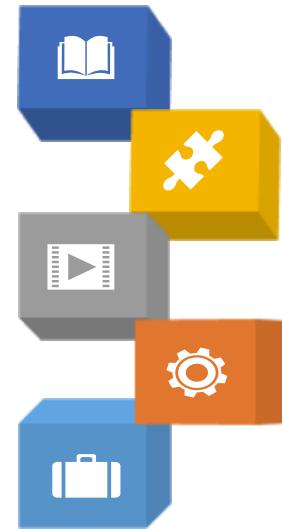




ХӨДӨЛМӨРИЙН ГАВЬЯАНЫ УЛААН ТУГИЙН ОДОНТ  
МОНГОЛ УЛСЫН ШИНЖЛЭХ УХААН  
ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ  
МЭДЭЭЛЭЛ, ХОЛБООНЫ ТЕХНОЛОГИЙН СУРГУУЛЬ



# Welcome to F.CSM 203

## Өгөгдлийн бүтэц ба алгоритм

### ЛЕКЦІХ



F.EE23  
Г.Ганчимэг

# СЭДЭВ. DATA STRUCTURES AND ALGORITHMS

## Агуулга



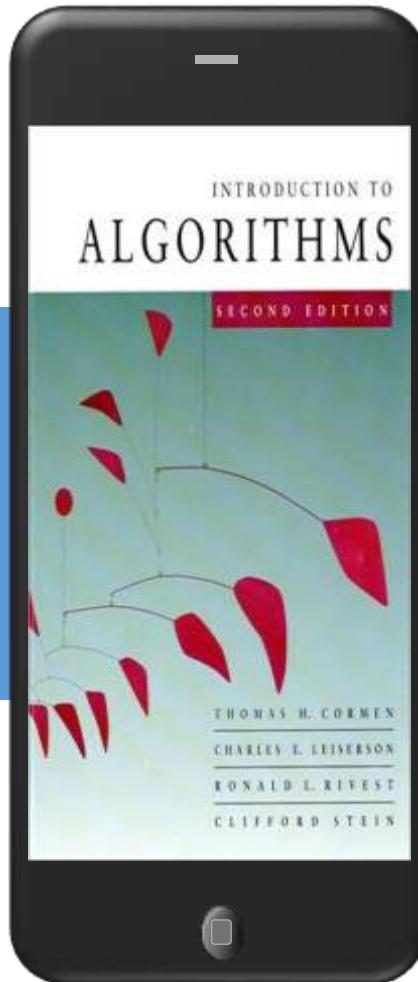
Мод



Хоёртын  
хайлтын мод



AVL  
Тэнцвэрт мод



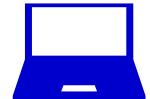
BFS - DFS



Hash table



Жишээ ба  
код

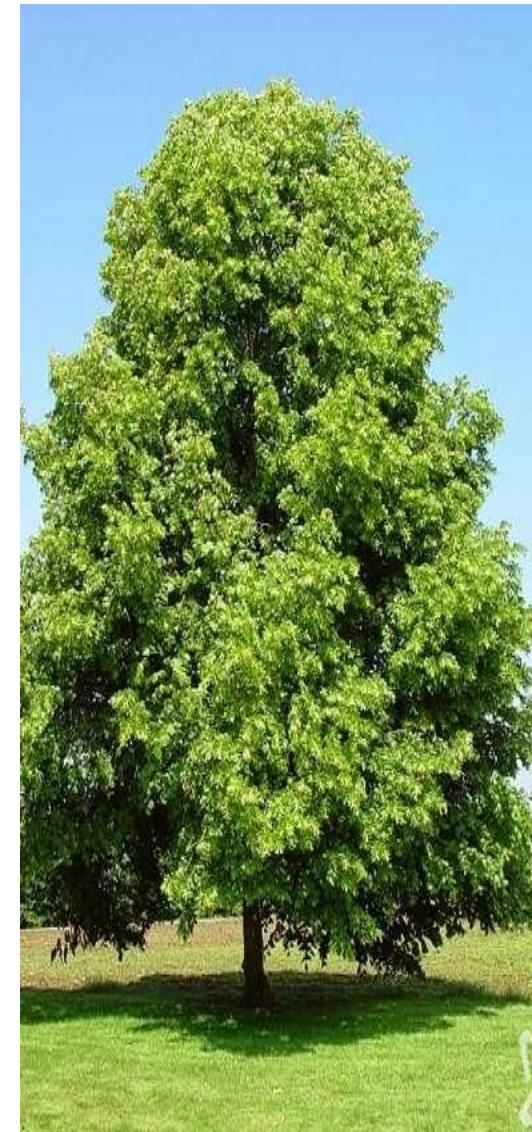


# Мод

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

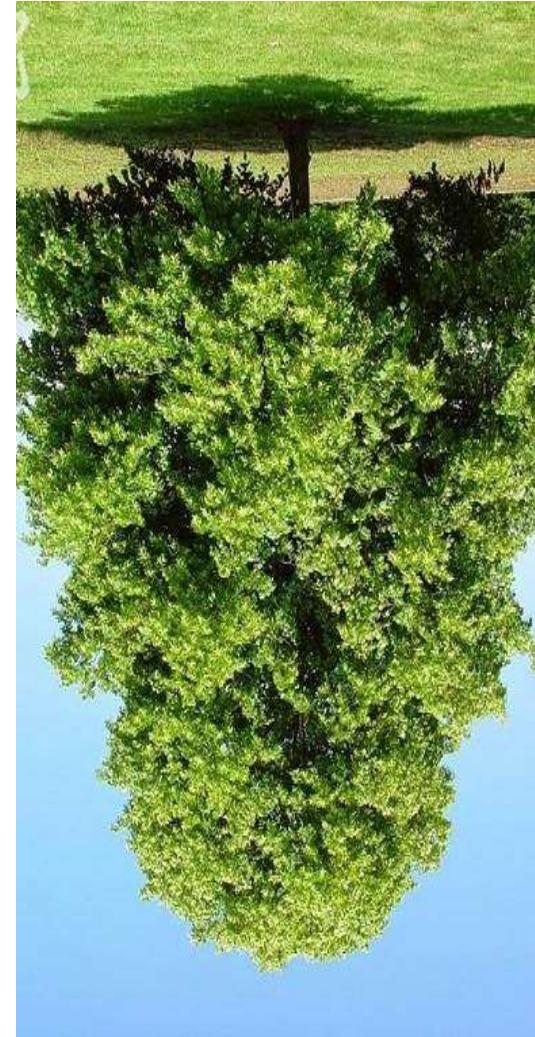
"A tree may grow a thousand  
feet tall, but its leaves will  
return to its roots."

Chinese Proverb



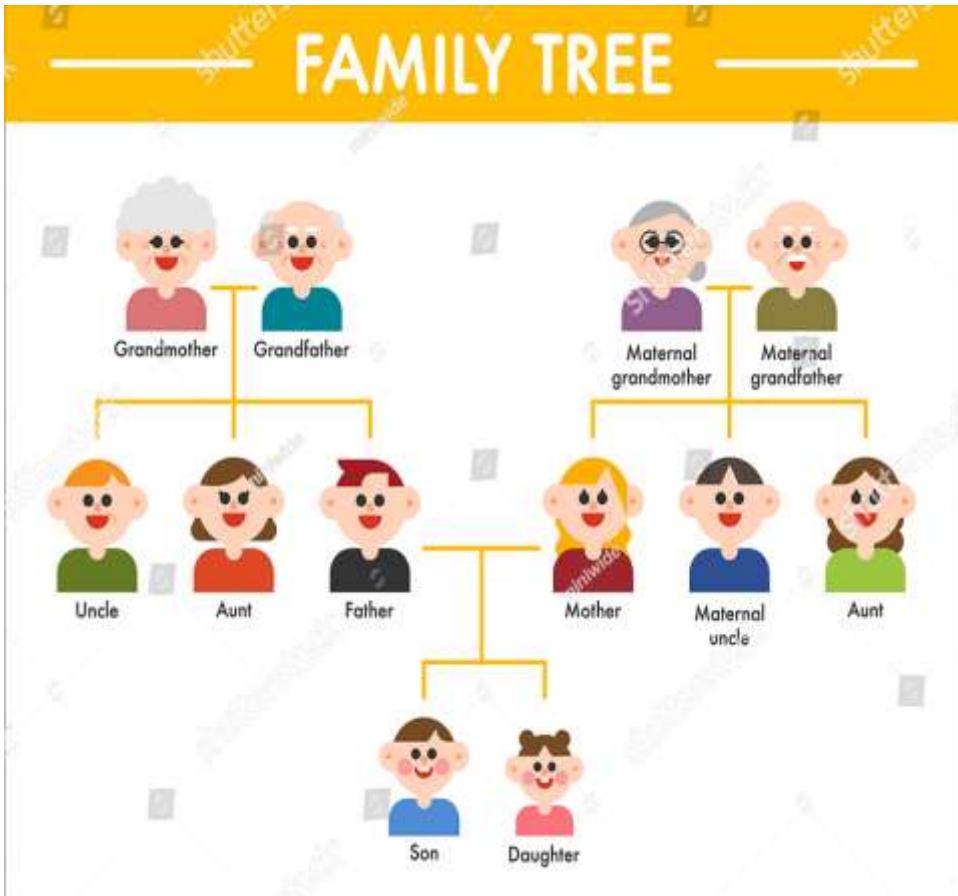
# Мод

- 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.



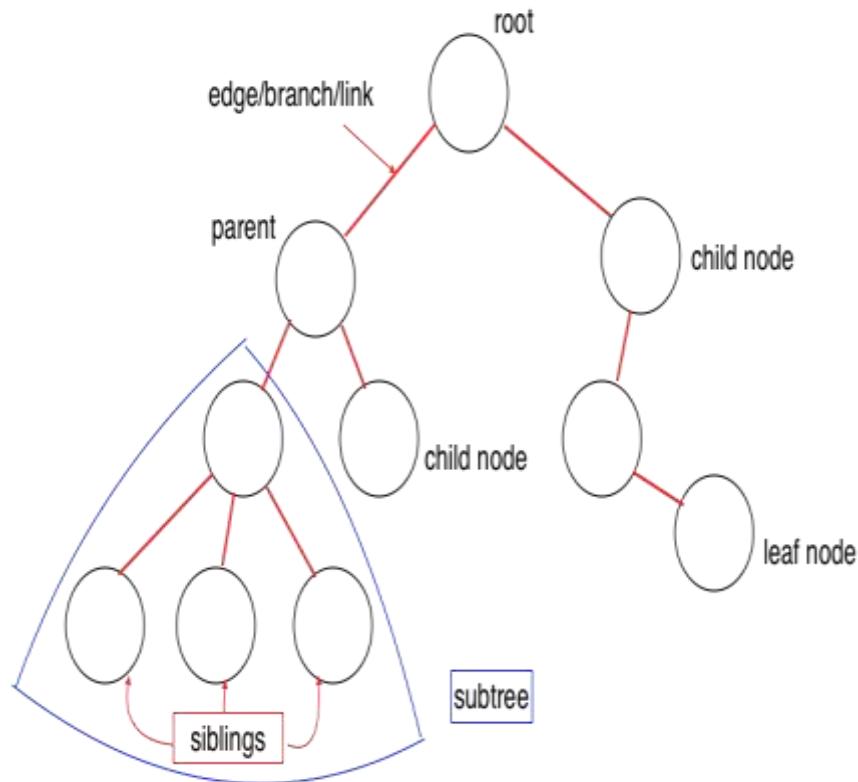
# Мод

- 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.

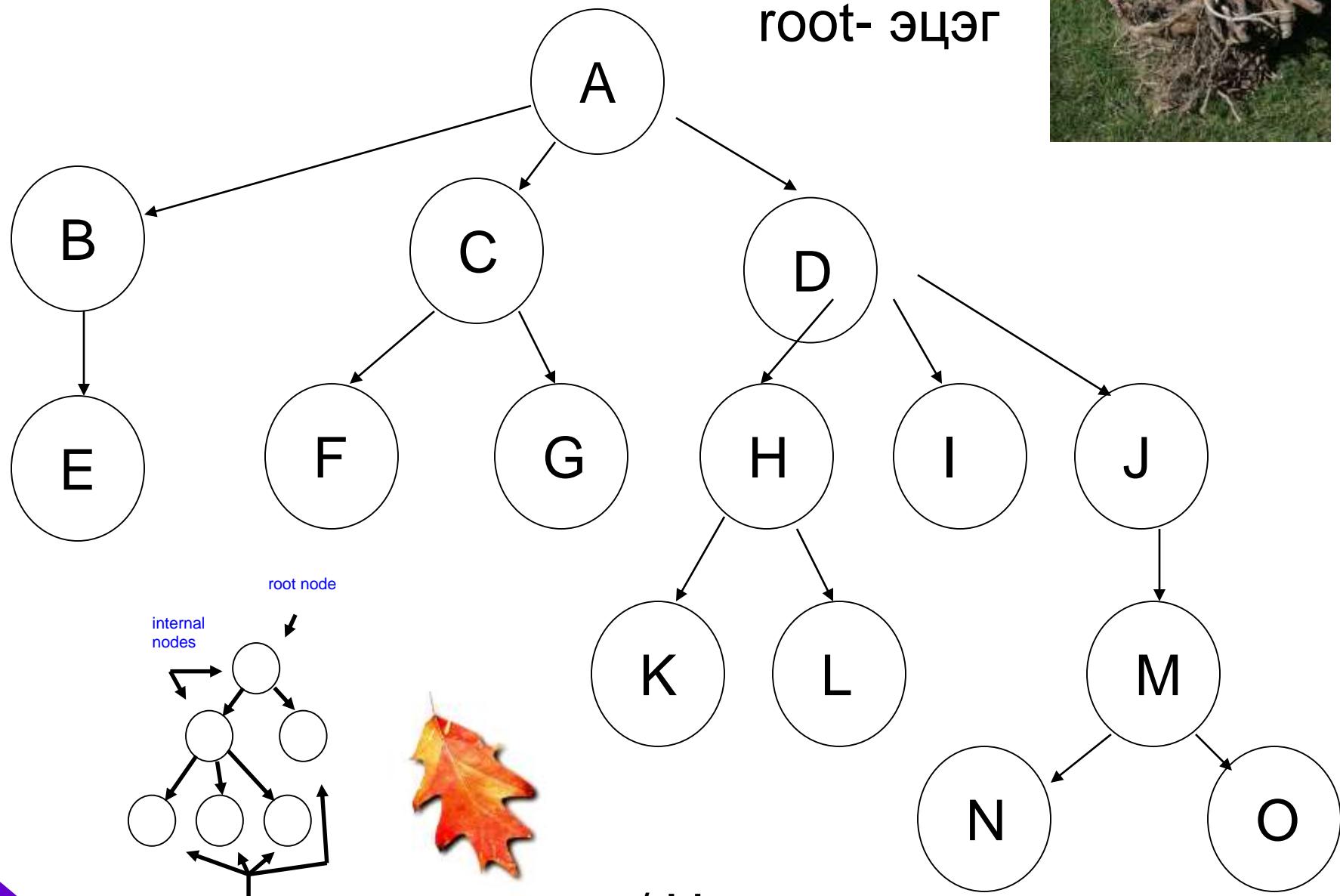


# Мод

- 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

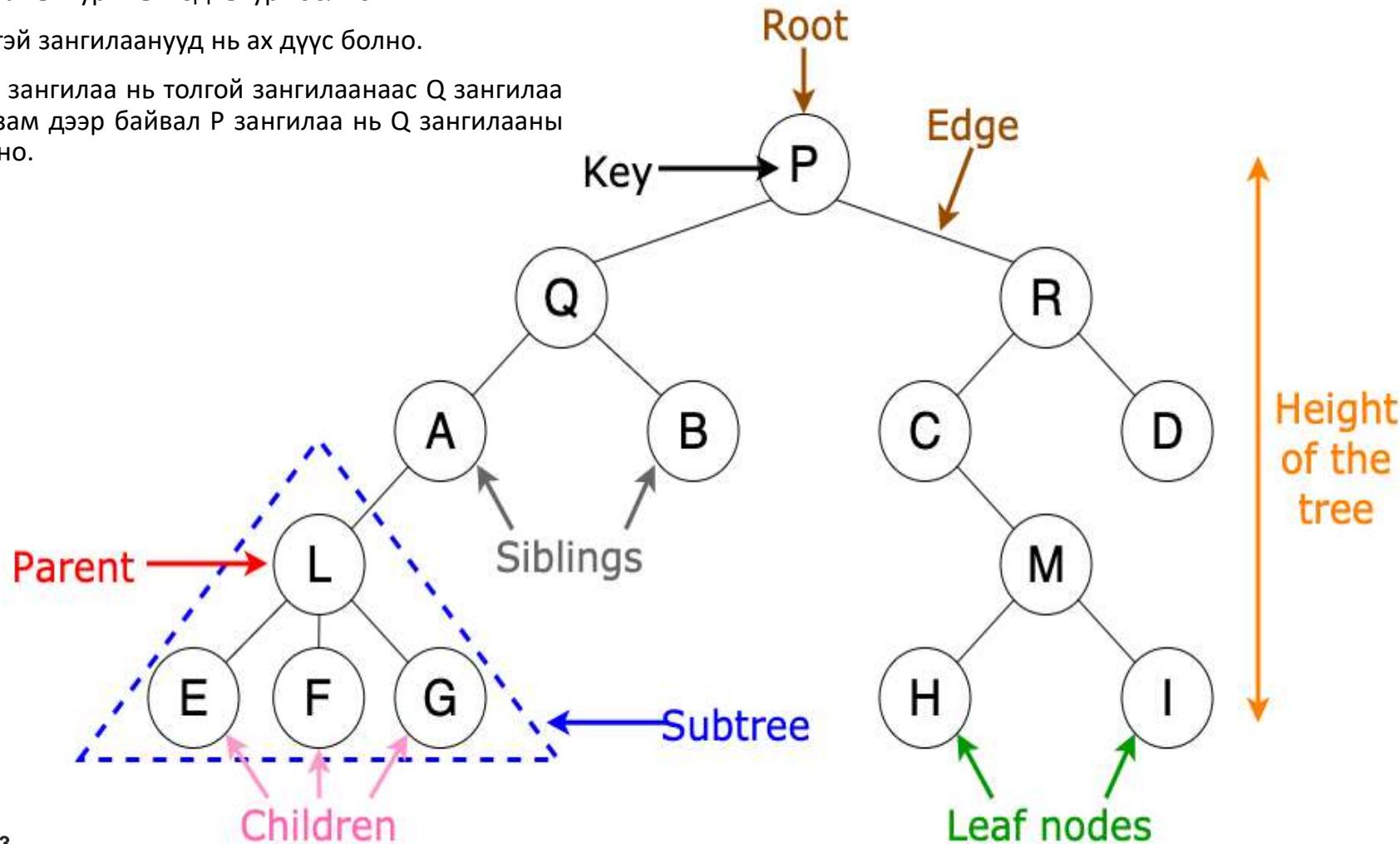


# Binary Tree / Хоёртын мод

- 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.

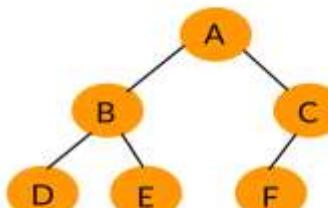
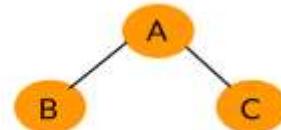
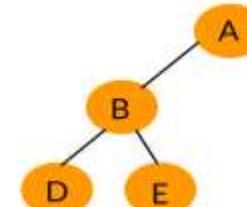
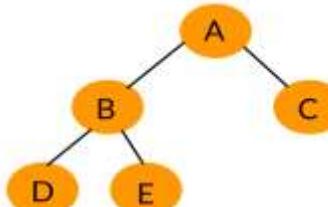
- Чиглэлтэй холбоос нь эцгээс хүү рүү чиглэж холбоно.
- Толгой зангилаа нь эцэггүй байна. Модонд ганц л толгой зангилаа байна.
- Навчин зангилаа нь хүүхэдгүй байна.
- Толгой зангилаанаас хамгийн доод талын зангилаа хүртэлх замын урт нь модны урт болно.
- Нэг эцэгтэй зангилаанууд нь ах дүүс болно.
- Хэрвээ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

## Модны үндэс

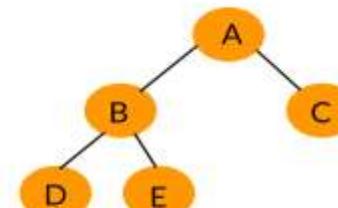


# Binary Tree / Хоёртын мод

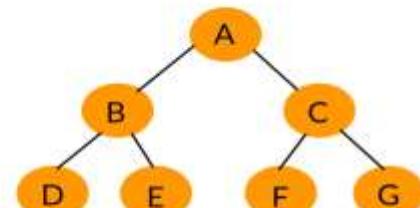
- Зангилааны хоёр дэд зангилааг зүүн дэд зангилаа ба баруун дэд зангилаа гэж ялгадаг. Хоёртын модны зангилаа нь нэг эсвэл хоёр хүүхэд зангилаа агуулж болно.
- Модны хамгийн сүүлчийн түвшингээс бусад түвшинд дотоод зангилаагаар гүйцэд дүүргэсэн бол түүнийг **дүүрэн хоёртын мод /Full Binary Tree/** гэнэ.
- 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

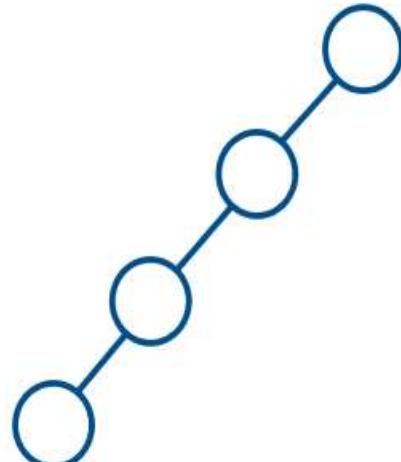


Perfect Binary Tree

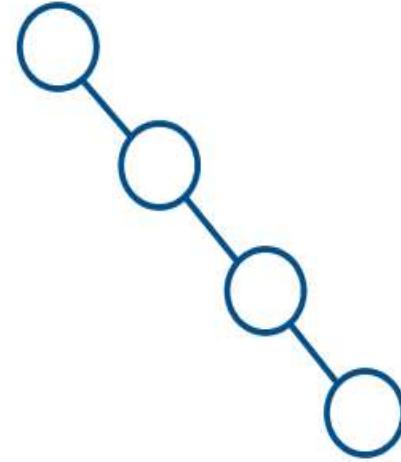
# Skewed Binary Tree / Ташуу мод

- 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.

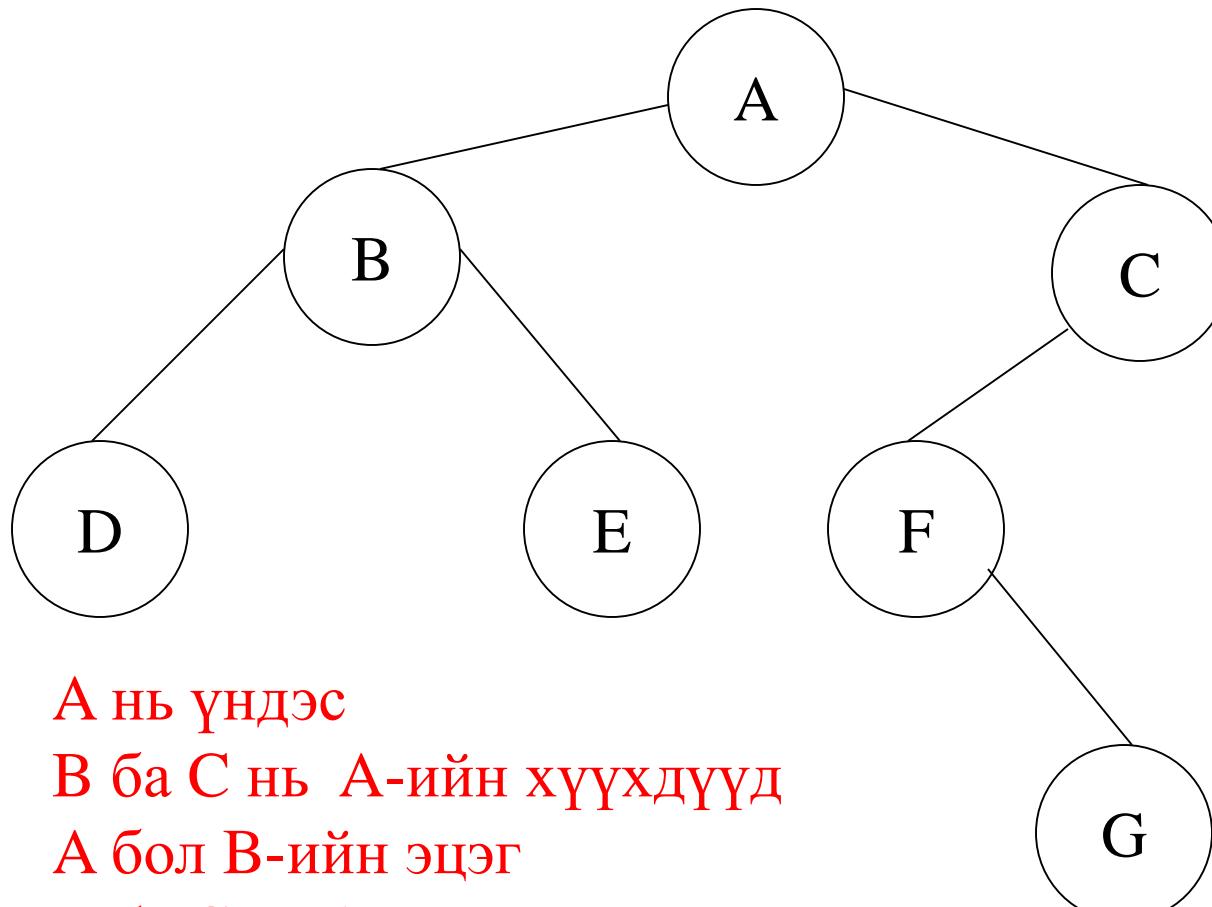
- Left skewed binary tree



- Right skewed binary tree



# Binary Tree / Хоёртын модны чанар

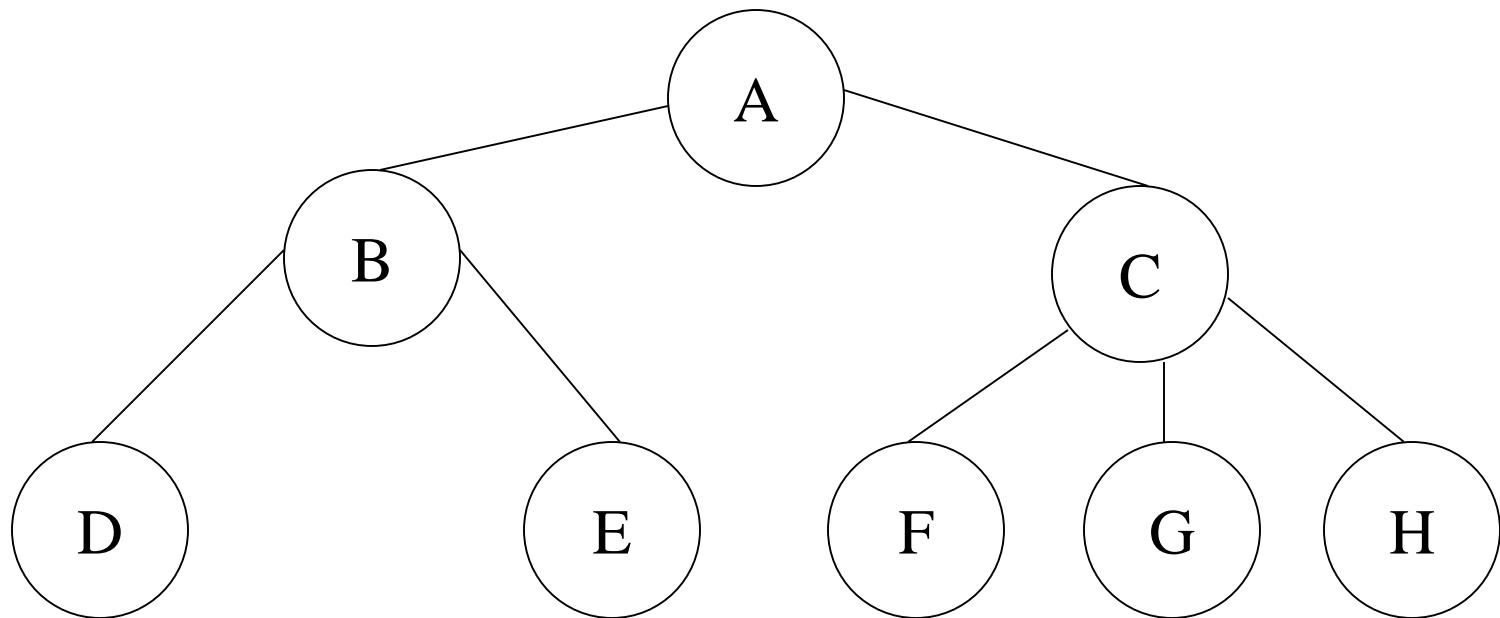


- А нь үндэс
- В ба С нь А-ийн хүүхдүүд
- А бол В-ийн эцэг
- В ба С нь А үндэсний дэд моднууд
- D, E, G нь навчнууд

# Хоёртын модны чанар

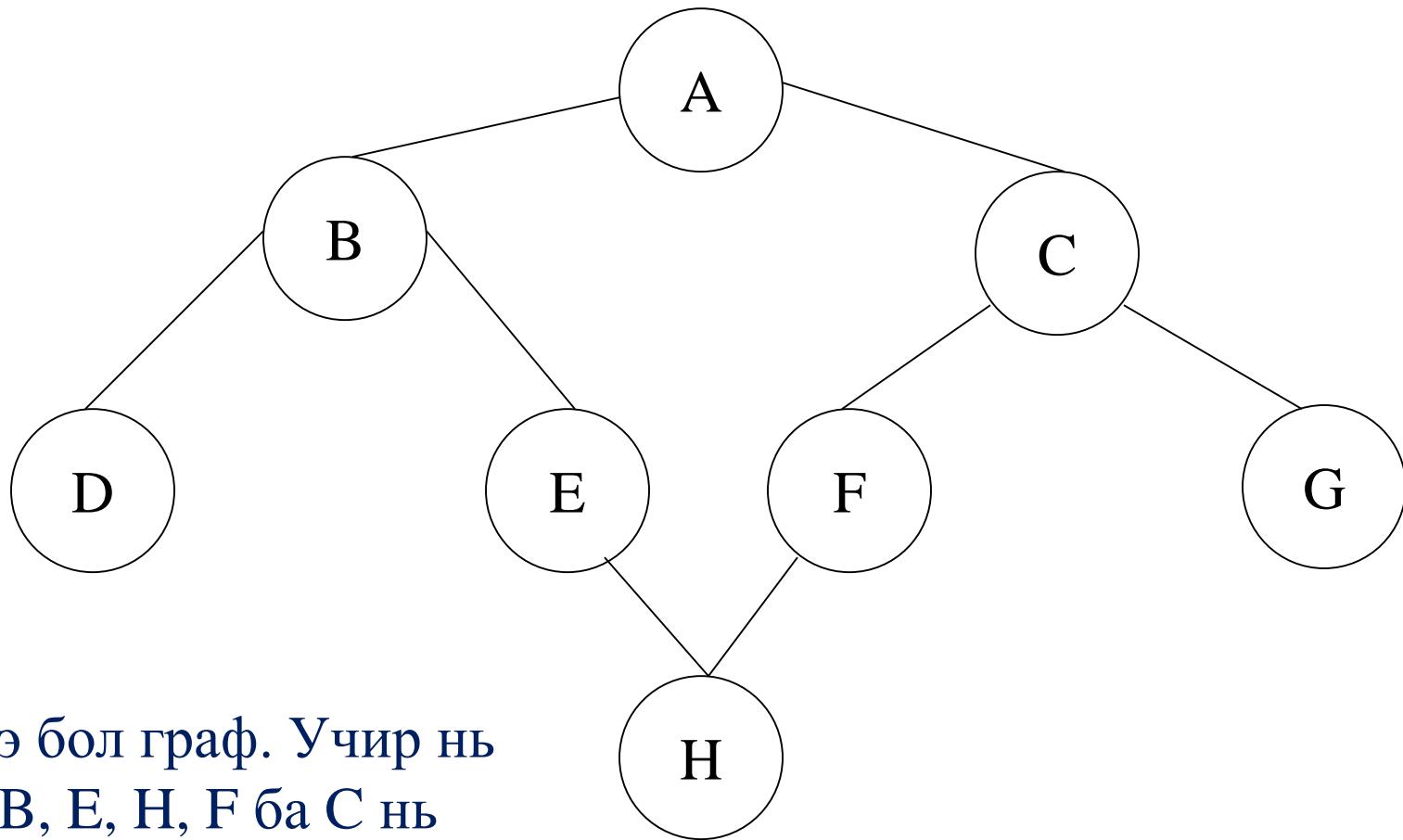
- **Чанар 1:** Модны аливаа хоёр зангилааг холбох ганц зам байна. Аливаа хоёр зангилааны хувьд ядаж нэг ерөнхий зангилаа олдоно. Ийм ерөнхий эх зангилаанаас уг хоёр зангилаанд хүрэх зам нь давтагдашгүй бөгөөд эдгээрийг нийлүүлснээр зөвхөн ганц зам олдоно.
- **Чанар 2:** N зангилаа бүхий мод N-1 холбоостой 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



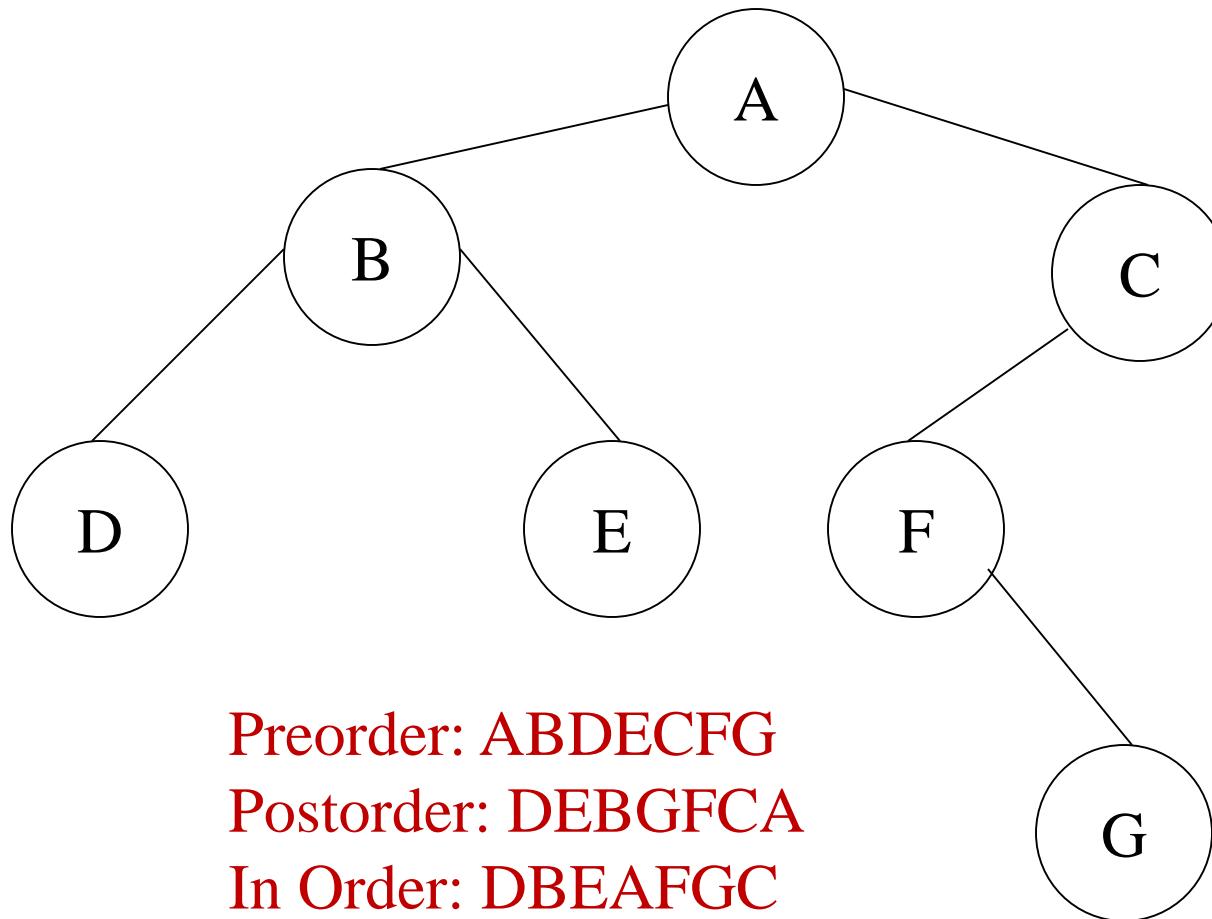
Энэ бол граф. Учир нь  
A, B, E, H, F ба C нь  
хоорондоо холбогдоно.

# 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<sub>F</sub>G

Postorder: DEBGFC<sub>A</sub>

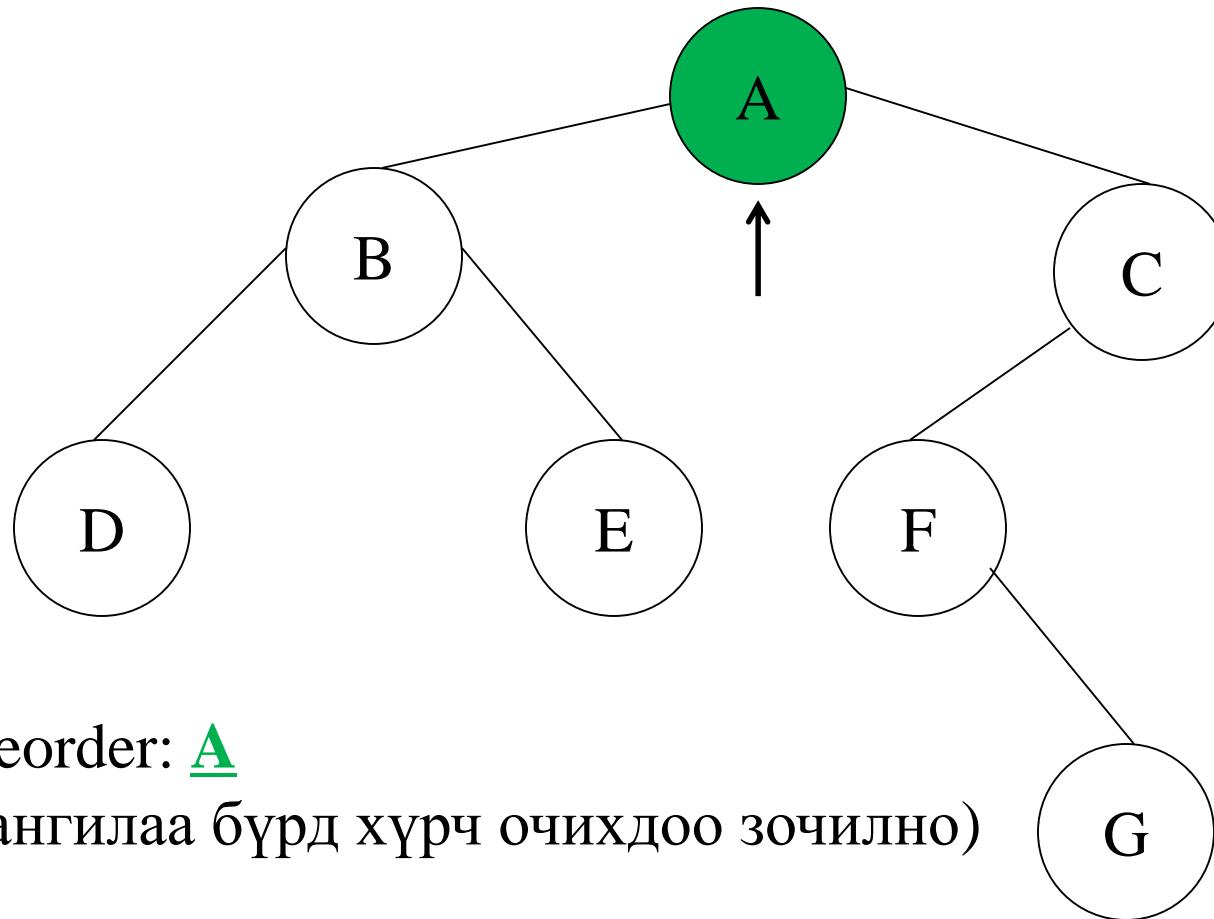
In Order: DBEA<sub>F</sub>GC

# 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.

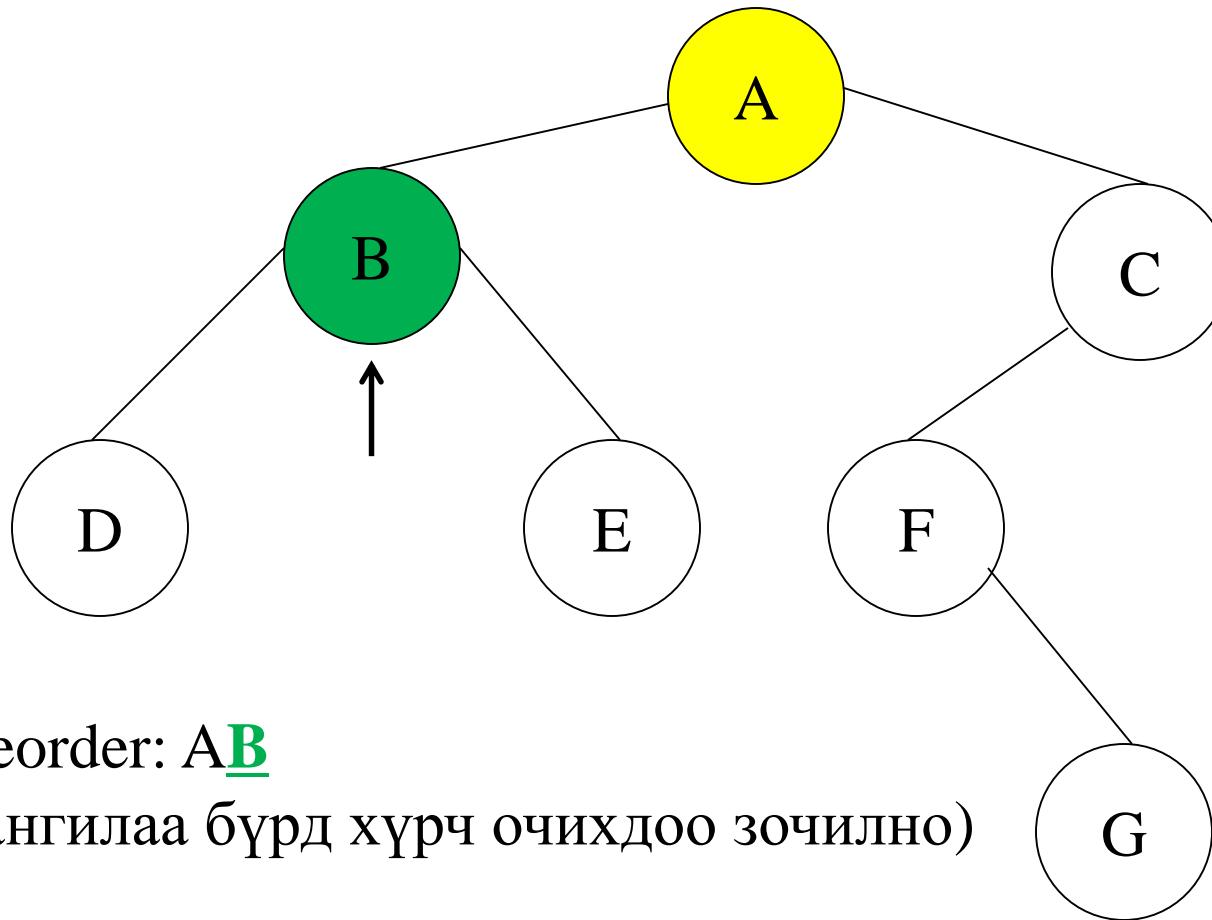
# Preorder нэвтрэлтийн жишээ



Preorder: A

(зангилаа бүрд хүрч очихдоо зочилно)

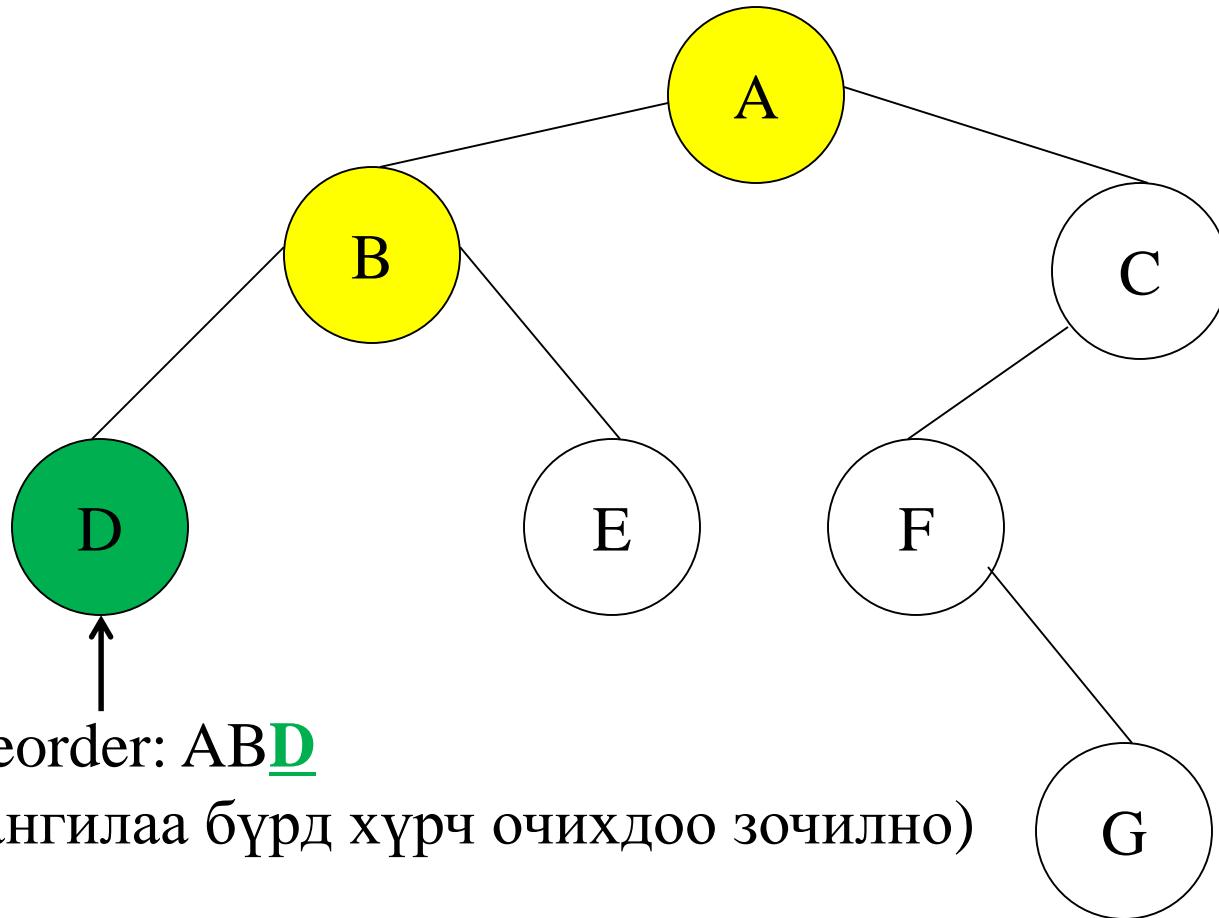
# Preorder нэвтрэлтийн жишээ



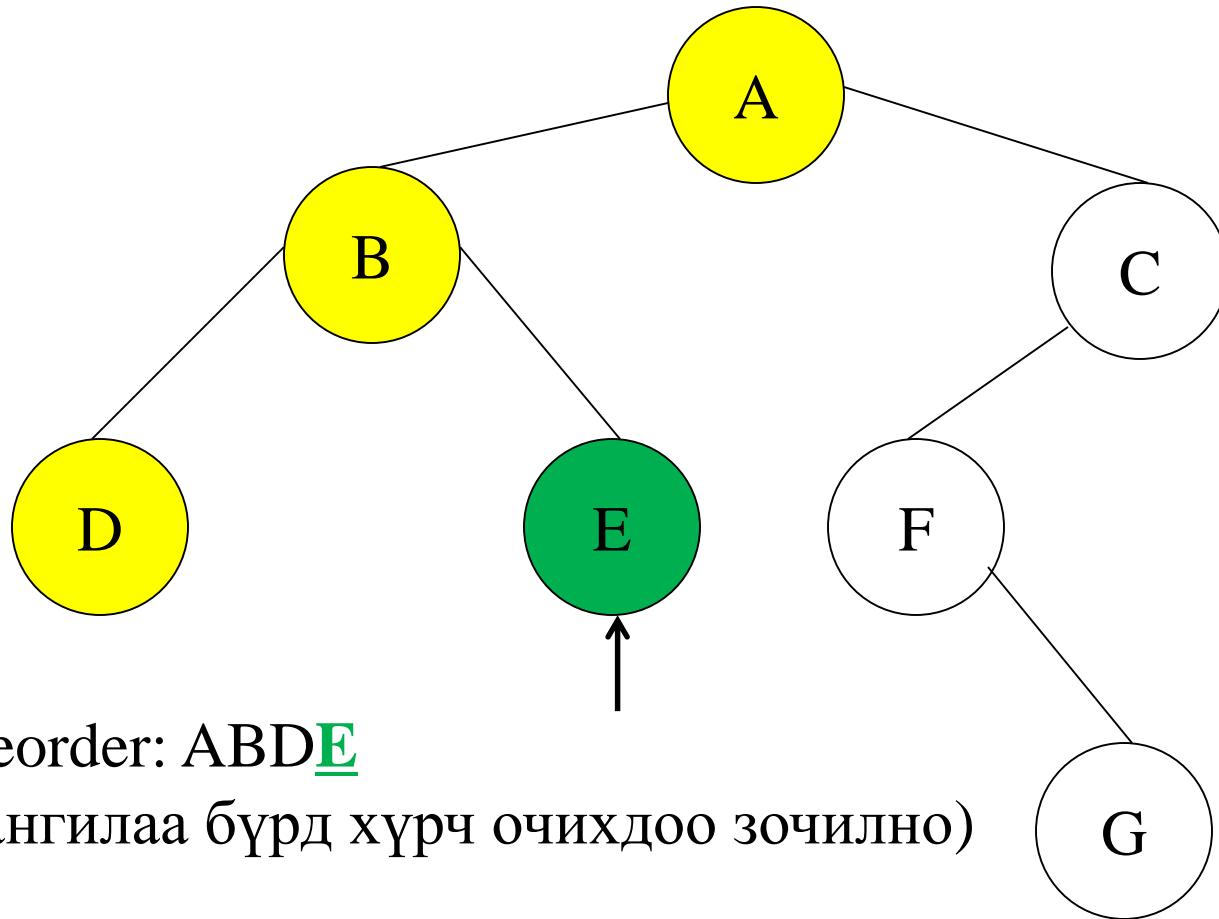
Preorder: AB

(зангилаа бүрд хүрч очихдоо зочилно)

# Preorder нэвтрэлтийн жишээ



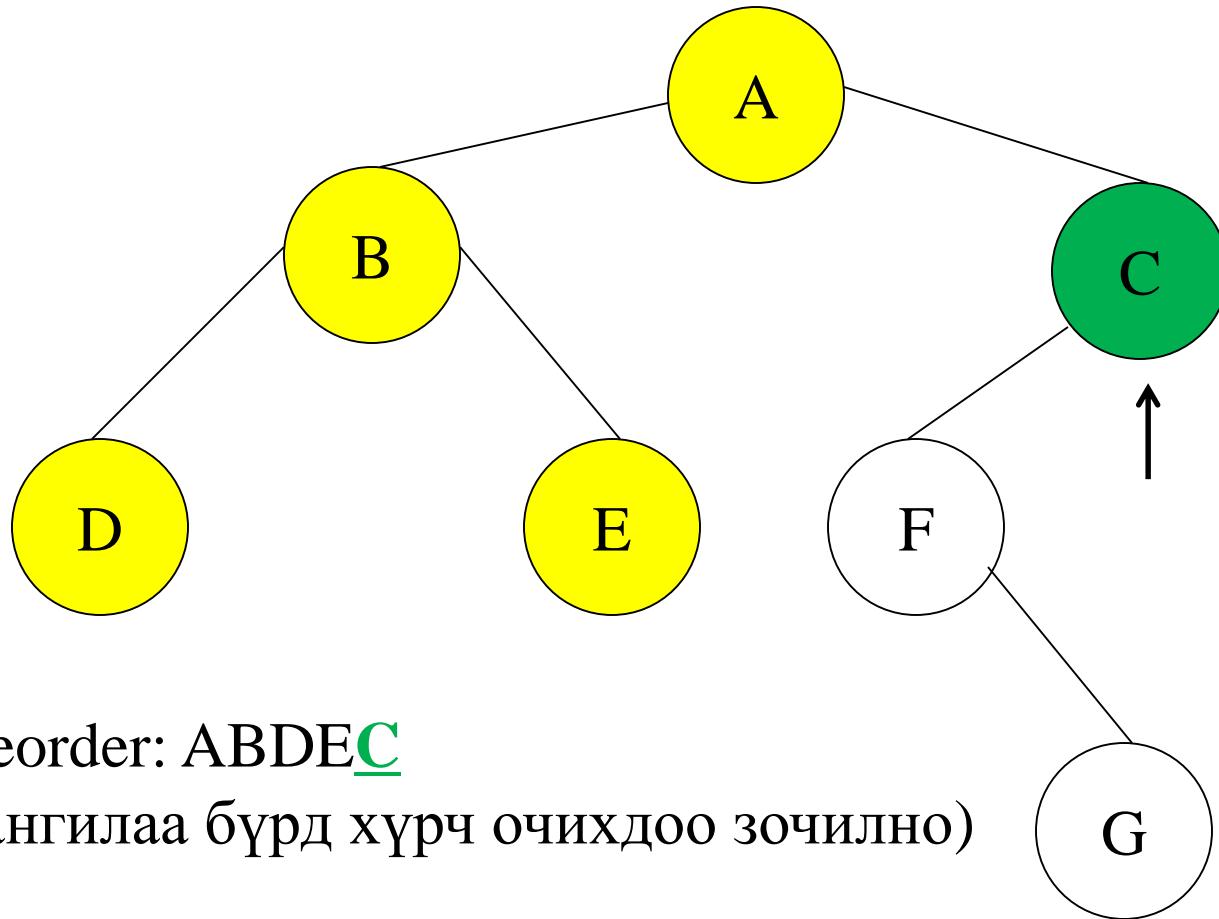
# Preorder нэвтрэлтийн жишээ



Preorder: ABDE

(зангилаа бүрд хүрч очихдоо зочилно)

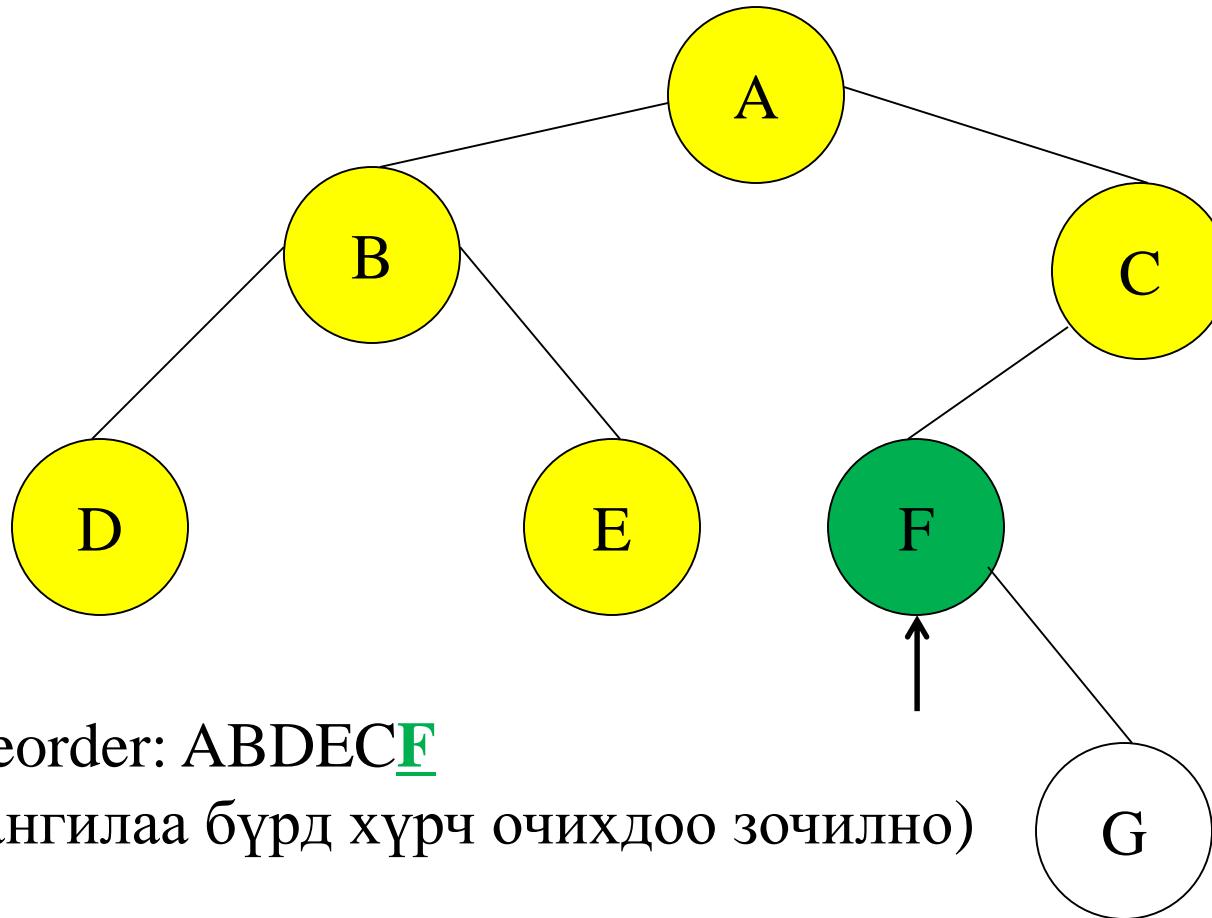
# Preorder нэвтрэлтийн жишээ



Preorder: ABDEC

(зангилаа бүрд хүрч очихдоо зочилно)

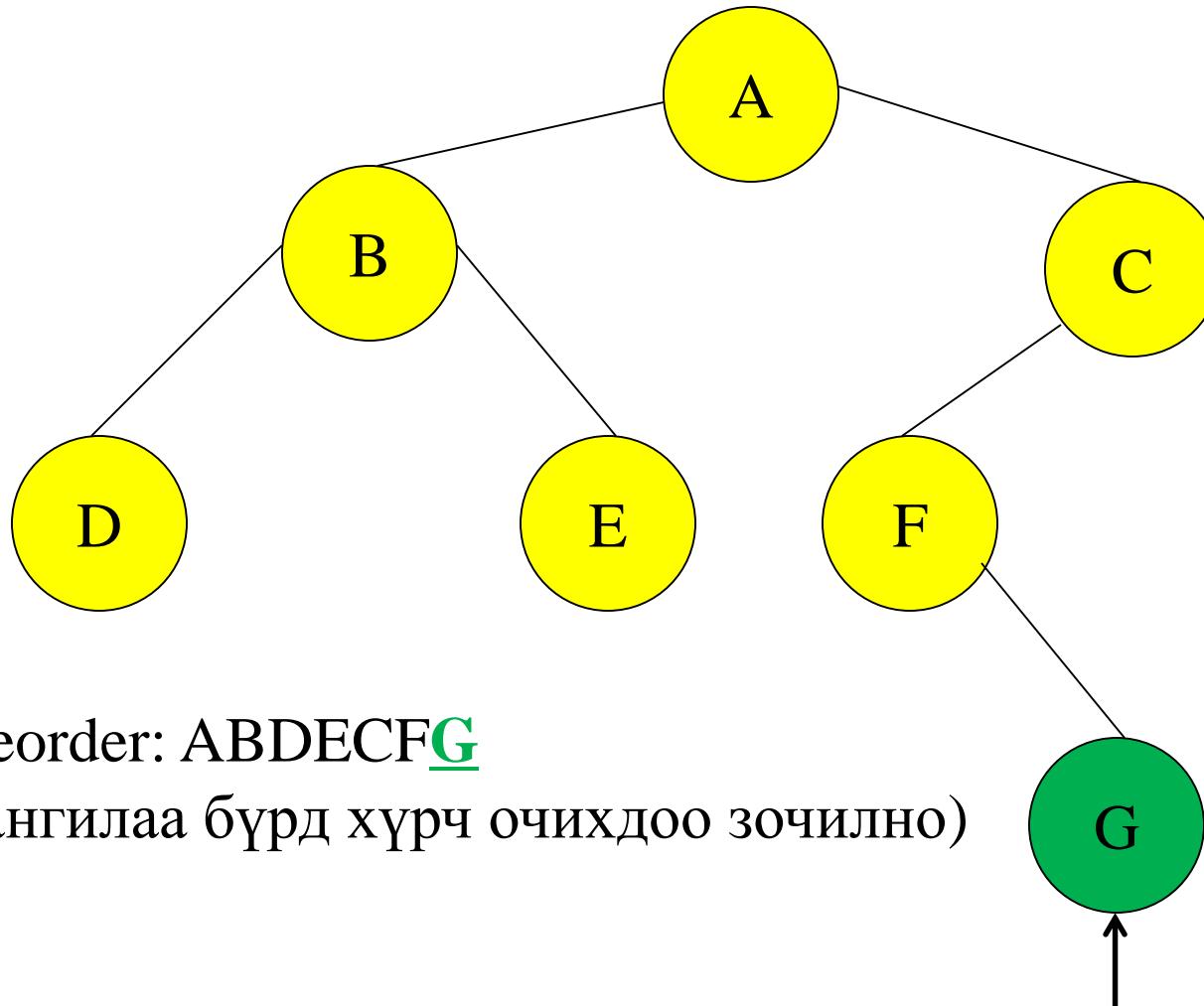
# Preorder нэвтрэлтийн жишээ



Preorder: ABDECF

(зангилаа бүрд хүрч очихдоо зочилно)

# Preorder нэвтрэлтийн жишээ



Preorder: AB~~DE~~CFG

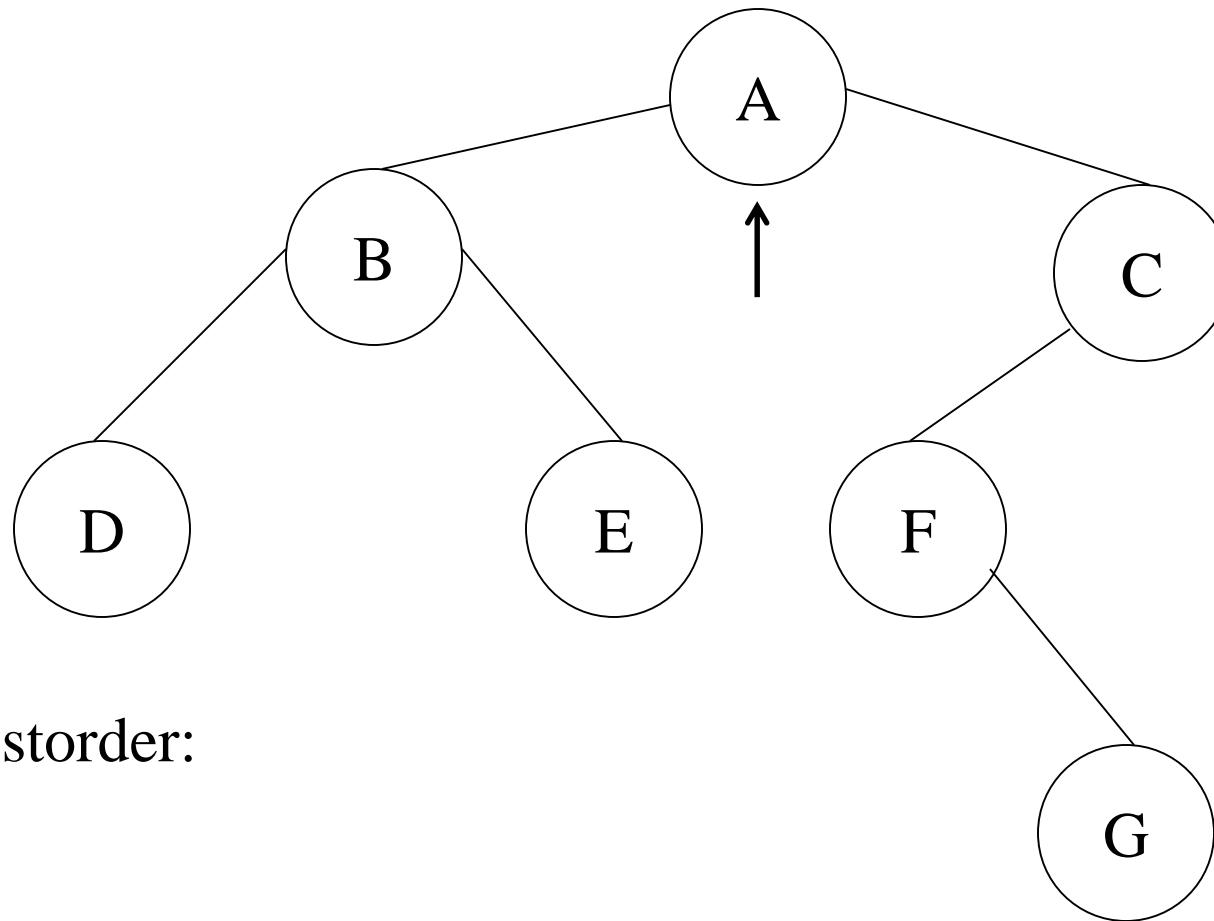
(зангилаа бүрд хүрч очихдоо зочилно)

# PostOrder нэвтрэлт:

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

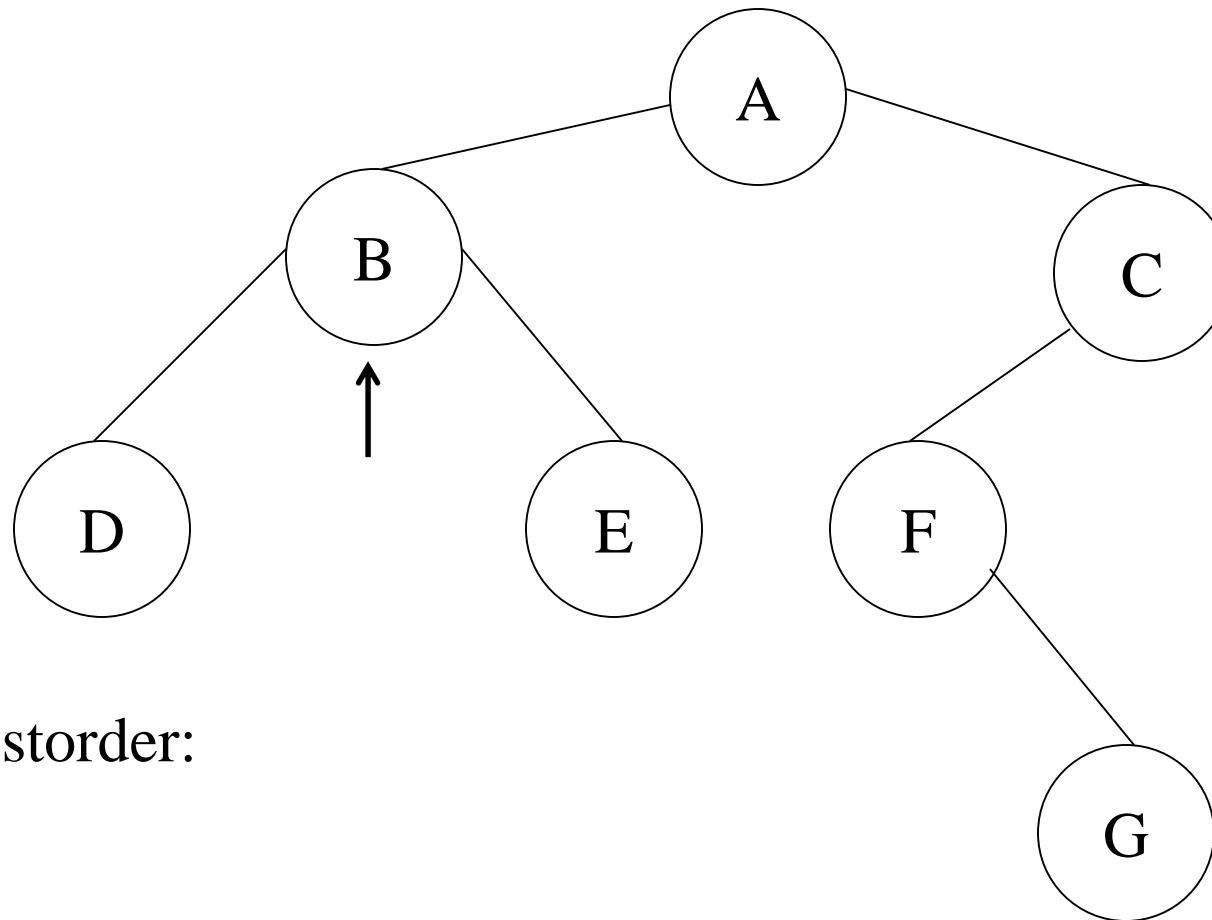
1. Зүүн хүүхэддээ нэвтрэнэ.  
2. Баруун хүүхэддээ нэвтрэнэ.  
3. Үндэсдээ нэвтрэнэ.

# Postorder нэвтрэлтийн жишээ



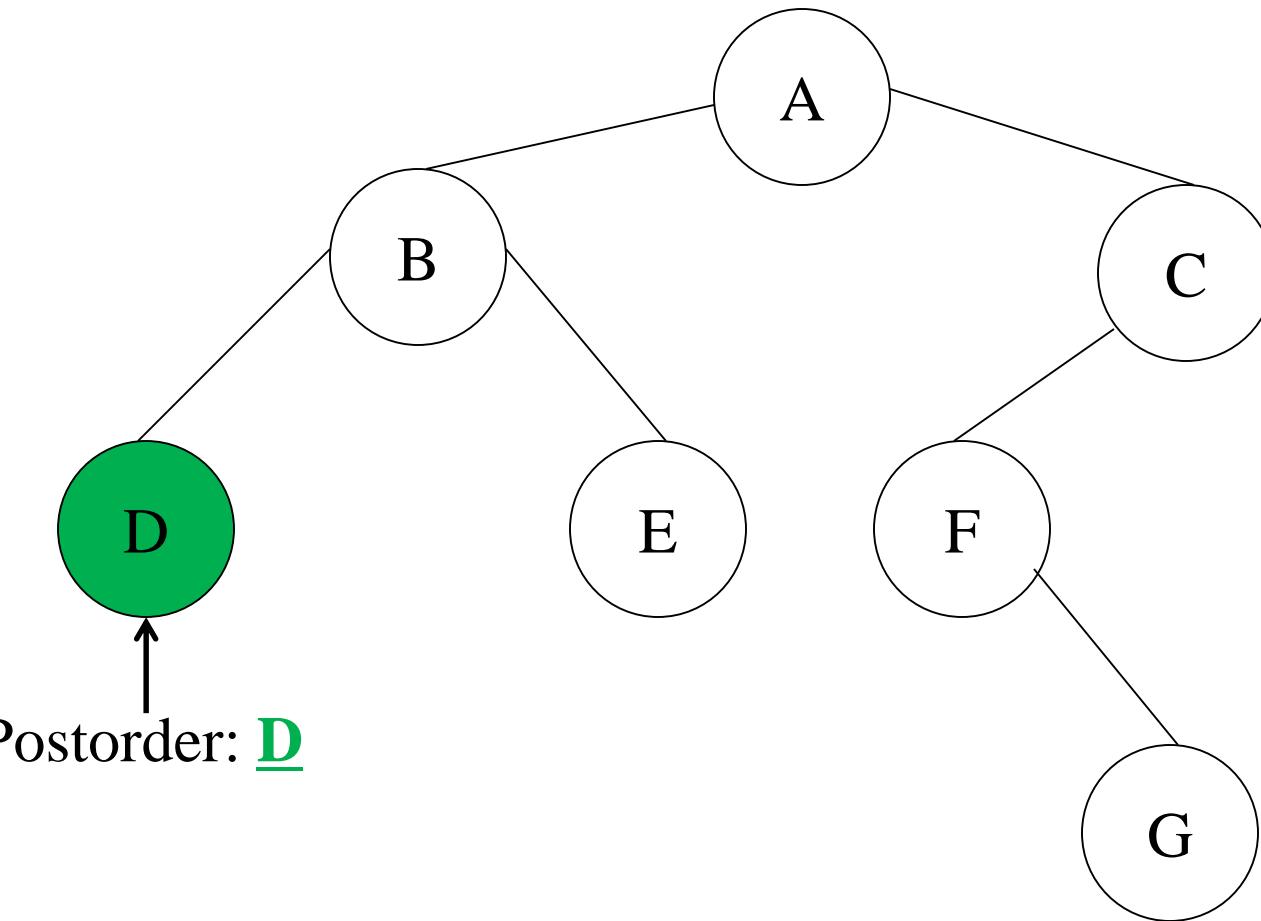
Postorder:

# Postorder нэвтрэлтийн жишээ

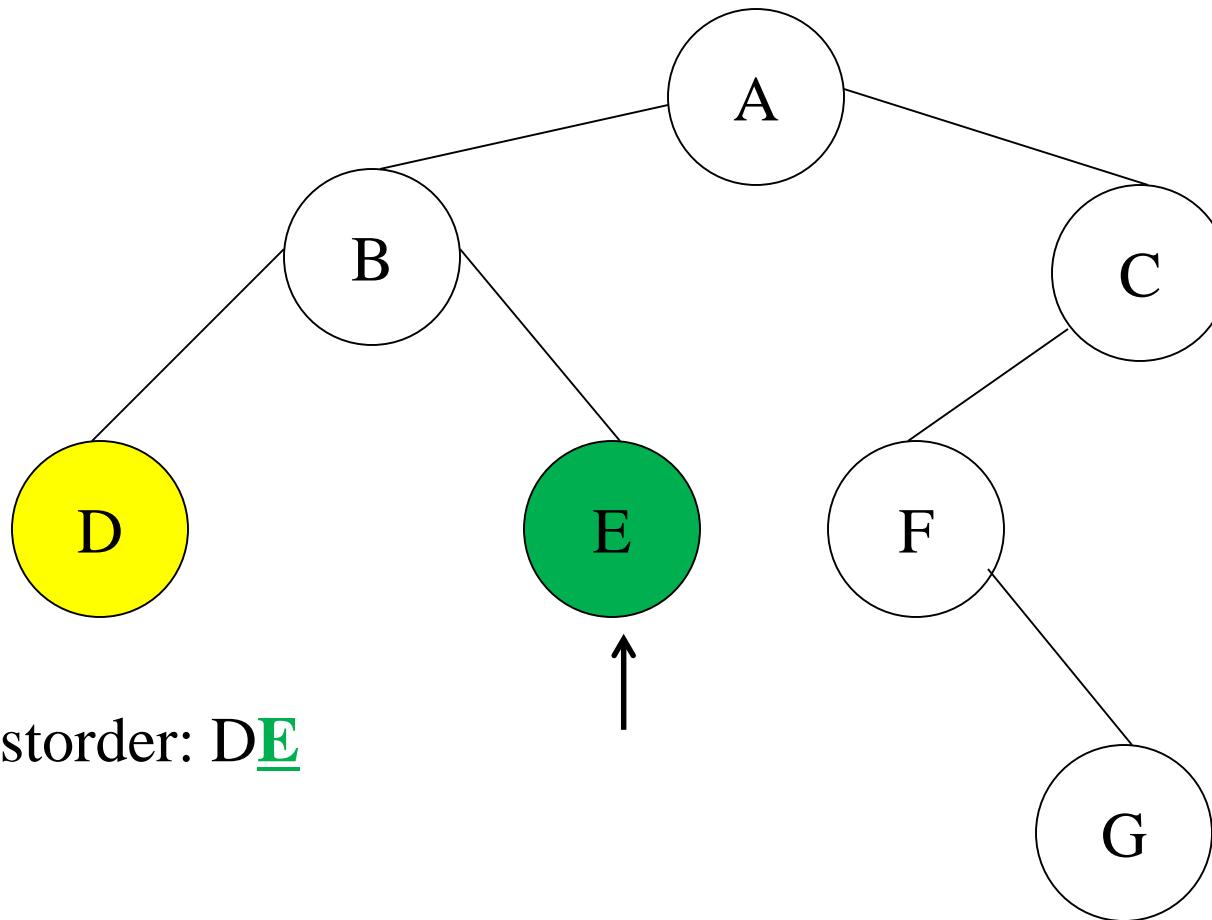


Postorder:

# Postorder нэвтрэлтийн жишээ

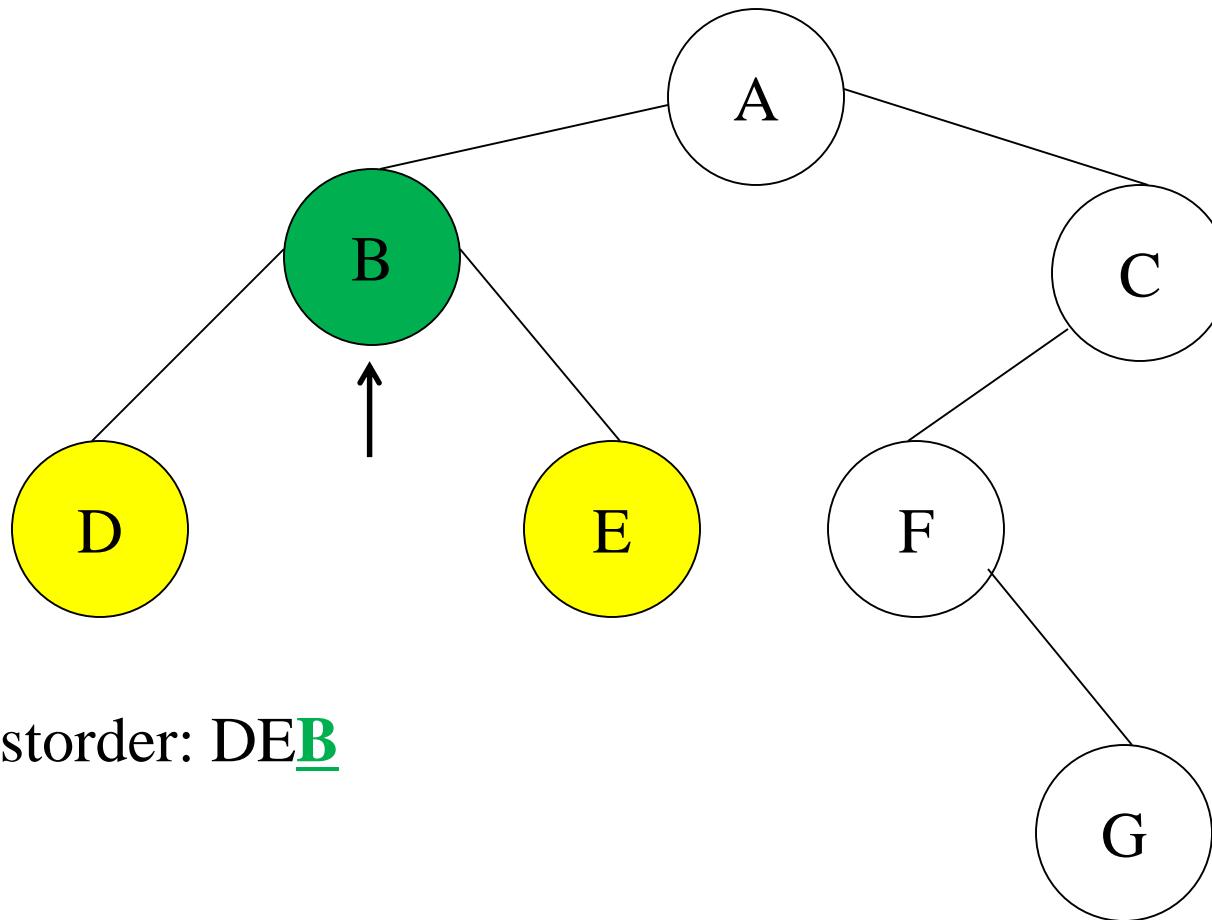


# Postorder нэвтрэлтийн жишээ

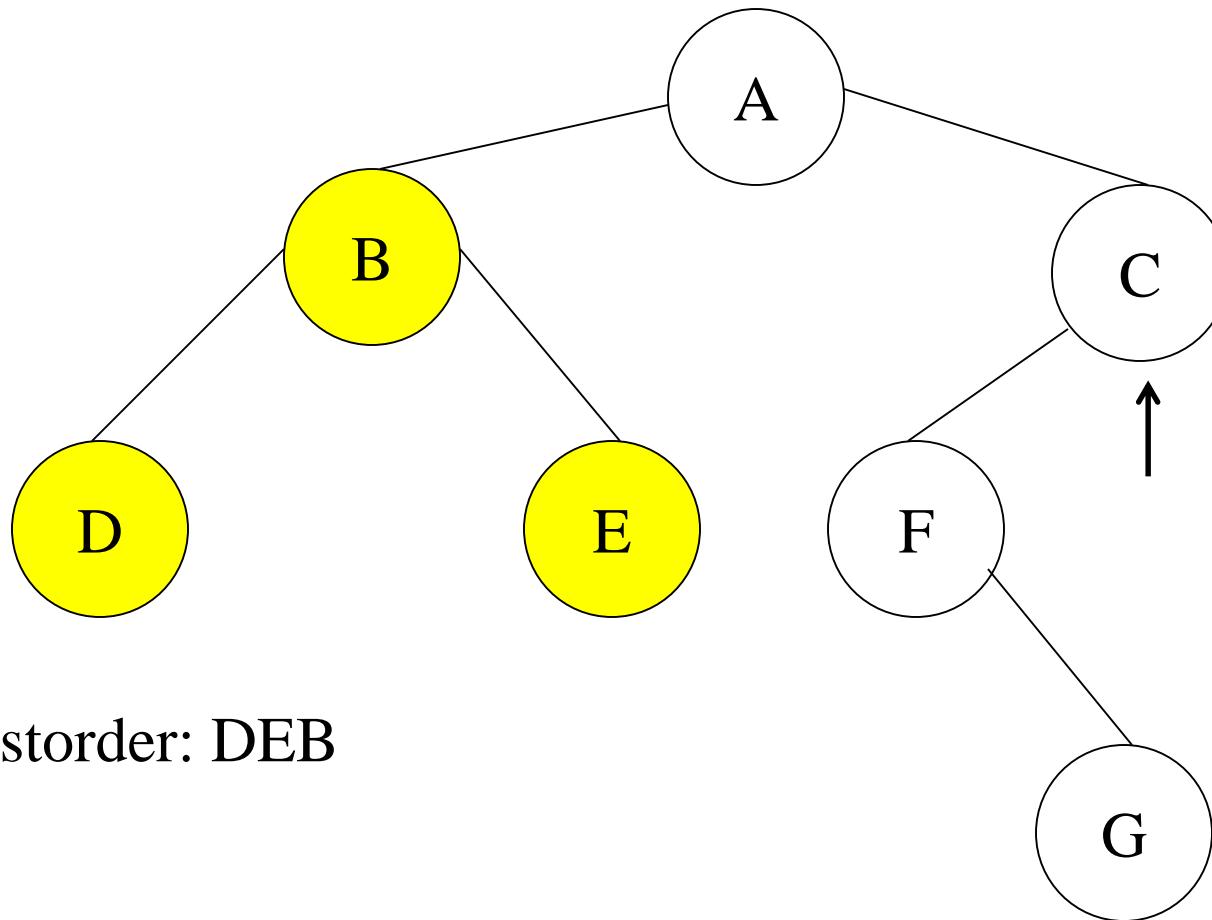


Postorder: DE

# Postorder нэвтрэлтийн жишээ

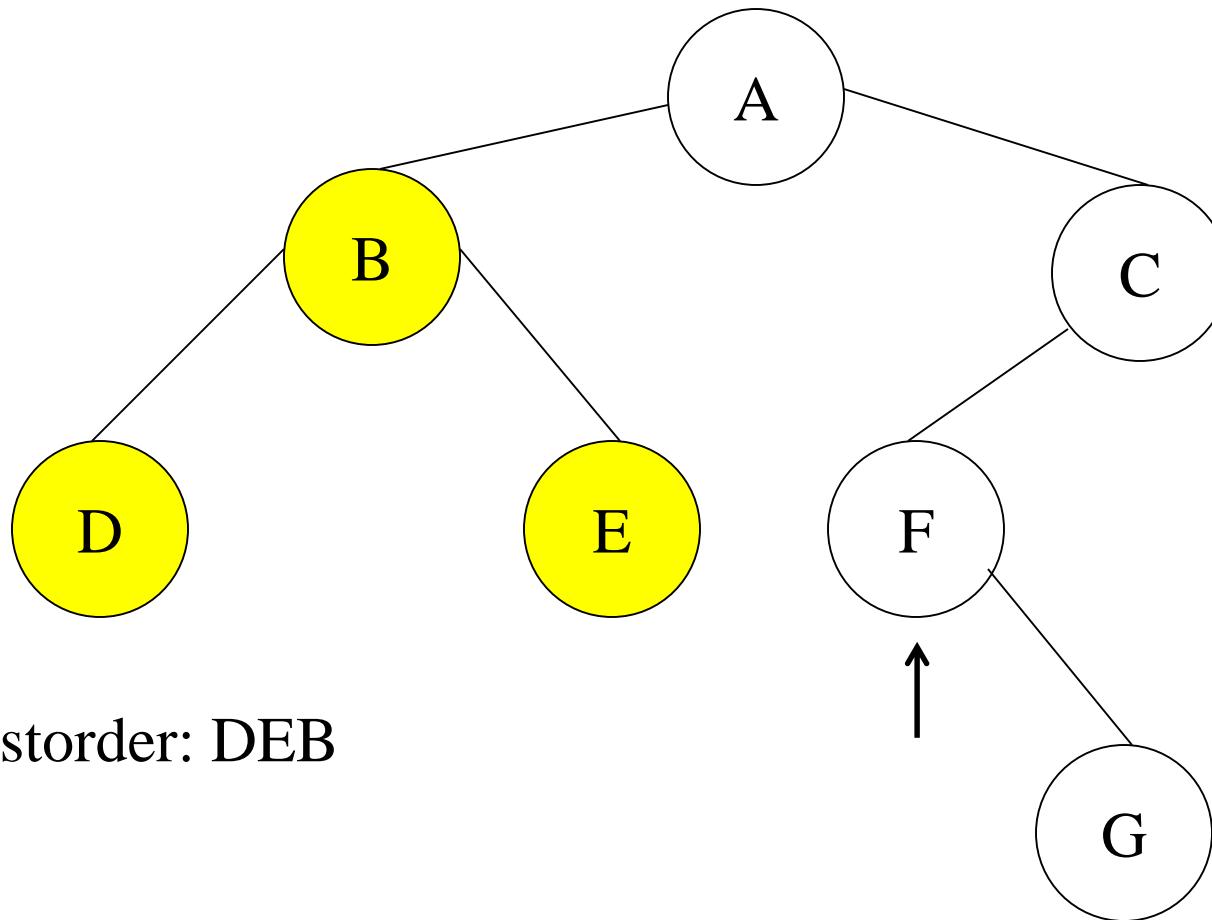


# Postorder нэвтрэлтийн жишээ



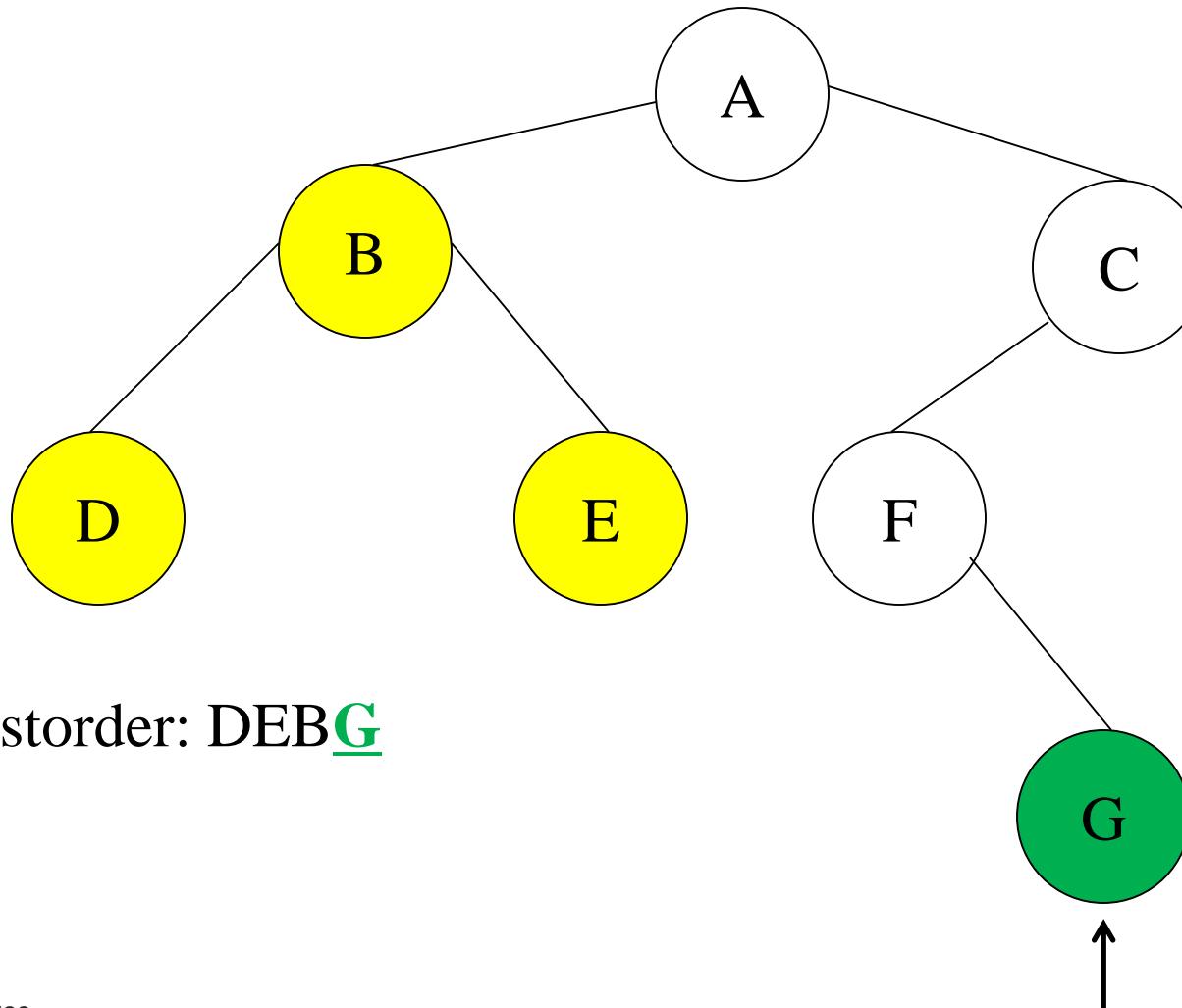
Postorder: DEB

# Postorder нэвтрэлтийн жишээ

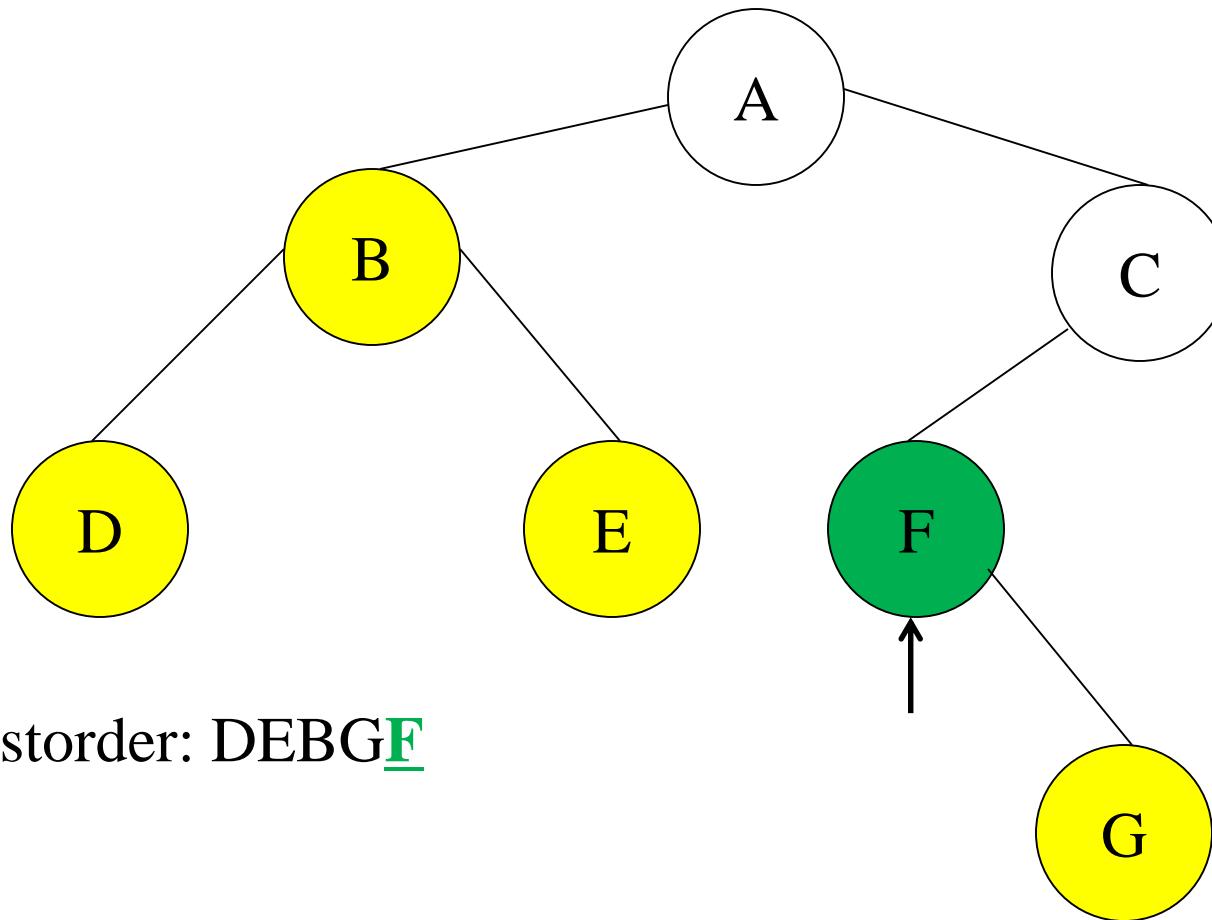


Postorder: DEB

# Postorder нэвтрэлтийн жишээ

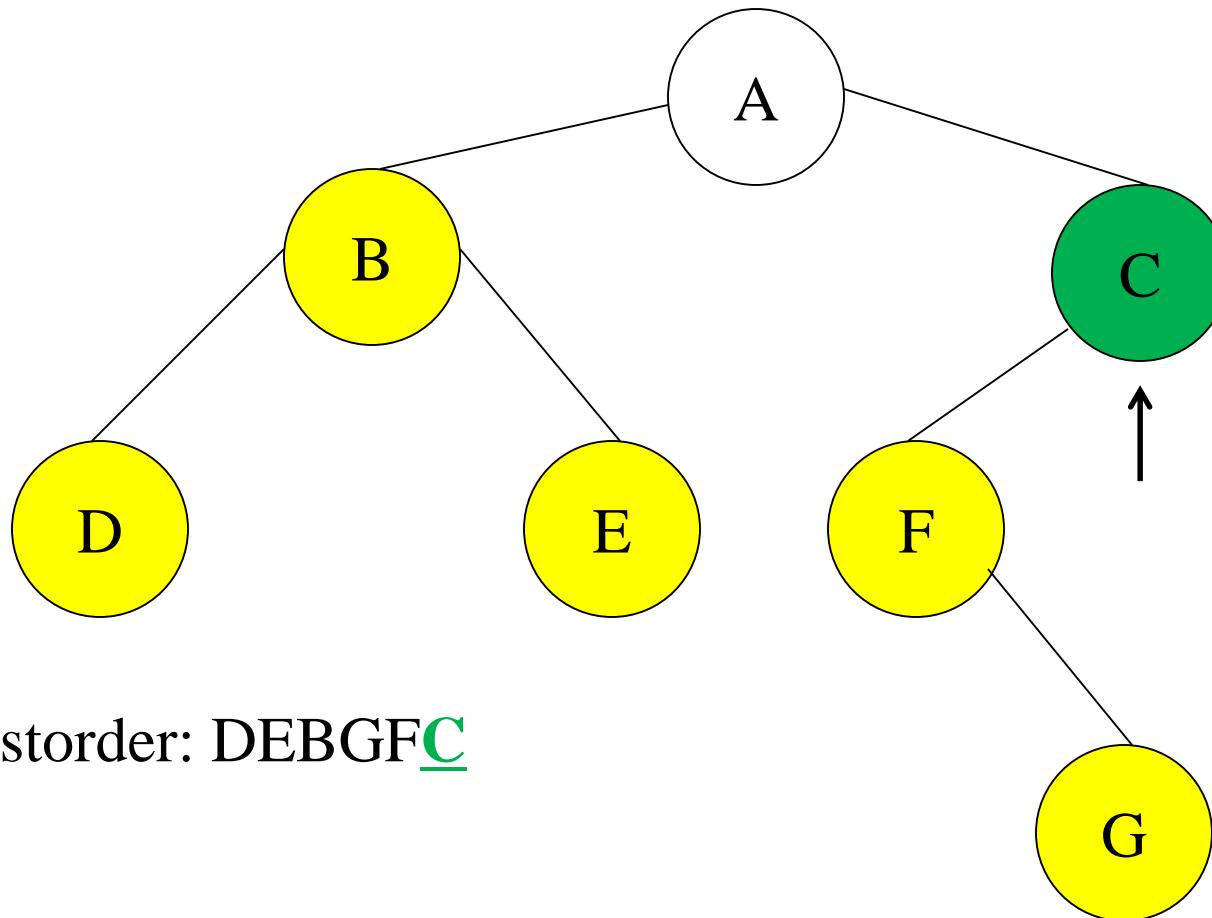


# Postorder нэвтрэлтийн жишээ



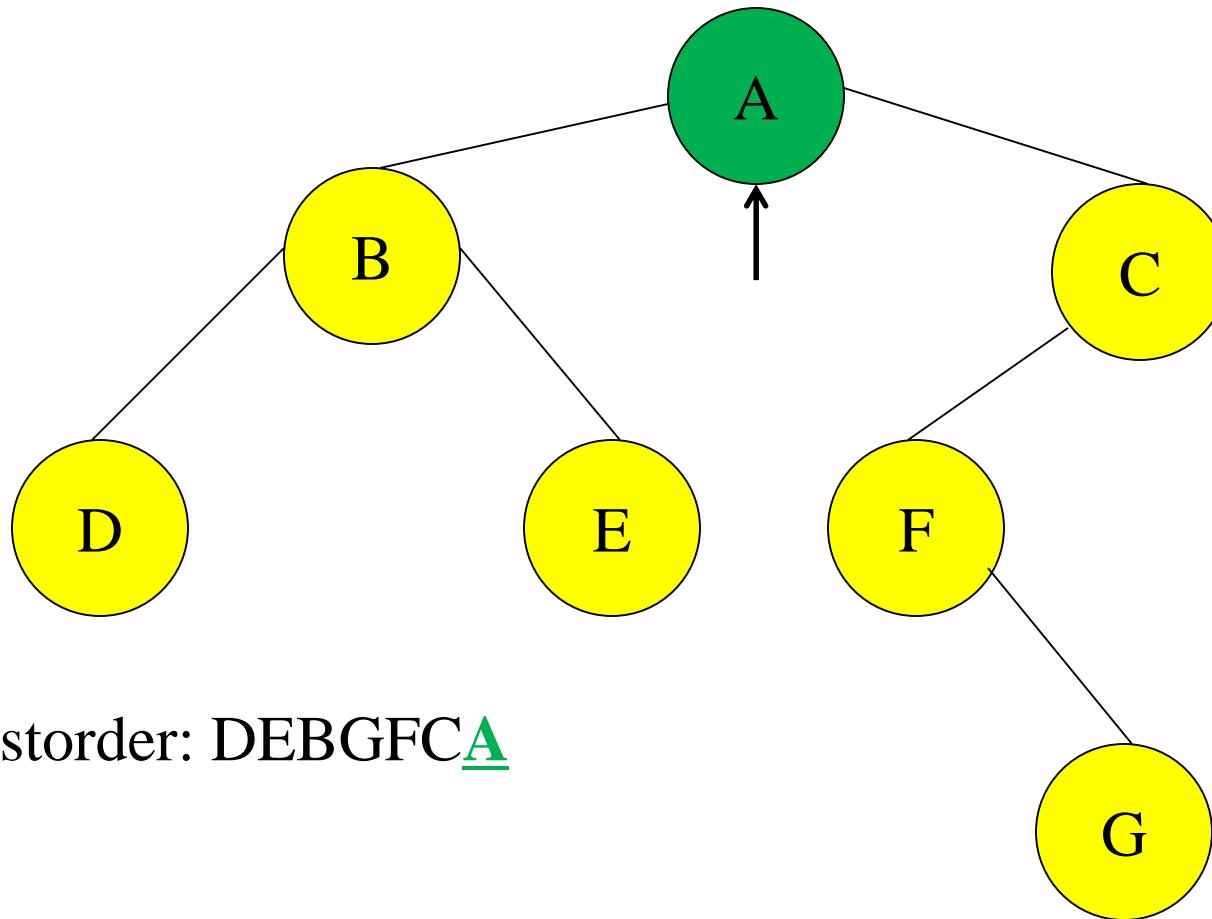
Postorder: DEBGF

# Postorder нэвтрэлтийн жишээ



Postorder: DEBGFC

# Postorder нэвтрэлтийн жишээ



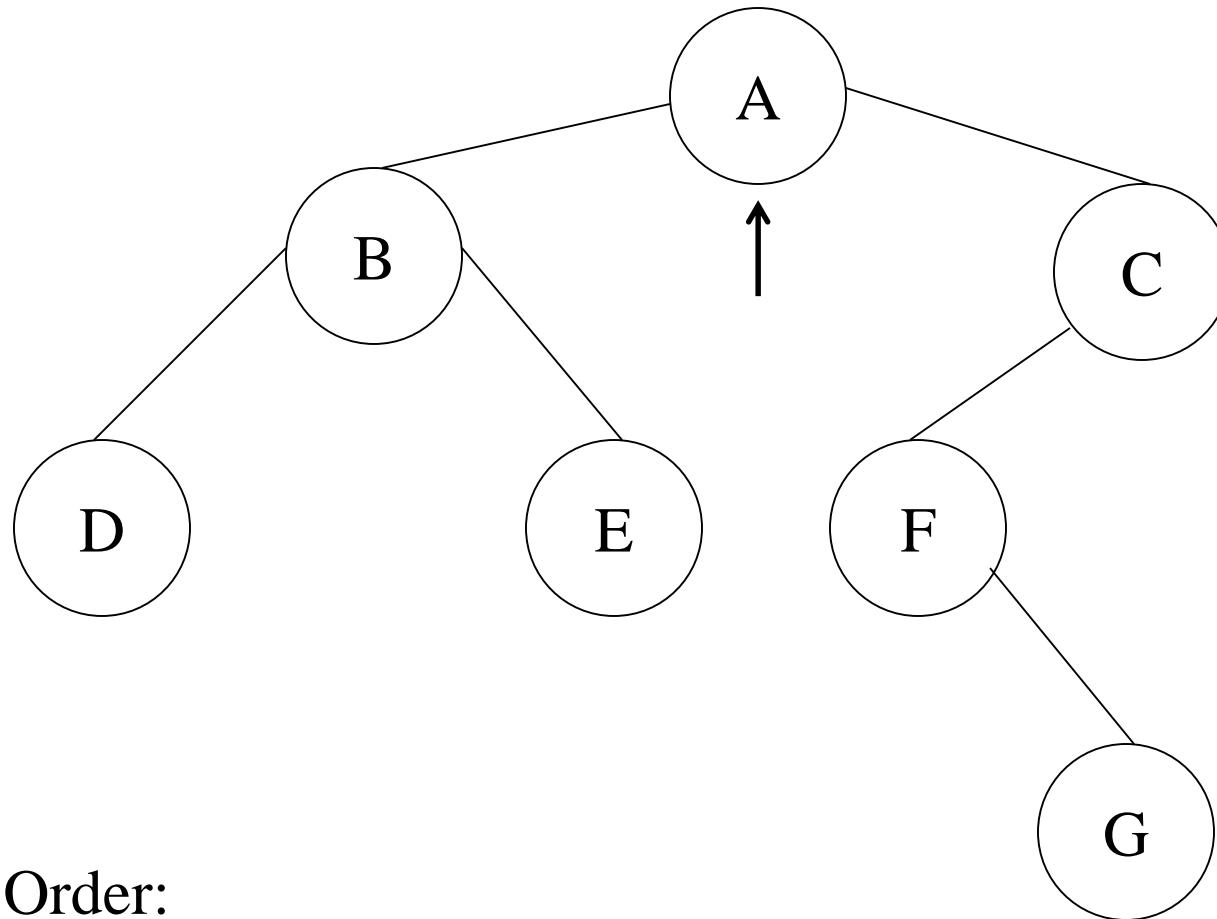
Postorder: DEBGFCA

# in Order нэвтрэлт:

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

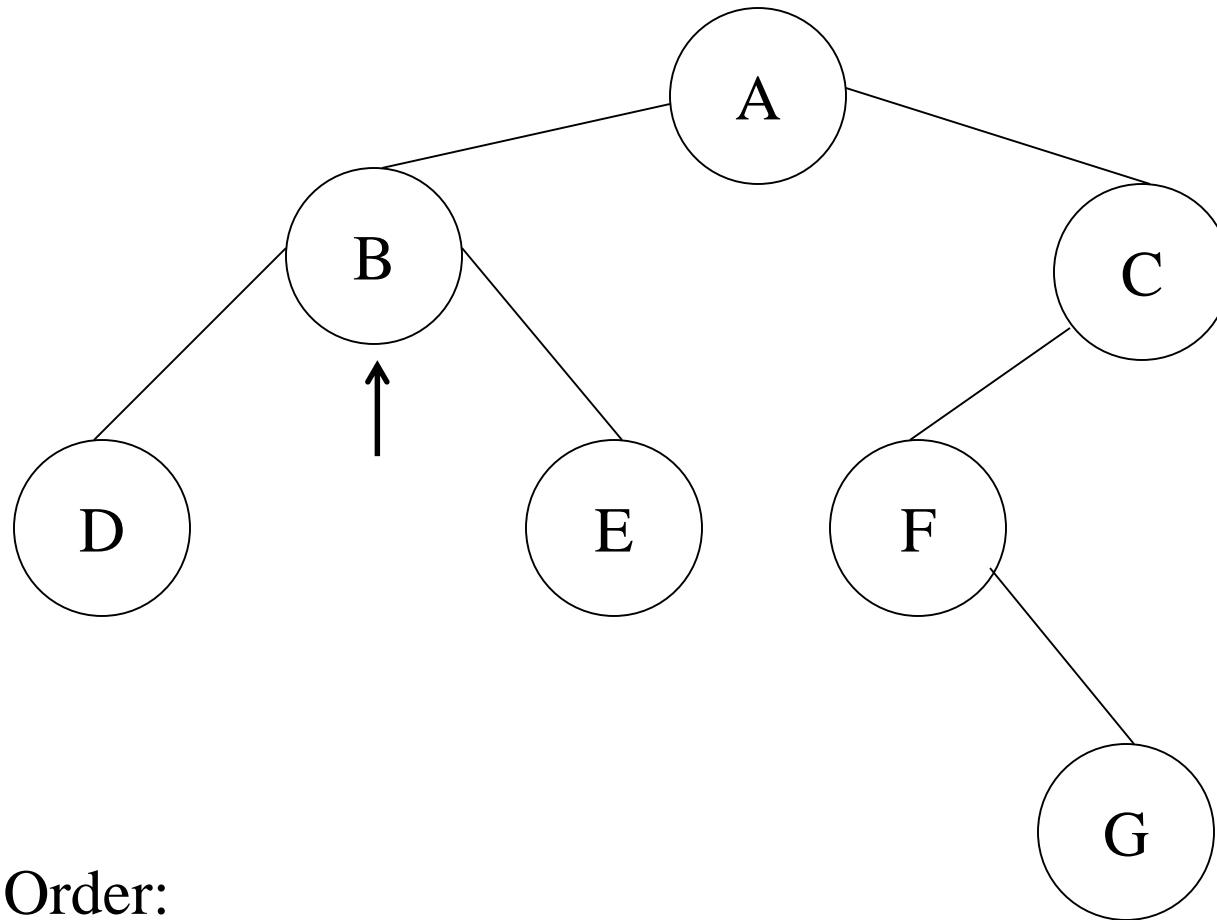
- 1. Зүүн хүүхэддээ нэвтрэнэ.
- 2. Үндэсдээ нэвтрэнэ.
- 3. Баруун хүүхэддээ нэвтрэнэ.

# In Order нэвтрэлтийн жишээ



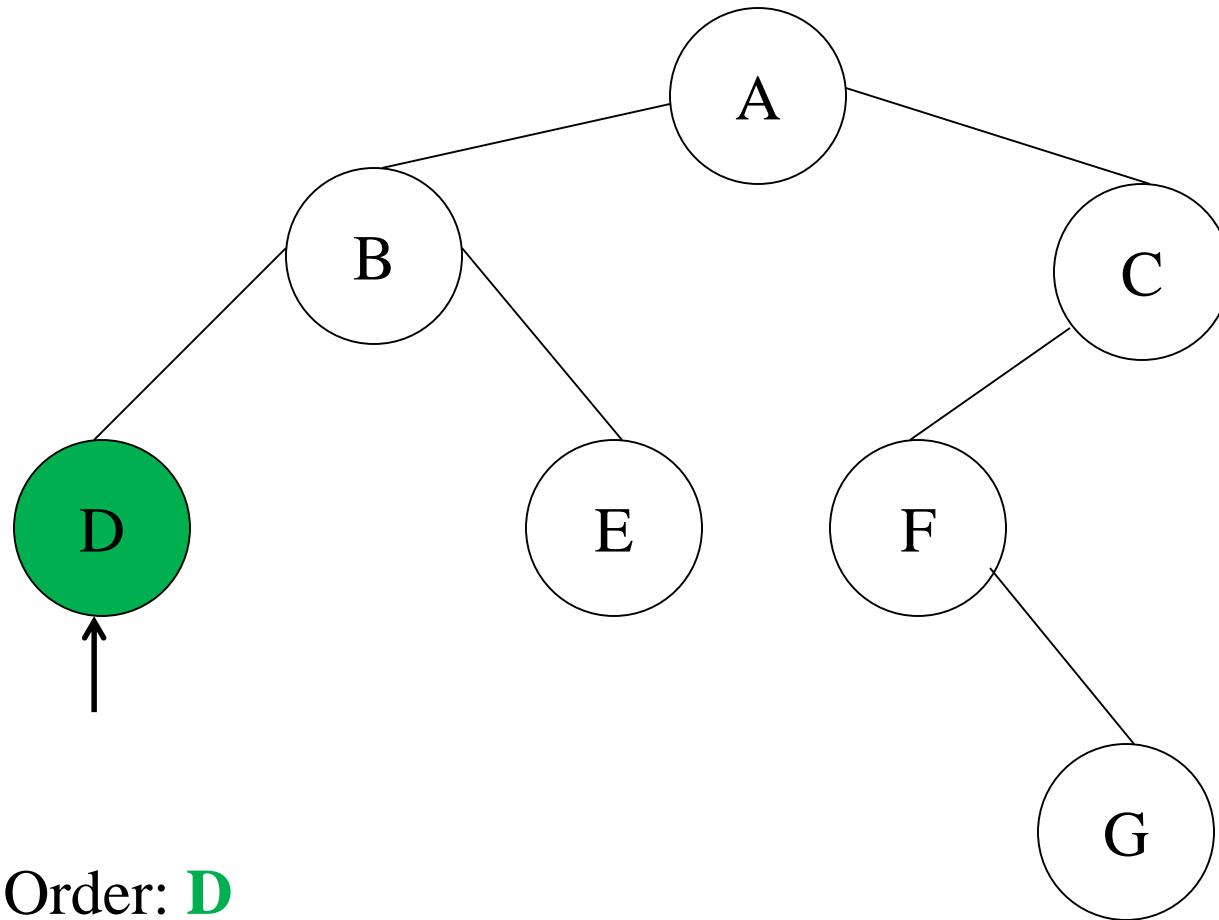
In Order:

# In Order нэвтрэлтийн жишээ



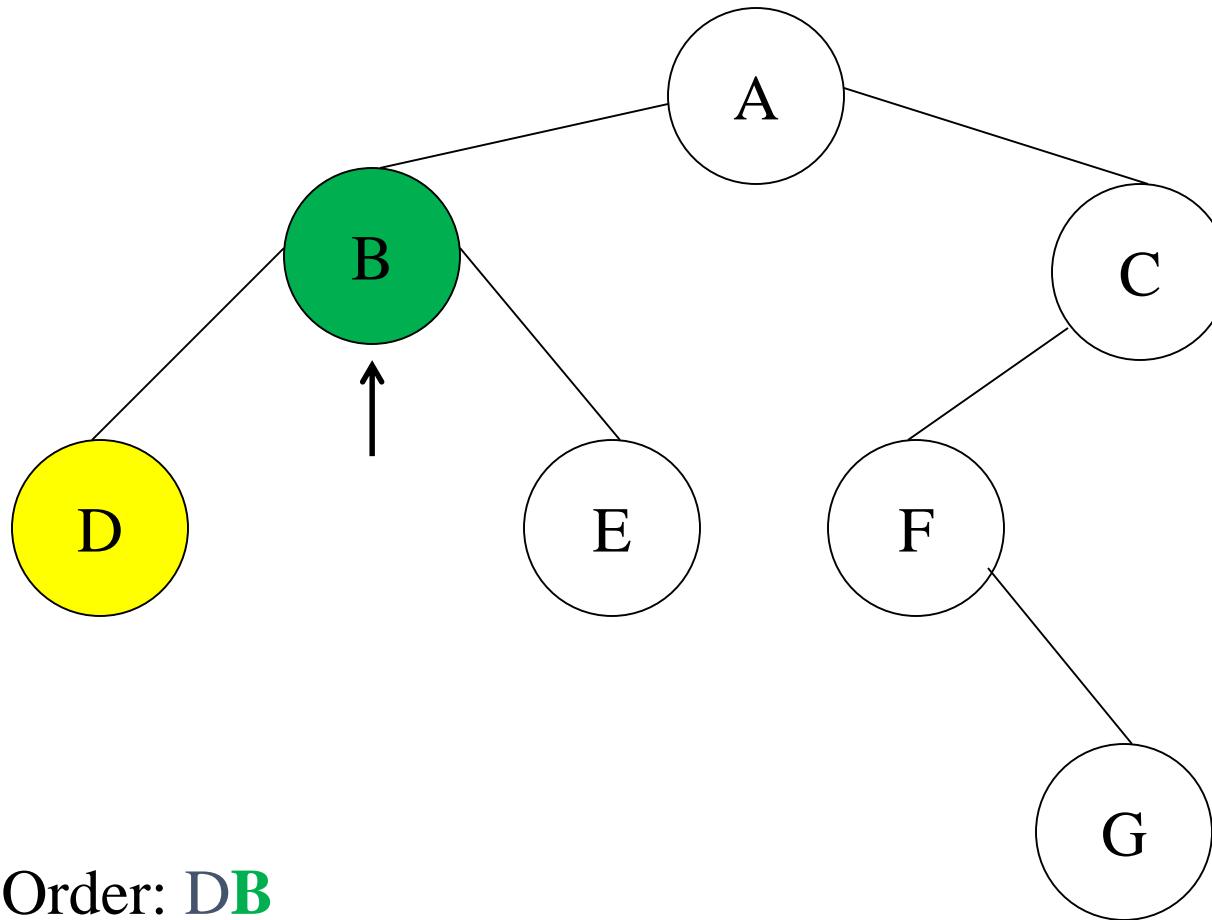
In Order:

# In Order нэвтрэлтийн жишээ



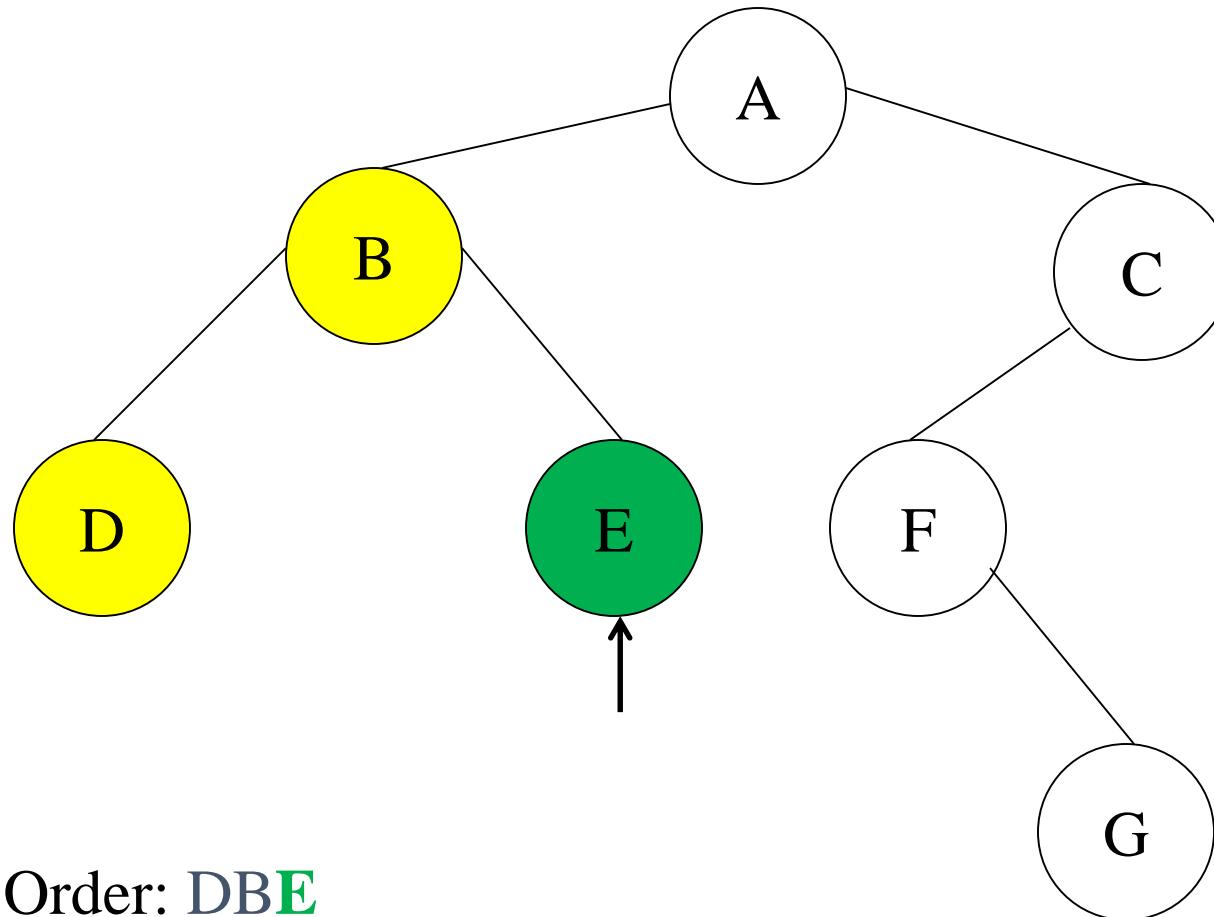
In Order: D

# In Order нэвтрэлтийн жишээ

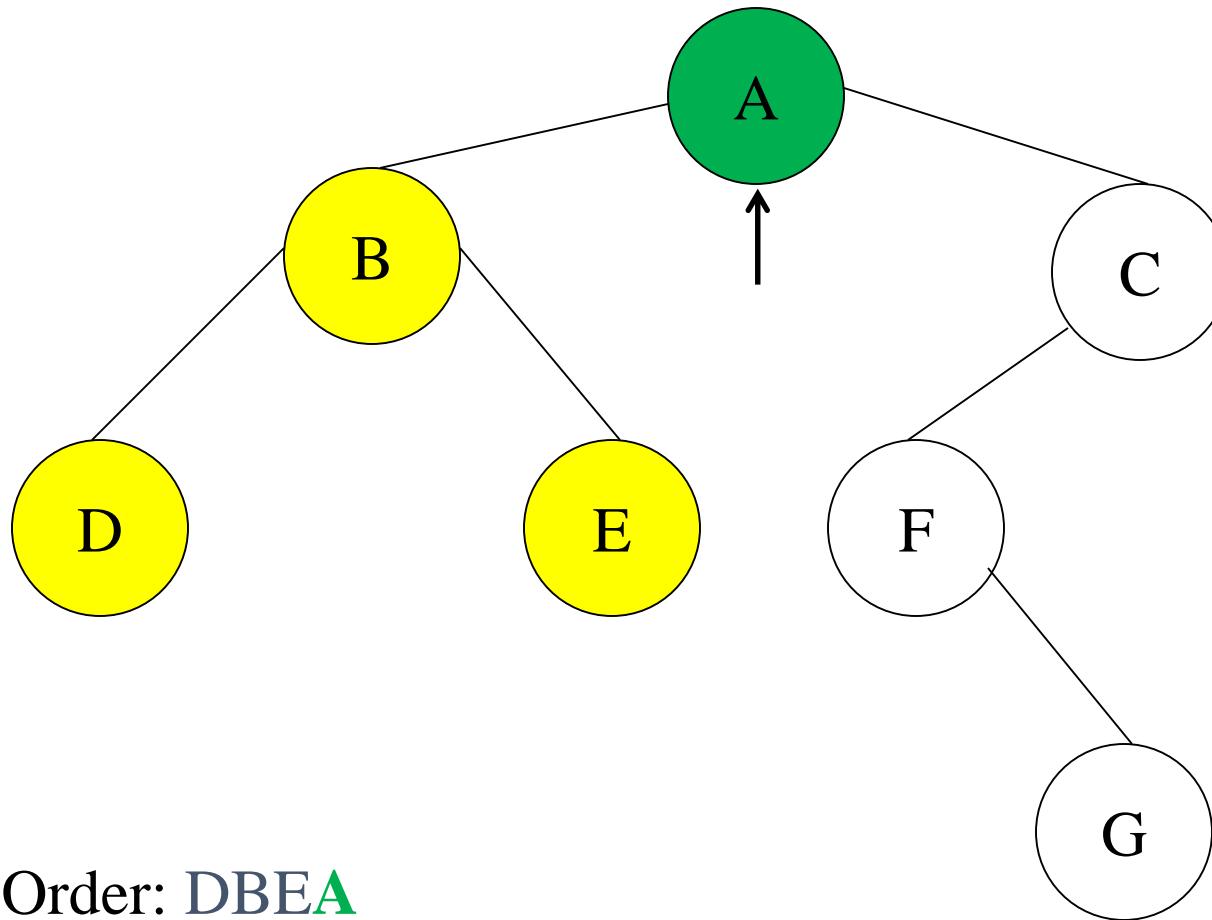


In Order: D**B**

# In Order нэвтрэлтийн жишээ

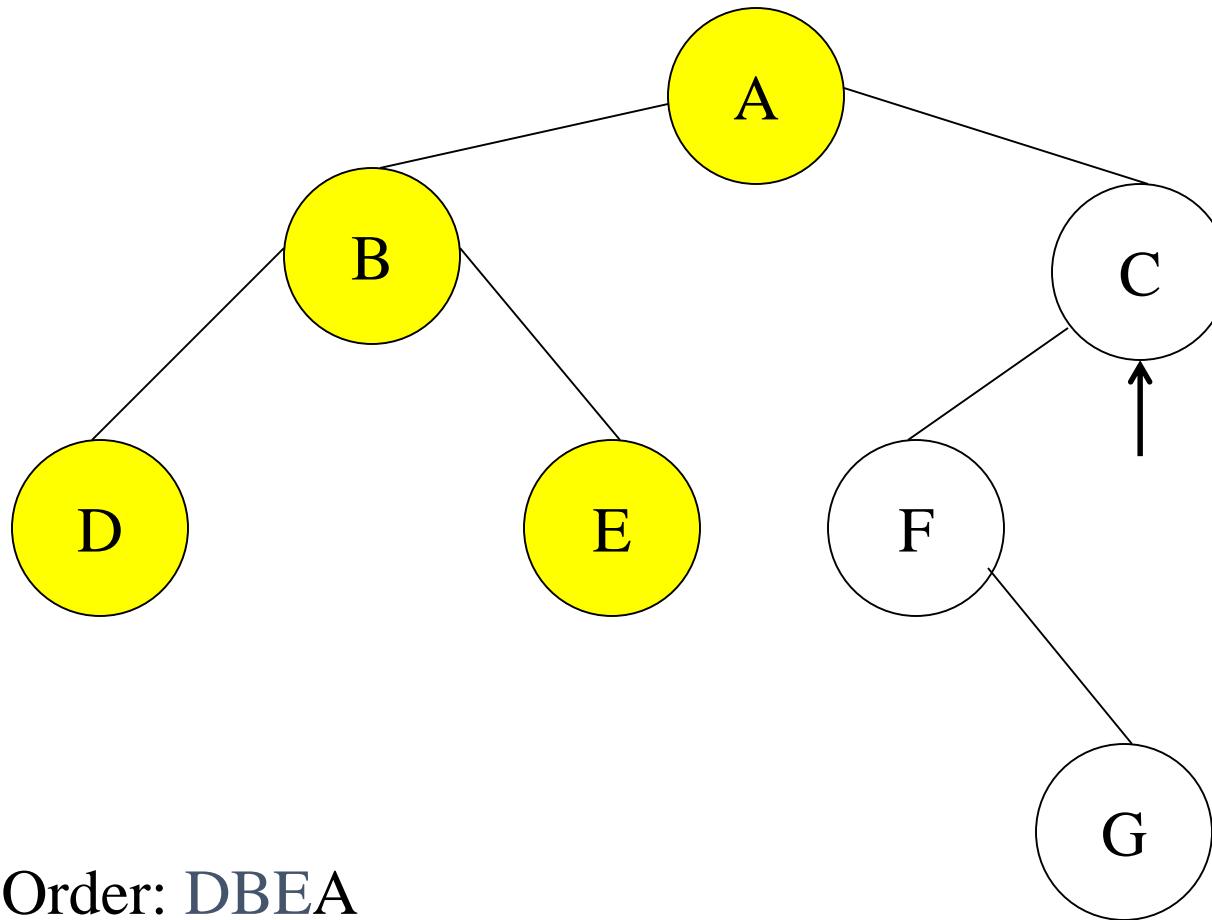


# In Order нэвтрэлтийн жишээ



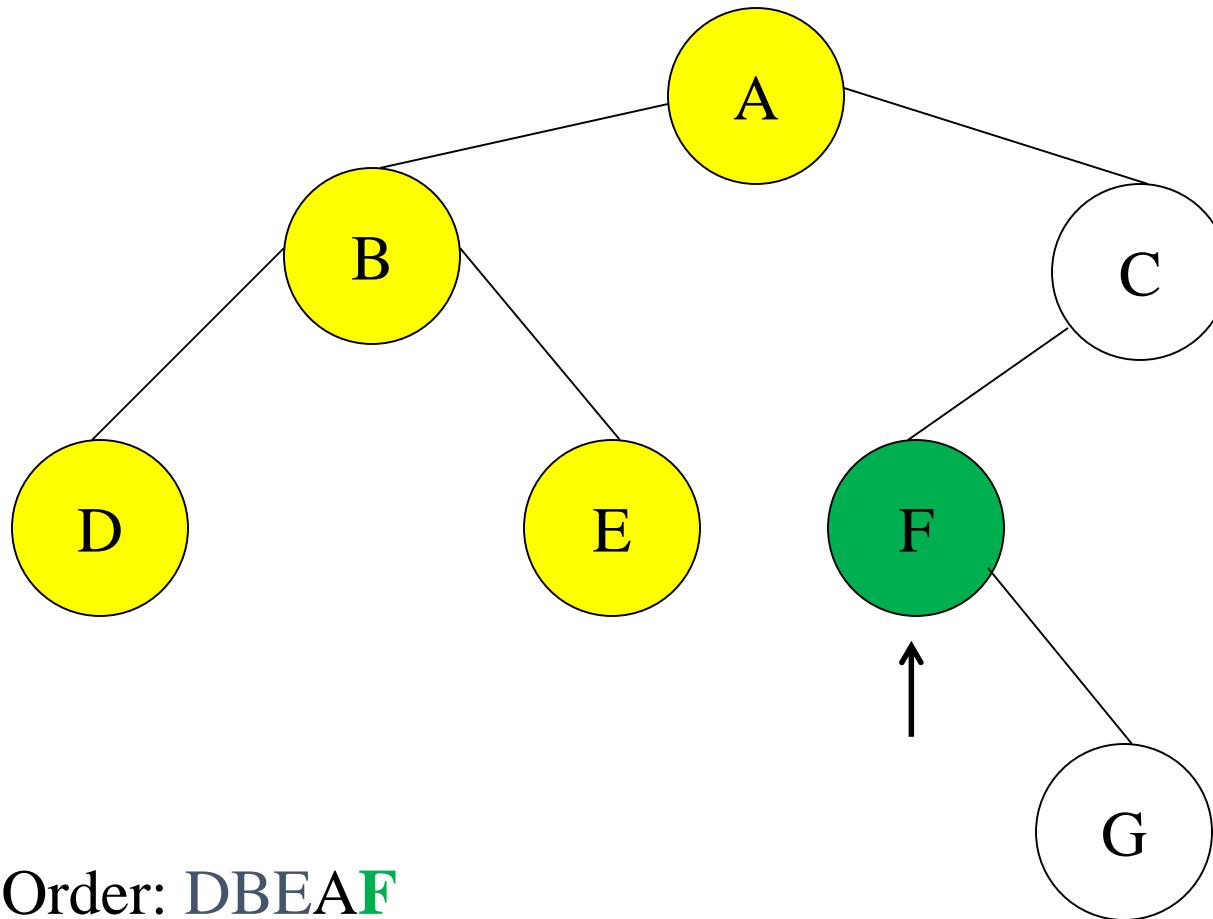
In Order: DBEAA

# In Order нэвтрэлтийн жишээ



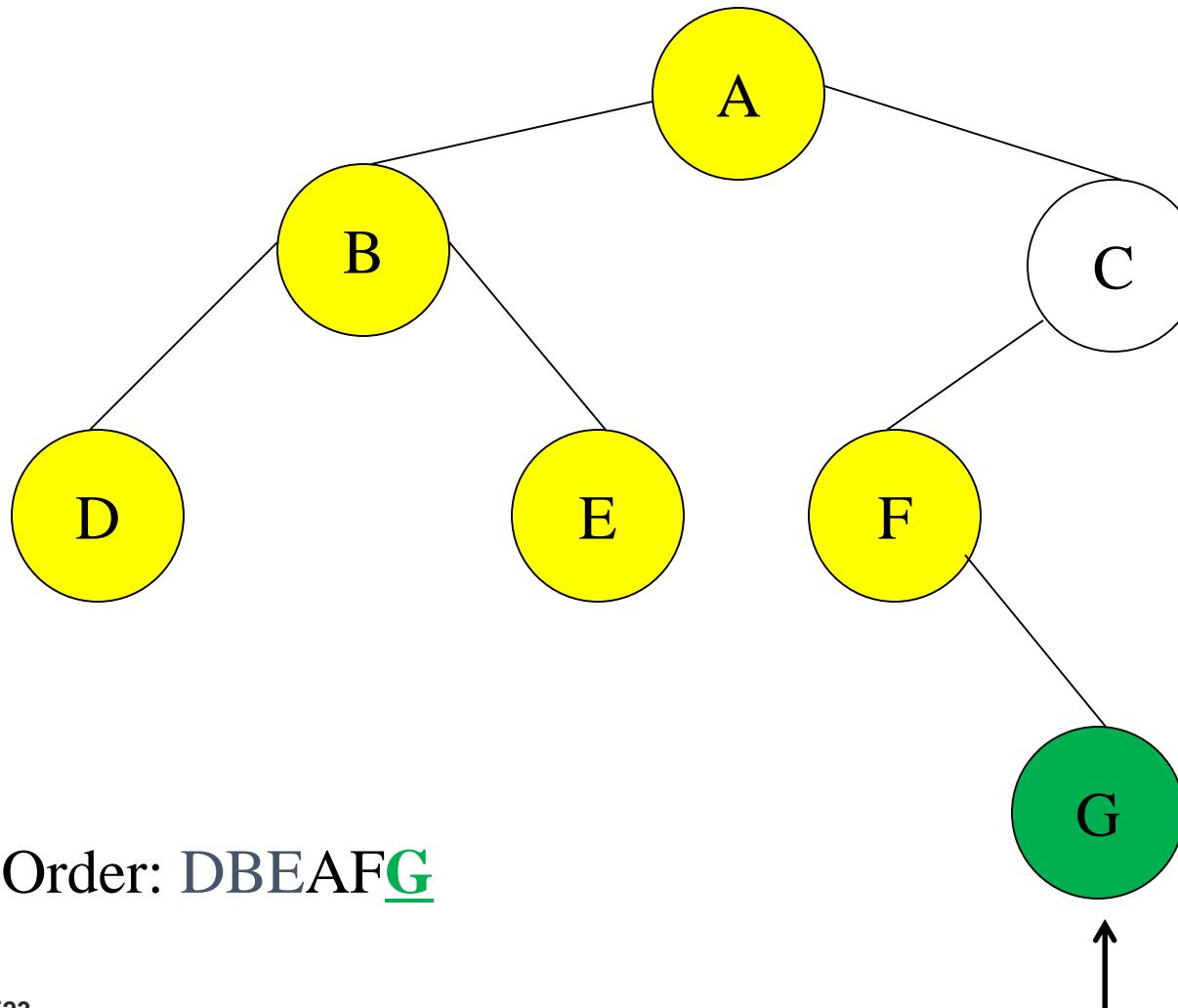
In Order: DBEA

# In Order нэвтрэлтийн жишээ

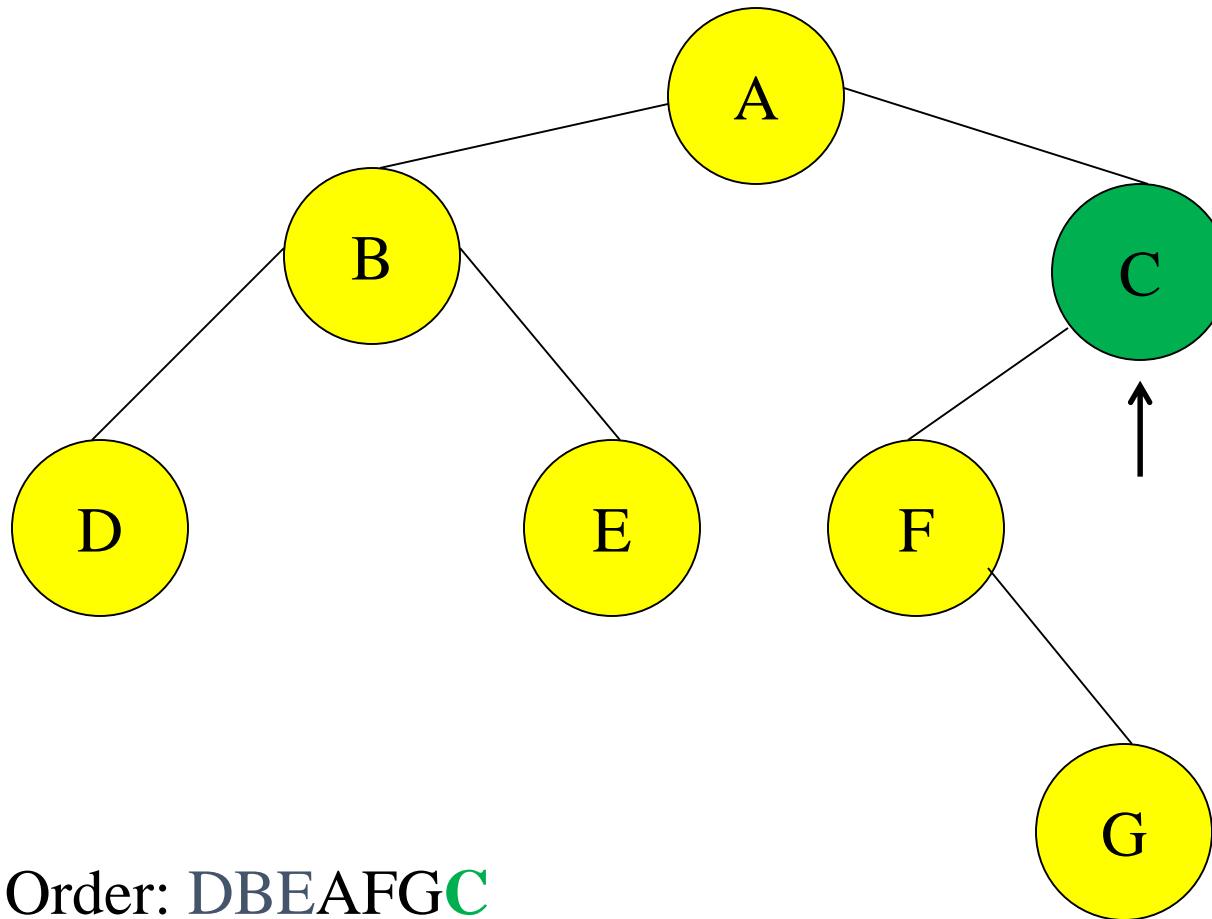


In Order: DBEAF

# In Order нэвтрэлтийн жишээ

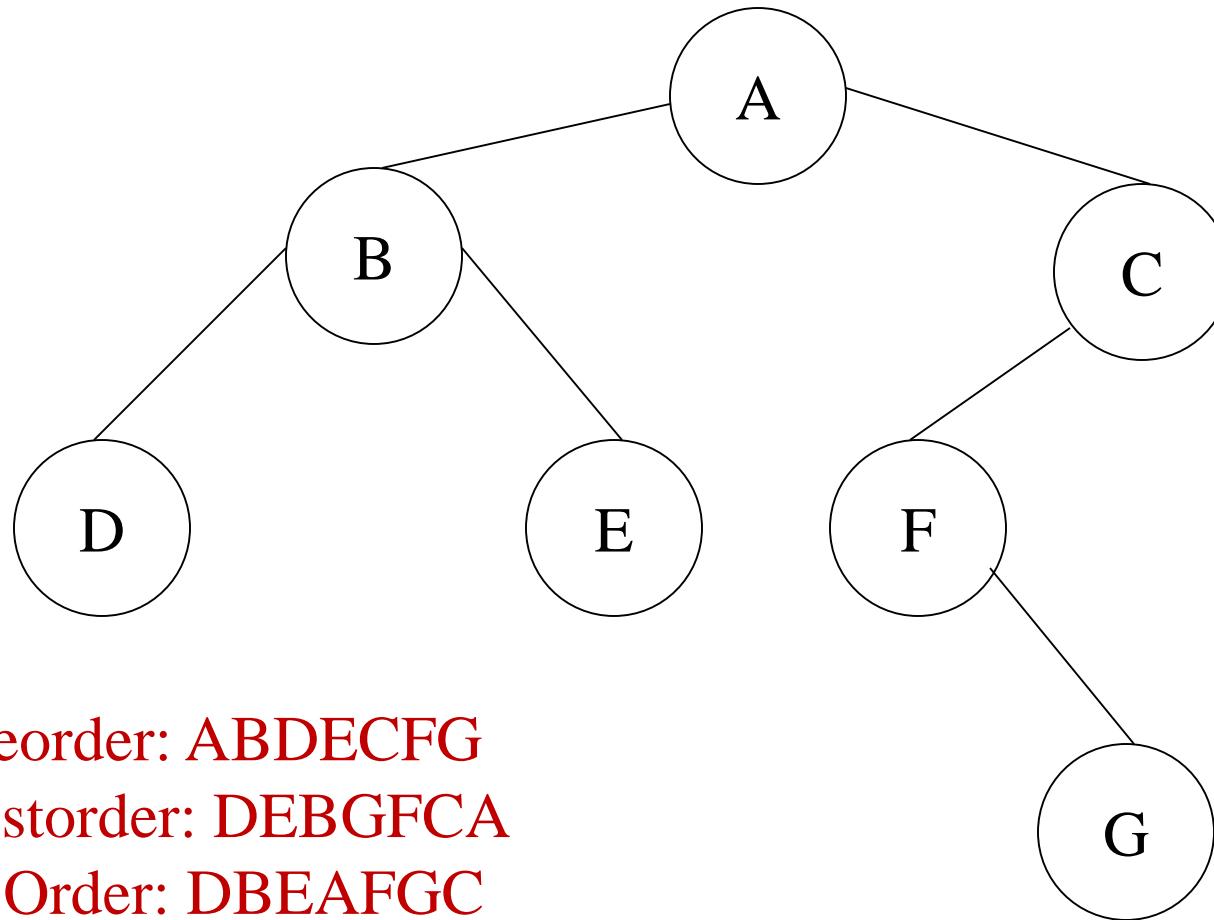


# In Order нэвтрэлтийн жишээ



In Order: DBEAFGC

# Модны нэвтрэлт



Preorder: ABDEC<sub>F</sub>G

Postorder: DEBG<sub>F</sub>C<sub>A</sub>

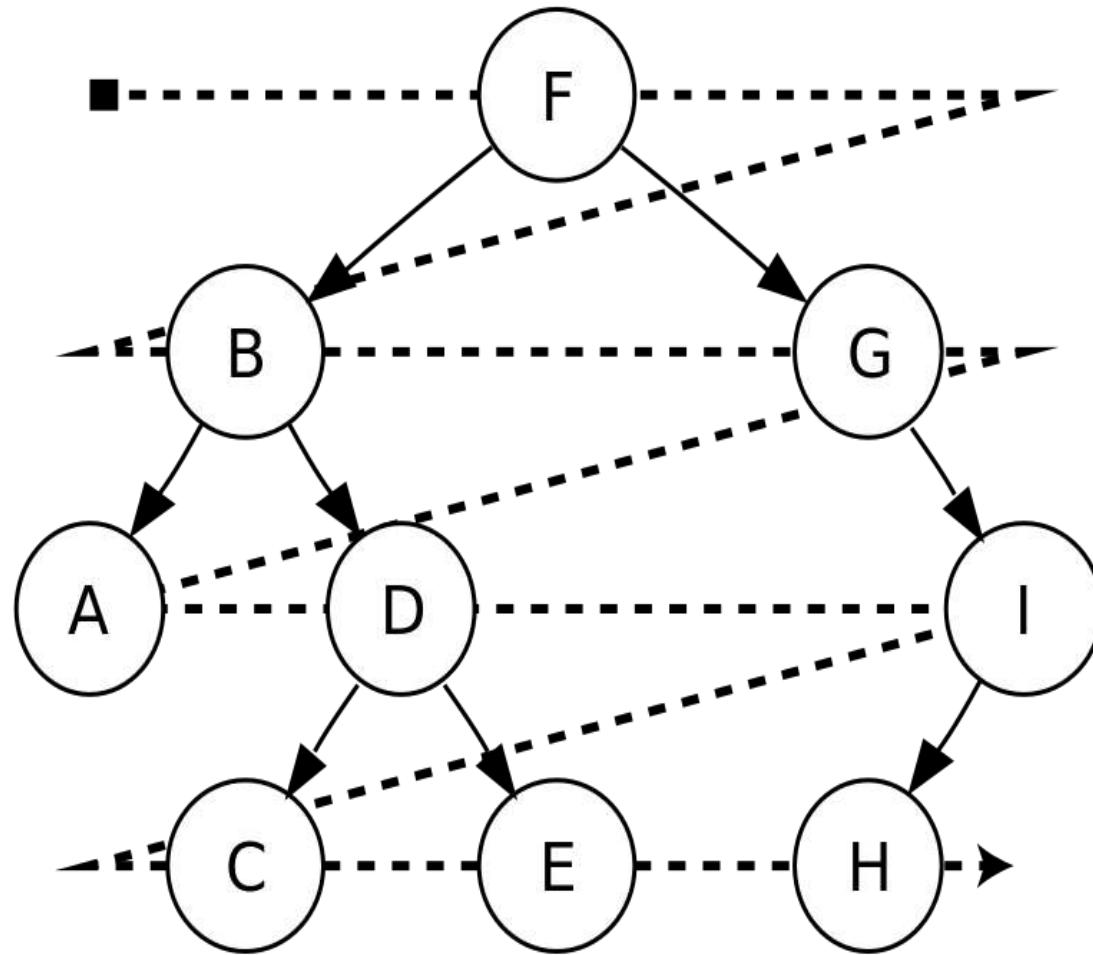
In Order: DBEA<sub>F</sub>GC

# LevelOrder нэвтрэлт

- Түвшин түвшингээр нь нэвтэрнэ.

