Algoritmalar

Çizge Algoritmaları

En Kısa Yol Problemi

- Çizgelerdeki bir diğer önemli problem de bir düğümden diğer bir düğüme olan en kısa yolun bulunmasıdır.
- Bu problem de günlük hayatımızda oldukça sık karşılaştığımız bir problemdir.
- A noktasında B noktasına en kısa yoldan gitmek veya ağ şebekelerinde veri paketlerini en uygun yoldan transfer etmek gibi.

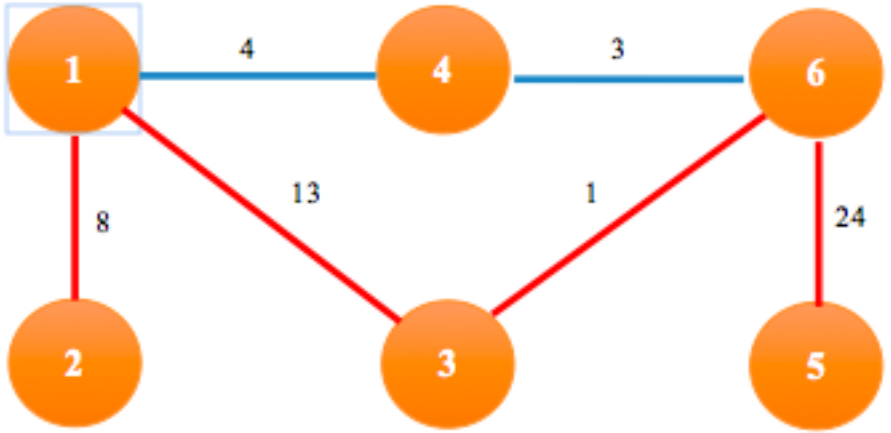
En Kısa Yol Problemi

- Bu noktada “kısa” teriminin en düşük maliyeti veya en uygun seçeneği ifade ettiğine dikkat ediniz.
- Bu ders kapsamında bu maliyetin uzunluk olduğu kabul edilecektir.
- Bir düğümden diğer bir düğüme olan yolun toplam maliyeti yol üzerindeki kenarların maliyetlerinin toplamına eşit olacaktır.

En Kısa Yol Problemi

- Aşağıdaki çizgede 1 düğümünden 6 düğümüne 4 numaralı düğüm üzerinden ulaşıldığında toplam yol maliyeti yol üzerindeki kenarların değerlerinin toplamıdır.

$$\{(1,4),(4,6)\}=4+3=7$$

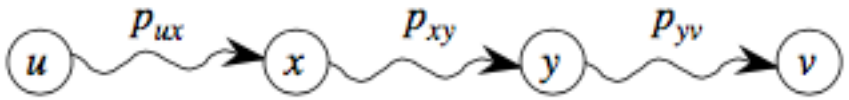


En Kısa Yol Problemi

- Tek kaynak en kısa yol problemi üzerinde durulacaktır
  - Tek bir kaynaktan ulaşılabilecek tüm düğümlere olan en kısa yolların hesaplanması
- Negatif kenar ağırlıkları
  - Negatif ağırlıklı döngü olmadığı zaman sorun değil
  - Negatif ağırlığa sahip döngü varsa, sürekli bu döngü hareket ederek yol kısaltılabilir

En Kısa Yol Problemi

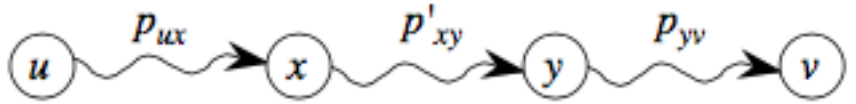
- Optimal altyapı özelliği
- Lemma: Eğer bir p yolu en kısa yol ise, p'nin herhangi bir parçası da en kısa yoldur.
- İspat: Kes-yapıştır yöntemi ile sipat edilebilir



- u düğümünden v düğümüne olan p yolu en kısa yol olsun
- Lemmaya göre x ve y düğümleri arasında kalan yol da en kısa yol olmalıdır
- Böyle olmadığını varsayalım. x ve y arasında p_{xy} yolundan daha kısa bir p'_{xy} yolu olsun

En Kısa Yol Problemi

- Böyle olmadığını varsayalım. x ve y arasında p_{xy} yolundan daha kısa bir p'_{xy} yolu olsun.



- Bu durumda u ve v düğümleri arasında yukarıda gösterilen yeni bir p' yolu vardır
- p ve p' yollarının uzunlukları karşılaştırılır

$$\begin{aligned} w(p') &= w(p_{ux}) + w(p'_{xy}) + w(p_{yv}) \\ &< w(p_{ux}) + w(p_{xy}) + w(p_{yv}) \\ &= w(p)\,. \end{aligned}$$

- En kısa yol p den daha kısa bir p' yolu olamayacağına göre x ve y arasındaki yolun (p_{xy}) en kısa yol olduğu ispat edilmiş olur.

En Kısa Yol Problemi

- Üçgen eşitsizliği
  - (u,v) kenarı için
  - Bir kaynak düğüm s den v ye olan uzaklık hesaplanırken, her zaman

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

- En kısa yollar döngü içeremez
  - Negatif ağırlıklı olamayacağını gösterdik
  - Pozitif ağırlıklı olan da ihmal edilerek en kısa yol oluşturulur

Dijkstra Algoritması

- Dijkstra Algoritması en kısa yol hesaplarında en çok kullanılan yöntemlerdendir.
- Bilgisayar ağları, taşımacılık, posta gibi hizmetlerde bir çok gerçek uygulamada kullanılmaktadır.
- Belki de en önemli uygulaması Internet'in temel yönlendirme algoritması olan OSPF (Open Shortest Path First) için bir temel oluşturmasıdır.

