

11.Hafta

En kısa yollar I-II-III

Devam

- Negatif Ağırlıklı En Kısa Yollar
- Doğruluk
- Çözümleme

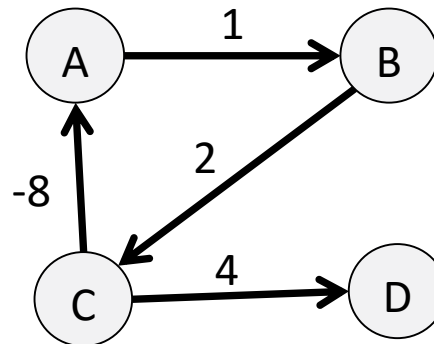


En Kısa Yollar II

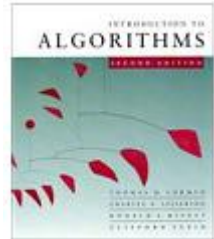
- Bellman-Ford algoritması

Negatif Maliyetli Çember

- Eğer graf **negatif maliyetli çember** içeriyorsa, en az maliyetli yol **tanımlanamaz**.



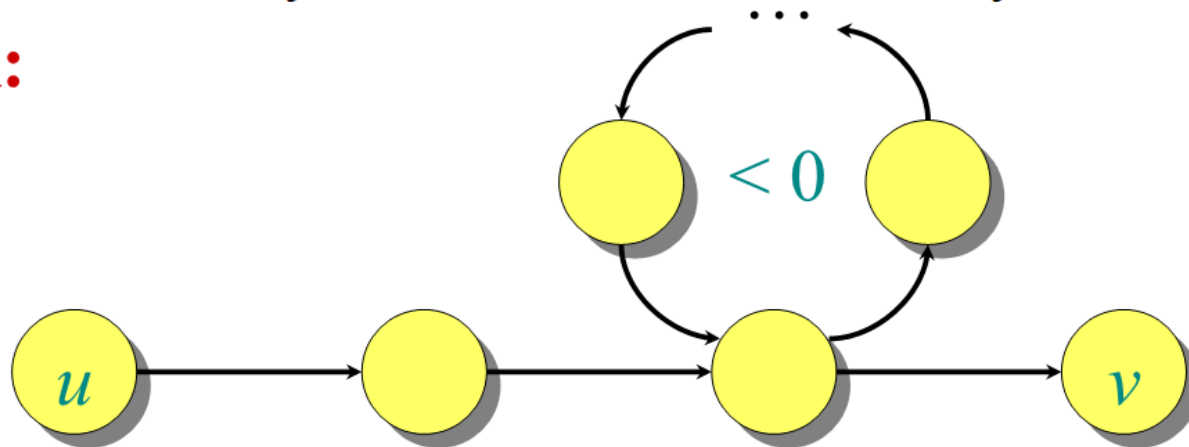
- A'dan D'ye en az maliyetli yol nedir?
 - veya B'den C'ye?



Negatif-ağırlık çevrimleri

Hatırlatma: Eğer grafik $G = (V, E)$ negatif ağırlık çevrimi içeriyorsa, en kısa yollardan bazıları bulunmayabilir.

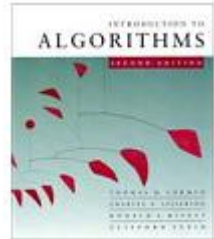
Örnek:



Bellman-Ford algoritması: Bir $s \in V$ kaynağından tüm $v \in V'$ lere bütün kısa yol uzunluklarını bulur ya da bir negatif ağırlık çevrimi olduğunu saptar.

Bellman-Ford Algoritması

- Ana mantık:
 - Her düğüm için bir uzaklık tahmini oluşturulur.
 - Başlangıç olarak maliyet(s)=0 diğer düğümler için maliyet(u)= ∞ olarak atanır.
 - En az maliyetli yol hesaplanana kadar tüm kenarlar üzerinden güncelleme yapılır.
- Algoritma ayrıca grafın negatif maliyetli kenarının olup olmadığını da bulur.
- Dijkstra'nın algoritmasına göre daha yavaş çalışır.
- (Dijkstra algoritması negatif kenarlarda kullanılmaz.)



Bellman-Ford algoritması

```

 $d[s] \leftarrow 0$ 
for each(her bir)  $v \in V - \{s\}$  (için)
    do(yap)  $d[v] \leftarrow \infty$ 
    
```

} **ilklendirme**

```

for(için)  $i \leftarrow 1$  to  $|V| - 1$  ' ( e )
    do for each edge(her kenar için yap)  $(u, v) \in E$ 
        do(yap) if(eğer)  $d[v] > d[u] + w(u, v)$ 
            then(sonra)  $d[v] \leftarrow d[u] + w(u, v)$ 
    
```

} **Gevşetme adımı**

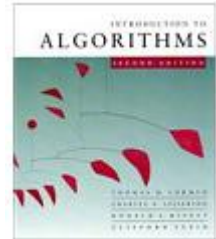
(her)**for each edge** $(u, v) \in E$ (kenarı için)

(yap eğer) **do if** $d[v] > d[u] + w(u, v)$

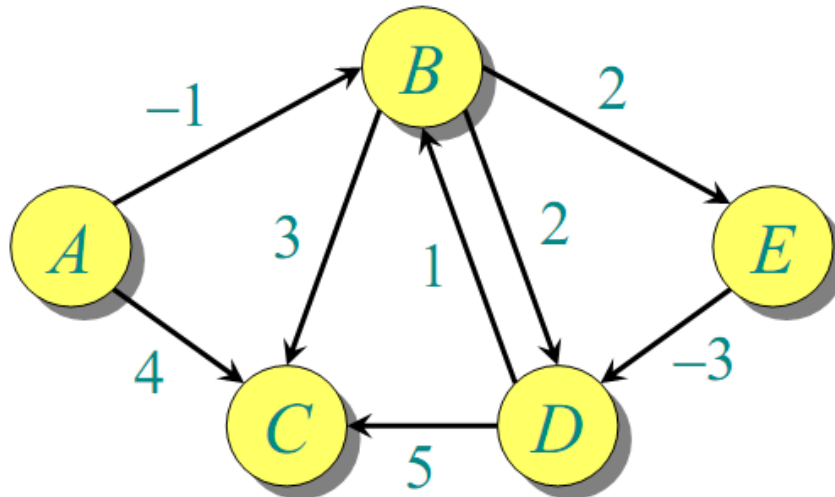
sonra bunu negatif ağırlık çevrimi var diyerek raporla

Sonunda, $d[v] = \delta(s, v)$, negatif ağırlık çevrimi yoksa.

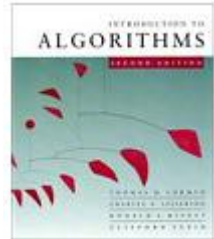
Süre = $O(VE)$.



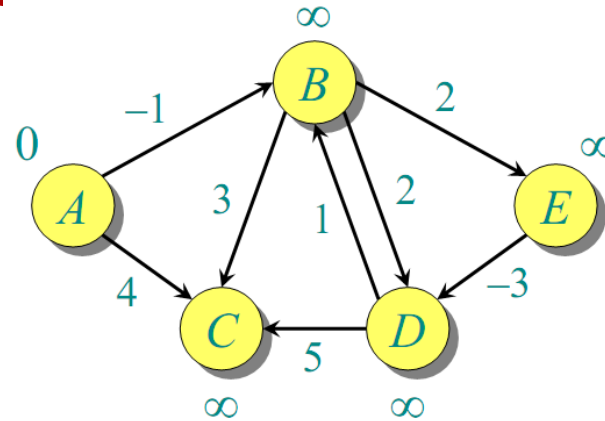
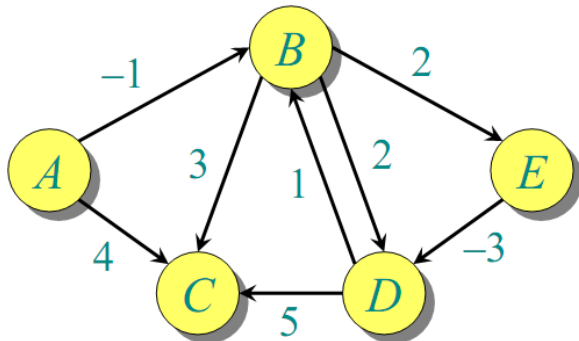
Bellman-Ford örneği



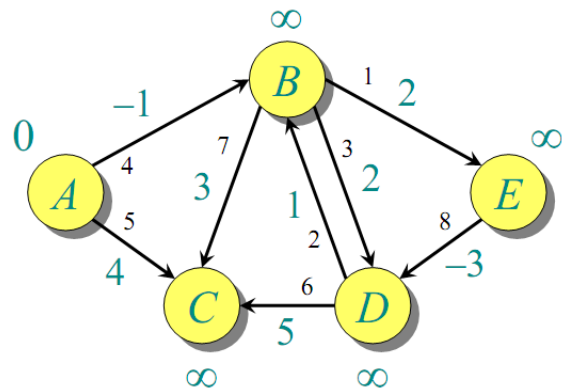
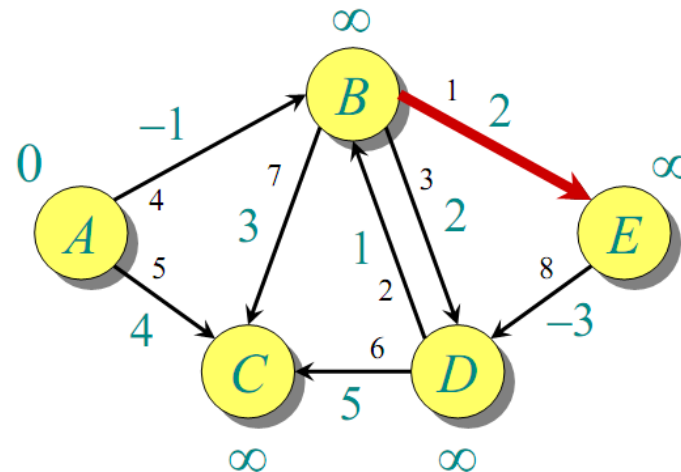
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
0	∞	∞	∞	∞



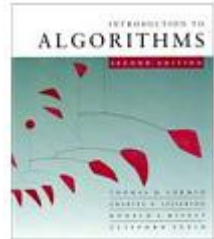
Bellman-Ford örneği



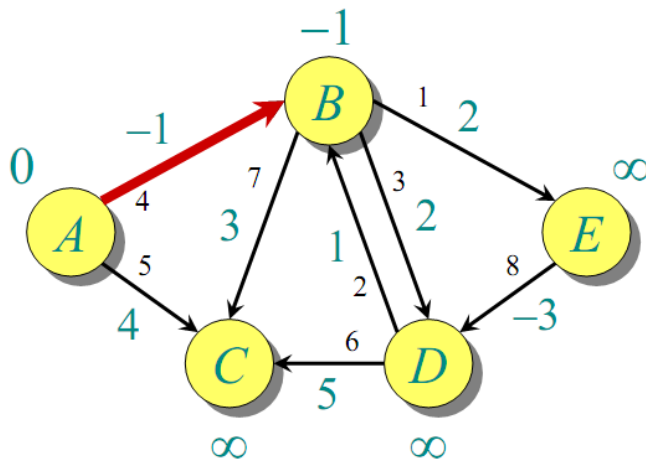
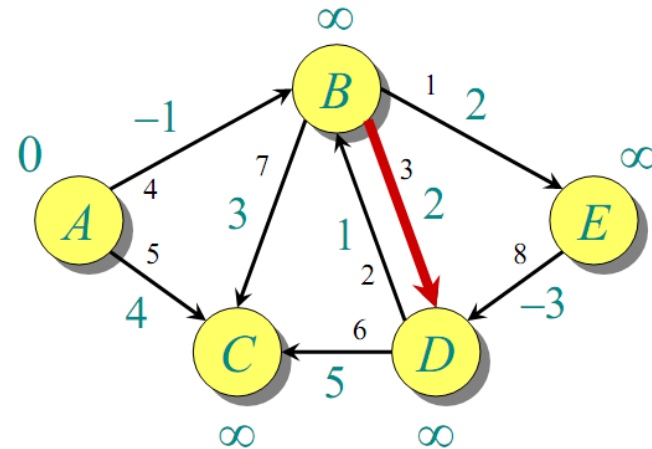
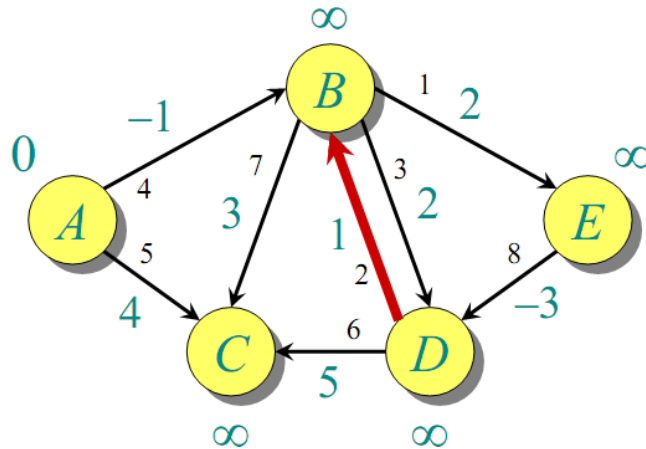
İklendirme.



