İkili Ağaçlar
Binary Trees

- Ağaç veri yapısı temel kavramları
- İkili Arama Ağaçları
- İkili Ağaçlar ve dolaşım
- İfade Ağaçları

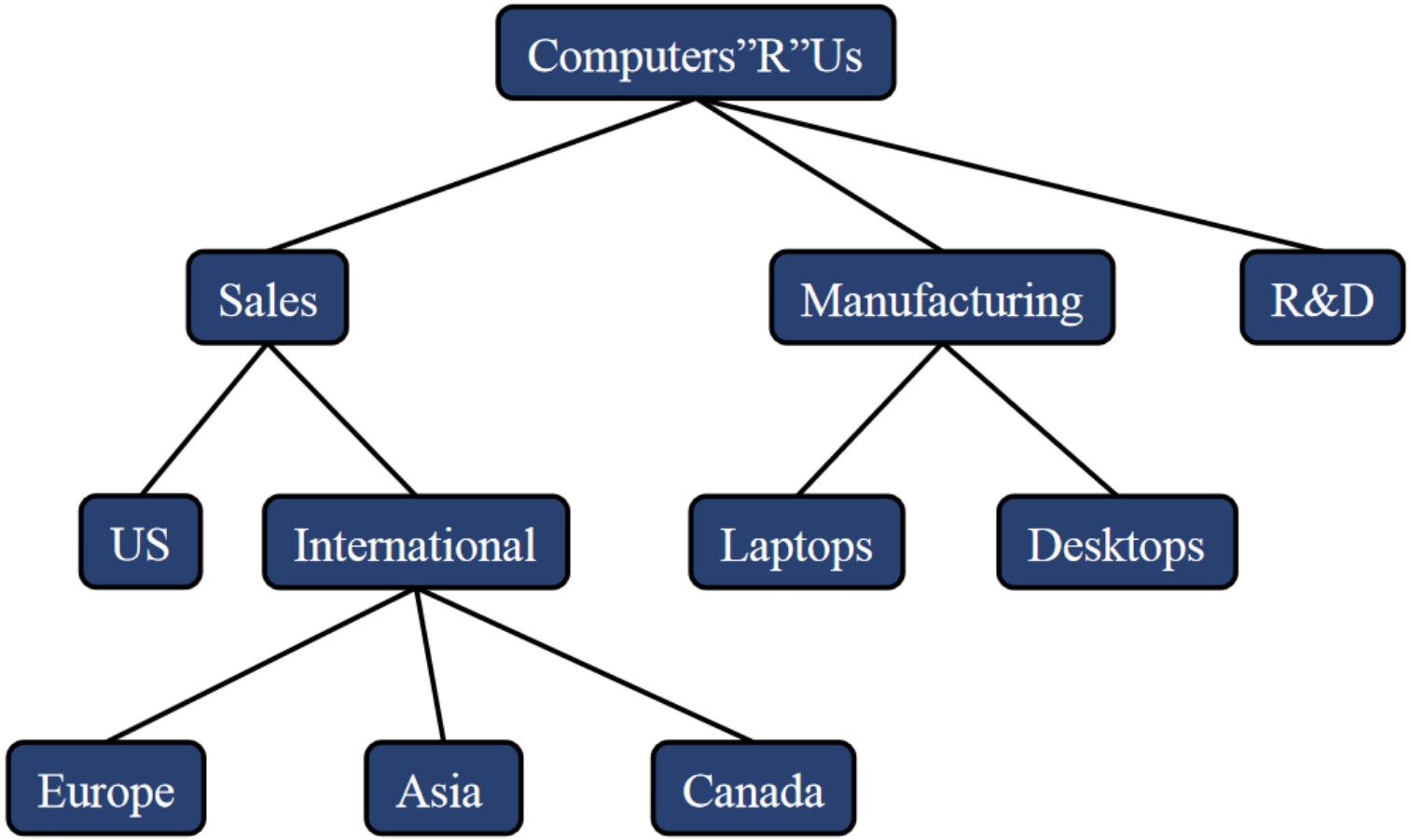
Ağaçlar: temel tanımlar ve terminoloji

- Doğrusal veri yapısı: önce ve sonra ilişkisi
- diziler, listeler, stack, kuyruklar, vb
- Ağaçlar verilerin hiyerarşik bir şekilde organize edildiği veri bir doğrusal olmayan veri yapısıdır

Tree: A Hierarchical ADT

Örnek Ağaç: Hiyerarşik yapı

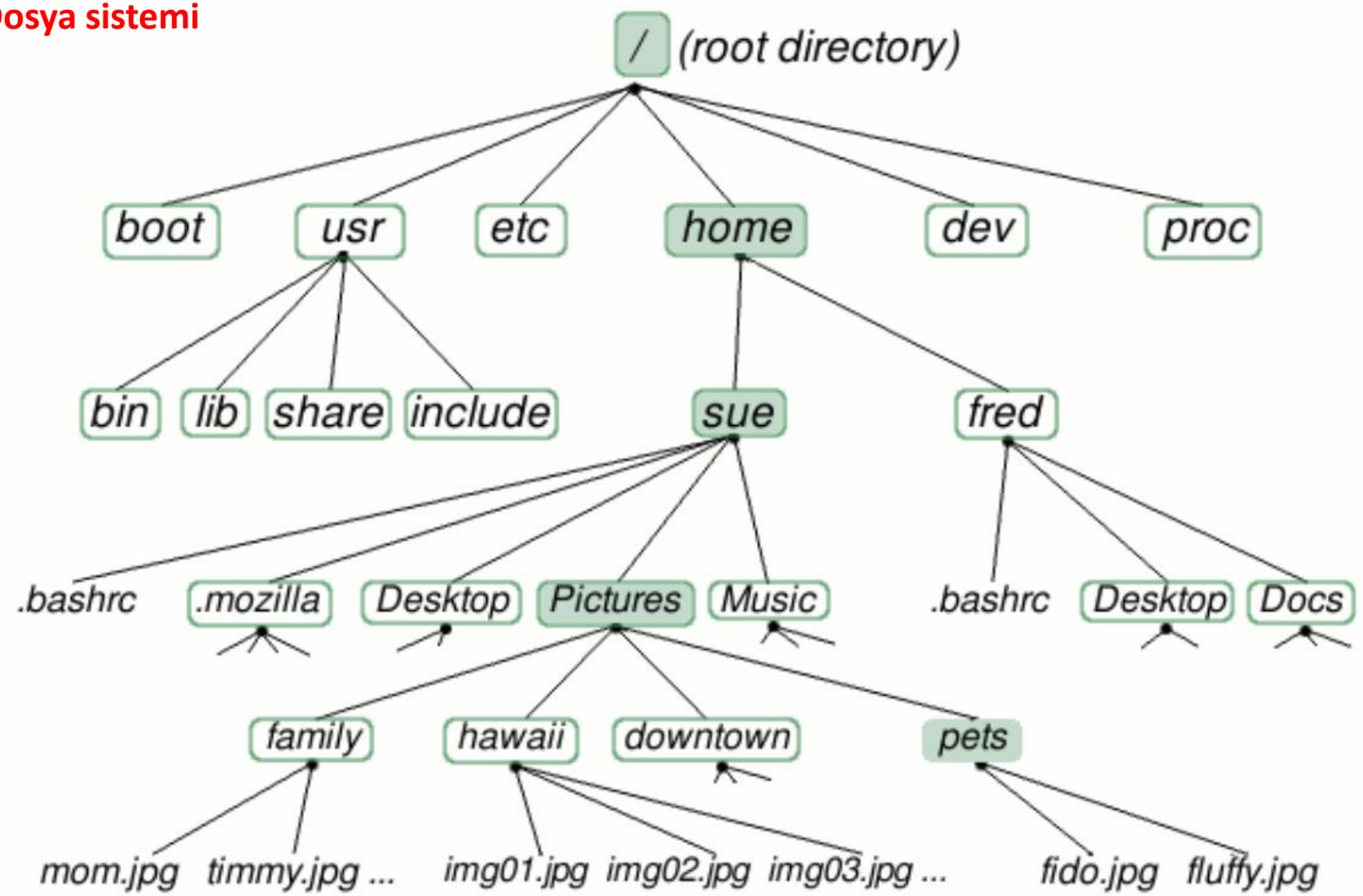
- A tree (upside down) is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Each element (except the top element) has a parent and zero or more children elements