Dijkstra Algoritması

- Bu algoritma yönlendirilmiş ve yönlendirilmemiş çizgeler için kullanılabilir.
- Çizgenin pozitif değerlerle ağırlıklandırılmış olması zorunludur (negatif kenar ağırlığı bulunamaz).
- Bu algoritma da açgözlü problem çözme yaklaşımını takip eder, verilen bir başlangıç düğümünün önce komşularına olan en kısa yolu bulur, daha sonra onların komşularına, ve aynı şekilde çizge içindeki tüm düğümlere olan en kısa yolları bulmuş olur.

Dijkstra (G,s)

Adım 1. $Initialize(Q)$ // min priority-queue

Adım 2. **for** every v in V

Adım 3. $d_v \leftarrow \infty$ // her düğüm başlangıçta sonsuz uzaklıkta

Adım 4. $p_v \leftarrow NIL$ // başlangıçta ebeveyn yok

Adım 5. $Insert(Q, v, d_v)$ //düğümler min priority-queue'ya yerleşir

Adım 6. $d_s \leftarrow 0$

Adım 7. $Decrease(Q, s, d_s)$ // başlangıç düğümünü güncelle

Adım 8. $V_T \leftarrow \emptyset$

Adım 9. **for** $i \leftarrow 0$ **to** $|V| - 1$

Adım 10. $u^* \leftarrow DeleteMin(Q)$ // kuyruktaki en küçük elemanı al

Adım 11. $V_T \leftarrow V_T \cup u^*$

Adım 12. **for** $V - V_T$ de u^*nun komşusu olan her u için **do**

Adım 13. **if** $d_{u^*} + w(u^*, u) < d_u$

Adım 14. $d_u \leftarrow d_{u^*} + w(u^*, u)$

Adım 15. $p_u \leftarrow u^*$

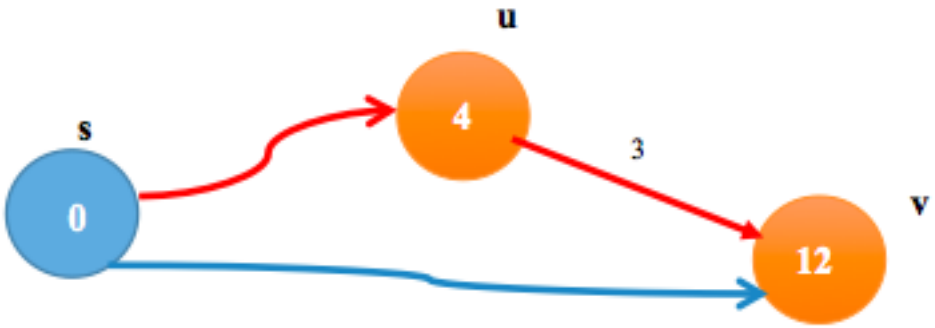
Adım 16. $Decrease(Q, u, d_u)$

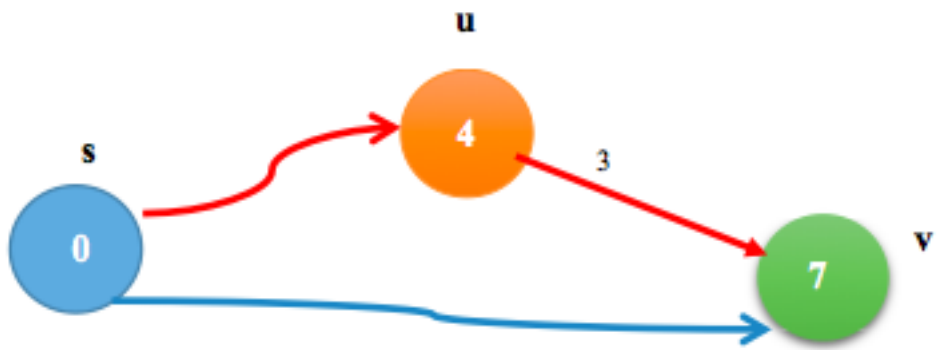
Dijkstra Algoritması

- Sözde koddan anlaşılacağı üzere min priority-queue'dan alınan her düğüm için o düğümün komşularının başlangıç düğümüne olan uzaklığının kısaltılıp kısaltılmayacağı kontrol edilmektedir (Adım 12-16).
- Bu işlem güncelleme (relaxation) olarak bilinir, aşağıdaki örnek güncelleme işlemini açıklamaktadır.
- Örnekte düğümlerin üzerindeki değerler o an itibarı ile başlangıç düğümüne (s) olan uzaklığı belirtmektedir.

Dijkstra Algoritması

- *v* düğümüne *s*'den altta mavi ile gösterilen yoldan ulaşma maliyeti 12'dir. Ancak *u* düğümü üzerinden ulaşılırsa bu maliyet düşebilir. *u* düğümüne *s* düğümünden ulaşma maliyeti 4 ve (*u,v*) kenarı değeri 3'dür. Buna göre 12 > 4+3 olduğundan *s* düğümünden *v* düğümüne ulaşmanın yeni maliyeti 4+3=7 olarak aşağıdaki gibi güncellenebilir.