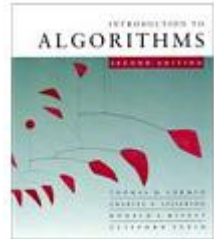
Köşe gevşetme düzeni



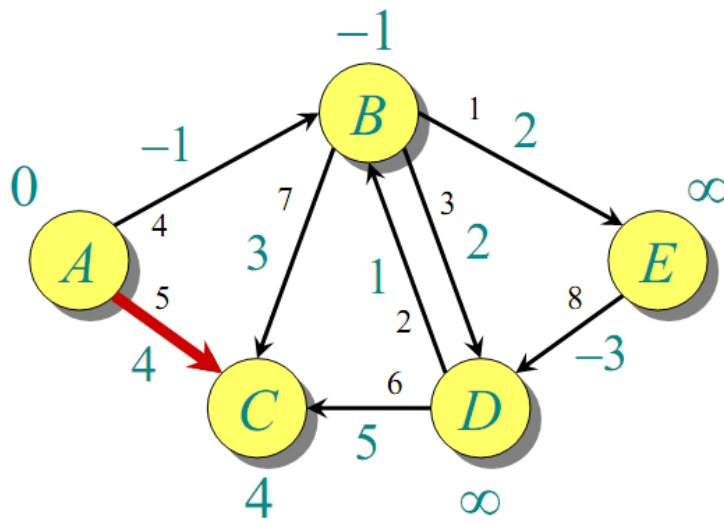
Bellman-Ford örneği



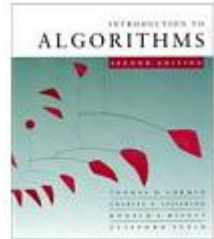
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
0	∞	∞	∞	∞
0	-1	∞	∞	∞



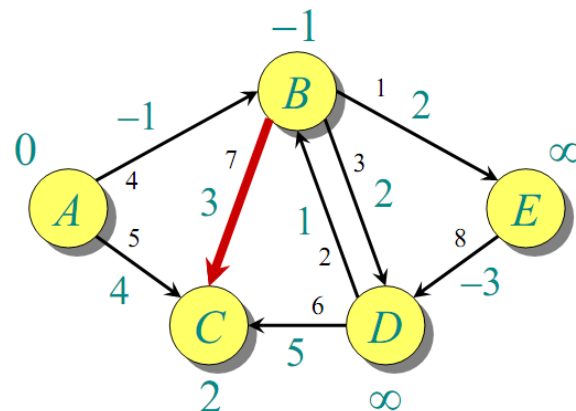
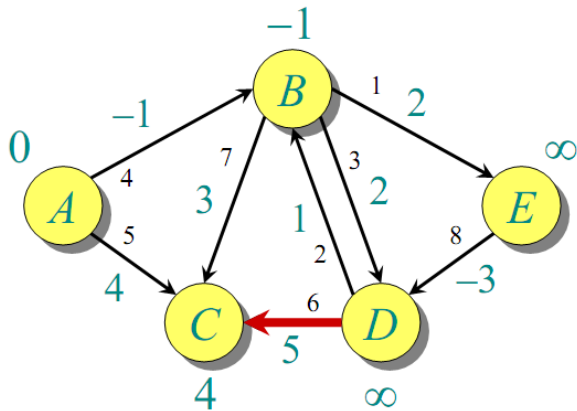
Bellman-Ford örneği



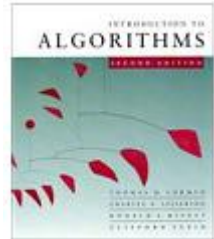
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞



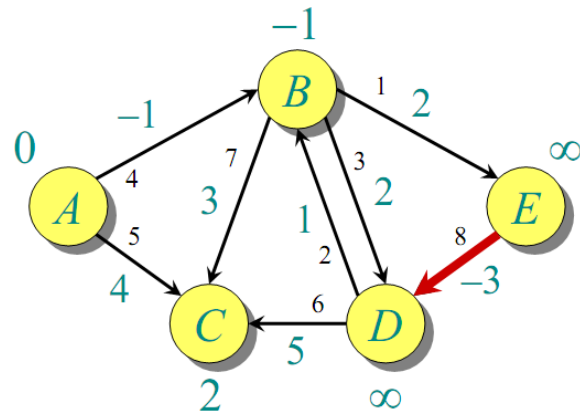
Bellman-Ford örneği



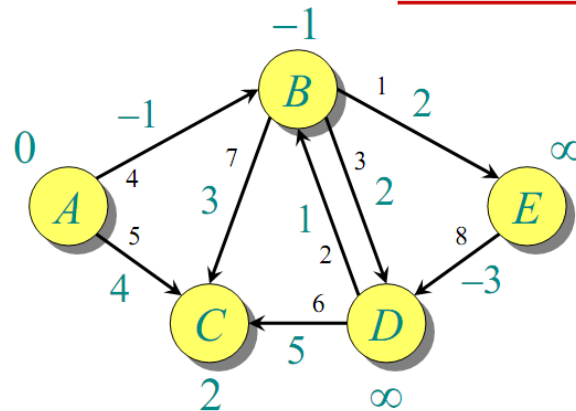
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞



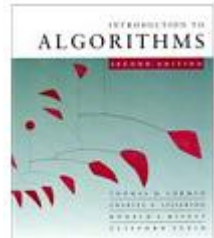
Bellman-Ford örneği



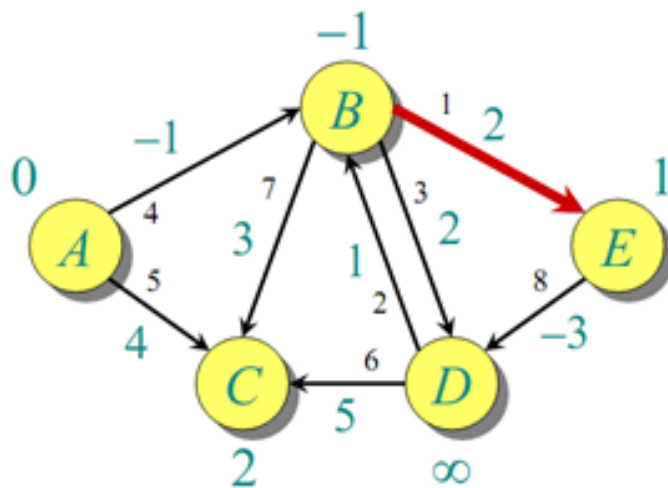
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞



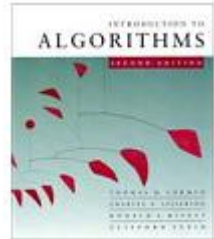
1. geçişin sonunda



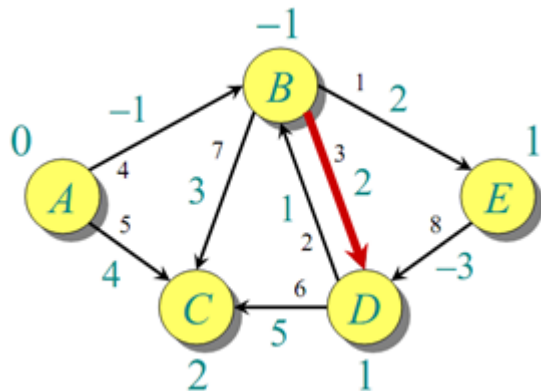
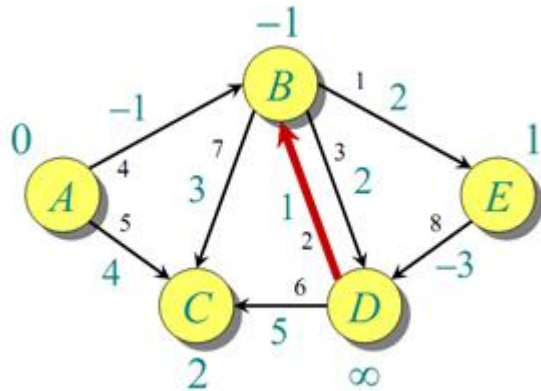
Bellman-Ford örneği



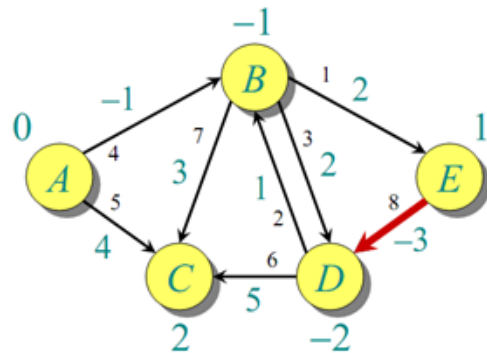
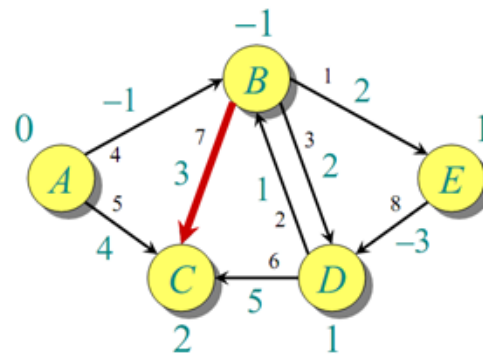
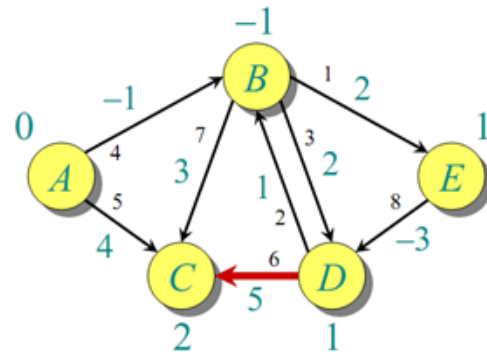
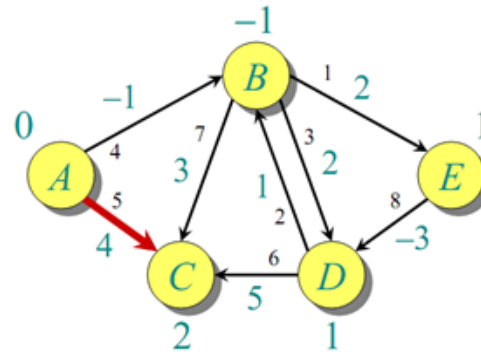
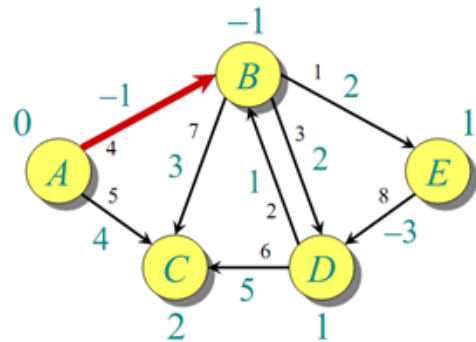
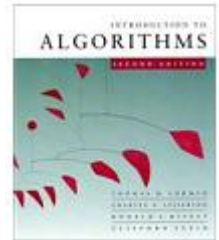
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1



