

Bölüm 4.2: En Kısa Yol Algoritmaları

Algoritmalar

Gevşetme

Dijkstra

Bellman-Ford

Negatif döngü

Floyd-Warshall

A* arama

BFS ile ağırlıksız yol

Karşılaştırma

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Kavramlar

Tek kaynaklı, tüm çiftler ve tek çift en kısa yol problemlerini ayırt etmek; **gevşetme** (relaxation) işlemini ve önceki (π) dizisini açıklamak.

Dijkstra ve Bellman-Ford

İki algoritmayı bir örnek çizge üzerinde adım adım çalıştırmak, uzaklık ve π tablolarını doldurmak, Java kodunu izlemek.

Negatif kenar ve döngü

Dijkstra'nın negatif kenarda neden yarıldığını göstermek; Bellman-Ford ve Floyd-Warshall ile negatif döngüyü tespit etmek.

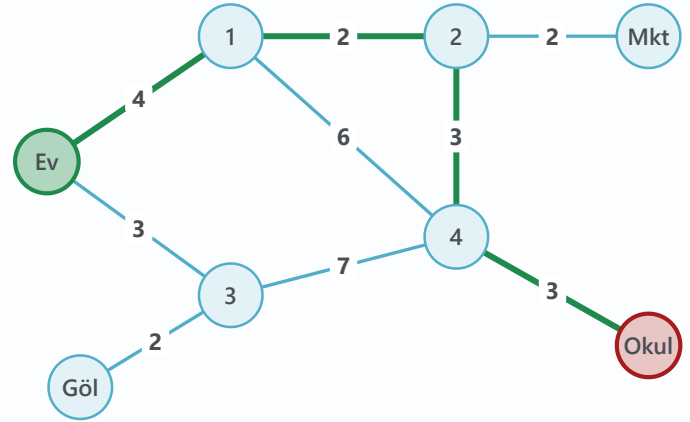
Floyd-Warshall ve A*

Tüm çiftler için D ve Π matrislerini $k = 1 \dots n$ güncellemek; A*'ın $f = g + h$ ile hedefe yönelen aramasını uygulamak ve algoritmaları karşılaştırmak.

GİRİŞ

Yol Ağı Bir Çizgedir

- Kavşaklar **düğüm** (vertex), yollar **kenar** (edge), yol uzunlukları ya da süreleri **ağırlık** (weight) olur.
- Bir yolun **uzunluğu** = üzerindeki kenar ağırlıklarının toplamı.
- **En kısa yol** (shortest path): iki düğüm arasındaki toplam ağırlığı en küçük yol; uzunluğu $\delta(s, v)$ ile gösterilir.
- Örnek: Ev $\rightarrow$ Okul için Ev-1-2-4-Okul = 4 + 2 + 3 + 3 = **12**; daha az kenarlı Ev-1-4-Okul ise 13 eder.



Kenar sayısı en az olan yol, ağırlıklı çizgede en kısa yol **olmayabilir**. Navigasyon, ağ yönlendirme (OSPF, RIP), oyunlarda yol bulma ve lojistik bu problemi çözer.

En Kısa Yol Problem Türleri

Tek kaynaklı

(single-source) s'den **tüm** düğümlere. Dijkstra, Bellman-Ford, BFS.

Tüm çiftler

(all-pairs) her (u, v) çifti; sonuç $n \times n$ matris. Floyd-Warshall.

Tek çift

(single-pair) yalnızca $s \rightarrow t$; hedefe yönelen arama. A^* .

Algoritma	Ne bulur?	Kısıt / not
Dijkstra	kaynaktan tüm düğümlere	kenar ağırlıkları negatif olmamalı
Bellman-Ford	kaynaktan tüm düğümlere	negatif kenarda çalışır, negatif döngüyü bulur; Dijkstra'dan yavaş
Floyd-Warshall	tüm çiftler arası	negatif kenarda çalışır; negatif döngüyü köşegenden tespit eder
A^*	başlangıçtan hedefe	sezgisel (heuristic) tahminle aramayı hızlandırır
BFS	kaynaktan tüm düğümlere	yalnızca ağırlıksız (ya da eşit ağırlıklı) çizgeler

GİRİŞ

Temel İşlem: Gevşetme (Relaxation)

Her düğüm için iki bilgi tutulur:

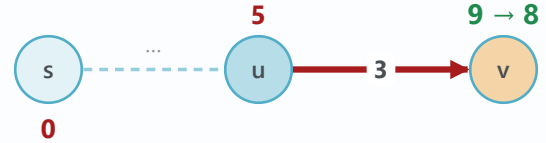
- $d[v]$: şimdiye kadar bulunan en kısa yol **tahmini** (başta ∞ , kaynakta 0)
- $\pi[v]$: bu yolda v'den önceki düğüm (önceki, predecessor)

RELAX(u, v, w):

eğer $d[u] + w(u, v) < d[v]$:

$d[v] = d[u] + w(u, v)$

$\pi[v] = u$



$$d[u] + w = 5 + 3 = 8 < 9 \rightarrow d[v] = 8, \pi[v] = u$$

Bu bölümdeki tüm algoritmalar gevşetmeyi kullanır; fark, kenarları **hangi sırayla** ve **kaç kez** gevşettikleridir. $d[v]$ yalnızca azalabilir ve hiçbir zaman gerçek $\delta(s, v)$ 'nin altına inmez.

En Kısa Yolların Özellikleri

Optimal alt yapı

En kısa yolun her alt yolu da en kısa yoldur. $s \rightsquigarrow u \rightsquigarrow v$ en kısaysa $s \rightsquigarrow u$ parçası da en kısadır. Dinamik programlama ve açgözlü yaklaşım buna dayanır.

Üçgen eşitsizliği

Her (u, v) kenarı için $\delta(s, v) \leq \delta(s, u) + w(u, v)$.
Gevşetme bu eşitsizliği bozan kenarları düzeltir.

Negatif döngü

Toplam ağırlığı negatif olan bir döngü kaynaktan ulaşılabilirse, döngüde dolaştıkça yol kısalır: en kısa yol **tanımsızdır** $(-\infty)$.

Basit yol

Negatif döngü yoksa en kısa yol döngü içermez; en fazla **$V - 1$** kenarlıdır. Bellman-Ford'un $V - 1$ turu buradan gelir.

Yolu geri kurmak için π dizisi kullanılır: hedeften başlayıp $\pi[v]$ ile kaynağa kadar geri gidilir; elde edilen düğümler ters sırada yazılır. Tüm π kenarları birlikte bir **en kısa yol ağacı** oluşturur.

Dijkstra Algoritması

- **Kaynak** düğümden diğer tüm düğümlere olan en kısa yolları bulur (tek kaynaklı).
- 1956'da **Edsger W. Dijkstra** tarafından geliştirildi, 1959'da yayımlandı.
- Yalnızca **negatif olmayan** ağırlıklı kenarlardan oluşan çizgelerde doğru çalışır.
- **Açgözlü** (greedy): her adımda, henüz kesinleşmemiş düğümler arasından d değeri **en küçük** olanı seçer ve kesinleştirir.
- Seçilen düğümün komşuları gevşetilir.

Sezgi: su dalgası

Kaynağa bir taş atıldığını düşünün. Dalga, düğümlere uzaklık sırasıyla ulaşır: önce en yakın düğüm, sonra bir sonraki... Dijkstra düğümleri tam da bu sırayla **kesinleştirir**. Kesinleşen düğümlerin kümesine **S** denir.

Dijkstra: Algoritma Adımları

Adım 1: Başlangıç düğümü seçilir, uzaklığı **0** atanır; diğer tüm düğümlerin uzaklığı ∞ olur.

İklendirme: $d[s] = 0$, $d[v] = \infty$, $\pi[v] = \text{tanımsız}$, $S = \emptyset$.

Adım 1 / 5

Dijkstra: Algoritma Adımları

Adım 1: Başlangıç düğümü seçilir, uzaklığı **0** atanır; diğer tüm düğümlerin uzaklığı ∞ olur.

İklendirme: $d[s] = 0$, $d[v] = \infty$, $\pi[v] = \text{tanımsız}$, $S = \emptyset$.

Adım 2: Seçilen düğümden başlayarak, henüz işlenmemiş **komşulara** olan uzaklıklar hesaplanır.

Komşu v için aday uzaklık: $d[u] + w(u, v)$.

Dijkstra: Algoritma Adımları

Adım 1: Başlangıç düğümü seçilir, uzaklığı **0** atanır; diğer tüm düğümlerin uzaklığı ∞ olur.

İklendirme: $d[s] = 0$, $d[v] = \infty$, $\pi[v] = \text{tanımsız}$, $S = \emptyset$.

Adım 2: Seçilen düğümden başlayarak, henüz işlenmemiş **komşulara** olan uzaklıklar hesaplanır.

Komşu v için aday uzaklık: $d[u] + w(u, v)$.

Adım 3: Daha kısa bir yol bulunursa komşunun uzaklığı **güncellenir** (gevşetme).

$d[u] + w(u, v) < d[v]$ ise $d[v]$ ve $\pi[v]$ değişir.

Dijkstra: Algoritma Adımları

Adım 1: Başlangıç düğümü seçilir, uzaklığı **0** atanır; diğer tüm düğümlerin uzaklığı ∞ olur.

İklendirme: $d[s] = 0$, $d[v] = \infty$, $\pi[v] = \text{tanımsız}$, $S = \emptyset$.

Adım 2: Seçilen düğümden başlayarak, henüz işlenmemiş **komşulara** olan uzaklıklar hesaplanır.

Komşu v için aday uzaklık: $d[u] + w(u, v)$.

Adım 3: Daha kısa bir yol bulunursa komşunun uzaklığı **güncellenir** (gevşetme).

$d[u] + w(u, v) < d[v]$ ise $d[v]$ ve $\pi[v]$ değişir.

Adım 4: İşlenen düğüm **işaretlenir** (S 'ye eklenir) ve uzaklığı en küçük işaretsiz düğüm **seçilir**.

Seçim, dizi ya da öncelikli kuyruk (priority queue) ile yapılır.

Dijkstra: Algoritma Adımları

Adım 1: Başlangıç düğümü seçilir, uzaklığı **0** atanır; diğer tüm düğümlerin uzaklığı ∞ olur.

İklendirme: $d[s] = 0$, $d[v] = \infty$, $\pi[v] = \text{tanımsız}$, $S = \emptyset$.

Adım 2: Seçilen düğümden başlayarak, henüz işlenmemiş **komşulara** olan uzaklıklar hesaplanır.

Komşu v için aday uzaklık: $d[u] + w(u, v)$.

Adım 3: Daha kısa bir yol bulunursa komşunun uzaklığı **güncellenir** (gevşetme).

$d[u] + w(u, v) < d[v]$ ise $d[v]$ ve $\pi[v]$ değişir.

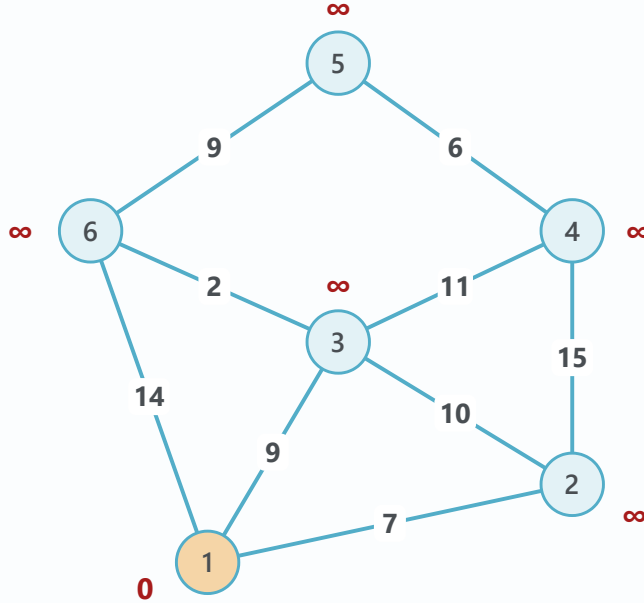
Adım 4: İşlenen düğüm **işaretlenir** (S 'ye eklenir) ve uzaklığı en küçük işaretsiz düğüm **seçilir**.

Seçim, dizi ya da öncelikli kuyruk (priority queue) ile yapılır.

Adım 5: Tüm düğümler işlenene kadar Adım 2–4 **tekrarlanır**.

Ulaşılamayan düğümlerin uzaklığı ∞ olarak kalır.

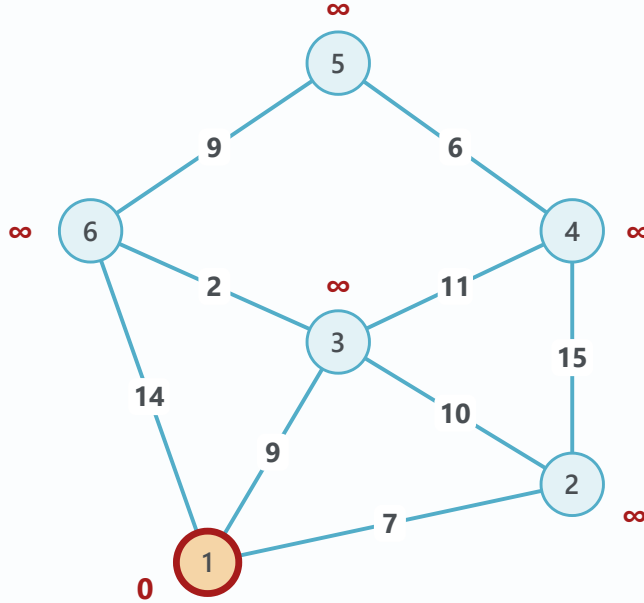
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	∞	–
3	∞	–
4	∞	–
5	∞	–
6	∞	–

İlklendirme: kaynak **1**, $d[1] = 0$, diğerleri ∞ . Hiçbir düğüm kesinleşmedi.

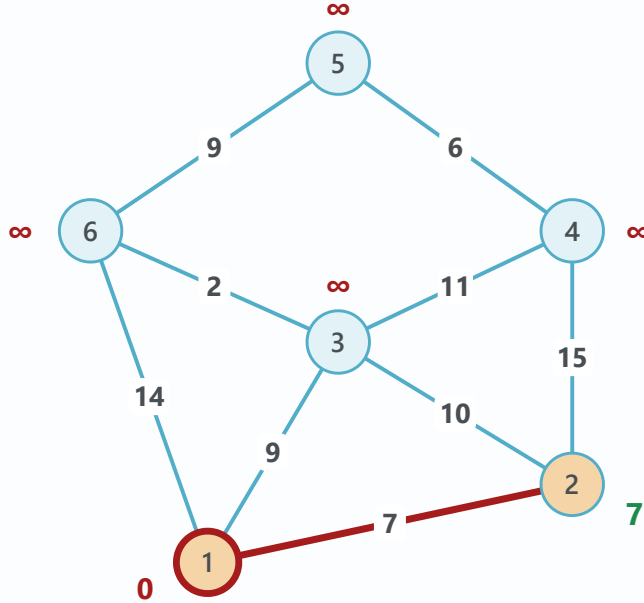
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	∞	–
3	∞	–
4	∞	–
5	∞	–
6	∞	–

Kesinleşmemişler içinde en küçük d: **1** ($d = 0$) seçilir ve kesinleşir.

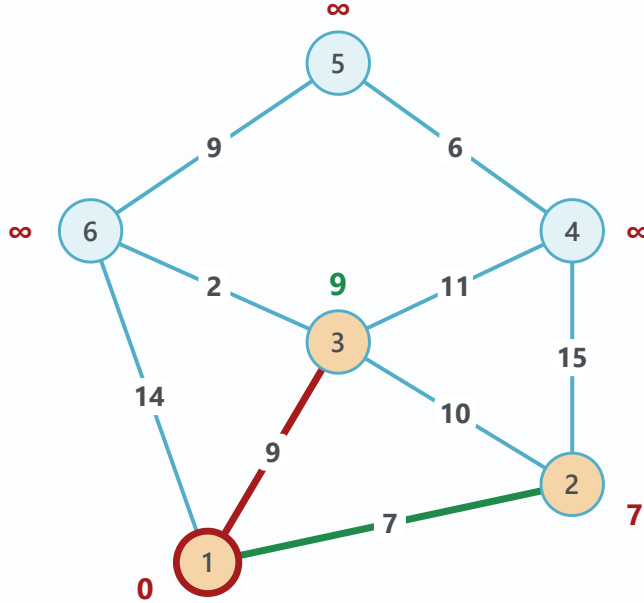
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	∞	–
4	∞	–
5	∞	–
6	∞	–

Kenar (1, 2): $d[1] + 7 = 7 < \infty \rightarrow \mathbf{d[2] = 7}, \pi[2] = 1.$

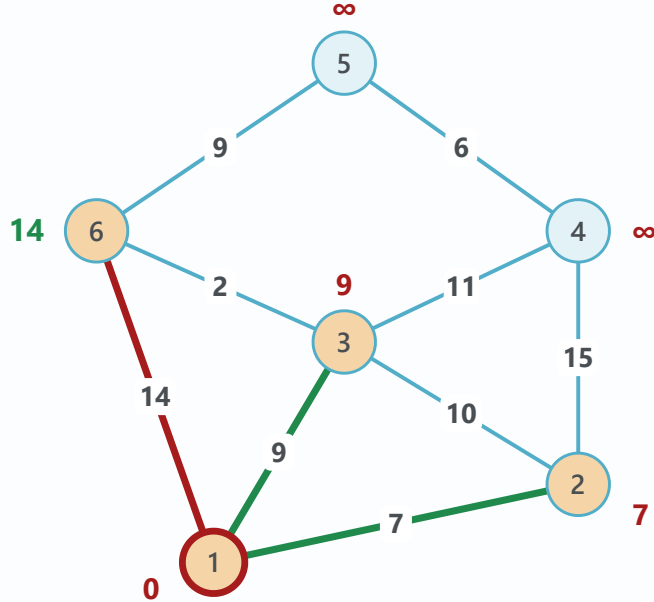
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	∞	–
5	∞	–
6	∞	–

Kenar (1, 3): $d[1] + 9 = 9 < \infty \rightarrow \mathbf{d[3] = 9}, \pi[3] = 1.$

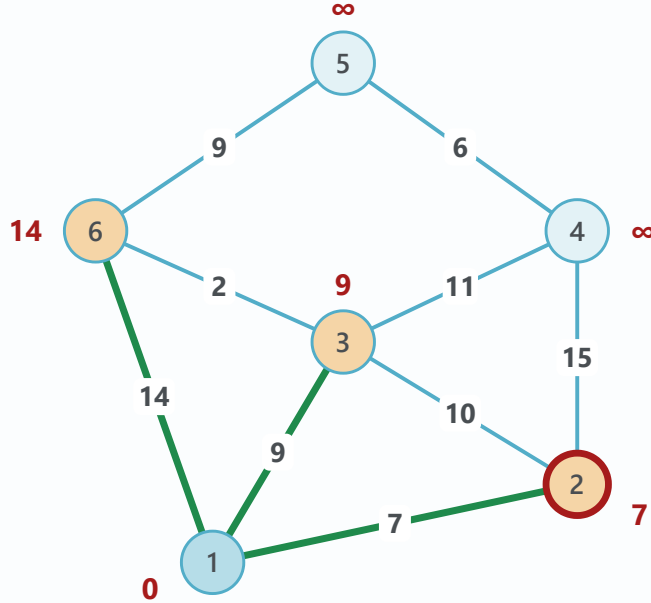
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	∞	–
5	∞	–
6	14	1

Kenar (1, 6): $d[1] + 14 = 14 < \infty \rightarrow \mathbf{d[6] = 14}, \pi[6] = 1.$

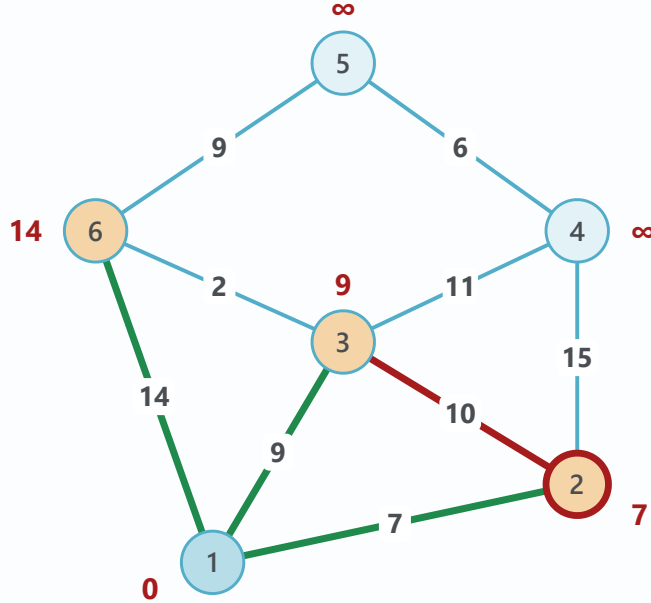
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	∞	–
5	∞	–
6	14	1

Kesinleşmemişler içinde en küçük d: **2** ($d = 7$) seçilir ve kesinleşir. Kesinleşmiş komşular (1) atlanır.

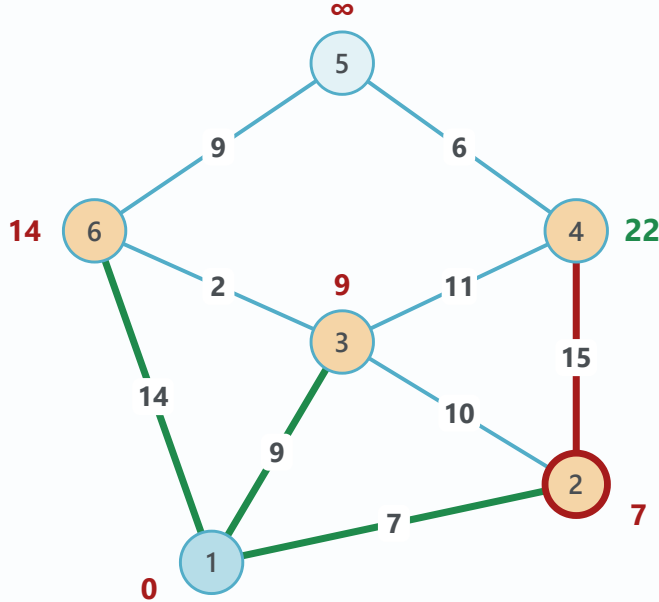
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	∞	–
5	∞	–
6	14	1

Kenar (2, 3): $d[2] + 10 = 17 \geq 9 \rightarrow$ değişiklik yok.

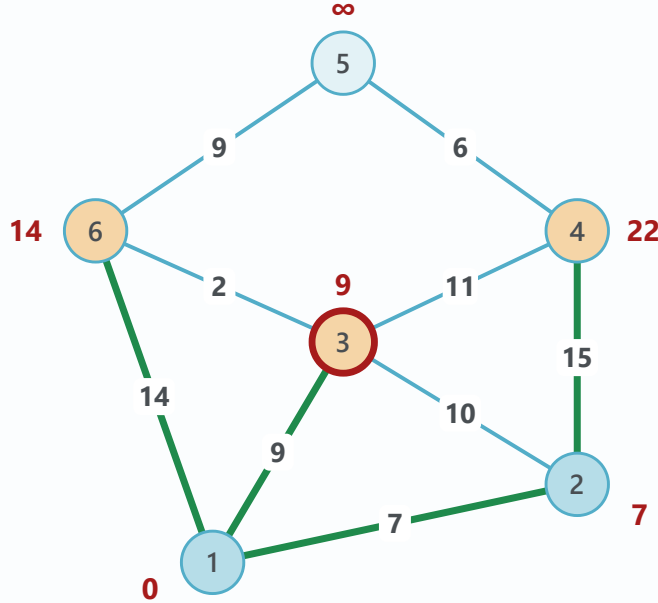
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	22	2
5	∞	–
6	14	1

Kenar (2, 4): $d[2] + 15 = 22 < \infty \rightarrow \mathbf{d[4] = 22}, \pi[4] = 2.$

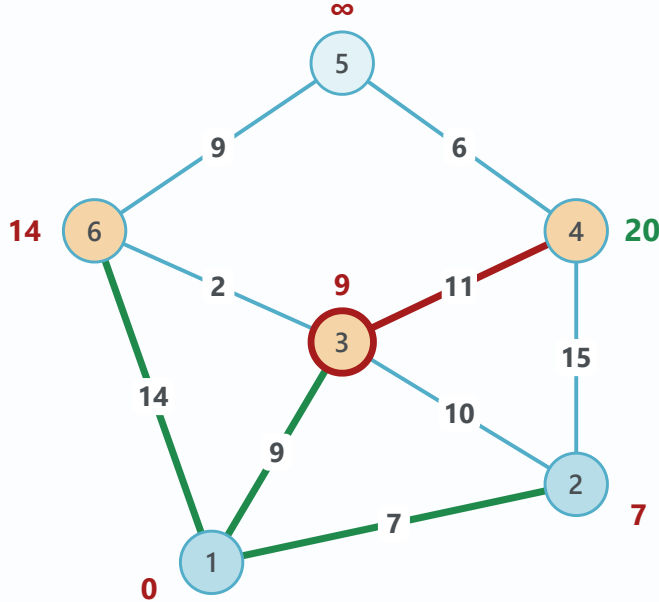
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	22	2
5	∞	–
6	14	1

Kesinleşmemişler içinde en küçük d: **3** ($d = 9$) seçilir ve kesinleşir. Kesinleşmiş komşular (1, 2) atlanır.

