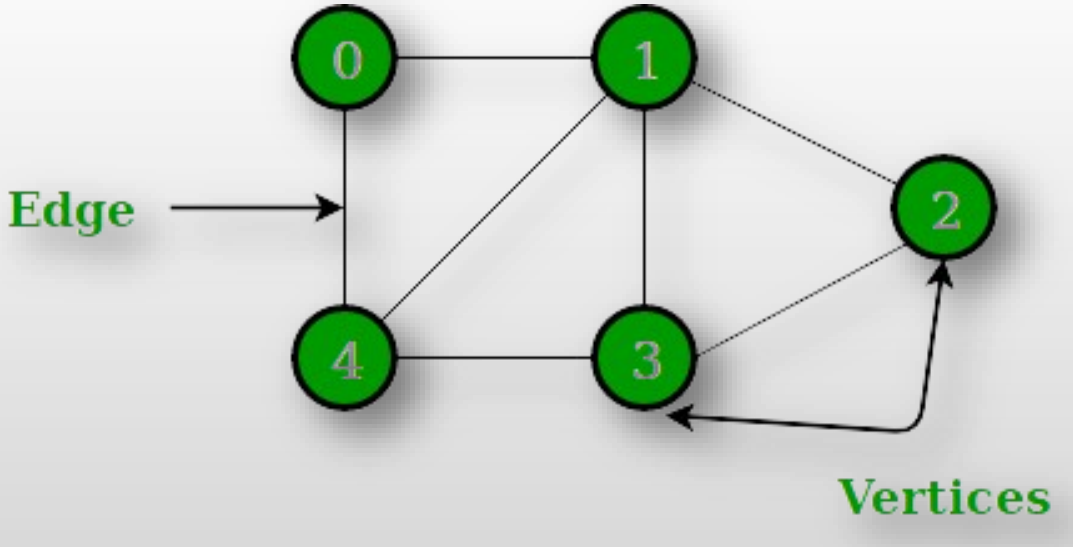
Bölüm 11: Çizge

Veri Yapıları

Çizge

- Noktaların (düğümler) ve bu noktaları birbirine bağlayan kenarların (yollar) bir araya gelmesiyle oluşur.
- Karmaşık ilişkileri, ağları ve yapıları temsil etmek için kullanılır.
- Çizge teorisi, düğümler ve kenarlar arasındaki ilişkileri inceler.

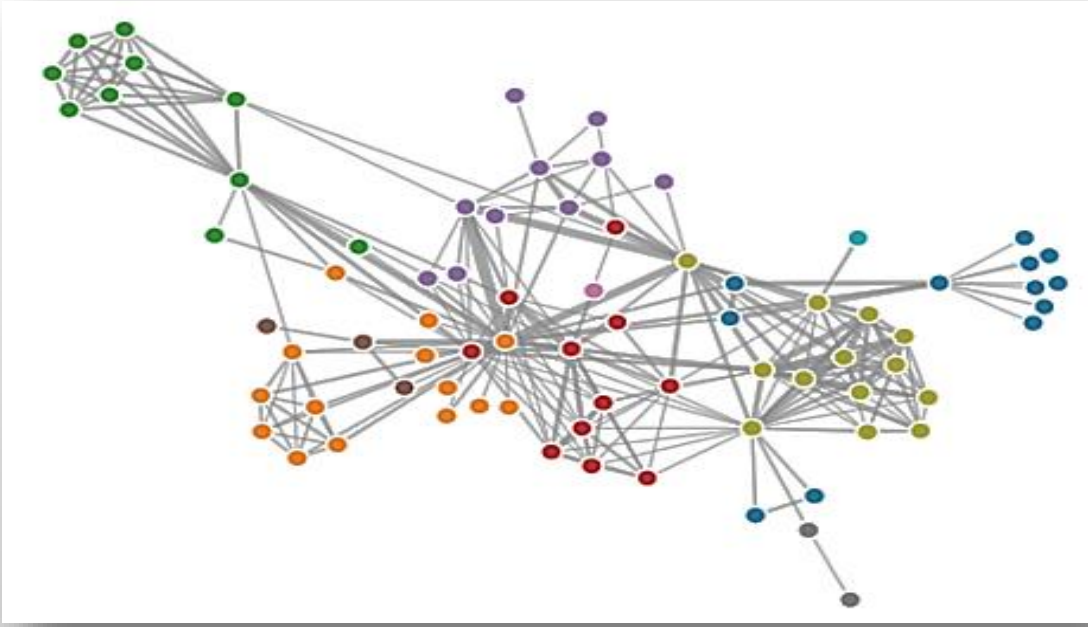


Nokta (Düğüm)

- Çizgenin temel yapı taşı.
- Şehirler, kişiler, bilgisayarlar gibi nesneleri temsil eder.
- **Örnek**:
  - Haritadaki şehirler.

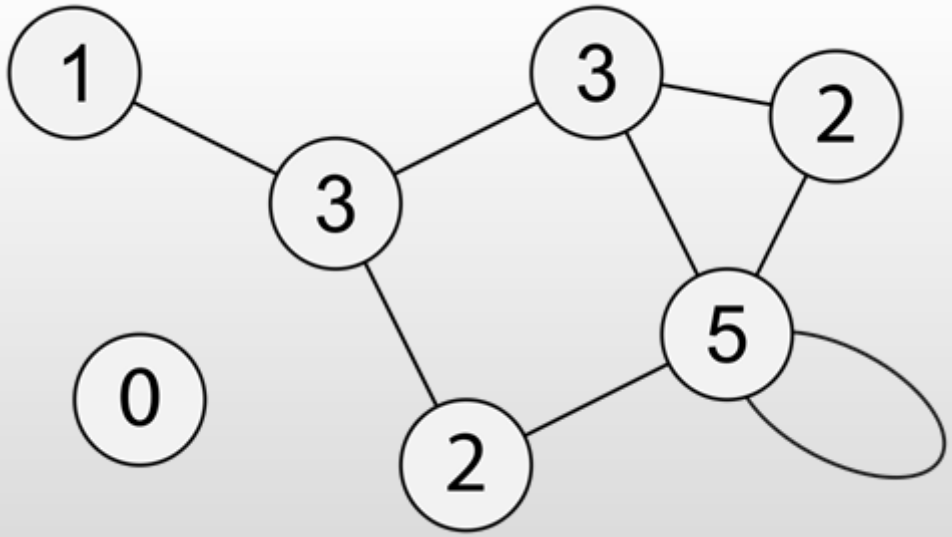
Kenar (Yol)

- Düğümleri birbirine bağlayan bağlantılardır.
- Kenarlar düğümler arasındaki ilişkiyi temsil eder.
- Eğer (v, w) bir kenar ise, v ve w birbirine bitişiktir.
- **Örnek**:
  - Şehirler arasındaki yollar
  - Ağ bağlantısı



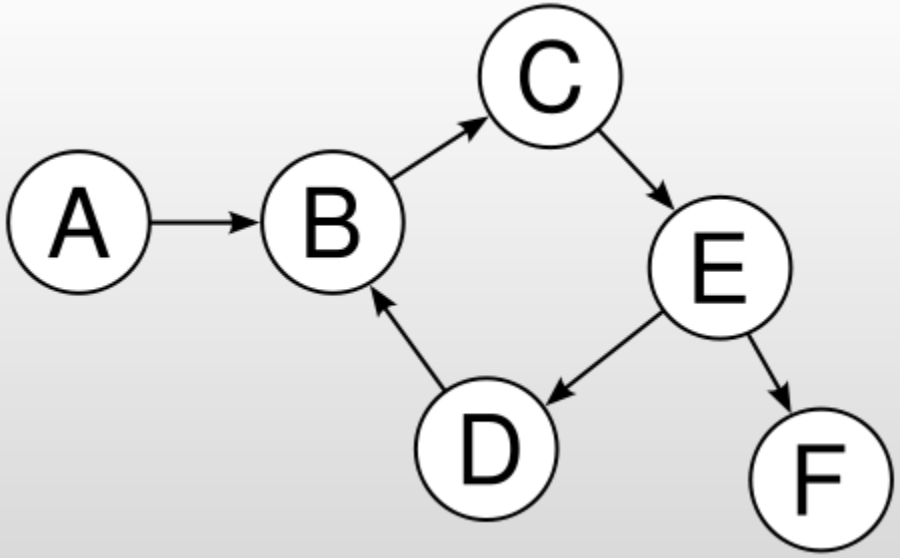
Derece (Degree)

- Düğümün sahip olduğu kenar sayısıdır.
- Derecesiz (0 derece) düğümler de olabilir.
- Örnek:
  - Bir kavşaktaki yolların sayısı.



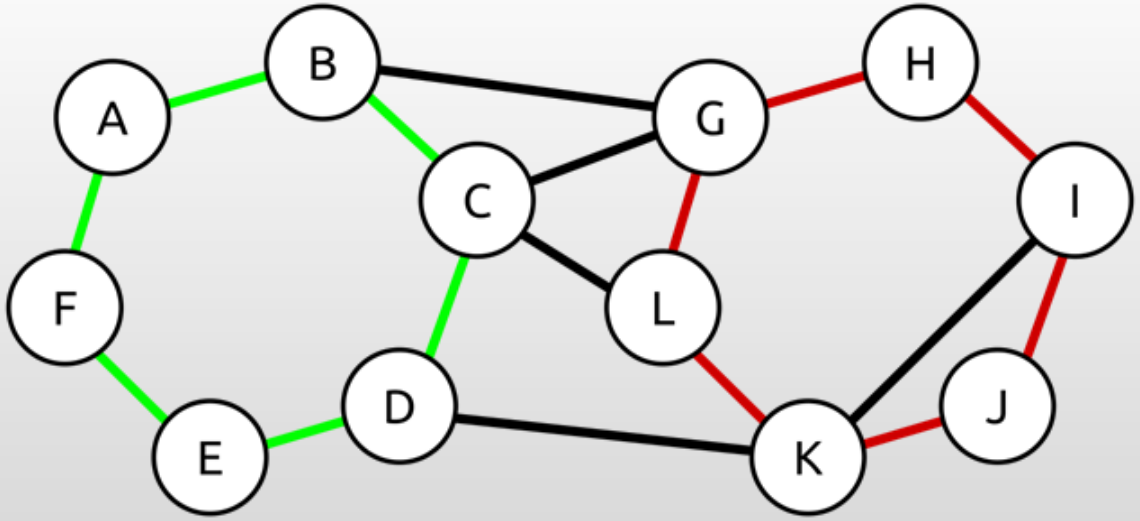
Yönlü Çizge

- Kenarlar, bir düğümden diğer düğüme belirli bir yöndedir.
- Düğüme geri dönüş yolu olmayabilir.
- Digraph olarak da adlandırılır.
- Örnek:
  - İnternet sayfaları ve bağlantıları.

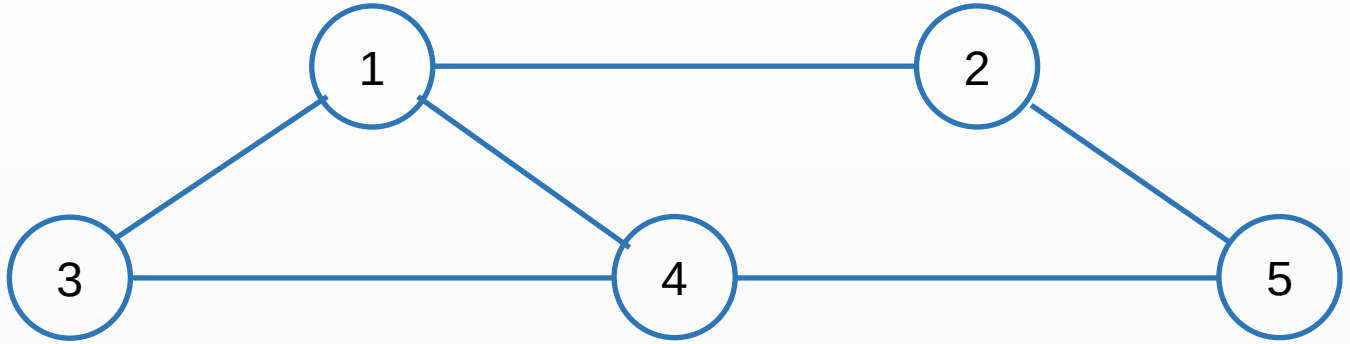


Döngü (Cycle)

- Bir düğümden başlayıp aynı düğümde sona eren bir yoldur.
- Basit bir döngü, diğer düğümleri sadece bir kez geçen bir döngüdür.
- Çizgeler içerisinde döngü olabilir.
- Ağaç yapısında döngü bulunmaz.
- Örnek:
  - Uçuş rotası



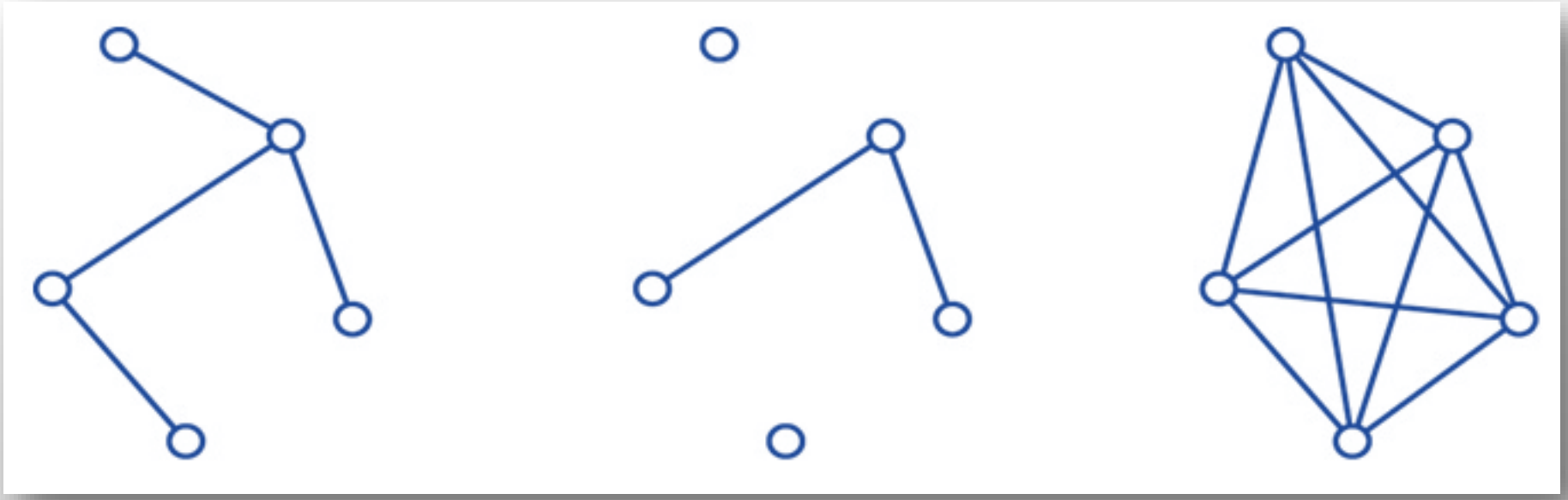
Örnek Çizge



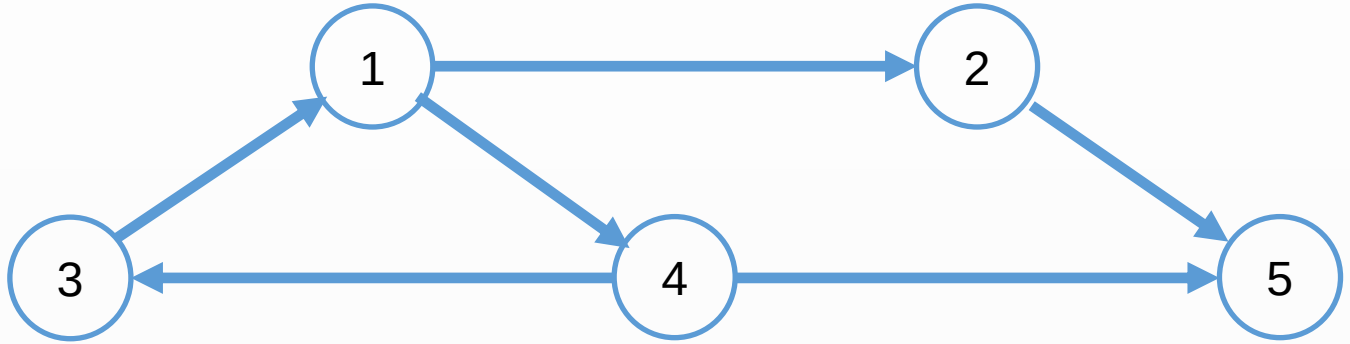
- **V =** { 1, 2, 3, 4, 5}
- **E =** { (1,2), (1,3), (1,4), (2,5), (3,4), (4,5) }
- Düğüm 1 ve Düğüm 2 **komşu**
- 1 -> 2 -> 5 **basit yol**
- 1 -> 3 -> 4 -> 1 **basit döngü**
- 1 -> 3 -> 4 -> 1 -> 2 -> **basit olmayan yol**
- 1 -> 3 -> 4 -> 1 -> 4 -> 1 **basit olmayan döngü**

Bağlantılı ve Tam Çizge

- Bağlantılı çizge, her iki farklı düğüm çifti arasında bir yol içerir.
- Tam çizge, her iki farklı düğüm çifti arasında bir kenar içerir.

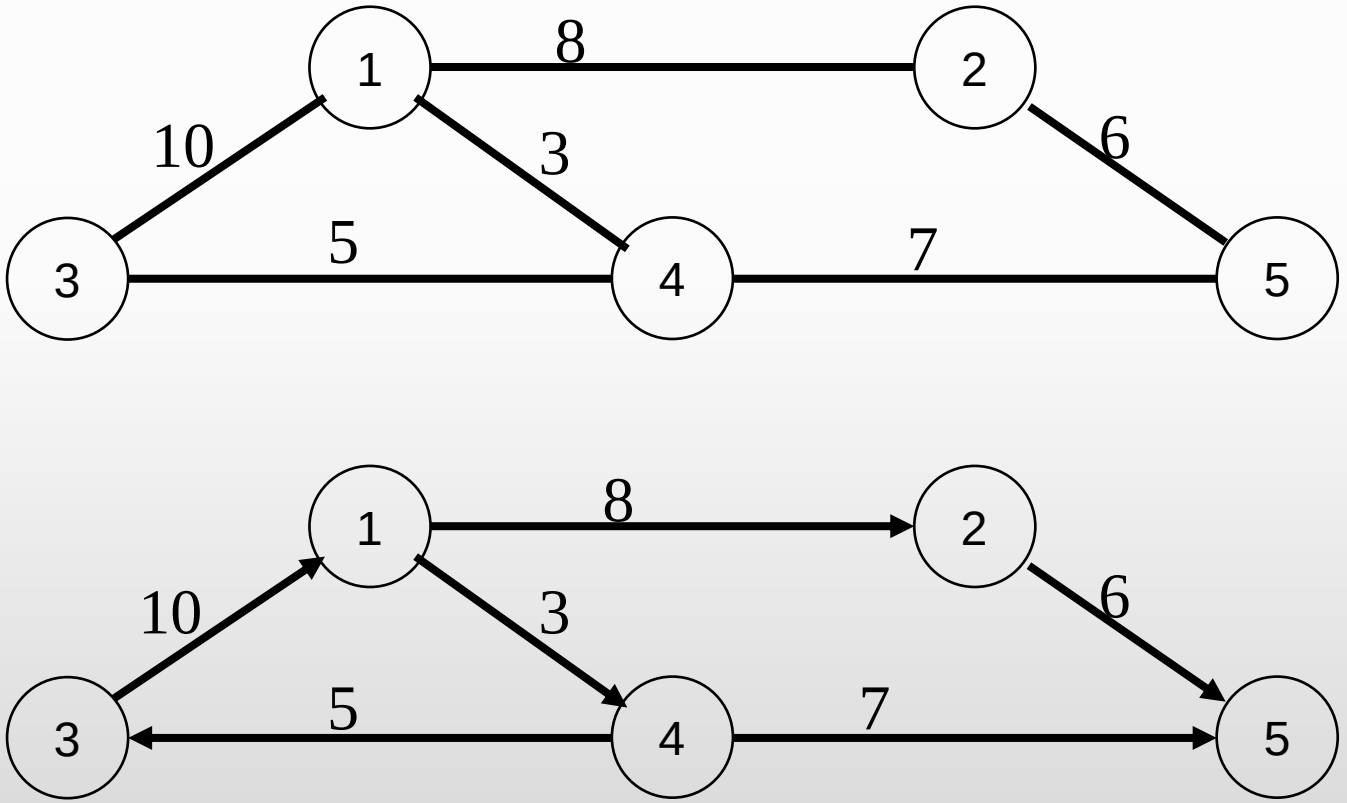


Yönlü Çizge



- **V =** { 1, 2, 3, 4, 5}
- **E =** { (1,2), (1,4), (2,5), (4,5), (3,1), (4,5) }
- Düğüm 2, Düğüm 1'in **komşusu**
- Düğüm 1, Düğüm 2'nin komşusu **değil!**
- 1 -> 2 -> 5 **yönlü yol**
- 1 -> 4 -> 3 -> 1 **yönlü döngü**

Ağırlıklı Çizge



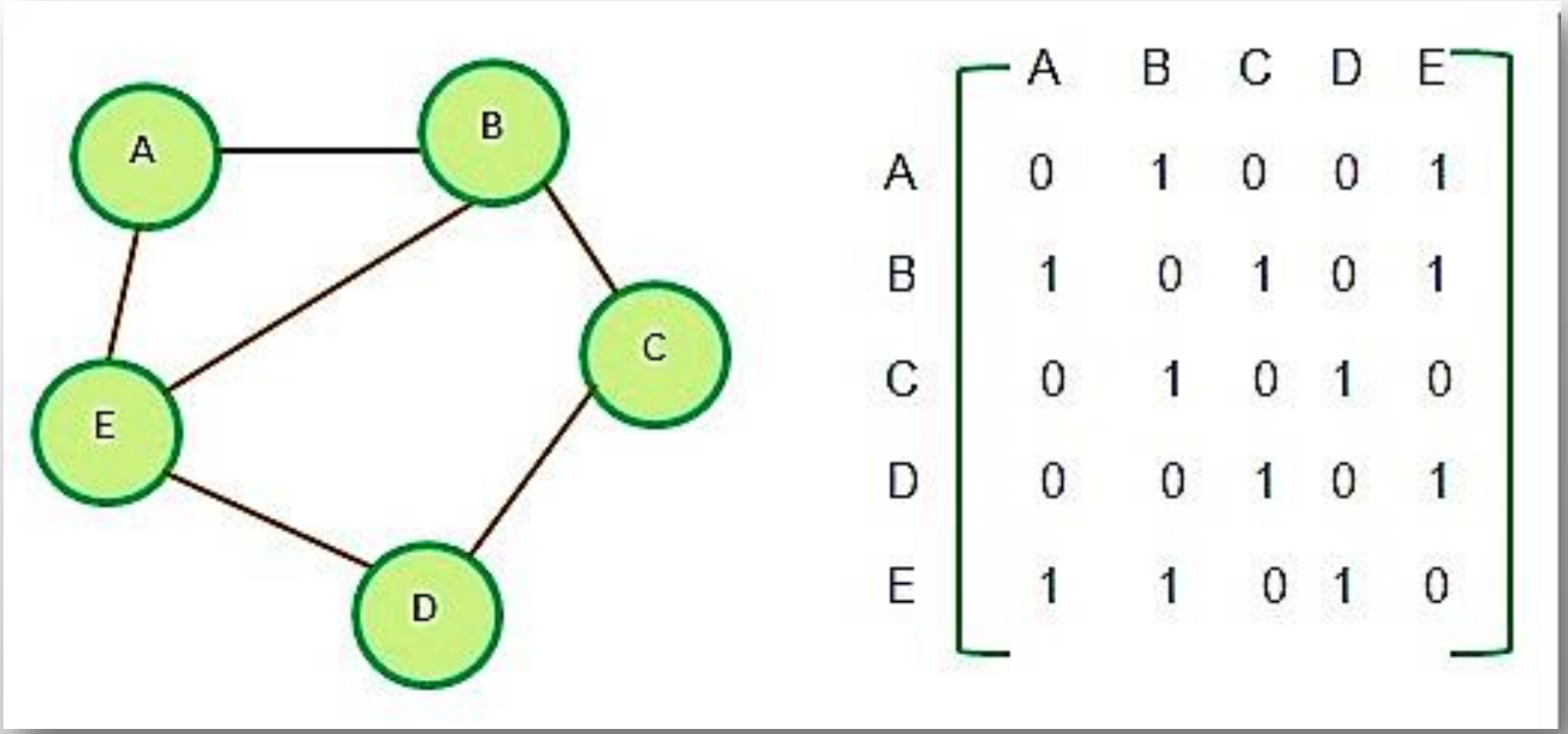
İki Temel Çizge Temsili

- Komşuluk Matrisi (Adjacency Matrix)
  - İki boyutlu bir dizidir.
  - Matrisin elemanları, iki düğüm arasında bir kenarın varlığını gösterir.
  - Örneğin, M[i][j] = 1 ise düğüm i ile düğüm j arasında bir kenar vardır.
  - Düğüm sayısı arttıkça daha fazla bellek kullanır.
- Komşuluk Listesi (Adjacency List)
  - Her düğüm için komşu düğümlerin bir listesini içerir.
  - Her düğümün listesi, o düğüme bağlı olan diğer düğümleri gösterir.
  - Örneğin, V[i] = {j, k, l} ise düğüm i'nin komşuları j, k ve l'dir.

