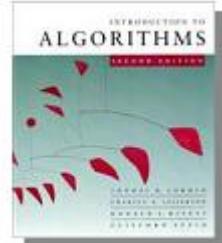


12.Hafta

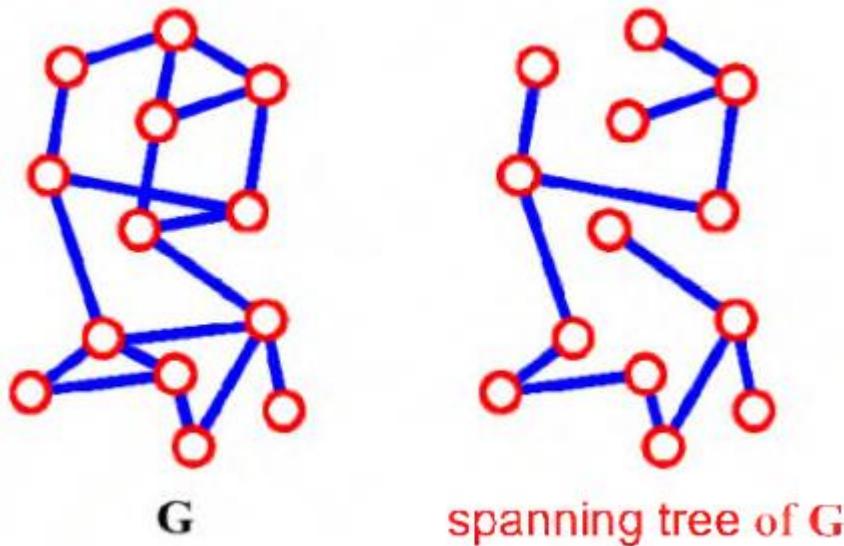
Minimum kapsayan ağaçlar

Minimum spanning trees (MST)

Kapsayan ağaç Spanning Tree (ST)

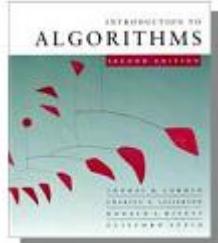


- Bir Kapsayan Ağaç (ST); G , grafındaki bir alt graftır ve aşağıdaki özelliklere sahiptir.
 - G grafındaki tüm düğümleri içerir
 - Bir ağaç yapısı oluşturur.



Minimum kapsayan ağaçlar

Minimum spanning trees (MST)

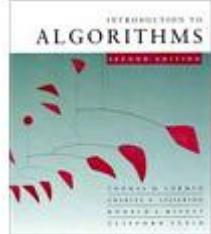


Girdi: $w : E \rightarrow \mathbb{R}$ ağırlık fonksiyonlu, $G = (V, E)$ bağlantılı, yönlendirilmemiş grafik.

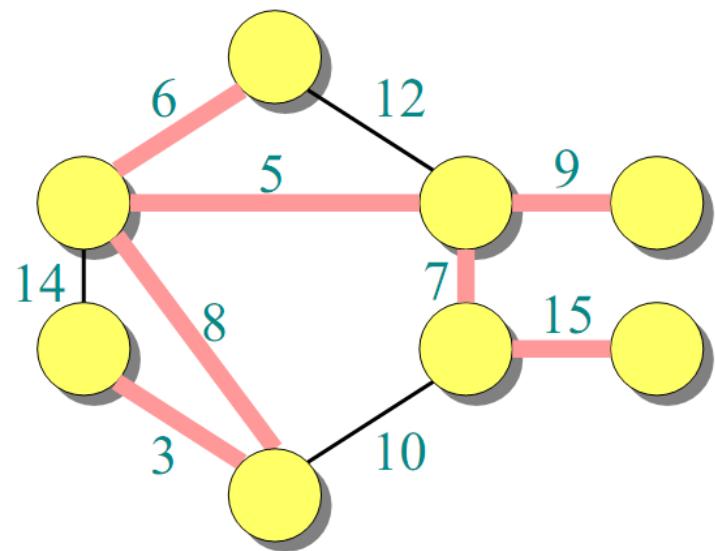
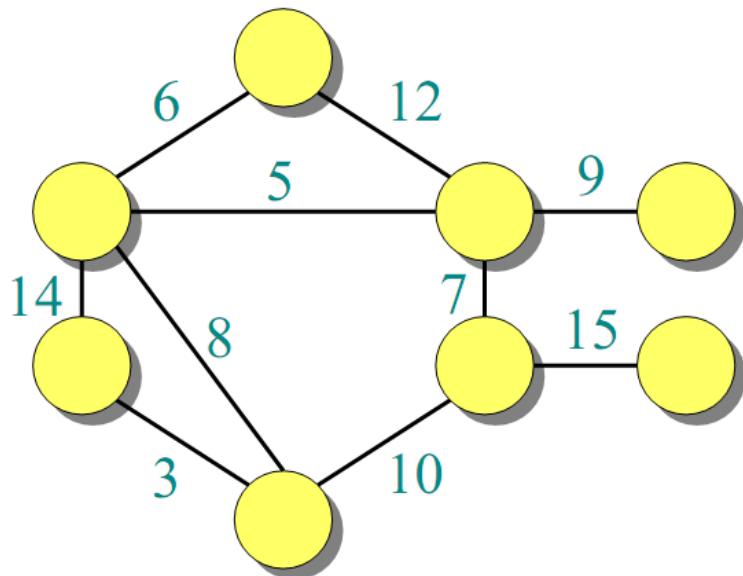
- Basitlik adına, farzedin ki tüm kenar ağırlıkları farklı olsun.

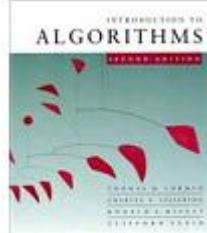
Çıktı: T kapsayan ağaç--en az ağırlıkla tüm köşeleri birleştiren bir ağaç:

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



MST (minimum kapsayan ağaç) örneği

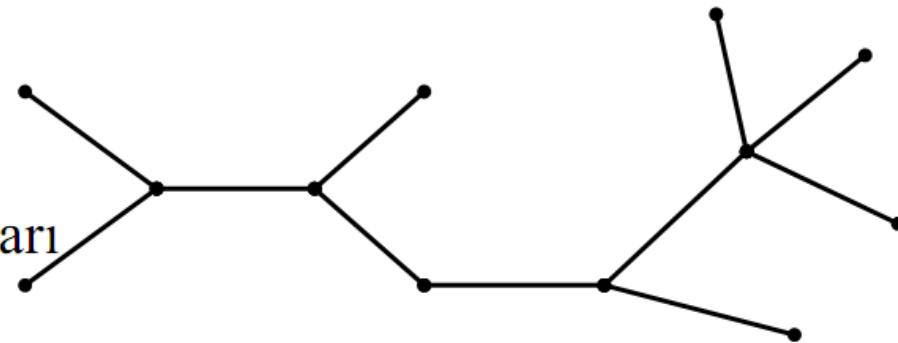




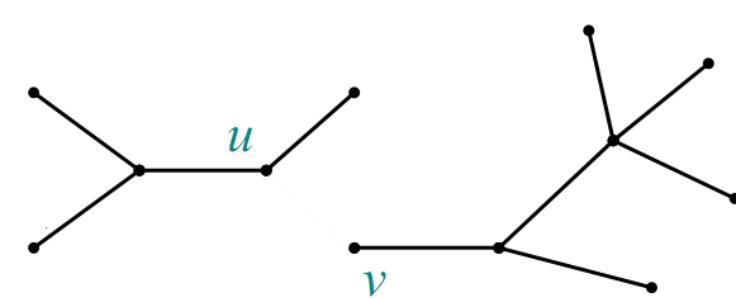
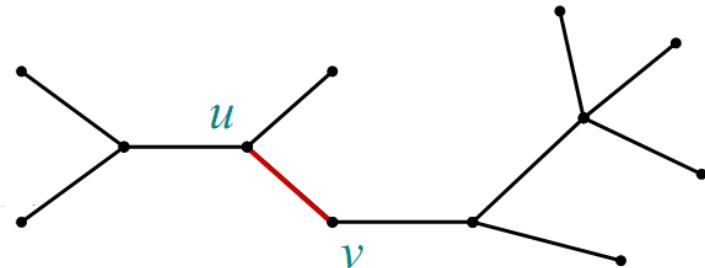
Optimal altyapı

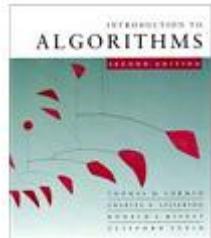
MST T :

(G' nin diğer kenarları gösterilmemiştir.)



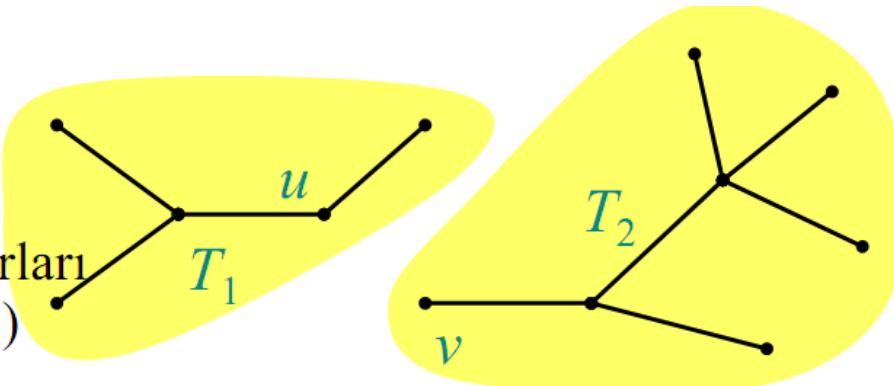
Herhangi bir $(u, v) \in T$ kenarını kaldır.





Optimal altyapı

MST T :
 $(G'$ nin diğer kenarları
gösterilmemiştir.)

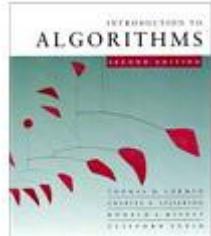


Herhangi bir $(u, v) \in T$ kenarını kaldırın. Bu durumda T , T_1 and T_2 olarak iki altağaca bölüntülenir.

Teorem. Altağac T_1 , $G_1 = (V_1, E_1)$ 'in bir MST'sidir ve T_1 'in köşeleri tarafından oluşturulan G' nin altgrafıgidir:

$$\begin{aligned} V_1 &= T_1 \text{' in köşeleri} \\ E_1 &= \{ (x, y) \in E : x, y \in V_1 \}. \end{aligned}$$

T_2 için de bu böyledir.



Optimal altyapının kanıtı

Kanıt. Kes ve yapıştır

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

Eğer T_1' G_1 için T_1 den daha az ağırlıklı bir kapsayan ağaçsa, bu durumda $T' = \{(u, v)\} \cup T_1' \cup T_2$ G için T' den daha az ağırlıklı bir kapsayan ağaç olur.

Aynı zamanda çakışan altproblemlerimiz de var mı?

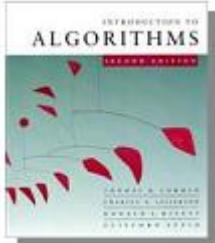
- Evet var.

Harika, o zaman dinamik programlama çalışabilir!

- Evet, fakat MST daha da verimli bir algoritmaya yol açan bir başka kuvvetli özelliğini sergiler.

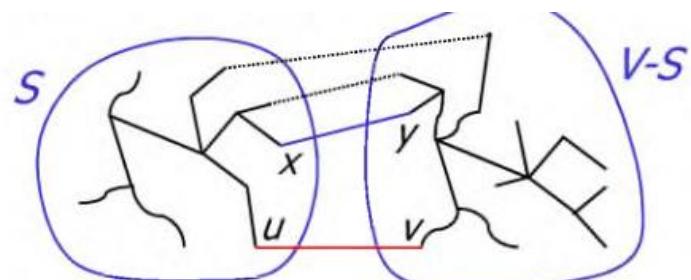
Greedy Yaklaşımı/Yöntemi

- Dolaşma yapılırken bir sonraki düğümü belirlemek için kullanılan bir karar verme/seçme yöntemidir.
- O andaki seçenekler içerisinde en iyi olarak gözükeni seçer.
- Bölgesel/yerel değerlendirmeler yapar.
- Yerel optimum daima global optimum anlamına gelmez dolayısıyla en iyi sonuca götürmeyebilir.
- Fakat bazı durumlarda en iyi sonuca götürür. (MST, en kısa yol alg. , Huffman coding)

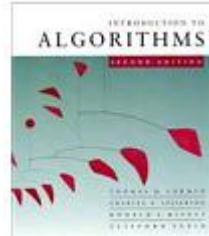


Greedy (Aç gözlü) Seçim

- MST'nin kenarları için $V-1$ tane seçim yapılması gerekmektedir.
- **Greedy seçim:** Lokal olarak optimal seçim (greedy) global olarak optimal çözümü oluşturur.
- **Teorem**
 - (u,v) düşük ağırlıklı bir kenar ancak $(u,v) \notin \text{MST}$ 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.

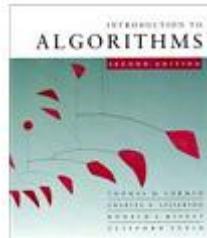


Açgözlü algoritmalar için Kalite işareteti



Açgözlü-seçim özelliği
Yerel olarak en uygun seçim genel olarak da en uygundur.

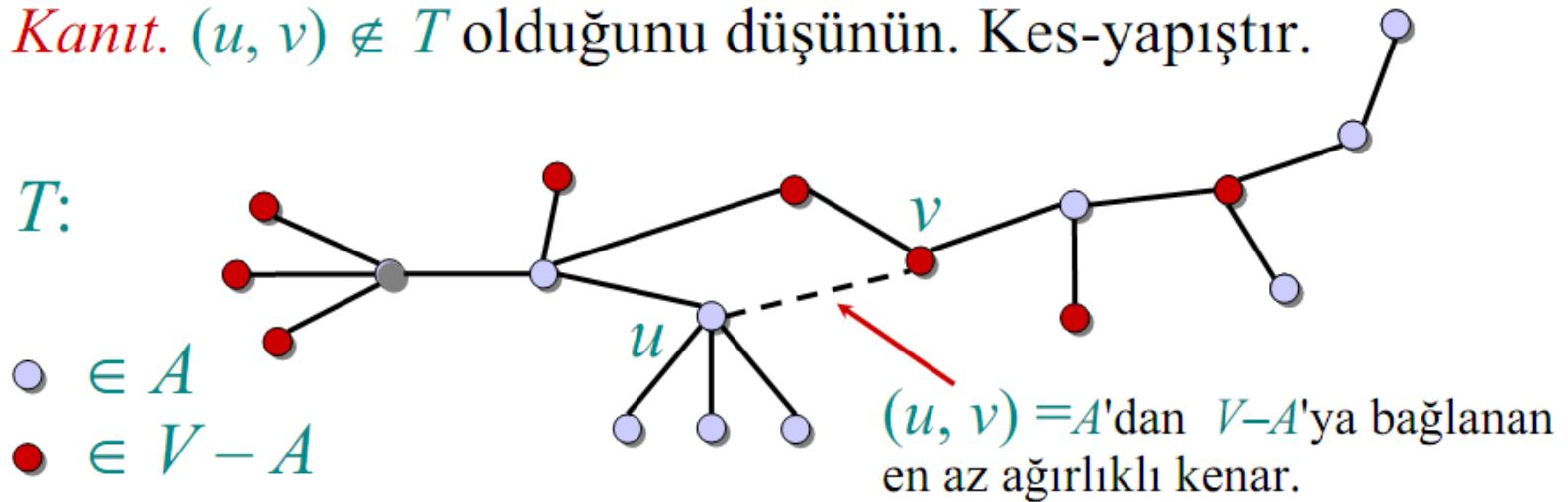
Teorem. T , $G = (V, E)$ 'nin MST'si olsun ve $A \subseteq V$ olsun. $(u, v) \in E$ 'nin A yi $V - A$ 'ya bağlayan en az ağırlıklı kenar olduğunu farzedin. O zaman $(u, v) \in T$ olur.

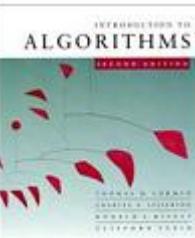


Teorem' in kanıtı

Kanıt. $(u, v) \notin T$ olduğunu düşünün. Kes-yapıştır.

T :

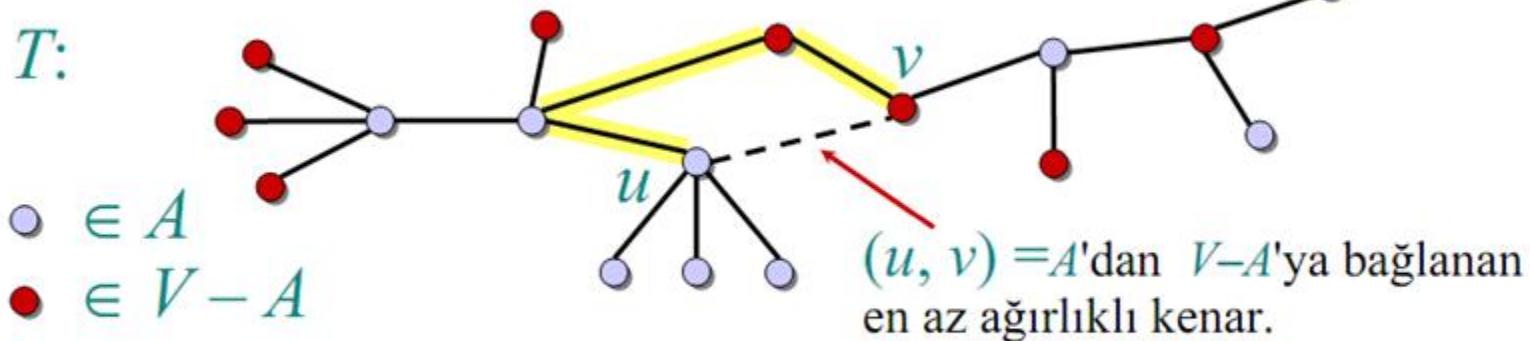




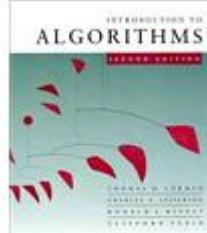
Teorem' in kanıtı

Kanıt. $(u, v) \notin T$ olduğunu düşünün. Kes-yapıştır.

T :



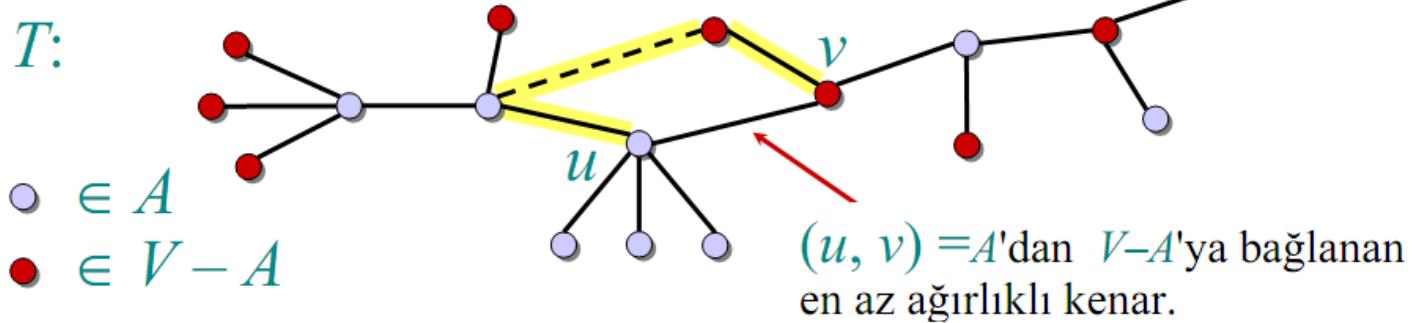
T' de u 'dan v 'ye benzeri olmayan basit yolu düşünün.



Teorem' in kanıtı

Kanıt. $(u, v) \notin T$ olduğunu düşünün. Kes-yapıştır.

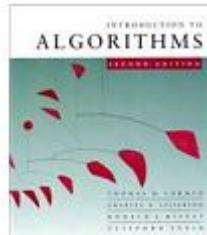
T :



T' de u 'dan v 'ye benzeri olmayan basit yolu düşünün.

