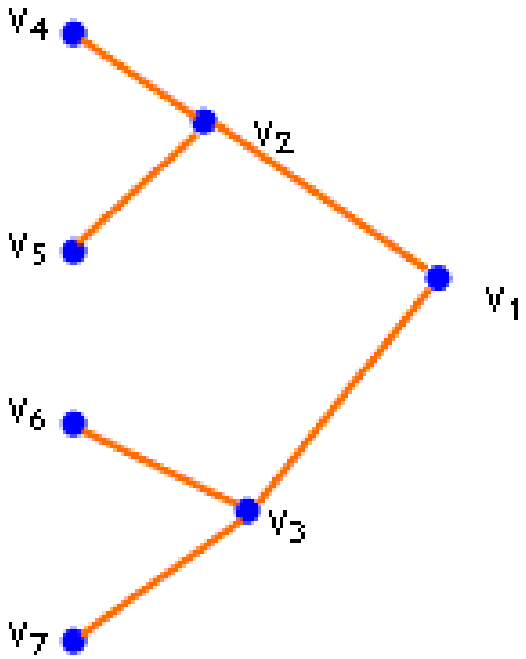
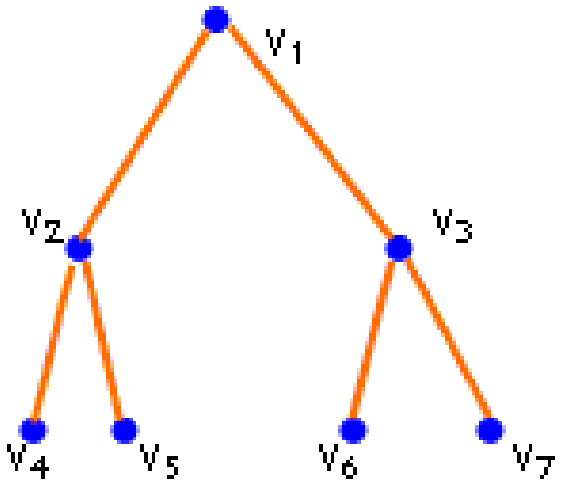
BÖLÜM 8

AĞAÇLAR

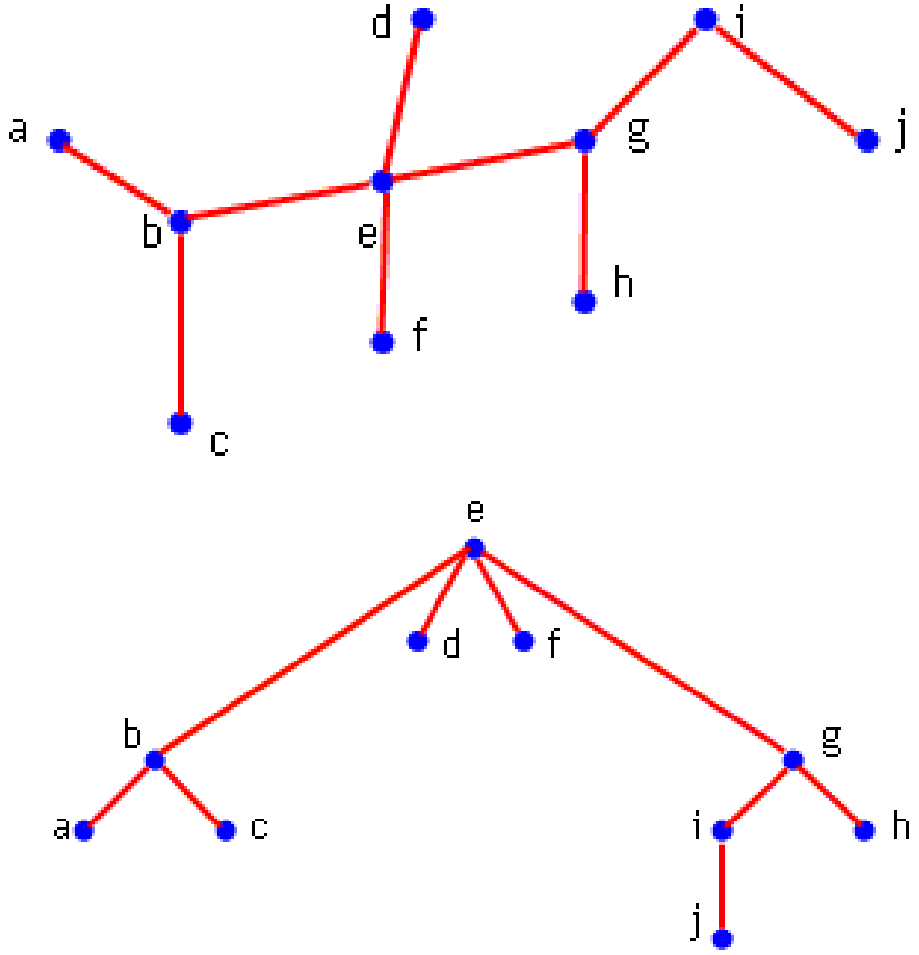
- **Tanım 8.1.1:** Bir T (*serbest) ağacı* aşağıdaki özelliği sağlayan bir basit çizgedir:
- T çizgesinde iki köşe v ve w ise, bu durumda v köşesinden w köşesine tek bir basit yol vardır.
- Bir *köklü ağaç* ise, ağacın belirli bir köşesinin kök olarak tasarlanmasıyla elde edilen ağaçtır.

Örnek 8.1.2

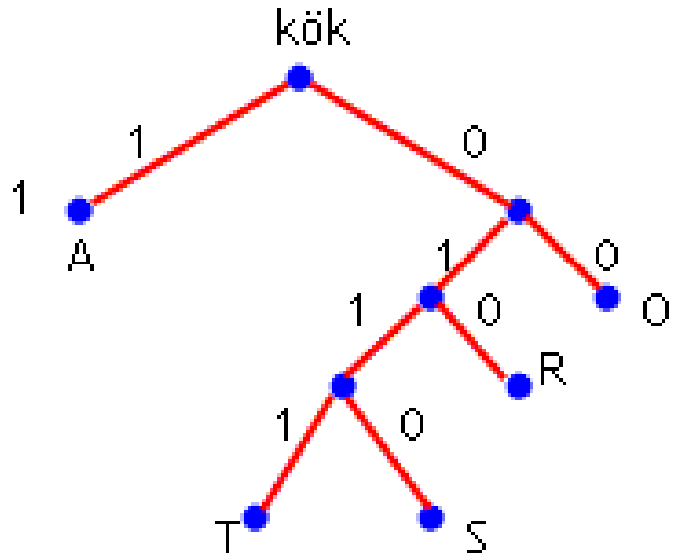




Örnek 8.1.4



Huffmann Kodlaması

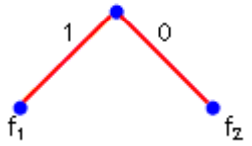


Algoritma 8.1.6

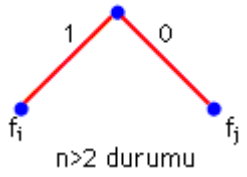
Girdi: n≥2 olmak üzere sıklıkların bir n dizisi

Çıktı: Bir optimal Huffman kodlamasını tanımlayan bir köklü ağaç

```
huffman(f,n)
{
   if (n==2)
   {
      f1 ve f2 sıklık tanımlasınlar
      T ağacı Şekil 1 olduğu gibi olsun
      return (T)
   }
   fi ve fj en küçük sıklıkları tanımlasınlar
   f listesinde fi ve fj yerine fi+fj alalım
   T'=huffman(f,n-1)
   T ağacını elde etmek için Şekil 2 'de olduğu gibi T'
      ağacında fi+fj ile etiketli bir köşe koy.
   return(T)
}
```

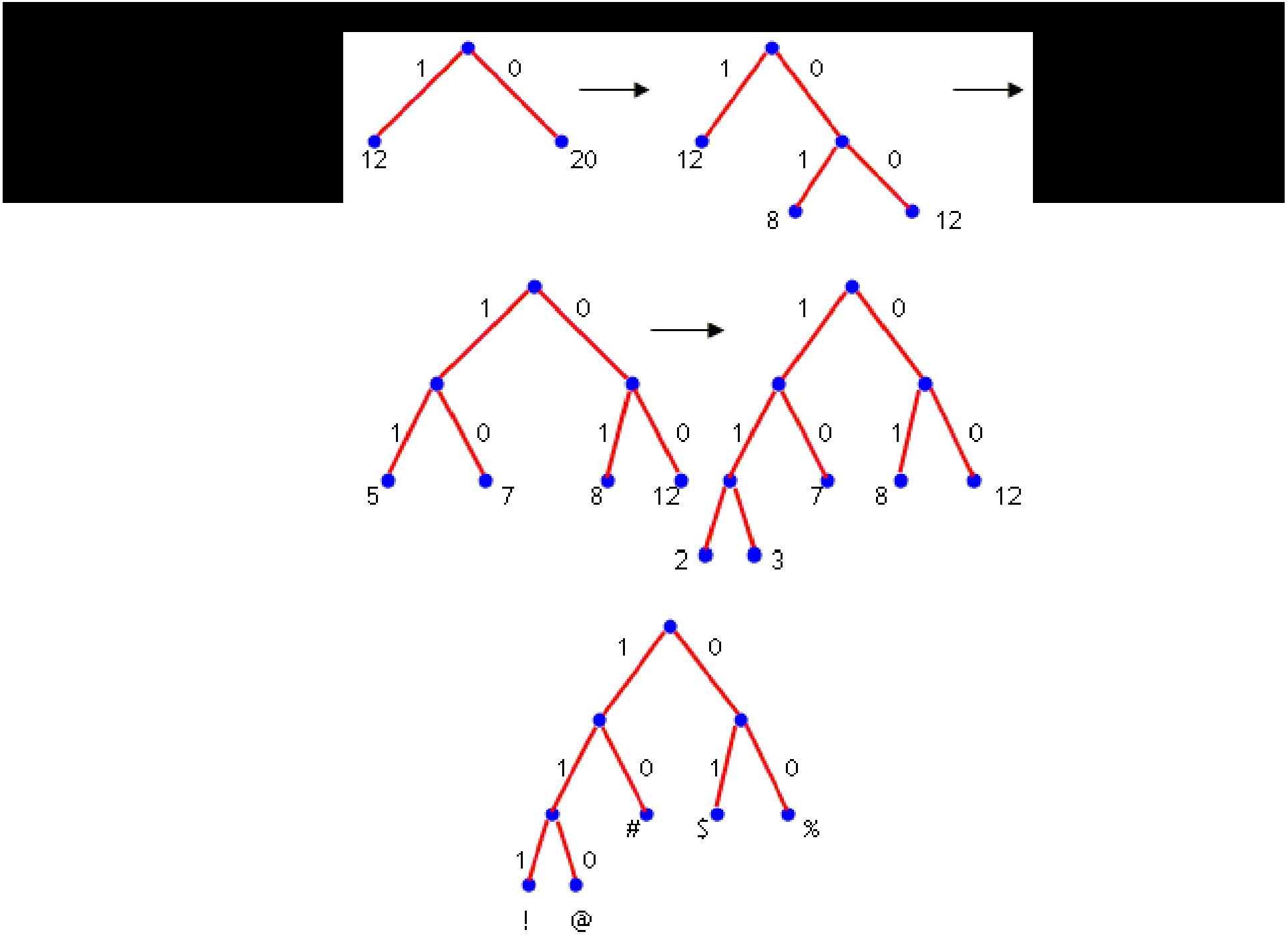


Şekil 1



Şekil 2

Karakter	Sıklık
!	2
@	3
#	7
$	8
%	12


1
0
12
20
1
0
12
1
0
8
12
1
0
1
0
1
0
5
7
8
12
1
0
1
0
1
0
1
0
7
8
12
2
3
1
0
1
0
1
0
#
$
%
1
0
!
@

AĞAÇLARIN KARAKTERİZASYONU

Tanım 8.2.1: v_o kökü ile bir ağaç T olsun. T ağacında x,y ve z köşeler ve $(v_o,v_1,\ldots,v_n)$ bir basit yol olsun. Bu durumda

a) v_{n-1} köşesi v_n köşesinin *üstüdür* denir.

b) $v_o,\ldots,v_{n-1}$ köşeleri v_n köşesinin *atasıdır* (ancestor) denir.

c) v_n köşesi v_{n-1} köşesinin altıdır denir.

d) y köşesinin atası x köşesi ise, y köşesi x köşesinin *torunudur* (descendant) denir.