http://www3.nccu.edu.tw/~yuf/Lec6.pdf

Linux/Unix file systems

Örnek Ağaç: Dosya sistemi

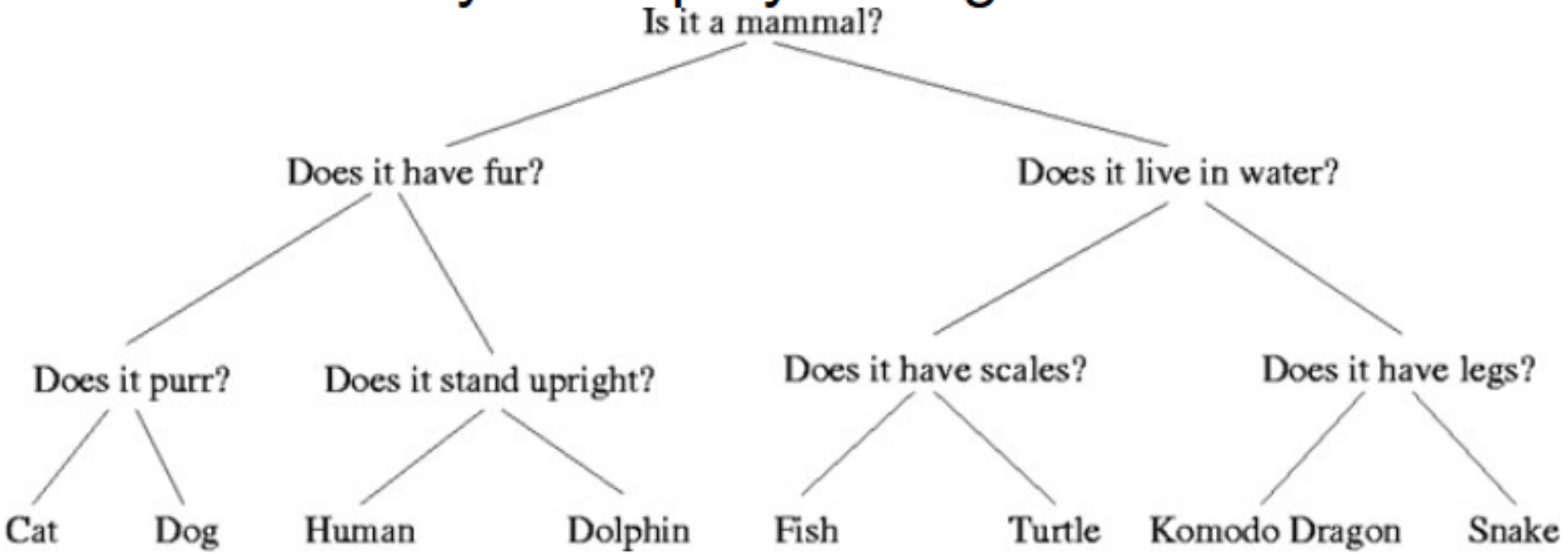


http://www3.nccu.edu.tw/~yuf/Lec6.pdf

Örnek Ağaç: Karar ağacı

Tree Example – Decision Tree

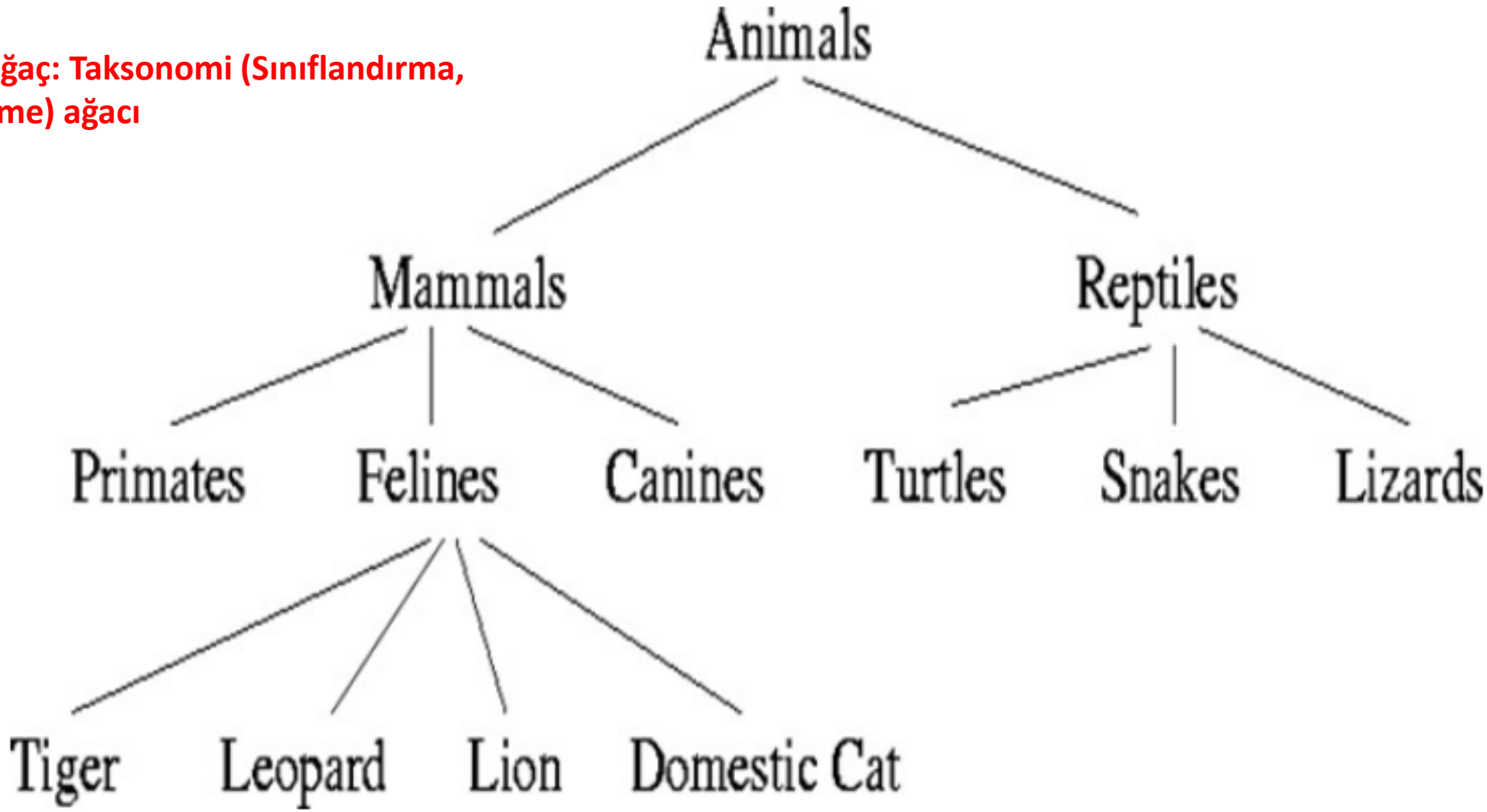
- tool that uses a tree-like graph or model of decisions and their possible consequences
  - including chance event outcomes, resource costs, and utility
- It is one way to display an algorithm.



https://slideplayer.com/slide/8222301/

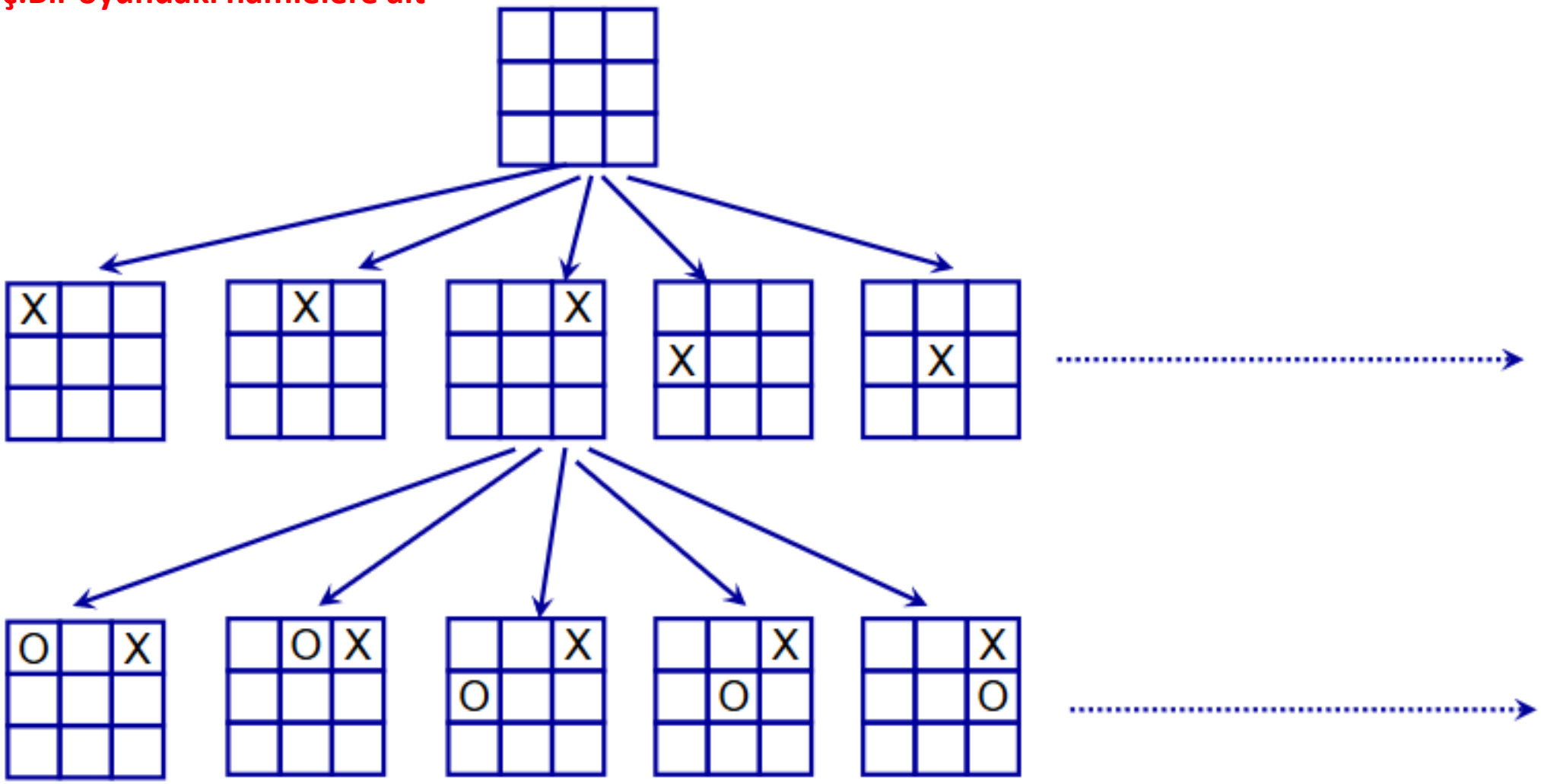
Tree Example – Taxonomy Tree

Örnek Ağaç: Taksonomi (Sınıflandırma, düzenleme) ağacı



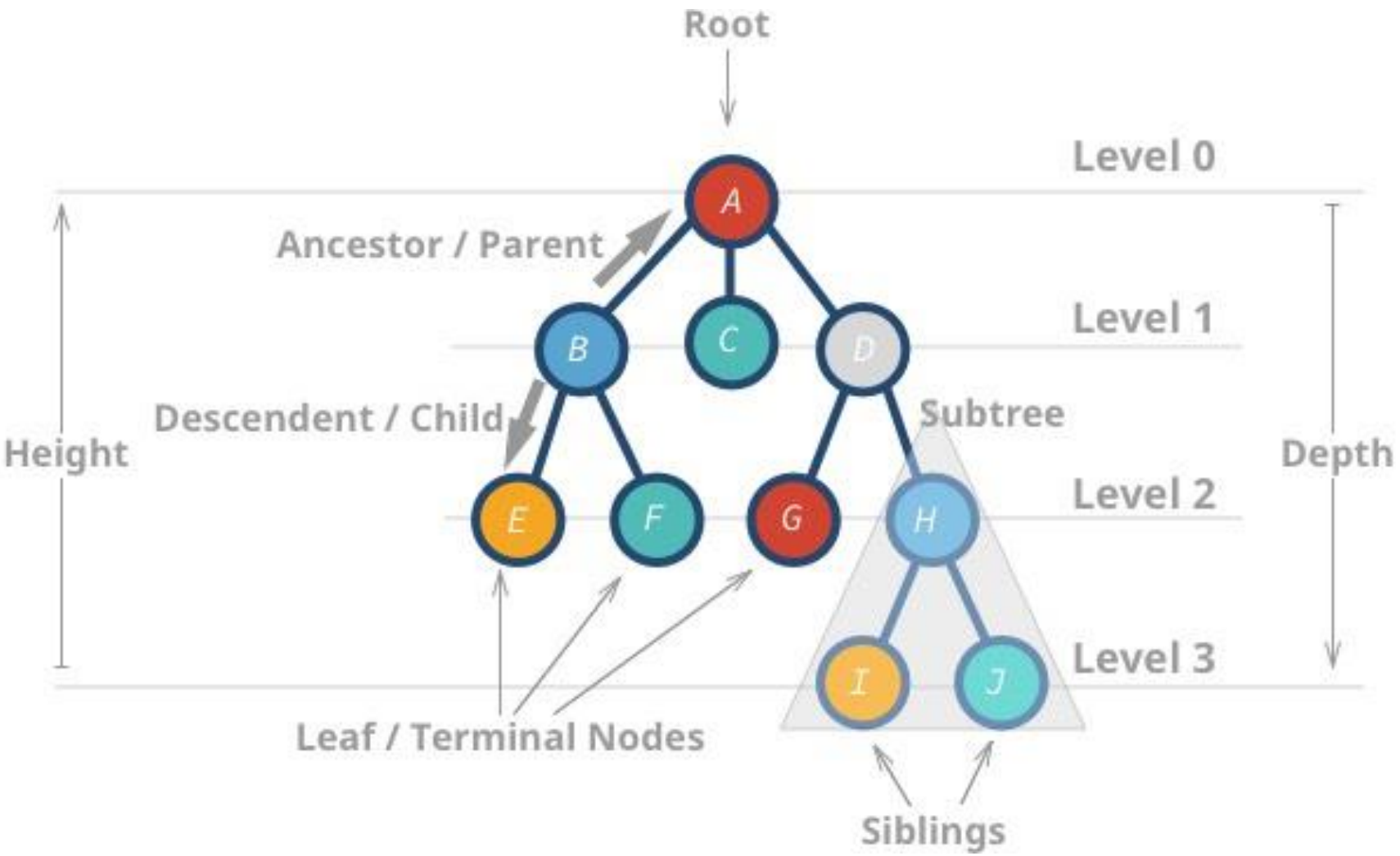
Tree Example – Tic Tac Toe

Örnek Ağaç:Bir oyundaki hamlelere ait ağaçlar



Ağaçlar: temel tanımlar ve terminoloji

- Kök (root), ağacın en üst düğümüdür
- Kenar (Edge), iki düğüm arasındaki bağlantıdır
- Çocuk (Child) bir ebeveyn (parent) düğümü olan bir düğümdür
- Ebeveyn (parent) , bir çocuk düğüme giden kenara sahip olan bir düğümdür
- Yaprak (Leaf), çocuk düğümü olmayan bir düğüm
- Yükseklik (Height) bir yaprağa olan en uzun mesafe. Ağacın yüksekliği (h), en uzaktaki yaprak ile kök arasındaki mesafe (kenar sayısı) 'dir.
- Derinlik (Depth): bir düğümden köke giden yolun uzunluğudur. Kökün derinliği 0. The depth of a node Y is
  - 0, if the Y is the root, or
  - 1 + the depth of the parent of Y
- Bir düğümün derecesi (Degree of a node): Düğüme bağlı çocuk sayısı.
- Ağacın derecesi (Degree of a tree): En yüksek dereceli düğüm ağacın derecesini belirtir.

