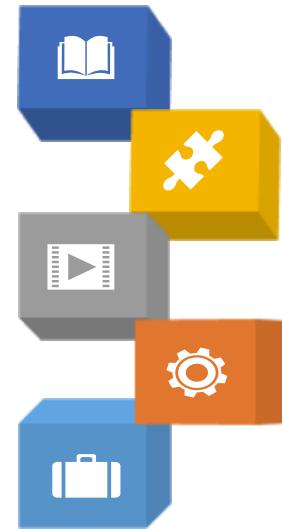




ХӨДӨЛМӨРИЙН ГАВЬЯАНЫ УЛААН ТУГИЙН ОДОНТ
МОНГОЛ УЛСЫН ШИНЖЛЭХ УХААН
ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
МЭДЭЭЛЭЛ, ХОЛБООНЫ ТЕХНОЛОГИЙН СУРГУУЛЬ



Welcome to F.CSM 203

Өгөгдлийн бүтэц ба алгоритм

ЛЕКЦІХ



F.EE23
Г.Ганчимэг

СЭДЭВ. DATA STRUCTURES AND ALGORITHMS

Агуулга



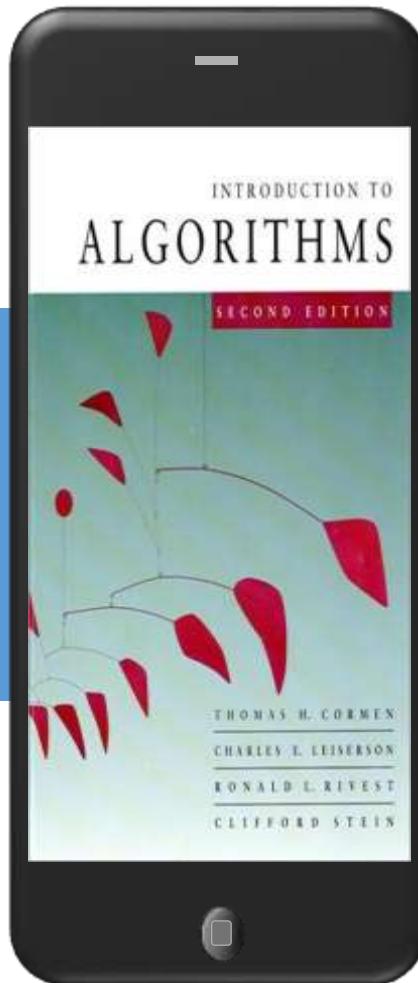
Мод



Хоёртын
хайлтын мод



AVL
Тэнцвэрт мод



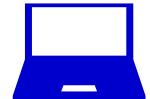
BFS - DFS



Hash table



Жишээ ба
код

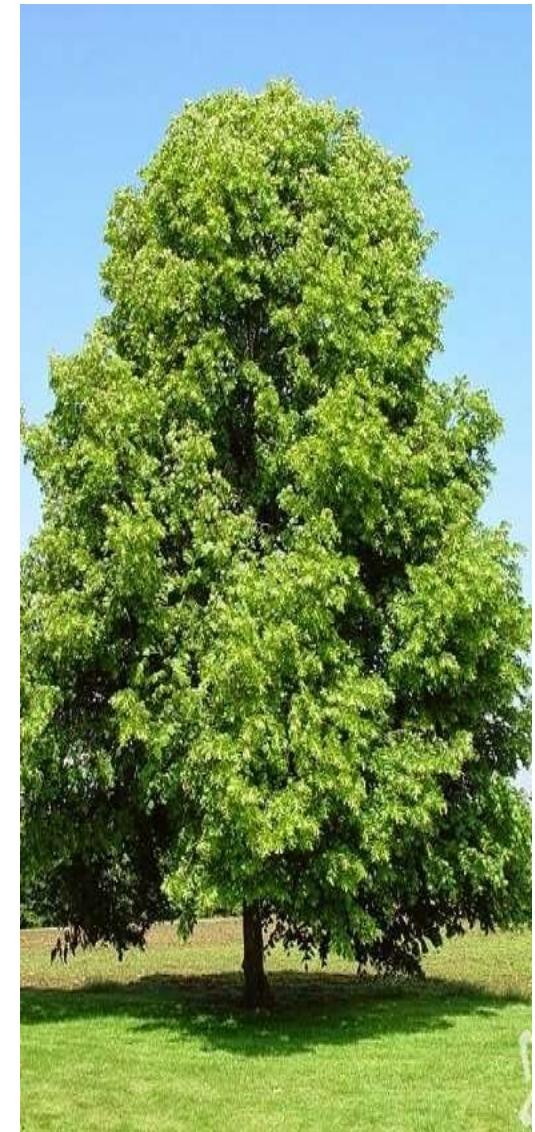


Мод

"Мод хэдэн мянган фут
өндөр ургаж болно, гэхдээ
навч нь үндэс рүүгээ буцна."

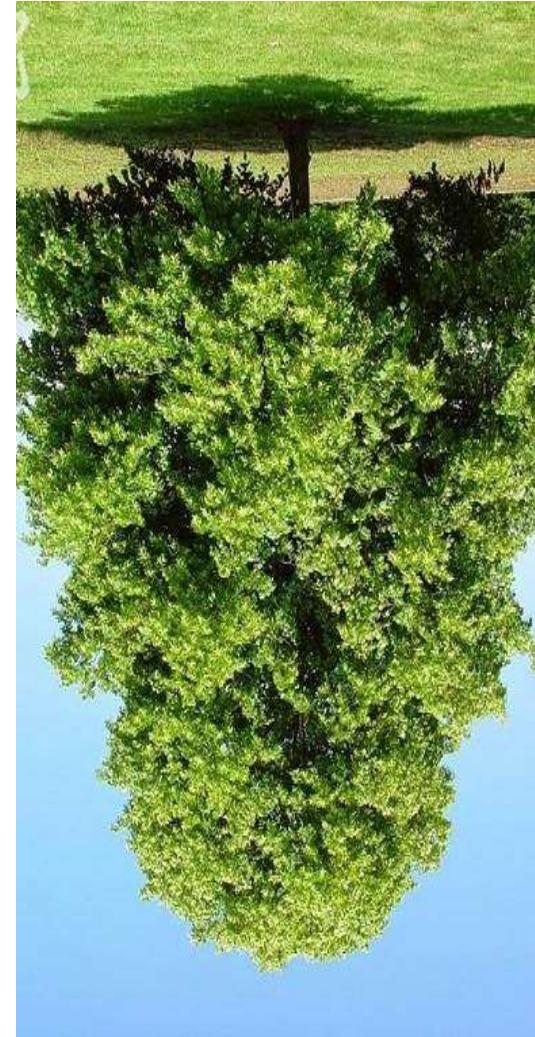
"A tree may grow a thousand
feet tall, but its leaves will
return to its roots."

Chinese Proverb



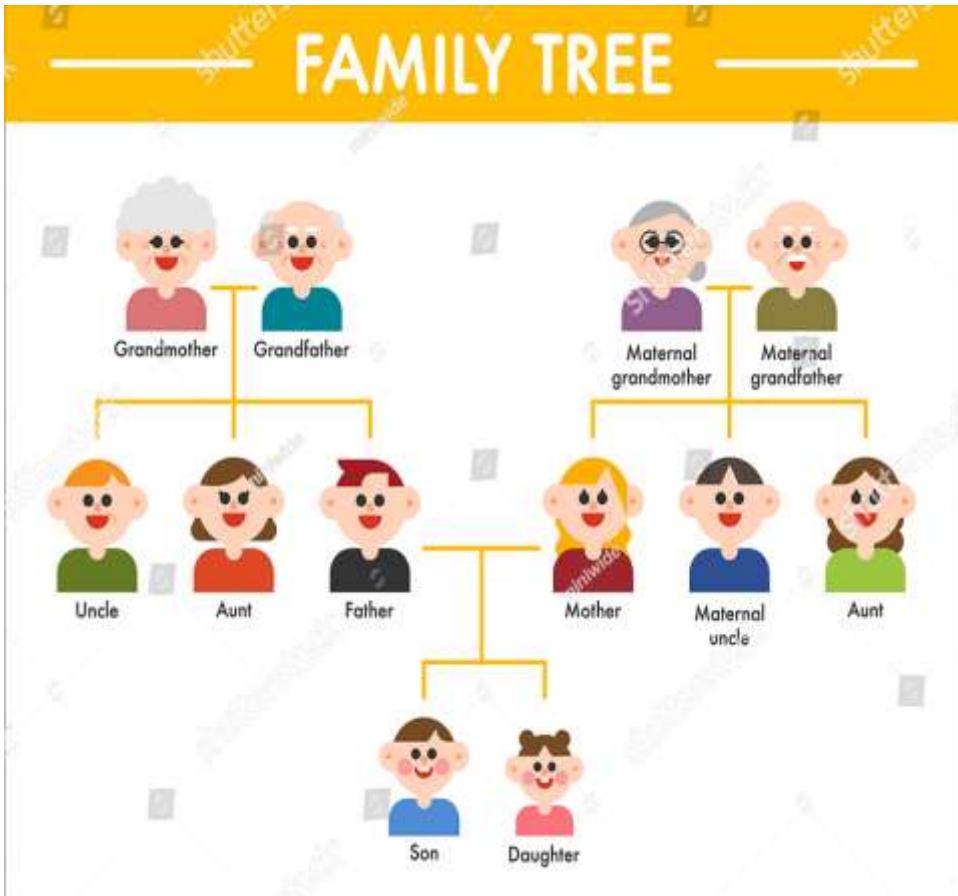
Мод

- Байгаль дээр үндэс, мөчир, навч бүхий ургамлыг мод гэдэг.
- Компьютерын ухаанд модтой бүтцийн хувьд төстэй өгөгдлийн бүтцийг мөн мод гэдэг.
- Гэхдээ компьютерын ухаанд **модыг уруу харуулж зурдаг**.
- Мод гэдэг бүтцийн хувьд **үндэс** бол хамгийн чухал элемент бөгөөд нэг модонд цорын **ганц үндэс** байна. Үндсээс мөчир, мөчрөөс навч гардаг.



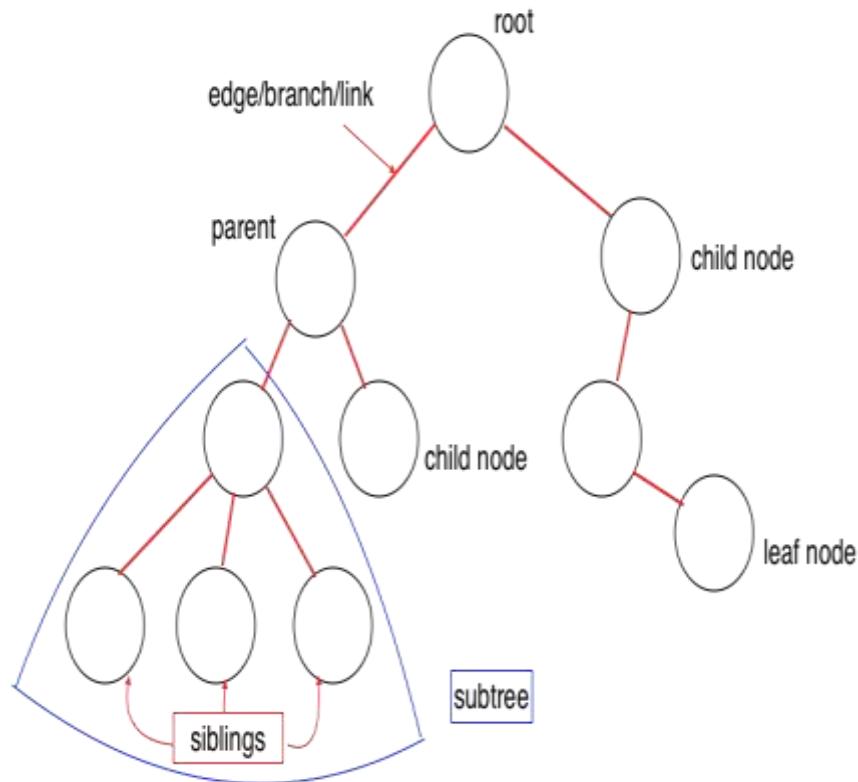
Мод

- Салаалж ургасан модны хэсгийг дэд мод гэнэ.
- Нэг зангилаанаас хэдэн элемент мөчир (навч) салаалсан гэдгээс нь хамааруулж хоёртын, N –тын мод гэж ялгаж болдог.
- Тэд нь дотор хоёрын мод онцгой байр эзэлдэг бөгөөд хоёртын модонд сууринласан олон хэрэглээ байдаг.
- Модны тухай ярихад өвөг эцэг, эцэг, хүүхэд, ахан дүүс гэсэн ухагдахуунуудыг бас хэрэглэдэг.



Мод

- Мод нь ямар нэг нөхцөлийг хангах тодорхой дүрмээр зохион байгуулагдсан зангилаанууд болон тэдгээрийг холбосон холбоосуудыг агуулдаг.
- Зангилаа нь мэдээлэлийг хадгалах ердийн нэг объект ба холбоос нь хоёр зангилааны хоорондын харилцааг тодорхойлдог.
- Холбоосоор холбогдсон модны дараалсан зангилаануудын жагсаалтыг зам гэнэ.

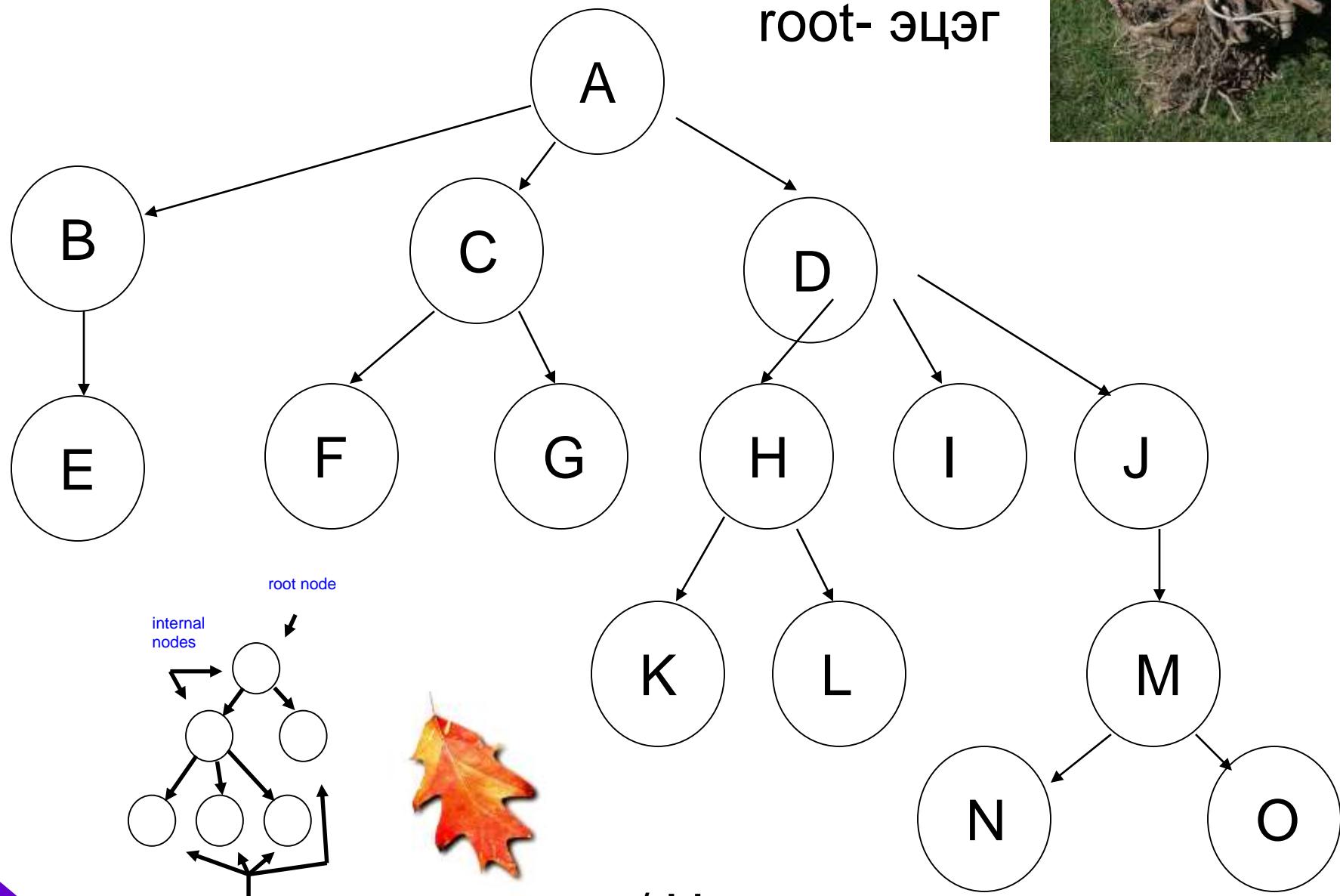


Мод

- Модыг ингэж сурх нь байгаль дээрхтэй адилтгаж ойлгоход хэрэгтэй ч компьютерын ухаанд дараах байдлаар зурдаг.
- Амьдрал дээр мод бүтцээр илэрхийлж болох, илүү тохиромжтой олон тооны өгөгдлийг нэрлэж болно. Жишээ нь хүмүүсийн ургийн бичиг, байгууллагын зохион байгуулалт, байгаль, техникийн нийлмэл системийн бүтэц гэх мэт.
- Өгөгдлийн мод бүтэц дээр боловсруулалт хийнэ гэдэг үнэн чанартаа модны үндсээс навч хүртэл дээшээ, доошоо салаа мөчрөөр дамжин нэвтрэлт хийнэ гэсэн үг юм.



Мод

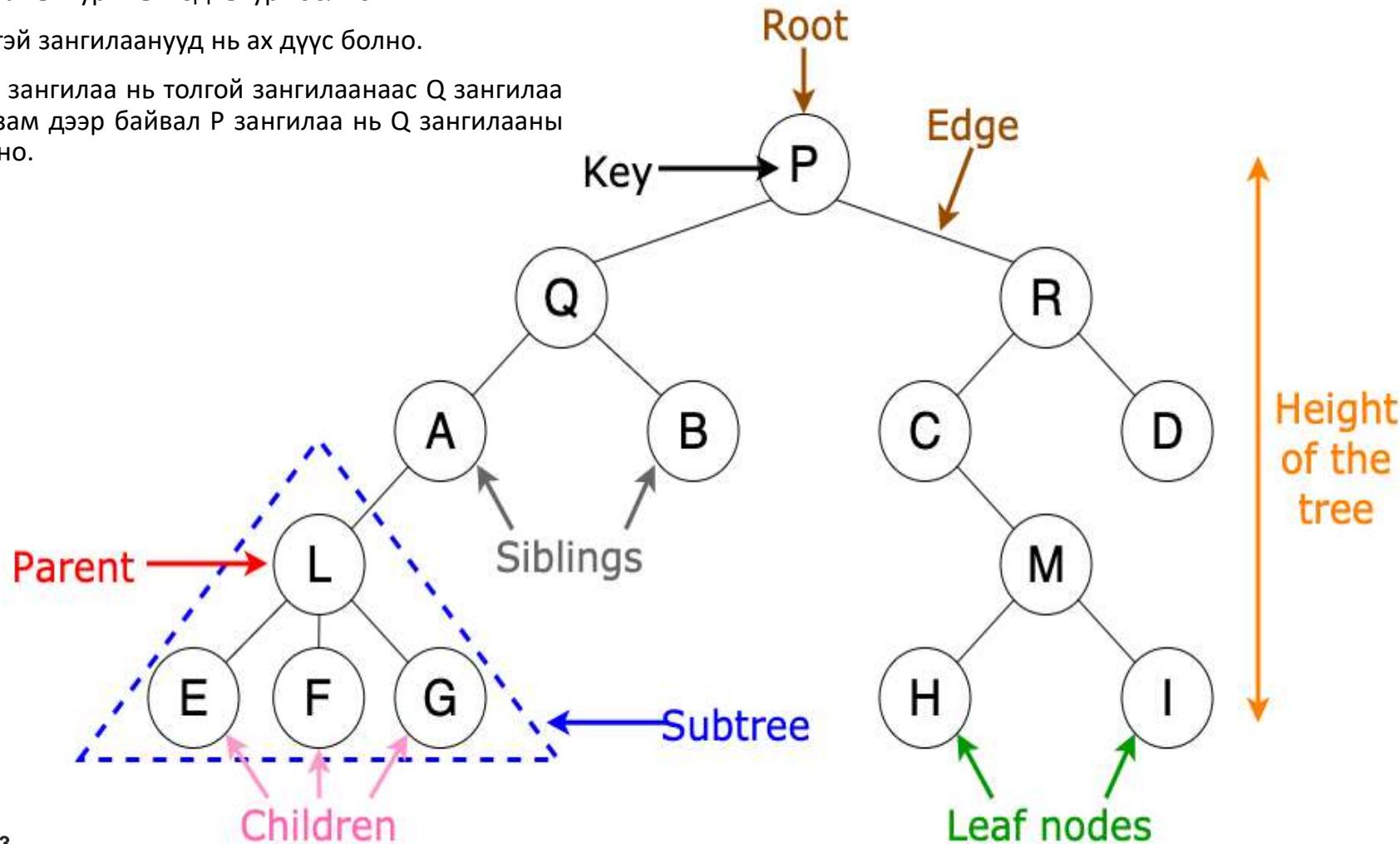


Binary Tree / Хоёртын мод

- Компьютерын ухаанд **хоёртын мод** нь өргөн хэрэглэгддэг өгөгдлийн бүтэц юм. Хоёртын мод нь модны онцгой нэг тохиолдол бөгөөд зангилаа бүр нь хоёроос илүүгүй дэд зангилаатай байдаг. Үндэс (root node) нь модны бүх зангилааны өвөг дээдэс нь болох ба модны аль ч зангилаа толгой зангилаанаас хамааралтай байна. Хэрэв нэг ч зангилаа агуулаагүй бол хоосон мод буюу null мод / зангилаа гэж нэрлэнэ.
- **Хоёртын мод** онд бүх зангилааны түвшин нь ихэвчлэн хоёр байдаг. **n** зангилаатай мод байвал **n-1** нь мөчир эсвэл түвшин болно.
- **Хоёртын мод** нь binarySearchTrees эсвэл binaryHeaps хэрэгжүүлдэг. Энэ нь хайх болон эрэмбэлэх гэсэн давуу талуудыг зэрэг тусгасан.
- **Хоёртын модны** онцгой хэрэглээ нь K-ary tree юм.

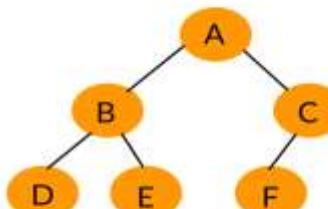
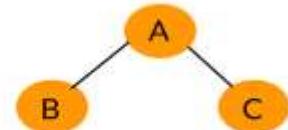
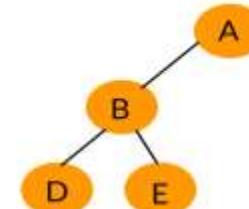
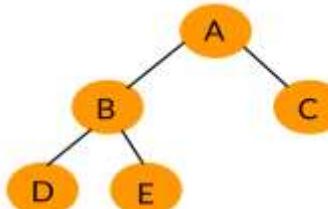
- Чиглэлтэй холбоос нь эцгээс хүү рүү чиглэж холбоно.
- Толгой зангилаа нь эцэггүй байна. Модонд ганц л толгой зангилаа байна.
- Навчин зангилаа нь хүүхэдгүй байна.
- Толгой зангилаанаас хамгийн доод талын зангилаа хүртэлх замын урт нь модны урт болно.
- Нэг эцэгтэй зангилаанууд нь ах дүүс болно.
- Хэрвээ Р зангилаа нь толгой зангилаанаас Q зангилаа хүртэлх зам дээр байвал Р зангилаа нь Q зангилааны эцэг болно.

Модны үндэс

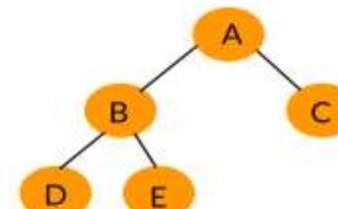


Binary Tree / Хоёртын мод

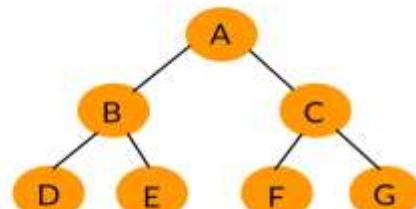
- Зангилааны хоёр дэд зангилааг зүүн дэд зангилаа ба баруун дэд зангилаа гэж ялгадаг. Хоёртын модны зангилаа нь нэг эсвэл хоёр хүүхэд зангилаа агуулж болно.
- Модны хамгийн сүүлчийн түвшингээс бусад түвшинд дотоод зангилаагаар гүйцэд дүүргэсэн бол түүнийг **дүүрэн хоёртын мод /Full Binary Tree/** гэнэ.
- Дүүрэн хоёртын модны хамгийн сүүлчийн түвшинд зөвхөн гадаад зангилааг агуулна. Харин дотоод зангилааг агуулах хамгийн сүүлчийн түвшингийн зөвхөн баруун талд зарим гадаад зангилаа байвал түүнийг **гүйцэд мод /Complete Binary Tree/** гэнэ.



Complete Binary Tree



Full Binary Tree

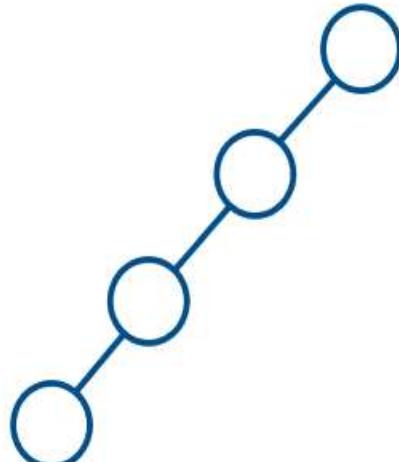


Perfect Binary Tree

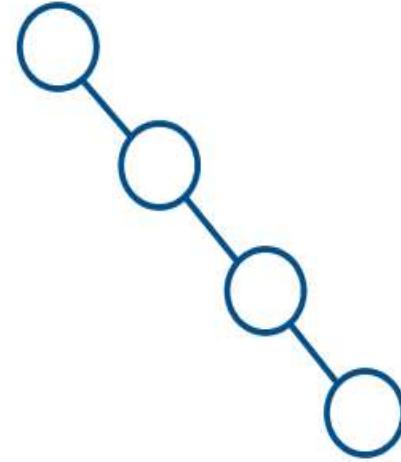
Skewed Binary Tree / Ташуу мод

- Хоёртын модны өөр нэг онцгой хэлбэр бол **ташуу хоёртын мод** юм. Ташуу хоёртын модыг зүүн ба баруун ташуу гэж үзэж болно.
- Хэрэв бүх зангилаа зүүн дэд модгүй бол баруун ташуу, хэрэв бүх зангилаа баруун дэд модгүй бол зүүн ташуу гэнэ.
- Мөн хөрөөний шүд хэлбэртэй байж болно. Өөрөөр хэлбэл зангилаа бүр зөвхөн зүүн эсвэл баруун дэд модтой байна гэсэн үг юм.
- Хоёртын мод нь компьютерын хэрэглээнд маш өргөн ашиглагдах ба тэрээр дүүрэн эсвэл гүйцэд хоёртын мод байх үед хамгийн үр ашигтай юм.

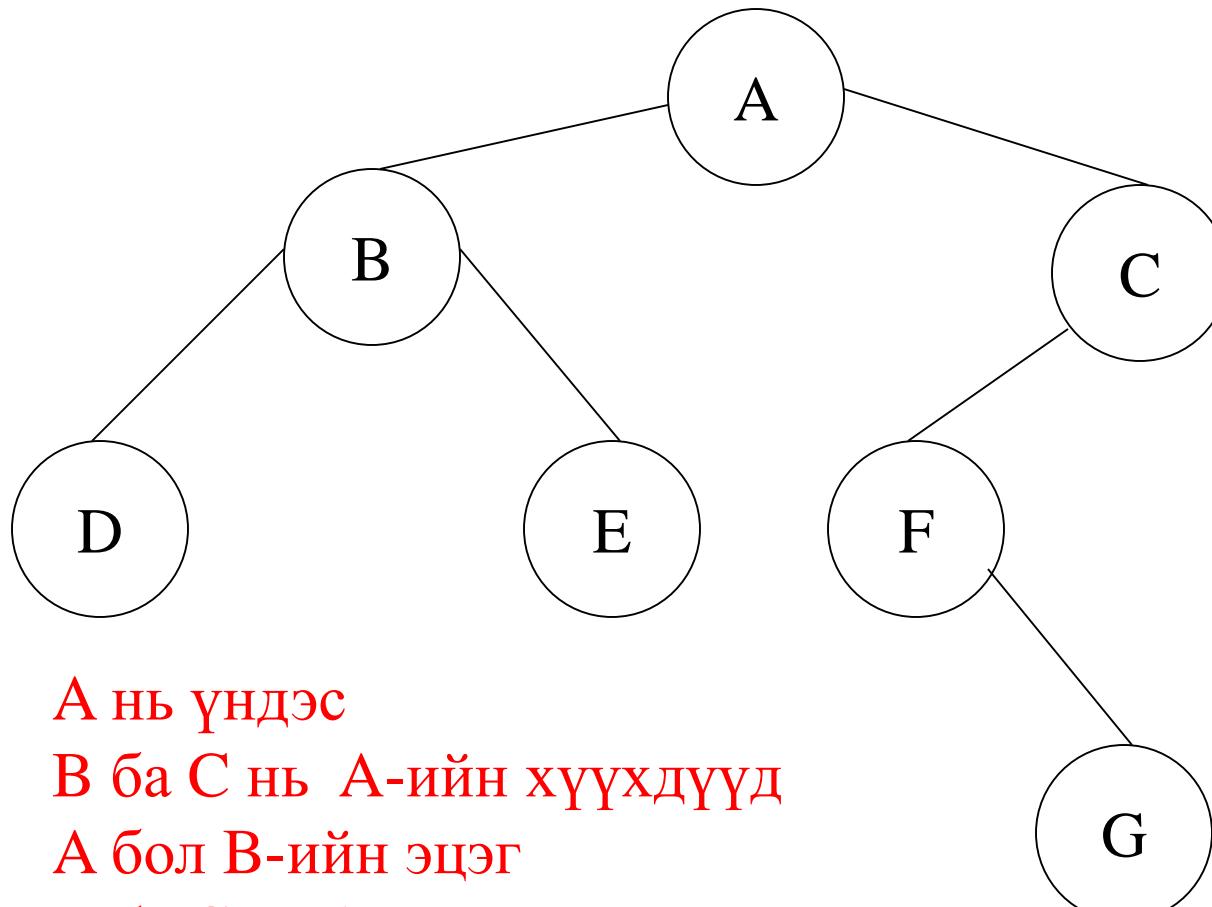
- Left skewed binary tree



- Right skewed binary tree



Binary Tree / Хоёртын модны чанар

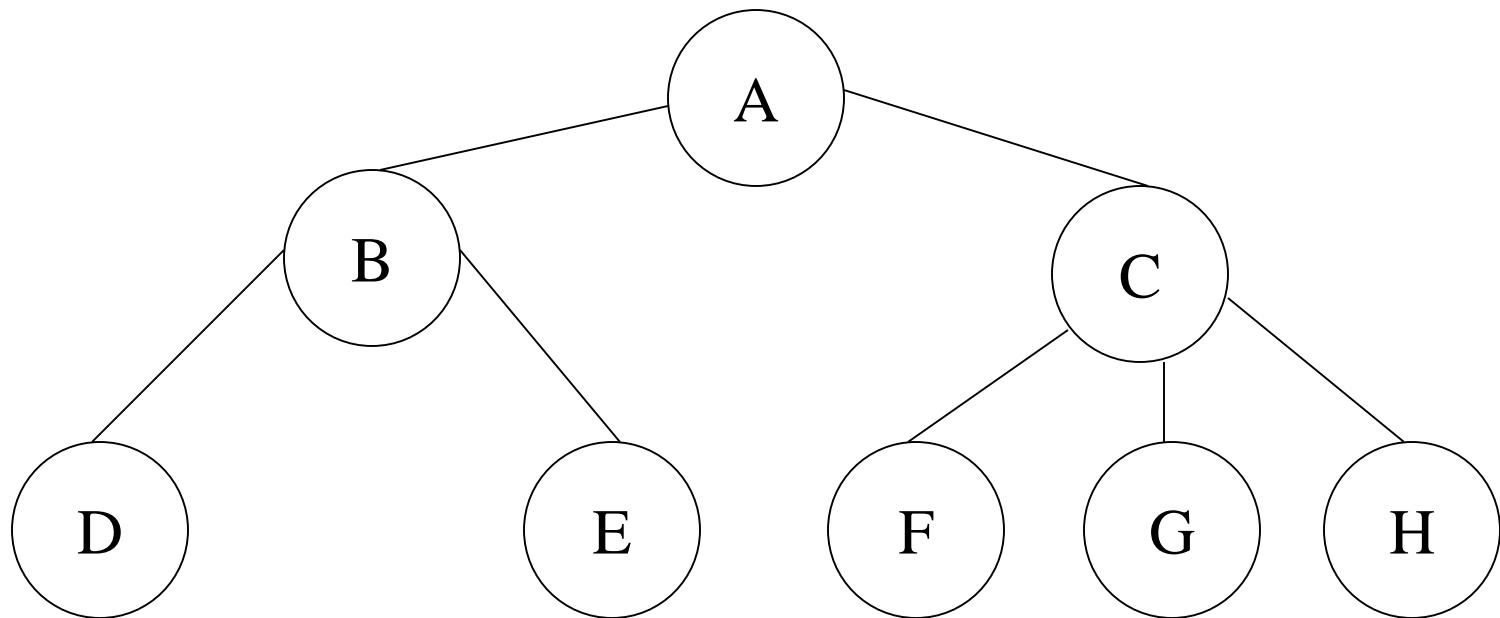


- А нь үндэс
- В ба С нь А-ийн хүүхдүүд
- А бол В-ийн эцэг
- В ба С нь А үндэсний дэд моднууд
- D, E, G нь навчнууд

Хоёртын модны чанар

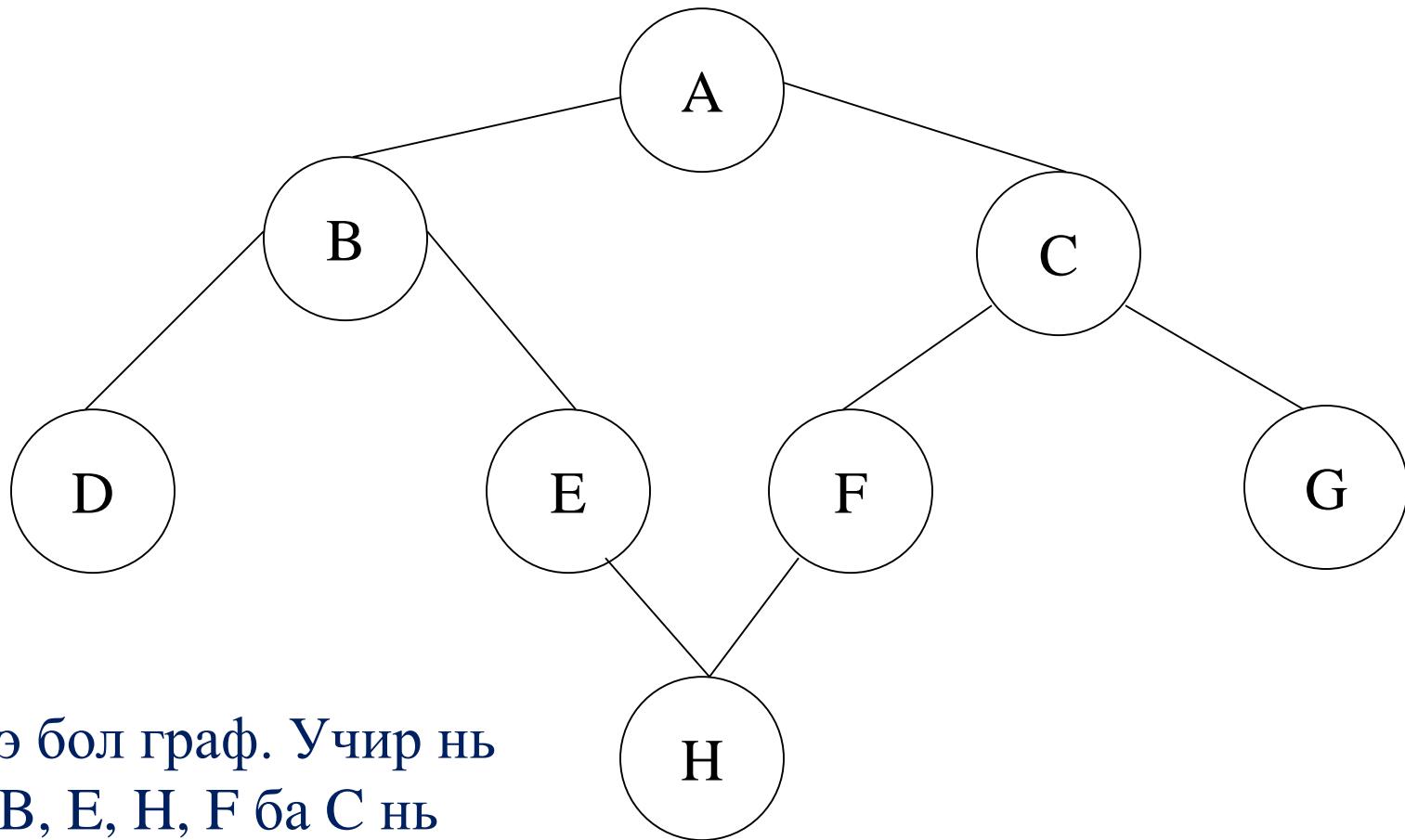
- **Чанар 1:** Модны аливаа хоёр зангилааг холбох ганц зам байна. Аливаа хоёр зангилааны хувьд ядаж нэг ерөнхий зангилаа олдоно. Ийм ерөнхий эх зангилаанаас уг хоёр зангилаанд хүрэх зам нь давтагдашгүй бөгөөд эдгээрийг нийлүүлснээр зөвхөн ганц зам олдоно.
- **Чанар 2:** N зангилаа бүхий мод N-1 холбоостой байна. Энэ чанар нь үндсэн зангилаанаас бусад бүх зангилаа нь зөвхөн ганц эх зангилаанд шууд холбогдоно гэдгээс мөрдөн гарна.
- **Чанар 3:** N дотоод зангилаа бүхий хоёртын мод нь N+1 гадаад зангилаатай байна.
- **Чанар 4:** N дотоод зангилаа бүхий хоёртын модны гадаад замын урт нь дотоод замын уртаас $2N$ -ээр илүү байдаг.
- **Чанар 5:** Хоёртын модны i дүгээр түвшинд хамгийн багадаа 2^i зангилаа байна.
- **Чанар 6:** n өндөртэй хоёртын модны максиум зангилааны тоо нь $2^n - 1$ тэнцүү байна (дүүрэн модны зангилааны тоо).
- **Чанар 7:** N дотоод зангилаа бүхий дүүрэн хоёртын модны өндөр нь ойролцоогоор $\log_2 N$ байна.

Энэ бол Хоёртын Мод биш



- Учир С нь гурван хүүхэд зангилаатай учраас энэ нь ерөнхий мод юм.

Энэ бол Хоёртын Мод биш



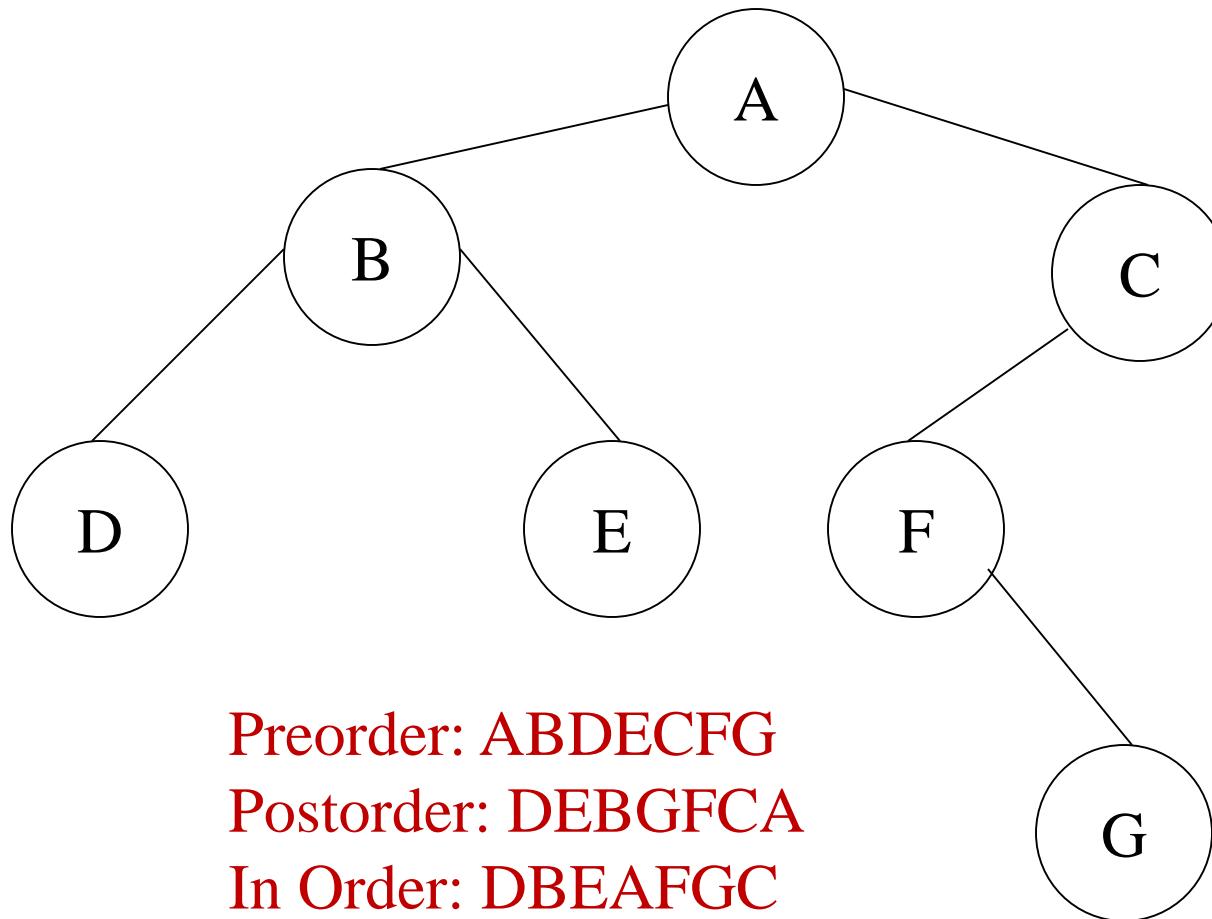
Энэ бол граф. Учир нь
A, B, E, H, F ба C нь
хоорондоо холбогдоно.

Хоёртын модны шинж чанарууд

- Төгс **хоёртын модны** н зангилааны тоог олохдоо дараах томъёог ашиглана. $n = 2^h + 1 - 1$ (h нь модны түвшин)
- **Хоёртын модны** н зангилааны тоог олохдоо өндөр h багадаа $n = h + 1$ ихдээ $n = 2^h + 1 - 1$ (h нь модны түвшин)
- Төгс **хоёртын модны** навчит зангилаануудын тоог олохдоо дараах томъёог ашиглана. $I = 2^h$
- Төгс **хоёртын модны** н зангилааны тоог олохдоо дараах томъёог ашиглана. $n = 2I - 1$ (I нь модны навчит зангилаануудын тоо)
- Бүрэн төгс **хоёртын модны** Null холбоосын тоог олохдоо (хүүхэдгүй зангилаанууд) $(n+1)$.
- Бүрэн төгс **хоёртын модны** дотоод зангилааны тоо (навчгүй зангилаа) $\lfloor n/2 \rfloor$.

Tree Traversal / Модны нэвтрэлт

- Хоёртын модонд нэвтрэлт нь сонгосон алгоритмаасаа хамаарч Preorder, Inorder, Postorder, Levelorder гэсэн 4 төрөлд хуваагддаг.



Preorder: ABDEC_FG

Postorder: DEBGFC_A

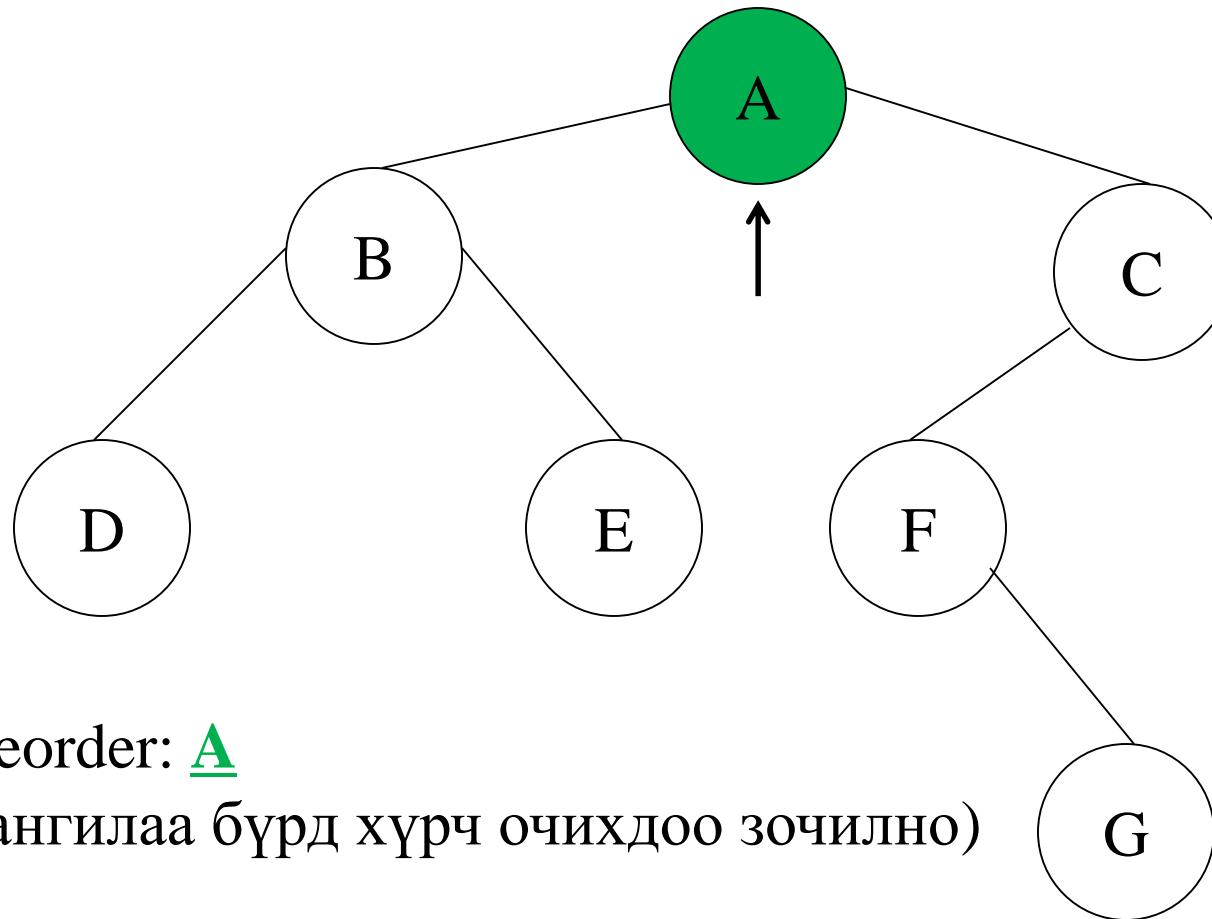
In Order: DBEA_FGC

Preorder нэвтрэлт

```
public static void preOrder(BinaryTreeNode t) { if (t !=  
null)  
{ visit(t);  
preOrder(t.leftChild);  
preOrder(t.rightChild);  
}  
}
```

- 1. Үндэсдээ нэвтрэнэ.
- 2. Зүүн хүүхэддээ нэвтрэнэ.
- 3. Баруун хүүхэддээ нэвтрэнэ.

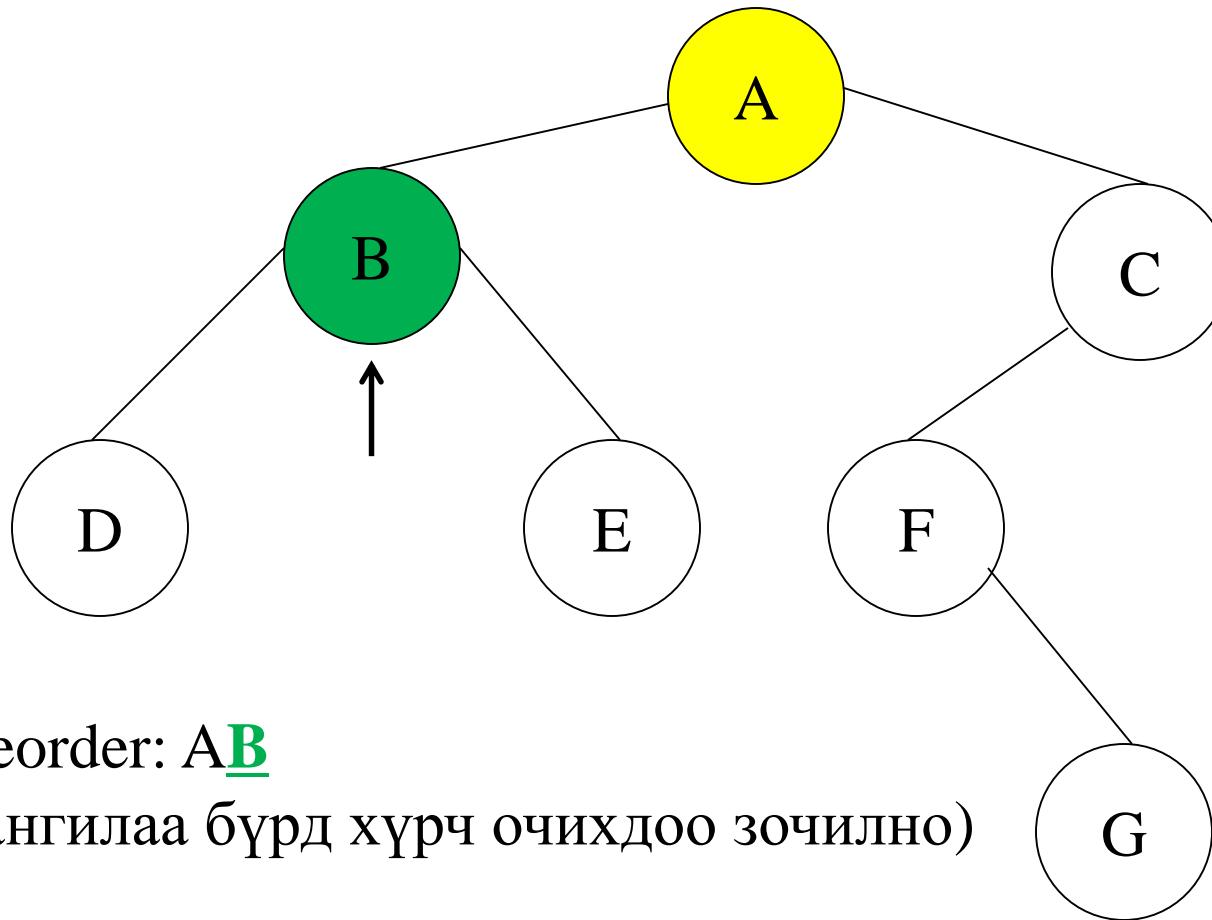
Preorder нэвтрэлтийн жишээ



Preorder: A

(зангилаа бүрд хүрч очихдоо зочилно)

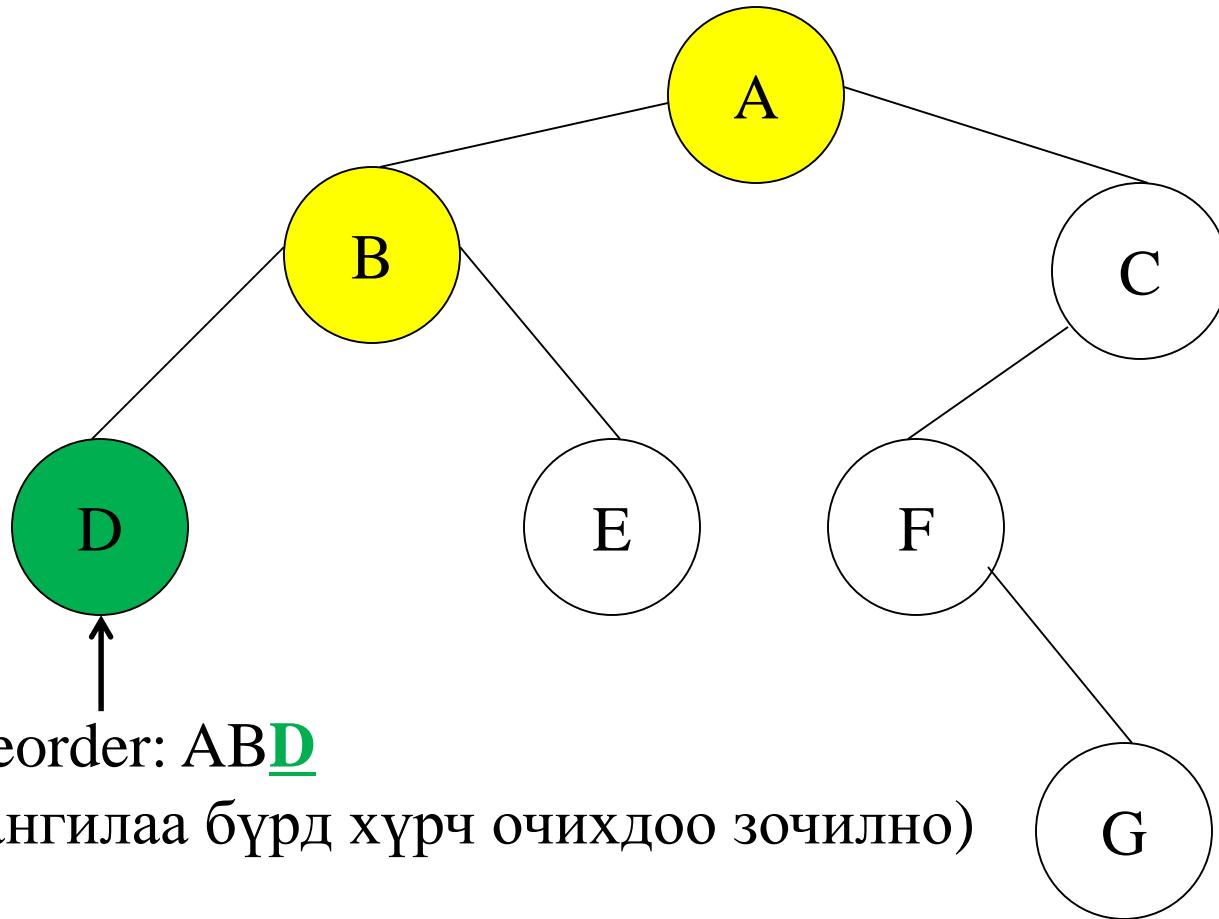
Preorder нэвтрэлтийн жишээ



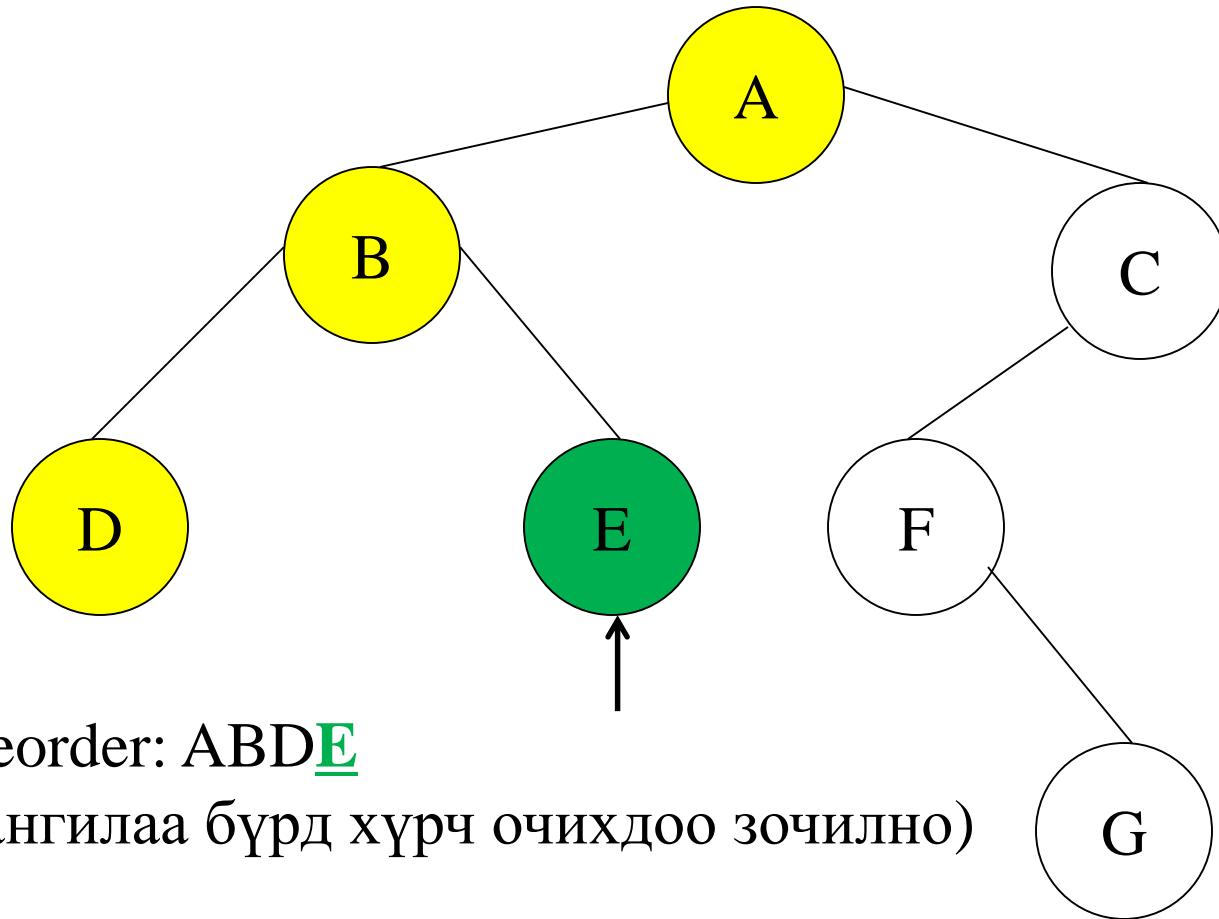
Preorder: AB

(зангилаа бүрд хүрч очихдоо зочилно)

Preorder нэвтрэлтийн жишээ



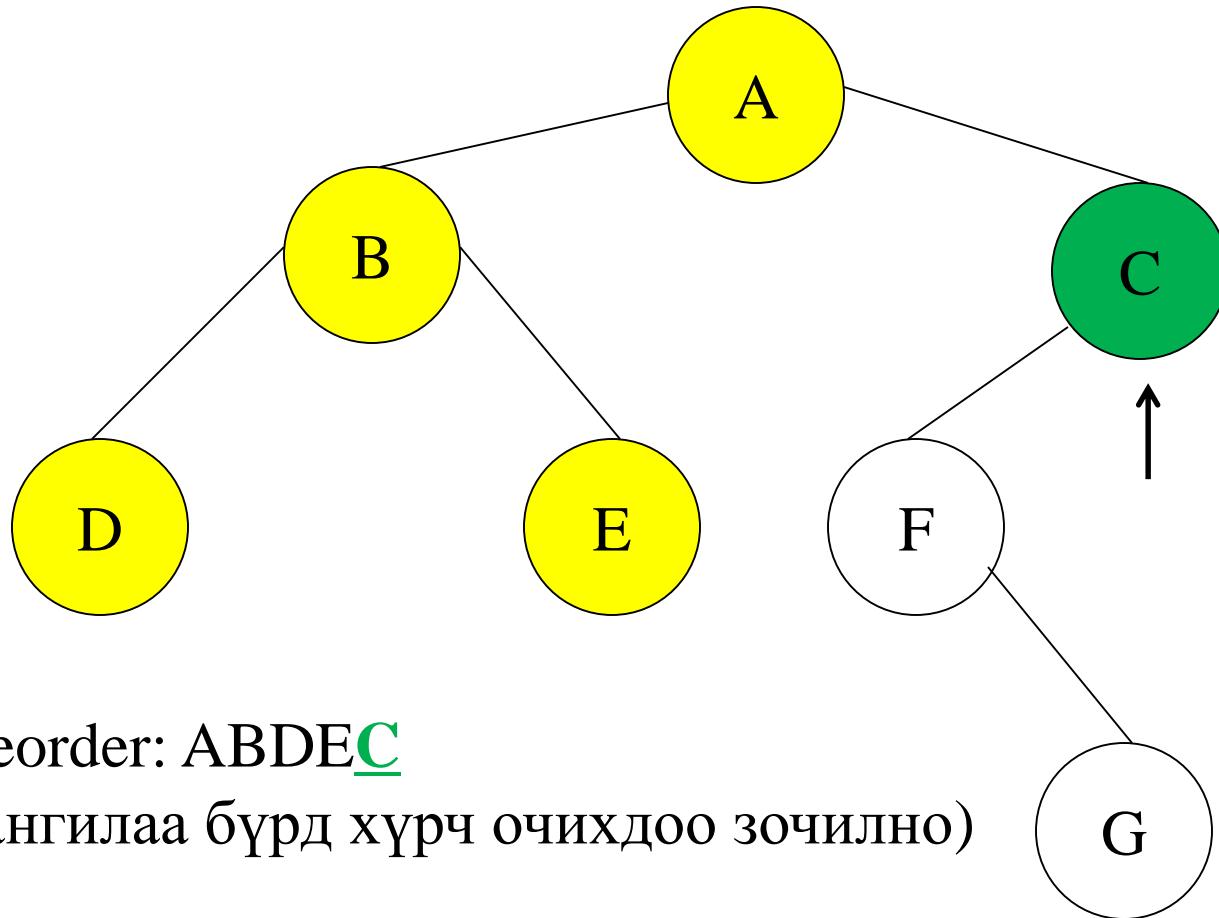
Preorder нэвтрэлтийн жишээ



Preorder: ABDE

(зангилаа бүрд хүрч очихдоо зочилно)

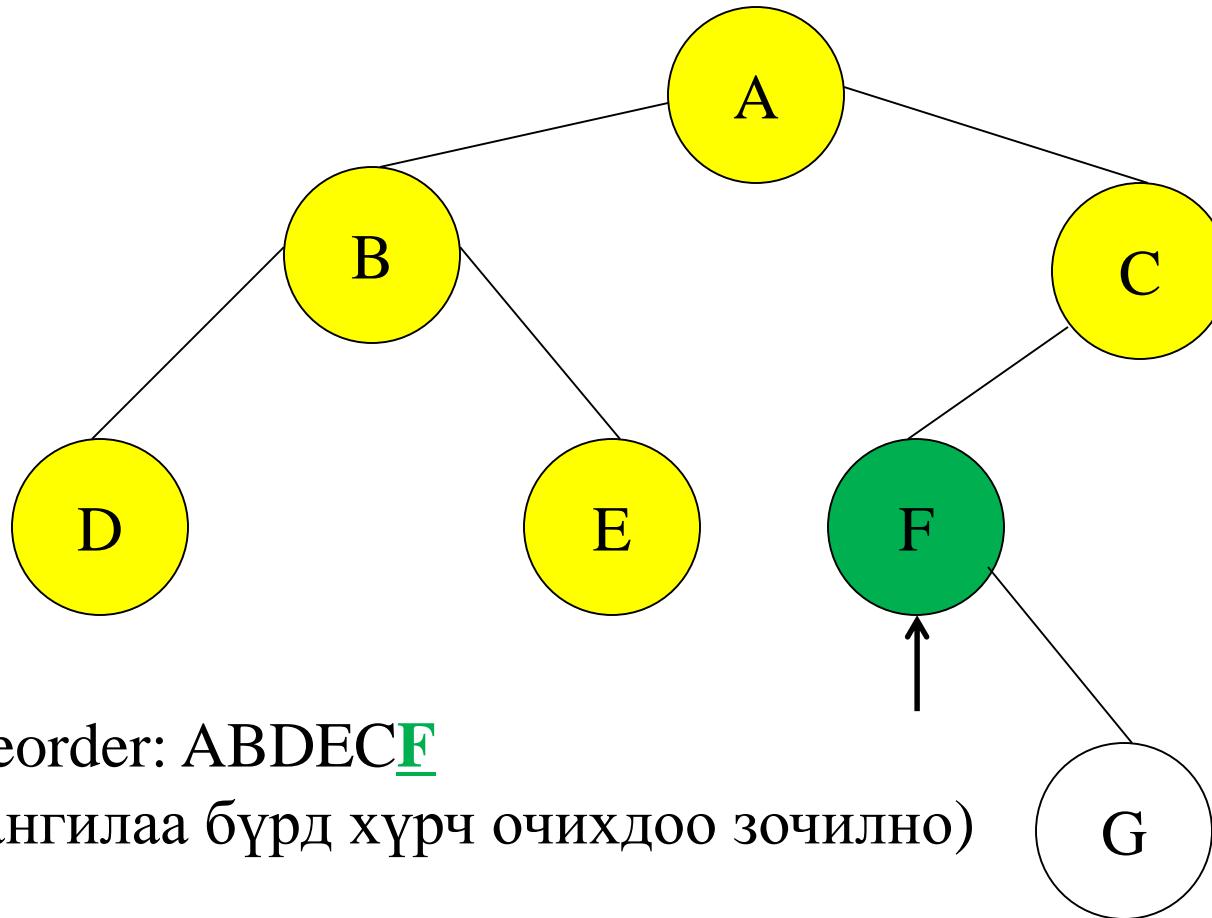
Preorder нэвтрэлтийн жишээ



Preorder: ABDEC

(зангилаа бүрд хүрч очихдоо зочилно)

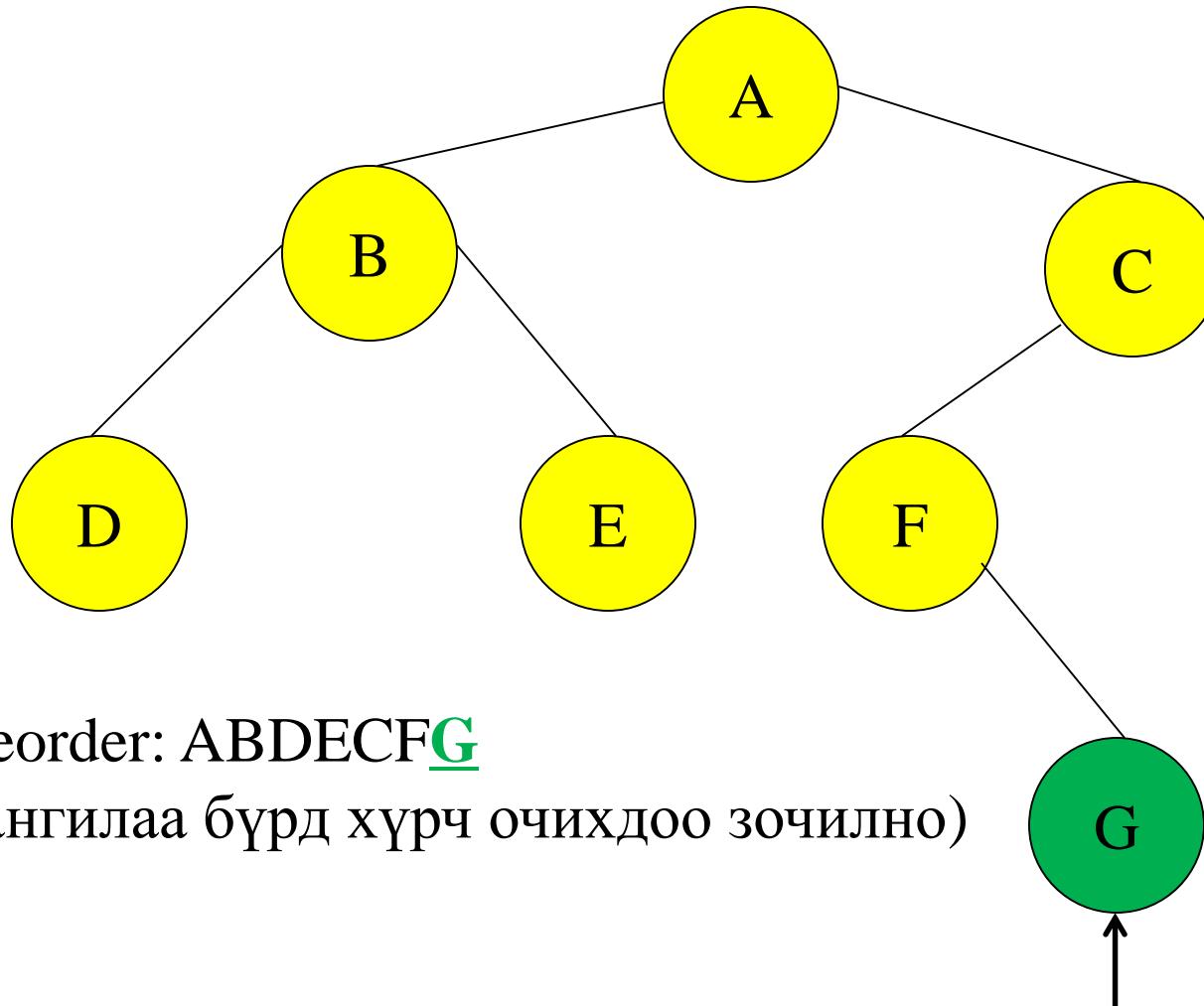
Preorder нэвтрэлтийн жишээ



Preorder: ABDECF

(зангилаа бүрд хүрч очихдоо зочилно)

Preorder нэвтрэлтийн жишээ



Preorder: AB~~DE~~CFG

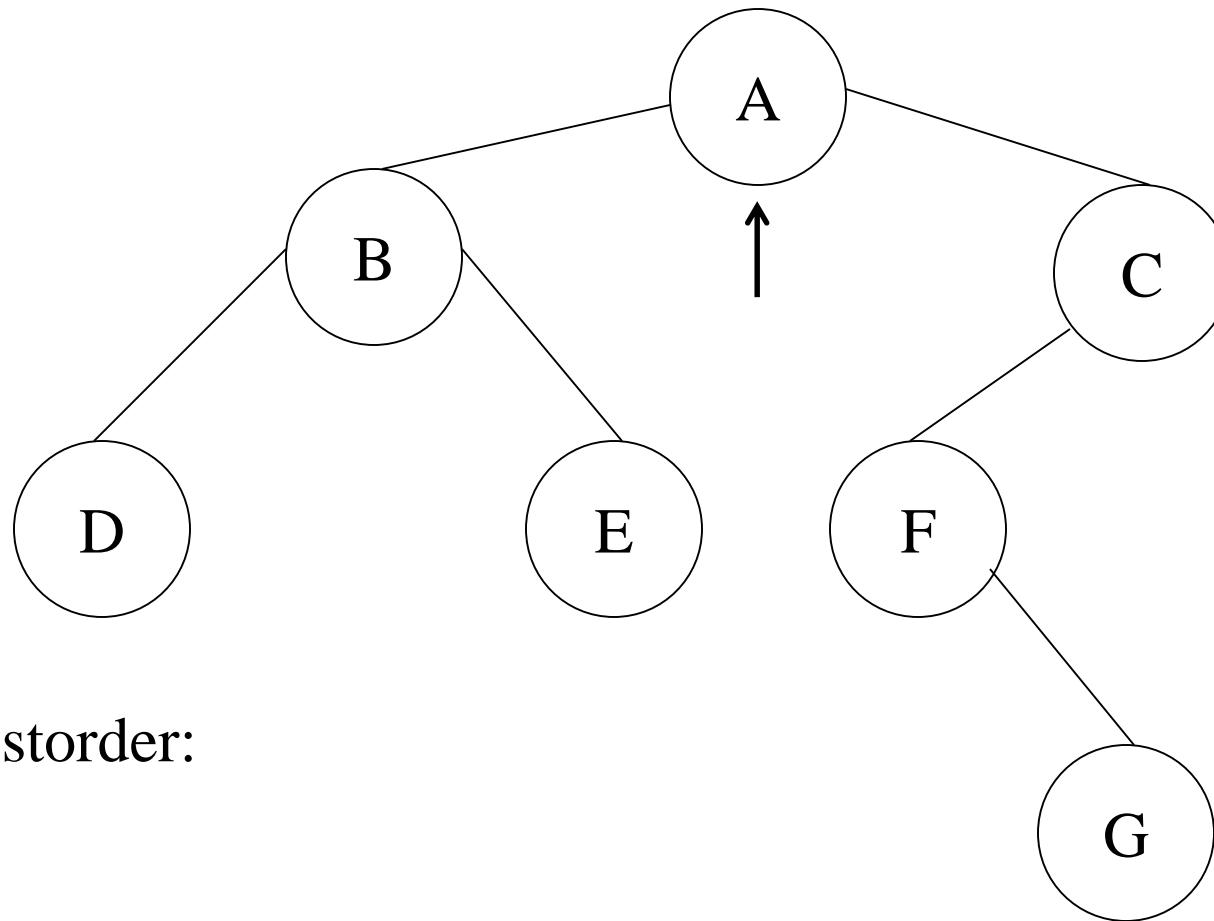
(зангилаа бүрд хүрч очихдоо зочилно)

PostOrder нэвтрэлт:

```
public static void postOrder(BinaryTreeNode t)
{ if (t != null)
{ postOrder(t.leftChild);
postOrder(t.rightChild);
visit(t);
}
}
```

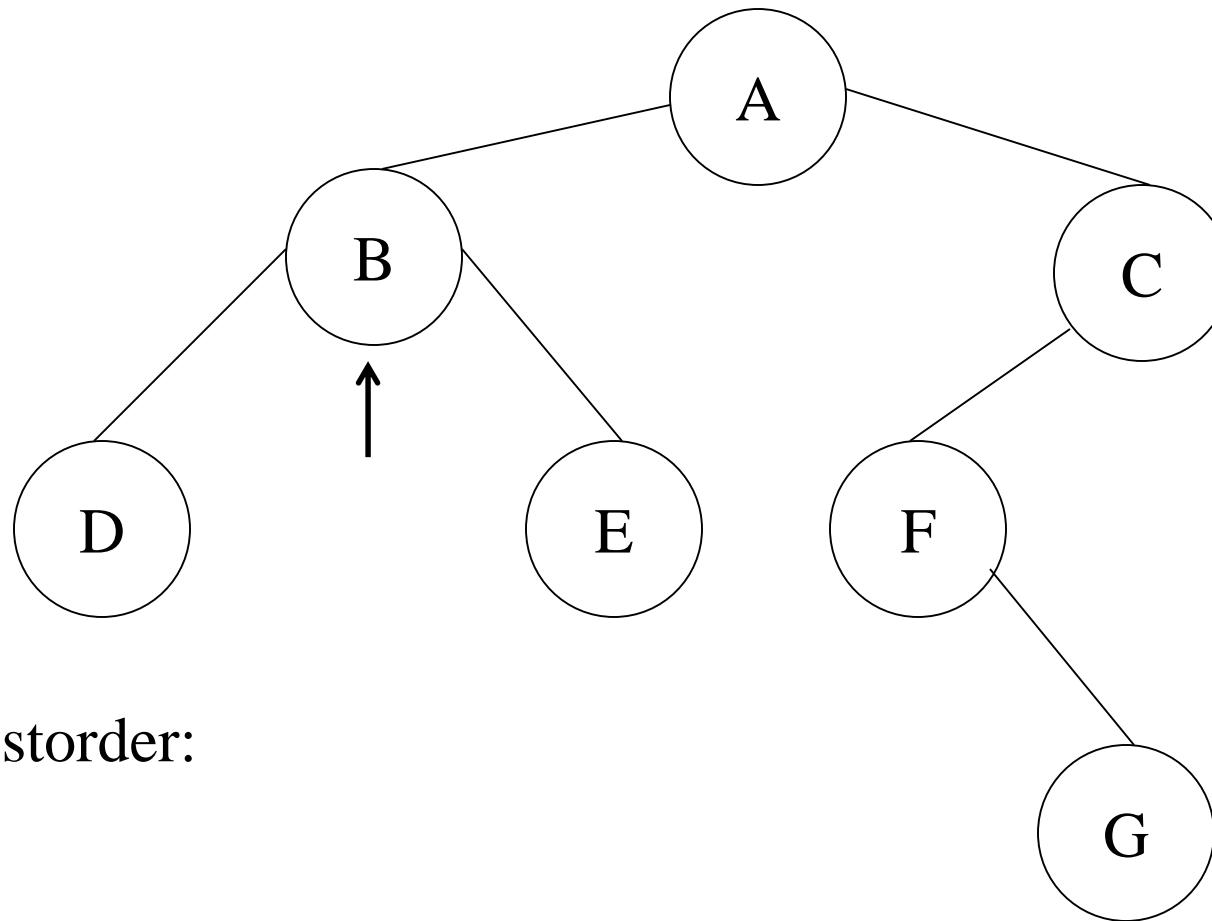
1. Зүүн хүүхэддээ нэвтрэнэ.
2. Баруун хүүхэддээ нэвтрэнэ.
3. Үндэсдээ нэвтрэнэ.

