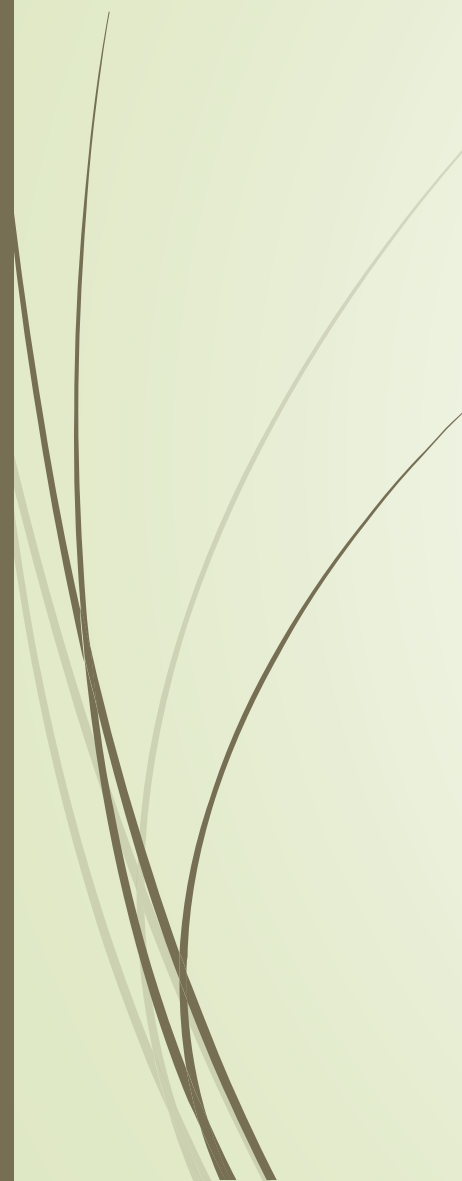



# Berilganlarning chiziqsiz strukturalari. Daraxtlar.

Reja:

1. Daraxt tushunchasi.
2. Daraxt turlari.
3. Binar daraxtni to'liq aylanib chiqish algoritmlari.

**Ma'ruzachi: M.Tojiyev**



Chiziqli ma'lumotlar tuzilmasida ma'lumotlar ketma-ket tartibda joylashadi. Chiziqli bo'lmagan ma'lumotlar strukturasi ma'lumotlar tasodifiy tartibda joylashadi. Daraxt bu juda ko'p foydalaniladigan tuzilma hisoblanadi. Dasturiy ilovalarda qo'llaniladigan mashhur chiziqli bo'lmagan ma'lumotlar strukturasi.



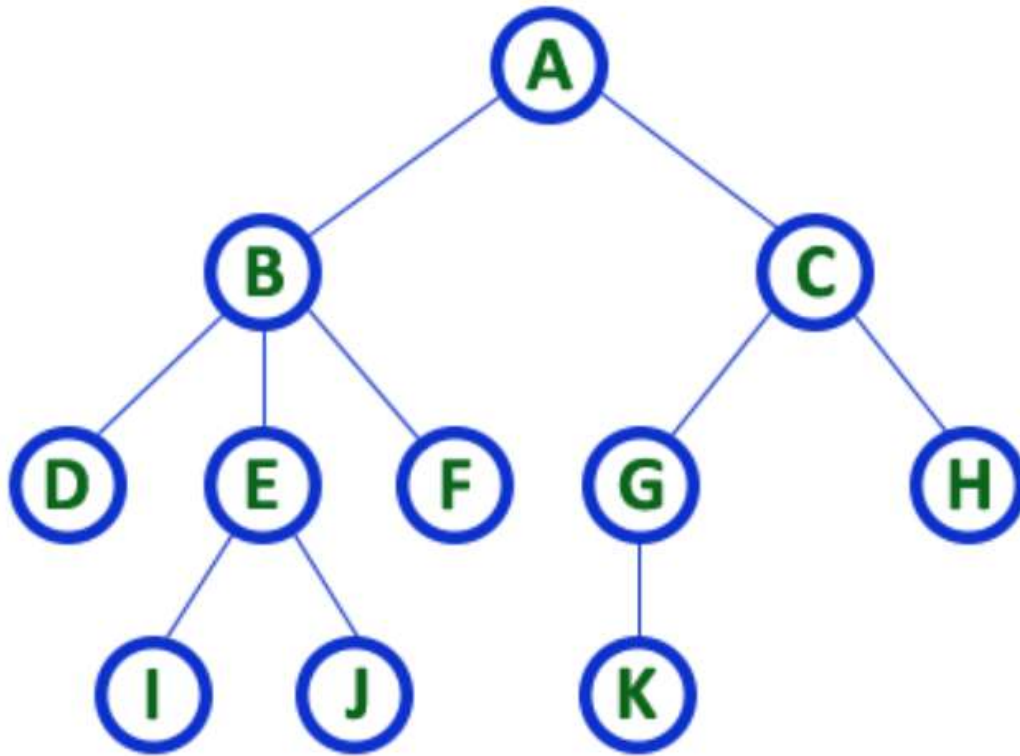
## 1. Daraxt tushunchasi:

Daraxt - bu ma'lumotlarni ierarxik tuzilishga tartibga soluvchi chiziqli bo'lmagan ma'lumotlar strukturasi va bu esa uning rekursiv ta'rifidir.

Kirishning nol darajasiga ega bo'lgan uch **daraxtning** ildizi, chiqish nol darajaga ega tugunlar esa **barglar** deb nomlanadi.

Daraxtlar ko'pincha ma'lumotlar o'rtasidagi ierarxik munosabatlarni ifodalash uchun ishlatiladi, masalan, kompyuterdagi fayl tizimi.

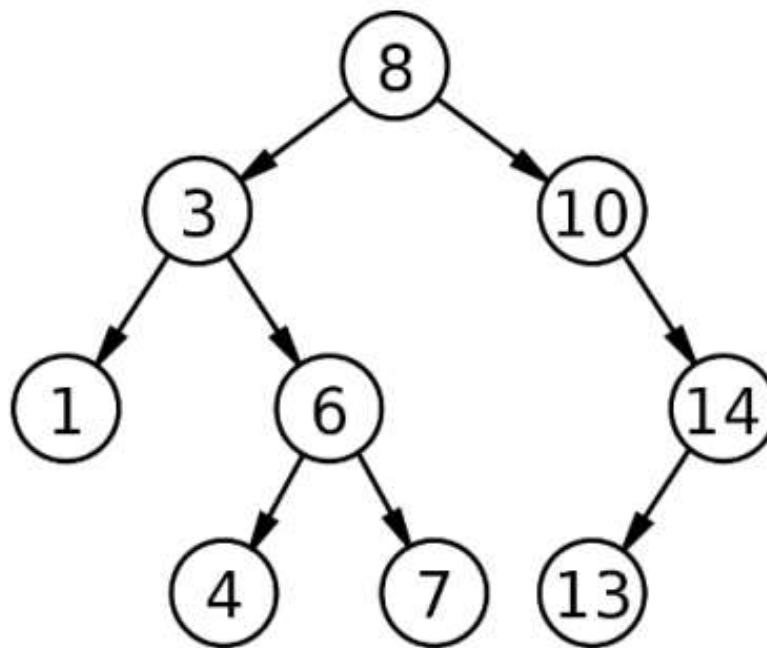
Daraxt ma'lumotlar tuzilmasida, agar bizda  $N$  sonli tugunlar bo'lsa, u holda biz eng ko'p  $N-1$  havolaga ega bo'lishimiz mumkin.



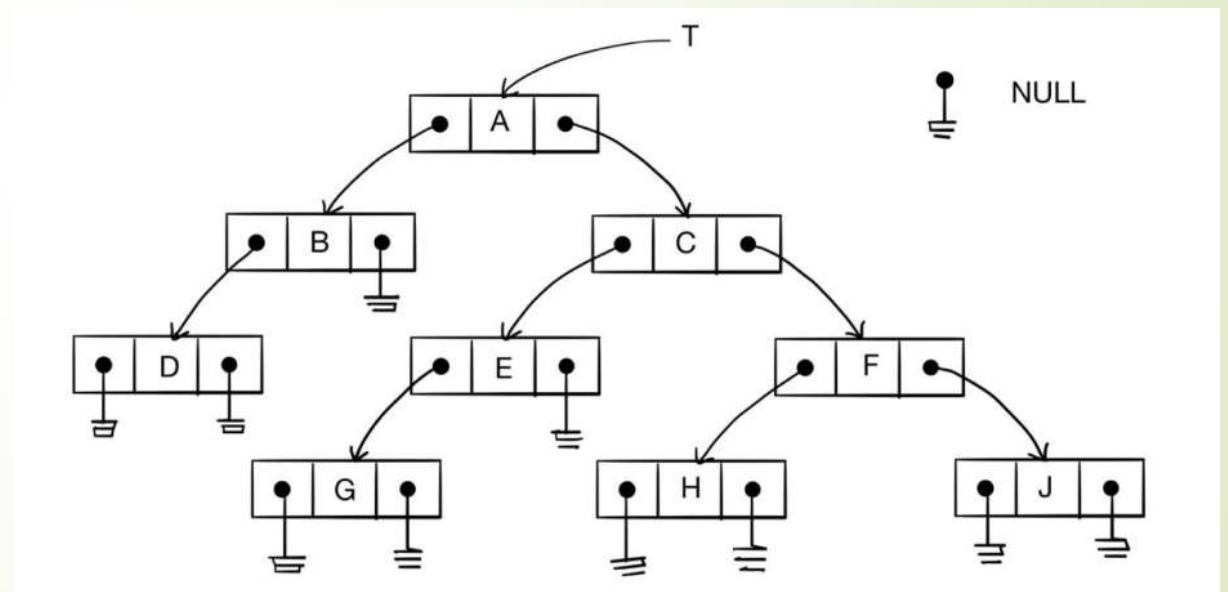
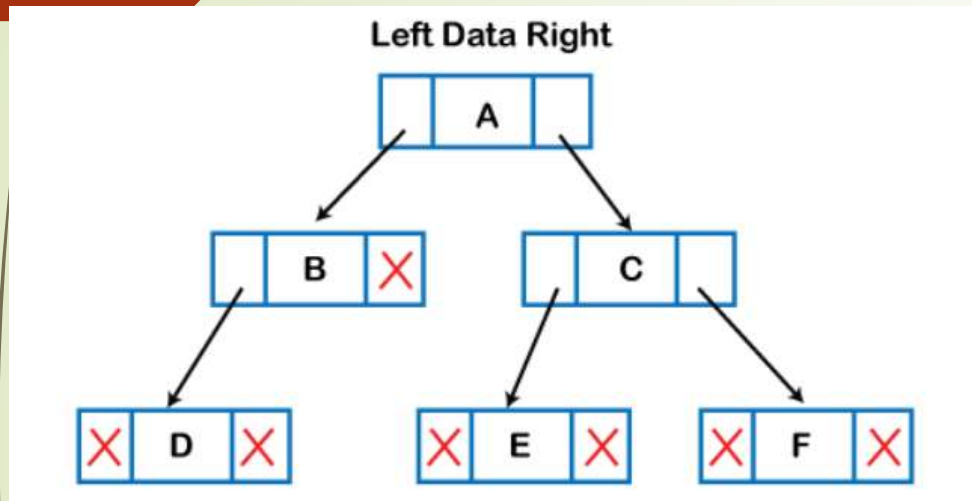
**TREE with 11 nodes and 10 edges**

- In any tree with ' $N$ ' nodes there will be maximum of ' $N-1$ ' edges
- In a tree every individual element is called as '**NODE**'

**Yo'naltirilgan (oriyentirlangan) daraxt** - bu faqat bitta vertical kirish nol darajasiga ega bo'lgan (boshqa yo'ylar unga olib kelmaydigan), boshqa uchlarning kirish darajasi 1 bo'lgan siklik orgraf (sikllarni o'z ichiga olmaydigan yo'naltirilgan graf).

