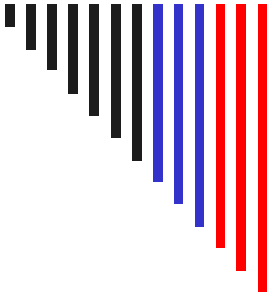
BLM212 Veri Yapıları

Introduction to Trees

2021-2022 Güz Dönemi



Introduction to Trees

Hedefler

- **Temel ağaç terminolojisi ve kavramlarını öğrenmek**
- **İkili ağacın temel özelliklerini tanımlamak ve tanımak.**
- ***Depth-first* ve *breadth-first* dolaşma stratejilerini kullanarak ağaçları işlemek**
- **İkili ağaç kullanarak ifadeleri parse etmek**
- **Huffman ağaçları tasarlamak ve gerçekleştirmek**
- **Genel ağaçların temel kullanımını ve işlenmesini kavramak**

History

- On dokuzuncu yüzyılın ortalarında, Gustav Kirchhoff matematikte ağaçlar üzerinde çalıştı.
- Birkaç yıl sonra, Arthur Cayley bunları cebirsel formüllerin yapısını incelemek için kullandı.
- 1951 yılında, Grace Hopper'ın onları aritmetik ifadeleri temsil etmek için kullanması.
- Hopper'ın çalışması, günümüzün ikili ağaç biçimleriyle güçlü bir benzerlik taşımaktadır.

Uygulama/Kullanım Alanları

- Bilgisayar Bilimlerinde ağaçlar;
  - Cebirsel formülleri ifade etmek,
  - Büyük dinamik listelerde arama yapmak için etkili bir yöntem olarak,
  - Yapay zeka sistemleri gibi çeşitli uygulamalar için,
  - Ve kodlama (encoding) algoritmalarında

kullanılmktadır.

Temel Ağaç Kavramları

- Bir ağaç (tree),
  - **düğüm** (node) adı verilen sonlu elemanlar kümesinden ve
  - **dal** (branch) adı verilen düğümleri birbirine bağlayan sonlu yönlendirilmiş çizgiler kümesinden oluşur.
- Bir düğümle ilişkili dalların sayısı, düğümün **derecesidir** (degree).

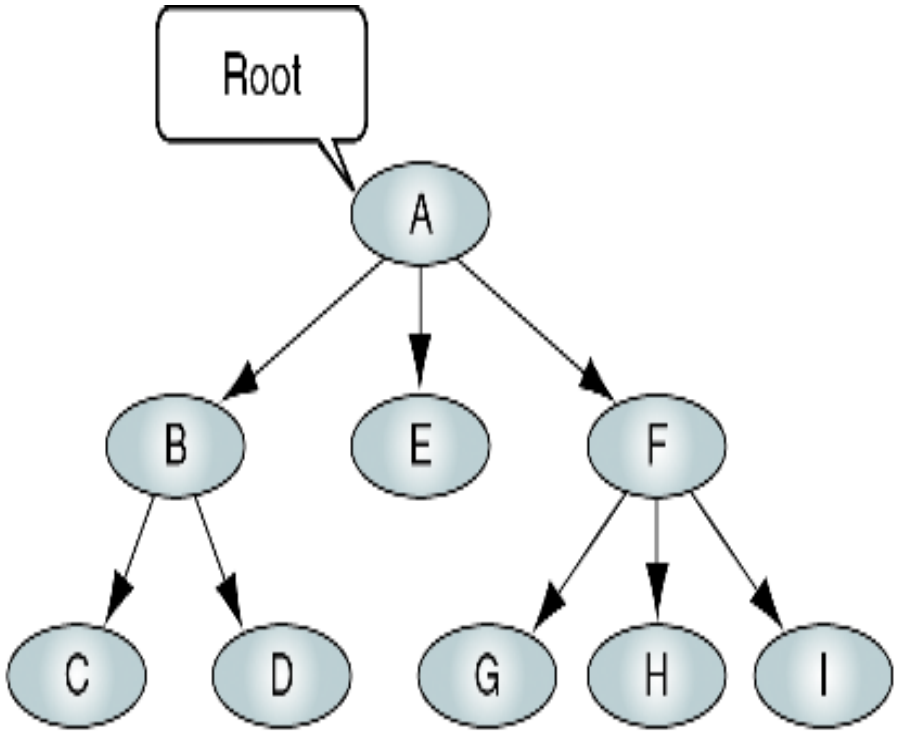


FIGURE 6-1 Tree

Temel Ağaç Kavramları

- Dal düğüme doğru yönlendirildiğinde, bu "indegree branch" tır.
- Dal düğümden dışarıya doğru yönlendirildiğinde, bu "outdegree branch" tır.
- indegree ve outdegree dalların toplamı düğümün derecesini gösterir.
- Ağaç boş değilse, ilk düğüme kök (root) adı verilir.

Temel Ağaç Kavramları

- Kök düğümün indegree'si tanım gereği sıfırdır.
- Kök hariç, bir ağaçtaki tüm düğümlerin tam olarak bir tane indegree'ye sahip olması gerekir; yani, yalnızca bir **selefi** (predecessor) olabilir.
- Ağaçtaki tüm düğümler, kendinden ayrılan sıfır, bir veya daha fazla dal içerebilir; yani sıfır, bir veya daha fazla outdegre'ye sahip olabilirler.

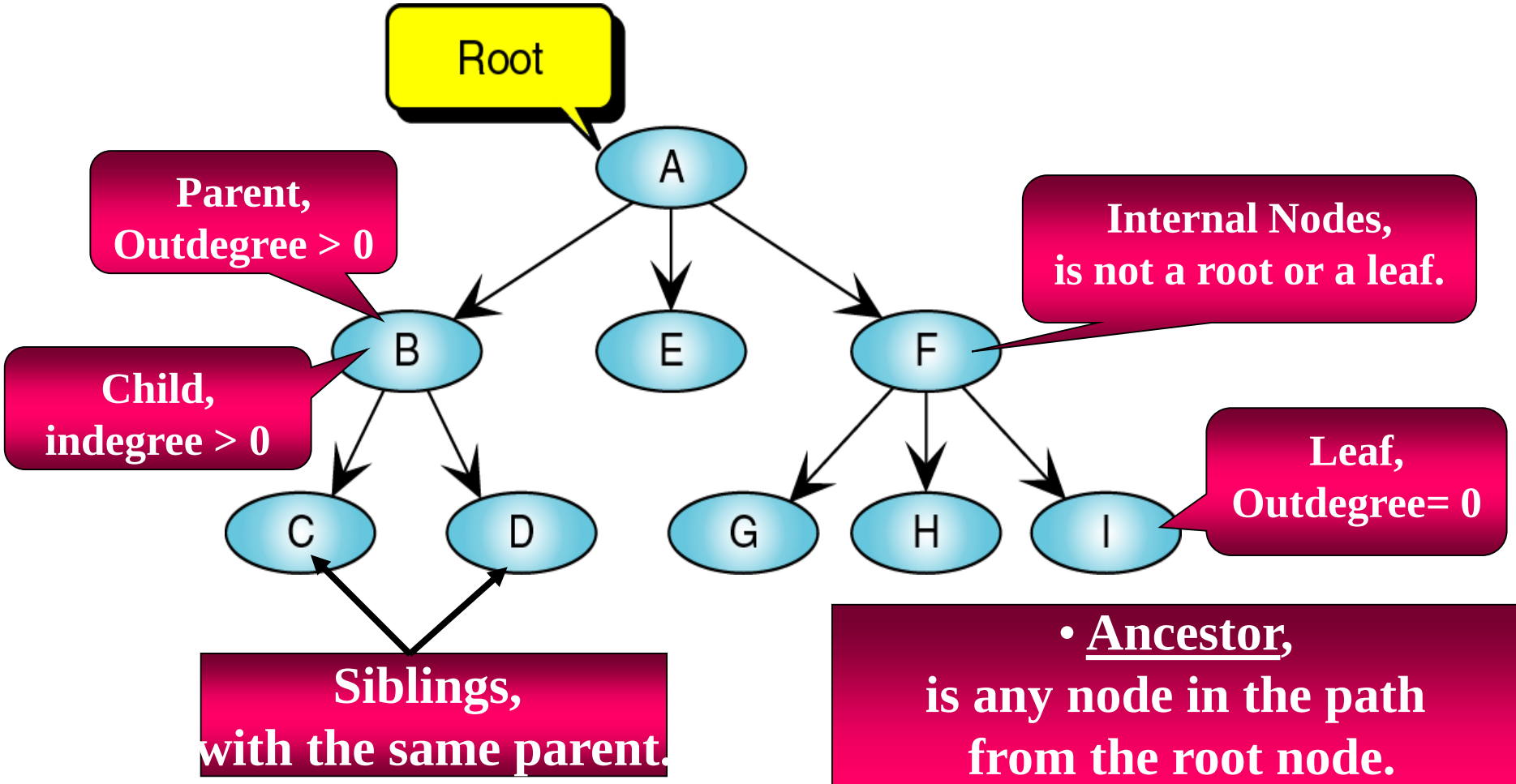
Temel Ağaç Kavramları

- **Yaprak** (leaf), outdegree'si sıfır olan düğümdür. Yani halefi (successor) olmayan bir düğümdür.
- Kök veya yaprak olmayan bir düğüm, **iç** (internal) düğüm olarak bilinir.
- Eğer bir düğümün halef düğümleri varsa **ebeveyndir** (parent). Yani sıfırdan büyük outdegree'ye sahiptir.
- Selefi olan bir düğüme **çocuk** (child) denir.

Temel Ağaç Kavramları

- Ebeveynleri aynı olan iki veya daha fazla düğüme **kardeş** (siblings) adı verilir.
- **Ata** (ancestor), kökten düğüme giden yoldaki herhangi bir düğümdür.
- **Soy** (descendant), ebeveyn düğümün altındaki yoldaki herhangi bir düğümdür.
  - Yani, belirli bir düğümden bir yaprağa giden yoldaki tüm düğümler, o düğümün soyundan gelir.

Temel Ağaç Kavramları



Temel Ağaç Kavramları

- Yol (path), her bir düğümün bir sonraki düğüme bitişik olduğu bir düğüm dizisidir.

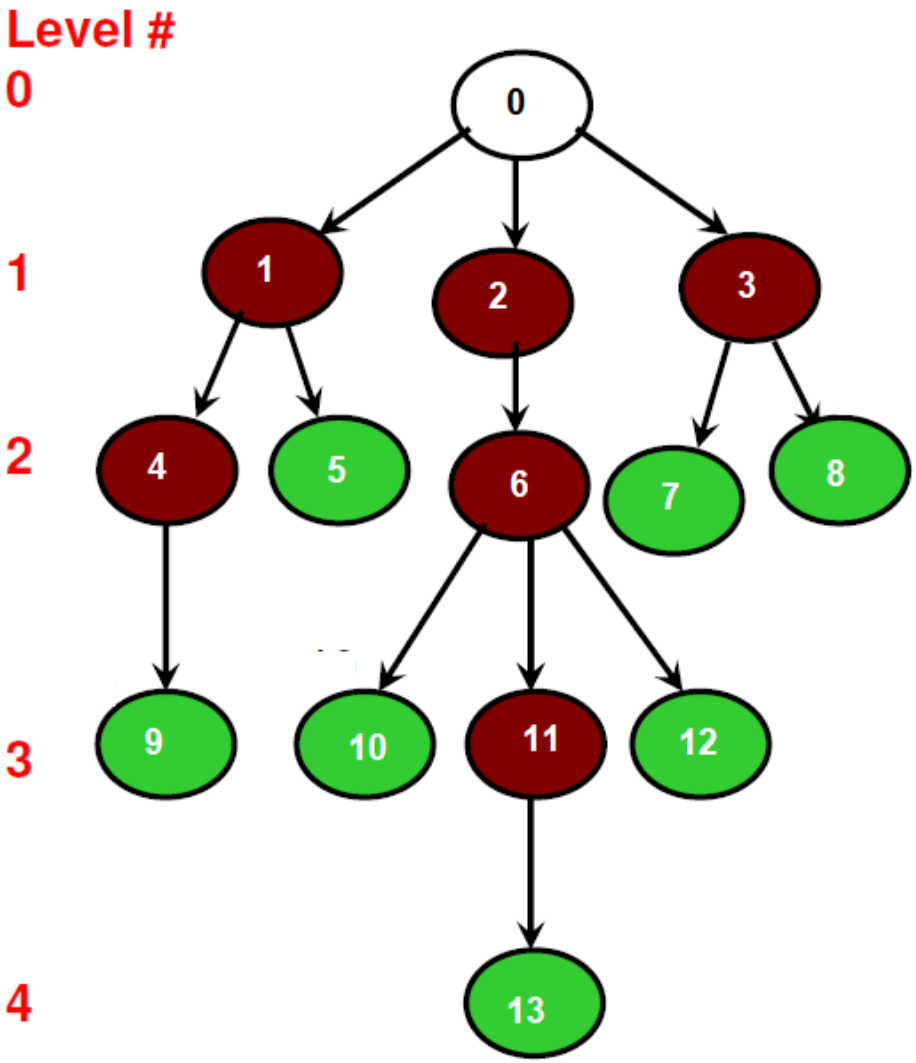
- Bir düğümün seviyesi (level), kökten olan uzaklığıdır.
  - Kök 0 seviyesinde,
  - Kökün çocukları 1 seviyesinde, vb.

Temel Ağaç Kavramları

- Bir düğümün derinliği (**depth** of a node), kökten o düğüme kadar olan kenarların sayısıdır.

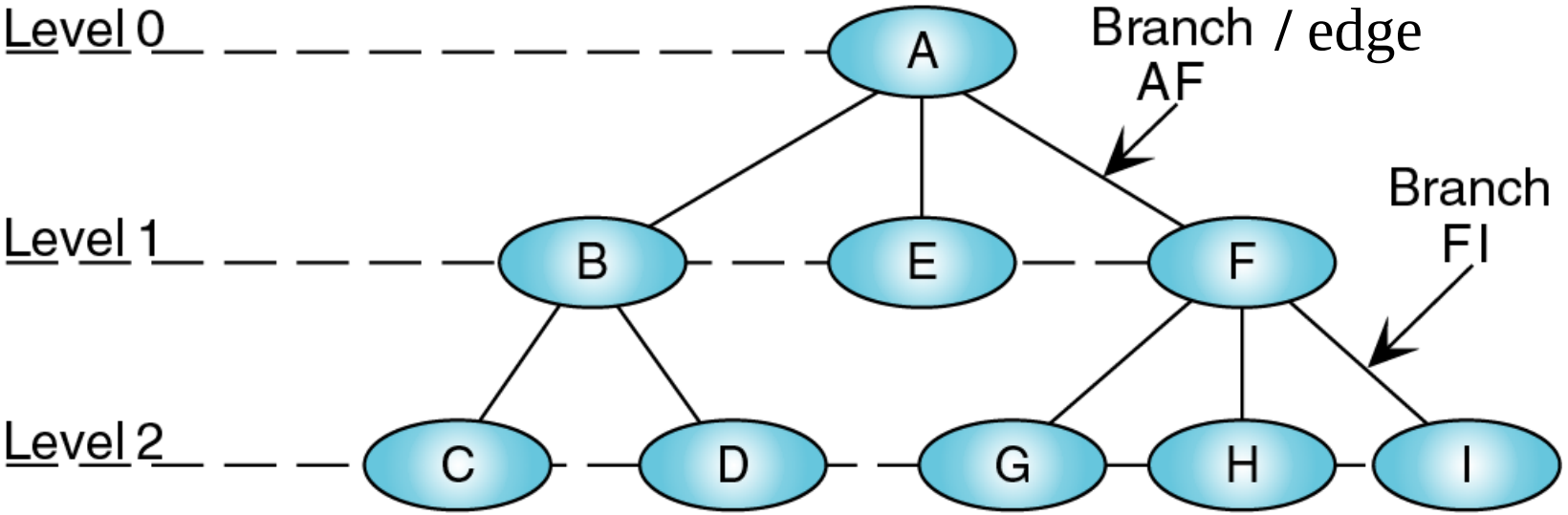
- Bir düğümün yüksekliği (**height** of a node), düğümden en derin yaprağa kadar olan kenarların sayısıdır.

