SAKARYA ÜNİVERSİTESİ

Ayrık İşlemsel Yapılar

Hafta 12

Prof.Dr. Nilüfer YURTAY

Ağaçlar

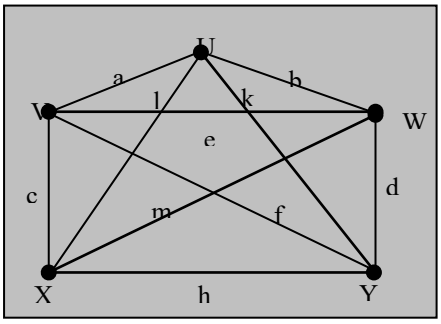
12.1 Ağaçların özellikleri

Grafların bir özel durumu olan ağaçlar bilgisayar biliminde geniş bir alanda kullanılmaktadır. Ağaçlar ilk olarak 1847 de Gustav Kirchhoff tarafından elektrik devrelerinde kullanılmıştır. Daha sonra Arthur Cayley tarafından kimyada kullanılmıştır. Günümüzde bilgisayar bilimlerinde veriyi organize etmek ve manipüle etmek için çok yaygın olarak kullanılmaktadır.

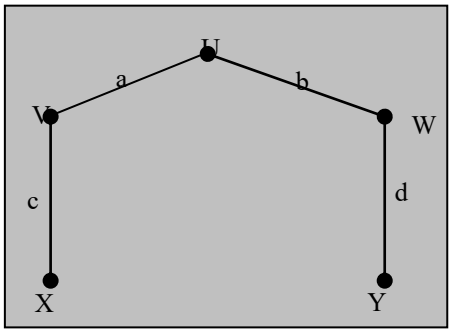
Varsayalım ki 5 ayrı şehir arasındaki telefon şebekesi kurmak istiyoruz. Her hangi iki şehrin arasında hat olsun istiyoruz. Zaman ve maliyet açısından mümkün olduğunca az hatla problemi çözmeliyiz. İki şehir arasında iletişim kurulması şart ancak bu iletişim diğer şehirlere olan hatlar üzerinden de kurulabilir. Şehirleri düğümlerle , mevcut hatları kenarlar olarak gösterirsek elde edeceğimiz graf mevcut telefon şebekesini modelleyecektir.

Şekil 12.1 de 5 şehir arasındaki olan tüm hatlar vardır. Bizim istediğimiz ise bu kenarlar arasında öyle kenarları seçelim ki bunlarla herhangi iki şehir arasında bağlantı(yol) olsun .

Şekil 12.2 de a,b,c,d kenarları buna bir önektir.



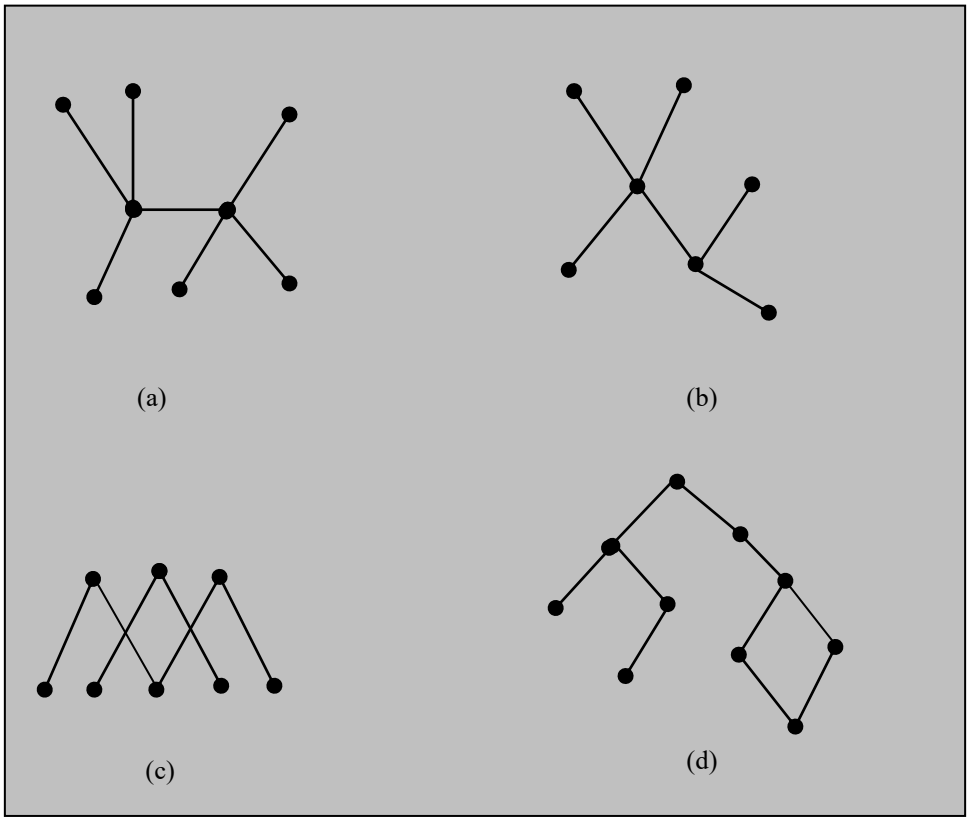
Şekil 12.1



Şekil 12.2

Şekilden görüldüğü gibi herhangi iki şehir arasında daima bir yol vardır. Şekil 4.2 deki grafı incelersek graf bağlı (connected) dır ve döngü yoktur. Bağlı ve döngü içermeyen bir grafa ağaç (tree) adı verilir.

Şekil 12.3(a) ve (b) birer ağaçtır. Şekil 12.3 (c) bağlı olmadığı için , şekil 12.3(d) ise döngü içerdiği için ağaç değildir.



Şekil 4.3

Teorem

U ve V ağaca ait iki düğüm olsun. U ve V arasında sadece bir adet basit yol vardır.

Teorem

Birden fazla düğümü olan bir T ağacında derecesi 1 olan en az iki düğüm vardır.

Teorem

N düğümlü bir ağaçta n-1 kenar vardır.

Teorem

(a) Bir ağaçtan bir kenar kaldırılırsa , kalan graf bağlı değildir,ağaç değildir.

(b) Bir ağaca bir kenar eklenirse (düğüm eklemeksizin) sonuç graf döngü içerir ve artılı ağaç değildir.

Teorem

Bir T grafı için aşagıdakiler eşdeğerdir

(a) T bir ağaçtır.

(b) T bağlıdır ve düğüm sayısı kenar sayısından bir fazladır.

(c) T de döngü yoktur ve düğüm sayısı kenar sayısından bir fazladır.

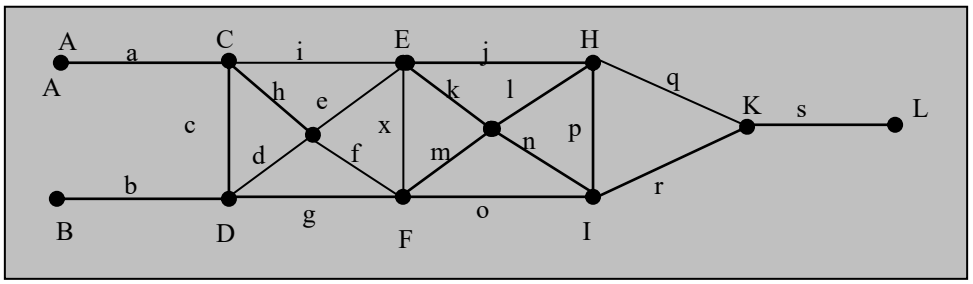
(d) T deki herhangi iki düğüm arasında sadece bir tek basit yol vardır.

(e) T baglıdır ve T den herhangi bir kenarın çıkarılması sonucu kalan graf bağlı değildir.

(f) T de döngü yoktur ve komşu olmayan iki düğüm arasına eklenen bir kenar sonuçta döngü içeren bir graf oluşturur.

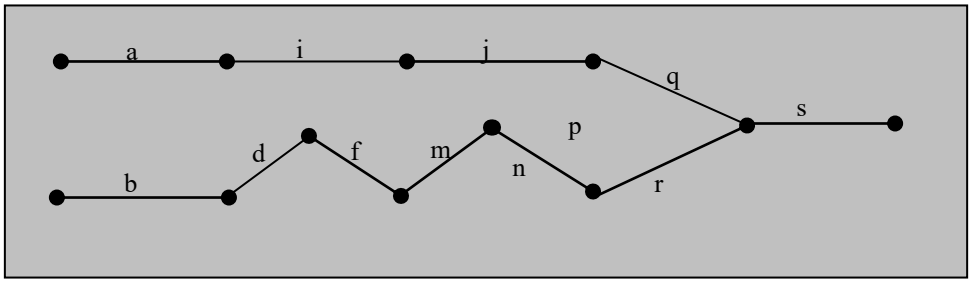
12.2 Kapsama (Spanning) Ağaçları

Çoğu problemde verilen bir grafta tüm düğümleri içerecek ağacın bulunması istenir. Basınçlı hava ile açma kapama yapan kesicilerin bulunduğu bir şalt sahasında 12 adet hava tankı olsun. Bu tankların hepsinin birbiri ile boru bağlantısı bulunmaktadır. Şekil 12.4a'da bu bağlantılar gösterilmektedir. Minimum sayıda boru kullanarak bağlantı nasıl yapılabilir ?



