Algoritma Analizi

Ders 13

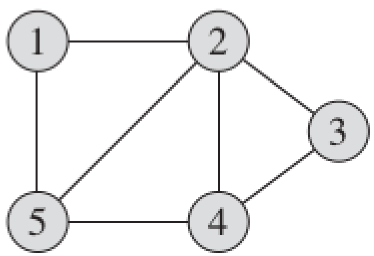
Doç. Dr. Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

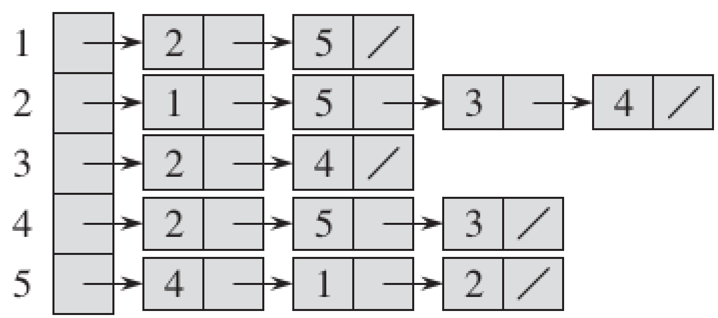
Graf algoritmaları

- Grafların gösterimi
  - G = (V,E) şeklinde tanımlanan bir graf gösterimi için kullanılan iki yöntem vardır.
    - Komşuluk listeleri (İng: adjaceny list)
    - Komşuluk matrisi (İng: adjacency matris)
  - Komşuluk listesi yöntemi seyrek graflar için daha uygundur.
    - $|E|$, $|V|^2$'den çok küçük olduğunda seyrek graf diyoruz.
  - Komşuluk matrisi yöntemi yoğun graflar için daha uygundur.
    - $|E|$, $|V|^2$'ye yakın olduğunda yoğun graf diyoruz.
  - Graflar yönlü veya yönsüz olabilir.

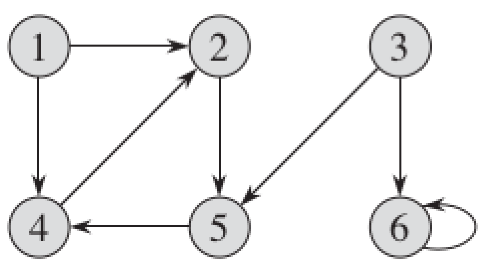
Graf algoritmaları

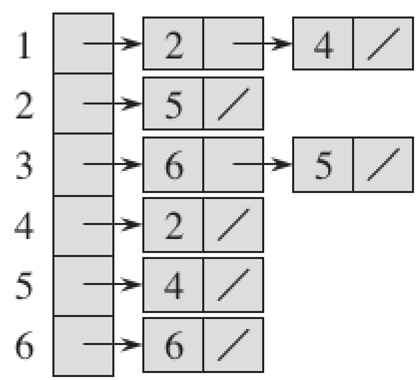
- Grafların gösterimi





	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0





	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Graf algoritmaları

- Grafların gösterimi
  - Eğer G bir yönlü graf ise, komşuluk listelerinin uzunluğu toplamı |E|‘dir.
  - Eğer G bir yönsüz graf ise, komşuluk listelerinin uzunluğu toplamı 2|E|‘dir.
  - Komşuluk listelerini kullanarak ağırlıklı graf oluşturabiliriz.
    - Her (u,v) kenarı için bir ağırlık değeri belirlenebilir.
      - $w : E \rightarrow R$
  - Komşuluk listelerini kullanıldığında hızlı bir şekilde bir kenarın var olup olmadığı söylenemez.
  - Komşuluk matrisi ise bu bilgiyi hızlı bir şekilde verebilir.
  - Ancak komşuluk matrisleri daha fazla hafıza gerektirir.
  - Komşuluk matrislerinin hafıza ihtiyaçlarını azaltmak için bazı yöntemler bulunmaktadır.
  - Komşuluk matrislerinde 1 yerine ağırlık değerini saklayarak ağırlıklı graf oluşturabiliriz.
  - Her iki yöntem için de kenar ve köşeler ile alakalı özellik (İng: attribute) saklanabilir.

Graf algoritmaları

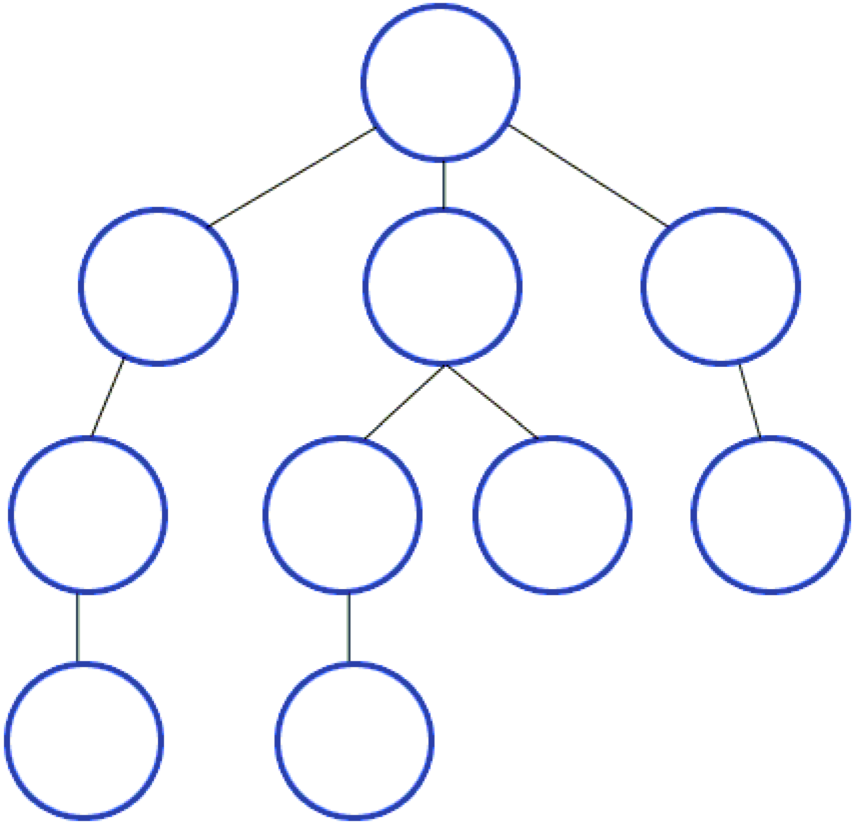
- Graf üzerinde hareket
  - Birçok problemde graf üzerindeki kenarları kullanarak düğümler arasında hareket etmek gerekir.
  - Örneğin, grafteki düğümler gidilebilecek şehirleri temsil edebilir.
    - Bu durumda bir şehirden diğerine giden yolu bulmak için düğümler arasında geçiş yapmak gerekir.
  - Bu durumda çoklukla kullanılan düğümler üzerinde hareket yöntemleri mevcuttur.
  - Bunlardan en bilinen iki tanesi
    - Derinlik öncelikli arama (İng: Depth First Search)
    - Enine arama (Breadth First Search)

Graf algoritmaları

- Derinlik öncelikli arama
  - Bu yöntem graf üzerinde dolaşmamızı sağlar.
  - Bu amaçla başlangıç düğümünün bir kenarından başlanır.
  - Git gide derinleşen (başlangıç noktasından uzaklaşan) şekilde sonraki düğümleri seçer.
  - Graf yapısını bir ağaca benzetirsek, en derine kadar düğümleri açar.
  - Bu arama yöntemi şu adımlardan oluşur.
    - 1. Bir başlangıç düğümü seçilir ve ziyaret edilir.
    - 2. Seçilen düğümün bir komşusu seçilir ve ziyaret edilir.
    - 3. Ziyaret edilecek komşu kalmayıncaya kadar 2. adım tekrar edilir.
    - 4. Komşu kalmadığı durumda geri dönülür ve önceki ziyaret edilmiş düğümler seçilerek 2. ve 3. adımlar tekrar edilir.

