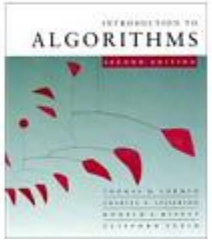
14.Hafta
Tüm-ikili en kısa yollar

- **Tüm-ikili en kısa yollar (All-Pairs Shortest Paths)**
- Matris-çarpımı algoritması
- Floyd-Warshall algoritması
- Johnson algoritması



Hatırlatma

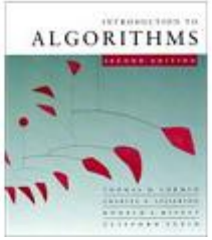
- Şu ana kadar tek-kaynaklı-en kısa-yollar üzerinde kaynak bir köşeden diğer köşelerin her birine en kısa yolu bulmak için geliştirilen algoritmalardan bahsedildi.
- Ağırlıksız durumda ve bütün kenar ağırlıkları bir olan graph için **BFS** (enine arama) uygulandı. Bu yapıda çalışma zamanı köşelerin sayıları 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+Vlog V)**.

Hatırlatma

- Ayrıca negatif ağırlıklar için geliştirişmiş 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 Acylic 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.



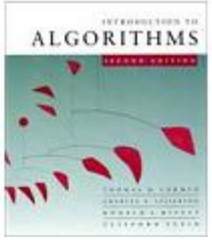
En kısa yollar

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

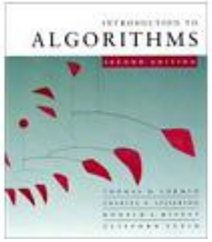
- Negatif olmayan kenar ağırlıkları
  - Dijkstra algoritması çarpı $|V|$: $O(VE + V^2 \lg V)$
- Genel
  - Bugün üç algoritma.



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|*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



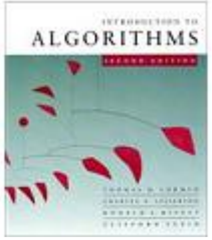
- Negatif ağırlıkların olduğu durumda

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

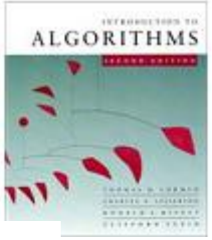
$d_{ij}^{(m)}$ $= i$ ' den j 'ye en kısa yol ağırlığı- en çok m sayıda kenarda kullanıldığında.

İddia:

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

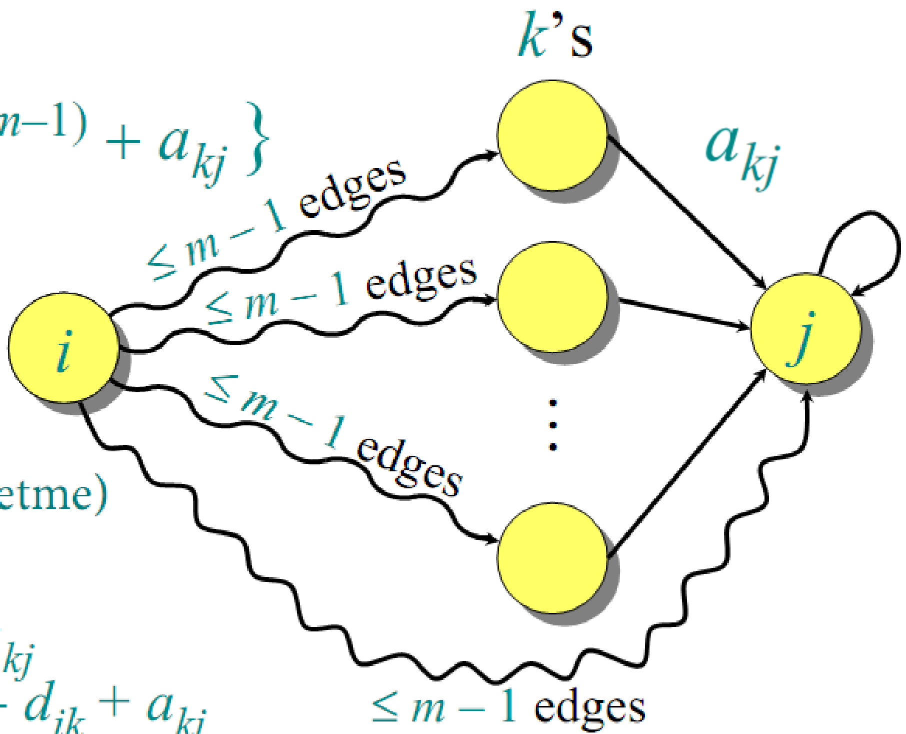
ve for(için) $m = 1, 2, \ldots, n - 1,$

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



İddianın Kanıtı

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



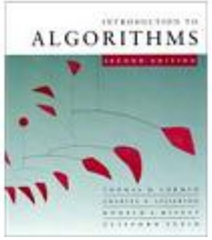
Relaxation! (gevşetme)

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:

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



Matris Çarpımı

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

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

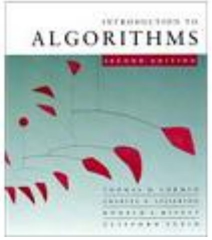
Time(süre) $= \Theta(n^3)$ standart algoritmayı kullanıyor.

