CENG 481 GRAF TEORİ VE UYGULAMALARI
Hafta 7

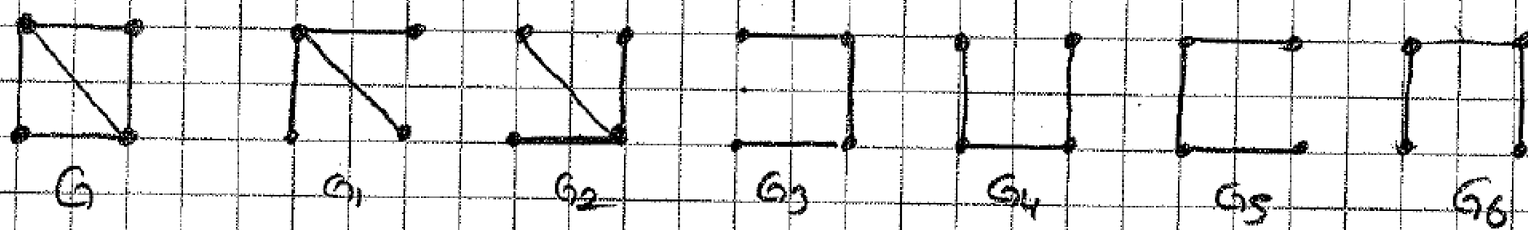
Prof. Dr. Tufan TURACI
tturaci@pau.edu.tr

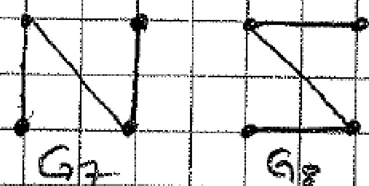
Hafta 7
Konular

1- Dallanmış Ağaç Algoritmaları

Dallanmış Alt ağaçlar

Bir G grafının tüm tepelerini içeren birleştirilmiş bir alt grafa dallanmış alt graf denir. Eğer dallanmış alt graf çevre içermiyorsa dallanmış (alt) ağaç denir.





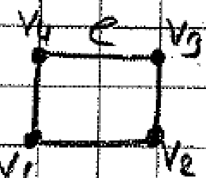
NOT ⇒ Tüm tepeleri içerecek.

Teorem: (Cayley)

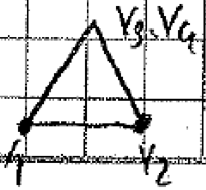
$Z(G)$, dallanmış ağaçların sayısını göstermek üzere,

$$Z(G) = Z(G-e) + Z(G.e)$$ büzülme

büzülme işlemi:

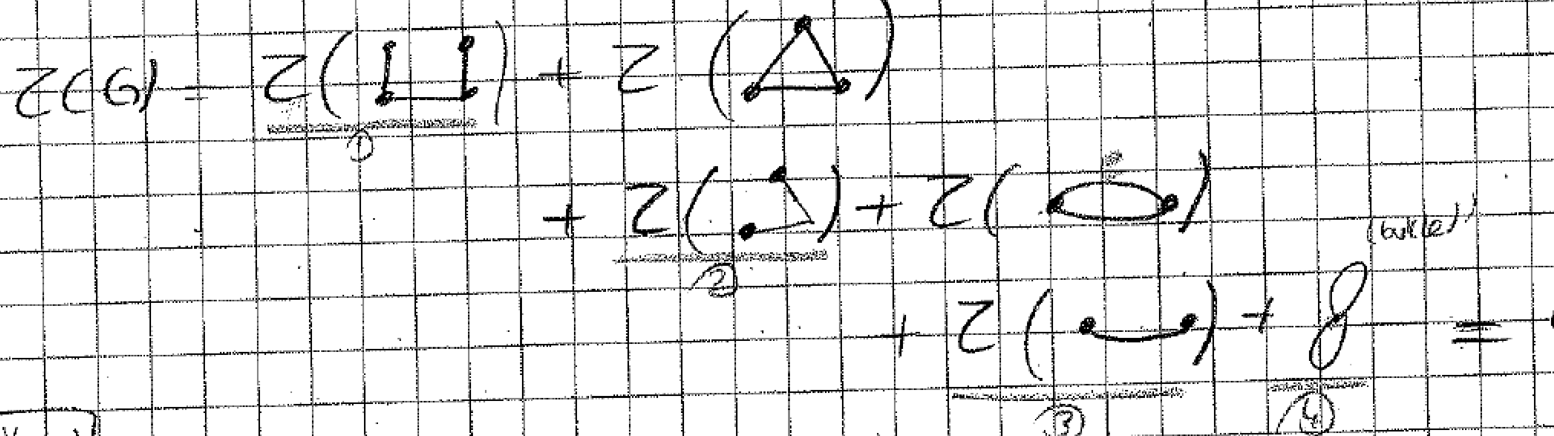


$G.e =$



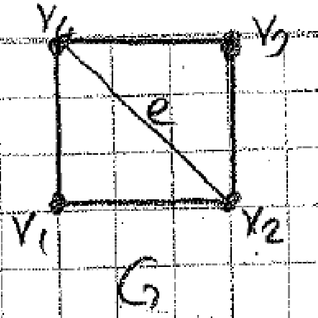
Ör. 4

 = 4

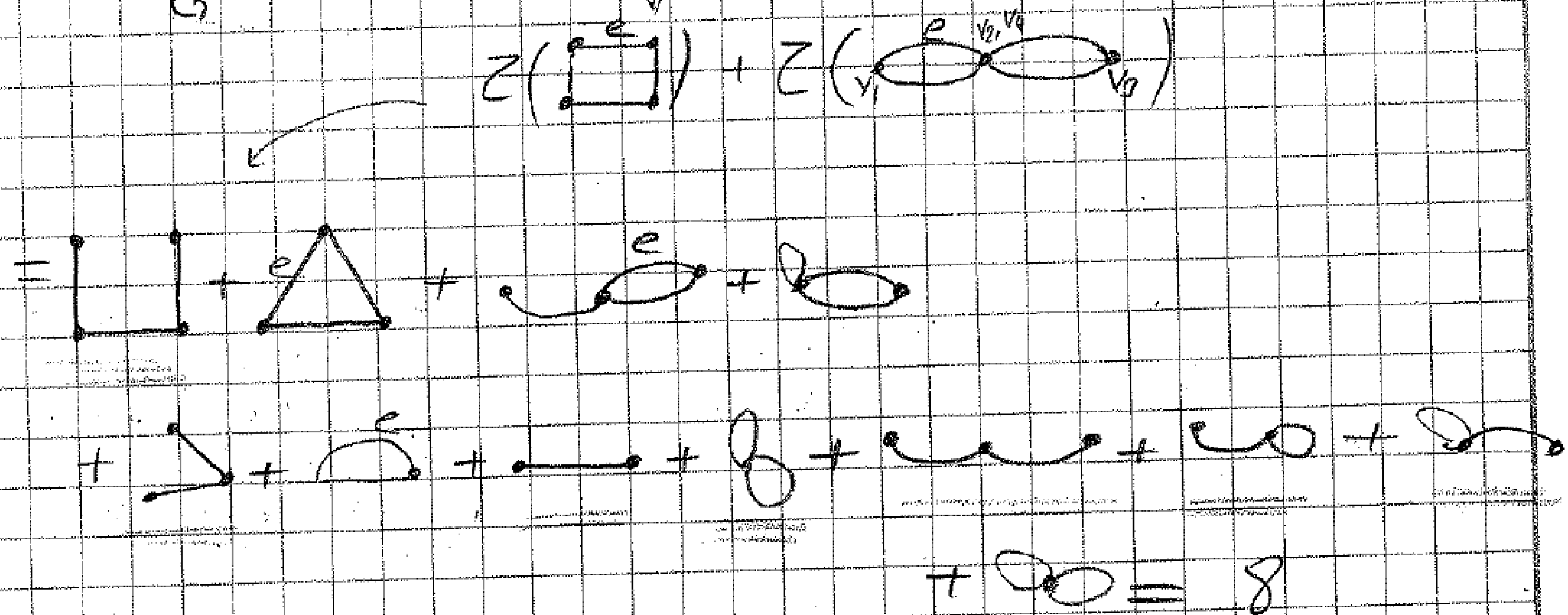


Yon 1

 $\Rightarrow$ 4 tane



$$Z(G) = Z(G-e) + Z(G.e)$$



DALLANMIŞ AĞAÇ ALGORİTMALARI

1. Algoritma:

Verilen bir G grafının dallanmış ağaçlarının herhangi birini bulmak için çeşitli algoritmalar kullanılır. Aşağıdaki algoritma ağaç oluşturma algoritması olarak bilinir.

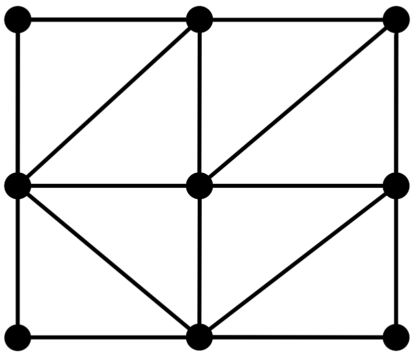
Adım1: Grafın herhangi bir e ayrıtını seç ve bu ayrıtın uç noktalarını v_1,v_2 ile isimlendir.(i=1, j=2 olsun.)

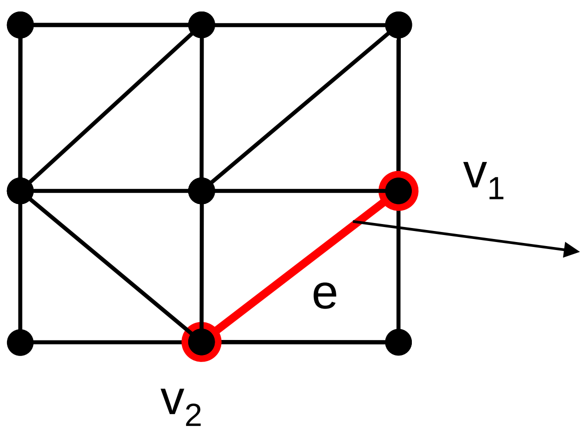
Adım2: v_i tepesinin , verilen graftaki tüm komşularının oluşturulan ağaçta olup olmadığını kontrol et.

Adım3: vi tepesinin grafta olup oluşturulan ağaçta olmayan bir komşu tepesi varsa bunu vj ile isimlendir ve (v_i v_j) ayrıtını ağaca ekle. Eğer j, grafın tepe sayısına eşitse dur. Şu an dallanmış bir ağaca sahipsin. Aksi halde j yi 1 arttır ve adım2 ye dön.

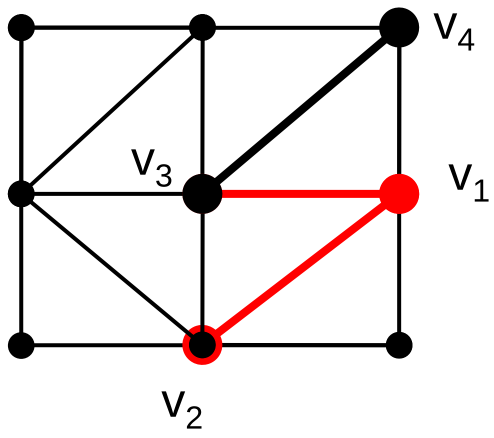
Bunu programlamak için matris (veri yapısını) kullanmak gereklidir.