(u, v) 'yi, bu yolda A' daki bir köşeyi

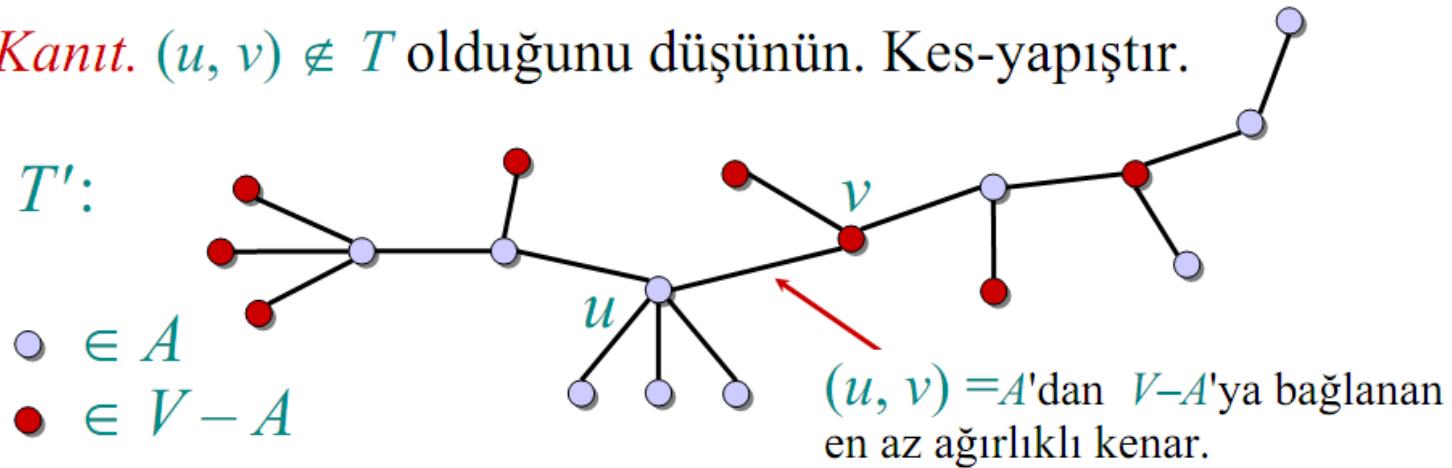
$V - A'$ daki bir köşeye bağlayan ilk kenarla değiştirin.



Teorem' in kanıtı

Kanıt. $(u, v) \notin T$ olduğunu düşünün. Kes-yapıştır.

T' :



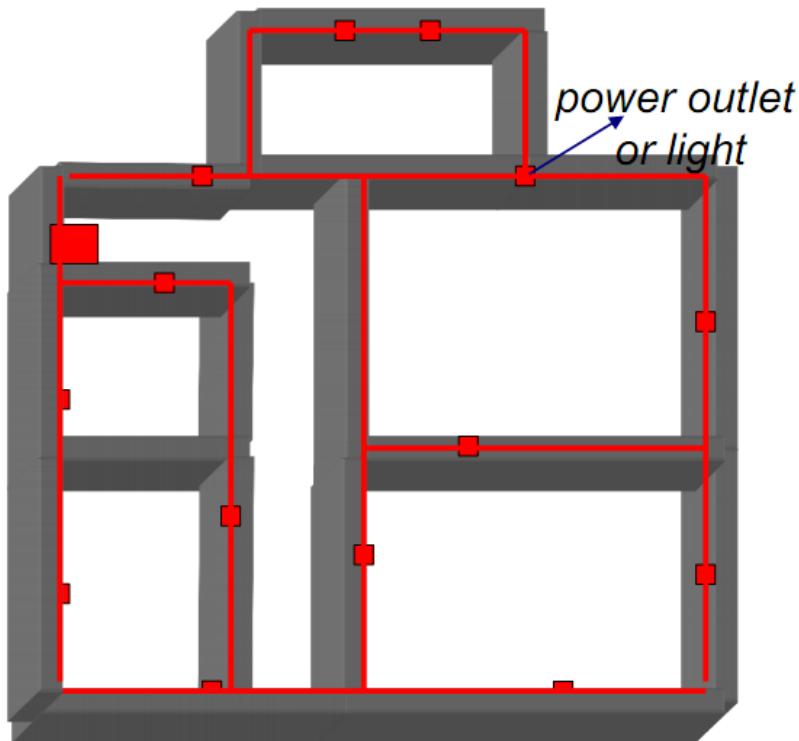
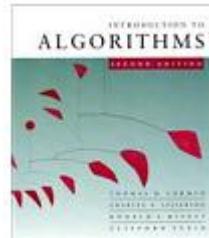
T' de u 'dan v 'ye benzeri olmayan basit yolu düşünün.

(u, v) 'yi, bu yolda A ' daki bir köşeyi

$V - A$ ' daki bir köşeye bağlayan ilk kenarla değiştirin.

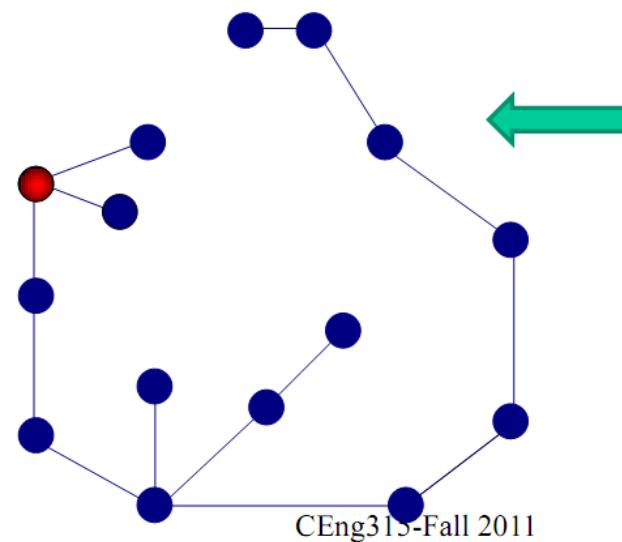
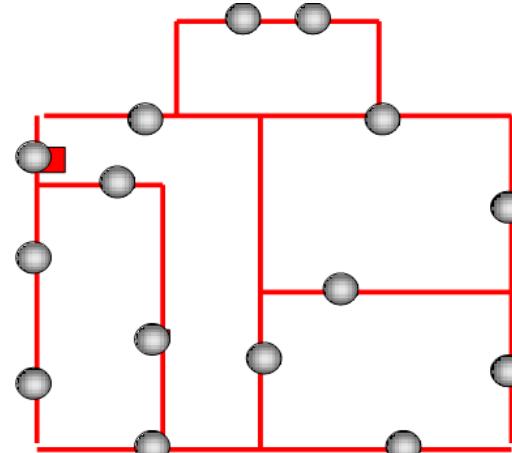
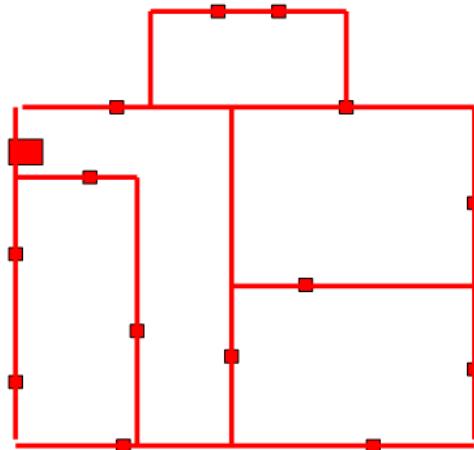
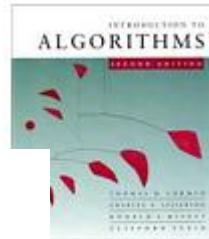
T 'den daha az ağırlıklı bir kapsayan ağaç oluşur. □

MST uygulaması için örnek

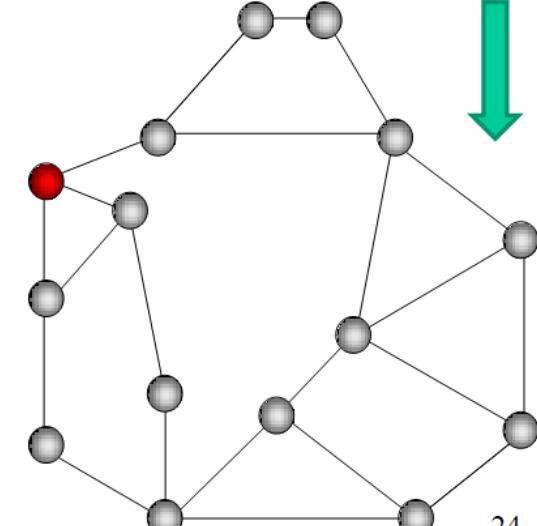


- Mininum kablo kullanarak bir evin elektriğini döseme

MST uygulaması için örnek

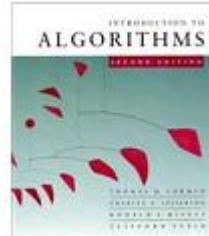


CEng315-Fall 2011



24

MST Algoritmaları



- En küçük yol ağacını belirlemek için birçok algoritma geliştirilmiştir.
 - **Prim'in Algoritması:** En az maliyetli kenardan başlayıp onun uçlarından en az maliyetle genişleyecek kenarın seçilmesine dayanır. Bir tane ağaç oluşur.
 - **Kruskal'ın Algoritması:** Daha az maliyetli kenarları tek tek değerlendирerek yol ağacını bulmaya çalışır. Ara işlemler birden çok ağaç oluşturabilir.
 - **Sollin'in Algoritması:** Doğrudan paralel programlamaya yatkındır. Aynı anda birden çok ağaçla başlanır ve ilerleyen adımlarda ağaçlar birleşerek tek bir yol ağacına dönüşür.

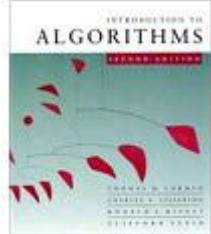
Prim'in Algoritması

- Düğüm tabanlı bir algoritmadır. Bir T ağacını her adımda bir düğüm ekleyerek büyütür.
- Bir kök düğüm ile başlar ve bütün düğümleri içine alıncaya kadar ağaç büyütür.
- **Adım-1:** Başlangıçta, herhangi bir noktayı ağaç oluşturmaya başlamak için seç.
- **Adım-2:** Oluşturulan ağaç'a eklemek için, şu ana kadar oluşturulmuş **ağaç üzerinden erişilebilen** ve daha önceden ağaç'a katılmamış olan **en küçük ağırlıklı kenarı** seç.
- **Adım-3:** Eğer bu kenarın ağaç'a katılması, bir **çember oluşmasına sebep olmuyorsa**, ağaç'a ekle.
- **Adım-4:** Ağaçtaki kenar sayısı **(N-1)**'e ulaşana kadar ikinci adıma geri dön.

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

```

MST-PRIM( $G, w, r$ )
1 for each  $u \in V[G]$ 
2   do  $key[u] \leftarrow \infty$ 
3    $\pi[u] \leftarrow \text{NIL}$ 
4    $key[r] \leftarrow 0$ 
5    $Q \leftarrow V[G]$ 
6   while  $Q \neq \emptyset$ 
7     do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
8       for each  $v \in \text{Adj}[u]$ 
9         do if  $v \in Q$  and  $w(u, v) < key[v]$ 
10           then  $\pi[v] \leftarrow u$ 
11              $key[v] \leftarrow w(u, v)$ 
```



Prim'in Algoritması

Fikir: $V - A'$ 'yı bir Q öncelikli sırası olarak koruyun.
 Q' daki her köşeyi, A' daki bir köşeye bağlayan en az ağırlıklı kenarın ağırlığıyla KEYleyin. (anahtarlayın)

$Q \leftarrow V$

$key[v] \leftarrow \infty$ tüm $v \in V$ 'ler için

$key[s] \leftarrow 0$ rastgele $s \in V$ 'için

(-iken) **while** $Q \neq \emptyset$

(yap) **do** $u \leftarrow \text{EXTRACT-MIN}(\text{en küçükü çıkar})(Q)$

her $v \in Adj[u]$ için

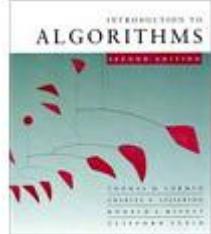
(yap-eğer) **do if** $v \in Q$ ve $w(u, v) < key[v]$

(sonra) **then** $key[v] \leftarrow w(u, v)$

$\pi[v] \leftarrow u$

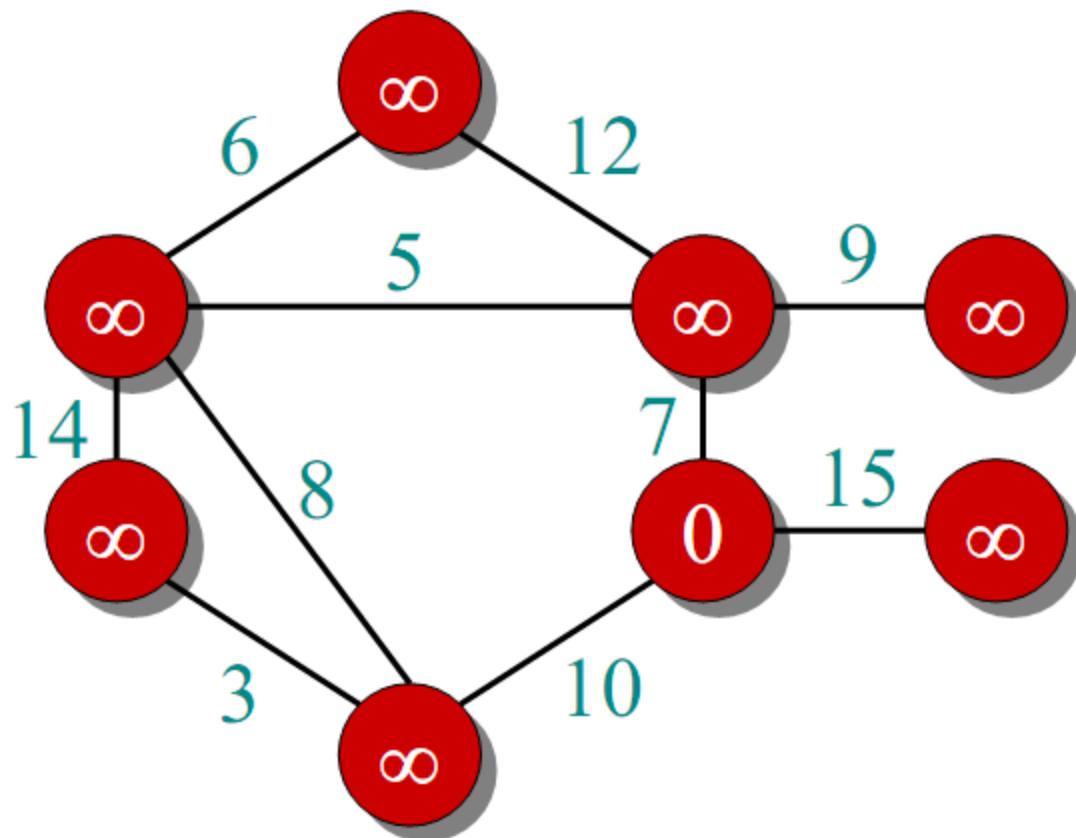
► DECREASE-KEY
 (anahtarı küçült)

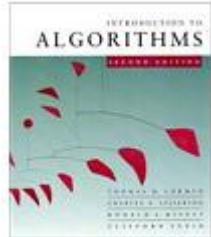
Sonunda, $\{(v, \pi[v])\}$, MST' yi biçimlendirir.