Postorder нэвтрэлтийн жишээ



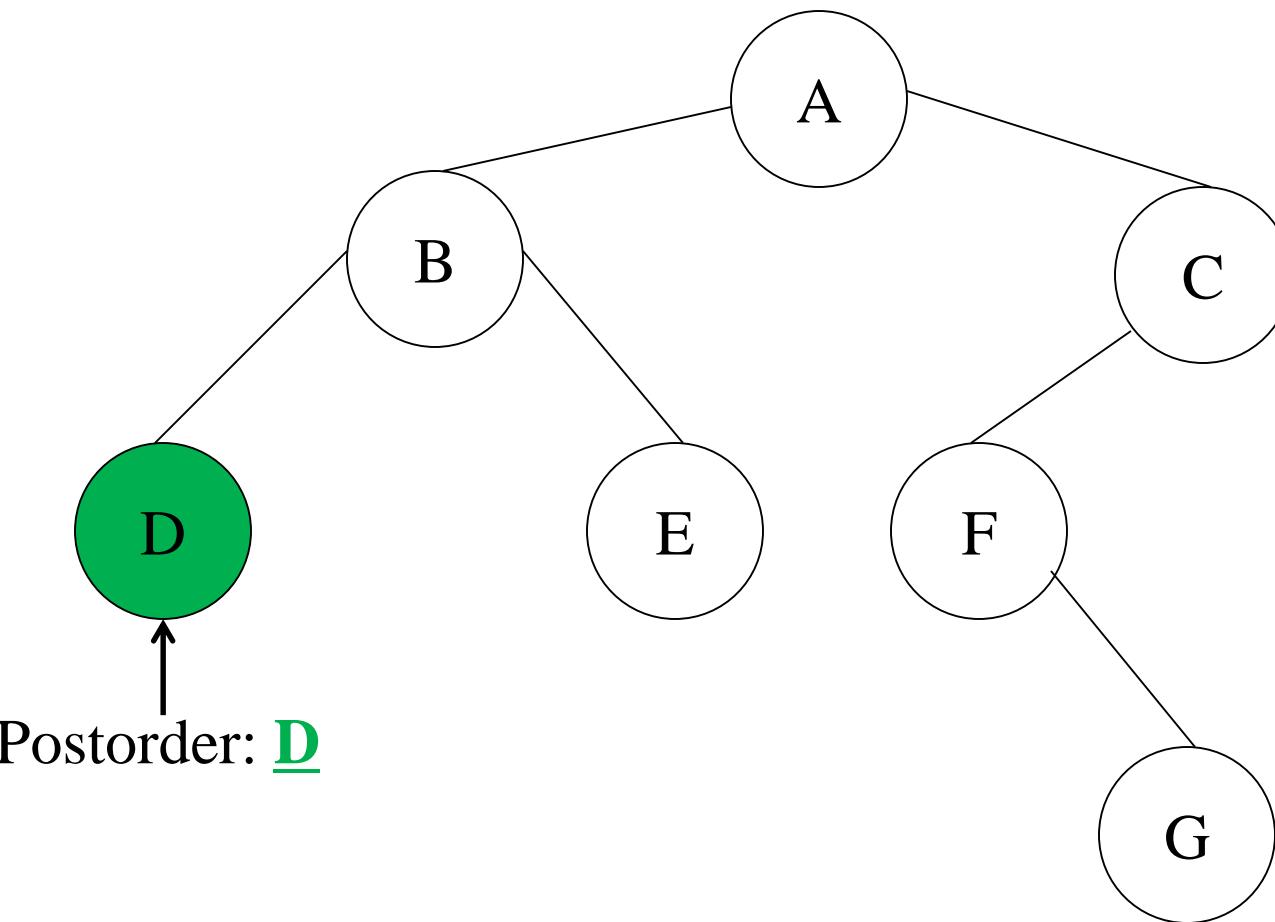
Postorder:

Postorder нэвтрэлтийн жишээ

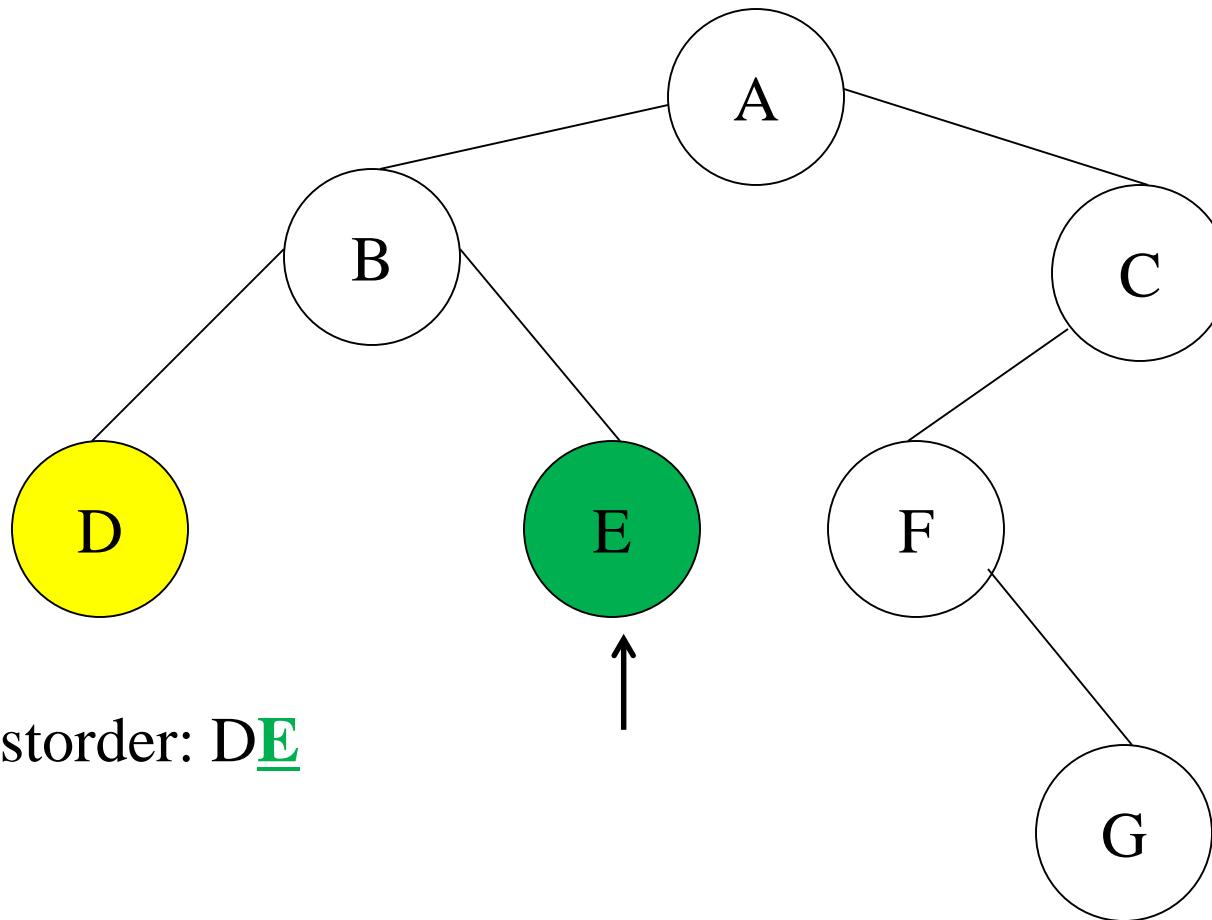


Postorder:

Postorder нэвтрэлтийн жишээ

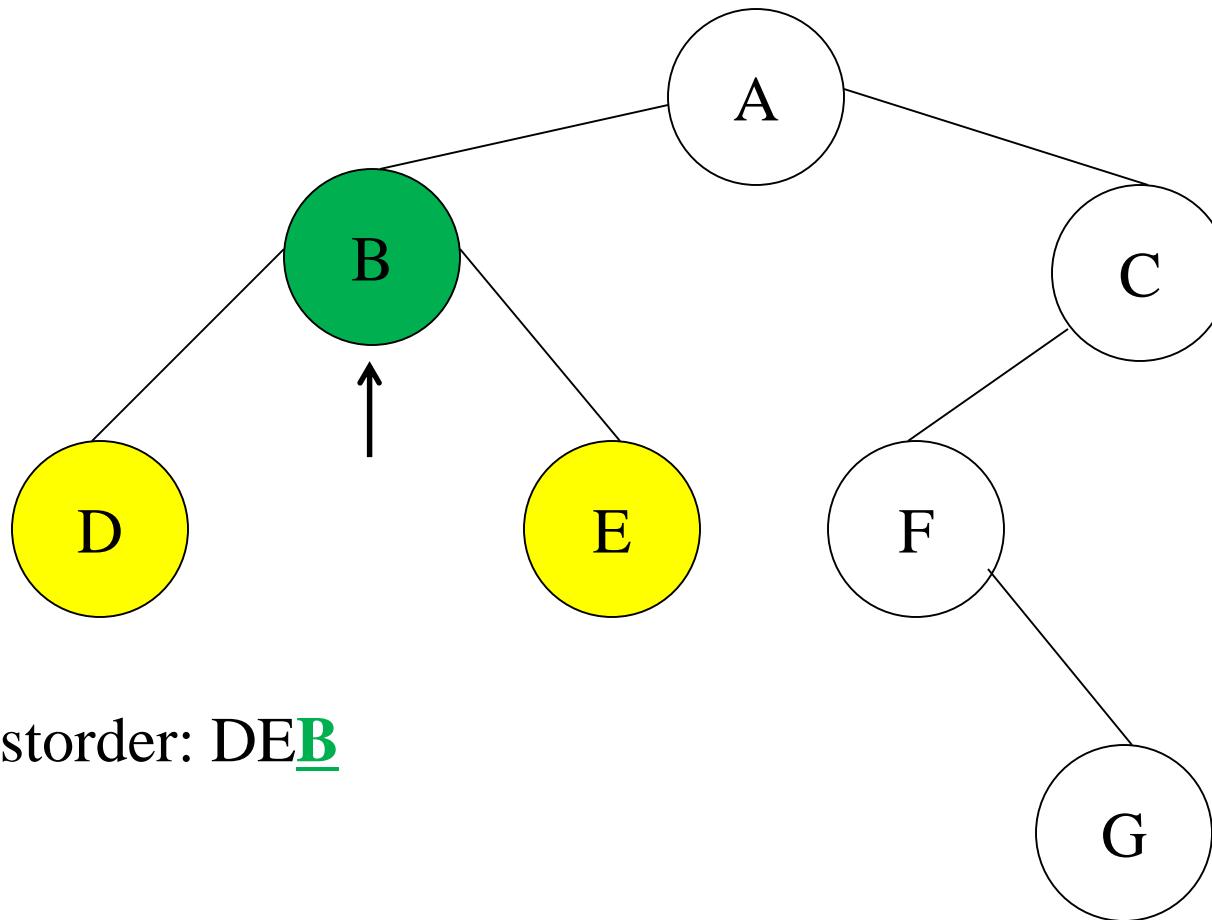


Postorder нэвтрэлтийн жишээ

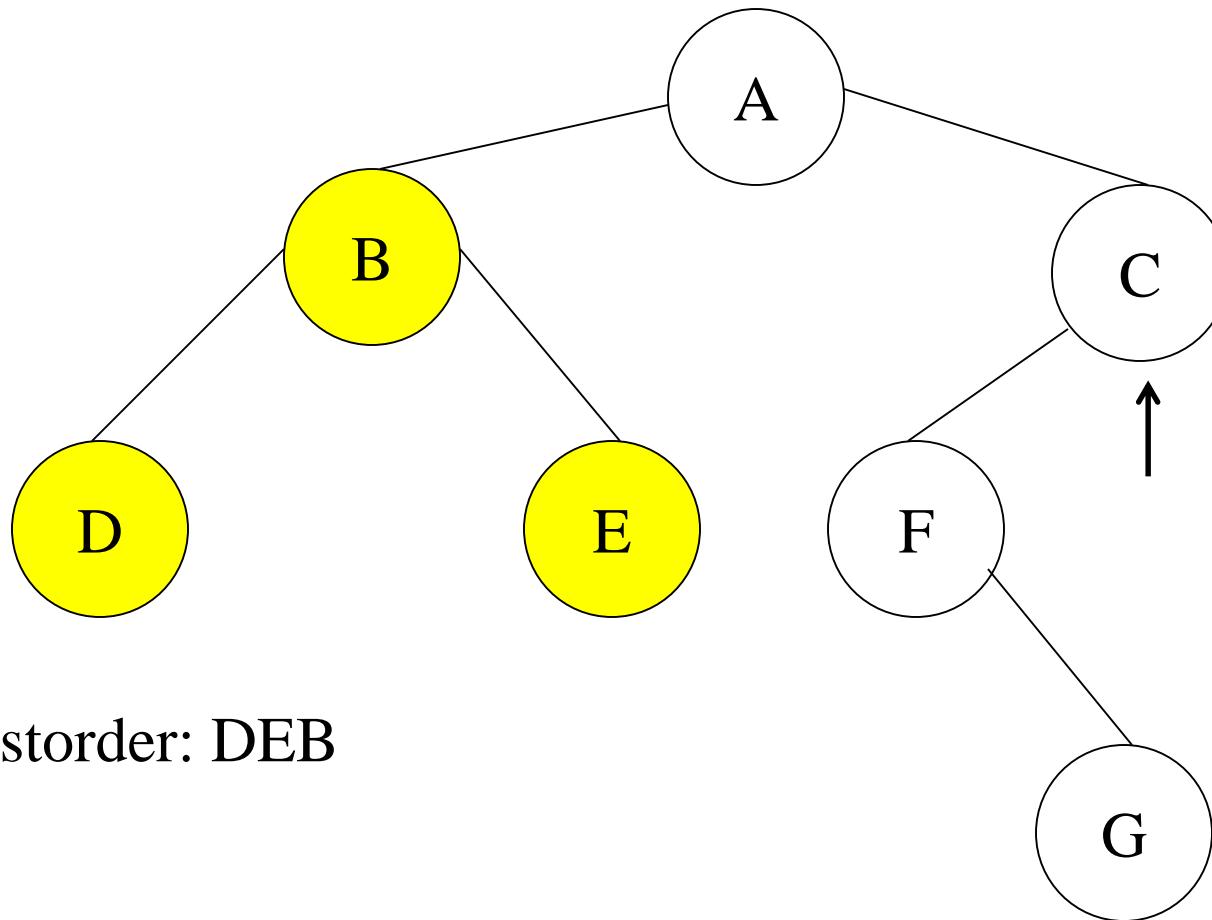


Postorder: DE

Postorder нэвтрэлтийн жишээ

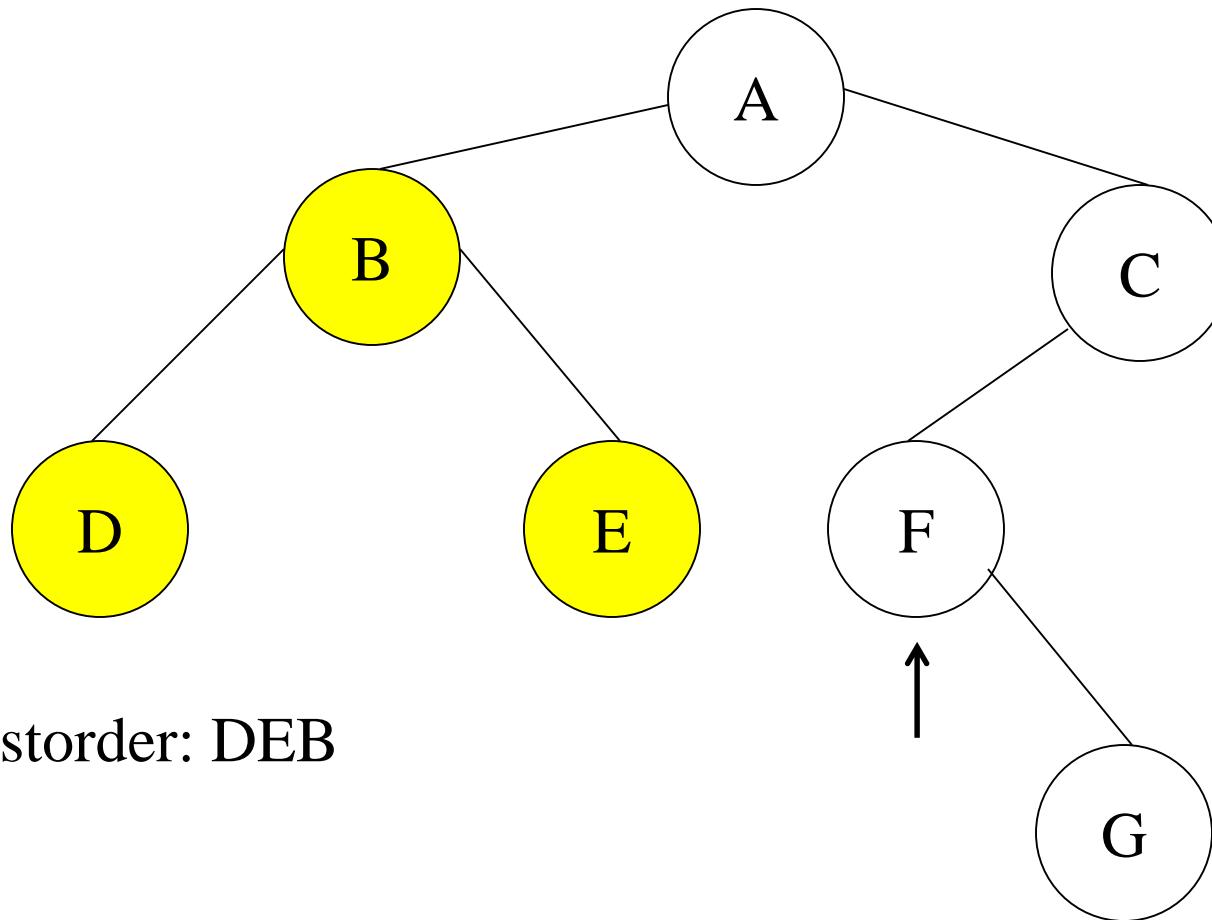


Postorder нэвтрэлтийн жишээ



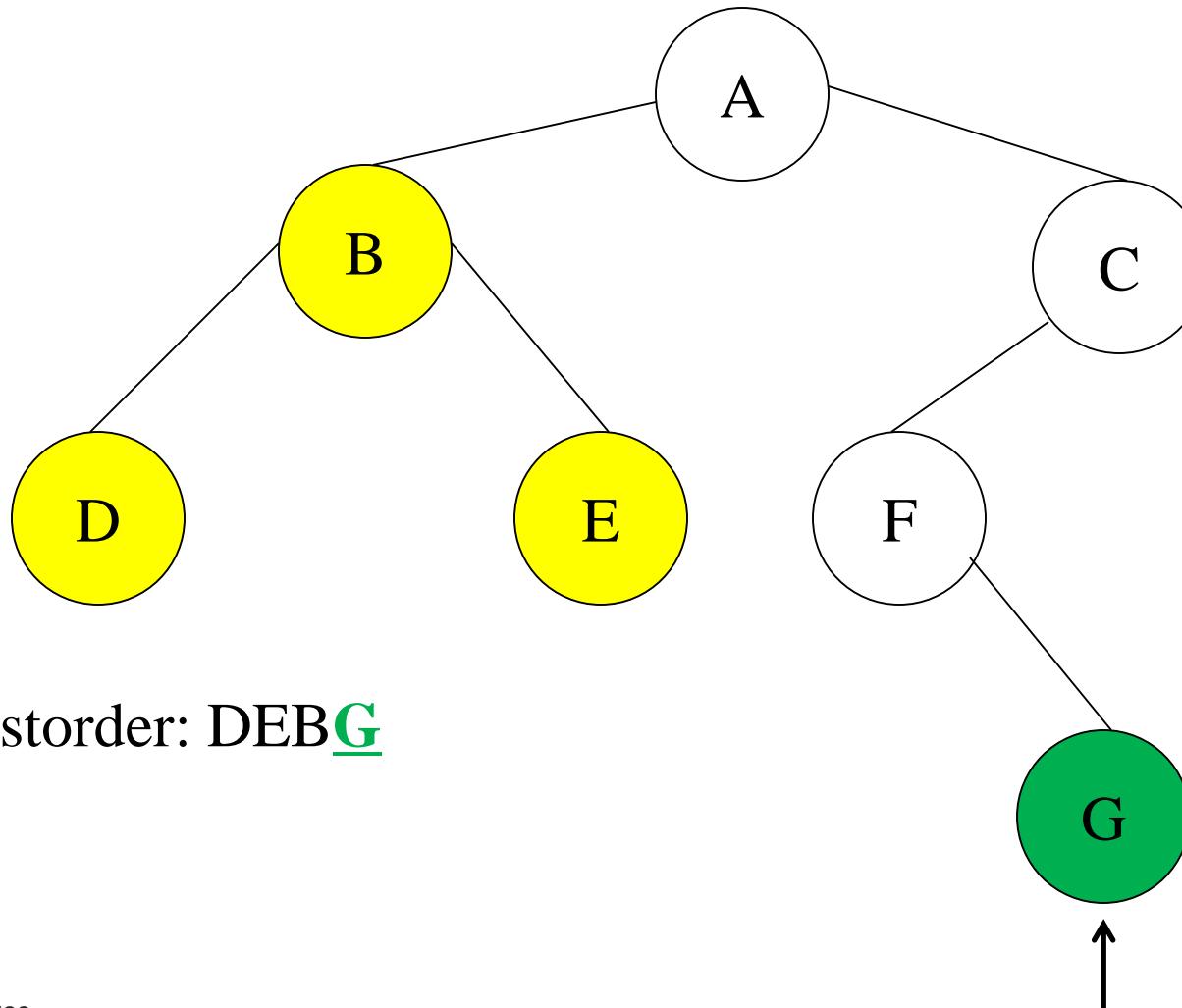
Postorder: DEB

Postorder нэвтрэлтийн жишээ

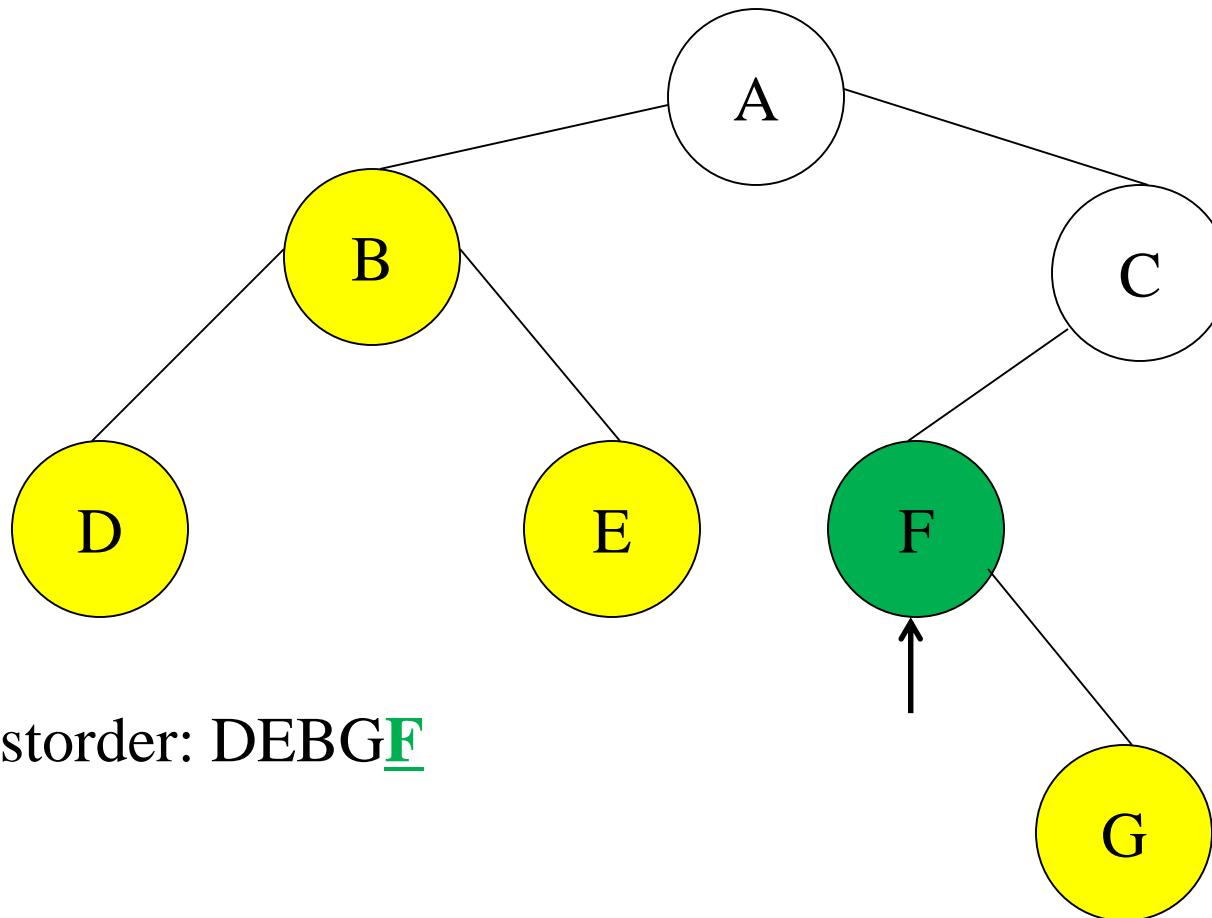


Postorder: DEB

Postorder нэвтрэлтийн жишээ

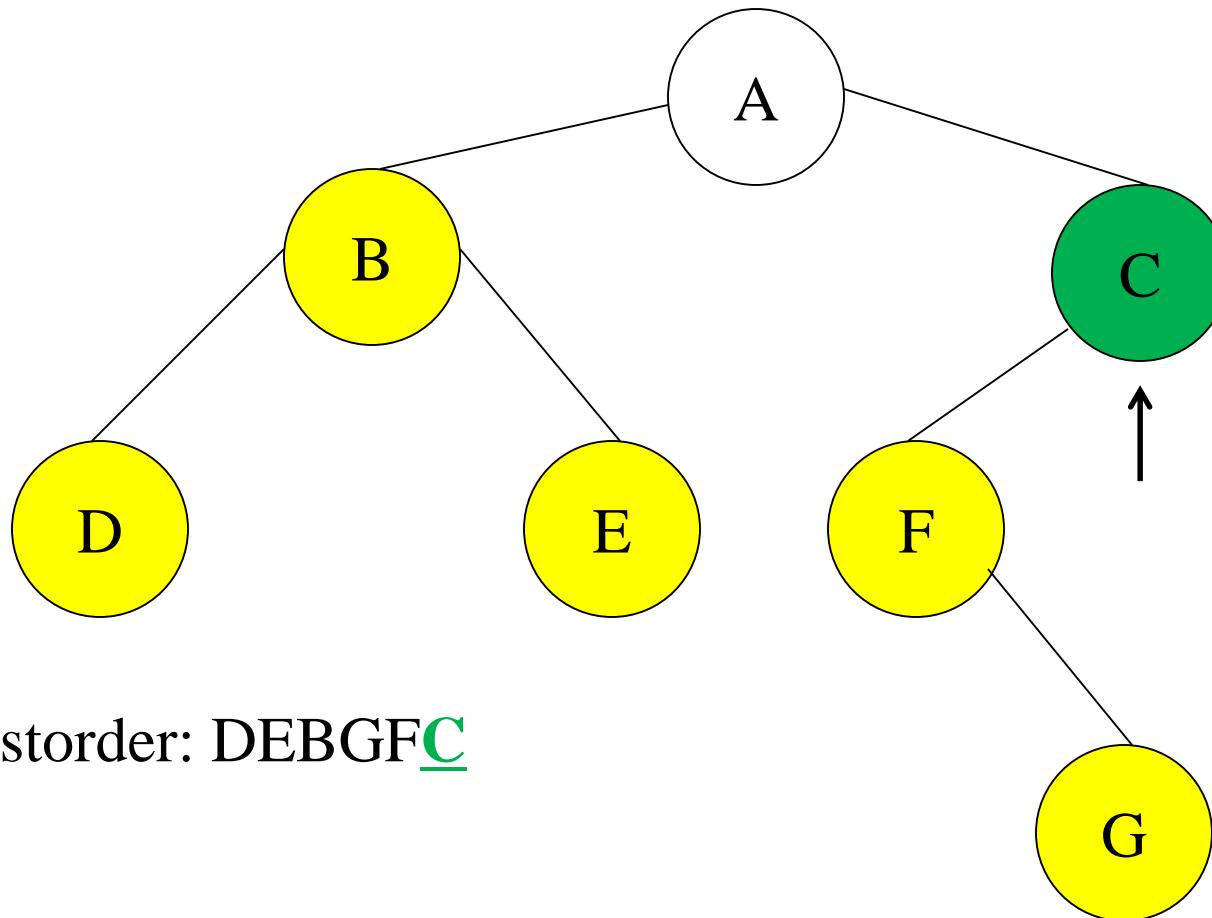


Postorder нэвтрэлтийн жишээ



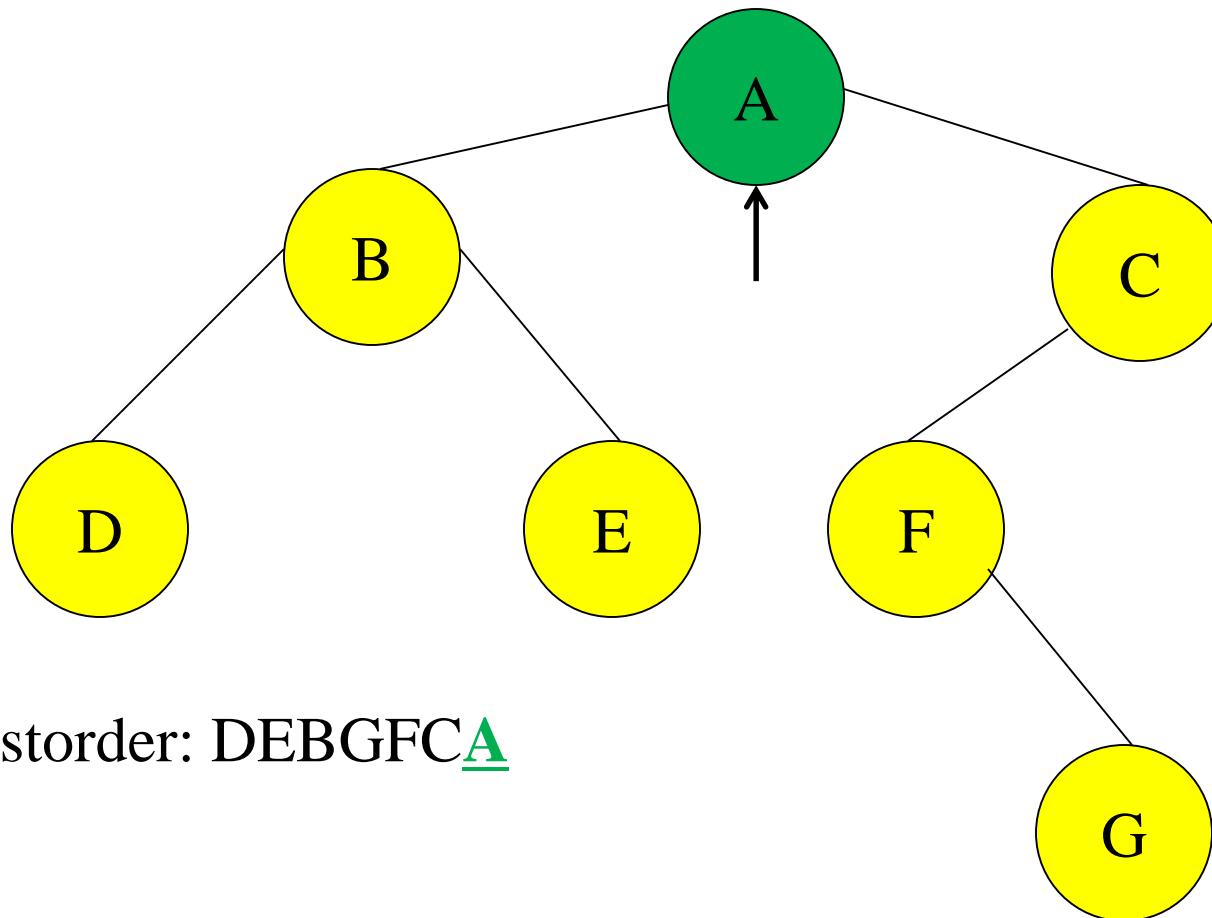
Postorder: DEBGF

Postorder нэвтрэлтийн жишээ



Postorder: DEBGFC

Postorder нэвтрэлтийн жишээ



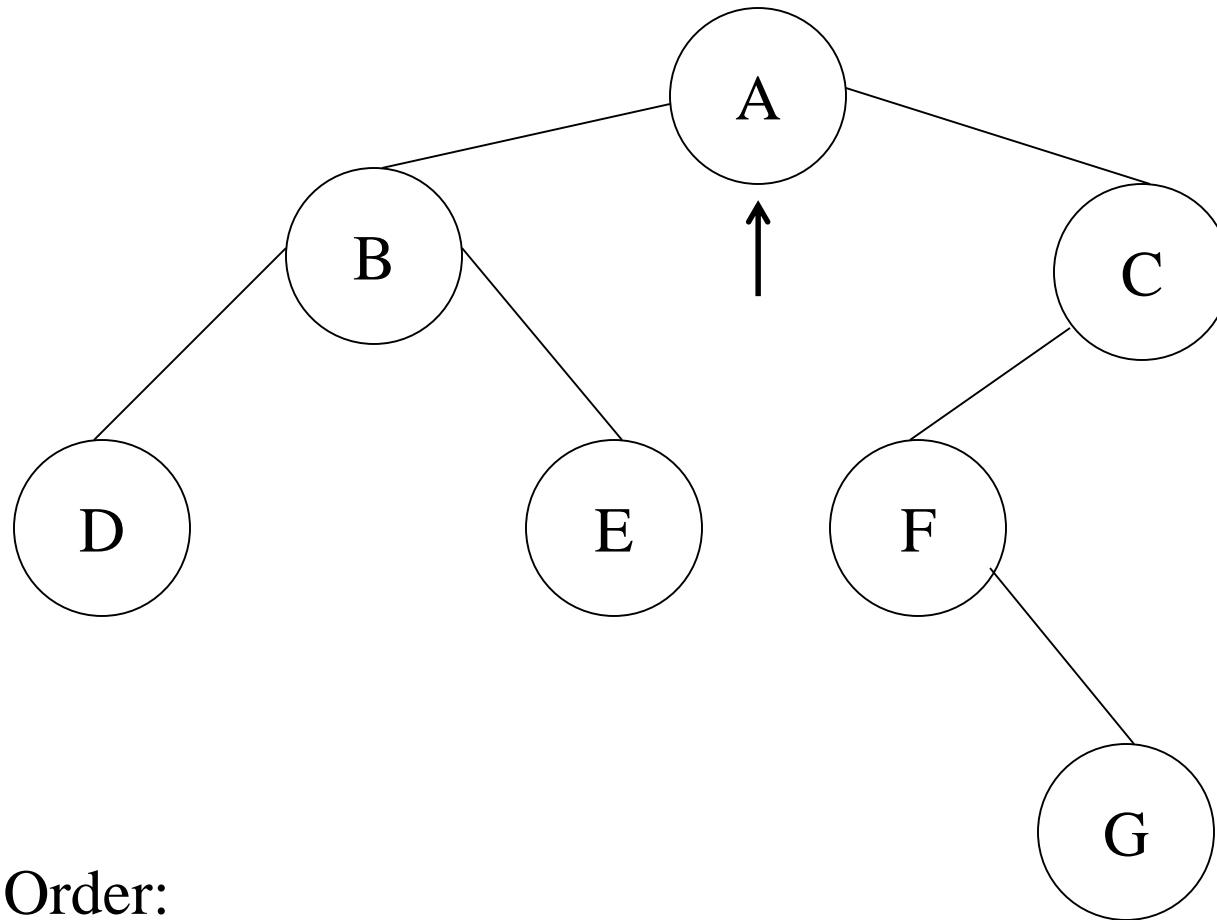
Postorder: DEBGFCA

in Order нэвтрэлт:

```
public static void inOrder(BinaryTreeNode t)
{ if (t != null)
{ inOrder(t.leftChild);
visit(t);
inOrder(t.rightChild);
}
}
```

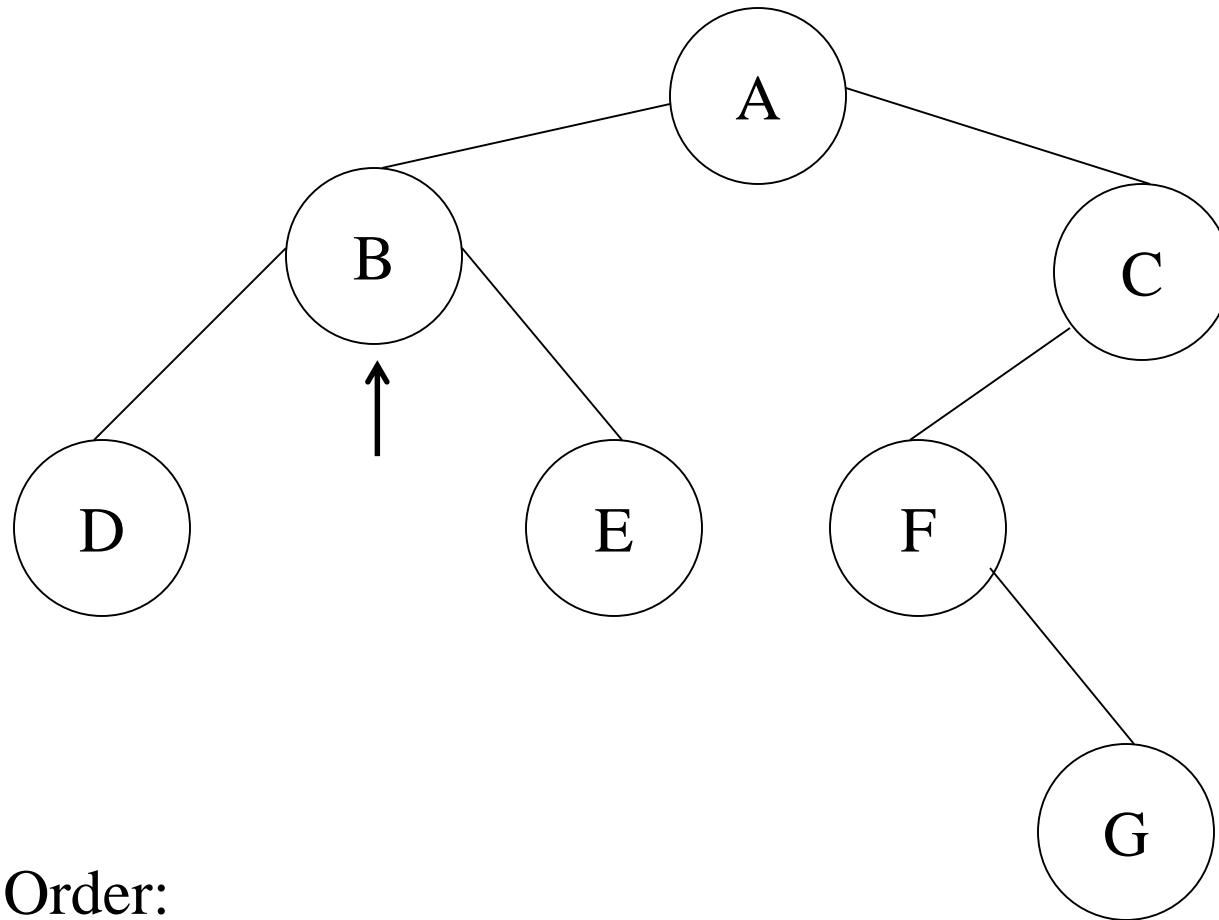
1. Зүүн хүүхэддээ нэвтрэнэ.
2. Үндэсдээ нэвтрэнэ.
3. Баруун хүүхэддээ нэвтрэнэ.

In Order нэвтрэлтийн жишээ



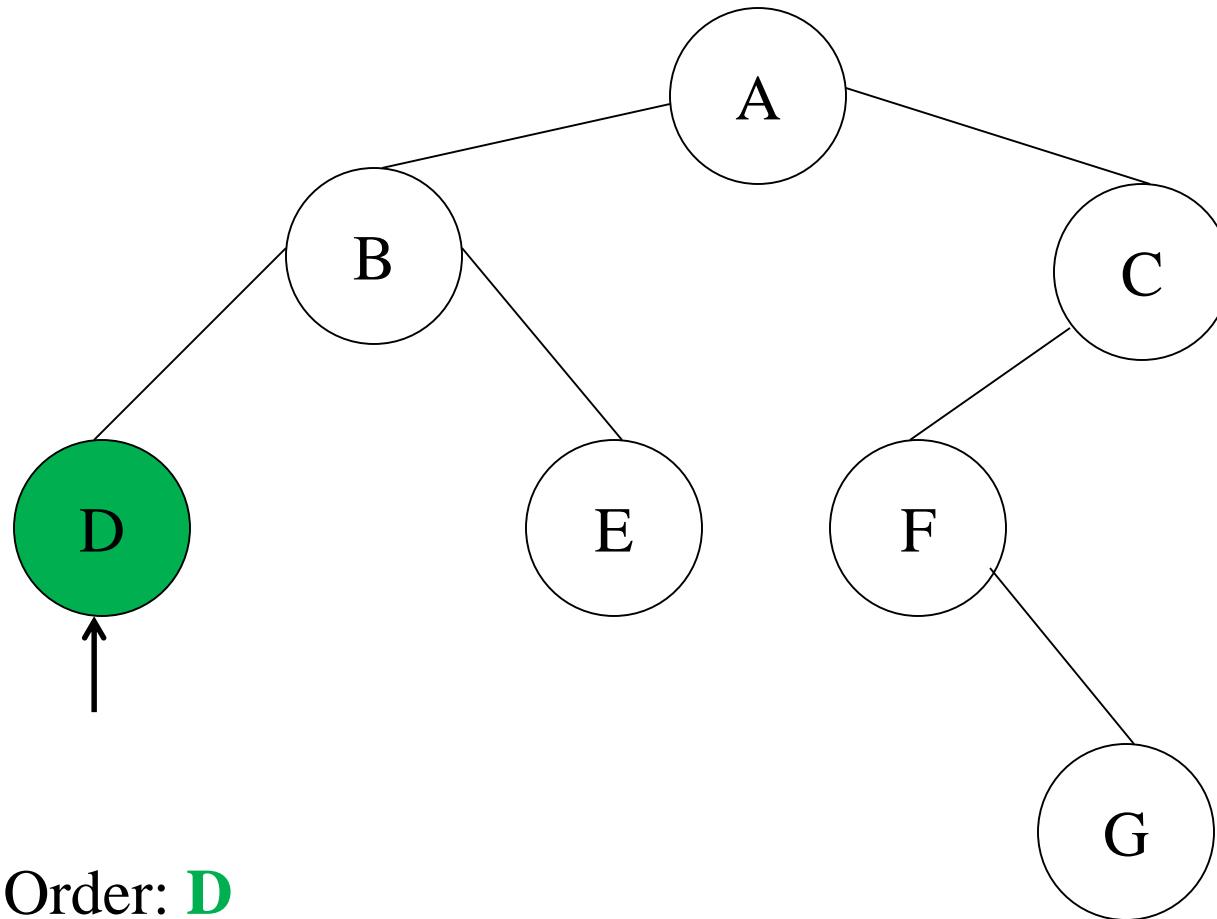
In Order:

In Order нэвтрэлтийн жишээ



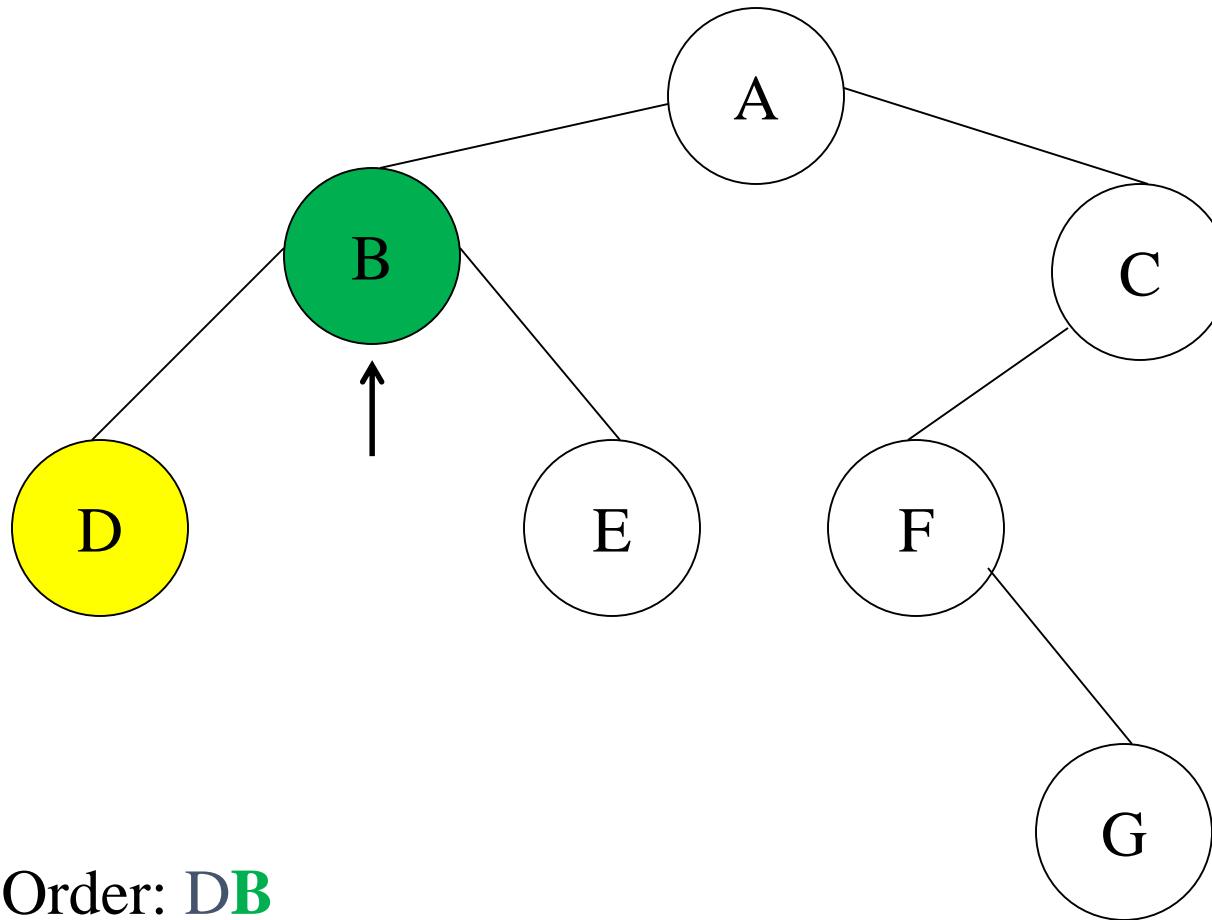
In Order:

In Order нэвтрэлтийн жишээ



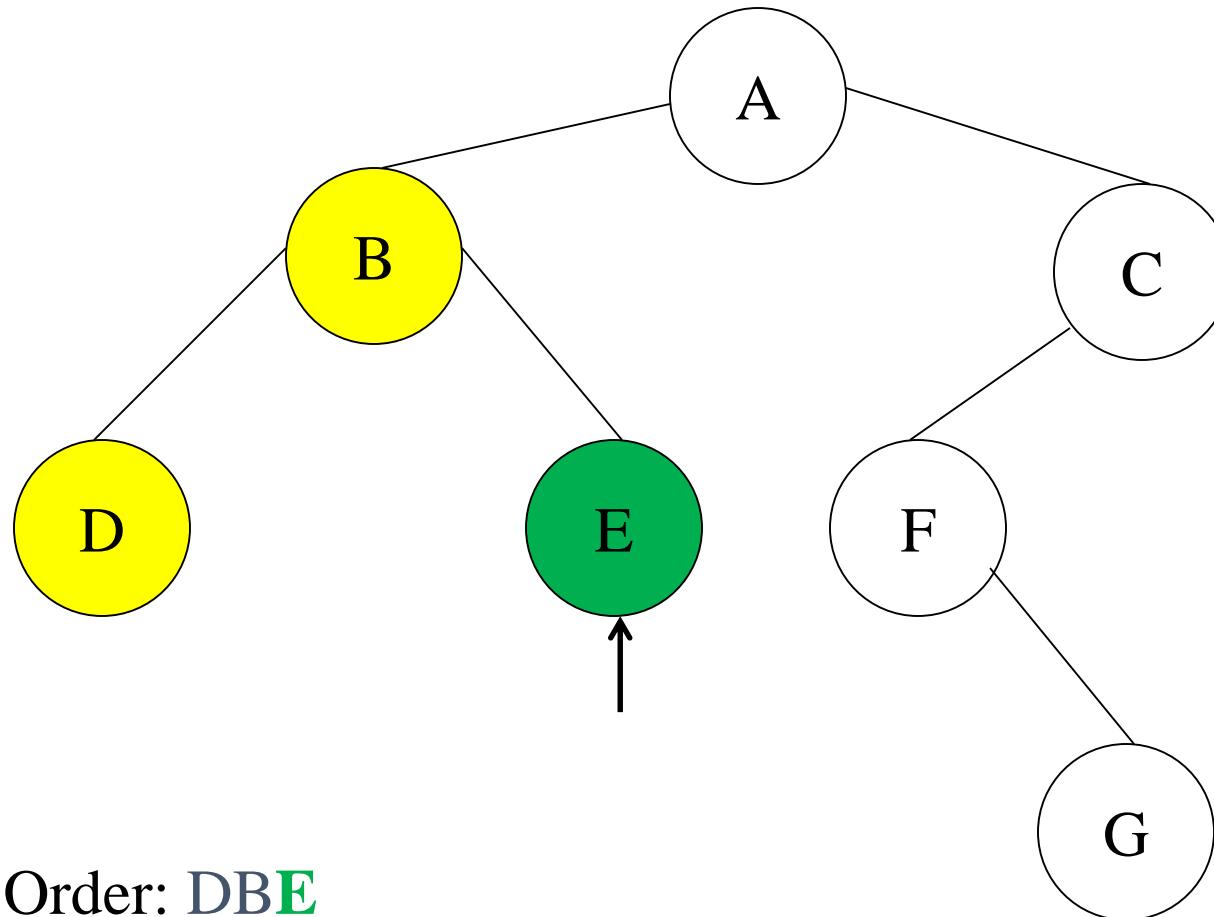
In Order: D

In Order нэвтрэлтийн жишээ

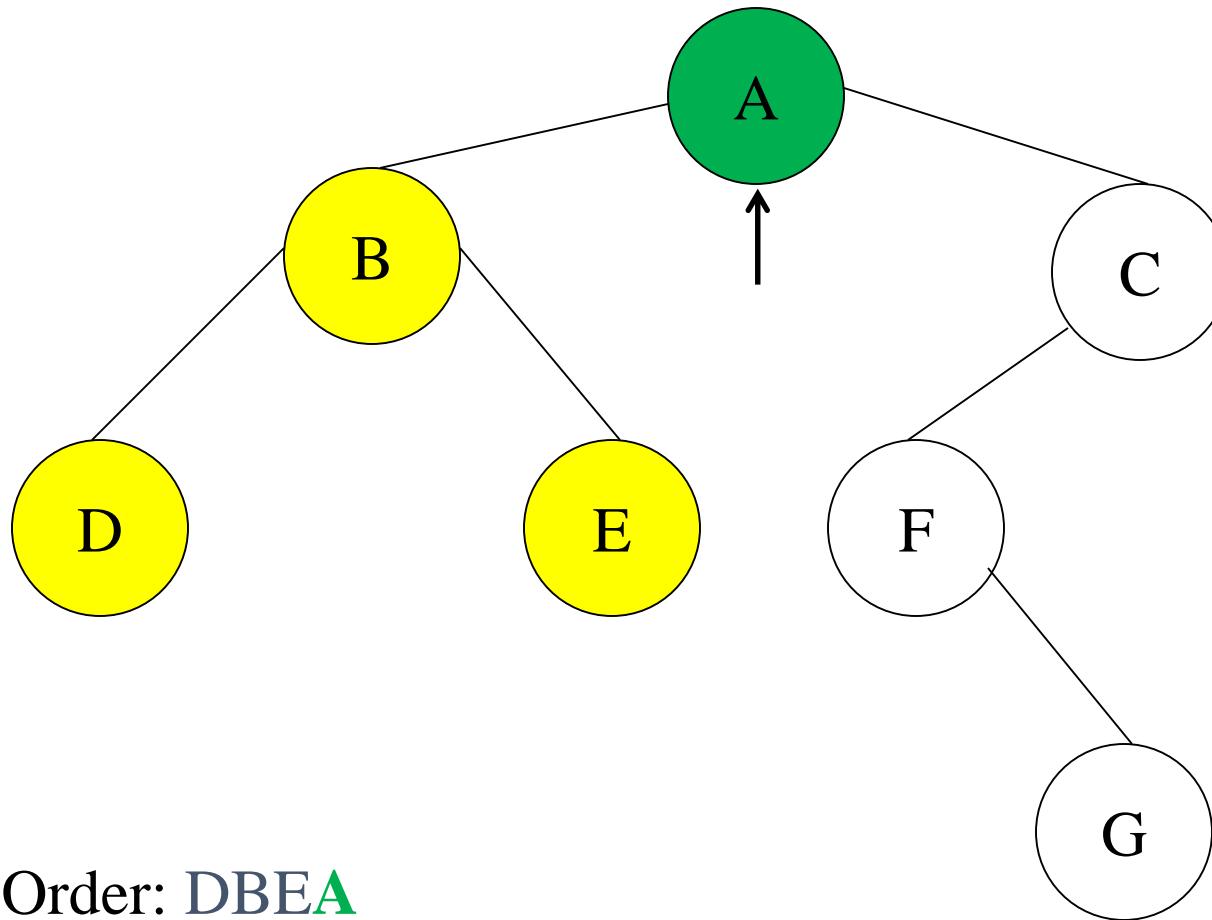


In Order: D**B**

In Order нэвтрэлтийн жишээ

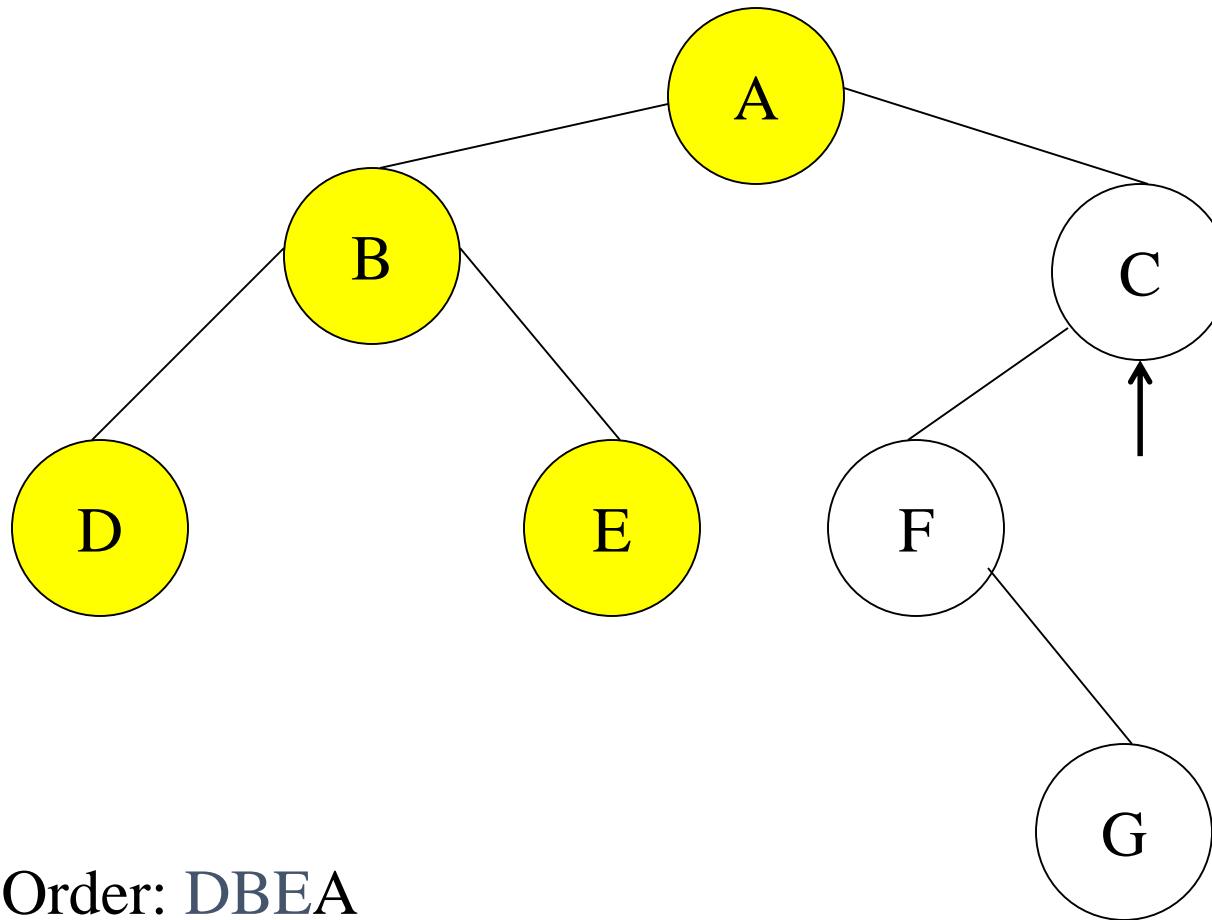


In Order нэвтрэлтийн жишээ



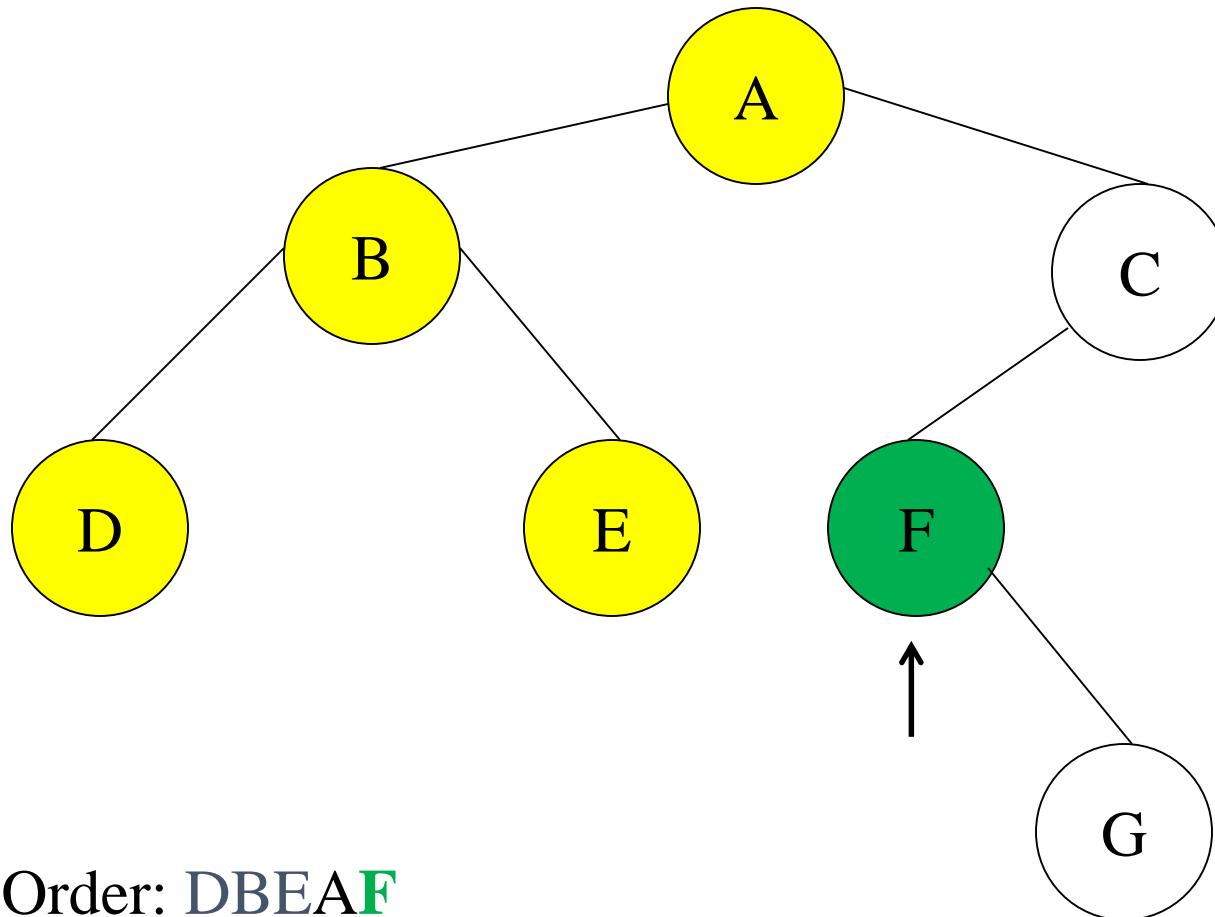
In Order: DBEAA

In Order нэвтрэлтийн жишээ



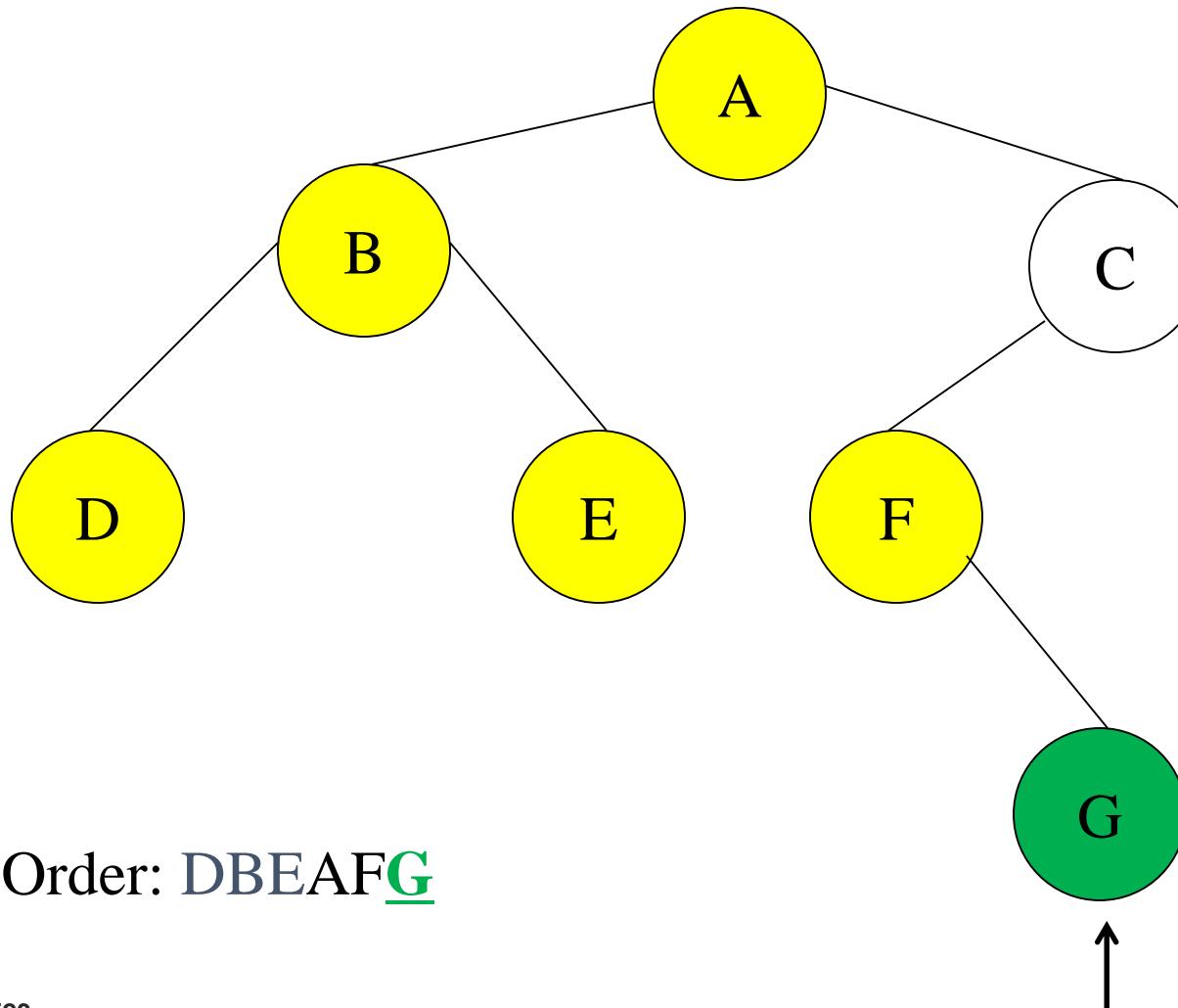
In Order: DBEA

In Order нэвтрэлтийн жишээ

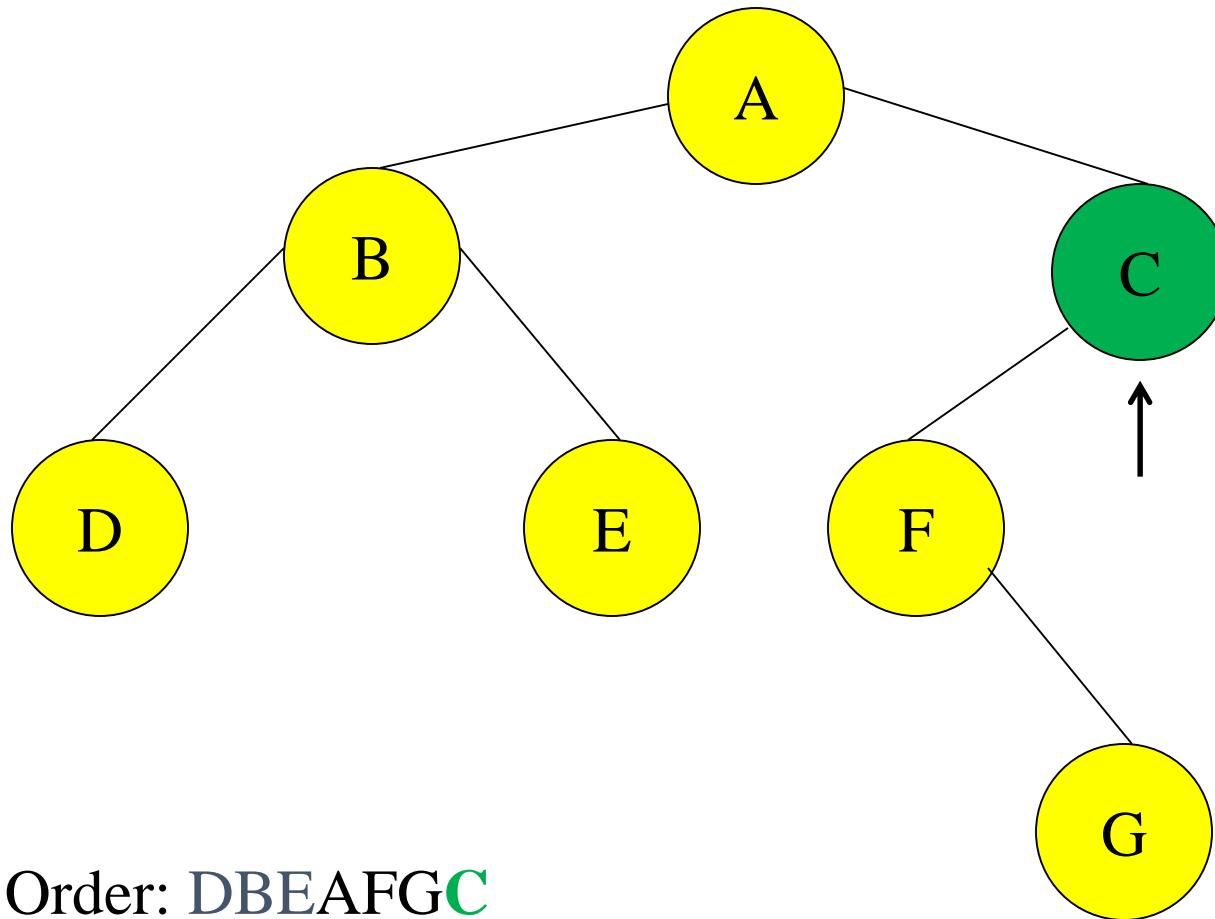


In Order: DBEAF

In Order нэвтрэлтийн жишээ

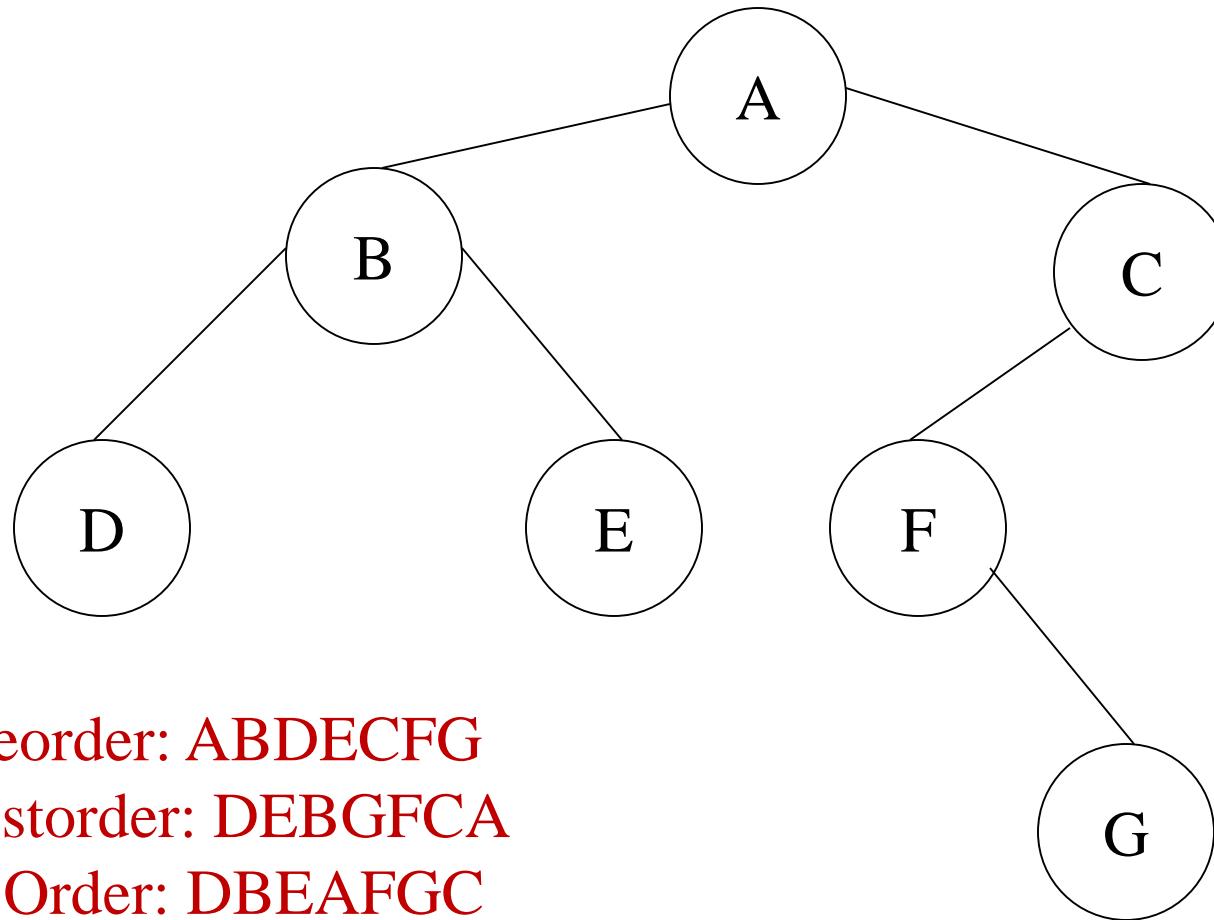


In Order нэвтрэлтийн жишээ



In Order: DBEAFGC

Модны нэвтрэлт



Preorder: ABDEC_FG

Postorder: DEBG_FC_A

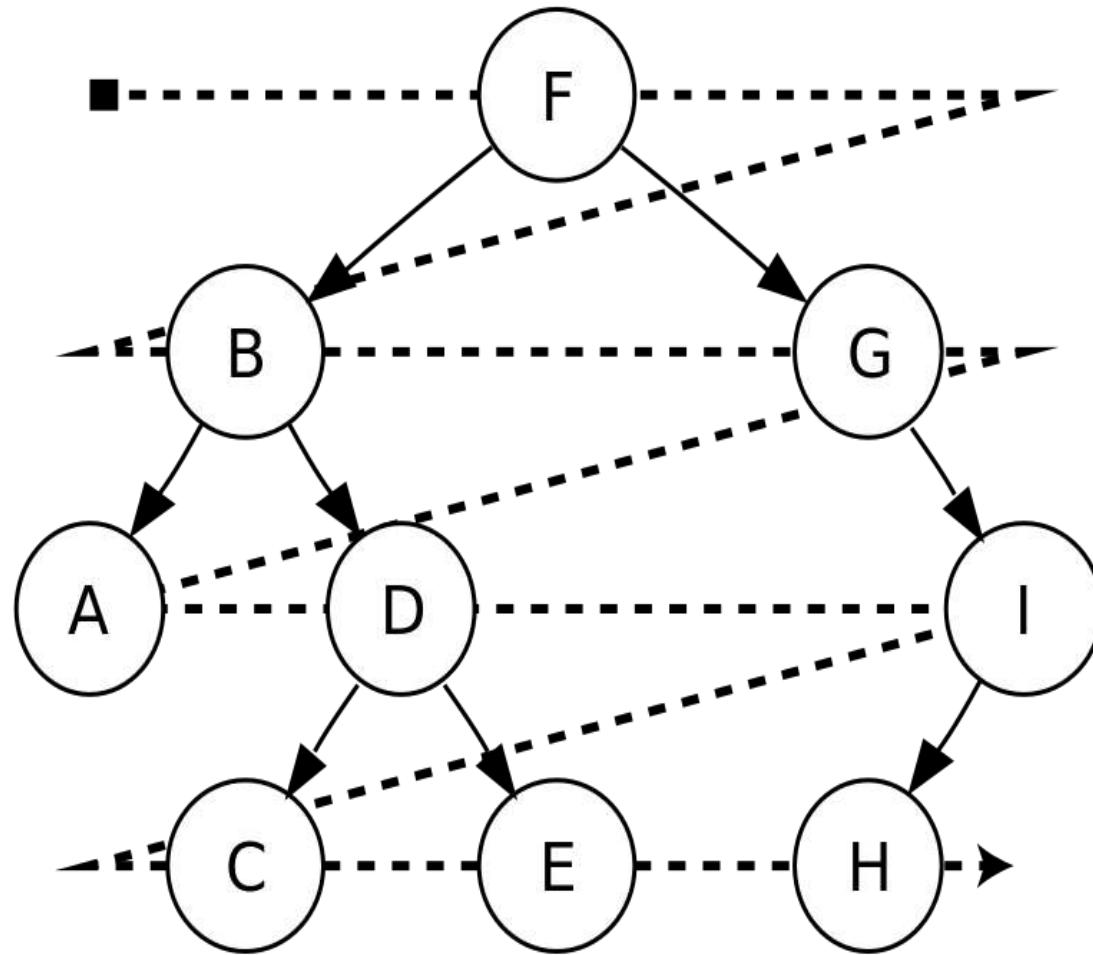
In Order: DBEA_FGC

LevelOrder нэвтрэлт

- Түвшин түвшингээр нь нэвтэрнэ.

