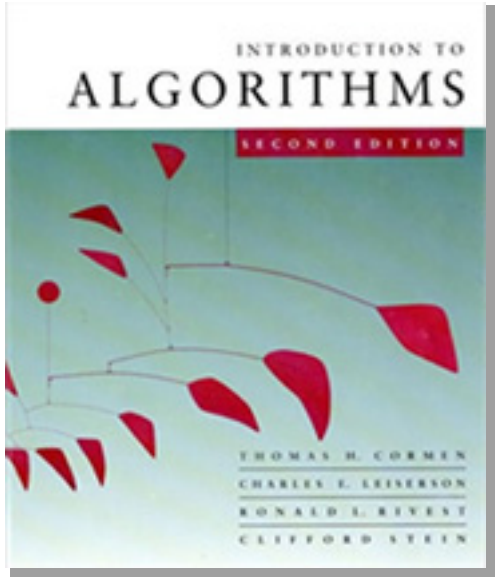
Algoritmalara Giriş
6.046J/18.401J

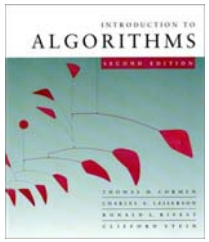


Ders19

En kısa yollar III

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

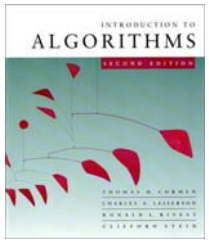
Prof. Charles E. Leiserson



En kısa yollar

Tek-kaynaklı en kısa yollar

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



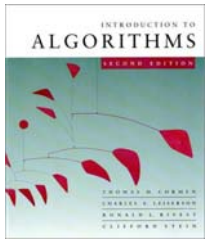
En kısa yollar

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.

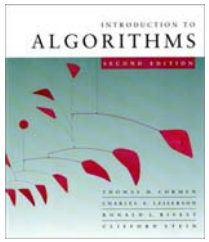


Tüm-ikili en kısa yollar

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 $n \times n$ matrisidir.



Tüm-ikili en kısa yollar

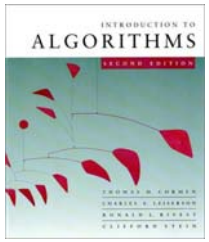
Girdi: $G = (V, E)$ yönlü grafiğinde, $V = \{1, 2, \ldots, n\}$ iken, $w : E \to \mathbb{R}$.
kenar-ağırlık fonksiyonuyla

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!



Dinamik programlama

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

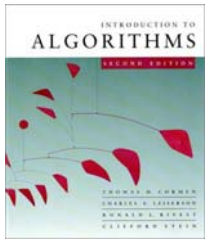
$d_{ij}^{(m)} =$ i ' den j 'ye en kısa yol ağırlığı-
en çok m sayıda kenarda kullanıldığında.