"+" $\rightarrow$ "min"ve "·" $\rightarrow$ "+" ya eşlemlersek?

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

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

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



Matris Çarpımı

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

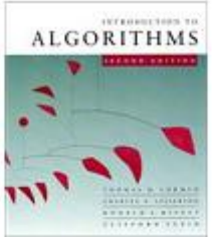
Sonuçta bunu hesaplayabiliriz

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

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

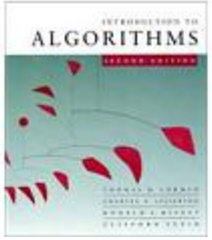
Hesaplayın: $\underbrace{A^2, A^4, \ldots, A^{2^{\lceil \lg(n-1) \rceil}}}_{O(\lg n)\text{karelemeler}}$.

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

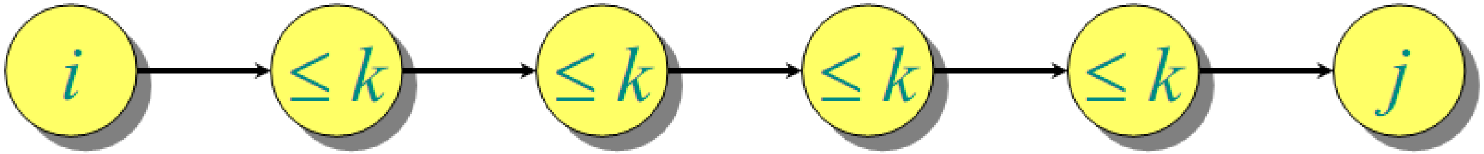
Time(süre) $= \Theta(n^3 \lg n)$.

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

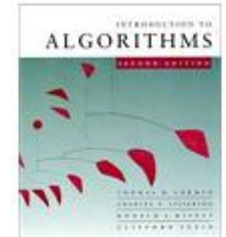
Tüm-ikili en kısa yollar
Floyd-Warshall algoritması



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

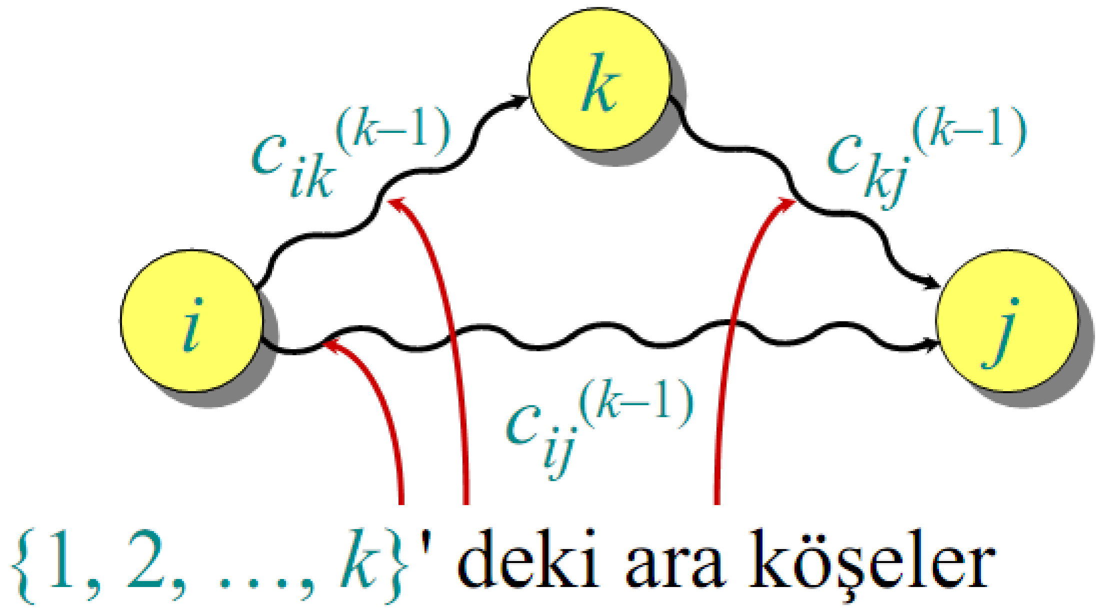


böylece, $\delta(i, j) = c_{ij}^{(n)}$.   ve $c_{ij}^{(0)} = a_{ij}$.