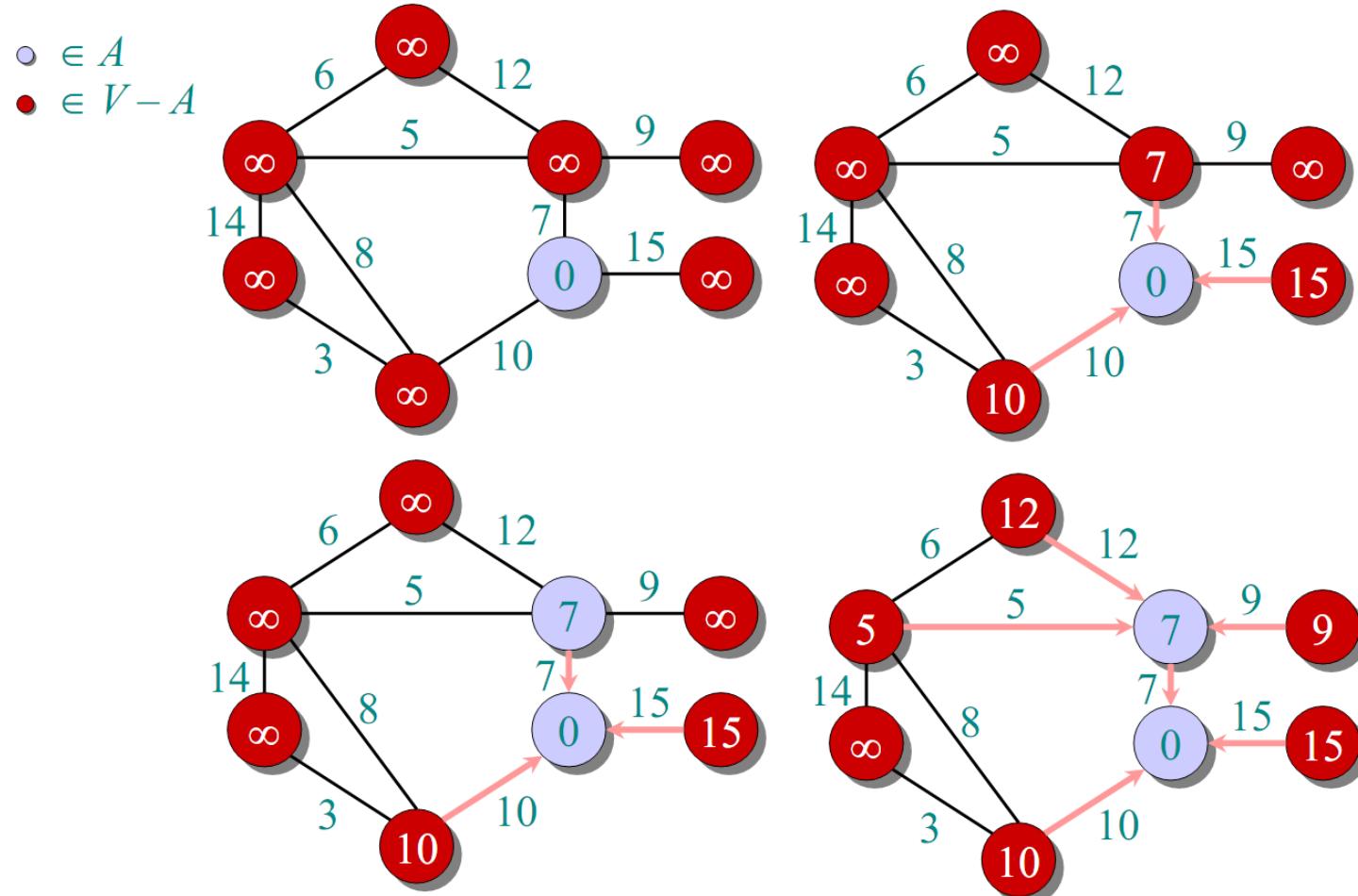
Prim'in algoritmasına örnek

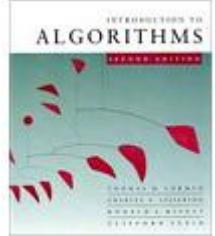
- $\in A$
- $\in V - A$



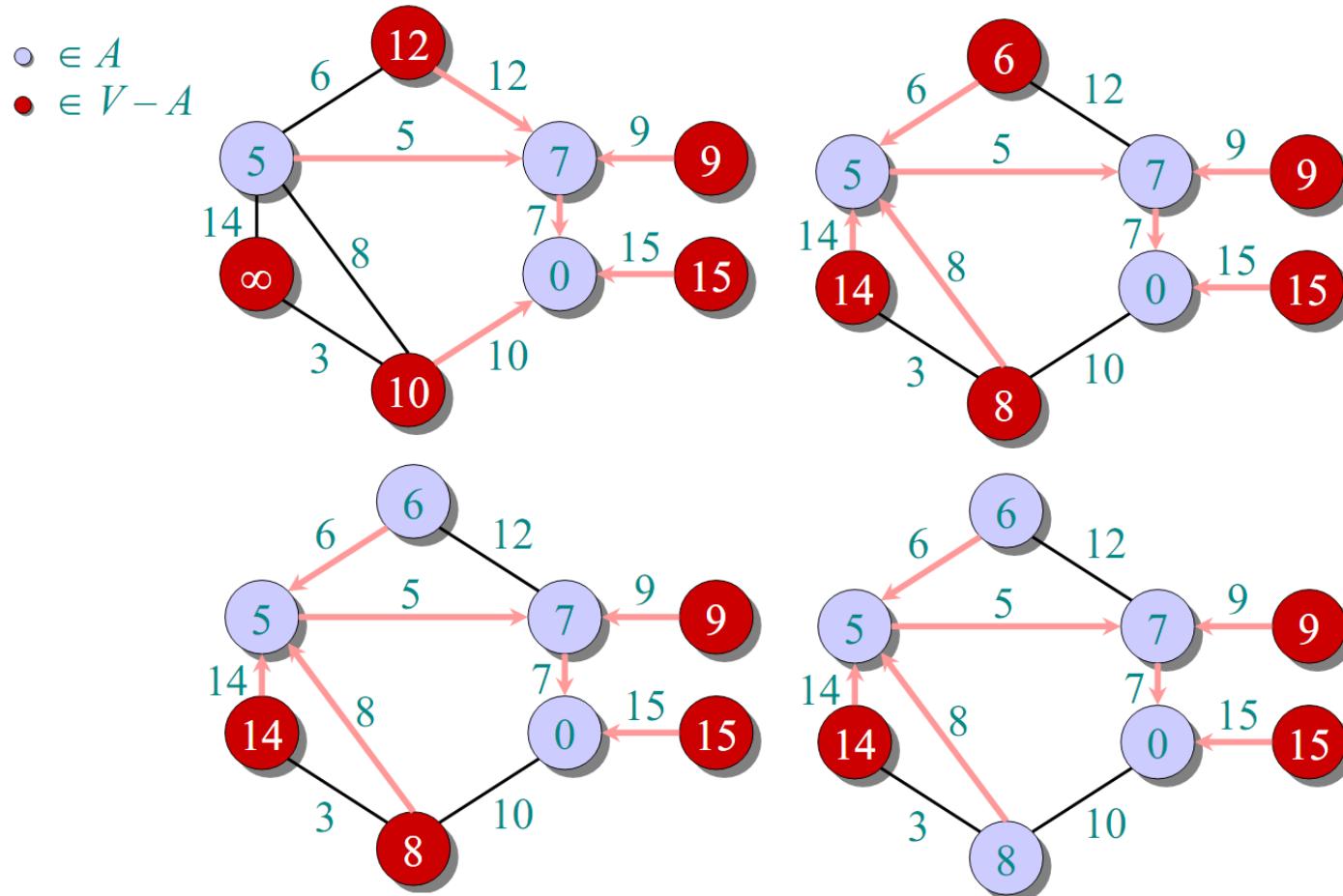


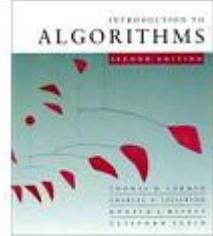
Prim'in algoritmasına örnek



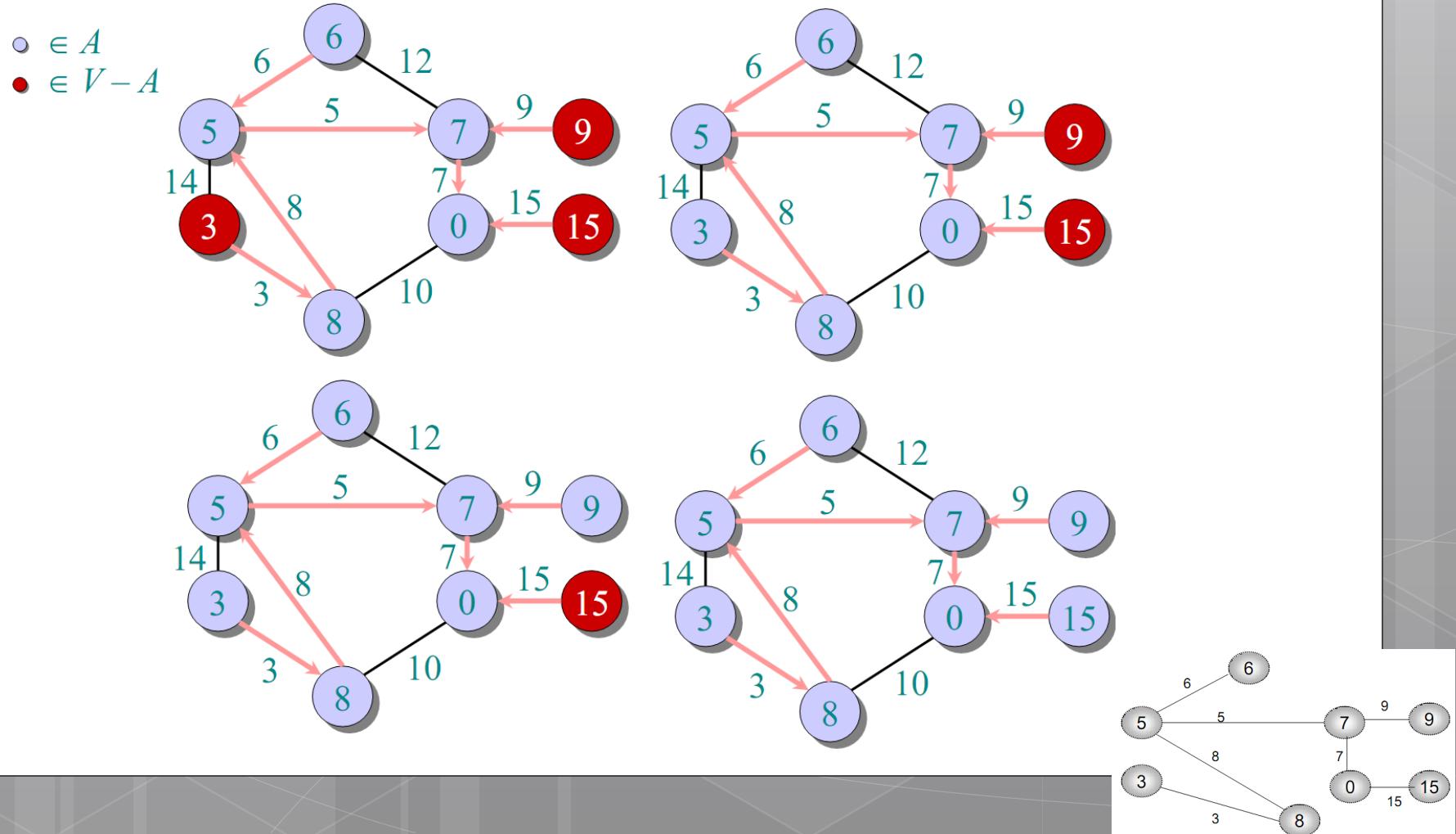


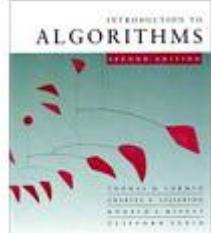
Prim'in algoritmasına örnek





Prim'in algoritmasına örnek



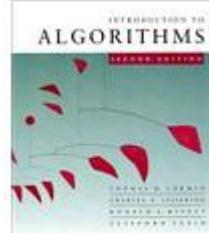


Prim' in çözümlemesi (analizi)

$\Theta(V)$ toplam $\left\{ \begin{array}{l} Q \leftarrow V \\ \text{key}[v] \leftarrow \infty \text{ tüm } v \in V \text{ ler için.} \\ \text{key}[s] \leftarrow 0 \text{ rastgele } s \in V \text{ ler için.} \end{array} \right.$
 $|V|$ kere $\left\{ \begin{array}{l} (-\text{iken}) \textbf{while } Q \neq \emptyset \\ \quad (\text{yap}) \textbf{do } u \leftarrow \text{En az } (Q) \text{ yu çıkar.} \\ \quad \quad \text{her } v \in \text{Adj}[u] \text{ için} \\ \quad \quad \quad \textbf{do if } v \in Q \text{ ve } w(u, v) < \text{key}[v] \\ \quad \quad \quad \quad (\text{yap-eğer}) \textbf{then } \text{key}[v] \leftarrow w(u, v) \\ \quad \quad \quad \quad (\text{sonra}) \quad \pi[v] \leftarrow u \end{array} \right.$

Tokalaşma önkuramı $\Rightarrow \Theta(E)$ varsayılan DECREASE-KEY'ler.
 (Anahtarı küçült)

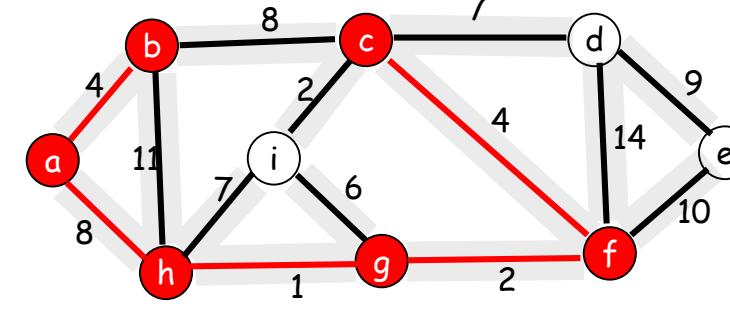
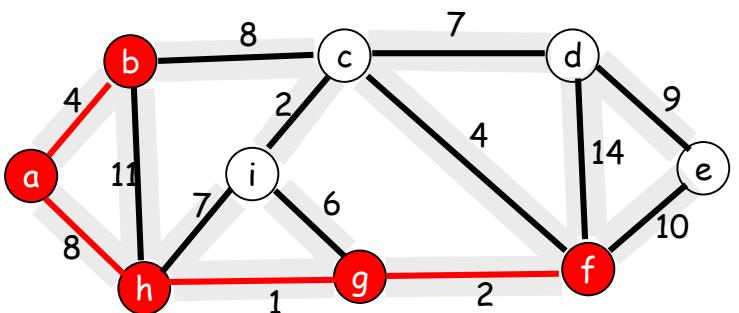
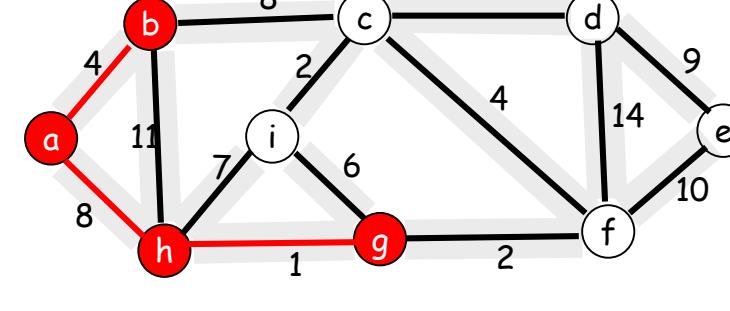
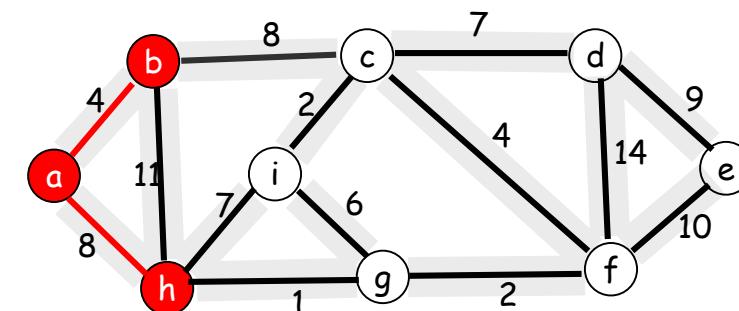
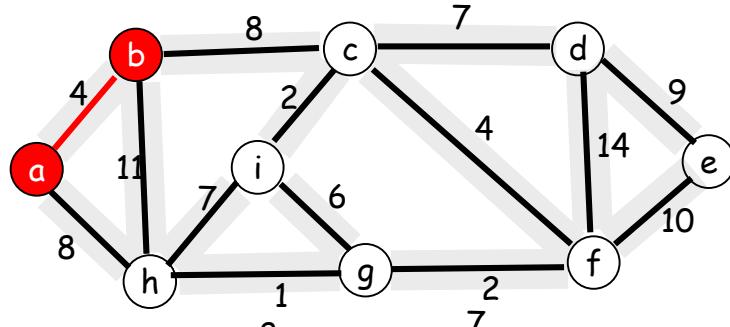
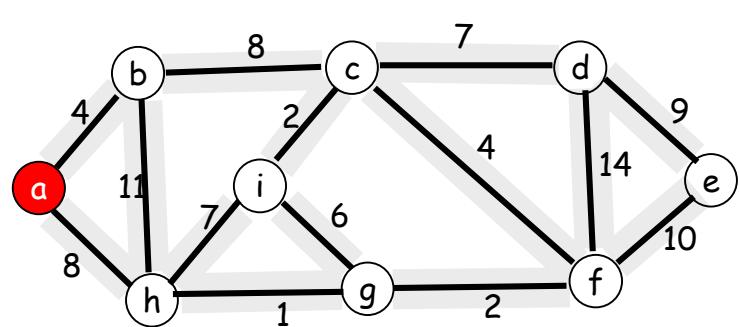
Time(süre) = $\Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$
 (en küçükü çıkar)



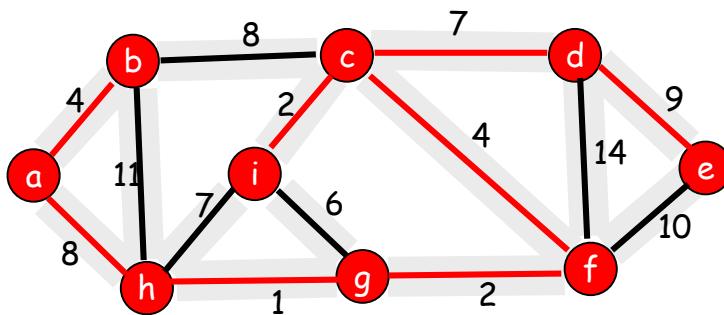
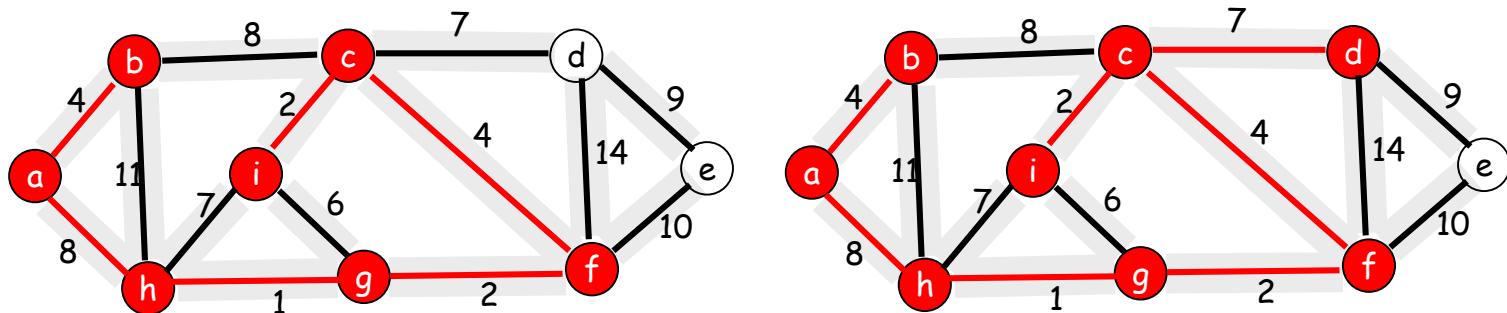
Prim' in çözümlemesi (devamı)

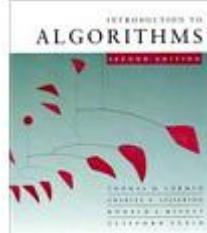
	Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Toplam
array	dizilim	$O(V)$	$O(1)$	$O(V^2)$
binary heap	ikili yığın	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	Fibonacci yığını	$O(\lg V)$ amortize edilmiş	$O(1)$ amortize edilmiş	$O(E + V \lg V)$ worst case en kötü durum

Prim'in Algoritması: Örnek



Prim'in Algoritması: Örnek

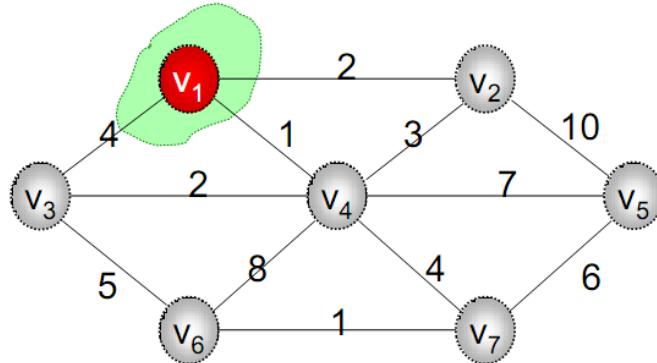




Prim'in algoritmasına örnek-Sınav

Prim's Algorithm: Example

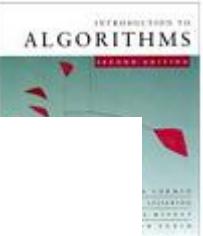
- Start with vertex v_1 . It is the initial current tree which we will grow to an MST



V ₁	0
----------------	---

Content of the priority queue

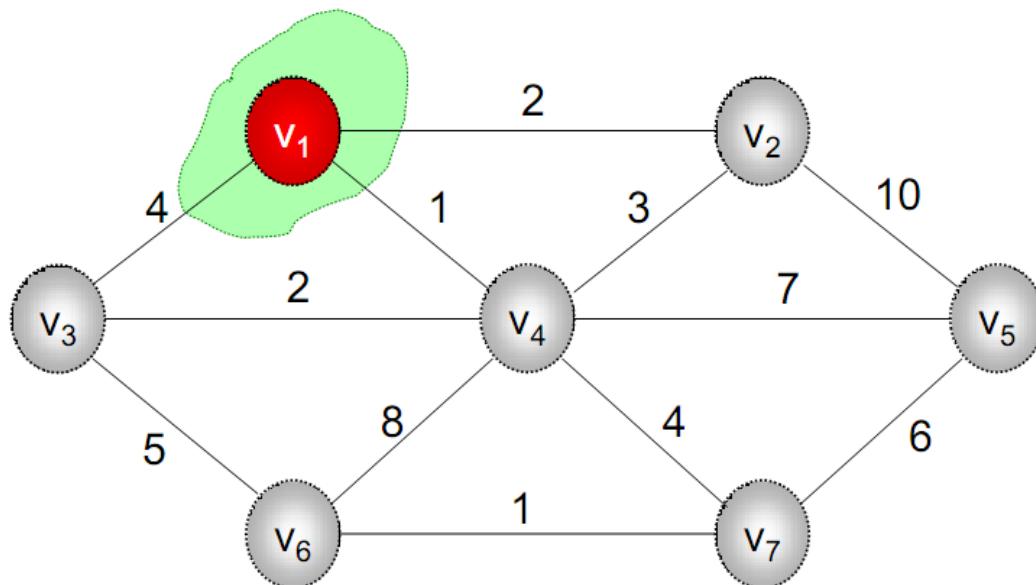
A connected, undirected graph G is given above.



Prim's Algorithm: Example

Step 1

Select an edge from graph: that is not in the current tree, that has the minimum cost, and that can be connected to the current tree.