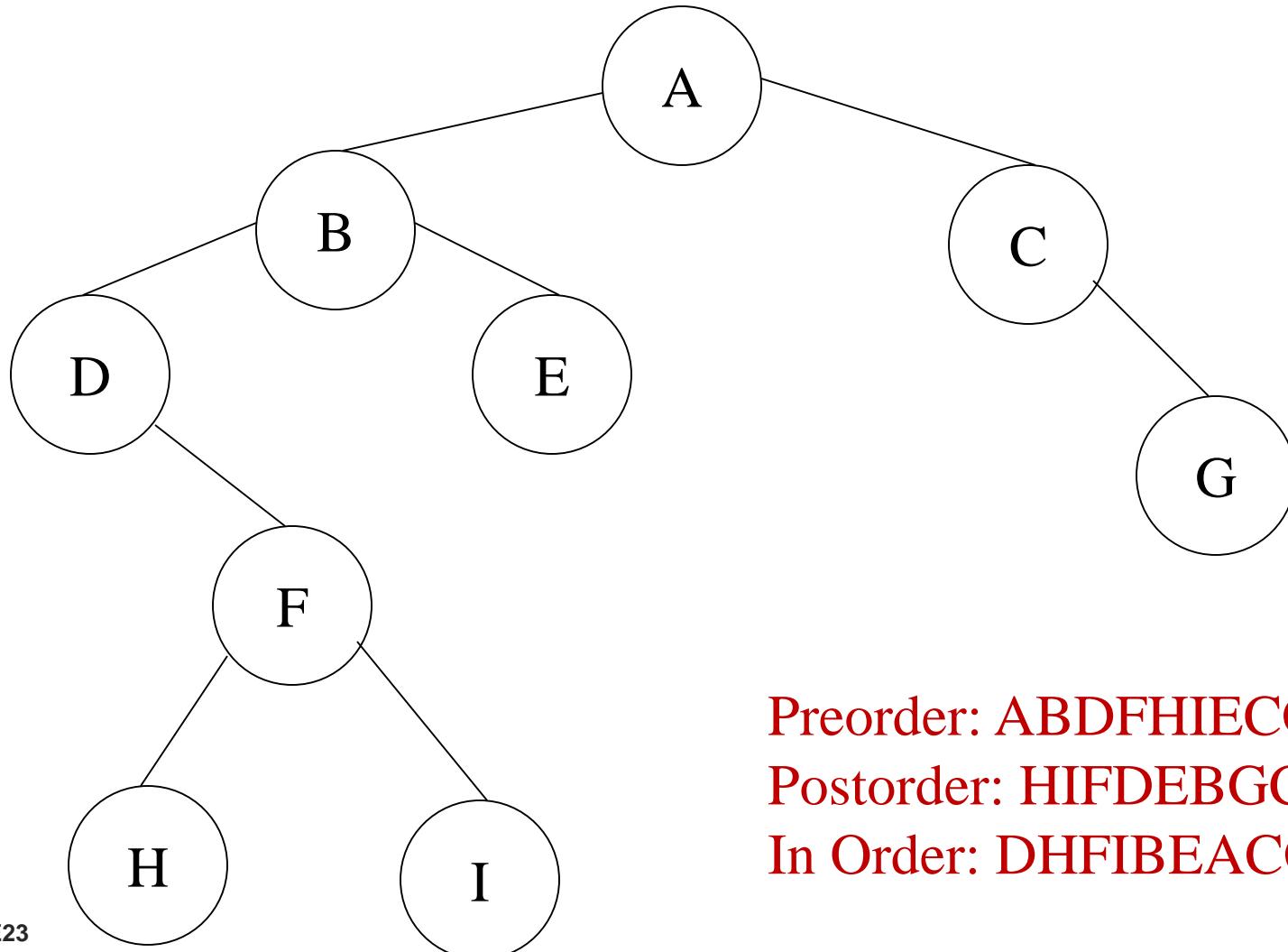
```
public static void LevelOrder(BinaryTreeNode t)
{ while (t != null)
{ // үндэсд зочлоод хүүхдүүдийг нь FIFO
// дараалалд хийнэ; зангилааг FIFO
// дарааллаас устгаж, дуудна t;
// дараалал хоосон бол устгаж null –ийг буцаана;
}
}
```

# Level-order нэвтрэлтийн жишээ

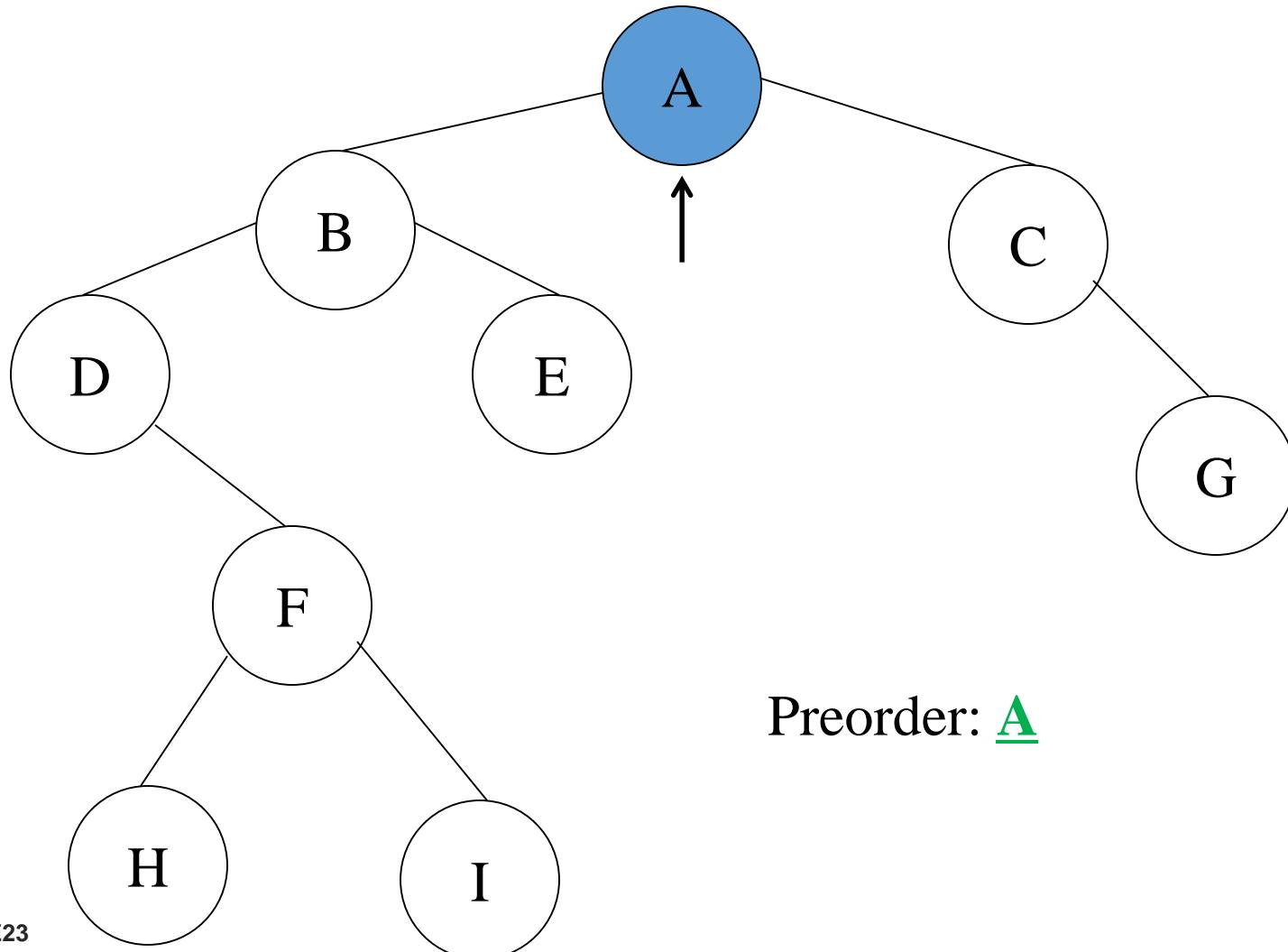


Level-order: F, B, G, A, D, I, C, E, H

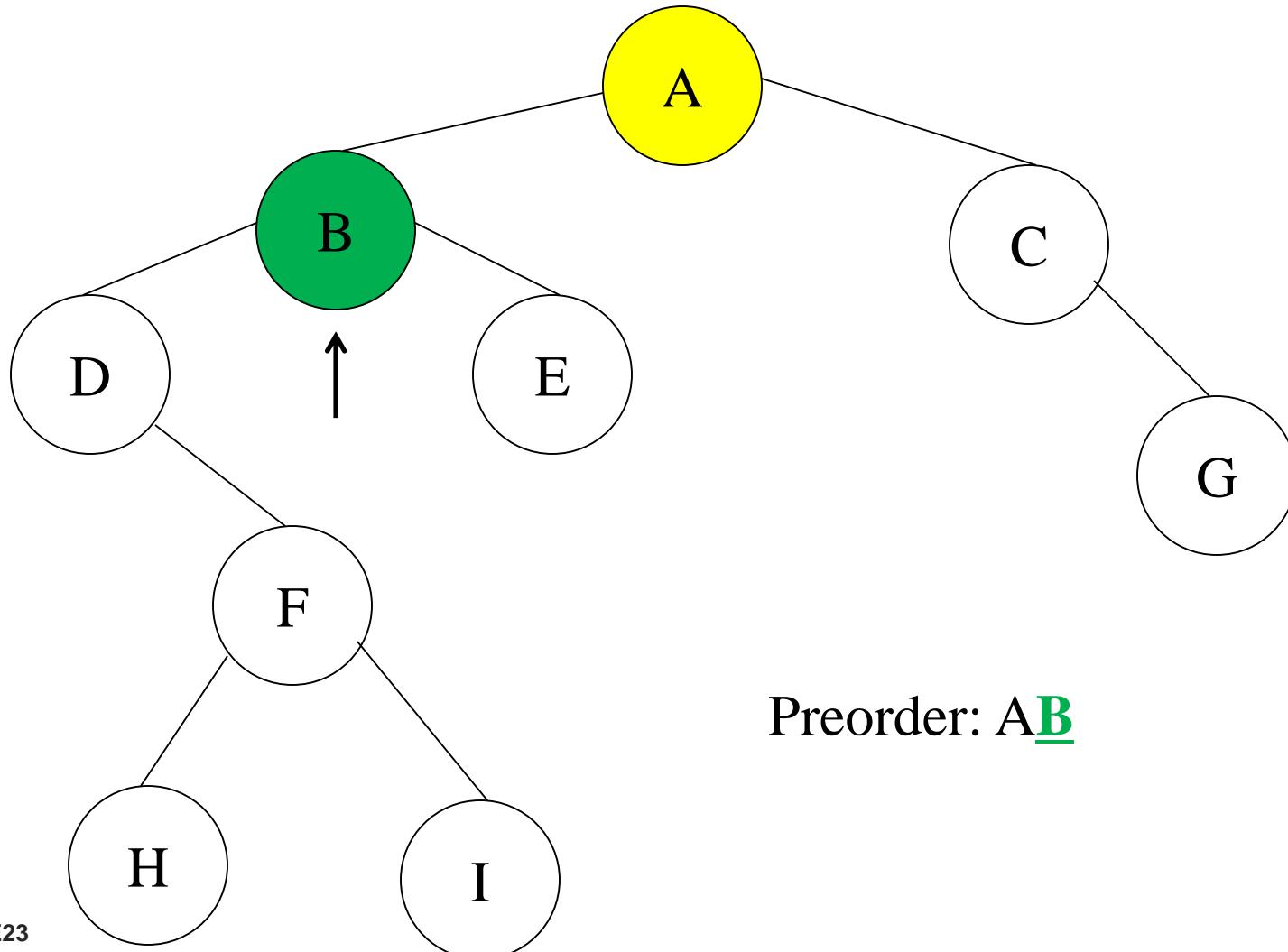
# Модны нэвтрэлтийн жишээ



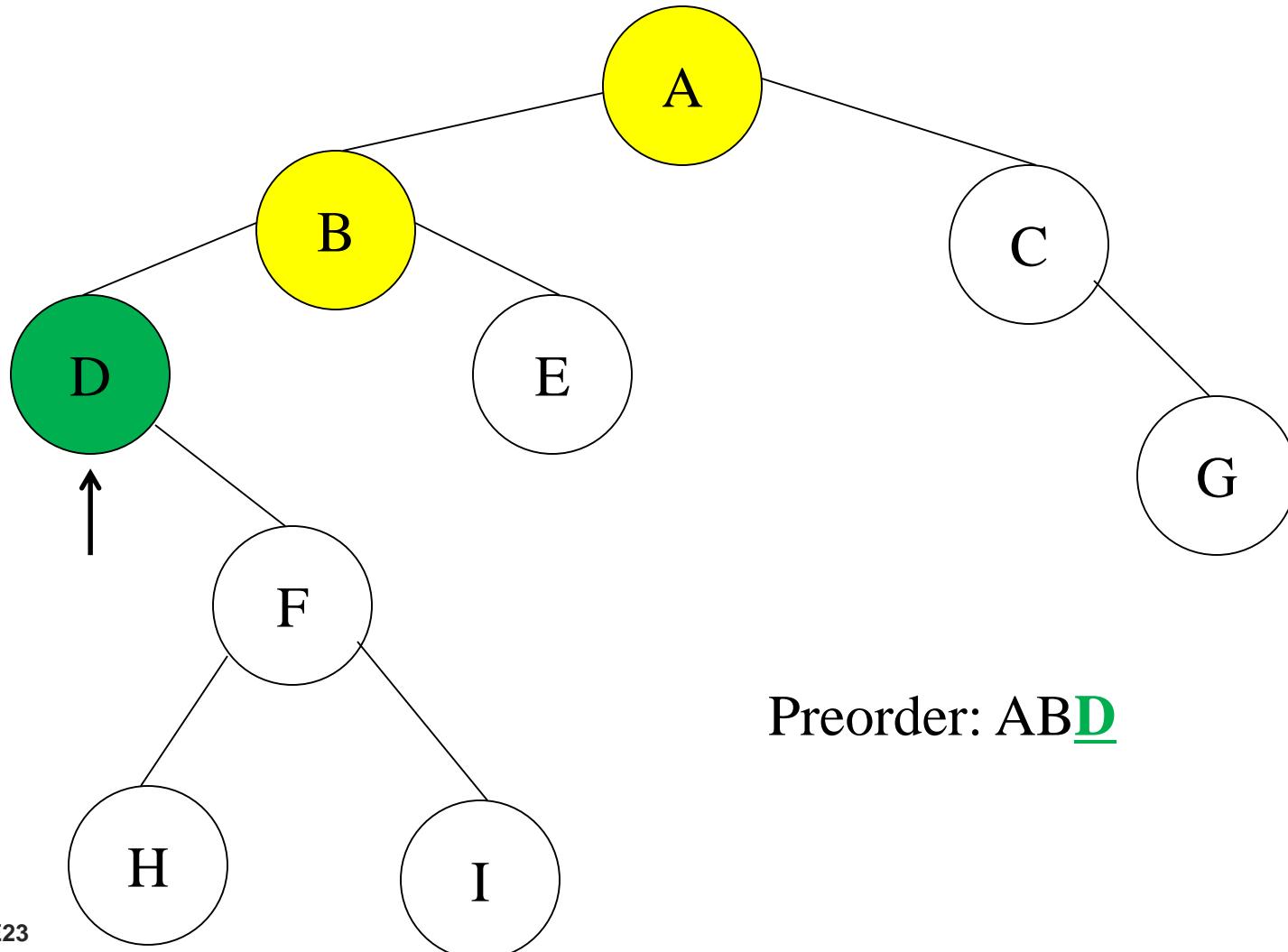
# Модны нэвтрэлтийн жишээ



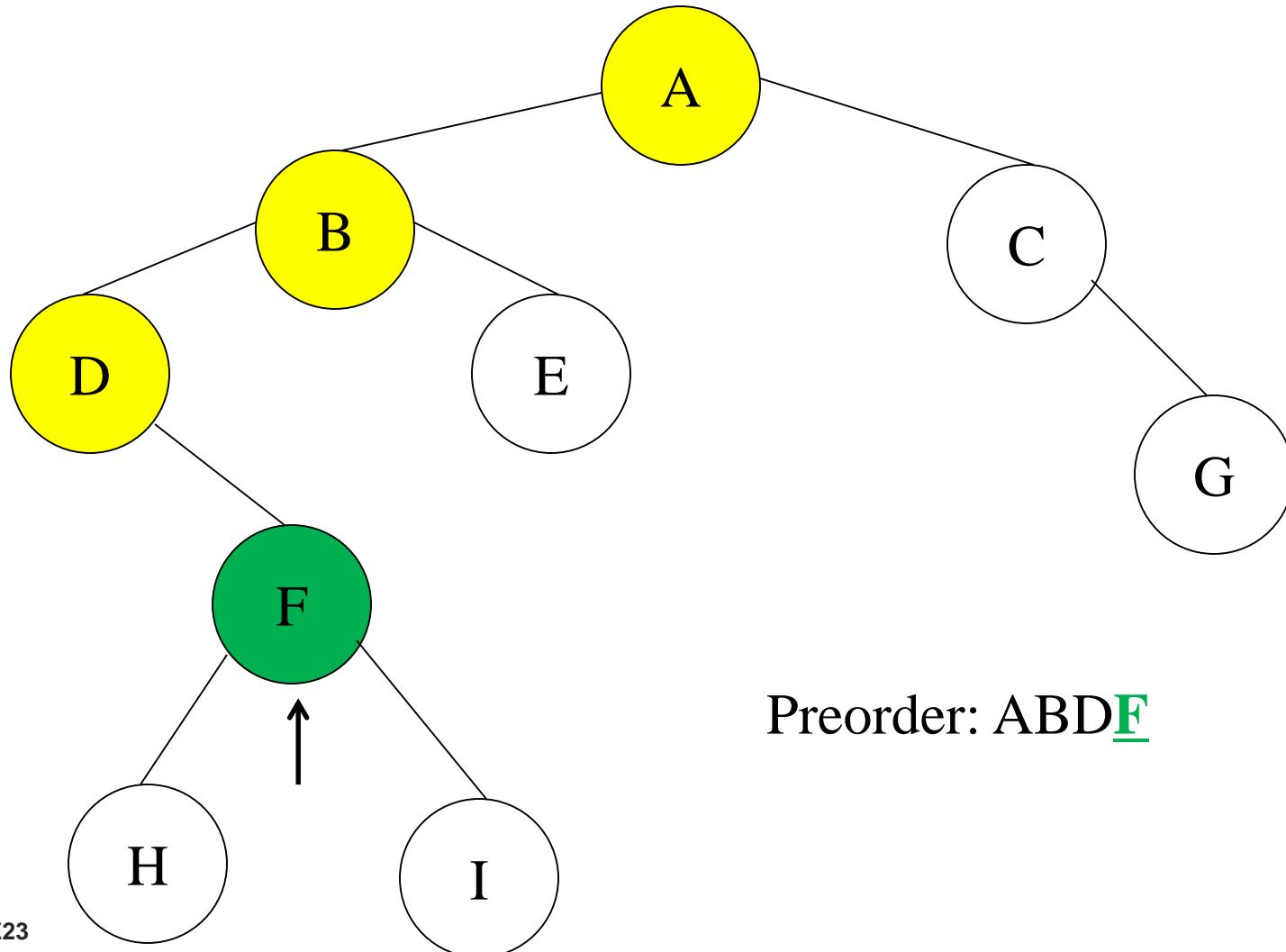
# Модны нэвтрэлтийн жишээ



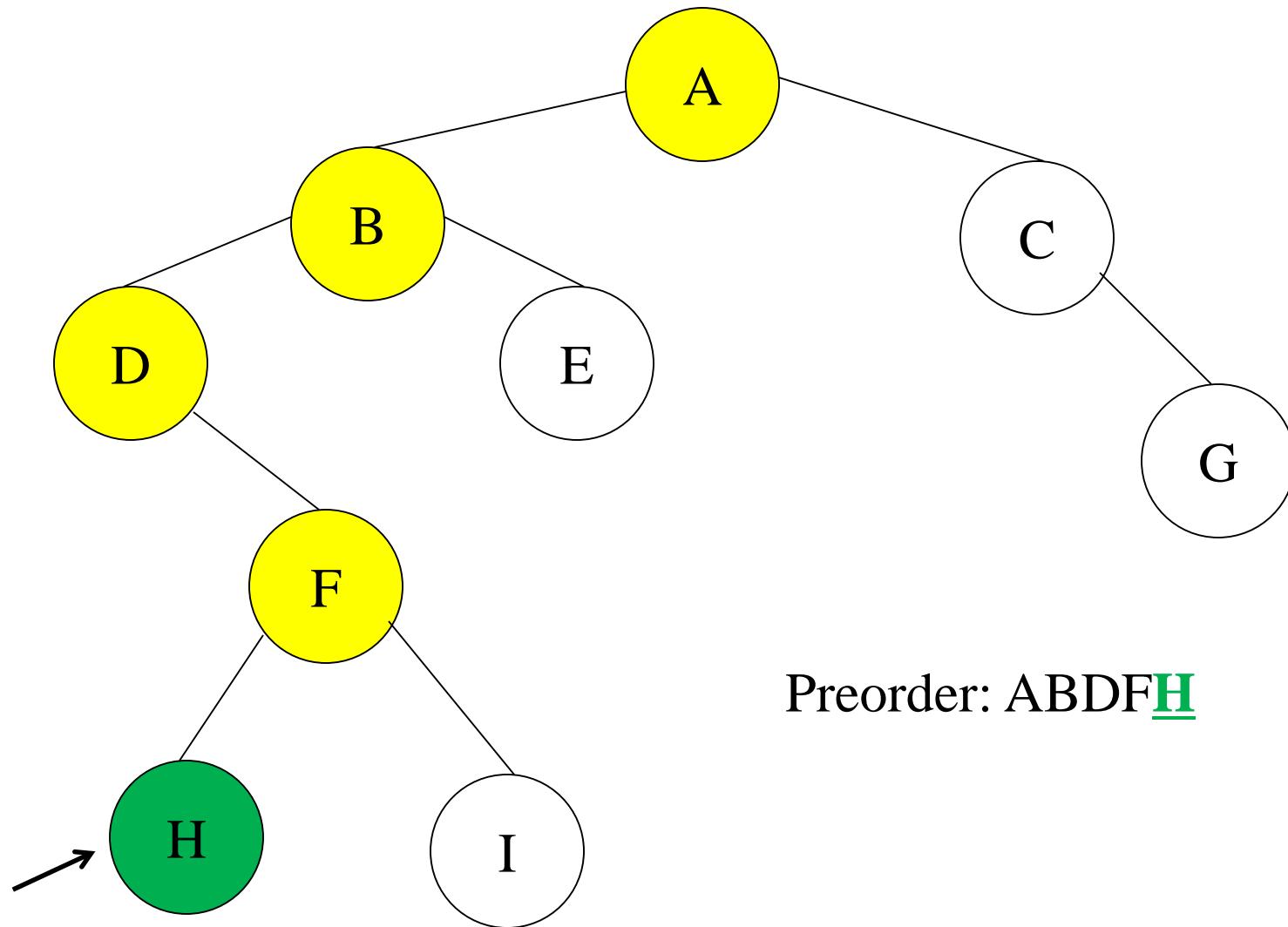
# Модны нэвтрэлтийн жишээ



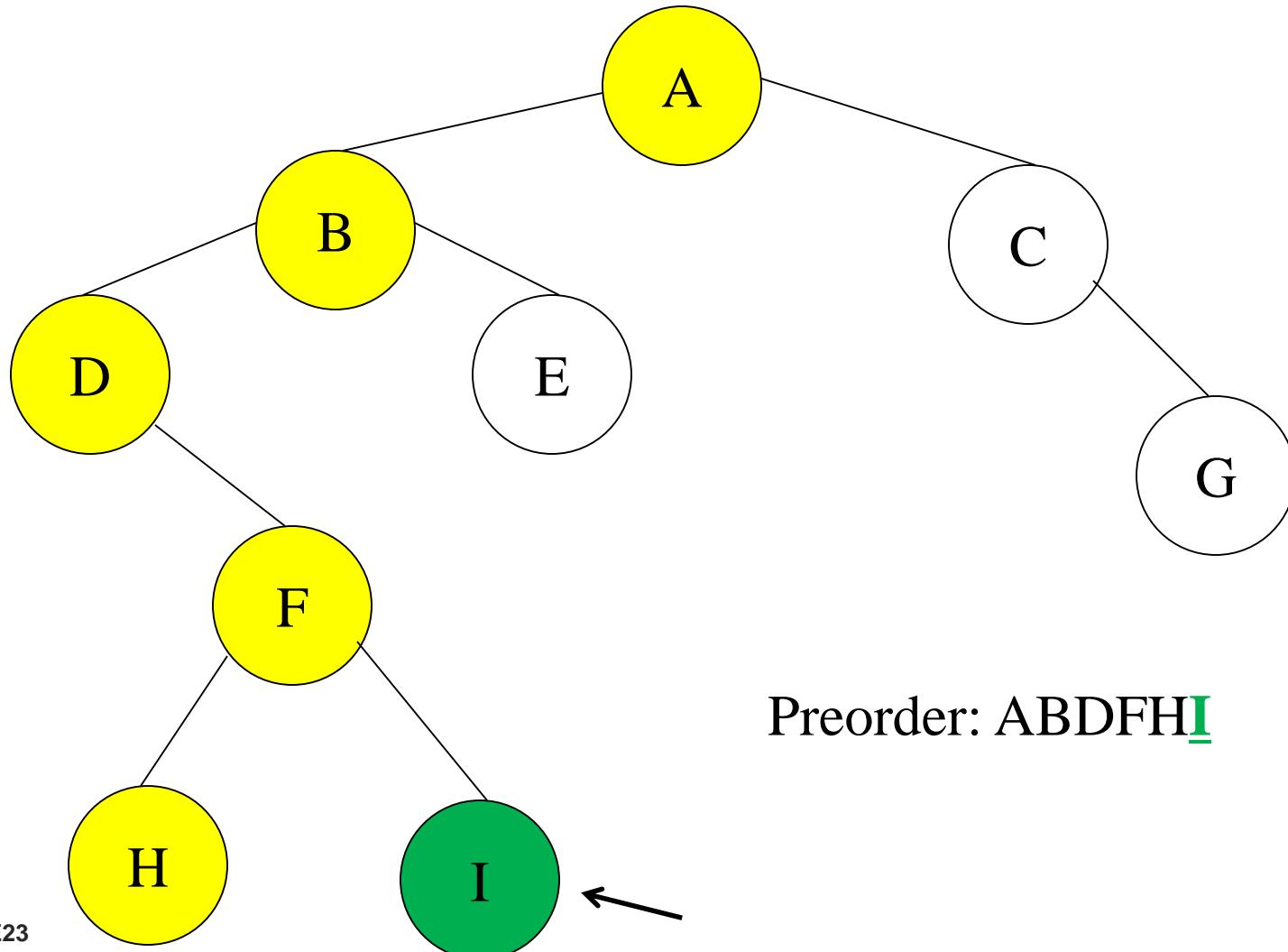
# Модны нэвтрэлтийн жишээ



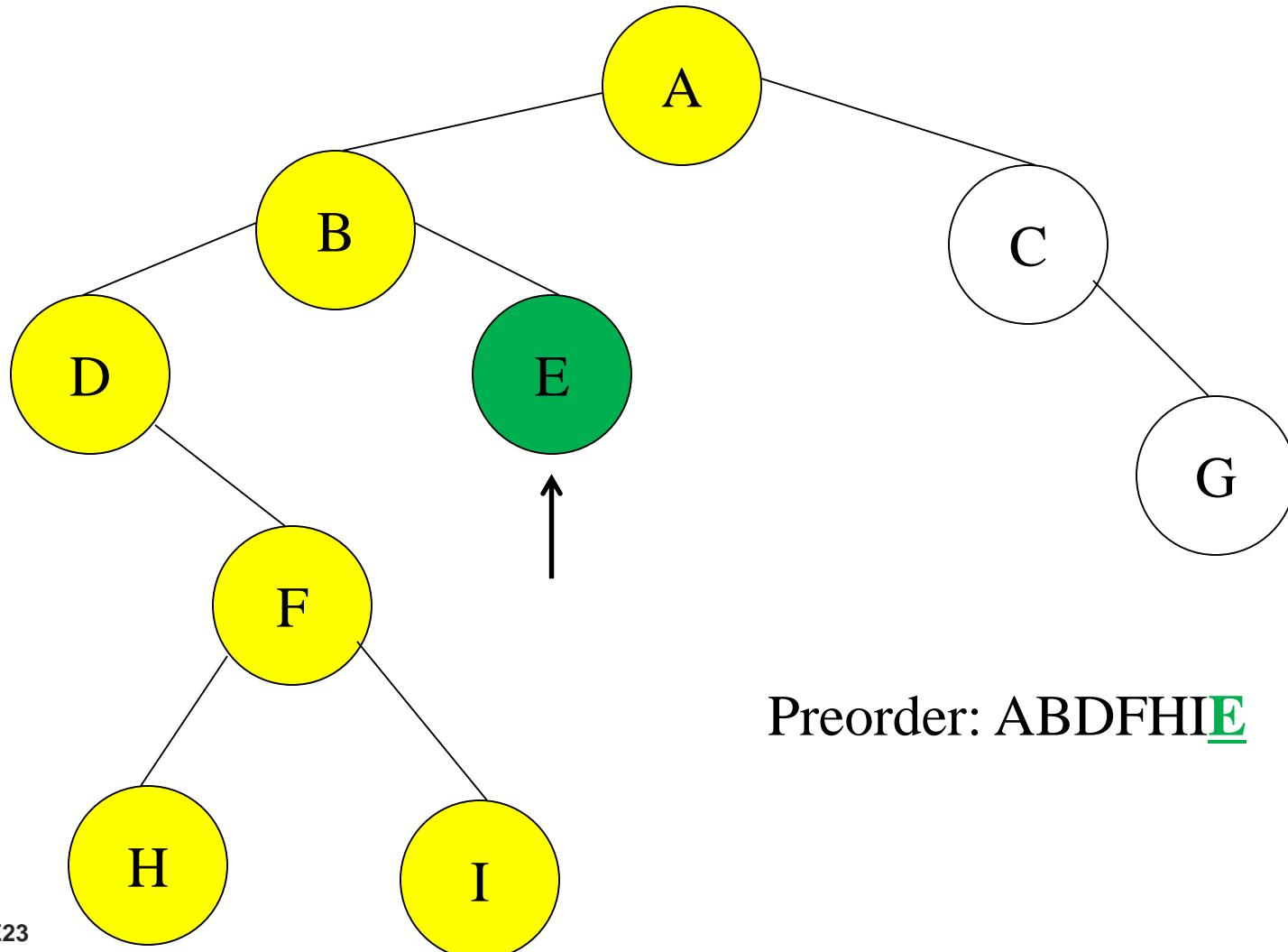
# Модны нэвтрэлтийн жишээ



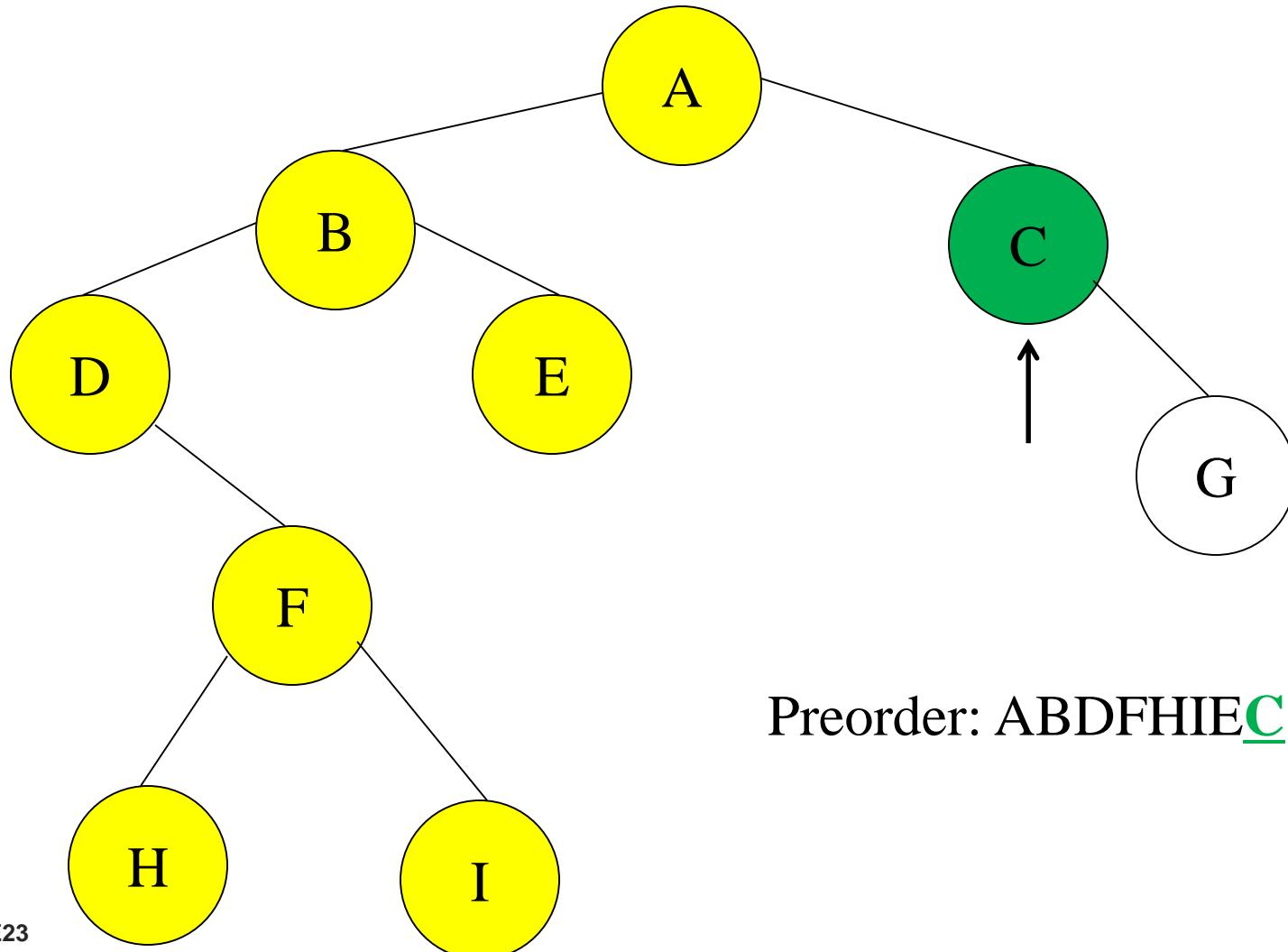
# Модны нэвтрэлтийн жишээ



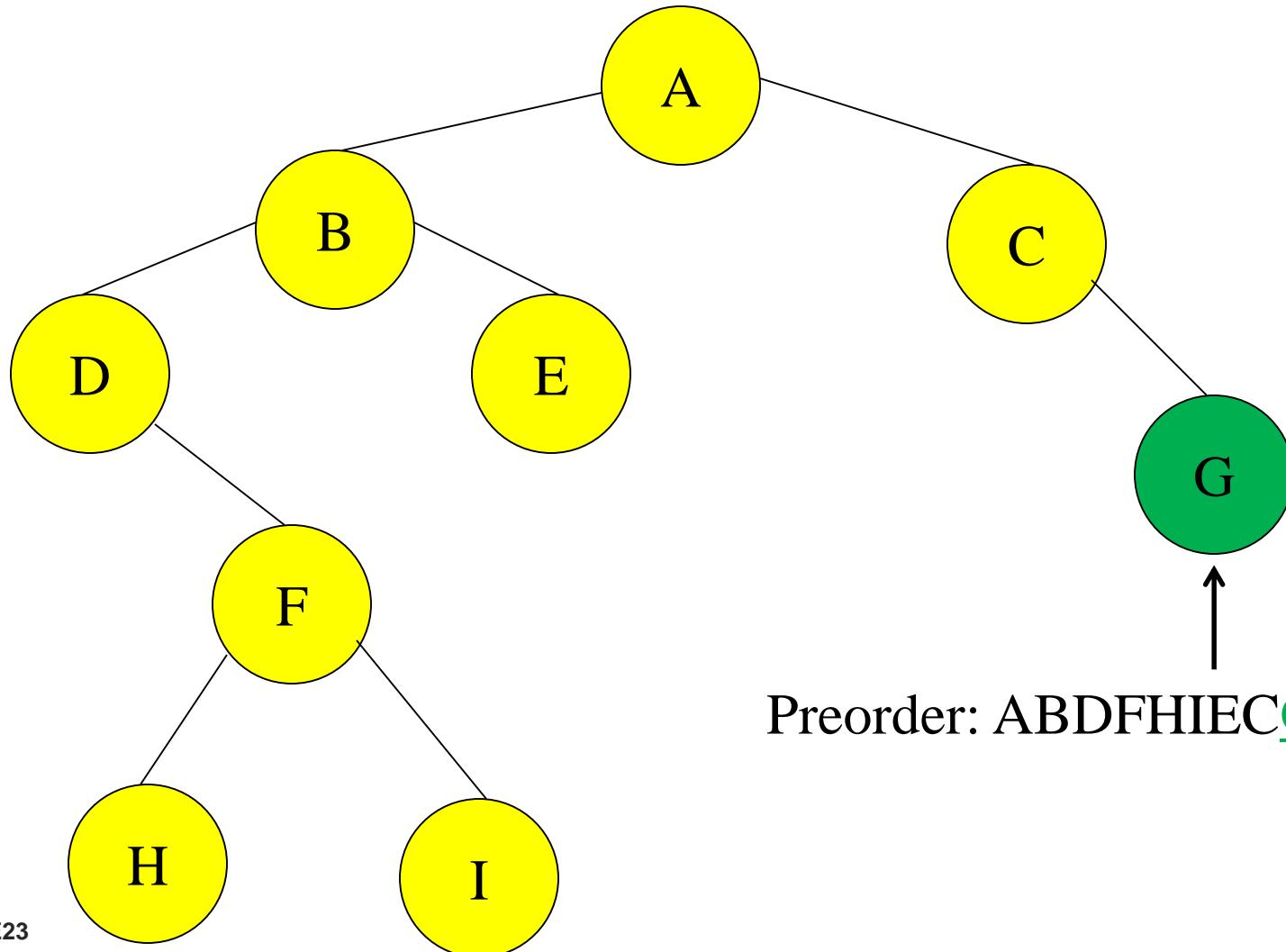
# Модны нэвтрэлтийн жишээ



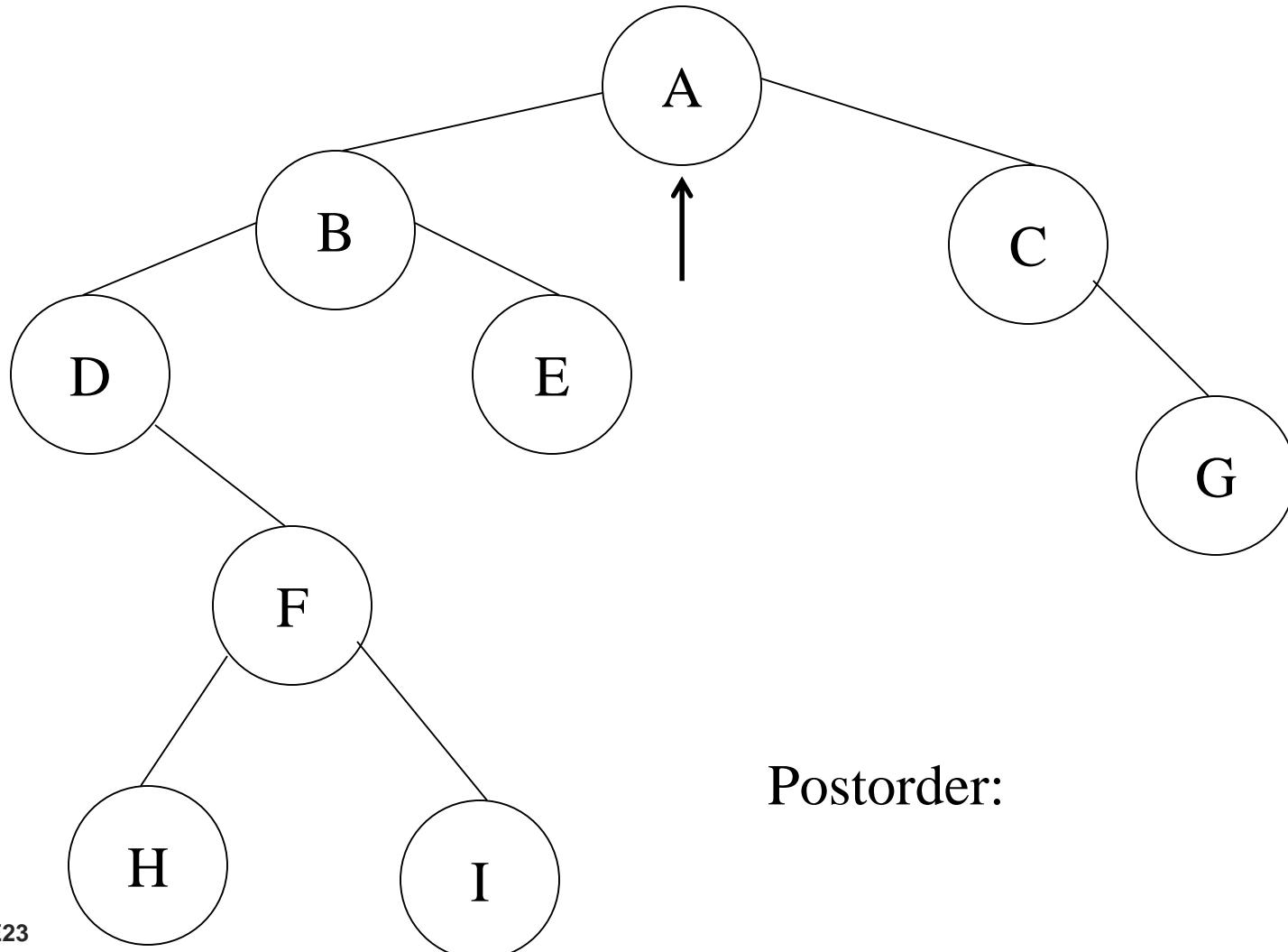
# Модны нэвтрэлтийн жишээ



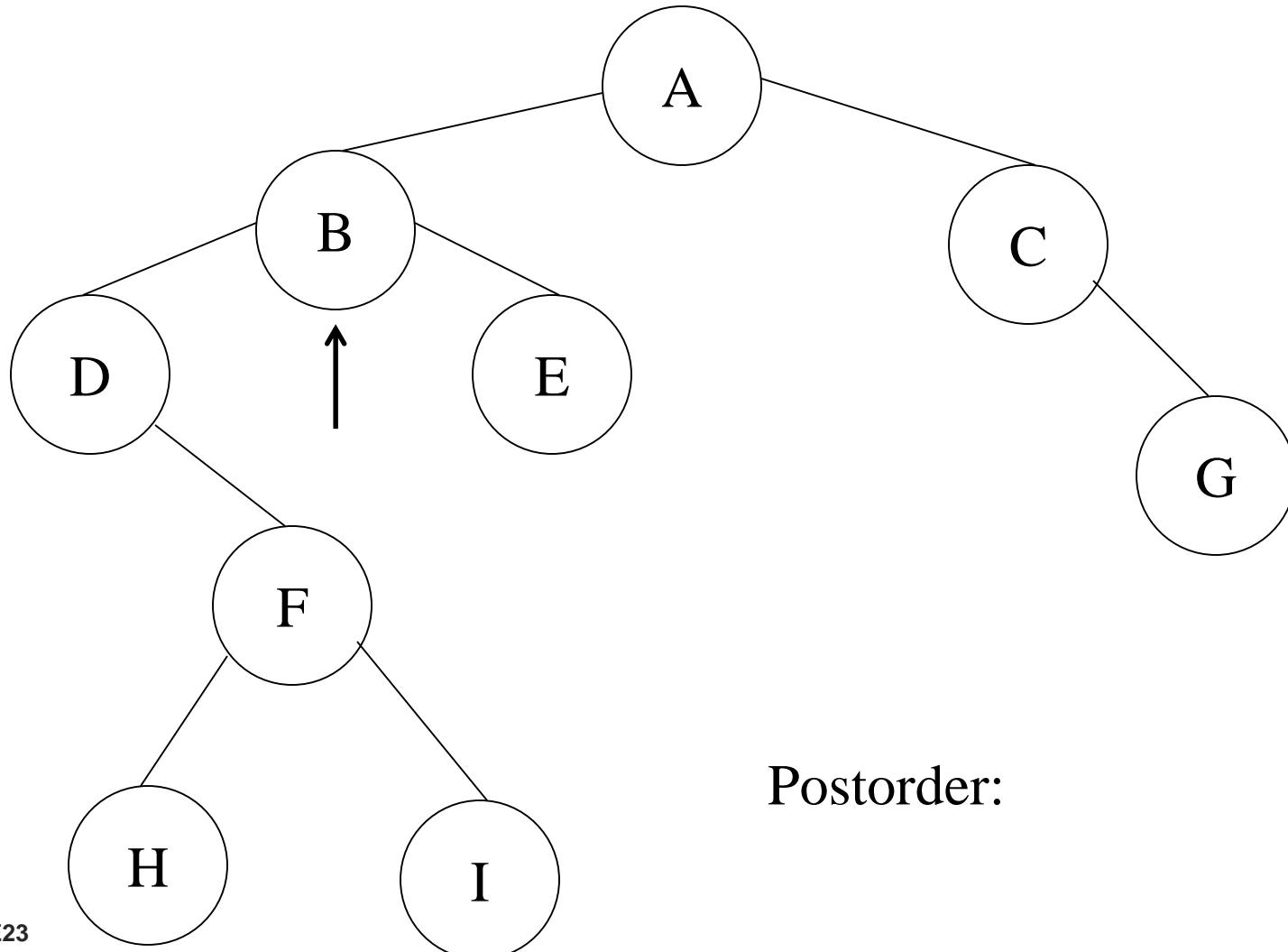
# Модны нэвтрэлтийн жишээ



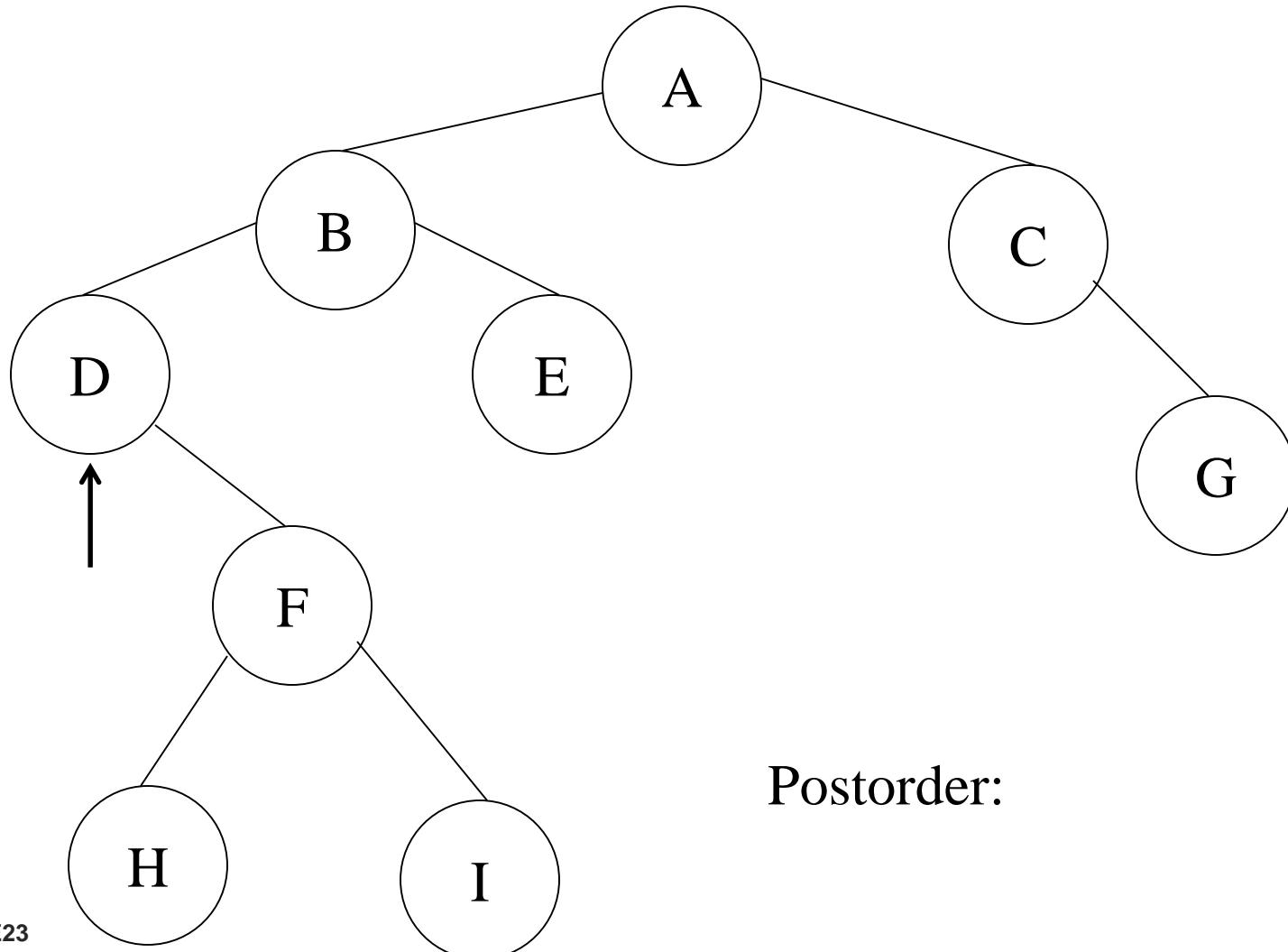
# Модны нэвтрэлтийн жишээ



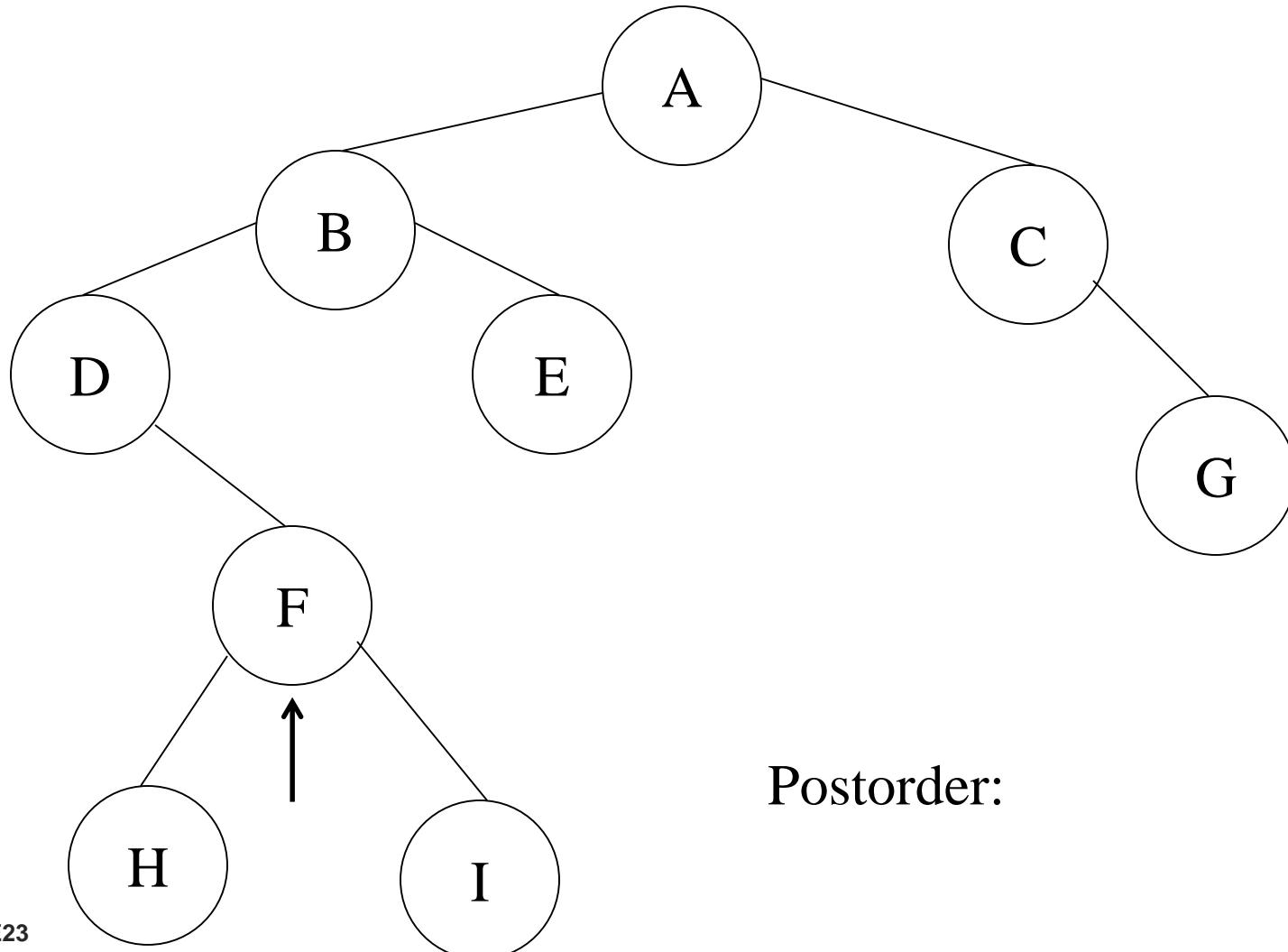
# Модны нэвтрэлтийн жишээ



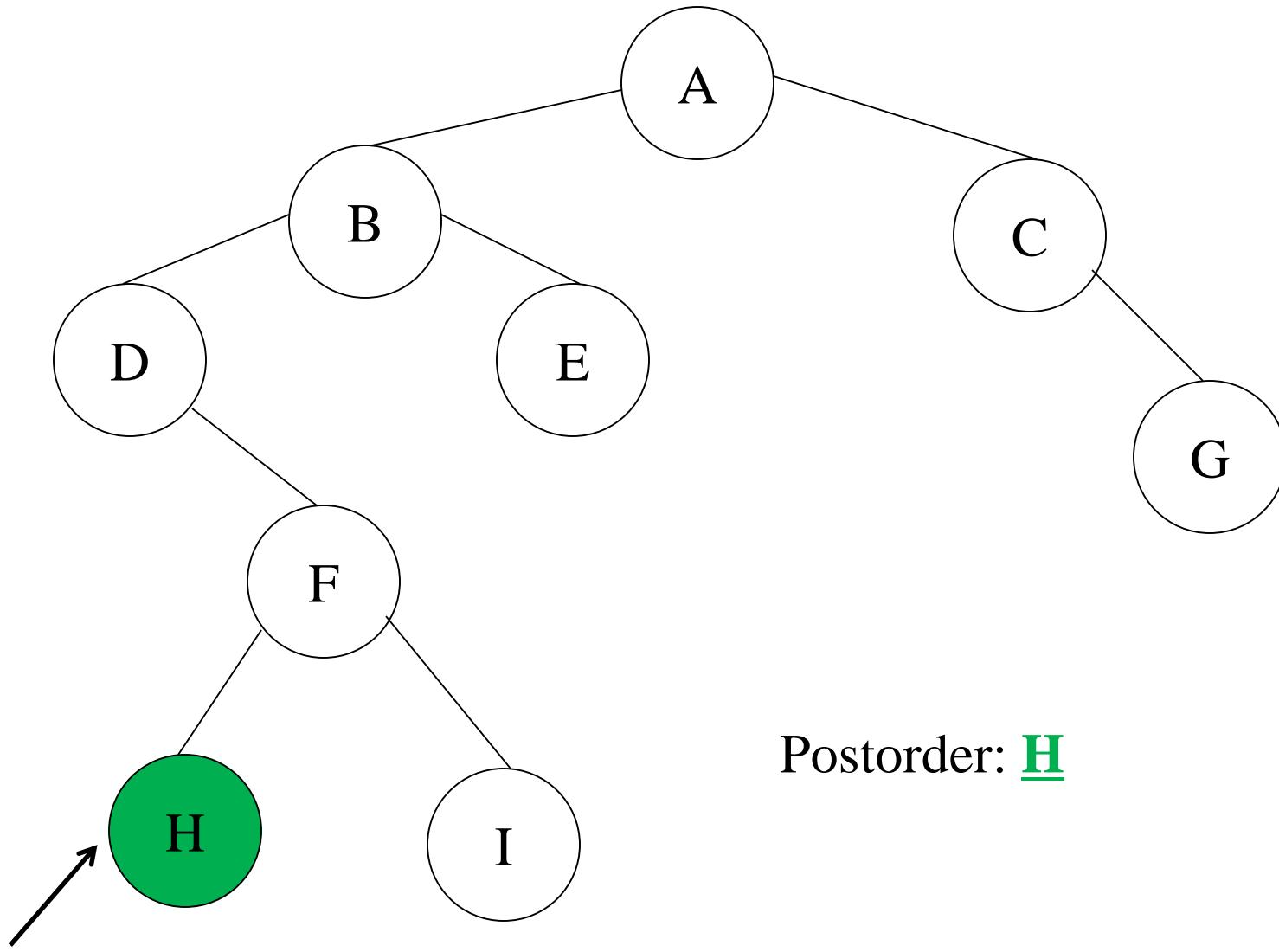
# Модны нэвтрэлтийн жишээ



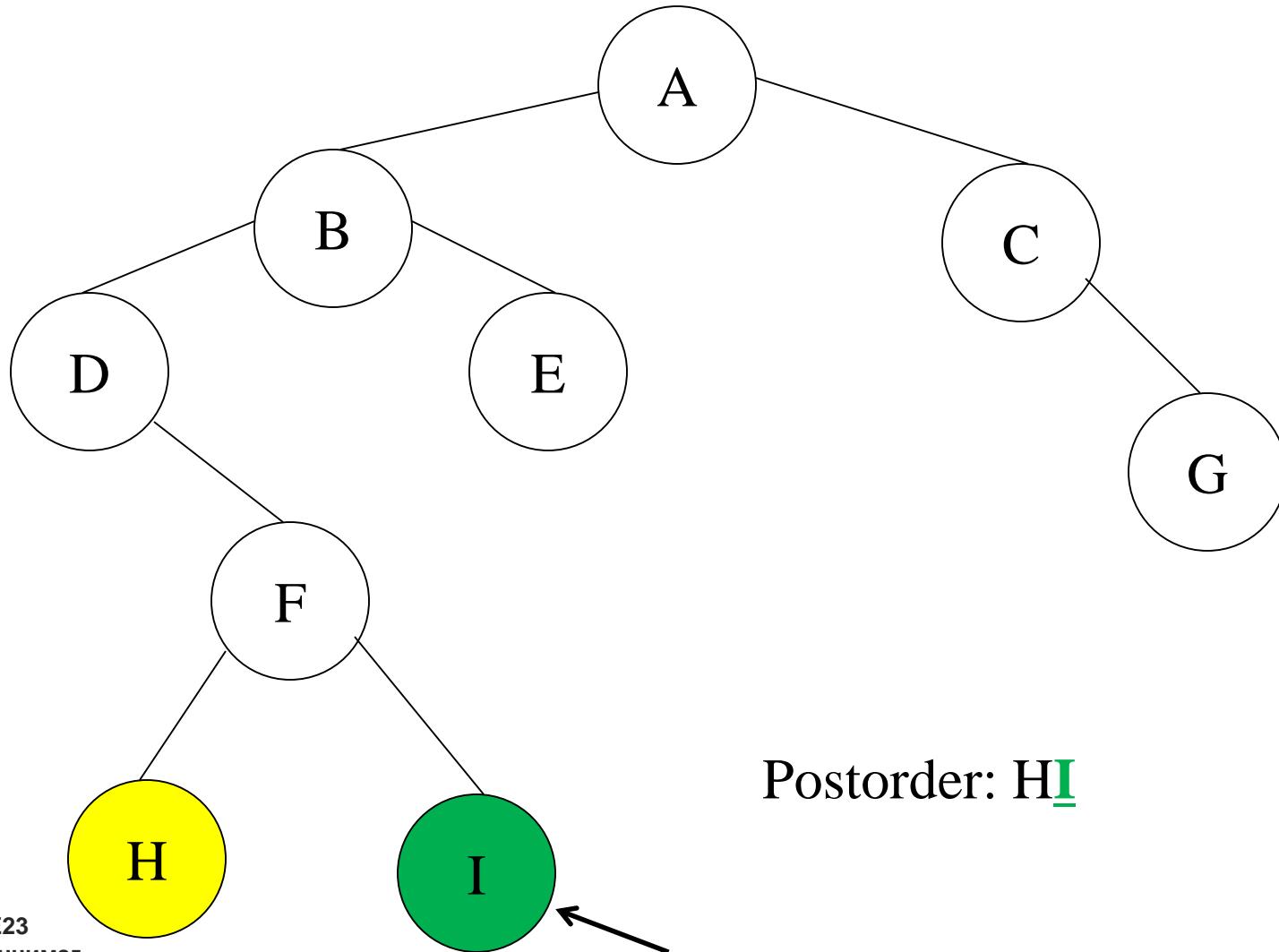
# Модны нэвтрэлтийн жишээ



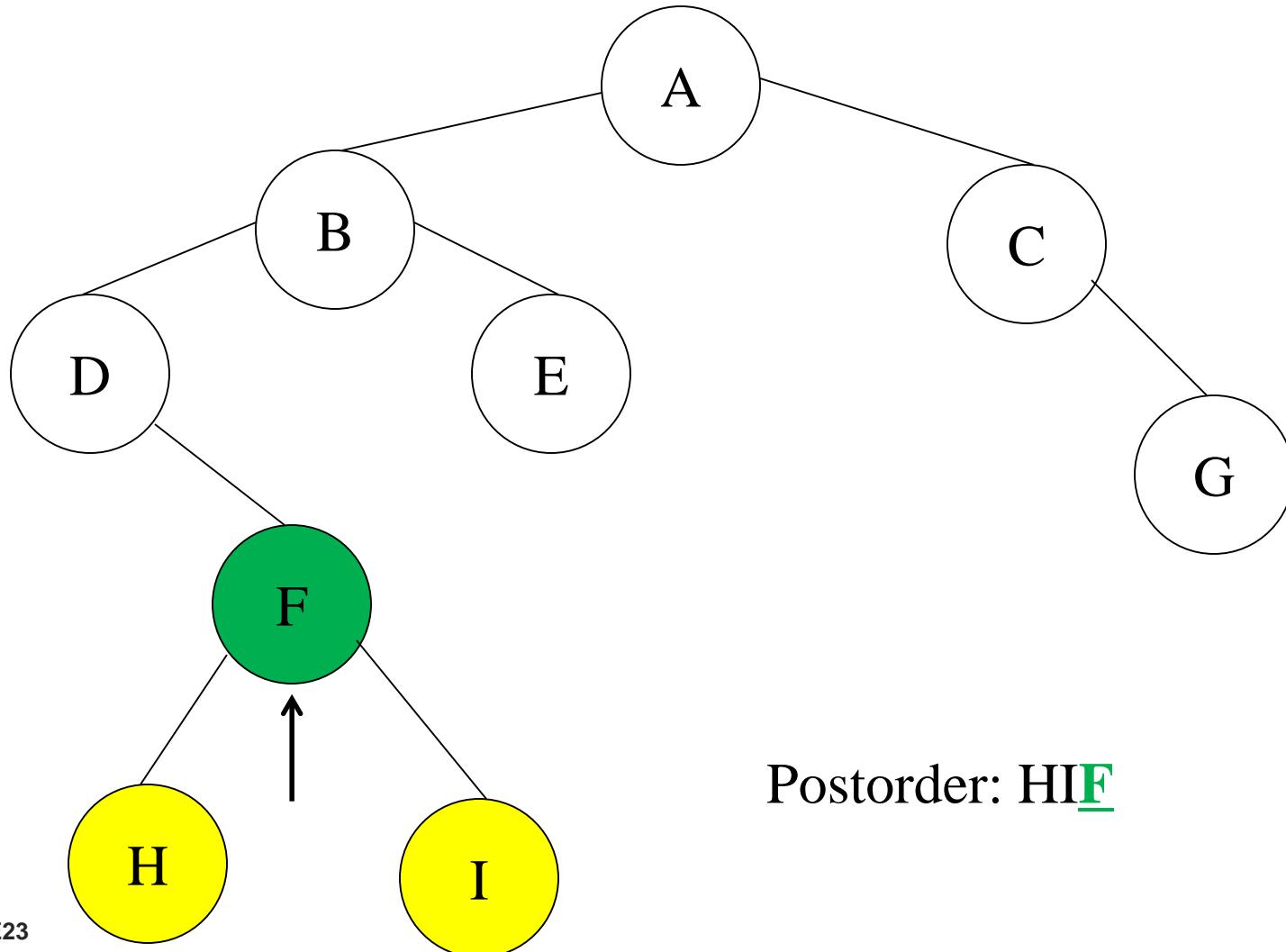
# Модны нэвтрэлтийн жишээ



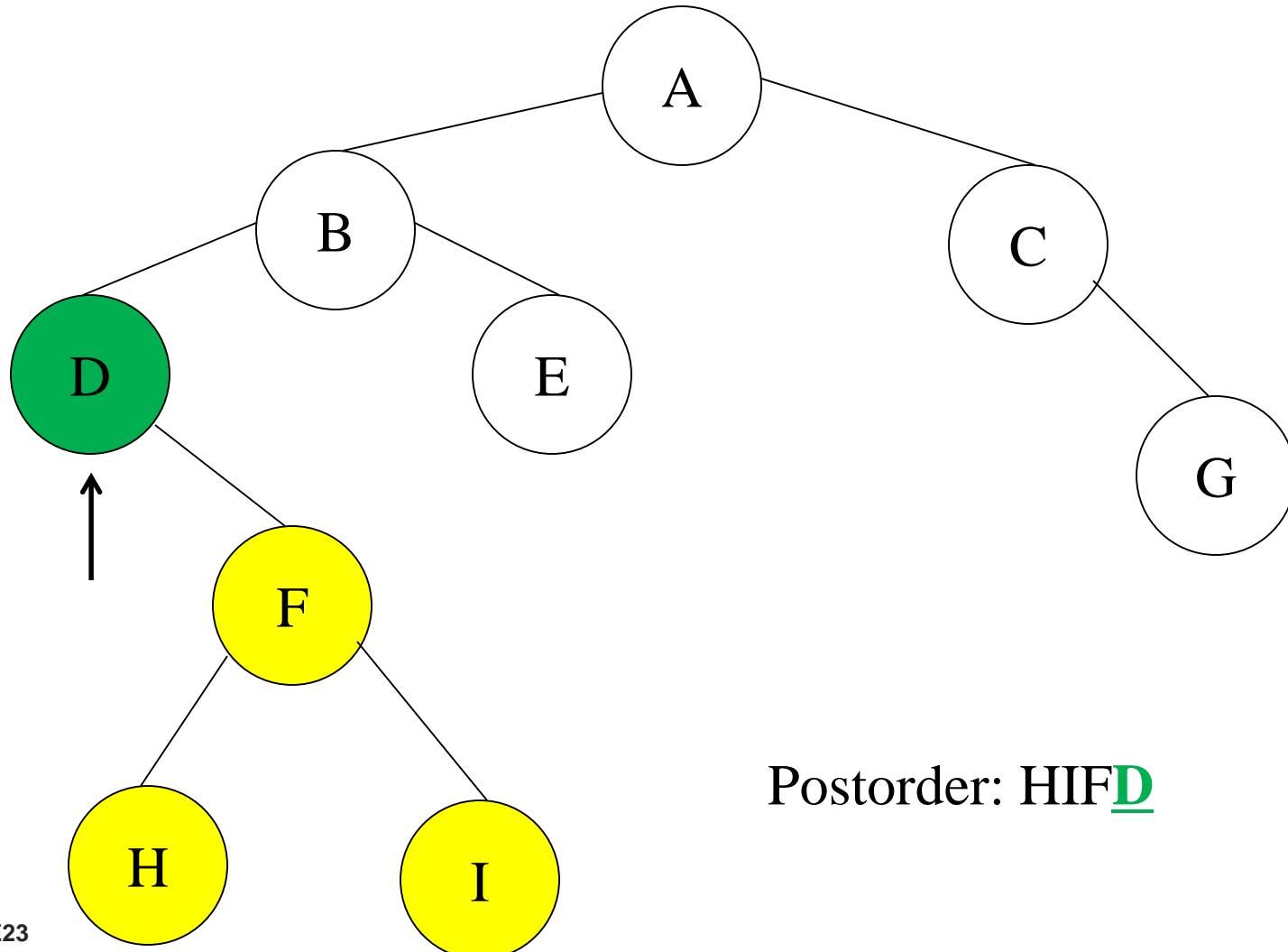
# Модны нэвтрэлтийн жишээ



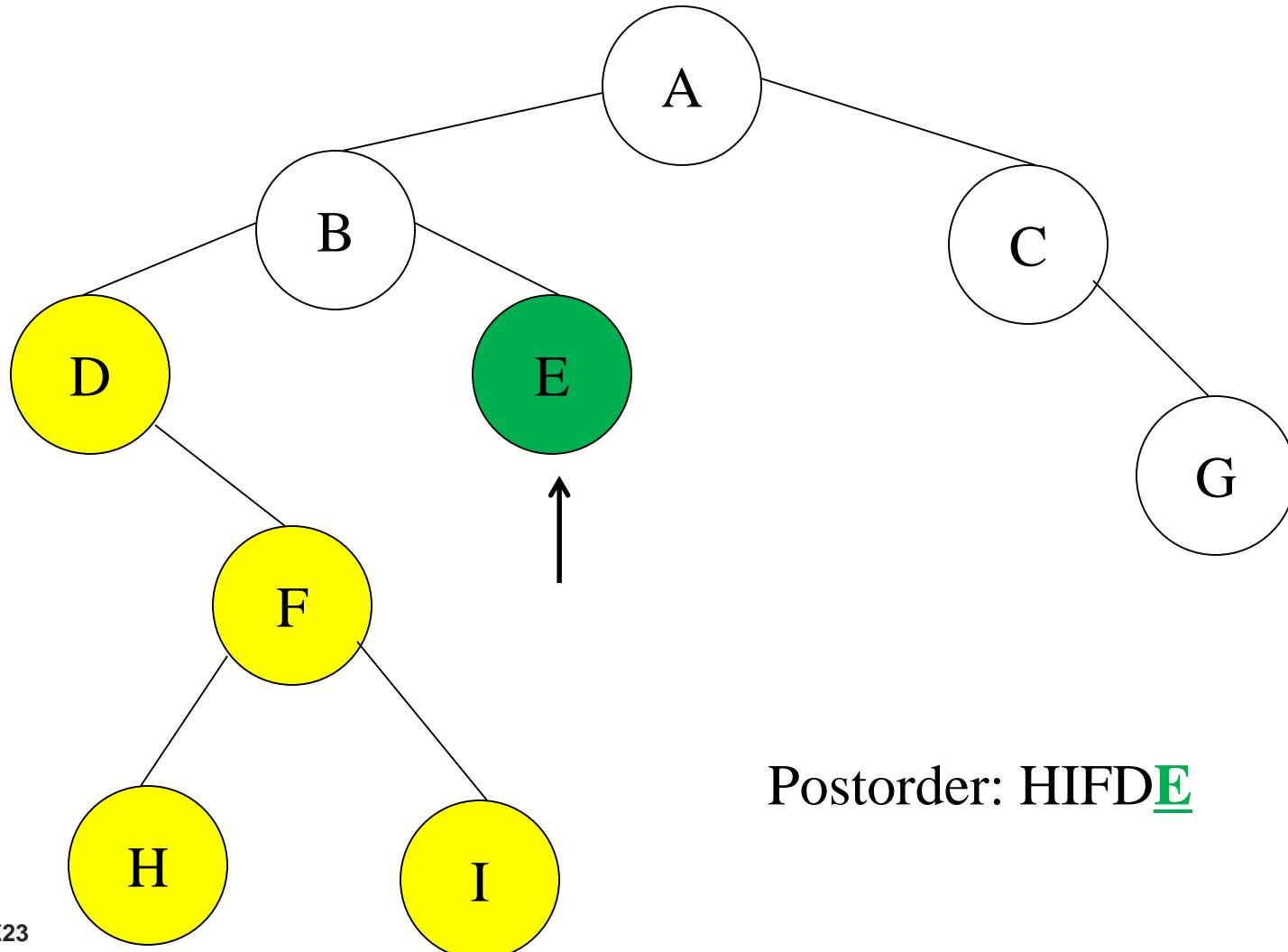
# Модны нэвтрэлтийн жишээ



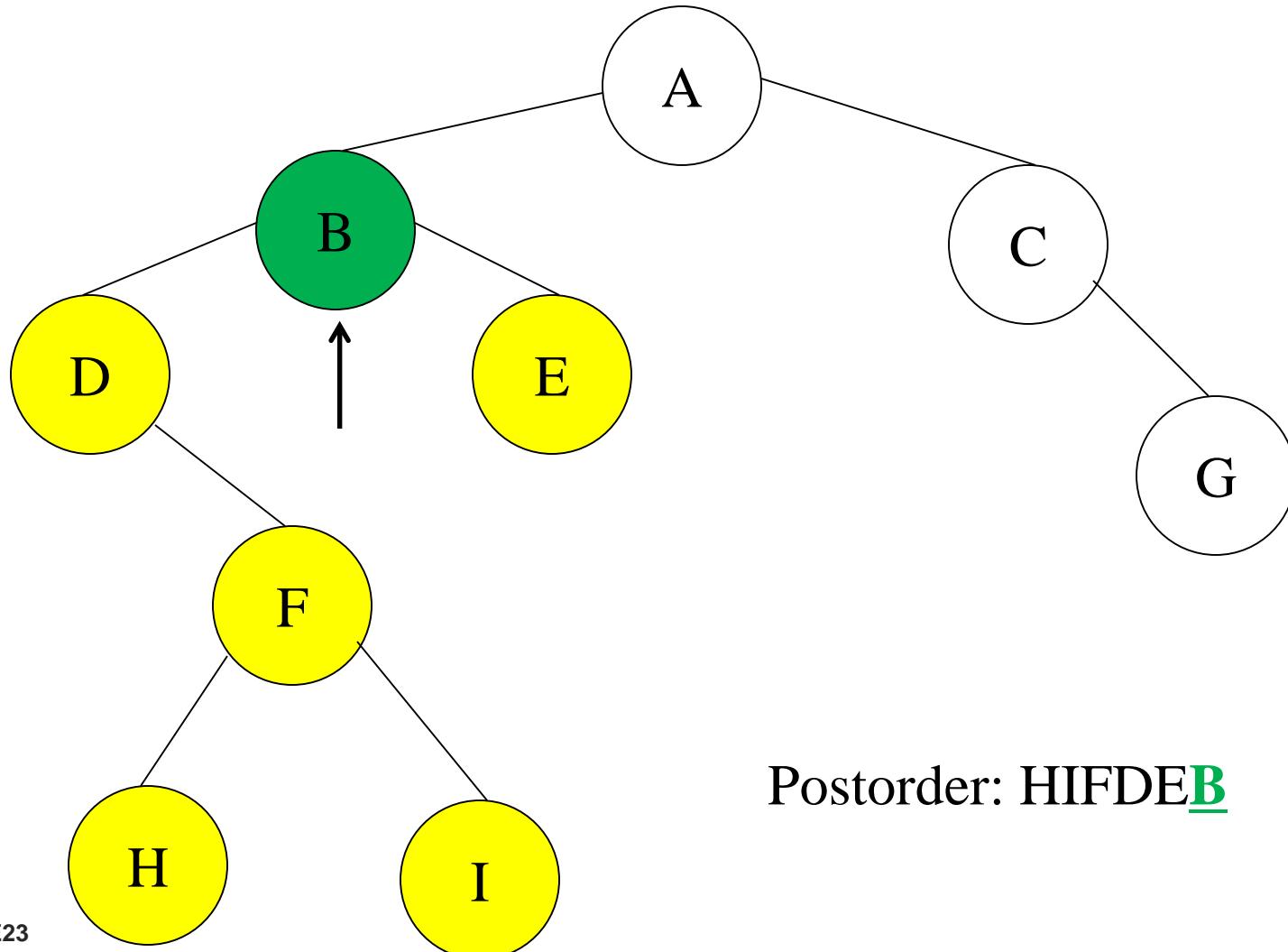
# Модны нэвтрэлтийн жишээ



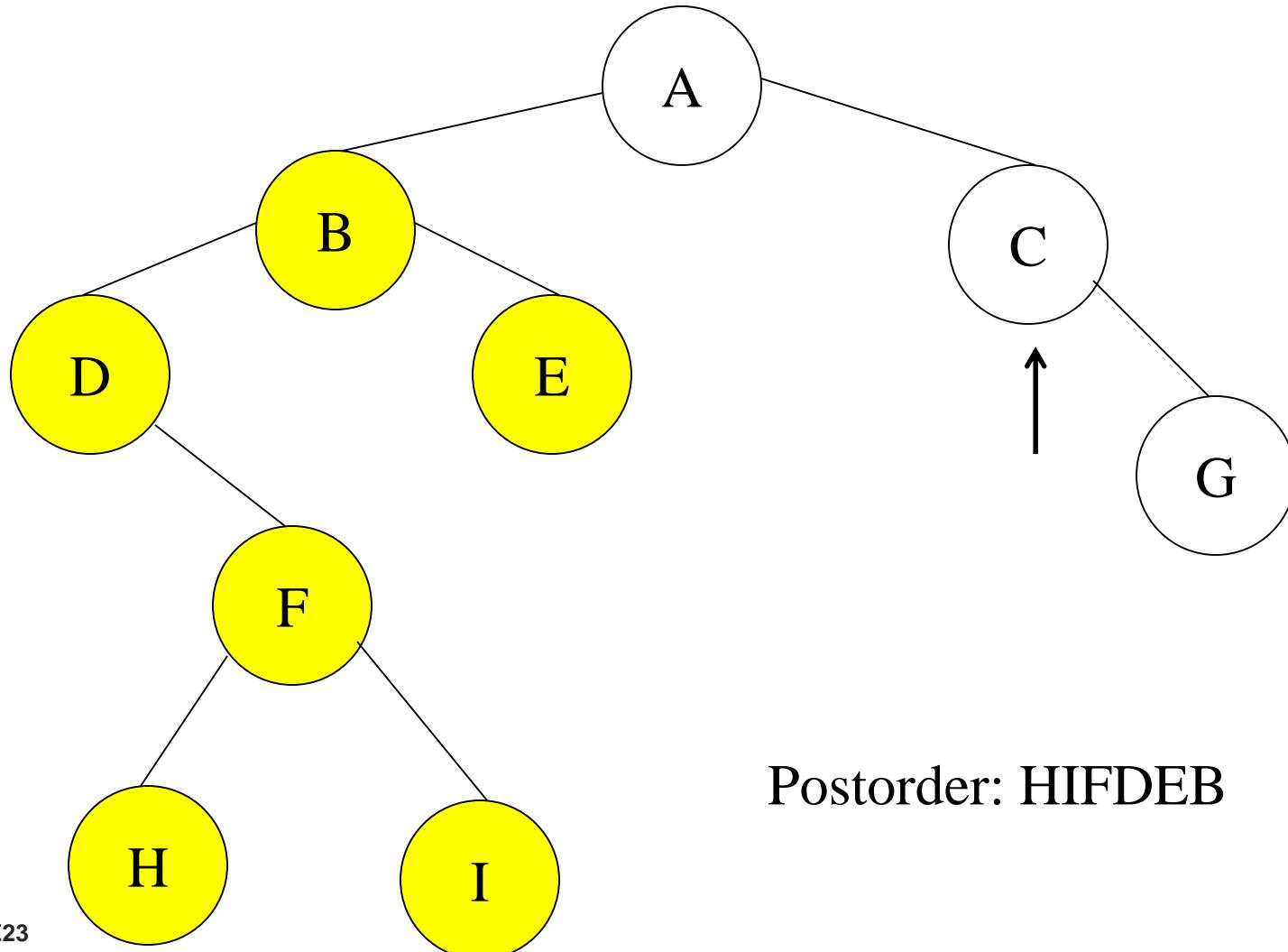
# Модны нэвтрэлтийн жишээ



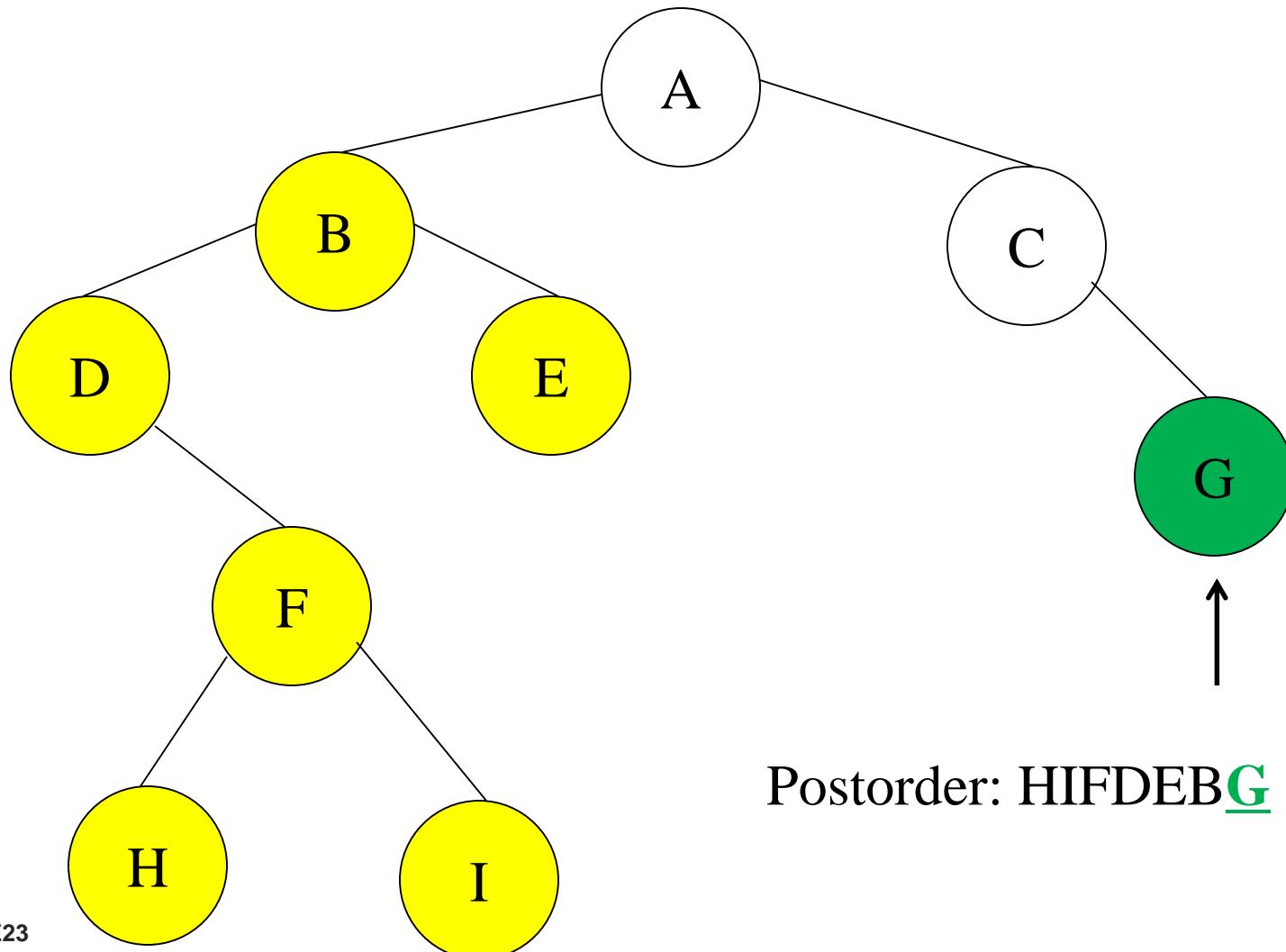
# Модны нэвтрэлтийн жишээ



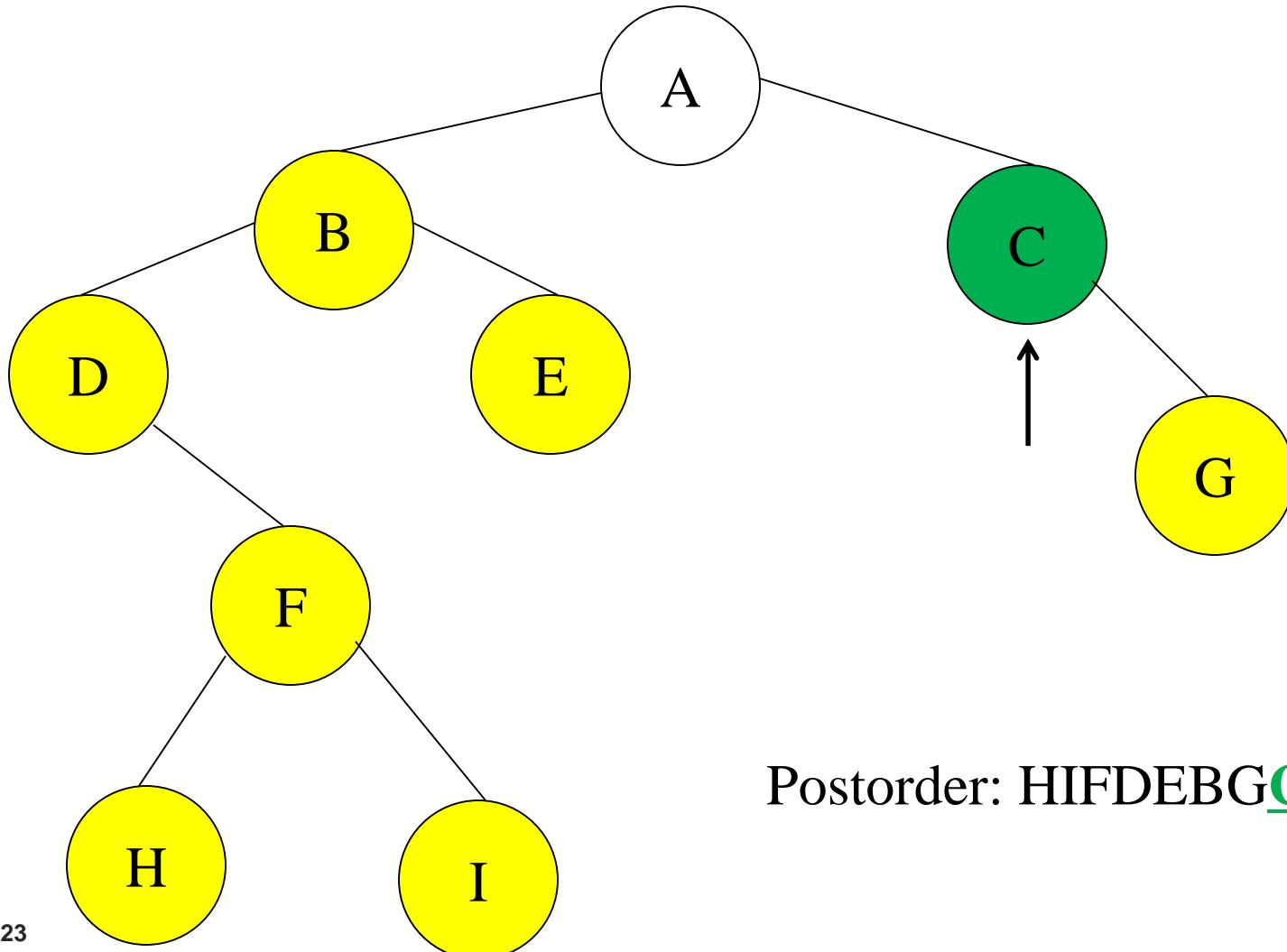
# Модны нэвтрэлтийн жишээ



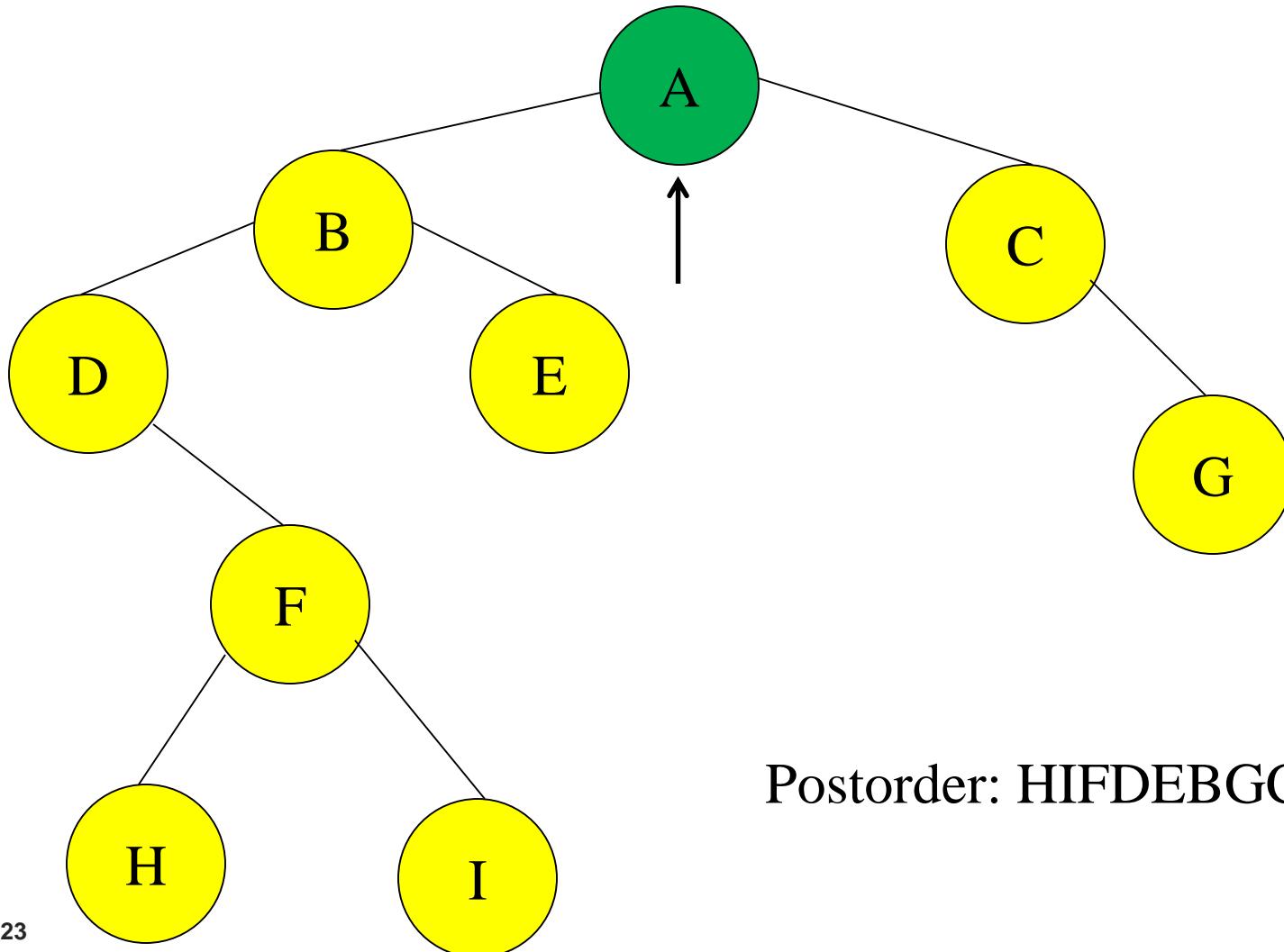
# Модны нэвтрэлтийн жишээ



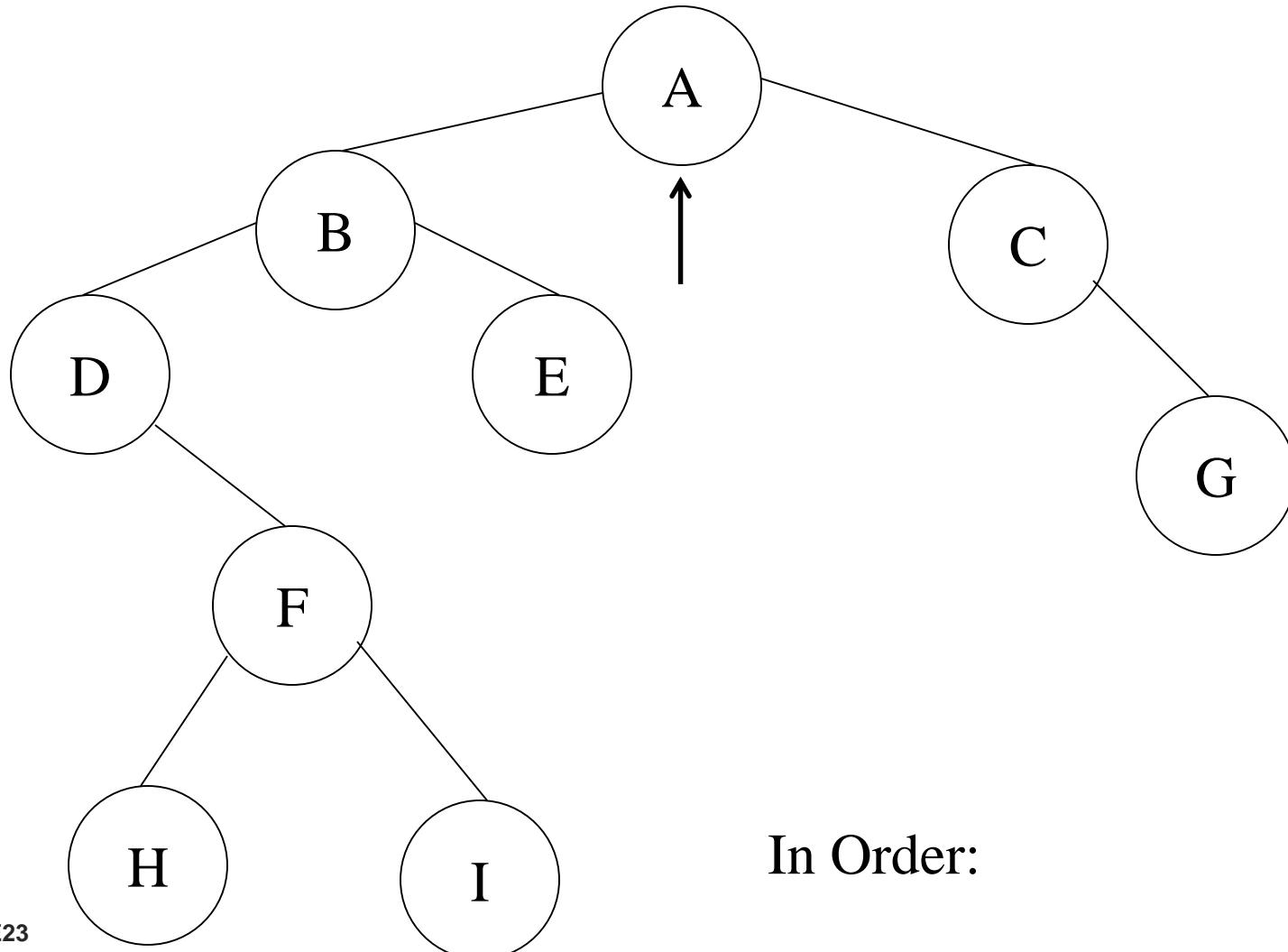
# Модны нэвтрэлтийн жишээ



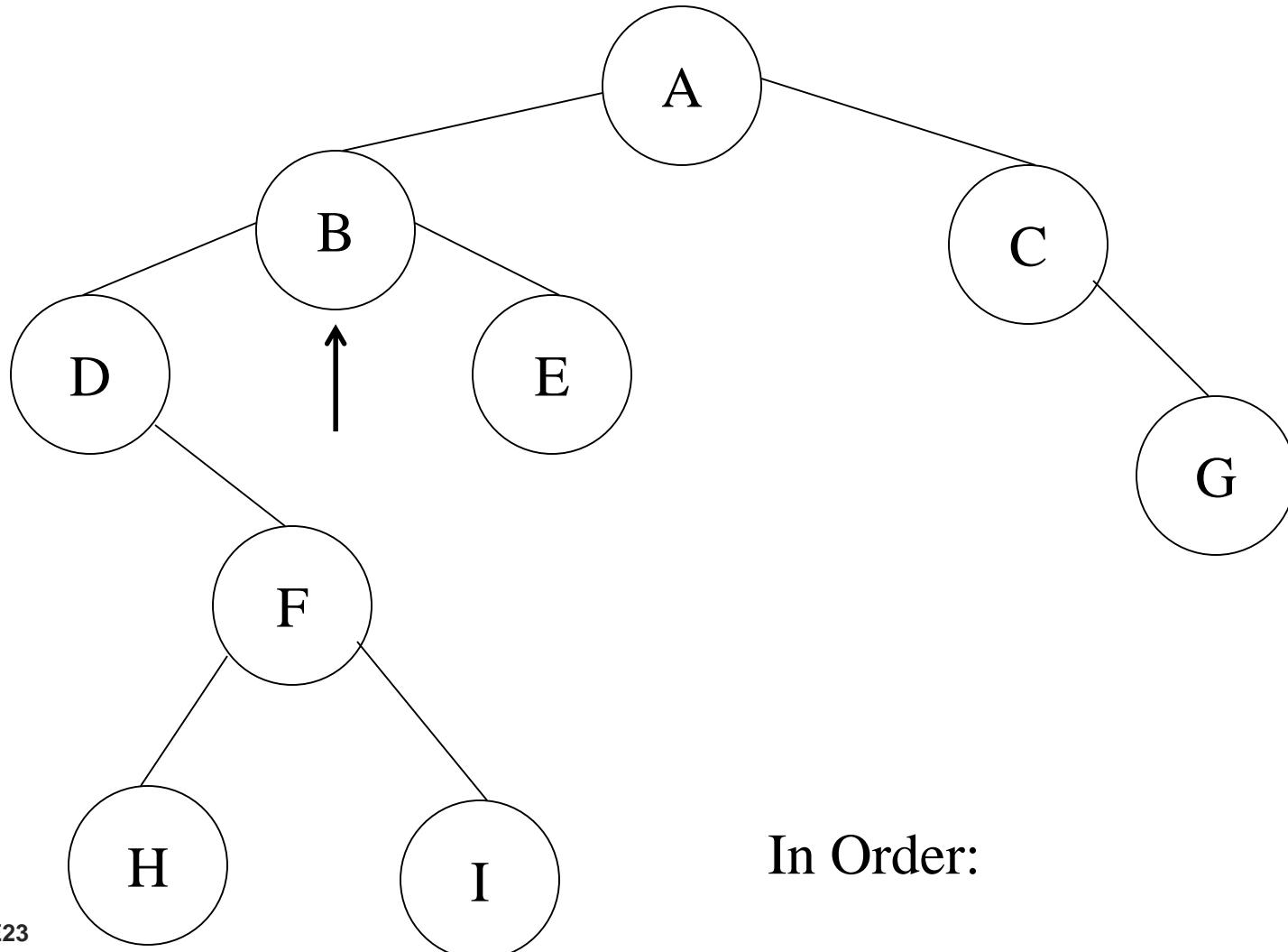
# Модны нэвтрэлтийн жишээ



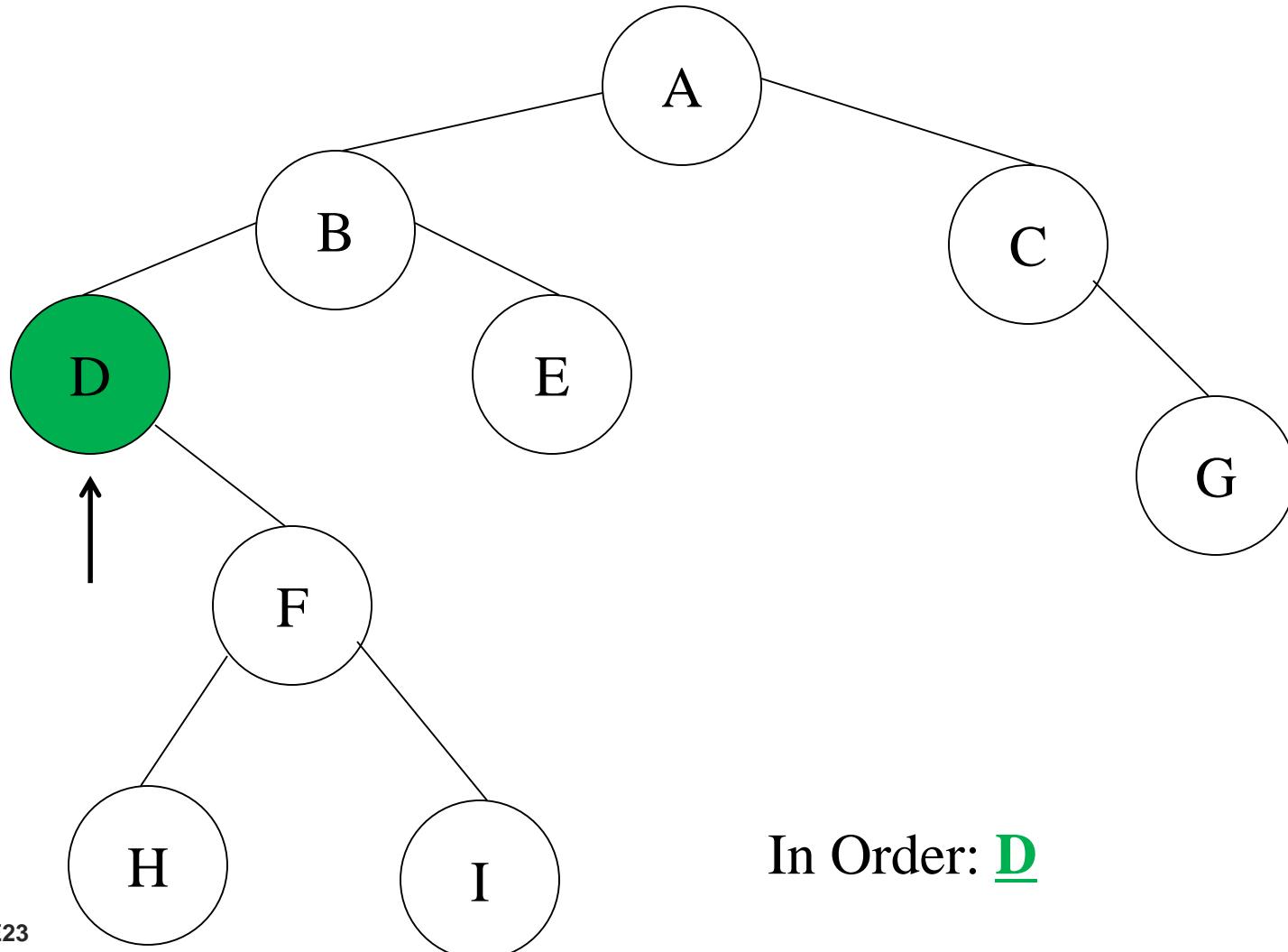
# Модны нэвтрэлтийн жишээ



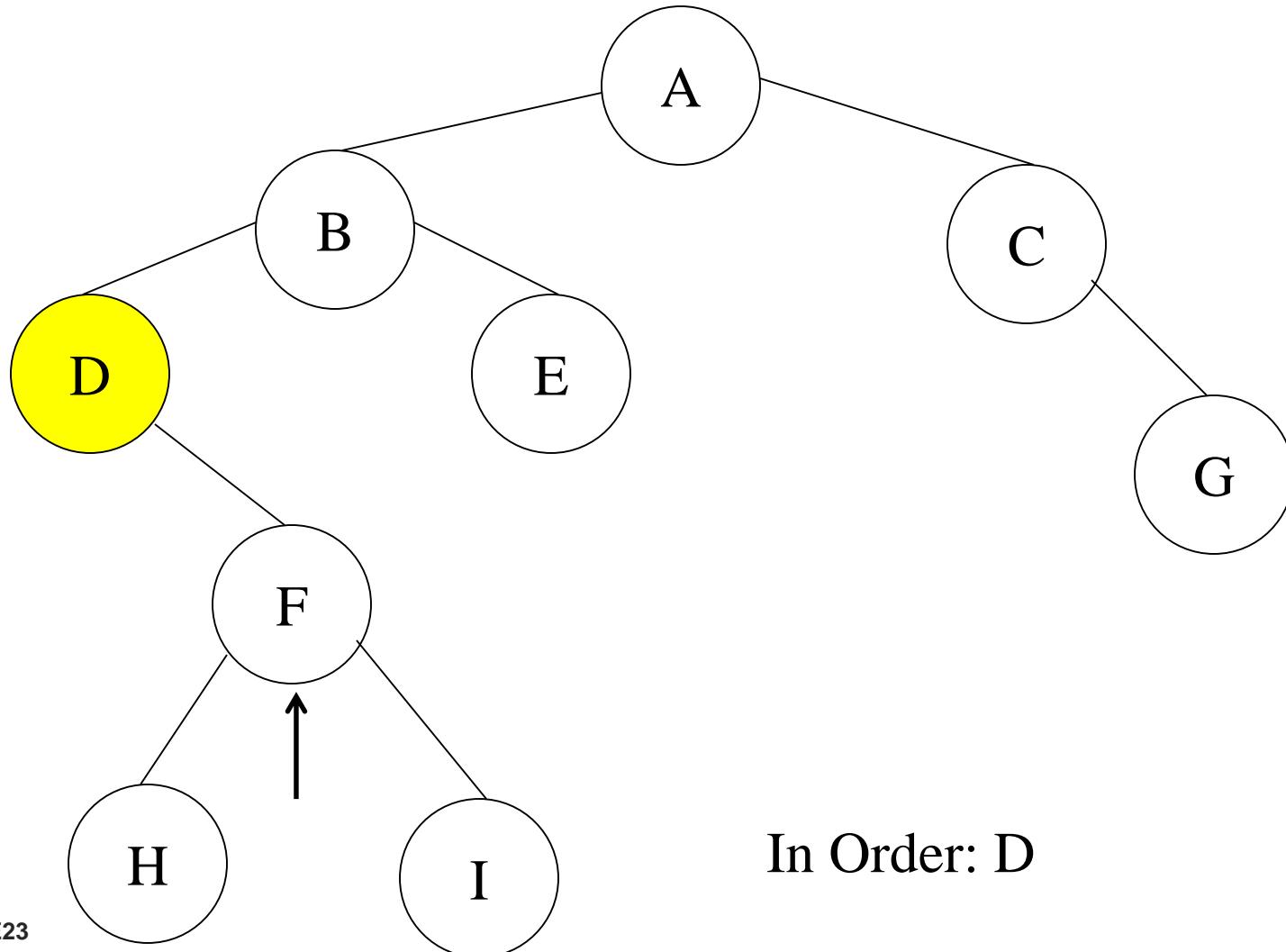
# Модны нэвтрэлтийн жишээ



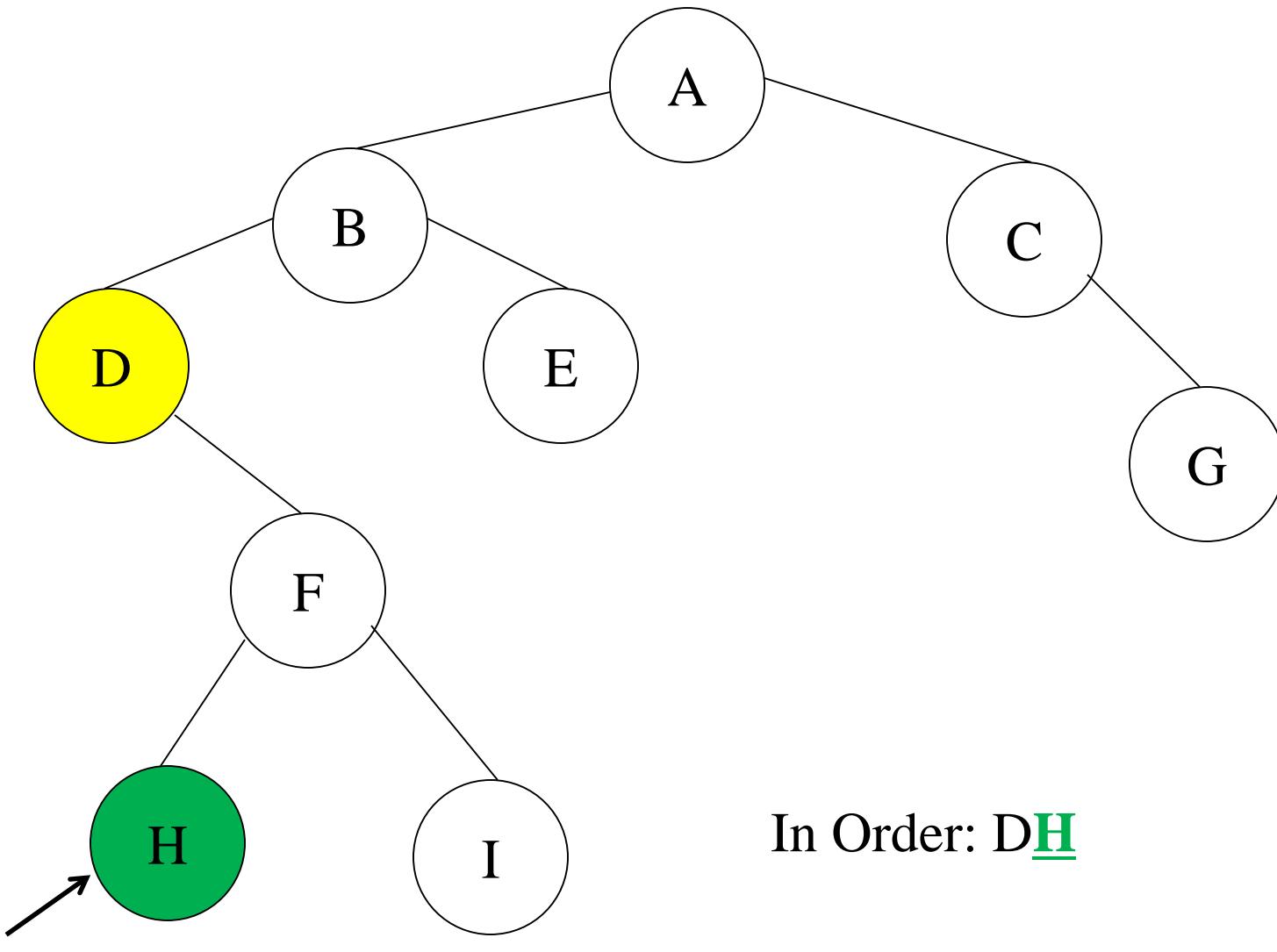
# Модны нэвтрэлтийн жишээ



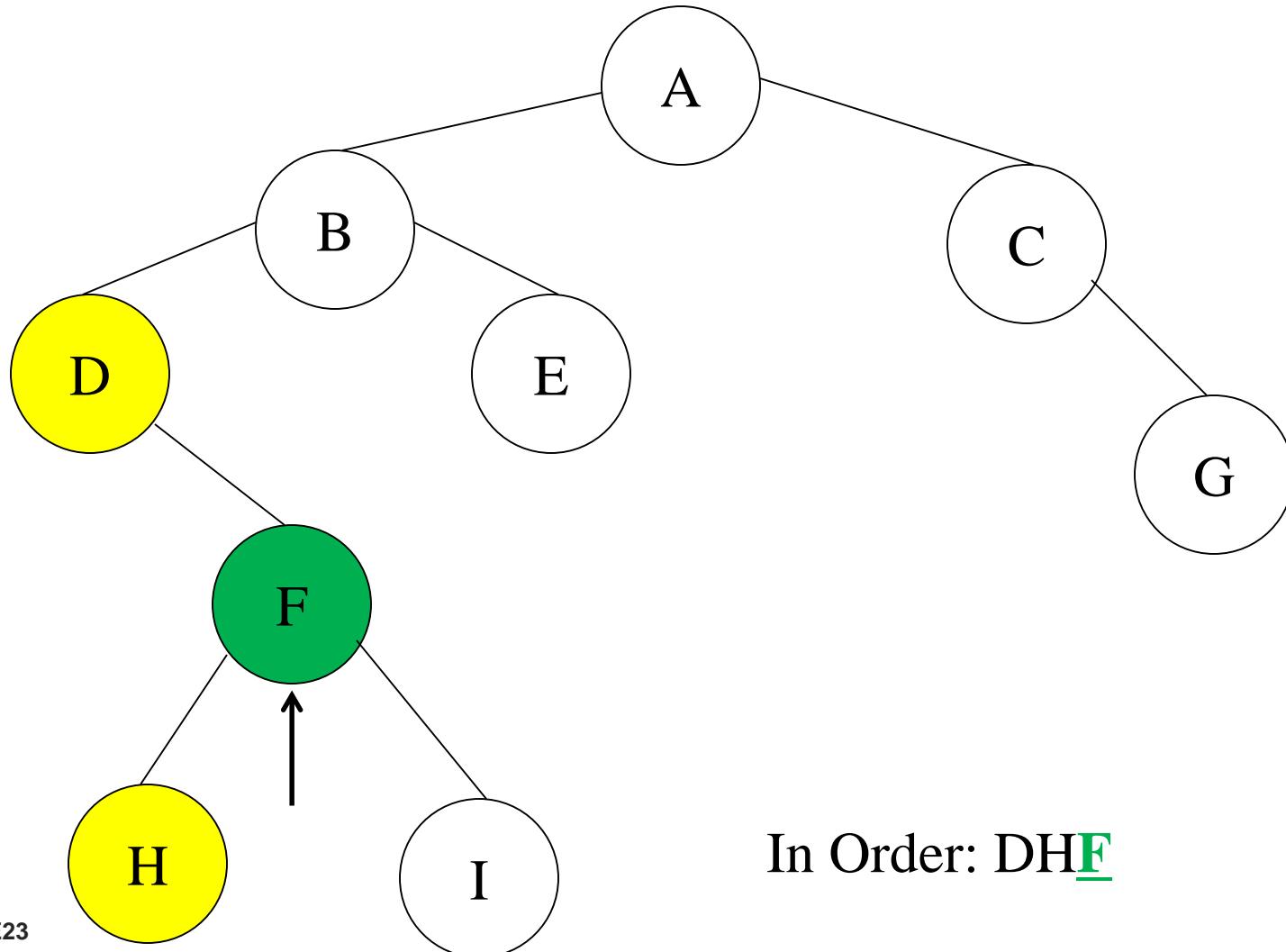
# Модны нэвтрэлтийн жишээ



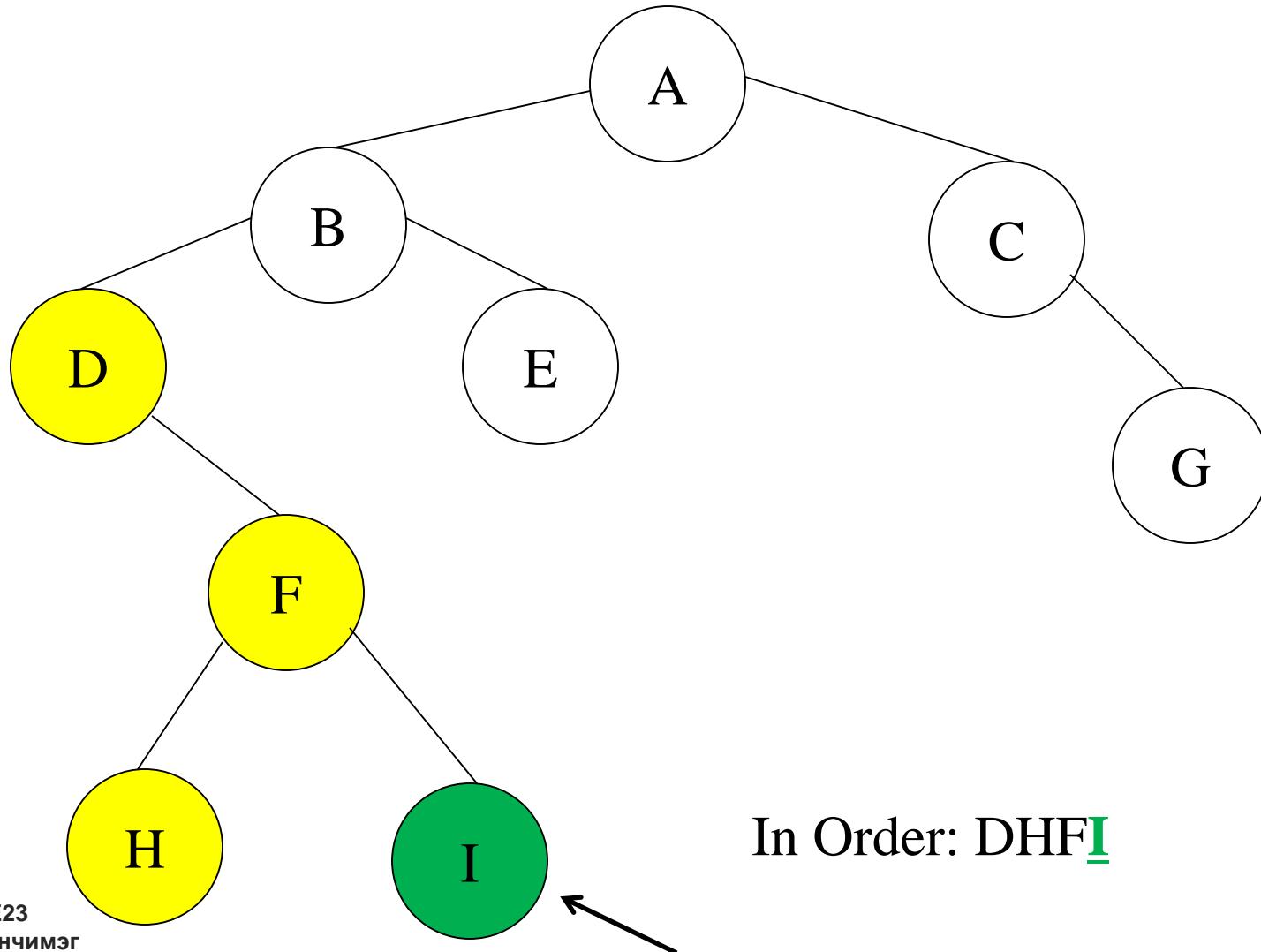
# Модны нэвтрэлтийн жишээ



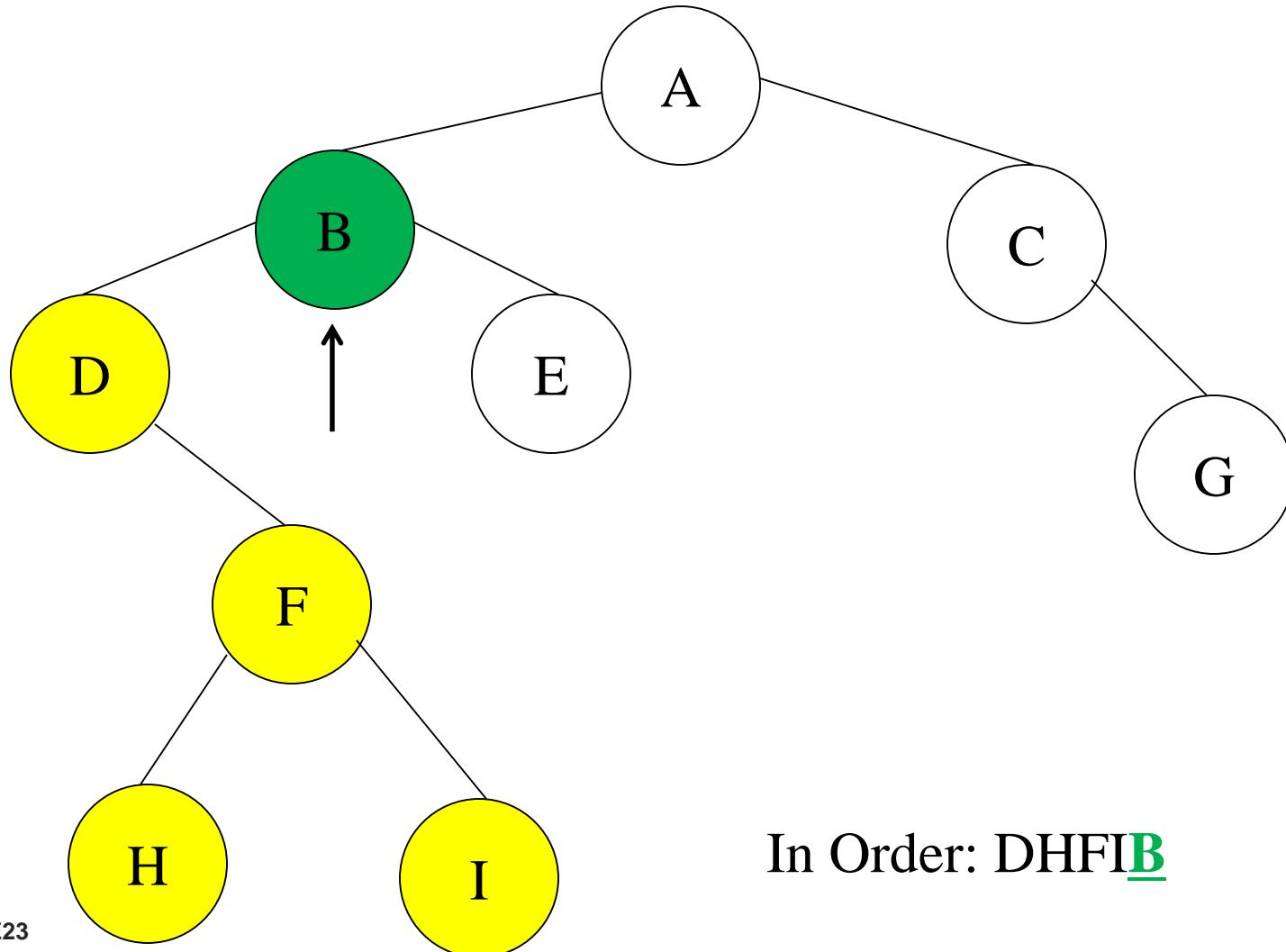
# Модны нэвтрэлтийн жишээ



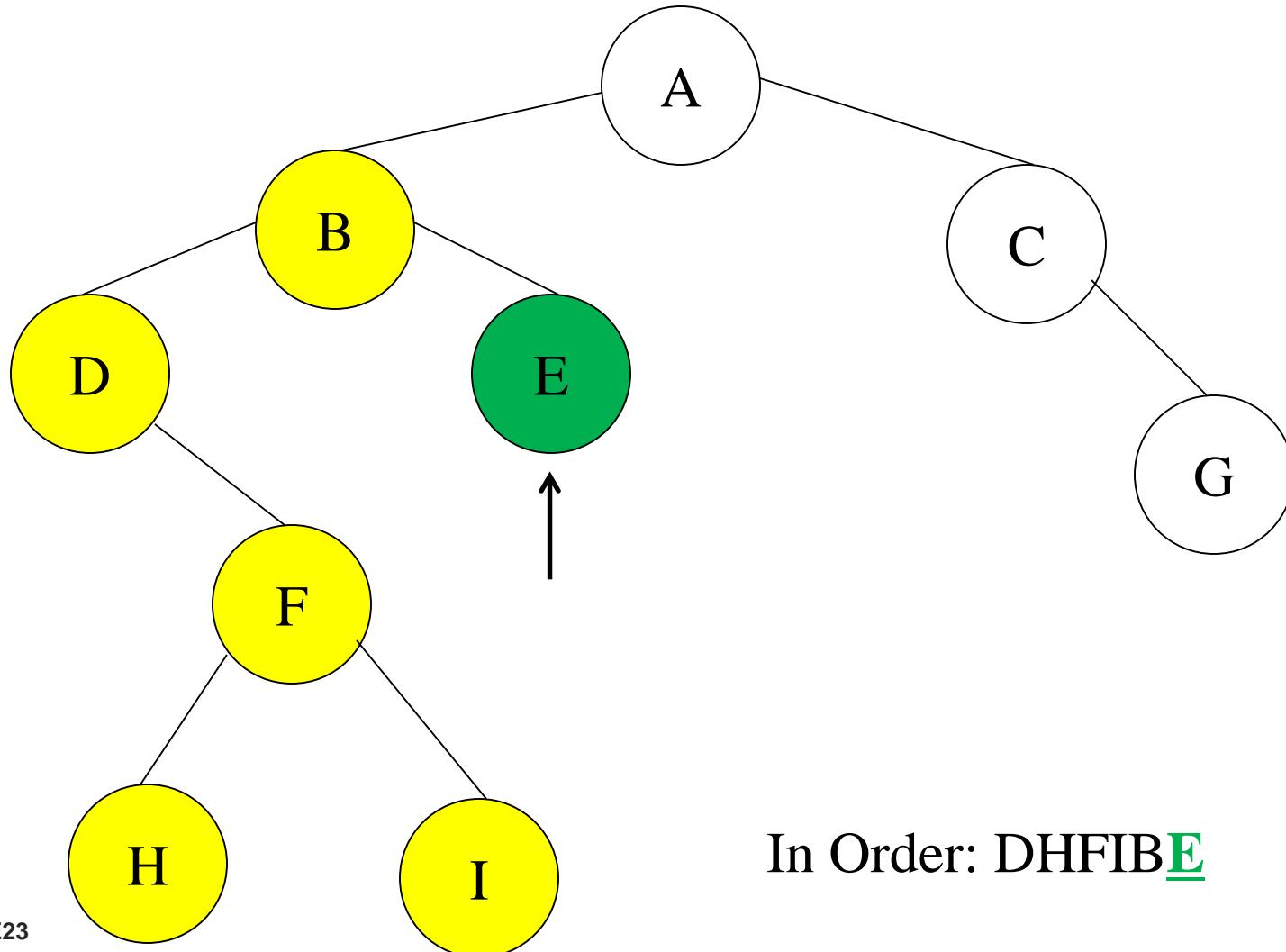
# Модны нэвтрэлтийн жишээ



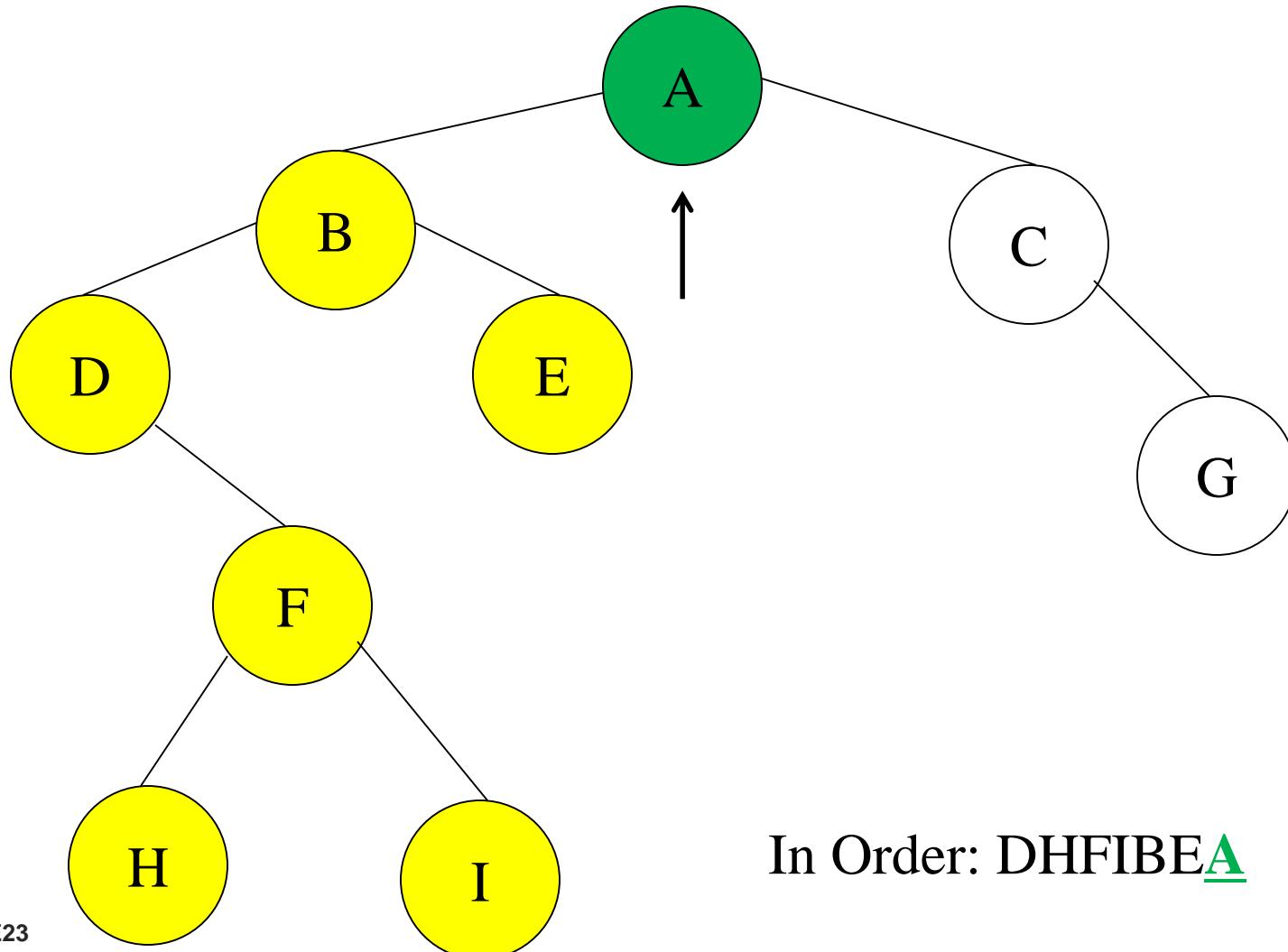
# Модны нэвтрэлтийн жишээ



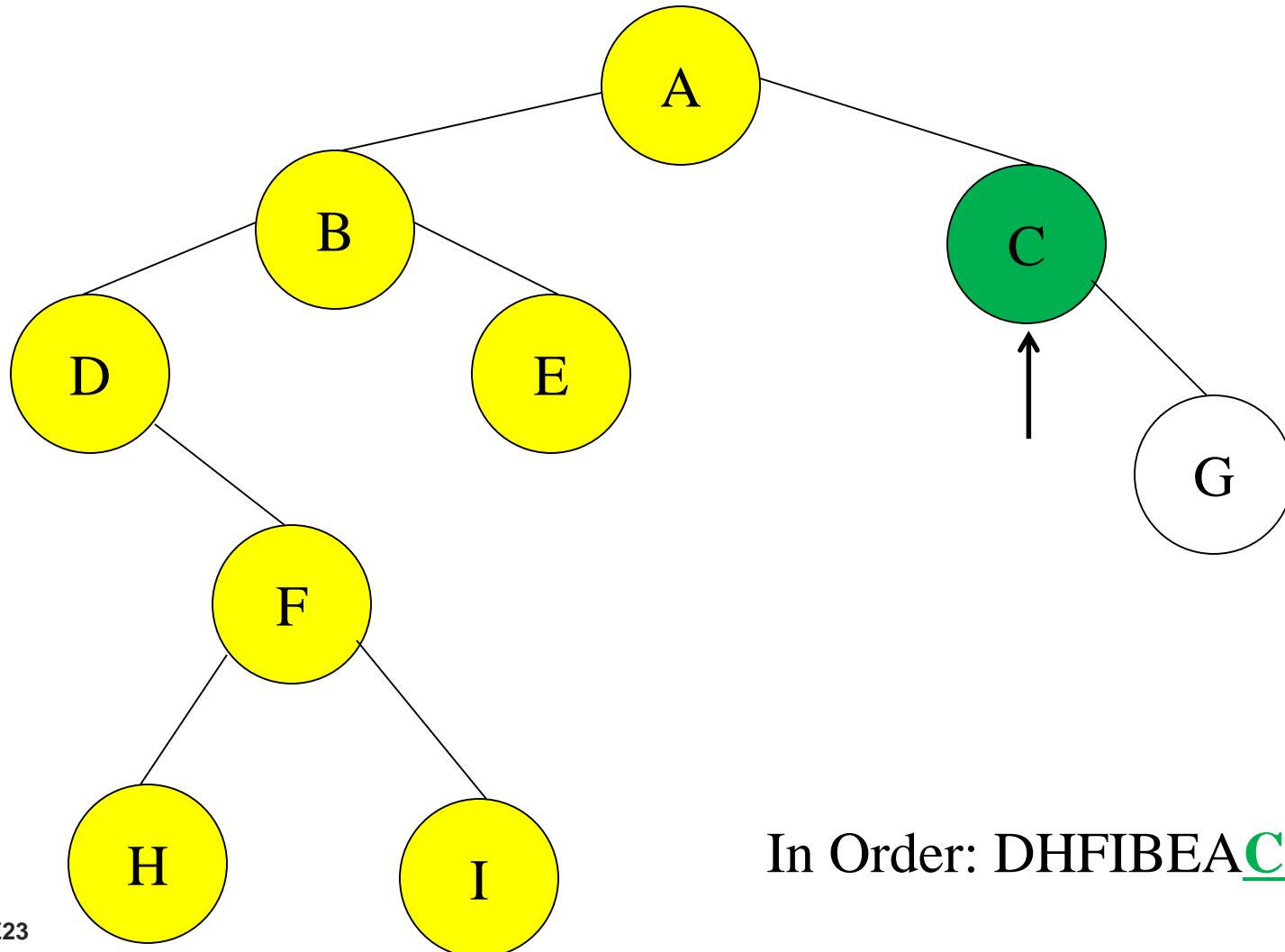
# Модны нэвтрэлтийн жишээ



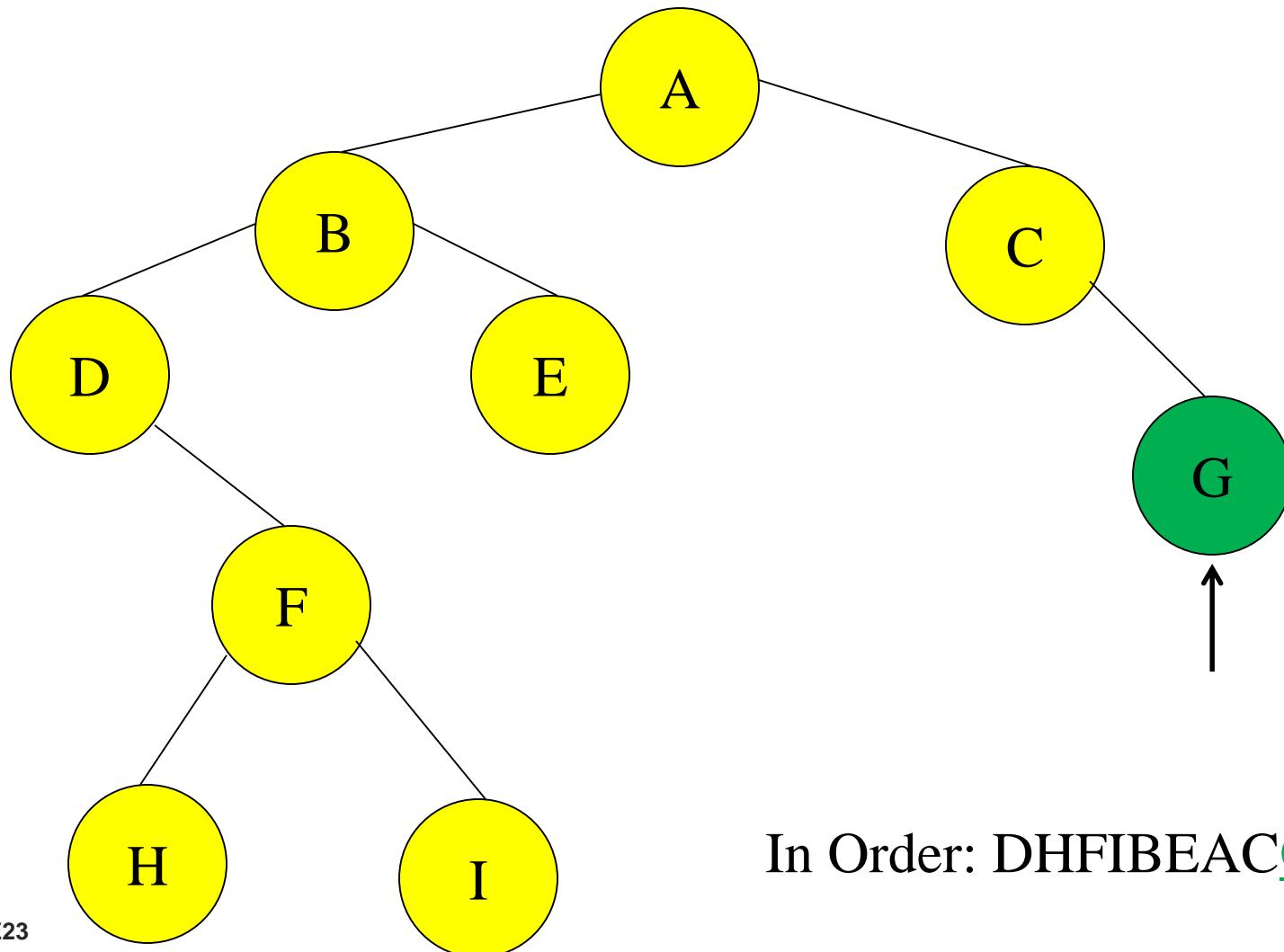
# Модны нэвтрэлтийн жишээ



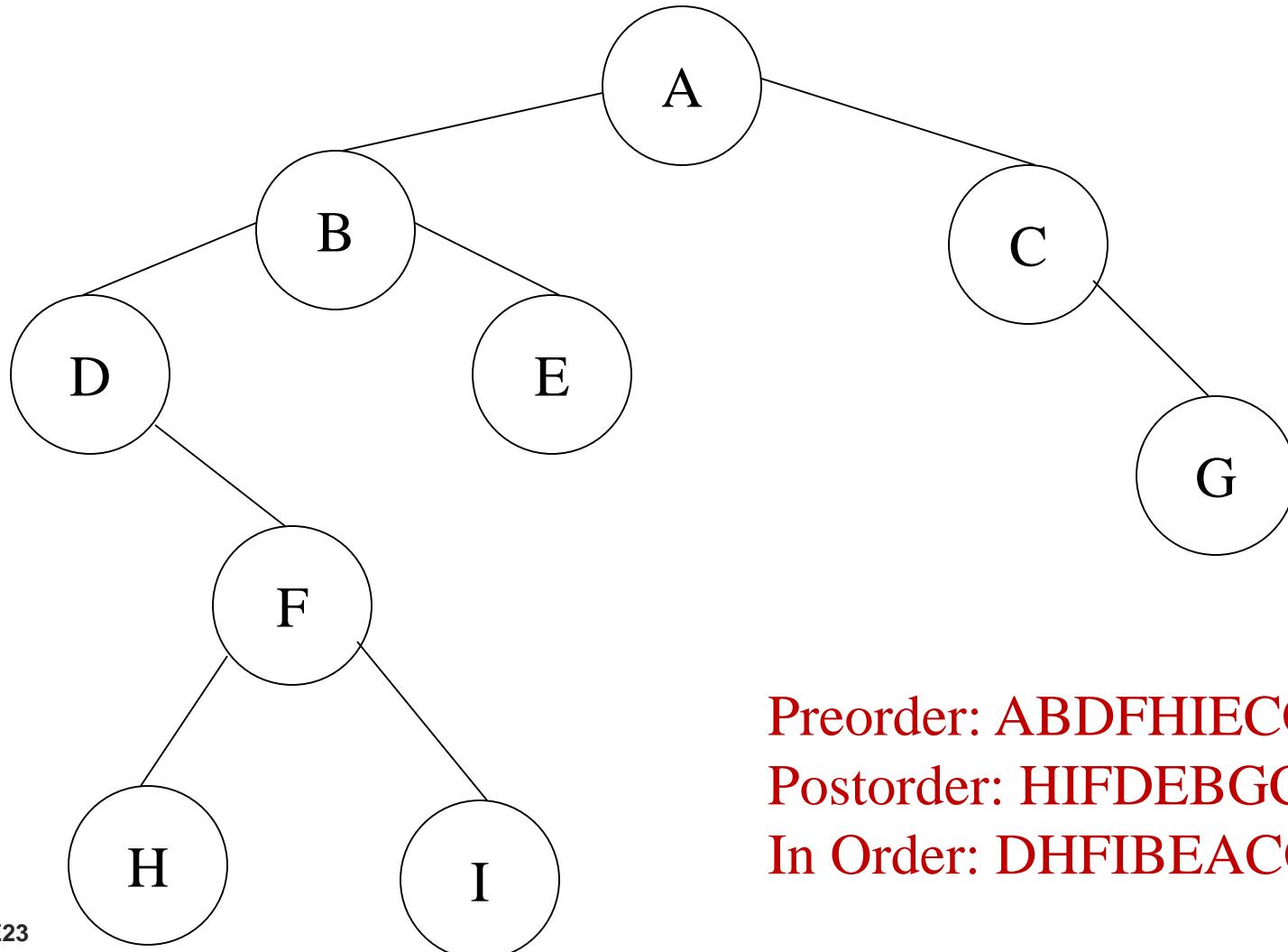
# Модны нэвтрэлтийн жишээ



# 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());        }        }
```

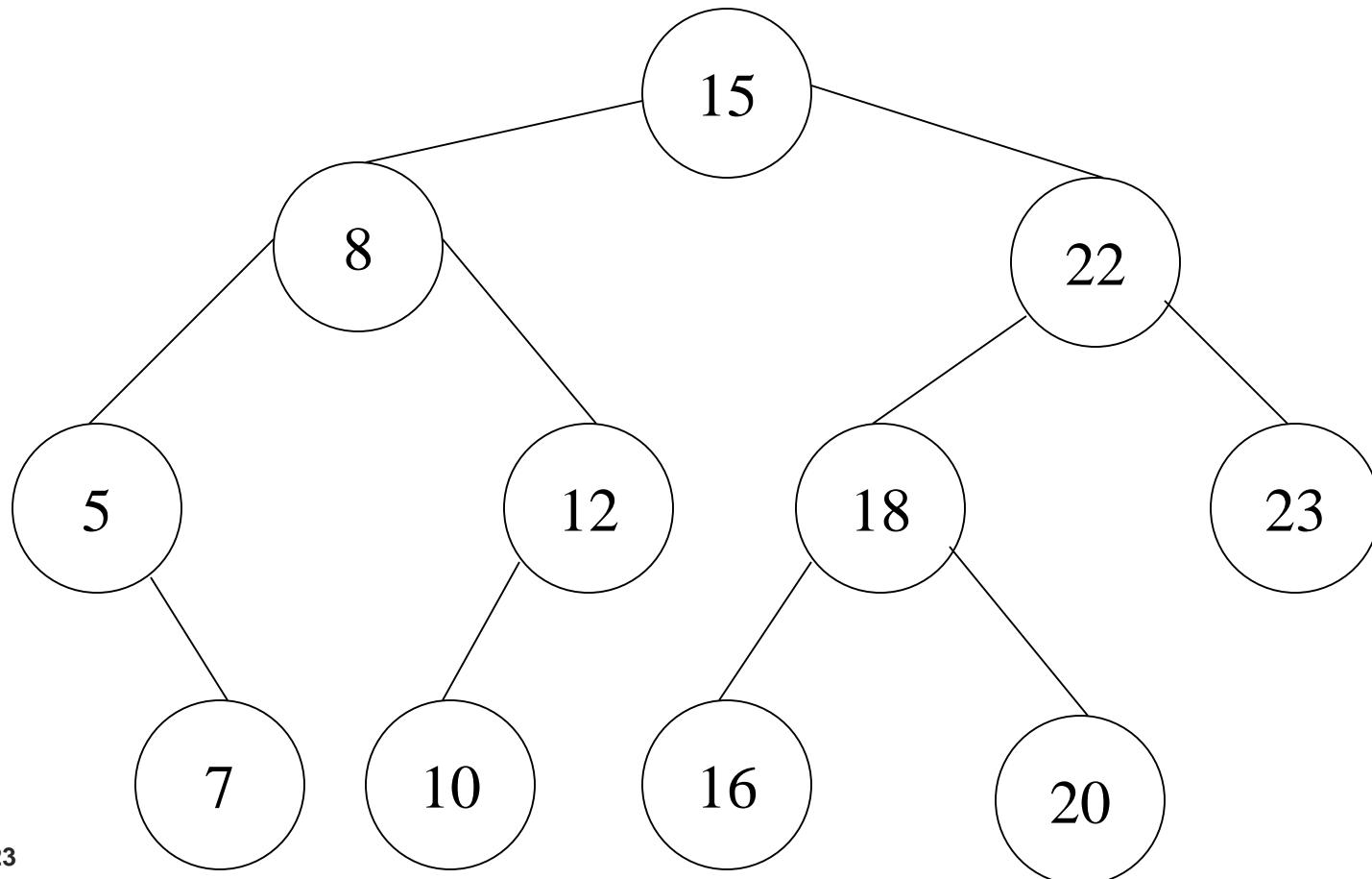
```
// addLeft() - Inserts a new node containing
//           x as the left child of p
public void addLeft(Node p, int x) {
    Node q = newNode(x);
    p.setLeft(q);
}

// 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;

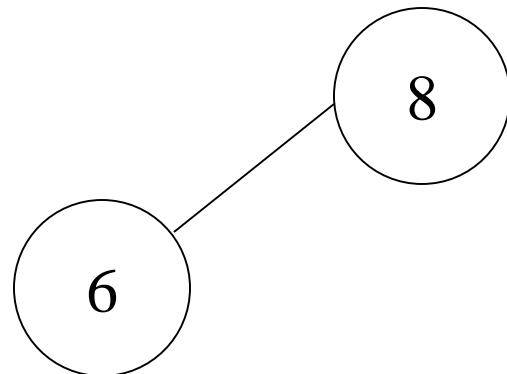
    p = root;
    q = null;
    while (p != null && !found) {
        q = p;
        if (p.getData() == x)
            found = true;
        else if (p.getData() > x)
            p = p.getLeft();
        else
            p = p.getRight();
    }
    if (found)
        return(q);
    else
        return(null);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

```
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();  
    }  
}
```

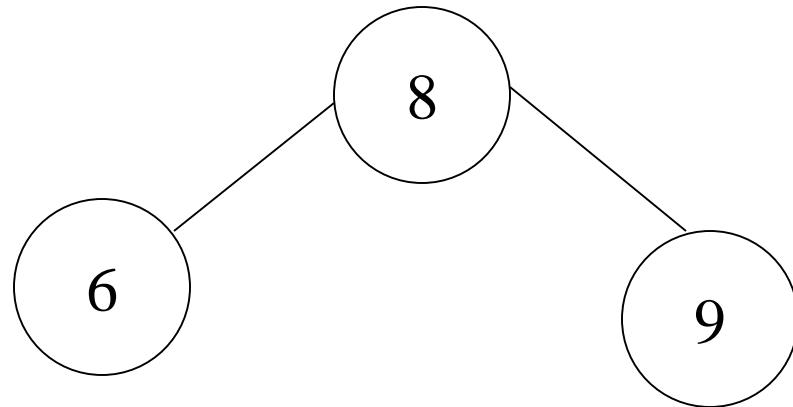
# Туршилтын мод

8

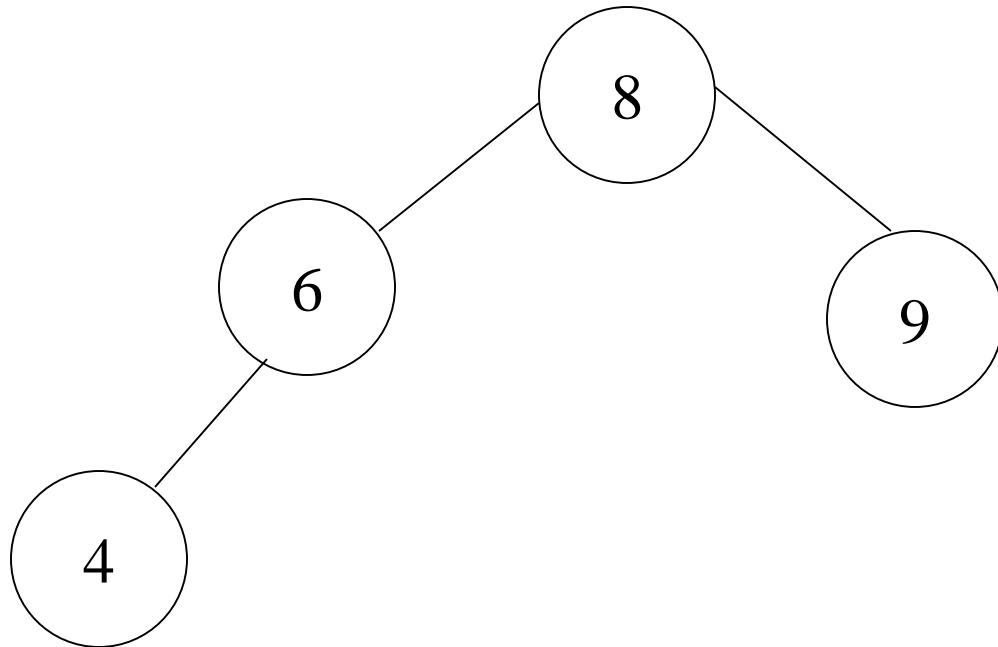
# Туршилтын мод



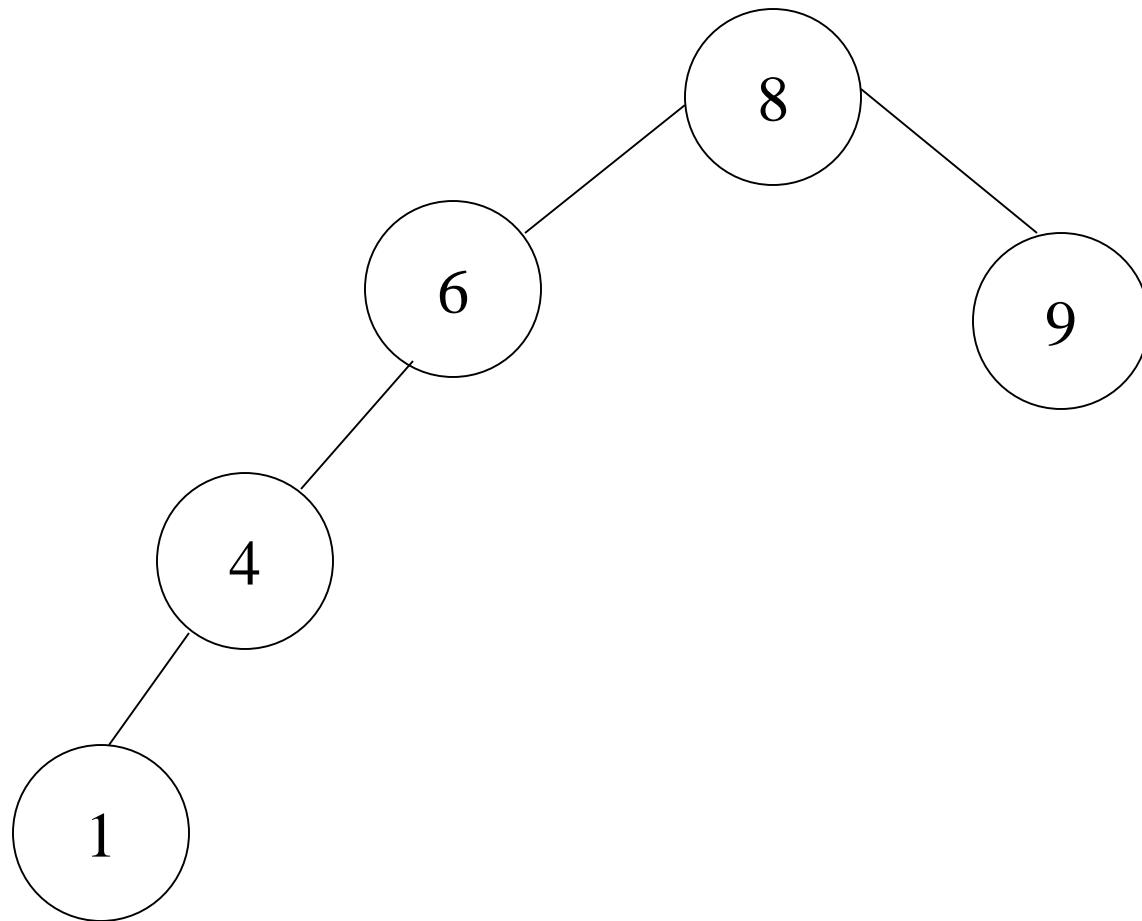
# Туршилтын мод



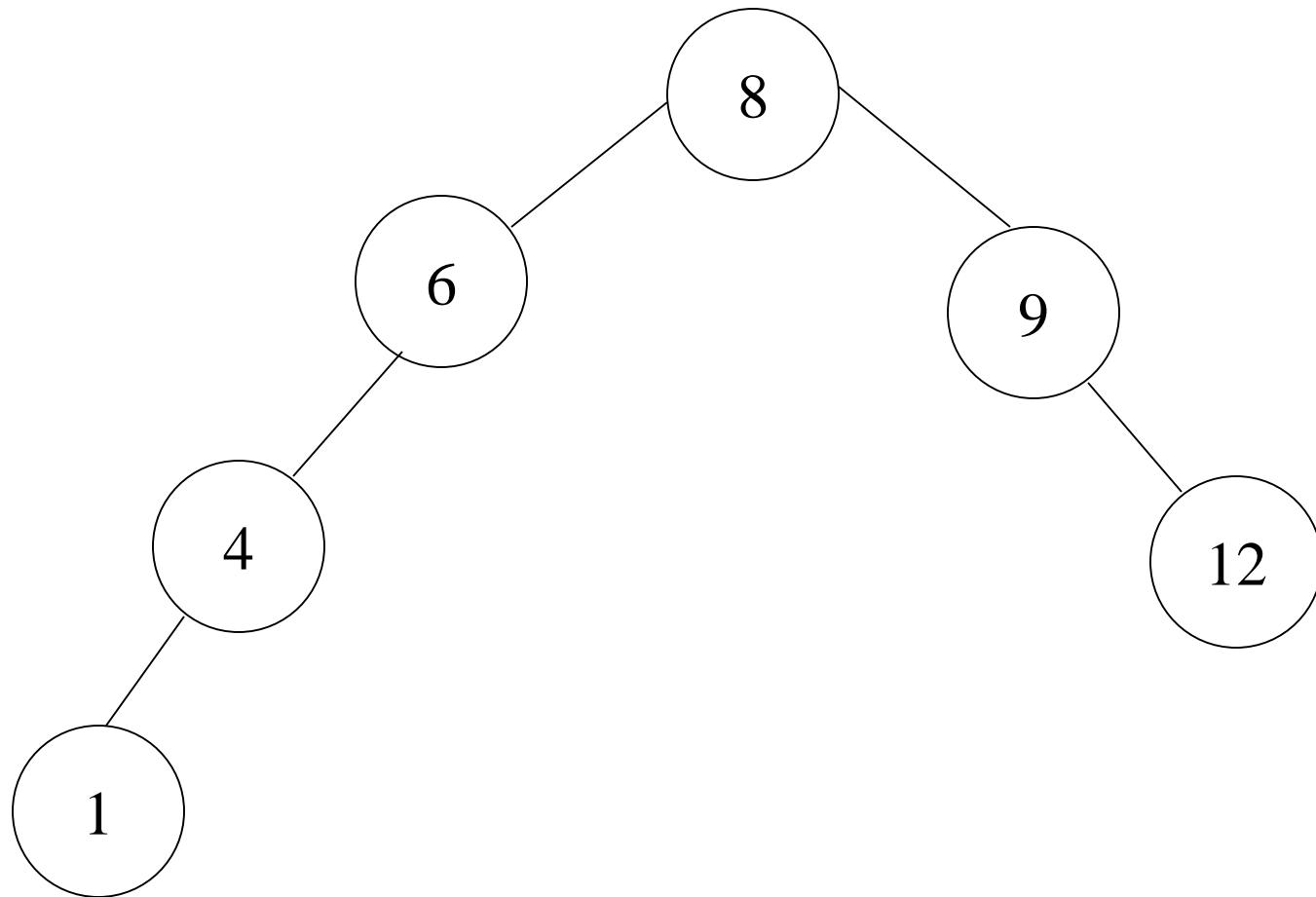
# Туршилтын мод



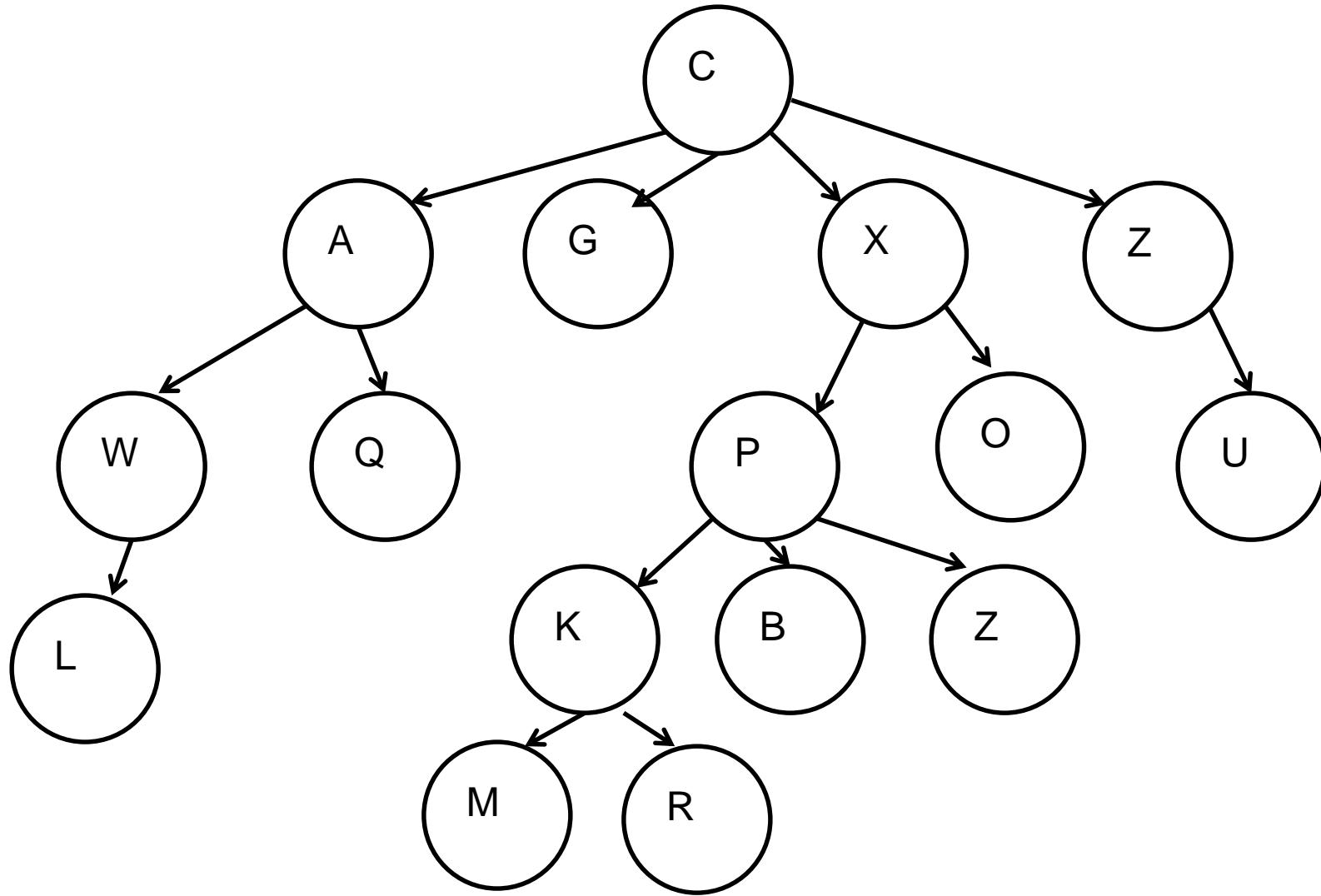
# Туршилтын мод



# Туршилтын мод



# Модны өргөнөөр хайх

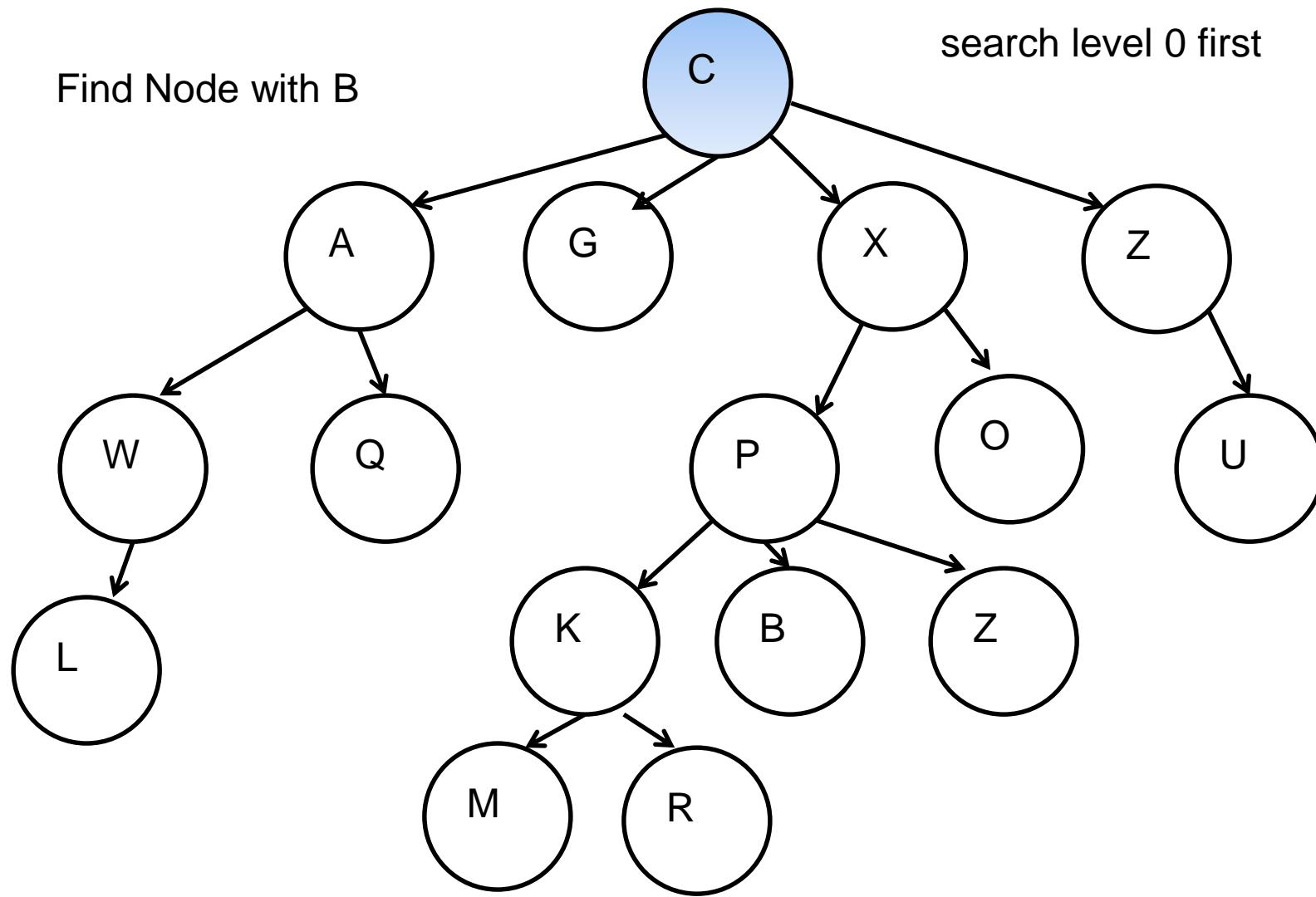


- Модны түвшний эрэмбэ дарааллыг эхний хайлт болгон ашиглана.
- Дараагийн түвшинд орохосоо өмнө бүх зангилааг нэг түвшинд хайна.

