BM 210 Algoritma Analizi (Analysis of Algorithms)

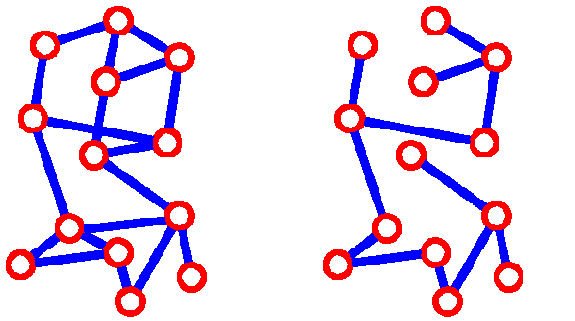
Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Greedy Algoritmaları
- Ağırlıklandırılmış Graflar
- Minimum Spanning Trees (En Az Yayılımlı Ağaçlar)
  - Greedy Seçim
  - Prim Algoritması
  - Kruskal Algoritması

Spanning Tree

- Bir **spanning tree**, **G** grafındaki bir alt graftır ve aşağıdaki özelliklere sahiptir;
  - bir ağaç yapısı oluşturur
  - **G** grafındaki tüm nodları içerir

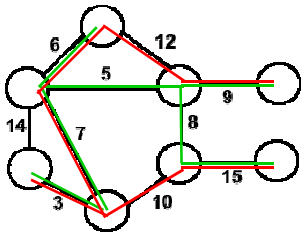


G

spanning tree of G

Minimum Spanning Trees

- Yönlendirilmemiş, bağlı bir graf $G = (V,E)$ olarak tanımlansın



- Spanning tree: bütün nodları bağlayan bir ağaçtır
- Optimizasyon problemi – **Minimum spanning tree** (MST): bir T ağacıdır, tüm nodları birbirine bağlar ve aşağıdaki ifadeyi minimize eder

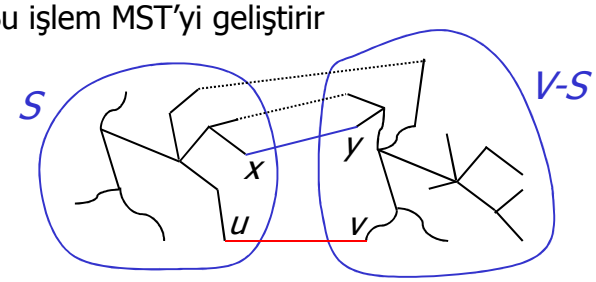
$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

Greedy Seçim

- MST' nin kenarları için $V - 1$ tane seçim yapılması gerekmektedir
- Greedy seçim : lokal olarak optimal (greedy) seçim global optimal çözümü oluşturur

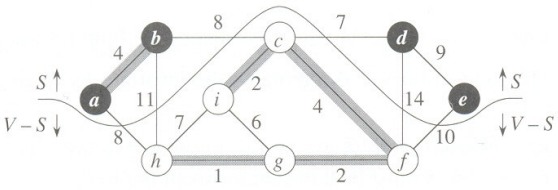
Greedy Seçim (2)

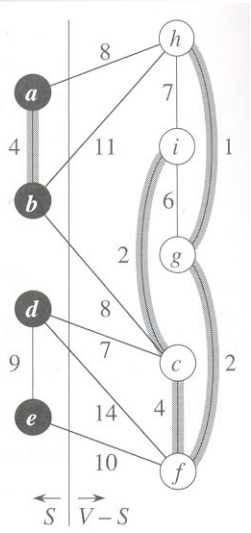
- Teorem
  - (u,v) düşük ağırlıklı bir kenar ancak $(u,v) \notin$ MST ise
  - MST içinde u' dan v'ye bir yol aranır
  - MST içinde (x, y) gibi bir yol bulunursa, (x, y) yerine (u,v) alınır
  - Bu işlem MST'yi geliştirir



Greedy Seçim (3)

- Örnekler





MST Algoritması

```
Generic-MST(G, w)
1 A←∅  // MST'ye ait kenarları tutar
2 while A bir spanning tree oluşturmuyorsa do
3     A için geçerli bir (u,v) kenarı bul
4     A←A∪{(u,v)}
5 return A
```