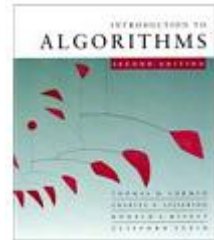
Bellman-Ford örneği



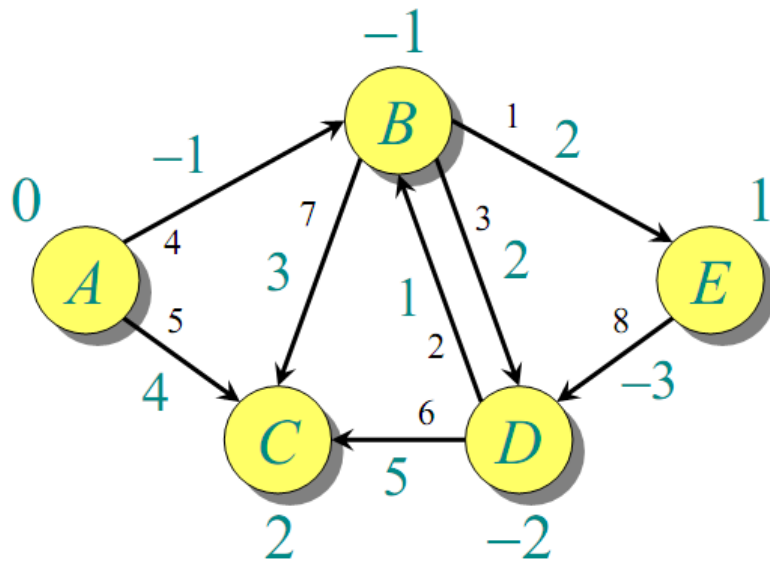
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1



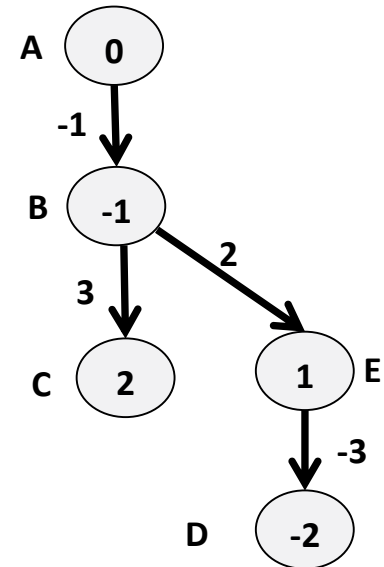
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1
0	-1	2	-2	1



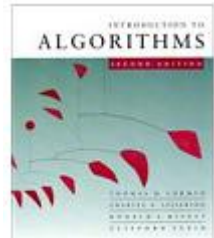
Bellman-Ford örneği



<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1
0	-1	2	-2	1



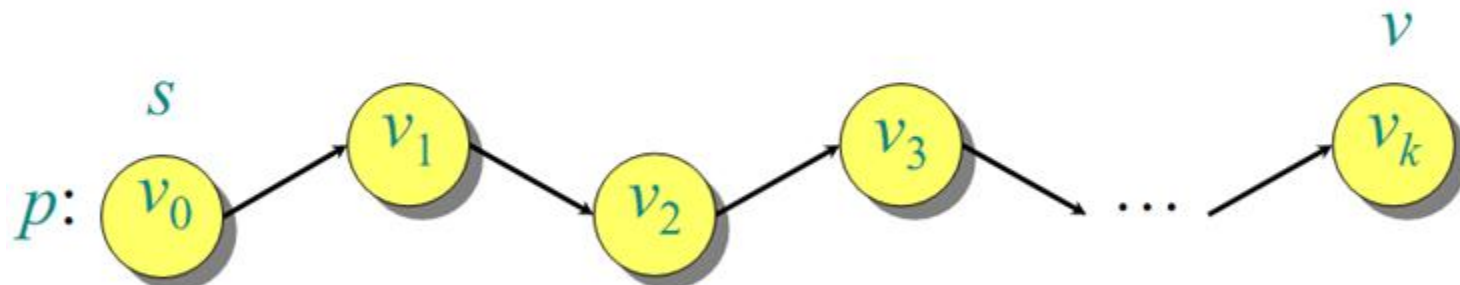
2. geçişin sonu (ve 3 ve 4).



Doğruluk

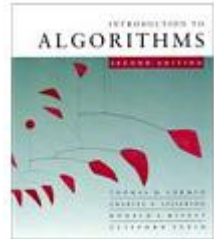
Teorem. Eğer $G = (V, E)$ hiç negatif ağırlık çevrimi içermiyorsa, sonrasında Bellman-Ford algoritması bütün $v \in V$ ler için $d[v] = \delta(s, v)$ yi çalıştırır.

Kanıt. $v \in V$ herhangi bir köşe olsun ve s' den v' ye, üzerinde en az sayıda köşe olan en kısa yolun p olduğunu farzedin.

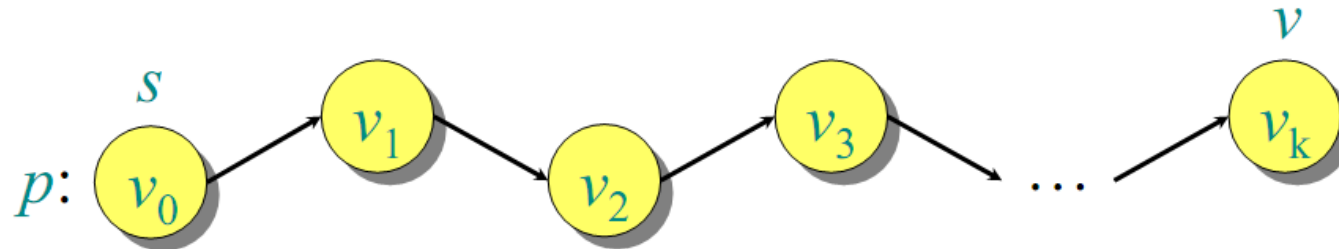


p en kısa yol ise,

$$\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i).$$



Doğruluk



İlk olarak, $d[v_0] = 0 = \delta(s, v_0)$ ve $d[v_0]$ sonraki gevşetmeler tarafından değiştirilmemiş.

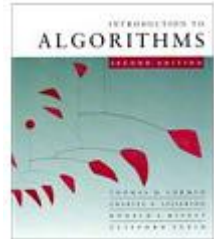
(Ders 14' teki $d[v] \geq \delta(s, v)$ kuramı sebebiyle).

- E' den 1 geçiş sonra, $d[v_1] = \delta(s, v_1)$.
- E' den 2 geçiş sonra, $d[v_2] = \delta(s, v_2)$.
- \vdots
- E' den k geçiş sonra, $d[v_k] = \delta(s, v_k)$.

Eğer G negatif ağırlık çevrimi içermiyorsa, p basittir.

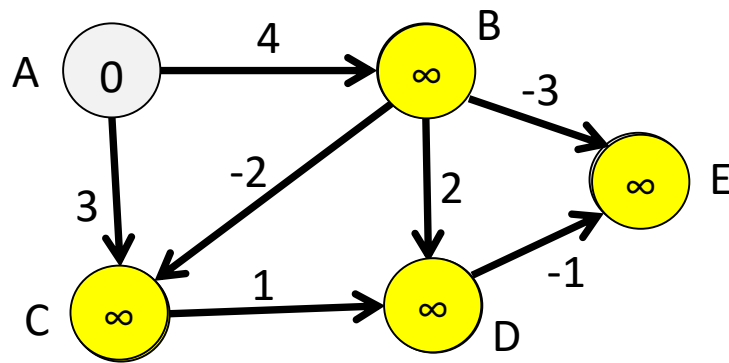
En uzun basit yolun $\leq |V| - 1$ kadar kenarı vardır. ◻

Negatif ağırlık çevrimlerini bulma



Doğal Sonuç. $|V|-1$ geçiş sonra $d[v]$ değeri birleşmede başarısız olursa, G' de s' den erişilebilir bir negatif ağırlık çevrimi vardır. □

Bellman-Ford Alg: Örnek

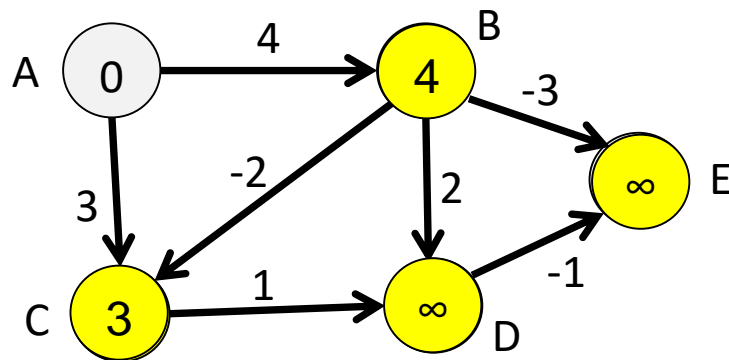


Düğüm	M	Ata
A	0	-
B	∞	
C	∞	
D	∞	
E	∞	

1	2	3	4	5	6	7
(C, D)	(A, B)	(A, C)	(B, C)	(B, D)	(B, E)	(D, E)

İlk Yineleme
(İlklendirme)

Bellman-Ford Alg: Örnek

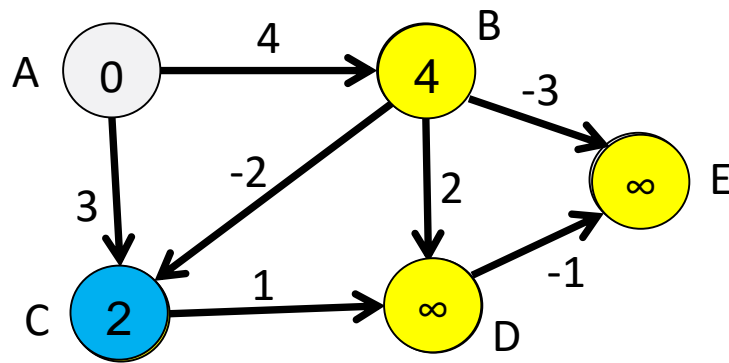


Düğüm	M	Ata
A	0	-
B	4	A
C	3	A
D	∞	
E	∞	

1	2	3	4	5	6	7
(C, D)	(A, B)	(A, C)	(B, C)	(B, D)	(B, E)	(D, E)

Köşe gevşetme düzeni

Bellman-Ford Alg: Örnek

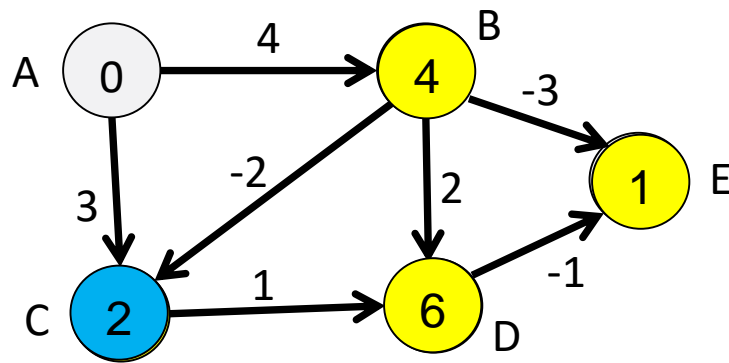


Düğüm	M	Ata
A	0	-
B	4	A
C	2	B
D	∞	
E	∞	

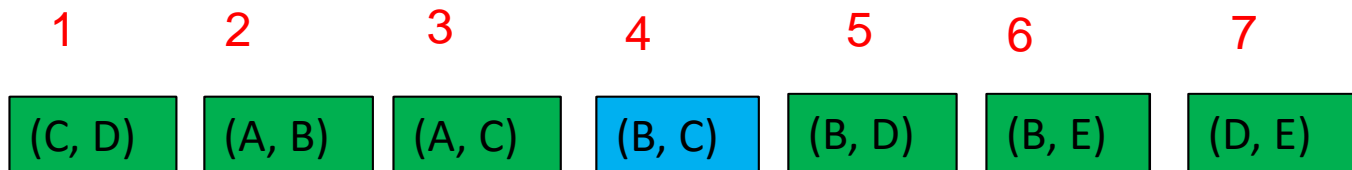
1	2	3	4	5	6	7
(C, D)	(A, B)	(A, C)	(B, C)	(B, D)	(B, E)	(D, E)