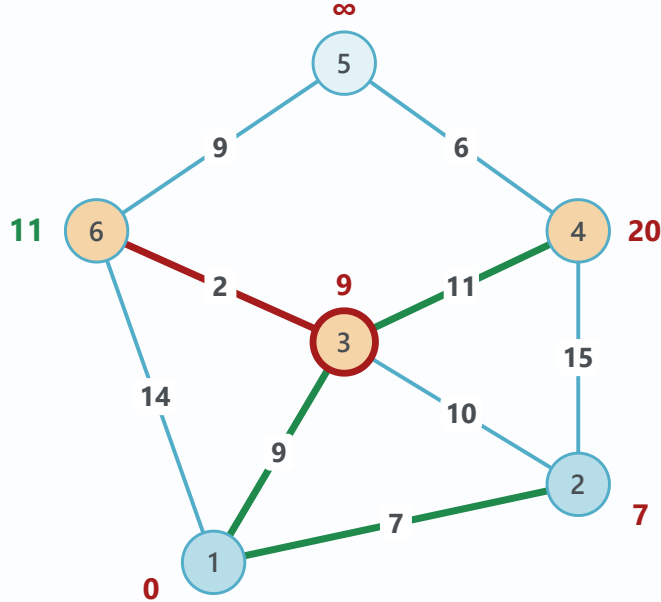
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	20	3
5	∞	–
6	14	1

Kenar (3, 4): $d[3] + 11 = 20 < 22 \rightarrow \mathbf{d[4] = 20}$, $\pi[4] = 3$.

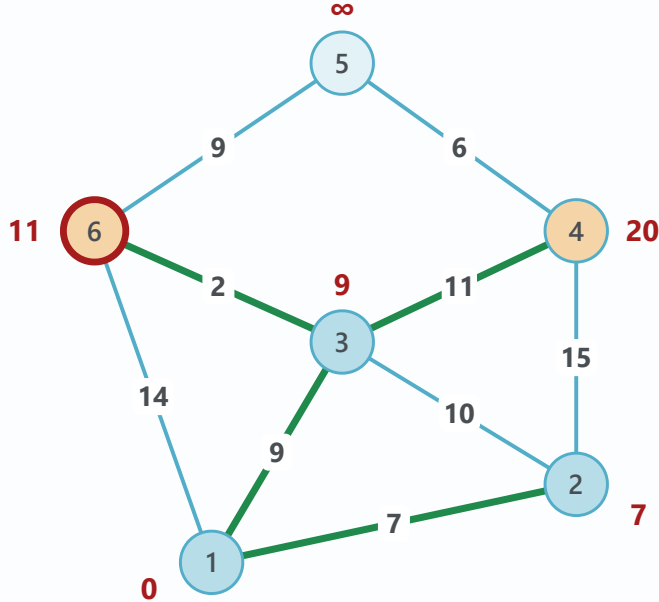
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	20	3
5	∞	–
6	11	3

Kenar (3, 6): $d[3] + 2 = 11 < 14 \rightarrow \mathbf{d[6] = 11}, \pi[6] = 3.$

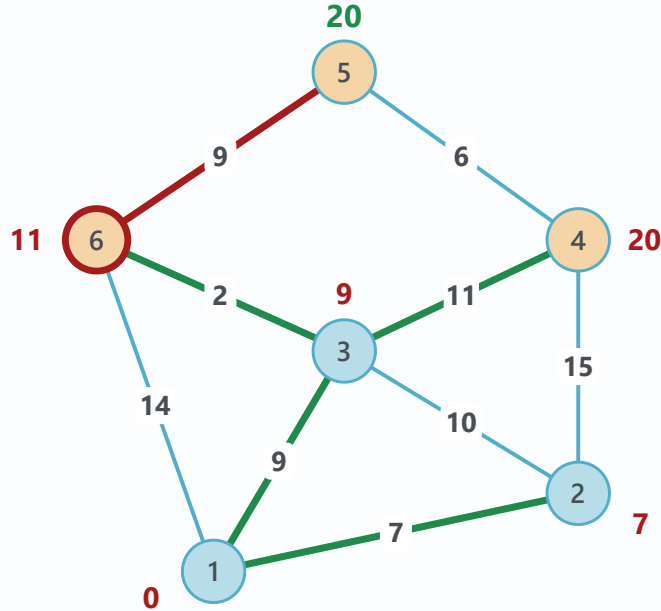
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	20	3
5	∞	–
6	11	3

Kesinleşmemişler içinde en küçük d: **6** ($d = 11$) seçilir ve kesinleşir. Kesinleşmiş komşular (1, 3) atlanır.

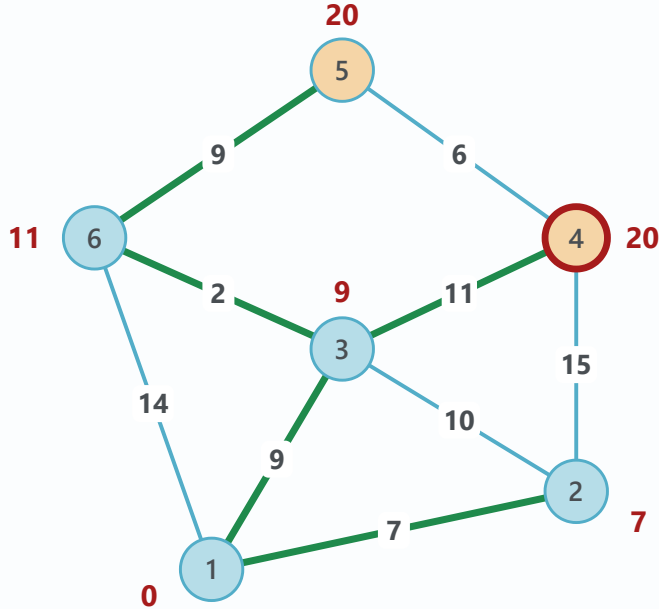
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3

Kenar (6, 5): $d[6] + 9 = 20 < \infty \rightarrow \mathbf{d[5] = 20}, \pi[5] = 6.$

Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1

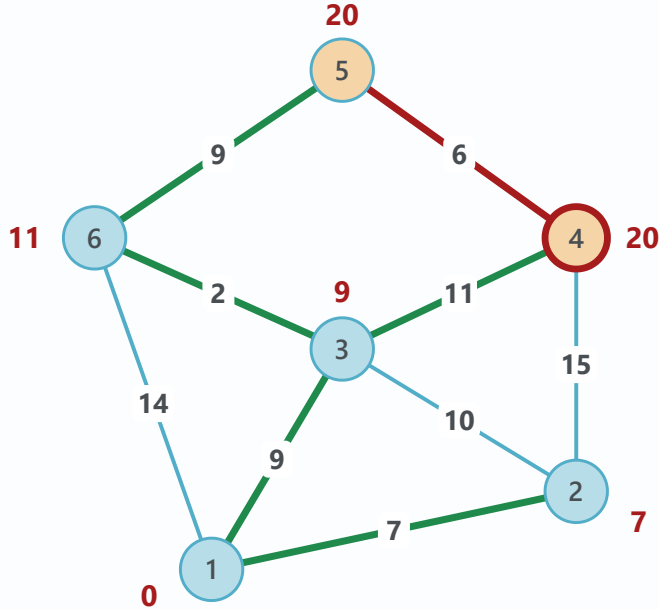


düğüm	d	π
1	0	–
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3

Kesinleşmemişler içinde en küçük d: **4** ($d = 20$) seçilir ve kesinleşir. (4 ve 5 eşit: 20; küçük numaralı 4 alındı.)

Kesinleşmiş komşular (2, 3) atlanır.

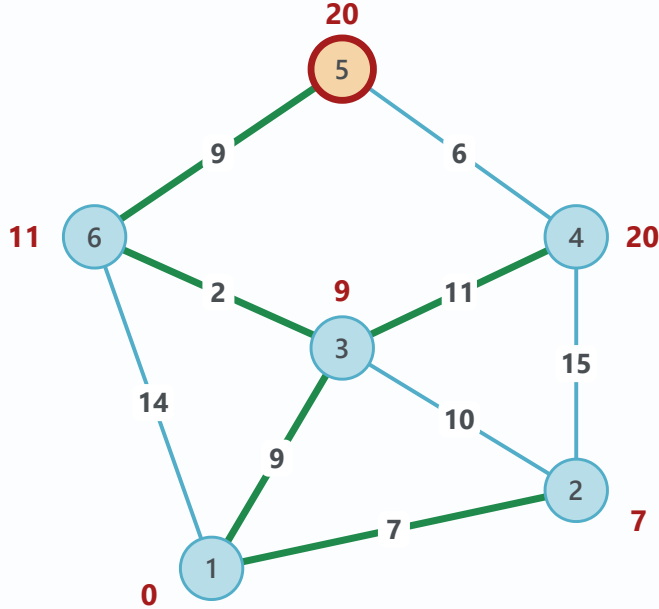
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3

Kenar (4, 5): $d[4] + 6 = 26 \geq 20 \rightarrow$ değişiklik yok.

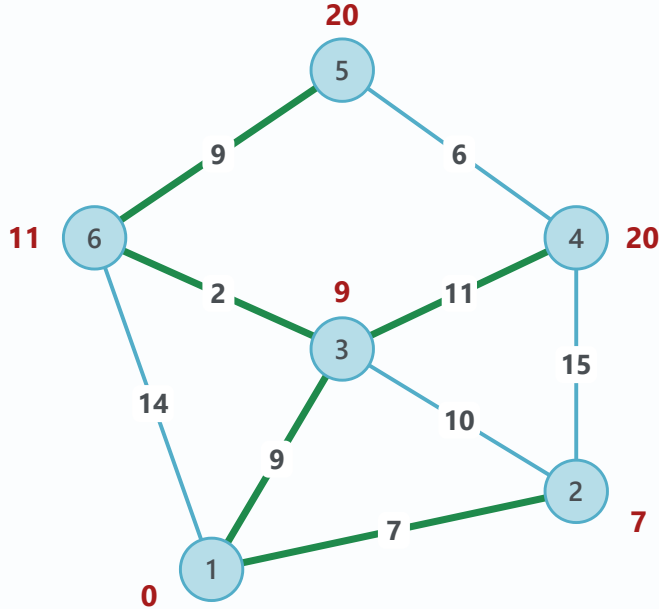
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3

Kesinleşmemişler içinde en küçük d: **5** ($d = 20$) seçilir ve kesinleşir. Kesinleşmiş komşular (4, 6) atlanır.

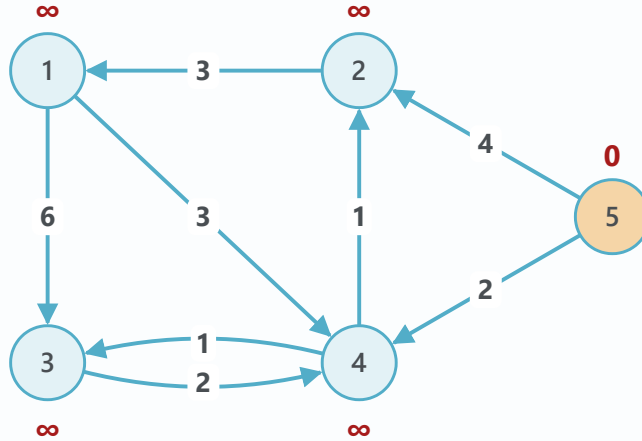
Dijkstra Örnek 1: Yönsüz Çizge, Kaynak 1



düğüm	d	π
1	0	–
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3

Bitti: 1'den uzaklıklar 2:7, 3:9, 6:11, 4:20, 5:20. Yeşil kenarlar **en kısa yol ağacını** oluşturur (ör. 1 → 3 → 6 → 5).

Dijkstra Örnek 2: Yönlü Çizge, Q ve S Kümeleri



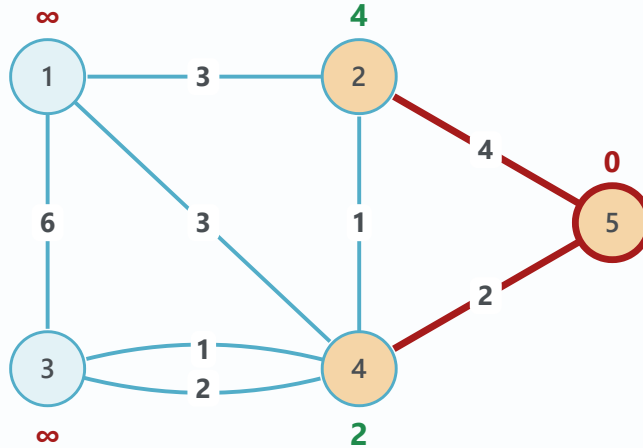
Q		
d	u	π
0	5	/
∞	1	/
∞	2	/
∞	3	/
∞	4	/

S		
u	d	π

İklendirme: kaynak **5**'in uzaklığı 0, diğerleri ∞ . Tüm düğümler Q'da, $S = \emptyset$.

DIJKSTRA

Dijkstra Örnek 2: Yönlü Çizge, Q ve S Kümeleri

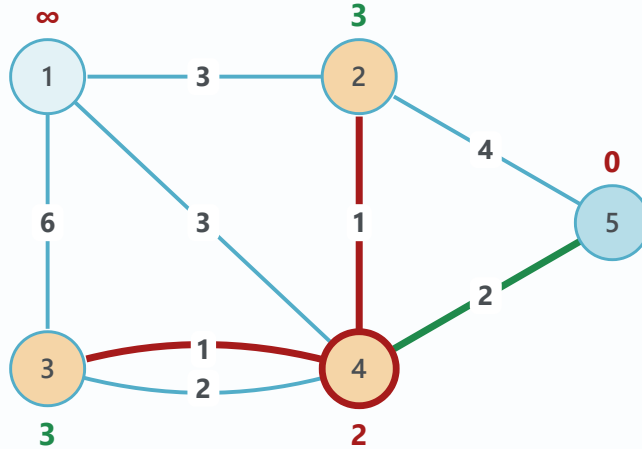


Q			S		
d	u	π	u	d	π
2	4	5	5	0	/
4	2	5			
∞	1	/			
∞	3	/			

Adım 1: Q'dan **5** alınır, $d = 0$ ile S'ye konur. Gevşetme: $(5,2): \infty \rightarrow \mathbf{4}$; $(5,4): \infty \rightarrow \mathbf{2}$.

Adım 2 / 7

Dijkstra Örnek 2: Yönlü Çizge, Q ve S Kümeleri

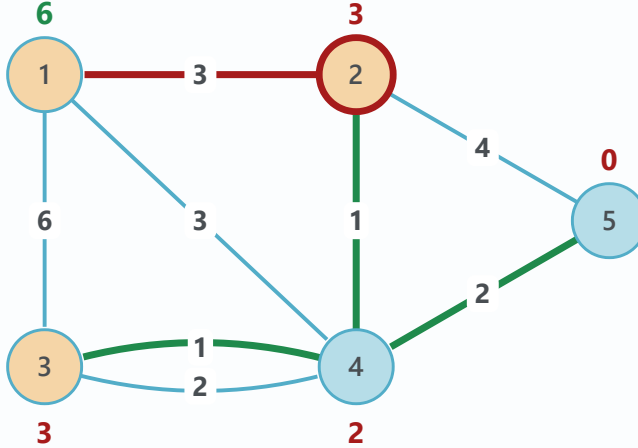


Q			S		
d	u	π	u	d	π
3	2	4	5	0	/
3	3	4	4	2	5
∞	1	/			

Adım 2: Q'dan **4** alınır, $d = 2$ ile S'ye konur. Gevşetme: $(4,2)$: $4 \rightarrow \mathbf{3}$; $(4,3)$: $\infty \rightarrow \mathbf{3}$.

Adım 3 / 7

Dijkstra Örnek 2: Yönlü Çizge, Q ve S Kümeleri

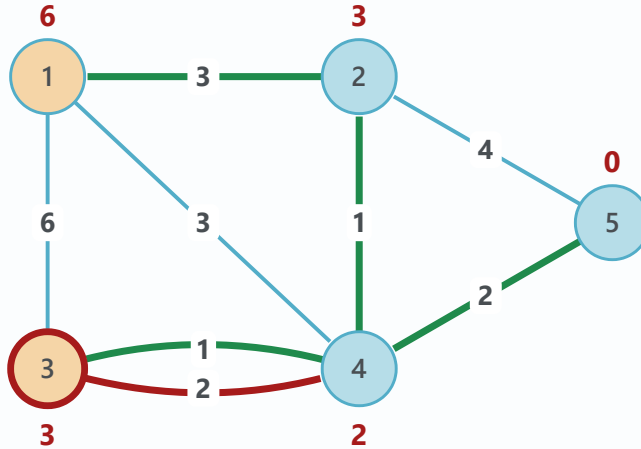


Q			S		
d	u	π	u	d	π
3	3	4	5	0	/
6	1	2	4	2	5
			2	3	4

Adım 3: Q'dan **2** alınır, $d = 3$ ile S'ye konur. Gevşetme: $(2,1): \infty \rightarrow \mathbf{6}$. (2 ve 3 eşit: 3; küçük numaralı 2 alındı.)

Adım 4 / 7

Dijkstra Örnek 2: Yönlü Çizge, Q ve S Kümeleri

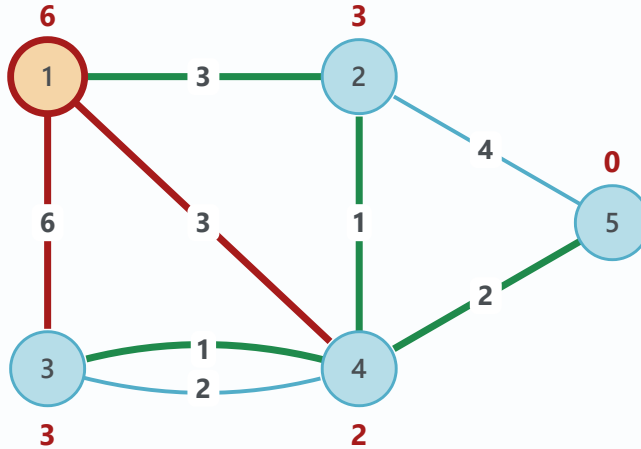


Q		
d	u	π
6	1	2

S		
u	d	π
5	0	/
4	2	5
2	3	4
3	3	4

Adım 4: Q'dan **3** alınır, $d = 3$ ile S'ye konur. Gevşetme: (3,4): 4 zaten S'de. Daha kısa yol çıkmaz (gevşetme yok).

Dijkstra Örnek 2: Yönlü Çizge, Q ve S Kümeleri



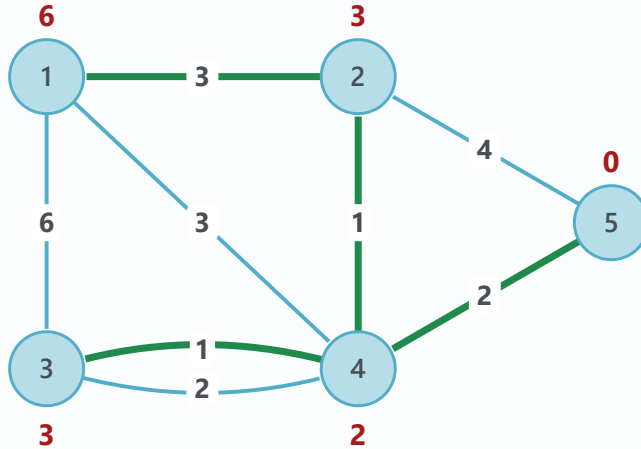
Q		
d	u	π

S		
u	d	π
5	0	/
4	2	5
2	3	4
3	3	4
1	6	2

Adım 5: Q'dan **1** alınır, $d = 6$ ile S'ye konur. Gevşetme: (1,3): 3 zaten S'de; (1,4): 4 zaten S'de. Daha kısa yol çıkmaz (gevşetme yok).

DIJKSTRA

Dijkstra Örnek 2: Yönlü Çizge, Q ve S Kümeleri



	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

Son durum: 5'ten en kısa yollar. Yalnızca π kenarları (yeşil) kalır: $5 \rightarrow 4 \rightarrow 2 \rightarrow 1$ (6), $5 \rightarrow 4 \rightarrow 3$ (3).

DIJKSTRA

Dijkstra: Sözde Kod

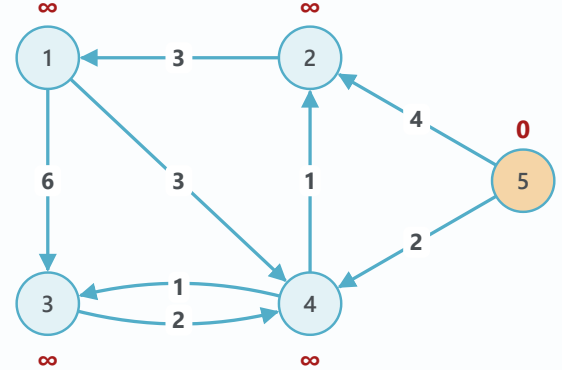
```
DIJKSTRA(G, kaynak):  
    mesafeler = [ ]; ziyaretEdildi = [ ]  
    öncelikKuyruğu = { }  
    her bir v için V içinde:  
        mesafeler[v] = sonsuz  
        ziyaretEdildi[v] = 0  
    mesafeler[kaynak] = 0  
    öncelikKuyruğu.ekle(kaynak, 0)  
    döngü öncelikKuyruğu boş değil iken:  
        u = öncelikKuyruğu.çıkar() // en küçük mesafe  
        eğer ziyaretEdildi[u] == 0:  
            ziyaretEdildi[u] = 1  
            her bir (u, v) için G[u] içinde:  
                eğer mesafeler[u] + ağırlık(u, v) < mesafeler[v]:  
                    mesafeler[v] = mesafeler[u] + ağırlık(u, v)  
                    öncelikKuyruğu.ekle(v, mesafeler[v])  
    döndür mesafeler
```

Tembel silme: bir düğüm kuyruğa birden çok kez girebilir; eski kayıt çıkarıldığında ziyaretEdildi kontrolü onu atlar. Yol için gevşetmede önceki[v] = u da saklanır.

