BM 210 Algoritma Analizi (Analysis of Algorithms)

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Ağırlıklandırılmış graflarda (weighted graphs) single-source en kısa yollar (shortest paths)
  - Shortest-Path Problemleri
  - Shortest Path Özellikleri ve Relaxation
  - Dijkstra Algoritması
  - Bellman-Ford Algoritması

Shortest Path

- Bir digraph $G = (V,E)$ için ağırlık fonksiyonu $W: E \rightarrow R$ şeklinde tanımlanabilir(kenarlara reel sayılar atar)
- $p = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ yolu için ağırlık aşağıdaki gibidir

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

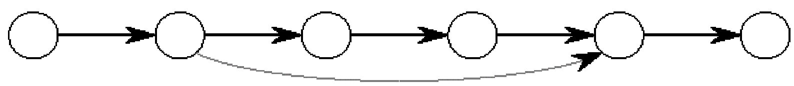
- Shortest path = minimum ağırlığa sahip bir yoldur
- Uygulamalar
  - statik/dinamic network routing
  - robot hareket planlaması
  - trafikte map/route oluşturulması

Shortest-Path Problemleri

- Shortest-Path problemleri
  - **Single-source (single-destination).** Verilen bir kaynak vertex'inden diğer tüm vertex'lere giden en kısa yolun bulunması.
  - **Single-pair.** Verilen iki vertex için ikisinin arasındaki en kısa yolun bulunması.
  - **All-pairs.** Her iki vertex çifti için en kısa yolun bulunması.

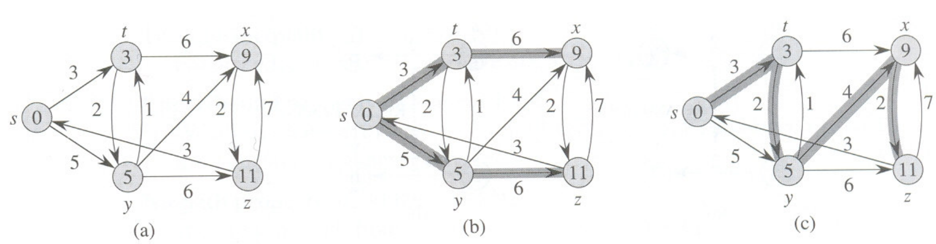
Optimal Substructure

- *Teorem*: en kısa yolların alt yollarıda (subpaths) en kısa yollardır
- *Proof:* (”cut ve paste”)
  - Eğer bazı alt yollar en kısa yol değilse, daha kısa alt yol alınarak daha kısa toplam yol oluşturulur



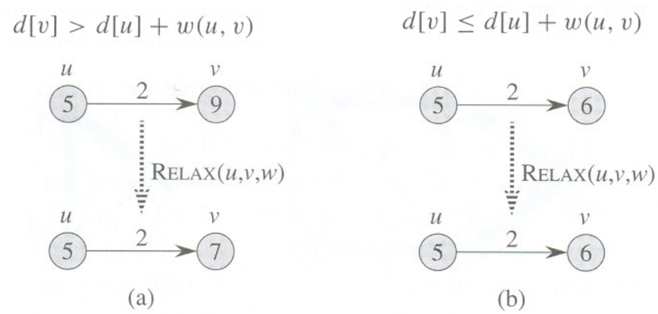
Negatif Ağırlıklar ve Döngüler?

- Negatif kenarlar olabilir ancak döngüler olamaz
- En kısa yollar döngüye sahip olamazlar
  - Bir G grafındaki herhangi bir en kısa yol *n-1* kenardan daha fazla kenara sahip olamaz, burada *n* toplam vertex sayısını ifade eder



Relaxation

- Her $v \in V$ için, $d[v]$ uzaklık değerinin s kaynak vertex'inden v hedef vertex'ine giden en kısa yolun ağırlığı olması sağlanmalıdır, başlangıçta ∞ olarak alınır
- Bir (u,v) kenarının relax edilmesi, u vertex'ine gelene kadar bulunmuş v vertex'ine giden en kısa yolu geliştirip gelişteremediğinin test edilmesidir.



Initilization ve Relaxation

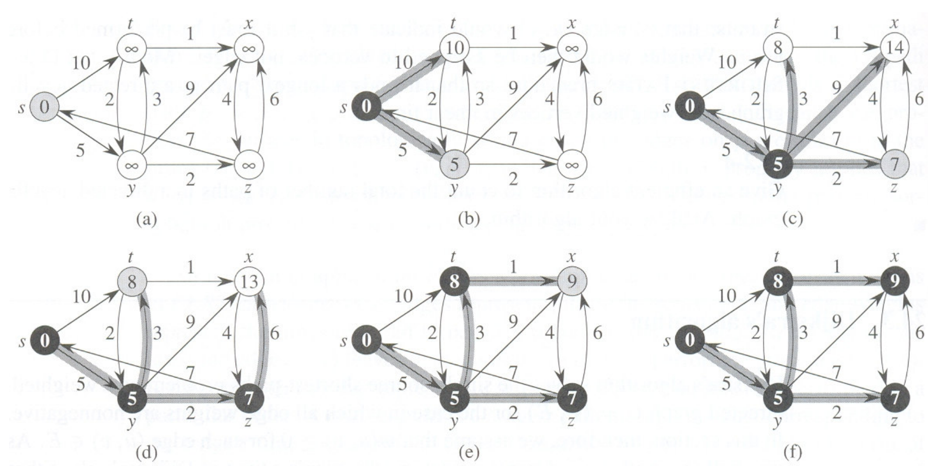
```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ V[G]
    do d[v] ← ∞
       π[v] ← NIL
d[s] ← 0
```