Köşe gevşetme düzeni

Bellman-Ford Alg: Örnek



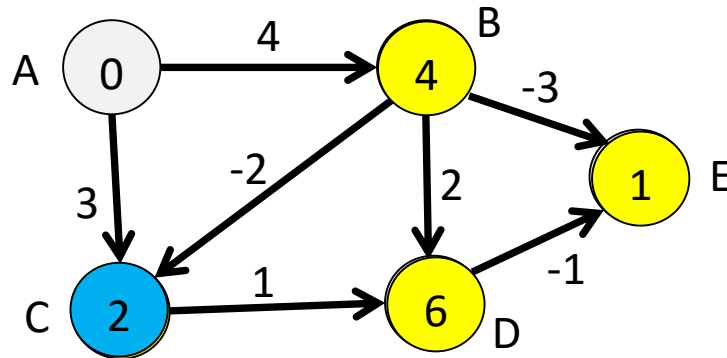
Düğüm	M	Ata
A	0	-
B	4	A
C	2	B
D	6	B
E	1	B



1.Tur tamamlandı

Bellman-Ford Alg: Örnek

- $B=AB$ 4, $\min(A,B)=0 \rightarrow 0+4=4$, $C=AC$ 3, $\min(A,C)=0 \rightarrow 0+3=3$
- $C=BC$ -2, $\min(B,C)=4 \rightarrow 4-2=2$, $D=BD$ 2, $\min(B,D)=4 \rightarrow 4+2=6$
- $E=BE$ -3, $\min(B,E)=4 \rightarrow 4-3=1$,



Düğüm	Ma.	Pred
A	0	-
B	4	A
C	2	B
D	6	B
E	1	B

(C, D)

(A, B)

(A, C)

(B, C)

(B, D)

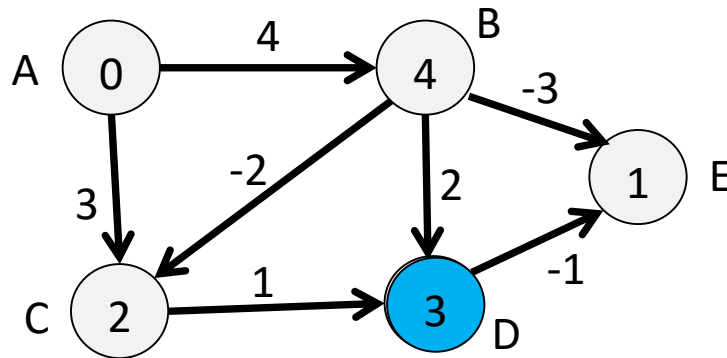
(B, E)

(D, E)

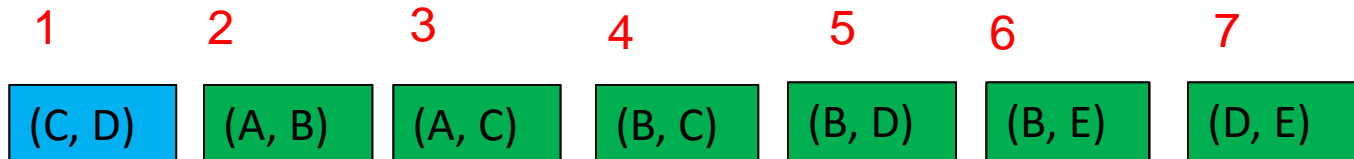
1.Tur sonu

Bellman-Ford Alg: Örnek

○ $D=CD$ 1, $\min(C,D)=2 \rightarrow 2+1=3$

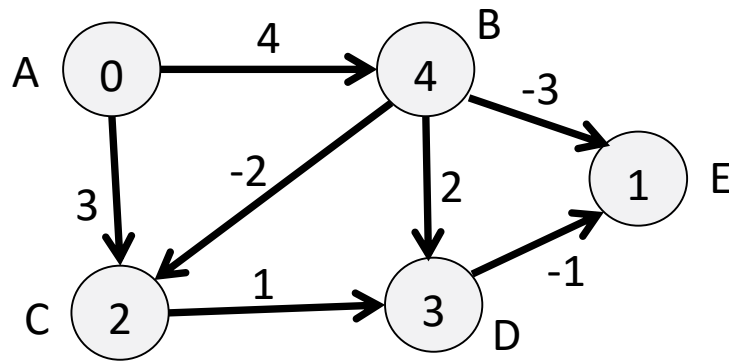


Düğüm	Ma	Pred
A	0	-
B	4	A
C	2	B
D	3	C
E	1	B



2. tur yineleme

Bellman-Ford Alg: Örnek



Düğüm	Ma	Pred
A	0	-
B	4	A
C	2	B
D	3	C
E	1	B

(C, D)

(A, B)

(A, C)

(B, C)

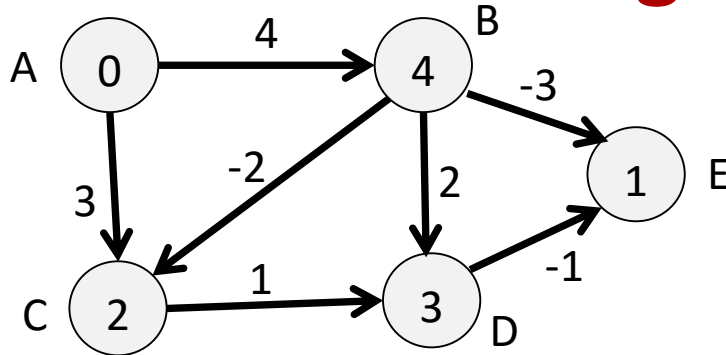
(B, D)

(B, E)

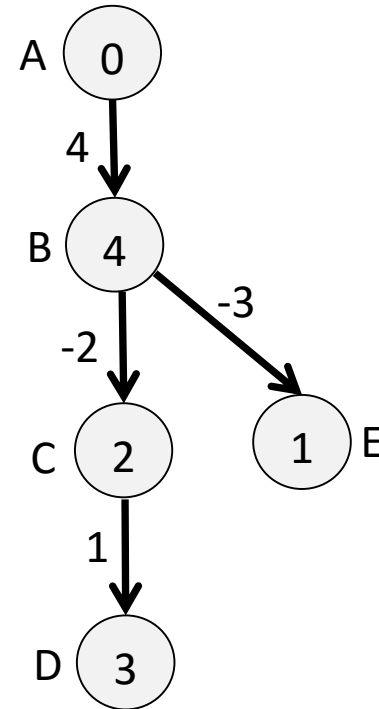
(D, E)

Üçüncü & Dördüncü Yineleme

Bellman-Ford Alg: Sonuç



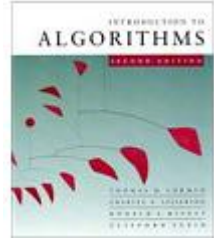
Düğüm	Ma	Pred
A	0	-
B	4	A
C	2	B
D	3	C
E	1	B



En kısa yollar III

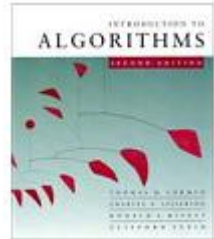
Tüm-ikili en kısa yollar All-Pairs Shortest Paths

- Tüm-ikili en kısa yollar
- Matris-çarpımı algoritması
- Floyd-Warshall algoritması
- Johnson algoritması



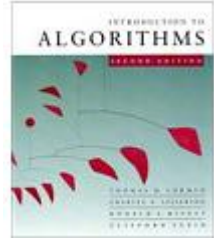
Hatırlatma

- Şu ana kadar tek-kaynaklı-en kısa-yollar üzerinde kaynak bir köşeden diğer köşelerin her birine en kısa yolu bulmak için geliştirilen algoritmalarından bahsedildi.
- Ağırlıksız durumda ve bütün kenar ağırlıkları bir olan graph için **BFS** (enine arama) uygulandı. Bu yapıda çalışma zamanı köşelerin sayıları ile kenarların sayılarının toplamından oluştuğundan doğrusal –zaman (**$O(V+E)$**) 'a mal olur.
- İkinci en kolay durum ise negatif olmayan kenar ağırlıkları yani **Dijkstra** algoritması.
- Eğer iyi bir “fibonacci heap structure” yani yığın yapısı kullanılırsa, yaklaşık doğrusal zamana mal olmakta, yani **$O(E+V \log V)$** .



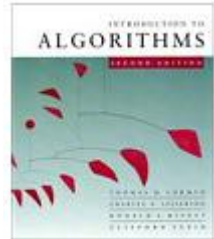
Hatırlatma

- Ayrıca negatif ağırlıklar için geliştirilmiş genel ağırlıklar için **Bellman-Ford** algoritmasından bahsedildi ve çalışma zamanı $O(VE)$ 'ye mal olmaktaydı(bu biraz daha kötü)
- Eğer log faktörlerini hesaba alınmazsa, E 'nin V düzeyinde olduğu, seyrek halde (komşuluk listesi) **Dijkstra**, doğrusal zamanlı ($O(V+E)$), **Bellman-Ford** ise (eğer bağlantılı bir grafik varsa) en az karaseldir ($O(V^2)$). Yoğun durumda (komşuluk matrisi) yani, E yaklaşık V^2 olduğunda, Dijkstra karesel $O(V^2)$, **Bellman-Ford** ise kübik $O(V^3)$ olur.



Hatırlatma

- **Dijkstra** ve **Bellman-Ford**, birbirlerinden bir **V** faktörü kadar farklıdırlar ve bu da oldukça kötüdür.
- Ancak, tek kaynaklı en kısa yollarda, negatif kenar ağırlıklarının olduğu durumlarda bildiklerimizin en iyisi **Bellman-Ford'** dur.
- Bu bağlamda daha önce **DAG-Directed Acyclic Graphs (Döngü Olmayan Yönlü Graflar)** örneğini de görmüştük ve orada Topolojik sıralama yapıyorduk.
- Demek oluyor ki, köşeler açısından bir düzenleme elde etmek için topolojik bir sıralama yapabiliriz. Sonra Bellman-Ford'u bir kere çalıştırırsak bir doğrusal zamanlı algoritma elde ederiz. DAG, ağırlıklarla dahi iyi bir uygulamanın nasıl yapılacağını bildiğimiz bir durum.



En kısa yollar

Tek-kaynaklı en kısa yollar

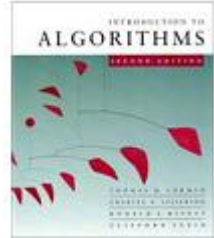
- Negatif olmayan kenar ağırlıkları
 - ♦ Dijkstra algoritması: $O(E + V \lg V)$
- Genel
 - ♦ Bellman-Ford: $O(VE)$
- DAG
 - ♦ Bellman-Ford' un bir turu: $O(V + E)$

Tüm-ikili en kısa yollar

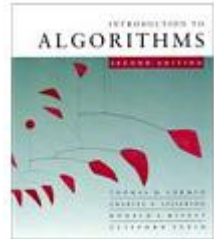
- Negatif olmayan kenar ağırlıkları
 - ♦ Dijkstra algoritması çarpı $|V|$: $O(VE + V^2 \lg V)$
- Genel
 - ♦ Bugün üç algoritma.