Şekil 12.4a

Problem bu durumuyla kapsama ağacının bulunması problemidir. Çözüm ise döngü içermeyecek ve tüm düğümleri ziyaret edecek şekilde kenarların bir alt kümesini bulmak olacaktır. Şekil 12.4b'deki a,i,j,q,s,r,n,m,f,d,b yolu tüm düğümleri içerdiğinden ve döngü içermediğinden mümkün bir kapsama ağacıdır.

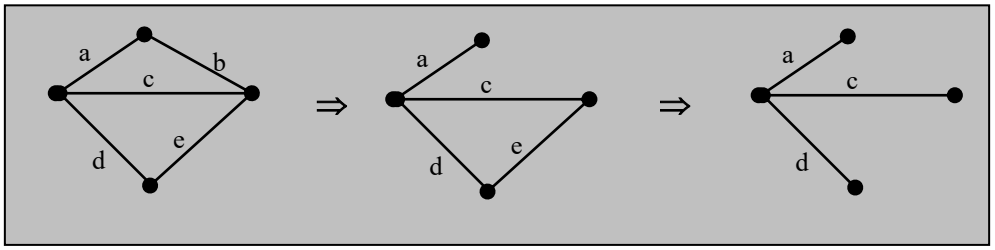


Şekil 12.4b

Bir G grafının, bir kapsama ağacı, grafın düğüm ve kenarlarını kullanarak G nin tüm düğümlerini içeren bir ağaçtır.

Grafın kendisi bir ağaç ise kapsama ağacı da kendisidir. Genel olarak bir grafın birden fazla kapsama ağacı vardır.

Bir grafın kapsama ağacını bulmak için bir çok yöntem vardır. Bunlardan birisi kenarları döngüleri yok edecek şekilde kaldırmak olabilir. Şekil 12.5'te bir örnekle yöntem açıklanmıştır.



Şekil 12.5

Bağlı bir grafta n düğüm ve e kenar var ise (e ≥ n) kenar kaldırma işlemini e-n+1kez yapmamız gerekecektir. Zira n düğümlü ağaçta n-1 adet kenar vardır.

12.3 Breadth-First Search Algoritması (BFS)

(Genişlik – öncelikli Arama Algoritması)

Kapsama ağacını elde etmek için döngüleri kaldırma yönteminde, öncelikle döngüleri saptamak gerekmektedir. BFS algoritması bilgisayarda gerçeklemeye daha uygun bir algoritmadır. Bölüm III.den algoritmayı tekrar hatırlarsak , bir S düğümü ile başlayacağız. Daha sonra S düğümüne komşu düğümleri bulup 1 etiketini vereceğiz. Sonra 1 etiketliler komşu düğümlere 2 etiketini nihayet grafta etiketlenmemiş düğüm kalmayıncaya kadar işleme her seferinde etiket değerini bir arttırarak devam edeceğiz. Sonuç olarak k etiketine bizi götüren kenarlar kapsama ağacını oluşturacaktır.

BFS Kapsama ağacı algoritmas (BFSKA)

Algoritma n düğümlü bir graf için varsa kapsama ağacıda bulur. Algoritmada L etiketli düğümler kümesi ve T ise L deki düğümleri birleştiren kenarlar kümesini göstermektedir. Algoritma aşağıdaki adımlardan oluşur.

Adım1 (Başlangıç düğümünü seç)

- ***(a) U düğümü seç , etiketi 0 olsun***
- ***(b) L={U} ve T={Ø}yap***
- ***(c) k=0***
- ***(d) Adım 2 (düğümleri etiketle)***

While (|L| <n ve L deki en az bir düğüm L de olmayan bir düğüme komşu)

Adım 2.1 k=k+1

2.2 (T ye bir kenar ekle)

while (Ldeki k-1 etiketli bir V düğümüne , L de olmayan bir W düğümü komşu)

- ***(a) Wnin etiketini k yap***
- ***(b) W yi L ye ekle***
- ***(c) V ve W düğümleri arasındaki kenarı T ye ekle***

endwhile

endwhile

adım3 (çözüm varmı?)

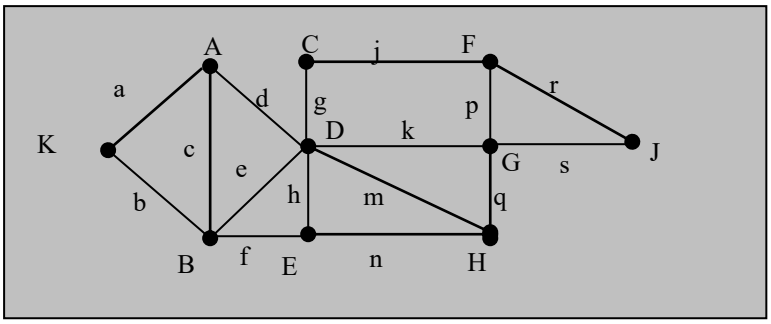
if |L| < n

graf bağlı değil ,buna göre kapsama ağaçı yok otherwise

T deki kenarlar ve bağlı düğümleri kapsama ağacını oluşturur.

Endif

Örnek 12.1



Şekil 12.6

Şekil 12.6 daki grafa BFSKA algoritmasını uygulayarak kapsama ağacını bulunuz.

Çözüm:

Adım1 K düğümü başlangıç düğümü olsun.

L={ K} T=Ø k= 0

Adım 2 while (|L|=1 >10 ve K düğümüne komşu düğümler var.

2.1 k=1

2.2 while K düğümüne A komşu

(a) A → 1

(b) L={ K,A}

(c) T={ a}

2.2 while K düğümüne B komşu

(a) B →1

(b) L={ K,A,B}

(c) T ={ a,b}

Adım 2 while (|L|=3<10 A,B düğümlerine komşu var) k=2

2.2 while A düğümüne D komşu D →2 ,L={ K,A,B,D},

T ={ a,b,d}

2.2 while B düğümüne E komşu E →2 ,L={ K,A,B,D,E},

T ={ a,b,d,f}

Adım 2 while (|L|=5<10 D,E düğümlerine komşu var) k=3

2.2 while D düğümüne C,G ve H komşu

L={ K,A,B,D,E,C,G,H}, T ={ a,b,d,f,g,k,m}

Adım 2 while (|L|=8<10 C ve G düğümlerine komşu var) k=4

2.2 while C düğümüne F komşu L={ K,A,B,D,E,C,G,H,F},

T ={ a,b,d,f,g,k,m,j}

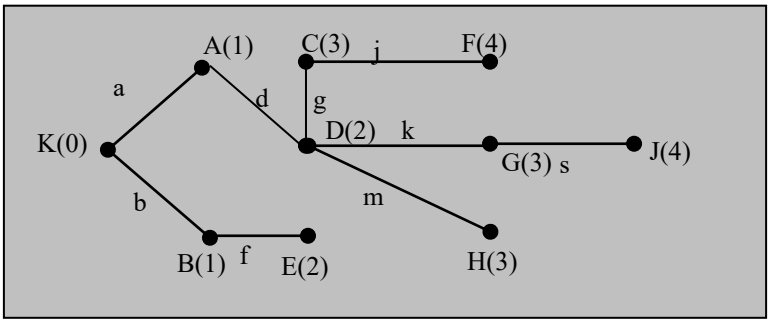
2.2 while B düğümüne E komşu E →2 ,

L={ K,A,B,D,E,C,G,H,F,J}, T ={ a,b,d,f,g,k,m,j,s}

Adım 2 |L|=10(endwhile)

Adım 2 |L|=10=n

T={a,b,d,f,g,k,m,j,s} kapsama ağacını oluşturur.



Şekil 12.7

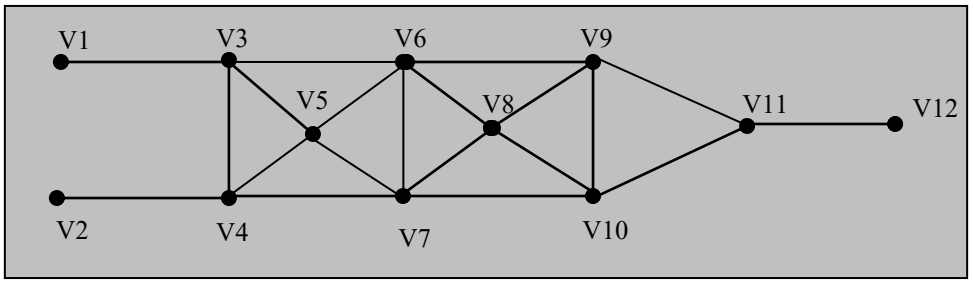
Kapsama ağacı şekil 12.7 deki gibi olacaktır.

Teorem

Bir graf , eğer ve yalnız eğer bir kapsama ağacı varsa bağlıdır.

Örnek 12.2

Bir G grafının komşuluk listesi aşağıda verilmiştir. BFSKA algoritmasını kullanarak kapsama ağacını bulunuz.