Graf algoritmaları

- Derinlik öncelikli arama



*Şekil https://en.wikipedia.org/wiki/Depth-first_search adresinden alınmıştır

Graf algoritmaları

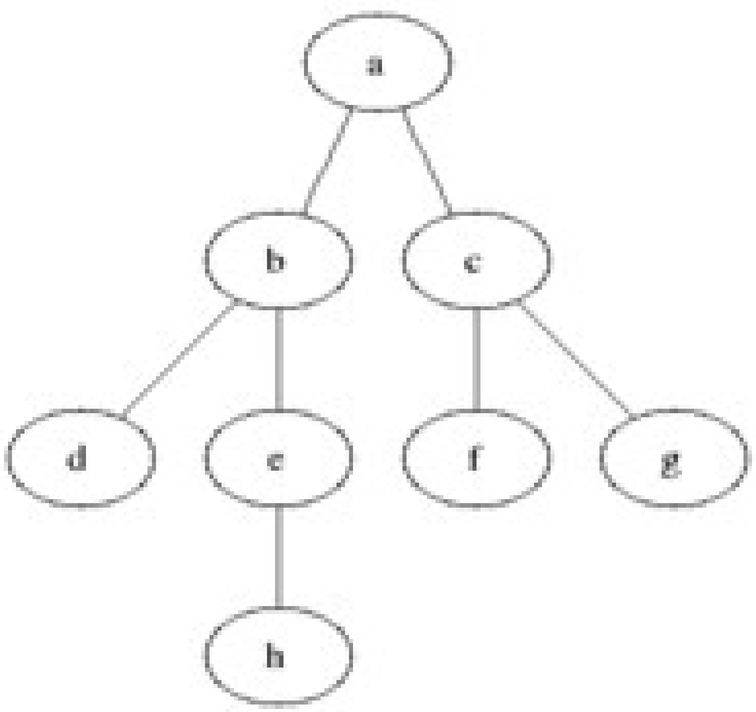
- Derinlik öncelikli arama
    - Derinlik öncelikli arama yığın yapısı kullanılarak gerçekleştirilebilir.
        - Önce başlangıç düğümünü yığına ekle
        - Başlangıç düğümünü ziyaret edilmiş olarak işaretle
        - Döngü: Yığın boşalana kadar devam et
            - Yığından eleman al
            - Alınan elemanın komşularını, daha önce ziyaret edilmemişse, yığına ekle ve ziyaret edilmiş olarak işaretle.

Graf algoritmaları

- Enine arama
    - Bu yöntem graf üzerinde dolaşmamızı sağlar.
    - Bu yaklaşıma göre düğümler seviye seviye ziyaret edilir.
        - Başlangıç düğümünden başlanır.
        - Daha sonra başlangıç düğümüne 1 uzaklıktaki bütün komşu düğümler ziyaret edilir.
        - Daha sonra başlangıç düğümüne 2 uzaklıktaki bütün komşu düğümler ziyaret edilir.
        - ...

Graf algoritmaları

- Enine arama



*Şekil https://en.wikipedia.org/wiki/Breadth-first_search adresinden alınmıştır

Graf algoritmaları

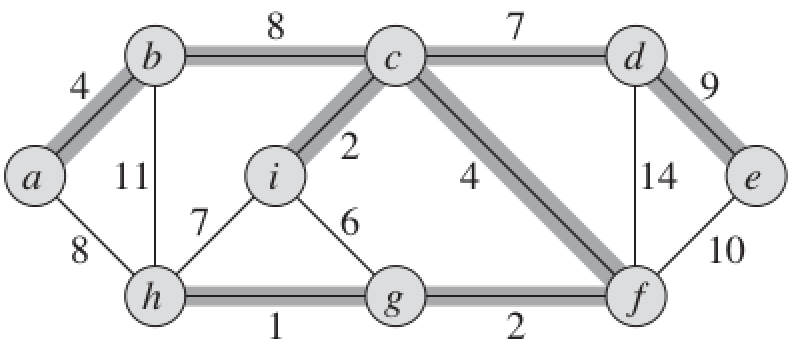
- Enine arama
    - Enine arama bir kuyruk yapısı kullanılarak gerçekleştirilebilir.
        - Önce başlangıç düğümünü kuyruğa ekle
        - Başlangıç düğümünü ziyaret edildi olarak işaretle
        - Döngü: Kuyruk boşalana kadar
            - Kuyruktan eleman al
            - Alınan elemanın komşularını, daha önce ziyaret edilmemişse, kuyruğa ekle ve ziyaret edilmiş olarak işaretle.
    - Enine arama ağırlıksız graflarda iki düğüm arasında en kısa yolu bulur.
    - Ağırlıklı graflarda ise iki düğüm arasındaki en kısa yolu bulmak için enine aramanın farklı versiyonları kullanılabilir.

Graf algoritmaları

- En-küçük kapsar ağaç (İng: Minimum spanning tree)
  - Elektronik devre oluştururken, sıklıkla birden fazla devre elemanının pinlerini birbirine bağlamanız gerekebilir.
    - n devre elemanını n – 1 kablo ile birbirine bağlayabiliriz.
    - En kısa kablo kullanan devre en tercih edilen olacaktır.
  - Bu problemi köşeleri birbirine bağlı, yönsüz bir G = (V, E) grafı ile modelleyebiliriz.
    - V, devre elemanlarının pinlerini, E ise kabloları temsil eder.
    - Her kenar, $(u,v) \in E$, bir uzunluğa, yani maliyete, sahip olacaktır.
    - Öyleyse, döngü içermeyen $T \subseteq E$, bütün köşeleri birleştiren ve en az maliyete sahip, kümesini bulmak istiyoruz.
      - $w(T) = \sum_{(u,v) \in T} w(u,v)$
  - Bu problem en-küçük kapsar ağaç problemi olarak adlandırılır.

Graf algoritmaları

- En-küçük kapsar ağaç problemi



Graf algoritmaları

- En-küçük kapsar ağaç oluşturma
  - G = (V,E) şeklinde köşe ve kenarlara sahip bir birleşik ve yönsüz grafımız olsun.
  - Ağırlık fonksiyonumuz w : E → R şeklinde tanımlanmış olsun.
  - Bu grafta bir en-küçük kapsar ağaç oluşturmak istiyoruz.
  - Bu bölümde göreceğimiz iki algoritma açgözlü (İng: greedy) yaklaşım ile çalışır.
    - Yani her adımda en iyi görünen adımı seçer.
  - Bu algoritmalar adım adım ağacı büyüterek en-küçük kapsar ağacı oluşturur.
    - Her adımda yeni bir kenar eklenir.
  - Yeni kenarlar, büyümekte olan A ağacına eklenirken aşağıda belirtilen döngü sabiti sağlanır:
    - Her döngü çalışması öncesinde, A bir en-küçük kapsar ağacın altkümesidir.
  - Algoritmanın yaklaşımına göre her adımda, yukarıda belirtilen döngü sabitini bozmayan yeni kenarlar, (u,v), A ağacına eklenir.