Tüm-ikili en kısa yollar

All-Pairs Shortest Paths



- Her iki köşe arasındaki en kısa yolun ağırlığını nasıl hesaplayabiliriz?
- Graph ağırlıksız ise, **BFS** algoritmasını kullanabiliriz.
- Çalışma zamanı $|V| * \text{BFS}$ olur yani, $O(V^2 + VE)$ olur. Yani en kısa yol ağırlığını hesaplayabilmemiz için kesinlikle, en az V^2 gibi bir süreye ihtiyacımız var. Çünkü çıkışın boyutu V^2 ; yani hesaplamanız gereken en kısa yol ağırlığı.
- Negatif olmayan kenar ağırlıkları durumunda, **Dijkstra'** yı, V defa uygulamak bunun koşma süresi de, gene $O(VE + V^2 \log V)$ olur. **BFS** yi uygulamak ile aynı (Eğer log faktörünü dikkate almazsanız, ağırlıklı terim budur).
- Bu durum, **Bellman-Ford** artı bir log faktörü zaman alıyor, ki eğer negatif olmayan kenar ağırlıklarınız varsa, bu sürede “tüm-ikili-en-kısa-yollar”ın hepsini hesaplayabiliriz.



Tüm-ikili en kısa yollar

- Negatif ağırlıkların olduğu durumda

Girdi: $G = (V, E)$ yönlü grafiğinde, $V = \{1, 2, \dots, n\}$ iken, $w : E \rightarrow \mathbb{R}$

kenar-ağırlık fonksiyonuyla

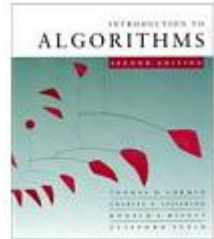
Çıktı:

Tüm $i, j \in V$ için $\delta(i, j)$ en kısa yol uzunluklarının $n \times n$ matrisi.

Fikir:

- Her köşeden Bellman-Ford' u bir tur çalıştır.
- Time (süre) = $O(V^2E)$.
- En kötü durumda yoğun grafik (n^2 kenarlı) $\Rightarrow \Theta(n^4)$ süre.

İlk deneme için iyi! Amacımız daha iyisini yapmak (Dinamik Programlama)



Dinamik programlama

Yönlü grafikte, $n \times n$ komşuluk matrisinin $A = (a_{ij})$ olduğunu düşünün,

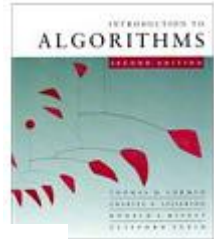
$d_{ij}^{(m)}$ = i 'den j 'ye en kısa yol ağırlığı-
en çok m sayıda kenarda kullanıldığında.

İddia:

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if(eğer) } i = j \text{ ise,} \\ \infty & \text{if(eğer) } i \neq j \text{ ise;} \end{cases}$$

ve for(için) $m = 1, 2, \dots, n - 1$,

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}.$$



İddianın Kanıtı

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$$

Relaxation! (gevşetme)

for $k \leftarrow 1$ to n

(için)

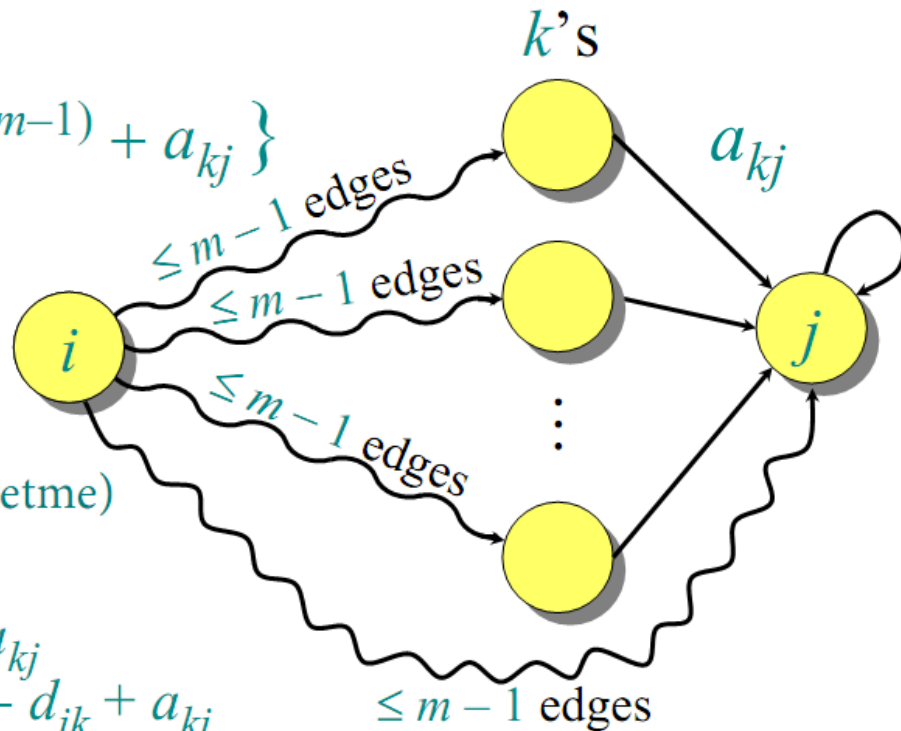
do if $d_{ij} > d_{ik} + a_{kj}$

(yap eğer)

then $d_{ij} \leftarrow d_{ik} + a_{kj}$

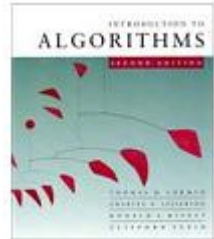
(sonra)

$O(n^4)$



Not: Negatif ağırlık çevrimi olmaması demek:

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \dots$$



Matris Çarpımı

C , A , ve B $n \times n$ matrislerse $C = A \cdot B$ yi hesapla:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}.$$

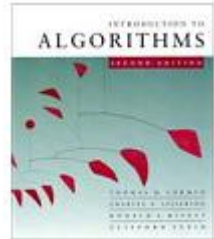
Time(süre) = $\Theta(n^3)$ standart algoritmayı kullanıyor.

“+” \rightarrow “min” ve “.” \rightarrow “+” ya eşlemlersek?

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}.$$

Böylece, $D^{(m)} = D^{(m-1)} \text{ “} \times \text{” } A.$ $= A^m$

$$\text{Özdeşlik matrisi} = I = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}^{(0)}). \quad = A^0$$



Matris Çarpımı

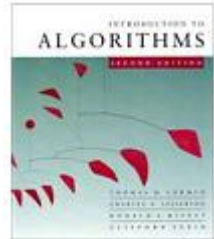
$(\min, +)$ çarpımı *çağrışımsal*dır, ve gerçekte sayılarla, *kapalı semiring(closed semiring)* olarak adlandırılan cebirsel bir yapı oluşturur.

Sonuçta bunu hesaplayabiliriz

$$\begin{aligned} D^{(1)} &= D^{(0)} \cdot A = A^1 \\ D^{(2)} &= D^{(1)} \cdot A = A^2 \\ &\vdots \\ D^{(n-1)} &= D^{(n-2)} \cdot A = A^{n-1}, \end{aligned}$$

yielding $D^{(n-1)} = (\delta(i, j))$ verir.

Time(süre) = $\Theta(n \cdot n^3) = \Theta(n^4)$. $n \times$ B-F' den daha iyi değil.



Geliştirilmiş matris çarpım algoritması

Tekrarlanan kareleme: $A^{2k} = A^k \times A^k$.

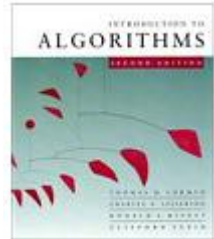
Hesaplayın: $A^2, A^4, \dots, A^{2^{\lceil \lg(n-1) \rceil}}$.

$O(\lg n)$ karelemeler

Not: $A^{n-1} = A^n = A^{n+1} = \dots$.

Time(süre) = $\Theta(n^3 \lg n)$.

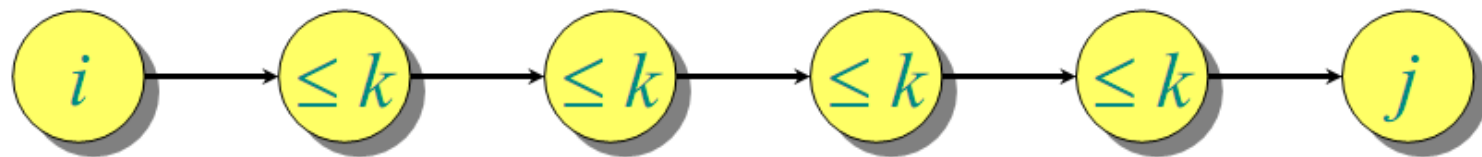
Negatif ağırlık çevrimlerini bulmak için, köşegendeki negatif değerleri $O(n)$ ek zamanında kontrol edin.



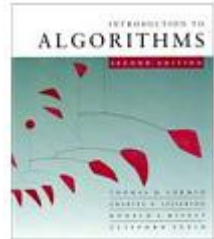
Tüm-ikili en kısa yollar

Floyd-Warshall algoritması

Tanımlama $c_{ij}^{(k)} = i$ den j 'ye, set $\{1, 2, \dots, k\}$ 'e deki ara köşeleri olan en kısa yolun ağırlığı.

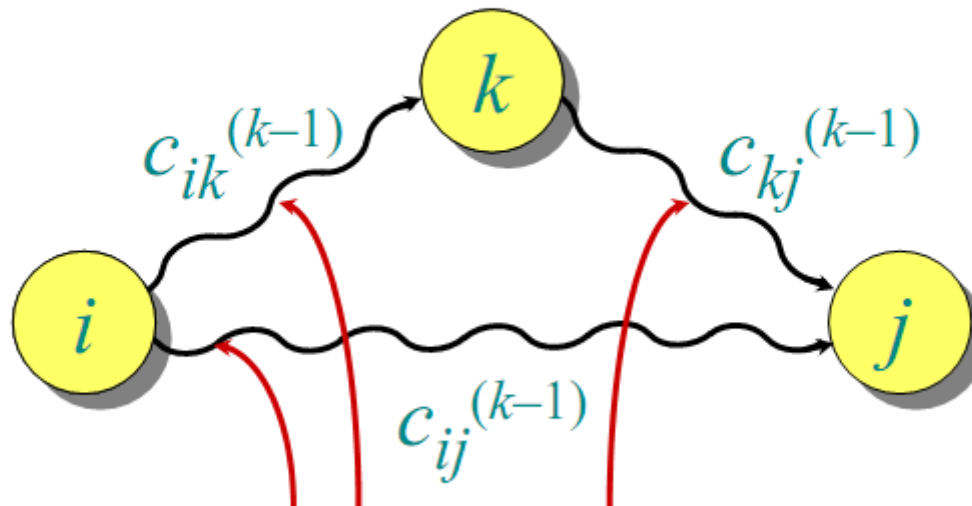


böylece, $\delta(i, j) = c_{ij}^{(n)}$. ve $c_{ij}^{(0)} = a_{ij}$.

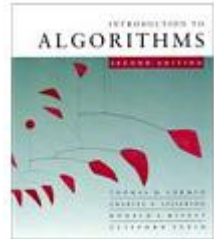


Floyd-Warshall yinelemesi

$$c_{ij}^{(k)} = \min_k \{c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\}$$



$\{1, 2, \dots, k\}$ ' deki ara köşeler



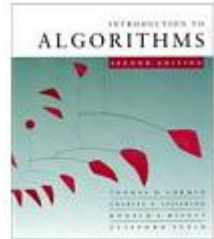
Floyd-Warshall için sözde kod

```

for  $k \leftarrow 1$  to  $n$ 
  (için) do for  $i \leftarrow 1$  to  $n$ 
    (için yap) do for  $j \leftarrow 1$  to  $n$ 
      (için yap) do if  $c_{ij} > c_{ik} + c_{kj}$ 
        (yap eğer) then  $c_{ij} \leftarrow c_{ik} + c_{kj}$ 
          (sonra)
    } Gevşetme
  
```

Notlar:

- Ekstra gevşetmelerin zararı olmayacağından üst simgeyi kullanmamak uygundur.
- $\Theta(n^3)$ zamanında çalışır.
- Kodlaması basittir.
- Pratikte verimlidir.