Floyd-Warshall yinelemesi

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



Floyd-Warshall için sözde kod

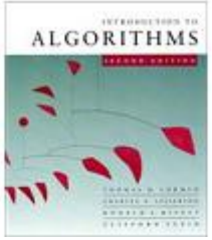
for $k \leftarrow 1$ **to** n
(için)
  do for $i \leftarrow 1$ **to** n
  (için yap)
    do for $j \leftarrow 1$ **to** n
    (için yap)
      do if $c_{ij} > c_{ik} + c_{kj}$
      (yap eğer)
        then $c_{ij} \leftarrow c_{ik} + c_{kj}$
        (sonra)

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)



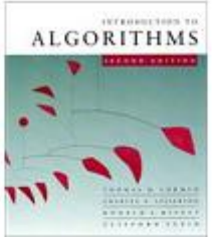
Hesaplayın $t_{ij} = \begin{cases} 1 & i \text{ den } j \text{ ye bir yol varsa,} \\ 0 & \text{diğer durumda.} \end{cases}$

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



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

FLOYD-WARSHALL(W)

```
n = W.rows
D^(0) = W
for k = 1 to n
    let D^(k) = (d_ij^(k)) be a new n × n matrix
    for i = 1 to n
        for j = 1 to n
            d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```

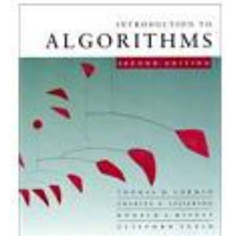
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 \,. \end{cases}$$

Constructing a shortest path

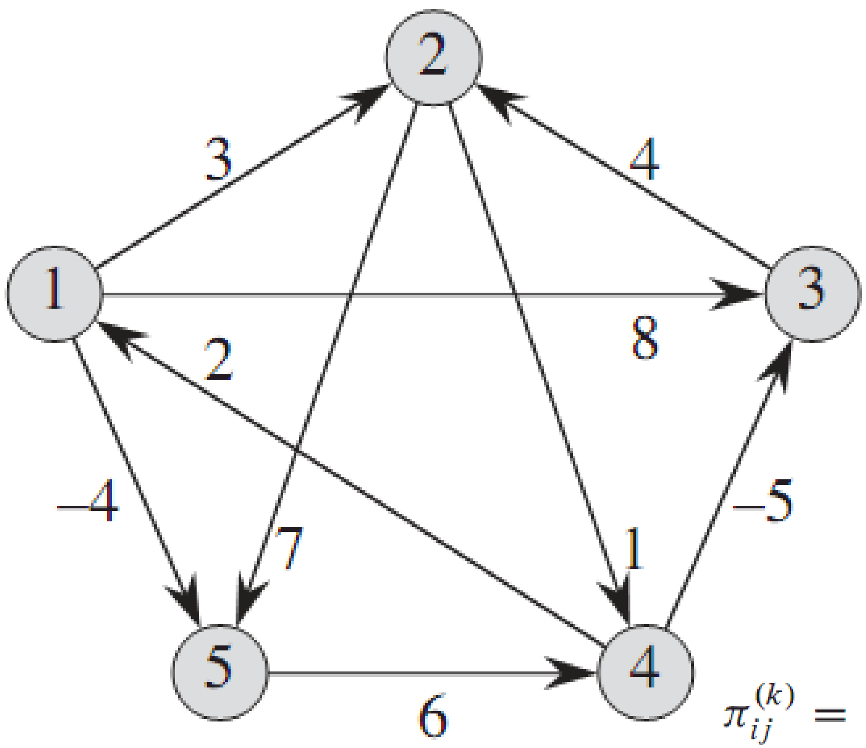
$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty \,, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \,. \end{cases}$$

Atasını hesaplama

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



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

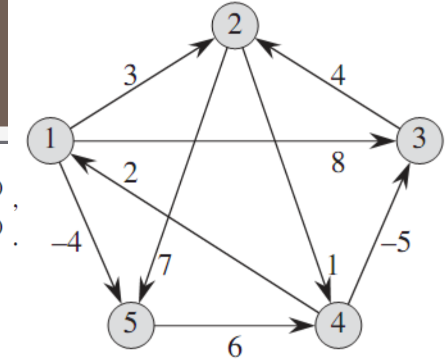
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0\,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1\,. \end{cases}$$