Örnek:

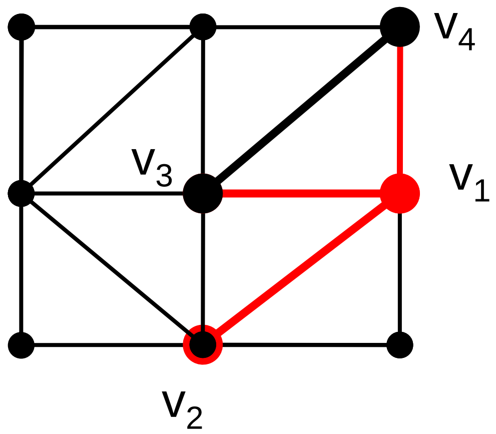




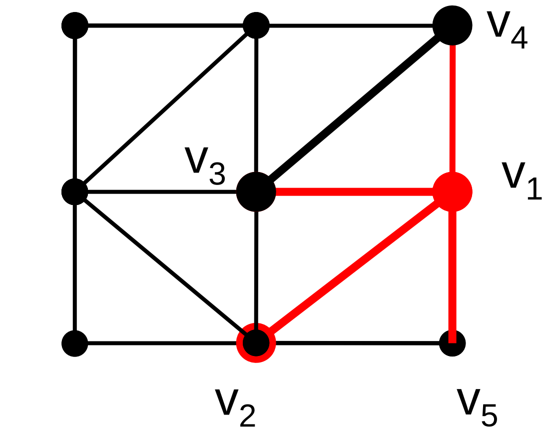
ayrıtını
seçelim
i=1 ve j=2



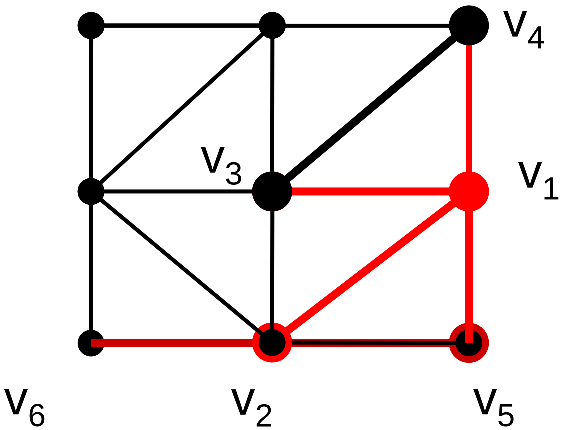
j=3



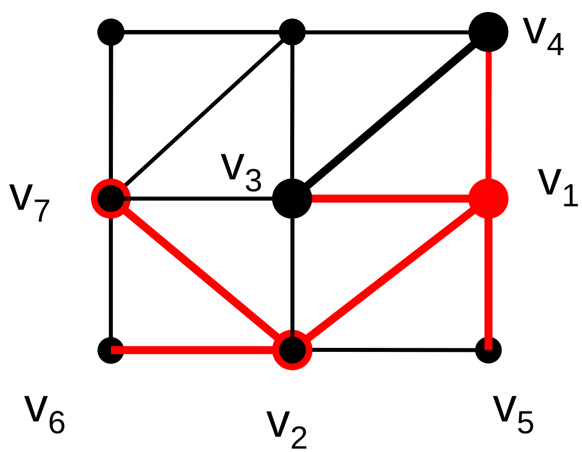
j=4



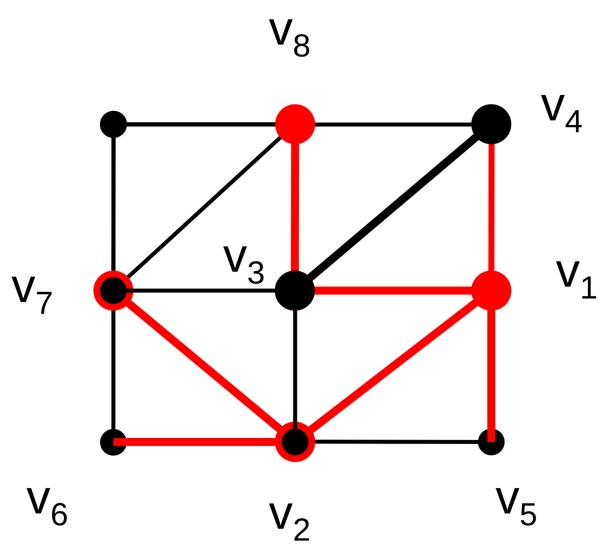
j=5



j=6 , i=2



$j=7,$



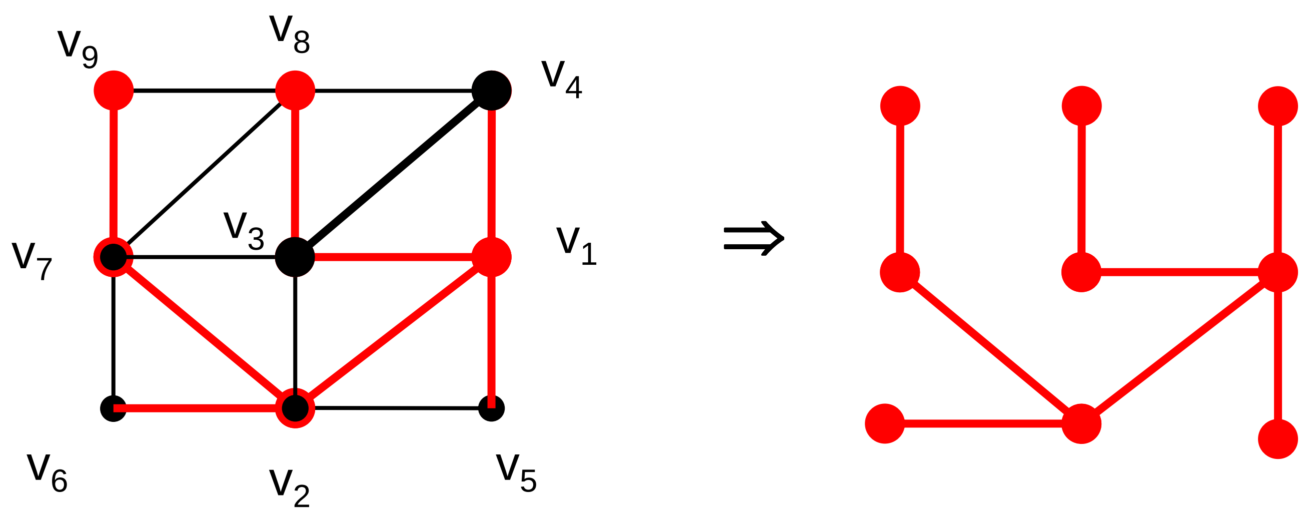
$j=8, \quad i=3$

J=9, i=4

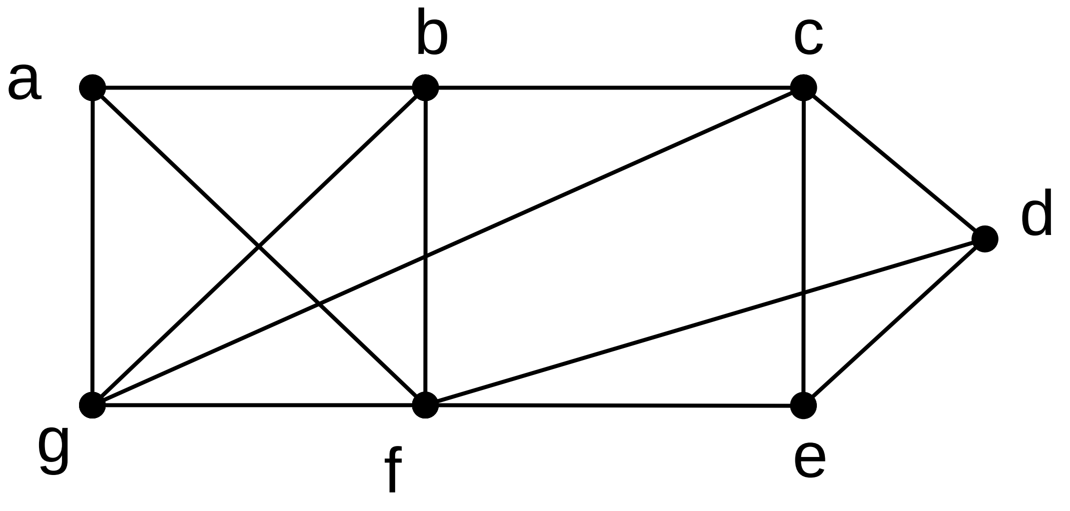
j=9, i=5

j=9, i=6

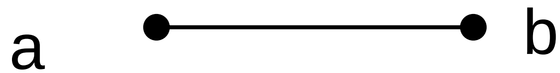
j=9, i=7



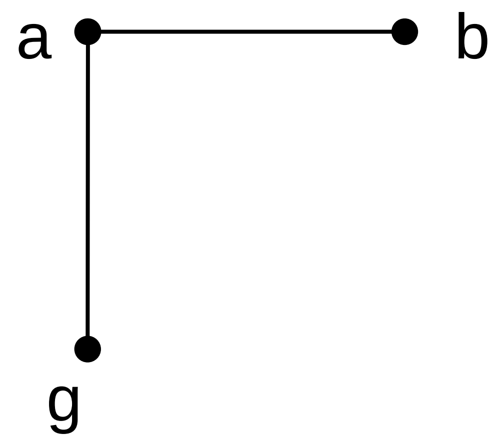
Örnek : Aşağıdaki grafın spanning (dallanmış) ağacını bulunuz.



1)

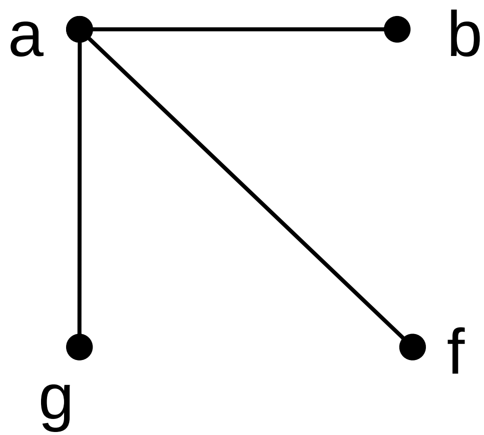


2)

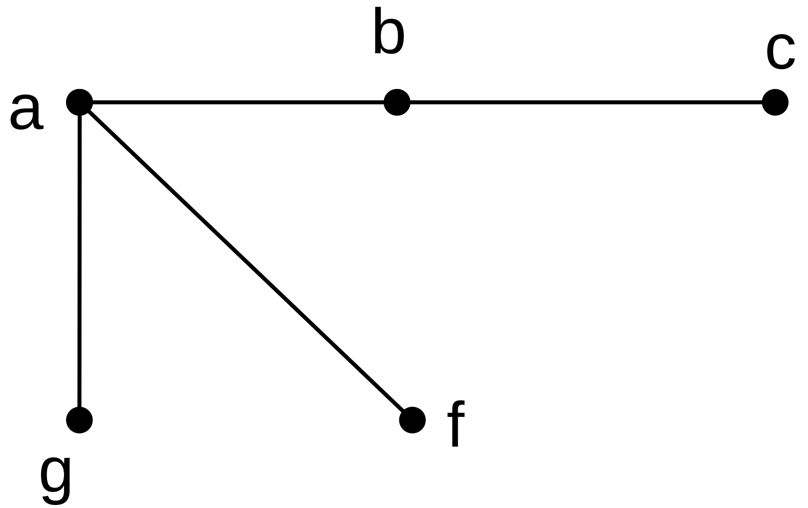


Çevre içermeyecek

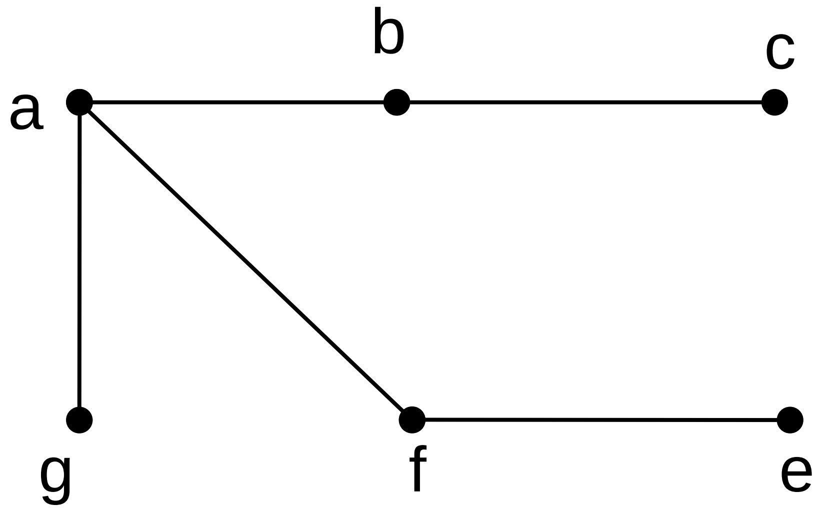
3) a tepesine bitişik ve ağaçta olmayan başka tepe seçilecek.