Bir yönlendirilmiş grafiğin geçişli kapanışı (transitive closure)

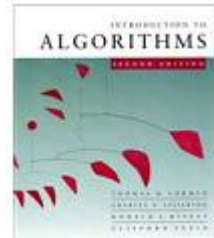
Hesaplayın $t_{ij} = \begin{cases} 1 & i \text{ den } j \text{ ye bir yol varsa,} \\ 0 & \text{diğer durumda.} \end{cases}$

Fikir: Floyd-Warshall' ı $(\min, +)$ yerine (\vee, \wedge) ile kullanın.

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}).$$

Time(süre) = $\Theta(n^3)$.

Floyd-Warshall Algoritması



FLOYD-WARSHALL(W)

$$W = \text{matrix of weights} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

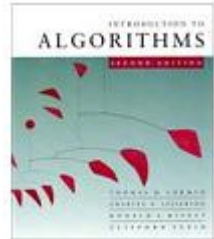
```

1   $n = W.rows$ 
2   $D^{(0)} = W$ 
3  for  $k = 1$  to  $n$ 
4      let  $D^{(k)} = (d_{ij}^{(k)})$  be a new  $n \times n$  matrix
5      for  $i = 1$  to  $n$ 
6          for  $j = 1$  to  $n$ 
7               $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ 
8  return  $D^{(n)}$ 

```

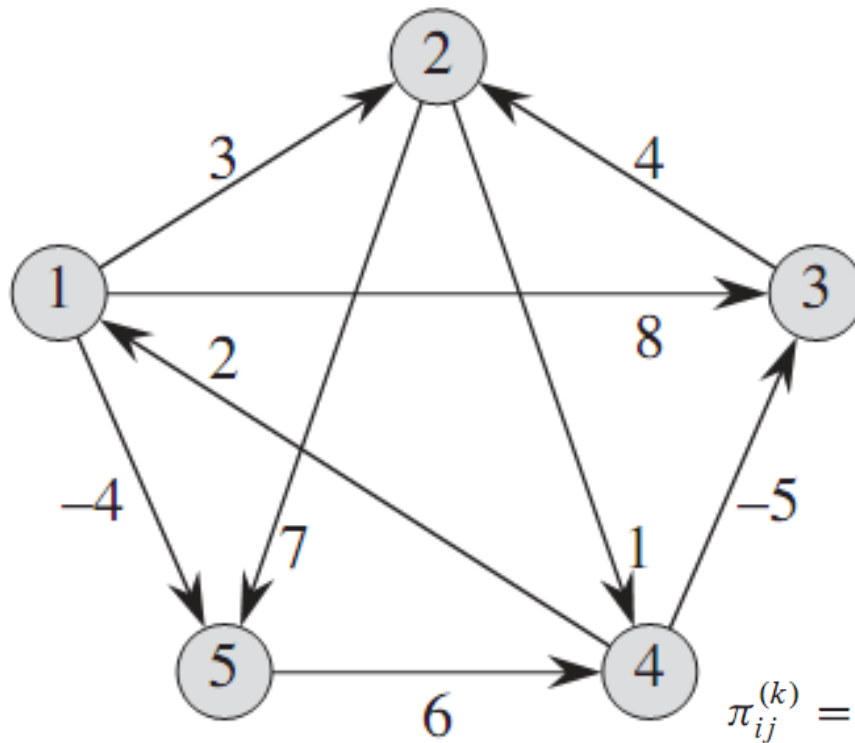
Atasını hesaplama

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases} \quad \pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$



Floyd-Warshall Örnek

Kenar ağırlıkları



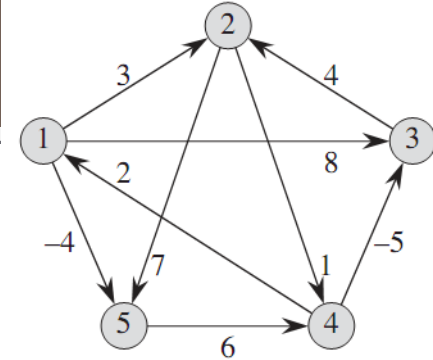
$$W = \text{matrix of weights} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

İkili en kısa yolları
hesaplama

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

Gidilecek düğümün
atasını hesaplama

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$



$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

Floyd-Warshall Örnek

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

Floyd-Warshall Örnek

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

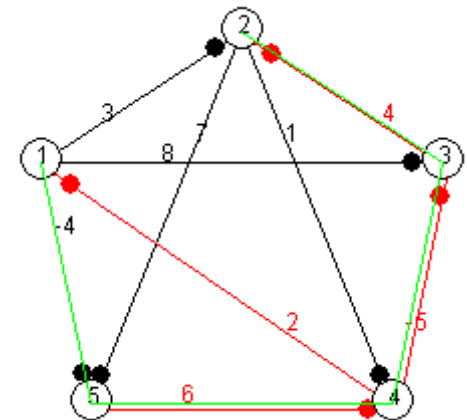
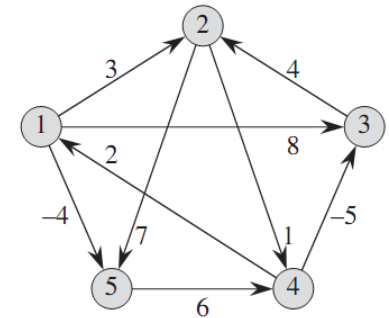
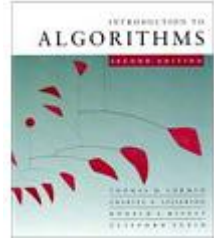


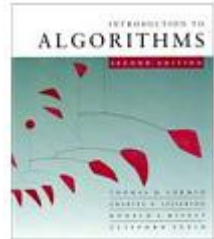
Figure 25.4 The sequence of matrices $D^{(k)}$ and $\Pi^{(k)}$ computed by the Floyd-Warshall algorithm

1.Düğüm için: 5'in atası 1, 4'ün atası 5, 3'ün atası 4, 2'nin atası da 3 tür.
 1-5=-4, 5-4 =-4+6=2, 4-3=-5+2=-3, 3-2=-3+4=1, Yol: 1-5-4-3-2

Johnson algoritması



- 1977 yılında Donald B. Johnson tarafından geliştirilmiştir. **Bellman Ford, Reweighting** ve **Dijkstra Algoritması** tabanlı, All pairs problemini çözmek için kullanılan bir algoritmadır.
- Sparse(dağınık) ve directed(yönlü) graflar için kullanılan güzel bir çözüm yoludur.
- Bağlantıların negatif olmasına izin vermektedir ve bu en önemli özelliğidir. Negatif bağlantıları reweighting yöntemiyle işlem sırasında ağırlıkları yeniden hesaplayarak pozitif ağırlıklara güncellemektedir. Bu yönüyle Floyd Warshall'a benzemektedir fakat $O(V^3)$ olan çalışma süresi Johnson Algoritması'nın tercih edilmesine neden olur.
- Ayrıca Floyd Marshall daha sık graflarda tercih edilirken, Johnson seyrek graflarda daha iyidir. Ağırlıkların pozitif olması durumunda Dijkstra'nın kullanılması daha iyi performans sağlar.



Grafik yeniden ağırlıklandırması

Johnson algoritması

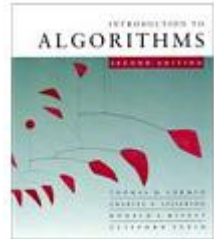
Teorem. $h : V \rightarrow \mathbb{R}$, fonksiyonu verilmiş, her $(u, v) \in E$ kenarını $w_h(u, v) = w(u, v) + h(u) - h(v)$ ile yeniden ağırlıklandırın. Bu durumda, her iki köşe arasındaki bütün yollar aynı miktarda yeniden ağırlıklandırılır.

Kanıt. $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$, G 'de bir yol olsun.

$$\begin{aligned}
 w_h(p) &= \sum_{i=1}^{k-1} w_h(v_i, v_{i+1}) \\
 &= \sum_{i=1}^{k-1} (w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1})) \\
 &= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) + h(v_1) - h(v_k) \\
 &= w(p) + h(v_1) - h(v_k).
 \end{aligned}$$

**Aynı
miktar!**





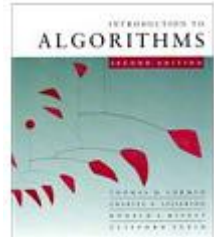
Yeniden ağırlıklandırılan grafiklerde en kısa yollar

D.Sonuç. $\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$. □

Fikir: $h : V \rightarrow \mathbb{R}$ fonksiyonunu bulun:
Tüm $(u, v) \in E'$ ler için $w_h(u, v) \geq 0$ olduğunda.

Sonra da yeniden ağırlıklandırılmış grafikte,
her köşeden Dijkstra'nın algoritmasını çalıştırın.

NOT: $w_h(u, v) \geq 0$ iff(eğer ve sadece eğer)
 $h(v) - h(u) \leq w(u, v)$.



Johnson algoritması

1. Şu fonksiyonu bulun $h : V \rightarrow \mathbb{R}$:

Tüm $(u, v) \in E'$ ler için $wh(u, v) \geq 0$ üzerinde Bellman-Ford' u çalıştırın $h(v) - h(u) \leq w(u, v)$ fark kısıtlarını çözün veya bir negatif ağırlık çevrimi varsa saptayın.

- Time(süre) = $O(VE)$.

2. Dijkstra'nın algoritmasını w_h' yi kullanarak, her köşeden $(u \in V)$, $\delta_h(u, v)$ ' hesaplayın (tüm $v \in V$ için).

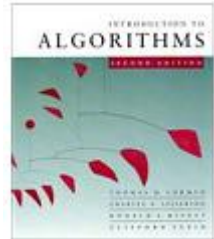
- Time(süre) = $O(VE + V^2 \lg V)$.

3. Her $(u, v) \in V \times V$ için,

$\delta(u, v) = \delta_h(u, v) - h(u) + h(v)$ hesaplayın.

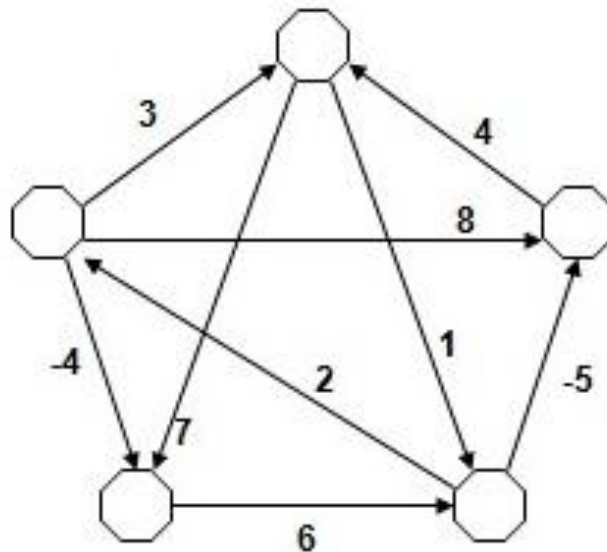
- Time(süre) = $O(V^2)$.

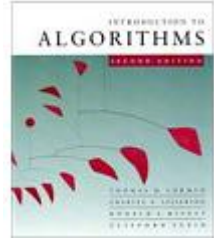
Toplam süre = $O(VE + V^2 \lg V)$.