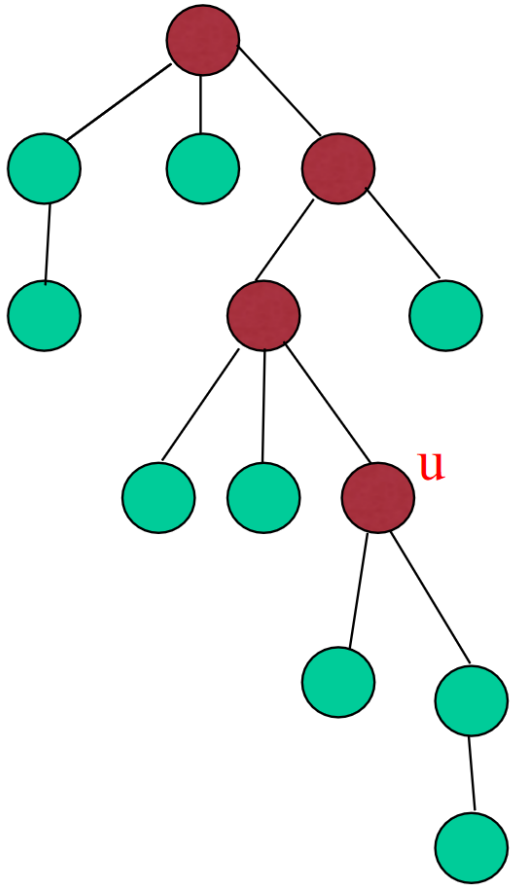


- Yandaki ağacın derinliği 3, örneğin H düğümünün derinliği 2.
- A düğümünü yüksekliği 3
- H düğümünün yüksekliği 1

https://adrianmejia.com/blog/2018/06/11/data-structures-for-beginners-trees-binary-search-tree-tutorial/

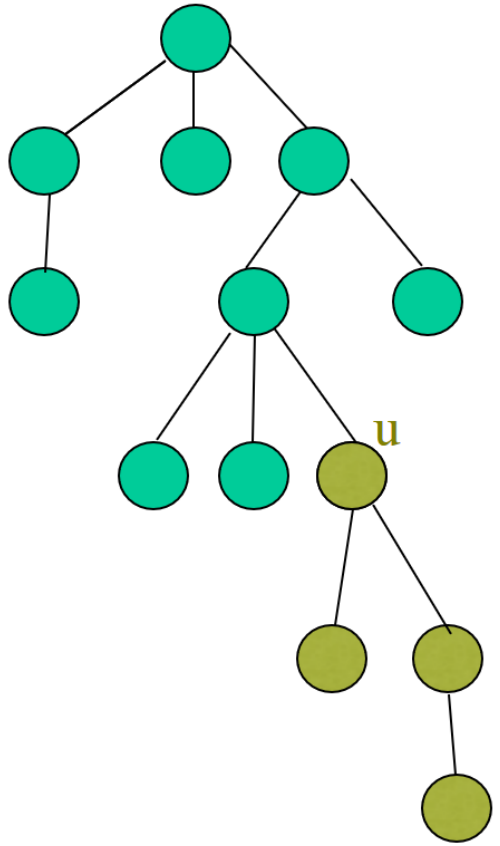
Trees

ancestors of u



Trees

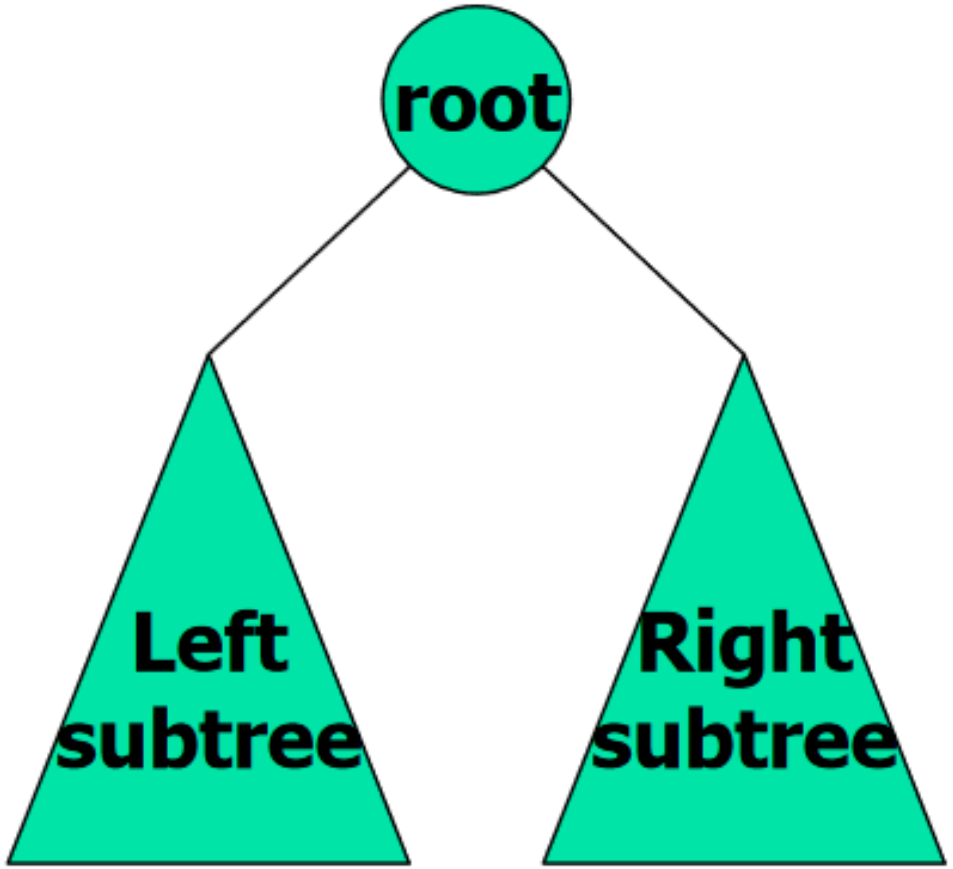
descendants of u



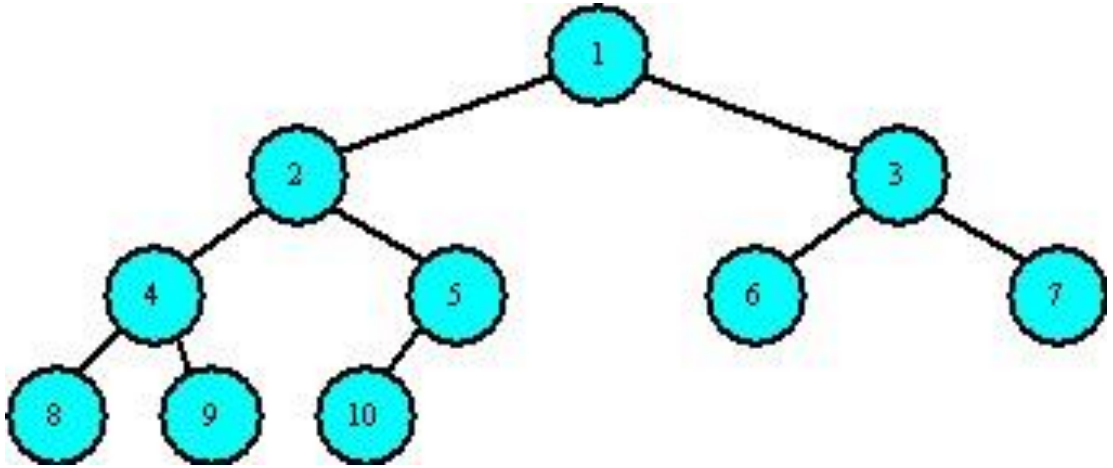
http://www.bowdoin.edu/~ltoma/teaching/cs210/spring09/Slides/210-Trees.pdf

İkili Ağaçlar (Binary Trees)

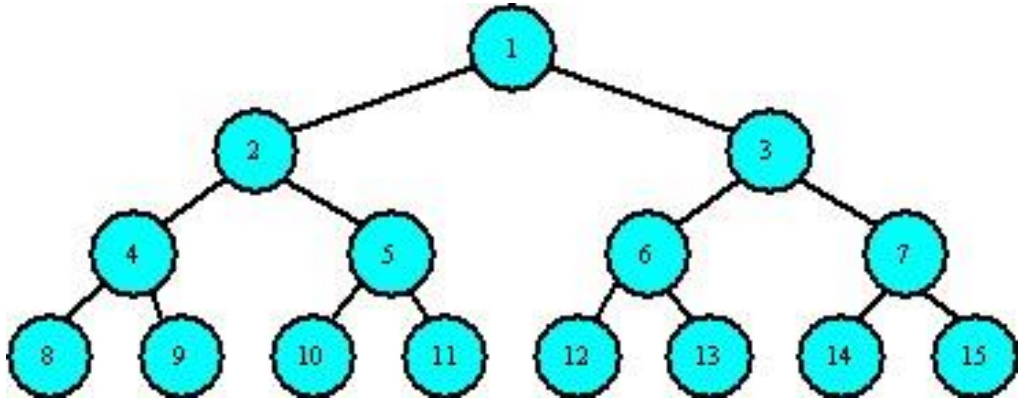
- Düğümlerin en fazla iki çocuğunun bulunduğu ağaç türüdür.



Complete Binary Tree

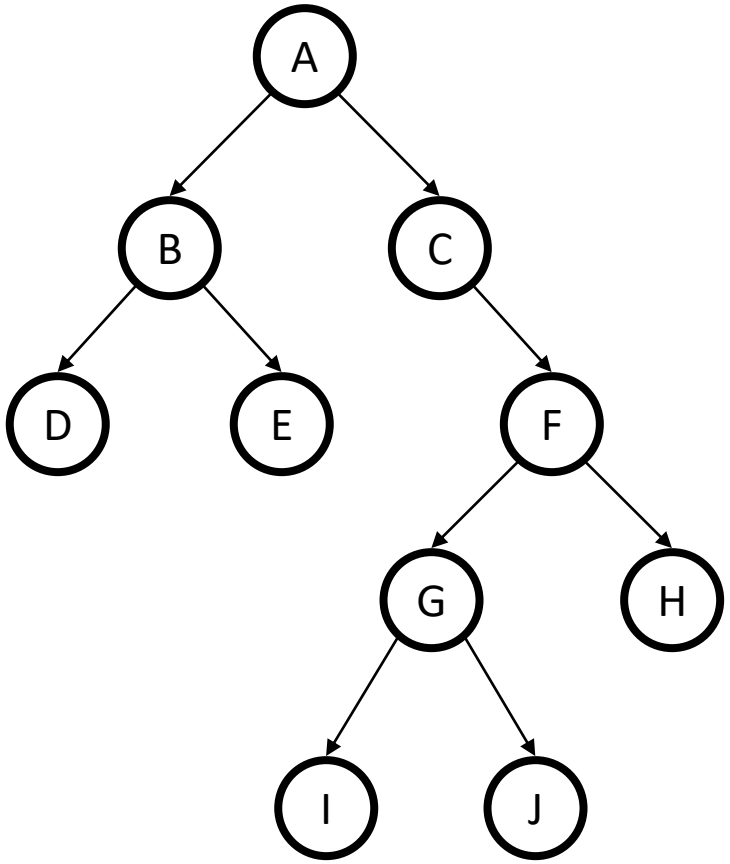


Full Binary Tree



Binary Trees

- Properties
  - max # of leaves = $2^{depth(tree)}$
  - max # of nodes = $2^{depth(tree)+1} - 1$
- We care a lot about the depth:
  - max depth = n-1
  - min depth = log(n) (why ?)
  - average depth for n nodes = $\sqrt{n}$ (over all possible binary trees)
- Representation:



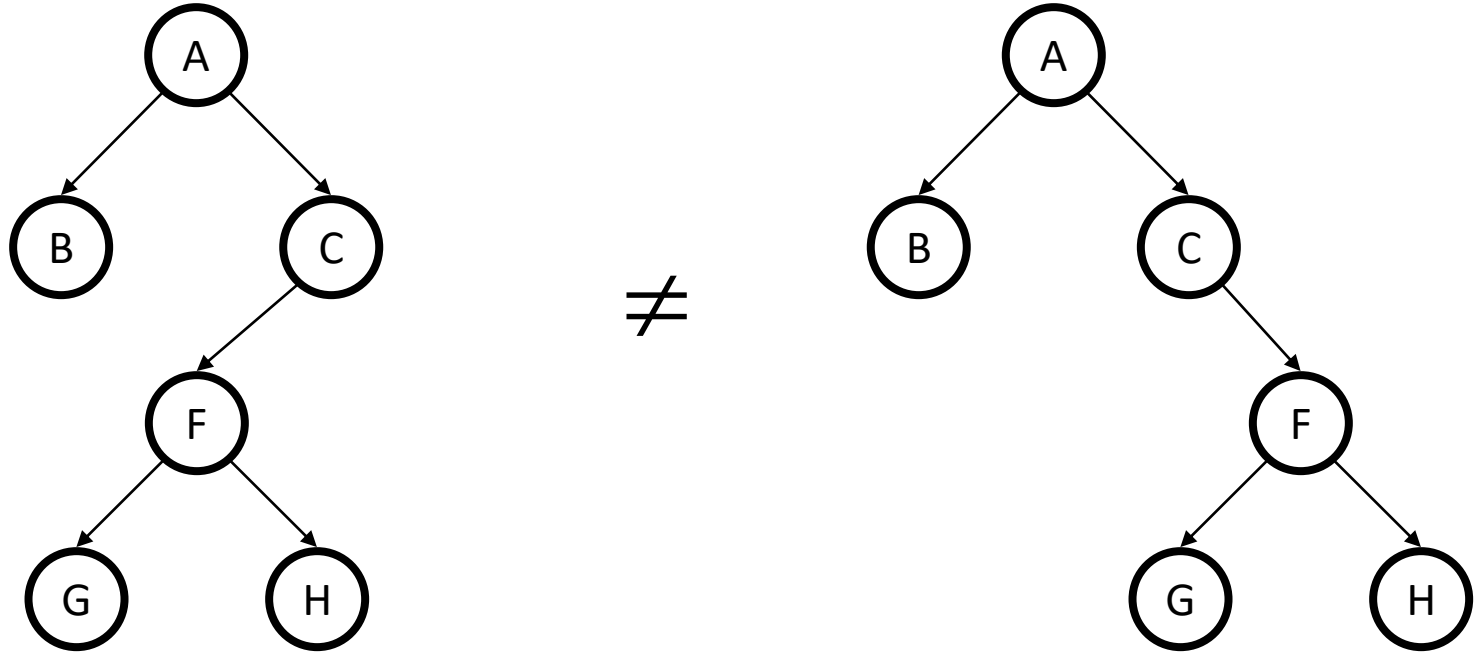
TreeNode:

Element	
Left	Right

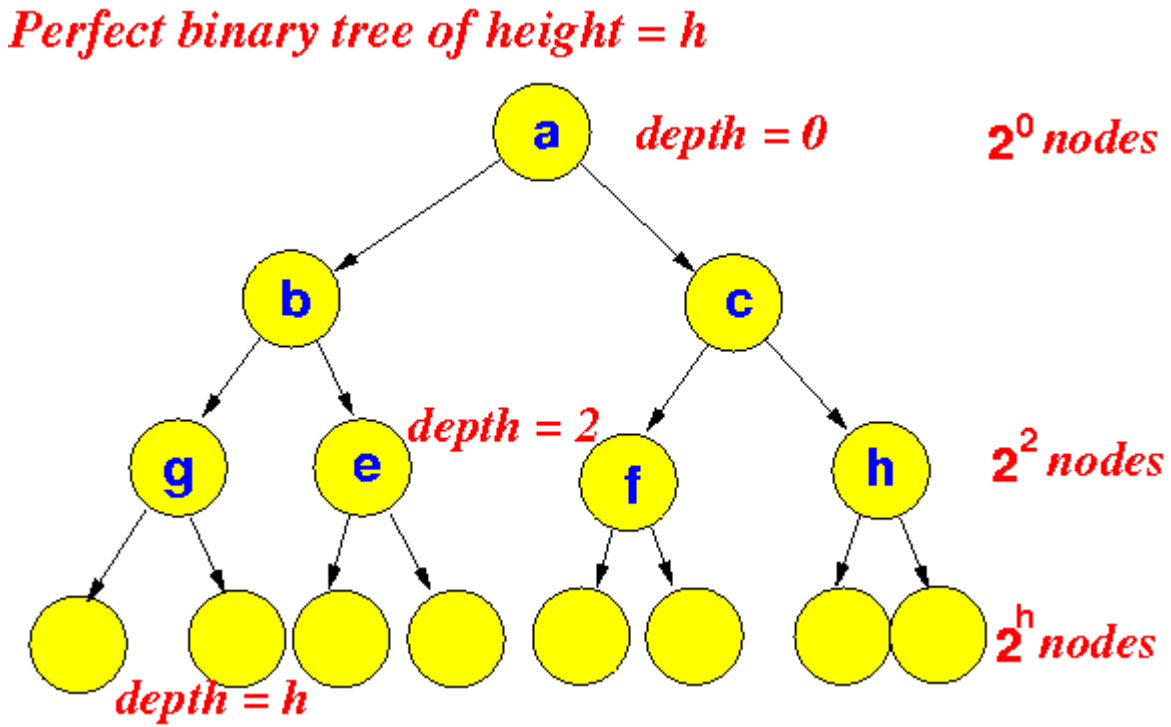
Binary Trees

Notice:

- we distinguish between *left* child and *right* child



İkili Ağaçlar (Binary Trees)



http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/9-BinTree/bin-tree.html

n düğüm sayısı, h yükseklik olmak üzere yüksekliğin değişim aralığı aşağıdaki gibi tanımlanır:

$N=2^{d+1}-1$

$N+1=2^{d+1}$

$d+1=\log_2(N+1)$

$d=\log_2(N+1)-1$

En fazla düğüm sayısı=$2^0+2^1+2^2+\ldots+2^h=2^{h+1}-1$

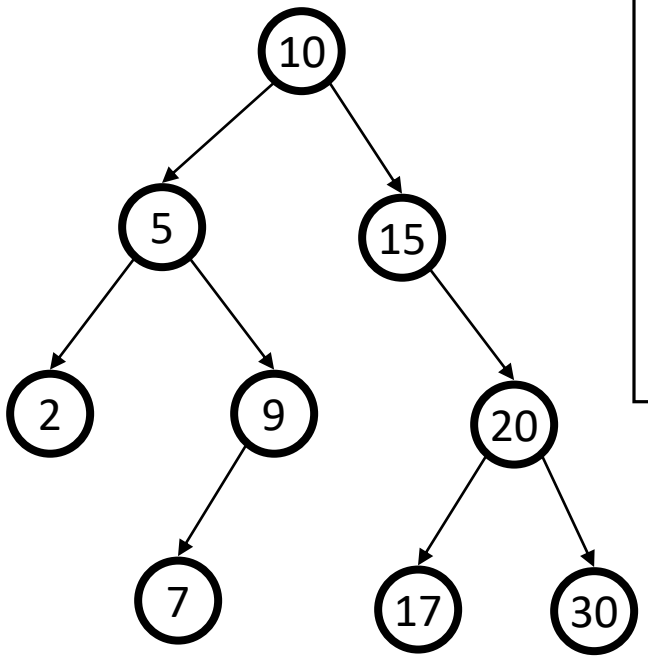
https://www.cise.ufl.edu/~sahni/cop3530/slides/lec204.pdf

İkili Arama Ağacı (Binary Search Tree-BST)

- İkili Arama Ağacı, aşağıdaki özelliklere sahip düğüm tabanlı bir ikili ağaç veri yapısıdır:
- Bir düğümün sol alt ağacı, yalnızca düğümün anahtarından daha küçük anahtarlı düğümleri içerir.
- Bir düğümün sağ alt ağacı, yalnızca düğümün anahtarından büyük anahtarlı sahip düğümleri içerir.
- Sol ve sağ alt ağaçların her biri de bir ikili arama ağacı olmalıdır
- Yinelenen düğümler olmamalıdır.

https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/

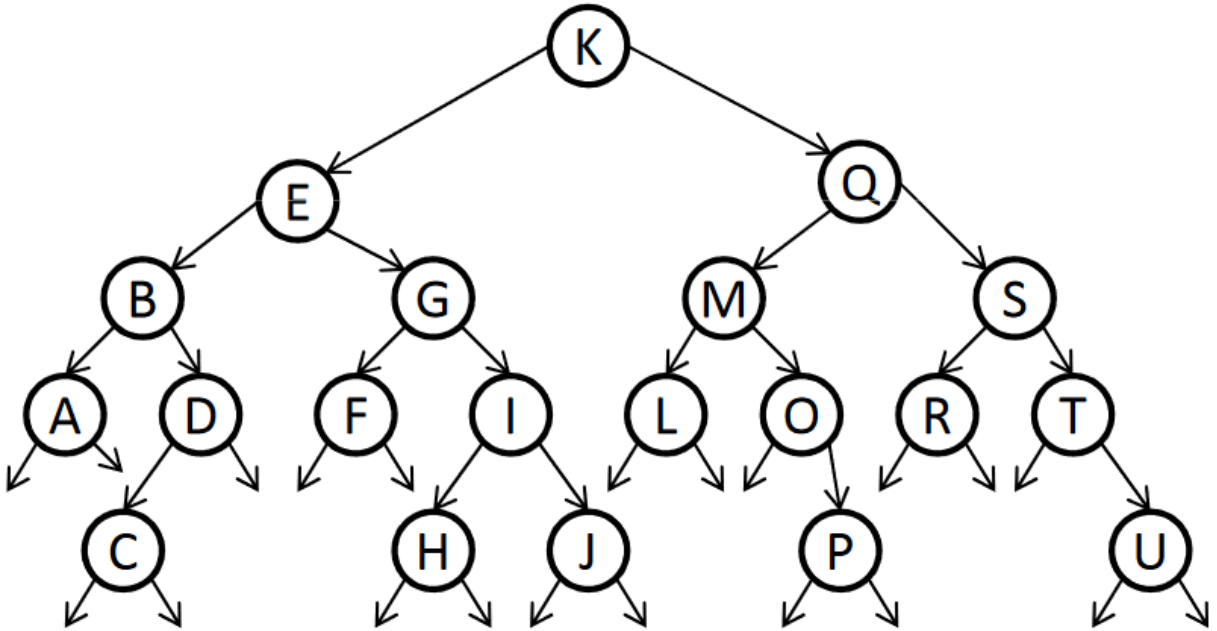
Eleman arama




```
Boolean find(int x, TreeNode T)
{ if  ( T == NULL )
      return false;
  if (x == T.Element)
      return true;
  if (x < T.Element)
     return find(x, T.Left);
  return find(x, T.Right);
}
```

What is the running time ?