Komşuluk Matrisi

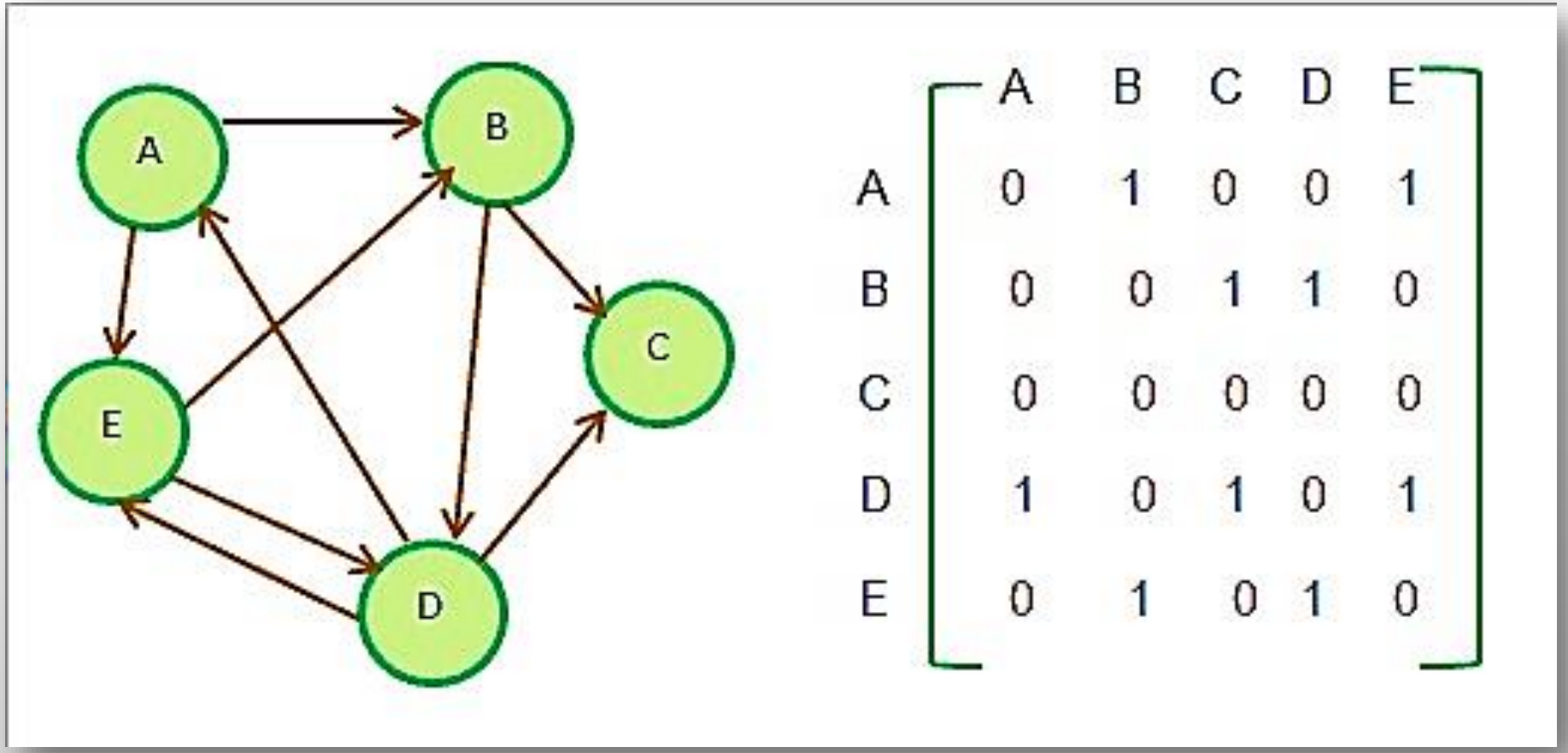
- matris[i][j], düğüm i'den j'ye bir kenar varsa 1, yoksa 0 olarak tanımlanır.
- Ağırlıklı çizgede,
  - matris[i][j] kenarı etiketleyen ağırlığı temsil eder.
  - matrix[i][j] = ∞, i'den j'ye kenar yoksa.
- Yönsüz çizgede matris simetriktir, matrix[i][j] ile matrix[j][i] eşittir.
- Matrisin alan karmaşıklığı O(|V|²)'dir (V: Düğüm sayısı).
- İki düğüm arasında kenar olup olmadığını belirlemek kolay.
- Düğümün komşularını bulmak zor.

Komşuluk Matrisi



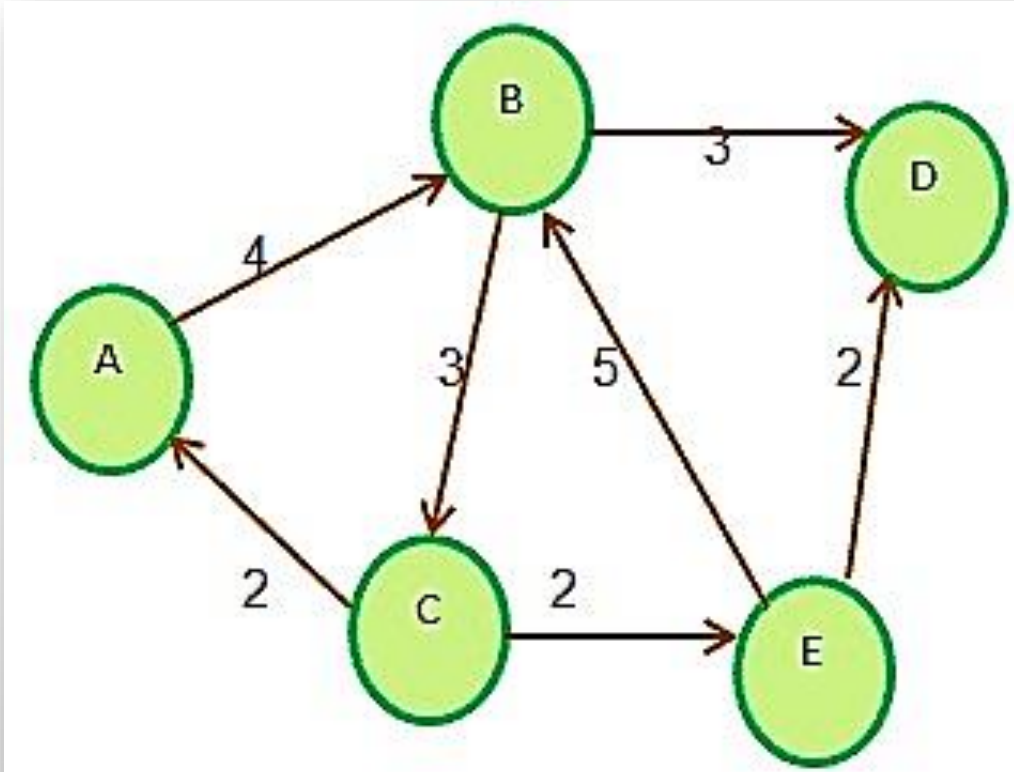
	A	B	C	D	E
A	0	1	0	0	1
B	1	0	1	0	1
C	0	1	0	1	0
D	0	0	1	0	1
E	1	1	0	1	0

Komşuluk Matrisi



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

Komşuluk Matrisi

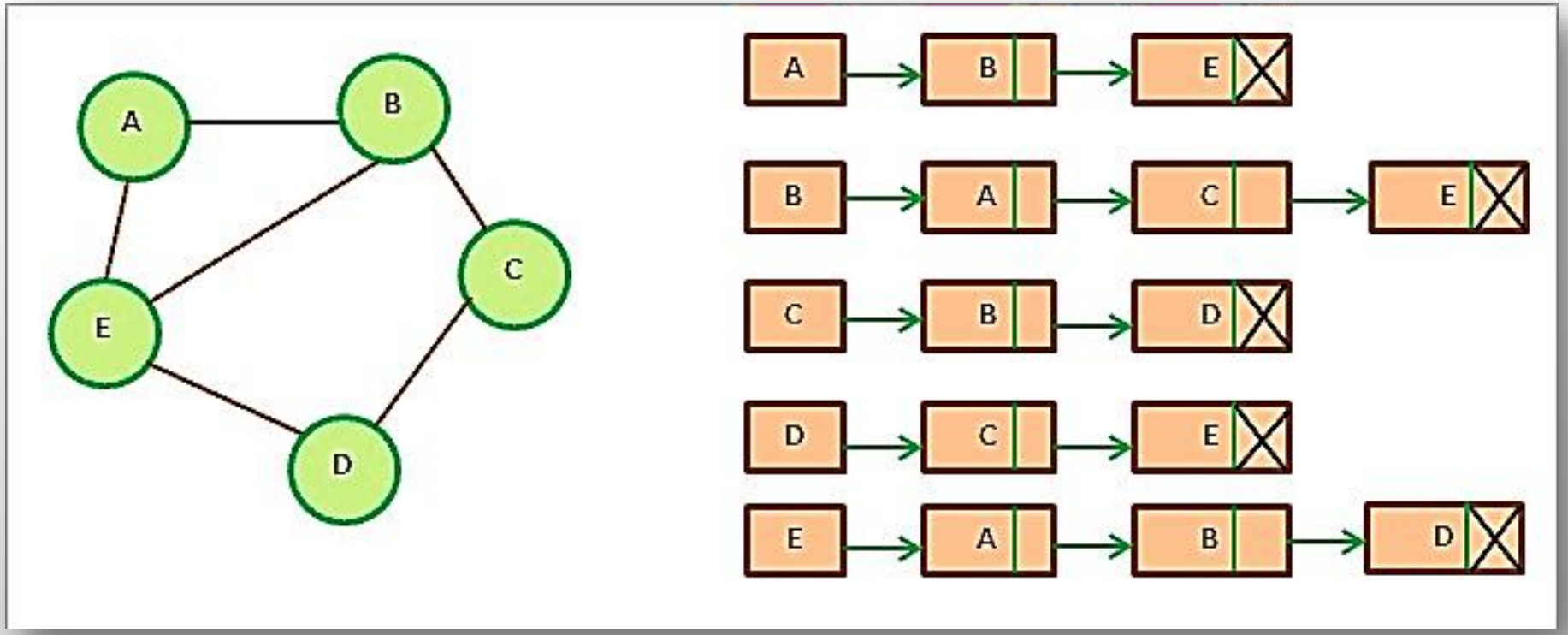


	A	B	C	D	E
A	0	4	0	0	0
B	0	0	3	3	0
C	2	0	0	0	2
D	0	0	0	0	0
E	0	5	0	2	0

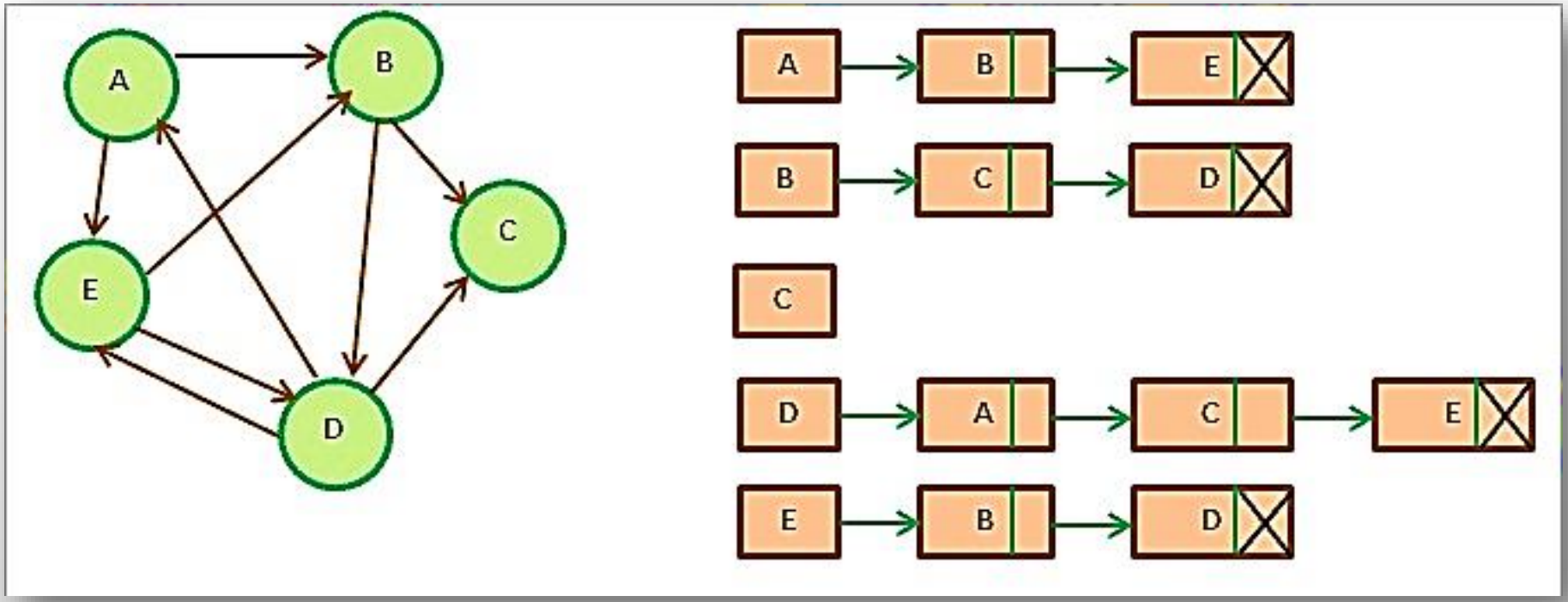
Komşuluk Listesi

- n adet düğüm içeren n bağlı listeden oluşur.
- Her düğümün kenarlarını temsil eden bağlı listeler bulunur.
- liste[i] düğümünün içeriği, düğüm i'ye komşu olan düğümleri içerir.
- Alan karmaşıklığı O(|E| + |V|)'dir (E: Kenar sayısı, V: Düğüm sayısı).
- Yönsüz çizgede her kenar (v,w) iki listeye de eklenir.
- Düğümün komşu düğümlerini bulmak kolay.
- İki düğüm arasında kenarın olup olmadığını belirlemek zor.

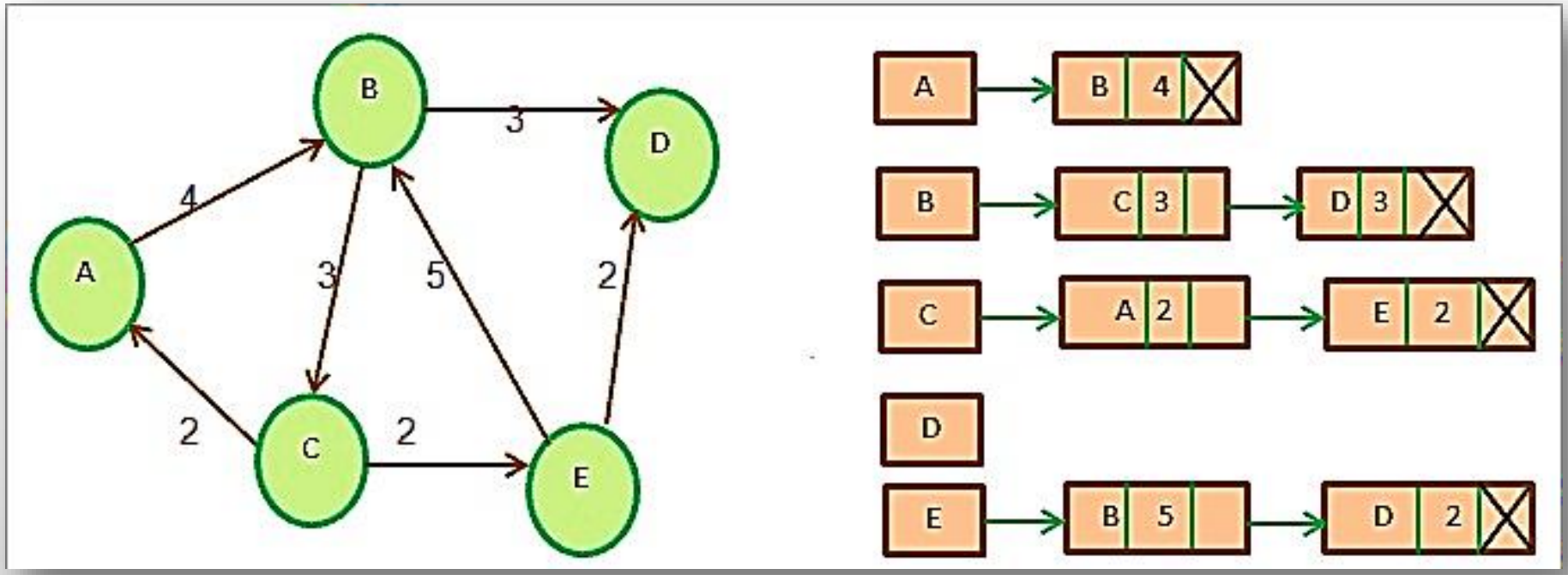
Komşuluk Listesi



Komşuluk Listesi



Komşuluk Listesi



Gezinme Algoritmaları

- Genişlik Öncelikli Arama (BFS)
- Derinlik Öncelikli Arama (DFS)

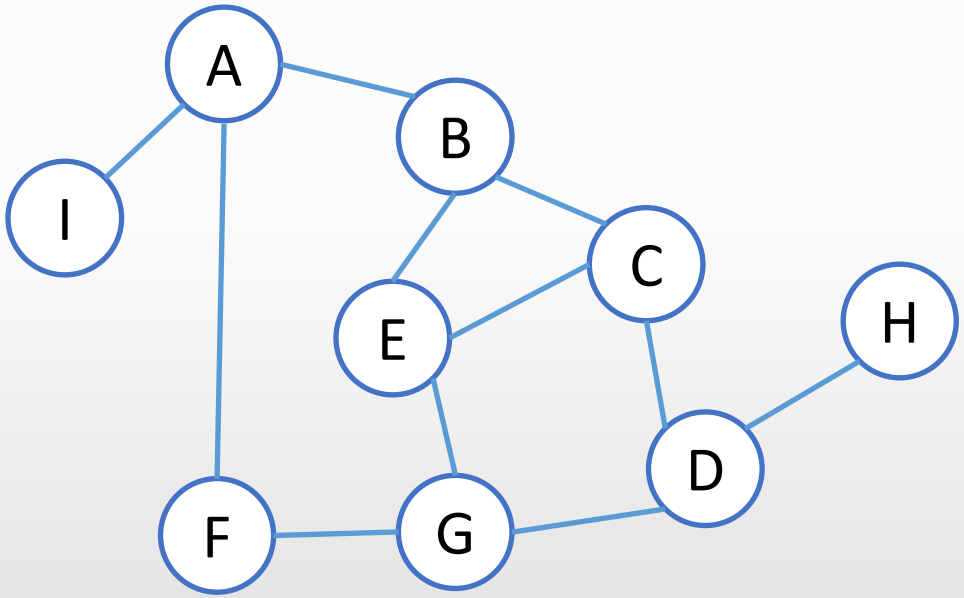
Genişlik Öncelikli Arama (BFS)

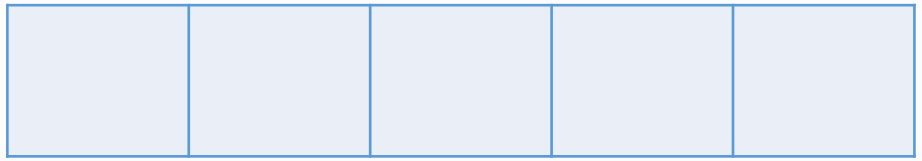
- Düğümleri seviye seviye gezerek keşfeden bir arama algoritmasıdır.
- Yakın komşulardan başlayarak tüm düğümleri gezer.
- Kuyruk yapısını kullanır.
- Her düğümü sadece bir kez ziyaret eder.
- İki düğüm arasında en kısa yolu bulmada kullanılır.
- Yakın düğümleri hızlı keşfeder.
- Bellek kullanımı yüksek olabilir.

Genişlik Öncelikli Arama Adımları

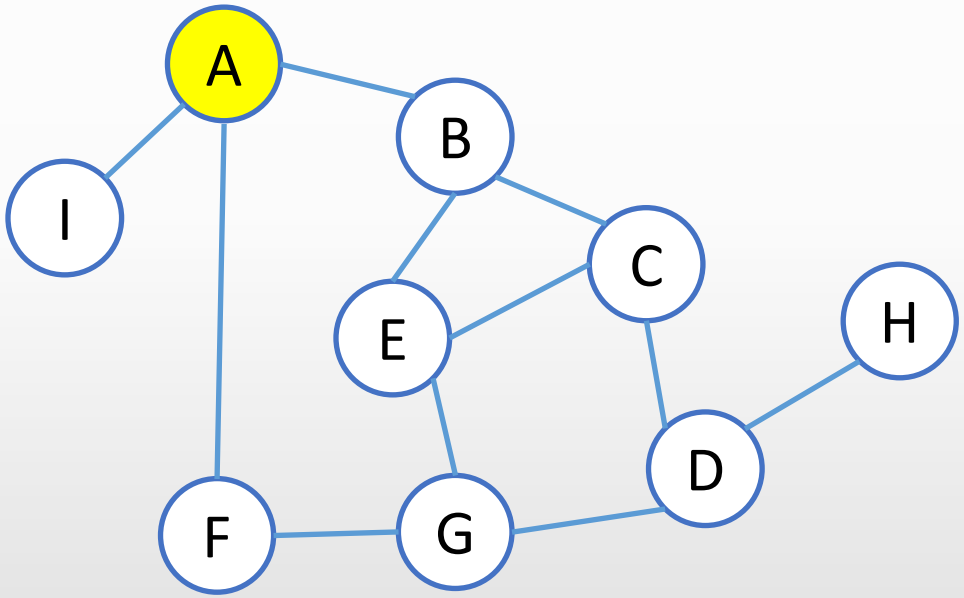
```
fonksiyon genislikOncelikliArama(baslangicDugumu) {
    kuyruk.ekle(baslangicDugumu);
    baslangicDugumu.ziyaretEdildi();
    while (!kuyruk.bosMu()) {
        w = kuyruk.cikar();
        for (w'nin ziyaret edilmemiş tüm komşuları) {
            u.ziyaretEdildi();
            kuyruk.ekle(u);
        }
    }
}
```

Genişlik Öncelikli Arama (BFS)



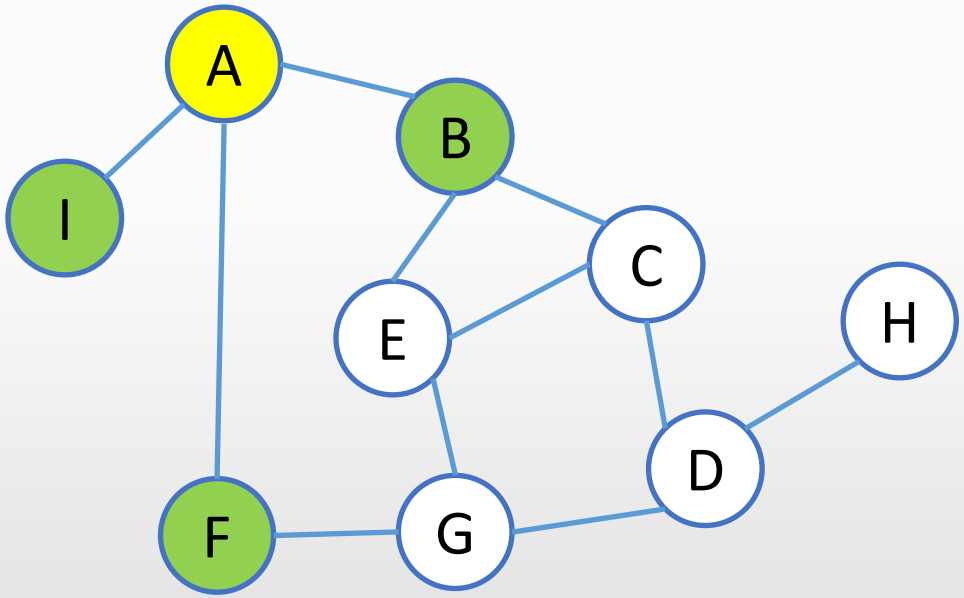


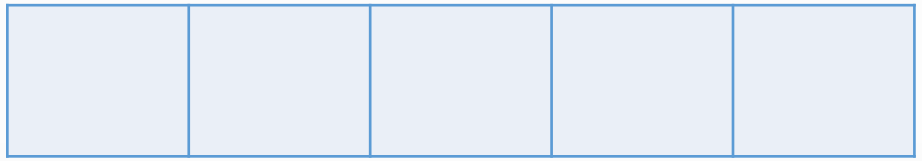
Genişlik Öncelikli Arama (BFS)



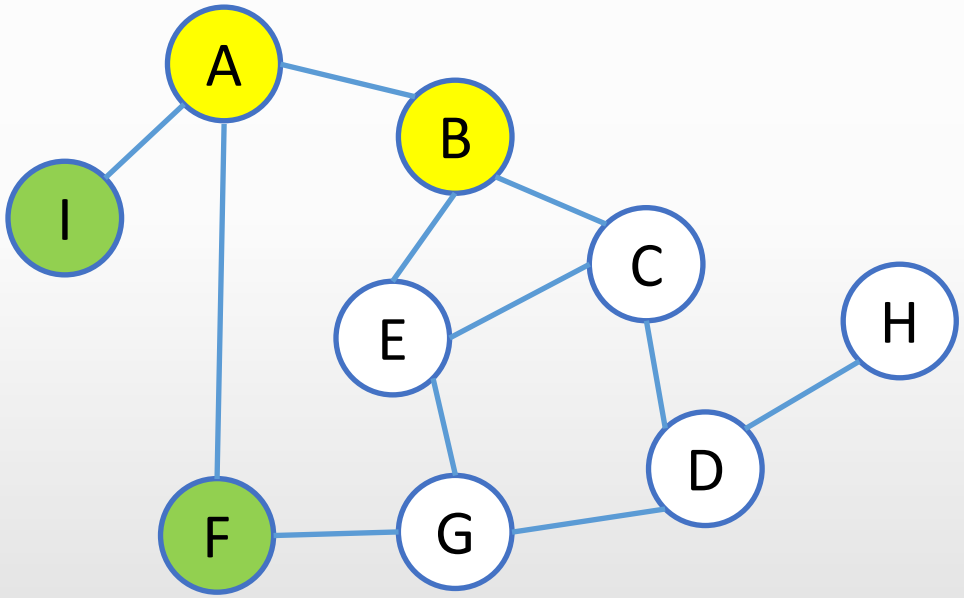
A				

Genişlik Öncelikli Arama (BFS)



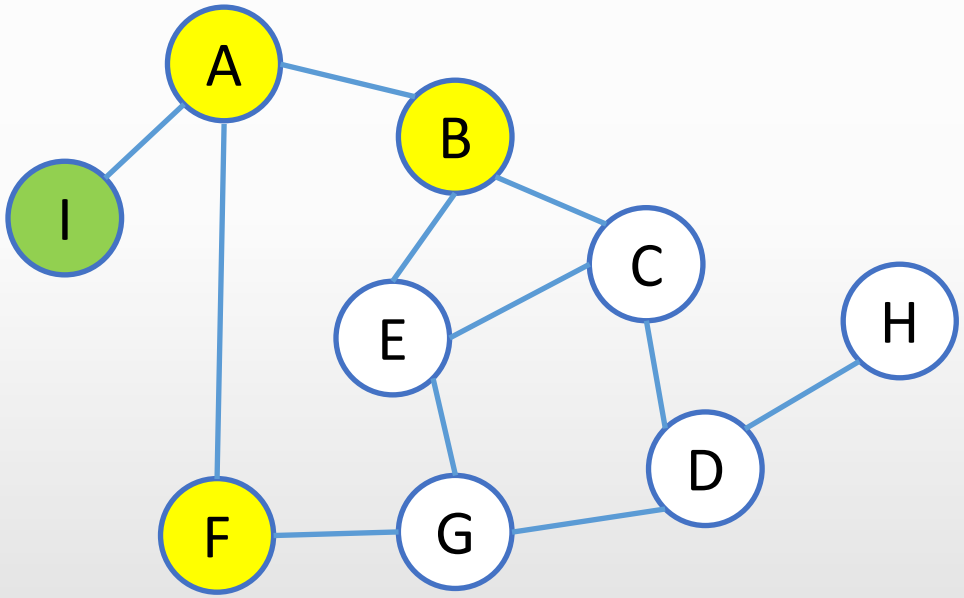


Genişlik Öncelikli Arama (BFS)



