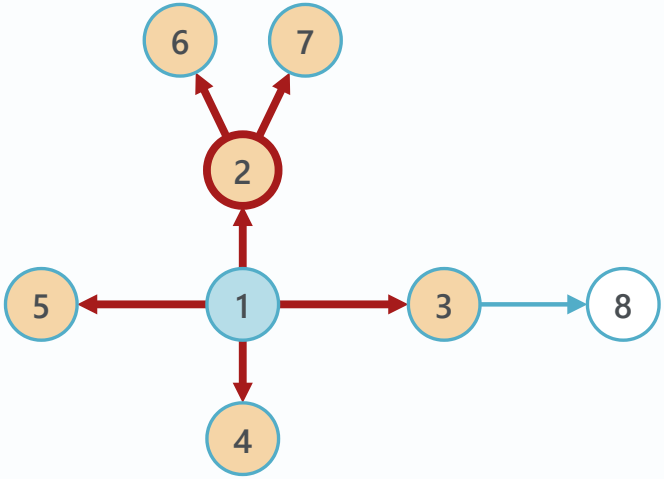
BFS Adım Adım: Katman Katman



işlenen sıra: 1

Kuyruktan **1** alınır; keşfedilmemiş komşuları 2, 3, 4, 5 işaretlenip kuyruğa eklenir (1. katman).

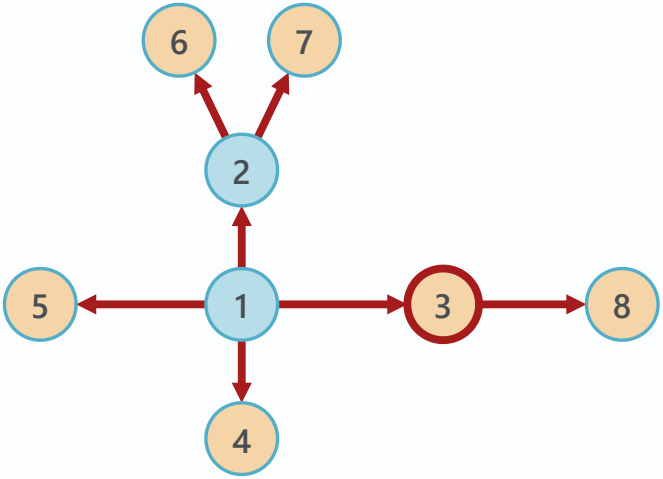
BFS Adım Adım: Katman Katman



işlenen sıra: 1 2

Kuyruktan **2** alınır; keşfedilmemiş komşuları 6, 7 işaretlenip kuyruğa eklenir (2. katman).

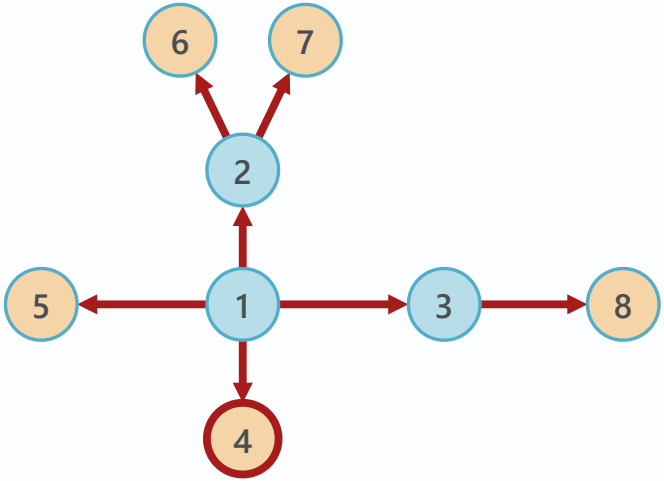
BFS Adım Adım: Katman Katman



işlenen sıra: 1 2 3

Kuyruktan **3** alınır; keşfedilmemiş komşusu 8 işaretlenip kuyruğa eklenir (2. katman).

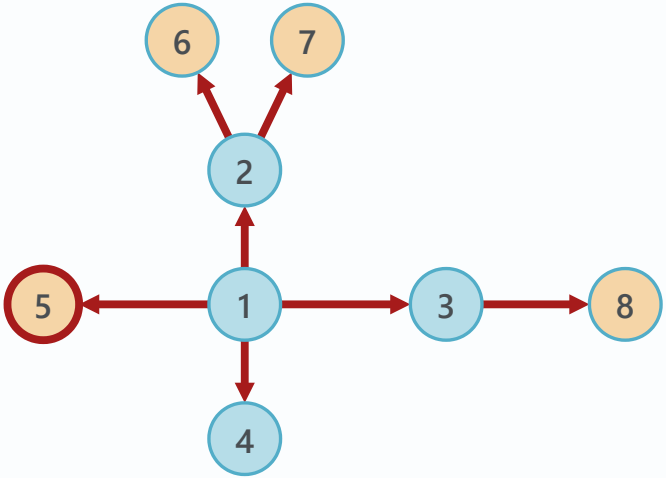
BFS Adım Adım: Katman Katman



işlenen sıra: 1 2 3 4

Kuyruktan **4** alınır; keşfedilmemiş komşusu yok.

BFS Adım Adım: Katman Katman

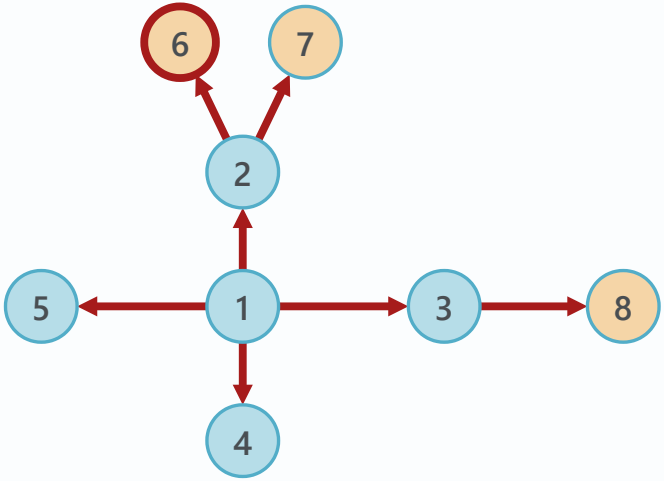


kuyruk | 6 | 7 | 8 |

işlenen sıra: 1 2 3 4 5

Kuyruktan **5** alınır; keşfedilmemiş komşusu yok.

BFS Adım Adım: Katman Katman

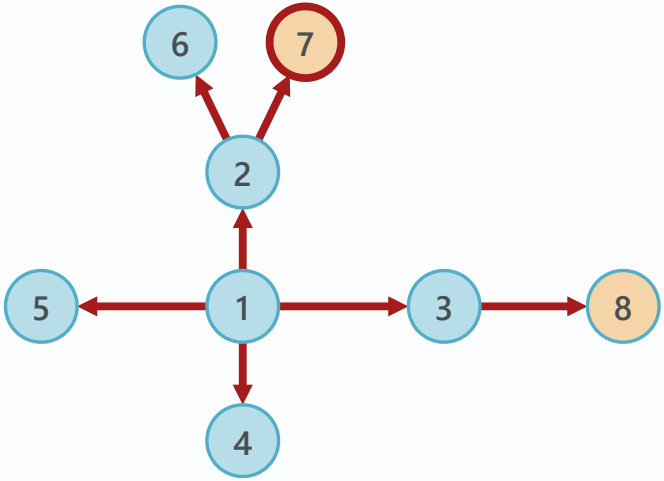


kuyruk | 7 | 8 | | |

işlenen sıra: 1 2 3 4 5 6

Kuyruktan **6** alınır; keşfedilmemiş komşusu yok.

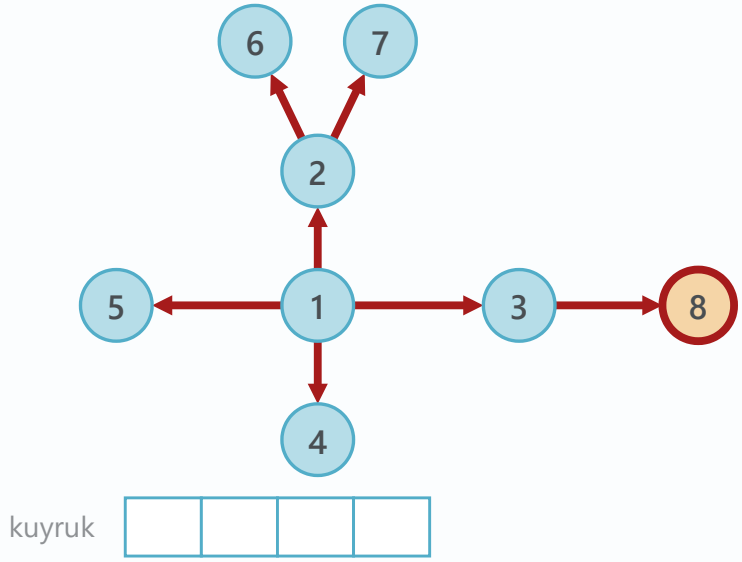
BFS Adım Adım: Katman Katman



işlenen sıra: 1 2 3 4 5 6 7

Kuyruktan **7** alınır; keşfedilmemiş komşusu yok.

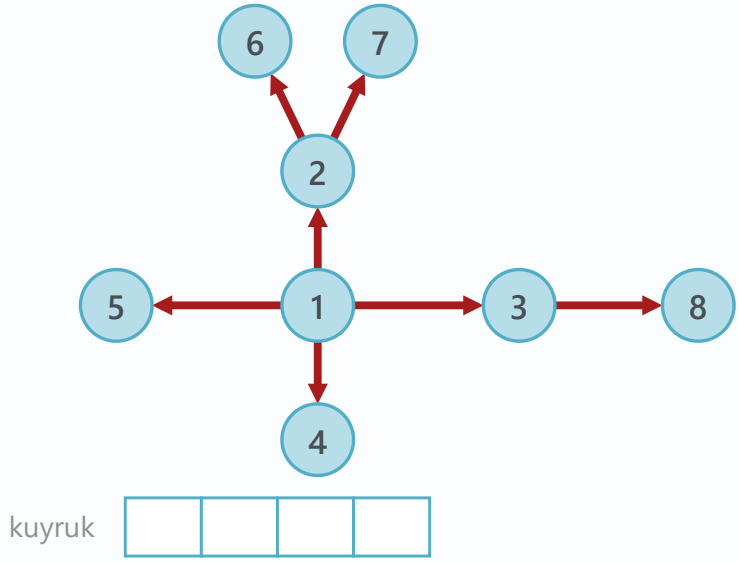
BFS Adım Adım: Katman Katman



işlenen sıra: 1 2 3 4 5 6 7 8

Kuyruktan **8** alınır; keşfedilmemiş komşusu yok → kuyruk boşaldı, BFS biter.

BFS Adım Adım: Katman Katman



işlenen sıra: 1 2 3 4 5 6 7 8

Sonuç: ziyaret sırası **1, 2, 3, 4, 5, 6, 7, 8**. Önce uzaklığı 1 olanlar (2, 3, 4, 5), sonra uzaklığı 2 olanlar (6, 7, 8): **katman katman**.

BFS: Sözde Kod

İngilizce sözde kod

```
procedure BFS(G, root) is
  let Q be a queue
  label root as explored
  Q.enqueue(root)
  while Q is not empty do
    v := Q.dequeue()
    for all edges from v to w
        in G.adjacentEdges(v) do
      if w is not labeled as explored then
        label w as explored
        w.parent := v
        Q.enqueue(w)
```

Türkçe

```
BFS(başlangıç_düğüm):
  boş bir kuyruk oluştur
  başlangıç_düğüm'ü keşfedildi işaretle
  başlangıç_düğüm'ü kuyruğa koy
  döngü kuyruk boş değilken:
    mevcut_düğüm = kuyruktan al
    her bir komşu_düğüm için
      eğer komşu_düğüm keşfedilmemiş ise
        komşu_düğüm'ü keşfedildi işaretle
        komşu_düğüm'ün ebeveyni = mevcut_düğüm
        komşu_düğüm'ü kuyruğa koy
```

- `w.parent := v` (ebeveyn) bağlantıları BFS ağacını oluşturur; hedeften köke doğru izlenince **en kısa yol** elde edilir.
- İşaretleme kuyruğa **eklerken** yapılır, çıkarırken değil: aksi hâlde aynı düğüm kuyruğa defalarca girer.

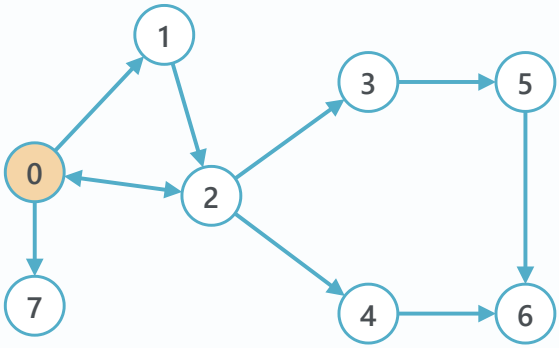
BFS: `bfs(0)` İzleme

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

DEĞİŞKENLER

`v = –` `kuyruk = [0]`

ÇIKTI



kuyruk (ön → arka)

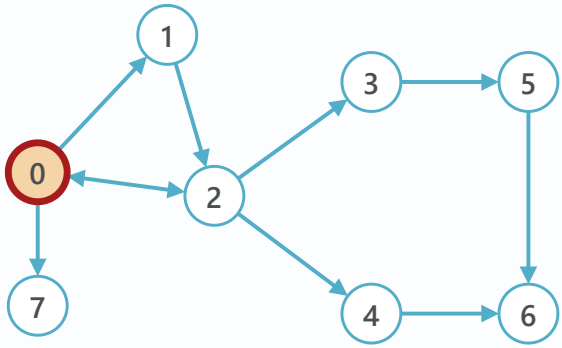
0

0 işaretlenir ve kuyruğa eklenir. Kuyruk yalnızca 0'ı içerir.

Adım 1 / 16

BFS: `bfs(0)` İzleme

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



kuyruk (ön → arka)

Kuyruğun önünden **0** alınır ve yazdırılır.

Adım 2 / 16

DEĞİŞKENLER

`v = 0` `kuyruk = []`

ÇIKTI

```
0
```

BFS: `bfs(0)` İzleme

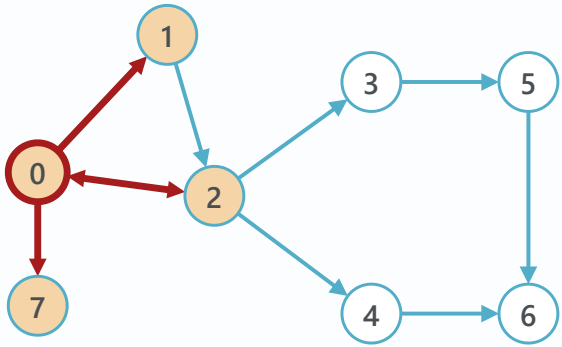
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
6      int v = kuyruk.poll();
7      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 0` `kuyruk = [1, 2, 7]`

ÇIKTI

```
0
```



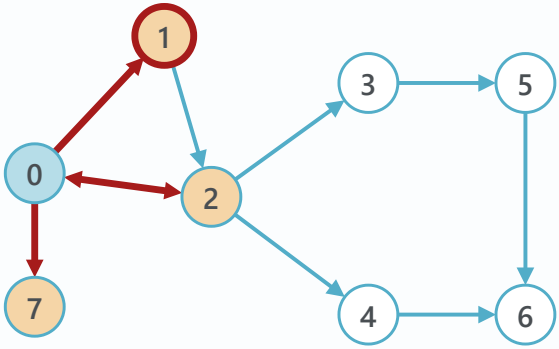
kuyruk (ön → arka)

1	2	7

0'ın komşuları: keşfedilmemiş komşular 1, 2, 7 işaretlenip kuyruğun **arkasına** eklenir.

BFS: `bfs(0)` İzleme

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



kuyruk (ön → arka)

2	7

Kuyruğun önünden **1** alınır ve yazdırılır.

Adım 4 / 16

DEĞİŞKENLER

`v = 1` `kuyruk = [2, 7]`

ÇIKTI

```
0 1
```

BFS: `bfs(0)` İzleme

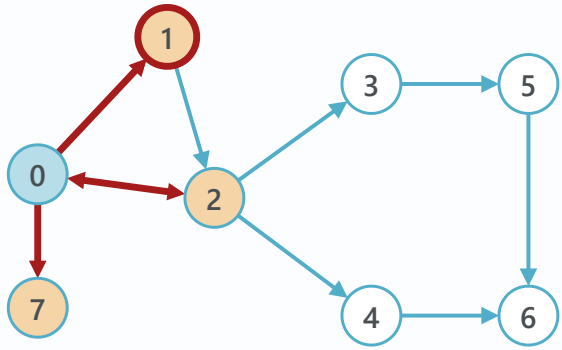
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
6      int v = kuyruk.poll();
7      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 1` `kuyruk = [2, 7]`

ÇIKTI

```
0 1
```



kuyruk (ön → arka)

2	7

1'in komşuları: 2 zaten işaretli → atlanır.

BFS: `bfs(0)` İzleme

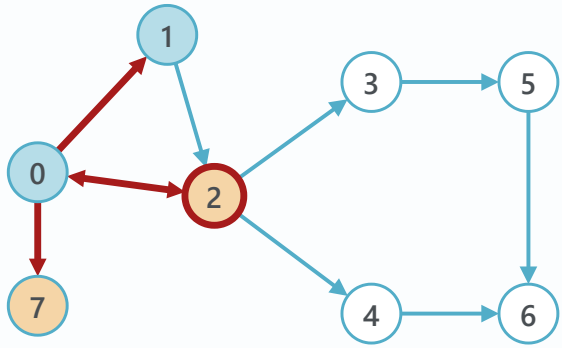
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
8      for (int w : komsu[v])
9        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 2` `kuyruk = [7]`

ÇIKTI

```
0 1 2
```



kuyruk (ön → arka)

7

Kuyruğun önünden **2** alınır ve yazdırılır.

BFS: `bfs(0)` İzleme

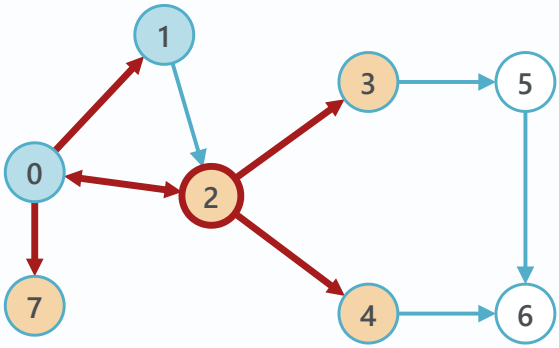
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
6      int v = kuyruk.poll();
7      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 2` `kuyruk = [7, 3, 4]`

ÇIKTI

```
0 1 2
```



kuyruk (ön → arka)

7	3	4

2'nin komşuları: keşfedilmemiş komşular 3, 4 işaretlenip kuyruğun **arkasına** eklenir; 0 zaten işaretli → atlanır.

BFS: `bfs(0)` İzleme

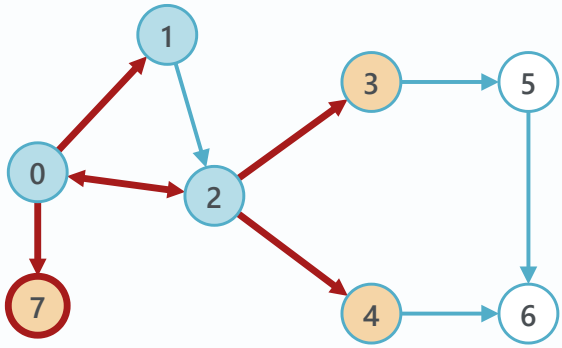
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
8      for (int w : komsu[v])
9        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 7` `kuyruk = [3, 4]`

ÇIKTI

```
0 1 2 7
```



kuyruk (ön → arka)

3	4

Kuyruğun önünden **7** alınır ve yazdırılır.

BFS: `bfs(0)` İzleme

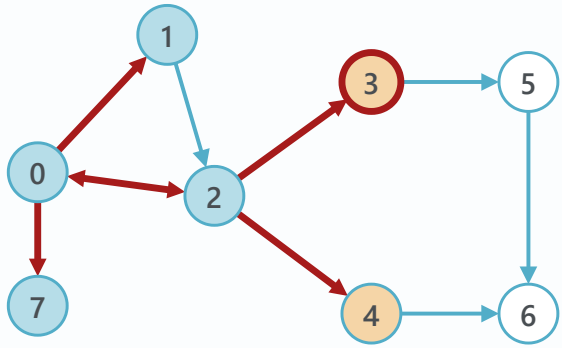
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
8      for (int w : komsu[v])
9        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 3` `kuyruk = [4]`

ÇIKTI

```
0 1 2 7 3
```



kuyruk (ön → arka)

4

Kuyruğun önünden **3** alınır ve yazdırılır.