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {  
    ▶ int[] uzaklik = new int[V + 1];  
    ▶ int[] onceki = new int[V + 1];  
    ▶ boolean[] ziyaret = new boolean[V + 1];  
    ▶ Arrays.fill(uzaklik, SONSUZ);  
    ▶ uzaklik[kaynak] = 0;  
    ▶ PriorityQueue<int[]> kuyruk =  
    ▶     new PriorityQueue<>((a, b) -> a[1] - b[1]);  
    ▶ kuyruk.add(new int[] { kaynak, 0 });  
  
10 while (!kuyruk.isEmpty()) {  
11     int u = kuyruk.poll()[0];  
12     if (ziyaret[u]) continue; // eski kayıt  
13     ziyaret[u] = true;  
14     for (Kenar k : komsu.get(u)) {  
15         int v = k.hedef, w = k.agirlik;  
16         if (!ziyaret[v]  
17             && uzaklik[u] + w < uzaklik[v]) {  
18             uzaklik[v] = uzaklik[u] + w;  
19             onceki[v] = u;  
20             kuyruk.add(new int[] { v, uzaklik[v] });  
21         }  
22     }  
23 }  
24 }
```



Diziler kurulur (düğümler 1..5, indis 0 kullanılmaz); uzaklik[5] = 0 ve (5, 0) kuyruğa eklenir.

Adım 1 / 27

DEĞİŞKENLER

u = -

v = -

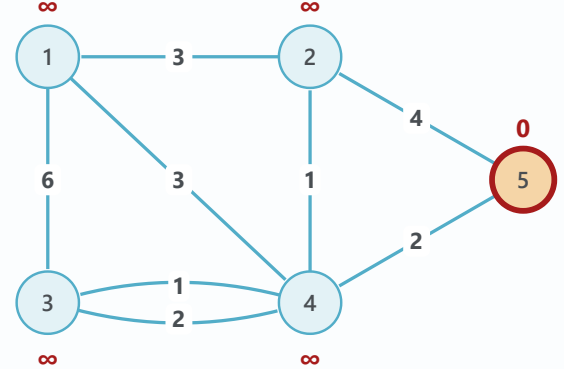
w = -

kuyruk = [(5,0)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    ► int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kuyruktan en küçük uzaklıklı kayıt **(5, 0)** alınır:
u = 5.

Adım 2 / 27

DEĞİŞKENLER

u = 5

v = -

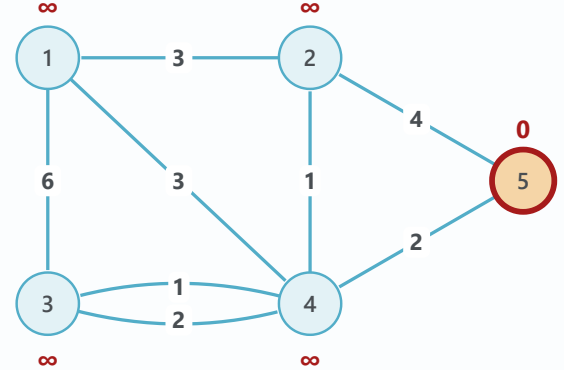
w = -

kuyruk = []

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ► ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



5 kesinleşir: ziyaret[5] = true.

Adım 3 / 27

DEĞİŞKENLER

u = 5

v = -

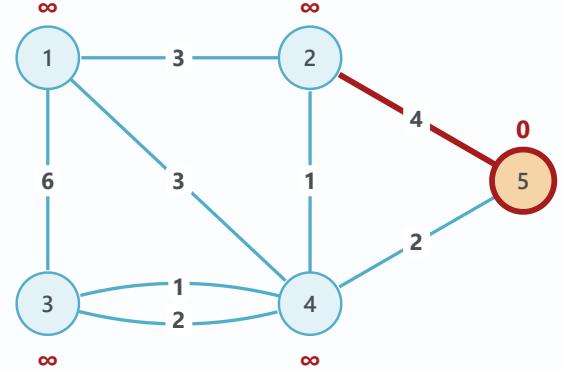
w = -

kuyruk = []

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      ▶ int v = k.hedef, w = k.agirlik;
16      ▶ if (!ziyaret[v]
17      ▶    && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kenar 5→2 (w = 4): $0 + 4 < \infty \rightarrow$ koşul **doğru**.

Adım 4 / 27

DEĞİŞKENLER

u = 5

v = 2

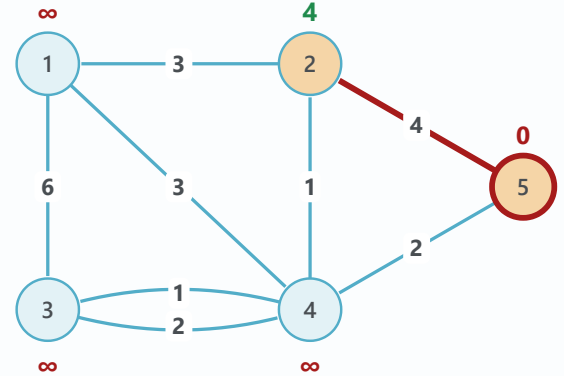
w = 4

kuyruk = []

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        ▶ uzaklik[v] = uzaklik[u] + w;
19        ▶ onceki[v] = u;
20        ▶ kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



uzaklik[2] = 4, onceki[2] = 5; (2, 4) kuyruğa eklenir.

Adım 5 / 27

DEĞİŞKENLER

u = 5

v = 2

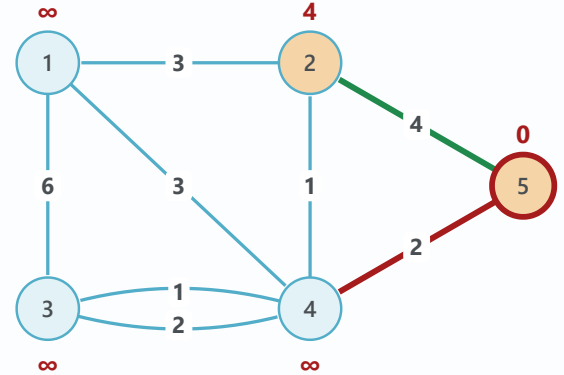
w = 4

kuyruk = [(2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      ▶ int v = k.hedef, w = k.agirlik;
16      ▶ if (!ziyaret[v]
17      ▶   && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kenar 5→4 (w = 2): $0 + 2 < \infty \rightarrow$ koşul **doğru**.

Adım 6 / 27

DEĞİŞKENLER

u = 5

v = 4

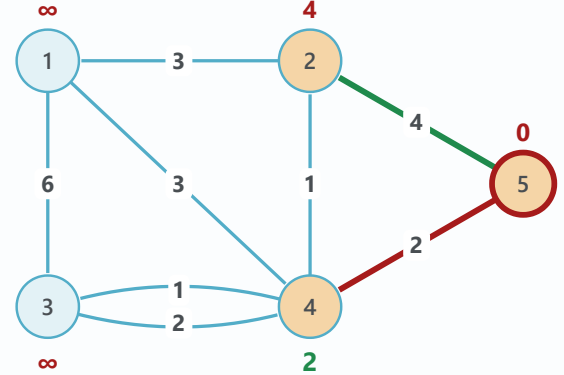
w = 2

kuyruk = [(2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        ▶ uzaklik[v] = uzaklik[u] + w;
19        ▶ onceki[v] = u;
20        ▶ kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



uzaklik[4] = 2, onceki[4] = 5; (4, 2) kuyruğa eklenir.

Adım 7 / 27

DEĞİŞKENLER

u = 5

v = 4

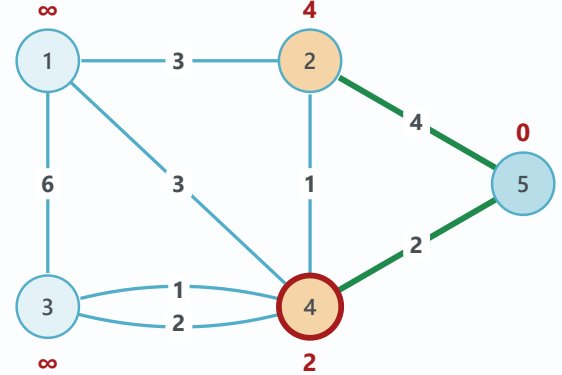
w = 2

kuyruk = [(4,2), (2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    ► int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kuyruktan en küçük uzaklıklı kayıt **(4, 2)** alınır:
u = 4.

Adım 8 / 27

DEĞİŞKENLER

u = 4

v = -

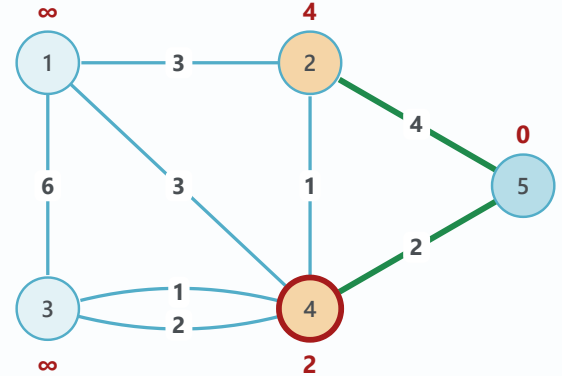
w = -

kuyruk = [(2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ► ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



4 kesinleşir: ziyaret[4] = true.

Adım 9 / 27

DEĞİŞKENLER

u = 4

v = -

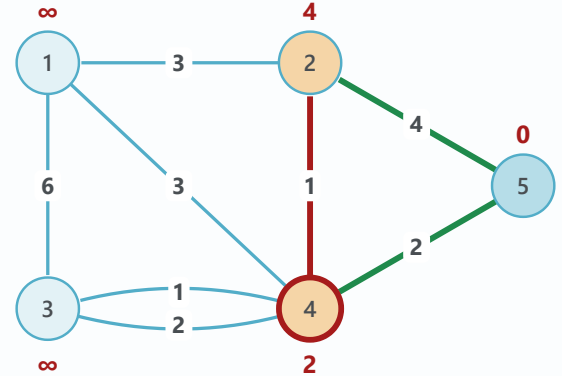
w = -

kuyruk = [(2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      ▶ int v = k.hedef, w = k.agirlik;
16      ▶ if (!ziyaret[v]
17      ▶   && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kenar 4→2 (w = 1): $2 + 1 < 4 \rightarrow$ koşul **doğru**.

Adım 10 / 27

DEĞİŞKENLER

u = 4

v = 2

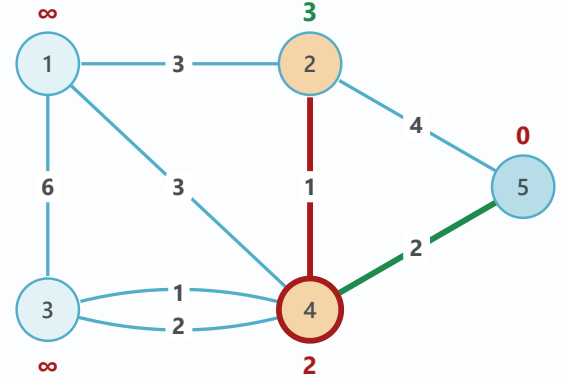
w = 1

kuyruk = [(2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        ▶ uzaklik[v] = uzaklik[u] + w;
19        ▶ onceki[v] = u;
20        ▶ kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



uzaklik[2] = 3, onceki[2] = 4; (2, 3) kuyruğa eklenir.

Adım 11 / 27

DEĞİŞKENLER

u = 4

v = 2

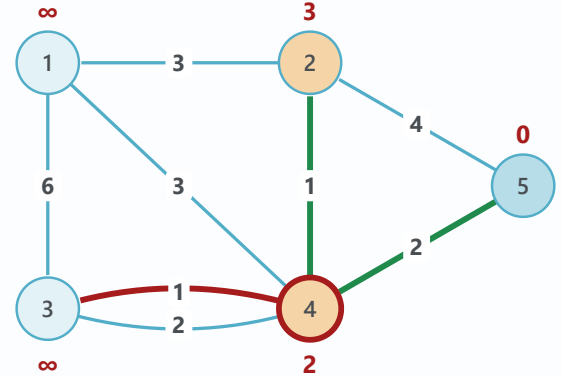
w = 1

kuyruk = [(2,3), (2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      ▶ int v = k.hedef, w = k.agirlik;
16      ▶ if (!ziyaret[v]
17      ▶   && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kenar 4→3 (w = 1): $2 + 1 < \infty \rightarrow$ koşul **doğru**.

Adım 12 / 27

DEĞİŞKENLER

u = 4

v = 3

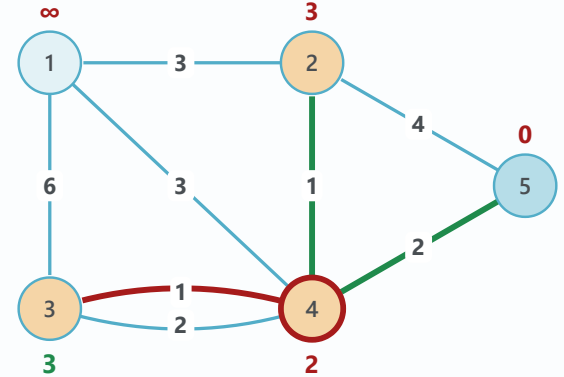
w = 1

kuyruk = [(2,3), (2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        ▶ uzaklik[v] = uzaklik[u] + w;
19        ▶ onceki[v] = u;
20        ▶ kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



uzaklik[3] = 3, onceki[3] = 4; (3, 3) kuyruğa eklenir.

Adım 13 / 27

DEĞİŞKENLER

u = 4

v = 3

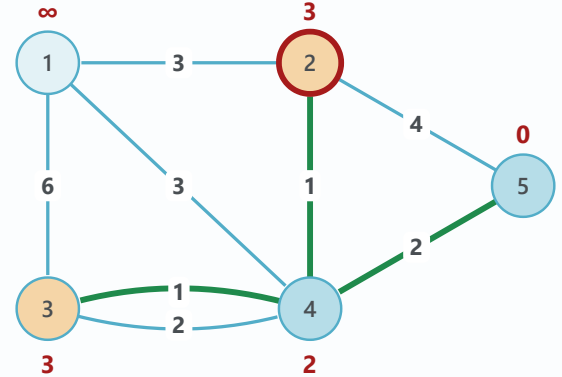
w = 1

kuyruk = [(2,3), (3,3), (2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    ► int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kuyruktan en küçük uzaklıklı kayıt **(2, 3)** alınır:
u = 2.

Adım 14 / 27

DEĞİŞKENLER

u = 2

v = -

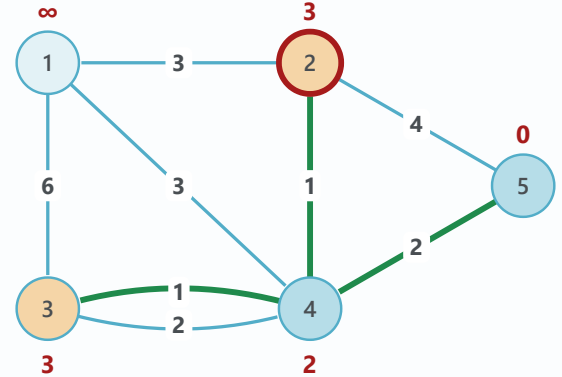
w = -

kuyruk = [(3,3), (2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ► ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



2 kesinleşir: ziyaret[2] = true.

Adım 15 / 27

DEĞİŞKENLER

u = 2

v = -

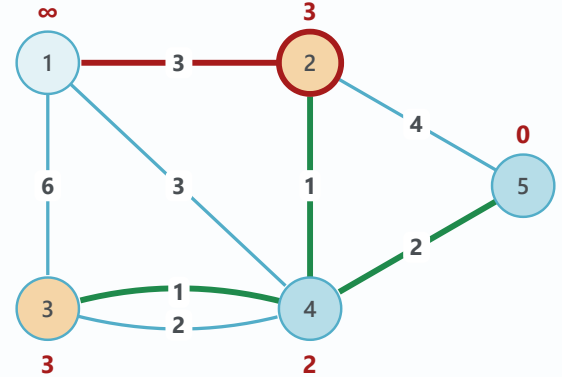
w = -

kuyruk = [(3,3), (2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      ▶ int v = k.hedef, w = k.agirlik;
16      ▶ if (!ziyaret[v]
17      ▶   && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kenar 2→1 (w = 3): $3 + 3 < \infty \rightarrow$ koşul **doğru**.

Adım 16 / 27

DEĞİŞKENLER

u = 2

v = 1

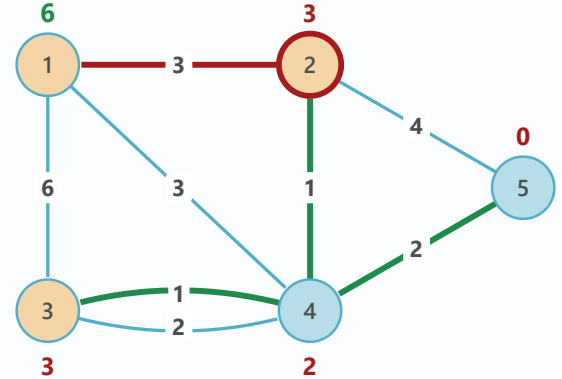
w = 3

kuyruk = [(3,3), (2,4)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        ▶ uzaklik[v] = uzaklik[u] + w;
19        ▶ onceki[v] = u;
20        ▶ kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



uzaklik[1] = 6, onceki[1] = 2; (1, 6) kuyruğa eklenir.

Adım 17 / 27

DEĞİŞKENLER

u = 2

v = 1

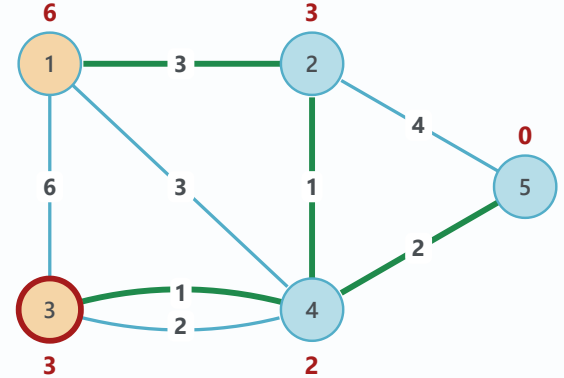
w = 3

kuyruk = [(3,3), (2,4), (1,6)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    ► int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kuyruktan en küçük uzaklıklı kayıt **(3, 3)** alınır:
u = 3.

Adım 18 / 27

DEĞİŞKENLER

u = 3

v = -

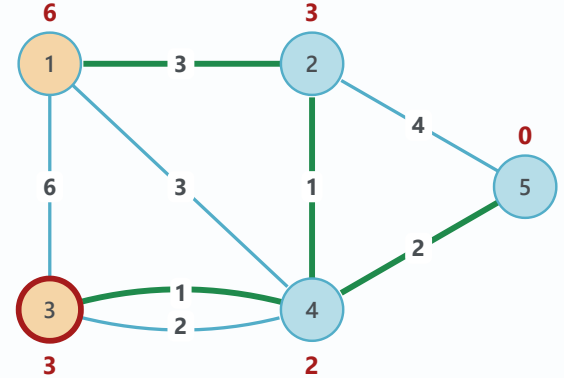
w = -

kuyruk = [(2,4), (1,6)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ► ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



3 kesinleşir: ziyaret[3] = true.

Adım 19 / 27

DEĞİŞKENLER

u = 3

v = -

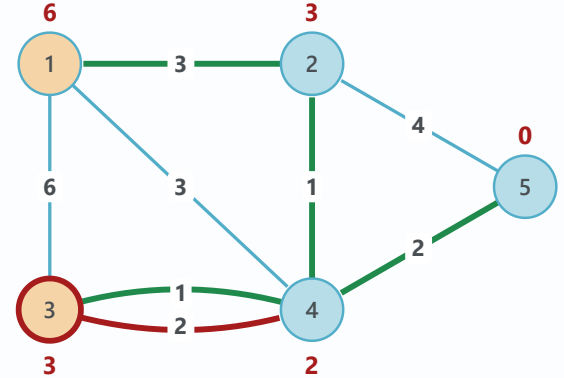
w = -

kuyruk = [(2,4), (1,6)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      ▶ int v = k.hedef, w = k.agirlik;
16      ▶ if (!ziyaret[v]
17      ▶   && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kenar 3→4 (w = 2): ziyaret[4] = true → koşul yanlış.

Adım 20 / 27

DEĞİŞKENLER

u = 3

v = 4

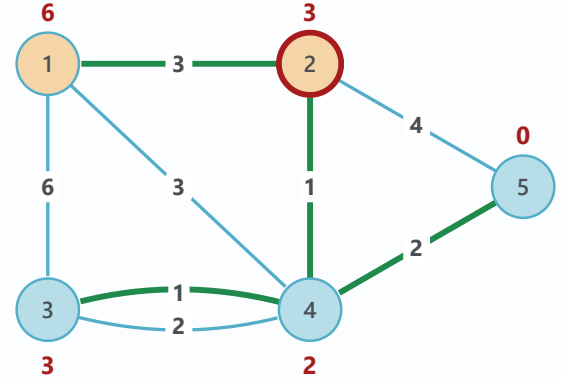
w = 2

kuyruk = [(2,4), (1,6)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2     int[] uzaklik = new int[V + 1];
3     int[] onceki = new int[V + 1];
4     boolean[] ziyaret = new boolean[V + 1];
5     Arrays.fill(uzaklik, SONSUZ);
6     uzaklik[kaynak] = 0;
7     PriorityQueue<int[]> kuyruk =
8         new PriorityQueue<>((a, b) -> a[1] - b[1]);
9     kuyruk.add(new int[] { kaynak, 0 });
10    while (!kuyruk.isEmpty()) {
11        ► int u = kuyruk.poll()[0];
12        if (ziyaret[u]) continue; // eski kayıt
13        ziyaret[u] = true;
14        for (Kenar k : komsu.get(u)) {
15            int v = k.hedef, w = k.agirlik;
16            if (!ziyaret[v]
17                && uzaklik[u] + w < uzaklik[v]) {
18                uzaklik[v] = uzaklik[u] + w;
19                onceki[v] = u;
20                kuyruk.add(new int[] { v, uzaklik[v] });
21            }
22        }
23    }
24 }
```



Kuyruktan en küçük uzaklıklı kayıt **(2, 4)** alınır:
u = 2.

Adım 21 / 27

DEĞİŞKENLER

u = 2

v = -

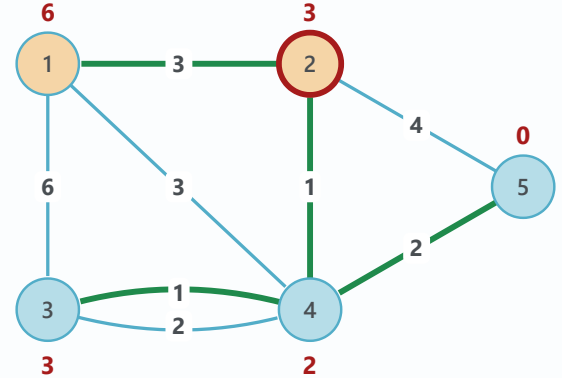
w = -

kuyruk = [(1,6)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    ▶ if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



ziyaret[2] zaten true: bu, 2'nin eski (2, 4) kaydı
→ `continue` ile atlanır (tembel silme).

Adım 22 / 27

DEĞİŞKENLER

u = 2

v = -

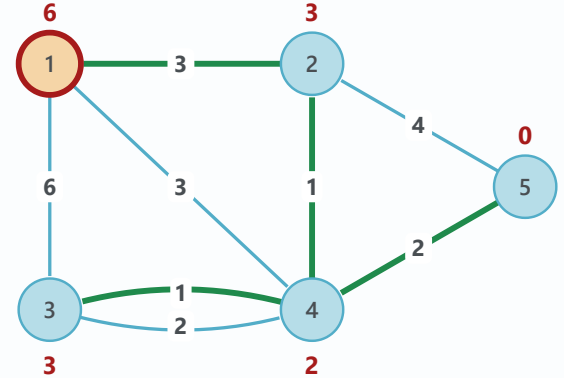
w = -

kuyruk = [(1,6)]

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    ► int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kuyruktan en küçük uzaklıklı kayıt **(1, 6)** alınır:
u = 1.

Adım 23 / 27

DEĞİŞKENLER

u = 1

v = -

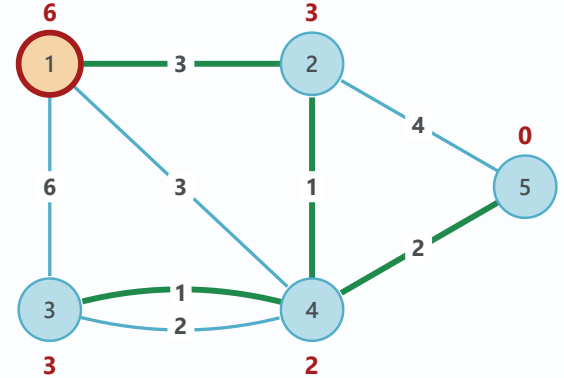
w = -

kuyruk = []

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ► ziyaret[u] = true;
14    for (Kenar k : komşu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



1 kesinleşir: ziyaret[1] = true.

Adım 24 / 27

DEĞİŞKENLER

u = 1

v = -

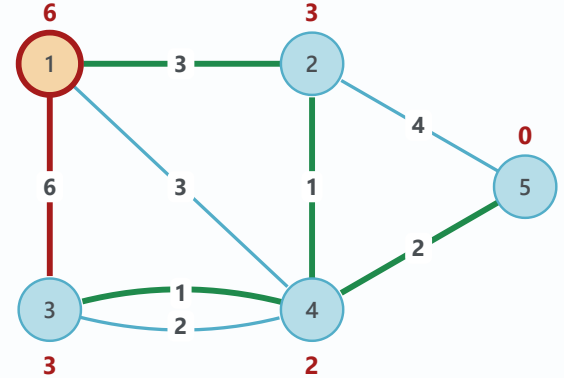
w = -

kuyruk = []

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      ▶ int v = k.hedef, w = k.agirlik;
16      ▶ if (!ziyaret[v]
17      ▶   && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kenar 1→3 (w = 6): ziyaret[3] = true → koşul yanlış.

Adım 25 / 27

DEĞİŞKENLER

u = 1

v = 3

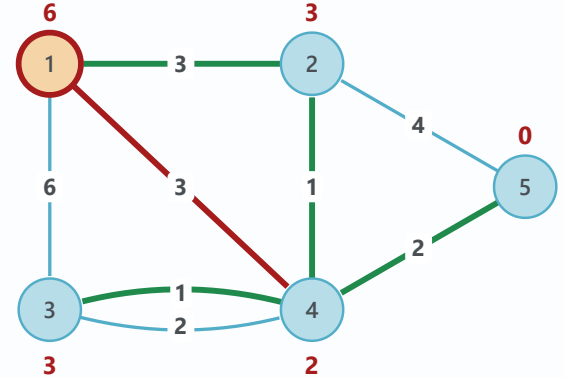
w = 6

kuyruk = []

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      ▶ int v = k.hedef, w = k.agirlik;
16      ▶ if (!ziyaret[v]
17      ▶   && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kenar 1→4 (w = 3): ziyaret[4] = true → koşul
yanlış.