Şekil 12.8

V1=V3

V2=V4

V3=V1,V4,V5,V6

V4=V2,V3,V5,V7

V5=V3,V4,V6,V7

V6=V3,V5,V7,V8,V9

V7=V4,V5,V6,V8,V10

V8=V6,V7,V9,V10

V9=V6,V8,V10,V11

V10=V7,V8,V9,V11

V11=V9,V10,V12

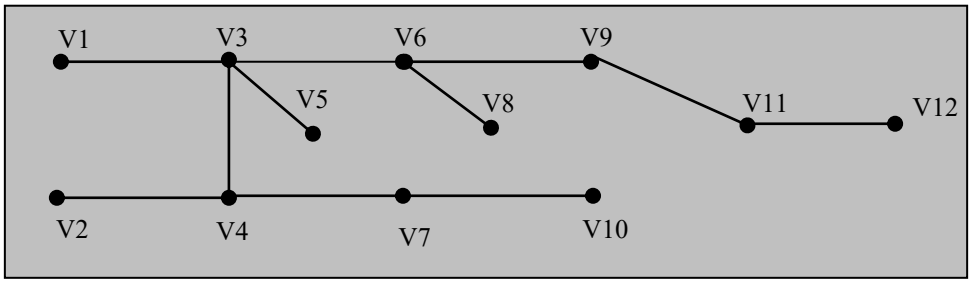
V12=V11

Başlangıç düğümü V1 olsun. k=0

k=	**1**	**2**	**3**	**4**	**5**
V1	0	X	X	X	X
V2	-	-	3	X	X
V3	1	X	X	X	X
V4	-	2	X	X	X
V5	-	2	X	X	X
V6	-	2	X	X	X
V7	-	-	3	X	X
V8	-	-	3	X	X
V9	-	-	3	X	X
V10	-	-	-	4	X
V11	-	-	-	4	X
V12	-	-	-	-	5

L={V1 | V3 | V4 V5 V6 | V2 V7 V8 V9 | V10 V11 | V12 }

T={ (V1 V3), (V3 V4), (V3 V5), (V3 V6), (V4 V2), (V4 V7), (V6 V8), (V6 V9), (V7 V10), (V9 V11),(V11 V12)}



Şekil 12.9

Örnek 12.3

Komşuluk listesi aşağıdaki gibi olan bir graf için kapsama ağacını bulunuz.

1 : 2,5

2 : 1,3

3 : 2,6

4 : 5,6

5 : 1,4

6 : 3,4

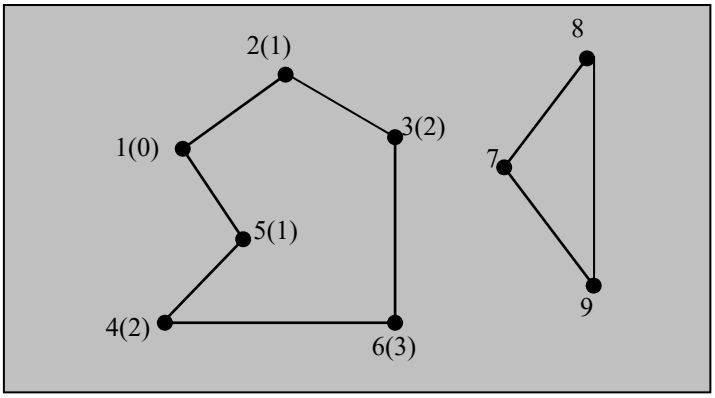
7 : 8,9

8 : 7,9

9 : 7,8

Çözüm:

Karşılık düşen graf şekil 4.10 da görülmektedir. Şekilde görüldüğü gibi graf bağlı olmadığı için kapsama ağacı yoktur.



Şekil 12.10

Aynı sonucu BFS algoritmasını kullanarak bulmaya çalışalım. L ={1} T ={0} k=0 olsun

	1	2	3
1	(0)	X	X
2	1	(X)	X
3	-	2	(X)
4	-	2	(X)
5	1	(X)	X
6	-	-	3
7	-	-	-
8	-	-	-
9	-	-	-

L={1 | 2,5 | 3,4 |6}

T= { (1,2),(1,5),(2,3),(5,4),(3,6)}

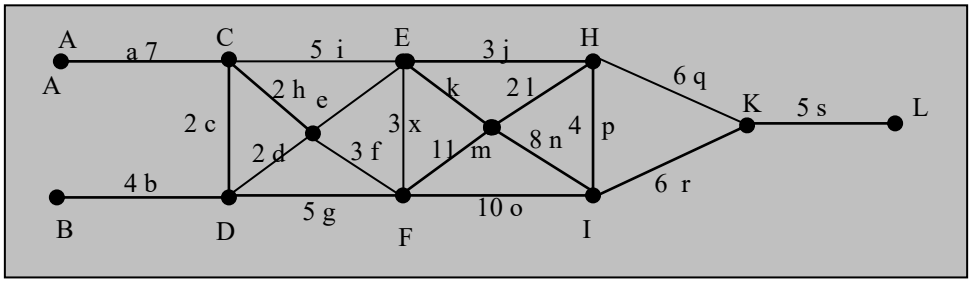
Algoritma 3.adımında |L| =6<9 olduğuna göre sonuç;

" graf bağlı değil, kapsama ağacı yok" biçimindedir.

12.4 Minimal ve Maksimal Kapsama Ağaçları

Şalt sahası örneğimizde, kapsama ağacını seçerken herhangi bir sınırlandırma kullanmamıştık. Ancak burada arazi durumu, mesafe gibi çeşitli sınırlandırmalar getirerek maliyeti en düşük olan kapsama

ağacının bulunmasını isteyebiliriz. Bu durumda problem ağırlıklı bir grafın minimum ağırlıklı olan kapsama ağacını bulmaya dönüşecektir.



Şekil 12.11

Ağırlıkların şekil 12.11' deki gibi verildiğini varsayalım. Şekil 12.11'de aijqsrnmfdb ağacını ele alırsak

W=7+5+3+6+5+6+8+11+3+2+4=60 ağırlıklıdır. Aynı şekilde ahicbkjqsgo ağacını dikkate aldığımızda W=7+2+5+2+4+10+3+6+5+5+10=59 elde edilir.

Minimal kapsama ağacını bulan bir algoritma Prim algoritmasıdır. Algoritmada bir düğüm seçilir ve ona bağlı minimum ağırlıklı kenar bulunur. Sonraki adımda buna bağlı minimum kenar seçilerek ağaç oluşturulur.

Prim algoritması

Algoritma varsa ağırlıklı bir grafın minimal kapsama ağacını bulur. Algoritmada T ağacı oluşturan kenarlar kümesi , L ise T deki kenarlara ilişkin düğümler kümesidir. Algoritma aşağıdaki adımlardan oluşur.

Adım 1 bir U düğümü seç L={ U } , T={Ø}

Adım 2 while (|L|< n ve L deki düğüm ile L de olmayan bir düğüm arasında enaz bir kenar var)

- ***(a) Bu kenarlar içinde ağırlığı enaz olan kenarı seç***
- ***(b) Kenarı T ye ekle***
- ***(c) Bu kenara ilişkin diğer düğümü L ye ekle***

Endwhile

Adım 3 if $|L|< n$

Graf bağlı değil , çözüm yok

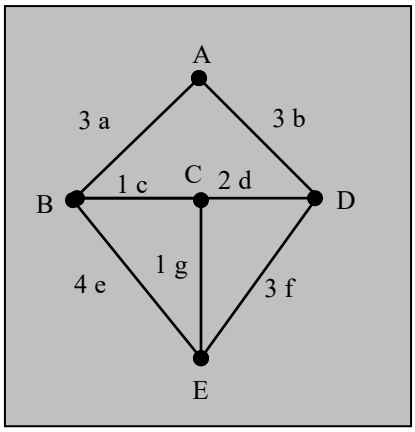
Otherwise

T' deki kenarlar minimal kapsama ağacını oluşturur

endif.

Örnek 12.4

Şekil 12. 12'deki graf için A düğümünden başlayarak minimal kapsama ağacını bulalım.



Şekil 12.12a

Adım 1

A düğümünü seçelim.

L={A}, T=∅ n=5

Adım 2

While

$|L|=1<5$ ve

1. A düğümüne a kenarı ve b kenarı bağlıdır. a'yı seçelim.

L={A,B},T={a} $|L|=2<5$

2. A ve B'ye b,e,c kenarları bağlıdır. c'yi seçelim.

L={A,B,C}, T={a,c} $|L|=3<5$

3. A,B,C'ye e,g,d,b kenarları bağlıdır. g'yi seçelim.

L={A,B,C,E}, T={a,c,g} $|L|=4<5$

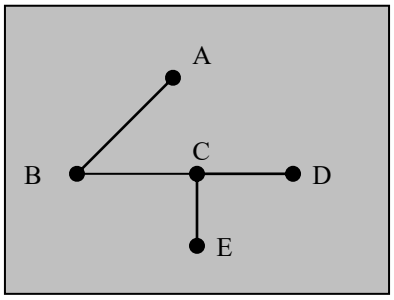
4. A,B,C,E'ye e, f,d,b kenarları bağlıdır. d'yi seçelim.

L={A,B,C,E,D}, T={a,c,g,d} $|L|=5$

L'deki elemanlara komşu yok.