# Breadth First Search / Өргөнөөр хайх

Find Node with B

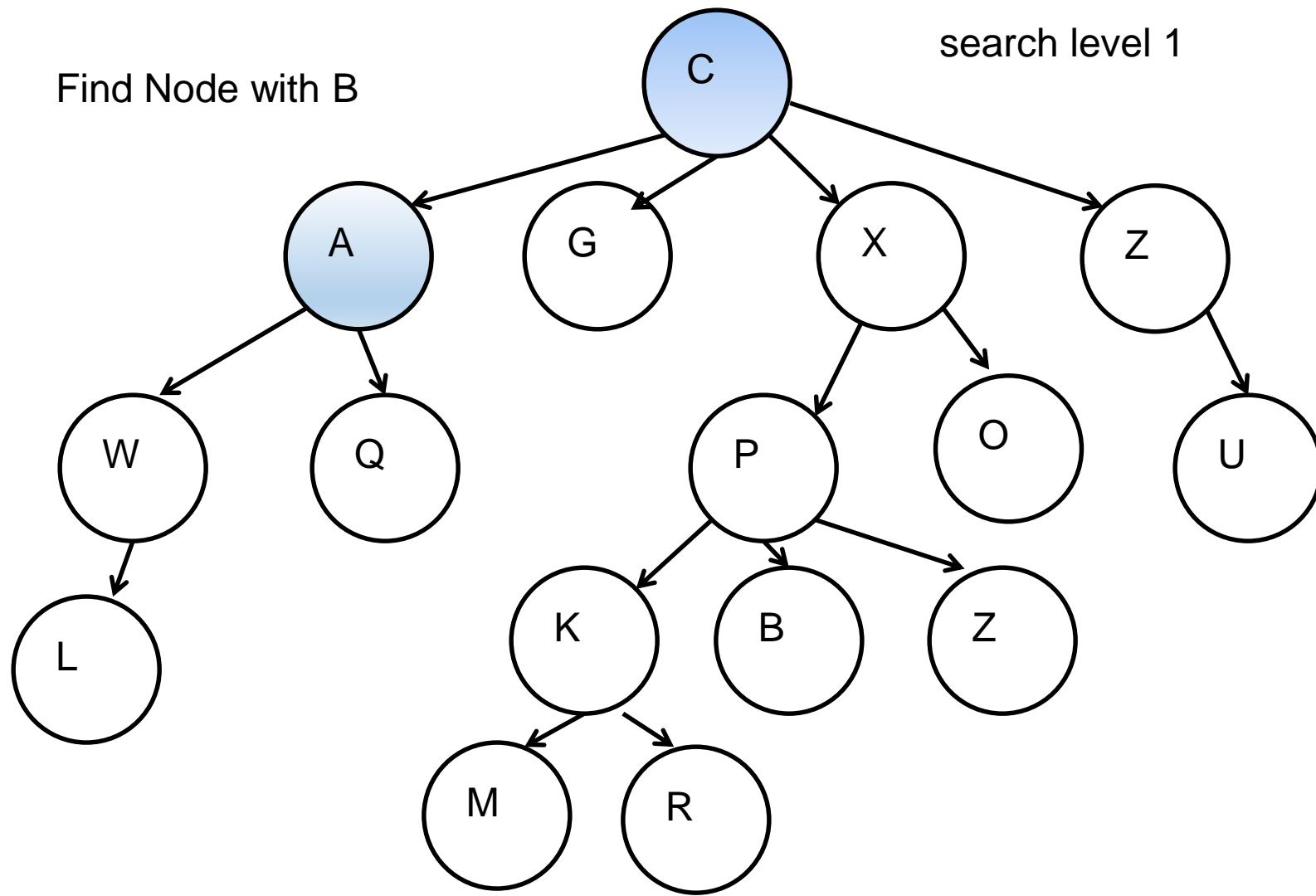
search level 0 first



# Breadth First Search / Өргөнөөр хайх

Find Node with B

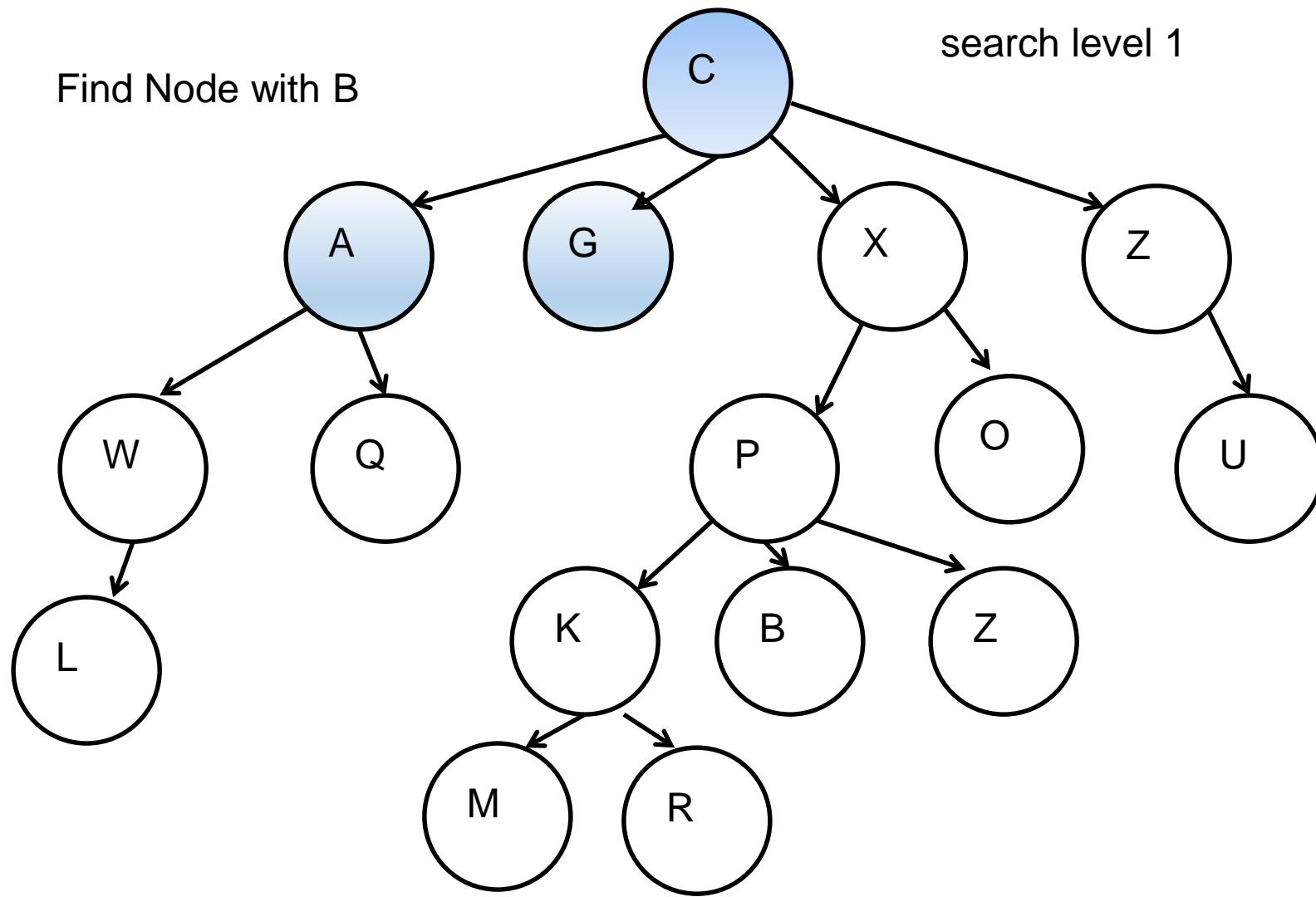
search level 1



# Breadth First Search / Өргөнөөр хайх

Find Node with B

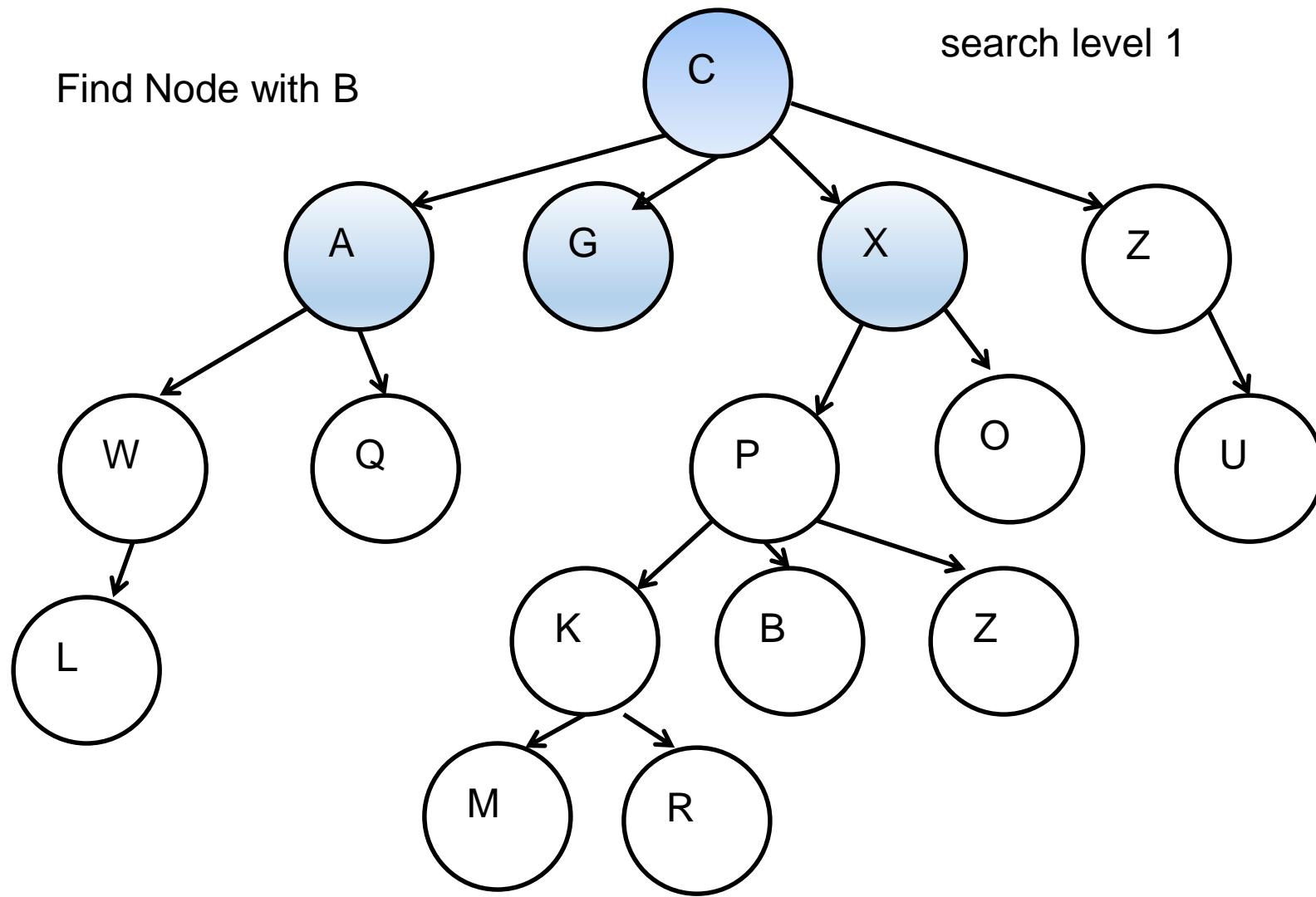
search level 1



# Breadth First Search / Өргөнөөр хайх

Find Node with B

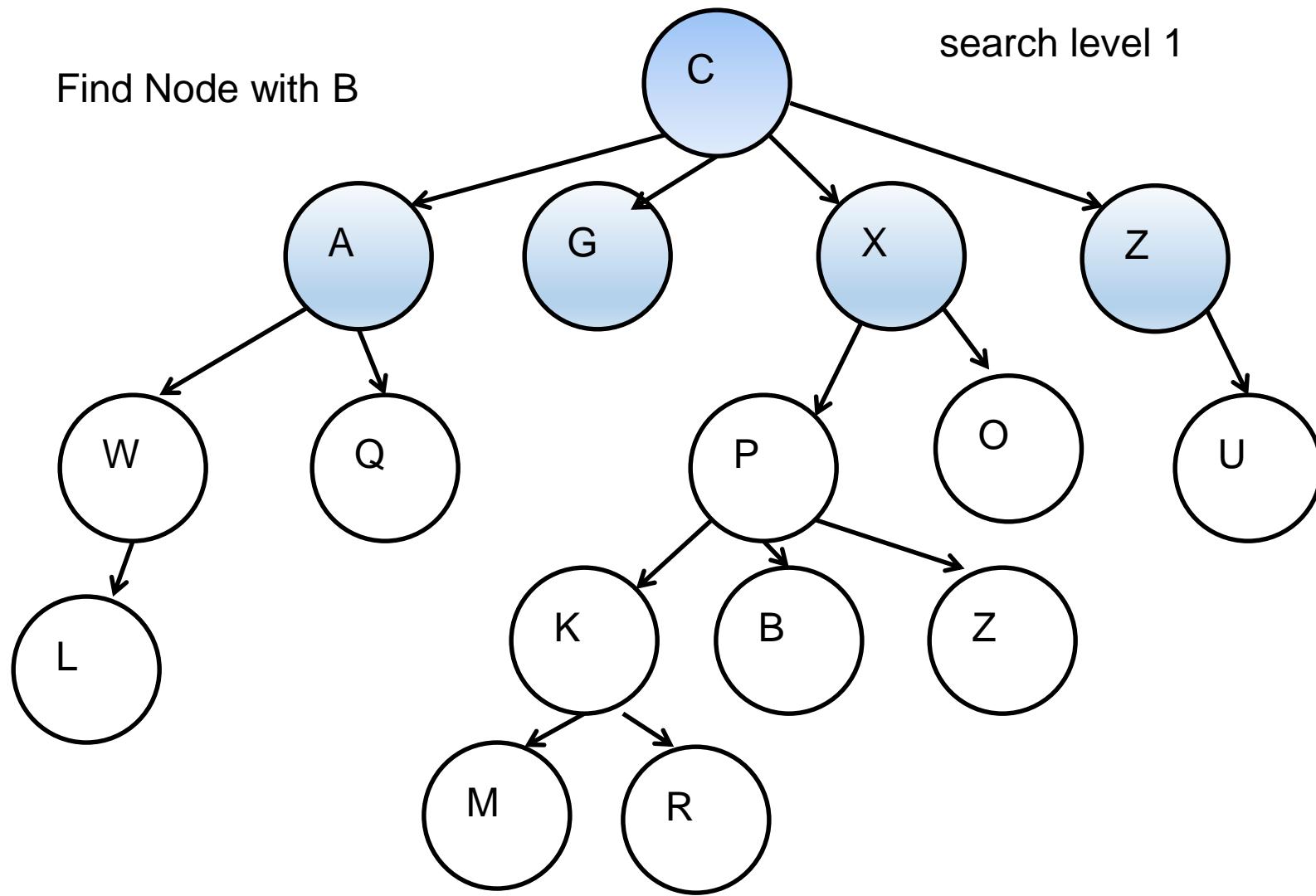
search level 1



# Breadth First Search / Өргөнөөр хайх

Find Node with B

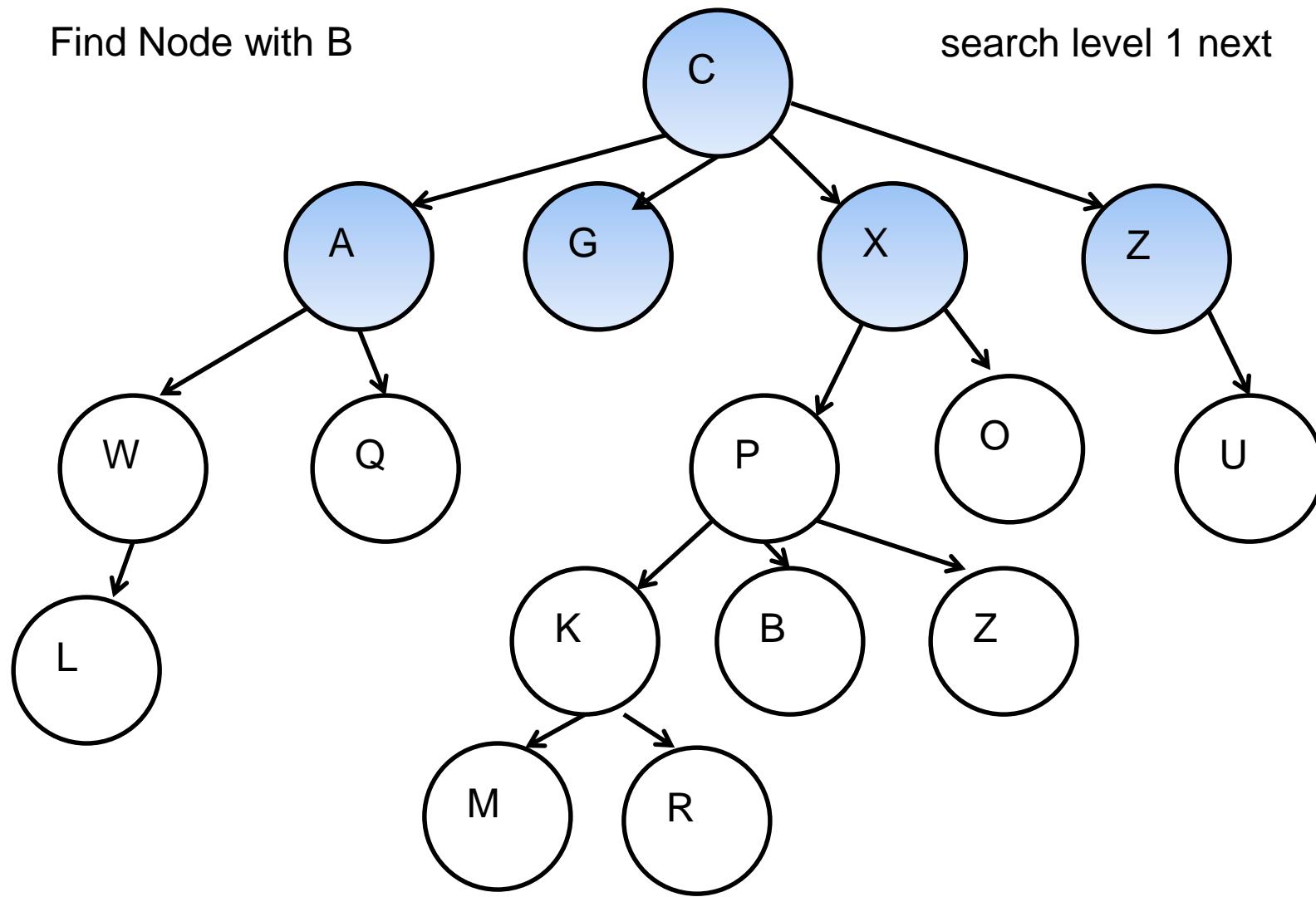
search level 1



# Breadth First Search / Өргөнөөр хайх

Find Node with B

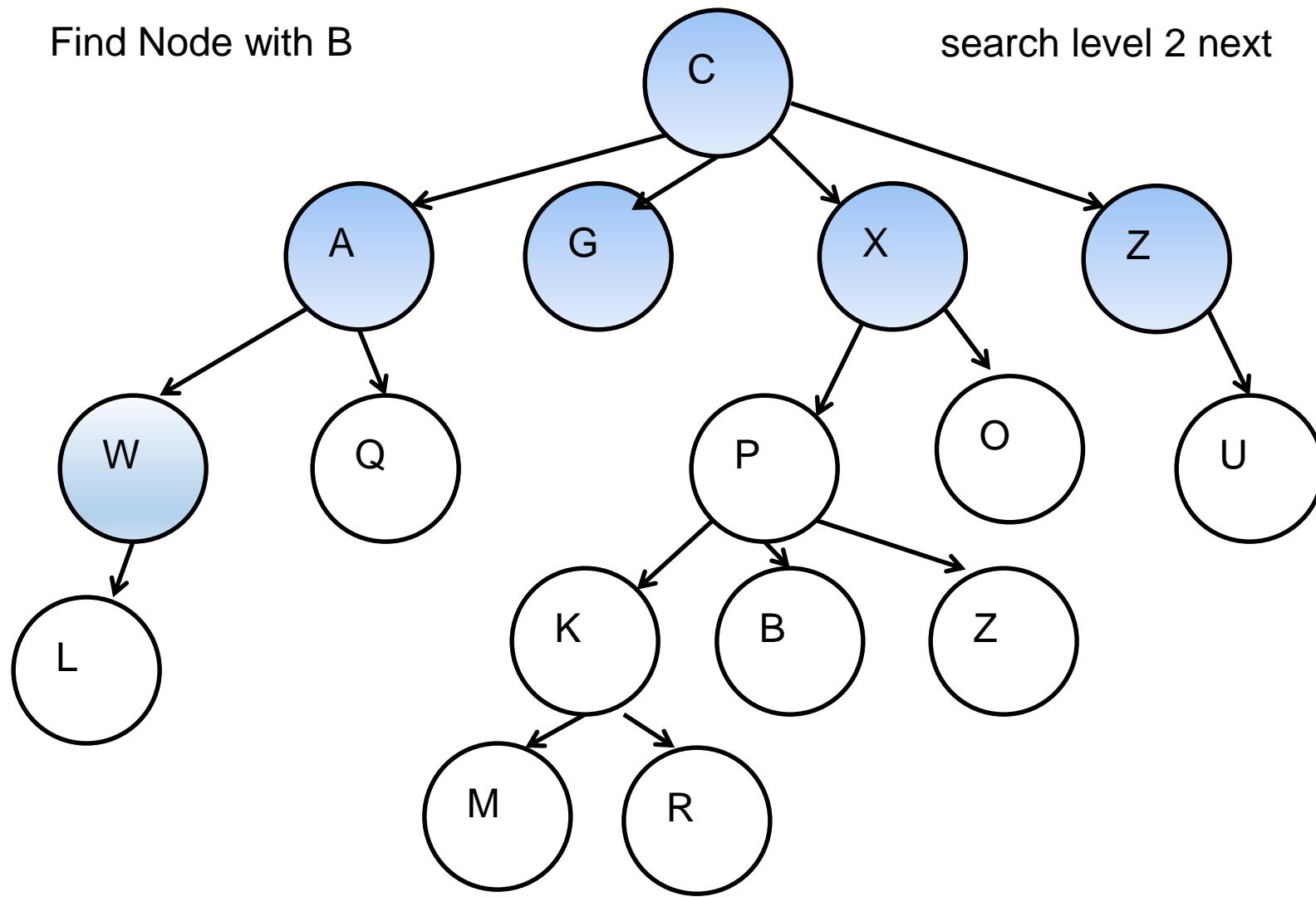
search level 1 next



# Breadth First Search / Өргөнөөр хайх

Find Node with B

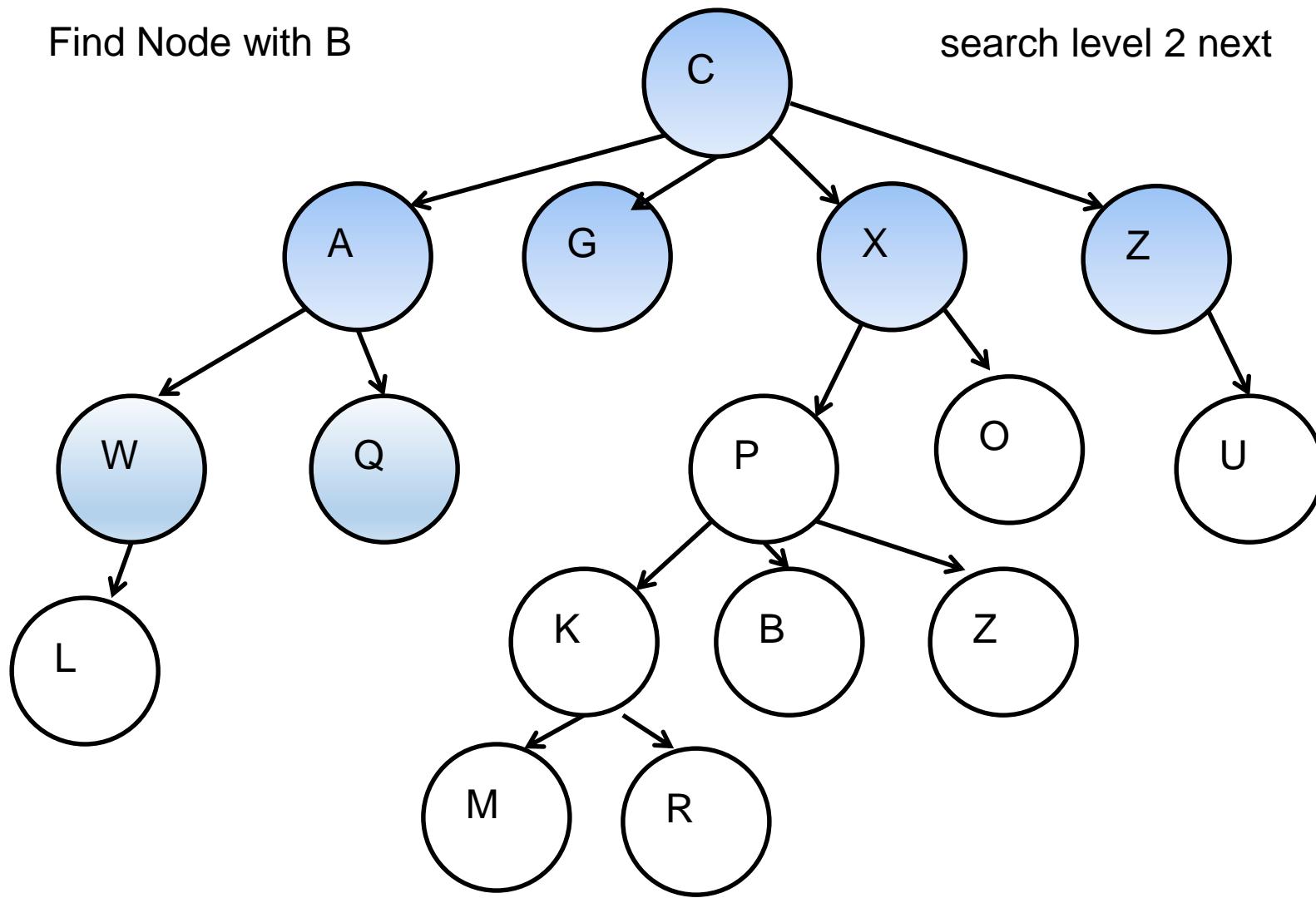
search level 2 next



# Breadth First Search / Өргөнөөр хайх

Find Node with B

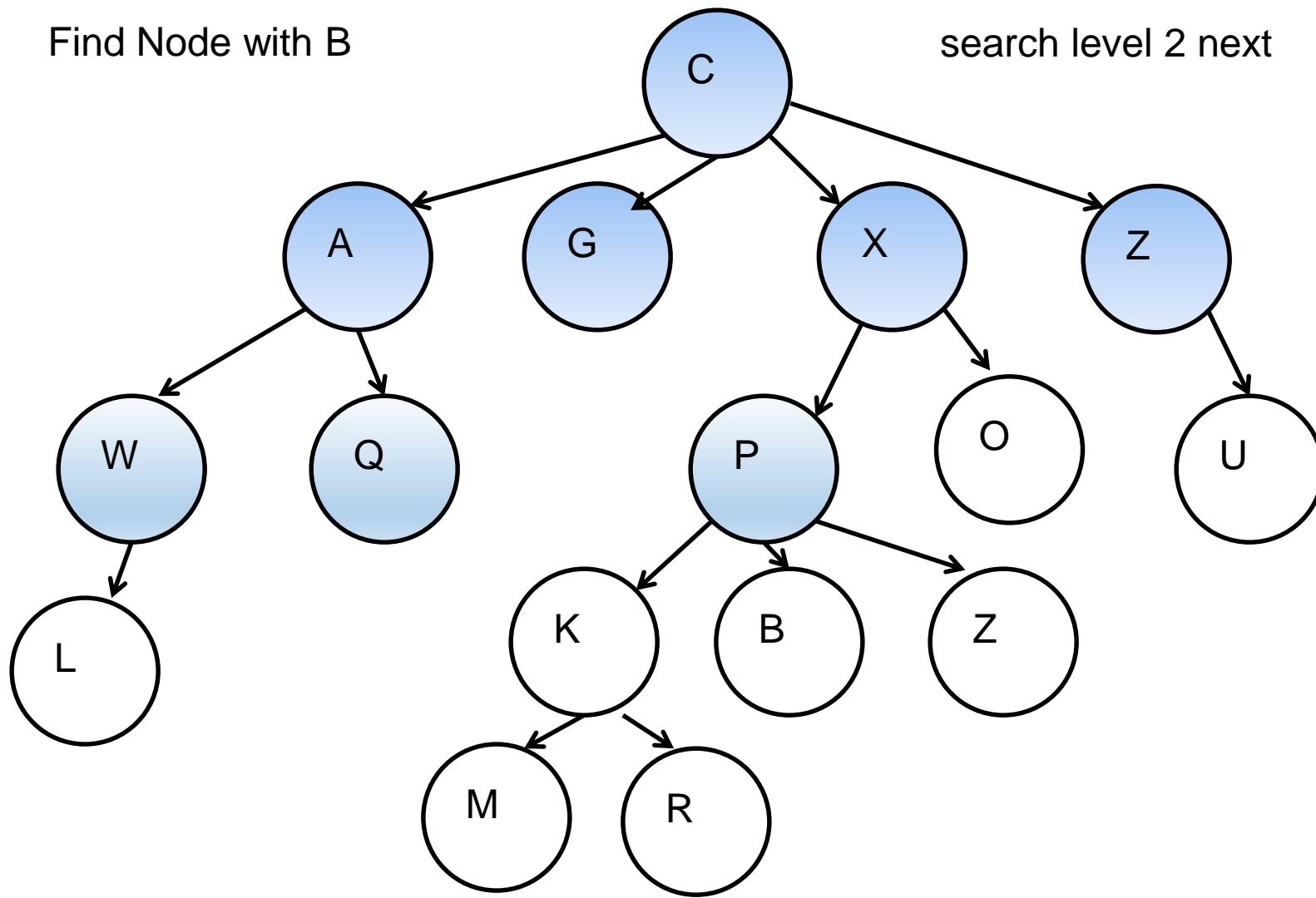
search level 2 next



# Breadth First Search / Өргөнөөр хайх

Find Node with B

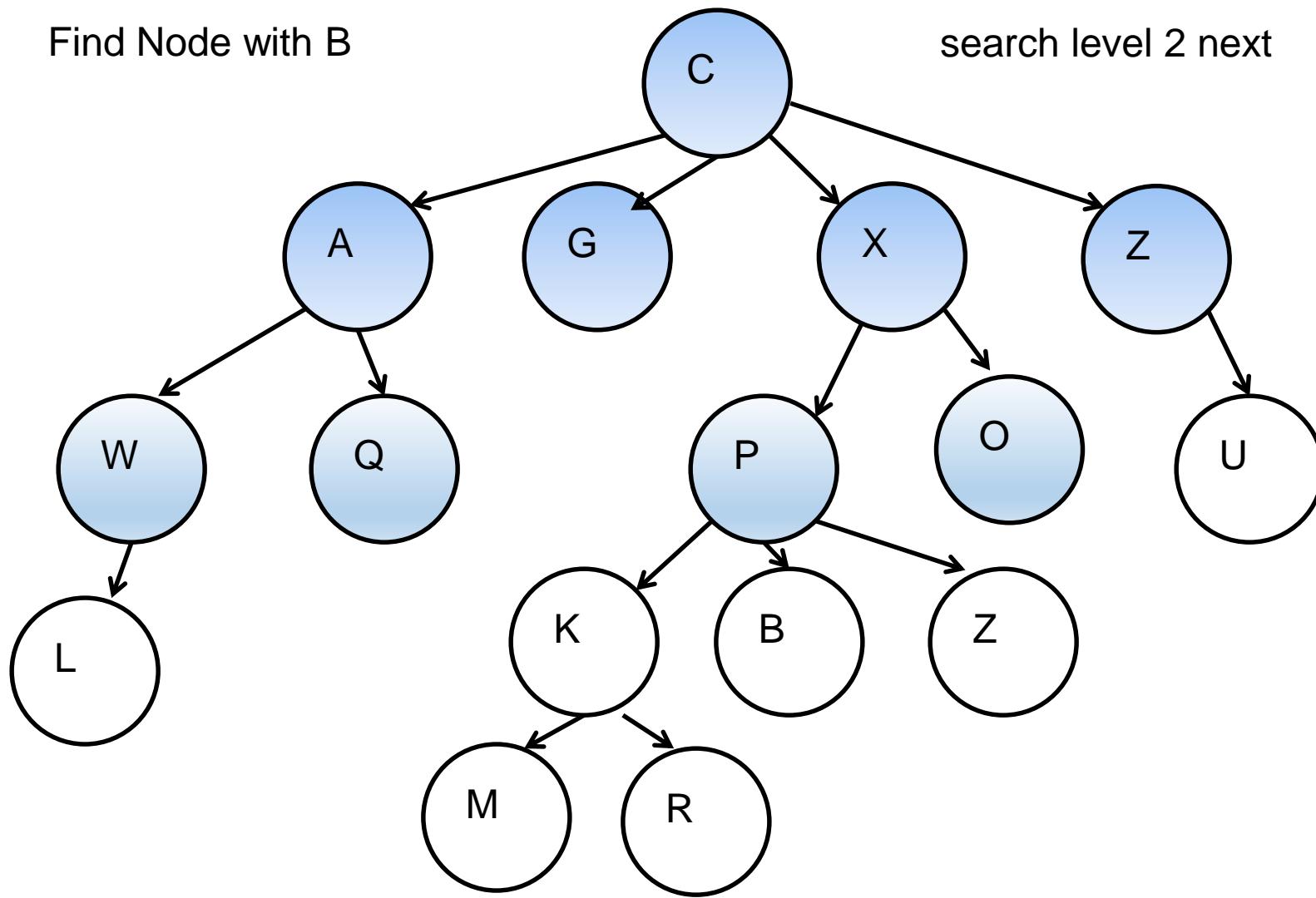
search level 2 next



# Breadth First Search / Өргөнөөр хайх

Find Node with B

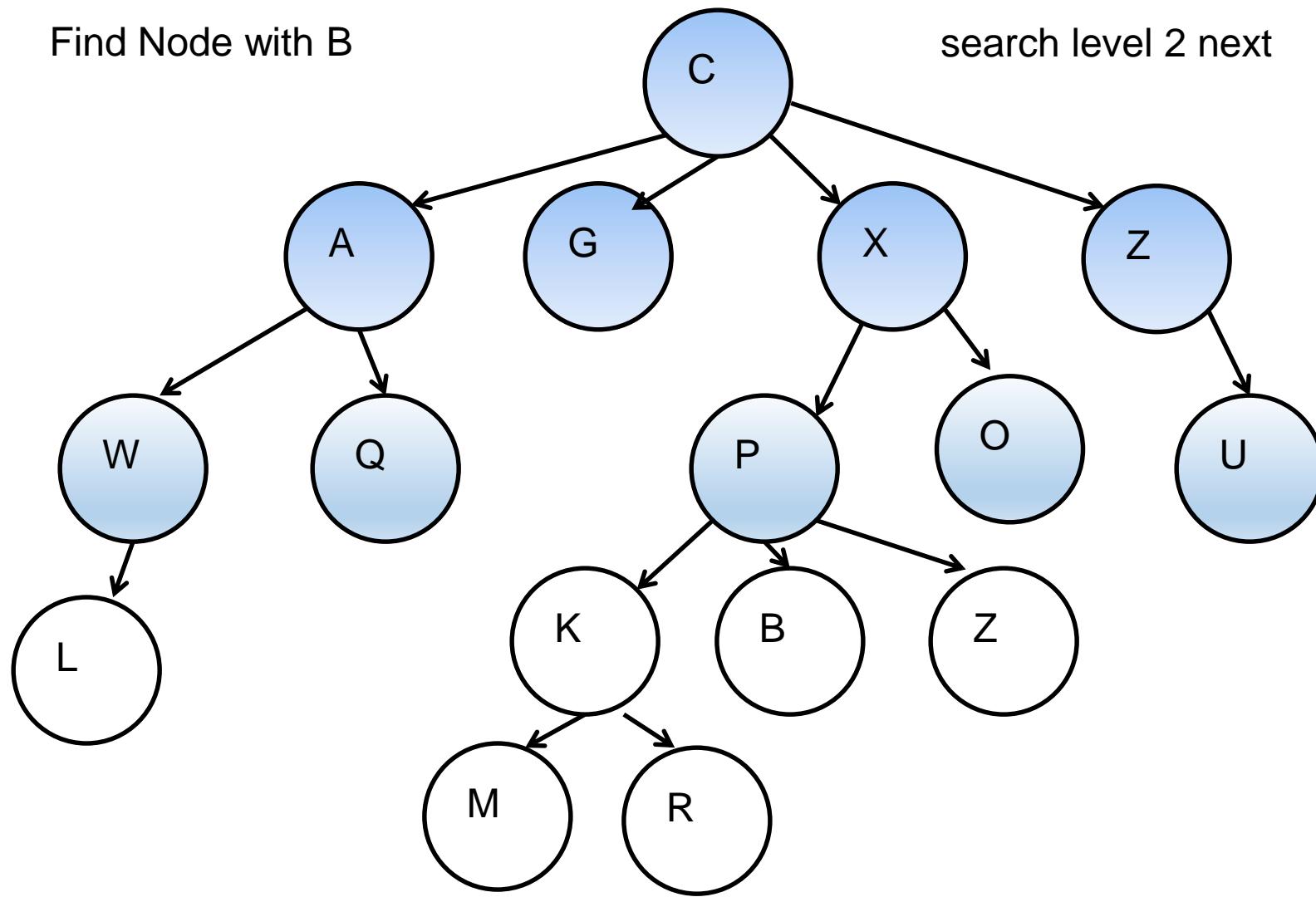
search level 2 next



# Breadth First Search / Өргөнөөр хайх

Find Node with B

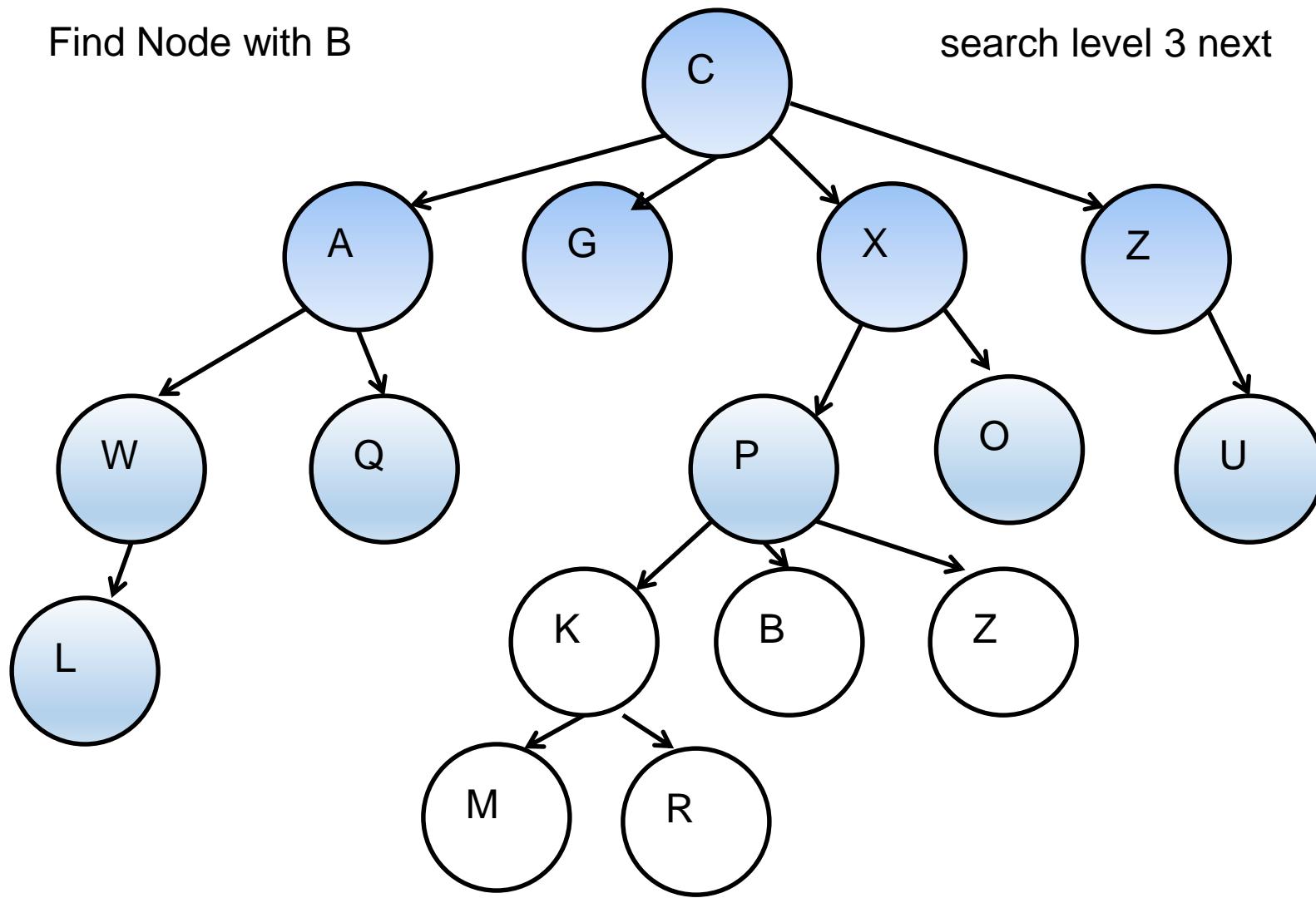
search level 2 next



# Breadth First Search / Өргөнөөр хайх

Find Node with B

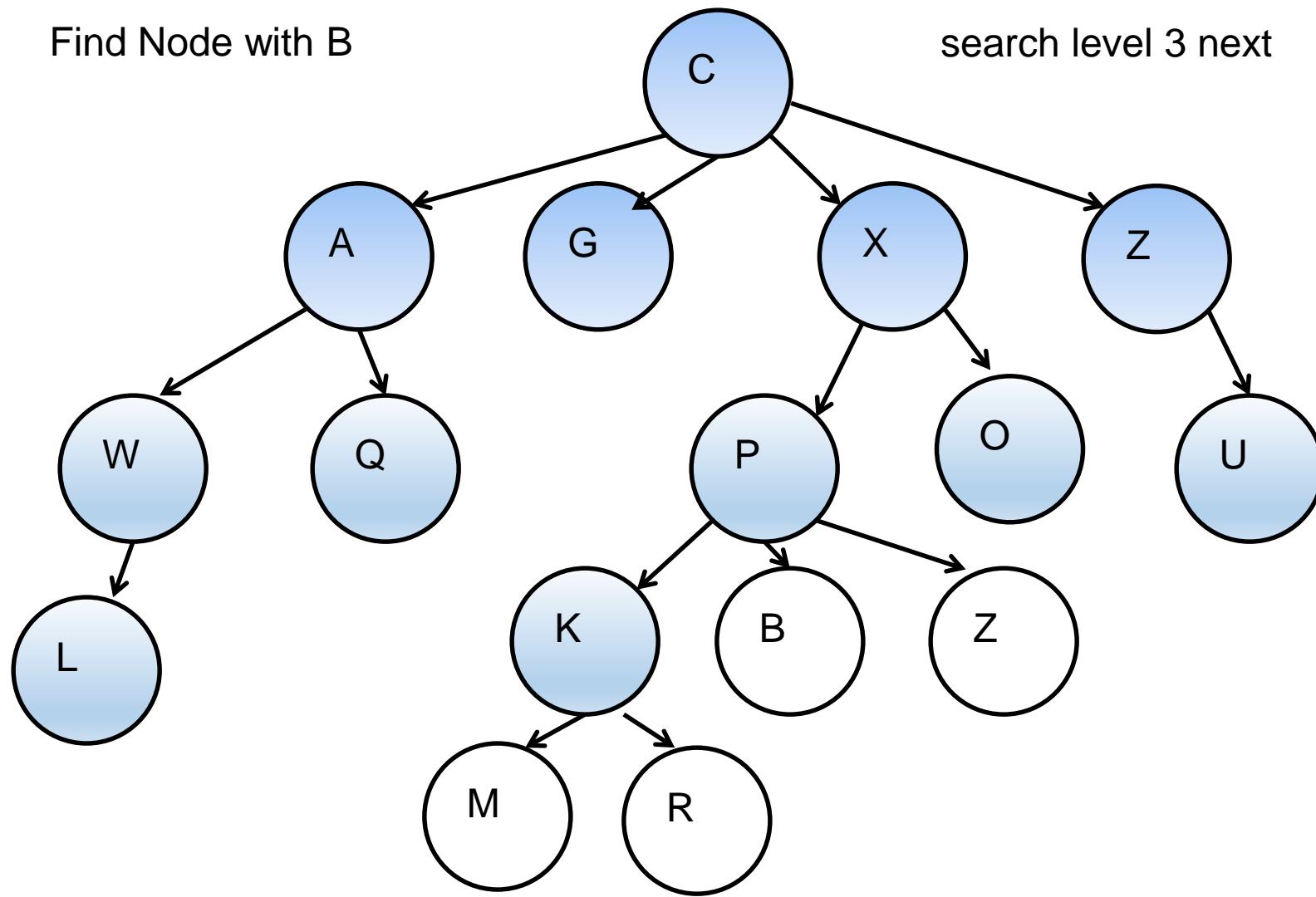
search level 3 next



# Breadth First Search / Өргөнөөр хайх

Find Node with B

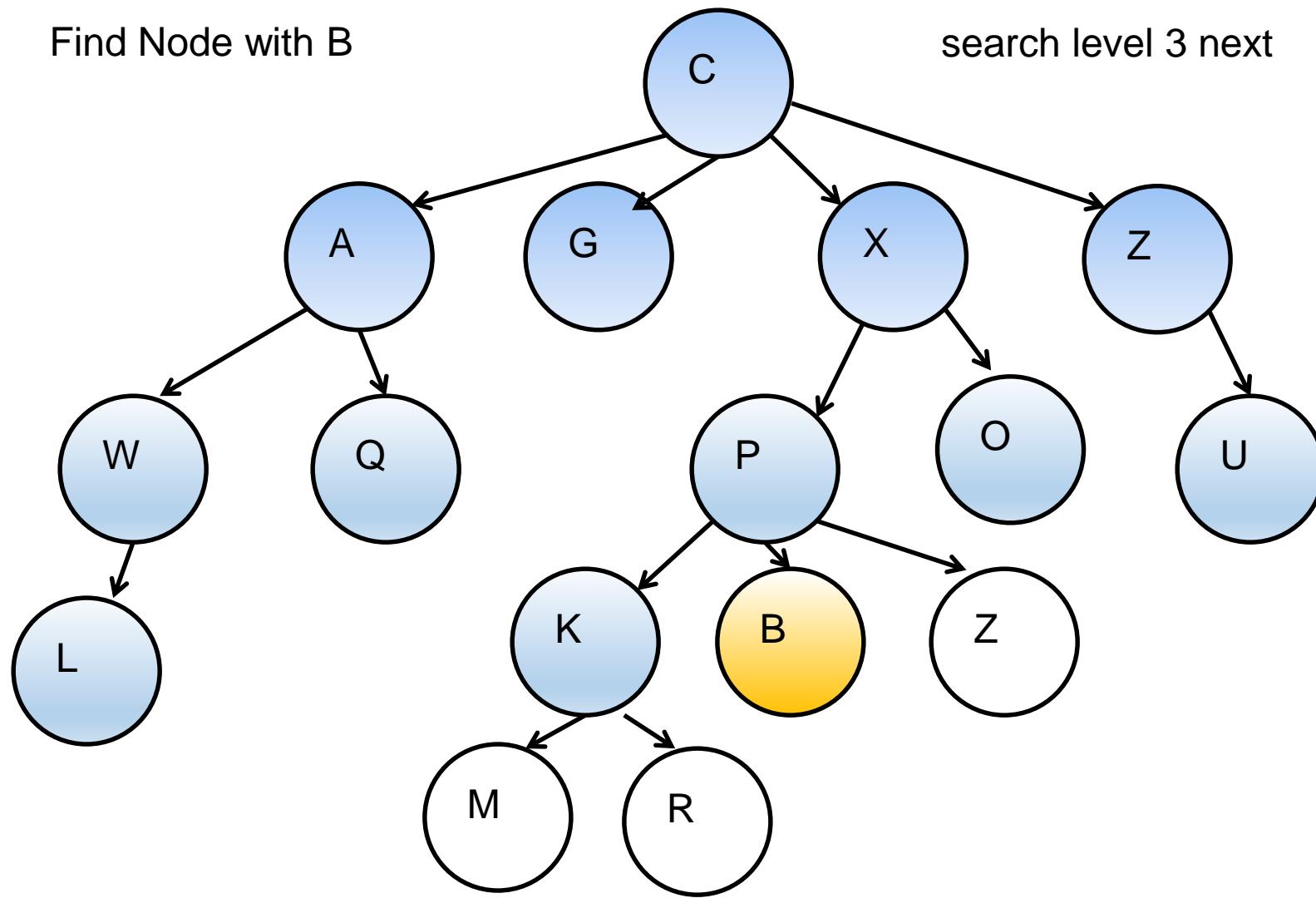
search level 3 next



# Breadth First Search / Өргөнөөр хайх

Find Node with B

search level 3 next

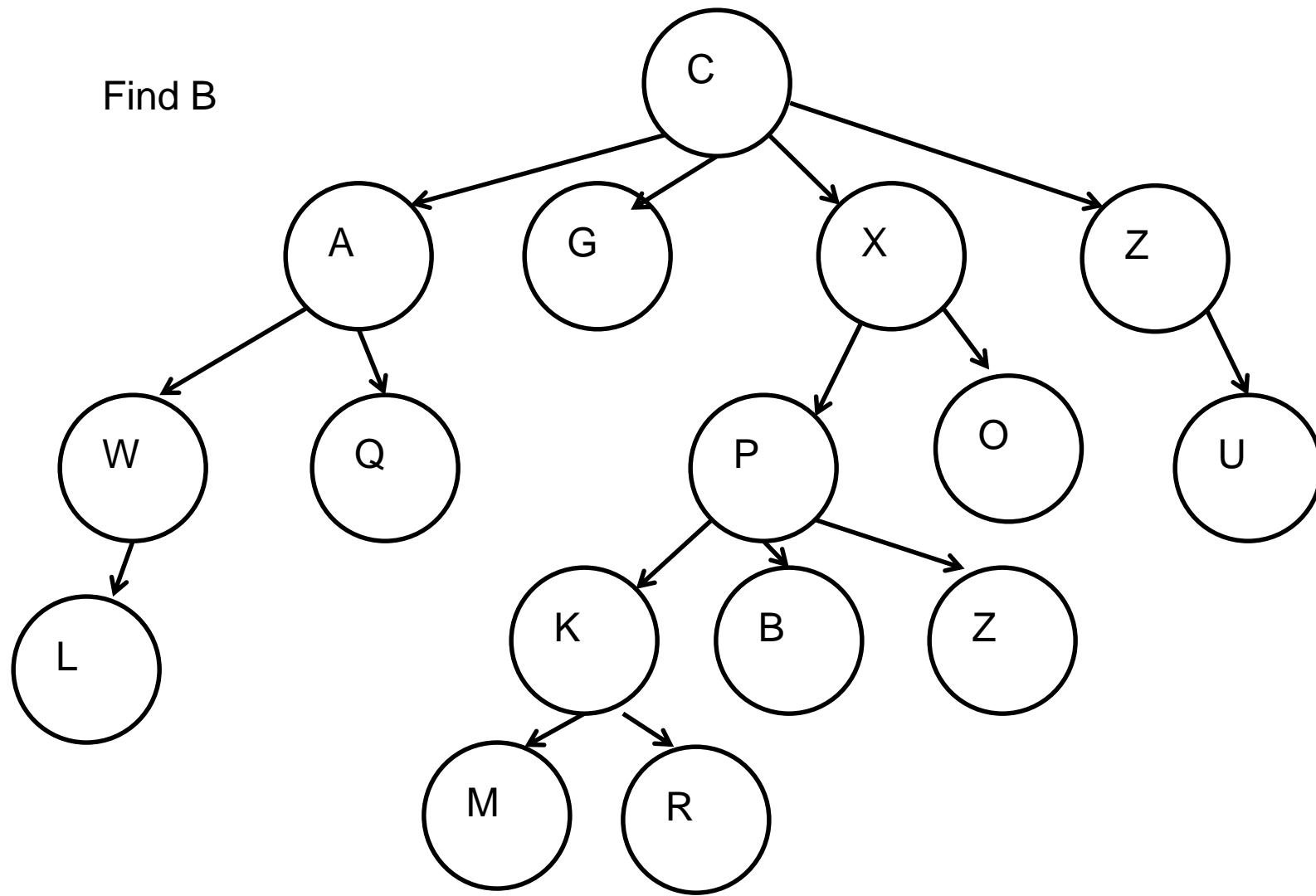


## BFS - DFS

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

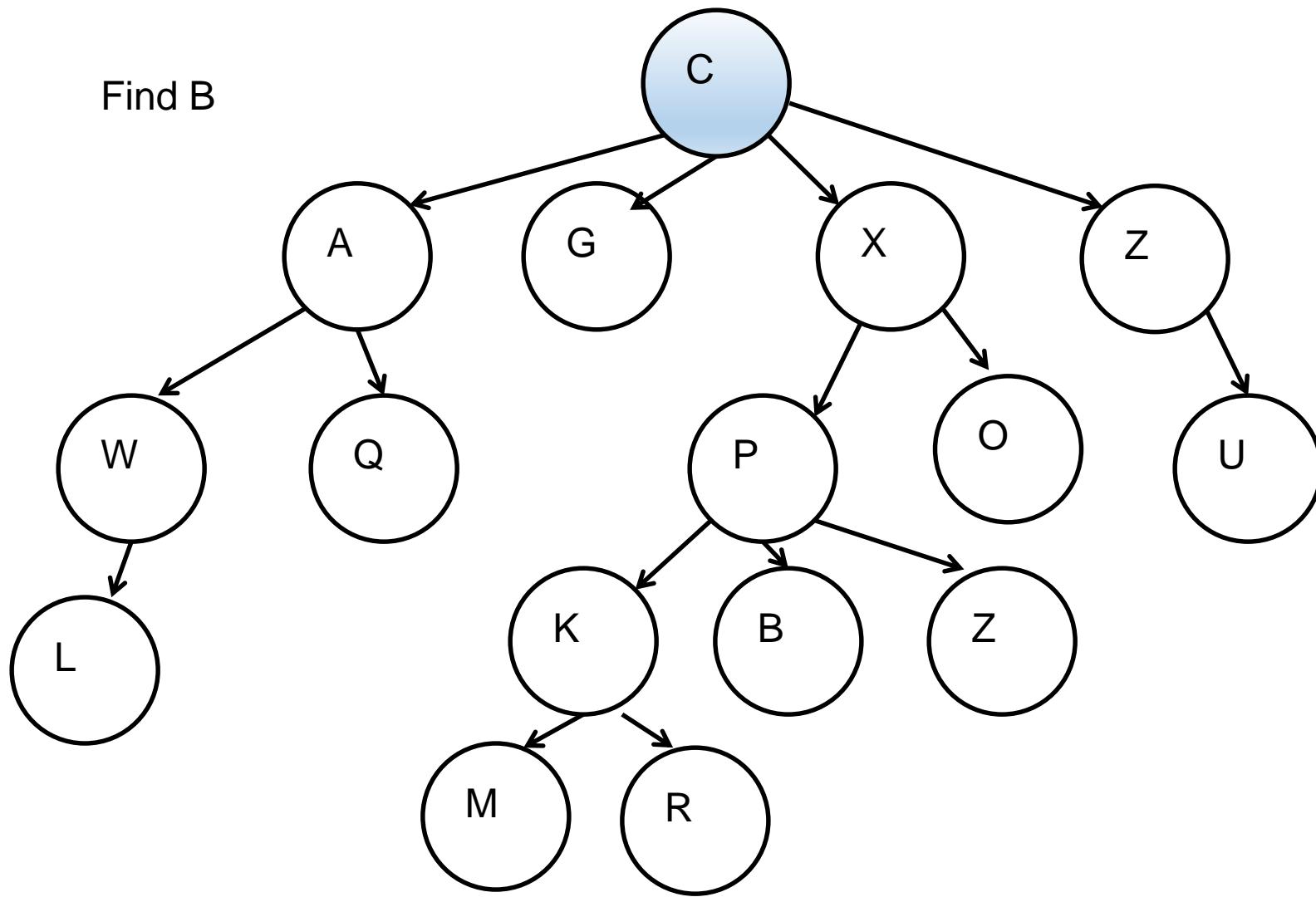
# Depth First Search of Tree / Гүнээр хайх

Find B



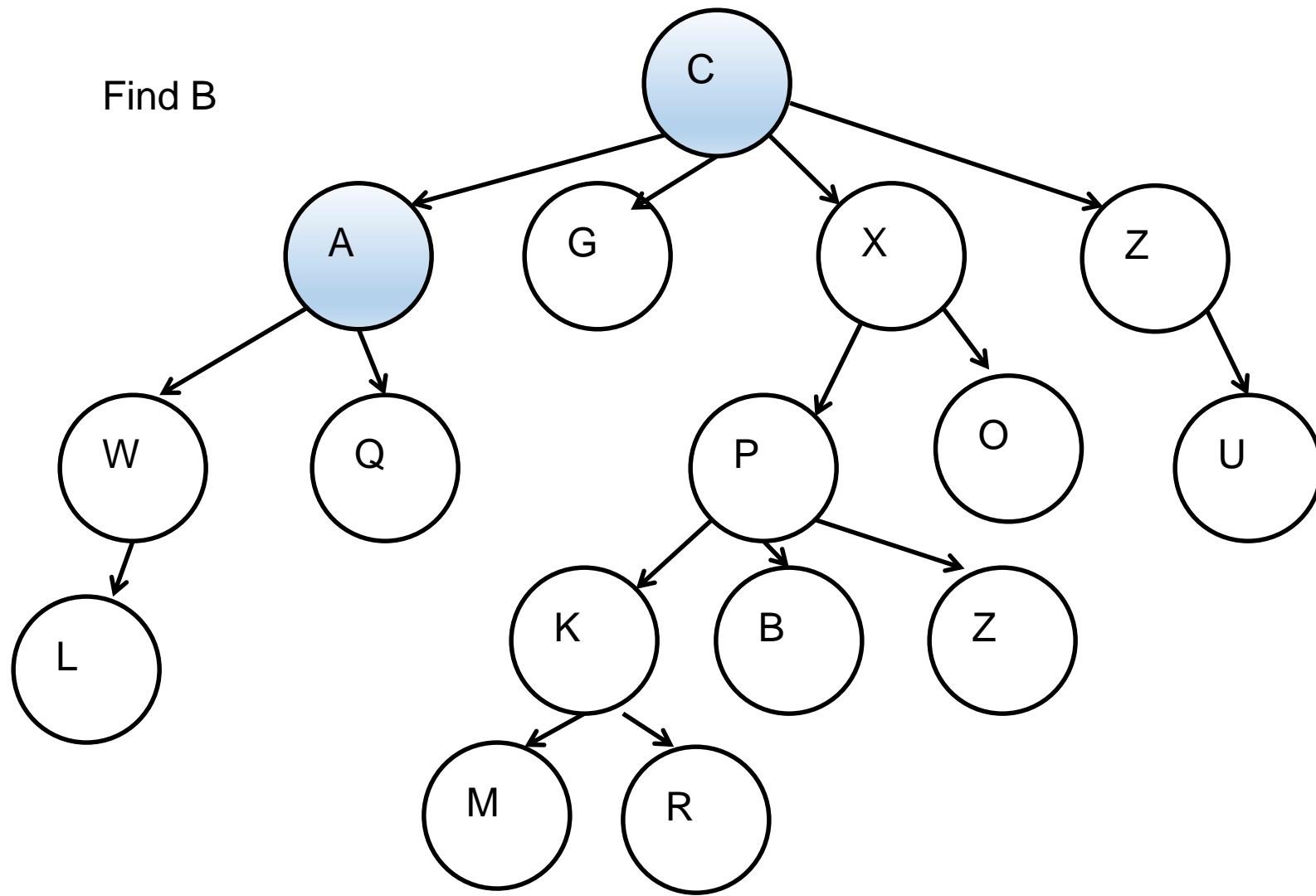
# Depth First Search of Tree / Гүнээр хайх

Find B



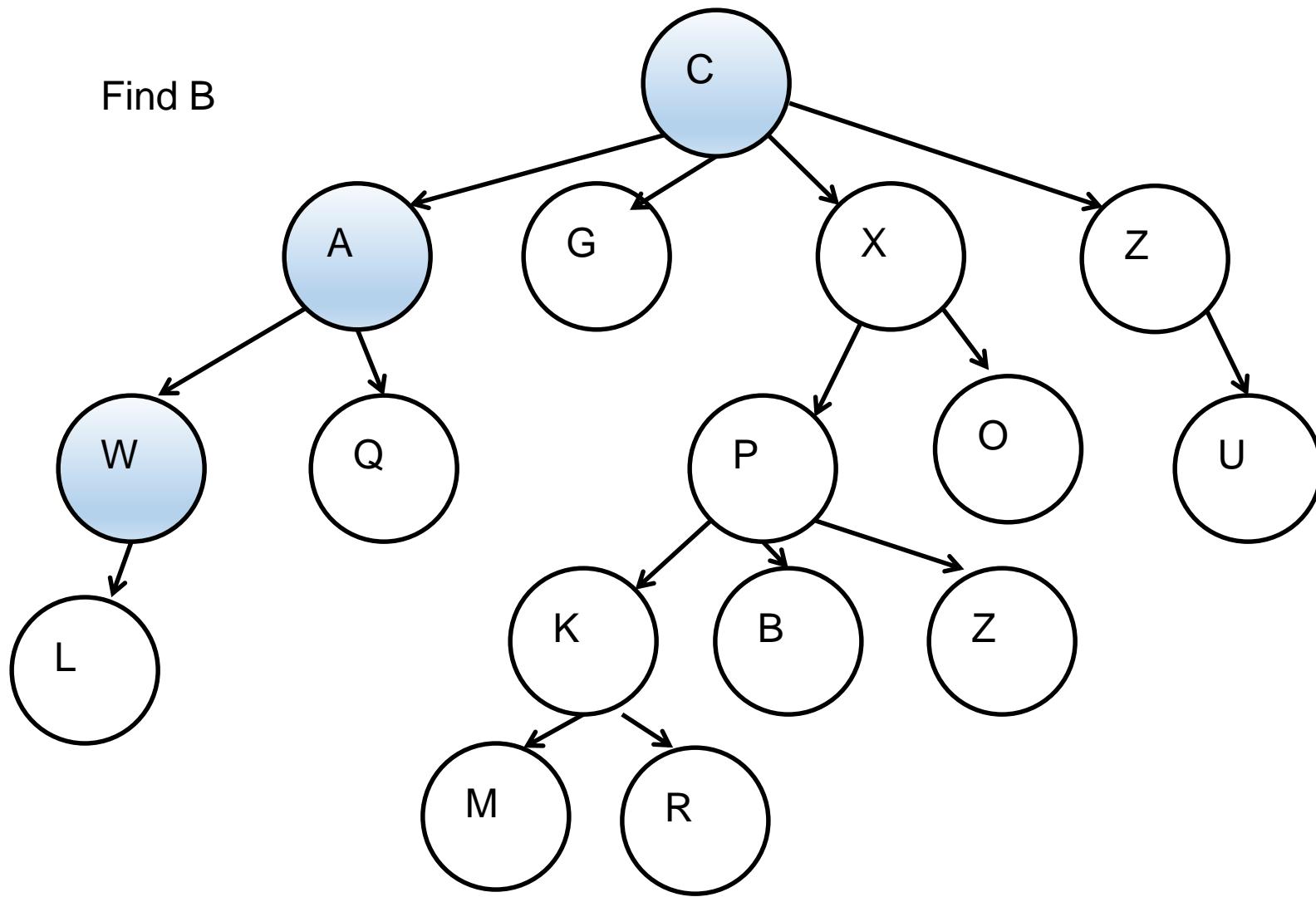
# Depth First Search of Tree / Гүнээр хайх

Find B



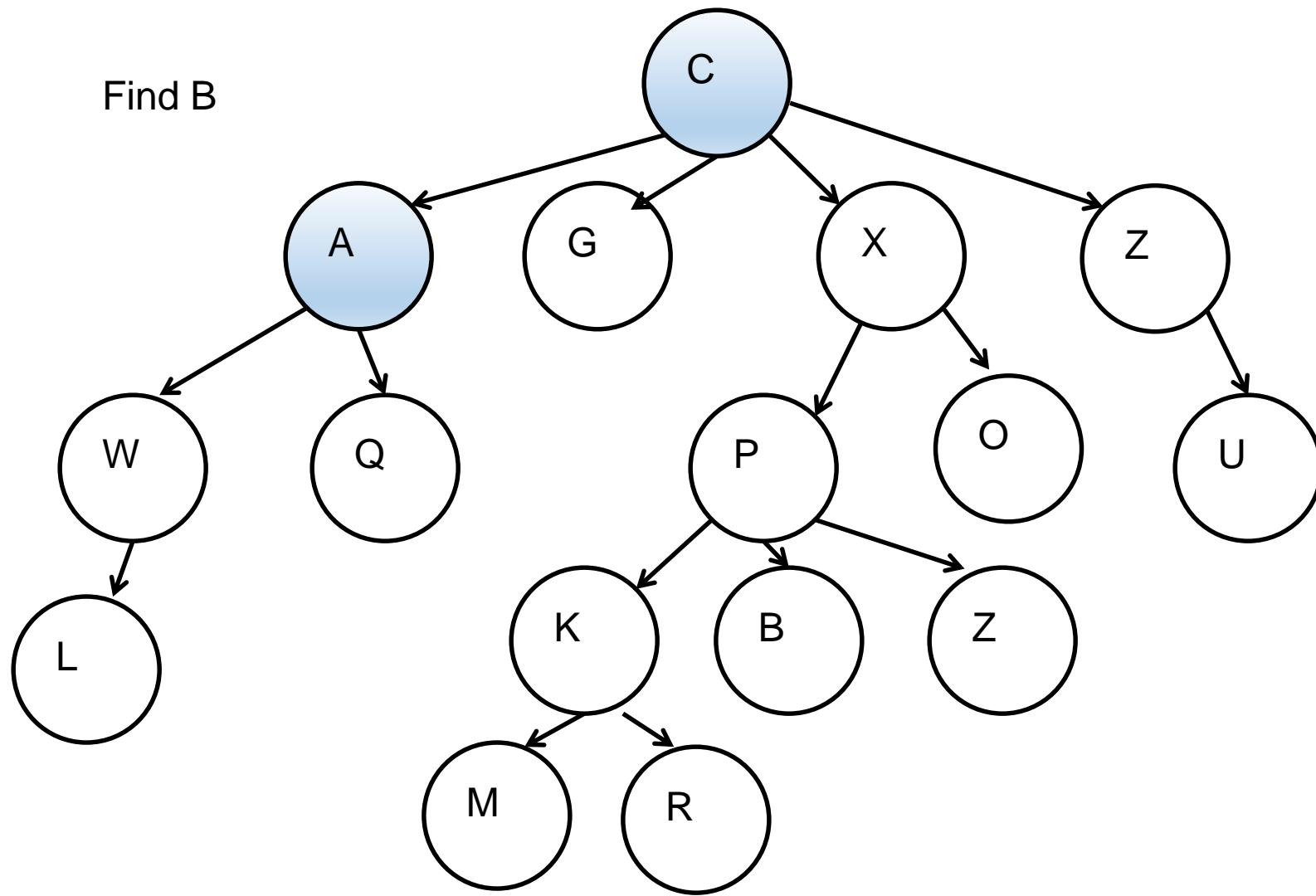
# Depth First Search of Tree / Гүнээр хайх

Find B



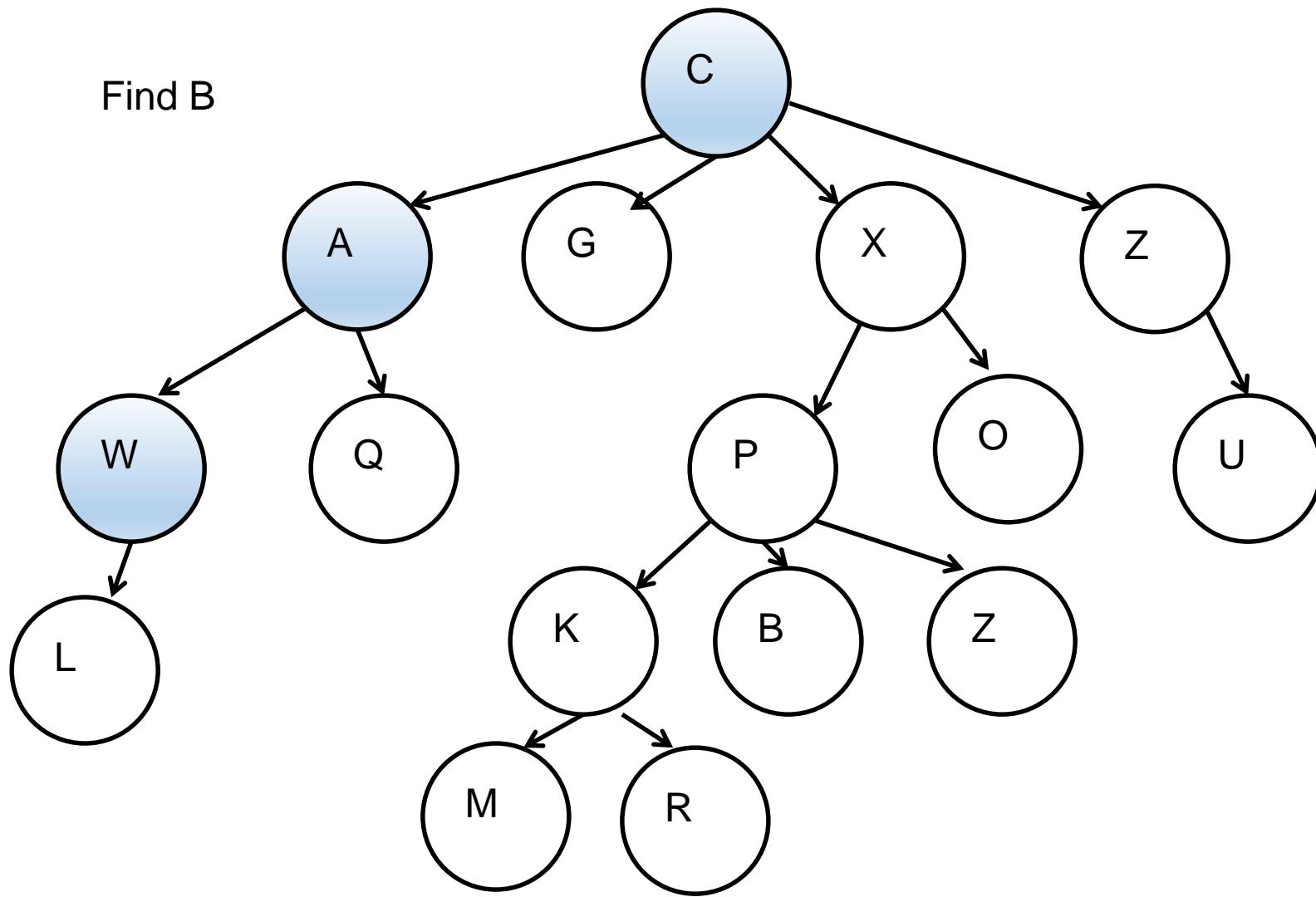
# Depth First Search of Tree / Гүнээр хайх

Find B



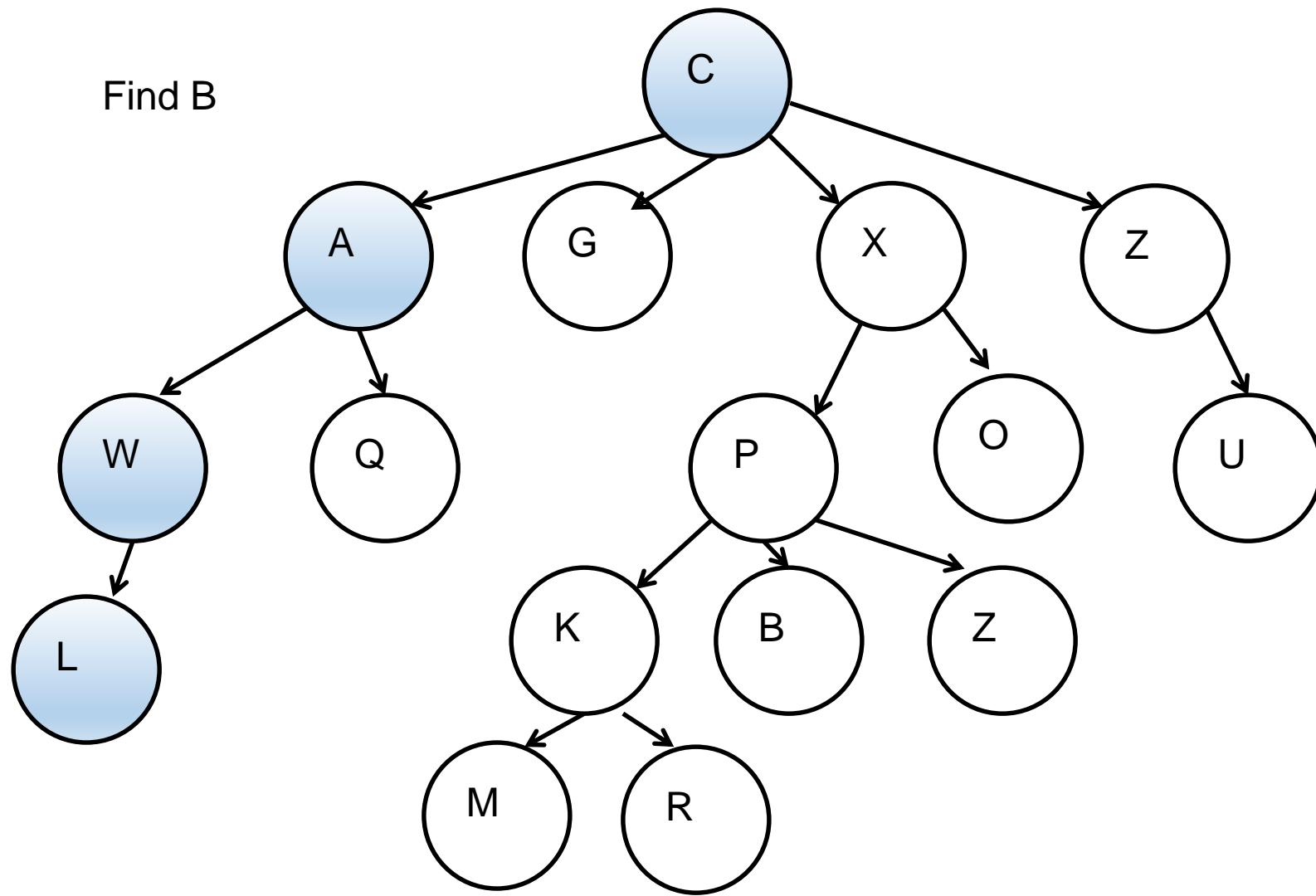
# Depth First Search of Tree / Гүнээр хайх

Find B



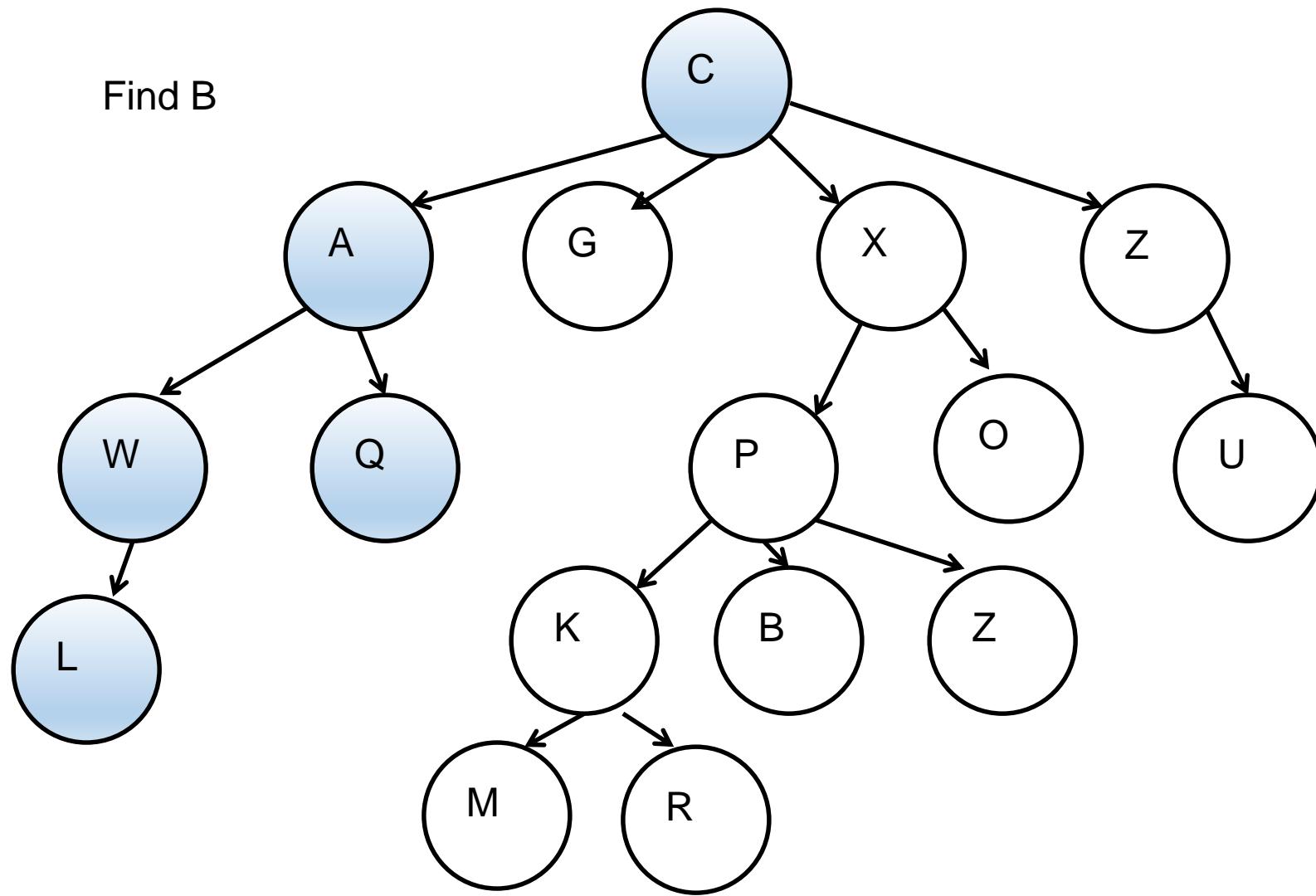
# Depth First Search of Tree / Гүнээр хайх

Find B



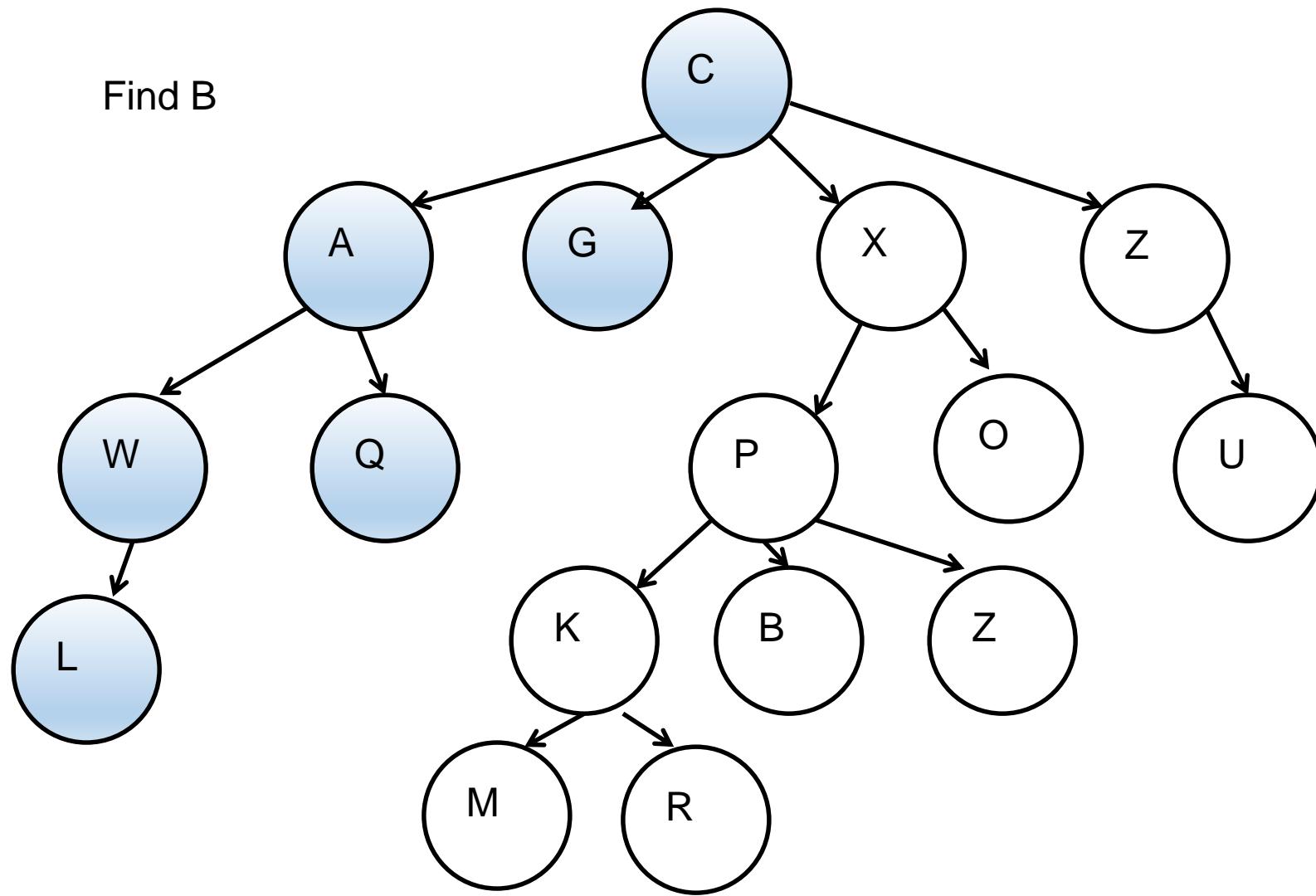
# Depth First Search of Tree / Гүнээр хайх

Find B



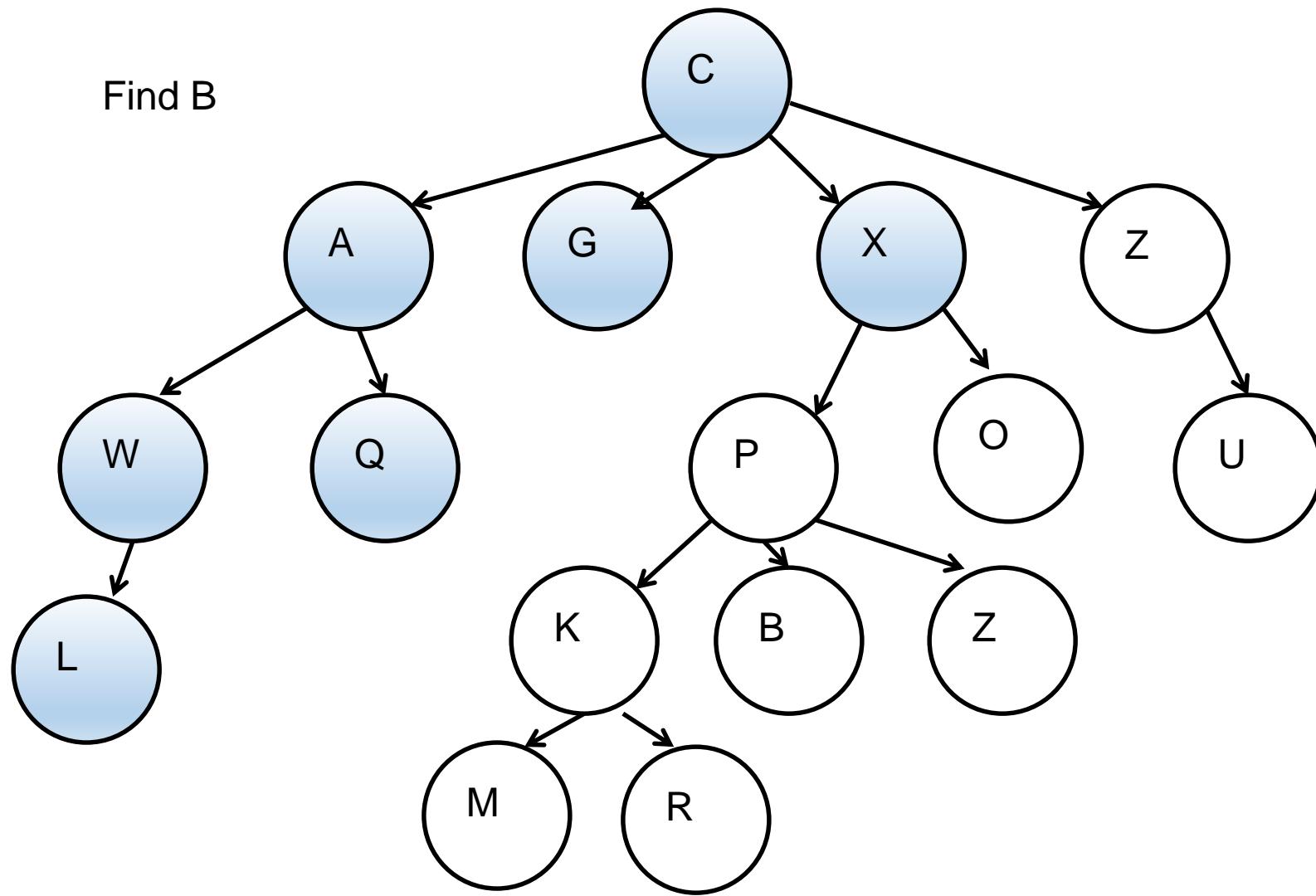
# Depth First Search of Tree / Гүнээр хайх

Find B



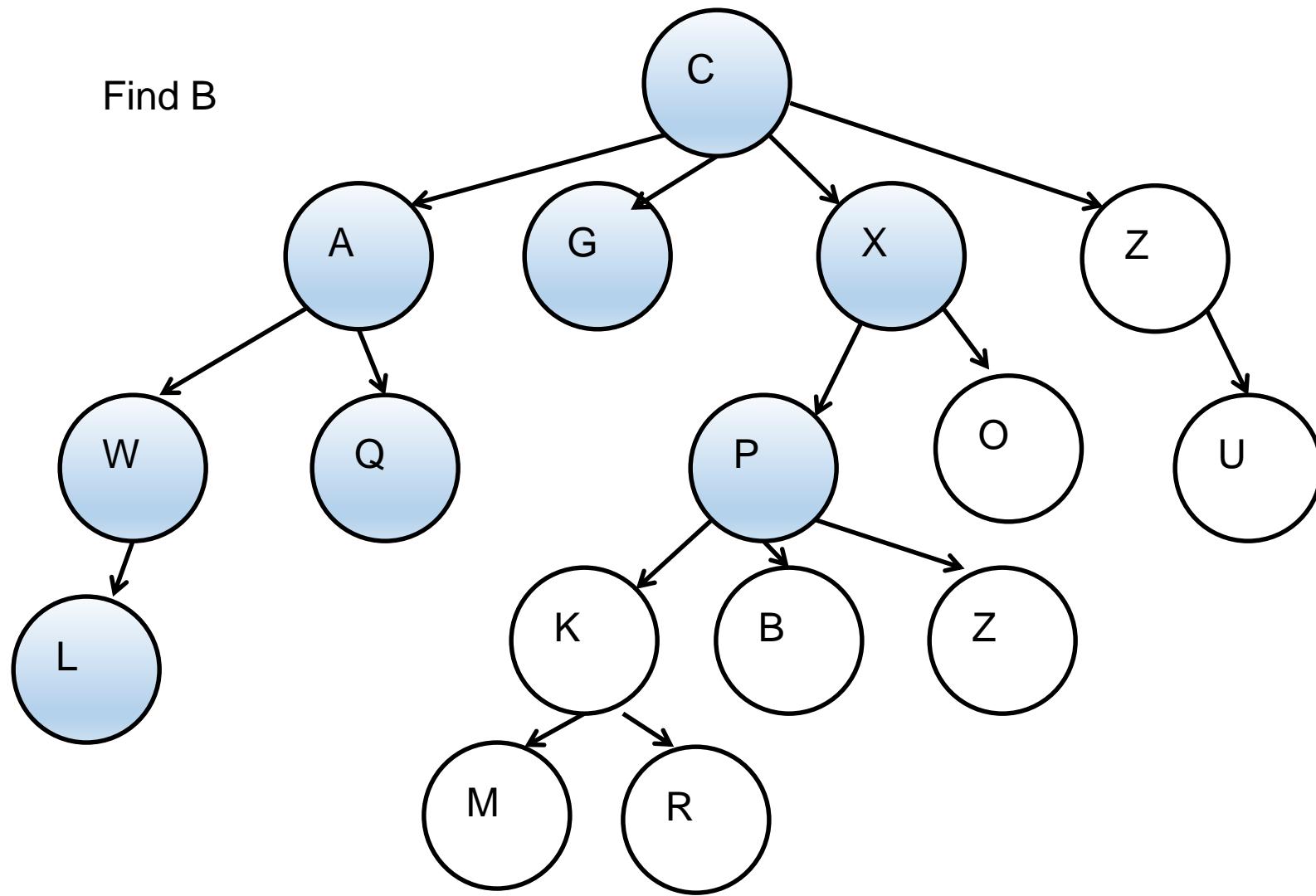
# Depth First Search of Tree / Гүнээр хайх

Find B



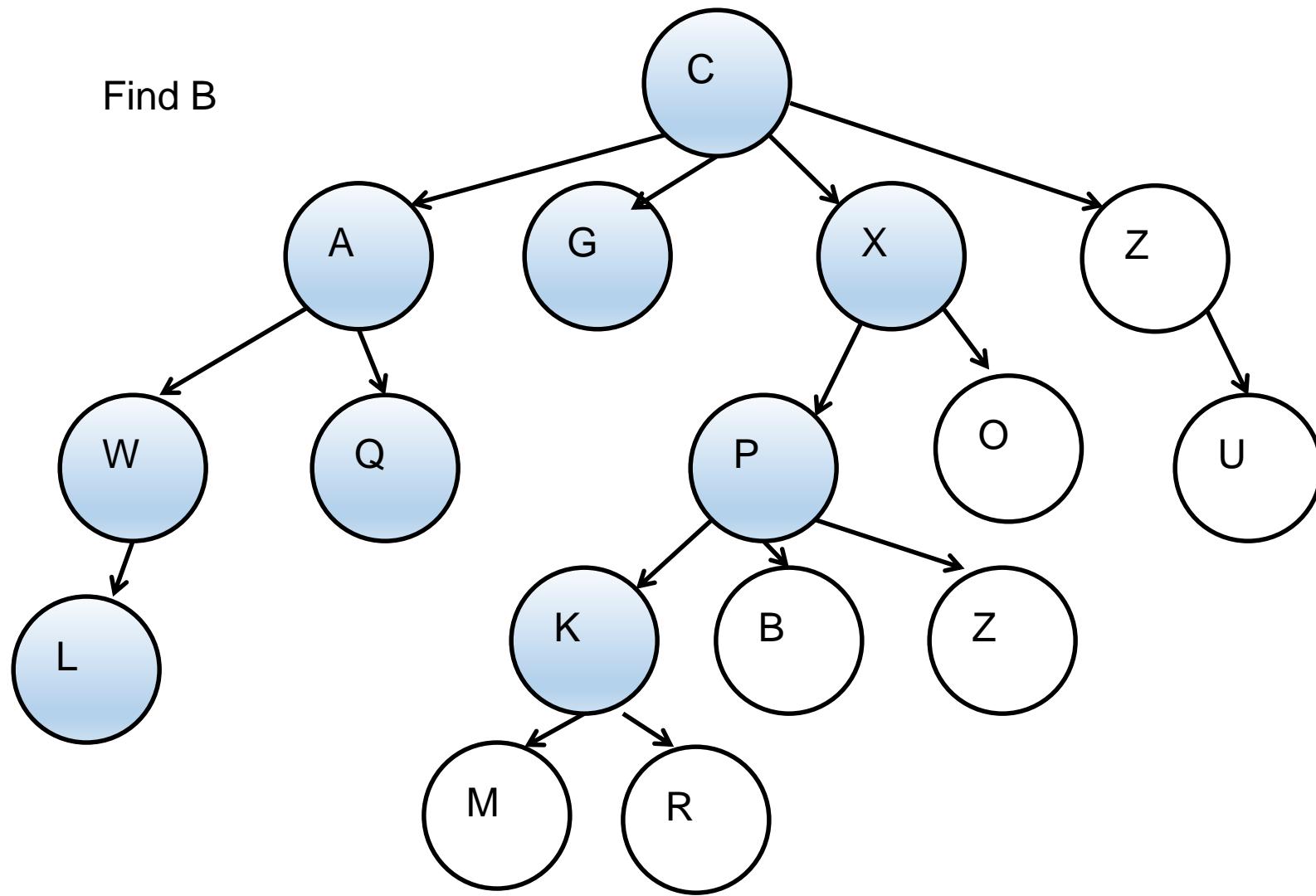
# Depth First Search of Tree / Гүнээр хайх

Find B



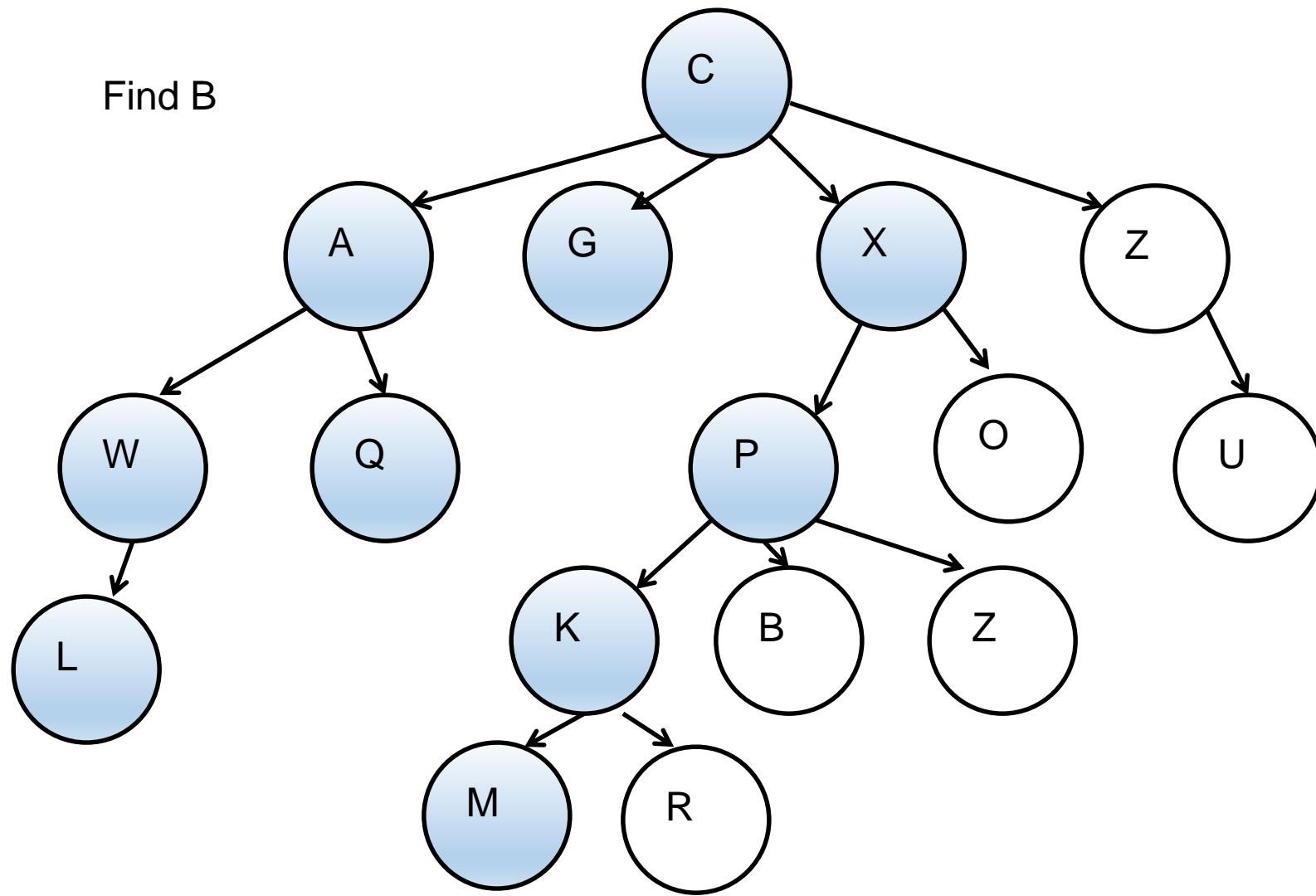
# Depth First Search of Tree / Гүнээр хайх

Find B



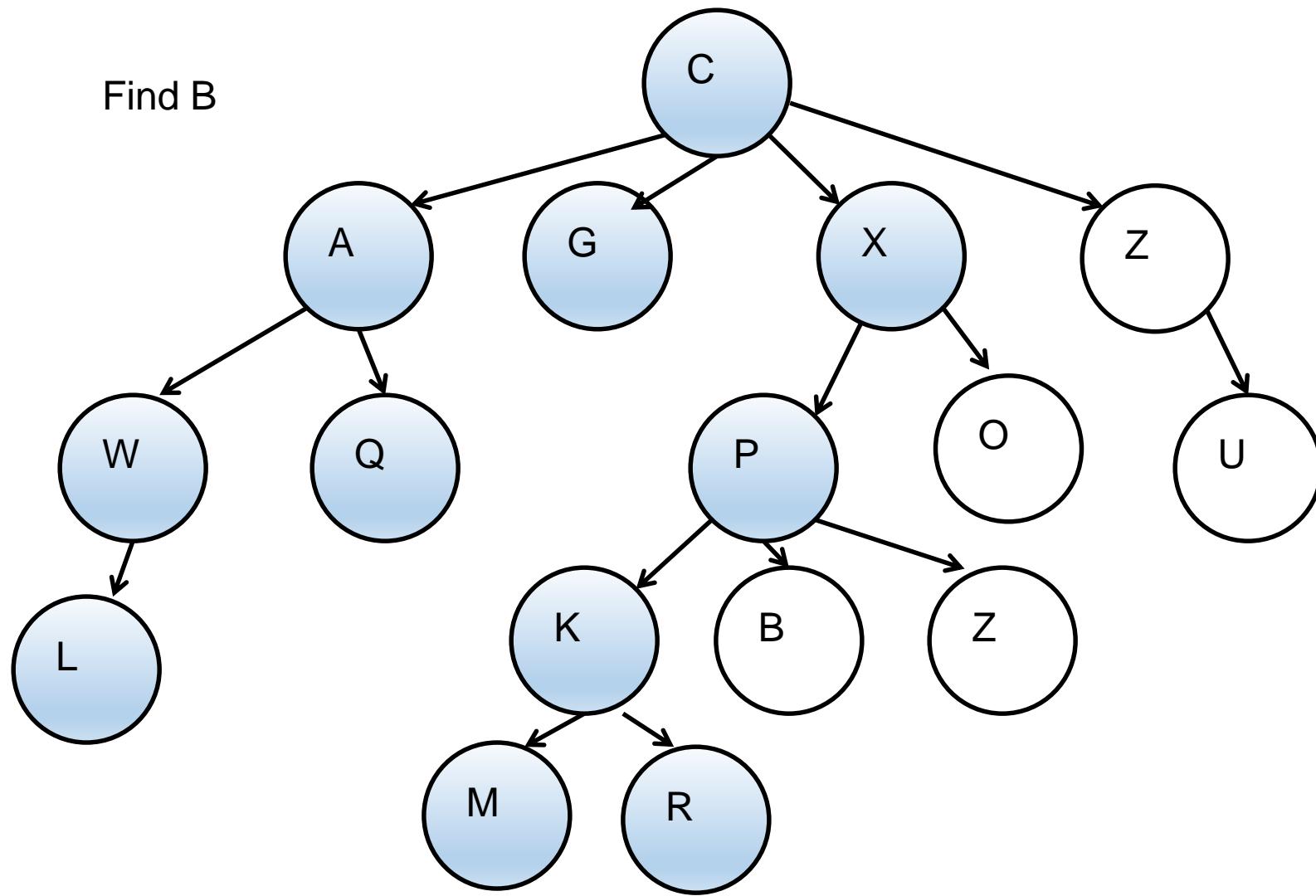
# Depth First Search of Tree / Гүнээр хайх

Find B



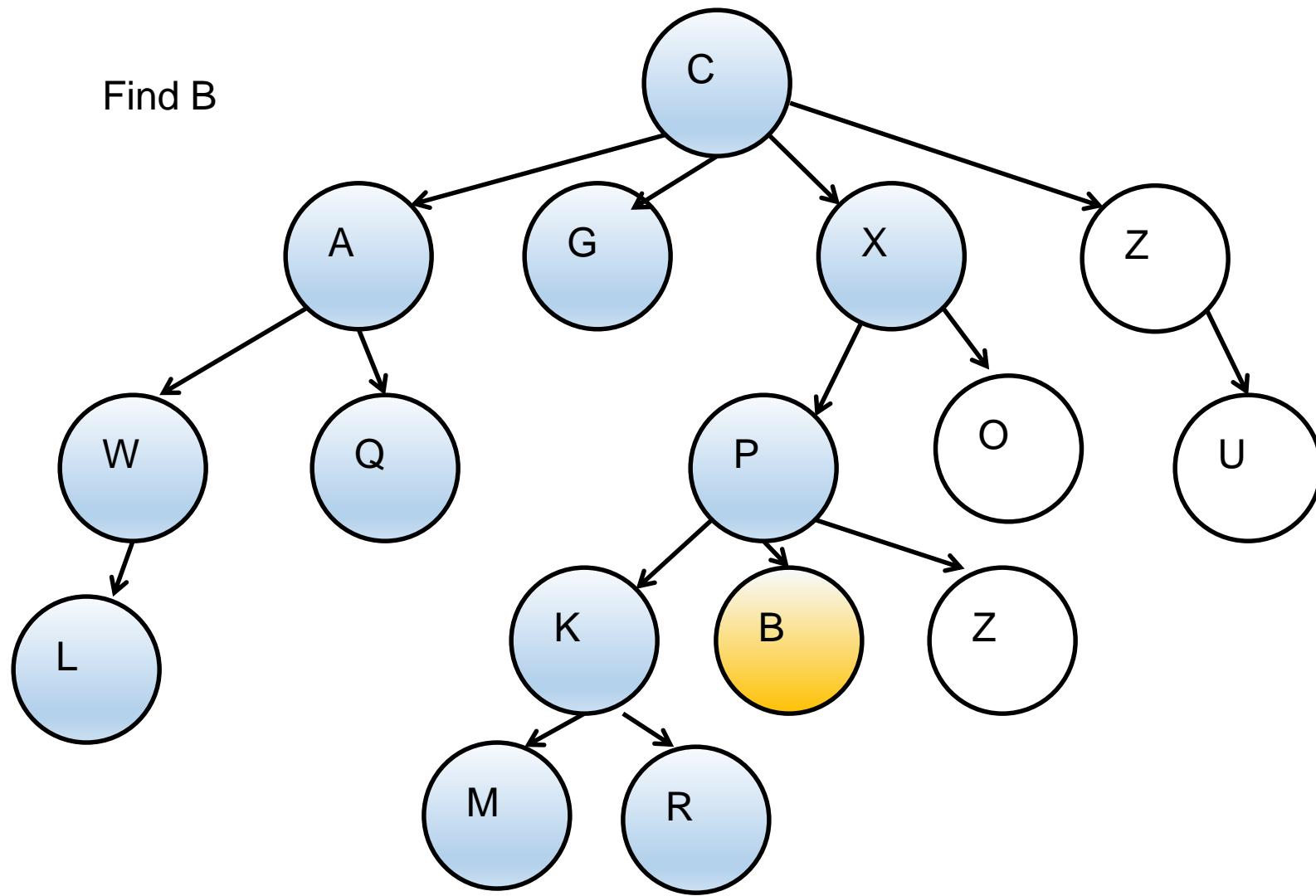
# Depth First Search of Tree / Гүнээр хайх

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

$\Theta(\log n)$ <sup>[1]</sup>

$O(\log n)$ <sup>[1]</sup>

Insert

$\Theta(\log n)$ <sup>[1]</sup>

$O(\log n)$ <sup>[1]</sup>

Delete

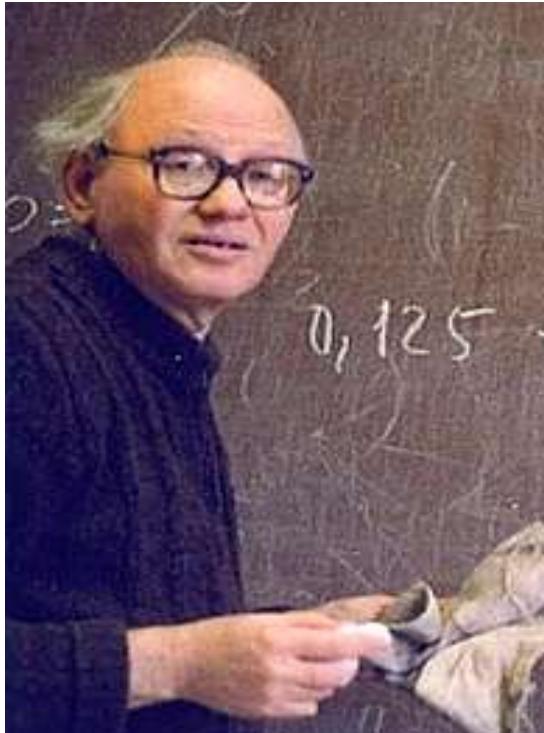
$\Theta(\log n)$ <sup>[1]</sup>

$O(\log n)$ <sup>[1]</sup>

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

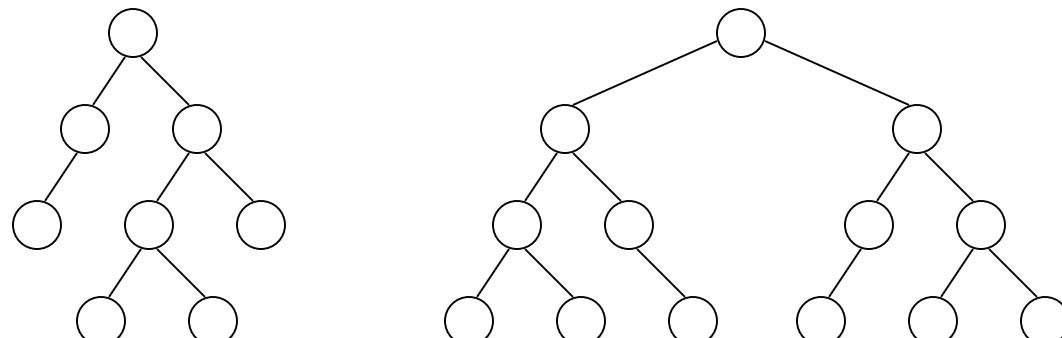
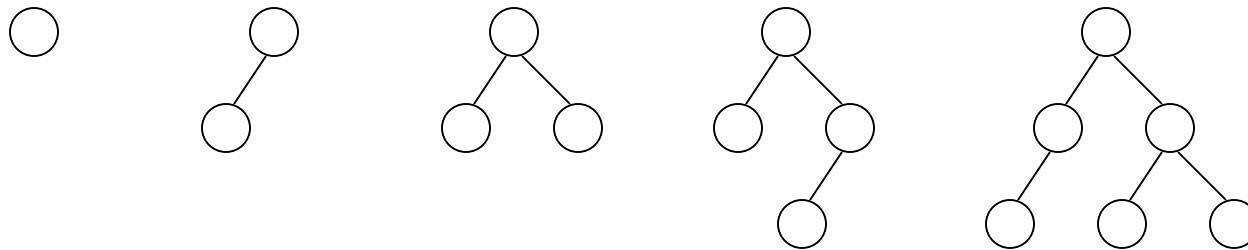
# AVL Tree / Тэнцвэртэй мод

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

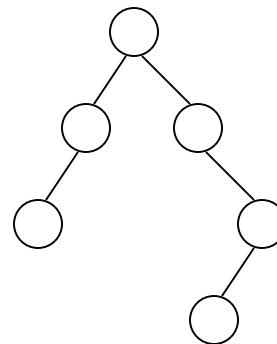
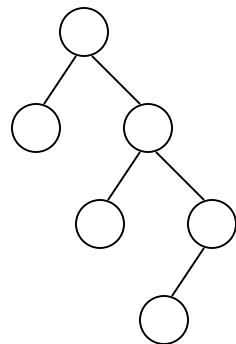
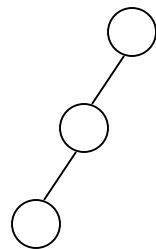


# AVL / Тэнцвэртэй мод

- 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

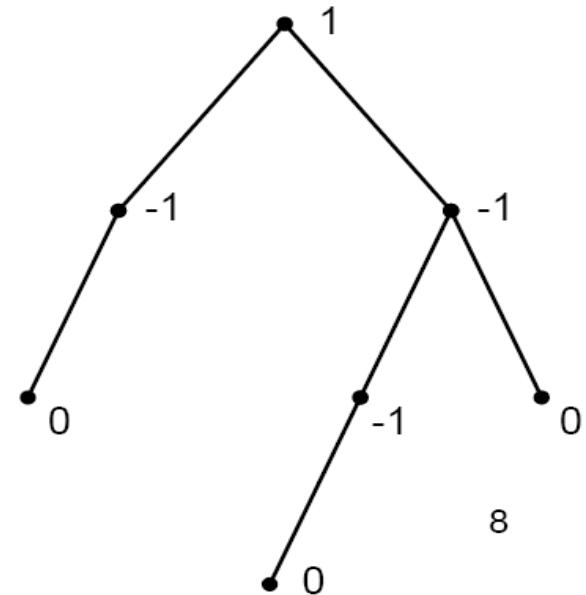
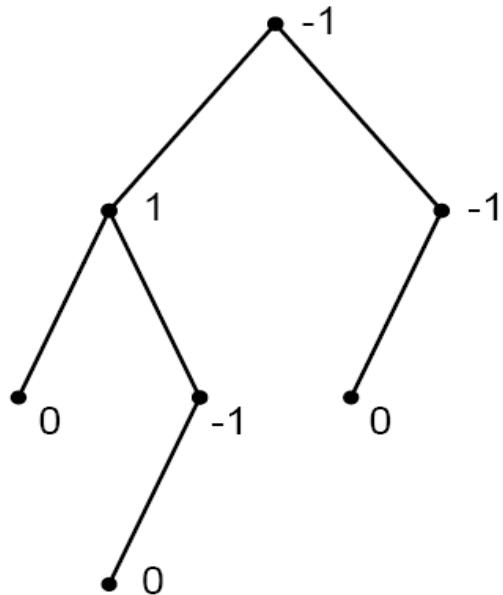
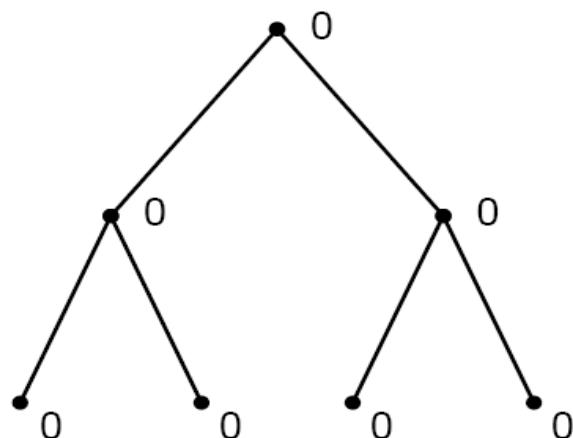
- Хоёртын хайлтын мод нь AVL мод мөн эсэхийг хянахын тулд зангилаа бүрд тэнцвэрийн хүчин зүйлийг шалгана.
- Өндөр (баруун дэд мод) – Өндөр (зүүн дэд мод)

Height(right subtree) – Height(left subtree)

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

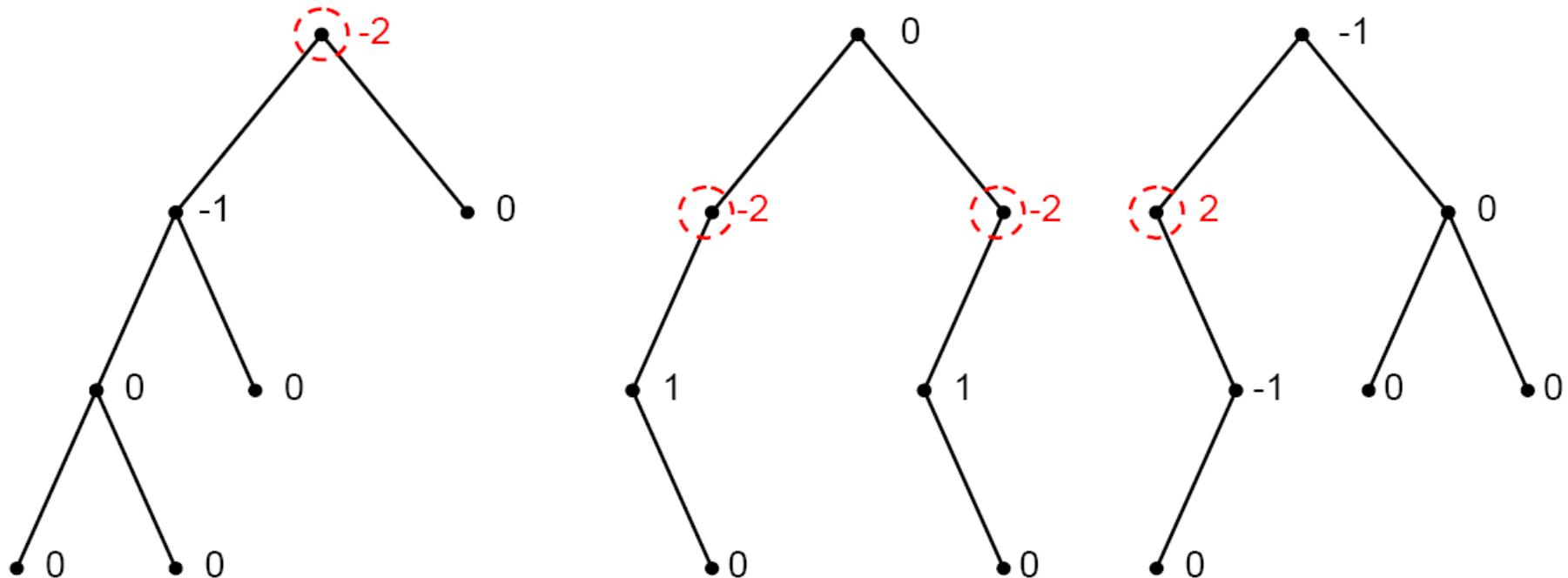
# AVL Мод

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



# AVL Мод биш

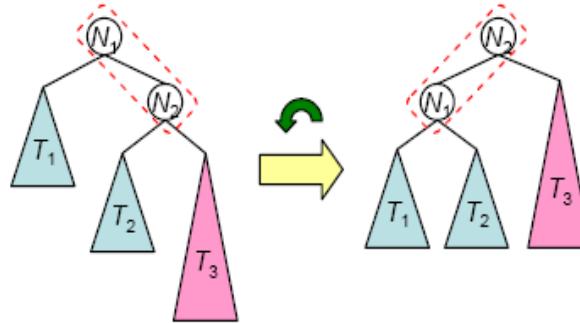
- Өндөр (баруун дэд мод) – Өндөр (зүүн дэд мод)



# Imbalance / Тэнцвэргүй байдал

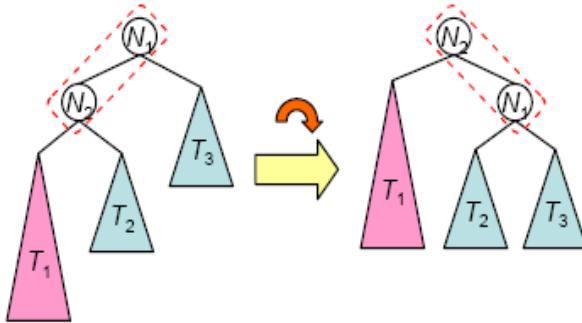
**Case 1:** insertion to *right* subtree of *right* child

Solution: *Left* rotation



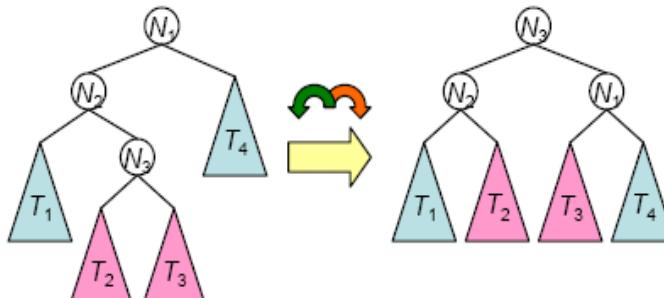
**Case 2:** insertion to *left* subtree of *left* child

Solution: *Right* rotation



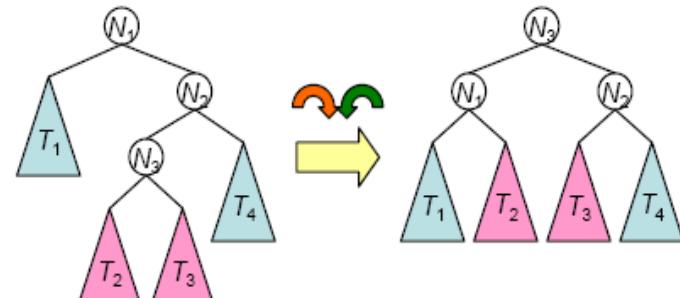
**Case 3:** insertion to *right* subtree of *left* child

Solution: *Left-right* rotation



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

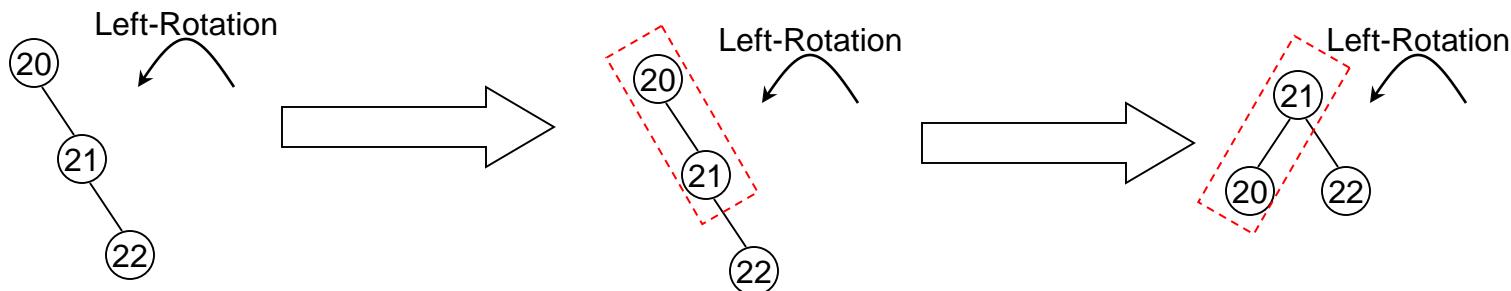
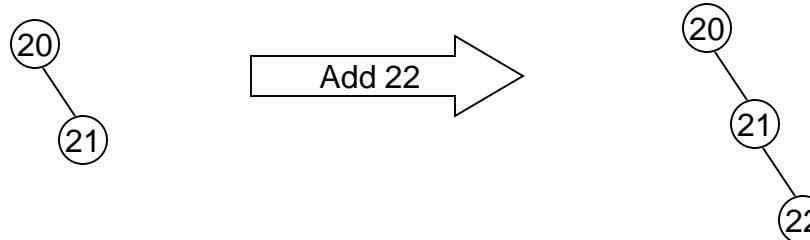
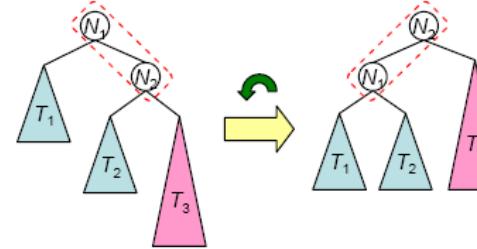
Solution: *Right-left* rotation



# Left-Rotation Зүүн-Эргүүлэлт

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

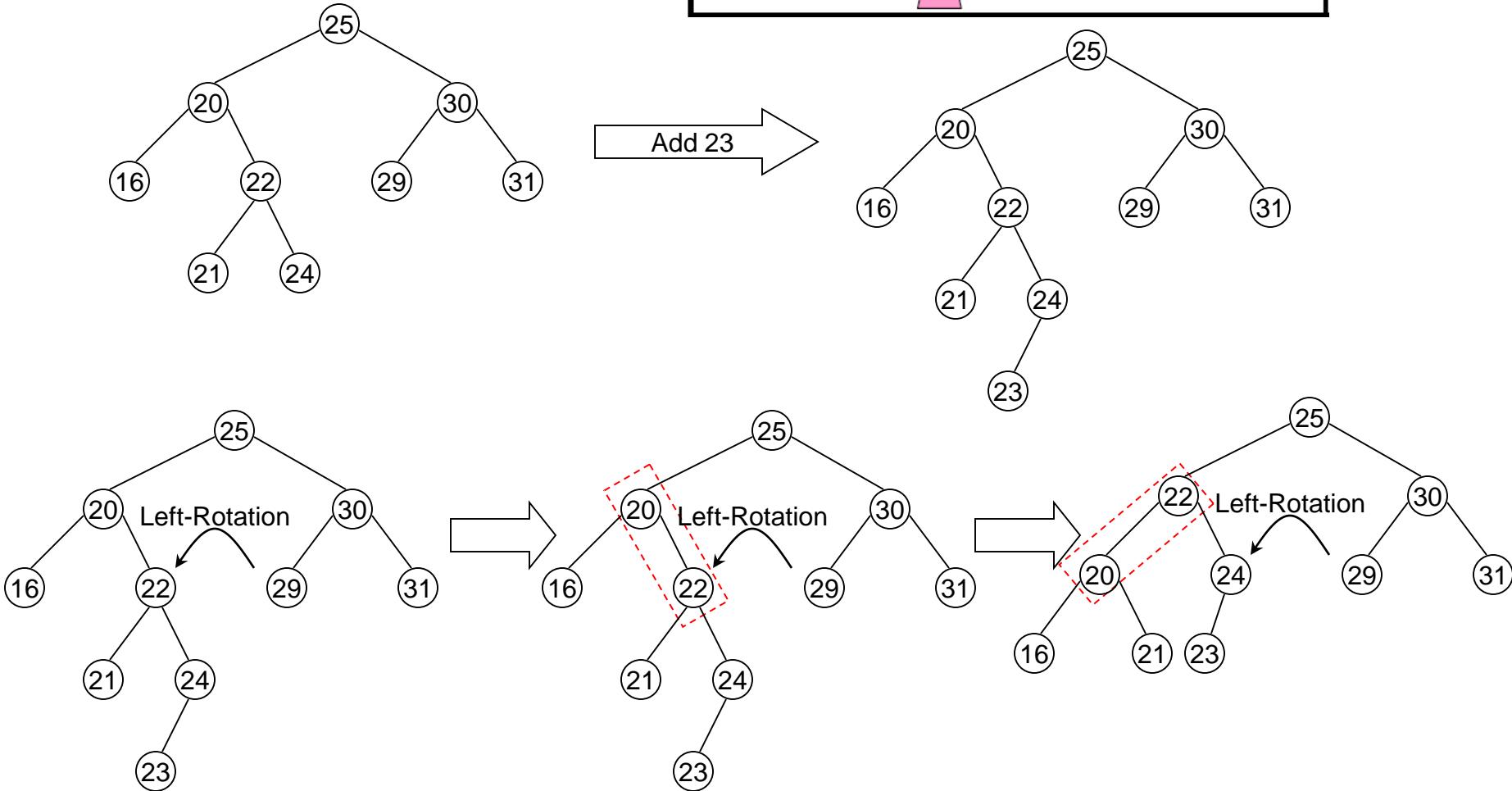
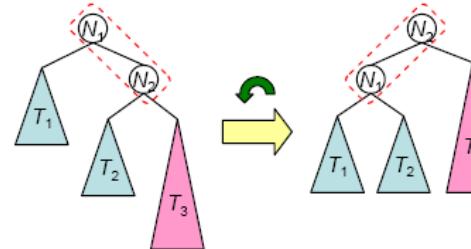
Solution: *Left* rotation



# Left-Rotation Зүүн-Эргүүлэлт

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

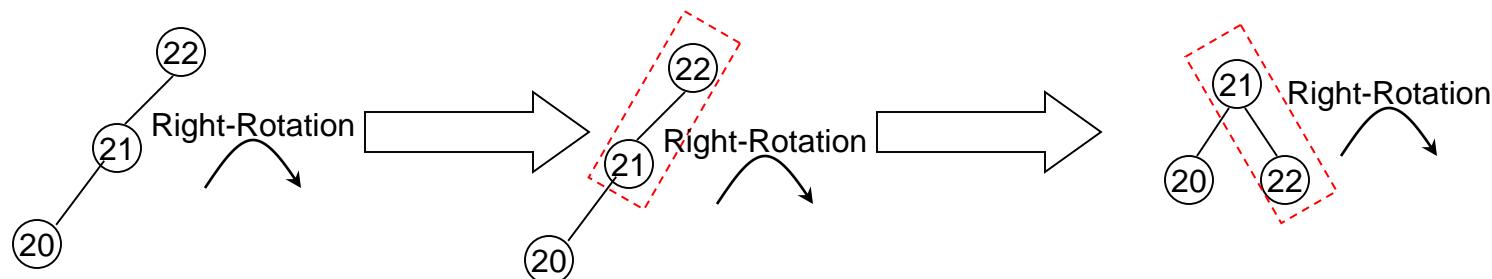
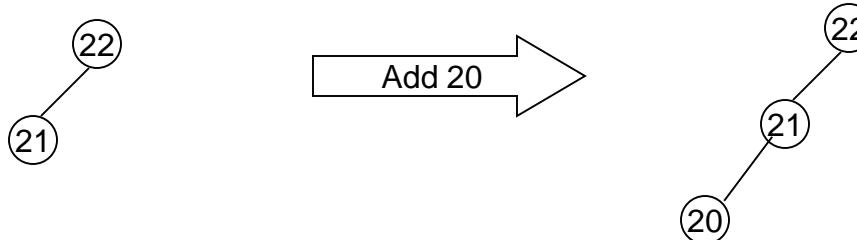
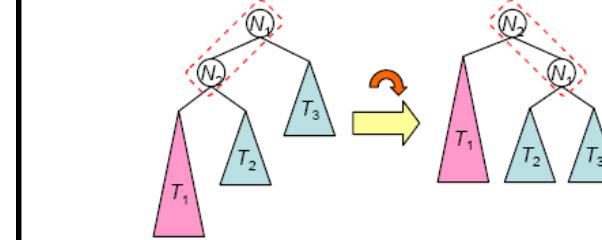
Solution: *Left* rotation



# Right-Rotation Баруун-Эргүүлэлт

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

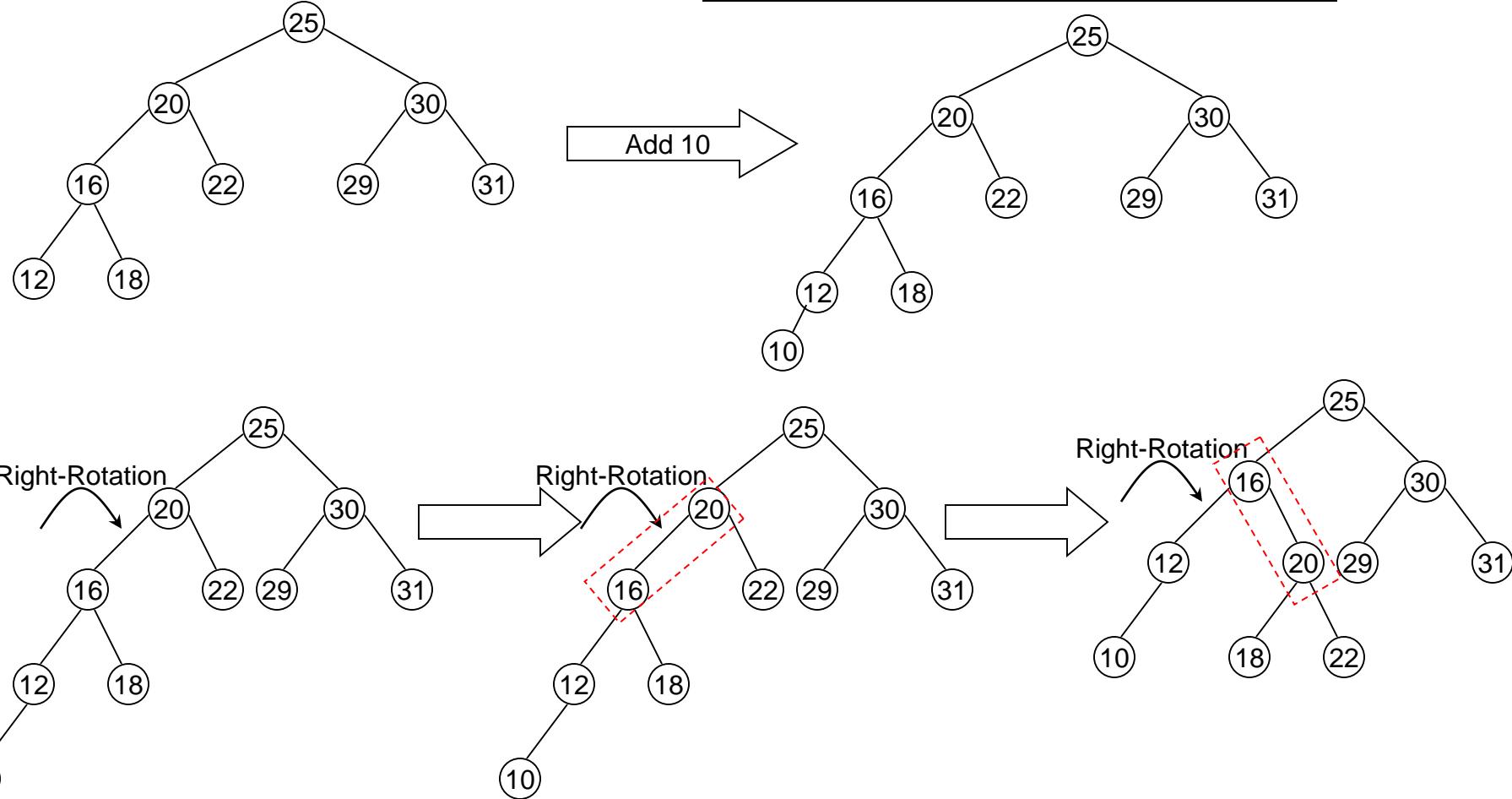
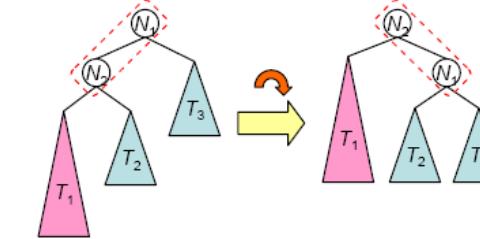
Solution: *Right* rotation



# Right-Rotation Баруун-Эргүүлэлт

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

Solution: *Right* rotation

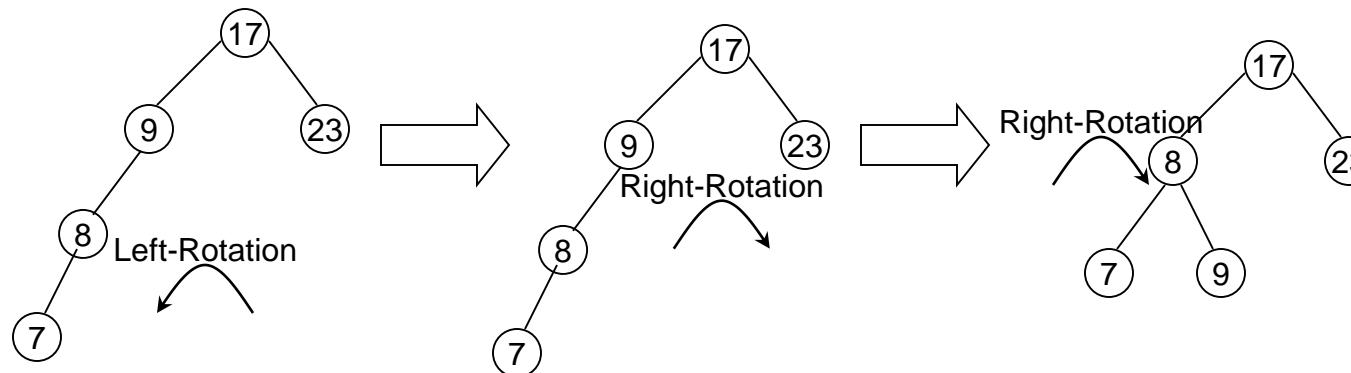
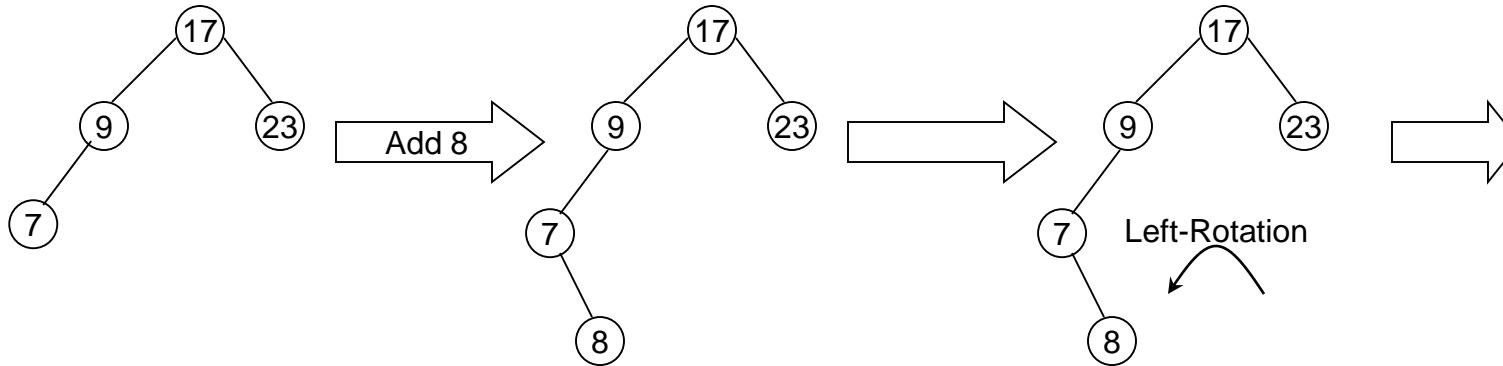
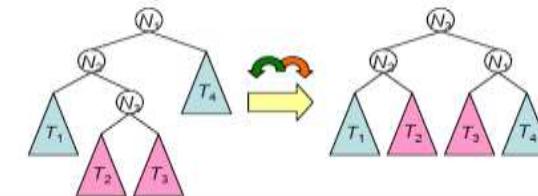


# Left-Right-Rotation

## Зүүн-Баруун-Эргүүлэлт

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

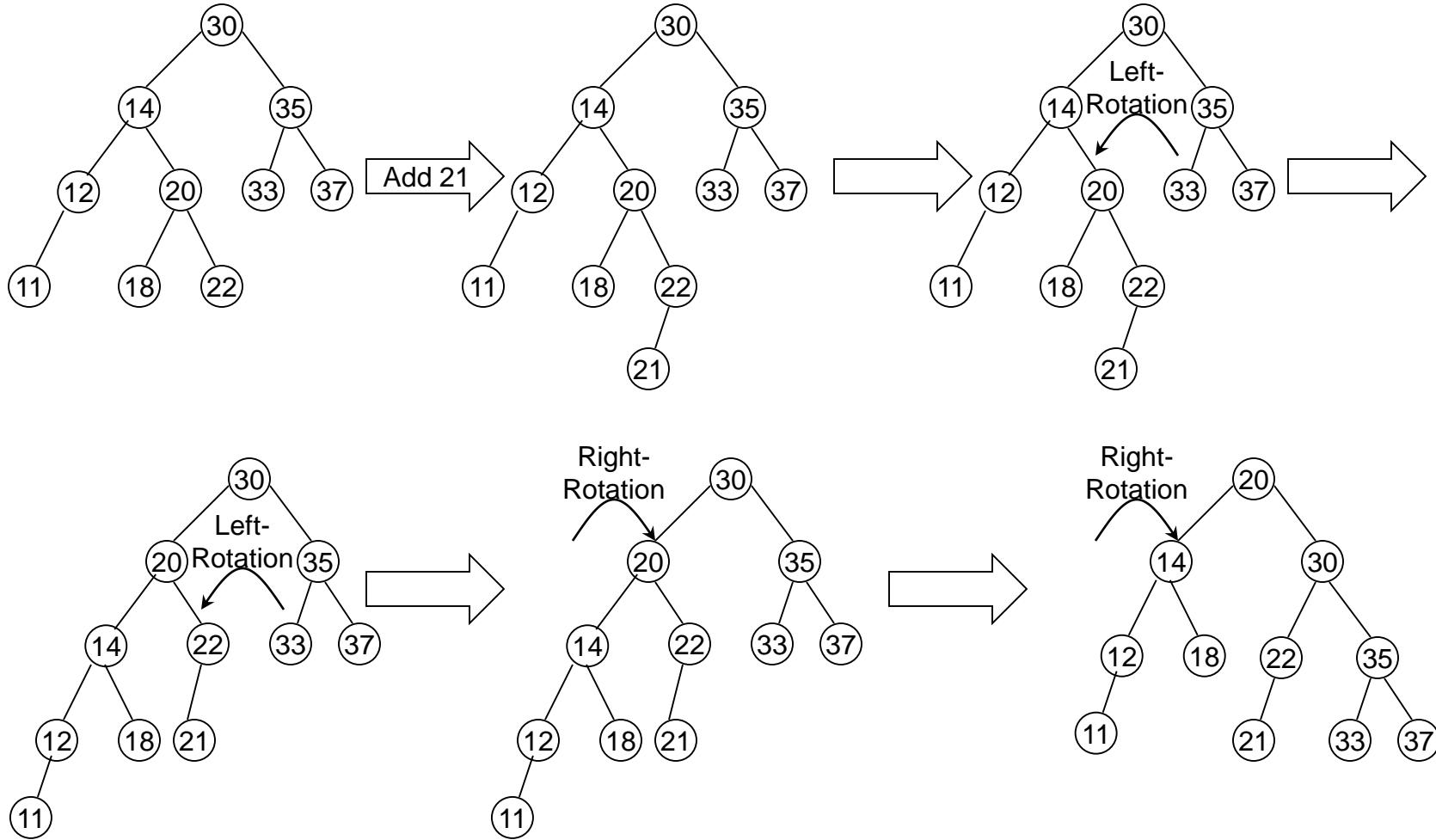
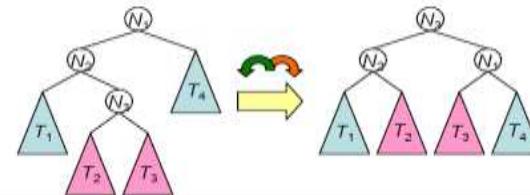
Solution: *Left-right* rotation



# Left-Right-Rotation Зүүн-Баруун-Эргүүлэлт

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

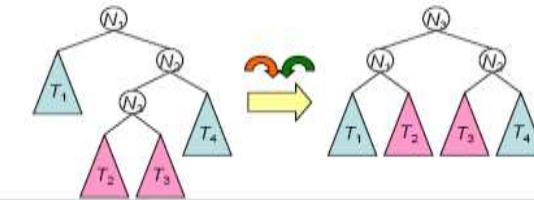
Solution: *Left-right* rotation



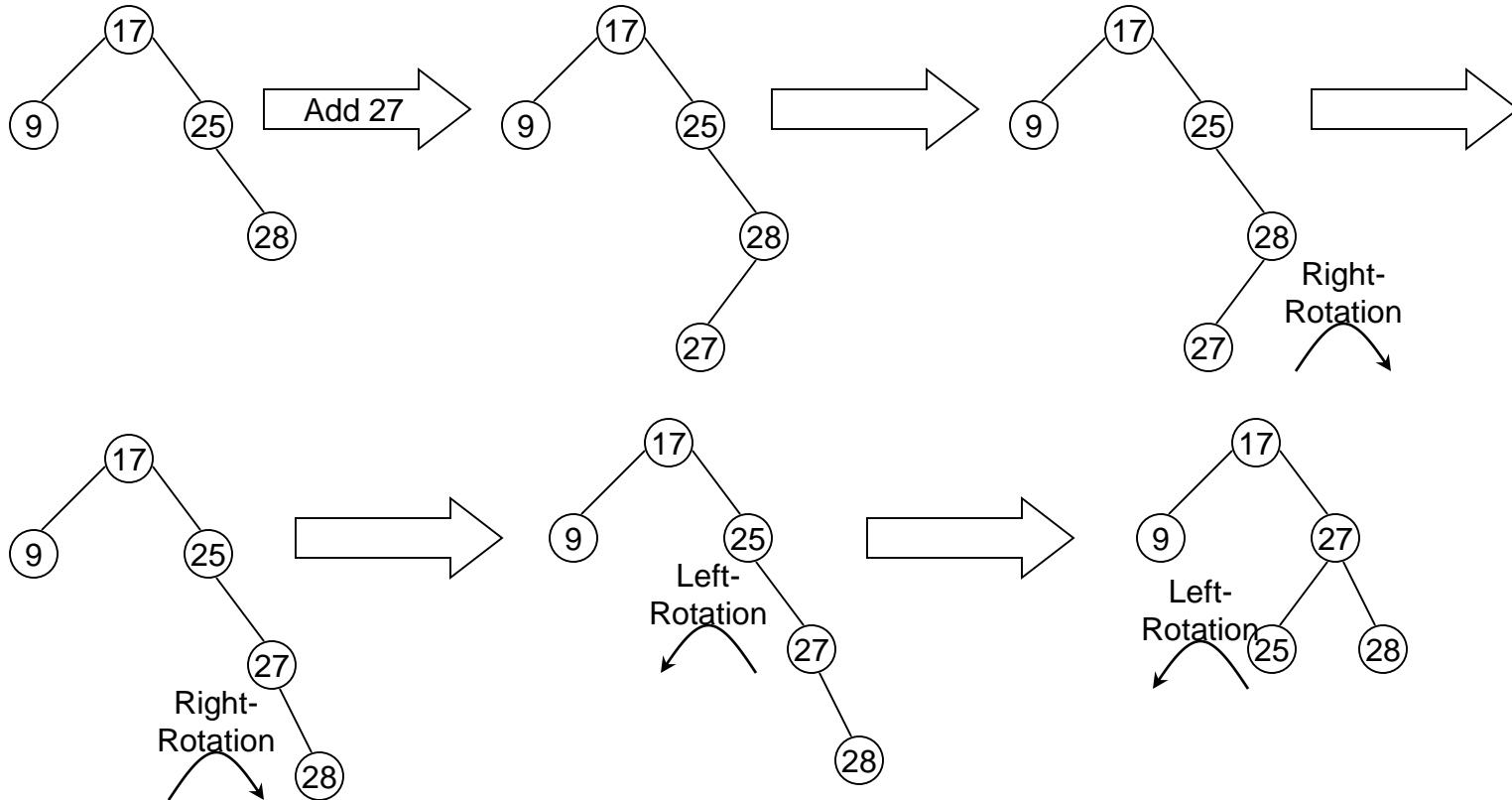
# Right-Left-Rotation Баруун-Зүүн-Эргүүлэлт

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

Solution: *Right-left* rotation



35



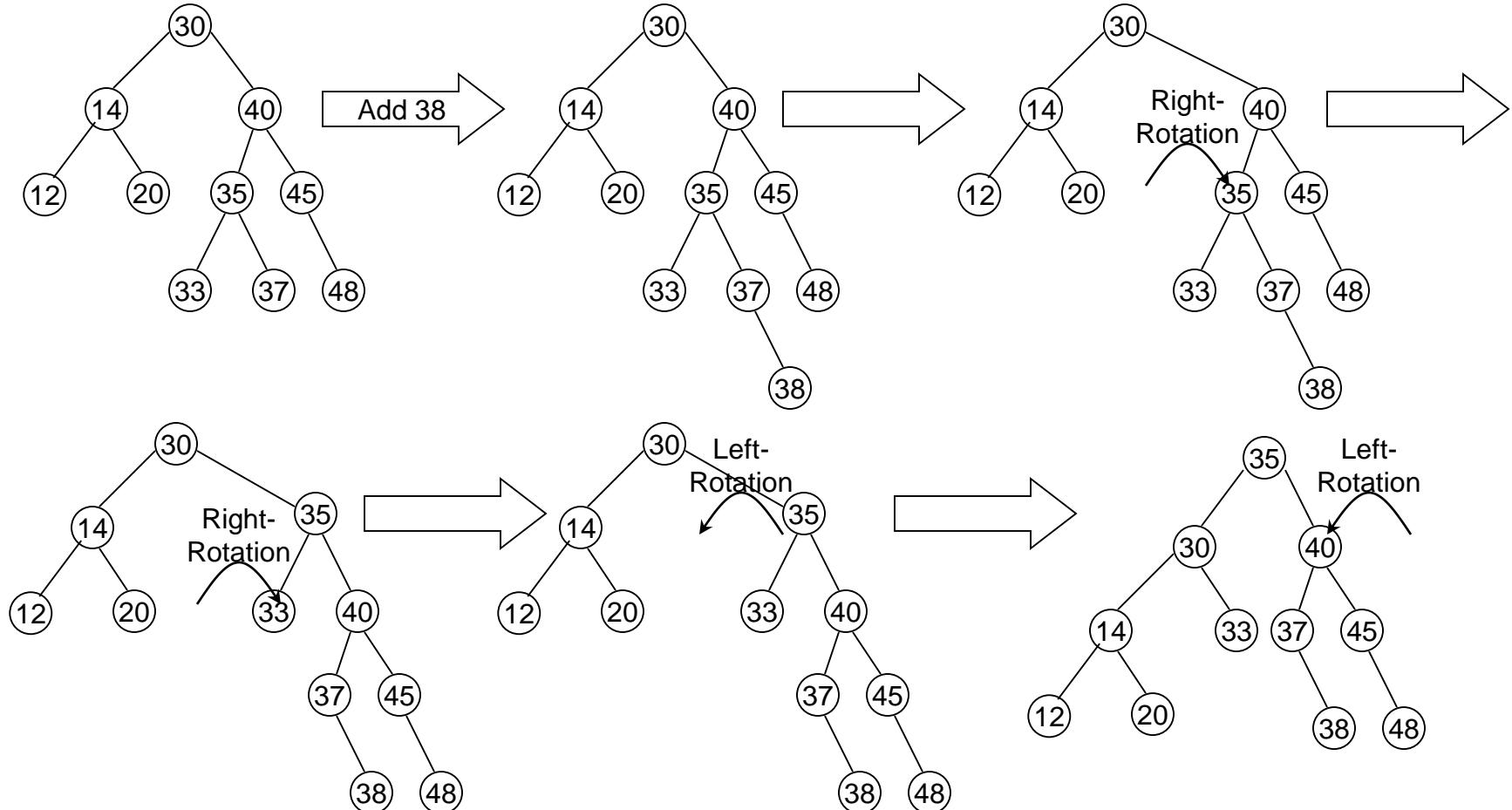
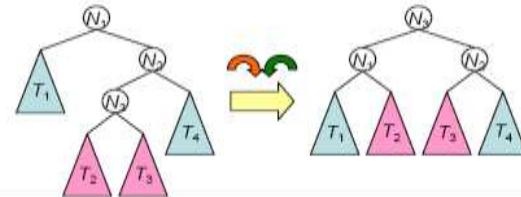
# Right-Left-Rotation

## Баруун-Зүүн-Эргүүлэлт

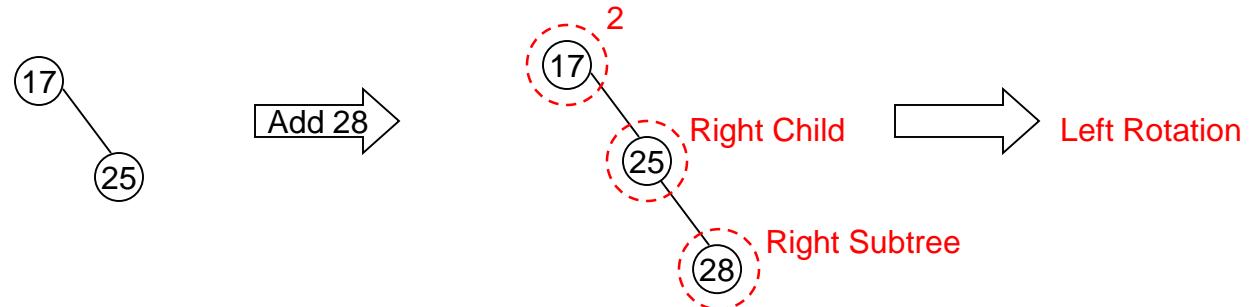
Case 4: insertion to *left*

subtree of *right* child

Solution: *Right-left* rotation

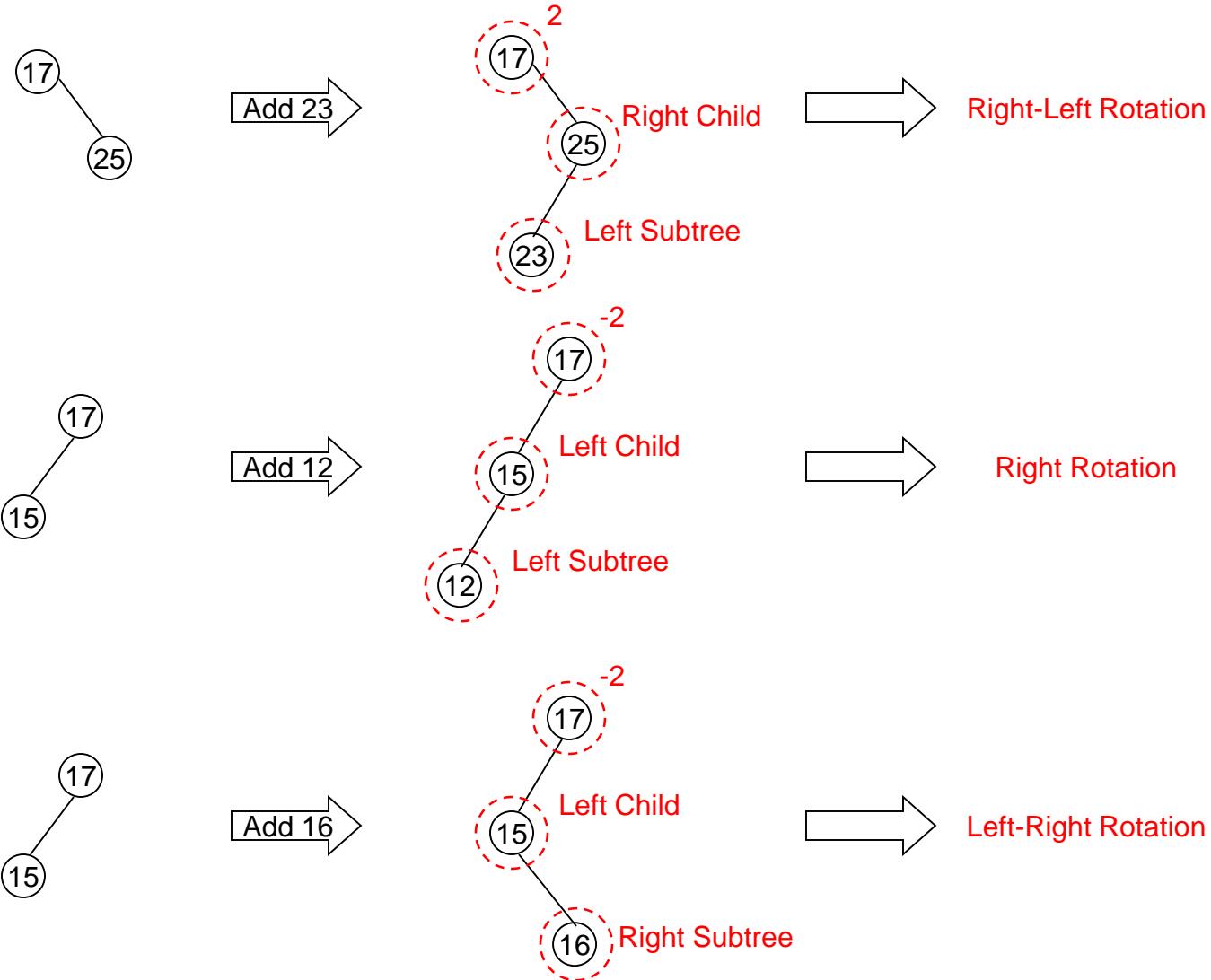


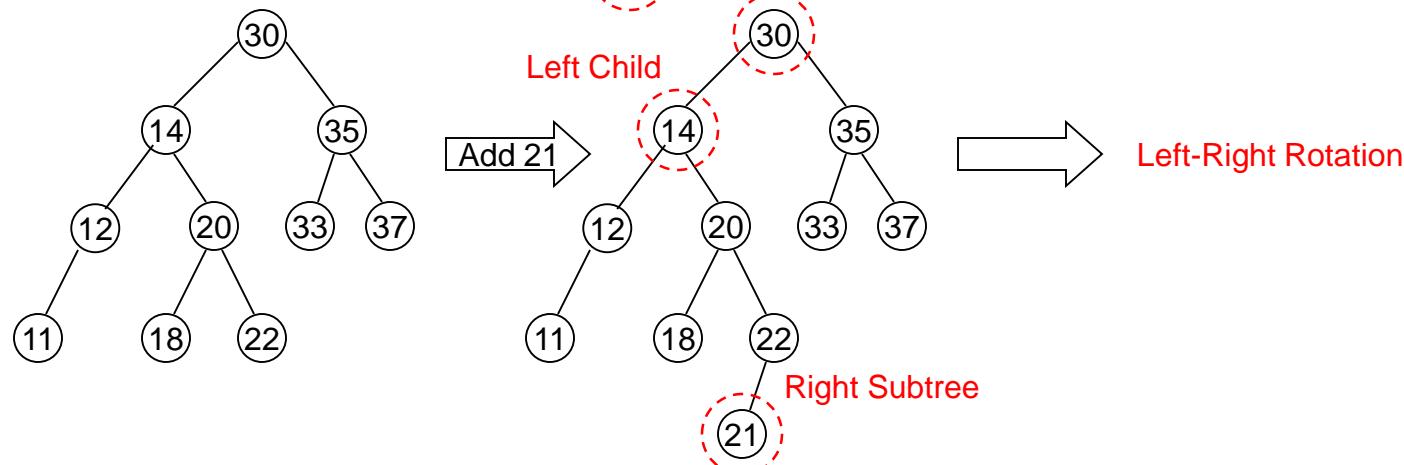
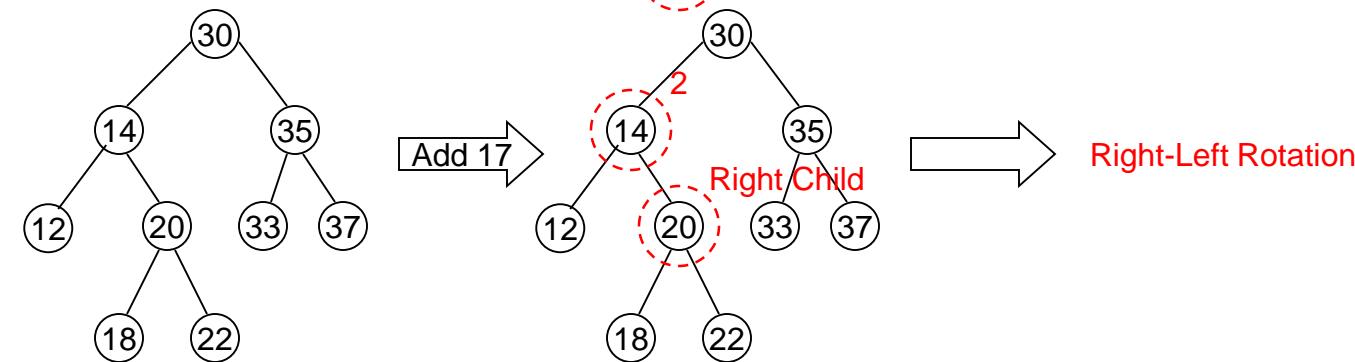
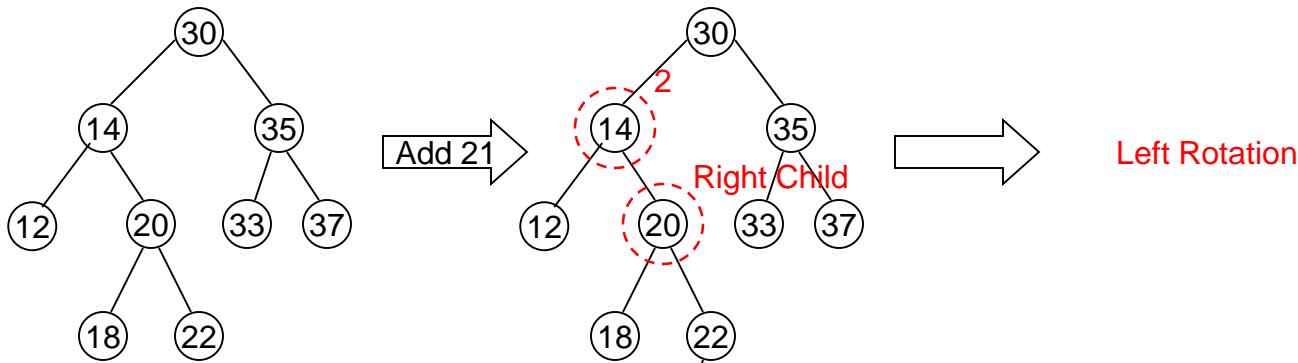
# Эргэлтийг хэрхэн тодорхойлох вэ?