Graf algoritmaları

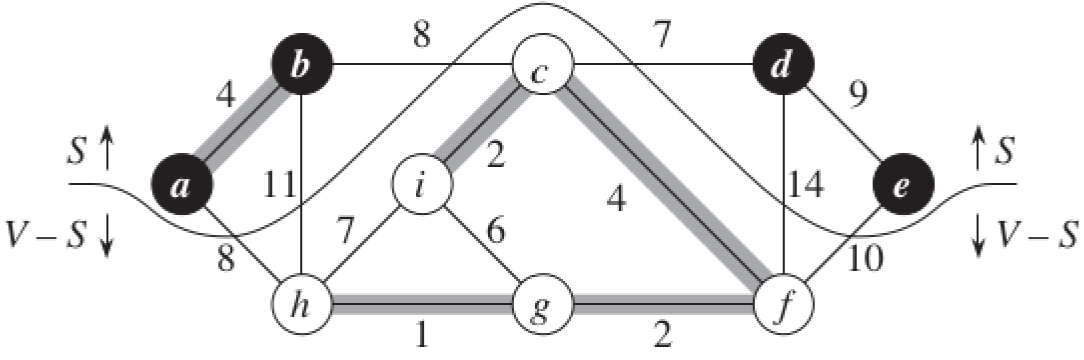
- En küçük kapsar ağaç oluşturma
  - Döngü sabitini bozmayan bu kenarlara güvenli kenar ismini veriyoruz.

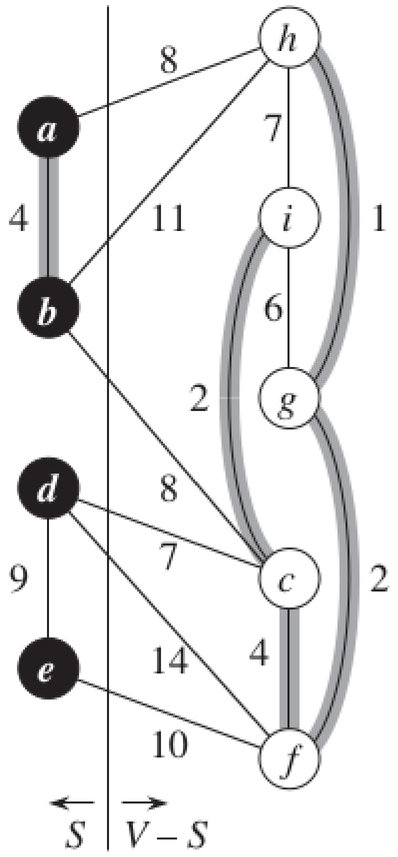
```
GENERIC-MST(G,w)
A = ∅
while A en-küçük kapsar ağaç değil ise
  A için güvenli olan bir (u,v) kenarı seç
  A = A U {(u,v)}
return A
```

  - Birkaç tanım:
    - Bir kesik, bir G = (V,E) grafını S ve (V – S) olmak üzere ikiye ayırır.
    - Bir (u,v) kenarı, bir köşesi S, diğer köşesi (V – S)‘de ise kesiği çaprazlar.
    - Köşeler altkümesi A içerisindeki hiçbir kenar bir kesiği çaprazlamıyor ise, bu kesik A kümesine uyar.
    - Bir kesiği çaprazlayan kenarlardan en düşük maliyete sahip olana hafif kenar denir.

Graf algoritmaları

- En küçük kapsar ağaç oluşturma





Graf algoritmaları

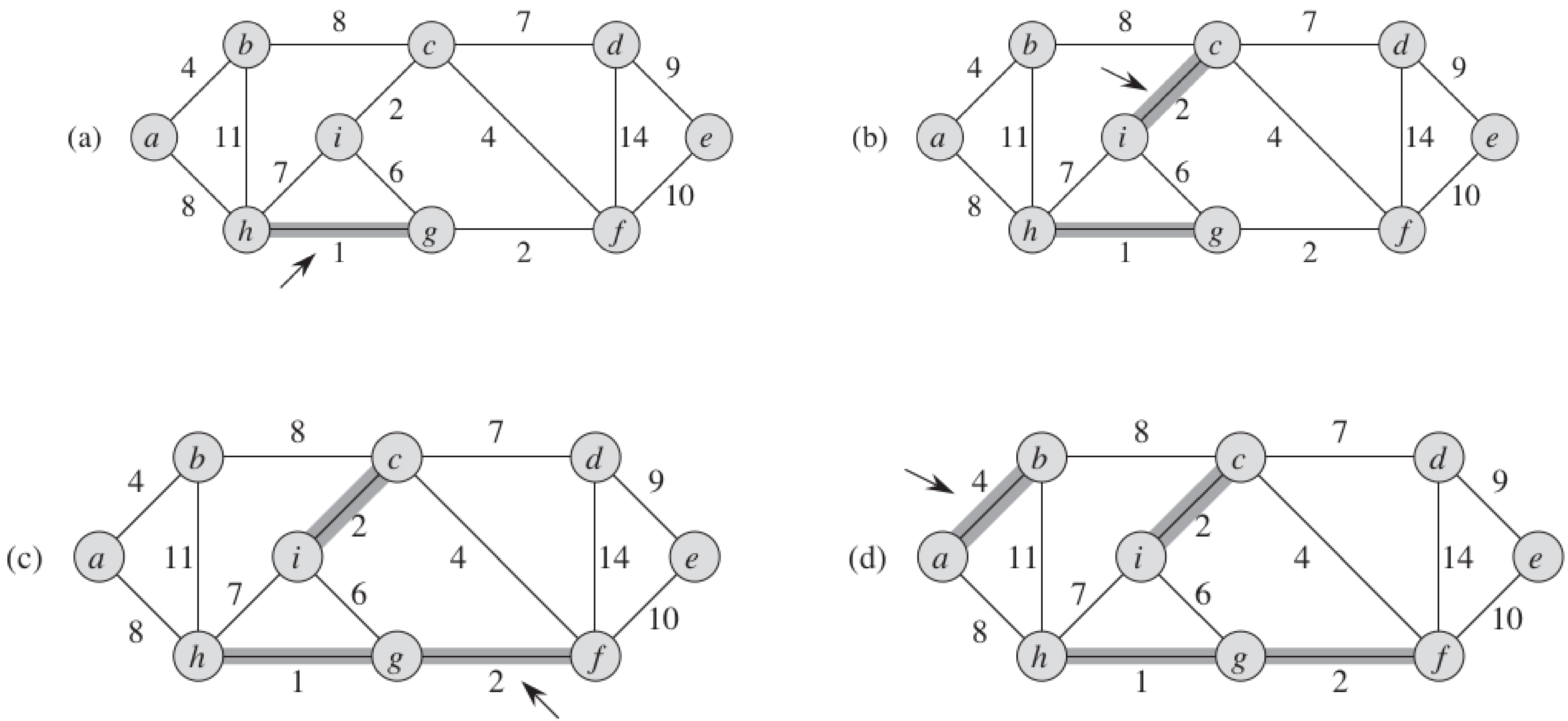
- En küçük kapsar ağaç oluşturma
  - Teorem: G = (V,E) birleşik, yönsüz ve her kenarının w fonksiyonu ile tanımlı bir maliyeti olan E üzerinde tanımlı bir graf olsun. A, E‘nin bir altkümesi ve bir en-küçük kapsar ağacın parçası olsun. (S, V – S) G‘nin A‘ya uyan bir kesiği olsun. (u,v), (S, V – S) kesiğini çaprazlayan bir hafif kenar olsun. Bu durumda (u,v) A için güvenlidir.
  - Bu bölümde göreceğimiz Kruskal ve Prim algoritmaları yukarıda belirtilen teoremi kullanır.