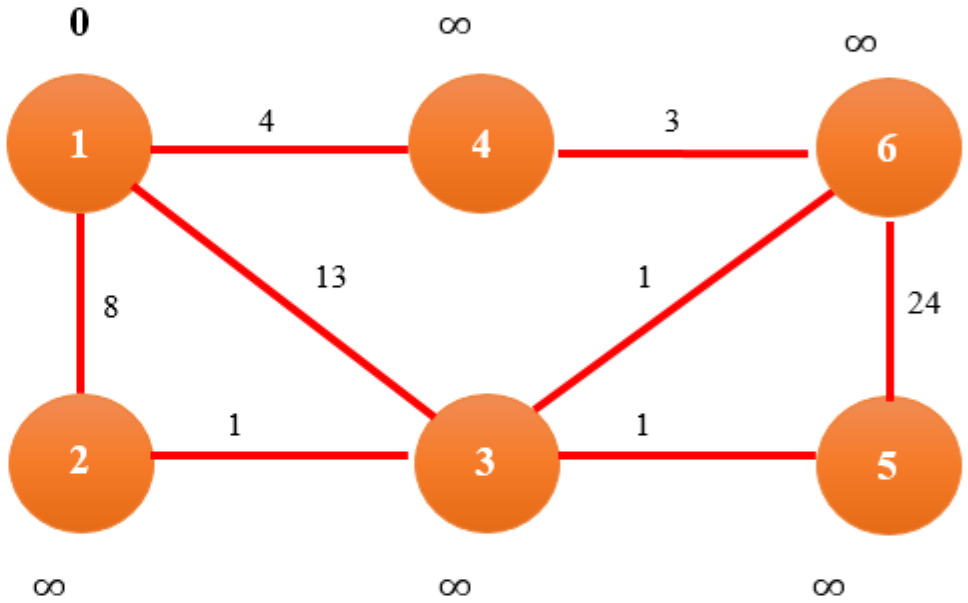


Dijkstra Algoritması

- Priority-queue'dan (kuyruk) alınan her düğümün tüm komşuları için bu güncelleme işlemi yapıldığında her düğüme olan en kısa yol bulunmuş olmaktadır.
- Algoritmada priority-queue min-heap kullanılarak gerçeklenebilir bu durumda algoritmanın toplam çalışma zamanı $O(E \lg V)$ değerine eşit olur.
- Aşağıda algoritmanın çalışmasını gösteren bir örnek verilmiştir.

Örnek

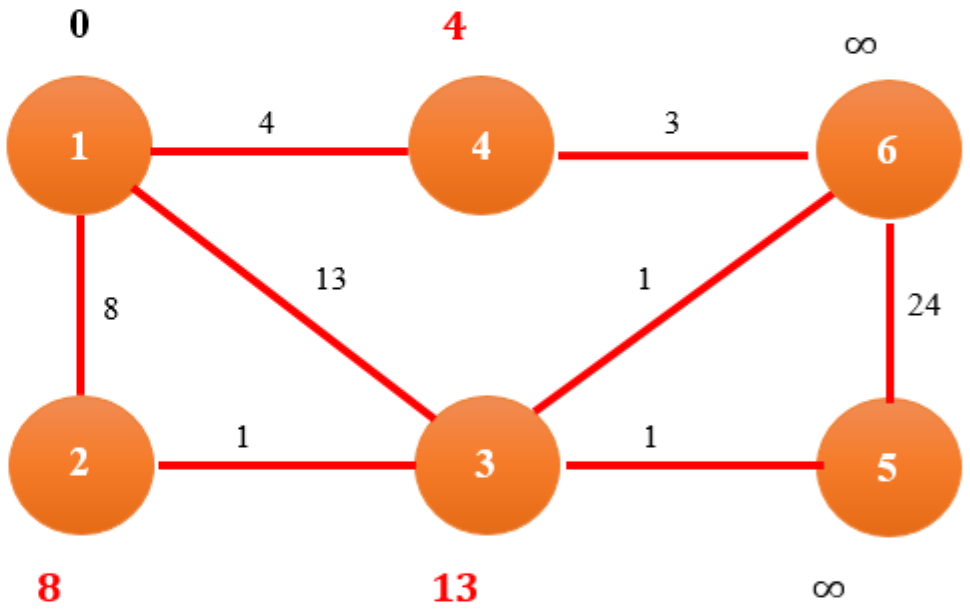
- 1 numaralı düğümden diğer tüm düğümlere olan en kısa yolları bulunuz.