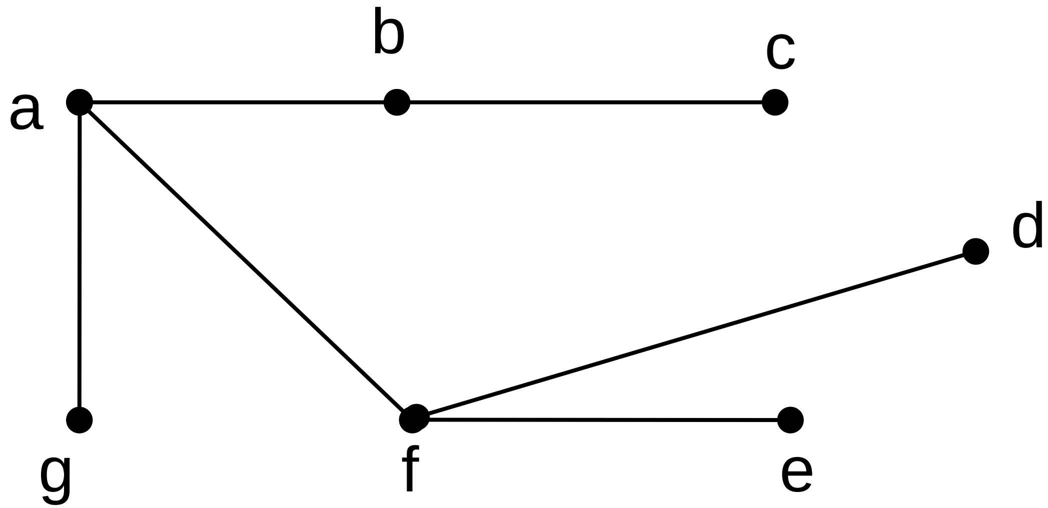
4) bc ağaçta olmadığı için seçebiliriz.



6) fe ağaçta olmadığı için seçebiliriz.



7) fd ağaçta olmadığı için seçebiliriz.

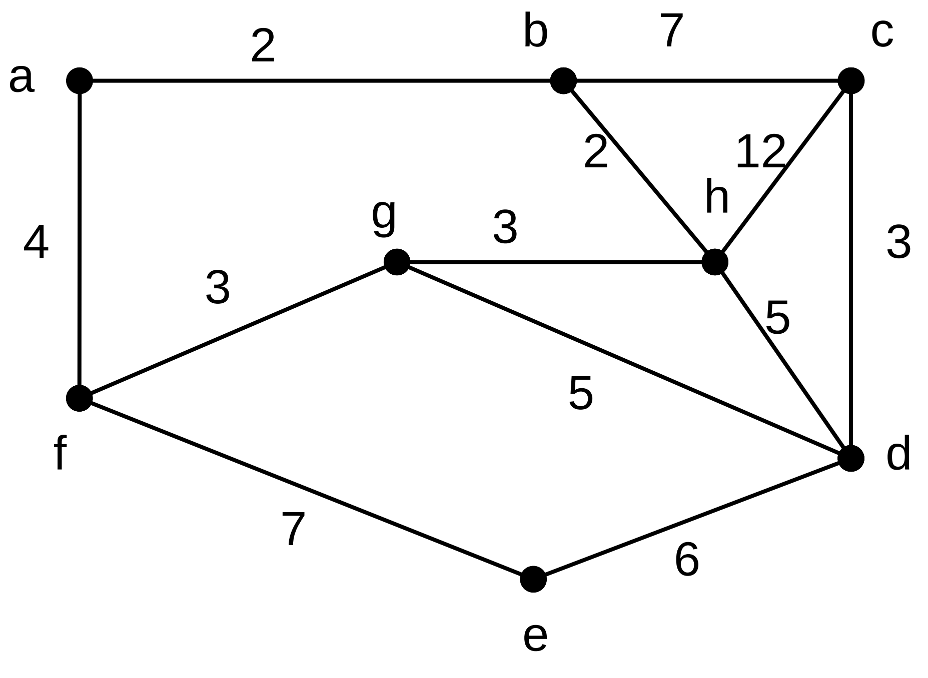


Başka ayrıt ekleyemeyiz. Hangi ayrıt eklenirse eklensin ağaç oluşur. Bu da istenmeyen durumdur.

2. Kruskal Algoritması:

Bu algoritmayı aşağıdaki problemi, ele alarak inceleyeceğiz. Aslında bu algoritma, önceki algoritma ile aynı mantığa sahip olup, sadece o algoritmanın ağırlıklı graflara uygulanmasıdır.

Problem: Bir eğlence parkının oluşturulmak istendiğini kabul edelim. Bu parkta yapılması olası olan tüm yollar daha önce belirlenmiş olup, parkın sahibi bu yollardan en az maliyetlisini seçerek, yaptırmak istiyor. Aşağıdaki grafta, parkın yapısı, olası tüm yollar ve herhangi iki nokta arasında yapılacak yolun maliyeti belirtildiğine göre en az maliyetli yolun hangisi olduğunu bulunuz.


a
b
c
d
e
f
g
h
2
7
2
12
3
4
3
3
5
5
7
6

Bu problemin çözümü için en küçük ağırlıklı dallanmış ağacı bulmalıyız. Bunun için Kruskalın algoritmasını kullanırız.

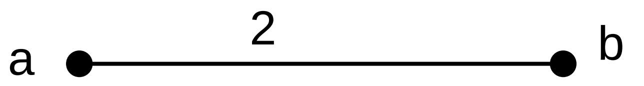
KRUSKAL ALGORİTMASI

Adım1: Graftaki en küçük ağırlıklı ayrıtı (birden fazla ise herhangi birisini) seç, ve bu ayrıt ile ağacı oluşturmaya başla.

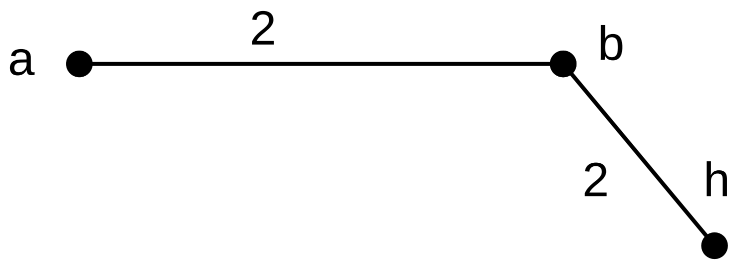
Adım2: Henüz ağaçta olmayan ve ağaca eklendiğinde çevre içermeyen en küçük ağırlıklı bir ayrıtı seç ve ağaca ekle.

Adım3: Dallanmış ağaca sahip olup olmadığını kontrol et, Eğer sahip ise dur, aksi halde Adım 2 ye git.

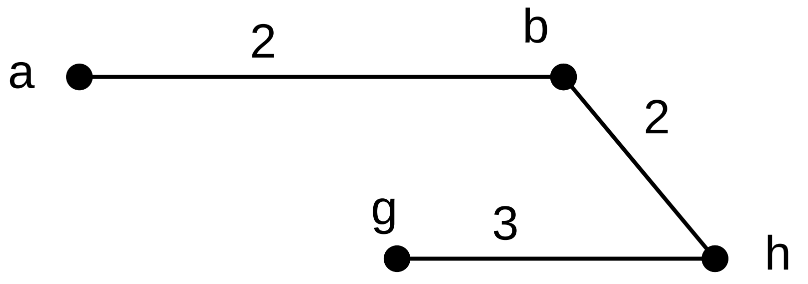
1)



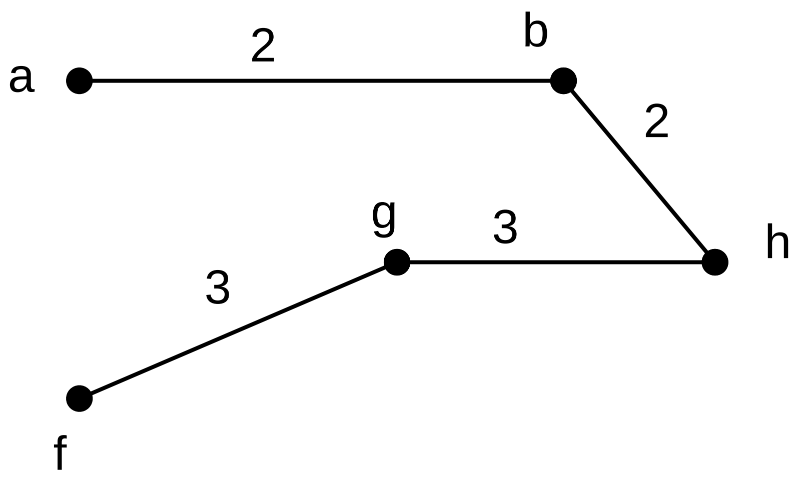
2)



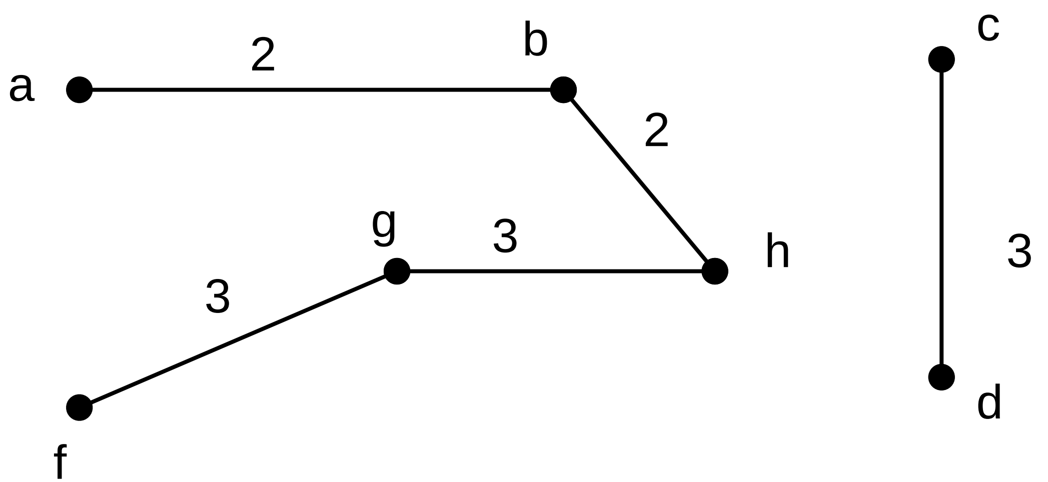
3)