- 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?



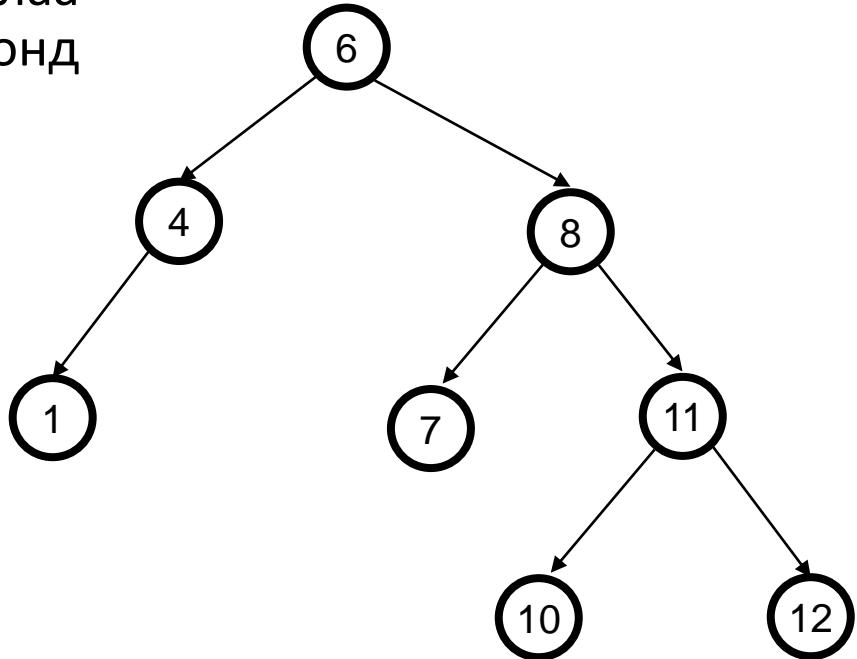


# AVL animation

- <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.

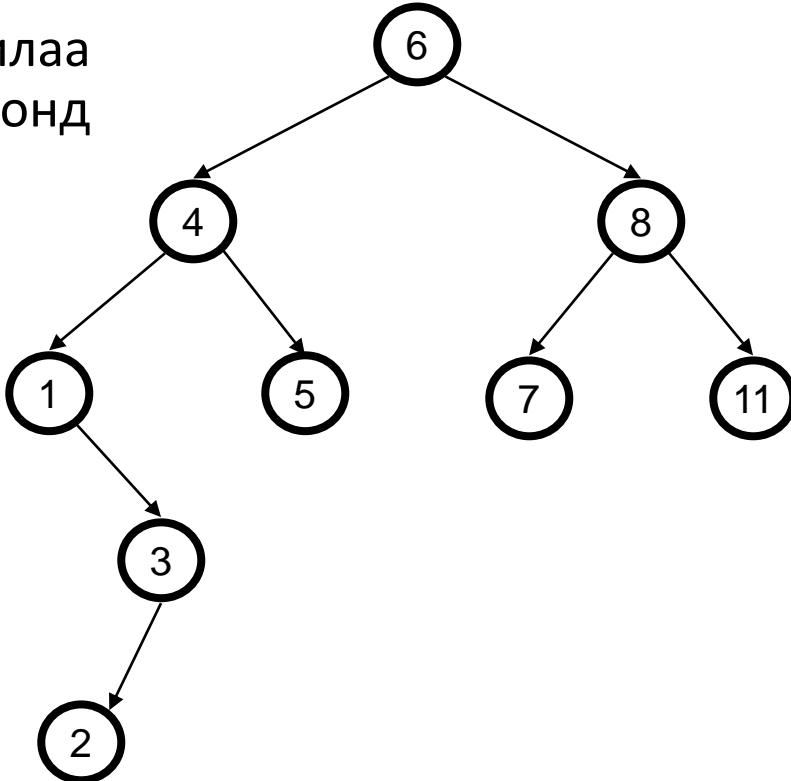


тэнцвэр(зангилаа) = өндөр(зангилаа.зүүн) – өндөр(зангилаа.баруун)

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

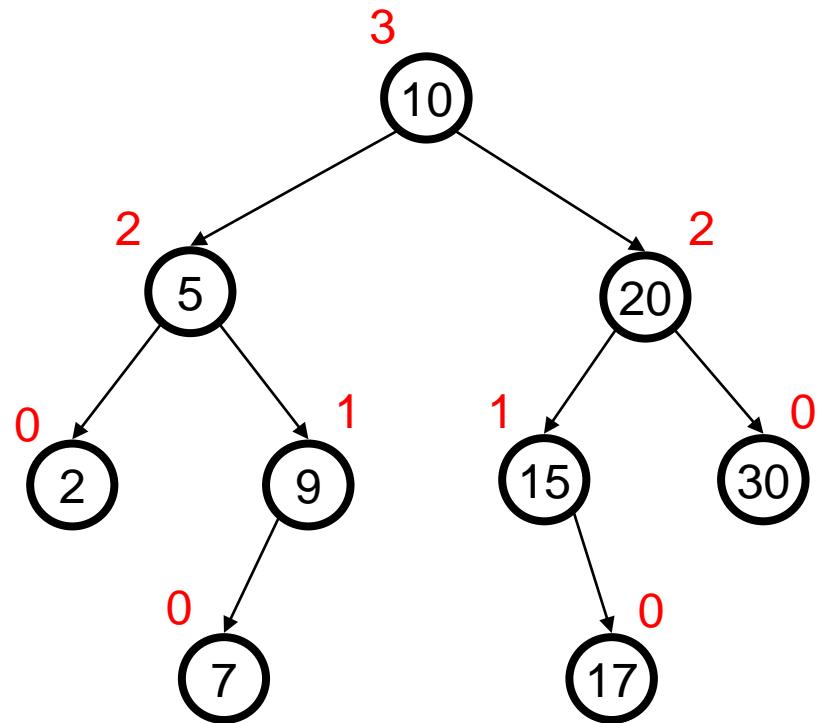
## Жишээ №2: Энэ AVL мод мөн үү?

**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)

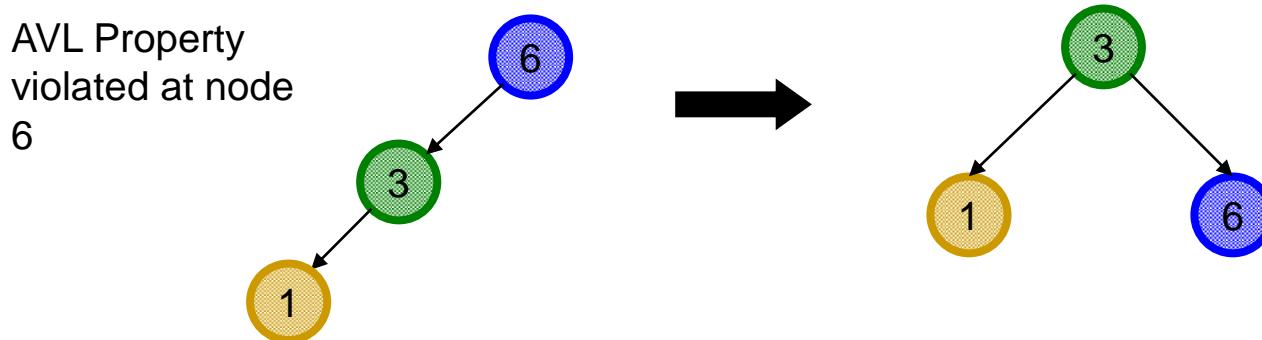
Insert(1)

Гурав дахь оруулга

Үүнийг засах цорын ганц арга зам юу вэ?

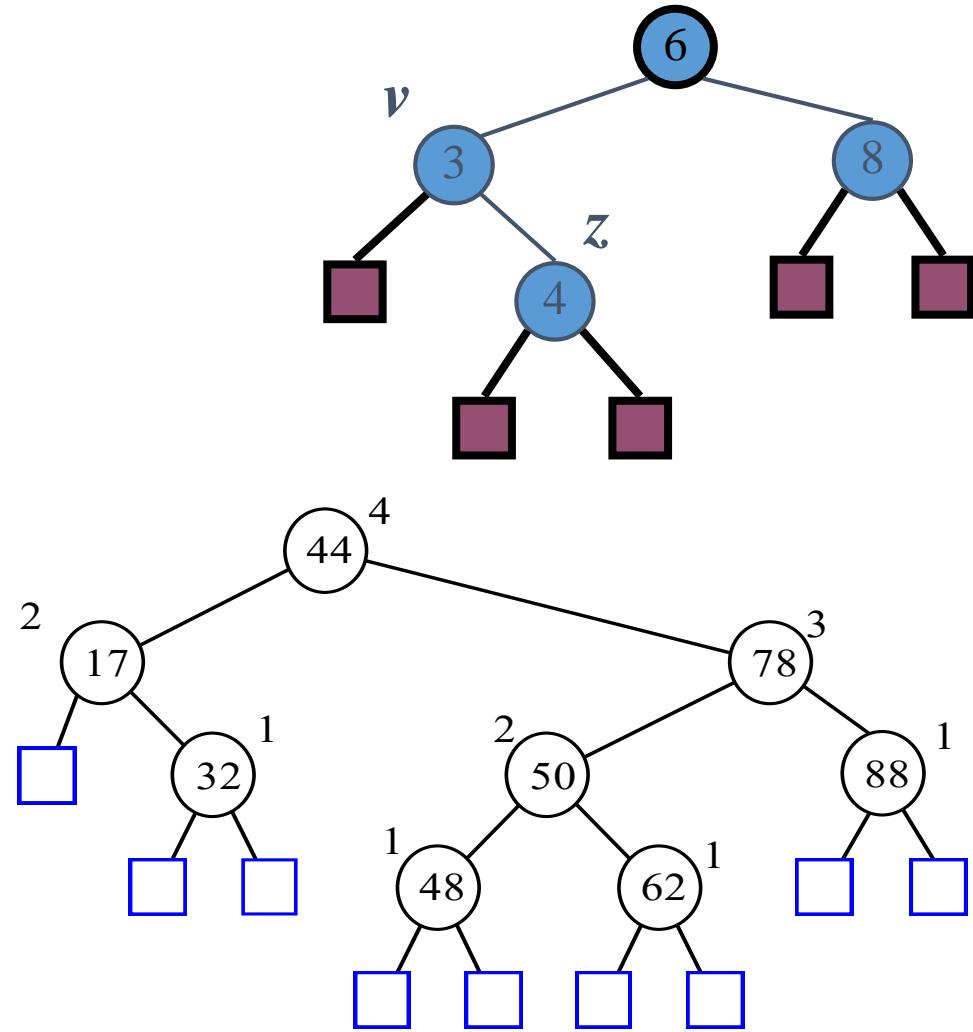
# Fix: Apply “Single Rotation”

- **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)



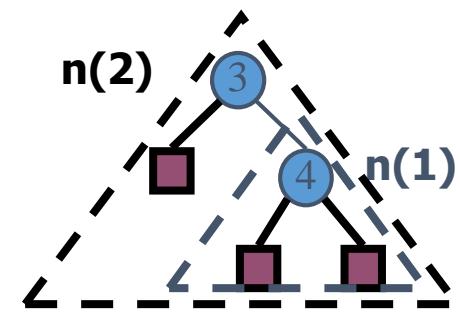
# AVL Tree Definition

- 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



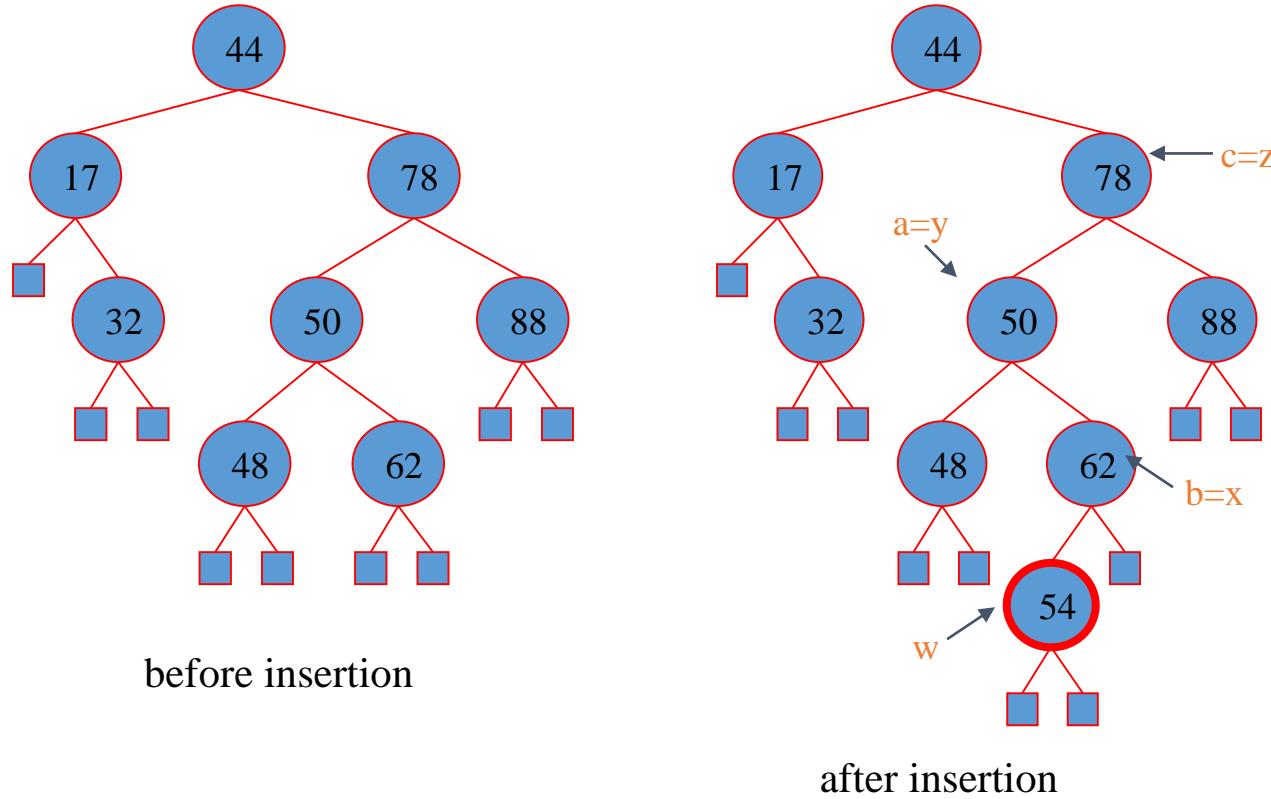
Fact: The height of an AVL tree storing  $n$  keys is  $O(\log n)$ .

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

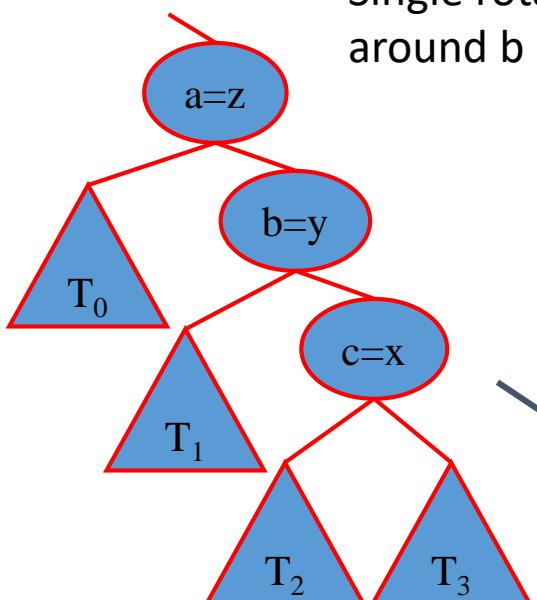
- Insertion is as in a binary search tree
- Always done by expanding an external node.
- Example:



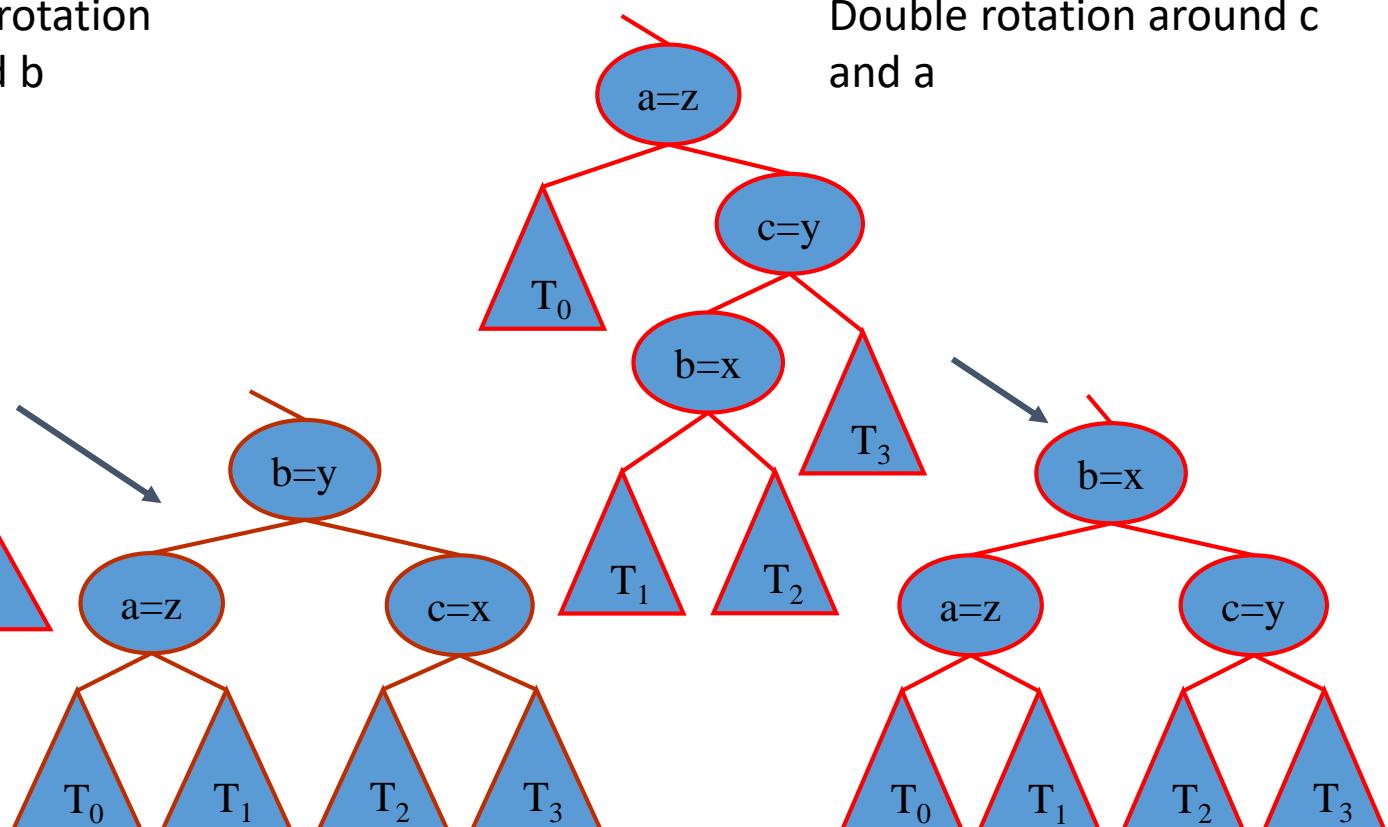
# Trinode Restructuring

- 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

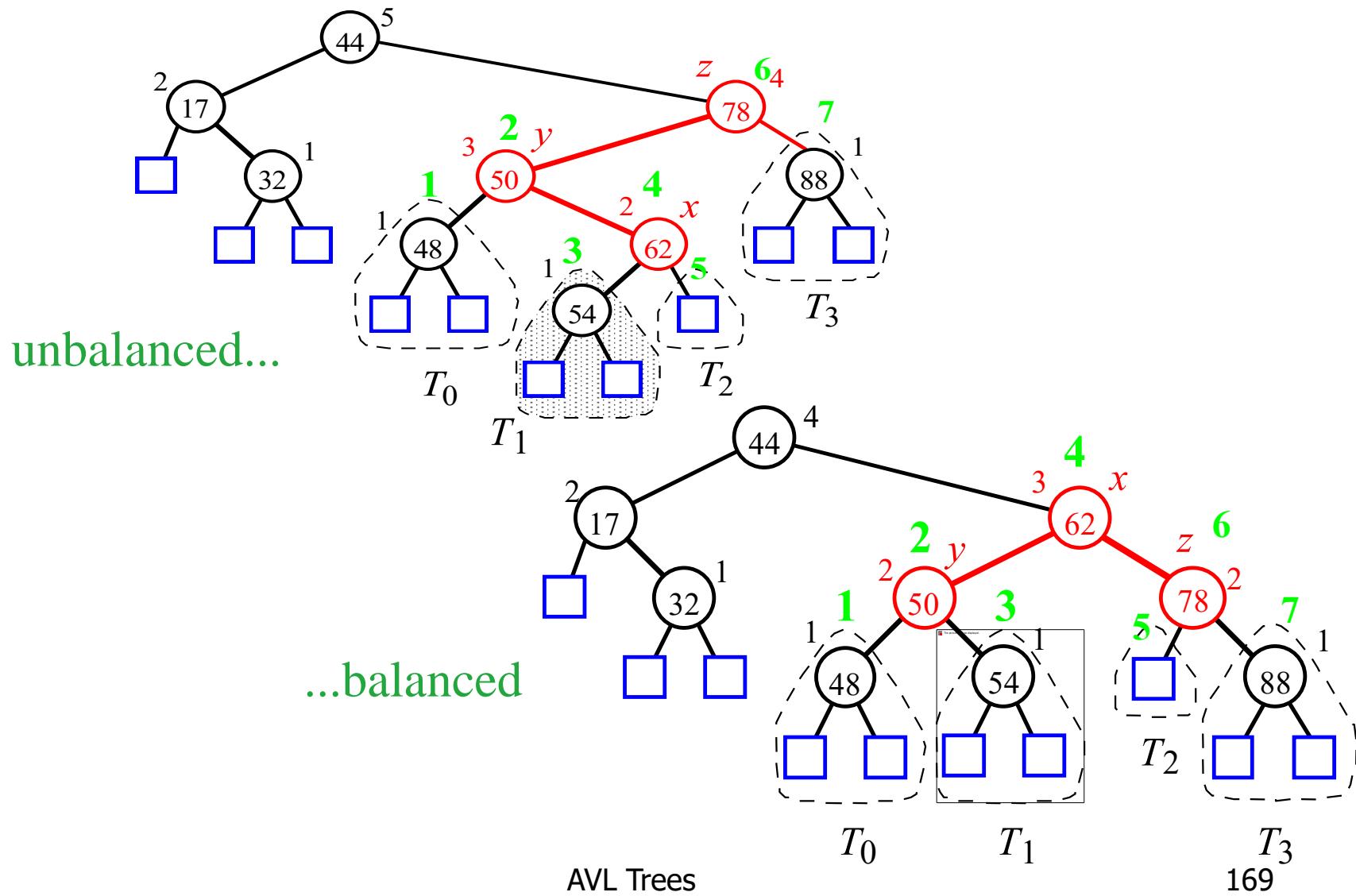
Single rotation  
around b



Double rotation around c  
and a

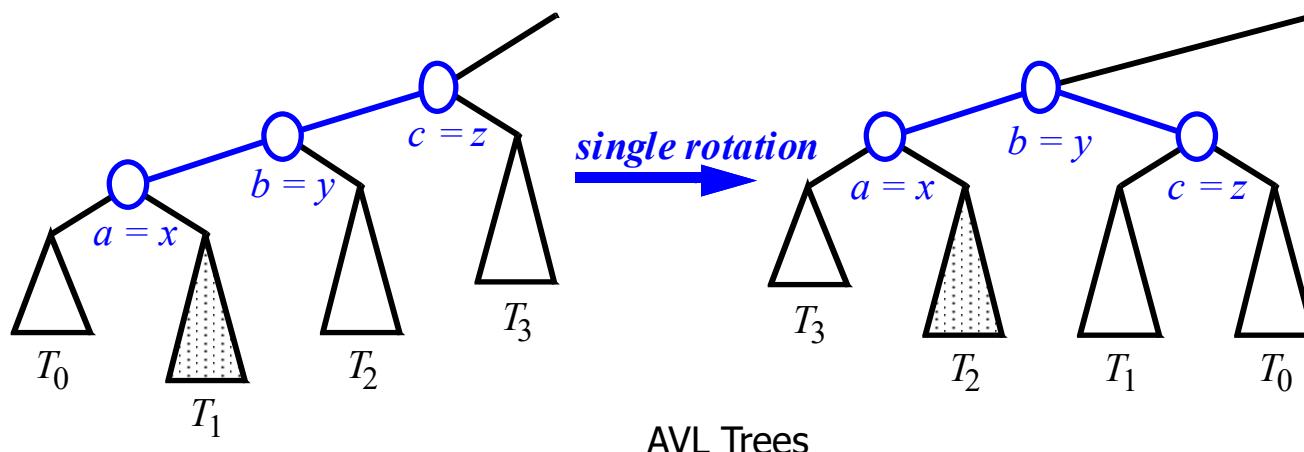
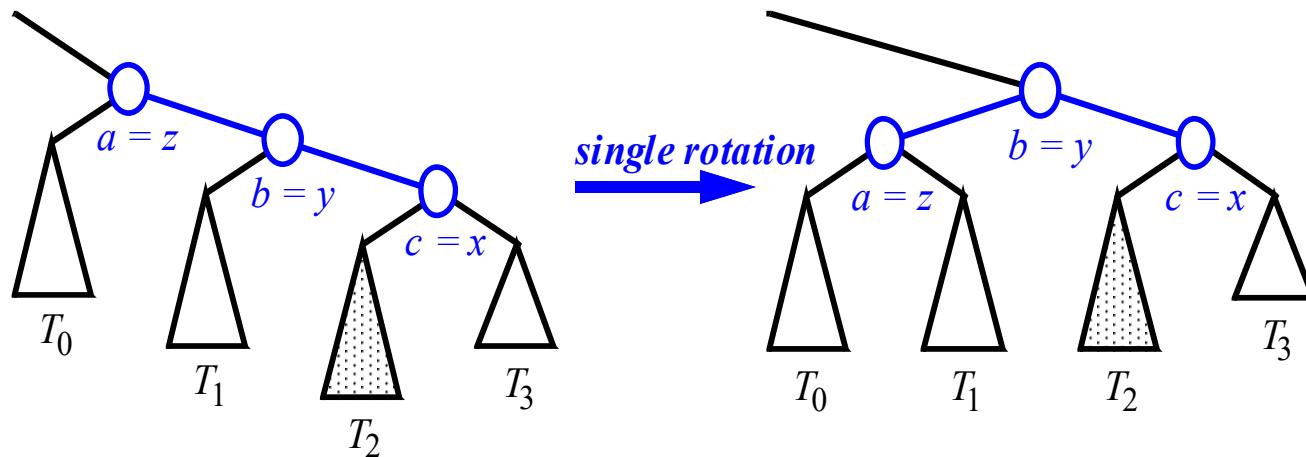


# Insertion Example, continued



# Restructuring (as Single Rotations)

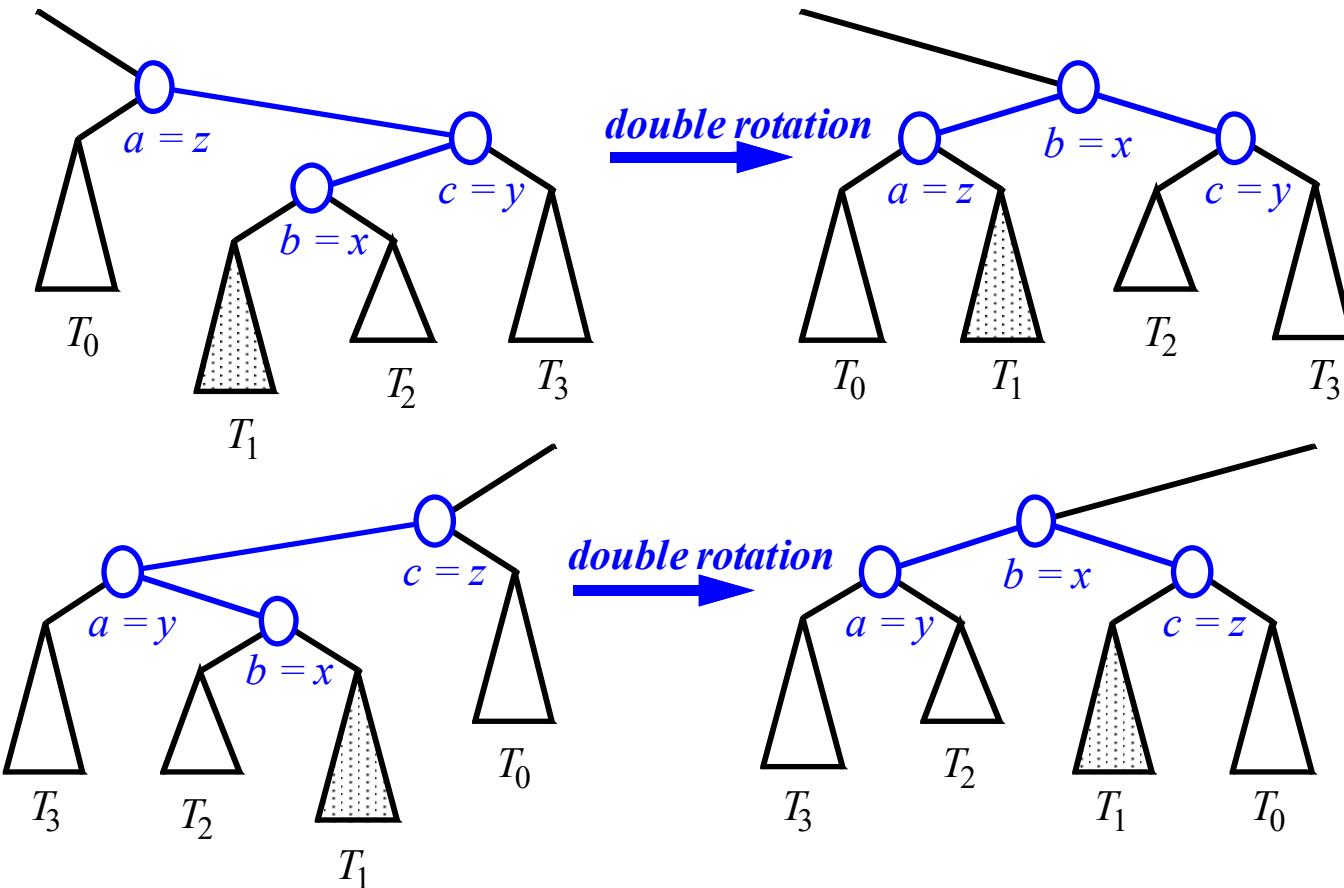
- Single Rotations:



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**

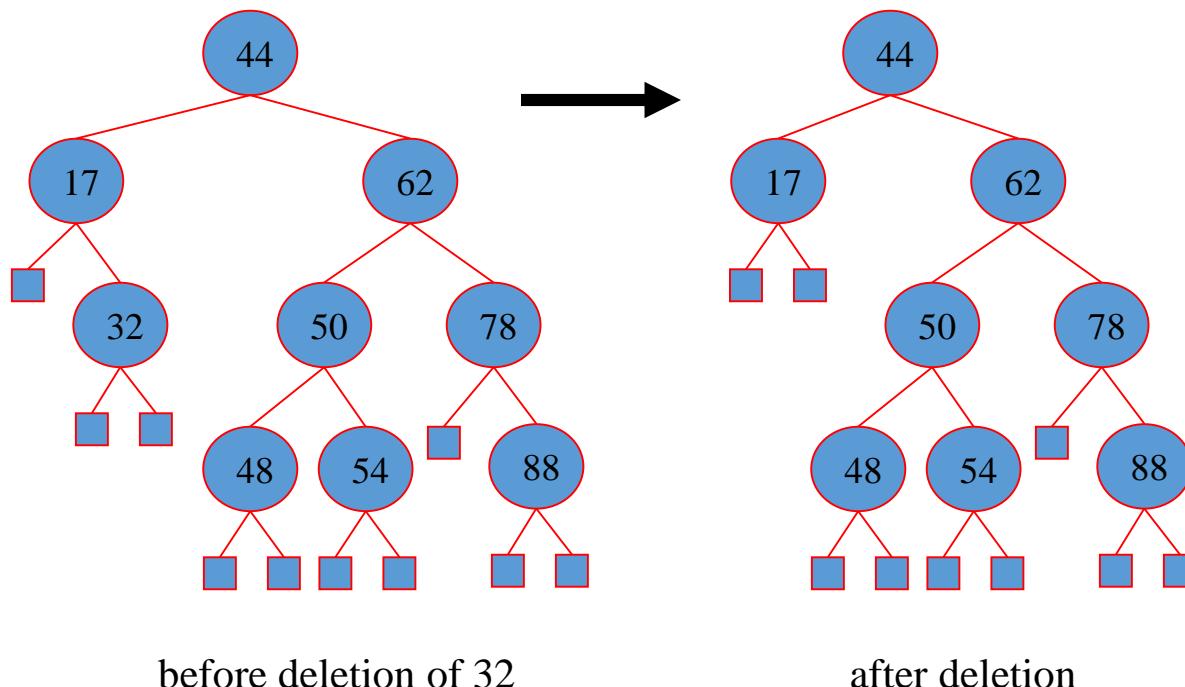
    Let  $y$  be the tallest child of  $v$  and let  $x$  be the tallest child of  $y$

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

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

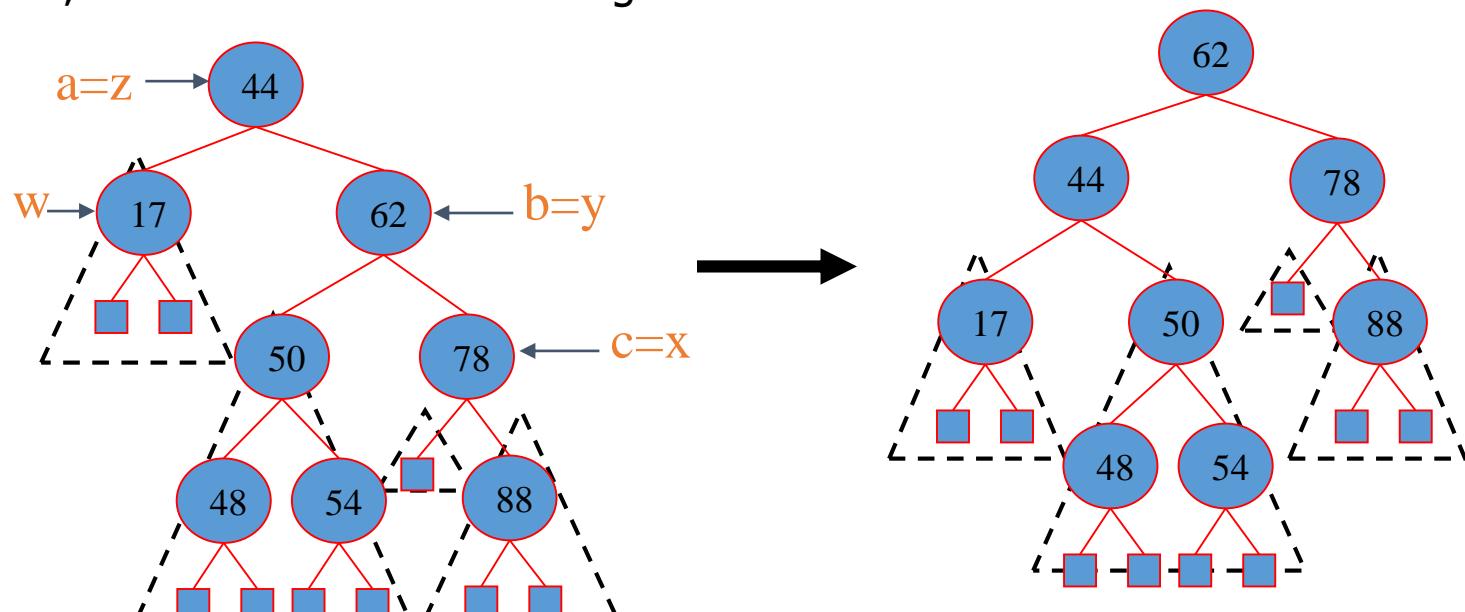
# Removal

- 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,  $z$

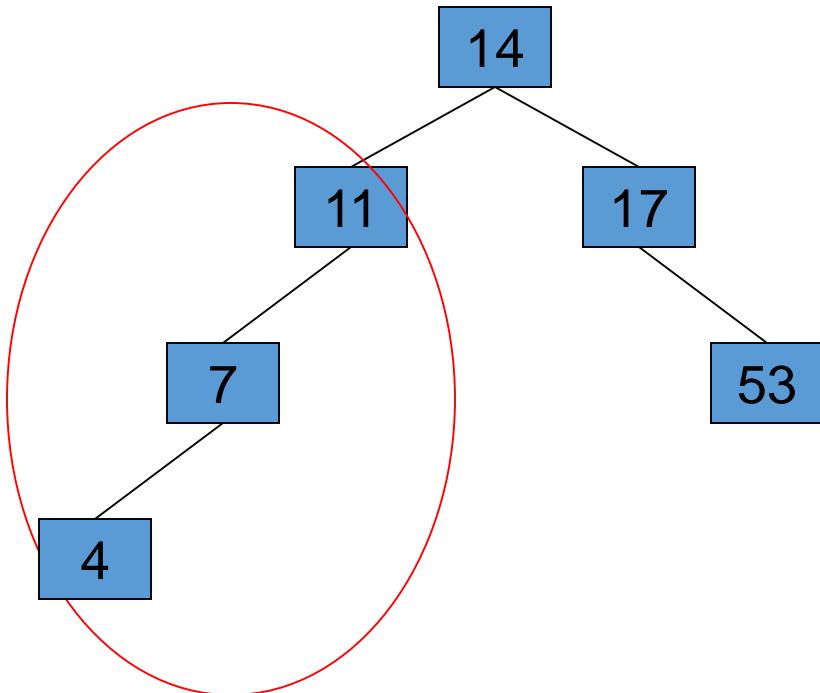
**rebalanceAVL**( $z, T$ )

# 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  $O(\log n)$

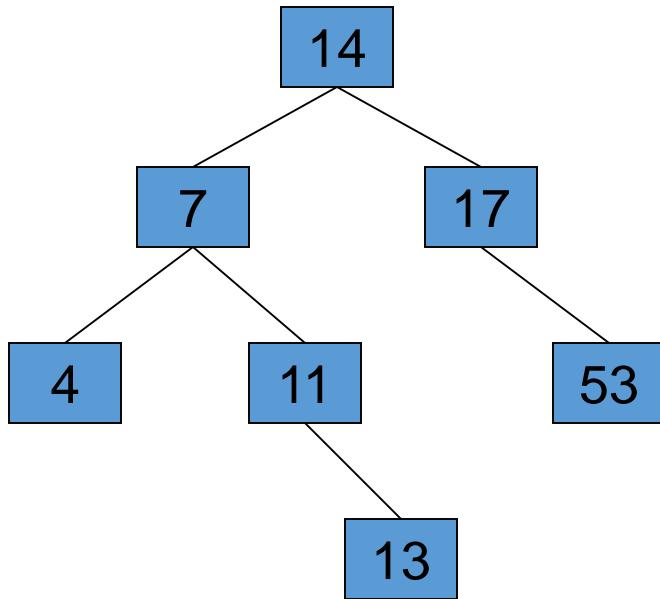
# AVL МОДНЫ ЖИШЭЭ

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



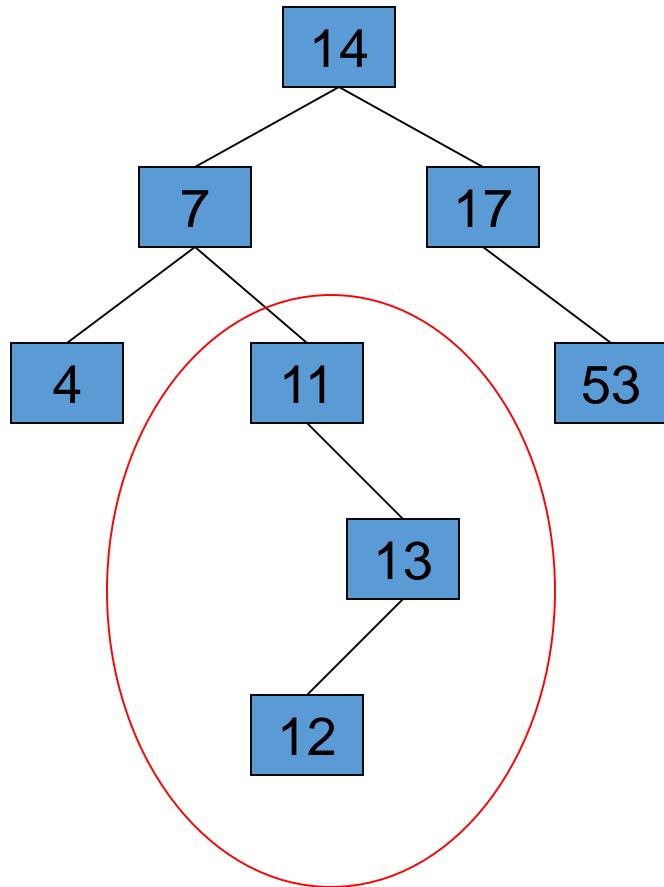
# AVL МОДНЫ ЖИШЭЭ

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



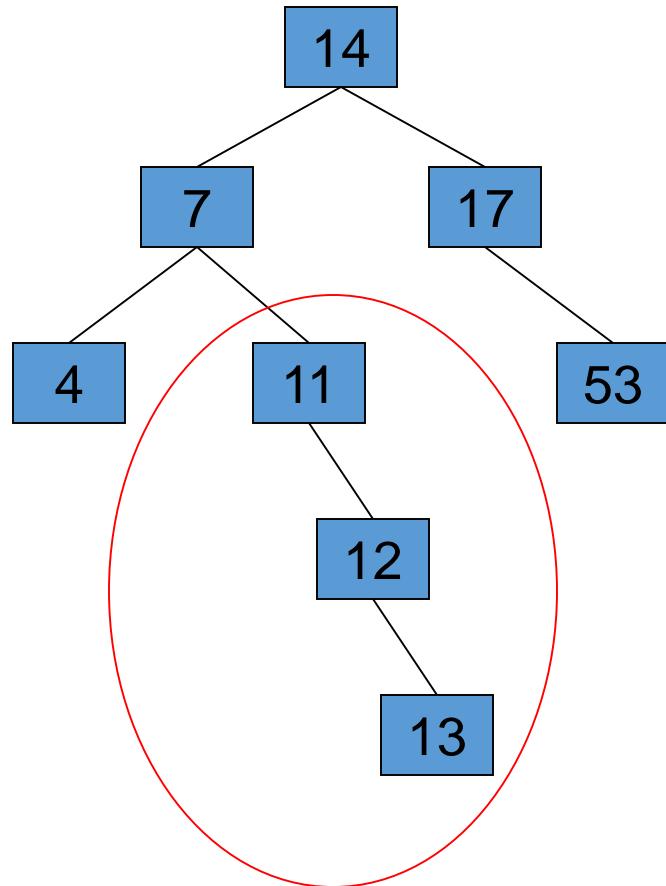
# AVL МОДНЫ ЖИШЭЭ

- Одоо 12-ыг оруулах



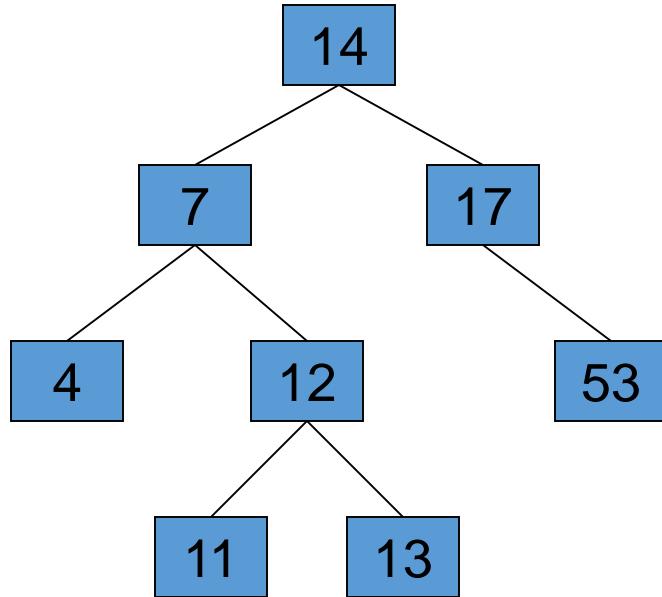
# AVL МОДНЫ ЖИШЭЭ

- Одоо 12-ыг оруулах



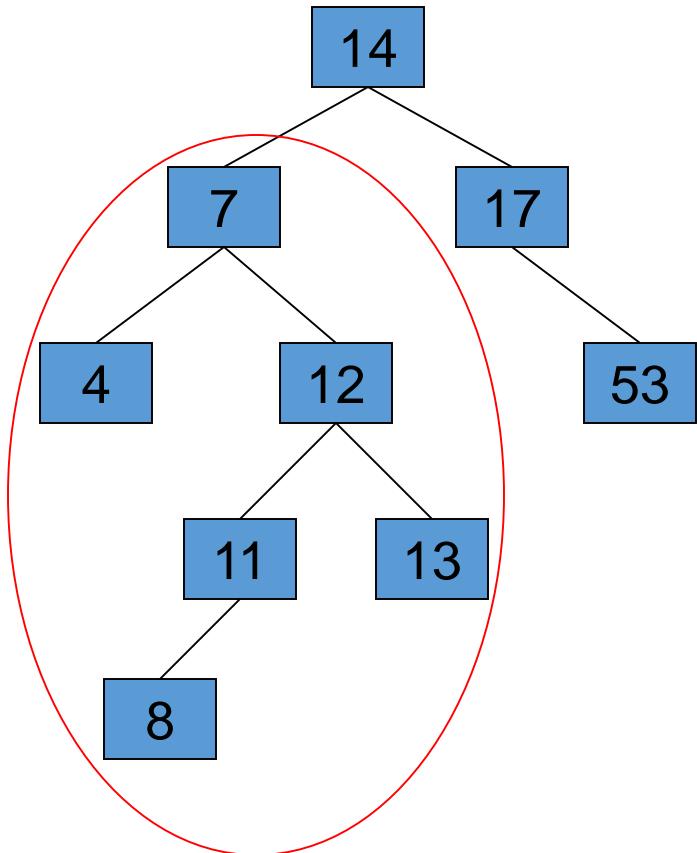
# AVL МОДНЫ ЖИШЭЭ

- Одоо AVL мод тэнцвэртэй байна.



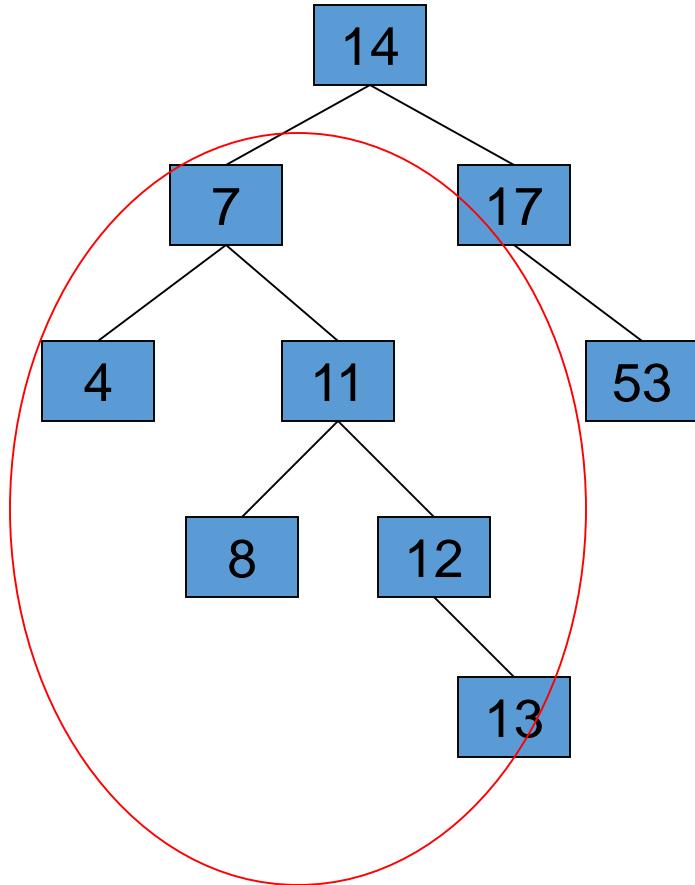
# AVL МОДНЫ ЖИШЭЭ

- Одоо 8-ыг оруулах



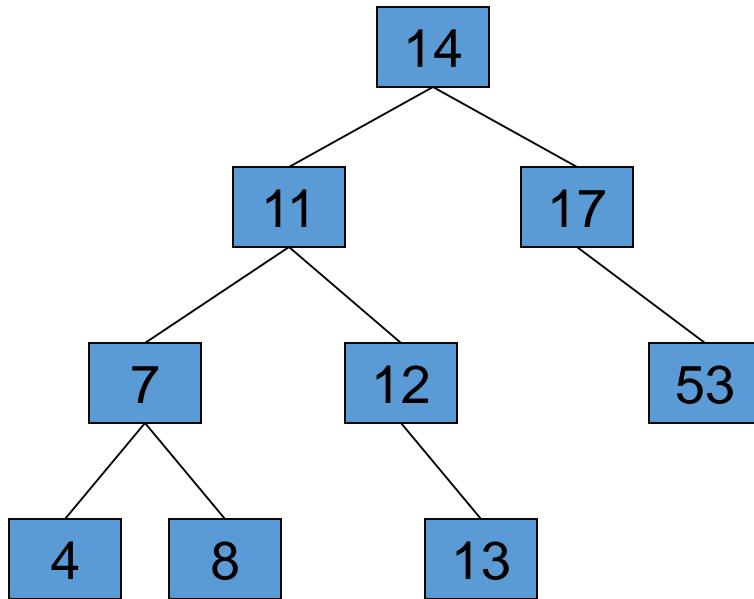
# AVL МОДНЫ ЖИШЭЭ

- Одоо 8-ыг оруулах



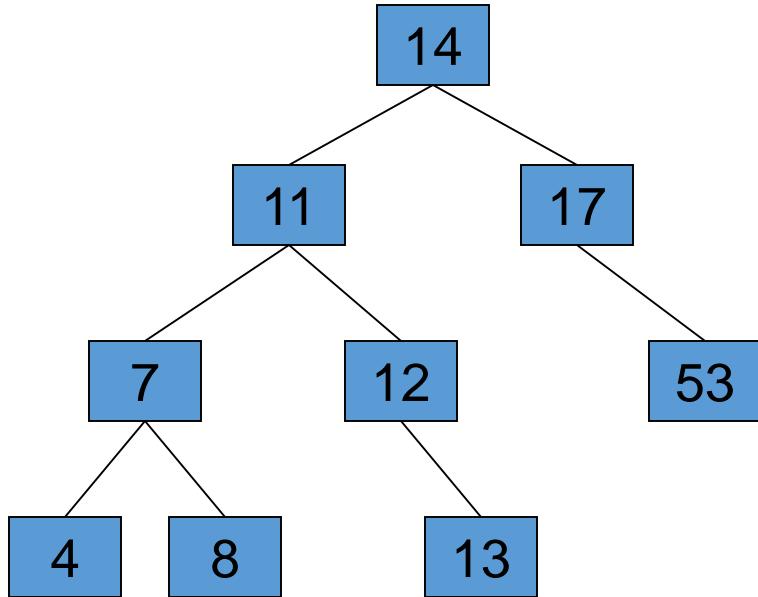
# AVL МОДНЫ ЖИШЭЭ

- Одоо AVL мод тэнцвэртэй байна.



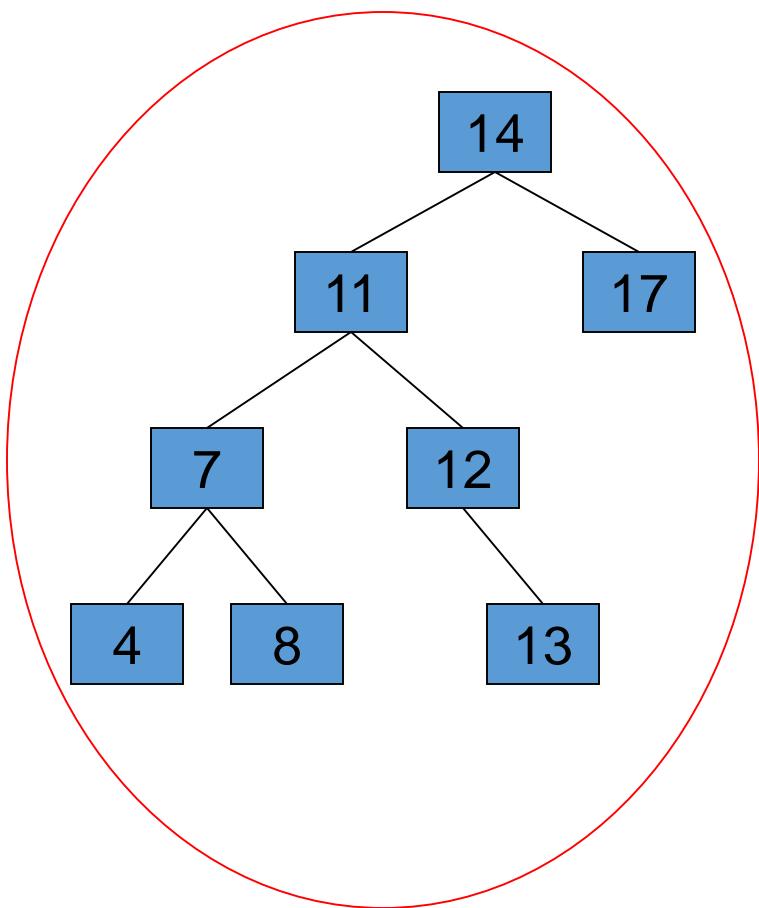
# AVL МОДНЫ ЖИШЭЭ

- Одоо 53-ыг устгах



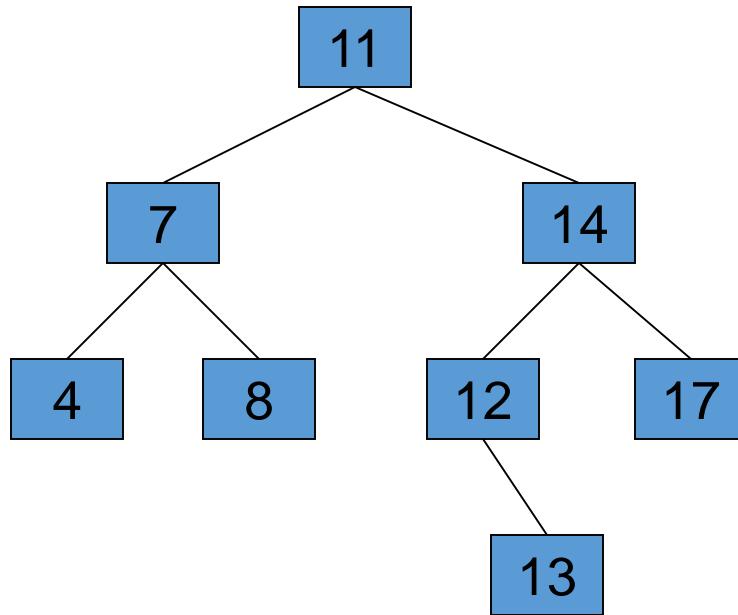
# AVL МОДНЫ ЖИШЭЭ

- Одоо 53-ыг устгасан ба тэнцвэргүй болсон.



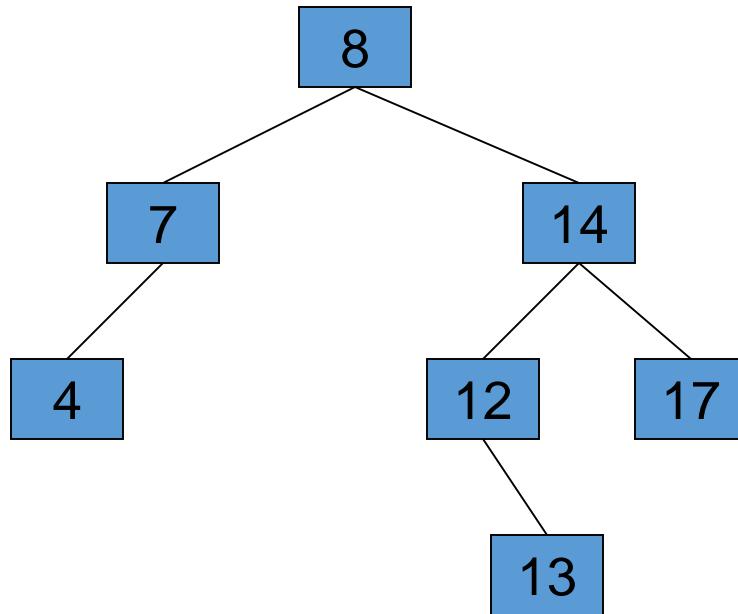
# AVL МОДНЫ ЖИШЭЭ

- Одоо 11-ийг устгаж тэнцвэртэй болгох.



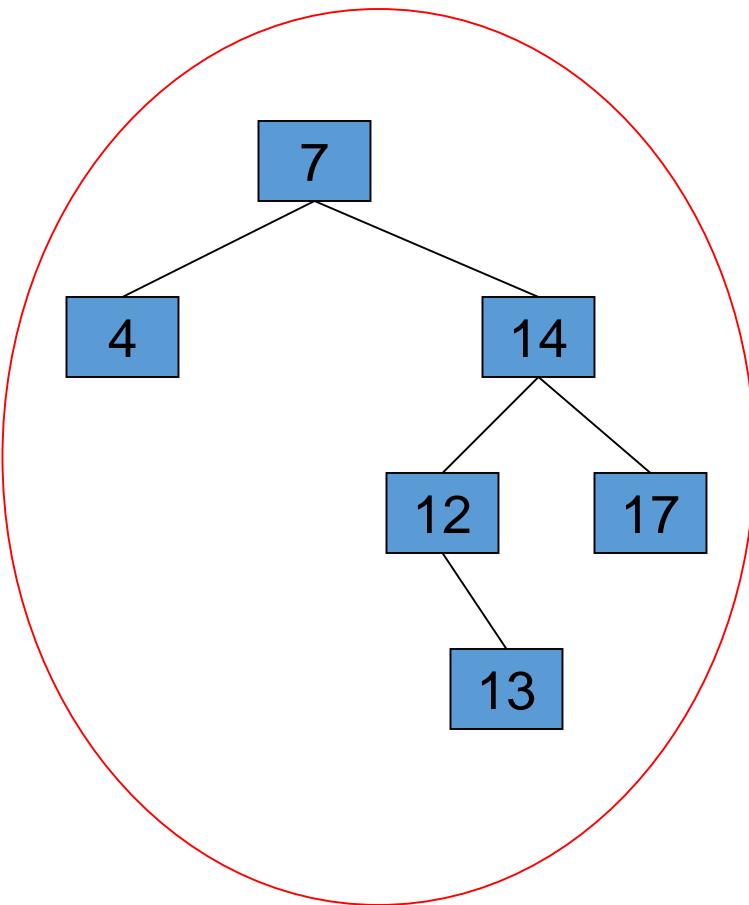
# AVL МОДНЫ ЖИШЭЭ

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



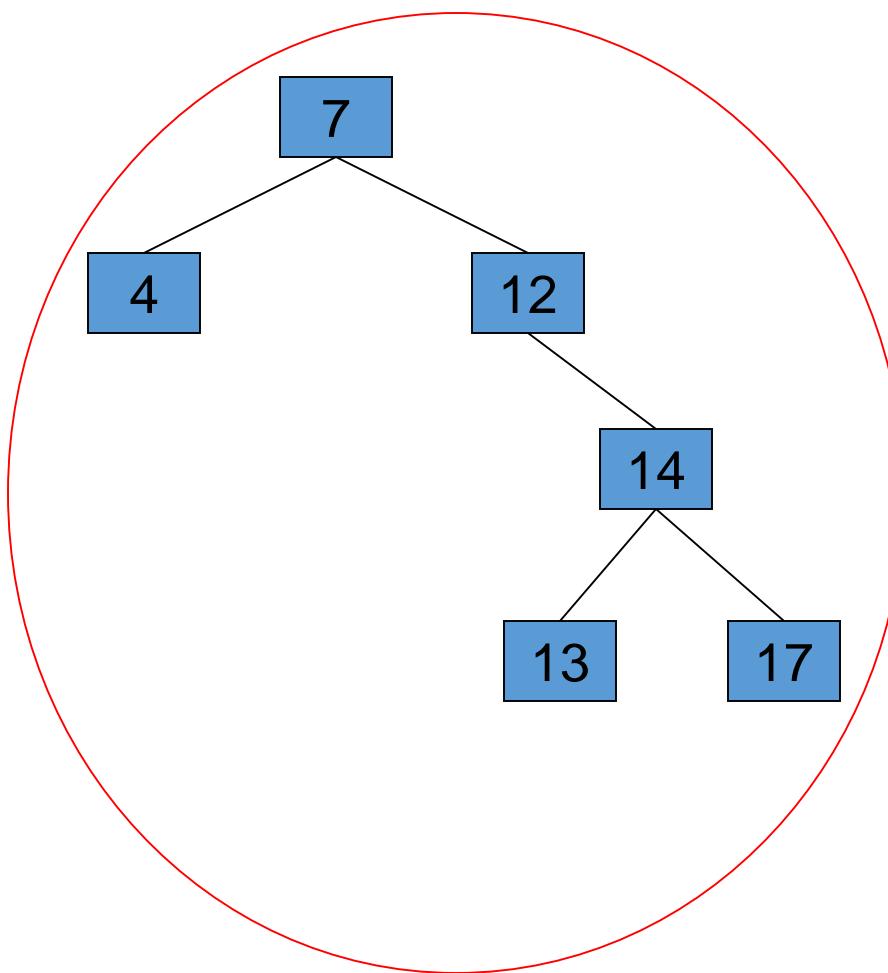
# AVL МОДНЫ ЖИШЭЭ

- Одоо 8-ыг устгасан ба тэнцвэргүй болсон.



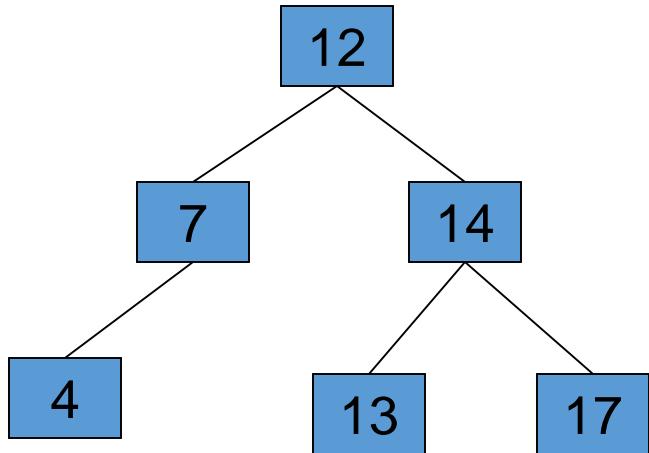
# AVL МОДНЫ ЖИШЭЭ

- Одоо 8-ыг устгасан ба тэнцвэргүй болсон.



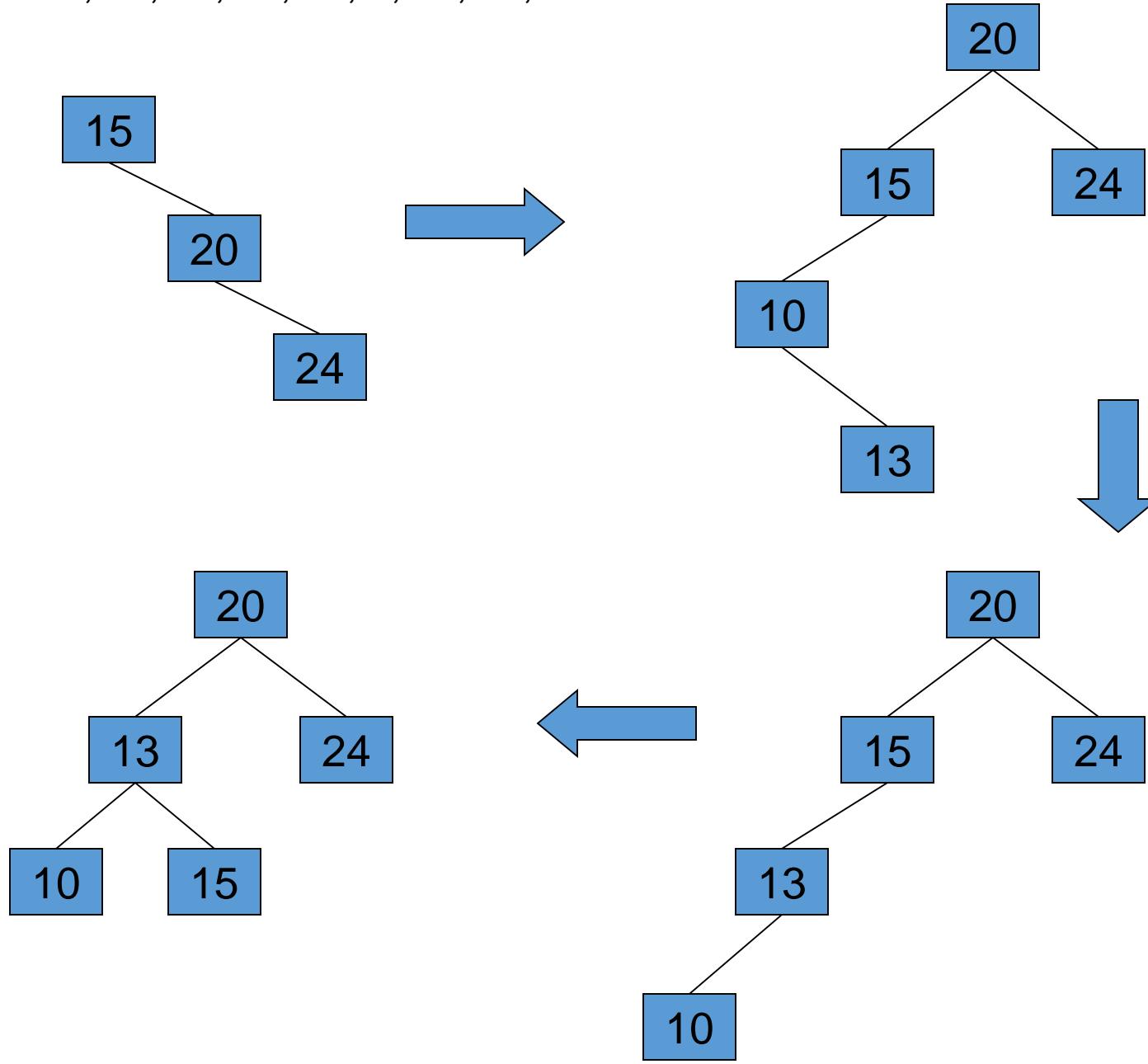
# AVL МОДНЫ ЖИШЭЭ

- Тэнцвэртэй !!!

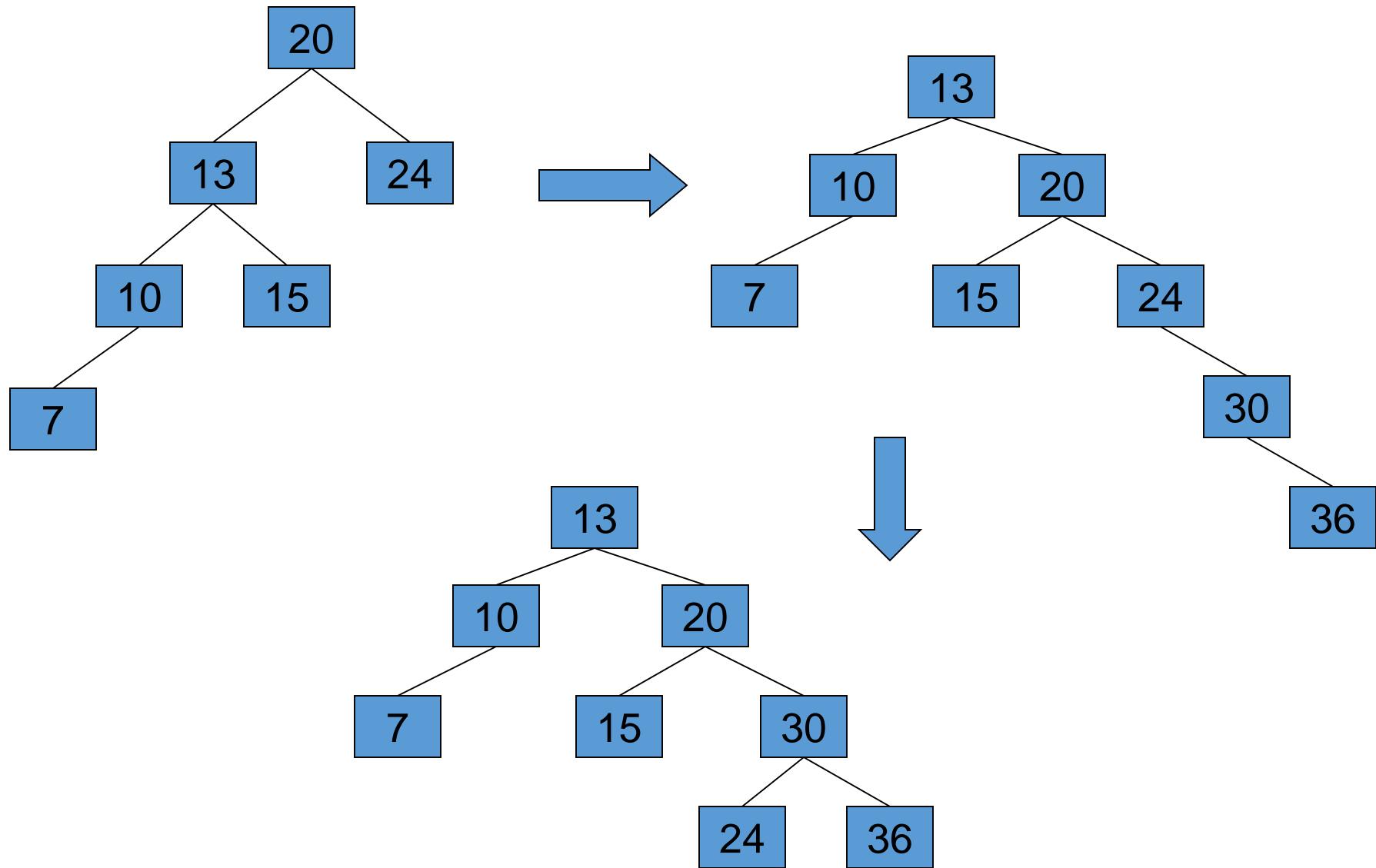


- Лабораторийн даалгавар: Дараах утгуудаар AVL модыг бүтээ. Үүнд: 15, 20, 24, 10, 13, 7, 30, 36, 25

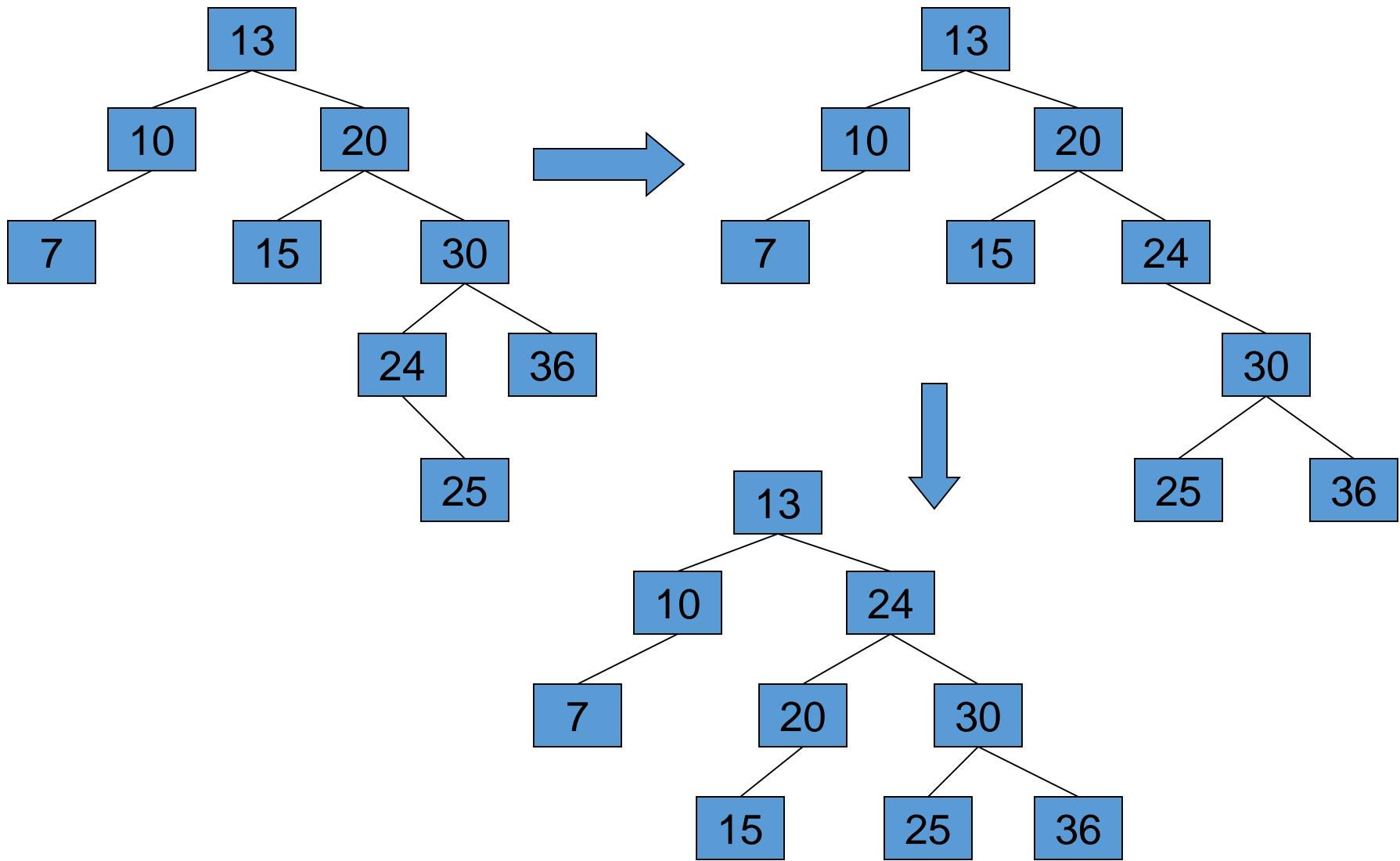
15, 20, 24, 10, 13, 7, 30, 36, 25



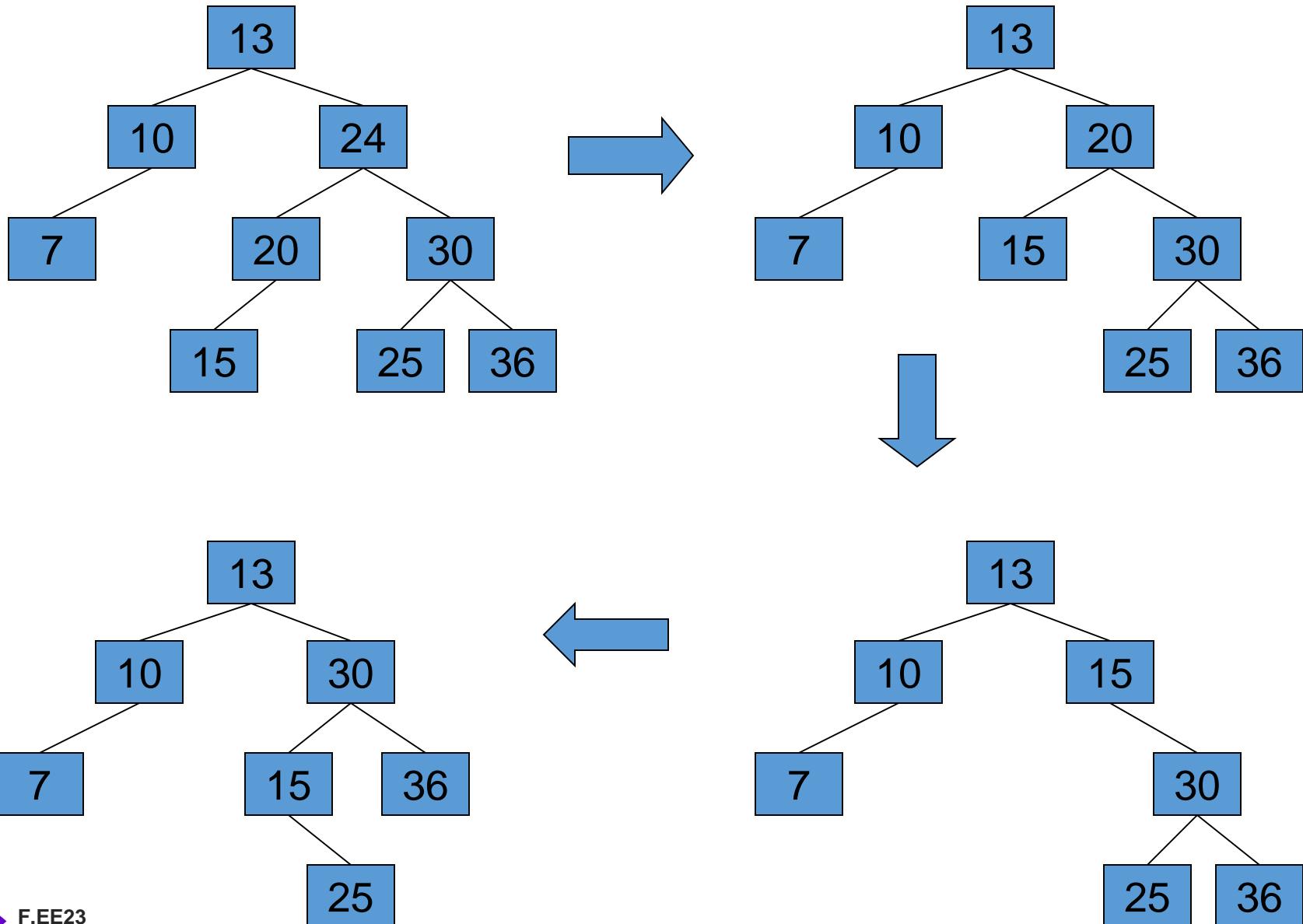
15, 20, 24, 10, 13, 7, 30, 36, 25



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

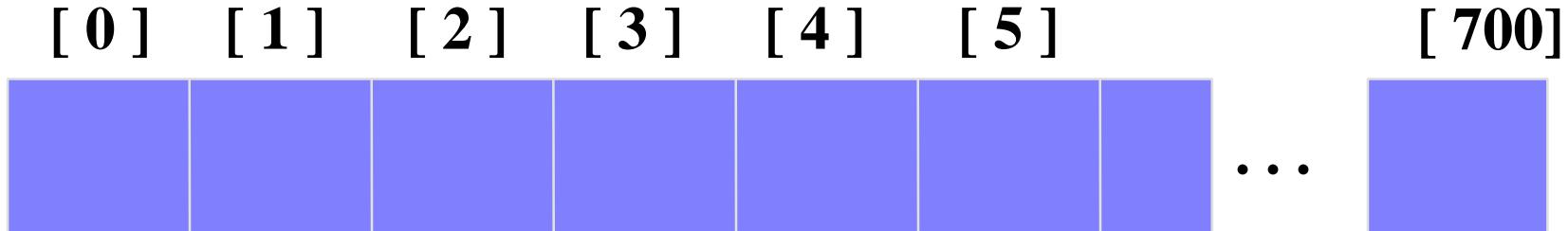


# AVL мадноос 24 ба 20-ыг утсга.



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

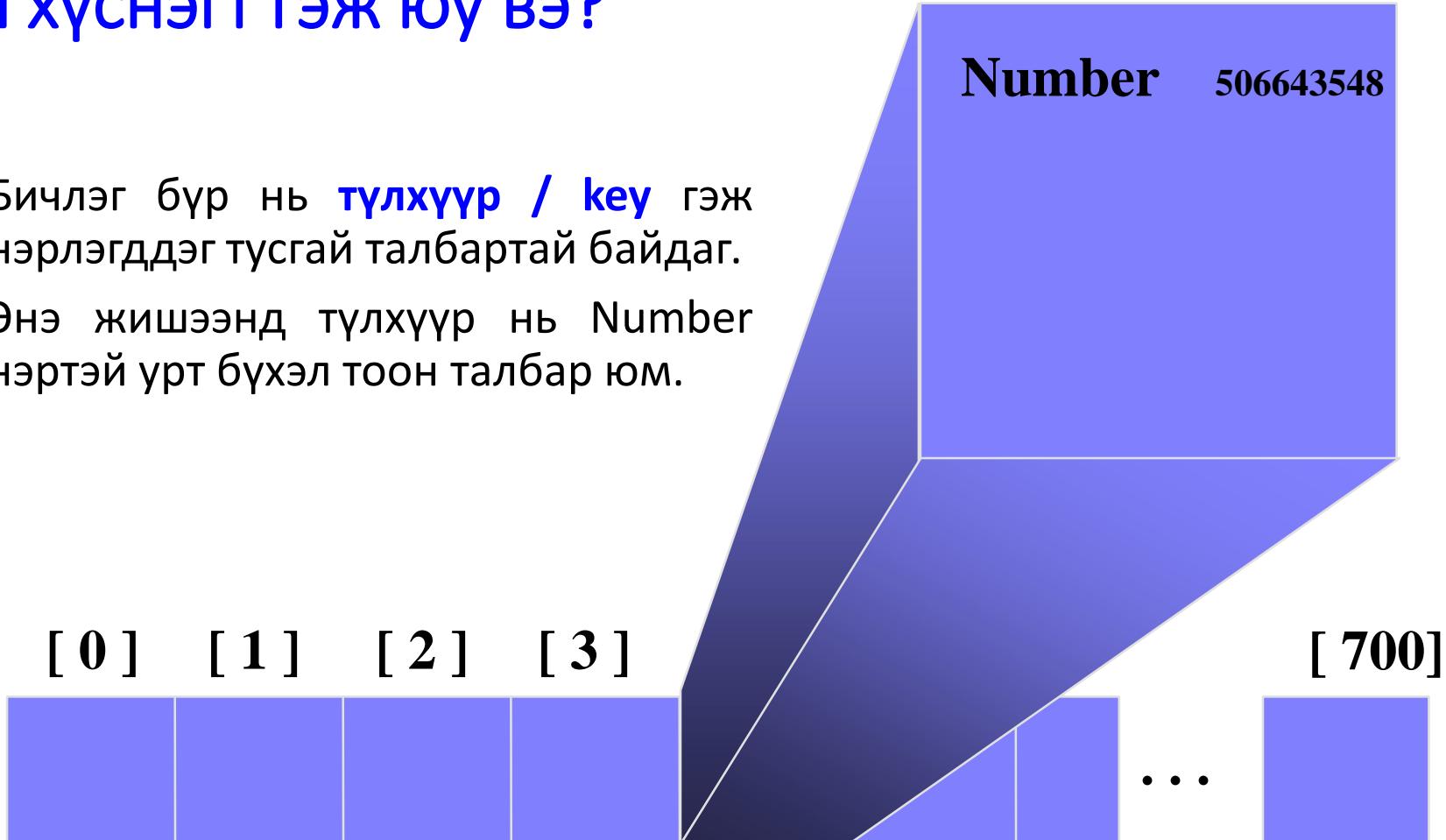
- Хэш хүснэгтийн хамгийн энгийн төрөл бол массив бичлэг юм.
- Энэ жишээнд 701 бичлэг байна.



**Бичлэгийн массив**

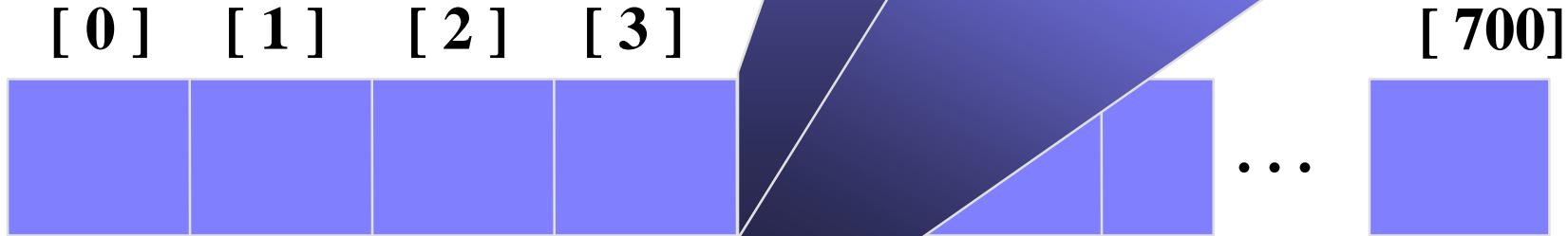
# Хэш хүснэгт гэж юу вэ?

- 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.



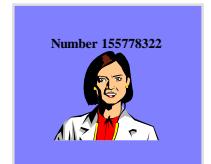
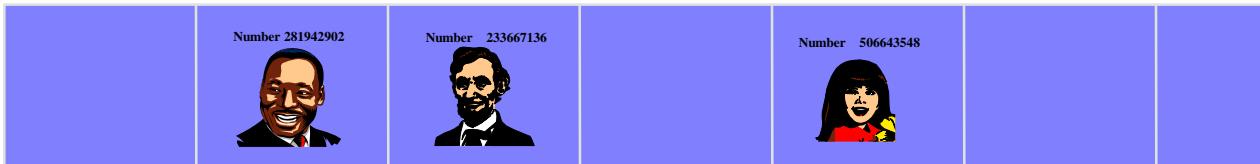
Number 506643548



# 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 ]

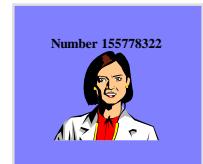
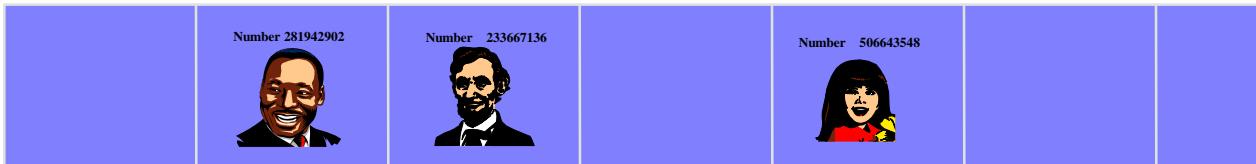


# Шинэ бичлэг нэмэх

- 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 ]



# Шинэ бичлэг нэмэх

- Хэш утгыг үүсгэх ердийн арга:



*(580625685 mod 701) гэж юу вэ?*

[ 0 ]

[ 1 ]

[ 2 ]

[ 3 ]

[ 4 ]

[ 5 ]

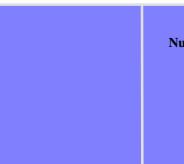
[ 700 ]



Number 281942902

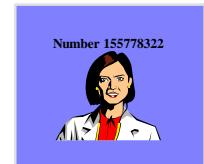


Number 233667136



Number 506643548

...



Number 155778322

# Шинэ бичлэг нэмэх

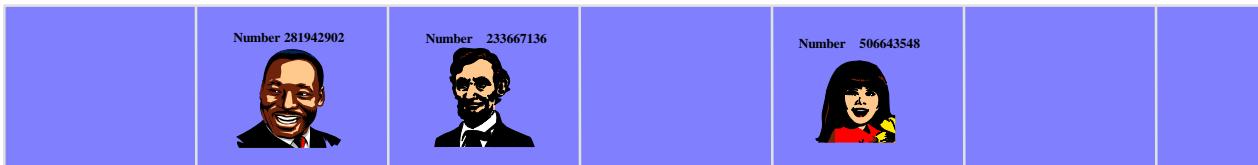
- Хэш утгыг үүсгэх ердийн арга:



*(580625685 mod 701) гэж юу вэ?*

3

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



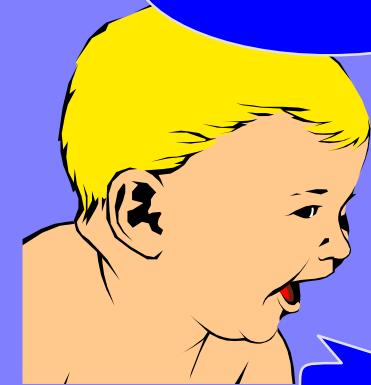
...



# 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



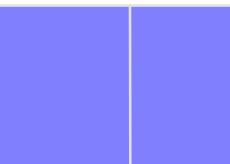
Number 233667136



Number 580625685



Number 506643548



...



Number 155778322

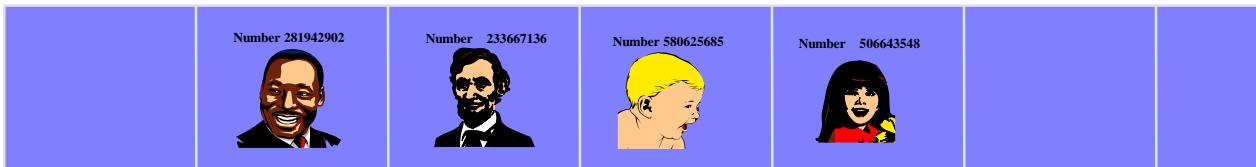
# Collisions / Мөргөлдөөн

- 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 ]



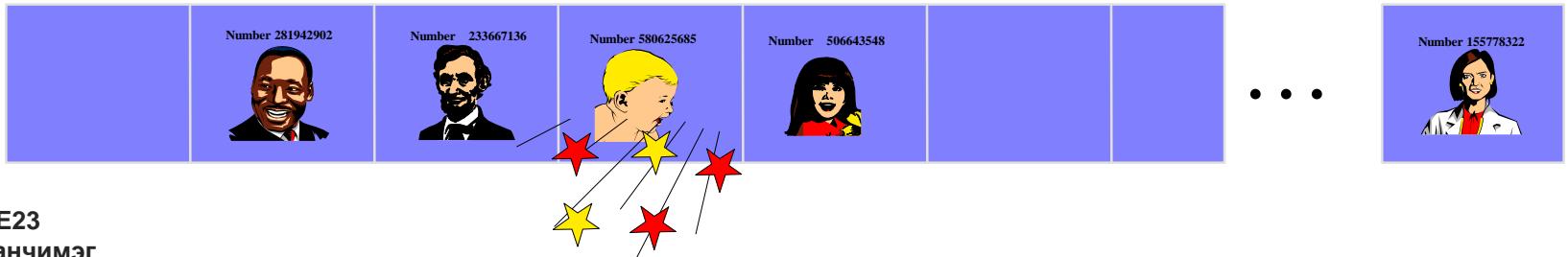
# Collisions / Мөргөлдөөн

- 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 ]



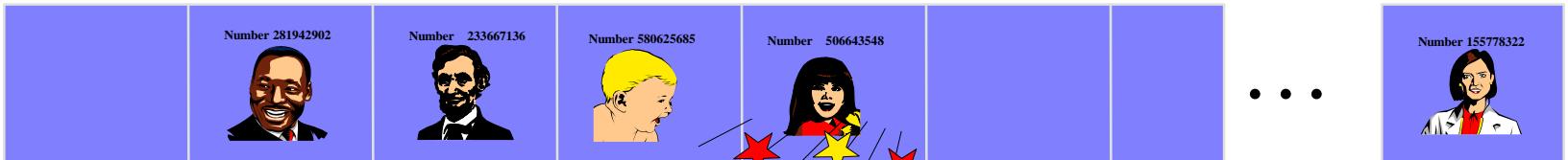
# Collisions / Мөргөлдөөн

- 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 ]

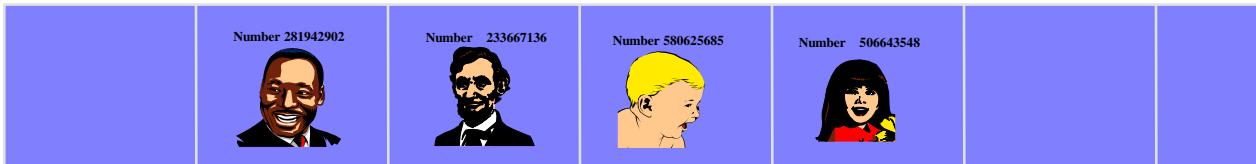


# Collisions / Мөргөлдөөн

- 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

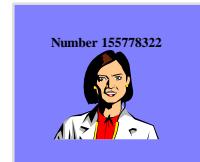


Number 506643548



Number 701466868

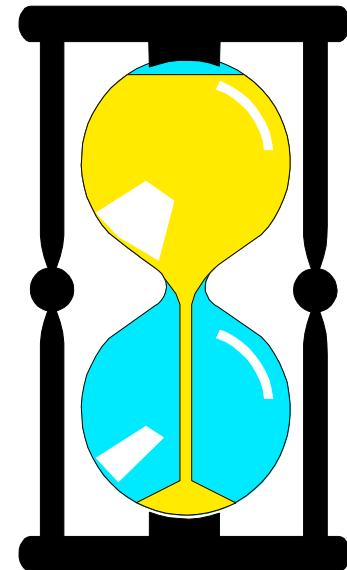
...



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

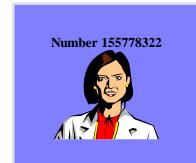


Number 506643548



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



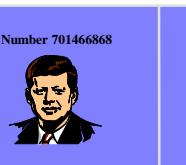
Number 233667136



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 ]

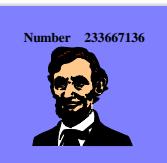
[ 4 ]

[ 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 ]

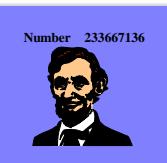
[ 4 ]

[ 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 ]

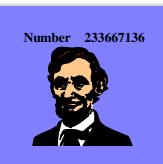
[ 4 ]

[ 5 ]

[ 700 ]



Number 281942902



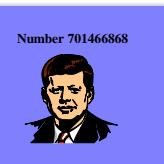
Number 233667136



Number 580625685

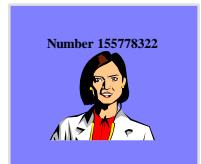


Number 506643548



Number 701466868

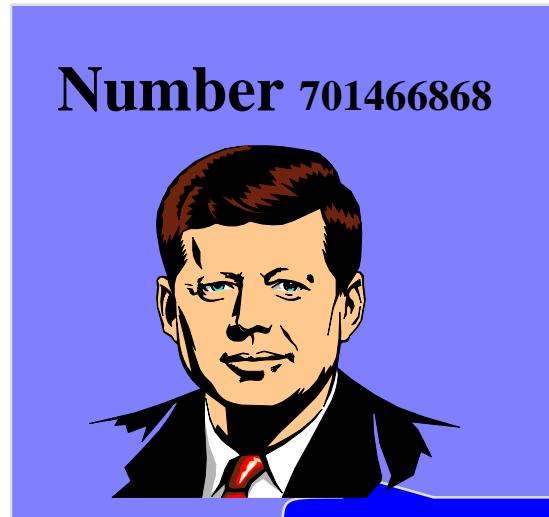
...



Number 155778322

# Түлхүүрийг хайх

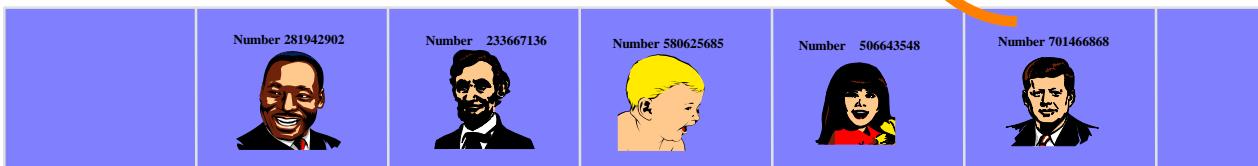
- Тухайн зүйл олдвол мэдээллийг шаардлагатай газар руу хуулж болно.



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 ]

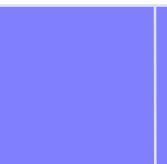
[ 2 ]

[ 3 ]

[ 4 ]

[ 5 ]

[ 700 ]



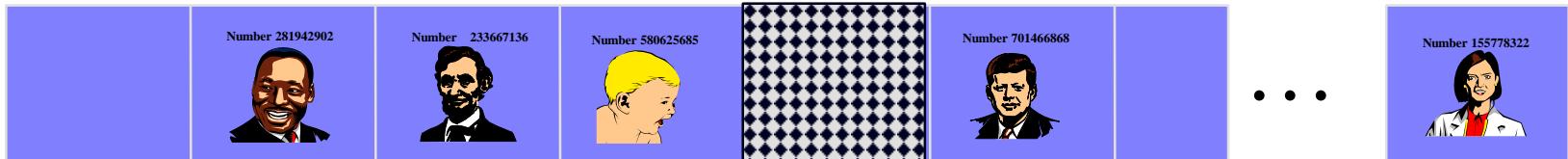
...



# 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.

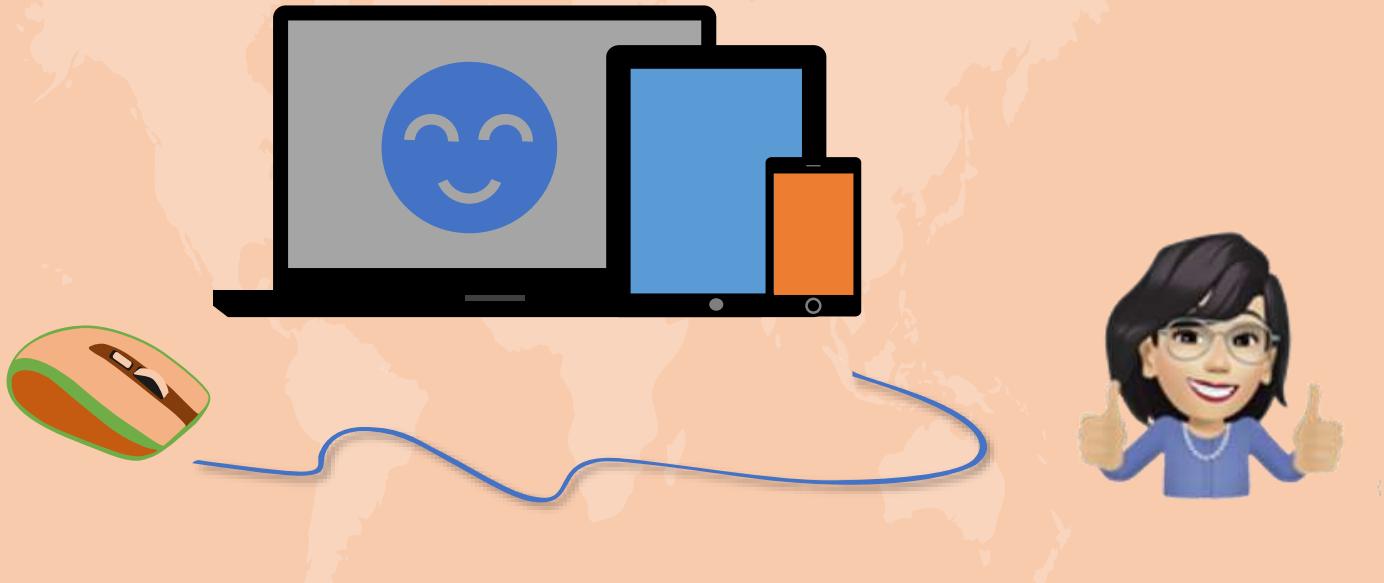
[ 0 ] [ 1 ] [ 2 ] [ 3 ] [ 4 ] [ 5 ] [ 700 ]





# Дүгнэлт

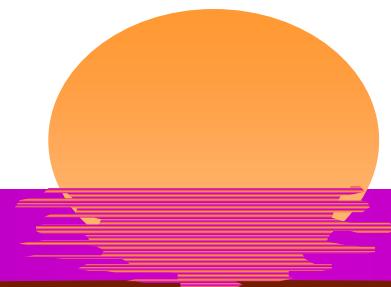
- Хэш хүснэгтүүд нь түлхүүр бүхий бичлэгийн цуглуулгыг хадгалдаг.
- Бичлэгийн байршил нь бичлэгийн түлхүүрийн хэш утгаас хамаарна.
- Мөргөлдөх үед дараагийн боломжтой байршлыг ашиглана.
- Тодорхой түлхүүр хайх нь ерөнхийдөө хурдан байдаг.
- Бичлэгийг үстгах үед тухайн нүдийг / газрыг тусгай аргаар тэмдэглэсэн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



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

# F.CSM203 - Өгөгдлийн бүтэц

ЛЕКЦI

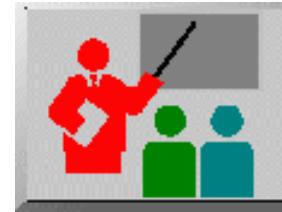


# Хичээлийн зорилго



- Өгөгдлийн бүтэц : дүрслэх, боловсруулах
- Бүх программ өгөгдөлтэй харьцдаг  
Тэгэхээр программ өгөгдлийг ямар нэг  
байдлаар дүрслэх хэрэгтэй.
- Өгөгдөлтэй харьцаж, боловсруулахад  
алгоритм хэрэгтэй.

# Хичээлийн зорилго



- Программыг хөгжүүлэхэд алгоритмын зохиомжийн аргууд хэрэгтэй.
- Өгөгдлийн бүтэц, алгоритм бол компьютерын ухааны суурь.

# Сурах бичиг



- Sahni Sartaj. Data Structures, Algorithms and Applications in Java. 2000. ISBN 0-07-109217-X
- Сургуулийн номын санд 1 хувьтай.
- Хичээлийн Веб хуудсанд шаардлагатай бүлгүүдийн хувилсан хуулбарыг pdf форматаар байрлуулсан.
- Сурах бичигт орсон жишээний эх код мөн zip форматаар тавигдана.

# Тавигдах шаардлага

- Java 
- Программчлалын үндэс



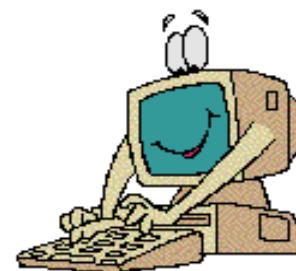
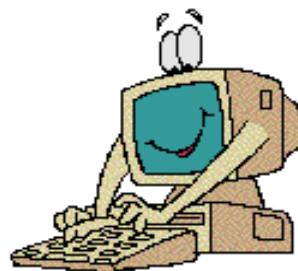
# Веб хуудас



- <https://elearn.sict.edu.mn/course/view.php?id=1904>
- Хичээлийн хөтөлбөр, лекцийн үзүүлэн, бие даалт, сурх бичиг, эх код гээд хэрэгтэй бүхнээ олж болно.

# Бие даалт

- Удирдамжийг удирдлага болгох
- Хугацаандаа өгсөн ажил үнэлэгдэнэ
- Нийт 4 бие даалт





# Лаборатории



- Лабораторийн ажиллах орчин - Eclipse  
([www.eclipse.org](http://www.eclipse.org))
- Dev C++
- Python
- <https://algorithm-visualizer.org/backtracking/hamiltonean-cycles>
- <https://www.cs.usfca.edu/~galles/visualization/Algorithms.html>
- Бусад

# Үнэлгээ

- 30 - улирлын шалгалт
- 10 - сорил шалгалт
- 32 - бие даалт ( $8 \times 4$  удаа)
- 28 - лабораторийн ажил, идэвх, ирц  
( $2 \times 14$  долоо хоног)



# Java

## Программын шинжүүд:

- Зөв үү?
- Программыг унших, кодыг ойлгох амархан үү?
- Программ хир баримтжуулагдсан?
- Өөрчлөлт оруулах амархан үү?
- Хир ой шаардагдах вэ?
- Хир удаан ажилах вэ?
- Программ хир нийтлэг вэ?
- Өөрчлөлтгүй өөр машин дээр ажиллах үү?

# Програмын бүтэц

Программ класс -> гишүүд (өгөгдөл, арга)

Бие даасан программ -> main()

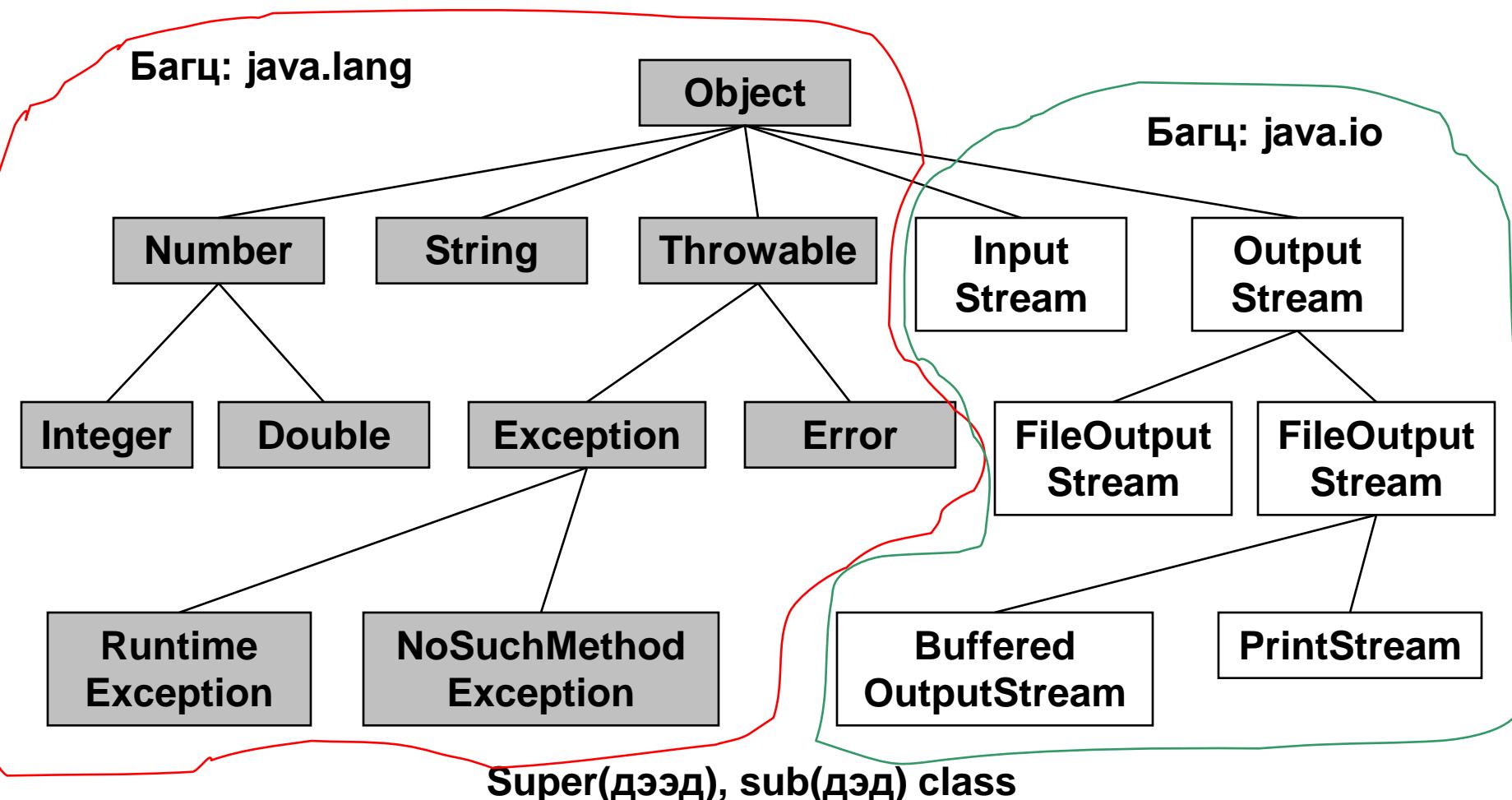
Багц: стандарт багц: java.awt, java.io, java.lang(Integer, String, Boolean ...), java.util

Сурах бичиг: applications, dataStructures, wrappers, exceptions, utilities, misc.

System.out.println("Welcome"); -> System: java.lang  
багцын класс, out: гишүүн (төрөл нь java.io багцын  
PrintStream)

Классыг импортлох: import java.io.\*

# Программын бүтэц



public class Welcome extends NemaOfSuperClass

public final class Welcame

# Java Виртуаль машин, Javadoc



Java хөрвүүлэгч: .java (эх код) -> .class  
(байткод)

Java виртуаль машин:

- + машинаас хамаарахгүй, цомхон, найдвартай
- удаан

Javadoc: /\*\* тайлбар \*/ , @param,  
@throws, @return ... -> HTML page

# Өгөгдлийн төрөл

Төрөл	Утга	Бит	Хүрээ
boolean	False	1	[false, true]
byte	0	8	[-128, 127]
char	\u0000	16	[\u0000, \xFFFF]
double	0.0	64	[+/-4.9e-324, +/-1.8e308]
float	0.0	32	[+/-1.4e-45, +/-3.4e38]
int	0	32	[-2147483648, 2147483647]
long	0	64	+/-9.2e17
short	0	16	[-32768, 32767]

Byte, Integer, Boolean, String -> Объект

# Арга

```
public static int abc{int a, int b, int c) {  
    return a+b*c+b/c}
```

- Параметр: зарласан, жинхэнэ
- Утга дамжина -> дотоод хувьсагч шиг
- Давхар ачаалах: public static float  
abc(...) {...}

# Онцгой тохиолдол

Дандаа алдаа байдаггүй!

- Унагах: if( $a < 0 \ || \ b < 0 \ || \ c < 0$ ) throw new  
IllegalArgumentException (“All parameters  
must be  $\geq 0$ ”);
- Олзолж, боловсруулах: try  
{System.out.Println(abc(2,-3,4));} catch  
(IllegalArgumentException e) {  
System.out.Println(“Positive parameters are  
needed!”);} finally { ... }

# Шинэ төрөл

```
public class Currency {  
    public static final boolean PLUS=true, MINUS=false;  
    private boolean sign;  
    private long dollars;  
    private byte cents;  
    public Currency(boolean sign, long dollars, byte cents) { ... }  
    public Currency() { this(PLUS,0L,(byte)0); }  
    public boolean getSign() { ... }  
    public long getDollars() { ... }  
    public void setSign(boolean sign) { ... }  
    public void setDollars(long dollars) { ... }  
    ...  
}
```

# Шинэ төрөл

- Өгөгдөл гишүүн (тогтмол): public static final boolean PLUS=true, MINUS=false;
- Өгөгдөл гишүүн (тохиолдол): private boolean sign; private long dollars; private byte cents;
- Арга гишүүн (байгуулагч): public Currency( ... ) { ... } public Currency() { ... }
- Арга гишүүн: public boolean getSign() { ... }  
public long getDollars() { ... } public void setSign(boolean sign) { ... } public void setDollars(long dollars) { ... }

# Хандалтыг өөрчлөгч

Өөрчлөгч	Гишүүний харагдах хүрээ
-	Тухайн багцын классууд
private	Тодорхой С класст
protected	Тухайн байгцын классууд, өөр багцын С дэд класст
public	Бүх багцын бүх класст

# Удамшил, аргыг орлох(солих)



Шинэ класс дээд классаас удамшдаг:  
тохиолдол өгөгдлийн гишүүний хуулбар,  
аргууд удамшдаг.

Currency класс Object классаас удамшсан:  
public boolean equals(Object obj); public  
String toString(); -> Object класс ->  
if(currA.equals(currB) { ... }

Currency.equals() -> Object.equals() –г орлоно

# Интерфейс

```
public interface LinearList {  
    public boolean isEmpty();  
    public int size();  
    public Object get(int index);  
    public int indexOf(Object elem);  
    public Object remove(int index);  
    public void add(int index, Object obj);  
    public String toString();  
}
```

# Хаягдал цуглуулах - Garbage Collection

```
int [] a = new int[50000];
```

```
Currency c = new Currency();
```

```
a = null;
```

```
c = null;
```

Хэрвээ a[] int биш Currency байсан бол  
7Mb ой хэрэгтэй болно.



# Pekypc

```
public static int factorial(int n) {  
    if (n <= 1)  
        return 1;  
    else  
        return n * factorial(n-1);  
}
```



# Өгөгдлийн бүтэц



Өгөгдлийн объект бол тохиолдол/утгын цуглуулга буюу олонлог юм. Жишээ нь:

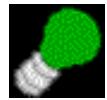
integer = {0, +1, -1, +2, -2, +3, -3, ...}

daysOfWeek = {S,M,T,W,Th,F,Sa}

# Өгөгдлийн объект

Тохиолдууд хамааралтай, хамааралгүй байж болно

```
myDataObject = {apple, chair, 2, 5.2, red, green, Jack}
```



# Өгөглийн бүтэц



Өгөгдлийн объектын холбоо нь нэг  
буюу хэд хэдэн тохиолдол дээр  
хэрэгжих үйлдлээр илэрхийлэгддэг

add, subtract, predecessor, multiply



## Шугаман (Дараалсан) Жагсаалт



тохиолдлууд дараах хэлбэртэй байна:

$$(e_0, e_1, e_2, \dots, e_{n-1})$$

Үүнд  $e_i$  жагсаалтын элемент

$n \geq 0$  – жагсаалт төгсгөлтэй

түүний хэмжээ нь  $n$



# Шугаман жагсаалт



$$L = (e_0, e_1, e_2, e_3, \dots, e_{n-1})$$

Холбоо:

$e_0$  – 0 дэх (эхний) элемент

$e_{n-1}$  – сүүлийн элемент

$e_i$  - элемент  $e_{i+1}$  элементийн өмнө орно

# Жишээ/Тохиолдол

CS204 –ийн оюутнууд =  
(Бат, Гэрэл, Дулмаа, Оюун, ..., Ганбат)

Days of Week = (S, M, T, W, Th, F, Sa)

Months = (Jan, Feb, Mar, Apr, ..., Nov, Dec)

# Шугаман жагсаалтын үйлдэл—size()

Жагсаалтын хэмжээг олох

$$L = (a, b, c, d, e)$$

Хэмжээ = 5

# Шугаман жагсаалтын үйлдэл—get(theIndex)

Өгөгдсөн индексийн элементийг авах

$$L = (a, b, c, d, e)$$

$$get(0) = a$$

$$get(2) = c$$

$$get(4) = e$$

$$get(-1) = \text{error}$$

$$get(9) = \text{error}$$

# Шугаман жагсаалтын үйлдэл — indexOf(theElement)

Элементийн индексийг олох

$$L = (a,b,d,b,a)$$

$$\text{indexOf}(d) = 2$$

$$\text{indexOf}(a) = 0$$

$$\text{indexOf}(z) = -1$$

# Шугаман жагсаалтын үйлдэл — remove(theIndex)

Өгөгдсөн индексийн элементийг устгаж,  
буцаах

$$L = (a, b, \cancel{c}, \cancel{d}, e, f, g)$$

*remove(2)* -с *c* буцах ба

$L = (a, b, d, e, f, g)$  болно

*d,e,f, g* –ийн индекс 1 –эр багасна

# Шугаман жагсаалтын үйлдэл — remove(theIndex)

Өгөгдсөн индексийн элементийг устгаж,  
буцаах

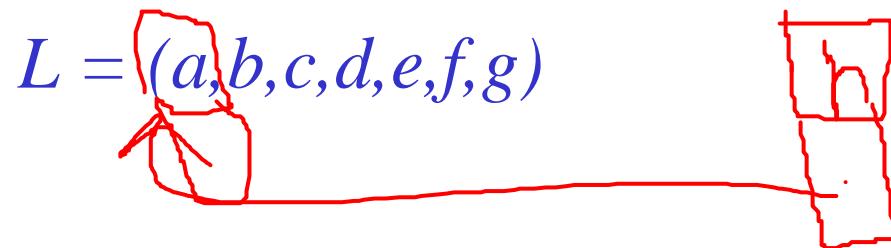
$$L = (a,b,c,d,e,f,g)$$

*remove(-1) => error*

*remove(20) => error*

# Шугаман жагсаалтын үйлдэл — add(theIndex, theElement)

Өгөгдсөн индексээр шинэ элемент  
нэмэх



$$add(0, h) \Rightarrow L = (h, a, b, c, d, e, f, g)$$

$a, b, c, d, e, f, g$  –ийн индекс 1-ээр  
нэмэгдэнэ

# Шугаман жагсаалтын үйлдэл — add(theIndex, theElement)

$$L = (a, b, c, d, e, f, g)$$

$add(2, h) \Rightarrow L = (a, b, h, c, d, e, f, g)$

$c, d, e, f, g$  –ийн индекс 1-ээр нэмэгдэнэ

$add(10, h) \Rightarrow \text{error}$

$add(-6, h) \Rightarrow \text{error}$

# Өгөгдлийн бүтцийн тодорхойлолт

## ❑ Хэлнээс хамаарахгүй

- Өгөгдлийн хийсвэр төрөл(Abstract Data Type)

## ❑ Java

- Интерфейс(Interface)
- Хийсвэр класс(Abstract Class)

# Шугаман жагсаалтын хийсвэр өгөгдлийн төрөл

Хийсвэр Өгөгдлийн Төрөл *LinearList*

{

тохиолдол

дараалсан элентийн цуглуулга(хоосон байж болно)

Үйлдлүүд

*isEmpty()*: жагсаалт хоосон бол true, үгүй бол false буцна

*size()*: хэмжээг буцаах (ө.х. жагсаалтын элементийн тоо)

*get(index)*: *index* дэх элементийг буцаах

*indexOf(x)*: эхэлж олдсон *x* –ийн индексийг буцаана.

байхгүй бол -1 буцна

*remove(index)*: *index* дэх элементийг устгаж, буцаана

устгагдсанаас хойши элементийн индекс 1-ээр багасна

*add(theIndex, x)*: *x* –ийг *index* дэх элемент болгож оруулах

оруулснаас хойши элементийн индекс 1-ээр ихэнэ

*output()*: элементүүдийг зүүнээс баруун тийш гаргах

}

# Шугаман жагсаалтын Java интерфейс

Интерфейст тогтмолууд болон  
хийсвэр аргууд(хэрэгжилт нь  
байхгүй арга) орж болно

# Шугаман жагсаалтын Java интерфейс

```
public interface LinearList
{
    public boolean isEmpty();
    public int size();
    public Object get(int index);
    public int indexOf(Object elem);
    public Object remove(int index);
    public void add(int index, Object obj);
    public String toString();
}
```

# Интерфейсийг хэрэгжүүлэх

```
public class ArrayList implements LinearList  
{  
    // бүх аргууд энд бичигдэнэ  
}
```

# Шугаман жагсаалтын хийсвэр класс

Хийсвэр класст тогтмолууд,  
хувьсагчид, хийсвэр болон хийсвэр  
бус аргууд байж болно

# Шугаман жагсаалтын хийсвэр класс

```
public abstract class LinearListAsAbstractClass
{
    public abstract boolean isEmpty();
    public abstract int size();
    public abstract Object get(int index);
    public abstract int indexOf(Object theElement);
    public abstract Object remove(int index);
    public abstract void add(int index,
                           Object theElement);
    public abstract String toString();
}
```

# Java классыг өргөтгөх

```
public class ArrayList
    extends LinearListAsAbstractClass
{
    // бүх хийсвэр классууд энд бичигдэнэ
}
```

# Олон интерфейсийг хэрэгжүүлэх

```
public class MyInteger implements Operable, Zero,  
    CloneableObject  
{  
    // Operable, Zero, CloneableObject –ийн  
    // бүх аргууд энд бичигдэнэ  
}
```



## Олон классыг өргөтгөх

### **JAVA –д үүнийг зөвшөөрөхгүй**

Класс олон интерфейсийг хэрэгжүүлж болох боловч зөвхөн 1 классыг өргөтгөж болно.

# Өгөгдлийг дурслэх аргууд



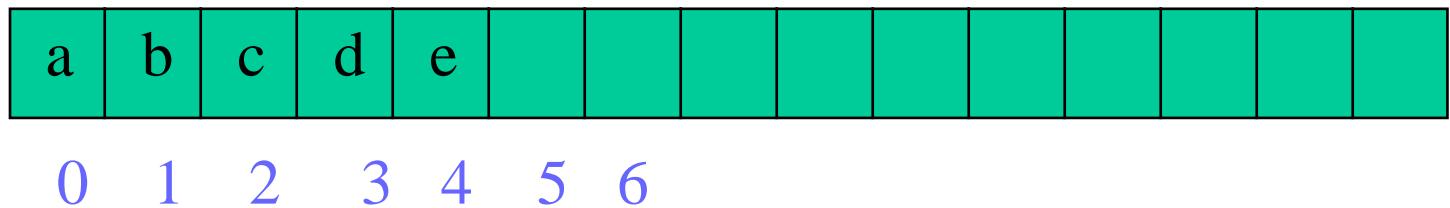
массив --- Бүлэг 5

холбоос --- Бүлэг 6

заагч --- Бүлэг 7

# Шугаман жагсаалтын массив дүрслэл

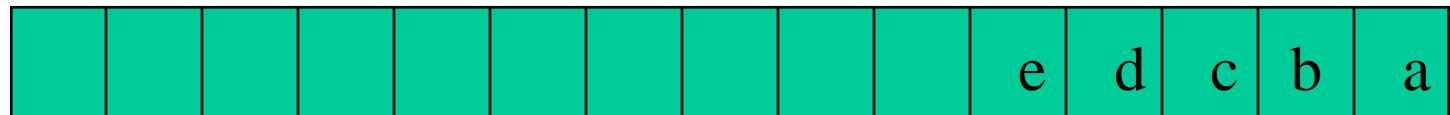
Нэг хэмжээст массив - `element[]`



$$L = (a, b, c, d, e)$$

*i* дэх элементийг `element[i]` –д хадгална.

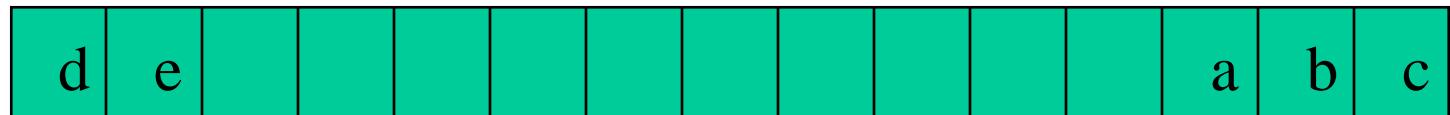
# Баруунаас зүүн тийш байршуулах



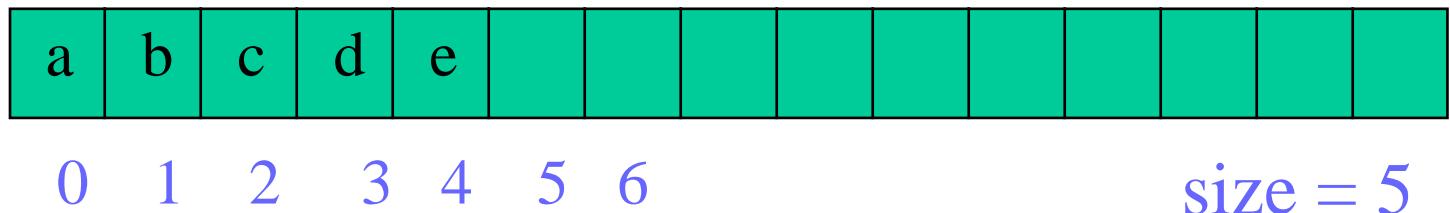
# Байршил алгасаж байршуулах



# Тойргоор байршуулах



# Бидний ашиглах дурслэл



**i** дэх элементийг `element[i]` –д хадгална

**size** элементийн тоог илэрхийлэхэд энэ хувьсагчийг ашиглана

# Элемент нэмэх/xacax

size = 5

a	b	c	d	e									
---	---	---	---	---	--	--	--	--	--	--	--	--	--

add(1,g)

size = 6

a	g	b	c	d	e								
---	---	---	---	---	---	--	--	--	--	--	--	--	--

# element[] массивын өгөгдлийн төрөл

Жагсаалтын өгөгдлийн төрөл тодорхойгүй.

element[] -өгөгдлийн төрлийг **Object** гэж тодорхойльё.

Манай жагсаалтанд энгийн төрлүүд (**int**, **float**, **double**, **char**, г.м.) орохгүй.

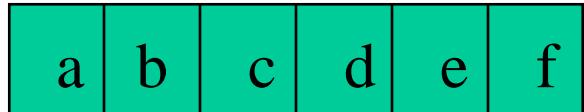
# element[] массивын урт

Жагсаалтанд хэдэн элемент байхыг мэдэхгүй.

Анхны уртыг нь зааж өгөөд хэрэгцээгээрээ  
нэмье.

# Массивын уртыг нэмэх

element[] массивын урт 6.



Эхлээд илүү урттай шинэ массив үүсгэнэ

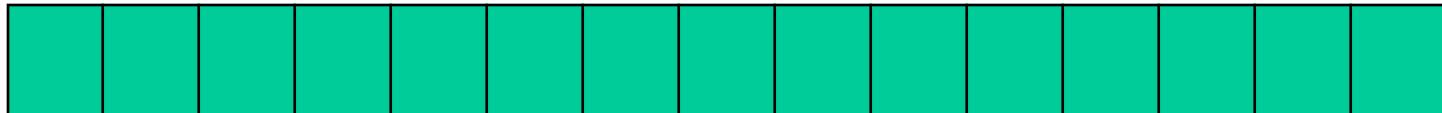
`newArray = new Object[15];`



# Массивын уртыг нэмэх

Элементүүдийг хуучнаас шинэ массивт хуулна.

a	b	c	d	e	f
---	---	---	---	---	---



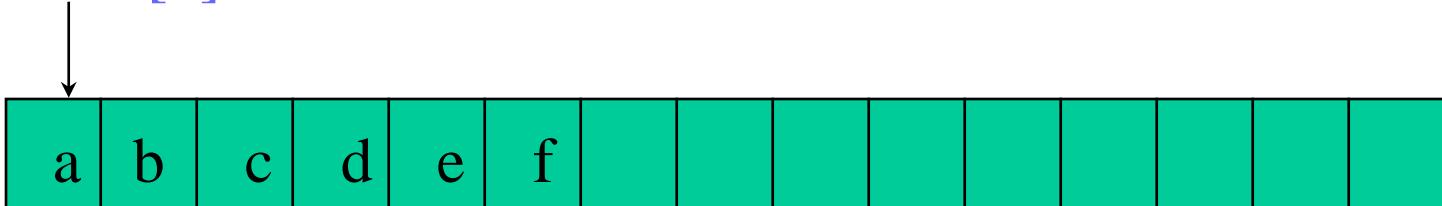
a	b	c	d	e	f									
---	---	---	---	---	---	--	--	--	--	--	--	--	--	--

# Массивын уртыг нэмэх

Эцэст нь шинэ массивын нэрийг  
өөрчилнө.

`element = newArray;`

`element[0]`



`element.length = 15`

# Нэг дор бичвэл

```
// зөв төрөл, урттай шинэ массив үүсгэх  
Object [] newArray = new Object [newLength];  
  
// элементүүдийг хуулах  
System.arraycopy(element, 0, newArray, 0,  
    element.length);  
  
// нэрийг өөрчлөх  
element = newArray;
```

```
public static Object [] changeLength(Object [] a,  
                                    int newLength)  
{  
    Object [] newArray = new Object [newLength];  
    System.arraycopy(...);  
    return newArray;  
}
```

// erroneous  
37

# ArrayList - класс



- Шугаман жагсаалтын нийтлэг зорилготой хэрэгжүүлэлт.
- Жагсаалтын элементийн тоо тодорхойгүй.

# Хоосон жагсаалтыг ҮҮСГЭХ

ArrayList a = new ArrayList(100),

    b = new ArrayList(),

    c;

ArrayList d = new ArrayList(1000),

    e = new ArrayList(),

    f;

# Шугаман жагсаалтыг ашиглах

```
System.out.println(a.size());  
a.add(0, Integer(2));  
b.add(0, Integer(4));  
System.out.println(a);  
b.remove(0);  
if (a.isEmpty())  
    a.add(0, Integer(5));
```

## Шугаман жагсаалтын массив

```
LinearList [] x = new LinearList [4];
x[0] = new ArrayLinearList(20);
x[1] = new Chain();
x[2] = new Chain();
x[3] = new ArrayLinearList();
for (int i = 0; i < 4; i++)
    x[i].add(0, new Integer(i));
```

# ArrayList класс

```
/** LinearList –ийн хэрэгжүүлэлт */
package dataStructures;
import java.util.*; // Iterator интерфейс
import utilities.*; // массивын хэмжээг өөрчлөх класс

public class ArrayList implements LinearList
{
    // Өгөгдөл гишүүд
    protected Object [] element; // массив
    protected int size; // элементийн тоо
    // Байгуулагч болон бусад аргууд
}
```



# Байгуулагч

```
/** initialCapacity хэмжээтэй жагсаалт үүсгэх
 * @throws initialCapacity < 1 бол
 * IllegalArgumentException унана
 */
public ArrayList(int initialCapacity)
{
    if (initialCapacity < 1)
        throw new IllegalArgumentException
            ("initialCapacity must be >= 1");
    // хэмжээний анхны утга 0
    element = new Object [initialCapacity];
}
```

# Өөр байгуулагч



/\*\* 10 –ын урттай жагсаалт \*/

```
public ArrayList()  
{  
    this(10);  
}
```

## isEmpty арга



```
/** @return хоосон бол true */
```

```
public boolean isEmpty()  
{ return size == 0; }
```

## size() арга

```
/** @return жагсаалтын хэмжээ */  
public int size()  
{ return size; }
```

## checkIndex арга

```
/** @throws index хэмжээнээс хальбал  
 * IndexOutOfBoundsException */  
void checkIndex(int index)  
{  
    if (index < 0 || index >= size)  
        throw new IndexOutOfBoundsException  
            ("index = " + index + " size = " + size);  
}
```

## get арга

```
/** @return index –ийн элемент
 * @throws index хэмжээнээс халььбал
 * IndexOutOfBoundsException унана */
public Object get(int index)
{
    checkIndex(index);
    return element[index];
}
```

# indexOf арга

```
/** @return theElement элементийн индекс,  
 * байхгүй бол -1 */  
public int indexOf(Object theElement)  
{  
    // theElement -г element[] -аас хайх  
    for (int i = 0; i < size; i++)  
        if (element[i].equals(theElement))  
            return i;  
  
    // theElement олдсонгүй  
    return -1;  
}
```

## remove арга

```
public Object remove(int index)
{
    checkIndex(index);

    // индекс зөв бол элементүүдийг шилжүүлэх
    Object removedElement = element[index];
    for (int i = index + 1; i < size; i++)
        element[i-1] = element[i];

    element[--size] = null; // хаягдалд өгөх
    return removedElement;
}
```

add арга

## add арга

```
// элементүүдийг баруун тийш гүйлгэх  
for (int i = size - 1; i >= index; i--)  
    element[i + 1] = element[i];  
  
element[index] = theElement;  
  
size++;  
}
```

## Баруун тийш хурдан гүйлгэх арга

```
System.arraycopy(element, index, element,  
                 index + 1, size - index);
```

# ХЭЛХЭЭСТ ХӨРВҮҮЛЭХ

```
public String toString()
{
    StringBuffer s = new StringBuffer("[");
    // буферт элементийг хийх
    for (int i = 0; i < size; i++)
        if (element[i] == null) s.append("null, ");
        else s.append(element[i].toString() + ", ");
    if (size > 0) s.delete(s.length() - 2, s.length());
        // сүүлийн ", " -г устгах
    s.append("]");
    // эквивалент хэлхээс үүсгэх
    return new String(s);
}
```



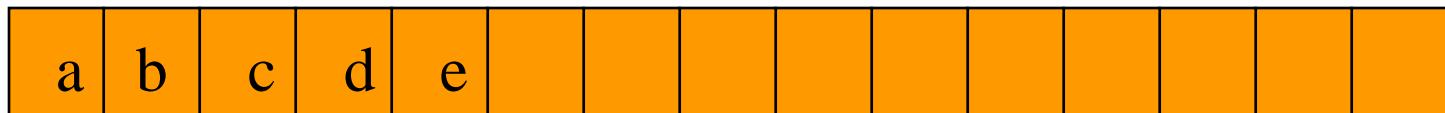
# Жагсаалт. Холбоост суурилсан дүрслэл



- Жагсаалтын элемент ойд зоргоороо байрлана
- Элементээс элементед шилжихдээ нэмэлт мэдээлэл (**холбоос**) ашиглана

# Ойн байршил

$L = (a, b, c, d, e)$  жагсаалт массив дүрслэлээр.

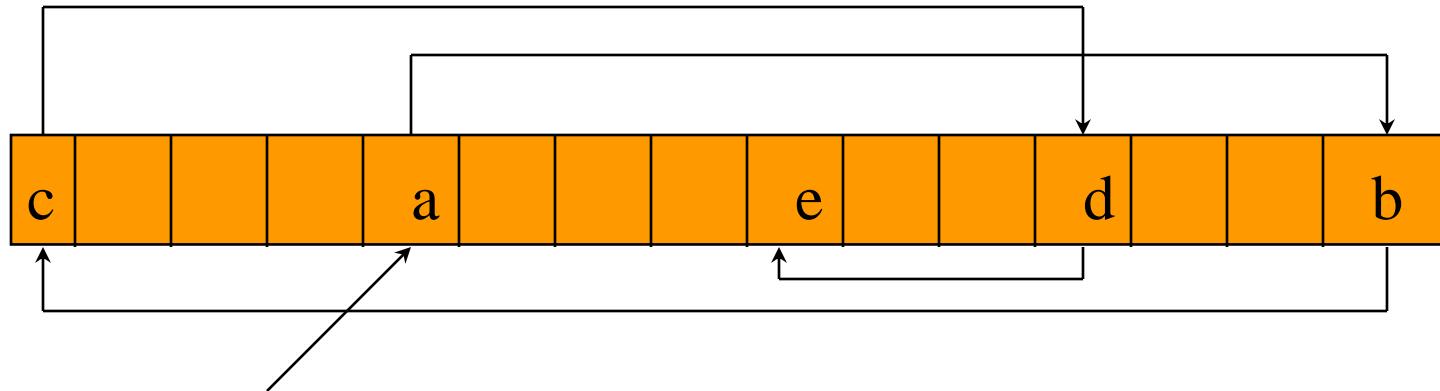


Холбоост дүрслэлд зоргоороо/санамсаргүй байршина.





# Холбоост дүрслэл



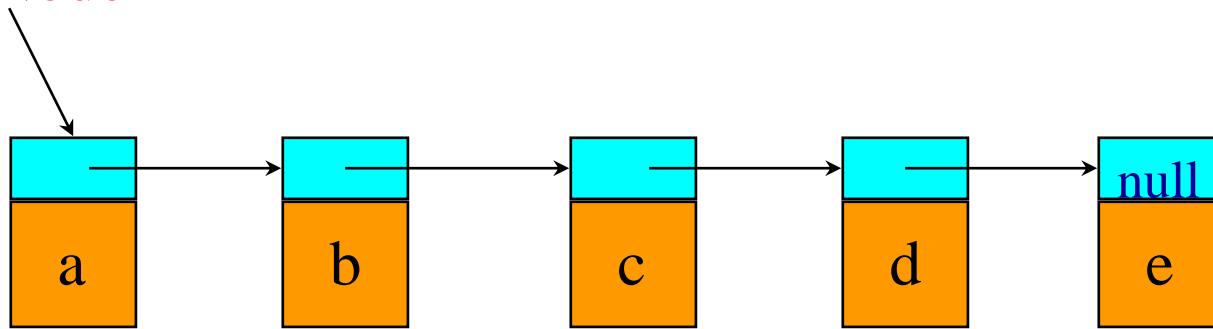
firstNode

е –ийн заагч (холбоос) null

хувьсагч firstNode эхний элемент a  
–д хүрэхэд ашиглагдана

# Холбоост жагсаалтыг зурах арга

firstNode

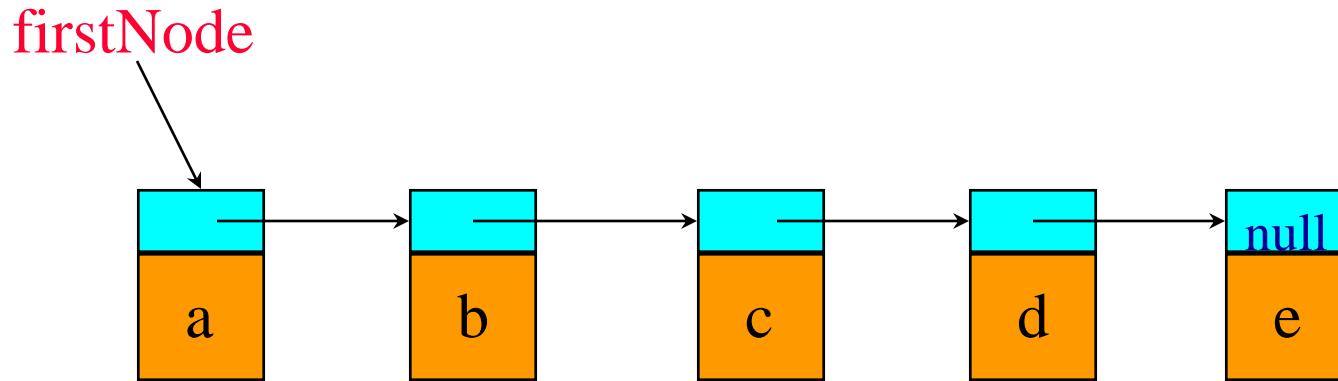


Зангилааны холбоос буюу заагчийн талбар



Зангилааны өгөгдлийн талбар

# Гинж



- Гинж бол зангилаа бүр нэг элементийг илэрхийлэх холбоост жагсаалт.
- Энд холбоос буюу заагч нэг элементээс нөгөөд хүрэхийг зааж байна.
- Сүүлийн зангилааний заагч **null** байна.