Graf algoritmaları

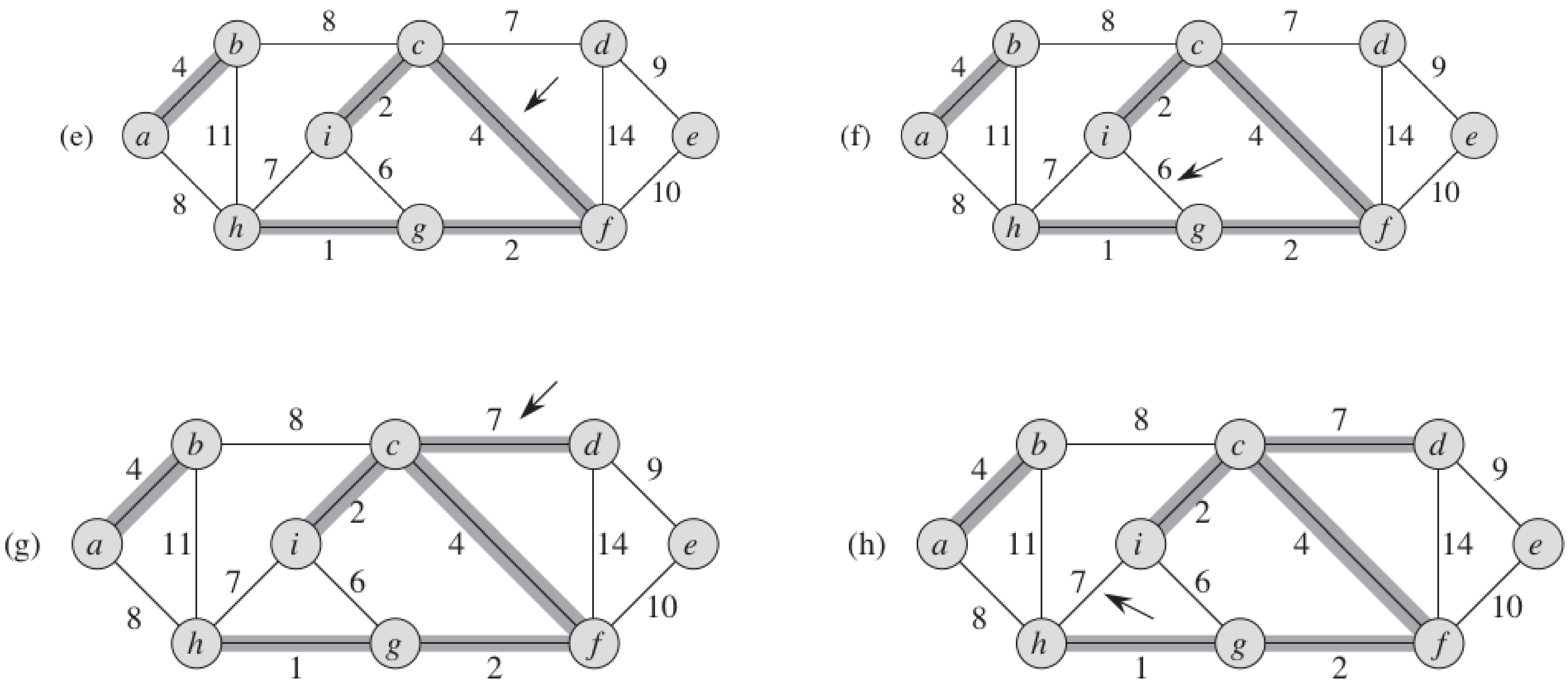
- Kruskal algoritması
  - Kruskal algoritması adım adım en-küçük kapsar ağacı oluşturur.
  - Bunu yaparken
    - İki farklı ağacı birleştiren en az maliyetli kenarı belirler.
    - Bu kenarı ağaca ekler.

```
MST-KRUSKAL(G,w)
A = ∅
for her v ∈ G.V kenarı için
  MAKE-SET(v)
G.E içerisindeki kenarları maliyetlerine göre azalmayan şekilde sırala
for maliyetlerine göre azalmayan sıralanmış her (u,v) ∈ G.E için
  if FIND-SET(u) ≠ FIND-SET(v)
    A = A U {(u,v)}
    UNION(u,v)
return A
```