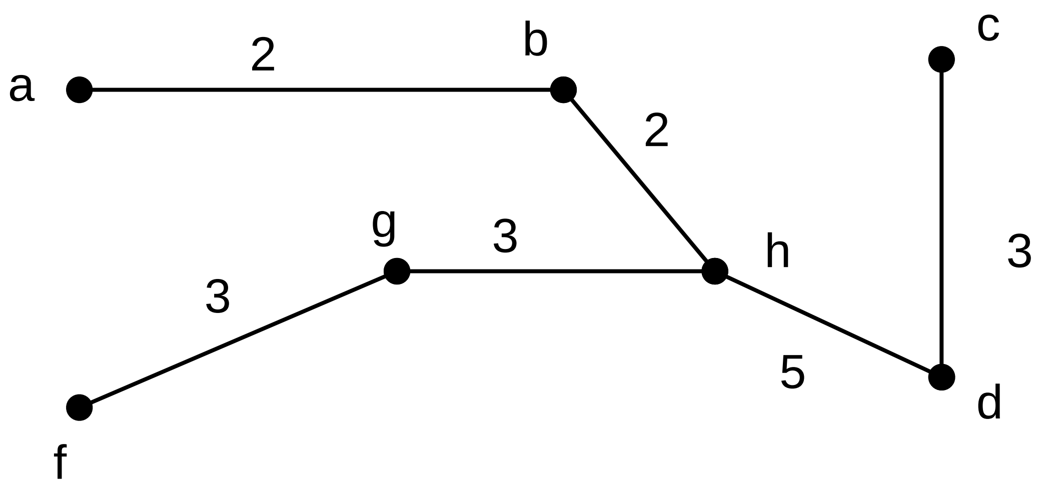
4)



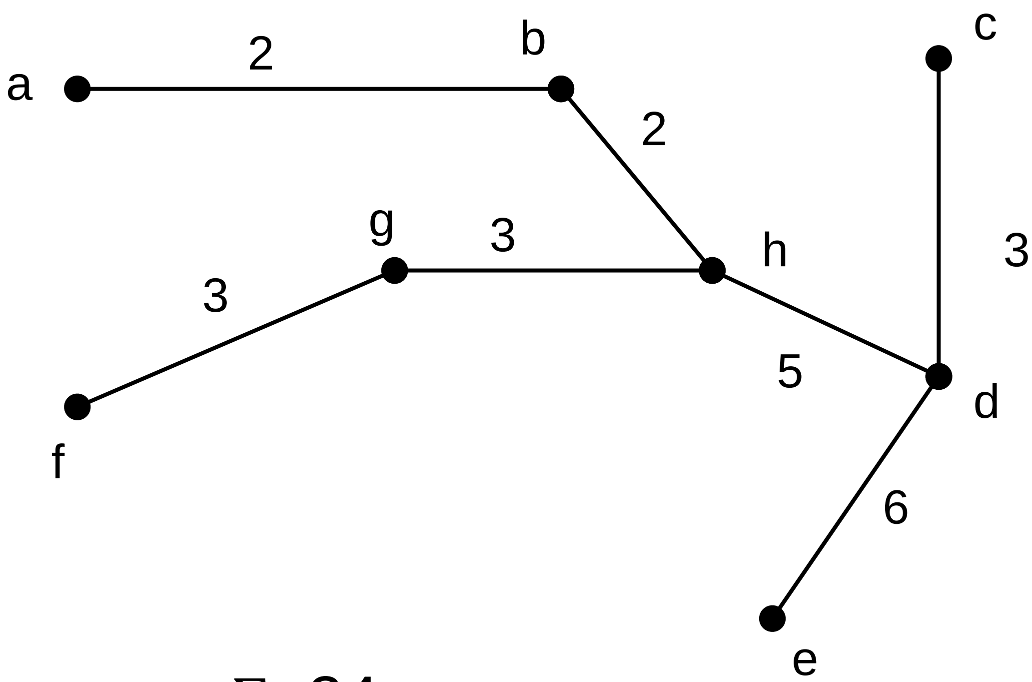
5)



6)



7)



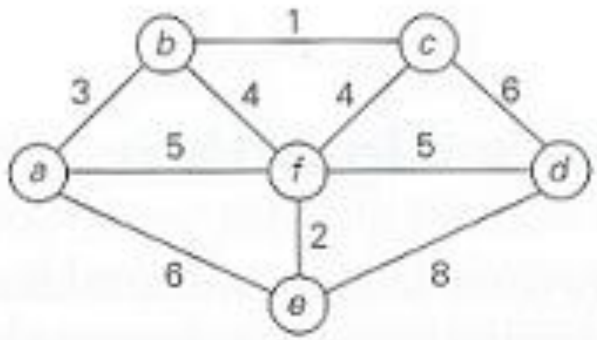
$\Sigma=24$

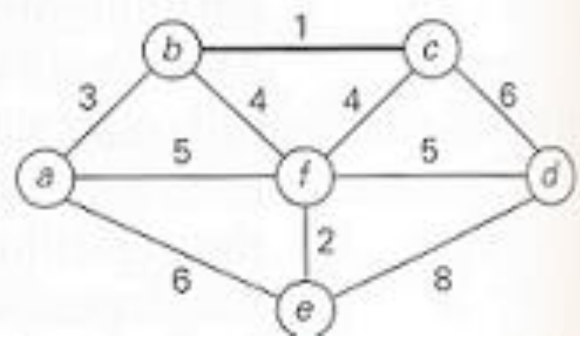
Sonuç: Bir önceki algoritma ile arasındaki fark biri rasgele dallanmış ağacı, diğeri minimum ağırlıklı dallanmış ağacı bulur.

Sözde Kod:

```
ALGORITHM  Kruskal(G)
    //Kruskal's algorithm for constructing a minimum spanning tree
    //Input: A weighted connected graph G = ⟨V, E⟩
    //Output: E_T, the set of edges composing a minimum spanning tree of G
    sort E in nondecreasing order of the edge weights w(e_{i_1}) ≤ ... ≤ w(e_{i_|E|})
    E_T ← ∅;   ecounter ← 0    //initialize the set of tree edges and its size
    k ← 0                      //initialize the number of processed edges
    while ecounter < |V| − 1 do
        k ← k + 1
        if E_T ∪ {e_{i_k}} is acyclic
            E_T ← E_T ∪ {e_{i_k}};   ecounter ← ecounter + 1
    return E_T
```

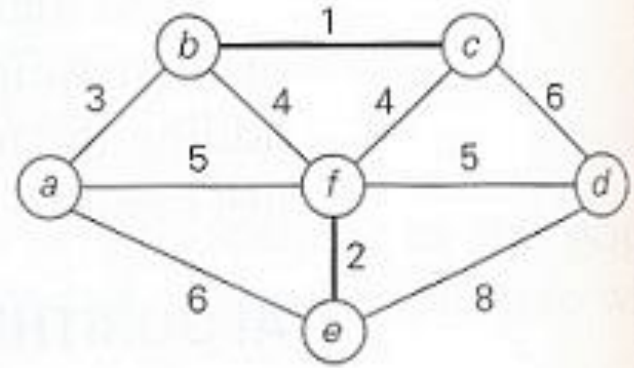
Kruskal Algoritması: Örnek



Tree edges	Sorted list of edges	Illustration
	bc cf ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	

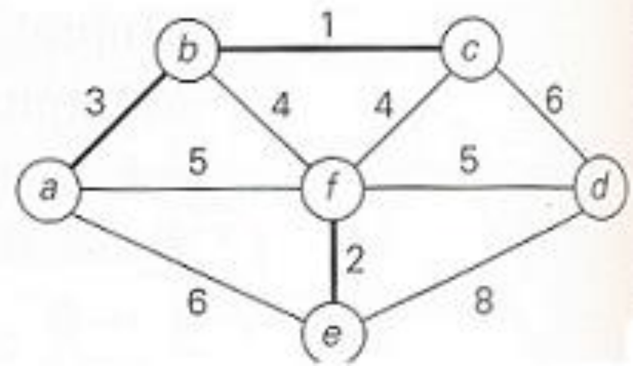
bc
1

bc	**ef**	ab	bf	cf	af	df	ae	cd	de
1	2	3	4	4	5	5	6	6	8



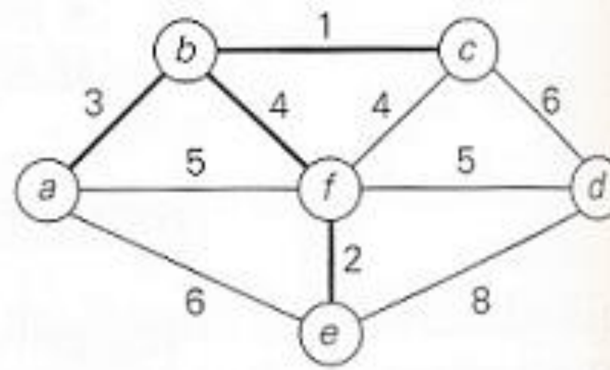
ef
2

bc	ef	**ab**	bf	cf	af	df	ae	cd	de
1	2	3	4	4	5	5	6	6	8



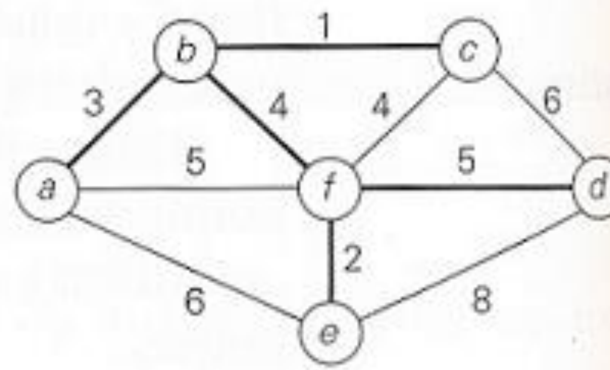
ab
3

bc	ef	ab	**bf**	cf	af	df	ae	cd	de
1	2	3	4	4	5	5	6	6	8



bf
4

bc	ef	ab	bf	cf	af	**df**	ae	cd	de
1	2	3	4	4	5	5	6	6	8



df
5

Kruskal Algoritması Analizi

ALGORITHM *Kruskal*(G)

//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights $w(e_{i_1}) \le \ldots \le w(e_{i_{|E|}})$
$\Theta(1)$ $E_T \leftarrow \varnothing$; $\quad ecounter \leftarrow 0$ $\quad$ //initialize the set of tree edges and its size
$\Theta(1)$ $k \leftarrow 0$ $\quad$ //initialize the number of processed edges
while $ecounter < |V| - 1$ **do**
$\quad k \leftarrow k + 1$
$\quad$ **if** $E_T \cup \{e_{i_k}\}$ is acyclic
$\quad\quad E_T \leftarrow E_T \cup \{e_{i_k}\}$; $\quad ecounter \leftarrow ecounter + 1$
return E_T

$(|E|+|V|)\cdot\alpha(|V|)$

Çevreyi kontrol eder.