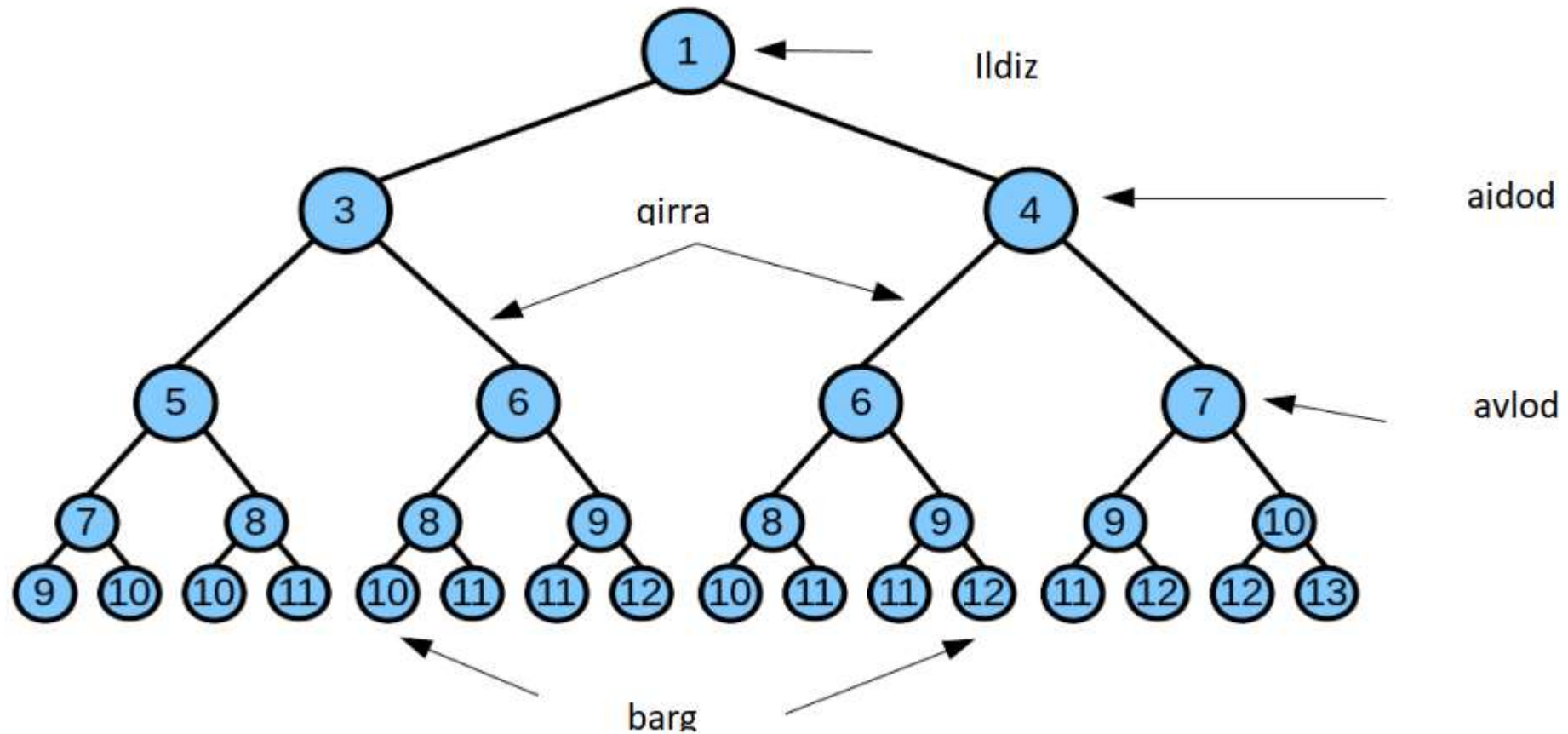


# Daraxtni dasturda ko'rinishi



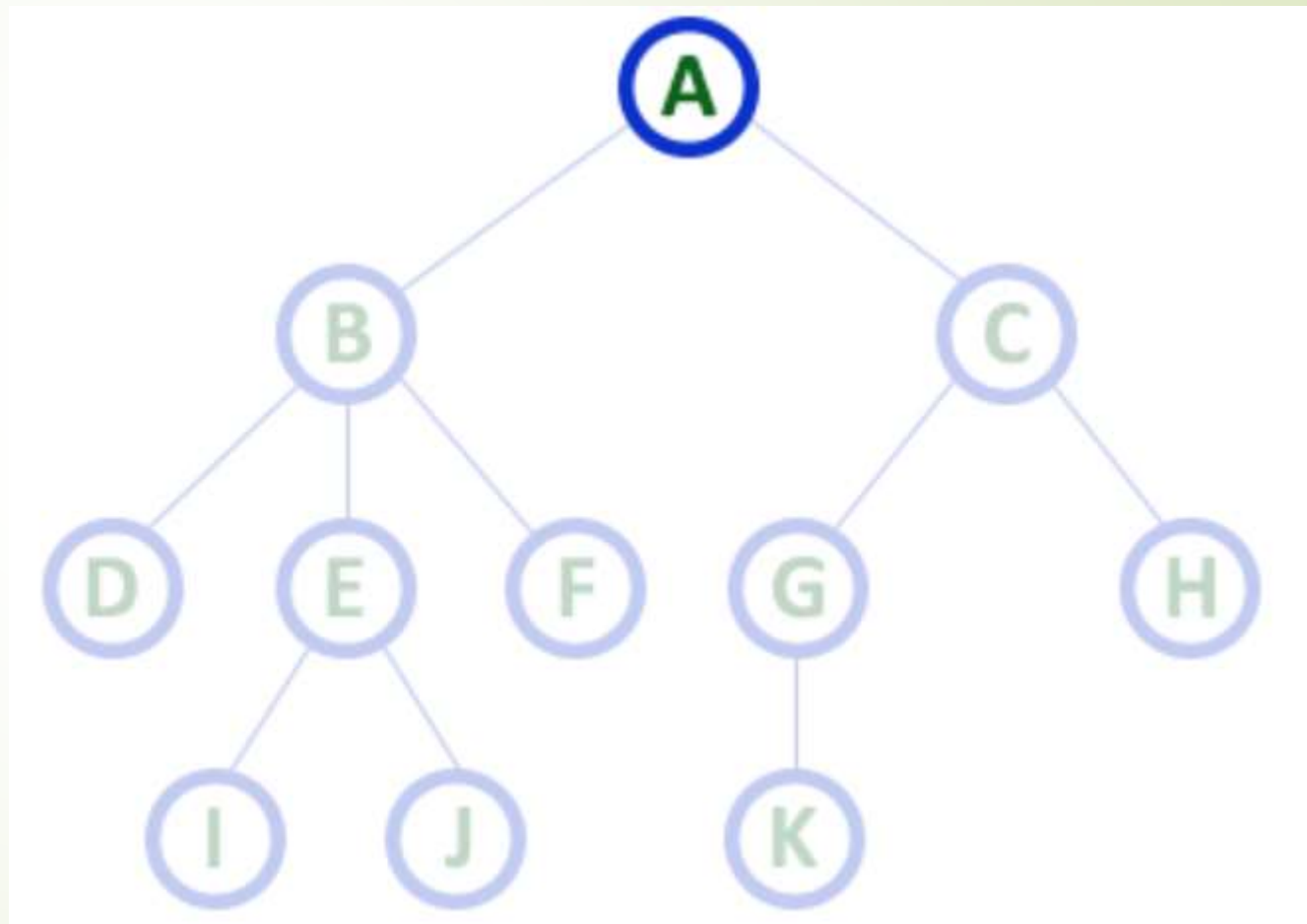


# DARAXTNING ASOSIY TUSHUNCHALARI



## ➤ **Ildiz tuguni** - daraxtning eng yuqori tuguni

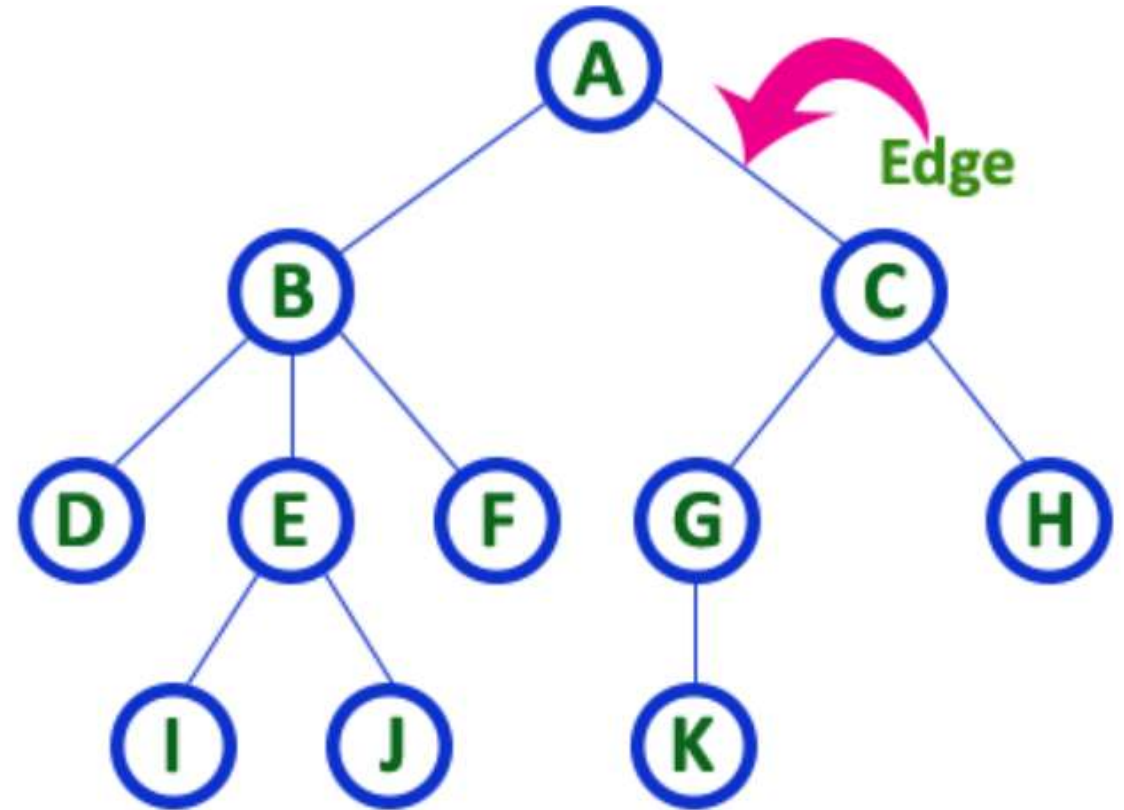
Daraxt ma'lumotlar strukturasi birinchi tugun **ildiz tugun** deb ataladi. Har bir daraxtda ildiz tugunlari bo'lishi kerak. Har qanday daraxtda faqat bitta ildiz tugun bo'lishi kerak. Daraxtda hech qachon bir nechta ildiz tugunlari bo'lmaydi.





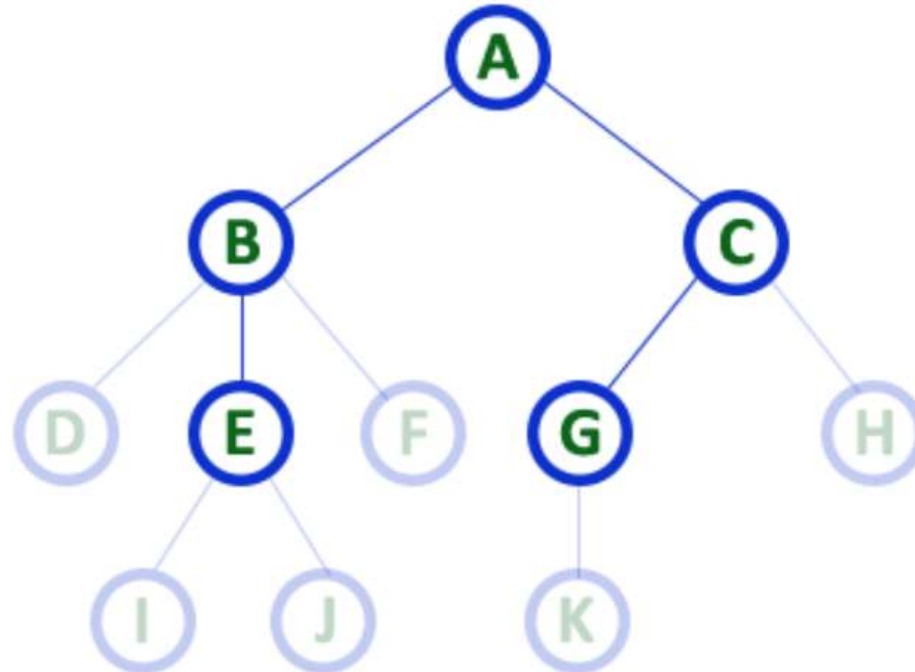
# Edge-qirra

Daraxt ma'lumotlar tuzilmasida har qanday ikkita tugun o'rtasidagi bog'lanish **qirra (edge)** deb ataladi. "N" tugunlari bo'lgan daraxt maksimal "N-1" qirralariga ega bo'ladi.



# Parent-Ota-ona

Daraxt ma'lumotlar tuzilmasida har qanday tugunning o'tmishdoshi bo'lgan tugun ota-ona tugun deb ataladi. Oddiy qilib aytganda, undan boshqa har qanday tugunga shoxiga ega bo'lgan tugun ota-ona tugun deb ataladi. Ota-ona tugunni "Bola tugunlari/bolalari bo'lgan tugun" deb ham aniqlash mumkin.

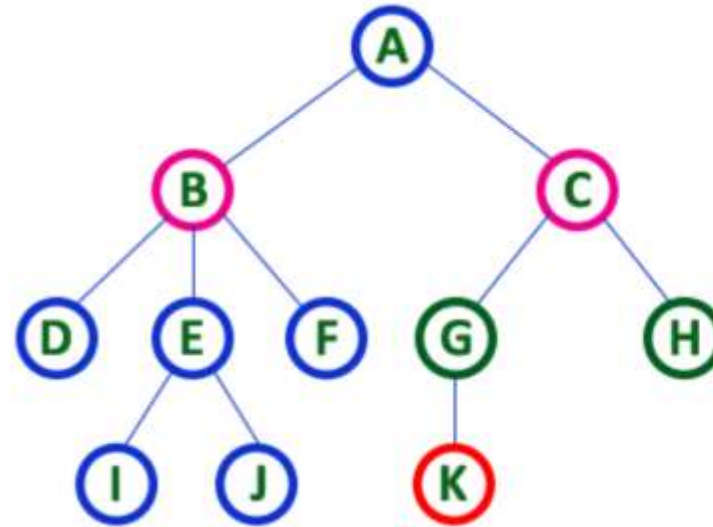


Here A, B, C, E & G are **Parent** nodes

- In any tree the node which has child / children is called '**Parent**'
- A node which is predecessor of any other node is called '**Parent**'

## Bola-Child

Daraxt ma'lumotlari tuzilmasida har qanday tugunning avlodi bo'lgan tugun "bola tugun" deb ataladi. Oddiy qilib aytganda, o'zining ajdod tugunlari bilan bog'liq bo'lgan tugunlar avlod (bola /qiz)tugun deb ataladi. Daraxtda har qanday ota-ona tugunida istalgan sonli avlod tugunlar bo'lishi mumkin. Daraxtda ildizdan tashqari barcha tugunlar bola tugunlari hisoblanadi.