Example: Height vs. Depth



Node	Depth (Level)	Height
0	0	4
1	1	2
2	1	3
3	1	1
4	2	1
5	2	0
6	2	2
7	2	0
8	2	0
9	3	0
10	3	0
11	3	1
12	3	0
13	4	0

Temel Ağaç Kavramları



Parents: A, B, F
Children: B, E, F, C, D, G, H, I
Siblings: {B,E,F}, {C,D}, {G,H,I}

Leaves C,D,E,G,H,I
Internal nodes B,F

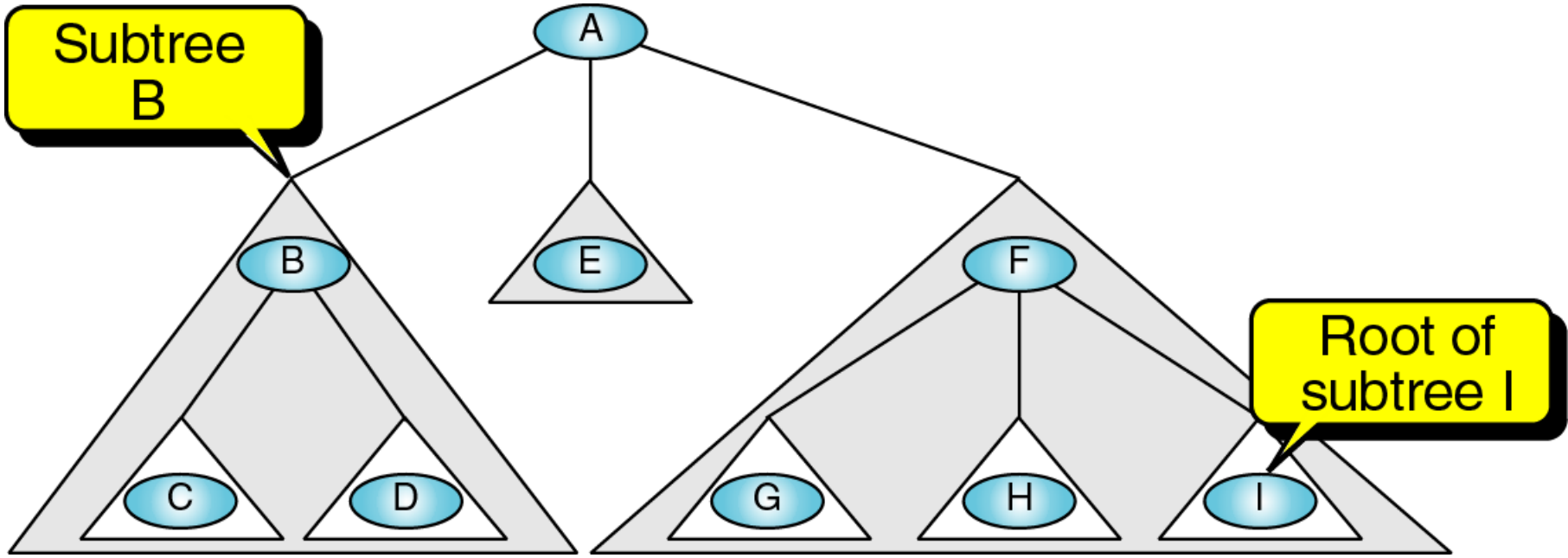
Height of tree = max. Level of leaf + 1

Ağacın boyu/yüksekliği (height), kökten en uzaktaki yaprağın seviyesi artı 1'dir.

- **Tanım gereği boş ağacın boyu -1'dir**

Temel Ağaç Kavramları

- Bir alt ağaç (subtree), kökün altındaki herhangi bir bağlı yapıdır.
- Alt ağaçtaki ilk düğüm alt ağaç kökü olarak bilinir.
- Bir alt ağaç, başka alt ağaçlara bölünebilir.



Bir ağacın özyinelemeli tanımı

- Ağaç, şunlardan birine uyan bir dizi düğümdür:
  - boş veya
  - kök olarak adlandırılan düğüme sahiptir ve bu düğümden hiyerarşik olarak sıfır veya daha fazla alt ağaca inen ağaçlardan oluşur. Ayrıca bu alt ağaçlar da ayrı ağaçlardır.

Ağaç gösterimi/temsili

Ağaçlar için üç farklı kullanıcı gösterimi vardır:

- **General Tree** – organizasyon şeması formatı
- **Indented list** – Bir parça listesinin bir öğenin montaj yapısını temsil ettiği malzeme liste sistemi
- **Parenthetical Listing**

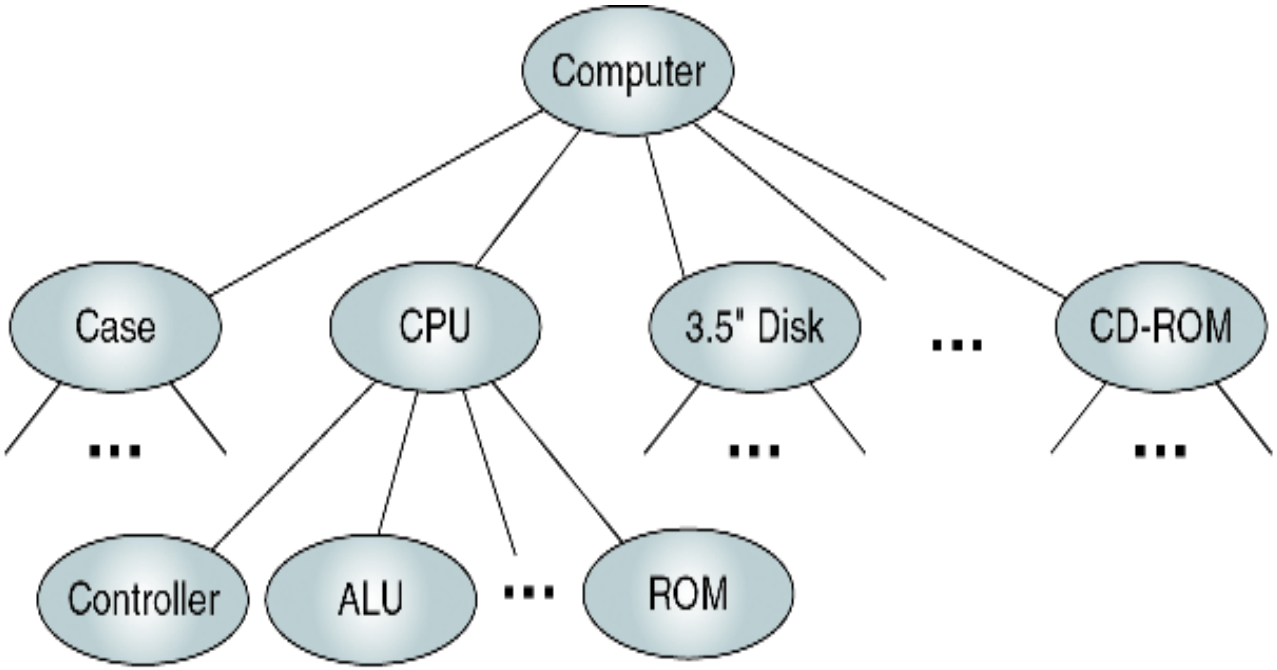


FIGURE 6-4 Computer Parts List as a General Tree

Indented list

Part number	Description
301	Computer
301-1	Case
...	...
301-2	CPU
301-2-1	Controller
301-2-2	ALU
...	...
301-2-9	ROM
301-3	3.5" Disk
...	...
301-9	CD-ROM
...	...

TABLE 6-1 Computer Bill of Materials

Parenthetical Listing

- **Parenthetical Listing**
  - Bu format cebirsel ifadelerde kullanılır.
  - Bir ağaç parantez notasyonuyla gösterildiğinde
    - Her **açma parantezi** yeni bir seviyenin başlangıcını belirtir ve
    - her **kapama parantezi** geçerli seviyeyi tamamlar ve ağaçta bir seviye yukarı çıkar.

Parenthetical Listing

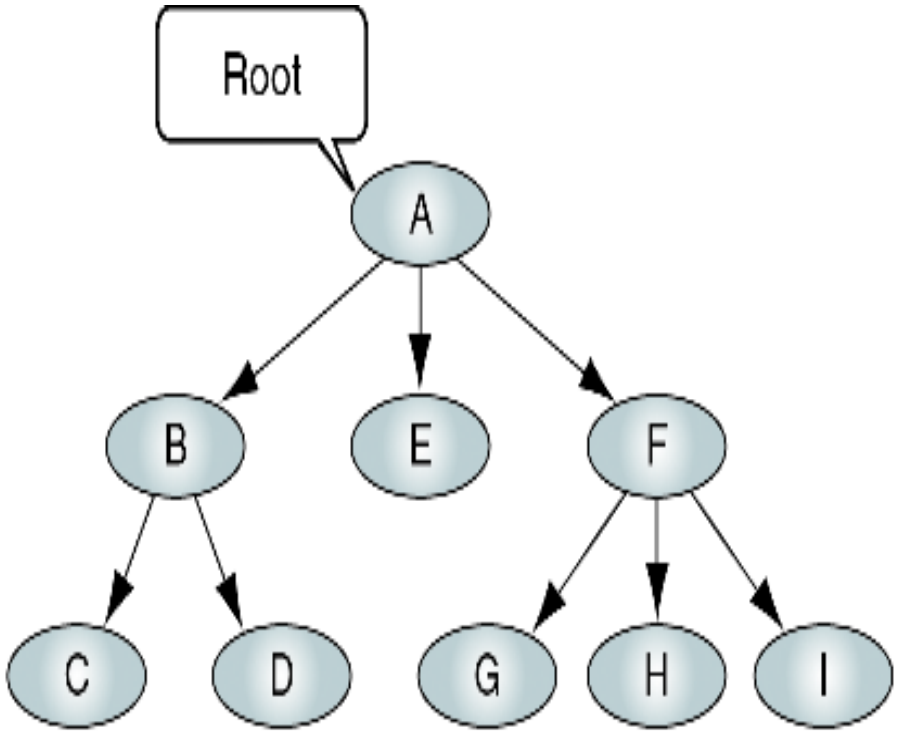
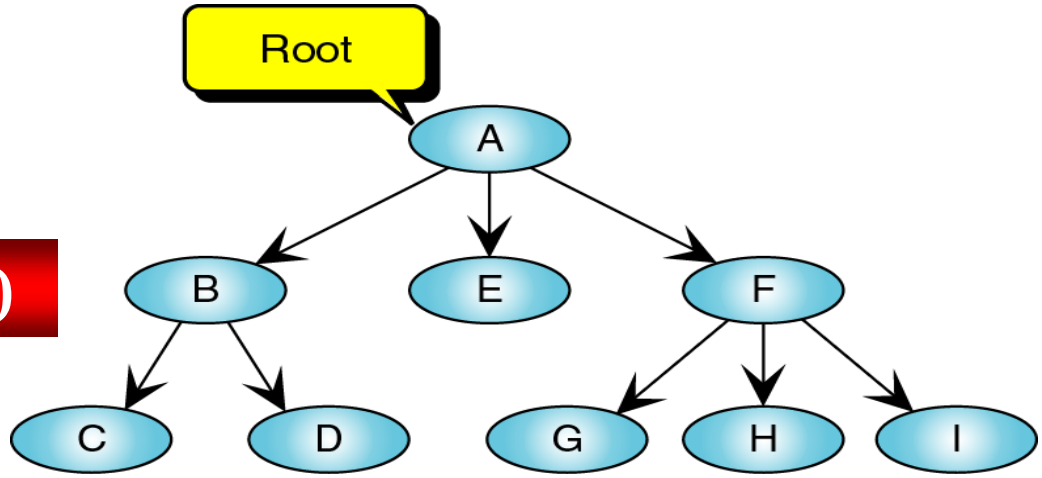


FIGURE 6-1 Tree

A (B (C D) E F (G H I))

Root (B (C D) E F (G H I))



algorithm **ConvertToParen**(val root <node pointer>, ref output <string>)
Convert a general tree to parenthetical notation.
Pre root is a pointer to a tree node.
Post output contains parenthetical notation.

1. Place root in output
2. If (root is parent)
    1. Place an open parenthesis in the output
    2. **ConvertToParen**(root's first child)
    3. While (more siblings)
        1. **ConvertToParen**(root's next child)
    4. Place close parenthesis in the output
3. Return

End **ConvertToParen**

6-2 Binary Trees (*İkili Ağaçlar*)

Bir ikili ağaç (binary tree) ikiden fazla soya sahip olamaz.

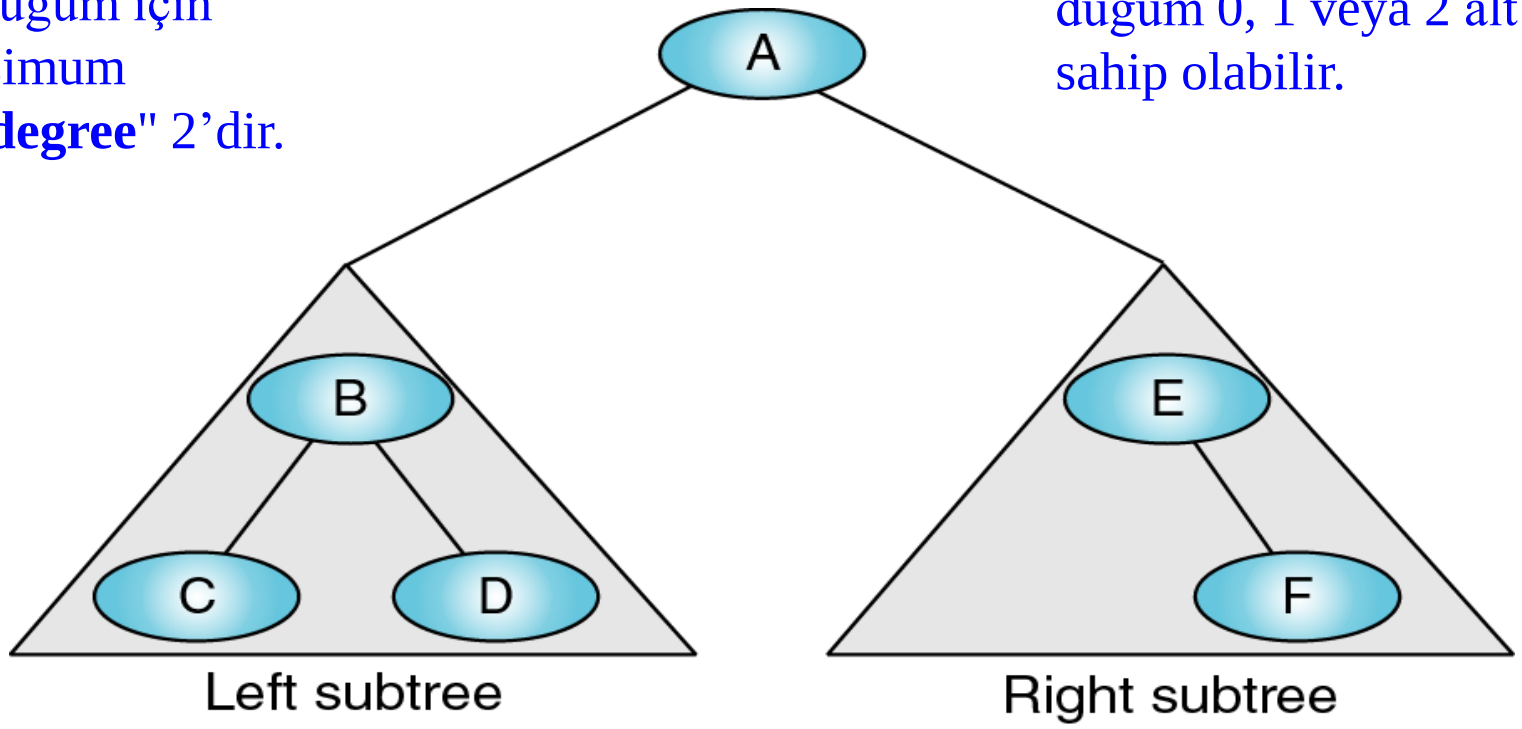
- **Properties**
- **Binary Tree Traversals**
- **Expression Trees**
- **Huffman Code**

Binary Trees

İkili ağaç (binary tree), hiçbir düğümün ikiden fazla alt ağaca sahip olamayacağı bir ağaçtır.