Ayrıt sıralamadan geldi: $O(E \cdot \lg E)$

Cycle için ayrık-set veri yapısı kullanılırsa

Çalışma süresi;

$$\underline{O((|E|+|V|)\cdot\alpha(V))} + \underline{O(|E|\cdot\lg|E|)}$$

$|E| > |V|-1$ old. dan

$$O(E\cdot\alpha(V)) + O(E\cdot\lg E)$$

$$|\alpha(V)| = O(\lg V) = O(\lg E)$$

Böylece toplam süre:

$O(E \cdot \lg E)$ 'dir. (Sıralamadan gelir.)

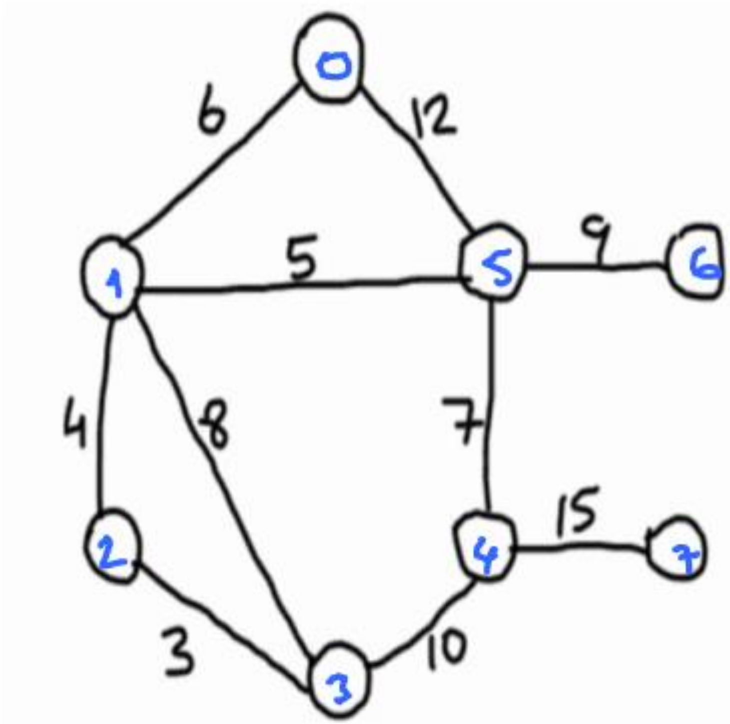
$|E| \leq |V|^2$ old. dan

$\log|E| \leq 2\log|V|$

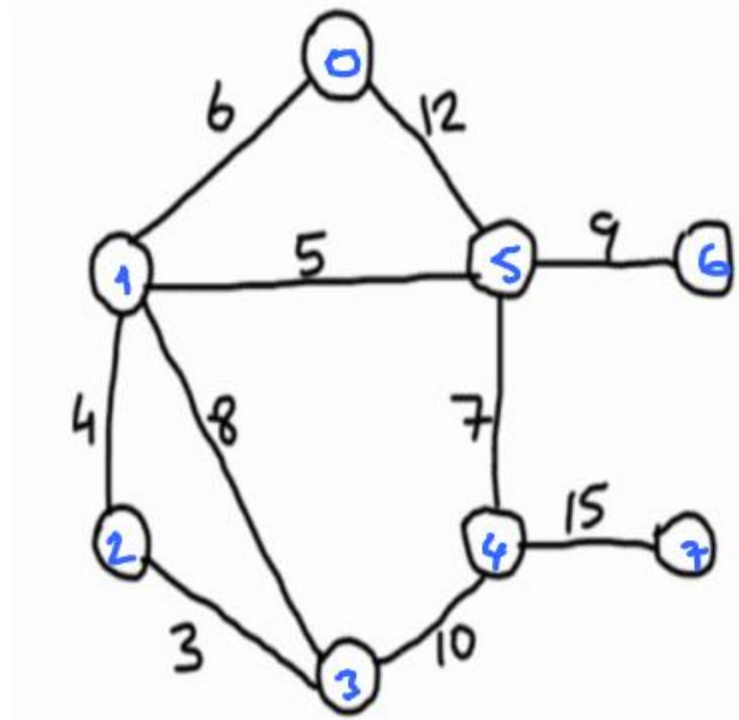
$O(\lg E) = O(\lg V)$ olur.

Sonuç olarak: $O(E \cdot \lg V)$ elde edilir.

Kruskal Algoritması Nasıl programlanabilir?



1. tepe	2. tepe	Ağrlık
0	1	6
1	2	4
2	3	3
3	4	10
4	5	7
5	0	12
1	5	5
1	3	8
5	6	9
4	7	15



Ağırlığa göre sırala: $O(E \cdot \lg E)$'lik bir algoritma ile.

1. tepe	2. tepe	Ağırlık
2	3	3
1	2	4
1	5	5
0	1	6
4	5	7
1	3	8
5	6	9
3	4	10
5	0	12
4	7	15

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

Tepelerin ziyaret edilip edilmediği kontrolü

1.adım

0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0

1. tepe	2. tepe	Ajırlık
2	3	3 → Ayrıt Eklendi.
1	2	4
1	5	5
0	1	6
4	5	7
1	3	8
5	6	9
3	4	10
5	0	12
4	7	15

2.adım

0	1	2	3	4	5	6	7
0	1	1	1	0	0	0	0

1.tepe	2.tepe	Ağırlık	
2	3	3	→ Ayrıt Eklendi.
1	2	4	→ Ayrıt Eklendi.
1	5	5	
0	1	6	
4	5	7	
1	3	8	
5	6	9	
3	4	10	
5	0	12	
4	7	15	

3. adım

0	1	2	3	4	5	6	7
0	1	1	1	0	1	0	0

1. tepe	2. tepe	Ağırlık	
2	3	3	Ayrıt Eklendi.
1	2	4	Ayrıt Eklendi.
1	5	5	Ayrıt Eklendi.
0	1	6	
4	5	7	
1	3	8	
5	6	9	
3	4	10	
5	0	12	
4	7	15	

4. adım

0	1	2	3	4	5	6	7
1	1	1	1	0	1	0	0

1. tepe	2. tepe	Ağırlık	
2	3	3	→ Ayrıt Eklendi.
1	2	4	→ Ayrıt Eklendi.
1	5	5	→ Ayrıt Eklendi.
0	1	6	→ Ayrıt Eklendi.
4	5	7	
1	3	8	
5	6	9	
3	4	10	
5	0	12	
4	7	15	

5. adım

0	1	2	3	4	5	6	7
1	1	1	1	1	1	0	0

1. tepe	2. tepe	Ağırlık	
2	3	3	→ Ayrıt Eklendi.
1	2	4	→ Ayrıt Eklendi.
1	5	5	→ Ayrıt Eklendi.
0	1	6	→ Ayrıt Eklendi.
4	5	7	→ Ayrıt Eklendi.
1	3	8	
5	6	9	
3	4	10	
5	0	12	
4	7	15	

Çözüm

1. tepe	2. tepe	Ağırlık	
2	3	3	Ayrıt Eklendi.
1	2	4	Ayrıt Eklendi.
1	5	5	Ayrıt Eklendi.
0	1	6	Ayrıt Eklendi.
4	5	7	Ayrıt Eklendi.
1	3	8	Eklenmedi.
5	6	9	
3	4	10	
5	0	12	
4	7	15	

7. adım

0	1	2	3	4	5	6	7
1	1	1	1	1	1	1	0

1. tepe	2. tepe	Ağırlık	
2	3	3	→ Ayrıt Eklendi.
1	2	4	→ Ayrıt Eklendi.
1	5	5	→ Ayrıt Eklendi.
0	1	6	→ Ayrıt Eklendi.
4	5	7	→ Ayrıt Eklendi.
1	3	8	→ Eklenmedi.
5	6	9	→ Ayrıt Eklendi.
3	4	10	
5	0	12	
4	7	15	

8. ve 9. adımda Ayrıtlar Eklenmez

1. tepe	2. tepe	Ağırlık	
2	3	3	→ Ayrıt Eklendi.
1	2	4	→ Ayrıt Eklendi.
1	5	5	→ Ayrıt Eklendi
0	1	6	→ Ayrıt Eklendi
4	5	7	→ Ayrıt Eklendi
1	3	8	→ Eklenmedi.
5	6	9	→ Ayrıt Eklendi.
3	4	10	→ Eklenmedi
5	0	12	→ Eklenmedi
4	7	15	

10.adım

0	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1

1. tepe	2. tepe	Ağırlık	
2	3	3	→ Ayrıt Eklendi.
1	2	4	→ Ayrıt Eklendi.
1	5	5	→ Ayrıt Eklendi.
0	1	6	→ Ayrıt Eklendi.
4	5	7	→ Ayrıt Eklendi.
1	3	8	→ Eklenmedi.
5	6	9	→ Ayrıt Eklendi.
3	4	10	→ Eklenmedi:
5	0	12	→ Eklenmedi:
4	7	15	→ Ayrıt Eklendi:

$w(T) = 3+4+5+6+7+9+15 = 49$.

3. PRİM ALGORİTMASI

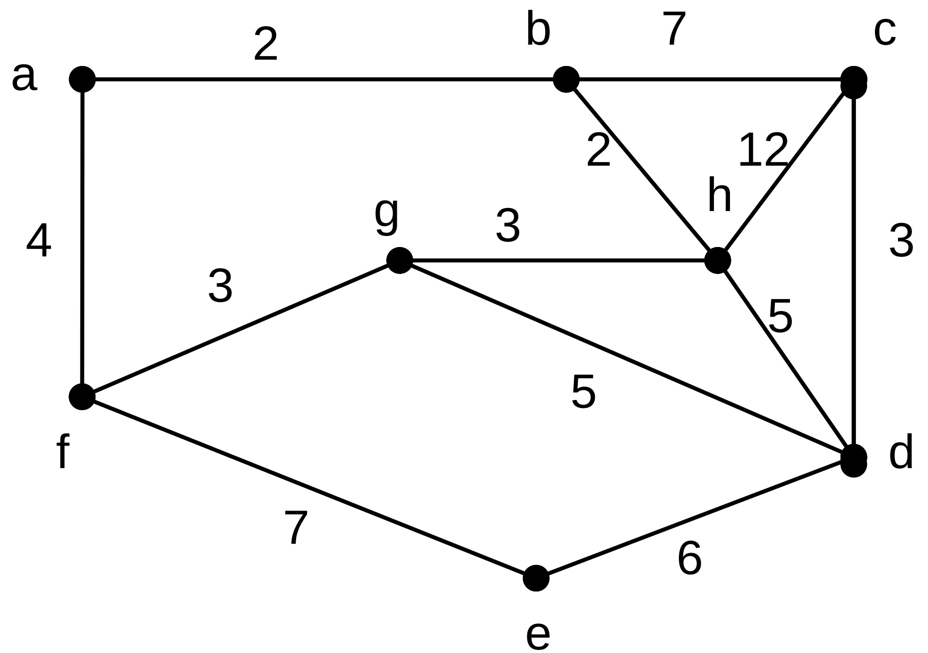
Ağırlıklandırılmış ve yönsüz graflarda dallanmış alt ağacı bulur.

Adım1: Graftaki herhangi *v* tepesi seç, ve bu tepe ile birlikte en düşük maliyetli ayrıt ile ağacı oluşturmaya başla.