Gidilecek düğümün atasını hesaplama

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

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

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



Floyd-Warshall Örnek

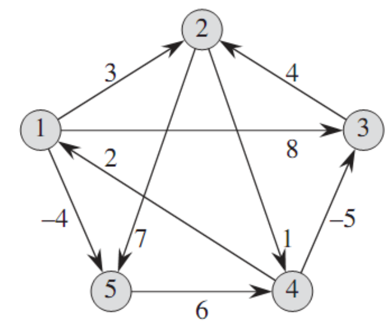
$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL}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

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

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

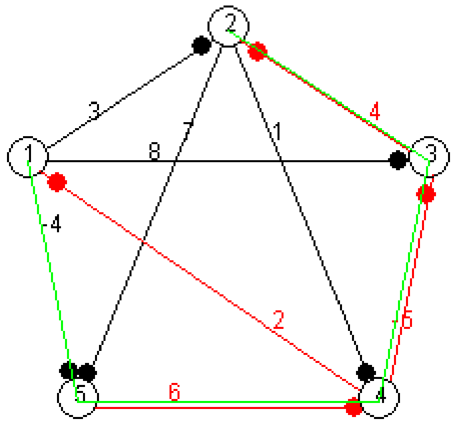
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0\,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1\,. \end{cases}$$

Floyd-Warshall Örnek



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

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

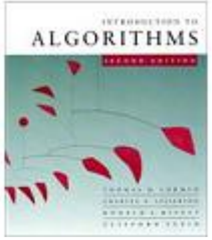


$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} &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

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

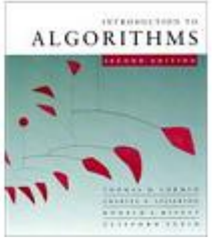
1.Düğüm için: 5'in atası 1, 4'ün atası 5, 3'ün atası 4, 2'nin atası da 3 tür.
1-5=-4, 5-4 =-4+6=2, 4-3=-5+2=-3, 3-2=-3+4=1, Yol: 1-5-4-3-2

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 Warshall 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



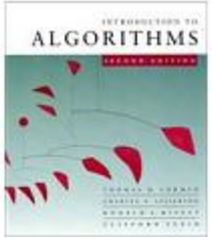
Teorem. $h : V \to \mathbb{R}$, fonksiyonu verilmiş, her $(u, v) \in E$ kenarını $w_h(u, v) = w(u, v) + h(u) - h(v)$ ile yeniden ağırlıklandırın. Bu durumda, her iki köşe arasındaki bütün yollar aynı miktarda yeniden ağırlıklandırılır.