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) $m = 1, 2, \ldots, n-1$,

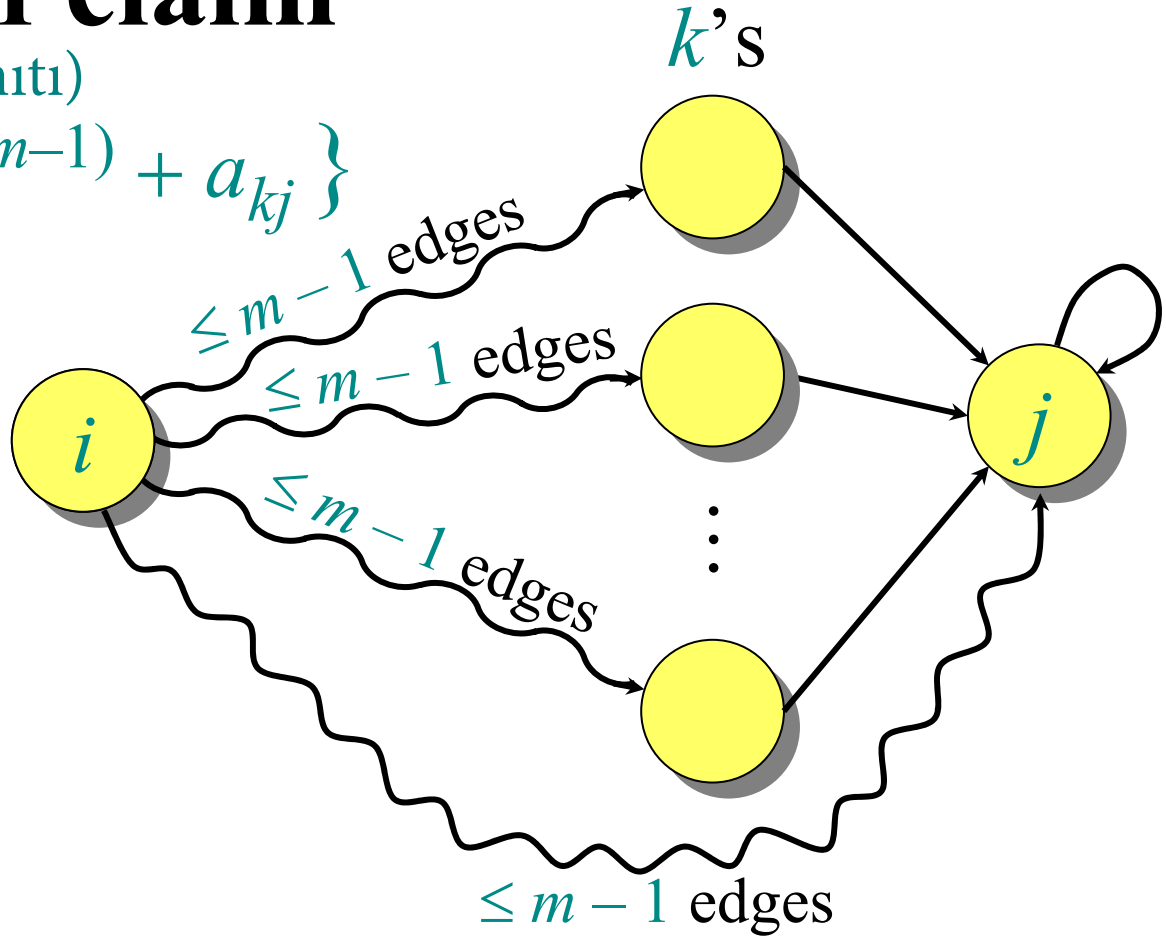
$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}.$$



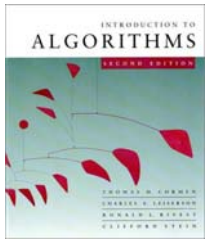
Proof of claim

(iddianın kanıtı)

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



edge(kenar)