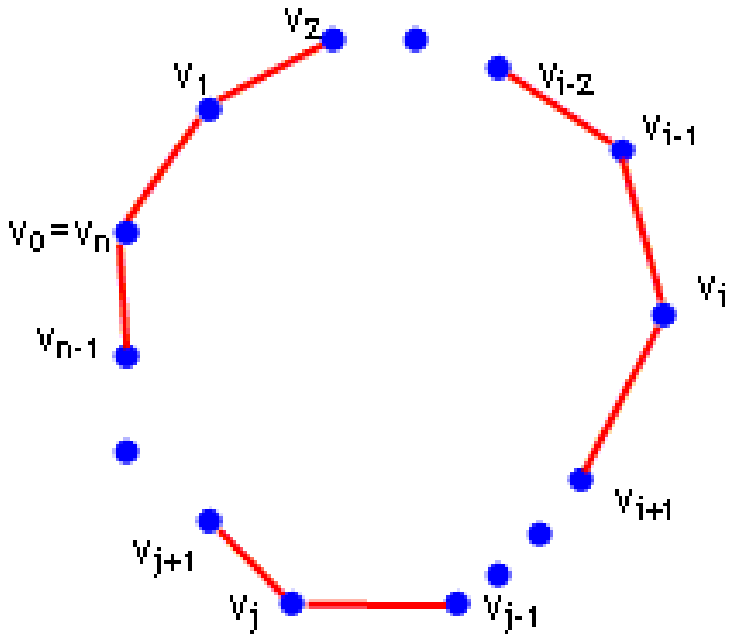
e) z köşesinin altları x ve y ise, x ve y köşelerine *kardeştirler* denir.

f) Eğer x köşesi bir son köşe değilse, x köşesine bir *iç köşedir* denir.

g) T ağacının bir x köşesinden köklenmiş olan bir *altağacı*, V köşeler kümesi x ve onun torunlarından oluşan, E kenarlar kümesinin

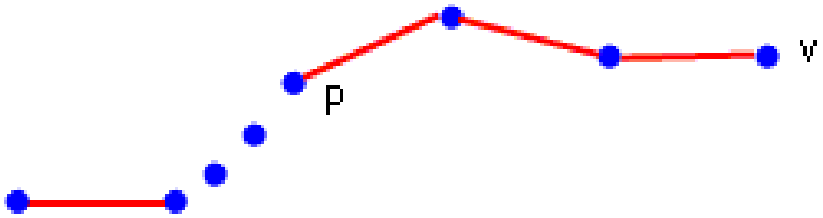
$$E=\{ e \mid e \text{ kenarı } x \text{ köşesinden } V \text{ 'deki bir köşeye olan basit bir yol üzerinde bir kenardır}\}$$

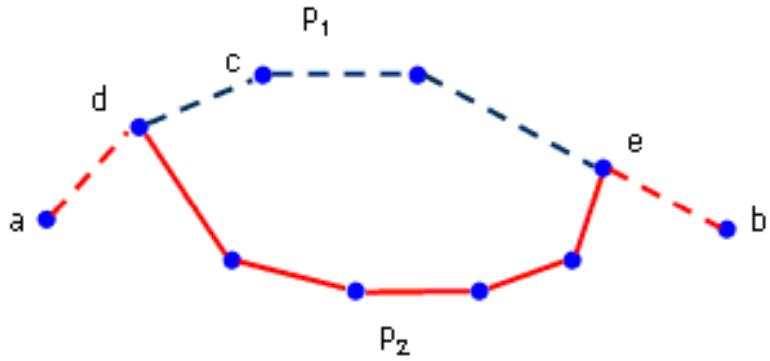
olduğu bir ağaçtır.


v_2
v_1
v_{i-2}
v_{i-1}
$v_0 = v_n$
v_i
v_{n-1}
v_{i+1}
v_{j+1}
v_j
v_{j-1}

Teorem 8.2.3: n adet köşeden oluşan bir çizge T olsun. Aşağıdaki ifadeler denktir.

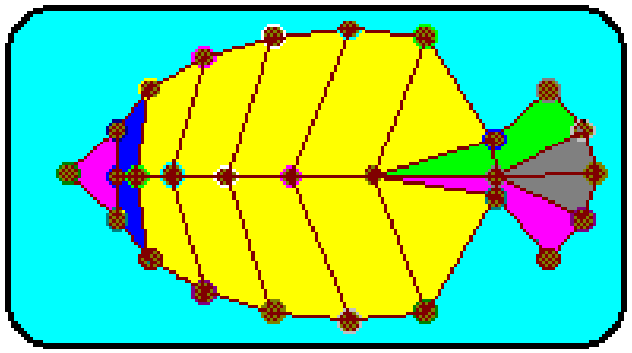
a) T bir ağaçtır
b) T bağlantılıdır ve çevrimsel içermez
c) T bağlantılıdır ve n-1 adet kenarı vardır
d) T çevrimsel içermez ve n-1 adet kenarı vardır

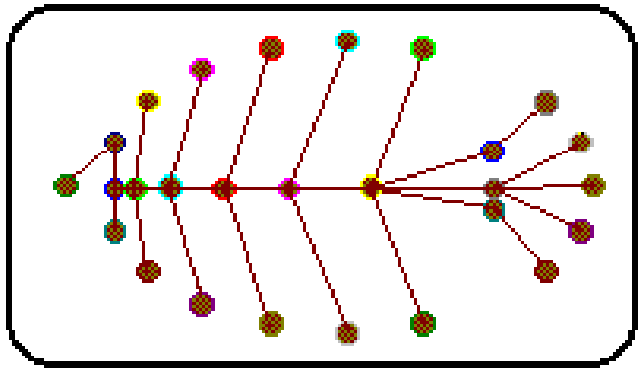




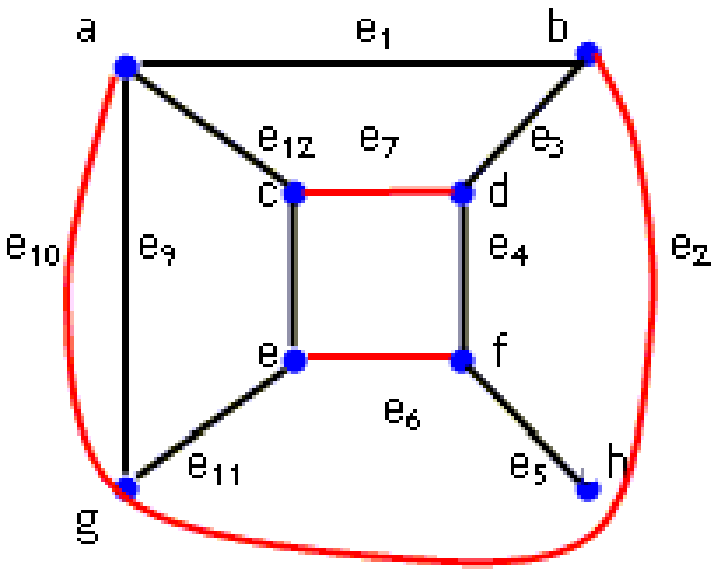
UZANIMLI AĞAÇLAR

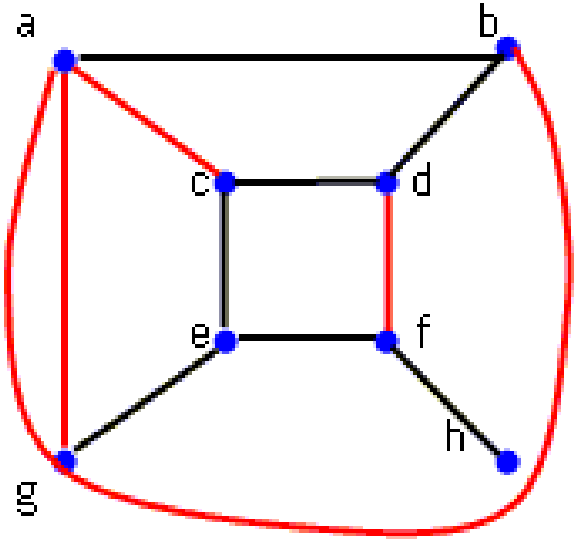
- **Tanım 8.3.1:** G bir çizge olsun. Eğer G çizgesinin bir T altçizgesi G çizgesinin tüm köşelerini içeren bir ağaç ise, T ağacına bir *uzanımlı ağaç* denir.





Örnek 8.3.2





- **Teorem 8.3.4:** *Bir G çizgesinin uzanımlı ağaca sahip olması için gerekli ve yeterli koşul çizgenin bağlantılı olmasıdır.*

Algoritma 8.3.6: Bir Uzanımlı Ağaç İçin Genişlik Öncelikli (Breadth-First) Arama

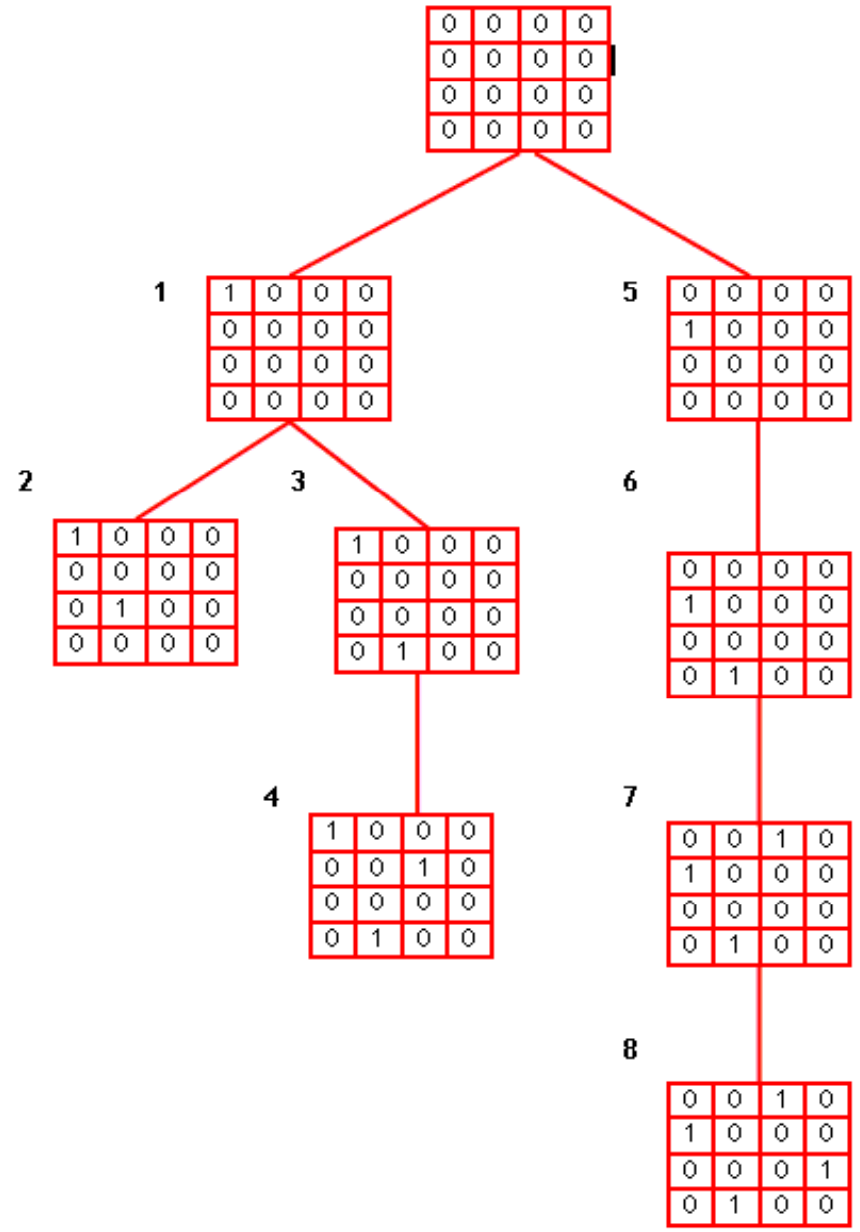
```
Girdi:  Sıralı v1,v2,...,vn köşeleriyle bir G bağlantılı çizgesi
Çıktı: T uzanımlı ağacı
bfs(V,E)
{
     // V= v1,...,vn sıralı köşeler, E= kenarlar
     // V'=T uzanımlı ağacının köşeleri, E'= T uzanımlı ağacının
     // kenarları
     // v1 =uzanımlı ağacın kökü
     // S = bir sıralı liste
     S=(v1)
     V'={v1}
     E'=∅
     while (true) {
            for  herbir x∈S için
                 for herbir y∈V-V' için
                     if ((x,y) bir kenarsa)
                            (x,y) kenarını E' 'ye ve y köşesini
          V' 'ye ekle
            if (eklenecek kenar yoksa)
               return(T)
            S= S 'nin altları (orjinal köşe sırasına göre sıralı)
      }
     }
```

Algoritma 8.3.7: Bir Uzanımlı Ağaç için Derinlik Öncelikli (Depth-First) Araması

```
Girdi: Sıralı v1,v2,...,vn köşeleriyle bir G bağlantılı çizgesi
Çıktı: T uzanımlı ağacı
dfs(V,E){
      // V'=T uzanımlı ağacının köşeleri,
      // E'= T uzanımlı ağacının kenarları
      // v1 =uzanımlı ağacın kökü
      // S = bir sıralı liste
      V'={v1}
      E'=∅
      w =v1
      while (true) {
            while (T 'ye eklendiğinde herhangi bir çevrimsel
            oluşturmayacak şekilde bir (w,v) kenarı varsa) {
                T 'ye eklendiğinde herhangi bir çevrimsel
                oluşturmayacak şekilde minimum k değerli
                (w,vk) kenarını seç
                    E' 'ye (w,vk) kenarını ekle
                    V' 'ye vk köşesini ekle
                    w = vk
                } // içteki while
          if (w == v1 )
                        return(T)
          w = T ağacında w köşesinin üstü  // geriye doğru
        }
}
```