Here **B & C** are **Children of A**

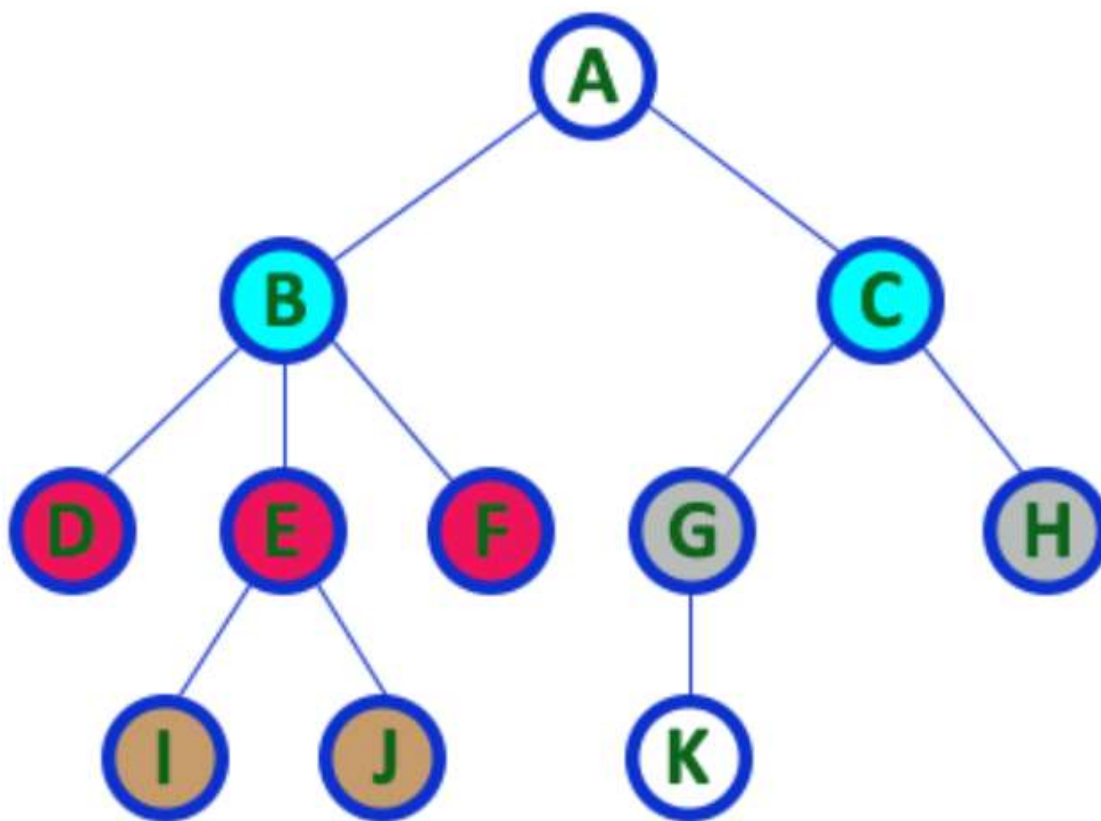
Here **G & H** are **Children of C**

Here **K** is **Child of G**

- descendant of any node is called as **CHILD Node**

## Birodarlar- **Birodarlar - Siblings** tugunlari

Birodar tugunlar: bir xil ota-onaga ega bo'lgan tugunlar birodarlar deb nomlanadi.



Here **B & C** are **Siblings**

Here **D E & F** are **Siblings**

Here **G & H** are **Siblings**

Here **I & J** are **Siblings**

- In any tree the nodes which has same Parent are called '**Siblings**'
- The children of a Parent are called '**Siblings**'



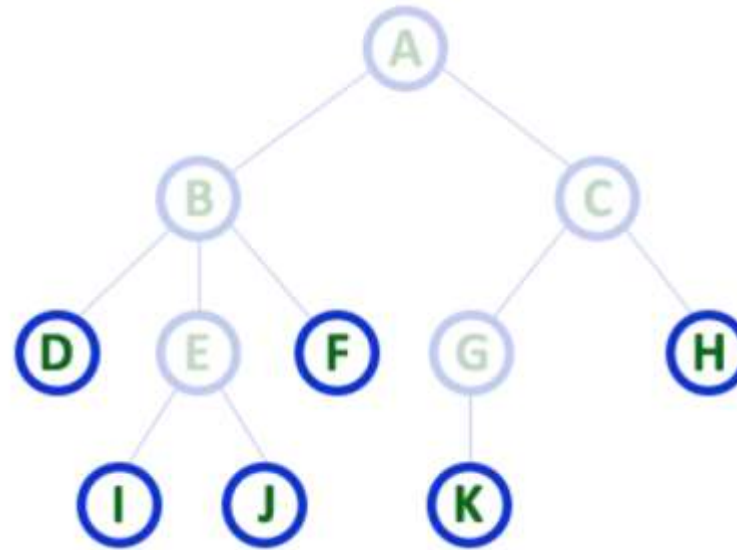
# Barg – Leaf tugunlar

Daraxt ma'lumotlar tuzilmasida hech qanday tugunlarga ega bo'lmagan tugun LEAF tugun deb ataladi.

Oddiy qilib aytganda, barg - bu tugunlar bo'lmagan tugun.

Daraxt ma'lumotlar strukturasi barg tugunlari tashqi tugunlar deb ham ataladi. Tashqi tugun ham bolalari bo'lmagan tugundir.

Daraxtda barg tuguniga "**terminal**" tugun ham deyiladi.



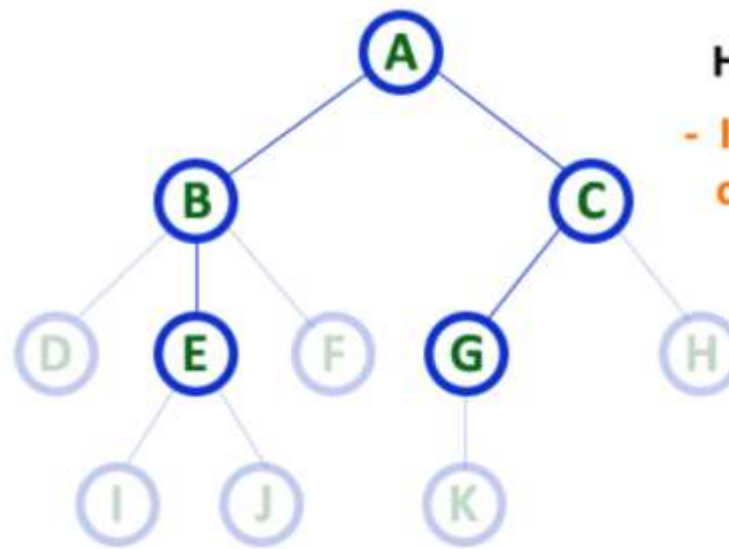
Here D, I, J, F, K & H are **Leaf** nodes

- In any tree the node which does not have children is called '**Leaf**'
- A node without successors is called a '**leaf**' node

# Ichki tugunlar (Internal Nodes)

Daraxt ma'lumotlar strukturasi kamida bitta bolaga ega bo'lgan tugun ichki tugun deb ataladi. Oddiy qilib aytganda, ichki tugun - bu kamida bitta bolaga ega tugun.

Daraxt ma'lumotlar strukturasi barg tugunlaridan boshqa tugunlar ichki tugunlar deb ataladi. Agar daraxtda bir nechta tugun bo'lsa, ildiz tugun ham ichki tugun hisoblanadi. Ichki tugunlar "terminal bo'lmagan" tugunlar deb ham ataladi.



Here A, B, C, E & G are **Internal** nodes

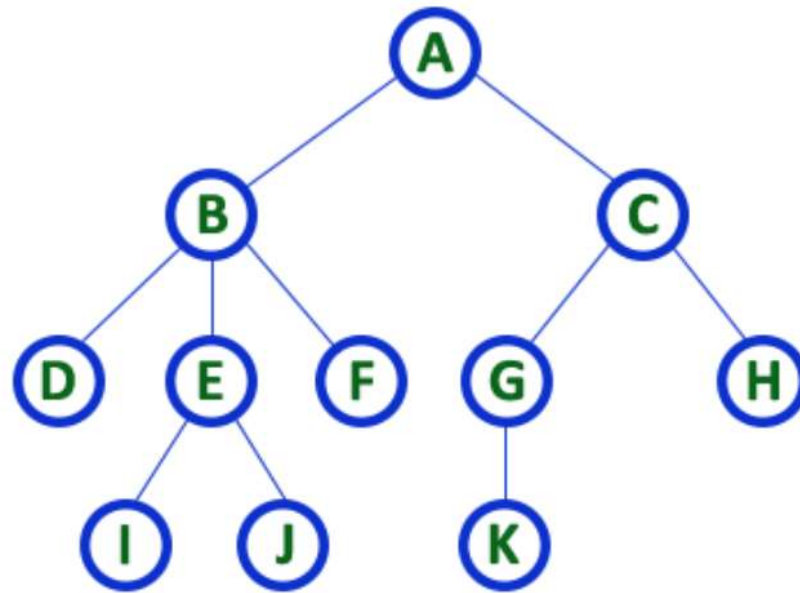
- In any tree the node which has at least one child is called '**Internal**' node

- Every non-leaf node is called as '**Internal**' node



# Daraja - Degree

Daraxt ma'lumotlar strukturasi tugunning umumiy bolalar soni ushbu tugun darajasi deb ataladi. Oddiy qilib aytganda, tugun darajasi uning bolalarining umumiy sonidir. Daraxtdagi barcha tugunlar orasidagi tugunning eng yuqori darajasi "daraxt darajasi" deb ataladi.



Here **Degree of B is 3**

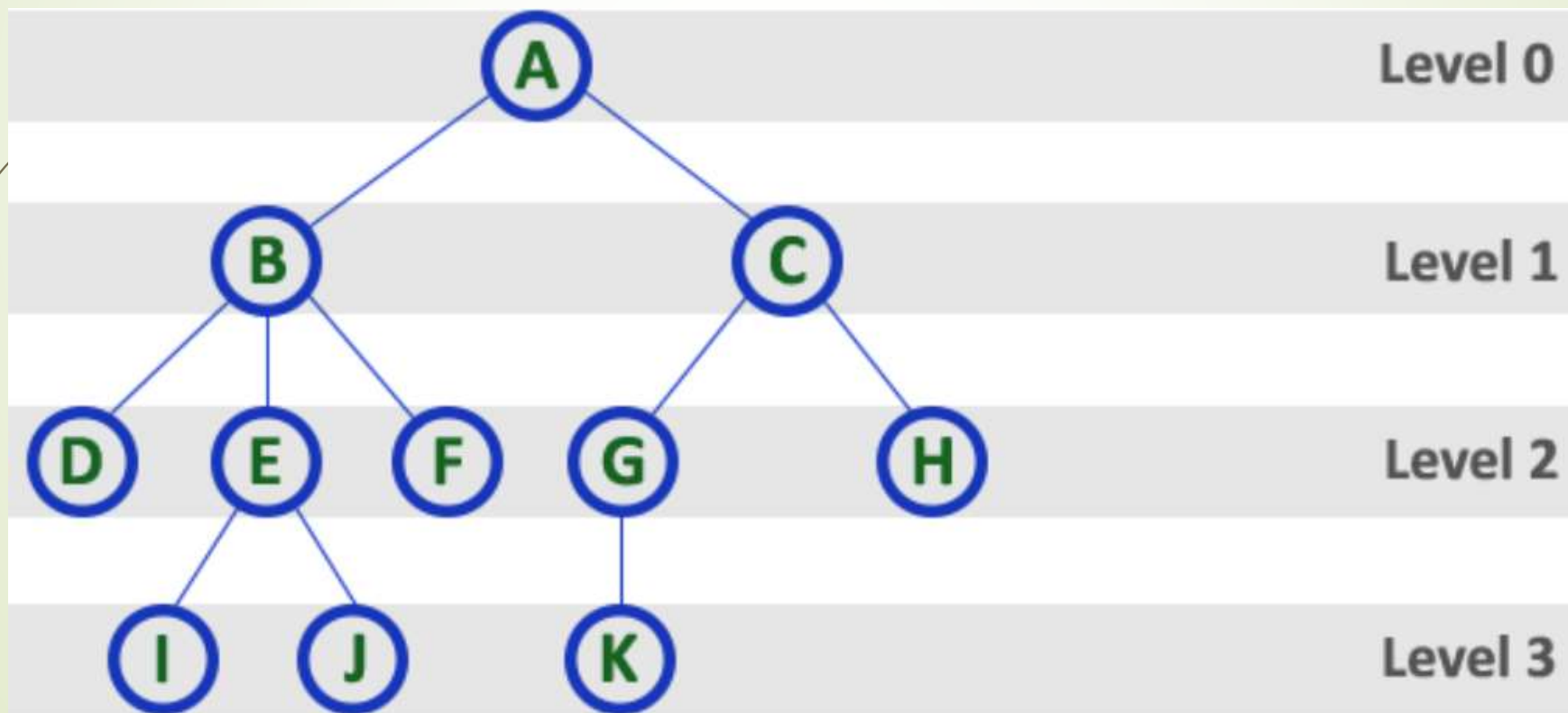
Here **Degree of A is 2**

Here **Degree of F is 0**

- In any tree, '**Degree**' of a node is total number of children it has.