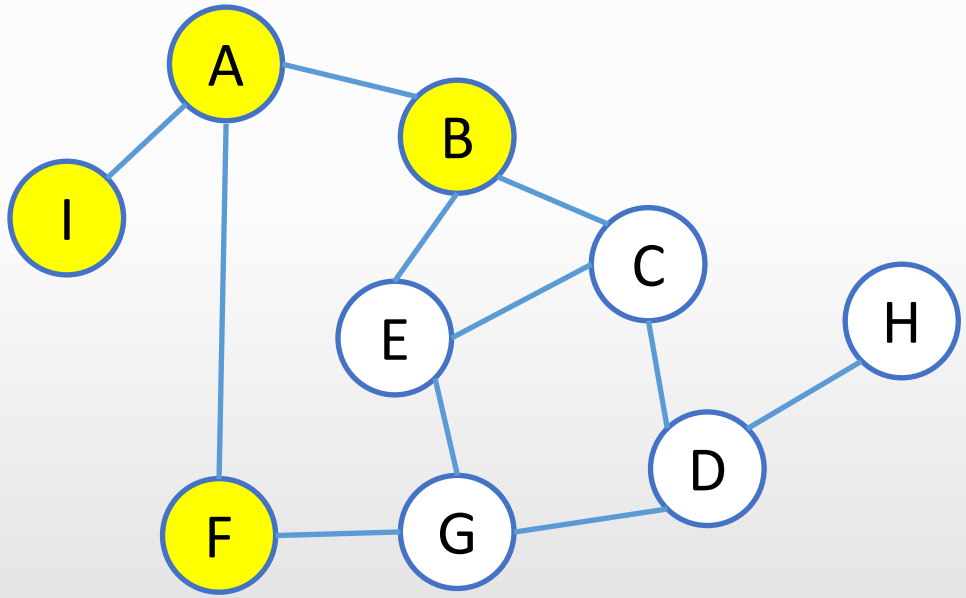
B				

Genişlik Öncelikli Arama (BFS)



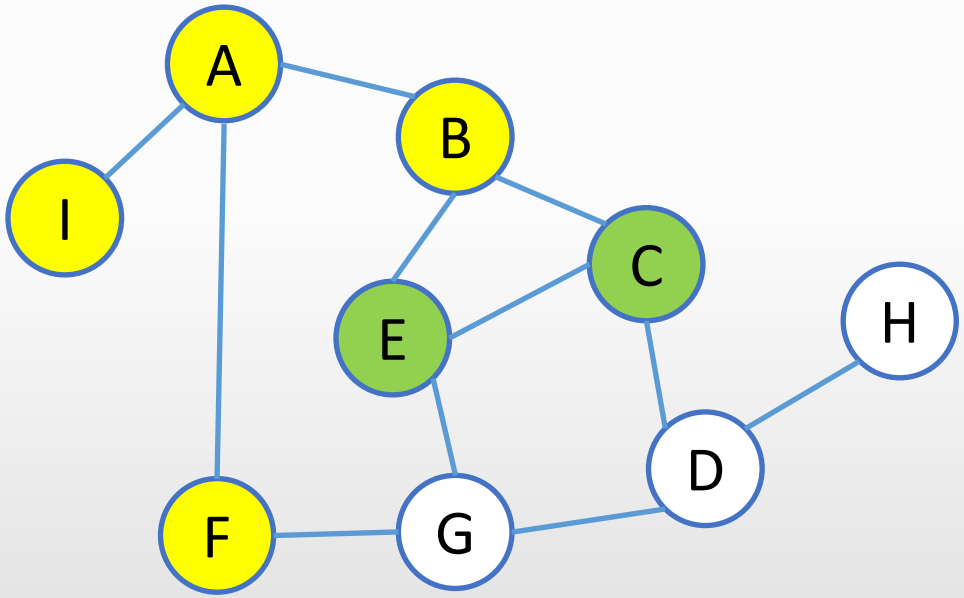
B	F			

Genişlik Öncelikli Arama (BFS)



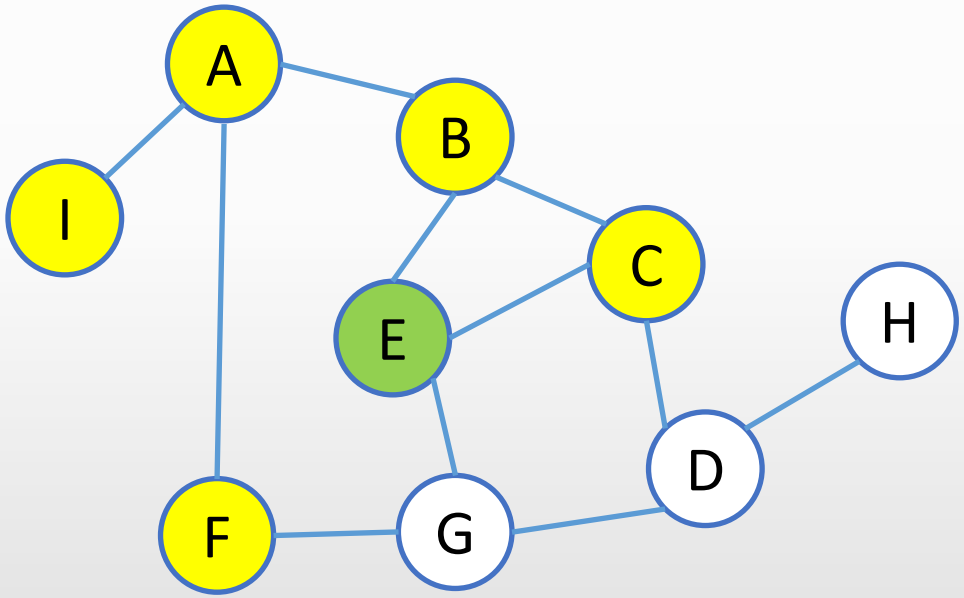
B	F	I		

Genişlik Öncelikli Arama (BFS)



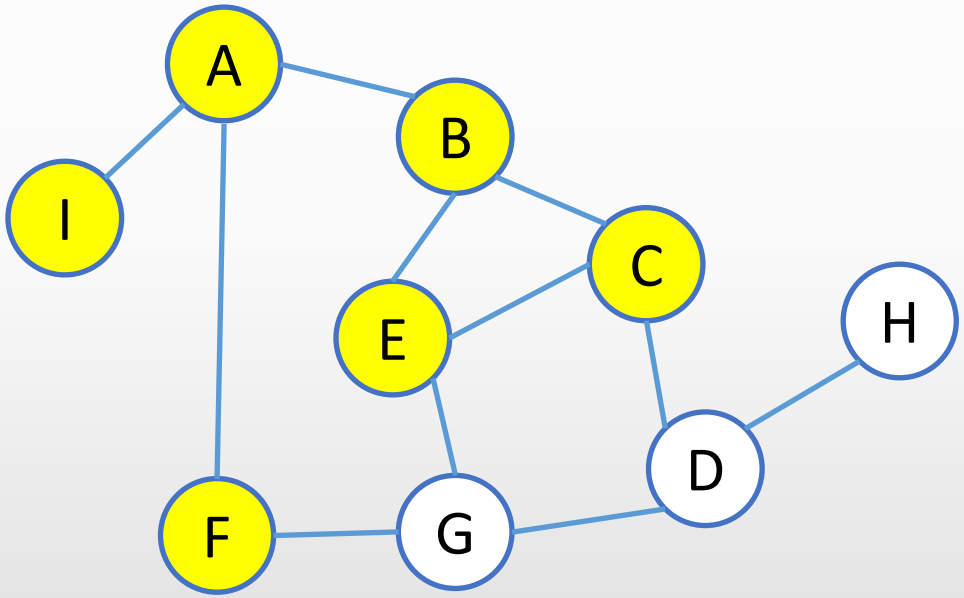
F	I			

Genişlik Öncelikli Arama (BFS)



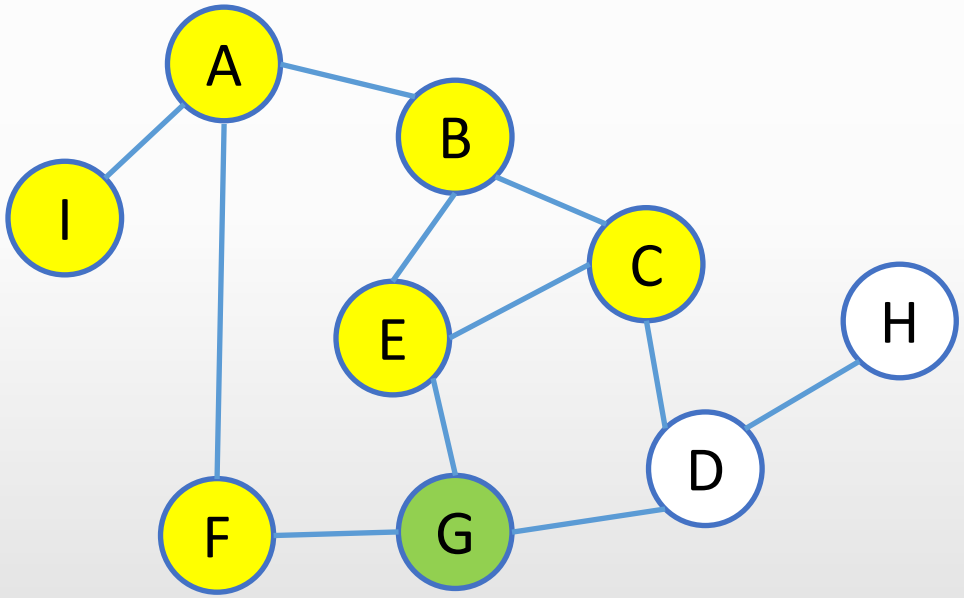
F	I	C		

Genişlik Öncelikli Arama (BFS)



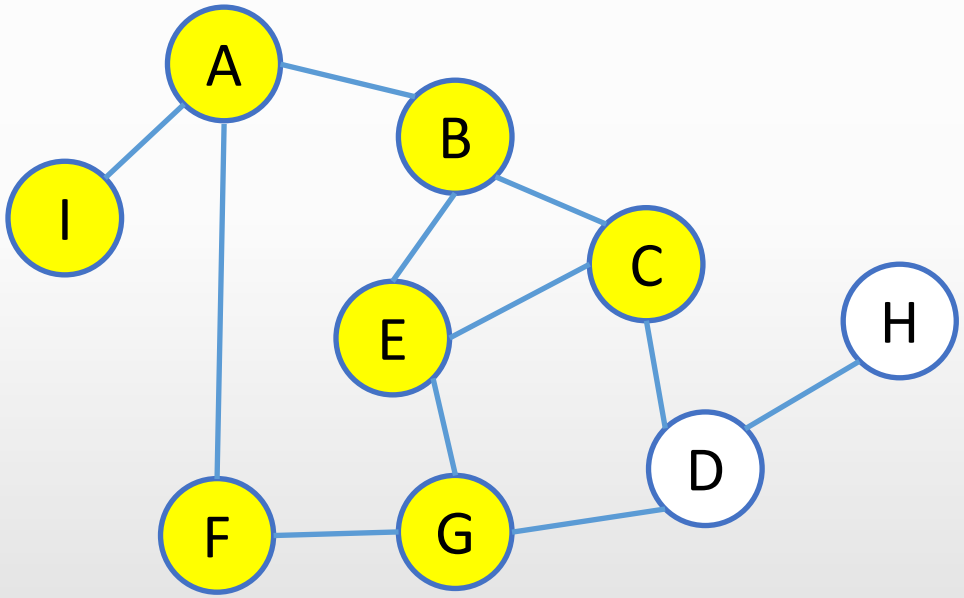
F	I	C	E	

Genişlik Öncelikli Arama (BFS)



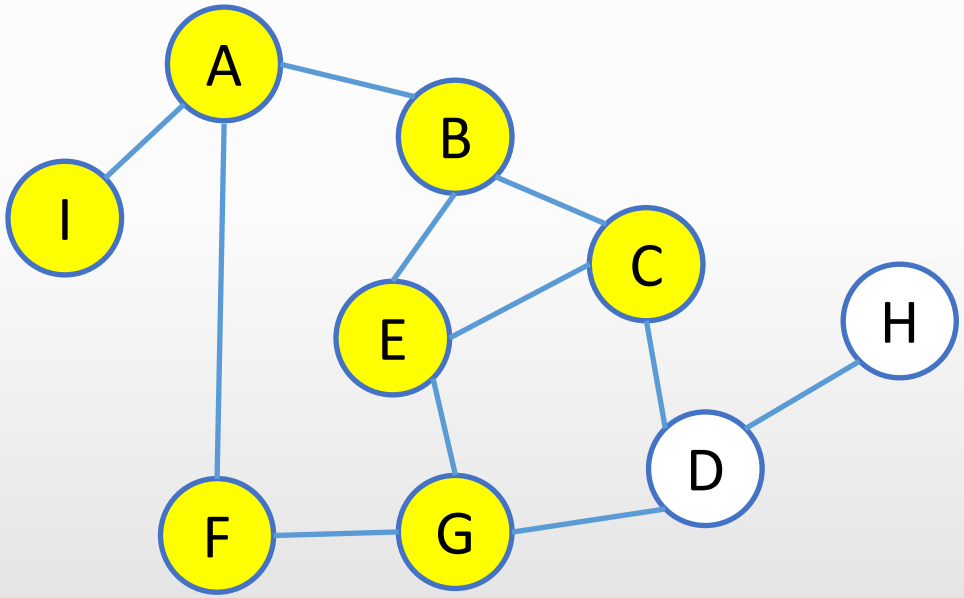
I	C	E		

Genişlik Öncelikli Arama (BFS)



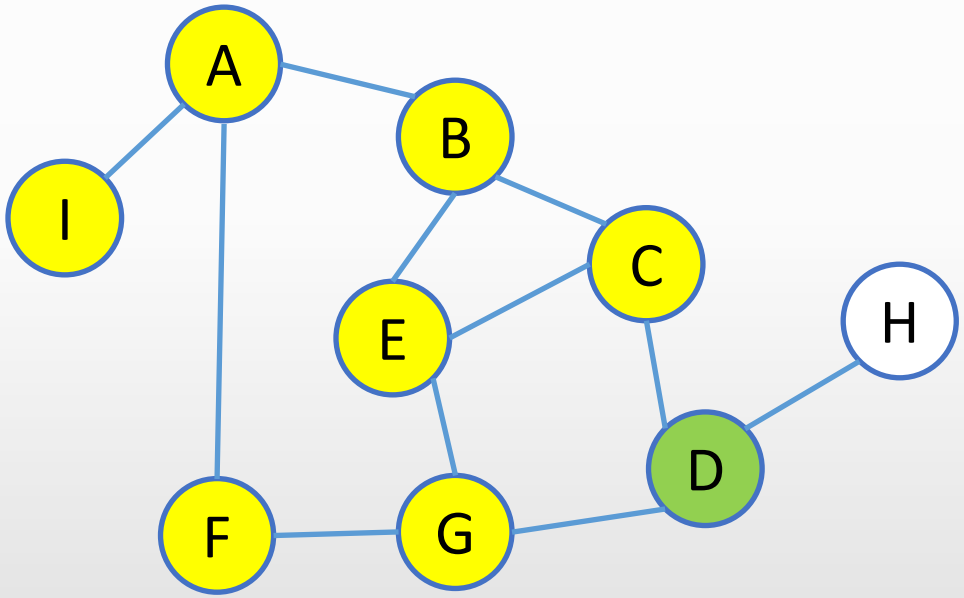
I	C	E	G	

Genişlik Öncelikli Arama (BFS)



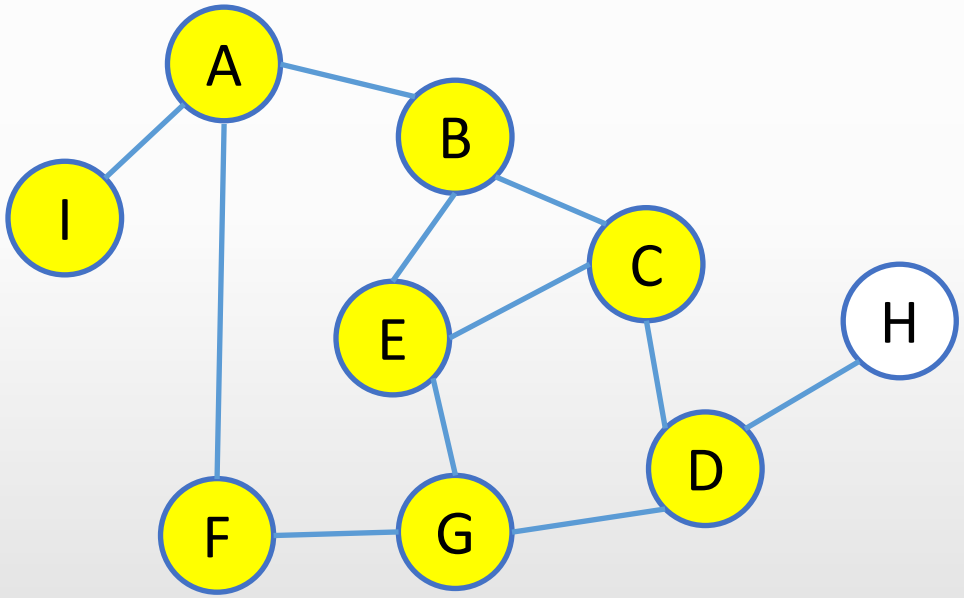
C	E	G		

Genişlik Öncelikli Arama (BFS)



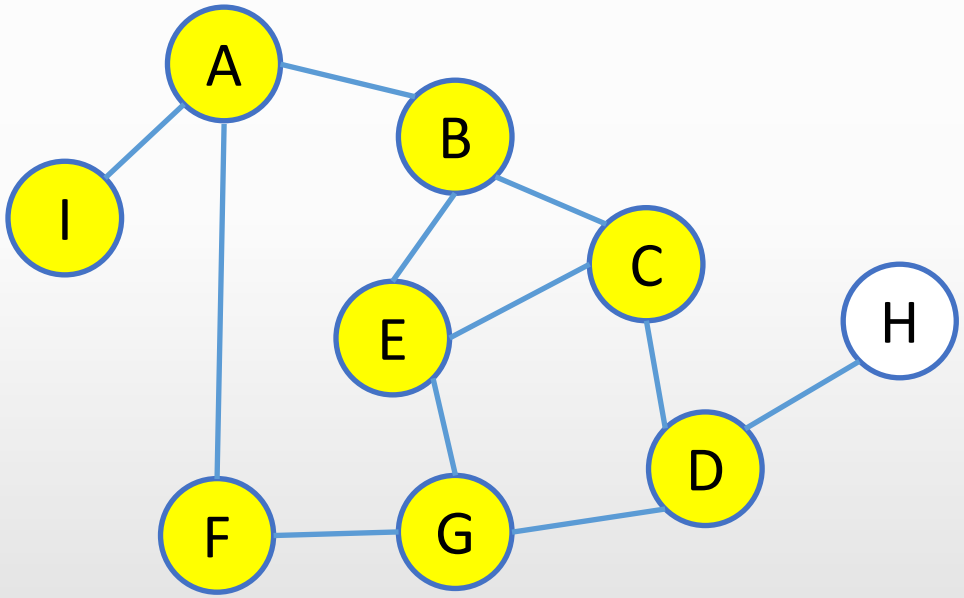
E	G			

Genişlik Öncelikli Arama (BFS)



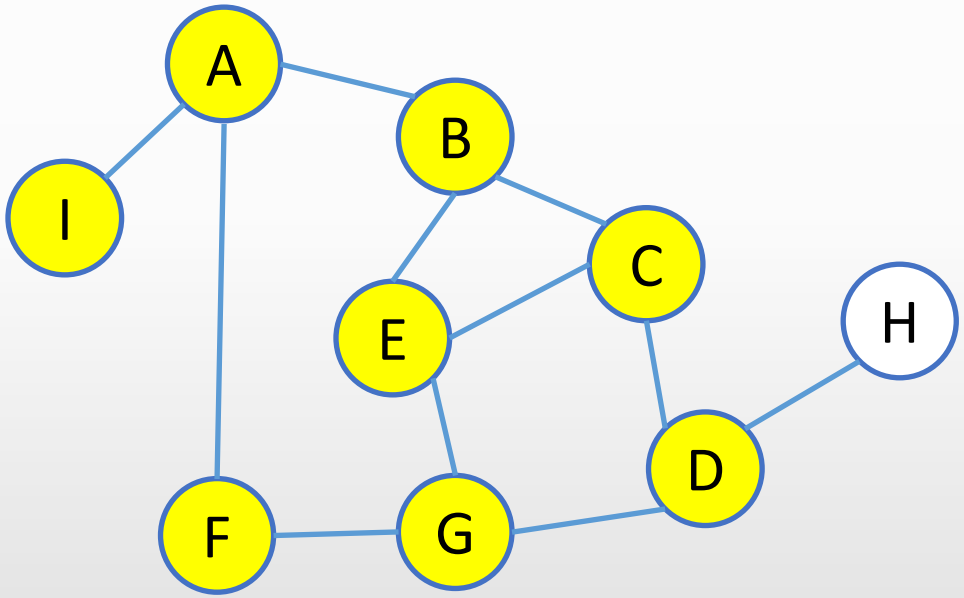
E	G	D		

Genişlik Öncelikli Arama (BFS)



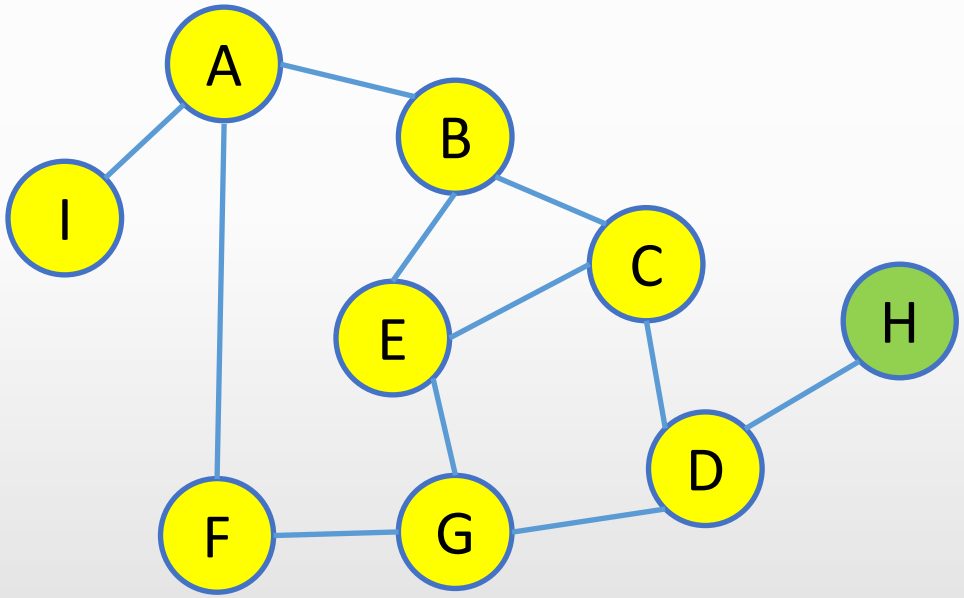
G	D			

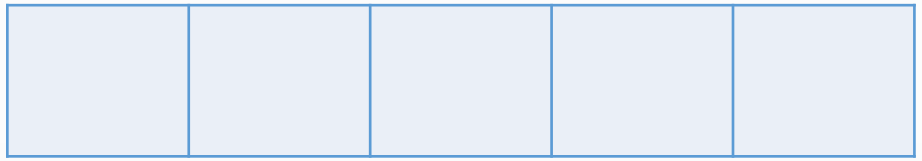
Genişlik Öncelikli Arama (BFS)



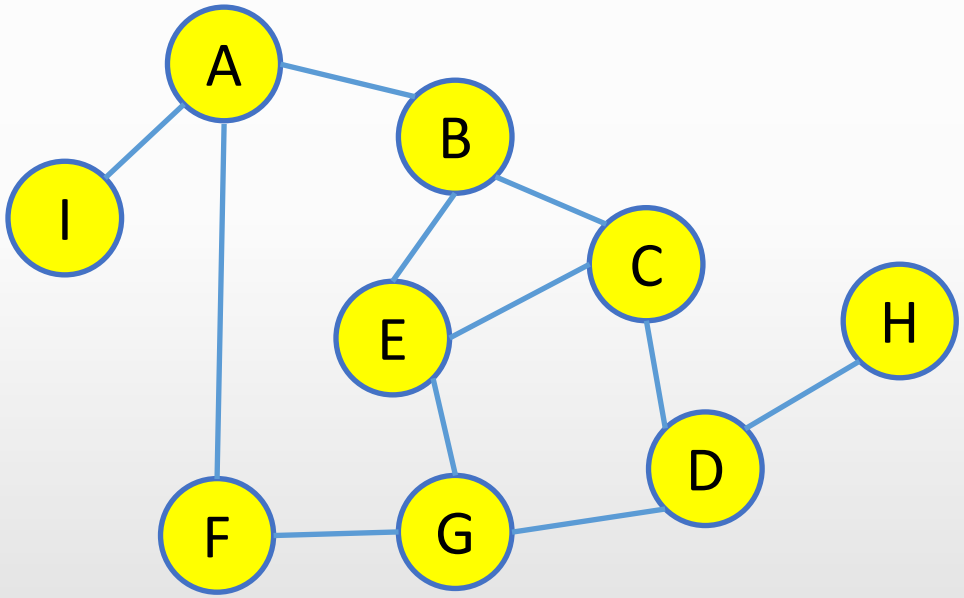
D				

Genişlik Öncelikli Arama (BFS)



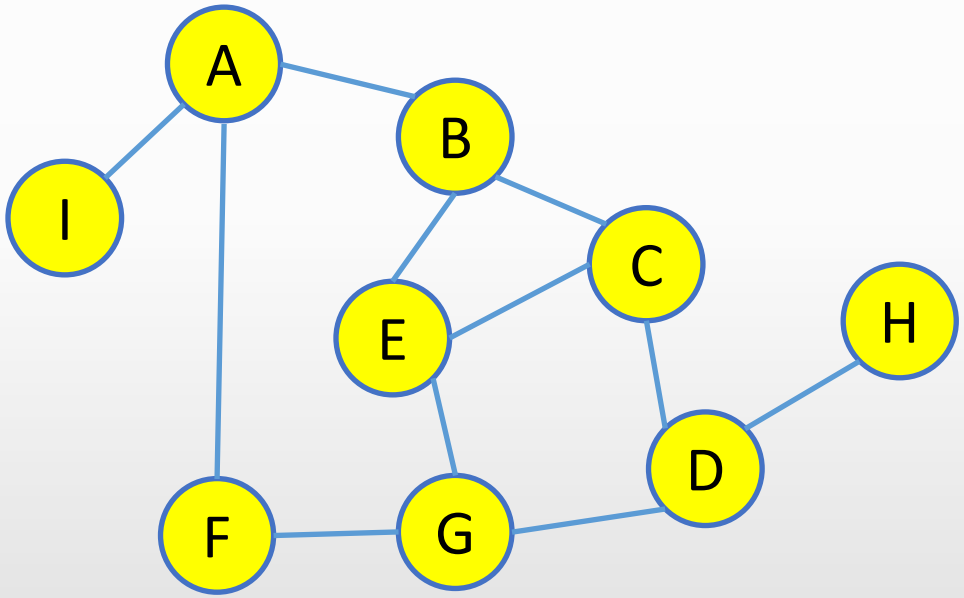


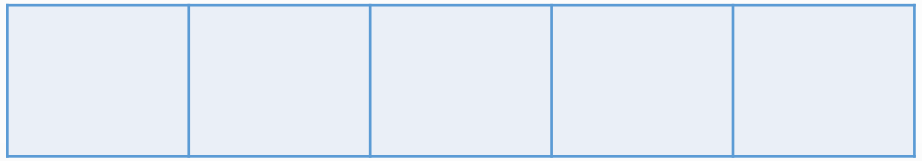
Genişlik Öncelikli Arama (BFS)



H				

Genişlik Öncelikli Arama (BFS)





Derinlik Öncelikli Arama (DFS)

- Düğümleri bir yol boyunca keşfeden bir arama algoritmasıdır.
- Derinlik önceliği olan bir yaklaşımı temsil eder.
- Yığın (stack) yapısı veya özyinelemeli çağrılar kullanılır.
- Her düğüm sadece bir kez ziyaret edilir.
- Düğümler derinlemesine keşfedilir.
- Bellek kullanımı düşüktür.
- Bir yolun sonuna kadar gitme yeteneği sağlar.