End while

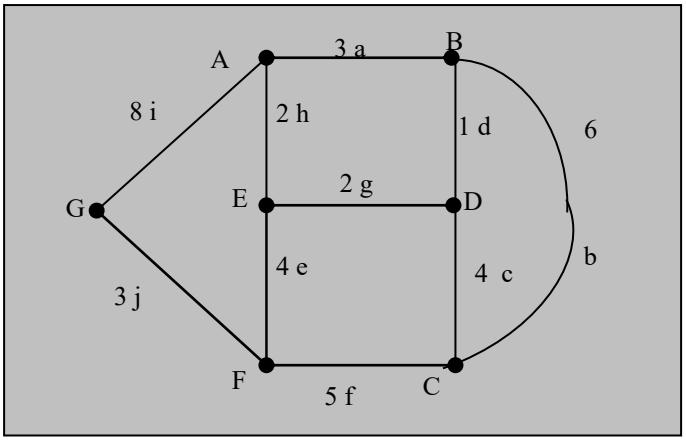
T'nin elemanları minimum kapsama ağacını oluşturur (Şekil 12.12b).



Şekil 12.12b

Örnek 12.5

Şekil 12.13'deki graf için Prim algoritmasını uygulayarak minimum kapsama ağacını bulunuz.



Şekil 12.13

Adım 1

A düğümü ile başlayalım.

L={A}, T=∅ n=7

Adım 2 while

1. $|L|$=1<7, A düğümüne i,h,a kenarları bağlıdır. Minimum ağırlığa sahip olan h seçilir.

L={A,E} T={h} $|L|$=2<7

2.A,E düğümlerine i,a,e,g kenarları bağlıdır. Minimum ağırlığa sahip olan g seçilir.

L={A,E,D} T={h,g} $|L|$=3<7

3.A,E,D düğümlerine i,a,e,d,c kenarları bağlıdır. Minimum ağırlığa sahip olan d seçilir.

L={A,E,D,B} T={h,g,d} $|L|$=4<7

4.A,E,D,B düğümlerine b,c,e,i kenarları bağlıdır. Minimum ağırlığa sahip olan e seçilir.

L={A,E,D,B,F} T={h,g,d,e} $|L|$=5<7

5.A,E,D,B,F düğümlerine i,j,f,c,b kenarları bağlıdır. Minimum ağırlığa sahip olan j seçilir.

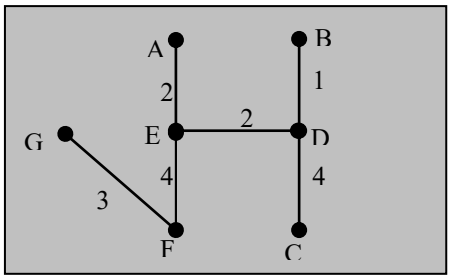
L={A,E,D,B,F,G} T={h,g,d,e,j} $|L|=6<7$

6.A,E,D,B,F,G düğümlerine f,c,b kenarları bağlıdır. Minimum ağırlığa sahip olan c seçilir.

L={A,E,D,B,F,G,C} T={h,g,d,e,j,c} $|L|=7$

Endwhile

Adım 3 Minimum kapsama ağacı hgdejc ağacıdır. Ağırlığı ise 16'dır (şekil 12.14)



Şekil 12.14

Kruskal Algoritması

Algoritma bir G grafı için varsa minimal kapsama ağacını bulur. n düğüm sayısı olmak üzere $n \geq 2$ dir. Algoritmada S ve T kümeleri G nin kenarlarının kümeleridir.

Adım 1 (koşullama)

(a) T={∅} (b) S={G nin tüm kenarları kümesi}

Adım 2 (T'yi genişlet)

.while ($|T|<n-1$ ve S boş küme değil)

(a) S den en küçük ağırlıklı kenarı (e) seç

(b) T'deki diğer kenarlar ile e kenarı çevrim yapmıyorsa T'ye e kenarını ekle

(c) e kenarını S den çıkar

endwhile

Adım 3 if $|T|<n-1$

G bağlı değil , minimal kapsama ağacı yok

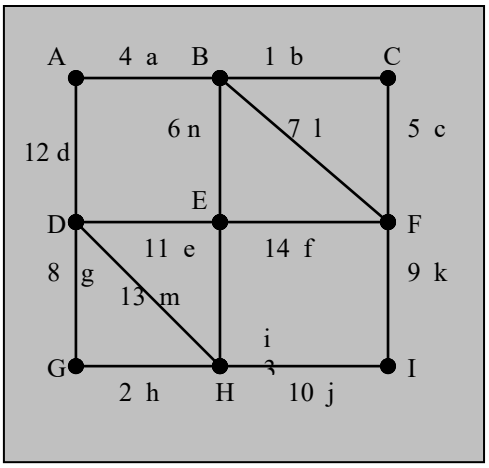
otherwise

T'nin kenarları ve bağlı düğümleri minimal kapsama ağacıdır

endif.

Örnek 12.6

Şekil 12.15'de verilen graf için Kruskal ve Prim algoritmalarını uygulayarak minimum kapsama ağaçlarını bulunuz.



Şekil 12.15

Kruskal algoritması

Adım 1 T={∅} n=9 S={a,b,c,d,e,f,g,h,i,j,k,l,m,n}

Adım 2 while |T|<n-1

1. a) en küçük b

   b) T={b} |T|=1<8

2. a) en küçük h

   b) T={b,h} |T|=2<8

3. a) en küçük i

   b) T={b,h,i} |T|=3<8

4. a) en küçük a

   b) T={b,h,i,a} |T|=4 <8

5. a) en küçük c

   b) T={b,h,i,a,c} |T|=5<8

6. a) en küçük n

   b) T={b,h,i,a,c,n} |T|=6<8

7. a) l ve f döngü oluşturuyor. g seçilir.

b) T={b,h,i,a,c,n,g} $|T|=7<8$

8. a) m,e,d döngü oluşturuyor. k seçilir.

b) T={b,h,i,a,c,n,g,k} $|T|=8$ bitti!

bhiacngk minimal kapsama ağacıdır ve ağırlığı 38'dir.

Prim algoritması

n=9

L={A} T={∅}

1. A'ya bağlı a,d kenarları var, en küçük ağırlıklı olan a'yı seçelim.

L={A,B} T={a} $|L|=2<9$

2. A,B'ye bağlı d,n,l,b kenarları var, en küçük ağırlıklı olan b'yi seçelim.

L={A,B,C} T={a,b} $|L|=3<9$

3. A,B,C'ye bağlı d,n,l,b,c kenarları var, en küçük ağılıklı olan c'yi seçelim.

L={A,B,C,F} T={a,b,c} $|L|=4<9$

4. A,B,C,F'ye bağlı d,n,f kenarları var, en küçük ağılıklı olan l'yi seçelim.

L={A,B,C,F,E} T={a,b,c,n} $|L|=5<9$

5. A,B,C,F,E'ye bağlı d,k,i,e kenarları var, en küçük ağılıklı olan i'yi seçelim.

L={A,B,C,F,E,H} T={a,b,c,n,i} $|L|=6<9$

6. A,B,C,F,E,H'ye bağlı d,k,e,h,j kenarları var, en küçük ağılıklı olan h'yi seçelim.

L={A,B,C,F,E,H,G} T={a,b,c,n,i,h} $|L|=7<9$

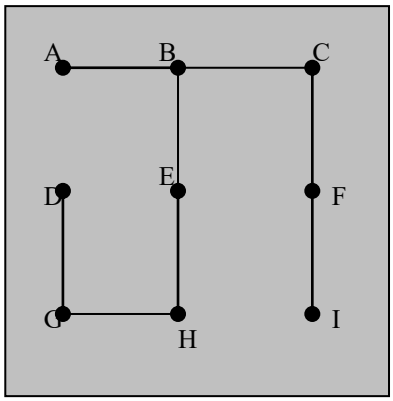
7. A,B,C,F,E,H,G'ye bağlı d,k,e,g,j kenarları var, en küçük ağılıklı olan g'yi seçelim.

L={A,B,C,F,E,H,G,D} T={a,b,c,n,i,h,g} $|L|=8<9$

8. A,B,C,F,E,H,G,D'ye bağlı j ve k kenarları var, en küçük ağılıklı olan k'yi seçelim.

L={A,B,C,F,E,H,G,D,I} T={a,b,c,n,i,h,g,k} $|L|=9$ bitti!

Sonuç olarak her iki algoritma ile aynı sonuç elde edilmiş oldu(şekil 12.16).

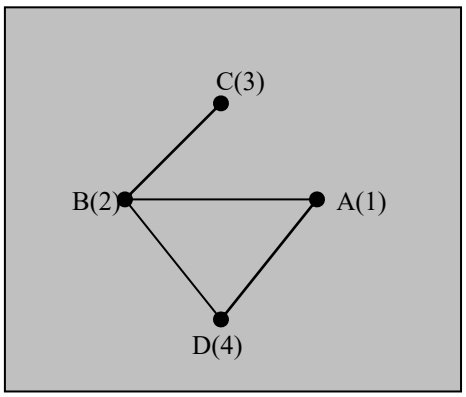


Şekil 12.16

12.5 Depth-Fırst Seach (Derinlik Öncelikli Arama)

Daha önce kapsama ağacının bulunmasında Genişlik Öncelik Arama (Breadth First Search) algoritmasının nasıl kullanıldığını görmüştük. Algoritma bir düğümden başlayarak komşu olan tüm düğümlere yayılmayı öngörüyordu. Bunların her birinden, onların komşularına, sonuçta tüm düğüme bitene kadar yayılma devam ediyordu. Bu yolla başlangıç düğümünden her bir düğüme mesafeleri ve kapsama ağacını elde ediyorduk.