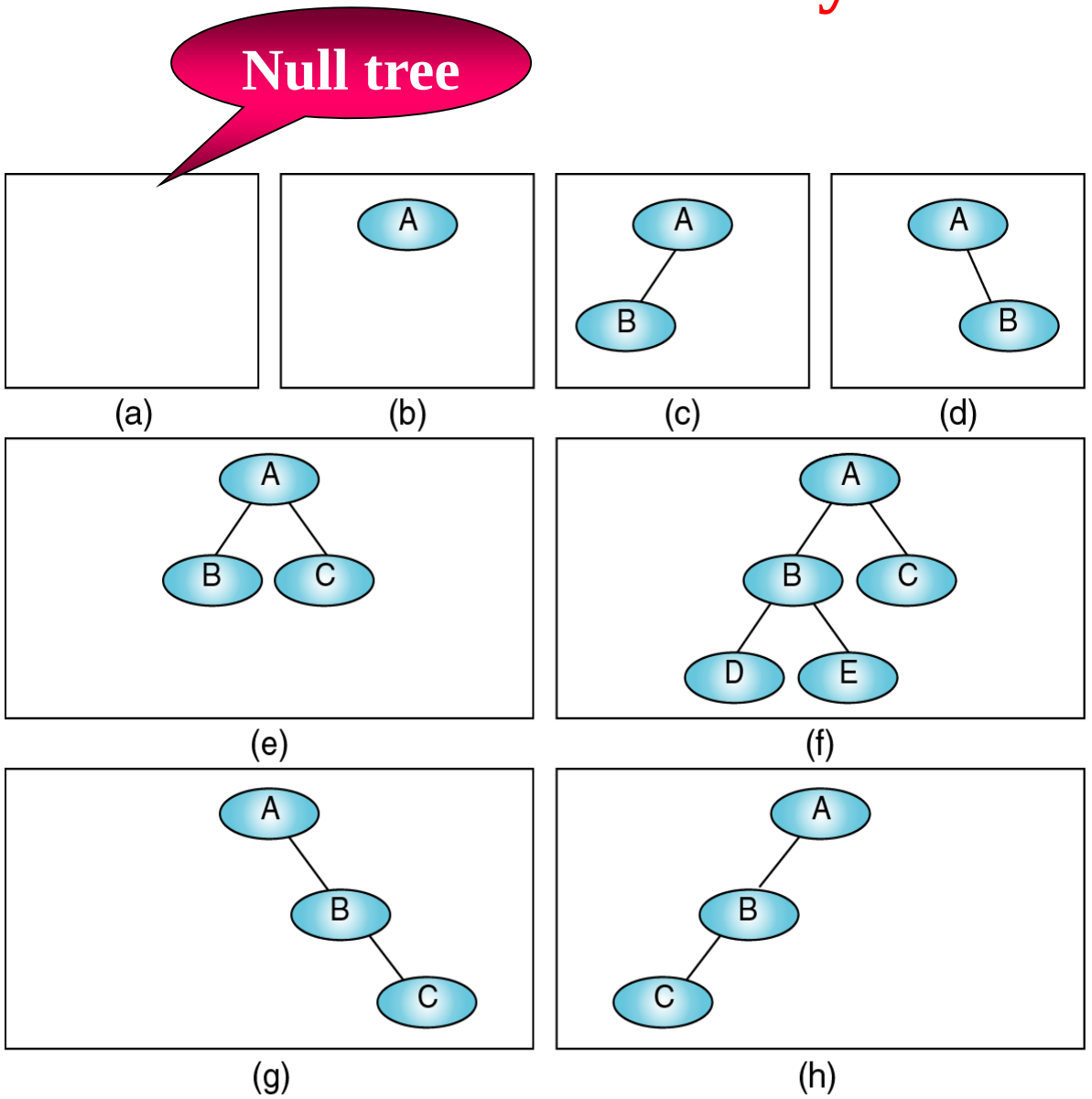
Bir düğüm için maksimum "**outdegree**" 2'dir.

Diğer bir ifadeyle bir düğüm 0, 1 veya 2 alt ağaca sahip olabilir.



Bu alt ağaçlar sol alt ağaç ve sağ alt ağaç olarak adlandırılır.

Binary Trees



Symmetry is not a tree requirement!

İkili Ağaçların Bazı Özellikleri

- İkili ağaçların boyu/yüksekliği matematiksel olarak tahmin edilebilir
- *N* adet düğümü bir ikili ağaçta tutmamız gerektiği durumda, azami yükseklik (maximum height) :

$$H_{\max} = N$$

Azami yüksekliğe sahip bir ağaç nadirdir. Ağaçtaki tüm düğümlerin yalnızca bir halefi olması durumunda oluşur.

İkili Ağaçların Bazı Özellikleri

- İkili bir ağacın minimum yüksekliği (minimum height) aşağıdaki şekilde belirlenir:

$$H_{\min} = \lceil \log_2 N \rceil + 1$$

Örneğin, ikili ağaçta tutulacak üç düğüm varsa (N = 3), o zaman $H_{\min}$ = 2.

İkili Ağaçların Bazı Özellikleri

- İkili ağacın yüksekliği H göz önüne alındığında, ağaçtaki minimum düğüm sayısı (minimum number of nodes) :

$$N_{\min} = H$$

İkili Ağaçların Bazı Özellikleri

- Maksimum düğüm sayısı için formül, her düğümün sadece iki soya sahip olabileceği gerçeğinden türetilmiştir. İkili ağacın yüksekliği H göz önüne alındığında, ağaçtaki maksimum düğüm sayısı (minimum number of nodes) :

$$N_{\max} = 2^{H} - 1$$

İkili Ağaçların Bazı Özellikleri (Toplu)

- Maximum height of tree for N nodes:

$$H_{max} = N$$

- The minimum height of the tree :

$$H_{min} = [\log_2 N] + 1$$

- If known the height of a tree:

$$N_{min} = H$$

$$N_{max} = 2^H - 1$$

İkili Ağaçların Bazı Özellikleri

- Bir ağaçtaki herhangi bir düğümün çocuklarına, istenen düğüme çıkan yalnızca bir dal yolu izlenerek erişilebilir.

- Kökün çocuğu olan 1. seviyedeki düğümlere yalnızca bir dal izleyerek erişilebilir; Bir ağacın 2. seviyesindeki düğümlere kökten sadece iki dal izleyerek erişilebilir.

- İkili bir ağacın denge faktörü (balance factor), sol ve sağ alt ağaçların arasındaki yükseklik farkıdır:

$$B = H_L - H_R$$

Balance of the tree

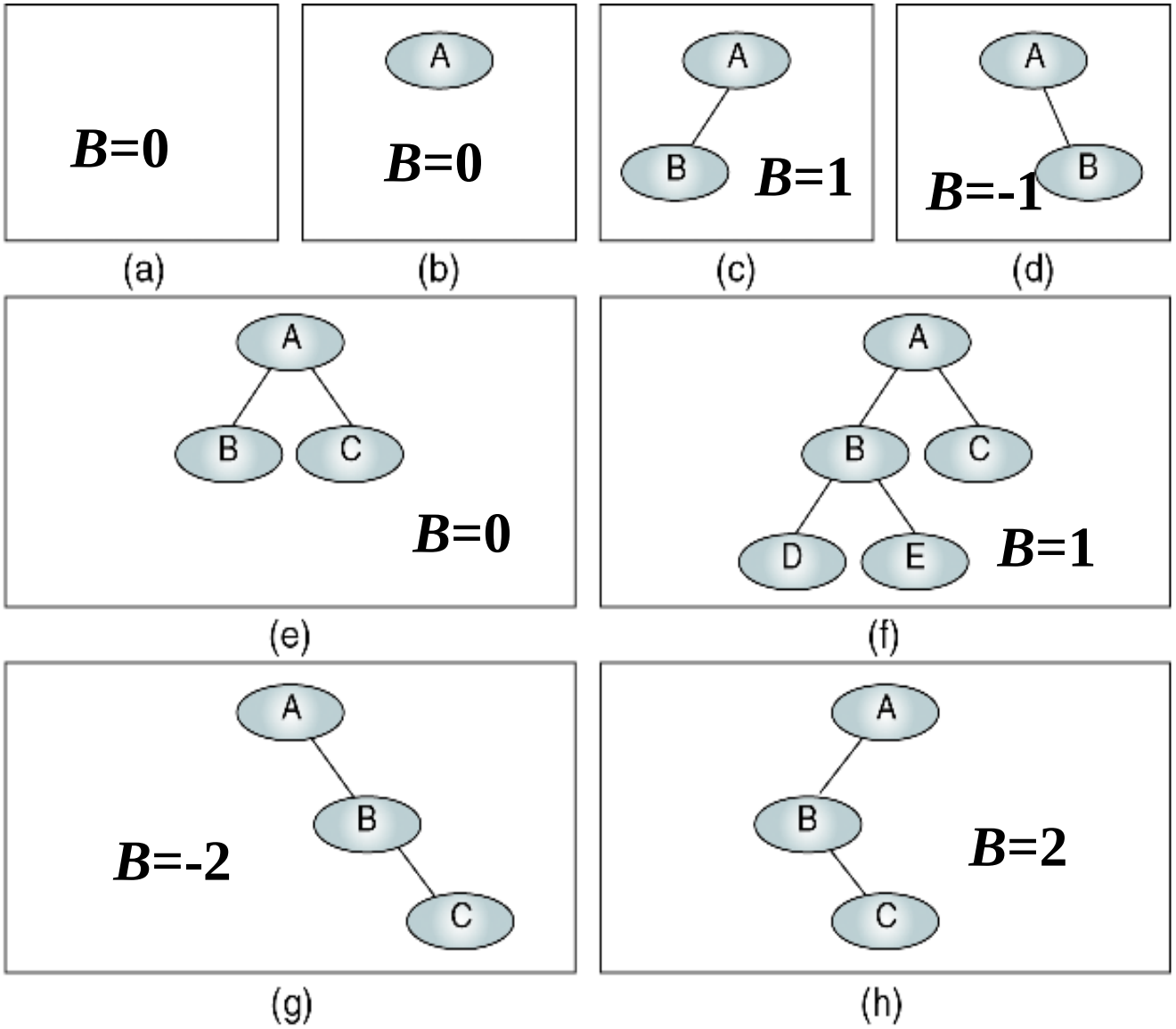


FIGURE 6-6 Collection of Binary Trees

İkili Ağaçların Bazı Özellikleri

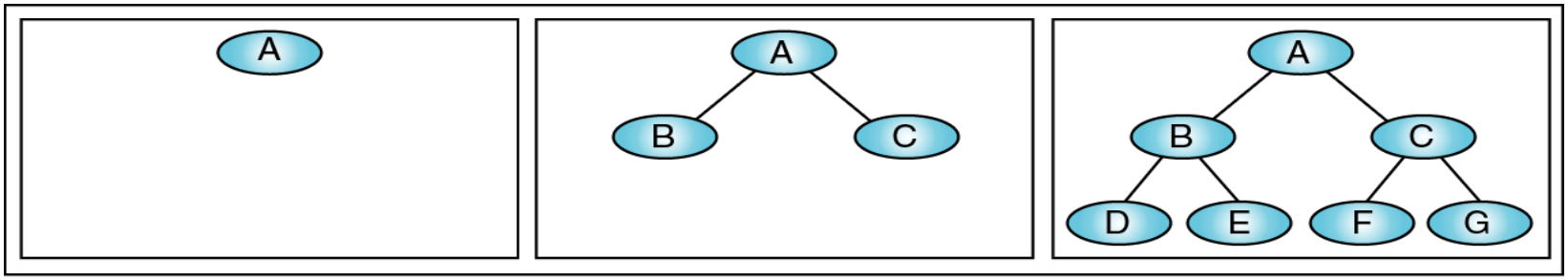
- Dengeli ikili ağaçta (balanced binary tree) (Rus matematikçiler Adelson-Velskii ve Landis'in tanımı) alt ağaçlarının yükseklik farkı birden fazla değildir (denge faktörü -1, 0 veya 1'dir) ve alt ağaçları da dengelidir.

Complete and nearly complete binary trees

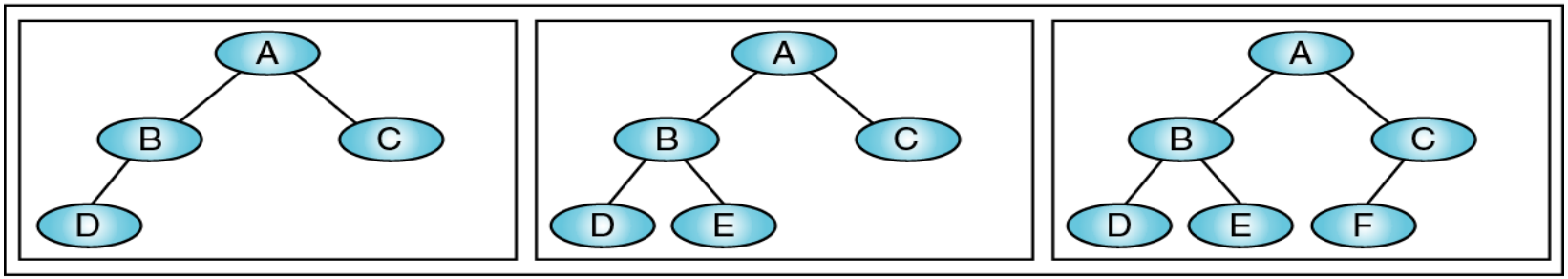
- Bir complete tree, sahip olduğu yüksekliği için maksimum sayıda düğüme sahiptir. Son seviye dolduğunda maksimum sayıya ulaşılır.

- Bir ağaç, düğümleri için minimum yüksekliğe sahipse ve son seviyedeki tüm düğümler solda bulunursa, nearly complete sayılır.

Binary Trees - Balance



(a) Complete trees (at levels 0, 1, and 2)



(b) Nearly complete trees (at level 2)

- A complete tree has the maximum number of entries for its height $N_{max} = 2^H - 1$
- The distance of a node from the root determines how efficiently it can be located.
  - The "balance factor" show that the balance of the tree

$$B = H_L - H_R$$

- If B = 0, 1 or -1; the tree is balanced.

Binary Tree Structure

- Yapıdaki her düğüm, saklanacak veriyi ve biri *sol alt ağaç* diğeri *sağ alt ağaç* olan iki işaretçiyi içermelidir.

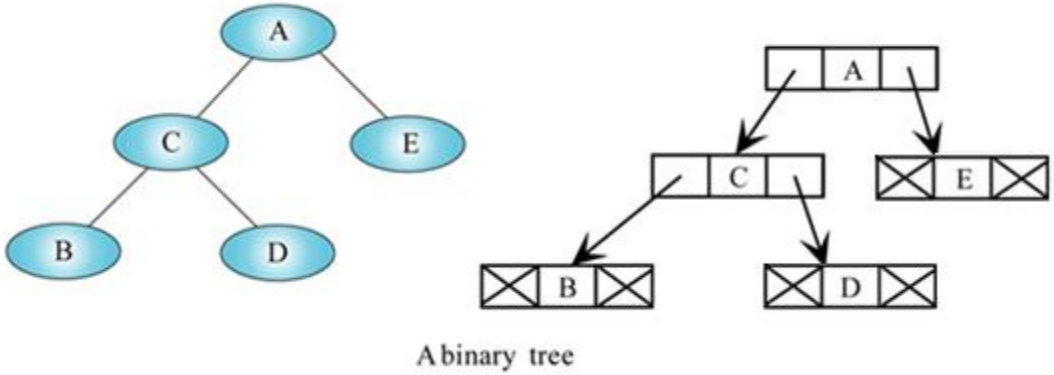
Node

leftSubTree <pointer to Node>

data <dataType>

rightSubTree <pointer to Node>

End Node



A binary tree

Binary Tree Traversal

- binary tree traversal, ağacın her bir düğümünün önceden belirlenmiş bir sırayla bir kez ve sadece bir kez işlenmesini gerektirir.
- depth-first traversal processing
  - ikinci bir çocuğu işlemeden önce kökün ilk çocuğu üzerinden o ilk çocuğun en uzak soyundan bir yol boyunca işlenir.

