

BÖLÜM 11

ÇİZGE ALGORİTMALARI

ANAHTAR KAVRAMLAR

Minimum Kapsayan Ağaç, En Kısa Yol



ÖĞRENME HEDEFLERİ

-  1 Prim algoritmasını anlayacak ve uygulayabilecek
-  2 Kruskal algoritmasını anlayacak ve uygulayabilecek
-  3 Dijkstra algoritmasını anlayacak ve uygulayabilecek

GİRİŞ

Çizge algoritmaları, çizge veri yapısı ile modellenebilen birçok problemi çözmek için kullanılır. Bu algoritmalar, ağ yönlendirmesinden zamanlamaya kadar bilgisayar bilimindeki çeşitli sorunları modellemek ve çözmek için kullanılır. En sık kullanılan çizge algoritmalarından ikisi, bir çizge için minimum kapsayan ağacı bulma problemi için geliştirilmiş Prim ve Kruskal algoritmalarıdır. Algoritmalar ağırlıklı yönlü ve yönsüz çizgelerle kullanılmak üzere tasarlanmış ve bir çizgedeki tüm düğümleri en az maliyetle birbirlerine bağlamanın en verimli yolunu bulmak için kullanılmaktadırlar. Prim ve Kruskal algoritmaları, ağ tasarımı, kümeleme ve veri madenciliği gibi çeşitli uygulamalarda kullanılmıştır. Bunlara ek olarak, bir çizgedeki iki düğüm arasındaki en kısa yolu bulmak için Dijkstra en kısa yol algoritması tercih edilir. Bu algoritma, ağ yönlendirme, ağ optimizasyonu ve iki nokta arasındaki en kısa mesafenin gerekli olduğu diğer uygulamalarda da kullanılmaktadır. Ayrıca Dijkstra algoritması, iki şehir arasındaki en kısa yolu bulmaktan bir paket teslimat hizmeti için en verimli rotayı bulmaya kadar çok çeşitli problemleri çözmek için kullanılmıştır.



1. MİNİMUM KAPSAYAN AĞAÇ

Minimum Kapsayan Ağaç problemi, verilen bir çizgedeki tüm düğümleri minimum toplam kenar ağırlığıyla birleştiren kenarların alt kümesini

bulmak şeklinde tanımlanmaktadır. Bu problemin çözümü için yaygın olarak iki algoritma kullanılmaktadır: Prim ve Kruskal algoritmaları.

1.1. Prim Algoritması

Minimum kapsayan ağacı bulmak için en sık kullanılan algoritma Prim algoritmasıdır. Prim algoritması, çizgede bir düğüm noktasını başlangıç noktası seçerek çalışmaktadır. Sonrasında algoritma başlangıç düğümünün komşularından

başlayarak sırasıyla bütün düğümleri kapsayan ağaca ekleyebileceği en az ağırlıklı kenarları kapsayan ağaca ekleyerek sonuçta döneceği minimum kapsayan ağacı oluşturmaktadır.

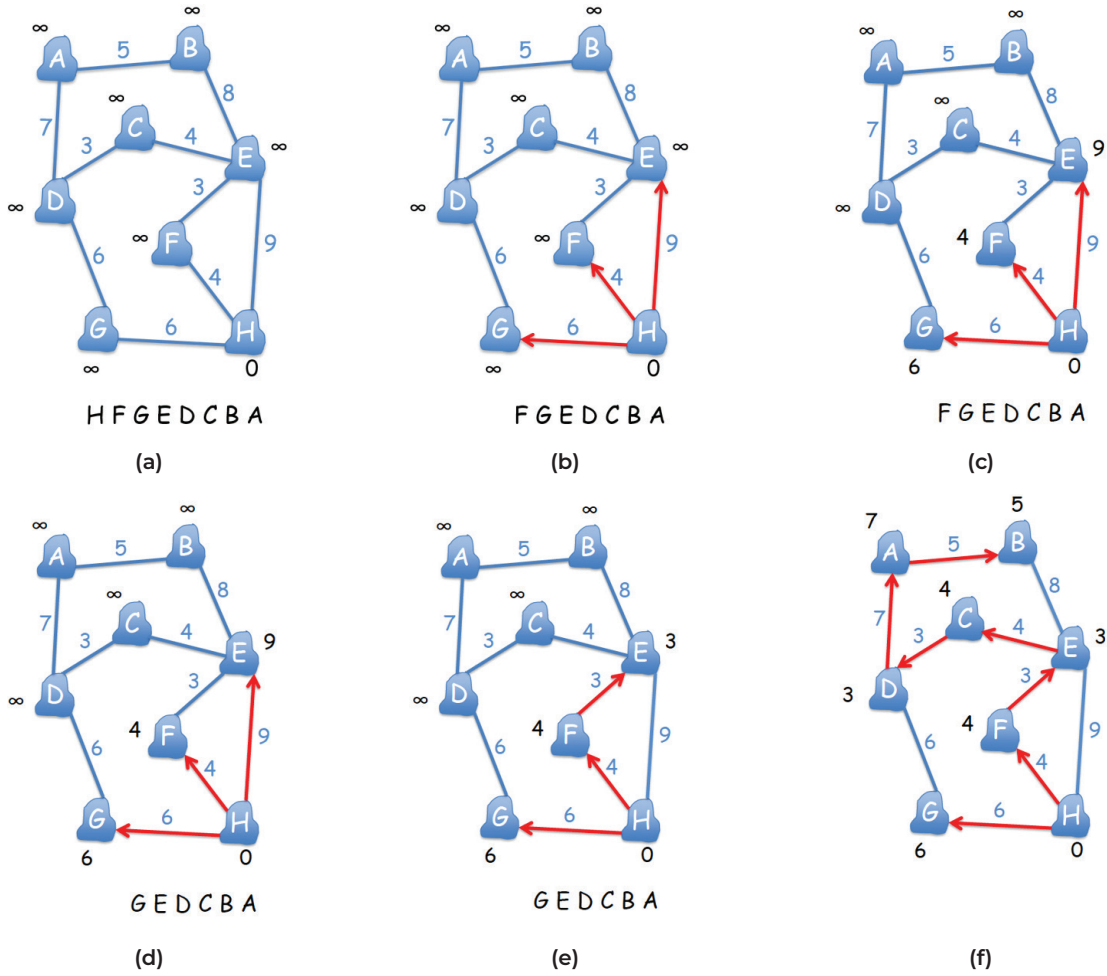
```

MST-PRIM (G, w, r)
for each vertex  $u \in G.V$ 
     $u.key = \infty$ 
     $u.par = NIL$ 
 $r.key = 0$ 
 $Q = \emptyset$ 
for each vertex  $u \in G.V$ 
    INSERT (Q, u)
while  $Q \neq \emptyset$ 
     $u = \text{EXTRACT-MIN}(Q)$ 
    for each vertex  $v$  in  $G.Adj[u]$ 
        if  $v \in Q$  and  $w(u, v) < v.key$ 
             $v.par = u$ 
             $v.key = w(u, v)$ 
    DECREASE-KEY (Q, v,  $w(u, v)$ )
  