Bir başka algoritma ise derinlik öncelikli arama algoritması olup, bu algoritmada düğümleri birbirini izleyen tamsayılarla etiketleyeceğiz. Buradaki temel fikir ise, bir V düğümünü etiketledikten sonra hemen ona komşu olanların içinden bir başka düğümü etiketlemektir. Eğer W düğümü V'ye komşu ise ona bir sonraki etiketi verip, hemen W düğümüne komşu bir düğümü aramaya geçeceğiz. Eğer V düğümünü etiketlenmemiş komşu düğümü yok ise, bu durum V düğümüne geldiğimiz yoldan geriye doğru adım adım ilerleyerek her geri gidişimizde düğüme komşu etiketlenmemiş bir U düğümü bulursak bu düğüme bir sonraki etiket vereceğiz. Arama işlemine U düğümünden başlayarak devam edeceğiz. Şekil 12.17'de algoritmanın uygulanması görülmektedir.



Şekil 12.17

Başlangıç düğümü A, etiketi 1 olacaktır. A düğümünün komşuları B ve D olup rasgele B yi seçelim. B nin etiketi 2 olacaktır. B nin komşuları C ve D , C yi seçelim. C nin başka komşusu yok , o halde geriye dönüp , B nin komşularına bakıyoruz. D düğümü etiketlenmemiş olduğu için bir sonraki, yani 4 etiketini alacaktır.

Derinlik- Öncelik Arama Algoritması

Algoritma , bir G grafı için eğer var ise kapsama ağacını bulur. Algoritmada L, etiketli düğümler kümesi, T sabit kenarlar kümesi ve Y düğümünün önceli, L de Y' yi etiketlemek için kullanılan düğümdür.

Algoritma aşağıdaki adımlardan oluşur.

Adım 1 (başlangıç düğümünü etiketle)

(a) Bir U düğümünü seç

(b) U ya 1 etiketini ver önceli yok olsun

(c) L={U} ve T=∅

(d) k=2 ve X=U

Adım 2 (diğer düğümleri etiketle)

Repeat

Adım 2.1 (X e komşu bir düğümü etiketle)

While (X e komşu, L' de olmayan bir Y düğümü var)

(a) {X,Y} kenarını T' ye ekle

(b) Y' ye k etiketini ver

(c) Y' nin öncelini X yap.

(d) Y yi L' ye ekle

(e) k=k+1

(f) X düğümü Y düğümünü işaret etsin.

Endwhile

Adım 2.2 (geri adım)

X yerine X' in öncelini yerleştir.

Until (G' deki tüm düğümler L' de ya da X=null)

Adım 3

İf (G' nin tüm düğümleri L' de)

T'deki kenarlar ve ilişkin düğümler kapsama ağacıdır

otherwise

G grafının kapsama ağacı yoktur, graf bağlı değildir

endif.

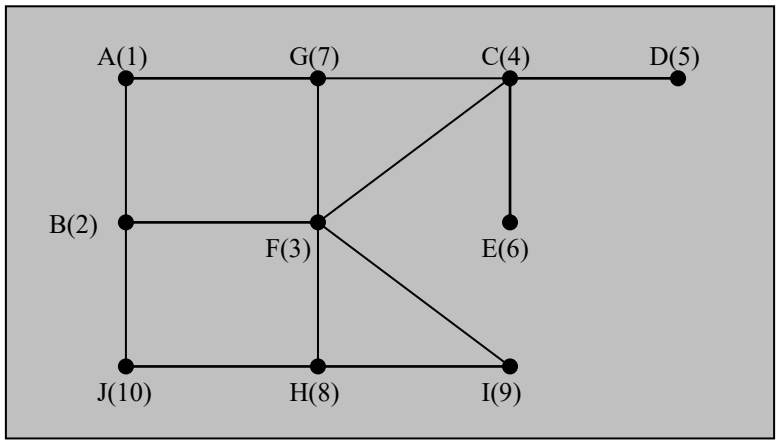
Derinlik- Öncelik Arama Algoritmasının Karmaşıklığı

Algoritmanın analizine bakarsak; n düğümlü e kenarlı bir graf için , her düğüm en fazla bir kez etiketlenmelidir. Her bir kenar ise en çok iki kere kullanılmaktadır. Bir kere etiketlenen düğümden, etiketlenmemiş düğüme, bir kerede öncelikle etiketlenen düğüme geri giderken kullanılmaktadır.

Buna göre en fazla $n+2e \leq n+2$ $c(n,2)=n+2.n(n-1)/2=n+ n^2-n=n^2$ elementer işlem yapılmaktadır.

Örnek 12.7

Şekil 12.18'deki G grafı için DFS algoritmasını uygulayalım.



Şekil 12.18

Adım 1 Başlangıç düğümü A, L={A},T=∅,k=2,X=A

Adım 2

Repeat

Adım 2.1 while

1.A'ya komşu B,G var.B'yi seçelim.

T={(A,B)} L={A,B} B→B(2), B←A, k=3, X⇒B

2.B'ya komşu F,J var.F'yi seçelim.

T={(A,B),(B,F)} L={A,B,F} F→F(3), F←B, k=4, X⇒F

3.F'ya komşu C,G,I,H var.C'yi seçelim.

T={(A,B),(B,F),(F,C)} L={A,B,F,C} C→C(4), C←F, k=5, X⇒C

4.C'ya komşu D,E var.D'yi seçelim.

T={(A,B),(B,F),(F,C),(C,D)} L={A,B,F,C,D} D→D(5), D←C, k=6, X⇒D

5.D'ye komşu yok.

Endwhile

Adım 2.2 X=C

Adım 2.1 while

1.C'ya komşu E,G var.E'yi seçelim.

T={(A,B),(B,F),(F,C),(C,D),(C,E)} L={A,B,F,C,D,E} E→E(6),E←C, k=7, X⇒E

2. E'ye komşu yok.

Adım 2.2 X=C

Adım 2.1 while

1.C'ya komşu G var.G'yi seçelim.

T={(A,B),(B,F),(F,C),(C,D),(C,E),(C,G)} L={A,B,F,C,D,E,G} G→G(7),G←C, k=8, X⇒G

2. G'ye komşu yok.

End While

Adım 2.2 X=C

Adım 2.1 while

C'ye komşu yok (end while)

Adım 2.2 X=F

Adım 2.1 while

1.F'ya komşu H,I var.H'yi seçelim.

T={(A,B),(B,F),(F,C),(C,D),(C,E),(C,G),(F,H)} L={A,B,F,C,D,E,G,H} H→H(8),H←F, k=9, X⇒H

2.H'ya komşu I,J var.I'yi seçelim.

T={(A,B),(B,F),(F,C),(C,D),(C,E),(C,G),(F,H),(H,I)} L={A,B,F,C,D,E,G,H,I} I→I(9),I←H, k=10, X⇒I

3. I'ye komşu yok (endwhile).

Adım 2.2 X=H

Adım 2.1 while

1.H'ya komşu J var.J'yi seçelim.

T={(A,B),(B,F),(F,C),(C,D),(C,E),(C,G),(F,H),(H,I),(H,J)} L={A,B,F,C,D,E,G,H,I,J} J→J(10),J←H, k=11,

X⇒J

2. J'ye komşu yok (endwhile).

Adım 2.2 X=H

Until {tüm düğümler L'de} bitti!

Adım 3 L'de tüm düğümler var

T={(A,B),(B,F),(F,C),(C,D),(C,E),(C,G),(F,H),(H,I),(H,J)} kapsama ağacıdır.

Teorem

DFS algoritması bir G grafına uygulansın.

T' deki kenarlar ve L'deki düğümler bir ağaç oluşturur.

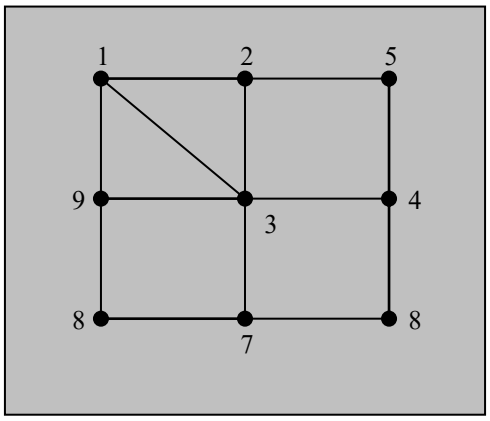
Eğer G bağlı ise, T ağacı bir kapsama ağacıdır.