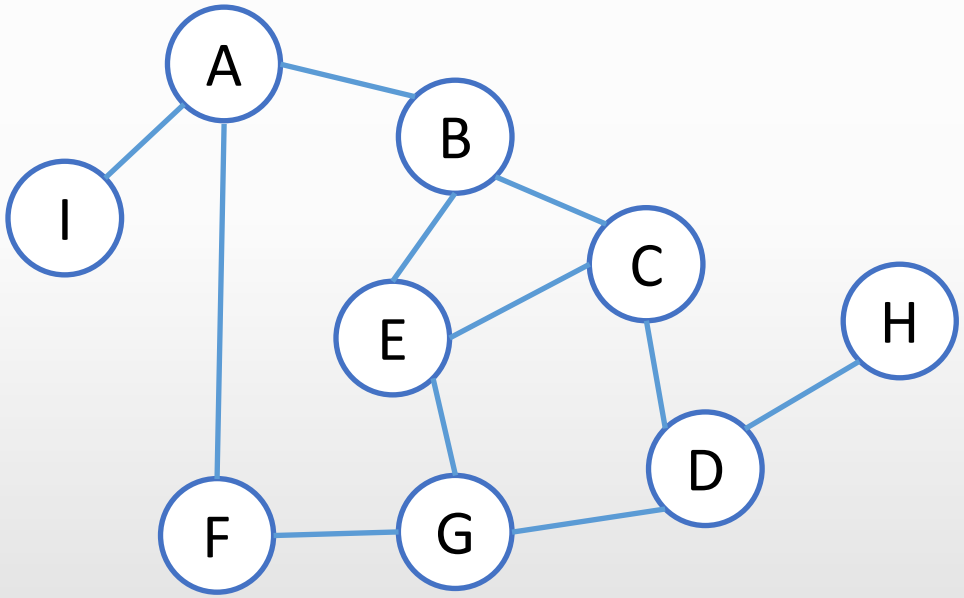
Derinlik Öncelikli Arama Adımları

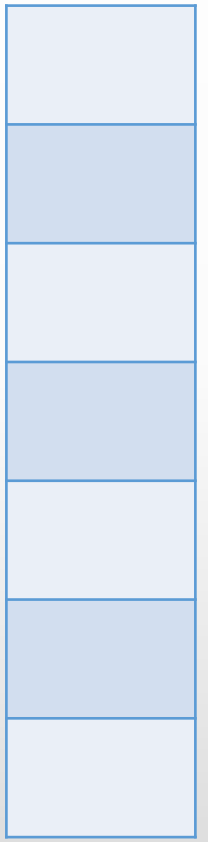
```
fonksiyon derinlikOncelikliArama(baslangic) {
    // Özyinelemeli Versiyon
    baslangic.ziyaretEdildi();

    for (baslangic düğümün ziyaret edilmemiş tüm komşuları) {
        derinlikOncelikliArama(komşu);
    }
}
```

Derinlik Öncelikli Arama Adımları

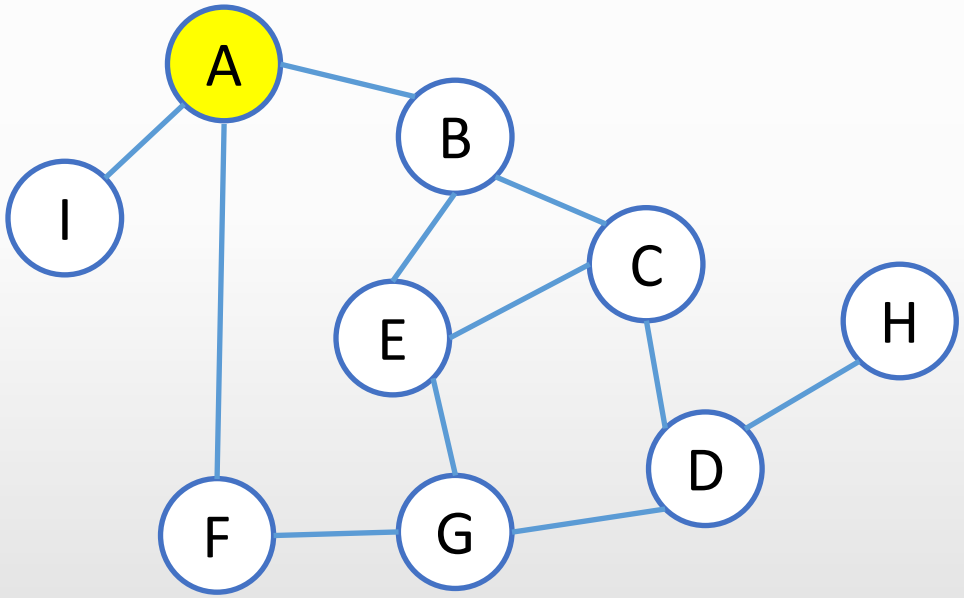
```
fonksiyon derinlikOncelikliArama(baslangic) {
    yigin.ekle(baslangic);
    baslangic.ziyaretEdildi();
    while (!yigin.bosMu()) {
        if (yığındaki düğümün tüm komşuları ziyaret edildi) {
            yigin.cikar();  // geri dön
        } else {
            komsu = ziyaret edilmemiş bir düğüm seç;
            yigin.ekle(komsu);
            komsu.ziyaretEdildi();
        }
    }
}
```

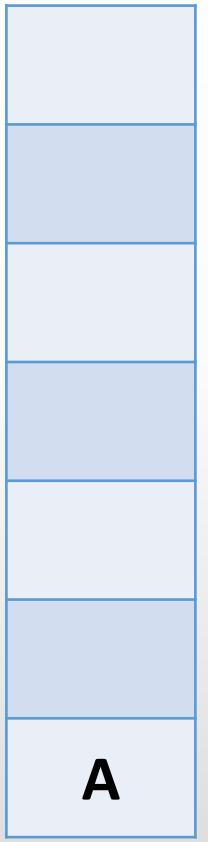
Derinlik Öncelikli Arama (DFS)



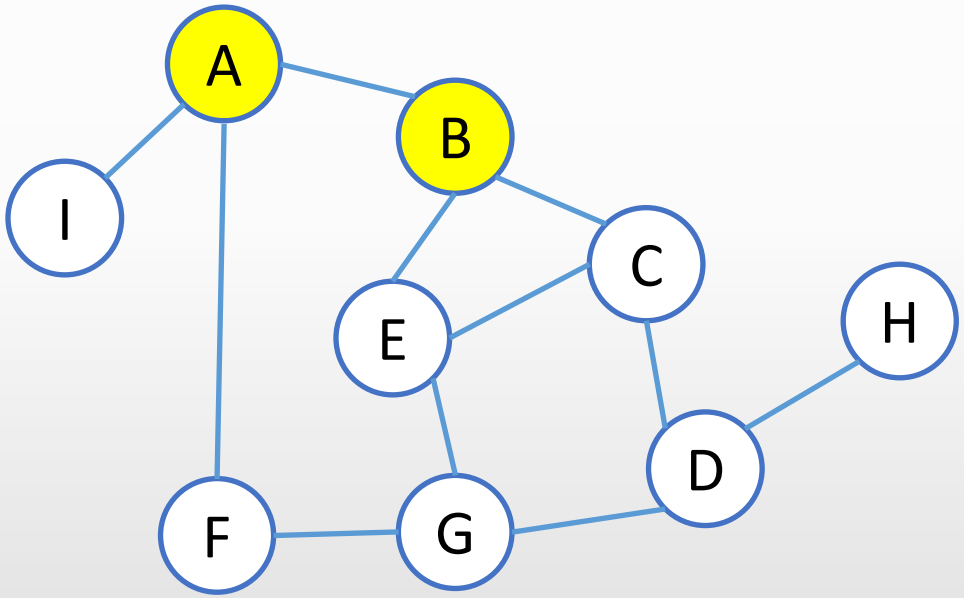


Derinlik Öncelikli Arama (DFS)

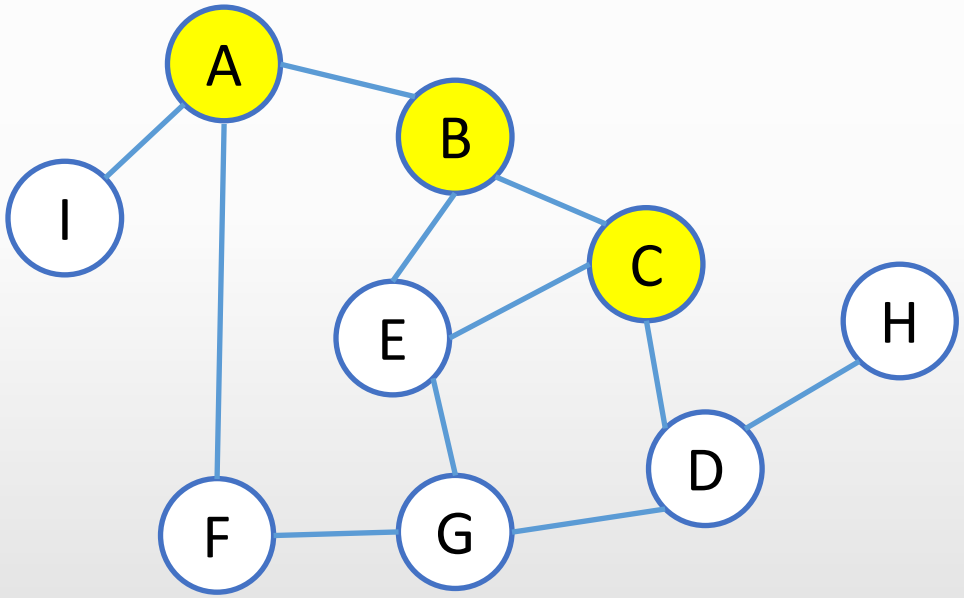


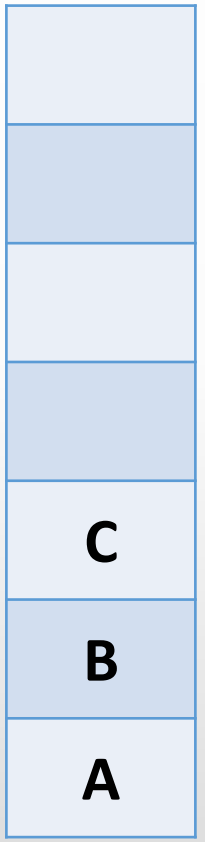


Derinlik Öncelikli Arama (DFS)

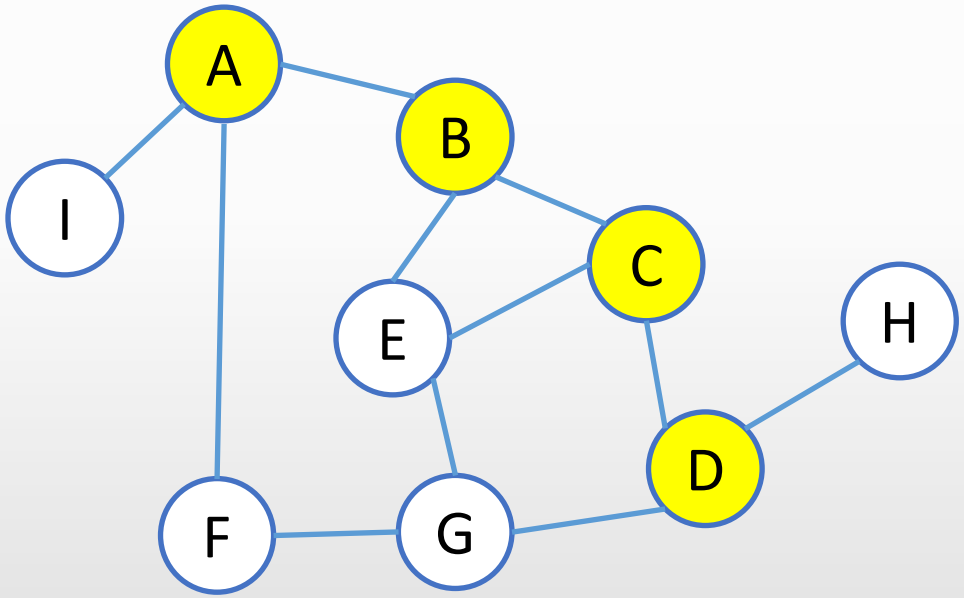


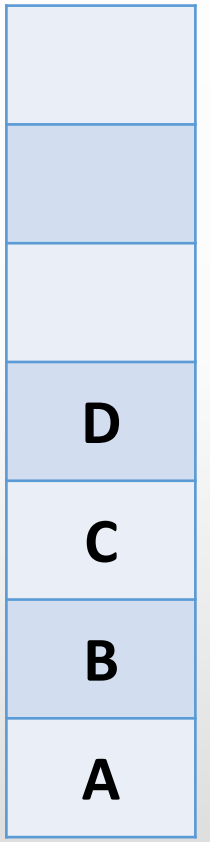
Derinlik Öncelikli Arama (DFS)



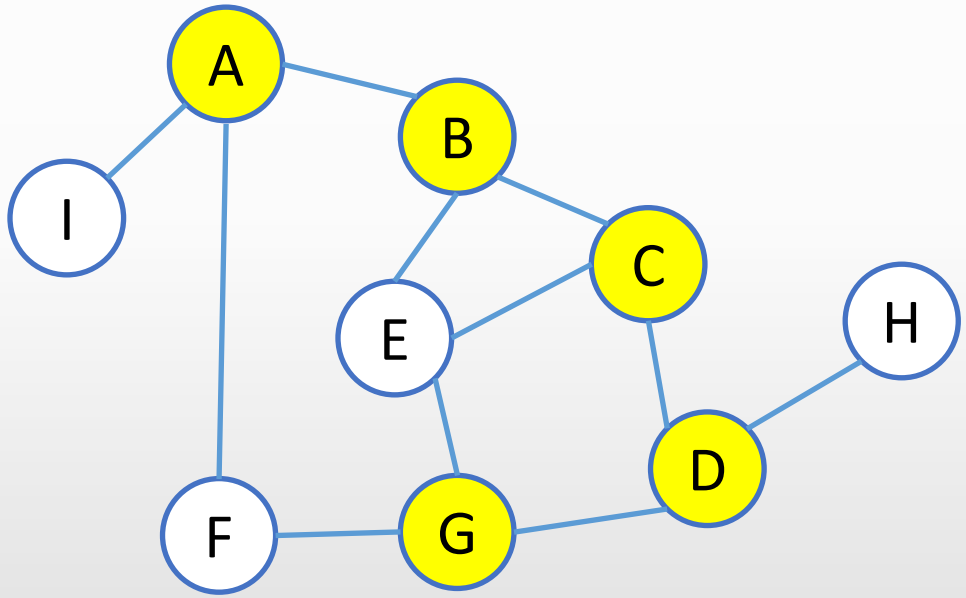


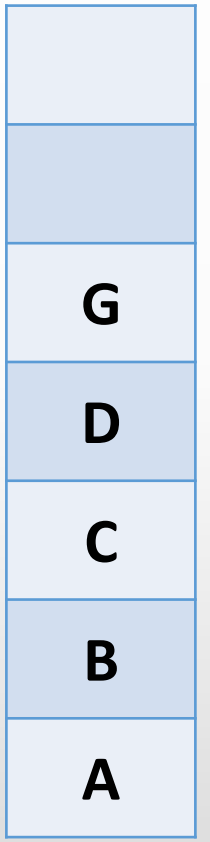
Derinlik Öncelikli Arama (DFS)



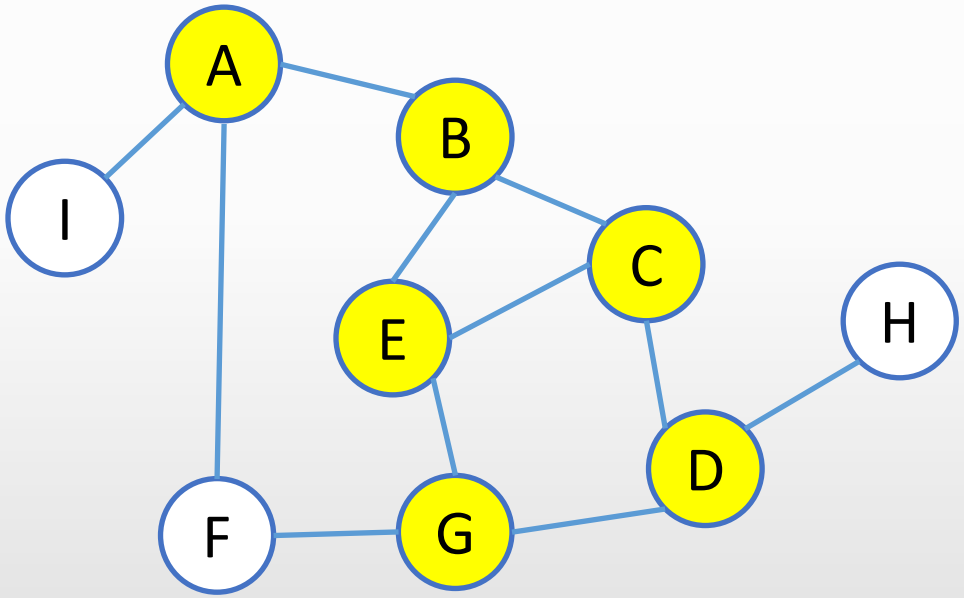


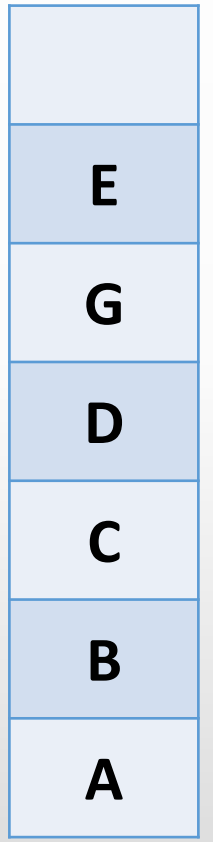
Derinlik Öncelikli Arama (DFS)



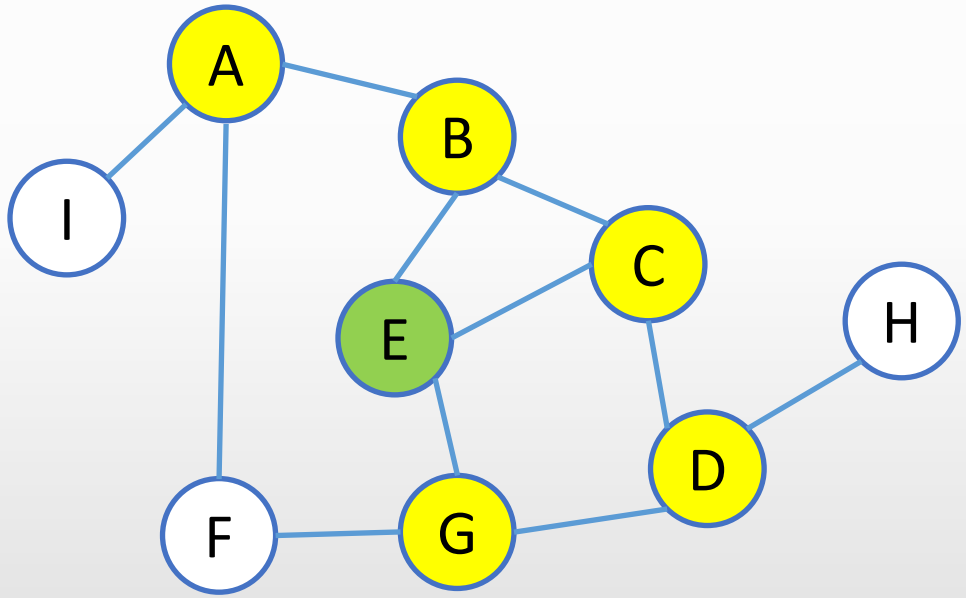


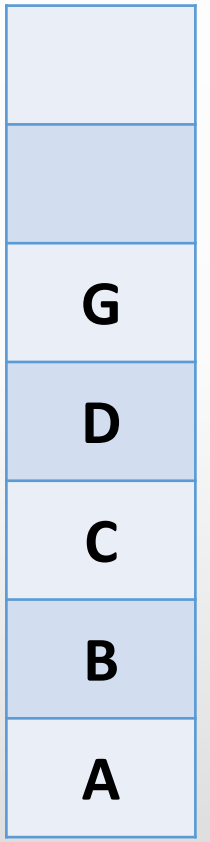
Derinlik Öncelikli Arama (DFS)



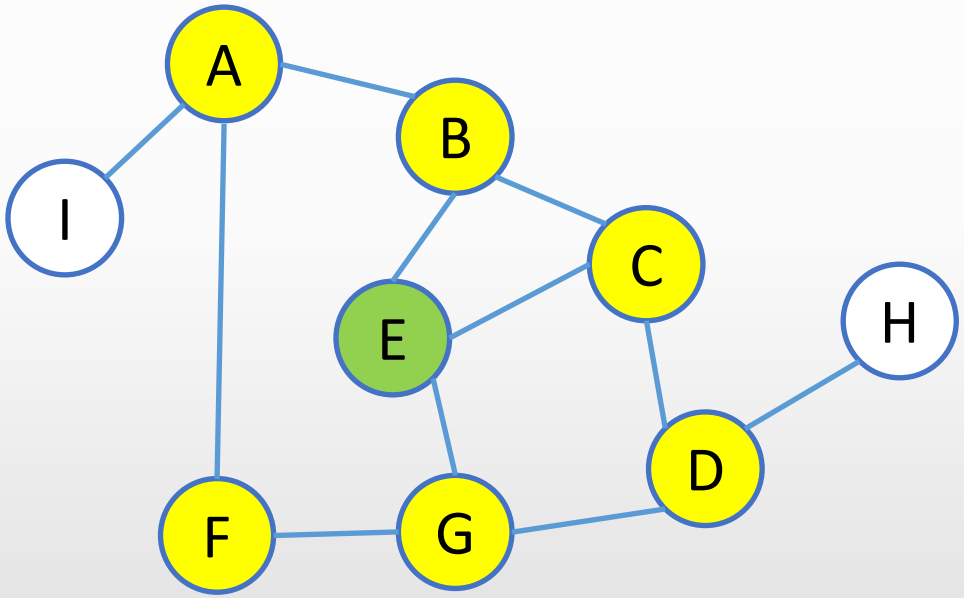


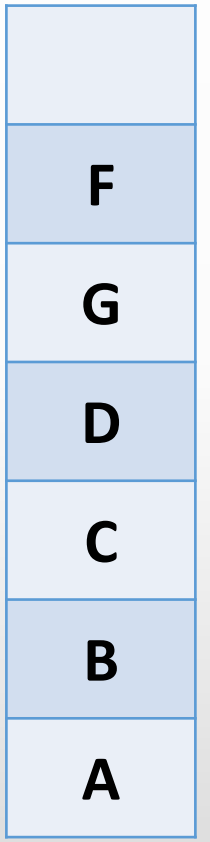
Derinlik Öncelikli Arama (DFS)



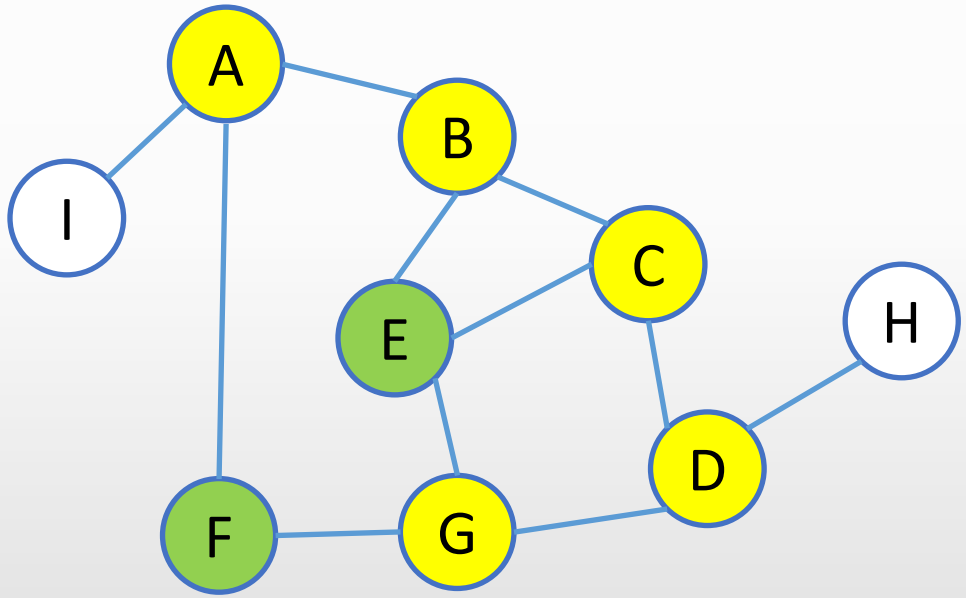


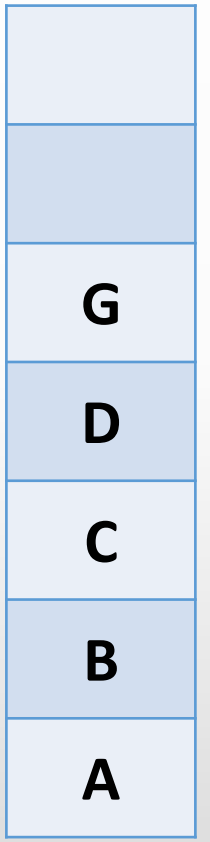
Derinlik Öncelikli Arama (DFS)