1	0
2	∞
3	∞
4	∞
5	∞
6	∞

Kuyruk

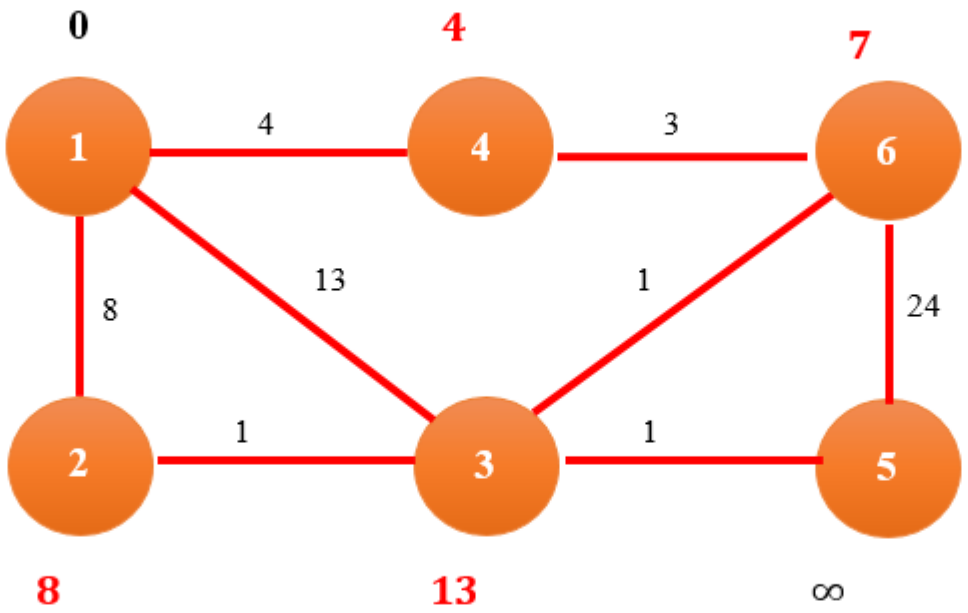
Örnek



4	4
2	8
3	13
5	∞
6	∞

Kuyruk

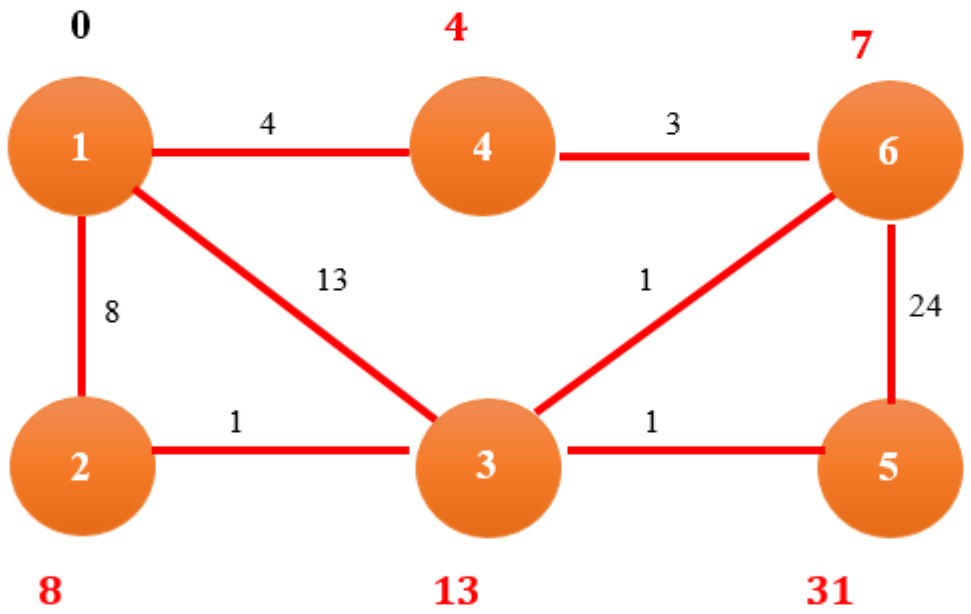
Örnek



6	7
2	8
3	13
5	∞

Kuyruk

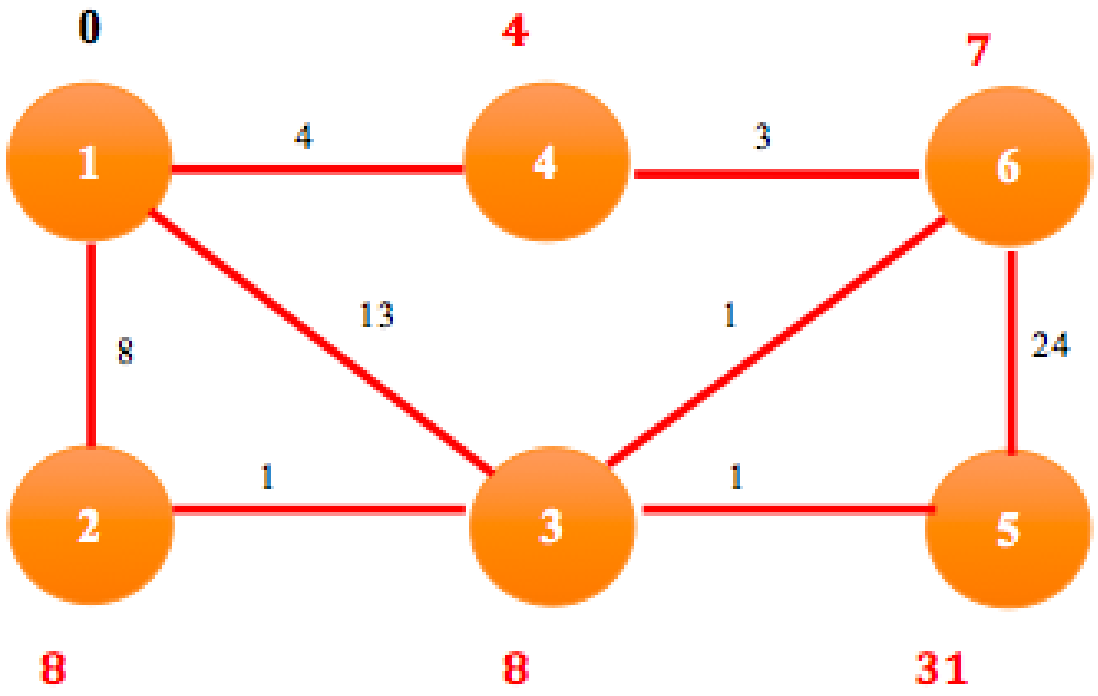
Örnek



2	8
3	8
5	31

Kuyruk

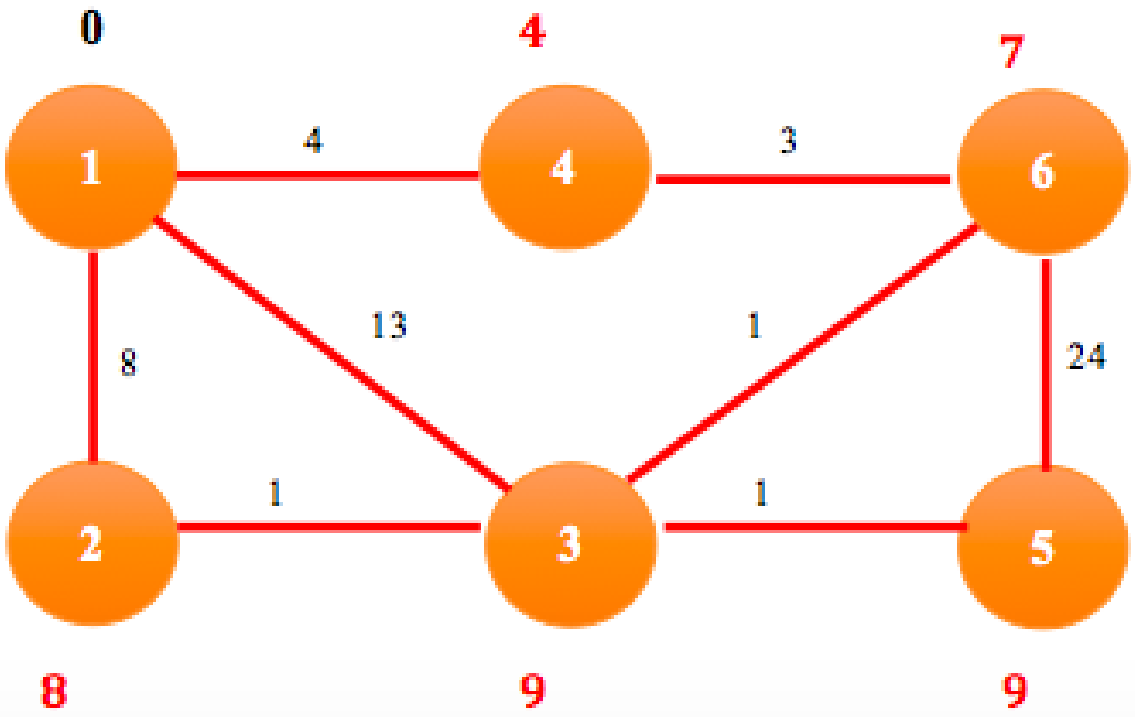
Örnek



3	8
5	31

Kuyruk

Örnek



5	9

Kuyruk

Bellman-Ford Algoritması

- Negatif kenarlar olabilir
- Negatif döngü olup olmadığını kontrol eder
  - Returns TRUE if there is no negative weight cycle
  - Returns FALSE if there is a negative weight cycle
- Dağıtık bir algoitmadır
  - Dijkstra merkezi bir algoritmaydı