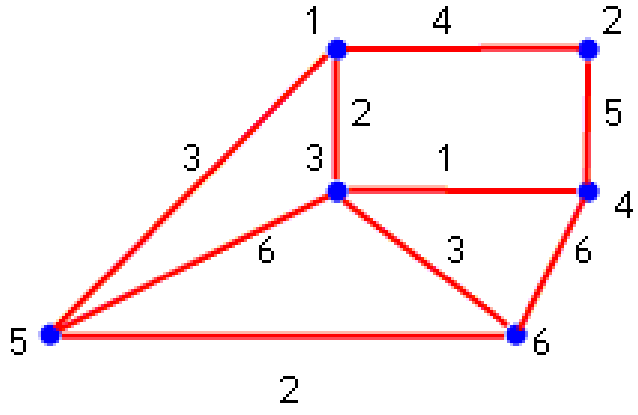
Algoritma 8.3.10: Dört Vezir Problemi

```
Girdi: Satır (row) boyu 4 olan bir dizi
Çıktı: çözüm varsa true ;  çözüm yoksa false
       [Eğer çözüm varsa, k. kıraliçe sutün k 'dadır,
                satır row(k) olur]
four_queens(row) {
        k = 1
        row(1)=0
        while (k>0) {
                row(k) = row(k)+1
                   // sutün k da geçerli bir hareket için bak
                  while ( (row(k) ≤ 4) and (sutün k, row(k)
                çakışması varsa))
                     // bir sonraki satırı dene
                     row(k) := row(k)+1

         if (row(k)≤ 4)
             if (k==4) // çözüm tamam
                      return true
                   else  //sonraki sutün
                   k = k+1
                   row(k) = 0
                }
                else
                      k = k+1
         } }
```

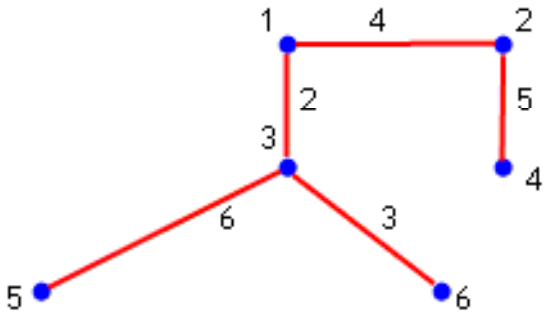

1
2
3
4
5
6
7
8

MİNİMAL UZANIMLI AĞAÇLAR

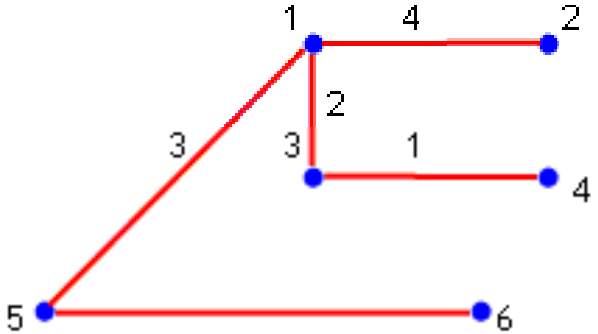
Örnek 8.4.2



G



T'



T

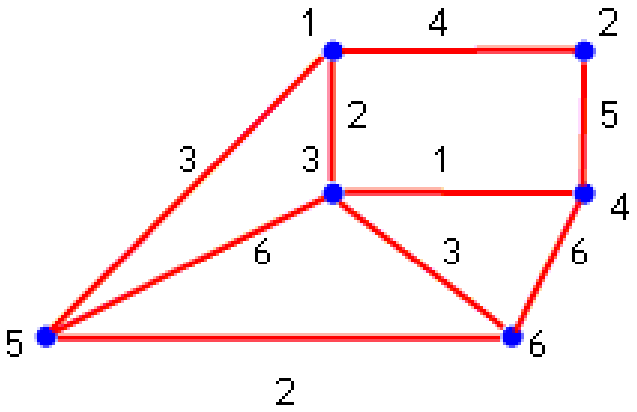
PRIM ALGORİTMASI

Girdi: 1,...,n köşeleri ve bir s başlangıç köşesi ile bir bağlantılı, ağırlıklı çizge. Eğer (i,j) bir kenar ise, (i,j) kenarının ağırlığını w(i,j) gösterir. Eğer (i,j) bir kenar değilse, w(i,j) değeri ∞ (herhangi güncel değerden büyük olan bir değer) değerine eşittir.

Çıktı: Bir minimal uzanımlı ağacının (mua) E kenarlar kümesi

```
prim(w,n,s){
          // eğer i köşesi mua 'ye eklenecekse v(i)=1
     // eğer i köşesi mua 'ya eklenmeyecekse v(i)=0
          1.      for i=1 to n
          2.          v(i) = 0
     // mua 'ya başlangıç köşesini ekleme
          3.      v(s)=1
       // bir boş kenar kümesi ile başlama
          4.      E = ∅
       // minimal uzanımlı ağaca n-1 adet kenar ekle
          5.      for i =1 to n-1 {
           // bir köşesi mua içinde ve diğer köşesi mua dışında
           // olan minimum ağırlıklı bir kenarı ekle
          6.           min = ∞
          7.           for j = 1 to n
          8.               if (v(j) = 1)  //j mua içinde bir köşe
          9.                    for k = 1 to n
          10.                       if (v(k)== 0  and w(j,k) < min){
          11.                          add_vertex = k
          12.                          e = (j,k)
          13.                          min = w(j,k)
          14.                       }  // if sonu
           // mua içine köşe ve kenar ekle
          16.         v(add_vertex) = 1
          17.         E = E ∪ {e}
          18.      }  // for sonu
          19.      return E
}
```

Örnek 8.4.4



G

Kenar	Ağırlık
(1,2)	4
(1,3)	2
(1,5)	3

Kenar	Ağırlık
(1,2)	4
(1,5)	3
(3,4)	1
(3,5)	6
(3,6)	3

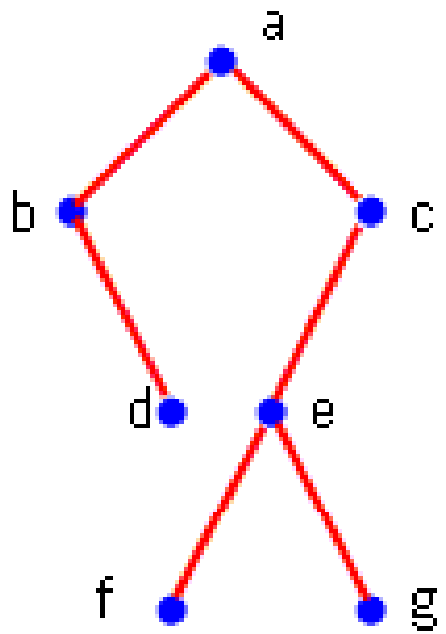
Kenar	Ağırlık
(1,2)	4
(1,5)	3
(2,4)	5
(3,5)	6
(3,6)	3
(4,6)	6

Kenar	Ağırlık
(1,2)	4
(2,4)	5
(3,6)	3
(4,6)	6
(5,6)	2

Kenar	Ağırlık
(1,2)	4
(2,4)	5

İKİLİ AĞAÇLAR

- **Tanım 8.5.1:** Bir *ikili ağaç* her bir köşesi ya iki alt ya tek bir alt ya da hiç bir alt köşe içermeyen bir köklü ağaçtır.



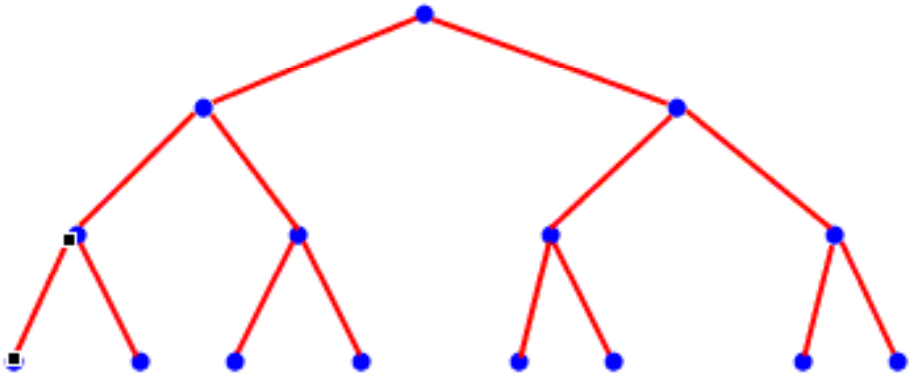
- **Teorem 8.5.3:** *Eğer T ağacı i adet iç köşesi ile bir tam ikili ağaç ise, bu durumda T ağacı i+1 adet son köşeye ve 2i+1 adet toplam köşeye sahiptir.*

- **Teorem 8.5.4:** *Yüksekliği h olan bir ikili ağacın son köşelerinin sayısı t ise, bu durumda*

$$\lg t \leq h$$

olur.

Örnek 8.5.5:

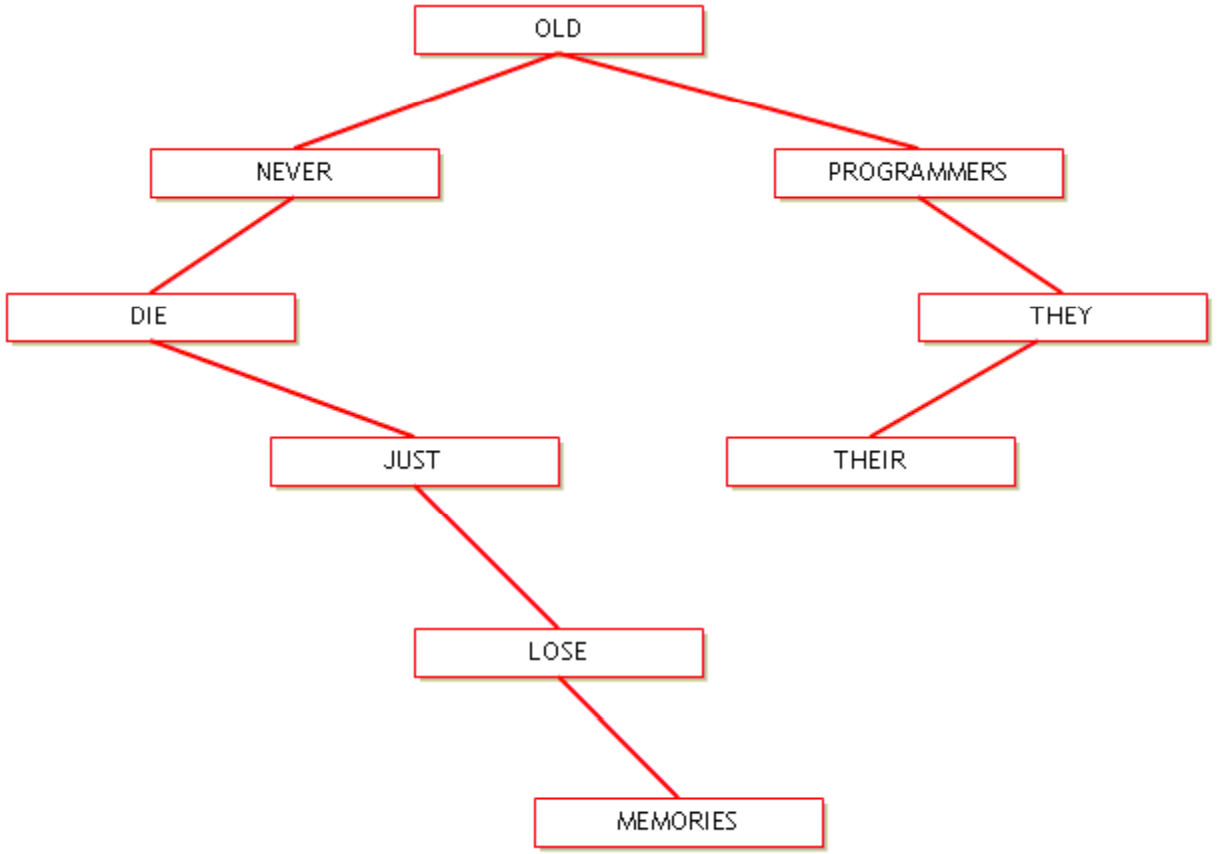


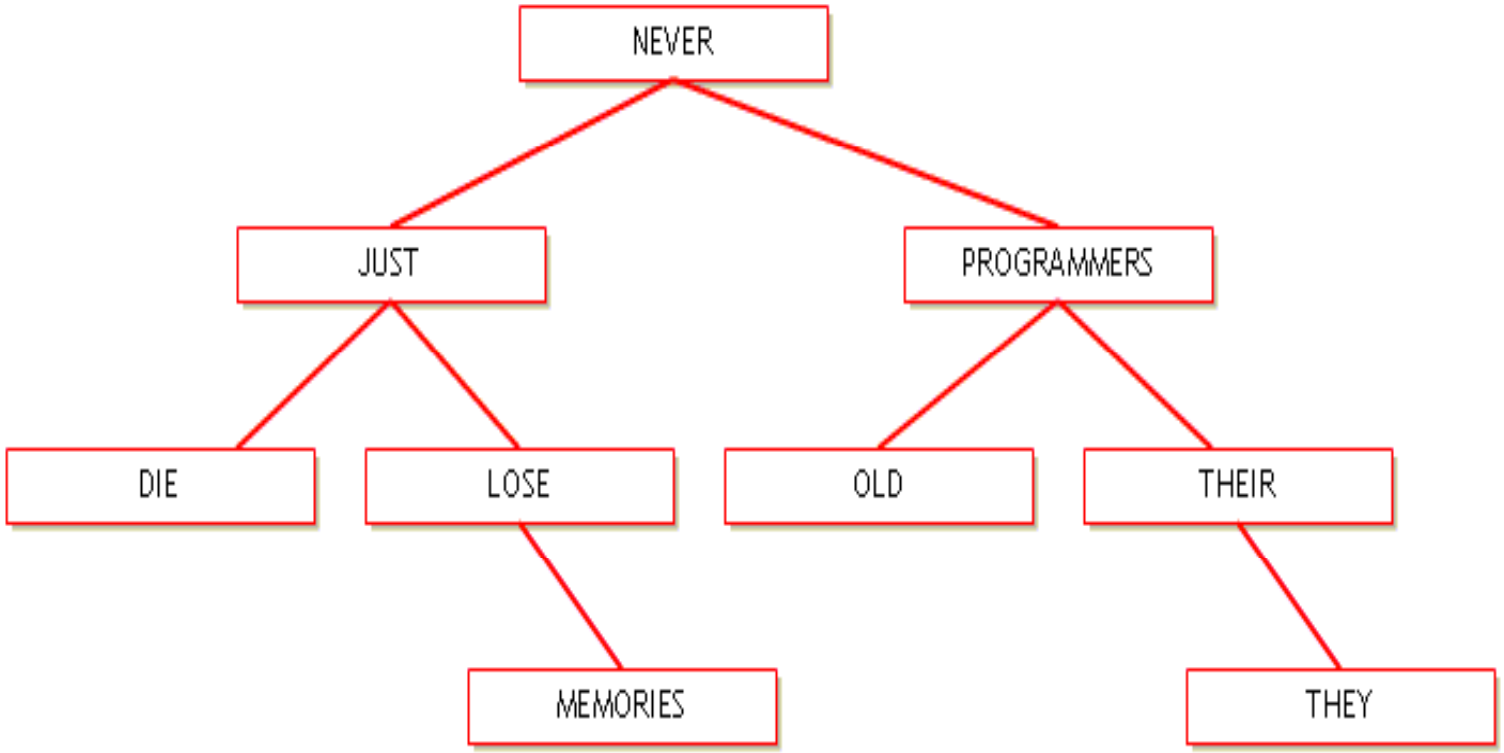
- **Tanım 8.5.6:** Verilerin köşeler ile ilişkilendirildiği ikili arama ağaçlarına bir *ikili arama ağacı* denir. T ağacındaki her bir v köşesi için, v köşesinin sol altağacının köşelerinde bulunan veriler v köşesindeki veriden daha küçük ve v köşesinin sağ altağacının köşelerinde bulunan veriler v köşesindeki veriden daha büyüktür

Örnek 8.5.7:

OLD PROGRAMMERS NEVER DIE

THEY JUST LOSE THEIR MEMORIES




NEVER
JUST
PROGRAMMERS
DIE
LOSE
OLD
THEIR
MEMORIES
THEY

Algoritma 8.5.8: İkili Arama Ağacı Oluşturma

Girdi: Birbirinden farklı kelimelerin bir $w_1,...,w_n$ dizisi ve dizi uzunluğu n
Çıktı: T ikili arama ağacı

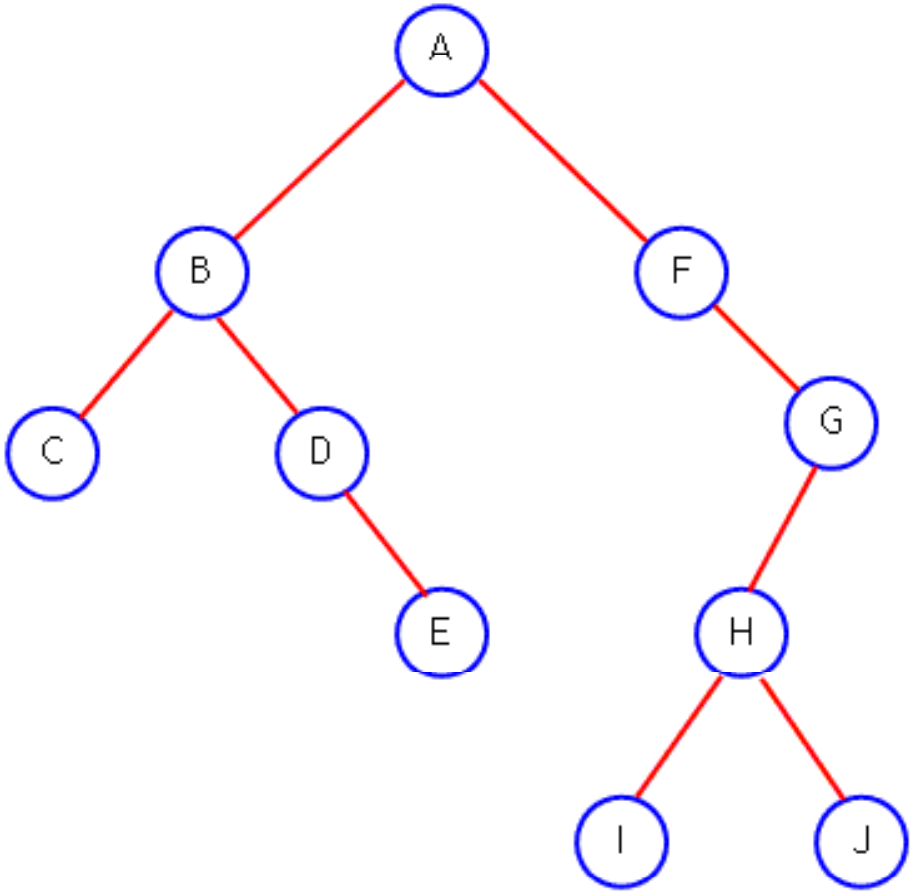
```
make_bin_search_tree(w,n) {
  tek bir köşeden (kök) oluşan ağaç T olsun,
  w1 değeri kökde tutulsun
  for i=2 to n {
      v = kök
      arama = true
      while (arama) [
           s = v köşesindeki kelime
           if (wi < s)
               if (v köşesinin sol altı yoksa) {
                    v köşesine l 'yi sol alt olarak ekle
                    wi kelimesini l köşesinde tut
                    arama = false //arama bitti
               }
               else
                     v = v köşesinin sol altı
           else  // wi>s
               if (v köşesinin sağ altı yoksa) {
                    v köşesine r 'yi sağ alt olarak ekle
                    r köşesinde wi kelimesini tut
                    arama = false    // arama bitti
                }
               else
                    v = v köşesinin sağ altı
        }  //while
   }  // for
  return T  }
```

AĞAÇ TARAMA

Önsıralı Tarama

```
Girdi: Bir ikili ağacın kökü PT
Çıktı: 3. satırda verilen süreçe bağlı olarak değişir.

preorder(PT){

1   if (PT boşsa)
2     return
3   PT 'ye süreç uygula
4   l = PT 'nin sol altı
5   preorder(l)
6   r = PT 'nin sağ altı
7   preorder(r)

}
```

Örnek 8.6.2

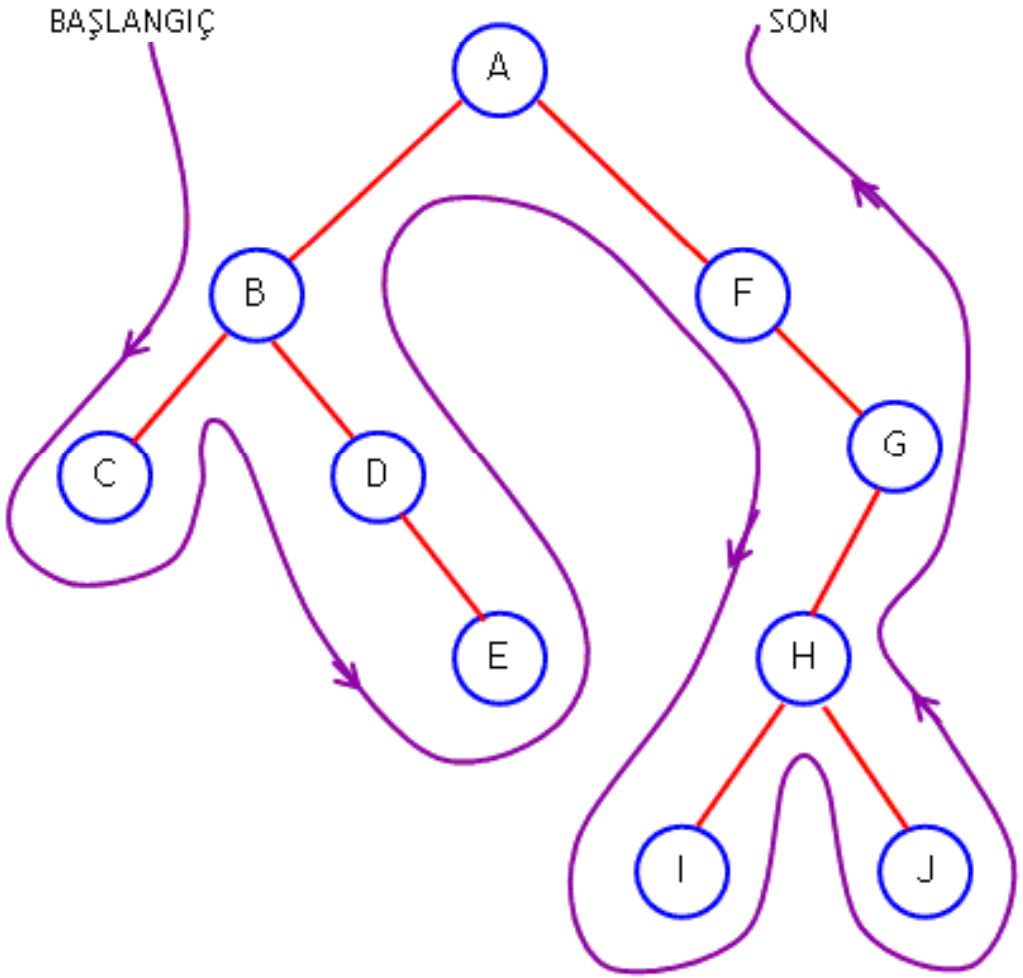


Inorder Tarama

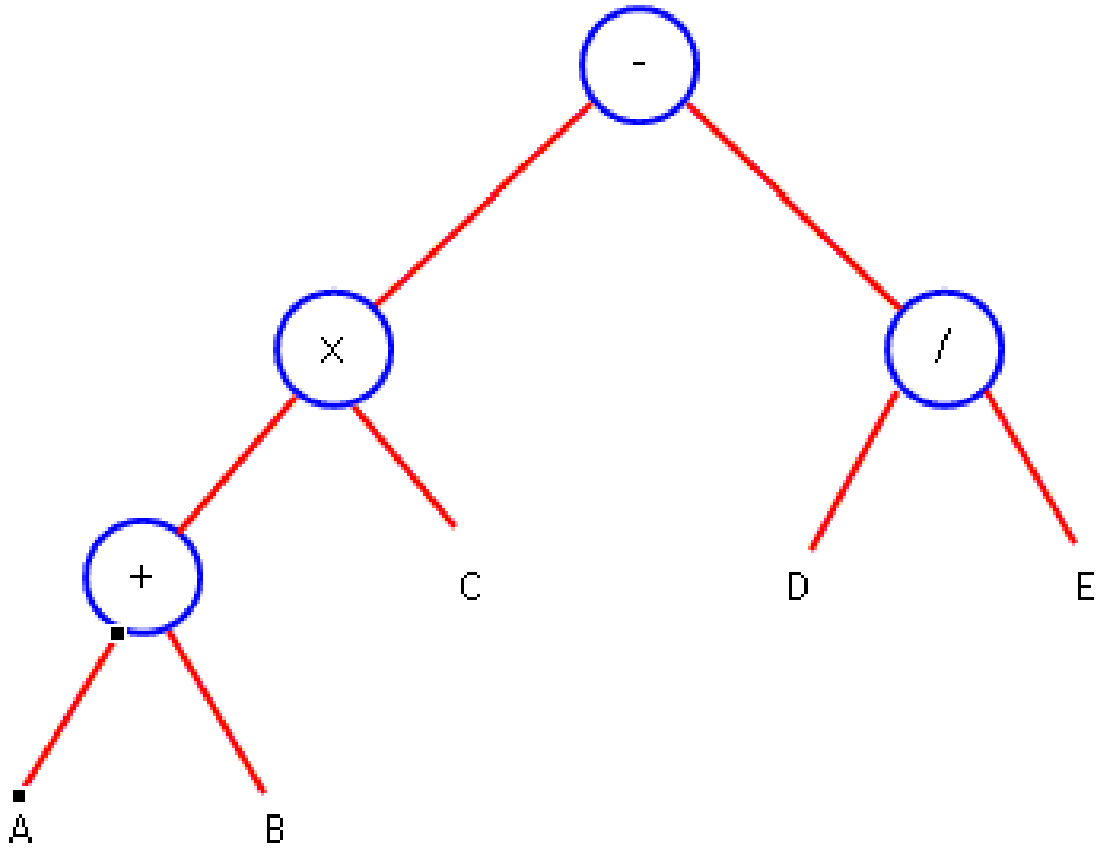
```
Girdi: Bir ikili ağacın kökü PT
Çıktı: 5. satırda verilen süreçe bağlı olarak değişir.

inorder(PT) {

    1.  if (PT boşsa)
    2.    return
    3.  l = PT 'nin sol altı
    4.  inorder(l)
    5.  PT 'ye süreç uygula
    6.  r = PT 'nin sağ altı
    7.  inorder(r)
}
```