V_4	1
V_2	2
V_3	4

Content of the priority queue

Prim's Algorithm: Example

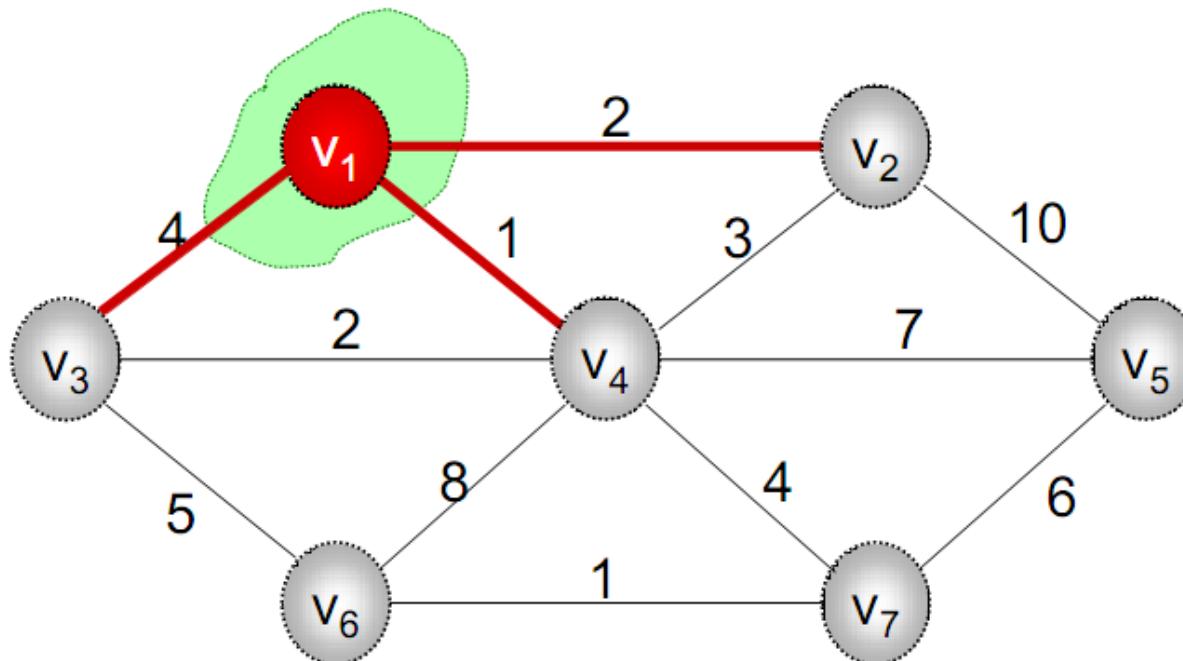
Step 1

The edges that can be connected are:

(v_1, v_2) : cost 2

(v_1, v_4) : cost 1

(v_1, v_3) : cost 4



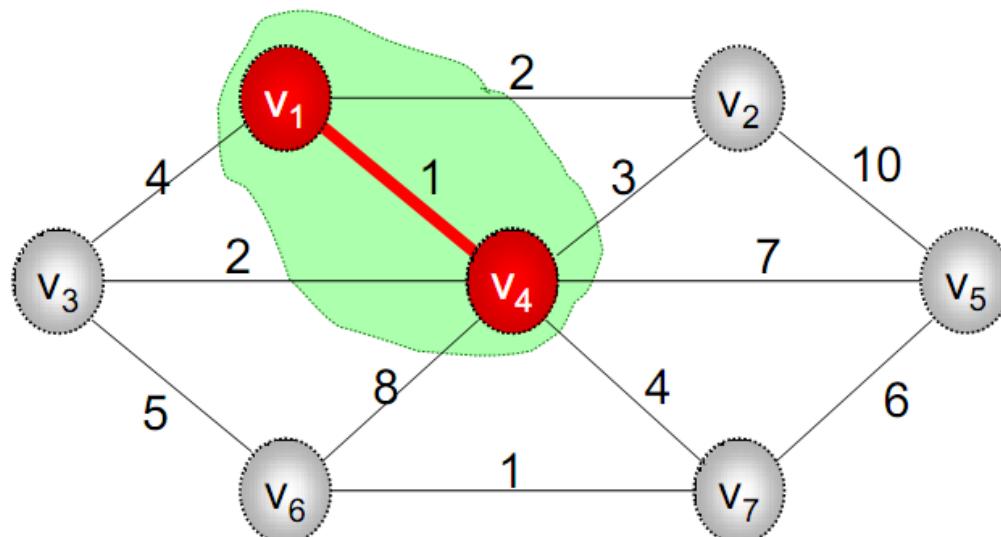
V_4	1
V_2	2
V_3	4

Content of the priority queue

Prim's Algorithm: Example

Step 2

We include the vertex v_4 , that is connected to the selected edge, to the current tree. In this way we grow the tree.

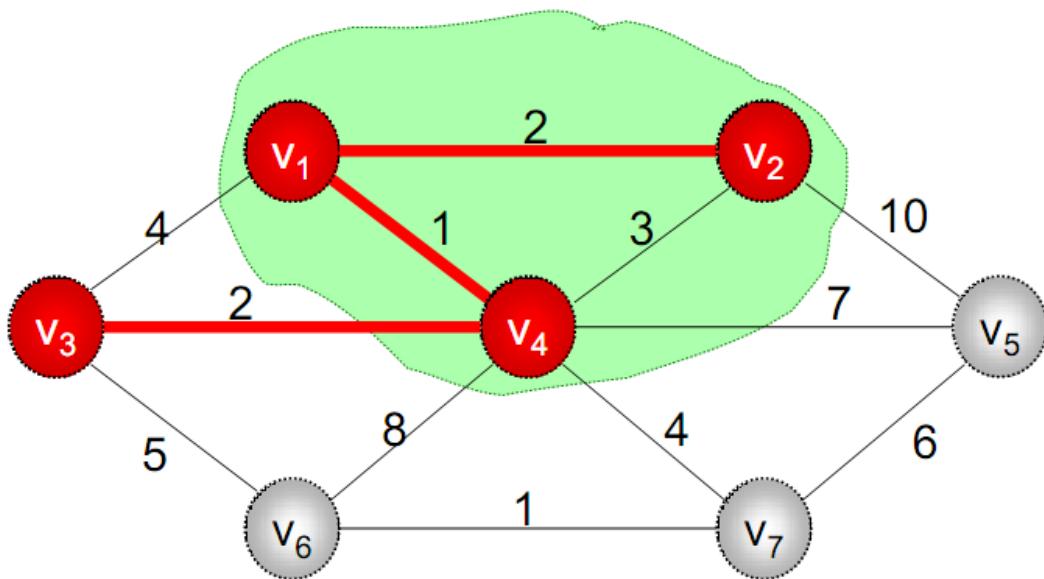


V_2	2
V_3	2
V_7	4
V_5	7
V_6	8

Content of the
priority queue

Prim's Algorithm

*Repeat steps: 1, and 2
Add either edge (v_4, v_3)*

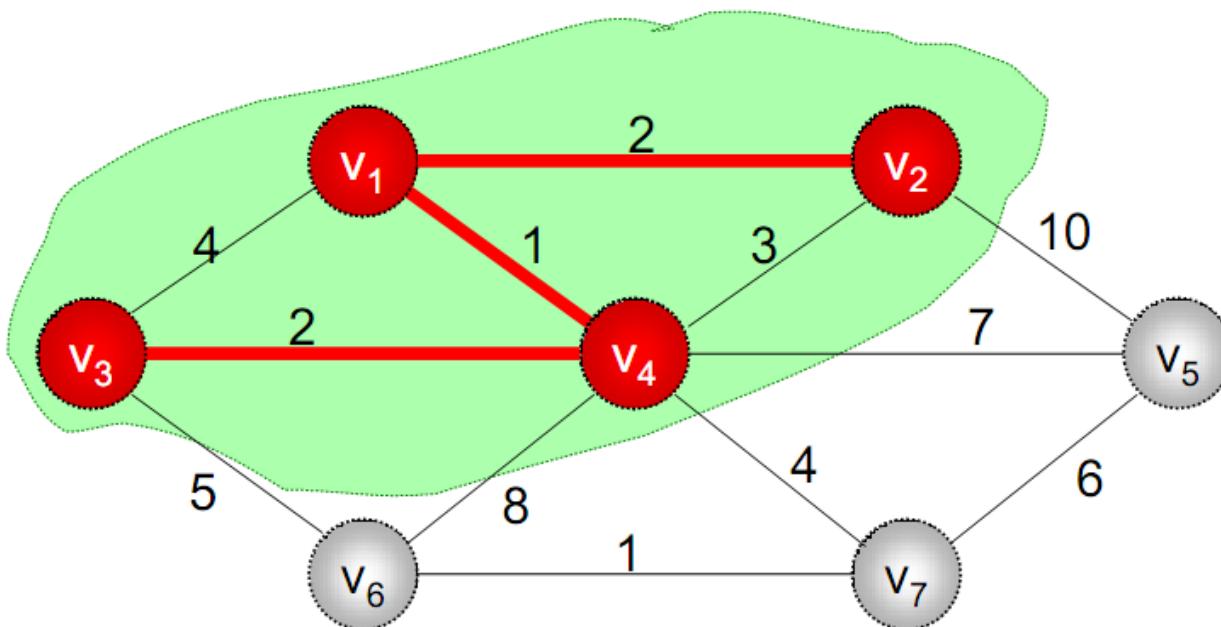


V ₃	2
V ₇	4
V ₅	7
V ₆	8

Content of the priority queue

Prim's Algorithm

Grow the tree!

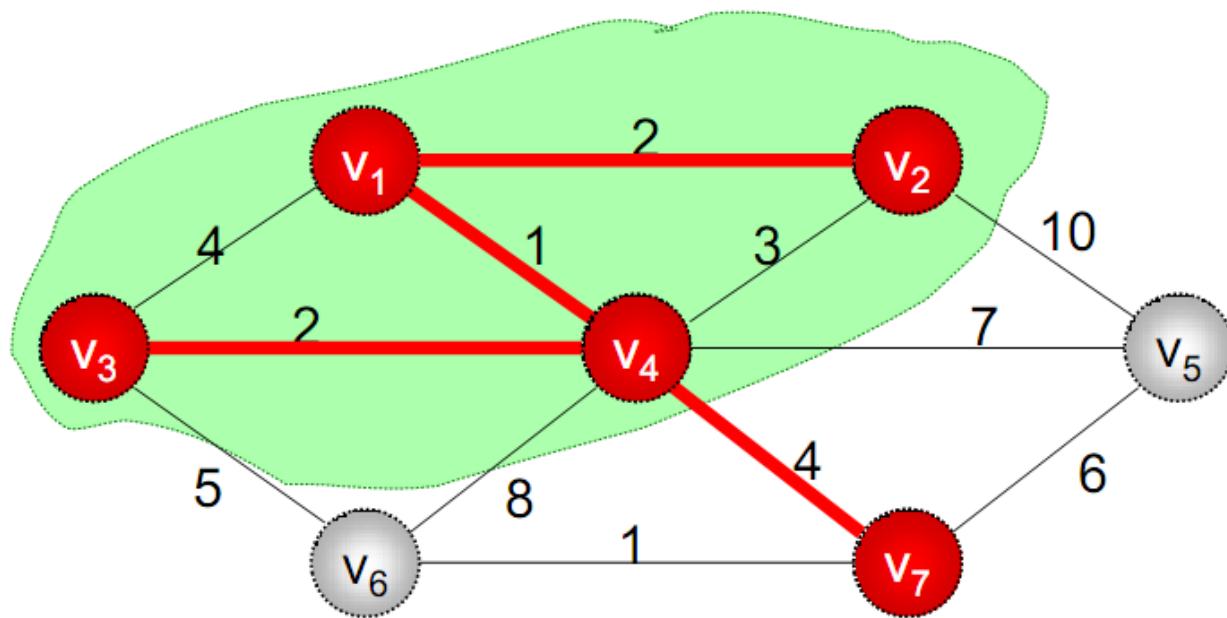


V_7	4
V_6	5
V_5	7

Content of the priority queue

Prim's Algorithm

Add edge (v_4, v_7)

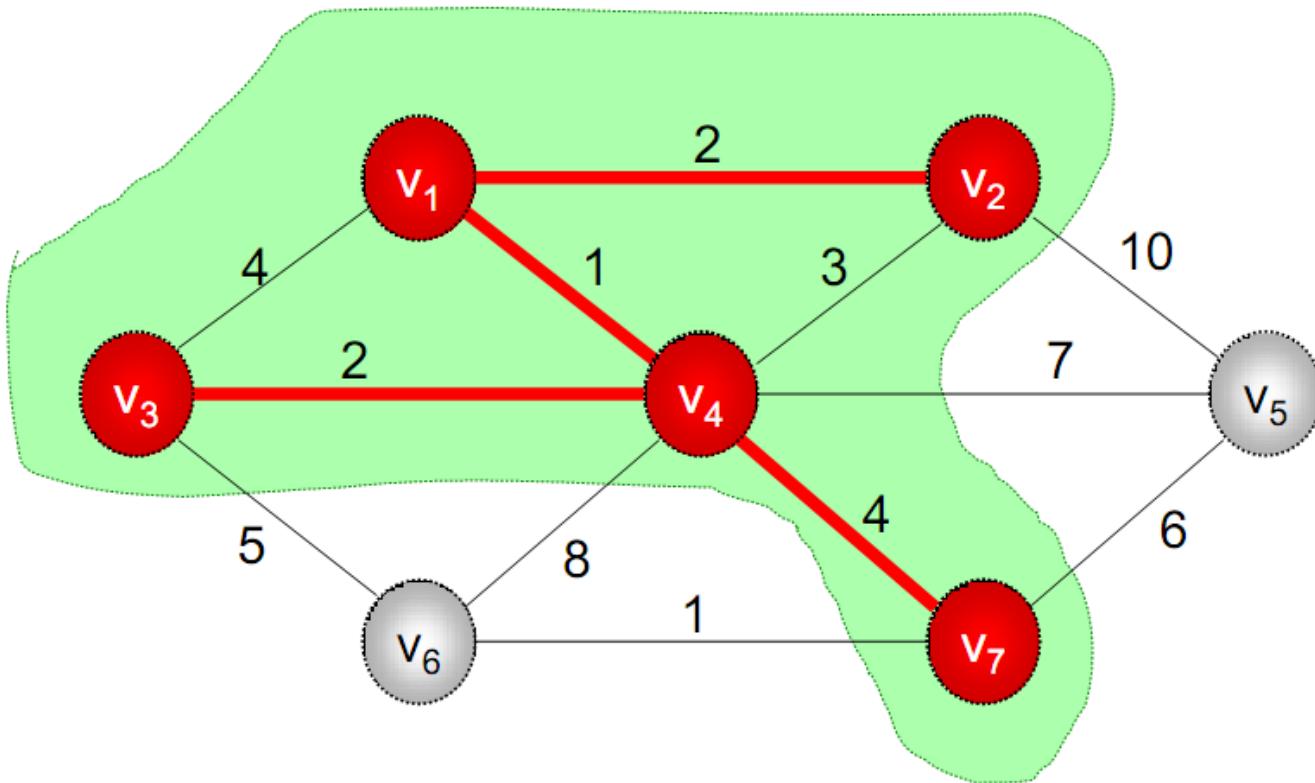


V ₇	4
V ₆	5
V ₅	7

Content of the
priority queue

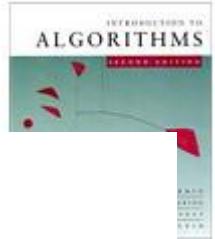
Prim's Algorithm

Grow the tree!



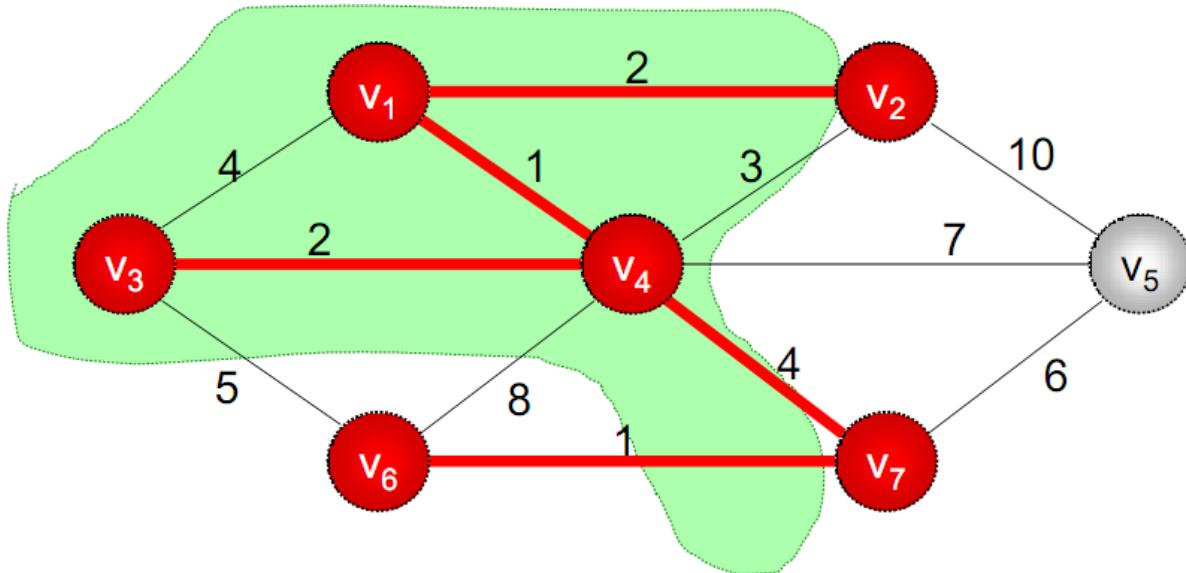
V_6	1
V_5	6

Content of the priority queue



Prim's Algorithm

Add edge (v_7, v_6)

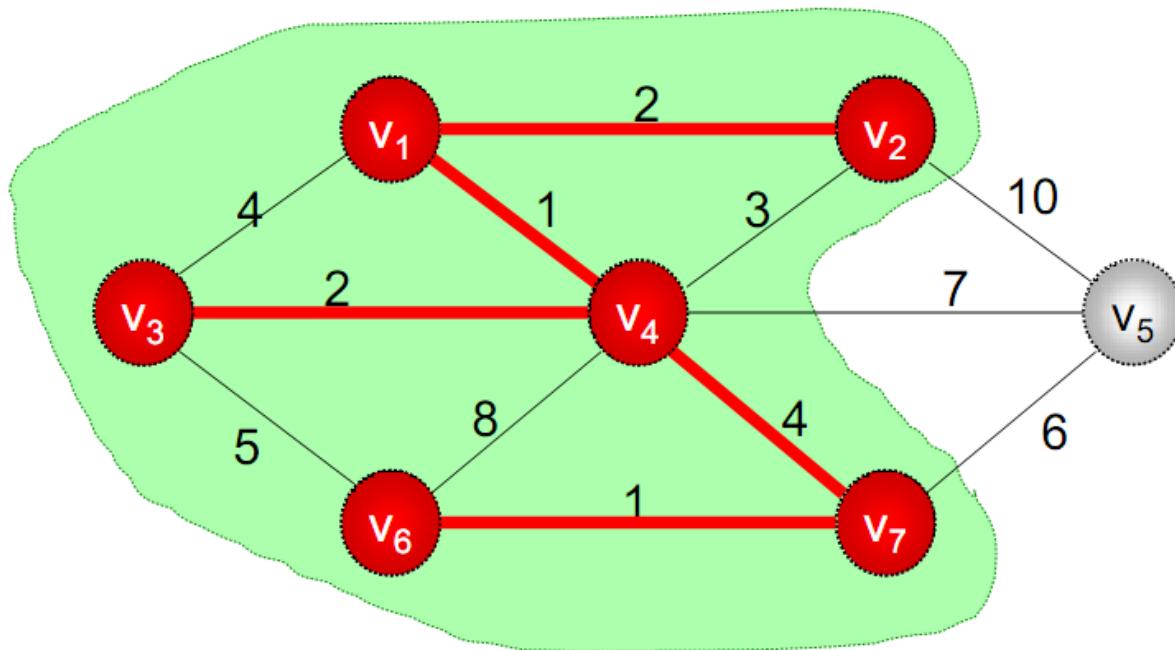


V ₆	1
V ₅	6

Content of the
priority queue

Prim's Algorithm

Grow the tree!