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

Dijkstra Algoritması

- Negatif olmayan ağırlığa sahip kenarlar olmalıdır
- Greedy yaklaşımı kullanır, MST için kullanılan Prim algoritması gibi çalışır
- Eğer tüm ağırlıklar 1 olursa breadth-first aramaya benzer
- Min-priority queue (*Q*) kullanılır. BFS algoritması FIFO kuyruk yapısı kullanır, burada PQ kullanılır ve her uzaklık değişiminde (relax) yeniden düzenlenir
- İşlem adımları
  - Çözülmüş vertex'ler için bir *S* kümesi oluşturur
  - Her adımda *u* vertex'ine komşu vertexlerden en yakın olanını seçerek *S* kümesine ekler ve *u*'nun tüm kenarları için relax işlemi yapar

Dijkstra Algoritması



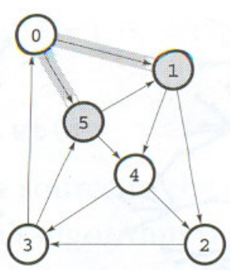
Dijkstra Algoritması

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S ← Ø
  Q ← V[G]
  while Q ≠ Ø
      do u ← EXTRACT-MIN(Q)
         S ← S ∪ {u}
         for each vertex v ∈ Adj[u]
             do RELAX(u, v, w)
```

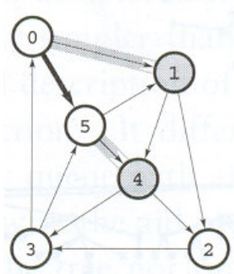
Çalışma Süresi = O(E* lg *V)

Dijkstra Algoritması (Örnek-1)

0-1	.41
1-2	*.51*
2-3	.50
4-3	.36
3-5	.38
3-0	.45
0-5	.29
5-4	.21
1-4	.32
4-2	.32