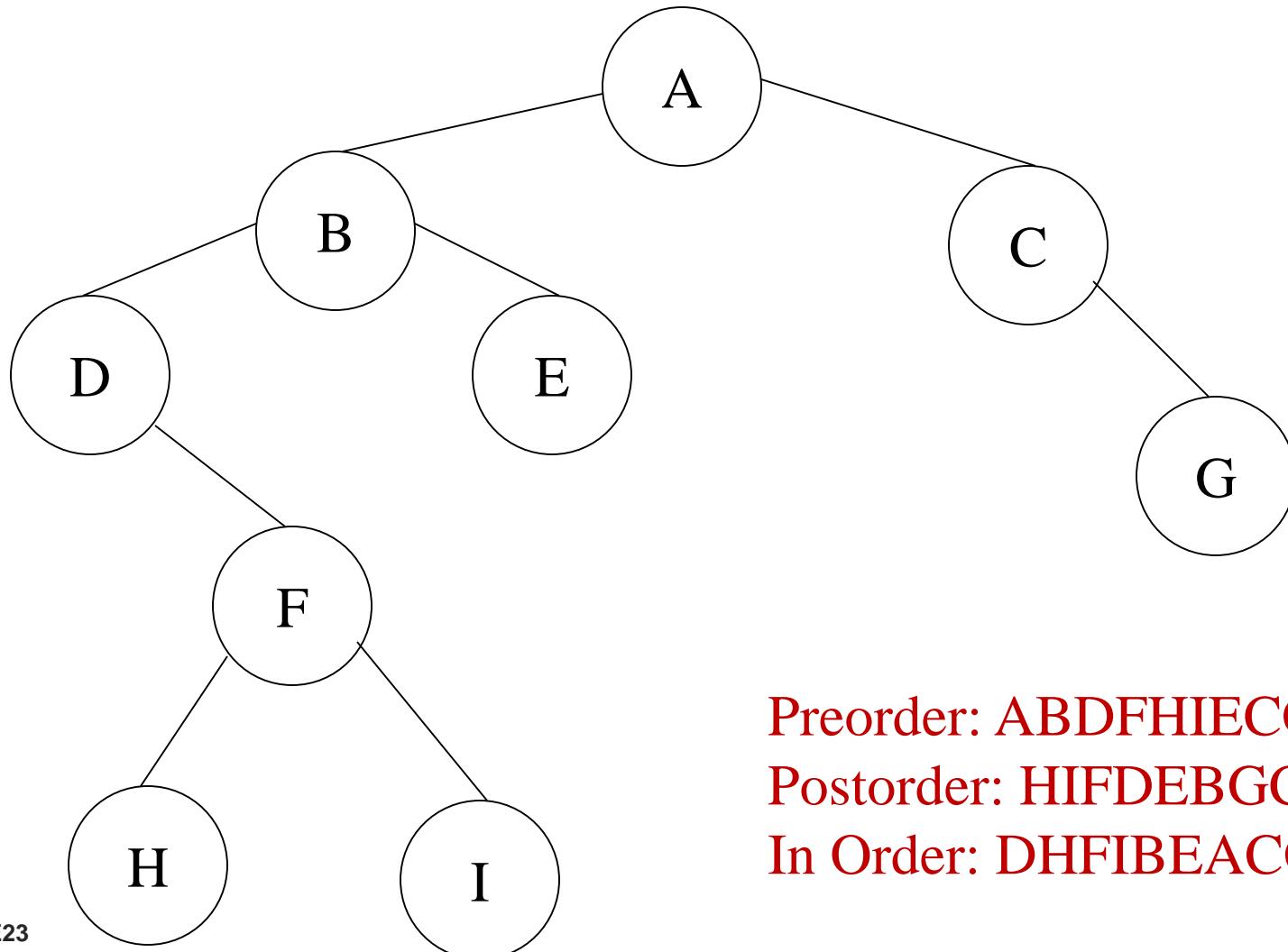
```
public static void LevelOrder(BinaryTreeNode t)
{ while (t != null)
{ // үндэсд зочлоод хүүхдүүдийг нь FIFO
// дараалалд хийнэ; зангилааг FIFO
// дарааллаас устгаж, дуудна t;
// дараалал хоосон бол устгаж null –ийг буцаана;
}
}
```