Adım 26 / 27

DEĞİŞKENLER

u = 1

v = 4

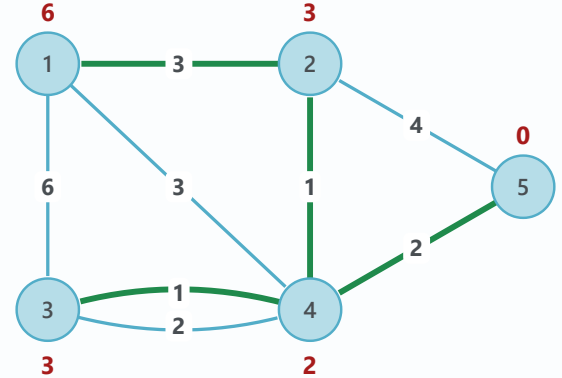
w = 3

kuyruk = []

DIJKSTRA

Dijkstra (Öncelikli Kuyruk): Kodu Adım Adım İzle

```
1 void dijkstra(int kaynak) {
2   int[] uzaklik = new int[V + 1];
3   int[] onceki = new int[V + 1];
4   boolean[] ziyaret = new boolean[V + 1];
5   Arrays.fill(uzaklik, SONSUZ);
6   uzaklik[kaynak] = 0;
7   PriorityQueue<int[]> kuyruk =
8     new PriorityQueue<>((a, b) -> a[1] - b[1]);
9   kuyruk.add(new int[] { kaynak, 0 });
10  while (!kuyruk.isEmpty()) {
11    int u = kuyruk.poll()[0];
12    if (ziyaret[u]) continue; // eski kayıt
13    ziyaret[u] = true;
14    for (Kenar k : komsu.get(u)) {
15      int v = k.hedef, w = k.agirlik;
16      if (!ziyaret[v]
17        && uzaklik[u] + w < uzaklik[v]) {
18        uzaklik[v] = uzaklik[u] + w;
19        onceki[v] = u;
20        kuyruk.add(new int[] { v, uzaklik[v] });
21      }
22    }
23  }
24 }
```



Kuyruk boş: döngü biter. uzaklik = [6, 3, 3, 2, 0], onceki = [2, 4, 4, 5, -].

Adım 27 / 27

DEĞİŞKENLER

u = -

v = -

w = -

kuyruk = []

DIJKSTRA

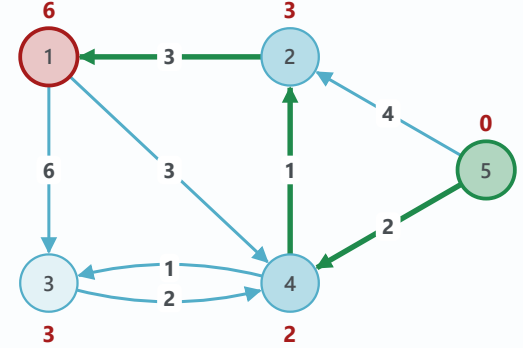
Yolu Geri Kurma: π (önceki) Dizisi

Örnek 2'nin sonucu: $\pi = [2, 4, 4, 5, /]$. Hedef 1 için geriye doğru:

$1 \leftarrow \pi[1] = 2 \leftarrow \pi[2] = 4 \leftarrow \pi[4] = 5$ (kaynak)

Ters çevrilince yol: $5 \rightarrow 4 \rightarrow 2 \rightarrow 1$, uzunluk $2 + 1 + 3 = 6 = d[1]$.

```
void yolYaz(int[] oncesi, int kaynak, int hedef) {  
    Deque<Integer> yigin = new ArrayDeque<>();  
    for (int v = hedef; v != kaynak; v = oncesi[v])  
        yigin.push(v);           // hedeften geriye  
    yigin.push(kaynak);  
    while (!yigin.isEmpty())  
        System.out.print(yigin.pop() + " ");  
}
```



Yığın (stack) geriye doğru toplanan düğümleri doğru sıraya çevirir.

Dijkstra: Algoritma Karmaşıklığı

Maliyet, **en küçük uzaklıklı düğümü seçme** işleminin nasıl yapıldığına bağlıdır:

Seçim yapısı	Seçim	Gevşetme	Toplam	Uygun çizge
Dizi (doğrusal tarama)	$V \times O(V)$	$E \times O(1)$	$O(V^2)$	yoğun ($E \approx V^2$)
İkili heap (öncelikli kuyruk)	$O(\log V)$	$O(\log V)$	$O((V + E) \log V)$	seyrek ($E \ll V^2$)
Fibonacci heap	$O(\log V)$ amortize	$O(1)$ amortize	$O(E + V \log V)$	kuramsal en iyi

Sayısal örnek

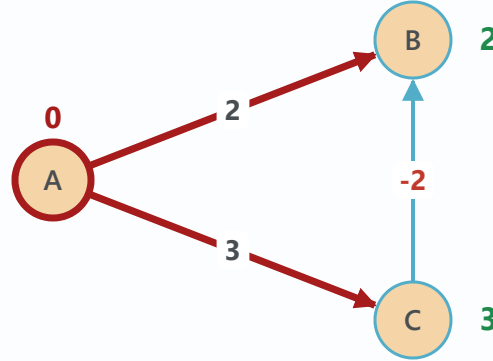
$V = 10^4$, $E = 5 \cdot 10^4$ (yol ağı): dizi $\approx 10^8$ işlem,
heap $\approx 6 \cdot 10^4 \cdot 14 \approx 8,4 \cdot 10^5$ işlem.

Bellek

d, π ve ziyaret dizileri $O(V)$; tembel silmeli
kuyrukta en fazla E kayıt $\rightarrow O(V + E)$.

Dijkstra Neden Negatif Kenarda Yanılır?

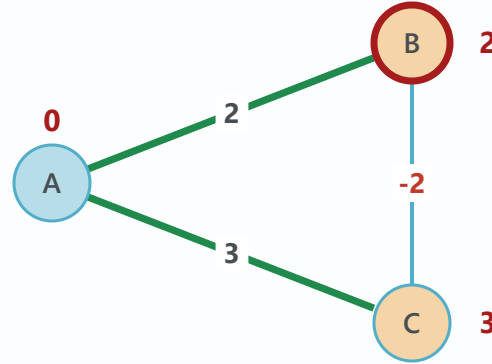
- Doğruluk kanıtı şuna dayanır: kesinleşen u için, S dışından geçen her yol en az $d[u]$ uzunluğundadır; çünkü kenarlar **negatif değildir** ve yol yalnızca uzayabilir.
- Negatif kenar bu varsayımı bozar: sonradan işlenen bir düğüm, kesinleşmiş bir düğüme daha kısa yol açabilir.
- Tüm ağırlıklara sabit eklemek de **çözüm değildir**: çok kenarlı yollar daha fazla ceza alır, en kısa yol değişir.
- Negatif kenar varsa **Bellman-Ford** kullanılır.



A kesinleşir; gevşetme ile $d[B] = 2$, $d[C] = 3$.

Dijkstra Neden Negatif Kenarda Yanılır?

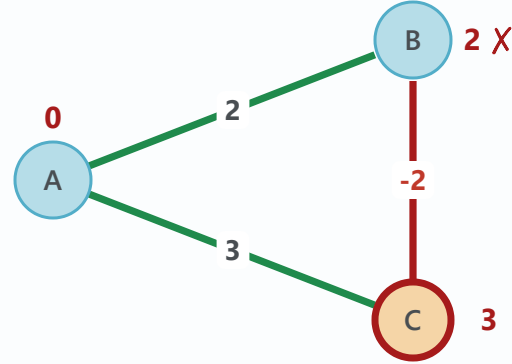
- Doğruluk kanıtı şuna dayanır: kesinleşen u için, S dışından geçen her yol en az $d[u]$ uzunluğundadır; çünkü kenarlar **negatif değildir** ve yol yalnızca uzayabilir.
- Negatif kenar bu varsayımı bozar: sonradan işlenen bir düğüm, kesinleşmiş bir düğüme daha kısa yol açabilir.
- Tüm ağırlıklara sabit eklemek de **çözüm değildir**: çok kenarlı yollar daha fazla ceza alır, en kısa yol değişir.
- Negatif kenar varsa **Bellman-Ford** kullanılır.



En küçük d : B (2). Dijkstra B'yi **kesinleştirir**:
"B'ye daha kısa yol olamaz" varsayımı.

Dijkstra Neden Negatif Kenarda Yanılır?

- Doğruluk kanıtı şuna dayanır: kesinleşen u için, S dışından geçen her yol en az $d[u]$ uzunluğundadır; çünkü kenarlar **negatif değildir** ve yol yalnızca uzayabilir.
- Negatif kenar bu varsayımı bozar: sonradan işlenen bir düğüm, kesinleşmiş bir düğüme daha kısa yol açabilir.
- Tüm ağırlıklara sabit eklemek de **çözüm değildir**: çok kenarlı yollar daha fazla ceza alır, en kısa yol değişir.
- Negatif kenar varsa **Bellman-Ford** kullanılır.



C işlenir: $d[C] + (-2) = 1 < 2$, ama B zaten S 'de. Sonuç $d[B] = 2$ **yanlış**; gerçek $\delta(A, B) = 1$ ($A \rightarrow C \rightarrow B$).

Bellman-Ford Algoritması

- **Kaynak** düğümden diğer tüm düğümlere olan en kısa yolları bulur (tek kaynaklı).
- Richard **Bellman** (1958) ve Lester **Ford** Jr. (1956) adlarını taşır; E. F. Moore da 1959'da aynı yöntemi verdi.
- **Negatif ağırlıklı kenarları** işleyebilir.
- Kaynaktan ulaşılabilen **negatif döngüleri** tespit edebilir.
- Dijkstra'dan yavaştır: $O(V \cdot E)$.

Algoritma ilkeleri

Her düğüm için en kısa yol **tahminlerini** tutan bir dizi kullanır. Başta tüm tahminler ∞ , kaynak 0'dır. Çizgedeki **tüm kenarlar** teker teker incelenir ve tahminler güncellenir; bu tarama $V - 1$ kez tekrarlanır. Düğüm seçimi yoktur, açgözlü değildir.

Bellman-Ford: Algoritma Adımları

Adım 1: Başlangıç düğümüne uzaklık **0**, diğer düğümlere ∞ atanır.

$\pi[v]$ = tanımsız.

Adım 1 / 4

Bellman-Ford: Algoritma Adımları

Adım 1: Başlangıç düğümüne uzaklık **0**, diğer düğümlere ∞ atanır.

$\pi[v]$ = tanımsız.

Adım 2: Kenarlar **tek tek** incelenir, düğümler arası uzaklıklar gevşetme ile güncellenir; bu tur **V** – **1** kez tekrarlanır.

Bir turda kenar sırası sabittir; sıra sonucu değil, kaç turda yakınsayacağını etkiler.

Bellman-Ford: Algoritma Adımları

Adım 1: Başlangıç düğümüne uzaklık **0**, diğer düğümlere ∞ atanır.

$\pi[v]$ = tanımsız.

Adım 2: Kenarlar **tek tek** incelenir, düğümler arası uzaklıklar gevşetme ile güncellenir; bu tur **V** – **1** kez tekrarlanır.

Bir turda kenar sırası sabittir; sıra sonucu değil, kaç turda yakınsayacağını etkiler.

Adım 3: Bir düğümün uzaklığı güncellenirse, sonraki incelemelerde onun **komşularının** uzaklıkları da güncellenebilir.

İyileşme, yol boyunca her turda en az bir kenar ilerler.

Bellman-Ford: Algoritma Adımları

Adım 1: Başlangıç düğümüne uzaklık **0**, diğer düğümlere ∞ atanır.

$\pi[v]$ = tanımsız.

Adım 2: Kenarlar **tek tek** incelenir, düğümler arası uzaklıklar gevşetme ile güncellenir; bu tur **V – 1** kez tekrarlanır.

Bir turda kenar sırası sabittir; sıra sonucu değil, kaç turda yakınsayacağını etkiler.

Adım 3: Bir düğümün uzaklığı güncellenirse, sonraki incelemelerde onun **komşularının** uzaklıkları da güncellenebilir.

İyileşme, yol boyunca her turda en az bir kenar ilerler.

Adım 4: Negatif döngü kontrolü için tüm kenarlar **bir kez daha** incelenir; hâlâ gevşeyen kenar varsa negatif döngü vardır.

Döngü yoksa $V - 1$ turdan sonra hiçbir kenar gevşemez.

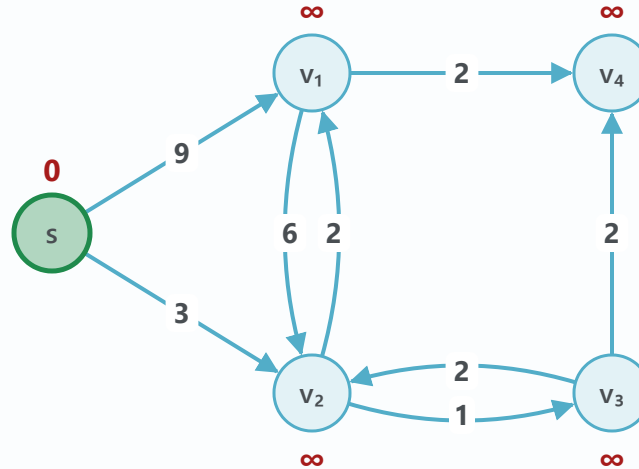
Neden $V - 1$ Tur? Karmaşıklık

- Negatif döngü yoksa her en kısa yol **en fazla $V - 1$ kenarlıdır**.
- **Değişmez:** i . turun sonunda, en fazla i kenarlı en kısa yolu olan her v için $d[v] = \delta(s, v)$.
- Tur i 'de yolun i . kenarı mutlaka gevşetilir; $V - 1$ tur tüm yolları tamamlar.
- Her tur E kenarı inceler: toplam **$O(V \cdot E)$** zaman, $O(V)$ ek bellek.

Çizge	Zaman
Seyrek, $E \approx V$	$O(V^2)$
Yoğun, $E \approx V^2$	$O(V^3)$
En iyi durum (erken durma)	$O(E)$

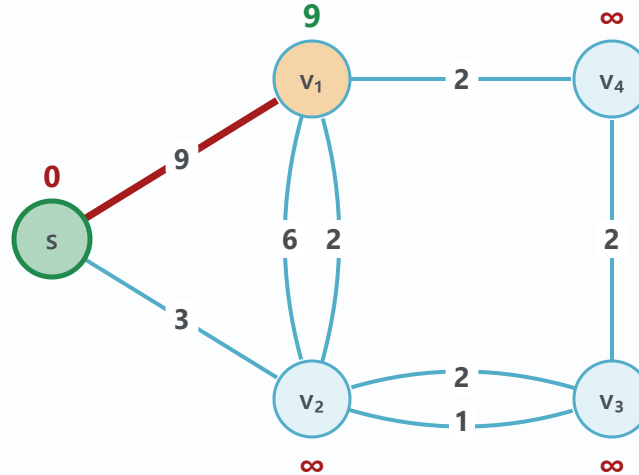
Erken durma: bir turda hiçbir d değişmezse sonraki turlar da değiştirmez; döngüden çıkılabilir.

Bellman-Ford Örnek 1: Kenar Kenar Gevşetme

(s, v₁) 9(s, v₂) 3(v₁, v₂) 6(v₁, v₄) 2(v₂, v₁) 2(v₂, v₃) 1(v₃, v₂) 2(v₃, v₄) 2

İlkendirme: $d[s] = 0$, diğerleri ∞ . Kenarlar her turda yukarıdaki sırayla incelenir.

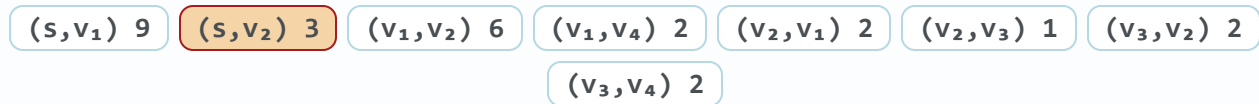
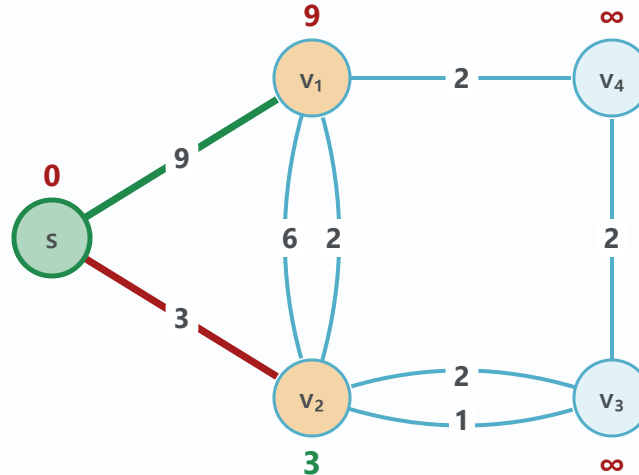
Bellman-Ford Örnek 1: Kenar Kenar Gevşetme



(s, v_1) 9	(s, v_2) 3	(v_1, v_2) 6	(v_1, v_4) 2	(v_2, v_1) 2	(v_2, v_3) 1	(v_3, v_2) 2
			(v_3, v_4) 2			

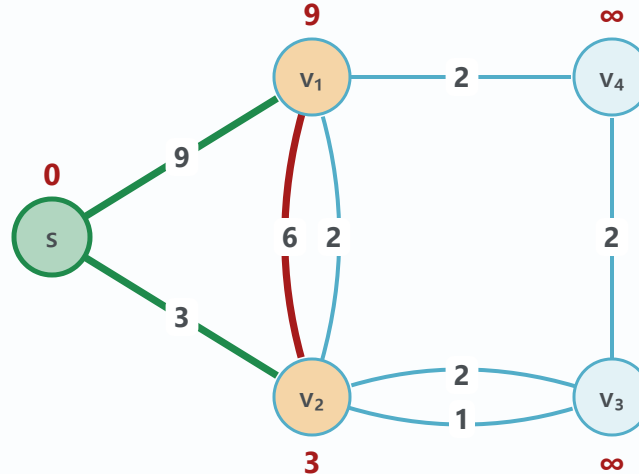
Tur 1, kenar (s, v_1) : $0 + 9 = 9 < \infty \rightarrow d[v_1] = 9$.

Bellman-Ford Örnek 1: Kenar Kenar Gevşetme



Tur 1, kenar (s, v_2) : $0 + 3 = 3 < \infty \rightarrow d[v_2] = 3$.

Bellman-Ford Örnek 1: Kenar Kenar Gevşetme



(s, v_1) 9

(s, v_2) 3

(v_1, v_2) 6

(v_1, v_4) 2

(v_2, v_1) 2

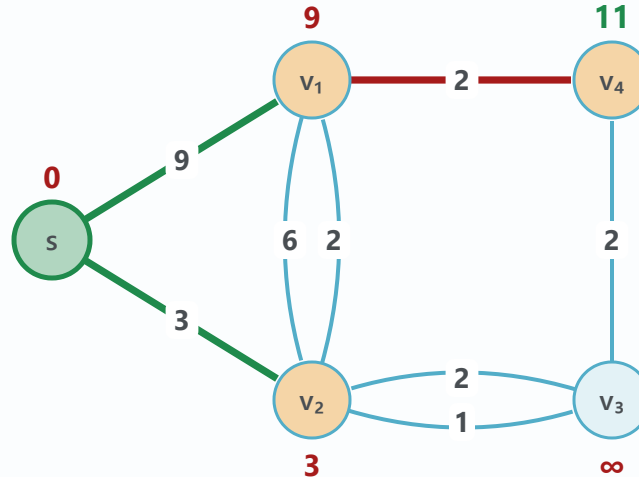
(v_2, v_3) 1

(v_3, v_2) 2

(v_3, v_4) 2

Tur 1, kenar (v_1, v_2) : $9 + 6 = 15 \geq 3 \rightarrow$ değişmez.

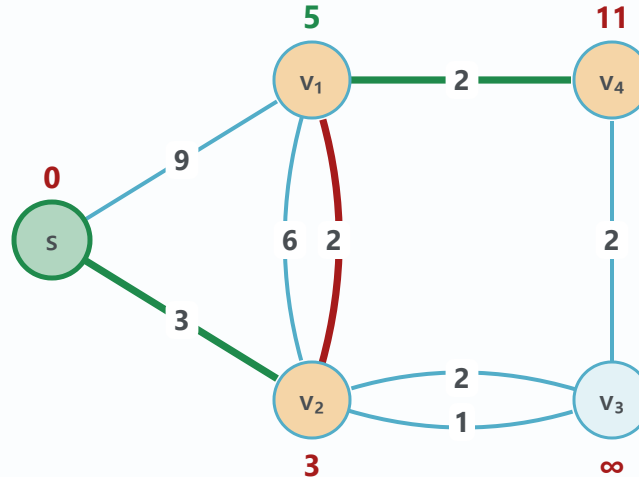
Bellman-Ford Örnek 1: Kenar Kenar Gevşetme



(s, v_1) 9	(s, v_2) 3	(v_1, v_2) 6	(v_1, v_4) 2	(v_2, v_1) 2	(v_2, v_3) 1	(v_3, v_2) 2
			(v_3, v_4) 2			

Tur 1, kenar (v_1, v_4) : $9 + 2 = 11 < \infty \rightarrow \mathbf{d[v_4] = 11}$.

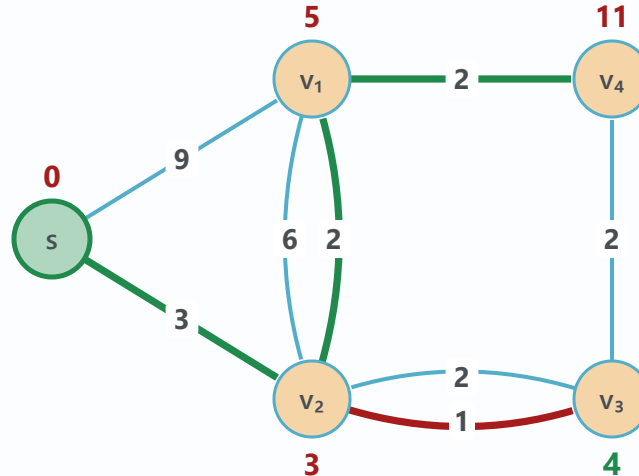
Bellman-Ford Örnek 1: Kenar Kenar Gevşetme



(s, v_1) 9 (s, v_2) 3 (v_1, v_2) 6 (v_1, v_4) 2 **(v_2, v_1) 2** (v_2, v_3) 1 (v_3, v_2) 2
 (v_3, v_4) 2

Tur 1, kenar (v_2, v_1) : $3 + 2 = 5 < 9 \rightarrow d[v_1] = 5$.

Bellman-Ford Örnek 1: Kenar Kenar Gevşetme



(s, v₁) 9

(s, v₂) 3

(v₁, v₂) 6

(v₁, v₄) 2

(v₂, v₁) 2

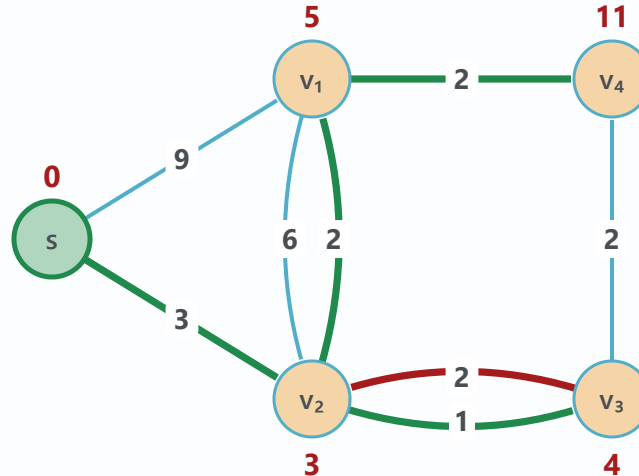
(v₂, v₃) 1

(v₃, v₂) 2

(v₃, v₄) 2

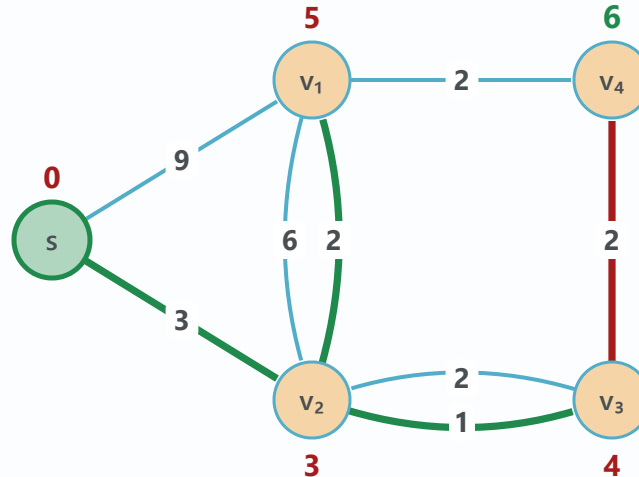
Tur 1, kenar (v₂, v₃): $3 + 1 = 4 < \infty \rightarrow \mathbf{d[v_3] = 4}$.

Bellman-Ford Örnek 1: Kenar Kenar Gevşetme

(s, v₁) 9(s, v₂) 3(v₁, v₂) 6(v₁, v₄) 2(v₂, v₁) 2(v₂, v₃) 1(v₃, v₂) 2(v₃, v₄) 2

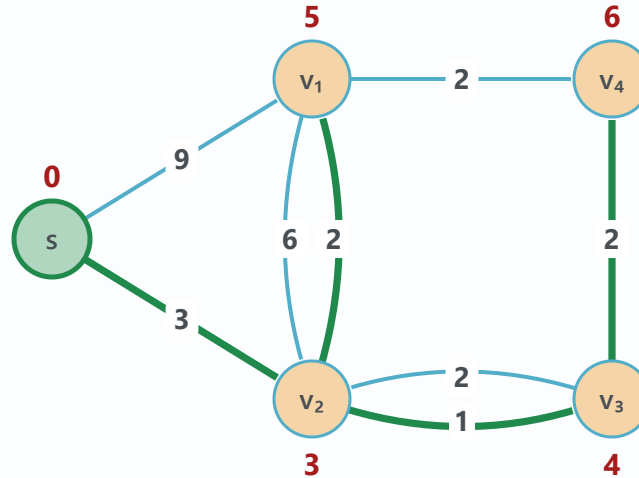
Tur 1, kenar (v₃, v₂): $4 + 2 = 6 \geq 3 \rightarrow$ değişmez.