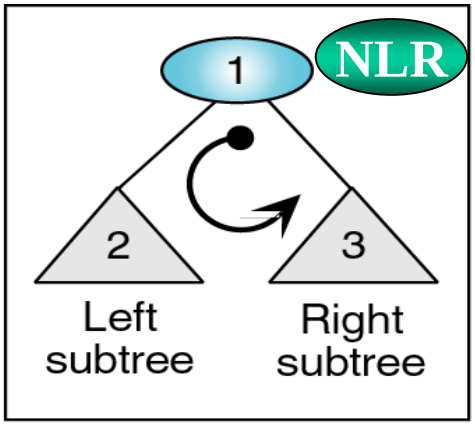
Binary Tree Traversal

- **binary tree traversal:** **ağacın her düğümünün bir kez işlenmesini gerektirir.**

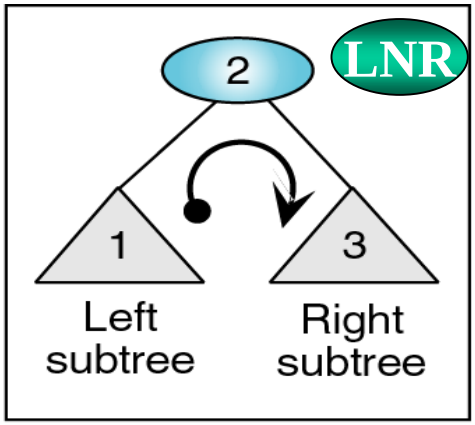
- **depth-first traversal** **: ikinci çocuk işlenmeden önce bir çocuğun soylarının hepsi işlenir.**

- **breadth-first traversal**: **her seviye bir sonraki seviyeye başlamadan önce tamamen işlenir.**

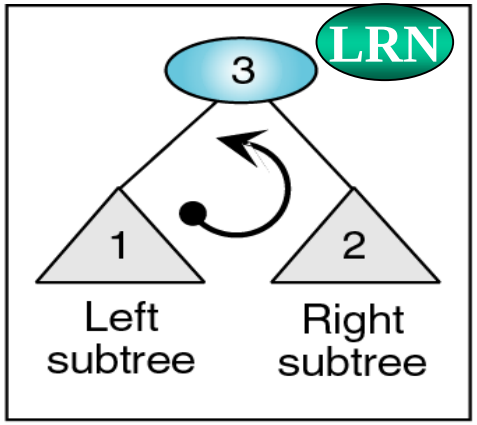
Binary Tree Traversal



(a) Preorder traversal



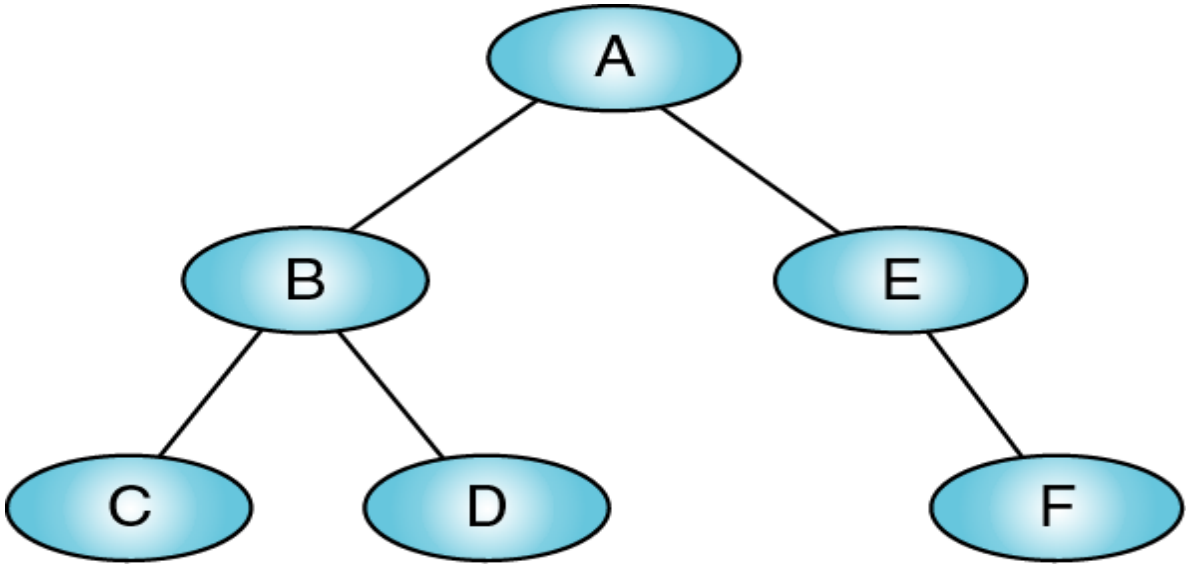
(b) Inorder traversal



(c) Postorder traversal

Üç farklı **depth-first traversal** düzeni

Binary Tree Traversals



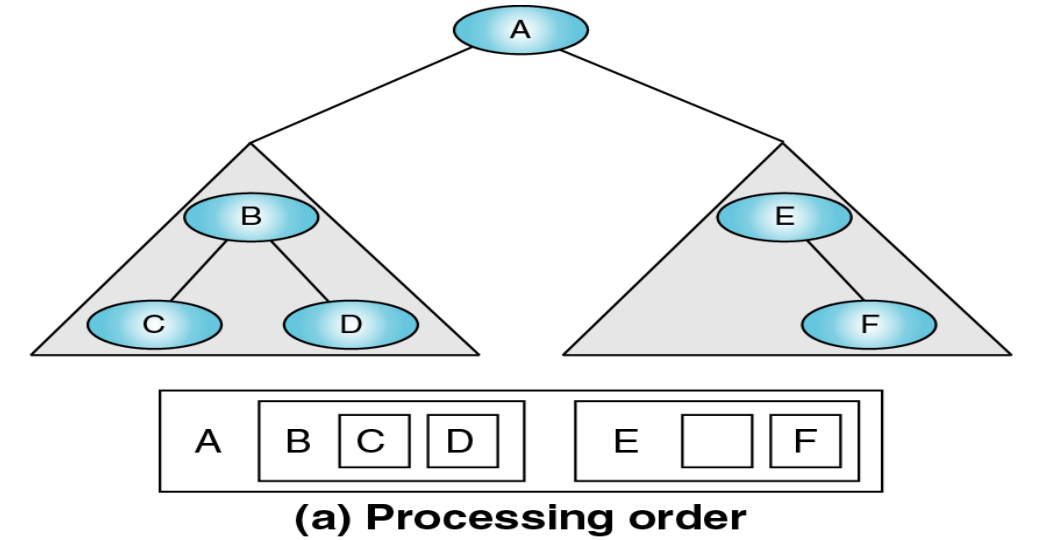
Preorder = ?
Inorder = ?
Postorder = ?

Figure 7-9

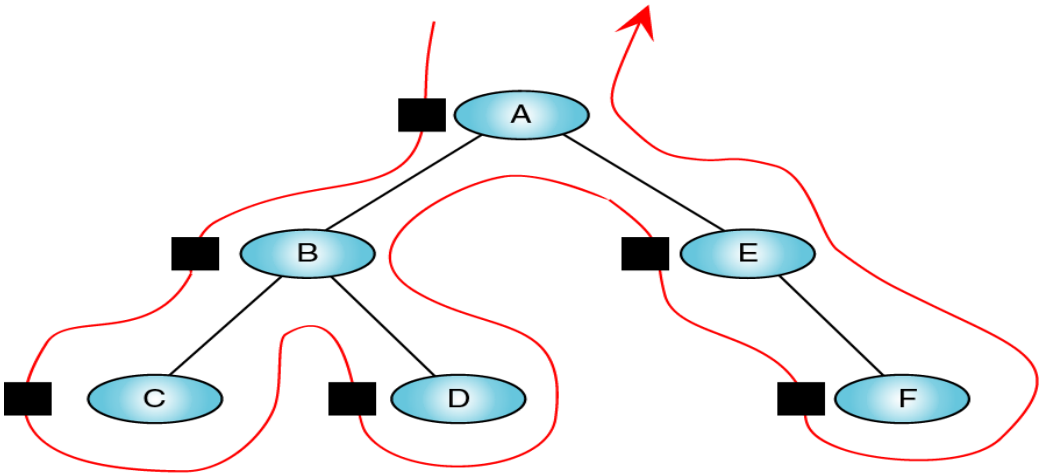
ALGORITHM 6-2 Preorder Traversal of a Binary Tree

```
Algorithm preOrder (root)
Traverse a binary tree in node-left-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   process (root)
   2   preOrder (leftSubtree)
   3   preOrder (rightSubtree)
2  end if
end preOrder
```

Binary Tree Traversals - Preorder



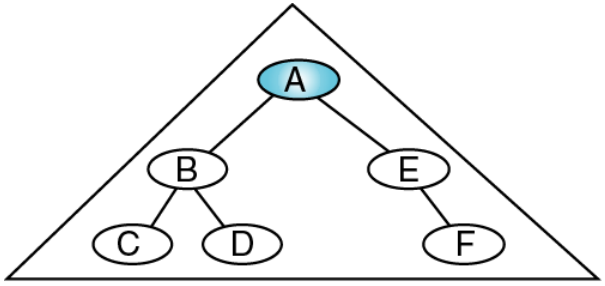
(a) Processing order



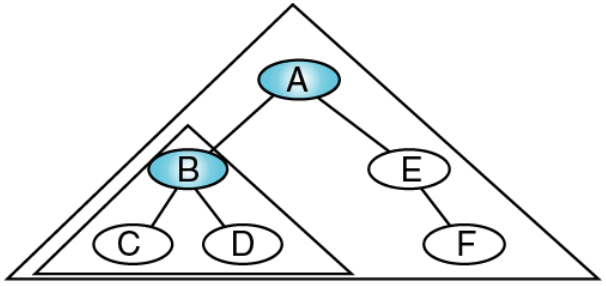
(b) "Walking" order

Preorder Traversal—A B C D E F

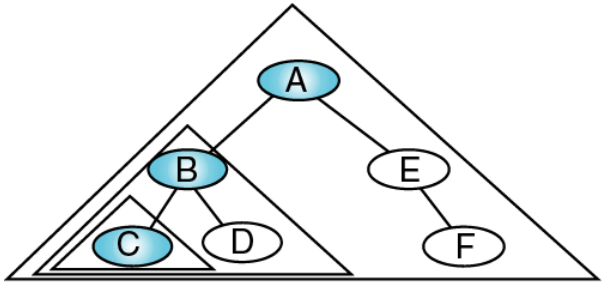
Binary Tree Traversals - Preorder



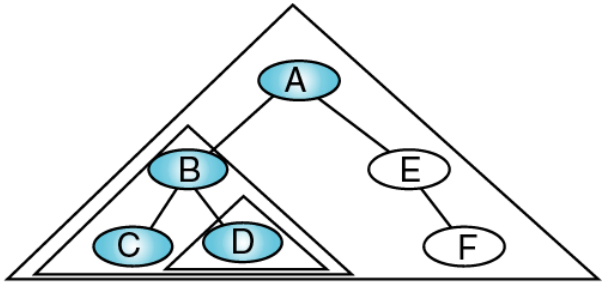
(a) Process tree A



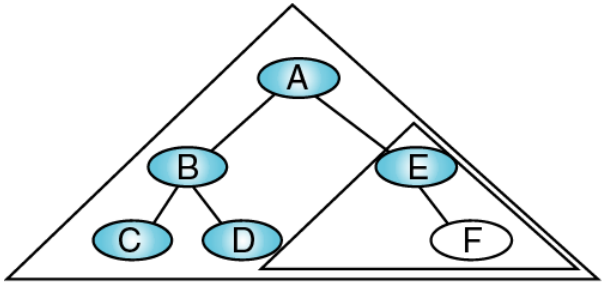
(b) Process tree B



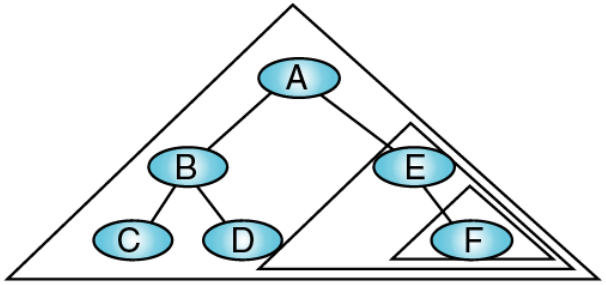
(c) Process tree C



(d) Process tree D



(e) Process tree E



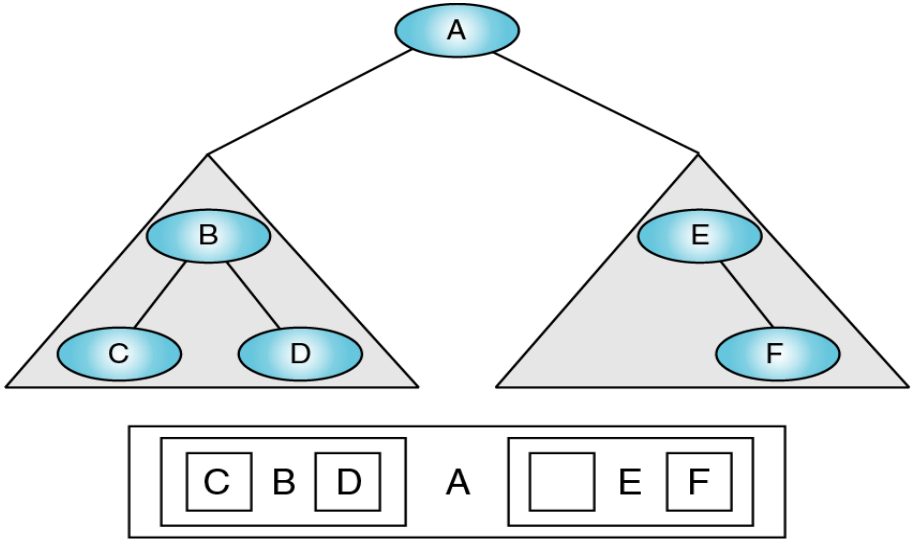
(f) Process tree F

Recursive algorithmic traversal of binary tree.