Son sıralı tarama

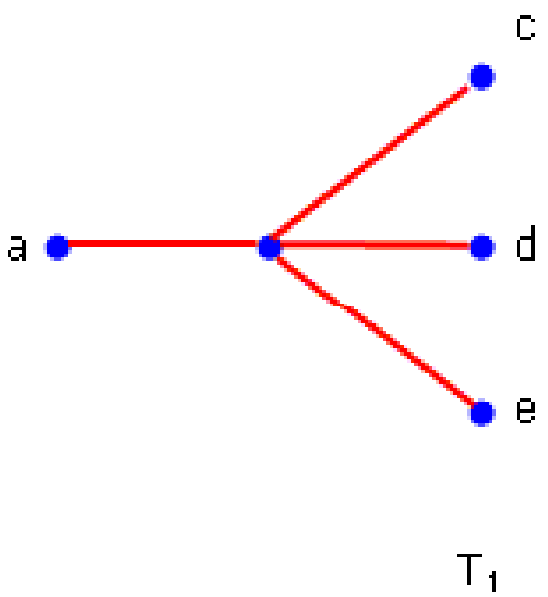
```
Girdi: Bir ikili ağacın kökü PT
Çıktı: 7. satırda verilen süreçe bağlı olarak değişir.

postorder(PT){

    1.  if (PT boşsa) then
    2.    return
    3.  l = PT 'nin sol altı
    4.  postorder(l)
    5.  r = PT 'nin sağ altı
    6.  postorder(r)
    7.  PT 'ye süreç uygula
}
```


BAŞLANGIÇ
SON
A
B
F
C
D
G
E
H
I
J

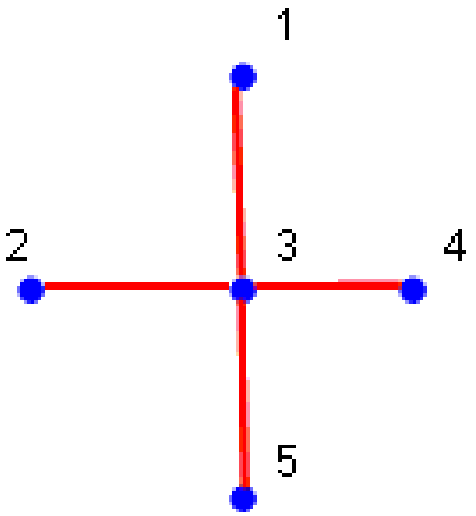
((A+B)xC)-(D/E)



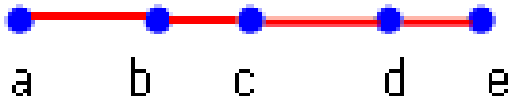
AĞAÇLARIN EŞYAPILI OLMALARI



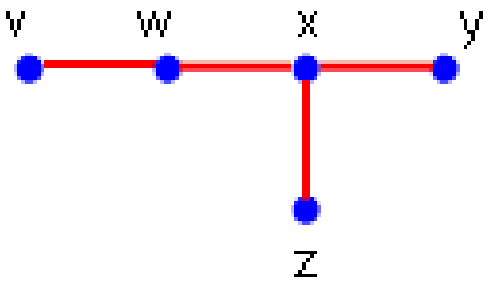
T_1



T_2

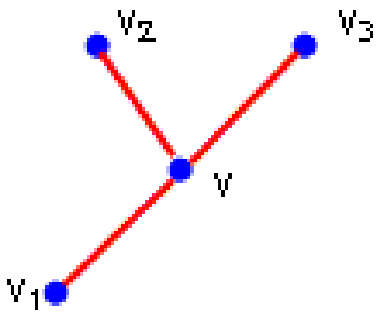


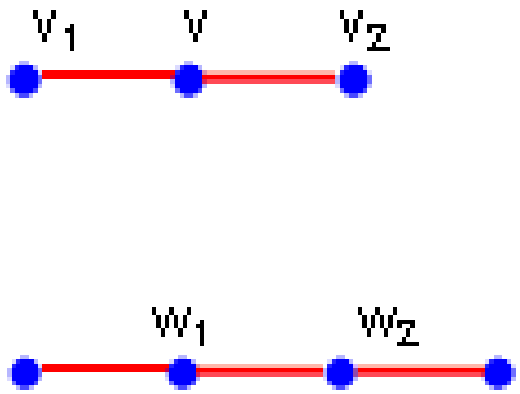
T_1



T_2

- **Teorem:** *Beş adet köşeye sahip olan üç adet eşyapılı olmayan ağaç vardır.*



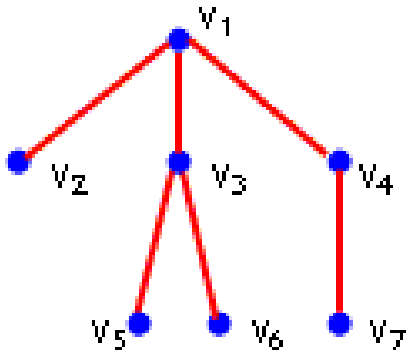


- **Tanım:** r_1 kökü ile bir köklü ağaç T_1 ve r_2 kökü ile bir köklü ağaç T_2 olsun. Aşağıdaki özellikleri sağlayan T_1 ağacının köşeler kümesinden T_2 ağacının köşeler kümesine bire-bir, üzerine bir f fonksiyonu varsa *T_1 ve T_2 ağaçlarına eşyapılıdırlar* denir.

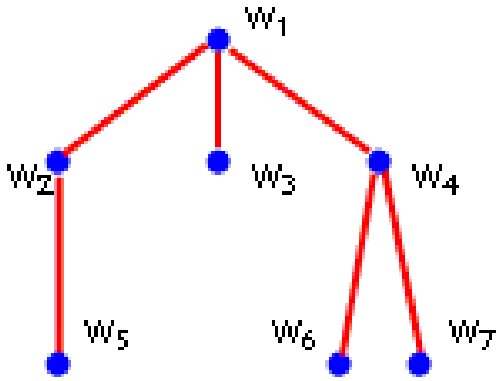
  a) T_1 ağacında v_i ve v_j köşeleri komşu köşe olması için gerekli ve yeterli koşul T_2 ağacında $f(v_i)$ ve $f(v_j)$ köşelerinin komşu olmasıdır.

  b) $f(r_1)=r_2$.

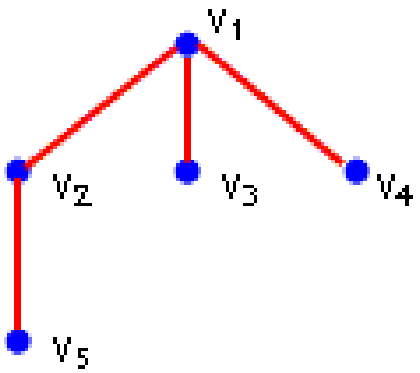
  Bu durumda f fonksiyonuna bir eşyapı dönüşümü denir.



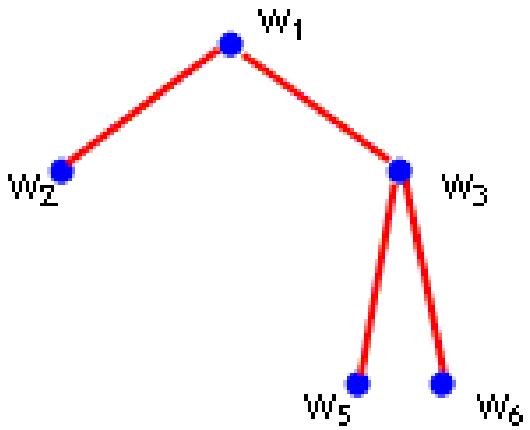
T_1



T_2

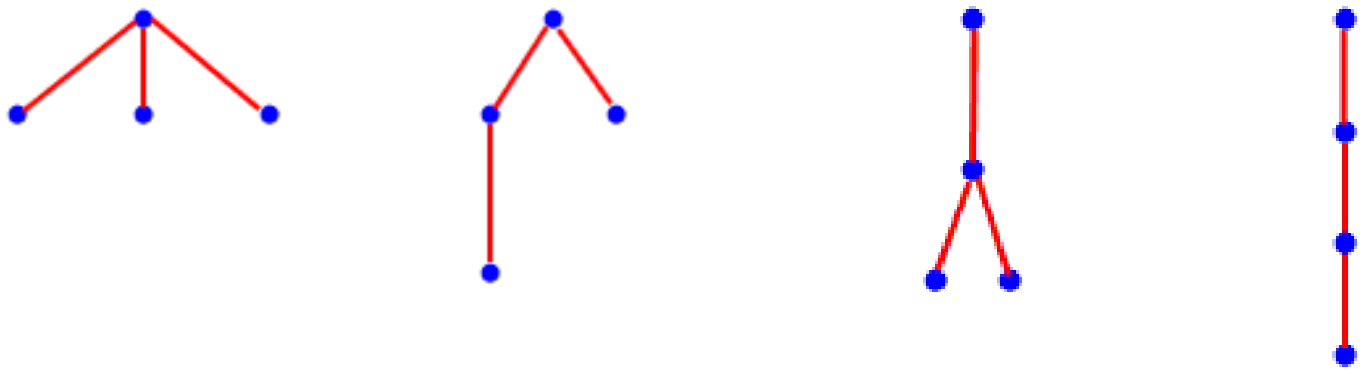


T_1



T_2

- **Teorem:** *Dört adet köşesi olan toplam dört adet eşyapılı olmayan köklü ağaç vardır. Bu dört köklü ağaç aşağıdaki şekilde verilmiştir.*



- **Tanım:** r_1 kökü ile bir ikili ağaç T_1 ve r_2 kökü ile bir ikili ağaç T_2 olsun. Aşağıdaki özellikleri sağlayan T_1 ağacının köşeler kümesinden T_2 ağacının köşeler kümesine bir bire-bir, üzerine f fonksiyonu varsa *T_1 ve T_2 ağaçlarına eşyapılıdırlar* denir:

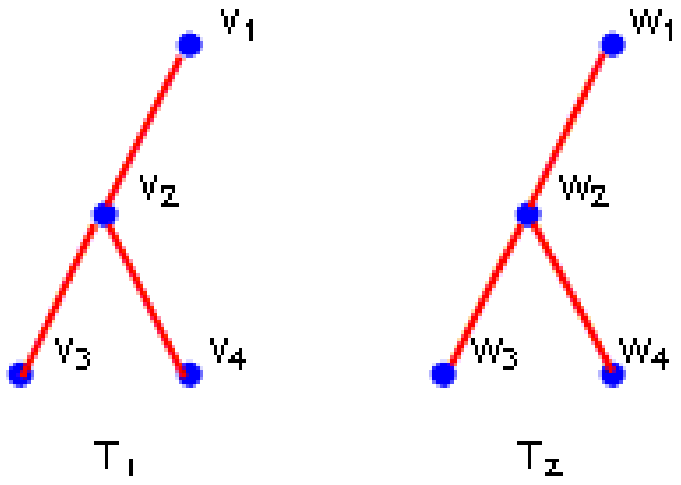
  a) T_1 ağacında v_1 ve v_2 köşelerinin komşu köşe olması için gerekli ve yeterli koşul T_2 ağacında $f(v_1)$ ve $f(v_2)$ köşelerini komşu köşe olmasıdır.

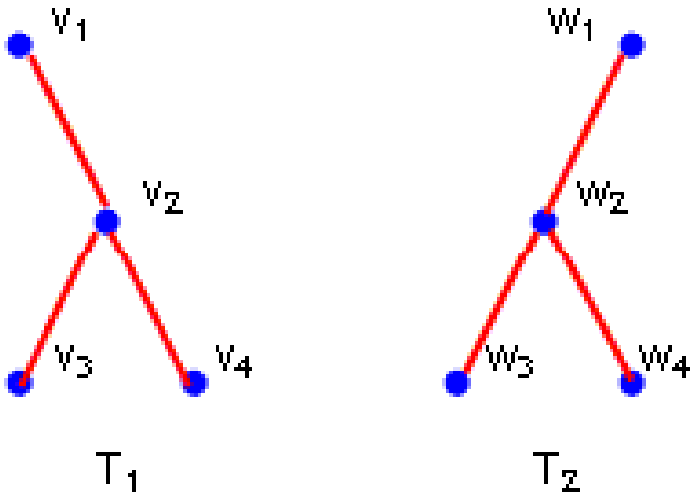
  b) $f(r_1)=r_2$

  c) T_1 ağacında w köşesinin sol altının v köşesi olması için gerekli ve yeterli koşul T_2 ağacında f(v) köşesinin f(w) köşesinin sol altı olmasıdır.