```

Şekil 11.1 Prim algoritmasına ait sözde kod

Prim algoritması bir çizge, ağırlık matrisi ve başlangıç düğümünü girdi olarak almaktadır. Algoritma ilk olarak çizgedeki bütün düğümlerin anahtar değerlerini sonsuz ve ebeveyni null olarak belirler. Daha sonra başlangıç düğümünün anahtar değerine 0 atar. Algoritma boş minimum öncelikli bir Q kuyruğu oluşturur ve bütün düğümleri kuyruğa anahtar değerlerine göre yerleştirir. Sonrasında algoritma, bir döngü içerisinde Q kuyruğu boş olmadığı sürece en küçük anahtar değerine sahip düğümü (bu düğümü u diyelim) kuyruktan çıkarır, ve u düğümünün

komşularının ebeveyn ve anahtar parametrelerini eğer komşular hala kuyruktaysa ve komşu ile u düğümünü birleştiren kenarın ağırlığı komşunun anahtar değerinden küçükse yeniler. Bu şekilde algoritma esnasında, herhangi bir düğümü kapsayan ağaca bağlayabilecek daha az ağırlıklı bir kenar keşfedilmişse, ilgili düğümün yeni keşfedilen kenar üzerinden ağaca eklenmesi; ve algoritma çalıştırılması esnasında olası bütün kenarları taradığından çizgenin herbir düğümünün en az ağırlıklı kenar üzerinden kapsayan ağaca bağlanması sağlanmaktadır.



Şekil 11.2 üzerinden Prim algoritmasının verilen bir çizge için minimum kapsayan ağacı nasıl oluşturduğu adım adım gösterilecektir:

Şekil 11.2 Prim algoritmasının örnek bir çizge üzerinde çalışma adımları

(a) Algoritma ilk olarak bütün düğümlerin anahtar değerlerine sonsuz değeri atar ve başlangıç düğümünün anahtar değerini 0 olarak günceller.

(b) Anahtar değeri en küçük olan düğümü (H düğümü) Q kuyruğundan çıkartır ve bu düğümün komşuları E, F ve G düğümlerinin hepsi hala Q kuyruğunda olduğundan ve bu komşu düğümlerin anahtar değerleri onları H düğümü ile birleştiren kenarın ağırlığından küçük olduğundan, bu komşu düğümlerin ebeveyn parametresini H düğümü olarak günceller.

(c) Ebeveyn parametresini H düğümü olarak güncellediği E, F ve G düğümlerinin anahtar değerlerini sırasıyla 9, 4 ve 6 olarak günceller ve Q kuyruğunu en küçük anahtar değere sahip düğüm en ulaşılabilir konumda olacak şekilde günceller.

(d) Kalan elemanlardan en küçük anahtar değerine sahip düğümü (F düğümü) Q kuyruğundan çıkartır.

(e) F düğümünün komşularından E düğümü hala Q kuyruğunda olduğundan ve F'yi E ile birleştiren kenarın ağırlığı E'nin anahtarının güncel değerinden küçük olduğundan, E düğümünün ebeveyn parametresini F düğümü olarak ve E'nin anahtar değerini 3 olarak günceller. Burada da E düğümünde gözlemlendiği üzere, algoritma E düğümünü finalde oluşturacağı minimum kapsayan ağaca bağlayacak daha az ağırlıklı bir kenar keşfettiğinde ilgili düğümü yeni keşfettiği kenar üzerinden ağaca bağlamaktadır.

(f) Algoritma benzer adımları diğer düğümlere de uygulayarak finalde minimum kapsayan

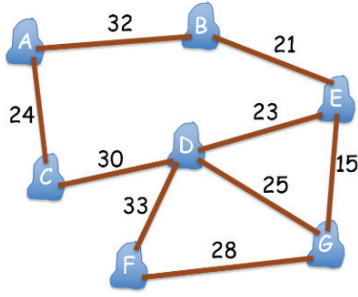
ağacı oluşturmaktadır. Şekilde kırmızı ile gösterilen düğümlerin ebeveyn parametreleri, ilgili

düğümün hangi kenar yoluyla minimum kapsayan ağaca bağlanacağını belirtmektedir.