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

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

Aynı miktar!

Yeniden ağırlıklandırılan grafiklerde en kısa yollar

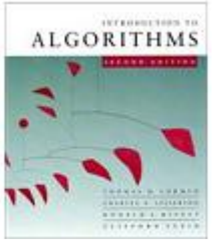


D.Sonuç. $\delta_h(u, v) = \delta(u, v) + h(u) - h(v).$

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

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).

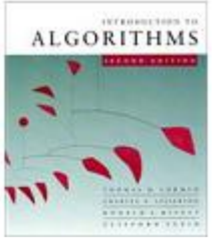
- Time(süre) = $O(VE + V^2 \lg V)$.

3. Her $(u, v) \in V \times V$ için,

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

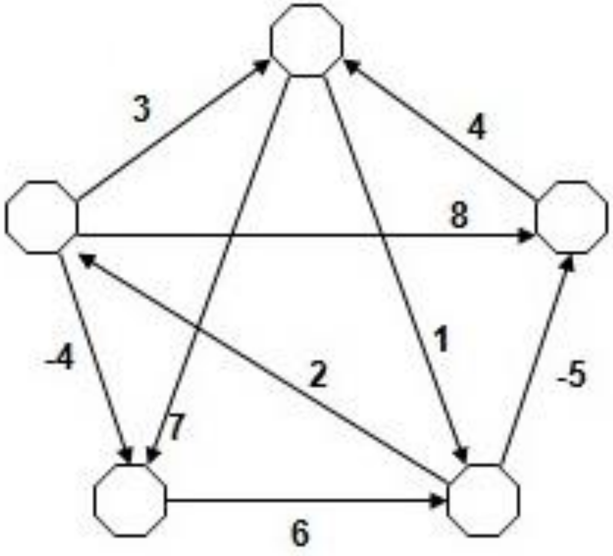
- Time(süre) = $O(V^2)$.

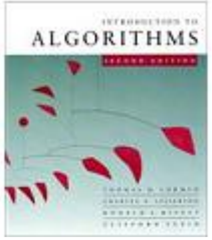
Toplam süre = $O(VE + V^2 \lg V)$.



Johnson algoritması

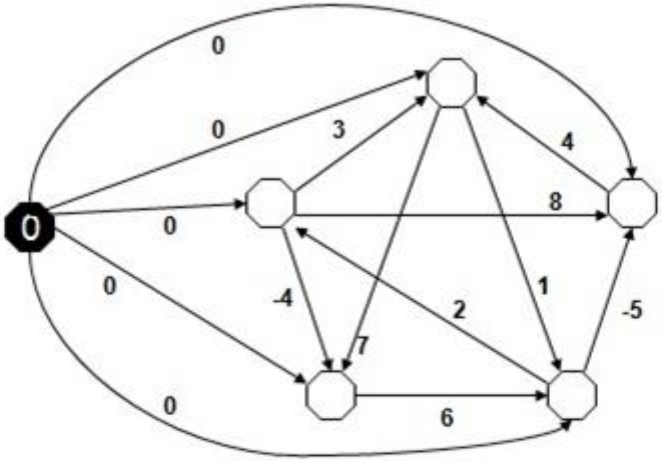
- Örnek: Algoritmanın adımlarını aşağıdaki graf üzerinde açıklayalım.

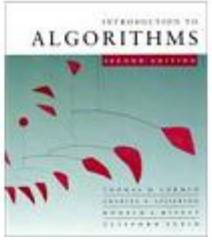




Johnson algoritması

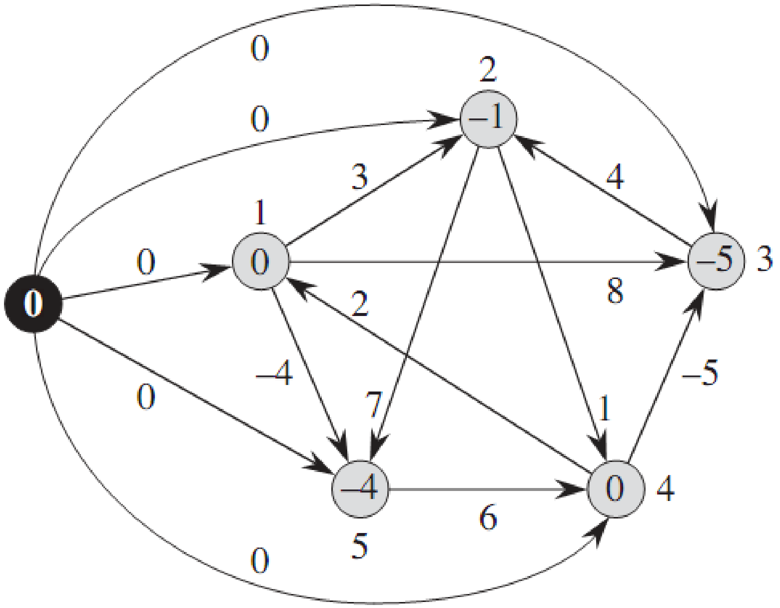
- **Adım 1:** Öncelikle grafa aşağıda görüldüğü üzere yeni bir düğüm ve bu düğümden grafta bulunan tüm düğümlere bağlantılar eklenir. Bu düğümün ve bağlantılarının ağırlığı sıfır olarak belirlenir.

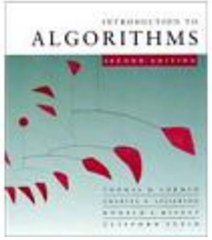




Johnson algoritması

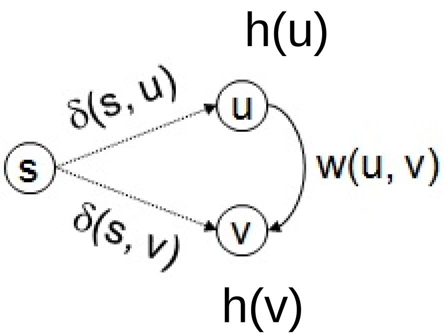
- **Adım 2 :** Her düğüm için Bellman-Ford Algoritması bir kez çalıştırılır ve düğümlerin ağırlıkları belirlenir. Aşağıda görüldüğü üzere her düğümün ağırlığı içerisine yazıldı.





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; **ŵ(u, v) = w(u, v) + d(s, u) - d(s, v)**