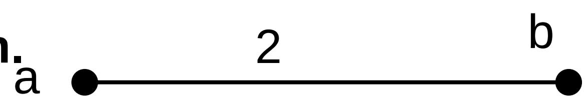
Adım2: Henüz ağaçta olmayan, ziyaret edilmiş tepelere bitişik olan ve ağaca eklendiğinde çevre içermeyen en küçük ağırlıklı bir ayrıtı seç ve ağaca ekle.

Adım3: Dallanmış ağaca sahip olup olmadığını kontrol et, Eğer sahip ise dur, aksi halde Adım 2 ye git.

Örnek:



a tepesinden başlayalım.

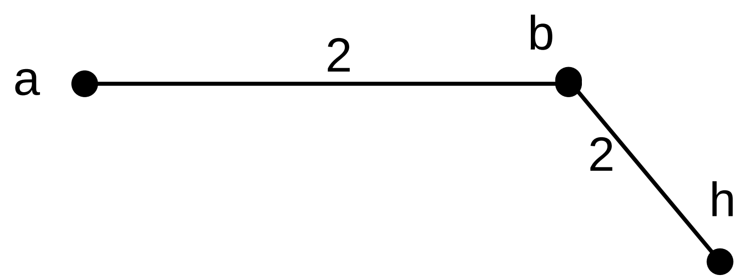


2. adım

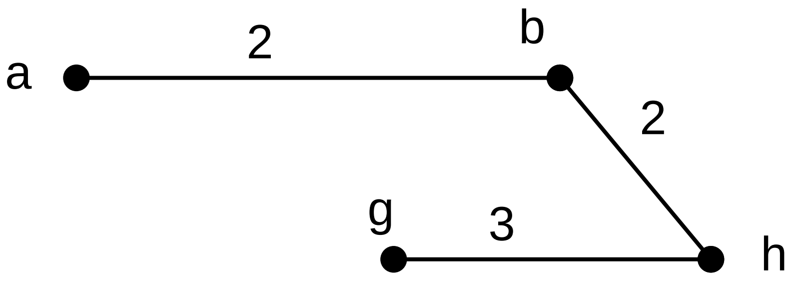
a – f arası :4

b – c arası :7

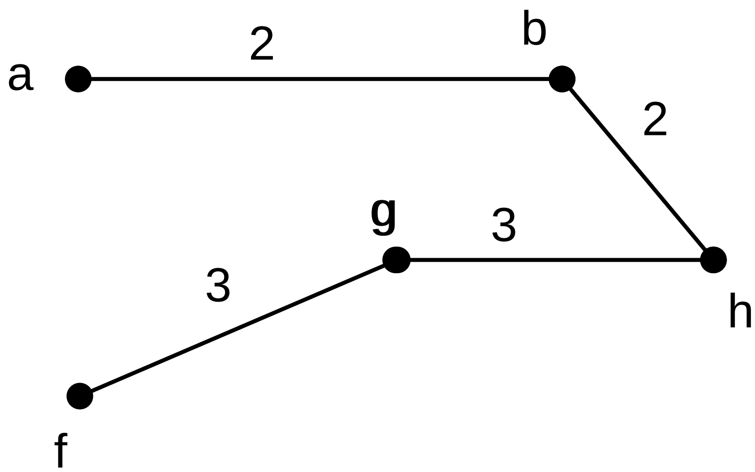
b – h arası : 2 2. adımda eklenir.



3. adım
a – f arası :4
b – c arası :7
h – g arası : 3   3. adımda eklenir.
h – c arası : 12
h – d arası : 5

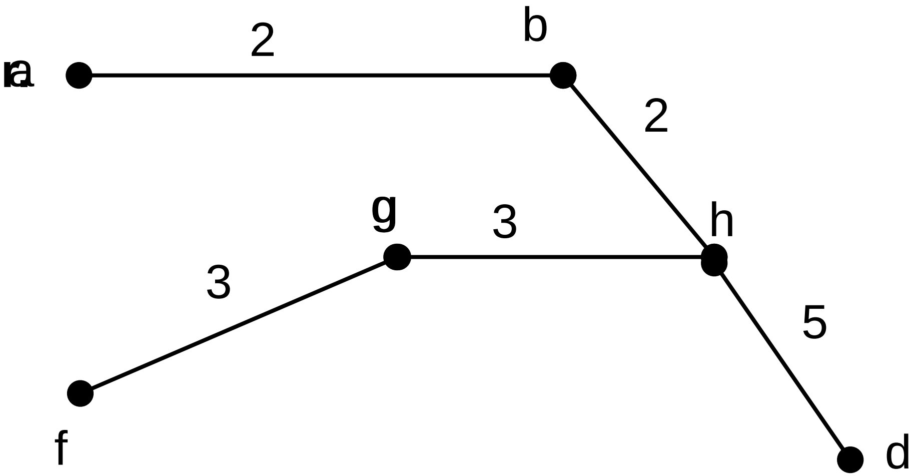


4. adım
a – f arası :4
b – c arası :7
h – c arası : 12
h – d arası : 5
g – d arası : 5
g – f arası : 3   4. adımda eklenir.



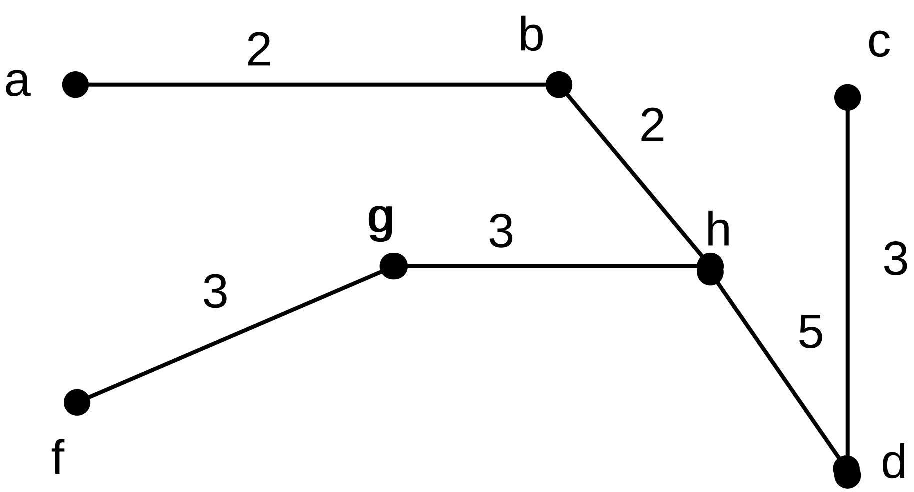
5. adım

a – f arası :4 eklenemez, çevre olur.
b – c arası :7
h – c arası : 12
h – d arası : 5 5. adımda eklenir.
g – d arası : 5
f – e arası: 7

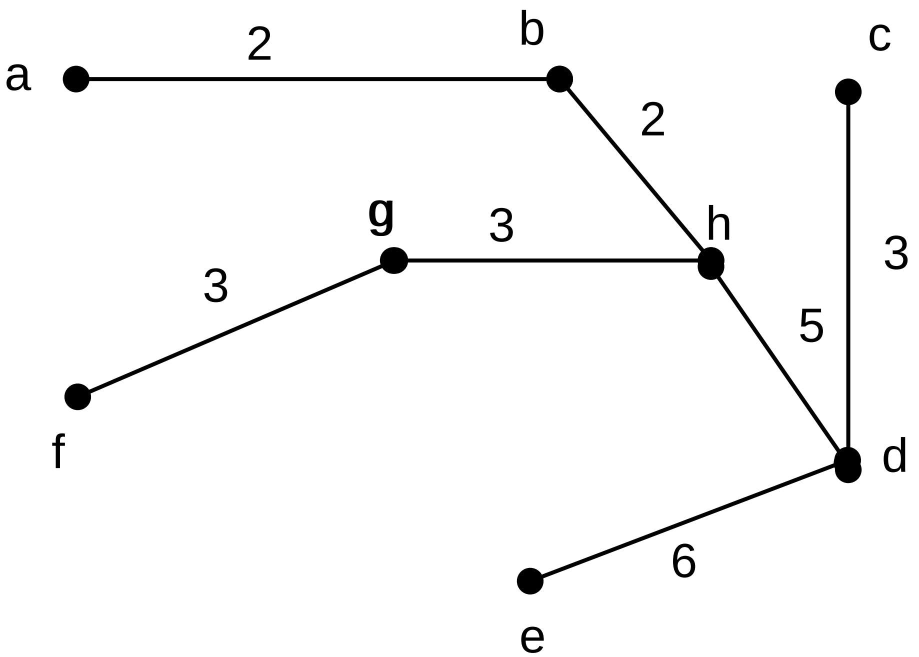


6. adım

b – c arası :7
h – c arası : 12
g – d arası : 5
f – e arası: 7
d – c arası: 3 6. adımda eklenir.
d – e arası: 6



7. adım
b – c arası :7
h – c arası : 12
g – d arası : 5 eklenmez, çevre oluşturur.
f – e arası: 7
d – e arası: 6 7. adımda eklenir.



8. adım
Dallanmış alt ağaç oluştu, dur.

Sonuç: Kruskal algoritması ile aynı minimum ağırlıklı dallanmış ağacı bulur.

Prim Algoritması (Aç gözlü Yaklaşım)
Sözde Kod:

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

Pİ değerleri minimum kapsayan ağaçtaki ayrıtlardır.

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

→ G'nin tepeleri

→ ilk olarak tüm tepeler ∞ olur.

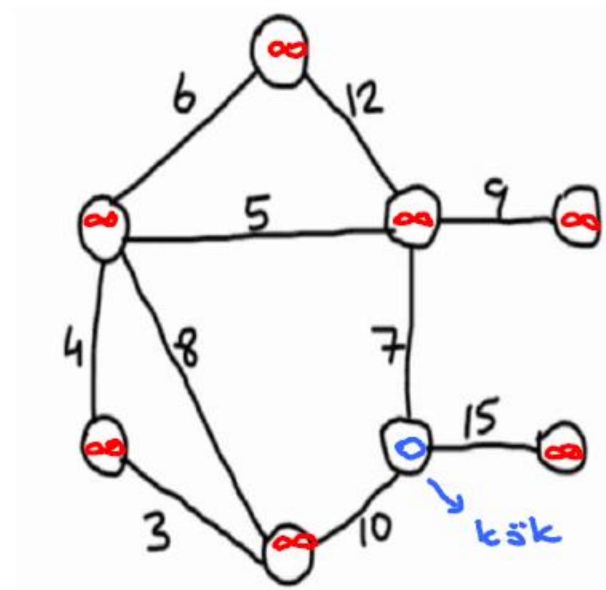
→ ağacın ilk kökü sıfır olur.