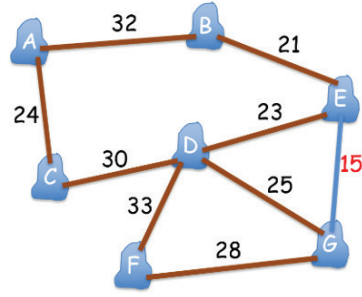
1.2. Kruskal Algoritması

Kruskal algoritması, minimum kapsayan ağacı bulmak için tercih edilen başka bir algoritmadır. Bu algoritma, çizgedeki kenarları ağırlığa göre sıralayarak ve ardından bunları birer birer kapsayan ağaca ekleyerek çalışır. Kapsayan ağaca

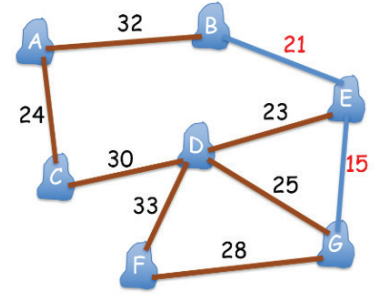
bir kenar eklenmeden önce, ilgili kenarın ağaca eklenmesi ile döngü olup oluşmadığı kontrol edilmelidir. Döngü oluşuyorsa, ilgili kenar ağaca eklenmez ve bir sonraki en az ağırlıklı kenar ile ağaç oluşum süreci devam ettirilir.



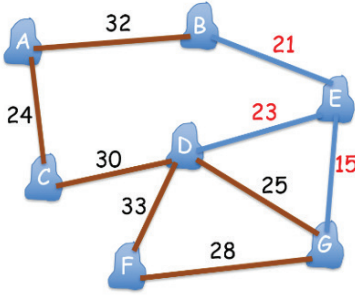
(a)



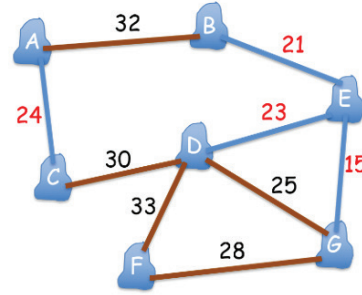
(b)



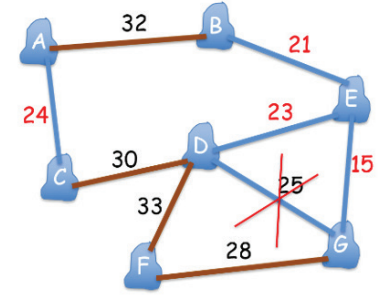
(c)



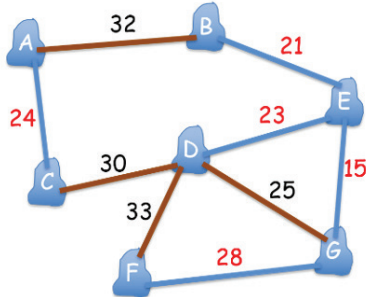
(d)



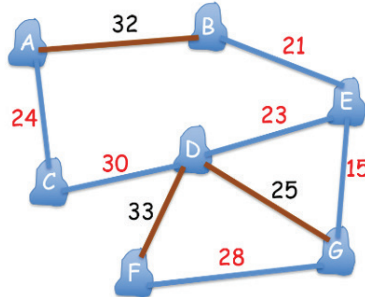
(e)



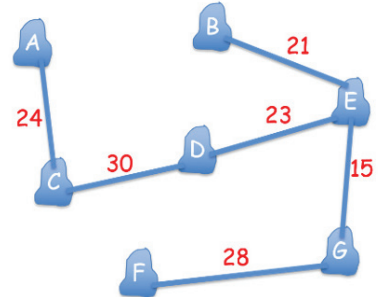
(f)



(g)



(h)



(i)

Şekil 11.3 Kruskal algoritmasının örnek bir çizge üzerinde çalışma adımları

Şekil 11.3 üzerinden Kruskal algoritmasının verilen bir çizge için minimum kapsayan ağacı nasıl oluşturduğu adım adım gösterilecektir:

(a) Düşümler ve kenar ağırlıkları örnek çizge üzerinde gösterilmiştir.

(b) En küçük ağırlığa sahip kenar (E, G) olduğu için ağaca eklenir.

(c) İkinci en küçük ağırlık değeri 21 olduğu için (B, E) kenarı döngü oluşturmadığı kontrol edilerek ağaca eklenir.

(d) (D, E) kenarı ağaca eklenir.

(e) (A, C) kenarı kalan kenarlar içerisinde en küçük olduğu ve döngü oluşturmadığı için ağaca eklenir.

(f) (D, G) kenarı kalan kenarlar içerisinde en küçük ağırlığa sahip olmasına rağmen eklenmesi durumunda döngü oluşturacağı için ağaca eklenmeden sıradaki kenar ile devam edilir.

(g) (F, G) kenarı kalan kenarlar içerisinde en küçük ağırlığa sahip olduğu için ve döngü oluşturmadığı için ağaca eklenir.

(h) En son olarak (C, D) kenarı eklenir. Dikkat edilirse n düğümlü bir çizge için; $n - 1$ döngü oluşturmayan en az ağırlıklı kenar, minimum kapsayan ağaç oluşturmak için yeterli olacaktır.