- Routing Information Protocol'ün temelini oluşturur
  - Internette ISP'ler bu algoritma ile yönlendirme yapar

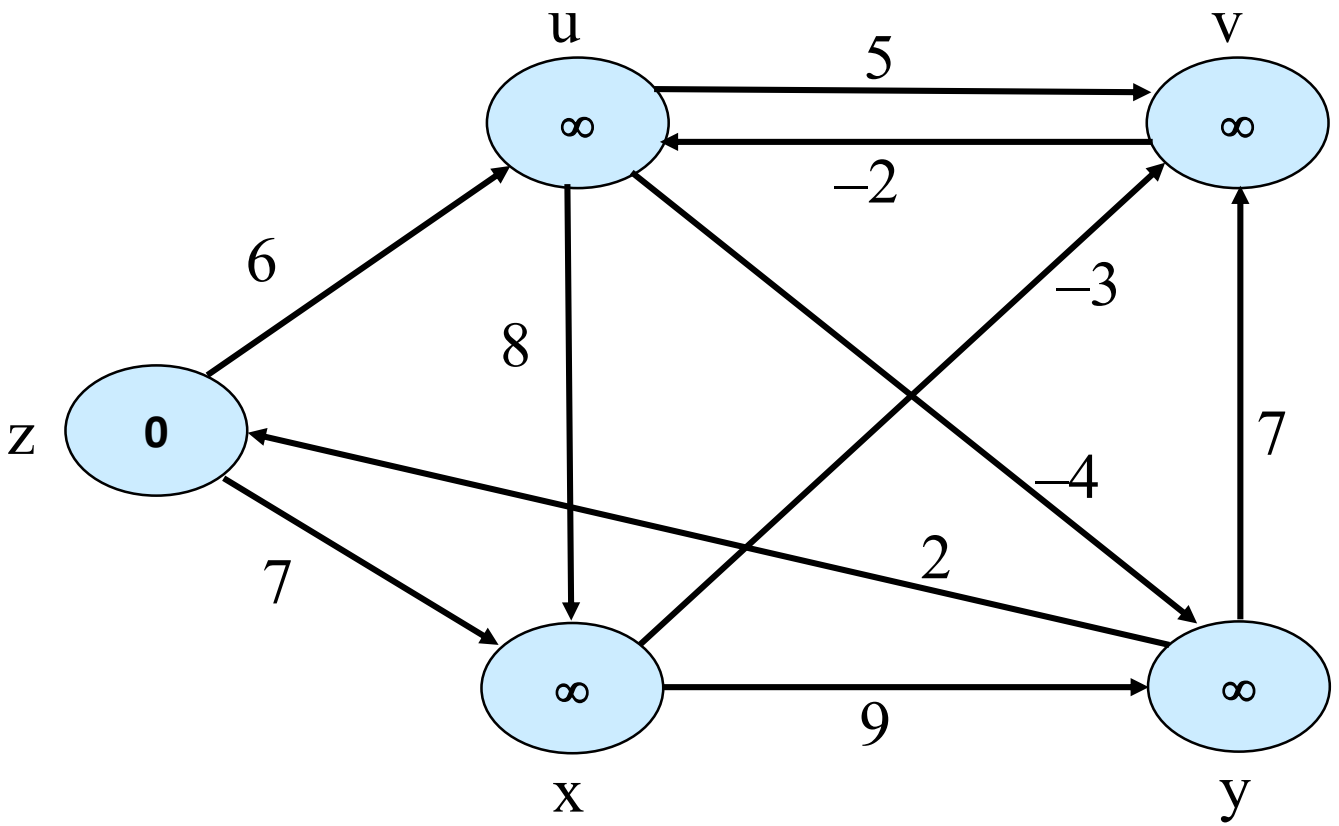
Bellman-Ford Algoritması

```
BELLMAN-FORD(V, E, w, s)
INIT-SINGLE-SOURCE(V, s)
for i ← 1 to |V| − 1
    do for each edge (u, v) ∈ E
        do RELAX(u, v, w)
for each edge (u, v) ∈ E
    do if d[v] > d[u] + w(u, v)
        then return FALSE
return TRUE
```

```
INIT-SINGLE-SOURCE(V, s)
for each v ∈ V
    do d[v] ← ∞
       π[v] ← NIL
d[s] ← 0
```

```
RELAX(u, v, w)
if d[v] > d[u] + w(u, v)
  then d[v] ← d[u] + w(u, v)
       π[v] ← u
```