→ kuyruğa at (tüm tepeleri)

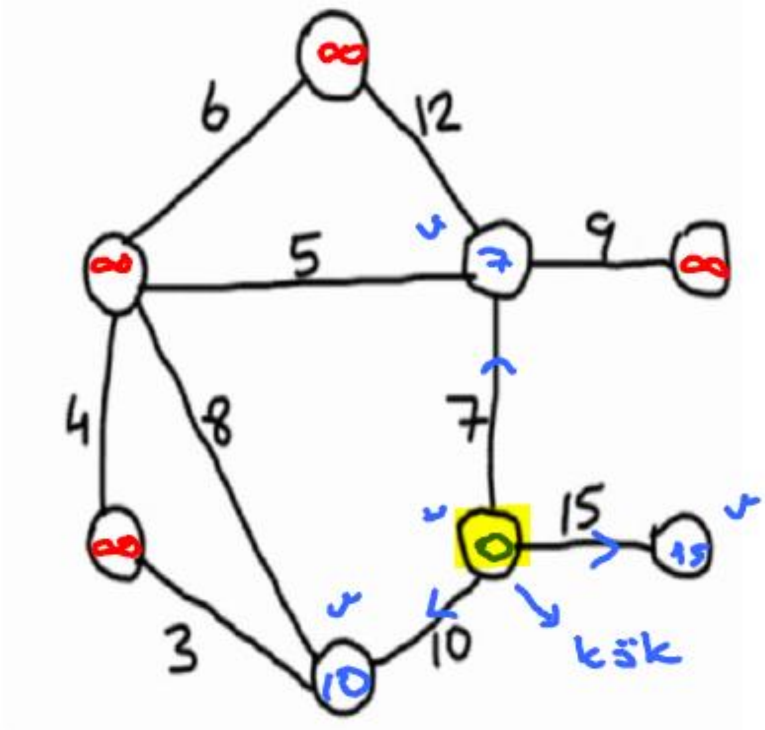
→ kuyruk boştan farklı old. sürece çalış

→ minimum olanı kuyruktan çıkar.

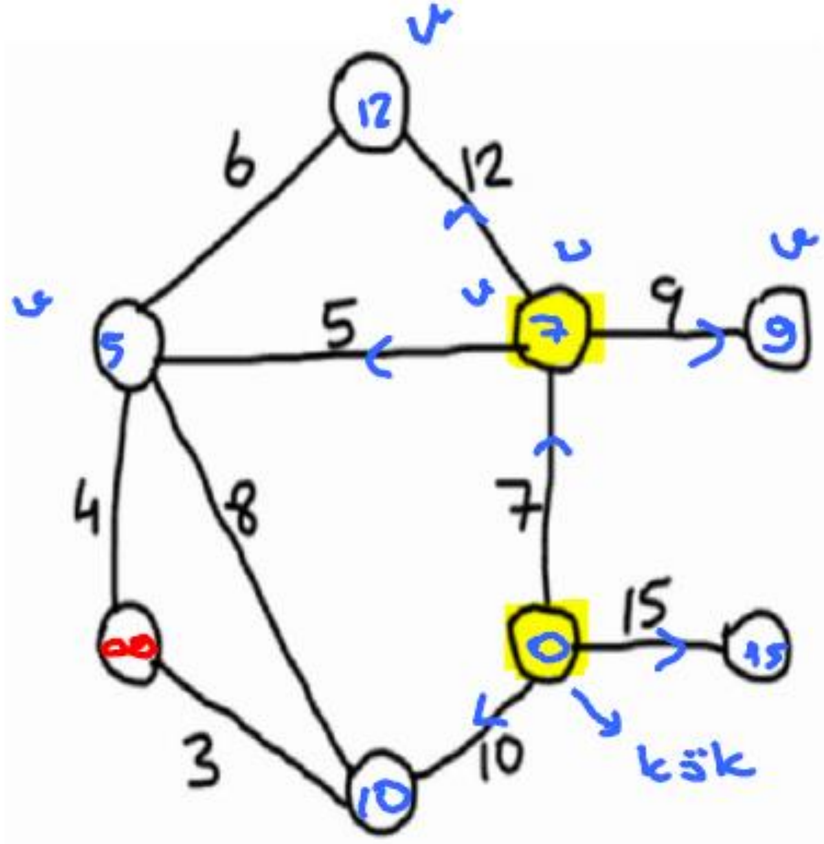
u'nun her bir komşusu için çalış.



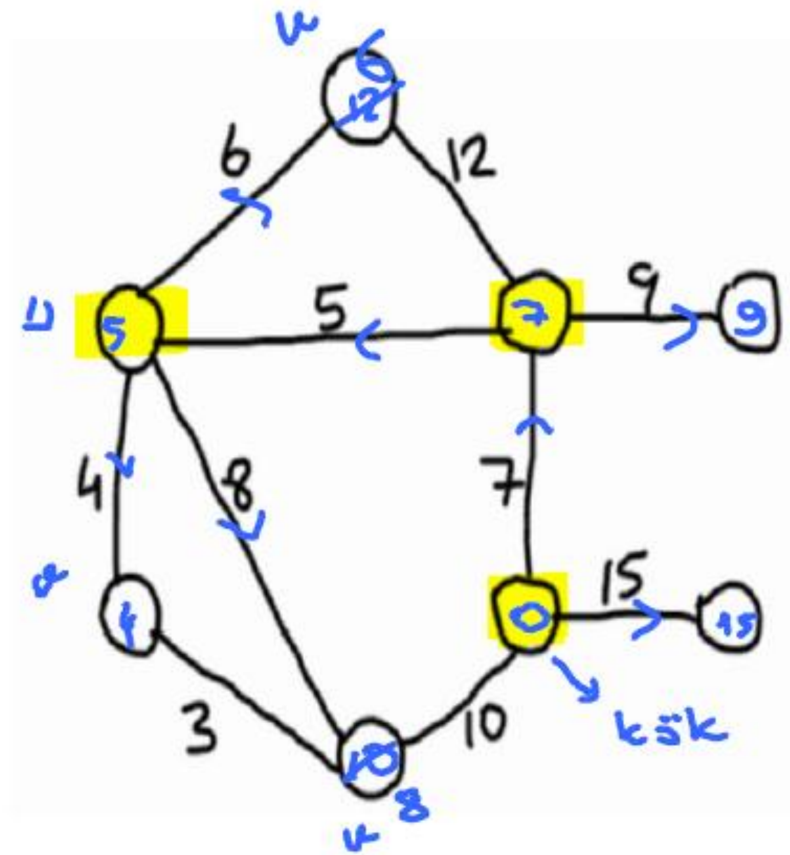
1. adım



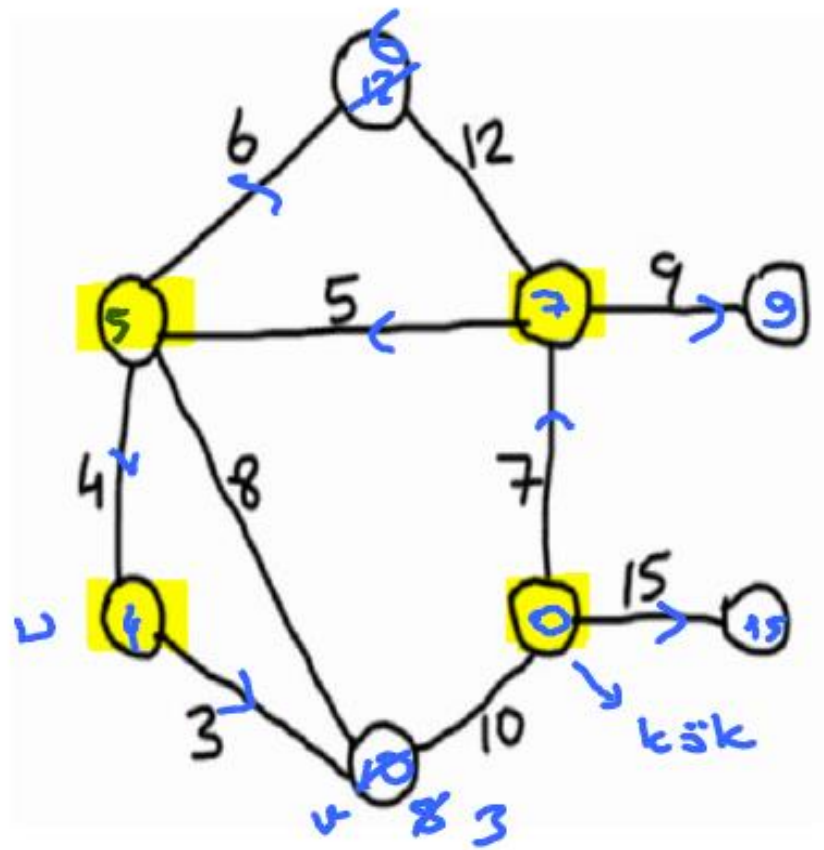
2.adım Kuyruktan min. key 'li tepeyi çıkar.



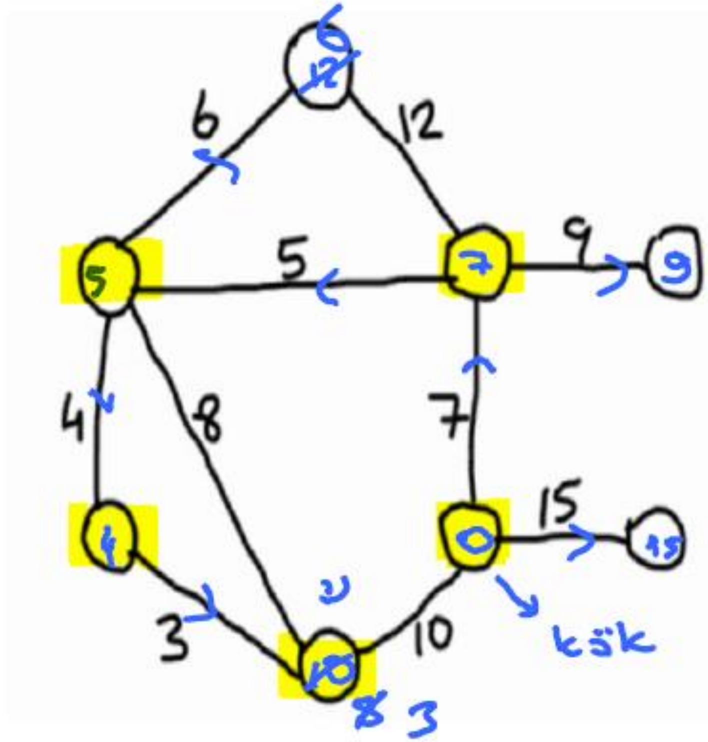
3. adım   Kuyruktan min. key li tepeyi çıkar.



4. adım Kuyruktan min. key'li tepeyi çıkar.

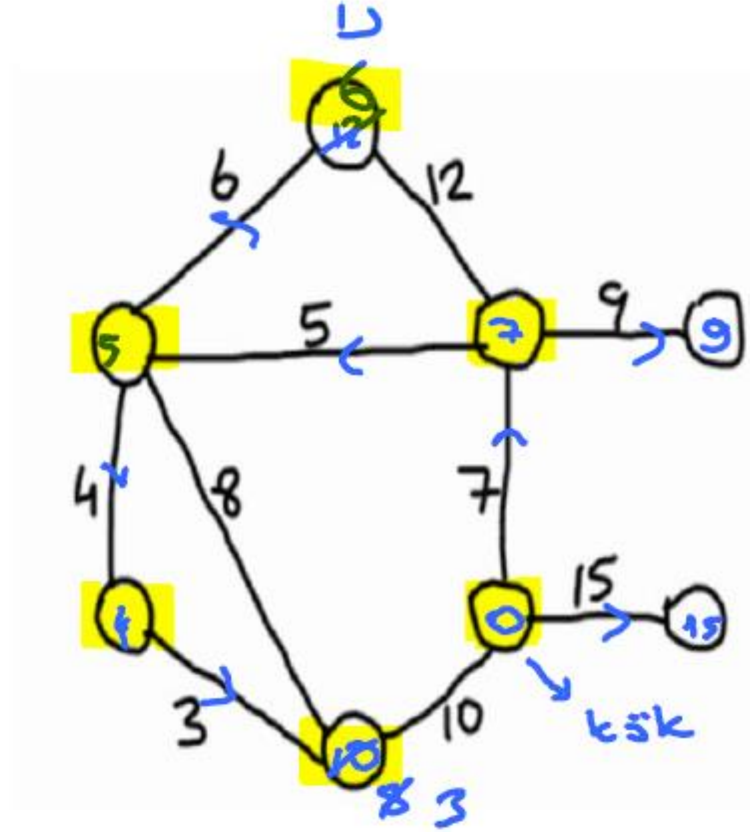


5. adım Kuyruktan min. key'li tepeyi çıkar.