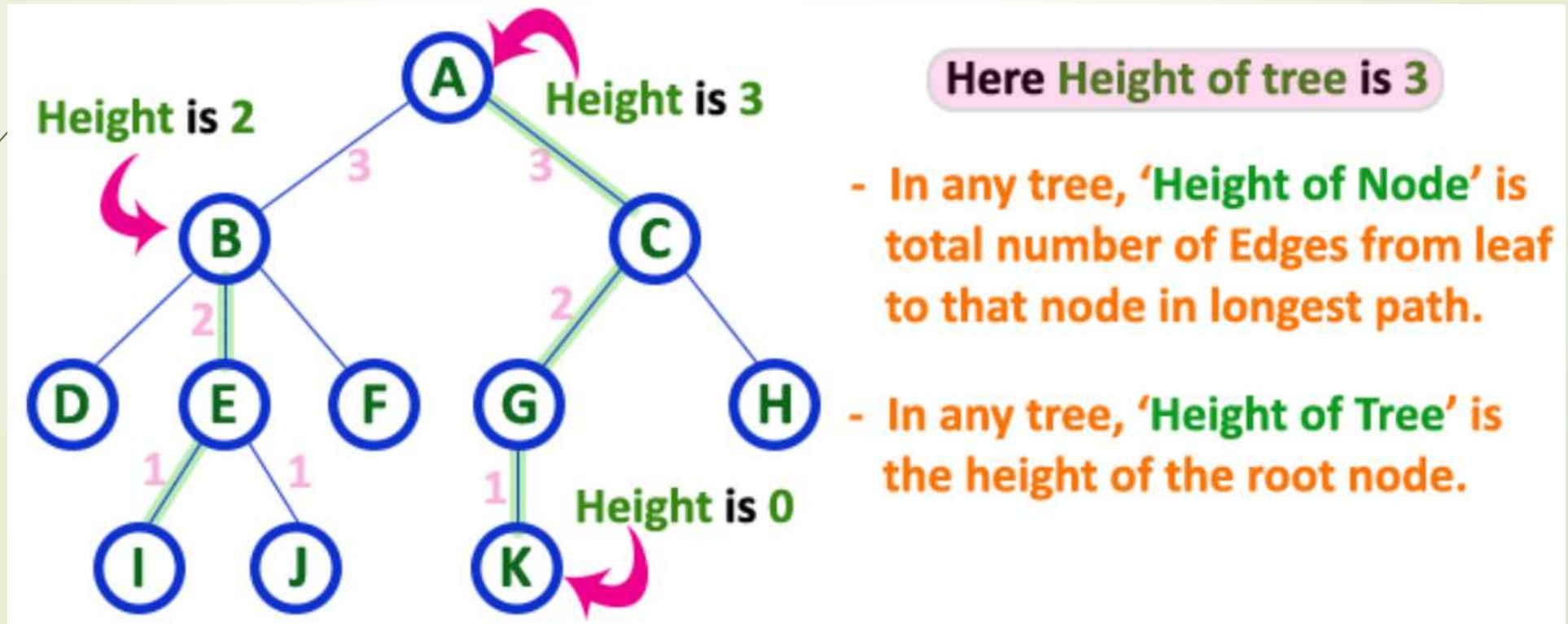
## Sath-Level

Daraxt ma'lumotlari tuzilmasida ildiz tugunlari 0 sathda, ildiz tugunlarining asosiy tugunlari sathda, 1-sathdagi tugunlar esa 2-sathda bo'ladi va hokazo... Oddiy qilib aytganda, daraxtda yuqoridan pastgacha bo'lgan har bir qadam sath deb ataladi va daraja hisoblagichi "0" dan boshlanadi va har bir darajada (qadam) bittaga ortadi.



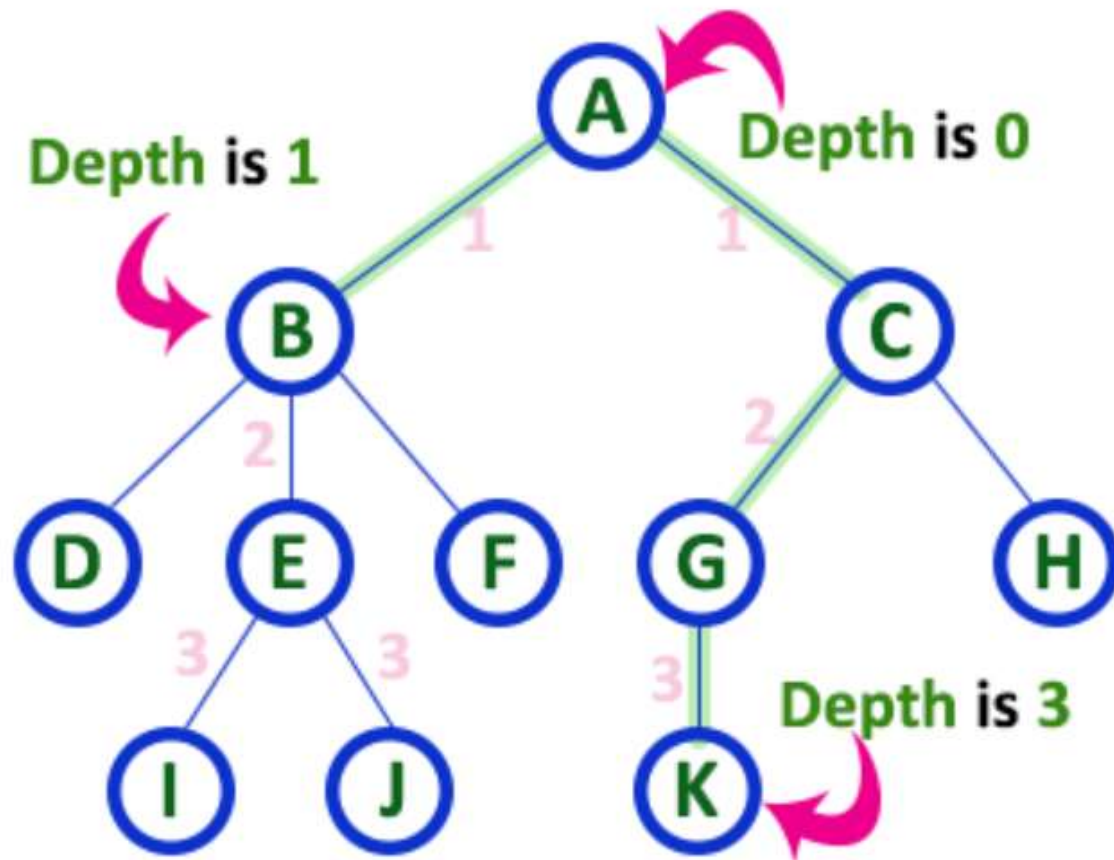
## Balandlik - Height

Daraxt ma'lumotlar tuzilmasida eng uzun yo'l bo'ylab barg tugunidan ma'lum bir tugungacha bo'lgan umumiy qirralarning soni ushbu tugunning balandligi deyiladi. Daraxtda ildiz tugunining balandligi daraxtning balandligi hisoblanadi. Daraxtda barcha barg tugunlarining balandligi "0" dir.



## Chuqurlik - Depth

Daraxt ma'lumotlari tuzilmasida ildiz tugunidan ma'lum bir tugungacha bo'lgan umumiy qirralarning soni ushbu tugunning **chuqurligi** deb ataladi. Oddiy qilib aytganda, daraxtdagi har qanday barg tugunining eng katta chuqurligi shu daraxtning chuqurligi deb ataladi. Daraxtda ildiz tugunining chuqurligi "0" ga teng.



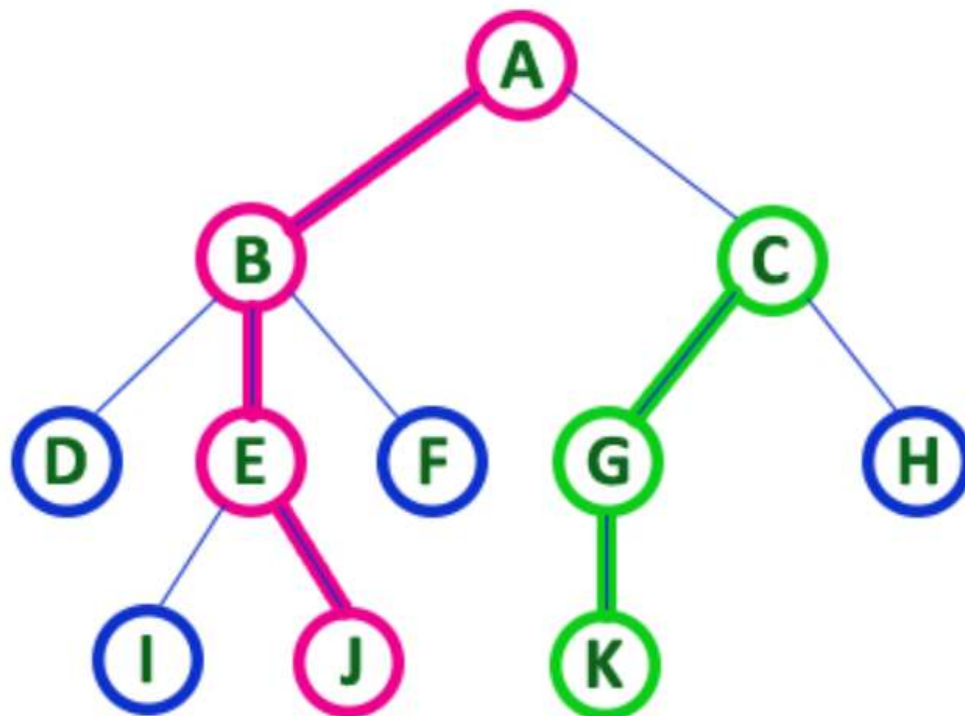
Here Depth of tree is 3

- In any tree, 'Depth of Node' is total number of Edges from root to that node.
- In any tree, 'Depth of Tree' is total number of edges from root to leaf in the longest path.



# Yo'l - Path

Daraxt ma'lumotlar tuzilmasida tugunlar va qirralarning bir tugundan boshqa tugungacha bo'lgan ketma-ketligi bu ikki tugun orasidagi **yo'l** deb ataladi. Yo'l uzunligi - bu yo'ldagi tugunlarning umumiy soni. Quyidagi misolda A - B - E - J yo'lining uzunligi 4 ga teng



- In any tree, '**Path**' is a sequence of nodes and edges between two nodes.

Here, '**Path**' between A & J is

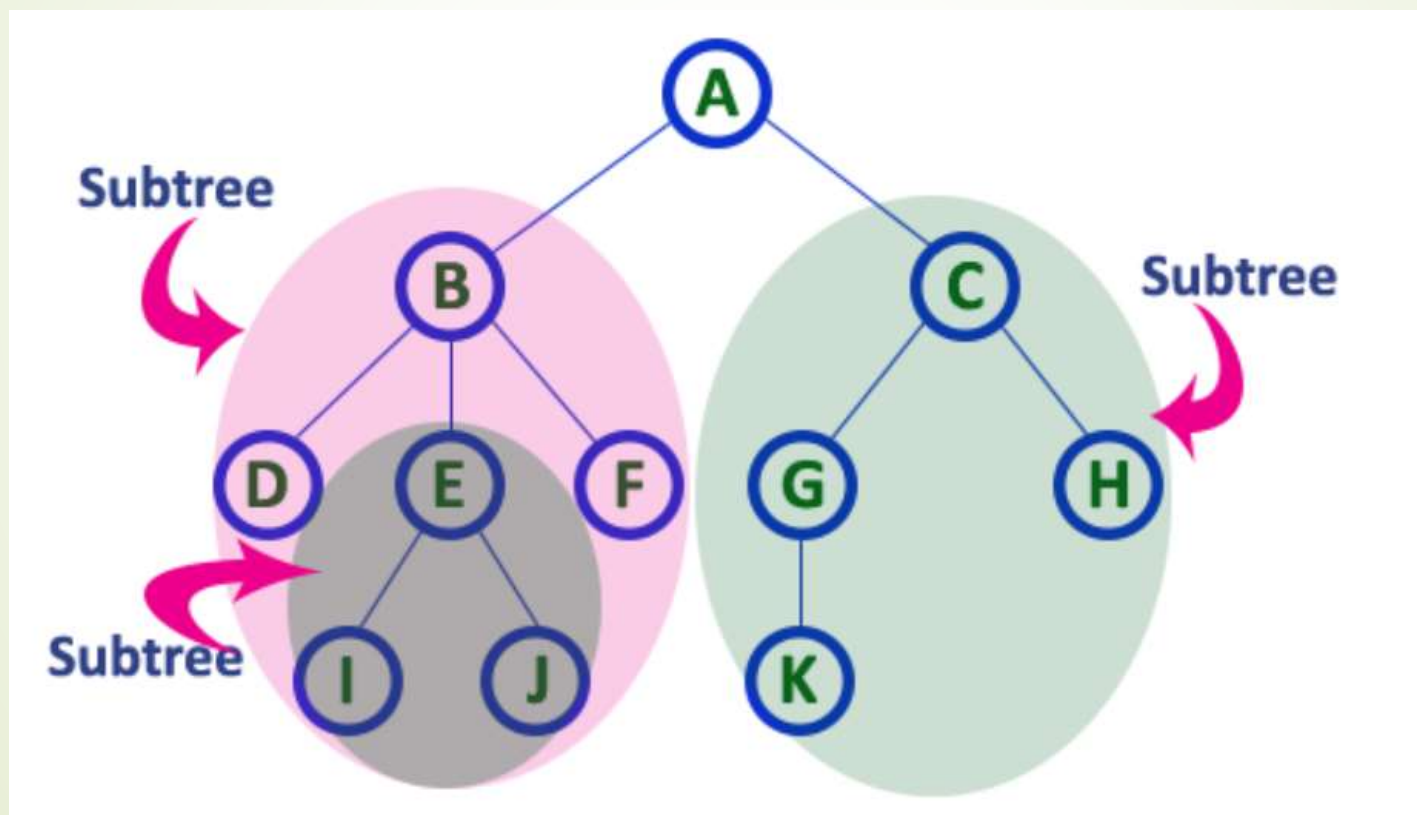
**A - B - E - J**

Here, '**Path**' between C & K is

**C - G - K**

# Daraxt osti - Sub Tree

**Daraxt osti** - bu alohida daraxt sifatida namoyish etilishi mumkin bo'lgan daraxtga o'xshash ma'lumotlar strukturasi bir qismidir. T daraxtining har qanday tuguni va uning barcha nasl tugunlari bilan birga T daraxtining pastki daraxti hisoblanadi.