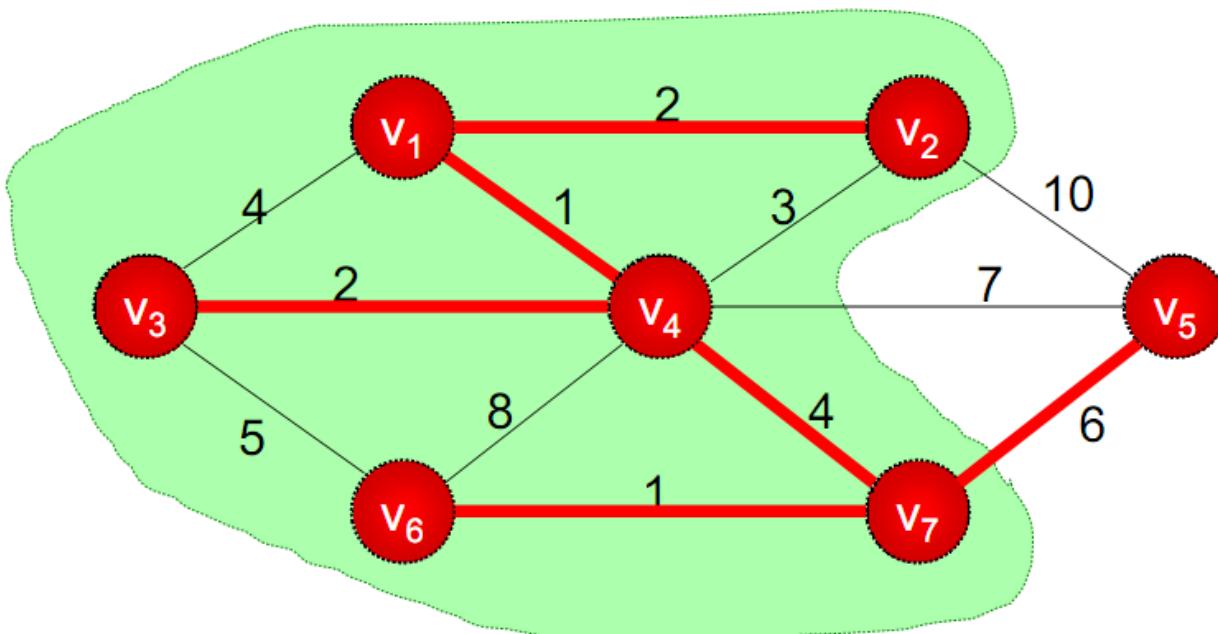


V ₅	6

Content of the
priority queue

Prim's Algorithm

Add edge (v_7, v_5)

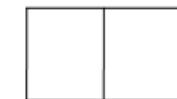
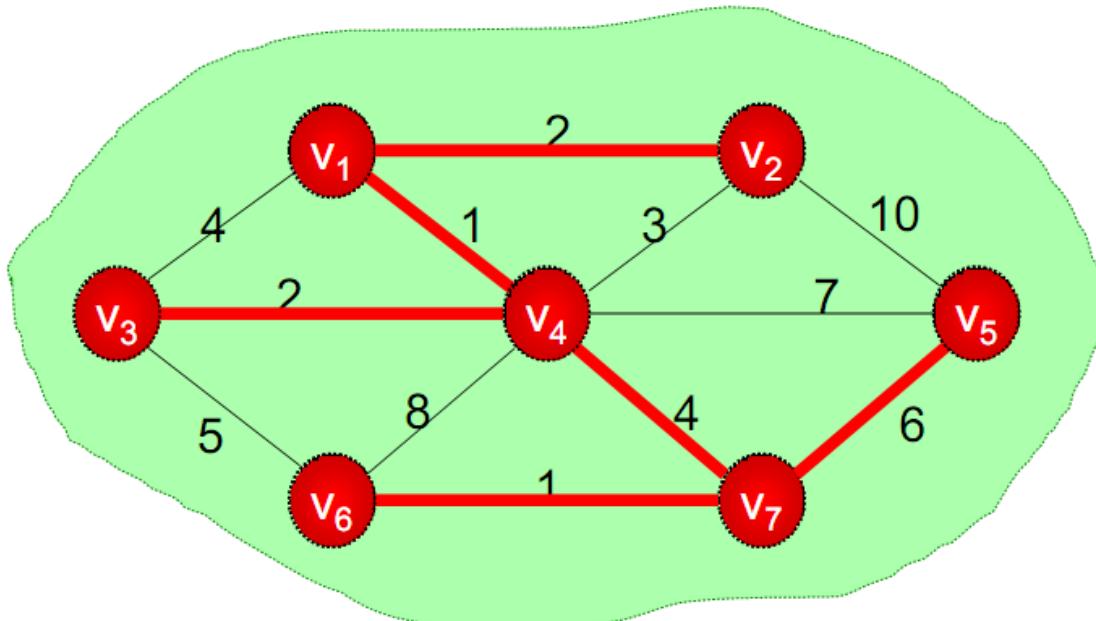


V ₅	6

Content of the priority queue

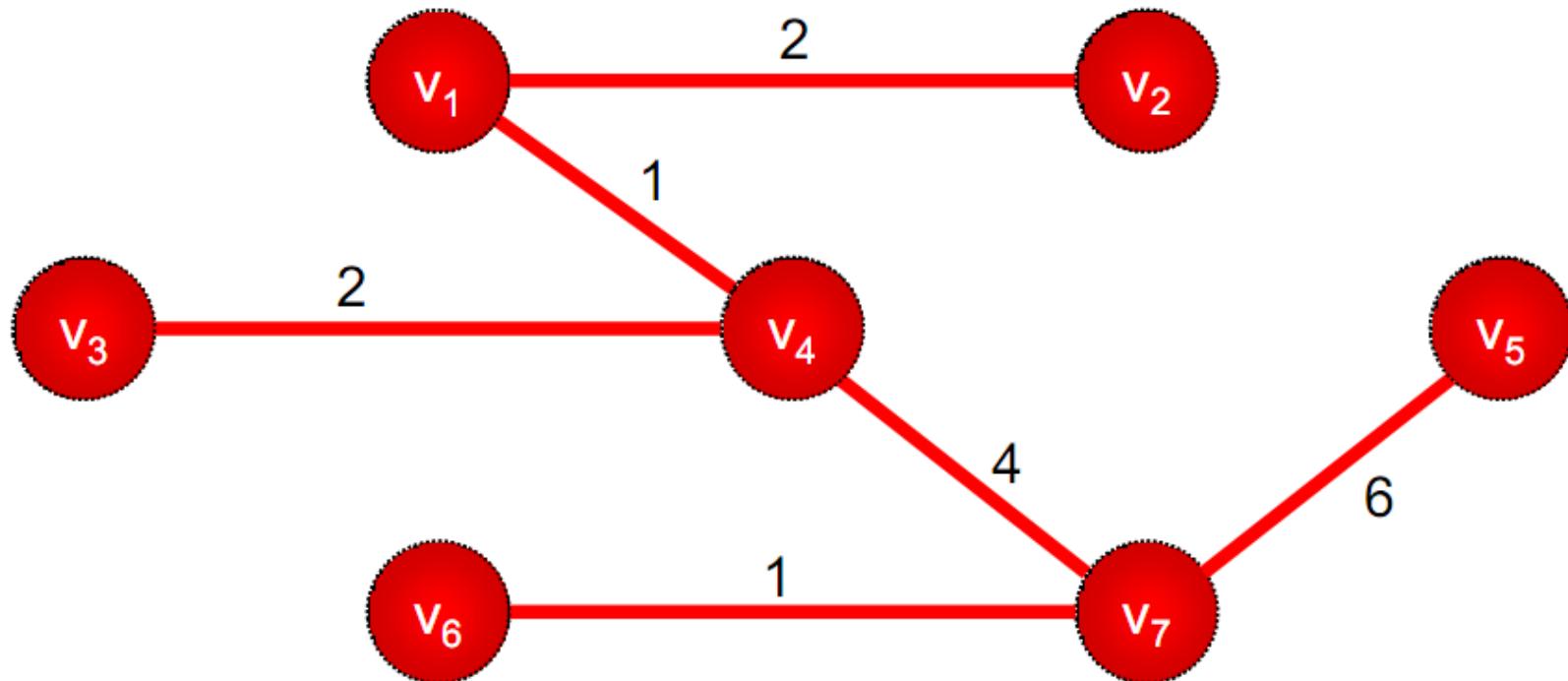
Prim's Algorithm: Example

Grow the tree!

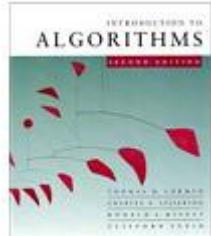


Content of the
priority queue

Prim's Algorithm: Example



Finished! The resulting MST is shown below!



MST algoritmaları

Kruskal algoritması:

- *Kopuk-küme veri yapısı* 'nı kullanır.
- Koşma süresi = $O(E \lg V)$.

Bugüne kadar en iyisi:

- Karger, Klein, and Tarjan [1993].
- Rastgele algoritma.
- $O(V + E)$ beklenen süre.

Kruskal'ın Algoritması

- Kenar tabanlı bir algoritmadır. Kenarlar küçükten büyüğe sıralanır.
- Graf üzerindeki düğümler, aralarında bağlantı olmayan N tane bağımsız küme gibi düşünülür.
- Daha sonra bu kümeler tek tek maliyeti en az olan kenarlarla birleştirilir (çevrim oluşturmayacak şekilde).
- Düğümler arasında bağlantı olan tek bir küme oluşturulmaya çalışılır.
- Küme birleştirme işleminde en az maliyetli olan kenardan başlanılır; daha sonra kalan kenarlar arasından en az maliyetli olanlar seçilir.

Kruskal'ın Algoritması – Kaba Kod

- Yol ağacını oluşturan kenarların tutulduğu A dizisini oluşturur.
- Grafdaki kenarları içeren K dizisini oluşturur. yolUzunluğuna başlangıç değerini ver.

```

while(K≠{Ø} && yolUzunluğu < N ) {
    K içerisinden en düşük maliyetli ki kenarını al ve
    onu K'den sil.

```

```
if(ki A'ya eklendiğinde çevrim oluşturmuyorsa) {
```

```
    ki'yi A'ya ekle.
```

```
    yolUzunluğu++
```

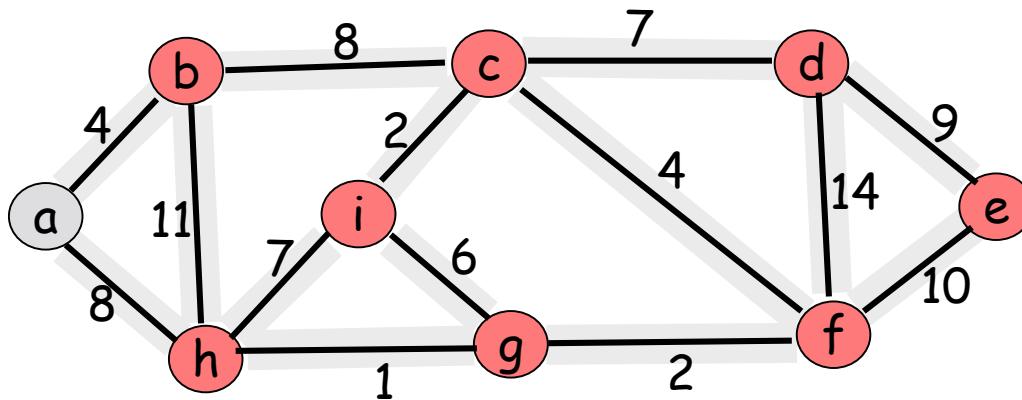
```
}
```

```
}
```

MST-KRUSKAL(G, w)

- 1 $A = \emptyset$
- 2 **for** each vertex $v \in G.V$
- 3 MAKE-SET(v)
- 4 sort the edges of $G.E$ into nondecreasing order by weight w
- 5 **for** each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
- 6 **if** FIND-SET(u) ≠ FIND-SET(v)
- 7 $A = A \cup \{(u, v)\}$
- 8 UNION(u, v)
- 9 **return** A

Kruskal'ın Algoritması: Örnek

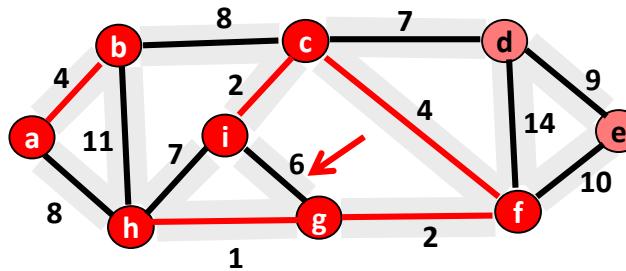
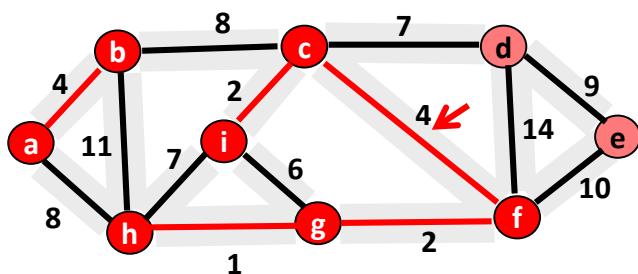
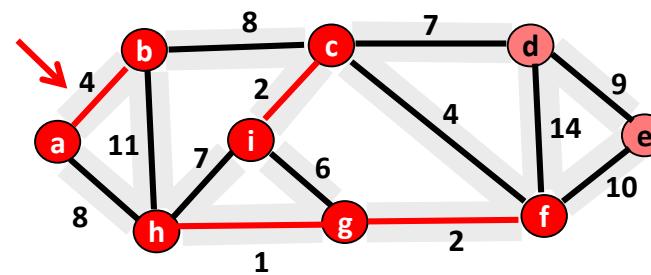
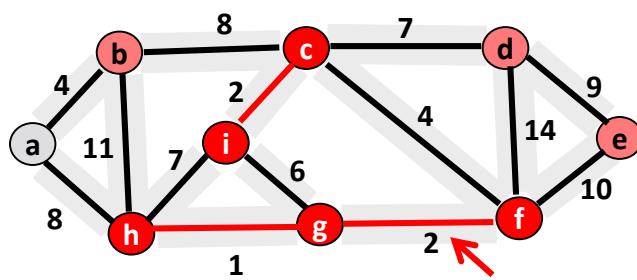
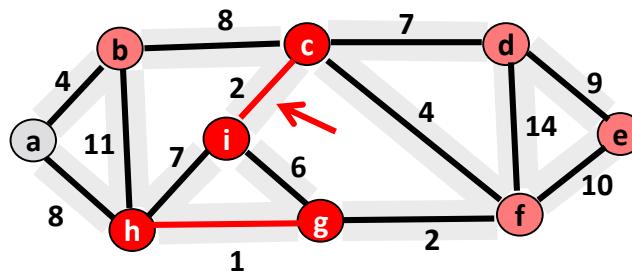
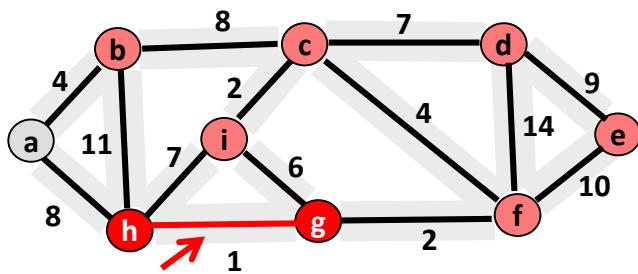


Sıralı kenar listesi

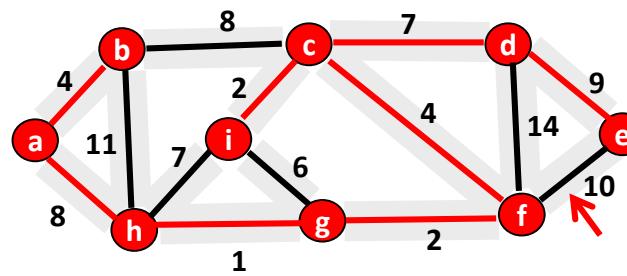
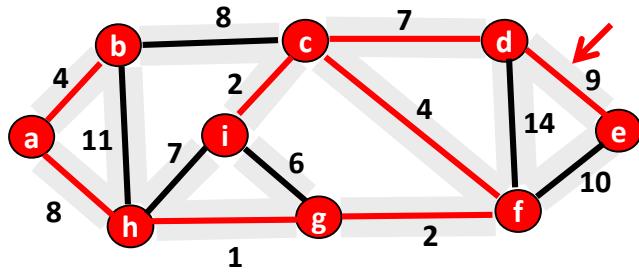
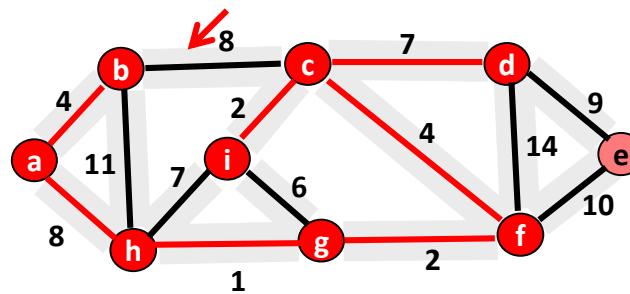
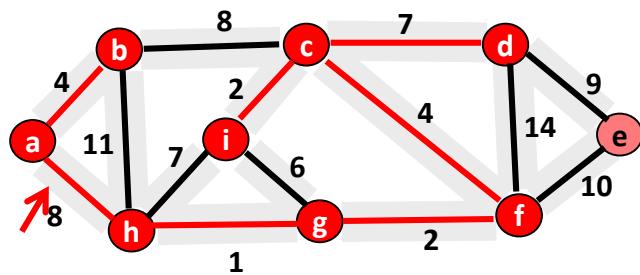
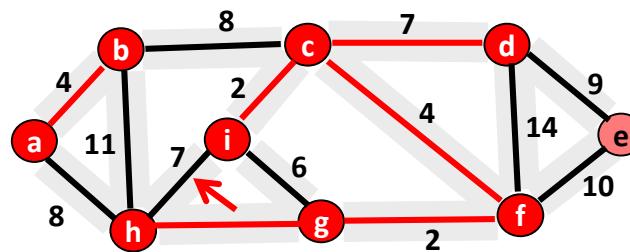
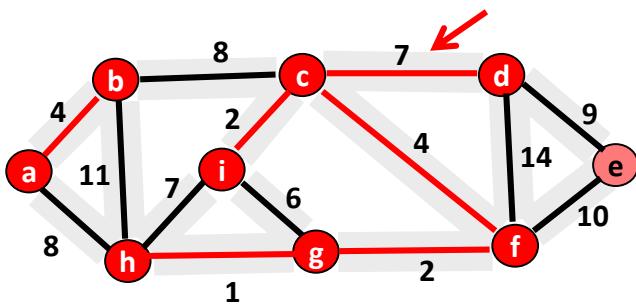
- (h, g)
- (i, c)
- (g, f)
- (a, b)
- (c, f)
- (i, g)
- (c, d)

- (i, h)
- (a, h)
- (b, c)
- (d, e)
- (e, f)
- (b, h)
- (d, f)