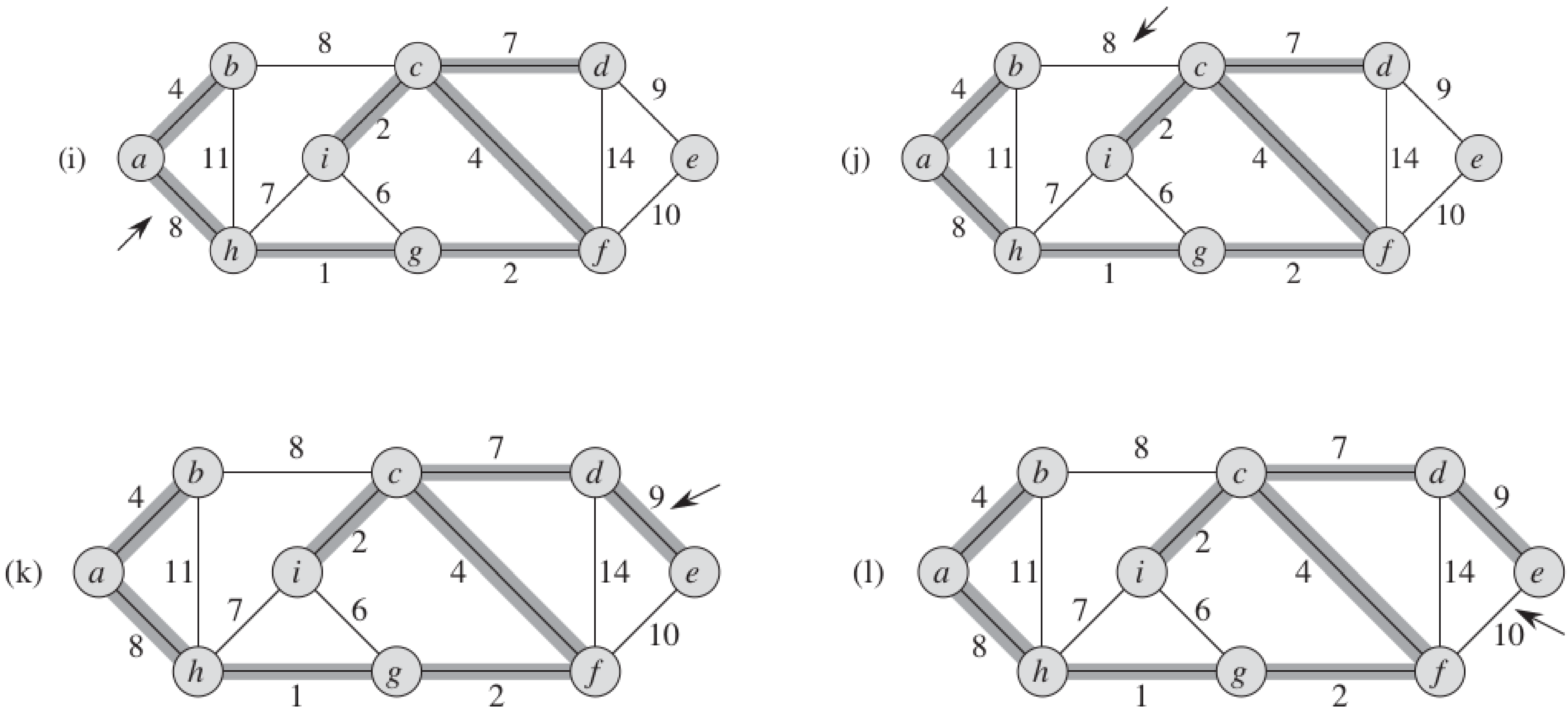
Graf algoritmaları



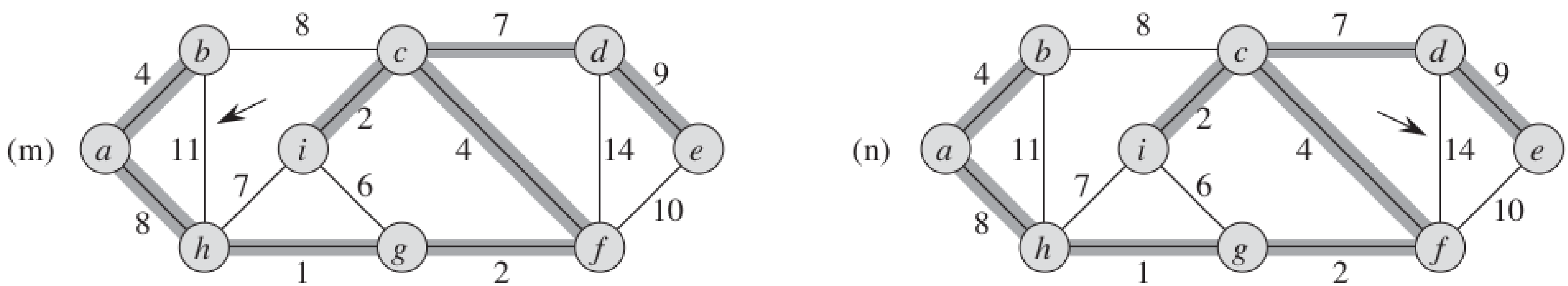
Graf algoritmaları



Graf algoritmaları



Graf algoritmaları



Graf algoritmaları

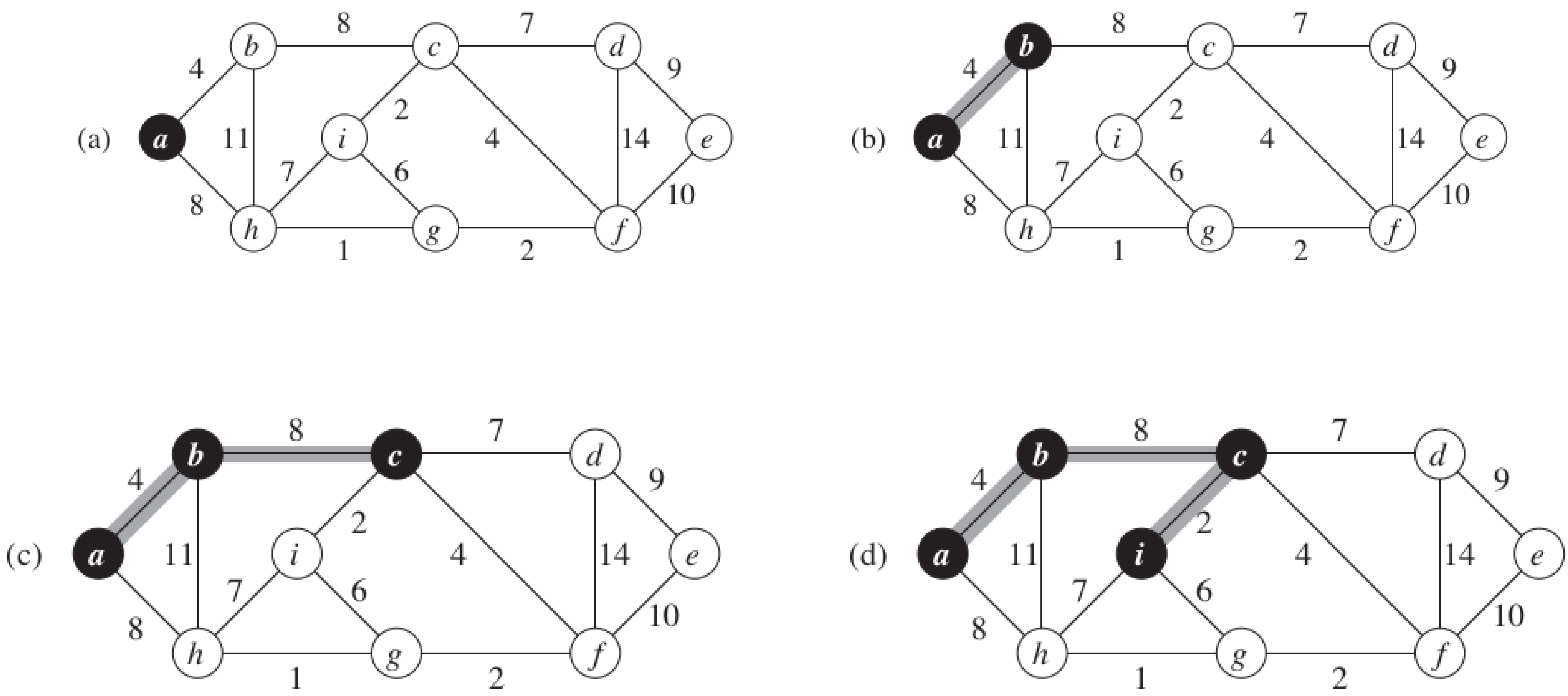
- Kruskal algoritması
  - Algoritmanın çalışma süresi birbirinden bağımsız-küme veri yapısını ne şekilde oluşturduğumuza bağlıdır.
    - 4. satırdaki sıralama O(E lgE)
    - 5-8'deki FIND-SET ve UNION ile 3'deki MAKE-SET
      - $O((V + E)\,\alpha(V))$
        - $\alpha(V)$ çok yavaş büyüyen bir fonksiyon olarak tanımlanmıştır.
      - G birleşik, yani $|E| \geq |V| - 1$
      - $O(E\alpha(V))$
      - $\alpha(|V|) = O(\lg V) = O(\lg E)$
    - Sonuç olarak toplam çalışma süresi O(E lg E)
    - $|E| < |V|^2$ olduğuna göre $\lg |E| = O(\lg V)$, öyleyse toplam çalışma süresini şu şekilde yazabiliriz.
      - O(E lg V)