BFS: `bfs(0)` İzleme

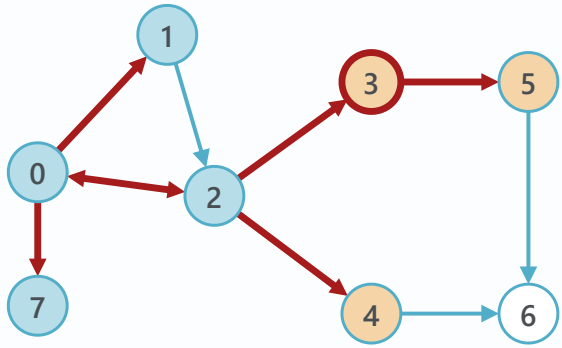
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
6      int v = kuyruk.poll();
7      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 3` `kuyruk = [4, 5]`

ÇIKTI

```
0 1 2 7 3
```



kuyruk (ön → arka)

4	5

3'ün komşuları: keşfedilmemiş komşu 5 işaretlenip kuyruğun **arkasına** eklenir.

BFS: `bfs(0)` İzleme

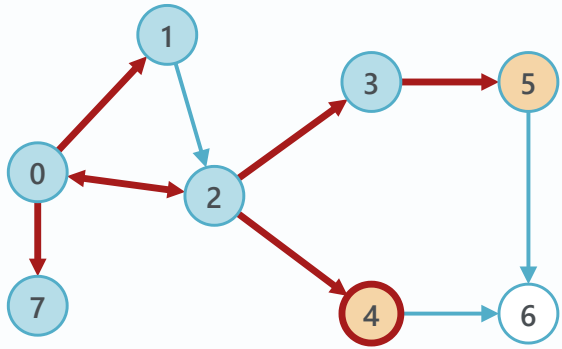
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
8      for (int w : komsu[v])
9        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 4` `kuyruk = [5]`

ÇIKTI

```
0 1 2 7 3 4
```



kuyruk (ön → arka)

5

Kuyruğun önünden **4** alınır ve yazdırılır.

BFS: `bfs(0)` İzleme

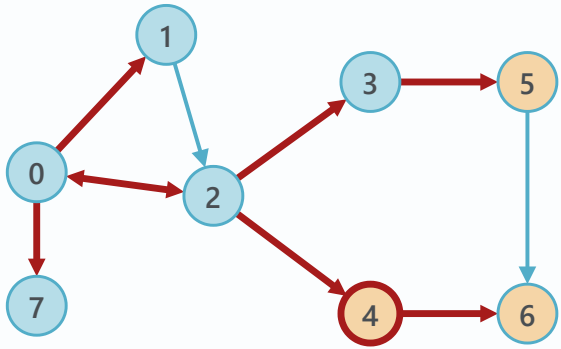
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
6      int v = kuyruk.poll();
7      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 4` `kuyruk = [5, 6]`

ÇIKTI

```
0 1 2 7 3 4
```



kuyruk (ön → arka)

5	6

4'ün komşuları: keşfedilmemiş komşu 6 işaretlenip kuyruğun **arkasına** eklenir.

BFS: `bfs(0)` İzleme

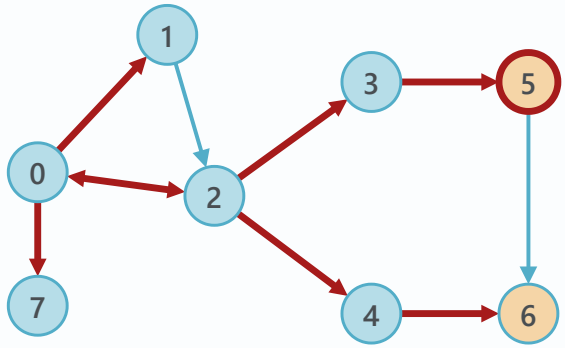
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
8      for (int w : komsu[v])
9        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 5` `kuyruk = [6]`

ÇIKTI

```
0 1 2 7 3 4 5
```



kuyruk (ön → arka)

6

Kuyruğun önünden **5** alınır ve yazdırılır.

BFS: `bfs(0)` İzleme

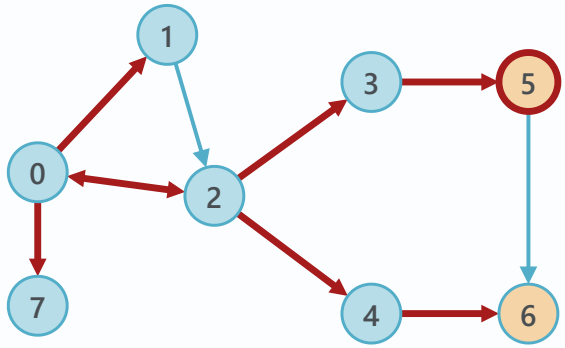
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
5    while (!kuyruk.isEmpty()) {
6      int v = kuyruk.poll();
7      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (!ziyaret[w])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

DEĞİŞKENLER

`v = 5` `kuyruk = [6]`

ÇIKTI

```
0 1 2 7 3 4 5
```



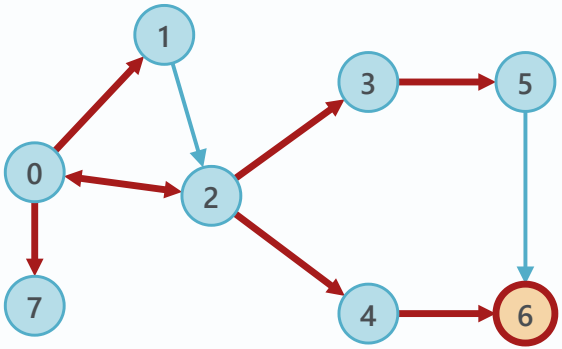
kuyruk (ön → arka)

6

5'in komşuları: 6 zaten işaretli → atlanır.

BFS: `bfs(0)` İzleme

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



kuyruk (ön → arka)

Kuyruğun önünden **6** alınır ve yazdırılır.

Adım 15 / 16

DEĞİŞKENLER

`v = 6` `kuyruk = []`

ÇIKTI

```
0 1 2 7 3 4 5 6
```

BFS: `bfs(0)` İzleme

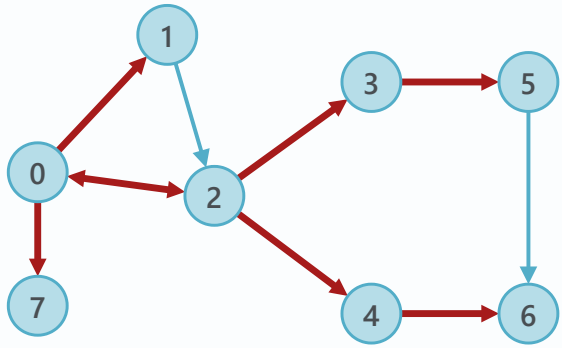
```
1  void bfs(int s) {
2    boolean[] ziyaret = new boolean[n];
3    Queue<Integer> kuyruk = new LinkedList<>();
4    ziyaret[s] = true; kuyruk.add(s);
▶    while (!kuyruk.isEmpty()) {
6      int v = kuyruk.poll();
7      System.out.print(v + " ");
8      for (int w : komsu[v])
9        if (!ziyaret[w])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

DEĞİŞKENLER

`v = –` `kuyruk = []`

ÇIKTI

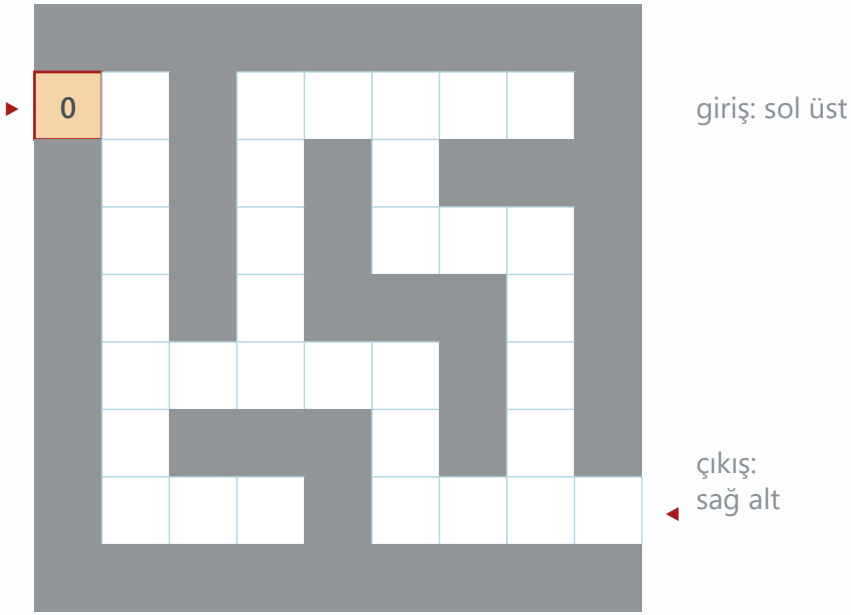
```
0 1 2 7 3 4 5 6
```



kuyruk (ön → arka)

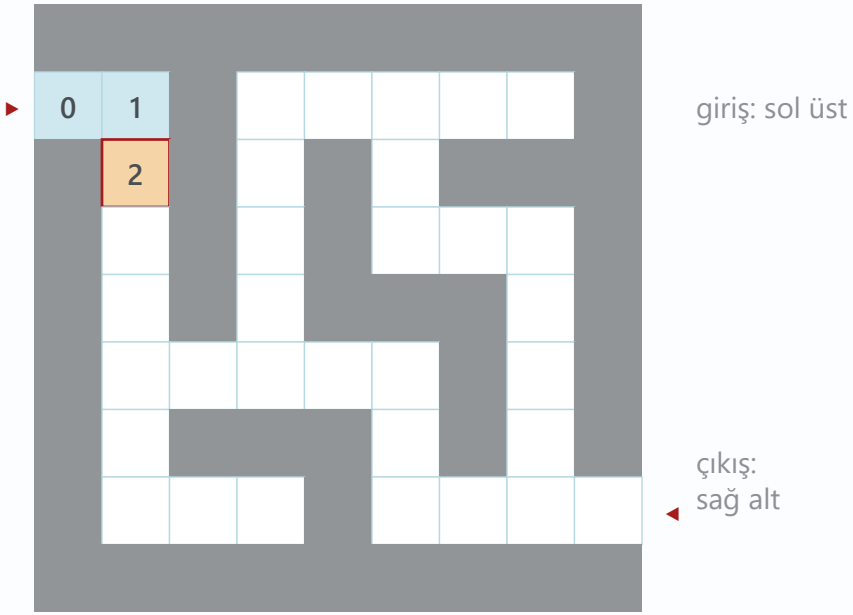
Kuyruk boş. Çıktı: **0 1 2 7 3 4 5 6**, katman katman: {0}, {1, 2, 7}, {3, 4}, {5, 6}. Kalın kenarlar BFS ağacıdır.

BFS ile Labirent: Tüm Yolları Eşzamanlı Dene



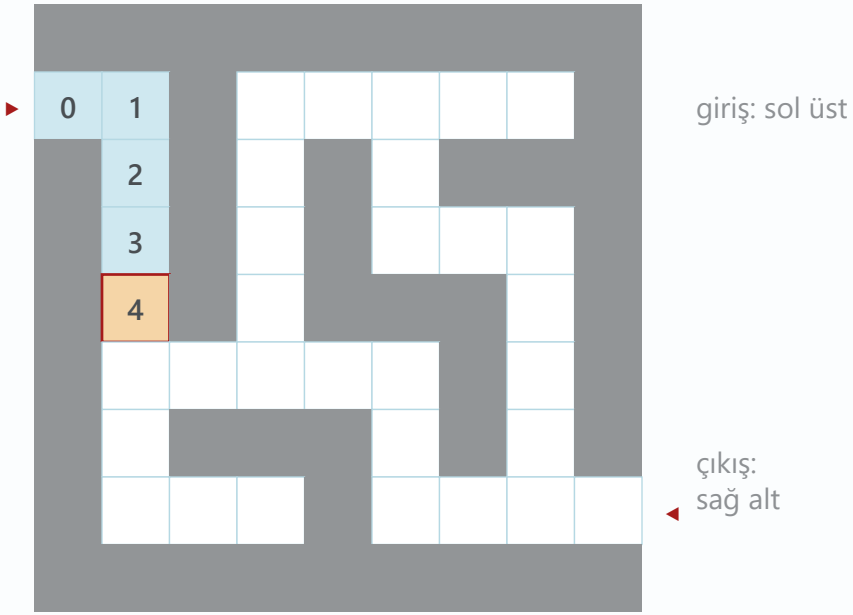
Giriş hücresi uzaklık 0 ile kuyruğa konur. Duvarlar (koyu) çizgede düğüm değildir; her boş hücre bir düğüm, komşu boş hücreler kenardır.

BFS ile Labirent: Tüm Yolları Eşzamanlı Dene



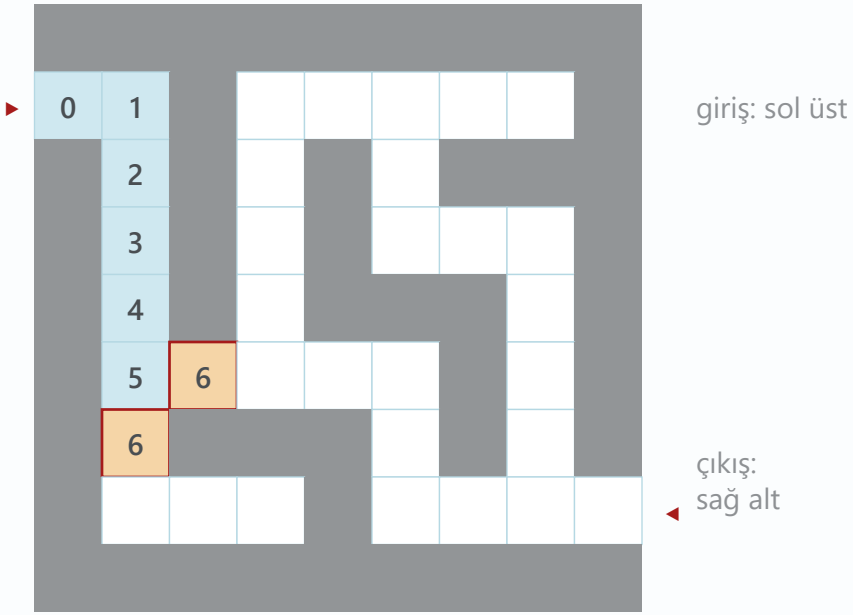
Dalga her adımda tüm yönlere bir hücre ilerler: uzaklığı 1 ve 2 olan hücreler. Sarı = şu anki **sınır** (frontier).

BFS ile Labirent: Tüm Yolları Eşzamanlı Dene



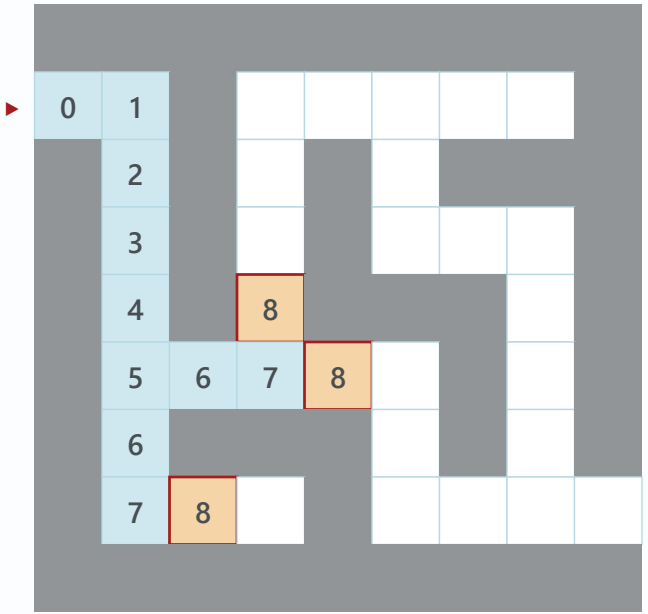
Tek koridor: aşağı inilir. Her hücreye yazılan sayı, girişten oraya **en az adım** sayısıdır.

BFS ile Labirent: Tüm Yolları Eşzamanlı Dene



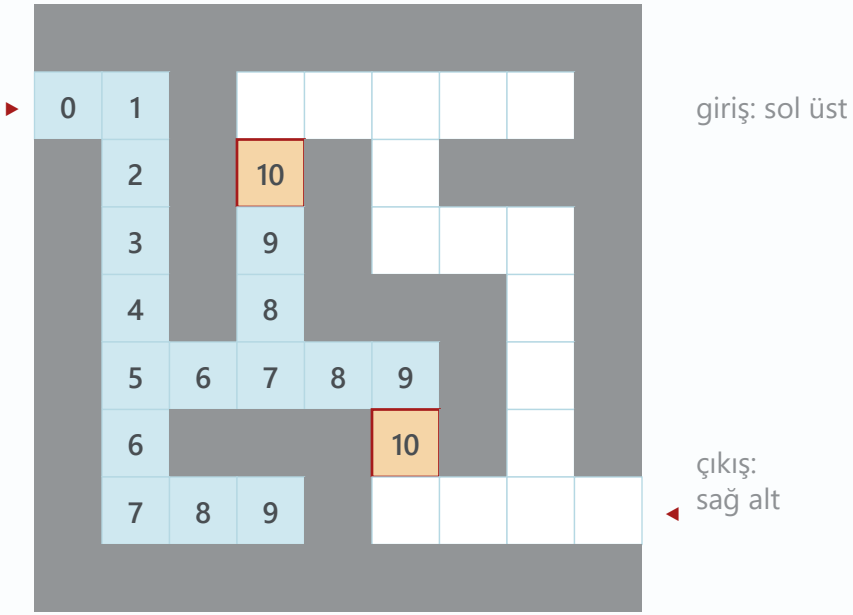
Uzaklık 5'te yol **çatallanır**: dalga hem sağa hem aşağı yayılır. BFS "var olan tüm yolları eşzamanlı dener".

BFS ile Labirent: Tüm Yolları Eşzamanlı Dene



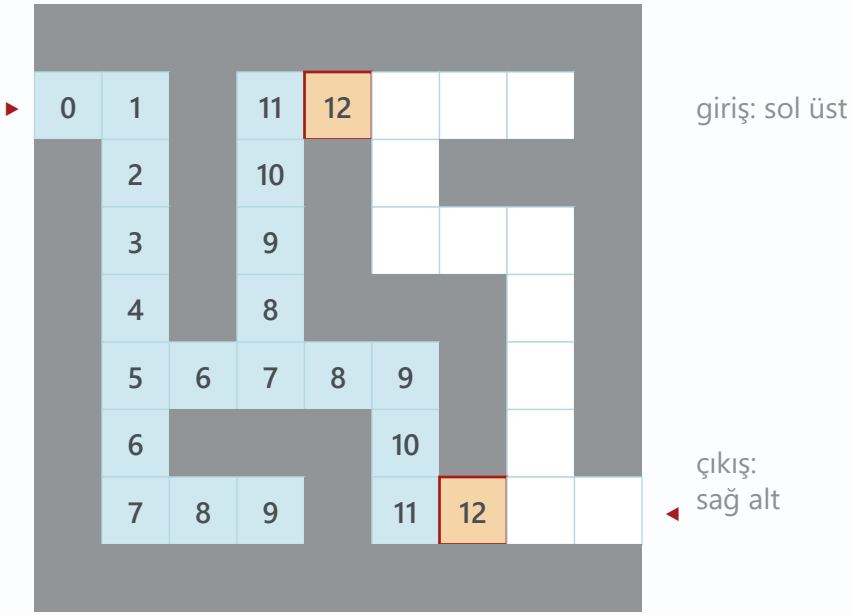
İki kol aynı anda ilerler: biri yukarıdaki odaya, biri aşağıya.

BFS ile Labirent: Tüm Yolları Eşzamanlı Dene



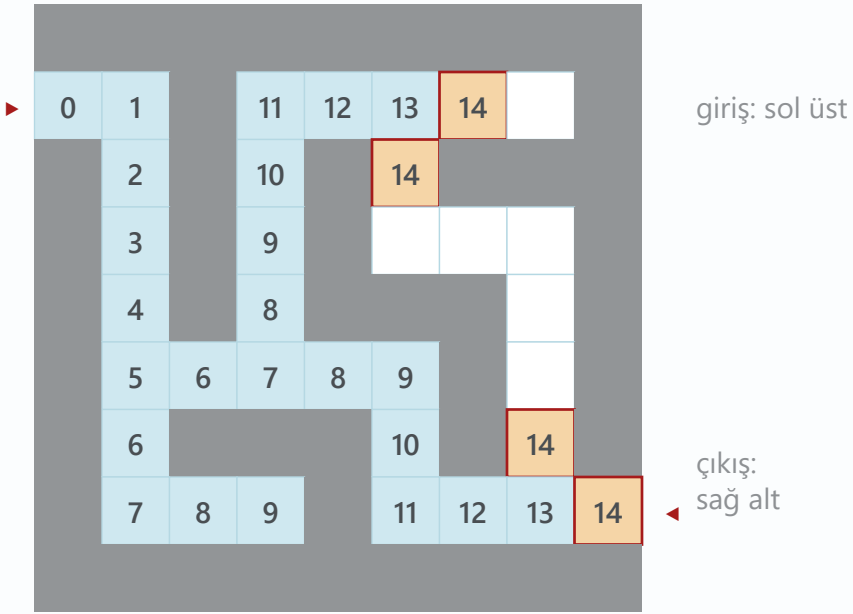
Uzaklık 10: dalga yukarıdaki koridora ve aşağıda alt koridora ulaştı.

BFS ile Labirent: Tüm Yolları Eşzamanlı Dene



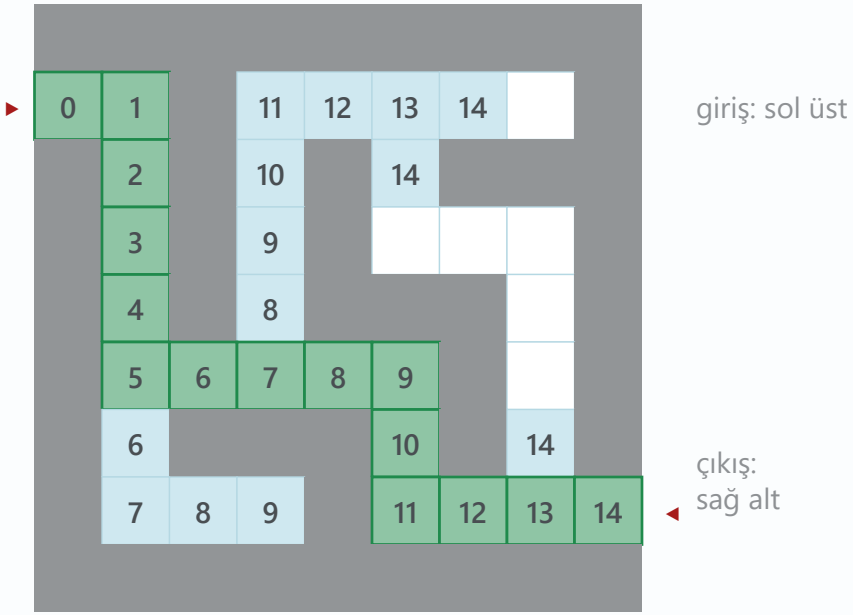
Alt koridor çıkışa yaklaşıyor; üst kol çıkmaz sokaklarda dolaşıyor.

BFS ile Labirent: Tüm Yolları Eşzamanlı Dene



Çıkış hücresine **14** adımda ulaşıldı. BFS ilk ulaştığı anda bu uzaklık kesin olarak en kısadır.

BFS ile Labirent: Tüm Yolları Eşzamanlı Dene



Ebeveyn (parent) bağlantıları çıkıştan geriye izlenerek **en kısa yol** (yeşil, 15 hücre, 14 adım) elde edilir. Kalan hücreler (uzaklık 15–17) bu yol için gereksizdi.

BFS Ağacı ve En Kısa Yol

düğüm	1	2	3	4	5	6	7	8
uzaklık	0	1	1	1	1	2	2	2
ebeveyn	—	1	1	1	1	2	2	3

"BFS Adım Adım" çizgesi, kaynak 1. Yol(8): 8 → ebeveyn 3 → ebeveyn 1, ters çevrilince **1** → **3** → **8** (2 kenar).

- Keşifte: `uzaklik[w] = uzaklik[v] + 1; ata[w] = v;`
- Kuyruktan çıkan uzaklıklar **azalmayan** sıradadır (0, 1, 1, 1, 1, 2, 2, 2); bu yüzden bir düğüme ilk ulaşan yol en kısadır.

Yolu geri kurma