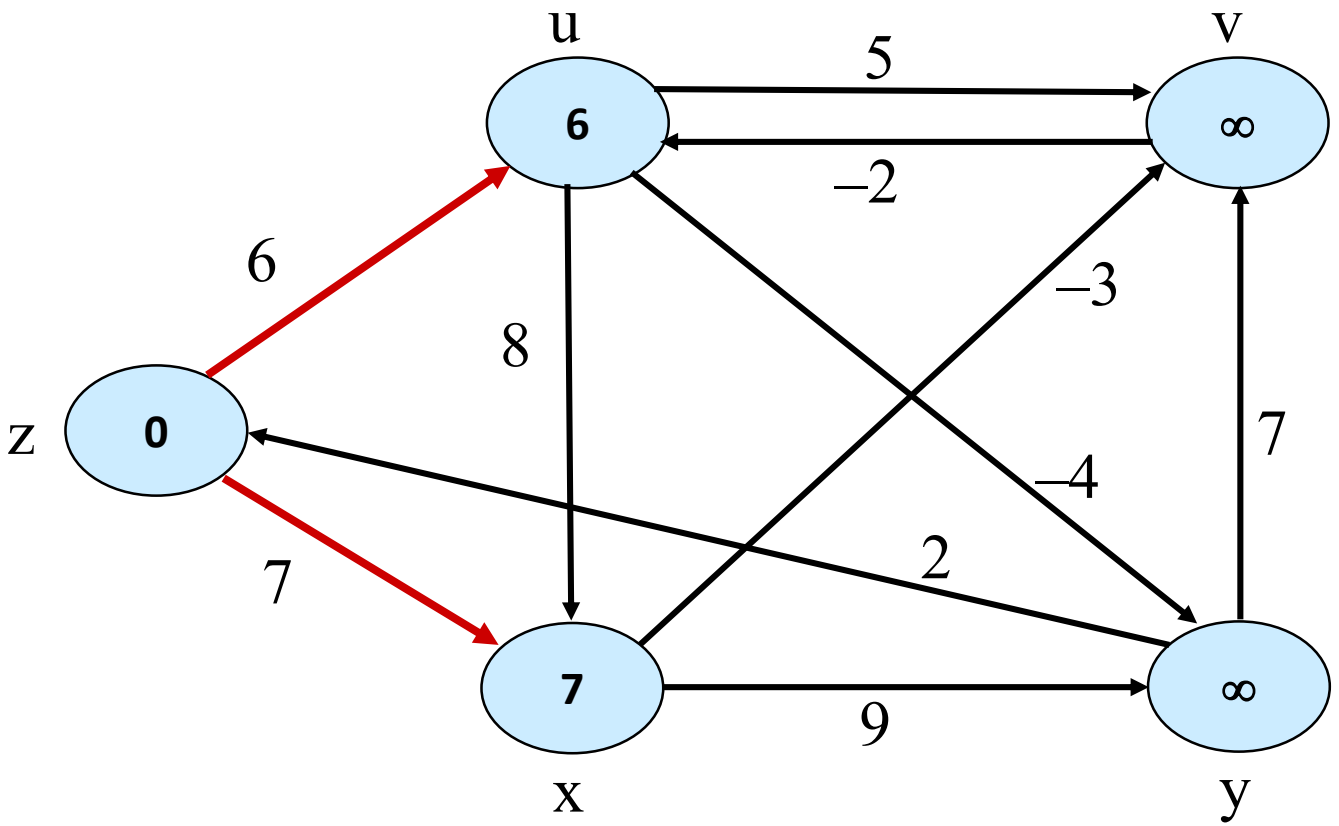
Örnek



	z	u	v	x	y
	1	2	3	4	5
0	0	∞	∞	∞	∞
1	0	6	∞	7	∞
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	–2