Level-order нэвтрэлтийн жишээ

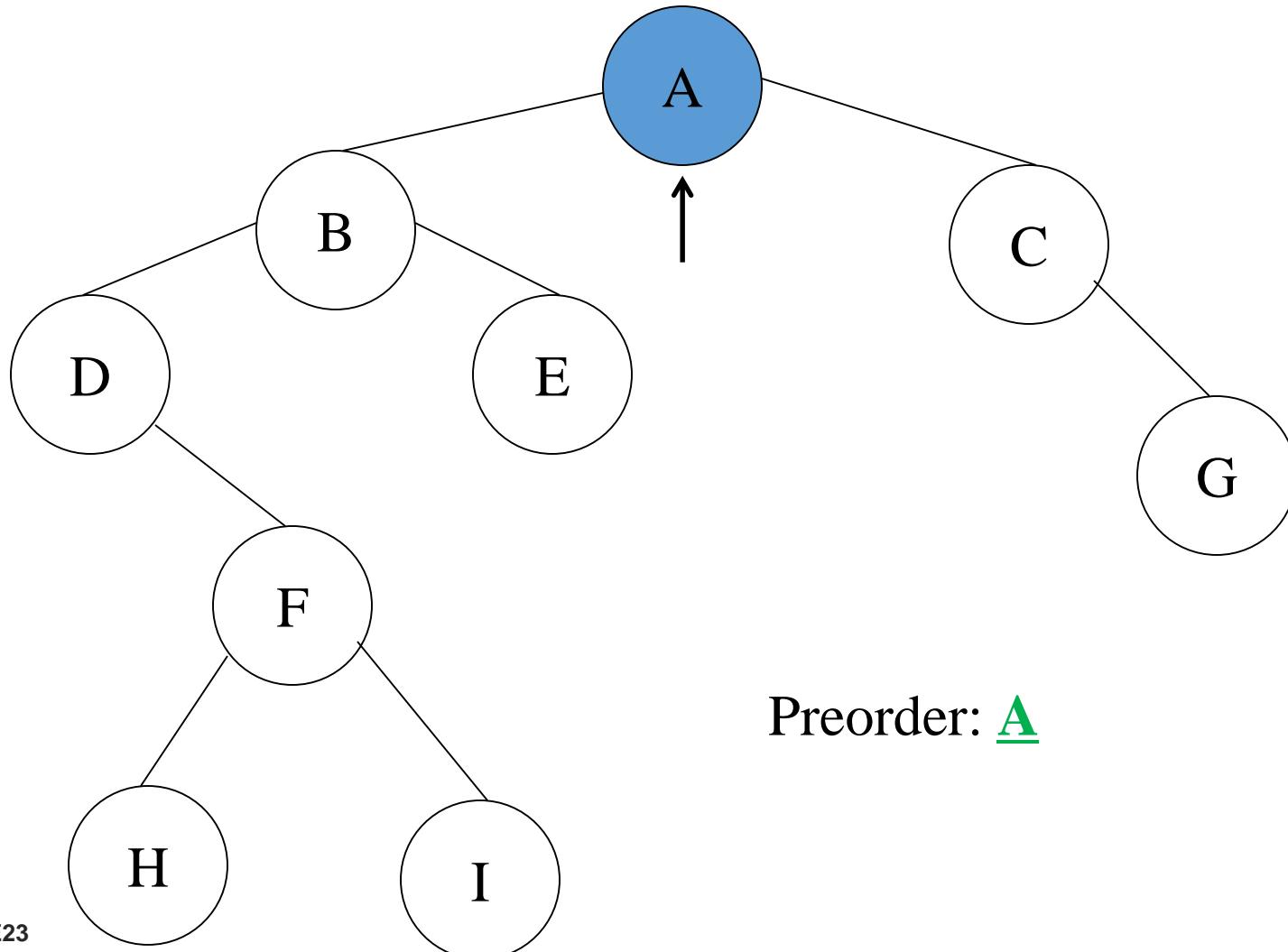


Level-order: F, B, G, A, D, I, C, E, H

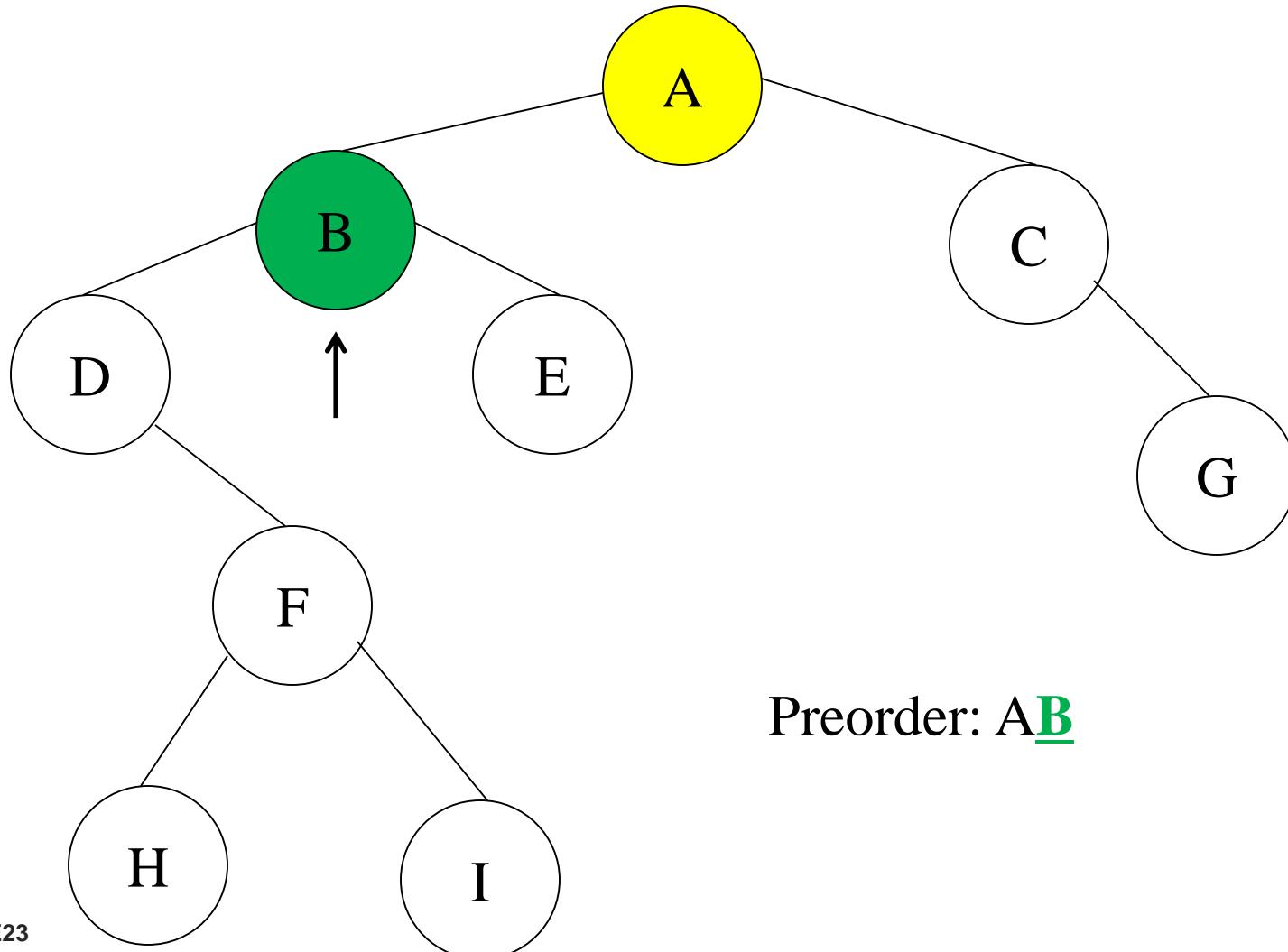
Модны нэвтрэлтийн жишээ



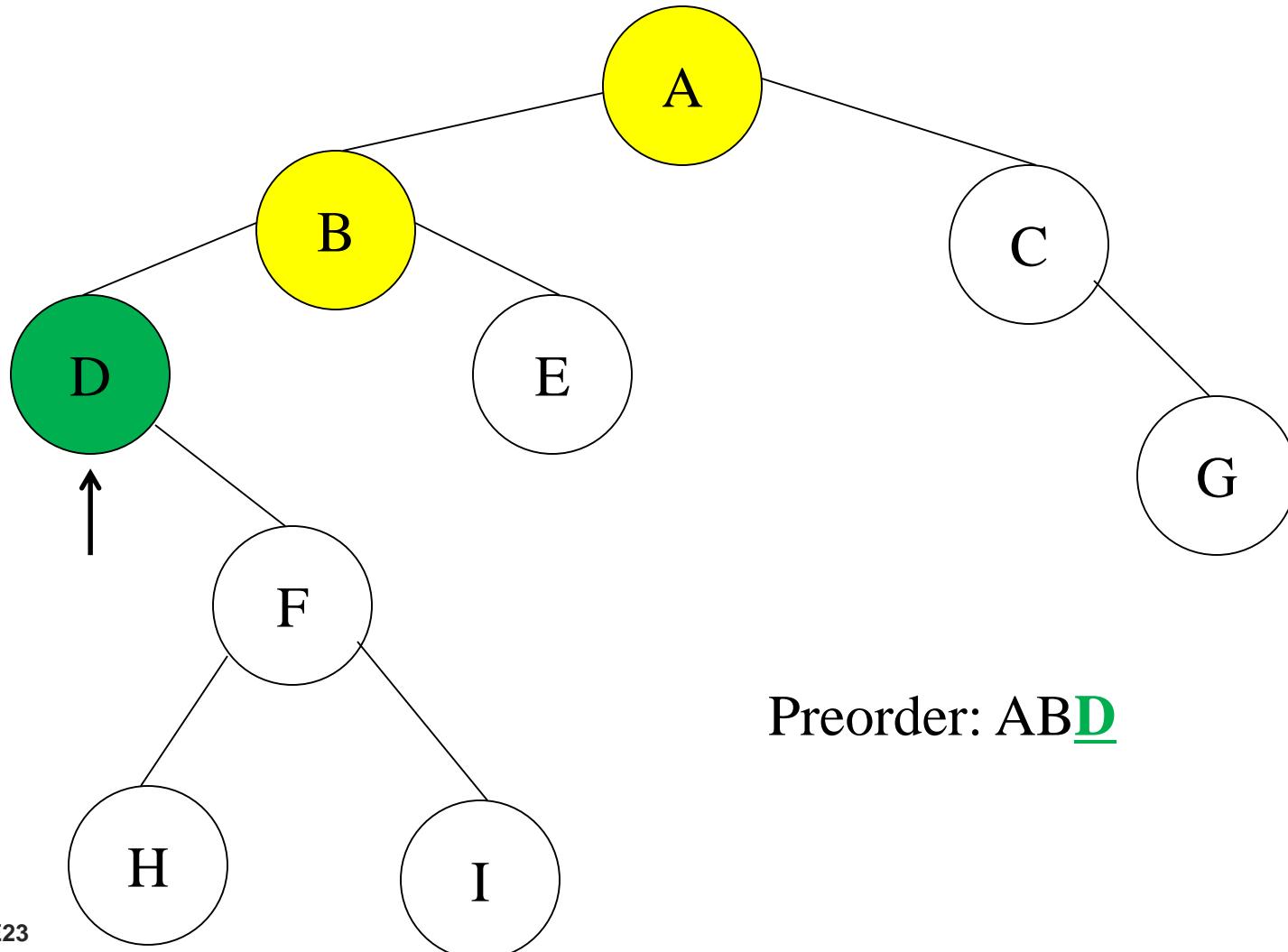
Модны нэвтрэлтийн жишээ



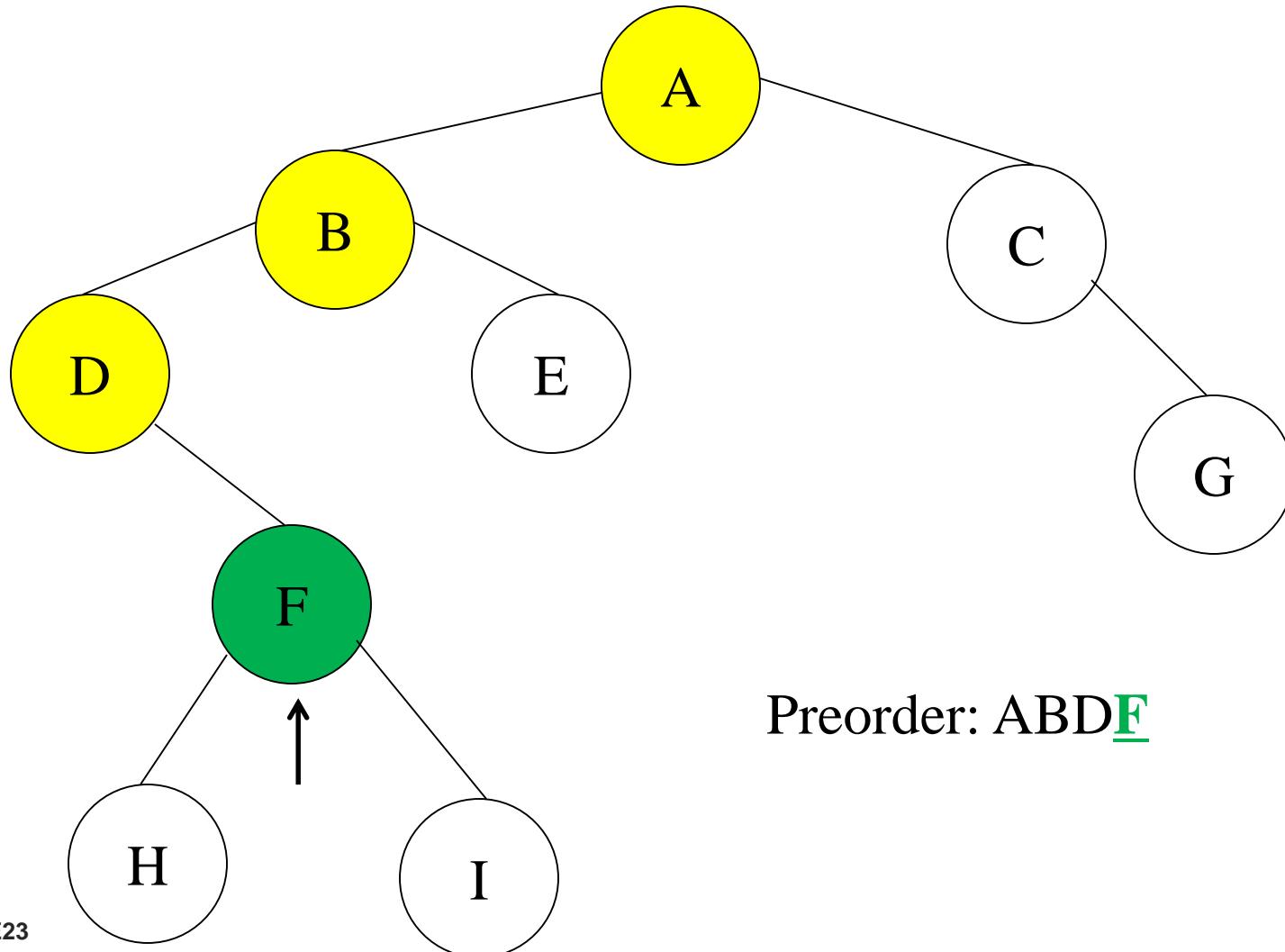
Модны нэвтрэлтийн жишээ



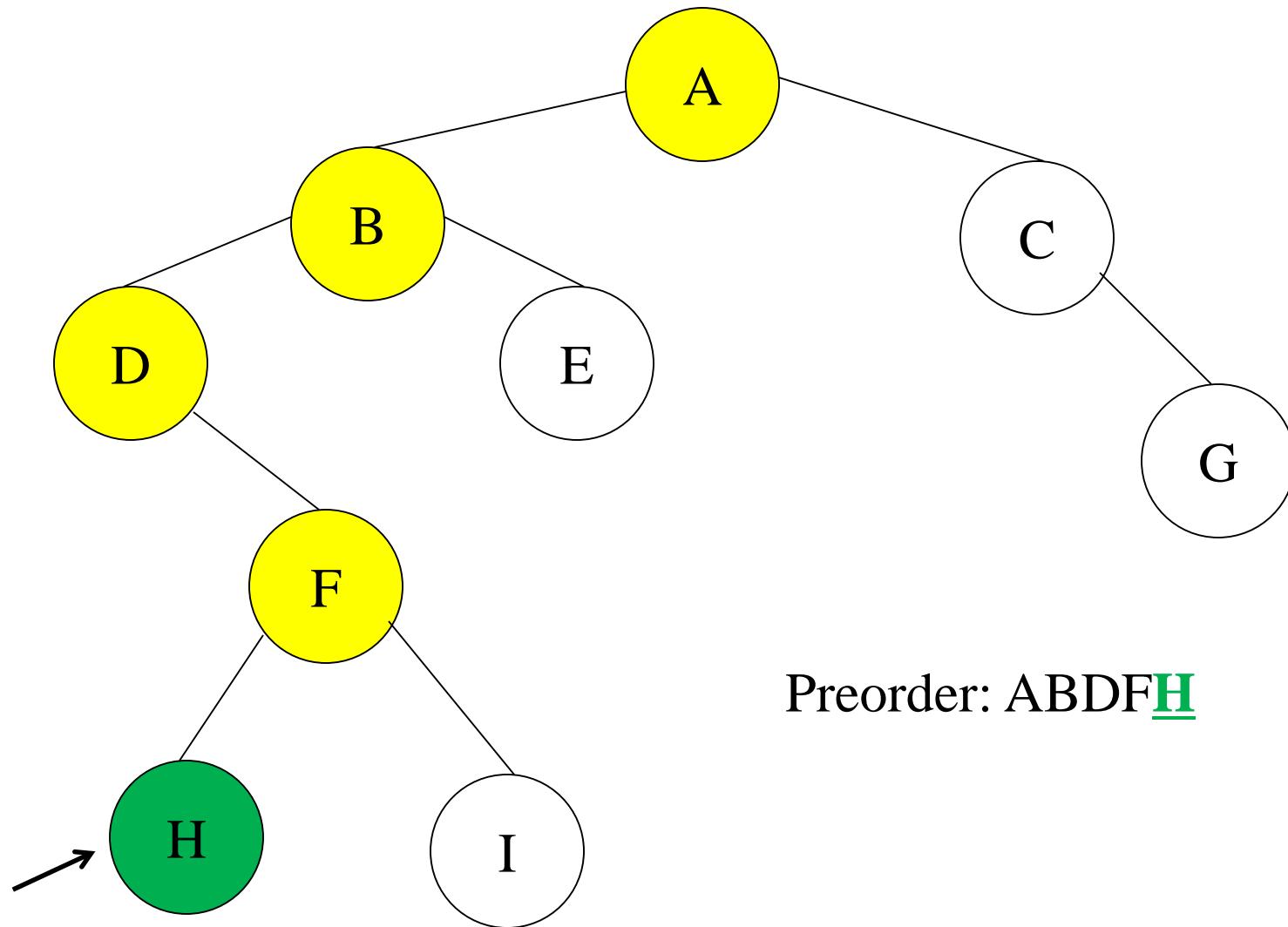
Модны нэвтрэлтийн жишээ



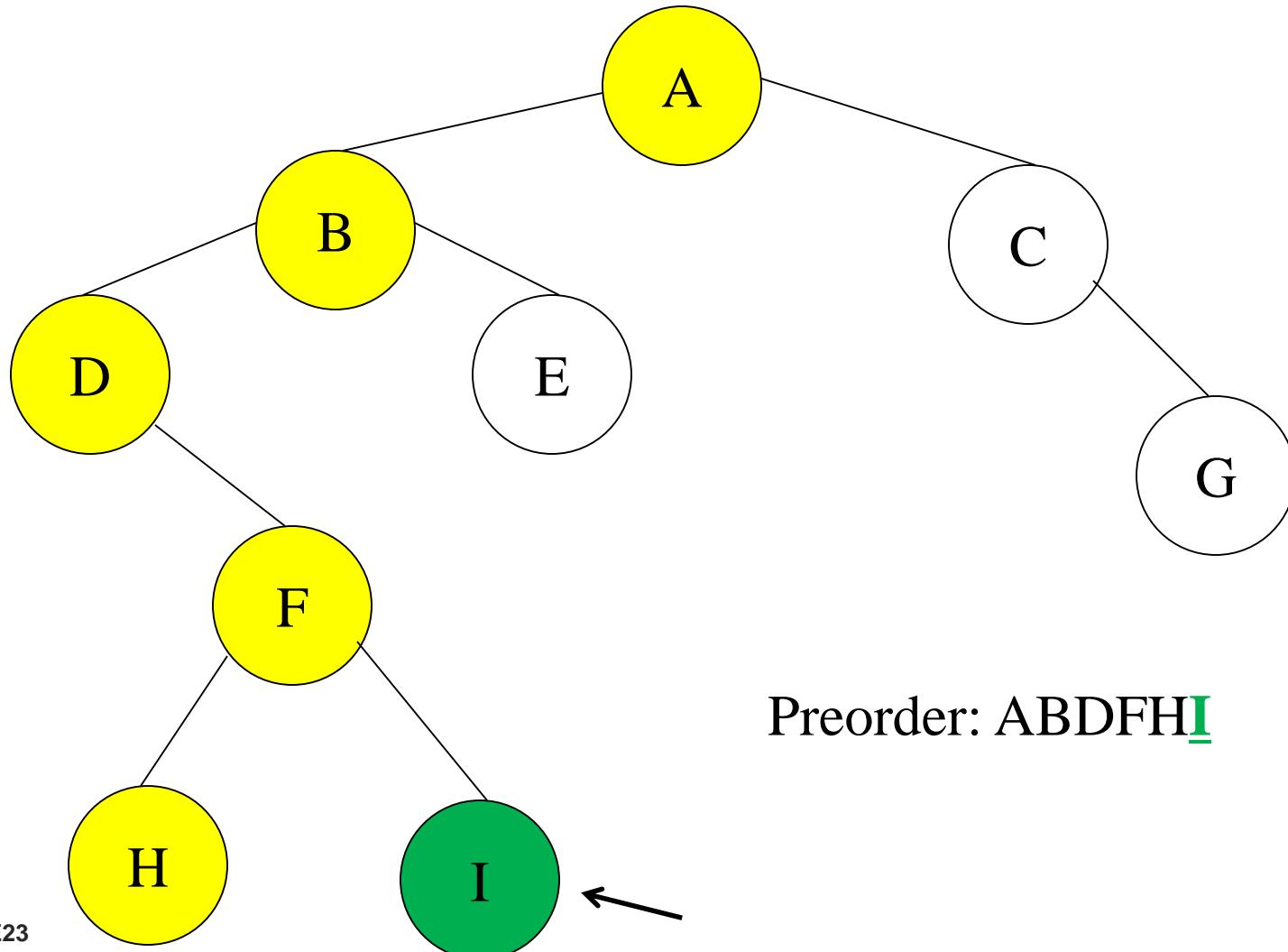
Модны нэвтрэлтийн жишээ



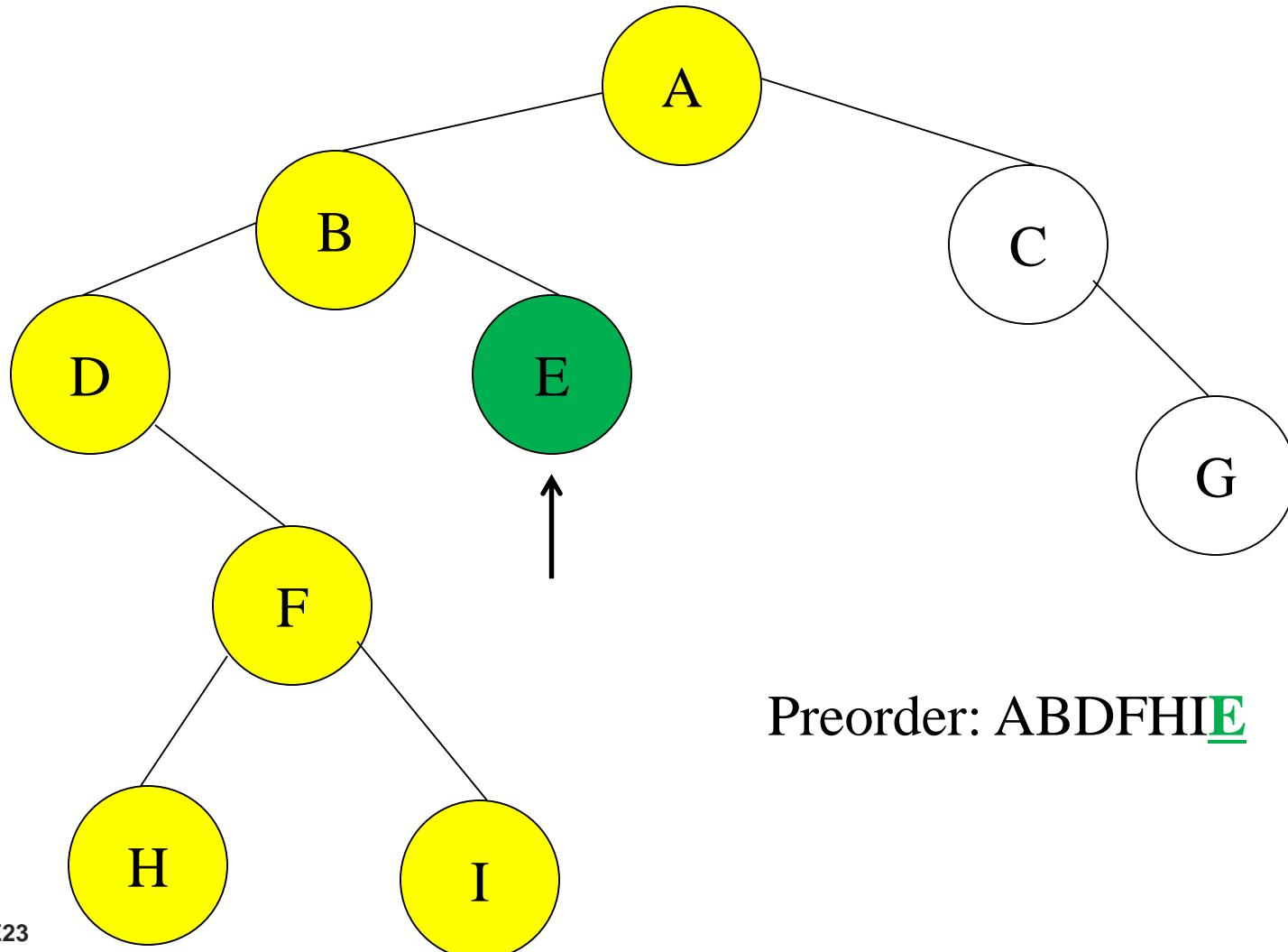
Модны нэвтрэлтийн жишээ



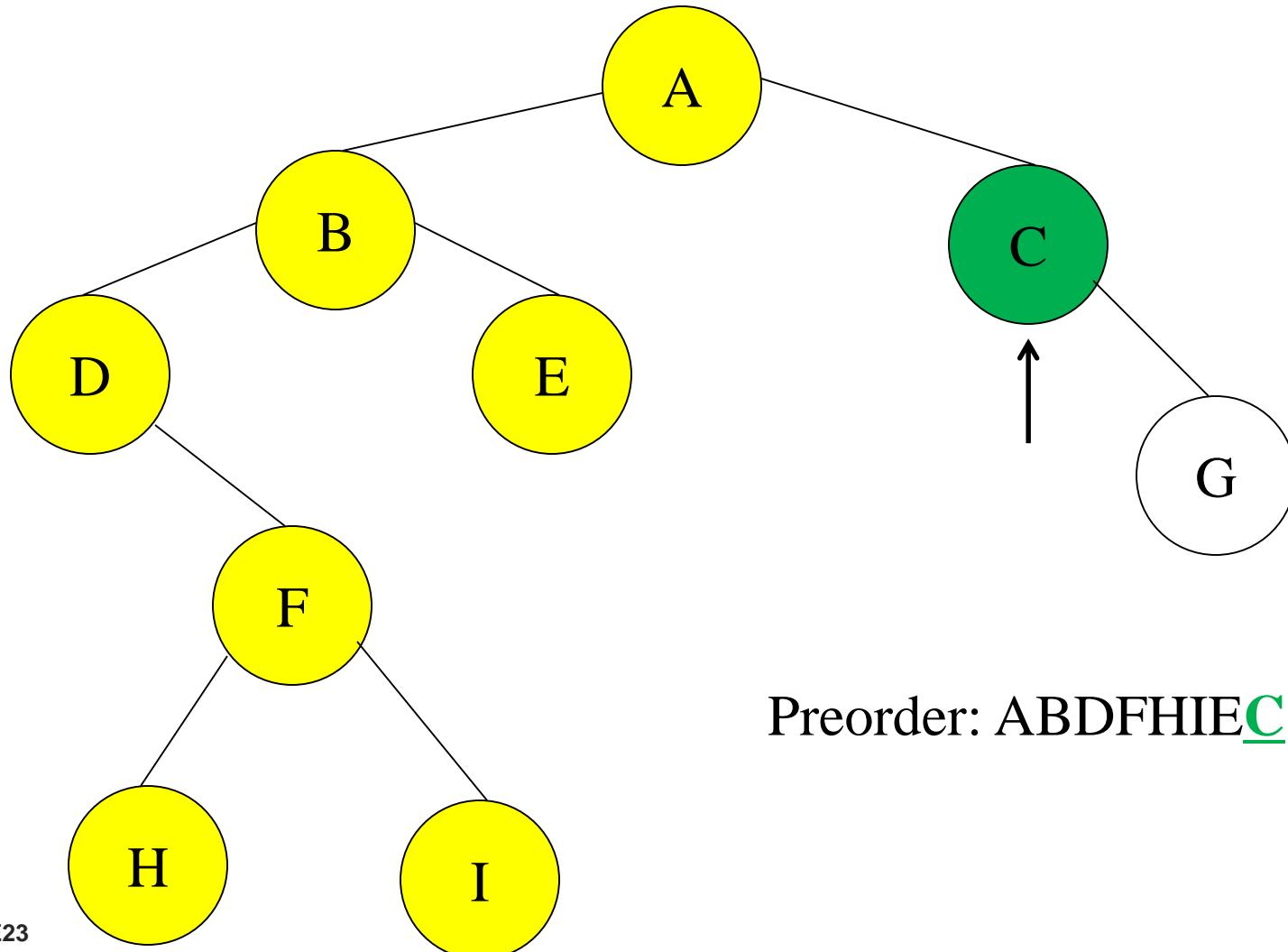
Модны нэвтрэлтийн жишээ



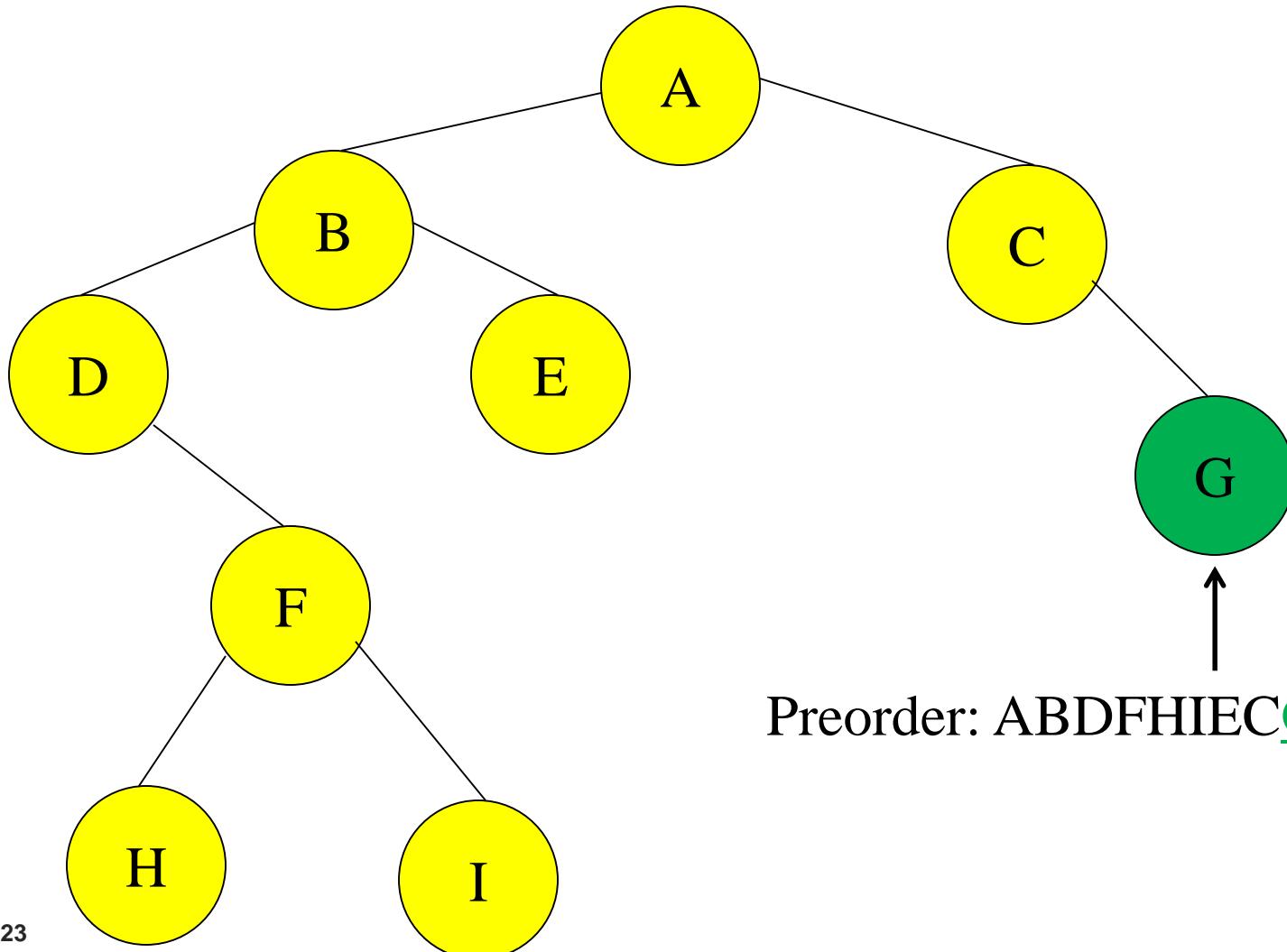
Модны нэвтрэлтийн жишээ



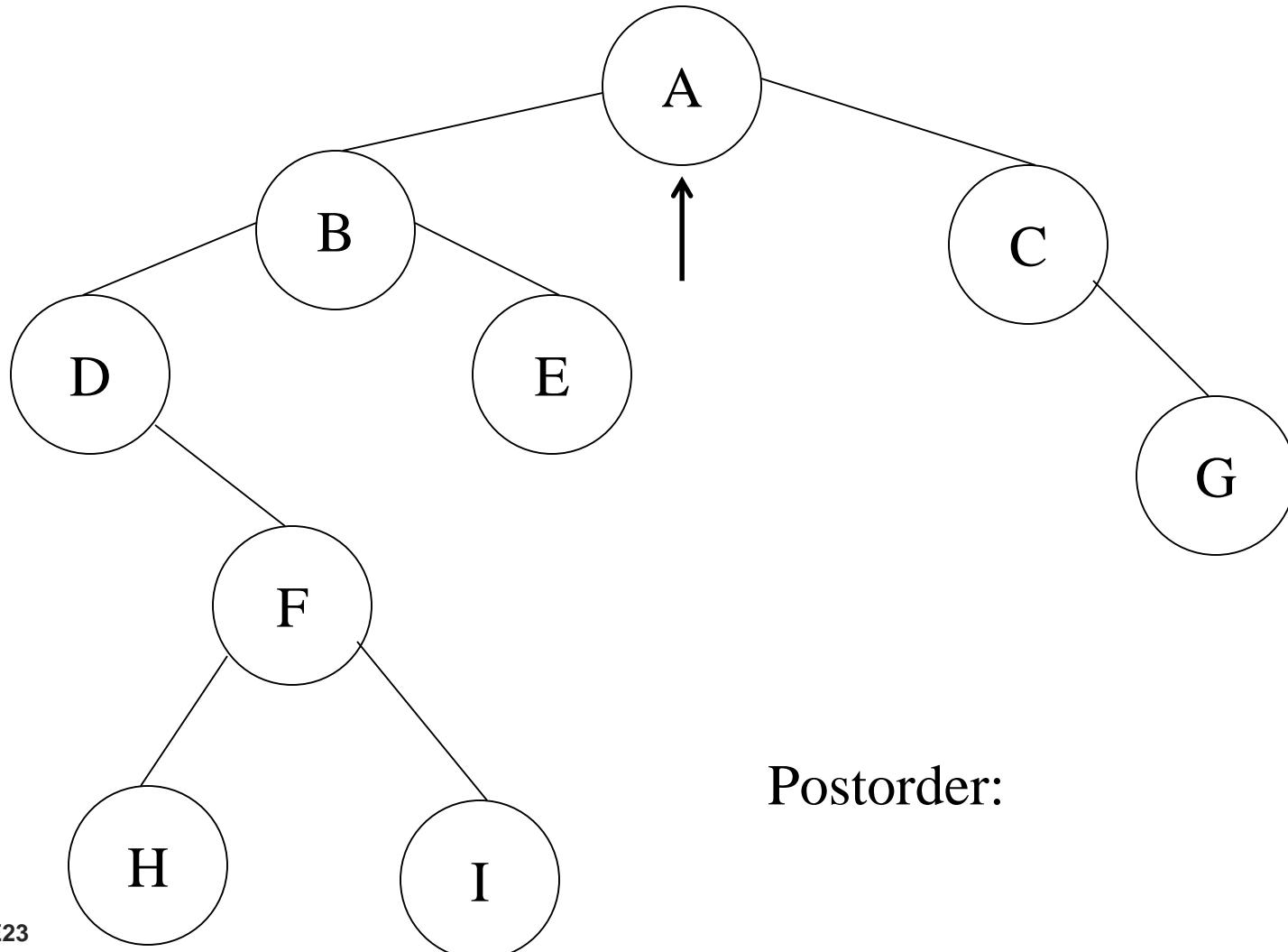
Модны нэвтрэлтийн жишээ



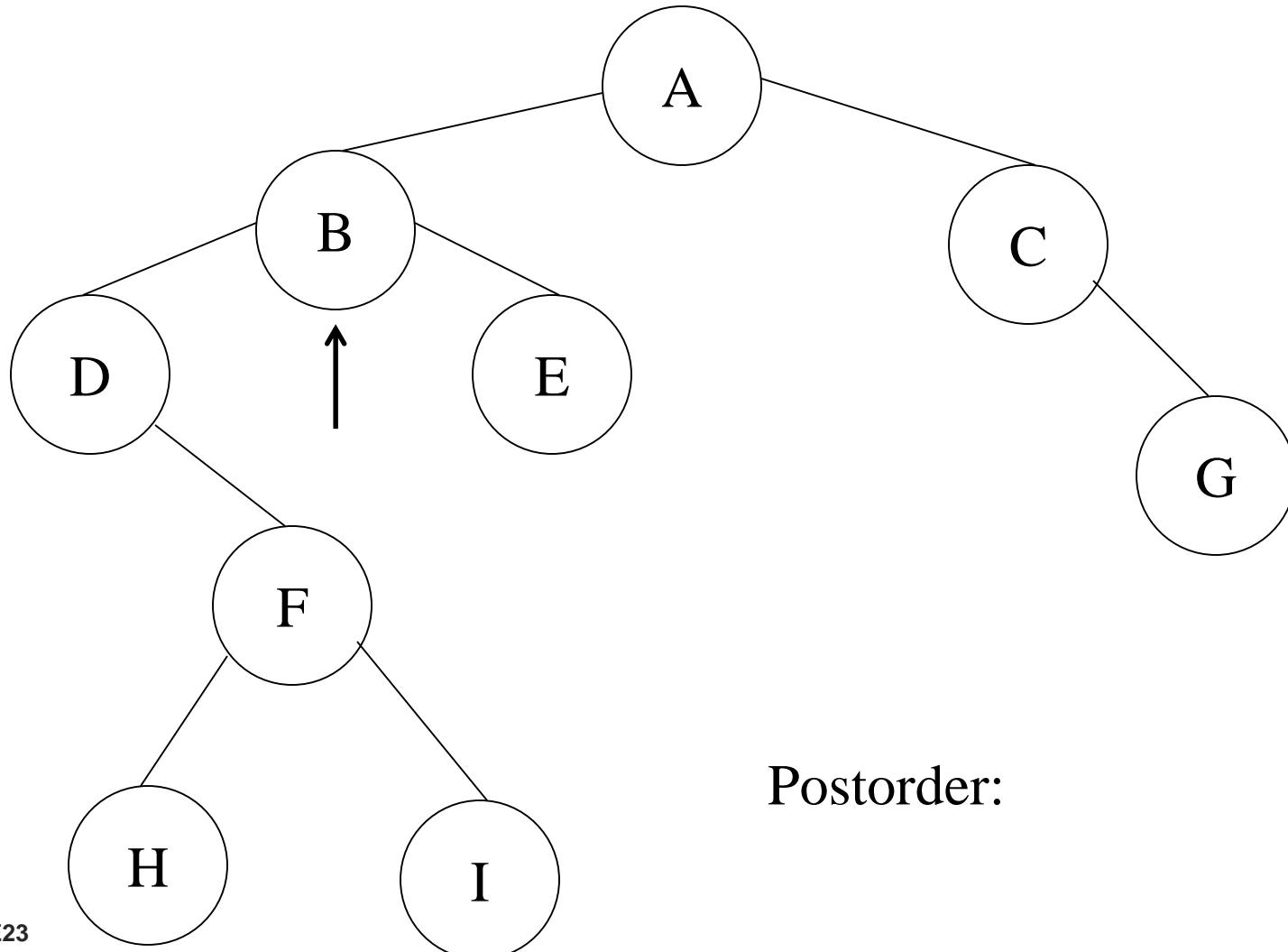
Модны нэвтрэлтийн жишээ



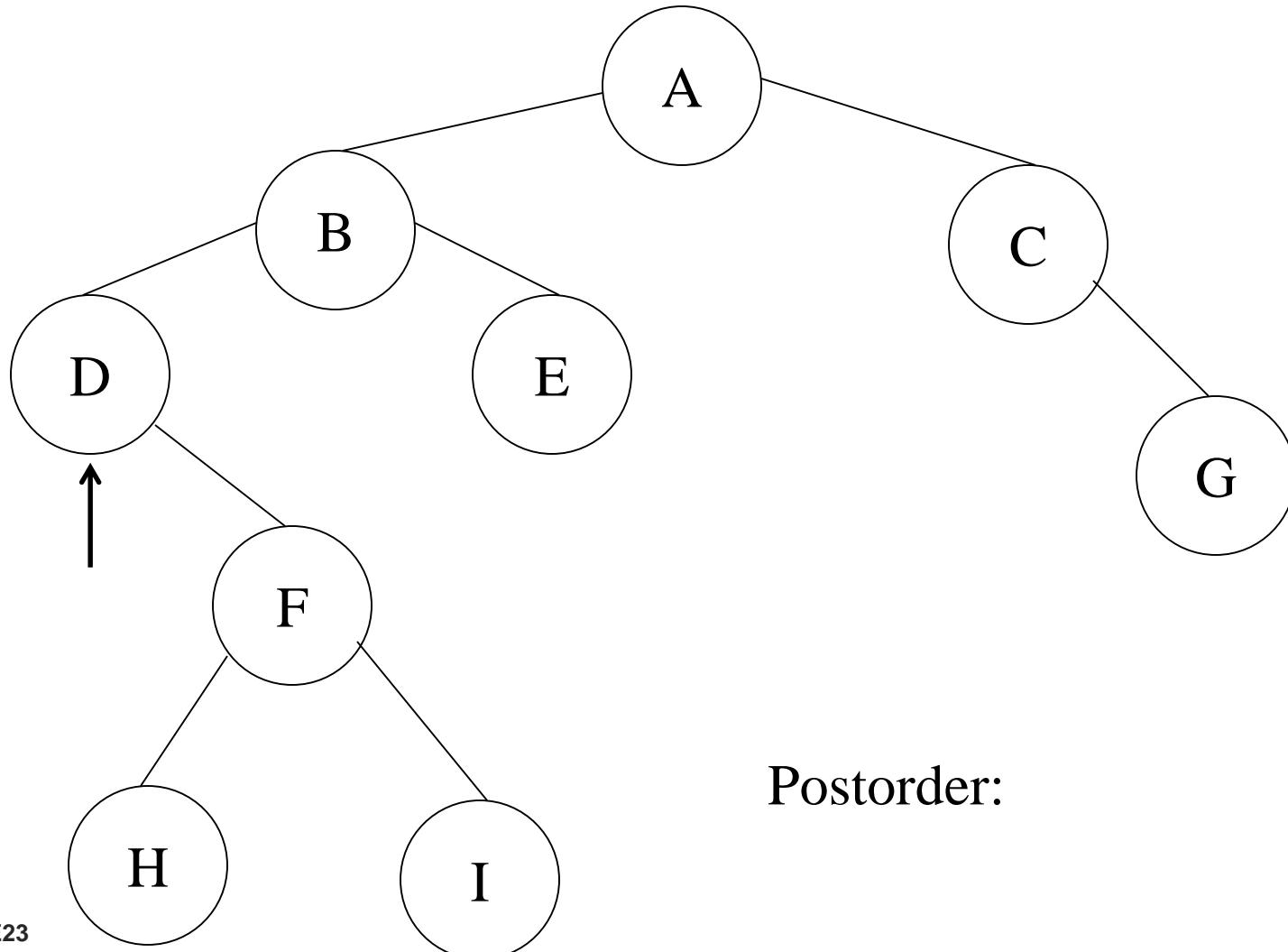
Модны нэвтрэлтийн жишээ



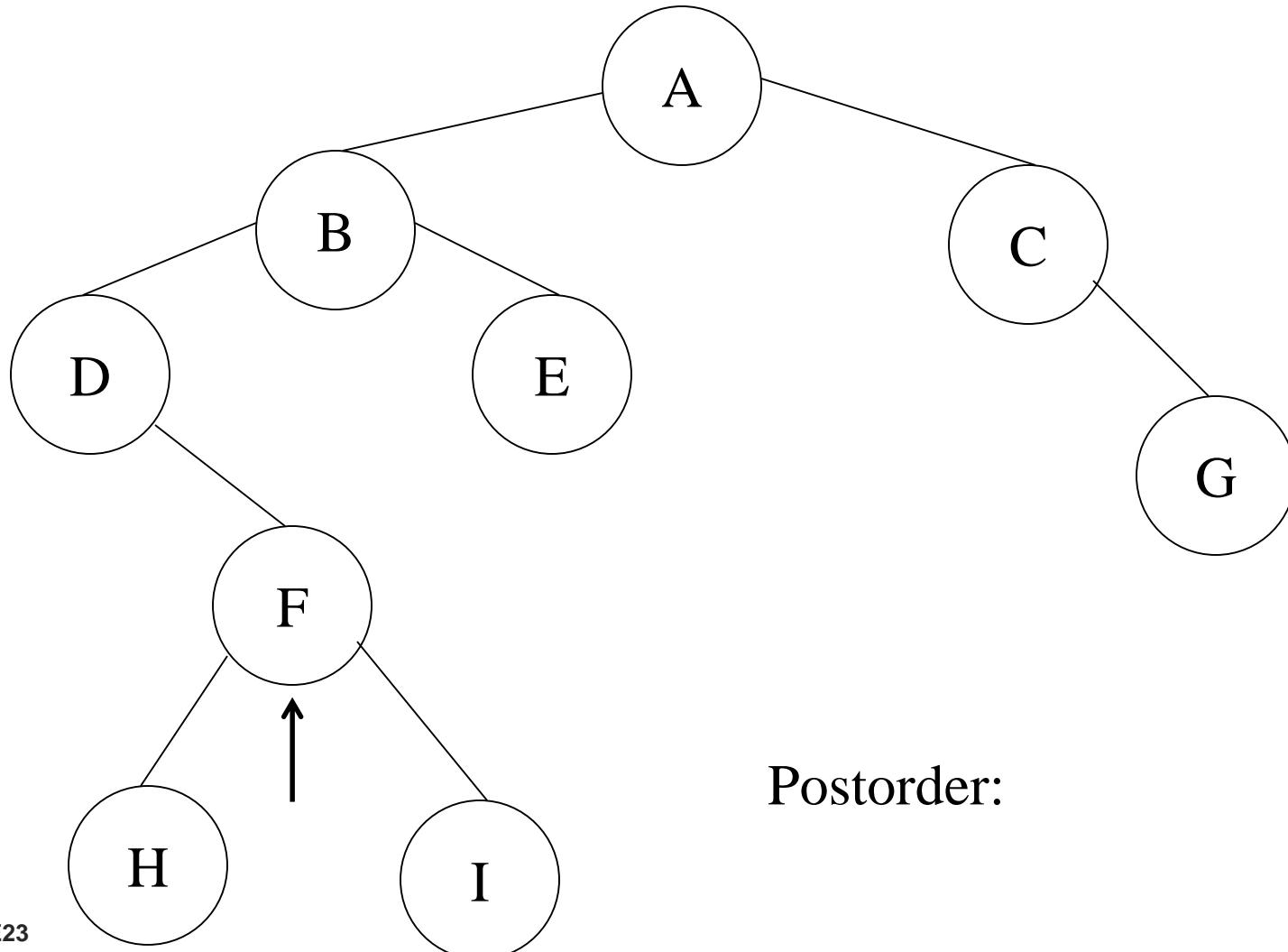
Модны нэвтрэлтийн жишээ



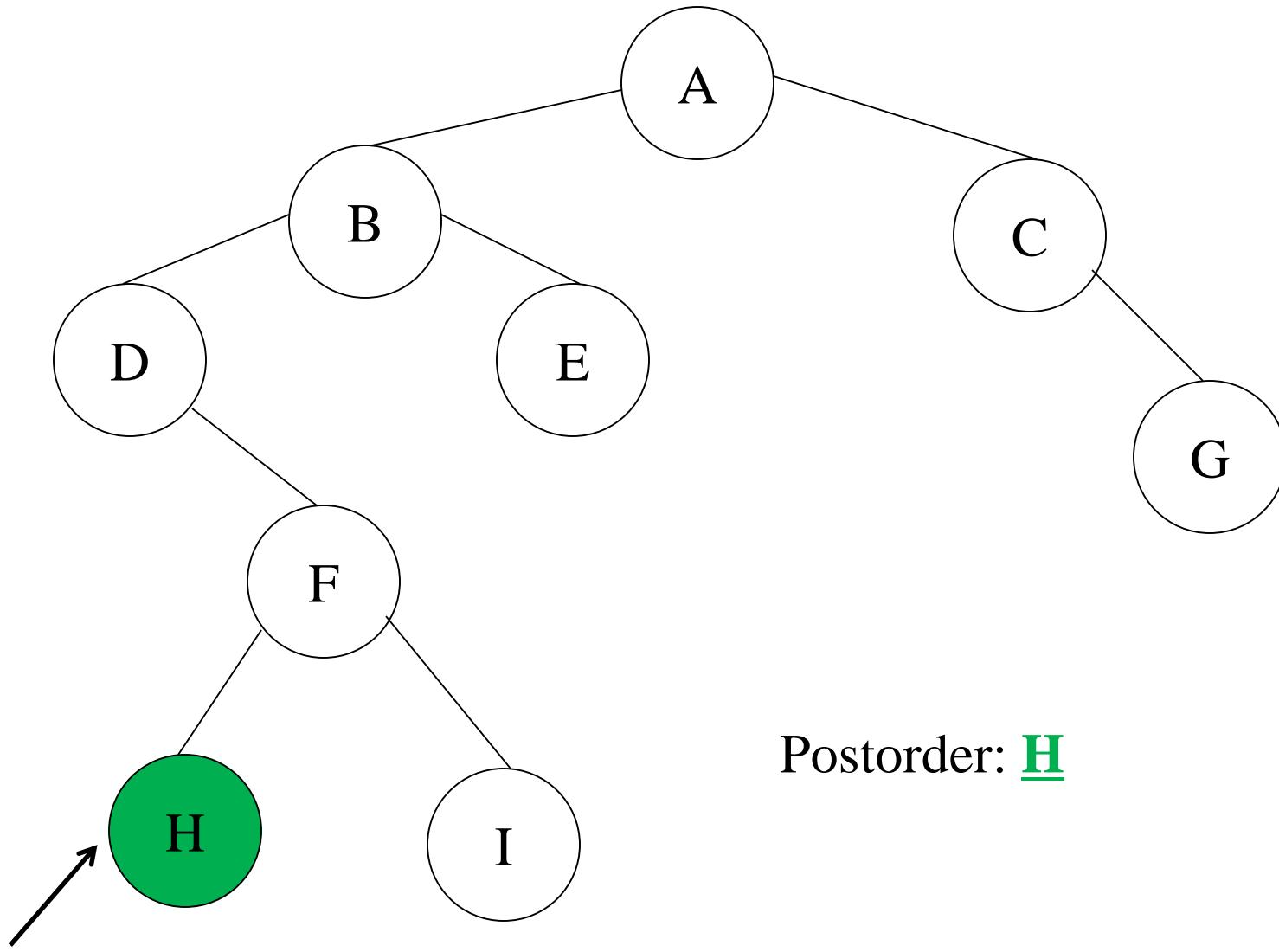
Модны нэвтрэлтийн жишээ



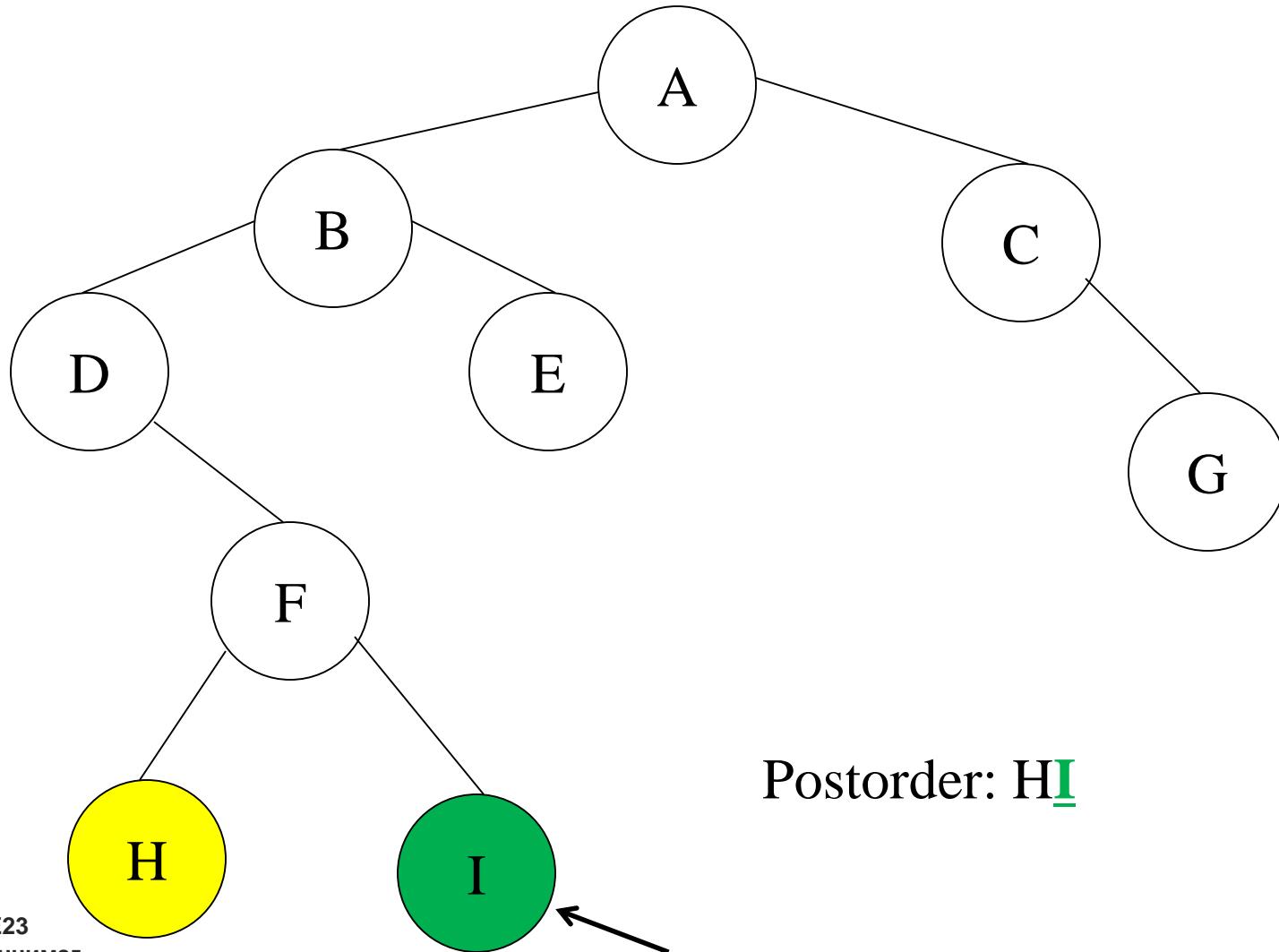
Модны нэвтрэлтийн жишээ



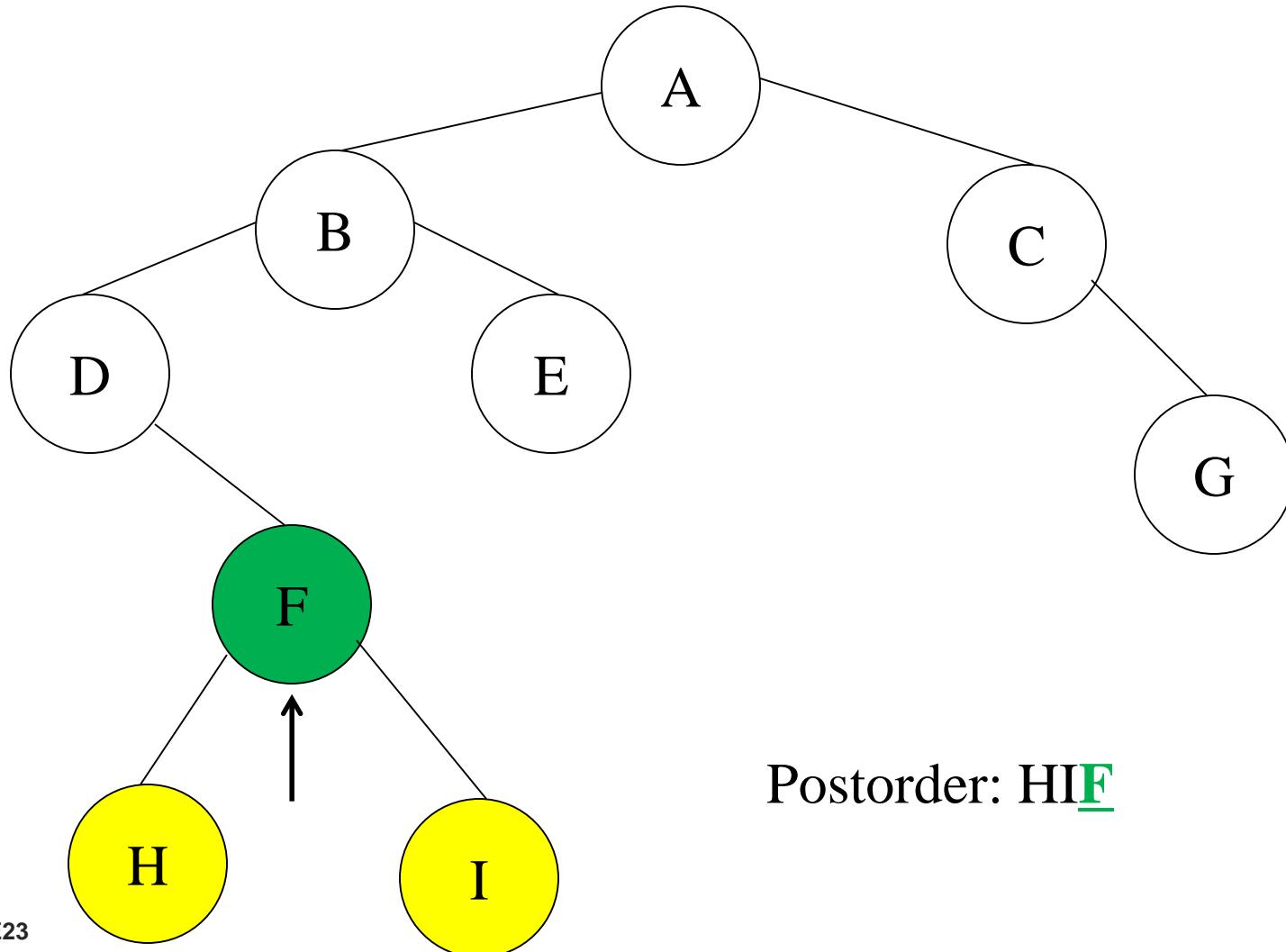
Модны нэвтрэлтийн жишээ



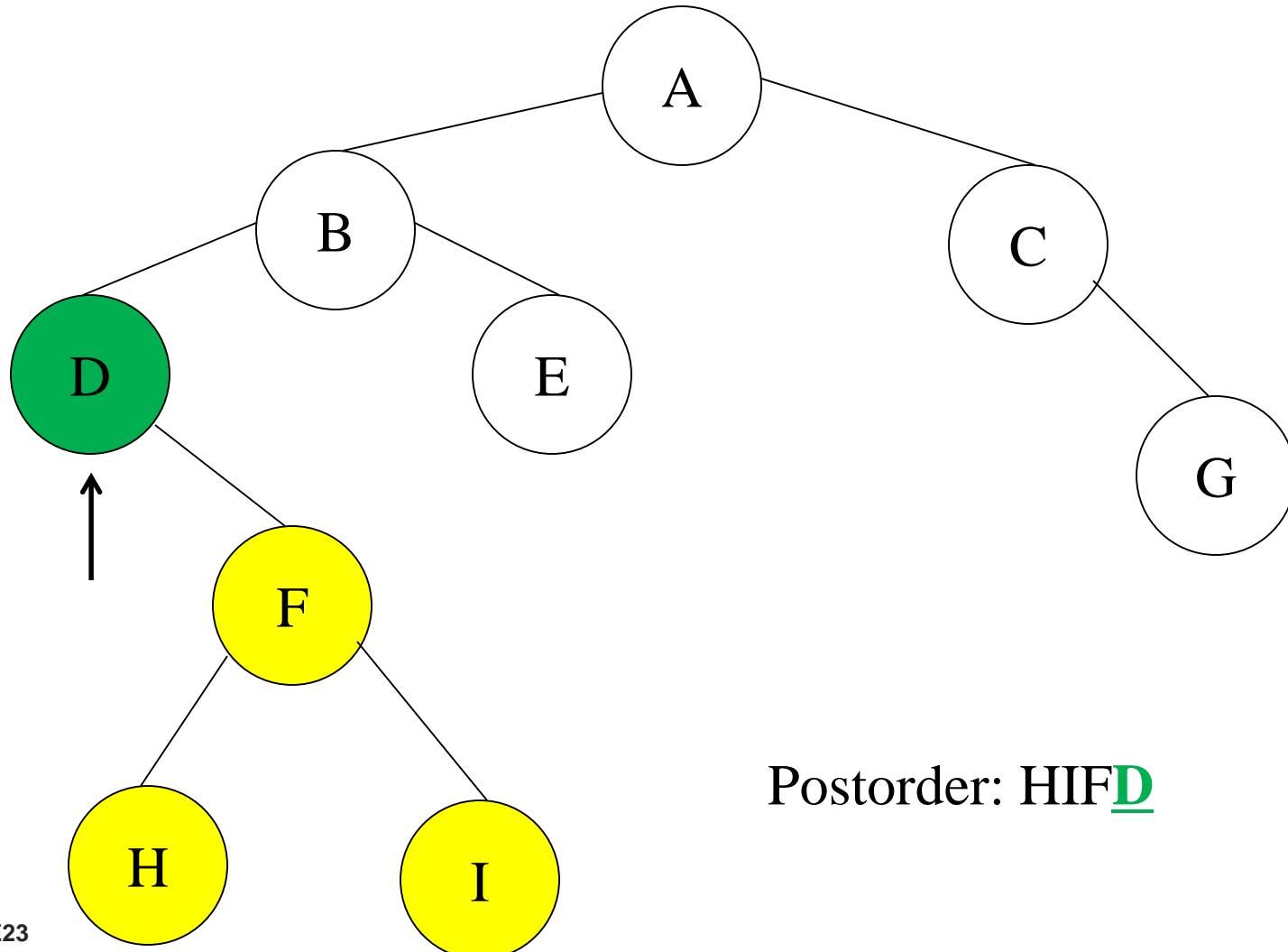
Модны нэвтрэлтийн жишээ



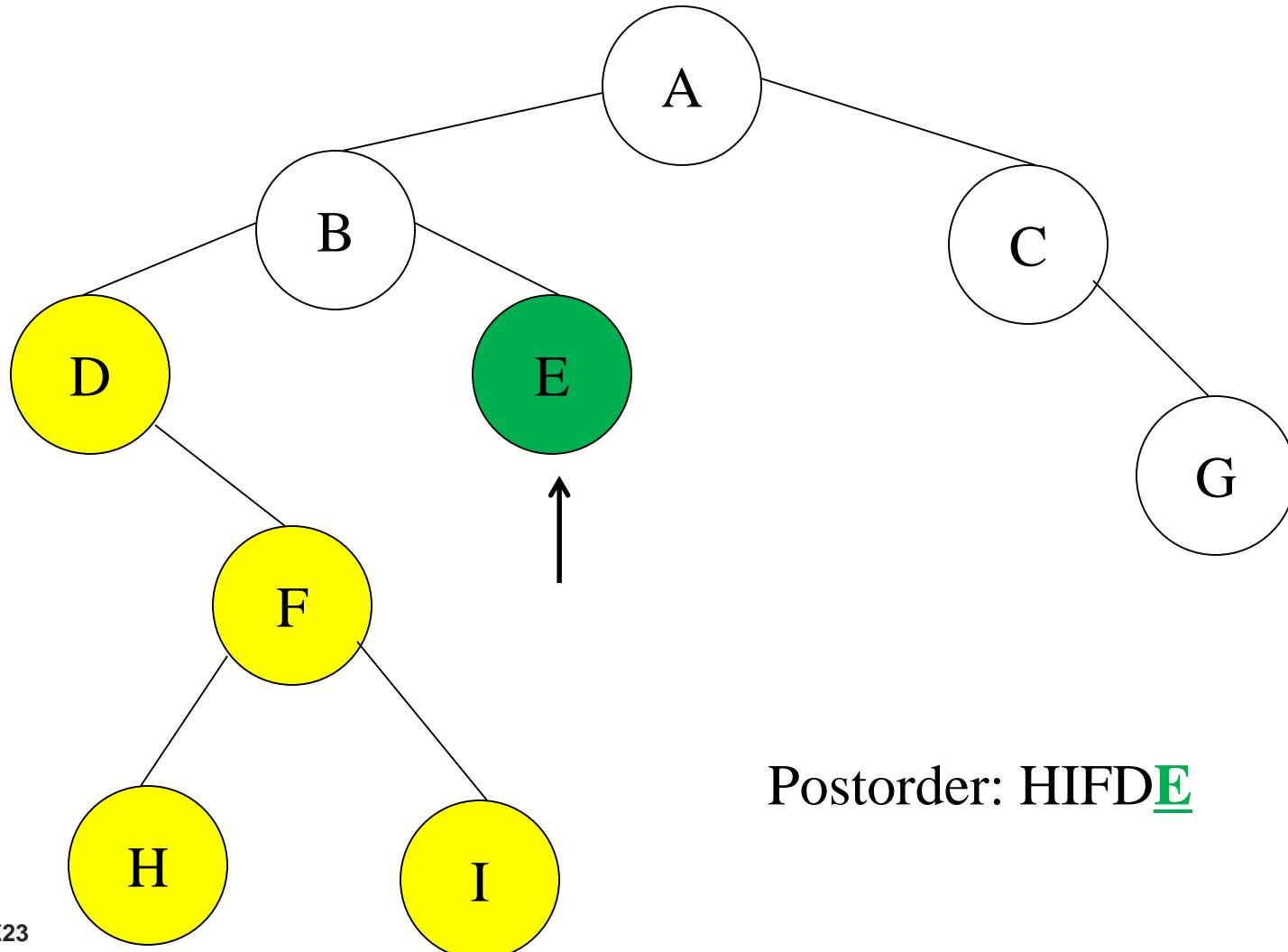
Модны нэвтрэлтийн жишээ



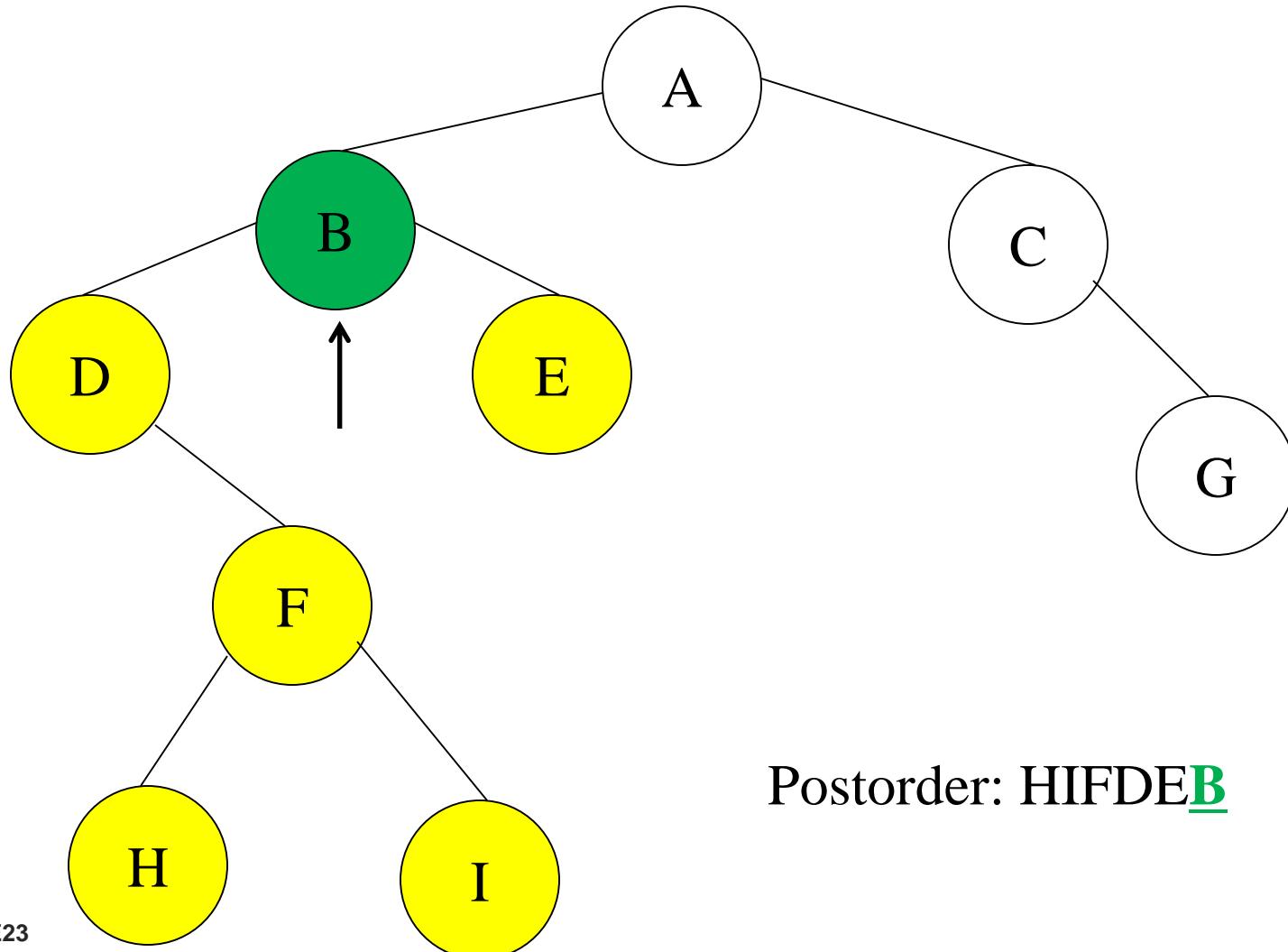
Модны нэвтрэлтийн жишээ



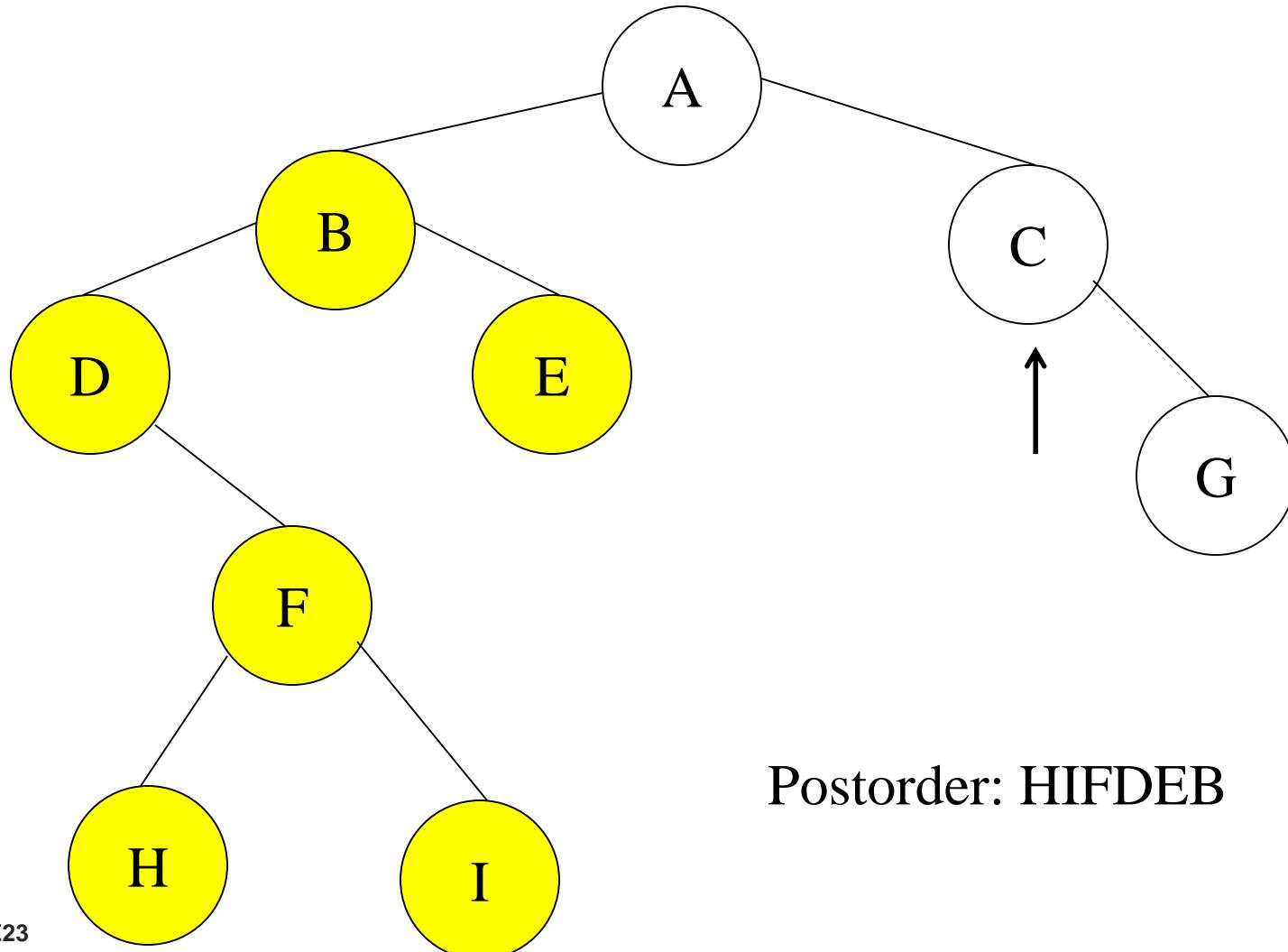
Модны нэвтрэлтийн жишээ



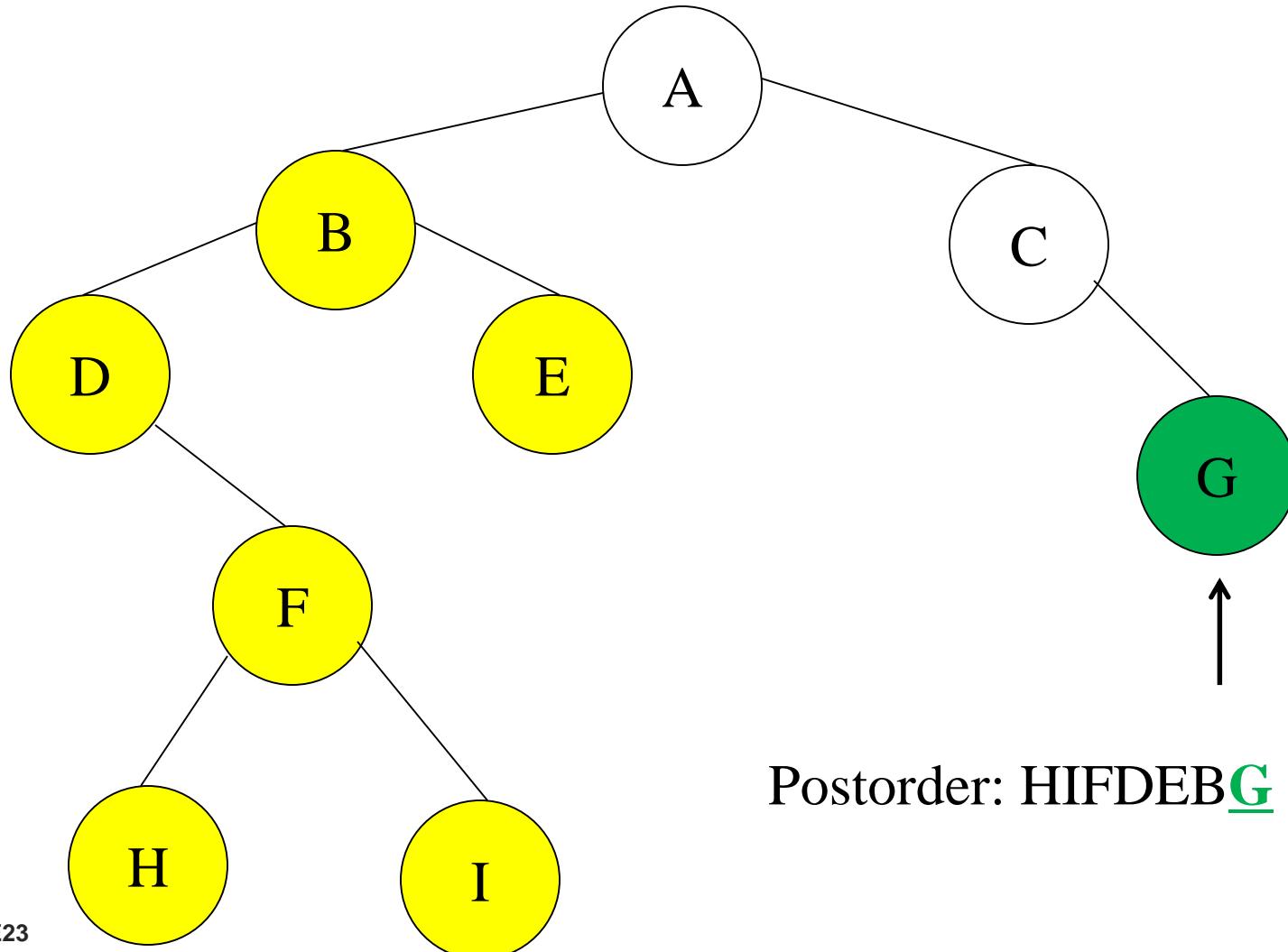
Модны нэвтрэлтийн жишээ



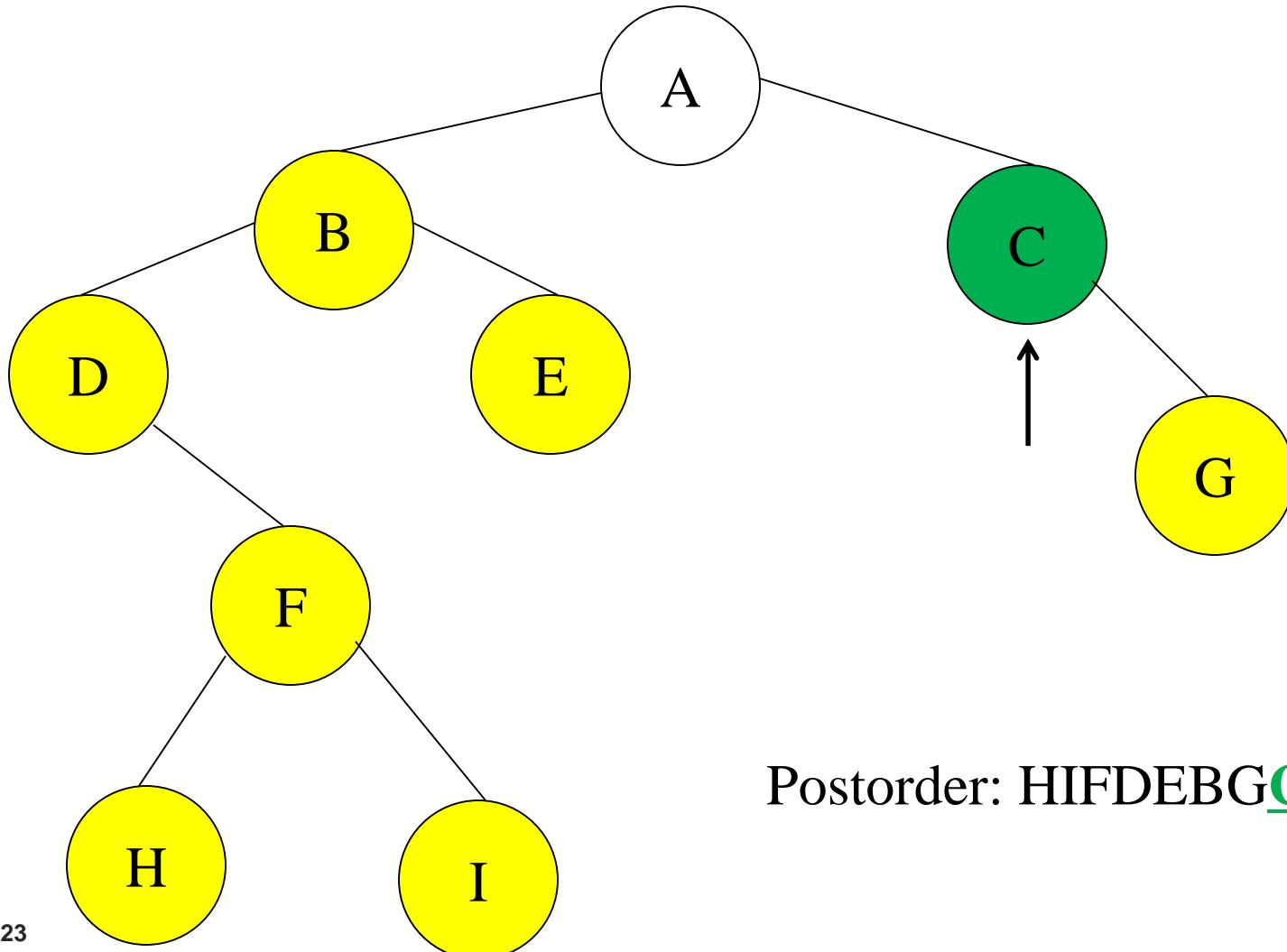
Модны нэвтрэлтийн жишээ



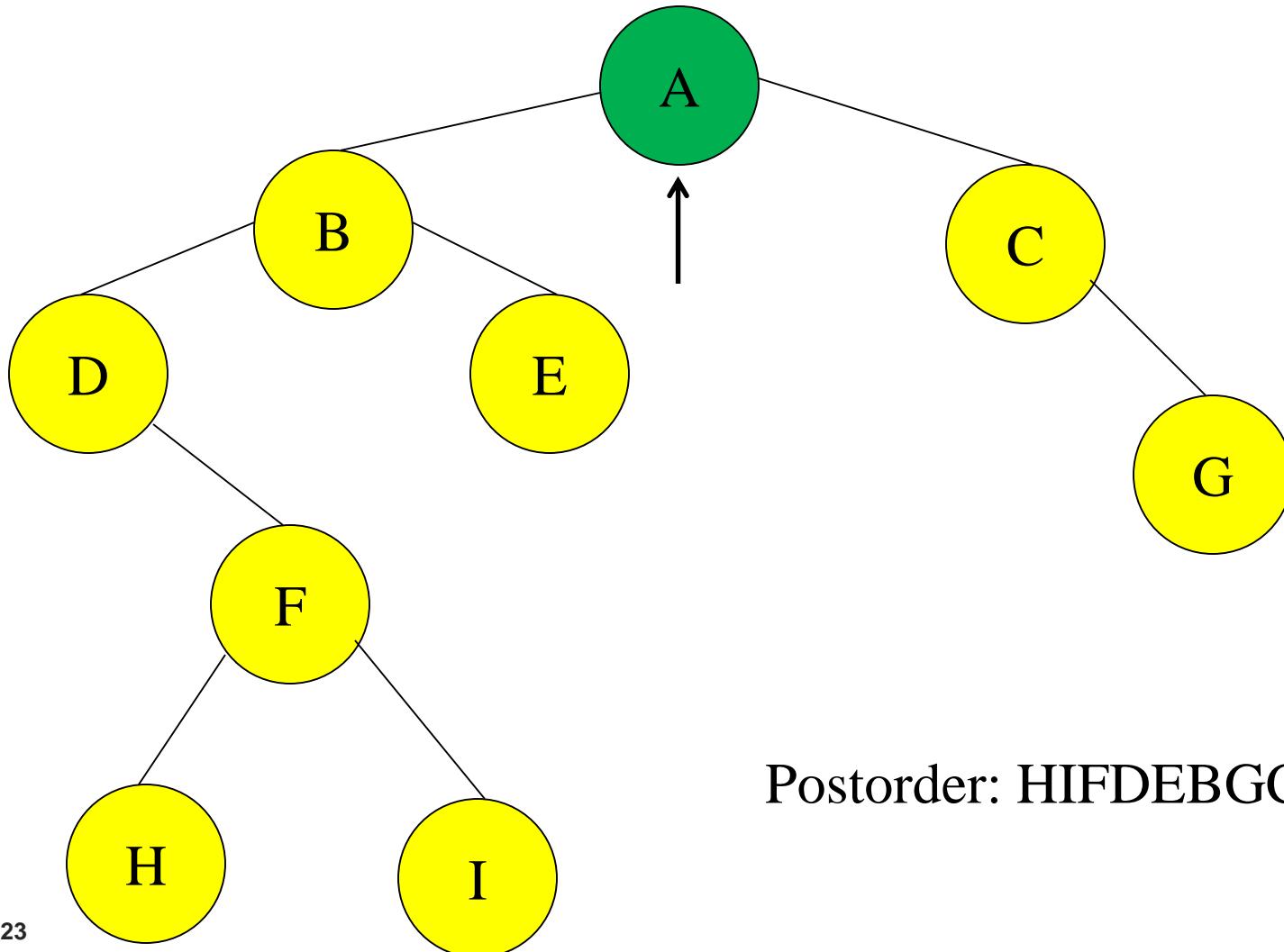
Модны нэвтрэлтийн жишээ



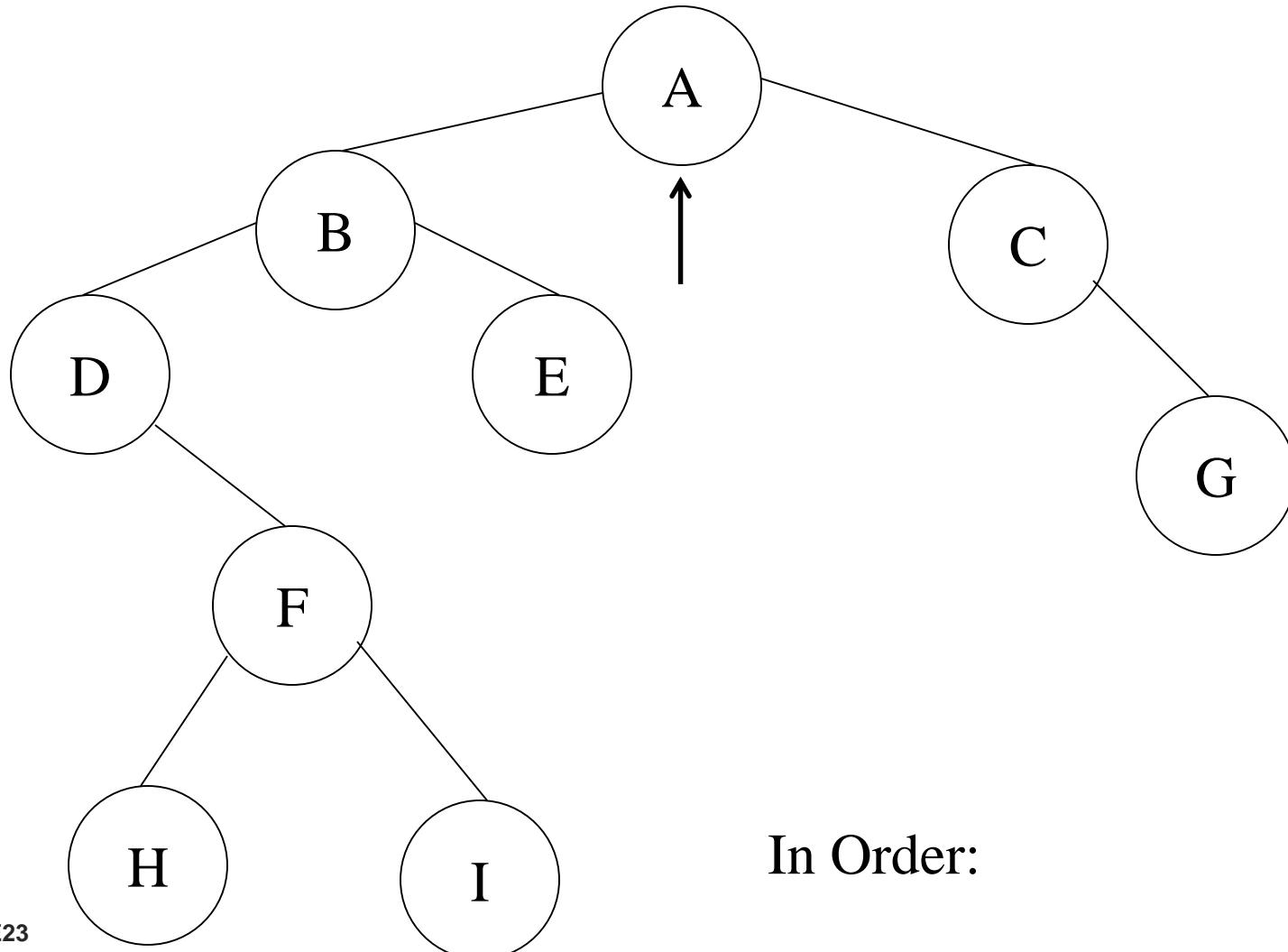
Модны нэвтрэлтийн жишээ



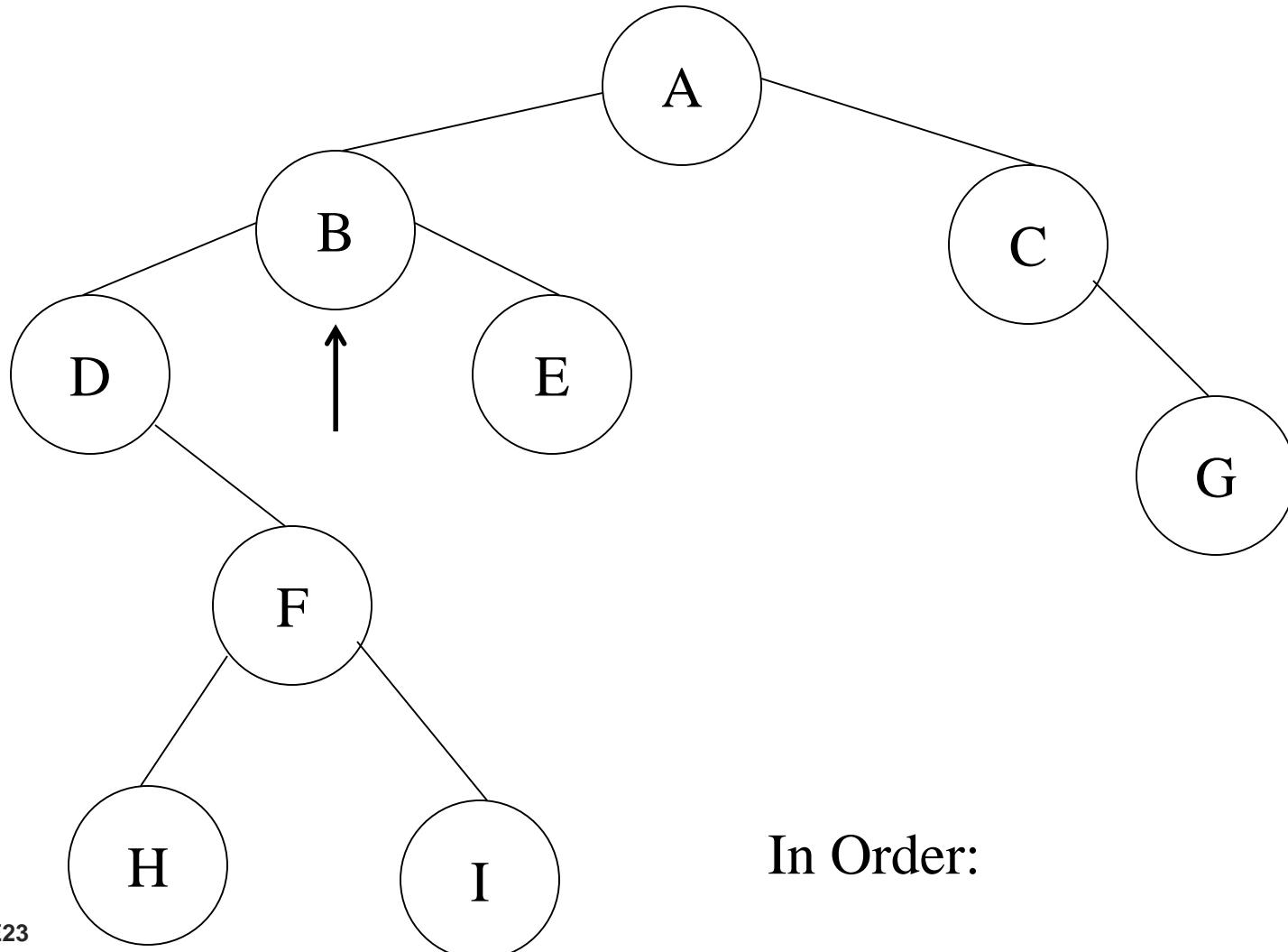
Модны нэвтрэлтийн жишээ



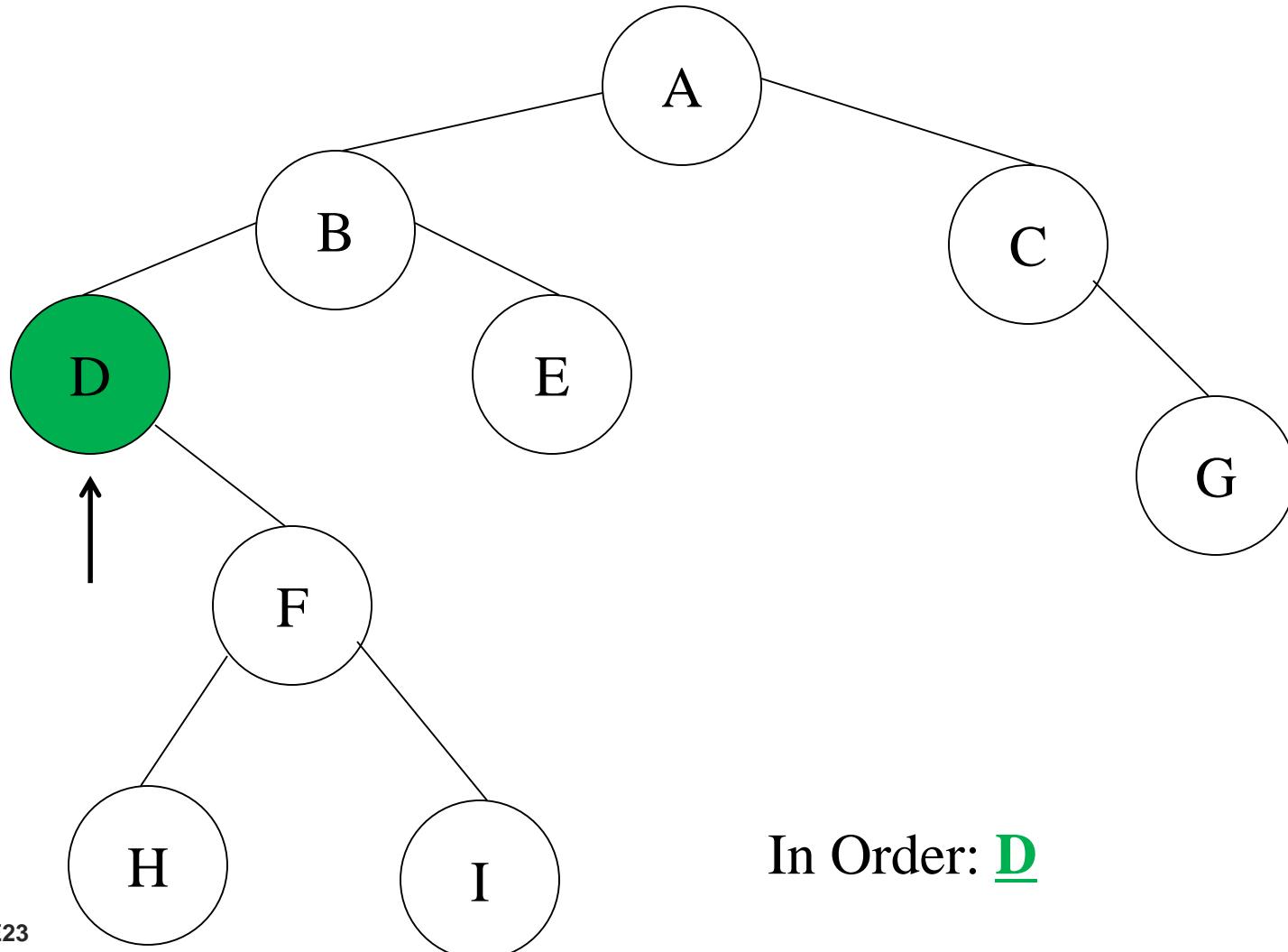
Модны нэвтрэлтийн жишээ



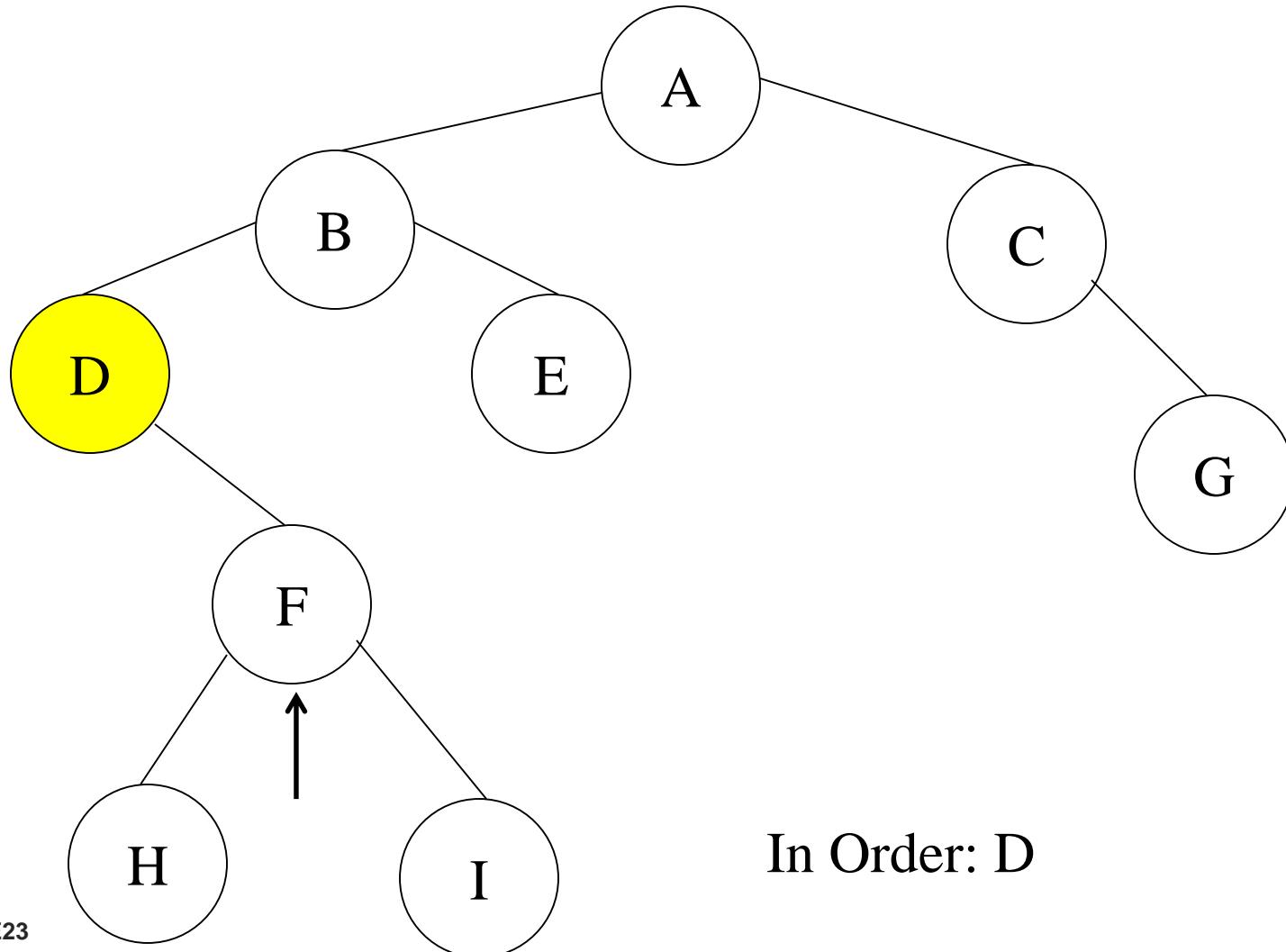
Модны нэвтрэлтийн жишээ



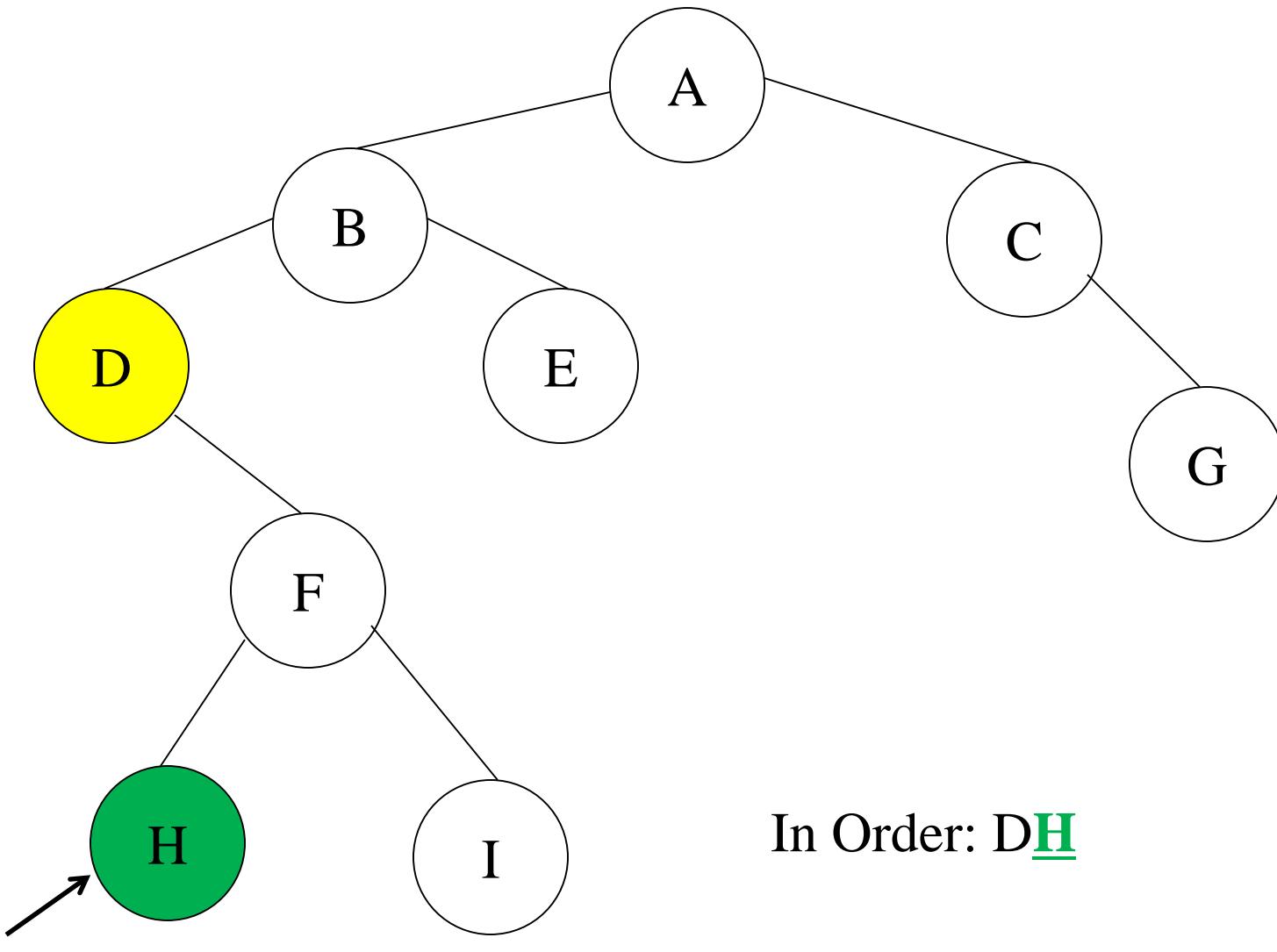
Модны нэвтрэлтийн жишээ



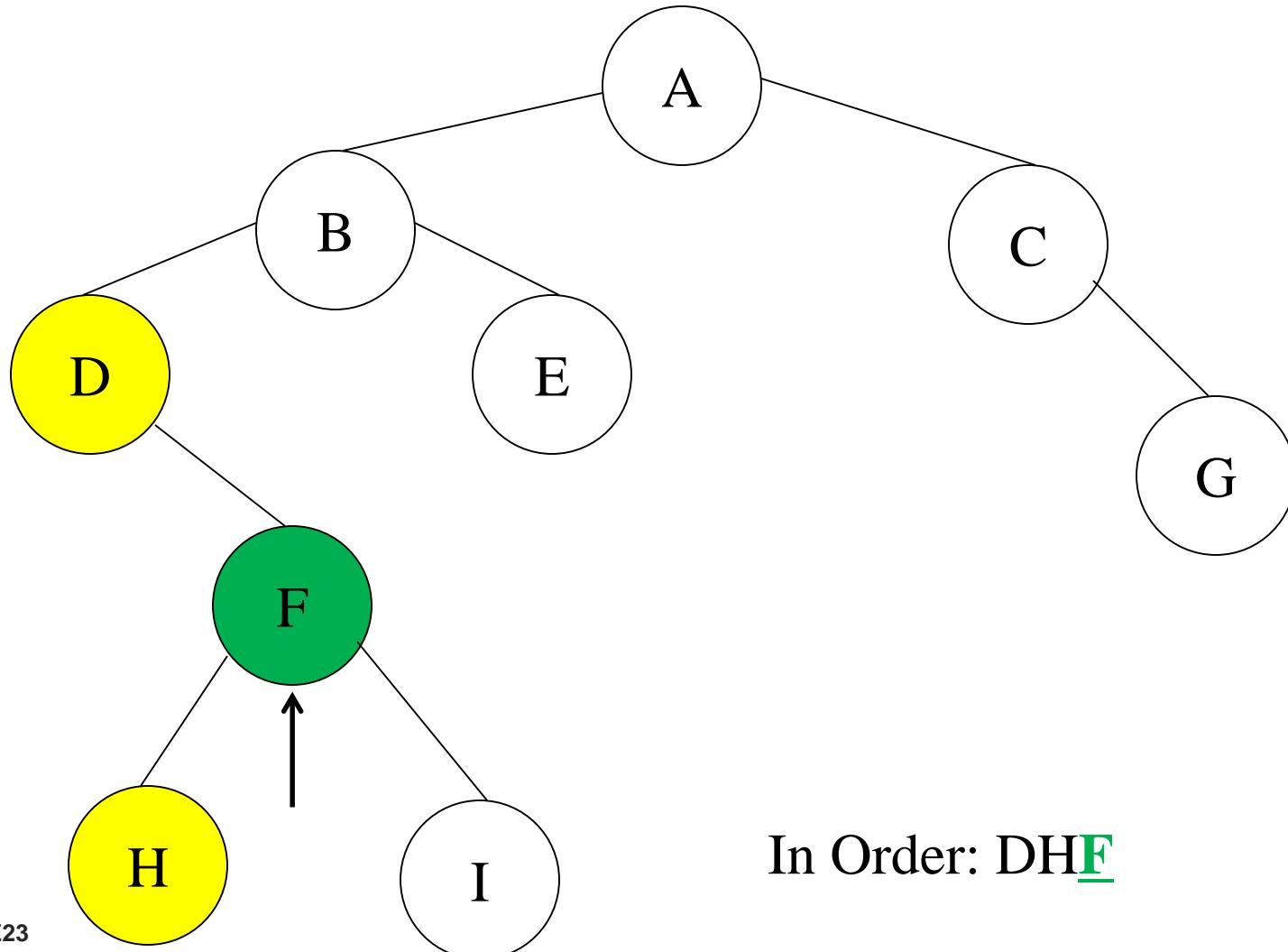
Модны нэвтрэлтийн жишээ



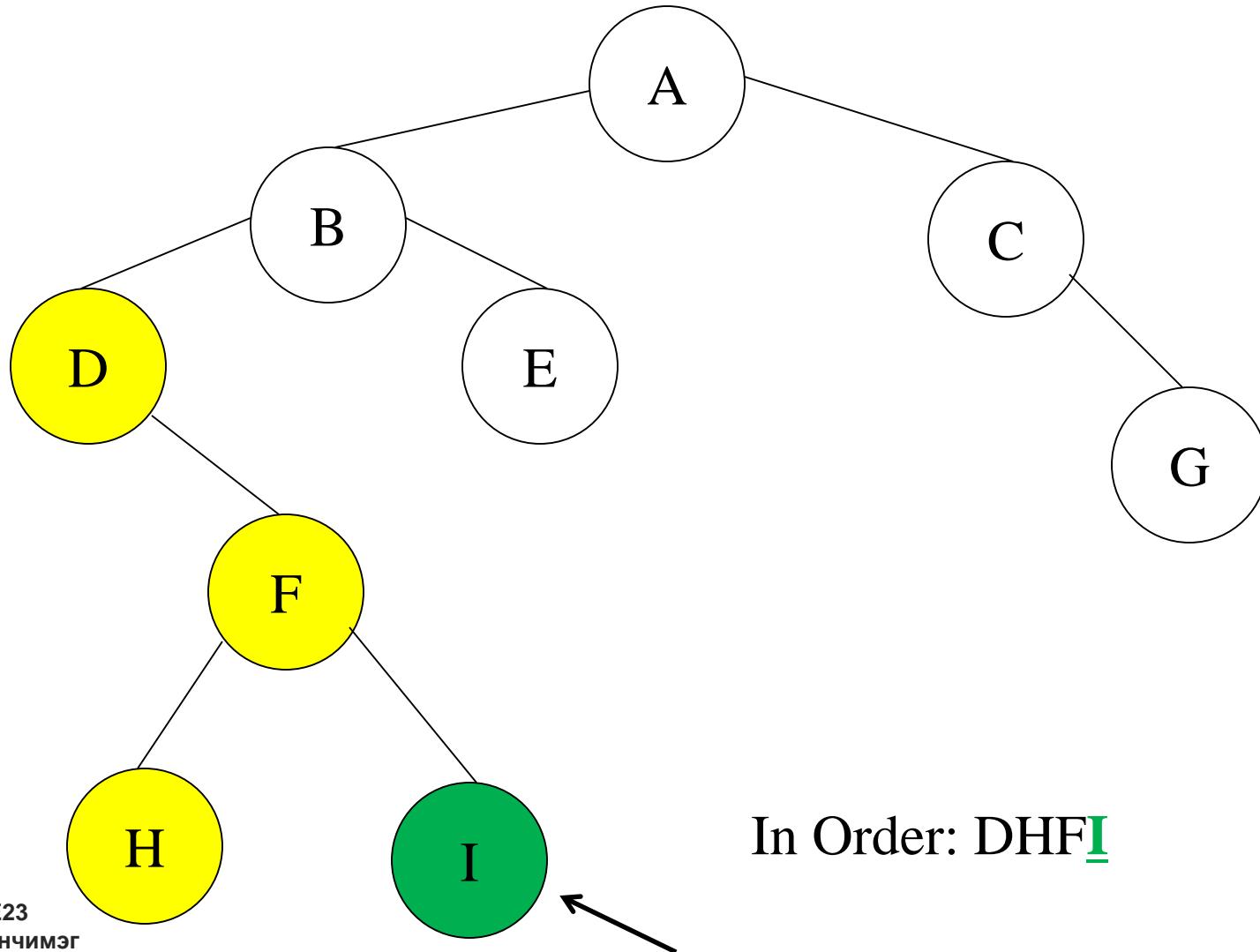
Модны нэвтрэлтийн жишээ



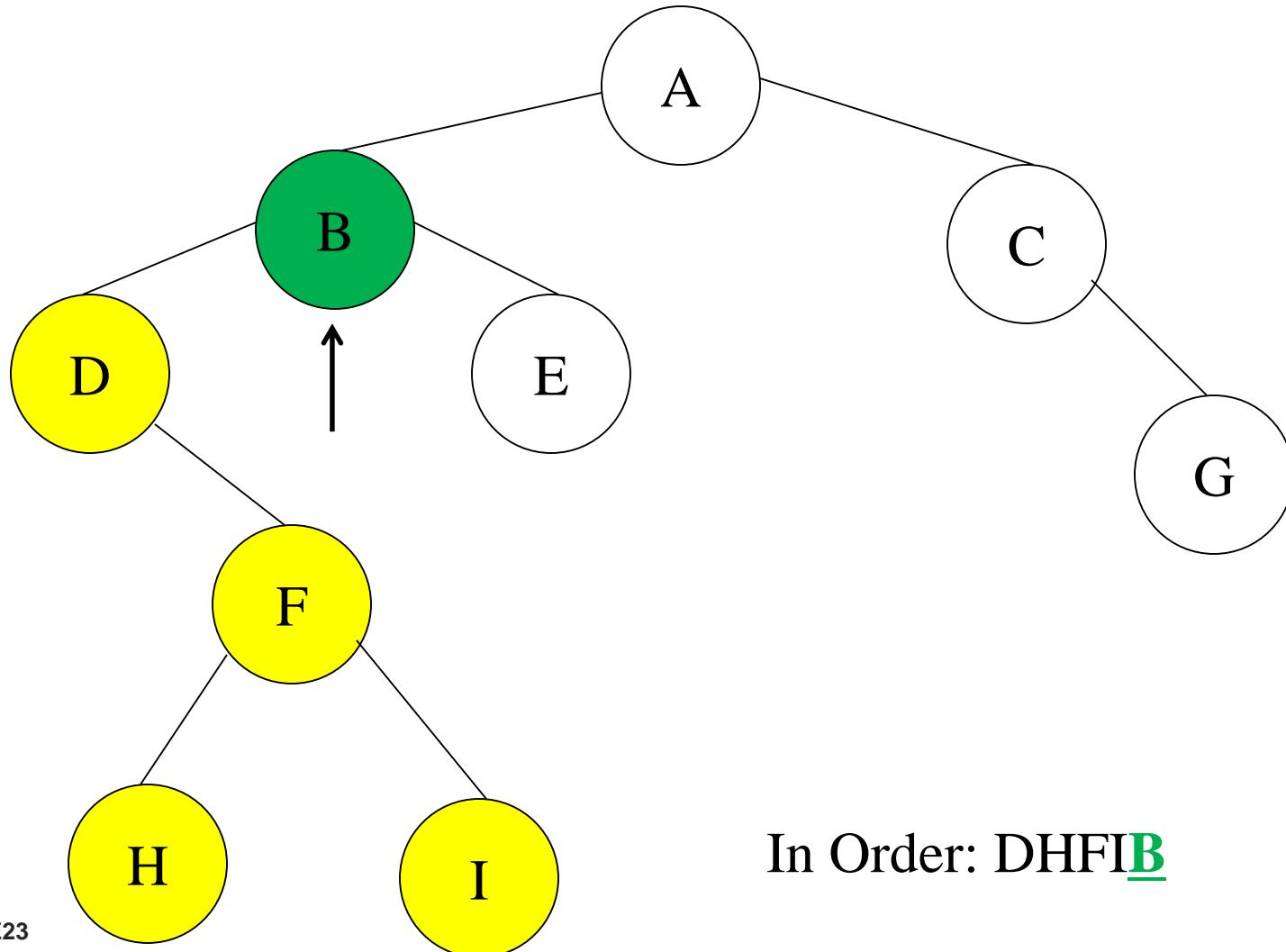
Модны нэвтрэлтийн жишээ



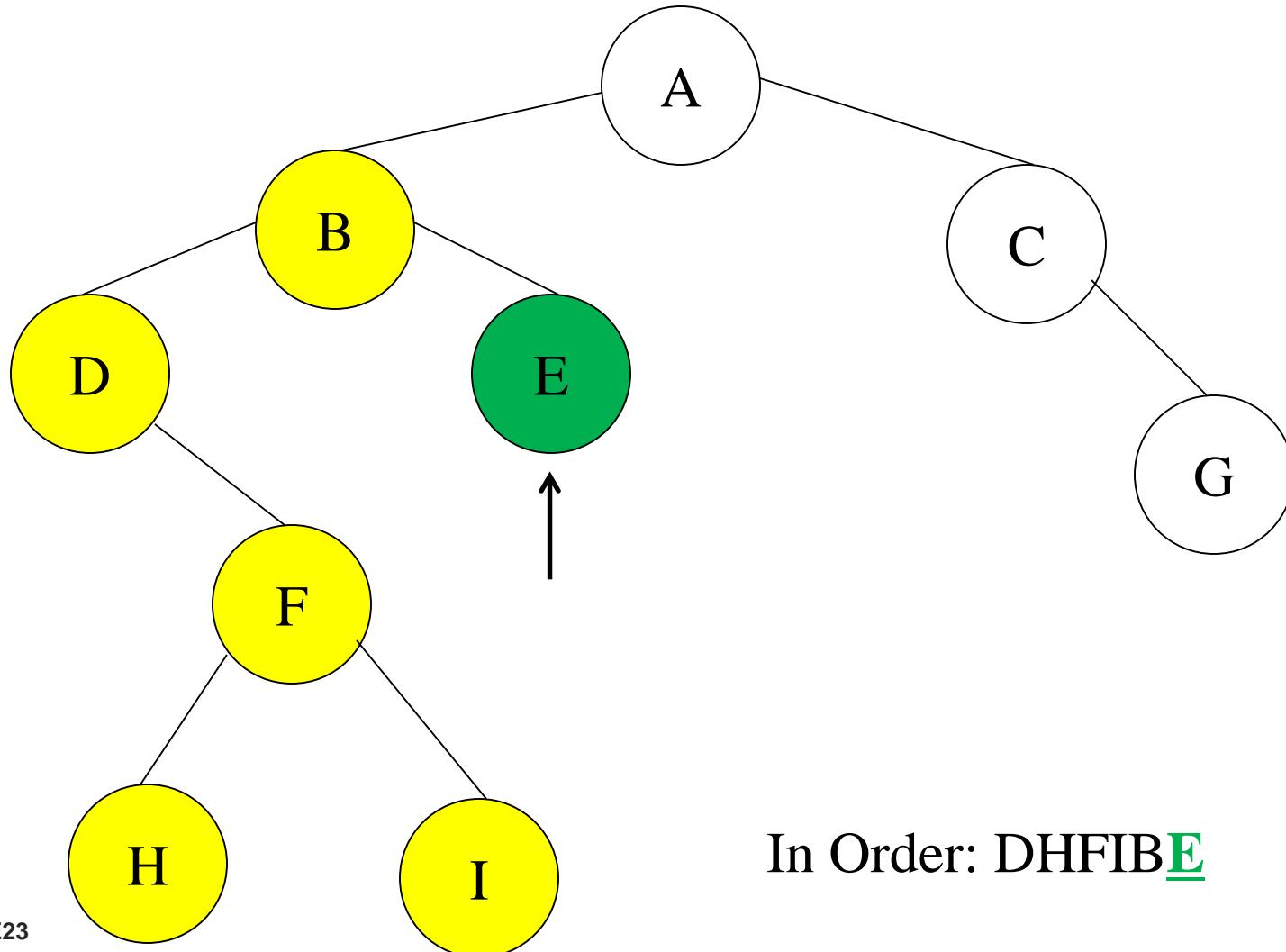
Модны нэвтрэлтийн жишээ



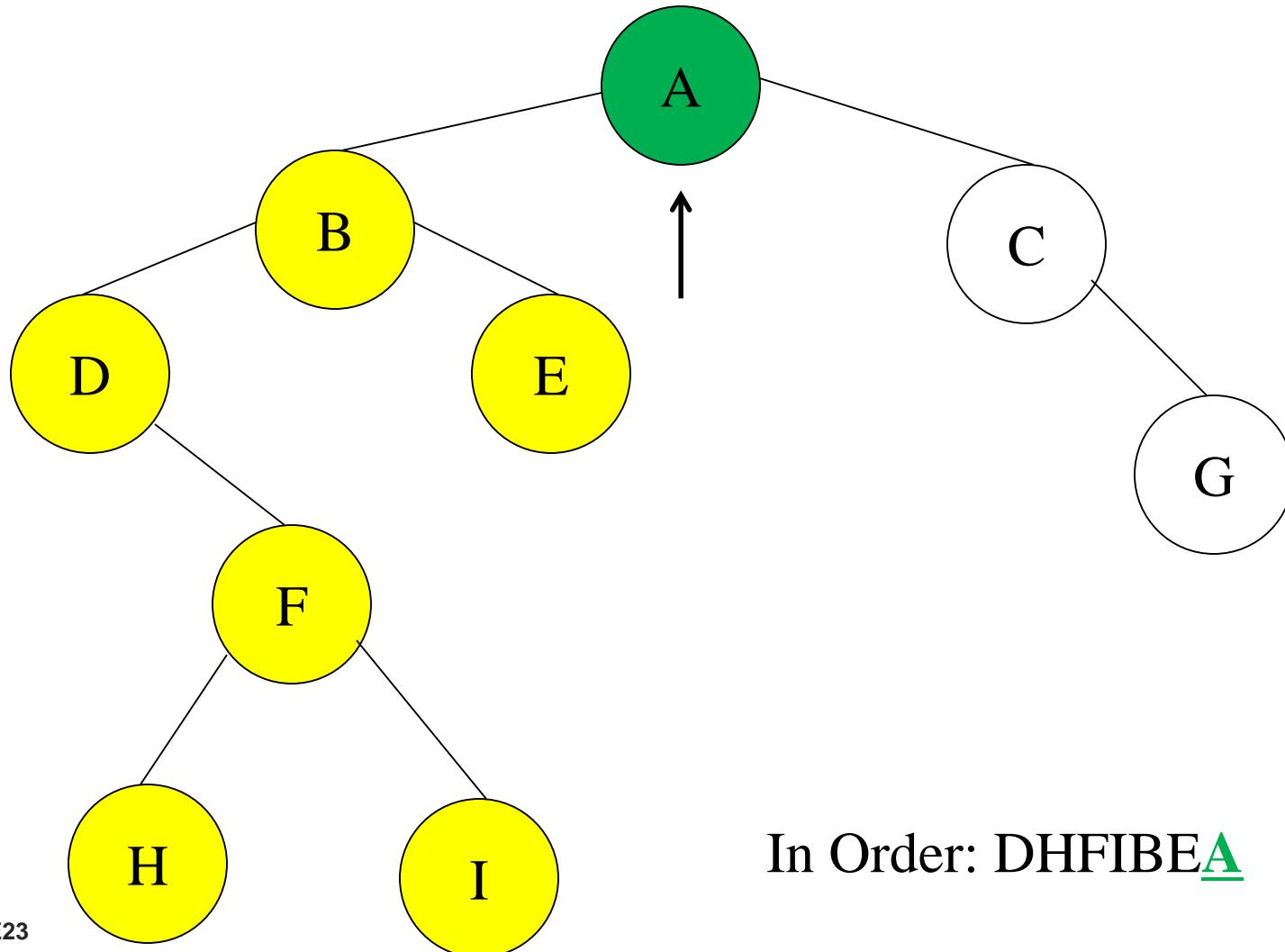
Модны нэвтрэлтийн жишээ



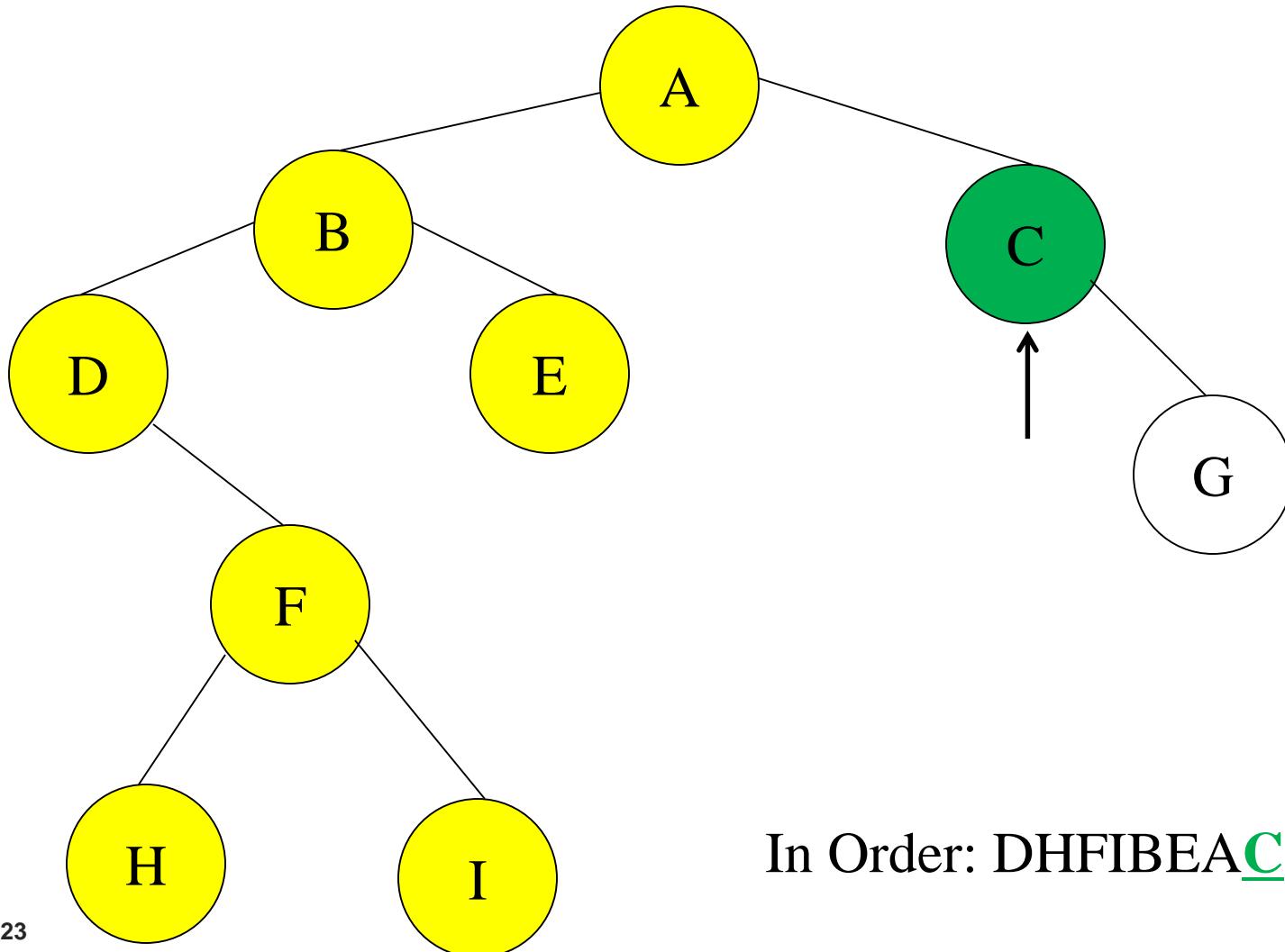
Модны нэвтрэлтийн жишээ



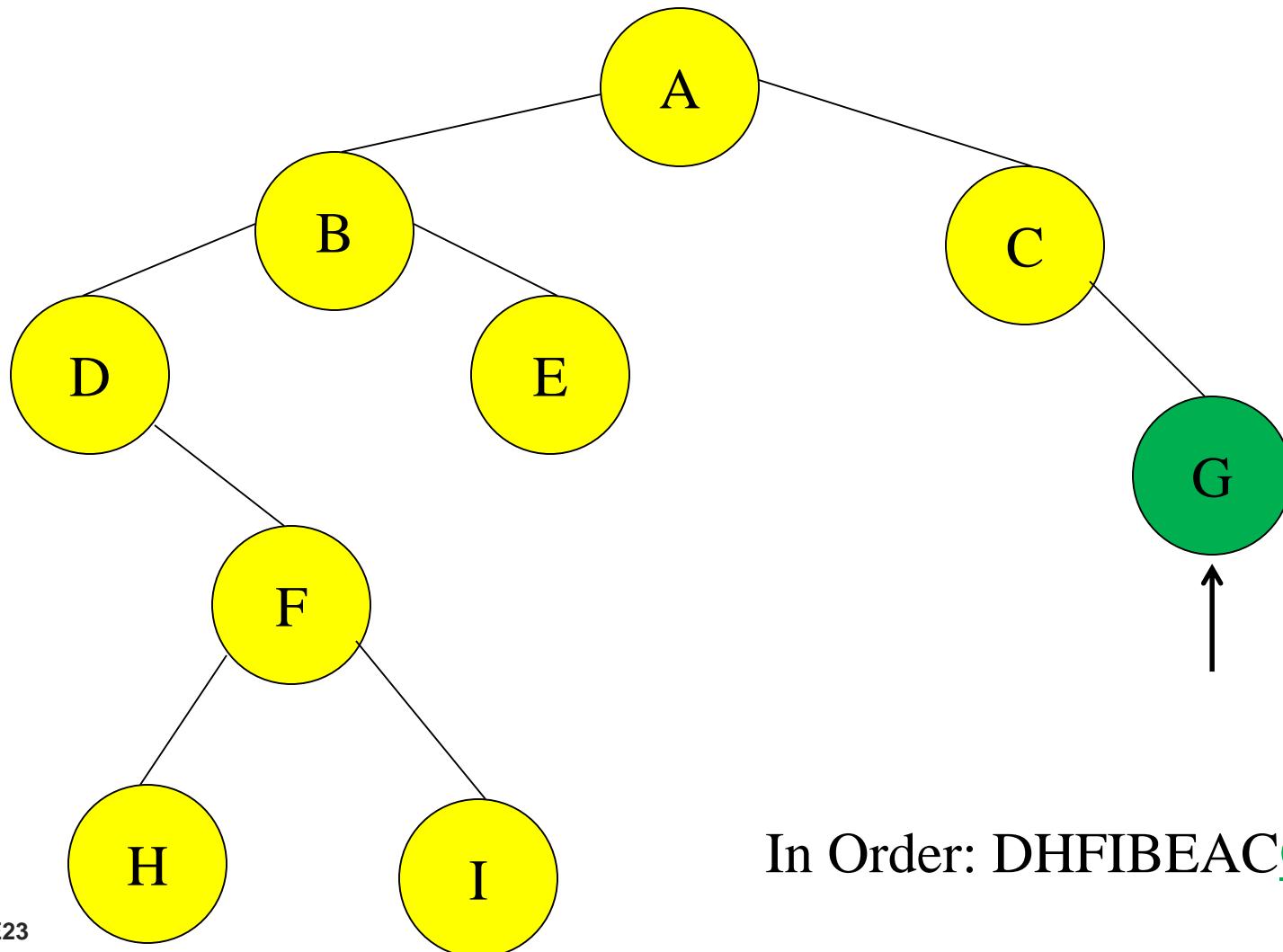
Модны нэвтрэлтийн жишээ



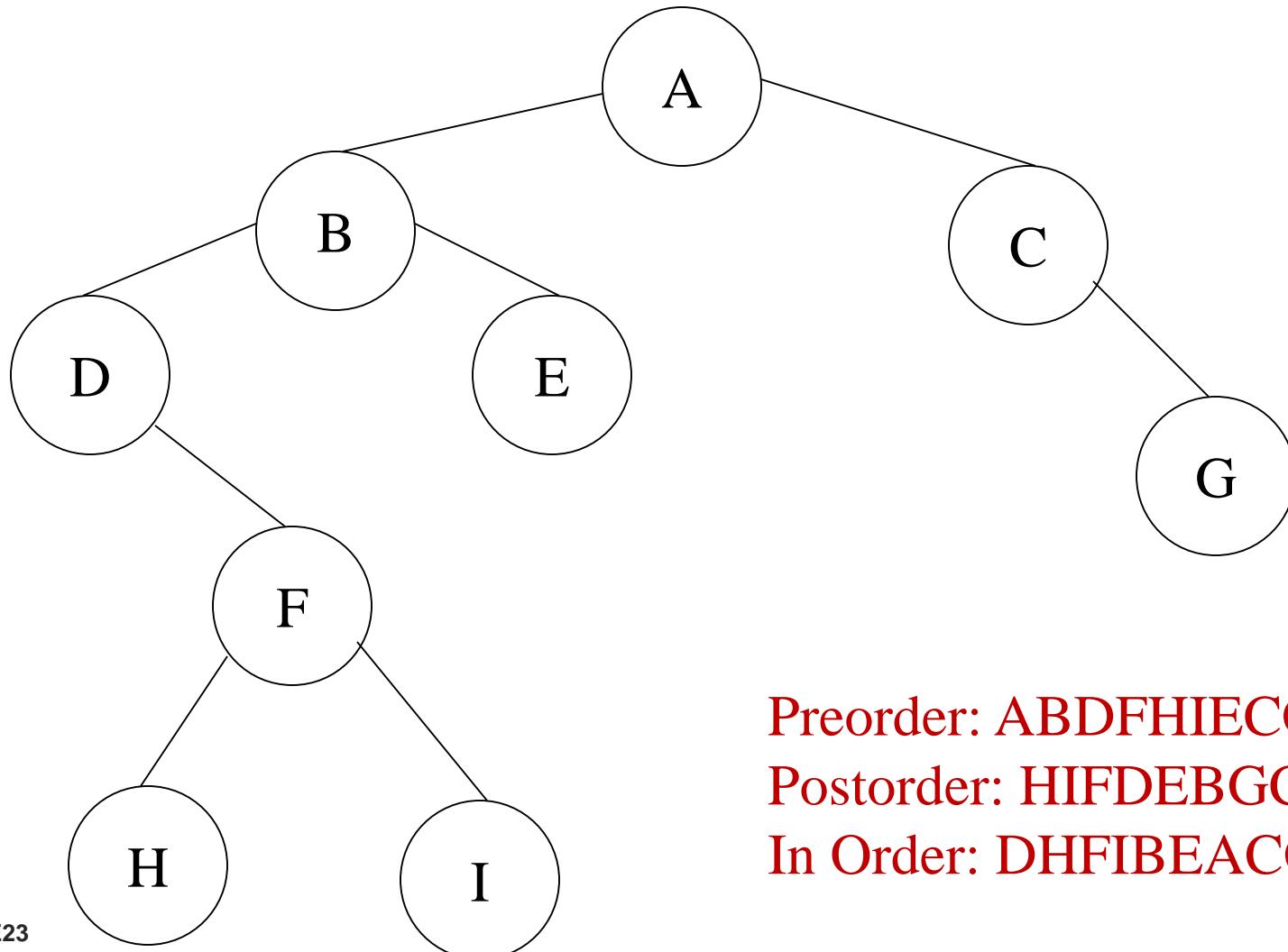
Модны нэвтрэлтийн жишээ



Модны нэвтрэлтийн жишээ



Модны нэвтрэлтийн жишээ



Хоёртын модны хэрэгжүүлэлт

- Бид хоёртын модыг нэг холбоост жагсаалттай ижил аргаар хэрэгжүүлж чадна. Үүнд: Хоёр зангилааны дараа дараагийн зангилаа байна.

```
public class Node {  
    private int data;  
    private Node left;  
    private Node right;
```

```
public int getData()  {
    return data;
}

public Node getLeft()  {
    return left;
}

public Node getRight()  {
    return right;
}

public void setData(int x)  {
    data = x;
}

public void setLeft(Node p)  {
    left = p;
}

public void setRight(Node p)  {
    right = p;
}      }
```

The `tree` Class

```
public class Tree {  
    private Node root;  
  
    // tree() - The default constructor - Starts  
    //           the tree as empty  
  
    public Tree() {  
        root = null;  
    }  
  
    // Tree() - An initializing constructor that  
    //           creates a node and places in it  
    //           the initial value
```

```
public Tree(int x) {  
    root = new Node();  
    root.setData(x);  
    root.setLeft(null);  
    root.setRight(null);  
}  
  
public Node getRoot() {  
    return root;        }  
  
// newNode() - Creates a new node with a  
// zero as data by default  
public Node newNode() {  
    Node p = new Node();  
    p.setData(0);  
    p.setLeft(null);  
    p.setRight(null);  
    return(p);        }
```

```
// newNode() - Creates a new node with the
//               parameter x as its value

public Node newNode(int x)  {

    Node p = new Node();

    p.setData(x);

    p.setLeft(null);

    p.setRight(null);

    return(p);        }

//travTree() - initializes recursive

//      traversal of tree

public void travTree() {

    if (root != null)

        travSubtree(root);

        System.out.println();        }

//travSubtree() - recursive method used to

// traverse a binary tree (inorder)

public void travSubtree(Node p) {

    if (p != null)  {

        travSubtree(p.getLeft());

        System.out.print(p.getData() + "\t");

        travSubtree(p.getRight());        }        }
```