```
List<Integer> yol(int[] ata, int s, int t) {
  LinkedList<Integer> yol = new LinkedList<>();
  for (int v = t; v != -1; v = ata[v])
    yol.addFirst(v);   // hedeften geriye
  if (yol.getFirst() != s)
    return null;       // t'ye ulaşılamadı
  return yol;
}
```

Yalnızca **ağırlıksız** (ya da tüm ağırlıkları eşit) çizgede geçerlidir. Ağırlıklı çizgede en kısa yol için Dijkstra gerekir.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

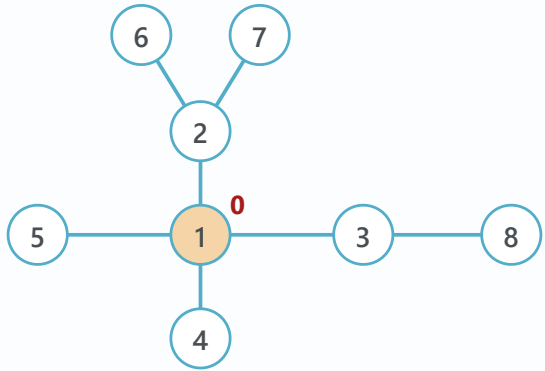
```
1  void bfsUzaklik(int s) {
▶    Arrays.fill(uzaklik, -1);
▶    Arrays.fill(ata, -1); uzaklik[s] = 0;
▶    Queue<Integer> kuyruk = new LinkedList<>();
▶    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
7      int v = kuyruk.poll();
8      for (int w : komsu[v])
9        if (uzaklik[w] == -1) {  // yeni
10         uzaklik[w] = uzaklik[v] + 1;
11         ata[w] = v;
12         kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = –` `kuyruk = [1]`

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

`ata = [-1, -1, -1, -1, -1, -1, -1, -1]`





Yönsüz çizge, kaynak 1 (diziler 1–8 indisleriyle gösteriliyor). uzaklik[] = −1 "keşfedilmedi" demektir; ata[] da −1 ile doldurulur (ebeveyn yok); 1'in uzaklığı 0, kuyrukta yalnızca 1 var.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

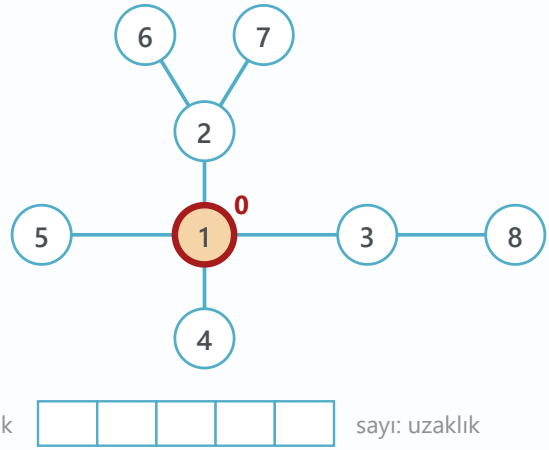
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
8      for (int w : komsu[v])
9        if (uzaklik[w] == -1) {  // yeni
10         uzaklik[w] = uzaklik[v] + 1;
11         ata[w] = v;
12         kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = 1` `kuyruk = []`

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

`ata = [-1, -1, -1, -1, -1, -1, -1, -1]`



Kuyruğun başındaki 1 çıkarıldı; şimdi komşuları 2, 3, 4, 5 sırayla incelenecek.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

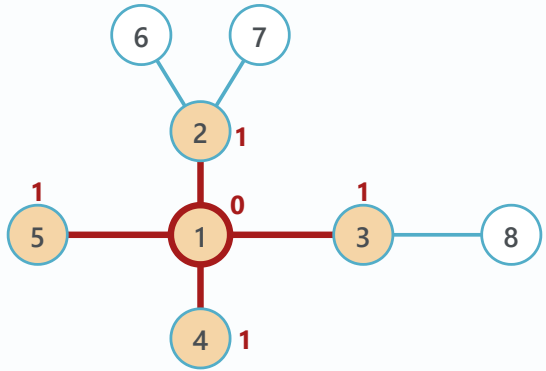
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
7      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (uzaklik[w] == -1) {  // yeni
▶          uzaklik[w] = uzaklik[v] + 1;
▶          ata[w] = v;
▶          kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = 1` `kuyruk = [2, 3, 4, 5]`

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

`ata = [-1, 1, 1, 1, 1, -1, -1, -1]`



kuyruk | 2 | 3 | 4 | 5 | | sayı: uzaklık

Dört komşu da keşfedilmemiş: uzaklıkları uzaklik[1] + 1 = 1, ebeveynleri 1 olur ve kuyruğa eklenirler.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

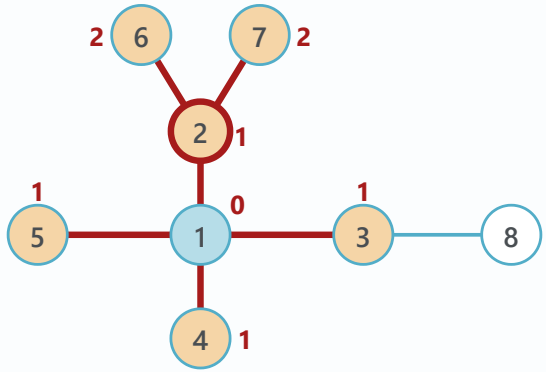
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (uzaklik[w] == -1) {  // yeni
▶          uzaklik[w] = uzaklik[v] + 1;
▶          ata[w] = v;
▶          kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = 2` `kuyruk = [3, 4, 5, 6, 7]`

`uzaklik = [0, 1, 1, 1, 1, 2, 2, -1]`

`ata = [-1, 1, 1, 1, 1, 2, 2, -1]`



kuyruk | 3 | 4 | 5 | 6 | 7 | sayı: uzaklık

2 çıkarıldı. Komşu 1 zaten keşfedilmiş, atlanır; 6 ve 7 keşfedilir: uzaklık 1 + 1 = 2, ebeveyn 2.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

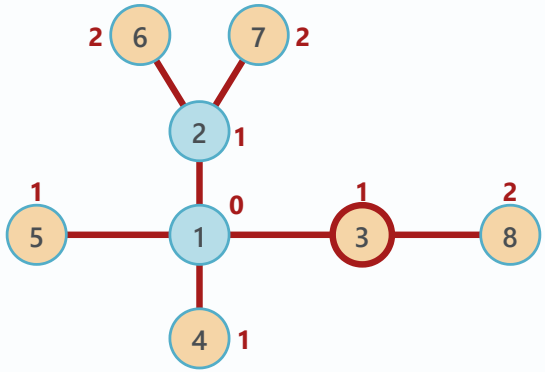
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (uzaklik[w] == -1) {  // yeni
▶          uzaklik[w] = uzaklik[v] + 1;
▶          ata[w] = v;
▶          kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = 3` `kuyruk = [4, 5, 6, 7, 8]`

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

`ata = [-1, 1, 1, 1, 1, 2, 2, 3]`



kuyruk | 4 | 5 | 6 | 7 | 8 | sayı: uzaklık

3 çıkarıldı. Komşu 1 zaten keşfedilmiş, atlanır; 8 keşfedilir: uzaklık 1 + 1 = 2, ebeveyn 3.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

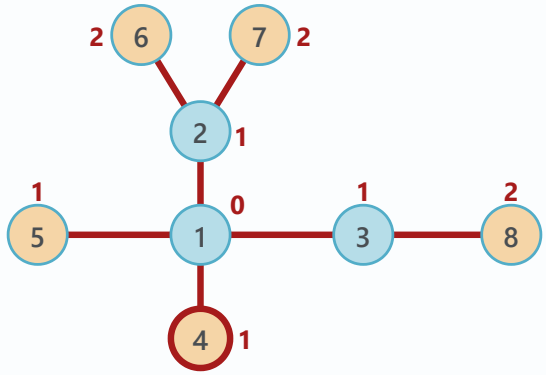
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (uzaklik[w] == -1) {  // yeni
10         uzaklik[w] = uzaklik[v] + 1;
11         ata[w] = v;
12         kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = 4` `kuyruk = [5, 6, 7, 8]`

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

`ata = [-1, 1, 1, 1, 1, 2, 2, 3]`



kuyruk | 5 | 6 | 7 | 8 | | sayı: uzaklık

4 çıkarıldı (uzaklık 1). Tek komşusu 1 zaten keşfedilmiş: kuyruğa yeni düğüm girmez.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

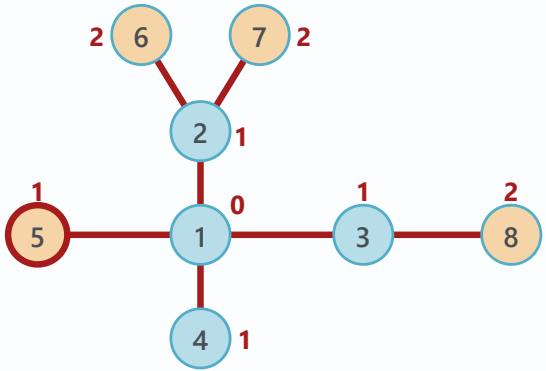
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (uzaklik[w] == -1) {  // yeni
10         uzaklik[w] = uzaklik[v] + 1;
11         ata[w] = v;
12         kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = 5` `kuyruk = [6, 7, 8]`

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

`ata = [-1, 1, 1, 1, 1, 2, 2, 3]`



kuyruk | 6 | 7 | 8 | | | sayı: uzaklık

5 çıkarıldı (uzaklık 1). Tek komşusu 1 zaten keşfedilmiş: kuyruğa yeni düğüm girmez.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

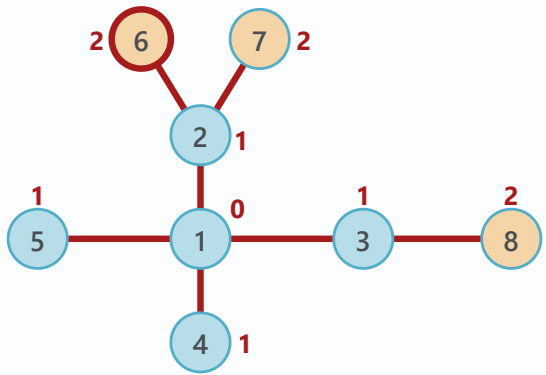
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (uzaklik[w] == -1) {  // yeni
10         uzaklik[w] = uzaklik[v] + 1;
11         ata[w] = v;
12         kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = 6` `kuyruk = [7, 8]`

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

`ata = [-1, 1, 1, 1, 1, 2, 2, 3]`



kuyruk | 7 | 8 | | | | sayı: uzaklık

6 çıkarıldı (uzaklık 2). Tek komşusu 2 zaten keşfedilmiş: kuyruğa yeni düğüm girmez.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

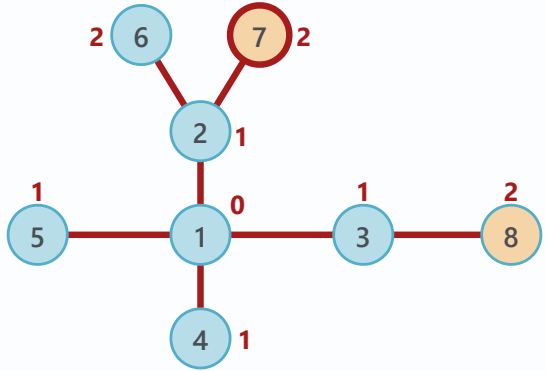
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (uzaklik[w] == -1) {  // yeni
10         uzaklik[w] = uzaklik[v] + 1;
11         ata[w] = v;
12         kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = 7` `kuyruk = [8]`

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

`ata = [-1, 1, 1, 1, 1, 2, 2, 3]`



kuyruk | 8 | | | | | sayı: uzaklık

7 çıkarıldı (uzaklık 2). Tek komşusu 2 zaten keşfedilmiş: kuyruğa yeni düğüm girmez.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

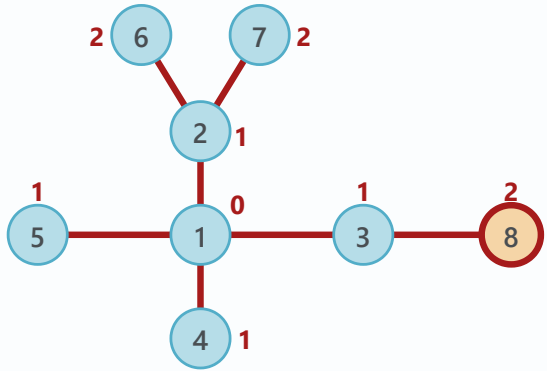
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
6    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (uzaklik[w] == -1) {  // yeni
10         uzaklik[w] = uzaklik[v] + 1;
11         ata[w] = v;
12         kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = 8` `kuyruk = []`

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

`ata = [-1, 1, 1, 1, 1, 2, 2, 3]`



kuyruk | | | | | | sayı: uzaklık

8 çıkarıldı (uzaklık 2). Tek komşusu 3 zaten keşfedilmiş: kuyruğa yeni düğüm girmez.

BFS: `uzaklik[]` ve `ata[]` Dizilerini Doldurmak

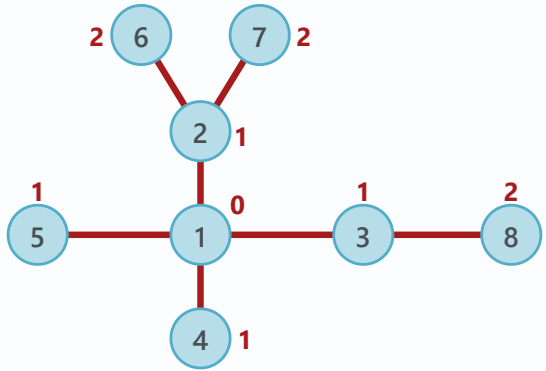
```
1  void bfsUzaklik(int s) {
2    Arrays.fill(uzaklik, -1);
3    Arrays.fill(ata, -1); uzaklik[s] = 0;
4    Queue<Integer> kuyruk = new LinkedList<>();
5    kuyruk.add(s);
▶    while (!kuyruk.isEmpty()) {
7      int v = kuyruk.poll();
8      for (int w : komsu[v])
9        if (uzaklik[w] == -1) {  // yeni
10         uzaklik[w] = uzaklik[v] + 1;
11         ata[w] = v;
12         kuyruk.add(w);
13       }
14   }
15 }
```

DEĞİŞKENLER

`v = –` `kuyruk = []`

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

`ata = [-1, 1, 1, 1, 1, 2, 2, 3]`



kuyruk | | | | | | sayı: uzaklık

Kuyruk boşaldı. Kalın kenarlar (ebeveyn bağlantıları) BFS ağacını oluşturur; ör. 8 → ebeveyn 3 → ebeveyn 1: en kısa yol 1 → 3 → 8, uzunluğu uzaklik[8] = 2.

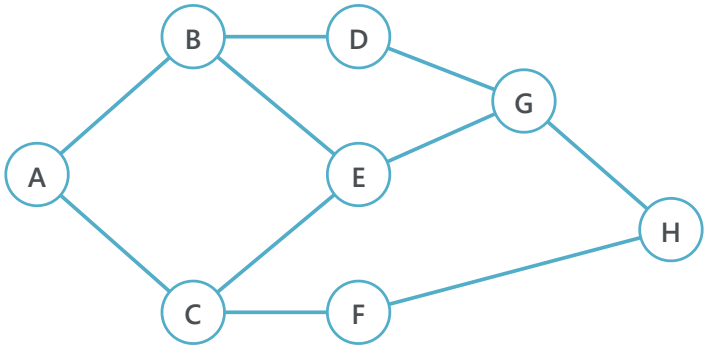
BFS ve DFS Karşılaştırması

	DFS	BFS
Yardımcı yapı	Yığın (LIFO) / özyineleme	Kuyruk (FIFO)
Gezinme biçimi	bir dalı sonuna kadar, sonra geri dönüş	katman katman (uzaklık sırasıyla)
Zaman	$O(V + E)$ liste · $O(V^2)$ matris	$O(V + E)$ liste · $O(V^2)$ matris
Ek bellek	$O(V)$: en uzun yol yığında	$O(V)$: en geniş katman kuyrukta
En kısa yol	hayır	evet (ağırlıksız çizgede)
Ürettiği ağaç	derin, ince DFS ağacı	sığ, geniş BFS ağacı
Tipik kullanım	döngü, bileşen, topolojik sıra, SCC, geri izleme	en az adım, düzey, iki parçalılık testi, yayılma

Aynı çizgede (8 düğümlü yönlü çizge, kaynak 0): DFS **0 1 2 3 5 6 4 7**, BFS **0 1 2 7 3 4 5 6**.

Deneyin: BFS ve DFS

Başlangıç A **BFS** **DFS** **adım** **sonuna git** **sıfırla**



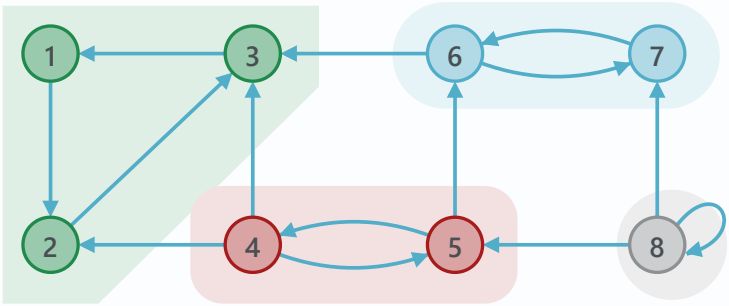
Bir başlangıç düğümü seçip **BFS** ya da **DFS**'e basın, sonra **adım** ile ilerleyin.