Eleman arama

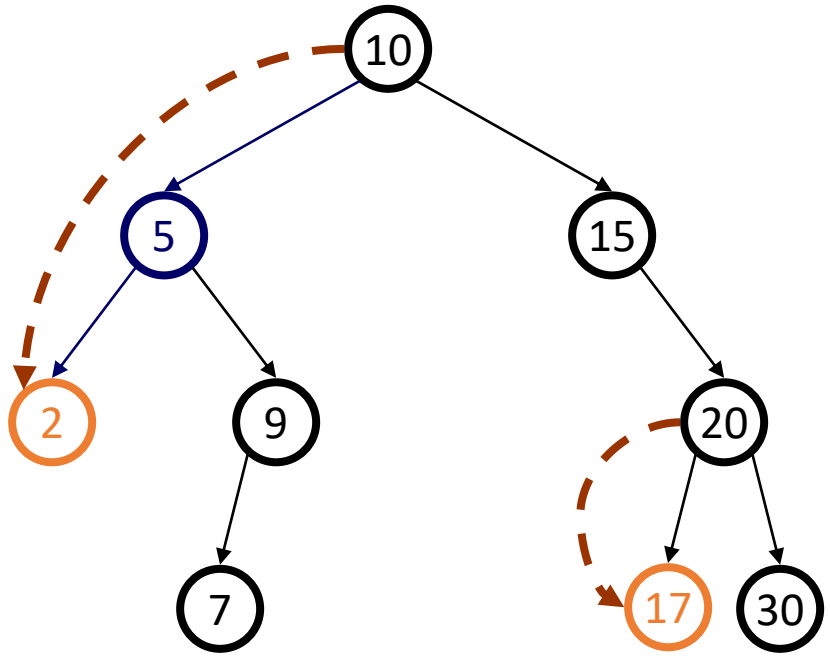


En iyi durumda ağaç dengelidir.
ElemanlarHesaplama karmaşıklığı: O(h)
h:yükseklik

http://www.cs.princeton.edu/courses/archive/spr11/cos423/Lectures/BinarySearchTrees.pdf

FindMin

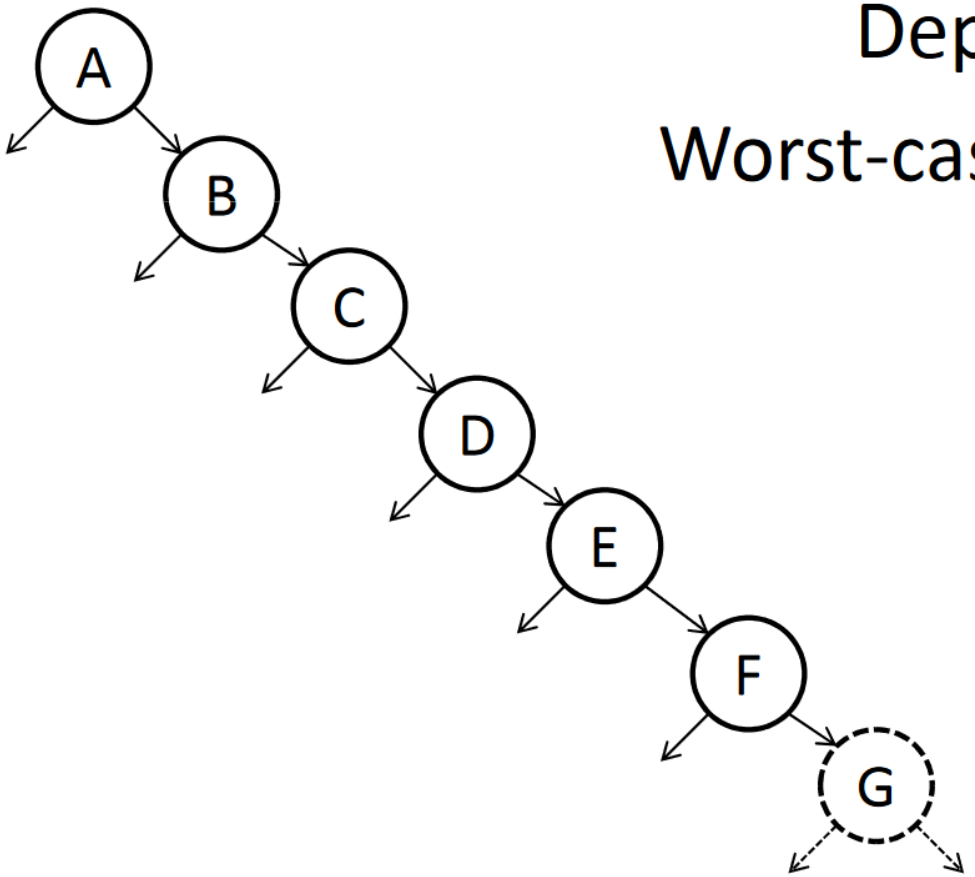
Ağacın minimum elemanını bulmak istersek ağacın en soluna, maksimum elemanını bulmak istersek en sağına gitmemiz gerekir.




```
TreeNode min(Node T) {
  if (T.Left == NULL)
    return T;
  else
    return min(T.Left); }
```

Eleman arama

Insert A, B, C, D, E, F, G,…



Depth of tree is $n - 1$.

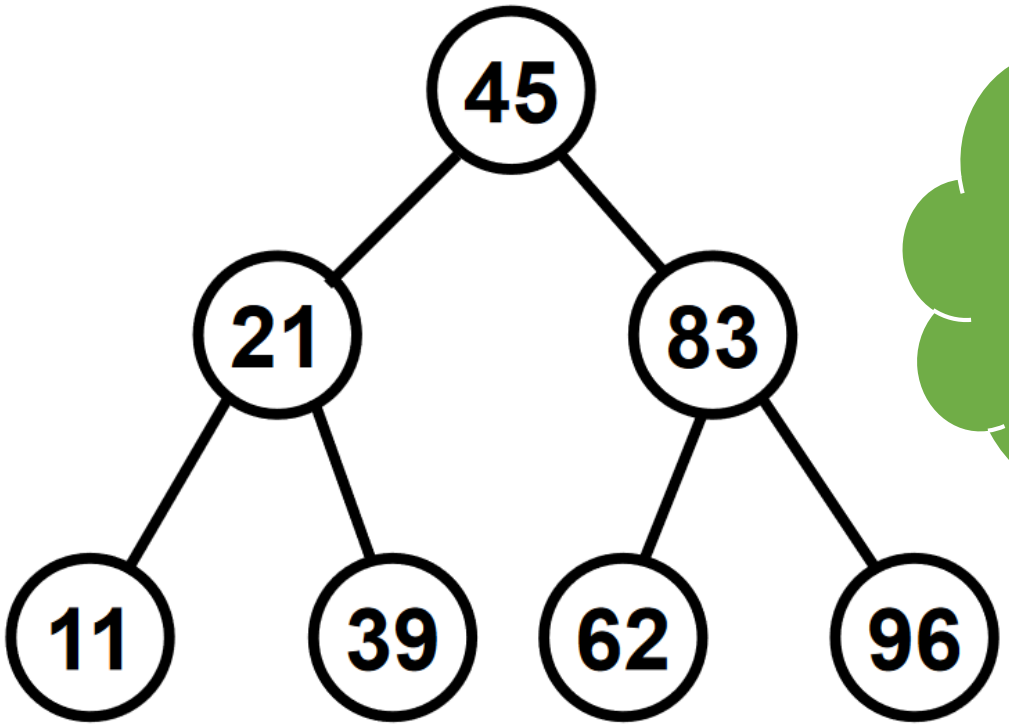
Worst-case access cost is n.

= list!

En kötü durumda ağaç doğrusal listeye dönüşür. N elemanlı bir ağaçta hesaplama karmaşıklığı O(n) olarak ortaya çıkar.