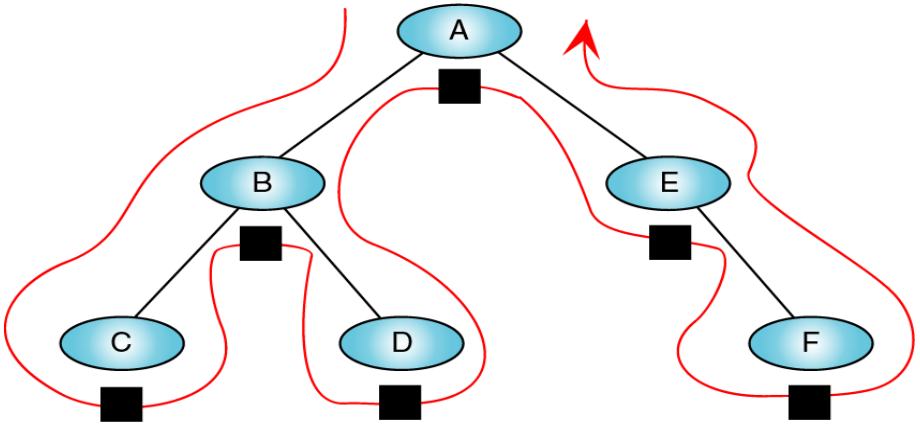
ALGORITHM 6-3 Inorder Traversal of a Binary Tree

```
Algorithm inOrder (root)
Traverse a binary tree in left-node-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   inOrder (leftSubTree)
   2   process (root)
   3   inOrder (rightSubTree)
2  end if
end inOrder
```

Binary Tree Traversals - Inorder



(a) Processing order



(b) "Walking" order

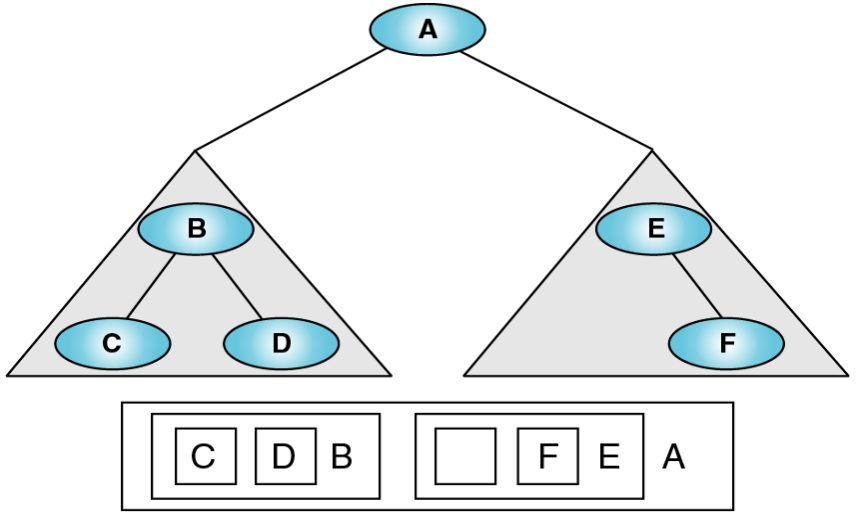
Figure 7-12

Inorder Traversal—C B D A E F

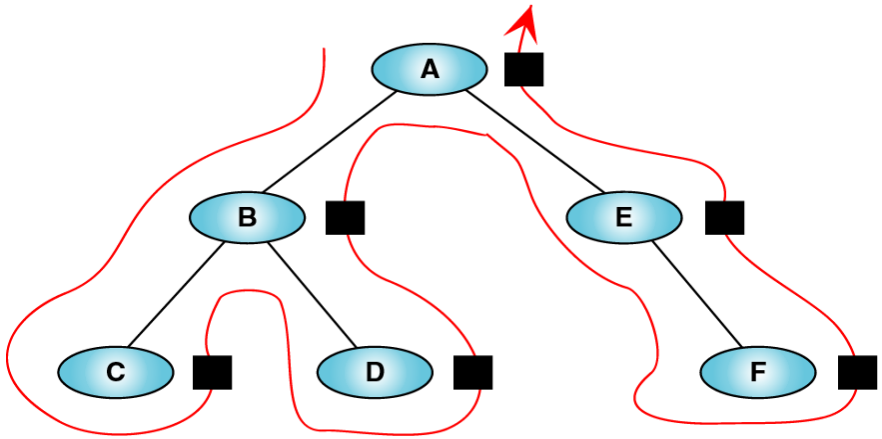
ALGORITHM 6-4 Postorder Traversal of a Binary Tree

```
Algorithm postOrder (root)
Traverse a binary tree in left-right-node sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   postOrder (left subtree)
   2   postOrder (right subtree)
   3   process (root)
2  end if
end postOrder
```

Binary Tree Traversals - Postorder



(a) Processing order



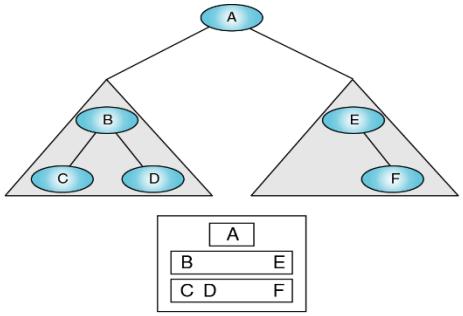
(b) "Walking" order

Figure 7-13

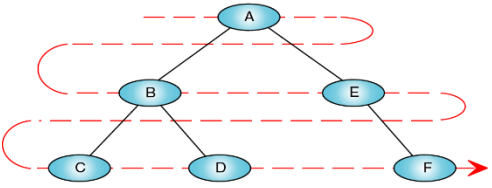
Postorder Traversal—C D B F E A

ALGORITHM 6-5 Breadth-first Tree Traversal

Breadth-first nedir?



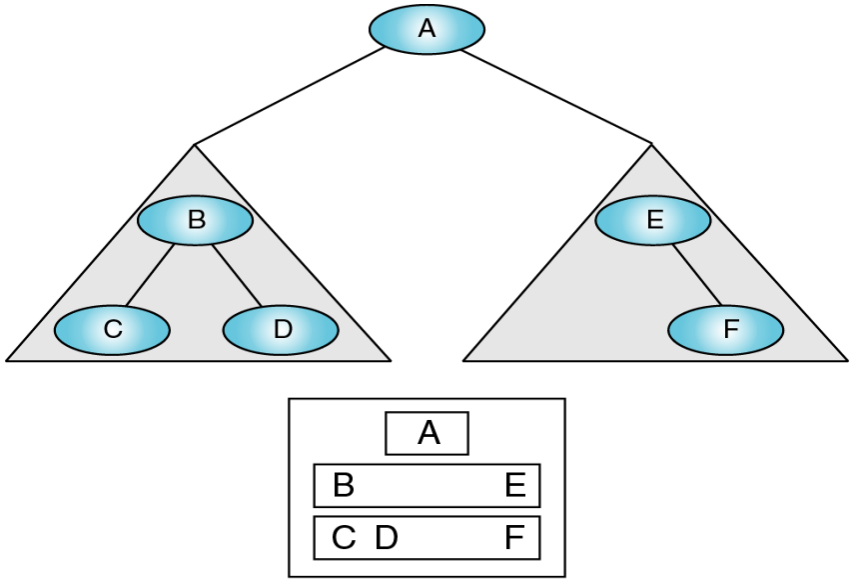
(a) Processing order



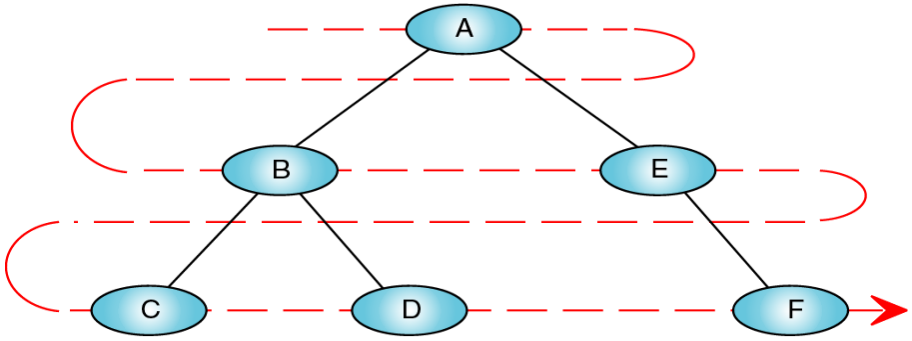
(b) "Walking" order

```
Algorithm breadthFirst (root)
Process tree using breadth-first traversal.
   Pre    root is node to be processed
   Post   tree has been processed
 1 set currentNode to root
 2 createQueue (bfQueue)
 3 loop (currentNode not null)
   1  process (currentNode)
   2  if (left subtree not null)
      1  enqueue (bfQueue, left subtree)
   3  end if
   4  if (right subtree not null)
      1  enqueue (bfQueue, right subtree)
   5  end if
   6  if (not emptyQueue(bfQueue))
      1  set currentNode to dequeue (bfQueue)
   7  else
      1  set currentNode to null
   8  end if
 4 end loop
 5 destroyQueue (bfQueue)
end breadthFirst
```

Binary Tree – Breadth-First Traversals



(a) Processing order

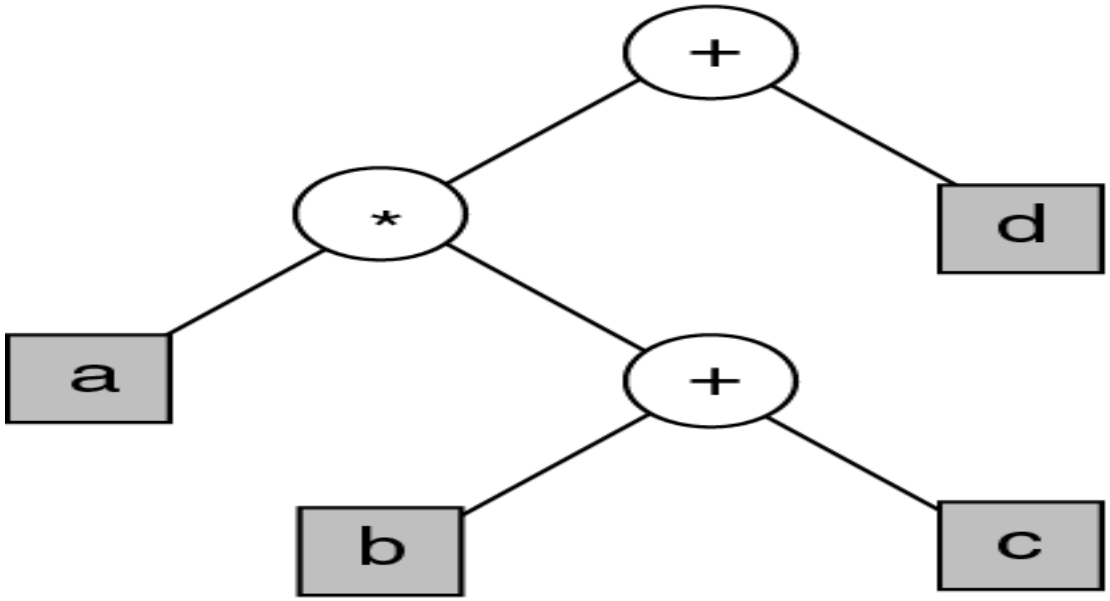


(b) "Walking" order

Figure 7-14

Expression Trees

```
a * (b + c) + d
```



Expression tree, şu özelliklere sahip bir ikili ağaçtır:

1. **Her yaprak bir operand**
2. **Kök ve iç düğümler operatör**
3. **Alt ağaçlar (subtrees), kökü bir operatör olan alt ifadelerdir.**

Infix Traversal Of An Expression Tree

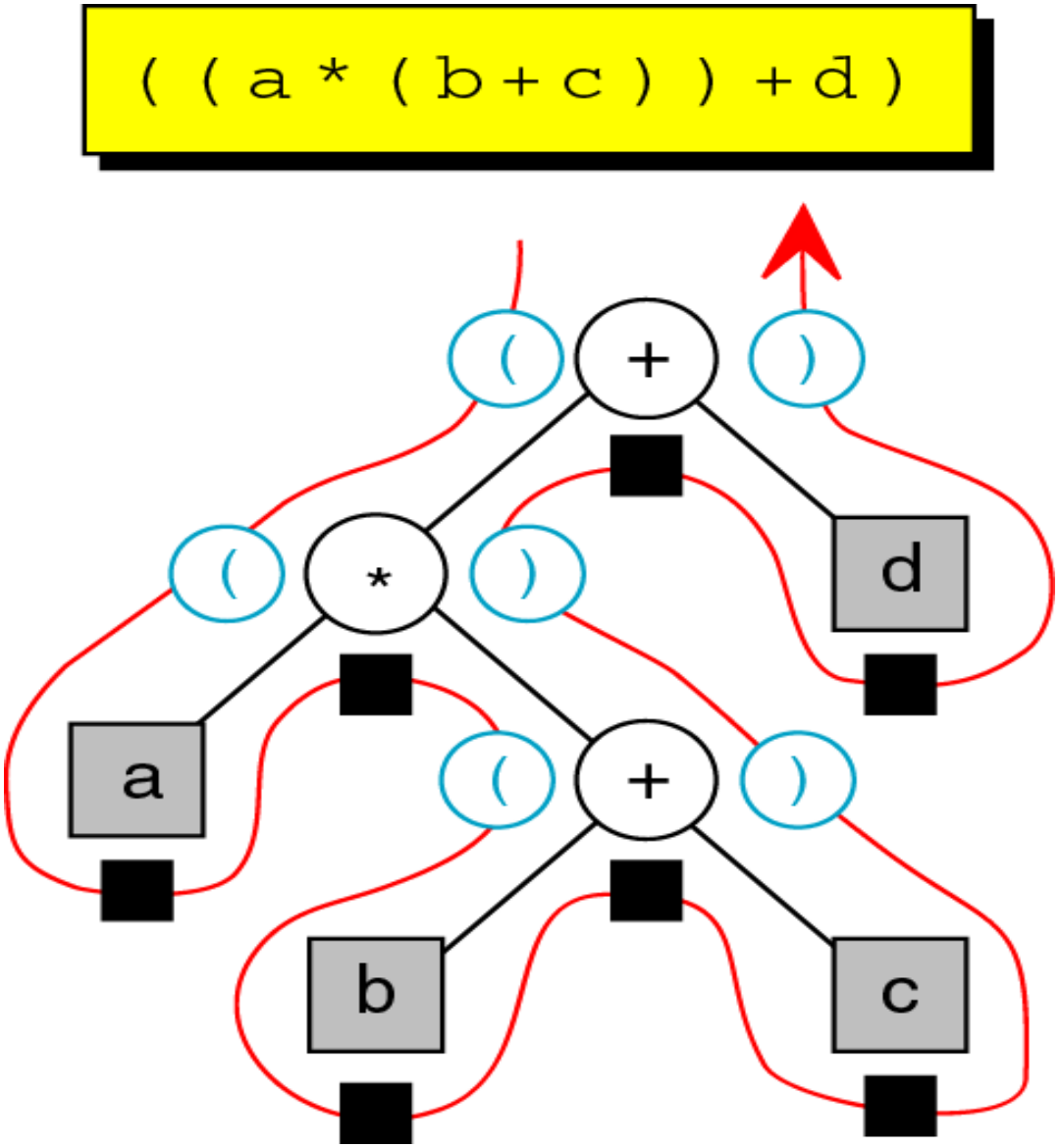


Figure 7-16

Infix Traversal Of An Expression Tree

algorithm **infix** (val tree <tree pointer>)

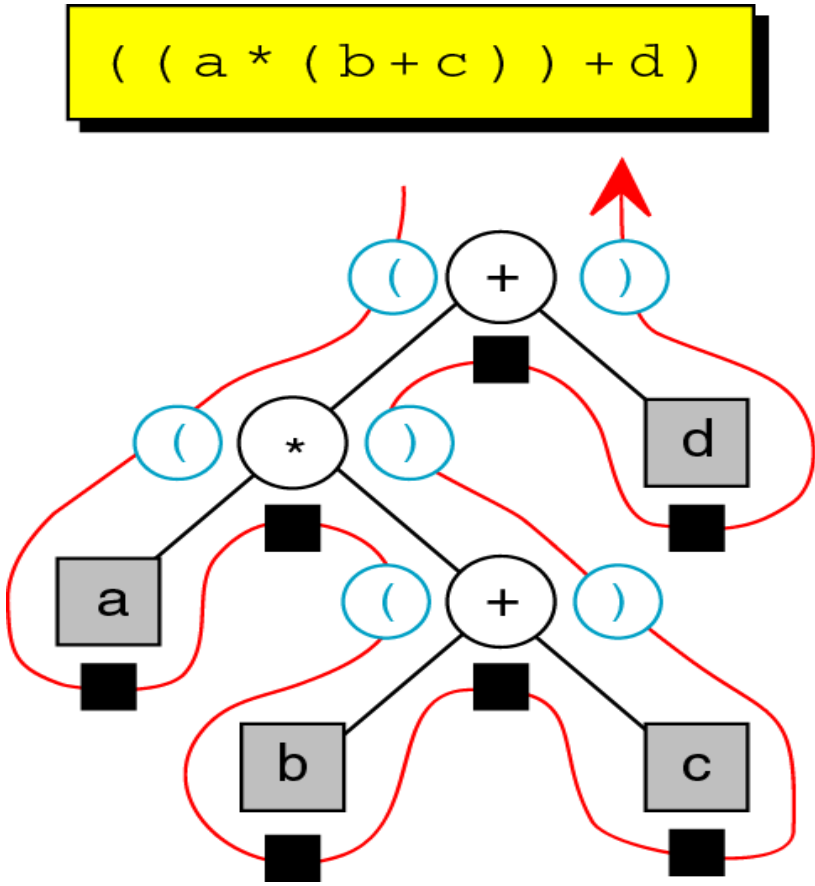
Print the infix expression for an expression tree.