5-1	.29



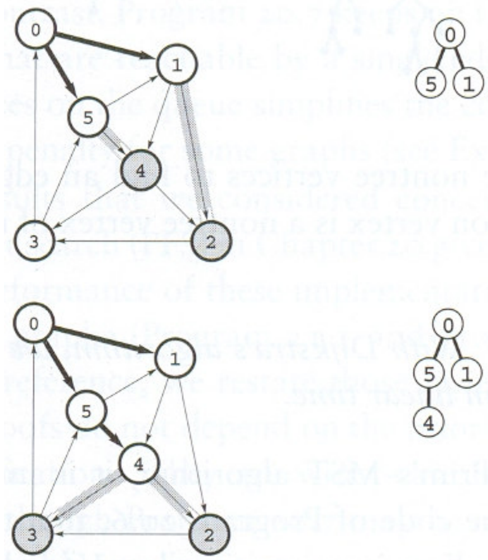
0-5 .29
0-1 .41



0-1 .41
5-4 .50

Dijkstra Algoritması (Örnek-2)

0-1 .41
1-2 .51
2-3 .50
4-3 .36
3-5 .38
3-0 .45
0-5 .29
5-4 .21
1-4 .32
4-2 .32
5-1 .29

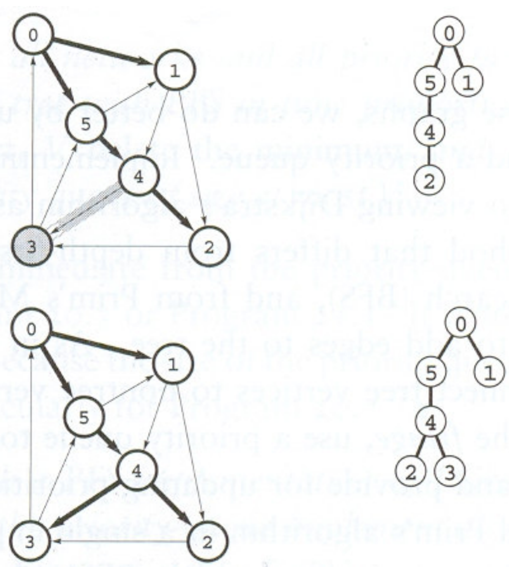


5-4 .50
1-2 .92

4-2 .82
4-3 .86

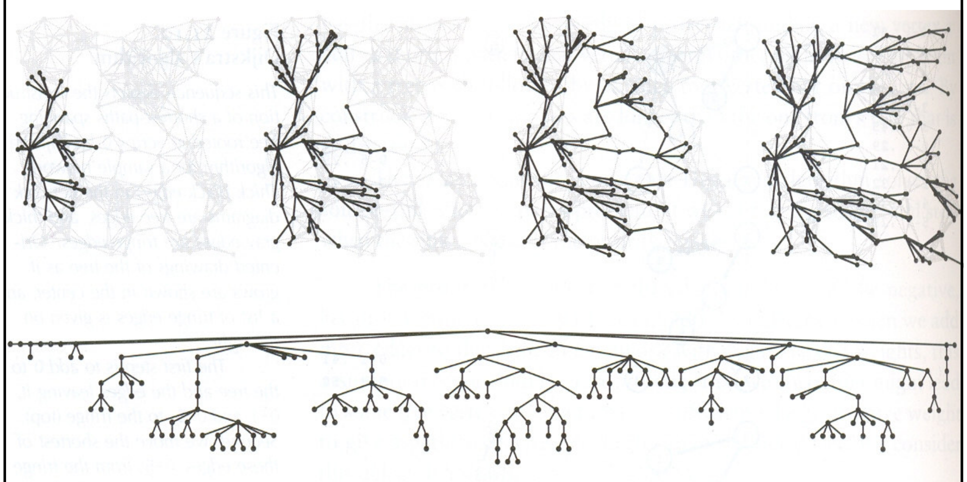
Dijkstra Algoritması (Örnek-3)

0-1 .41
1-2 .51
2-3 .50
4-3 .36
3-5 .38
3-0 .45
0-5 .29
5-4 .21
1-4 .32
4-2 .32
5-1 .29



4-3 .86

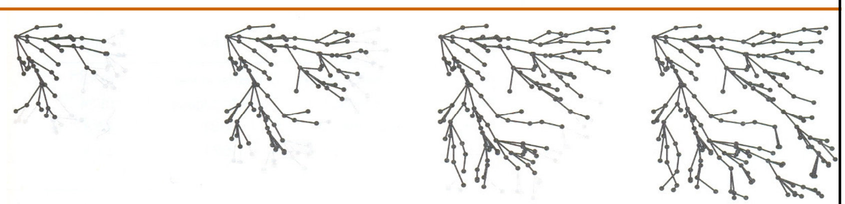
Dijkstra Algoritması (Örnek-4)



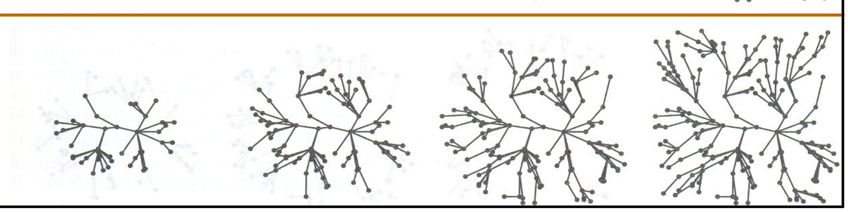
Dijkstra Algoritması (Örnek-5)

Sol Kenar

Sol Üst Köşe



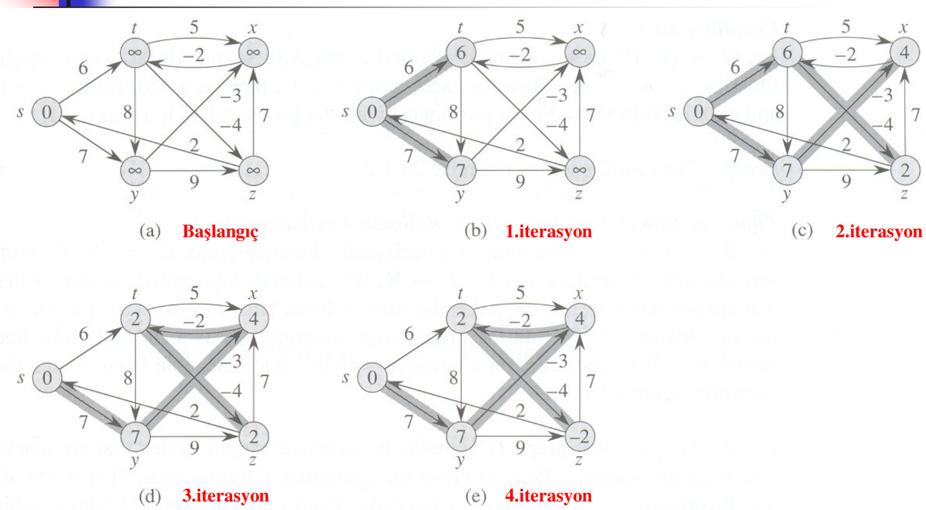
Orta



Bellman-Ford Algoritması

- Dijkstra algoritması negatif kenarlarda kullanılamaz
- Bellman-Ford algoritması negative döngüleri kontrol eder ve negatif döngü varsa *false* değeri döndürür, yoksa en kısa yol ağacını döndürür

Bellman-Ford Algoritması



Kenarlar için relax sırası – (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)

Bellman-Ford Algoritması

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

Çalışma Süresi = O(V E)