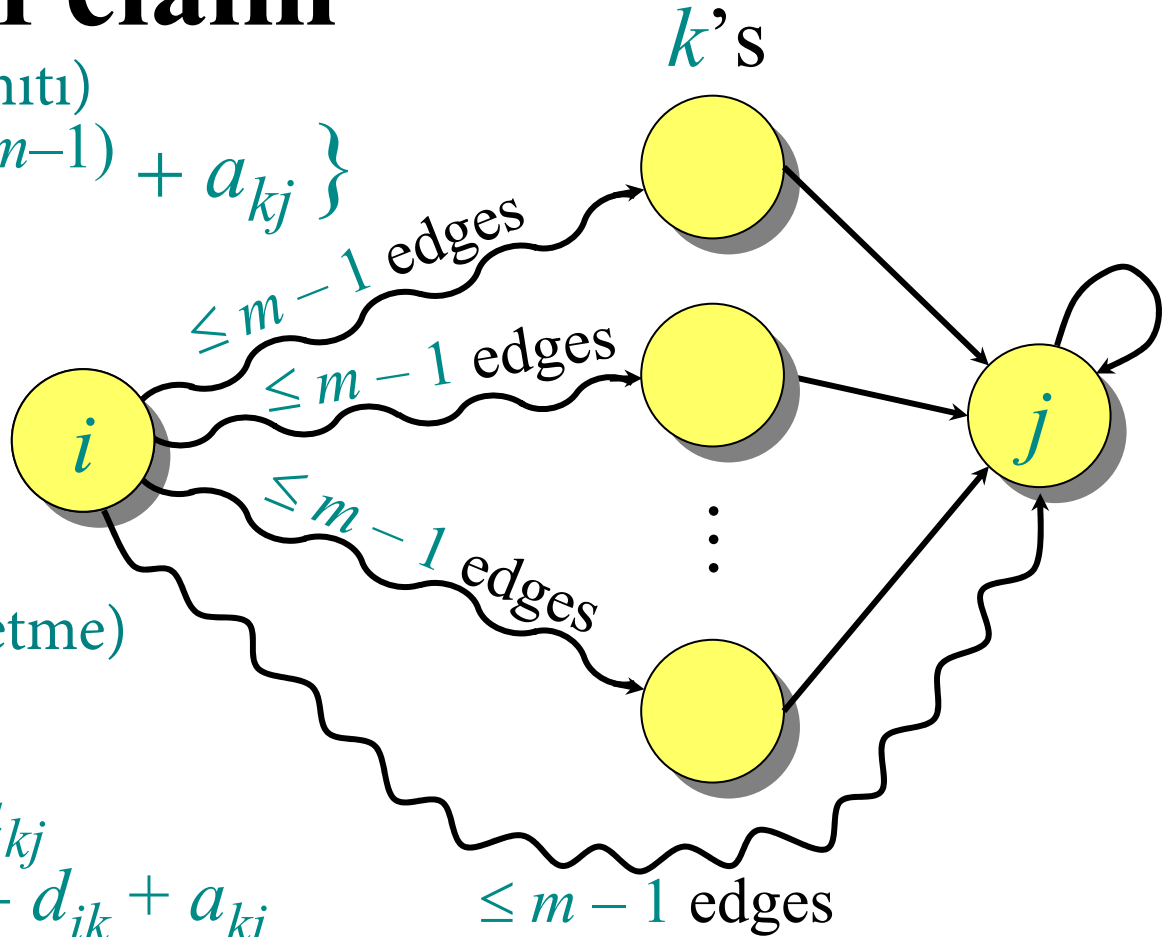
Proof of claim

(iddianın kanıtı)

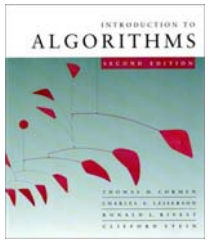
$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$$

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)



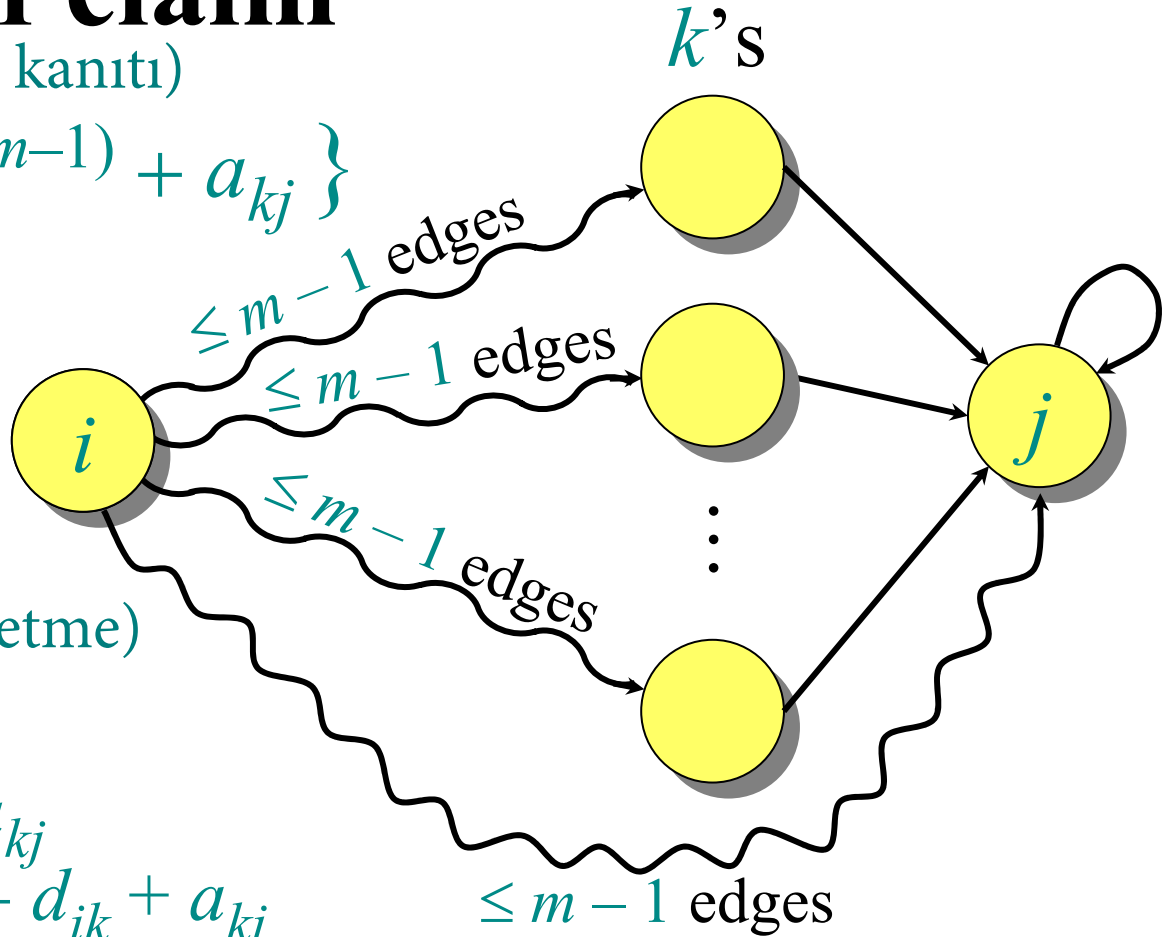
edge(kenar)