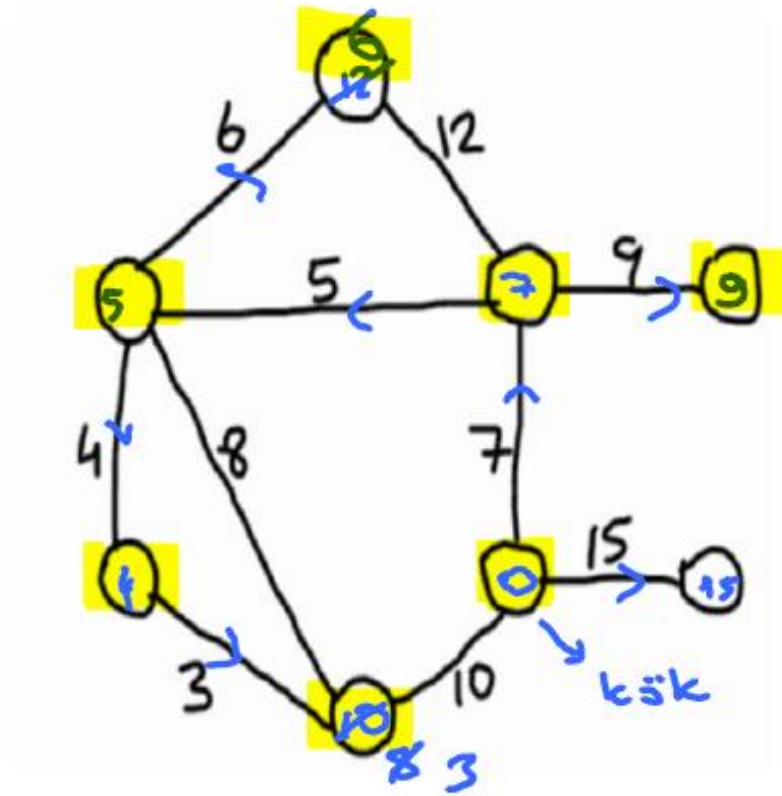
Kuyrukta olan, u'nun komşusu yok.

6. adım Kuyruktan min. key'li tepeyi çıkar.



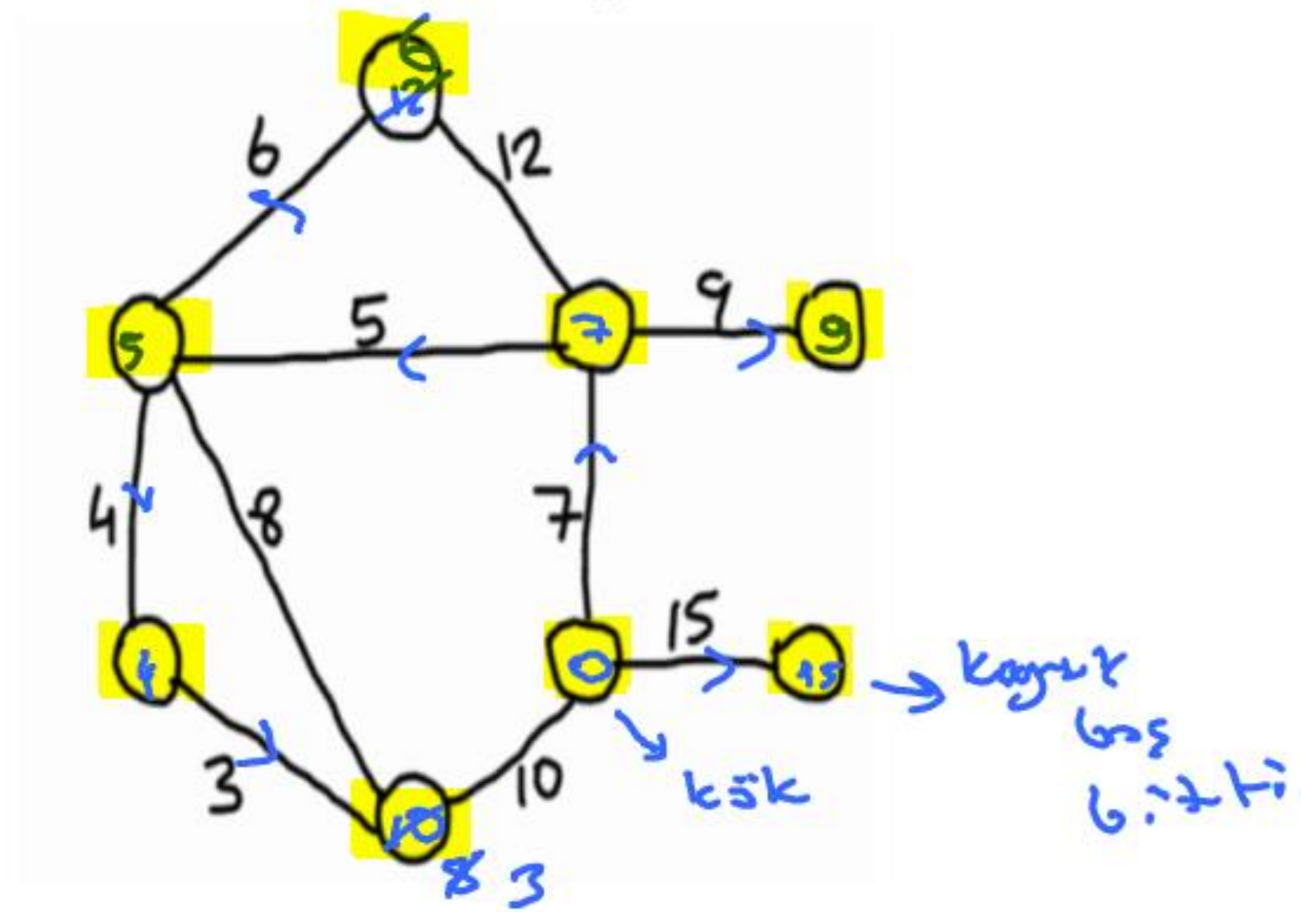
kuyrukta olan, 15'in komşusu yok.

7 adım kuyruktan min. key li tepeyi çıkar.

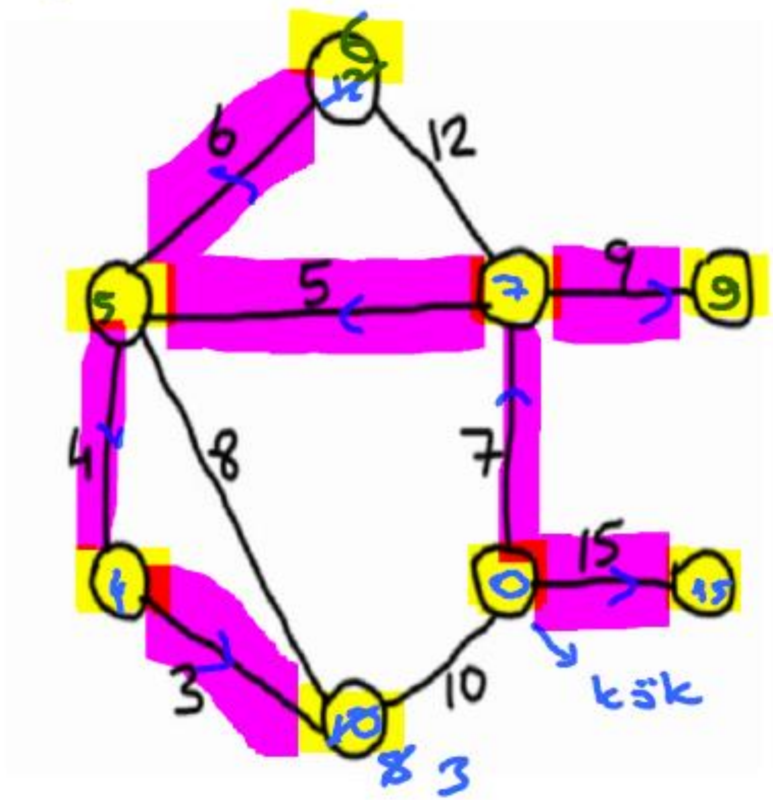


kuyrukta olan, u'nun komşusu yok.

8. adım Kuyruktan min. key'li tepeyi çıkar.



Son durum:



$w(T) = 7 + 5 + 4 + 3 + 6 + 9 + 15 = 49$.

Prim Algoritmasının Analizi

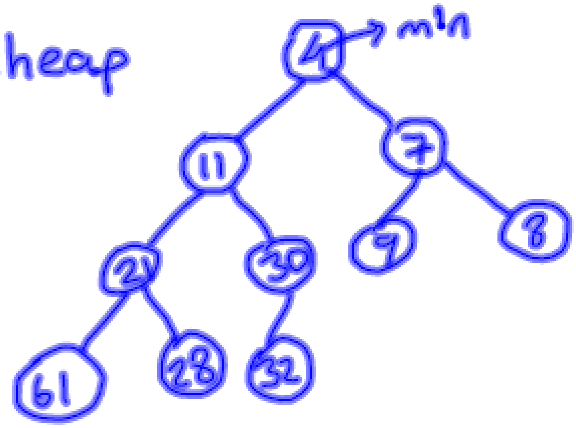
MST-PRIM(G, w, r)

```
|V| defa {  for each u ∈ G.V
                u.key = ∞
                u.π = NIL
Θ(1)  ←     r.key = 0
Θ(V)  ←     Q = G.V
            while Q ≠ ∅
                u = EXTRACT-MIN(Q)                → kuyruk
Θ(V)   derece(u) {  for each v ∈ G.Adj[u]
                    if v ∈ Q and w(u, v) < v.key  } Gevşetme adımı
                        v.π = u
                        v.key = w(u, v)           → decrease_key
```

$$T(|V|) = \Theta(V) \cdot T_{extract\_min} + \Theta(E) \cdot T_{decrease\_key}$$

Q	$T_{extract\_min}$	$T_{decrease\_key}$	Toplam
array	$\Theta(V)$	$\Theta(1)$	$\Theta(V^2)+\cancel{\Theta(E)}=\Theta(V^2)$
heap	$\Theta(\lg V)$	$O(\lg V)$	$\cancel{\Theta(V\lg V)}+\Theta(E\lg V)=\Theta(E\lg V)$

min_heap



$h = \lg V$

KAYNAKLAR

[1] Chartrand, G.-Lesniak, L., (1986) : *Graphs and Digraphs*, Wadsworth & Brooks, California

[2] West D.B. (2001) : *Introduction to Graph Theory*, *Prentice Hall*, USA.

[3] Graf Teoriye Giriş, Şerife Büyükköse ve Gülistan Kaya Gök, Nobel Yayıncılık

[4] Discrete Mathematical Structures for Computer Science, Ronald E. Prather, Houghton Mifflin Company, (1976).

[5] Christofides, N., 1986. Graph Theory an Algorithmic Approach, Academic Press, London

[6] Algoritmalar (Teoriden Uygulamalara), Vasif V. NABİYEV, Seçkin Yayıncılık