(i) Kruskal algoritmasının oluşturduğu minimum kapsayan ağaç.

2. EN KISA YOL

Tek kaynaklı en kısa yol algoritmaları, belirli bir kaynak düğüm ile bir çizgedeki diğer tüm düğümler arasındaki en kısa yolu bulmak için kullanılır. Bu algoritmalar, harita üzerindeki iki konum arasındaki en verimli rotayı bulmak, bir

bilgisayar ağındaki trafiği yönlendirmek veya diğer birçok optimizasyon problemlerini çözmek için tercih edilir. En popüler tek kaynaklı en kısa yol algoritmaları Dijkstra, Bellman-Ford ve Floyd-Warshall'dır.

2.1. Dijkstra Algoritması

Dijkstra algoritması, belirli bir başlangıç düğümünden mesafe değerlerine göre çizgenin diğer tüm düğümlerine giden en kısa yolları bulur. İlk olarak, başlangıçtan kendisine en yakın düğümüne, ardından en yakın ikinci düğüme giden en kısa yolu bulur. Tüm düğümler ziyaret edilene

kadar başlangıç düğümünden çizgedeki diğer tüm düğümlere giden en kısa yolları art arda hesaplayarak çalışır. Bu algoritma ancak çizgedeki tüm kenar negatif olmayan ağırlıklara sahip olduğunda doğru sonuç üretebilmektedir.

Dijkstra(G,s)

```

for each u of V
    u.dis =  $\infty$ 
    u.par = nil
s.dis = 0
initialize an empty set S
create a minimum priority Q on V
while Q  $\neq$  { }
    u = ExtractMin(Q)
    S = S  $\cup$  {u}
    for each v of Adj(u)
        if v.dis > u.dis + w(u,v)
            v.dis = u.dis + w(u,v)
            v.par = u
    update Q

```

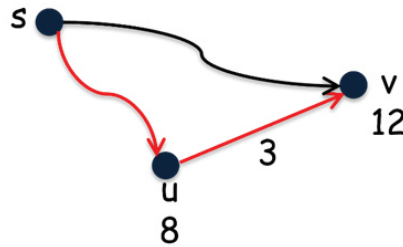
Şekil 11.4 Dijkstra algoritmasına ait sözde kod

Sözde kod ile gösterimi Şekil 11.4'te verilen Dijkstra algoritması bir çizgeyi ve başlangıç düğümünü parametre olarak alır. Algoritma öncelikli olarak, çizgenin bütün düğümlerinin mesafe ve ebeveyn parametrelerine başlangıç değerlerini atar. Burada mesafe değerinin başlangıç değeri sonsuz ve ebeveyn parametresinin başlangıç değeri olarak null'dur. Başlangıç düğümünün mesafe değeri 0 olarak güncellenir ve algoritmanın bu düğümden başlaması sağlanır. S boş kümesi

ve bütün düğümlerin mesafe değerlerine bağlı olmak üzere minimum öncelikli bir Q kuyruğu oluşturulur. Bir döngü içerisinde Q kuyruğu boş olmadığı sürece kuyruktaki en küçük mesafe değerine sahip düğüm çıkarılır, S kümesine eklenir ve her bir komşusu için gevşetme işlemi uygulanır. Gevşetme işleminin amacı, başlangıç düğümünden diğer düğümlere daha kısa uzunluklu bir yol keşfedildiğinde, ilgili düğüme giden en kısa yolu keşfedilen yola göre güncellemektir.

Relax(u, v)

```
if v.dis > u.dis + w(u,v)
    v.dis = u.dis + w(u,v)
    v.par = u
```

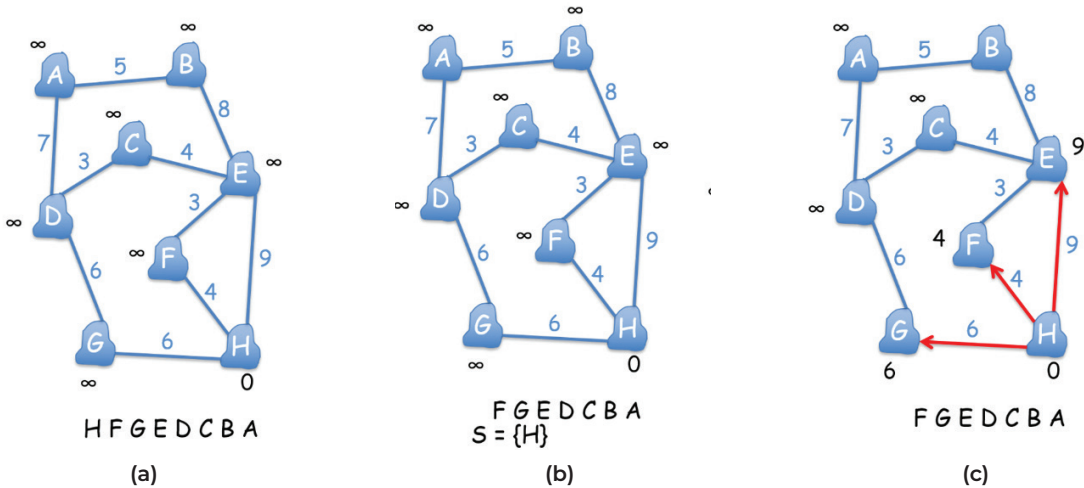


$v.dis > u.dis + w(u,v)$ $\rightarrow$ $v.dis = u.dis + w(u,v)$
 $12 > 8 + 3$ $\rightarrow$ $v.dis = 11$
 $v.par = u$