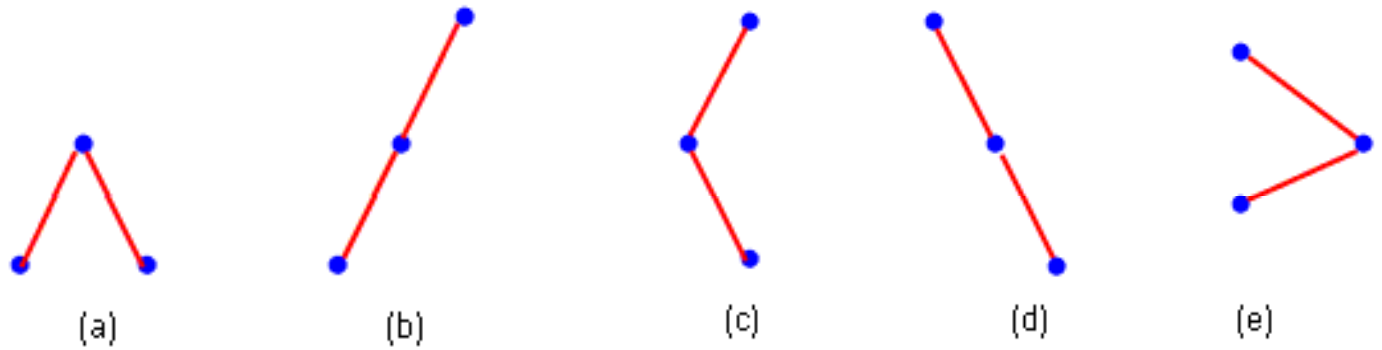
  d) T_1 ağacında w köşesinin sağ altının v köşesi olması için gerekli ve yeterli koşul T_2 ağacında f(v) köşesinin f(w) köşesinin sağ altı olmasıdır.

  f fonksiyonuna bir *eşyapı dönüşümü* denir.


v1
v2
v3
v4
T1
w1
w2
w3
w4
T2


v1
v2
v3
v4
T1
w1
w2
w3
w4
T2

- **Teorem:** *Üç adet köşeye sahip olan beş adet eşyapılı olmayan ikili ağaç vardır. Bu beş adet ikili ağaç aşağıdaki şekilde gösterilmiştir.*



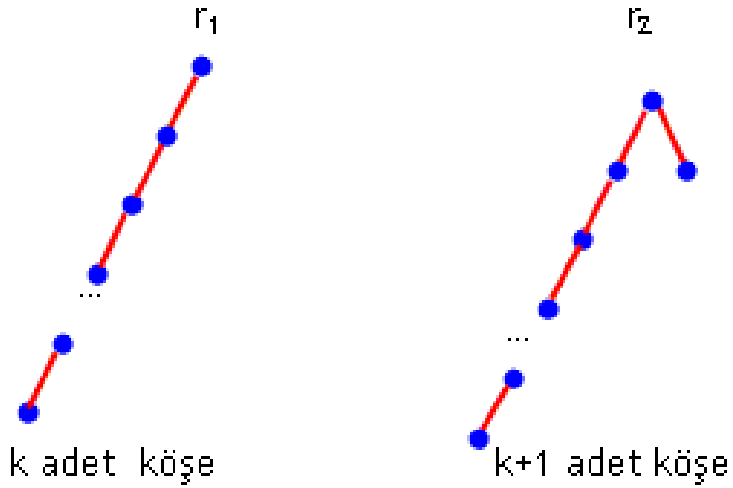
- **Teorem:** *n adet köşeden oluşan toplam C(2n,n)/n+1 sayıda eşyapılı olmayan ikili ağaç vardır.*

Girdi: r_1 ve r_2 kökleriyle iki ikili ağaç verilsin (Eğer ilk ağaç boş ise, bu durumda r_1 null özel değerini alır. Eğer ikinci ağaç boş ise, r_2 null özel değerini alır).

Çıktı: Ağaçlar eşyapılı ise, **true;** değilse, **false**

```
procedure bin_tree_isom(r1,r2)
  if r1=null and r2=null then
      return (true)
           // şimdi artık r1 ya da r2 'den biri null
  değil
  if r1=null or r2=null then
      return (false)
       // şimdi artık r1 ,r2 her ikiside null değil
  lc_r1=r1 'in sol altı
  lc_r2=r2 'in sol altı
  rc_r1=r1 'in sağ altı
  rc_r2=r2 'in sağ altı
  return (bin_tree_isom(lc_r1,lc_r2)
        and bin_tree_isom(rc_r1,rc_r2)
end bin_tree_isom
```

- **Teorem:** *İki ağaç içinde bulunan tüm köşelerin sayısı n olmak üzere bir önceki bin_tree_isom algoritması en kötü çalışma zaman* $\Theta(n)$ *ile verilir.*

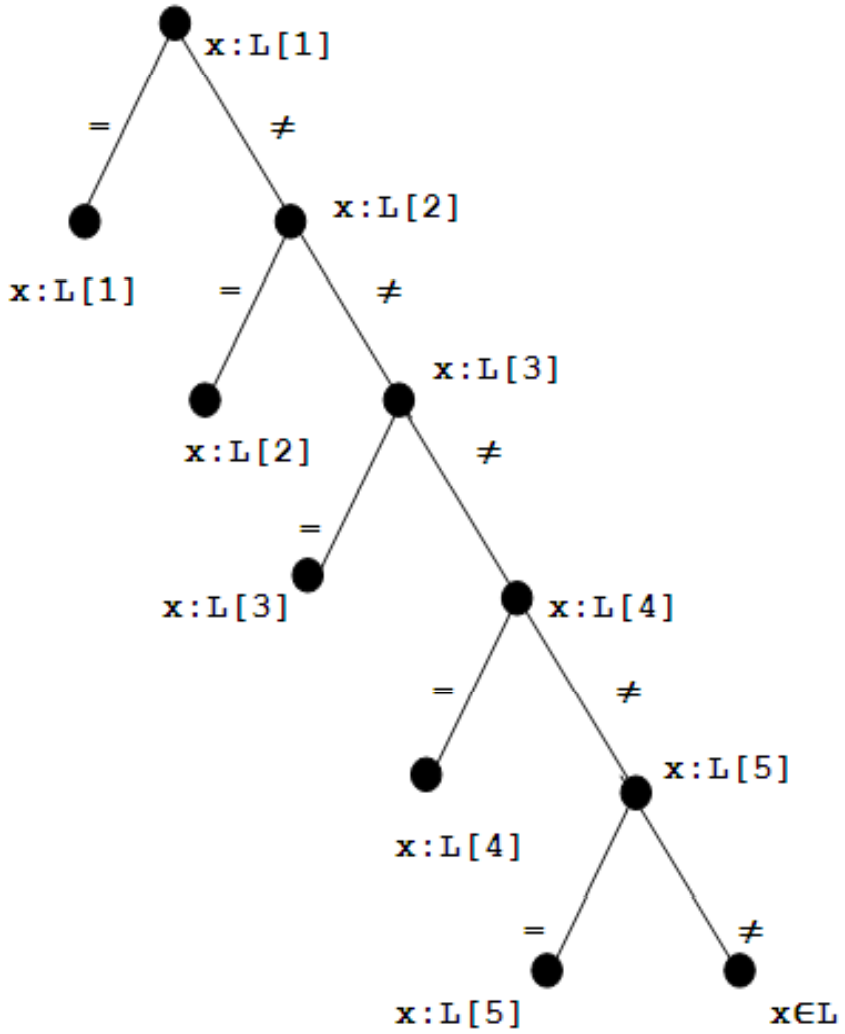


KARAR AĞAÇLARI

TANIM: İç düğümlerinin bir eylemi ifade ettiği, ağaçtaki dallanma yönlerinin eylemin çıktısını ifade ettiği ve yaprakların sonuç çıktığı ifade ettiği ağaçlara *karar ağaçları* denir.

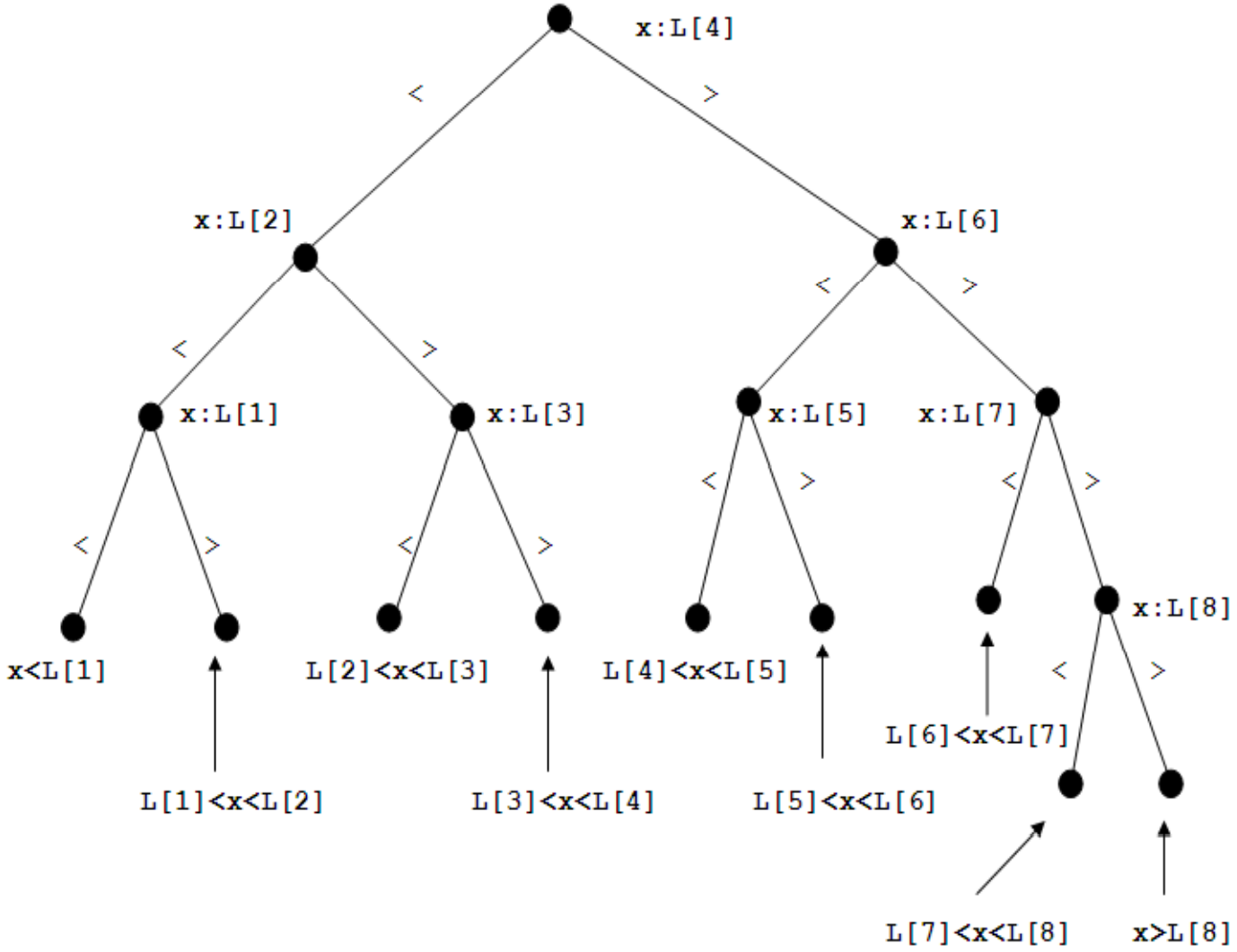
Dizisel Arama Algoritması

```
SequentialSearch(list L; integer n; itemtype x)
// searches a list L of n items for item x
integer i
    i=1
    while (L[i]≠x ve i<n) do
        i=i+1
    end while
    if (L[i]=x ) then
        write "found"
    else
        write "not found"
    end if
end function
```

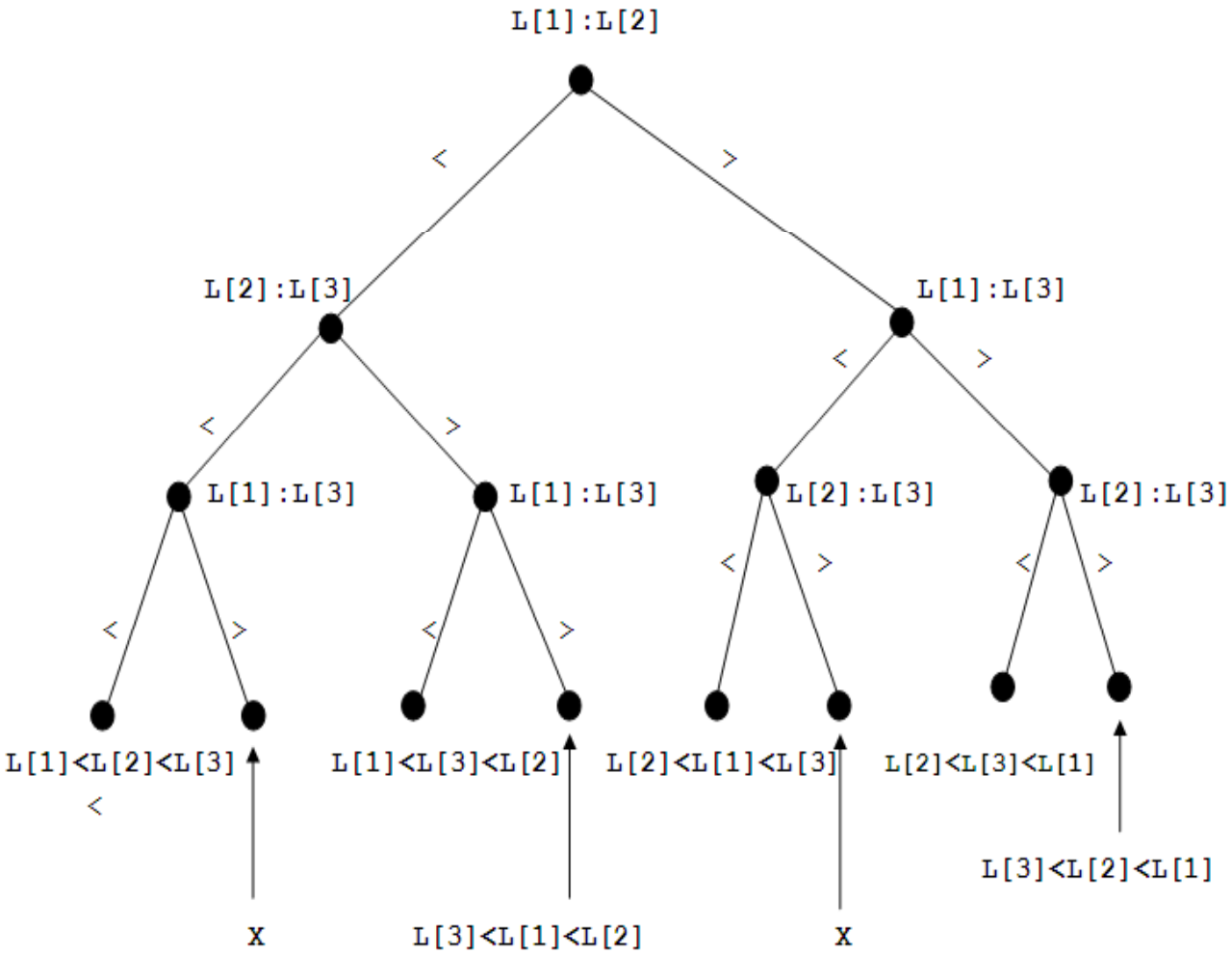

x:L[1]
=
≠
x:L[1]
x:L[2]
=
≠
x:L[2]
x:L[3]
≠
=
x:L[3]
x:L[4]
=
≠
x:L[4]
x:L[5]
=
≠
x:L[5]
x∈L