Pre tree is a pointer to an expression tree

Post the infix expression has been printed

1. If (tree not empty)
    1. if (tree->token is an operand)
        1. print (tree->token)
    2. else
        1. print (open parenthesis)
        2. infix(tree->left)
        3. print(tree->token)
        4. infix(tree->right)
        5. print(close parenthesis)
2. Return

end **infix**



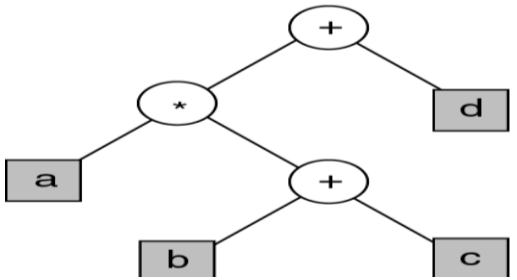
Postfix Traversal Of An Expression Tree

Bir ifadenin postfix traversal'ı, herhangi bir ikili ağacın temel **postorder traversal**'ını kullanır.

Parantezlere gerek yoktur

ALGORITHM 6-7 Postfix Traversal of an Expression Tree

```
Algorithm postfix (tree)
Print the postfix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the postfix expression has been printed
1  if (tree not empty)
   1  postfix (tree left subtree)
   2  postfix (tree right subtree)
   3  print   (tree token)
2  end if
end postfix
```



a b c + * d +

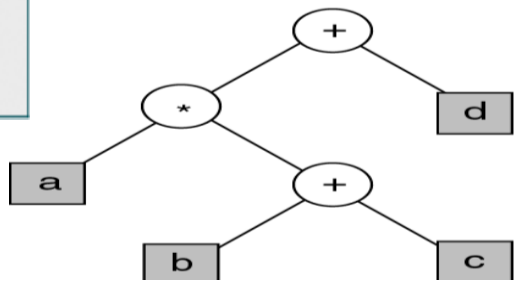
Prefix Traversal Of An Expression Tree

Standart **preorder traversal**'ını kullanır.

Parantezlere gerek yoktur

ALGORITHM 6-8 Prefix Traversal of an Expression Tree

```
Algorithm prefix (tree)
Print the prefix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the prefix expression has been printed
1  if (tree not empty)
   1  print  (tree token)
   2  prefix (tree left subtree)
   3  prefix (tree right subtree)
2  end if
end prefix
```



+ * a + b c d

Huffman Code

Latin alfabesi üzerine kurulu 7 bitlik bir karakter kümesidir.

- ASCII (The American Standard Code for Information Interchange):
  - Sabit uzunluklu bir koddur
  - Her karakter için 7 bit
- Bazı karakterler, örneğin 'E' diğerlerinden daha sık görülür.
- Her karakter maksimum bit sayısını kullanır
- Huffman, bunu daha verimli hale getirir
  - Sık görülenlere daha kısa kodlar atanır
  - Daha seyrek olanlar için daha uzun kodlar
- Tipik frekanslar:

Örneğin, İngilizce'de sıkça görülen iki karakter olan E veya T'ye **bir bitlik** kod atanabilir.

E ve T'den daha az sıklıkla görülen A, O ve R'nin her birine **iki bitlik** kod atanabilir.

Character	Weight	Character	Weight	Character	Weight
A	10	I	4	R	7
C	3	K	2	S	5
D	4	M	3	T	12
E	15	N	6	U	5
G	2	O	8		

TABLE 6-2 Character Weights for a Sample of Huffman Code

Huffman Code

Huffman Code için ağaç oluşturma adımları:

1. Karakter kümesi, görülme sıkılığına (**frequency**) göre sıraya dizilir. (En büyükten en küçüğe veya tersi)
   - Her karakter şimdi ağacın yaprak seviyesindeki bir düğümdür.
2. Müşretek ağırlığı (**combined weight**) en küçük olan iki düğüm bulunur, bunlar birleştirilir ve üçüncü bir düğüm oluşturmak üzere birleştirerek iki seviyeli basit bir ağaç (a **simple two-level tree**) elde ederiz.
   - Yeni düğümün ağırlığı, orijinal iki düğümün birleşik ağırlıklarıdır. Yapraklardan bir seviye yukarı çıkan bu düğüm, diğer düğümlerle birleştirilmeye uygundur.
3. Her seviyede tüm düğümler tek bir ağaca birleştirilinceye kadar 2. adım tekrar edilir.

Huffman...

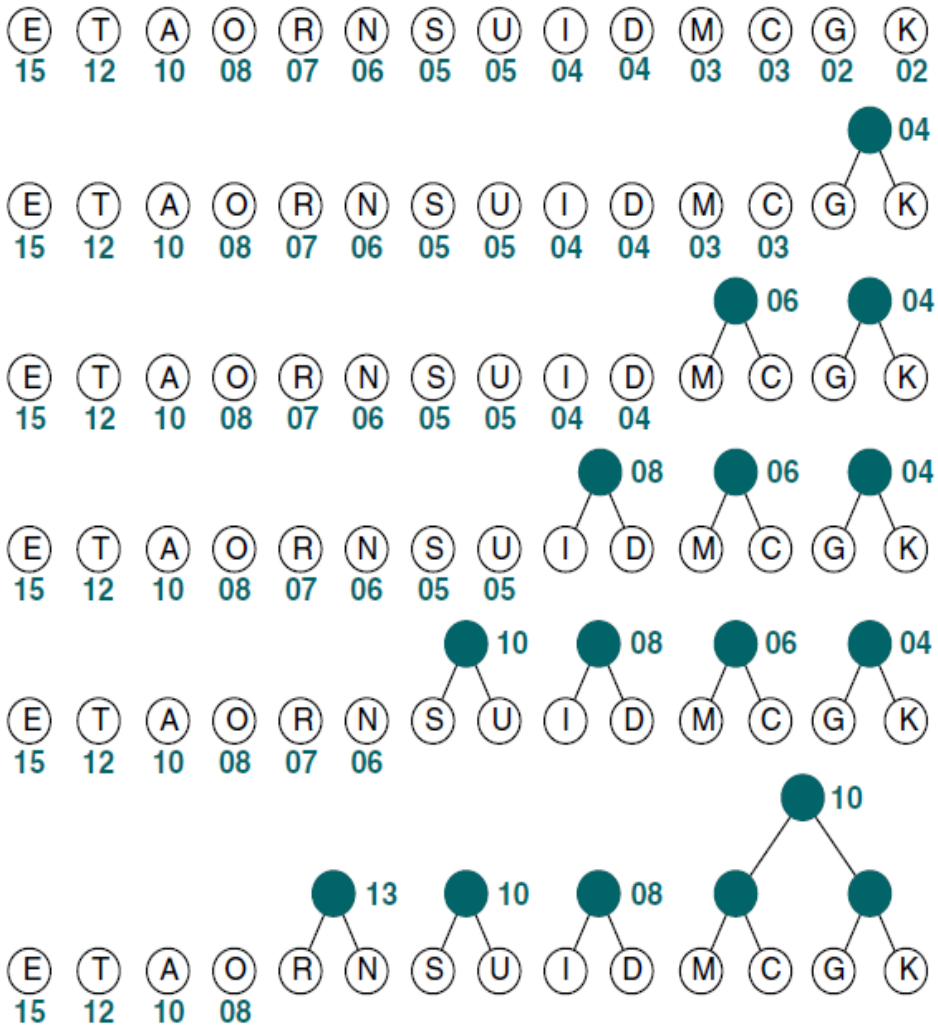


FIGURE 6-17 Huffman Tree, Part 1

Huffman...

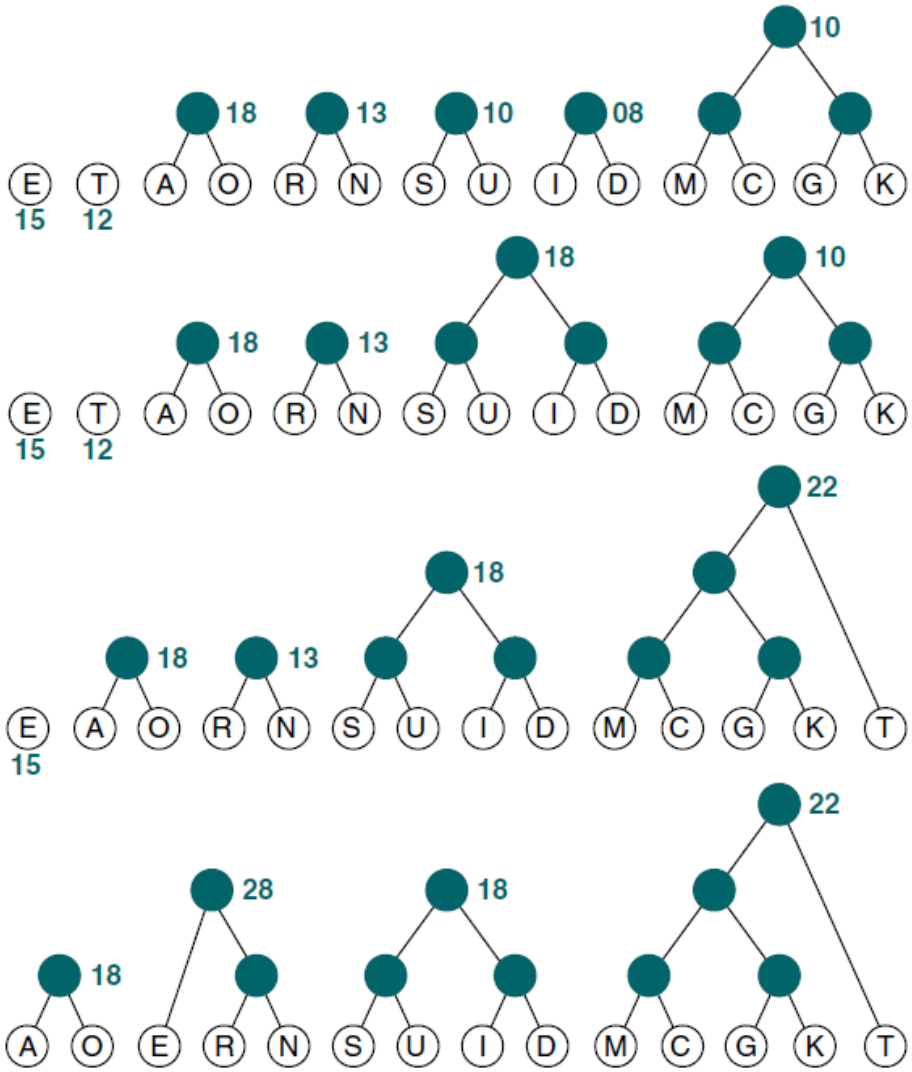


FIGURE 6-18 Huffman Tree, Part 2

Huffman...

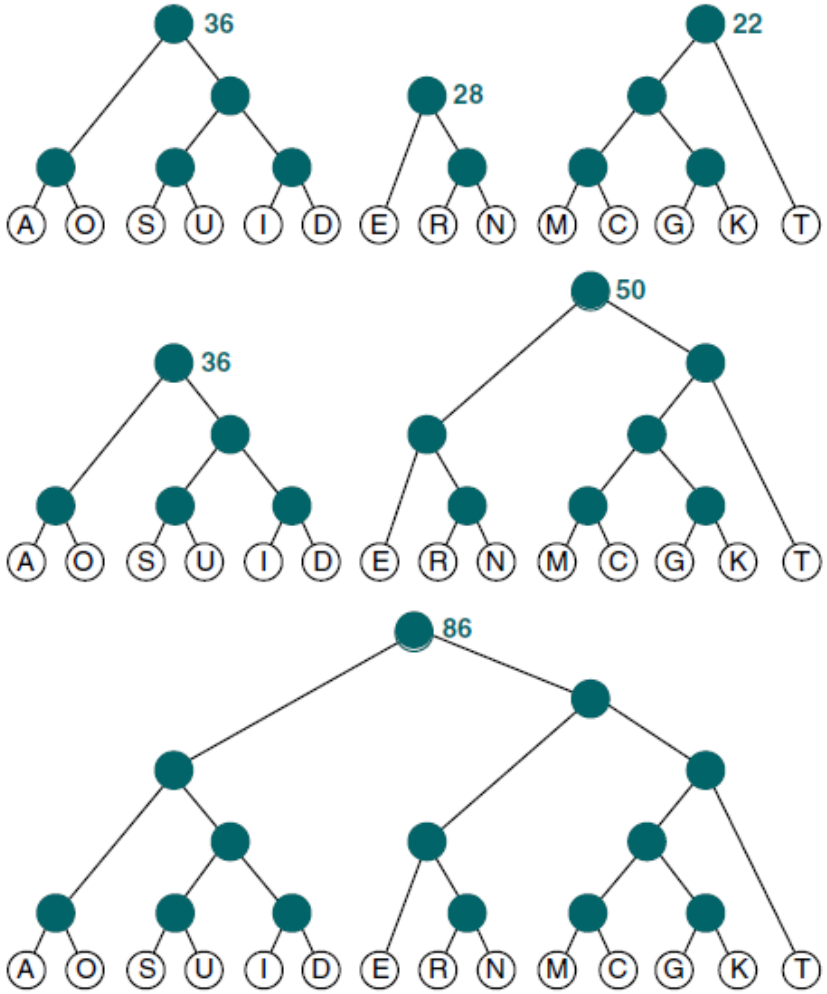
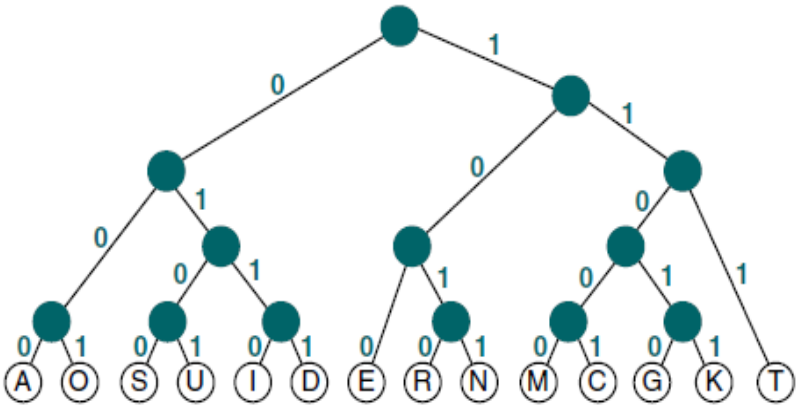


FIGURE 6-19 Huffman Tree, Part 3

Huffman...

- Şimdi her karaktere bir kod atarız.
- Her dal için bit değeri atanır:
  - 0 = left branch,
  - 1 = right branch.
- Bir karakterin kodu kökten başlayarak ve dalları takip ederek bulunur.

Huffman...



A = 000	U = 0101	E = 100	M = 11000	K = 11011
O = 001	I = 0110	R = 1010	C = 11001	T = 111
S = 0100	D = 0111	N = 1011	G = 11010	

FIGURE 6-20 Huffman Code Assignment

En sık görülen harflerin daha az bit ile temsil edildiğine dikkat edin.

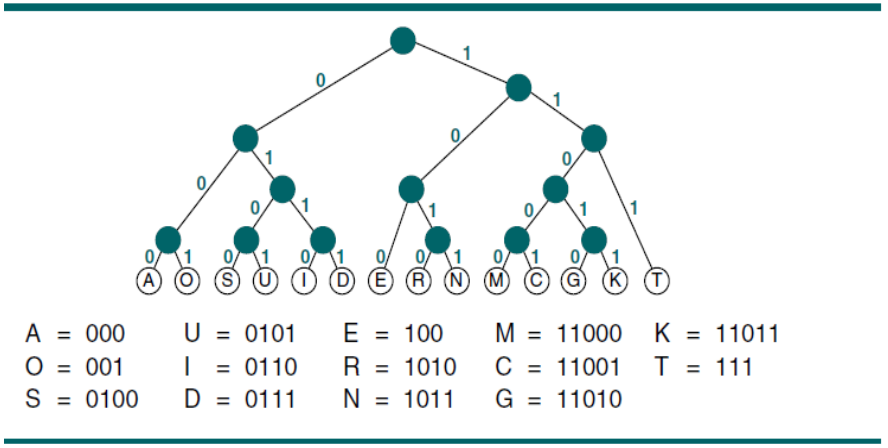
Huffman...

- Huffman kodu **veri sıkıştırma** için yaygın olarak kullanılır; gönderilen veya tutulan bit sayısını azaltır.

- Hiçbir karakterin ASCII eşdeğerinden daha uzun olmadığı değişken uzunluklu bir kodlama (**variable-length encoding**) sistemi olduğu için iletim zamanından tasarruf sağlar.

Huffman...