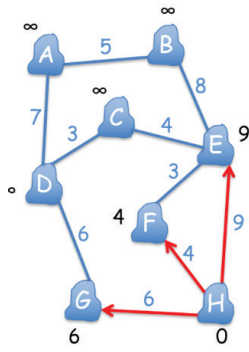
Şekil 11.5 Gevşetme işlemini ait bir örnek

Şekil 11.5'te gösterildiği gibi s düğümünden v düğümüne olan en kısa mesafe değerinin 12 olması durumunda, v düğümüne u düğümü üzerinden gidilmesi daha az maliyetli olacağı için v düğümünün mesafe değeri gevşetilerek 11 olarak

belirlenecektir. Şekil 11.6 üzerinden Dijkstra algoritmasının H düğümünden diğer düğümlere en kısa yolları nasıl bulduğu adım adım gösterilecektir:

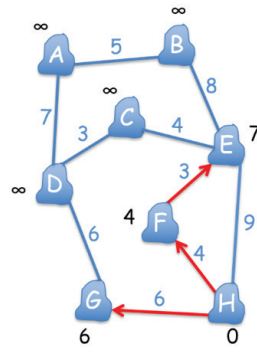


Şekil 11.6 Dijkstra algoritmasının örnek bir çizge üzerinde çalışma adımları



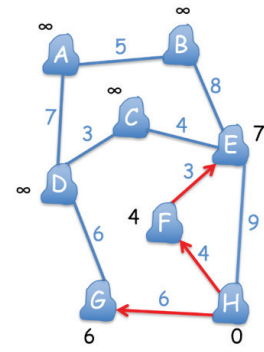
GEDCBA
S = {H, F}

(d)



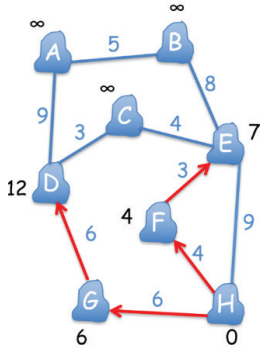
GEDCBA
S = {H, F}

(e)



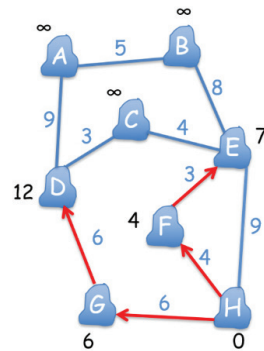
EDCBA
S = {H, F, G}

(f)



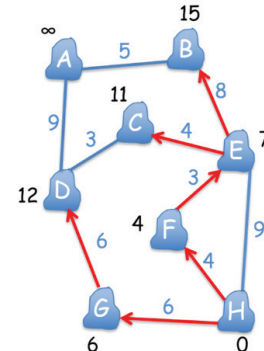
EDCBA
S = {H, F, G}

(g)



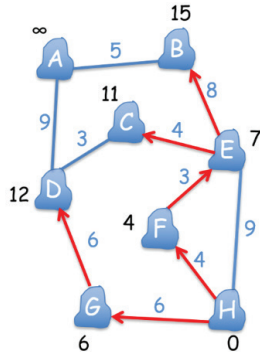
DCBA
S = {H, F, G, E}

(h)



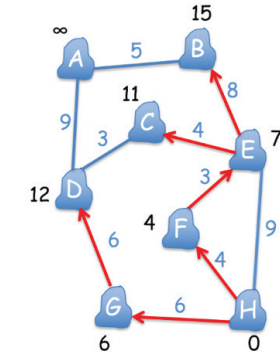
DCBA

(i)



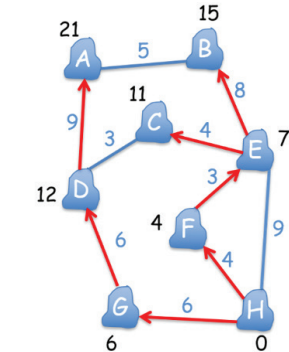
D B A
S = {H, F, G, E, C}

(j)



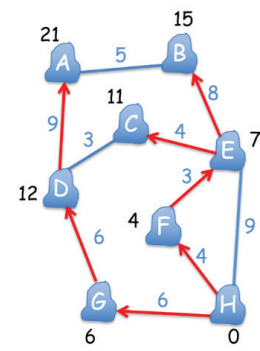
B A
S = {H, F, G, E, C}

(k)

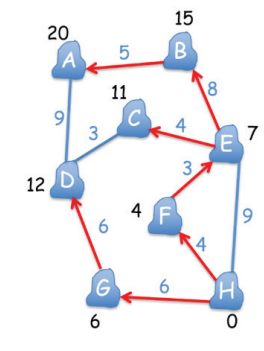


B A
S = {H, F, G, E, C, D}

(l)

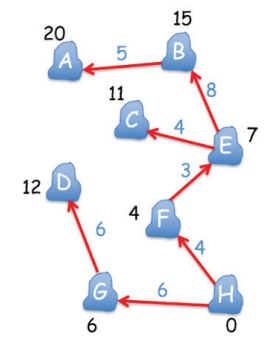


(m)



A
S = {H, F, G, E, C, D, B}

(n)



S = {H, F, G, E, C, D, B, A}

(o)

(a) Algoritma ilk olarak bütün düğümlere sonsuz mesafe değeri atar ve ardından başlangıç düğümünün mesafesini 0 olarak günceller ve çalışmaya başlar.