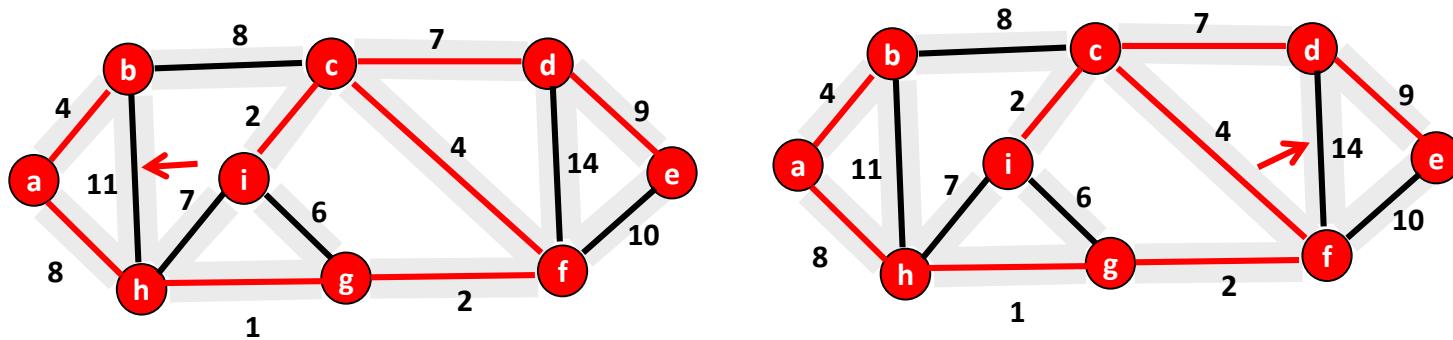
Kruskal'ın Algoritması: Örnek



Kruskal'ın Algoritması: Örnek



Kruskal'ın Algoritması: Örnek

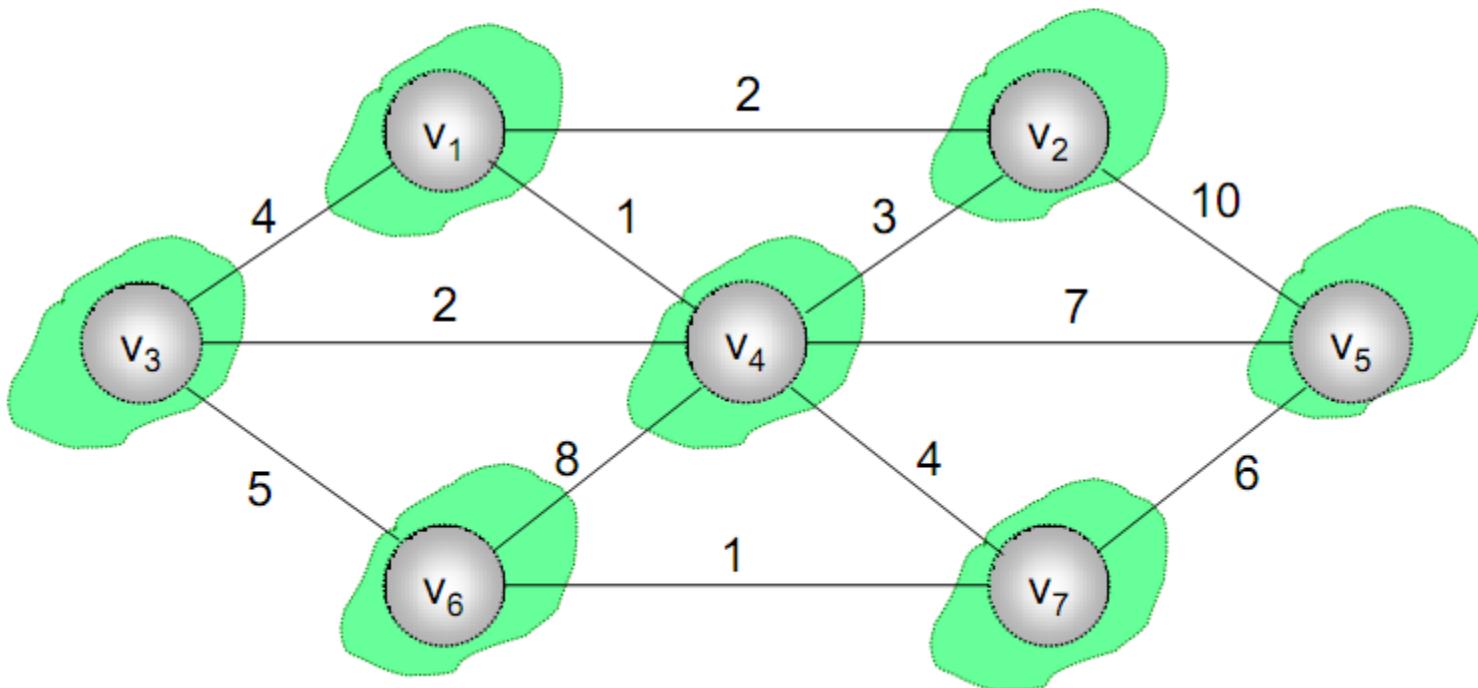


Sıralı kenar listesi

(h, g) (i, c) (g, f) (a, b) (c, f) (i, g) (c, d)

(i, h) (a, h) (b, c) (d, e) (e, f) (b, h) (d, f)

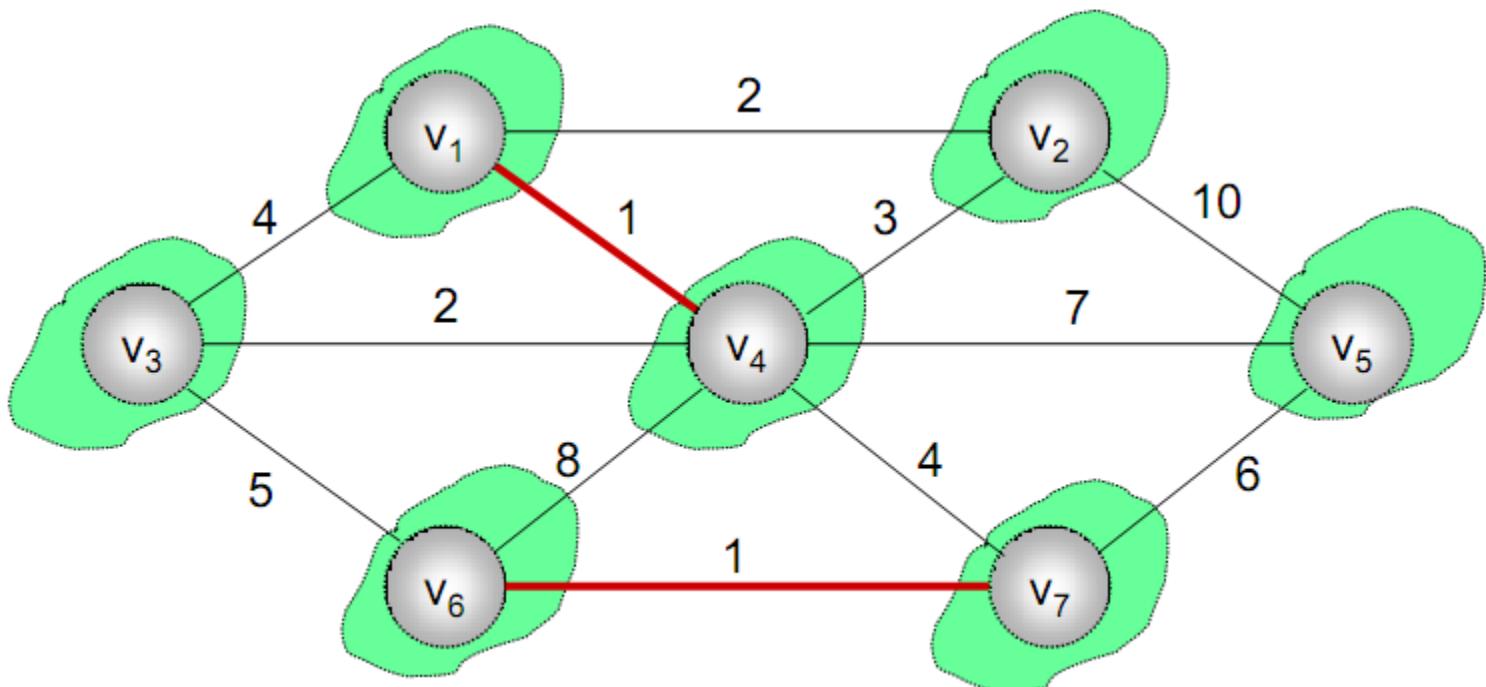
Initial Forest



Initial Forest

Step 1

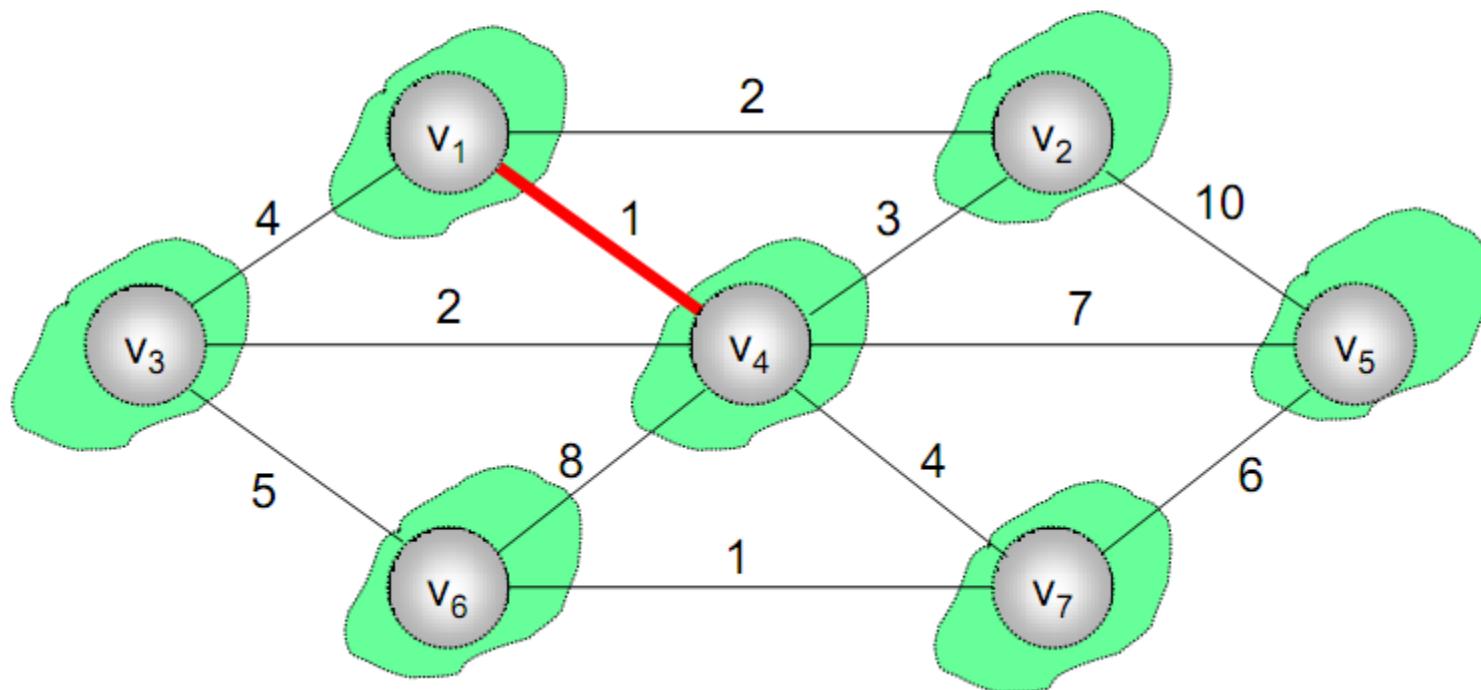
*Candidate edges are shown
(edges that have low cost and
edges that connect two trees)*



Initial Forest

Step 2

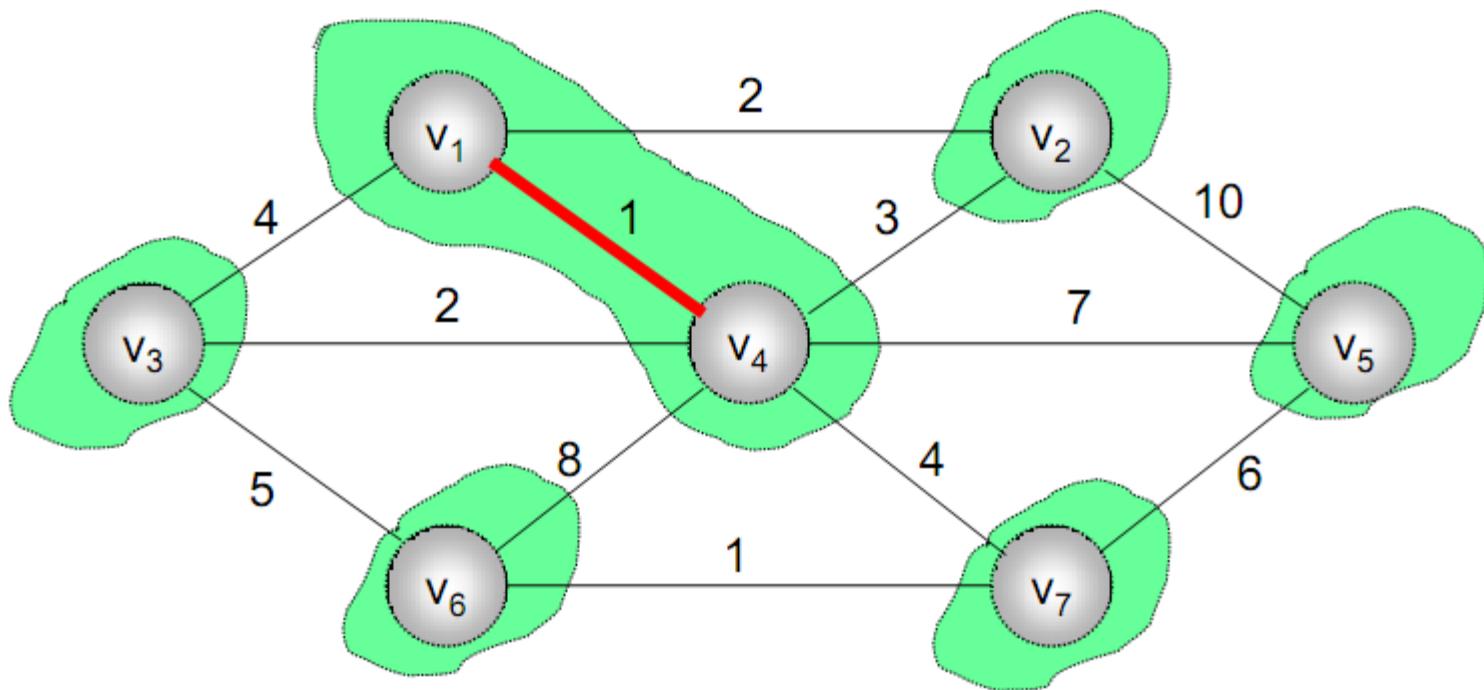
*Accept one of the candidate edges: (v_1, v_4)
(we can do random accept here).*



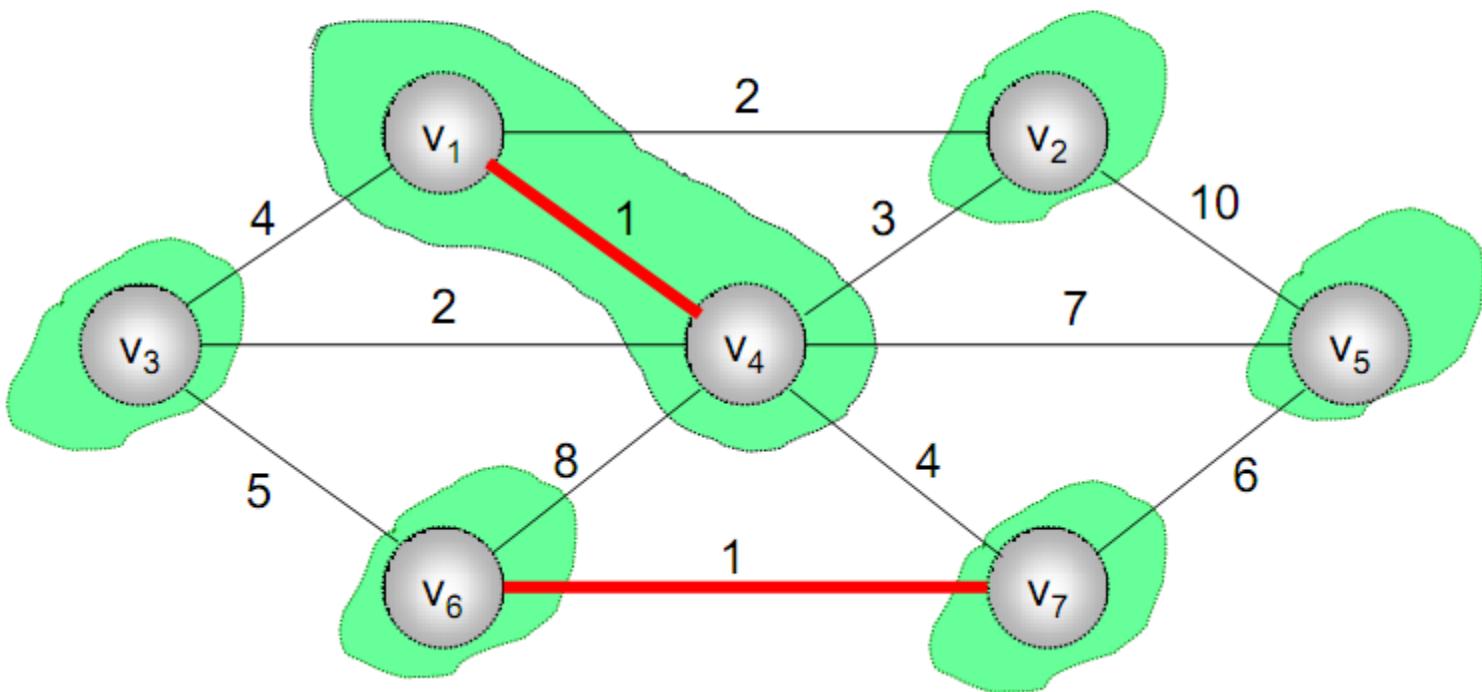
Initial Forest

Step 3

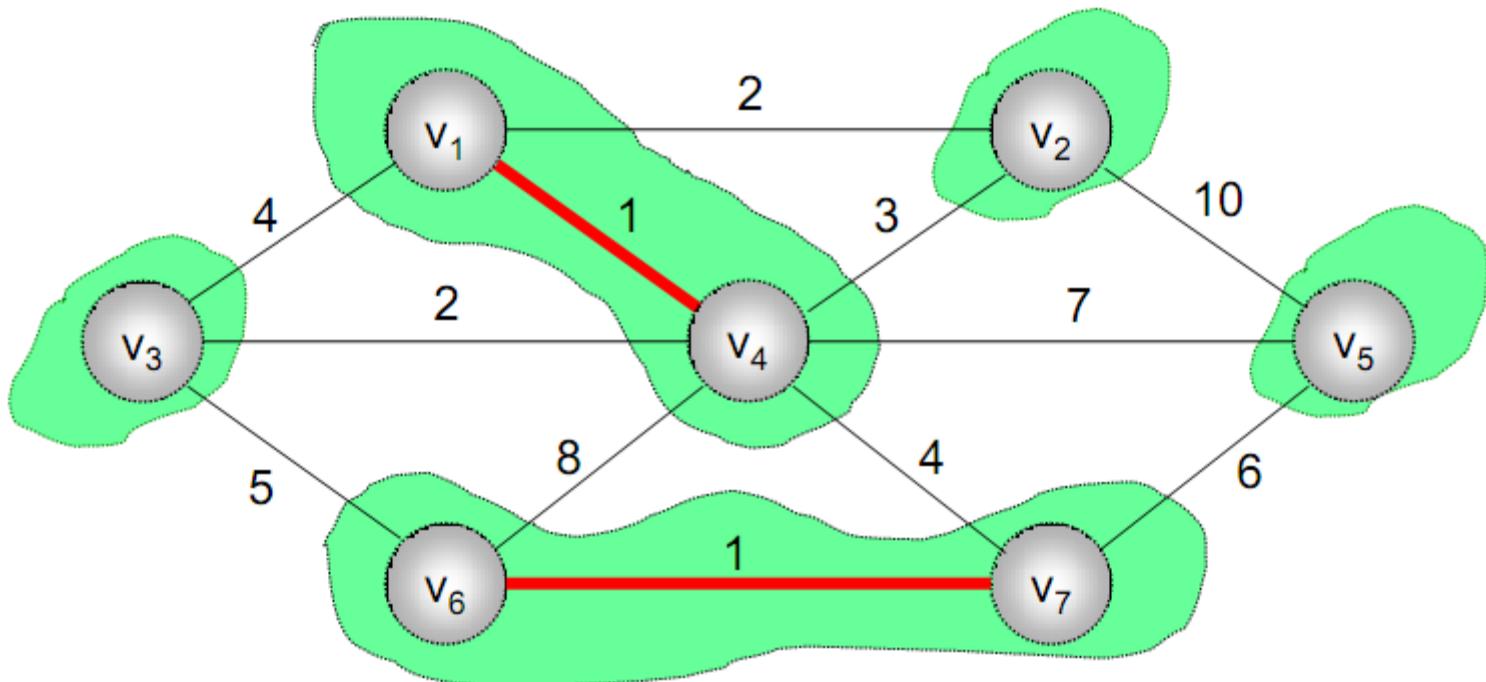
*Merge the two trees connected by that edge.
Obtain a new tree in this way.*