# Зангилааг дүрслэх

```
package dataStructures;
```

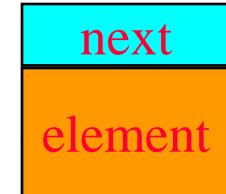
```
class ChainNode
```

```
{
```

```
// багцын харагдах өгөгдөл гишүүд
```

```
Object element;
```

```
ChainNode next;
```



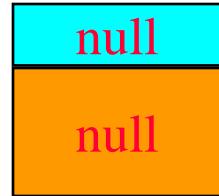
```
// байгуулагчид
```

```
}
```

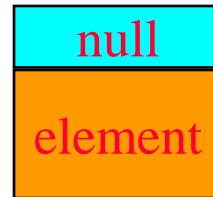
# ChainNode –ийн байгуулагчид



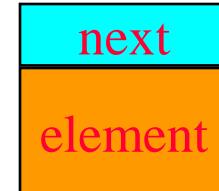
```
ChainNode() { }
```



```
ChainNode(Object element)  
{this.element = element; }
```

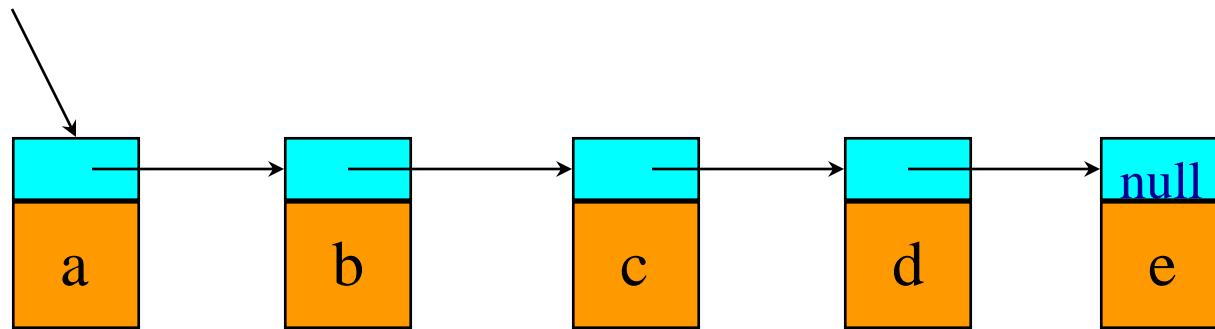


```
ChainNode(Object element, ChainNode next)  
{this.element = element;  
this.next = next; }
```



# get(0)

firstNode



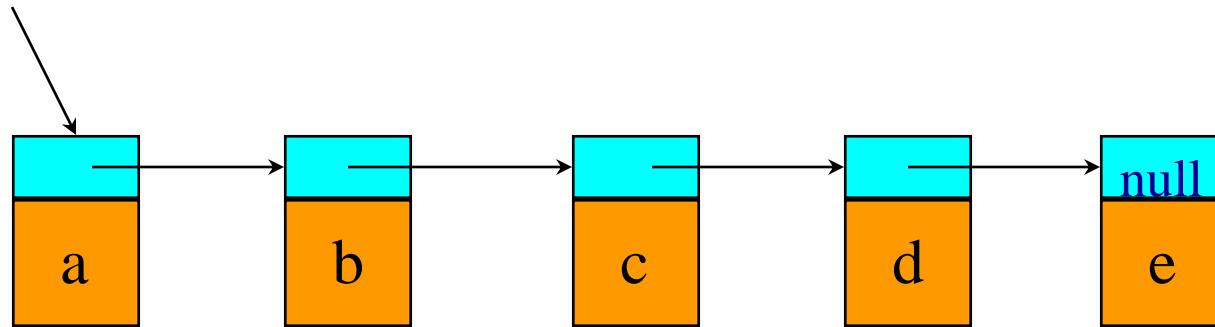
checkIndex(0);

desiredNode = firstNode; // Эхний зангилааг авах

return desiredNode.element;

# get(1)

firstNode

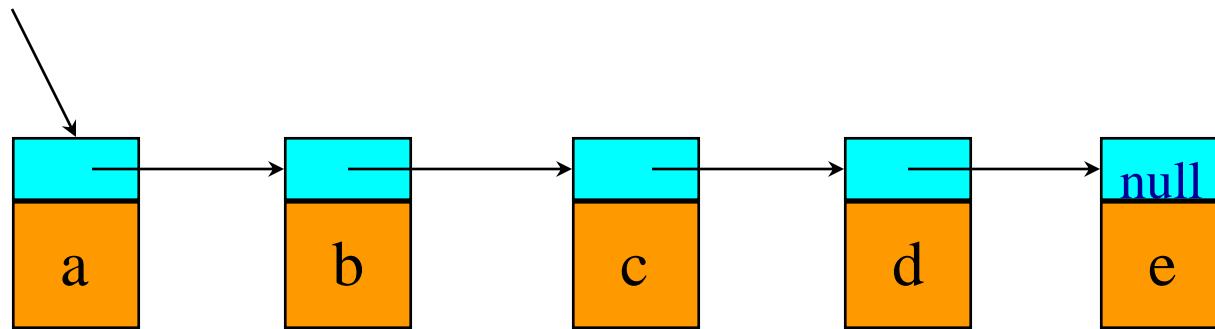


checkIndex(1);

```
desiredNode = firstNode.next; // хоёр дахь зангилааг авах  
return desiredNode.element;
```

# get(2)

firstNode

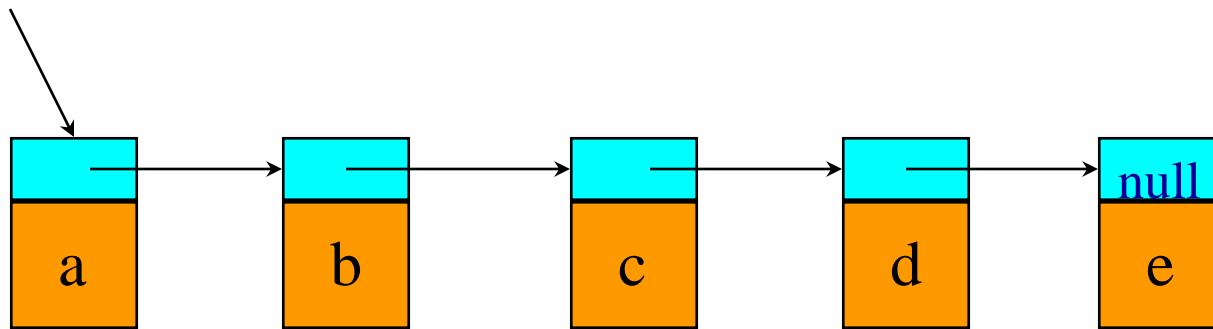


checkIndex(2);

```
desiredNode = firstNode.next.next; // гурав дахь зангилааг авах  
return desiredNode.element;
```

# get(5)

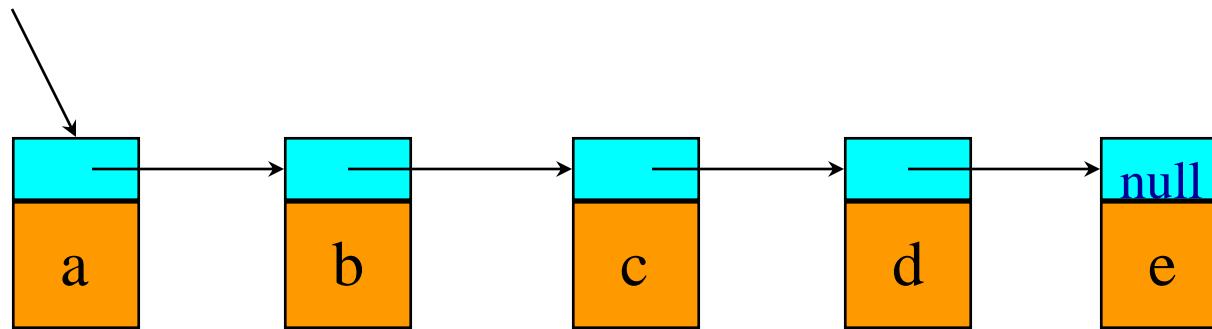
firstNode



```
checkIndex(5);           // онцгой тохиолдол унана  
desiredNode = firstNode.next.next.next.next;  
                         // desiredNode = null  
return desiredNode.element; // null.element
```

# NullPointerException

firstNode



desiredNode =

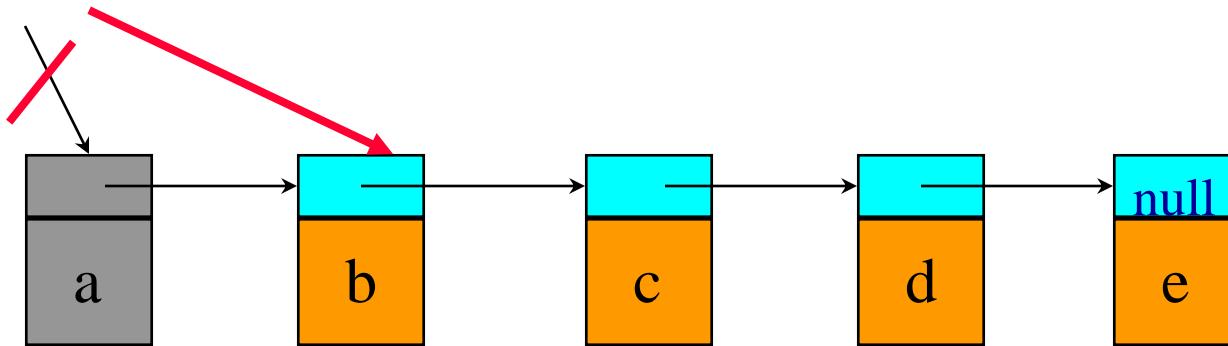
```
firstNode.next.next.next.next.next;
```

// компьютер тэнэг байдалд орж

// NullPointerException унана

# Элемент устгах

firstNode

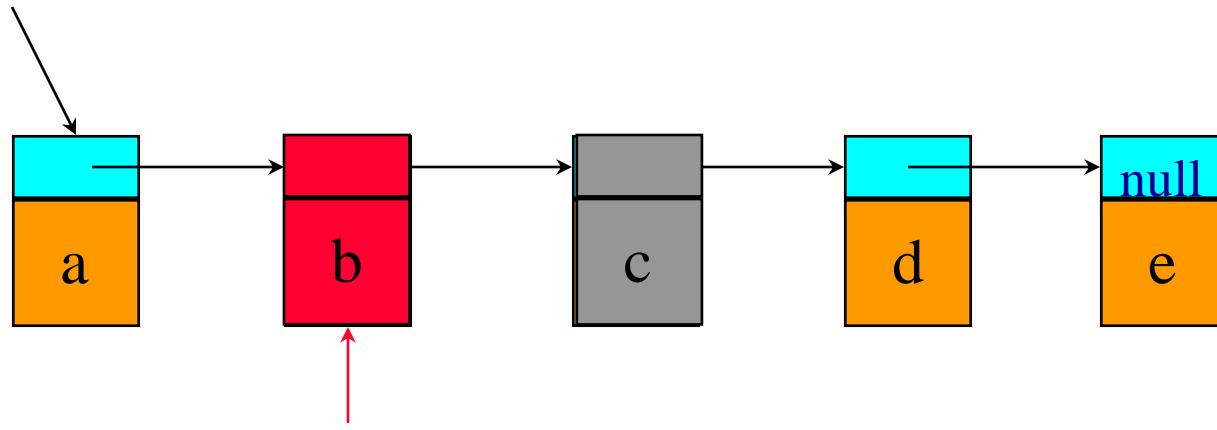


remove(0)

firstNode = firstNode.next;

# remove(2)

firstNode

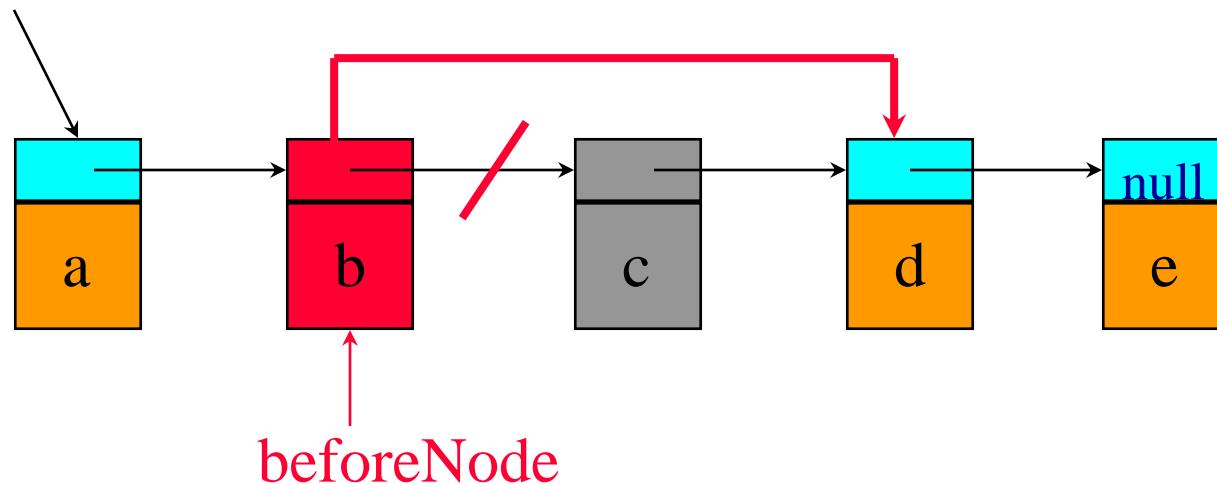


Зангилааг устгахаас өмнө түүнд хүрэх ёстой

**beforeNode = firstNode.next;**

# remove(2)

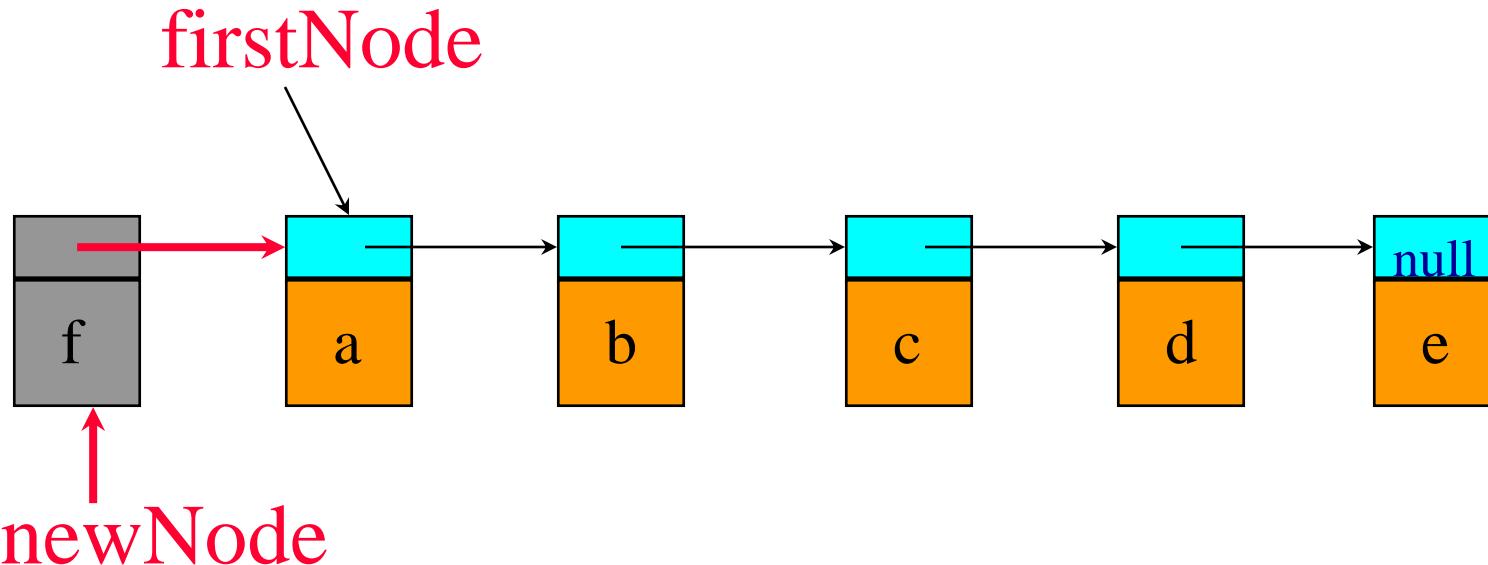
firstNode



Одоо **beforeNode** –д байгаа заагчийг өөрчилж болно

**beforeNode.next = beforeNode.next.next;**

$\text{add}(0, 'f')$

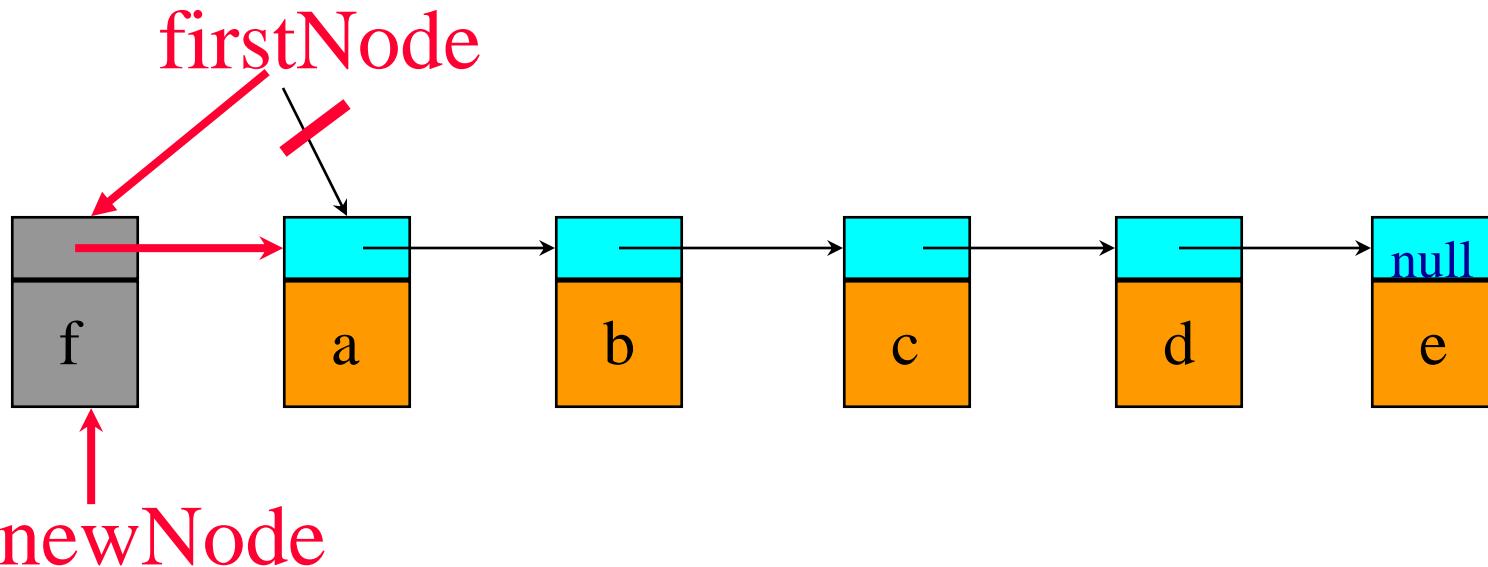


Алхам 1: зангилааг үүсгэн өгөгдөл, холбоосын талбарыг тогтооно

ChainNode newNode =

new ChainNode(new Character('f'), firstNode);

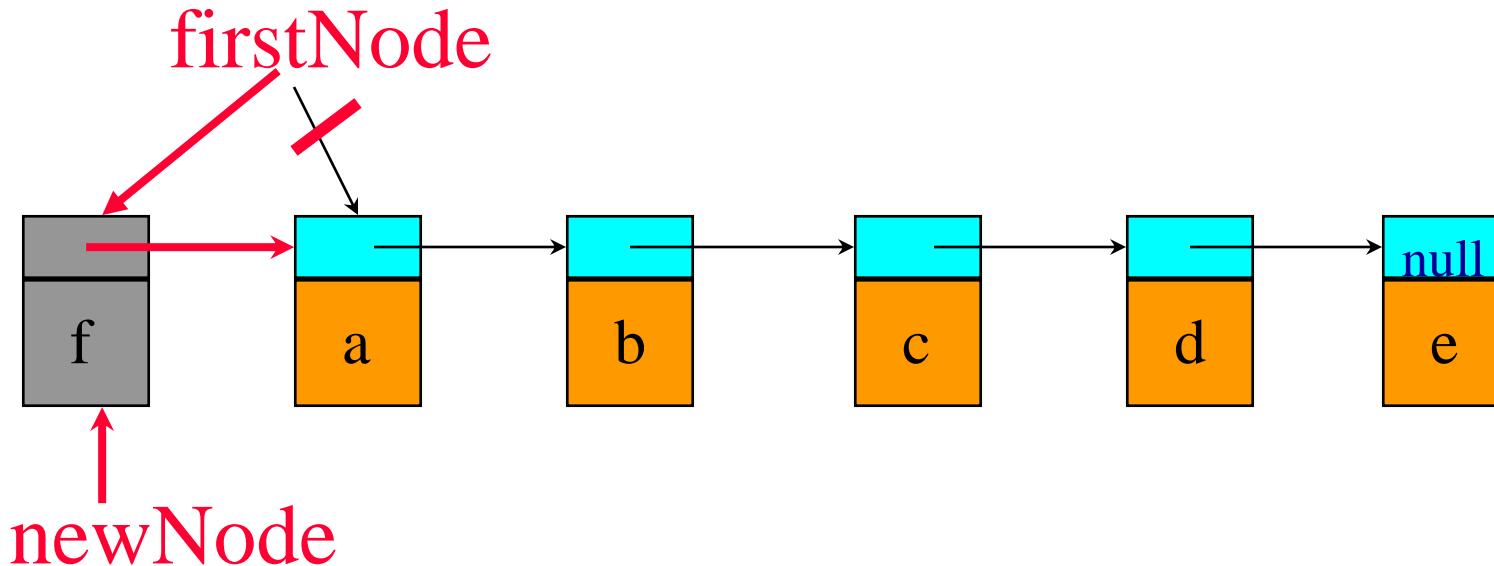
$\text{add}(0, 'f')$



Алхам 2: `firstNode` –г шинэчлэхэ

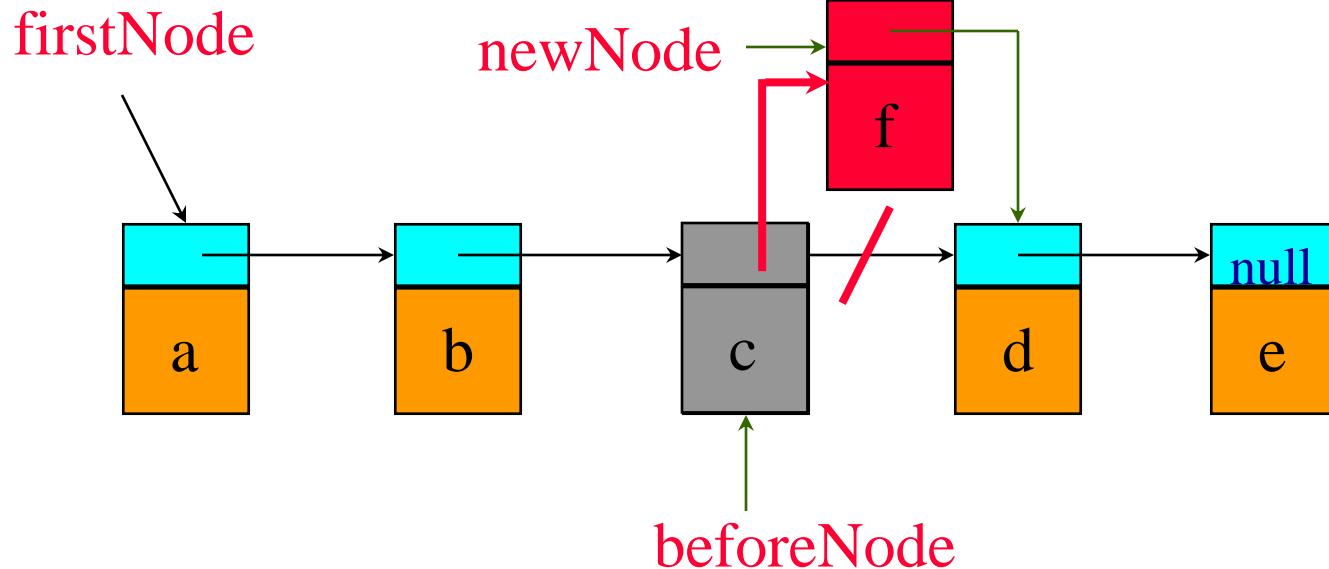
`firstNode = newNode;`

# Нэг алхмын add(0,'f')



```
firstNode = new ChainNode(  
    new Character('f'), firstNode);
```

# add(3,'f')

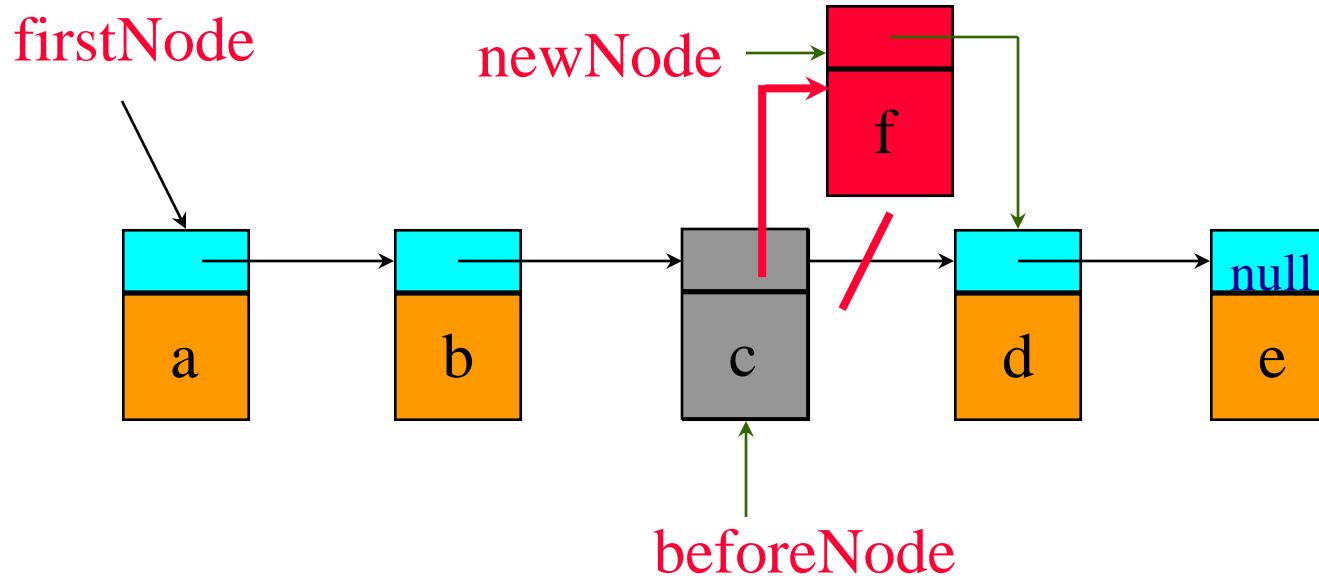


- Эхлээд **2** гэсэн индекстэй зангилааг олох
- дараа нь зангилааг үүсгэн өгөгдөл, холбоосын талбарыг тогтооно

```
ChainNode newNode = new ChainNode(new Character('f'),  
                                beforeNode.next);
```

- Эцэст нь **beforeNode** -г **newNode** –тай холбоно
- ```
beforeNode.next = newNode;
```

# Хоёр алхмын add(3,'f')



```
beforeNode = firstNode.next.next;
```

```
beforeNode.next = new ChainNode(new Character('f'),  
                                beforeNode.next);
```

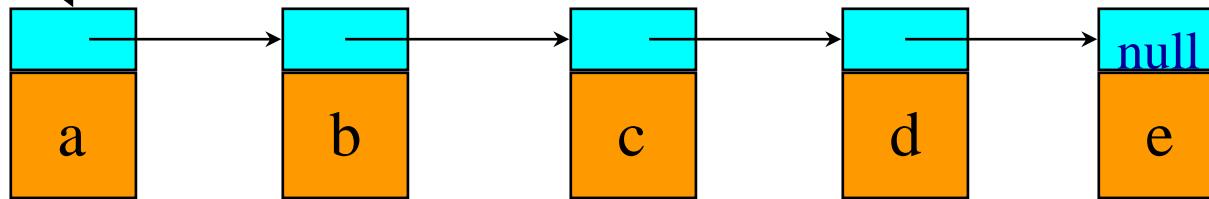


# Класс Chain



# Класс Chain

firstNode

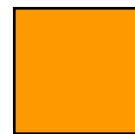


size = Элементийн тоо

ChainNode –г ашиглах



next (datatype ChainNode)



element (datatype Object)



# Класс Chain



```
/** LinearList холбоост хэрэгжүүлэлт*/
package dataStructures;
import java.util.*; // Iterator –г ашиглана
public class Chain implements LinearList
{
    // өгөгдөл гишүүд
    protected ChainNode firstNode;
    protected int size;

    // Chain –ий аргууд
}
```



## Байгуулагчид



```
/** хоосон жагсаалт үүсгэх */
public Chain(int initialCapacity)
{
    // firstNode болон size –ий анхны утга
    // null болон 0 байх болно

}

public Chain()
{
    this(0);
}
```

# isEmpty арга



```
/** @return жагсаалт хоосон бол true */  
public boolean isEmpty()  
{ return size == 0; }
```

# size() арга

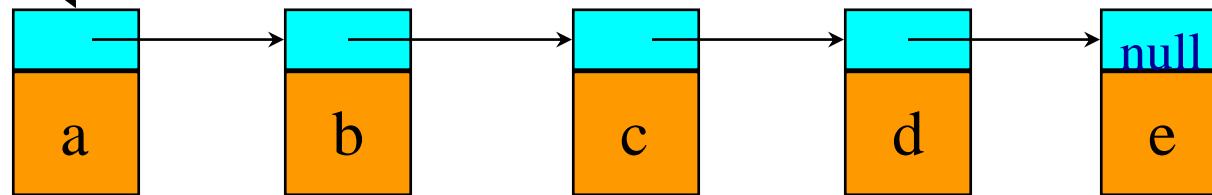
```
/** @return элементийн тоо */  
public int size()  
{ return size; }
```

## checkIndex арга

```
/** @throws index -н утга 0, size - 1 -н хооронд биш  
  бол IndexOutOfBoundsException унана */  
void checkIndex(int index)  
{  
    if (index < 0 || index >= size)  
        throw new IndexOutOfBoundsException  
            ("index = " + index + " size = " + size);  
}
```

firstNode

get arga



```
public Object get(int index)
{
    checkIndex(index);

    // хэрэгтэй зангилаанд хүрэх
    ChainNode currentNode = firstNode;
    for (int i = 0; i < index; i++)
        currentNode = currentNode.next;

    return currentNode.element;
}
```

# indexOf арга

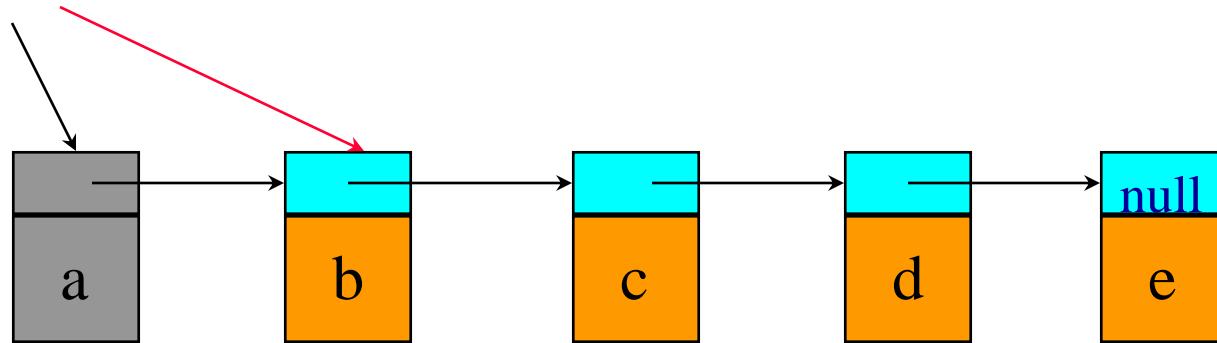
```
public int indexOf(Object theElement)
{
    // theElement –г хайх
    ChainNode currentNode = firstNode;
    int index = 0; // currentNode –н индекс
    while (currentNode != null &&
           !currentNode.element.equals(theElement))
    {
        // дараагийн зангилаанд хүрэх
        currentNode = currentNode.next;
        index++;
    }
}
```

# indexOf арга

```
// элемент олдсон уу?  
if (currentNode == null)  
    return -1;  
else  
    return index;  
}
```

# Элементийг устгах

firstNode



remove(0)

firstNode = firstNode.next;



## Элементийг устгах

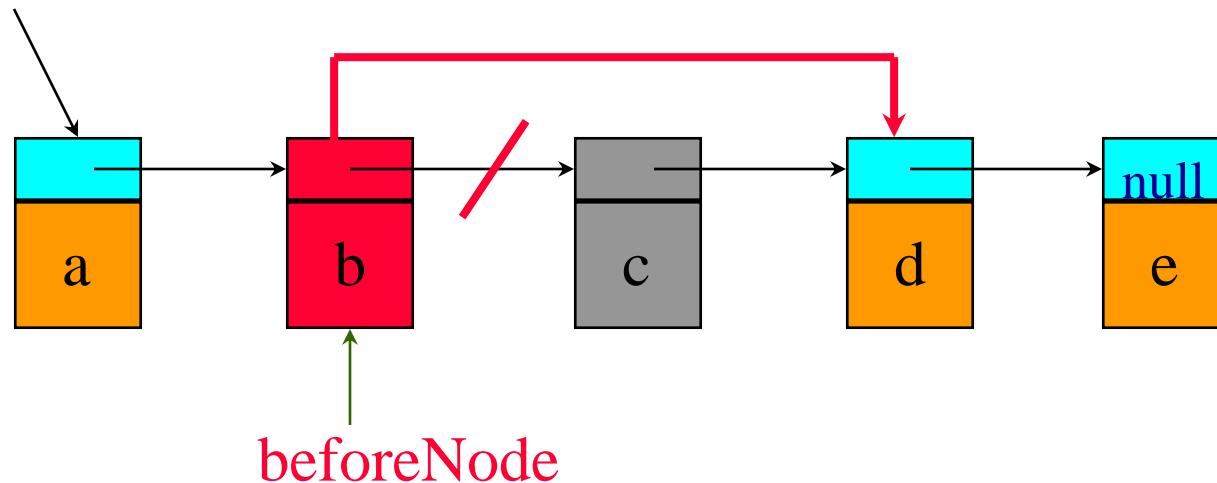


```
public Object remove(int index)
{
    checkIndex(index);

    Object removedElement;
    if (index == 0) // эхний зангилааг устгах
    {
        removedElement = firstNode.element;
        firstNode = firstNode.next;
    }
}
```

## remove(2)

firstNode



beforeNode –г олж, заагчийг өөрчилнө.

`beforeNode.next = beforeNode.next.next;`



## Элементийг устгах



else

```
{ // хэрэгтэй зангилааны өмнөхыг олоход q -г ашиглана  
ChainNode q = firstNode;  
for (int i = 0; i < index - 1; i++)  
    q = q.next;
```

```
removedElement = q.next.element;  
q.next = q.next.next; // хайсан зангилаагаа устгах
```

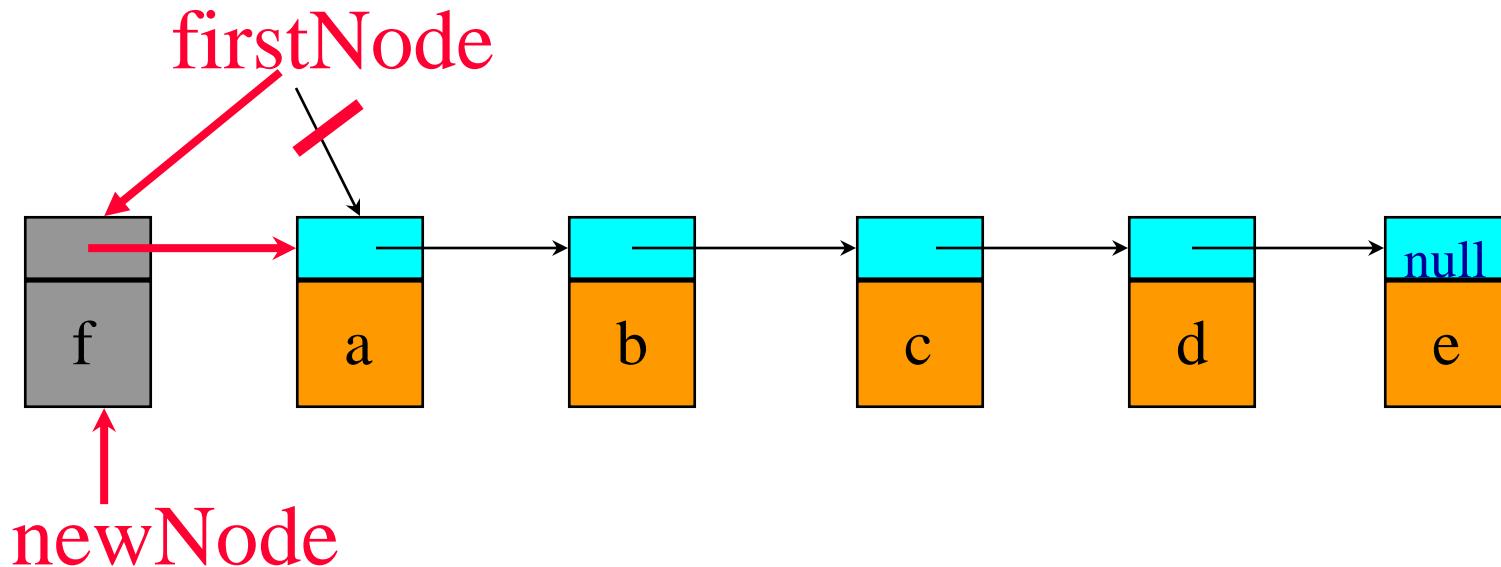
}

size--;

return removedElement;

}

# Нэг алхмын add(0,'f')



```
firstNode = new ChainNode('f', firstNode);
```

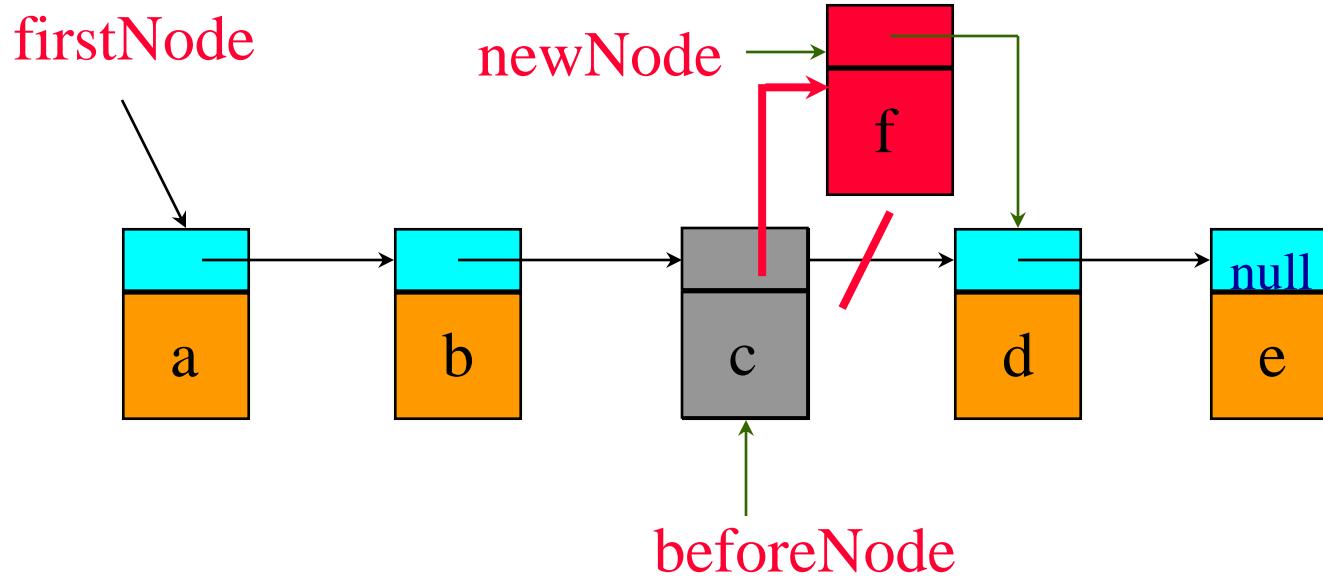


## Элемент нэмэх



```
public void add(int index, Object theElement)  
{  
    if (index < 0 || index > size)  
        // индекс буруу байна  
        throw new IndexOutOfBoundsException  
            ("index = " + index + " size = " + size);  
  
    if (index == 0)  
        // эхэнд оруулах  
    firstNode = new ChainNode(theElement, firstNode);
```

# Хоёр алхмын add(3,'f')



```
beforeNode = firstNode.next.next;
```

```
beforeNode.next = new ChainNode('f', beforeNode.next);
```



## Элемент нэмэх



```
else
```

```
{ // шинэ элементийн өмнөхийг олох
```

```
ChainNode p = firstNode;
```

```
for (int i = 0; i < index - 1; i++)
```

```
    p = p.next;
```

```
// p -ийн дараа оруулах
```

```
p.next = new ChainNode(theElement, p.next);
```

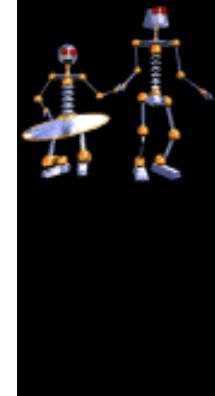
```
}
```

```
size++;
```

```
}
```

# Чанарын үзүүлэлт

Үйлдэл бүрийг 40,000 давтлаа



# Чанарын үзүүлэлт

Үйлдэл бүрийг 40,000 давтлаа

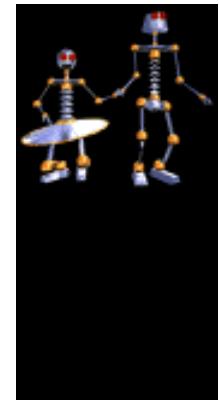


| Үйлдэл        | FastArrayList | Chain  |
|---------------|---------------|--------|
| авах          | 5.6ms         | 157sec |
| нэмэх-сайн    | 31.2ms        | 304ms  |
| нэмэх-дундаж  | 5.8sec        | 115sec |
| нэмэх-муу     | 11.8sec       | 157sec |
| устгах-сайн   | 8.6ms         | 13.2ms |
| устгах-дундаж | 5.8sec        | 149sec |
| устгах-муу    | 11.7sec       | 157sec |

# Чанарын ҮЗҮҮЛЭЛТ



Индексжүүлсэн AVL ('62 - G.M **A**delson-**V**elsky & E.M **L**andis)  
мод (IAVL)



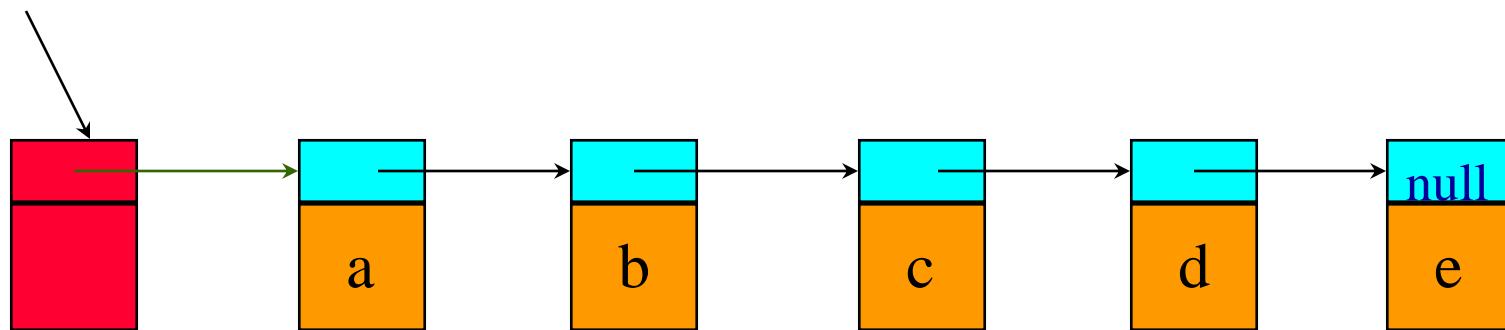
## Чанарын ҮЗҮҮЛЭЛТ

Индексжүүлсэн AVL мод (IAVL)

| Үйлдэл        | FastArrayList | Chain  | IAVL   |
|---------------|---------------|--------|--------|
| авах          | 5.6ms         | 157sec | 63ms   |
| нэмэх-сайн    | 31.2ms        | 304ms  | 253ms  |
| нэмэх-дундаж  | 5.8sec        | 115sec | 392ms  |
| нэмэх-муу     | 11.8sec       | 157sec | 544ms  |
| устгах-сайн   | 8.6ms         | 13.2ms | 1.3sec |
| Устгах-дундаж | 5.8sec        | 149sec | 1.5sec |
| устгах-муу    | 11.7sec       | 157sec | 1.6sec |

# Толгойн зангилаатай гинж

headerNode

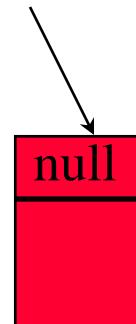




# Толгойн зангилаатай хоосон гинж

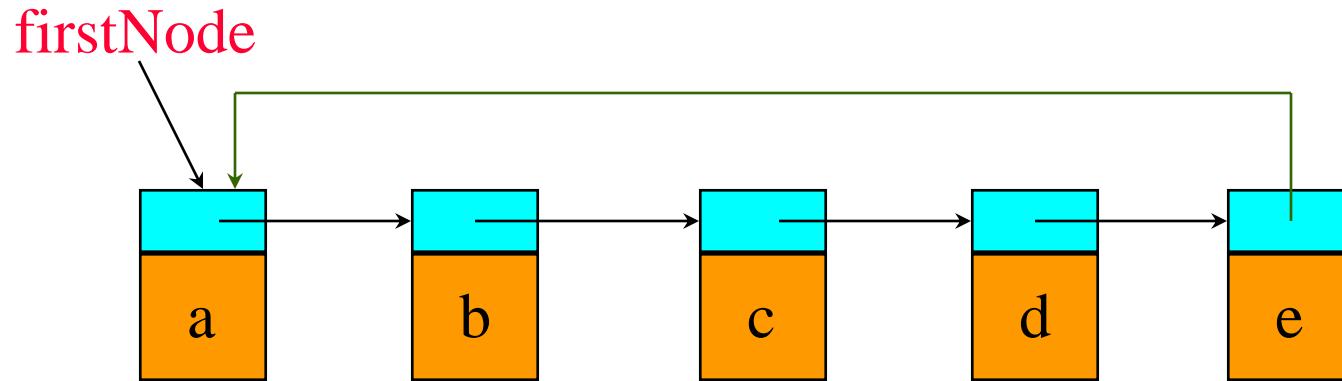


headerNode



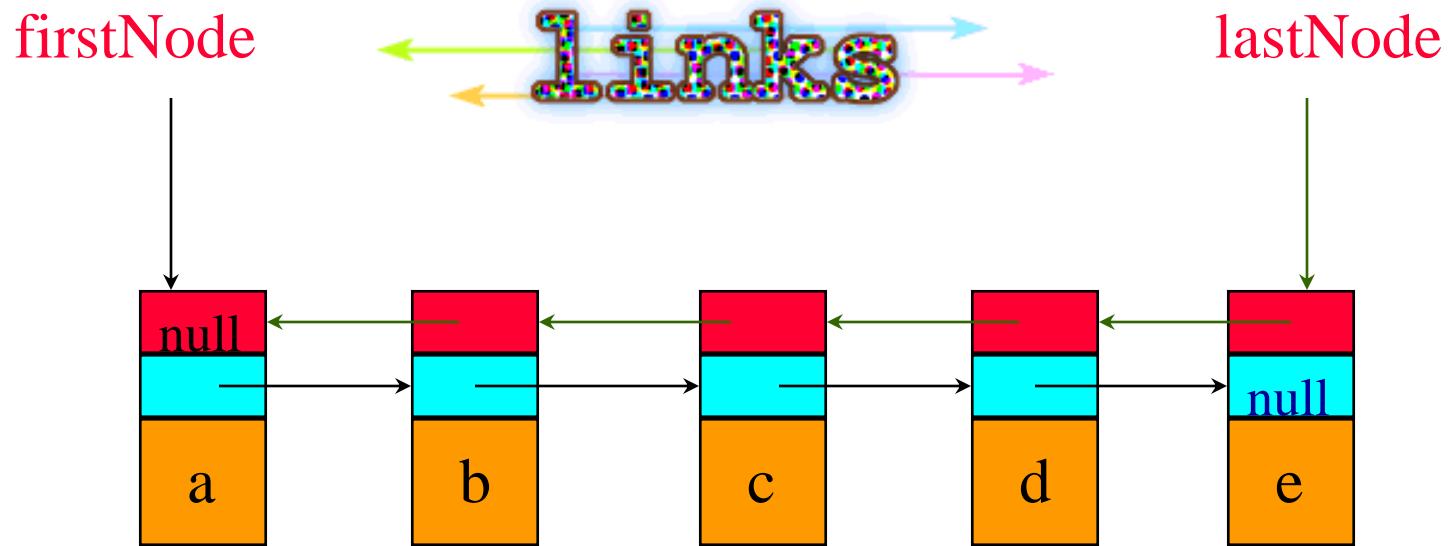


# Цагириг жагсаалт



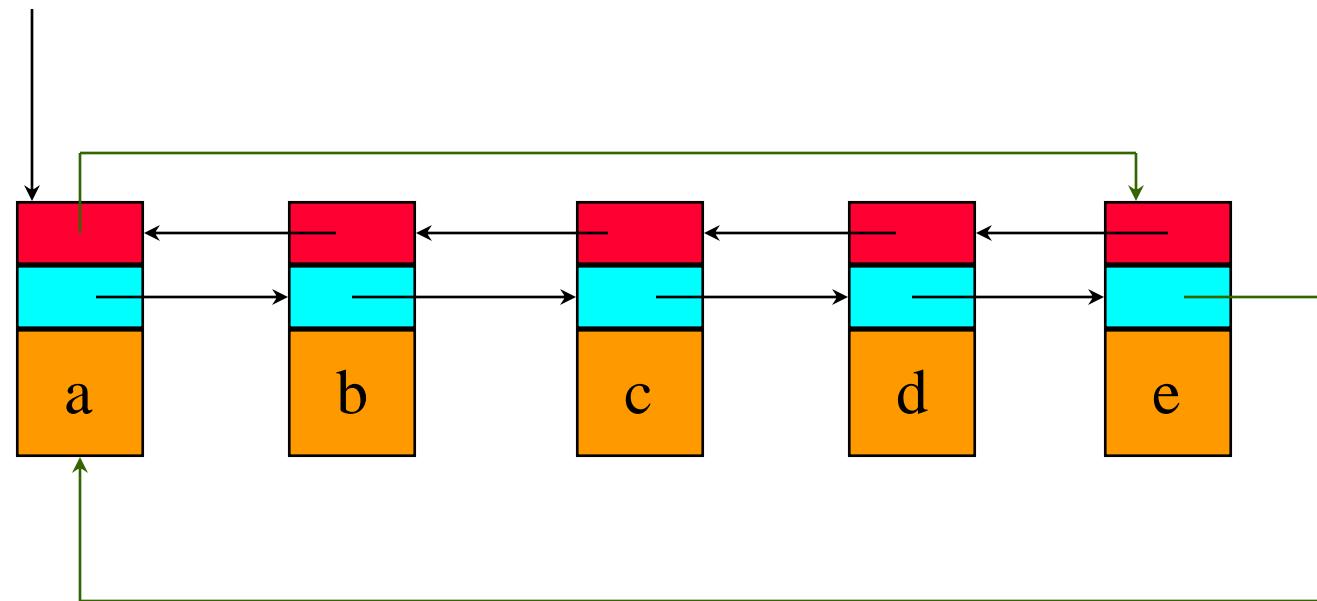


# Хос холбоост жагсаалт



# Хос холбоост цагариг жагсаалт

firstNode

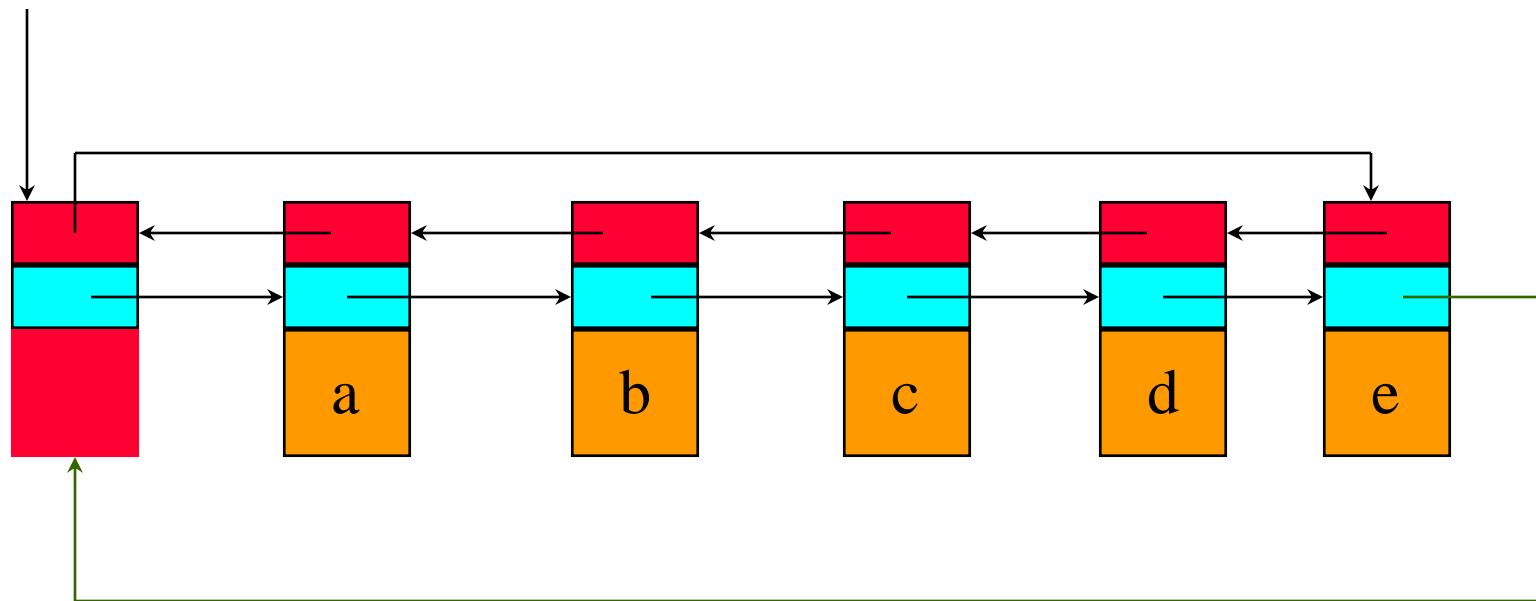




# Толгойн зангилаатай хос холбоост цагариг жагсаалт



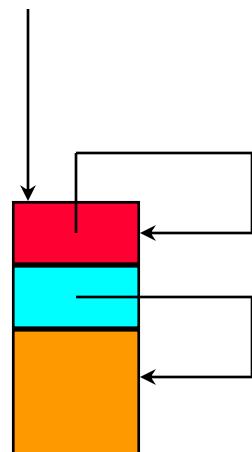
headerNode



# Толгойн зангилаатай хос холбоост цагариг хоосон жагсаалт



headerNode





# java.util.LinkedList



- Шугаман жагсаалтын холбоост хэрэгжүүлэлт.
- Толгойн зангилаатай хос холбоост цагариг жагсаалт.
- **LinearList** –ийн бүх аргаас гадна олон зүйл энд бий.

# Дууриамал Заагч



# Java -ийн Заагчийн дутагдал

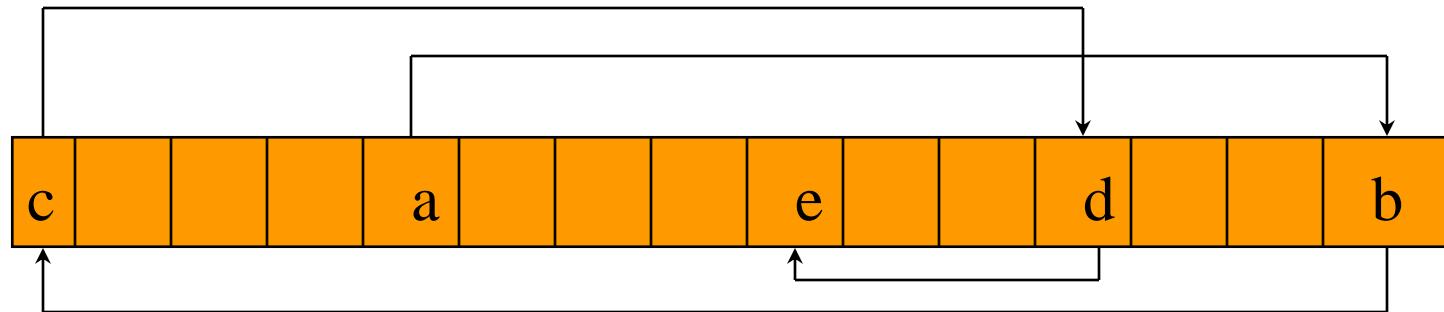


- Зөвхөн өгөгдлийн дотоод бүтцэт ашиглагддаг.
- Өгөгдлийн бүтцийг сэргээхэд дараалалжуулалт(serialization) хэрэгтэй болдог.
- Арифметик биш

# Жишээ

- Дискийн халгалах байгууламж – FAT  
(File Allocation Table)
- Интернетээр өгөгдөл дамжуулах –  
TCP/IP Protocol

# Дууриамал-Заагч Ойн байршил



Өгөгдлийн бүтэц ойд массив хэлбэртэй, түүний байршил бүрт **element** (**Object** төрөл) болон **next** (**int** төрөл) гэсэн талбаруудтай.

# Зангилааны дурслэл

```
package dataStructures;
```

```
class SimulatedNode
```

```
{
```

```
// өгөгдөл гишүүд
```

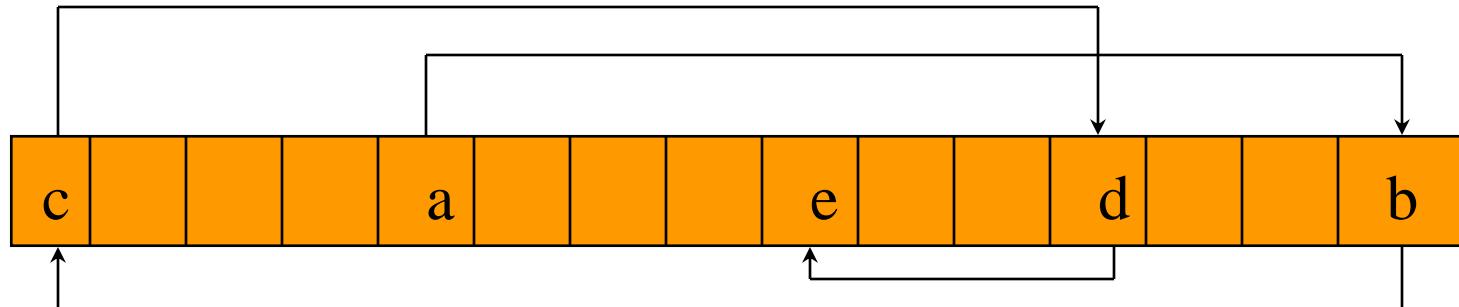
```
Object element;
```

```
int next; //Анхаарлаа энд хандуулна уу
```

```
// байгуулагч энд бичигдэнэ
```

```
}
```

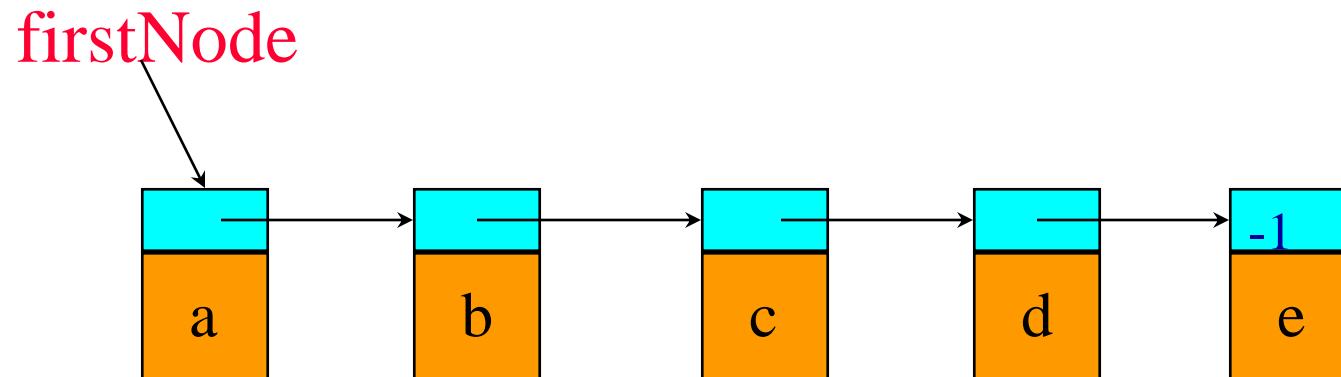
# Энэ бүхэн яаж харагдах вэ



| next<br>element | 11 |   |   |   | 14 |   |  |  | -1 |  |  | 8  |  | 0  |
|-----------------|----|---|---|---|----|---|--|--|----|--|--|----|--|----|
|                 | c  |   |   |   | a  |   |  |  | e  |  |  | d  |  | b  |
|                 | 0  | 1 | 2 | 3 | 4  | 5 |  |  | 8  |  |  | 11 |  | 14 |

firstNode = 4

# Зурагдаа холбоост дүрслэлтэй адил байна





# Ойн удирдлага

Холбоос (Java эсхүл дууриамал заагч) дараах зүйлийг шийдэхийг шаардана:

- ашиглагдаагүй ойг хянах (ө.х, санах байгууламжийн нөөц)
- зангилаа хуваарилах  
**Java -д new гэсэн арга бий**
- ашиглагдахгүй болсон зангилааг чөлөөлөх  
**Java хаягдал цуглуулах механизмтай**



# Хаягдал цуглуулах



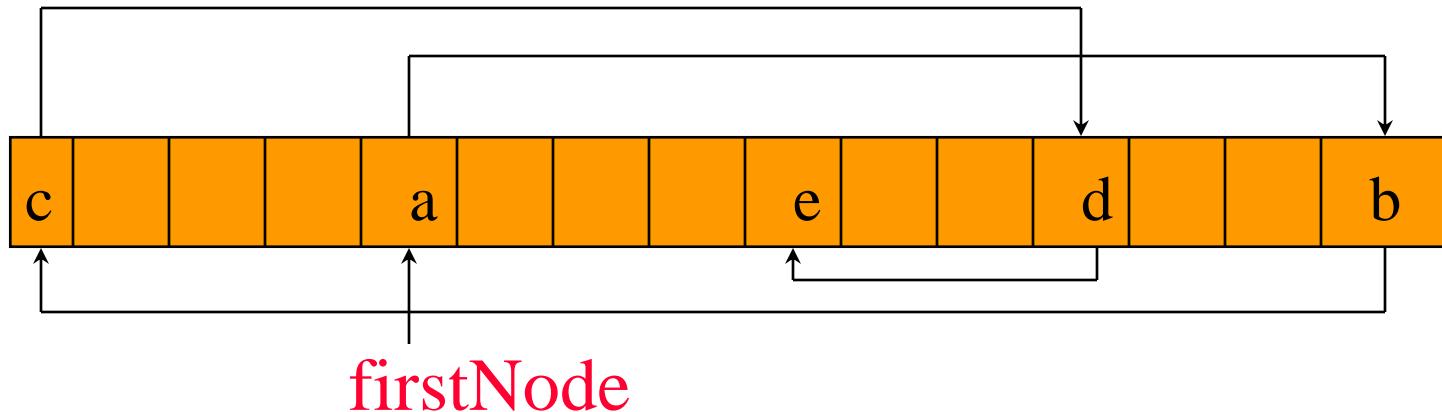
Систем ашиглагдаагүй зангилаа/ойг шүүрдэж санах байгууламжийн чөлөөт нөөцөд буцаадаг. (**сүүдрийн горимонд ажиллах !**)

Үүнийг хоёр/гурван алхмаар хийдэг:

- Ашиглаагүй зангилааг тэмдэглэх.
- Сул зайд нягтруулах (заавал биш). – дискийн дефрагментацтай төстэй
- Чөлөөлөгдсөн занилааг санах байгууламжийн нөөцөд шилжүүлэх.



# ТЭМДЭГЛЭХ

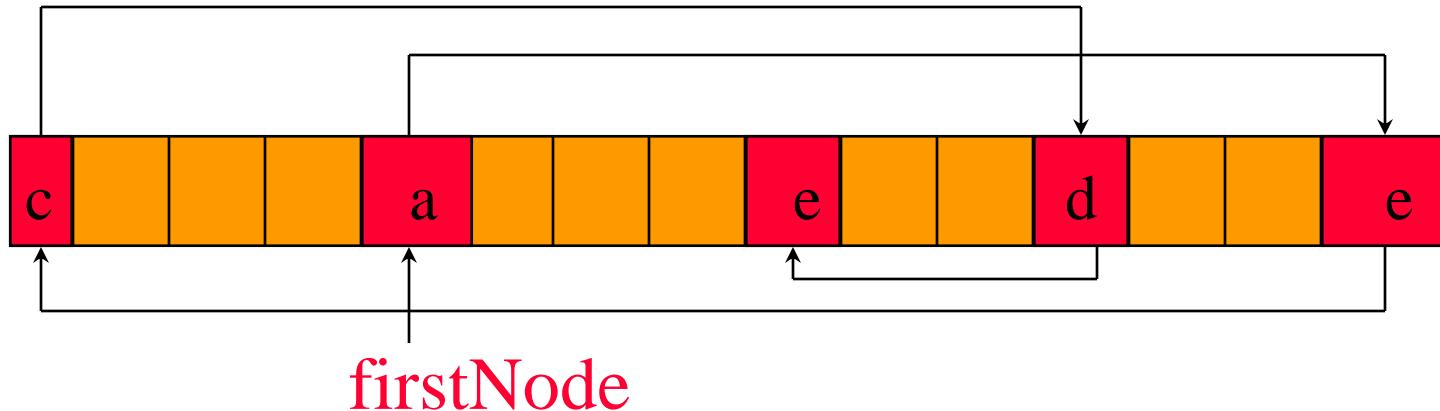


Бүх зангилааг тэмдэггүй болго (бүх тэмдэгийн битүүдийг false болгох).

Хаяг агуулсан хувьсагч бүрээс эхлэх бүх заагчийг даган очсон зангилаа бүрээ тэмдэглэнэ.



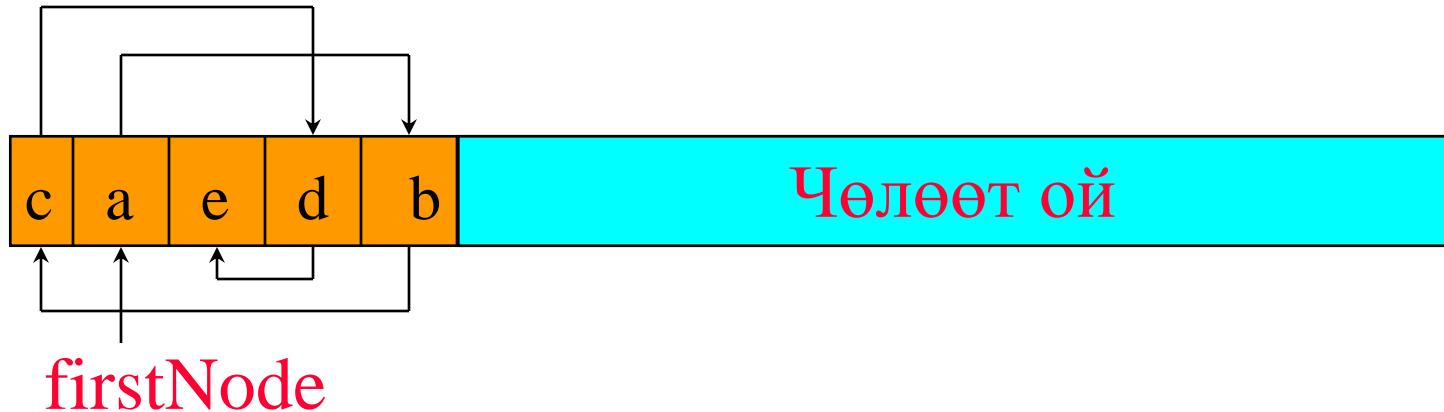
# ТЭМДЭГЛЭХ



**firstNode** –ээс эхлэн **firstNode** –оос хүрж болох бүх зангилааг тэмдэглэнэ.

Үүнийг хаягийн бүх хувьсагчийн хувьд давтана.

# Нягтруулах



Бүх тэмдэглэгдсэн зангилааг (ө.х.,  
ашиглагдаж байгаа зангилаанууд) ойн  
аль нэг төгсгөл рүү шилжүүлэх,  
шаардлагатай заагч нарыг өөрчлөх.

# Суларсан ойг санах байгууламжийн нөөцөд хийх



Тэмдэглэгдээгүй зангилааг ойд шүүнэ (хэрвээ нягтруулалт хийгдээгүй бол), эсхүл нэгэн чөлөөт блокыг (сул ой байгаа бол) нөөцөөд хийнэ.

## ⊕ Хаягдал цуглуулахын давуу тал ⊕

- Програм зохиогч суларсан зангилааг чөлөөлөхөд санаа тавих хэрэггүй.
- Гэхдээ хэрэггүй болсон объектыг илэрхийлж байгаа хувьсагчийг **null** болгохоос нааш хаягдал цуглуулалт ажиллахгүй.

## ⊕ Хаягдал цуглуулахын давуу тал ⊕

- Ой ихтэй компьютер дээр програм хурдан ажиллах боломжтой.

## - Хаягдал цуглуулахын сул тал -

- Хаягдал цуглуулах хугацаа ойн хэмжээтэй шууд хамааралтай (сул ойн хэмжээтэй хамаагүй).

# Хаягдал цуглуулах алтернатив арга

Зангилааг чөлөөлөх/суллах аргатай болох.

Ө.х., C++ -ийн **delete** арга

Ингэснээр чөлөөлөгдсөн зангилаа үргэлж санах байгууламжийн нөөцөд орох болно.

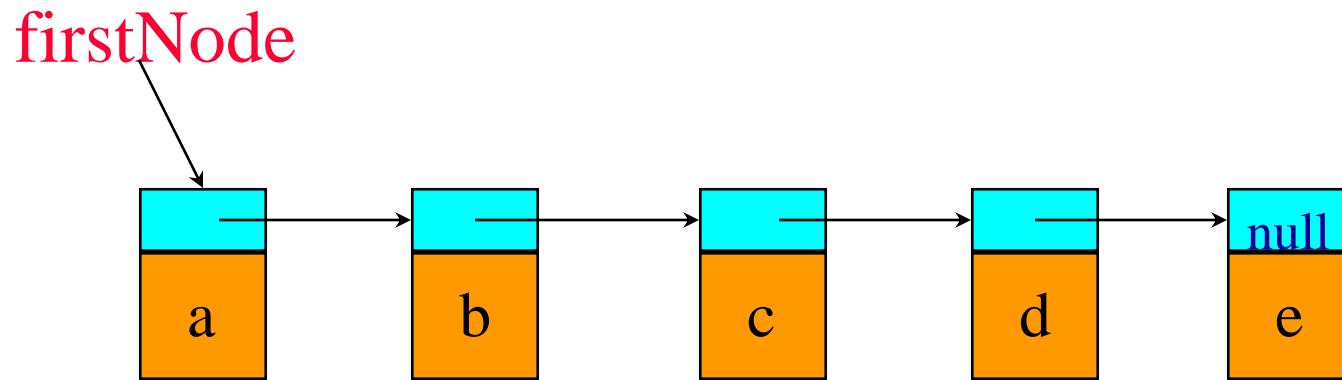
## ✚ Алтернатив аргын давуу тал ✚

- Зангилааг чөлөөлөх хугацаа нийт ойн хэмжээ бус, чөлөөлөгдсөн зангилааны тоотой пропорционал байдаг.

## - Алтернатив аргын сул тал -

- Хэрэглэгч өгөгдлийн бүтцийн зангилааг чөлөөлөх аргыг бичих хэрэгтэй.
- Зангилааг чөлөөлөхөд зарсан хугацаа дахин ашиглагдахгүй.
- Ойн хэмжээг нэмэгдүүлэх нь программын ажиллах хугацааг богиносгодрggүй.

# Зангилаа бүхэн ижил хэмжээтэй бол санах байгууламжийн нөөцийн зохион байгуулалт



- Чөлөөлөгдсөн зангилааны гинжийг зхицуулах
- Гинжний эхнээс хуваарилах
- Чөллөгдсөн зангилааг гинжний эхэнд нэмэх

# Дууриамал Загчийн Ойн Удирдлага

```
/** дууриамал заагчийн ойг удирдах класс */
package dataStructures;
import utilities.*;
public class SimulatedSpace1
{
    // өгөгдөл гишүүд
    private int firstNode;
    SimulatedNode [] node; // багцад харагдана

    // байгуулагч болон бусад аргууд
}
```

# Байгуулагч



```
public SimulatedSpace1(int numberOfNodes)
{
    node = new SimulatedNode [numberOfNodes];

    // зангилааг үүсгэж гинжинд холбох
    for (int i = 0; i < numberOfNodes - 1; i++)
        node[i] = new SimulatedNode(i + 1);

    // массив болон гинжийн сүүлийн зангилаа
    node[numberOfNodes - 1] = new SimulatedNode(-1);
    // firstNode –ийн анхны утга 0
}
```

# Зангилаа Хуваарилах

```
public int allocateNode(Object element, int next)
{
    // Хоосон зангилаа хуваарилж, талбаруудын утгыг тогтоох.
    if (firstNode == -1)
    {
        // Энд бичигдэх кодыг орхилоо
    }

int i = firstNode;           // анхны зангилааг хуваарилах
firstNode = node[i].next;   // firstNode дараагийн чөлөөт
зангилааг заана
node[i].element = element;
node[i].next = next;
return i;
}
```

# Зангилааг Чөлөөлөх

```
public void deallocateNode(int i)
{
    // Зангилаа i -г чөлөөлөх.
    // i -г хоосон жагсаалтын эхний зангилаа болгох
    node[i].next = firstNode;
    firstNode = i;

    // элементийн хаягаар илэрхийлэгдэх орон зайг хаягдалд өгч
    // чөлөөлөх
    node[i].element = null;
}
```

## ⊕    Дууриамал Заагч    ⊕

- Дахин холбоос үүсгэхгүйгээр зангилаануудын гинжийг хуваарилж болно.
- Гинжний эхлэл болон төгсгөлийн зангилаа мэдэгдэж байвал зангилааны гинжийг богино хугацаанд чөлөөлж болно.

# 🔥 Дууриамал Заагч 🔥

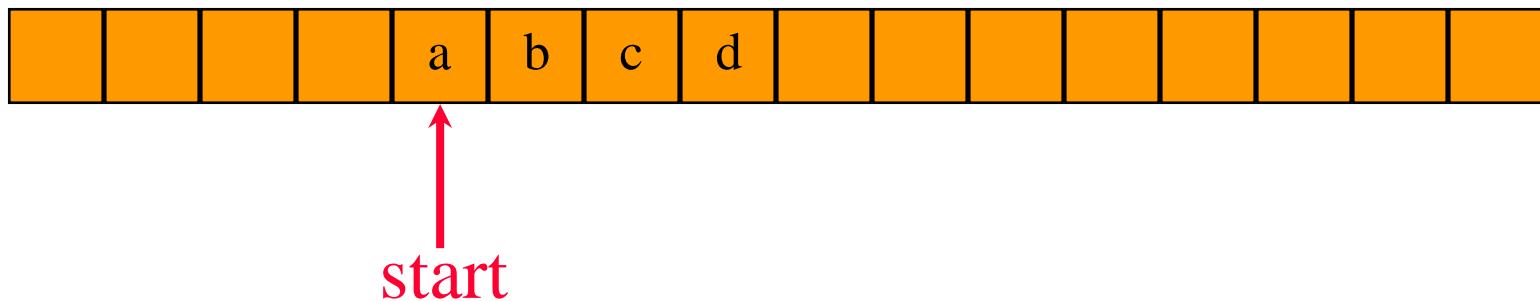
- Java –ийн хаягаас дууриамал заагч илт давуу талтай болох нь харагдахгүй байвал ашиглах хэрэггүй.

# Массив



# Java, C, C++ -д нэг хэмжээст массивыг дүрслэх

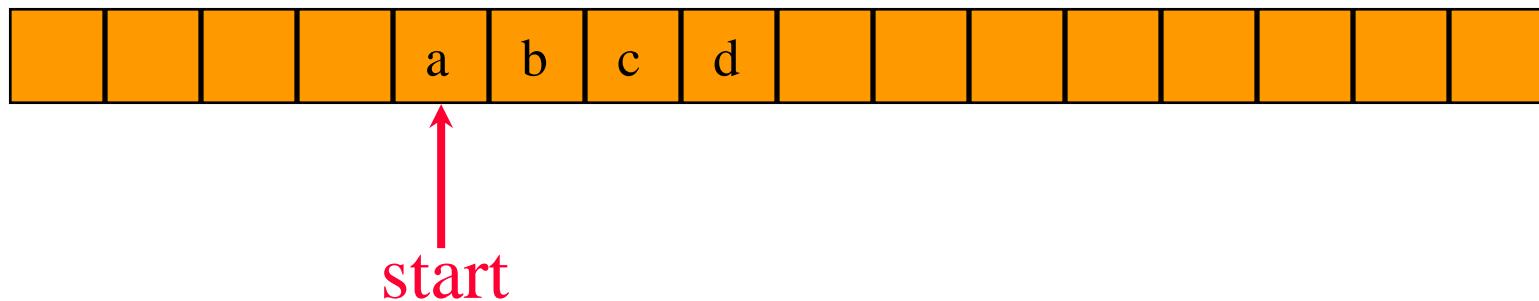
## Ой



- Нэг хэмжээст массив  $x = [a, b, c, d]$
- Ойн дараалсан байрлалд тусна
- байршил( $x[i]$ ) = start + i

# НЭМЭЛТ ОЙН ХЭМЖЭЭ

Ой



Нэмэлт ой = 4 байт ( **start** )  
+ 4 байт ( **x.length** )  
= 8 байт

( **x** массивын элементүүдийг хадгалах ой  
ороогүй )

# Хоёр хэмжээст массив

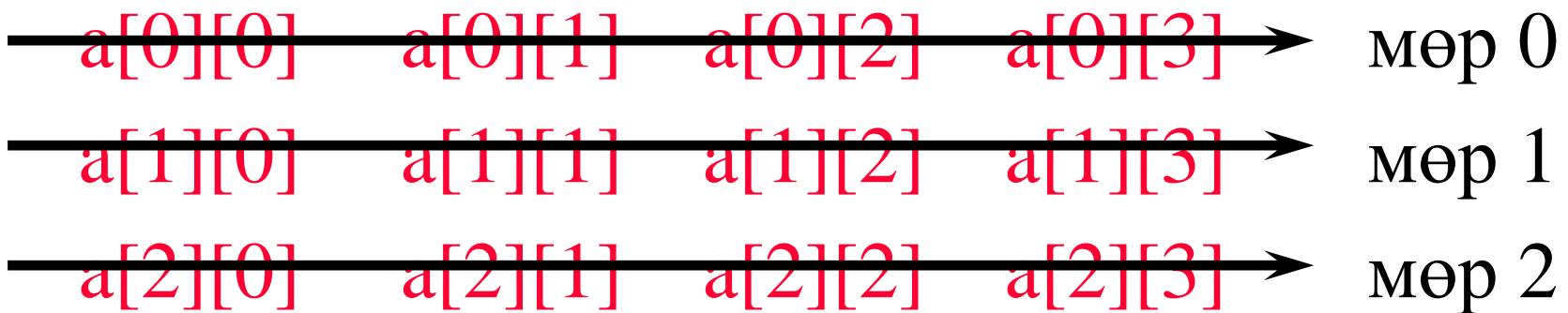
Хоёр хэмжээст массив **a** –г зарлах:

**int [][]a = new int[3][4];**

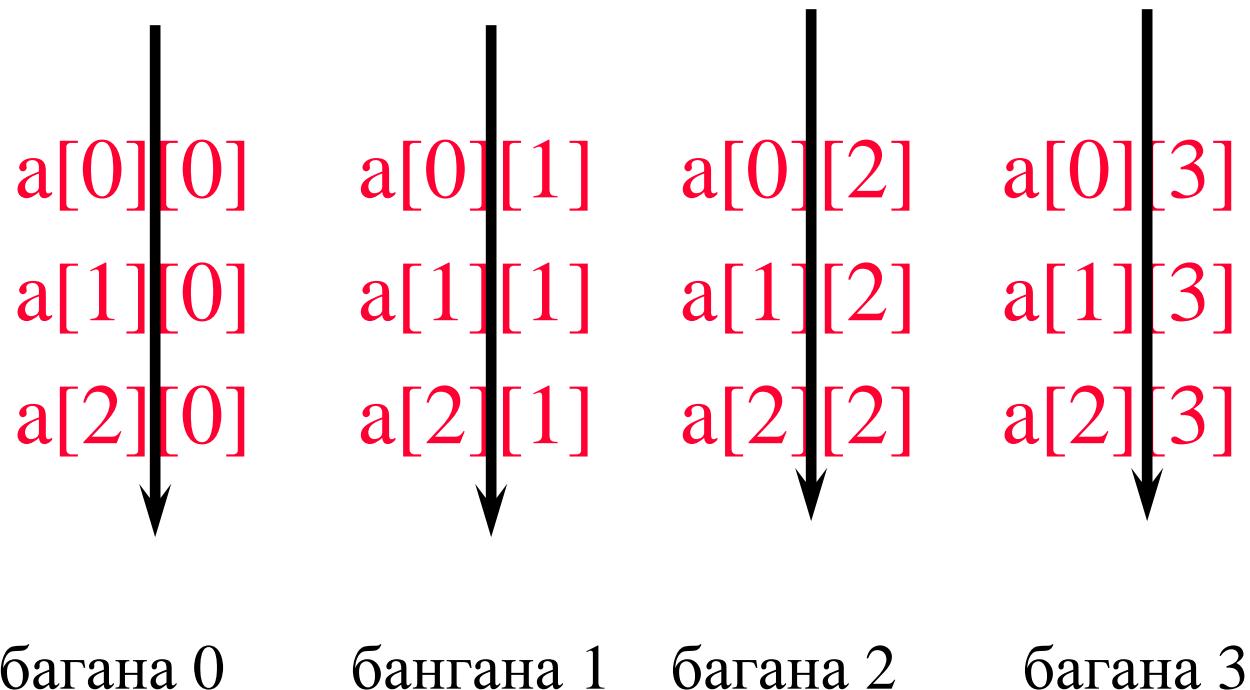
Гэж зарлагдсан массивын элементүүдийг хүснэгт байдлаар үзүүлбэл:

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| <b>a[0][0]</b> | <b>a[0][1]</b> | <b>a[0][2]</b> | <b>a[0][3]</b> |
| <b>a[1][0]</b> | <b>a[1][1]</b> | <b>a[1][2]</b> | <b>a[1][3]</b> |
| <b>a[2][0]</b> | <b>a[2][1]</b> | <b>a[2][2]</b> | <b>a[2][3]</b> |

## Хоёр хэмжээст массивын мөрүүд



# Хоёр хэмжээст массивын баганууд



# Java, C, C++ -д хоёр хэмжээст массивыг дүрслэх

2-хэмжээст массив **X**

a, b, c, d

e, f, g, h

i, j, k, l

2D массивыг 1D мөрүүдийн массив

x = [row0, row1, row2]

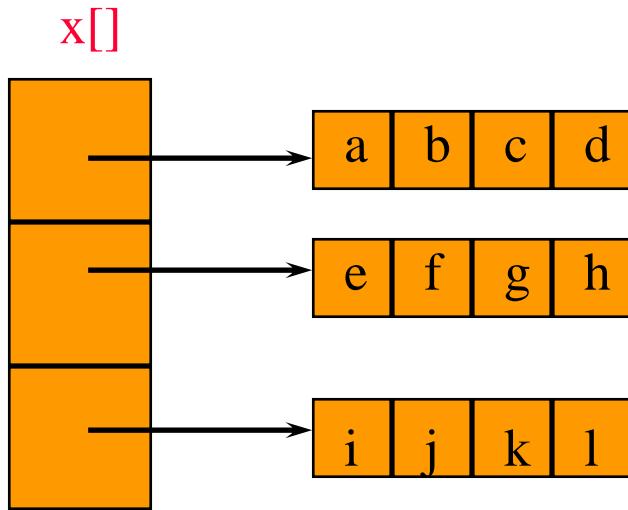
row0 = [a,b, c, d]

row1 = [e, f, g, h]

row2 = [i, j, k, l]

гэж **4** 1D массив байдлаар хадгалж болно

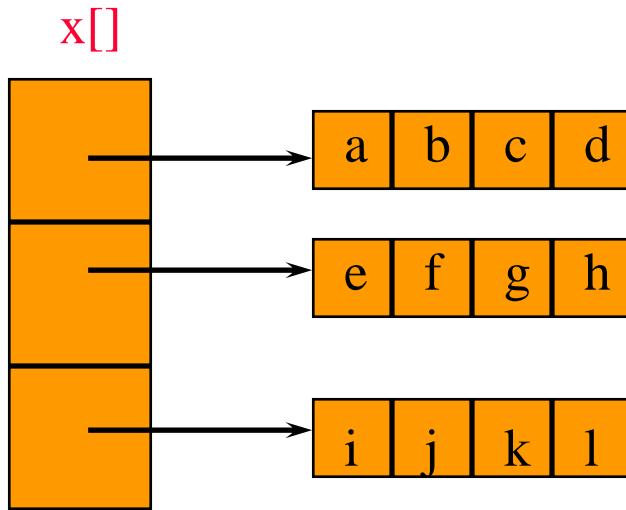
# Java, C, C++ -д хоёр хэмжээст массивыг дүрслэх



`x.length = 3`

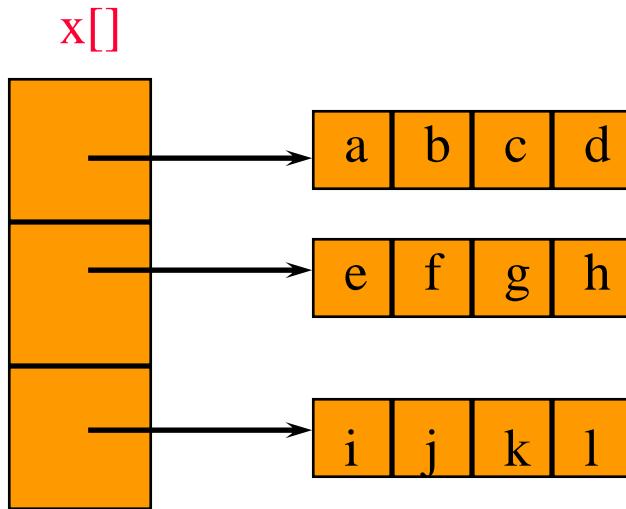
`x[0].length = x[1].length = x[2].length = 4`

# НЭМЭЛТ ОЙН ХЭМЖЭЭ



Нэмэлт ой = 4 1D массивын нэмэлт ой  
=  $4 * 8$  байт  
= 32 байт  
= (мөрийн тоо + 1) x 8 байт

# Java, C, C++ -д массивыг дүрслэх



- Ийм дүрслэлийг **массивуудын-массив** дүрслэл гэнэ.
- 4 1D массивт 3, 4, 4, 4 гэсэн хэмжээстэй дараалсан ой шаардагдана.
- **number of rows** хэмжээтэй 1 блок + **number of columns** хэмжээтэй **number of rows** блокууд

# Мөрийг чухалчилсан тусгал

- 3 x 4 массивын жишээ:

a b c d  
e f g h  
i j k l

- Элементүүдийг мөр мөрөөр нь цуглуулж 1D **y** массивт хувиргах.
- Элемеэнтүүд мөрөндөө зүүнээс баруун тийш.
- Мөрүүд дээрээс доош цуварна.
- Тэгвэл **y[] = {a, b, c, d, e, f, g, h, i, j, k, l}**

|       |       |       |     |       |  |  |
|-------|-------|-------|-----|-------|--|--|
| row 0 | row 1 | row 2 | ... | row i |  |  |
|-------|-------|-------|-----|-------|--|--|

# Элемент $x[i][j]$ олох



- $x$  массив  $r$  мөр,  $c$  баганатай гэвэл
- Мөр бүр  $c$  элементтэй
- $i$  дэх мөрийн өмнө  $i$  мөр орно
- Ө.х.  $ic$  элемент  $x[i][0]$  –ий өмнө орно
- Ө.х.  $x[i][j]$  1D массивын  $ic + j$  баршилд хадгалагдана

# НЭМЭЛТ ОЙН ХЭМЖЭЭ

|       |       |       |     |       |  |  |
|-------|-------|-------|-----|-------|--|--|
| row 0 | row 1 | row 2 | ... | row i |  |  |
|-------|-------|-------|-----|-------|--|--|

4 байт ( `start` - 1D массив ) +  
4 байт ( `length` - 1D массив ) +  
4 байт ( `c` (баганын тоо)  
= 12 байт

(мөрийн тоо = `length /c`)

# Сул тал

rc урттай дараалсан ой хэрэгтэй.

week5

# Баганыг чухалчилсан тусгал

a b c d

e f g h

i j k l

- Элементүүдийг багана баганаар нь цуглуулж 1D **y** массивт хувирагх.
- Элементүүд баганадаа дээрээс доош.
- Баганууд зүүнээс баруун тийш цуварна.
- Тэгвэл **y = {a, e, i, b, f, j, c, g, k, d, h, l}**

# Матриц

Утгын хүснэгт. Мөр, баганатай, 0 биш 1-ээс эхэлж дугаарлагдана.

a b c d      мөр 1

e f g h      мөр 2

i j k l      мөр 3

- $x[i][j]$  оронд  $x(i,j)$  тэмдэглэгээг хэрэглэе.
- Матрицыг 2D массиваар дурсэлж болно.

# Матрицад 2D массив хэрэглэхийн дутагдал

- Индекс 1 –ээр өөрчлөгддөг.
- Java -ийн массивт матрицын **add**, **transpose**, **multiply**, зэрэг үйлдлүүд байхгүй.
  - **x** , **y** 2D массив бол. **x + y**, **x -y**, **x \* y**, гэж болохгүй.
- Объект хандалтат **Matrix** классыг хөгжүүлснээр матрицын бүх үйлдлийг шийдэж болно. Сурах бичгээс хар.

# Диагналь матриц

**n x n** матрицын тэг биш бүх утгууд  
диагналь дээр байрлана.

# Диагналь матриц

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{matrix}$$
A 4x4 matrix with red numbers. The main diagonal elements are 1, 2, 3, and 4. A blue arrow points from the top-left element (1) towards the bottom-right element (4), indicating the direction of the main diagonal.

- $x(i,j)$  диагналь дээр байхын тулд  $i = j$
- $n \times n$  диагналийн элементийн тоо  $n$
- Диагналийн бус элемент тэг байна
- $n^2$  элементийн оронд зөвхөн диагналийн элементийг хадгална

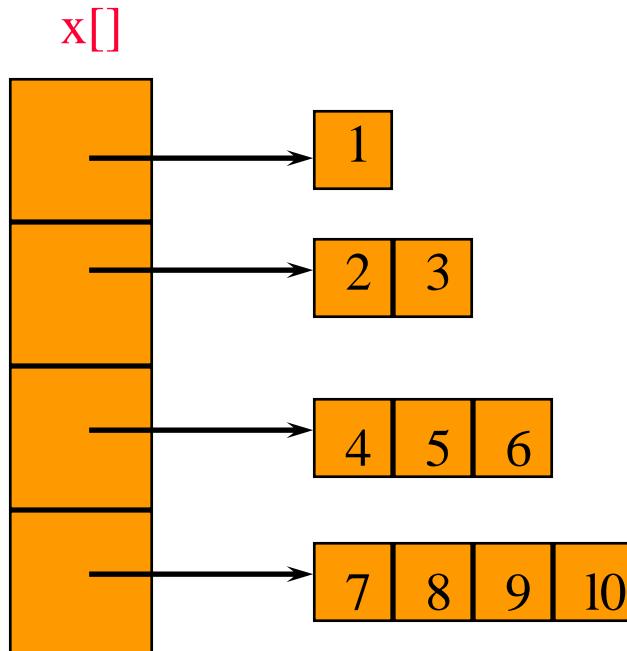
# Доод гурвалжин матриц

$n \times n$  матрицын тэг биш элементүүд диагнль болон түүнээс доош байрлана.

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 10 \end{matrix}$$

- $x(i,j)$  доод гурвалжинд байхын тулд  $i \geq j$ .
- Доод гурвалжингийн элементийн тоо  $1 + 2 + \dots + n = n(n+1)/2$ .
- Зөвхөн доод гурвалжинг хадгална

# Массивуудын массив дүрслэл



Ердийн биш 2D массив ... мөрүүд дэх  
элементийн тоо харилцан адилгүй байна.

# Ердийн биш массивыг ҮҮСГЭЖ, ашиглах

// хоёр хэмжээст массивын хувьсагчийг зарлахдаа

// мөрийн тоог зааж өгөх

```
int [][] irregularArray = new int [numberOfRows][];
```

// мөр бүрийн элементэд зай хуваарилах

```
for (int i = 0; i < numberOfRows; i++)
```

```
    irregularArray[i] = new int [size[i]];
```

// массивыг ердийн массивын адил ашиглах

```
irregularArray[2][3] = 5;
```

```
irregularArray[4][6] = irregularArray[2][3] + 2;
```

```
irregularArray[1][1] += 3;
```

# Доод гурвалжин массивыг 1D массиваар байршуулах

Мөрийг чухалчлах дарааллийг ашиглахдаа  
доод гурвалжинд орохгүй элементийг  
орхино.

Матрицын жишээ

1 0 0 0

2 3 0 0

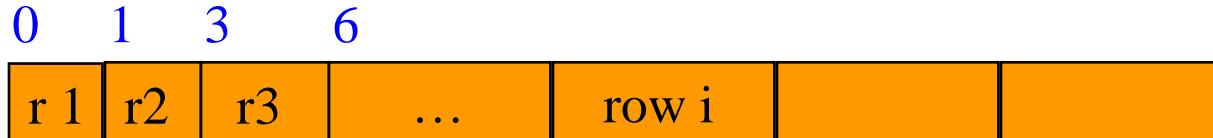
4 5 6 0

7 8 9 10

бол

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

## [i][j] элементийн индекс



- Дараалал: мөр 1, мөр 2, мөр 3, ...
- Мөр  $i$  дараах мөрүүдийн өмнө 1, 2, ...,  $i-1$
- $i$ -р мөрийн хэмжээ  $i$ .
- $i$ -р мөрөөс өмнө орсон элементийн тоо
$$1 + 2 + 3 + \dots + i-1 = i(i-1)/2$$
- Иймд элемент  $(i,j)$  1D массивын  $i(i-1)/2 + j - 1$  байршилд байна.

# Сийрэг матриц



сийрэг ... олон элемент нь тэг  
нягт ... цөөн элемент тэг

# Сийрэг матрицын жишээ

диагналь

гурвалсан диагналь

доод гурвалжин (?)

Эд бүтцийн сийрэг матриц.

1D массиваар илэрхийлж болно, элементийг олохдоо тусгалын функц ашиглана.

# Бүтцийн бус сийрэг матриц

Нислэгийн матриц.

- Нисэх буудлын дугаар  $1 - n$
- $\text{flight}(i,j) = \text{list of nonstop flights from airport } i$   
буудлаас  $j$  буудал хүртэл зогсолтгүй нисэх  
нислэгийн жагсаалт
- $n = 1000$  (гэж үзье)
- $n \times n$  массивын байж болох жагсаалт  $\Rightarrow 4$  сая  
байт
- Нийт нислэгийн тоо  $= 20,000$  (гэж үзье)
- $20,000$  нислэгийн жагсаалтанд  $\Rightarrow 80,000$  байт  
хэрэгтэй болно

# Бүтцийн бус сийрэг матриц

Веб хуудасны матриц.

Веб хуудсууд  $1 - n$

$web(i,j) = i$  -ээс  $j$  хуудас хүрэх холбоос

Веб-ийн шинжилгээ.

# Веб хуудасны матриц

- $n = 2 \text{ billion}$  (өдөрт  $1 \text{ million}$  өсч байна)
- $n \times n$  бүхэл массив  $\Rightarrow 16 * 10^{18} \text{ bytes}$  ( $16 * 10^9 \text{ GB}$ )
- Хуудас бүр  $10$  (жишээ нь) хуудастай дунджаар холбогддог
- Дунджаар нэг мөрөнд  $10$  тэг биш оролт
- Тэг биш элементийг хадгалах шаардагдах ой ойролцоогоор  $20 \text{ billion} \times 4 \text{ bytes} = 80 \text{ billion bytes}$  ( $80 \text{ GB}$ )

# Бүтцийн бус сийрэг матрицын дүрслэл

Мөрийг чухалчилсан дан шугаман жагсаалт.  
мөрийг чухаллах дарааллаар сийрэг матрицын тэг  
биш элементийг шүүрдэх  
тэг биш элемент бүрийг дараах байдлаар дүрслэх  
(row, column, value)

Энэ гурвалсан жагсаалтыг массив жагсаалт, эсхүл  
холбоост-гинжин жагсаалтаар дүрсэлнэ

# Дан шугаман жагсаалтын жишээ

0 0 3 0 4  
0 0 5 7 0  
0 0 0 0 0  
0 2 6 0 0

list =

row

$\begin{bmatrix} 1 & 1 & 2 & 2 & 4 & 4 \\ 3 & 5 & 3 & 4 & 2 & 3 \end{bmatrix}$

column

value

# Массив шугаман жагсаалтын дүрслэл

|        |        |                                                       |
|--------|--------|-------------------------------------------------------|
|        | row    | $\begin{bmatrix} 1 & 1 & 2 & 2 & 4 & 4 \end{bmatrix}$ |
| list = | column | $\begin{bmatrix} 3 & 5 & 3 & 4 & 2 & 3 \end{bmatrix}$ |
|        | value  | $\begin{bmatrix} 3 & 4 & 5 & 7 & 2 & 6 \end{bmatrix}$ |

|  |         |                                                       |
|--|---------|-------------------------------------------------------|
|  | element | $\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix}$ |
|  | row     | $\begin{bmatrix} 1 & 1 & 2 & 2 & 4 & 4 \end{bmatrix}$ |
|  | column  | $\begin{bmatrix} 3 & 5 & 3 & 4 & 2 & 3 \end{bmatrix}$ |
|  | value   | $\begin{bmatrix} 3 & 4 & 5 & 7 & 2 & 6 \end{bmatrix}$ |

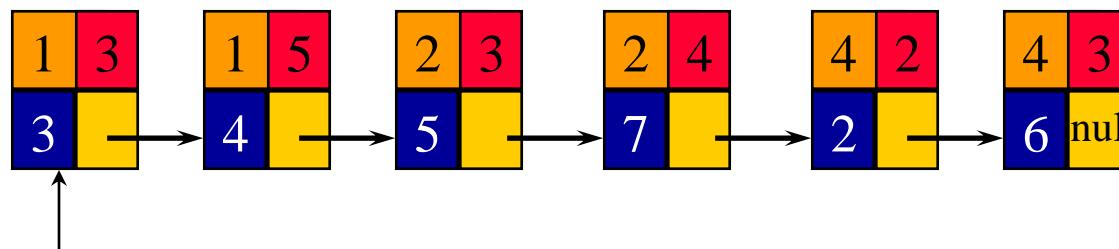
# Гинжин дүрслэл

Зангилааны бүтэц.

|       |      |
|-------|------|
| row   | col  |
| value | next |

# Энгийн гинж

|      |     |                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|------|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|      | row | <table border="1"><tr><td>1</td><td>1</td><td>2</td><td>2</td><td>4</td><td>4</td></tr><tr><td>3</td><td>5</td><td>3</td><td>4</td><td>2</td><td>3</td></tr><tr><td>3</td><td>4</td><td>5</td><td>7</td><td>2</td><td>6</td></tr></table> | 1 | 1 | 2 | 2 | 4 | 4 | 3 | 5 | 3 | 4 | 2 | 3 | 3 | 4 | 5 | 7 | 2 | 6 |
| 1    | 1   | 2                                                                                                                                                                                                                                         | 2 | 4 | 4 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 3    | 5   | 3                                                                                                                                                                                                                                         | 4 | 2 | 3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 3    | 4   | 5                                                                                                                                                                                                                                         | 7 | 2 | 6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| list | =   | column                                                                                                                                                                                                                                    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|      |     | value                                                                                                                                                                                                                                     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |



firstNode

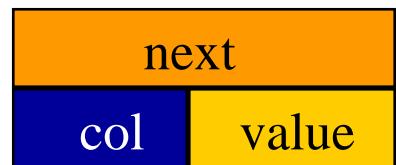
## Мөр бүрт нэг шугаман жагсаалт

0 0 3 0 4  
0 0 5 7 0  
0 0 0 0 0  
0 2 6 0 0

row1 = [(3, 3), (5,4)]  
row2 = [(3,5), (4,7)]  
row3 = []  
row4 = [(2,2), (3,6)]

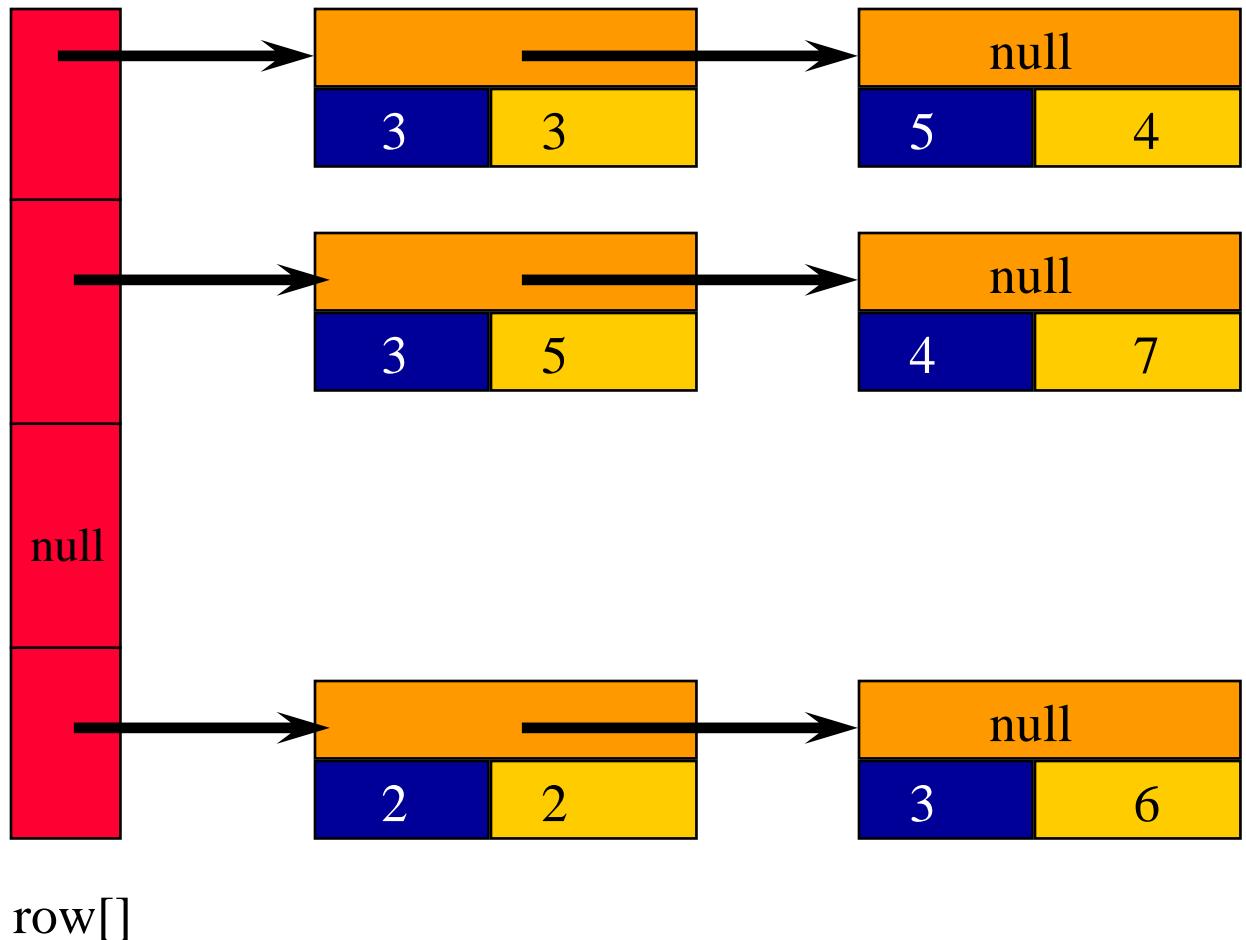
# Мөрний массивын гинж

Зангилааны бүтэц.



# Мөрний массивын гинж

0 0 3 0 4  
0 0 5 7 0  
0 0 0 0 0  
0 2 6 0 0



# Ортогналь жагсаалтын дүрслэл

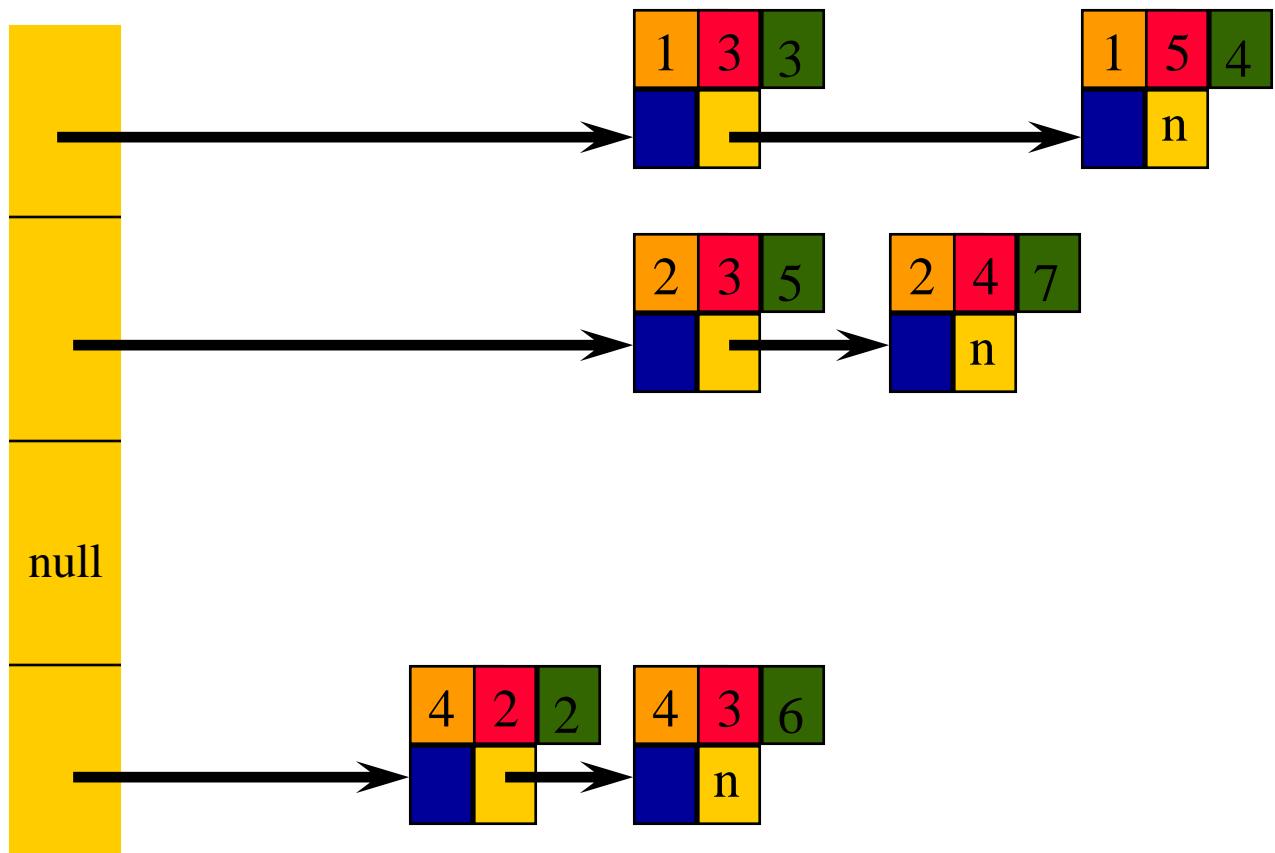
Мөр ба баганын жагсаалт.

Зангилааны бүтэц.

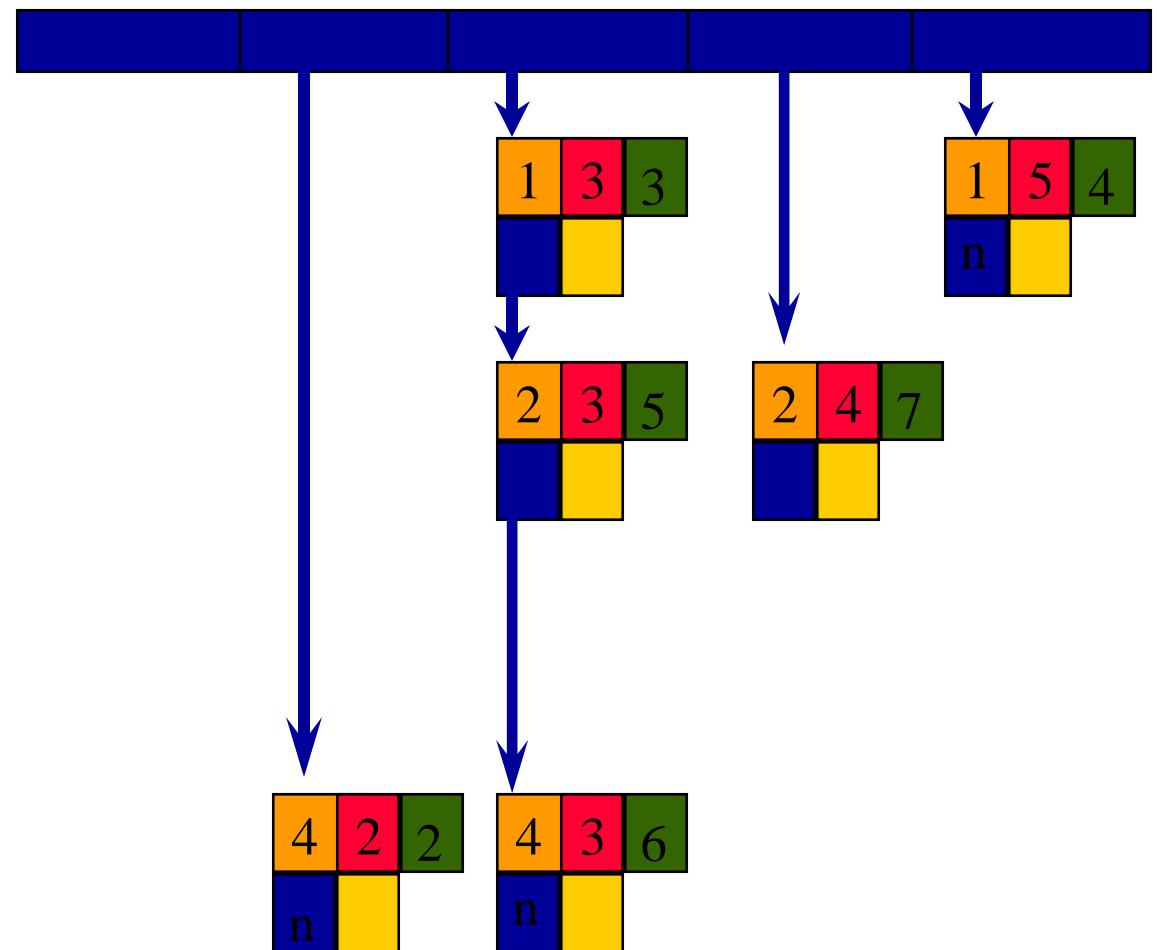
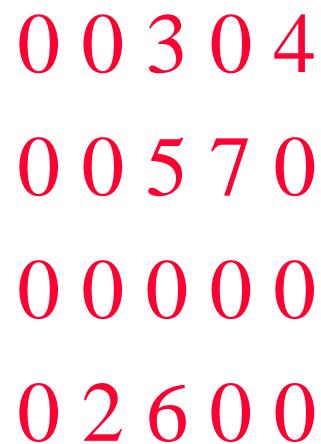
|      |      |       |
|------|------|-------|
| row  | col  | value |
| down | next |       |

# Мөрний жагсаалт

0 0 3 0 4  
0 0 5 7 0  
0 0 0 0 0  
0 2 6 0 0

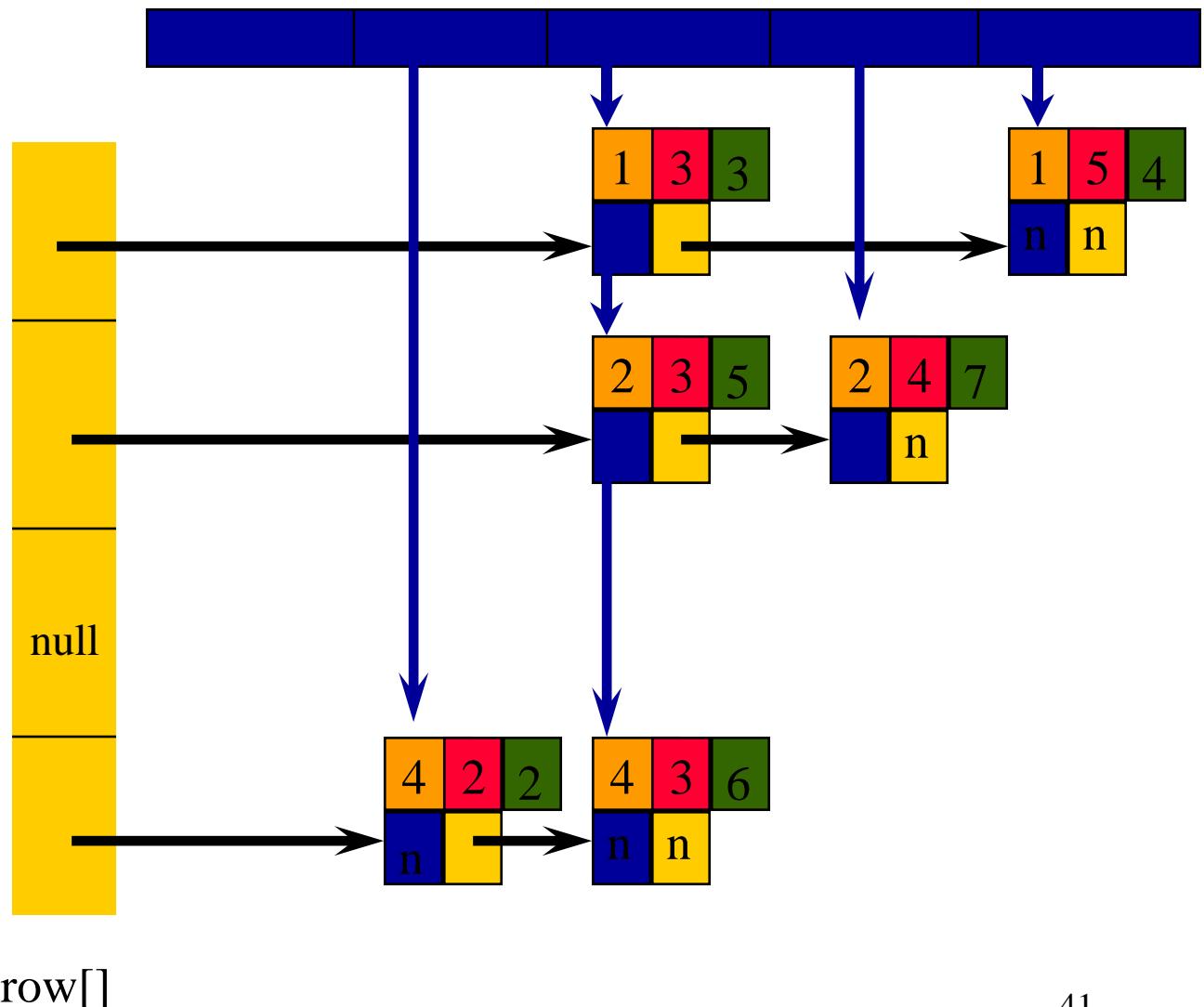


## Баганын жагсаалт



# Ортогналь жагсаалт

0 0 3 0 4  
0 0 5 7 0  
0 0 0 0 0  
0 2 6 0 0



# Хувилбарууд

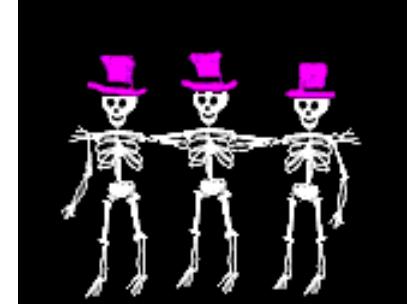
Гинжны оронд цагираг жагсаалтыг  
ашиглаж болно.

# Ойн ойролцоо хэрэгцээ

500 x 500 матриц, 1994 тэг биш элемент

2D массив  $500 \times 500 \times 4 = 1\text{million}$  байт  
Дан массив жагсаалт  $3 \times 1994 \times 4 = 23,928$  байт  
Мөр бүрт 1 гинж  $23928 + 500 \times 4 = 25,928$

# Хугацааны үзүүлэлт



Matrix Transpose

500 x 500 матриц, 1994 тэг биш элемент

2D массив

210 ms

Дан массив жагсаалт

6 ms

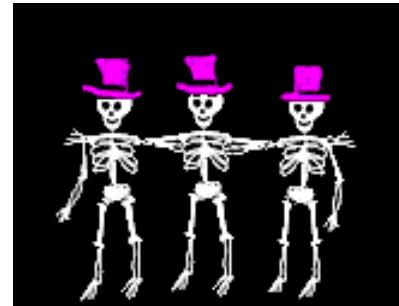
Мөр бүрт 1 гинж

12 ms

# Хугацааны үзүүлэлт

Matrix Addition.

500 x 500 матриц, 1994 ба 999 тэг биш  
элемент



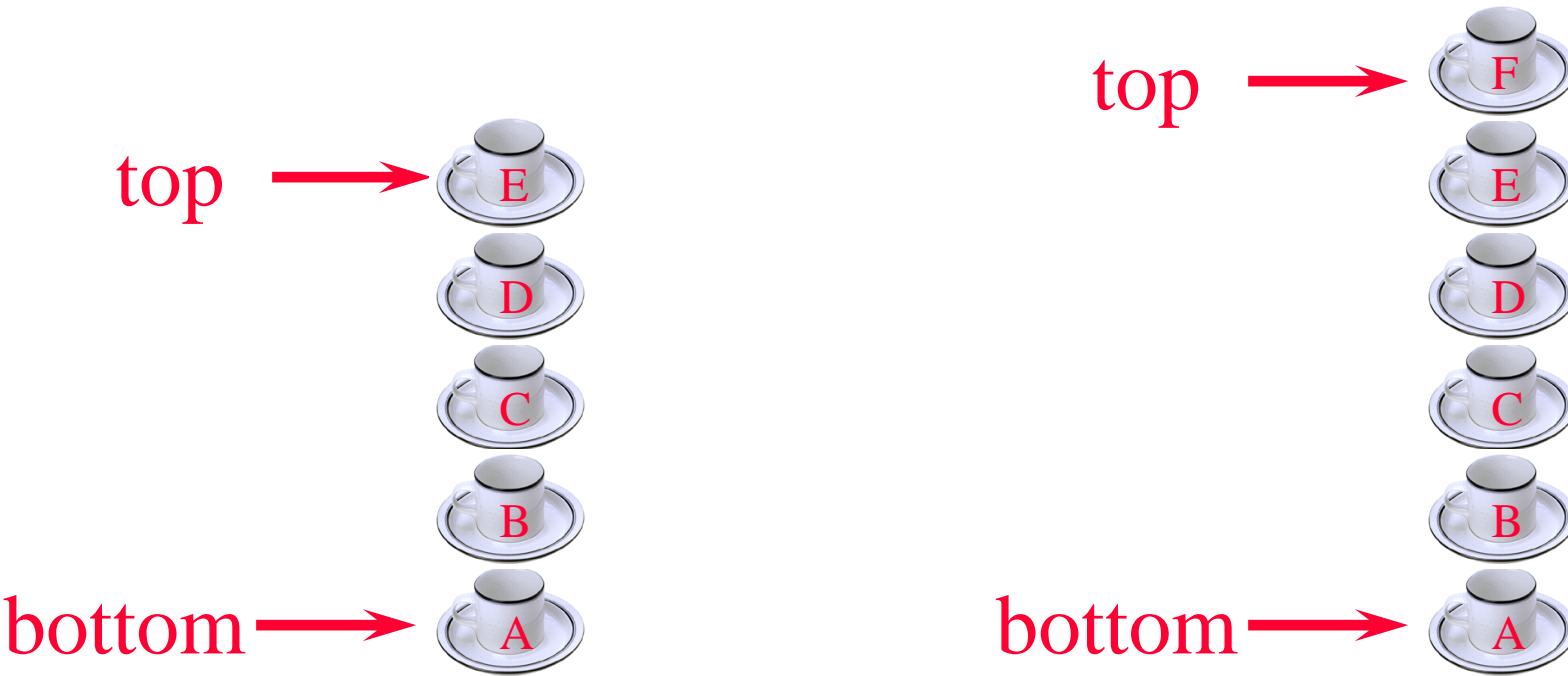
|                     |        |
|---------------------|--------|
| 2D массив           | 880 ms |
| Дан массив жагсаалт | 18 ms  |
| Мөр бүрт 1 гинж     | 29 ms  |

# Стек



- Шугаман жагсаалт.
- Нэг төгсгөлийг нь **top-орой**.
- Нөгөө төгсгөлийг нь **bottom-ёроол**.
- Нэмэлт ба хасалтыг зөвхөн **оройгоос** нь хийнэ.

# Аяганы стек



- Стект аяга нэмэх.
- Стекээс аяга авах.
- Стек бол LIFO жагсаалт.

# Интерфейс - Stack

```
public interface Stack
```

```
{
```

```
    public boolean empty();
```

```
    public Object peek();
```

```
    public void push(Object theObject);
```

```
    public Object pop();
```

```
}
```

# Хаалт хослох

- $((a+b)^*c+d-e)/(f+g)-(h+j)^*(k-l))/(m-n)$ 
  - $u$  байршлын нээх хаалтад харгалзах  $v$  байршлын хаах хаалтын хослолыг  $(u,v)$  гэж бичвэл
    - (2,6) (1,13) (15,19) (21,25) (27,31) (0,32) (34,38)
- $(a+b))^*((c+d)$ 
  - (0,4)
  - байршил 5 –ын хаах халтад харгалзах нээх хаалт алга
  - (8,12)
  - байршил 7 –ийн нээх хаалтад харгалзах хаах хаалт алга

# Хаалт хослох

- Илэрхийллийг зүүнээс баруун тийш шинжих
- Нээх хаалт тааралдвал түүний байршлыг стект нэмнэ
- Хаах хаалт тааралдвал харгалзах байршлыг стекээс авна

# ЖИШЭЭ

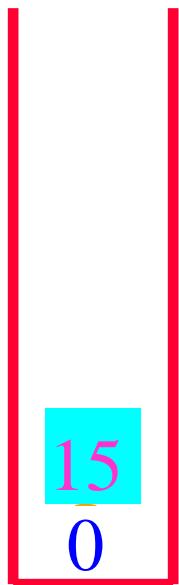
- $((a+b)^*c+d-e)/(f+g)-(h+j)^*(k-l))/(m-n)$



2  
1  
0

# Жишээ

- $((a+b)^*c+d-e)/(f+g)-(h+j)^*(k-l))/(m-n)$

|                                                                                          |       |        |
|------------------------------------------------------------------------------------------|-------|--------|
| <br>15 | (2,6) | (1,13) |
| 0                                                                                        |       |        |

# Жишээ

- $((a+b)^*c+d-e)/(f+g)-(h+j)^*(k-l))/(m-n)$

|    |
|----|
| 21 |
| 0  |

(2,6) (1,13) (15,19)

# Жишээ

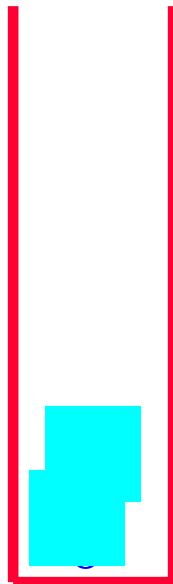
- $((a+b)^*c+d-e)/(f+g)-(h+j)^*(k-l))/(m-n)$

|    |
|----|
| 27 |
| 0  |

(2,6) (1,13) (15,19) (21,25)

# ЖИШЭЭ

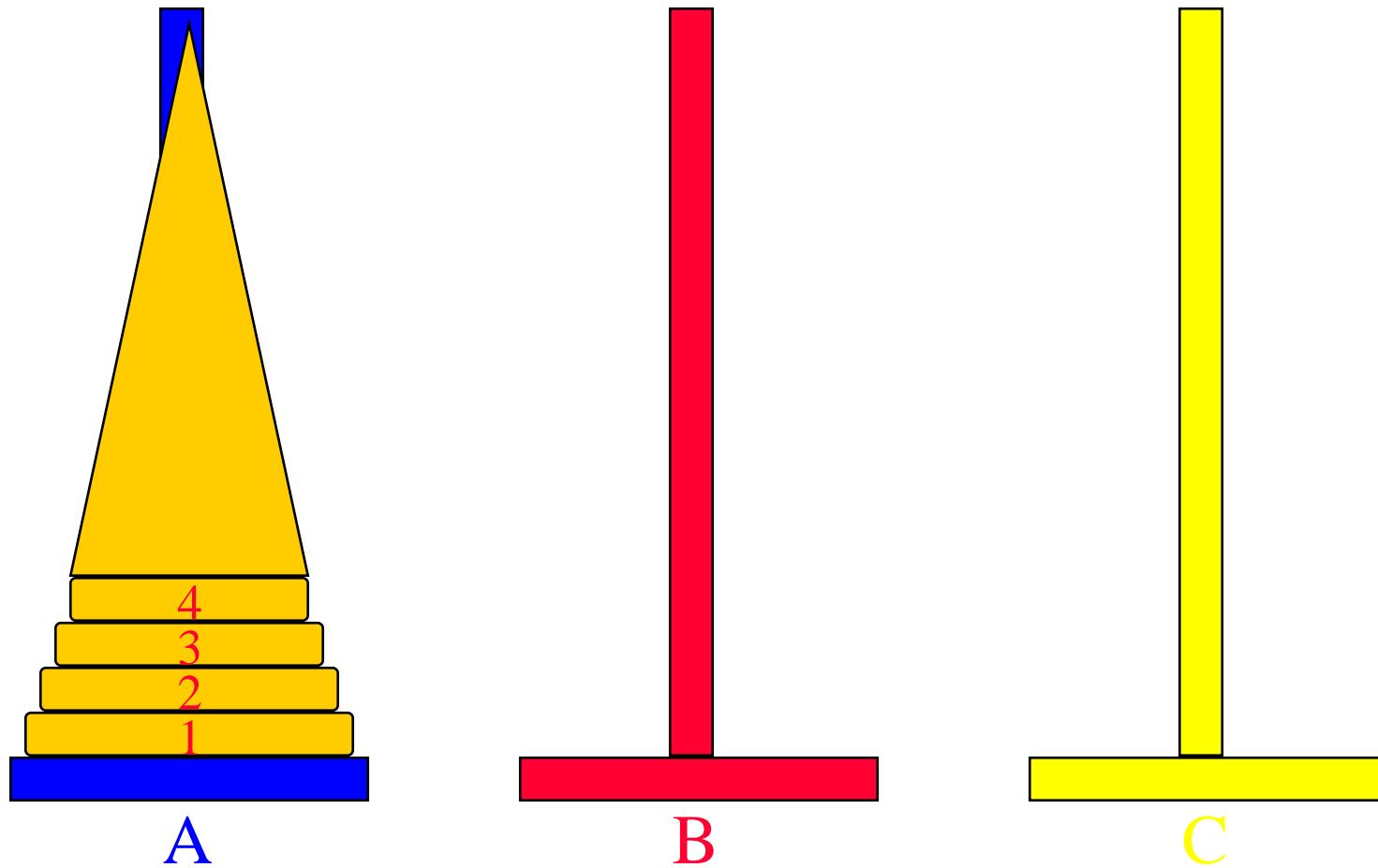
- $((a+b)^*c+d-e)/(f+g)-(h+j)^*(k-l))/(m-n)$



(2,6) (1,13) (15,19) (21,25)(27,31) (0,32)

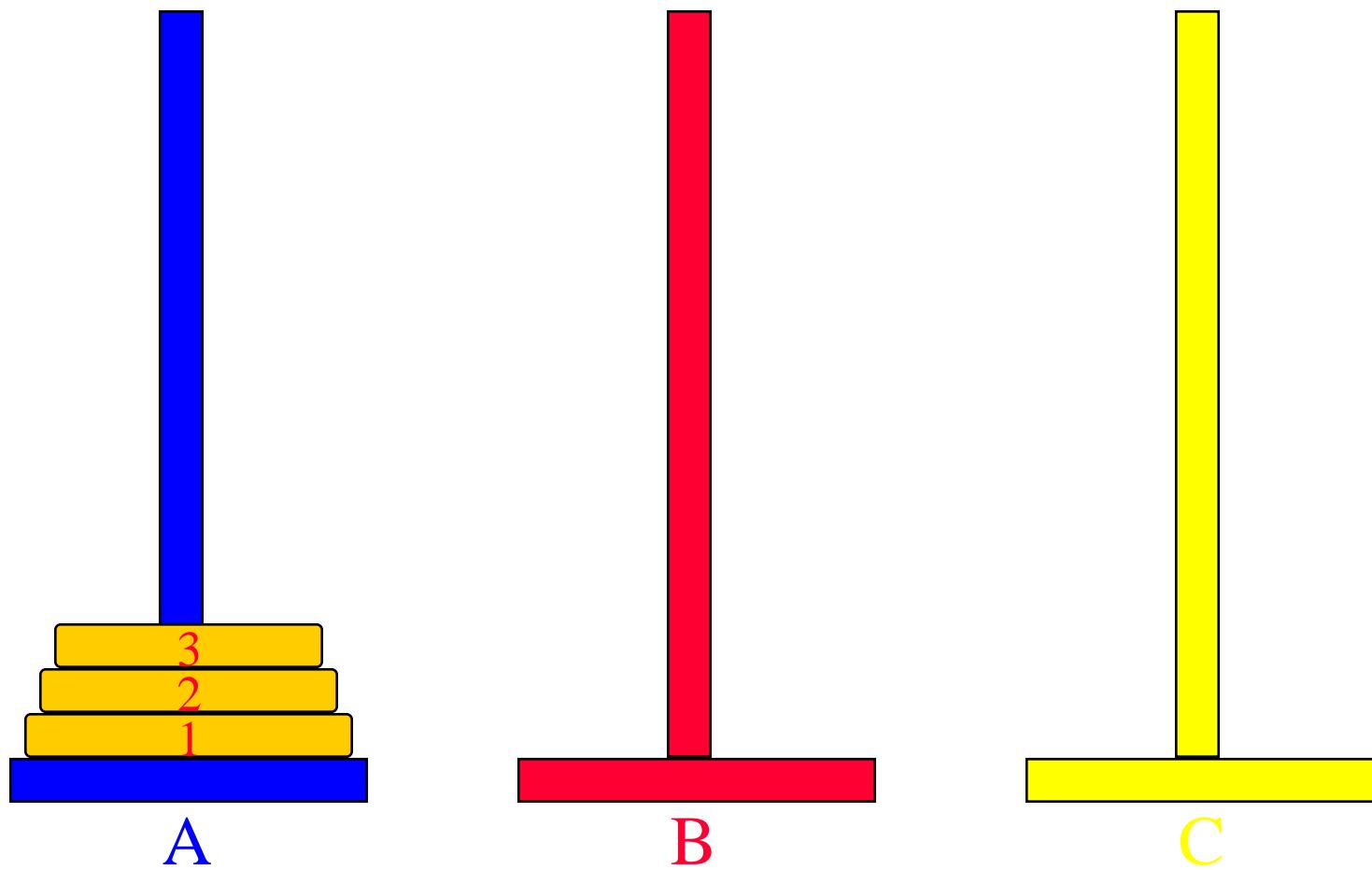
- ГЭХ МЭТ

# Ханойн цамхаг



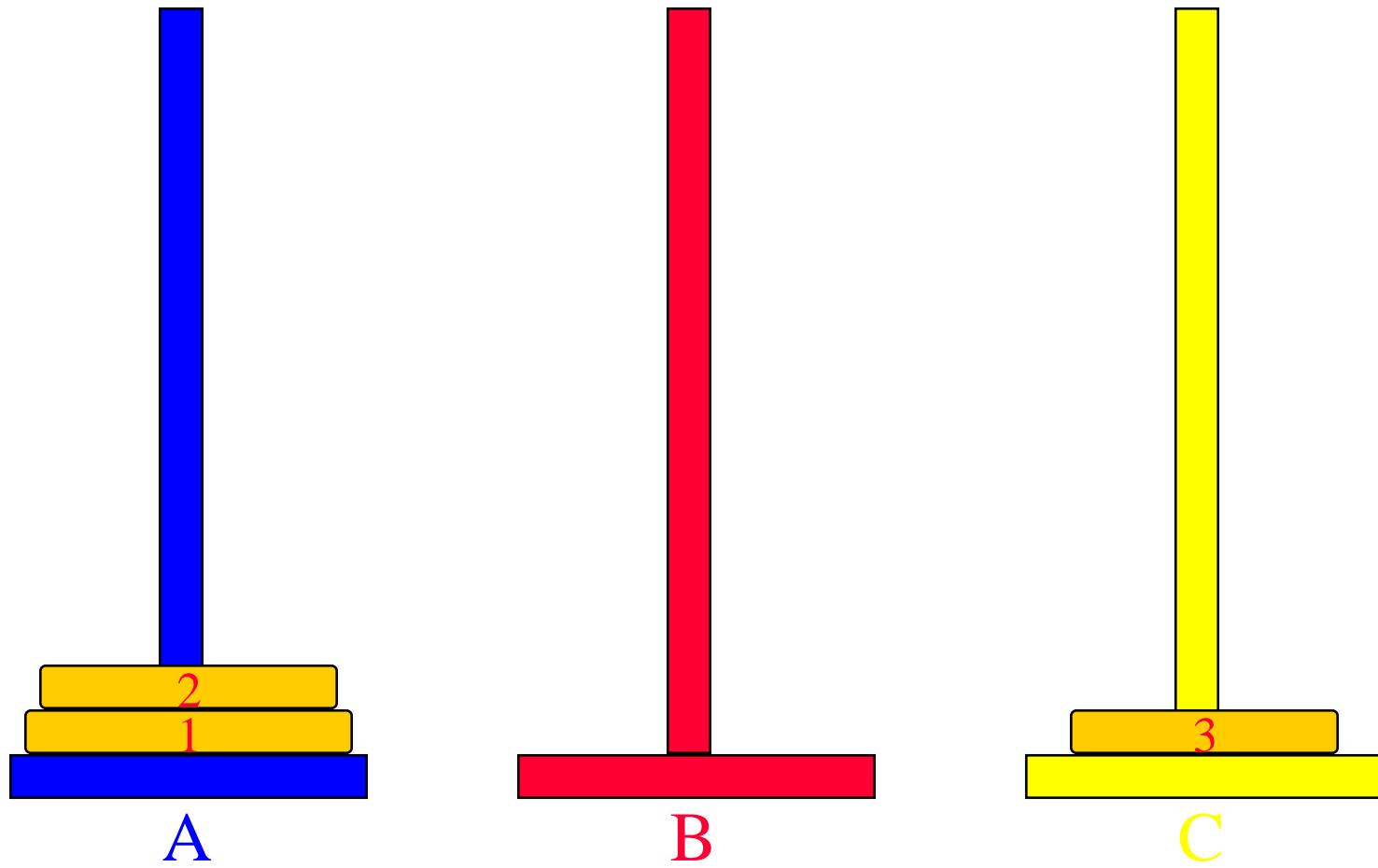
- 64 алтан дискийг А цамхгаас С цамхагт шилжүүлнэ
- Цамхаг бүр стек шиг ажиллана
- Том дискийг жижиг диск дээр тавьж болохгүй

# Ханойн цамхаг



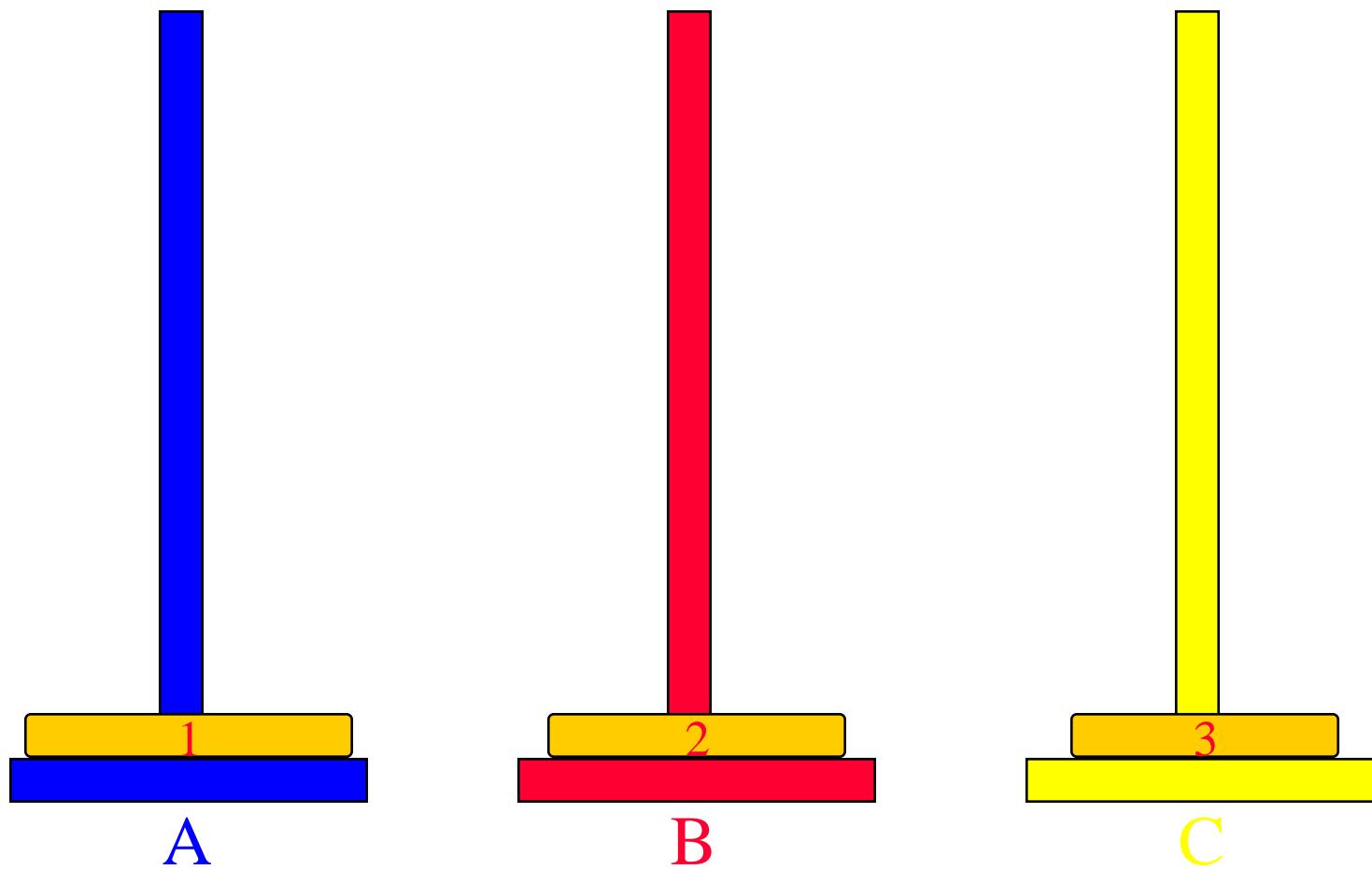
- 3 дисктэй Ханойн цамхаг

# Ханойн цахаг



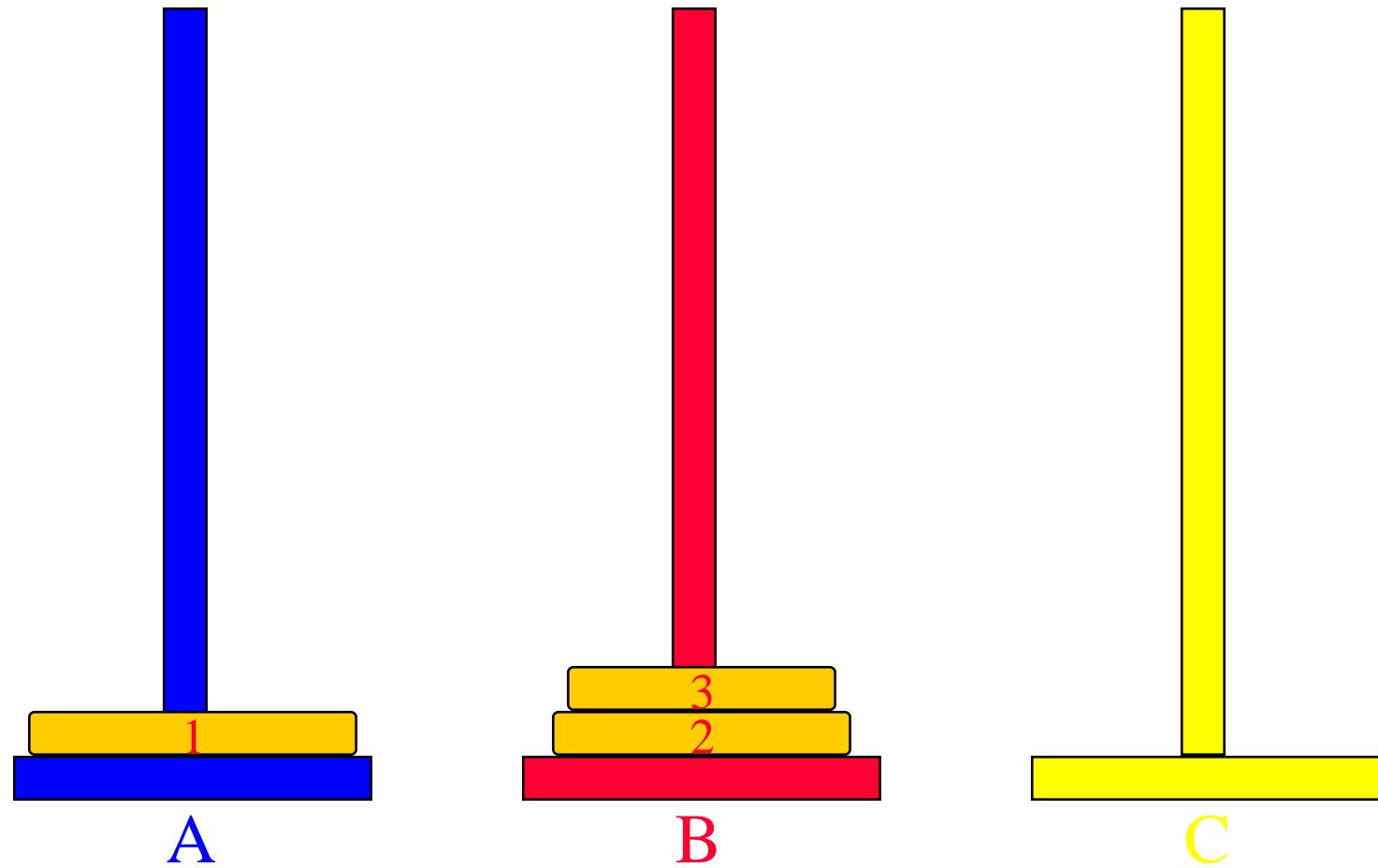
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



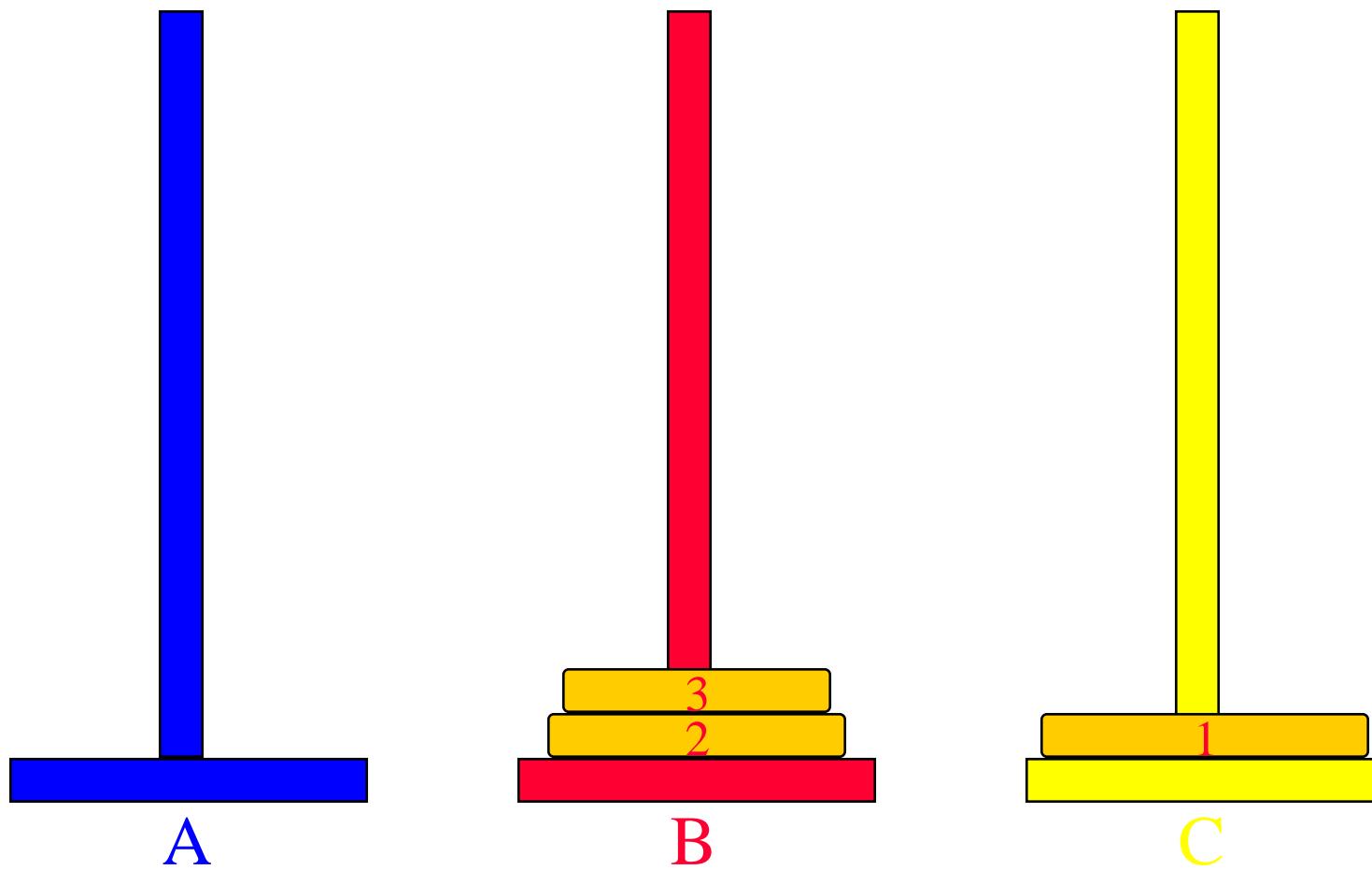
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



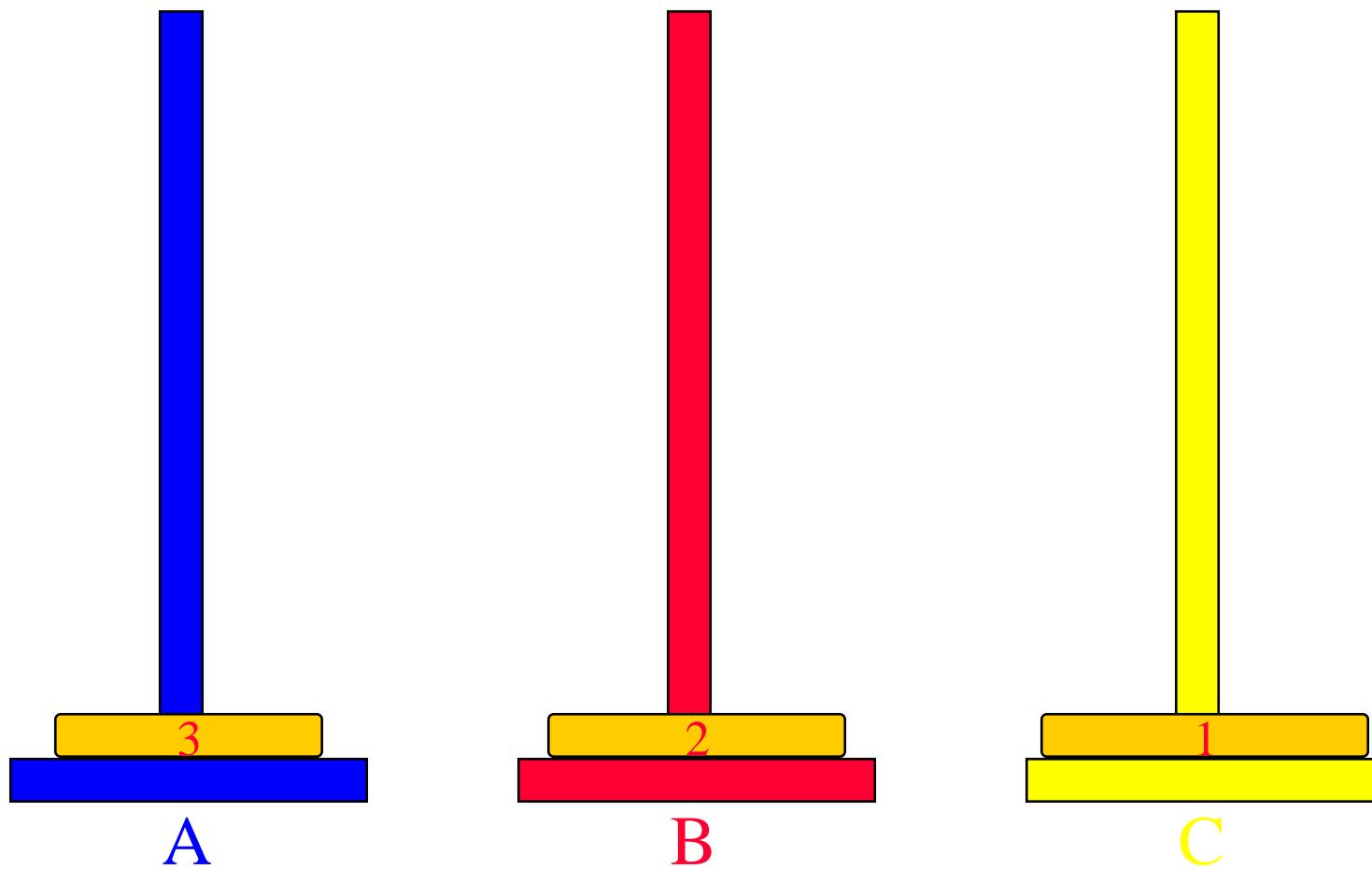
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



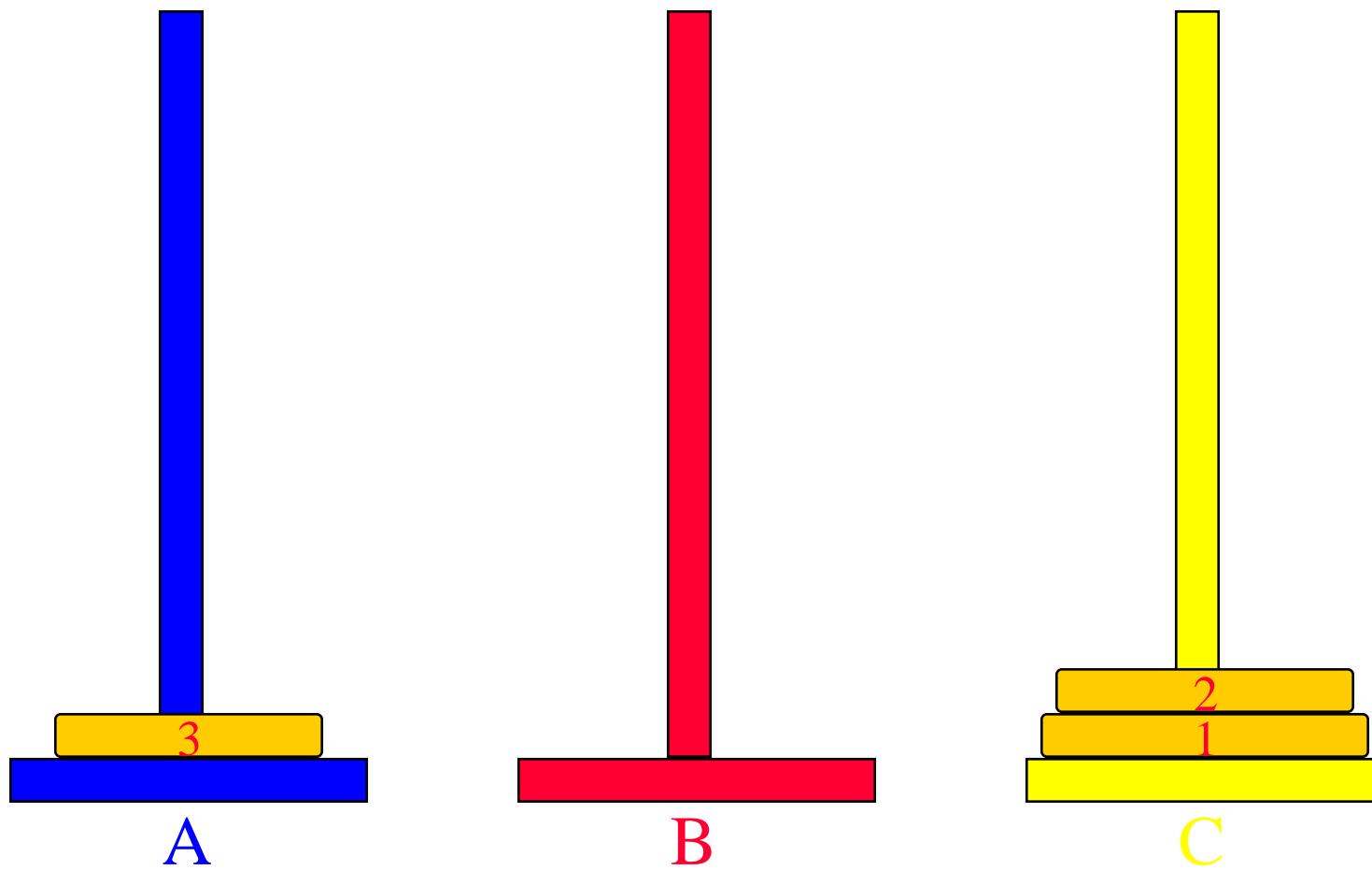
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



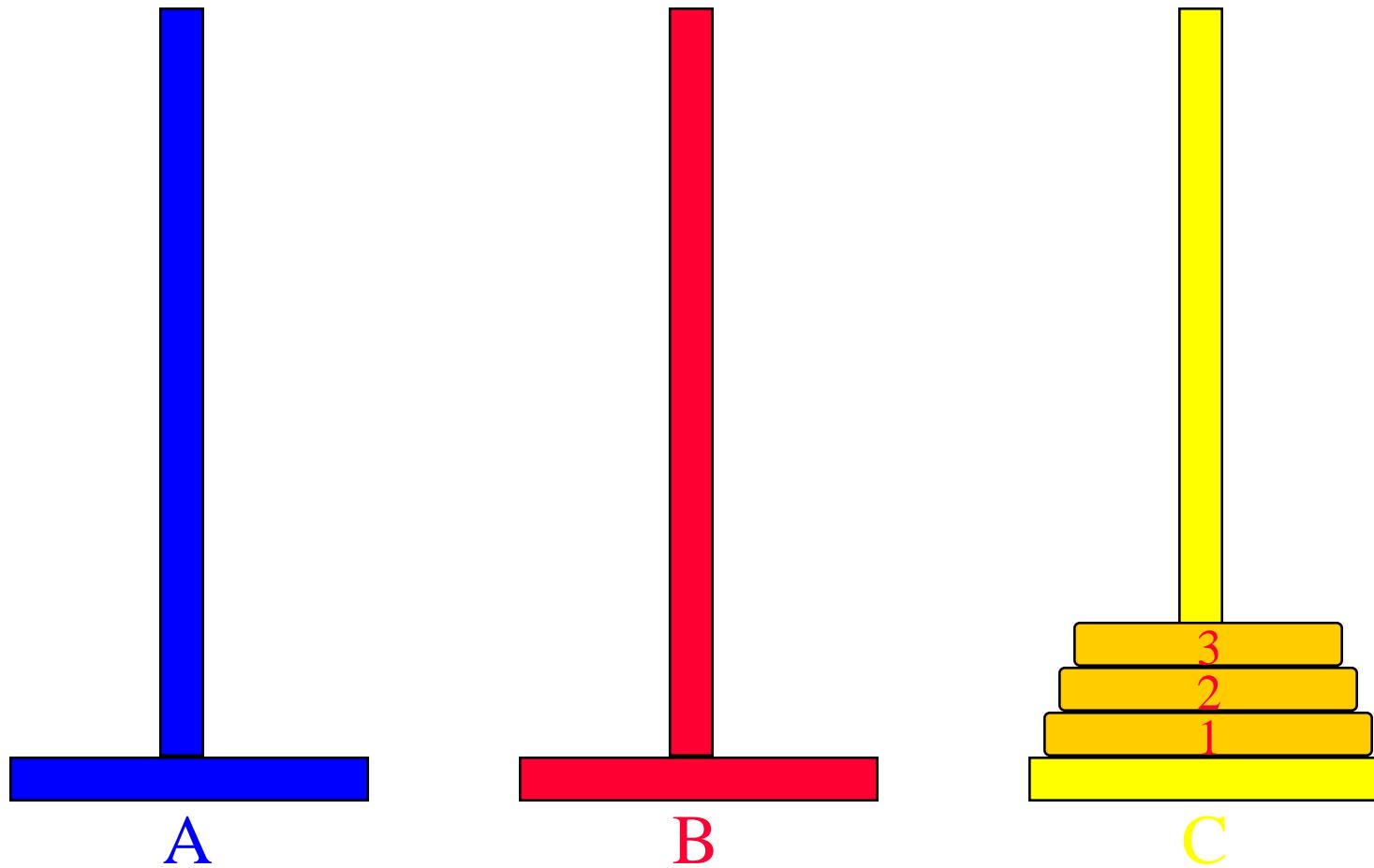
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



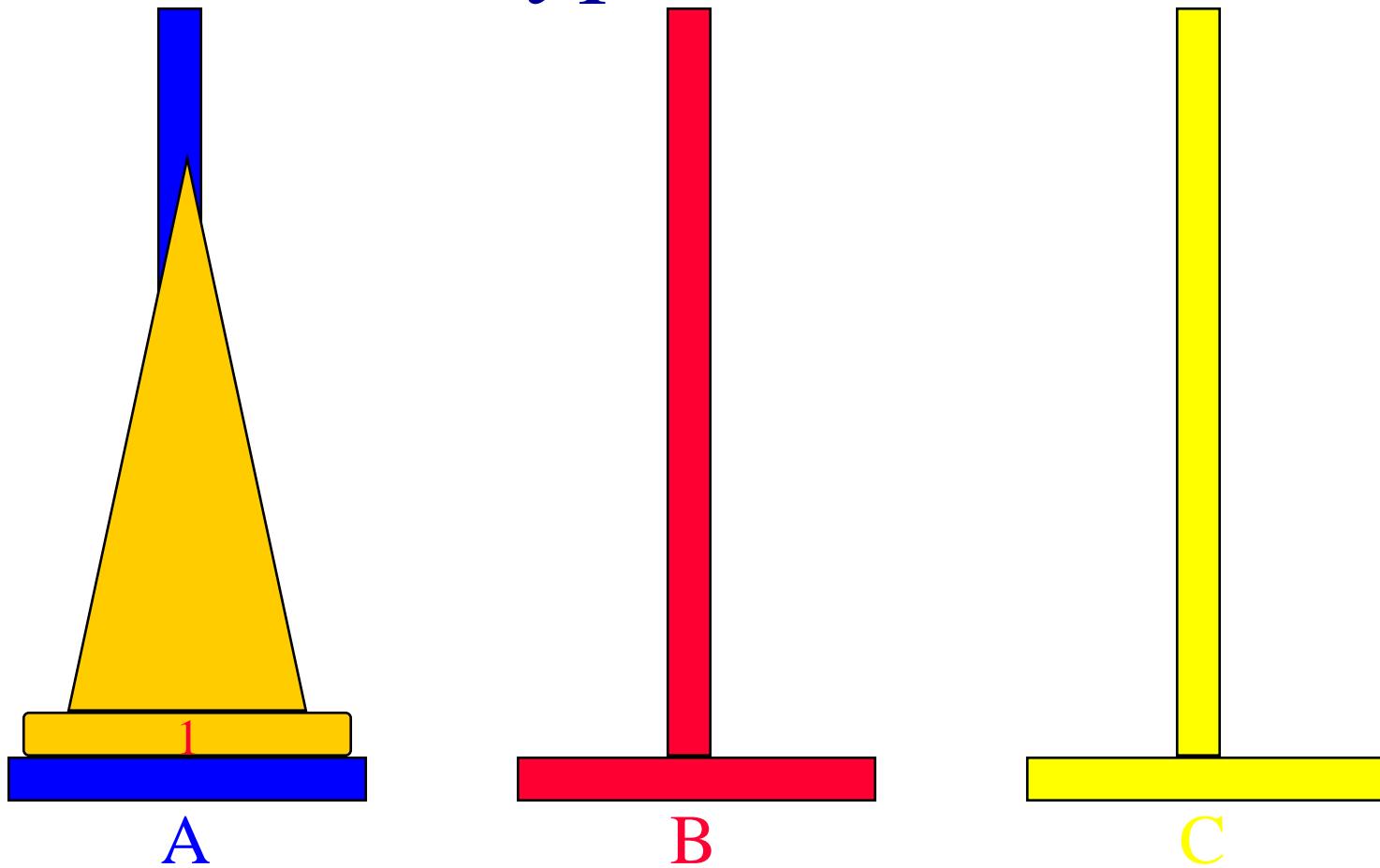
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



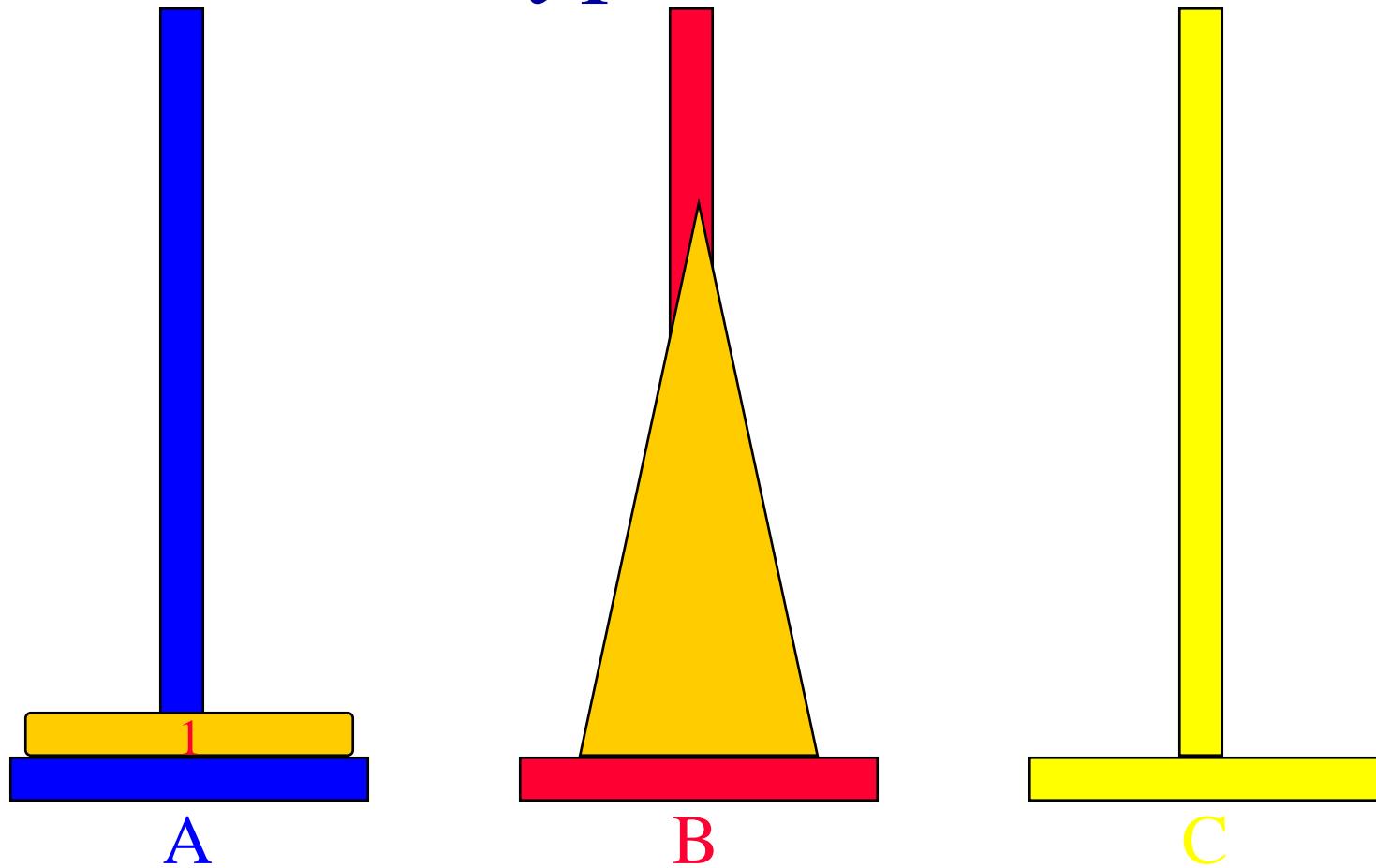
- 3 дисктэй Ханойн цамхаг
- 7 дискийг хөдөлгөлөө

# Рекурсив шийдэл



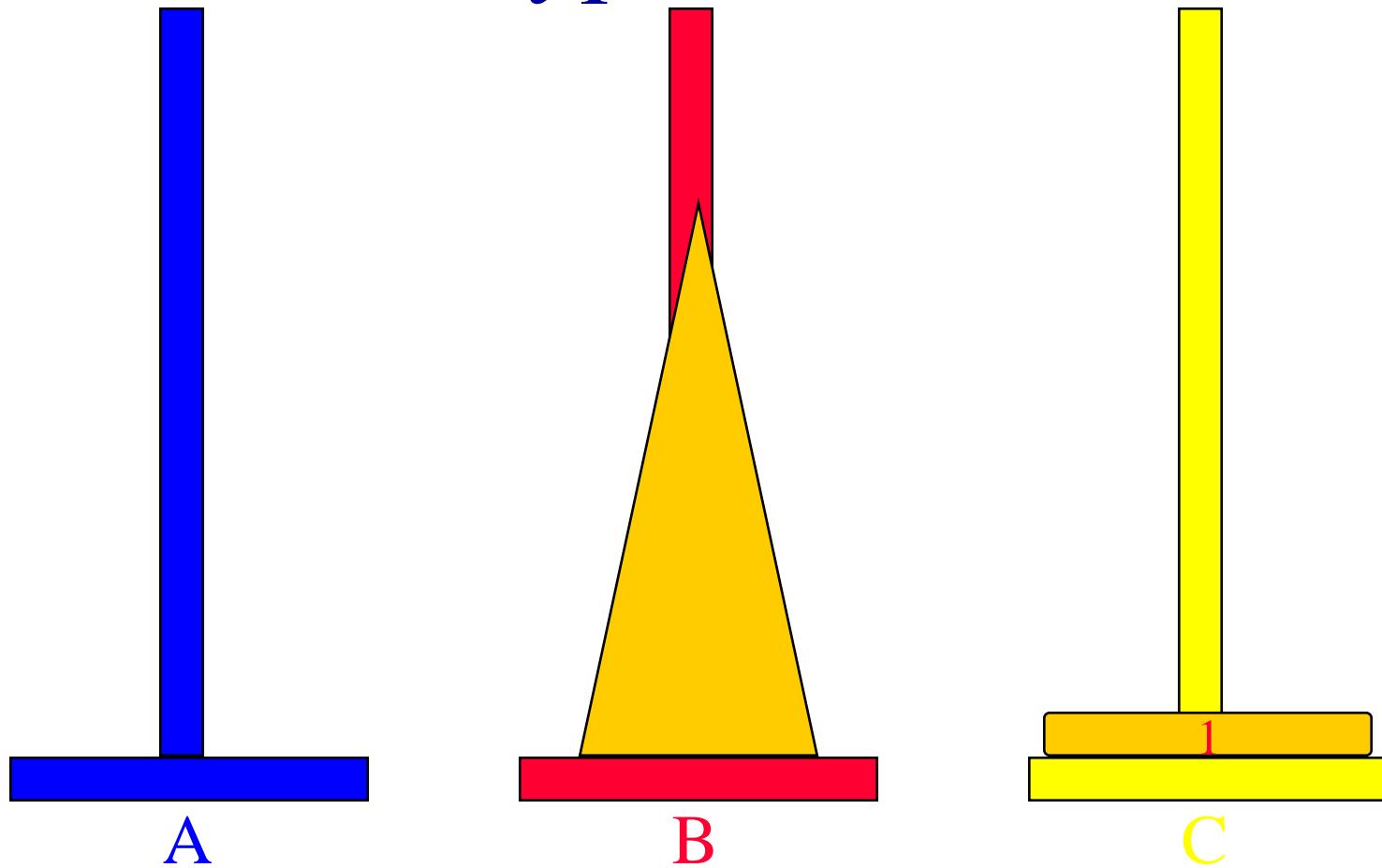
- $n > 0$  алтан дискийг **A** -аас **C** рүү **B** –г ашиглан шилжүүлнэ
- оройн  $n-1$  дискийг **A** -аас **B** рүү **C** –г ашиглан шилжүүлнэ

# Рекурсив шийдэл



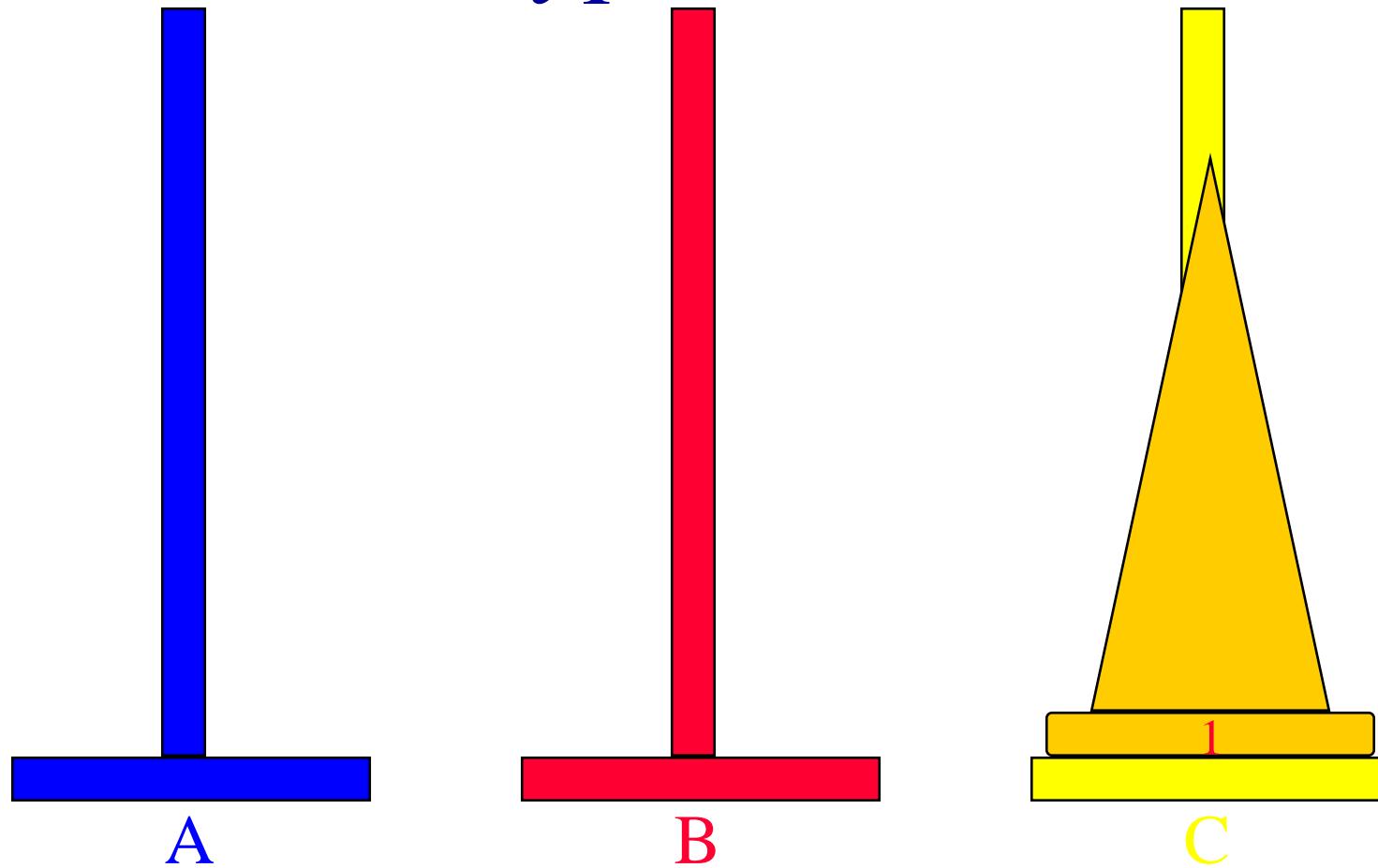
- оройн дискийг **A** -аас **C** рүү шилжүүлнэ

# Рекурсив шийдэл



- оройн  $n-1$  дискийг B -ээс C рүү A –г ашиглан шилжүүлнэ

# Рекурсив шийдэл

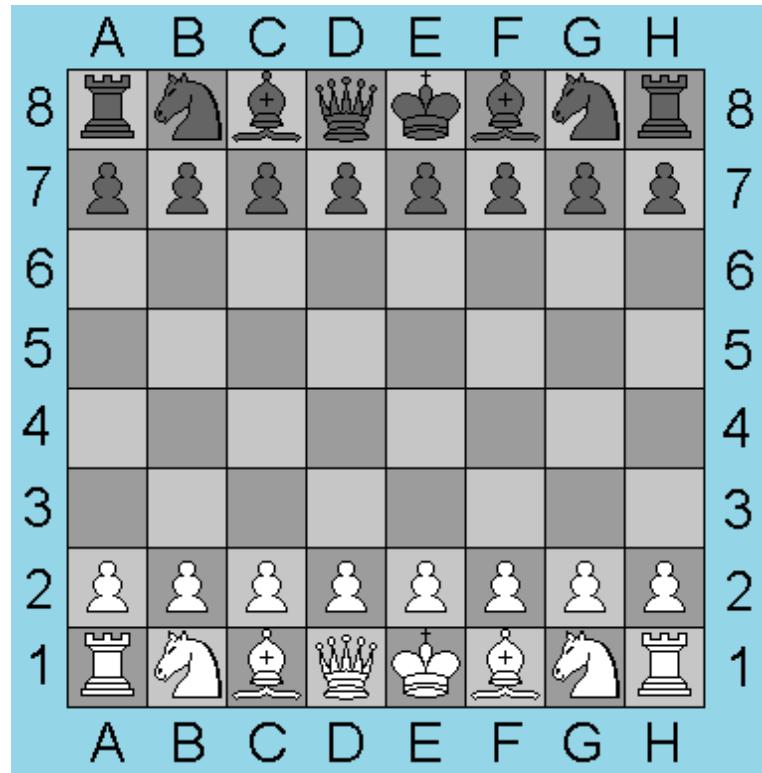


- $\text{moves}(n) = 0$ ,  $n = 0$  бол
- $\text{moves}(n) = 2 * \text{moves}(n-1) + 1 = 2^{n-1}$ ,  $n > 0$  бол<sup>3</sup>

# Ханойн цамхаг

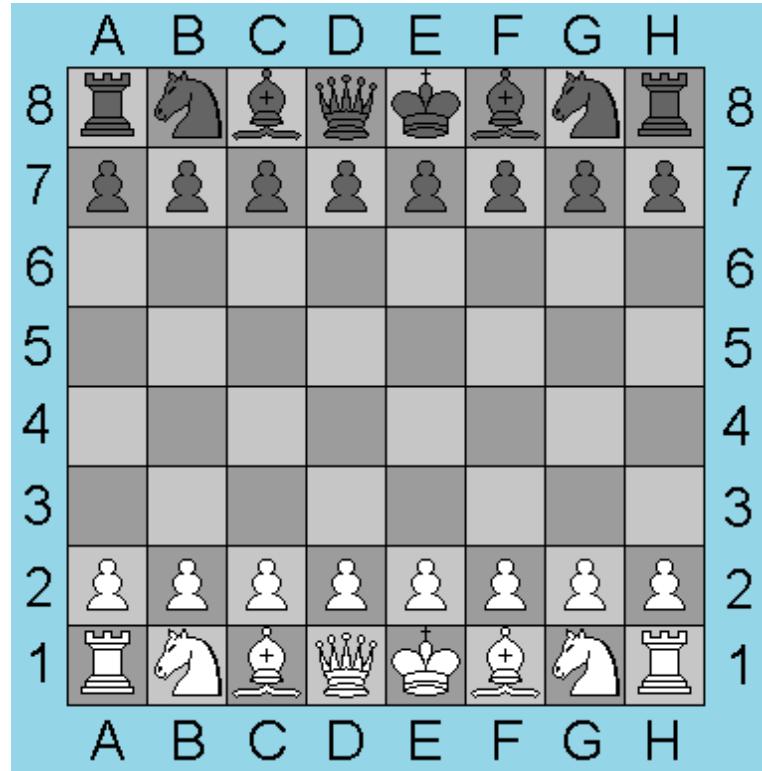
- $\text{moves}(64) = 1.8 * 10^{19}$  (оиролцоогоор)
- $10^9$  шилжүүлэлт/сек хурдтай компьютер **570** орчим жил зарцуулна.
- 1 диск шилжүүлэлт/мин хийдэг лам  $3.4 * 10^{13}$  жил зарцуулна.