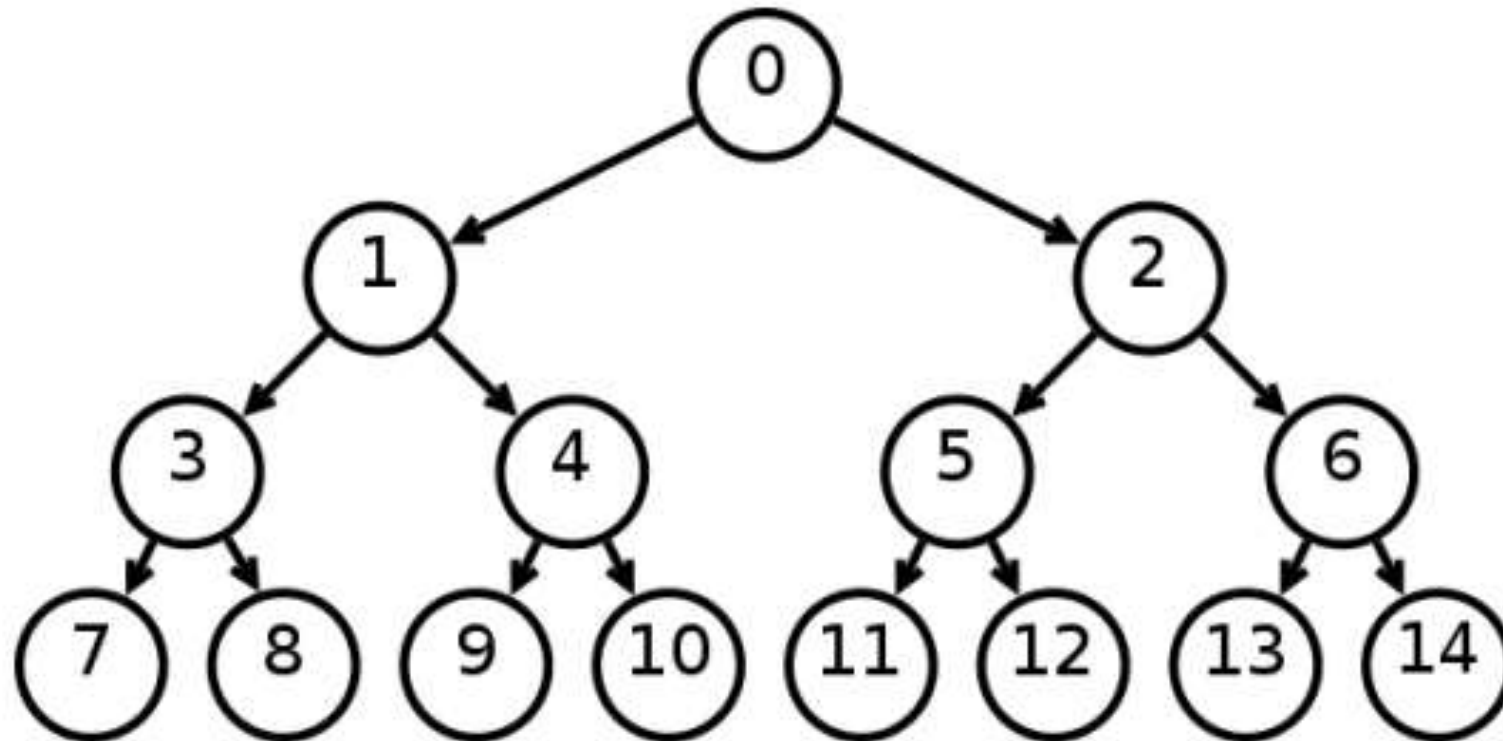


# Daraxt turlari.

1. Ikkilik daraxtlar.
2. Ikkilik qidiruv daraxtlari.
3. AVL daraxtlari.
4. B-daraxtlar.

## Binar (ikkilik) daraxtlar

**Ikkilik daraxt** - bu har bir tugunda ko'pi bilan ikkita avlod (bola) bo'lgan ma'lumotlarning iyerarxik tuzilishi. Odatda, birinchisi ajdod tuguni, avlodlar esa chap va o'ng merosxo'rlar deb nomlanadi.



# Big O

## Array

## BS Tree

Qidirish

$O(\log_2 n)$

$O(\log_2 n)$

Element qo'shish

$O(n)$

$O(\log_2 n)$

Element o'chrisih

$O(n)$

$O(\log_2 n)$

Element o'chrisih

$O(n)$

$O(\log_2 n)$

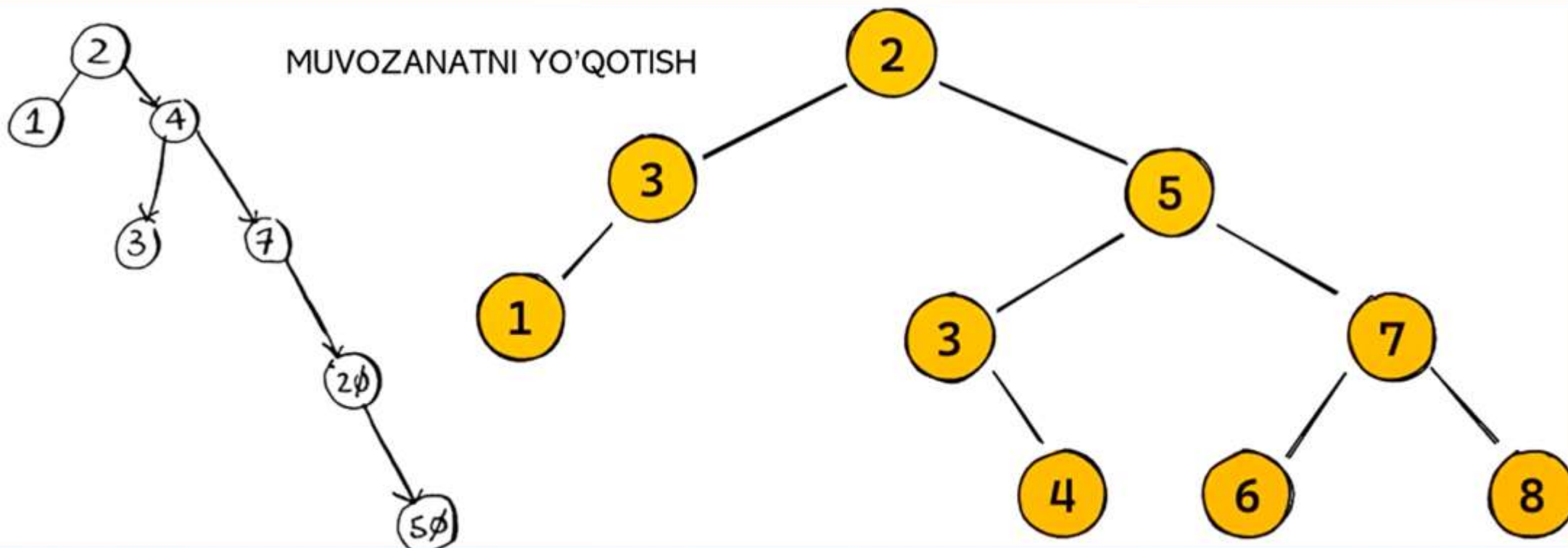
Element qo'shish

$O(n)$

$O(\log_2 n)$

# TREES (SHAJARA) MA'LUMOTLAR TUZILMASI

## Binary Search Tree kamchiligi





# TREES (SHAJARA) MA'LUMOTLAR TUZILMASI

- Boshqa (tree) shajara turlari:
  - Red-black tree (o'zini muvozanatga keltiradi)
  - B-tree (ma'lumotlar bazasida ishlatiladi)
  - Heap
  - Splay tree

# Binary Tree da bajarish mumkin bo'lgan operatsiyalar

## ➤ Asosiy operatsiyalar

- Daraxt tarkibiga ma'lumot kiritish
- Daraxt tarkibidan ma'lumot o'cherish
- Element qidirish
- Daraxt da sayohat

## Yordamchi operatsiyalar

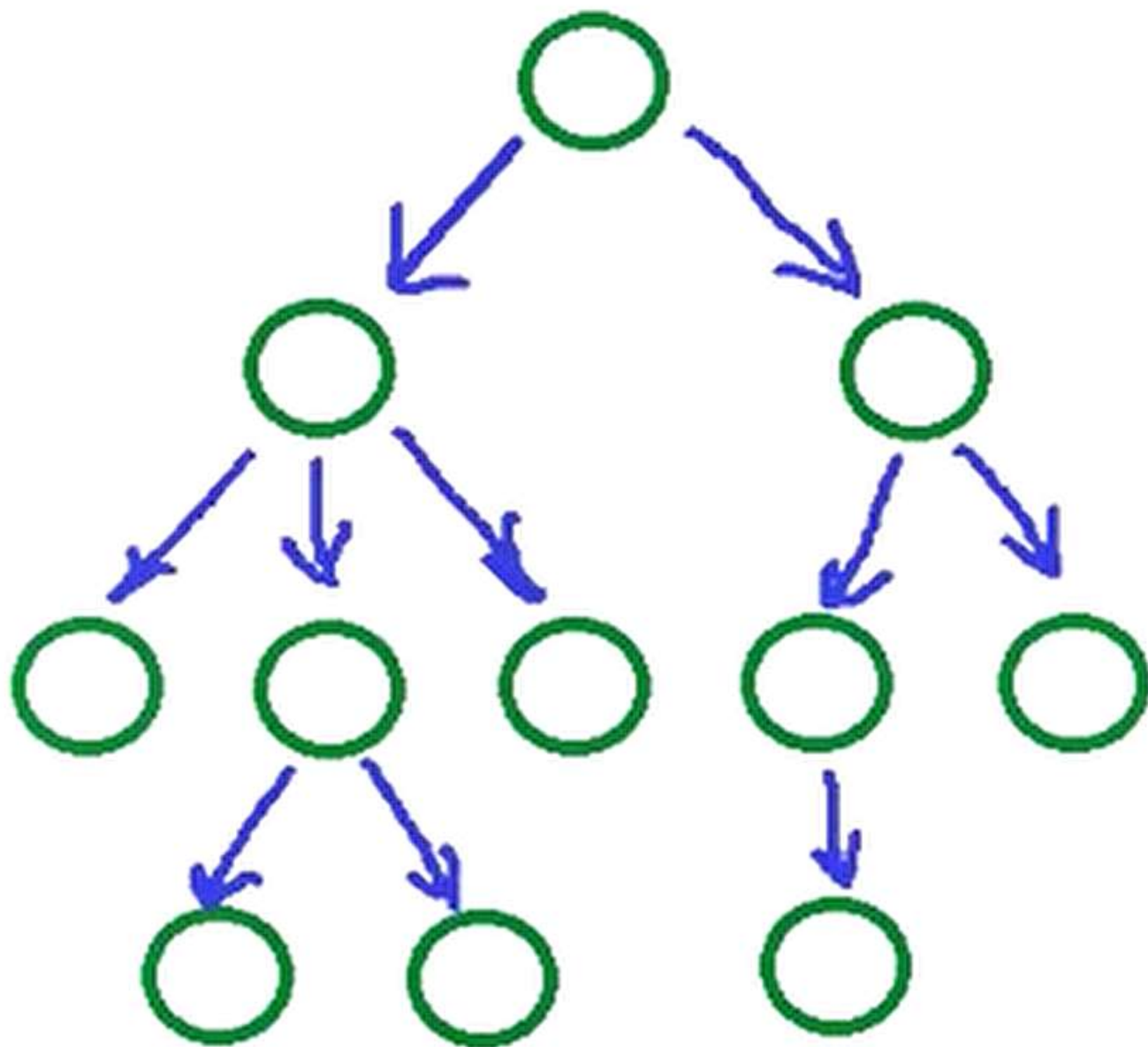
Daraxt o'lchamini topish

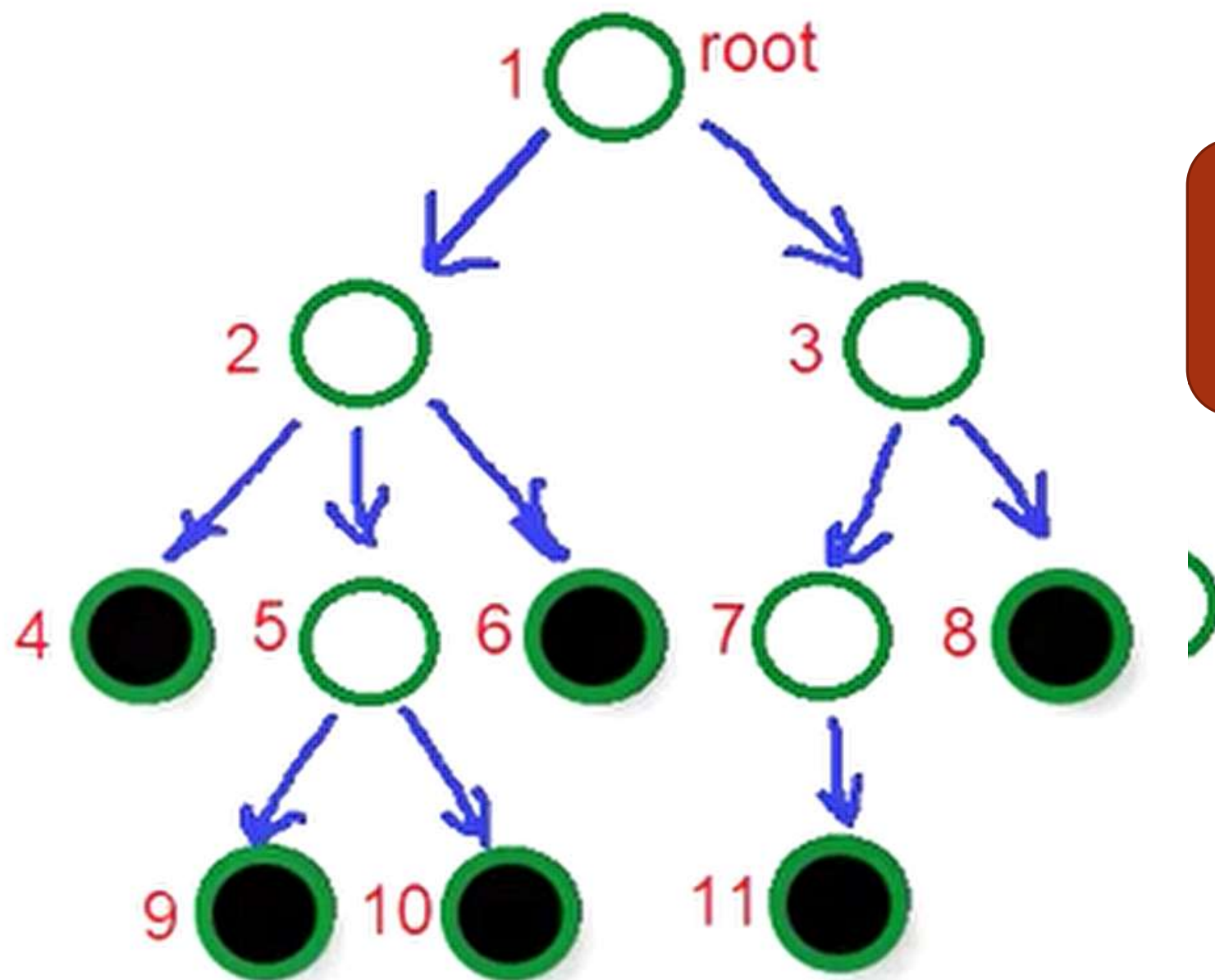
Daraxt balandligini topish

Maksimal summaga ega bo'lgan levelni topish

Berilgan juftlik tugunlari uchun eng kam umumiy ajdodni (EKUA) topish va boshqalar.