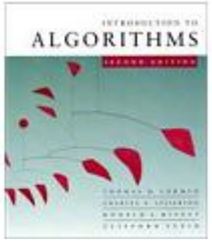


$$\underbrace{w(u,v)+\delta(s,u)-\delta(s,v)}_{\hat{w}(u,v)}\geq 0$$

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

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

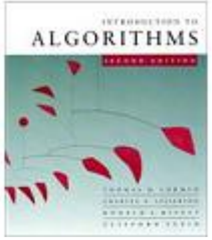
Johnson algoritması



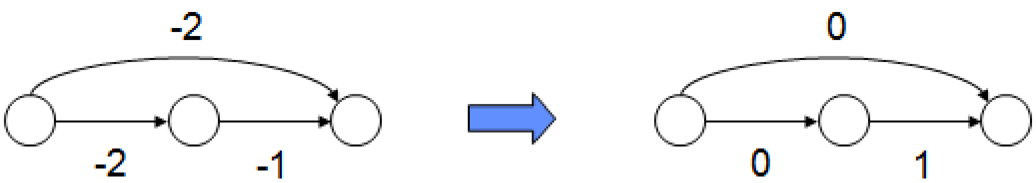
- Ağırlıkların yeniden hesaplanması Reweighting Teorem ile yapılır. Formülde yeni kenar ağırlığı eski kenar ağırlığı ile düğümün, algoritmanın birinci adımında eklenen yeni düğümüne olan ağırlığı ile toplanır. Bu değerden gidilecek olan düğümün S düğümüne olan ağırlığı çıkarılır ve yeni ağırlık bulunur. ($d_h[u,v]= d[u,v]+h[u]-h[v]$)

- Reweighting işlemi sonucunda shortest path değişmezken, ağırlıkların hepsi nonnegative olur. Burada akla şu soru gelebilir, ağırlıkları bu şekilde hesaplayacağımıza tüm düğümlere minimum bağlantı uzunluğunu eklesek olmaz mı? Olmaz çünkü bu en kısa yolun değişmesine sebep olabilir.

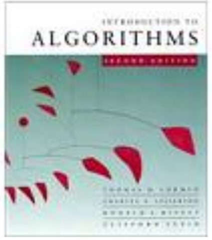
Johnson algoritması



- Aşağıdaki grafikta tüm düğümlere minimum düğüm ağırlığır 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.

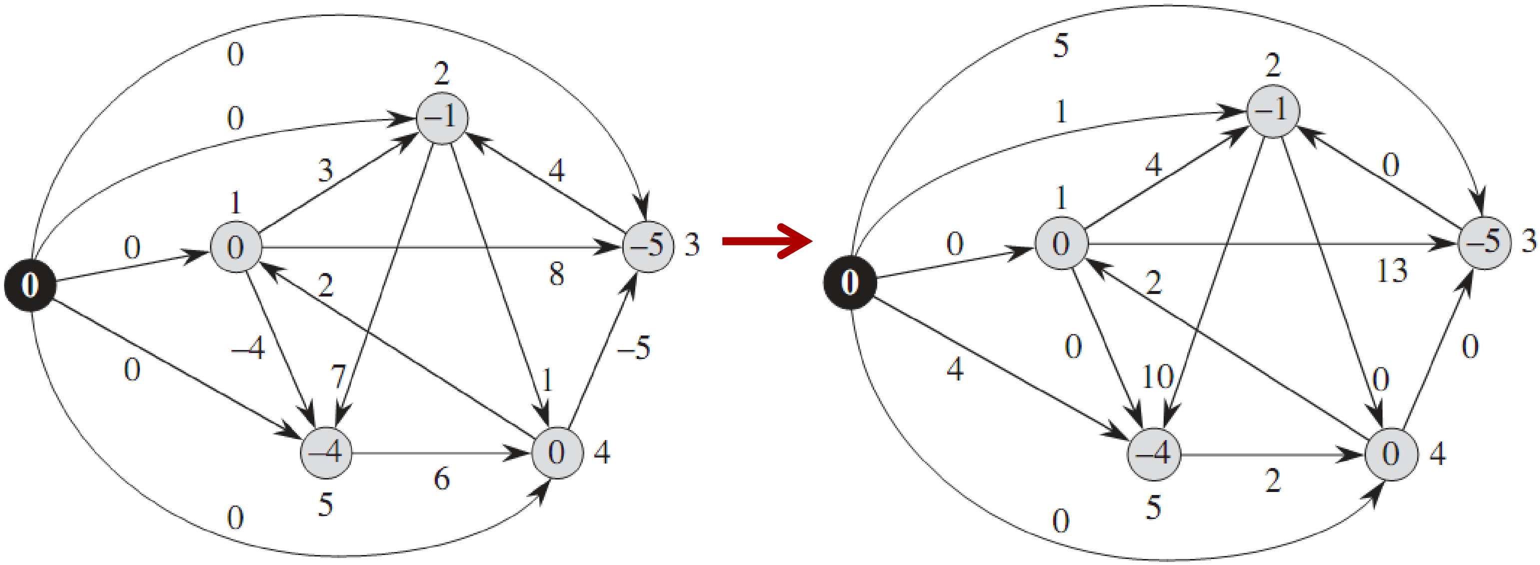


Johnson algoritması

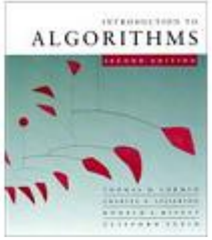


- Aşağıda ağırlıkların güncellenmiş hali bulunmaktadır. Kenarların yeni ağırlıklarının bulunma işlemine örnek verecek olursak ; eski ağırlığı 8 olan kenarın (1-3) yeni ağırlığı yukarıda bahsettiğimiz yöntemle 8 + 0 – (-5) = 13 ‘e olarak hesaplanır. (1-2 kenarı için:3+0-(-1)=4

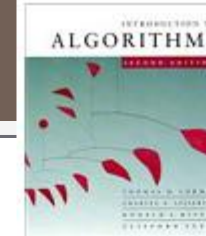