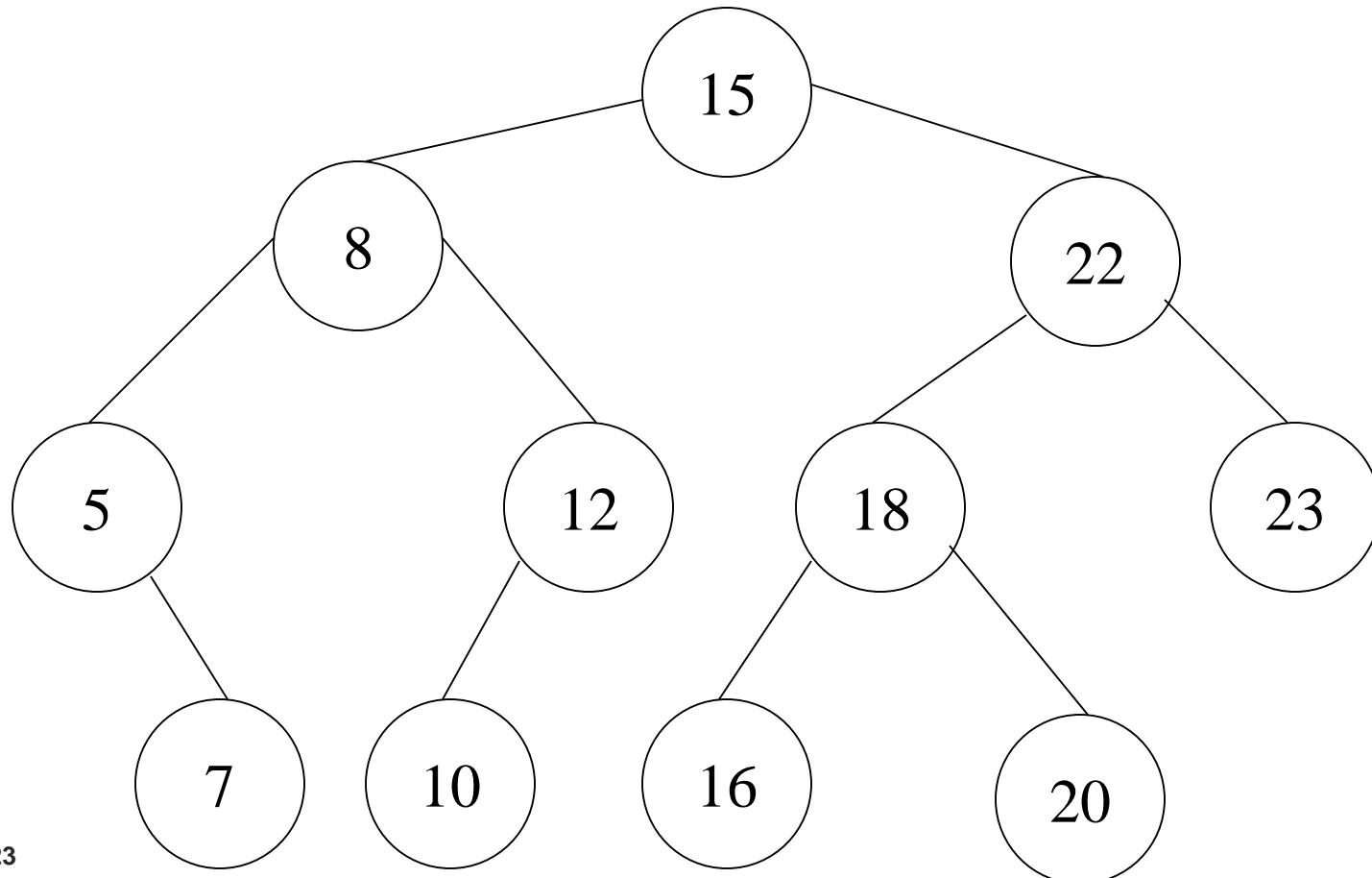
```
// addLeft() - Inserts a new node containing
//           x as the left child of p
public void addLeft(Node p, int x) {
    Node q = newNode(x);
    p.setLeft(q);
}

// addRight() - Inserts a new node containing
//           x as the right child of p
public void addRight(Node p, int x) {
    Node q = newNode(x);
    p.setRight(q);
}
```

Модны хайлт

Хэрэв бид модон дээр х утгатай шинэ зангилаа нэмбэл энгийн хайлтын мод үүсгэж болно. Үүнд:

- Х нь зангилааны утгаас бага байх болгонд бид зүүн тийшээ доошилно.
- Х нь зангилааны утгаас их байх бүрд бид баруун тийшээ доошилно.



```
// insert() - Insert value x in a new node to
//                         be inserted after p
public void    insert(int x)  {
    Node      p, q;
    boolean   found = false;    p = root;    q = null;

    while (p != null && !found)  {
        q = p;
        if (p.getData() == x)
            found = true;
        else if (p.getData() > x)
            p = p.getLeft();
        else
            p = p.getRight();
    } if (found)
        error("Duplicate entry");

if (q.getData() > x)
    addLeft(q, x);
else
    addRight(q, x);
    //q = newNode(x);
}
```

```
// isXThere() -      Is there a node in the
//                           tree containing x?
public boolean isXThere(int x)  {
    Node  p;
    boolean   found = false;

    p = root;
    while (p != null && !found)  {

        if (p.getData() == x)
            found = true;
        else if (p.getData() > x)
            p = p.getLeft();
        else
            p = p.getRight();
    }
    return(found);
} public void  error(String message)  {
    System.out.println(message);
    System.exit(0);
}
```

```
// getNode() - Get the pointer for the
//               node containing x
public Node getNode(int x) {
    Node     p, q;
    boolean found = false;

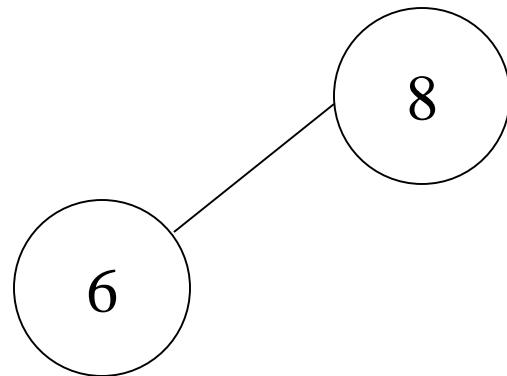
    p = root;
    q = null;
    while (p != null && !found) {
        q = p;
        if (p.getData() == x)
            found = true;
        else if (p.getData() > x)
            p = p.getLeft();
        else
            p = p.getRight();
    }
    if (found)
        return(q);
    else
        return(null);
}
```

```
public class TestTree {  
    public static void main(String[] args) {  
        Tree mytree = new Tree(8);  
        mytree.addLeft(mytree.getRoot(), 6);  
        mytree.addRight(mytree.getRoot(), 9);  
        mytree.insert(4);  
        mytree.insert(1);  
        mytree.insert(12);  
  
        if (mytree.isXThere(13))  
            System.out.println("great");  
        else  
            System.out.println("not great, Bob");  
        mytree.travTree();  
    }  
}
```

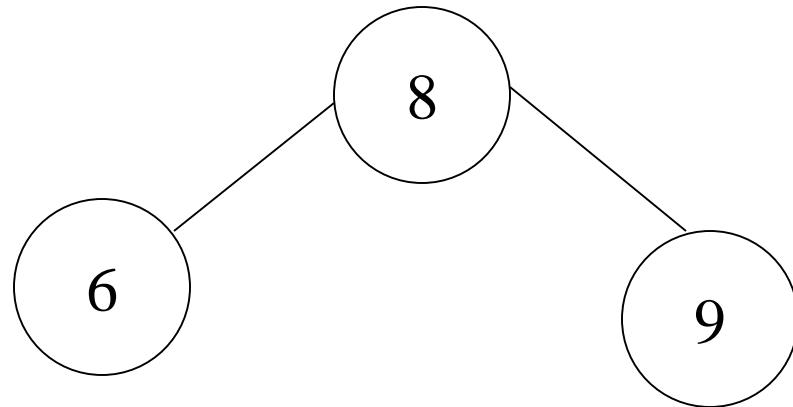
Туршилтын мод

8

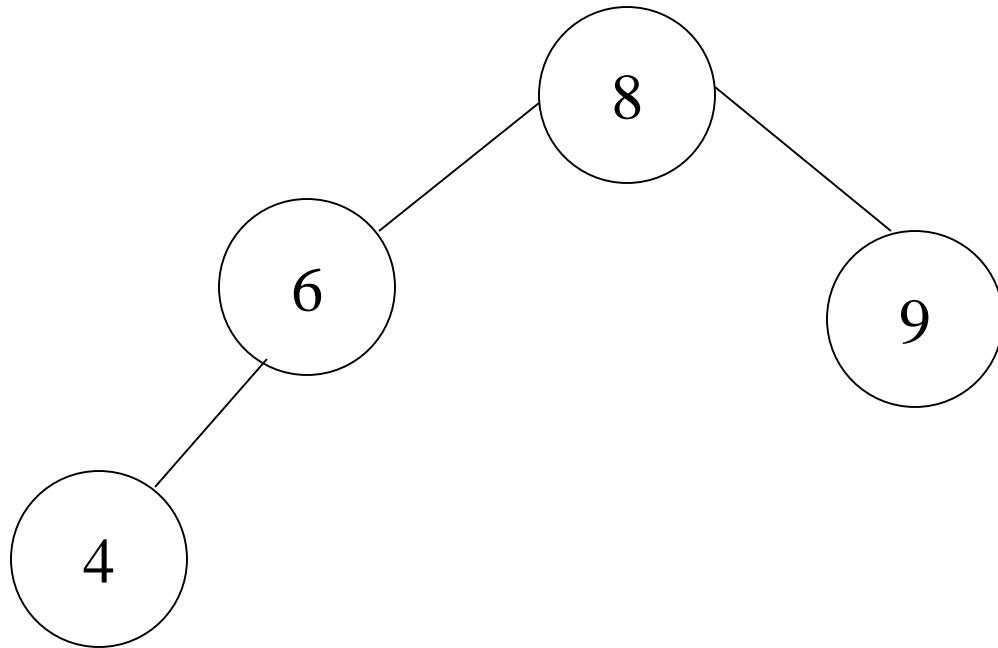
Туршилтын мод



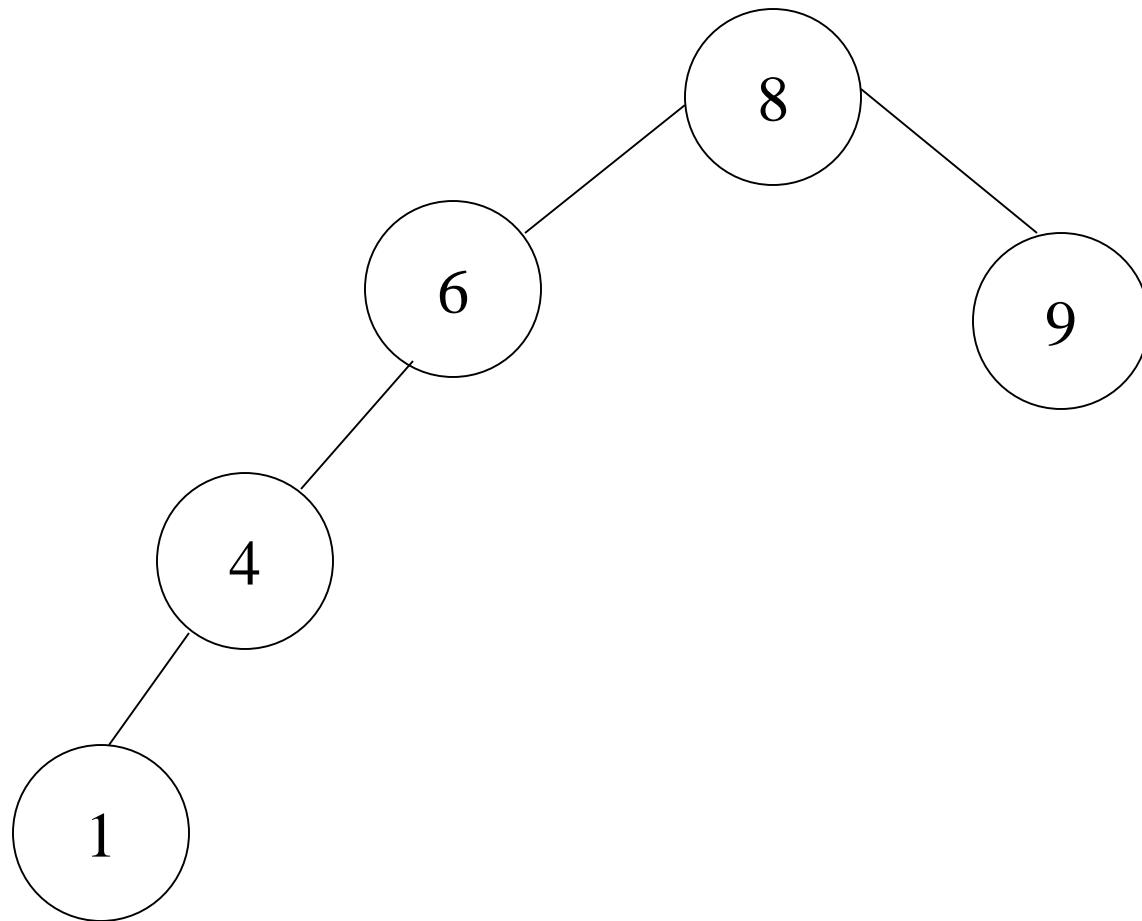
Туршилтын мод



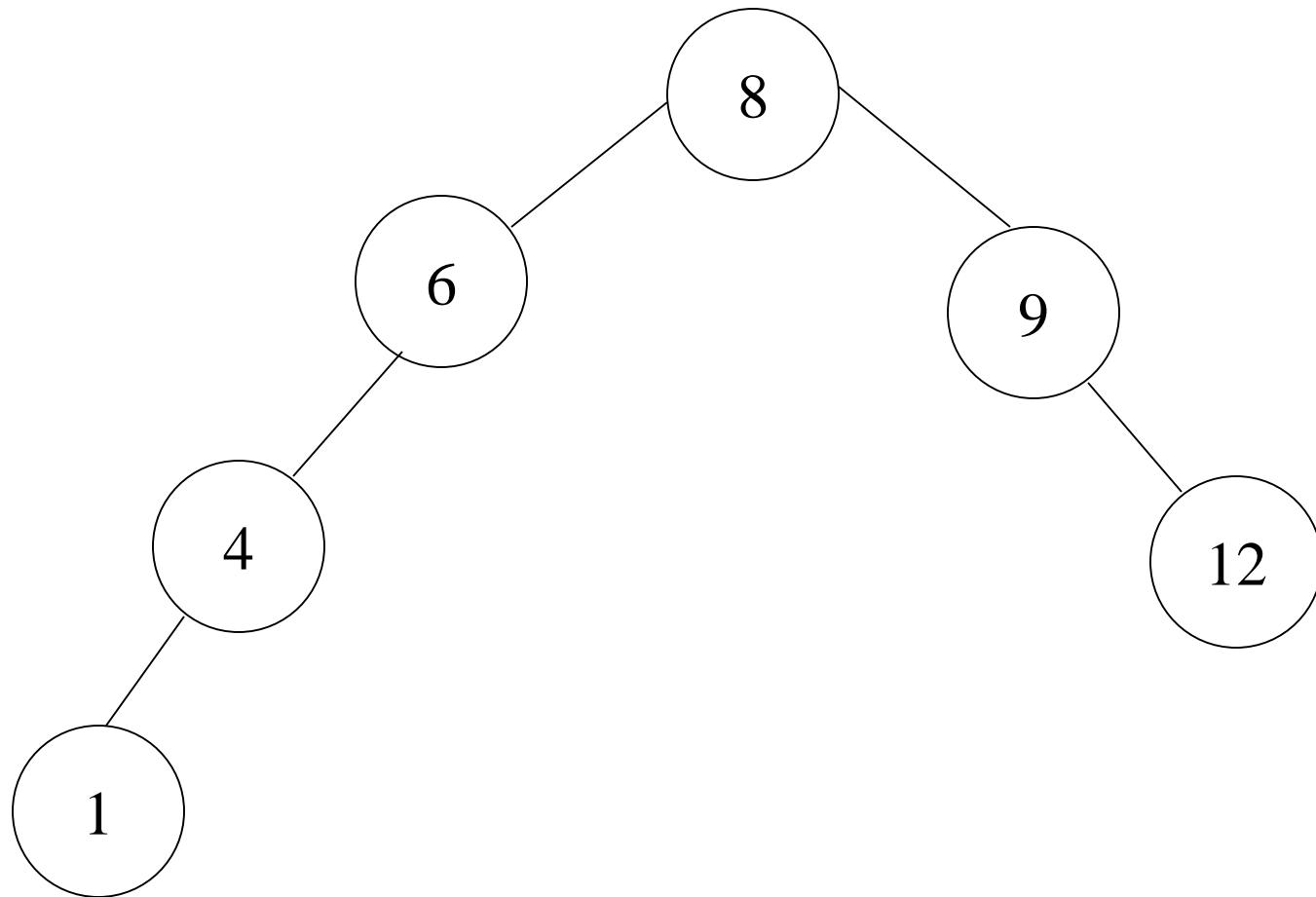
Туршилтын мод



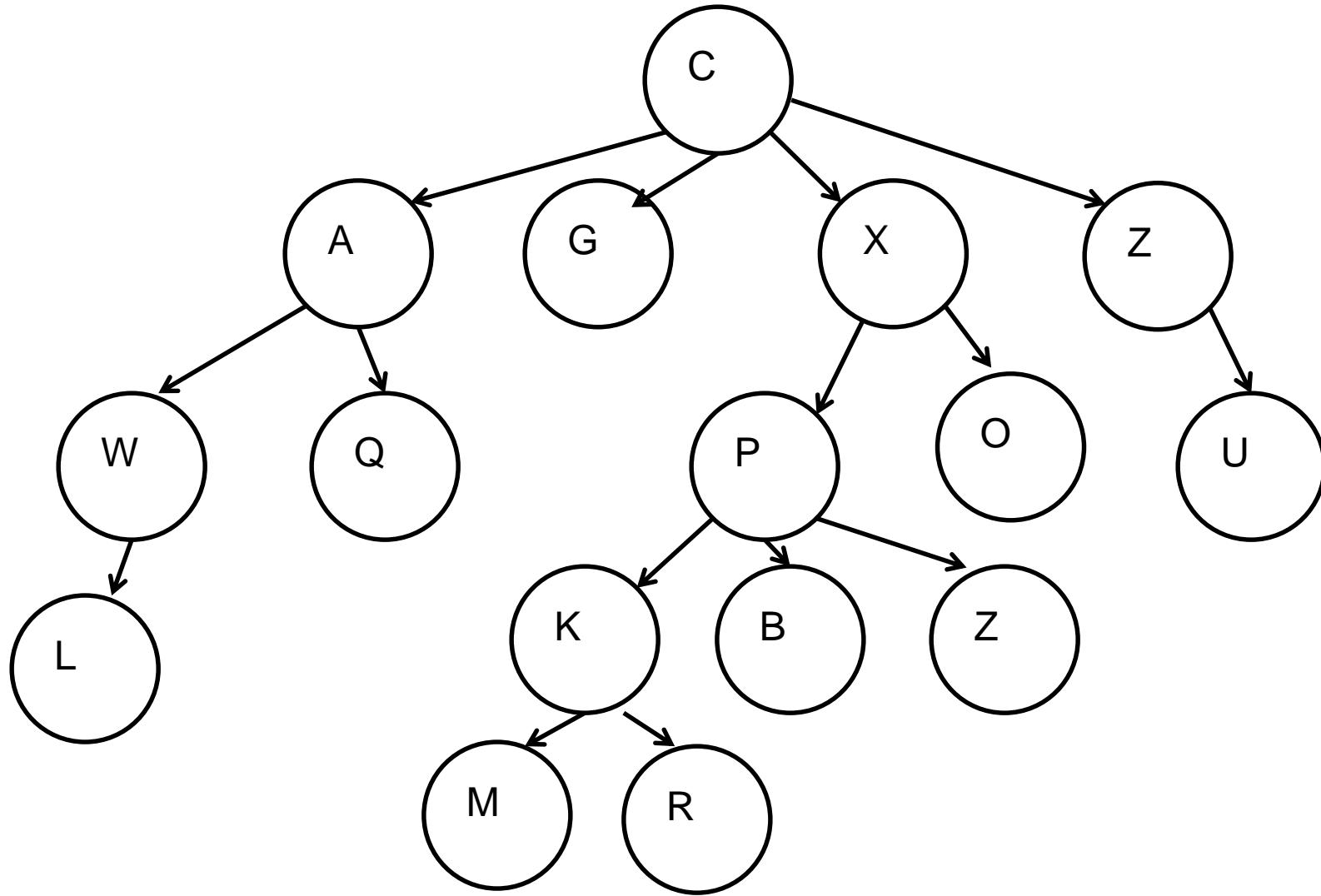
Туршилтын мод



Туршилтын мод



Модны өргөнөөр хайх

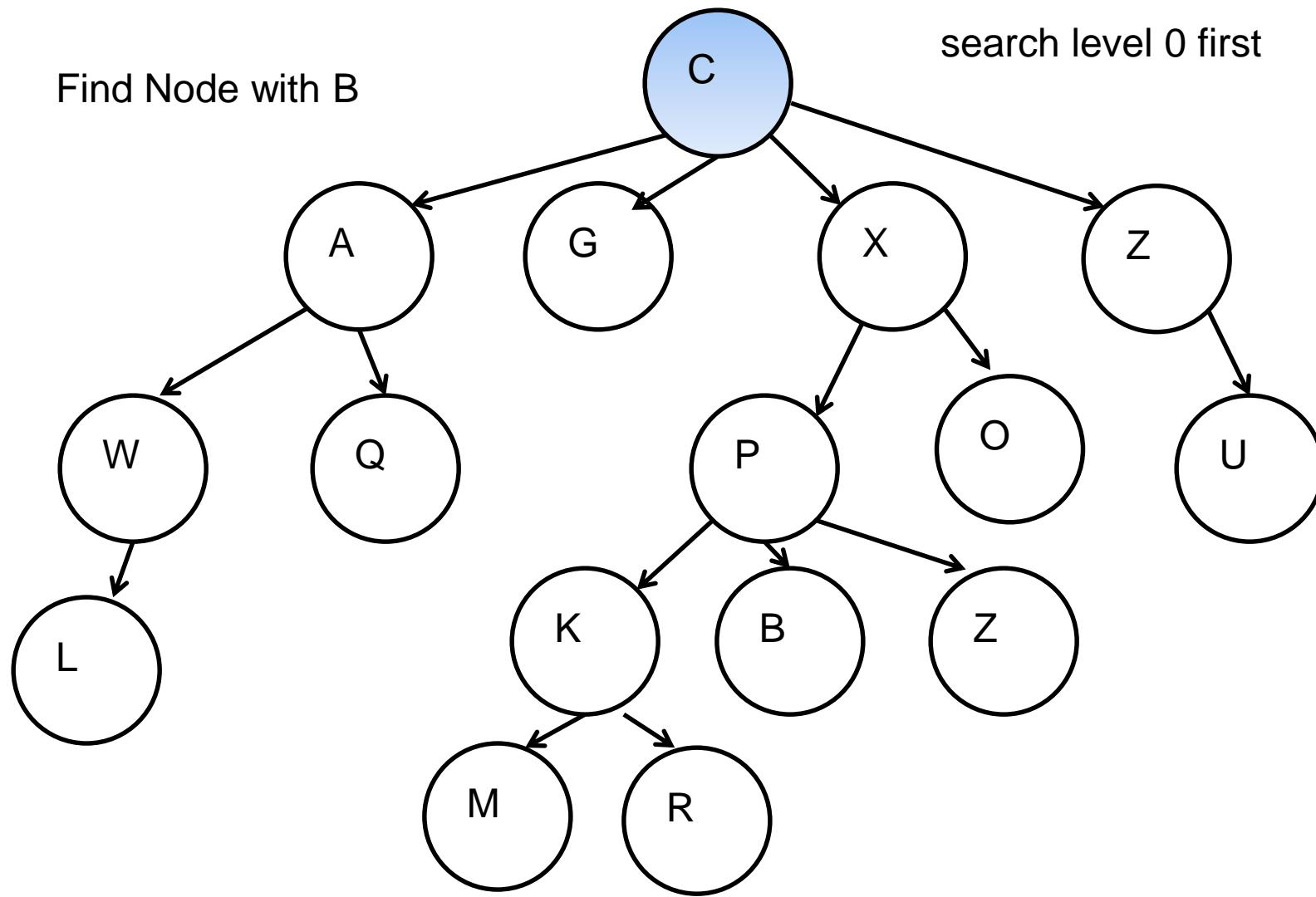


- Модны түвшний эрэмбэ дарааллыг эхний хайлт болгон ашиглана.
- Дараагийн түвшинд орохосоо өмнө бүх зангилааг нэг түвшинд хайна.

Breadth First Search / Өргөнөөр хайх

Find Node with B

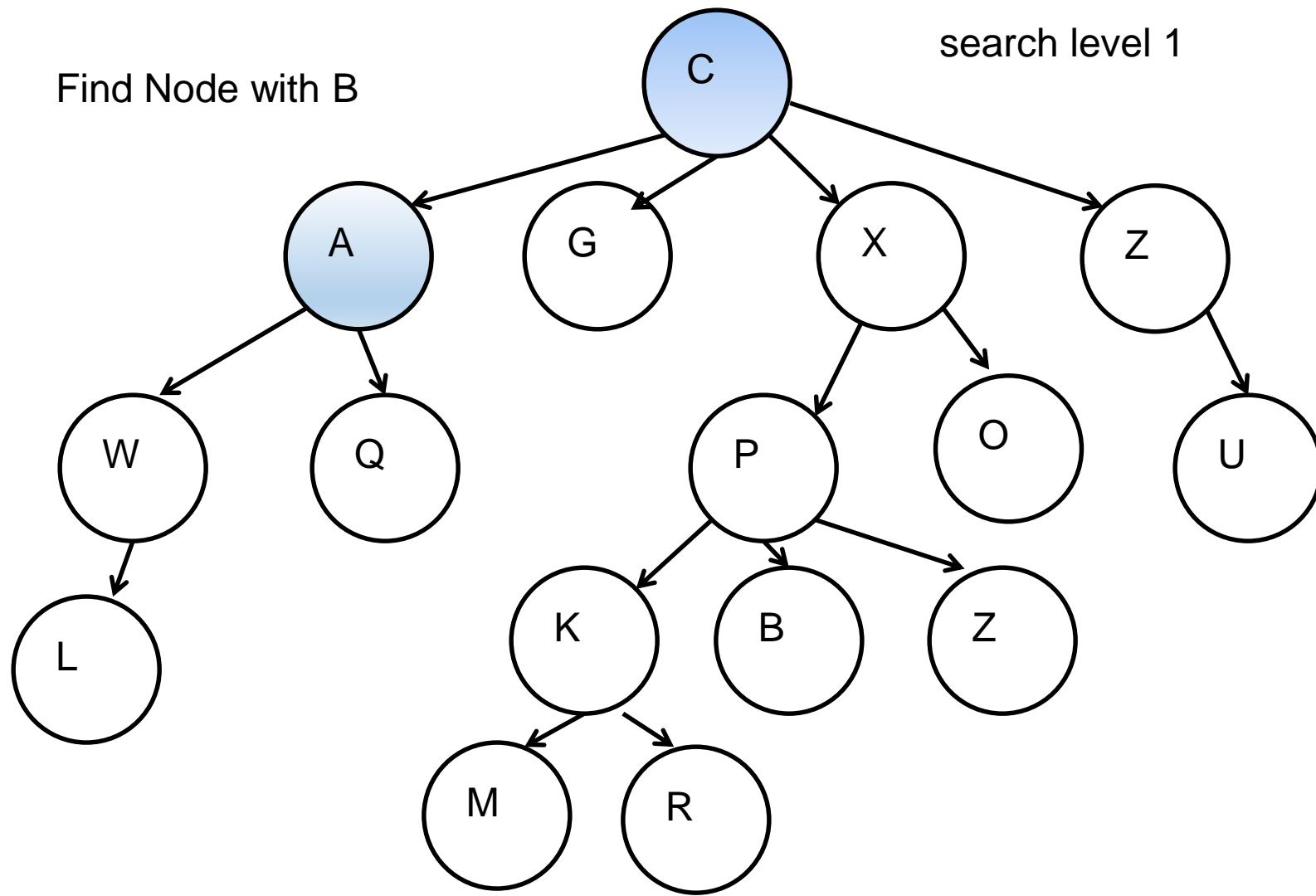
search level 0 first



Breadth First Search / Өргөнөөр хайх

Find Node with B

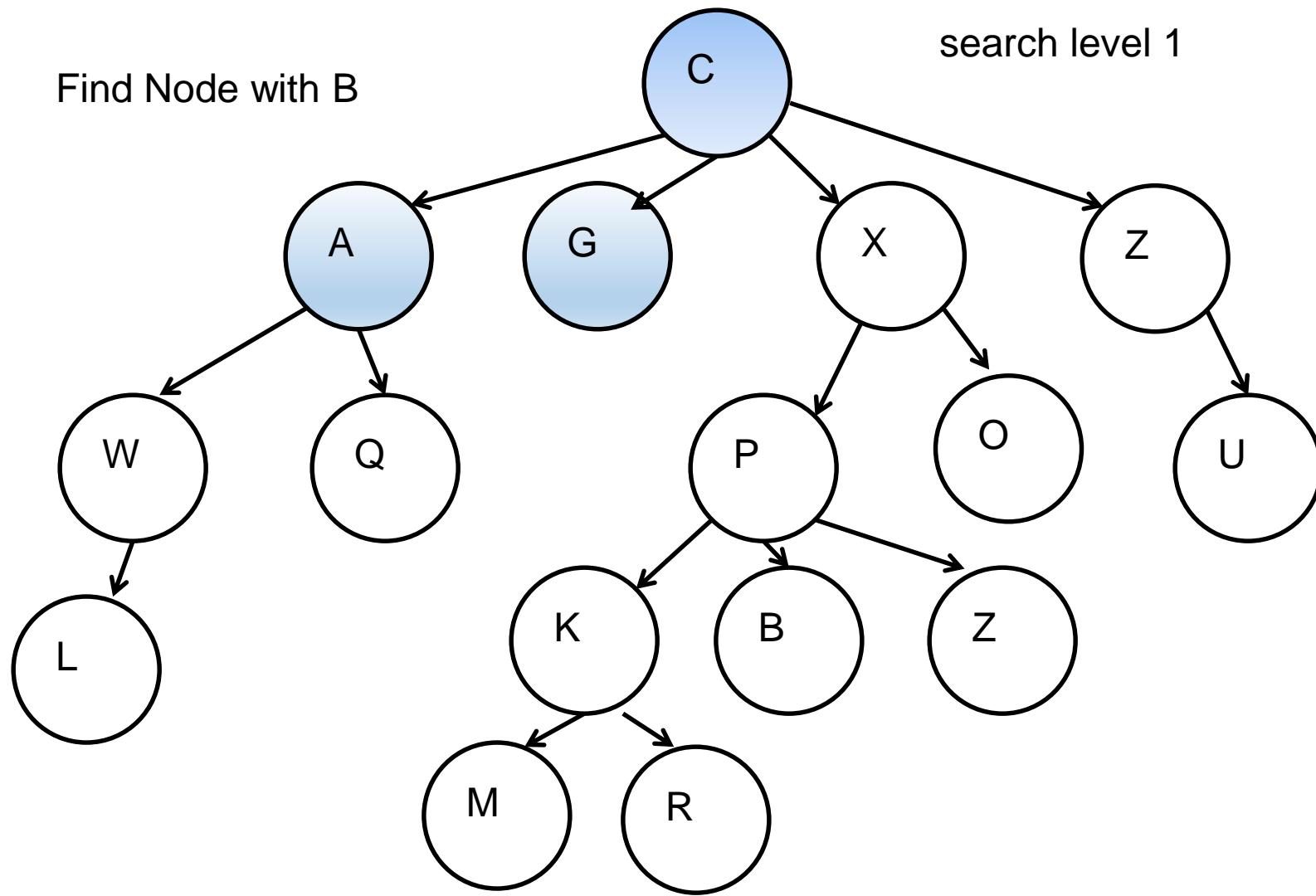
search level 1



Breadth First Search / Өргөнөөр хайх

Find Node with B

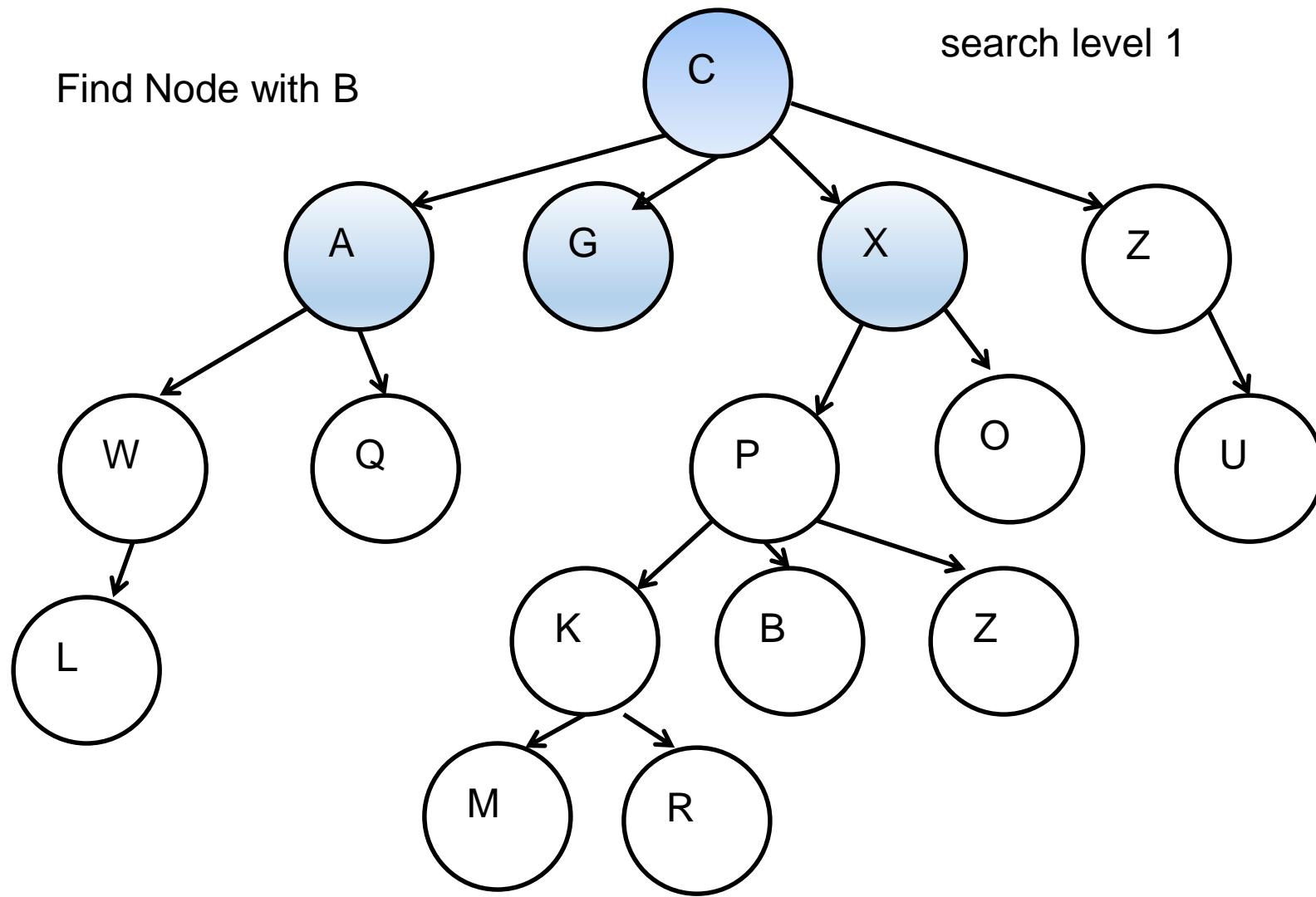
search level 1



Breadth First Search / Өргөнөөр хайх

Find Node with B

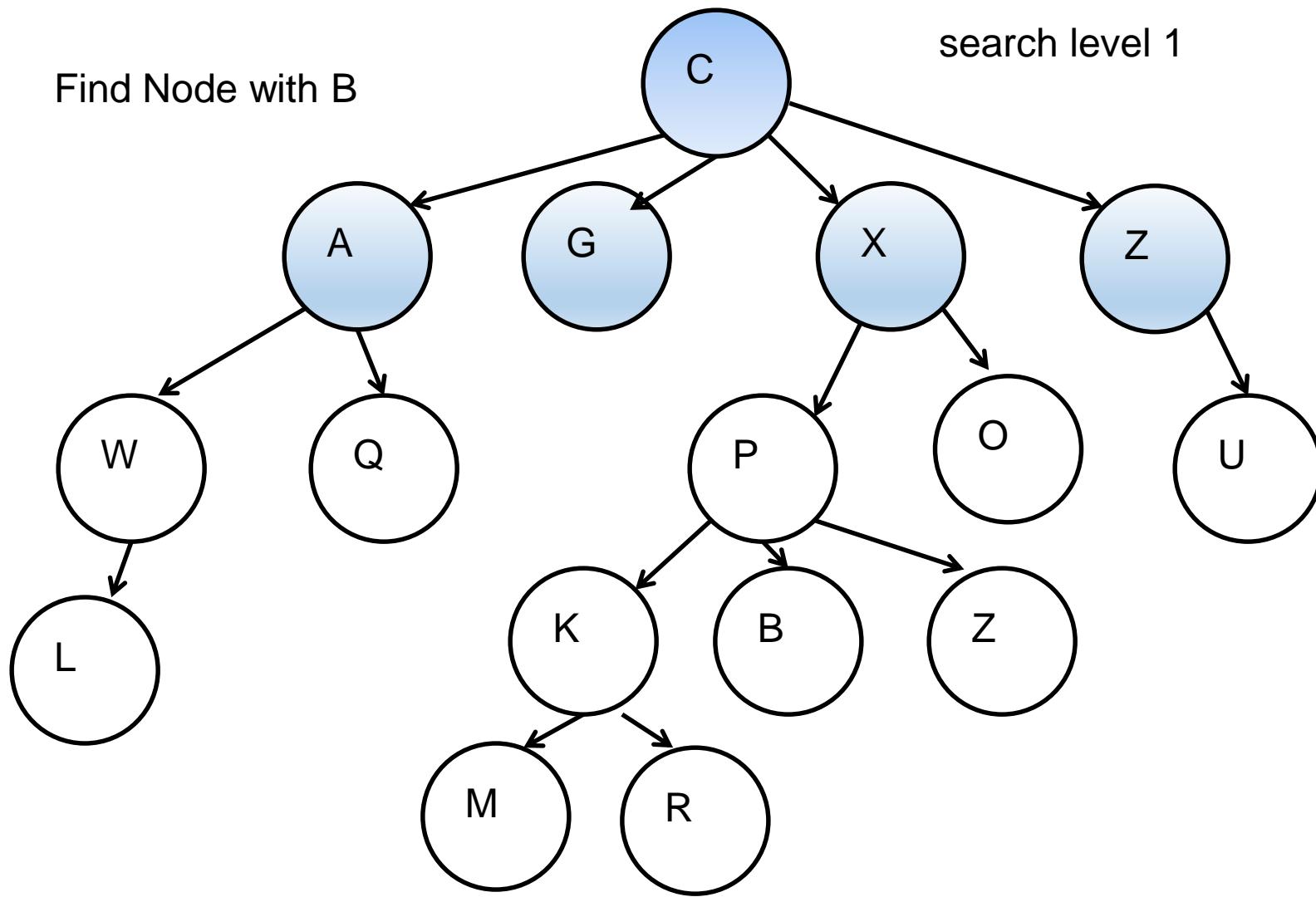
search level 1



Breadth First Search / Өргөнөөр хайх

Find Node with B

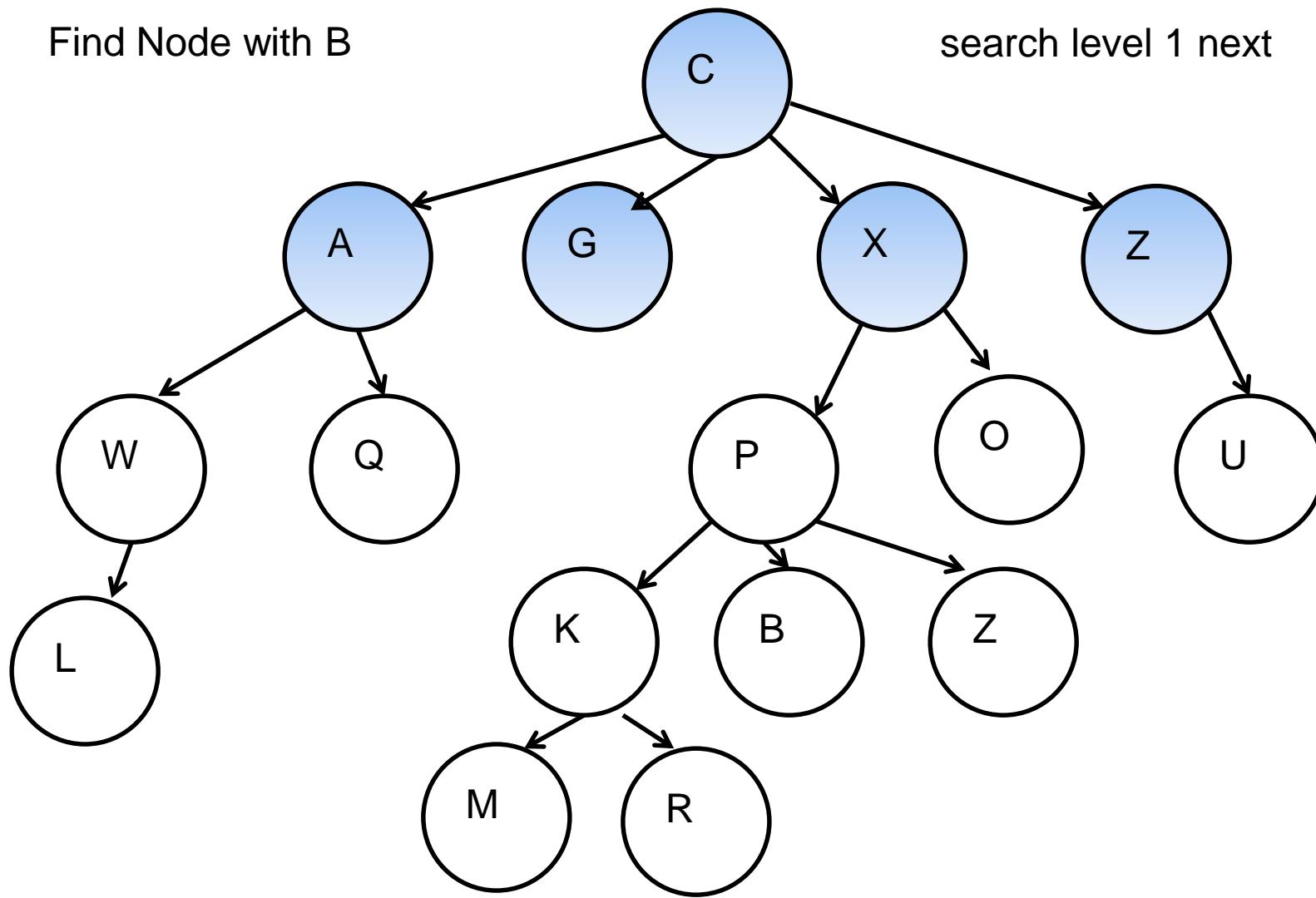
search level 1



Breadth First Search / Өргөнөөр хайх

Find Node with B

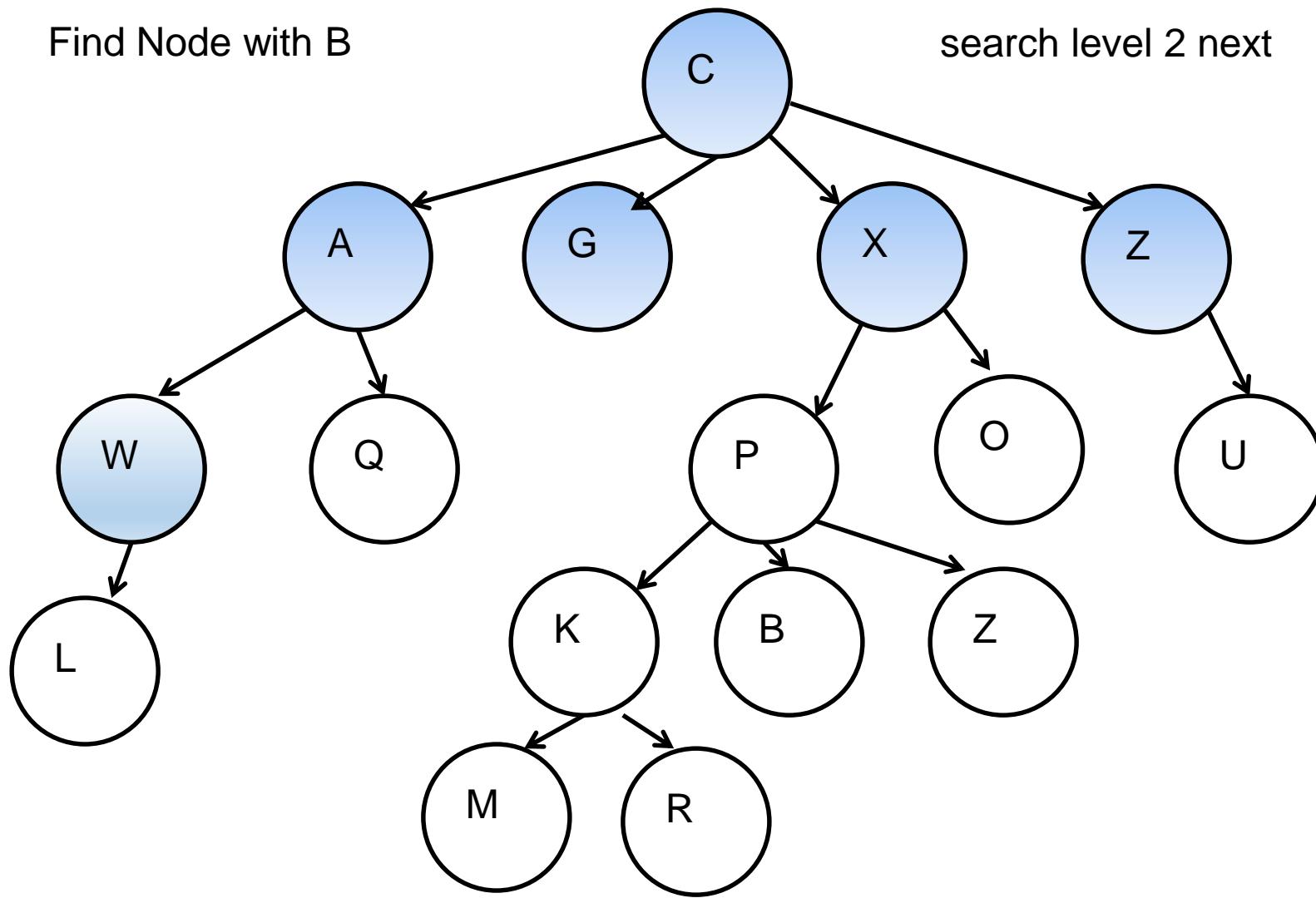
search level 1 next



Breadth First Search / Өргөнөөр хайх

Find Node with B

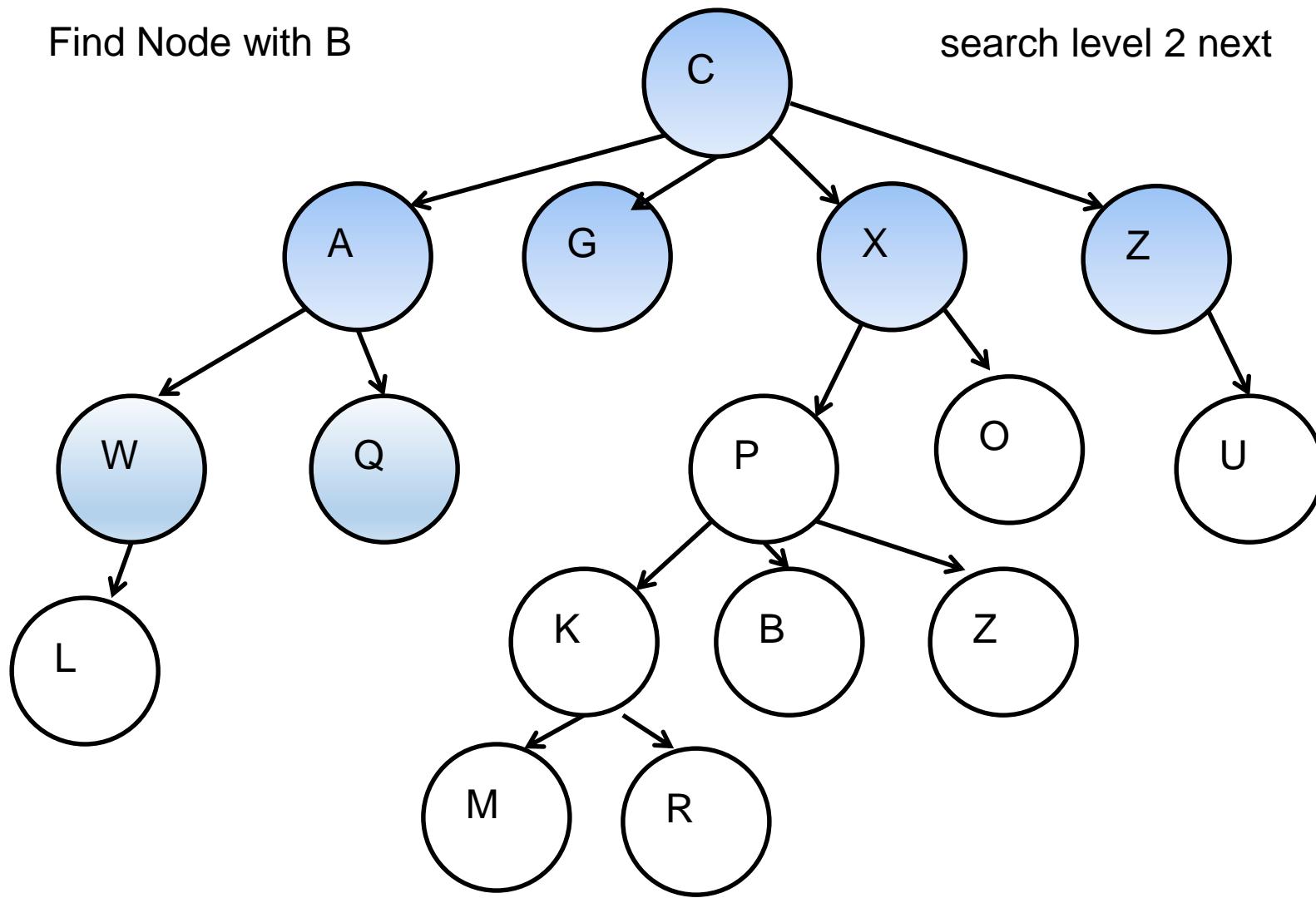
search level 2 next



Breadth First Search / Өргөнөөр хайх

Find Node with B

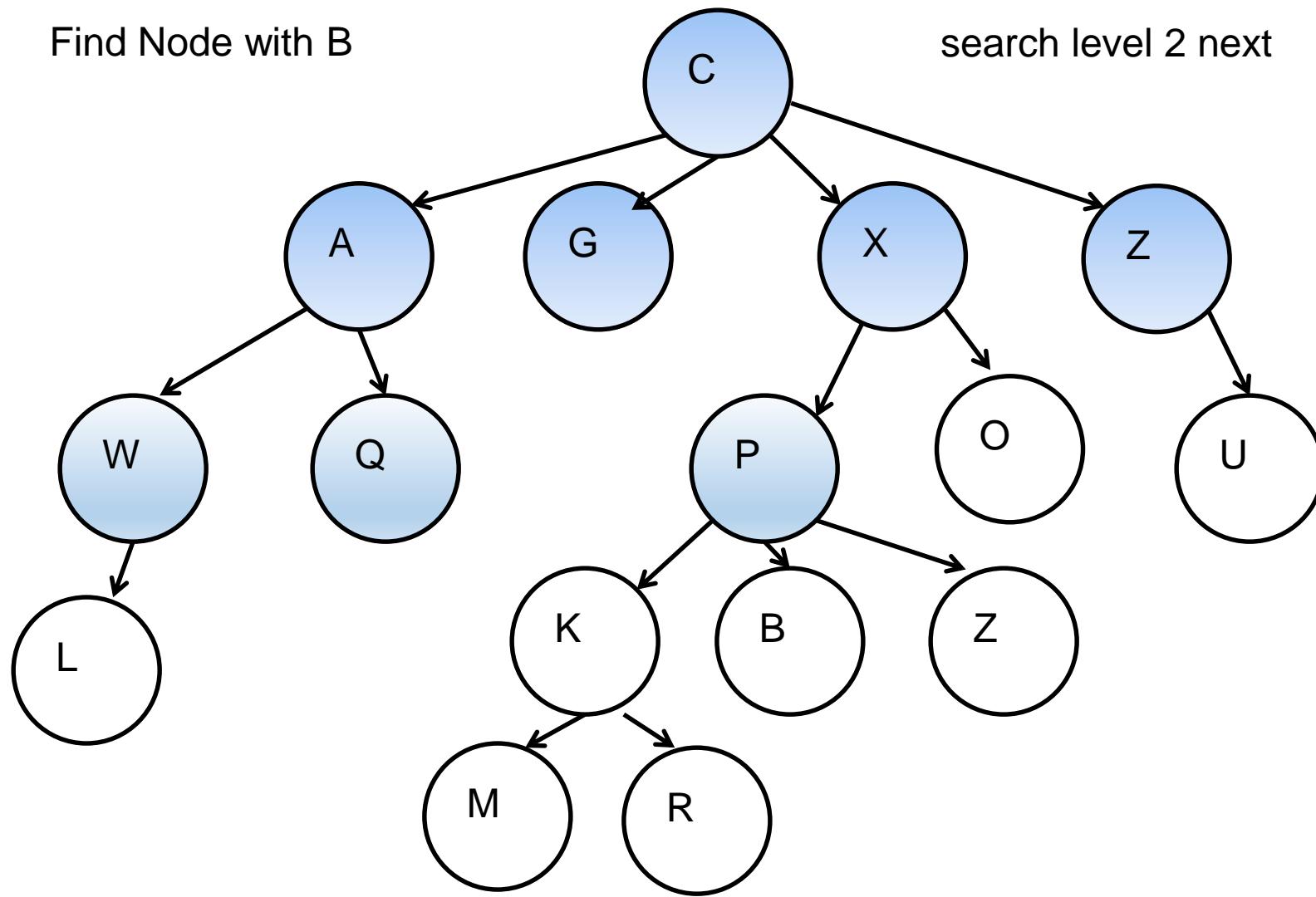
search level 2 next



Breadth First Search / Өргөнөөр хайх

Find Node with B

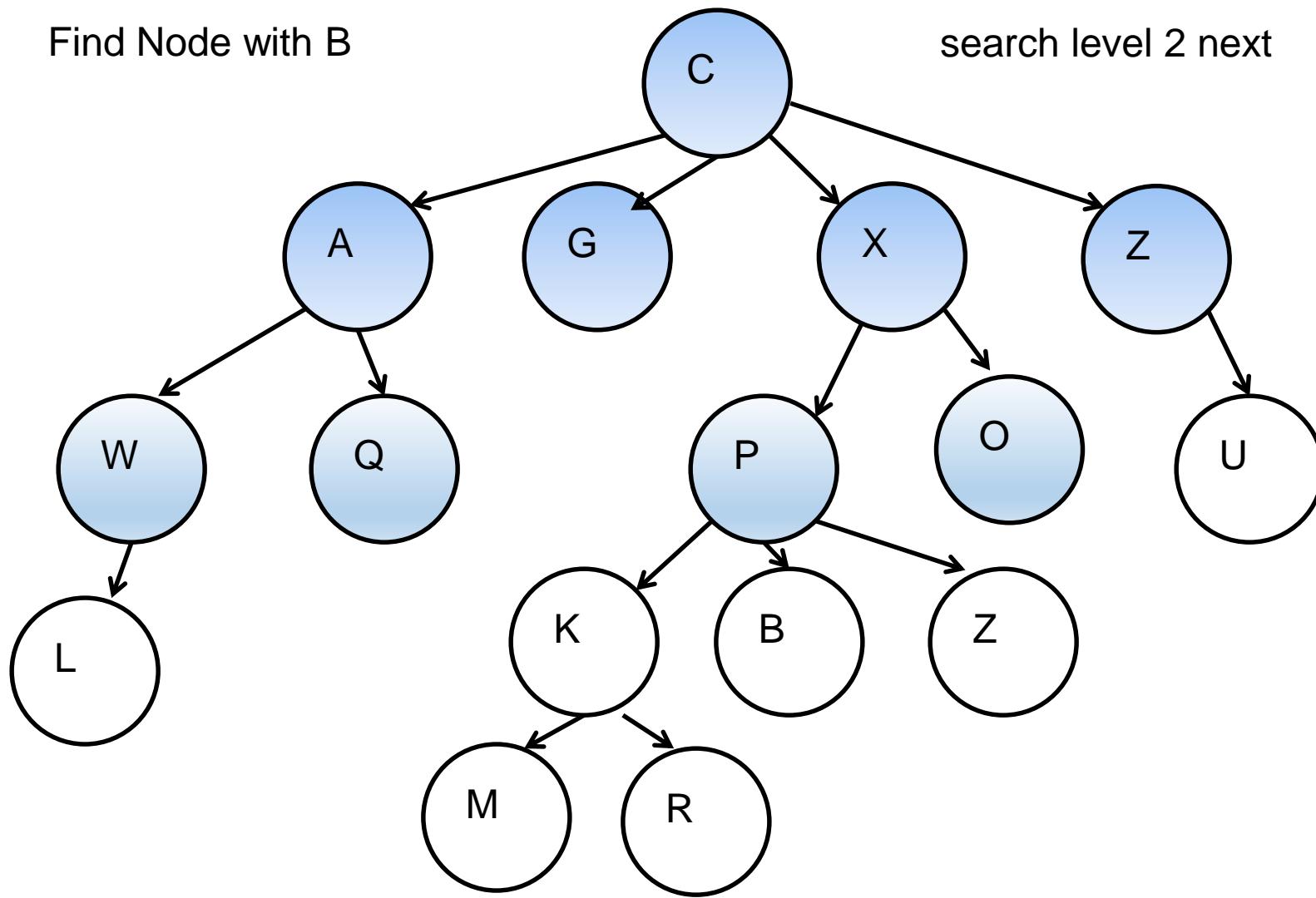
search level 2 next



Breadth First Search / Өргөнөөр хайх

Find Node with B

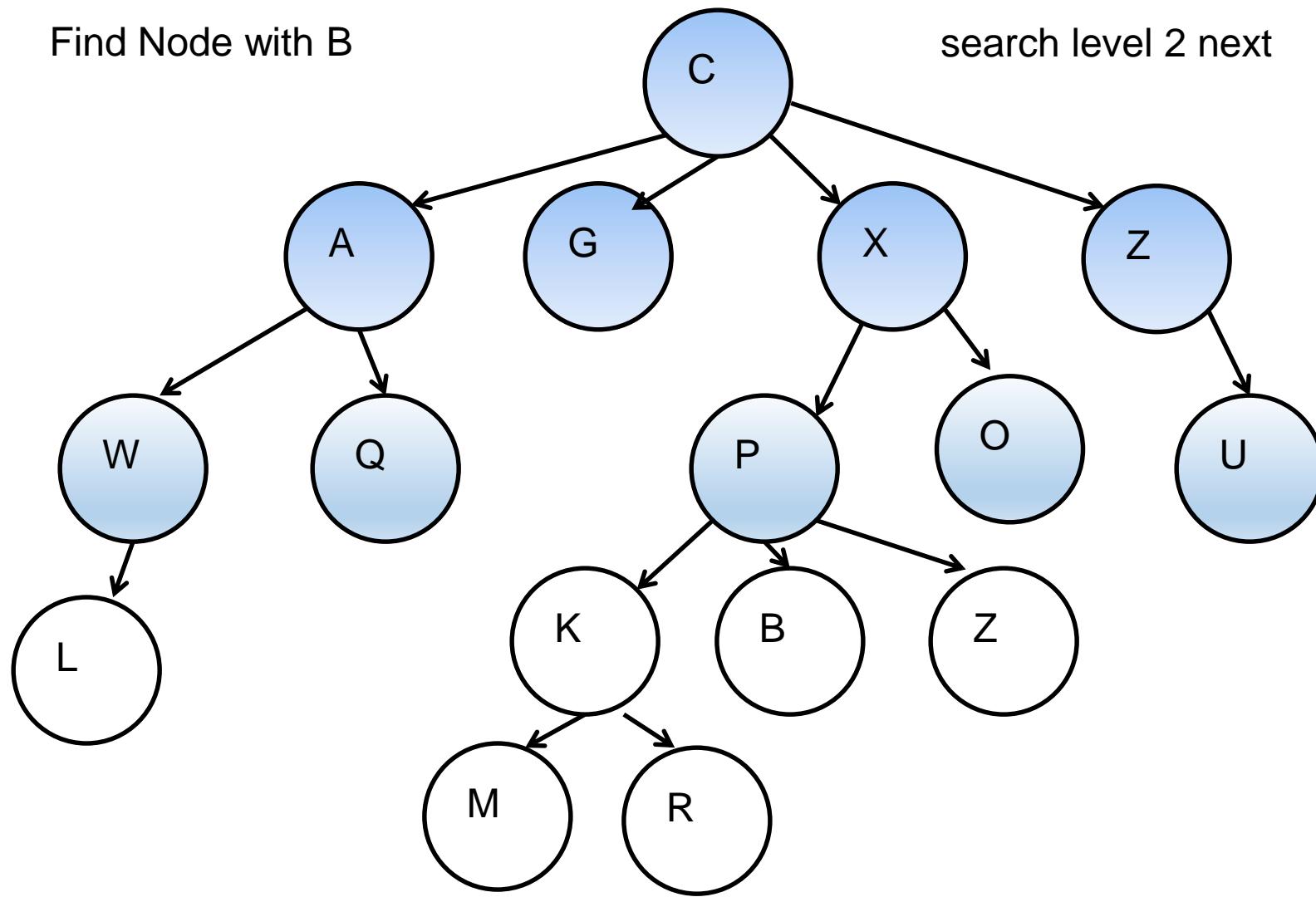
search level 2 next



Breadth First Search / Өргөнөөр хайх

Find Node with B

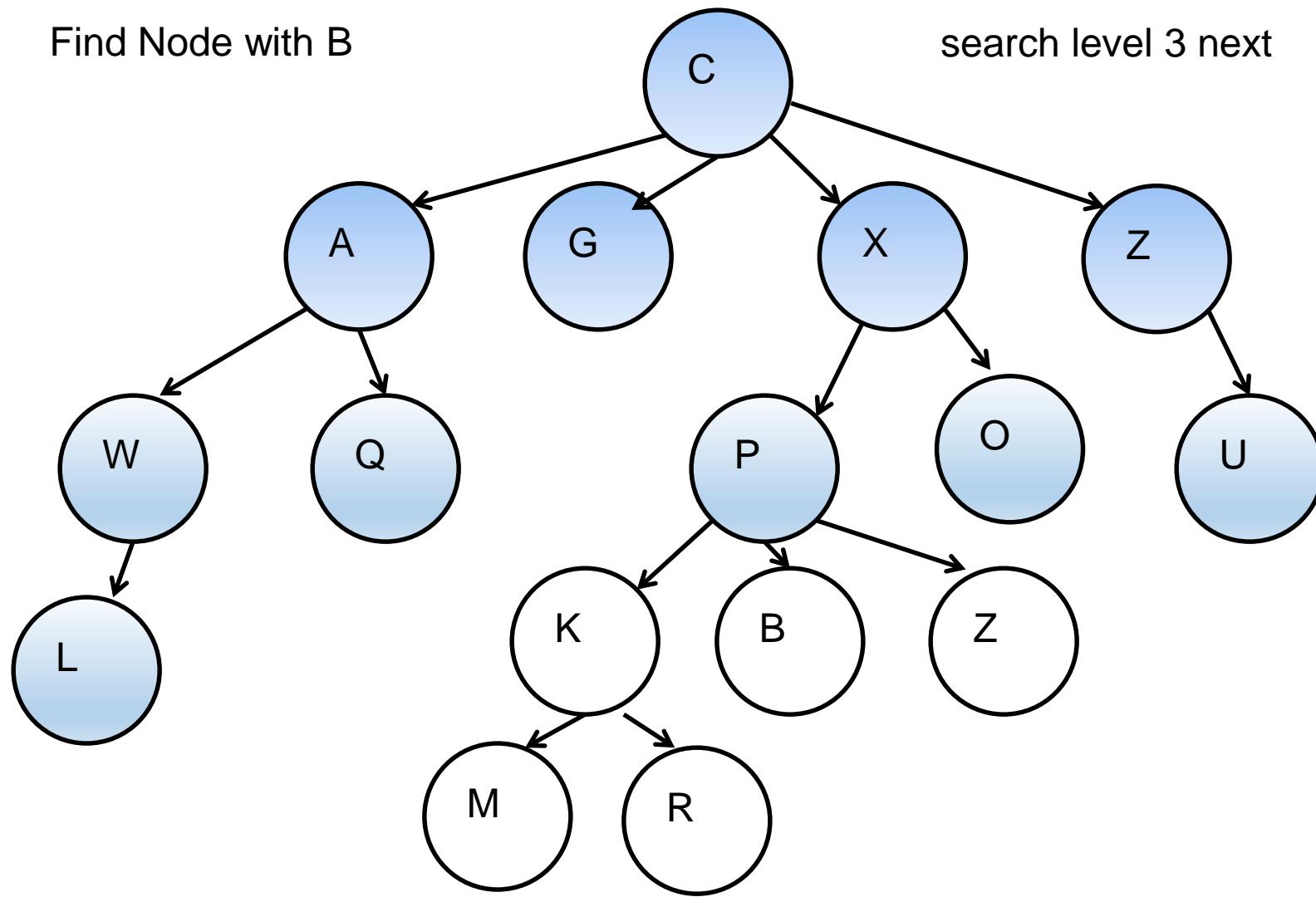
search level 2 next



Breadth First Search / Өргөнөөр хайх

Find Node with B

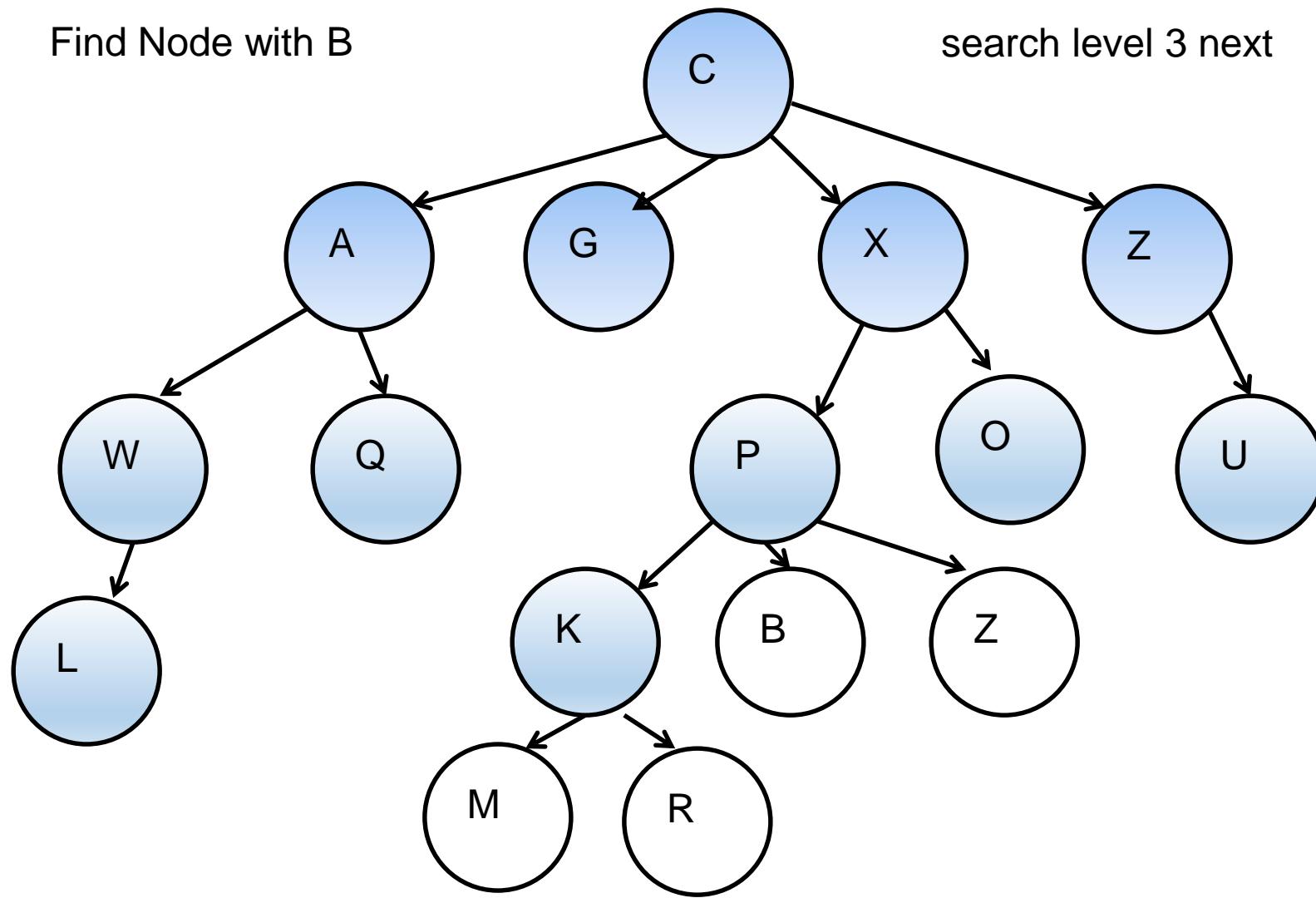
search level 3 next



Breadth First Search / Өргөнөөр хайх

Find Node with B

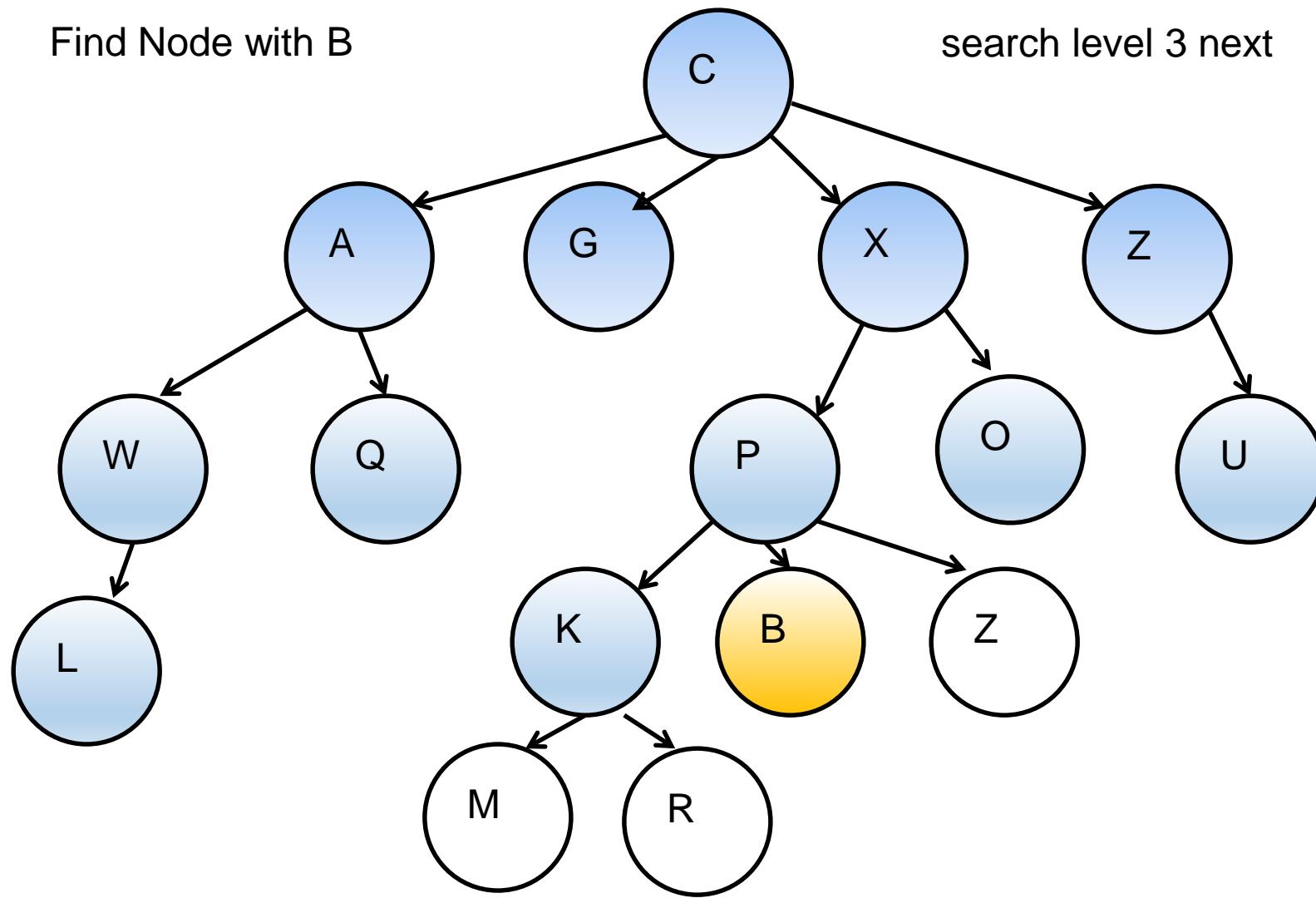
search level 3 next



Breadth First Search / Өргөнөөр хайх

Find Node with B

search level 3 next

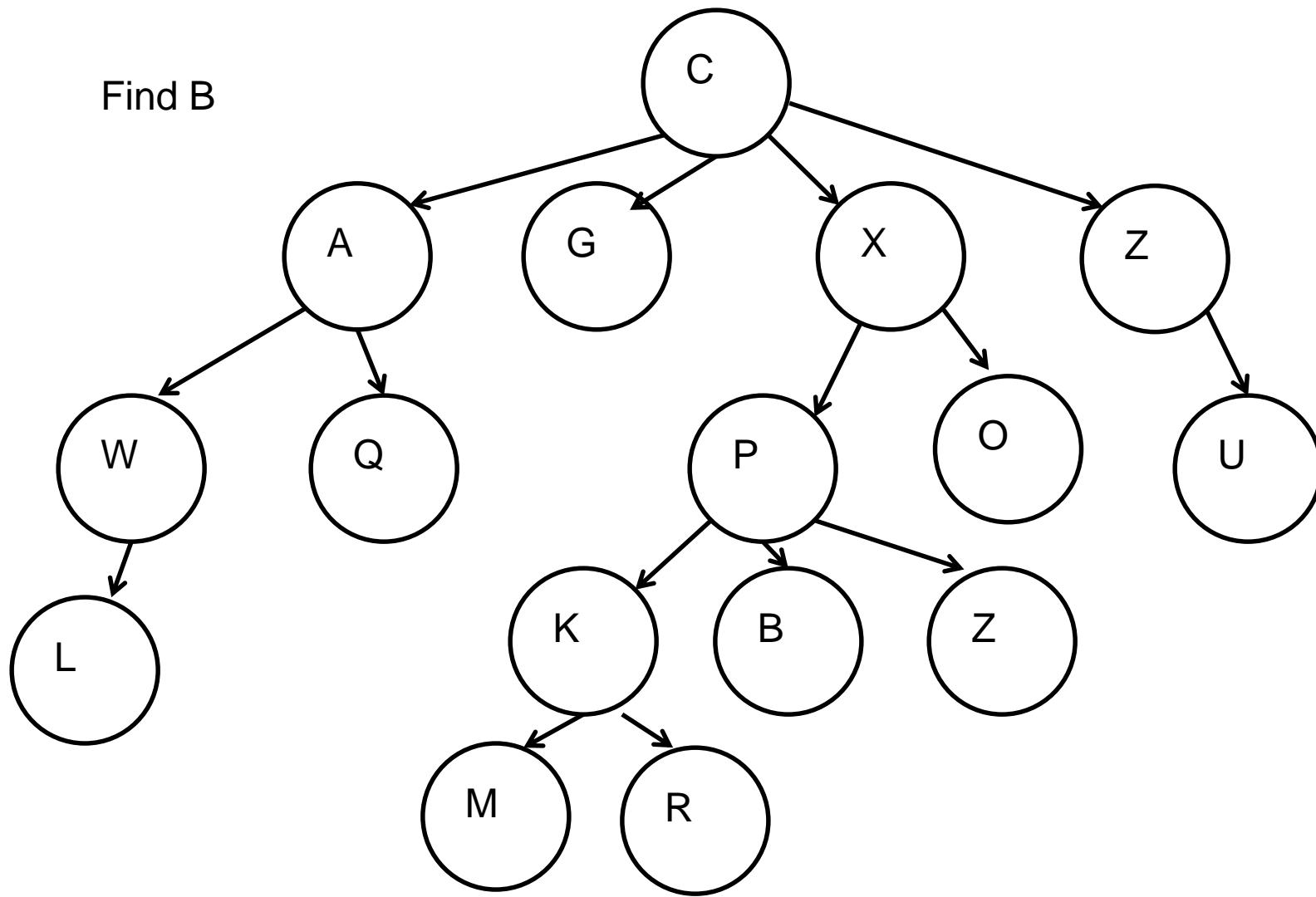


BFS - DFS

- BFS Өргөнөөр хайх хайлтыг ихэвчлэн дараалал ашиглан гүйцэтгэдэг.
- DFS Гүнээр хайх хайлтыг ихэвчлэн стекээр, далд рекурсоор эсвэл давталттайгаар тодорхой стекээр хэрэгжүүлдэг.