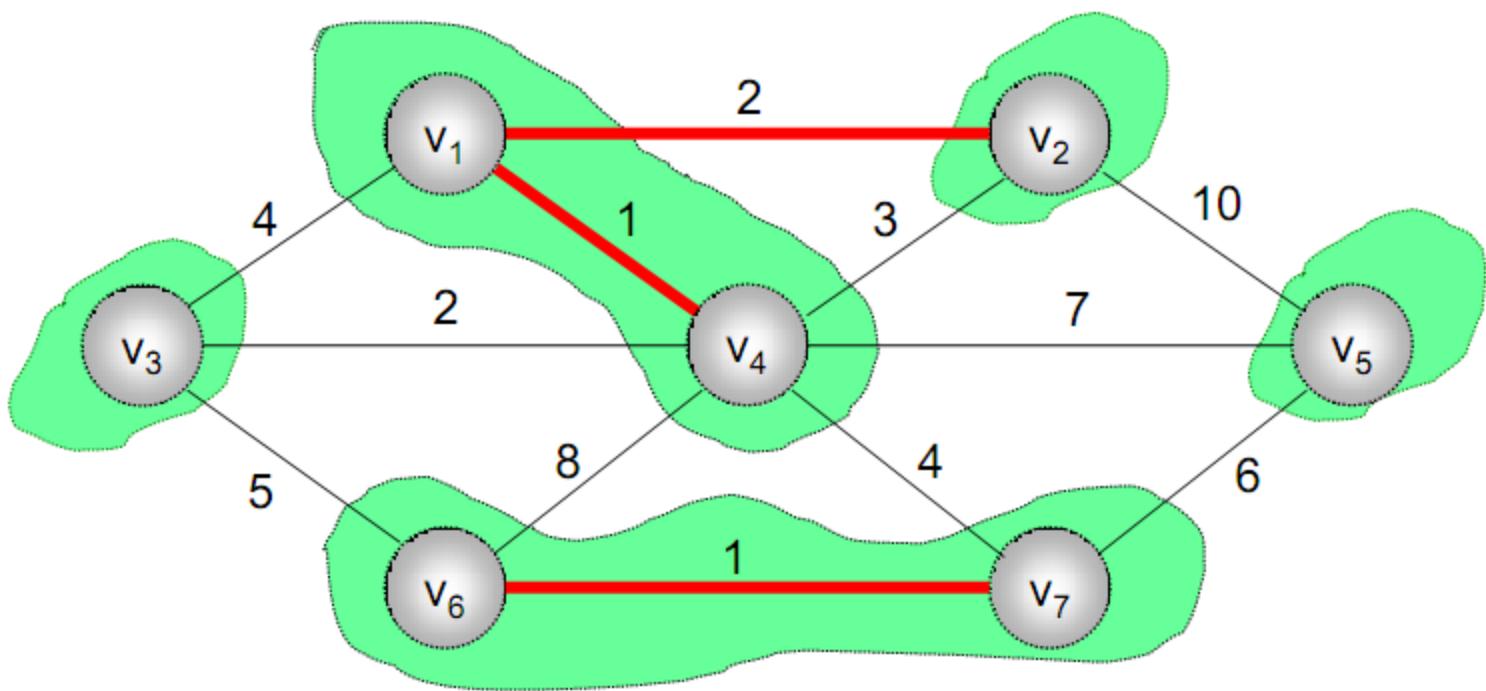
*Repeat previous steps!
Edge (v_6-v_7) is accepted.*



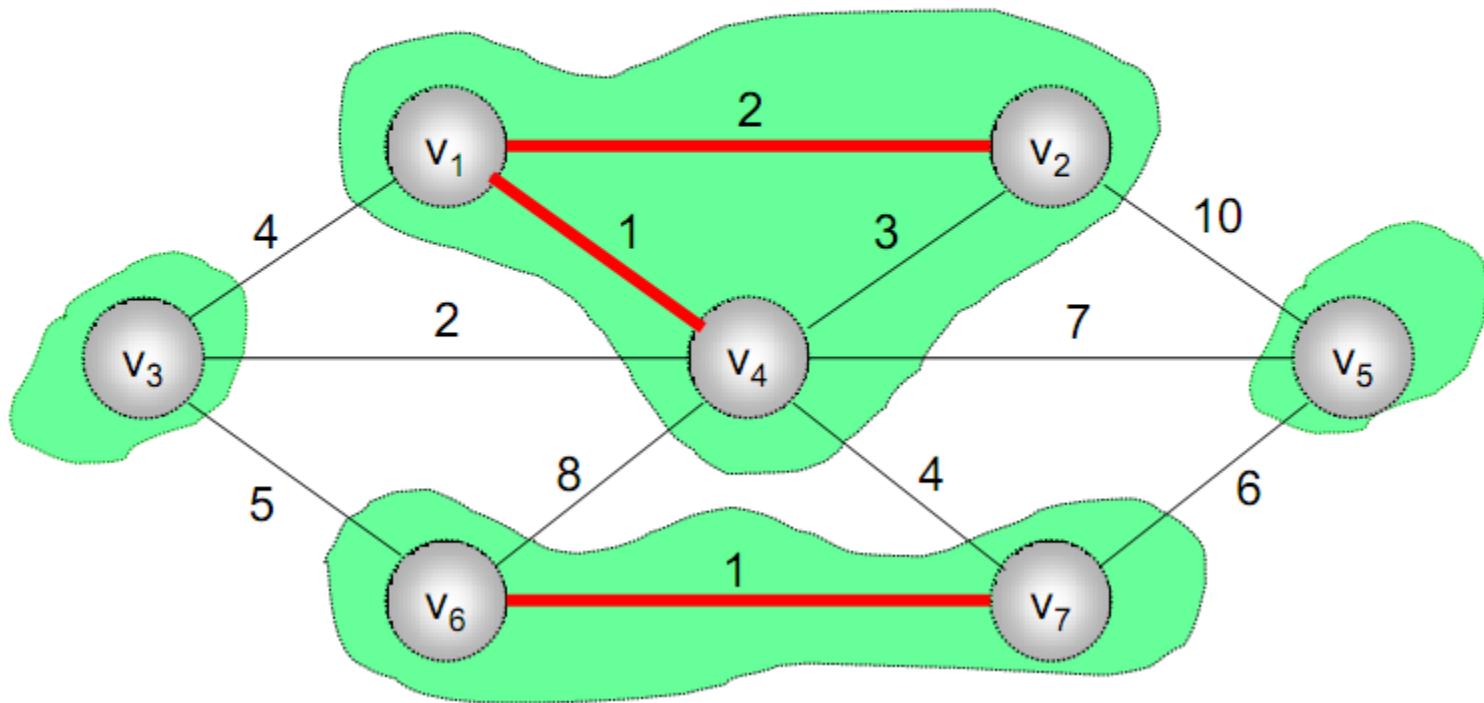
Merge the two trees connected by that edge!



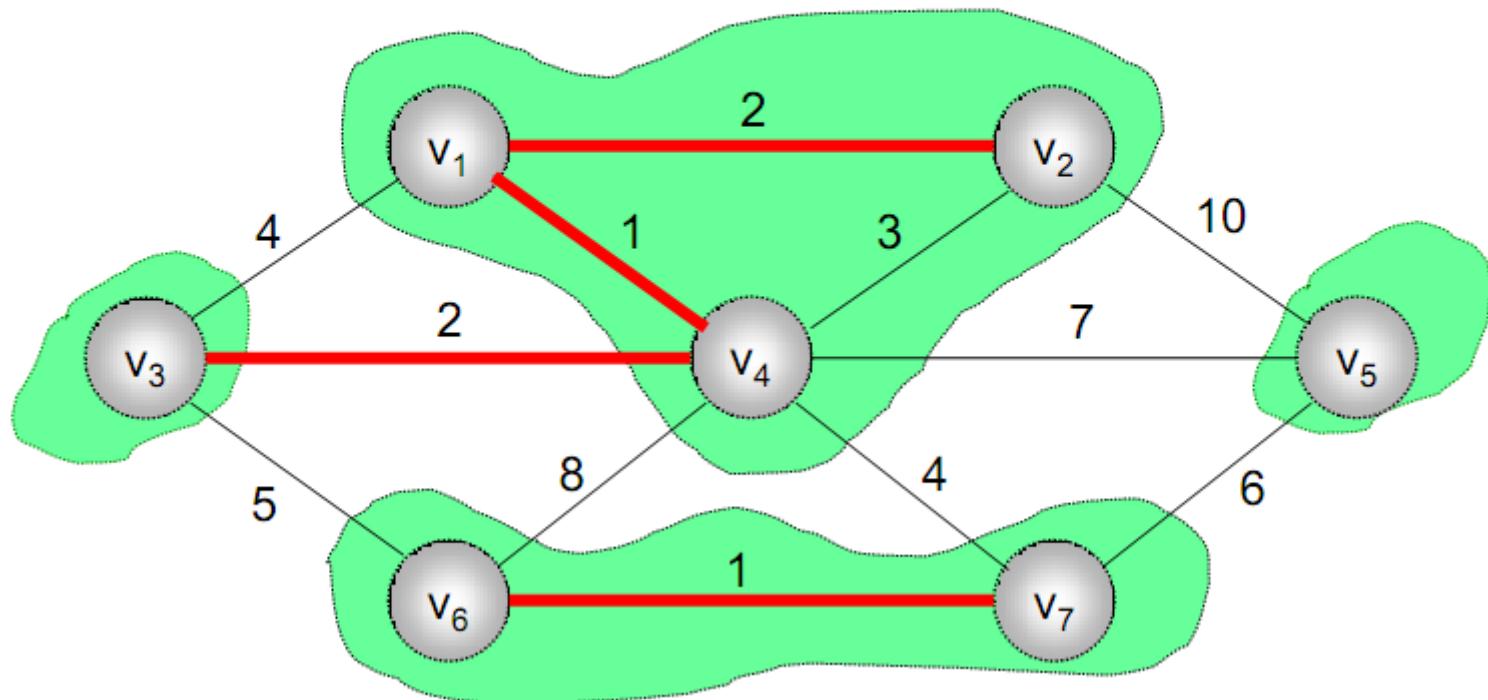
Accept edge (v_1, v_2)



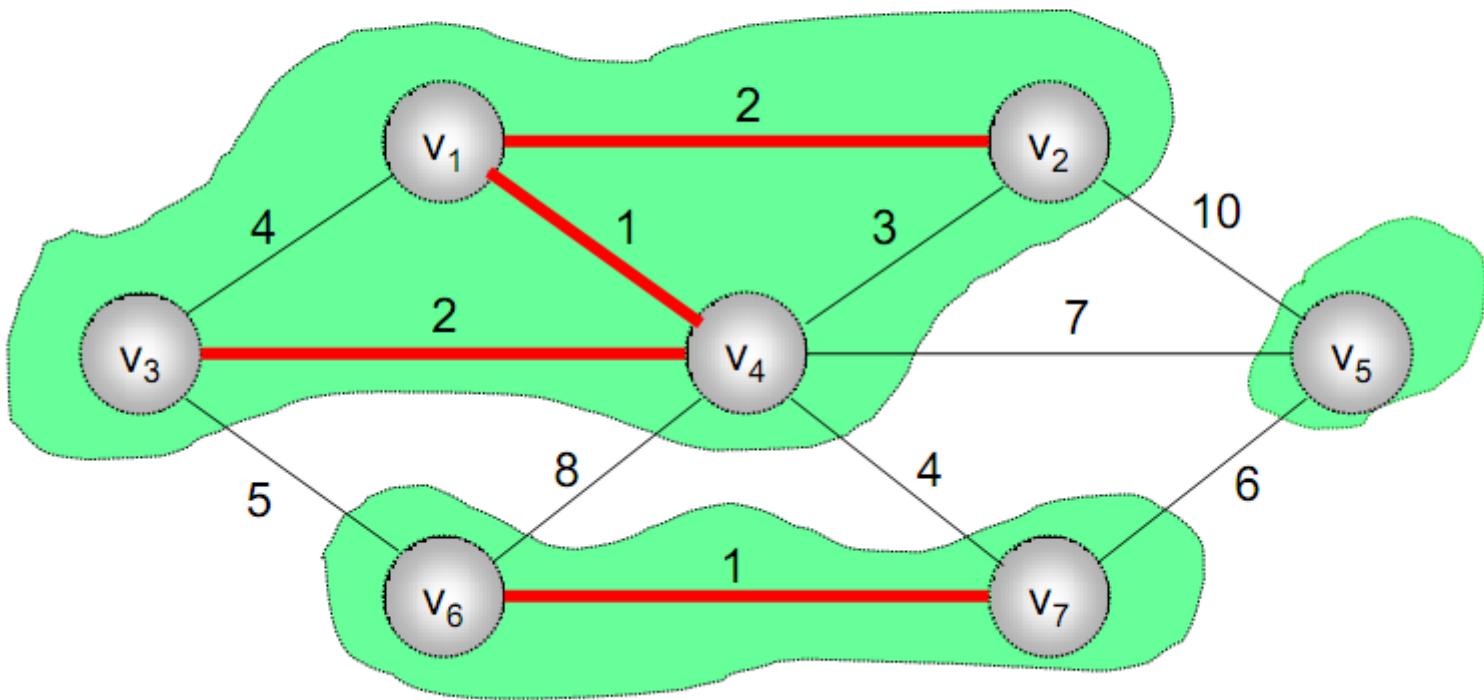
Merge the two trees connected by that edge!



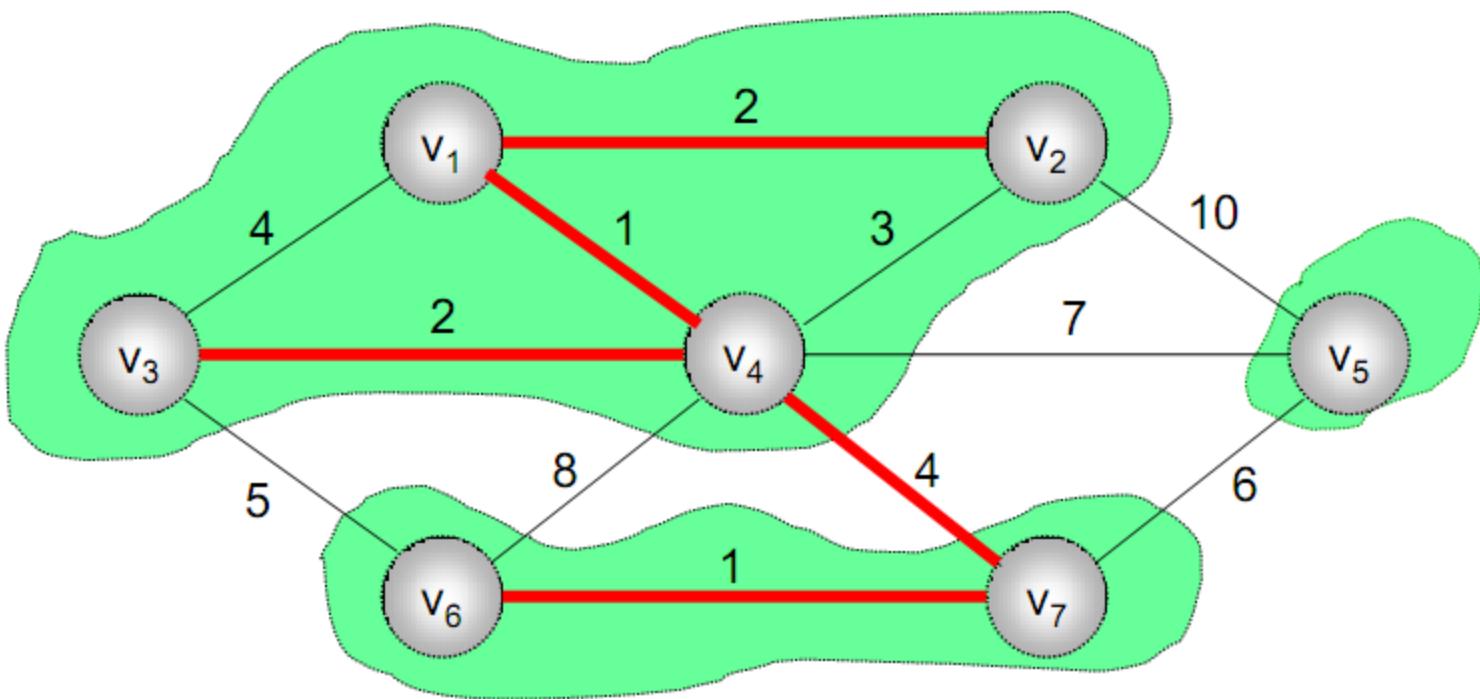
Accept edge (v_3, v_4)



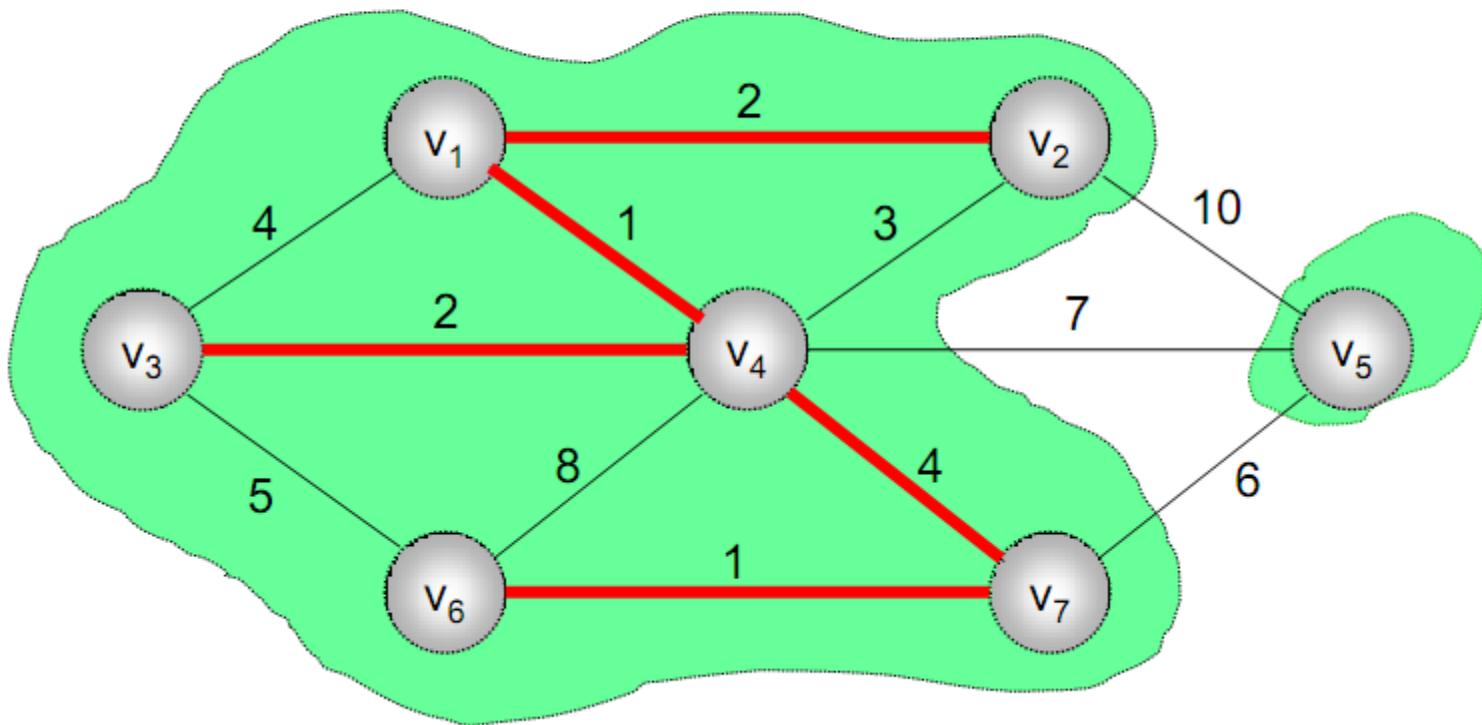
Merge the two trees connected by that edge!