(b) En küçük mesafe değerli H düğümü Q kuyruğundan çıkartılarak S kümesine eklenir.

(c) H düğümünün komşuları E, F ve G düğümlerine gevşetme işlemi uygulanır: düğümlerin mesafe değerleri sırasıyla 9, 4, ve 6 olarak ve ebeveyn parametreleri H düğümü olarak güncellenir. Q minimum öncelikli kuyruğu düğümlerin yeni mesafe değerlerine göre güncellenir.

(d) Q kuyruğundaki en kısa mesafe değerine sahip F düğümü kuyruktan çıkartılarak S kümesine eklenir.

(e) F düğümünün komşusu E düğümüne gevşetme işlemi uygulanır: mesafe değeri 9 iken gevşetilmesiyle yeni mesafe değeri olarak 7 belirlenir ve ebeveyn parametresi F düğümü olarak güncellenir.

(f) Q kuyruğundaki en kısa mesafe değeri G düğümüne ait olduğu için bu düğüm kuyruktan çıkartılır ve S kümesine eklenir.

(g) G düğümünün komşusu D düğümüne gevşetme işlemi uygulanır: mesafe değeri 12 ve ebeveyn parametresi G düğümü olarak güncellenir.

(h) Q kuyruğundaki en kısa mesafe değeri E düğümüne ait olduğu için bu düğüm kuyruktan çıkartılır ve S kümesine eklenir.

(i) E düğümünün komşuları B ve C düğümlerine gevşetme işlemi uygulanır: düğümlerin me-

safe değerleri sırasıyla 15 ve 11 olarak ve ebeveyn parametreleri E düğümü olarak güncellenir. Q minimum öncelikli kuyruğu düğümlerin yeni mesafe değerlerine göre güncellenir.

(j) Q kuyruğundaki en kısa mesafe değerine sahip C düğümü kuyruktan çıkartılarak S kümesine eklenir.

(k) C düğümünün komşusu G düğümünün anahtar değeri 12, C'nin anahtar değeri ile (C, E) kenarının ağırlığı toplamından küçük olduğundan G düğümüne gevşetme işlemi uygulanmaz.

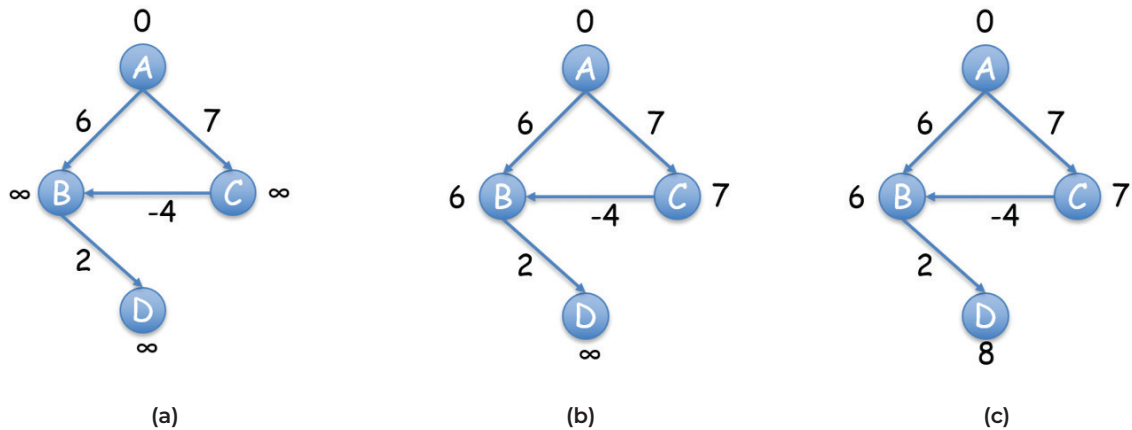
(l) Q kuyruğundaki en kısa mesafe değerine sahip D düğümü kuyruktan çıkartılarak S kümesine eklenir. D düğümünün komşusu A düğümüne gevşetme işlemi uygulanır: ebeveyn parametresi D düğümü ve anahtar değeri 21 olarak güncellenir.

(m) Q kuyruğundaki en kısa mesafe değerine sahip B düğümü kuyruktan çıkartılarak S kümesine eklenir.

(n) B düğümünün komşusu A düğümüne gevşetme işlemi uygulanır: ebeveyn parametresi B düğümü ve anahtar değeri 20 olarak güncellenir.

(o) Q kuyruğundaki en kısa mesafe değerine sahip A düğümü kuyruktan çıkartılarak S kümesine eklenir ve nihayetinde ebeveyn parametreleri kullanılarak en kısa yol ağacı oluşturulur.

Yukarıda da belirtildiği gibi Dijkstra algoritması negatif değerlere sahip bir çizge üzerinde doğru en kısa yolları bulamayacaktır. Burada bu durum bir örnek üzerinden anlatılacaktır.



Şekil 11.7 Negatif ağırlıklı bir kenar olması durumunda Dijkstra algoritmasının doğru sonuç üretmediğini gösteren örnek

A düğümünü kaynak düğüm olarak kabul edilmesi durumunda Dijkstra algoritması diğer düğümlere olan en kısa mesafeyi bulacaktır. Dijkstra algoritmasının çalıştırılması sonunda elde edilen mesafe değerleri Şekil 11.7.c'de gös-

terilmiştir. D düğümüne C düğümü üzerinden gidilmesi durumunda gerçek en kısa mesafe 5 olacaktır. Örnekten anlaşılacağı üzere Dijkstra D algoritması negatif kenar olması durumunda doğru sonuç üretememektedir.