Graf algoritmaları

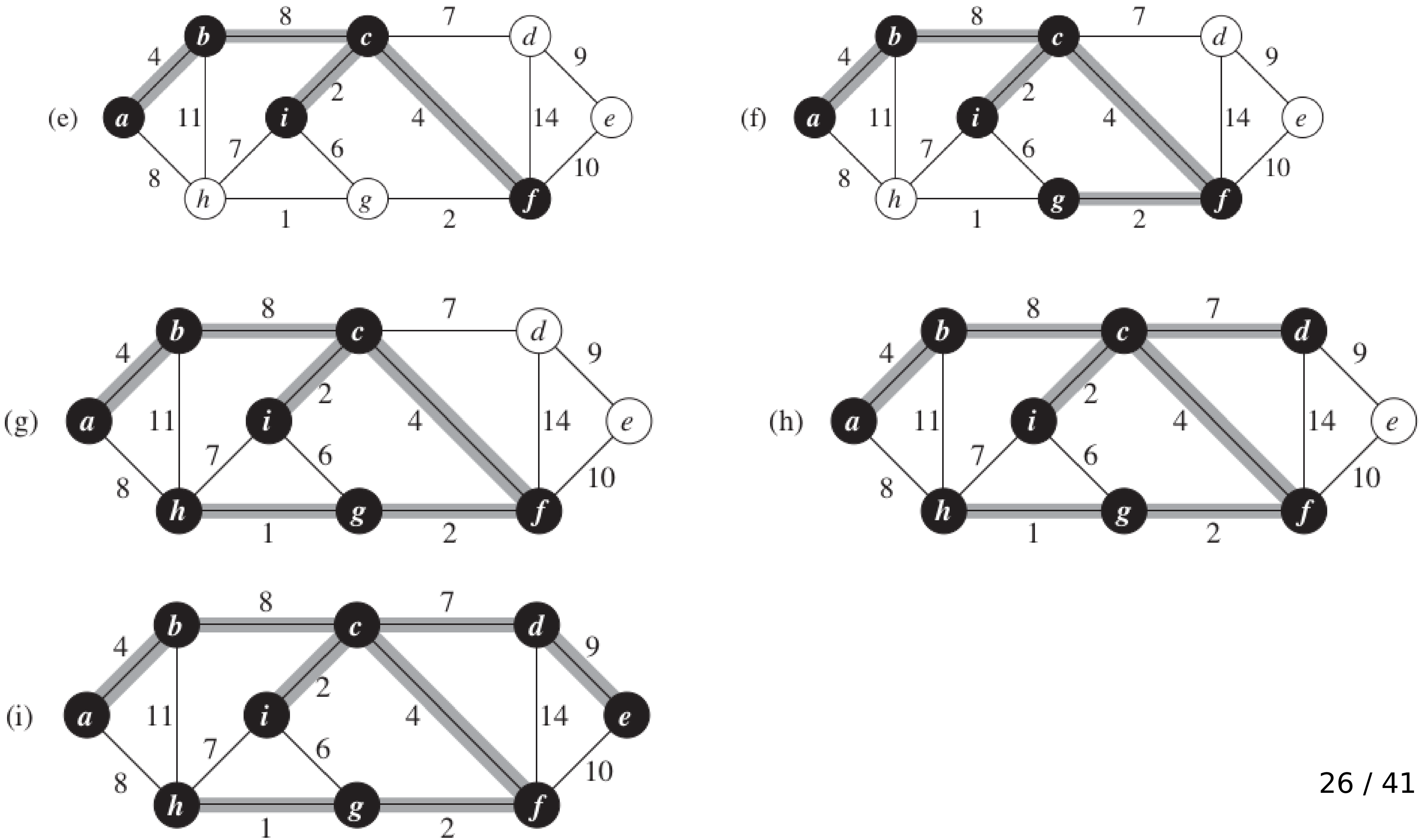
- Prim algoritması
  - Prim algoritması ile tek bir ağaç oluşturulur ve her adımda bu ağaca yeni kenarlar eklenir.
  - Tüm kenarlar eklenince ye kadar kenar eklemeye devam eder.
  - Her adımda A ağacına, henüz ağaca eklenmemiş kenarlardan biri, bir hafif kenar, eklenir.

```
MST-PRIM(G,w, r)
for her u ∈ G.V
  u.key = ∞
  u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
   u = EXTRACT-MIN(Q)
   for her v ∈ G.adj[u]
     if v ∈ Q and w(u,v) < v.key
        v.π = u
        v.key = w(u,v)
```

Graf algoritmaları



Graf algoritmaları



Graf algoritmaları

- Prim algoritması
  - Algoritmanın çalışma süresi min-öncelik sırasını oluşturma şeklimize göre farklılık gösterir.
  - BUILD-MIN-HEAP kullanırsak 1-5 satırları O(V)
  - While döngüsü |V| kez çalışıyor, EXTRACT-MIN O(lg V) süre alır
  - Öyleyse EXTRACT-MIN için toplam çağrı süres O(V lg V) olur.
  - 8-11 satırları arasındaki döngü O(E) süre alır.
  - 11. satırdaki işlem min-heap‘de DECREASE-KEY operasyonudur, bu operasyon O(lg V) süre alır.
  - Öyleyse toplam süre O(V lg V) + O(E lg V) = O(E lg V)
    - Kruskal algoritması ile asimtotik olarak aynıdır.

Graf algoritmaları

- En-kısa yol problemi
  - G = (V,E) şeklinde w : E → R şeklinde bir maliyet fonksiyonu ile tanımlanmış maliyet değerleri içeren bir grafımız olsun.
  - p = {v_0, v_1, v_2, v_3, ..., v_k} yolunun maliyeti, w(p), bu yol üzerindeki kenarların maliyetlerinin toplamıdır:
    - $w(p)=\sum_{i=1}^{k} w(v_{i-1}, v_i)$
  - u köşesinden v köşesine en kısa yol maliyeti, δ(u,v) şu şekilde tanımlanır:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{u ile v arasında yol varsa} \\ \infty & \text{aksi takdirde} \end{cases}$$

  - w(p) = δ(u,v) olan herhangi bir yol en-kısa yoldur.

Graf algoritmaları

- En-kısa yol türleri
  - Tek-hedef en-kısa yol problemi
    - Her bir v köşesinden hedef t köşesine en kısa yol bulunur.
  - Tek-ikili en-kısa yol problemi
    - Verilen iki köşe u ve v için, u‘dan v‘ye en kısa yol bulunur.
  - Her-ikili en-kısa yol problemi
    - Her u ve v köşeleri için u‘dan v‘ye en kısa yol bulunur.
- En-kısa yol özelliği
  - G = (V,E) şeklinde, maliyetleri w : E → R fonksiyonuyla hesaplanmış bir yönlü grafımız olsun. P = $\{v_0, v_1, \ldots, v_k\}$ v_0 köşesinden v_k köşesine en kısa yol olsun. Herhangi bir i, j ikilisi için, şöyle ki $0 \leq i \leq j \leq k$, $p_{ij} = \{v_i, v_{i+1}, \ldots, v_j\}$ v_i‘den v_j‘ye olan alt yol olsun. Bu durumda p_{ij}, v_i‘den v_j‘ye en kısa yoldur.

Graf algoritmaları

- Negatif-değerli kenarlar
  - Graf negatif değerli bir kenar içerebilir.
  - Ancak graf üzerinde kaynaktan ulaşılabilen maliyeti negatif olan bir zincir olmamalıdır.
    - Aksi takdirde durmadan bu zincir üzerinde hareket edilerek maliyet azaltılır.
    - u köşesinden v‘ye giden yol üzerinde negatif maliyete sahip bir zincir var ise
      - $\delta(u,v) = -\infty$
  - Bir en-kısa yol, pozitif bir zincir içeremez.
    - Bu zinciri izlemediğimizde daha kısa bir yola erişebiliriz.

Graf algoritmaları

- Genişletme (İng: relaxation)
  - Bu bölümde inceleyeceğimiz algoritmalar genişletme isimli bir teknik kullanmaktadırlar.
  - Bu amaçla her v köşesi için aşağı belirtilen bilgiler saklanır:
    - v.d → en-kısa uzaklık tahmini
    - v.π → en-kısa yol üzerinde bir önceki köşe
  - Algoritma çalışmaya başlarken ilk olarak bu değerler atanır.