Bellman-Ford Örnek 1: Kenar Kenar Gevşetme

(s, v₁) 9(s, v₂) 3(v₁, v₂) 6(v₁, v₄) 2(v₂, v₁) 2(v₂, v₃) 1(v₃, v₂) 2(v₃, v₄) 2

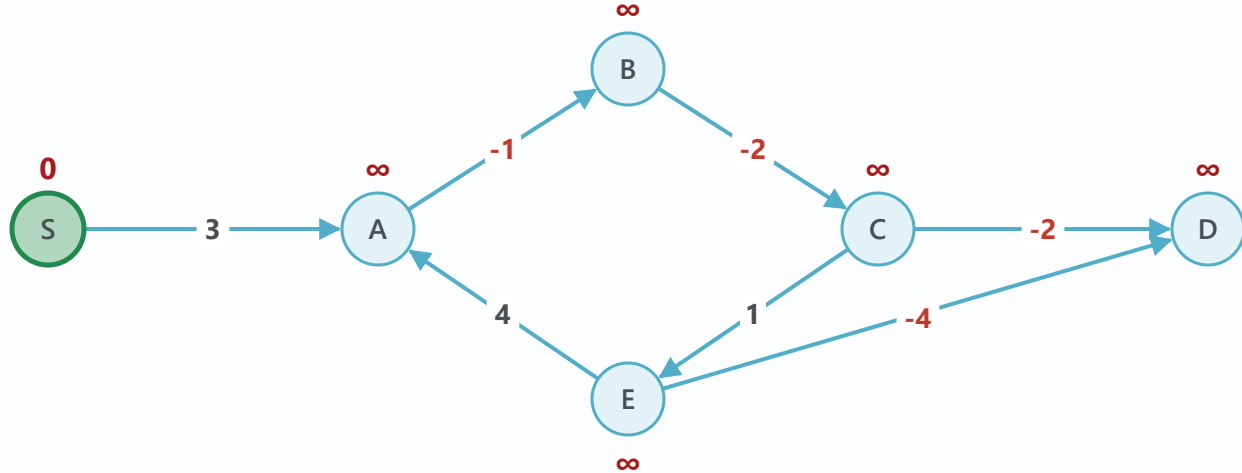
Tur 1, kenar (v₃, v₄): $4 + 2 = 6 < 11 \rightarrow \mathbf{d[v_4] = 6}$.

Bellman-Ford Örnek 1: Kenar Kenar Gevşetme

(s, v₁) 9(s, v₂) 3(v₁, v₂) 6(v₁, v₄) 2(v₂, v₁) 2(v₂, v₃) 1(v₃, v₂) 2(v₃, v₄) 2

Tur 2: sekiz kenarın hiçbiri gevşemez → **erken durma**. Sonuç: $v_1 = 5$, $v_2 = 3$, $v_3 = 4$, $v_4 = 6$ (bu sırada tek tur yetti).

Bellman-Ford Örnek 2: Negatif Kenarlar



(E,D) -4

(E,A) 4

(C,D) -2

(C,E) 1

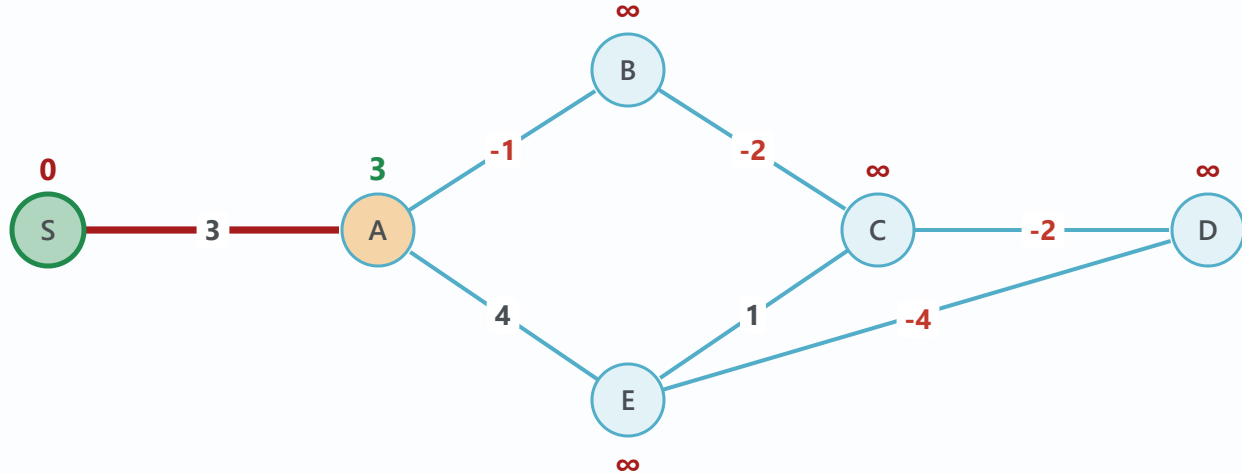
(B,C) -2

(A,B) -1

(S,A) 3

Negatif kenarlı çizge, kaynak S. Kenarlar her turda **yukarıdaki sırayla** incelenir (sondan başa: en kötü sıra).

Bellman-Ford Örnek 2: Negatif Kenarlar



(E,D) -4

(E,A) 4

(C,D) -2

(C,E) 1

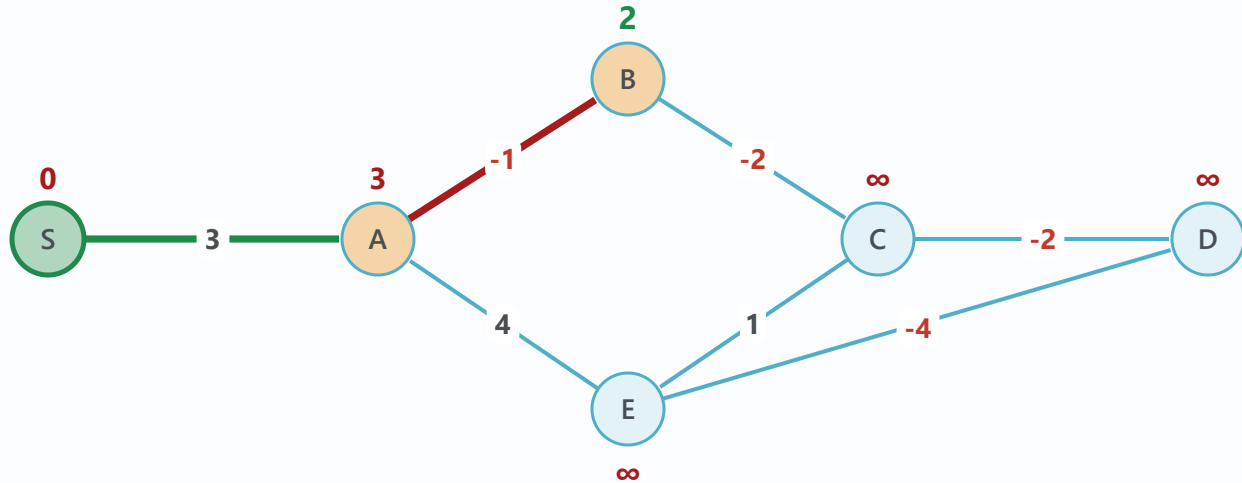
(B,C) -2

(A,B) -1

(S,A) 3

Tur 1: yalnızca (S,A): $\infty \rightarrow 3$. Diğer kenarların başı hâlâ ∞ .

Bellman-Ford Örnek 2: Negatif Kenarlar



(E,D) -4

(E,A) 4

(C,D) -2

(C,E) 1

(B,C) -2

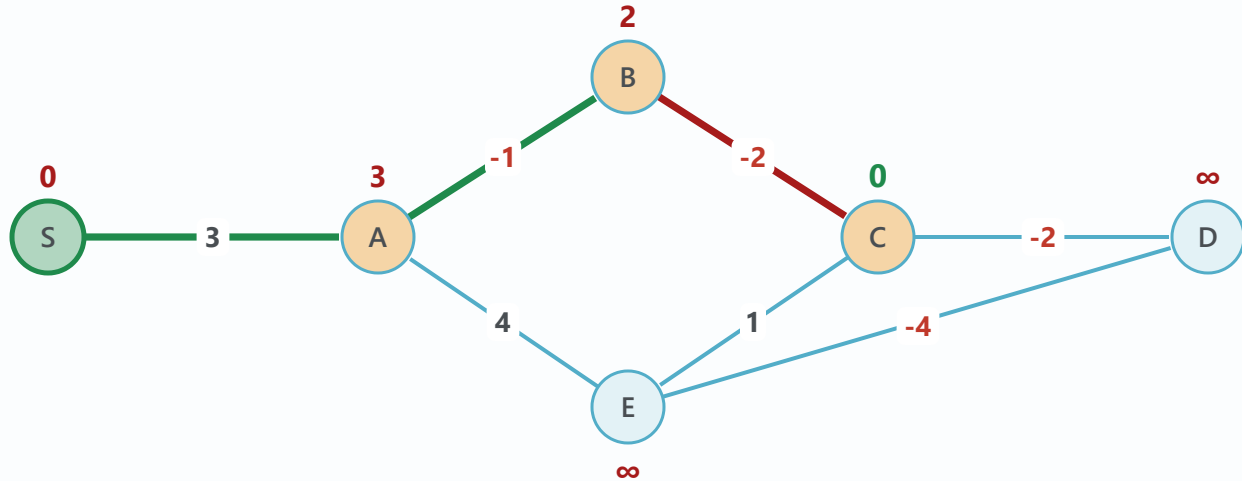
(A,B) -1

(S,A) 3

Tur 2: yalnızca (A,B): $\infty \rightarrow 2$.

Adım 3 / 7

Bellman-Ford Örnek 2: Negatif Kenarlar



(E,D) -4

(E,A) 4

(C,D) -2

(C,E) 1

(B,C) -2

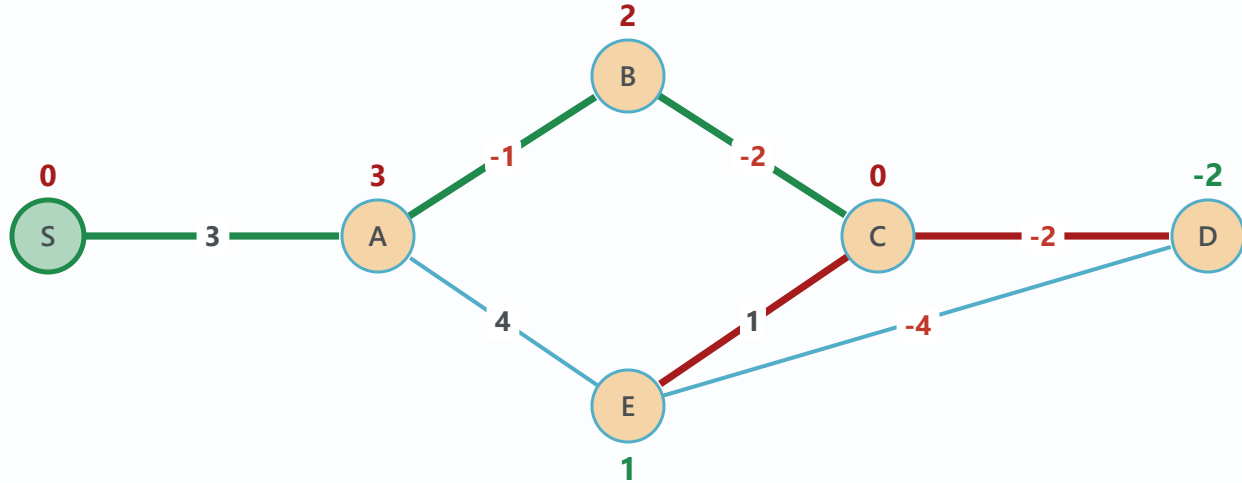
(A,B) -1

(S,A) 3

Tur 3: yalnızca (B,C): $\infty \rightarrow 0$.

Adım 4 / 7

Bellman-Ford Örnek 2: Negatif Kenarlar



(E,D) -4

(E,A) 4

(C,D) -2

(C,E) 1

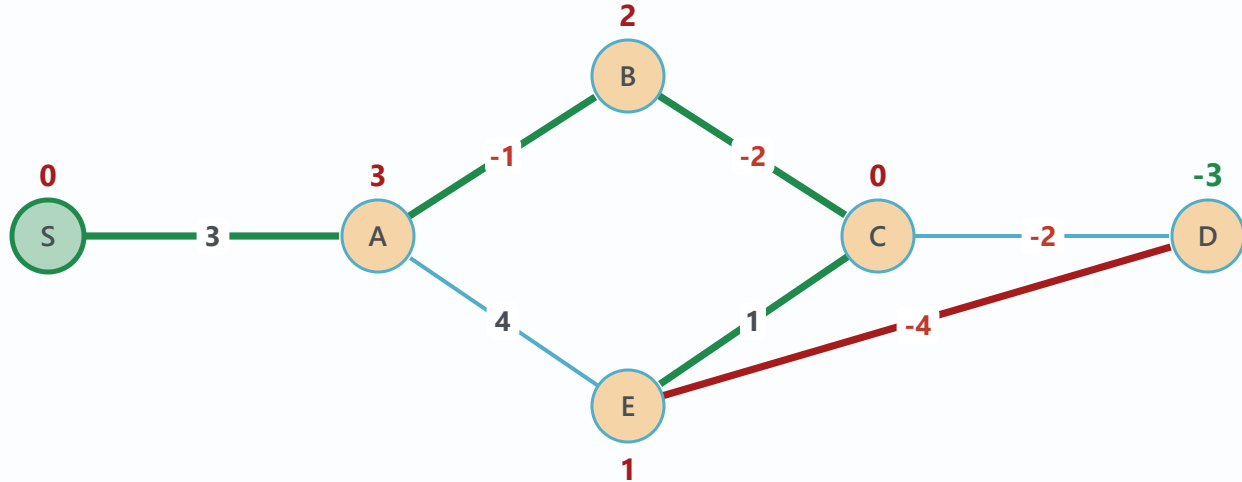
(B,C) -2

(A,B) -1

(S,A) 3

Tur 4: yalnızca (C,D): $\infty \rightarrow -2$; (C,E): $\infty \rightarrow 1$.

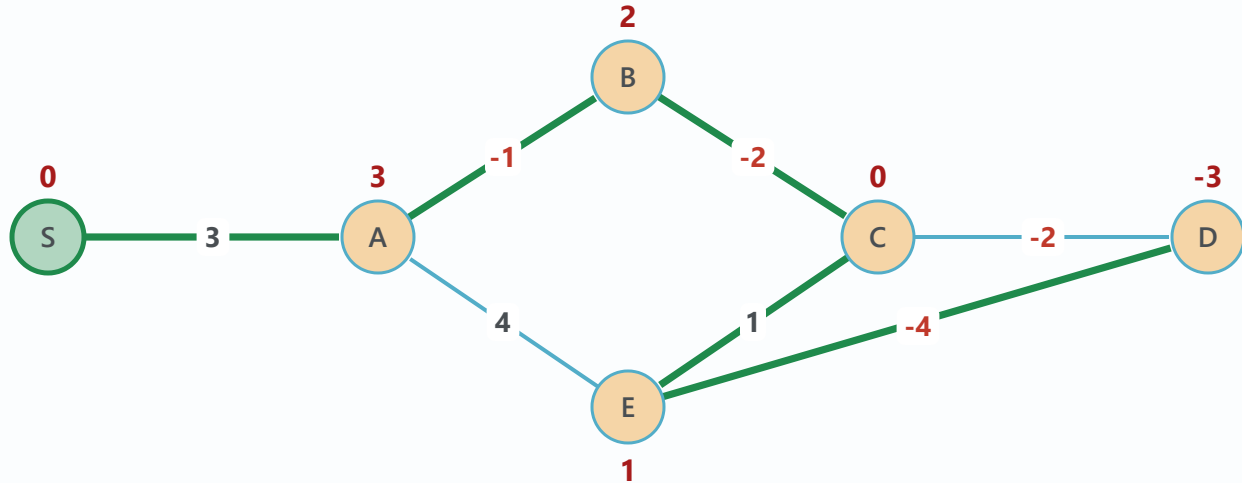
Bellman-Ford Örnek 2: Negatif Kenarlar



(E,D) -4 (E,A) 4 (C,D) -2 (C,E) 1 (B,C) -2 (A,B) -1 (S,A) 3

Tur 5: yalnızca (E,D): -2 → **-3**. E üzerinden D'ye daha kısa yol: $1 + (-4) = -3$.

Bellman-Ford Örnek 2: Negatif Kenarlar



(E,D) -4

(E,A) 4

(C,D) -2

(C,E) 1

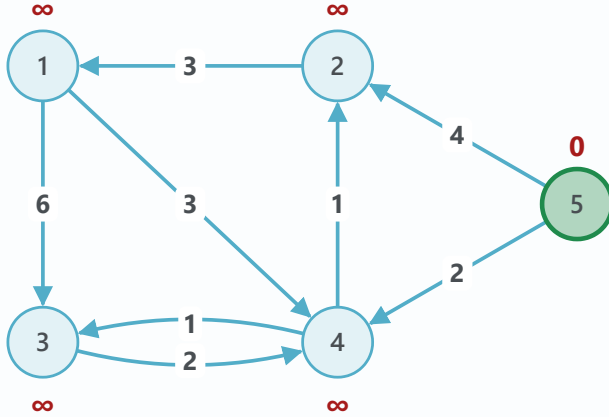
(B,C) -2

(A,B) -1

(S,A) 3

Denetim turu: hiçbir kenar gevşemez → negatif döngü yok ($A \rightarrow B \rightarrow C \rightarrow E \rightarrow A = -1 -2 + 1 + 4 = +2$). $V - 1 = 5$ turun hepsi gerekti.

Bellman-Ford Örnek 3: Dijkstra Örnek 2'deki Çizge

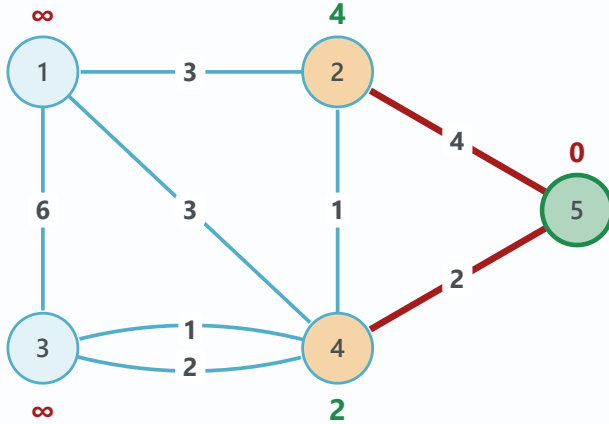


	1	2	3	4	5
d	∞	∞	∞	∞	0
π	/	/	/	/	/

(1,3) 6	(1,4) 3	(2,1) 3	(3,4) 2
(4,2) 1	(4,3) 1	(5,2) 4	(5,4) 2

İlkendirme: kaynak 5'in uzaklığı 0, diğerleri ∞ . Kenar sırası: (1,3), (1,4), (2,1), (3,4), (4,2), (4,3), (5,2), (5,4).

Bellman-Ford Örnek 3: Dijkstra Örnek 2'deki Çizge

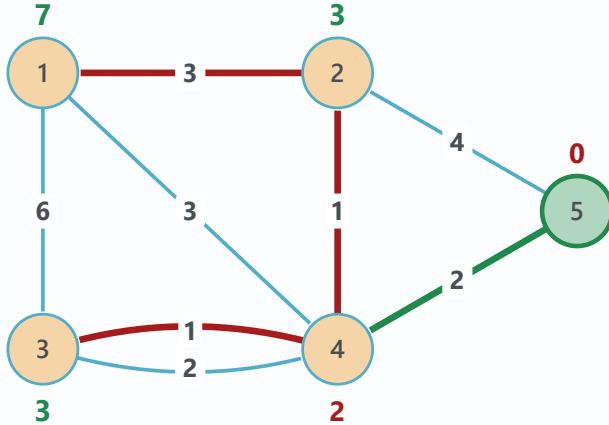


	1	2	3	4	5
d	∞	4	∞	2	0
π	/	5	/	5	/

(1,3) 6	(1,4) 3	(2,1) 3	(3,4) 2
(4,2) 1	(4,3) 1	(5,2) 4	(5,4) 2

Tur 1: (5,2): $\infty \rightarrow 4$; (5,4): $\infty \rightarrow 2$.

Bellman-Ford Örnek 3: Dijkstra Örnek 2'deki Çizge

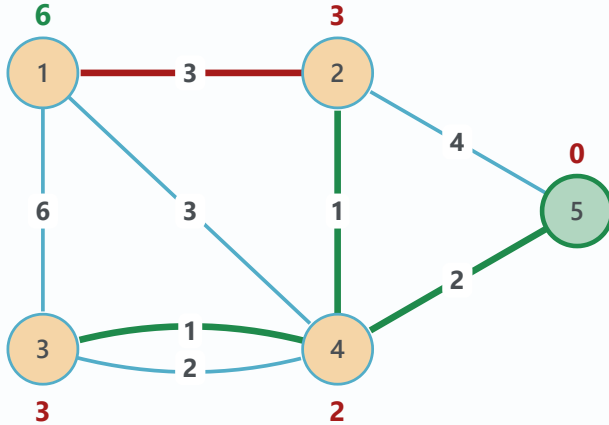


	1	2	3	4	5
d	7	3	3	2	0
π	2	4	4	5	/

(1,3) 6	(1,4) 3	(2,1) 3	(3,4) 2
(4,2) 1	(4,3) 1	(5,2) 4	(5,4) 2

Tur 2: (2,1): $\infty \rightarrow 7$; (4,2): $4 \rightarrow 3$; (4,3): $\infty \rightarrow 3$. (4,2) kenarı 2'ye daha kısa yol buldu.

Bellman-Ford Örnek 3: Dijkstra Örnek 2'deki Çizge

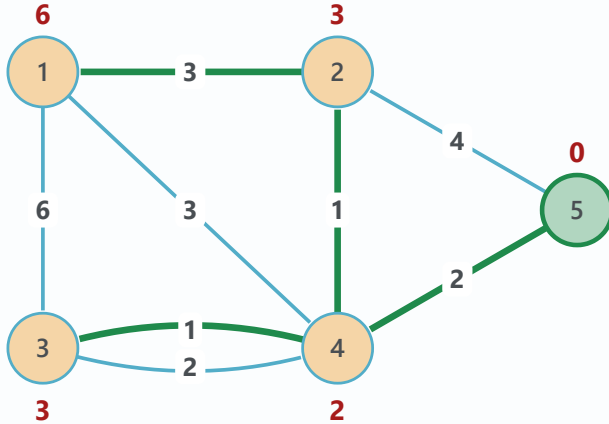


	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

(1,3) 6	(1,4) 3	(2,1) 3	(3,4) 2
(4,2) 1	(4,3) 1	(5,2) 4	(5,4) 2

Tur 3: (2,1): 7 $\rightarrow$ 6. 2 bir önceki turda iyileştiği için (2,1) yeniden gevşer.

Bellman-Ford Örnek 3: Dijkstra Örnek 2'deki Çizge



	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

(1,3) 6

(1,4) 3

(2,1) 3

(3,4) 2

(4,2) 1

(4,3) 1

(5,2) 4

(5,4) 2

Tur 4: daha kısa yol bulunamadı (hiçbir kenar gevşemez).

Adım 5 / 5

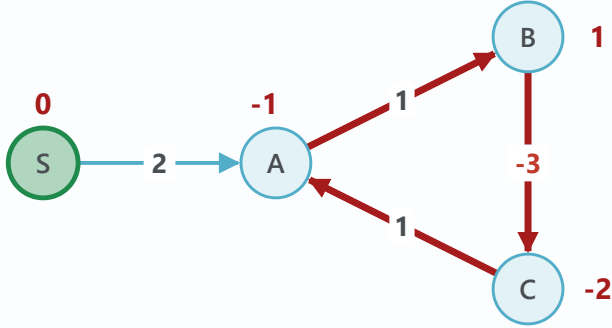
Negatif Döngü Kontrolü (V. Tur)

Her kenar için son bir kez gevşetme denenir: $d[v] > d[u] + w(u, v)$ olan bir kenar kalırsa negatif döngü vardır. Örnek 3'ün sonucunda:

kenar	$d[v]$	$d[u] + w(u, v)$	sonuç
(1, 3)	$d[3] = 3$	$d[1] + w = 6 + 6 = 12$	$3 > 12 \checkmark$
(1, 4)	$d[4] = 2$	$d[1] + w = 6 + 3 = 9$	$2 > 9 \checkmark$
(2, 1)	$d[1] = 6$	$d[2] + w = 3 + 3 = 6$	$6 > 6 \checkmark$
(3, 4)	$d[4] = 2$	$d[3] + w = 3 + 2 = 5$	$2 > 5 \checkmark$
(4, 2)	$d[2] = 3$	$d[4] + w = 2 + 1 = 3$	$3 > 3 \checkmark$
(4, 3)	$d[3] = 3$	$d[4] + w = 2 + 1 = 3$	$3 > 3 \checkmark$
(5, 2)	$d[2] = 3$	$d[5] + w = 0 + 4 = 4$	$3 > 4 \checkmark$
(5, 4)	$d[4] = 2$	$d[5] + w = 0 + 2 = 2$	$2 > 2 \checkmark$

Hiçbir kenar gevşemiyor $\rightarrow$ **negatif döngü yok**; $d = [6, 3, 3, 2, 0]$ kesindir.