# Шатрын хөлөг



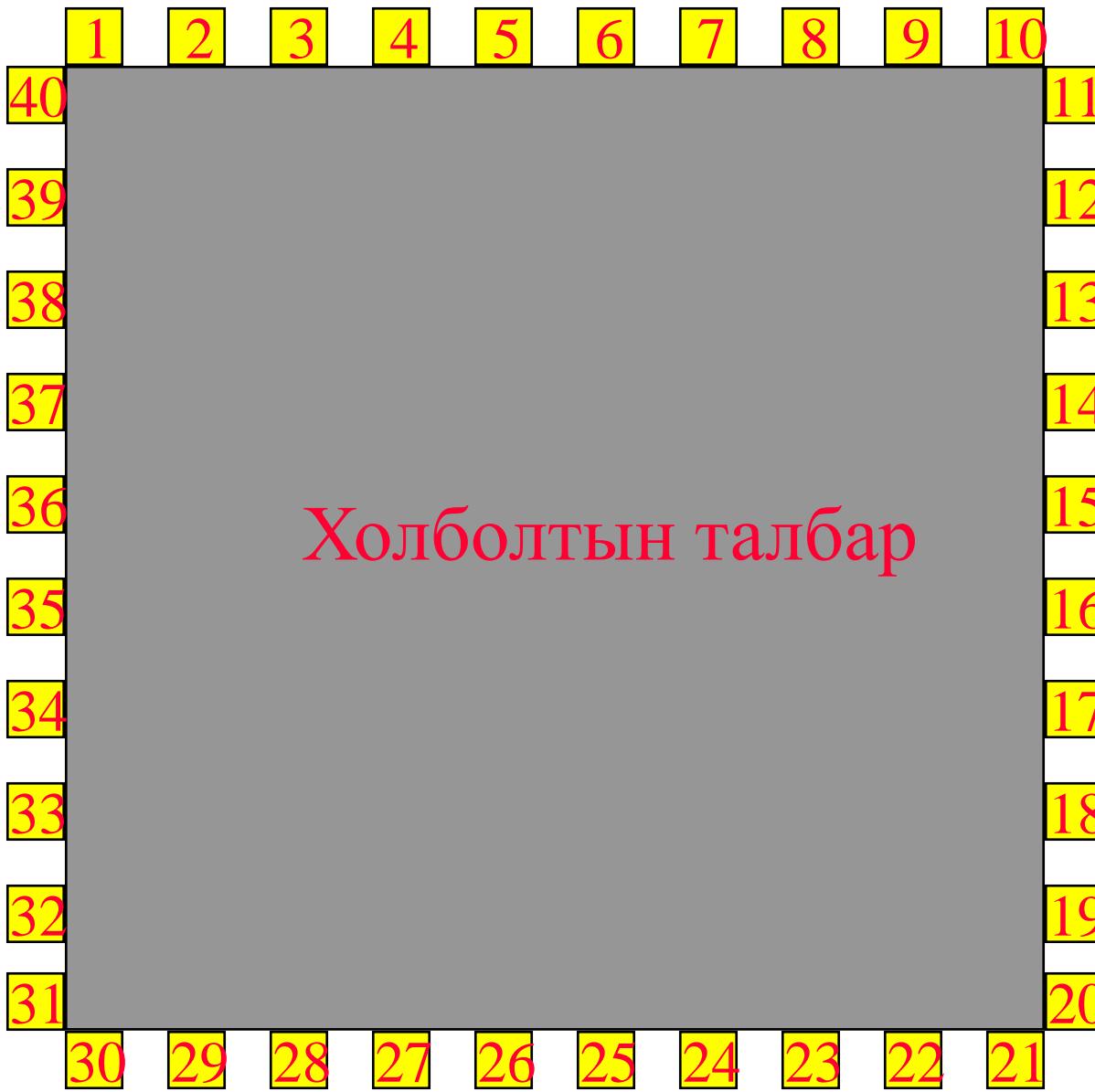
- Эхний нүдэнд 1 будаа, дараагийнхад 2, дараагийнхад 4, гэх мэт.
- Шаардлагатай талбай тэлхийг бүрхэнэ.

# Шатрын хөлөг

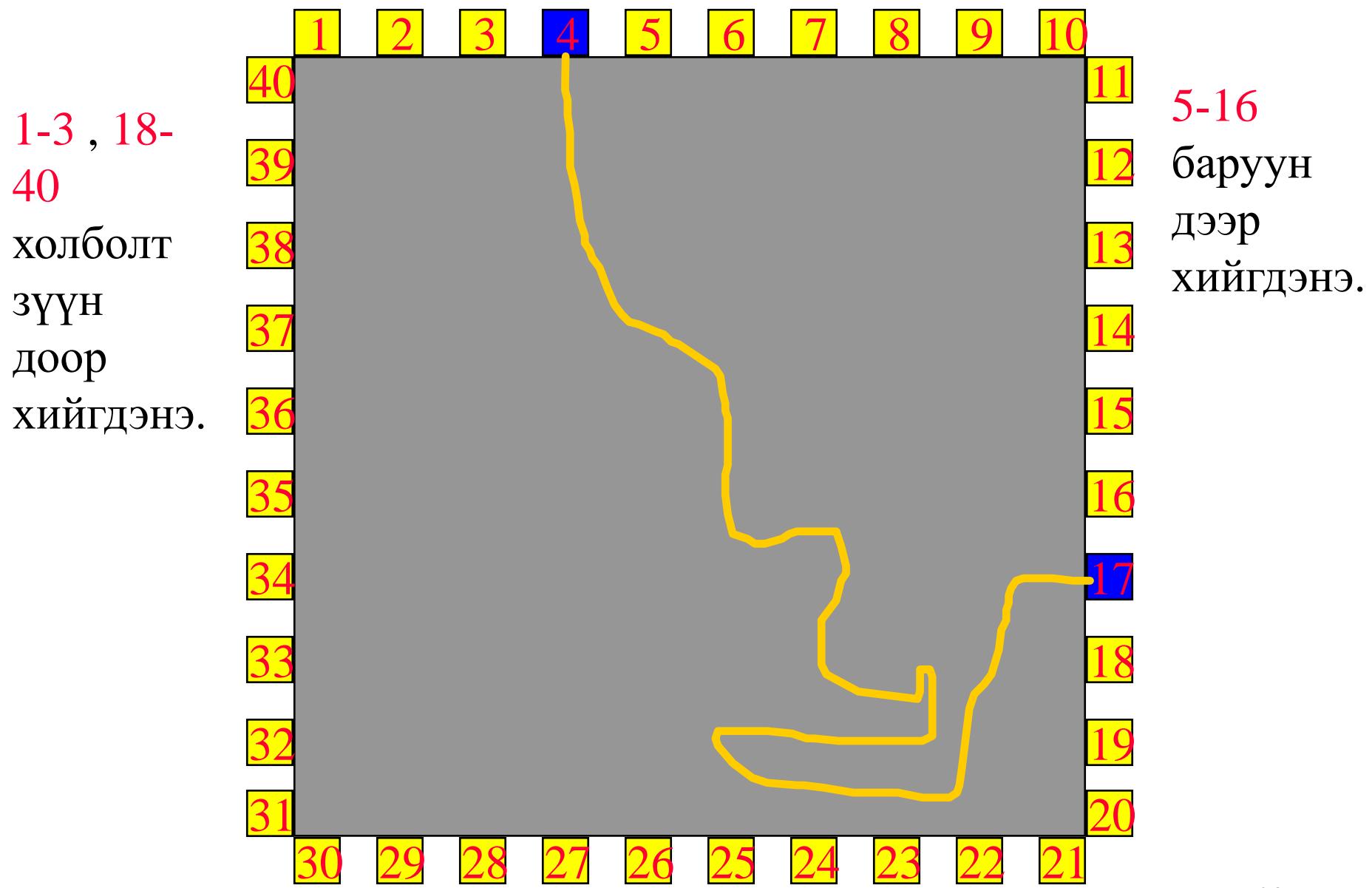


- Эхний нүдэнд 1 цент, дараагийнхад 2, дараагийнхад 4, гэх мэт.
- $\$3.6 * 10^{17}$  (холбооны төсөв  $\sim \$2 * 10^{12}$ ) .

# Холболтын хайрцаг



## 2 цэгийн холболт

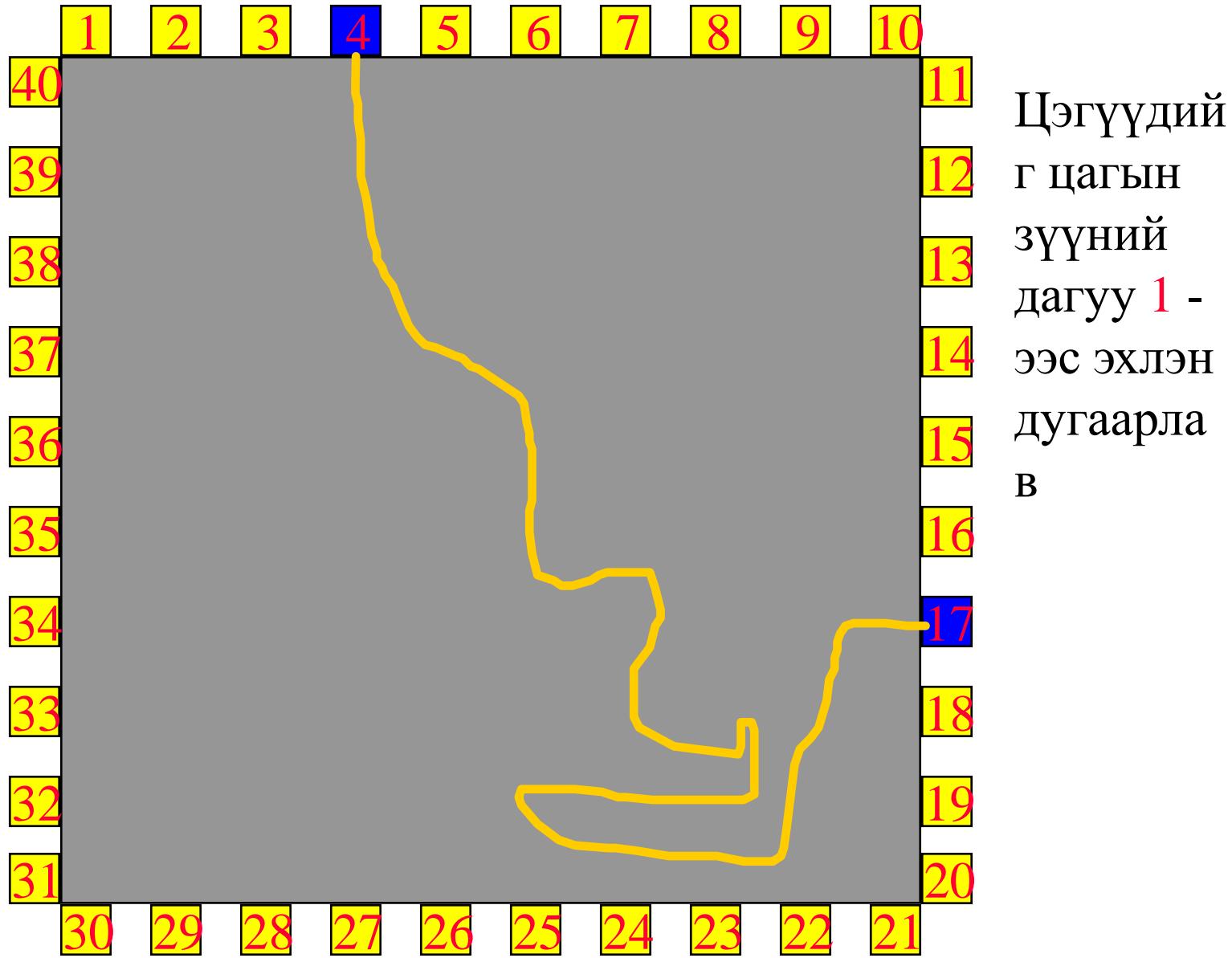


# 2 цэгийн холболт

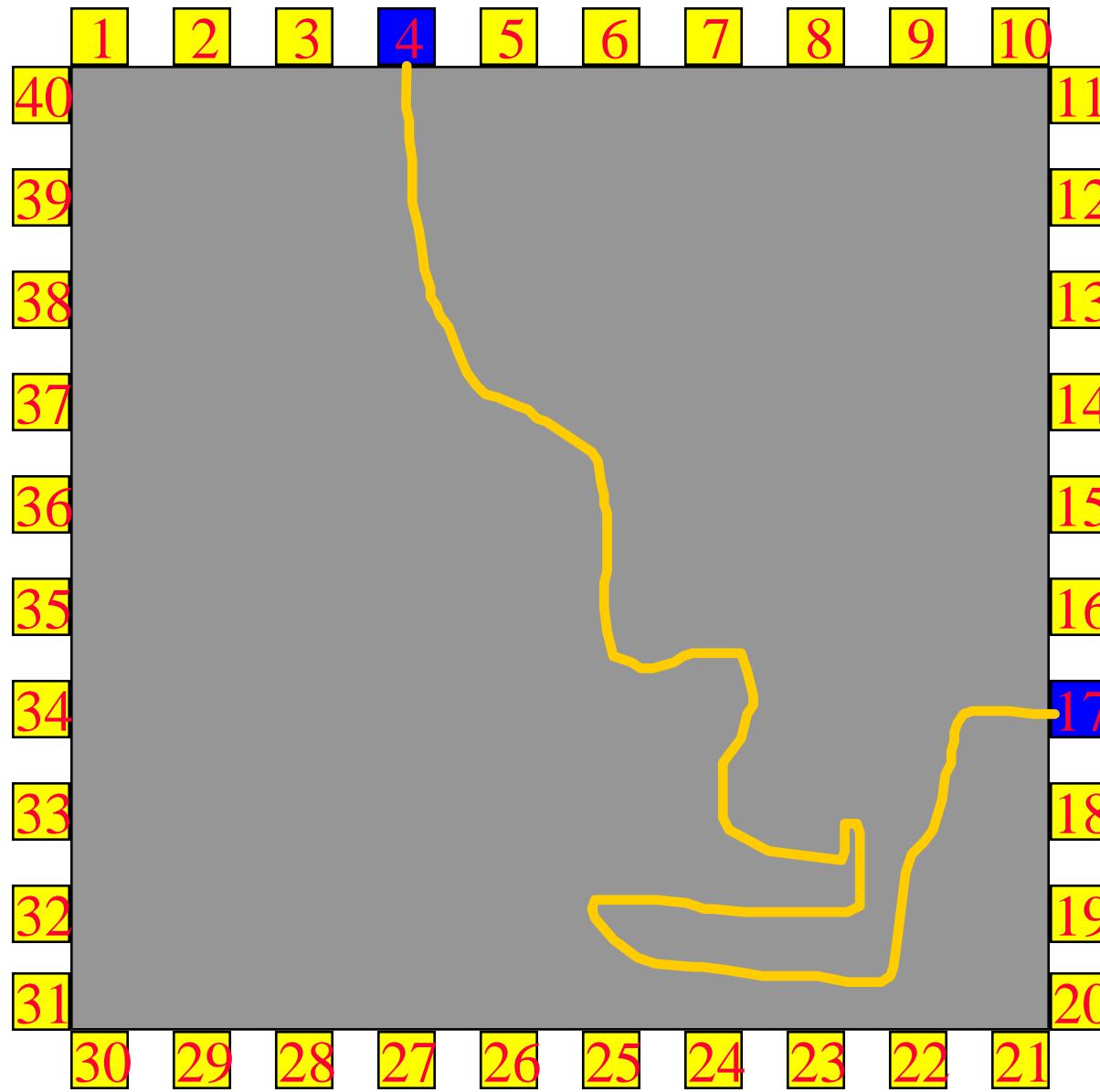
( $u, v$ ),  $u < v$   
болов 2-  
цэгийн  
холболт.

$u$  эхлэх  
цэг.

$v$  төгсөх  
цэг.



## 2 ЦЭГИЙН ХОЛБОЛТ



# Аргыг Дуудах ба Буцах

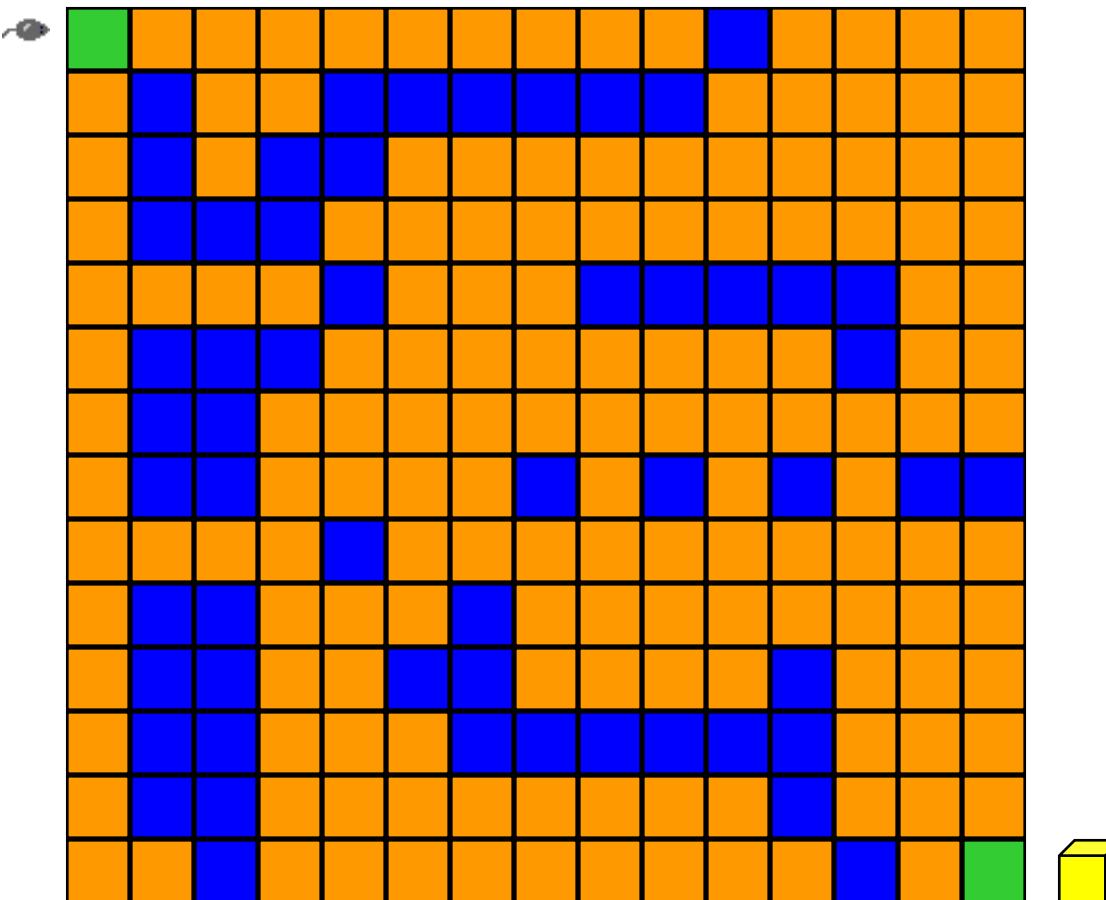
```
public void a()  
{ ...; b(); ...}  
  
public void b()  
{ ...; c(); ...}  
  
public void c()  
{ ...; d(); ...}  
  
public void d()  
{ ...; e(); ...}  
  
public void e()  
{ ...; c(); ...}
```

буцах хаяг d()  
буцах хаяг c()  
буцах хаяг e()  
буцах хаяг d()  
буцах хаяг c()  
буцах хаяг b()  
буцах хаяг a()

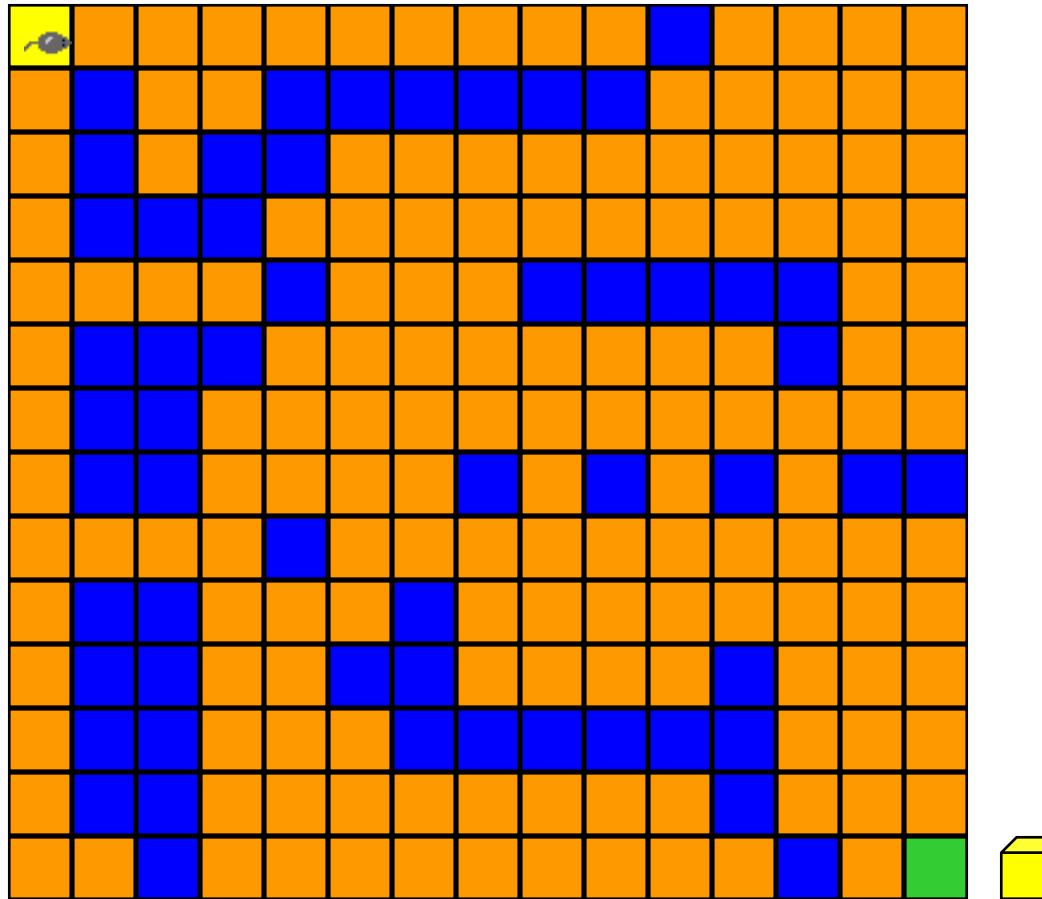
# Try-Throw-Catch

- try блокт ороход, энэ блокын хаягийг стект хийнэ.
- Онцгой тохиолдол унахад, стекын оройд байгаа try блокын хаягийг авна(стек хоосон бол зогсоно).
- Гаргаж авсан try блокт харгалзах catch блок байхгүй бол, өмнөх алхам руу буцна.
- Гаргаж авсан try блокт харгалзах catch блок байгаа бол, тэр catch блок хэрэгжинэ.

# Төөрөлдсөн оготно

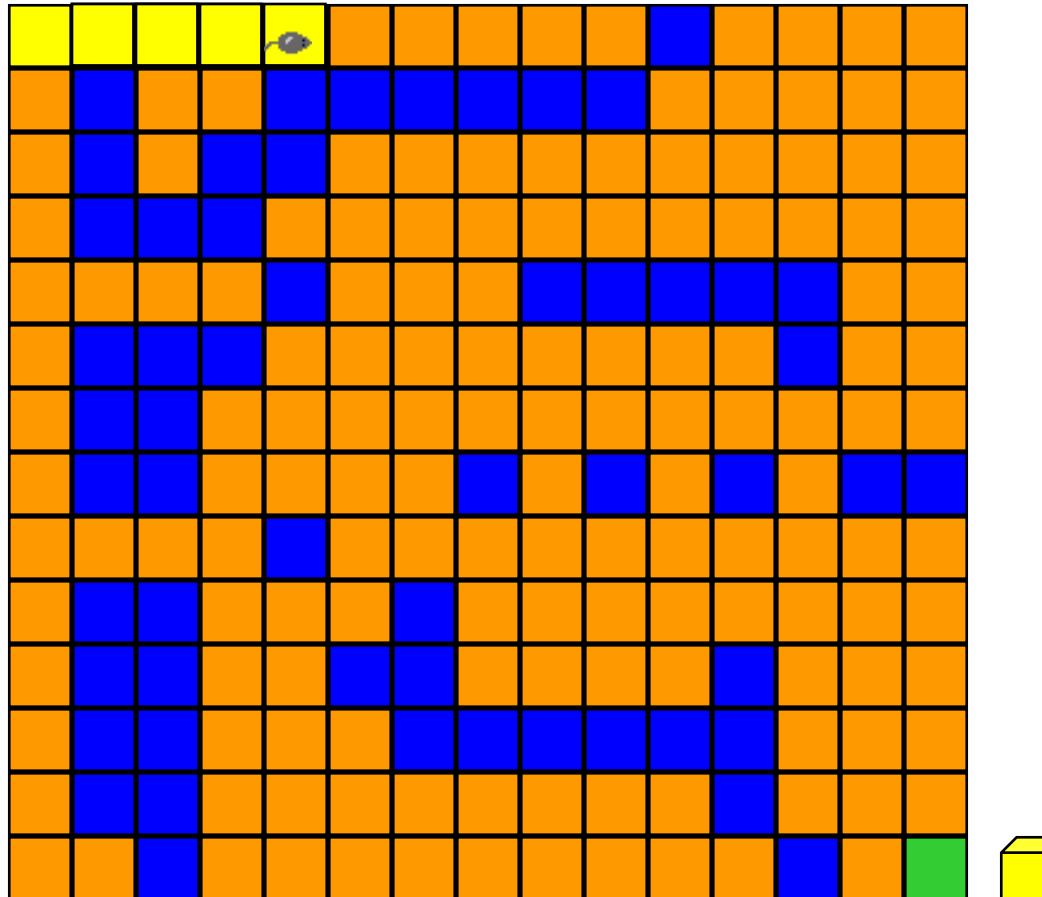


# Төөрөлдсөн оготно



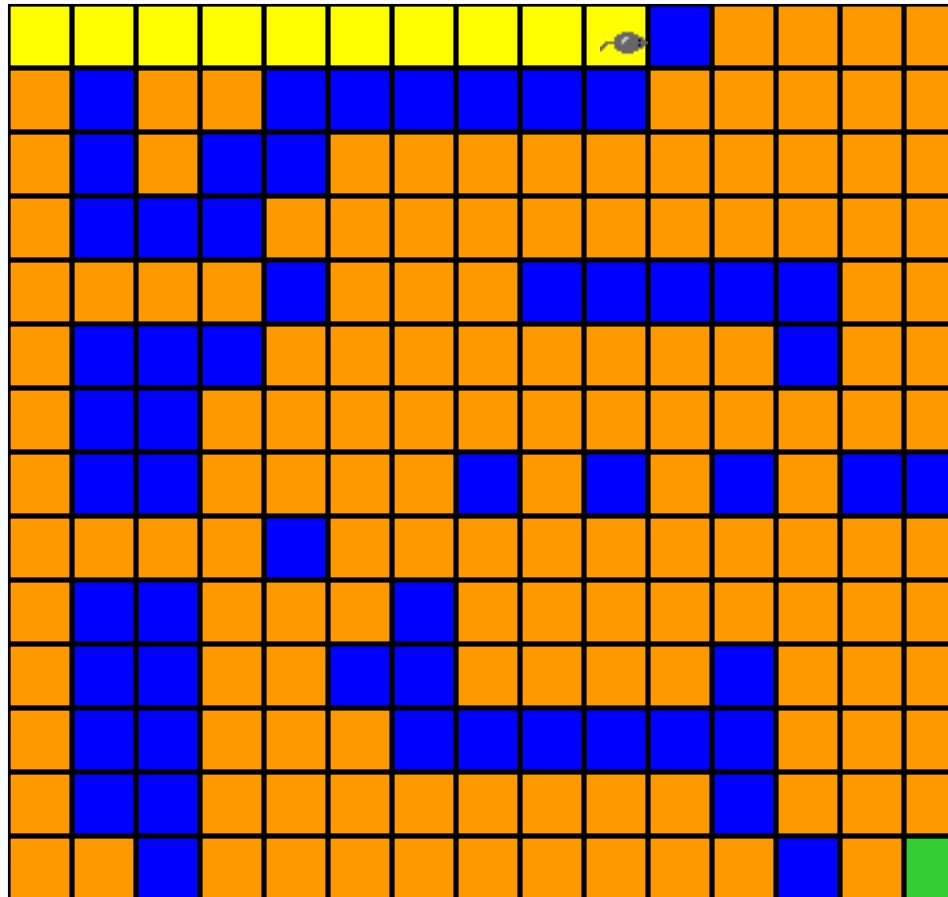
- Явах чиглэл: **right, down, left, up**
- Дахин орохгүйн тулд нүдийг хаана.

# Төөрөлдсөн оготно



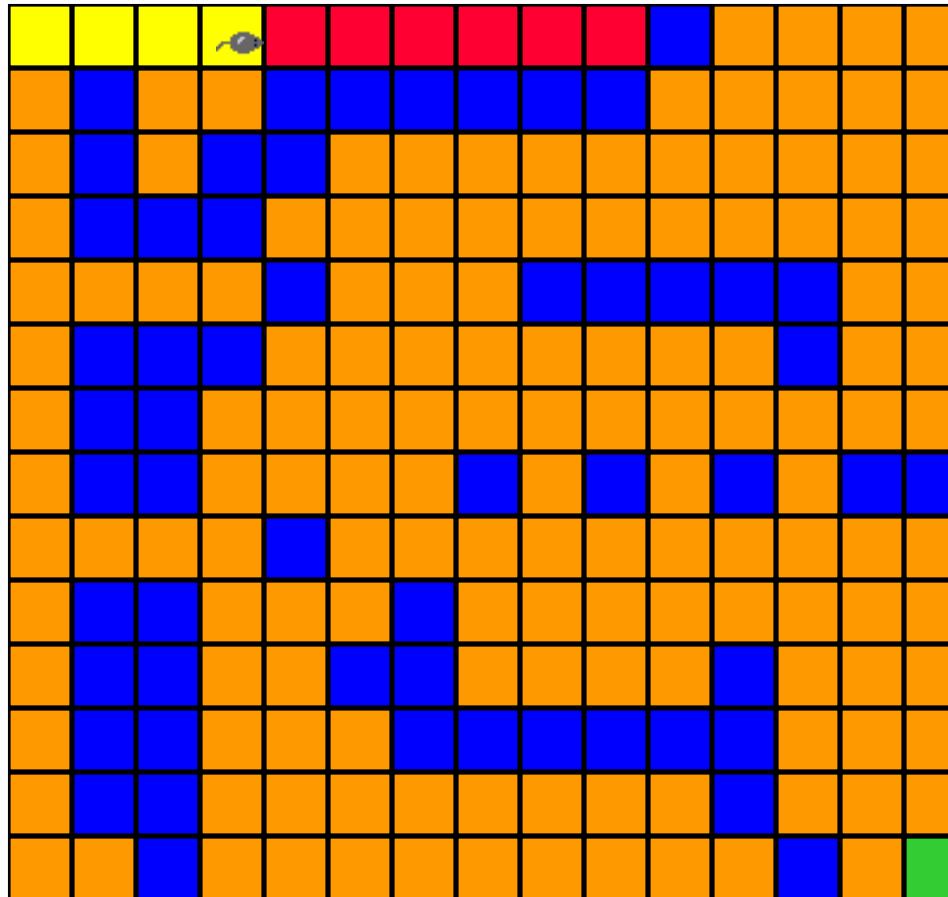
- Явах чиглэл: **right, down, left, up**
- Дахин орохгүйн тулд нүдийг хаана.

# Төөрөлдсөн оготно



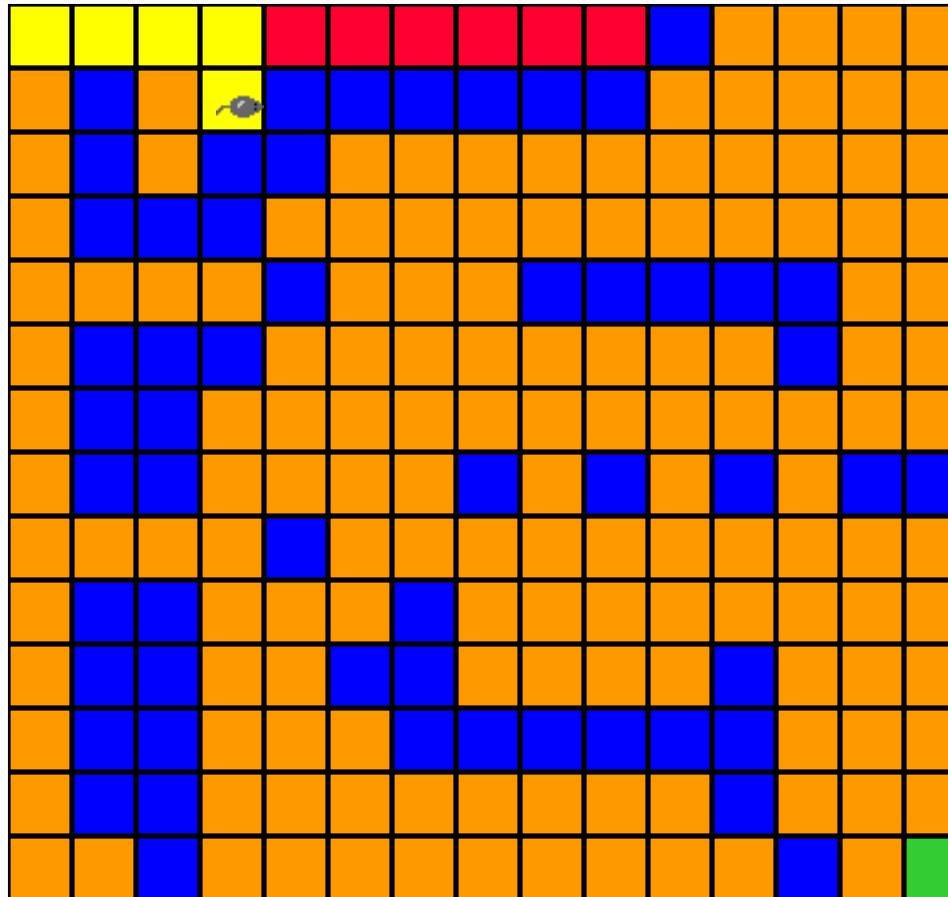
- Урагш явах боломжтой нүд хүртэл ухрах.

# Төөрөлдсөн оготно



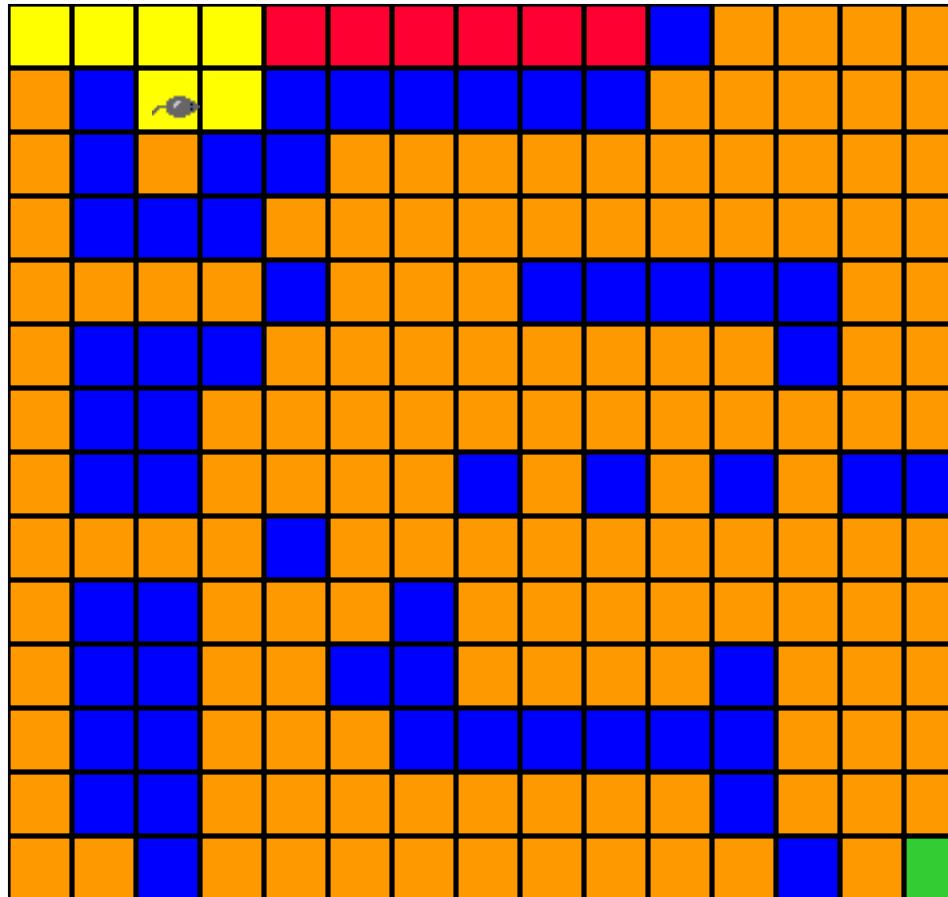
- Доош явах.

# Төөрөлдсөн оготно



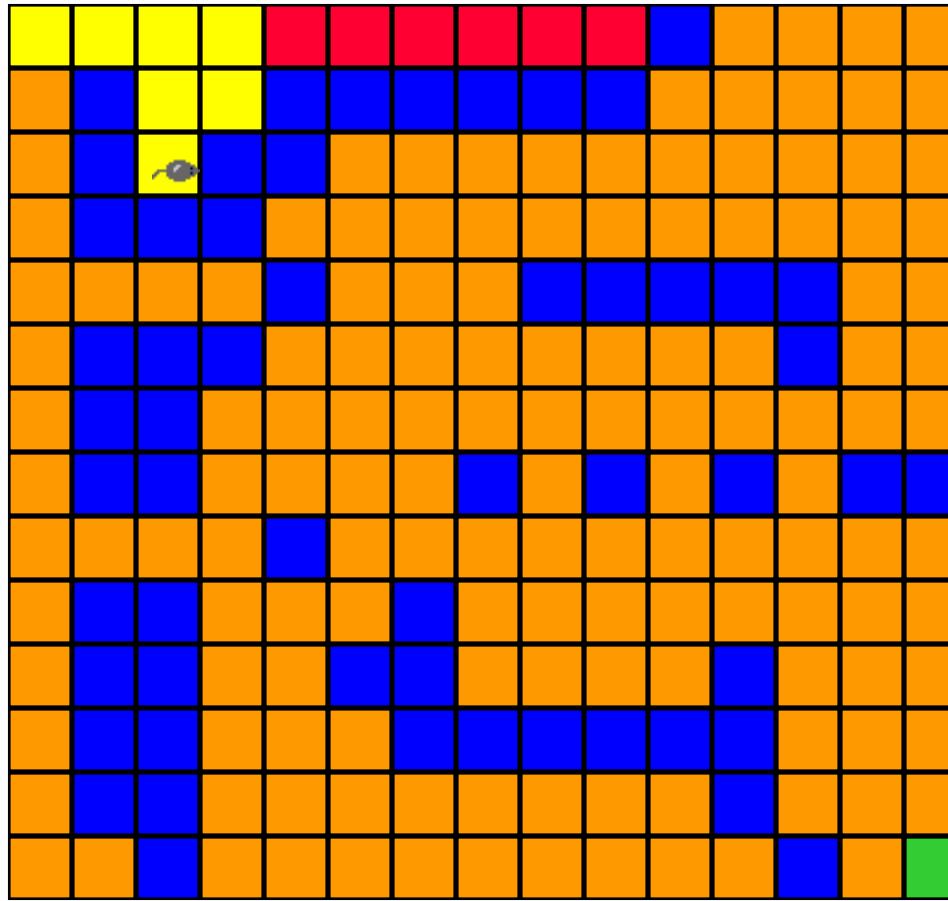
- Зүүн тийш явах.

# Төөрөлдсөн оготно



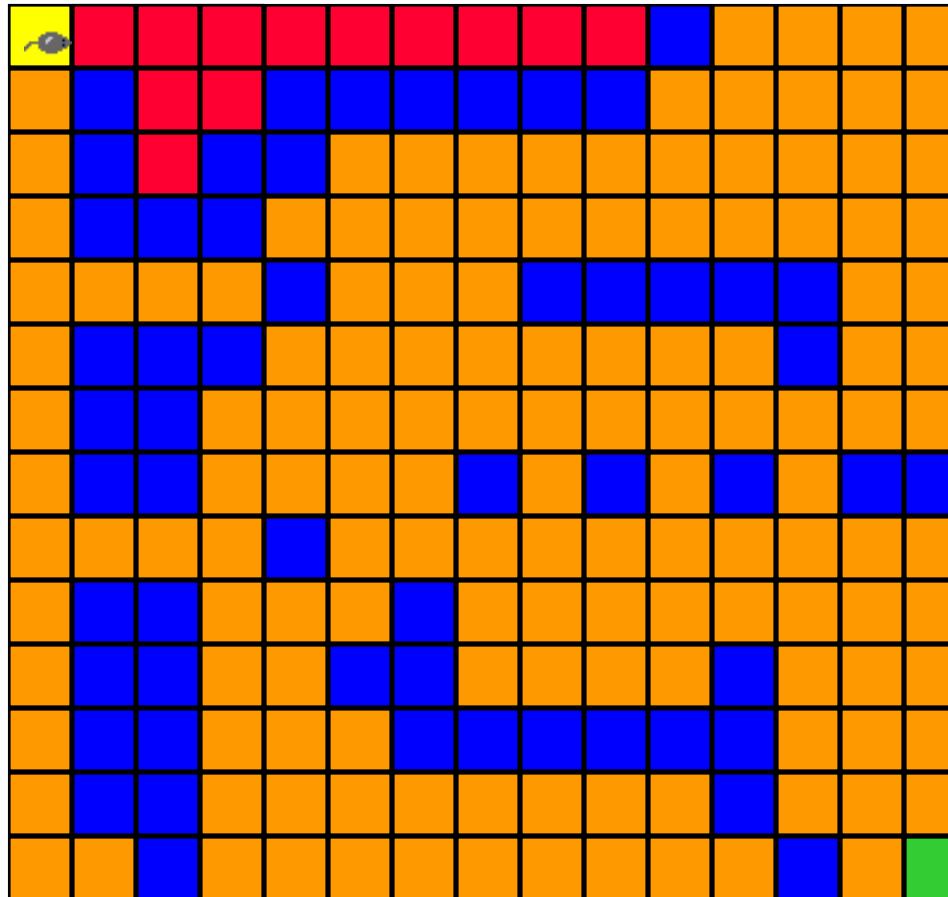
- Доош явах.

# Төөрөлдсөн оготно



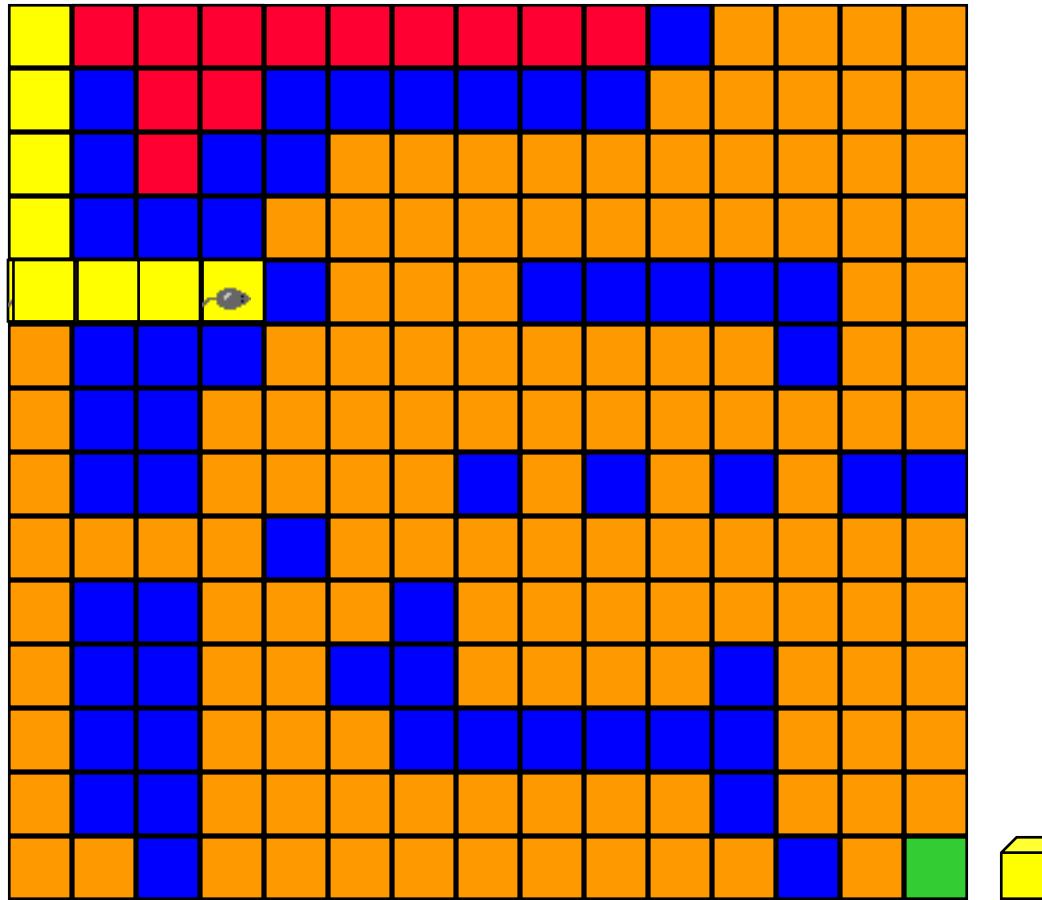
- Урагш явах боломжтой нүд хүртэл ухрах.

# Төөрөлдсөн оготно



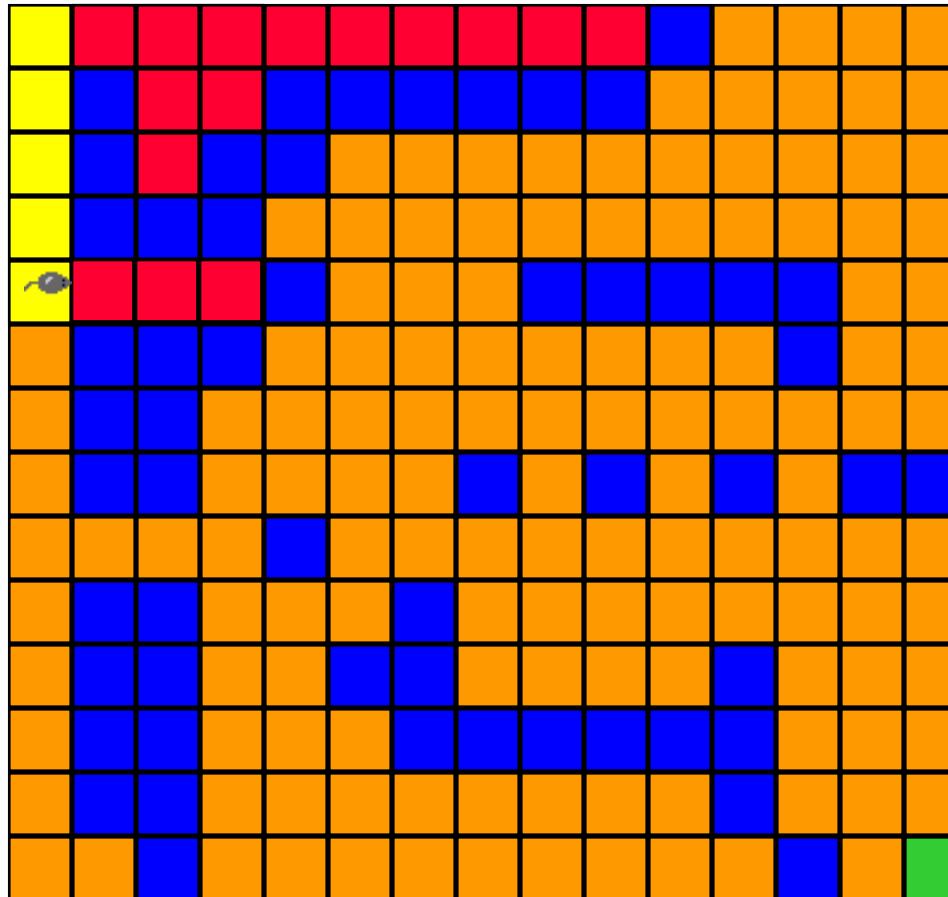
- Урагш явах боломжтой нүд хүртэл ухрах.
- Доош явах.

# Төөрөлдсөн оготно



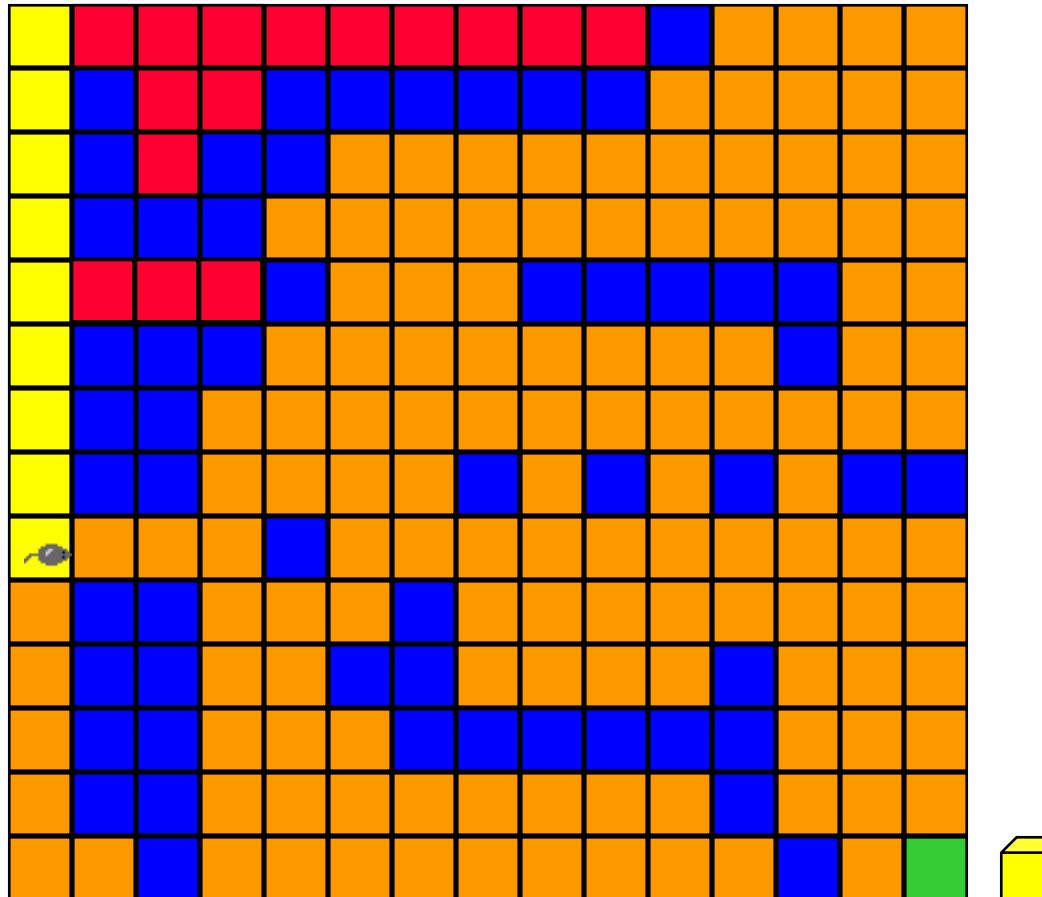
- Баруун тийш явах.
- Буцах.

# Төөрөлдсөн оготно



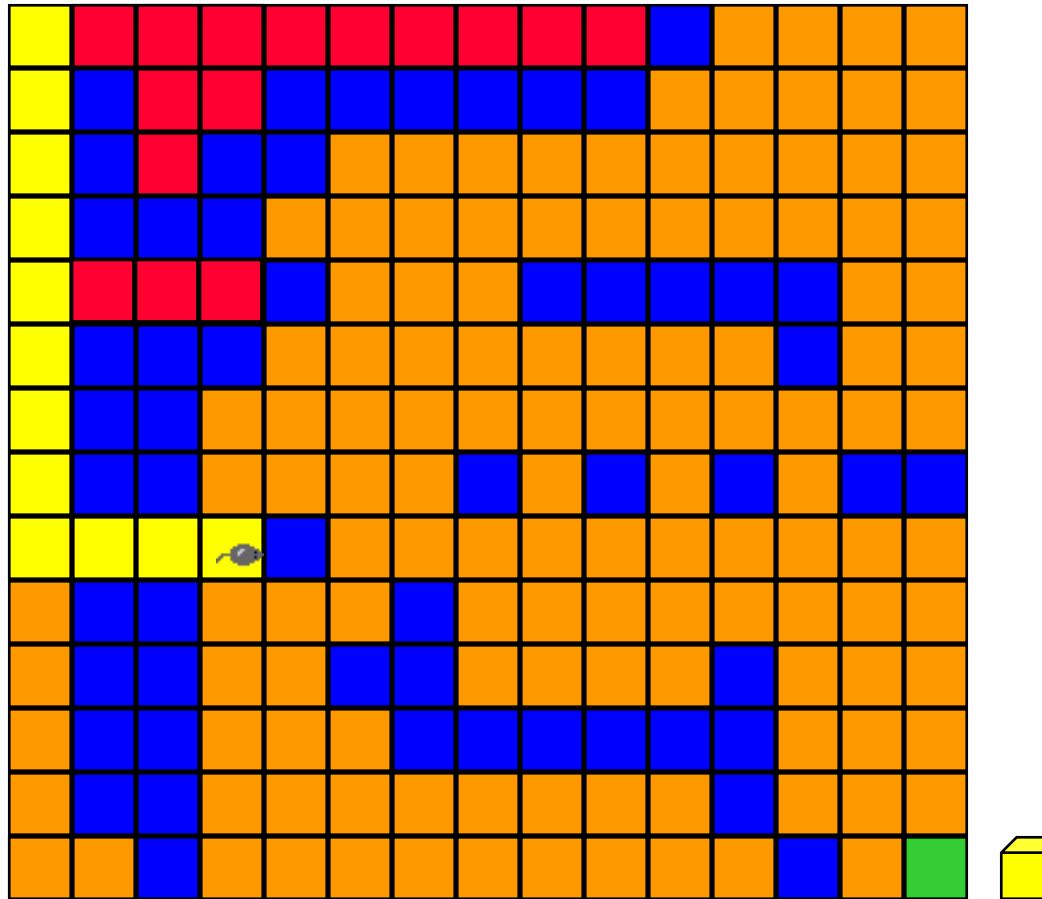
- Доош явах.

# Төөрөлдсөн оготно



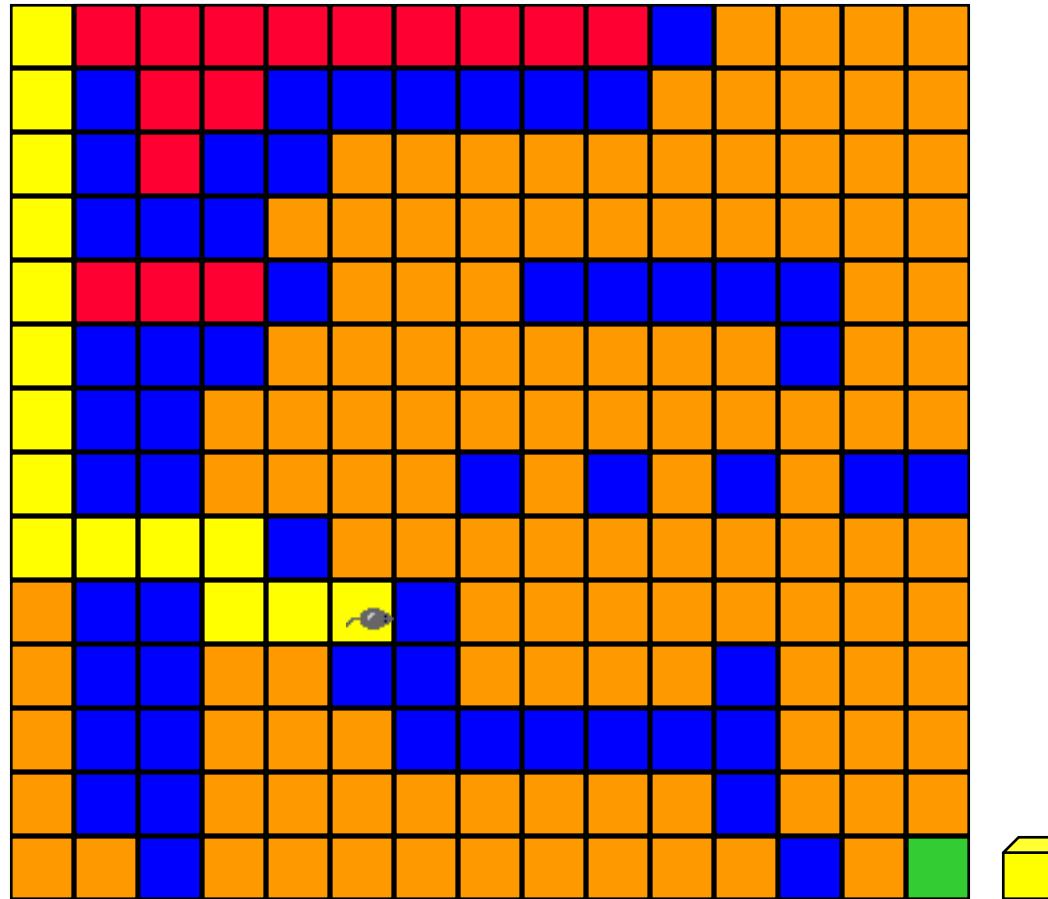
- Баруун тийш явах.

# Төөрөлдсөн оготно



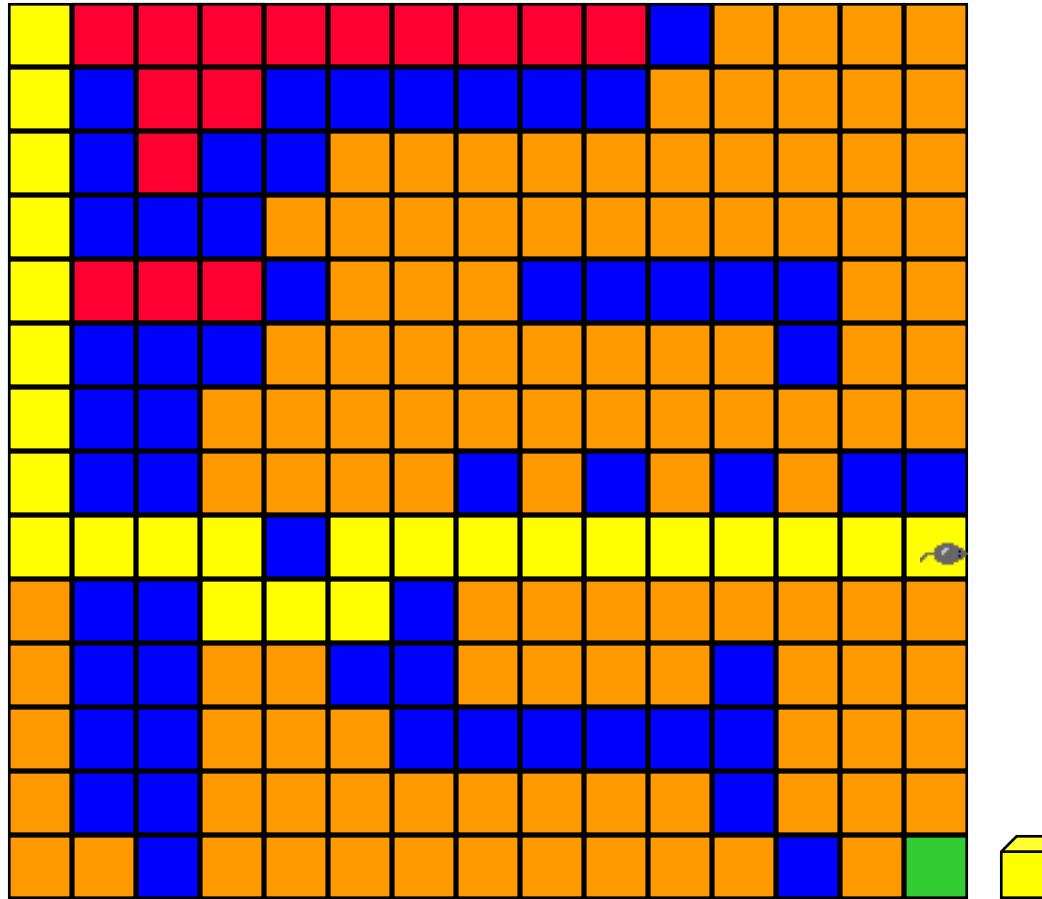
- Нэг доошлоод баруун тийш явах.

# Төөрөлдсөн хулгана



- Нэг дээшлээд баруун тийш явах.

# Төөрөлдсөн оготно



- Доош явж гарангаа бяслагийг идэх.
- Оготны орсон нүднээс тухайн байршил хүртэлх зам стектэй адилхан ажиллана.

# Стек

```
public interface Stack
```

```
{
```

```
    public boolean empty();
```

```
    public Object peek();
```

```
    public void push(Object theObject);
```

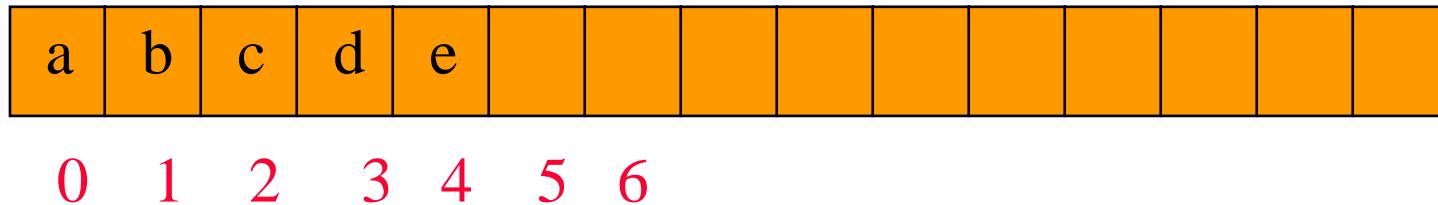
```
    public Object pop();
```

```
}
```

# Linear List классаас уламжлах

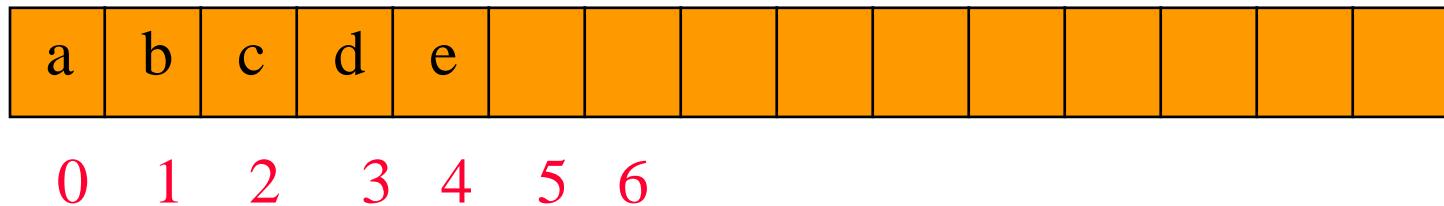
- `ArrayList`
- `Chain`

# ArrayList –ээс уламжлах



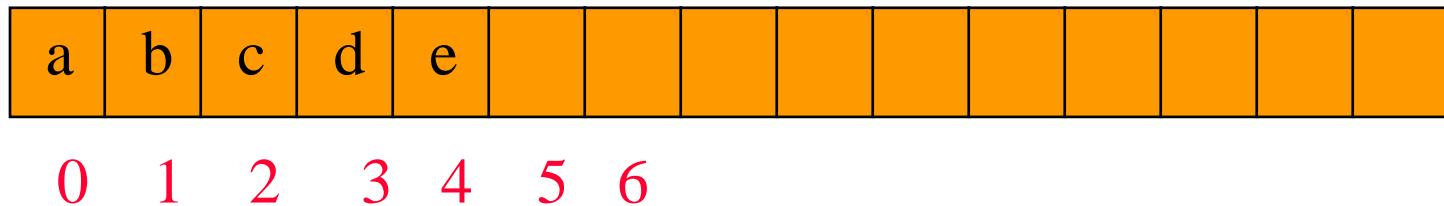
- стекын орой нь шугаман жагсаалтын зүүн, баруун төгсгөлийн нэг байна
- empty() => isEmpty()
- peek() => get(0) эсхүл get(size() - 1)

# ArrayList –ээс уламжлах



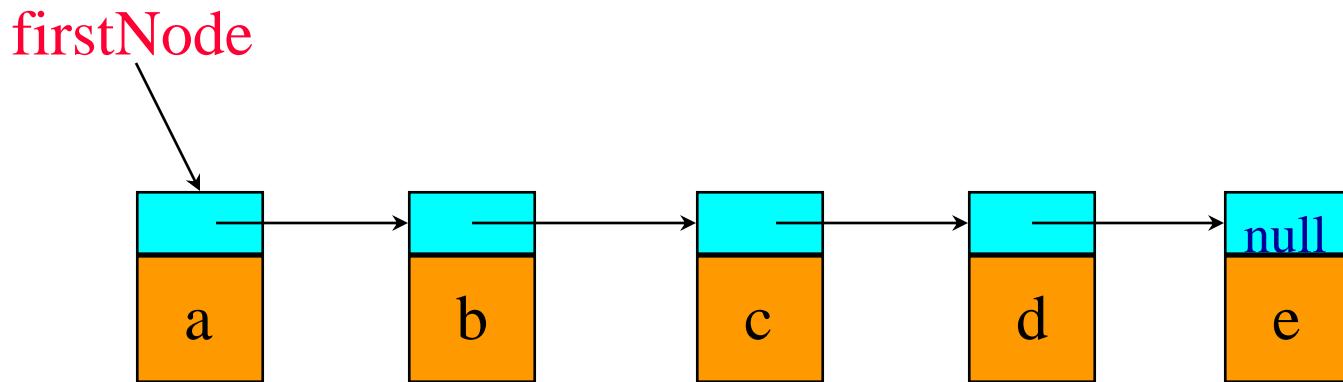
- шугаман жагсаалтын зүүн төгсгөл нь орой бол
  - `push(theObject)` => `add(0, theObject)`
  - `pop()` => `remove(0)`

# ArrayList –ээс уламжлах



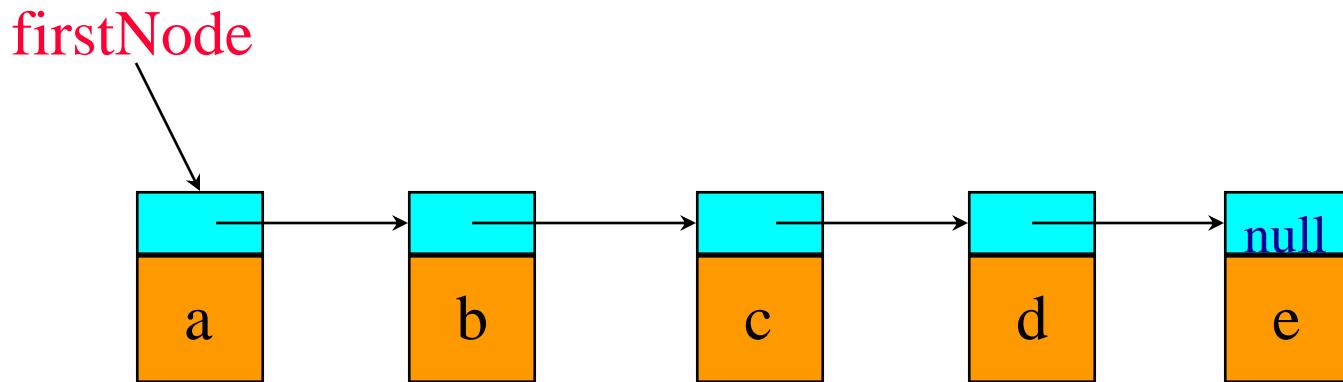
- шугаман жагсаалтын баруун төгсгөл орой бол
  - push(theObject) => add(size(), theObject)
  - pop() => remove(size()-1)
- жагсаалтын баруун төгсгөлийг стекын орой болгож ашиглая

# Chain –ээс уламжлах



- Шугаман жагсаалтын зүүн, баруун төгсгөлийн нэг стекын орой
- `empty()` => `isEmpty()`

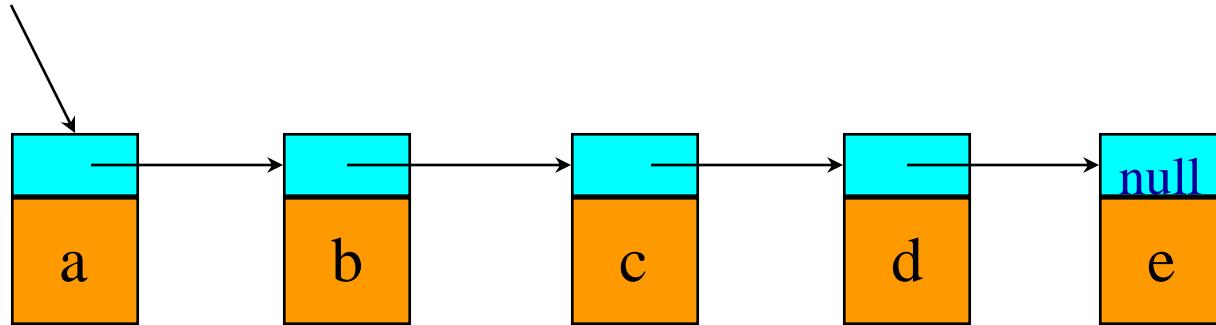
# Chain –ээс уламжлах



- шугаман жагсаалтын зүүн төгсгөл орой бол
  - peek() => get(0)
  - push(theObject) => add(0, theObject)
  - pop() => remove(0)

# Chain –ээс уламжлах

firstNode



— шугаман жагсаалтын баруун төгсгөл орой бол

- peek() => get(size() - 1)
- push(theObject) => add(size(), theObject)
- pop() => remove(size()-1)
- жагсаалтын зүүн төгсгөлийг стекын орой болгож ашиглай

# ArrayList –ЭЭС уламжлах

```
package dataStructures;  
import java.util.*; // онцгой тохиолдолд  
  
public class DerivedArrayList  
    extends ArrayList  
    implements Stack  
{  
    // байгуулагчид  
    // Stack интерфейсийн аргууд  
}
```



# Байгуулагчид



/\*\* тодорхой багтаамжтай стек  
байгуулах \*/

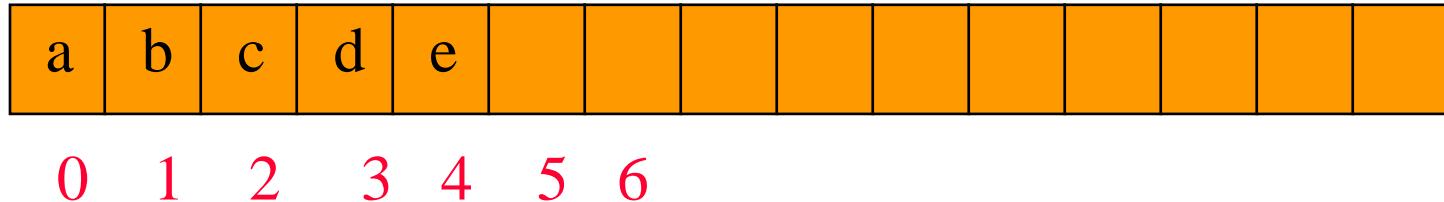
```
public DerivedArrayStack(int initialCapacity)  
{ super(initialCapacity); }
```

/\*\* 10 –ын багтаамжтай стек байгуулах\*/

```
public DerivedArrayStack()  
{ this(10); }
```



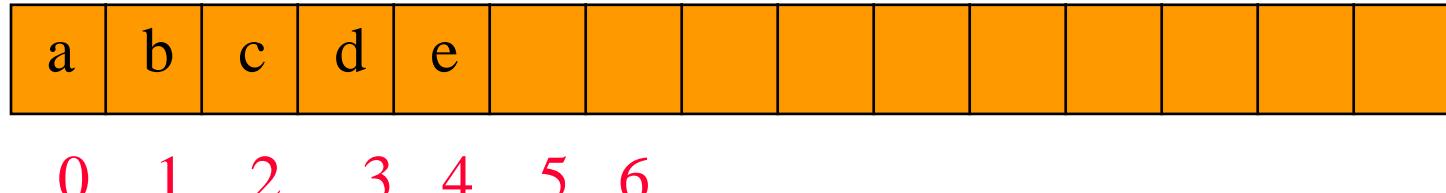
# empty() һа peek()



```
public boolean empty()  
{ return isEmpty(); }
```

```
public Object peek()  
{  
    if (empty())  
        throw new EmptyStackException();  
    return get(size() - 1)  
}
```

# push(theObject) һәм pop()



```
public void push(Object theElement)  
{ add(size(), theElement); }
```

```
public Object pop()  
{  
    if (empty())  
        throw new EmptyStackException();  
    return remove(size() - 1);  
}
```

# ДҮГНЭЛТ

- **ArrayList** –ээс уламжлахын давуу тал
  - Уламжлагдсан классын код маш энгийн, хөгжүүлэхэд амар.
  - Кодыг бага зэрэг зүгшрүүлэх.
  - Стекыг холбоосоор хэрэгжүүлэх кодыг амархан гарган авч болно.
    - **extends ArrayList** -г **extends Chain** –ээр солино
    - Бүтээмжийг дээшлүүлэх үүднээс жагсаалтын зүүн төгсгөлийг орой болгохоор өөрчлөлт оруулах хэрэгтэй.

# Дутагдал

- **ArrayList**-ийн бүх public аргуудыг стект хэрэглэж болно.
  - `get(0)` ... ёроолын элементийг авах
  - `remove(5)`
  - `add(3, x)`
  - Тэгэхээр жинхэнэ стекын хэрэгжилт биш болж байна.
  - Хэрэглэгдэхгүй аргуудыг давхар ачаалах ёстой.

```
public Object get(int theIndex)
```

```
{throw new UnsupportedOperationException();}
```

Өмнө ашигласан `get(i)` -г `super.get(i)` болгож өөрчилнө

# Дутагдал

- Код шаардлагагүй ажлыг хийдэг.
  - `peek()` стек хоосон эсэхийг `get` дуудахаас өмнө шалгадаг. Тэгэхээр индексийг шалгах `get` шаардлагагүй болж байна.
  - `add(size(), theElement)` индексийг шалгадаг, орохгүй `for` давталт орсон. Аль нь ч хэрэггүй.
  - `pop()` болохоор `remove` –г дуудахаас өмнө стек хоосон эсэхийг шалгадаг. `remove` индекс шалгаж, орохгүй `for` тавталттай. Аль нь ч хэрэггүй.
  - Иймд код хэрэгцээгүй удаан ажиллана.

# ДҮГНЭЛТ

- Кодыг шинээр бичсэнээр хурдан ажиллах боловч хөгжүүлэх гэж хугацаа зарна.
- Програм хангамж хөгжүүлэх зардал, чанарын үзүүлэлт хоёроос сонгох хэрэгтэй.
- Зах зээл гарах хугацаа, чанарын үзүүлэлт хоёроос сонгох хэрэгтэй.
- Анхны кодыг амархан хийгээд, дараа нь чанарын үзүүлэлтийг сайжруулах.



# Хурдан pop()

```
if (empty())
    throw new EmptyStackException();
return remove(size() - 1);
```

оронд нъ

```
try {return remove(size() - 1);}
catch(IndexOutOfBoundsException e)
{throw new EmptyStackException();}
```

# ЭХНЭЭС НЬ КОДЧИЛОХ

- 1D stack массив ашиглах, төрөл нь Object.
  - `ArrayList` –д массив `element` ашигласан шиг
- int `top` хувьсагч ашиглах
  - Стекын элементүүд `stack[0:top]` -д
  - Оройн элемент нь `stack[top]`.
  - Ёроолын элемент нь `stack[0]`.
  - `top = -1` бол стек хоосон
  - Стек дэх элементийн тоо `top+1`.



# ЭХНЭЭС НЬ КОДЧИЛОХ

```
package dataStructures;  
import java.util.EmptyStackException;  
import utilities.*; // ChangeArrayLength  
public class ArrayStack implements Stack  
{  
    // Өгөгдөл гишүүд  
    int top;          // Стекын орой  
    Object [] stack; // Элементийн массив  
    // байгуулагчид  
    // Stack интерфейсийн аргууд  
}
```

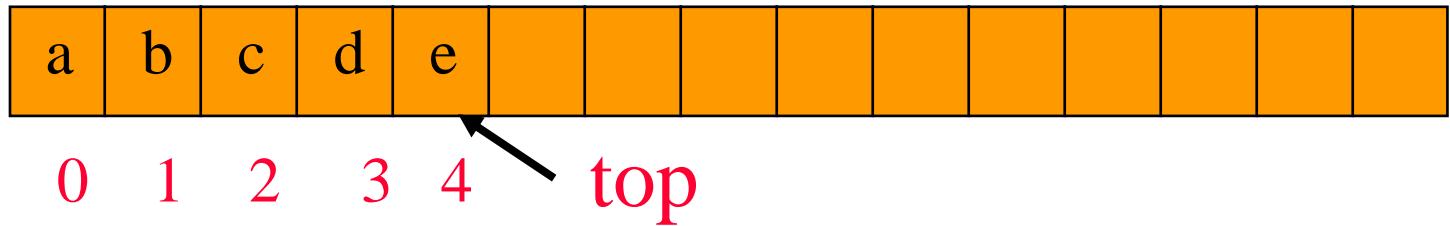


# Байгуулагчид



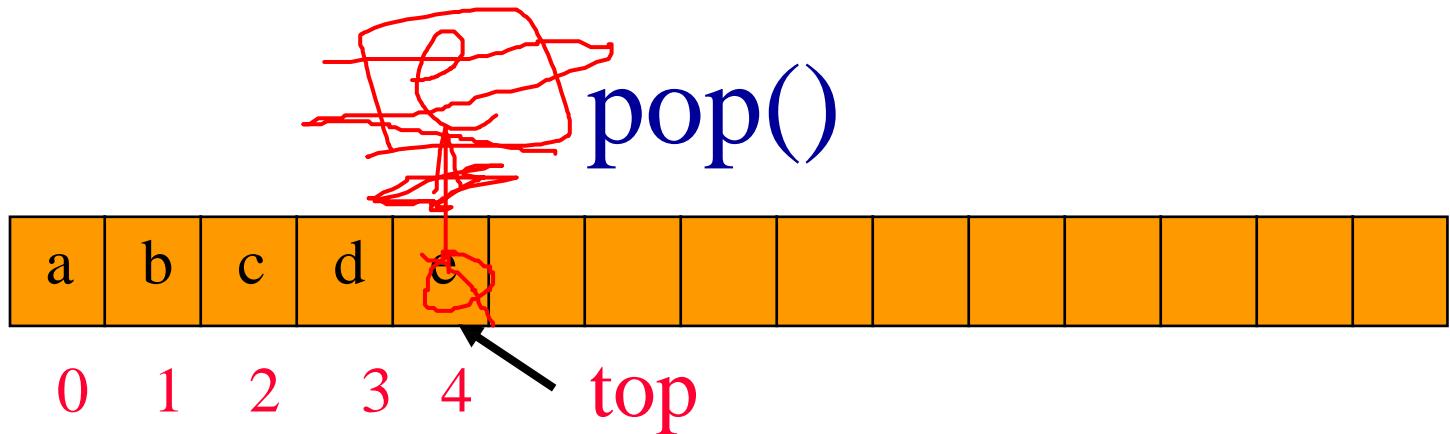
```
public ArrayStack(int initialCapacity)
{
    if (initialCapacity < 1)
        throw new IllegalArgumentException
            ("initialCapacity must be >= 1");
    stack = new Object [initialCapacity];
    top = -1;
}
public ArrayStack()
{ this(10);}
```

# push(...)



```
public void push(Object theElement)
{
    // хэрэгтэй бол массивын хэмжээг ихэсгэх
    if (top == stack.length - 1)
        stack = ChangeArrayLength.changeLength1D
            (stack, 2 * stack.length);

    // theElement стекын оройд хийх
    stack[++top] = theElement;
}
```



```

public Object pop()
{
    if (empty())
        throw new EmptyStackException();
    Object topElement = stack[top];
    stack[top--] = null; // хаягдалд өгөх
    return topElement;
}

```

# Холбоосон стекыг эхнээс нь кодчилох

- Сурах бичгээс харна уу.

# java.util.Stack

- **java.util.Vector** -аас уламжлагдсан
- **java.util.Vector** бол шугаман жагсаалтын массив хэрэгжүүлэлт.

# Чанарын үзүүлэлт



500,000 `pop`, `push`, `peek` үйлдэл

| Класс                     | анхны багтаамж |         |
|---------------------------|----------------|---------|
|                           | 10             | 500,000 |
| ArrayStack                | 0.44s          | 0.22s   |
| DerivedArrayList          | 0.60s          | 0.38s   |
| DerivedArrayListWithCatch | 0.55s          | 0.33s   |
| java.util.Stack           | 1.15s          | -       |
| DerivedLinkedList         | 3.20s          | 3.20s   |
| LinkedList                | 2.96s          | 2.96s   |

# Дараалал



- Шугаман жагсаалт.
- Нэг төгсгөл нь **front - нүүр**.
- Нөгөө төгсгөл нь **rear - сүүл**.
- Нэмэхдээ зөвхөн **сүүлд** нь.
- Устгахдаа зөвхөн **нүүрнийх** нь.

# Автобусны буудлын дараалал



# Автобусны буудлын дараалал



front



rear



# Автобусны буудлын дараалал



front

rear



# Автобусны буудлын дараалал



front

rear



# Интерфейс - Queue

```
public interface Queue
```

```
{
```

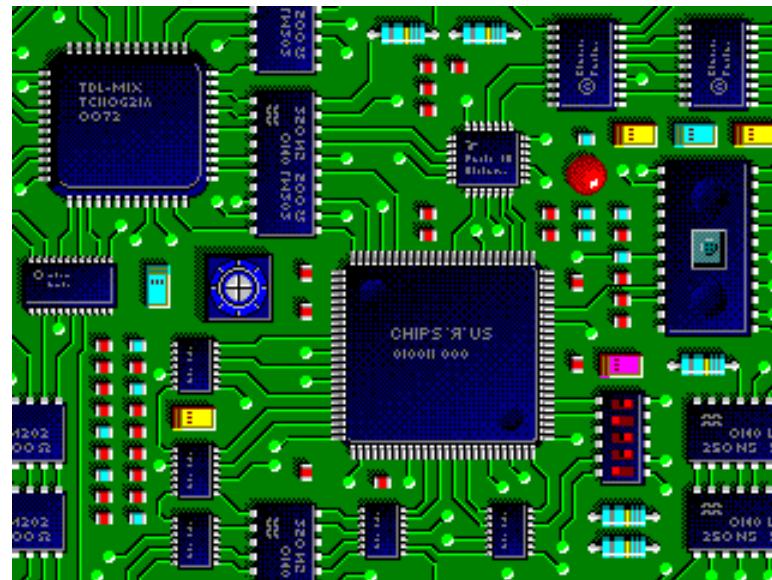
```
    public boolean isEmpty();  
    public Object getFrontElement();  
    public Object getRearElement();  
    public void put(Object theObject);  
    public Object remove();
```

```
}
```

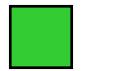
# Стек хэрэглэсэн жишээг эргэж харья

- Стекыг дарааллаар сольж болохгүй жишээ.
  - Хаалт хослох.
  - Ханойн цамхаг.
  - Switchbox routing.
  - Аргыг дуудах ба буцах.
  - Try-catch-throw хэрэгжүүлэлт.
- Стекыг дарааллаар сольж болох жишээ.
  - Төөрөлдсөн оготно.
    - Үр дүн нь гарах дөт замыг олох.

# Зам гаргах



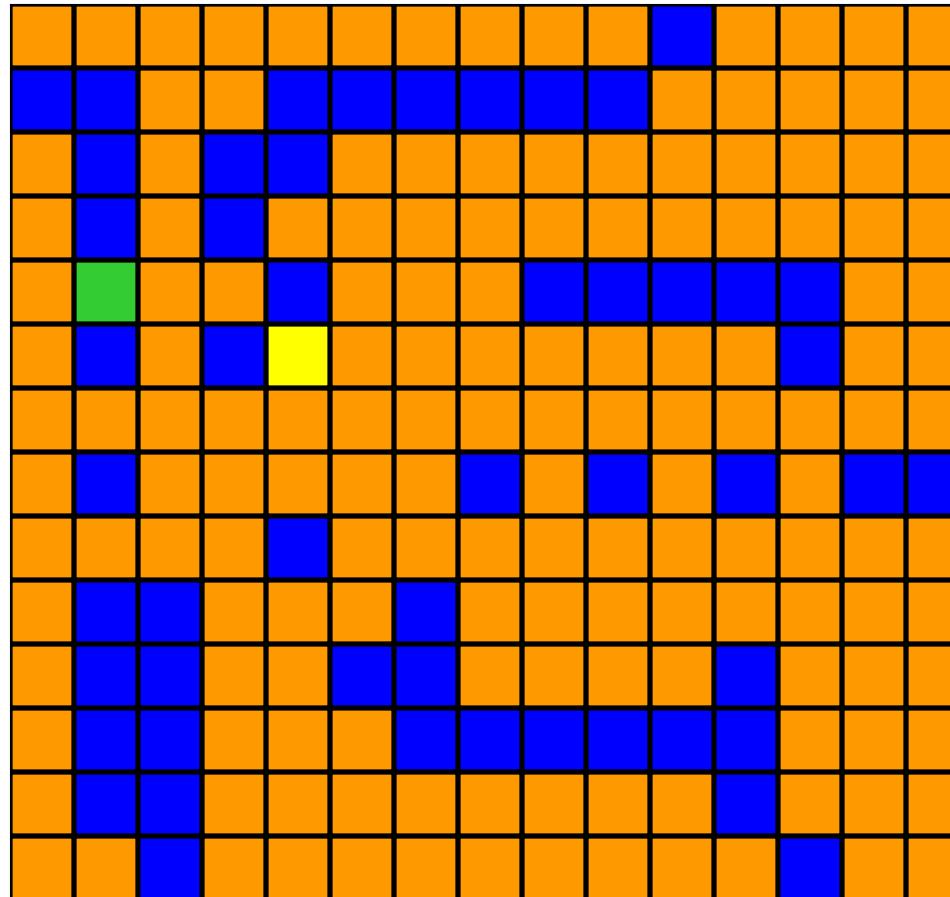
# Lee'ын зам гаргагч



start pin



end pin



Гараанаас хүрч болох нүдийг **1** гэж  
тэмдэглэе.

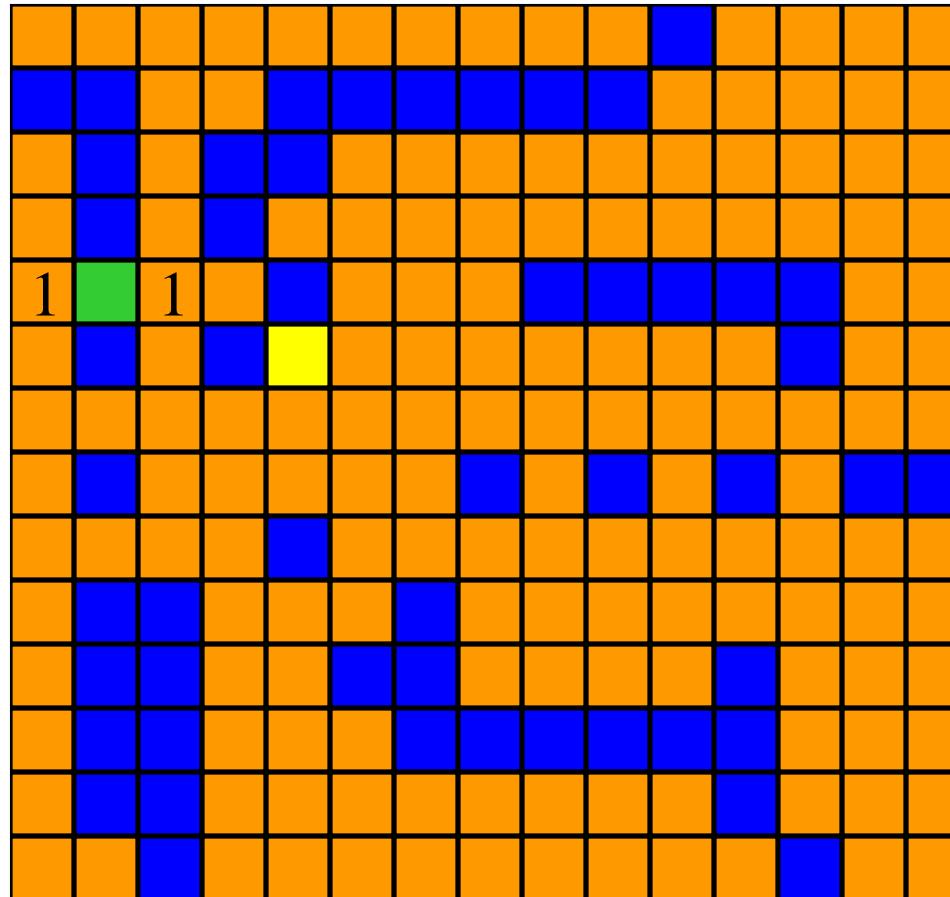
# Lee'ын зам гаргагч



start pin



end pin

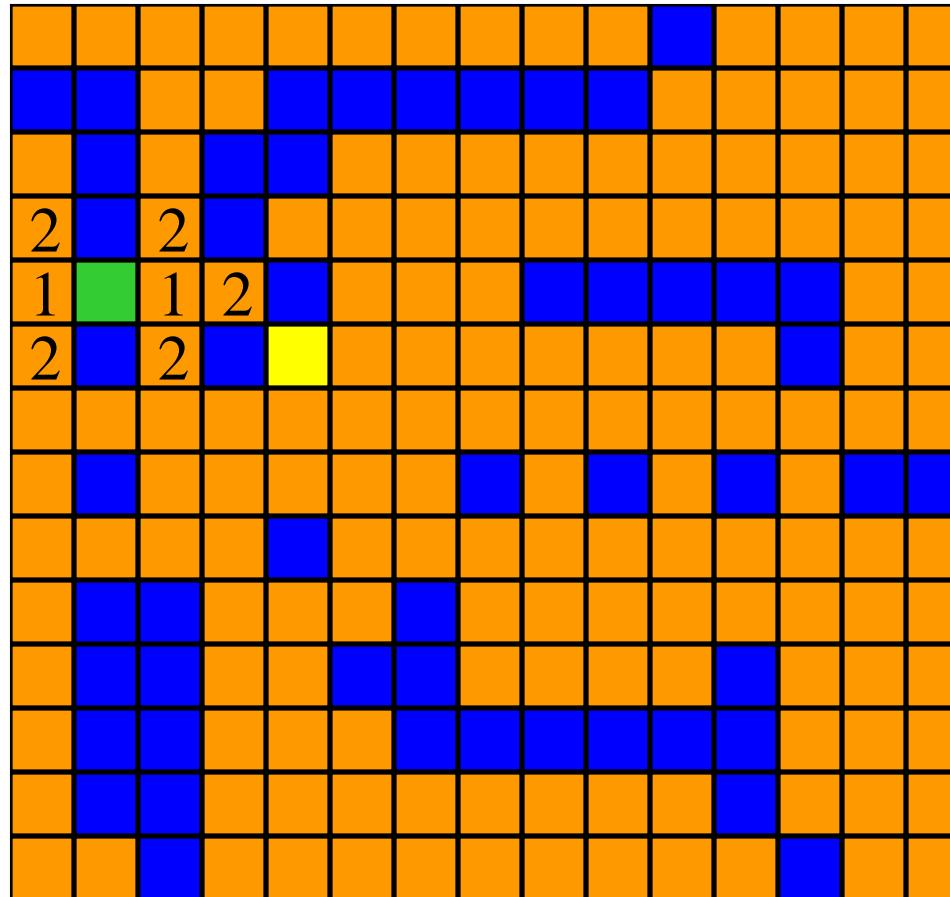


Гараанаас хүрч болох нүдийг **2** гэж  
тэмдэглэе.

# Lee'ын зам гаргагч

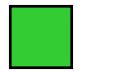
start pin

end pin



Гараанаас хүрч болох нүдийг **3** гэж  
тэмдэглэе.

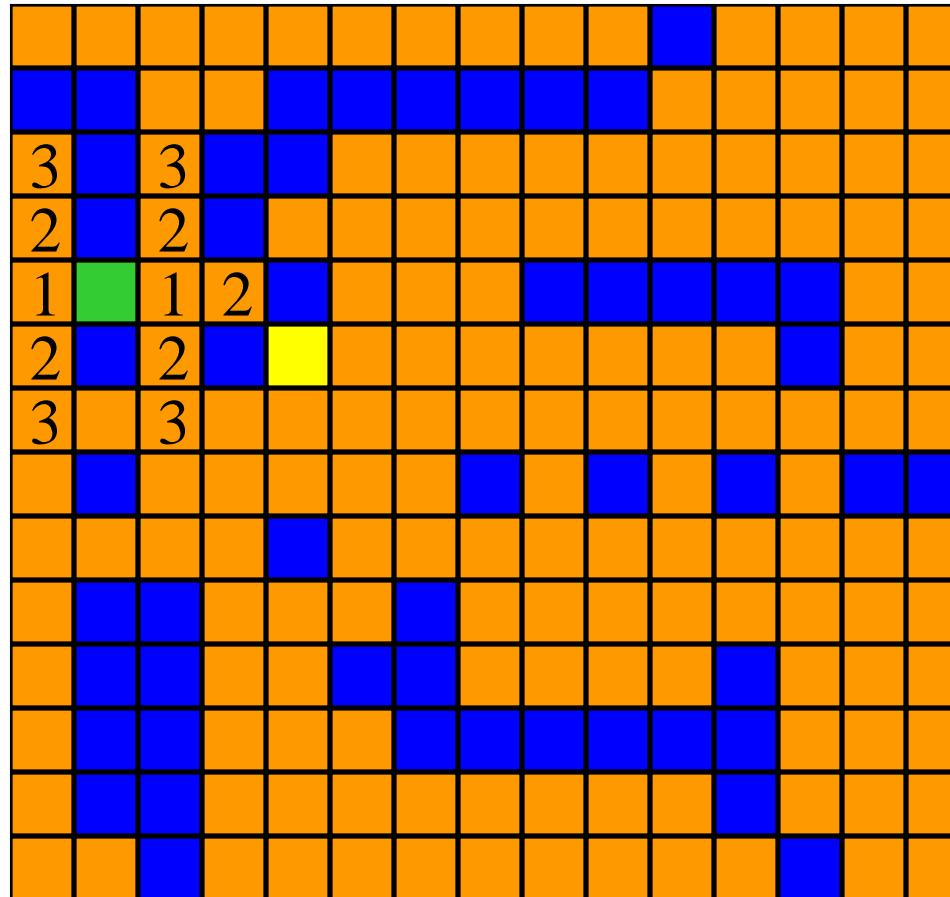
# Lee'ын зам гаргагч



start pin



end pin

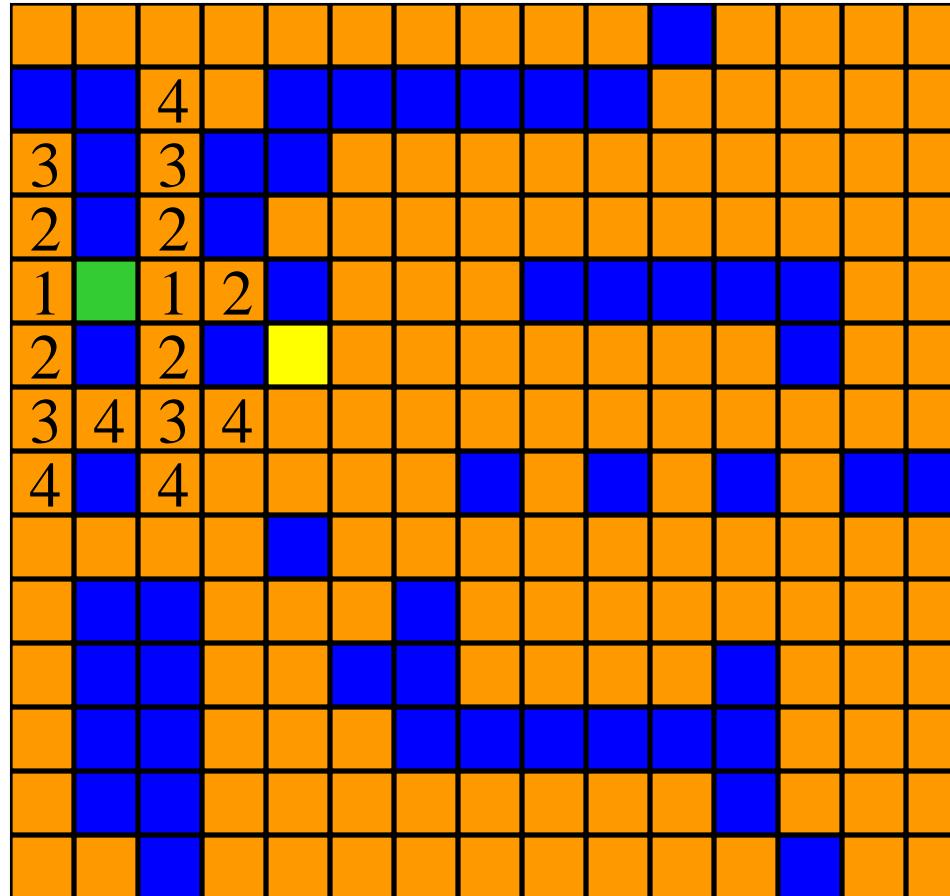


Гараанаас хүрч болох нүдийг **4** гэж  
тэмдэглэе.

# Lee'ын зам гаргагч

 start pin

 end pin

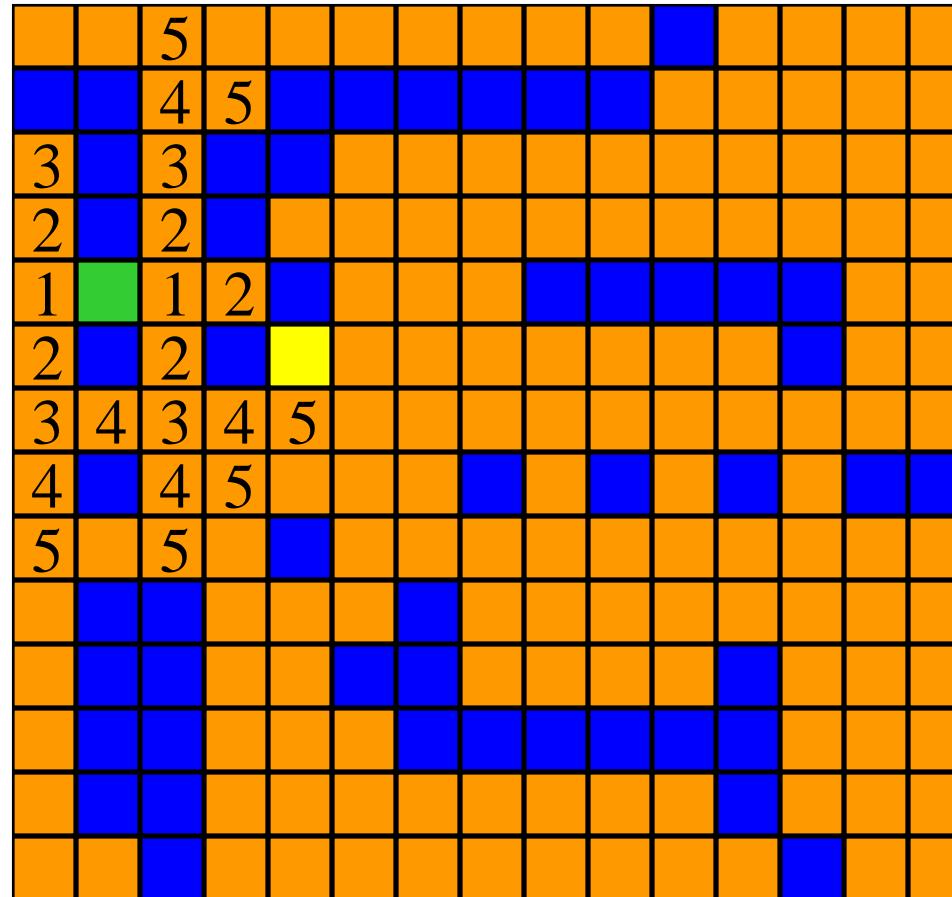


Гараанаас хүрч болох нүдийг **5** гэж  
тэмдэглэе.

# Lee'ын зам гаргагч

 start pin

 end pin

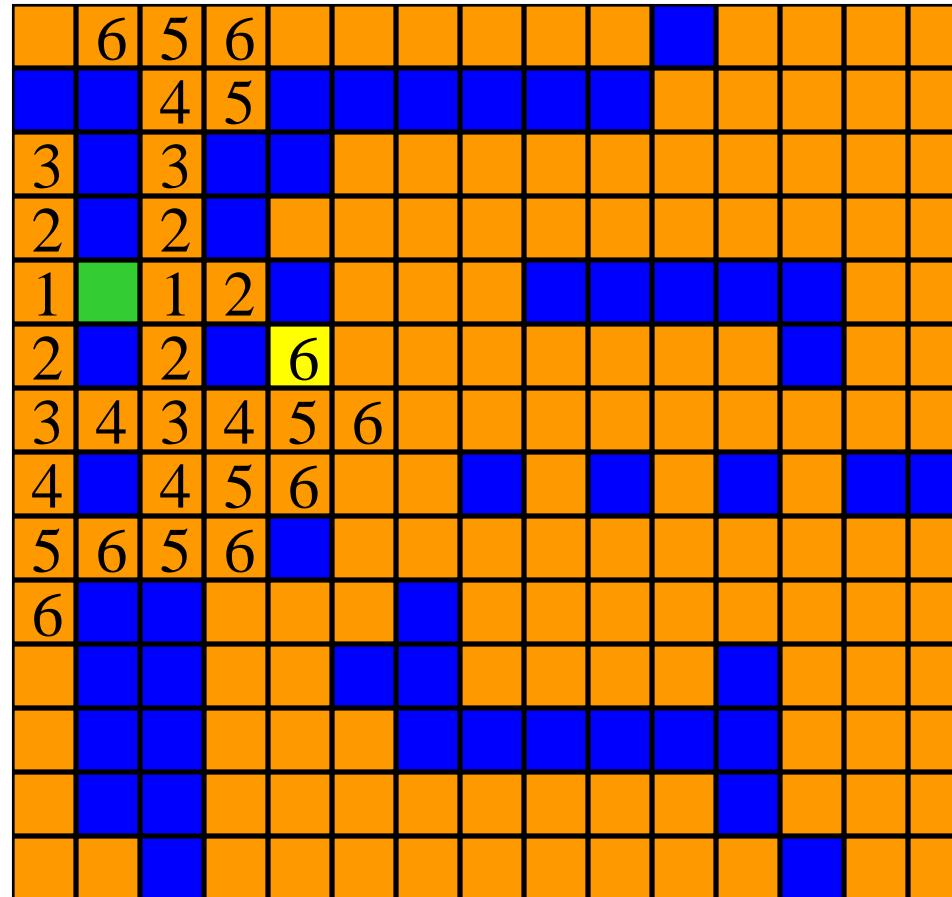


Гараанаас хүрч болох нүдийг **6** гэж  
тэмдэглэе.

# Lee'ын зам гаргагч

start pin

end pin

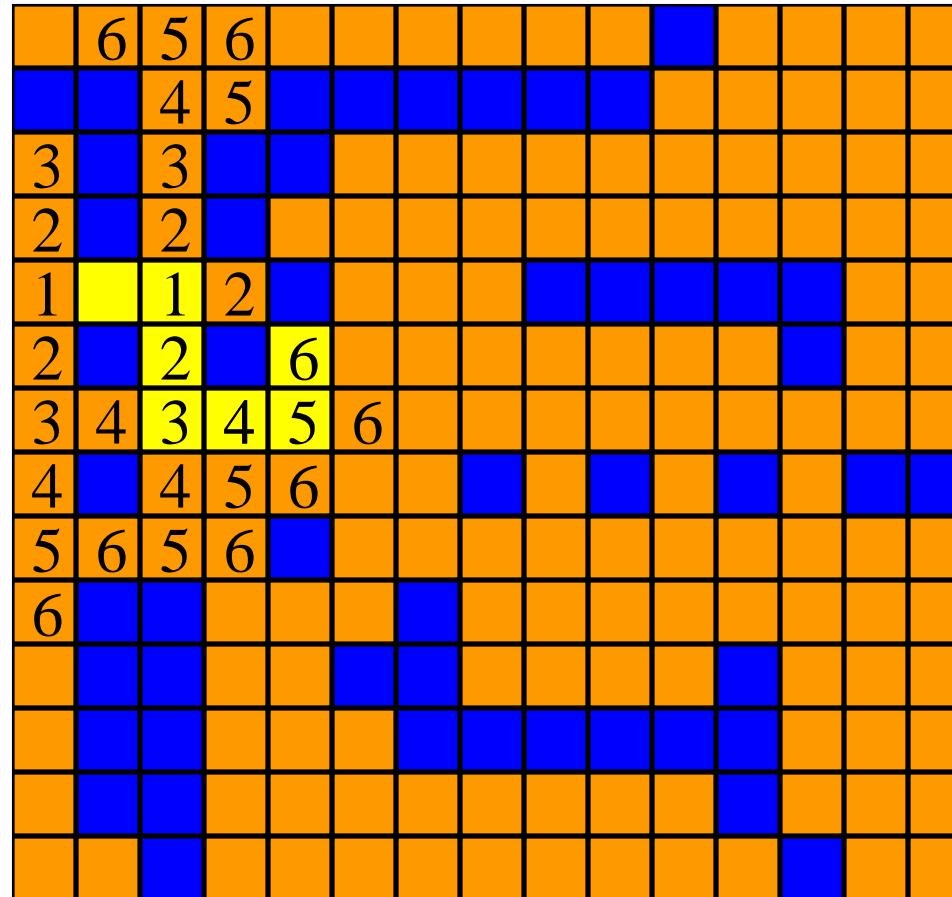


Төгсгөлд хүрлээ. Буцья.

# Lee'ын зам гаргагч

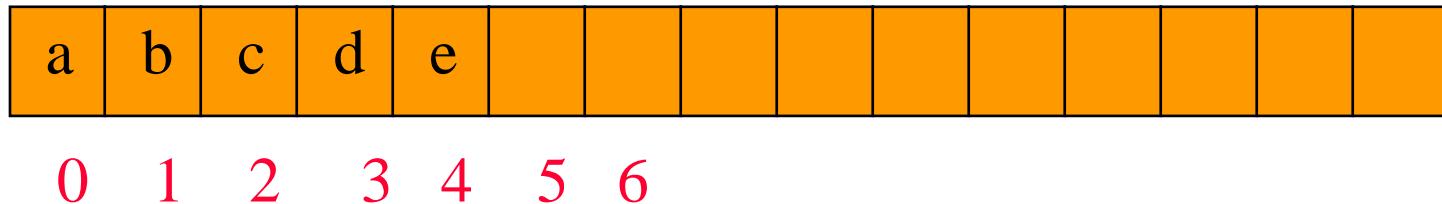
 start pin

 end pin



Төгсгөлд хүрлээ. Буцья.

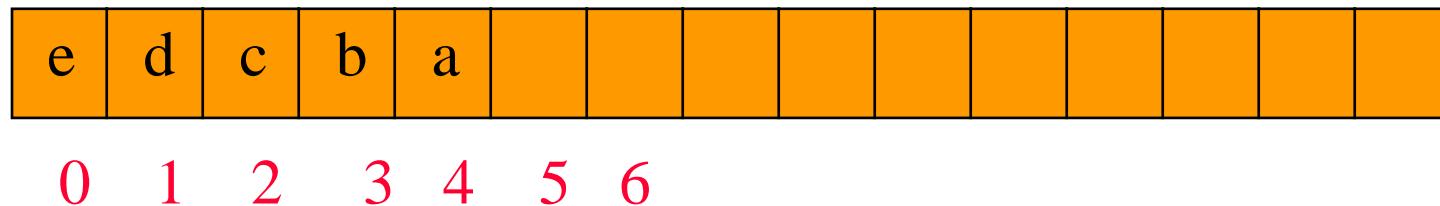
# ArrayList – ээс уламжлах



➤ Жагсаалтын зүүн төгсгөл нүүр, баруун төгсгөл нь сүүл бол

- Queue.isEmpty() => super.isEmpty()
- getFrontElement() => get(0)
- getRearElement() => get(size() - 1)
- put(theObject) => add(size(), theObject)
- remove() => remove(0)

# ArrayList – ээс уламжлах

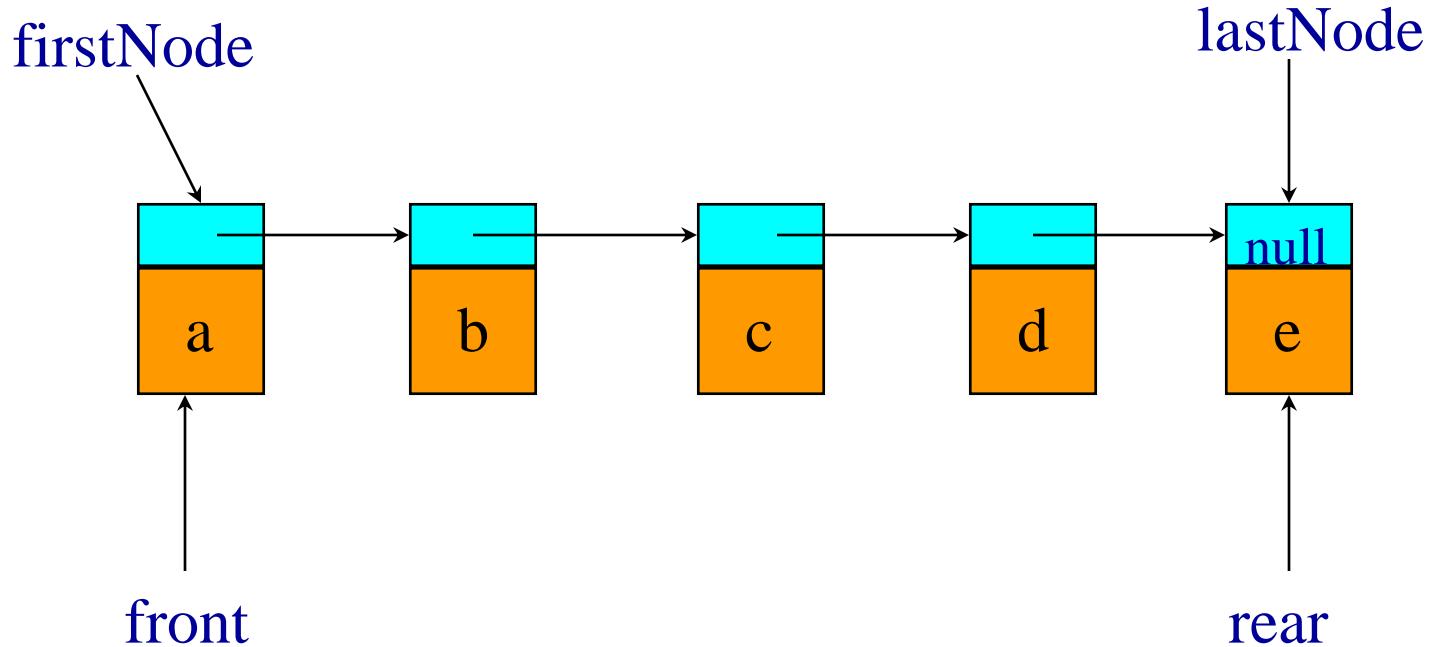


- Жагсаалтын зүүн төгсгөл сүүл, баруун төгсгөл нь нүүр бол
  - Queue.isEmpty() => super.isEmpty()
  - getFrontElement() => get(size() - 1)
  - getRearElement() => get(0)
  - put(theObject) => add(0, theObject)
  - remove() => remove(size() - 1)

# ArrayList –ЭЭС уламжлах

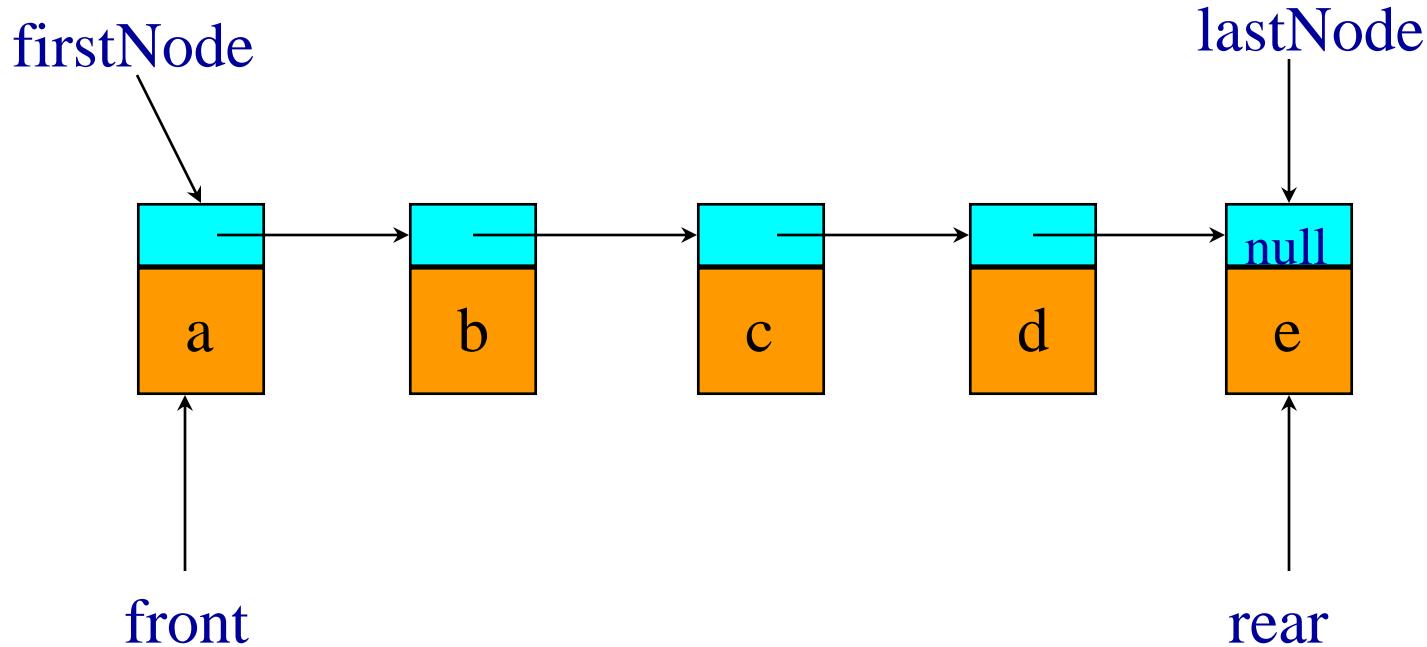
- Үйлдлүүдийг гүйцэтгэх хугацаа богино (массивыг хоёр дахин ихэсгэхээс бусад), бидэнд сайжруулсан массив дүрслэл хэрэгтэй

# ExtendedChain –ээс уламжлах



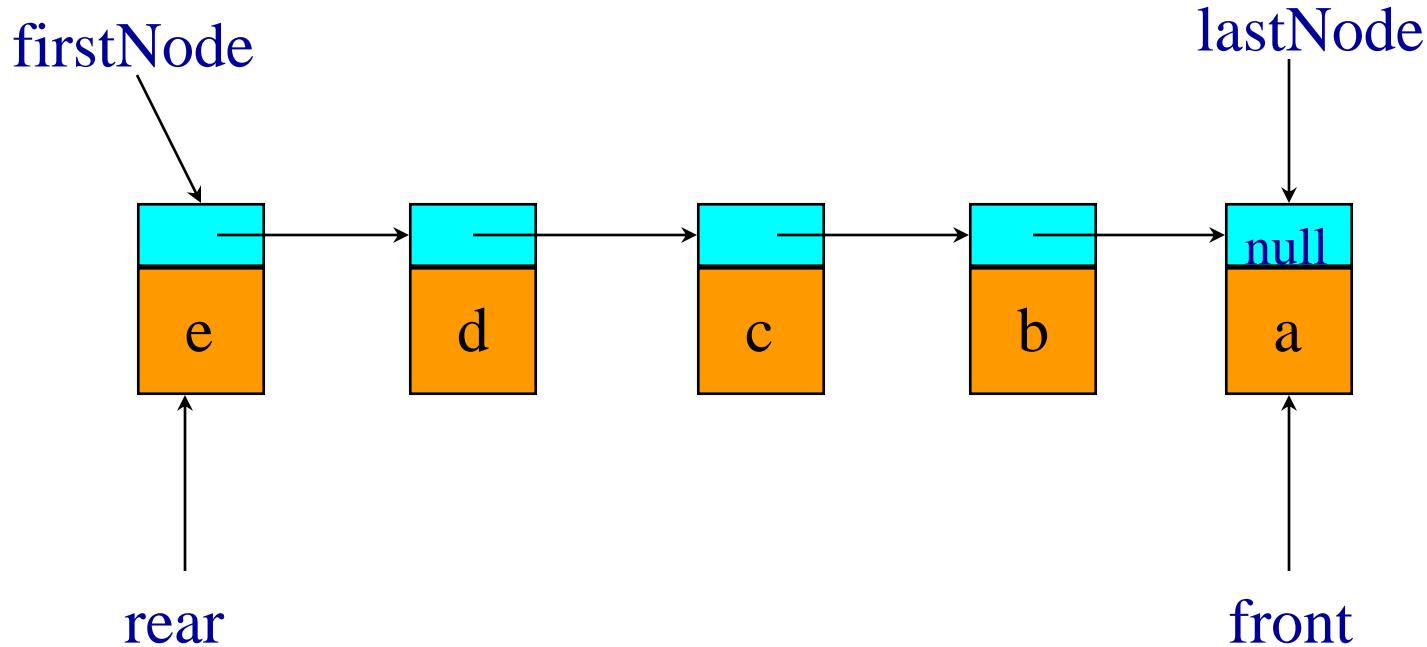
- жагсаалтын зүүн төгсгөл нь нүүр, баруун төгсгөл нь сүүл бол
  - Queue.isEmpty() => super.isEmpty()
  - getFrontElement() => get(0)

# ExtendedChain –ээс уламжлах



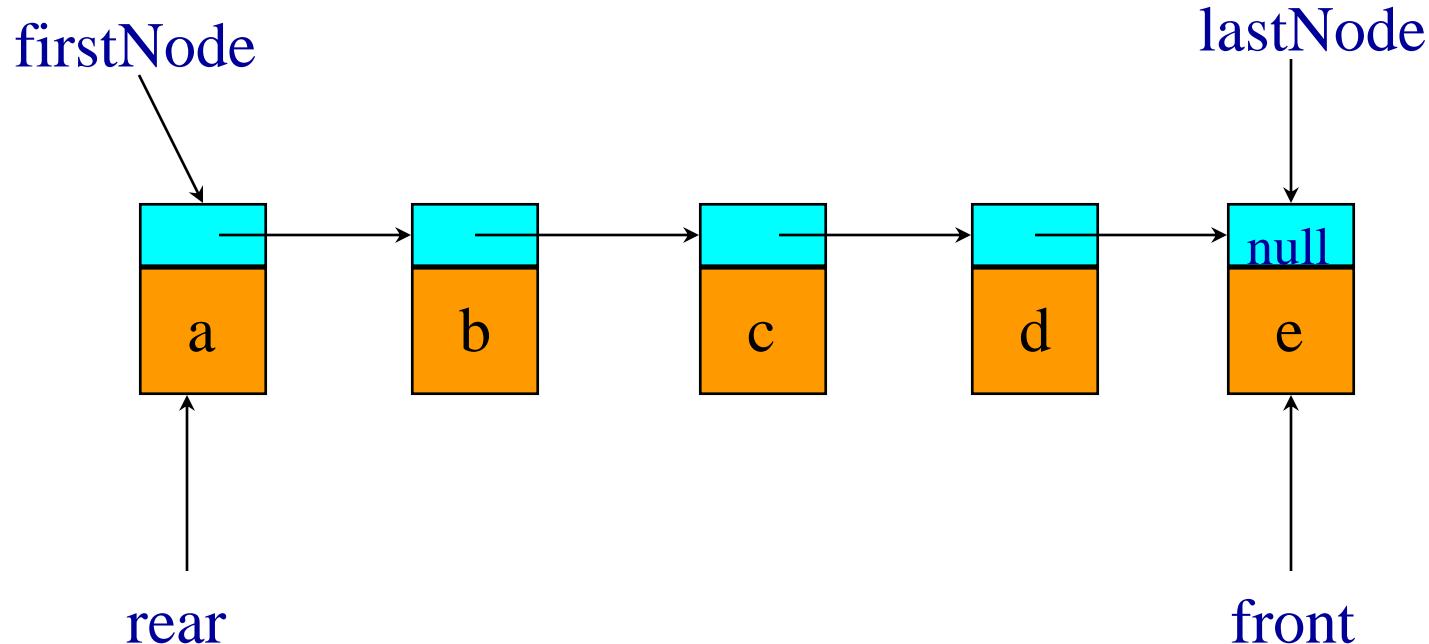
- `getRearElement() => getLast()` ... шинэ арга
  - `put(theObject) => append(theObject)`
  - `remove() => remove(0)`

# Derive From ExtendedChain



- Жагсаалтын зүүн төгсгөл сүүл, баруун төгсгөл нь нүүр бол
  - Queue.isEmpty() => super.isEmpty()
  - getFrontElement() => getLast()

# ExtendedChain –ээс уламжлах



- `getRearElement() => get(0)`
  - `put(theObject) => add(0, theObject)`
  - `remove() => remove(size-1)`

# Өөрчилсөн холбоост код

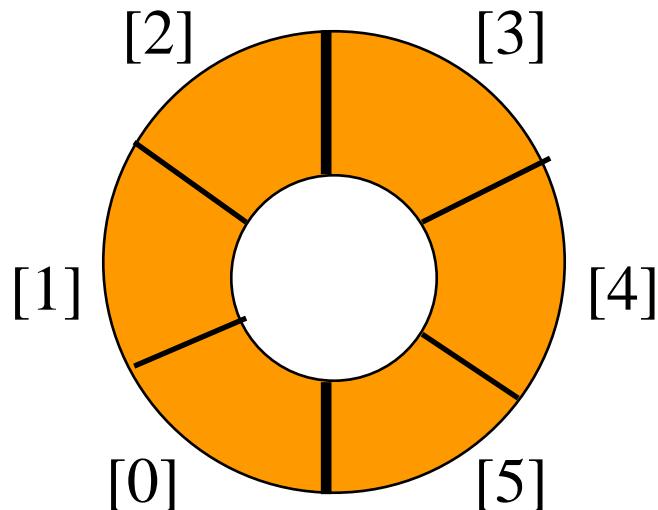
- **ExtendedChain** –ээс уламжилснаас сайн үзүүлэлт хэрэгтэй бол **Queue** –д зориулсан холбоост классыг эхнээс нь кодчилох хэрэгтэй

# Өөрчилсөн массив дараалал

- 1D массив **queue** -г ашиглай

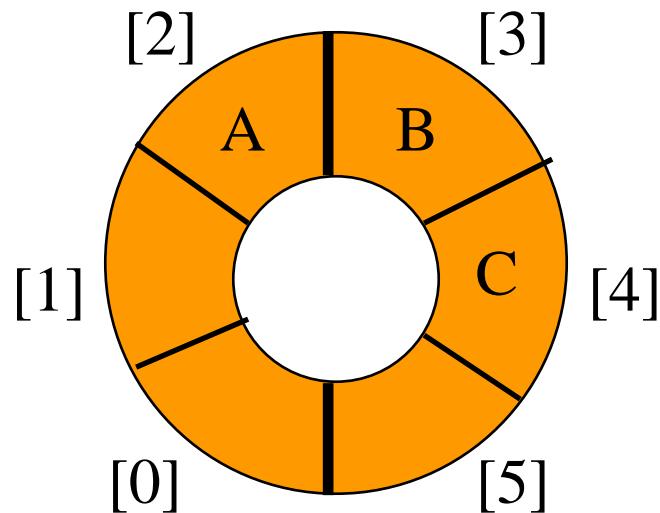
queue[] 

- Массивын цагираг харагдац.



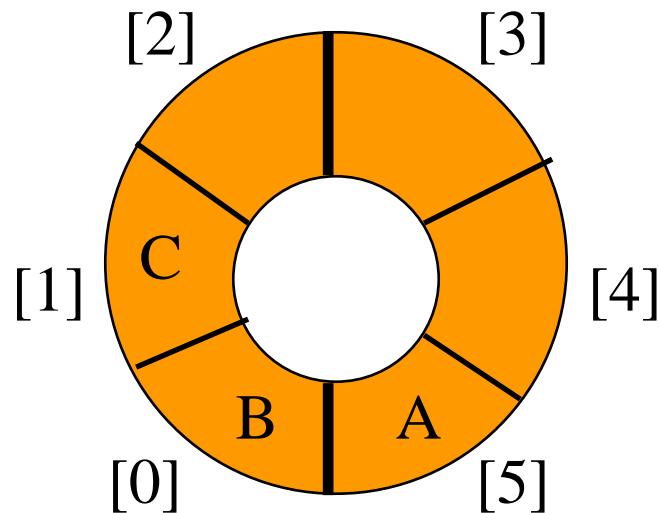
# Өөрчилсөн массив дараалал

- З элементтэй хувилбар.



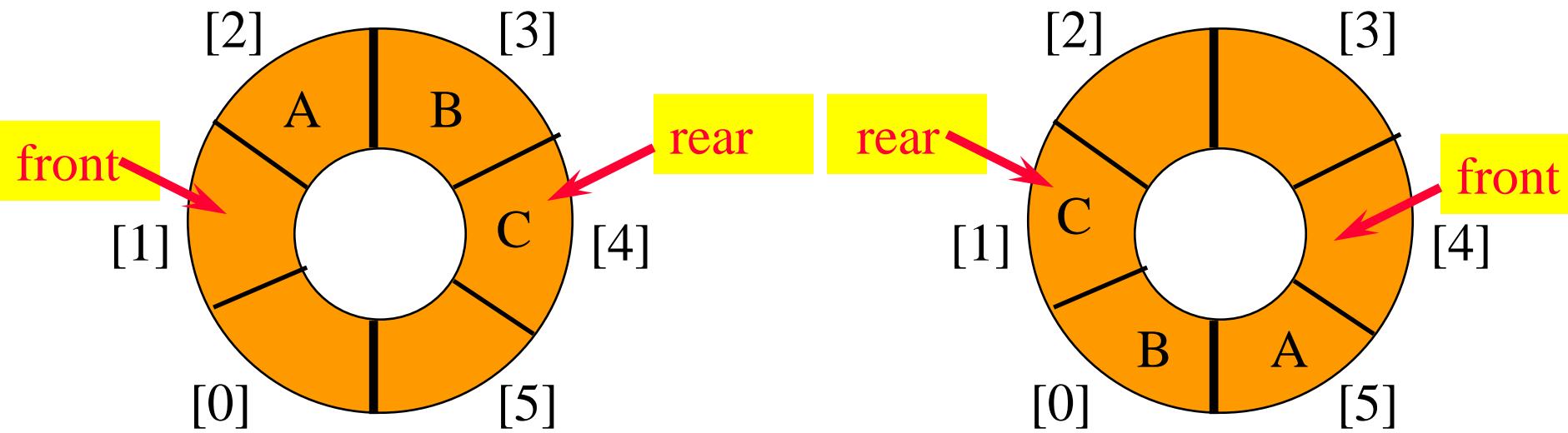
# Өөрчилсөн массив дараалал

- З элементтэй өөр нэг хувилбар.



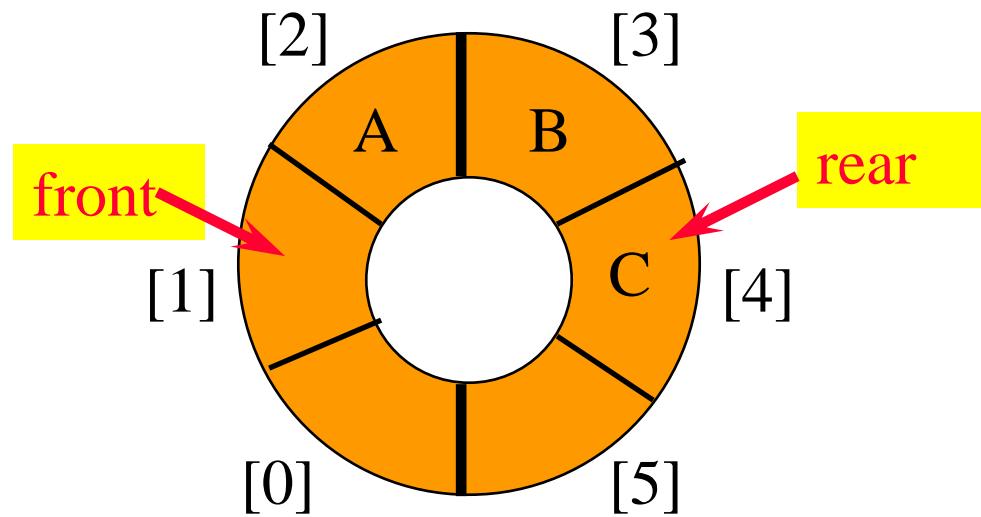
# Өөрчилсөн массив дараалал

- Бүхэл хувьсагч **front** , **rear** -г ашиглай
  - front** эхний элементээс цагийн зүүний дагуу явсан нэг байршил
  - rear** сүүлийн элементийн байршлыг өгнө



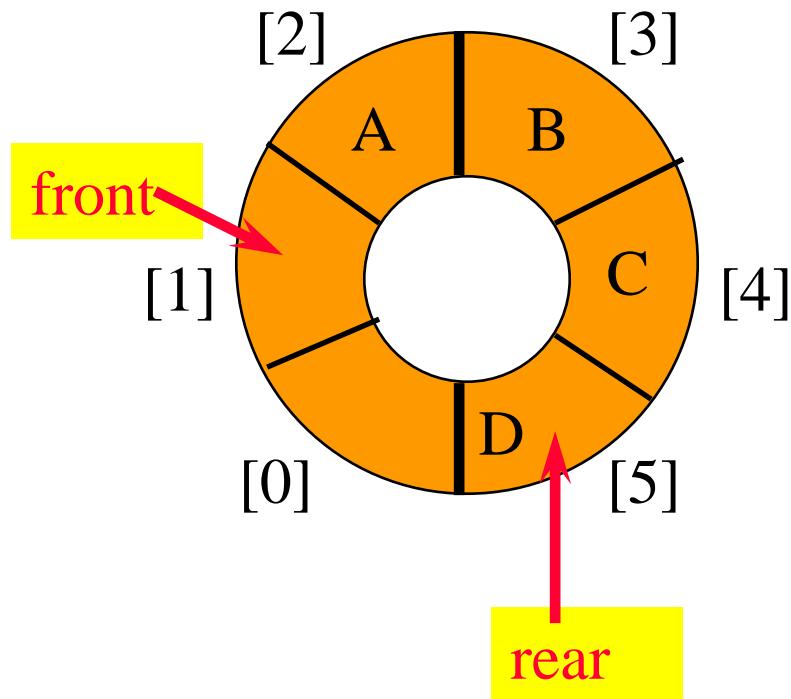
# Элемент нэмэх

- **rear** цагийн дагуу нэгээр хөдөлгөнө.



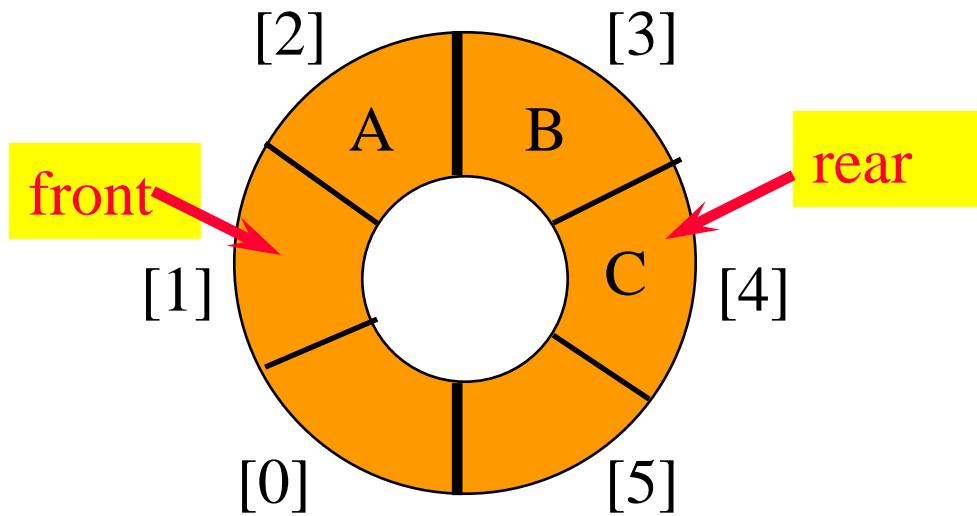
# Элемент нэмэх

- **rear** цагийн дагуу нэгээр хөдөлгөнө.
- **queue[rear]** -д элементийг хийнэ



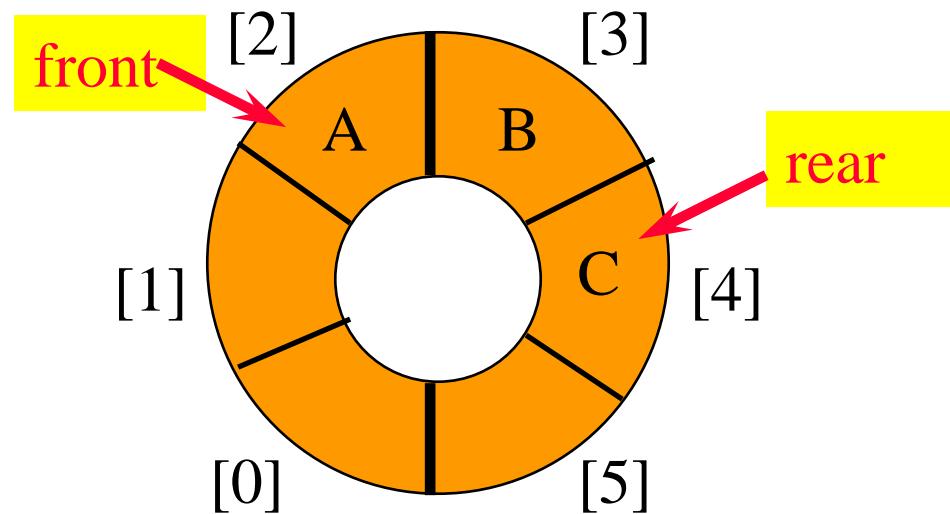
# Элементийг устгах

- **front** цагийн дагуу нэгээр хөдөлгөнө.



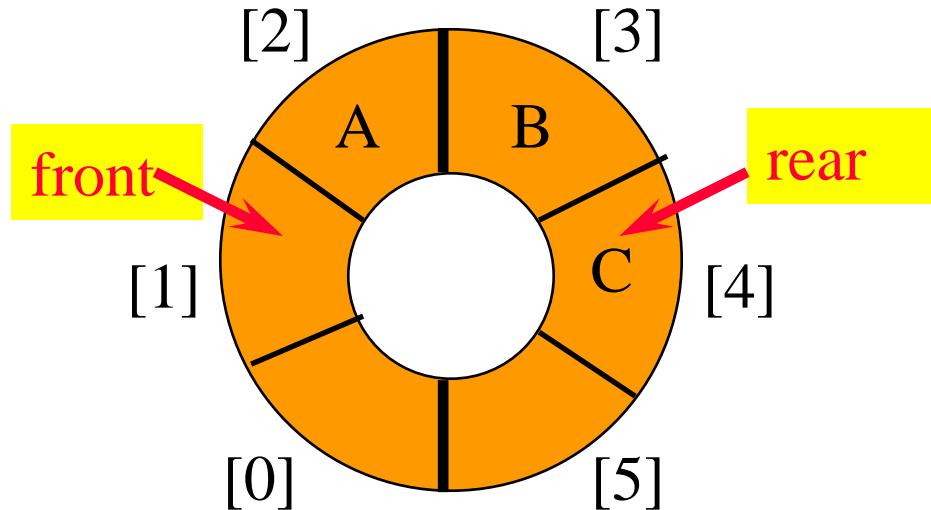
# Элемент устгах

- **front** цагийн дагуу нэгээр хөдөлгөнө.
- **queue[front]** -аас элемент авна.