Yönsüz çizge; komşular alfabetik sırayla gezilir. DFS özyinelemeli sürümün sırasını verir (yığına komşular ters sırada itilir). Sarı: kuyrukta / yığında · turuncu çerçeve: şu an işlenen · koyu: bitti.

TARJAN SCC

Güçlü Bağlantılı Bileşenler

- Yönlü çizgede u ve v **karşılıklı ulaşılabilir**: u'dan v'ye **ve** v'den u'ya yol var.
- **Güçlü bağlantılı bileşen** (strongly connected component, SCC): her düğümün diğer her düğüme ulaşabildiği **maksimal** düğüm kümesi.
- Her düğüm tam bir SCC'ye aittir; tek düğüm de bir SCC olabilir ({8}).
- "Bir düğüm ve ondan erişilebilen tüm düğümler" SCC **değildir**: 8'den herkese ulaşılır ama kimse 8'e ulaşamaz.
- Tarjan algoritması (1972) tüm SCC'leri **tek bir DFS** ile bulur.



SCC'ler: {1, 2, 3}, {4, 5}, {6, 7}, {8}

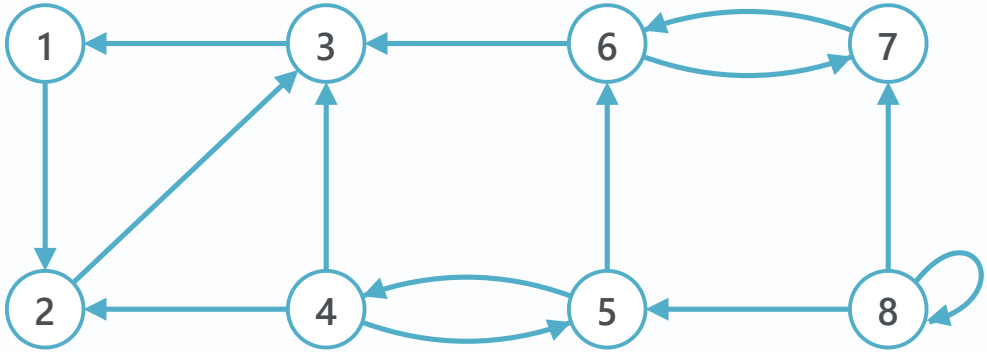
Her SCC tek düğüme indirgenirse elde edilen **bileşen çizgesi** döngüsüzdür (DAG): sosyal ağlarda topluluklar, derleyicilerde karşılıklı bağımlı modüller, 2-SAT çözümü.

Tarjan: İlkeler ve Adımlar

1. **DFS tabanlıdır**: çizgenin tüm düğümleri sırayla denenir; ziyaret edilmemiş her düğümden DFS başlatılır.
2. DFS sırasında ziyaret edilen her düğüme bir **indeks** atanır: düğümün **keşif sırası** (1, 2, 3, …). Mevcut DFS yolundaki ve SCC'si henüz belirlenmemiş düğümler bir **yığında** tutulur.
3. Her düğüm için **enDüşükBağ** (low-link) hesaplanır: düğümün DFS alt ağacından, en fazla bir kenarla **hâlâ yığında olan** düğümlere ulaşarak erişilebilen en küçük indeks.
   - Ağaç kenarı v → w: dönüşte `enDüşükBağ(v) = min(enDüşükBağ(v), enDüşükBağ(w))`
   - Yığındaki w'ye kenar: `enDüşükBağ(v) = min(enDüşükBağ(v), indeks(w))`
   - Bitmiş bir SCC'deki w'ye kenar: yok sayılır.
4. v'nin işi bitince **enDüşükBağ(v) = indeks(v)** ise v bir SCC'nin **kökü**dür: yığından v'ye kadar (v dahil) çekilen düğümler bir SCC oluşturur.

enDüşükBağ(v) < indeks(v) ise v'nin alt ağacından daha eski bir yığın düğümüne geri dönülebiliyordur; v o düğümle aynı SCC'dedir, kök değildir.

Tarjan Adım Adım: indeks, enDüşükBağ

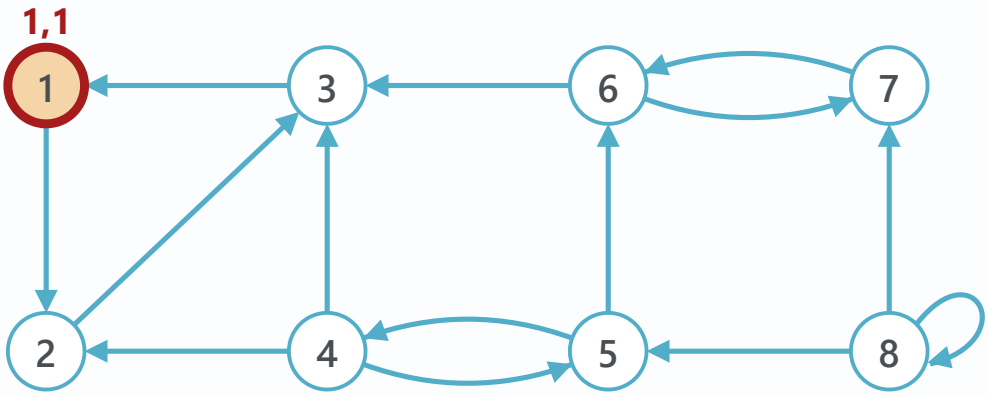


yığın

SCC'ler: —

Yönlü çizge, 8 düğüm. Her düğümün yanına **indeks,enDüşükBağ** yazılacak. Beyaz = ziyaret edilmedi, sarı = yığında, renkli = SCC'si belirlendi.

Tarjan Adım Adım: indeks, enDüşükBağ

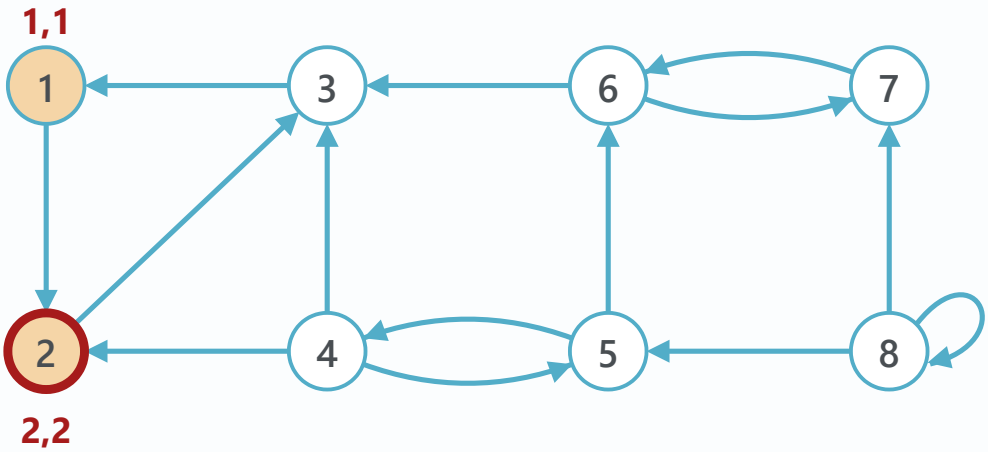


yığın 1

SCC'ler: —

`SCC(1)` : indeks = enDüşükBağ = **1**; 1 yığına itilir.

Tarjan Adım Adım: indeks, enDüşükBağ

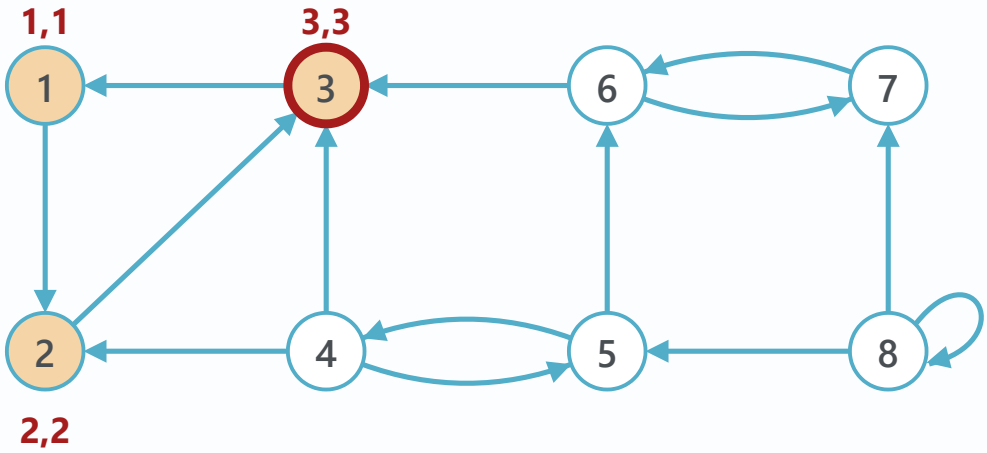


yığın | 1 | 2 |

SCC'ler: —

`SCC(2)` : indeks = enDüşükBağ = **2**; 2 yığına itilir.

Tarjan Adım Adım: indeks, enDüşükBağ

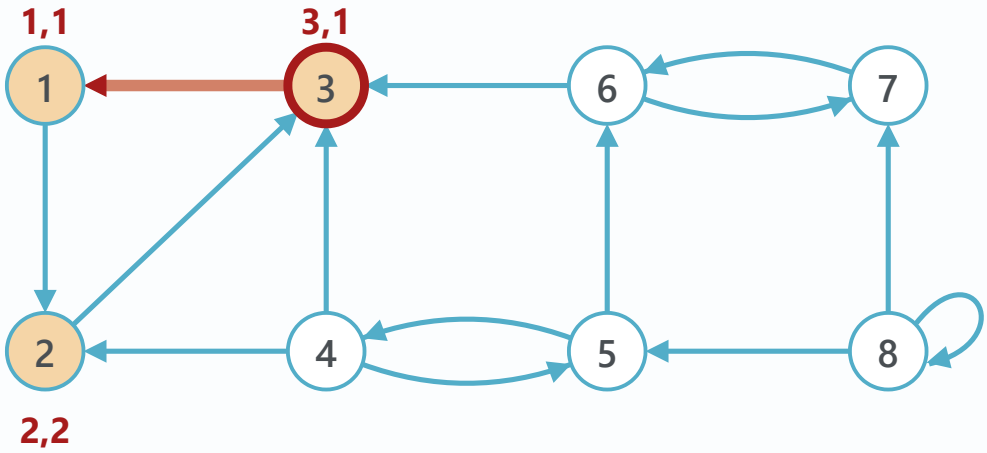


yığın | 1 | 2 | 3 |

SCC'ler: —

`SCC(3)` : indeks = enDüşükBağ = **3**; 3 yığına itilir.

Tarjan Adım Adım: indeks, enDüşükBağ

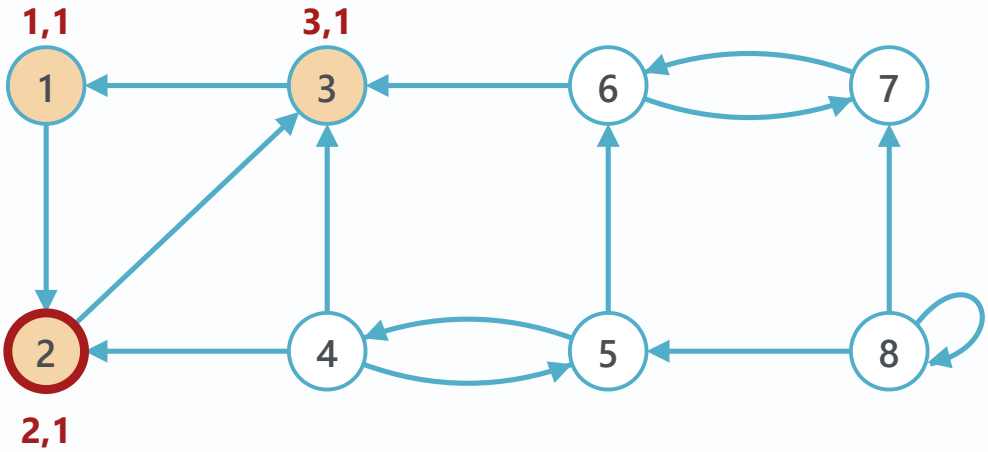


yığın | 1 | 2 | 3 |

SCC'ler: —

Kenar 3 → 1: 1 **yığında** → enDüşükBağ(3) = min(3, indeks(1) = 1) = **1**.

Tarjan Adım Adım: indeks, enDüşükBağ

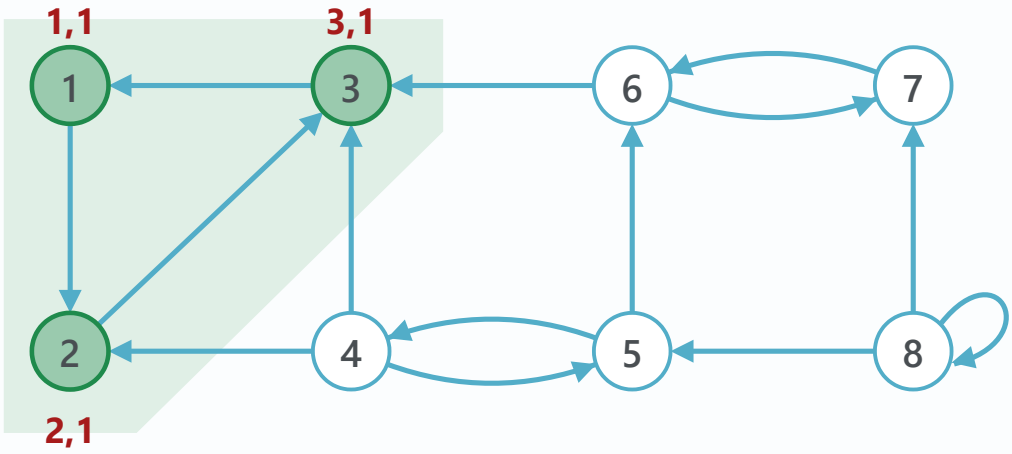


yığın | 1 | 2 | 3 |

SCC'ler: —

3 bitti; enDüşükBağ 1 < indeks 3 → kök değil, yığında kalır. 2'ye dönüldü: enDüşükBağ(2) = min(2, enDüşükBağ(3) = 1) = **1**.

Tarjan Adım Adım: indeks, enDüşükBağ

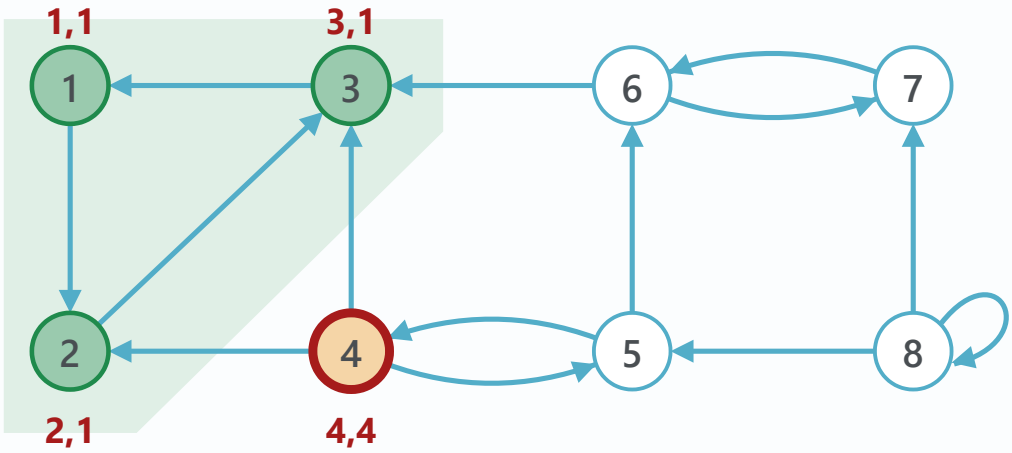


yığın

SCC'ler: {1, 2, 3}

2 bitti; enDüşükBağ 1 < indeks 2 → kök değil, yığında kalır. 1'e dönüldü; enDüşükBağ(1) = 1 değişmedi. 1'de enDüşükBağ = indeks = 1 → 1 **köktür**: yığından 1'e kadar çekilenler bir SCC: **{1, 2, 3}**.

Tarjan Adım Adım: indeks, enDüşükBağ

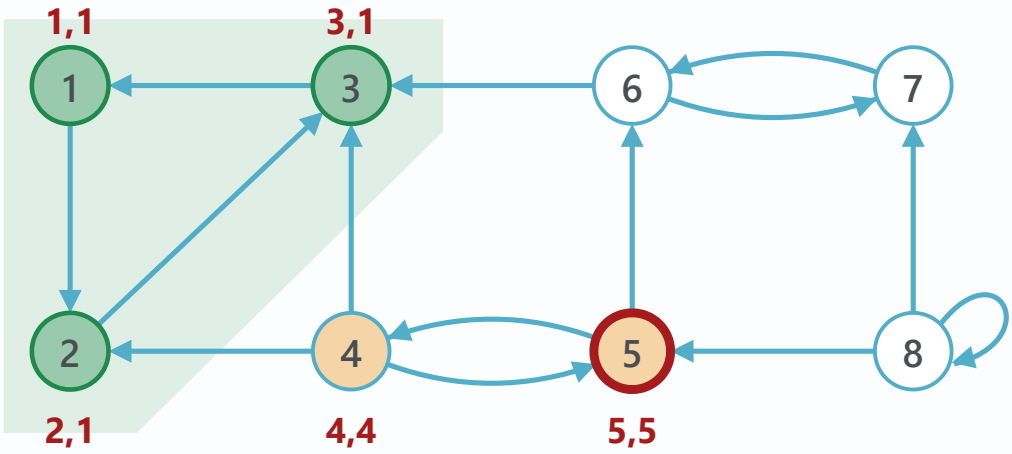


yığın [4]

SCC'ler: {1, 2, 3}

`SCC(4)` : indeks = enDüşükBağ = **4**; 4 yığına itilir.

Tarjan Adım Adım: indeks, enDüşükBağ

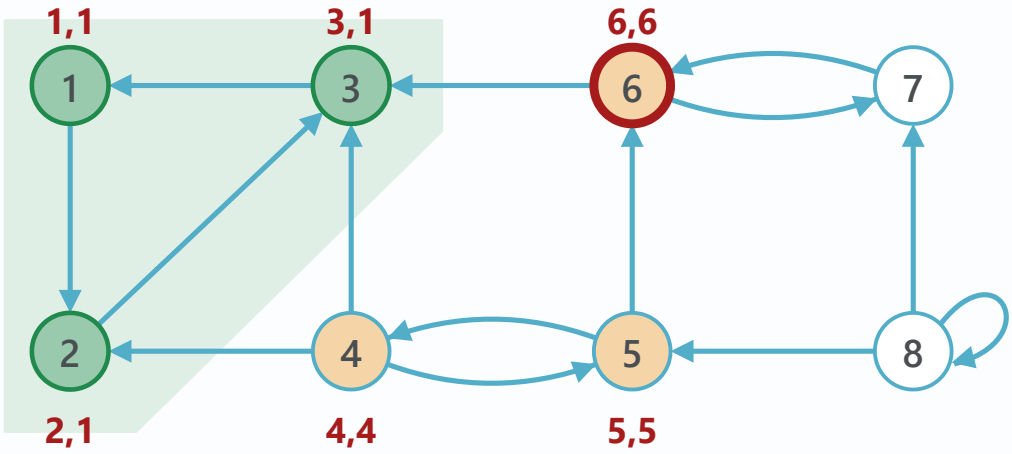


yığın | 4 | 5 |

SCC'ler: {1, 2, 3}

4 → 2 yok sayılır (2 bitmiş bir SCC'de, yığında değil). 4 → 3 yok sayılır (3 bitmiş bir SCC'de, yığında değil). `SCC(5)` : indeks = enDüşükBağ = **5**; 5 yığına itilir.

Tarjan Adım Adım: indeks, enDüşükBağ

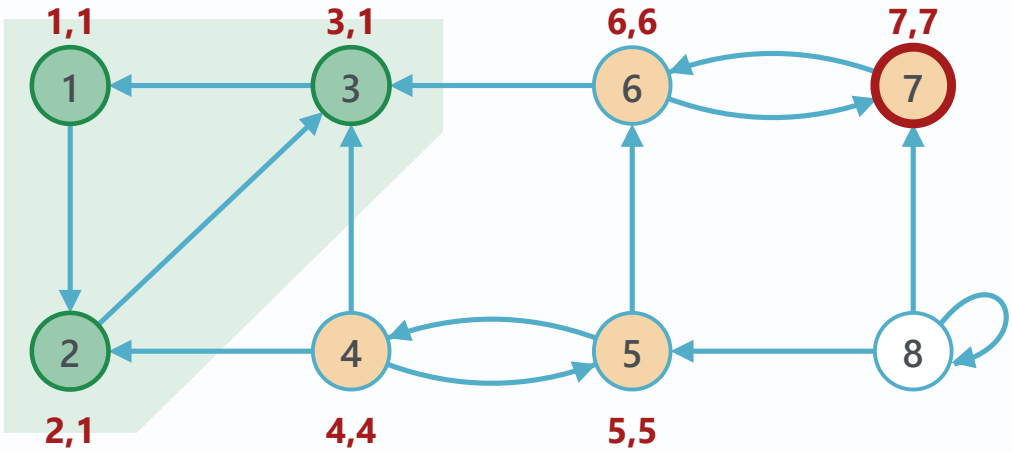


yığın | 4 | 5 | 6 |

SCC'ler: {1, 2, 3}

`SCC(6)` : indeks = enDüşükBağ = **6**; 6 yığına itilir.