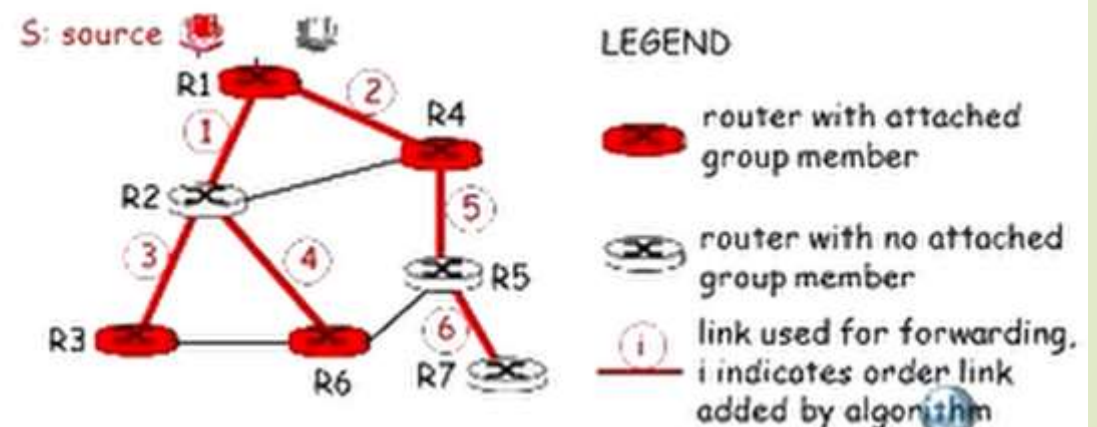
Leaf(Yaproq)

# Daraxt ma'lumot tuzilmasidan qachon foydalaniladi?

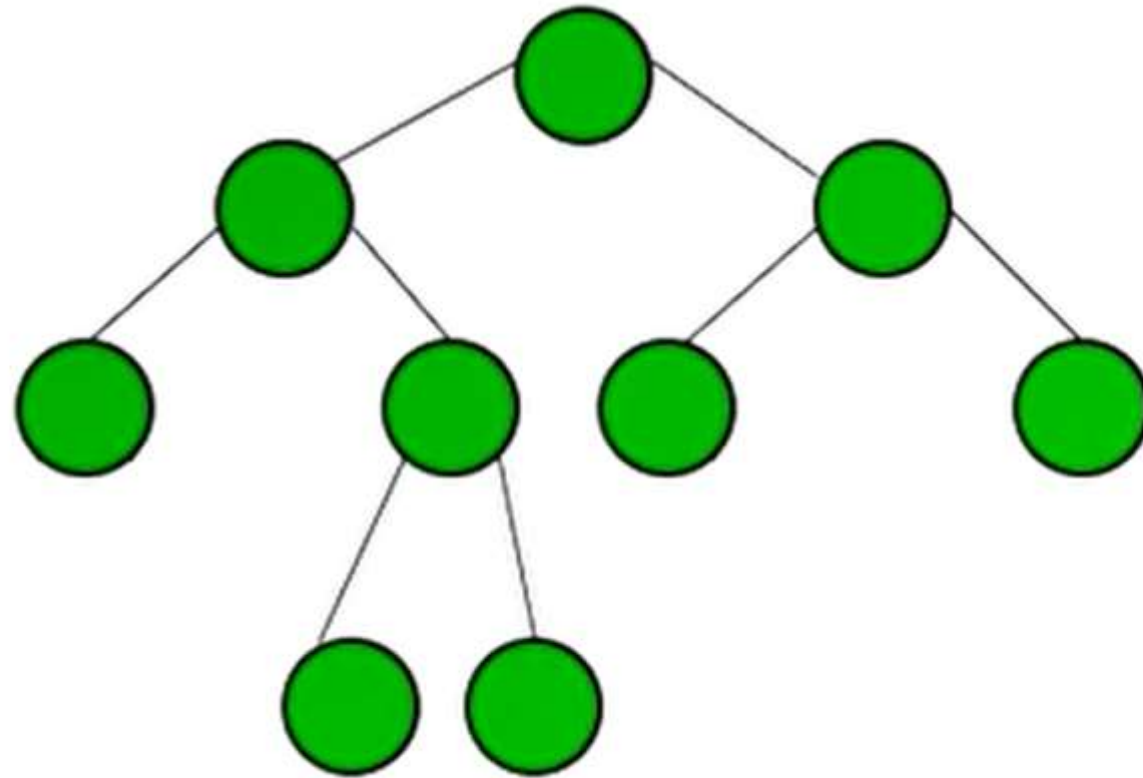
1. Fayl tizimi
2. Tez Izlab topish
3. Lug'at (Dictionary)

## Shortest Path Tree

- mcast forwarding tree: tree of shortest path routes from source to all receivers
  - Dijkstra's algorithm



## Proper binary tree

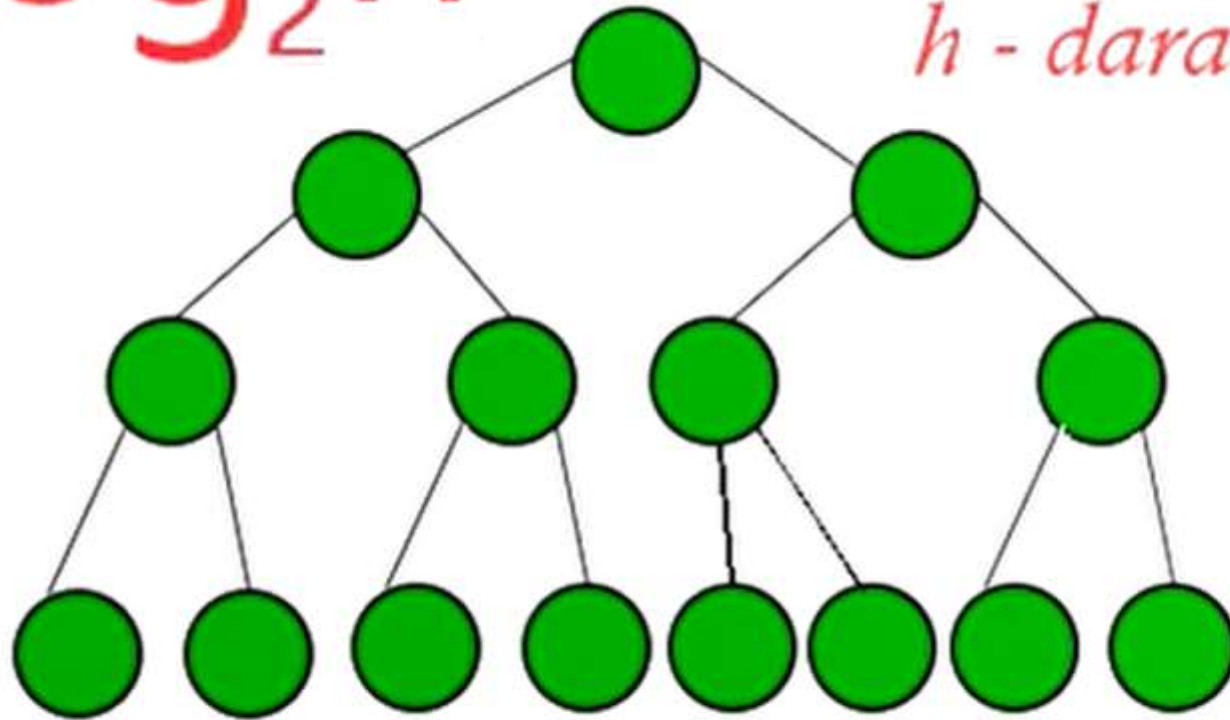


$$i = 2^i$$



$O(\log_2 n)$

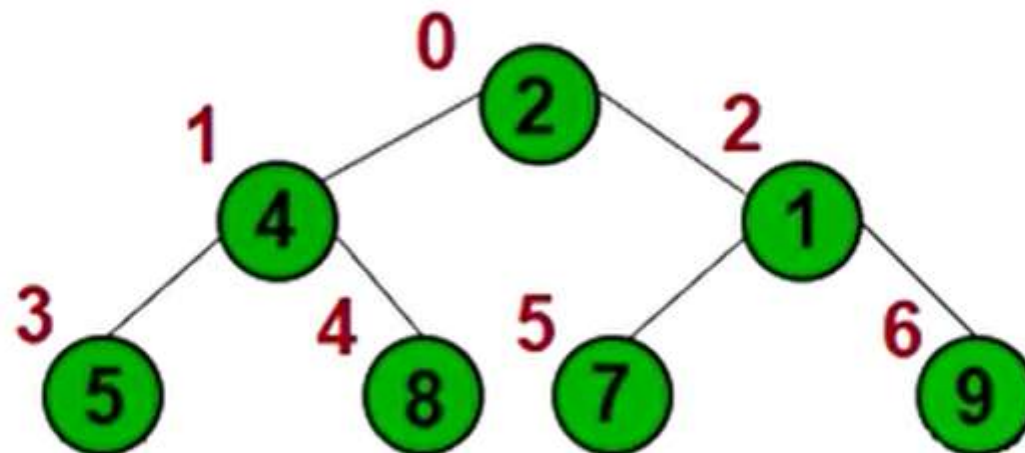
*$h$  - daraxtning balandligi*



Perfect binary tree

$$2^{(h+1)} - 1$$

i - indeksida joylashgan node uchun,  
chap bolasining indeksi =  $2i + 1$   
o'ng bolasining indeksi =  $2i + 2$



2	4	1	5	8	7	9
0	1	2	3	4	5	6



# C# da ikkilik qidiruv daraxti uchun qidiruv, qo'shish va o'chirish operatsiyalari :

Node klassi ikkilik qidiruv daraxtidagi bitta tugunni ifodalaydi. U uchta maydonga ega:

- **kalit:** Bu maydon tugun ko'rsatadigan butun son qiymatini saqlaydi.
- **chap:** Bu maydon chap tugunga havolani saqlaydi.
- **o'ng:** Bu maydon o'ng tugunga havolani saqlaydi.

```
class Node
{
    public int key;
    public Node left, right;

    Ссылка: 1
    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}
```

Binar-daraxt klassi ikkilik qidiruv daraxtini ifodalovchi asosiy sinfdir.

U bitta maydonga ega:

root: Bu maydon daraxtning ildiz tuguniga havolani saqlaydi.

Konstruktor ildiz tugunini null ga initsializatsiya qiladi.

```
class Binar_daraxt
{
    private Node root;

    Ссылка: 1
    public Binar_daraxt()
    {
        root = null;
    }
}
```

# Insert metod

Insert usuli ikkilik qidiruv daraxtiga berilgan kalit qiymatiga ega yangi tugunni kiritadi. Kirish sifatida butun son kalit qiymatini oladi va voidni qaytaradi. Buni ildiz tugunidan daraxtni rekursiv kesib o'tish, kalit qiymatini har bir tugunning kalit qiymati bilan solishtirish va yangi tugunni o'tish yo'lidagi oxirgi tugunning bolasi sifatida kiritish orqali amalga oshiriladi.

```
public void Insert(int key)
{
    root = InsertRec(root, key);
}

Ссылка: 3
private Node InsertRec(Node root, int key)
{
    if (root == null)
    {
        root = new Node(key);
        return root;
    }

    if (key < root.key)
        root.left = InsertRec(root.left, key);
    else if (key > root.key)
        root.right = InsertRec(root.right, key);

    return root;
}
```

## Search metodi

Qidiruv usuli ikkilik qidiruv daraxtida berilgan kalit qiymatiga ega tugunni qidiradi. Kirish sifatida u butun son kalit qiymatini oladi va topilsa tugunga havolani qaytaradi, topilmasa esa nullni qaytaradi. Buni ildiz tugunidan daraxtni rekursiv ravishda kesib o'tish, kalit qiymatini har bir tugunning kalit qiymati bilan solishtirish va agar topilgan bo'lsa, tugunni qaytarish orqali amalga oshiradi.

```
public Node Search(int key)
{
    return SearchRec(root, key);
}

Ссылка: 3
private Node SearchRec(Node root, int key)
{
    if (root == null || root.key == key)
        return root;

    if (root.key > key)
        return SearchRec(root.left, key);

    return SearchRec(root.right, key);
}
```