Johnson algoritması

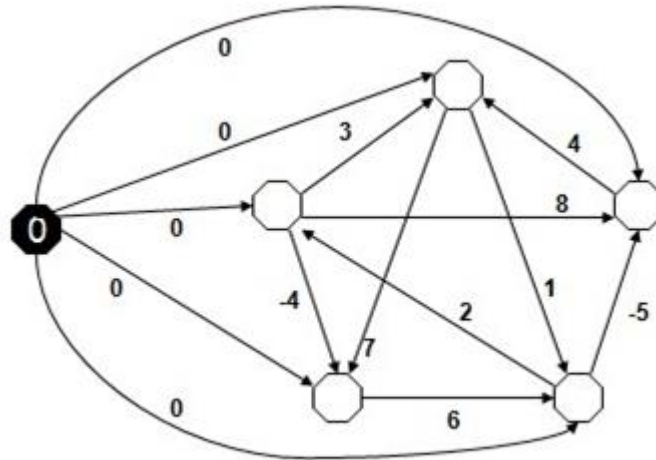
- Örnek: Algoritmanın adımlarını aşağıdaki graf üzerinde açıklayalım.

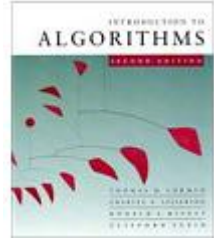




Johnson algoritması

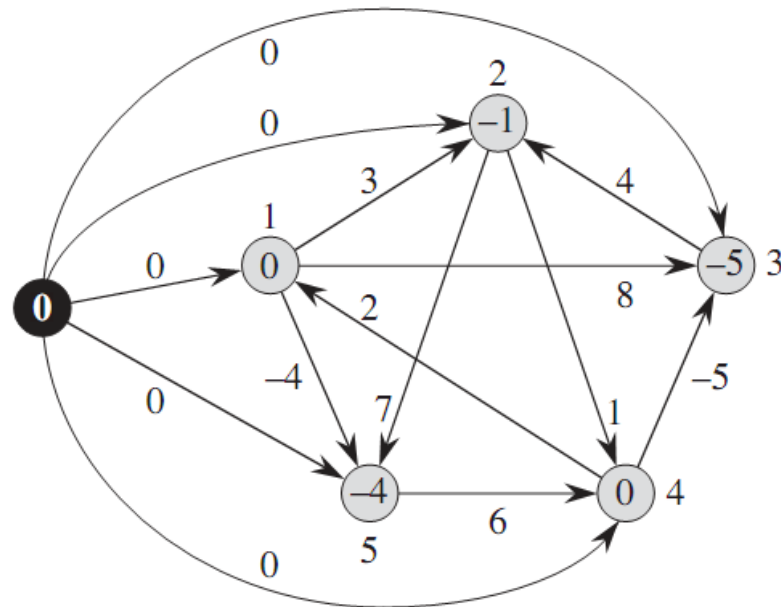
- **Adım 1:** Öncelikle grafa aşağıda görüldüğü üzere yeni bir düğüm ve bu düğümden grafa bulunan tüm düğümlere bağlantılar eklenir. Bu düğümün ve bağlantılarının ağırlığı sıfır olarak belirlenir.

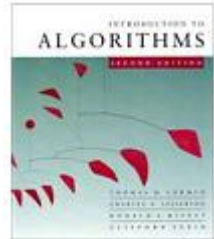




Johnson algoritması

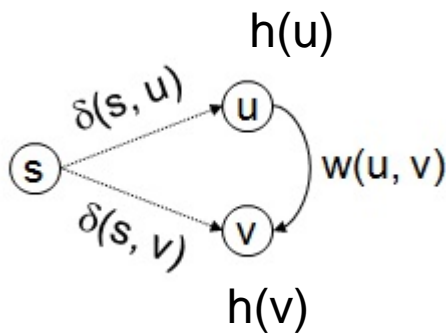
- Adım 2 : Her düğüm için Bellman-Ford Algoritması bir kez çalıştırılır ve düğümlerin ağırlıkları belirlenir. Aşağıda görüldüğü üzere her düğümün ağırlığı içerisine yazıldı.





Johnson algoritması

- Adım 3 :** Adım 4'te Dijkstra Algoritması kullanılacaktır. Bilindiği üzere Dijkstra Algoritması negatif bağlantı uzunluklarını kabul etmemektedir. Bu yüzden bu adımda ağırlıklar tekrar hesaplama yöntemiyle yenilecek ve negatif bağlantı kalmayacaktır. Yeniden hesaplama yöntemi şu şekildedir; $\hat{w}(u, v) = w(u, v) + d(s, u) - d(s, v)$

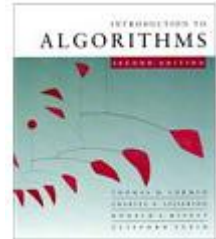


$$\underbrace{w(u, v) + \delta(s, u) - \delta(s, v)}_{\hat{w}(u, v)} \geq 0$$

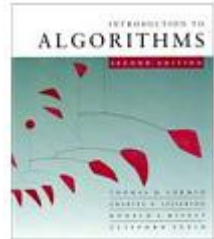
$$\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$$

$$\delta(u, v) = \delta_h(u, v) - h(u) + h(v)$$

Johnson algoritması



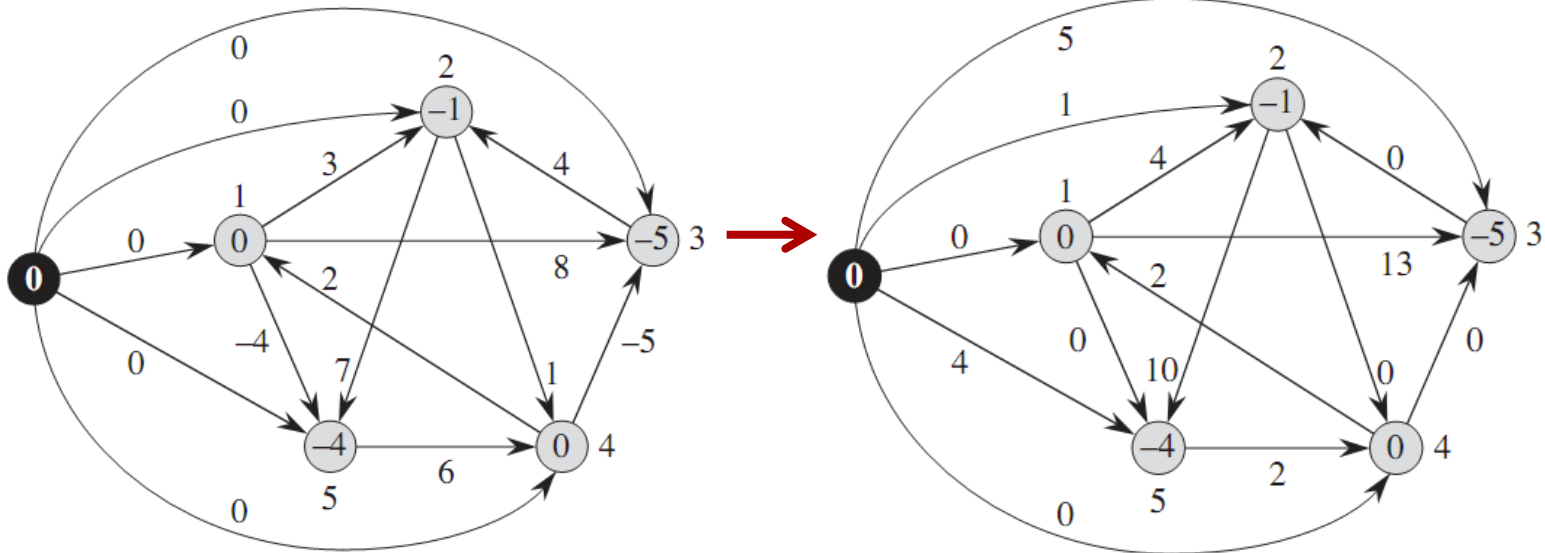
- Ağırlıkların yeniden hesaplanması Reweighting Teorem ile yapılır. Formülde yeni kenar ağırlığı eski kenar ağırlığı ile düğümün, algoritmanın birinci adımında eklenen yeni düğümüne olan ağırlığı ile toplanır. Bu değerden gidilecek olan düğümün S düğümüne olan ağırlığı çıkarılır ve yeni ağırlık bulunur. ($d_h[u,v] = d[u,v] + h[u] - h[v]$)
- Reweightin işlemi sonucunda shortest path değişmezken, ağırlıkların hepsi nonnegative olur. Burada akla şu soru gelebilir, ağırlıkları bu şekilde hesaplayacağımıza tüm düğümlere minimum bağlantı uzunluğunu ekleysek olmaz mı? Olmaz çünkü bu en kısa yolun değişmesine sebep olabilir.

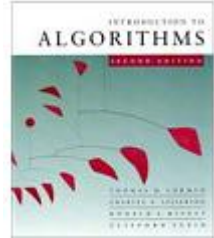


Johnson algoritması

- Aşağıda ağırlıkların güncellenmiş hali bulunmaktadır. Kenarların yeni ağırlıklarının bulunma işlemine örnek verecek olursak ; eski ağırlığı 8 olan kenarın (1-3) yeni ağırlığı yukarıda bahsettiğimiz yöntemle $8 + 0 - (-5) = 13$ 'e olarak hesaplanır. (1-2 kenarı için: $3 + 0 - (-1) = 4$

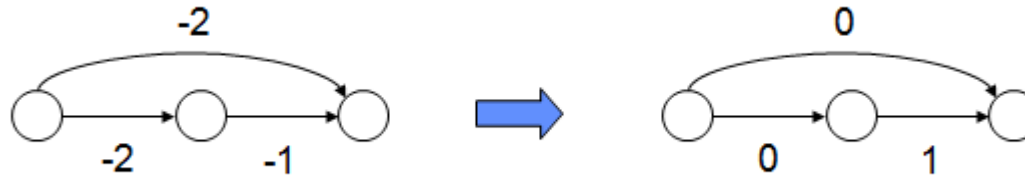
$$\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$$