İkili Arama Algoritması

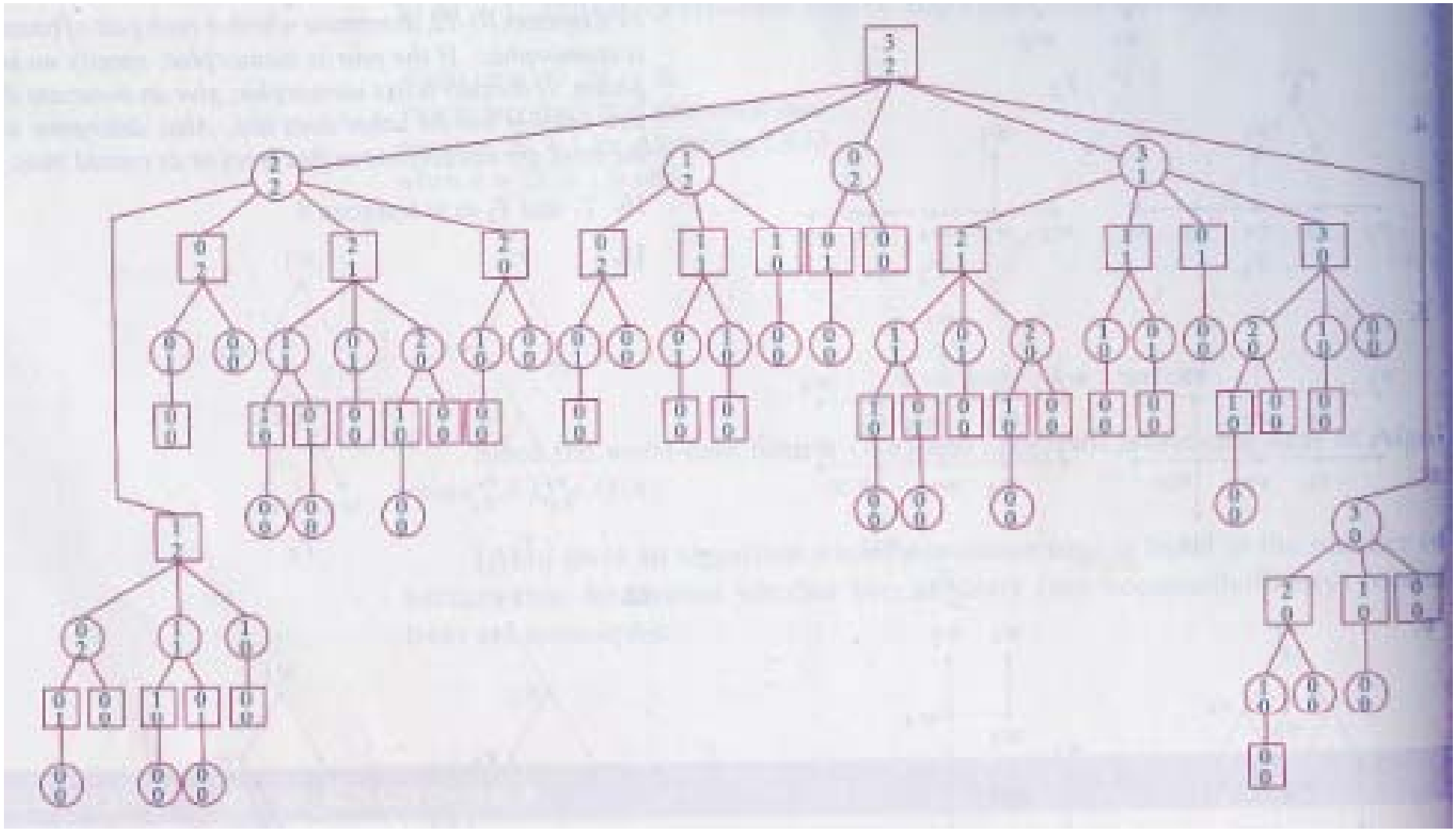
```
BinarySearch(list L; integer i; integer j; itemtype x)
// searches sorted list L from L[i] to L[j] for item x
    if i> j then
        write "not found"
    else
        find the index k of the middle item in the list L[i]-L[j]
        if (x = middle item) then
                write "found"
        else
                if (x< middle item) then
                        BinarySearch(L,i,k-1,x)
                else
                        BinarySearch(L,k+1,j,x)
                end if
        end if
    end if
end function
```

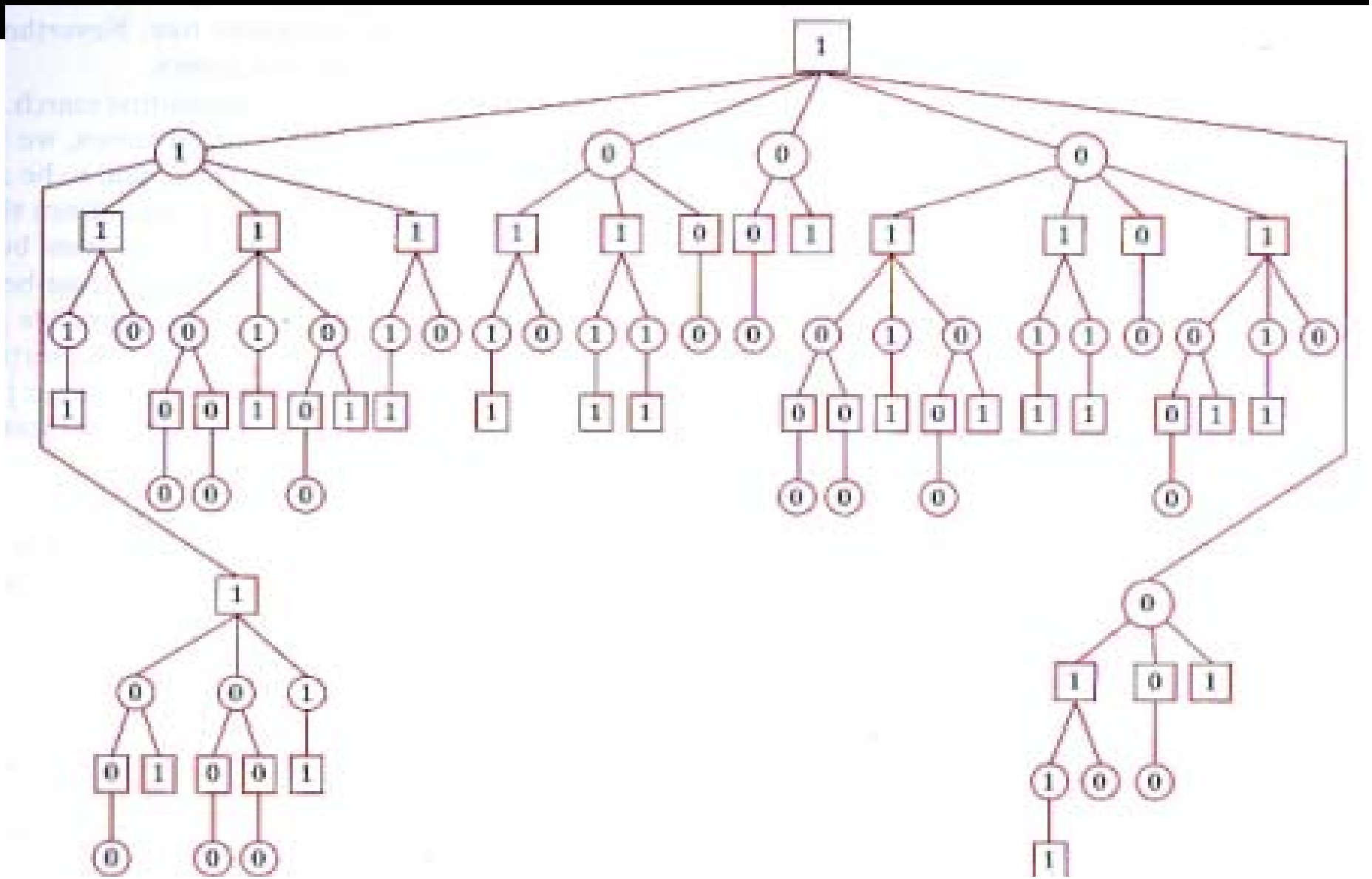

x:L[4]
<
>
x:L[2]
x:L[6]
<
>
<
>
x:L[1]
x:L[3]
x:L[5]
x:L[7]
<
>
<
>
<
>
<
>
x:L[8]
x<L[1]
L[2]<x<L[3]
L[4]<x<L[5]
<
>
L[6]<x<L[7]
L[1]<x<L[2]
L[3]<x<L[4]
L[5]<x<L[6]
L[7]<x<L[8]
x>L[8]

Sıralama Amaçlı



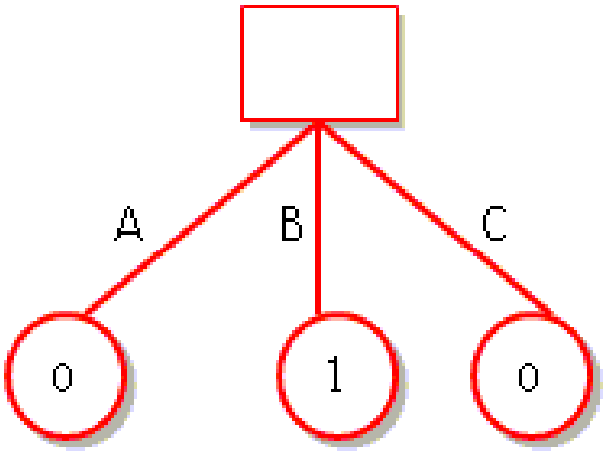
Oyun Ağaçları

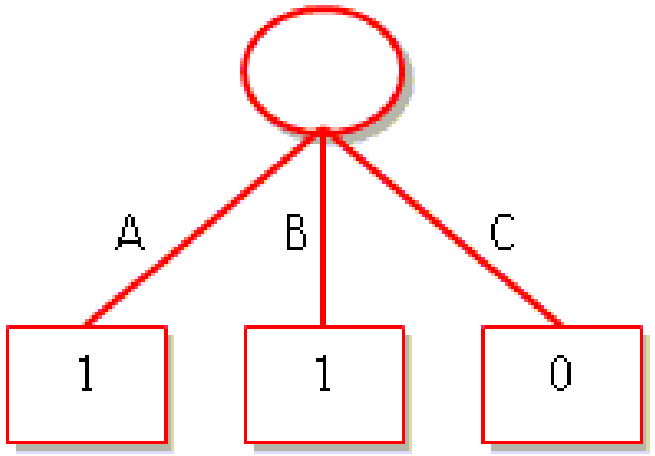


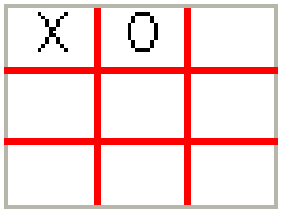


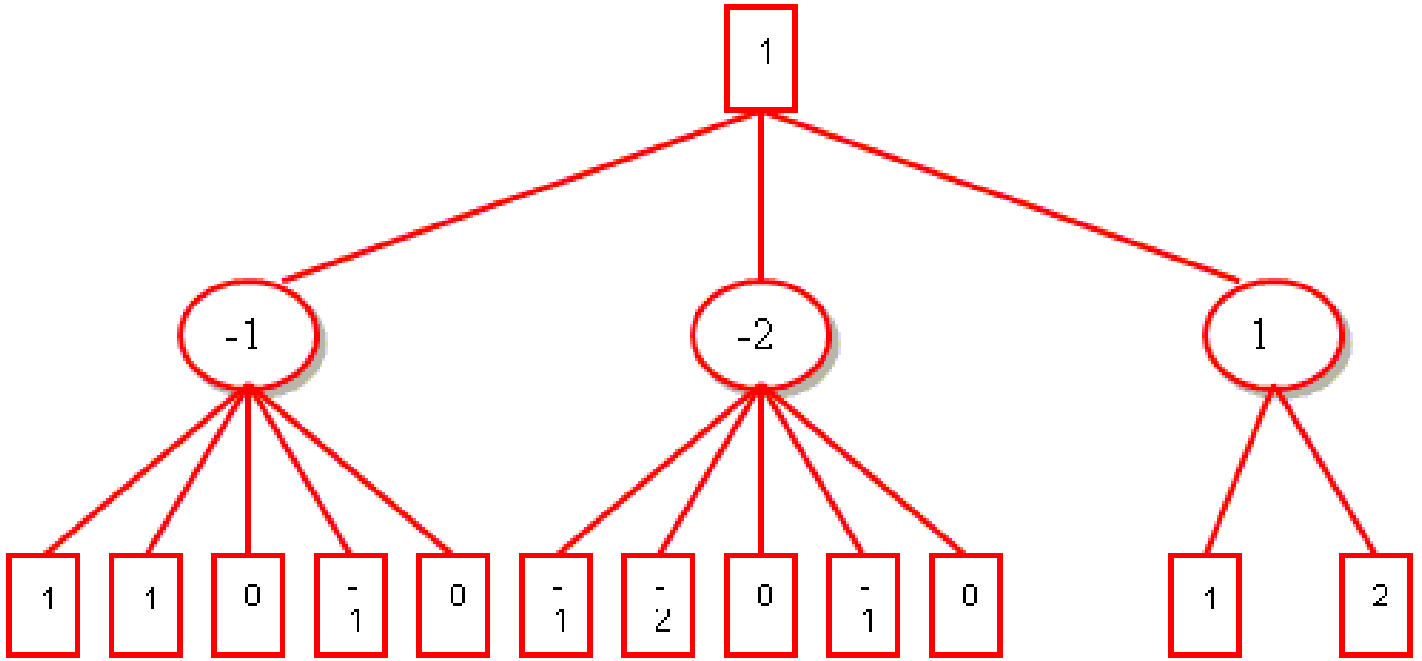
Oyun Ağaçları

- Ağaçlar oyun-geliştirme stratejilerinde kullanılabilir.