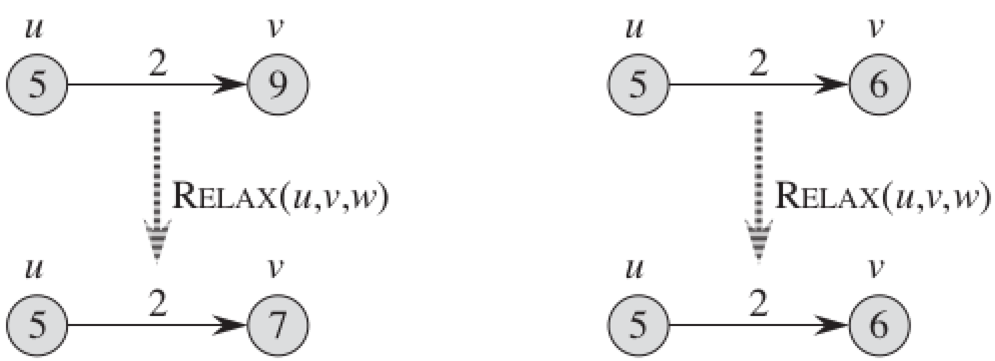
```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

Graf algoritmaları

- Genişletme
  - Genişletme işlemi ile bir v köşesine için yaptığımız en-kısa yol tahminini geliştirmeye çalışırız.

RELAX(u, v, w)

if $v.d > u.d + w(u, v)$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$



Graf algoritmaları

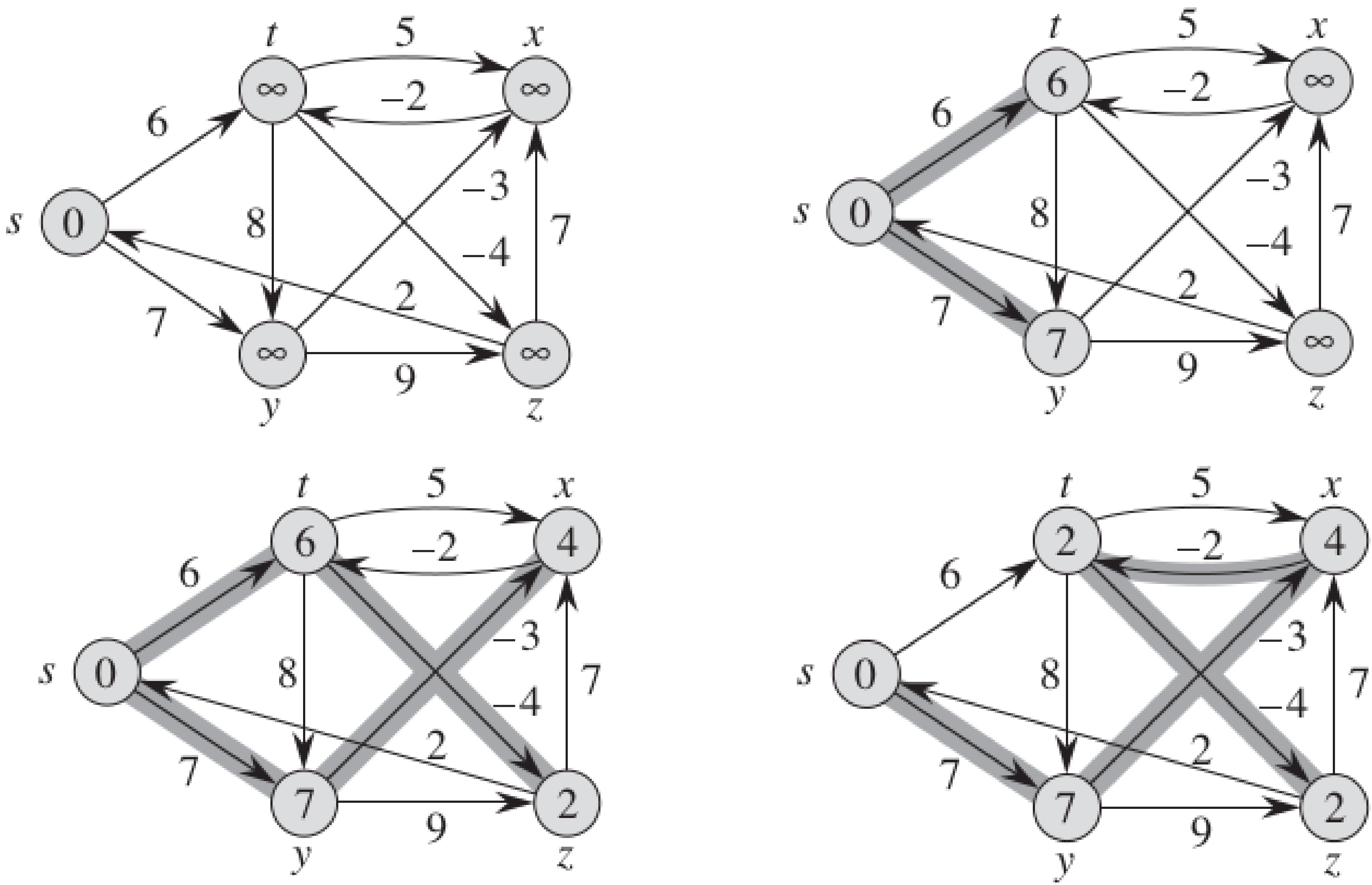
- Bellman-Ford algoritması
  - Tek-kaynak en-kısa yol problemini her tür maliyet içeren graflarda çözer
    - Graf negatif maliyetler içerebilir.
  - Verilen G = (V,E) grafı, s kaynağı ve $w : E \rightarrow R$ maliyet fonksiyonu ile Bellman-Ford algoritması grafta negatif değerli bir zincir olup olmadığını döner.
  - Eğer negatif değerli bir zincir var ise algoritma çözüm yoktur sonucunu verir.
  - Bu özellikte bir zincir yoksa en-kısa yollar ve bu yolların maliyetini döner.

Graf algoritmaları

- Bellman-Ford algoritması

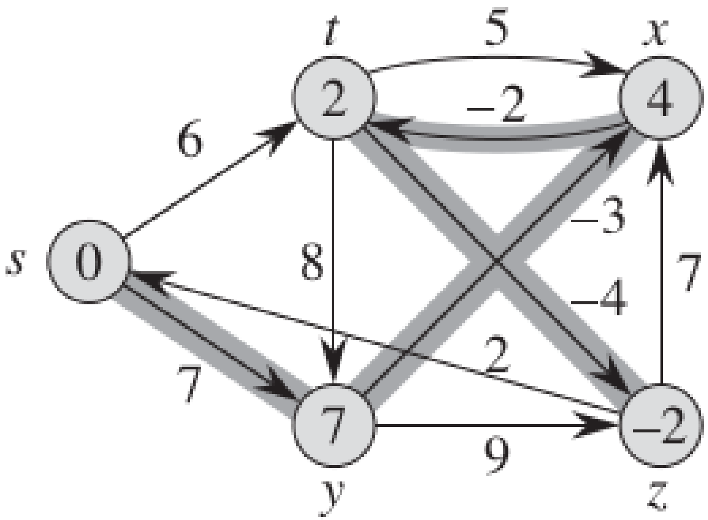
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

Graf algoritmaları



Örnekte kenarlar şu sıralya genişletiliyor: (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y).