Proof of claim

(iddianın kanıtı)

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

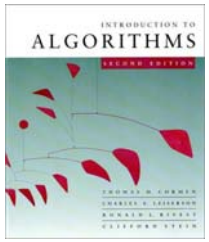


Relaxation! (gevşetme)

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)

Not: Negatif ağırlık çevrimi olmaması demek:

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

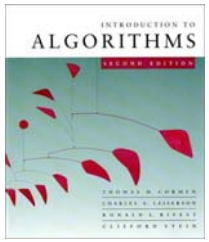


Matris Çarpımı

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

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

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



Matris Çarpımı

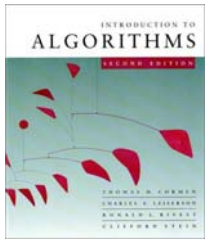
C, A ve B $n \times n$ matrislerse

$C = A \cdot B$ ' yi hesapla:

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

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

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



Matris çarpımı

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

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

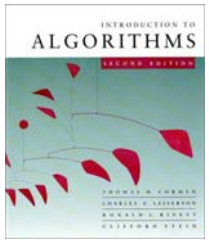
Time(süre) = $\Theta(n^3)$ standart algoritmayı kullanıyor.

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

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

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

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



Matris çarpımı (devamı)

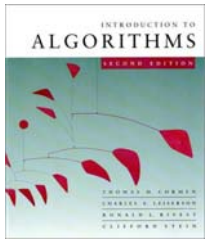
(min, +) çarpımı ***çağrışımsal***dır, ve gerçek sayılarla, ***kapalı semiring(closed semiring)*** olarak adlandırılan cebirsel bir yapı oluşturur.

Sonuçta bunu hesaplayabiliriz

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

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

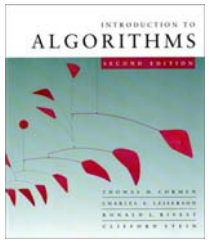
Hesaplayın: $A^2, A^4, \ldots, A^{2^{\lceil \lg(n-1) \rceil}}$.

$O(\lg n)$karelemeler

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

Time(süre) = $\Theta(n^3 \lg n)$.

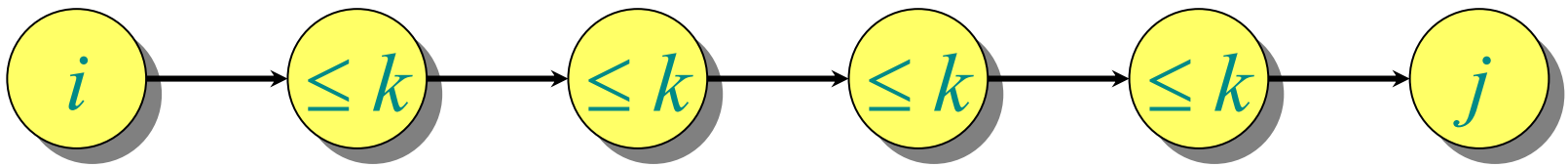
Negatif ağırlık çevrimlerini bulmak için, köşegendeki negatif değerleri $O(n)$ ek zamanında kontrol edin.