$$\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$$



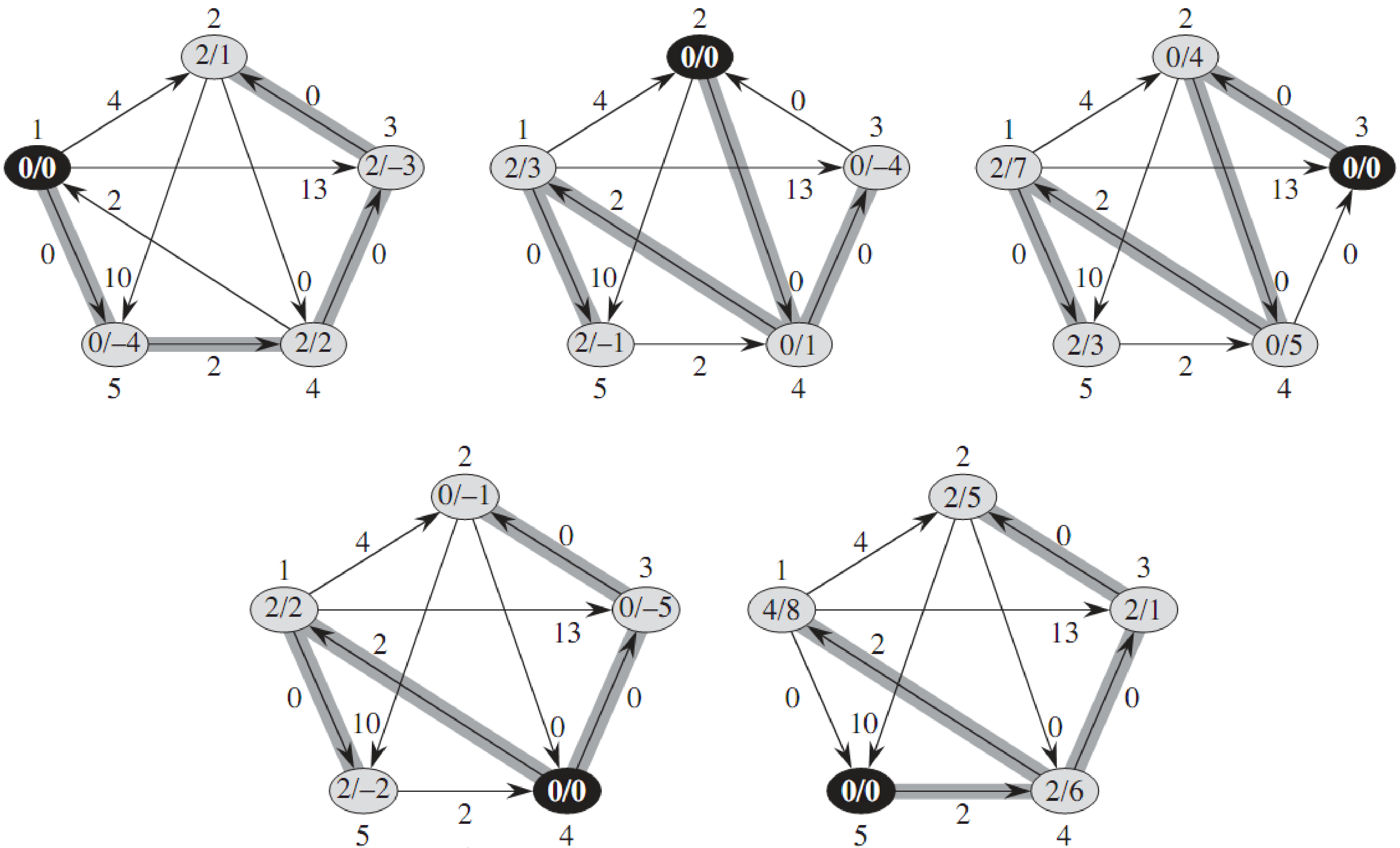
Johnson algoritması

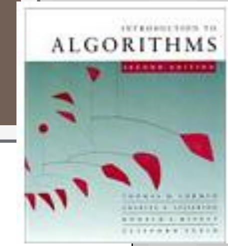


- **Adım 4 :** Birinci adımda eklenen düğüm graftan silinir ve geri kalan tüm düğümler için Dijkstra Algoritması uygulanır. Tüm çiftler arası en kısa yol bulunur. Aşağıda tüm düğümler için ağırlıkların Dijkstra Algoritması'yla yeniden hesaplanması görülmektedir.



Johnson algoritması





Johnson algoritması

JOHNSON(G, w)

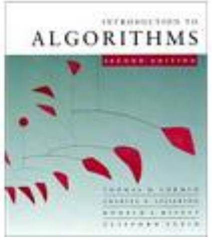
compute G', where $G'.V = G.V \cup \{s\}$,
  $G'.E = G.E \cup \{(s, v) : v \in G.V\}$, and
  $w(s, v) = 0$ for all $v \in G.V$
if BELLMAN-FORD(G', w, s) == FALSE
  print "the input graph contains a negative-weight cycle"
else for each vertex $v \in G'.V$
    set $h(v)$ to the value of $\delta(s, v)$
      computed by the Bellman-Ford algorithm
  for each edge $(u, v) \in G'.E$
    $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$
  let $D = (d_{uv})$ be a new $n \times n$ matrix
  for each vertex $u \in G.V$
    run DIJKSTRA$(G, \hat{w}, u)$ to compute $\hat{\delta}(u, v)$ for all $v \in G.V$
    for each vertex $v \in G.V$
      $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$
  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 Djikstra 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)

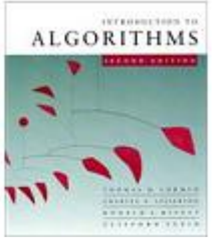
Örnek

Floyd-Warshall Algoritması



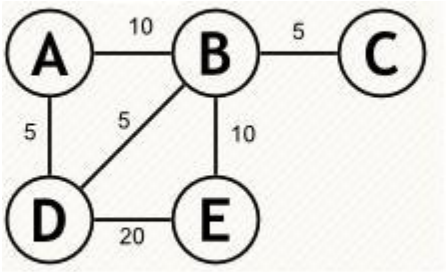
- for i = 1 to N
- for j = 1 to N
- if(i'den j'ye bir yol varsa)
- yol[0][i][j] = i ile j arasındaki mesafe