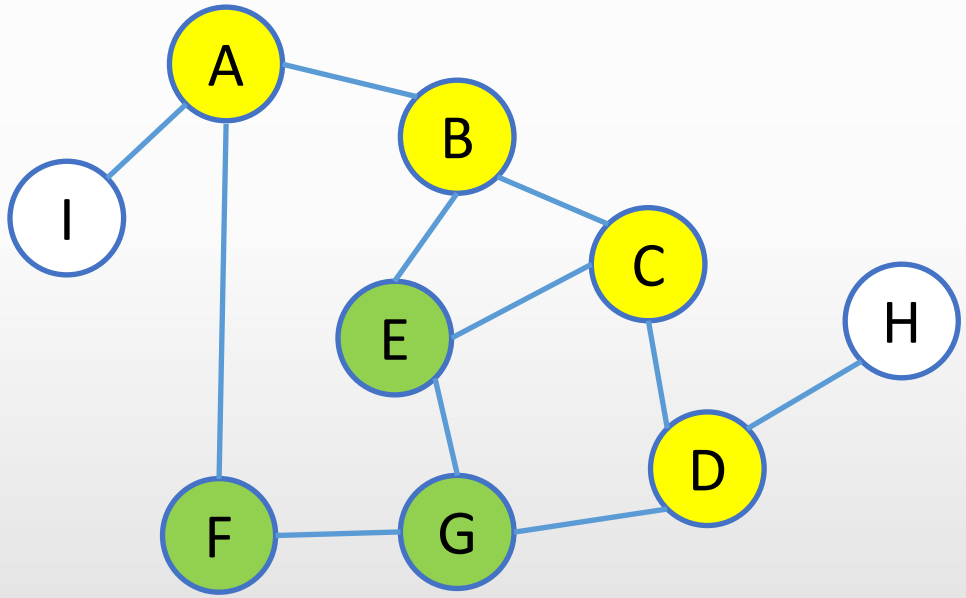


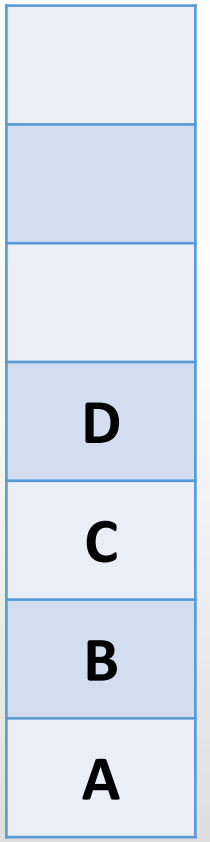
Derinlik Öncelikli Arama (DFS)



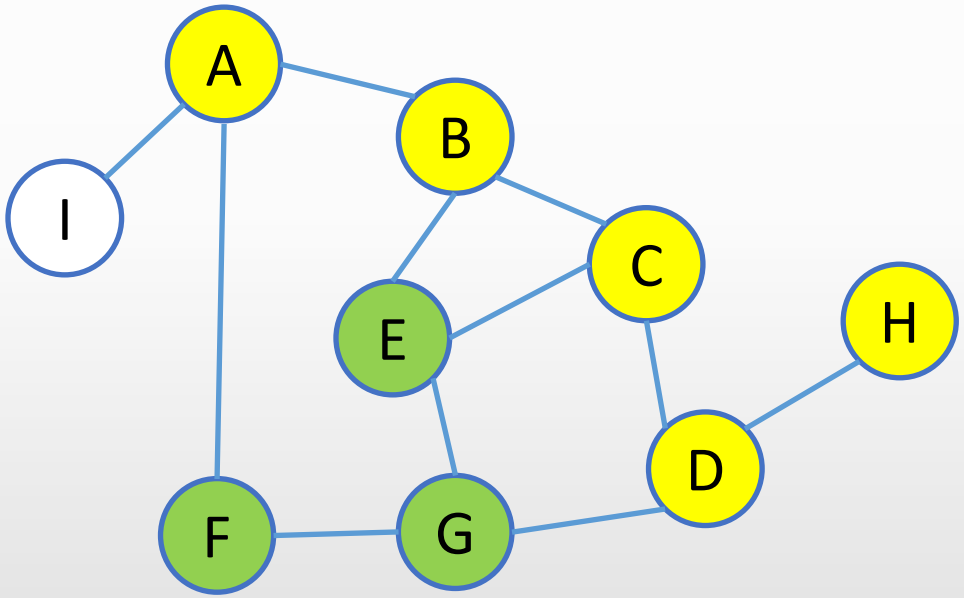


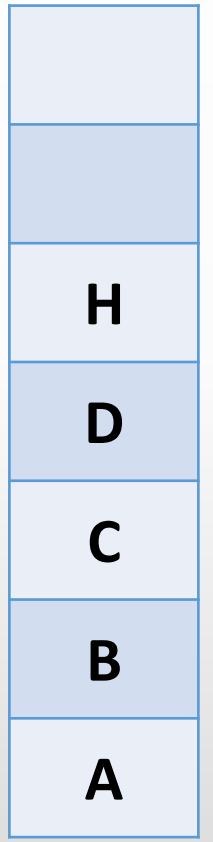
Derinlik Öncelikli Arama (DFS)



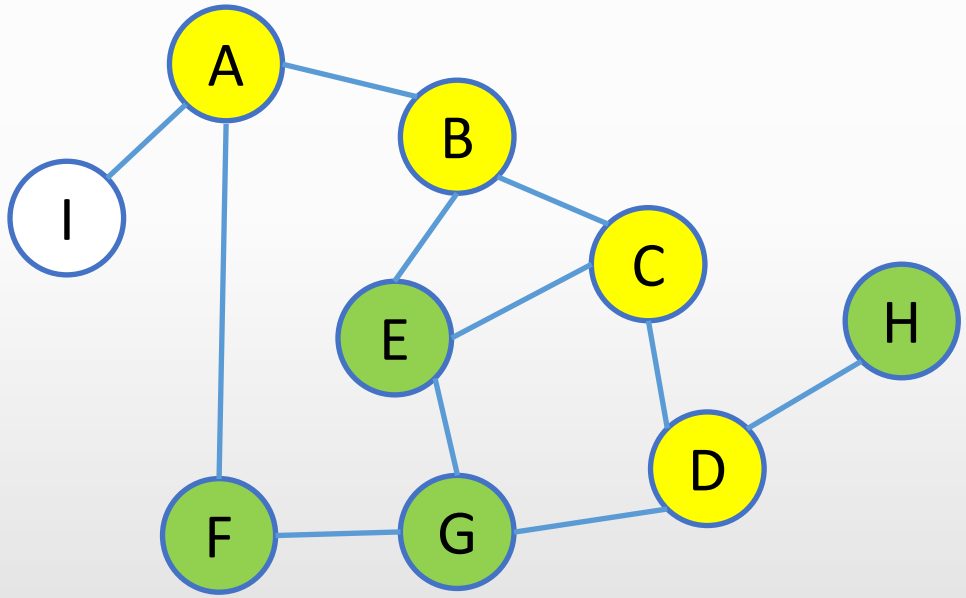


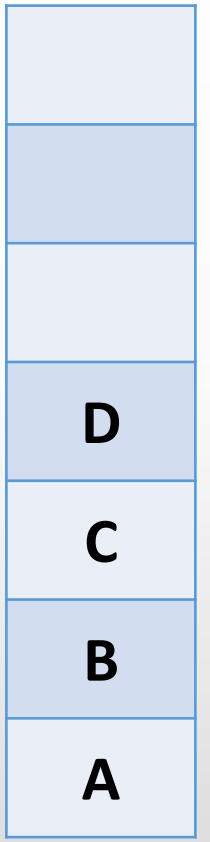
Derinlik Öncelikli Arama (DFS)



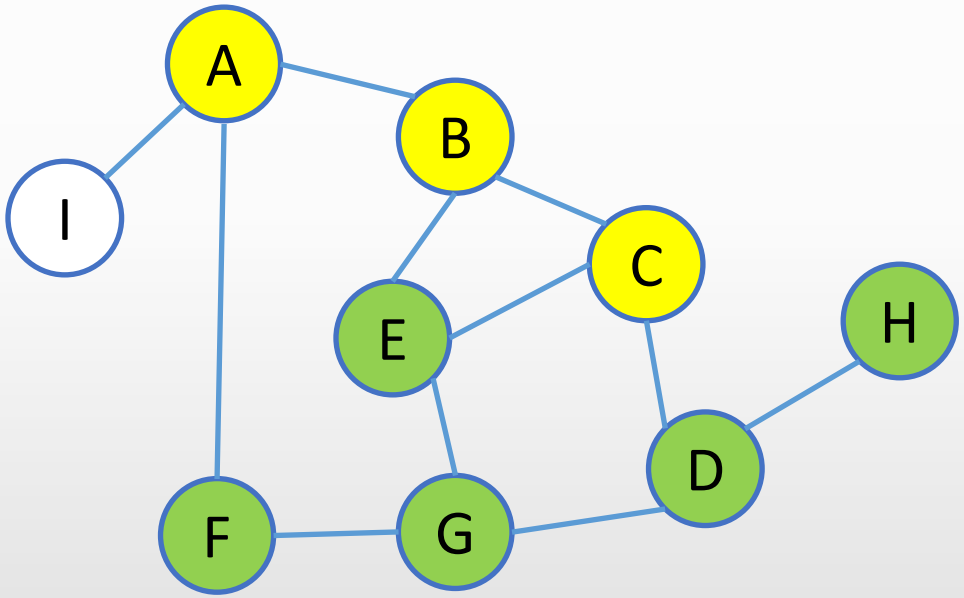


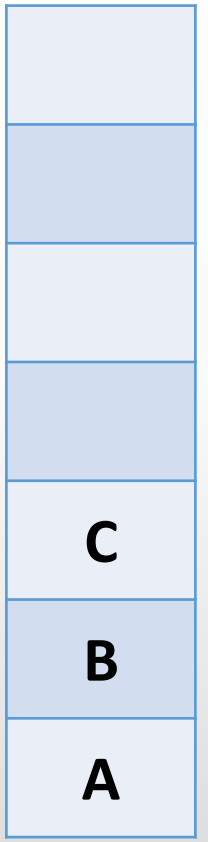
Derinlik Öncelikli Arama (DFS)



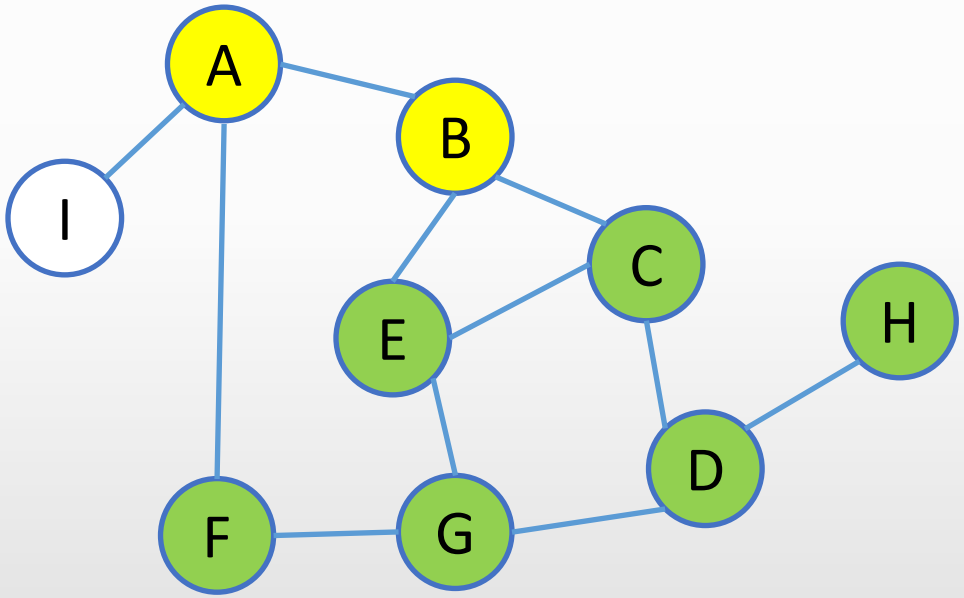


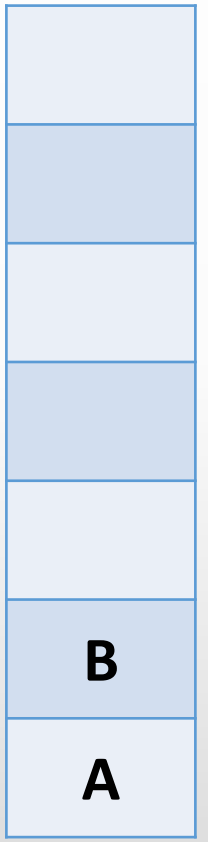
Derinlik Öncelikli Arama (DFS)



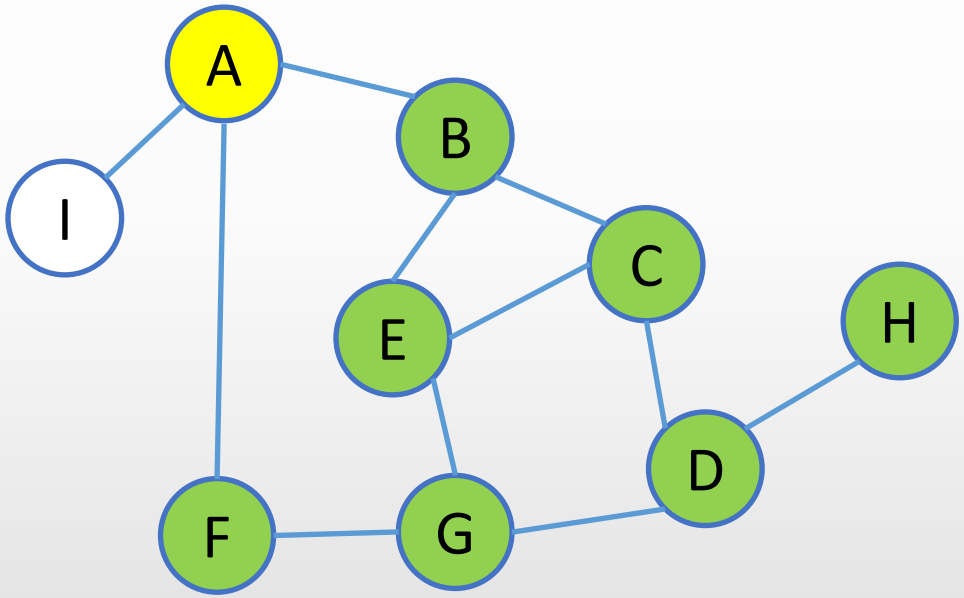


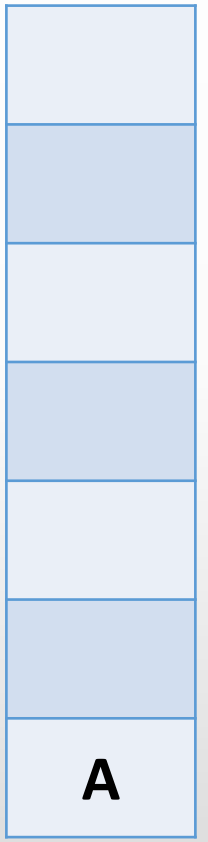
Derinlik Öncelikli Arama (DFS)



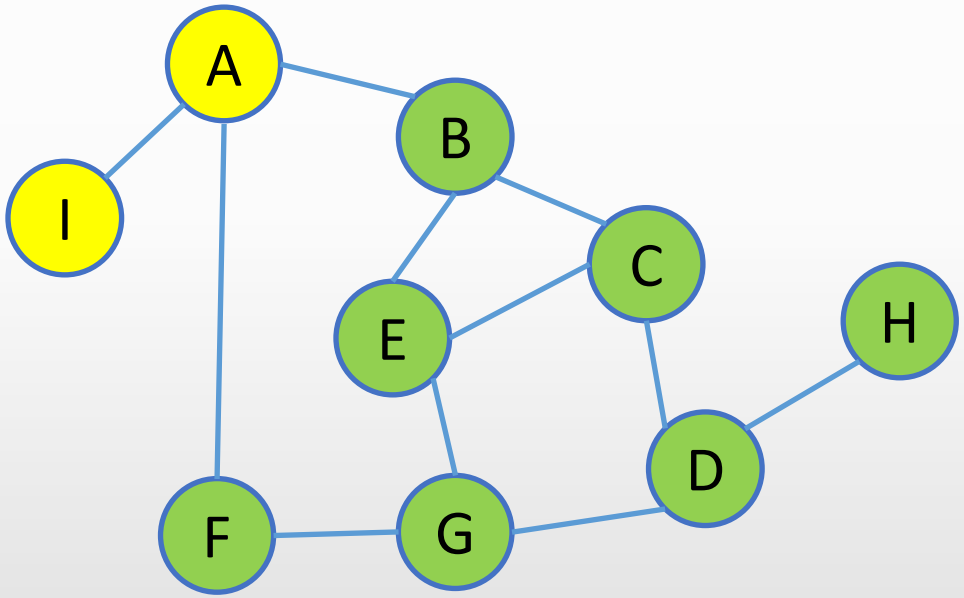


Derinlik Öncelikli Arama (DFS)

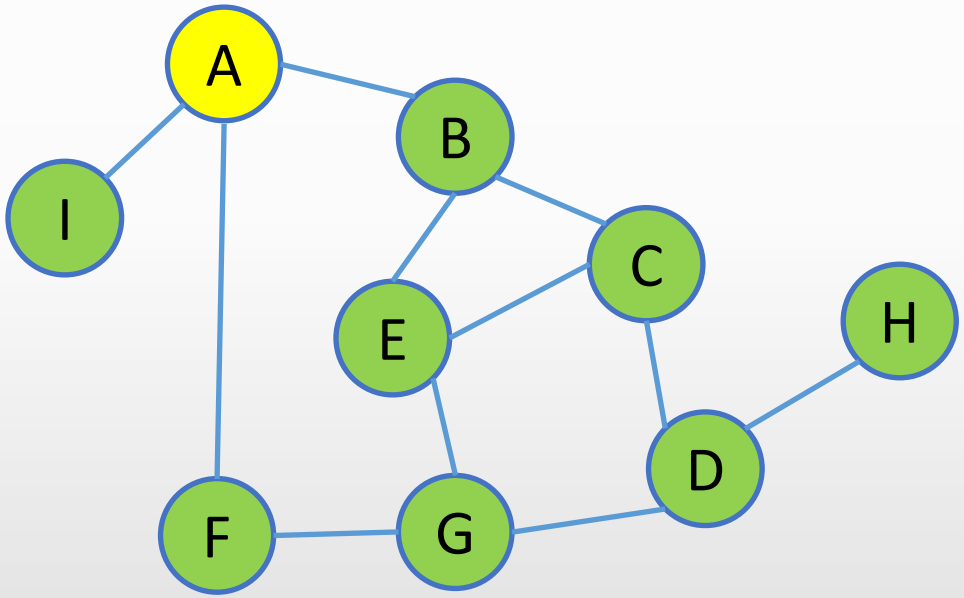


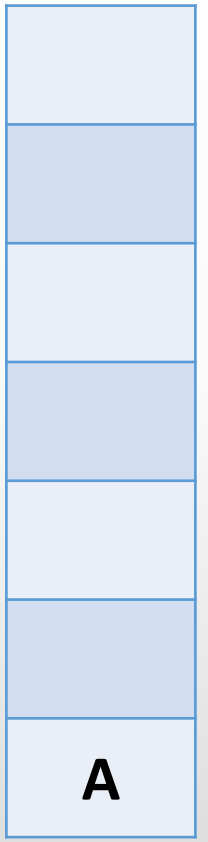


Derinlik Öncelikli Arama (DFS)

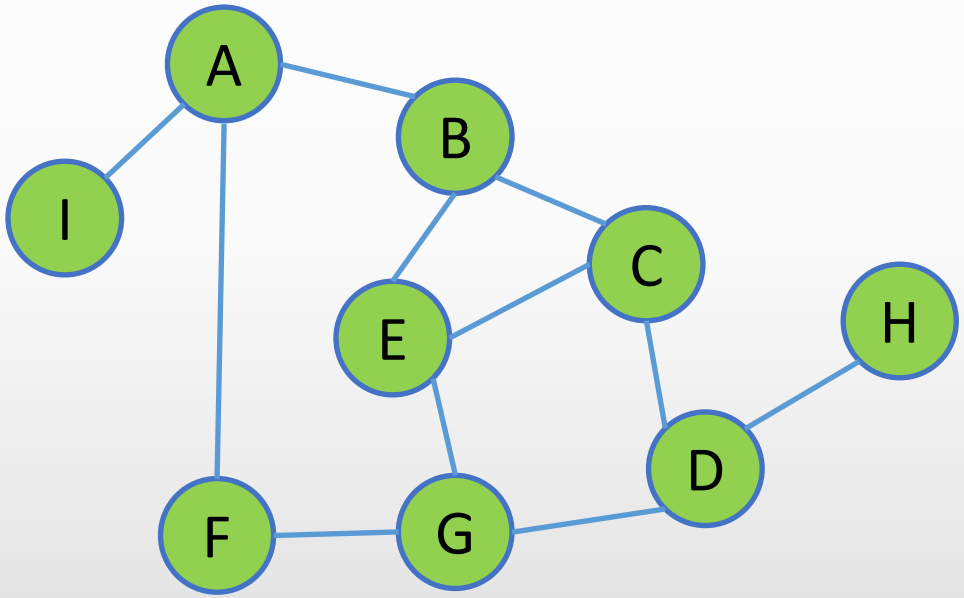


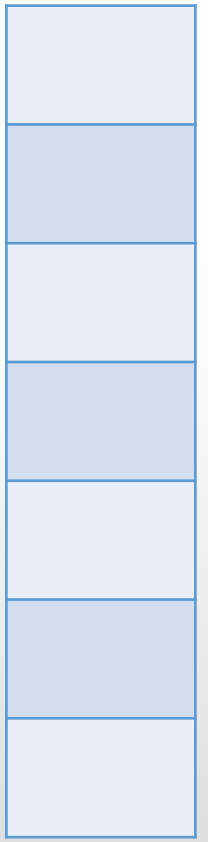
Derinlik Öncelikli Arama (DFS)





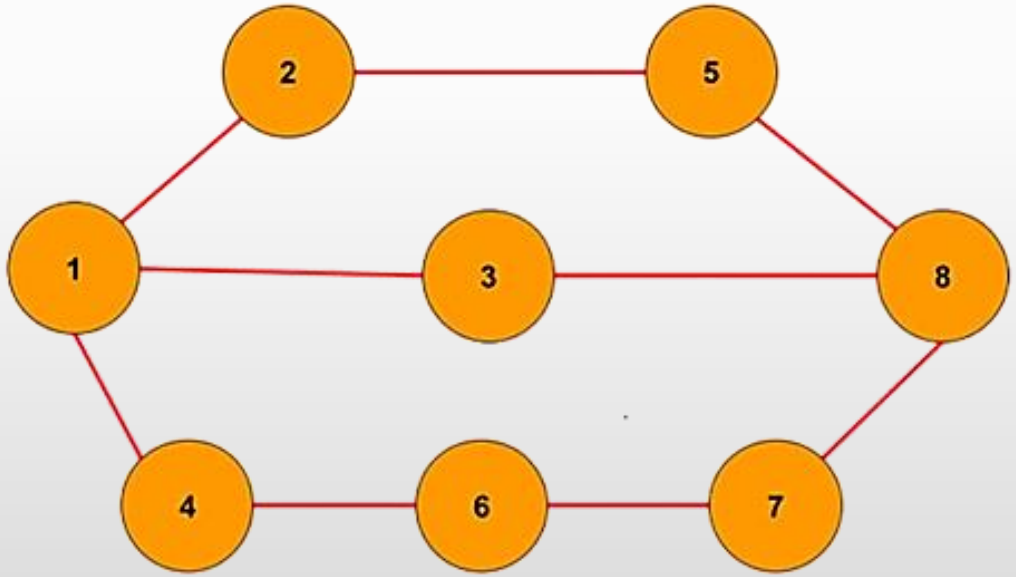
Derinlik Öncelikli Arama (DFS)





Ağırlıksız En Kısa Yol Problemi

- Kaynak düğümden diğer tüm düğümlere olan en kısa yolları bulur.
- Her kenarın aynı ağırlığa sahip olduğu varsayılır.