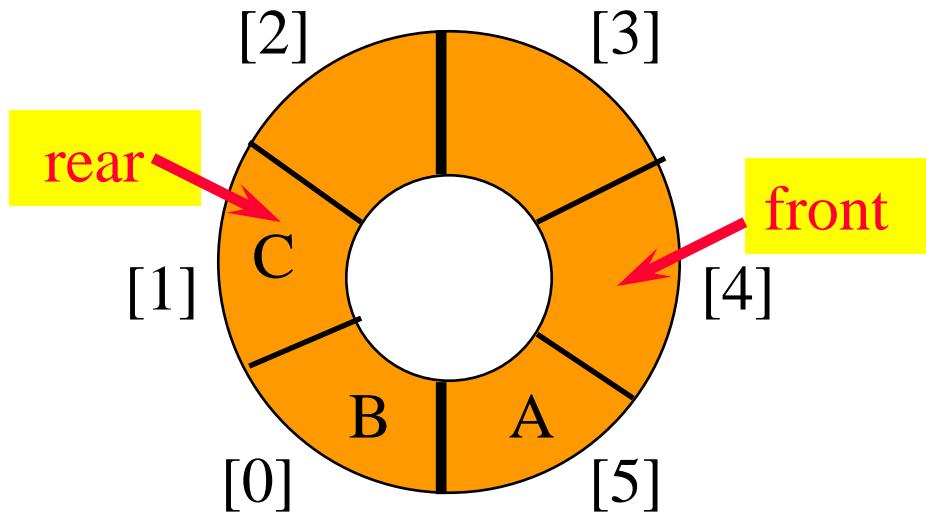
# rear –г цагийн дагуу хөдөлгөх

- $\text{rear}++;$
- if ( $\text{rear} == \text{queue.length}$ )  $\text{rear} = 0;$

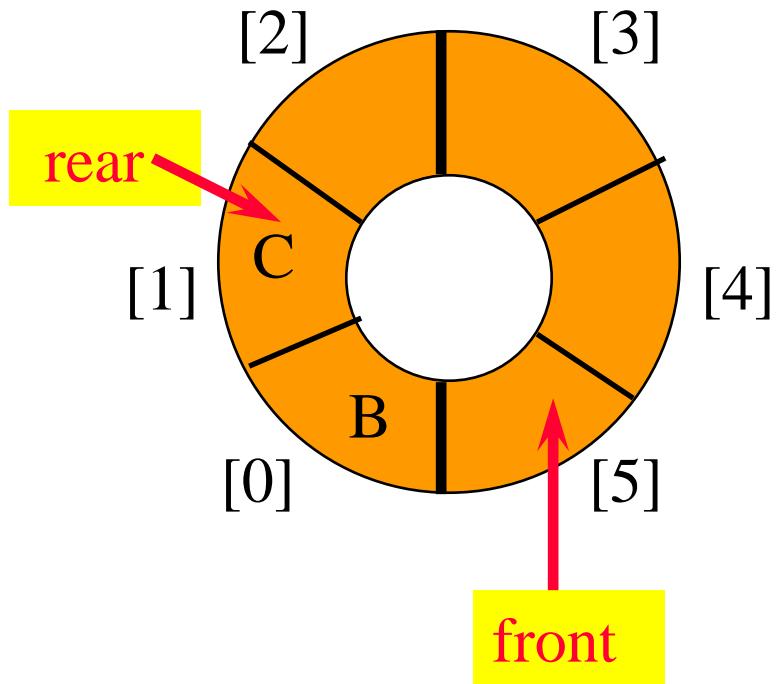


- $\text{rear} = (\text{rear} + 1) \% \text{queue.length};$

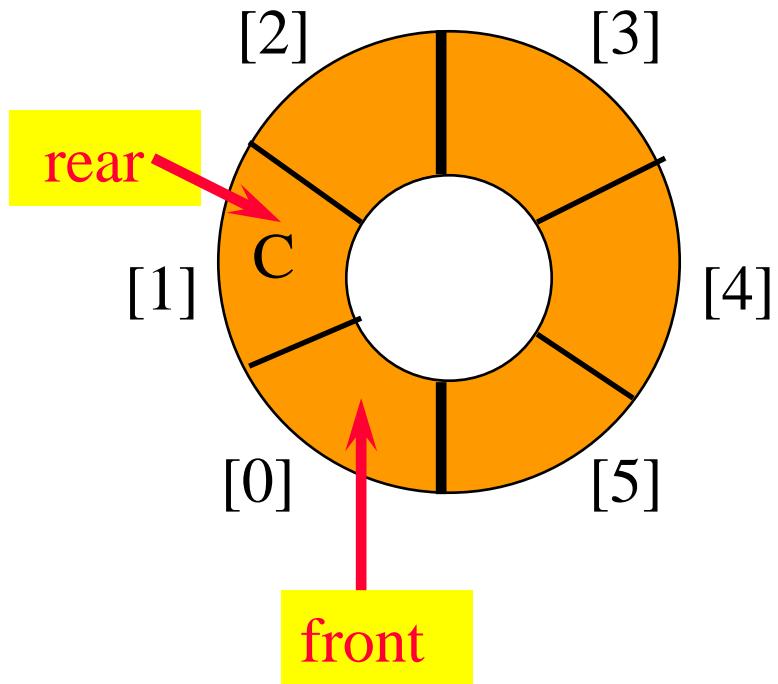
# Дарааллыг хоослох



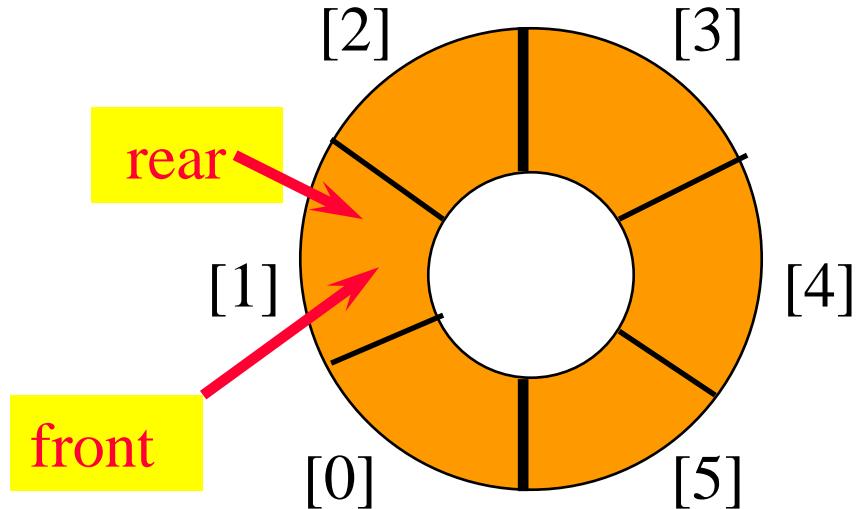
# Дарааллыг хоослох



# Дарааллыг хоослох

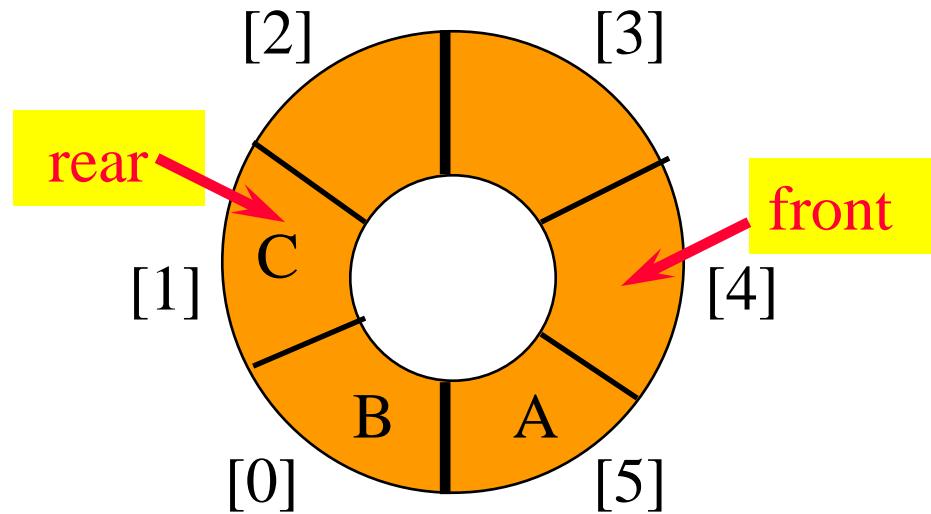


# Дарааллыг хоослох

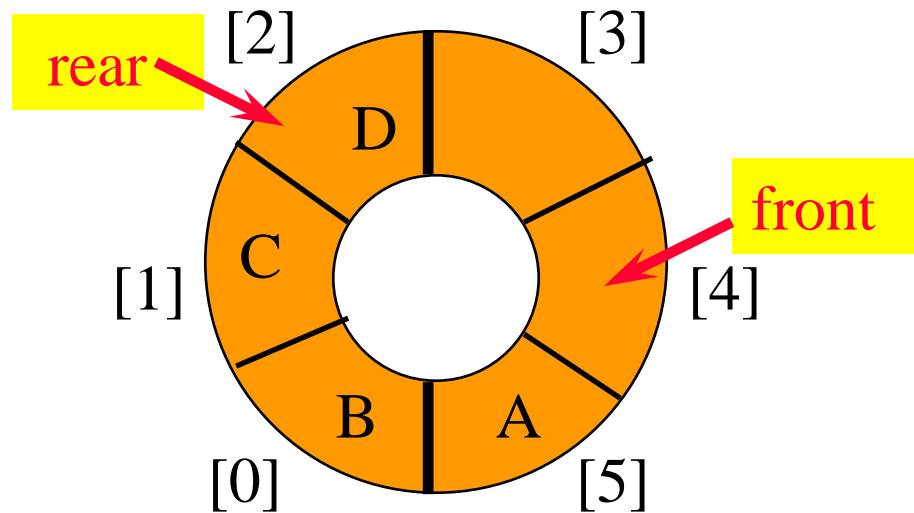


- Дараалсан устгалтуудаар **front = rear** болоход дараалал хоослогдоно
- Анх байгуулагдахад хоосон байна.
- Иймд эхэндээ **front = rear = 0**.

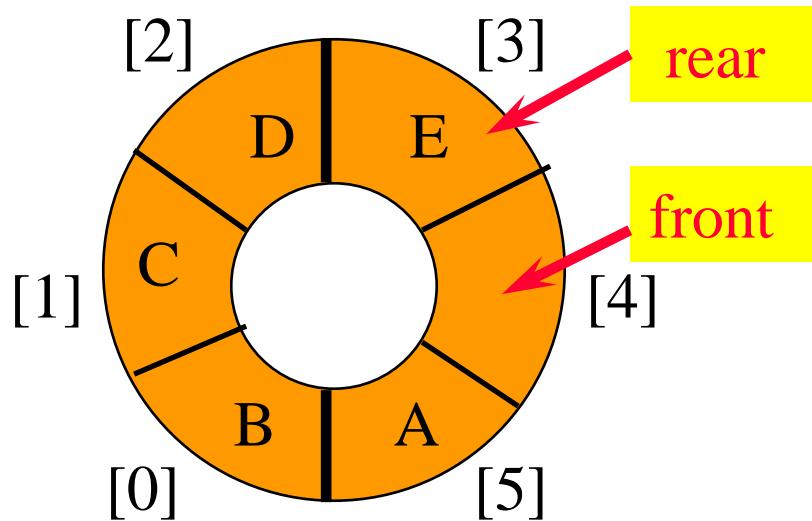
# Дарааллын савыг дүүргэх



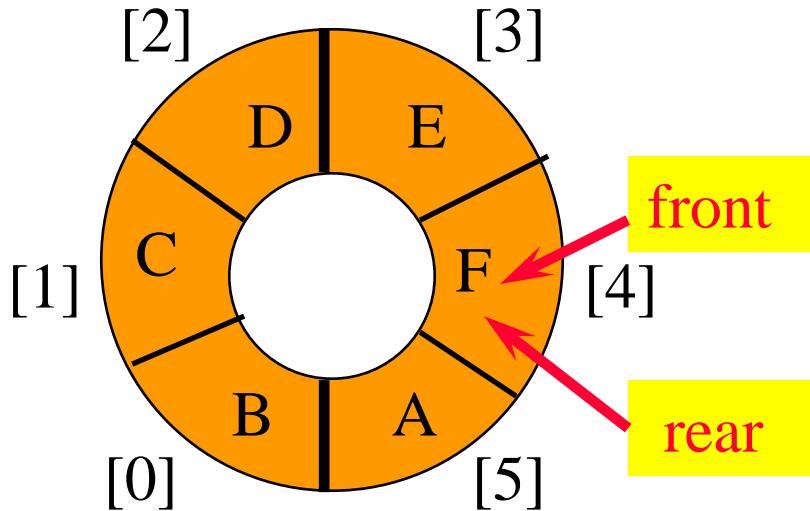
# Дарааллын савыг дүүргэх



# Дарааллын савыг дүүргэх



# Дарааллын савыг дүүргэх



- Дараалсан нэмэлтүүдээр **front = rear** болоход сав дүүрнэ
- Сав дүүрэн үү, хоосон уу гэдгийг ялгах хэзүү боллоо!

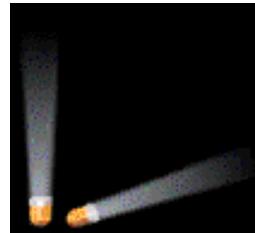
# Анхаар!!!!

- Засвар.
  - Дарааллыг бүү дүүргэ.
    - Нэмэгдэх элемент дарааллыг дүүргэх бол массивын хэмжээг нэмэгдүүл.
    - Үүнийг сурах бичигт үзүүлсэн.
  - Булын хувьсагч `lastOperationIsPut` -г ашигла
    - `put` үйлдэл бүрийн дараа `true` болго
    - `remove` үйлдэл бүрийн дараа `false` болго
    - дараалал хоосон -> `(front == rear) && !lastOperationIsPut`
    - Дараалал дүүрэн -> `(front == rear) && lastOperationIsPut`

# Анхаар!!!!

- Засвар (Үргэлжлэл).
  - Бүхэл хувьсагч `size` -г ашигла
    - `put` үйлдэл бүрийн дараа `size++`.
    - `remove` үйлдэл бүрийн дараа `size--`.
    - Дараалал хоосон -> (`size == 0`)
    - Дараалал дүүрэн -> (`size == queue.length`)
  - Эхний хувилбараар чанарын үзүүлэлт арай дээр байх болно.

# Толь бичиг



- Хосуудын цуглуулга.
  - (key, element)
  - Хос бүр өөр түлхүүртэй.
- Үйлдлүүд.
  - `get(theKey)`
  - `put(theKey, theElement)`
  - `remove(theKey)`

# Хэрэглээ

- CS203 –г сонгосон оюутнууд.
  - **(key, element) =** (оюутны нэр, бие даалт болон шалгалтын дүнгийн шугаман жагсаалт)
  - Бүх түлхүүр ялгаатай.
- **Ө.Дөлгөөн** гэсэн түлхүүртэй элементийг авах
- **Д.Туяа** гэсэн түлхүүртэй элементийг өөрчлөх
  - **put()**.
  - **remove()**.

# Давхцалттай толь бичиг

- Түлхүүр давхцаж болно.
- Үгсийн толь бичиг.
  - Хос нь (үг, утга).
  - Нэг үг хэд хэдэн утгатай байж болно.
    - (гар, хүний эрхтэн)
    - (гар, гадаглах хөдөлгөөн)
    - (гар, компьютерийн оруулах төхөөрмж)
    - ГЭХ МЭТ.

# Шугаман жагсаалтаар дүрслэх

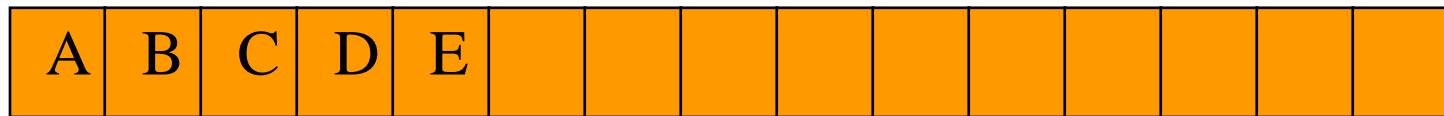
- $L = (e_0, e_1, e_2, e_3, \dots, e_{n-1})$
- $e_i$  бүхэн (key, element).
- 5-хостой толь бичиг  $D = (a, b, c, d, e)$ .
  - $a = (aKey, aElement), b = (bKey, bElement),$   
Г.М.
- Массив эсхүл Холбоост дүрслэл.

# Массив дурслэл



- get(theKey)
- put(theKey, theElement)
- remove(theKey)

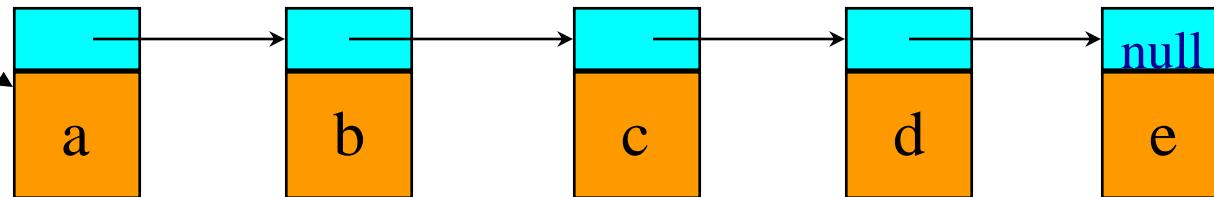
# Эрэмбэлэгдсэн массив



- Элементүүд түлхүүрийн өсөх дарааллаар байрлана.
- `get(theKey)`
  - $O(\log \text{size})$
- `put(theKey, theElement)`
  - $O(\log \text{size})$  давхцлыг шалгахад,  $O(\text{size})$  нэмэхэд.
- `remove(theKey)`
  - $O(\text{size})$ .

# Эрэмбэлэгдээгүй гинж

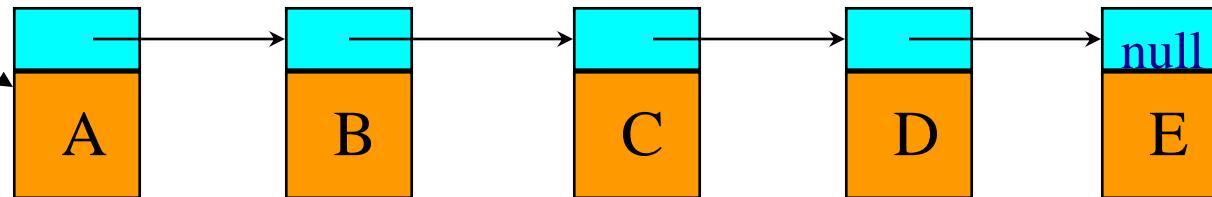
firstNode



- `get(theKey)`
  - $O(\text{size})$
- `put(theKey, theElement)`
  - $O(\text{size})$  давхцлыг шалгахад,  $O(1)$  зүүн талд нь нэмэхэд.
- `remove(theKey)`
  - $O(\text{size})$ .

# Эрэмбэлэгдсэн гинж

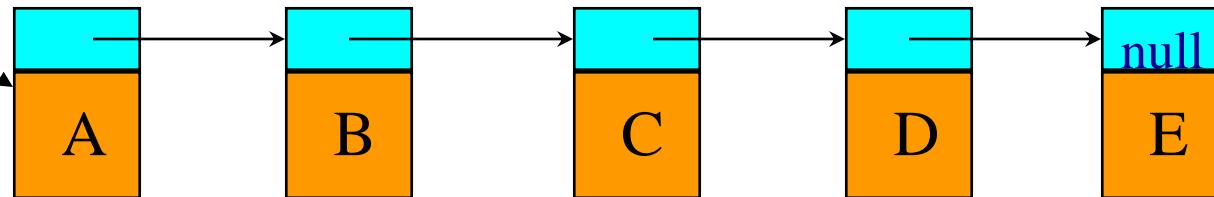
firstNode



- Элементүүд түлхүүрийн өсөх дараалалтай.
- `get(theKey)`
  - $O(\text{size})$
- `put(theKey, theElement)`
  - $O(\text{size})$  давхцлыг шалгахад,  $O(1)$  зөв газар нь нэмэхэд.

# Эрэмбэлэгдсэн гинж

firstNode



- Элементүүд түлхүүрийн өсөх дараалалтай.
- `remove(theKey)`
  - $O(\text{size})$ .

# Алгасах жагсаалт

- Муу тохиолдолд **get, put, remove** -  $O(\text{size})$ .
- Дундаж хугацаа -  $O(\log \text{size})$ .
- Skip lists – алгасах жагсаалтыг бид алгасна

# ХЭШ ХҮСНЭГТ

- Муу тохиолдолд **get, put, remove** -  $O(\text{size})$ .
- Дундаж хугацаа  $O(1)$ .

# ТӨГС ХЭШ

- 1D массив(хүснэгт) ашиглая `table[0:b-1]`.
  - Массивын байршил бүр **багц**.
  - Ер нь багц толь бичгийн зөвхөн нэг хосыг хадгалах ёстой.
- Ашиглах хэш функц  $f$  нь түлхүүр  $k - g [0, b-1]$  завсар дахь хүснэгтийн индекст хувиргах ёстой.
  - $f(k)$  бол түлхүүр  $k$ -ийн **багцны үүр**
- Толь бичгийн (**key, element**) хос бүр `table[f[key]]` гэсэн багцны үүрэнд хадгалагдана

# Төгс хэшийн жишээ

- Хосууд: (22,a), (33,c), (3,d), (73,e), (85,f).
- Хэш хүснэгт `table[0:7]`,  $b = 8$ .
- Хэш функц `key/11`.
- Хосууд хүснэгтэд дараах байдлаар хадгалагдана:

|       |     |        |        |     |     |        |        |
|-------|-----|--------|--------|-----|-----|--------|--------|
| (3,d) |     | (22,a) | (33,c) |     |     | (73,e) | (85,f) |
| [0]   | [1] | [2]    | [3]    | [4] | [5] | [6]    | [7]    |

- `get, put, remove` - ажиллах хугацаа  $O(1)$ .

# Яавал буруу тийш явах вэ?

|       |  |        |        |  |  |        |        |
|-------|--|--------|--------|--|--|--------|--------|
| (3,d) |  | (22,a) | (33,c) |  |  | (73,e) | (85,f) |
|-------|--|--------|--------|--|--|--------|--------|

[0] [1] [2] [3] [4] [5] [6] [7]

- **(26,g)** хаашаа явах вэ?
- Нэг багцны үүртэй түлхүүрүүдийг **синоним** гэнэ
  - 22 ба 26 бол ашиглаж байгаа хэш функцийн хувьд синонимууд.
- **(26,g)** –д харгалзах багцны үүр эзлэгдсэн байна.

# Яавал буруу тийш явах вэ?

|       |  |        |        |  |  |        |        |
|-------|--|--------|--------|--|--|--------|--------|
| (3,d) |  | (22,a) | (33,c) |  |  | (73,e) | (85,f) |
|-------|--|--------|--------|--|--|--------|--------|

- Θөр түлхүүртэй шинэ хосод харгалзах багцны үүр эзлэгдсэнээс **collision-зөрчил** үүснэ.
- Шинэ хосод харгалзах багцны үүрэнд зайд байхгүйгээс **overflow-халилт** үүснэ.
- Багцны үүр зөвхөн нэг хосыг хадгалдаг бол зөрчил болон халилт зэрэг үүснэ.
- Халилтыг шийдэх арга хэрэгтэй.

# Хэш хүснэгтийн асуудлууд

- Хэш функцийг сонгох.
- Халилтыг зохицуулах арга.
- Хэш хүснэгтийн хэмжээ (Багцны тоо).

# Хэш функцууд

- Хоёр хэсэг:
  - Түлхүүр бүхэл биш бол бүхэл болгох.
    - Шийдэх арга нь `hashCode()`.
  - Бүхэл тоог багцны үүрт буулгах.
    - $f(k)$  функц  $[0, b-1]$  межид бүхэл утгатай, үүнд  $b$  бол хүснэгт дэх багцны тоо.

# Тэмдэгт мөрийг бүхэл тоо руу хувиргах

- Java –ийн тэмдэгт бүр **2** байт уртай.
- int бол **4** байт.
- **2** тэмдэгтэй **s** тэмдэгт мөрийг давтагдашгүй **4** байт int –д хувиргахдаа:

```
int answer = s.charAt(0);  
  
answer = (answer << 16) + s.charAt(1);
```
- **2** тэмдэгтээс урт тэмдэгт мөрөнд давтагдашгүй **int** дүрслэл байхгүй.

# ТЭМДЭГТ МӨРИЙГ СӨРӨГ БИШ БҮХЭЛ ТОО РУУ ХУВИРГАХ

```
public static int integer(String s)
```

```
{
```

```
    int length = s.length();
```

```
        // s -ийн тэмдэгтийн тоо
```

```
    int answer = 0;
```

```
    if (length % 2 == 1)
```

```
    { // урт нь сондгой бол
```

```
        answer = s.charAt(length - 1);
```

```
        length--;
```

```
}
```

# Тэмдэгт мөрийг сөрөг биш бүхэл тоо руу хувиргах

```
// урт нь тэгш бол
for (int i = 0; i < length; i += 2)
{ // нэг удаа 2 тэмдэгтийг
    answer += s.charAt(i);
    answer += ((int) s.charAt(i + 1)) << 16;
}
return (answer < 0) ? -answer : answer;
```

# Багцны үүрийн буулгалт

|       |  |        |        |  |  |        |        |
|-------|--|--------|--------|--|--|--------|--------|
| (3,d) |  | (22,a) | (33,c) |  |  | (73,e) | (85,f) |
|-------|--|--------|--------|--|--|--------|--------|

[0] [1] [2] [3] [4] [5] [6] [7]

- Хамгийн нийтлэг арга бол хуваалт(**divisor**).

homeBucket =

`Math.abs(theKey.hashCode()) % divisor;`

- divisor** багцны тоо **b** -тэй тэнцүү
- $0 \leq \text{homeBucket} < \text{divisor} = b$

# Жигд Хэш функц

|       |  |        |        |  |  |        |        |
|-------|--|--------|--------|--|--|--------|--------|
| (3,d) |  | (22,a) | (33,c) |  |  | (73,e) | (85,f) |
|-------|--|--------|--------|--|--|--------|--------|

[0] [1] [2] [3] [4] [5] [6] [7]

- **keySpace** бол байж болох түлхүүрийн олонлог болог.
- Хэш функц **keySpace** олонлогийн түлхүүрийг багцад тусгахдаа дунджаар ижил тооны түлхүүр нэг багцад тусч байвал функцийг **жигд хэш функц** гэдэг

# Жигд Хэш функц

|       |  |        |        |  |  |        |        |
|-------|--|--------|--------|--|--|--------|--------|
| (3,d) |  | (22,a) | (33,c) |  |  | (73,e) | (85,f) |
|-------|--|--------|--------|--|--|--------|--------|

[0] [1] [2] [3] [4] [5] [6] [7]

- Θөрөөр хэлбэл, санамсаргүй сонгосон түлхүүр багц  $i$  –д тусах магадлал  $1/b, 0 \leq i < b$ .
- Жигд хэш функц түлхүүр санамсаргүй сонгогдох тохиолдолд халилтыг минимум болгодог.

# Хуваалтын хэш

- **keySpace** = бүх **int**-үүд
- Ямар ч **b**-ийн хувьд дунджаар  $2^{32}/b$  тооны **int**-үүд багц **i** –д тусдаг (хуваагддаг).
- Иймд, **keySpace** = бүх **int**-үүд бол хуваалтын арга нэгэн жигд хэш функц болно
- Амьдралд, түлхүүрүүд хамааралтай байдаг.
- Тэгэхээр, хуваагч **b** –ийн сонголт багцын үүрийн буулгалтанд нөлөөлдөг.

# Хуваагчийг сонгох

- Түлхүүрүүд ер нь хамааралтай байдгаас тэгш, сондгой багцны үүрт буулгалт давамгайлах тал байдаг.
- Хуваагч тэгш тоо бол, сондгой бүхэл сондгой багцад, тэгш бүхэл тэгш багцад тусдаг.
  - $20\% 14 = 6$ ,  $30\% 14 = 2$ ,  $8\% 14 = 8$
  - $15\% 14 = 1$ ,  $3\% 14 = 3$ ,  $23\% 14 = 9$

# Хуваагчийг сонгох

- Хуваагч сондгой бол, сондгой (тэгш) бүхэл дурын үүрд хуваагдаж болдог.
  - $20\% 15 = 5$ ,  $30\% 15 = 0$ ,  $8\% 15 = 8$
  - $15\% 15 = 0$ ,  $3\% 15 = 3$ ,  $23\% 15 = 8$
- Багцны үүрт нэгэн жигд тараах илүү боломжтой.
- Иймд тэгш хуваагчийг битгий ашигла.

# Хуваагчийг сонгох

- Амьдралд жигд биш тархалт хуваагчийг **3, 5, 7, ...** мэтийн анхны тооны үржвэр байдлаар сонгосноос болдог
- Хэрвээ **p** нь **b**-ийн хуваагч бол **p** өсөх тутам энэ нөлөөлөл багасдаг.
- Зөв сонголтоор **b** анхны тоо байх нь чухал.
- Эсхүл, **b-g** сонгоходоо **20**-оос доош тоонд хуваагддаггүй байх явдал

# Java.util.HashTable



- Сондгой тоог хуваагч болгон ашигладаг.
- Ингэснээр олон хосыг толь бичигт оруулах зорилгоор хэш хүснэгтийн хэмжээг өөрчлөх боломж олгодог.
  - Жишээ нь, массивыг 2 дахин ихэсгэхэд, **b** (сондгой) урттай 1D массив **table** –ийн уртыг **2b+1** болгоно(мөн сондгой).

# Халилтыг зохицуулах

- Шинэ хос (**key, element**) -ийн хувьд багцын үүр дүүрэн бол халилт үүсдэг.
- Халилтыг зохицуулахдаа:
  - Хэш хүснэгтээс голдуу дүүрэн бус байдаг багцыг хай.
    - Шугаман тандалт.
    - Квадрат тандалт.
    - Санамсаргүй тандалт.
  - Багц бүрт ижил үүртэй бүх хосуудын жагсаалтыг хадгалах замаар халилтаас зайлсхийж болно.
    - Массив шугаман жагсаалт.
    - Гинж.

# Шугаман тандалт – Get ба Put

- divisor = b (багцын тоо) = 17.
- Багцын үүр = key % 17.

| 0  | 4 | 8  | 12 | 16 |
|----|---|----|----|----|
| 34 | 0 | 45 | 28 | 12 |

- 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45 гэсэн түлхүүртэй хосуудыг хийлгээ

# Шугаман тандалт – Remove

| 0  | 4 | 8  | 12 | 16 |
|----|---|----|----|----|
| 34 | 0 | 45 |    | 28 |

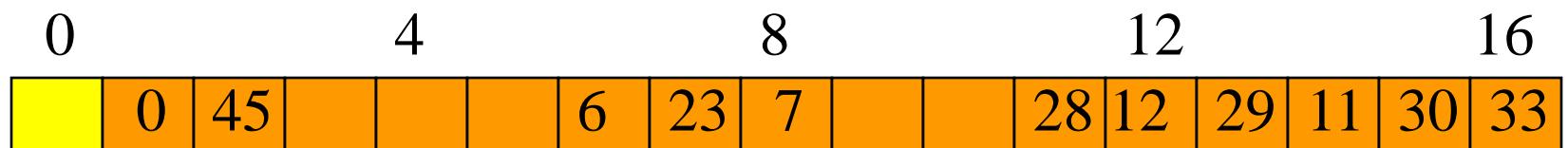
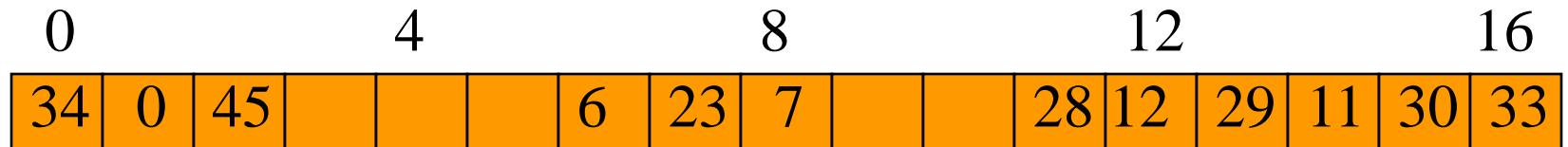
- remove(0)

| 0  | 4 | 8  | 12 | 16 |
|----|---|----|----|----|
| 34 |   | 45 |    | 28 |

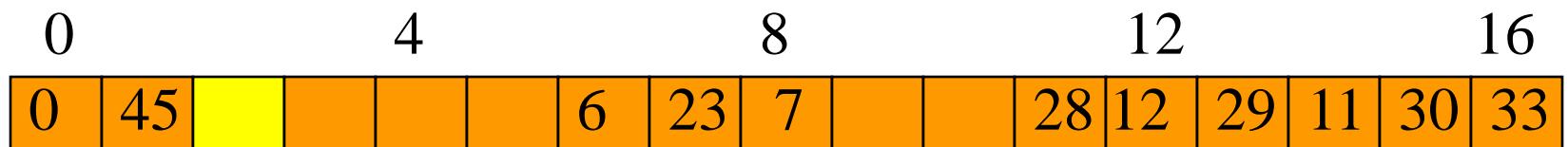
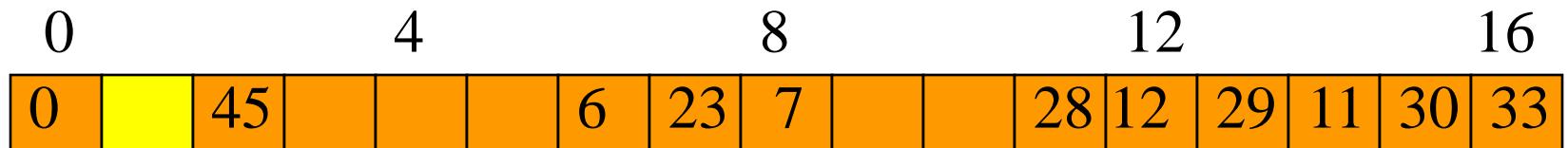
- Чөлөөлөгдсөн багцыг ашиглаж болох хосыг хайх.

| 0  | 4  | 8 | 12 | 16 |
|----|----|---|----|----|
| 34 | 45 |   | 28 | 33 |

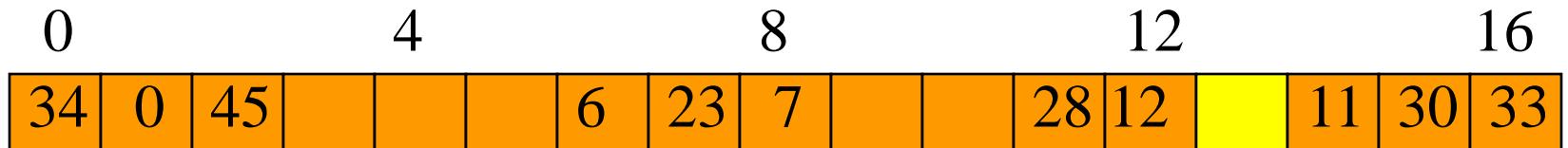
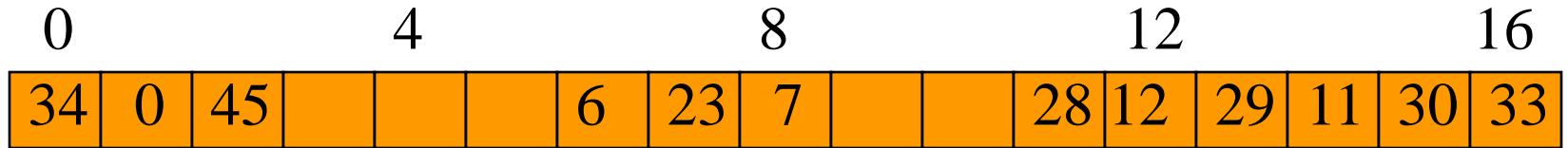
# Шугаман тандалт – remove(34)



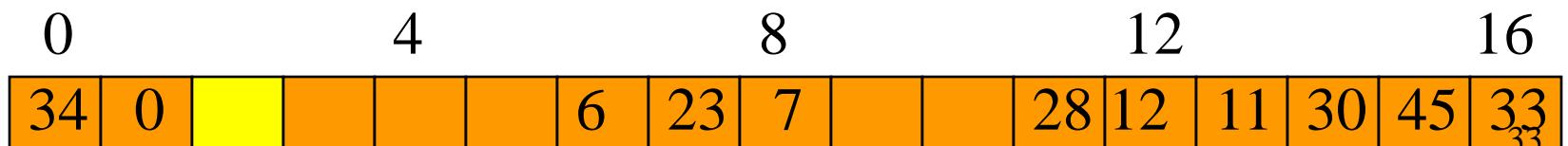
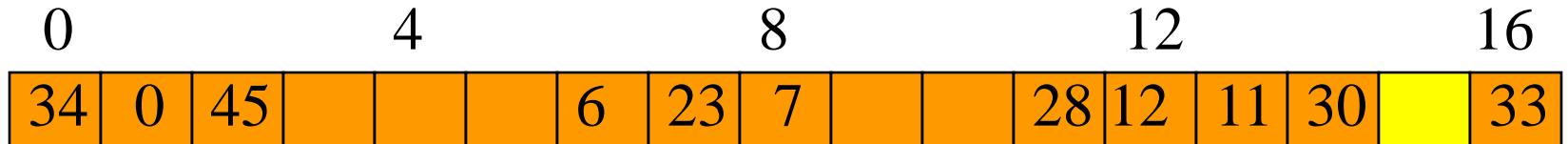
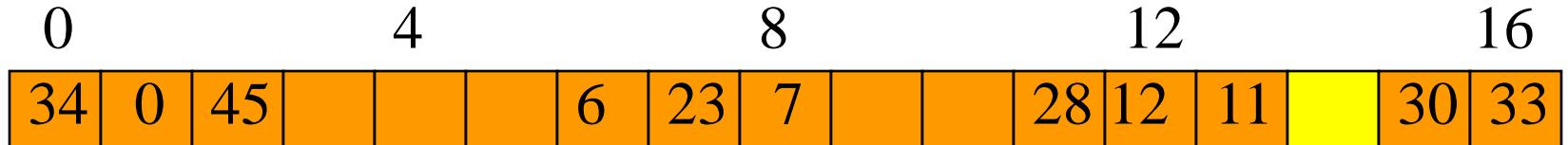
- Чөлөөлөгдсөн багцыг ашиглаж болох хосыг хайх.



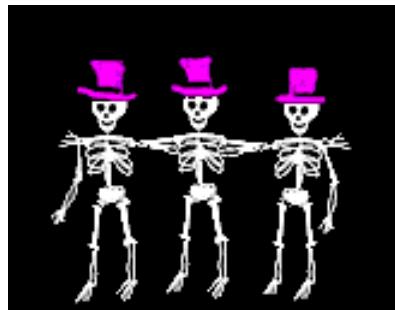
# Шугаман тандалт – remove(29)



- Чөлөөлөгдсөн багцыг ашиглаж болох хосыг хайх.



# Шугаман тандалтын ҮЗҮҮЛЭЛТ



| 0  | 4 | 8  | 12 | 16                |
|----|---|----|----|-------------------|
| 34 | 0 | 45 |    | 28 12 29 11 30 33 |

- get/put/remove үйлдлийн муу тохиолдлын хугацаа  $\Theta(n)$ , Үүнд **n** – хүснэгт дэх хосын тоо.
- Бүх хос нэг ҮҮРТ ороход ҮҮСНЭ.

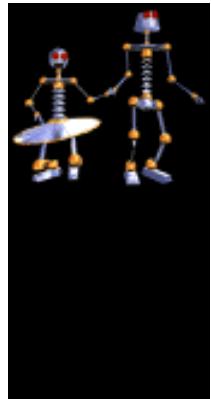


# Дундаж ҮЗҮҮЛЭЛТ

| 0  | 4 | 8  | 12 | 16 |
|----|---|----|----|----|
| 34 | 0 | 45 | 6  | 23 |

- $\alpha = \text{ачааллын нягтрал} = (\text{хосын тоо})/b$ .
  - $\alpha = 12/17$ .
- $S_n = \text{амжилттай хайлтаар шалгах багцын (дундаж) тоо}$  ( $n$  - том тоо бол)
- $U_n = \text{амжилтгүй хайлтаар шалгах багцын (дундаж) тоо}$  ( $n$  - том тоо бол)
- тэгвэл `put`, `remove` үйлдлүүдийн хугацаа  $U_n$ -аар тодорхойлогдоно

# Дундаж ҮЗҮҮЛЭЛТ



- $S_n \sim 1/2(1 + 1/(1 - \alpha))$
- $U_n \sim 1/2(1 + 1/(1 - \alpha)^2)$
- $0 <= \alpha <= 1.$

| $\alpha$    | $S_n$ | $U_n$ |
|-------------|-------|-------|
| <b>0.50</b> | 1.5   | 2.5   |
| <b>0.75</b> | 2.5   | 8.5   |
| <b>0.90</b> | 5.5   | 50.5  |

$\alpha <= 0.75$  байхыг  
зөвлөдөг.

# Хэш хүснэгтийн зохиомж

- Өгөгдсөн шаардлагаас хамаарч, ачааллын нягтралын зөвшөөрөгдөх дээд хэмжээг тогтоох.
- Амжилттай хайлт хийхэд 10 –с илүүгүй харьцуулалт хэрэгтэй бол (шаардлага).
  - $S_n \sim \frac{1}{2}(1 + 1/(1 - \alpha))$
  - $\alpha <= 18/19$
- Амжилтгүй хайлт хийхэд 10 –с илүүгүй харьцуулалт хэрэгтэй бол (шаардлага).
  - $U_n \sim \frac{1}{2}(1 + 1/(1 - \alpha)^2)$
  - $\alpha <= 4/5$
- Иймд  $\alpha <= \min\{18/19, 4/5\} = 4/5$ .

# Хэш хүснэгтийн зохиомж

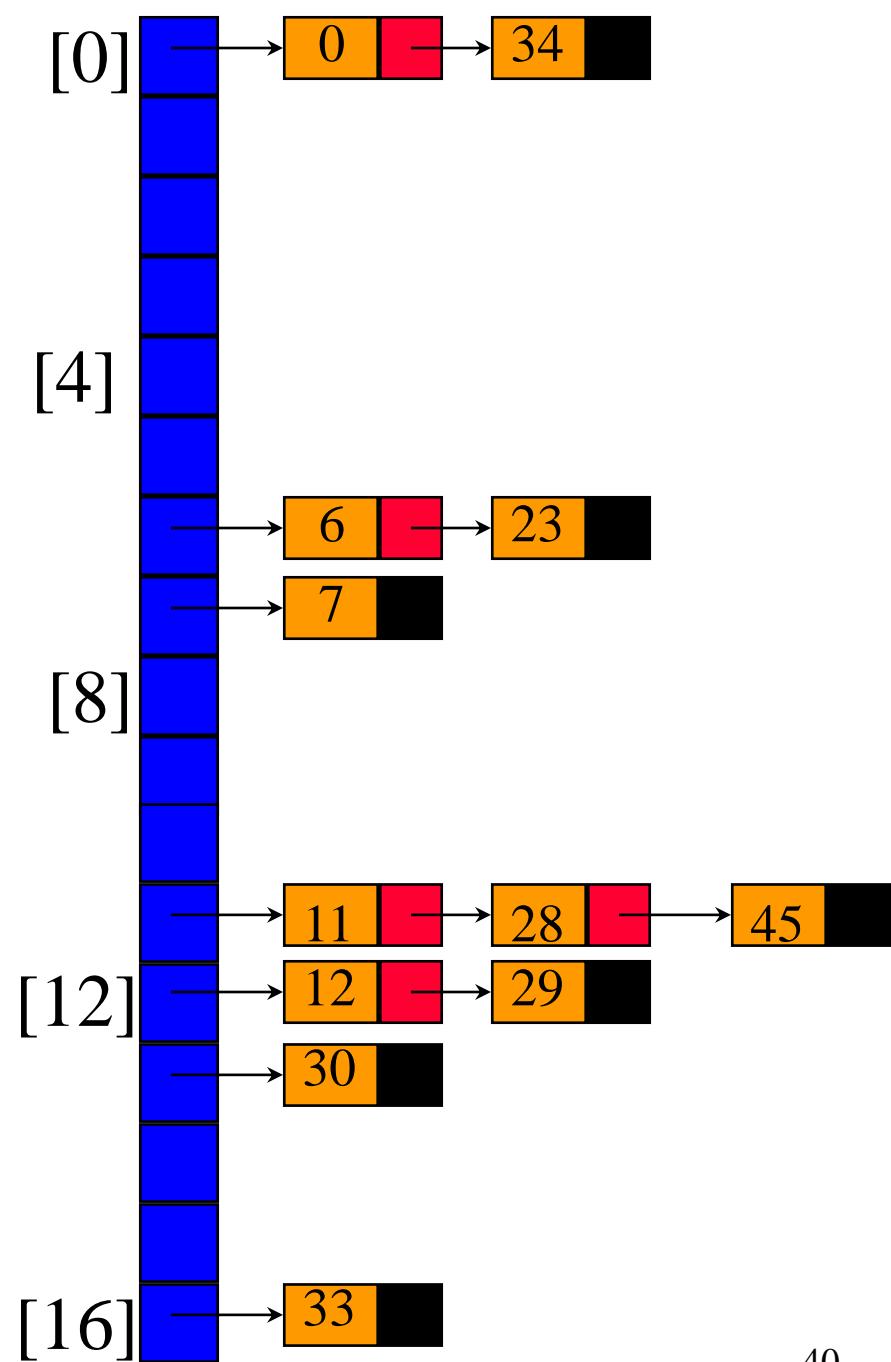
- Динамик хүснэгтийн хэмжээ.
  - Ачааллын нягтрал хүссэн хэмжээнээс хэтэрвэл (**4/5** манай жишээнд), хэш хүснэгтийн хэмжээг одоогийнхоос нь ойролцоогоор 2 дахин нэмэгдүүлнэ.
- Тогтсон хүснэгтийн хэмжээ.
  - Хосын максимум тоог мэднэ.
  - **1000** -с илүүгүй хостой.
  - Ачааллын нягтрал  $\leq 4/5 \Rightarrow b \geq 5/4 * 1000 = 1250$ .
  - $b$  (ө.х. **divisor**) **20** –с дооши анхны тоонд хуваагддаггүй сондгой тоо, эсхүл анхны тоо байхаар сонго.

# Синонимын шугаман жагсаалт

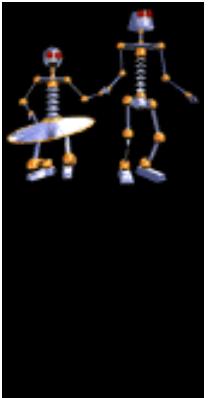
- Багц бүрт ижил үүртэй хосуудын шугаман жагсаалтыг халдгалах.
- Шугаман жагсаалт эрэмбэлэгдсэн байж болно.
- Шугаман жагсаалт нь массив эсхүл гинж байж болно.

# Эрэмбэлэгдсэн гинж

- 6, 12, 34, 29,  
28, 11, 23, 7, 0,  
33, 30, 45  
түлхүүртэй  
хосуудыг  
хийх
- Багцны  $\text{typ} = \text{key \% } 17$ .



# Дундаж ҮЗҮҮЛЭЛТ



- $\alpha \geq 0$  байна.
- Гинжний дундаж урт  $\alpha$ .
- $S_n \sim 1 + \alpha / 2$ .
- $U_n \leq \alpha$ ,  $\alpha < 1$  бол
- $U_n \sim 1 + \alpha / 2$ ,  $\alpha \geq 1$  бол

# java.util.Hashtable



- Эрэмбэлээгүй гинж.
- Анхдагч хуваагч  $b = \text{divisor} = 101$
- Анхдагч  $\alpha \leq 0.75$
- Ачааллын нягтрал зөвшөөрөгдсөн максимум нягтрааас давбал хэш хүснэгтийн шинэ урт  $\text{newB} = 2b+1$ .

# Өгөгдлийг шахах



- Өгөгдлийн хэмжээг багасгах.
  - Санах ойг багасгаж, ингэснээр санах ойн зардлыг багасгана.
    - Шахалтын зэрэг = анхны хэмжээ/шахагдсан хэмжээ
  - Өгөгдөл дамжуулах, хүлээн авах хугацааг багасгана.

# Хаягдалгүй, хаягдалтай шахалт

- `compressedData = compress(originalData)`
- `decompressedData = decompress(compressedData)`
- `originalData = decompressedData`, бол хаягдалгүй шахалт
- `originalData != decompressedData`, бол хаягдалтай шахалт

# Хаягдалгүй, хаягдалтай шахалт

- Хаягдалтай шахагч, хаягдалгүй шахагчаас илүү өндөр шахалтын зэрэгтэй
  - магадгүй 100 : 2.
- Хаягдалгүй шахалтыг жишээ нь текст файлын хэрэглээнд ашигладаг.
- Хаягдалтай шахалтыг дурсний хэрэглээнд өргөн хэрэглэдэг.
  - Видео дамжуулахад бага зэргийн хаягдлыг хүний нүд ялгадаггүй.



## Текст шахалт



- Хаягдалгүй шахах нь чухал.
- Түгээмэл тархсан **zip** болон Unix-н **compress** шахагч LZW(1984) (Lempel-Ziv-Welch) аргыг ашигладаг.



## LZW шахалт



- Эх текстэд орсон тэмдэгтийн цувааг динамик байдлаар тодорхойлогддог кодоор сольдог.
- Кодын хүснэгтийг шахагдсан текстэд кодчилодоггүй. Учир нь буцааж задлахад хэрэг болдог.



# LZW шахалт



- Текстийн тэмдэгтэд хязгаарлалт хийе  $\{a, b\}$ .
  - Амьдралд цагаан толгой ASCII олонлогийн **256** тэмдэгтэй.
- Цагаан толгойн тэмдэгтүүдэд олгох кодыг **0** -ээс эхлэн дугаарлай
- Аххны кодын хүснэгт:

|      |   |   |
|------|---|---|
| code | 0 | 1 |
| key  | a | b |



# LZW шахалт



|      |   |   |
|------|---|---|
| code | 0 | 1 |
| key  | a | b |

- Эх текст = **abababbabaabbabbaabba**
- Эх текстийг зүүнээс баруун тийш гүйлгэх замаар шахья.
- Кодын хүснэгтэд код нь орсон хамгийн урт угтвар **p** –г хайна.
- **p** –г түүний код **pCode** –оор дүрсэлж, дараачийн боломжтой кодыг **pc** -д онооно. Үүнд **c** бол шахах текстийн дараачийн тэмдэгт.



# LZW шахалт



|      |   |   |    |
|------|---|---|----|
| code | 0 | 1 | 2  |
| key  | a | b | ab |

- Эх текст = abababbabaabbabbaabba
- $p = a$
- $pCode = 0$
- $c = b$
- $a$  -г 0 -ээр дүрслээд  $ab$  –г кодын хүснэгтэд нэмнэ.
- Шахсан текст = 0



# LZW шахалт



|      |   |   |    |    |
|------|---|---|----|----|
| code | 0 | 1 | 2  | 3  |
| key  | a | b | ab | ba |

- Эх текст = ababbabbabaabbabbaabba
- Шахсан текст = 0
- $p = b$
- $pCode = 1$
- $c = a$
- $b$  -г 1 а-ээр дүслээд **ba** –г кодын хүснэгтэд нэмнэ.
- Шахсан текст = 01



# LZW шахалт



|      |   |   |    |    |     |
|------|---|---|----|----|-----|
| code | 0 | 1 | 2  | 3  | 4   |
| key  | a | b | ab | ba | aba |

- Эх текст = ababbabbabaabbabbabaabba
- Шахсан текст = 01
- $p = ab$
- $pCode = 2$
- $c = a$
- $ab$  -г 2 -оор дүслээд **aba** кодын хүснэгтэд нэмнэ.
- Шахсан текст = 012



# LZW шахалт



| code | 0 | 1 | 2  | 3  | 4   | 5   |
|------|---|---|----|----|-----|-----|
| key  | a | b | ab | ba | aba | abb |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012
- $p = ab$
- $pCode = 2$
- $c = b$
- $ab$  -г 2 -оор дүрслээд  $abb$  –г кодын хүснэгтэд нэмнэ.
- Шахсан текст = 0122



# LZW шахалт



|      |   |   |    |    |     |     |     |
|------|---|---|----|----|-----|-----|-----|
| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   |
| key  | a | b | ab | ba | aba | abb | bab |

- Эх текст = ababab**babaabbabbabbaabba**
- Шахсан текст = 0122
- $p = ba$
- $pCode = 3$
- $c = b$
- **ba** -г 3 –аар дүрслээд **bab** –г кодын хүснэгтэд нэмнэ.
- Шахсан текст = 01223



# LZW шахалт



|      |   |   |    |    |     |     |     |     |
|------|---|---|----|----|-----|-----|-----|-----|
| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   |
| key  | a | b | ab | ba | aba | abb | bab | baa |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 01223
- $p = ba$
- $pCode = 3$
- $c = a$
- $ba$  -г 3 -аар дүрслээд  $baa$  –г кодын хүснэгтэд нэмнэ.
- Шахсан текст = 012233



# LZW шахалт



|      |   |   |    |    |     |     |     |     |      |
|------|---|---|----|----|-----|-----|-----|-----|------|
| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   | 8    |
| key  | a | b | ab | ba | aba | abb | bab | baa | abba |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233
- $p = abb$
- $pCode = 5$
- $c = a$
- $abb$  -г 5 -аар дүрслээд  $abba$  –г кодын хүснэгтэд нэмнэ.
- Шахсан текст = 0122335



# LZW шахалт



|      |   |   |    |    |     |     |     |     |      |        |
|------|---|---|----|----|-----|-----|-----|-----|------|--------|
| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   | 8    | 9      |
| key  | a | b | ab | ba | aba | abb | bab | baa | abba | abbaaa |

- Эх текст = abababbabaabb**abbaabba**
- Шахсан текст = 0122335
- $p = abba$
- $pCode = 8$
- $c = a$
- **abba** -г 8 -аар дүрслээд **abbaa** –г кодын хүснэгтэд нэмнэ.
- Шахсан текст = 01223358



# LZW шахалт



| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   | 8    | 9      |
|------|---|---|----|----|-----|-----|-----|-----|------|--------|
| key  | a | b | ab | ba | aba | abb | bab | baa | abba | abbaaa |

- Эх текст = abababbabaabbabba**abba**
- Шахсан текст = 01223358
- p = **abba**
- pCode = 8
- c = null
- **abba** -г 8 –аар дүрсэлнэ
- Шахсан текст = 012233588

# Кодын хүснэгтийн дурслэл

|      |   |   |    |    |     |     |     |     |      |       |
|------|---|---|----|----|-----|-----|-----|-----|------|-------|
| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   | 8    | 9     |
| key  | a | b | ab | ba | aba | abb | bab | baa | abba | abbaa |

- Толь бичиг.
  - Хосууд (`key, element`) = (`key,code`).
  - Үйлдлүүд : `get(key)` , `put(key, code)`
- Кодын хязгаар  $2^{12}$ .
- Хэш хүснэгт ашиглах.
  - Хувьсах урттай түлхүүрийг ижил ижил урттай болгох.
  - Түлхүүр бүр `pc` хэлбэртэй. Үүнд: `r` тэмдэгт мөр бол хүснэгтэд өмнө нь байгаа түлхүүр.
  - `pc` -г (`pCode`)`c` -оор сольно



# Кодын хүснэгтийн дурслэл



|      |   |   |    |    |     |     |     |     |           |   |
|------|---|---|----|----|-----|-----|-----|-----|-----------|---|
| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   | 8         | 9 |
| key  | a | b | ab | ba | aba | abb | bab | baa | abbaabbaa |   |

|      |   |   |    |    |    |    |    |    |    |    |
|------|---|---|----|----|----|----|----|----|----|----|
| code | 0 | 1 | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
| key  | a | b | 0b | 1a | 2a | 2b | 3b | 3a | 5a | 8a |

## LZW задлалт

|      |   |   |
|------|---|---|
| code | 0 | 1 |
| key  | a | b |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233588
- Кодыг зүүнээс баруун тийш текстэд хөрвүүлнэ
- 0 бол a.
- Задалсан текст = a
- pCode = 0 , p = a.
- p = a -ийн араас дараачийн тэмдэгтийг кодын хүснэгтэд оруулна.

## LZW задлалт

|      |   |   |    |
|------|---|---|----|
| code | 0 | 1 | 2  |
| key  | a | b | ab |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233588
- 1 бол b.
- Задалсан текст = ab
- pCode = 1 , p = b.
- lastP = a –ийн араас залгасан p –н эхний тэмдэгтийг кодын хүснэгтэд оруулна.

## LZW задлалт

|      |   |   |    |    |
|------|---|---|----|----|
| code | 0 | 1 | 2  | 3  |
| key  | a | b | ab | ba |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233588
- 2 бол ab.
- Задалман текст = abab
- pCode = 2 , p = ab.
- lastP = b –ийн араас залгасан p –н эхний тэмдэгтийг кодын хүснэгтэд оруулна.

## LZW задлалт

|      |   |   |    |    |     |
|------|---|---|----|----|-----|
| code | 0 | 1 | 2  | 3  | 4   |
| key  | a | b | ab | ba | aba |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233588
- 2 бол ab
- Задалсан текст = ababab.
- pCode = 2 , p = ab.
- lastP = ab –ийн араас залгасан p –н эхний тэмдэгтийг кодын хүснэгтэд оруулна.

## LZW задлалт

| code | 0 | 1 | 2  | 3  | 4   | 5   |
|------|---|---|----|----|-----|-----|
| key  | a | b | ab | ba | aba | abb |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233588
- 3 бол ba
- Задалсан текст = abababba.
- pCode = 3 , p = ba.
- lastP = ab –ийн араас залгасан p –н эхний тэмдэгтийг кодын хүснэгтэд оруулна.

## LZW задалт

| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   |
|------|---|---|----|----|-----|-----|-----|
| key  | a | b | ab | ba | aba | abb | bab |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233588
- 3 бол ba
- Задалсан текст = abababbaba.
- pCode = 3 , p = ba.
- lastP = ba –ийн араас залгасан p –н эхний тэмдэгтийг кодын хүснэгтэдоруулна.

## LZW задлалт

|      |   |   |    |    |     |     |     |     |
|------|---|---|----|----|-----|-----|-----|-----|
| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   |
| key  | a | b | ab | ba | aba | abb | bab | baa |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233588
- 5 бол abb
- Задалсан текст = abababbabaabb.
- pCode = 5 , p = abb.
- lastP = ba –ийн араас залгасан p –н эхний тэмдэгтийг кодын хүснэгтэд оруулна.

## LZW задлалт

| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   | 8    |
|------|---|---|----|----|-----|-----|-----|-----|------|
| key  | a | b | ab | ba | aba | abb | bab | baa | abba |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233588
- 8 бол ???
- Код хүснэгтэд байхгүй бол, түүний түлхүүр нь lastP –ий араас залгасан lastP-ий эхний тэмдэгт байх болно
- lastP = abb
- Тэгэхээр 8 бол abba.



## LZW задалт

| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   | 8    | 9     |
|------|---|---|----|----|-----|-----|-----|-----|------|-------|
| key  | a | b | ab | ba | aba | abb | bab | baa | abba | abbaa |

- Эх текст = abababbabaabbabbaabba
- Шахсан текст = 012233588
- 8 бол abba
- Задалсан текст = abababbabaabbabbaabba.
- pCode = 8 , p = abba.
- lastP = abba –ийн араас орсон p –ийн эхний тэмдэгт кодын хүснэгтэд орно.

# Кодын хүснэгтийн дурслэл

|      |   |   |    |    |     |     |     |     |           |   |
|------|---|---|----|----|-----|-----|-----|-----|-----------|---|
| code | 0 | 1 | 2  | 3  | 4   | 5   | 6   | 7   | 8         | 9 |
| key  | a | b | ab | ba | aba | abb | bab | baa | abbaabbaa |   |

- Толь бичиг.
  - Хосууд (*key, element*) = (*code, what the code represents*) = (*code, codeKey*).
  - Үйлдлүүд : *get(key)* , *put(key, code)*
- Түлхүүрүүд бүхэл тоо **0, 1, 2, ...**
- 1D **codeTable** массивыг ашиглай
  - *codeTable[code] = codeKey*.
  - Кодын түлхүүр бүр **pc** хэлбэртэй. Үүнд **p** тэмдэгт мөр бол хүснэгтэд өмнө нь орсон кодын түлхүүр.
  - **pc** -г (**pCode**)**c** -аар сольно

# Хугацааны үзүүлэлт



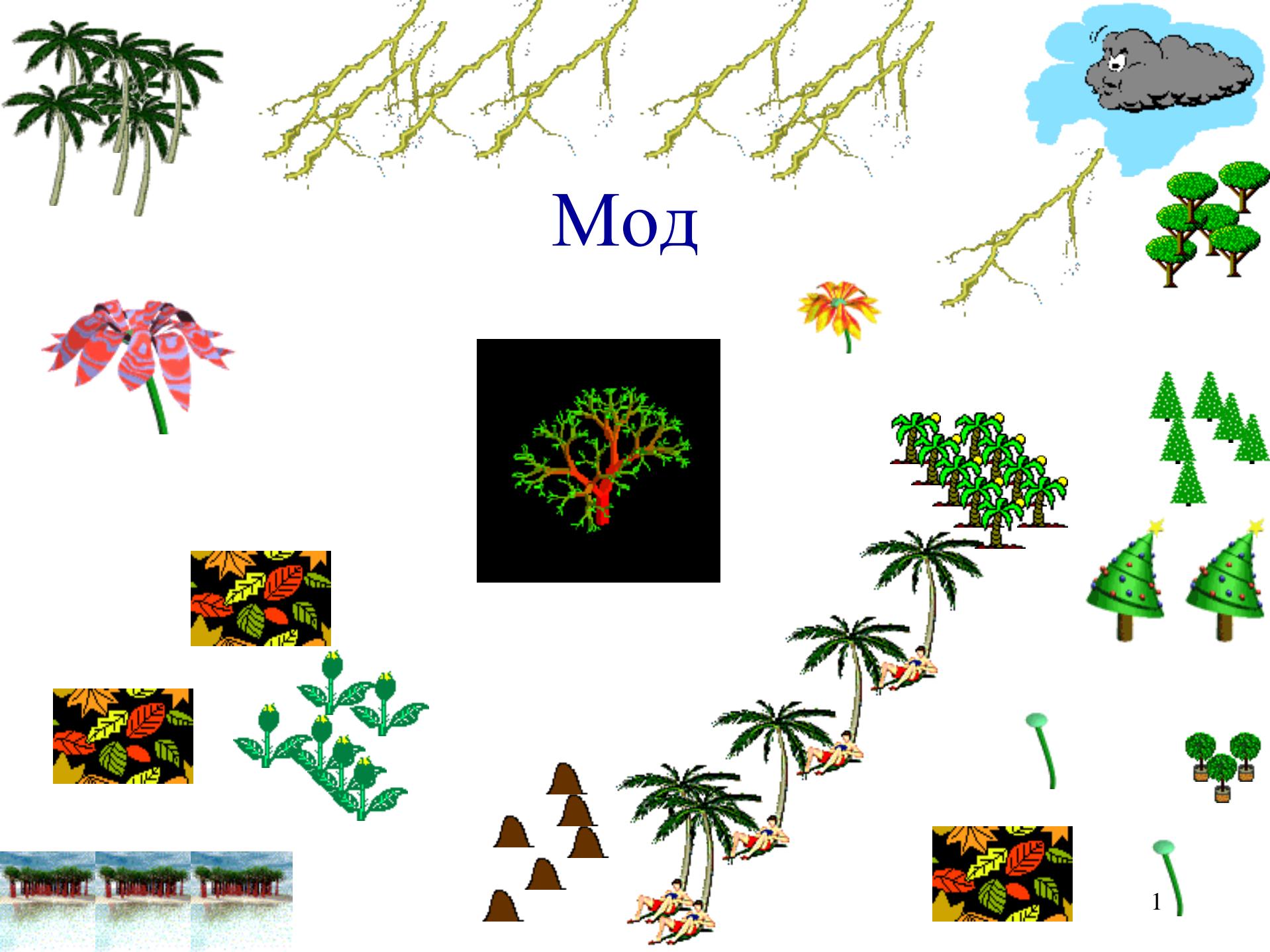
- Шахалт.
  - $O(n)$  дундаж хугацаа, үүнд **n** шахагдсан текстийн урт.
- Задлалт.
  - $O(n)$  хугацаа, үүнд **n** задалсан текстийн урт.

- “Анхны тоонд хуваах үлдэгдэл” аргачлал нь бидний мэдэх хаашиング (hashing) алгоритм билээ. Энэ аргачлалд түлхүүр утга нь  $N$  тоонд хуваагддаг ба мөн түүний ижил хаашын утга хэмээн нэрлэгддэг үлдэгдэл нь шууд хаашын хүснэгтэнд индексээр хэрэглэгддэг.  $N$  бол хаяглаж болох зайны хэмжээтэй тэнцүү эсвэл хамгийн ойр дөхөх анхны тоо юм. Хэрэв 20 хаяглалт хийхүйц зтай бол дараах сонголтуудын аль нь түлхүүр утга 136 байхад тооцоолж гарах хаашын утга вэ? Энд, анхны тоо гэдэг нь нэгээс бусад бүх тоонд тэгш хуваагдаггүй тоог хэлнэ. 2, 3, 5, 7, 11, ба 13 нь эхний 6 анхны тоо юм.
  - A) 0
  - B) 1
  - C) 3
  - D) 16

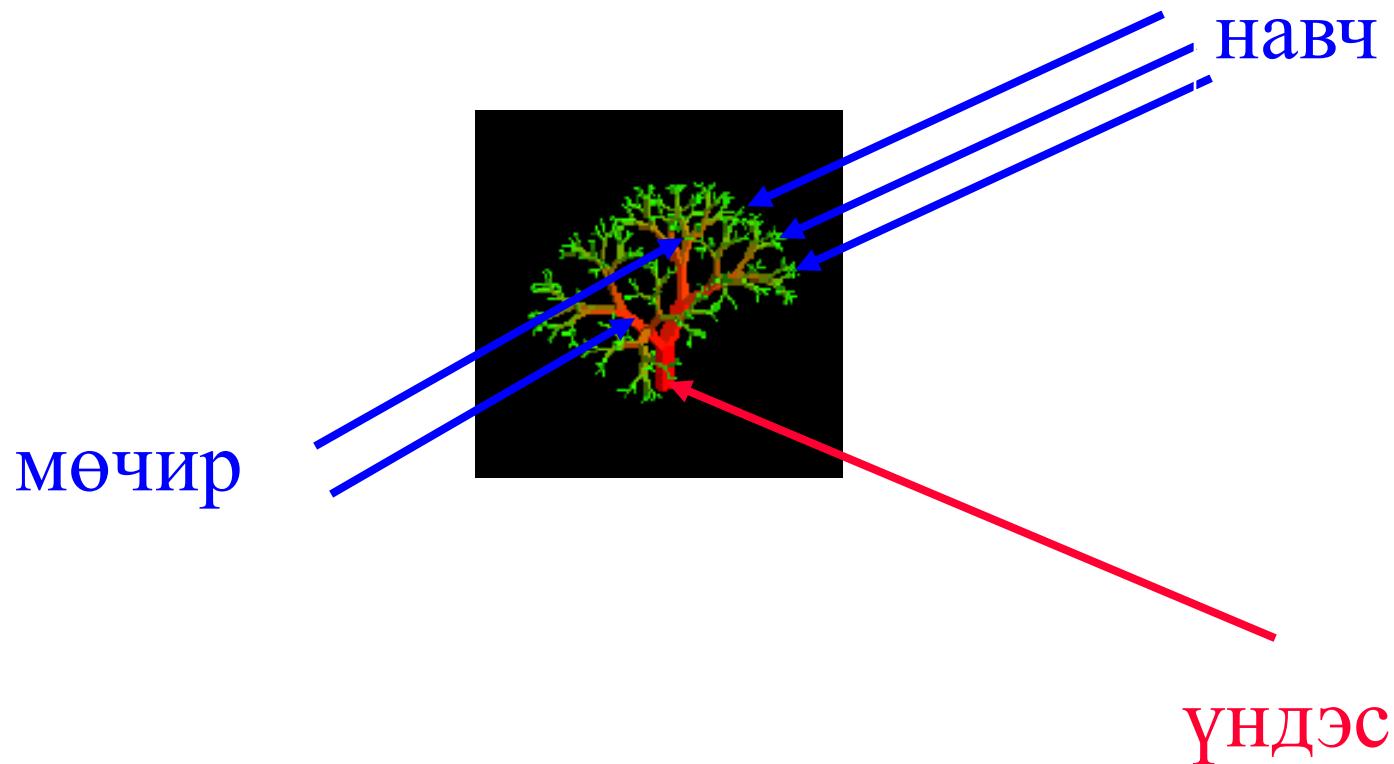
- [www.itpec.org](http://www.itpec.org)



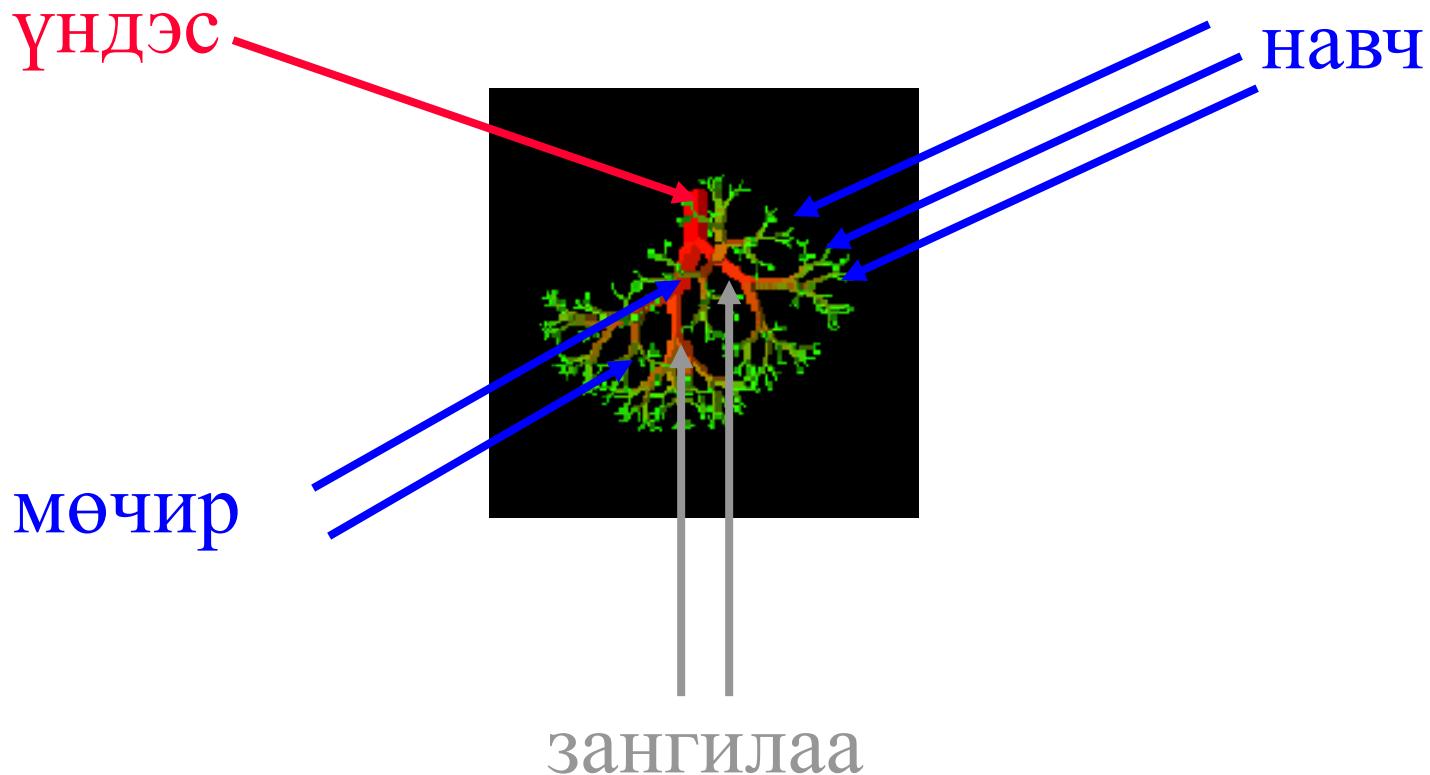
# Мод



# Байгальд хайртай хүний нүдээр



# Компьютерийн хүний нүдээр





# Шугаман жагсаалт ба Мод



- Шугаман жагсаалт цуварч эрэмбэлэгдсэн өгөгдөл тохиромжтой.
  - $(e_0, e_1, e_2, \dots, e_{n-1})$
  - Долоо хоногийн өдрүүд.
  - Жилийн сарууд.
  - Ангийн оюутнууд.
- Мод үелэж эрэмбэлэгдсэн өгөгдөл тохиромжтой.
  - Байгуулгын ажиллагсад.
    - Захирал, дэд захирал, менежер, гэх мэт.
  - Java-ийн классууд.
    - Object үечлэлийн оройд байдаг.
    - Object –ийн дэд классууд дараа нь, гэх мэт .

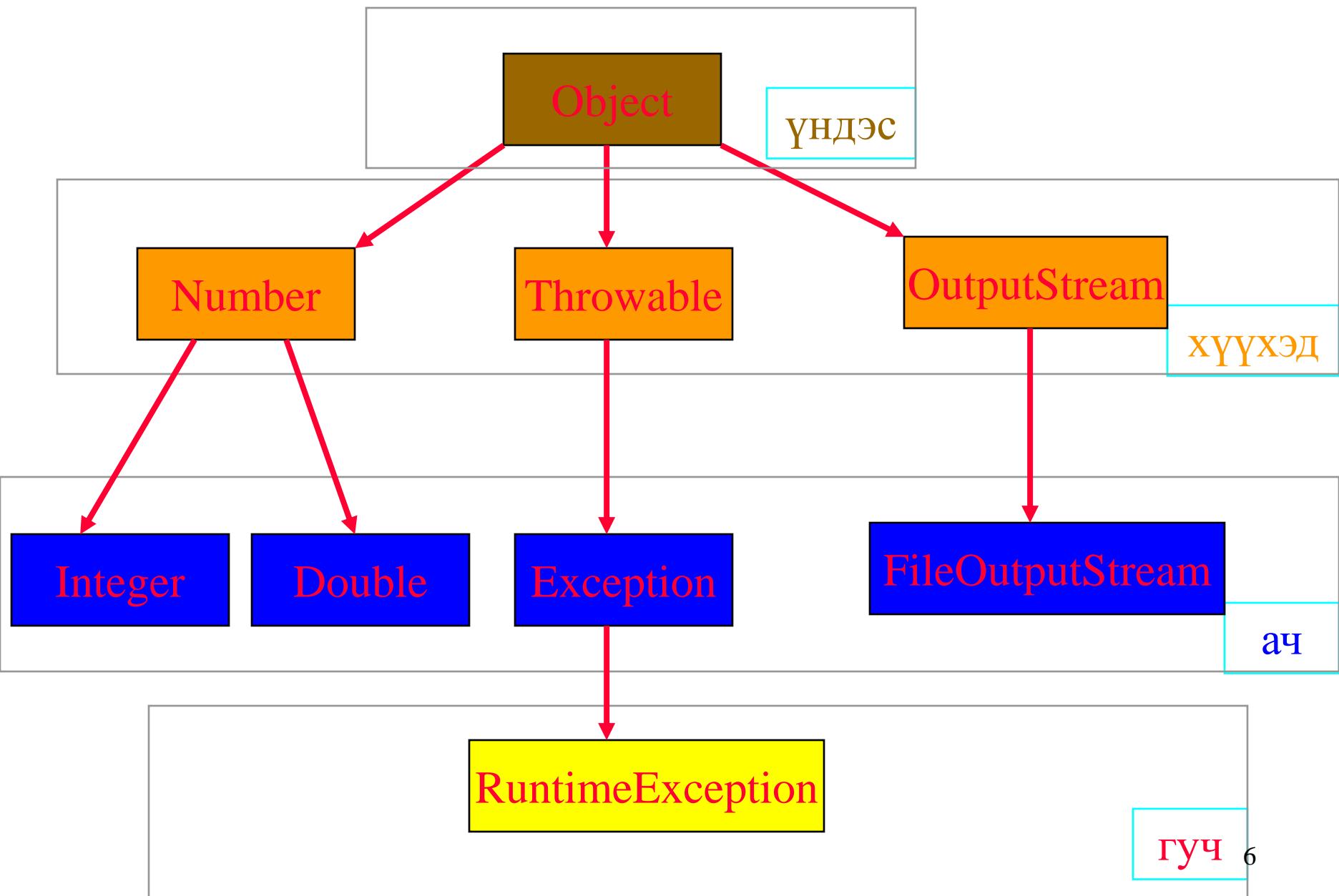


# Үелсэн өгөгдөл ба Мод



- Дээд/оройн үеийн элемент бол **root-үндэс**.
- Хоёрдахь үеийн элементүүд бол үндсээс гарсан **children-хүүхдүүд**.
- Гуравдахь үеийн элементүүд бол үндсээс гарсан **grandchildren-ач нар**, гэх мэт.
- Хүүхэдгүй элемент бол **leaves-навчис**.

# Java-ийн классууд





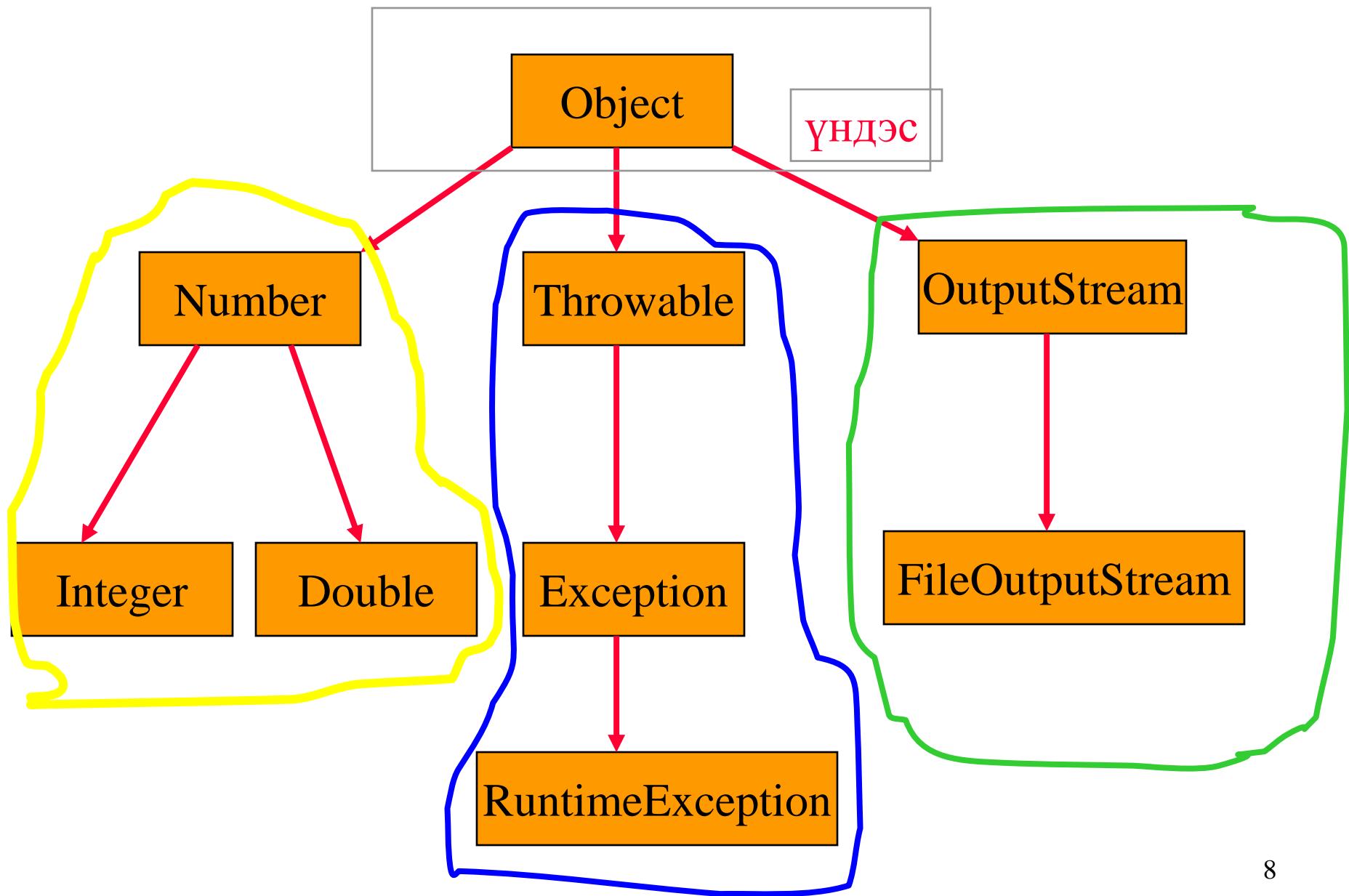
# Тодорхойлолт



- Мод **t** элементүүдийн хоосон бус төгсгөлтэй олонлог.
- Элементүүдийн нэгийг нь үндэс гэнэ.
- Бусад элементүүд(хэрвээ байгаа бол) модолж хуваагдана, түүнийг **t** -ийн дэд мод гэнэ

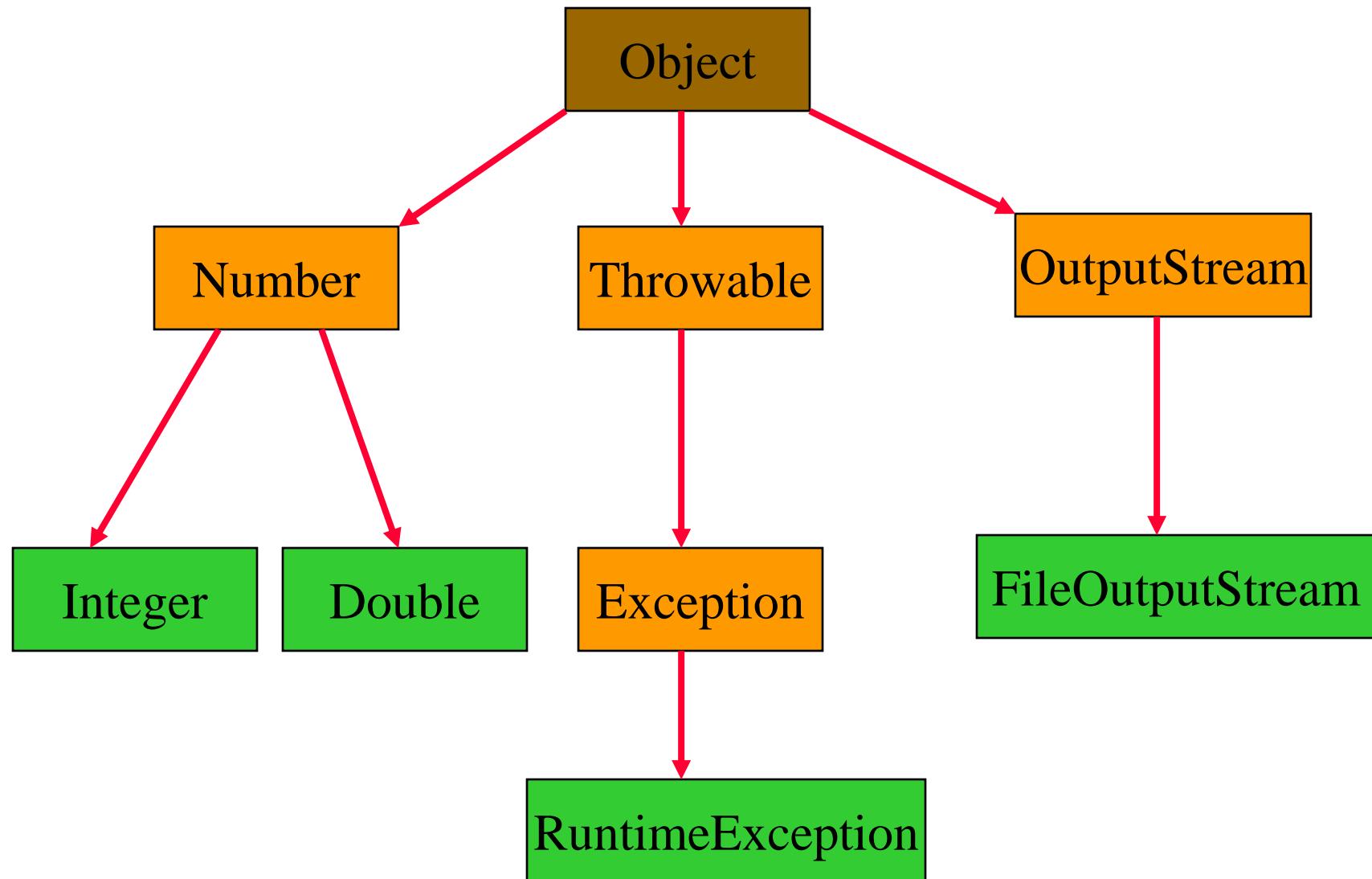


# ДЭД МОД

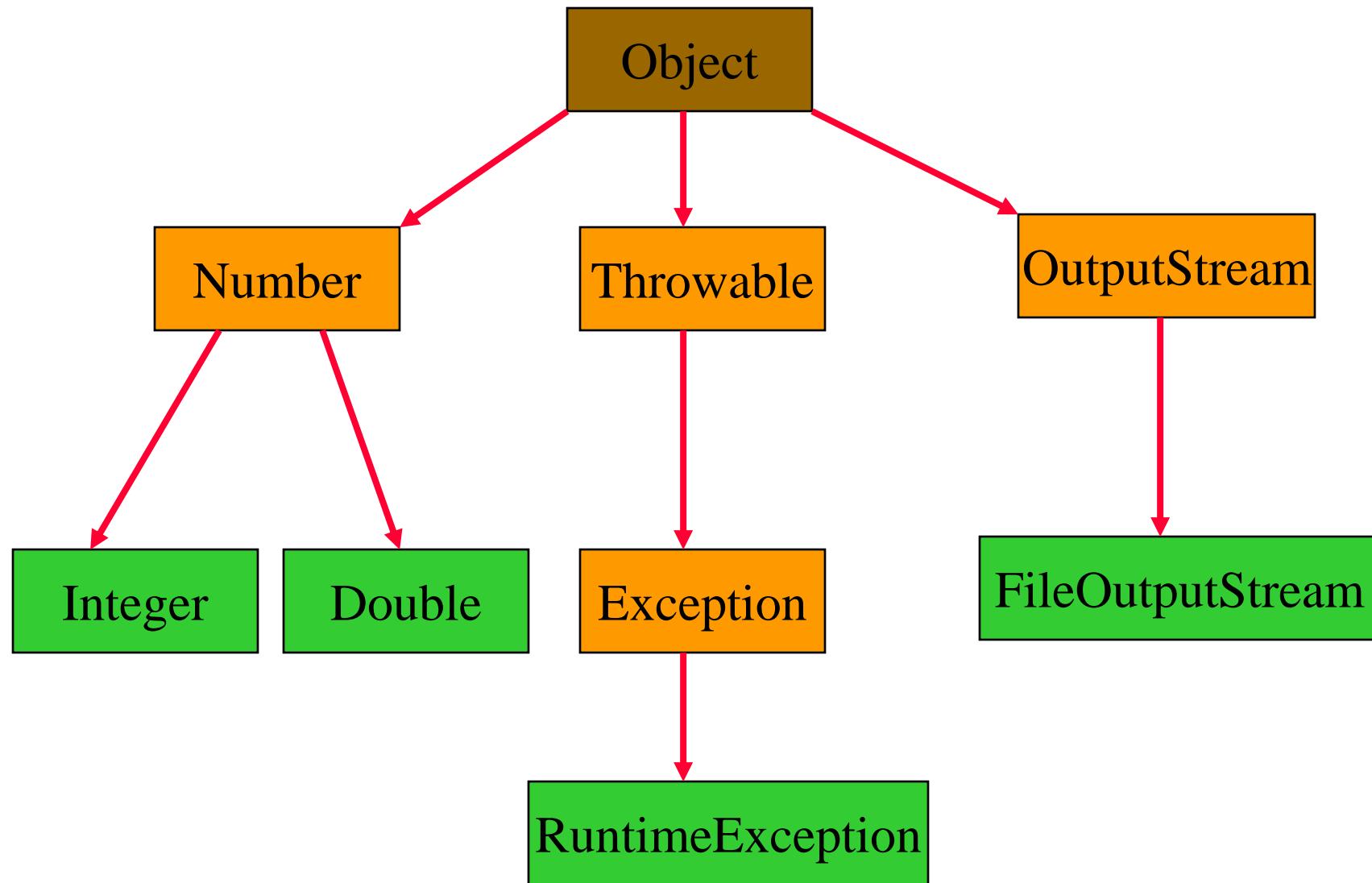




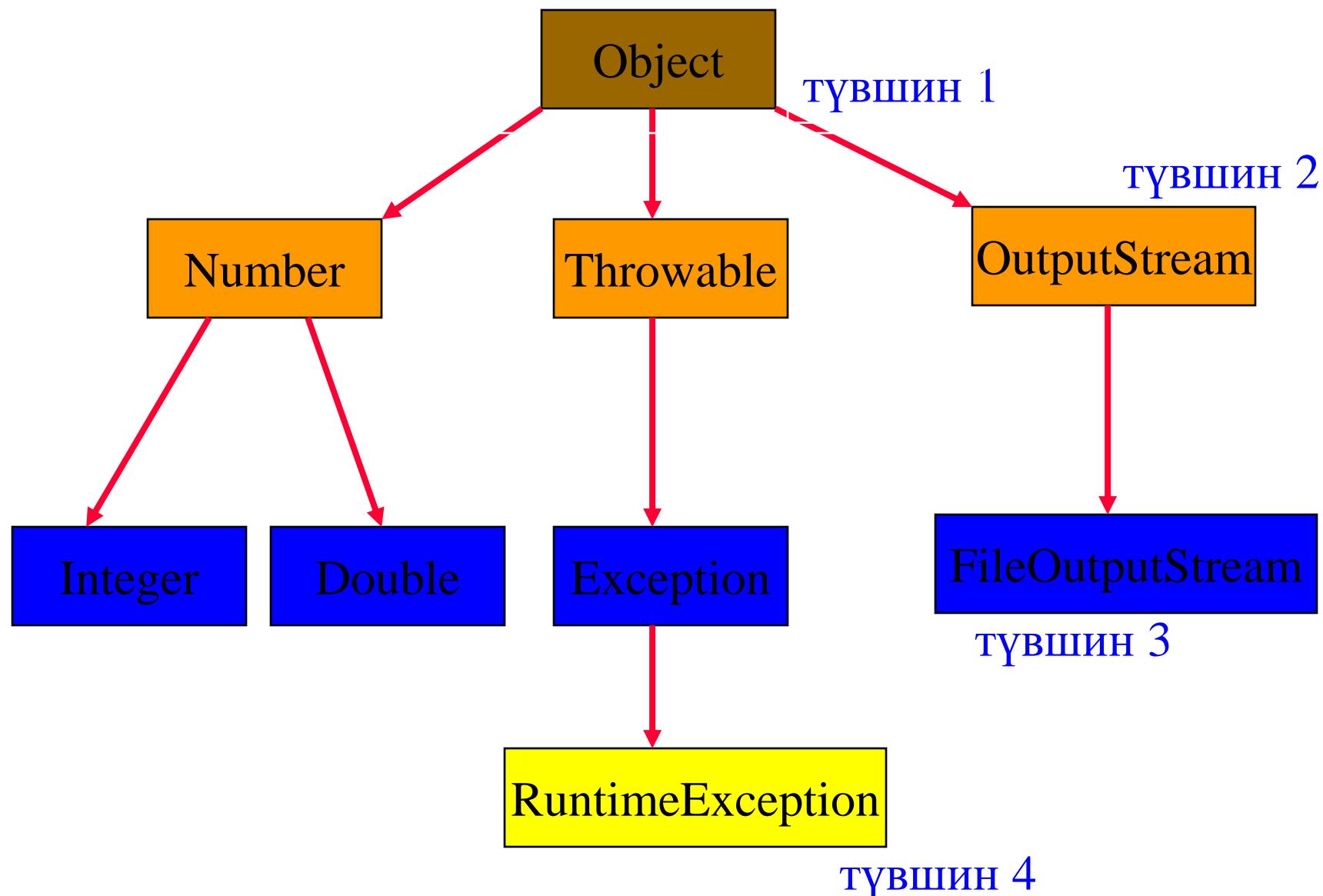
# Навчис



# Parent, Grandparent, Siblings, Ancestors, Descendants – Эцэг, Өвөг эцэг, Ах дүүс, Удам, Хойч



# Түвшин-үе



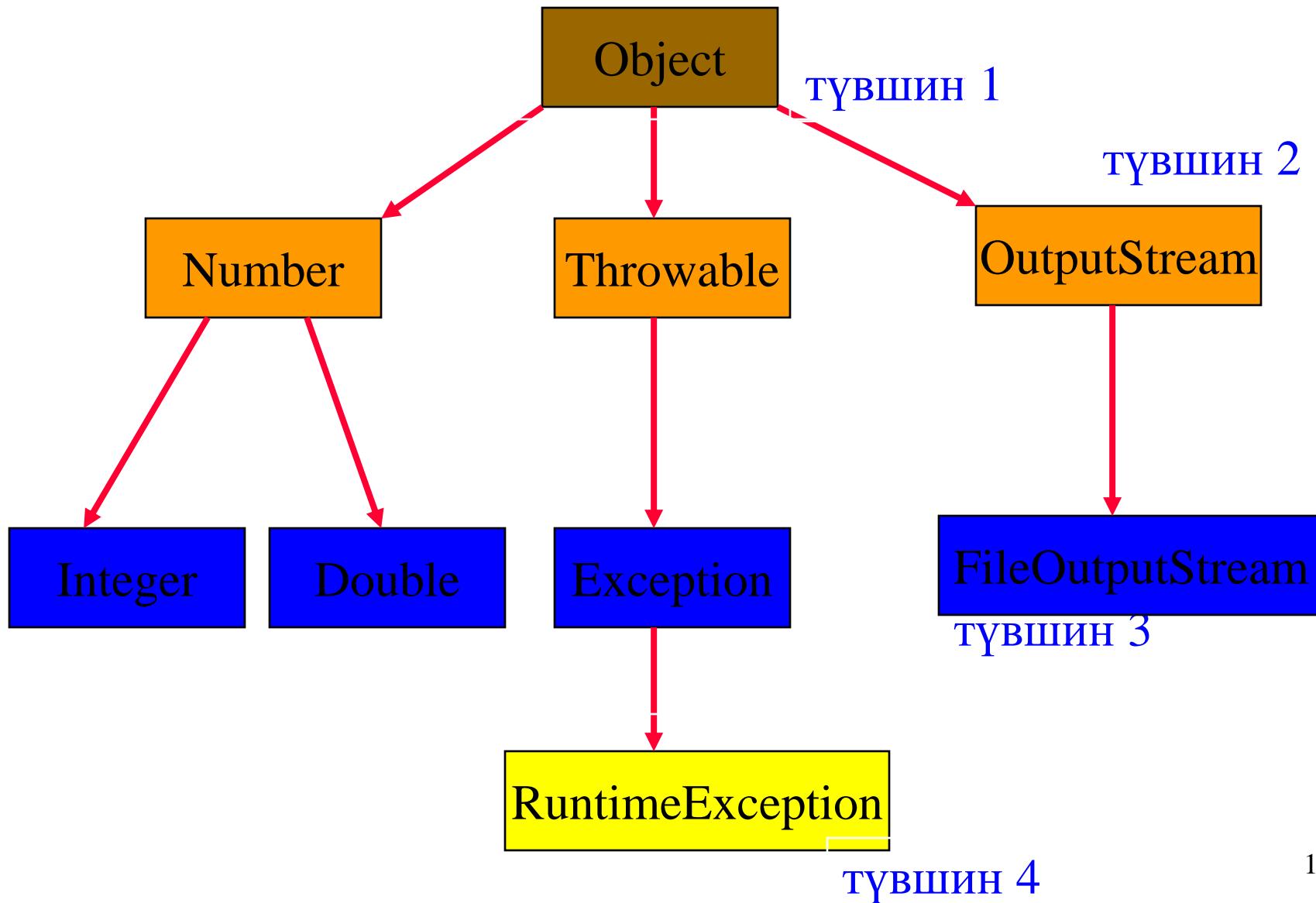


# Анхааруулга

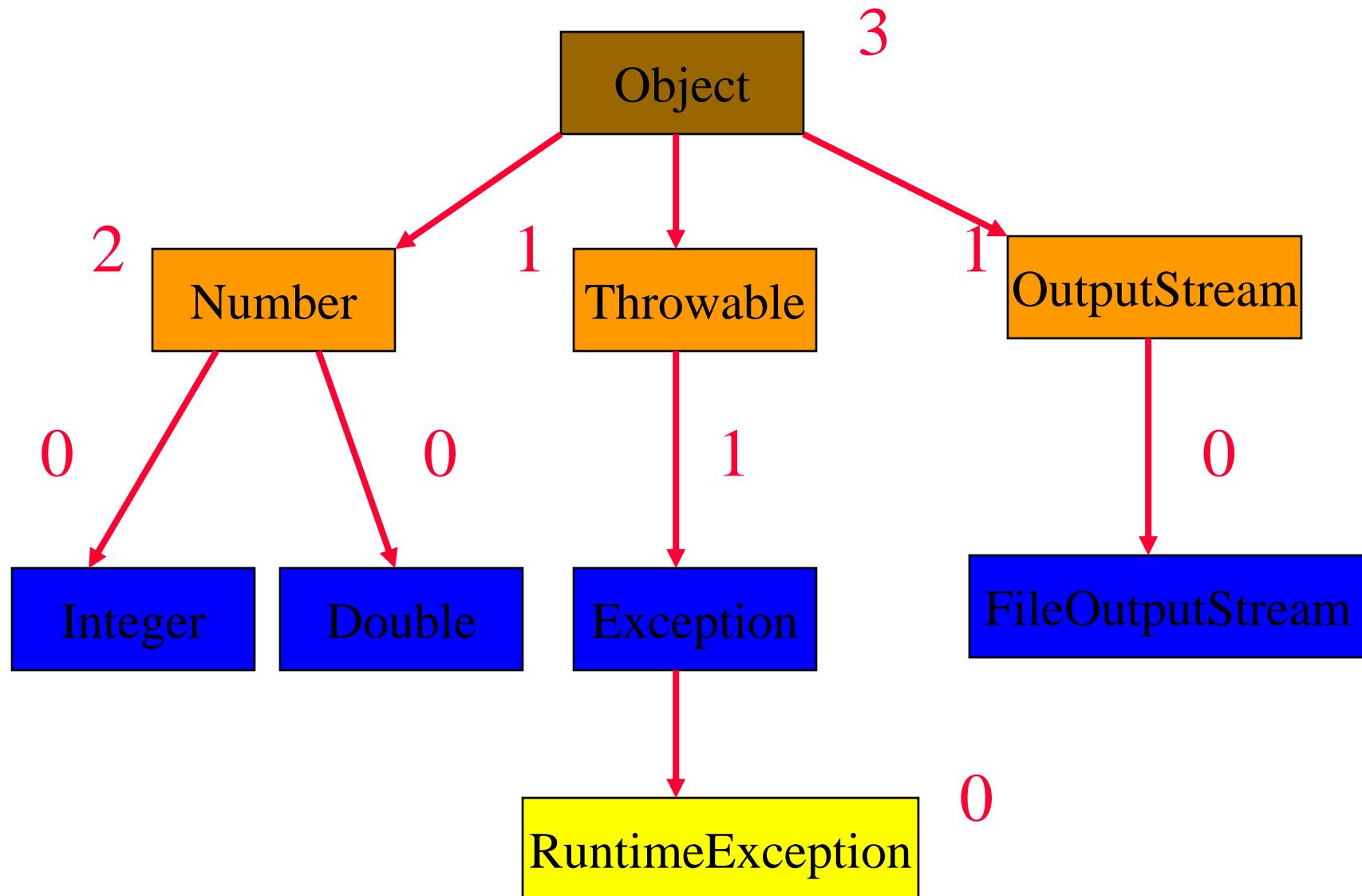


- Зарим номонд түвшинг **1** -ээс биш 0 –ээс эхэлж дугаарладаг.
- Үндэсний түвшин **0**.
- Түүний хүүхдийн түвшин **1**.
- Ач/зээгийн түвшин **2**.
- Гэх мэт.
- Бид түвшинг **1** -ээс эхэлж дугаарлана

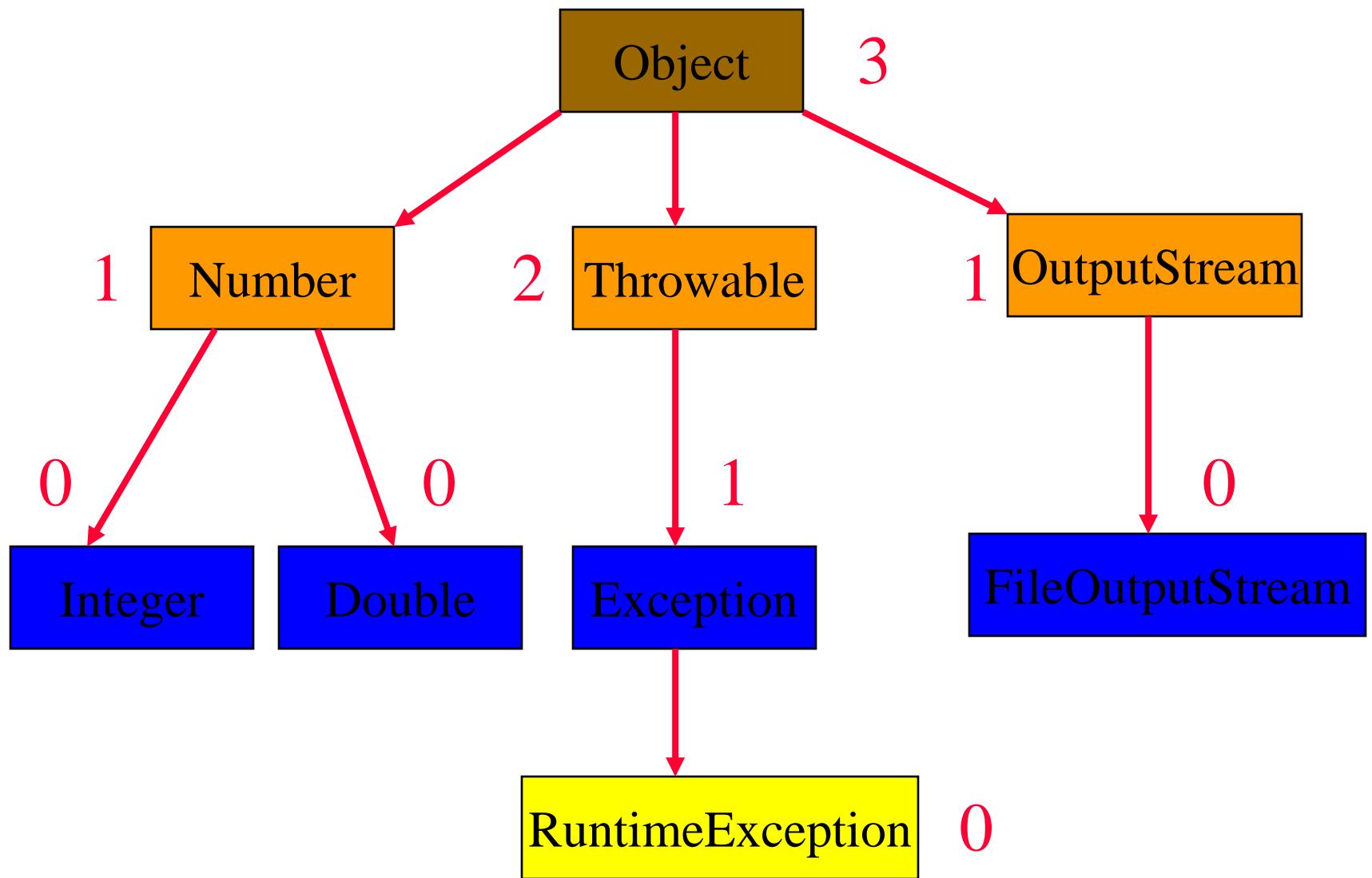
# height = depth = түвшний тоо



# Зангилааны зэрэг = Хүүхдийн тоо



# Модны зэрэг = Max(Зангилааны зэрэг)



Модны зэрэг = 3.

# Хоёртын мод

- Элементүүдийн төгсгөлтэй (хоосон байж болно) цуглуулга.
- **Хоосон бус** хоёртын мод **root-үндэс** элементтэй.
- Бусад элементүүд (байгаа бол) **хоёр** хоёртын модонд хуваагдана.
- Тэднийг хоёртын модны **left-зүүн** ба **right-баруун** дэд моднууд гэнэ.

# Мод, Хоёртын модны ялгаа

- Хоёртын модны зангилааны зэрэг **2** -оос ихгүй байхад <> Хязгааргүй.
- Хоёртын мод хоосон байж болно <>Хоосон бус.

# Мод, Хоёртын модны ялгаа

- Хоёртын модны дэд моднууд эрэмбэлэгдсэн  $\leftrightarrow$  Эрэмбэлэгдээгүй.



- Хоёртын мод гэж харвал ялгаатай.
- Мод гэж харвал адилхан.

# Арифметик илэрхийлэл

- $(a + b) * (c + d) + e - f/g*h + 3.25$
- Илэрхийлэл 3 зүйлийг нэгтгэдэг:
  - Үйлдэл (+, -, /, \*).
  - Гишигд (a, b, c, d, e, f, g, h, 3.25, (a + b), (c + d), etc.).
  - Зааглагч ((, )).

# Үйлдлийн зэрэг

- Үйлдэлд орох гишүүдийн тоо.
- Хоёр гишүүнтэй үйлдэл. Binary Operator
  - $a + b$
  - $c / d$
  - $e - f$
- Нэг гишүүнтэй үйлдэл. Unary Operator
  - $+ g$
  - $- h$

# Infix хэлбэр

- Илэрхийлэл бичих ердийн арга.
- Хоёр гишүүнтэй үйлдэл зүүн, баруун гишүүний **хооронд** бичигдэнэ.
  - $a * b$
  - $a + b * c$
  - $a * b / c$
  - $(a + b) * (c + d) + e - f/g*h + 3.25$

# Үйлдлийн ахлах чанар

- Үйлдлүүд яаж хийгдэх вэ?
  - $a + b * c$
  - $a * b + c / d$
- Priority-Үйлдлийн ахлах чанараар зохицуулагдана.
  - $\text{priority}(*) = \text{priority}(/) > \text{priority}(+) = \text{priority}(-)$
- Гишуун хоёр үйлдлийн хооронд байвал илүү ахлах чанартай үйлдэлд харьяалагдана.

# Tie Breaker-Хайнцааг таслах

- Гишиг ижил ахлах чанартай хоёр үйлдлийн хооронд байвал зүүн талын үйлдэлд харьяалагдана.
  - $a + b - c$
  - $a * b / c / d$

# Зааглагч

- Зааглагчийн дотор бичигдсэн дэд илэрхийллийг нэг гишүүн гэж үзнэ.
  - $(a + b) * (c - d) / (e - f)$

# Infix илэрхийлэл задлан хийхэд хэцүү

- Үйлдлийн ахлах чанар, хайнцааг таслах, зааглагч шаардлагатай.
- Энэ нь компьютерын бодолтыг хүндруулдэг.
- Postfix болон prefix илэрхийллийн хэлбэрүүд үйлдлийн ахлах чанар, хайнцаа таслах, зааглагчаас хамаarahгүй.
- Иймд эдгээр хэлбэрийн илэрхийллийг компьютер амархан боддог.

# Postfix хэлбэр

- Хувьсагч, тогтмолууд адилхан бичигдэнэ.
  - a, b, 3.25
- Үйлдлийн гишүүдийн дараалал Infix, Postfix хэлбэрүүдэд адилхан.
- Үйлдлүүд postfix хэлбэрийн гишүүдийнхээ **ард** шууд бичигддэг.
  - Infix = a + b
  - Postfix = ab+

# Postfix Жишиээ

- Infix =  $a + b * c$ 
  - Postfix = a b c \* +
- Infix =  $a * b + c$ 
  - Postfix = a b \* c +
- Infix =  $(a + b) * (c - d) / (e + f)$ 
  - Postfix = a b + c d - \* e f + /

# Нэг гишигүүнтэй Үйлдэл

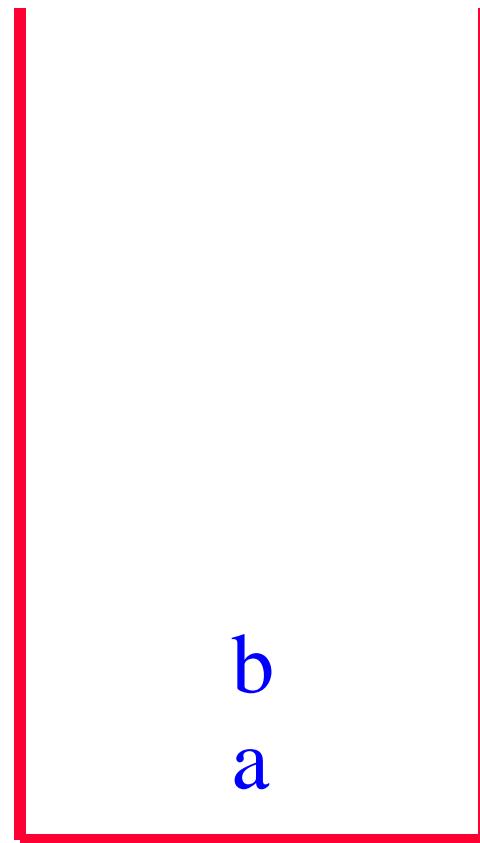
- Шинэ тэмдгээр орлуулна.
  - $+ a \Rightarrow a @$
  - $+ a + b \Rightarrow a @ b +$
  - $- a \Rightarrow a ?$
  - $- a - b \Rightarrow a ? b -$

# Postfix бодолт

- Postfix илэрхийллийг зүүнээс баруун тийш шинжихдээ гишүүдийг стект хийнэ.
- Үйлдэл таарагалдвал стекээс хэрэгтэй гишүүдээ аваад үйлдлийг гүйцэтгэж хариуг стект хийнэ.
- Postfix –д үйлдэл гишүүдийнхээ араас ордог болохоор энэ арга ажиллана.

# Postfix бодолт

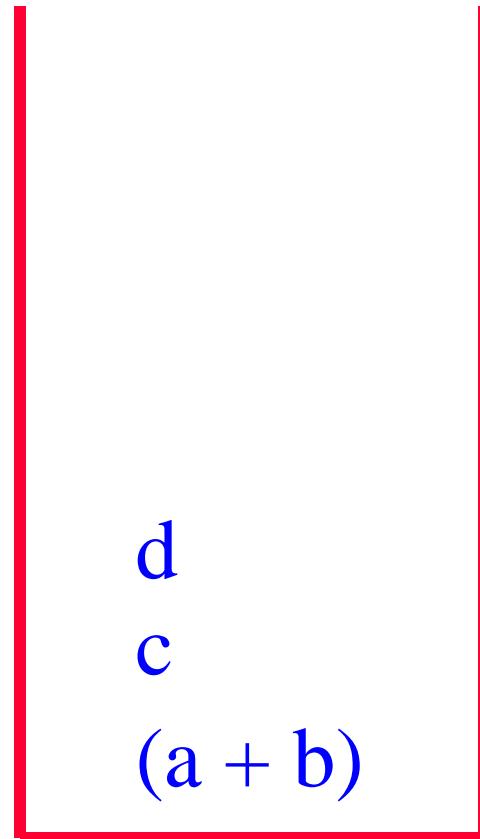
- $(a + b) * (c - d) / (e + f)$
- a b + c d - \* e f + /
- a b + c d - \* e f + /
- a b + c d - \* e f + /
- a b + c d - \* e f + /



стек

# Postfix бодолт

- $(a + b) * (c - d) / (e + f)$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$



стек

# Postfix бодолт

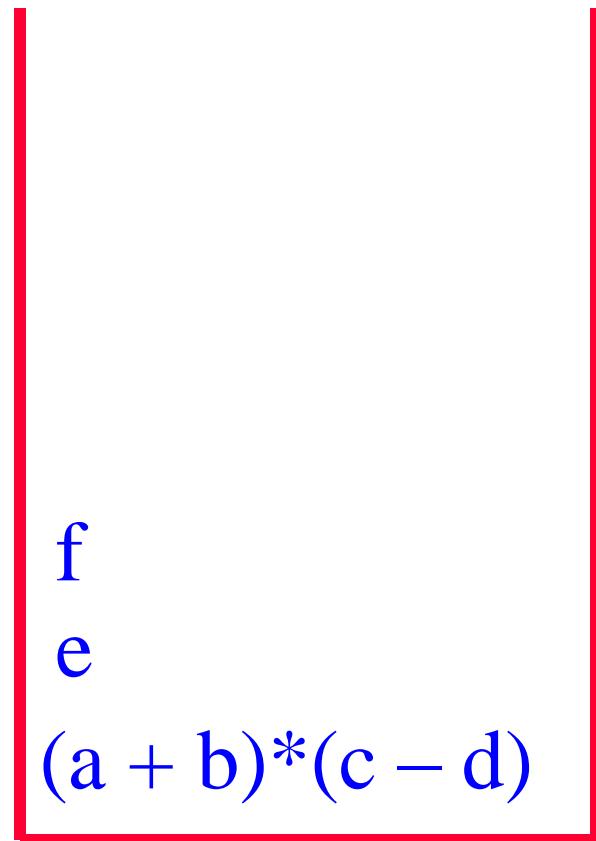
- $(a + b) * (c - d) / (e + f)$
- $a\ b\ +\ c\ d\ -\ *\ e\ f\ +\ /$
- $a\ b\ +\ c\ d\ -\ *\ e\ f\ +\ /$

(c - d)  
(a + b)

стек

# Postfix бодолт

- $(a + b) * (c - d) / (e + f)$
- a b + c d - \* e f + /
- a b + c d - \* e f + /
- a b + c d - \* e f + /
- a b + c d - \* e f + /
- a b + c d - \* e f + /



стек

# Postfix бодолт

- $(a + b) * (c - d) / (e + f)$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$
- $a b + c d - * e f + /$

( $e + f$ )  
 $(a + b)^*(c - d)$

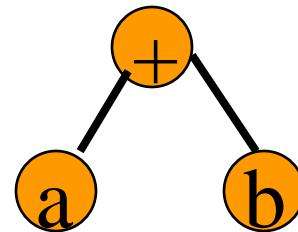
стек

# Prefix хэлбэр

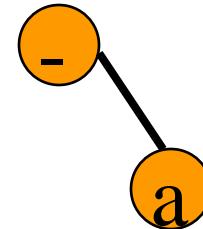
- Хувьсагч, тогтмолууд адилхан бичигдэнэ.
  - $a, b, 3.25$
- Үйлдлийн гишүүдийн дараалал infix, prefix хэлбэрүүдэд адилхан
- Үйлдлүүд postfix хэлбэрийн гишүүдийнхээ **ӨМНӨ** шууд бичигддэг.
  - Infix =  $a + b$
  - Postfix =  $ab+$
  - Prefix =  $+ab$

# Хоёртын модны хэлбэр

- $a + b$

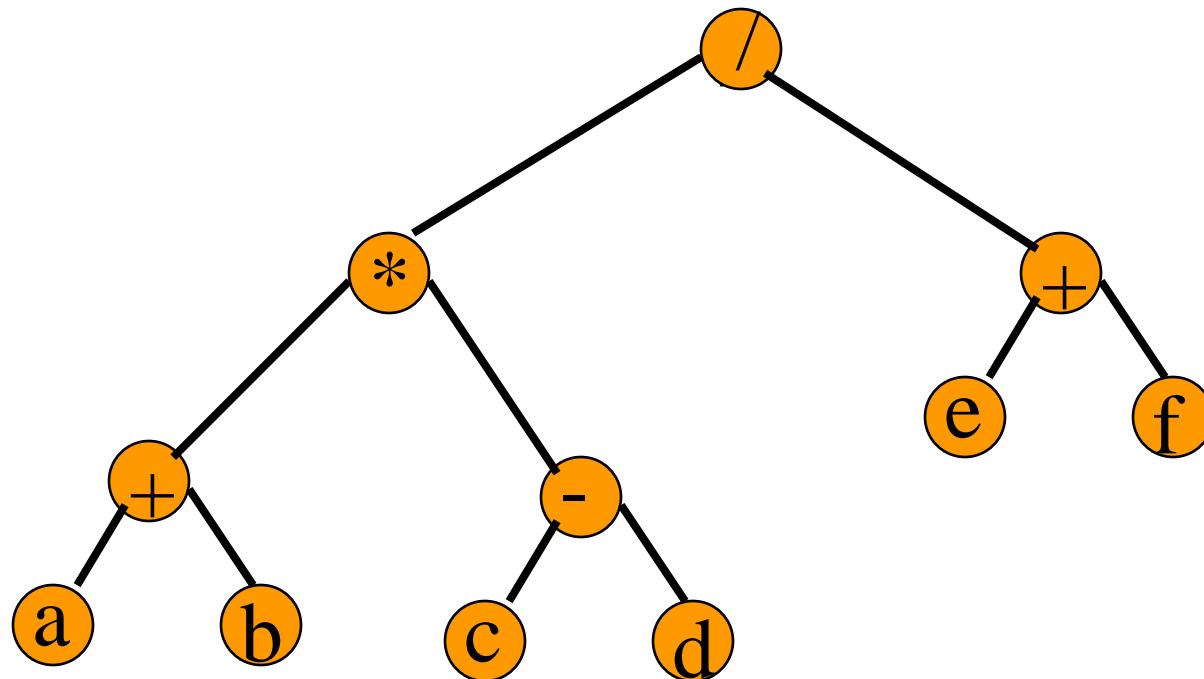


- $- a$



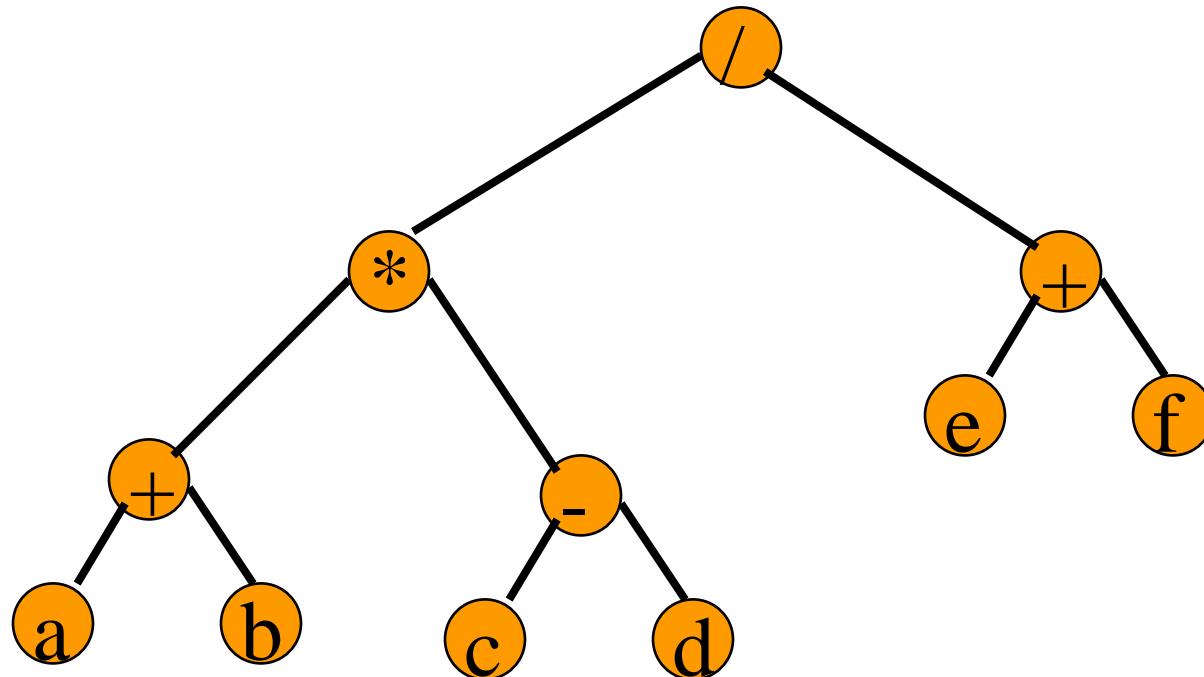
# Хоёртын модны хэлбэр

- $(a + b) * (c - d) / (e + f)$

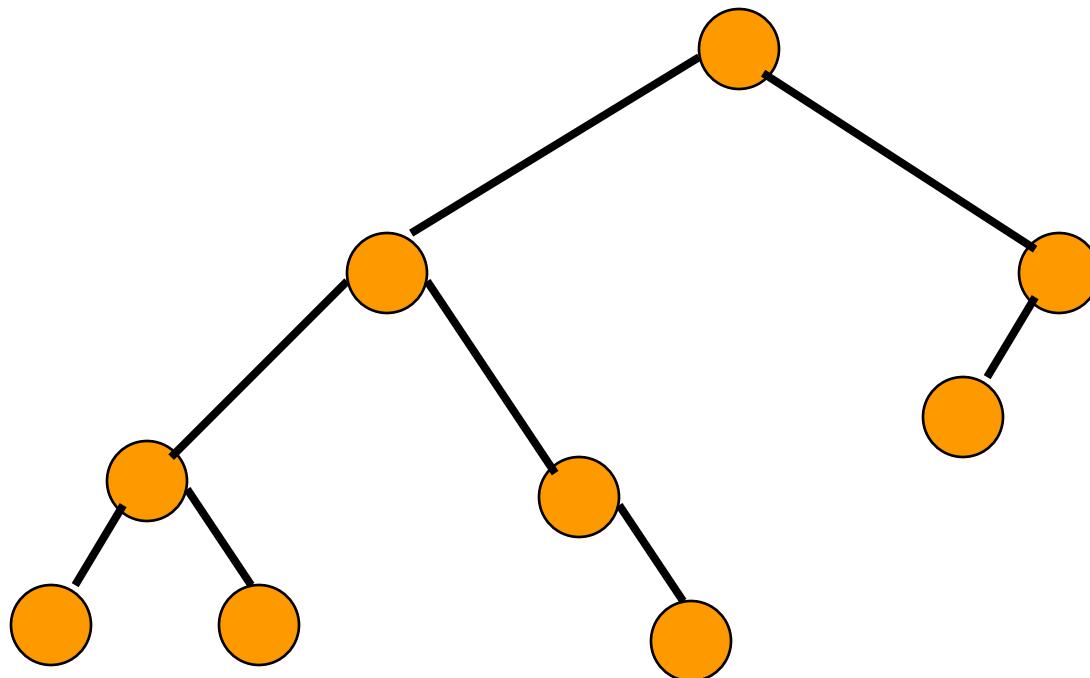


# Хоёртын модны хэлбэрийн давуу тал

- Үйлдлийн зүүн, баруун гишүүд ил харагддаг.
- Илэрхийллийн хоёртын модны хэлбэр дээр кодыг оновчлох алгоритм сайн ажилладаг.
- Илэрхийллийг рекурсив аргаар бодоход хялбар.

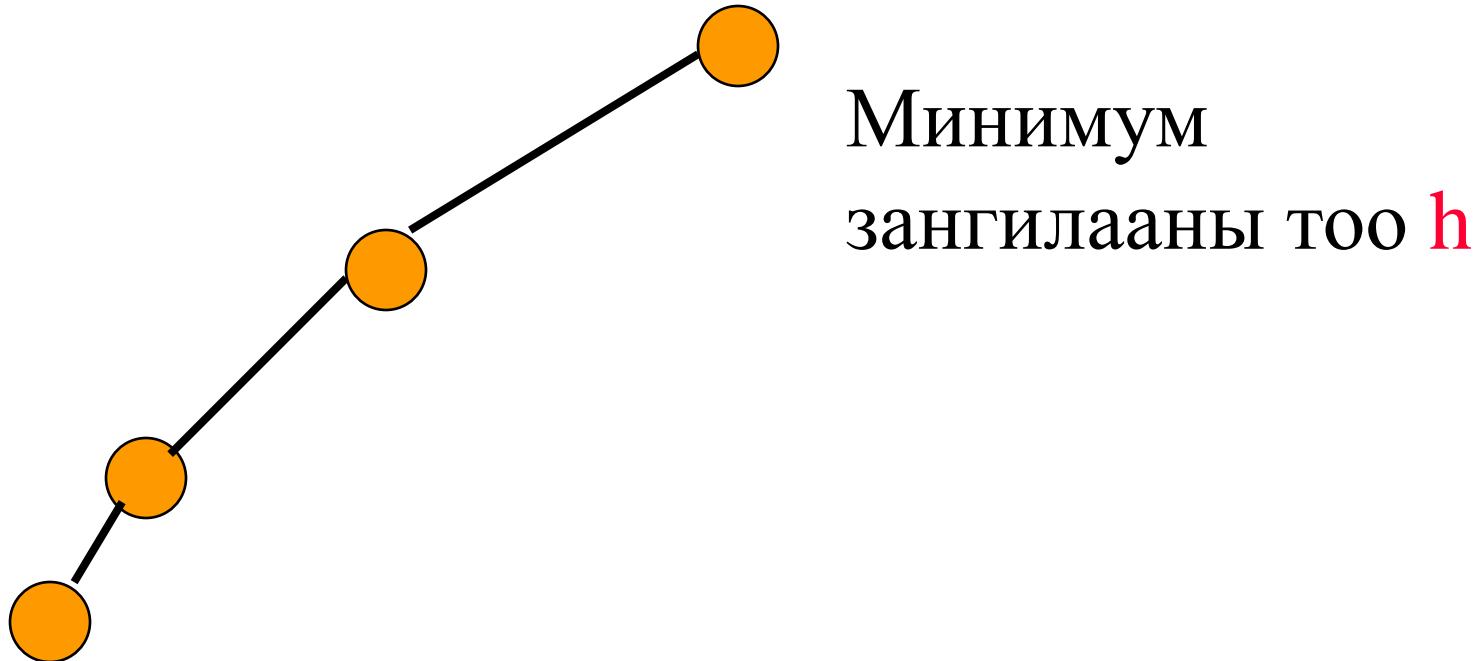


# Хоёртын модны шинж ба дүрслэл



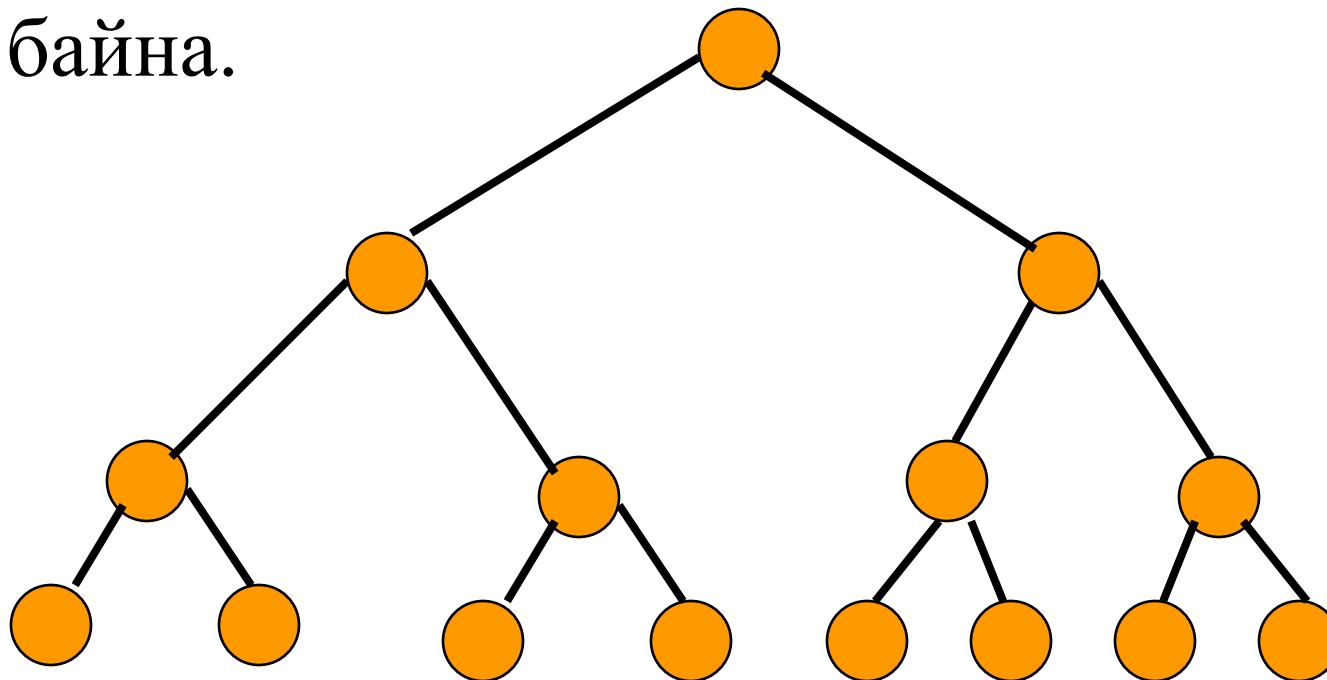
# Минимум зангилааны тоо

- $h$  өндөртэй хоёртын модны минимум зангилааны тоо
- Эхний  $h$  түвшин тус бүрд ядаж нэг зангилаа байна



# Максимум зангилааны тоо

- Эхний  $h$  түвшин бүрт боломжит бүх зангилаа байна.



Максимум зангилааны тоо

$$= 1 + 2 + 4 + 8 + \dots + 2^{h-1}$$

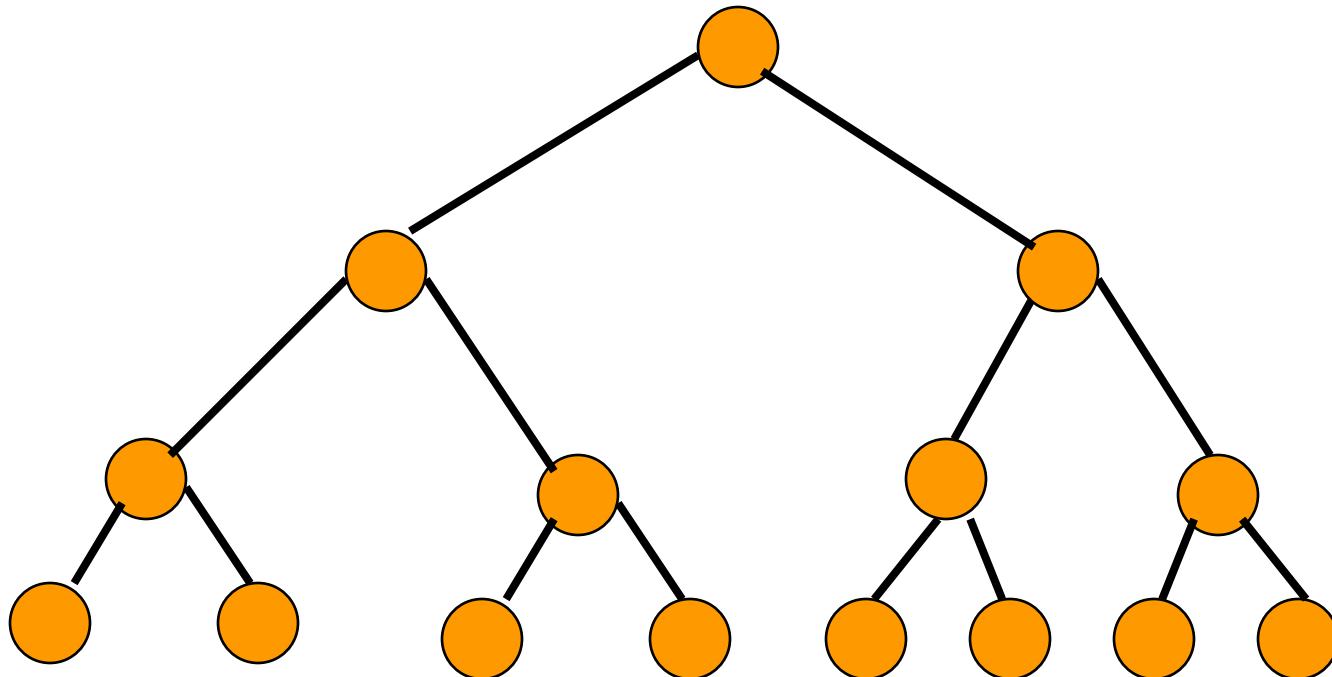
$$= 2^h - 1$$

# Зангилааны тоо ба Θндөр

- Хэрвээ  $n$  нь  $h$  өндөртэй хоёртын модны зангилааны тоо бол:
- $h \leq n \leq 2^h - 1$
- $\log_2(n+1) \leq h \leq n$

# Хоёртын бүтэн мод

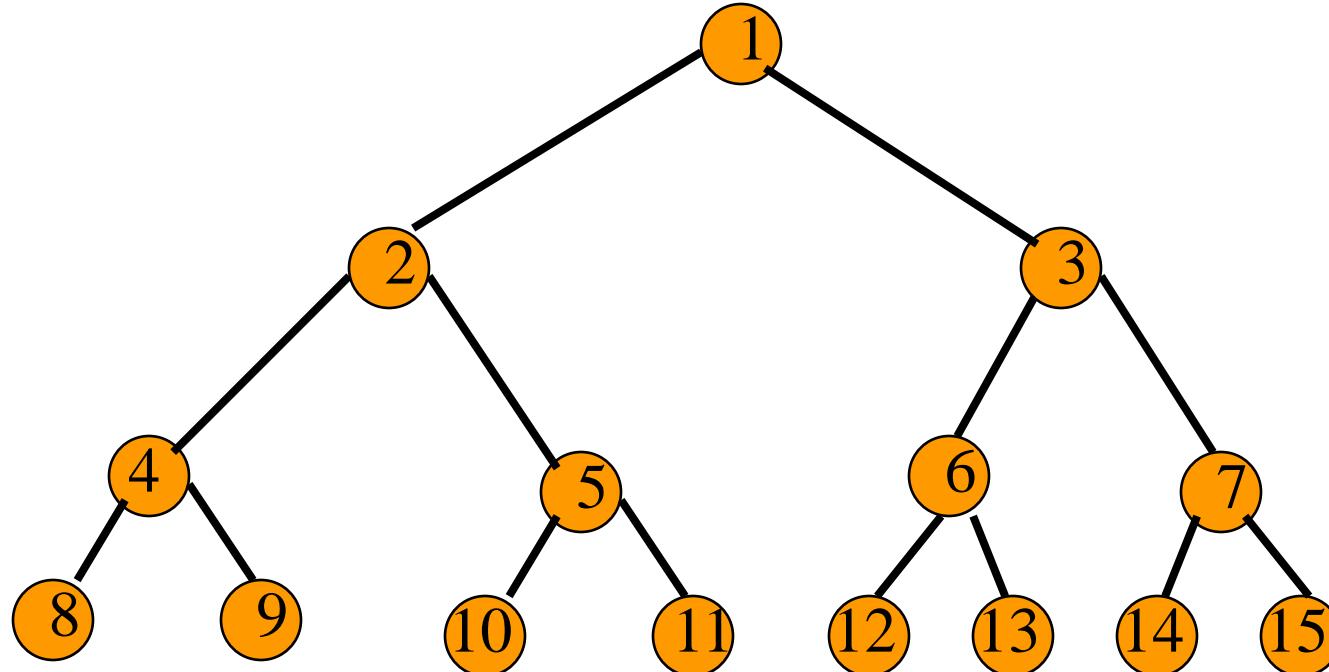
- $h$  өндөртэй хоёртын бүтэн модонд  $2^h - 1$  зангилаа байна.



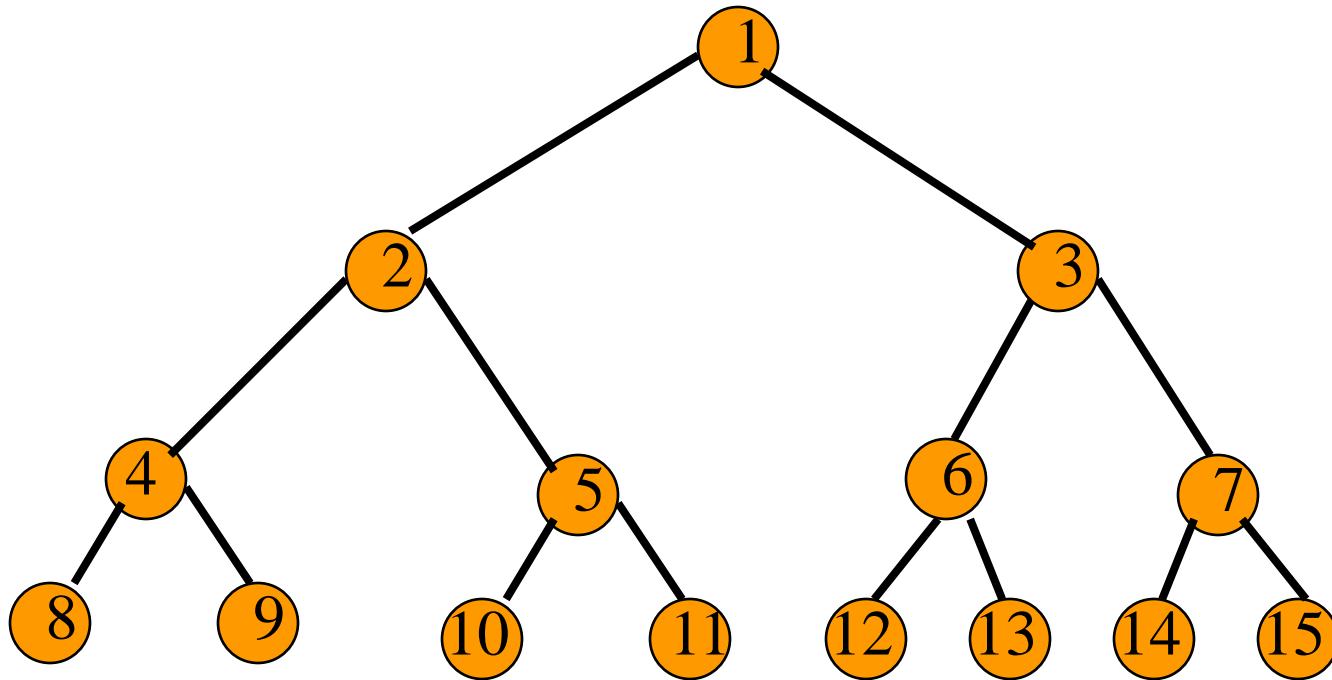
4 –н өндөртэй бүтэн хоёртын мод.

# Хоёртын бүтэн модны зангилааг дугаарлах

- Зангилааны дугаар  $1 - 2^h - 1$ .
- Түвшин дээрээс доош дугаарлагдана
- Түвшин дотроо зүүнээс баруун тийш.

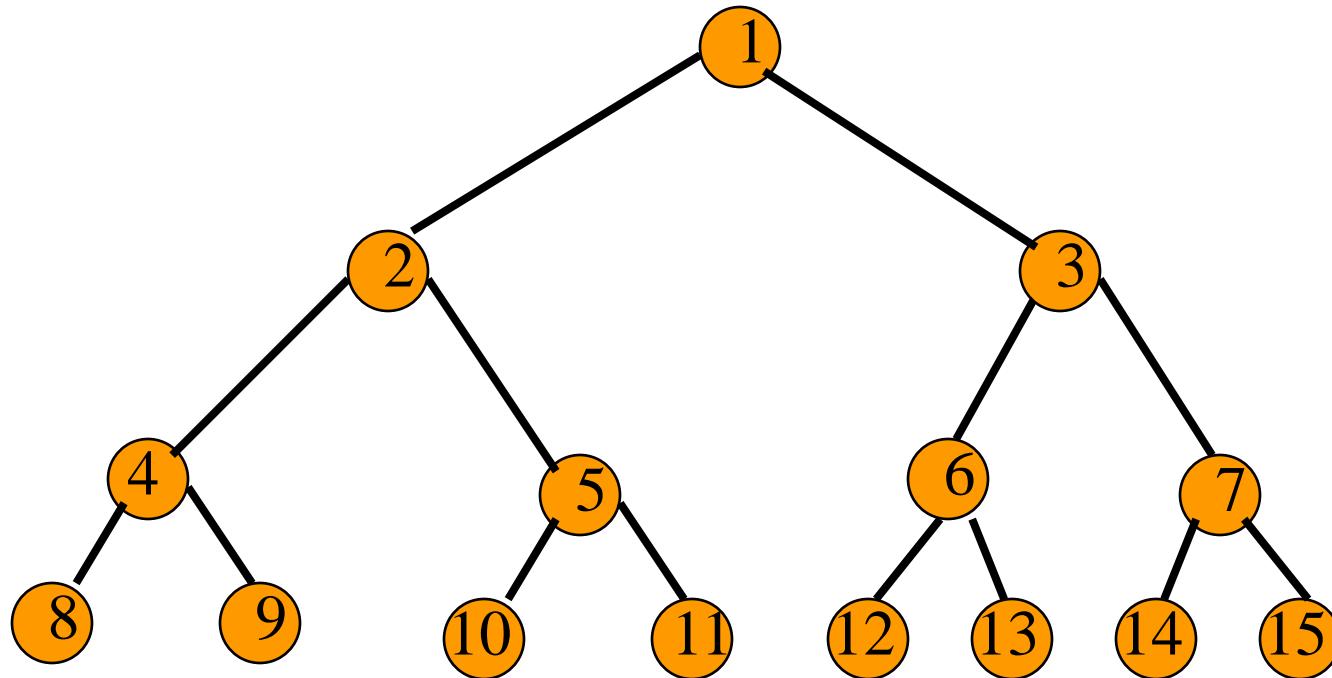


# Зангилааны дугаарын шинж



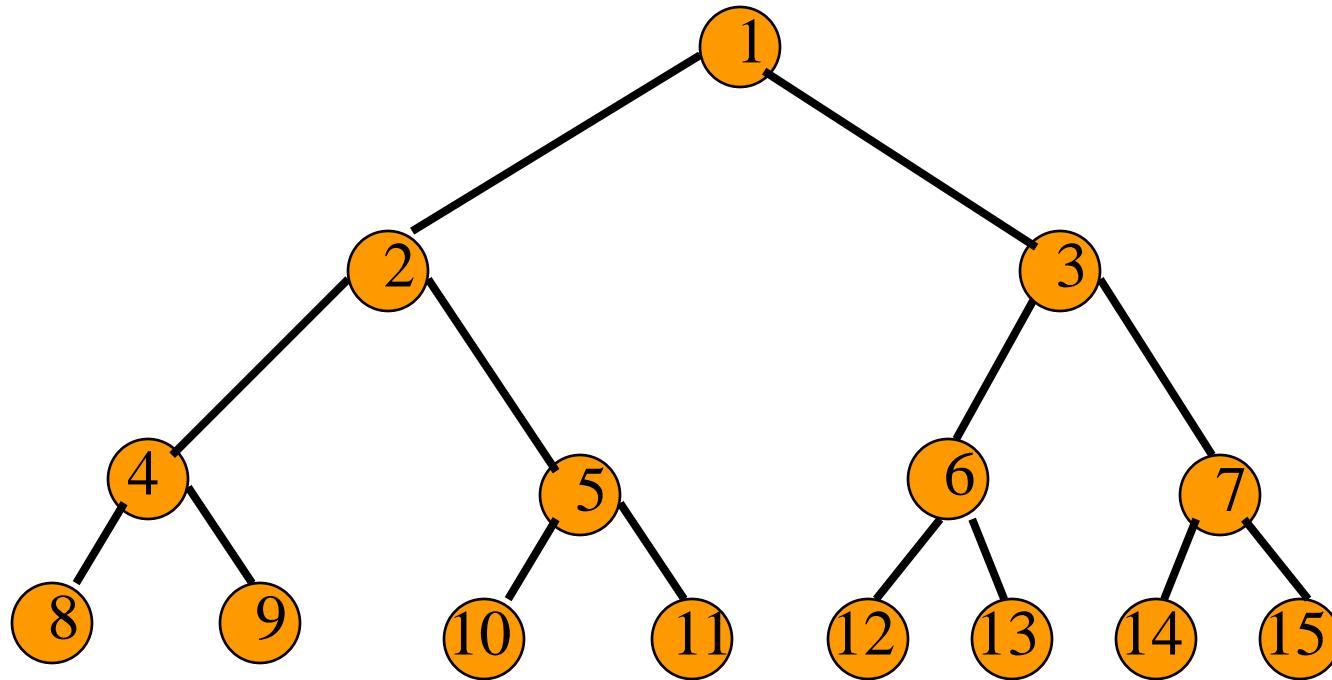
- Зангилаа  $i$  –ийн эцэг  $i / 2$ , ( $i \neq 1$ )
- Зангилаа 1 бол үндэс, эцэггүй.

# Зангилааны дугаарын шинж



- $2i > n$  биш бол зангилаа  $i$  –н зүүн хүү нь  $2i$ , үүнд  $n$  зангилааны тоо.
- Хэрвээ  $2i > n$ , зангилаа  $i$  зүүн хүүгүй.

# Зангилааны дугаарын шинж

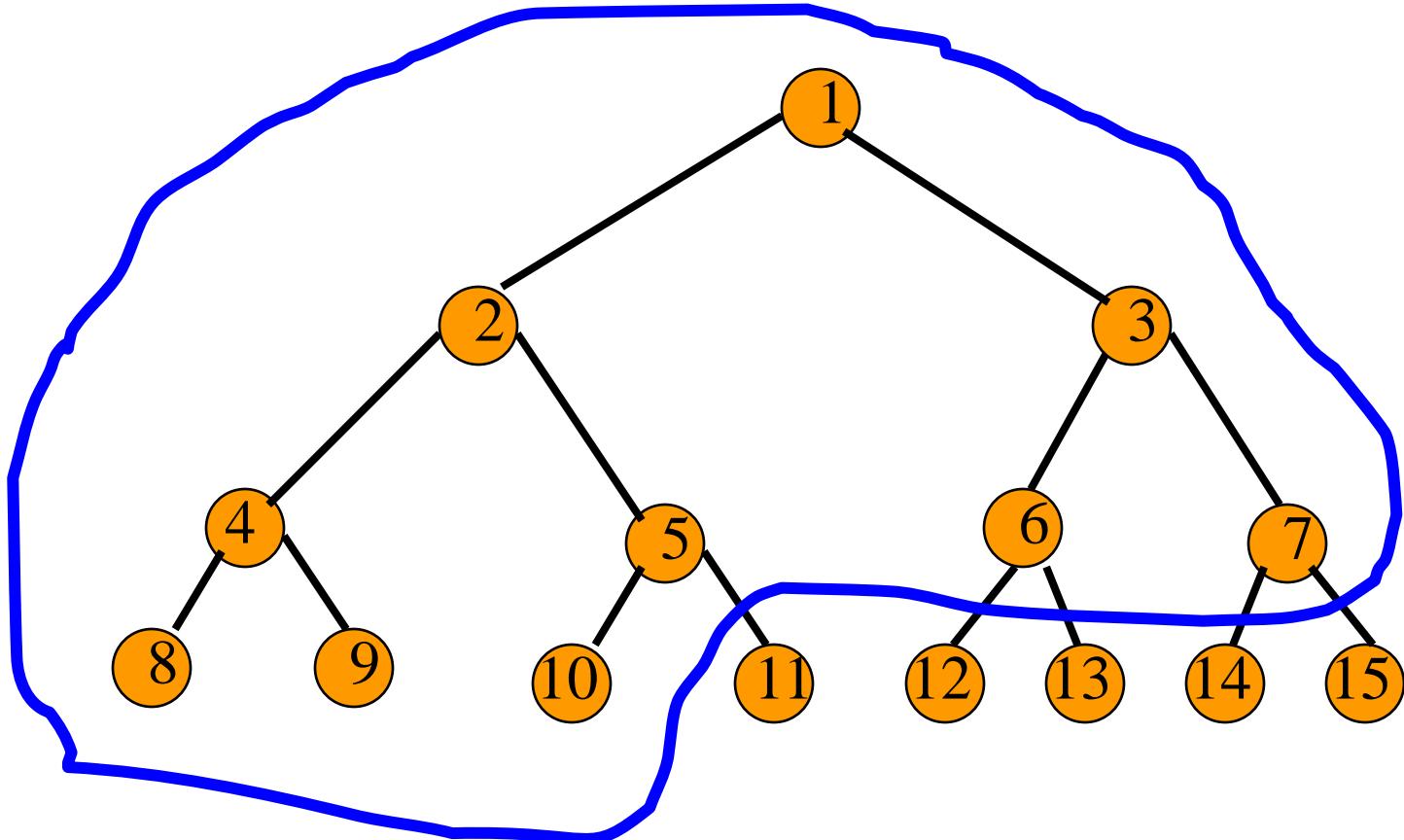


- $2i+1 > n$ , биш бол зангилаа  $i$  –н баруун хүү, Үүнд  $n$  зангилааны тоо.
- Хэрвээ  $2i+1 > n$ , зангилаа  $i$  баруун хүүгүй.

# n зангилаатай төгс хоёртын мод

- Дор хаяж n зангилаатай бүтэн модноос эхэл.
- Θмнө үзсэнээр зангилааг дугаарла.
- 1 –ээс n хүртэл дугаарлагдсан зангилаатай хоёртын модыг орь ганц n зангилаатай төгс хоёртын мод гэнэ.

# Жишээ



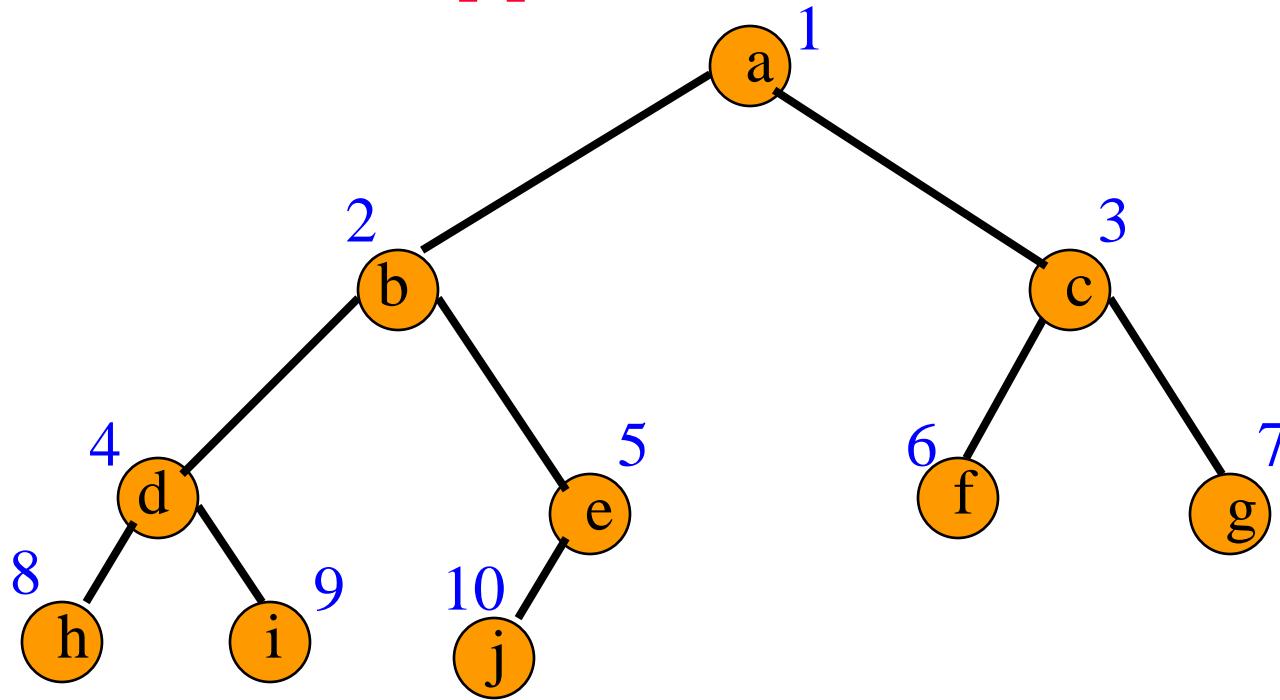
- 10 зангилаатай төгс хоёртын мод.

# Хоёртын модыг дүрслэх

- Массив дүрслэл.
- Холбоост дүрслэл.

# Массив дурслэл

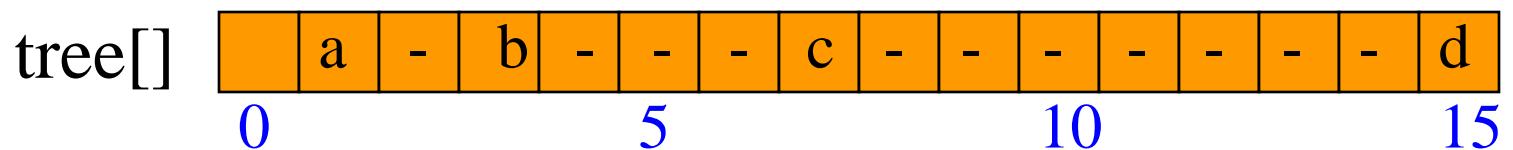
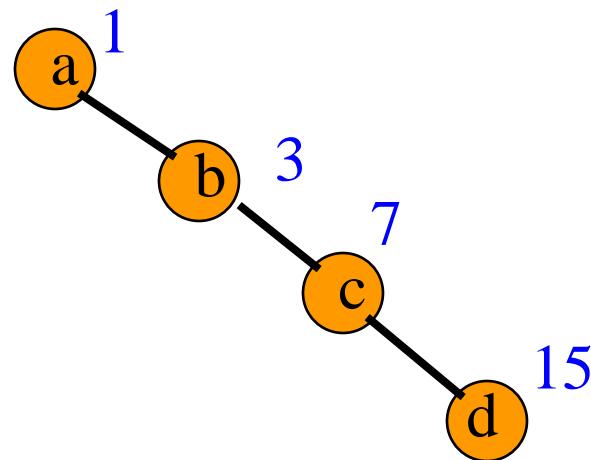
- Бүтэн хоёртын модыг дугаарлах схемээр зангилаануудыг дугаарла. *i* дугаартай зангилаа `tree[i]` –д хадгалагдана



`tree[]`

|   |   |   |   |   |   |   |   |   |   |    |
|---|---|---|---|---|---|---|---|---|---|----|
|   | a | b | c | d | e | f | g | h | i | j  |
| 0 |   |   |   |   | 5 |   |   |   |   | 10 |

# Баруун-хазайлттай хоёртын мод



- **n** зангилаатай хоёртын модонд шаардлагатай массивын урт **n+1** ба  $2^n$  хооронд байна

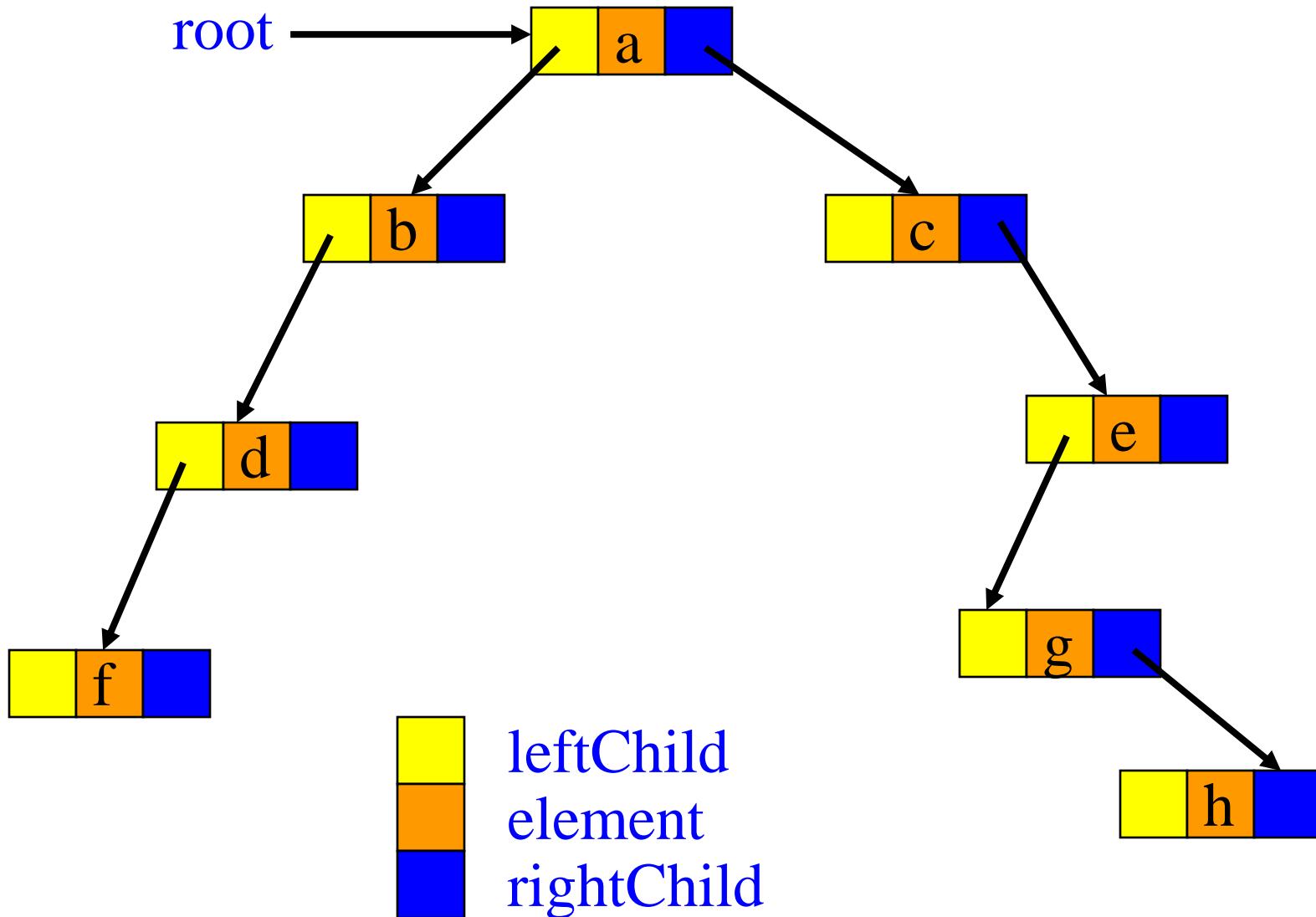
# Холбоост дүрслэл

- Хоёртын модны зангилаа бүр **BinaryTreeNode** гэсэн өгөгдлийн төрлийн объект байна.
- **n** зангилаатай хоёртын модонд шаардлагатай орон зай **n \* (нэг зангилааны орон зайд)**.