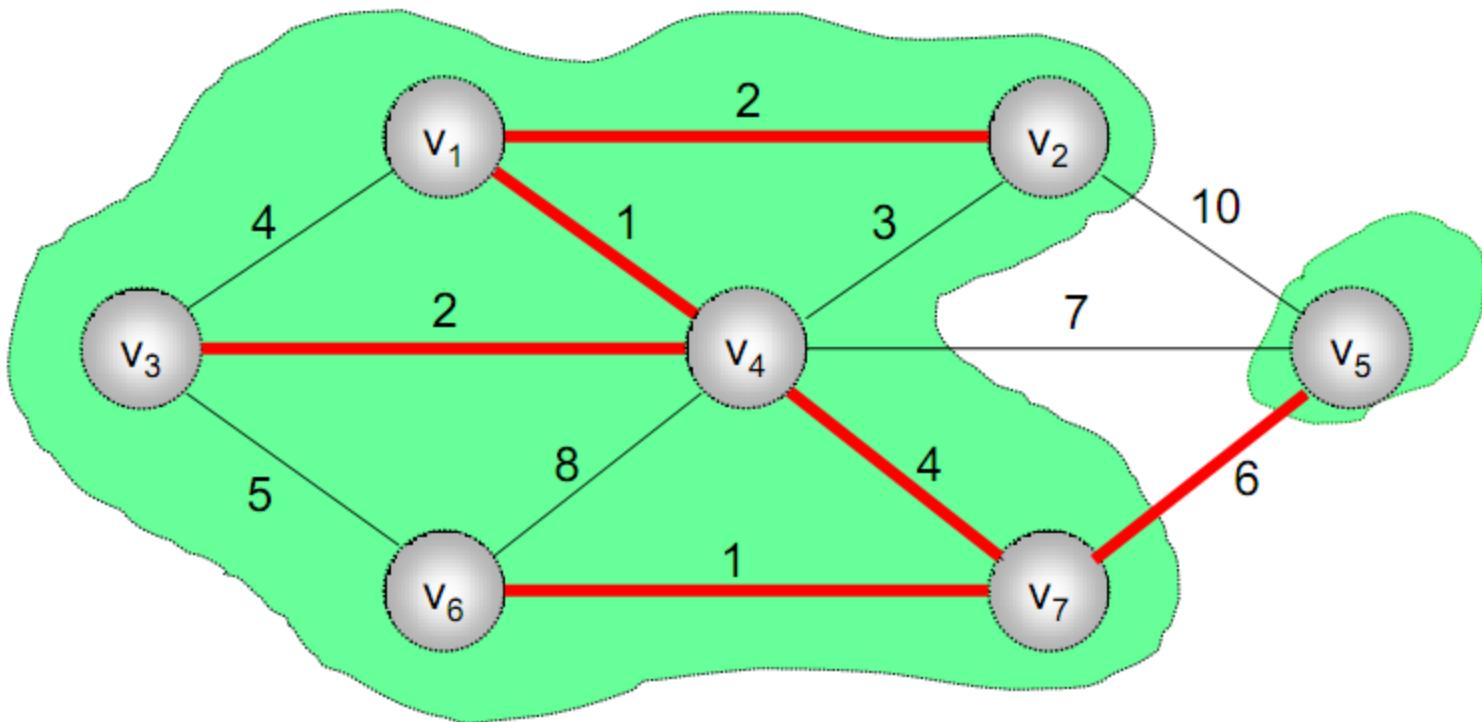
Accept edge (v_4, v_7)



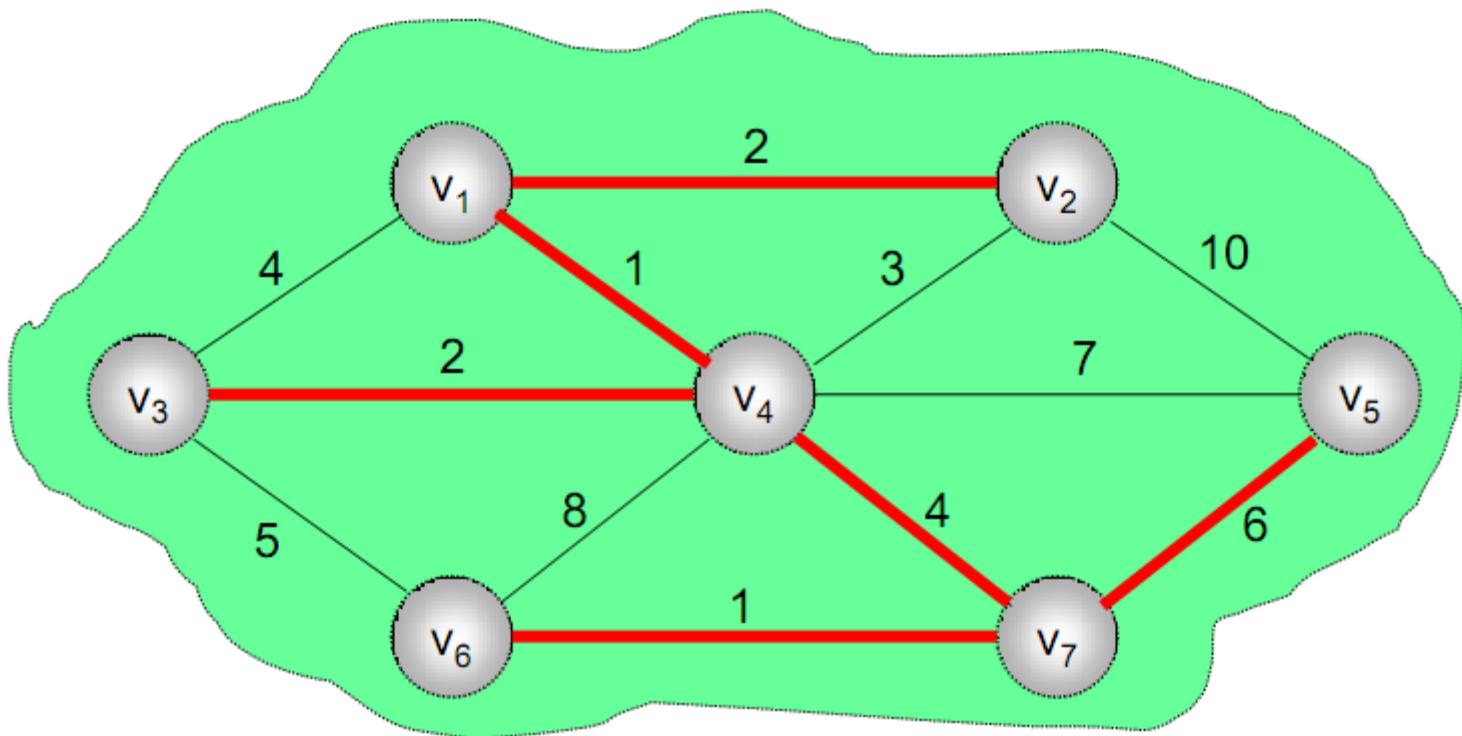
Merge the two trees connected by that edge!



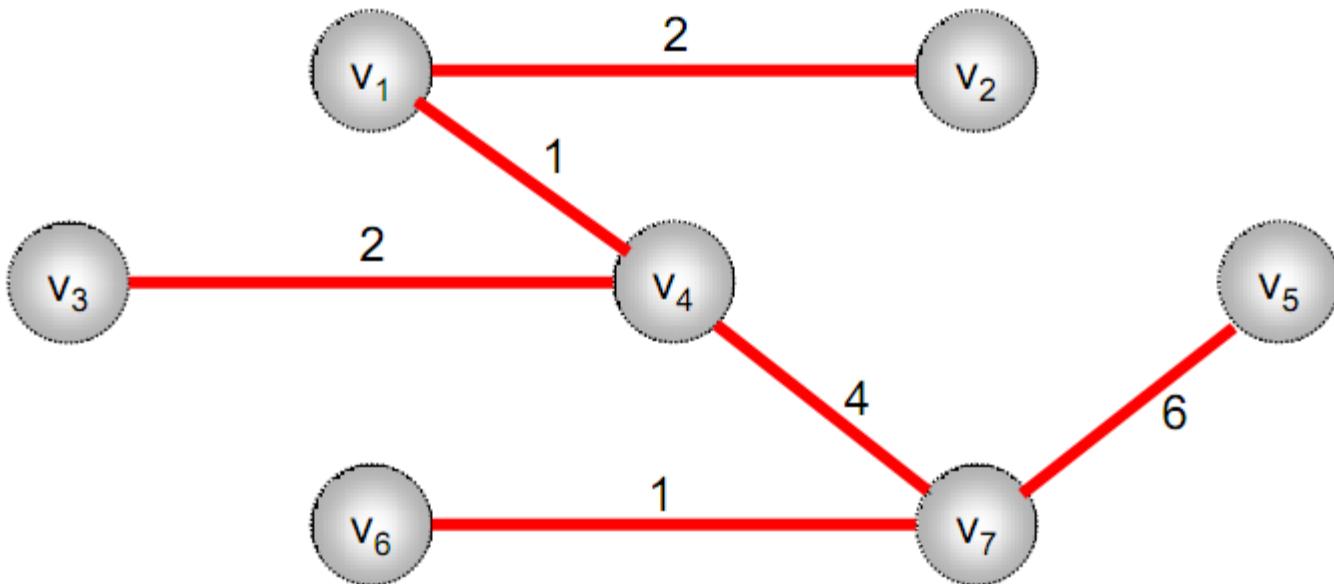
Accept edge (v_7, v_5)



Merge the two trees connected by that edge!



Finished!
The resulting MST is shown below!



Sollin(Boruvka) 'in Algoritması

- Minimum geçişli ağaçların bulunmasına yönelik bu algoritma 1926 yılında Boruvka tarafından önerilmiştir.
- Sollin algoritması aynı anda birden çok ağaçla başlayan ve sonraki adımlarda ağaçların birleşerek tek bir ağaca dönüştüğü bir algoritmadır.
- Bu algoritmada, bir adımda birden fazla kenar seçilir ve yol ağacı ara işlemlerde alt ağaç olarak bulunan ağaçlara eklenmek suretiyle bulunur.
- Yani yol ağacı belirlenené kadar ara işlemlerde birden fazla ağaç bulunur.
- Kruskal ve Prim'in algoritmalarına göre sonuca daha az adımda ulaşan Sollin algoritmasının ara işlemleri bu algoritmala göre daha fazla olabilir.
- Çalışma zamanı $O(E \log V)$

Sollin 'in Algoritması- Kaba Kod

$SK \leftarrow \{ \emptyset \}$; boş seçilen kenarlar (SK) dizisi oluşturur.

Başlangıç anında her düğüm için En Küçük Maliyetli (EKM) kenarı seç ve sonuçta tüm düğümleri içeren altağaçları (ormanı) oluşturur.

while (ormandaki ağaç sayısı > 1 VEYA kenar sayısı < N)

{

Her ağaç, genişletmek için, o ağaca herhangi bir yerden bağlı en az maliyetli kenarı al. O kenarı ilgili altağaca ekle.

Aynı kenar birden çok seçilmişse ilki dışındakileri önemseme.

}

Sollin'in Algoritması

BORÜVKA(V, E):

$$F = (V, \emptyset)$$

while F has more than one component
 choose leaders using DFS
 $\text{FINDSAFEEDGES}(V, E)$
 for each leader \bar{v}
 add $\text{safe}(\bar{v})$ to F

loop iterates $O(\log V)$ times

FINDSAFEEDGES(V, E):

for each leader \bar{v}

$$\text{safe}(\bar{v}) \leftarrow \infty$$

for each edge $(u, v) \in E$

$$\bar{u} \leftarrow \text{leader}(u)$$

$$\bar{v} \leftarrow \text{leader}(v)$$

$$\text{if } \bar{u} \neq \bar{v}$$

$$\text{if } w(u, v) < w(\text{safe}(\bar{u}))$$

$$\text{safe}(\bar{u}) \leftarrow (u, v)$$

$$\text{if } w(u, v) < w(\text{safe}(\bar{v}))$$

$$\text{safe}(\bar{v}) \leftarrow (u, v)$$

Each call to FINDSAFEEDGES takes $O(E)$ time

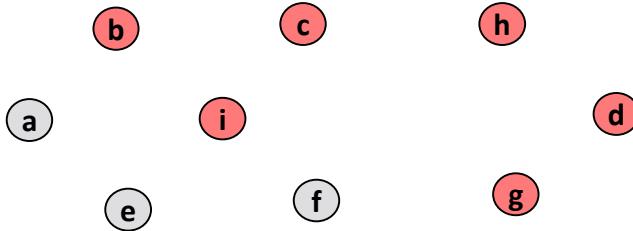
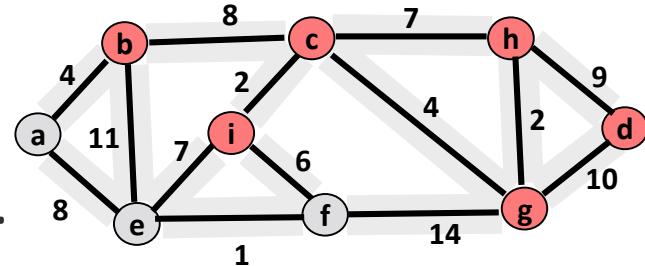
running time of Borùvka's algorithm is $O(E \log V)$.

Sollin 'in Algoritması- Kaba Kod

- Adım 1: Algoritmada her düğüm en küçük maliyet değerine sahip komşu düğümü belirler.
- Adım2- Belirlediği bu düğümle kendisi arasında bir kenar oluşturur. Bu belirleme sürecinde aynı kenarın birden fazla seçildiği durumda, ilkinden sonraki durumlar grafın yönsüz graf olması nedeniyle önemsiz kabul edilir.
- Adım 3- Meydana gelen alt ağaçlar daha sonra kenar maliyeti değeri dikkate alınarak yeniden birleştirilir. Uygulama, sonunda tek bir ağaç kalana kadar devam eder.

Sollin'in Algoritması: Örnek

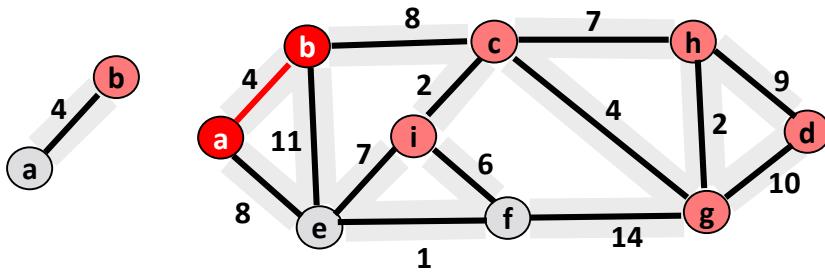
- Öncelikler tüm ayrıtları siliniz.



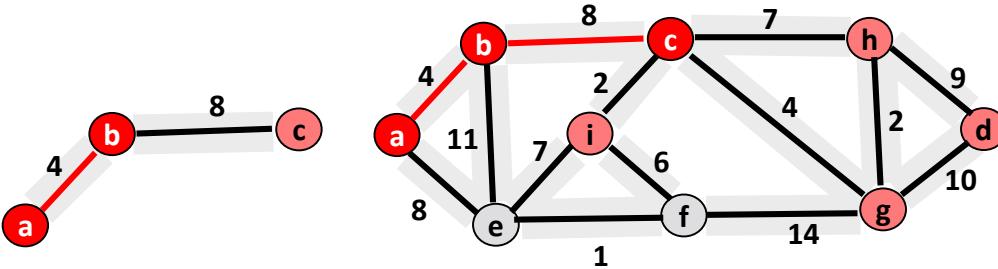
- Başlangıç durumu
- Düğümler sıralanır $\{a,b,c,d,e,f,g,h,i\}$ ve sıradaki her düğüm için minimum maliyetli yollar seçilir.

Sollin'in Algoritması: Örnek

- a düğümü için (a-b) seçilir (en düşük maliyetli kenar) bu kenar çizilerek maliyeti M dizisine eklenir $M=\{4\}$

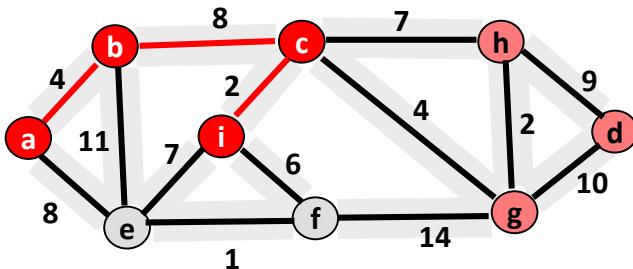


- b düğümü için (b-c) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir $M=\{4+8\}=12$

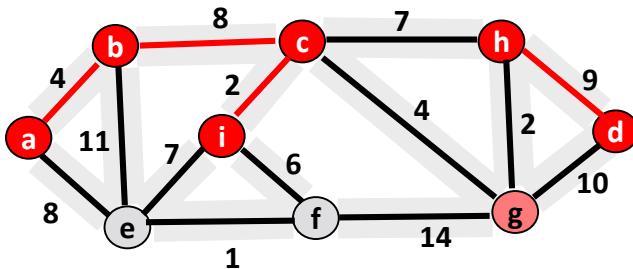


Sollin'in Algoritması: Örnek

- c düğümü için ($c-i$) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir, $M=\{12+2\}=14$

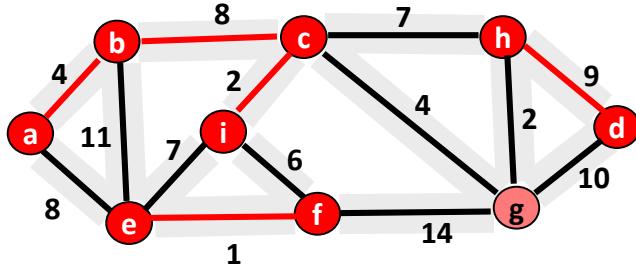


- d düğümü için ($d-h$) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir, $M=\{14+9\}=23$



Sollin'in Algoritması: Örnek

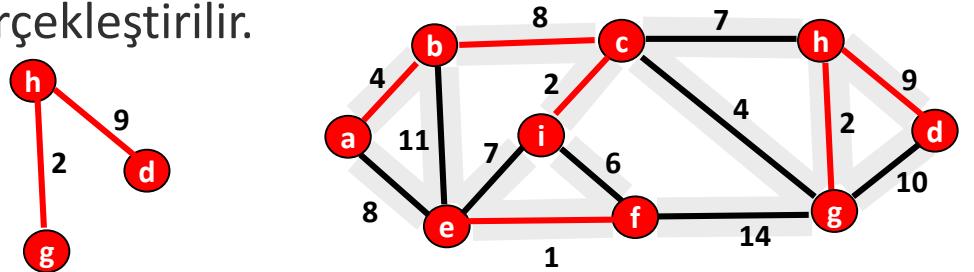
- e düğümü için (e-f) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir, $M=\{23+1\}=24$



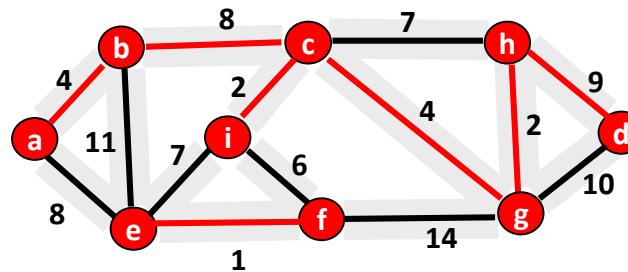
- f düğümü için (f-e) seçilir bu kenar daha önce çizildiği için maliyeti toplam minimum maliyet M dizisine eklenmez.

Sollin'in Algoritması: Örnek

- g düğümü için (g-h) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir, $M=\{24+2\}=26$. Bu aşamada d-g-h altağacı bulunur. Bu adımdan sonra altağaçlar birleştirilmelidir. Kalan düğümler içinden minimum maliyetlisi seçilerek birleştirme gerçekleştirilecektir.

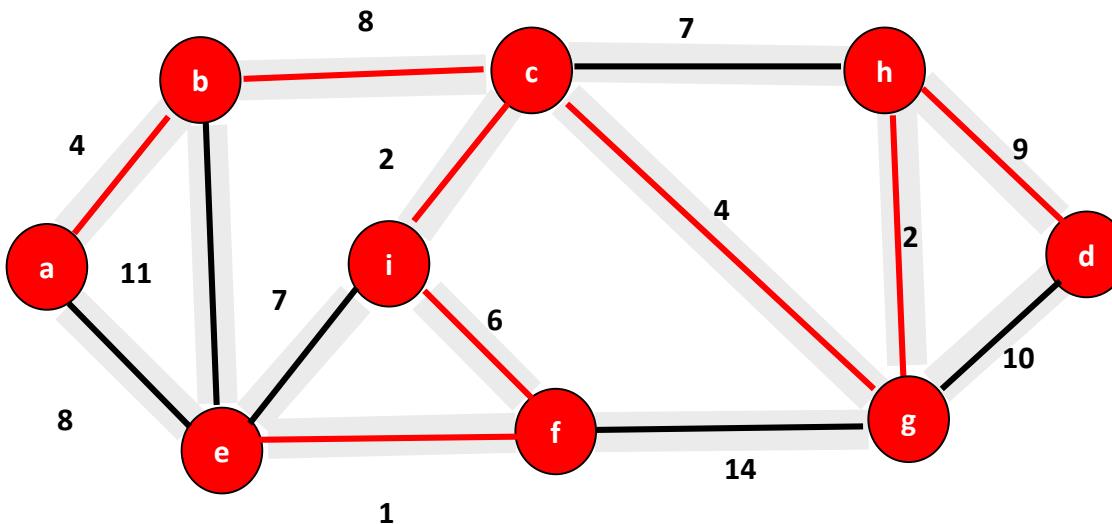


- g-c düğümleri en düşük olduğu için seçilip birleştirildi.
 $M=\{26+4\}=30$



Sollin'in Algoritması: Örnek

- e-f alt ağacı aynı mantıkla f-i düğümü ile birleştirilerek ağacın son hali çizilir. $M=\{30+6\}=36$.



13.Hafta

En kısa yollar- I

En kısa yolların özellikleri

- Dijkstra algoritması
- Doğruluk
- Çözümleme
- Enine arama