# Delete metodi

Uchirish (Delete) usuli berilgan kalit qiymatiga ega bo'lgan tugunni ikkilik qidiruv daraxtidan o'chiradi. Kirish sifatida butun son kalit qiymatini oladi va voidni qaytaradi. Buni ildiz tugunidan daraxtni rekursiv ravishda kesib o'tish, kalit qiymatini har bir tugunning kalit qiymati bilan solishtirish va agar topilgan bo'lsa, tugunni o'chirish orqali amalga oshiriladi. Agar tugunning bitta bolasi bo'lsa, bola tugunni almashtiradi. Agar tugunning ikkita bolasi bo'lsa, o'ng pastki daraxtning minimal qiymati tugunni almashtiradi.

```
public void Delete(int key)
{
    root = DeleteRec(root, key);
}

Ссылка: 4
private Node DeleteRec(Node root, int key)
{
    if (root == null)
        return root;

    if (key < root.key)
        root.left = DeleteRec(root.left, key);
    else if (key > root.key)
        root.right = DeleteRec(root.right, key);
    else
    {
        if (root.left == null)
            return root.right;
        else if (root.right == null)
            return root.left;

        root.key = MinValue(root.right);

        root.right = DeleteRec(root.right, root.key);
    }
    return root;
}
```

```
private int MinValue(Node root)
{
    int minv = root.key;

    while (root.left != null)
    {
        minv = root.left.key;
        root = root.left;
    }

    return minv;
}
```

```
public void InOrder()
{
    InOrderRec(root);
}
```

Ссылка: 3

```
private void InOrderRec(Node root)
{
    if (root != null)
    {
        InOrderRec(root.left);
        Console.Write(root.key + " ");
        InOrderRec(root.right);
    }
}
```

```

Binar_daraxt tree = new Binar_daraxt();

tree.Insert(50);
tree.Insert(30);
tree.Insert(20);
tree.Insert(40);
tree.Insert(70);
tree.Insert(60);
tree.Insert(80);
tree.Insert(55);
tree.Insert(10);
tree.Insert(100);

Console.WriteLine("Daraxtning ketma-ket elementlari:");
tree.InOrder();

Console.WriteLine("\n Kalit qiymati 60 ni qidirish:");
Node result = tree.Search(60);
if (result != null)
    Console.WriteLine("Kalit topildi.");
else
    Console.WriteLine("Kalit topilmadi.");


Console.WriteLine("Kalit qiymati 55 ni uchirish:");
tree.Delete(55);
Console.WriteLine("O'zgartirilgan daraxtning tartibli o'tishi:");
tree.InOrder();

```

```

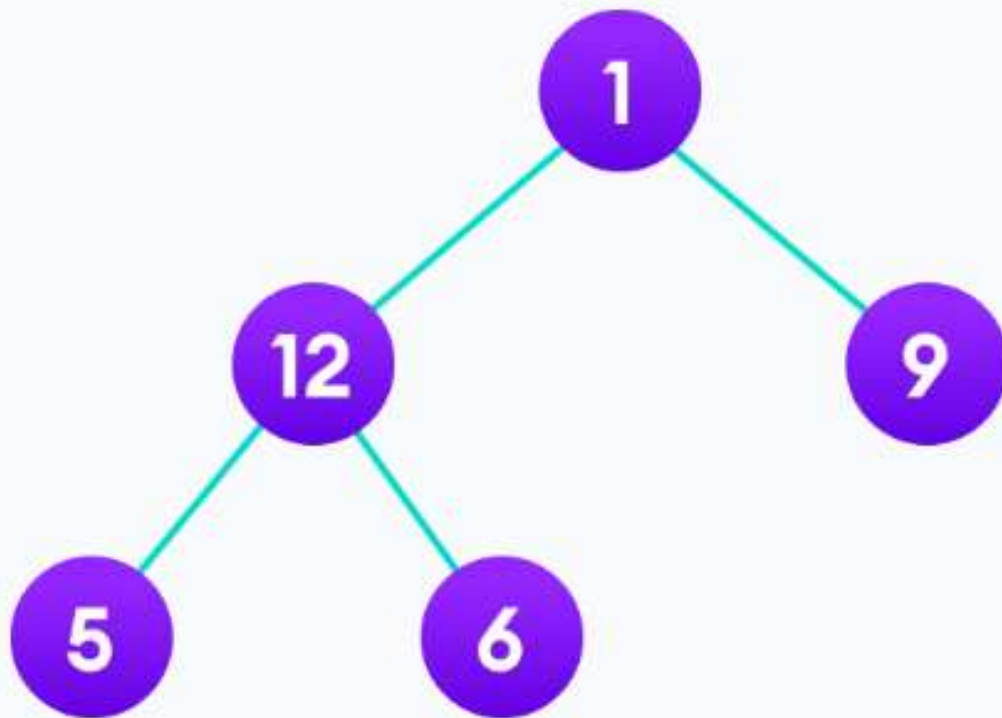
Daraxtning ketma-ket elementlari:
10 20 30 40 50 55 60 70 80 100
Kalit qiymati 60 ni qidirish:
Kalit topildi.
Kalit qiymati 55 ni uchirish:
O'zgartirilgan daraxtning tartibli o'tishi:
10 20 30 40 50 60 70 80 100

```



Daraxtni chetlab o'tish uchun uchta oddiy  
algoritmilar mavjud:  
preorder (oldindan rejalashtiruvchi),  
simmetrik (inorder) va  
teskari (postorder).  
Ikkilik daraxt qidiruvida odatda simmetrik  
chetlab o'tish algoritmi ishlatiladi.





Inorder Tree Traversal

Консоль отладки Microsoft Visual Studio

```
In Order traversal  
5->12->6->1->9->
```

```
public class Node
{
    public int item;
    public Node left, right;

    Ссылка: 5
    public Node(int key)
    {
        item = key;
        left = right = null;
    }
}
```

```
public class Tree
{
    // root of Tree
    Node root;

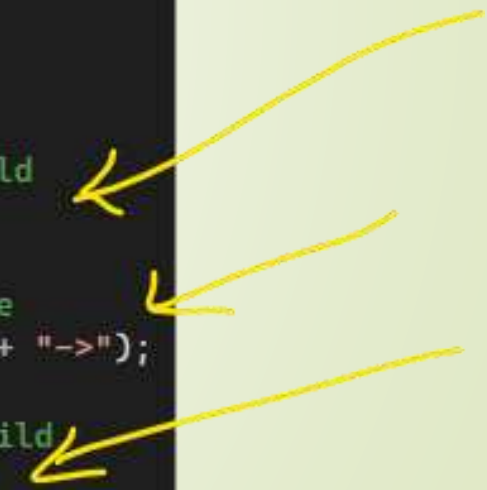
    Ссылка: 1
    public Tree()
    {
        root = null;
    }

    Ссылка: 3
    void InOrder(Node node)
    {
        if (node == null)
            return;

        // traverse the left child
        InOrder(node.left);

        // traverse the root node
        Console.Write(node.item + "->");

        // traverse the right child
        InOrder(node.right);
    }
}
```



```
static void Main(string[] args)
```

```
{
```

```
    // create an object of Tree
```

```
    Tree tree = new Tree();
```

```
    // create nodes of tree
```

```
    tree.root = new Node(1);
```

```
    tree.root.left = new Node(12);
```

```
    tree.root.right = new Node(9);
```

```
    // create child nodes of left child
```

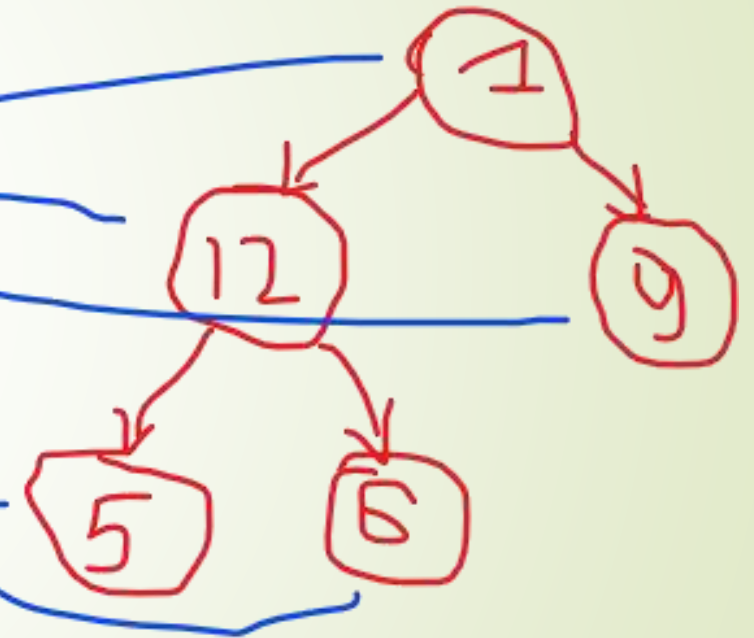
```
    tree.root.left.left = new Node(5);
```

```
    tree.root.left.right = new Node(6);
```

```
    Console.WriteLine("In Order traversal");
```

```
    tree.InOrder(tree.root);
```

```
}
```



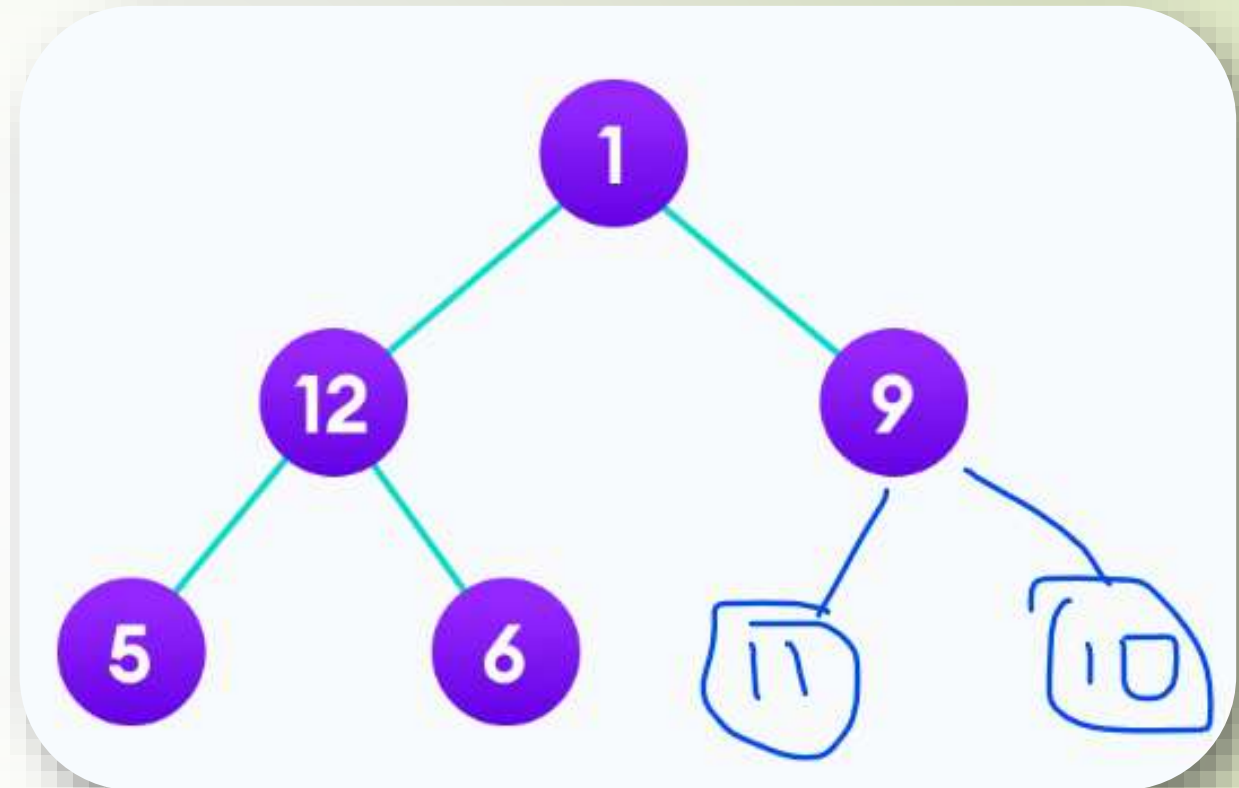


```
// create nodes of tree
tree.root = new Node(1);
tree.root.left = new Node(12);
tree.root.right = new Node(9);

// create child nodes of left child
tree.root.left.left = new Node(5);
tree.root.left.right = new Node(6);

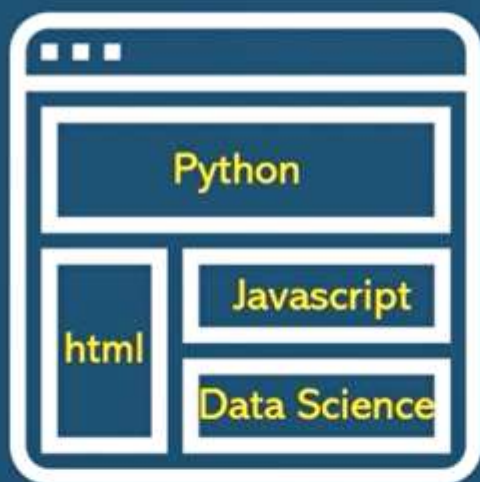
tree.root.right.right = new Node(10);
tree.root.right.left = new Node(11);

Console.WriteLine("In Order traversal");
tree.InOrder(tree.root);
```





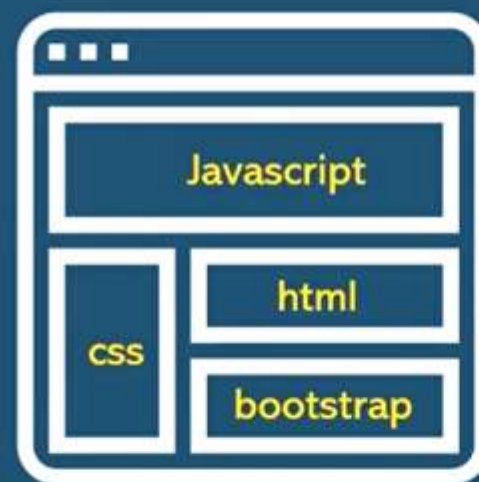
# INVERTED INDEXES MA'LUMOTLAR TUZILMASI



sayyod.com



program.uz



itcomman.uz

# Inverted index

sayyod.com	program.uz	itcomman.uz
python	python	javascript
javascript	django	html
data science	data science	bootstrap
html	sql	css