Tarjan Adım Adım: indeks, enDüşükBağ

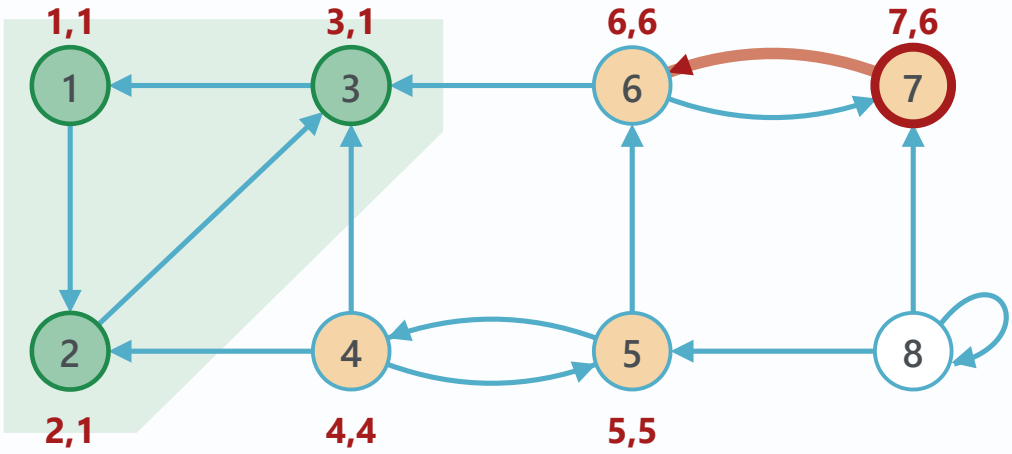


yığın | 4 | 5 | 6 | 7 |

SCC'ler: {1, 2, 3}

6 → 3 yok sayılır (3 bitmiş bir SCC'de, yığında değil). `SCC(7)` : indeks = enDüşükBağ = **7**; 7 yığına itilir.

Tarjan Adım Adım: indeks, enDüşükBağ

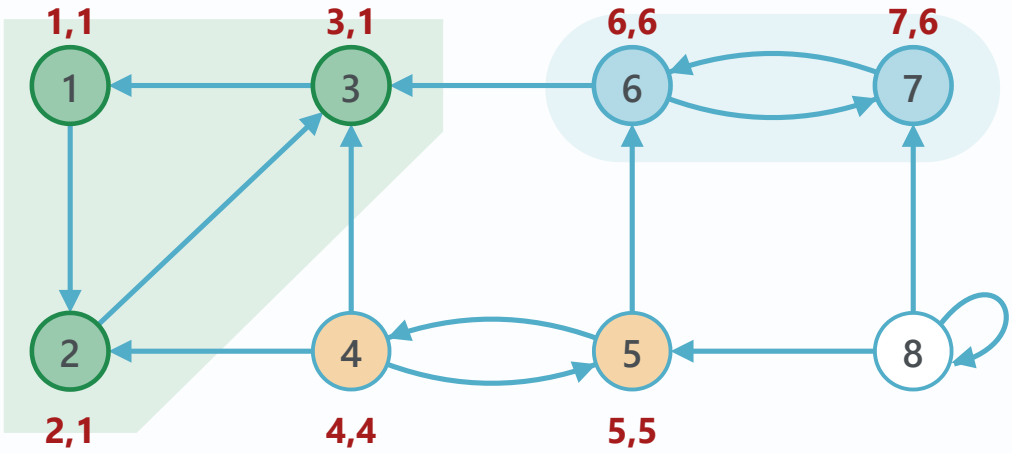


yığın | 4 | 5 | 6 | 7 |

SCC'ler: {1, 2, 3}

Kenar 7 → 6: 6 **yığında** → enDüşükBağ(7) = min(7, indeks(6) = 6) = **6**.

Tarjan Adım Adım: indeks, enDüşükBağ

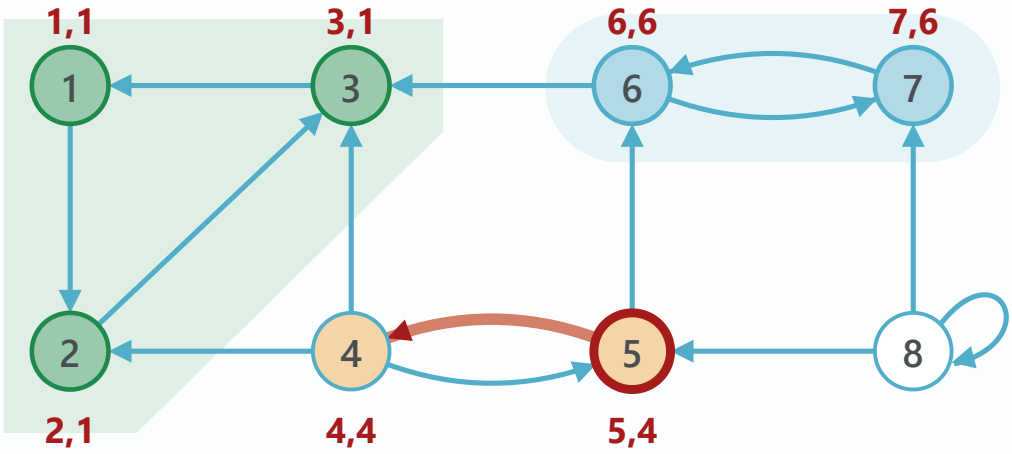


yığın | 4 | 5 |

SCC'ler: {1, 2, 3} · {6, 7}

7 bitti; enDüşükBağ 6 < indeks 7 → kök değil, yığında kalır. 6'ya dönüldü; enDüşükBağ(6) = 6 değişmedi. 6'da enDüşükBağ = indeks = 6 → 6 **köktür**: yığından 6'ya kadar çekilenler bir SCC: **{6, 7}**.

Tarjan Adım Adım: indeks, enDüşükBağ

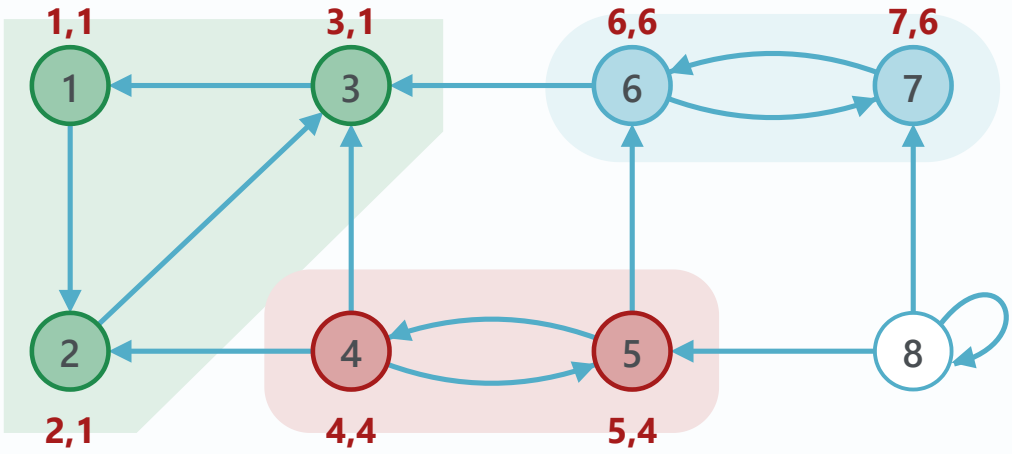


yığın | 4 | 5 |

SCC'ler: {1, 2, 3} · {6, 7}

5'e dönüldü; enDüşükBağ(5) = 5 değişmedi. Kenar 5 → 4: 4 **yığında** → enDüşükBağ(5) = min(5, indeks(4) = 4) = **4**.

Tarjan Adım Adım: indeks, enDüşükBağ

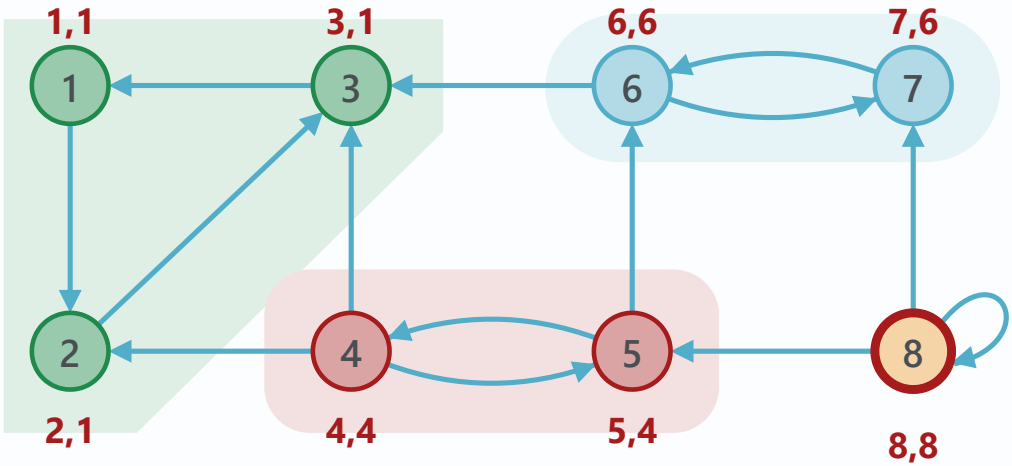


yığın

SCC'ler: {1, 2, 3} · {6, 7} · {4, 5}

5 bitti; enDüşükBağ 4 < indeks 5 → kök değil, yığında kalır. 4'e dönüldü; enDüşükBağ(4) = 4 değişmedi. 4'te enDüşükBağ = indeks = 4 → 4 **köktür**: yığından 4'e kadar çekilenler bir SCC: **{4, 5}**.

Tarjan Adım Adım: indeks, enDüşükBağ

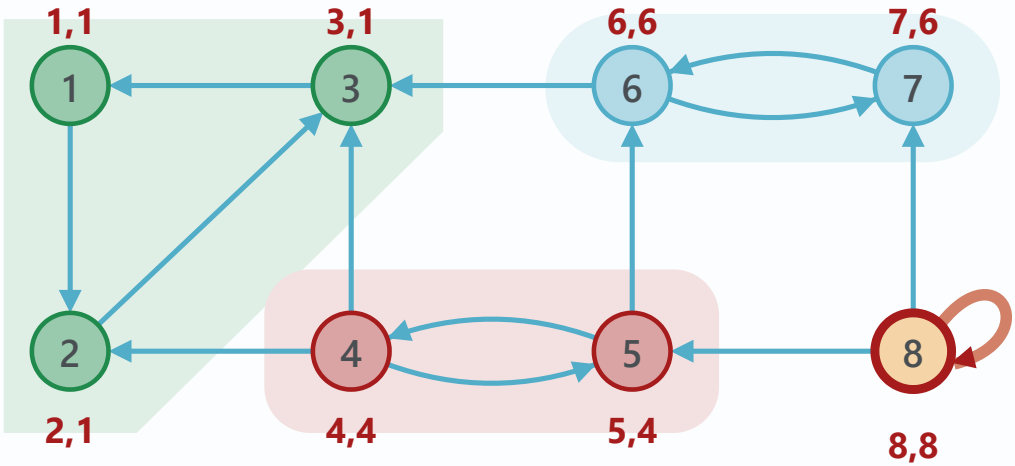


yığın 8

SCC'ler: {1, 2, 3} · {6, 7} · {4, 5}

`SCC(8)` : indeks = enDüşükBağ = **8**; 8 yığına itilir.

Tarjan Adım Adım: indeks, enDüşükBağ

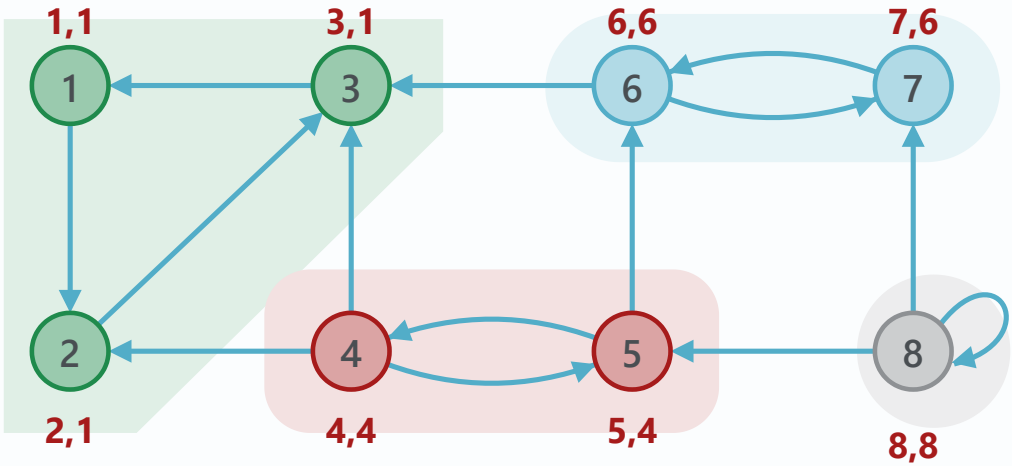


yığın 8

SCC'ler: {1, 2, 3} · {6, 7} · {4, 5}

8 → 5 yok sayılır (5 bitmiş bir SCC'de, yığında değil). 8 → 7 yok sayılır (7 bitmiş bir SCC'de, yığında değil). Öz döngü 8 → 8: 8 yığında ama enDüşükBağ = min(8, 8) = 8 değişmez.

Tarjan Adım Adım: indeks, enDüşükBağ

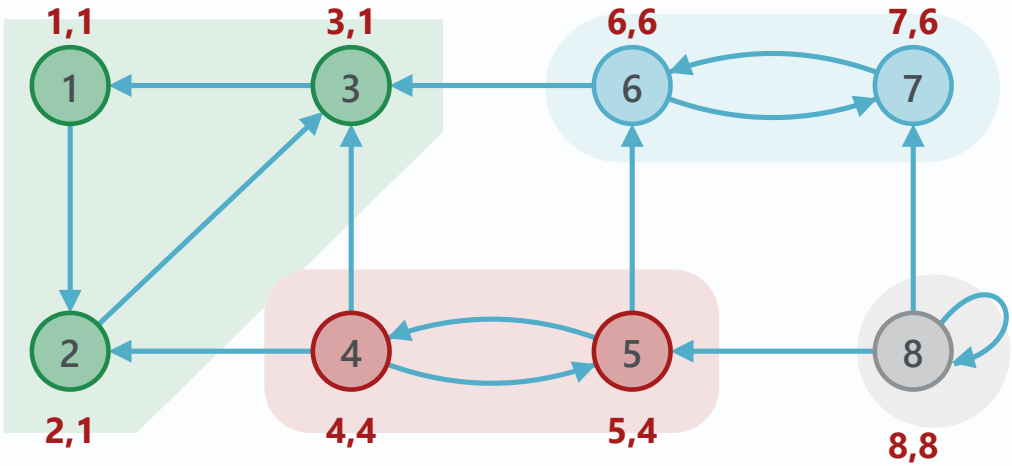


yığın

SCC'ler: {1, 2, 3} · {6, 7} · {4, 5} · {8}

8'de enDüşükBağ = indeks = 8 → 8 **köktür**: yığından 8'e kadar çekilenler bir SCC: **{8}**.

Tarjan Adım Adım: indeks, enDüşükBağ



yığın boş

SCC'ler: {1, 2, 3} · {6, 7} · {4, 5} · {8}

Sonuç: **4 SCC**. Her SCC, kökü (indeks = enDüşükBağ olan düğüm: 1, 6, 4, 8) bittiği anda bulundu; bulunma sırası ters topolojik sıradır.

Tarjan: Sözde Kod

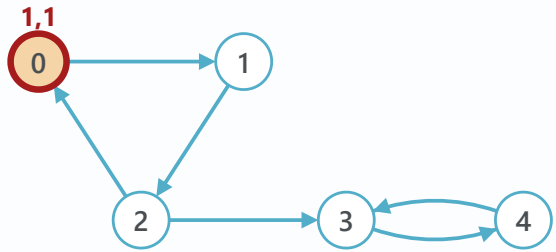
```
TARJAN_SCC(G):
  indeks = 1
  yığın = [ ]
  SCCler = [ ]
  her bir v için V içinde:
    eğer v.indeks tanımlı değilse:
      SCC(v)
  döndür SCCler
```

```
SCC(v):
  v.indeks = v.enDüşükBağ = indeks++
  yığın.ekle(v)
  her bir (v, w) için E içinde:
    eğer w.indeks tanımlı değilse:
      SCC(w)
      v.enDüşükBağ = min(v.enDüşükBağ, w.enDüşükBağ)
    değilse eğer w yığında ise:
      v.enDüşükBağ = min(v.enDüşükBağ, w.indeks)
  eğer v.enDüşükBağ == v.indeks:
    scc = [ ]
    döngü w ≠ v olduğu sürece:  // en az bir kez
      w = yığın.çıkar()
      scc.ekle(w)
    SCCler.ekle(scc)
```

- Görselde indeks 1'den başlatıldı (`indeks = 1`); 0'dan başlatmak da aynı SCC'leri verir.
- "yığında mı?" sorusu, yığına paralel bir `yiginda[]` dizisiyle O(1) zamanda yanıtlanır.

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
▶    indeks[v] = dusuk[v] = ++sayac;
▶    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın `0`

Çizge: 0→1, 1→2, 2→0, 2→3, 3→4, 4→3;
`indeks[w] == 0` = ziyaret edilmedi.
`scc(0)` : indeks ve dusuk = 1; 0 yığına itilir.

Adım 1 / 20

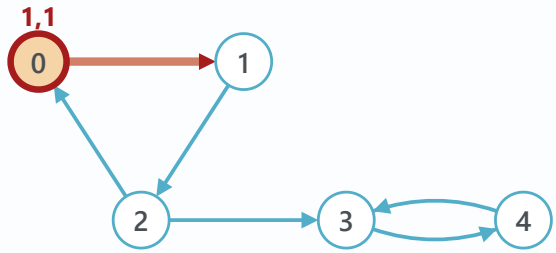
DEĞİŞKENLER

`v = 0` `w = –` `dusuk = [1, 0, 0, 0, 0]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
▶      if (indeks[w] == 0) {
▶        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın [0]

0 → 1: 1 ziyaret edilmemiş → özyinelemeli çağrı.

Adım 2 / 20

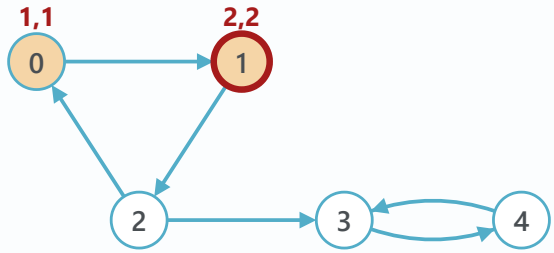
DEĞİŞKENLER

`v = 0` `w = 1` `dusuk = [1, 0, 0, 0, 0]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
▶    indeks[v] = dusuk[v] = ++sayac;
▶    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın | 0 | 1 |

`scc(1)` : indeks ve dusuk = 2; 1 yığına itilir.

Adım 3 / 20

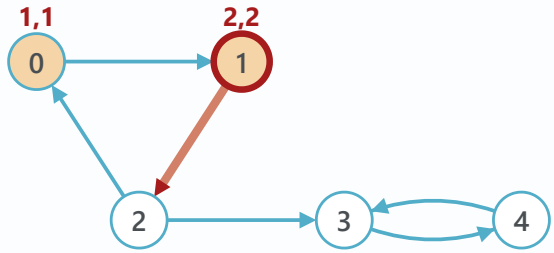
DEĞİŞKENLER

`v = 1` `w = –` `dusuk = [1, 2, 0, 0, 0]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
▶      if (indeks[w] == 0) {
▶        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın | 0 | 1 |

1 → 2: 2 ziyaret edilmemiş → özyinelemeli çağrı.

Adım 4 / 20

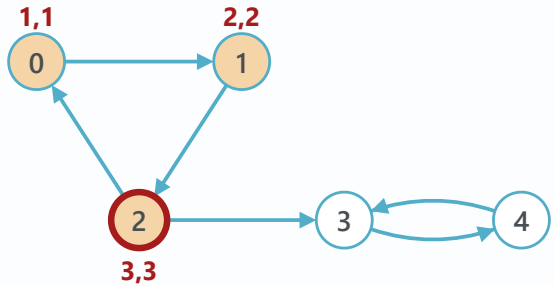
DEĞİŞKENLER

`v = 1` `w = 2` `dusuk = [1, 2, 0, 0, 0]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
▶    indeks[v] = dusuk[v] = ++sayac;
▶    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



`scc(2)` : indeks ve dusuk = 3; 2 yığına itilir.