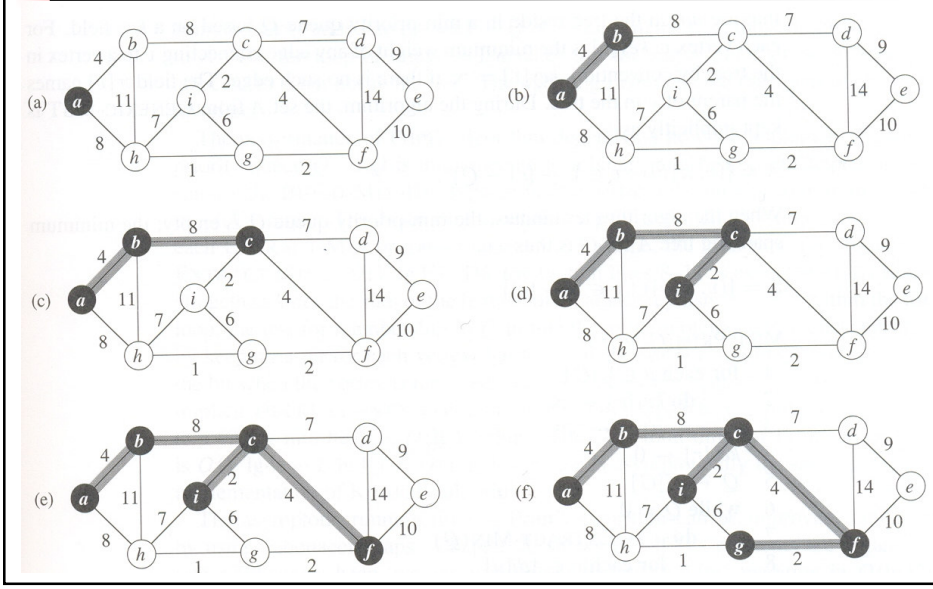
Geçerli kenar – A'nın özelliğini bozmayan kenar (Örn:. döngü oluşturmayan kenar)

```
MoreSpecific-MST(G, w)
1    A←∅  // MST'ye ait kenarları tutar
2    while A bir spanning tree oluşturmuyorsa do
3.1     G' yi (S, V-S) şeklinde kes
3.2     S ile V-S' yi bağlayan min ağırlıklı (u,v)
      kenarını al
4     A←A∪{(u,v)}
5 return A
```

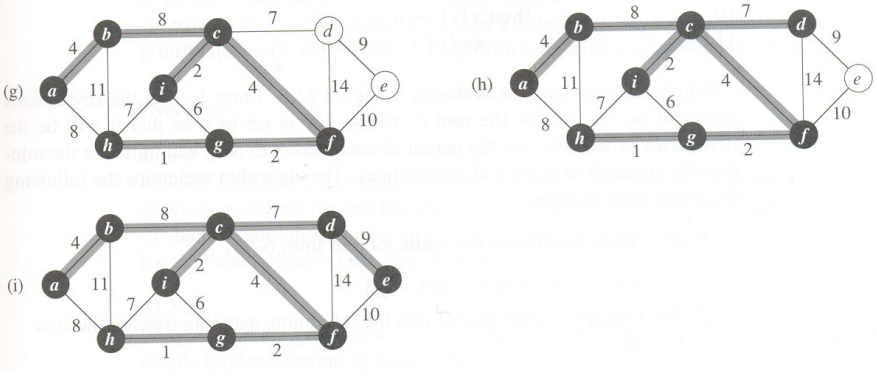
Prim Algoritması

- Vertex tabanlı bir algoritmadır
- Bir T ağacını her adımda bir vertex ekleyerek büyütür
- Bir root vertex *r* ile başlar ve tüm vertex'leri içine alıncaya kadar ağaç büyültülür

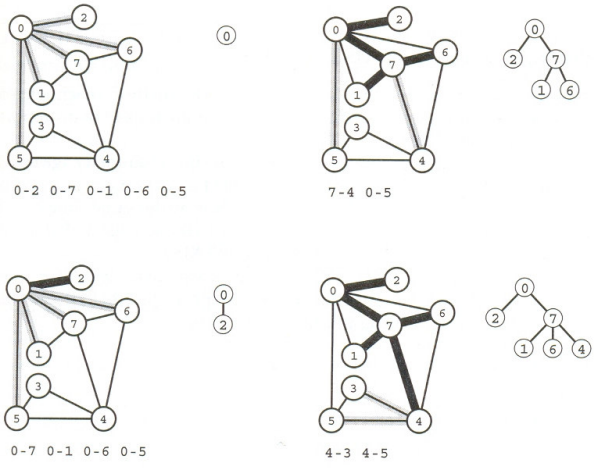
Prim Algoritması - Örnek (1)



Prim Algoritması - Örnek (2)