HASH JADVAL		
kalit	qiymat	
python	sayyod.com,	program.uz
javascript	sayyod.com,	itcomman.uz
data science	sayyod.com,	program.uz
html	sayyod.com,	itcomman.uz
django	program.uz	
sql	program.uz	
bootstrap	itcomman.uz	
css	itcomman.uz	

javascript



# FURYE ALMASHTIRISHLARI ALGORITMI

- Signallarga ishlov berish uchun ishlatiladi
- Audioni alohida chastotalarga ajratish
- Rasmlarga ishlov berish
- Jrayonlarni bashorat qilish
- Turli sensorlar
- DNK tahlili va hokazo

# PARALLEL ALGORITMLAR

- Ulkan ma'lumotlarga ishlov berishni tezlashtirish usuli
- Zamonaviy kompyuterlarda bir nechta yadroli prosessorlar o'rnatilgan
- Katta vazifani bir nechta mayda vazifalarga bo'lib parallel bajarish natijaga tezroq olib keladi
- Lekin parallel algoritmlarni yaratish oson emas:
  - Jarayonlar bir-biriga bog'liq bo'lishi mumkin
  - Parallel bajarilgan vazifalarni jamlash vaqt oladi
  - Jarayonlarni muvozanat qilish qiyinligi



# BLOOM FILTERS

- Google kuniga millionlab saytlarni indeksasiya qiladi
- Har kuni yana millionlab yangi sahifalar (video/rasm/post) yaratiladi
- Bitta sahifani qayta indeksasiya qilmaslik uchun indeksasiya qilingan saytlar ro'yxatini saqlab borish kerak
- Oson yo'li indeksasiya qilingan saytlarni hash jadvali ko'rinishida saqlash
  - Hash jadvalidan o'qish vaqti  $O(1)$  ga teng
  - Muammo: Millilardlab saytlar haqidagi jadvlarni saqlash uchun terbayatlab joy kerak
  - Yechim: Bloom filter

# BLOOM FILTERS

- Bloom filter bu ehtimoliy (probabilistic) ma'lumotlar tuzilmasi
- Hash jadvali o'rniga Bloom filterga sayt manzilini berish va bu sayt indeksasiya qilinganligi ehtimolligini bilish mumkin
  - Misol: 87% ehtimollik bilan indeksasiya qilingan
- Bloom filterlari 100% aniqlik bermaydi, lekin juda kam joy egallaydi

# HyperLogLog

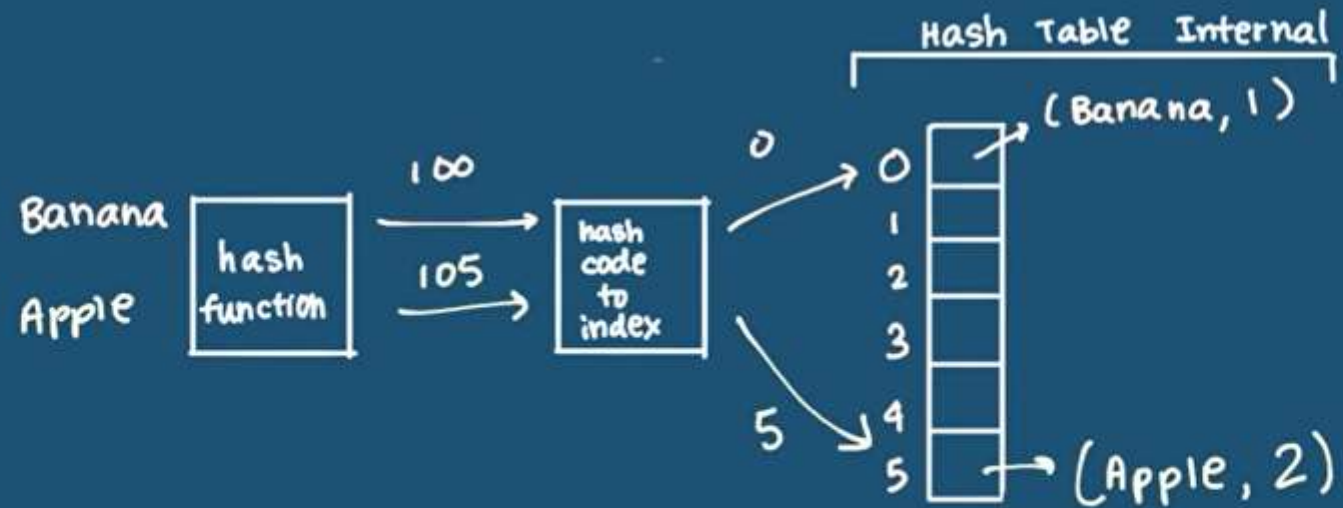
- Amazon e-bozori foydalanuvchi ko'rgan (izlagan) mahsulotlar ro'yxatini saqlab borishi kerak
- Millionlab foydalanuvchilar milliardlab mahsulotlar qidiradi
- Har bir foydalanuvchi haqida bu ma'lumotlarni saqlab borish (logging) xotirada juda katta joy talab qiladi
- Yechim: HyperLogLog
- Bu yechim ham Bloom Filter ka'bi ehtimollar nazariyasiga asoslangan va tahminiy yechim qaytaradi



# Secure Hash Algorithm (SHA)

- Hash jadval

Adding: Banana → 1  
Apple → 2





# Secure Hash Algorithm (SHA)

- SHA berilgan matnni boshqa noyob matnga o'zgartiradi

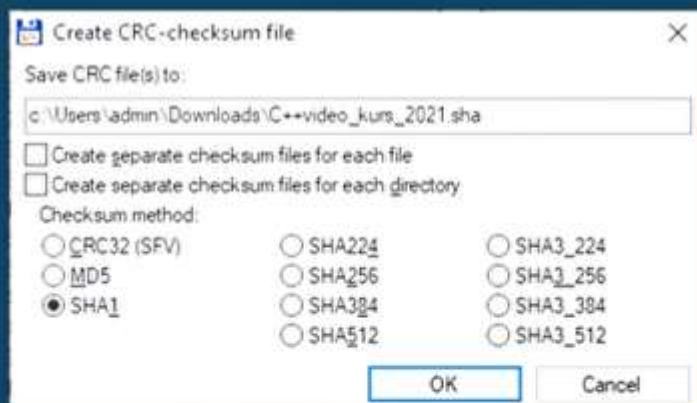
“hello”  $\Rightarrow$  2cf24db...

“algorithm”  $\Rightarrow$  b1eb2ec...

“password”  $\Rightarrow$  5e88489...

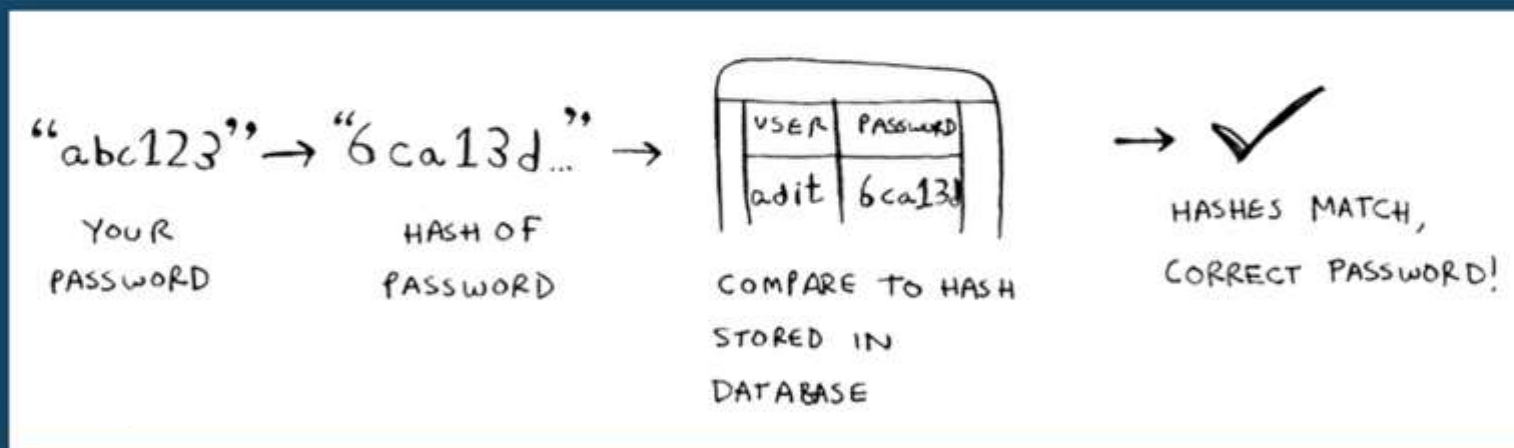
# Secure Hash Algorithm (SHA)

- **Ishlatilishi:** Katta fayllarni solishtirish



# Secure Hash Algorithm (SHA)

- **Ishlatilishi:** Parollarni saqlash/solishtirish



Hash qiymatidan parolni tiklab bo'lmaydi!

# Secure Hash Algorithm (SHA)

- Ishlatilishi: Mualliflik huquqini himoya qilish
- Kuchli hash funksiya o'xshash matnlar uchun ham mutlaqo tasodifiy hash qiymatlari qaytaradi

dog → cd6357

dot → e392da

- Lekin ba'zida bizga o'xshash narsalarni solishtirish talab qilinadi
- Bunday holatda Simhash (similar hash) funksiyasi yordam beradi
  - Youtube – yuklangan videolarni solishtirish uchun
  - Truli platformalar o'g'rilangan kitob/dasturlar yuklanganini tekshirish uchun



# Diffie-Helman Algoritmi

- Tasavvur qiling siz biror kishiga kalit so'z bilan shifrlangan xabar yubordingiz
- Bu odam xabarni ochishi uchun kalit so'zni bilishi kerak
- Kalit so'zni qanday qilib yuborasiz?
- Yechim: Diffie-Helman algoritmi
- Bu algoritmda foydalanuvchida 2 ta kalit bo'ladi: Ochiq va maxfiy
- Ochiq kalitdan istalgan odam foydalanishi va xabarlarni shifrlashi mumkin
- Lekin xabarni qayta ochish uchun maxfiy kalitni bilish shart bo'ladi
- Demak ochiq kalit ommaviy, yopiq kalit esa maxfiylikicha sizda turadi

# Diffie-Helman Algoritmi

## PUBLIC KEY CRYPTOGRAPHY



RSA

# Linear Programming

- Chegaralangan resurslar bilan maksimal natijaga erishish algoritmlari
- Misol:
  - Duradgor rom yasash uchun  $3\text{m}^3$  daraxt va 1L lak ishlatadi
  - Eshik uchun  $5\text{m}^3$  daraxt va 2L lak ishlatadi
  - Har bir romdan 20\$, eshikdan 35\$ foyda oladi.
  - Savol:  $40\text{m}^3$  daraxt va 10L lak bilan maksimum qancha daromad olish mumkin?
- Yuqoridai kabi muammolar Simplex Liner algoritmlari yordamida yechiladi.

<https://www.docs.outtalent.com>

<https://www.leetcode.com>




## Ikkilik qidiruv va binar daraxt qidiruvi farq

Ikkilik qidiruv va binar daraxt qidiruvi mos ravishda tartiblangan ro'yxat va ikkilik qidiruv daraxtidagi elementlarni topish uchun ishlatiladigan ikki xil algoritmdir. Quyida ular orasidagi farqlar keltirilgan:

Ma'lumotlar tuzilishi: Ikkilik qidiruv tartiblangan massivda, ikkilik qidiruv daraxti esa binar qidiruv daraxtida amalga oshiriladi.

Qidiruv metodologiyasi: Ikkilik qidiruvda massiv maqsadli element topilmaguncha yoki massivda yo'q deb topilguncha massiv qayta-qayta ikkiga bo'linadi. Ikkilik daraxtni qidirishda qidiruv daraxtni ildiz tugunidan kesib o'tish, maqsadli qiymatni kamayish tartibida tugun qiymatlari bilan solishtirish va maqsad qiymatdan kichik yoki kattaroq bo'lishiga qarab chapga yoki o'ngga siljitish orqali amalga oshiriladi. tugun qiymati.





Vaqt murakkabligi: Ikkilik qidiruv  $O(\log n)$  vaqt murakkabligiga ega, chunki har bir iteratsiyada qidiruv maydoni ikki barobarga kamayadi. Ikkilik daraxt qidiruvi ham  $O(\log n)$  ning o'rtacha vaqt murakkabligiga ega, lekin agar daraxt muvozanatsiz bo'lsa va bog'langan ro'yxatga o'xshab ko'rinsa, eng yomon holatda u  $O(n)$  ga tushishi mumkin.

Sig'im (hajm) murakkablik: Ikkilik qidiruv  $O(1)$  fazo murakkabligiga ega, chunki u massivning boshlanish va tugatish indekslarini saqlash uchun faqat doimiy hajmdagi joyni talab qiladi. Ikkilik daraxtni qidirish  $O(h)$  fazoviy murakkablikka ega, bu erda  $h$  - daraxt balandligi. Eng yomon holatda, daraxt qiyshayganda, daraxtning balandligi  $n$  bo'lishi mumkin, bu erda  $n$  - daraxtdagi tugunlar soni, natijada  $O(n)$  fazoning murakkabligi.

Umuman olganda, binar qidiruv ham, ikkilik daraxt qidirish ham elementlarni topish uchun ishlatiladigan algoritmlardir, lekin ular ma'lumotlar tuzilishi, qidirish texnikasi, vaqt murakkabligi va sig'im murakkabligi bilan farqlanadi.