Adım 5 / 20

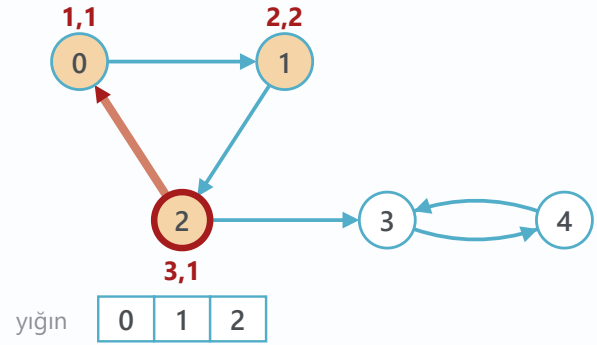
DEĞİŞKENLER

`v = 2` `w = –` `dusuk = [1, 2, 3, 0, 0]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
▶      } else if (yiginda[w])
▶        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



2 → 0: 0 **yığında** (geri kenar) → dusuk[2] = min(3, indeks[0] = 1) = **1**.

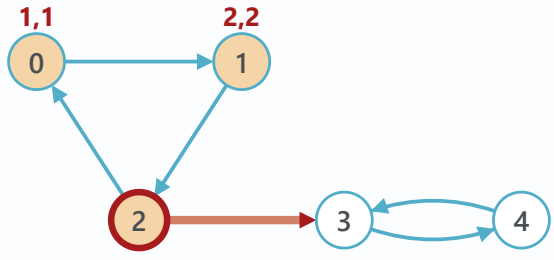
DEĞİŞKENLER

`v = 2` `w = 0` `dusuk = [1, 2, 1, 0, 0]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
▶      if (indeks[w] == 0) {
▶        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2

2 → 3: 3 ziyaret edilmemiş → özyinelemeli çağrı.

Adım 7 / 20

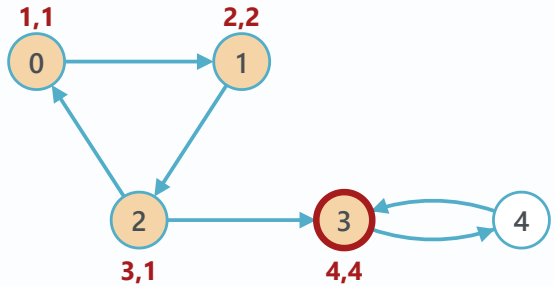
DEĞİŞKENLER

`v = 2` `w = 3` `dusuk = [1, 2, 1, 0, 0]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
▶    indeks[v] = dusuk[v] = ++sayac;
▶    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



`scc(3)` : indeks ve dusuk = 4; 3 yığına itilir.

Adım 8 / 20

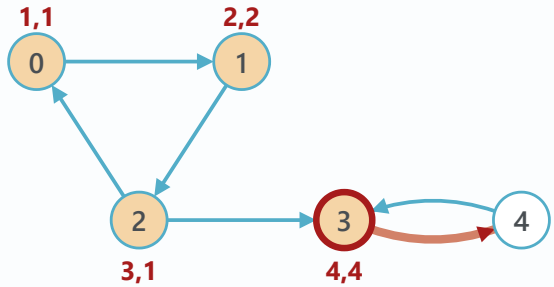
DEĞİŞKENLER

`v = 3` `w = –` `dusuk = [1, 2, 1, 4, 0]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
▶      if (indeks[w] == 0) {
▶        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



3 → 4: 4 ziyaret edilmemiş → özyinelemeli çağrı.

Adım 9 / 20

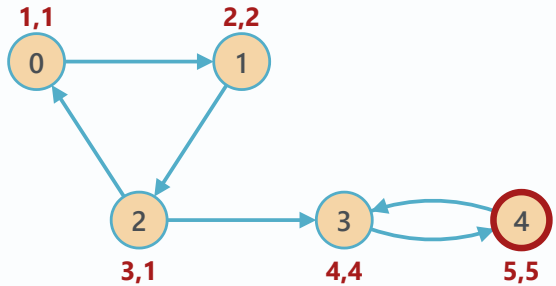
DEĞİŞKENLER

`v = 3` `w = 4` `dusuk = [1, 2, 1, 4, 0]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
▶    indeks[v] = dusuk[v] = ++sayac;
▶    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2	3	4

`scc(4)` : indeks ve dusuk = 5; 4 yığına itilir.

Adım 10 / 20

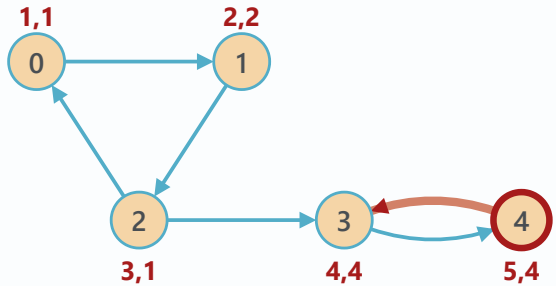
DEĞİŞKENLER

`v = 4` `w = –` `dusuk = [1, 2, 1, 4, 5]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
▶      } else if (yiginda[w])
▶        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2	3	4

4 → 3: 3 **yığında** (geri kenar) → dusuk[4] = min(5, indeks[3] = 4) = **4**.

Adım 11 / 20

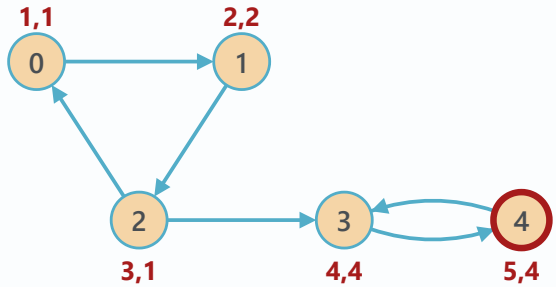
DEĞİŞKENLER

`v = 4` `w = 3` `dusuk = [1, 2, 1, 4, 4]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
▶    if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2	3	4

dusuk[4] = 4 ≠ indeks[4] = 5 → 4 kök değil, yığında kalır; çağrı döner.

Adım 12 / 20

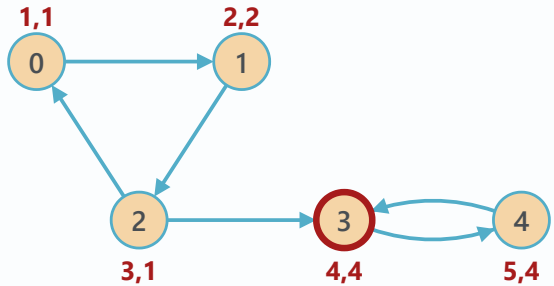
DEĞİŞKENLER

`v = 4` `w = –` `dusuk = [1, 2, 1, 4, 4]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
▶        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2	3	4

`scc(4)` döndü: dusuk[3] = min(4, dusuk[4] = 4) = **4**.

Adım 13 / 20

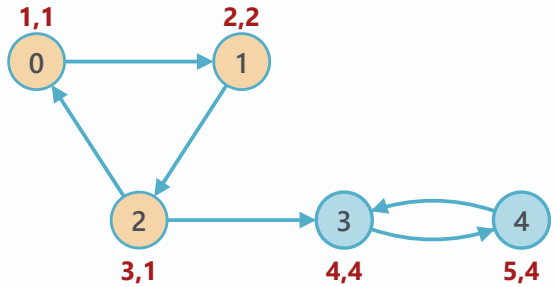
DEĞİŞKENLER

`v = 3` `w = 4` `dusuk = [1, 2, 1, 4, 4]`

ÇIKTI

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
▶    if (dusuk[v] == indeks[v]) {
▶      int w;
▶      do {
▶        w = yigin.pop(); yiginda[w] = false;
▶        System.out.print(w + " ");
▶      } while (w != v);
▶      System.out.println();
18   }
19 }
```



yığın

0	1	2

dusuk[3] == indeks[3] = 4 → 3 köktür; 4, 3 çekilip yazdırılır: SCC **{3, 4}**.

DEĞİŞKENLER

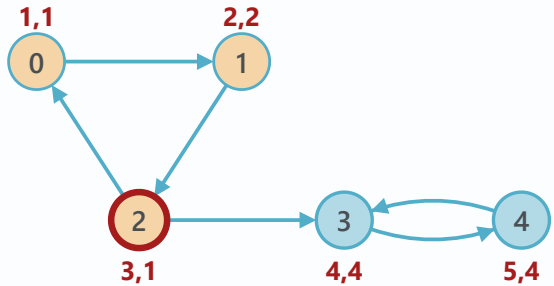
`v = 3` `w = –` `dusuk = [1, 2, 1, 4, 4]`

ÇIKTI

```
4 3
```

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
▶        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2

`scc(3)` döndü: dusuk[2] = min(1, dusuk[3] = 4) = **1**.

Adım 15 / 20

DEĞİŞKENLER

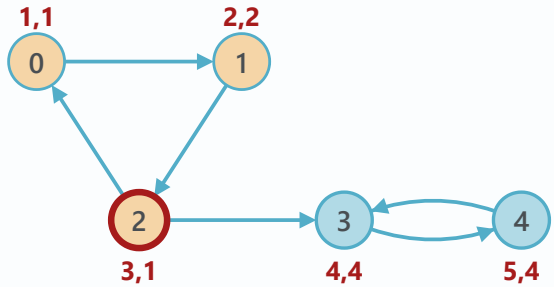
`v = 2` `w = 3` `dusuk = [1, 2, 1, 4, 4]`

ÇIKTI

```
4 3
```

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
▶    if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2

dusuk[2] = 1 ≠ indeks[2] = 3 → 2 kök değil, yığında kalır; çağrı döner.

Adım 16 / 20

DEĞİŞKENLER

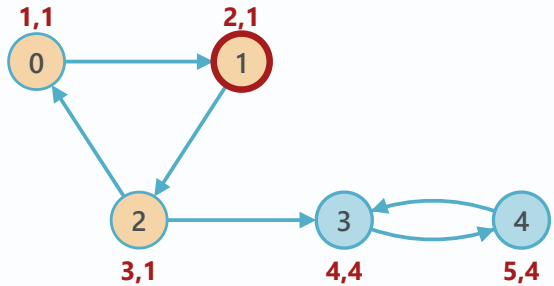
`v = 2` `w = –` `dusuk = [1, 2, 1, 4, 4]`

ÇIKTI

```
4 3
```

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
▶        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2

`scc(2)` döndü: dusuk[1] = min(2, dusuk[2] = 1) = **1**.

Adım 17 / 20

DEĞİŞKENLER

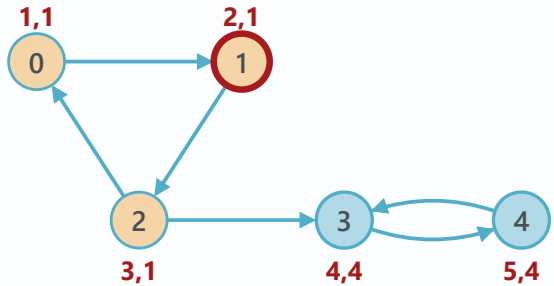
`v = 1` `w = 2` `dusuk = [1, 1, 1, 4, 4]`

ÇIKTI

```
4 3
```

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
▶    if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2

dusuk[1] = 1 ≠ indeks[1] = 2 → 1 kök değil, yığında kalır; çağrı döner.

Adım 18 / 20

DEĞİŞKENLER

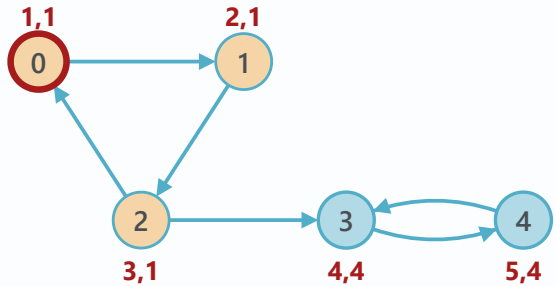
`v = 1` `w = –` `dusuk = [1, 1, 1, 4, 4]`

ÇIKTI

```
4 3
```

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
▶        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
11   if (dusuk[v] == indeks[v]) {
12     int w;
13     do {
14       w = yigin.pop(); yiginda[w] = false;
15       System.out.print(w + " ");
16     } while (w != v);
17     System.out.println();
18   }
19 }
```



yığın

0	1	2

`scc(1)` döndü: dusuk[0] = min(1, dusuk[1] = 1) = **1**.

Adım 19 / 20

DEĞİŞKENLER

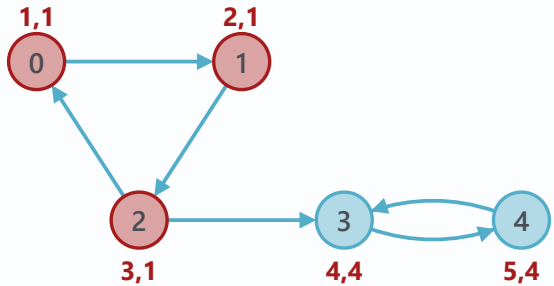
`v = 0` `w = 1` `dusuk = [1, 1, 1, 4, 4]`

ÇIKTI

```
4 3
```

Tarjan: `scc(0)` İzleme

```
1  void scc(int v) {
2    indeks[v] = dusuk[v] = ++sayac;
3    yigin.push(v); yiginda[v] = true;
4    for (int w : komsu[v]) {
5      if (indeks[w] == 0) {
6        scc(w);
7        dusuk[v] = Math.min(dusuk[v], dusuk[w]);
8      } else if (yiginda[w])
9        dusuk[v] = Math.min(dusuk[v], indeks[w]);
10   }
▶    if (dusuk[v] == indeks[v]) {
▶      int w;
▶      do {
▶        w = yigin.pop(); yiginda[w] = false;
▶        System.out.print(w + " ");
▶      } while (w != v);
▶      System.out.println();
18   }
19 }
```



yığın

dusuk[0] == indeks[0] = 1 → 0 köktür; 2, 1, 0 çekilip yazdırılır: SCC **{0, 1, 2}**.

Adım 20 / 20

DEĞİŞKENLER

`v = 0` `w = –` `dusuk = [1, 1, 1, 4, 4]`

ÇIKTI

```
4 3
2 1 0
```

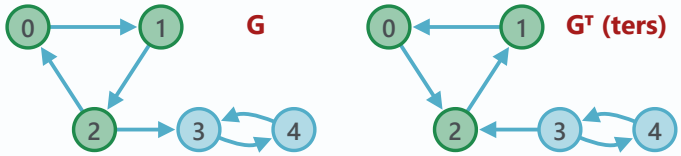
Karmaşıklık ve Kosaraju

Tarjan'ın karmaşıklığı: O(V + E)

- V düğüm sayısı, E kenar sayısı.
- Her düğüm bir kez ziyaret edilir, yığına bir kez itilip bir kez çekilir: O(V).
- Her kenar bir kez incelenir: O(E).
- Ek bellek: indeks, enDüşükBağ, yiginda dizileri ve yığın: O(V).

Kosaraju algoritması

1. DFS ile düğümler **bitiş sırasına** göre yığına itilir.
2. Çizgenin **tersi** (transpose) oluşturulur.
3. Yığından çekilen sırayla ters çizgede DFS yapılır; her DFS ağacı bir SCC'dir.

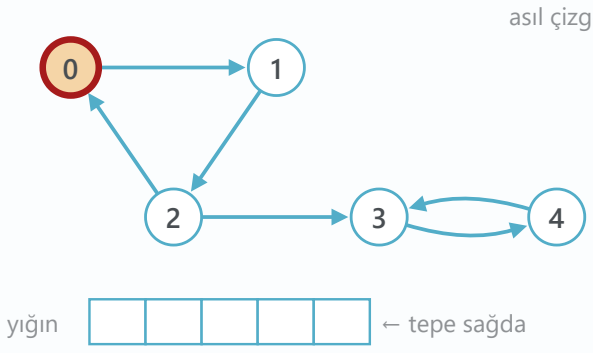


"Tarjan Adım Adım" çizgesinde Kosaraju da aynı dört SCC'yi bulur: {1, 2, 3}, {4, 5}, {6, 7}, {8}.

	Tarjan	Kosaraju
DFS sayısı	1	2 (biri ters çizgede)
Ek yapı	indeks / enDüşükBağ, yığın	ters çizge (O(V + E) bellek)
Zaman	O(V + E)	O(V + E)

Kosaraju: İki DFS ile SCC

```
1  void kosaraju() {
2    for (int v = 0; v < n; v++)  // 1. tur
3      if (!ziyaret[v]) dfs1(v);
4    Arrays.fill(ziyaret, false); // 2. tur
5    while (!yigin.isEmpty()) {
6      int v = yigin.pop();
7      if (!ziyaret[v]) {   // yeni SCC
8        dfs2(v); System.out.println(); }
9    }
10 }
11 void dfs1(int v) {
▶    ziyaret[v] = true;
13   for (int w : komsu[v])
14     if (!ziyaret[w]) dfs1(w);
15   yigin.push(v);       // v bitti
16 }
```



1. tur: asıl çizgede DFS. `dfs1(0)` : 0 işaretlenir. Yığın (sağda tepe) boş; bir düğüm ancak **işi bitince** yığına itilecek.

Adım 1 / 12

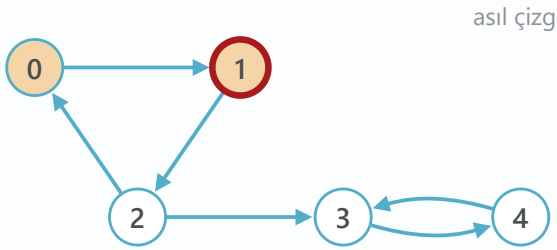
DEĞİŞKENLER

`v = 0` `yigin = []`

ÇIKTI

Kosaraju: İki DFS ile SCC

```
 1 void kosaraju() {
 2   for (int v = 0; v < n; v++)  // 1. tur
 3     if (!ziyaret[v]) dfs1(v);
 4   Arrays.fill(ziyaret, false); // 2. tur
 5   while (!yigin.isEmpty()) {
 6     int v = yigin.pop();
 7     if (!ziyaret[v]) {   // yeni SCC
 8       dfs2(v); System.out.println(); }
 9   }
10 }
11 void dfs1(int v) {
▶    ziyaret[v] = true;
13   for (int w : komsu[v])
▶      if (!ziyaret[w]) dfs1(w);
15   yigin.push(v);       // v bitti
16 }
```



Komşu 1 ziyaret edilmemiş → `dfs1(1)` : 1 işaretlenir.

Adım 2 / 12

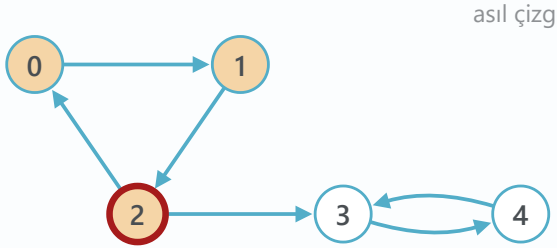
DEĞİŞKENLER

`v = 1` `yigin = []`

ÇIKTI

Kosaraju: İki DFS ile SCC

```
 1 void kosaraju() {
 2   for (int v = 0; v < n; v++)  // 1. tur
 3     if (!ziyaret[v]) dfs1(v);
 4   Arrays.fill(ziyaret, false); // 2. tur
 5   while (!yigin.isEmpty()) {
 6     int v = yigin.pop();
 7     if (!ziyaret[v]) {   // yeni SCC
 8       dfs2(v); System.out.println(); }
 9   }
10 }
11 void dfs1(int v) {
▶    ziyaret[v] = true;
13   for (int w : komsu[v])
▶      if (!ziyaret[w]) dfs1(w);
15   yigin.push(v);       // v bitti
16 }
```



Komşu 2 ziyaret edilmemiş → `dfs1(2)` : 2 işaretlenir.

Adım 3 / 12

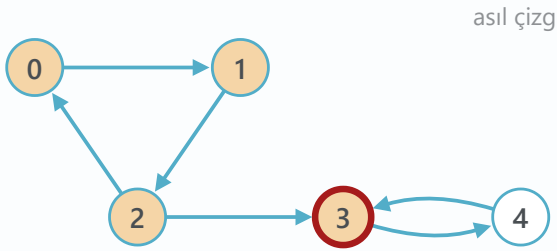
DEĞİŞKENLER

`v = 2` `yigin = []`

ÇIKTI

Kosaraju: İki DFS ile SCC

```
 1 void kosaraju() {
 2   for (int v = 0; v < n; v++)  // 1. tur
 3     if (!ziyaret[v]) dfs1(v);
 4   Arrays.fill(ziyaret, false); // 2. tur
 5   while (!yigin.isEmpty()) {
 6     int v = yigin.pop();
 7     if (!ziyaret[v]) {   // yeni SCC
 8       dfs2(v); System.out.println(); }
 9   }
10 }
11 void dfs1(int v) {
▶    ziyaret[v] = true;
13   for (int w : komsu[v])
▶      if (!ziyaret[w]) dfs1(w);
15   yigin.push(v);       // v bitti
16 }
```



çağrılar: dfs1(0) → dfs1(1) → dfs1(2) → dfs1(3)

Komşu 3 ziyaret edilmemiş → `dfs1(3)` : 3 işaretlenir. (2'nin ilk komşusu 0 zaten ziyaretli, atlandı.)