- else
- yol[0][i][j] = sonsuz

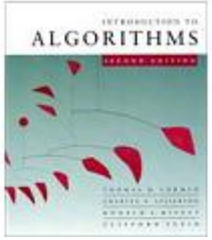
- for k = 1 to N
- for i = 1 to N
- for j = 1 to N
- yol[k][i][j] = min(yol[k-1][i][j], yol[k-1][i][k]
- + yol[k-1][k][j])



Floyd-Warshall Örnek

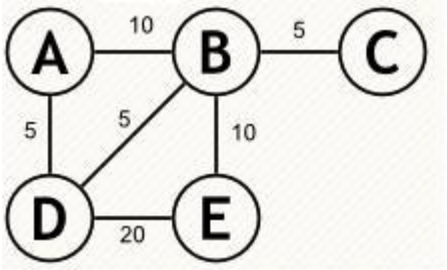
- Algoritmanın çalışmasını daha iyi anlayabilmek için aşağıdaki örnek üzerinden adım adım algoritmayı kullanarak en kısa yolu bulalım:



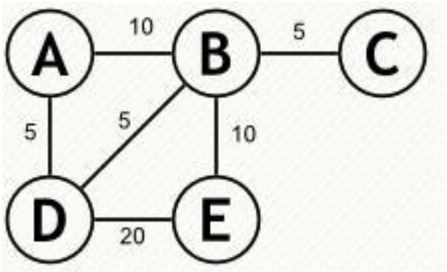


Floyd-Warshall Örnek

- Yukarıdaki şekil incelendiğinde A’dan E’ye giden birden çok yol bulunabilir:
- Yol 1: A -> B -> E 20
- Yol 2: A -> D -> E 25
- Yol 3: A -> B -> D -> E 35
- Yol 4: A -> D -> B -> E 20



- Yukarıdaki yollar çıkarıldıktan sonra en kısasının 20 uzunluğunda olduğu bulunabilir. Şimdi bu yollardan en kısasını Floyd-Warshall algoritmasının nasıl bulduğunu adım adım inceleyelim:



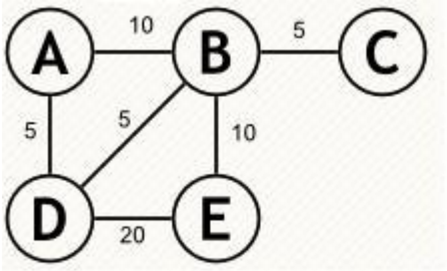
Floyd-Warshall Örnek

- 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.

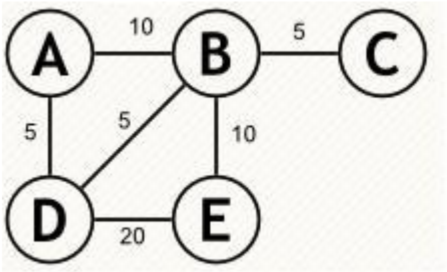
Floyd-Warshall Örnek



- 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

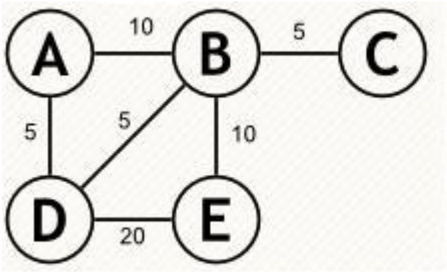
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0\,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1\,. \end{cases}$$

Floyd-Warshall Örnek



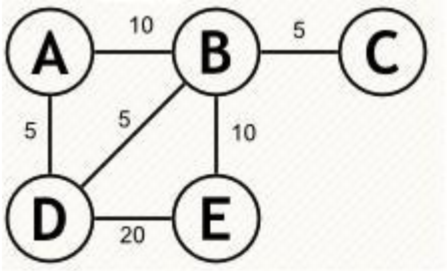
- B üzerinden atlanarak ulaşılan düğümleri güncelleyelim

	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 atlananan 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 Örnek

- 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).