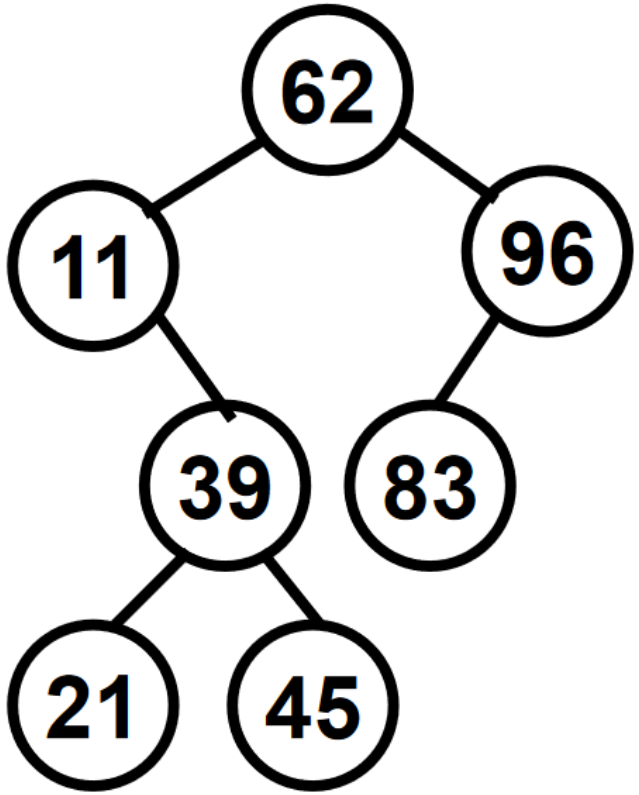
Eleman ekleme- örnek

45 21 39 83 62 96 11



Aynı kümeyi farklı sıra ile girersek farklı BST ortaya çıkar

62 96 11 39 21 83 45



https://www3.cs.stonybrook.edu/~cse214/lecture_slides/unit6.pdf

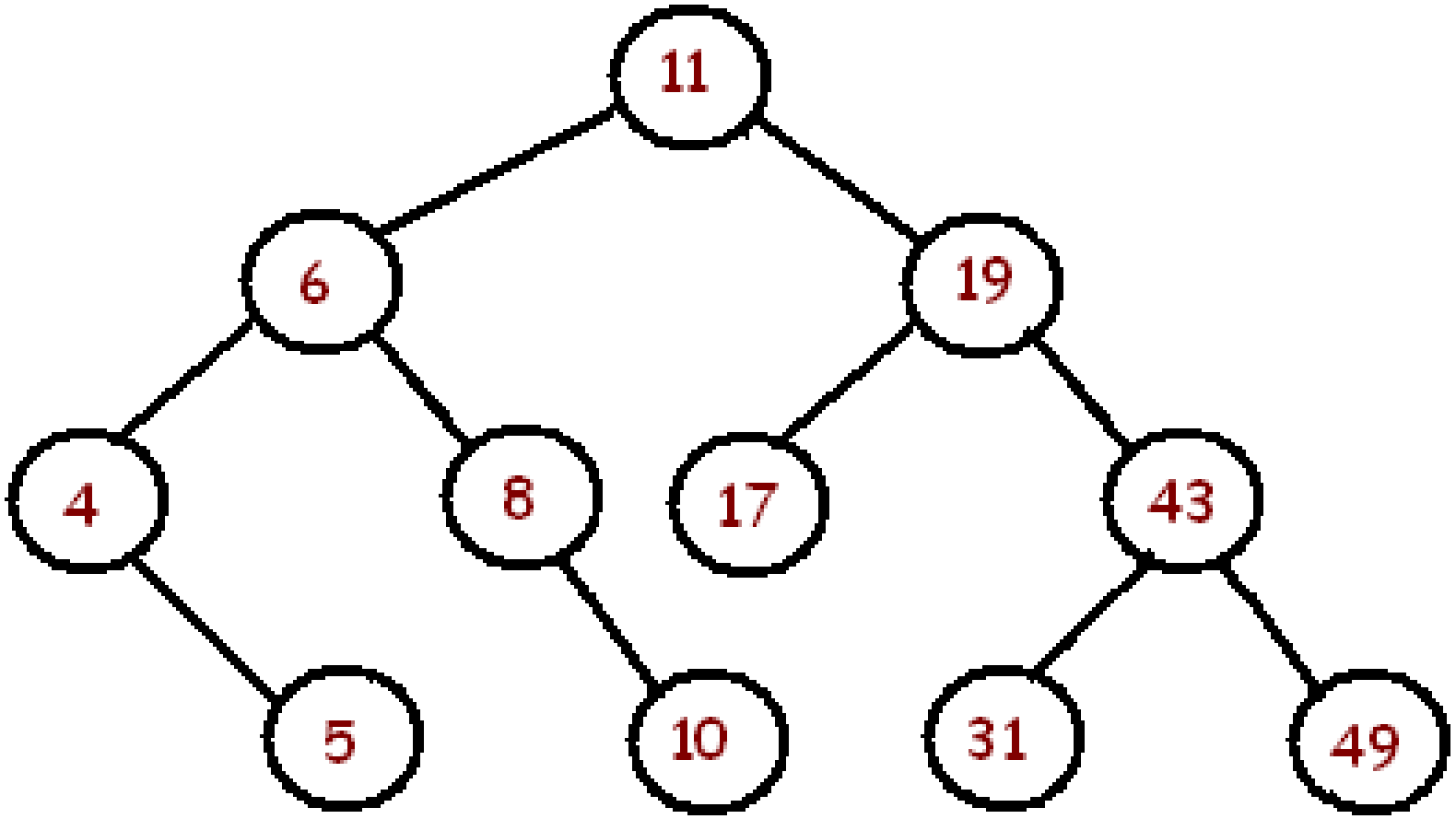
Eleman ekleme

Soru: Aşağıda verilen elemanları sırayla ikili arama ağacına eklersek, elde edilecek iki ağacın şeklini çiziniz.

Elemanlar: 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

Eleman ekleme

Elemanlar: 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

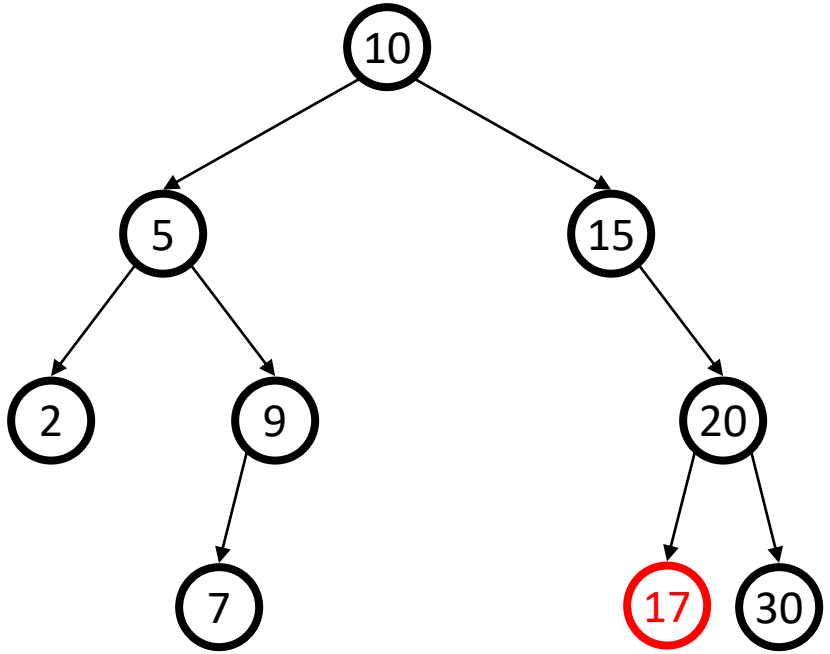


Eleman silme

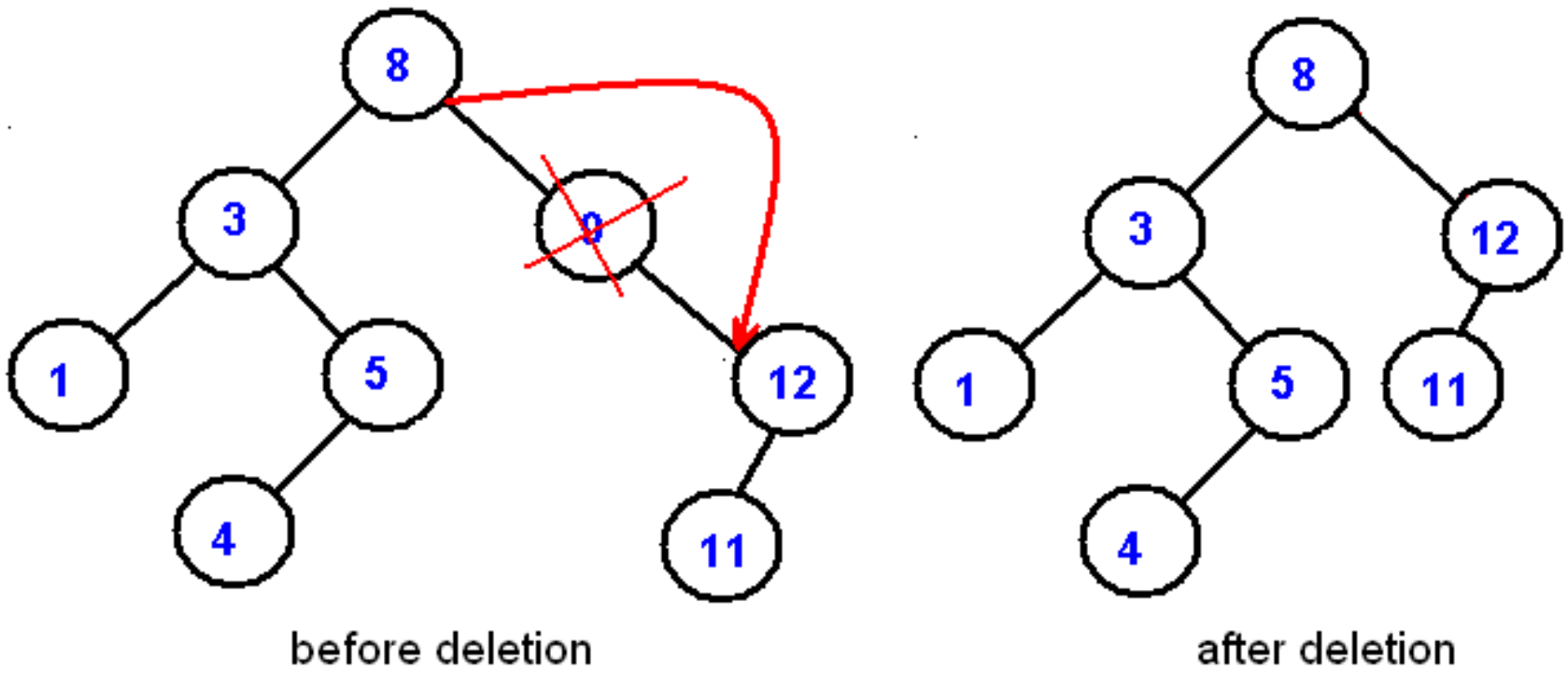
- Silinecek düğümün durumuna göre işlemler yapılır.
- Elemanın yaprak olması. (En basit durum)
- Düğümün bir çocuğunun bulunması.
- Düğümün iki çocuğunun bulunması.

Deletion - Leaf Case

Delete(17)

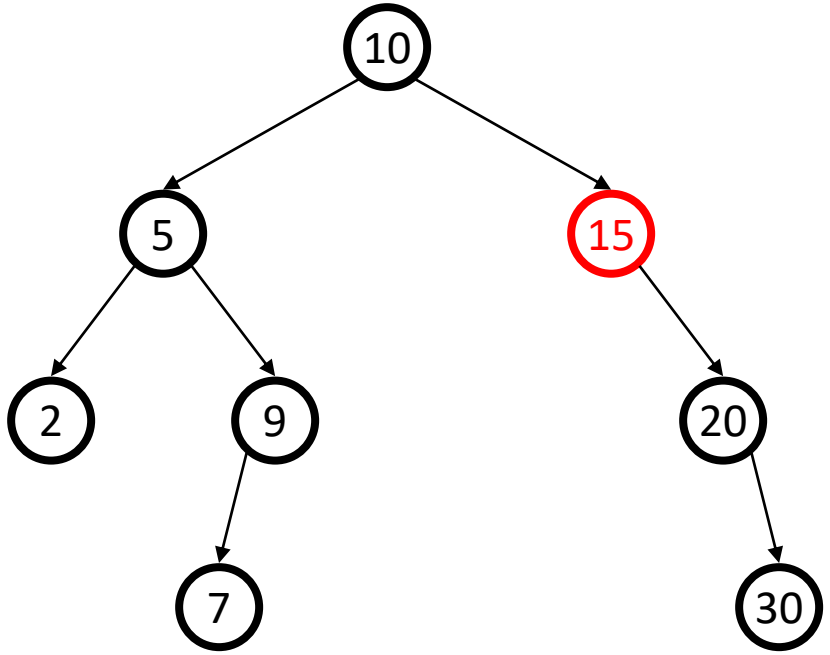


Eleman silme- Düğümün bir çocuğunun bulunması



Deletion - One Child Case

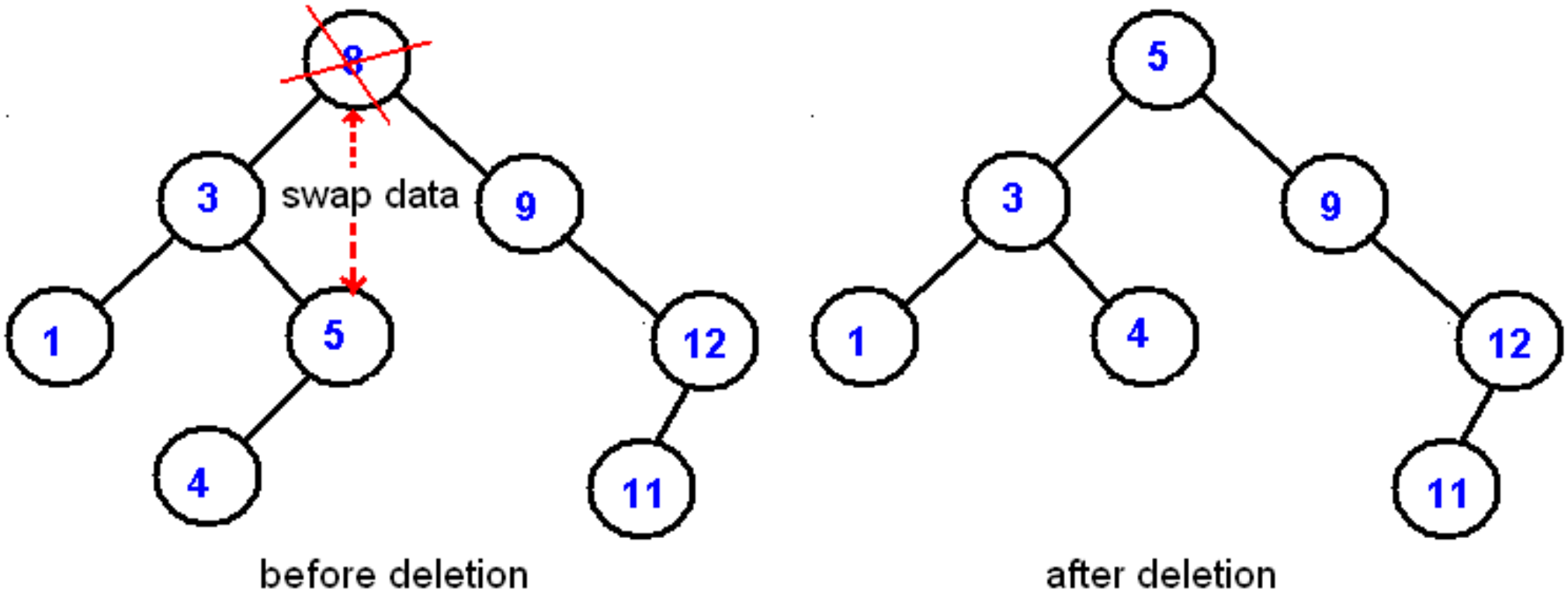
Delete(15)