DFS algoritması graflarla ilgili problemlerin çözümü için bir çok şekilde kullanılmaktadır. Bir kullanım örneği olarak, herhangi bir grafı, sıkı bağlı yönlü grafa çevirme problemini ele alalım. Hatırlayacağımız gibi grafta eğer köprü olan bir kenar yok ise grafın sıkı bağlı yönlü grafa dönüştürülebileceğini öngörmüştük. Ancak köprüyü belirlemek için bir yöntem önermemiştik. DFS algoritmasını bu amaçla kullanabiliriz. Şöyle ki DFS algoritmasını uygulayarak T kapsama ağacını bulalım. Eğer köprü olan kenar mor ise bu kenar T ağacı üzerinde olacaktır. O halde T ağacından bir kenarı seçip, graftan çıkarıp tekrar DFS algoritması uyguladığımızda, geriye kalan graf bağlı ise yani kapsama ağacı var ise bu kenar köprü değildir. Aksi halde köprüdür. T deki tüm kenarlar için bu sınamayı yaptığımızda hiçbiri köprü değilse bu grafı sıkı bağlı yönlendirebiliriz demektir.

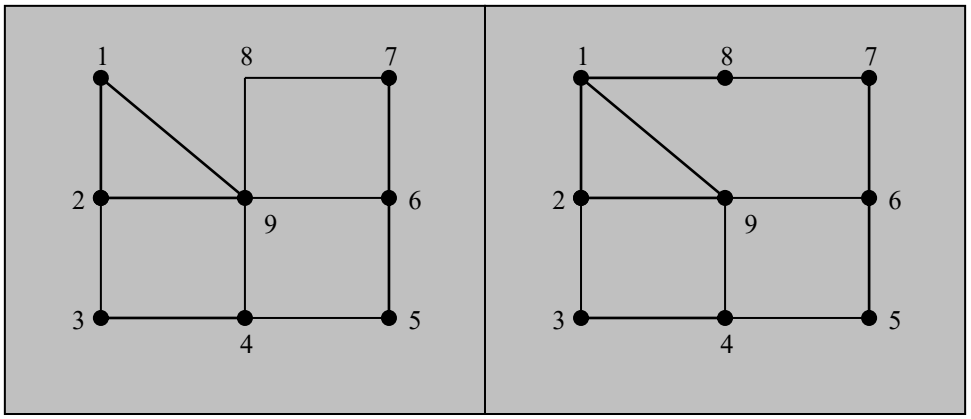
Şimdi de grafa normal yön vereceğimizi ele alalım. Ağaç üzerindeki kenarlara (ağaç kenarları—tree edges) küçük etiketten büyük etikete doğru , geri kenarlara (back edges) da büyük etiketten küçüğe doğru yön verirsek sıkı bağlı yönlü grafı elde ederiz.

Örnek 12.8

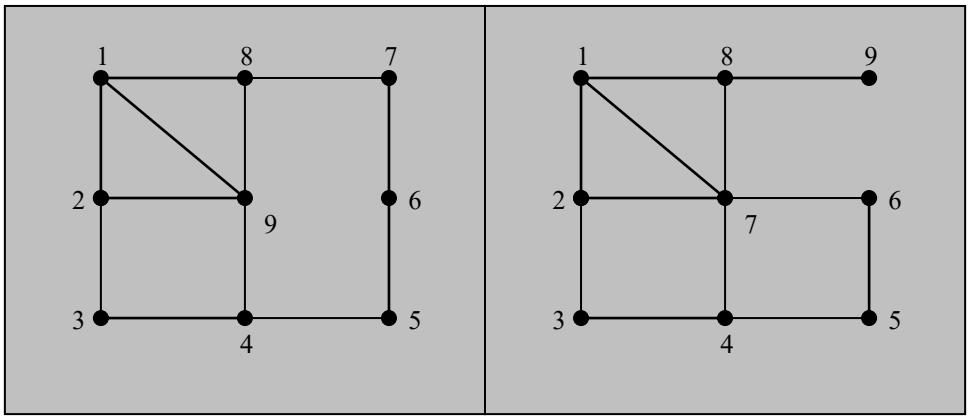
Şekil 12.19'daki graf, sıkı bağlı yönlü graf yapılabilir mi? Nasıl?



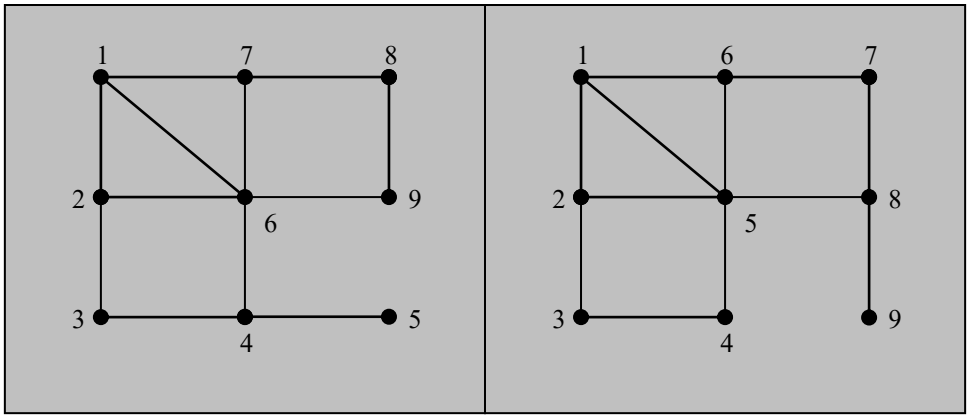
Şekil 12.19



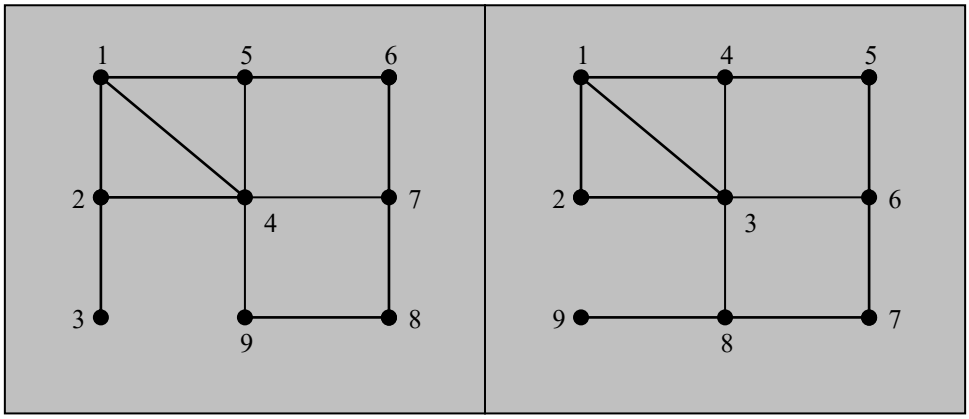
(1) (2)



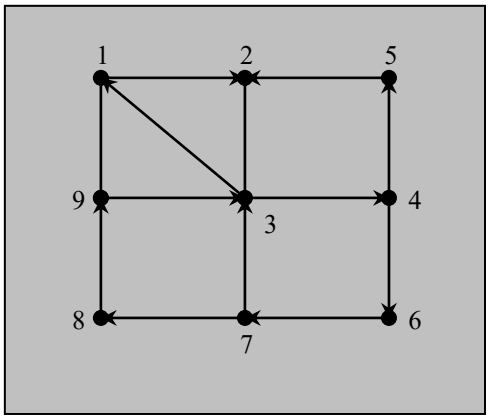
(3) (4)



(5) (6)



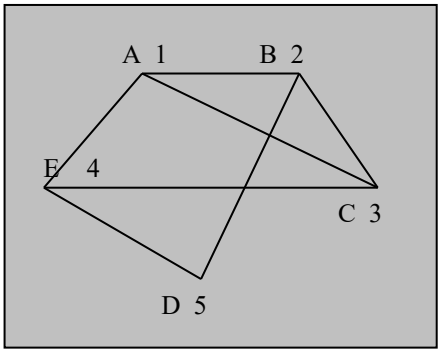
(7) (8)