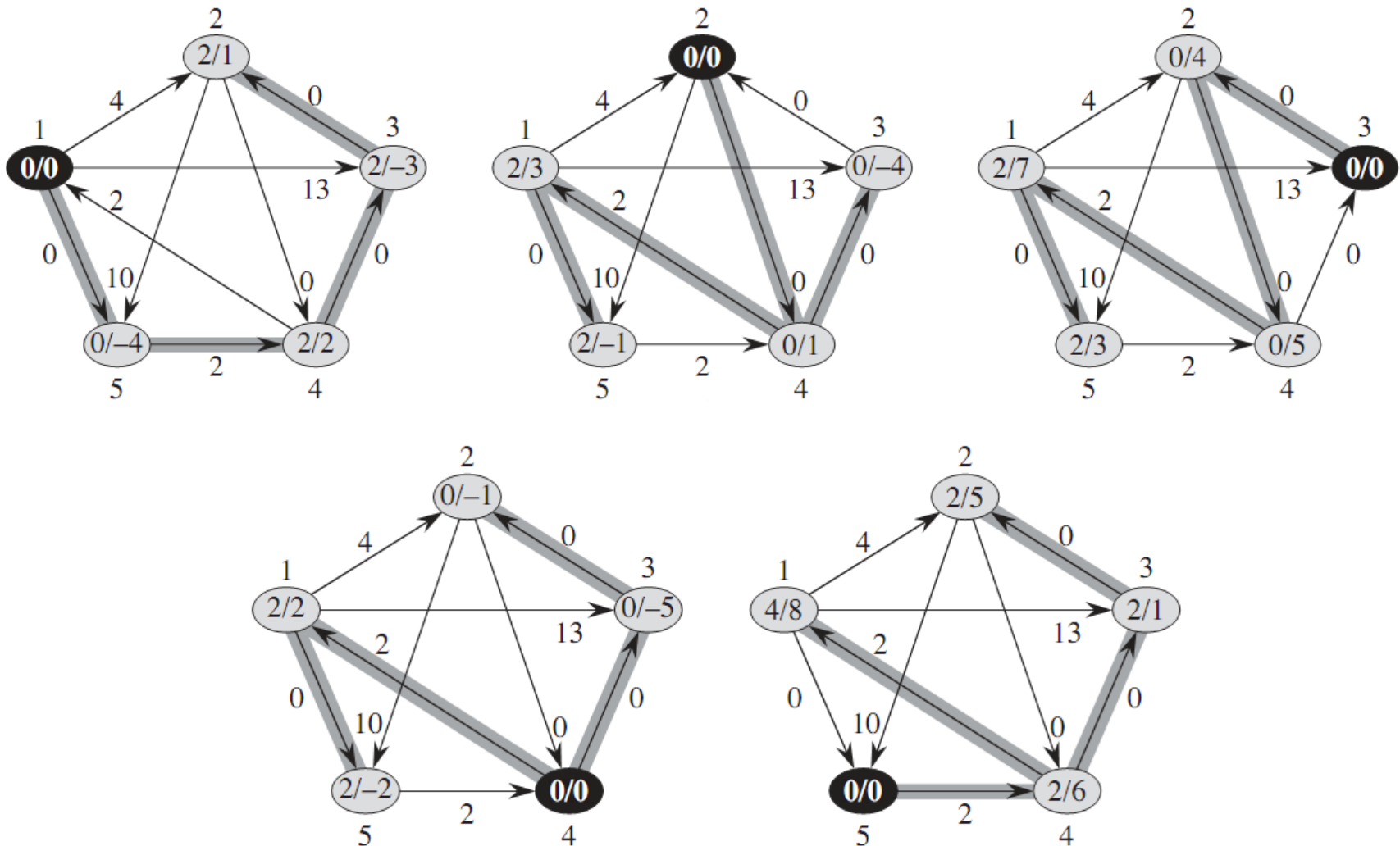
Johnson algoritması

- Aşağıdaki grafikta tüm düğümlere minimum düğüm ağırlığının eklenmesi durumunda ortaya çıkacak bozulma görülmektedir. Birinci durumda kısa yol alttaki iken, ikinci durumda üstteki oluyor. Bu metod görüldüğü üzere kısa yolun değişmesine sebep oluyor.



- **Adım 4 :** Birinci adımda eklenen düğüm graftan silinir ve geri kalan tüm düğümler için Dijkstra Algoritması uygulanır. Tüm çiftler arası en kısa yol bulunur. Aşağıda tüm düğümler için ağırlıkların Dijkstra Algoritması'yla yeniden hesaplanması görülmektedir.

Johnson algoritması



Johnson algoritması

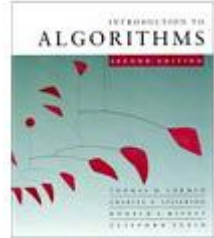
JOHNSON(G, w)

```

1  compute  $G'$ , where  $G'.V = G.V \cup \{s\}$ ,
    $G'.E = G.E \cup \{(s, v) : v \in G.V\}$ , and
    $w(s, v) = 0$  for all  $v \in G.V$ 
2  if BELLMAN-FORD( $G', w, s$ ) == FALSE
3      print “the input graph contains a negative-weight cycle”
4  else for each vertex  $v \in G'.V$ 
5      set  $h(v)$  to the value of  $\delta(s, v)$ 
       computed by the Bellman-Ford algorithm
6  for each edge  $(u, v) \in G'.E$ 
7       $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$ 
8  let  $D = (d_{uv})$  be a new  $n \times n$  matrix
9  for each vertex  $u \in G.V$ 
10     run DIJKSTRA( $G, \hat{w}, u$ ) to compute  $\hat{\delta}(u, v)$  for all  $v \in G.V$ 
11     for each vertex  $v \in G.V$ 
12          $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$ 
13  return  $D$ 

```

Johnson algoritması analizi

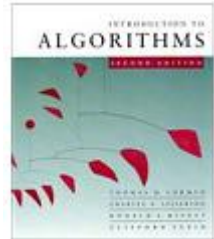


- Algoritmanın adımlarını kontrol edelim.
- İlk adımda yeni düğüm ekleniyor ve her düğümün ağırlığı Bellman Ford ile hesaplanıyor. Bu durumun getirdiği karmaşıklık $O(VE)$ 'dir.
- Daha sonra negatif kenarlardan kurtarmak için reweighting işlemi yapılıyor. Bu durumun getirdiği karmaşıklık $O(E)$ 'dir.
- Her düğüm için Dijkstra Algoritması'nın getirdiği karmaşıklık $O(V^2 \log V + VE \log V)$ 'dir.
- Bu durumda Johnson Algoritması'nın karmaşıklığı $O(V^2 \log V + VE \log V)$ olarak hesaplanır.
- (V: Düğümler Kümesi , E: Bağlantılar Kümesi)



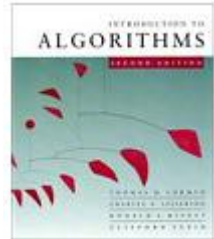
Örnek

Floyd-Warshall Algoritması



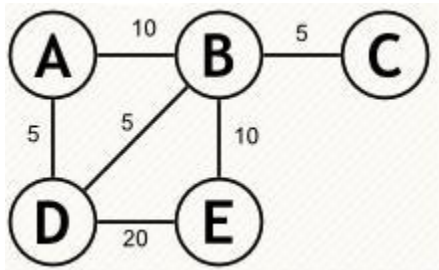
- for $i = 1$ to N
- for $j = 1$ to N
- if (i 'den j 'ye bir yol varsa)
- $\text{yol}[0][i][j] = i$ ile j arasındaki mesafe
- else
- $\text{yol}[0][i][j] = \text{sonsuz}$

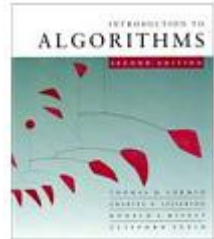
- for $k = 1$ to N
- for $i = 1$ to N
- for $j = 1$ to N
- $\text{yol}[k][i][j] = \min(\text{yol}[k-1][i][j], \text{yol}[k-1][i][k]$
- $+ \text{yol}[k-1][k][j])$



Floyd-Warshall Örnek

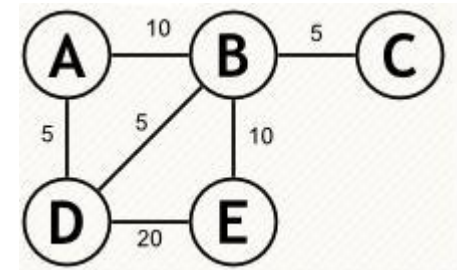
- Algoritmanın çalışmasını daha iyi anlayabilmek için aşağıdaki örnek üzerinden adım adım algoritmayı kullanarak en kısa yolu bulalım:



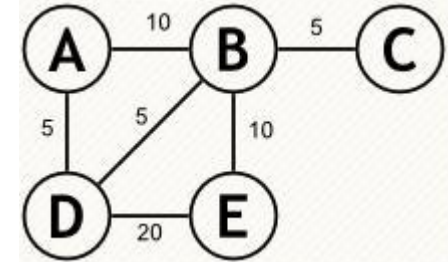


Floyd-Warshall Örnek

- Yukarıdaki şekil incelendiğinde A'dan E'ye giden birden çok yol bulunabilir:
- Yol 1: A \rightarrow B \rightarrow E 20
- Yol 2: A \rightarrow D \rightarrow E 25
- Yol 3: A \rightarrow B \rightarrow D \rightarrow E 35
- Yol 4: A \rightarrow D \rightarrow B \rightarrow E 20
- Yukarıdaki yollar çıkarıldıktan sonra en kisasının 20 uzunluğunda olduğu bulunabilir. Şimdi bu yollardan en kisasını Floyd-Warshall algoritmasının nasıl bulduğunu adım adım inceleyelim:

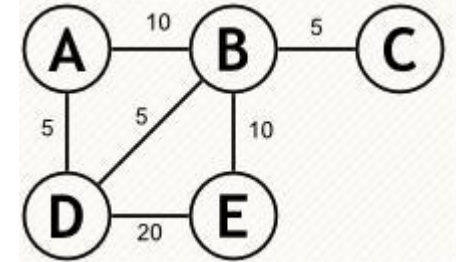


Floyd-Warshall Örnek



- 1. Adımda komşuluk listesine göre matris inşa edilir.
- Yukarıdaki şekilde doğrudan ilişkisi bulunan düğümler ve ağırlıkları aşağıda verilmiştir:
- A B C D E
- A 0 10 ∞ 5 ∞
- B 10 0 5 5 10
- C ∞ 5 0 ∞ ∞
- D 5 5 ∞ 0 20
- E ∞ 10 ∞ 20 0
- Yukarıdaki grafta doğrudan ilişkisi bulunmayan düğümlerin değerleri ∞ olarak gösterilmektedir. Diğer değerler doğrudan ağırlıkları göstermektedir.

Floyd-Warshall Örnek

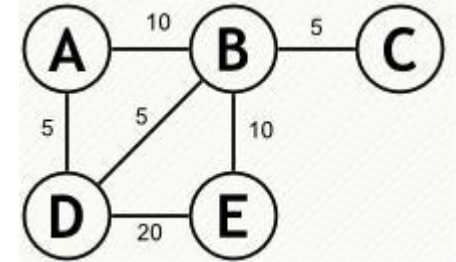


- Şimdi algoritmanın 2. adımına geçerek yolların tutulduğu bu matrisi adım adım güncelleyelim:

- A B C D E
- A 0 10 ∞ 5 ∞
- B 10 0 5 5 10
- C ∞ 5 0 ∞ ∞
- D 5 5 ∞ 0 20
- E ∞ 10 ∞ 20 0

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

Floyd-Warshall Örneği



- B üzerinden atlanarak ulaşılan düğümleri güncelleyelim
- A B C D E
- A 0 10 15 5 20
- B 10 0 5 5 10
- C 15 5 0 10 15
- D 5 5 10 0 15
- E 20 10 15 15 0

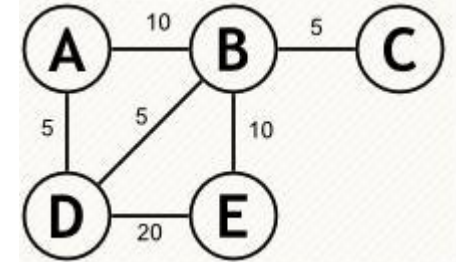
Floyd-Warshall Örnek

- C üzerinden atlanan düğümler:

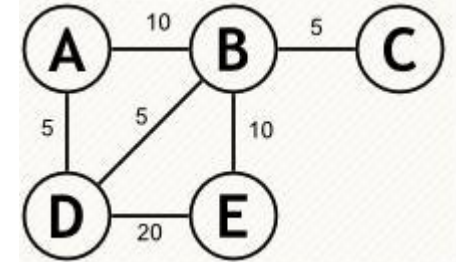
- A B C D E
- A 0 10 15 5 20
- B 10 0 5 5 10
- C 15 5 0 10 15
- D 5 5 10 0 15
- E 20 10 15 15 0

- D üzerinden atlanan düğümler:

- A B C D E
- A 0 10 15 5 20
- B 10 0 5 5 10
- C 15 5 0 10 15
- D 5 5 10 0 15
- E 20 10 15 15 0



Floyd-Warshall Örneği



- E üzerinden atlanan düğümler:
 - A B C D E
 - A 0 10 15 5 20
 - B 10 0 5 5 10
 - C 15 5 0 10 15
 - D 5 5 10 0 15
 - E 20 10 15 15 0
- Yukarıda son elde edilen bu matriste görüldüğü üzere herhangi bir düğümden diğer bütün düğümlere giden en kısa yollar çıkarılmıştır. Örneğin A düğümünden E'ye 20 uzunluğunda veya C düğümünden D'ye 10 uzunluğunda yolla gidilebilir.
- Yukarıdaki matrislerde diyagona göre simetri bulunmasının sebebi grafın yönsüz graf (undirected graph) olmasıdır. Şayet graf yönlü graf (directed graph) olsaydı bu simetri bozulurdu (tabi yönlerin ağırlıklarının aynı olmaması durumunda).