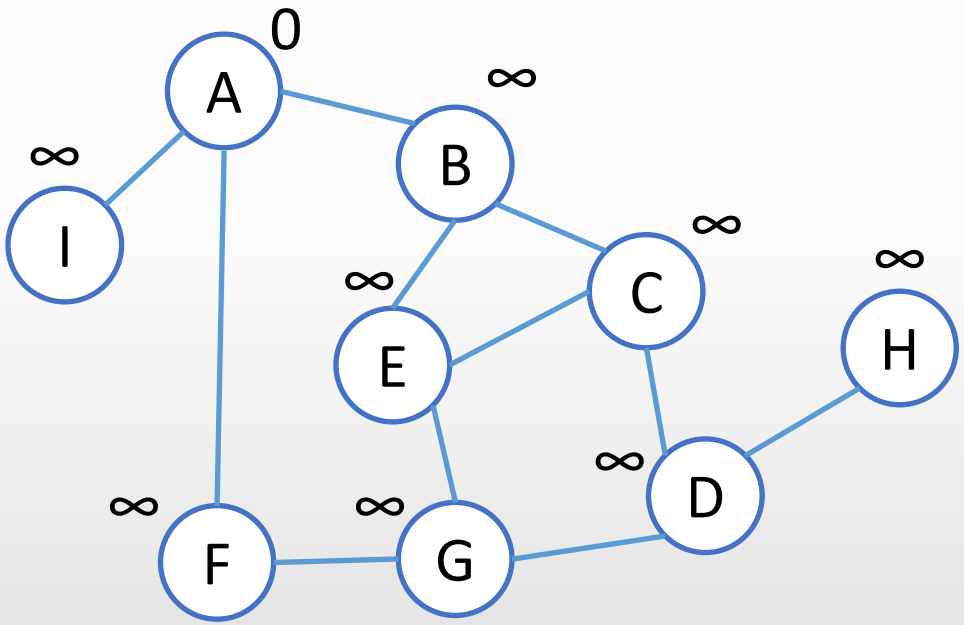
Ağırlıksız En Kısa Yol Algoritma

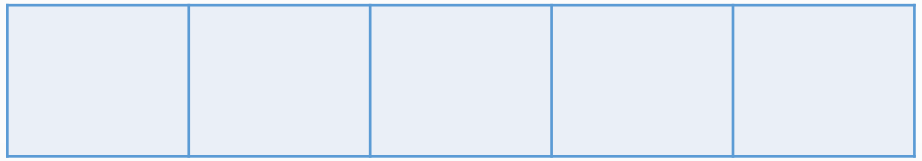
```
fonksiyon ağırlıksız_en_kısa_yol(çizge, başlangıç)
    mesafeler[çizgedeki tüm düğümler] = ∞;
    mesafeler[başlangıç] = 0;
    kuyruk.ekle(başlangıç);
    while !kuyruk.boşMu()
        mevcut_düğüm = kuyruk.çıkar()
        for (çizge[mevcut_düğüm] tüm komşuları)
            if (mesafeler[komşu] == ∞)
                mesafeler[komşu] = mesafeler[mevcut_düğüm] + 1
                kuyruk.ekle(komşu)
    return mesafeler
```

Ağırlıksız En Kısa Yol Bulma

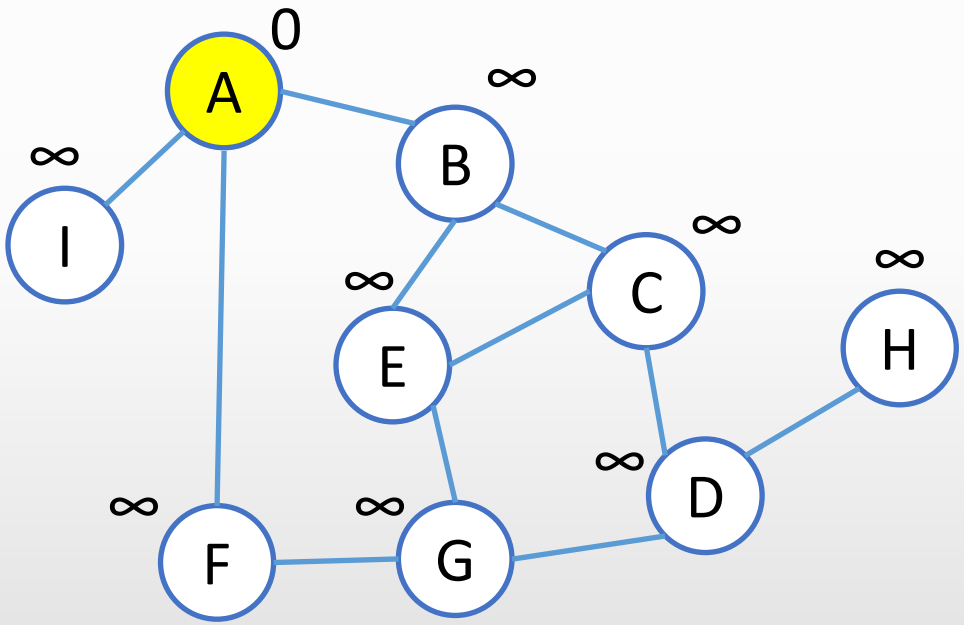
```
public Map<Integer, Integer> agirliksizEnKisaYol(int baslangic) {
  Map<Integer, Integer> mesafeler = new HashMap<>();
  Queue<Integer> kuyruk = new LinkedList<>();
  for (int dugum : cizge.keySet()) {
    mesafeler.put(dugum, Integer.MAX_VALUE);
  }
  mesafeler.put(baslangic, 0);
  kuyruk.offer(baslangic);
  while (!kuyruk.isEmpty()) {
    int mevcutDugum = kuyruk.poll();
    for (int komsu : cizge.get(mevcutDugum)) {
      if (mesafeler.get(komsu) == Integer.MAX_VALUE) {
        mesafeler.put(komsu, mesafeler.get(mevcutDugum) + 1);
        kuyruk.offer(komsu);
      }
    }
  }
  return mesafeler;
}
```