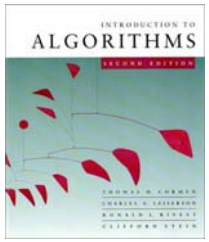
Floyd-Warshall algoritması

Bu da dinamik programlamadır, fakat daha hızlıdır!

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

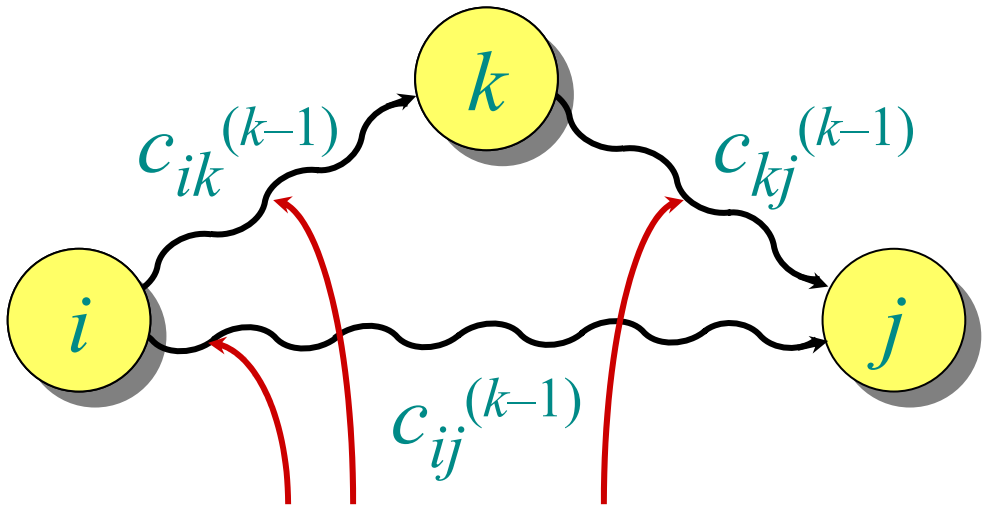


böylece, $\delta(i, j) = c_{ij}^{(n)}$. ve $c_{ij}^{(0)} = a_{ij}$.

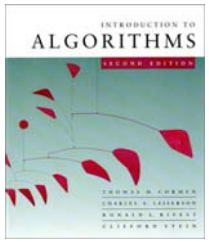


Floyd-Warshall yinelemesi

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



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



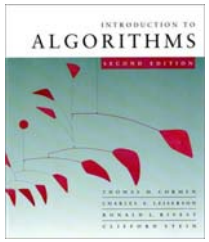
Floyd-Warshall için sözde kod

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.



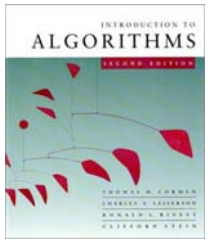
Bir yönlendirilmiş grafiğin geçişli kapanışı

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.

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

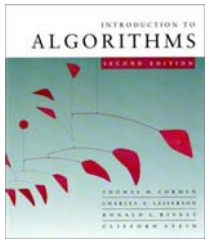
Time(süre) $= \Theta(n^3)$.



Grafik yeniden ağırlıklandırması

Teorem. Bir $h : V \to \mathbb{R}$, 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.



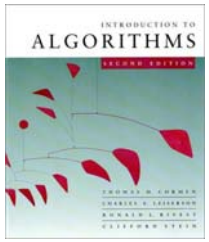
Grafik yeniden ağırlıklandırması

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.

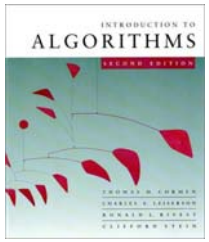
$$
\begin{aligned}
w_h(p) &= \sum_{i=1}^{k-1} w_h(v_i, v_{i+1}) \\
&= \sum_{i=1}^{k-1} \bigl(w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1})\bigr) \\
&= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) + h(v_1) - h(v_k) \\
&= w(p) + h(v_1) - h(v_k) .
\end{aligned}
$$

Aynı miktar!



Yeniden ağırlıklandırılan grafiklerde en kısa yollar

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



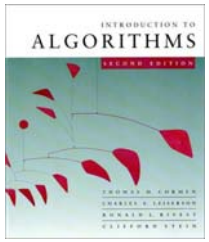
Yeniden ağırlıklandırılan grafiklerde en kısa yollar

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 $= O(VE + V^2 \lg V)$.