- Aşağıdaki bit dizisinin geliştirilen bu Huffman kodunu kullandığını farzedersek




```
000110100010010111110000001010110111100111
```

000110100010010111110000001010110111100111

A G O O D M A R K E T

İlk biti aldıktan sonra, kökten başlıyoruz ve sonraki iki parçayı okuduktan sonra yaprak A'ya gelen 000 yolu takip ediyoruz.

```
AGOODMARKET
```

Bir yaprağa ulaşıldığında o karakter çözülmüş olur.

General Trees

- Genel ağaç, her bir düğümün sınırsız **outdegree**'ye sahip olabileceği bir ağaçtır.
  - Her bir düğüm, ihtiyacını karşılayacak gereken sayıda çocuğa sahip olabilir.

Insertions into General Trees

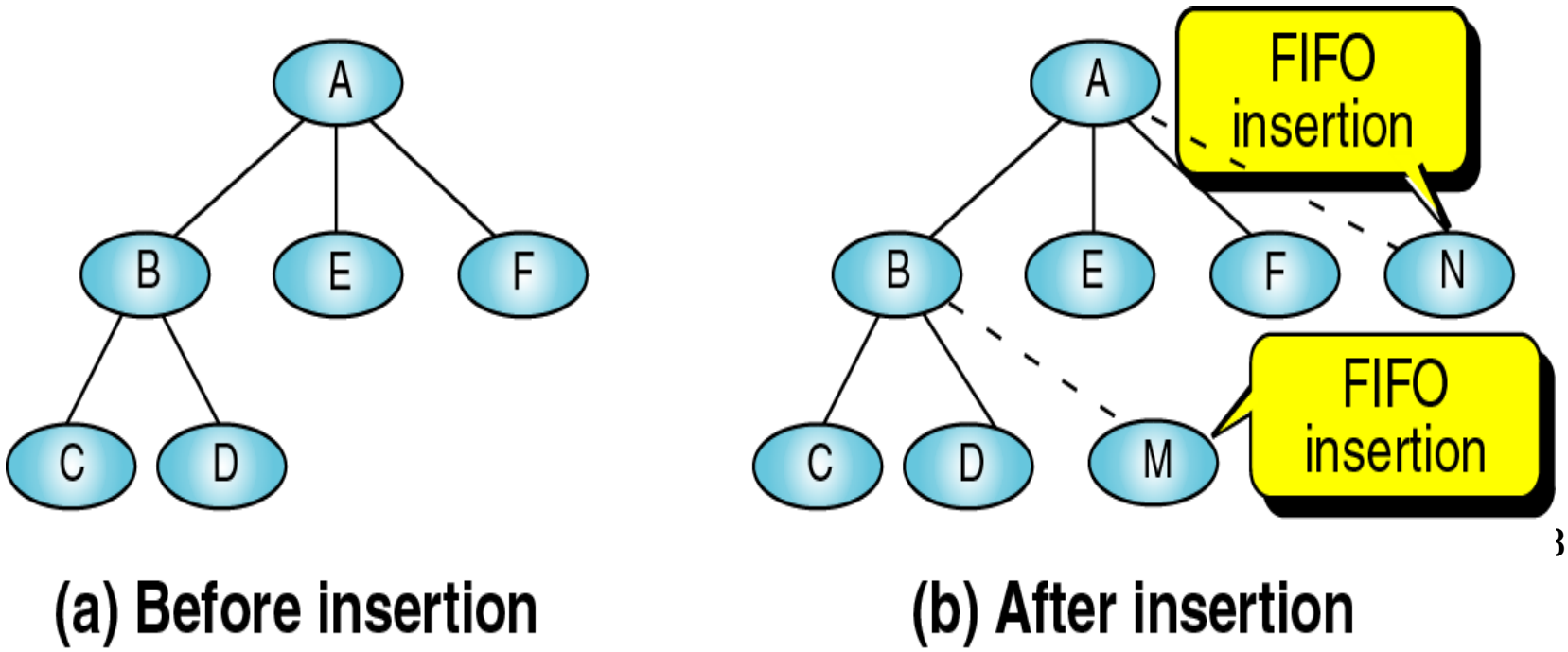
- Genel bir ağaca bir düğüm eklemek için, kullanıcının düğümün ebeveynini belirtmesi/sağlaması gerekir.
- Ebeveynler dikkate alındığında, üç farklı kural kullanılabilir:

1) first in–first out (FIFO) insertion,
2) last in–first out (LIFO) insertion,
3) key-sequenced insertion

Insertion Into General Trees

Uygulama, verilerin girildiği sırada işlenmesini gerektirdiğinde FIFO sırası kullanılır.

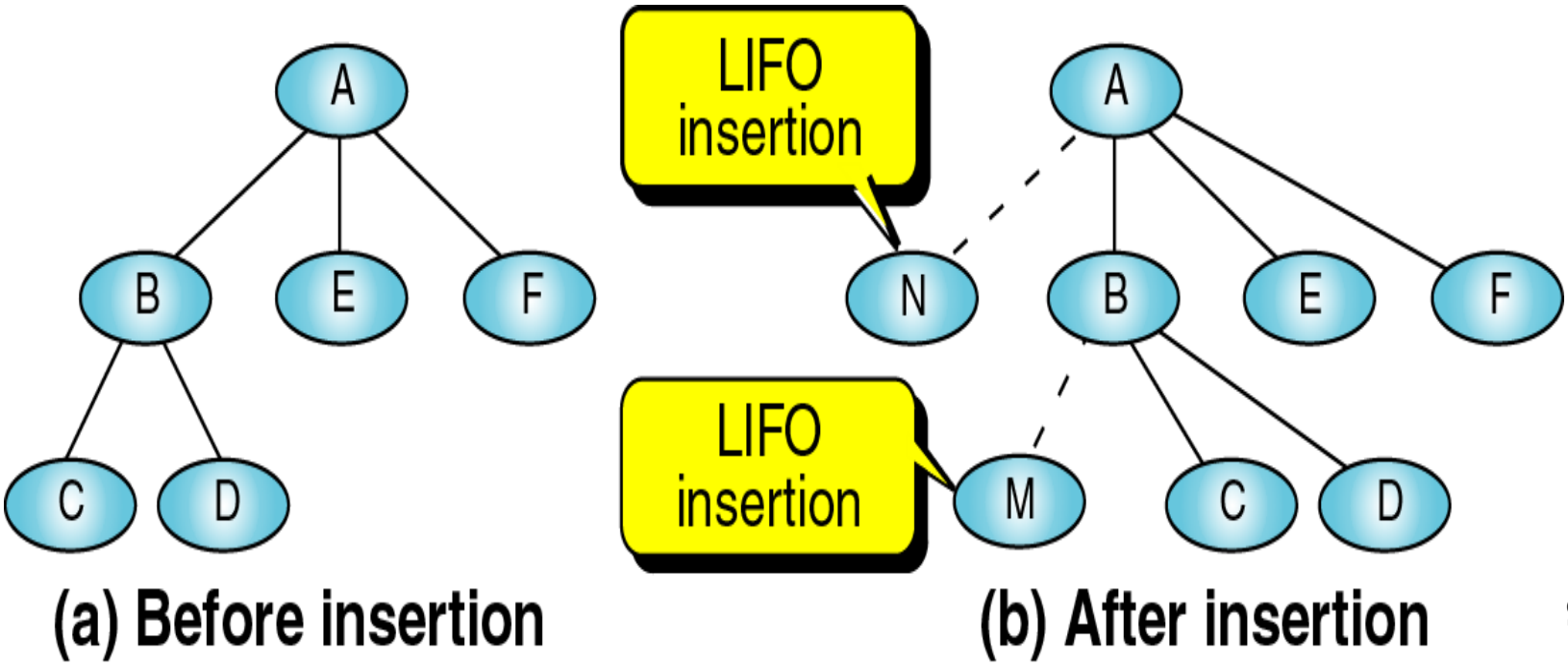
FIFO insertion; düğümler kardeş listesinin sonuna eklenir, (kuyruğun sonuna ekleme gibi).



Insertion Into General Trees

Kardeş listesini oluşturuldukları sıranın tersi sırada işlemek için LIFO yerleştirmeyi kullanıyoruz.

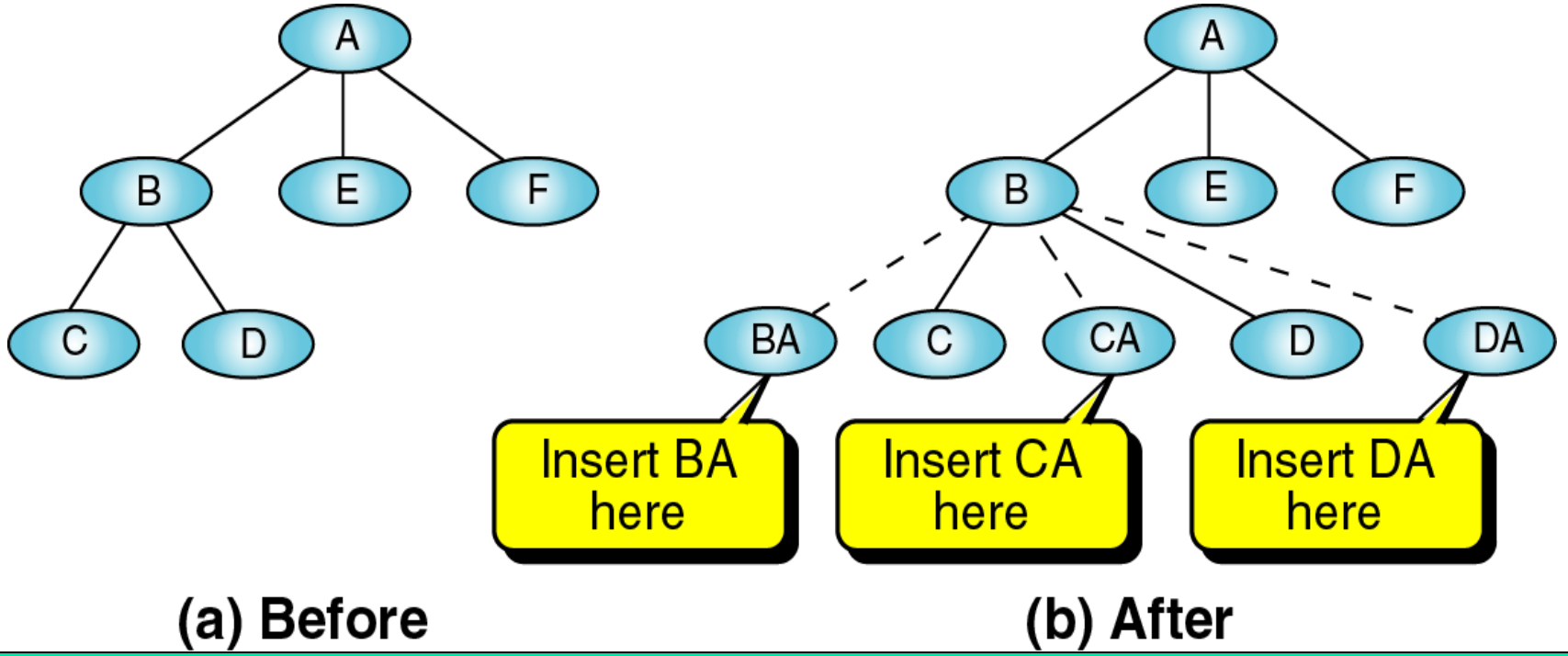
LIFO insertion; yeni düğümü kardeş listesinin başına yerleştirir, (yığının tepesine eleman yerleştirme gibi).



Insertion Into General Trees

Ekleme kurallarının belki de en yaygın olanı

Key-sequence insertion; yeni düğümü kardeş düğümleri arasında anahtar sıralamasıyla yerleştirir.



Anahtar sırasına yerleştirme mantığı, bağlı listeye yerleştirme mantığına benzer. Ebeveynin ilk çocuğundan başlayarak, doğru yerleştirme noktasını bulana kadar kardeş işaretçileri (sağ) izleriz ve sonra selefler ve halefler (varsa) ile bağlar kurarız.

General Tree Deletions

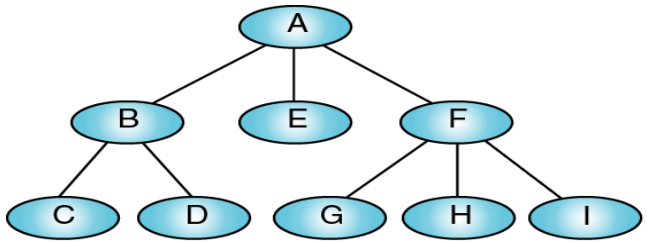
- Genel ağaca eleman eklemek için standart kurallar geliştirememize rağmen, standart silme kuralları geliştirebiliriz.
- İlk kural şudur: **bir düğüm yalnızca bir yapraksa silinebilir**.
  - Genel ağaçta bu, herhangi bir çocuğu varsa bir düğümün silinemeyeceği anlamına gelir.
- Kullanıcı, çocuğu olan bir düğümü silmeye çalışırsa, program, düğümü çocukları silinene kadar silinemediğini bildiren bir hata mesajı verir.

Changing a General Tree to a Binary Tree

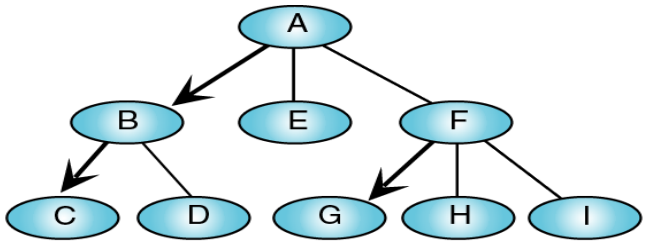
- İkili ağaçlar, genel ağaçlardan daha kolay temsil edilir.
  - Bu nedenle, bir ikili ağaç formatı kullanarak genel ağaçları temsil edebilmek istiyoruz.
- İkili ağaç formatı, sol ve sağ işaretçilerin anlamı değiştirilerek benimsenebilir.
- Genel ağaçta kullanabileceğimiz iki ilişki vardır:
  - Ebeveynden çocuğa (Parent to child),
  - Kardeşten kardeşe (Sibling to sibling)

Bu iki ilişkiyi kullanarak, herhangi bir genel ağacı ikili ağaç olarak temsil edebiliriz.

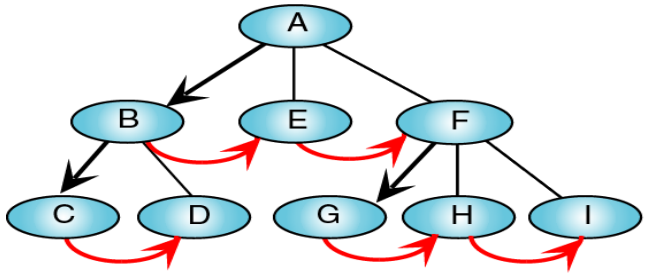
Converting
General Trees To Binary Trees



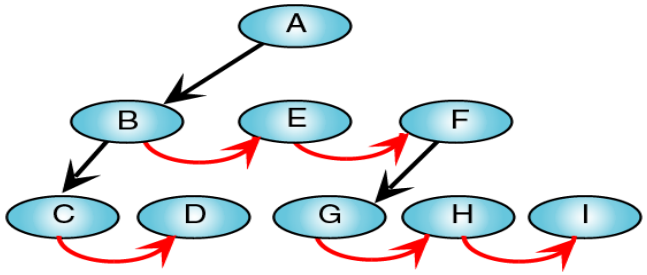
(a) The general tree



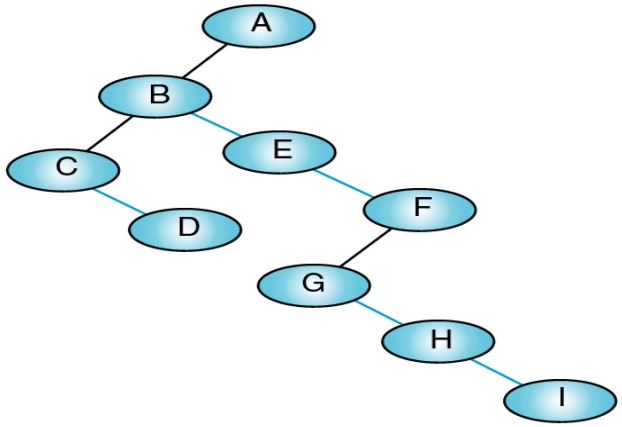
(b) Identify leftmost children



(c) Connect siblings



(d) Delete unneeded branches



(e) The resulting binary tree

b) İlk önce ebeveynden ilk çocuğa olan dalı belirleriz.