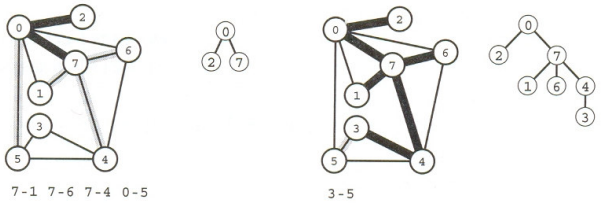


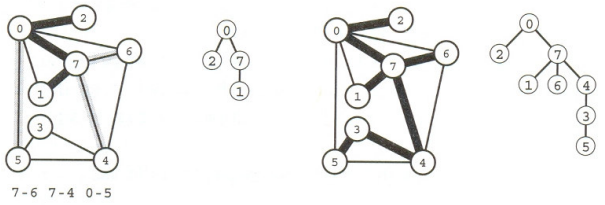
Prim Algoritması - Örnek (3)



	0	1	2	3	4	5	6	7
mst	0	7-1	0-2	4-3	7-4	3-5	7-6	0-7
wt		.21	.29	.34	.46	.18	.25	.31

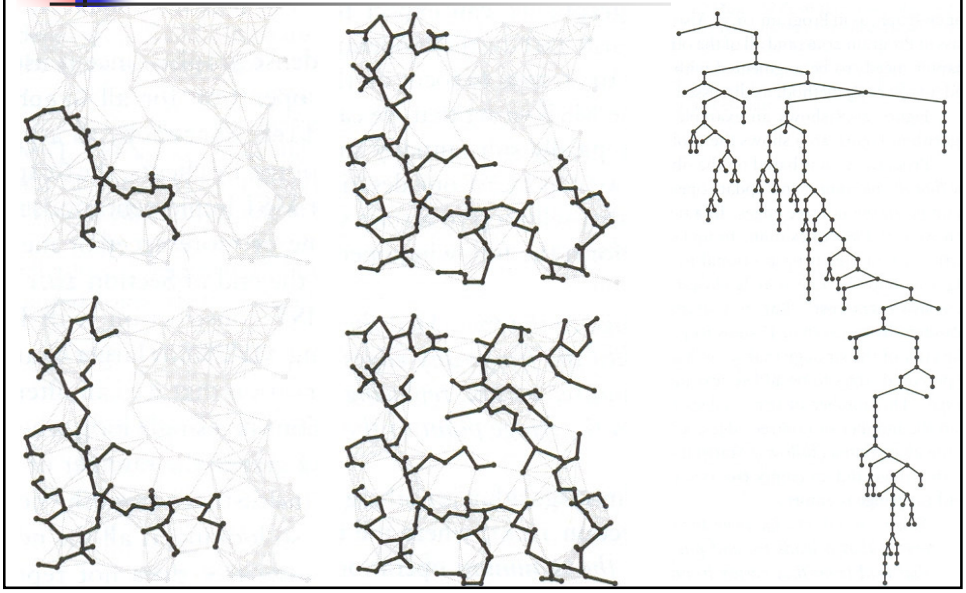
Prim Algoritması - Örnek (4)





	0	1	2	3	4	5	6	7
mst	0	7-1	0-2	4-3	7-4	3-5	7-6	0-7
wt		.21	.29	.34	.46	.18	.25	.31

Prim Algoritması - Örnek (5)



Prim Algoritması

- Algoritmanın çalışması sırasında aşağıdaki durum korunur

$$A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$$

- Algoritmanın çalışması bittiğinde Q kuyruğu boştur ve A minimum-spanning tree'yi gösterir

$$A = \{(v, \pi[v]) : v \in V - \{r\}\}$$

```
MST-PRIM(G, w, r)
 1  for each u ∈ V[G]
 2       do key[u] ← ∞
 3          π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  while Q ≠ ∅
 7       do u ← EXTRACT-MIN(Q)
 8          for each v ∈ Adj[u]
 9               do if v ∈ Q and w(u, v) < key[v]
10                  then π[v] ← u
11                       key[v] ← w(u, v)
```