Negatif Döngü Tespiti: Örnek



$A \rightarrow B \rightarrow C \rightarrow A$ döngüsü: $1 + (-3) + 1 = -1$.

Kenar sırası: (S,A), (A,B), (B,C), (C,A).

tur	S	A	B	C
1	0	1	3	0
2	0	0	2	-1
3	0	-1	1	-2

- $V - 1 = 3$ tur sonunda $d = [0, -1, 1, -2]$; döngüdeki değerler her turda 1 azalıyor.
- Denetim: (A, B) için $d[A] + 1 = 0 < d[B] = 1 \rightarrow$ **negatif döngü var.**
- Döngüdeki ve döngüden ulaşılan düğümlerin en kısa yolu $-\infty$ 'dur (tanımsız).

BELLMAN-FORD

Bellman-Ford: Sözde Kod

```
BELLMAN_FORD(G, kaynak):  
    mesafeler = [ ]  
    her bir v için V içinde:  
        mesafeler[v] = sonsuz  
    mesafeler[kaynak] = 0  
    döngü i = 1 den |V| - 1:  
        her bir (u, v, ağırlık) için E içinde:  
            eğer mesafeler[u] + ağırlık < mesafeler[v]:  
                mesafeler[v] = mesafeler[u] + ağırlık  
    her bir (u, v, ağırlık) için E içinde:  
        eğer mesafeler[u] + ağırlık < mesafeler[v]:  
            hata "Negatif ağırlıklı döngü var"  
    döndür mesafeler
```

İki bölüm

Dış döngü $V - 1$ tur; her turda **tüm** kenarlar. Ardından tek bir denetim turu.

Kenar listesi

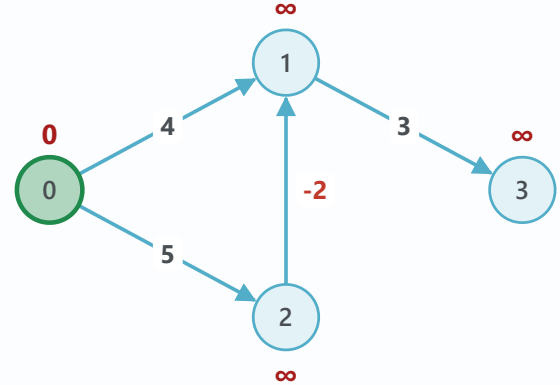
Bellman-Ford düğüm seçmediği için çizgeyi bir kenar listesi E olarak tutmak yeterlidir.

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             int u = k.bas, v = k.son, w = k.agirlik;
9             if (uzaklik[u] != SONSUZ
10                && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19            && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Kaynak 0. uzaklik = [0, ∞, ∞, ∞]. Kenar listesi: (2,1,-2), (0,1,4), (0,2,5), (1,3,3).

Adım 1 / 19

DEĞİŞKENLER

i = -

u = -

v = -

w = -

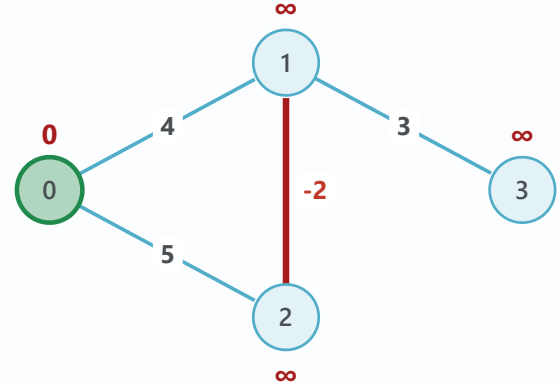
uzaklik = [0, ∞, ∞, ∞]

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2   int[] uzaklik = new int[V];
3   int[] onceki = new int[V];
4   Arrays.fill(uzaklik, SONSUZ);
5   uzaklik[kaynak] = 0;
6   for (int i = 1; i < V; i++) {
7     for (Kenar k : kenarlar) {
8       ▶ int u = k.bas, v = k.son, w = k.agirlik;
9       ▶ if (uzaklik[u] != SONSUZ
10        ▶   && uzaklik[u] + w < uzaklik[v]) {
11         uzaklik[v] = uzaklik[u] + w;
12         onceki[v] = u;
13       }
14     }
15   }
16   for (Kenar k : kenarlar) { // denetim
17     int u = k.bas, v = k.son, w = k.agirlik;
18     if (uzaklik[u] != SONSUZ
19       && uzaklik[u] + w < uzaklik[v])
20       throw new IllegalStateException(
21         "Negatif döngü");
22   }
23 }

```



Tur 1, kenar (2,1): $uzaklik[2] = \infty \rightarrow$ koşul yanlış, atlanır.

Adım 2 / 19

DEĞİŞKENLER

$i = 1$ $u = 2$ $v = 1$ $w = -2$

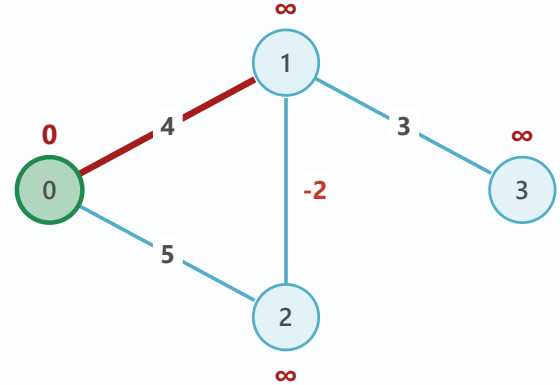
$uzaklik = [0, \infty, \infty, \infty]$

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             ▶ int u = k.bas, v = k.son, w = k.agirlik;
9             ▶ if (uzaklik[u] != SONSUZ
10                && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Tur 1, kenar (0,1): $0 + (4) = 4 < \infty \rightarrow$ koşul doğru.

Adım 3 / 19

DEĞİŞKENLER

$i = 1$

$u = 0$

$v = 1$

$w = 4$

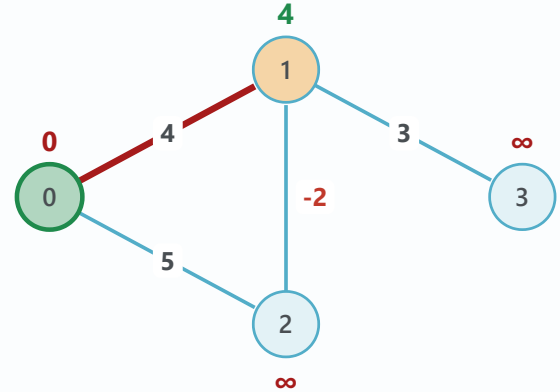
$uzaklik = [0, \infty, \infty, \infty]$

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             int u = k.bas, v = k.son, w = k.agirlik;
9             if (uzaklik[u] != SONSUZ
10                && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



uzaklik[1] = 4, onceki[1] = 0.

Adım 4 / 19

DEĞİŞKENLER

i = 1

u = 0

v = 1

w = 4

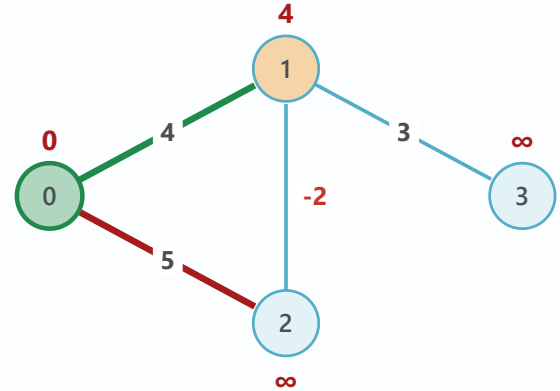
uzaklik = [0, 4, ∞, ∞]

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             ▶ int u = k.bas, v = k.son, w = k.agirlik;
9             ▶ if (uzaklik[u] != SONSUZ
10                ▶ && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Tur 1, kenar (0,2): $0 + (5) = 5 < \infty \rightarrow$ koşul doğru.

Adım 5 / 19

DEĞİŞKENLER

$i = 1$

$u = 0$

$v = 2$

$w = 5$

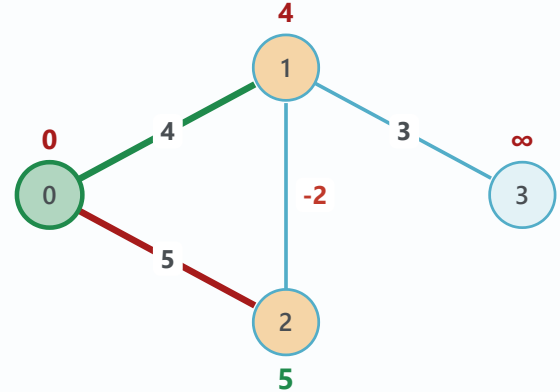
$uzaklik = [0, 4, \infty, \infty]$

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2   int[] uzaklik = new int[V];
3   int[] onceki = new int[V];
4   Arrays.fill(uzaklik, SONSUZ);
5   uzaklik[kaynak] = 0;
6   for (int i = 1; i < V; i++) {
7     for (Kenar k : kenarlar) {
8       int u = k.bas, v = k.son, w = k.agirlik;
9       if (uzaklik[u] != SONSUZ
10          && uzaklik[u] + w < uzaklik[v]) {
11         ▶ uzaklik[v] = uzaklik[u] + w;
12         ▶ onceki[v] = u;
13       }
14     }
15   }
16   for (Kenar k : kenarlar) { // denetim
17     int u = k.bas, v = k.son, w = k.agirlik;
18     if (uzaklik[u] != SONSUZ
19         && uzaklik[u] + w < uzaklik[v])
20       throw new IllegalStateException(
21         "Negatif döngü");
22   }
23 }

```



uzaklik[2] = 5, onceki[2] = 0.

Adım 6 / 19

DEĞİŞKENLER

i = 1

u = 0

v = 2

w = 5

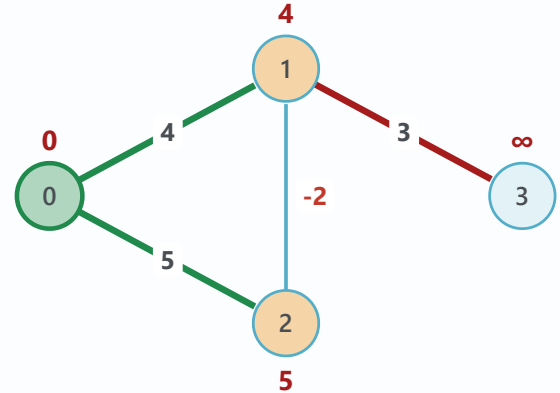
uzaklik = [0, 4, 5, ∞]

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             ▶ int u = k.bas, v = k.son, w = k.agirlik;
9             ▶ if (uzaklik[u] != SONSUZ
10                ▶ && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Tur 1, kenar (1,3): $4 + (3) = 7 < \infty \rightarrow$ koşul doğru.

Adım 7 / 19

DEĞİŞKENLER

i = 1

u = 1

v = 3

w = 3

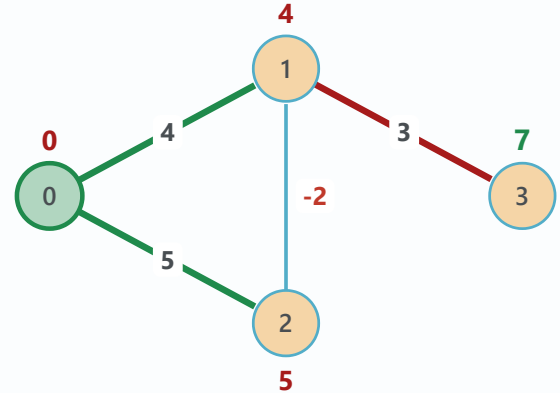
uzaklik = [0, 4, 5, ∞]

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             int u = k.bas, v = k.son, w = k.agirlik;
9             if (uzaklik[u] != SONSUZ
10                && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19            && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



uzaklik[3] = 7, onceki[3] = 1. Tur 1 bitti: (2,1) kenarı 2 henüz ∞ iken incelendiği için boş geçti.

Adım 8 / 19

DEĞİŞKENLER

i = 1

u = 1

v = 3

w = 3

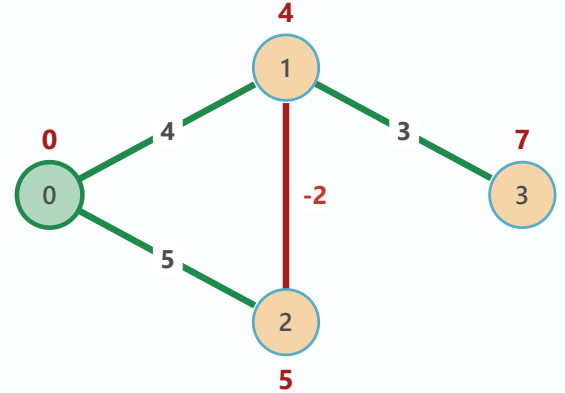
uzaklik = [0, 4, 5, 7]

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             ▶ int u = k.bas, v = k.son, w = k.agirlik;
9             ▶ if (uzaklik[u] != SONSUZ
10                ▶ && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Tur 2, kenar (2,1): $5 + (-2) = 3 < 4 \rightarrow$ koşul doğru.

Adım 9 / 19

DEĞİŞKENLER

$i = 2$

$u = 2$

$v = 1$

$w = -2$

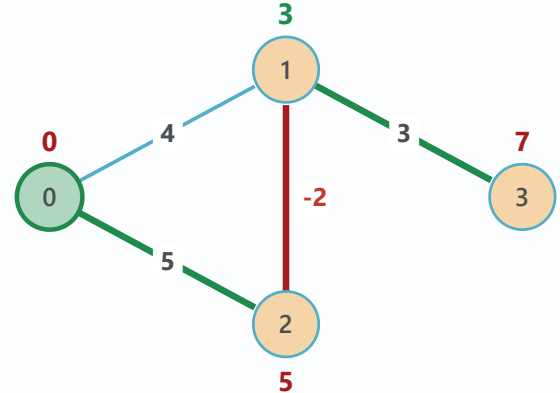
$uzaklik = [0, 4, 5, 7]$

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             int u = k.bas, v = k.son, w = k.agirlik;
9             if (uzaklik[u] != SONSUZ
10                && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



uzaklik[1] = 3, onceki[1] = 2.

Adım 10 / 19

DEĞİŞKENLER

i = 2

u = 2

v = 1

w = -2

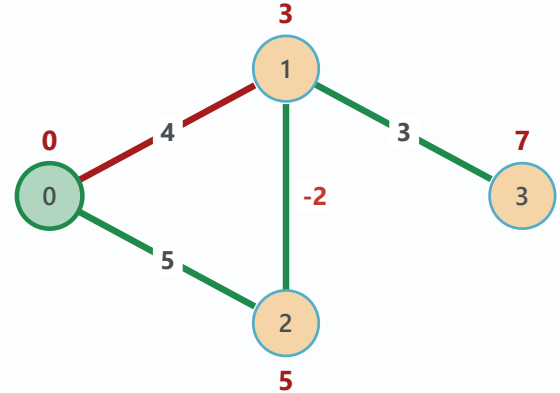
uzaklik = [0, 3, 5, 7]

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             ▶ int u = k.bas, v = k.son, w = k.agirlik;
9             ▶ if (uzaklik[u] != SONSUZ
10                ▶ && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Tur 2, kenar (0,1): $0 + (4) = 4 \geq 3 \rightarrow$ değişiklik yok.

Adım 11 / 19

DEĞİŞKENLER

$i = 2$

$u = 0$

$v = 1$

$w = 4$

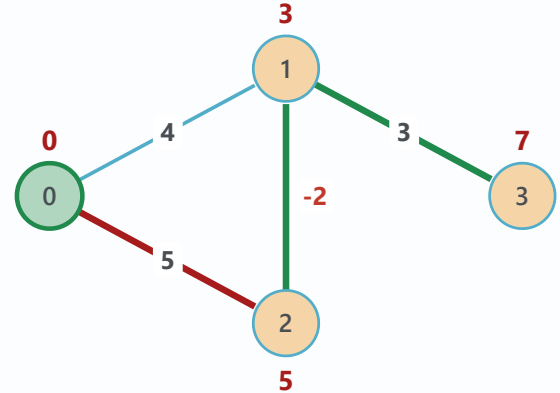
$uzaklik = [0, 3, 5, 7]$

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             ▶ int u = k.bas, v = k.son, w = k.agirlik;
9             ▶ if (uzaklik[u] != SONSUZ
10                ▶ && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Tur 2, kenar (0,2): $0 + (5) = 5 \geq 5 \rightarrow$ değişiklik yok.

Adım 12 / 19

DEĞİŞKENLER

i = 2

u = 0

v = 2

w = 5

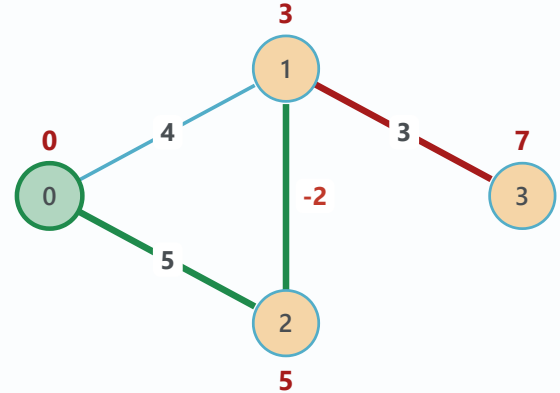
uzaklik = [0, 3, 5, 7]

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2   int[] uzaklik = new int[V];
3   int[] onceki = new int[V];
4   Arrays.fill(uzaklik, SONSUZ);
5   uzaklik[kaynak] = 0;
6   for (int i = 1; i < V; i++) {
7     for (Kenar k : kenarlar) {
8       ▶ int u = k.bas, v = k.son, w = k.agirlik;
9       ▶ if (uzaklik[u] != SONSUZ
10        ▶   && uzaklik[u] + w < uzaklik[v]) {
11         uzaklik[v] = uzaklik[u] + w;
12         onceki[v] = u;
13       }
14     }
15   }
16   for (Kenar k : kenarlar) { // denetim
17     int u = k.bas, v = k.son, w = k.agirlik;
18     if (uzaklik[u] != SONSUZ
19       && uzaklik[u] + w < uzaklik[v])
20       throw new IllegalStateException(
21         "Negatif döngü");
22   }
23 }

```



Tur 2, kenar (1,3): $3 + (3) = 6 < 7 \rightarrow$ koşul doğru.

Adım 13 / 19

DEĞİŞKENLER

$i = 2$

$u = 1$

$v = 3$

$w = 3$

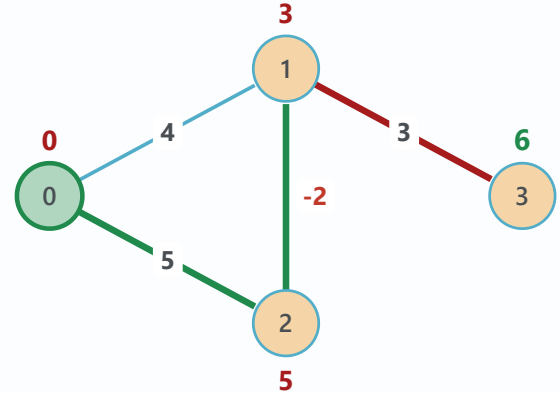
$uzaklik = [0, 3, 5, 7]$

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             int u = k.bas, v = k.son, w = k.agirlik;
9             if (uzaklik[u] != SONSUZ
10                && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



uzaklik[3] = 6, onceki[3] = 1. Tur 2 bitti:
negatif kenar 1'i 3'e, o da 3'ü 6'ya indirdi.

Adım 14 / 19

DEĞİŞKENLER

i = 2

u = 1

v = 3

w = 3

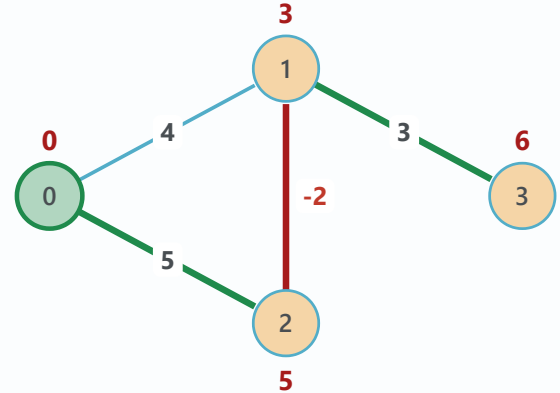
uzaklik = [0, 3, 5, 6]

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2   int[] uzaklik = new int[V];
3   int[] onceki = new int[V];
4   Arrays.fill(uzaklik, SONSUZ);
5   uzaklik[kaynak] = 0;
6   for (int i = 1; i < V; i++) {
7     for (Kenar k : kenarlar) {
8       ▶ int u = k.bas, v = k.son, w = k.agirlik;
9       ▶ if (uzaklik[u] != SONSUZ
10        ▶   && uzaklik[u] + w < uzaklik[v]) {
11         uzaklik[v] = uzaklik[u] + w;
12         onceki[v] = u;
13       }
14     }
15   }
16   for (Kenar k : kenarlar) { // denetim
17     int u = k.bas, v = k.son, w = k.agirlik;
18     if (uzaklik[u] != SONSUZ
19       && uzaklik[u] + w < uzaklik[v])
20       throw new IllegalStateException(
21         "Negatif döngü");
22   }
23 }

```



Tur 3, kenar (2,1): $5 + (-2) = 3 \geq 3 \rightarrow$ değişiklik yok.

Adım 15 / 19

DEĞİŞKENLER

$i = 3$ $u = 2$ $v = 1$ $w = -2$

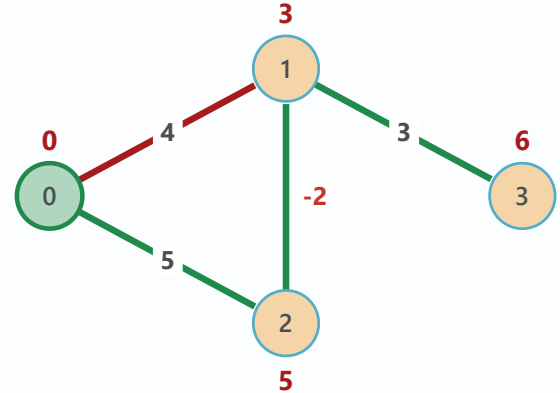
$uzaklik = [0, 3, 5, 6]$

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             ▶ int u = k.bas, v = k.son, w = k.agirlik;
9             ▶ if (uzaklik[u] != SONSUZ
10                ▶ && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Tur 3, kenar (0,1): $0 + (4) = 4 \geq 3 \rightarrow$ değişiklik yok.

Adım 16 / 19

DEĞİŞKENLER

$i = 3$

$u = 0$

$v = 1$

$w = 4$

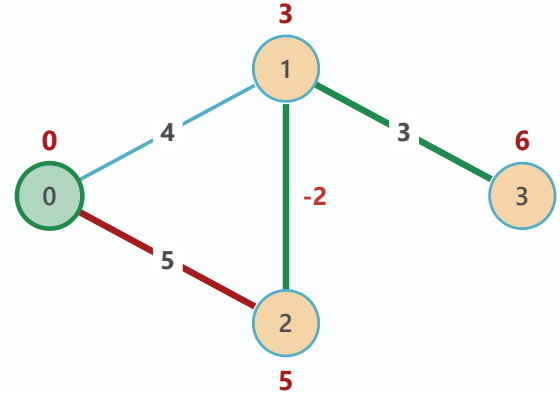
$uzaklik = [0, 3, 5, 6]$

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             ▶ int u = k.bas, v = k.son, w = k.agirlik;
9             ▶ if (uzaklik[u] != SONSUZ
10             ▶     && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Tur 3, kenar (0,2): $0 + (5) = 5 \geq 5 \rightarrow$ değişiklik yok.