Adım 4 / 12

DEĞİŞKENLER

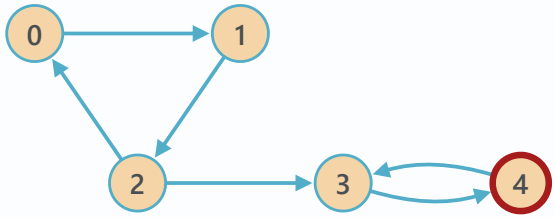
`v = 3` `yigin = []`

ÇIKTI

Kosaraju: İki DFS ile SCC

```
 1 void kosaraju() {
 2   for (int v = 0; v < n; v++)  // 1. tur
 3     if (!ziyaret[v]) dfs1(v);
 4   Arrays.fill(ziyaret, false); // 2. tur
 5   while (!yigin.isEmpty()) {
 6     int v = yigin.pop();
 7     if (!ziyaret[v]) {   // yeni SCC
 8       dfs2(v); System.out.println(); }
 9   }
10 }
11 void dfs1(int v) {
12   ziyaret[v] = true;
13   for (int w : komsu[v])
14     if (!ziyaret[w]) dfs1(w);
▶   yigin.push(v);        // v bitti
16 }
```

asıl çizge



yığın | 4 | | | | | ← tepe sağda

çağrılar: dfs1(0) → dfs1(1) → dfs1(2) → dfs1(3) → dfs1(4)

4'ün tek komşusu 3 zaten ziyaretli: 4 bitti → **yığına itilir**. Bitiş sırası yığında birikir.

Adım 5 / 12

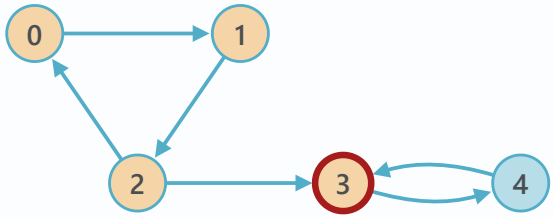
DEĞİŞKENLER

`v = 4` `yigin = [4]`

ÇIKTI

Kosaraju: İki DFS ile SCC

```
 1 void kosaraju() {
 2   for (int v = 0; v < n; v++)  // 1. tur
 3     if (!ziyaret[v]) dfs1(v);
 4   Arrays.fill(ziyaret, false); // 2. tur
 5   while (!yigin.isEmpty()) {
 6     int v = yigin.pop();
 7     if (!ziyaret[v]) {   // yeni SCC
 8       dfs2(v); System.out.println(); }
 9   }
10 }
11 void dfs1(int v) {
12   ziyaret[v] = true;
13   for (int w : komsu[v])
14     if (!ziyaret[w]) dfs1(w);
▶    yigin.push(v);        // v bitti
16 }
```



çağrılar: dfs1(0) → dfs1(1) → dfs1(2) → dfs1(3)

`dfs1(4)` döndü; 3'ün başka komşusu yok: 3 bitti, yığına itilir.

DEĞİŞKENLER

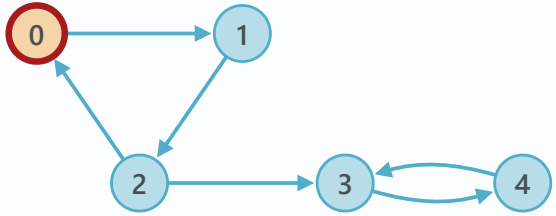
`v = 3` `yigin = [4, 3]`

ÇIKTI

Kosaraju: İki DFS ile SCC

```
 1 void kosaraju() {
 2   for (int v = 0; v < n; v++)  // 1. tur
 3     if (!ziyaret[v]) dfs1(v);
 4   Arrays.fill(ziyaret, false); // 2. tur
 5   while (!yigin.isEmpty()) {
 6     int v = yigin.pop();
 7     if (!ziyaret[v]) {   // yeni SCC
 8       dfs2(v); System.out.println(); }
 9   }
10 }
11 void dfs1(int v) {
12   ziyaret[v] = true;
13   for (int w : komsu[v])
14     if (!ziyaret[w]) dfs1(w);
▶    yigin.push(v);        // v bitti
16 }
```

asıl çizge



yığın | 4 | 3 | 2 | 1 | 0 | ← tepe sağda

çağrılar: dfs1(0)

2, 1 ve 0 sırayla biter ve itilir. Yığının tepesinde 0 var: **en son biten** düğüm, kaynak bileşendedir.

Adım 7 / 12

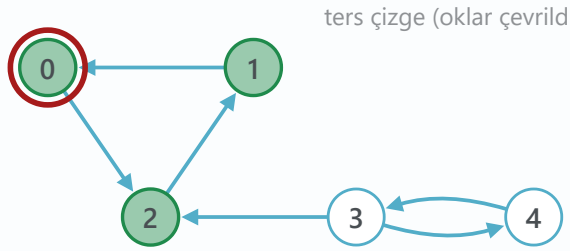
DEĞİŞKENLER

`v = 0` `yigin = [4, 3, 2, 1, 0]`

ÇIKTI

Kosaraju: İki DFS ile SCC

```
1  void kosaraju() {
2    for (int v = 0; v < n; v++)  // 1. tur
3      if (!ziyaret[v]) dfs1(v);
▶    Arrays.fill(ziyaret, false); // 2. tur
▶    while (!yigin.isEmpty()) {
▶      int v = yigin.pop();
▶      if (!ziyaret[v]) {   // yeni SCC
▶        dfs2(v); System.out.println(); }
9    }
10 }
11 void dfs1(int v) {
12   ziyaret[v] = true;
13   for (int w : komsu[v])
14     if (!ziyaret[w]) dfs1(w);
15   yigin.push(v);        // v bitti
16 }
```



2. tur: `ziyaret` temizlendi, oklar **ters** çevrildi. Yığından 0 çekilir; ters çizgede `dfs2(0)` 0 → 2 → 1'e ulaşır ve bunları yazar: SCC {0, 1, 2}.

DEĞİŞKENLER

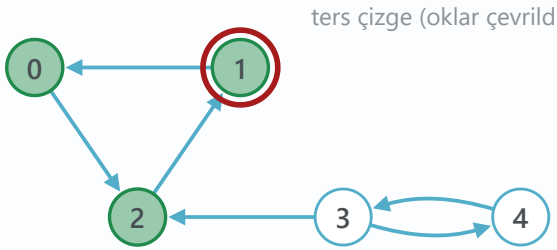
`v = 0` `yigin = [4, 3, 2, 1]`

ÇIKTI

```
0 2 1
```

Kosaraju: İki DFS ile SCC

```
1  void kosaraju() {
2    for (int v = 0; v < n; v++)  // 1. tur
3      if (!ziyaret[v]) dfs1(v);
4    Arrays.fill(ziyaret, false); // 2. tur
▶    while (!yigin.isEmpty()) {
▶      int v = yigin.pop();
▶      if (!ziyaret[v]) {   // yeni SCC
8        dfs2(v); System.out.println(); }
9    }
10 }
11 void dfs1(int v) {
12   ziyaret[v] = true;
13   for (int w : komsu[v])
14     if (!ziyaret[w]) dfs1(w);
15   yigin.push(v);        // v bitti
16 }
```



1 çekilir ama zaten ziyaretli (0'ın bileşeninde yazıldı): atlanır. Aynı şey sıradaki 2 için de olacak.

Adım 9 / 12

DEĞİŞKENLER

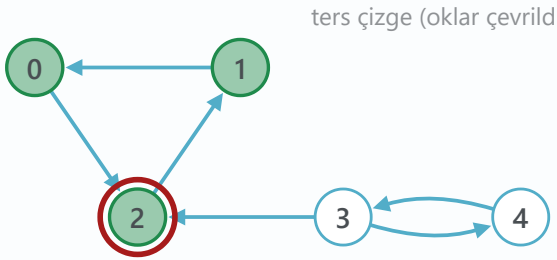
`v = 1` `yigin = [4, 3, 2]`

ÇIKTI

```
0 2 1
```

Kosaraju: İki DFS ile SCC

```
1  void kosaraju() {
2    for (int v = 0; v < n; v++)  // 1. tur
3      if (!ziyaret[v]) dfs1(v);
4    Arrays.fill(ziyaret, false); // 2. tur
▶    while (!yigin.isEmpty()) {
▶      int v = yigin.pop();
▶      if (!ziyaret[v]) {   // yeni SCC
8        dfs2(v); System.out.println(); }
9    }
10 }
11 void dfs1(int v) {
12   ziyaret[v] = true;
13   for (int w : komsu[v])
14     if (!ziyaret[w]) dfs1(w);
15   yigin.push(v);        // v bitti
16 }
```



2 de ziyaretli: atlanır.

Adım 10 / 12

DEĞİŞKENLER

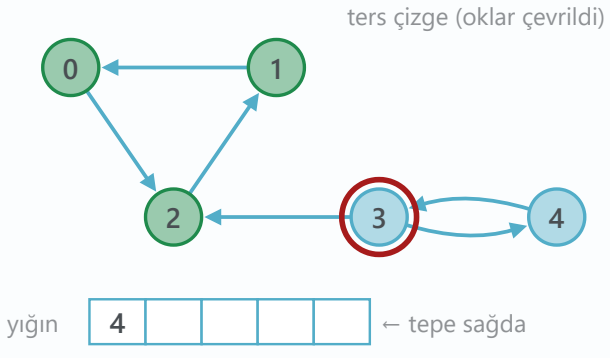
`v = 2` `yigin = [4, 3]`

ÇIKTI

```
0 2 1
```

Kosaraju: İki DFS ile SCC

```
1  void kosaraju() {
2    for (int v = 0; v < n; v++)  // 1. tur
3      if (!ziyaret[v]) dfs1(v);
4    Arrays.fill(ziyaret, false); // 2. tur
5    while (!yigin.isEmpty()) {
▶      int v = yigin.pop();
▶      if (!ziyaret[v]) {   // yeni SCC
▶        dfs2(v); System.out.println(); }
9    }
10 }
11 void dfs1(int v) {
12   ziyaret[v] = true;
13   for (int w : komsu[v])
14     if (!ziyaret[w]) dfs1(w);
15   yigin.push(v);        // v bitti
16 }
```



3 çekilir, ziyaret edilmemiş: ters çizgede `dfs2(3)` 3 → 4'e ulaşır (3 → 2 ters kenarı ziyaretli 2'ye gider, atlanır). SCC {3, 4}.

DEĞİŞKENLER

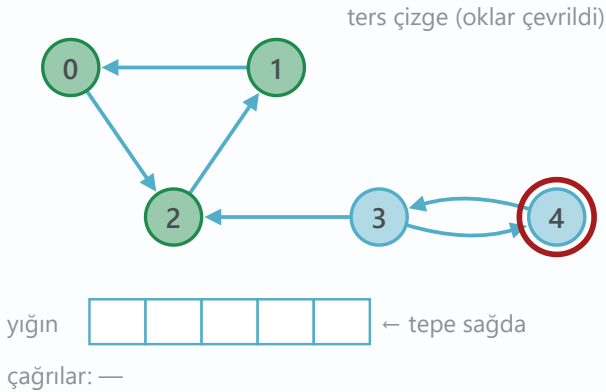
`v = 3` `yigin = [4]`

ÇIKTI

```
0 2 1
3 4
```

Kosaraju: İki DFS ile SCC

```
1  void kosaraju() {
2    for (int v = 0; v < n; v++)  // 1. tur
3      if (!ziyaret[v]) dfs1(v);
4    Arrays.fill(ziyaret, false); // 2. tur
▶    while (!yigin.isEmpty()) {
▶      int v = yigin.pop();
▶      if (!ziyaret[v]) {   // yeni SCC
8        dfs2(v); System.out.println(); }
9    }
10 }
11 void dfs1(int v) {
12   ziyaret[v] = true;
13   for (int w : komsu[v])
14     if (!ziyaret[w]) dfs1(w);
15   yigin.push(v);        // v bitti
16 }
```



4 çekilir, ziyaretli: atlanır. Yığın boşaldı; iki SCC bulundu: {0, 1, 2} ve {3, 4}.

Adım 12 / 12

DEĞİŞKENLER

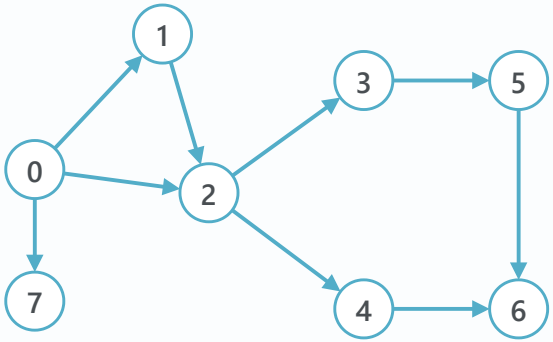
`v = 4` `yigin = []`

ÇIKTI

```
0 2 1
3 4
```

UYGULAMALAR

Topolojik Sıralama



Giriş dereceleri: 0:0, 1:1, 2:2, 3:1, 4:1, 5:1, 6:2, 7:1

- Döngüsüz yönlü çizgede (DAG) her u → v kenarı için u'nun v'den **önce** geldiği sıralama: ön koşullu dersler, derleme sırası, görev planlama.
- **Kahn** (BFS tabanlı): giriş derecesi 0 olanlar kuyruğa konur; çıkarılan düğümün kenarları silinir, derecesi 0'a düşen kuyruğa eklenir.
- **DFS tabanlı**: düğümler bitiş sırasına göre listelenir, liste ters çevrilir.

Kahn	**0 1 7 2 3 4 5 6**
DFS (ters bitiş)	**0 7 1 2 4 3 5 6**

İkisi de geçerli: topolojik sıra tek değildir. Çıkarılan düğüm sayısı V'den azsa çizgede **döngü** vardır (DFS örneğindeki 2 → 0 kenarı eklenirse sıralama yapılamaz). Zaman O(V + E).

Topolojik Sıralama: Kahn Algoritması İzleme

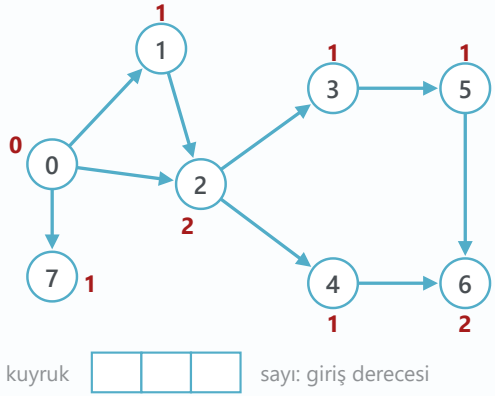
```
1  void kahn() {
▶    int[] giris = new int[n];
▶    for (int v = 0; v < n; v++)
▶      for (int w : komsu[v]) giris[w]++;
5    Queue<Integer> kuyruk = new LinkedList<>();
6    for (int v = 0; v < n; v++)
7      if (giris[v] == 0) kuyruk.add(v);
8    while (!kuyruk.isEmpty()) {
9      int v = kuyruk.poll();
10     System.out.print(v + " ");
11     for (int w : komsu[v])
12       if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```

DEĞİŞKENLER

`v = –` `kuyruk = []`

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

ÇIKTI



Döngüsüz yönlü çizge (DAG). Önce her düğümün **giriş derecesi** sayılır: kaç kenar ona giriyor? Düğüm yanındaki küçük sayılar `giris[]` değerleridir.

Adım 1 / 10

Topolojik Sıralama: Kahn Algoritması İzleme

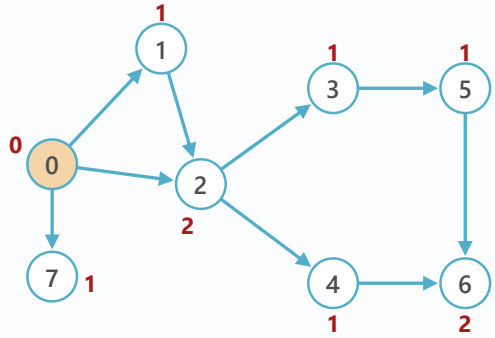
```
1  void kahn() {
2    int[] giris = new int[n];
3    for (int v = 0; v < n; v++)
4      for (int w : komsu[v]) giris[w]++;
5    Queue<Integer> kuyruk = new LinkedList<>();
▶    for (int v = 0; v < n; v++)
▶      if (giris[v] == 0) kuyruk.add(v);
8    while (!kuyruk.isEmpty()) {
9      int v = kuyruk.poll();
10     System.out.print(v + " ");
11     for (int w : komsu[v])
12       if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```

DEĞİŞKENLER

`v = –` `kuyruk = [0]`

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

ÇIKTI



Giriş derecesi 0 olan tek düğüm 0: hiçbir ön koşulu yok, kuyruğa girer.

Adım 2 / 10

Topolojik Sıralama: Kahn Algoritması İzleme

```
1  void kahn() {
2    int[] giris = new int[n];
3    for (int v = 0; v < n; v++)
4      for (int w : komsu[v]) giris[w]++;
5    Queue<Integer> kuyruk = new LinkedList<>();
6    for (int v = 0; v < n; v++)
7      if (giris[v] == 0) kuyruk.add(v);
8    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```

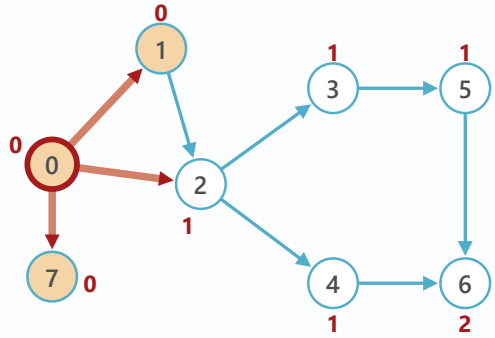
DEĞİŞKENLER

`v = 0` `kuyruk = [1, 7]`

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

ÇIKTI

```
0
```

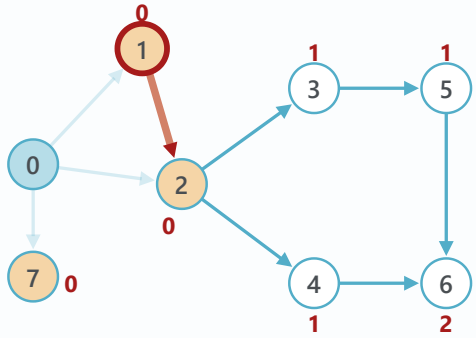


kuyruk | 1 | 7 | | sayı: giriş derecesi

0 çıkarılıp yazılır; 0'ın kenarları silinmiş sayılır: 1, 2 ve 7 düğümlerinin derecesi 1 azalır. Sıfıra inen 1 ve 7 kuyruğa eklenir.

Topolojik Sıralama: Kahn Algoritması İzleme

```
1  void kahn() {
2    int[] giris = new int[n];
3    for (int v = 0; v < n; v++)
4      for (int w : komsu[v]) giris[w]++;
5    Queue<Integer> kuyruk = new LinkedList<>();
6    for (int v = 0; v < n; v++)
7      if (giris[v] == 0) kuyruk.add(v);
8    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```



kuyruk | 7 | 2 | | sayı: giriş derecesi

1 çıkarılıp yazılır; 1'in kenarları silinmiş sayılır: 2 düğümünün derecesi 1 azalır. Sıfıra inen 2 kuyruğa eklenir.

Adım 4 / 10

DEĞİŞKENLER

`v = 1` `kuyruk = [7, 2]`

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

ÇIKTI

```
0 1
```

Topolojik Sıralama: Kahn Algoritması İzleme

```
1  void kahn() {
2    int[] giris = new int[n];
3    for (int v = 0; v < n; v++)
4      for (int w : komsu[v]) giris[w]++;
5    Queue<Integer> kuyruk = new LinkedList<>();
6    for (int v = 0; v < n; v++)
7      if (giris[v] == 0) kuyruk.add(v);
8    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```

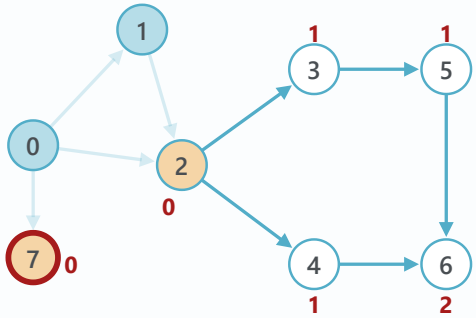
DEĞİŞKENLER

`v = 7` `kuyruk = [2]`

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

ÇIKTI

```
0 1 7
```



kuyruk | 2 | | | sayı: giriş derecesi

7 çıkarılıp yazılır. Çıkan kenarı yok; hiçbir derece değişmez.

Topolojik Sıralama: Kahn Algoritması İzleme

```
1  void kahn() {
2    int[] giris = new int[n];
3    for (int v = 0; v < n; v++)
4      for (int w : komsu[v]) giris[w]++;
5    Queue<Integer> kuyruk = new LinkedList<>();
6    for (int v = 0; v < n; v++)
7      if (giris[v] == 0) kuyruk.add(v);
8    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```

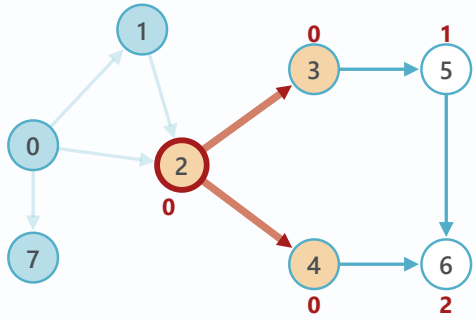
DEĞİŞKENLER

`v = 2` `kuyruk = [3, 4]`

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

ÇIKTI

```
0 1 7 2
```



kuyruk | 3 | 4 | | sayı: giriş derecesi

2 çıkarılıp yazılır; 2'nin kenarları silinmiş sayılır: 3 ve 4 düğümlerinin derecesi 1 azalır. Sıfıra inen 3 ve 4 kuyruğa eklenir.