Çalışma Süresi = $O(E \lg V)$

Ağaca en yakın vertex'i seçer

V'yi MST'ye bağlayan minimum ağırlık değeridir

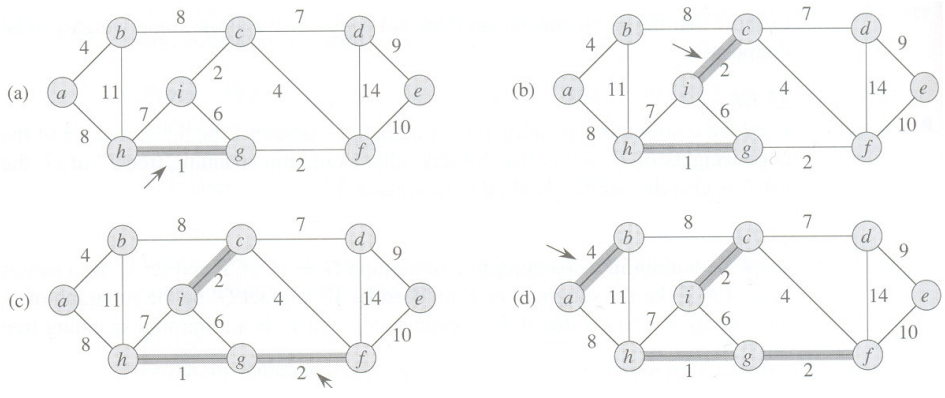
Greedy Algoritmalar

- **Greedy** algoritmaları eğer bir *optimal substructure* varsa kullanılabilir.
- *greedy seçim* her adımda optimal çözümü oluşturur

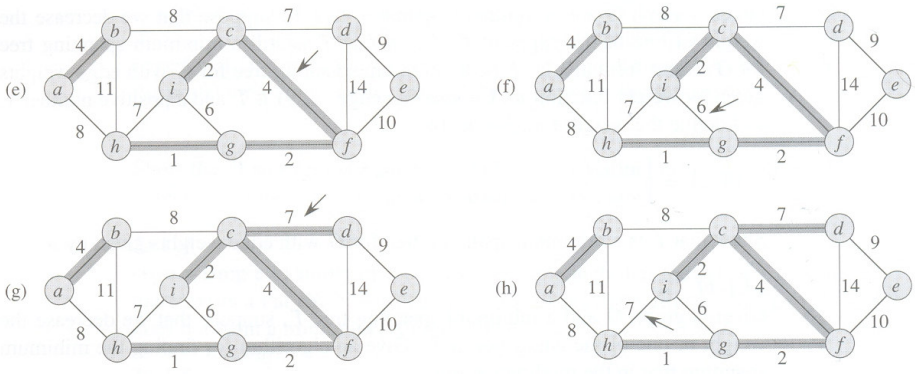
Kruskal Algoritması

- Kenar tabanlı algoritmadır
- Her adımda bir kenarı ekler ve kenarları artan ağırlık sırasında alır
- Algoritma bir *A* ağacını oluşturur– (*A* - **forest of trees)**. Bir kenar iki farklı ağacı birleştiriyorsa kabul edilir

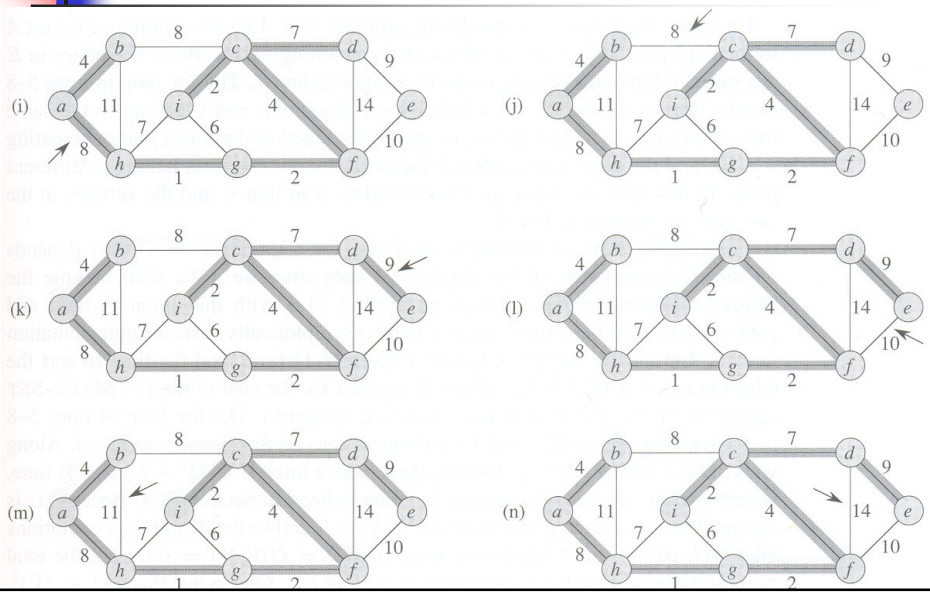
Kruskal Algoritması - Örnek (1)



Kruskal Algoritması - Örnek (2)