Örnek

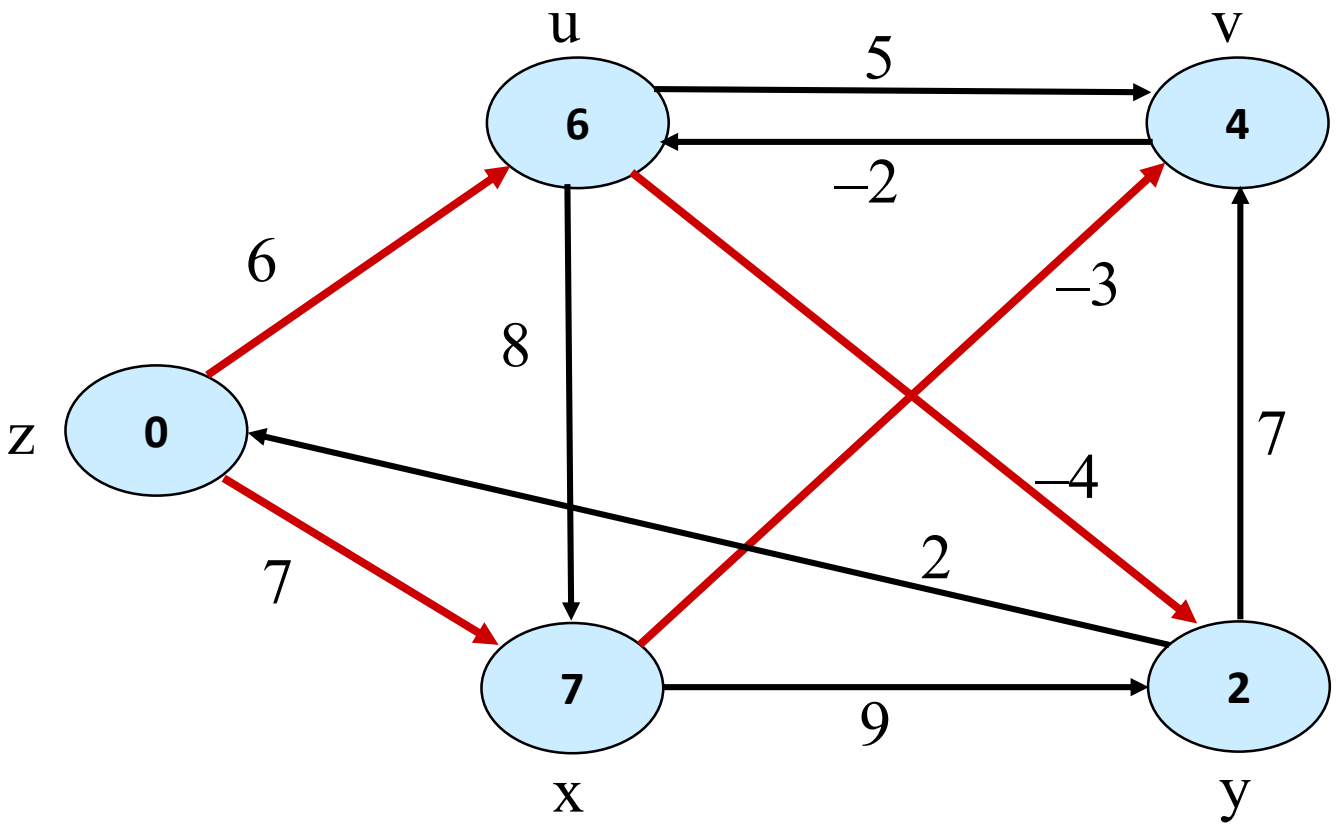
1. Geçiş



	z	u	v	x	y
	1	2	3	4	5
0	0	∞	∞	∞	∞
1	0	6	∞	7	∞
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	–2