Graf algoritmaları



Graf algoritmaları

- Bellman-Ford algoritması
  - Birinci satırdaki başlangıç $O(V)$ zaman alır.
  - 2-4 satırlarındaki her bir geçiş $\Theta(E)$ zaman alır, toplam $|V| - 1$ kez çalışır.
  - 5-7 satırlarındaki döngü $O(E)$ süre alır.
  - Toplam çalışma süresi $O(V E)$.

Graf algoritmaları

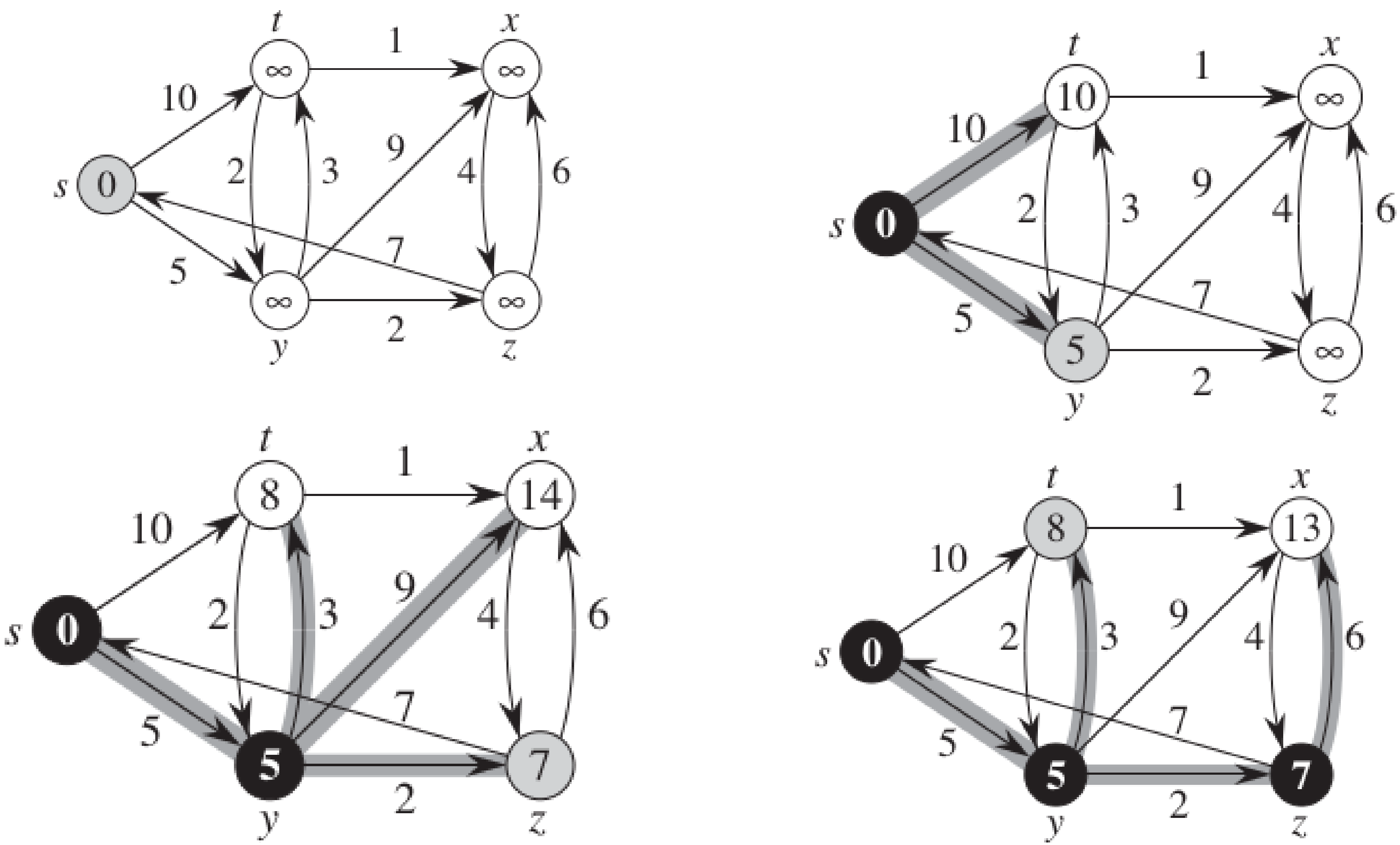
- Dijkstra algoritması
  - Her kenarı negatif-olmayan maliyet içeren yönlü graf G = (V,E) için tek-kaynak en-kısa yol hesabı yapar.
  - Doğru bir şekilde oluşturulduğunda Bellman-Ford algoritmasında daha verimli çalışır.
  - Son en-kısa yol uzunluğu bulunmuş köşeler kümesi, S, hesaplanır.
  - Algoritma tekrar ederek V – S içerisinde en-kısa yol tahminine sahip köşeyi seçer, bu köşeyi S kümesine ekler ve geri kalan, yeni eklenen köşeye komşu olan köşelerin her birini genişletir.

Graf algoritmaları

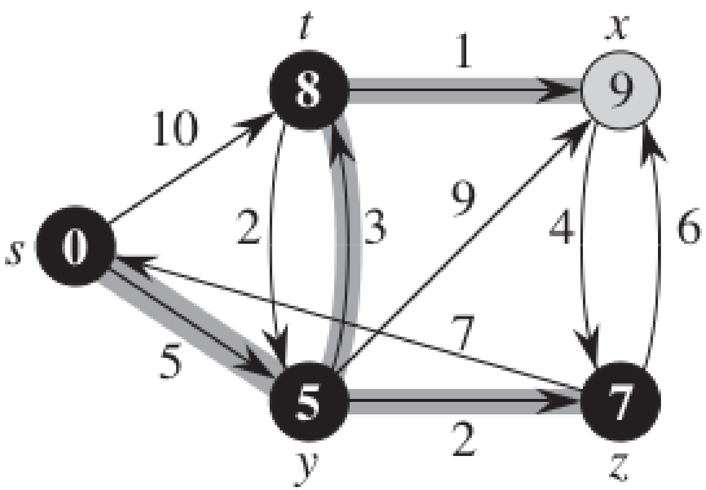
- Dijkstra algoritması

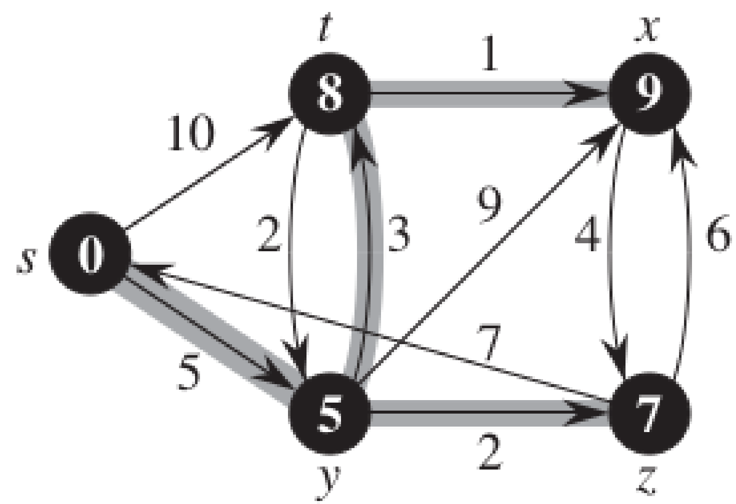
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Graf algoritmaları



Graf algoritmaları





Verimli bir Fibonacci heap yapısı ile Dijkstra‘nın algoritması O(V lg V + E) sürede çalışabilir.