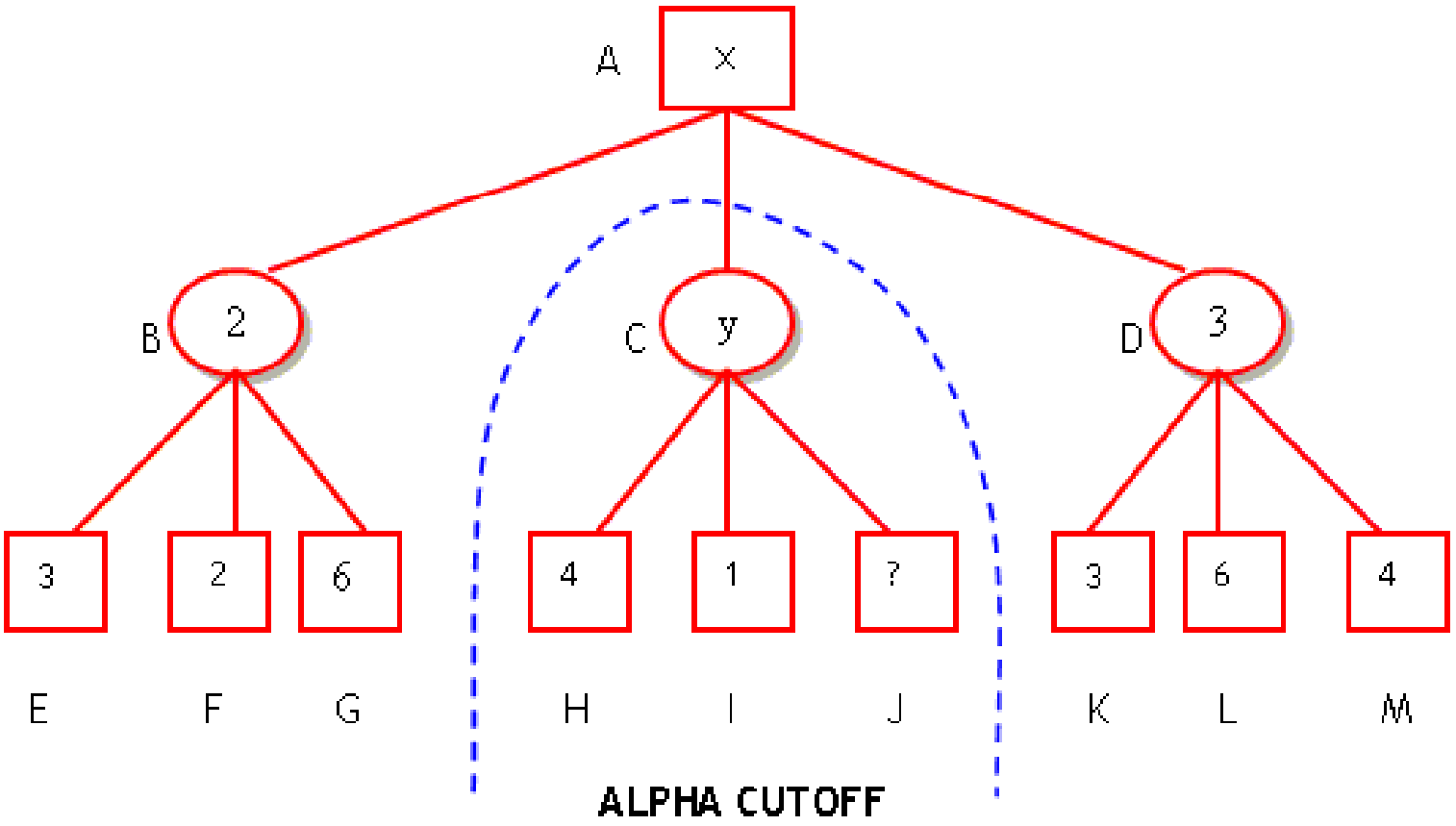


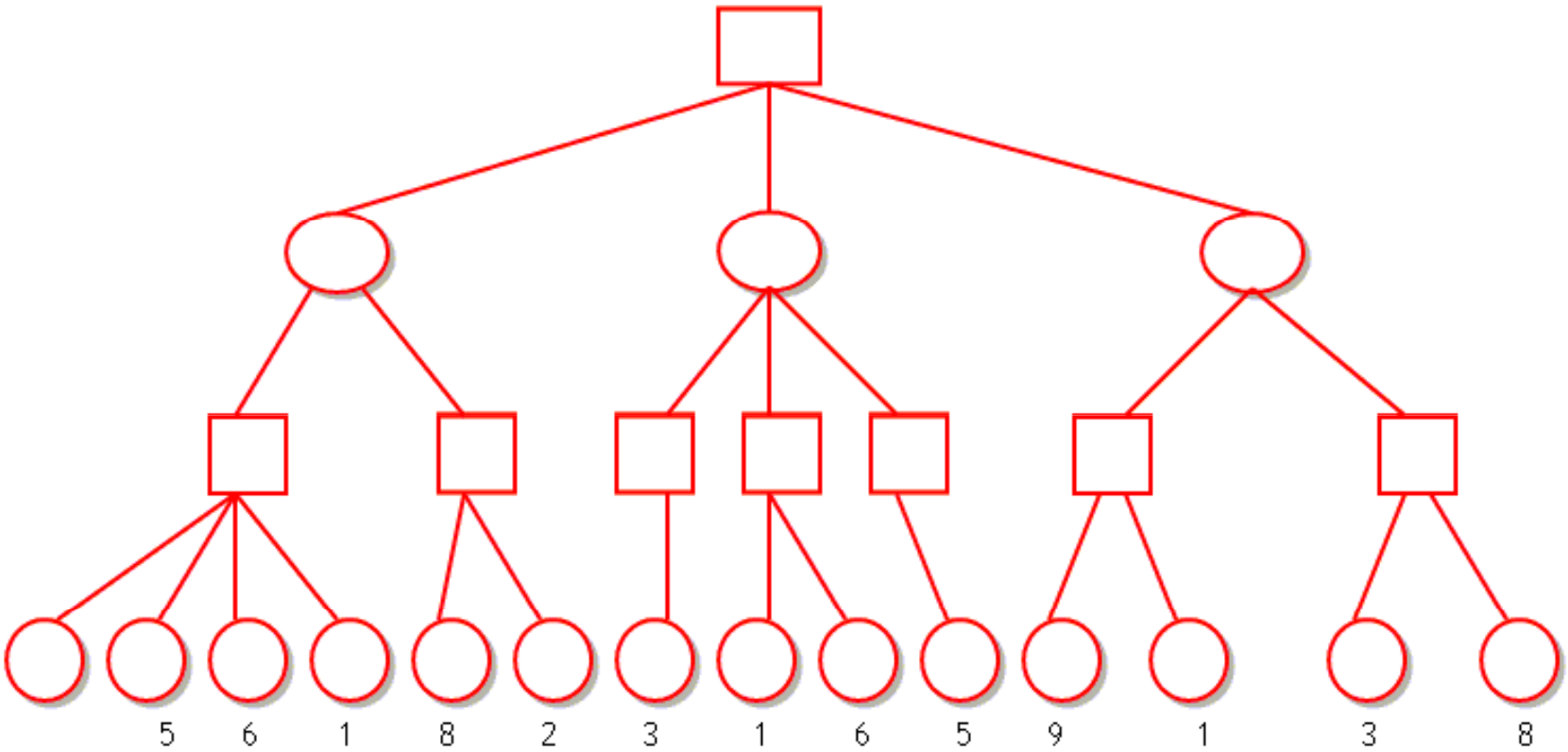


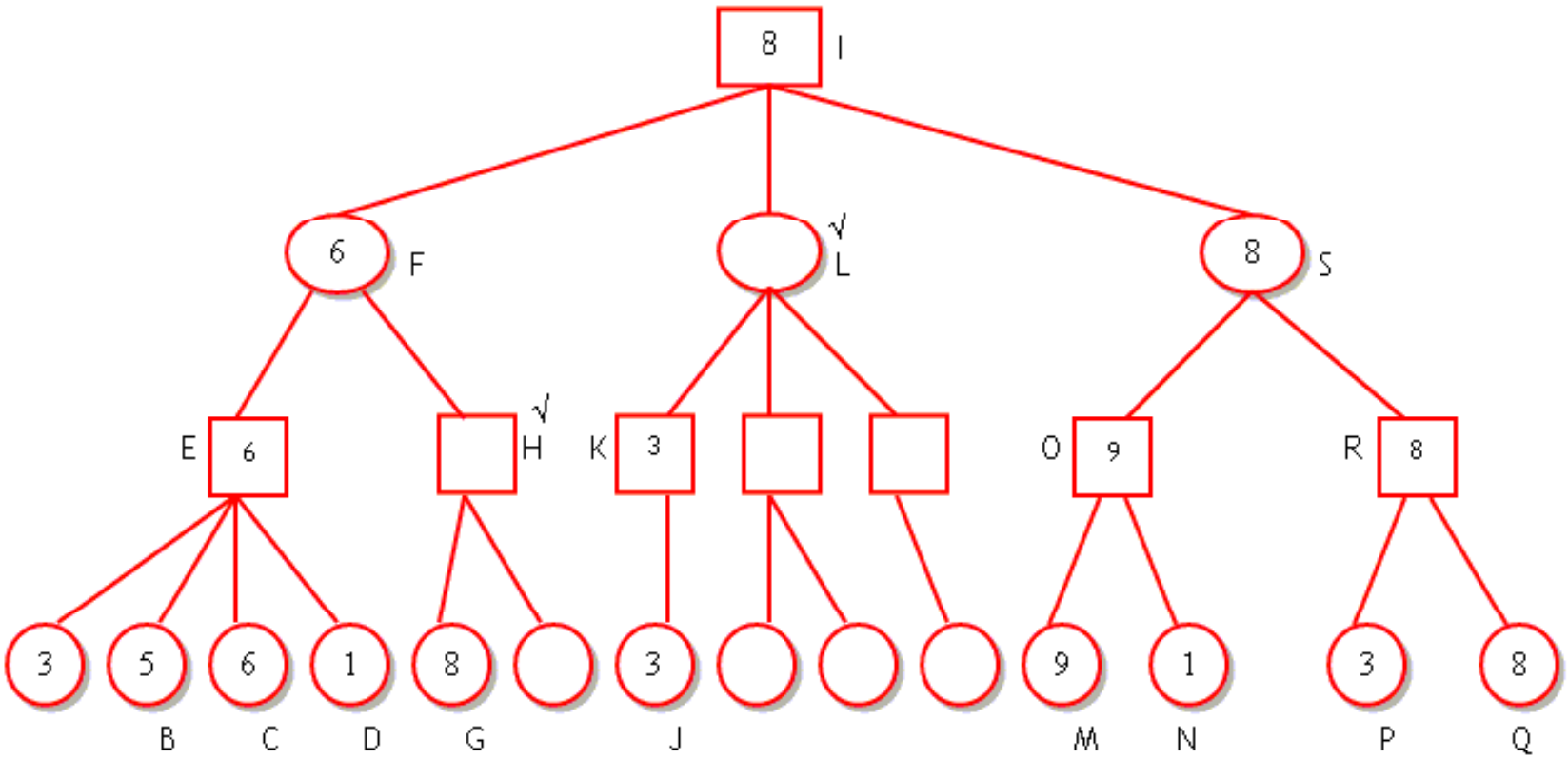

X
O


1
-1
-2
1
1
1
0
-1
0
-1
-2
0
-1
0
1
2

- Bir oyun ağacının değerlendirilmesinde ya da bir oyun ağacının bir kısmının değerlendirilmesinde zaman-tüketimi azaltmak önemlidir.
- Zaman tüketimini azaltan tekniklerden biri *alpha-beta budaması* tekniğidir. Genel olarak alpha-beta budamasında oyun ağacında bir çok köşe atlanır.


A
X
B
2
C
y
D
3
3
2
6
4
1
?
3
6
4
E
F
G
H
I
J
K
L
M
ALPHA CUTOFF


5 6 1 8 2 3 1 6 5 9 1 3 8


8
I
6
F
√
L
8
S
E
6
√
H
K
3
O
9
R
8
3
5
6
1
8
3
9
1
3
8
B
C
D
G
J
M
N
P
Q