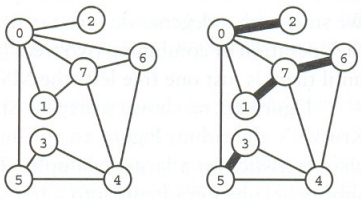


Kruskal Algoritması - Örnek (3)



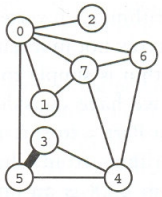
Kruskal Algoritması - Örnek (4)

```
0-6 .51      5-3 .18
0-1 .32      7-1 .21
0-2 .29      7-6 .25
4-3 .34      0-2 .29
5-3 .18      7-0 .31
7-4 .46      0-1 .32
5-4 .40      4-3 .34
0-5 .60      5-4 .40
6-4 .51      7-4 .46
7-0 .31   *  0-6 .51
7-6 .25   *  6-4 .51
7-1 .21   *  0-5 .60
```

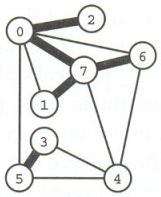


a

e



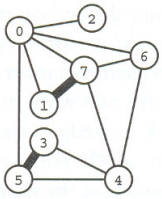
b



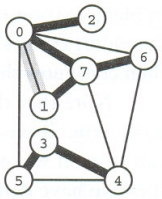
f

Kruskal Algoritması - Örnek (5)

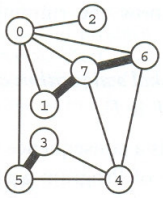
```
0-6 .51      5-3 .18
0-1 .32      7-1 .21
0-2 .29      7-6 .25
4-3 .34      0-2 .29
5-3 .18      7-0 .31
7-4 .46      0-1 .32
5-4 .40      4-3 .34
0-5 .60      5-4 .40
6-4 .51      7-4 .46
7-0 .31   *  0-6 .51
7-6 .25   *  6-4 .51
7-1 .21   *  0-5 .60
```



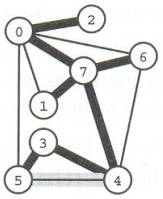
c



g



d



h

Kruskal Algoritması

```
MAKE-SET(x)
1  p[x] ← x
2  rank[x] ← 0

UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1  if rank[x] > rank[y]
2     then p[y] ← x
3     else p[x] ← y
4          if rank[x] = rank[y]
5             then rank[y] ← rank[y] + 1
```

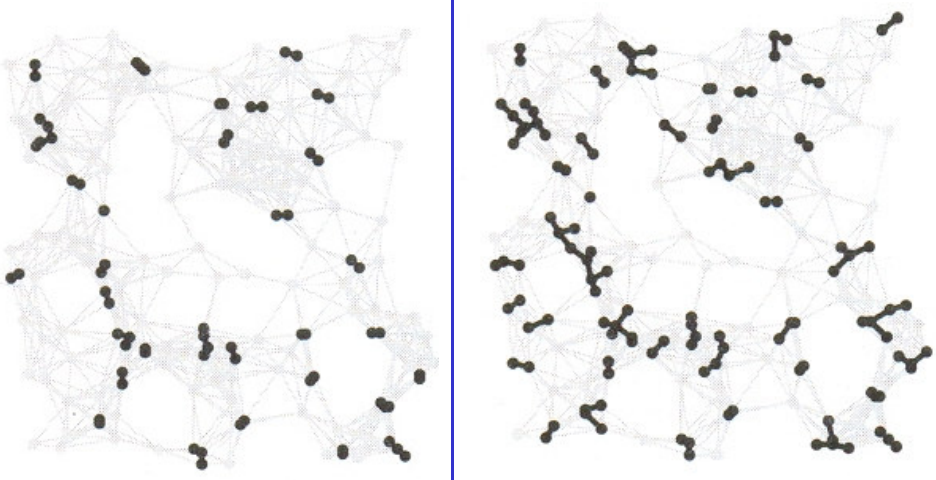
```
FIND-SET(x)
1  if x ≠ p[x]
2     then p[x] ← FIND-SET(p[x])
3  return p[x]
```

Kruskal Algoritması

```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each vertex v ∈ V[G]
3       do MAKE-SET(v)
4  sort the edges of E into nondecreasing order by weight w
5  for each edge (u, v) ∈ E, taken in nondecreasing order by weight
6       do if FIND-SET(u) ≠ FIND-SET(v)
7             then A ← A ∪ {(u, v)}
8                  UNION(u, v)
9  return A
```

Çalışma Süresi = O(E lg V)

Kruskal Algoritması (Örnek-1)



Kruskal Algoritması (Örnek-2)