Topolojik Sıralama: Kahn Algoritması İzleme

```
1  void kahn() {
2    int[] giris = new int[n];
3    for (int v = 0; v < n; v++)
4      for (int w : komsu[v]) giris[w]++;
5    Queue<Integer> kuyruk = new LinkedList<>();
6    for (int v = 0; v < n; v++)
7      if (giris[v] == 0) kuyruk.add(v);
8    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```

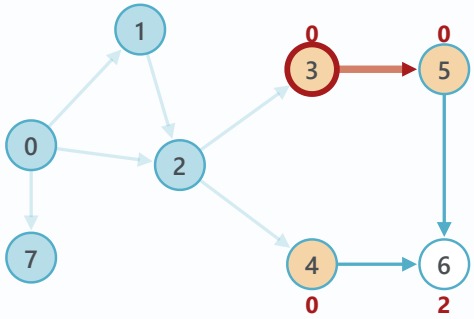
DEĞİŞKENLER

`v = 3` `kuyruk = [4, 5]`

`giris = [0, 0, 0, 0, 0, 0, 2, 0]`

ÇIKTI

```
0 1 7 2 3
```

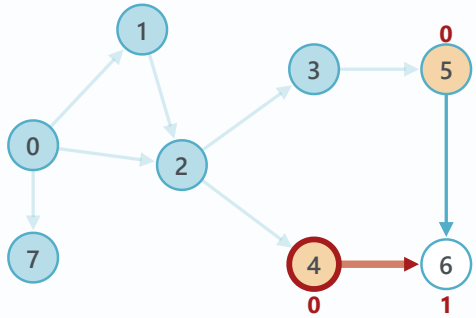


kuyruk | 4 | 5 | | sayı: giriş derecesi

3 çıkarılıp yazılır; 3'ün kenarları silinmiş sayılır: 5 düğümünün derecesi 1 azalır. Sıfıra inen 5 kuyruğa eklenir.

Topolojik Sıralama: Kahn Algoritması İzleme

```
1  void kahn() {
2    int[] giris = new int[n];
3    for (int v = 0; v < n; v++)
4      for (int w : komsu[v]) giris[w]++;
5    Queue<Integer> kuyruk = new LinkedList<>();
6    for (int v = 0; v < n; v++)
7      if (giris[v] == 0) kuyruk.add(v);
8    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```



4 çıkarılıp yazılır; 4'ün kenarları silinmiş sayılır: 6 düğümünün derecesi 1 azalır. Sıfıra inen yok (ör. 6'nın hâlâ 1 ön koşulu var).

DEĞİŞKENLER

`v = 4` `kuyruk = [5]`

`giris = [0, 0, 0, 0, 0, 0, 1, 0]`

ÇIKTI

```
0 1 7 2 3 4
```

Topolojik Sıralama: Kahn Algoritması İzleme

```
1  void kahn() {
2    int[] giris = new int[n];
3    for (int v = 0; v < n; v++)
4      for (int w : komsu[v]) giris[w]++;
5    Queue<Integer> kuyruk = new LinkedList<>();
6    for (int v = 0; v < n; v++)
7      if (giris[v] == 0) kuyruk.add(v);
8    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```

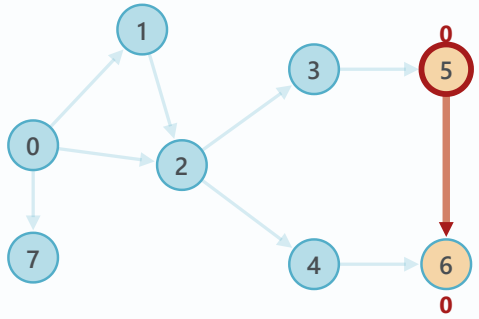
DEĞİŞKENLER

`v = 5` `kuyruk = [6]`

`giris = [0, 0, 0, 0, 0, 0, 0, 0]`

ÇIKTI

```
0 1 7 2 3 4 5
```



kuyruk | 6 | | | sayı: giriş derecesi

5 çıkarılıp yazılır; 5'in kenarları silinmiş sayılır: 6 düğümünün derecesi 1 azalır. Sıfıra inen 6 kuyruğa eklenir.

Topolojik Sıralama: Kahn Algoritması İzleme

```
 1 void kahn() {
 2   int[] giris = new int[n];
 3   for (int v = 0; v < n; v++)
 4     for (int w : komsu[v]) giris[w]++;
 5   Queue<Integer> kuyruk = new LinkedList<>();
 6   for (int v = 0; v < n; v++)
 7     if (giris[v] == 0) kuyruk.add(v);
 8   while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      System.out.print(v + " ");
▶      for (int w : komsu[v])
▶        if (--giris[w] == 0) kuyruk.add(w);
13   }
14 }
```

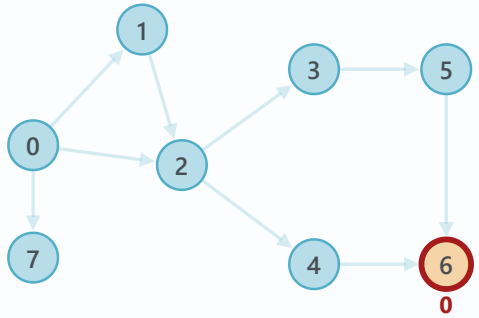
DEĞİŞKENLER

`v = 6` `kuyruk = []`

`giris = [0, 0, 0, 0, 0, 0, 0, 0]`

ÇIKTI

```
0 1 7 2 3 4 5 6
```



6 çıkarılıp yazılır. Çıkan kenarı yok; hiçbir derece değişmez. Kuyruk boşaldı: 8 düğümün hepsi çıktı, demek ki döngü yok. Sıra: **0 1 7 2 3 4 5 6**.

UYGULAMALAR

Gezinmenin Diğer Kullanımları

Bağlantılı bileşenler

Ziyaret edilmemiş her düğümden yeni bir DFS/BFS başlatılır; başlatma sayısı = bileşen sayısı.

İki parçalılık testi

BFS katmanları iki renge boyanır; aynı renkli iki komşu varsa tek uzunlukta döngü vardır, çizge iki parçalı değildir.

Döngü tespiti

Yönsüz: ebeveyn olmayan ziyaretli komşu. Yönlü: yığındaki (sarı) düğüme giden geri kenar.

Köprü ve eklem noktası

Tarjan'ın enDüşükBağ fikri: enDüşükBağ(w) > keşif(v) ise v–w köprüdür.

Yinelemeli derinleştirme (IDDFS)

Derinlik sınırı 0, 1, 2… artan DFS'ler: DFS'in belleği, BFS'in sığ çözüm garantisi.

Rastgele yürüyüş

Her adımda rastgele bir komşuya geçiş; PageRank ve büyük ağlardan örnekleme.

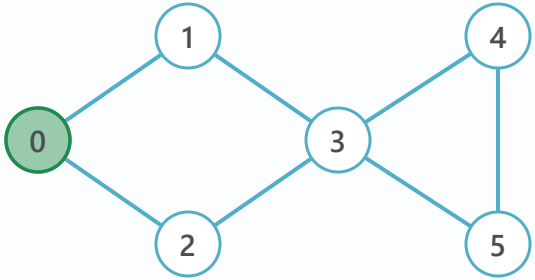
BFS ile İki Parçalılık Testi

```
1  boolean ikiParcaliMi(int s) {
▶    Arrays.fill(renk, -1);
▶    renk[s] = 0; kuyruk.add(s);
4    while (!kuyruk.isEmpty()) {
5      int v = kuyruk.poll();
6      for (int w : komsu[v])
7        if (renk[w] == -1) {
8          renk[w] = 1 - renk[v];
9          kuyruk.add(w);
10       } else if (renk[w] == renk[v])
11         return false;  // aynı renk
12   }
13   return true;
14 }
```

DEĞİŞKENLER

`v = –` `w = –` `kuyruk = [0]`

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





Yönsüz çizge: 0-1-3-2 döngüsü ve 3-4-5 üçgeni. renk = −1 boyanmamış demektir; kaynak 0, renk 0 (yeşil) ile boyanıp kuyruğa girer.

Adım 1 / 6

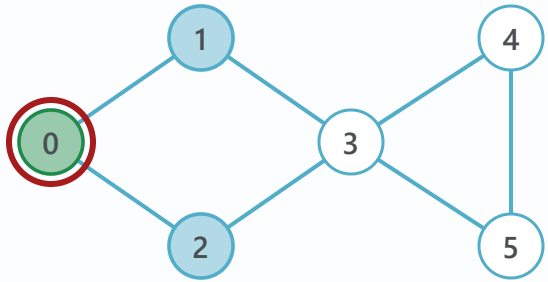
BFS ile İki Parçalılık Testi

```
1  boolean ikiParcaliMi(int s) {
2    Arrays.fill(renk, -1);
3    renk[s] = 0; kuyruk.add(s);
4    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (renk[w] == -1) {
▶          renk[w] = 1 - renk[v];
▶          kuyruk.add(w);
10       } else if (renk[w] == renk[v])
11         return false;  // aynı renk
12   }
13   return true;
14 }
```

DEĞİŞKENLER

`v = 0` `w = –` `kuyruk = [1, 2]`

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



kuyruk | 1 | 2 | | renk 0 | renk 1

0 çıkarıldı (renk 0). Boyanmamış komşular 1 ve 2 karşı renge (1) boyanıp kuyruğa eklenir.

Adım 2 / 6

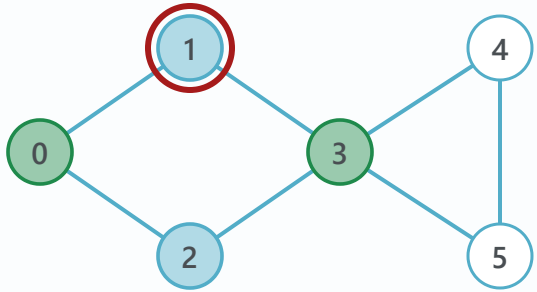
BFS ile İki Parçalılık Testi

```
1  boolean ikiParcaliMi(int s) {
2    Arrays.fill(renk, -1);
3    renk[s] = 0; kuyruk.add(s);
4    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (renk[w] == -1) {
▶          renk[w] = 1 - renk[v];
▶          kuyruk.add(w);
10       } else if (renk[w] == renk[v])
11         return false;  // aynı renk
12   }
13   return true;
14 }
```

DEĞİŞKENLER

`v = 1` `w = –` `kuyruk = [2, 3]`

`renk = [0, 1, 1, 0, –1, –1]`



kuyruk | 2 | 3 | | renk 0 | renk 1

1 çıkarıldı (renk 1). Boyanmamış komşu 3 karşı renge (0) boyanıp kuyruğa eklenir. Komşu 0 zaten karşı renkte: kenar iki kümeyi bağlıyor, sorun yok.

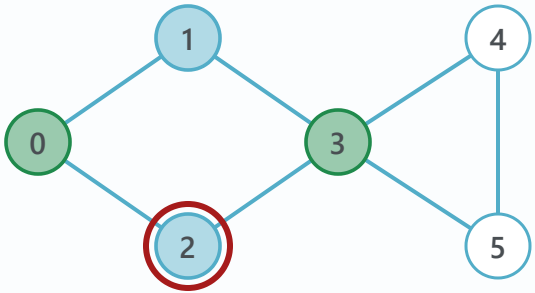
BFS ile İki Parçalılık Testi

```
1  boolean ikiParcaliMi(int s) {
2    Arrays.fill(renk, -1);
3    renk[s] = 0; kuyruk.add(s);
4    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (renk[w] == -1) {
8          renk[w] = 1 - renk[v];
9          kuyruk.add(w);
▶        } else if (renk[w] == renk[v])
11         return false;  // aynı renk
12   }
13   return true;
14 }
```

DEĞİŞKENLER

`v = 2` `w = –` `kuyruk = [3]`

`renk = [0, 1, 1, 0, –1, –1]`



kuyruk | 3 | | | renk 0 | renk 1

2 çıkarıldı (renk 1). Komşu 0 ve 3 zaten karşı renkte: kenarlar iki kümeyi bağlıyor, sorun yok.

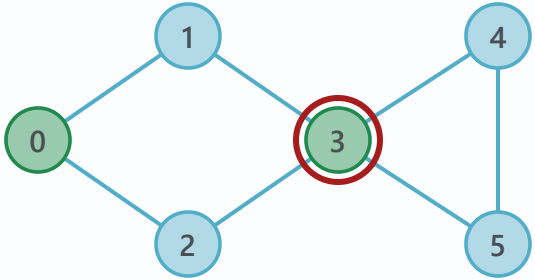
BFS ile İki Parçalılık Testi

```
1  boolean ikiParcaliMi(int s) {
2    Arrays.fill(renk, -1);
3    renk[s] = 0; kuyruk.add(s);
4    while (!kuyruk.isEmpty()) {
▶      int v = kuyruk.poll();
▶      for (int w : komsu[v])
▶        if (renk[w] == -1) {
▶          renk[w] = 1 - renk[v];
▶          kuyruk.add(w);
10       } else if (renk[w] == renk[v])
11         return false;  // aynı renk
12   }
13   return true;
14 }
```

DEĞİŞKENLER

`v = 3` `w = –` `kuyruk = [4, 5]`

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



kuyruk | 4 | 5 | | renk 0 | renk 1

3 çıkarıldı (renk 0). Boyanmamış komşular 4 ve 5 karşı renge (1) boyanıp kuyruğa eklenir. Komşu 1 ve 2 zaten karşı renkte: kenarlar iki kümeyi bağlıyor, sorun yok.

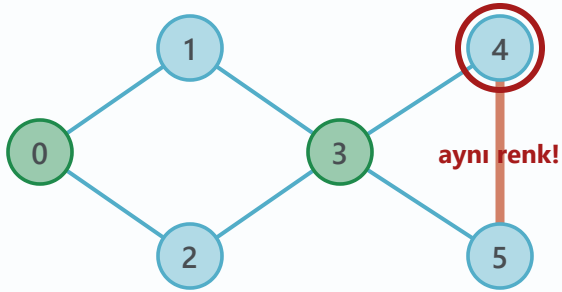
BFS ile İki Parçalılık Testi

```
1  boolean ikiParcaliMi(int s) {
2    Arrays.fill(renk, -1);
3    renk[s] = 0; kuyruk.add(s);
4    while (!kuyruk.isEmpty()) {
5      int v = kuyruk.poll();
6      for (int w : komsu[v])
7        if (renk[w] == -1) {
8          renk[w] = 1 - renk[v];
9          kuyruk.add(w);
▶        } else if (renk[w] == renk[v])
▶          return false;  // aynı renk
12   }
13   return true;
14 }
```

DEĞİŞKENLER

`v = 4` `w = 5` `kuyruk = [5]`

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



kuyruk | 5 | | | renk 0 | renk 1

4 çıkarıldı; komşusu 3 farklı renkte (sorun yok), komşusu 5 ise **aynı renkte** (1). Bir kenarın iki ucu aynı kümeye düştü: 3-4-5 üçgeni gibi **tek uzunlukta döngü** var → `false`.

Özet

Konu	Hatırlanacaklar
Gösterim	liste O(V + E) bellek, matris O(V^2); gezinme listeyle O(V + E), matrisle O(V^2)
DFS	yığın / özyineleme; derine in, geri dön; ziyaret dizisi döngüye karşı şart
Yığınla DFS	çekerken işaretle; bir düğüm birden çok kez itilebilir; sıra komşu sırasının tersi
BFS	kuyruk; eklerken işaretle; katman katman; ebeveyn dizisiyle ağırlıksız en kısa yol
Labirent	hücre = düğüm; BFS tüm yolları eşzamanlı dener, çıkışa ilk ulaştığında yol en kısadır
SCC	karşılıklı ulaşılabilir maksimal küme; bileşen çizgesi DAG'dir
Tarjan	indeks = keşif sırası; enDüşükBağ; enDüşükBağ = indeks → kök, yığından SCC çek; O(V + E)
Uygulamalar	topolojik sıra (Kahn / ters bitiş), bileşenler, iki parçalılık, döngü, köprü

ÖZET

Etkileşimli Simülatörler

Çizge Gezinme Simülatörü

Rastgele yönsüz çizgeler (düğüm sayısı ayarlanabilir, düğümler sürüklenebilir) üzerinde BFS ve DFS'in adım adım gezinmesi.

Çizge Yapısı Algoritmaları

Küçük örnek çizgelerde topolojik sıralama (Kahn), güçlü bağlantılı bileşenler (Kosaraju) ve eklem noktaları.

İki Parçalılık, Döngü ve Köprü

Aynı yönsüz çizge üç açıdan incelenir: BFS ile iki renge boyama, DFS ile döngü tespiti ve köprü kenarlarının bulunması.

ÖZET

Sık Yapılan Hatalar

Ziyaret dizisini unutmak

Döngülü çizgede DFS sonsuz özyinelemeye, BFS sonsuz kuyruğa girer. "BFS döngülü çizgede çalışmaz" sanısının kaynağı budur; işaretlemeyle sorunsuz çalışır.

BFS'te çıkarırken işaretlemek

Aynı düğüm birçok komşudan kuyruğa eklenir; sonuç yine doğru olabilir ama kuyruk O(E)'ye büyür. Doğrusu eklerken işaretlemektir.

DFS'ten en kısa yol beklemek

DFS'in hedefe ilk ulaştığı yol en kısa değildir. Ağırlıksızda BFS, ağırlıklıda Dijkstra gerekir.

Yığınla DFS'in sırası

Komşular sırayla itilince sıra ters çıkar (0 7 2 … ≠ 0 1 2 …). Özyinelemeli sırayı isteyen ters sırada itmelidir.

Tarjan'da "yığında mı?" denetimini atlamak

Yalnızca yığındaki w için `min(…, indeks(w))` yapılır; SCC'si bulunmuş w yok sayılır. Aksi hâlde kökler kaybolur.

Bağlı olmayan çizge

Tek bir kaynaktan gezinme yalnızca o bileşeni görür. Tüm çizge için gezinme her ziyaret edilmemiş düğümden yeniden başlatılır.

ÖZET

Örnek Problemler

Yönsüz çizge, komşular alfabetik: A–B, A–C, B–D, C–D, C–E, D–F, E–F, F–G.

1. A'dan BFS ziyaret sırasını ve her düğümün uzaklığını yazın.
2. A'dan özyinelemeli DFS ziyaret sırası nedir?
3. A'dan G'ye en kısa yol (BFS ebeveyn dizisiyle) nedir?
4. Çizge iki parçalı mıdır? Kümeleri verin.
5. Yönlü çizge 0→1, 1→2, 2→0, 1→3, 3→4, 4→5, 5→3, 5→6: Tarjan (0'dan, indeks 1'den) ile SCC'leri bulunma sırasıyla yazın.
6. A→C, B→C, B→D, C→E, D→E, E→F için Kahn sırası (kuyruk alfabetik başlar)?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. Örnek problemlerdeki çizgede A'dan **yığınla DFS** (komşular alfabetik itilir) sırası nedir? Toplam kaç itme yapılır?
2. V = 1000, E = 3000 olan bir çizgede BFS'in komşuluk listesi ve matris ile yaklaşık iş miktarını karşılaştırın.
3. BFS'in ağırlıklı çizgede en kısa yolu bulamadığını gösteren küçük bir örnek verin.
4. Bir yönsüz çizgenin bağlantılı bileşen sayısını bulan algoritmayı tarif edin; karmaşıklığı nedir?
5. Tarjan'da "bitmiş SCC'deki w" için `min(v.enDüşükBağ, w.indeks)` yapılsaydı "Tarjan Adım Adım" örneğinde ne olurdu?

▶ **Cevapları göster**

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, MIT Press, 2022 (Bölüm 20: Elementary Graph Algorithms).
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, Addison-Wesley, 2011 (Bölüm 4.1–4.2).
- R. E. Tarjan, "Depth-First Search and Linear Graph Algorithms", *SIAM Journal on Computing* 1(2), 1972.
- A. B. Kahn, "Topological Sorting of Large Networks", *Communications of the ACM* 5(11), 1962.
- Kleinberg ve Tardos, *Algorithm Design*, Pearson, 2006 (Bölüm 3: Graphs).

Son

Bölüm 4.1: Çizge Gezinme Algoritmaları

Sonraki bölüm: Bölüm 4.2: En Kısa Yol Algoritmaları · Terim sözlüğü