Ağırlıksız En Kısa Yol Bulma



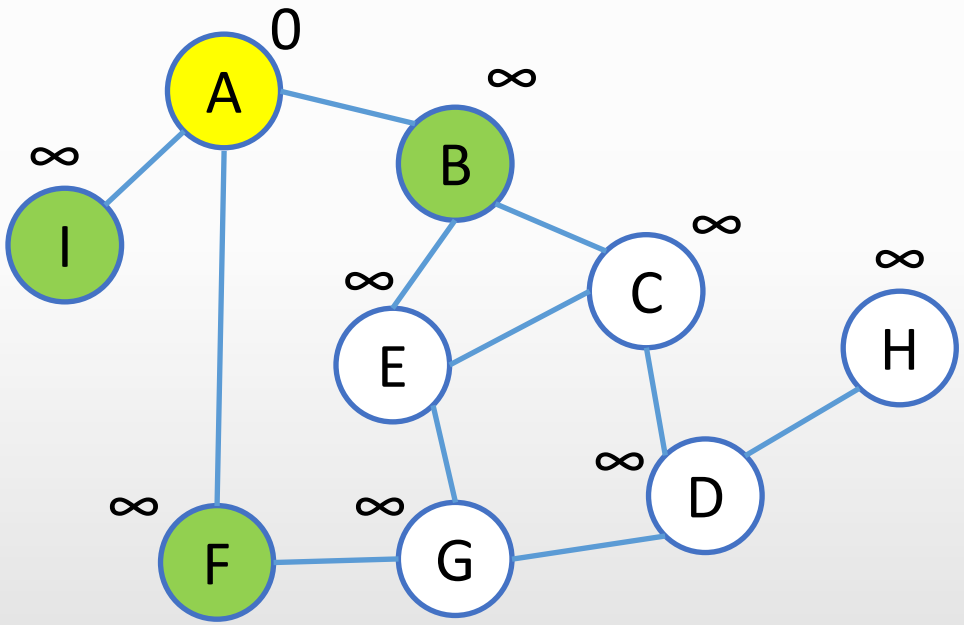


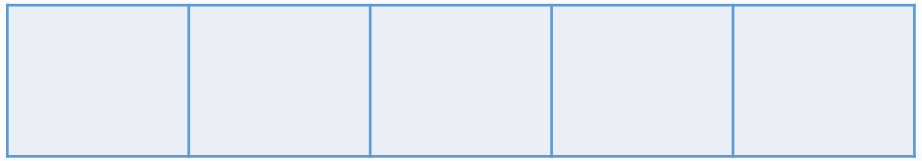
Ağırlıksız En Kısa Yol Bulma



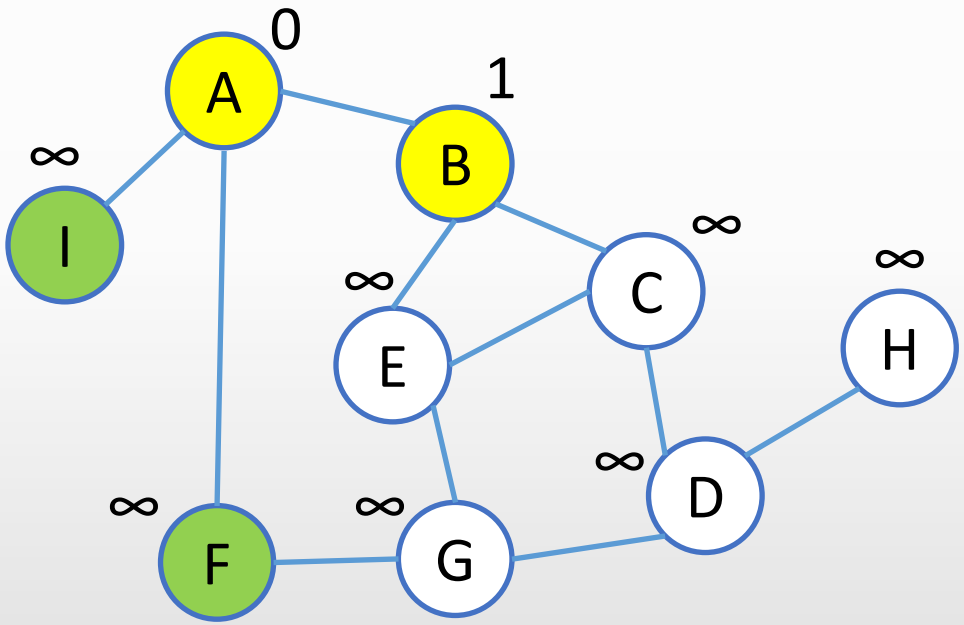
A				

Ağırlıksız En Kısa Yol Bulma



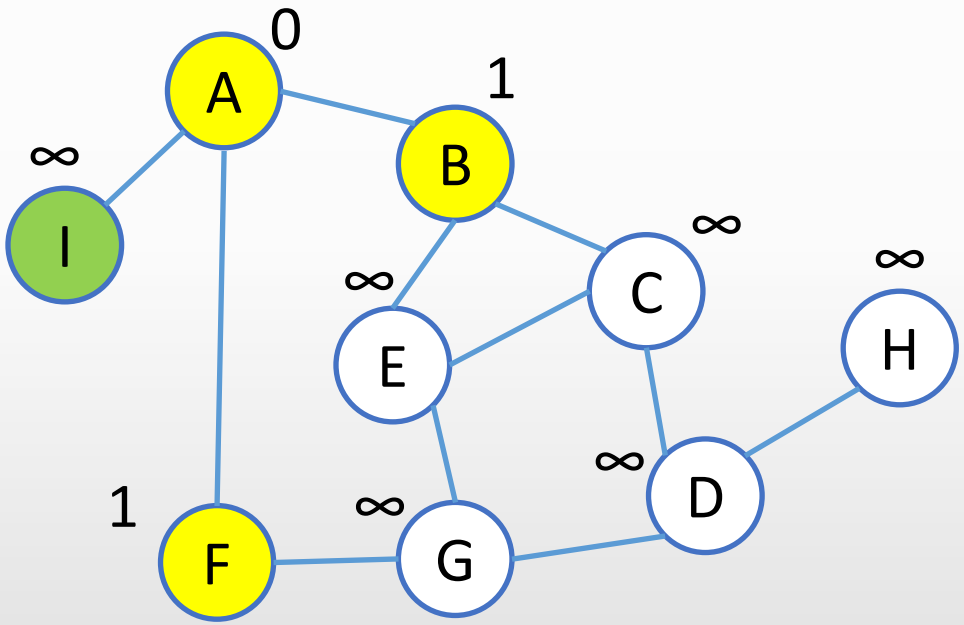


Ağırlıksız En Kısa Yol Bulma



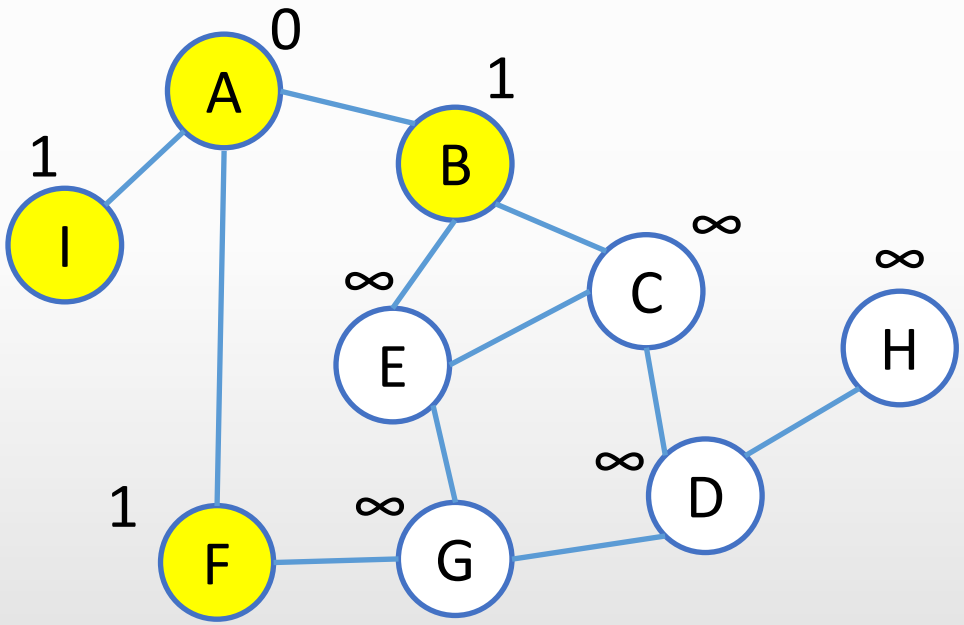
B				

Ağırlıksız En Kısa Yol Bulma



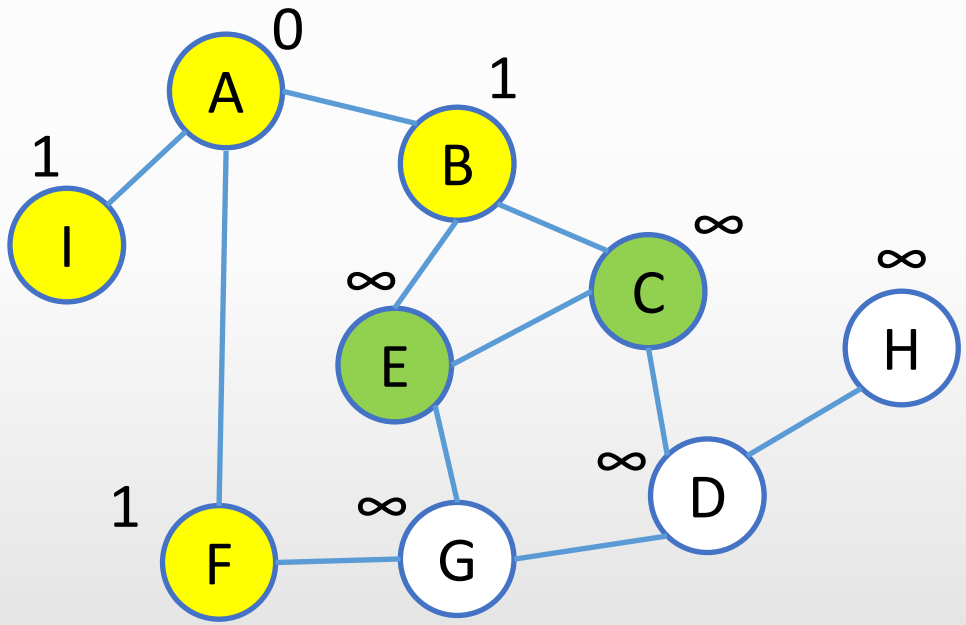
B	F			

Ağırlıksız En Kısa Yol Bulma



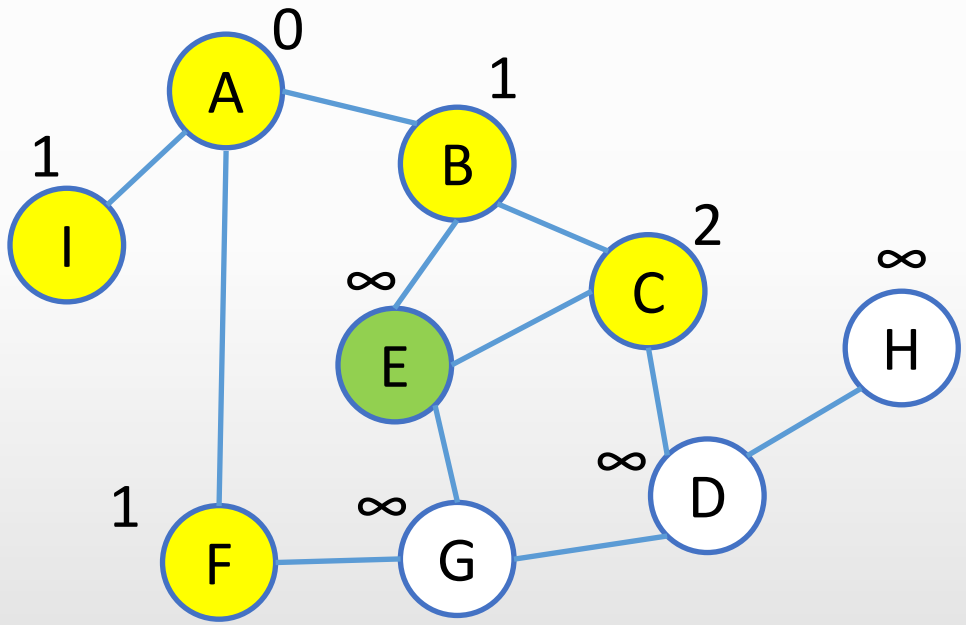
B	F	I		

Ağırlıksız En Kısa Yol Bulma



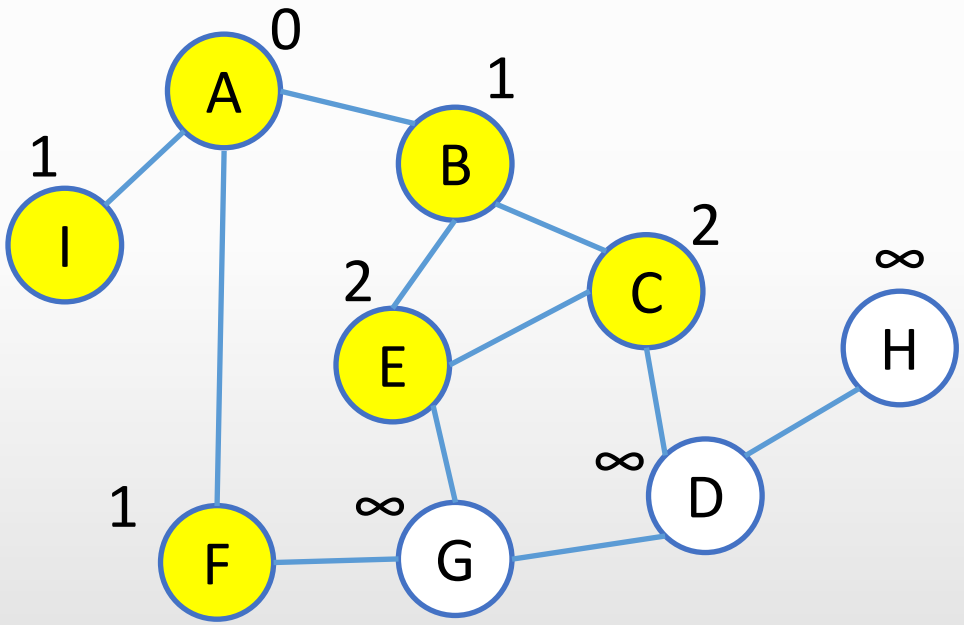
F	I			

Ağırlıksız En Kısa Yol Bulma



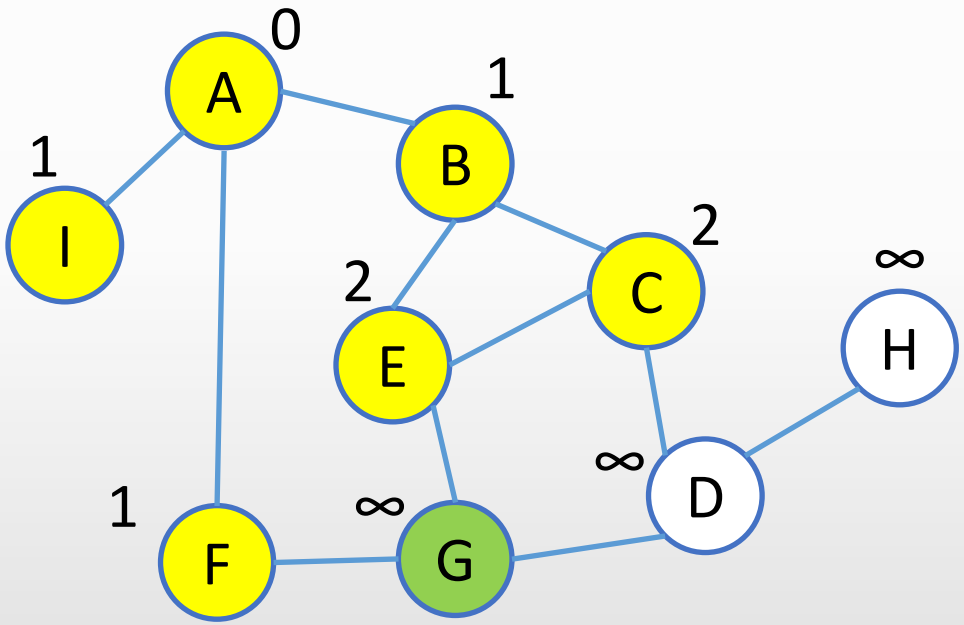
F	I	C		

Ağırlıksız En Kısa Yol Bulma



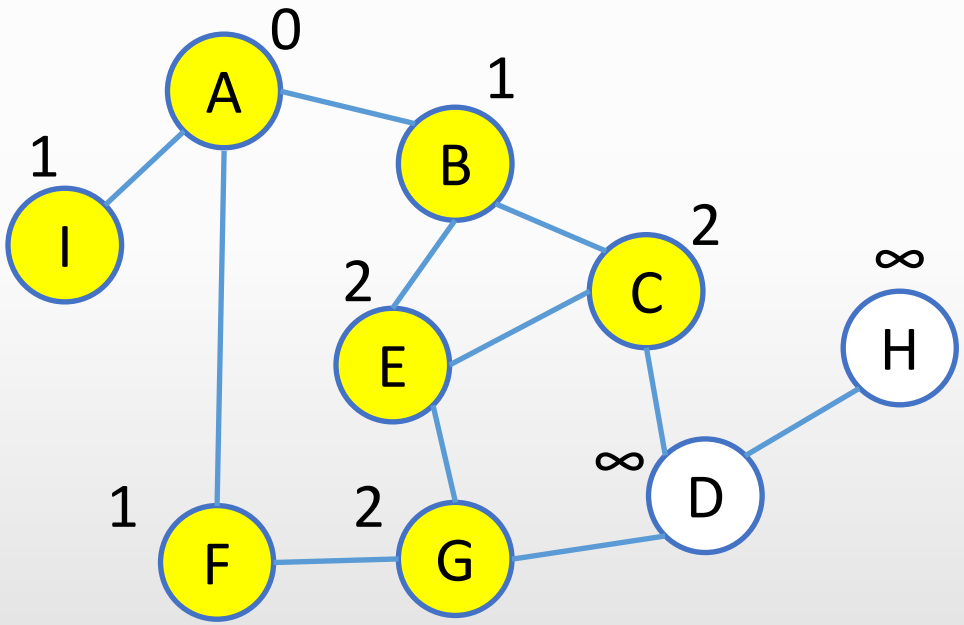
F	I	C	E	

Ağırlıksız En Kısa Yol Bulma



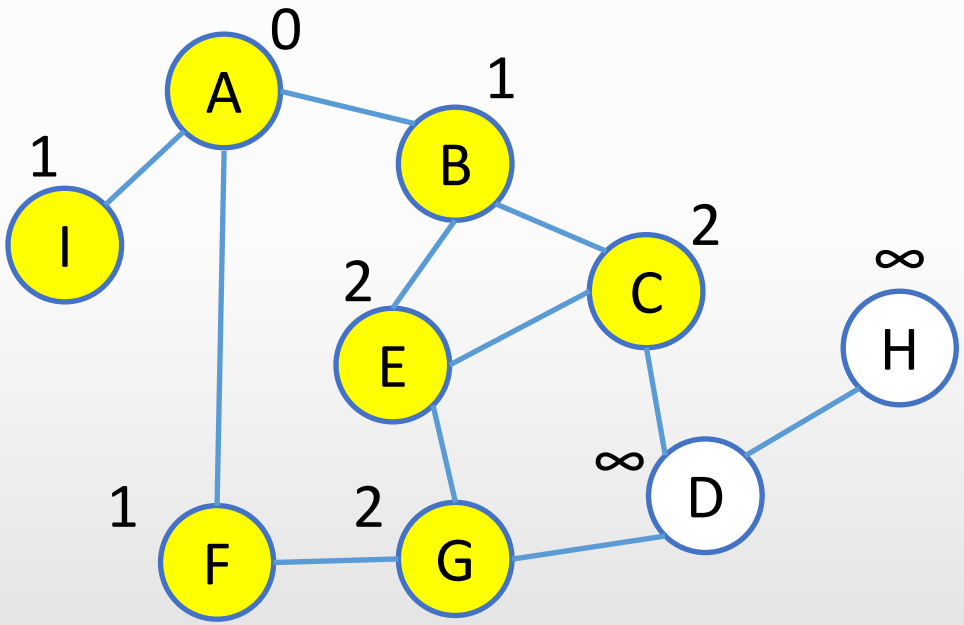
I	C	E		

Ağırlıksız En Kısa Yol Bulma



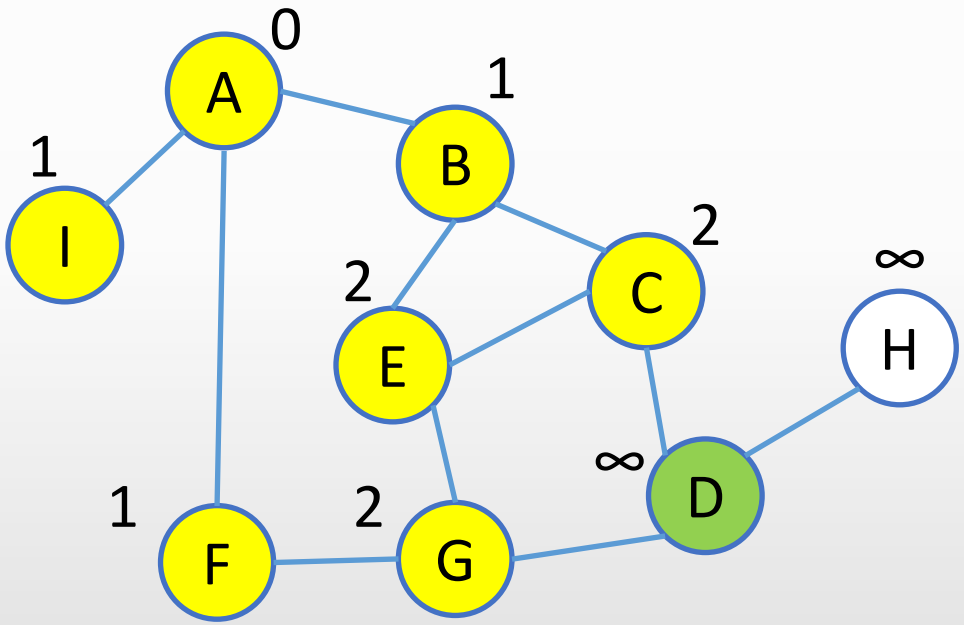
I	C	E	G	

Ağırlıksız En Kısa Yol Bulma



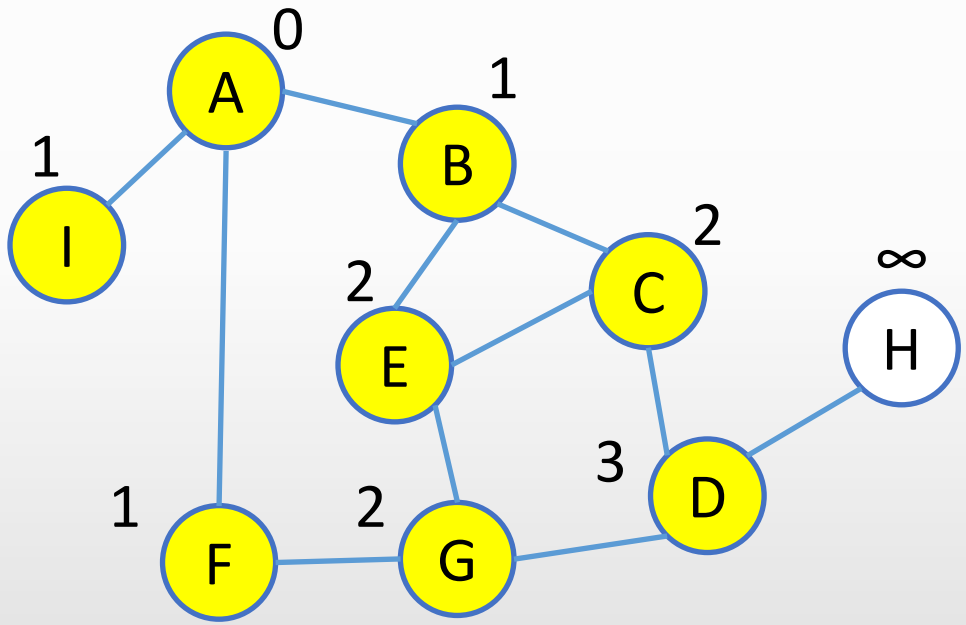
C	E	G		

Ağırlıksız En Kısa Yol Bulma



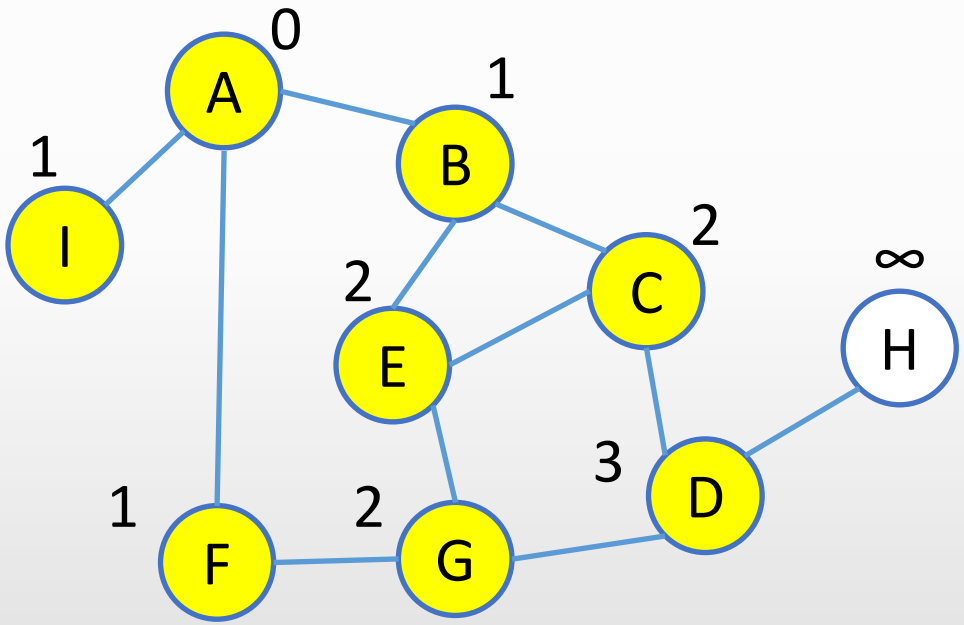
E	G			

Ağırlıksız En Kısa Yol Bulma



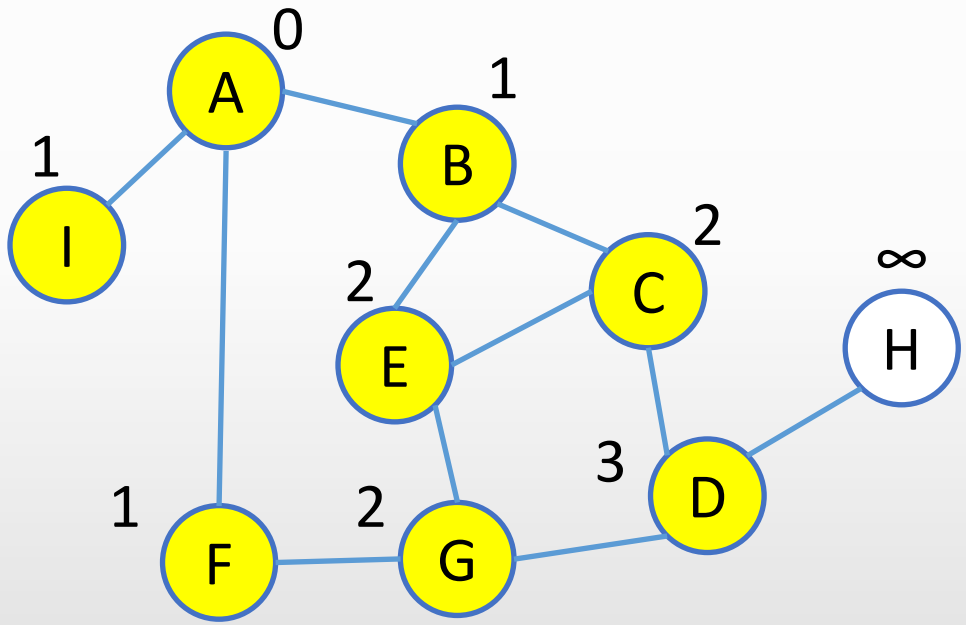
E	G	D		

Ağırlıksız En Kısa Yol Bulma



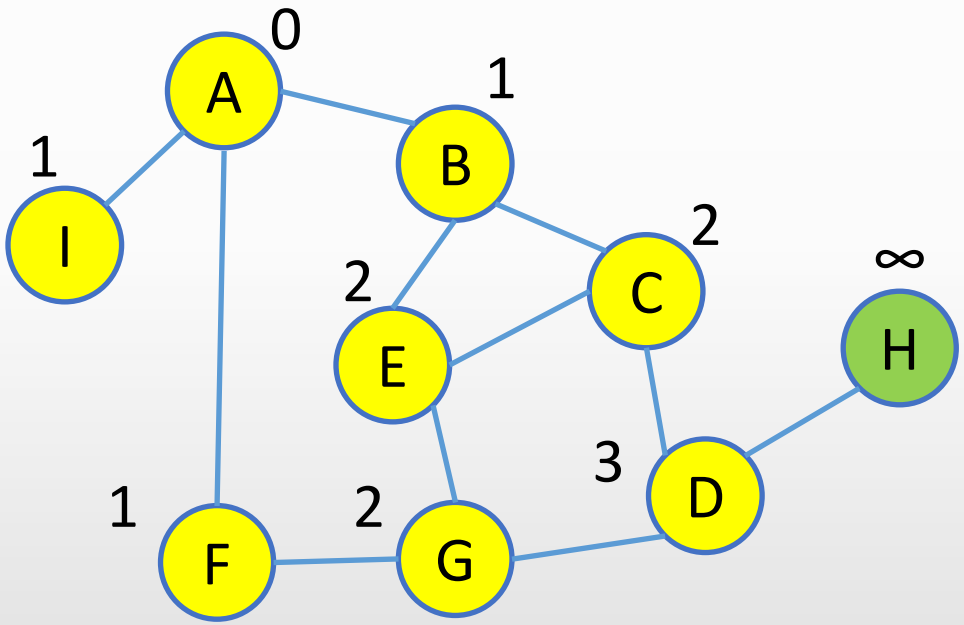
G	D			

Ağırlıksız En Kısa Yol Bulma

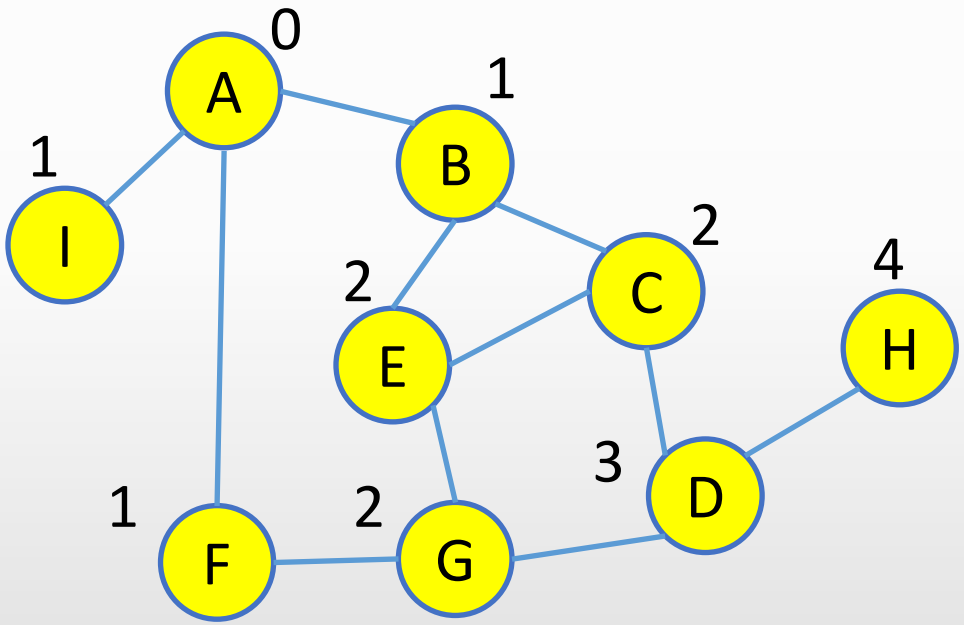


D				

Ağırlıksız En Kısa Yol Bulma

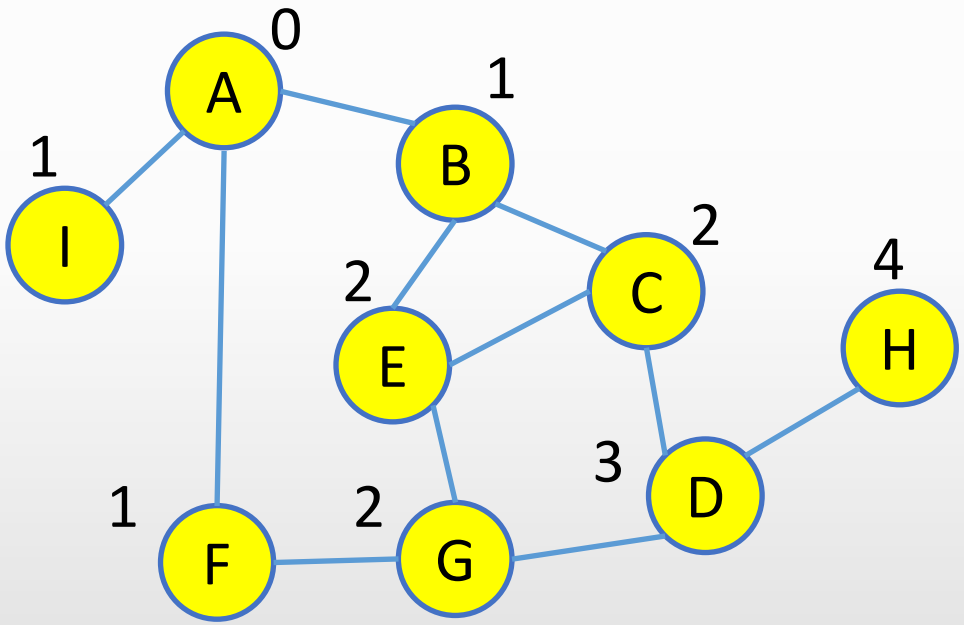


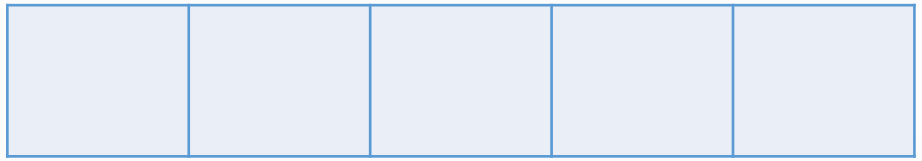
Ağırlıksız En Kısa Yol Bulma



H				

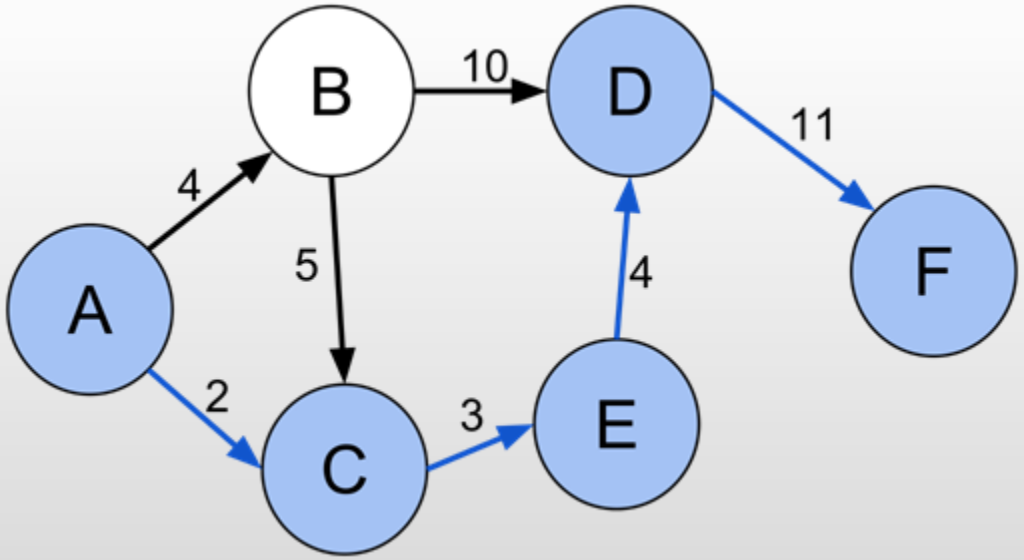
Ağırlıksız En Kısa Yol Bulma





Ağırlıklı En Kısa Yol Bulma

- Kaynak düğümden diğer tüm düğümlere olan en kısa yolları bulur.
- Her kenar farklı bir ağırlığa sahiptir.
- Ağırlıklar uzaklık, maliyet gibi bir ölçüm birimi olabilir.



Ağırlıklı En Kısa Yol Bulma Algoritması

```
fonksiyon Dijkstra (çizge, başlangıç)
    for (her düğüm)
        mesafe[v] = SONSUZ;  önceki[v] = TANIMSIZ;  kuyruk.ekle(v); //öncelikli
    mesafe[başlangıç] = 0;
    while (!kuyruk.bosMu())
        u = kuyruk.cikar(); // en küçük mesafe[u]
        for (u'nun kuyrukta olan her komşusu)
            alternatif = mesafe[u] + çizge.kenar(u, v);
            if (alternatif < mesafe[v])
                mesafe[v] ← alternatif;
                önceki[v] ← u;
```

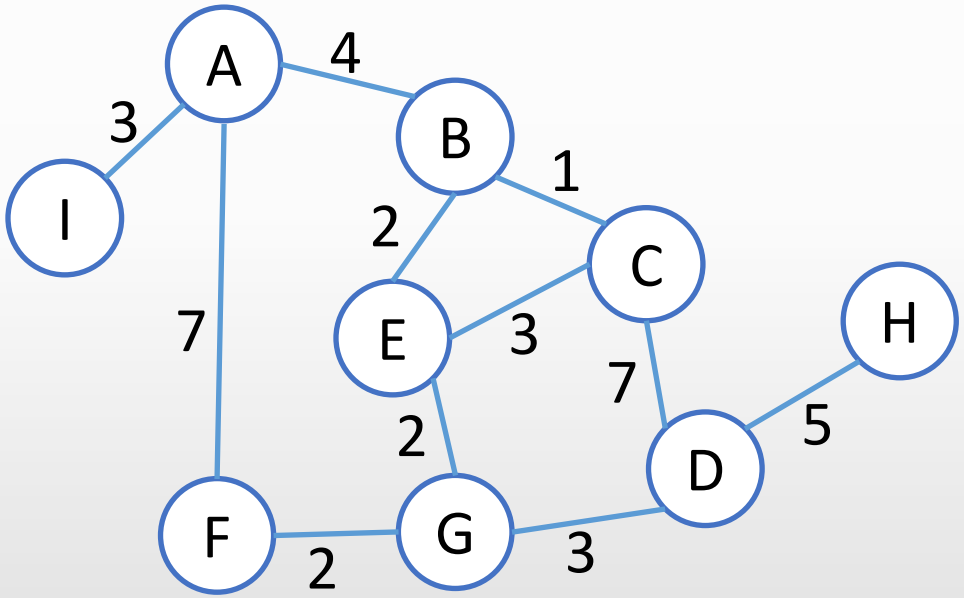
Dijkstra Algoritması

```
public void dijkstra(Cizge cizge, int kaynak) {
  for (int dugum : cizge.dugumler)
    mesafeler.put(dugum, Integer.MAX_VALUE);
  mesafeler.put(kaynak, 0);
  kuyruk.offer(kaynak);
  while (!kuyruk.isEmpty()) {
    int simdiki = kuyruk.poll();
    ziyaretEdilen.add(simdiki);
    for (Map.Entry komsu : cizge.dugumler.get(simdiki)) {
      if (!ziyaretEdilen.contains(komsu.getKey())) {
        int alternatif = mesafeler.get(simdiki) + komsu.getValue();
        if (alternatif < mesafeler.get(komsu.getKey())) {
          mesafeler.put(komsu.getKey(), alternatif);
          kuyruk.offer(komsu.getKey());
        }
      }
    }
  }
}
```

Dijkstra Algoritması Analizi

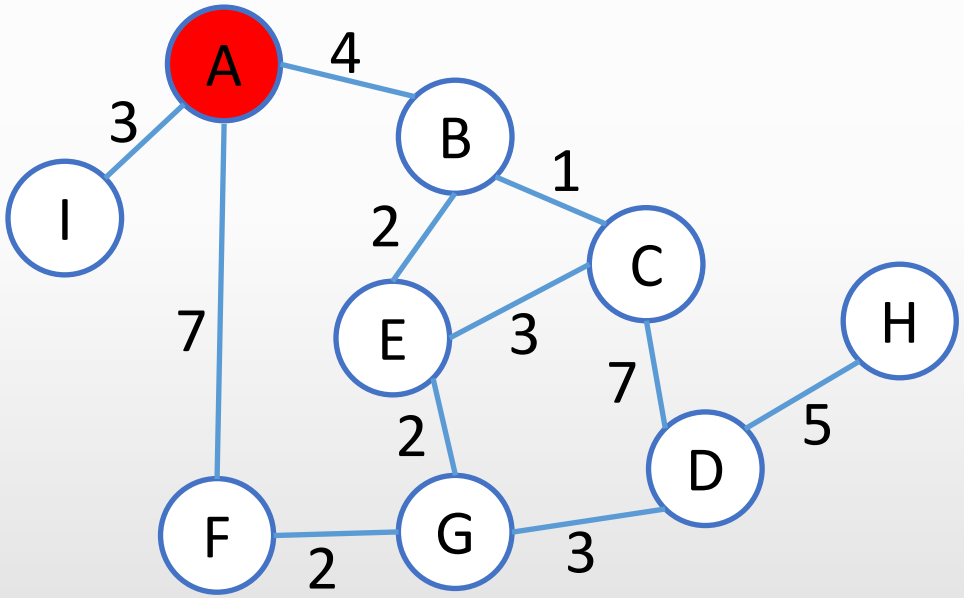
- Öncelik Kuyruğu Kullanılırsa:
  - **Karmaşıklık**: O((V + E) log V)
  - Kuyruğa ekleme/çıkarma işlemleri logaritmik zaman karmaşıklığına sahiptir. İşlemler, düğüm (V) ve kenar sayısı (E) ile doğru orantılıdır.
- Öncelik Kuyruğu Kullanılmazsa:
  - **Karmaşıklık**: $O(V^2 + E)$
  - En küçük mesafeli düğümü bulmak için her iterasyonda tüm düğümler üzerinde döngü yapılması V^2 zaman karmaşıklığına sahiptir. Her düğümün komşularının kontrol edilmesi, toplam kenar sayısı (E) kadar ek işlem gerektirir.

Dijkstra Algoritması



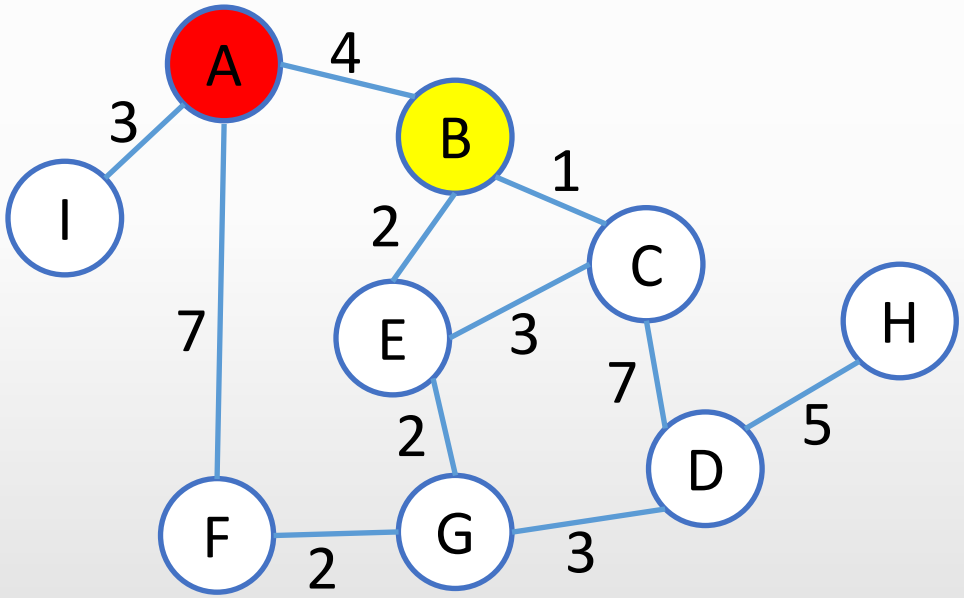
A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	∞	∞	∞	∞

Dijkstra Algoritması



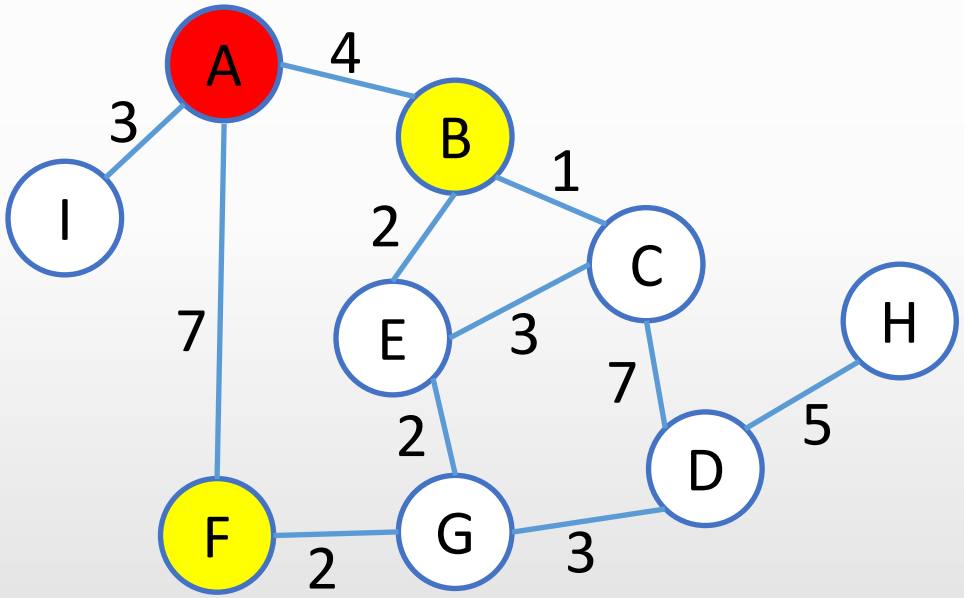
A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	∞	∞	∞	∞