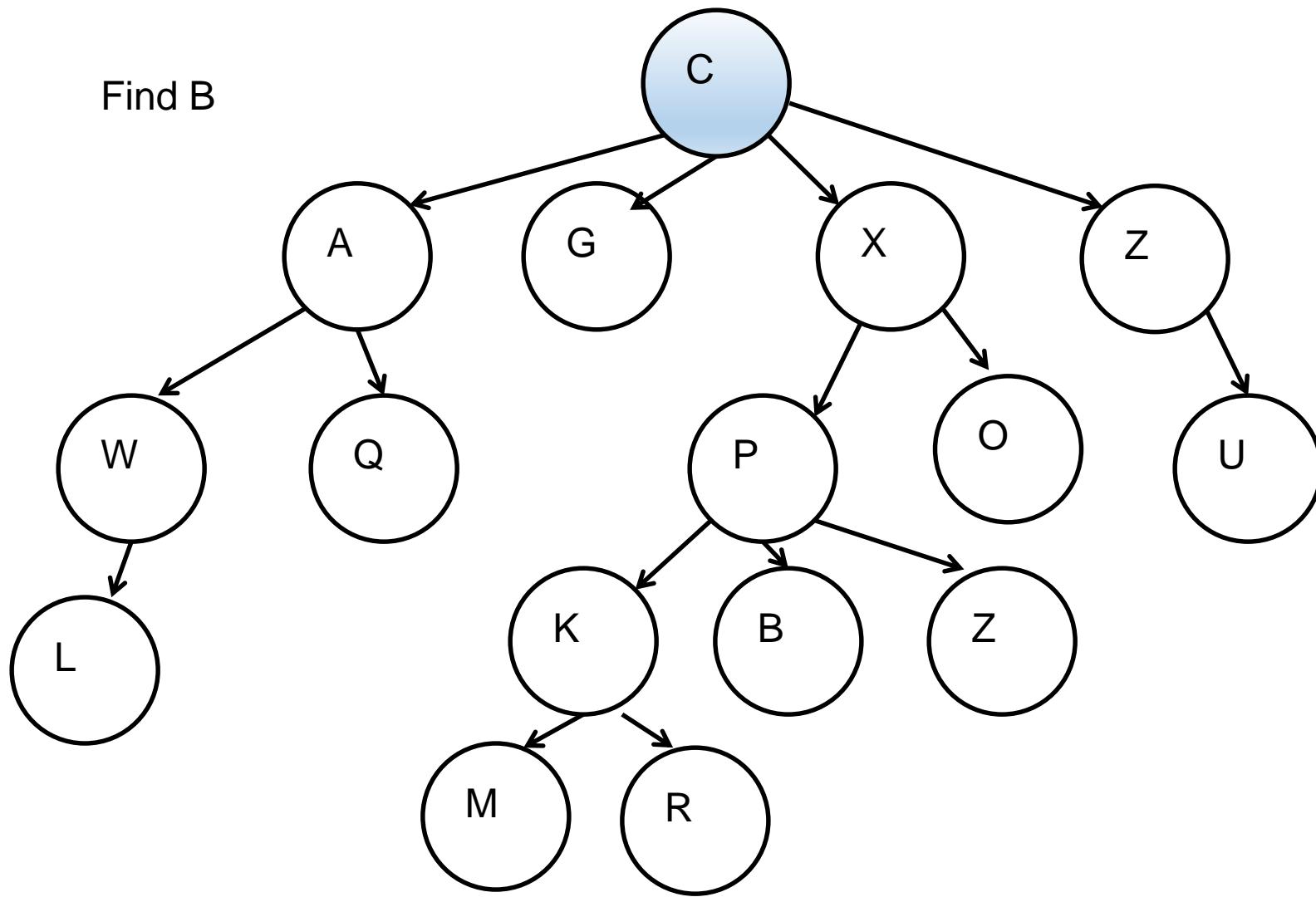
Depth First Search of Tree / Гүнээр хайх

Find B



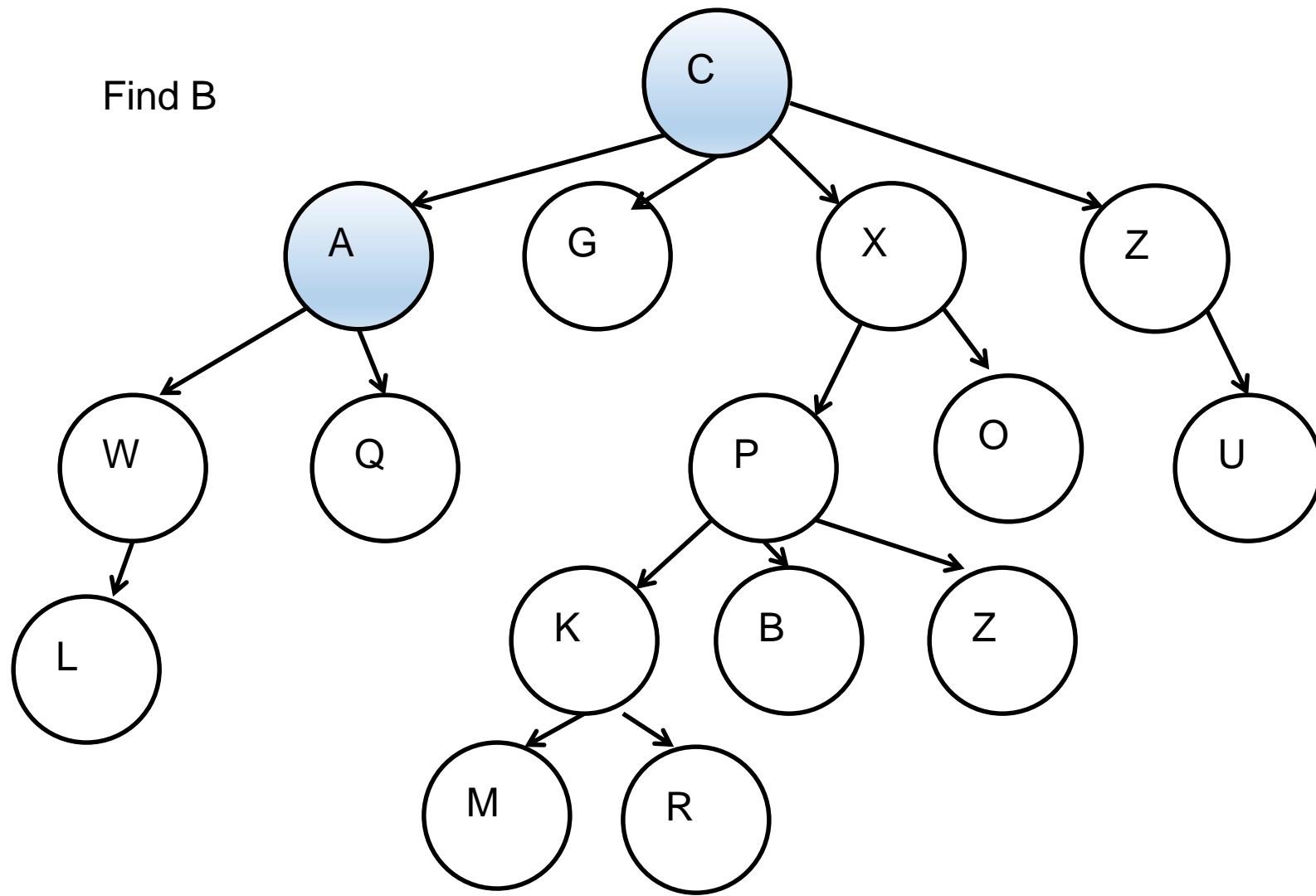
Depth First Search of Tree / Гүнээр хайх

Find B



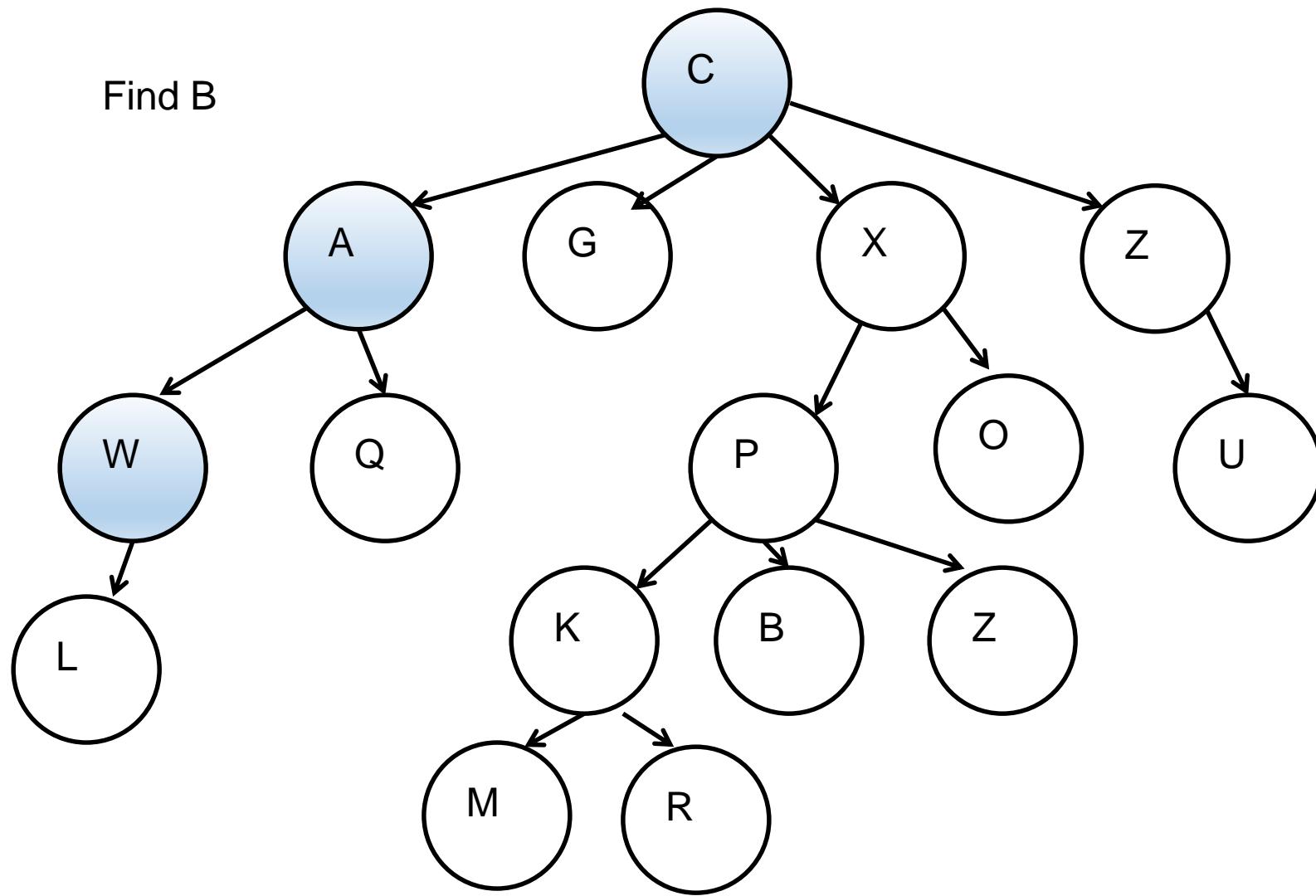
Depth First Search of Tree / Гүнээр хайх

Find B



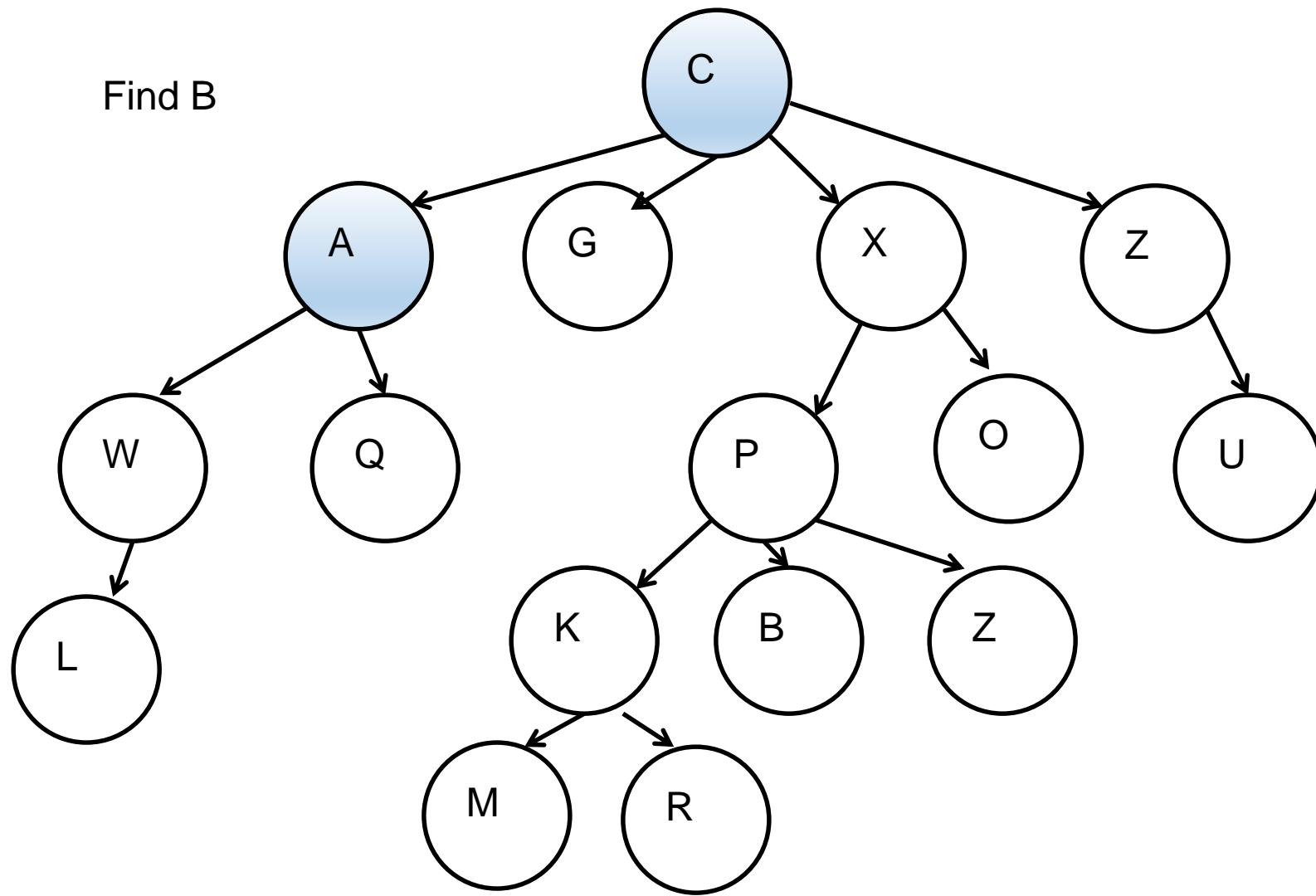
Depth First Search of Tree / Гүнээр хайх

Find B



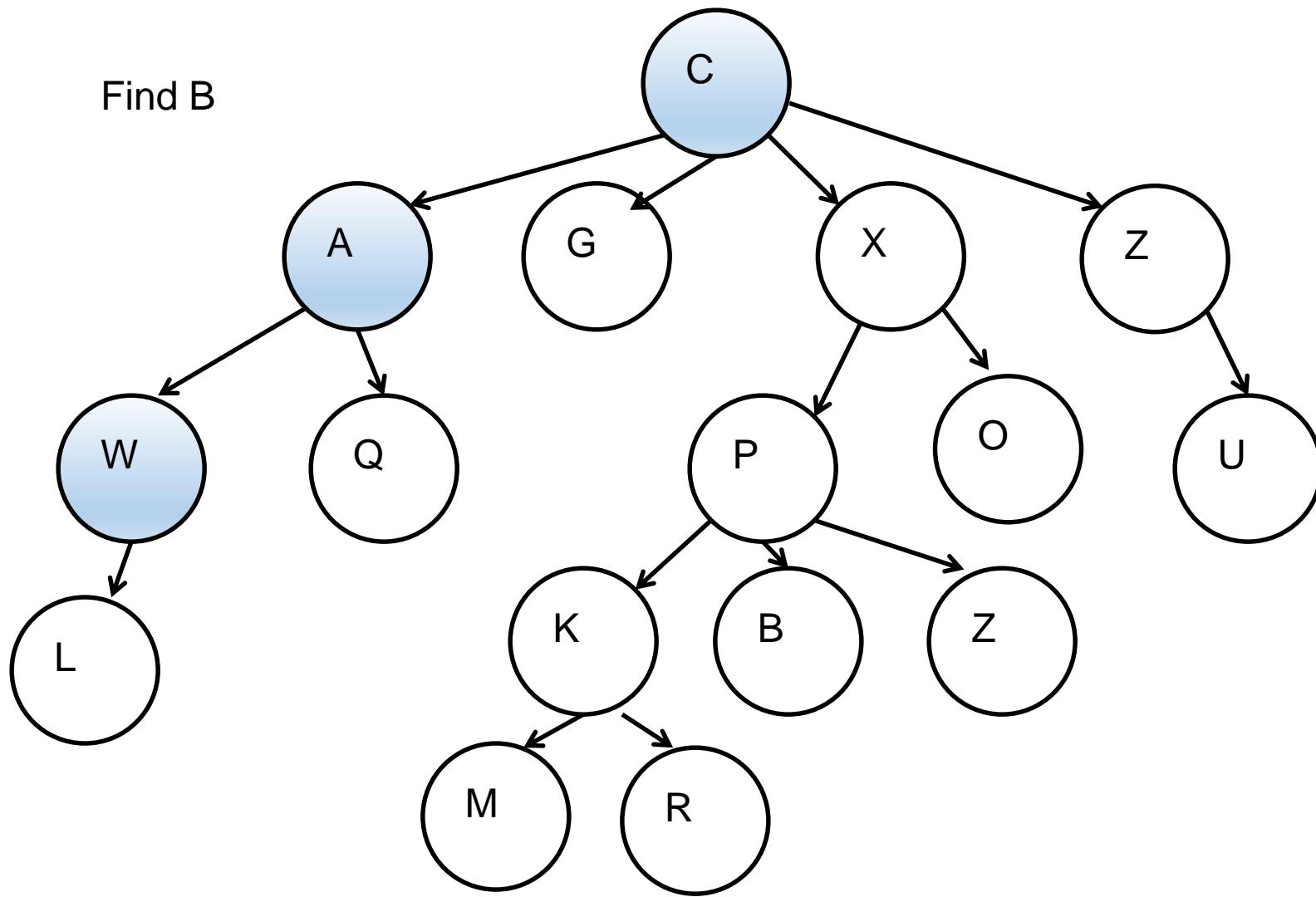
Depth First Search of Tree / Гүнээр хайх

Find B



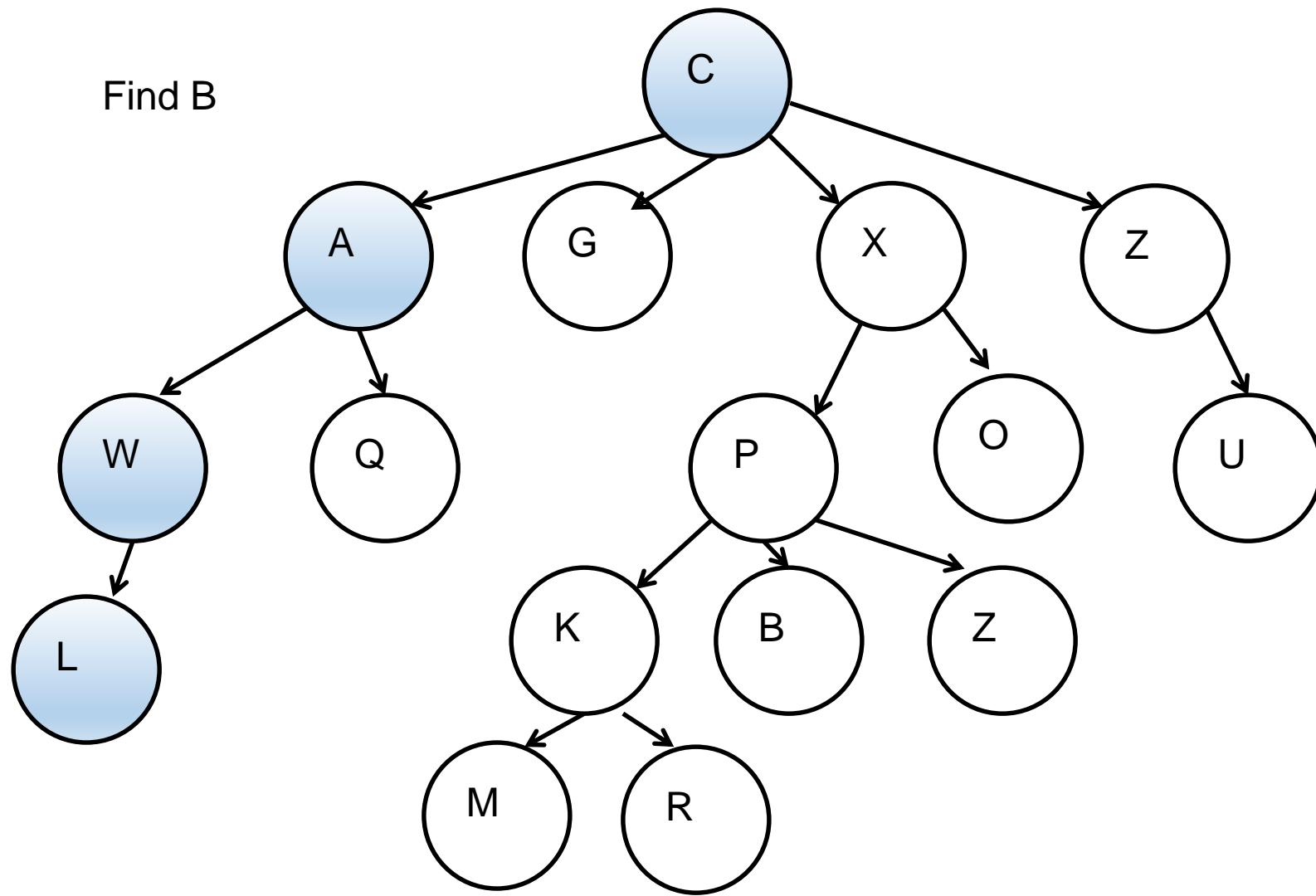
Depth First Search of Tree / Гүнээр хайх

Find B



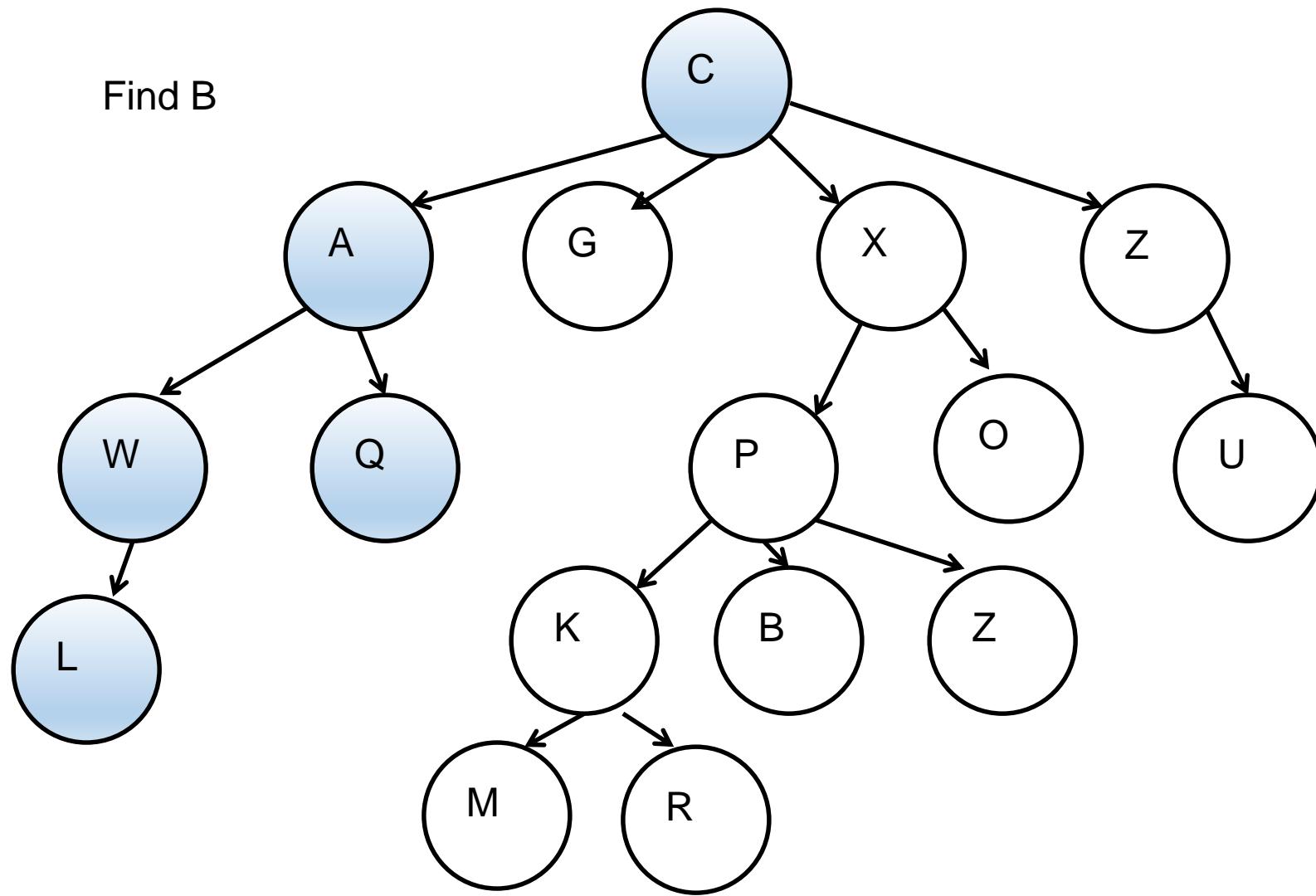
Depth First Search of Tree / Гүнээр хайх

Find B



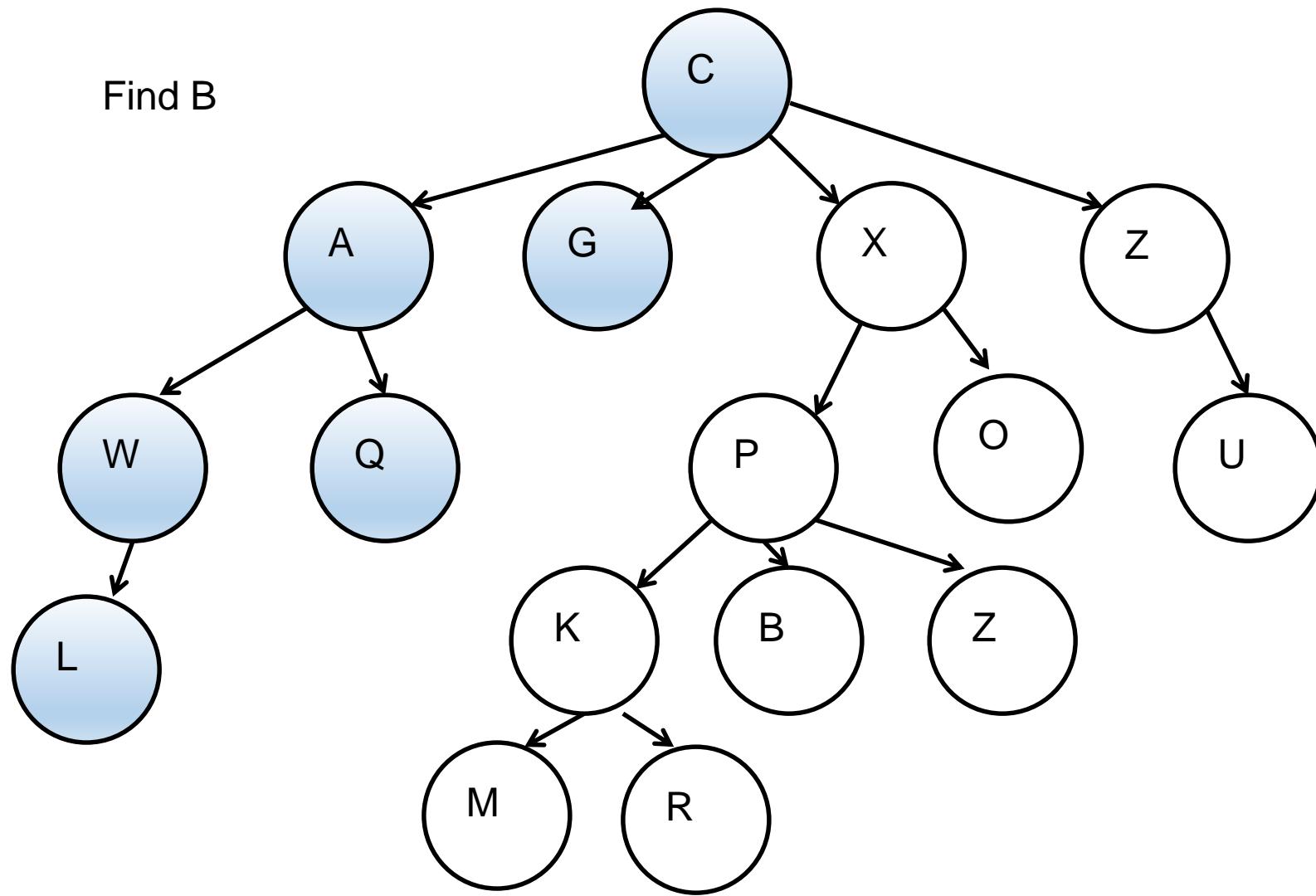
Depth First Search of Tree / Гүнээр хайх

Find B



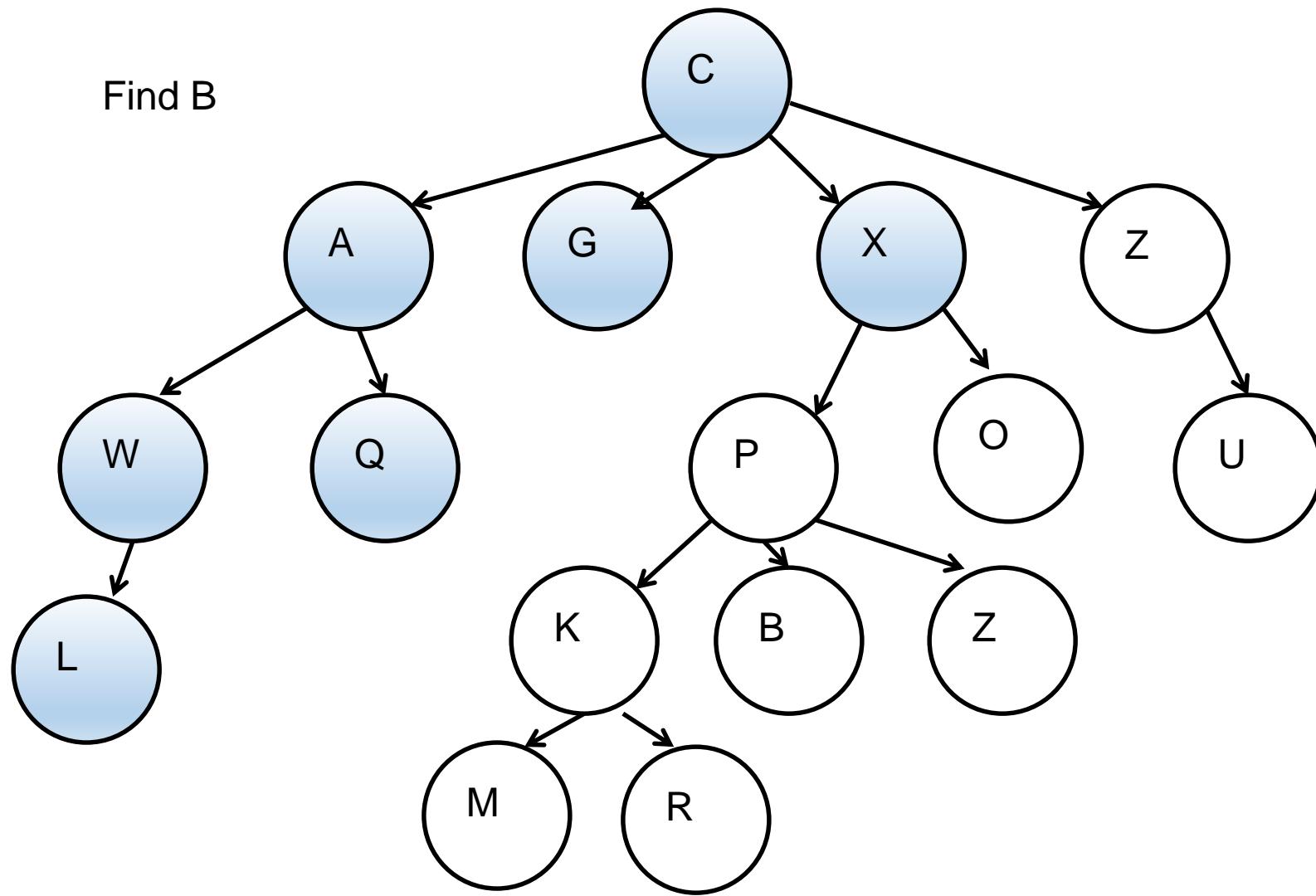
Depth First Search of Tree / Гүнээр хайх

Find B



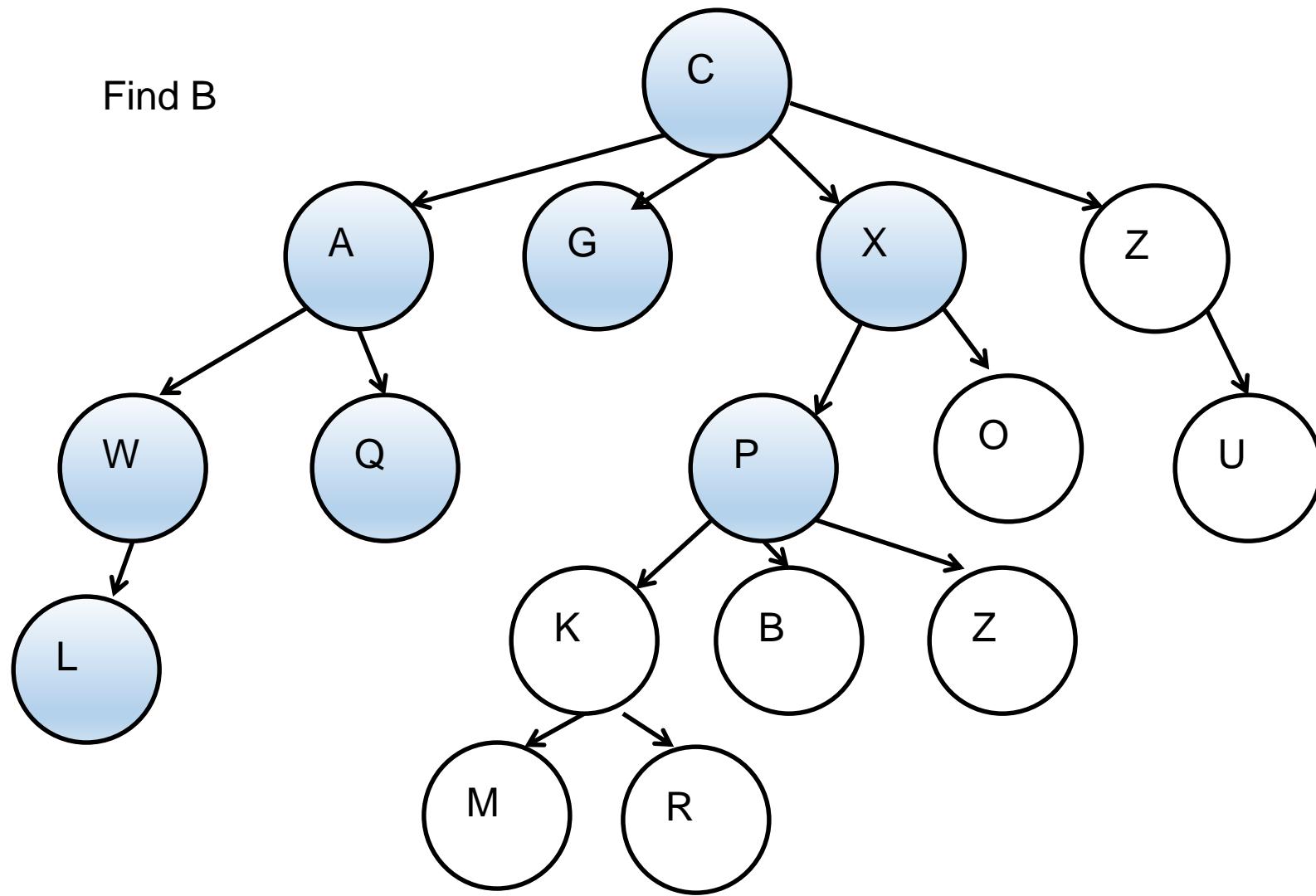
Depth First Search of Tree / Гүнээр хайх

Find B



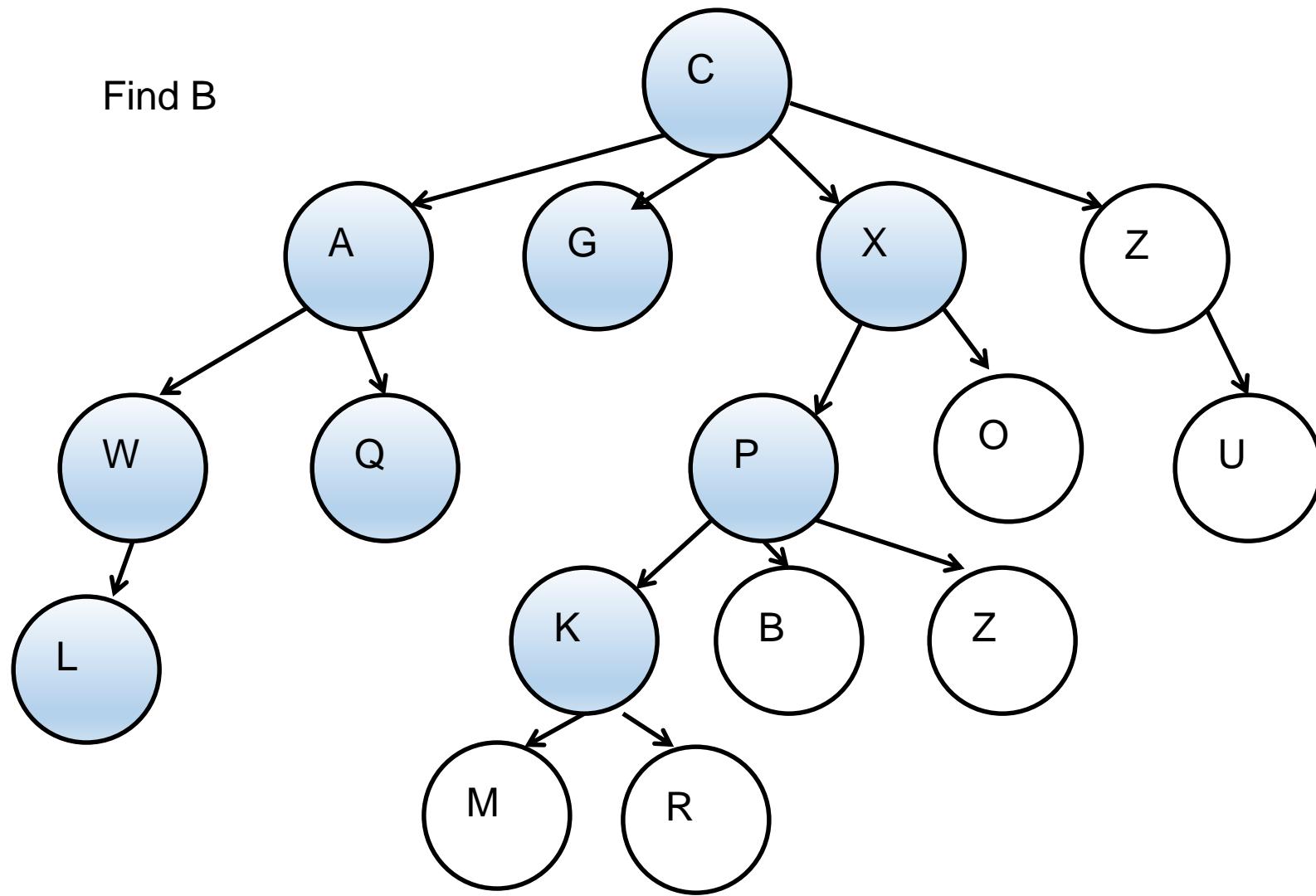
Depth First Search of Tree / Гүнээр хайх

Find B



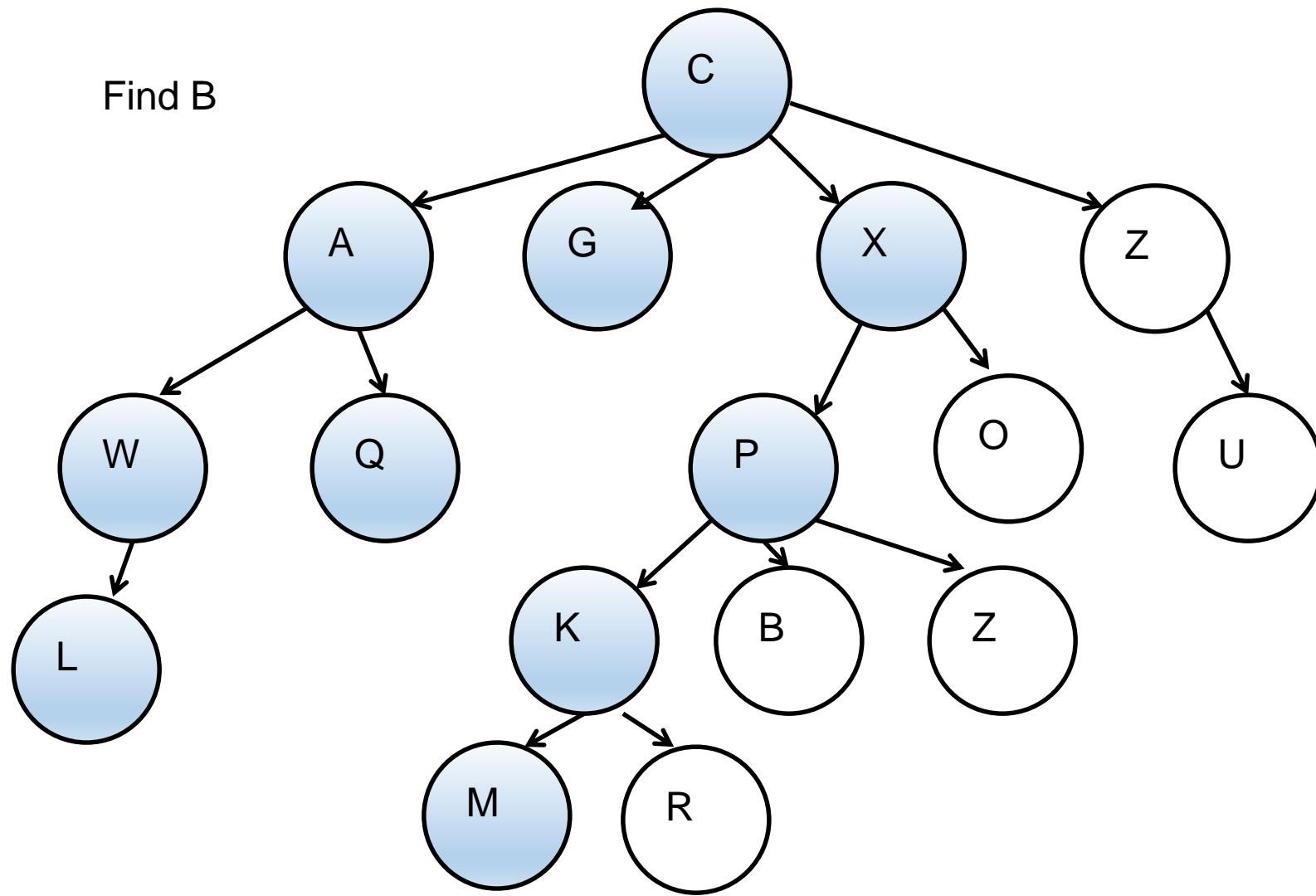
Depth First Search of Tree / Гүнээр хайх

Find B



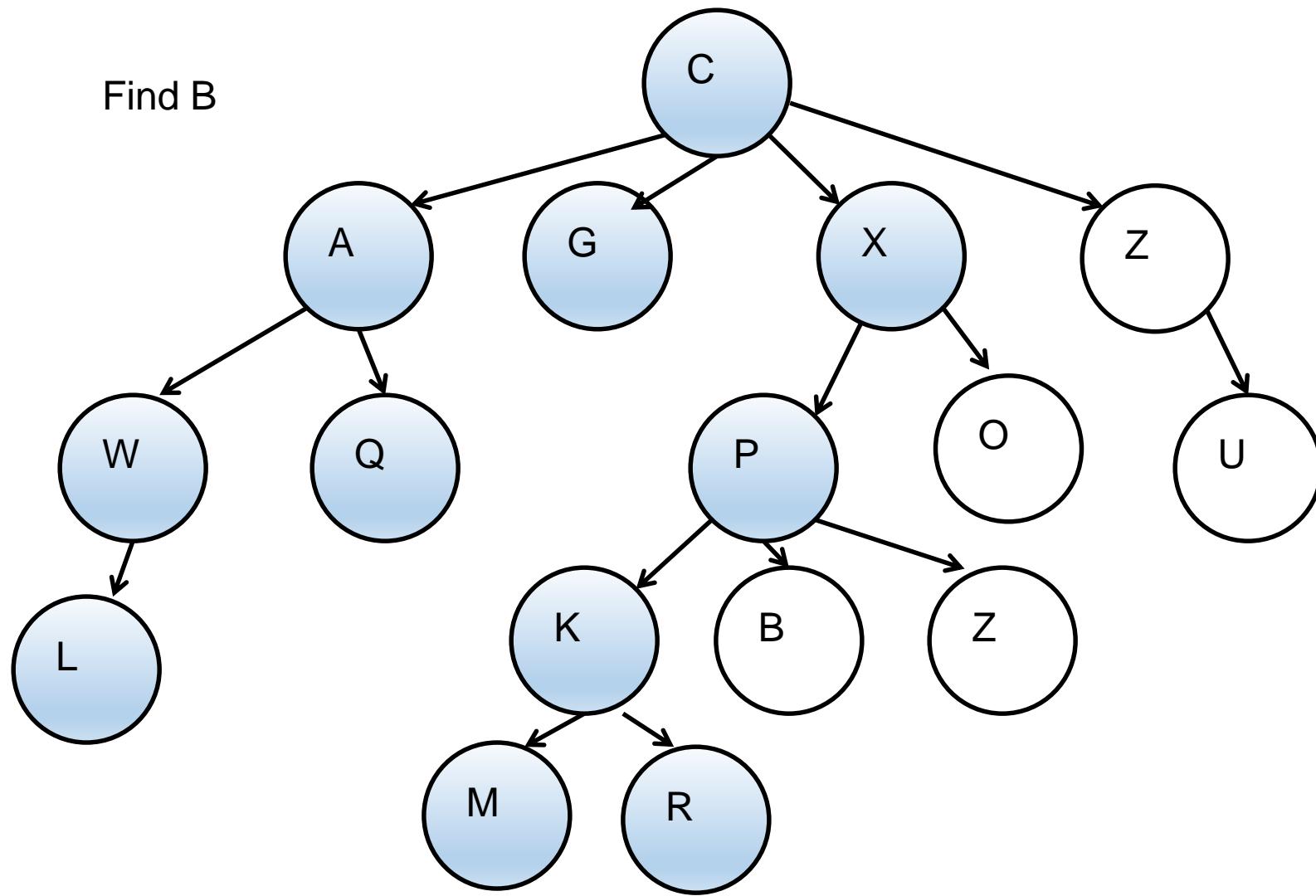
Depth First Search of Tree / Гүнээр хайх

Find B



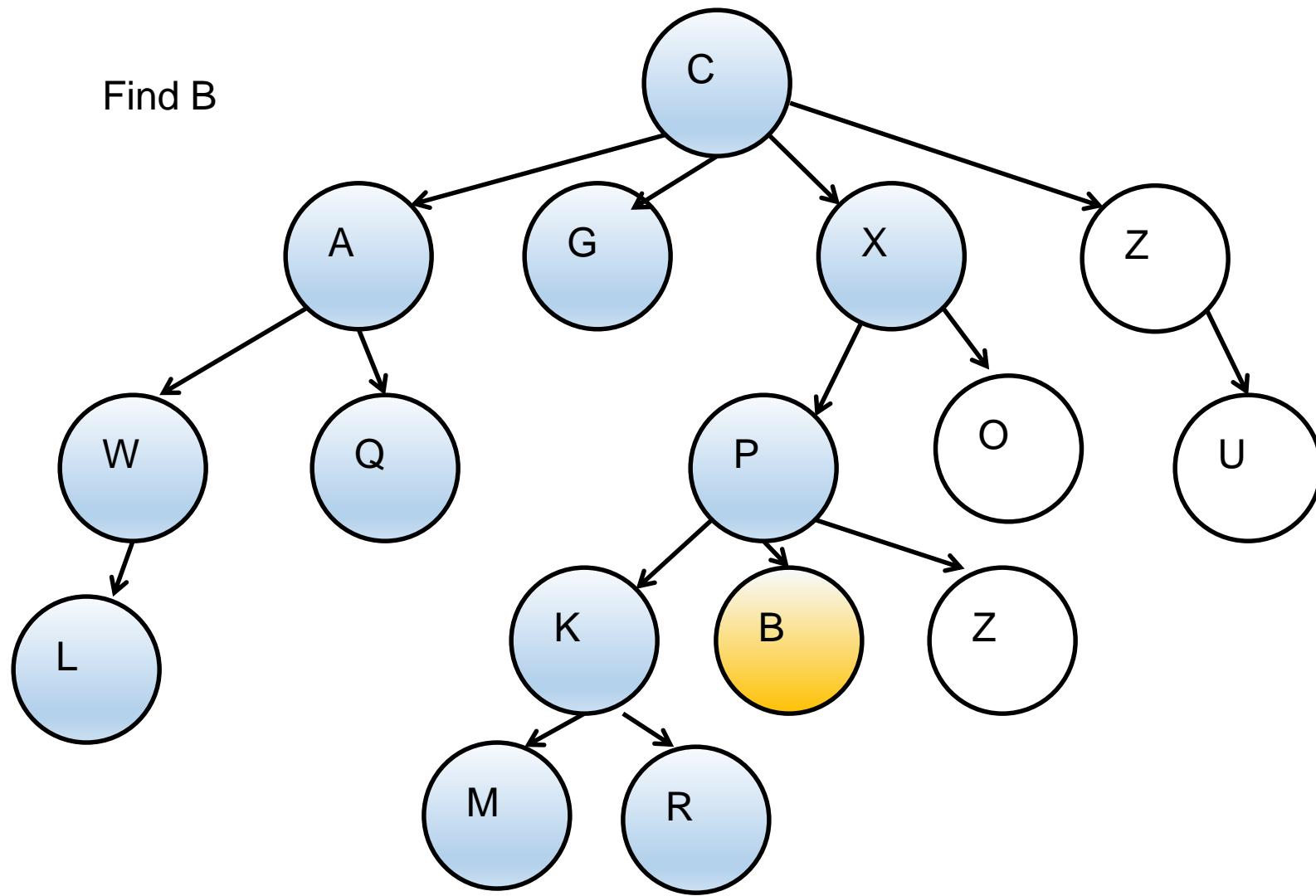
Depth First Search of Tree / Гүнээр хайх

Find B



Depth First Search of Tree / Гүнээр хайх

Find B



Binary Search Tree Complexities

- Time Complexity

Operation	Best Case Complexity	Average Case Complexity	Worst Case Complexity
Search	$O(\log n)$	$O(\log n)$	$O(n)$
Insertion	$O(\log n)$	$O(\log n)$	$O(n)$
Deletion	$O(\log n)$	$O(\log n)$	$O(n)$

Complexities of Different Operations on an AVL Tree

$\Theta(\log n)$ $O(\log n)$

AVL tree

AVL tree

Type

Tree

Invented

1962

Invented
by

Georgy Adelson-Velsky and
Evgenii Landis

Time complexity in big O notation

Algorithm

Average

Worst case

Space

$\Theta(n)$

$O(n)$

Search

$\Theta(\log n)$ ^[1]

$O(\log n)$ ^[1]

Insert

$\Theta(\log n)$ ^[1]

$O(\log n)$ ^[1]

Delete

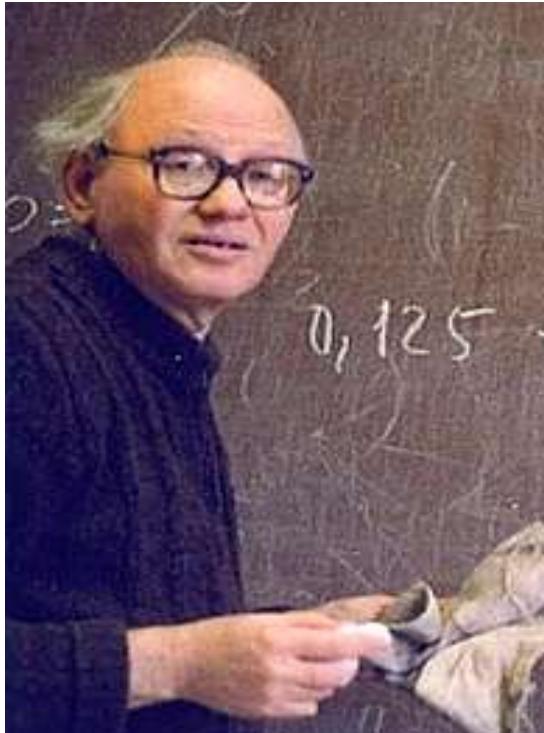
$\Theta(\log n)$ ^[1]

$O(\log n)$ ^[1]

[1] Eric Alexander. "[AVL Trees](#)". Archived from the original on July 31, 2019.

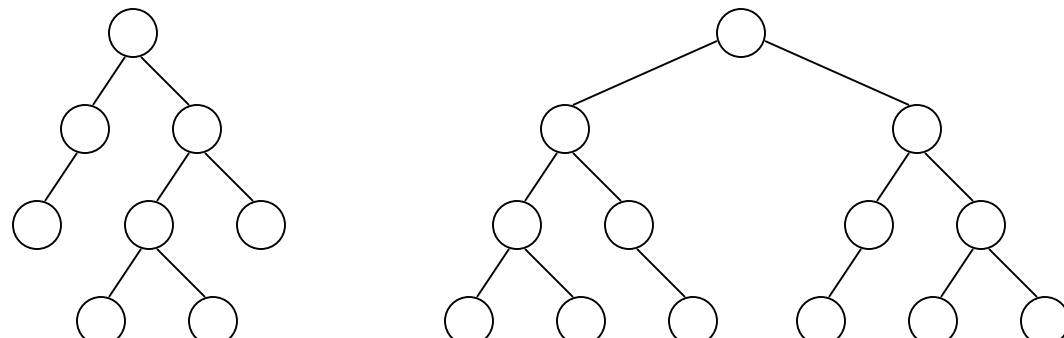
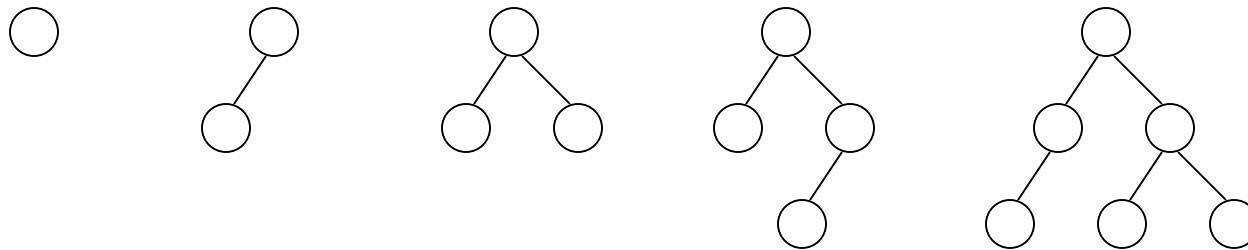
AVL Tree / Тэнцвэртэй мод

- 1962 онд Georgy **A**delson-**V**elsky ба Evgenii **L**andis нар зохион бүтээсэн.

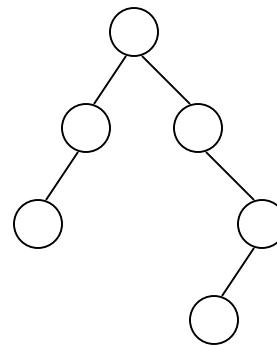
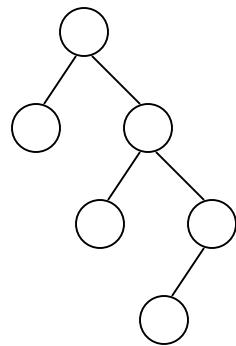
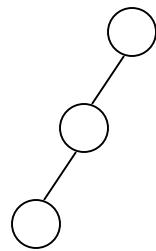


AVL / Тэнцвэртэй мод

- AVL мод (өндрийн тэнцвэржүүлсэн мод) нь хоёртын хайлтын мод бөгөөд дараах байдалтай байна. Үүнд:
 - Үндэсний зүүн ба баруун дэд модны өндөр нь хамгийн ихдээ 1-ээр ялгаатай байна.
 - Үндэсний зүүн ба баруун дэд мод нь AVL мод юм.



AVL Мод биш



Balance Factor / Тэнцвэрийн хүчин зүйл

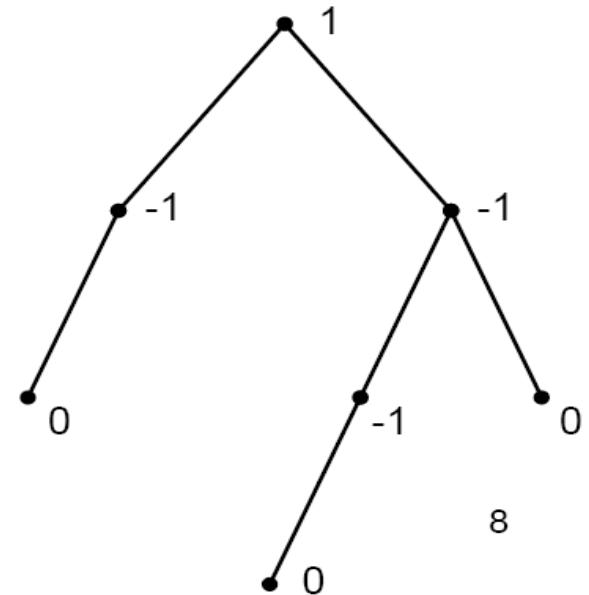
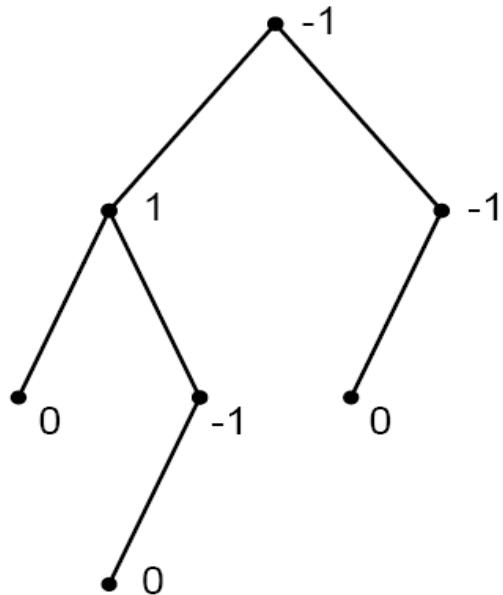
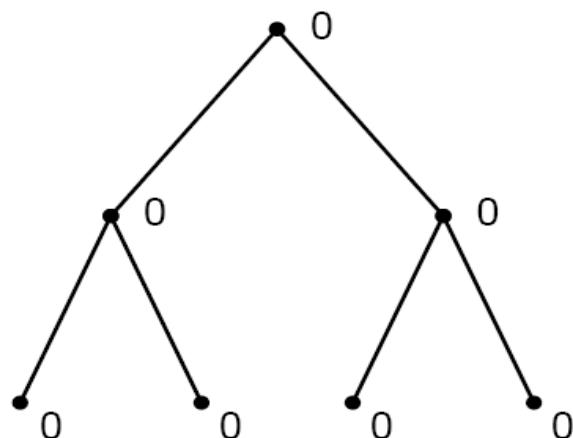
- Хоёртын хайлтын мод нь AVL мод мөн эсэхийг хянахын тулд зангилаа бүрд тэнцвэрийн хүчин зүйлийг шалгана.
- Өндөр (баруун дэд мод) – Өндөр (зүүн дэд мод)

Height(right subtree) – Height(left subtree)

- Үр дүн: **Worst-case** гүнээр хайх
 - Binary tree property (same as BST)
 - Order property (same as for BST)

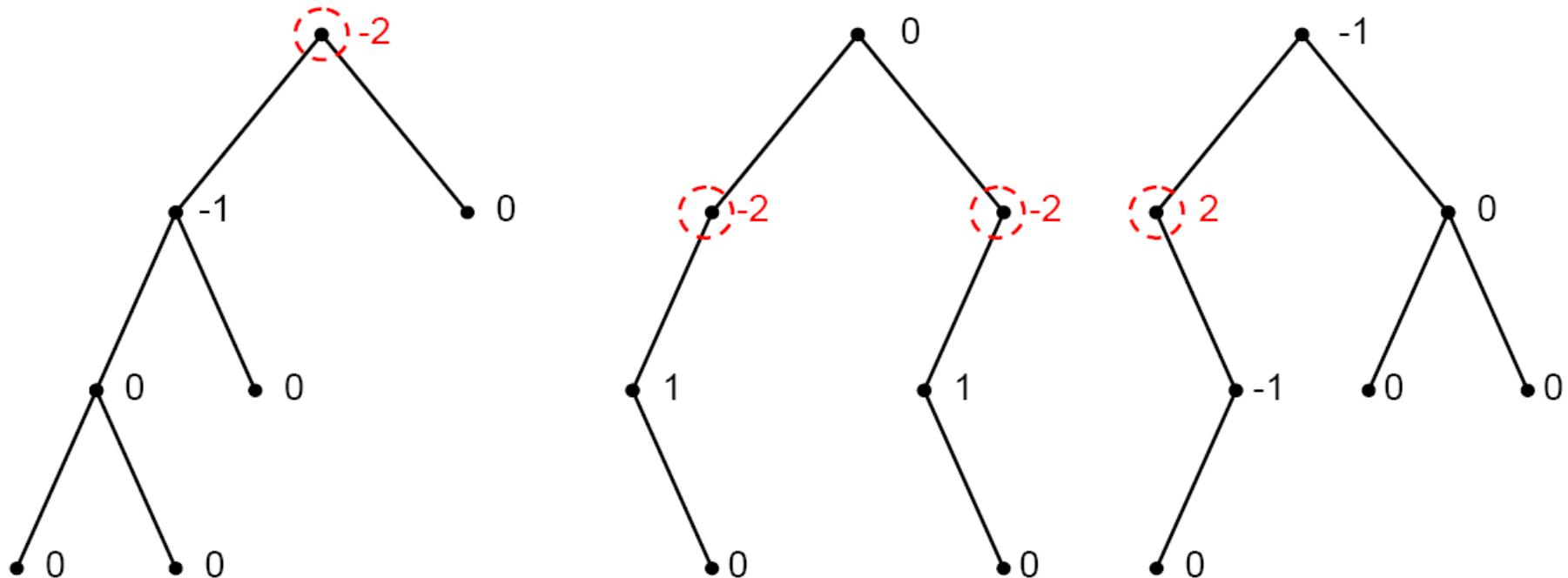
AVL Мод

- Өндөр (баруун дэд мод) – Өндөр (зүүн дэд мод)



AVL Мод биш

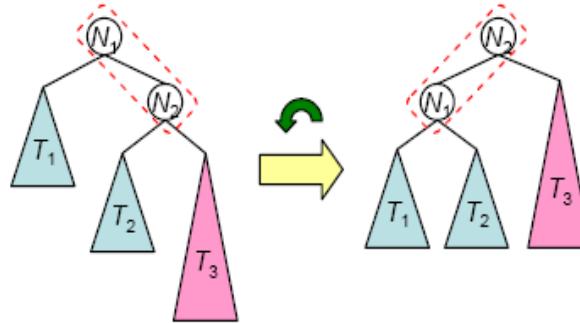
- Өндөр (баруун дэд мод) – Өндөр (зүүн дэд мод)



Imbalance / Тэнцвэргүй байдал

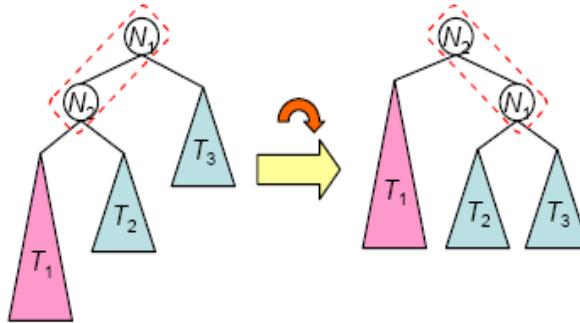
Case 1: insertion to *right* subtree of *right* child

Solution: *Left* rotation



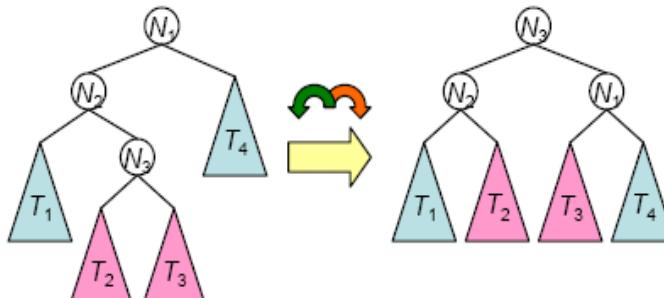
Case 2: insertion to *left* subtree of *left* child

Solution: *Right* rotation



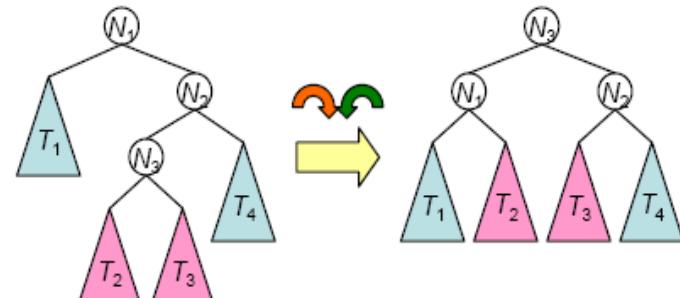
Case 3: insertion to *right* subtree of *left* child

Solution: *Left-right* rotation



Case 4: insertion to *left* subtree of *right* child

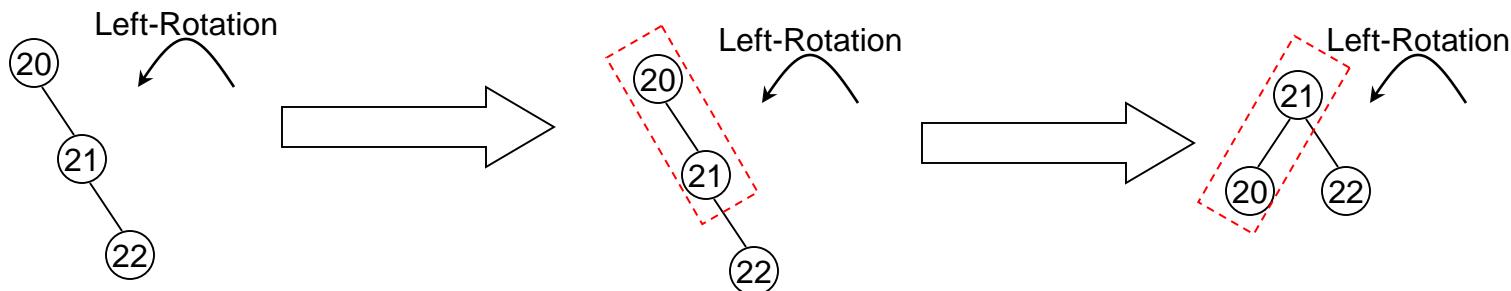
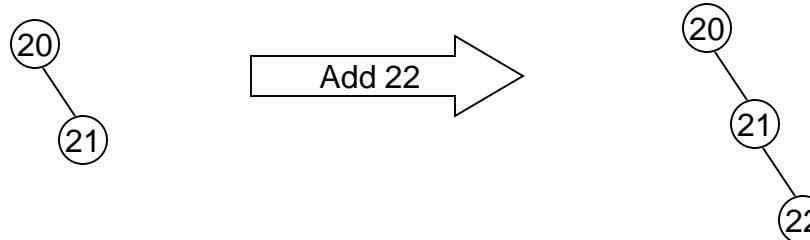
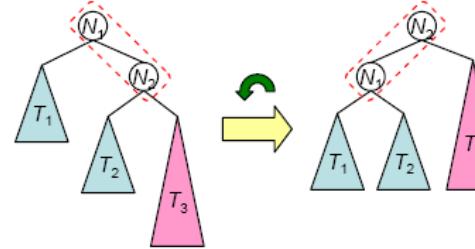
Solution: *Right-left* rotation