BÖLÜM ÖZETİ

Bu bölüm, çizge algoritmalarını ele almakta olup, minimum kapsayan ağaç ve en kısa yol problemlerini çözmek için kullanılan temel algoritmaları incelemektedir. Çizge algoritmaları, ağ yönlendirmesi, zamanlama, veri madenciliği ve optimizasyon gibi birçok alanda kullanılmaktadır. Bu bağlamda en yaygın kullanılan algoritmalar Prim, Kruskal ve Dijkstra algoritmalarıdır.

Bir çizgede bulunan tüm düğümleri en düşük toplam kenar ağırlığıyla bağlayan en küçük ağacı oluşturan algoritmalar minimum kapsayan ağaç algoritmaları olarak adlandırılır. Prim ve Kruskal algoritmaları, bu problemi çözmek için kullanılan en yaygın yöntemlerdir.

Prim Algoritması, bir başlangıç düğümünden yola çıkarak her adımda en düşük maliyetli kenarı seçerek ağacı oluşturur. Algoritma, çizgedeki düğümleri birer birer ekleyerek her zaman en düşük ağırlıklı bağlantıyı tercih eder. Özellikle yoğun bağlantılara sahip çizgelerde daha verimli çalışmaktadır.

Kruskal Algoritması, kenarları ağırlıklarına göre sıralayarak en düşük ağırlıklı kenardan başlayıp minimum kapsayan ağacı oluşturmaktadır. Kruskal algoritmasının temel farkı, kenarları ağırlıklarına göre sıralı bir şekilde işleyerek döngü oluşturmadan ağacı tamamlamasıdır. Özellikle seyrek çizgelerde daha etkilidir.

Çizgelerdeki iki düğüm arasındaki en kısa mesafeyi bulmaya yönelik kullanılan algoritmalar en kısa yol algoritmalarıdır. Bu algoritmalar genellikle ulaşım, ağ trafiği yönlendirme ve optimizasyon problemlerinde kullanılır.

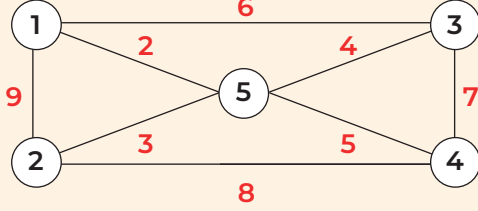
Dijkstra Algoritması, belirli bir başlangıç düğümünden diğer tüm düğümlere olan en kısa yolları bulmak için kullanılır. Algoritma, başlangıç düğümünden itibaren en kısa yolu bulmaya çalışarak ilerler ve her adımda en düşük mesafeye sahip düğümü işleyerek nihai en kısa yolu belirler. Negatif ağırlıklı kenarlarda doğru çalışmaz; bu yüzden ağırlıkların negatif olmadığı çizgelerde kullanılır.

Dijkstra algoritması, öncelikli kuyruk yapısını kullanarak, en düşük maliyetli düğümü işleyerek ilerler. Algoritma, belirli bir noktadan hareket ederek her düğüme en kısa yolu belirleyip, en kısa yol ağaçlarını oluşturur. Bunun sonucunda, belirlenen düğüme en düşük maliyetle ulaşmak mümkün hale gelir.

Bu bölümde, Prim ve Kruskal algoritmalarının minimum kapsayan ağaç problemlerine, Dijkstra algoritmasının ise en kısa yol problemlerine nasıl uygulandığı incelenmiştir. Prim algoritması, ağaç yapısını oluştururken düğümleri merkeze alarak en düşük maliyetli kenarları eklerken, Kruskal algoritması kenarları sıralayarak döngü oluşturmadan ilerlemektedir. Dijkstra algoritması ise belirli bir düğümden hareketle en kısa yolları hesaplamak için kullanılır. Bu üç algoritma, ulaşım, ağ yönlendirme, optimizasyon ve veri madenciliği gibi alanlarda önemli uygulamalara sahiptir.

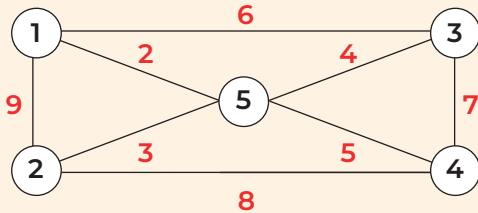
GÖZDEN GEÇİRELİM

1. Aşağıda verilen çizgeye ait Minimum Kapsayan ağaçta hangi kenar bulunmaz?



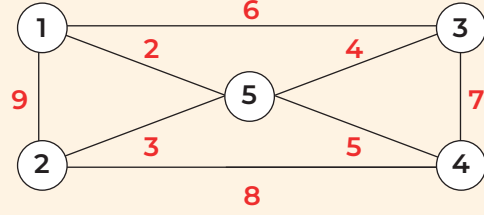
- a) (1, 5)
- b) (2, 5)
- c) (3, 5)
- d) (4, 5)
- e) (1, 3)

2. Aşağıda verilen çizge üzerinde Kruskal Minimum Kapsayan ağaç algoritması çalıştırılacaktır. Algoritma ilk olarak hangi kenardan çalışmaya başlar?