Dijkstra Algoritması



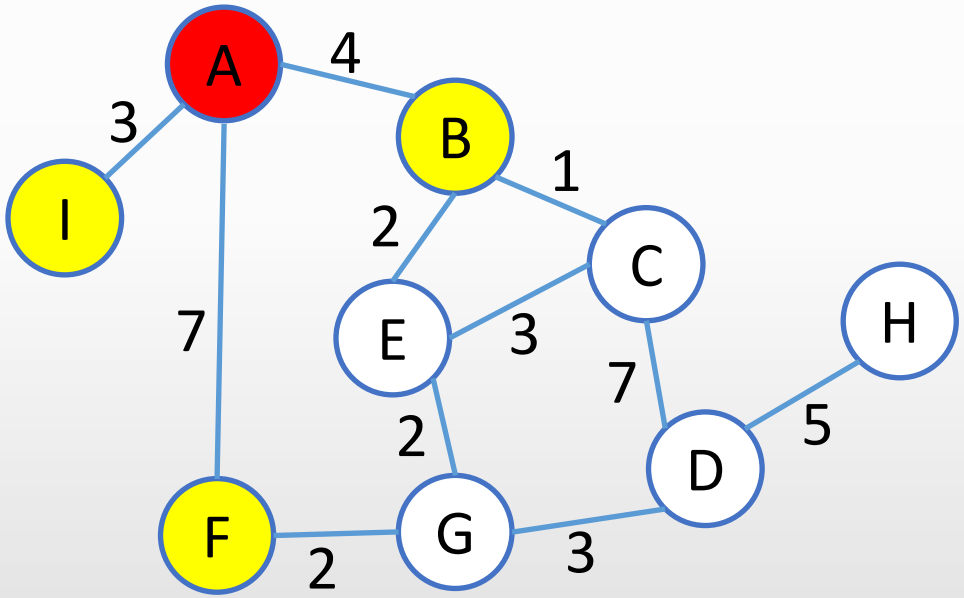
A	B	C	D	E	F	G	H	I
0	4	∞	∞	∞	∞	∞	∞	∞

Dijkstra Algoritması



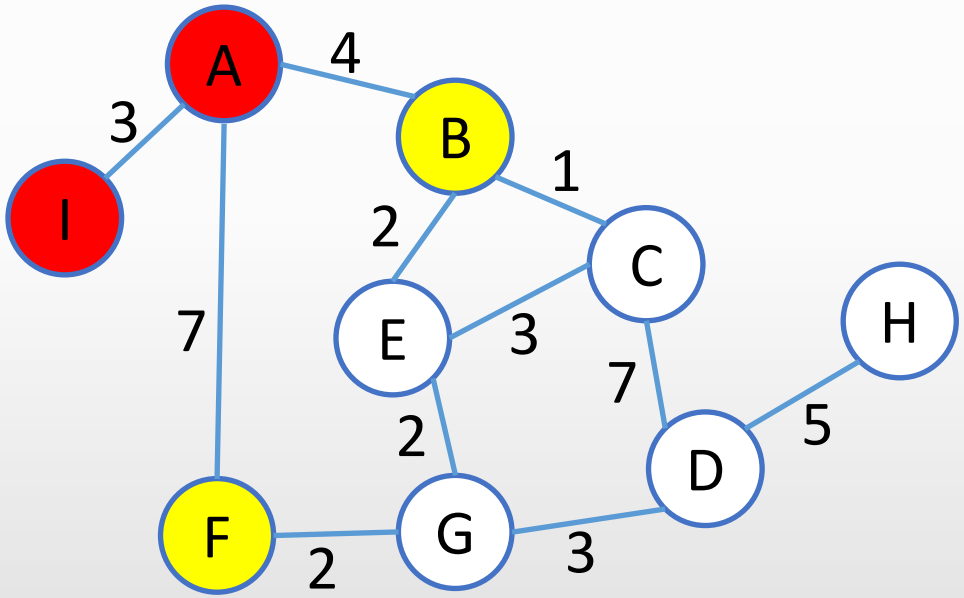
A	B	C	D	E	F	G	H	I
0	4	∞	∞	∞	7	∞	∞	∞

Dijkstra Algoritması



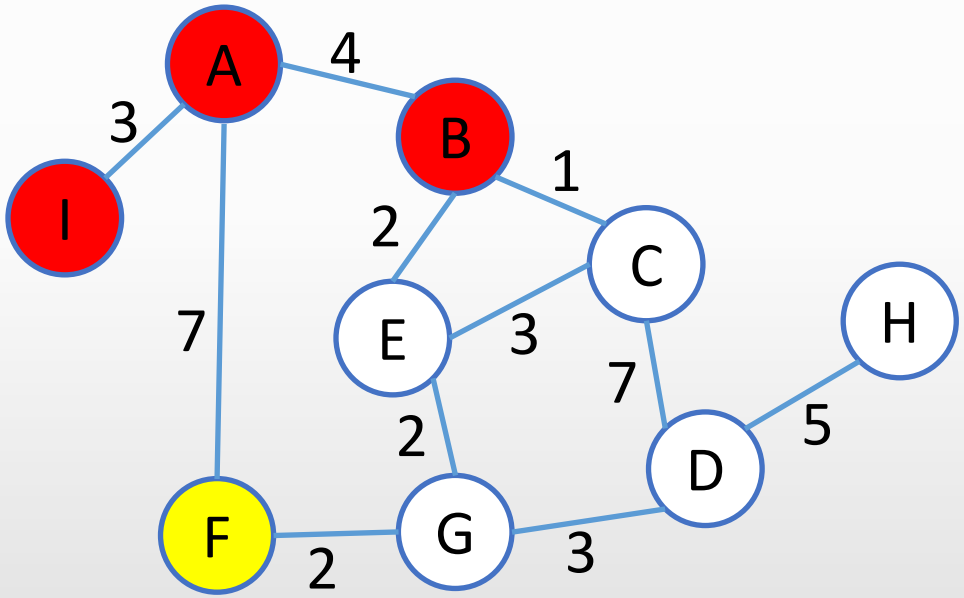
A	B	C	D	E	F	G	H	I
0	4	∞	∞	∞	7	∞	∞	3

Dijkstra Algoritması



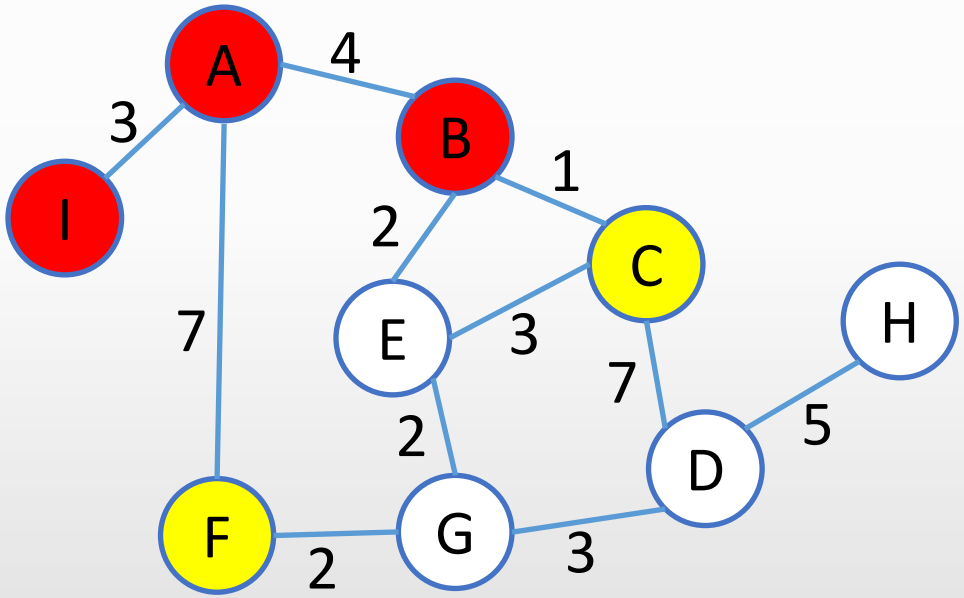
A	B	C	D	E	F	G	H	I
0	4	∞	∞	∞	7	∞	∞	3

Dijkstra Algoritması



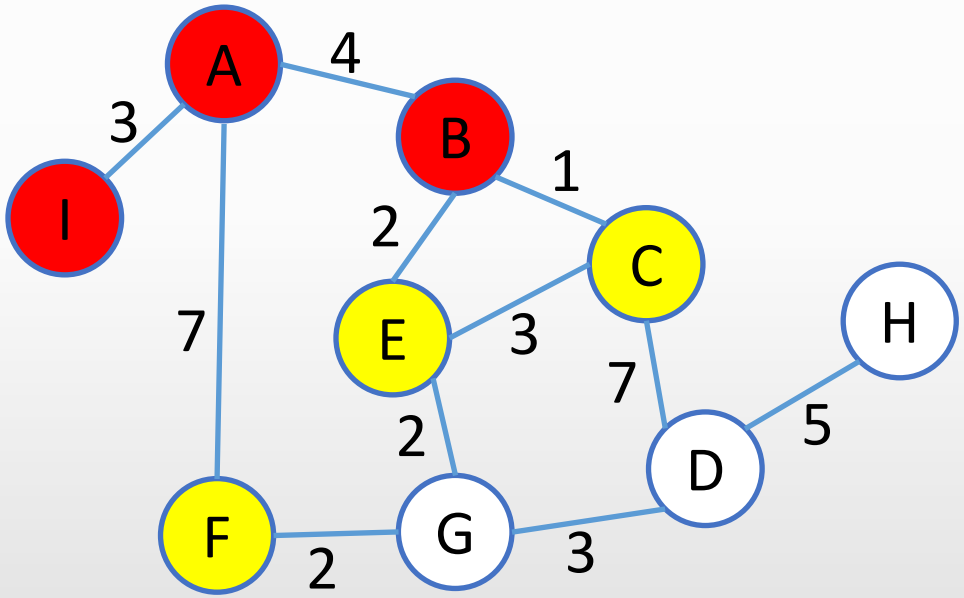
A	B	C	D	E	F	G	H	I
0	4	∞	∞	∞	7	∞	∞	3

Dijkstra Algoritması



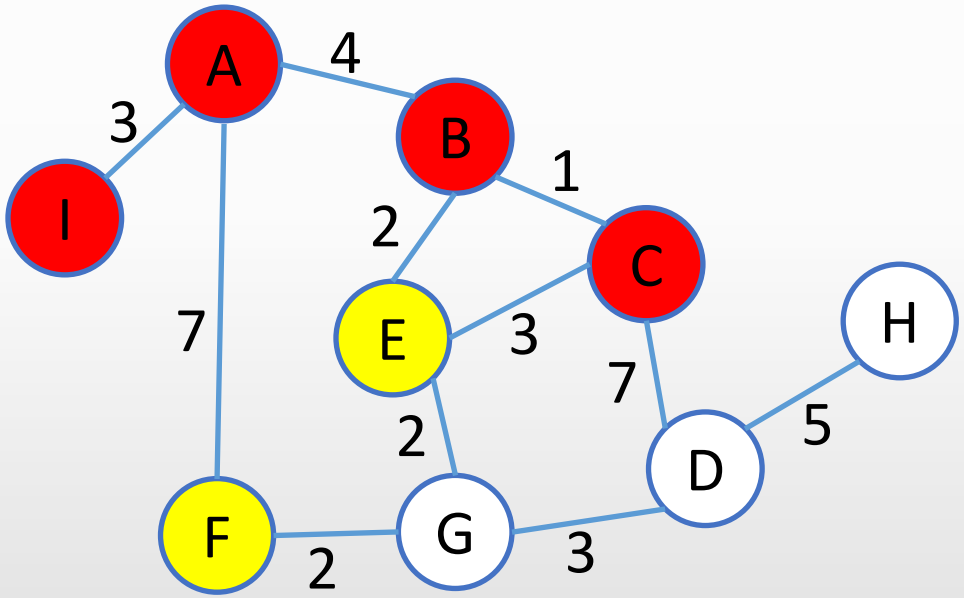
A	B	C	D	E	F	G	H	I
0	4	5	∞	∞	7	∞	∞	3

Dijkstra Algoritması



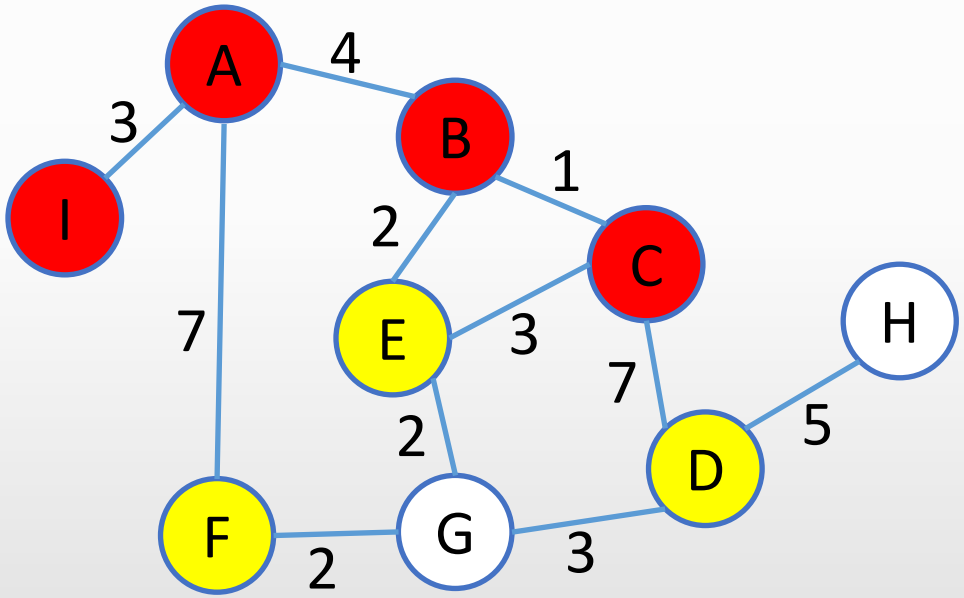
A	B	C	D	E	F	G	H	I
0	4	5	∞	6	7	∞	∞	3

Dijkstra Algoritması



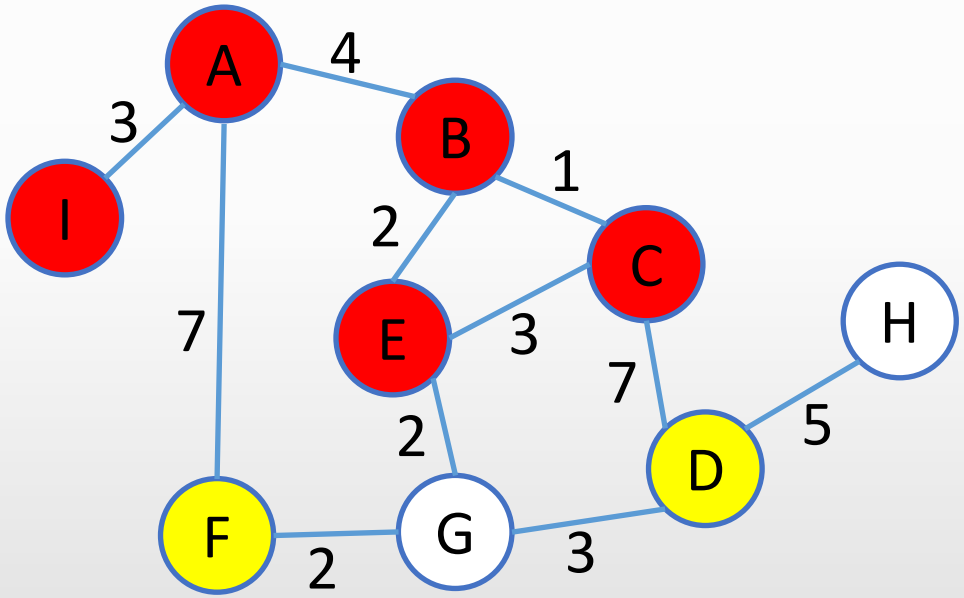
A	B	C	D	E	F	G	H	I
0	4	5	∞	6	7	∞	∞	3

Dijkstra Algoritması



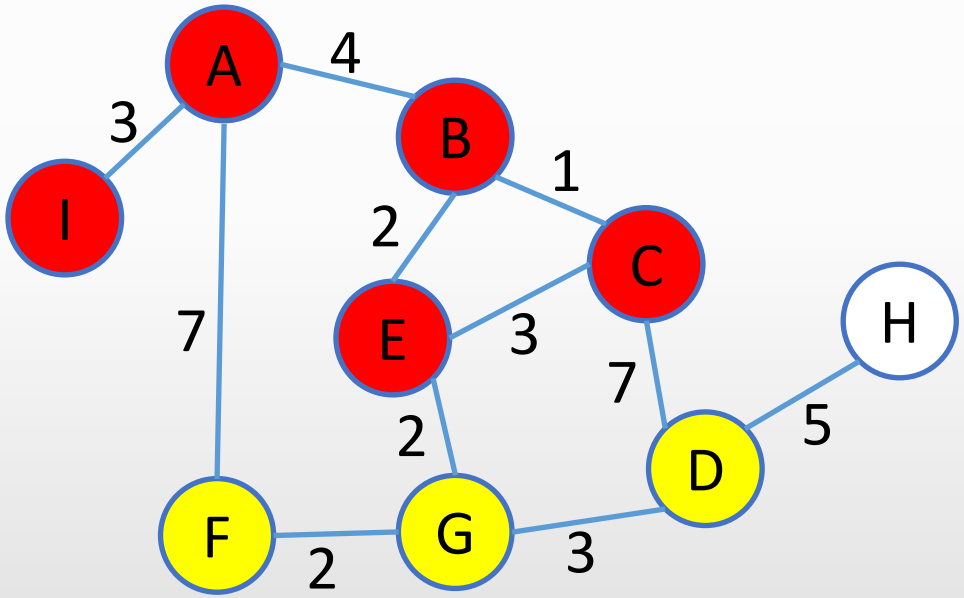
A	B	C	D	E	F	G	H	I
0	4	5	12	6	7	∞	∞	3

Dijkstra Algoritması



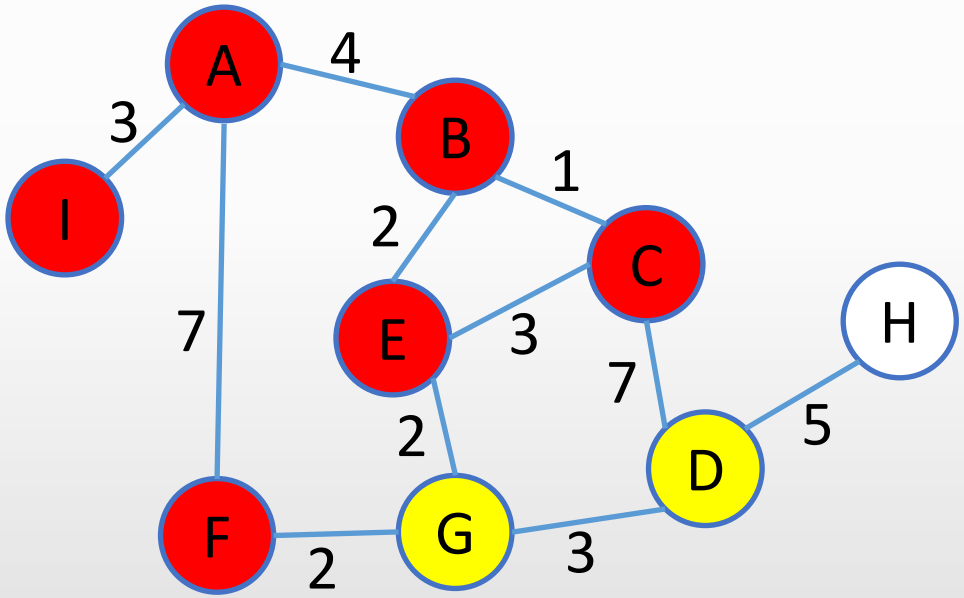
A	B	C	D	E	F	G	H	I
0	4	5	12	6	7	∞	∞	3

Dijkstra Algoritması



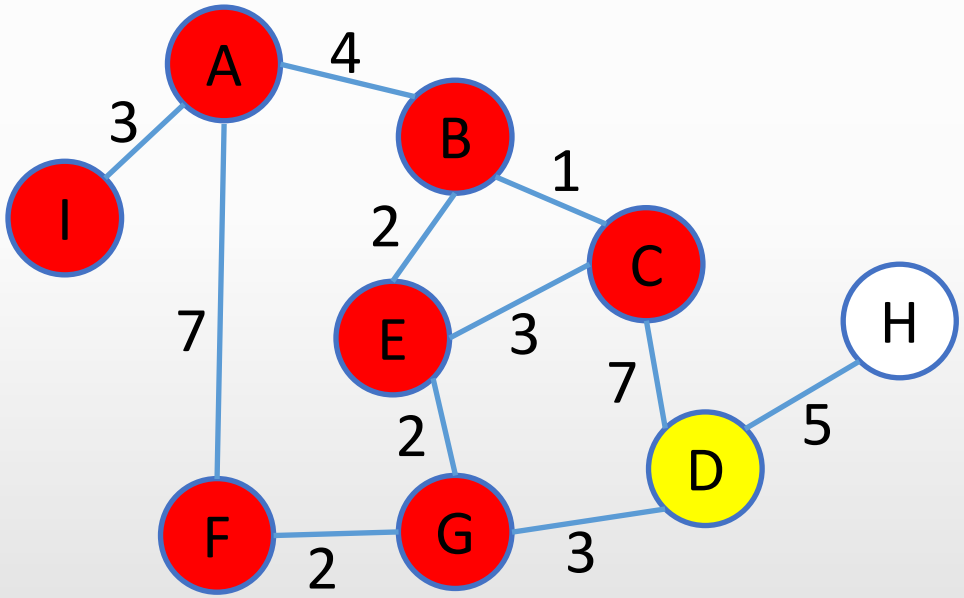
A	B	C	D	E	F	G	H	I
0	4	5	12	6	7	8	∞	3

Dijkstra Algoritması



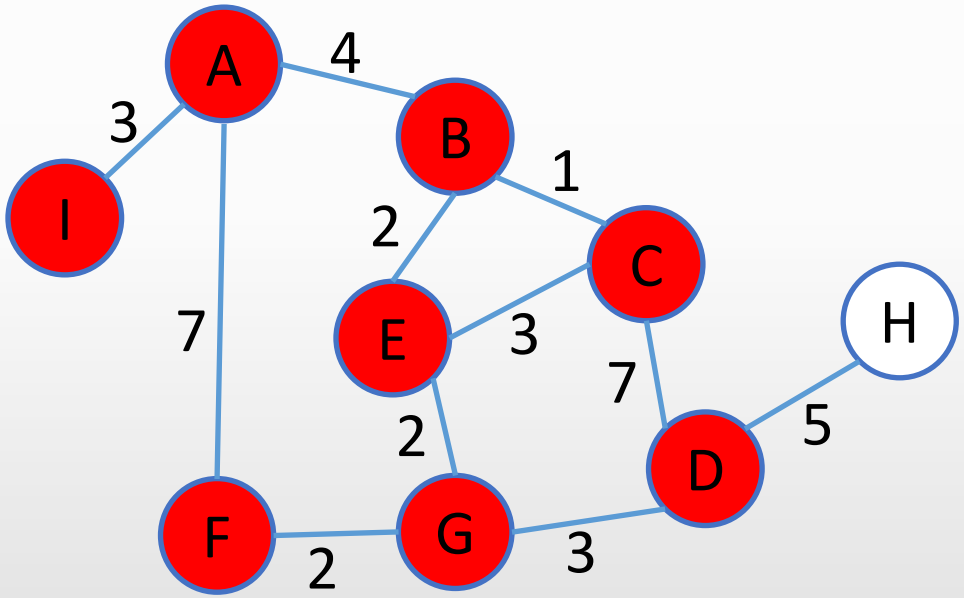
A	B	C	D	E	F	G	H	I
0	4	5	12	6	7	8	∞	3

Dijkstra Algoritması



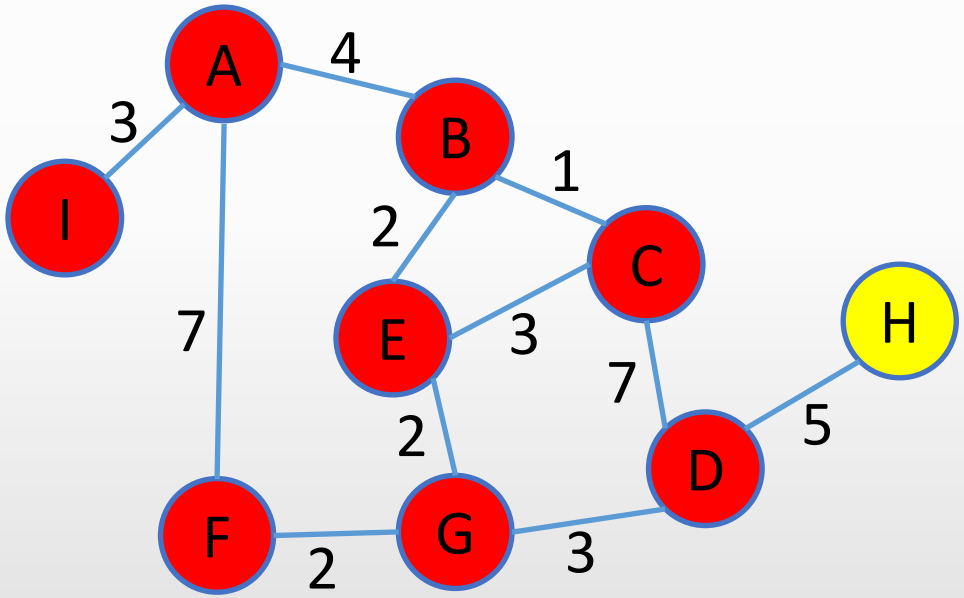
A	B	C	D	E	F	G	H	I
0	4	5	11	6	7	8	∞	3

Dijkstra Algoritması



A	B	C	D	E	F	G	H	I
0	4	5	11	6	7	8	∞	3

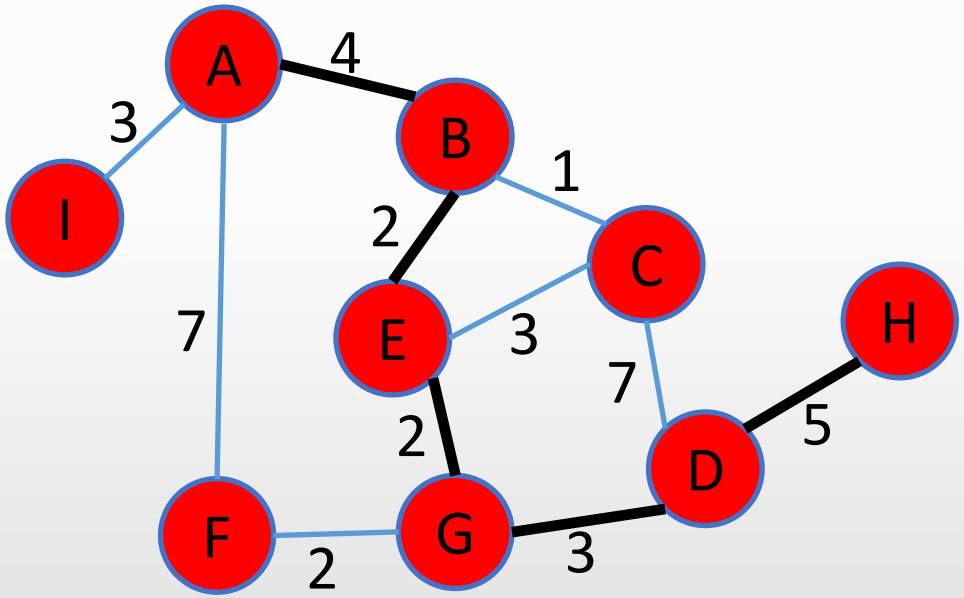
Dijkstra Algoritması



A	B	C	D	E	F	G	H	I
0	4	5	11	6	7	8	16	3

Dijkstra Algoritması

- Yol: A -> B-> E -> G -> D -> H



A	B	C	D	E	F	G	H	I
0	4	5	11	6	7	8	16	3

SON