- Her bir ebeveynden gelen bu dallar ikili ağaçta işaretçiler haline gelir

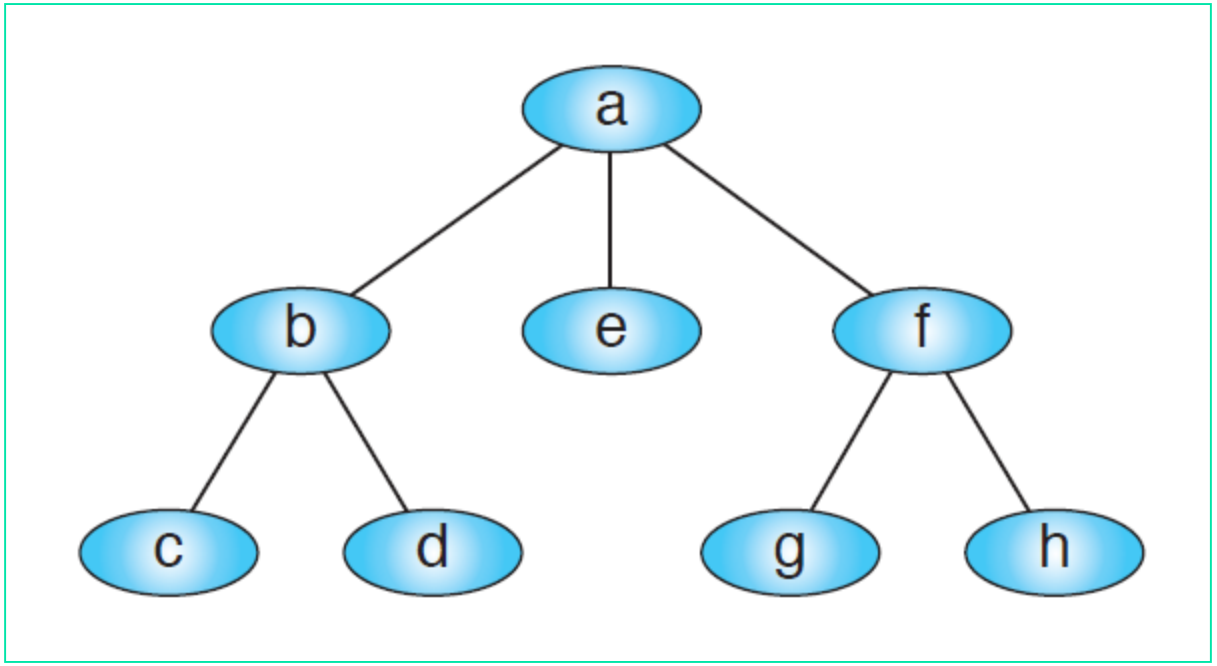
c) Sonra kardeşleri, en soldaki çocuktan başlayarak, her kardeş için sağ kardeşine bir dal kullanarak bağlarız.

d) Üçüncü ve son adım ebeveynden çocuklara olan tüm gereksiz dallar yok edilir.

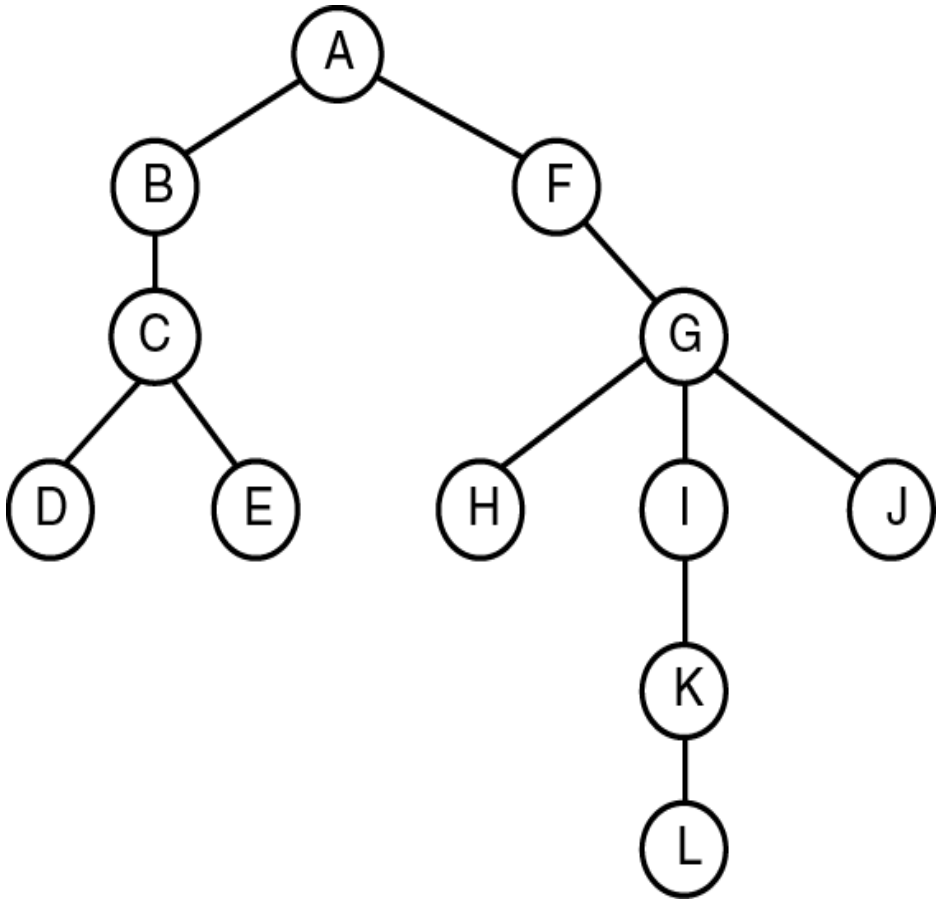
Alıştırmalar

- Aşağıdaki parantez notasyonun ağaç gösterimini çiziniz:

a (b (c d) e f (g h))



Alıştırmalar

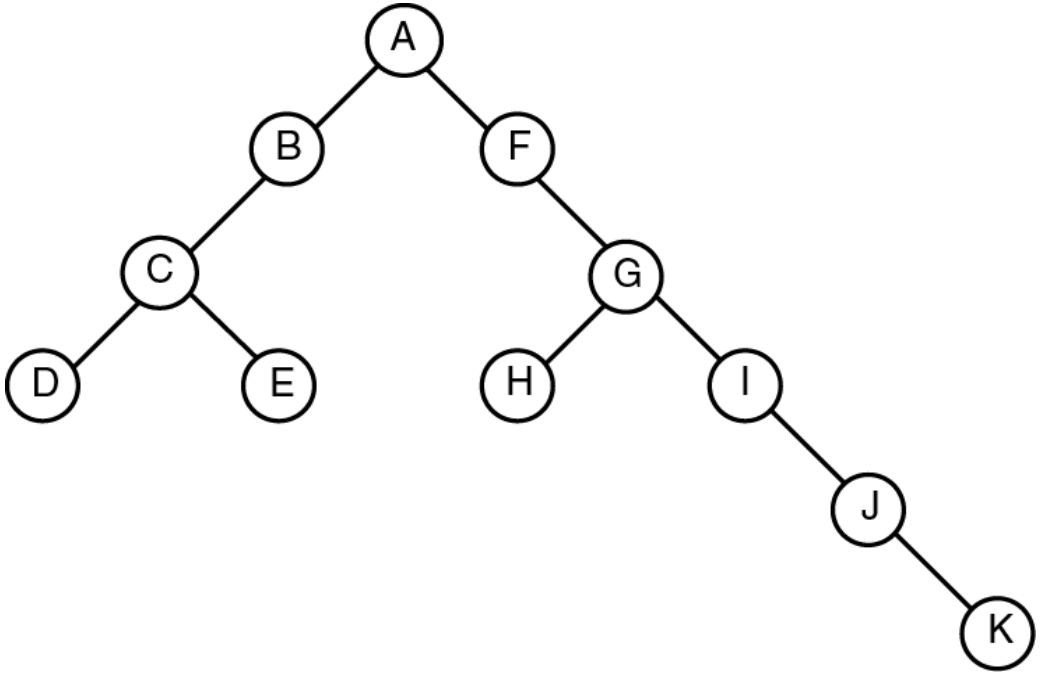


Find:

1. **Root** : A
2. **Leaves** :D,E,H,J,L
3. **Internal nodes** :B,F,C,G,I,K
4. **Ancestors of H** :G,F,A
5. **Descendents of F** :G,H,I,K,L,J
6. **Indegree of F** :1
7. **Outdegree of B** :1
8. **Siblings of H** :I,J
9. **Parent of K** :I
10. **Level of G** :2
11. **Depth of I** :3
12. **Height of I** :2

Alıştırmalar

Aşağıdaki ağacın denge faktörü (balance factor) nedir?



Balance factor=3-5= **-2**

Alıştırmalar

28 düğümlü bir ağacın maksimum ve minimum yüksekliği nedir?

$$\mathbf{H_{max} = N=28}$$

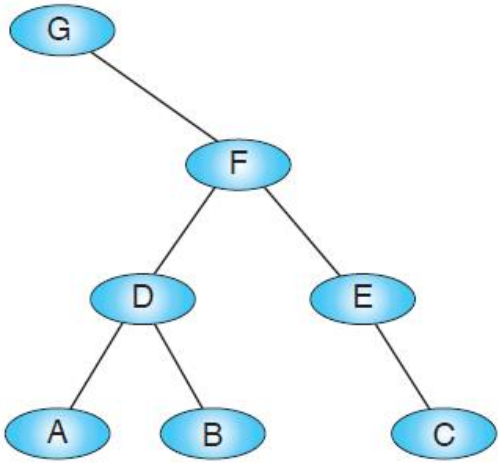
$$\mathbf{H_{min} = [\log_2 N] + 1} = \log_2 28 + 1 = 4 + 1 = \mathbf{5}$$

Alıştırmalar

Bir ikili ağacın yedi düğümü vardır. Ağacın preorder ve postorder gezinmeleri aşağıda verilmiştir. Bu ağacı çizilebilir misiniz? Değilse açıklayın.

Preorder: GFDABEC
Postorder: ABDCEFG

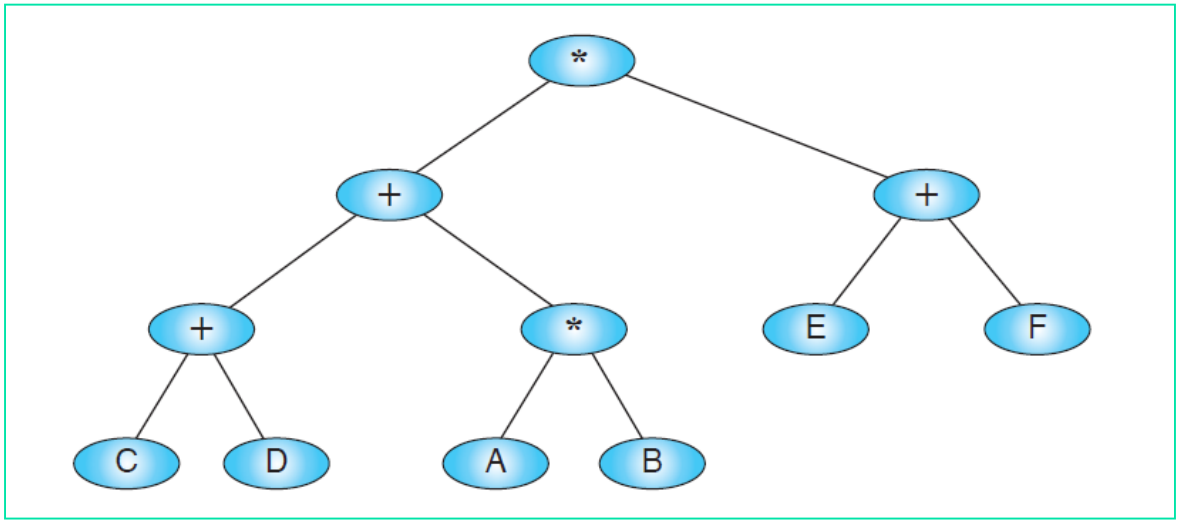
Bu sorunun birkaç olası çözümü vardır. Bunlardan biri yanda verilmiştir. Eşsiz bir çözüm için, INORDER gezinmeyi de bilmemiz gerekir.



Alıştırmalar

Aşağıdaki **infix** ifadesi için ifade ağacını (expression tree) çizin ve ardından prefix ve postfix ifadelerini bulun:

$$(C + D + A \times B) \times (E + F)$$

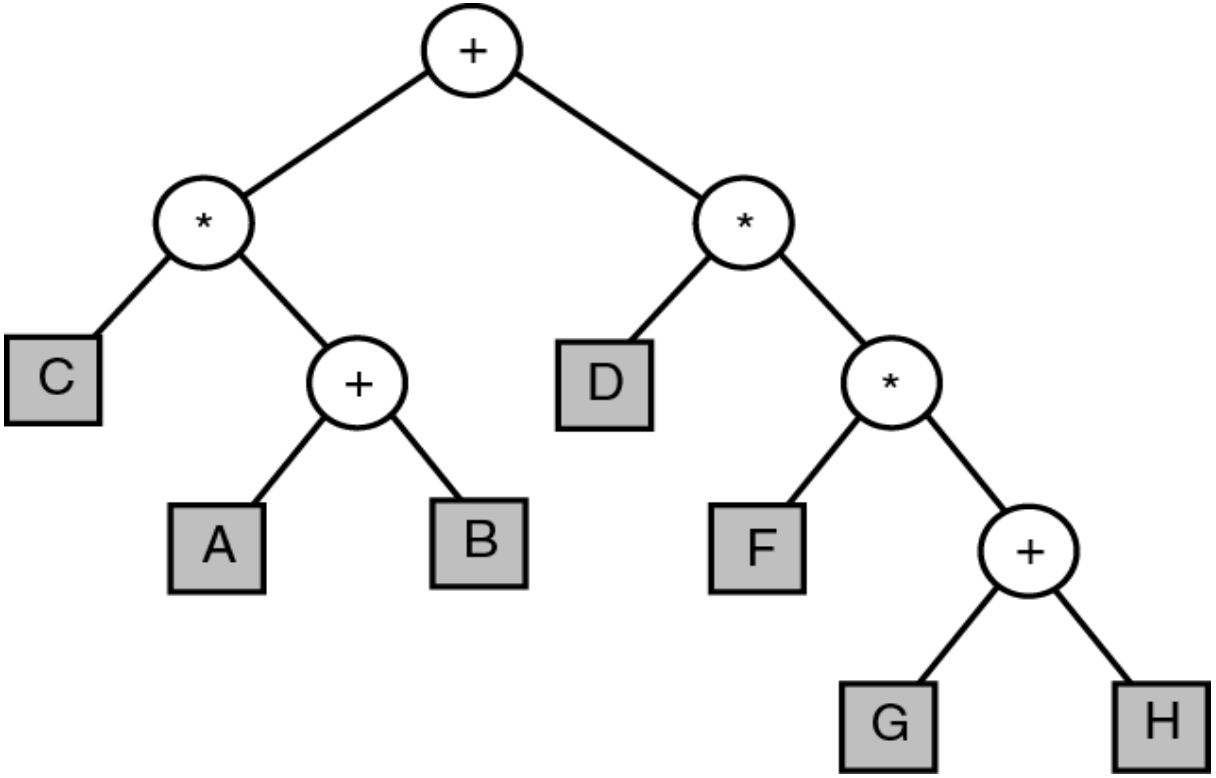


Prefix: * + + C D * A B + E F

Postfix: C D + A B * + E F + *

Lab Uygulaması

Aşağıdaki ağacın infix, prefix ve postfix ifadelerini bulun



Lab Uygulaması

- **Aşağıdakileri yapan bir program yazınız:**
  - **Aşağıda verilen ikili ağacı yaratın;**
  - **Ağacın infix, prefix ve postfix ifadelerini ekrana basmak için bir menü seçimi oluşturun.**
  - **Ağacı seçilen ifade türüne göre ekrana bastırın.**