Eleman silme- Düğümün iki çocuğunun bulunması

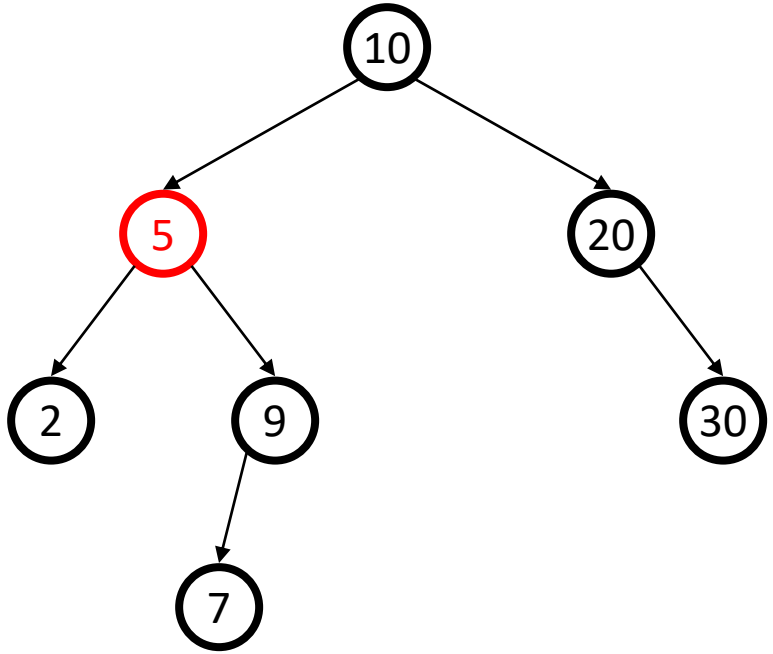
- Silinecek düğümü sol alt ağacın en büyük düğümü ile değiştirilir.
- Veya silinecek düğüm, sağ alt ağacın en küçük düğümü ile değiştirilebilir.

Eleman silme- Düğümün bir çocuğunun bulunması



Deletion - Two Children Case

Delete(5)

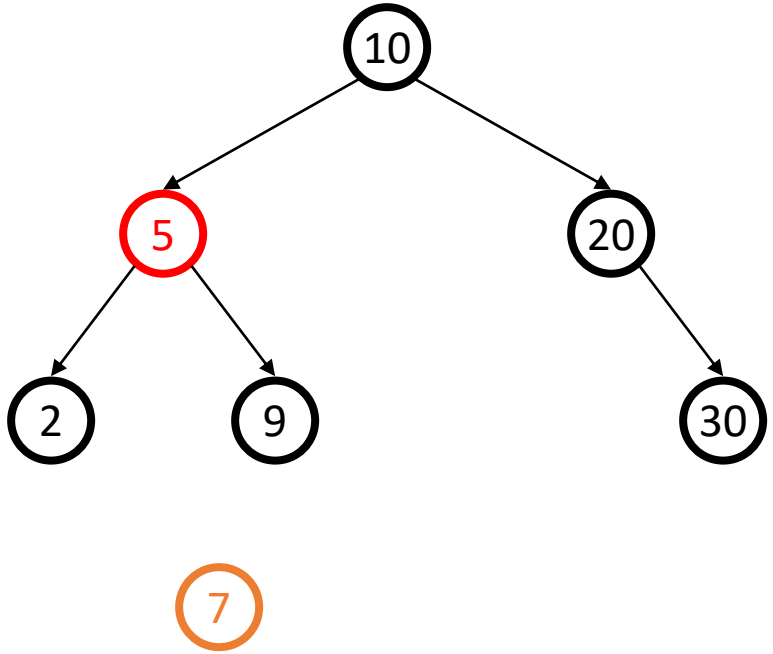


replace node with value guaranteed to be between the left and right subtrees:
the successor

https://courses.cs.washington.edu/courses/cse326/

Deletion - Two Children Case

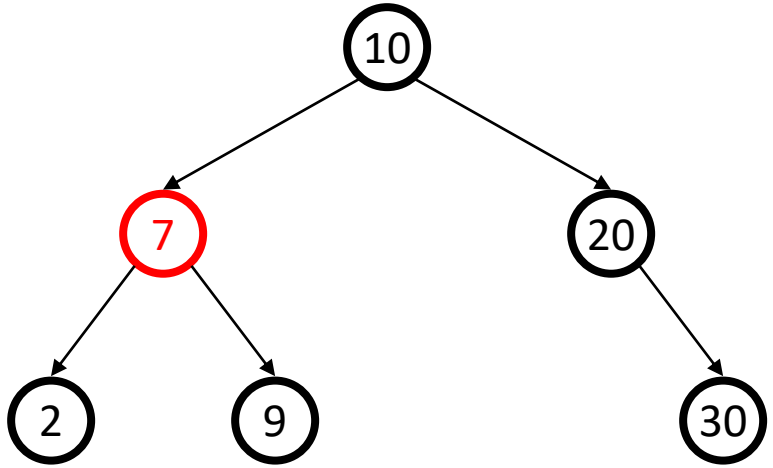
Delete(5)



always easy to delete the successor – always has either 0 or 1 children!

Deletion - Two Child Case

Delete(5)



Finally copy data value from deleted successor into original node

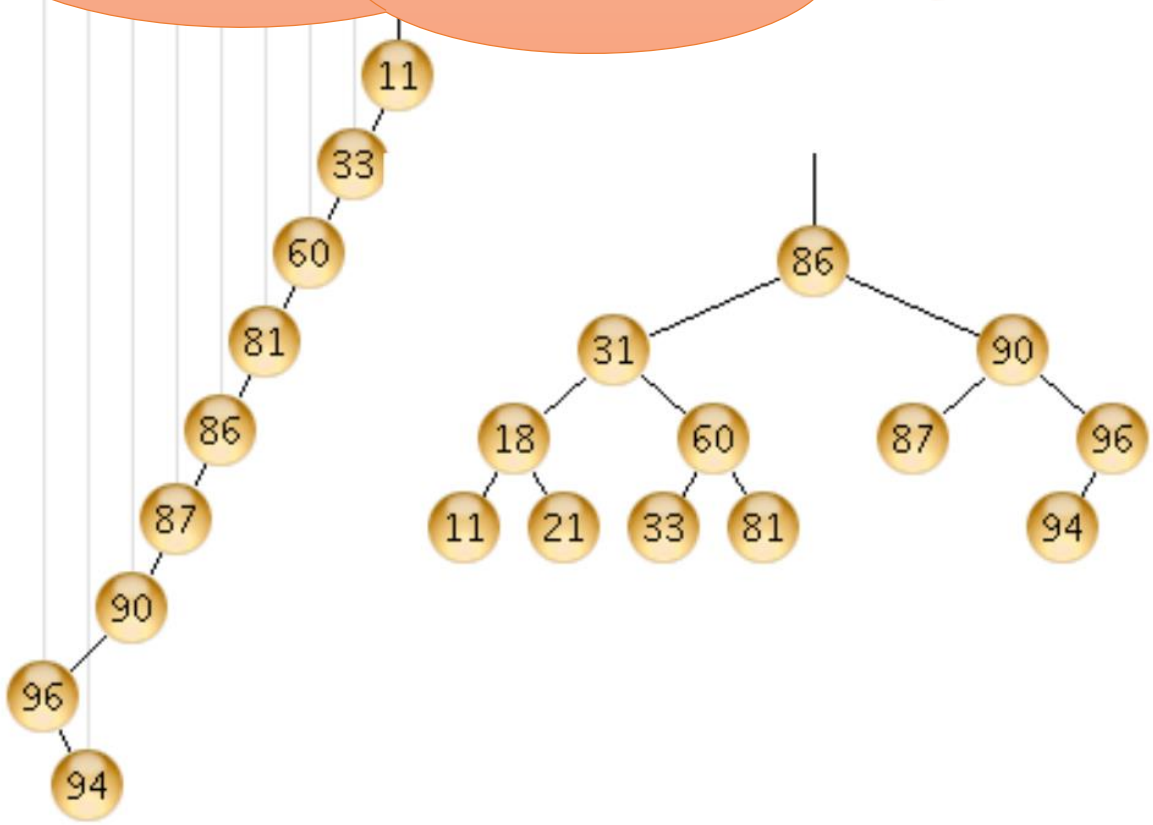
What is the cost of a delete operation ?
Can we use the *predecessor* instead of successor ?

BST performance

- Because of search property, all operations follow one root-leaf path
  - insert: O(h)
  - delete: O(h)
  - search: O(h)

Ekleme, silme ve arama işlemlerinin hesaplama karmaşıklığı ağacın yüksekliği ile doğru orantılıdır. Ancak yükseklik, ***aynı veri kümesi için*** düğümlerin eklenme sıralarına göre aşağıdaki şekillerde görüldüğü gibi değişebilir.
(log(n+1)-1)<=h<=n-1

- We know that in a tree of n nodes
  - h >= lg (n+1) - 1
  - h <= n-1
- So in the worst case h is O(n)
  - BST insert, search, delete: O(n)
  - just like linked lists/arrays



Örnek: Eleman ekleme ve silme

Soru: Aşağıda verilen elemanları sırayla ikili arama ağacına eklersek, elde edilecek iki ağacın şeklini çiziniz. İçerisinde 10 verisi bulunan düğümü silindiğinde yeni ağacı iki farklı durumu ele alarak çiziniz.

Elemanlar: 10, 5, 7, 17, 5, 11, 4, 19, 41, 45, 30

Ağaç dolaşım (Tree traversal):

- **Depth-First Search (önce boyuna arama, önce derinlik)**
- Derinlik-ilk arama, bir sonraki kardeşleri keşfetmeden önce her çocuk için mümkün olduğunca derine inen bir geçiş türüdür.
- Çeşitleri: in-order, pre-order ve post-order.

- **Breadth-First Search (önce enine arama, önce genişlik)**
- Bir sonraki seviyeye gitmeden önce bir seviyenin tüm düğümlerini ziyaret eder.
- Bu tür bir geçiş, level-order (seviye-düzeni) olarak da adlandırılır ve ağacın kökünden başlayarak soldan sağa tüm düzeylerini ziyaret eder.

Preorder Traversal (Practice):

Algorithm Preorder(tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Inorder Traversal (Practice):

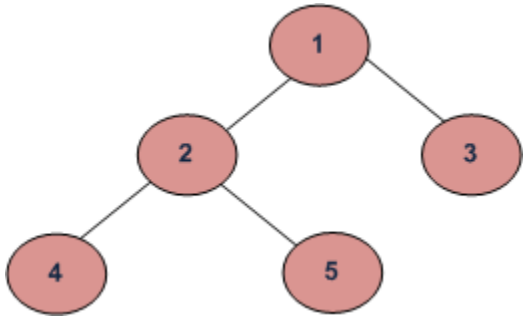
Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Postorder Traversal (Practice):

Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root.