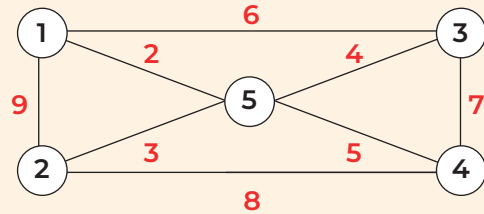
- a) (1, 5)
- b) (2, 4)
- c) (3, 4)
- d) (2, 5)
- e) (1, 3)

3. Aşağıda verilen çizge üzerinde Prim Minimum Kapsayan ağaç algoritması "5" düğümü seçilerek başlatılması durumunda, algoritma sonucunda "4" düğümünün anahtar parametresinin alacağı değer ne olacaktır?



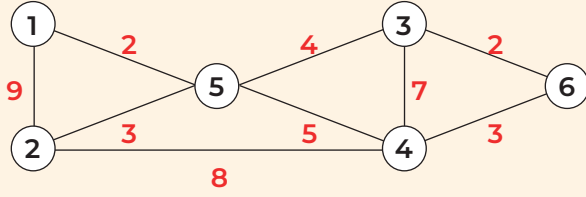
- a) 4
- b) 5
- c) 6
- d) 7
- e) 8

4. Aşağıda verilen çizgeye ait Minimum Kapsayan ağaçta kaç adet kenar bulunmalıdır?



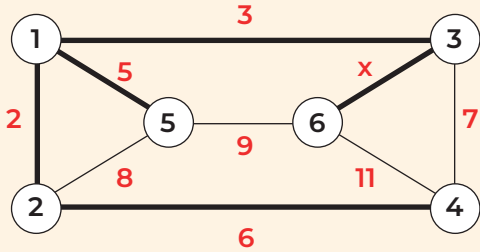
- a) 3
- b) 4
- c) 5
- d) 6
- e) 7

5. Aşağıda verilen çizgeye ait Minimum Kapsayan ağaçta hangi kenar bulunmaz?



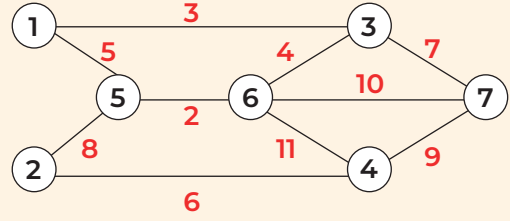
- a) 4
- b) 6
- c) 8
- d) 10
- e) 12

6. Aşağıda verilen çizgede kalın kenarlar Minimum Kapsayan Ağacı oluşturmaktadır. Bu durumda "x" ile ifade edilen kenar ağırlığı aşağıdakilerden hangisi olamaz.



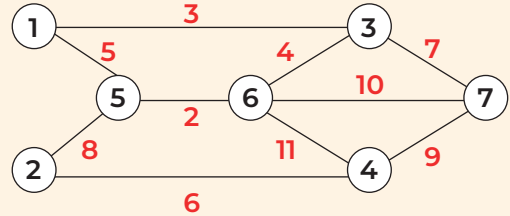
- a) 2
- b) 4
- c) 6
- d) 8
- e) 10

7. Aşağıda verilen çizgeye ait Minimum Kapsayan ağaçta hangi kenar bulunmaz?



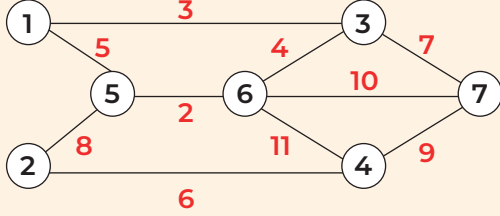
- a) (1, 5)
- b) (1, 3)
- c) (5, 6)
- d) (3, 7)
- e) (2, 4)

8. Aşağıda verilen çizge üzerinde Kruskal Minimum Kapsayan ağaç algoritması çalıştırılacaktır. Algoritma ilk olarak hangi kenardan çalışmaya başlar?



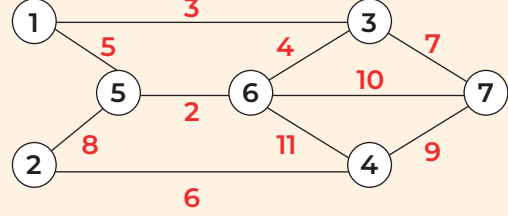
- a) (1, 5)
- b) (2, 4)
- c) (2, 5)
- d) (3, 6)
- e) (5, 6)

9. Aşağıda verilen çizge üzerinde Prim Minimum Kapsayan ağaç algoritması “6” düğümü seçilerek başlatılması durumunda, algoritma sonucunda “1” düğümünün anahtar değeri ne olacaktır?



- a) 3
- b) 4
- c) 5
- d) 7
- e) 9

10. Aşağıda verilen çizgeye ait Minimum Kapsayan ağaçta kaç adet kenar bulunmalıdır?



- a) 3
- b) 4
- c) 5
- d) 6
- e) 7

Yanıt Anahtarı: 1-E 2-A 3-B 4-B 5-C 6-E 7-A 8-E 9-A 10-D

Kaynakça

Levitin A. (2012) Introduction to The Design & Analysis of Algorithms, 3rd Edition, Pearson.

Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. (2009) Introduction to Algorithms, 3rd Edition, MIT Press.

Çölkesen T.R. (2019) Veri Yapıları ve Algoritmalar, Papatya Yayıncılık

Sedgewick R., Wayne K. (2014) Algorithms, 4th Edition, Addison-Wesley

Çamoğlu K. (2021) Algoritma, Kodlab