Left-Rotation Зүүн-Эргүүлэлт

Case 1: insertion to *right* subtree of *right* child

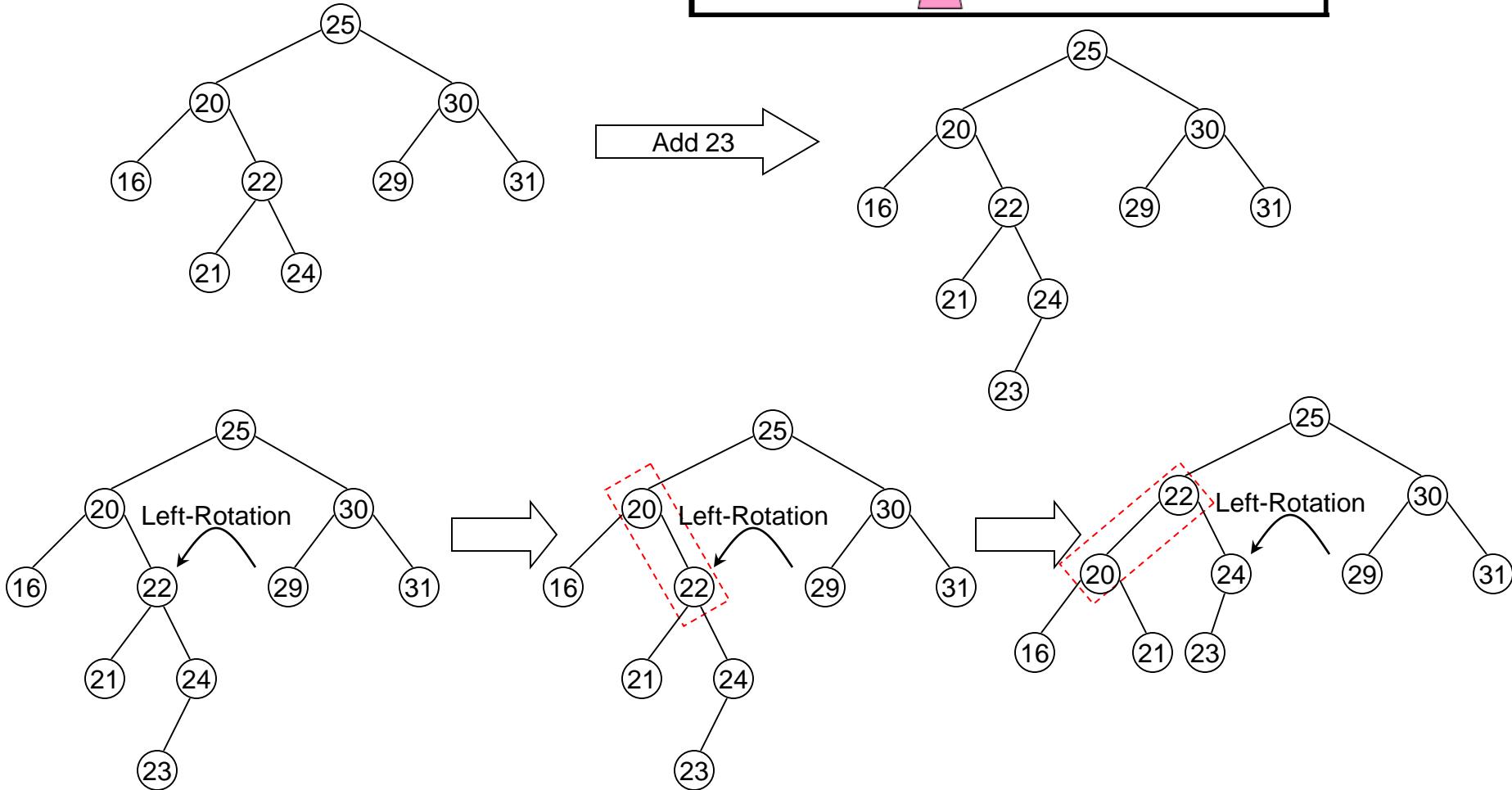
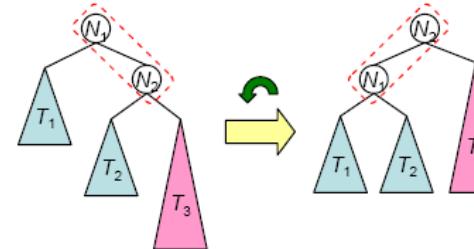
Solution: *Left* rotation



Left-Rotation Зүүн-Эргүүлэлт

Case 1: insertion to *right* subtree of *right* child

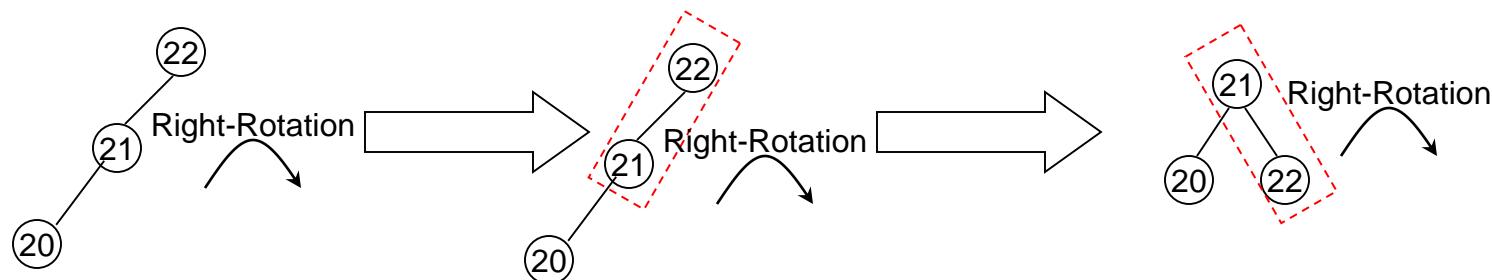
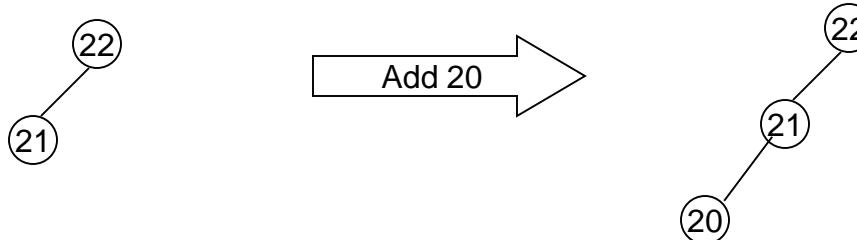
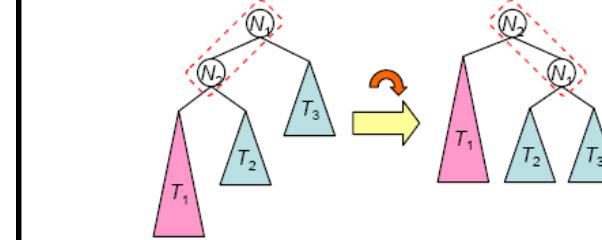
Solution: *Left* rotation



Right-Rotation Баруун-Эргүүлэлт

Case 2: insertion to *left* subtree of *left* child

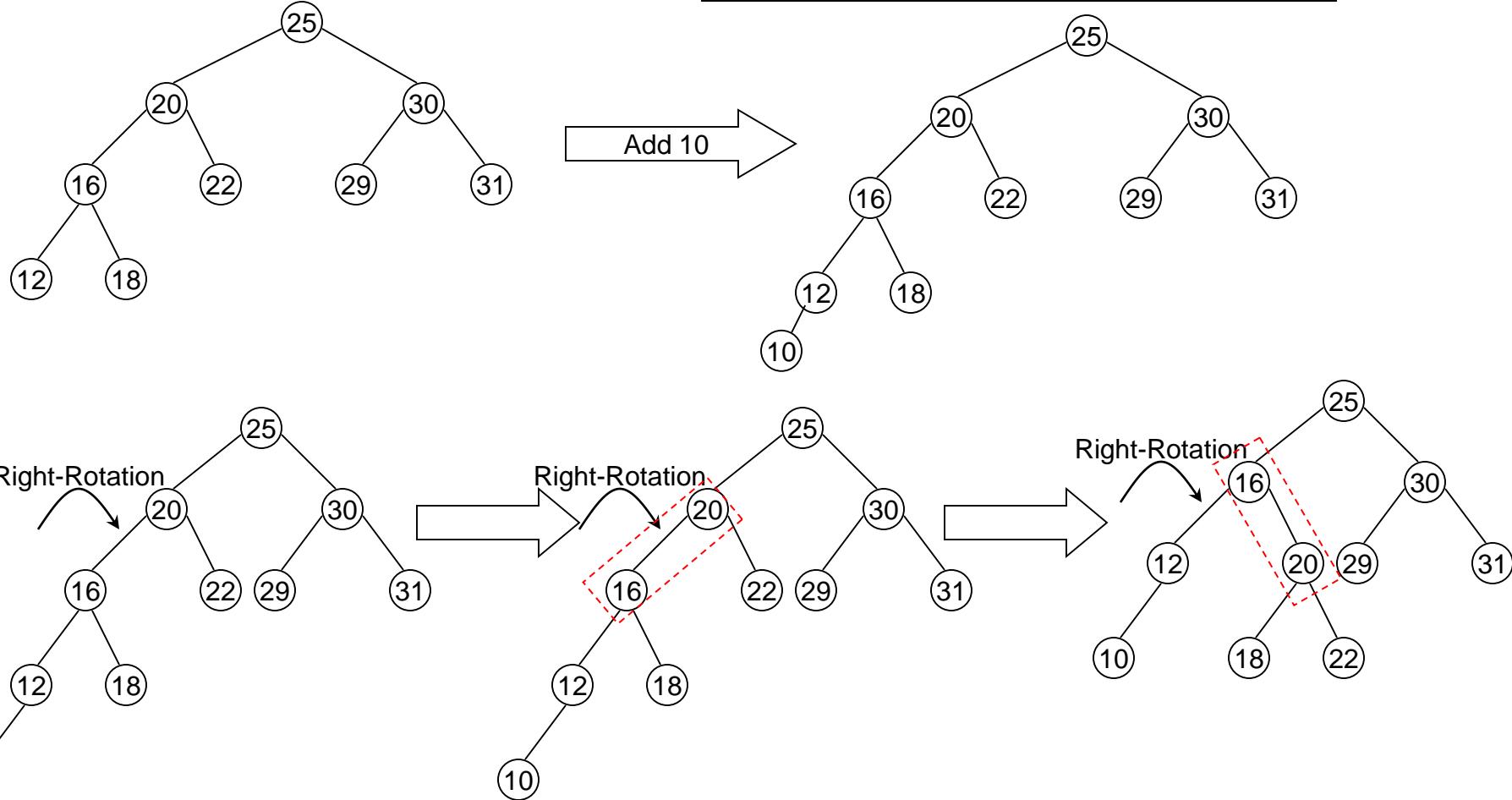
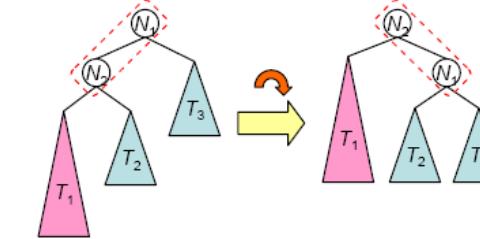
Solution: *Right* rotation



Right-Rotation Баруун-Эргүүлэлт

Case 2: insertion to *left* subtree of *left* child

Solution: *Right* rotation

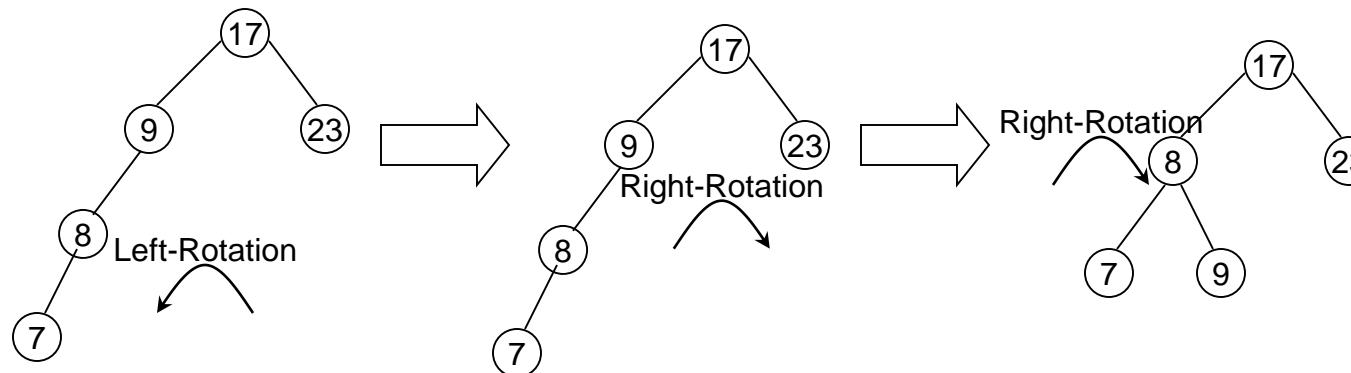
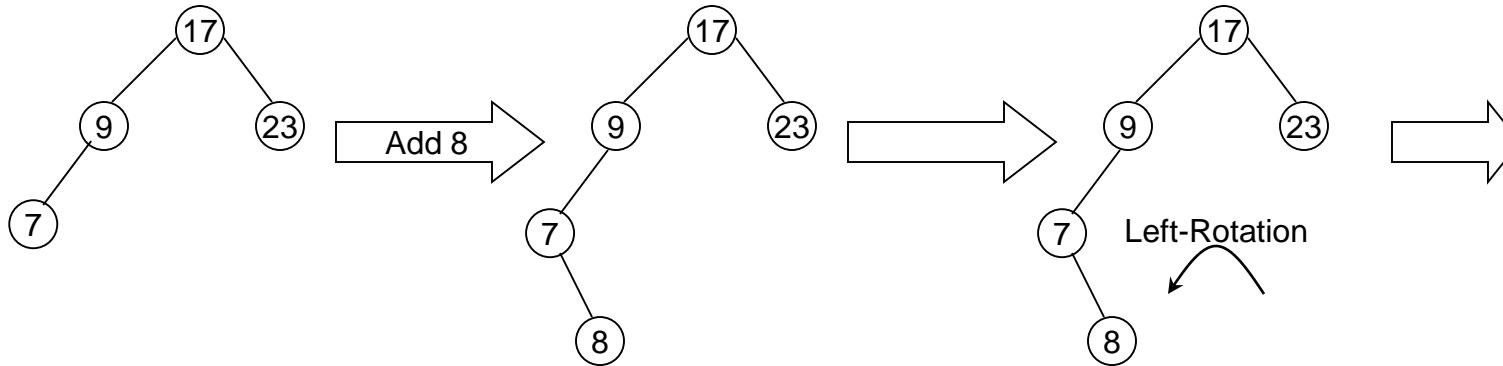
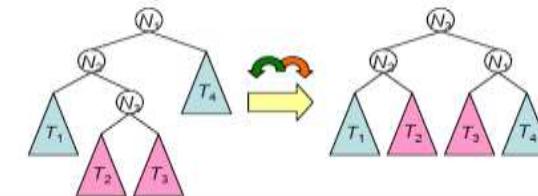


Left-Right-Rotation

Зүүн-Баруун-Эргүүлэлт

Case 3: insertion to *right* subtree of *left* child

Solution: *Left-right* rotation

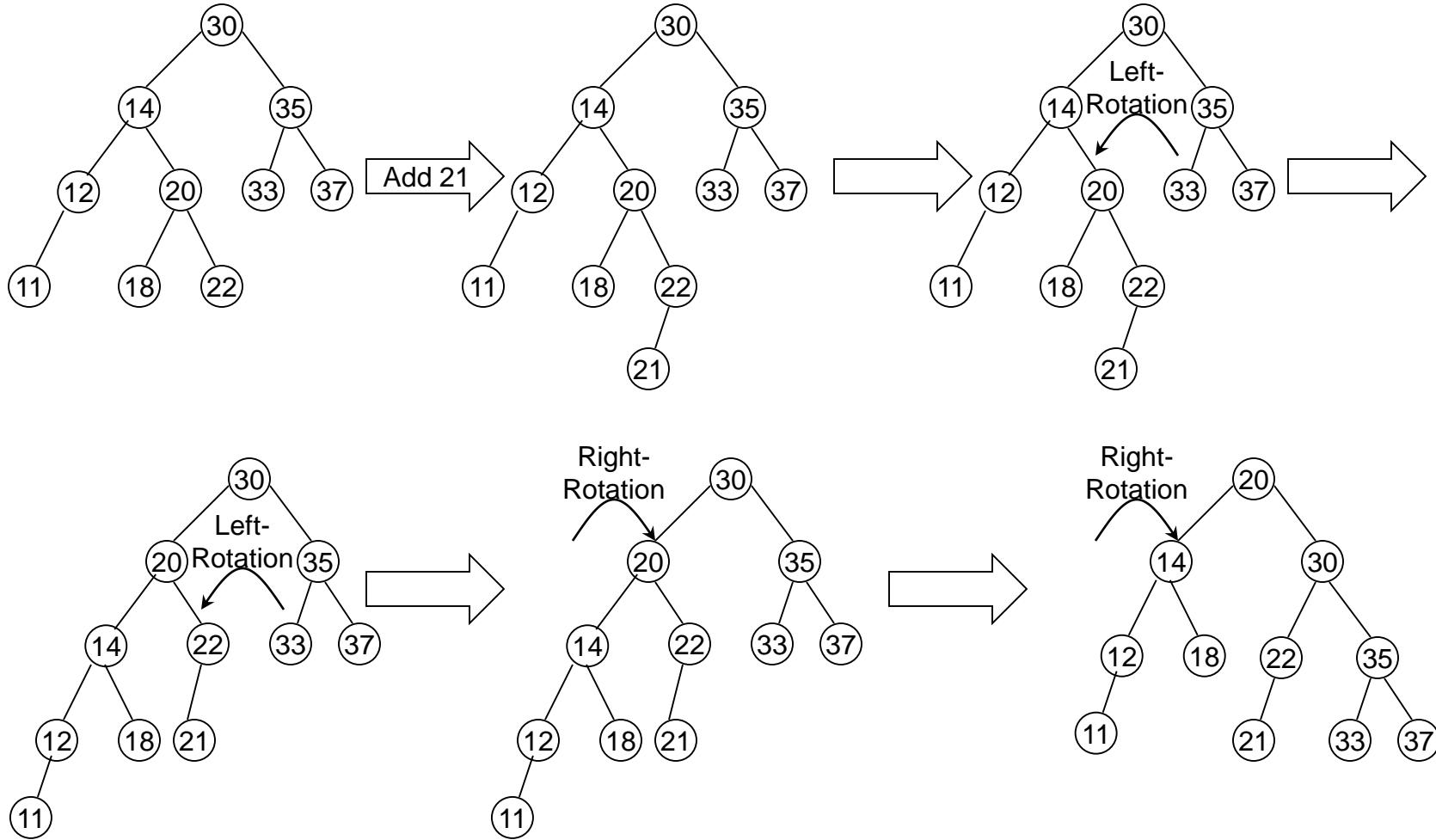
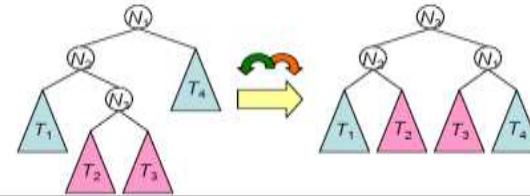


Left-Right-Rotation

Зүүн-Баруун-Эргүүлэлт

Case 3: insertion to *right* subtree of *left* child

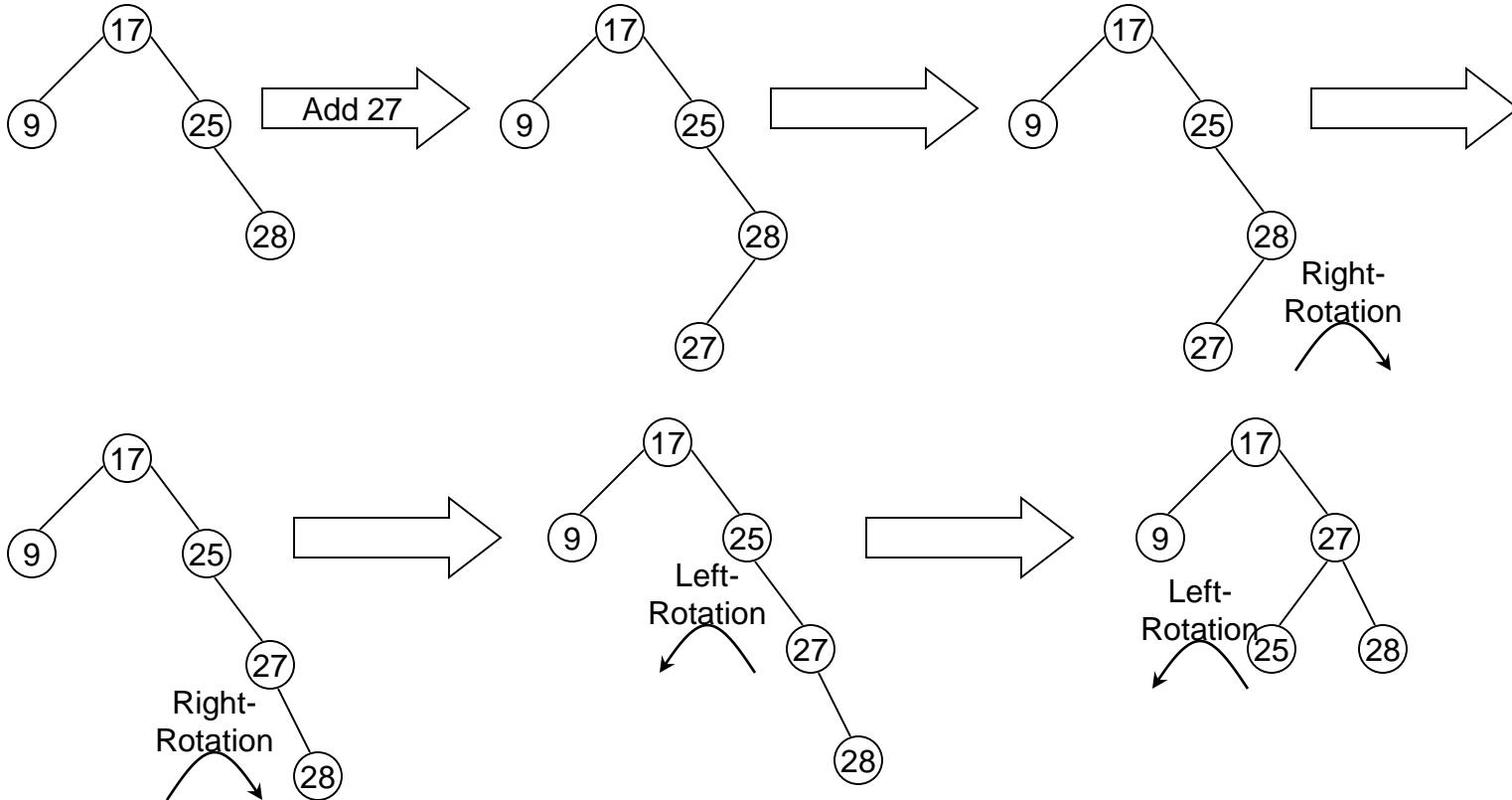
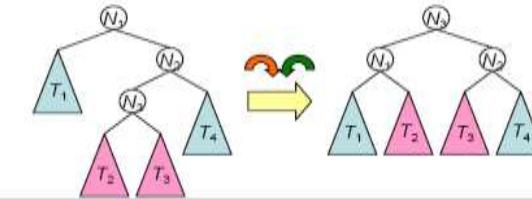
Solution: *Left-right* rotation



Right-Left-Rotation Баруун-Зүүн-Эргүүлэлт

Case 4: insertion to *left* subtree of *right* child

Solution: *Right-left* rotation



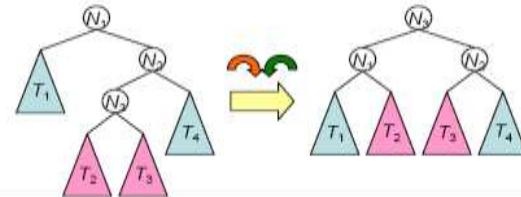
Right-Left-Rotation

Баруун-Зүүн-Эргүүлэлт

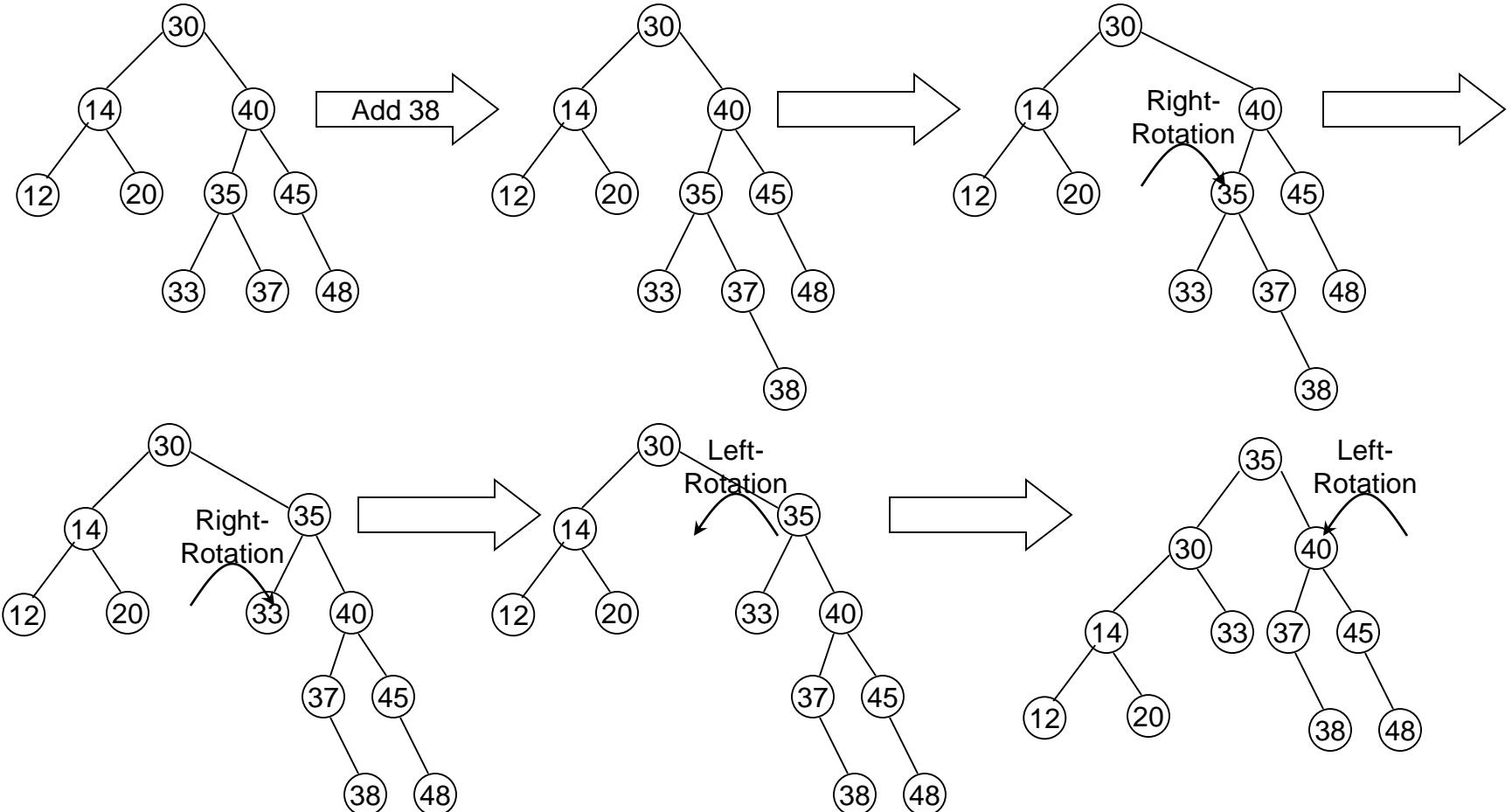
Case 4: insertion to *left*

subtree of *right* child

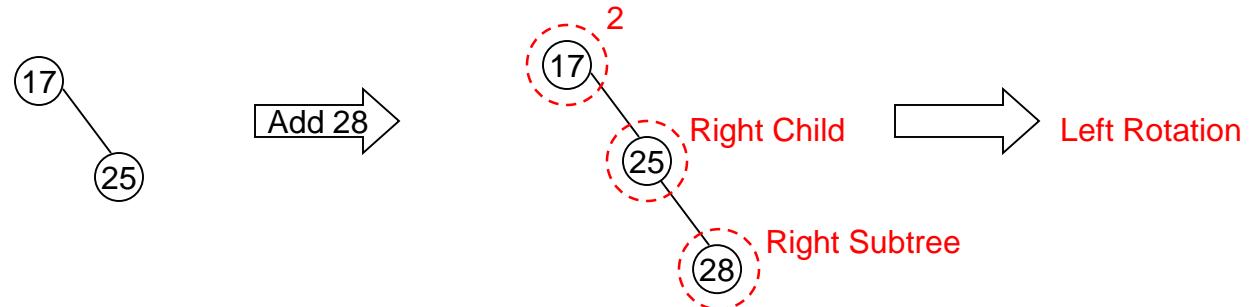
Solution: *Right-left* rotation



35

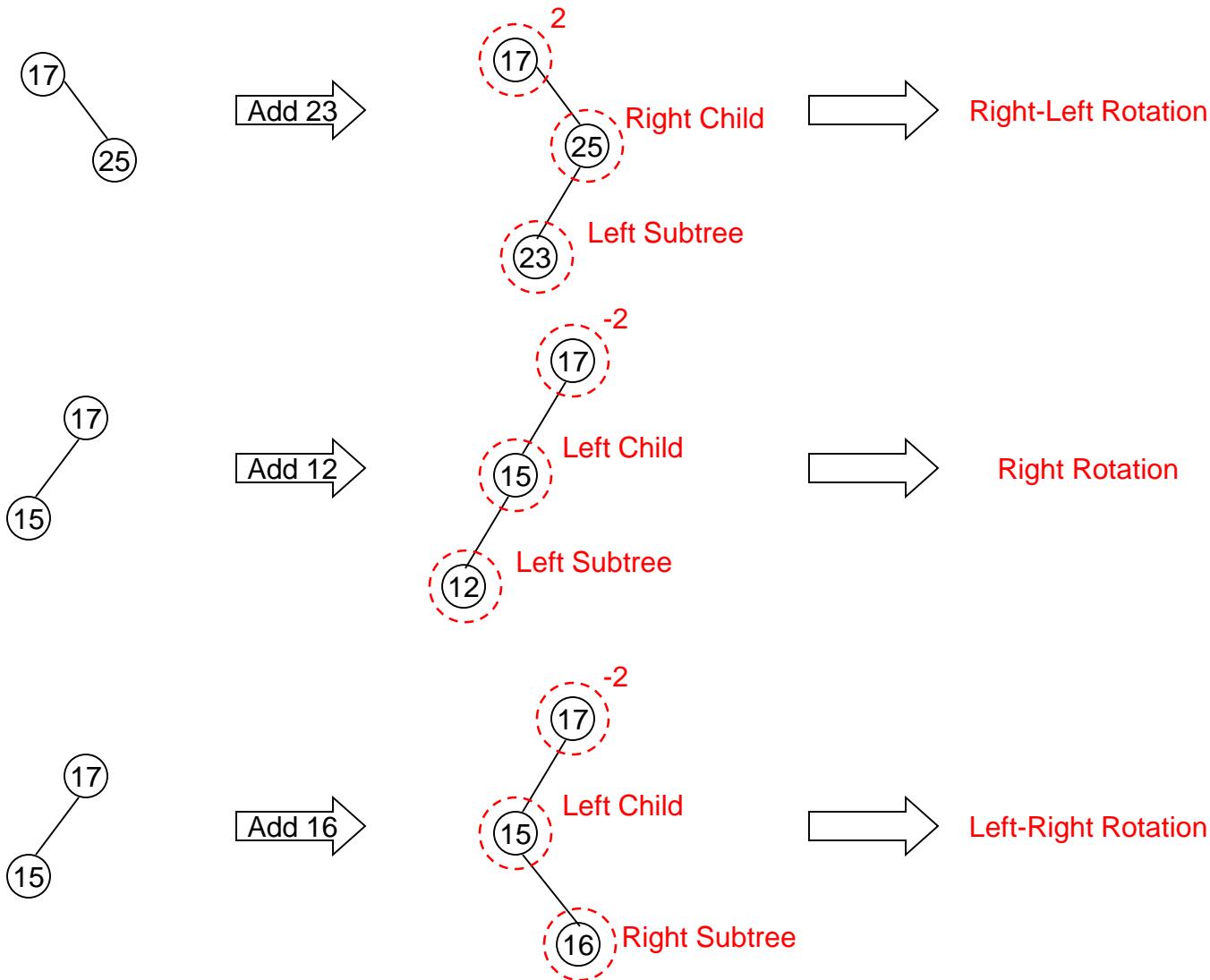


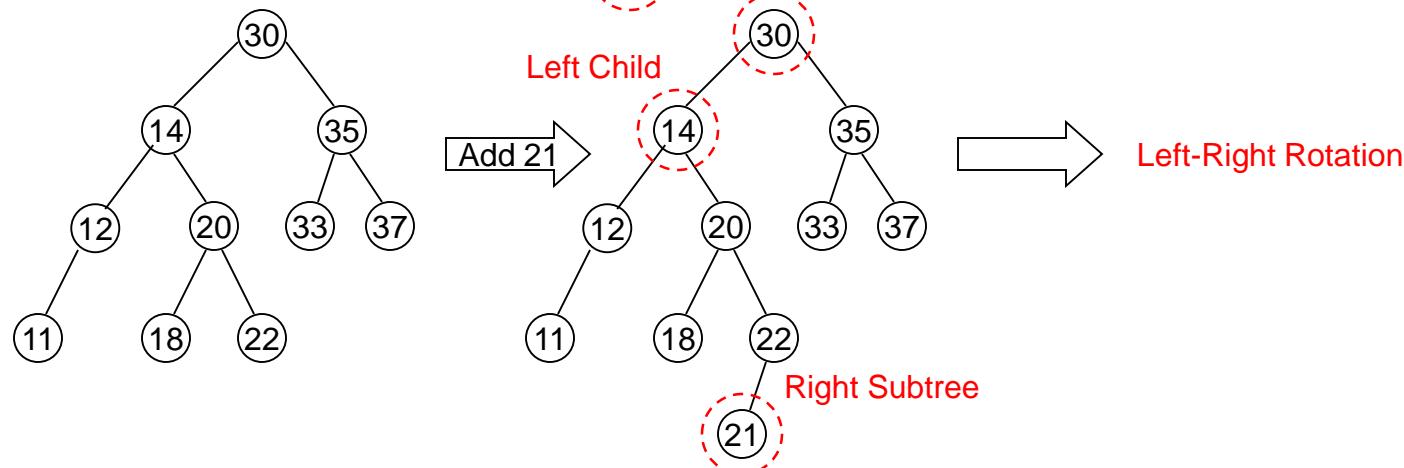
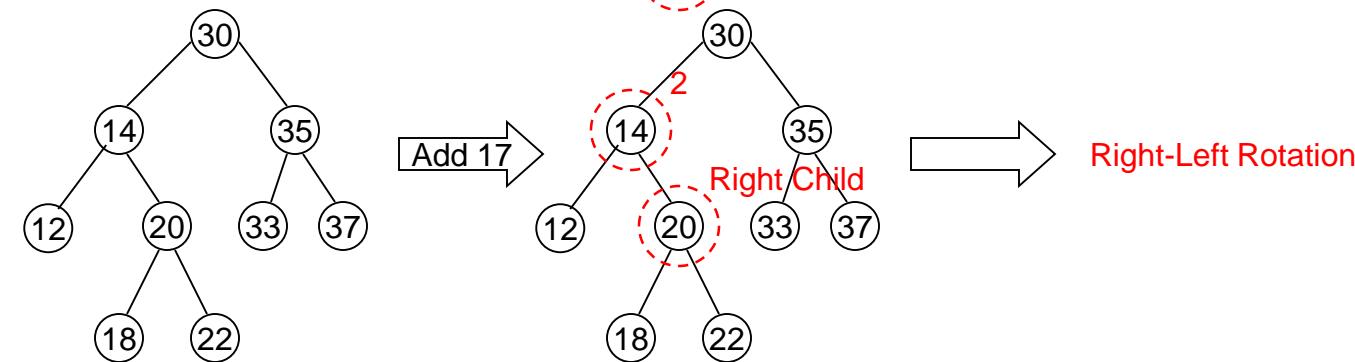
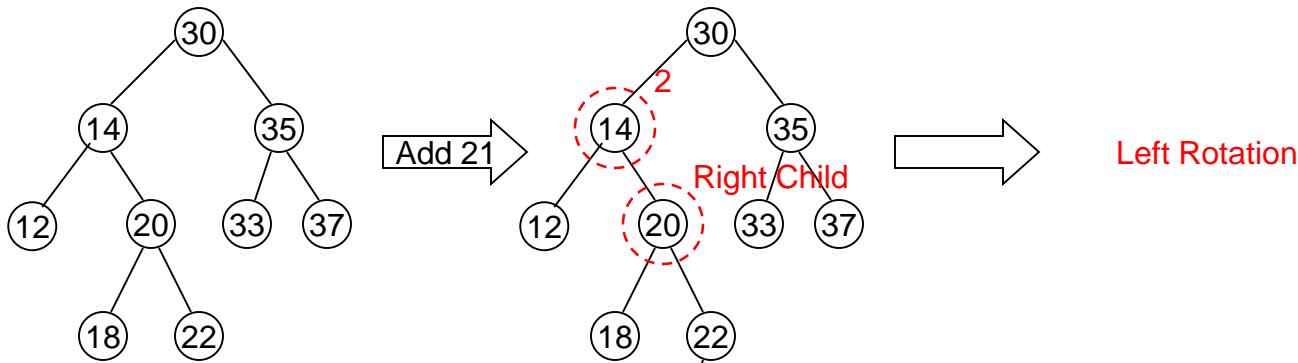
Эргэлтийг хэрхэн тодорхойлох вэ?



- Эхлээд тэнцвэргүй байдлыг үүсгэдэг зангилаа олно (тэнцвэрийн хүчин зүйл)
- Дараа нь тэнцвэргүй зангилааны харгалзах хүүхдийг олно (зүүн зангилаа эсвэл баруун зангилаа)
- Эцэст нь тухайн хүүхдийн харгалзах дэд модыг олно (зүүн эсвэл баруун)

Эргэлтийг хэрхэн тодорхойлох вэ?



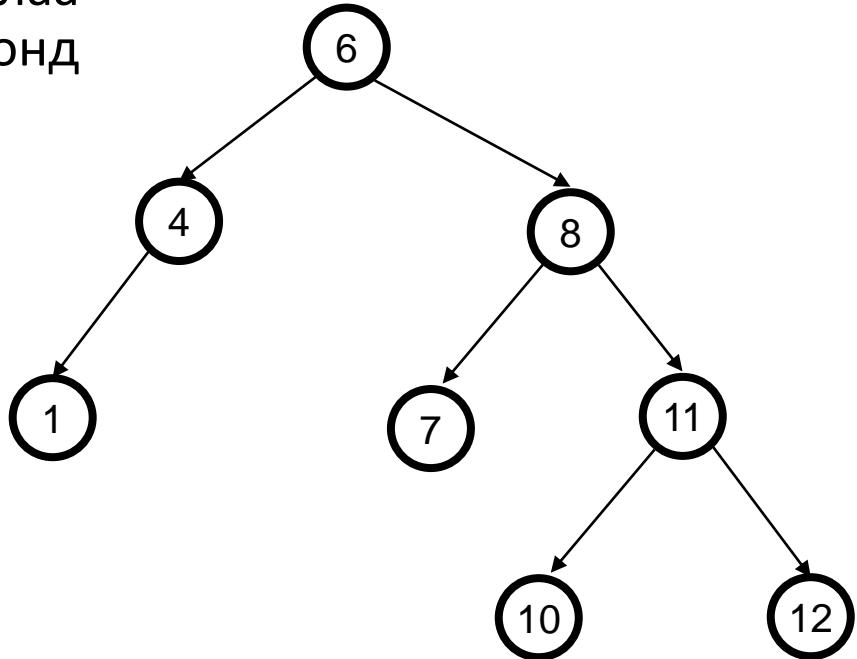


AVL animation

- <https://yongdanielliang.github.io/animation/web/AVLTree.html>
- <https://visualgo.net/en/bst>
- <http://www.motleytech.net/balanced-binary-tree-avl-tree-animation.html>
- <https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>

Жишээ №1: Энэ AVL мод мөн үү?

Тэнцвэрийн нөхцөл: зангилаа бүрийн тэнцэл -1-ээс 1-ийн хооронд байна.

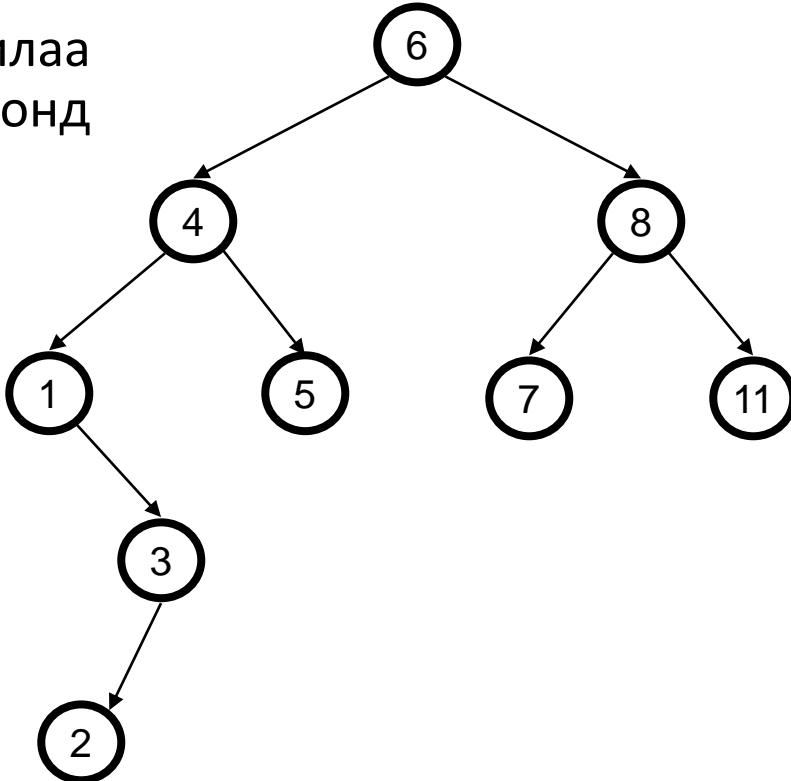


тэнцвэр(зангилаа) = өндөр(зангилаа.зүүн) – өндөр(зангилаа.баруун)

where $\text{balance}(\text{node}) = \text{height}(\text{node.left}) - \text{height}(\text{node.right})$

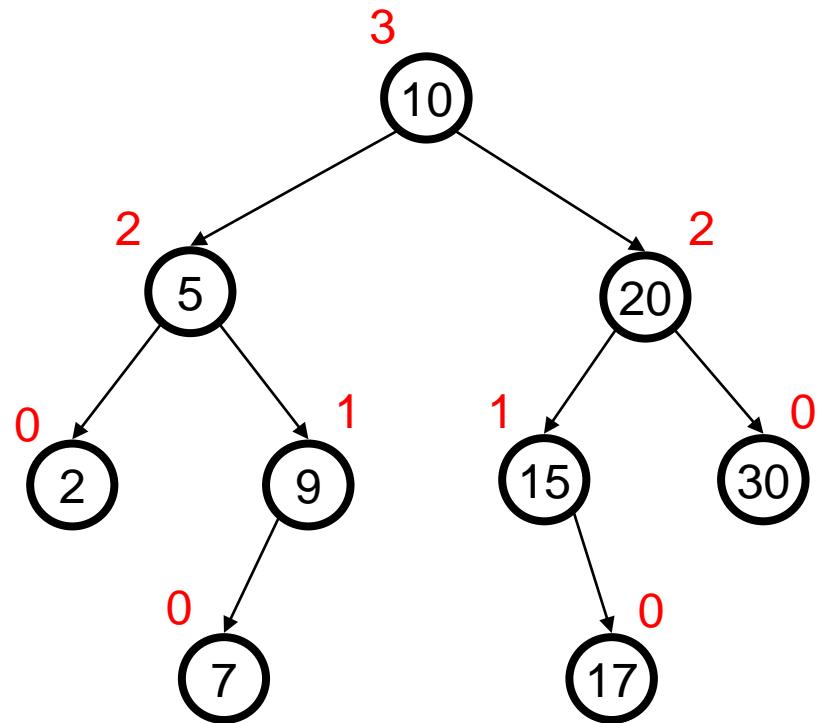
Жишээ №2: Энэ AVL мод мөн үү?

Тэнцвэрийн нөхцөл: зангилаа бүрийн тэнцэл -1-ээс 1-ийн хооронд байна.



тэнцвэр(зангилаа) = өндөр(зангилаа.зүүн) – өндөр(зангилаа.баруун)

AVL Trees



AVL tree operations / AVL модны үйлдлүүд

- AVL find:
 - Same as usual BST find
- AVL insert:
- AVL delete:
 - The “easy way” is lazy deletion
 - Otherwise, do the deletion and then check for several imbalance cases (we will skip this)

Оруулах жишээ

Insert(6)

Insert(3)

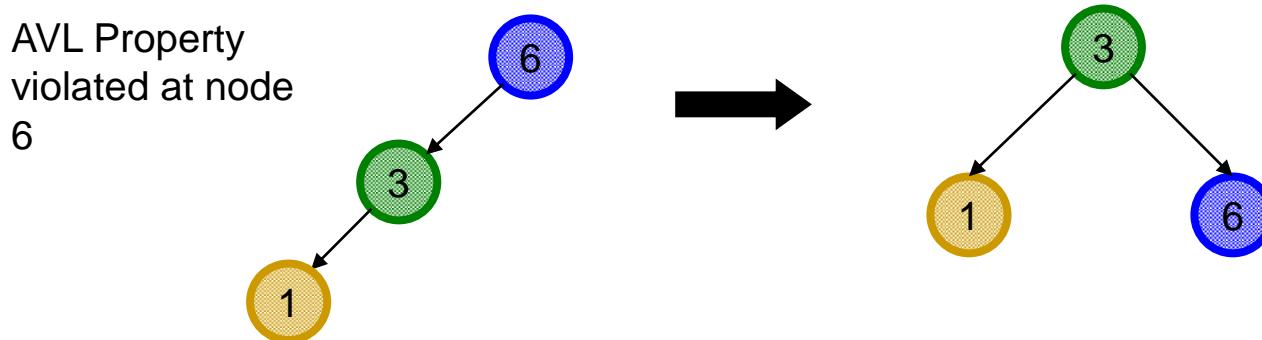
Insert(1)

Гурав дахь оруулга

Үүнийг засах цорын ганц арга зам юу вэ?

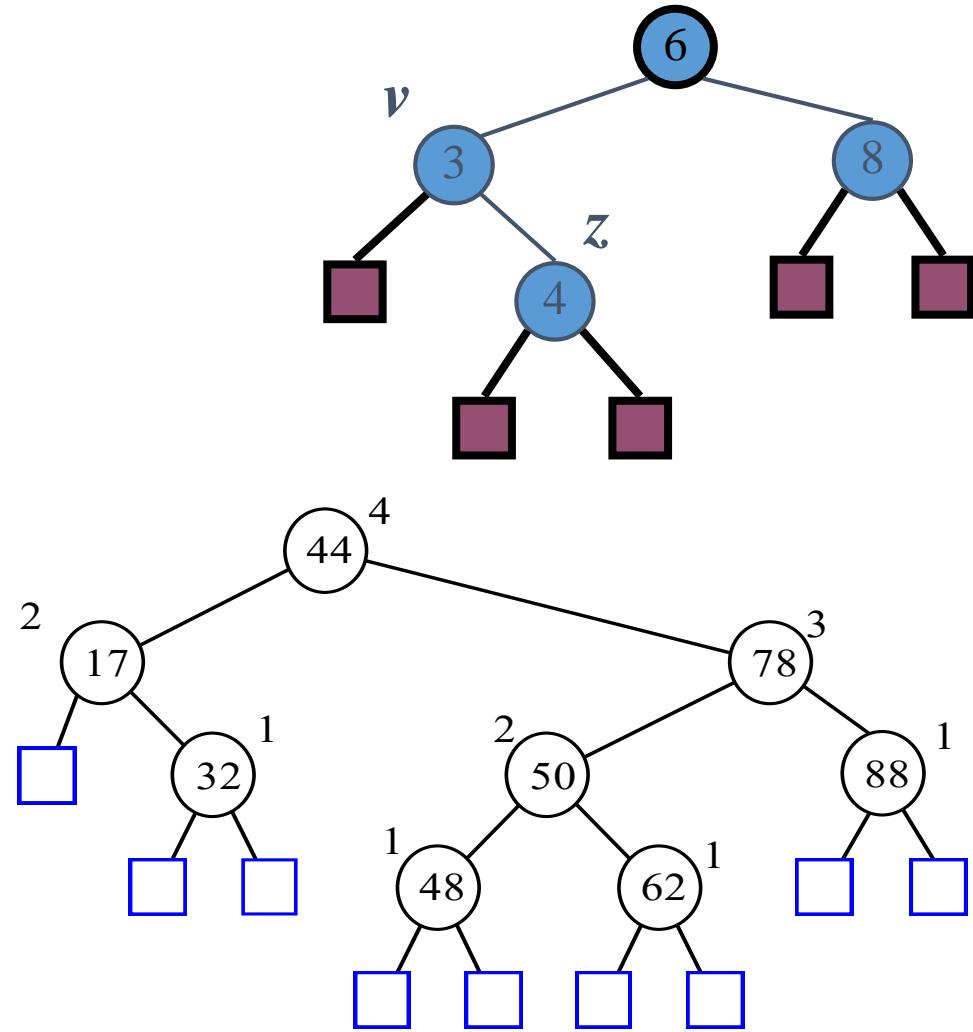
Fix: Apply “Single Rotation”

- **Single rotation:** The basic operation we'll use to rebalance
 - Move child of unbalanced node into parent position
 - Parent becomes the “other” child (always okay in a BST!)
 - Other subtrees move in only way BST allows (we'll see in generalized example)



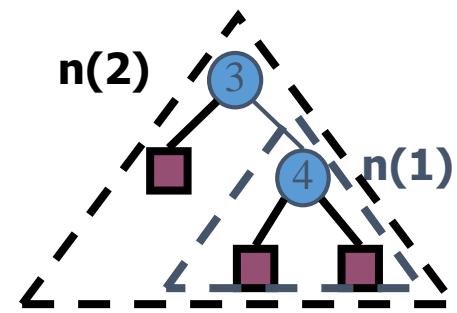
AVL Tree Definition

- AVL trees are rank-balanced trees.
- The **rank**, $r(v)$, of each node, v , is its height.
- **Rank-balance rule:** An AVL Tree is a binary search tree such that for every internal node v of T , the heights (ranks) of the children of v can differ by at most 1.



An example of an AVL tree where the ranks are shown next to the nodes

Height of an AVL Tree



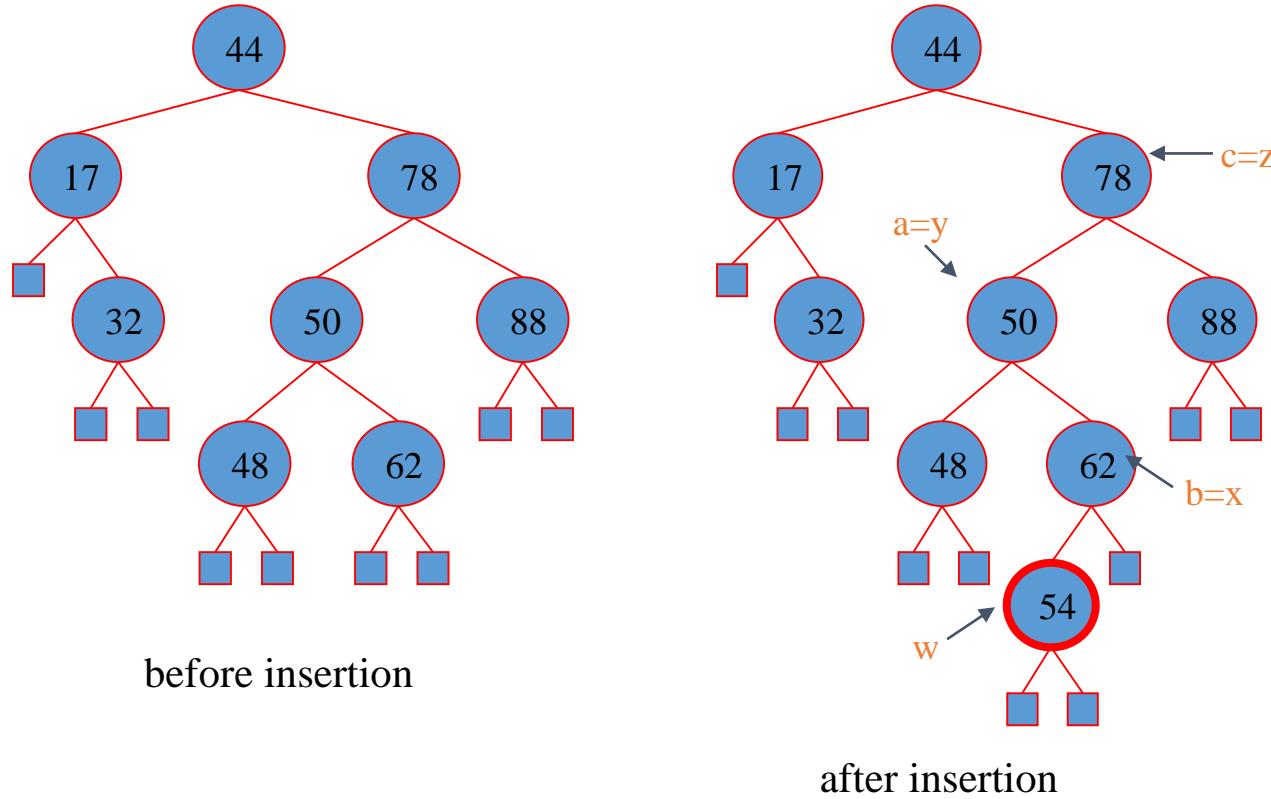
Fact: The height of an AVL tree storing n keys is $O(\log n)$.

Proof (by induction): Let us bound $n(h)$: the minimum number of internal nodes of an AVL tree of height h .

- We easily see that $n(1) = 1$ and $n(2) = 2$
- For $n > 2$, an AVL tree of height h contains the root node, one AVL subtree of height $n-1$ and another of height $n-2$.
- That is, $n(h) = 1 + n(h-1) + n(h-2)$
- Knowing $n(h-1) > n(h-2)$, we get $n(h) > 2n(h-2)$. So
 $n(h) > 2n(h-2)$, $n(h) > 4n(h-4)$, $n(h) > 8n(h-6)$, ... (by induction),
 $n(h) > 2^{i}n(h-2i)$
- Solving the base case we get: $n(h) > 2^{\frac{h}{2}-1}$
- Taking logarithms: $h < 2\log n(h) + 2$
- Thus the height of an AVL tree is $O(\log n)$

Insertion

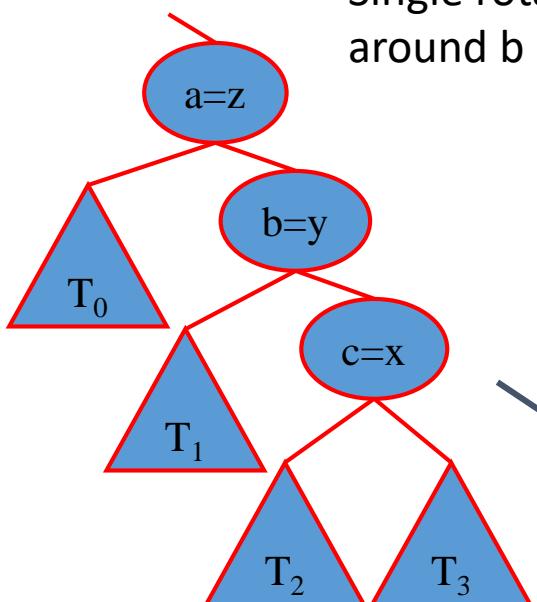
- Insertion is as in a binary search tree
- Always done by expanding an external node.
- Example:



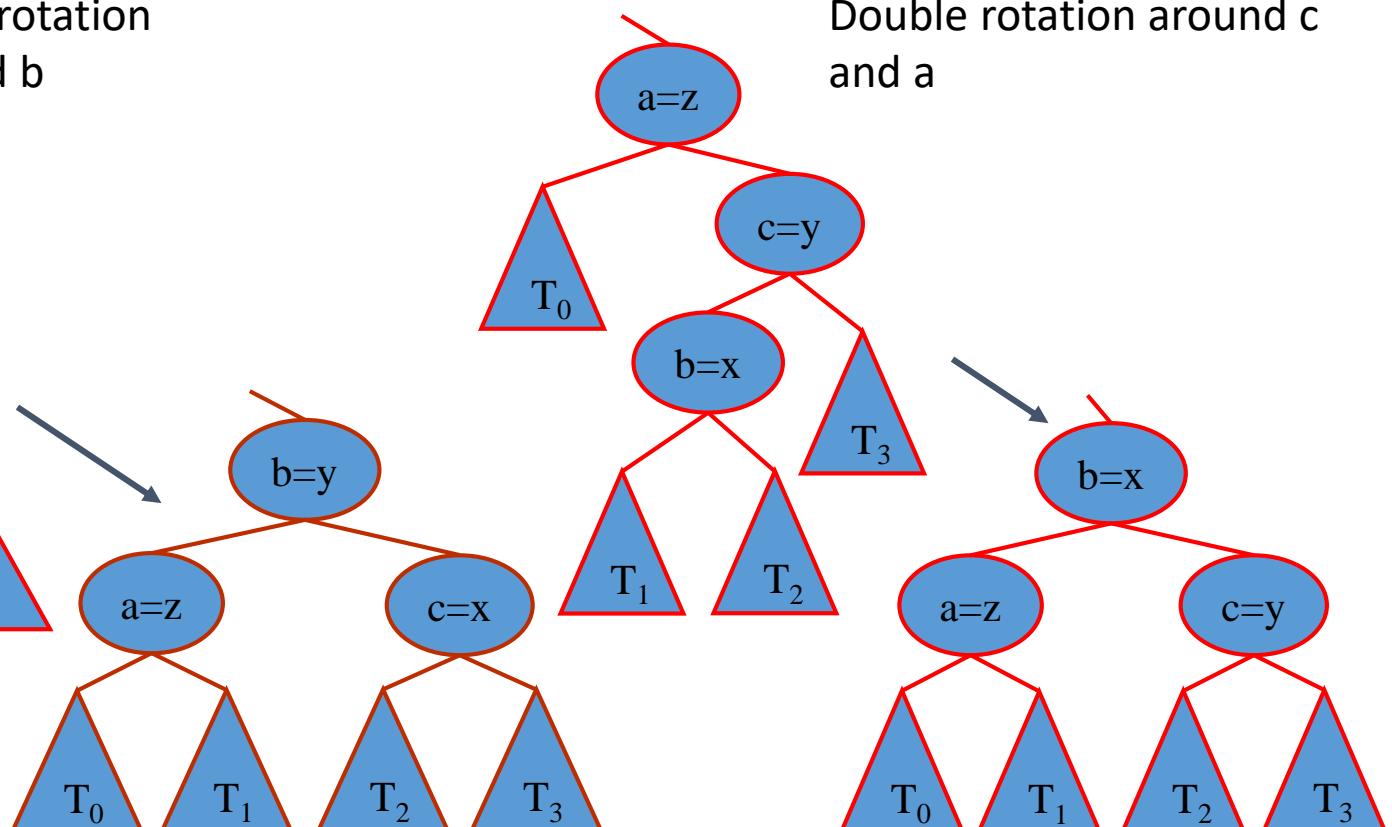
Trinode Restructuring

- Let (a,b,c) be the inorder listing of x, y, z
- Perform the rotations needed to make b the topmost node of the three

Single rotation
around b

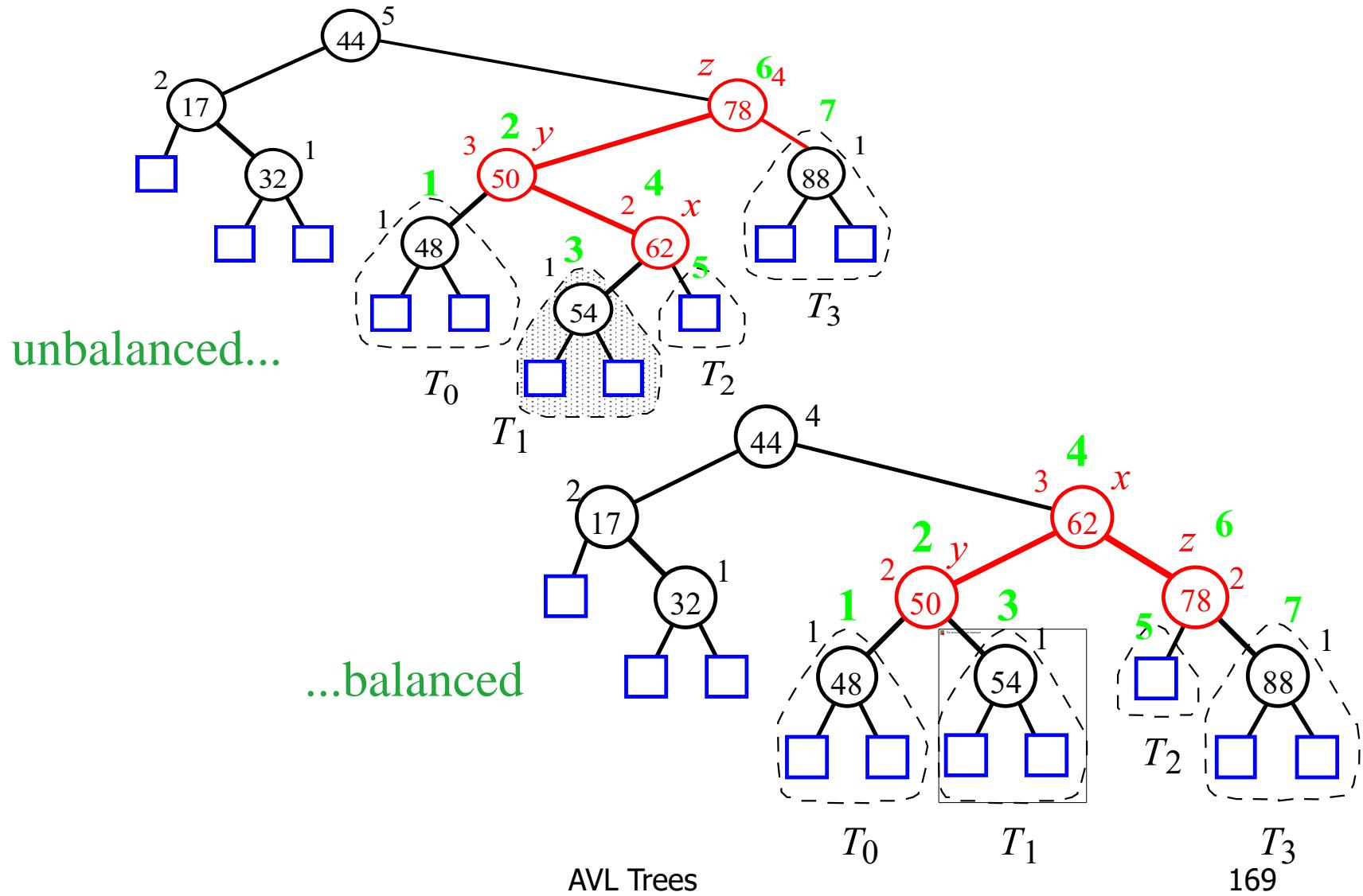


Double rotation around c
and a



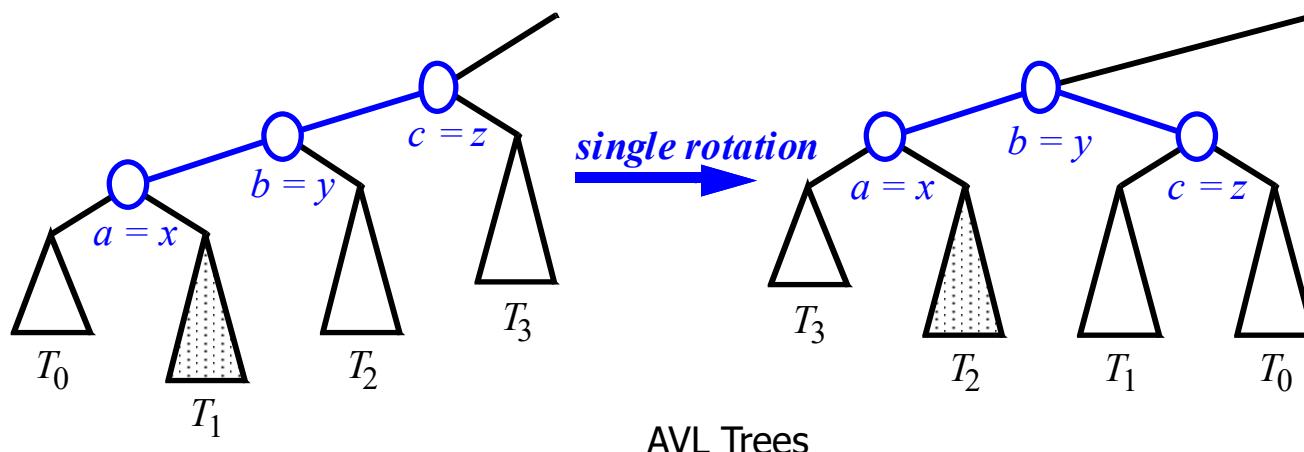
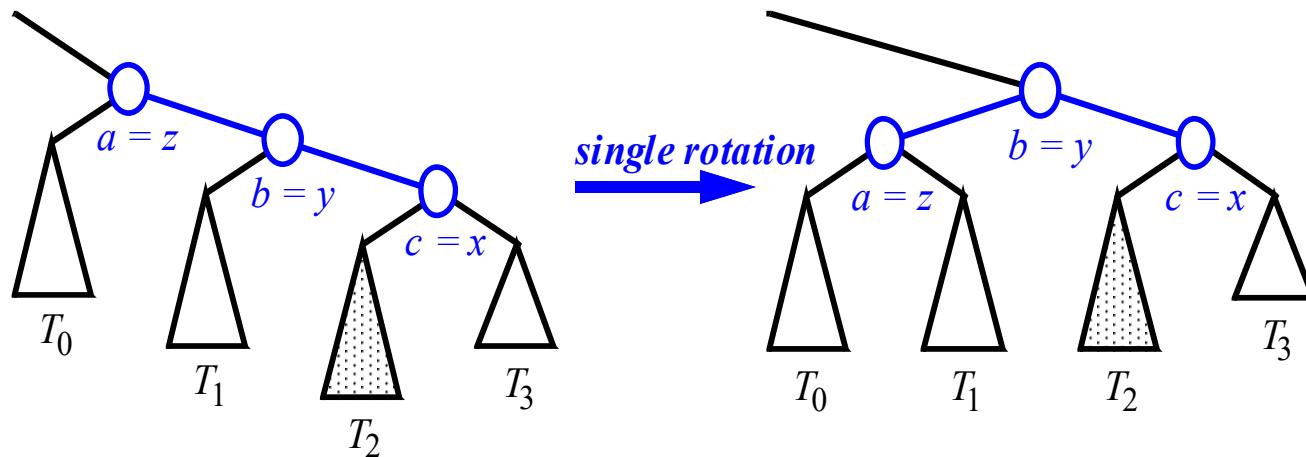
AVL Trees

Insertion Example, continued



Restructuring (as Single Rotations)

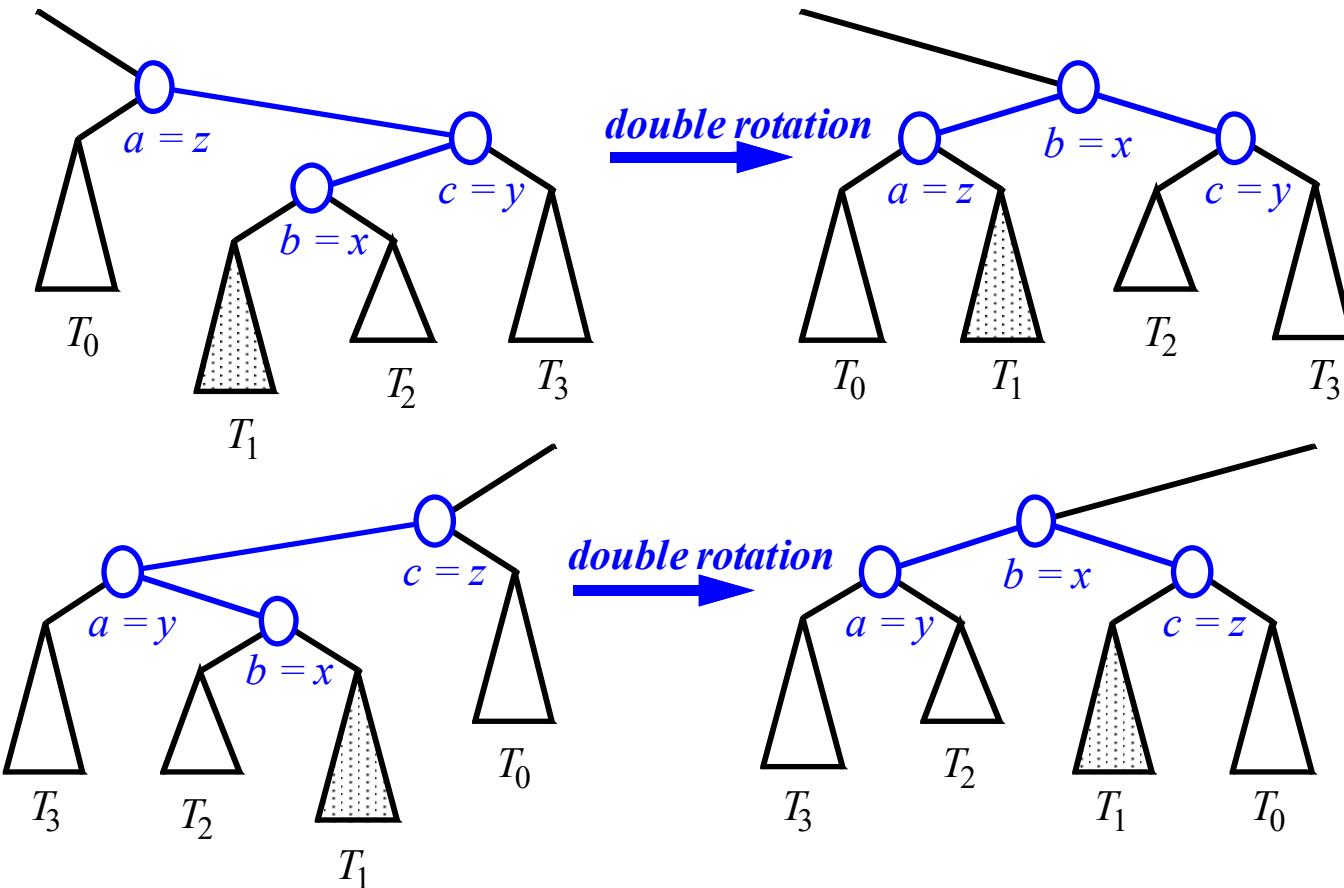
- Single Rotations:



AVL Trees

Restructuring (as Double Rotations)

- double rotations:



Pseudo-code

- Insertion.

Algorithm insertAVL(k, e, T):

Input: A key-element pair, (k, e) , and an AVL tree, T

Output: An update of T to now contain the item (k, e)

$v \leftarrow \text{IterativeTreeSearch}(k, T)$

if v is not an external node **then**

return “An item with key k is already in T ”

Expand v into an internal node with two external-node children

$v.\text{key} \leftarrow k$

$v.\text{element} \leftarrow e$

$v.\text{height} \leftarrow 1$

$\text{rebalanceAVL}(v, T)$

Pseudo-code

- Rebalance at a node violating the rank rule.