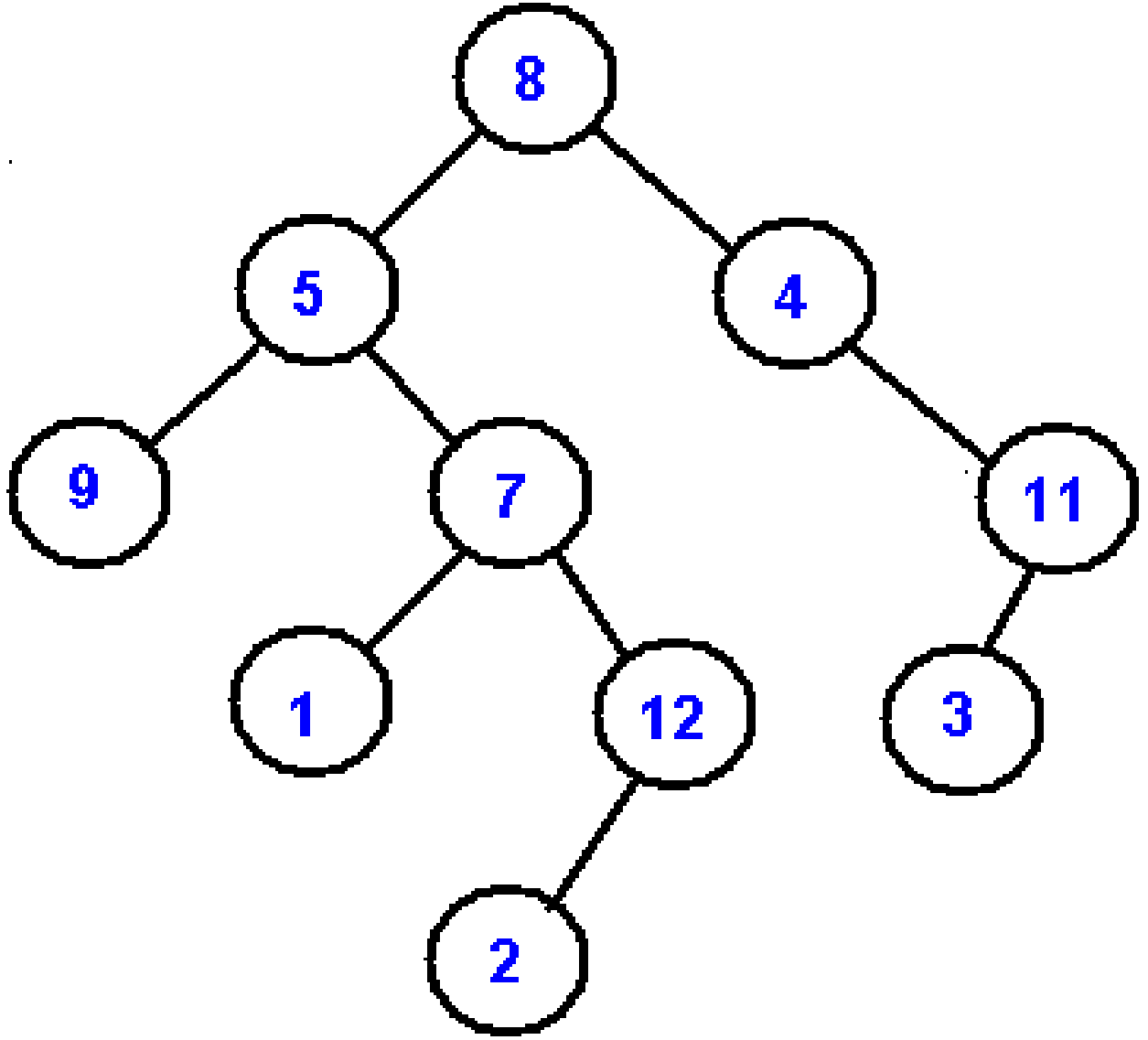
Depth First Traversals:
(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Breadth First or Level Order Traversal : 1 2 3 4 5

https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

Ağaç dolaşım (Tree traversal) örneği:

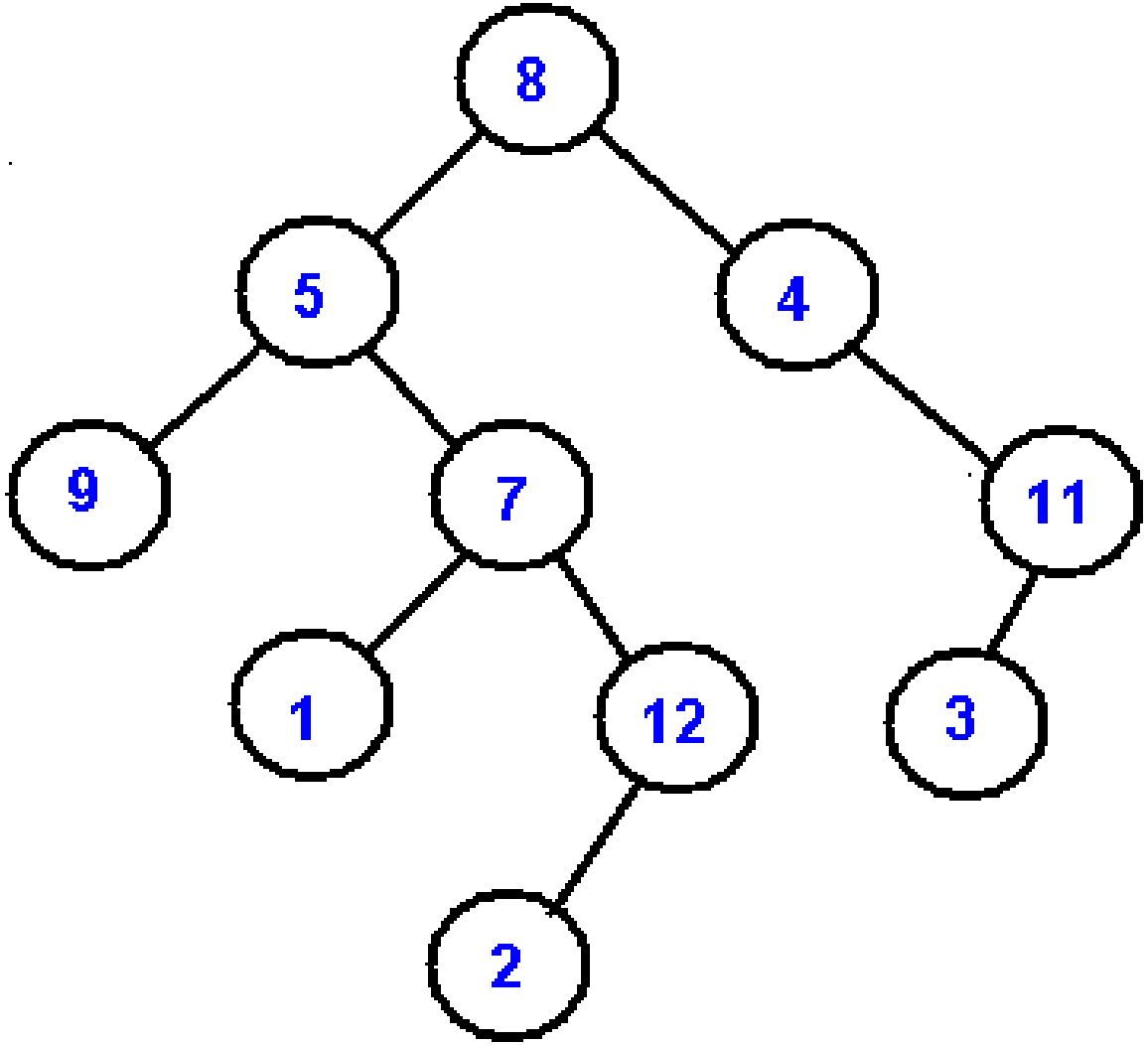
Yandaki ağaç için preorder,inorder, postorder ve levelorder yöntemlerine göre dolaşım sıralarını yazınız.



Ağaç dolaşım (Tree traversal) örneği:

Yandaki ağaç için preorder,inorder, postorder ve levelorder yöntemlerine göre dolaşım sıralarını yazınız.

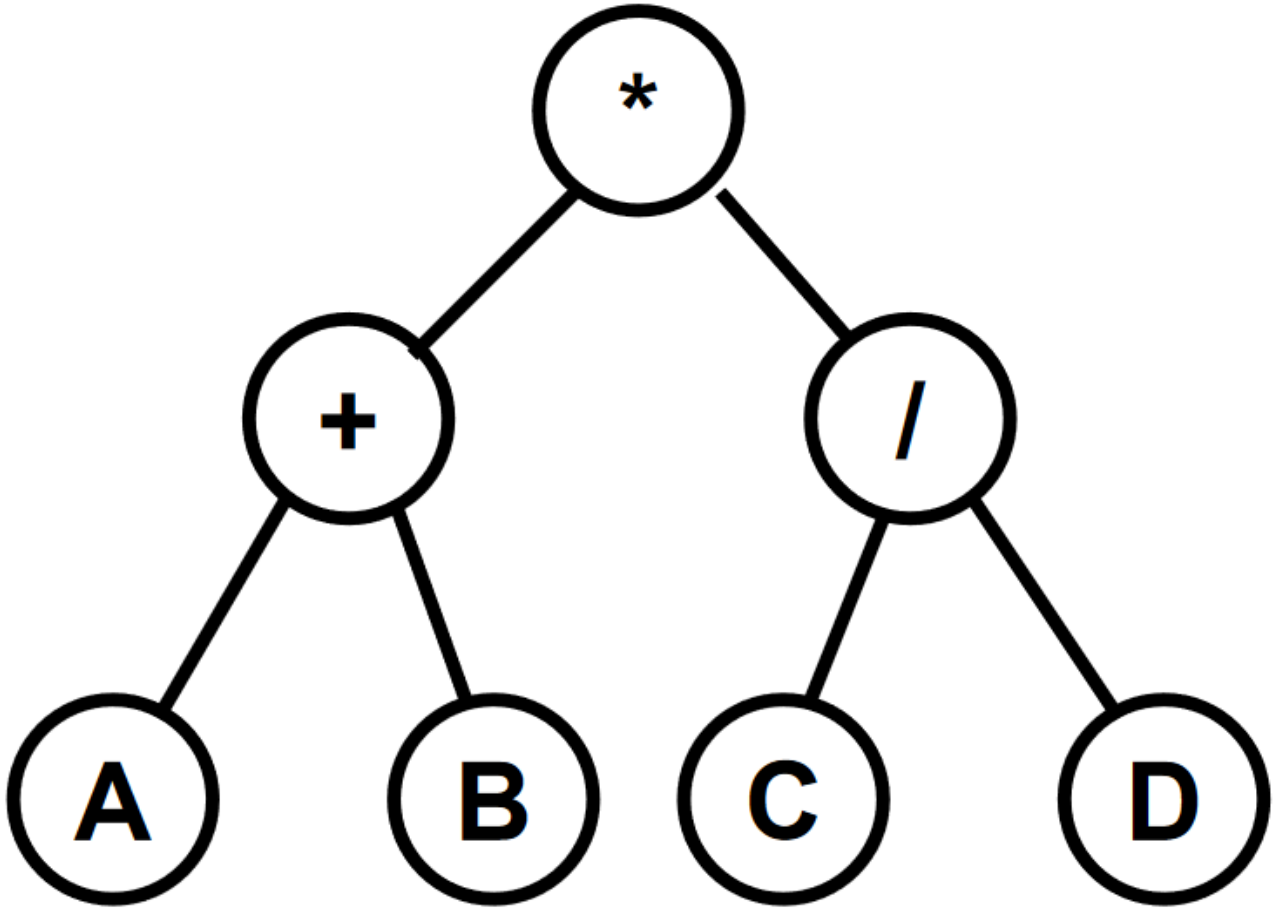
PreOrder : 8, 5, 9, 7, 1, 12, 2, 4, 11, 3
InOrder : 9, 5, 1, 7, 2, 12, 8, 4, 3, 11
PostOrder : 9, 1, 2, 12, 7, 5, 3, 11, 4, 8
LevelOrder: 8, 5, 4, 9, 7, 11, 1, 12, 3, 2



https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html

Traversal Example

(expression tree)



preorder

***+AB/CD**

inorder

A+B*C/D

postorder

AB+CD/*

https://www3.cs.stonybrook.edu/~cse214/lecture_slides/unit6.pdf