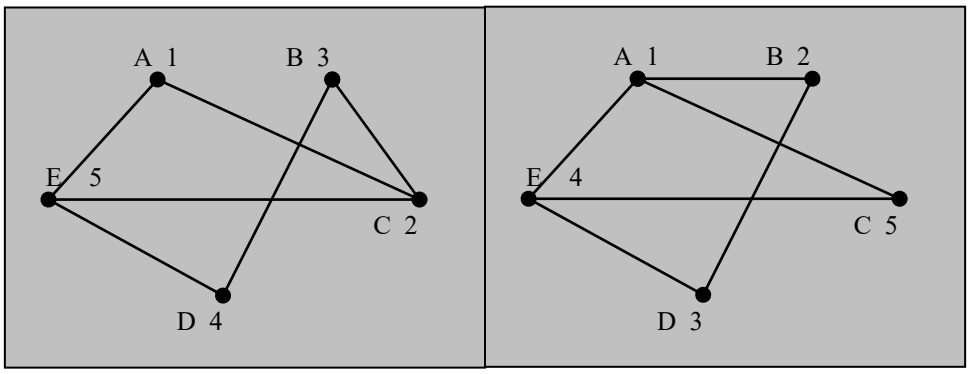
Sıkı bağlı yönlü graf

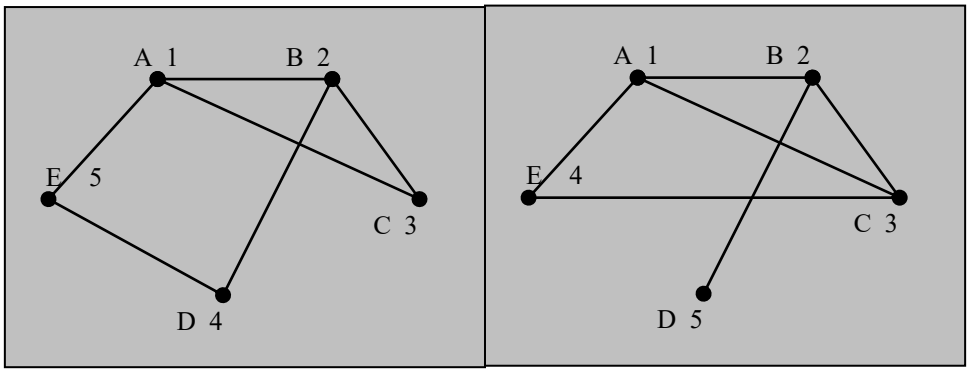
Örnek 12.9

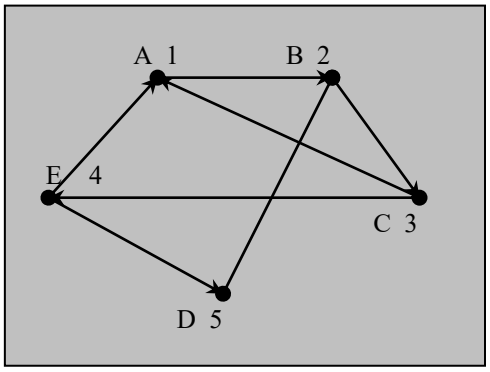
Şekil 12.20'deki grafın sıkı bağlı olup olmadığını inceleyelim.


A 1
B 2
E 4
C 3
D 5

Şekil 12.20

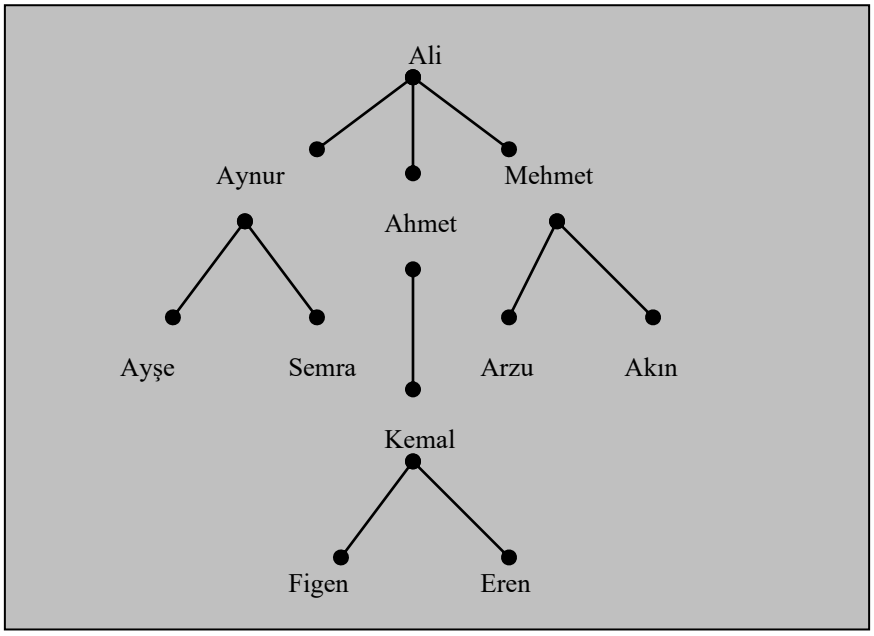

A 1
B 3
E 5
C 2
D 4
A 1
B 2
E 4
C 5
D 3


A 1
B 2
E 5
C 3
D 4
A 1
B 2
E 4
C 3
D 5


A 1
B 2
E 4
C 3
D 5

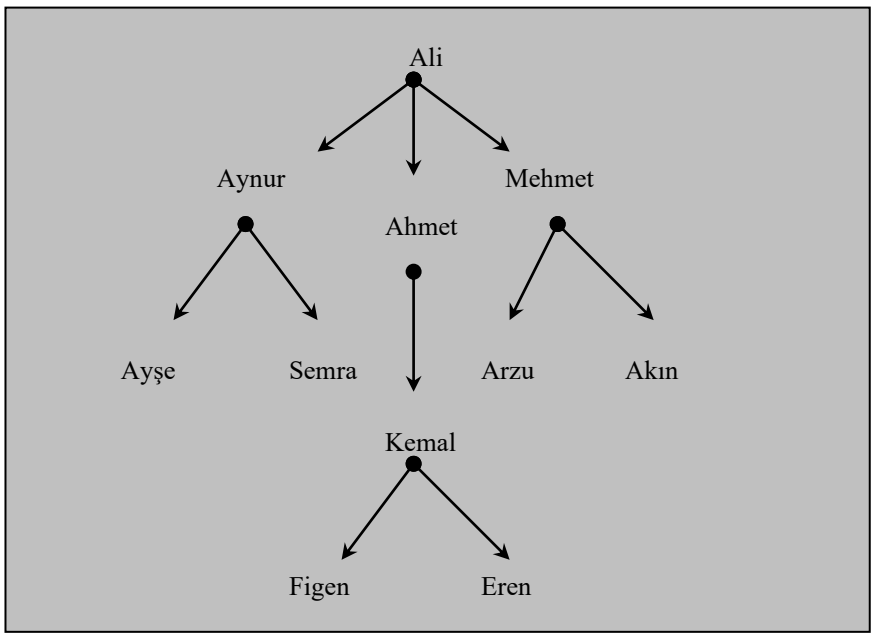
12.6 Köklü Ağaçlar (Rooted Trees)

Şekil 12.21 de bir soy ağacı görülmektedir .Yukarıdan aşağıya doğru izlendiğinde .... nin ebebeynidir. (Ali, Mehmet in ebebeyni) biçiminde bir ilişki kurulmuştur. Bu soyağacı aynı zamanda bu ilişkiyi gösterecek bir yönlü graf olarak da gösterilebilir. (Şekil 12.22)



Şekil 12.21

Tüm ok yönleri aşağıya doğru olduğuna göre, ne olduğu bilindikten sonra, okları çizmek gerekmeyebilir. Şekil 12.22 deki yönlü grafa bakarsak, bu grafta giriş derecesi 0 olan bir düğüm vardır ve diğer tüm düğümlerin giriş dereceleri ise 1 dir. Buna ek olarak eğer yönleri göz önüne almazsak elimizde bir ağaç vardır.



Şekil 12.22

Buna göre bir T köklü ağacı iki özelliği sağlar;

(1) Yönler göz önüne alınmadığı zaman, sonuçta elde edilen yönsüz graf bir ağaçtır.

(2) Giriş derecesi sıfır olan bir tek R(kök) düğümü vardır ve diğer tüm düğümlerin giriş dereceleri 1 dir.

Şekil 12.21'de Ali köktür. Genel olarak köklü ağaçta kök yukarıda olmak üzere aşağı doğru çizilerek yönler belirtilmez.

Teorem

Bir köklü ağaçta;

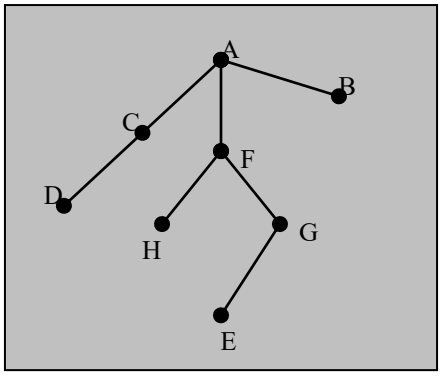
(a) **Düğüm sayısı, yönlü kenar sayısından bir fazladır.**

(b) **Ağaçta yönlü döngü yoktur.**

(c) **Kökten, diğer bütün düğümlere sadece bir tek yönlü basit yol vardır**

Bir köklü ağaçta, Bir U düğümünden V düğümüne doğru bir kenar var ise, U, V nin ebeveyni (parent), ya da V, U nun çocuğu (child) dur denir. Bir V düğümüne giden yönlü yolun üzerindeki diğer düğümlere V nin ataları(descendant) denir. Bir uç düğüm (terminal) çocuğu olmayan düğümdür. İç düğüm ise çocukları olan düğümdür.