Adım 17 / 19

DEĞİŞKENLER

i = 3

u = 0

v = 2

w = 5

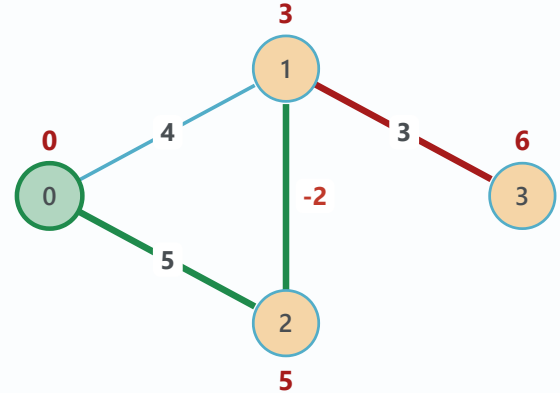
uzaklik = [0, 3, 5, 6]

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2     int[] uzaklik = new int[V];
3     int[] onceki = new int[V];
4     Arrays.fill(uzaklik, SONSUZ);
5     uzaklik[kaynak] = 0;
6     for (int i = 1; i < V; i++) {
7         for (Kenar k : kenarlar) {
8             ▶ int u = k.bas, v = k.son, w = k.agirlik;
9             ▶ if (uzaklik[u] != SONSUZ
10                ▶ && uzaklik[u] + w < uzaklik[v]) {
11                 uzaklik[v] = uzaklik[u] + w;
12                 onceki[v] = u;
13             }
14         }
15     }
16     for (Kenar k : kenarlar) { // denetim
17         int u = k.bas, v = k.son, w = k.agirlik;
18         if (uzaklik[u] != SONSUZ
19             && uzaklik[u] + w < uzaklik[v])
20             throw new IllegalStateException(
21                 "Negatif döngü");
22     }
23 }

```



Tur 3, kenar (1,3): $3 + (3) = 6 \geq 6 \rightarrow$ değişiklik yok.

Adım 18 / 19

DEĞİŞKENLER

$i = 3$

$u = 1$

$v = 3$

$w = 3$

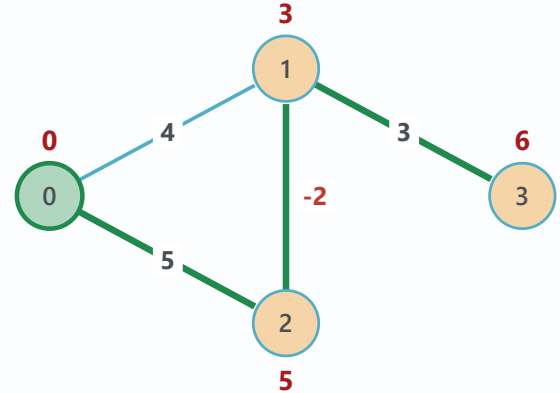
$uzaklik = [0, 3, 5, 6]$

Bellman-Ford: Kodu Adım Adım İzle

```

1 void bellmanFord(int kaynak) {
2   int[] uzaklik = new int[V];
3   int[] onceki = new int[V];
4   Arrays.fill(uzaklik, SONSUZ);
5   uzaklik[kaynak] = 0;
6   for (int i = 1; i < V; i++) {
7     for (Kenar k : kenarlar) {
8       int u = k.bas, v = k.son, w = k.agirlik;
9       if (uzaklik[u] != SONSUZ
10          && uzaklik[u] + w < uzaklik[v]) {
11         uzaklik[v] = uzaklik[u] + w;
12         onceki[v] = u;
13       }
14     }
15   }
16   for (Kenar k : kenarlar) { // denetim
17     int u = k.bas, v = k.son, w = k.agirlik;
18     if (uzaklik[u] != SONSUZ
19        && uzaklik[u] + w < uzaklik[v])
20       throw new IllegalStateException(
21         "Negatif döngü");
22   }
23 }

```



Tur 3 hiçbir şeyi değiştirmedir. Denetim turu (satır 16–20): hiçbir kenar gevşemez, istisna fırlatılmaz. Sonuç [0, 3, 5, 6].

Adım 19 / 19

DEĞİŞKENLER

i = -

u = 1

v = 3

w = 3

uzaklik = [0, 3, 5, 6]

Floyd-Warshall Algoritması

- **Tüm düğüm çiftleri** arasındaki en kısa yolları bulur (all-pairs).
- Bernard Roy (1959), Stephen **Warshall** (1962, geçişli kapanış) ve Robert **Floyd** (1962, en kısa yol) tarafından birbirinden bağımsız verilmiştir.
- **Negatif ağırlıklı kenarlarla** çalışır; negatif döngüyü **tespit eder** (o durumda en kısa yol tanımsızdır).
- **Dinamik programlama** kullanır; sonuçları bir $V \times V$ matriste (D) tutar.

Tek kaynak mı, tüm çiftler mi?

Bellman-Ford ve Dijkstra **tek kaynaktan** diğer düğümlere en kısa yolları bulur. Floyd-Warshall ise **tüm çiftler** arasındaki en kısa yolları tek seferde, üç iç içe döngüyle hesaplar. Kod çok kısadır; yoğun çizgelerde pratiktir.

Floyd-Warshall: Adımlar ve Özyineleme Bağıntısı

1. **Adım 1:** D matrisi kenar ağırlıklarıyla doldurulur:
 $D[i][i] = 0$; doğrudan kenar yoksa ∞ .
2. **Adım 2:** Ara düğüm $k = 1, 2, \dots, n$ sırayla ele alınır; her (i, j) çifti için...
3. **Adım 3:** k üzerinden geçen yolun uzunluğu $D[i][k] + D[k][j]$ hesaplanır ve mevcut $D[i][j]$ ile karşılaştırılır.
4. **Adım 4:** Daha kısaysa yeni değer matrise yazılır (Π matrisi de güncellenir).

Alt problem

$D^{(k)}[i][j]$: ara düğümleri yalnızca $\{1, \dots, k\}$ kümesinden seçilen en kısa $i \rightsquigarrow j$ yolunun uzunluğu.

$$D^{(k)}[i][j] = \min(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$

k ya yolda yoktur ya da yol k 'den bir kez geçer. $D^{(n)}$ sonuçtur.

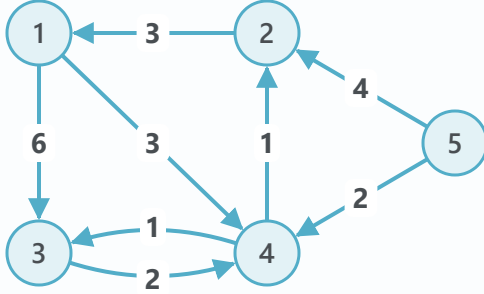
Floyd-Warshall: Karmaşıklık

- Zaman $\Theta(V^3)$: 3 iç içe for döngüsü, her biri V kez.
 - ilk (en dış) döngü tüm **ara** düğümleri (k) gezer,
 - ikinci döngü tüm **kaynak** düğümleri (i) gezer,
 - son döngü tüm **hedef** düğümleri (j) gezer.
- Bellek $\Theta(V^2)$: D yerinde (in-place) güncellenebilir; k. satır ve sütun k turunda değişmez.

Tüm çiftler için	Zaman
Floyd-Warshall	$O(V^3)$
$V \times$ Dijkstra (dizi)	$O(V^3)$
$V \times$ Dijkstra (heap)	$O(V(V + E) \log V)$
$V \times$ Bellman-Ford	$O(V^2 E)$

Döngü sırası önemlidir: **k en dışta** olmalıdır. k içeride olursa sonuç yanlış çıkar.

Floyd-Warshall Örneği: D ve Π Matrisleri



$$D^{(0)}$$

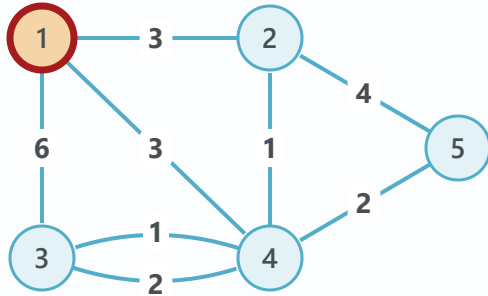
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	∞	∞	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

$$\Pi^{(0)}$$

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	/	/	/
3	/	/	/	3	/
4	/	4	4	/	/
5	/	5	/	5	/

$k = 0$ (ilklendirme): D kenar ağırlıklarıyla doldurulur, köşegen 0, kenar yoksa ∞ . $\Pi[i][j] = i$ (doğrudan kenar varsa), yoksa /.

Floyd-Warshall Örneği: D ve Π Matrisleri

 $D^{(1)}$

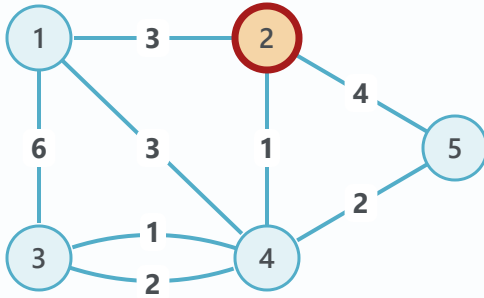
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

 $\Pi^{(1)}$

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	1	1	/
3	/	/	/	3	/
4	/	4	4	/	/
5	/	5	/	5	/

$k = 1$: düğüm 1 üzerinden $2 \rightsquigarrow 3$ ($3 + 6 = 9$) ve $2 \rightsquigarrow 4$ ($3 + 3 = 6$) yolları bulundu.

Floyd-Warshall Örneği: D ve Π Matrisleri



$$D^{(2)}$$

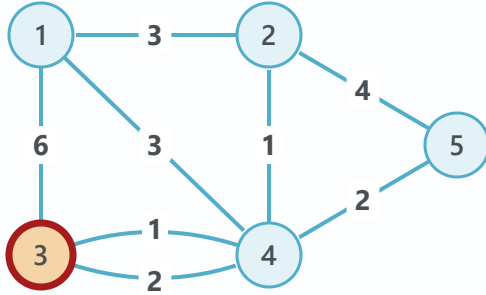
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	4	1	1	0	∞
5	7	4	13	2	0

$$\Pi^{(2)}$$

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	1	1	/
3	/	/	/	3	/
4	2	4	4	/	/
5	2	5	1	5	/

$k = 2$: düğüm 2 üzerinden $4 \rightsquigarrow 1$ ($1 + 3 = 4$), $5 \rightsquigarrow 1$ ($4 + 3 = 7$) ve $5 \rightsquigarrow 3$ ($4 + 9 = 13$) bulundu.

Floyd-Warshall Örneği: D ve Π Matrisleri

 $D^{(3)}$

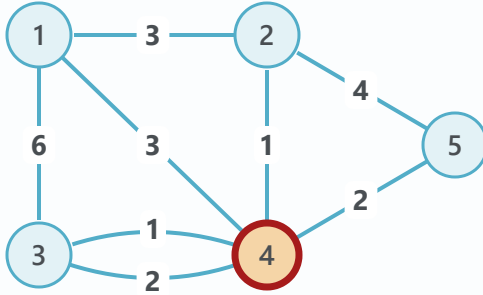
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	4	1	1	0	∞
5	7	4	13	2	0

 $\Pi^{(3)}$

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	1	1	/
3	/	/	/	3	/
4	2	4	4	/	/
5	2	5	1	5	/

$k = 3$: düğüm 3 üzerinden kısa yol bulunamadı (3'ten yalnızca 4'e kenar var, hiçbir değer iyileşmez).

Floyd-Warshall Örneği: D ve Π Matrisleri

 $D^{(4)}$

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

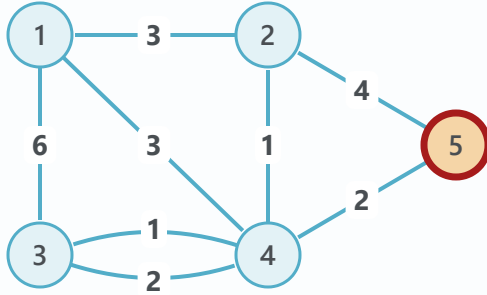
 $\Pi^{(4)}$

	1	2	3	4	5
1	/	4	4	1	/
2	2	/	4	1	/
3	2	4	/	3	/
4	2	4	4	/	/
5	2	4	4	5	/

$k = 4$: düğüm 4 üzerinden $1 \rightsquigarrow 2$, $1 \rightsquigarrow 3$, $2 \rightsquigarrow 3$, $3 \rightsquigarrow 1$, $3 \rightsquigarrow 2$, $5 \rightsquigarrow 1$, $5 \rightsquigarrow 2$, $5 \rightsquigarrow 3$ kısaltıldı (8 hücre). Ör. $5 \rightsquigarrow 3$: $2 + 1 = 3 < 13$.

Adım 5 / 6

Floyd-Warshall Örneği: D ve Π Matrisleri

 $D^{(5)}$

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

 $\Pi^{(5)}$

	1	2	3	4	5
1	/	4	4	1	/
2	2	/	4	1	/
3	2	4	/	3	/
4	2	4	4	/	/
5	2	4	4	5	/

$k = 5$: düğüm 5 üzerinden kısa yol yok (5'e gelen kenar olmadığından $D[i][5] = \infty$). Son matrisler hazır.

Son Durum ve Yolun Okunması

D (uzaklıklar)						Π (önceki)					
	1	2	3	4	5		1	2	3	4	5
1	0	4	4	3	∞	1	/	4	4	1	/
2	3	0	7	6	∞	2	2	/	4	1	/
3	6	3	0	2	∞	3	2	4	/	3	/
4	4	1	1	0	∞	4	2	4	4	/	/
5	6	3	3	2	0	5	2	4	4	5	/

- $\Pi[i][j]$: i'den j'ye en kısa yolda **j'den önceki** düğüm.
- $5 \rightsquigarrow 1$: $\Pi[5][1] = 2$, $\Pi[5][2] = 4$, $\Pi[5][4] = 5 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 1$, $D[5][1] = 6$.
- $1 \rightsquigarrow 2$: $\Pi[1][2] = 4$, $\Pi[1][4] = 1 \rightarrow 1 \rightarrow 4 \rightarrow 2$, $D[1][2] = 4$.
- 5. sütun ∞ : 5'e hiçbir düğümden ulaşamaz.
- 5. satır, Dijkstra ve Bellman-Ford'un kaynak 5 için bulduğu $[6, 3, 3, 2, 0]$ ile aynıdır.

Floyd-Warshall: Sözde Kod

1. İklendirme

```
FLOYD_WARSHALL(G):  
    mesafeler = [ ][ ]  
    her bir u için V içinde:  
        her bir v için V içinde:  
            eğer u == v:  
                mesafeler[u][v] = 0  
            yoksa eğer (u, v) E içinde:  
                mesafeler[u][v] = ağırlık(u, v)  
            yoksa:  
                mesafeler[u][v] = sonsuz
```

2. Üç iç içe döngü ve denetim (m = uzaklıklar)

```
her bir k için V içinde:  
    her bir i için V içinde:  
        her bir j için V içinde:  
            eğer  $m[i][k] + m[k][j] < m[i][j]$ :  
                 $m[i][j] = m[i][k] + m[k][j]$   
her bir i için V içinde:  
    eğer mesafeler[i][i] < 0:  
        hata "Negatif ağırlıklı döngü var"  
döndür mesafeler
```

Floyd-Warshall: Java