Algorithm rebalanceAVL(v, T):

Input: A node, v , where an imbalance may have occurred in an AVL tree, T

Output: An update of T to now be balanced

$v.\text{height} \leftarrow 1 + \max\{v.\text{leftChild}().\text{height}, v.\text{rightChild}().\text{height}\}$

while v is not the root of T **do**

$v \leftarrow v.\text{parent}()$

if $|v.\text{leftChild}().\text{height} - v.\text{rightChild}().\text{height}| > 1$ **then**

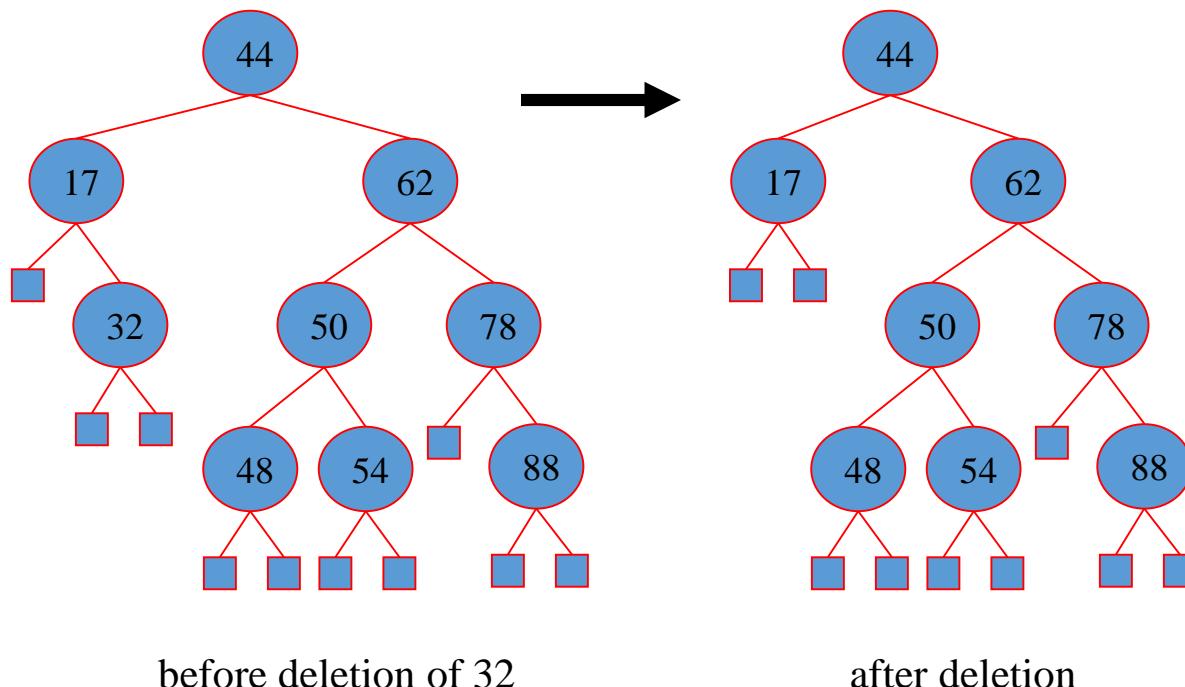
 Let y be the tallest child of v and let x be the tallest child of y

$v \leftarrow \text{restructure}(x)$ // trinode restructure operation

$v.\text{height} \leftarrow 1 + \max\{v.\text{leftChild}().\text{height}, v.\text{rightChild}().\text{height}\}$

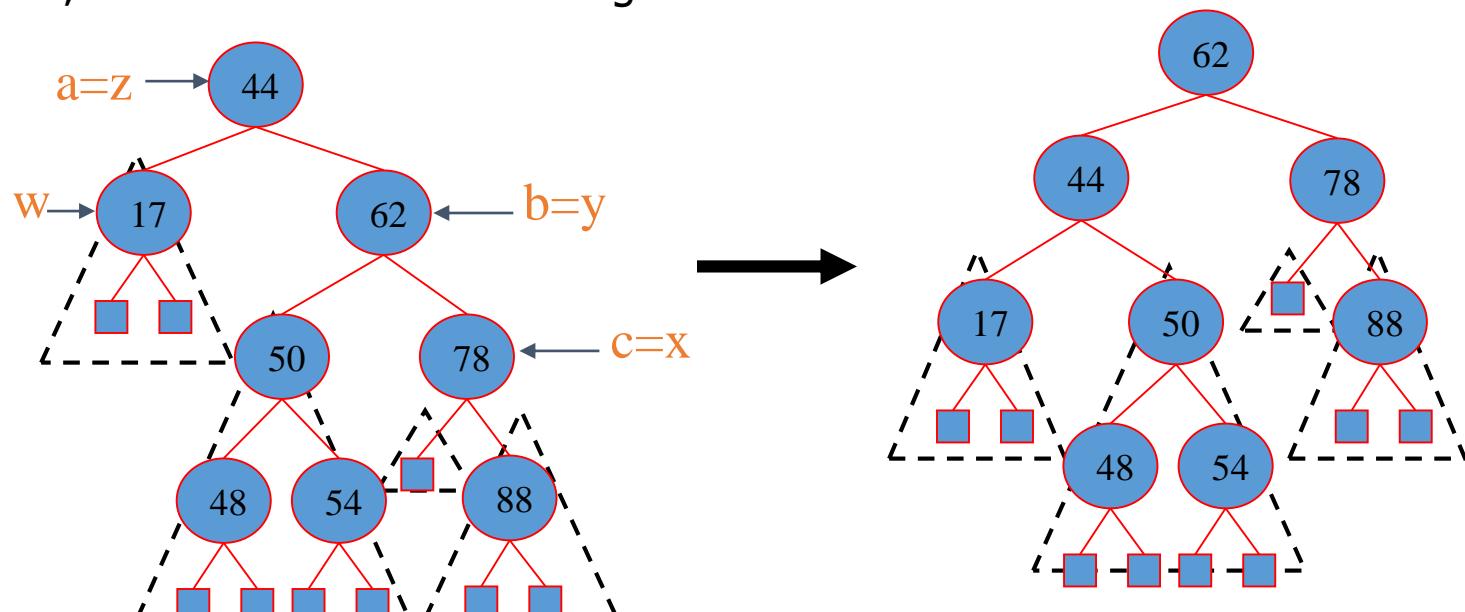
Removal

- Removal begins as in a binary search tree, which means the node removed will become an empty external node. Its parent, w, may cause an imbalance.
- Example:



Rebalancing after a Removal

- Let z be the first unbalanced node encountered while travelling up the tree from w . Also, let y be the child of z with the larger height, and let x be the child of y with the larger height
- We perform a trinode restructuring to restore balance at z
- As this restructuring may upset the balance of another node higher in the tree, we must continue checking for balance until the root of T is reached



Pseudo-code

- Removal

Algorithm removeAVL(k, T):

Input: A key, k , and an AVL tree, T

Output: An update of T to now have an item (k, e) removed

$v \leftarrow \text{IterativeTreeSearch}(k, T)$

if v is an external node **then**

return “There is no item with key k in T ”

if v has no external-node child **then**

Let u be the node in T with key nearest to k

Move u ’s key-value pair to v

$v \leftarrow u$

Let w be v ’s smallest-height child

Remove w and v from T , replacing v with w ’s sibling, z

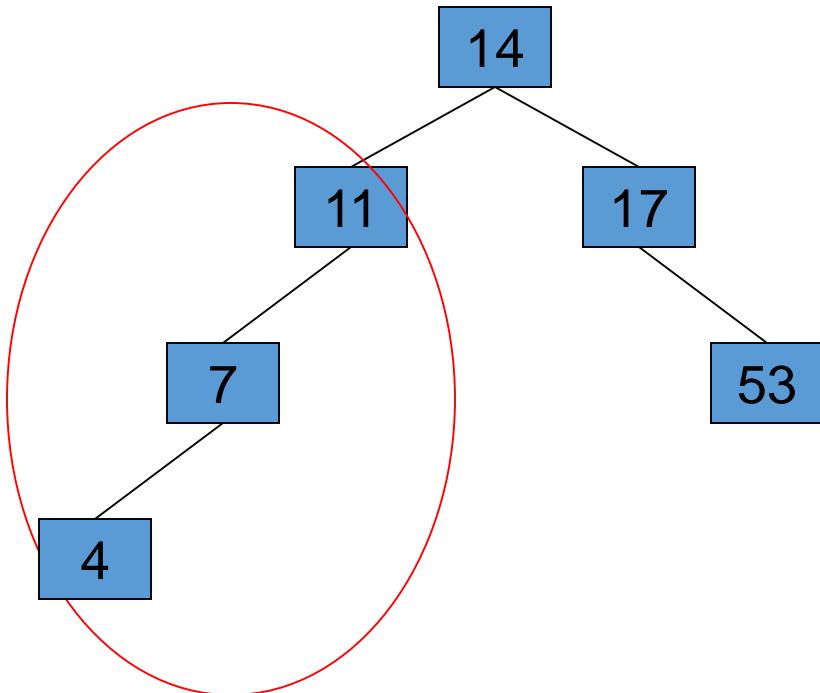
rebalanceAVL(z, T)

AVL Tree Performance

- AVL tree storing n items
 - The data structure uses $O(n)$ space
 - A single restructuring takes $O(1)$ time
 - using a linked-structure binary tree
 - Searching takes $O(\log n)$ time
 - height of tree is $O(\log n)$, no restructures needed
 - Insertion takes $O(\log n)$ time
 - initial find is $O(\log n)$
 - restructuring up the tree, maintaining heights is $O(\log n)$
 - Removal takes $O(\log n)$ time
 - initial find is $O(\log n)$
 - restructuring up the tree, maintaining heights is $O(\log n)$

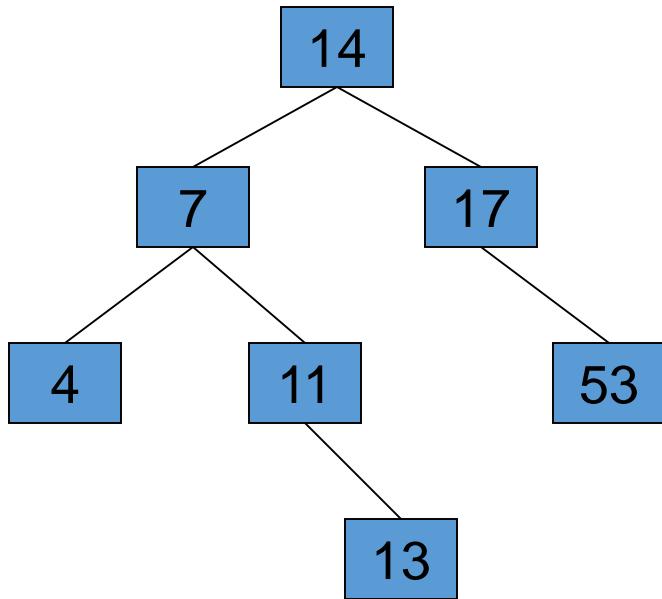
AVL МОДНЫ ЖИШЭЭ

- Хоосон AVL модонд 14, 17, 11, 7, 53, 4, 13-ыг оруулна.



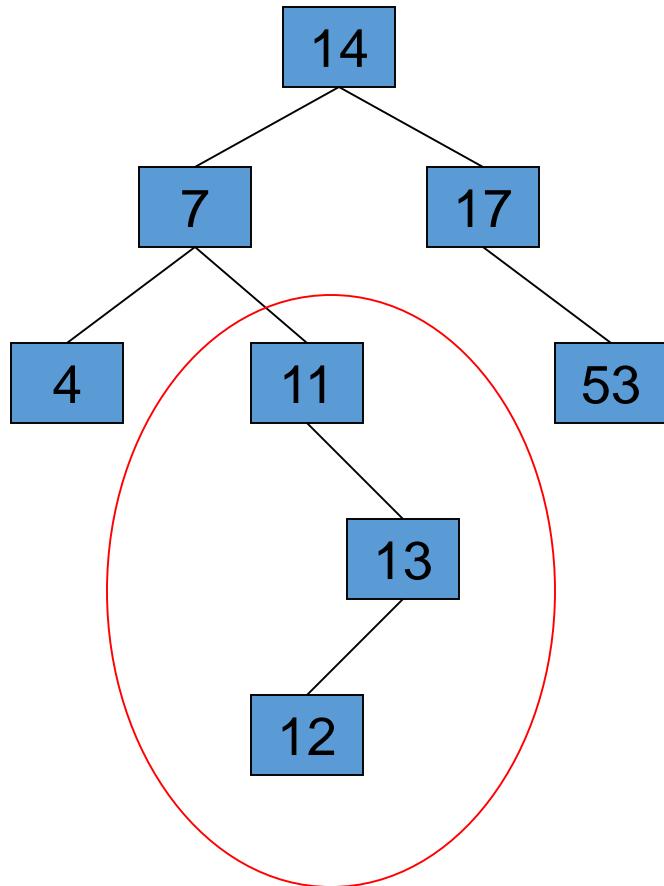
AVL МОДНЫ ЖИШЭЭ

- Хоосон AVL модонд 14, 17, 11, 7, 53, 4, 13-ыг оруулна.



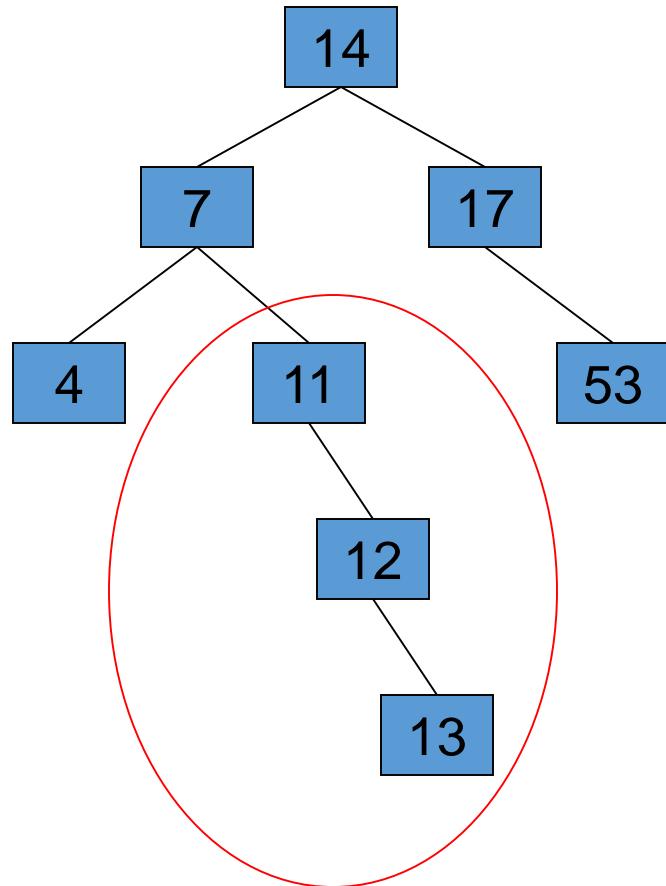
AVL МОДНЫ ЖИШЭЭ

- Одоо 12-ыг оруулах



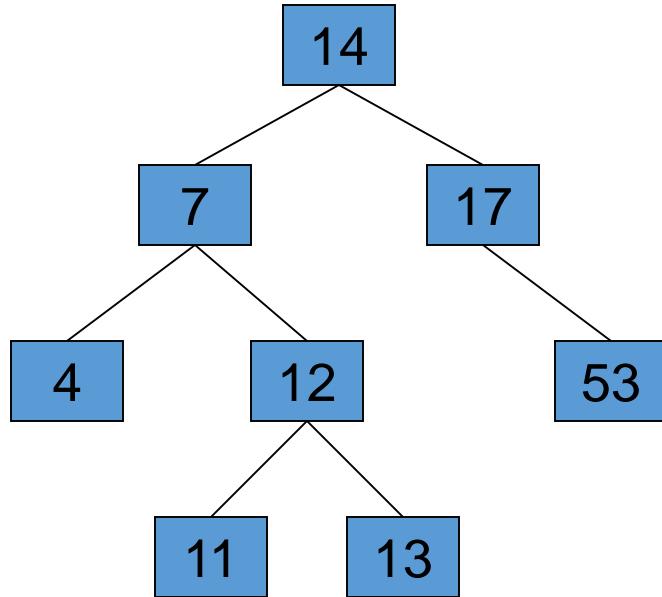
AVL МОДНЫ ЖИШЭЭ

- Одоо 12-ыг оруулах



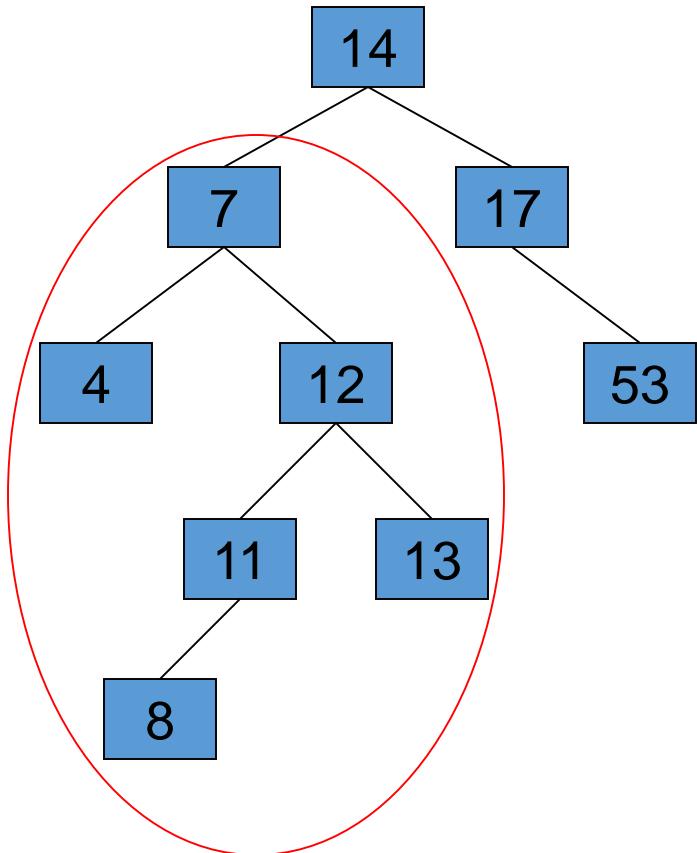
AVL МОДНЫ ЖИШЭЭ

- Одоо AVL мод тэнцвэртэй байна.



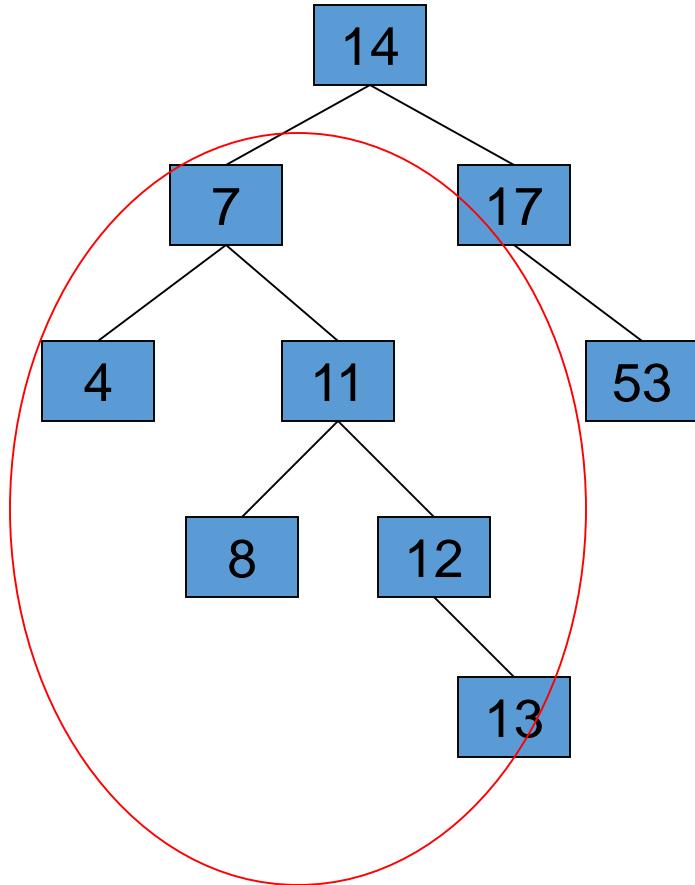
AVL МОДНЫ ЖИШЭЭ

- Одоо 8-ыг оруулах



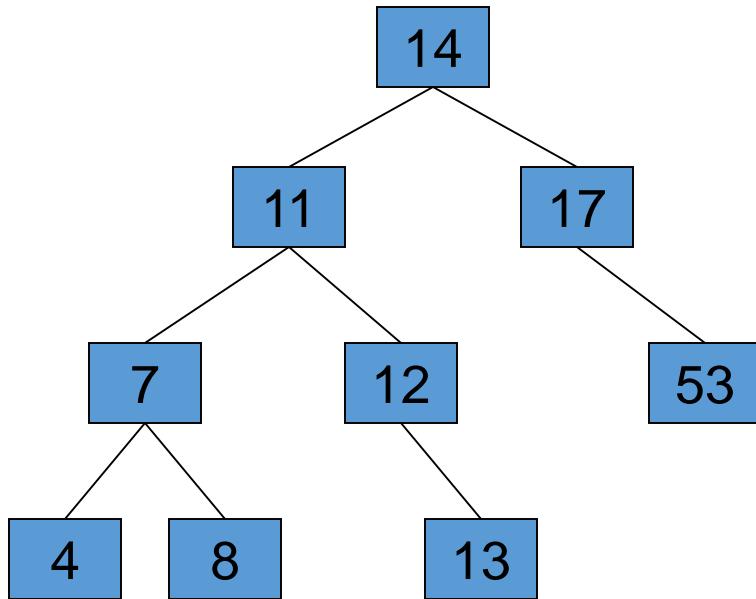
AVL МОДНЫ ЖИШЭЭ

- Одоо 8-ыг оруулах



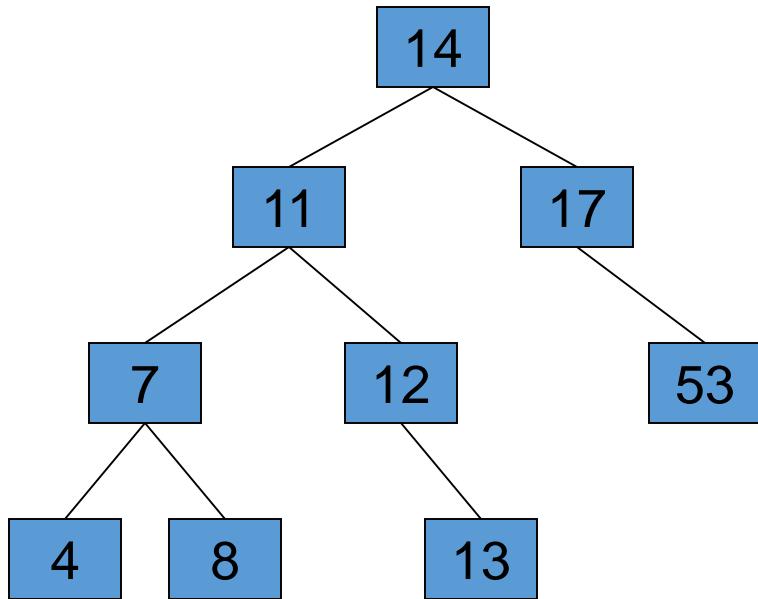
AVL МОДНЫ ЖИШЭЭ

- Одоо AVL мод тэнцвэртэй байна.



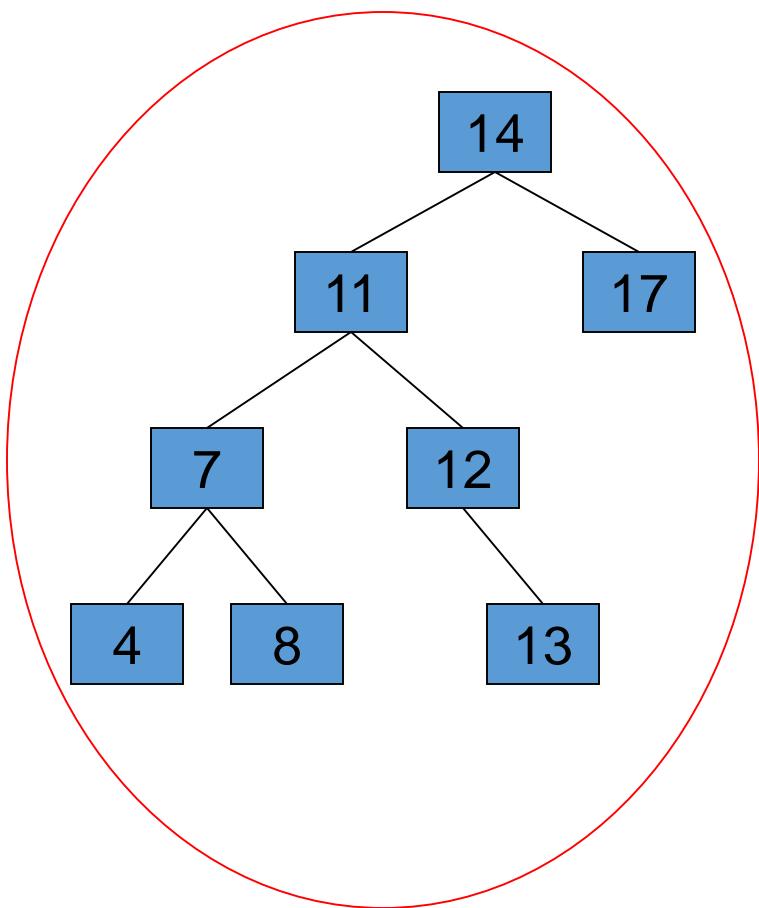
AVL МОДНЫ ЖИШЭЭ

- Одоо 53-ыг устгах



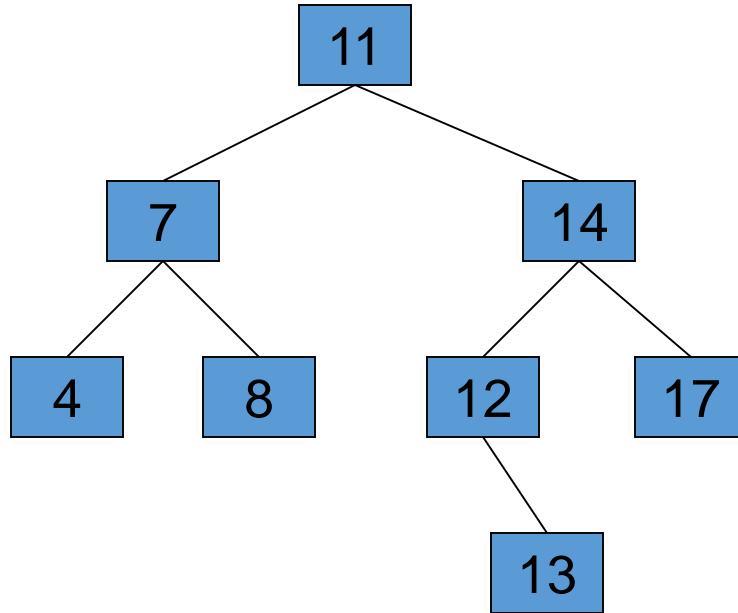
AVL МОДНЫ ЖИШЭЭ

- Одоо 53-ыг устгасан ба тэнцвэргүй болсон.



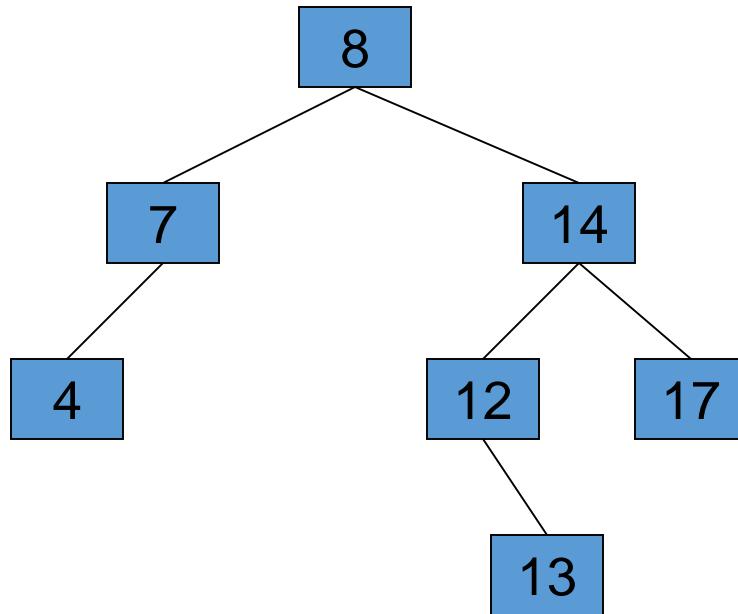
AVL МОДНЫ ЖИШЭЭ

- Одоо 11-ийг устгаж тэнцвэртэй болгох.



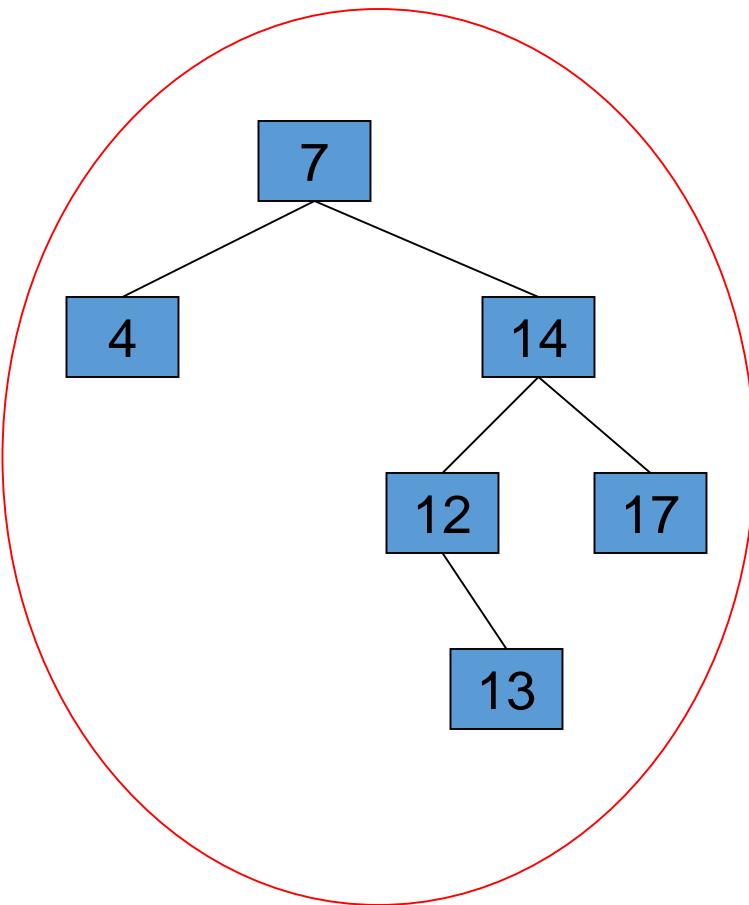
AVL МОДНЫ ЖИШЭЭ

- Одоо 11-ийг устгаад зүүн талынх нь хамгийн их утгаар нь солино.



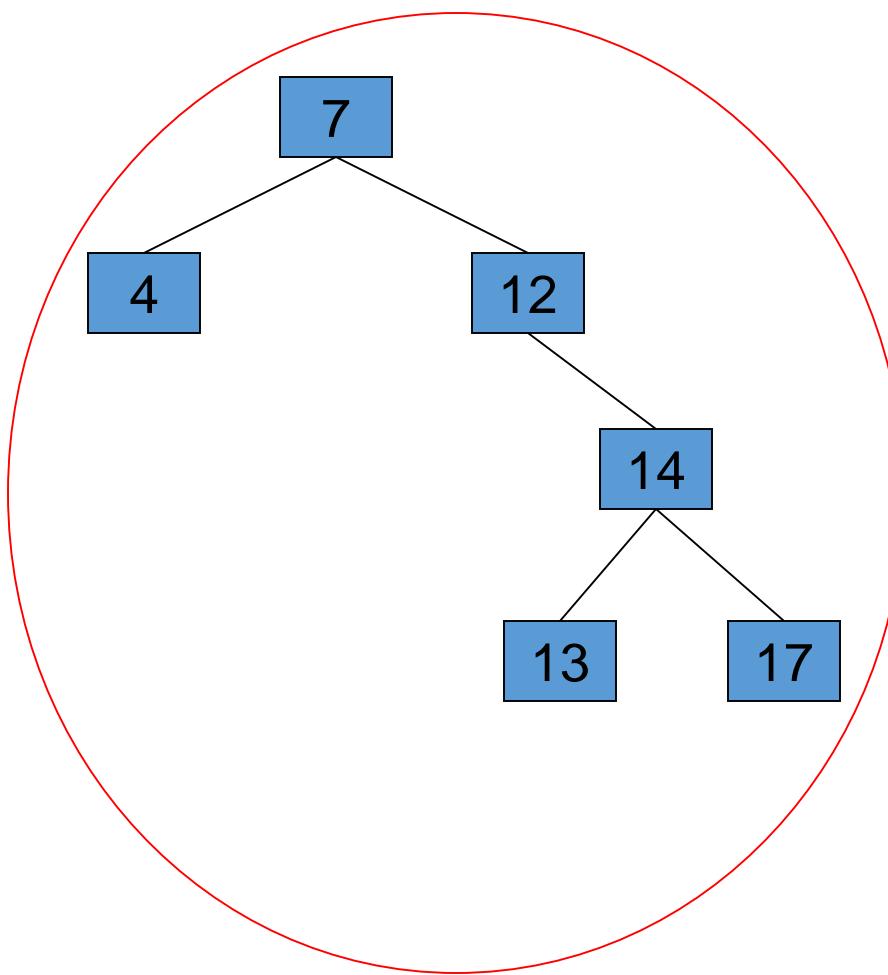
AVL МОДНЫ ЖИШЭЭ

- Одоо 8-ыг устгасан ба тэнцвэргүй болсон.



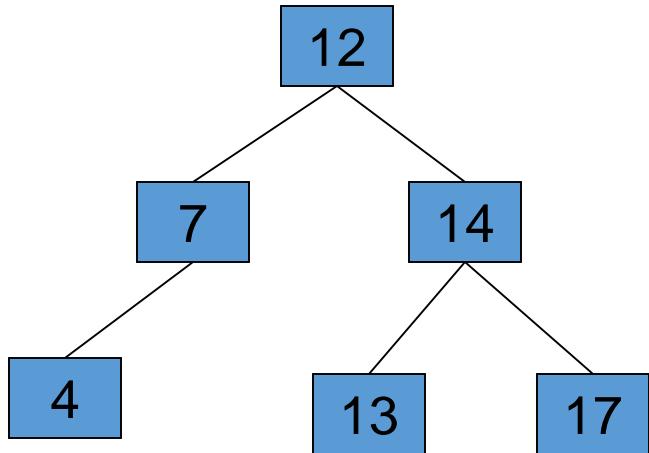
AVL МОДНЫ ЖИШЭЭ

- Одоо 8-ыг устгасан ба тэнцвэргүй болсон.



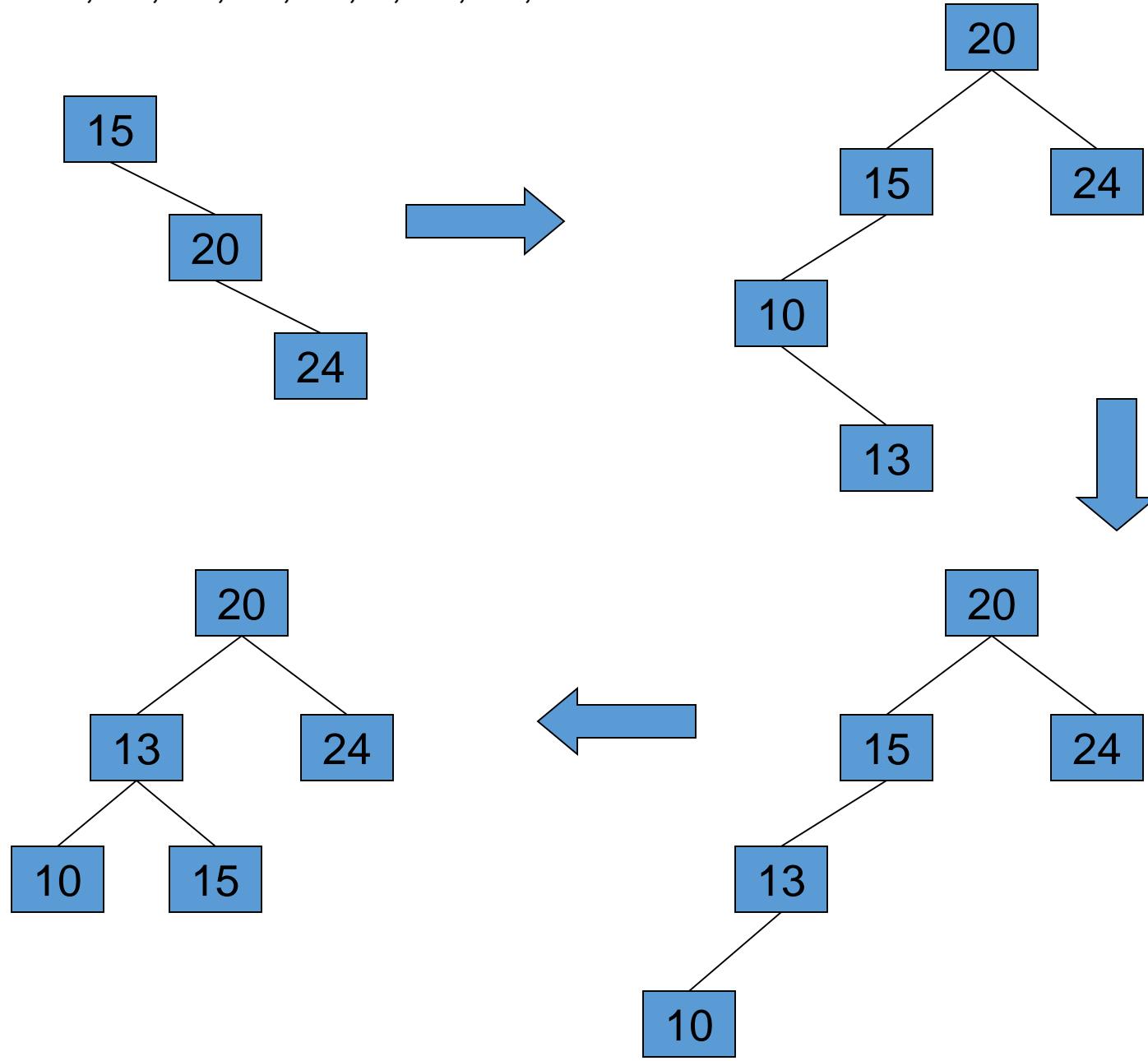
AVL МОДНЫ ЖИШЭЭ

- Тэнцвэртэй !!!

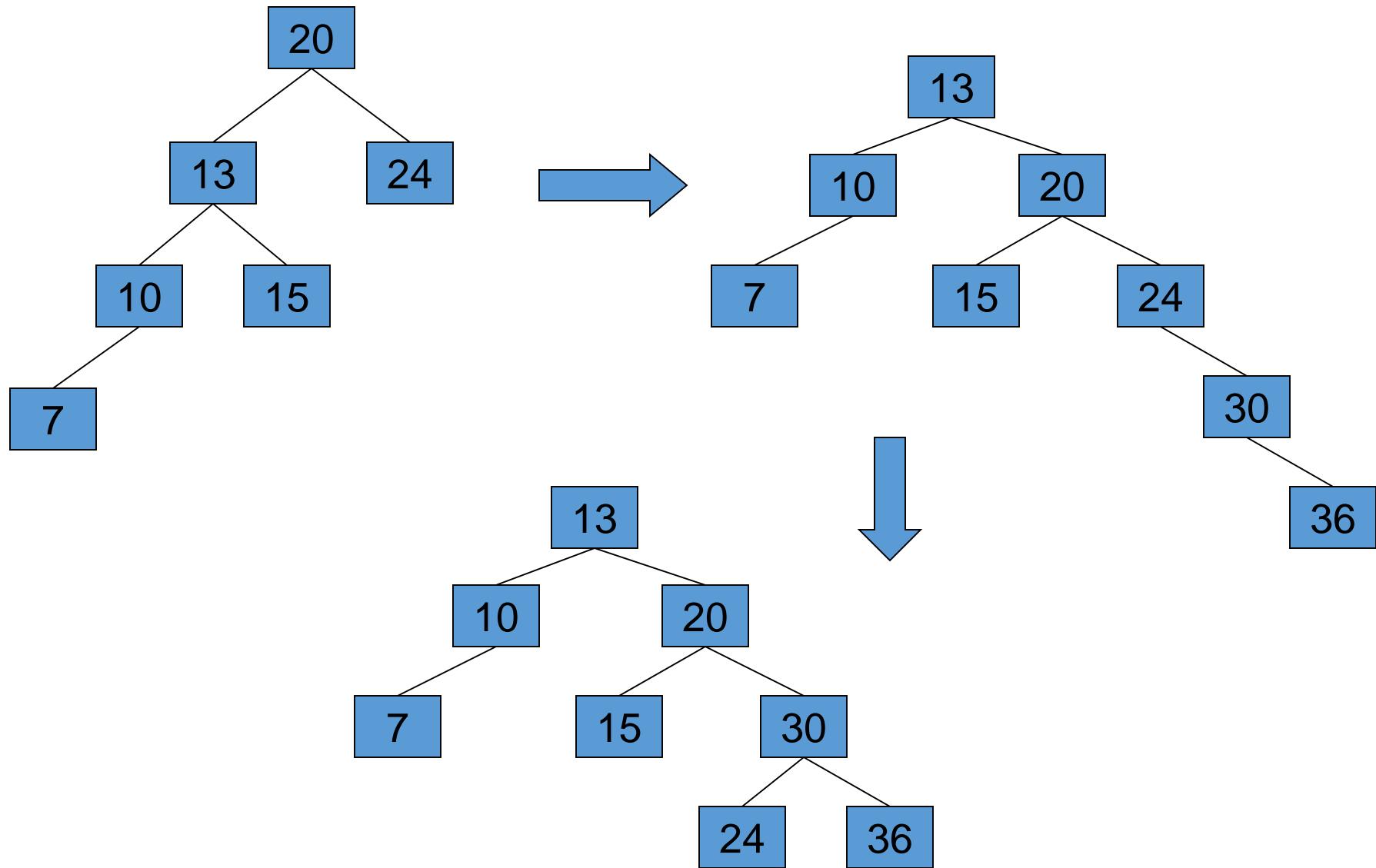


- Лабораторийн даалгавар: Дараах утгуудаар AVL модыг бүтээ. Үүнд: 15, 20, 24, 10, 13, 7, 30, 36, 25

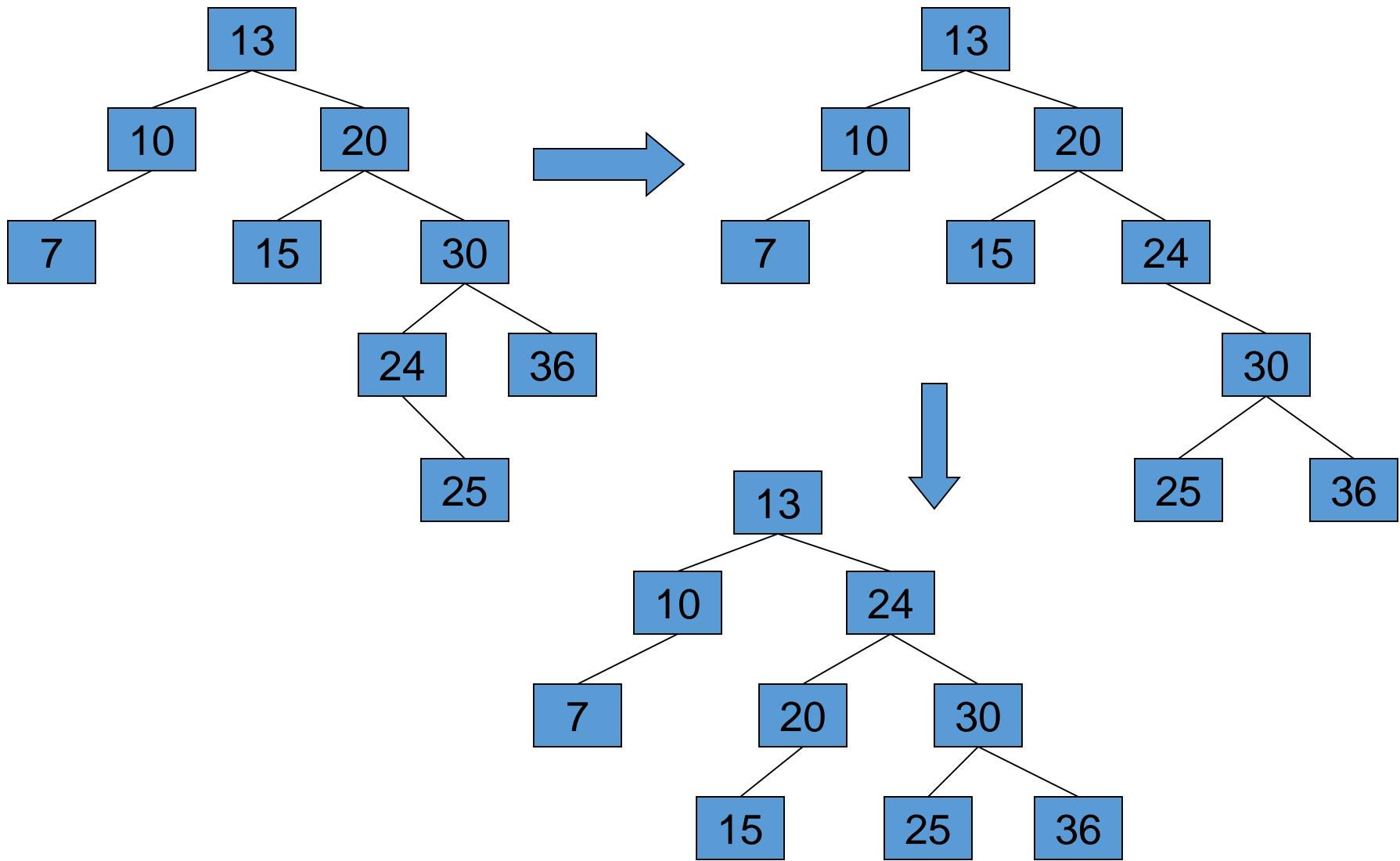
15, 20, 24, 10, 13, 7, 30, 36, 25



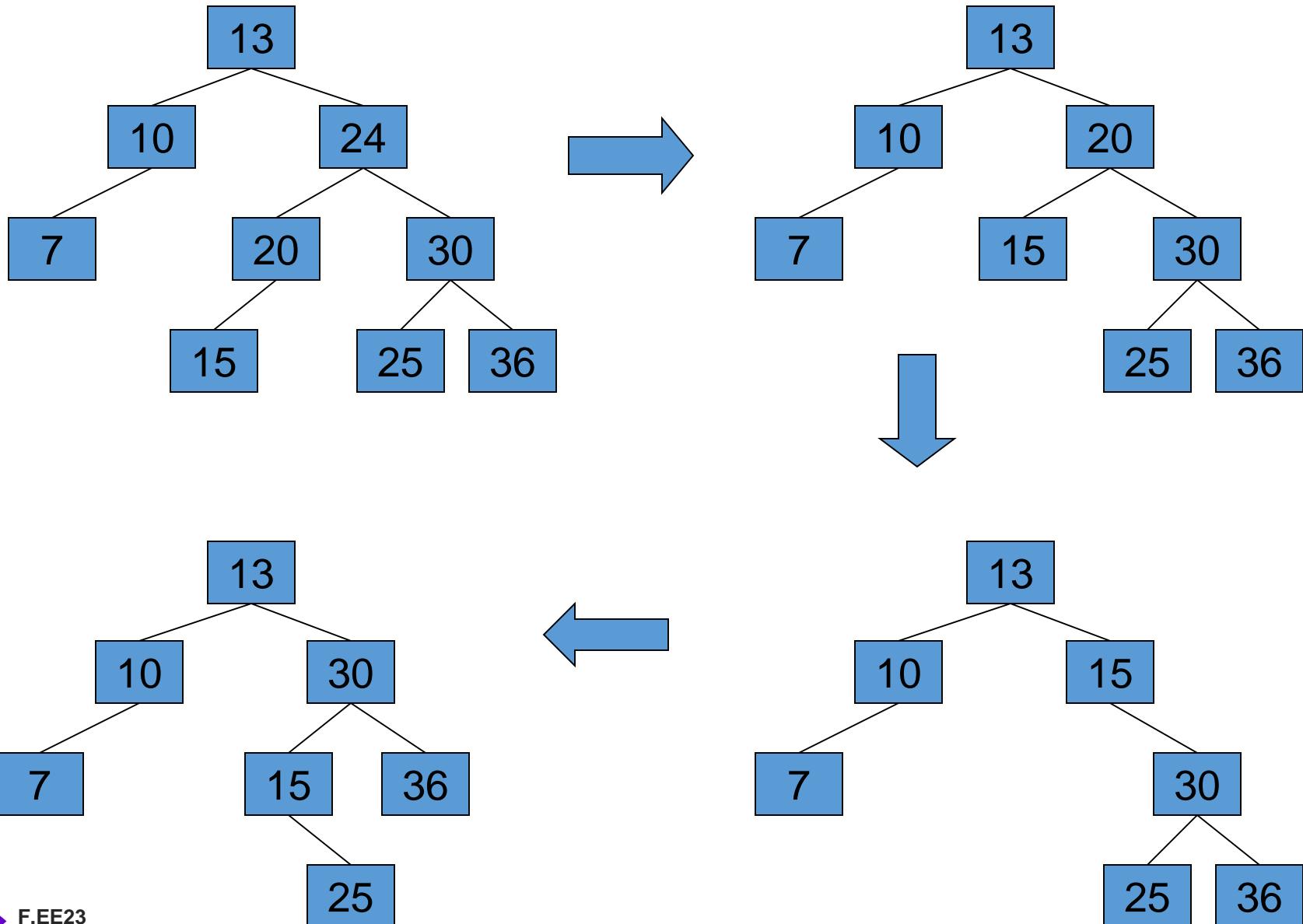
15, 20, 24, 10, 13, 7, 30, 36, 25



15, 20, 24, 10, 13, 7, 30, 36, 25

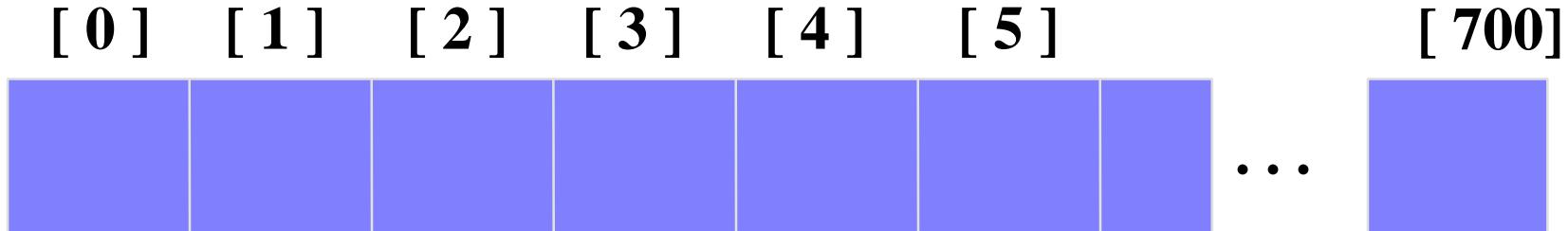


AVL мадноос 24 ба 20-ыг утсга.



What is a Hash Table ? / Хэш хүснэгт гэж юу вэ?

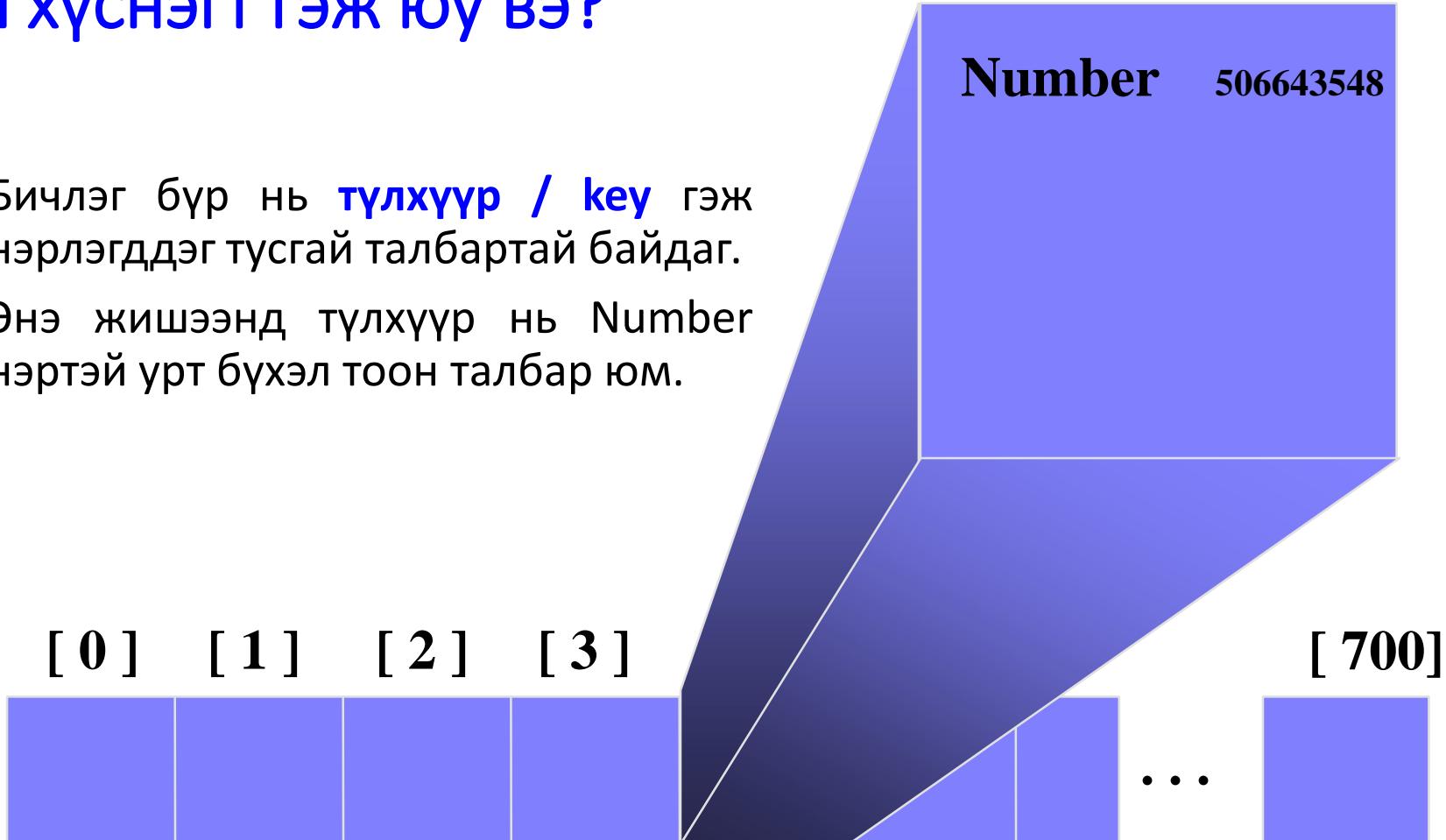
- Хэш хүснэгтийн хамгийн энгийн төрөл бол массив бичлэг юм.
- Энэ жишээнд 701 бичлэг байна.



Бичлэгийн массив

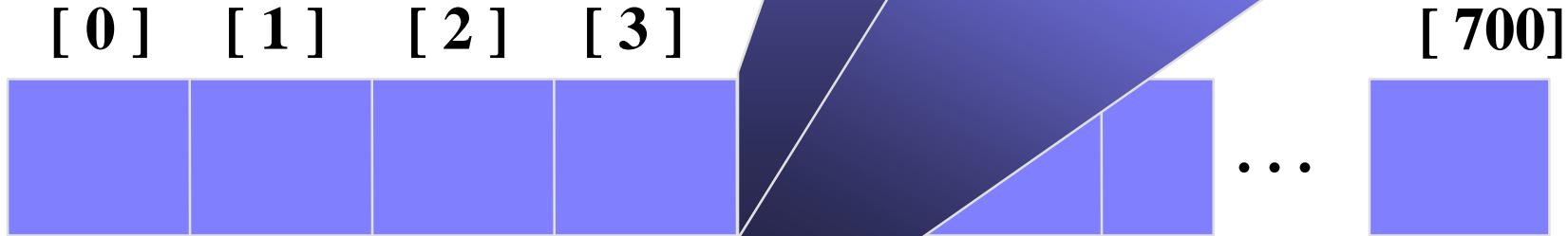
Хэш хүснэгт гэж юу вэ?

- Бичлэг бүр нь **түлхүүр / key** гэж нэрлэгддэг тусгай талбартай байдаг.
- Энэ жишээнд түлхүүр нь Number нэртэй урт бүхэл тоон талбар юм.



Хэш хүснэгт гэж юу вэ?

- Энэ дугаар нь тухайн хүний таних дугаар байж болох бөгөөд бусад бичлэгт тухайн хүний талаарх мэдээлэл байна.



Number 506643548



ХЭШ ХҮСНЭГТ ГЭЖ ЮУ ВЭ?

- When a hash table is in use, some spots contain valid records, and other spots are "empty".
- Хэш хүснэгт ашиглагдаж байх үед зарим үүр нь хүчинтэй бичлэгүүдийг агуулж, бусад үүрнүүд нь "хоосон" байдаг.

[0] [1] [2] [3] [4] [5] [700]



Шинэ бичлэг нэмэх

- Шинэ бичлэг оруулахын тулд key түлхүүрийг ямар нэгэн байдлаар массивын индекс болгон хувиргах ёстой.
- Индексийг түлхүүрийн хэш үтга гэж нэрлэдэг.



[0]

[1]

[2]

[3]

[4]

[5]

[700]



Number 281942902

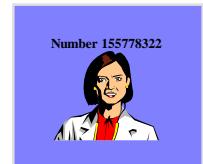


Number 233667136



Number 506643548

...



Number 155778322

Шинэ бичлэг нэмэх

- Хэш утгыг үүсгэх ердийн арга:



(580625685 mod 701) гэж юу вэ?

[0]

[1]

[2]

[3]

[4]

[5]

[700]



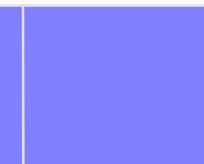
Number 281942902



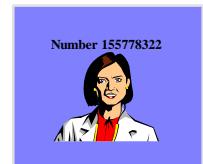
Number 233667136



Number 506643548



...



Number 155778322

Шинэ бичлэг нэмэх

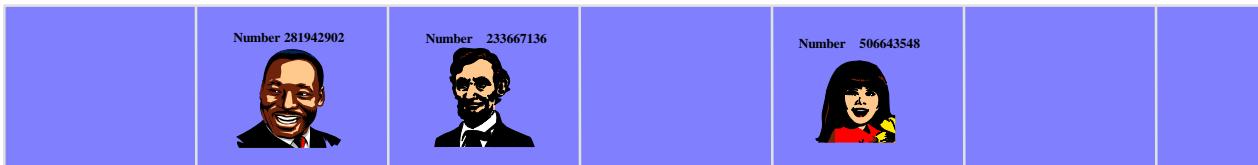
- Хэш утгыг үүсгэх ердийн арга:



(580625685 mod 701) гэж юу вэ?

3

[0] [1] [2] [3] [4] [5] [700]



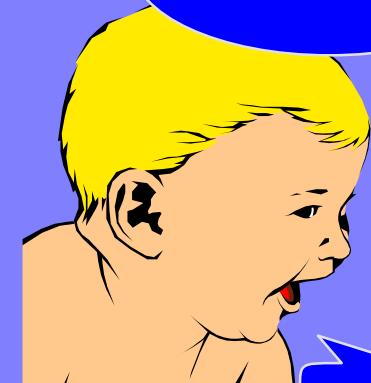
...



Шинэ бичлэг нэмэх

- Шинэ бичлэгийн байршилд хэш утгыг ашиглана.

Number 580625685



[3]

[0]

[1]

[2]

[700]



Number 281942902



Number 233667136

...

...



Number 155778322

Шинэ бичлэг нэмэх

- Шинэ бичлэгийн байршилд хэш утгыг ашиглана.
- Хэш утгыг үргэлж бичлэгийн байршлыг олоход ашигладаг.

[0]

[1]

[2]

[3]

[4]

[5]

[700]



Number 281942902



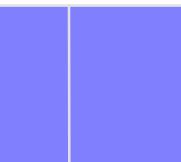
Number 233667136



Number 580625685



Number 506643548



...



Number 155778322

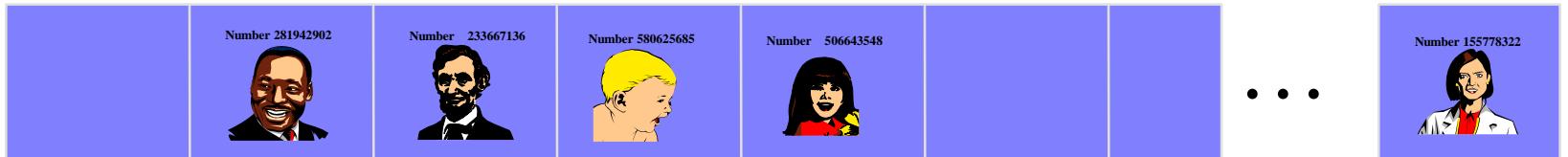
Collisions / Мөргөлдөөн

- Энд оруулах өөр нэг шинэ бичлэг байна, хэш утга нь 2.
- Заримдаа хоёр өөр бичлэг нь ижил хэш утгатай байж болно.



Миний хэш
утга нь [2].

[0] [1] [2] [3] [4] [5] ... [700]



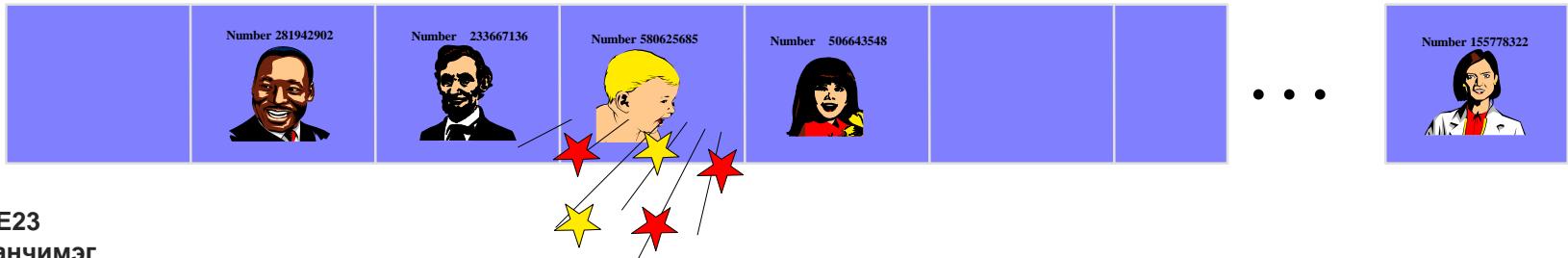
Collisions / Мөргөлдөөн

- Үүнийг мөргөлдөөн гэж нэрлэдэг, учир нь [2] дээр өөр нэг хүчинтэй бичлэг байгаа.

Мөргөлдөөн гарах үед,
тухайн утгыг байрлуулах
хоосон нүд олохын тулд
урагшилна.



[0] [1] [2] [3] [4] [5] ... [700]



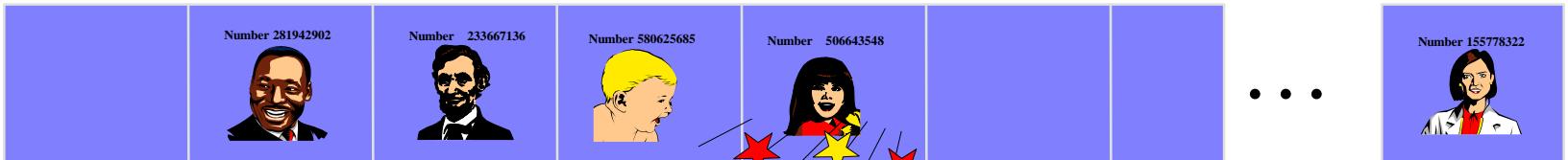
Collisions / Мөргөлдөөн

- Үүнийг мөргөлдөөн гэж нэрлэдэг, учир нь [2] дээр өөр нэг хүчинтэй бичлэг байгаа.

Мөргөлдөөн гарах үед,
тухайн утгыг байрлуулах
хүртэл урагшaa хоосон
газар олно.



[0] [1] [2] [3] [4] [5] ... [700]



Collisions / Мөргөлдөөн

- Үүнийг мөргөлдөөн гэж нэрлэдэг, учир нь [2] дээр өөр нэг хүчинтэй бичлэг байгаа.

Мөргөлдөөн гарах үед,
тухайн утгыг байрлуулах
хүртэл урагшaa хоосон
газар олно.

[0] [1] [2] [3] [4] [5] ... [700]



Collisions / Мөргөлдөөн

- Үүнийг мөргөлдөөн гэж нэрлэдэг, учир нь [2] дээр өөр нэг хүчинтэй бичлэг байгаа.
- Шинэ бичлэгийн хэш үтгыг үргэлж өхний боломжтой хоосон газар байрлуулна.

Шинэ бичлэг
байршуулах
хоосон газар олдлоо.

[0]

[1]

[2]

[3]

[4]

[5]

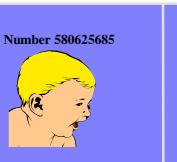
[700]



Number 281942902



Number 233667136



Number 580625685



Number 506643548



Number 701466868

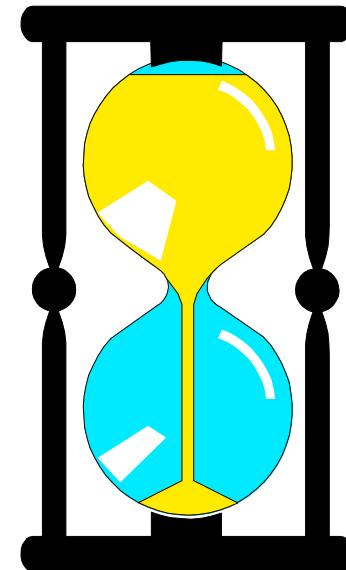
...



Number 155778322

Асуулт

Хэрэв мөргөлдөхгүй бол таныг энэ
Хүснэгтийн хаана байрлуулах вэ?
Нийгмийн даатгалын дугаар эсвэл өөр
дуртай дугаараа ашиглана уу.



[0]

[1]

[2]

[3]

[4]

[5]

[700]



Number 281942902



Number 233667136



Number 580625685



Number 506643548



Number 701466868

...



Number 155778322

Түлхүүрийг хайх

- Түлхүүрт хавсаргасан өгөгдлийг маш хурдан олох боломжтой.
- Түлхүүр дээр нь түлгүүрлан тодорхой зүйлийг хайх нь нэлээд хялбар байдаг.

Number 701466868

[0]

[1]

[2]

[3]

[4]

[5]

[700]



Number 281942902



Number 233667136



Number 580625685



Number 506643548



Number 701466868

...



Number 155778322

Түлхүүрийг хайх

- Хэш утгыг тооцоол.
- Түлхүүрийн массивын байршлыг шалганаа уу.

Number 701466868

Миний хэш
утга нь [2].

Би биш

[0]

[1]

[2]

[3]

[4]

[5]

[700]



Number 281942902



Number 233667136



Number 580625685



Number 506643548



Number 701466868

...



Number 155778322

Түлхүүрийг хайх

- Түлхүүрийг олох эсвэл хоосон газар хүрэх хүртлээ урагшил.
- ... хэрэв дараагийн байршил бидний хайж байгаа газар биш бол цаашаа ...

Number 701466868

Миний хэш
утга нь [2].

Би биш

[0]

[1]

[2]

[3]

[4]

[5]

[700]



Number 281942902



Number 233667136



Number 580625685



Number 506643548



Number 701466868

...



Number 155778322

Түлхүүрийг хайх

- Түлхүүрийг олох эсвэл хоосон газар хүрэх хүртлээ урагшил.
- Хүссэн түлхүүрээ олох хүртлээ урагшил ...

Number 701466868

Миний хэш
утга нь [2].

Би биш.

[0]

[1]

[2]

[3]

[4]

[5]

[700]



Number 281942902



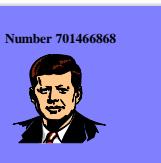
Number 233667136



Number 580625685



Number 506643548



Number 701466868

...



Number 155778322

Түлхүүрийг хайх

- Түлхүүрийг олох эсвэл хоосон газар хүрэх хүртлээ урагшил.
- Энэ тохиолдолд бид түлхүүрийг [5] байршлаас олсон.

Number 701466868

Миний хэш
утга нь [2].

Тийм!

[0]

[1]

[2]

[3]

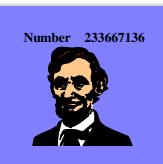
[4]

[5]

[700]



Number 281942902



Number 233667136



Number 580625685



Number 506643548



Number 701466868

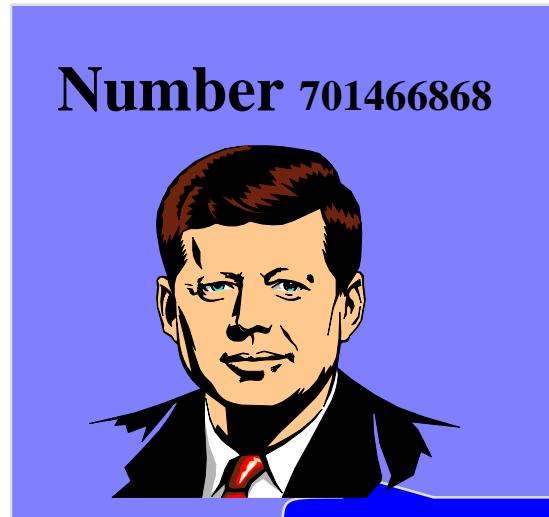
...



Number 155778322

Түлхүүрийг хайх

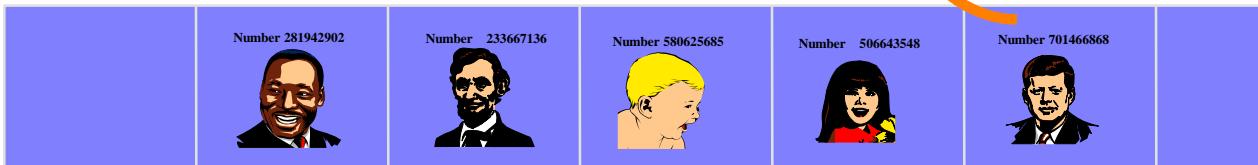
- Тухайн зүйл олдвол мэдээллийг шаардлагатай газар руу хуулж болно.



Миний хэш
утга нь [2].

Тийм!

[0] [1] [2] [3] [4] [5] [700]



...



Deleting a Record / Бичлэг үстгах

- Бичлэгийг хэш хүснэгтээс үстгаж болно.

Намайг
үстга.

[0]

[1]

[2]

[3]

[4]

[5]

[700]

...



Deleting a Record / Бичлэг үстгах

- Бичлэгийг хэш хүснэгтээс үстгаж болно.
- Гэхдээ энэ нь хайлт хийхэд саад учруулж болзошгүй тул байршлыг энгийн "хоосон нүд" болгож болохгүй.

[0]

[1]

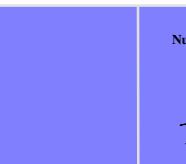
[2]

[3]

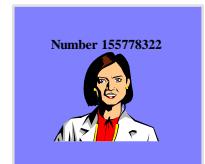
[4]

[5]

[700]



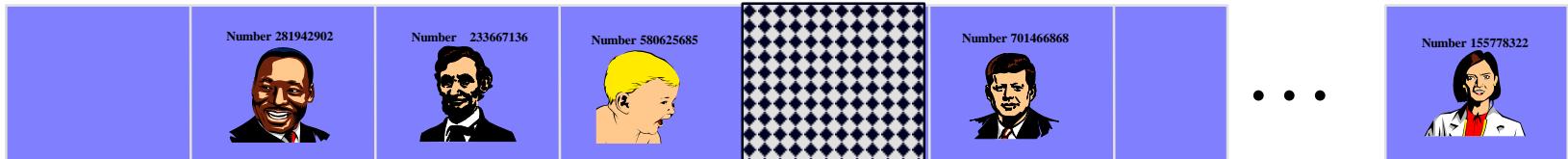
...



Deleting a Record / Бичлэг үстгах

- Бичлэгийг хэш хүснэгтээс үстгаж болно.
- Гэхдээ энэ нь хайлт хийхэд саад учруулж болзошгүй тул байршлыг энгийн "хоосон нүд" болгож болохгүй.
- Хайлтын явцад тухайн цэгт ямар нэгэн зүйл байсан гэдгийг олж мэдэхийн тулд тухайн байршлыг ямар нэг тусгай аргаар тэмдэглэсэн байх ёстой.

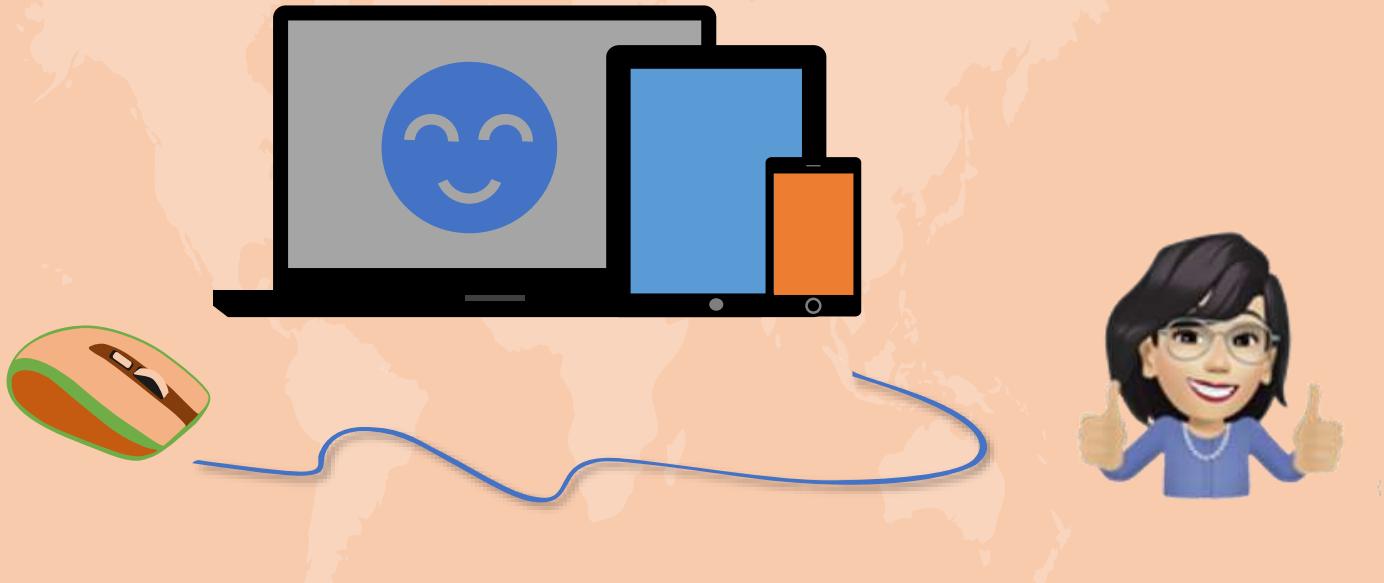
[0] [1] [2] [3] [4] [5] [700]





Дүгнэлт

- Хэш хүснэгтүүд нь түлхүүр бүхий бичлэгийн цуглуулгыг хадгалдаг.
- Бичлэгийн байршил нь бичлэгийн түлхүүрийн хэш утгаас хамаарна.
- Мөргөлдөх үед дараагийн боломжтой байршлыг ашиглана.
- Тодорхой түлхүүр хайх нь ерөнхийдөө хурдан байдаг.
- Бичлэгийг үстгах үед тухайн нүдийг / газрыг тусгай аргаар тэмдэглэсэн байх ёстой бөгөөд хайлтууд тухайн нүдийг ашиглаж байсныг мэдэх болно.



Хичээлээ шимтэн судлах оюутан танд баярлалаа.
Сурлагын өндөр амжилт хүсье.