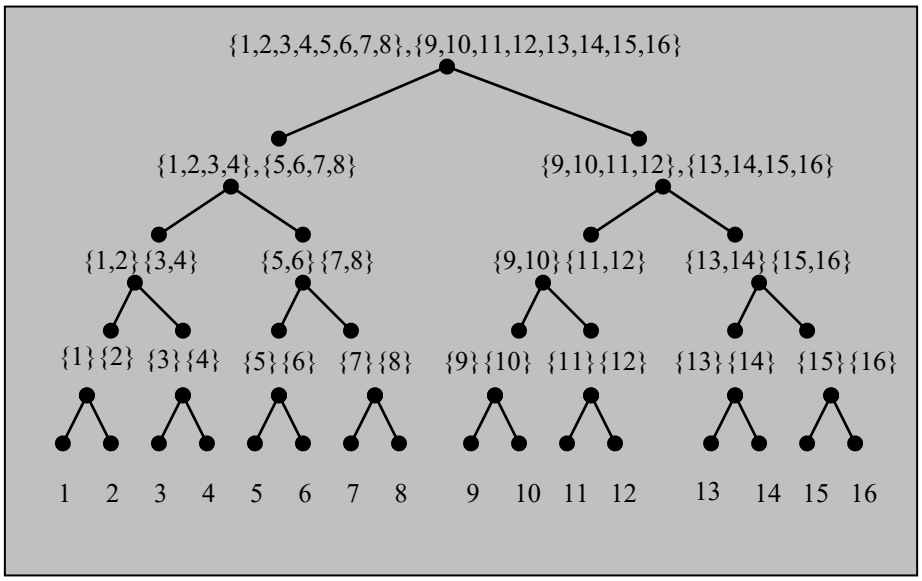
Şekil 12.23'te A kök; F, B ve C nin ebeveyni, B,D,H,E uç, C,F,G iç düğümlerdir. A,F,G düğümleri E nin atalarıdır. H düğümü A ve (F nin) torunudur.



Şekil 12.23

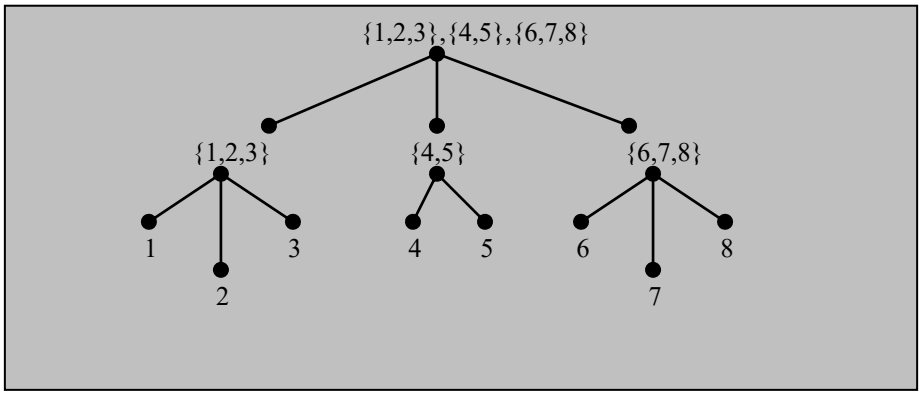
Örnek 12.10

16 adet metal paradan 15 tanesi aynı bir tanesi daha ağırdır. İki gözlü bir terazi ile ağır olanı an az tartımla bulmak istiyoruz. Paraları 1,2,.....16 ile etiketleyelim. Paraları ikiye bölüp sol ve sağ kefelerde tarttığımızı düşünelim. Ağır gelen tarafı yine ikiye bölüp tartarsak Şekil 12.24'te gösterilen köklü ağaçtan da anlaşılacağı gibi 3 tartı sonucunda ağır olan para bulunmaktadır.



Şekil 12.24

Bu tür ağaçlara genel olarak karar ağaçları (decision trees) denir. Şekil 12.25'te bir başka örnek karar ağacı gösterilmiştir. Burada her düğümün üç çocuğu olabilmektedir. Bu durumda iki denemede sonucu bulmak olasıdır.

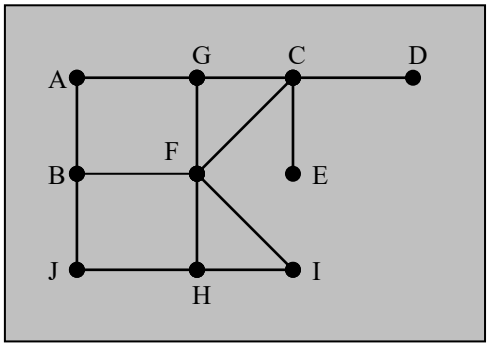


Şekil 4.25

Teorem

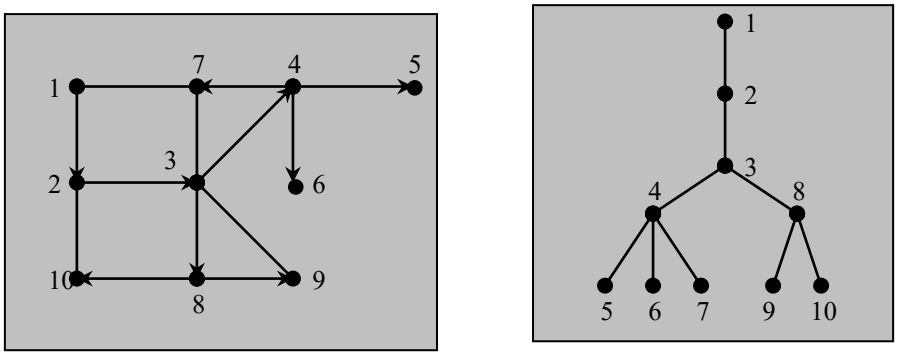
Bir G grafına DFS algoritması uygulanırsa, elde edilen T ağacı düşük numaradan yüksek numaraya doğru düzenlendiğinde 1 numaralı düğüm kök olmak üzere köklü ağaç oluşur.

Örnek 12.11



Şekil 12.26

Şekil 12.26'daki grafa DFS algoritması uygulanıp gerekli düzenleme yapılırsa, Şekil 12.27b'deki köklü ağaç elde edilecektir.

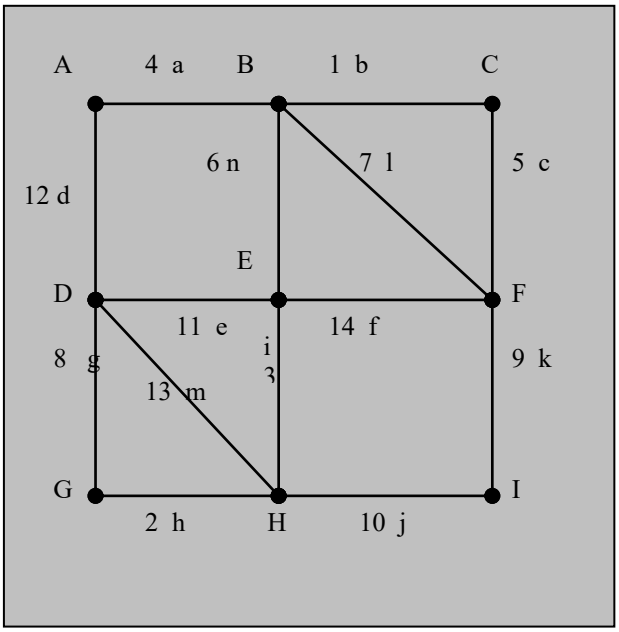


(a) (b)

Şekil 12.27

Ödev

Şekilde verilen graf için Kruskal ve Prim algoritmalarını uygulayarak minimum kapsama ağaçlarını bulunuz.



Kaynaklar

F.Selçuk,N.Yurtay,N.Yumuşak,Ayrık İşlemsel Yapılar, Sakarya Kitabevi,2005.
İ.Kara, Olasılık, Bilim Teknik Yayınevi, Eskişehir, 2000.
"Soyut Matematik", S.Aktaş,H.Hacısalihoğlu,Z.Özel,A.Sabuncuoğlu, Gazi Ünv.Yayınları,1984,Ankara.
"Applied Combinatorics", Alan Tucker, John Wiley&Sons Inc, 1994.
"Applications of Discrete Mathematics", John G. Michaels, Kenneth H. Rosen, McGraw-Hill International Edition, 1991.
"Discrete Mathematics", Paul F. Dierker and William L.Voxman, Harcourt Brace Jovanovich International Edition, 1986.
"Discrete Mathematic and Its Applications", Kenneth H. Rosen, McGraw-Hill International Editions, 5th Edition, 1999.
"Discrete Mathematics", Richard Johnson Baugh, Prentice Hall, Fifth Edition, 2001.
"Discrete Mathematics with Graph Theory" , Edgar G. Goodaire, Michael M. Parmenter, Prentice Hall, 2nd Edition, 2001.
"Discrete Mathematics Using a Computer", Cordelia Hall and John O'Donnell, Springer, 2000.
"Discrete Mathematics with Combinatorics", James A. Anderson, Prentice Hall, 2000.
"Discrete and Combinatorial Mathematics", Ralph P. Grimaldi, Addison-Wesley, 1998.
"Discrete Mathematics", John A. Dossey, Albert D. Otto, Lawrence E. Spence, C. Vanden Eynden, Pearson Addison Wesley; 4th edition 2001.
"Essence of Discrete Mathematics", Neville Dean, Prentice Hall PTR, 1st Edition, 1996.
"Mathematics:A Discrete Introduction", Edvard R. Schneiderman, Brooks Cole; 1st edition, 2000.
"Mathematics for Computer Science", A.Arnold and I.Guessarian, Prentice Hall, 1996.
"Theory and Problems of Discrete Mathematics", Seymour Lipschuts, Marc. L. Lipson, Shaum's Outline Series, McGraw-Hill Book Company, 1997.
"2000 Solved Problems in Discrete Mathematics", Seymour Lipschuts, McGraw- Hill Trade, 1991.