```
void floydWarshall(int[][] d) { // V x V
    for (int k = 0; k < V; k++) // ara
        for (int i = 0; i < V; i++) // kaynak
            for (int j = 0; j < V; j++) // hedef
                if (d[i][k] != SONSUZ
                    && d[k][j] != SONSUZ
                    && d[i][k] + d[k][j] < d[i][j]) {
                    d[i][j] = d[i][k] + d[k][j];
                    onceki[i][j] = onceki[k][j];
                }
    for (int i = 0; i < V; i++)
        if (d[i][i] < 0)
            throw new IllegalStateException(
                "Negatif döngü");
}
```

- Matris **yerinde** güncellenir; ayrı $D^{(k)}$ kopyası gerekmez.
- SONSUZ denetimi, `Integer.MAX_VALUE` toplamının taşmasını (overflow) önler.
- `onceki[i][j] = onceki[k][j]` : yeni yolda j'den önceki düğüm, $k \rightsquigarrow j$ yolundakiyle aynıdır.
- Negatif döngü: bir düğümün **kendine** uzaklığı negatif çıkarsa ($D[i][i] < 0$).

A* (A Yıldız) Arama Algoritması

- İki nokta (başlangıç → hedef) arasındaki en kısa yolu bulan bir **arama** algoritmasıdır.
- 1968'de Peter **Hart**, Nils **Nilsson** ve Bertram **Raphael** tarafından geliştirildi.
- Dijkstra'nın (düzgün maliyetli arama) gerçek maliyetini, açgözlü **en iyi ilk arama** (Greedy Best-First Search) yaklaşımının hedef tahminiyle birleştirir.
- Düzgün çalışması için iyi bir **tahmin fonksiyonu** (sezgisel, heuristic) gerekir.
- Düğüm sayısı arttıkça zaman ve bellek maliyeti hızla artar.

Algoritma ilkeleri

Her düğüm için bir **tahmin** $h(n)$ kullanılır: düğümün hedefe olan tahmini uzaklığı. Her adımda **açık listedeki** düğümler arasından gerçek maliyet ile tahmini maliyetin toplamı en küçük olan seçilir. Böylece algoritma hedefe doğru ilerlerken en az maliyetli yolu da korur.

A* Nasıl Çalışır? $f(n) = g(n) + h(n)$

$g(n)$: gerçek maliyet

Başlangıç düğümünden n'ye kadar **bulunan** yolun maliyeti ("g-maliyet"). Dijkstra'daki $d[n]$ ile aynıdır.

$h(n)$: sezgisel

n'den hedefe **tahmini** maliyet ("h-maliyet"). Kuş uçuşu uzaklık, Manhattan uzaklığı gibi.

$f(n) = g(n) + h(n)$

n üzerinden geçen bir çözümün tahmini toplam maliyeti. Açık listeden en küçük f'li düğüm genişletilir.

Seçim ölçütü	Algoritma	Davranış
yalnızca $g(n)$	Dijkstra	her yöne eşit yayılır; doğru ama çok düğüm açar
yalnızca $h(n)$	Açgözlü en iyi ilk	hedefe koşar; hızlı ama en kısa yolu garanti etmez
$g(n) + h(n)$	A*	hedefe yönelir; h uygunsa en kısa yolu garanti eder

Sezgisel tahminlere göre hedefe **yakın görünen** düğümler önceliklendirilir; ama g sayesinde pahalı bir yoldan gelen düğüm geri planda kalır.

A*: Algoritma Adımları

1. Başlangıç düğümünü seç, $g = 0$ ata ve **açık listeye** (open list, öncelikli kuyruk) ekle. Diğer düğümlerin g değeri ∞ .

Açık liste: keşfedilmiş ama henüz genişletilmemiş düğümler.

A*: Algoritma Adımları

1. Başlangıç düğümünü seç, $g = 0$ ata ve **açık listeye** (open list, öncelikli kuyruk) ekle. Diğer düğümlerin g değeri ∞ .

Açık liste: keşfedilmiş ama henüz genişletilmemiş düğümler.

2. Açık listeden **$f = g + h$** değeri en küçük düğümü çıkar ve **kapalı listeye** (closed list) koy.

Seçim ölçütü g ya da h tek başına değil, toplamaları $f = g + h$ 'dir.

A*: Algoritma Adımları

1. Başlangıç düğümünü seç, $g = 0$ ata ve **açık listeye** (open list, öncelikli kuyruk) ekle. Diğer düğümlerin g değeri ∞ .

Açık liste: keşfedilmiş ama henüz genişletilmemiş düğümler.

2. Açık listeden **$f = g + h$** değeri en küçük düğümü çıkar ve **kapalı listeye** (closed list) koy.

Seçim ölçütü g ya da h tek başına değil, toplamaları $f = g + h$ 'dir.

3. Çıkarılan düğüm **hedefse** çözüm bulundu: yolu önceki işaretçileriyle geri kur.

Hedef, açık listeye eklendiği anda değil, çıkarıldığı anda kesinleşir.

A*: Algoritma Adımları

1. Başlangıç düğümünü seç, $g = 0$ ata ve **açık listeye** (open list, öncelikli kuyruk) ekle. Diğer düğümlerin g değeri ∞ .

Açık liste: keşfedilmiş ama henüz genişletilmemiş düğümler.

2. Açık listeden **$f = g + h$** değeri en küçük düğümü çıkar ve **kapalı listeye** (closed list) koy.

Seçim ölçütü g ya da h tek başına değil, toplamaları $f = g + h$ 'dir.

3. Çıkarılan düğüm **hedefse** çözüm bulundu: yolu önceki işaretçileriyle geri kur.

Hedef, açık listeye eklendiği anda değil, çıkarıldığı anda kesinleşir.

4. Değilse her komşu için $g' = g(u) + w(u, v)$ hesapla; g' daha küçükse g , f ve önceki değerlerini **güncelle**, komşuyu açık listeye ekle.

Bu, Dijkstra'daki gevşetmenin aynısıdır.

A*: Algoritma Adımları

1. Başlangıç düğümünü seç, $g = 0$ ata ve **açık listeye** (open list, öncelikli kuyruk) ekle. Diğer düğümlerin g değeri ∞ .

Açık liste: keşfedilmiş ama henüz genişletilmemiş düğümler.

2. Açık listeden **$f = g + h$** değeri en küçük düğümü çıkar ve **kapalı listeye** (closed list) koy.

Seçim ölçütü g ya da h tek başına değil, toplamaları $f = g + h$ 'dir.

3. Çıkarılan düğüm **hedefse** çözüm bulundu: yolu önceki işaretçileriyle geri kur.

Hedef, açık listeye eklendiği anda değil, çıkarıldığı anda kesinleşir.

4. Değilse her komşu için $g' = g(u) + w(u, v)$ hesapla; g' daha küçükse g , f ve önceki değerlerini **güncelle**, komşuyu açık listeye ekle.

Bu, Dijkstra'daki gevşetmenin aynısıdır.

5. 2. adıma dön. Açık liste boşalırsa hedefe **yol yoktur**.

Seçim yalnızca son düğümün komşuları arasından değil, açık listenin tamamından yapılır.

Sezgisel Fonksiyonun Özellikleri

Kabul edilebilir (admissible)

$h(n) \leq h^*(n)$: gerçek kalan maliyeti **asla aşmaz**. Bu durumda A* hedefi çıkardığında bulduğu yol en kısadır.

Tutarlı (consistent)

$h(u) \leq w(u, v) + h(v)$ ve $h(\text{hedef}) = 0$. Tutarlı h ile bir düğüm kapalı listeye girdiğinde kesinleşir (yeniden açılmaz).

Izgarada sezgiseller

4 yön: Manhattan $|\Delta x| + |\Delta y|$. 8 yön: oktil (octile) $14 \cdot \min + 10 \cdot (\max - \min)$. Serbest düzlem: Öklid uzaklığı.

Uç durumlar

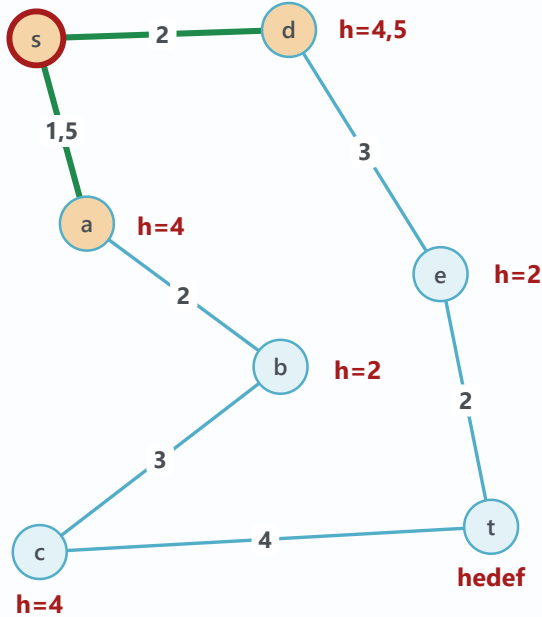
$h = 0 \rightarrow A^* = \text{Dijkstra}$. $h = h^*$ (mükemmel) $\rightarrow$ yalnızca en kısa yol üzerindeki düğümler açılır.
 $h > h^* \rightarrow$ hızlı ama en kısa yol garanti edilmez.

A*: Algoritma Karmaşıklığı

- Arama ağacı açısından en kötü durumda zaman ve bellek $O(b^d)$:
 - **b**: dallanma faktörü (bir düğümün ortalama komşu sayısı), **d**: hedefin derinliği (çözümdeki adım sayısı).
- Sonlu bir çizgede tutarlı h ile A*, Dijkstra gibi en fazla $O((V + E) \log V)$ iş yapar.
- Sezgisel ne kadar iyiye o kadar az düğüm açılır: **mükemmel** tahminde ($h = h^*$) yalnızca yol üzerindeki d düğüm genişletilir.
- Zayıf nokta **bellektir**: açık ve kapalı listeler büyür; IDA*, SMA* gibi türevler bellek sınırlıdır.

Dikkat: $O(b^d)$ mükemmel tahminin değil, en **kötü** durumun (h işe yaramaz, ör. $h = 0$) sınırıdır; mükemmel tahminde arama yol uzunluğuyla doğrusaldır.

A* Örnek: s'den t'ye (h değerleri düğüm yanında)



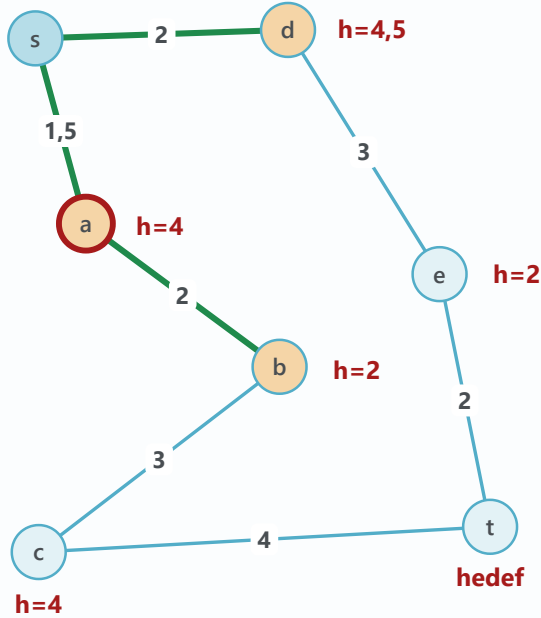
Açık liste

düğüm	g	h	f
a	1,5	4	5,5
d	2	4,5	6,5

Kapalı: s

s genişletilir: komşular a ($g = 1,5$) ve d ($g = 2$) açılır. $f(a) = 1,5 + 4 = \mathbf{5,5}$, $f(d) = 2 + 4,5 = 6,5$.

A* Örnek: s'den t'ye (h değerleri düğüm yanında)



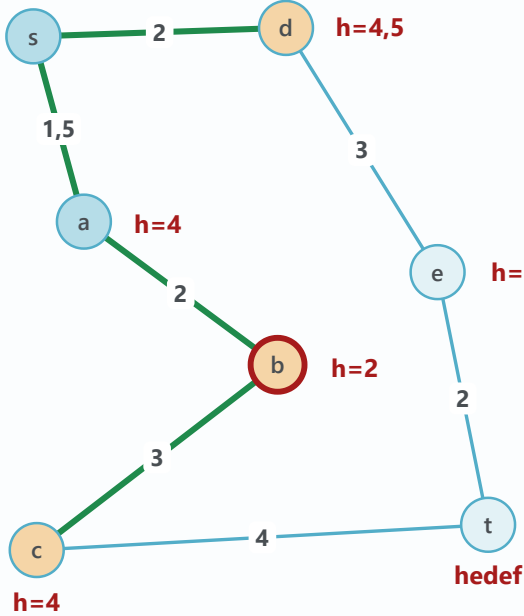
Açık liste

düğüm	g	h	f
b	3,5	2	5,5
d	2	4,5	6,5

Kapalı: s, a

En küçük f: a (5,5). a genişletilir $\rightarrow$ b: $g = 3,5$, $f(b) = 3,5 + 2 = 5,5$.

A* Örnek: s'den t'ye (h değerleri düğüm yanında)



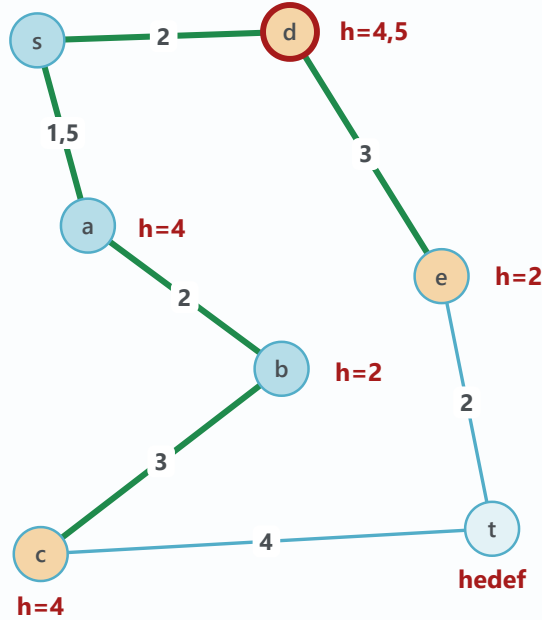
Açık liste

düğüm	g	h	f
d	2	4,5	6,5
c	6,5	4	10,5

Kapalı: s, a, b

b genişletilir ($5,5 < 6,5$) $\rightarrow$ c: $g = 6,5$, $f(c) = 6,5 + 4 = 10,5$. Bu yol pahalılaşıyor.

A* Örnek: s'den t'ye (h değerleri düğüm yanında)



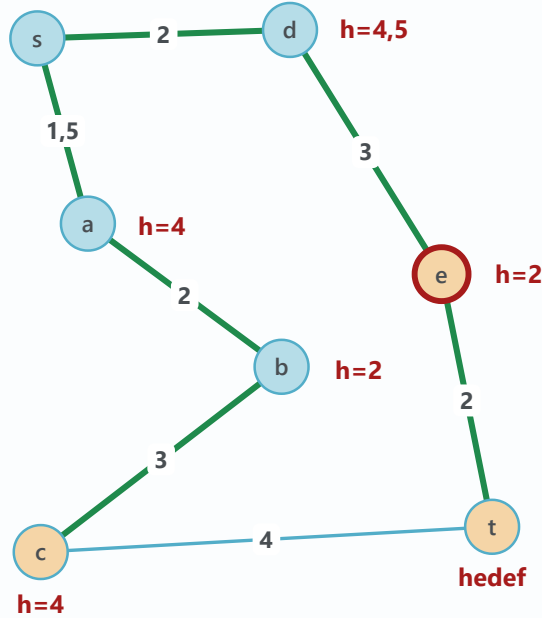
Açık liste

düğüm	g	h	f
e	5	2	7
c	6,5	4	10,5

Kapalı: s, a, b, d

Artık en küçük f: d (6,5). Arama diğer kola geçer → e: $g = 5$, $f(e) = 5 + 2 = 7$.

A* Örnek: s'den t'ye (h değerleri düğüm yanında)



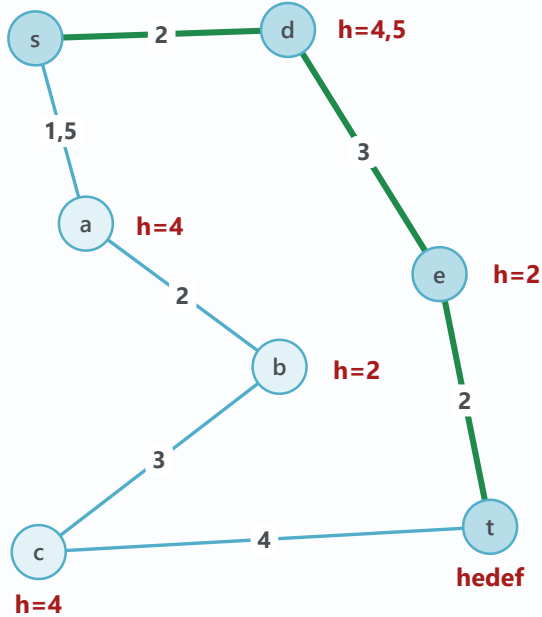
Açık liste

düğüm	g	h	f
t	7	0	7
c	6,5	4	10,5

Kapalı: s, a, b, d, e

e genişletilir $\rightarrow$ t: $g = 5 + 2 = 7$, $f(t) = 7 + 0 = \mathbf{7} < f(c) = 10,5$.

A* Örnek: s'den t'ye (h değerleri düğüm yanında)



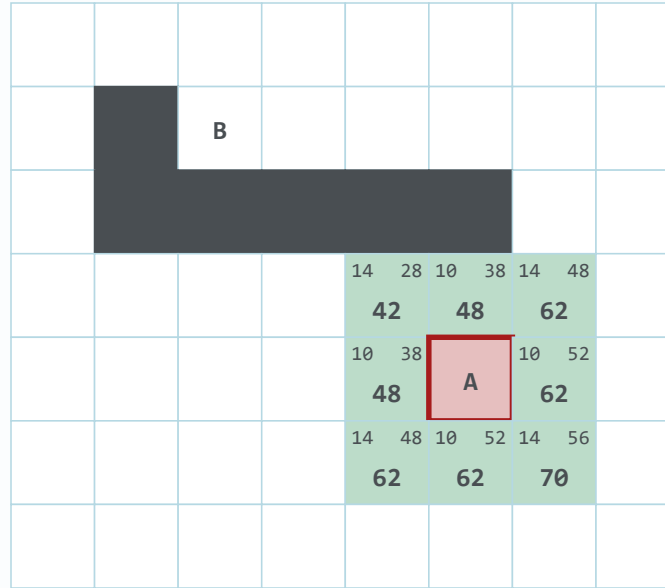
Açık liste

düğüm	g	h	f
c	6,5	4	10,5

Kapalı: s, a, b, d, e, t

t açık listeden çıkarıldı: **hedef**. En kısa yol $s \rightarrow d \rightarrow e \rightarrow t$, maliyet 7. c hiç genişletilmedi.

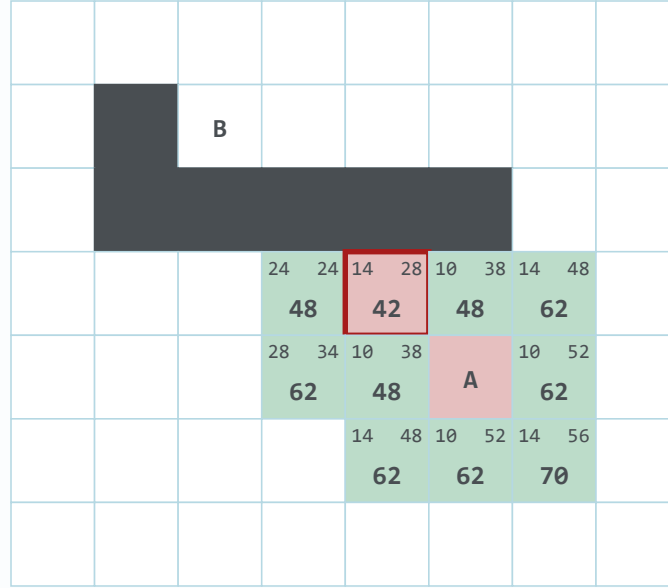
A* Örnek: Izgarada Duvarın Etrafından



■ açık
 ■ kapalı
 ■ yol
 ■ duvar
 sol üst g · sağ üst h · orta f

A genişletilir: 8 komşu açılır. Düz adım 10, çapraz 14; h = oktil uzaklık (duvarlar yok sayılır). En küçük f = 42.

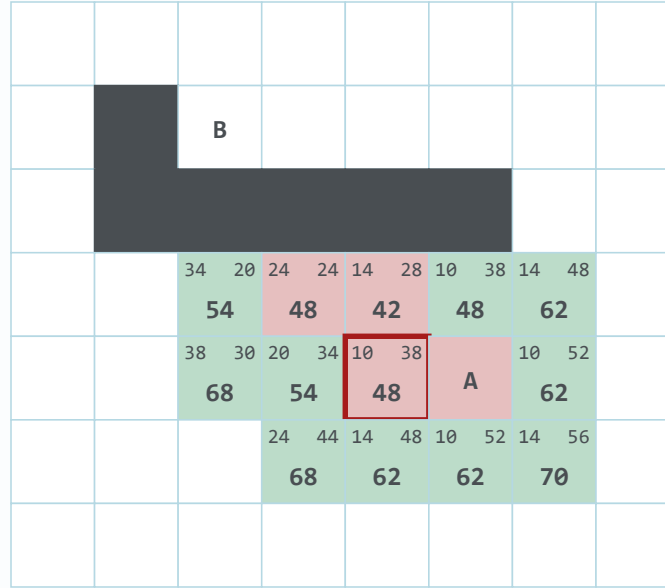
A* Örnek: Izgarada Duvarın Etrafından



■ açık ■ kapalı ■ yol ■ duvar sol üst g · sağ üst h · orta f

2. genişletme: en küçük $f = 42$ olan hücre (turuncu çerçeve) kapalı listeye geçer, komşuları açılır/güncellenir.

A* Örnek: Izgarada Duvarın Etrafından



■ açık ■ kapalı ■ yol ■ duvar sol üst g · sağ üst h · orta f

4. genişletme: en küçük $f = 48$ olan hücre (turuncu çerçeve) kapalı listeye geçer, komşuları açılır/güncellenir.

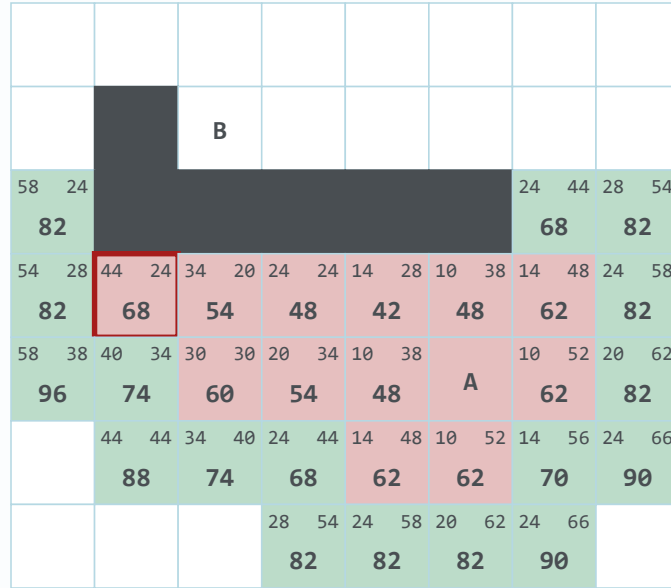


Adım 4 / 8



Adım 5 / 8

A* Örnek: Izgarada Duvarın Etrafından



■ açık ■ kapalı ■ yol ■ duvar sol üst g · sağ üst h · orta f

13. genişletme: en küçük $f = 68$ olan hücre (turuncu çerçeve) kapalı listeye geçer, komşuları açılır/güncellenir.
Alt yol tıkalı: $f = 68$ değerine çıkıldı, arama duvarın iki yanına yayılıyor.

A* Örnek: Izgarada Duvarın Etrafından

				72	10	62	14	52	24	48	34	52	44		
				82		76		76		82		96			
				68	0	58	10	48	20	38	30	34	40	38	50
				B	68	68		68		68		74		88	
58	24											24	44	28	54
82												68		82	
54	28	44	24	34	20	24	24	14	28	10	38	14	48	24	58
82		68		54		48		42		48		62		82	
58	38	40	34	30	30	20	34	10	38		A	10	52	20	62
96		74		60		54		48				62		82	
		44	44	34	40	24	44	14	48	10	52	14	56	24	66
		88		74		68		62		62		70		90	
						28	54	24	58	20	62	24	66		
						82		82		82		90			

■ açık ■ kapalı ■ yol ■ duvar sol üst g · sağ üst h · orta f

17. genişletme ($f = 68$): duvarın sağından dolaşan yol B'ye ulaştı; B açık listede $f = 68$.

A* Örnek: Izgarada Duvarın Etrafından

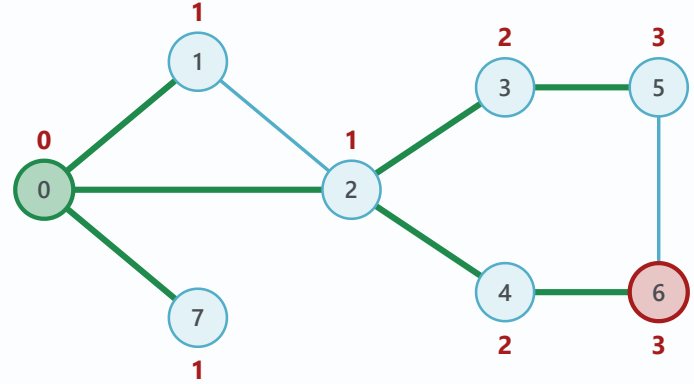
		72	10	62	14	52	24	48	34	52	44				
		82		76		76		82		96					
		68	0	58	10	48	20	38	30	34	40	38	50		
		B	68	68		68		68		74		88			
58	24									24	44	28	54		
82										68		82			
54	28	44	24	34	20	24	24	14	28	10	38	14	48	24	58
82		68		54		48		42		48		62		82	
58	38	40	34	30	30	20	34	10	38		A	10	52	20	62
96		74		60		54		48				62		82	
		44	44	34	40	24	44	14	48	10	52	14	56	24	66
		88		74		68		62		62		70		90	
					28	54	24	58	20	62	24	66			
					82		82		82		90				

■ açık ■ kapalı ■ yol ■ duvar sol üst g · sağ üst h · orta f

B çıkarıldı: yol A → yukarı → sağ üst çapraz → sol üst çapraz → duvar üstünden üç adım sola, maliyet **68** (2 çapraz × 14 + 4 düz × 10). 18 hücre genişletildi.

BFS: Ağırlıksız Çizgede En Kısa Yol

- Tüm kenarlar eşit (ağırlık 1) ise en kısa yol = en az kenarlı yol.
- **Genişlik öncelikli arama** (BFS) düğümleri kaynağa uzaklık sırasıyla, katman katman ziyaret eder: $d[v] = d[u] + 1$.
- Kuyruk (FIFO), Dijkstra'nın öncelikli kuyruğun yerini alır: $O(V + E)$.
- Örnek (yandaki çizge): $0 \rightarrow 6$ en kısa yolu **0** $\rightarrow$ **2** $\rightarrow$ **4** $\rightarrow$ **6** (3 kenar).
- **İki yönlü arama** (bidirectional): BFS hem kaynaktan hem hedeften başlatılır, iki dalga buluşunca durulur; yaklaşık $2 \cdot b^{d/2}$ düğüm açılır.



Sayılar: BFS seviyesi (kenar sayısı). Yeşil: BFS ağacı.

Karıştırılmaması Gereken: Gezgin Satıcı (TSP)

- **En kısa yol:** iki düğüm arasında en ucuz yol; Dijkstra ile **polinom** zamanda çözülür.
- **Gezgin Satıcı Problemi** (Travelling Salesman Problem, TSP): **tüm** şehirleri birer kez ziyaret edip başlangıca dönen en kısa tur.
- TSP **NP-zordur**; bilinen kesin algoritmalar üsteldir.
- **Kaba kuvvet** (brute force): bir şehir sabitlenir, $(n - 1)!/2$ tur denenir.
- **En yakın komşu** (nearest neighbor) sezgiseli: her adımda en yakın ziyaret edilmemiş şehre git; hızlı ($O(n^2)$) ama en iyiyi garanti etmez.

n şehir	farklı tur $(n - 1)!/2$
5	12
10	181.440
15	$\approx 4,36 \cdot 10^{10}$
20	$\approx 6,08 \cdot 10^{16}$

Ayrıntı: Bölüm 14 (Dal Sınır).
Simülatörde iki yöntemi karşılaştırabilirsiniz.

KARŞILAŞTIRMA

En Kısa Yol Algoritmalarının Karşılaştırması

Algoritma	Problem	Negatif kenar	Negatif döngü	Zaman	Yöntem
BFS	tek kaynak, ağırlıksız	—	—	$O(V + E)$	katman katman
Dijkstra	tek kaynak	hayır	hayır	$O(V^2) / O((V + E) \log V)$	açgözlü
Bellman-Ford	tek kaynak	evet	tespit eder	$O(V \cdot E)$	tekrarlı gevşetme (DP)
Floyd-Warshall	tüm çiftler	evet	tespit eder	$\Theta(V^3)$	dinamik programlama
A*	tek çift	hayır	hayır	en kötü $O(b^d)$	sezgisel + açgözlü

Ağırlıksız mı?

BFS yeterli.

Ağırlıklar ≥ 0 ?

Dijkstra; hedef tek ve iyi h varsa A*.

Negatif kenar?

Bellman-Ford.

Tüm çiftler, küçük V?

Floyd-Warshall.

ÖZET

Özet

Konu	Hatırlanacaklar
Gevşetme	$d[u] + w < d[v]$ ise $d[v]$ ve $\pi[v]$ güncellenir; tüm algoritmaların ortak işlemi
Dijkstra	en küçük d 'li düğümü kesinleştir, komşularını gevşet; ağırlıklar ≥ 0 ; $O((V + E) \log V)$
Bellman-Ford	tüm kenarları $V - 1$ tur gevşet; V . turda gevşeyen kenar = negatif döngü; $O(V \cdot E)$
Floyd-Warshall	$D[i][j] = \min(D[i][j], D[i][k] + D[k][j])$, k en dışta; $D[i][i] < 0 \rightarrow$ negatif döngü; $\Theta(V^3)$
A*	$f = g + h$; kabul edilebilir h ile en kısa yol; $h = 0 \rightarrow$ Dijkstra
BFS	ağırlıksız çizgede en az kenarlı yol; $O(V + E)$
Yol	$\pi(\Pi)$ işaretçileriyle hedeften geriye; en kısa yol ağacı

Etkileşimli Simülatörler

En Kısa Yol Algoritmaları

Rastgele bağlı, ağırlıklı bir çizge üretin; Dijkstra, Bellman-Ford ve Floyd-Warshall'ı A düğümünden son düğüme çalıştırıp adım günlüğünü karşılaştırın. Düğümler sürüklenebilir.

A* Yol Bulma

Izgaraya tıklayarak duvar ekleyip kaldırın, A*'ı çalıştırın; keşfedilen hücreleri ve bulunan en kısa yolu izleyin (4 yön, Manhattan sezgiseli).

Gezgin Satıcı Problemi

Şehirler üretin; hızlı en yakın komşu sezgiselini kesin kaba kuvvet aramasıyla karşılaştırın ve tur uzunluklarını görün.

ÖZET

Sık Yapılan Hatalar

Negatif kenarda Dijkstra

Kesinleşmiş bir düğüme sonradan daha kısa yol açılabilir; sonuç sessizce yanlış olur. Bellman-Ford kullanın.

$\infty + w$ taşması

`Integer.MAX_VALUE + w` negatife döner ve "daha kısa" görünür. Önce `d[u]  $\neq$  SONSUZ` denetleyin.

Floyd-Warshall döngü sırası

k ara düğüm döngüsü **en dışta** olmalı; i, j, k sırası yanlış sonuç verir.

Tur sayısı

Bellman-Ford'da $V - 1$ tur gerekebilir (Örnek 2); tek turda biteceğini varsaymayın. Denetim turu ayrıdır.

A*'da hedefte durmak

Hedef açık listeye **eklendiğinde** değil, listeden **çıkarıldığında** durulur. Fazla tahmin eden h en kısa yolu kaçıırır.

Yönsüz kenarda negatif ağırlık

Yönsüz negatif kenar $u \rightarrow v \rightarrow u$ ile kendi başına negatif dögüdür.

ÖZET

Örnek Problemler

Yönlü çizge: $A \rightarrow B$ 4, $A \rightarrow C$ 2, $C \rightarrow B$ 1, $B \rightarrow D$ 5, $C \rightarrow D$ 8, $C \rightarrow E$ 10, $D \rightarrow E$ 3; kaynak A.

1. Dijkstra düğümleri hangi sırayla kesinleştirir? Son d değerleri nedir?
2. A'dan E'ye en kısa yolu π dizisiyle yazın.
3. Kenarlar yukarıdaki alfabetik sırayla ($A \rightarrow B$, $A \rightarrow C$, $B \rightarrow D$, $C \rightarrow B$, $C \rightarrow D$, $C \rightarrow E$, $D \rightarrow E$) incelenirse Bellman-Ford'un her turunda kaç gevşetme olur?
4. $1 \rightarrow 2$ (4), $2 \rightarrow 3$ (1), $1 \rightarrow 3$ (7), $3 \rightarrow 1$ (-2) çizgesinin Floyd-Warshall son D matrisi nedir? Negatif döngü var mı?
5. $V = 1000$, $E = 5000$ ve negatif kenarlar var. Hangi algoritma, yaklaşık kaç işlem?

► Cevapları göster

ÖZET

Alıştırmalar

1. Dijkstra Örnek 1'deki çizgede kaynağı **5** olarak tüm uzaklıkları bulun.
2. `dijkstra(kaynak, hedef)` : hedef kuyruktan çıkınca duran sürümü yazın.
3. Her düğüm için **en kısa yol sayısını** da hesaplayan Dijkstra yazın.
4. Bellman-Ford'a erken durma ekleyin; en iyi durumda karmaşıklık nedir?
5. Floyd-Warshall'ı boolean matrisle **geçişli kapanış** (Warshall) için uyarlayın.
6. Manhattan uzaklığı 4 yönlü ızgarada neden kabul edilebilir, 8 yönlüde neden değildir?

► Cevaplar / ipuçları

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022 (Bölüm 22–23).
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, 2011 (Bölüm 4.4: Shortest Paths).
- E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik* 1, 1959.
- R. Bellman, "On a Routing Problem", *Quarterly of Applied Mathematics* 16, 1958.
- R. W. Floyd, "Algorithm 97: Shortest Path", *Communications of the ACM* 5(6), 1962; S. Warshall, "A Theorem on Boolean Matrices", *Journal of the ACM* 9(1), 1962.
- P. E. Hart, N. J. Nilsson ve B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Trans. SSC* 4(2), 1968.

Son

Bölüm 4.2: En Kısa Yol Algoritmaları

Önceki bölüm: Bölüm 4.1: Çizge Gezinme Algoritmaları · Sonraki bölüm: Bölüm 4.3: Minimum Kapsayan Ağaç Algoritmaları · [Terim sözlüğü](#)