Örnek

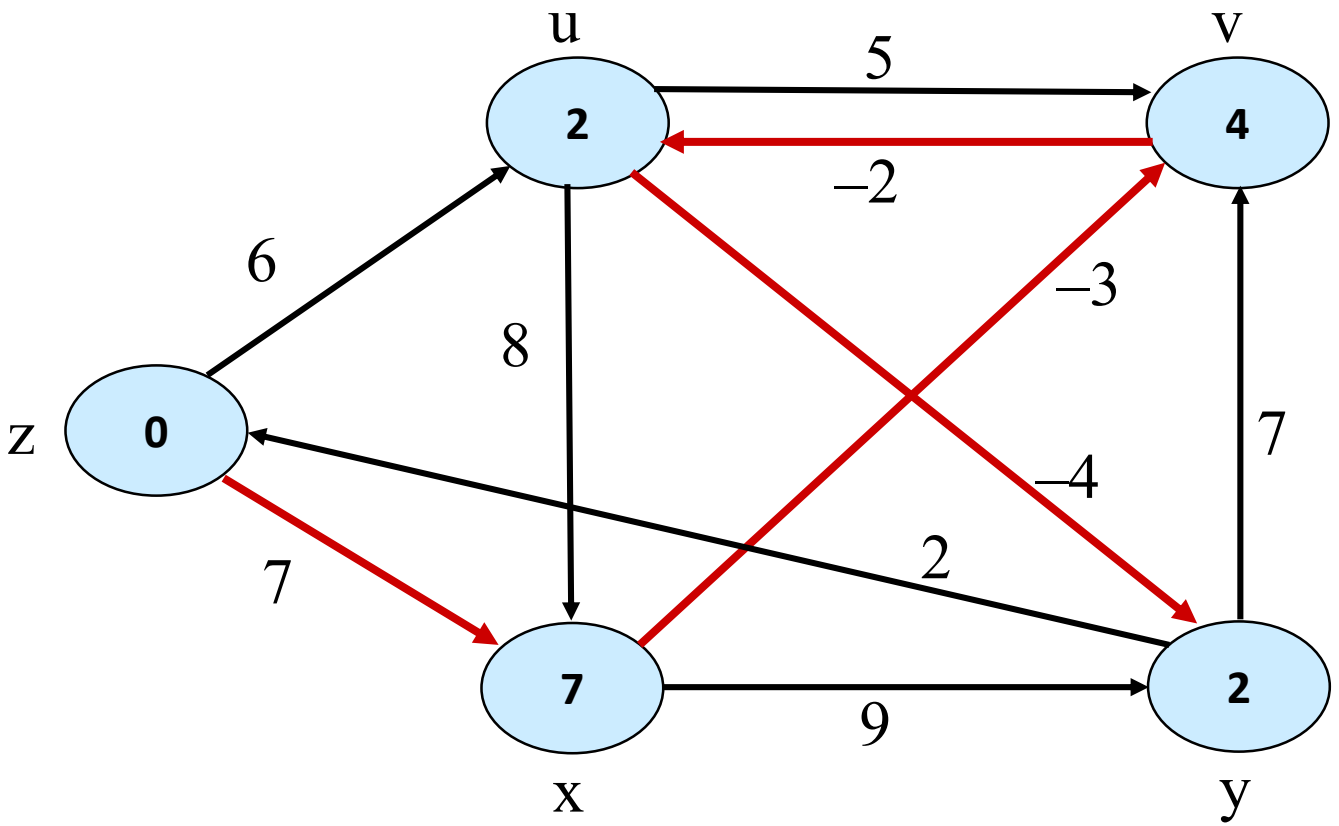
2. Geçiş



	z	u	v	x	y
	1	2	3	4	5
0	0	∞	∞	∞	∞
1	0	6	∞	7	∞
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	–2

Örnek

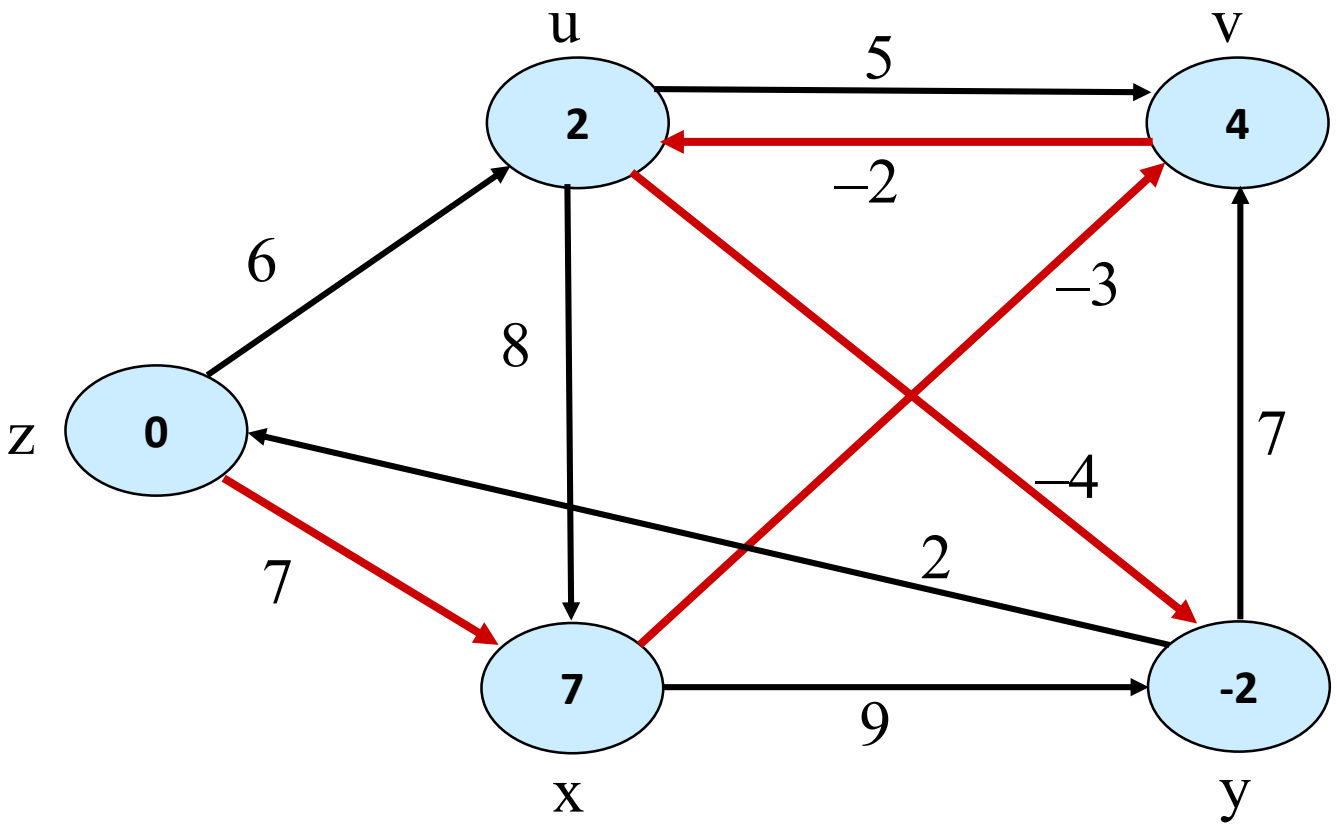
3. Geçiş



	z	u	v	x	y
	1	2	3	4	5
0	0	∞	∞	∞	∞
1	0	6	∞	7	∞
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	–2

Örnek

4. Geçiş



	z	u	v	x	y
	1	2	3	4	5
0	0	∞	∞	∞	∞
1	0	6	∞	7	∞
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	–2