# BinaryTreeNode класс

```
package dataStructures;  
public class BinaryTreeNode  
{  
    Object element;  
    BinaryTreeNode leftChild; // зүүн дэд мөд  
    BinaryTreeNode rightChild; // баруун дэд мөд  
    // байгуулагч болон бусад аргууд  
    // энд бичигдэнэ  
}
```

# Холбоост дурслэлийн жишээ



# Хоёртын модны зарим үйлдлүүд

- Θндрийг олох.
- Зангилааны тоог олох.
- Хувилах.
- Хоёр хоёртын мод хувилагдсан эсэхийг тогтоох.
- Хоёртын модыг харуулах.
- Хоёртын modoор дүрслэгдсэн арифметик илэрхийллийг бодох.
- Илэрхийллийн infix хэлбэрийг гаргах.
- Илэрхийллийн prefix хэлбэрийг гаргах.
- Илэрхийллийн postfix хэлбэрийг гаргах.

# Хоёртын modoор нэвтрэх

- Ихэнх хоёртын модны үйлдлүүд нь хоёртын modoор **нэвтрэх** замаар хийгддэг.
- Аялахдаа хоёртын модны элемент бүрээр зөвхөн нэг удаа **зочилдог**.
- Элементээр **зочлохдоо** тэр элементтэй холбоотой үйлдэл (хувилах, харуулах, үйлдлийг бодох, гэх мэт.) хийгддэг.

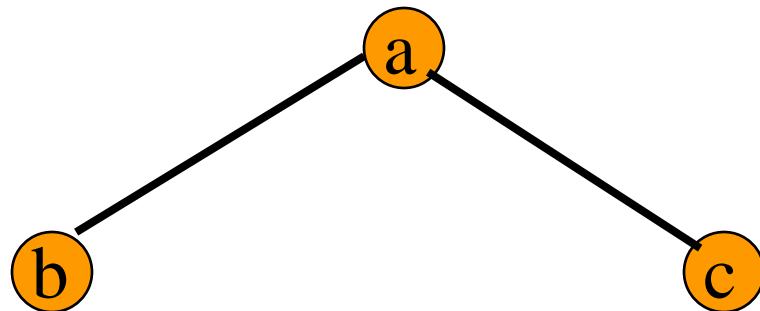
# Хоёртын modoор нэвтрэх аргууд

- Preorder
- Inorder
- Postorder
- Level order

# Preorder нэвтрэлт

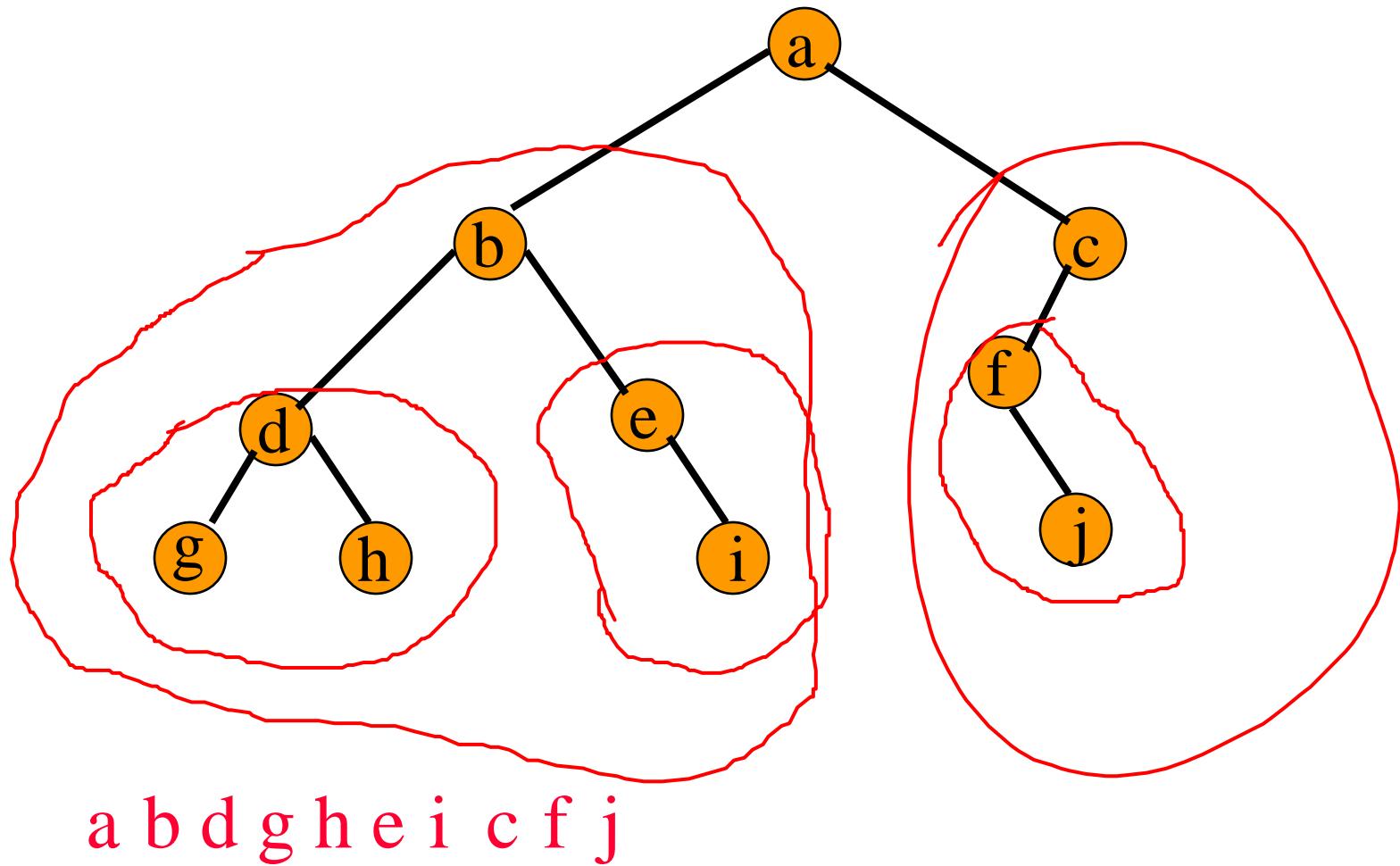
```
public static void preOrder(BinaryTreeNode t)
{
    if (t != null)
    {
        visit(t);
        preOrder(t.leftChild);
        preOrder(t.rightChild);
    }
}
```

# Preorder жишээ (visit = print)

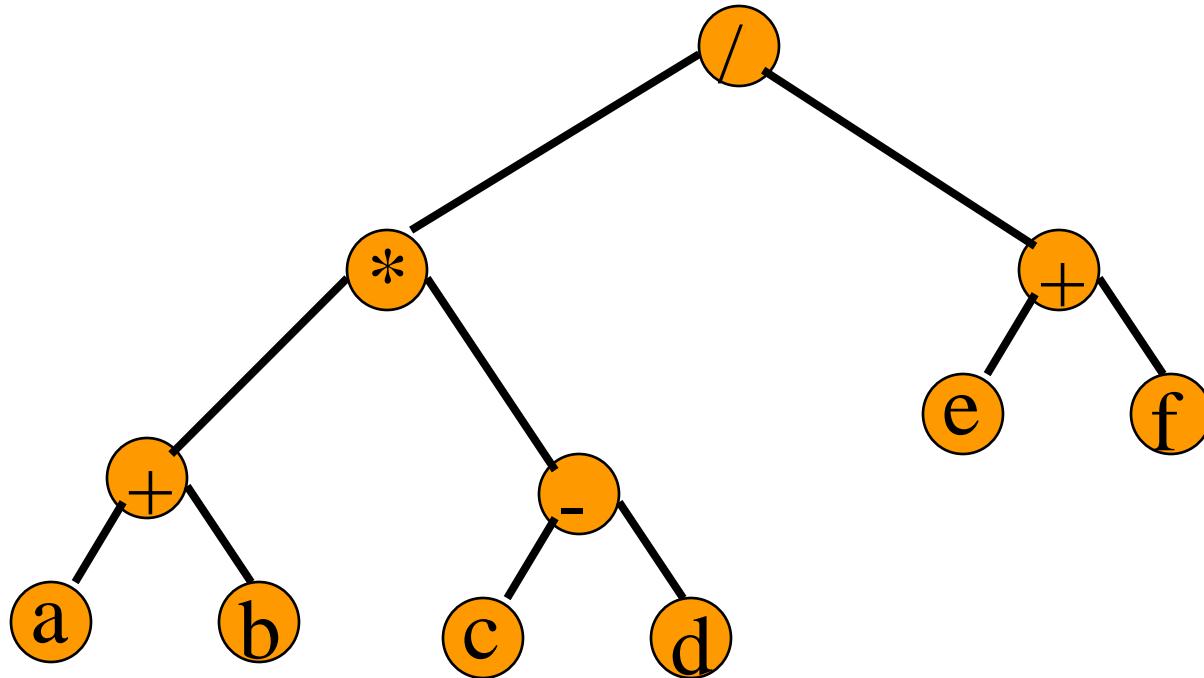


a b c

# Preorder жишээ (visit = print)



# Preorder илэрхийллийн мод



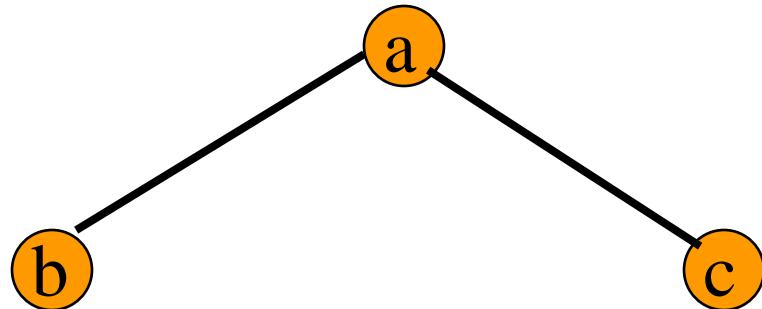
/ \* + a b - c d e f

Энэ мод илэрхийллийн prefix хэлбэрийг өгнө!

# Inorder нэвтрэлт

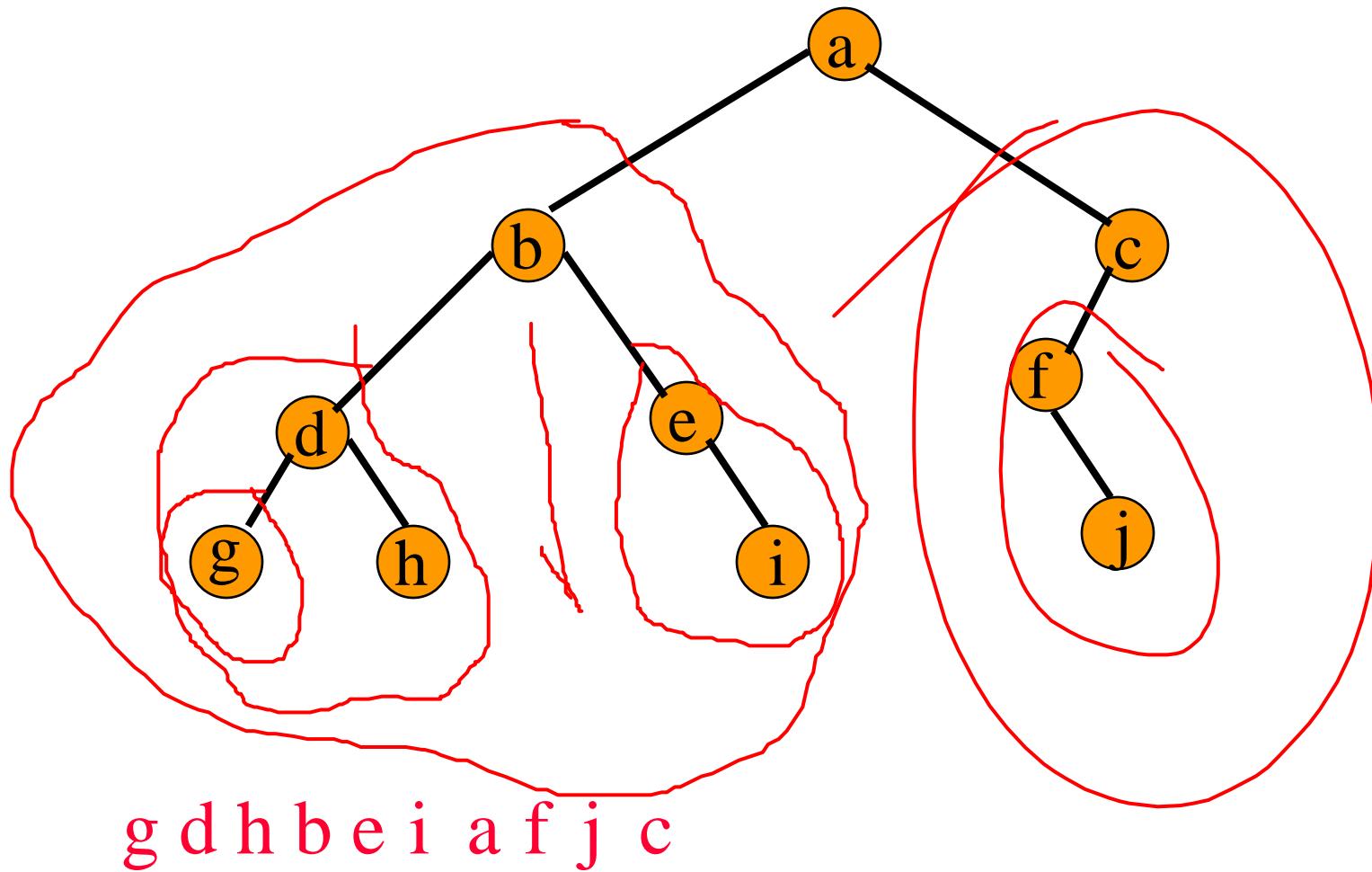
```
public static void inOrder(BinaryTreeNode t)
{
    if (t != null)
    {
        inOrder(t.leftChild);
        visit(t);
        inOrder(t.rightChild);
    }
}
```

# Inorder жишиээ (visit = print)

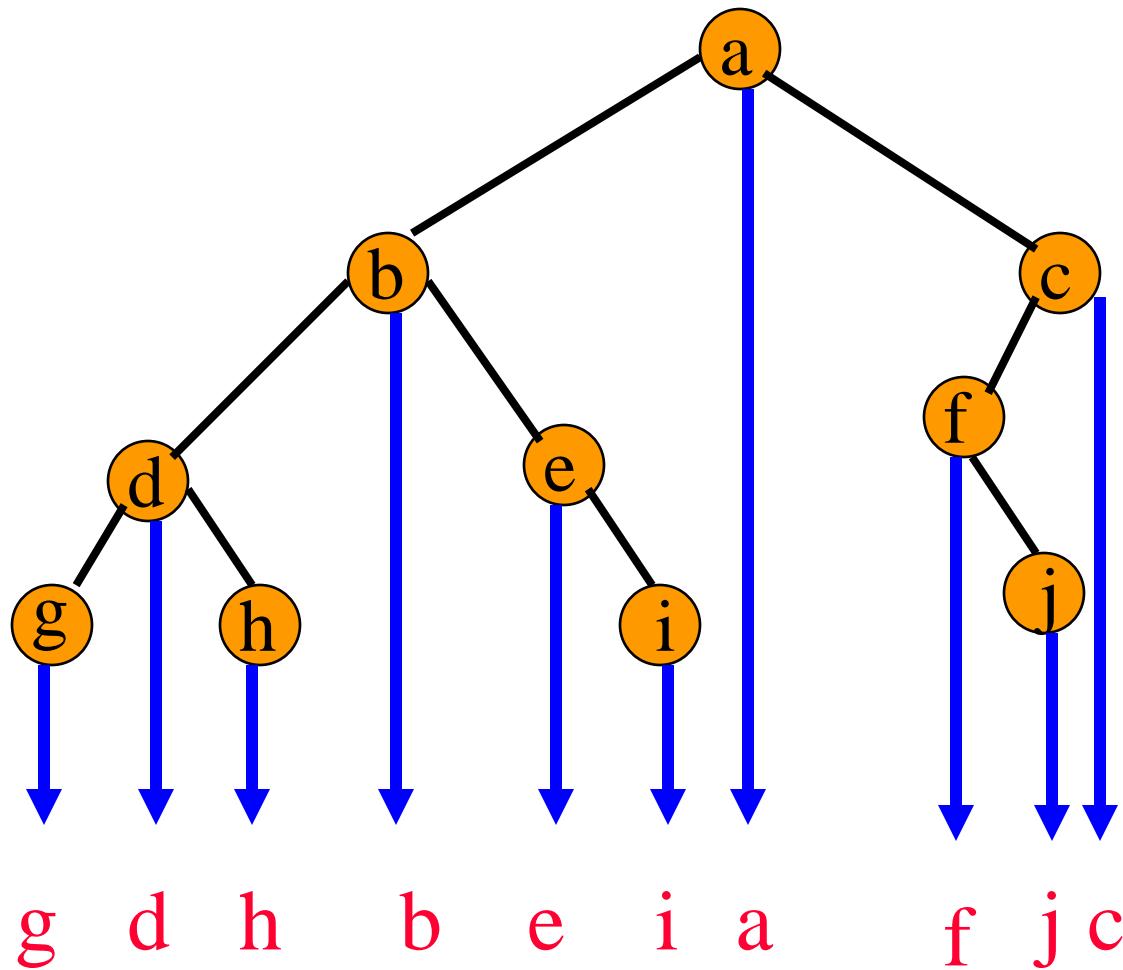


b a c

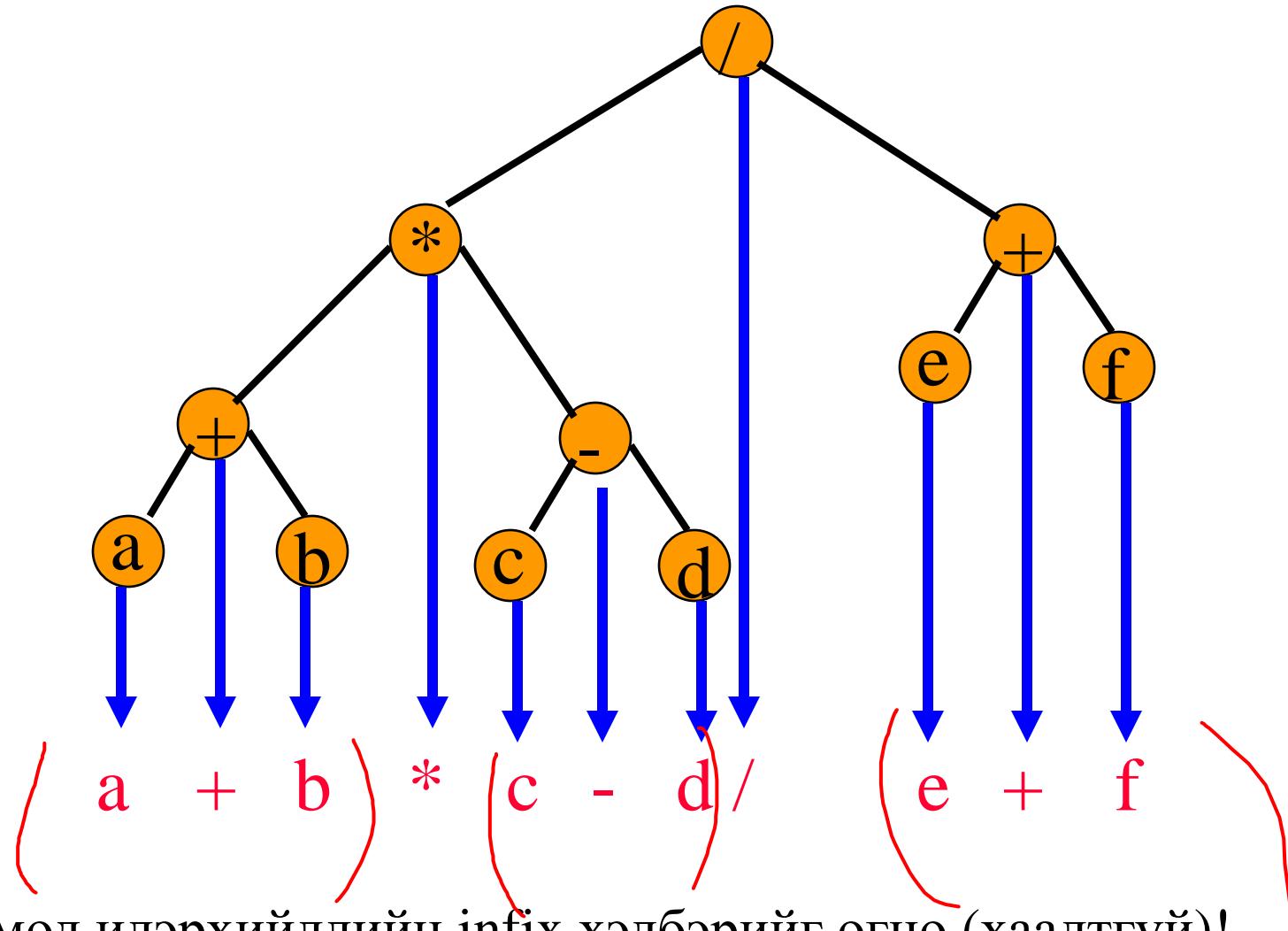
# Inorder жишиээ (visit = print)



# Inorder тусгалаар (Squishing)



# Inorder илэрхийллийн мод

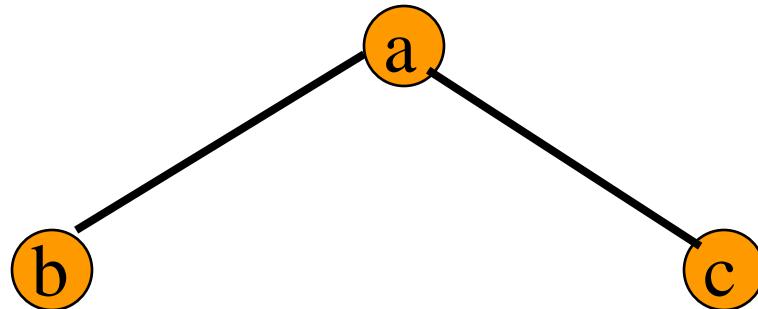


Энэ мод илэрхийллийн infix хэлбэрийг өгнө (хаалтгүй)!

# Postorder нэвтрэлт

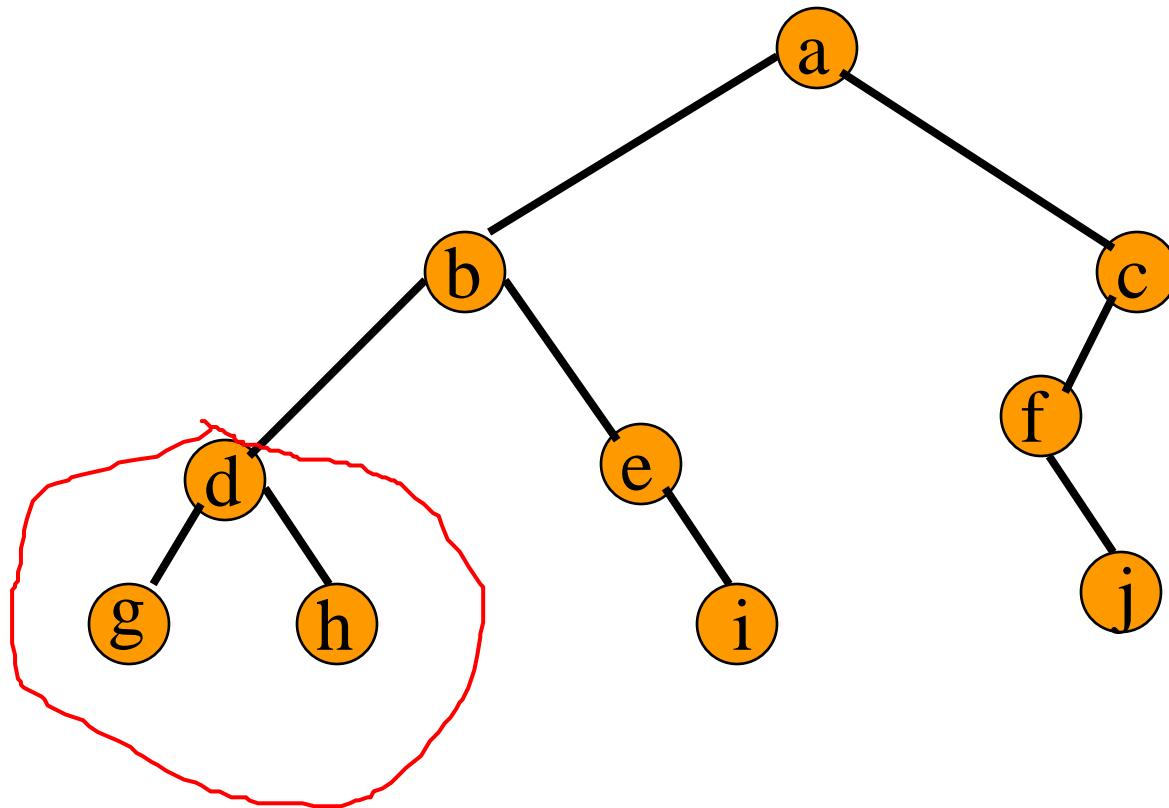
```
public static void postOrder(BinaryTreeNode t)
{
    if (t != null)
    {
        postOrder(t.leftChild);
        postOrder(t.rightChild);
        visit(t);
    }
}
```

# Postorder жишээ (visit = print)



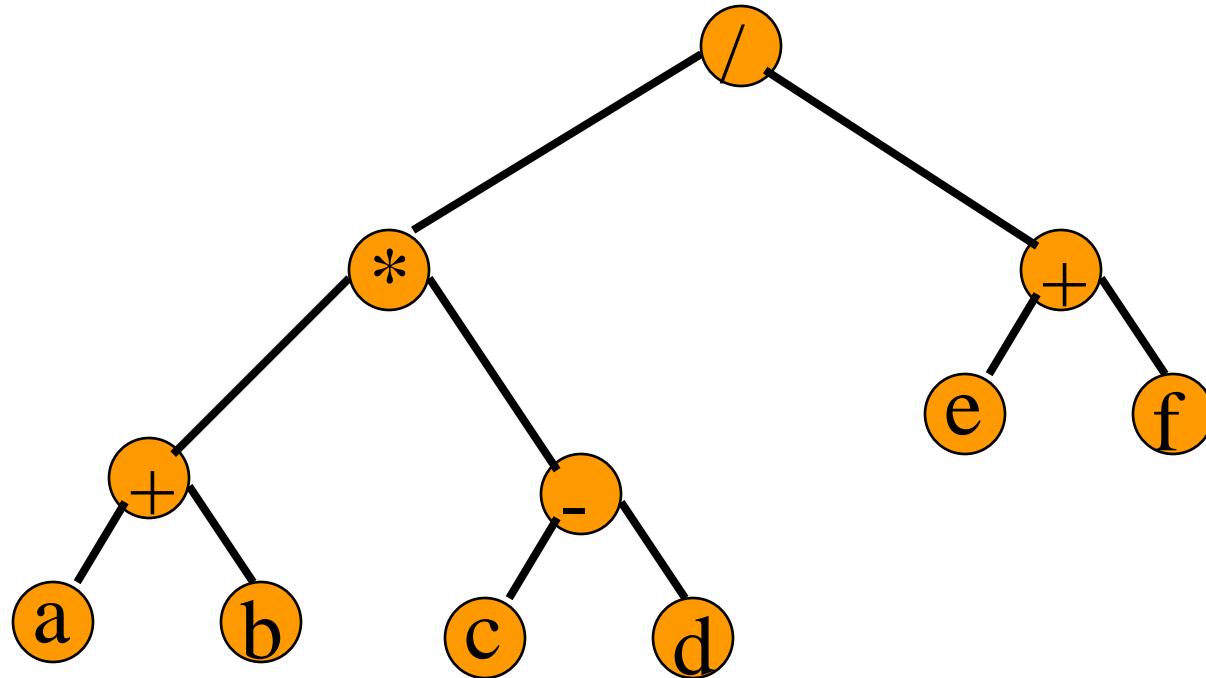
b c a

# Postorder жишиэ (visit = print)



g h d i e b j f c a

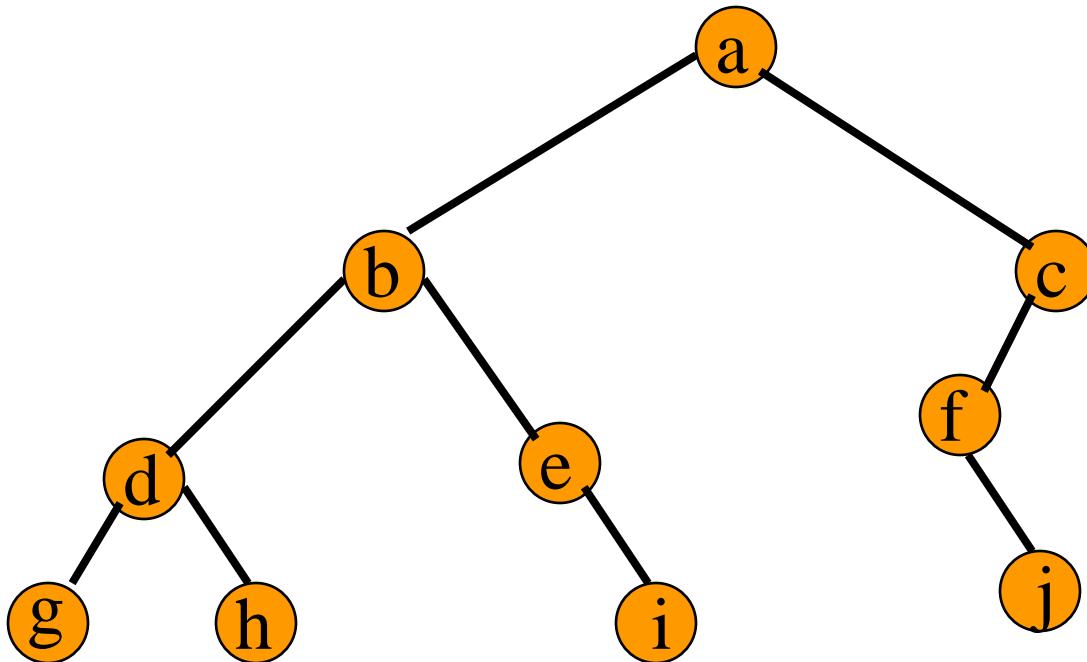
# Postorder илэрхийллийн мод



a b + c d - \* e f + /

Энэ мод илэрхийллийн postfix хэлбэрийг өгнө!

# Нэвтрэлтийн хэрэглээ



- Хувилах - clone.
- Θндрийг олох.
- Зангилааны тоог олох.

# LevelOrder нэвтрэлт

**t** модны үндэс.

**while (t != null)**

{

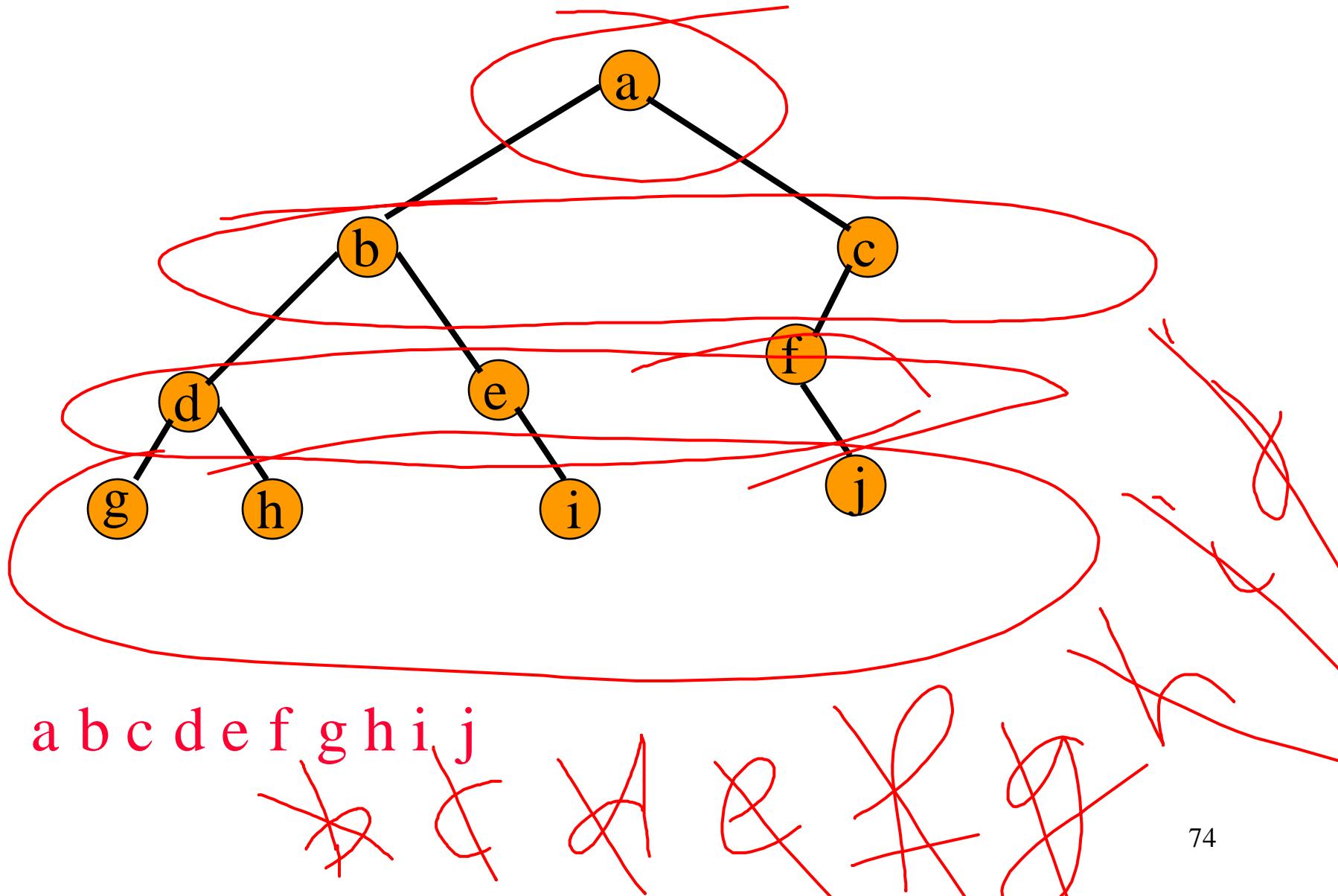
**t** –д зочлоод хүүхдүүдийг нь FIFO дараалалд хийнэ;

зангилааг FIFO дарааллаас устгаж, дуудна **t**;

**// дараалал хоосон бол устгал null –г буцаана**

}

# Level-Order жишиээ (visit = print)

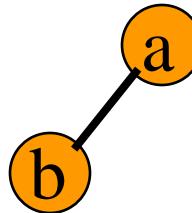


# Хоёртын модыг байгуулах

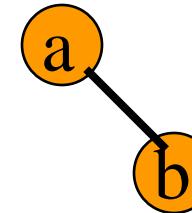
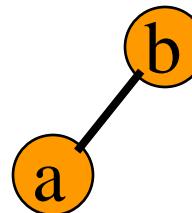
- Хоёртын модны элемент бүр ялгаатай гэж тооцьё.
- Өгөгдсөн нэвтрэлтийн дарааллаар хоёртын модыг байгуулж болох уу?
- Нэвтрэлтийн дараалалд нэгээс их элемент байгаа бол цорын ганц хоёртын мод байхгүй.
- Иймд гарсаж авсан дарааллаар яг тэр чигээр модыг сэргээх байгуулах боломжгүй.

# Зарим жишээ

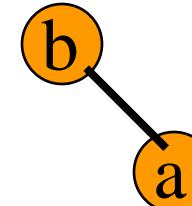
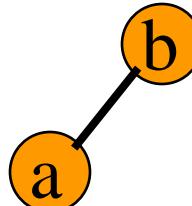
preorder  
= ab



inorder  
= ab



postorder  
= ab



level order  
= ab

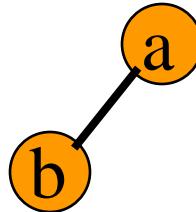


# Хоёртын модыг байгуулах

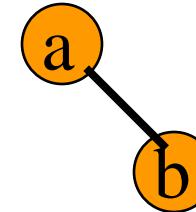
- Өгөгдсөн нэвтрэлтийн хоёр дарааллаар хоёртын модыг байгуулж болох уу?
- Ямар хоёр дараалал өгөгдсөнөөс хамаарна.

# Preorder ба Postorder

preorder = ab



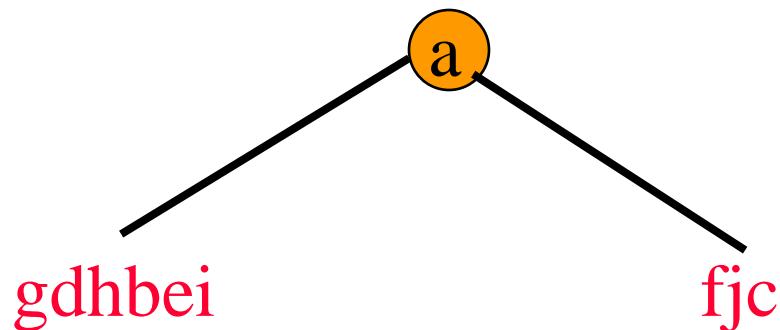
postorder = ba



- Preorder ба postorder давтагдашгүй хоёртын мөдүг тодорхойлж болохгүй.
- Preorder ба level order –оор болохгүй(дээрх жишээ).
- Postorder ба level order –оор болохгүй(дээрх жишээ).

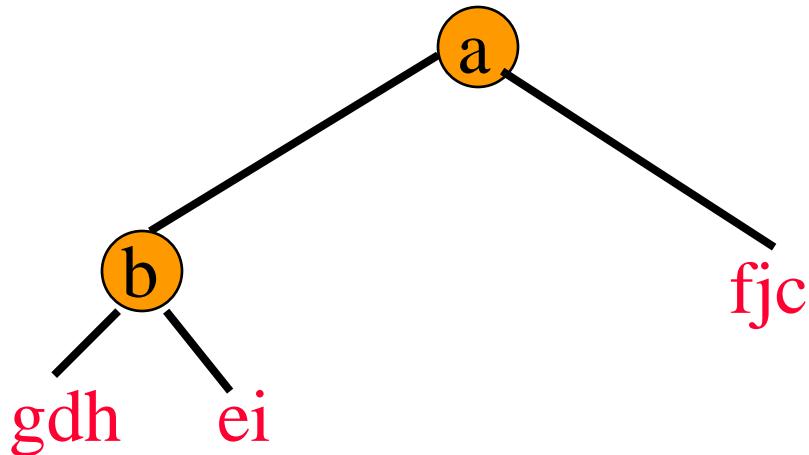
# Inorder ба Preorder

- inorder = ~~g d h b e i a f j c~~
- preorder = ~~a b d g h e i c f j~~
- preorder –г зүүнээс баруун тийш шинжэхдээ inorder –г ашиглаж зүүн, баруун дэд моднуудыг салгана.
- **a** бол үндэс; **gdhbei** зүүн дэд мод; **fjc** баруун дэд мод.

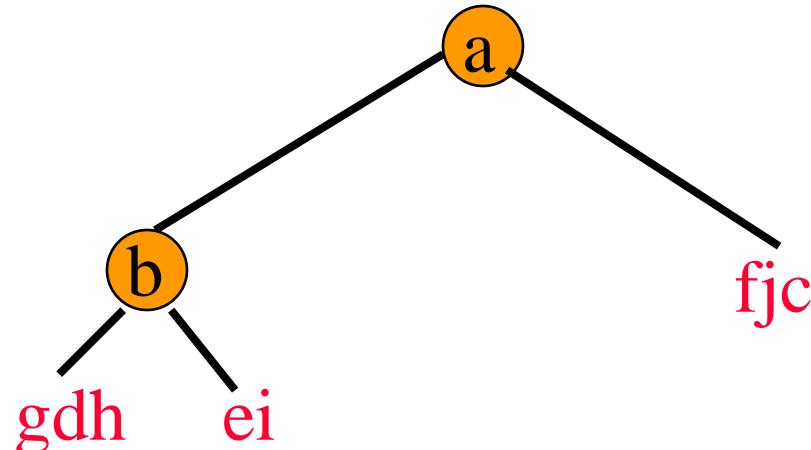


# Inorder ба Preorder

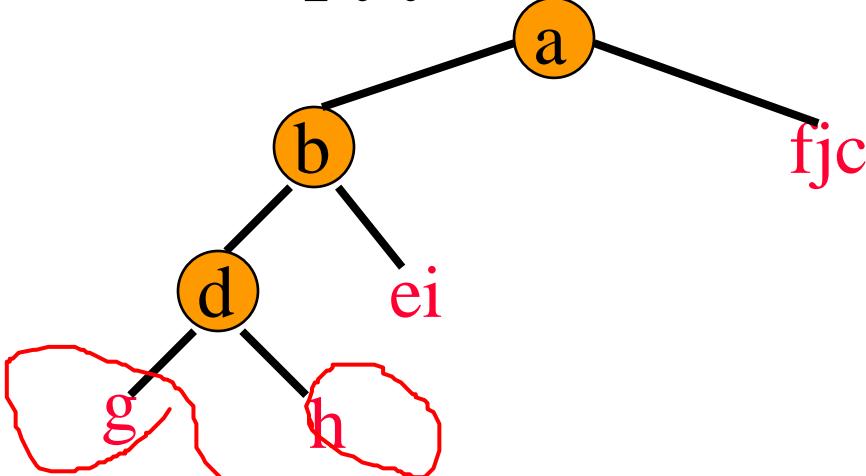
- 
- preorder = a **b** d g h e i c f j
  - **b** бол дараачийн үндэс; **gdh** зүүн дэд мөд; **ei** баруун дэд мөд.



# Inorder ба Preorder



- preorder = a b d g h e i c f j
- **d** бол дараачийн үндэс; **g** зүүн дэд мод; **h** баруун дэд мод.



# Inorder ба Postorder

- postorder –г баруунаас зүүн тийш шинжэхдээ inorder –г ашиглаж зүүн, баруун дэд моднуудыг салгана.
- inorder = g d h b e i a f j c
- postorder = g h d i e b j f c a
- a модны үндэс; gdhbei зүүн дэд мод; fjc баруун дэд мод.

# Inorder ба Level Order

- level order –г зүүнээс баруун тийш шүүрдэхдээ inorder –г ашиглаж зүүн, баруун дэд моднуудыг салгана.
- inorder = g d h b e i **a** f j c
- level order = **a** b c d e f g h i j
- Модны үндэс **a**; **gdhbei** зүүн дэд мод; **fjc** баруун дэд мод.



## Хоёр төрлийн эрэмбэтэй дараалал байдаг:

- Минимум эрэмбэтэй дараалал.
- Максимум эрэмбэтэй дараалал.

# Минимум эрэмбэтэй дараалал

- Элементүүдийн цуглуулга.
- Элемент бүр эрэмбэ буюу түлхүүртэй.
- Хийгдэх үйлдлүүд:
  - isEmpty
  - size
  - add/put (эрэмбэтэй дараалал элемент оруулах)
  - get (min эрэмбэтэй элементийг авах)
  - remove (min эрэмбэтэй элементийг устгах)

# Максимум эрэмбэтэй дараалал

- Элементүүдийн цуглуулга.
- Элемент бүр эрэмбэ буюу түлхүүртэй.
- Хийгдэх үйлдлүүд:
  - isEmpty
  - size
  - add/put (эрэмбэтэй дараалал элемент оруулах)
  - get (**max** эрэмбэтэй элементийг авах)
  - remove (**max** эрэмбэтэй элементийг устгах)

# Үйлдлийг гүйцэтгэх хугацаа

Сайн хэрэгжүүлэлт нь heap-овоолго , leftist tree-зүүний мод.

isEmpty, size, get => O(1)

put , remove => O(log n) , Үүнд n –  
эрэмбэтэй дарааллын хэмжээ

# Хэрэглээ

## Эрэмбэлэлт

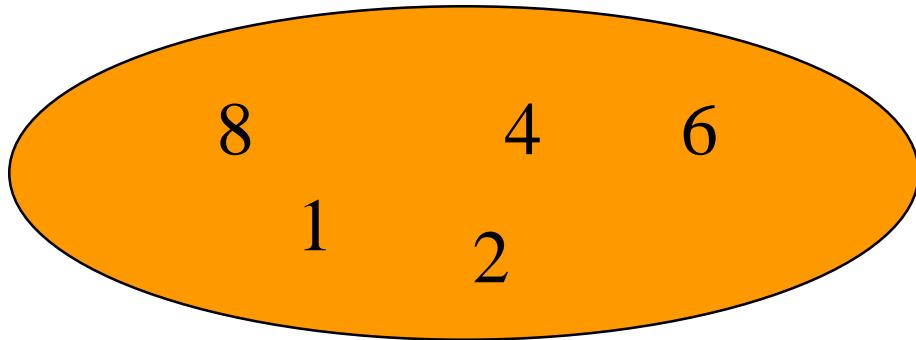
- Элементийн түлхүүрийг эрэмбэ болгон ашиглах
- Эрэмбэлэх элементүүдийг эрэмбэтэй дараалалд хийх
- Эрэмбийн дараалаар элементийг гаргаж авах
  - Хэрвээ **min** эрэмбэтэй дараалал бол, элементүүдийг эрэмбийн(түлхүүрийн) өсөх дарааллаар гаргана
  - Хэрвээ **max** эрэмбэтэй дараалал бол, элементүүдийг эрэмбийн(түлхүүрийн) буурах дарааллаар гаргана

# Эрэмбэлэлтийн жишээ

6, 8, 2, 4, 1 гэсэн түлхүүртэй таван  
элементийг тах эрэмбэтэй дарааллын  
аргаар эрэмбэлэе.

- Таван элементийг тах эрэмбэтэй дараалалд хийнэ.
- remove тах үйлдлийг таван удаа гүйцэтгэхдээ устгагдсан элементүүдийг эрэмбэлэсэн массивт варуунаас зүүн тийш хийнэ.

# Max эрэмбэтэй дараалалд хийсний дараа

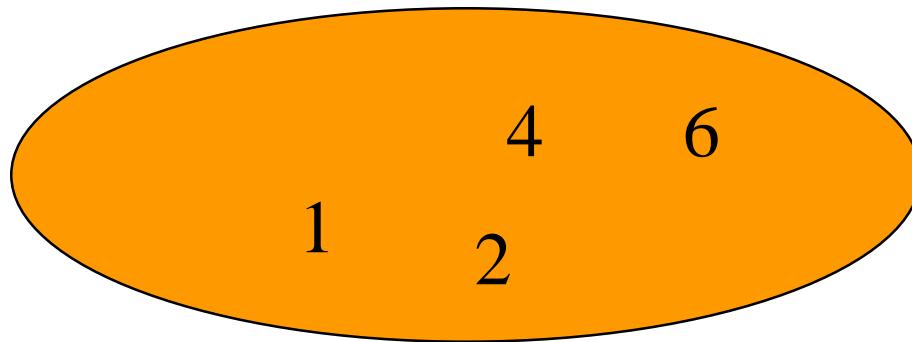


Max  
эрэмбэтэй  
дараалал



Эрэмбэлэгдсэн массив

# Эхний Remove Max үйлдлийн дараа

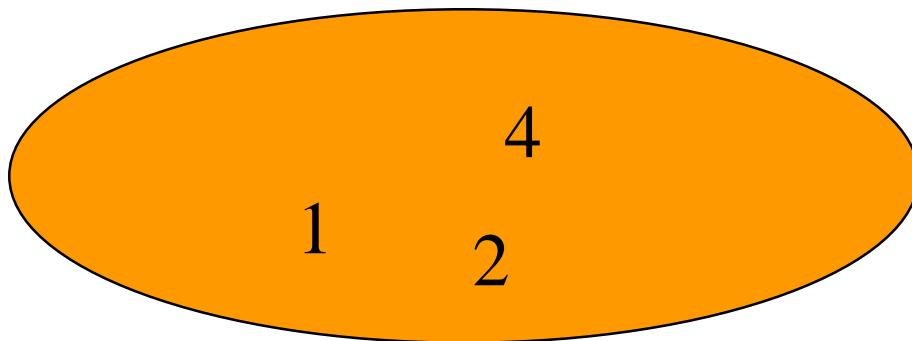


Max  
эрэмбэтэй  
дараалал



Эрэмбэлэгдсэн массив

# Хоёр дахь Remove Max үйлдлийн дараа

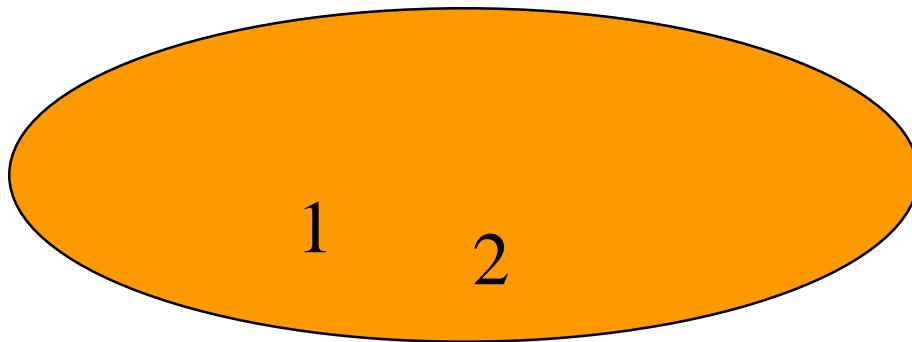


Max  
эрэмбэтэй  
дараалал



Эрэмбэлэгдсэн массив

# Гурав дахь Remove Max үйлдлийн дараа

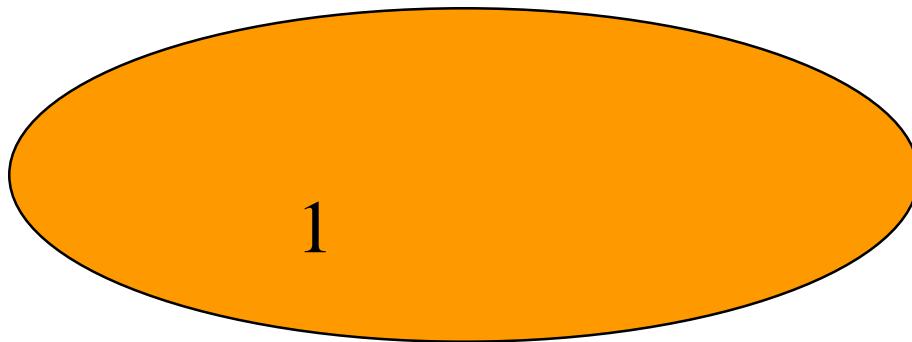


Max  
эрэмбэтэй  
дараалал



Эрэмбэлэгдсэн массив

# Дөрөв дэх Remove Max үйлдлийн дараа

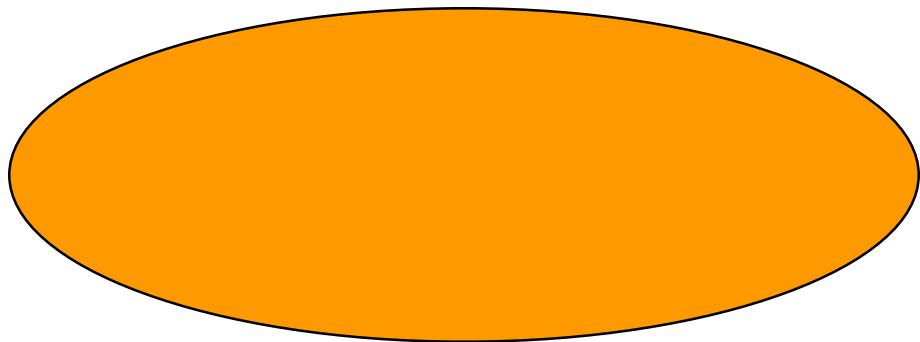


Max  
эрэмбэтэй  
дараалал



Эрэмбэтэй массив

# Тав дахь Remove Max үйлдлийн дараа



Max  
эрэмбэтэй  
дараалал

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|

Эрэмбэлэгдсэн массив

# Эрэмбэлэлтийн хугацаа

**n** элементийн эрэмбэлэлт.

- **n** put үйлдэл =>  $O(n \log n)$ .
- **n** remove max үйлдэл =>  $O(n \log n)$ .
- Нийт хугацаа  $O(n \log n)$ .
- 2-р бүлэгт үзсэн эрэмбэлэлтийн аргаар  $O(n^2)$ .

# Овоолго(Heap)-ын Эрэмбэлэлт

Овоолгыг хэрэгжүүлэхдээ тах эрэмбэтэй дараалыг ашиглана.

Эхний рүт үйлдлүүдийг овоолгыг идэвхижүүлэх алхмаар орлуулахад **O(n)** хугацаа шаардана.

# Машины төлөвлөлт

- $m$  ижил төрлийн машин (өрөм, зүсэгч, г.м.)
- $n$  ажил гүйцэтгэх хэрэгтэй
- Машинуудыг ажлаар ачаалахдаа сүүлийн ажлыг гүйцэтгэх хугацааг минимум болгох

# Машины төлөвлөлтийн жишээ

3 машин, 7 ажил

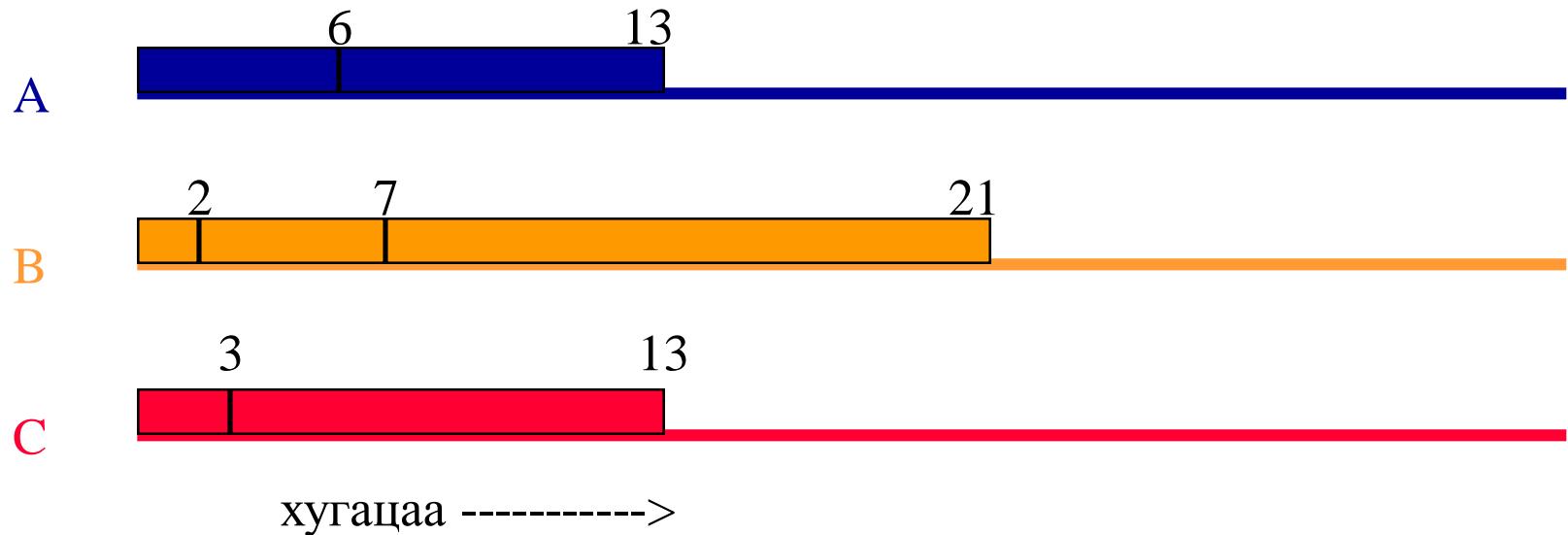
Ажлын хугацаа [6, 2, 3, 5, 10, 7, 14]

Боломжит төлөвлөлт:



хугацаа ----->

# Машины төлөвлөлтийн жишээ



Дуусах хугацаа = 21

Зорилго: Дуусах хугацааг минимум болгох.

# LPT Төлөвлөлт

Longest Processing Time - эхлээд.

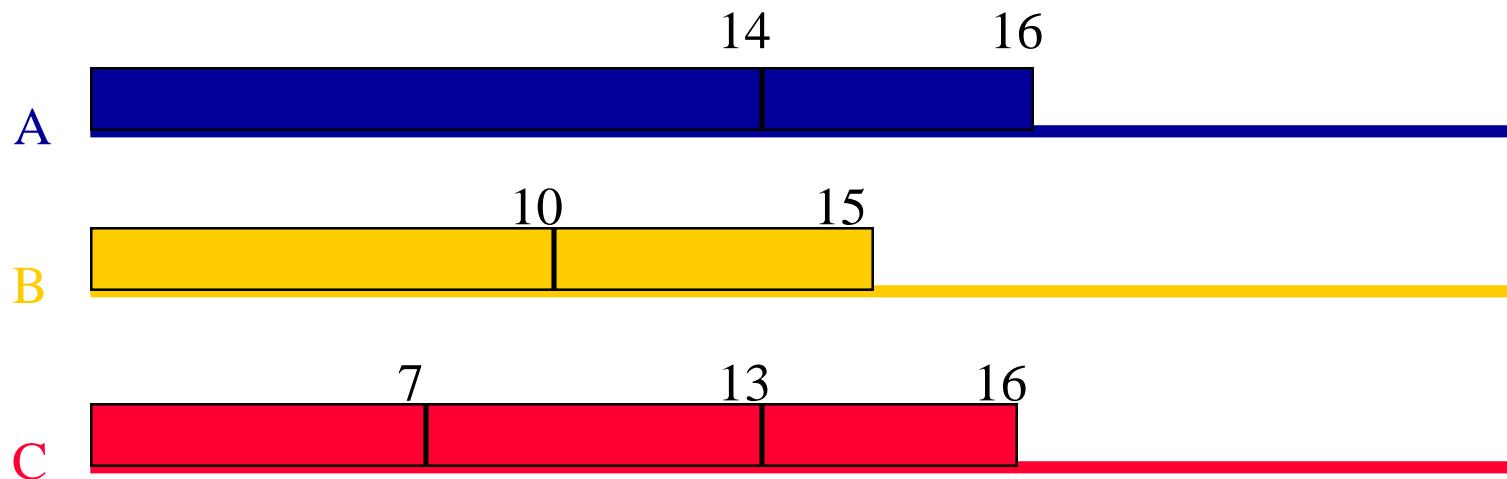
Ажлууд дараах байдлаар эрэмбэлэгджээ

14, 10, 7, 6, 5, 3, 2

Ажил бүрийг түрүүлж дуусгасан машин  
дээр хуваарилна.

# LPT төлөвлөлт

[14, 10, 7, 6, 5, 3, 2]



Дуусах хугацаа 16!

# LPT төлөвлөлт

- LPT дүрэм баталгаат минимум дуусах хугацааг өгч чадахгүй.
- $(LPT \text{ дуусах хугацаа}) / (Min \text{ дуусах хугацаа}) \leq 4/3 - 1/(3m)$   
Үүнд **m** - машины тоо.
- Ерөнхийдөө LPT дуусах хугацаа, min дуусах хугацаад ойрхон очдог .
- Min дуусах хагацаатай төлөвлөлтийг NP (nondeterministic polynomial)-hard өгдөг.

# LPT төлөвлөлтийн хугацаа

- Ажлуудыг хугацаа буурах дарааллаар эрэмбэлнэ.
  - $O(n \log n)$  хугацаа ( $n$  ажлын тоо)
- Ажлуудыг энэ дарааллаар хуваарилна.
  - Ажлыг түрүүлж бэлэн болсон машинд хуваарилна
  - $m$  ( $m$  машины тоо) дуусах хугацаанаас minimum-г олох ёстой
  - Энгийн стратеги хэрэглэхэд  $O(m)$  хугацаа зарна
  - Бүх  $n$  ажлыг төлөвлөхөд  $O(mn)$  хугацаа орно.

# Min эрэмбэтэй дарааллыг ашиглах

- Min эрэмбэтэй дараалалд **m** машины дуусах хугацаа байна.
- Эхлээд бүх дуусах хугацаа **0**.
- Ажлыг төлөвлөхийн тулд эрэмбэтэй дарааллаас минимум дуусах хугацаатай машиныг устгана.
- Сонгосон машины дуусах хугацааг өөрчлөөд буцаагаад эрэмбэтэй дараалалд хийнэ.

# Min эрэмбэтэй дарааллыг ашиглах

- $m$  put үйлдлээр эрэмбэтэй дарааллыг идэвхижүүлнэ
- 1 remove min , 1 put үйлдлүүдийг хуваарилагдсан ажил бүрт хийнэ
- put , remove min үйлдэл бүрт  $O(\log m)$  хугацаа орно
- Төлөвлөх хугацаа  $O(n \log m)$
- Нийт хугацаа
$$O(n \log n + n \log m) = O(n \log (mn))$$

# Хофманы код

Хаягдалгүй шахаад тустай.

LZW аргатай хослуулж болно.

Номноос унш.

Товч тайлбар:

{ a, x, u, z }

a-0, x-10, u-110, z-111 (давтамжаар хувьсах урттай код)

аахуаxз (7 байт) -> 0010110010111 (13 бит)

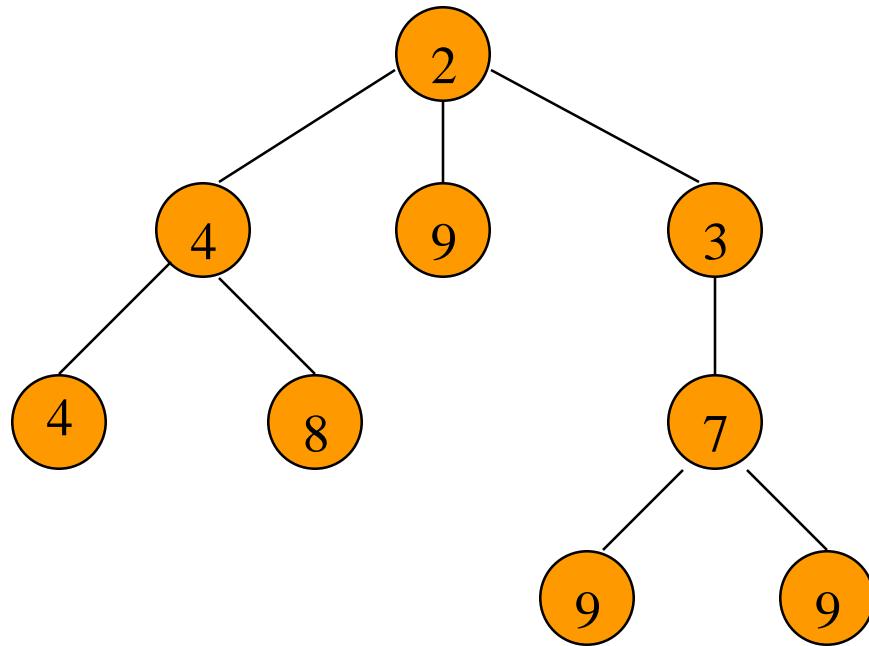
## Min модны тодорхойлолт

Модны зангилаа бүр утгатай.

Аливаа зангилааг үндэс гэж үзвэл утга нь  
дэд МОДОНДОО минимум байна.

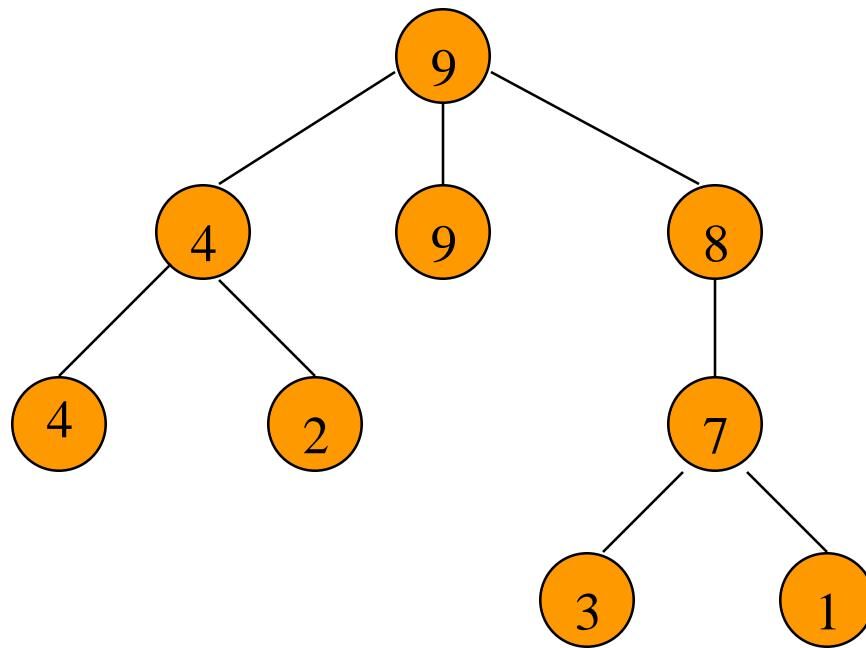
Өөрөөр хэлбэл бага утгатай хойч байхгүй.

# Min модны жишээ



Үндэс минимум элементтэй.

# Мах модны жишээ

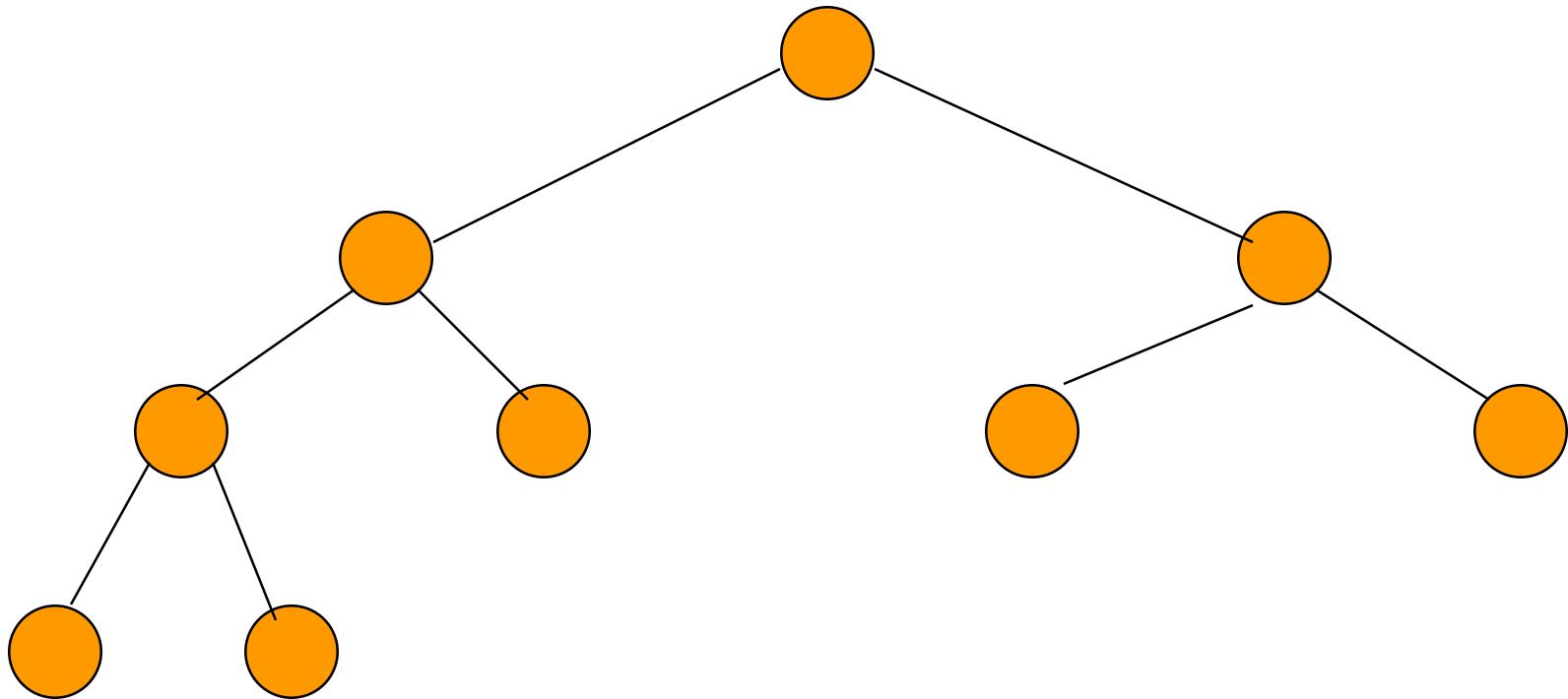


Үндэс максимум элементтэй.

# Min овоолгын тодорхойлолт

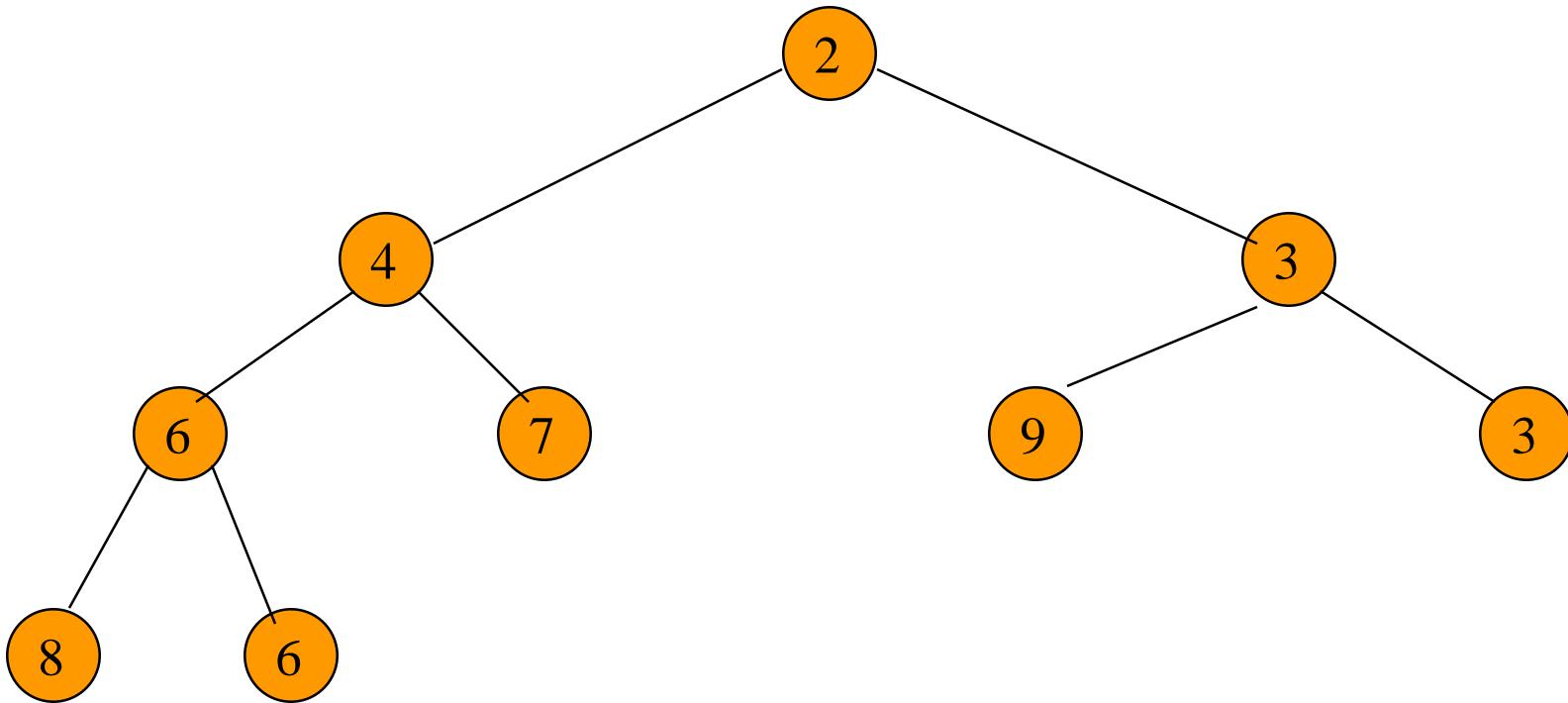
- Төгс хоёртын мод
- min мод

# 9 зангилаатай Min овоолго



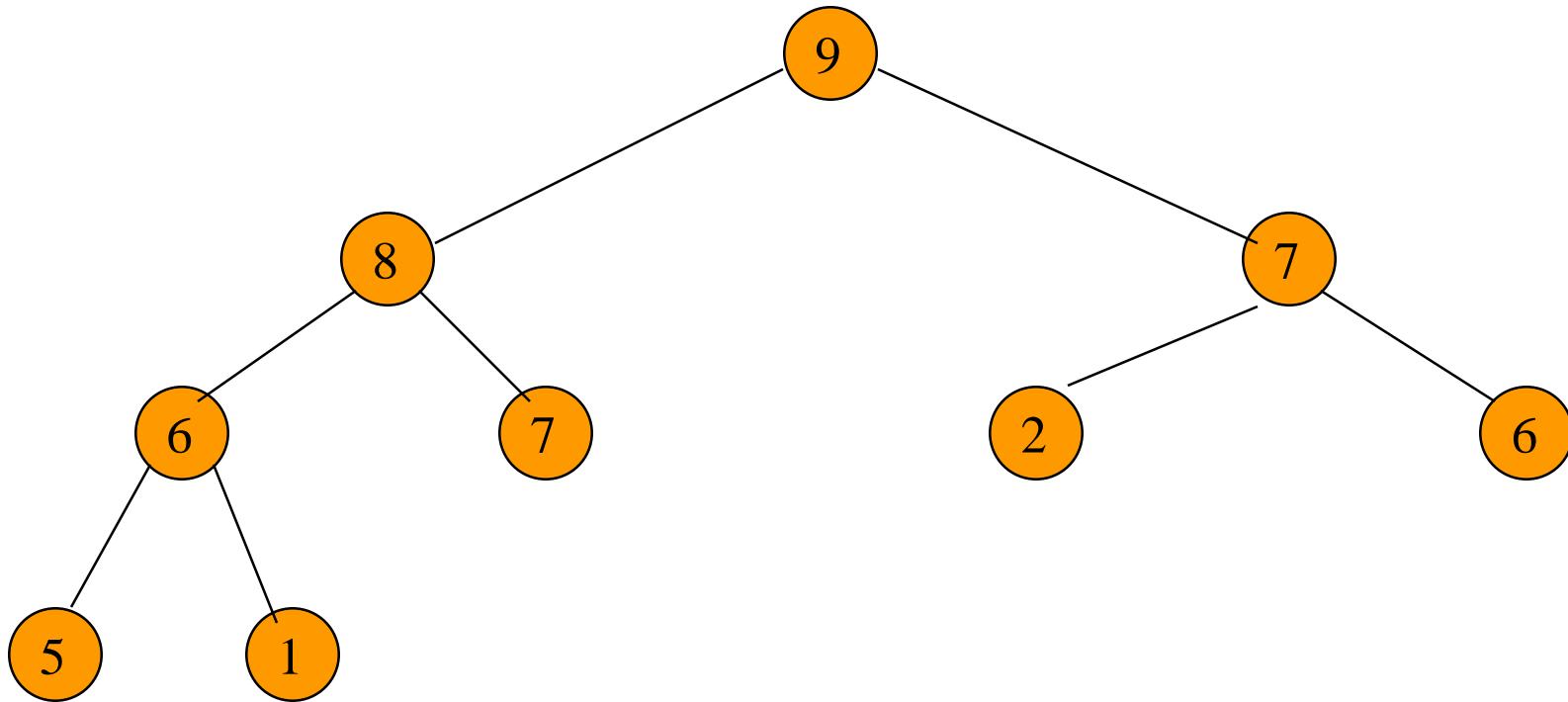
9 зангилаатай төгс хоёртын мод.

# 9 зангилаатай Min овоолго



9 зангилаатай төгс хоёртын мод  
мөн min мод.

# 9 зангилаатай Мах овоолго

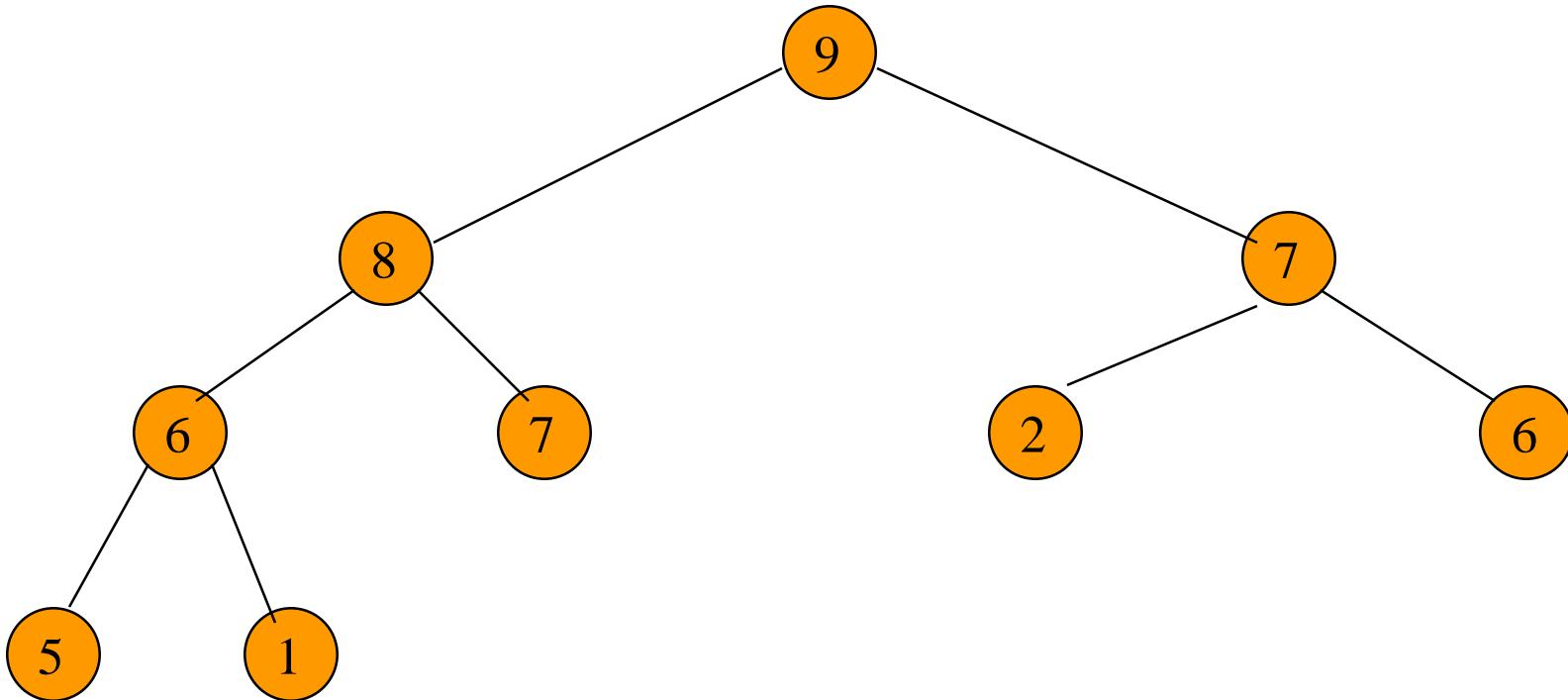


9 зангилаатай төгс хоёртын мод  
мөн max мод.

# Овоолгын өндөр

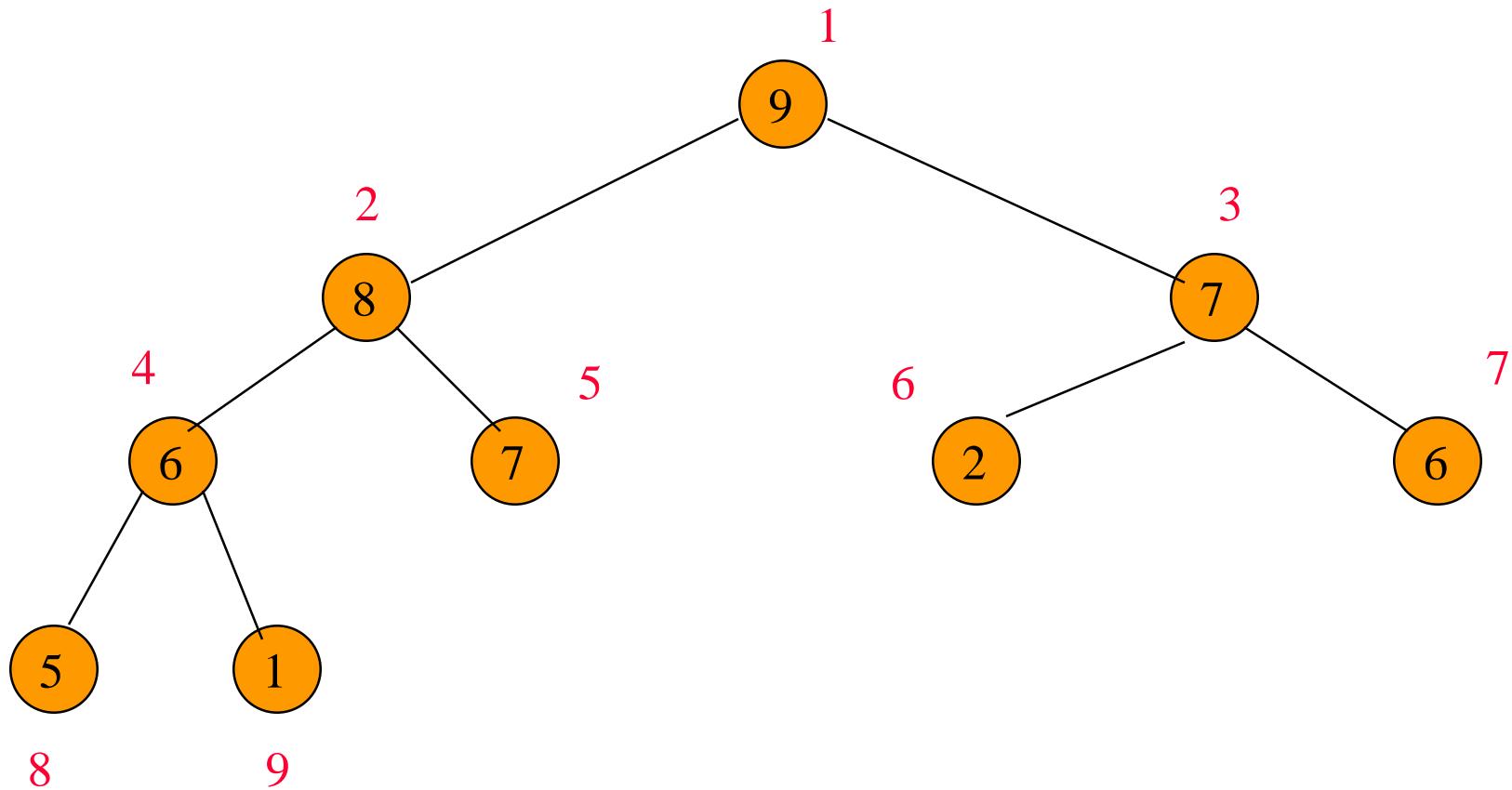
Нэгэнт овоолго нь төгс хоёртын  
мод болохоор  $n$  зангилаатай  
овоолгын өндөр  $\log_2(n+1)$ .

Овоолгыг массиваар оновчтой дүрсэлж болно

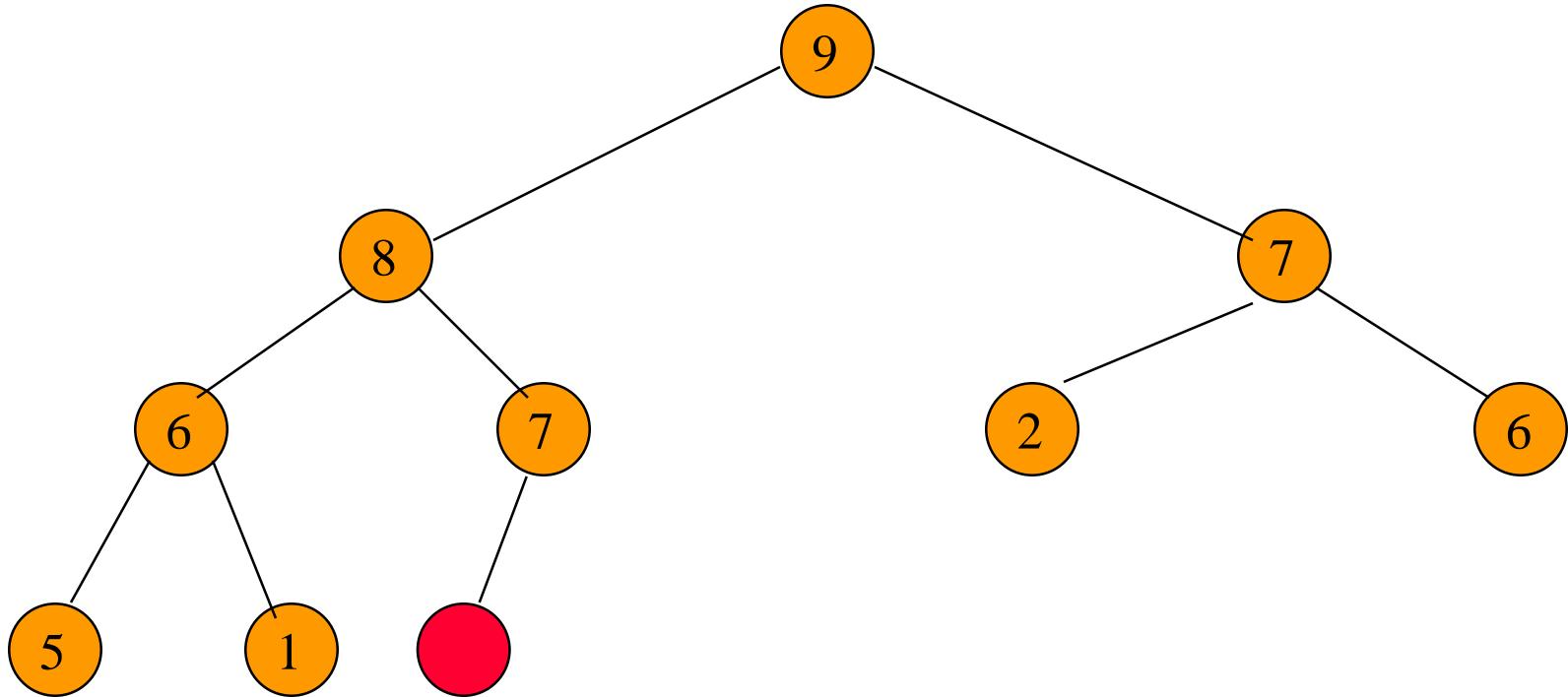


0 1 2 3 4 5 6 7 8 9 10

# Овоолгоор дээш, доош явах

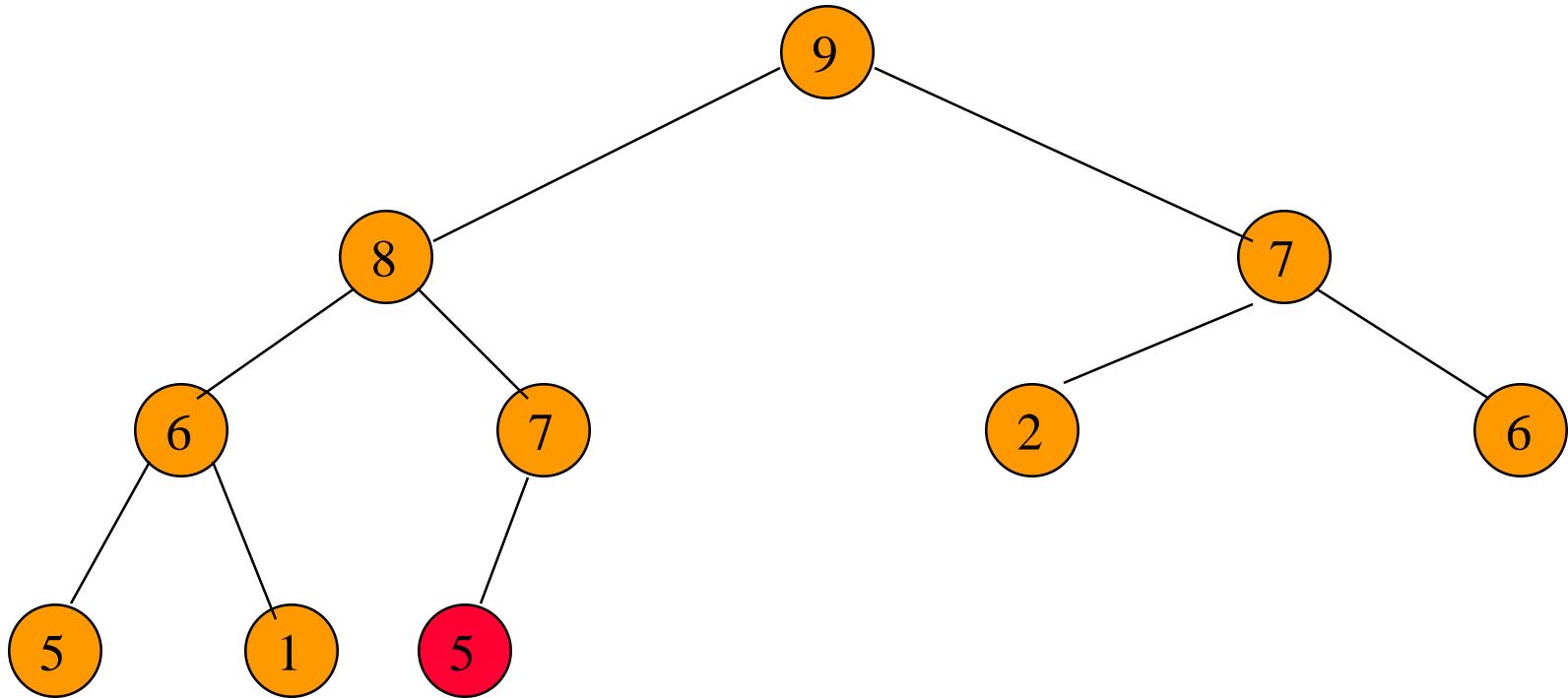


# Мах овоолгод элемент хийх



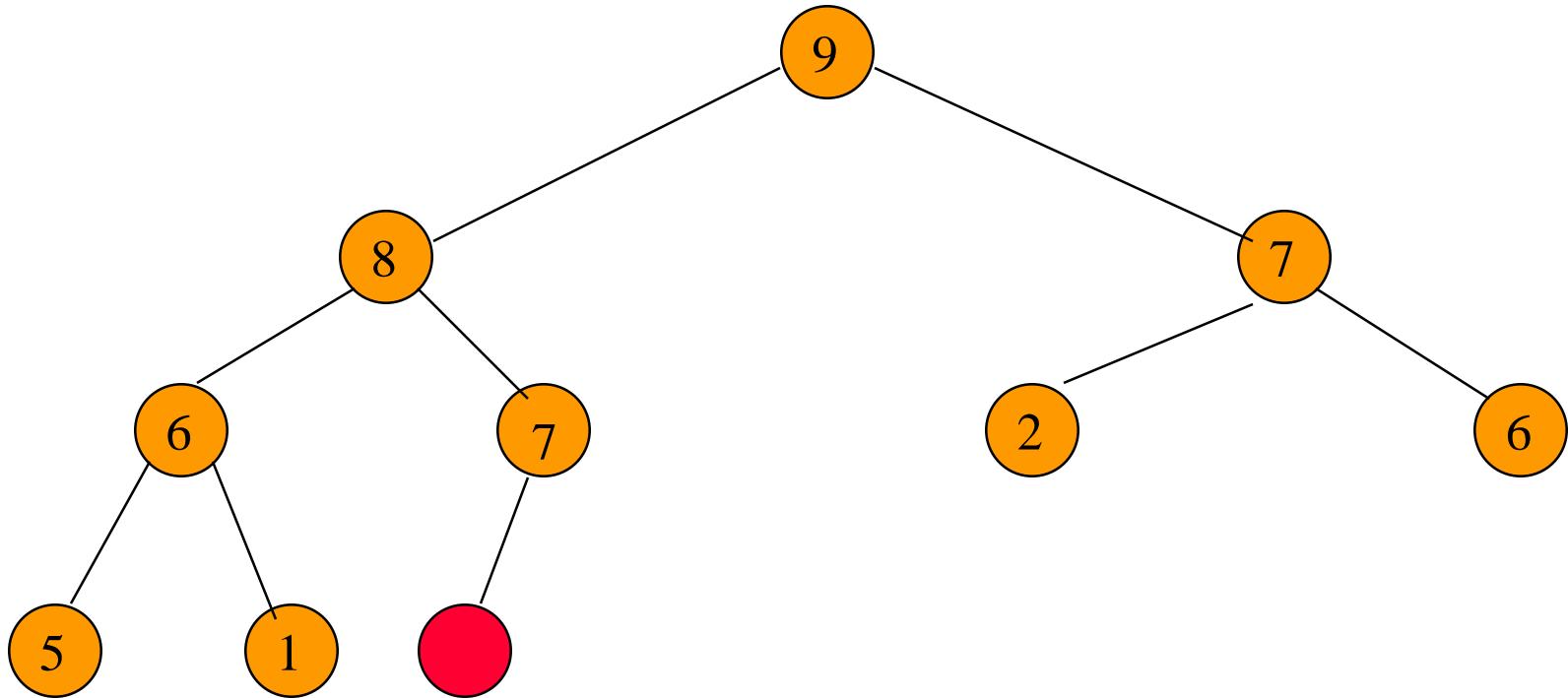
10 зангилаатай төгс хоёртын мод.

# Мах овоолгод элемент хийх



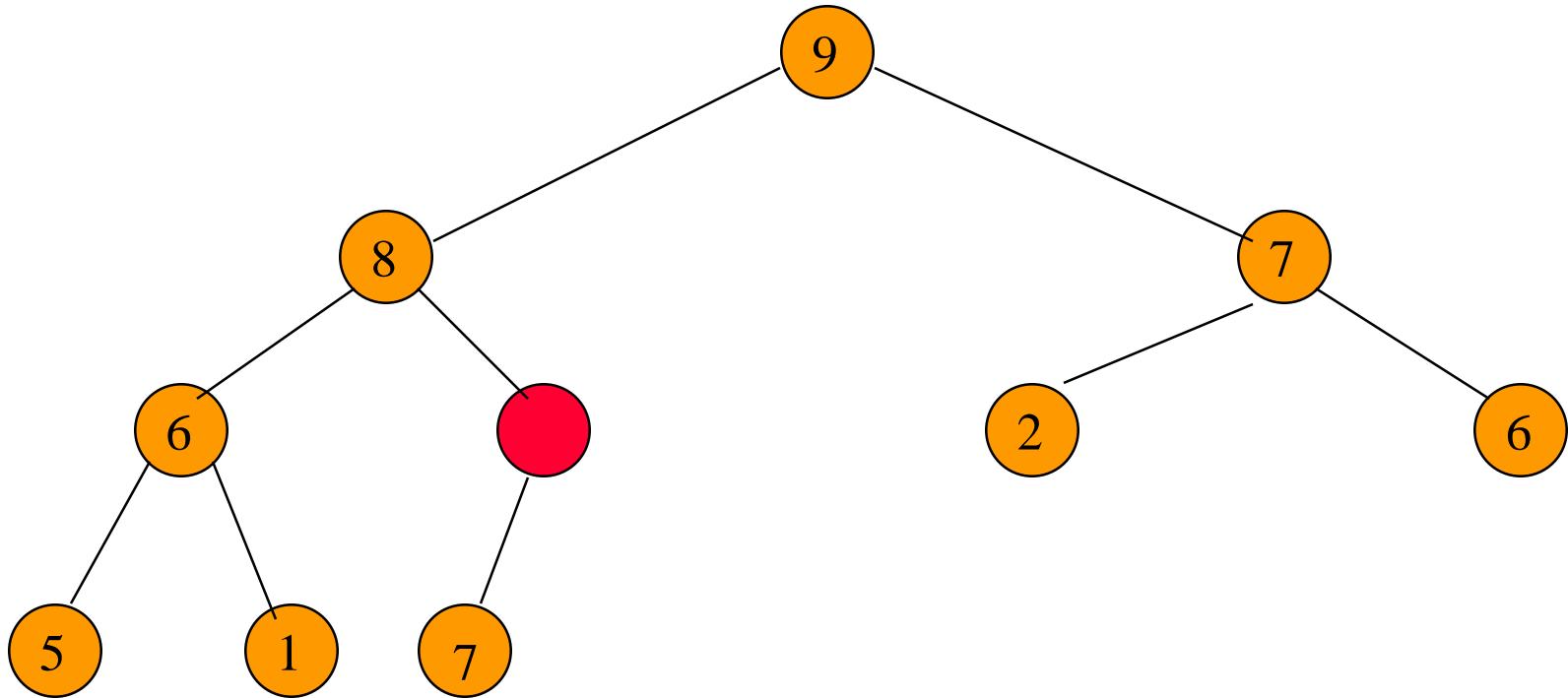
Шинэ элемент 5.

# Мах овоолгод элемент хийх



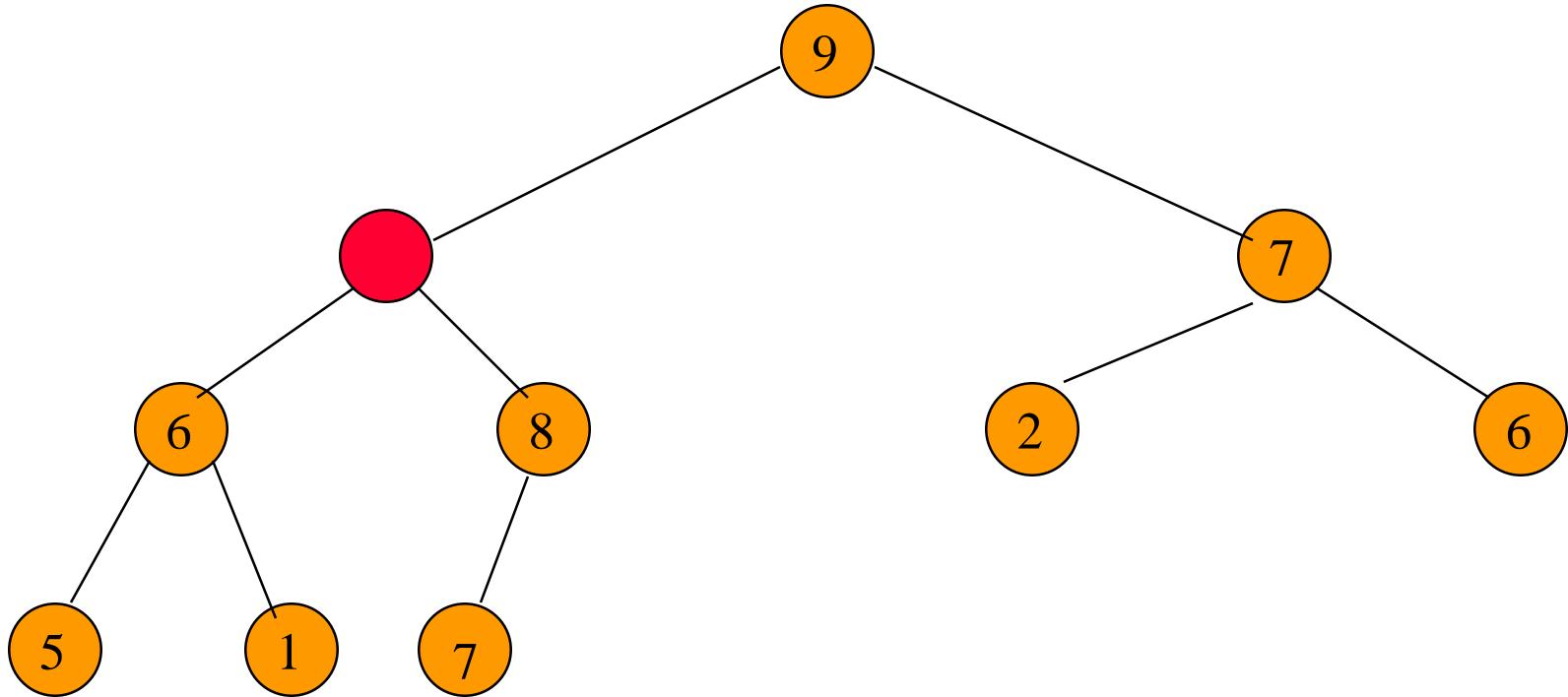
Шинэ элемент 20.

# Мах овоолгод элемент хийх



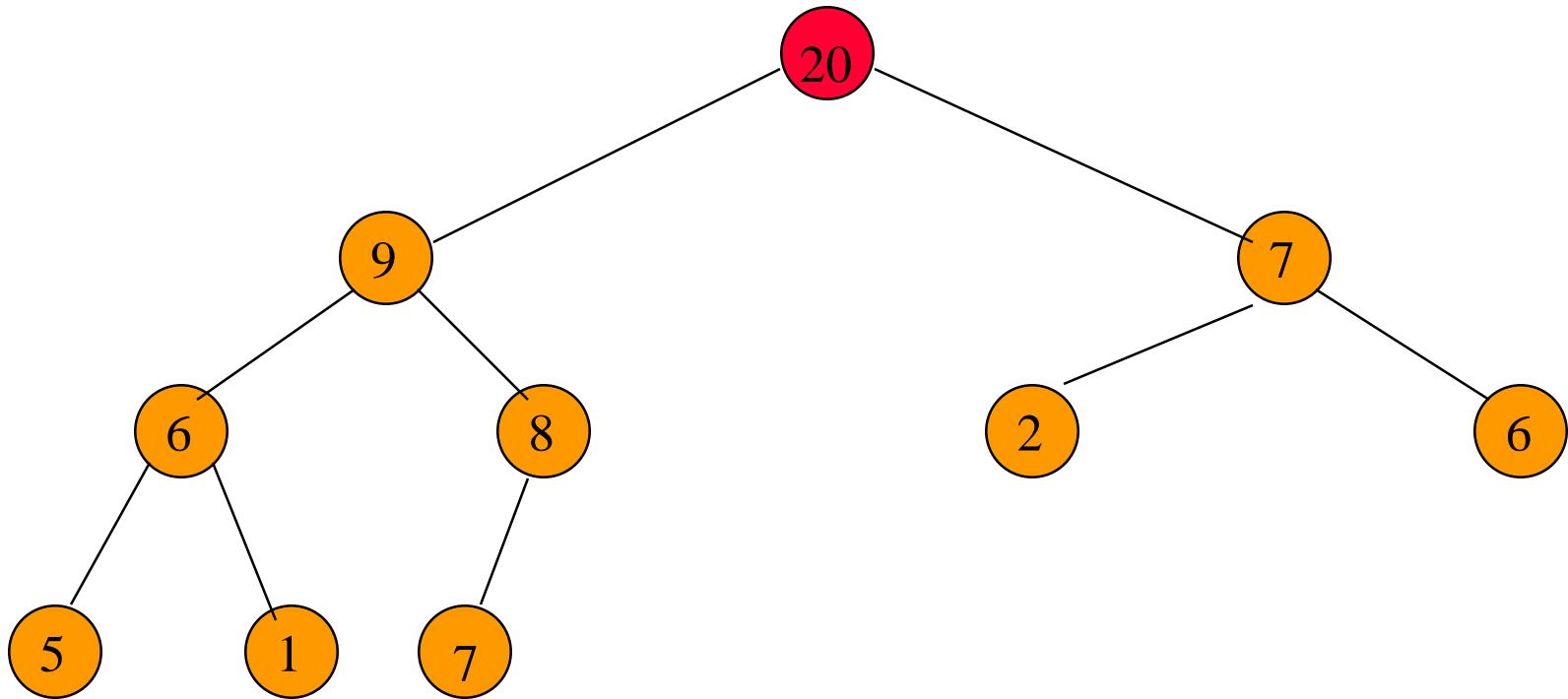
Шинэ элемент 20.

# Мах овоолгод элемент хийх



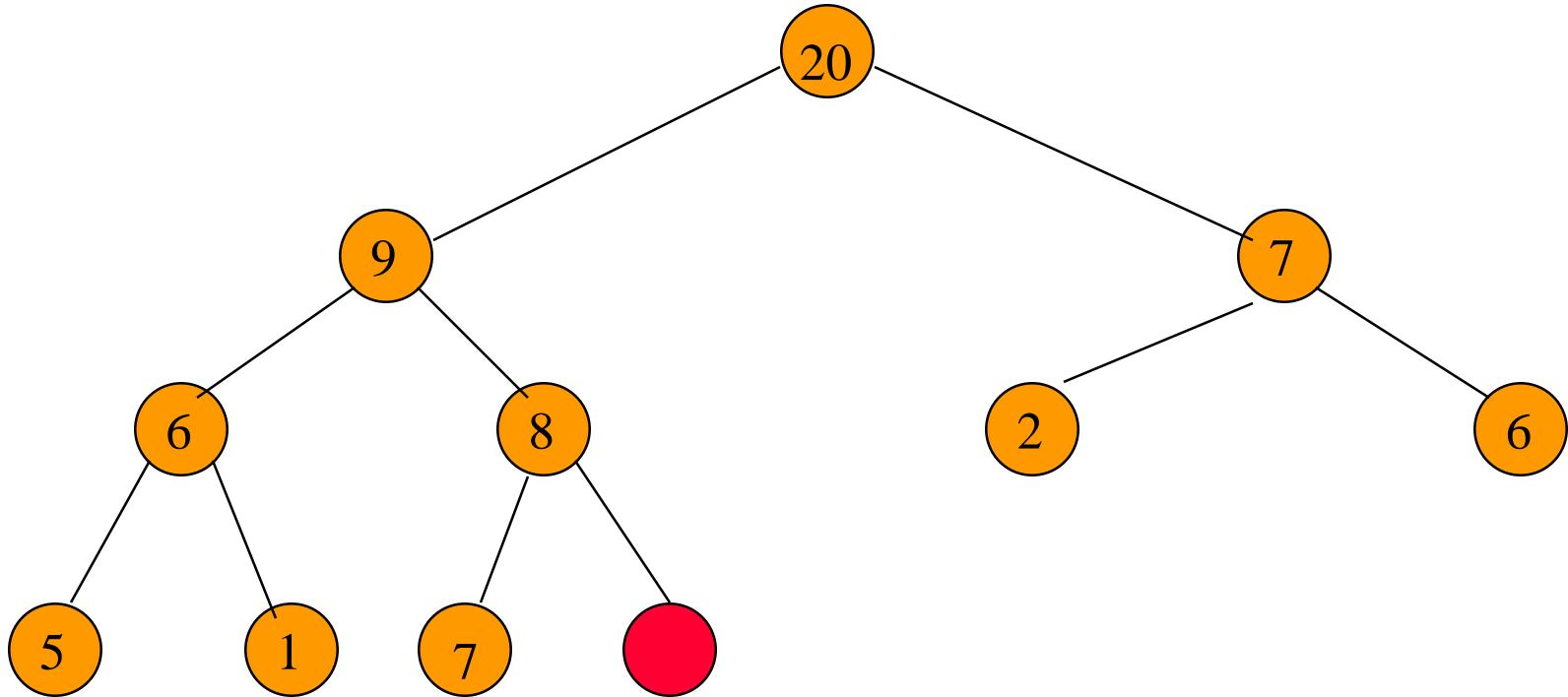
Шинэ элемент 20.

# Мах овоолгод элемент хийх



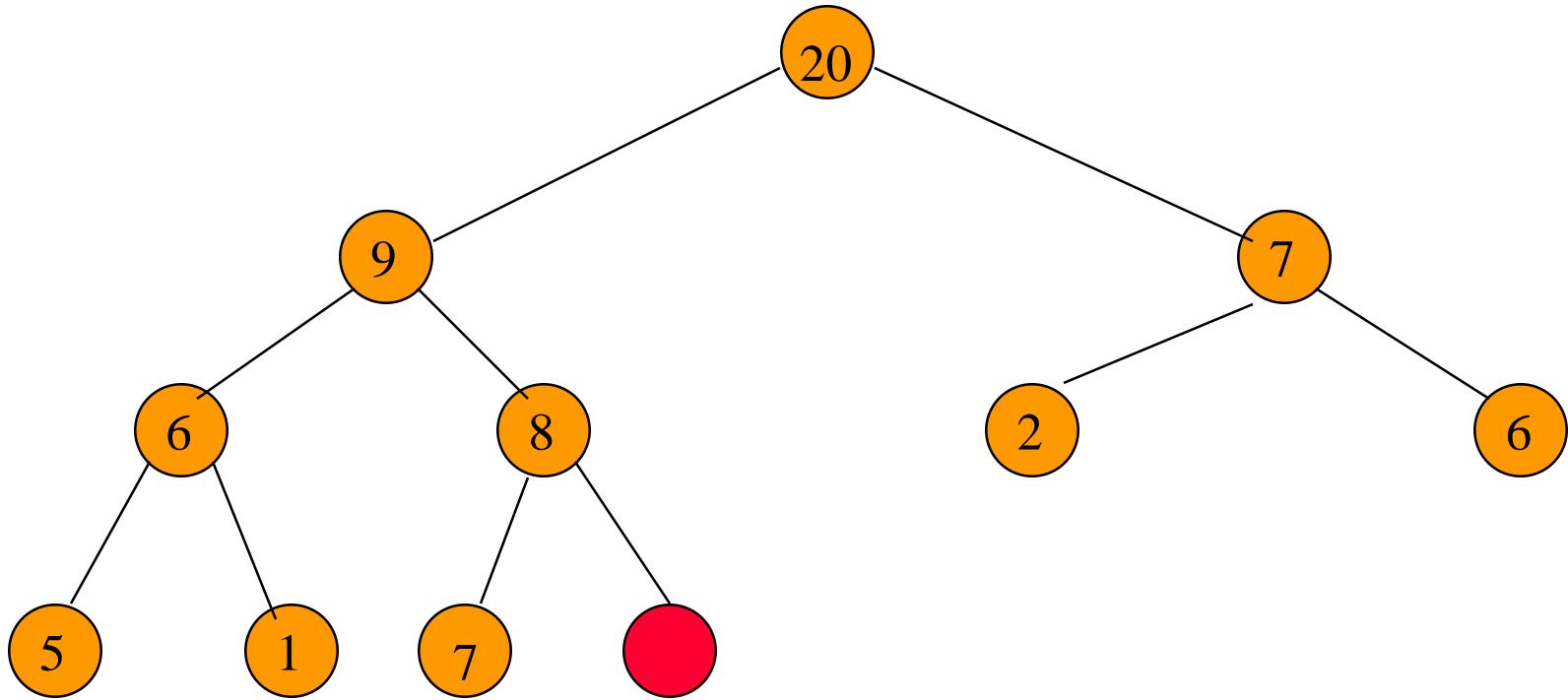
Шинэ элемент 20.

# Мах овоолгод элемент хийх



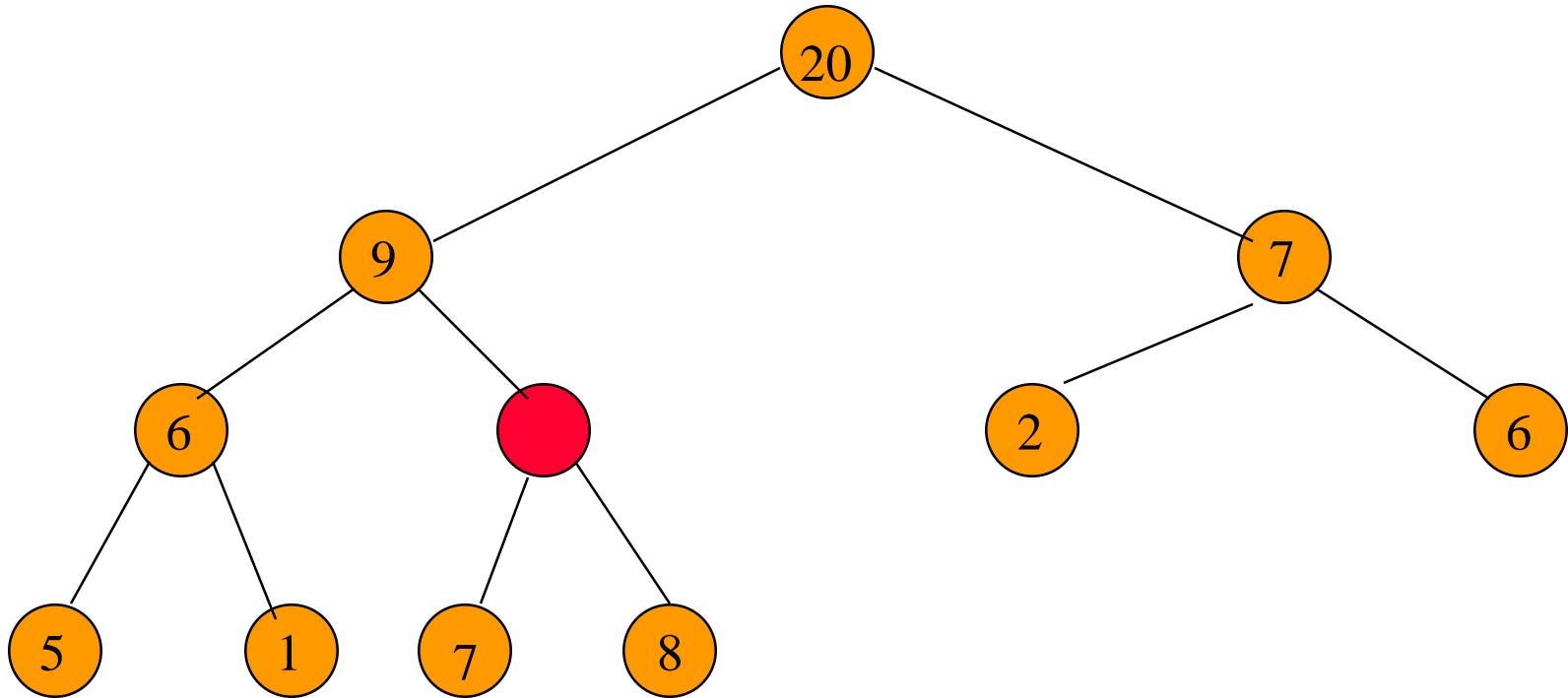
11 зангилаатай төгс хоёртын мод.

# Мах овоолгод элемент хийх



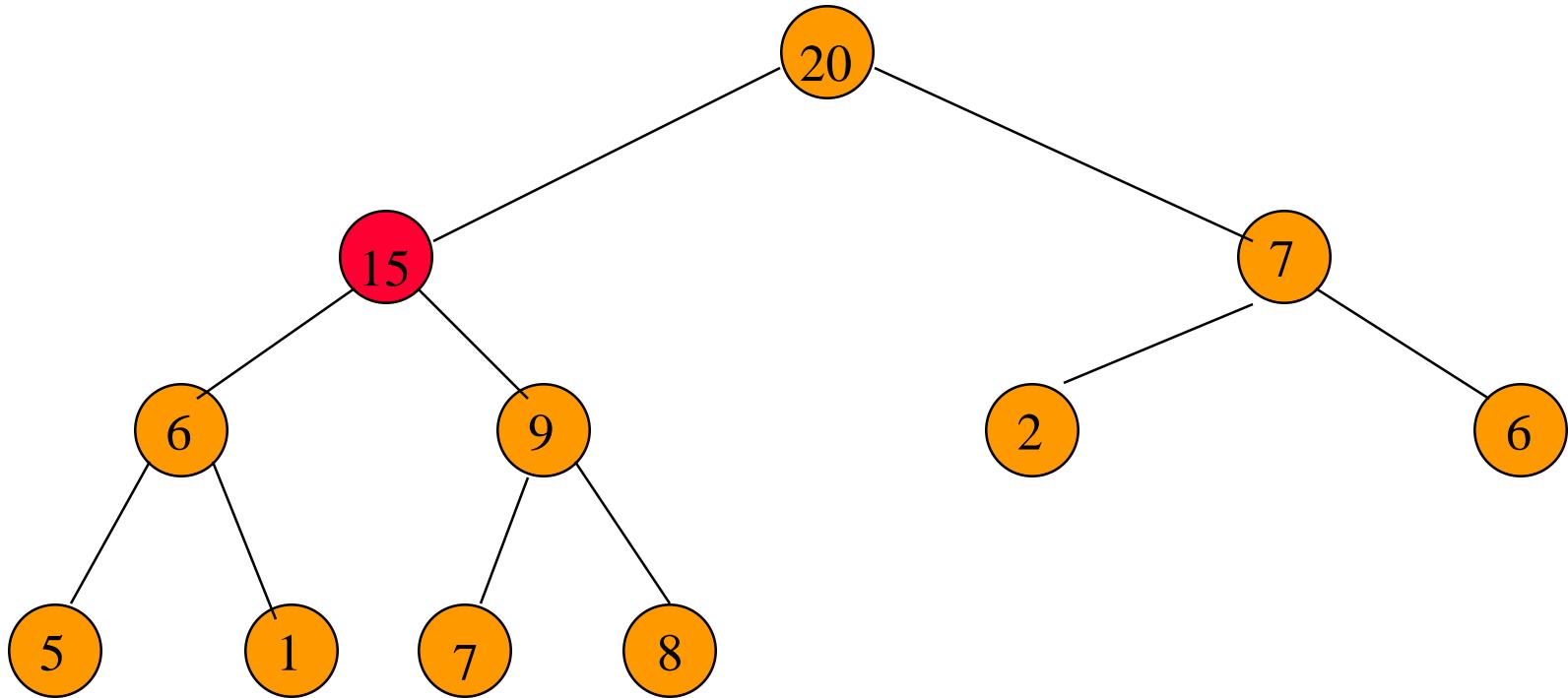
Шинэ элемент 15.

# Мах овоолгод элемент хийх



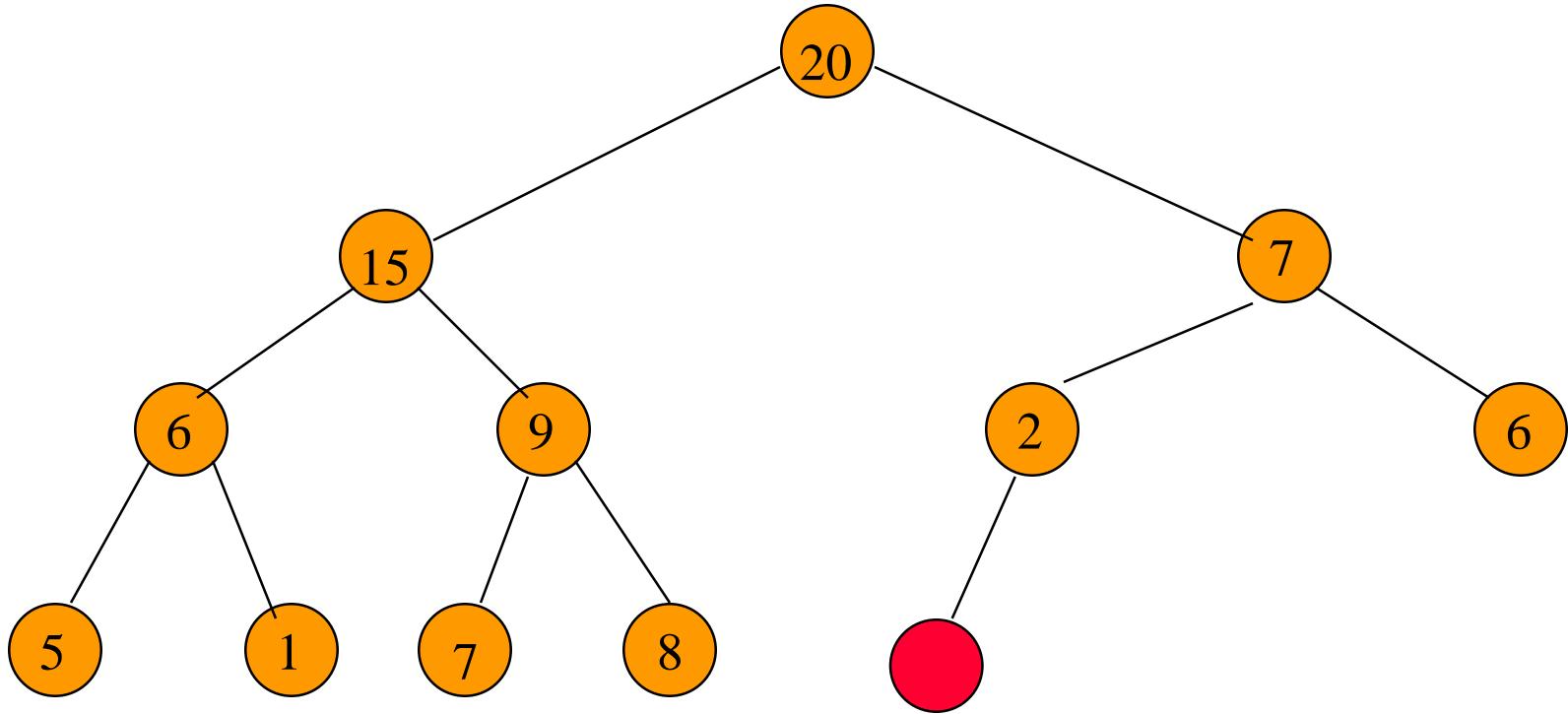
Шинэ элемент 15.

# Мах овоолгод элемент хийх



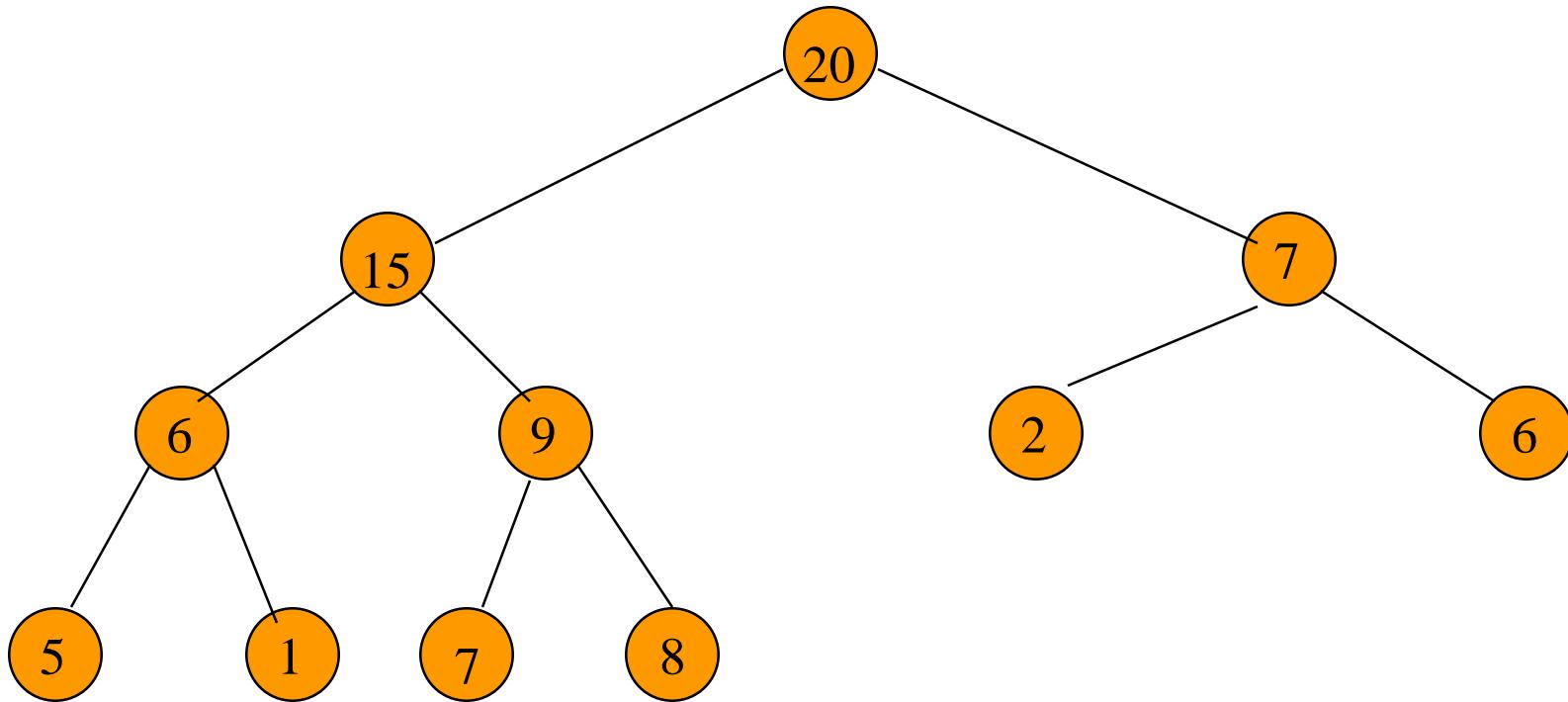
Шинэ элемент 15.

# Put үйлдлийн хугацаа



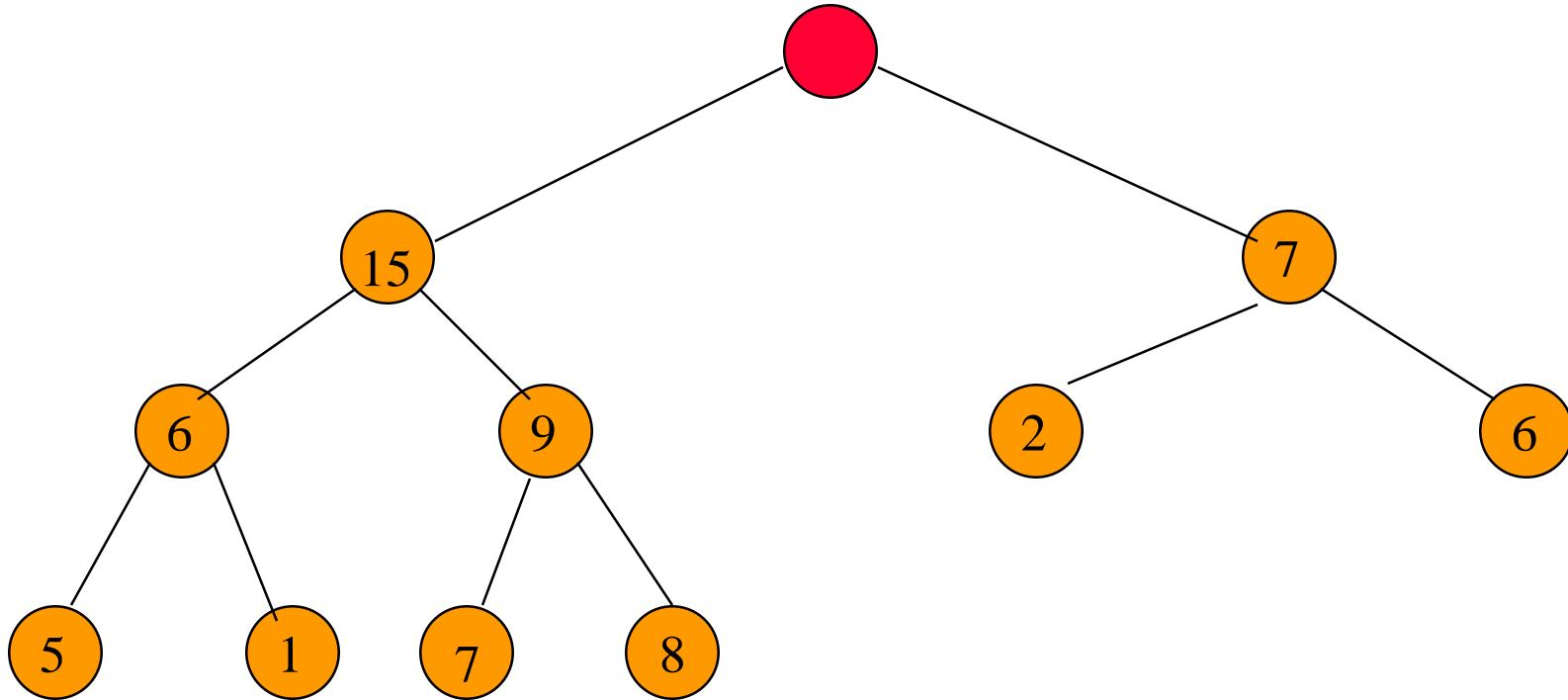
Хугацаа  $O(\log n)$ , үүнд  $n$  овоолгын  
хэмжээ.

# Max элементийн устгах



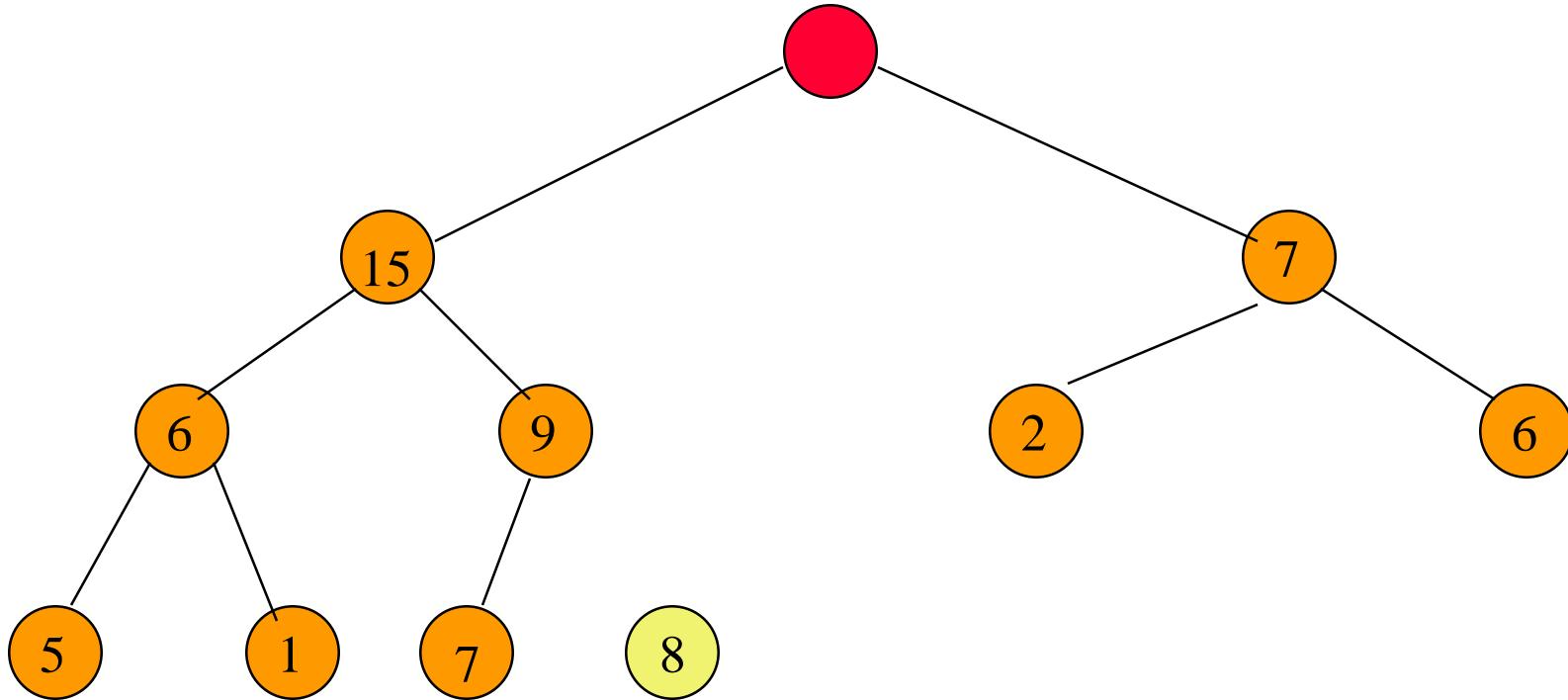
Max элемент нь үндэс.

# Мах элементийн устгах



max элементийг устгасны дараа.

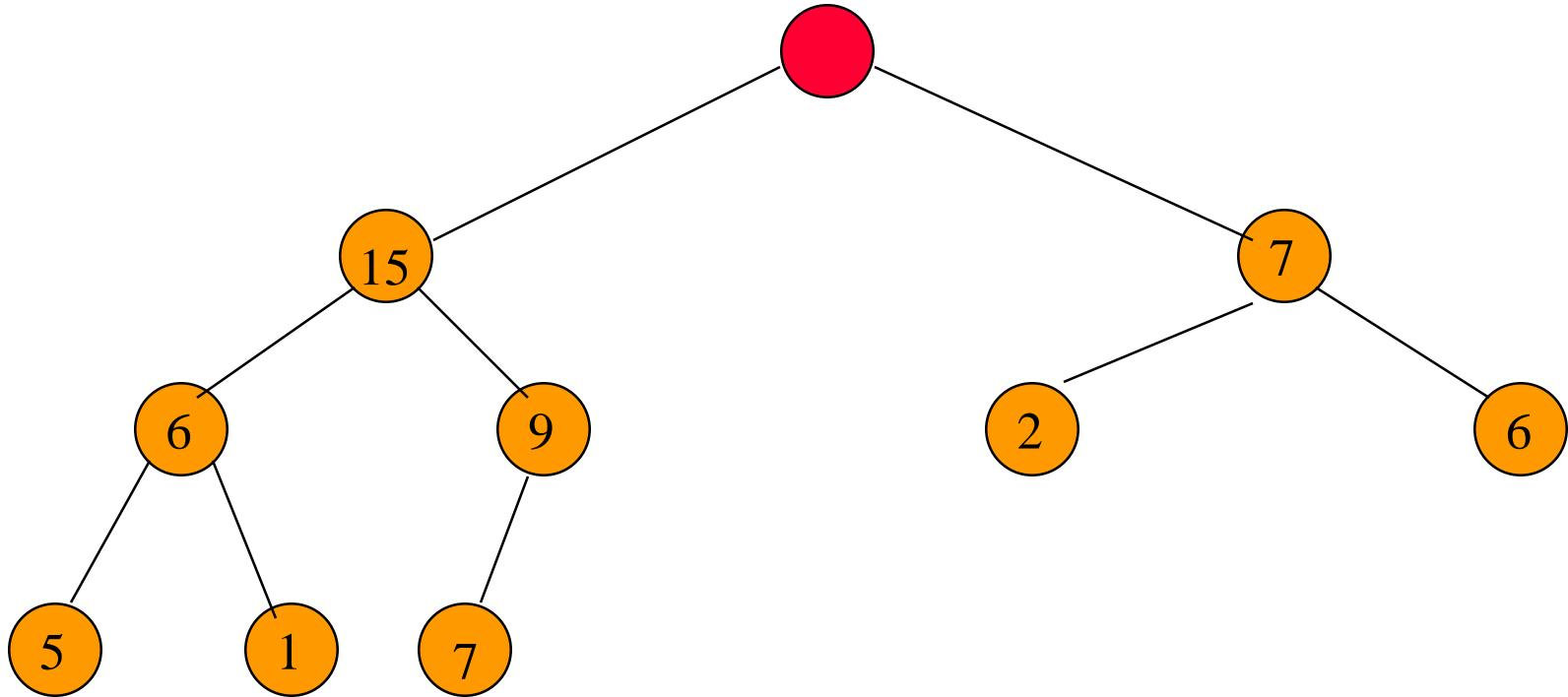
# Мах элементийн устгах



10 зангилаатай овоолго.

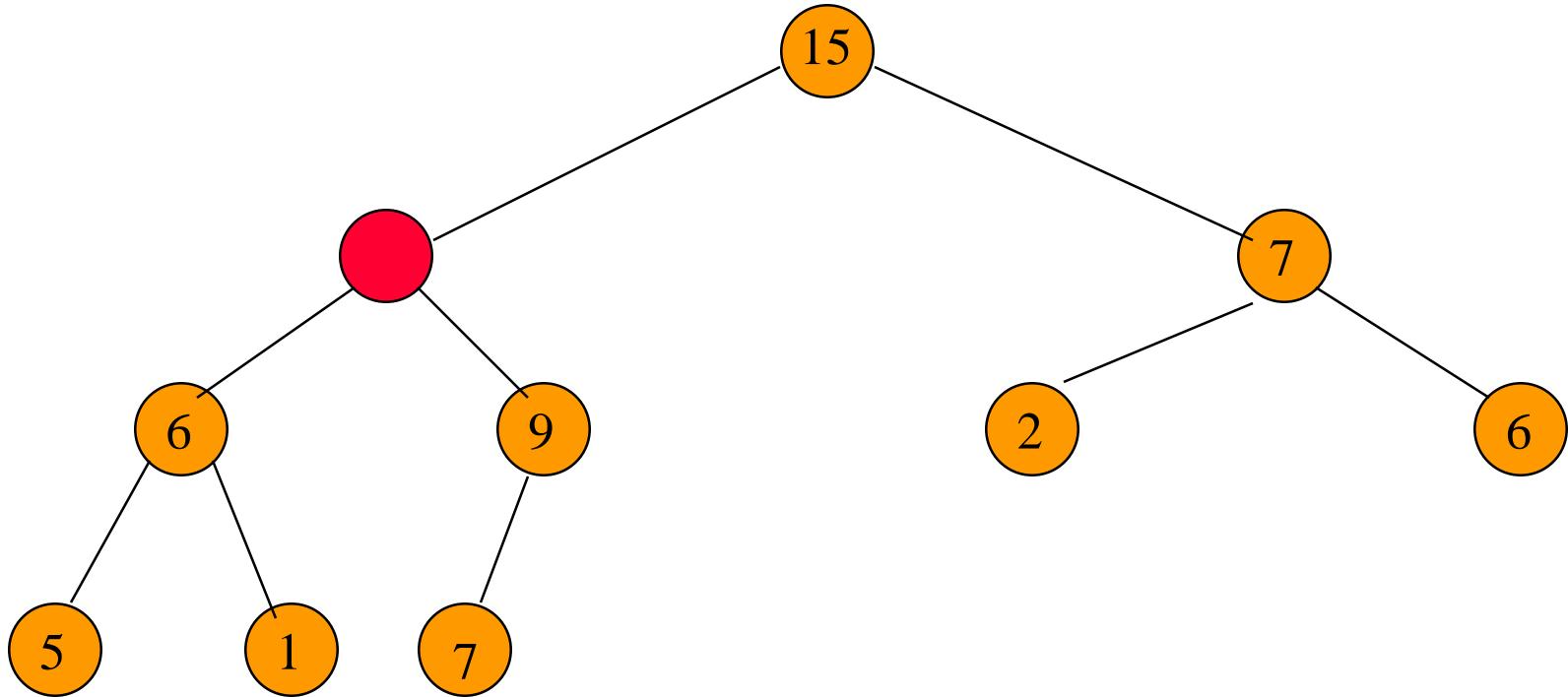
Овоолгод 8 –г дахин хийх.

# Мах элементийн устгах



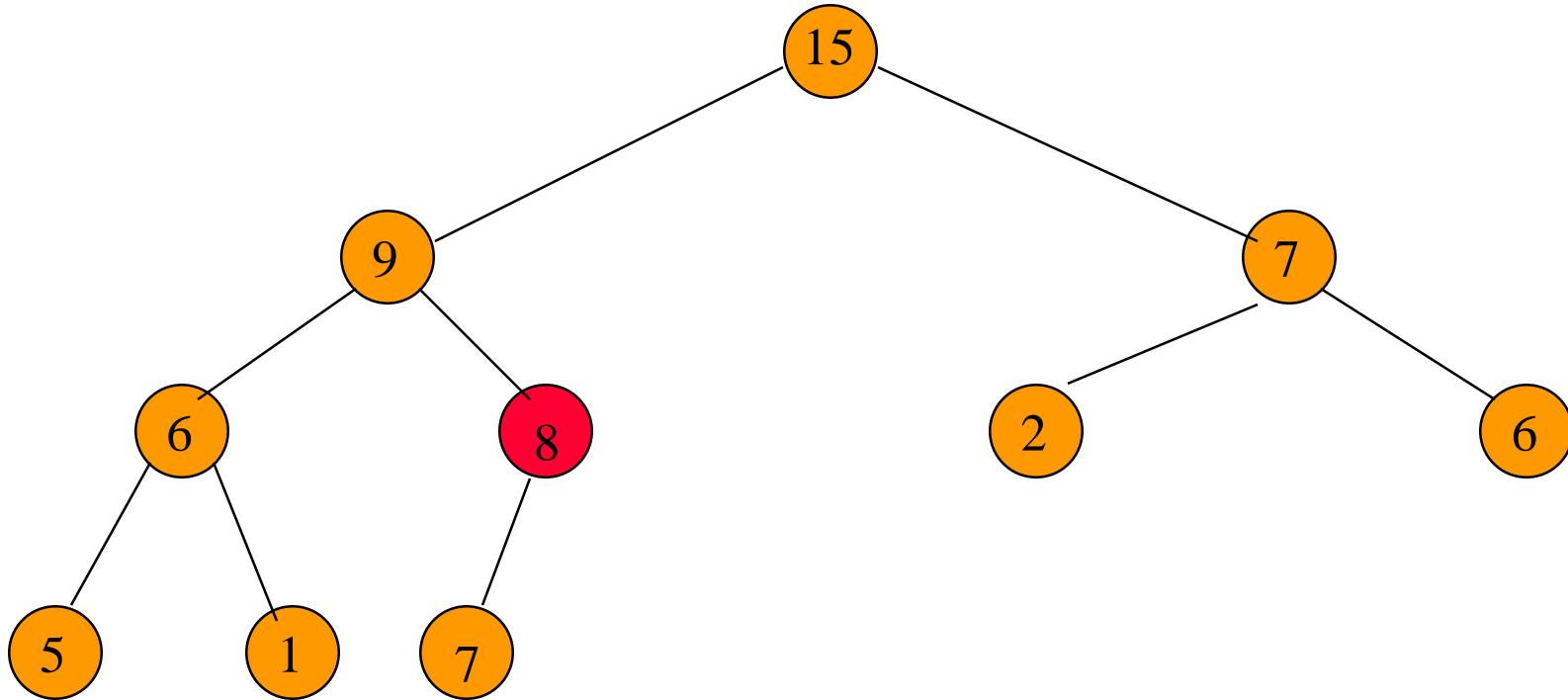
Овоолгод **8** –г дахин хийх.

# Мах элементийн устгах



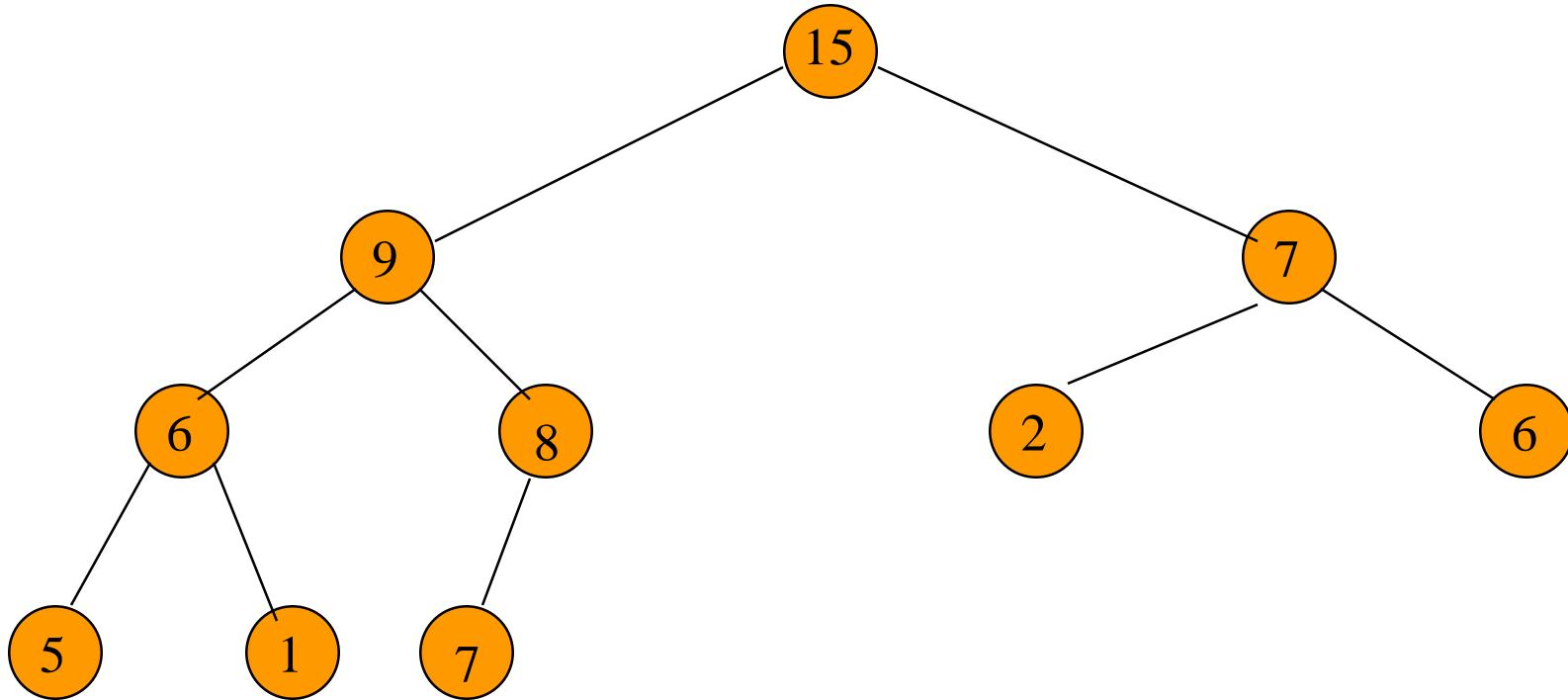
Овоолгод 8 –г дахин хийх.

# Мах элементийн устгах



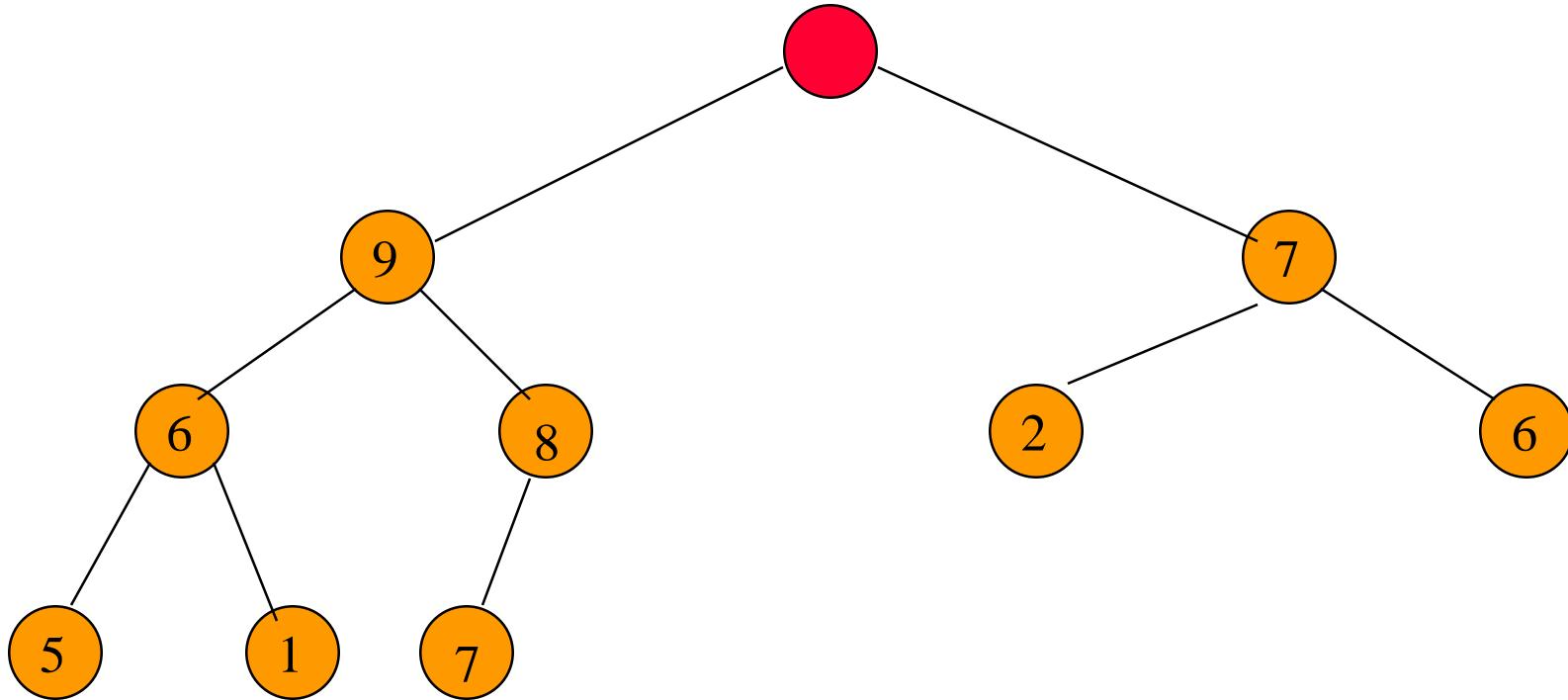
Овоолго 8 –г дахин хийх.

# Мах элементийн устгах



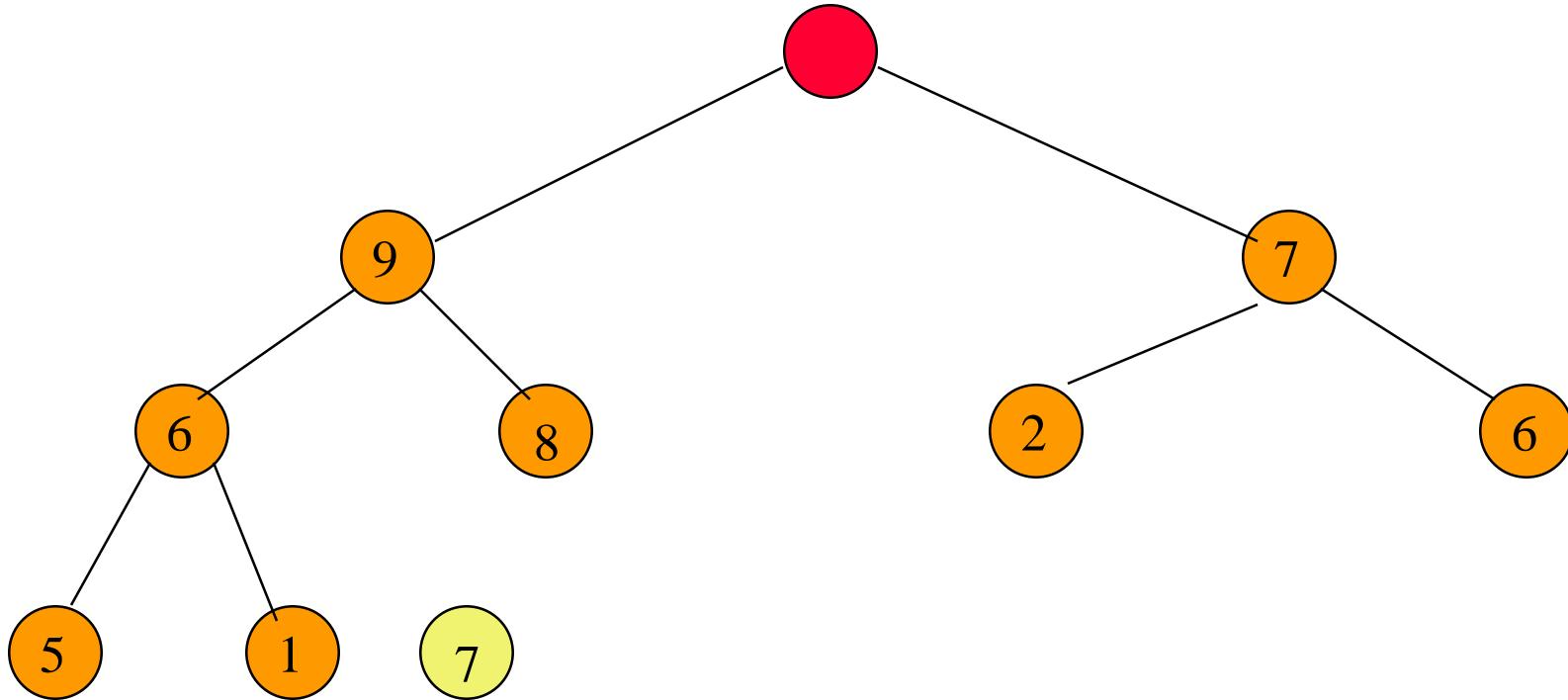
Мах элемент **15**.

# Мах элементийн устгах



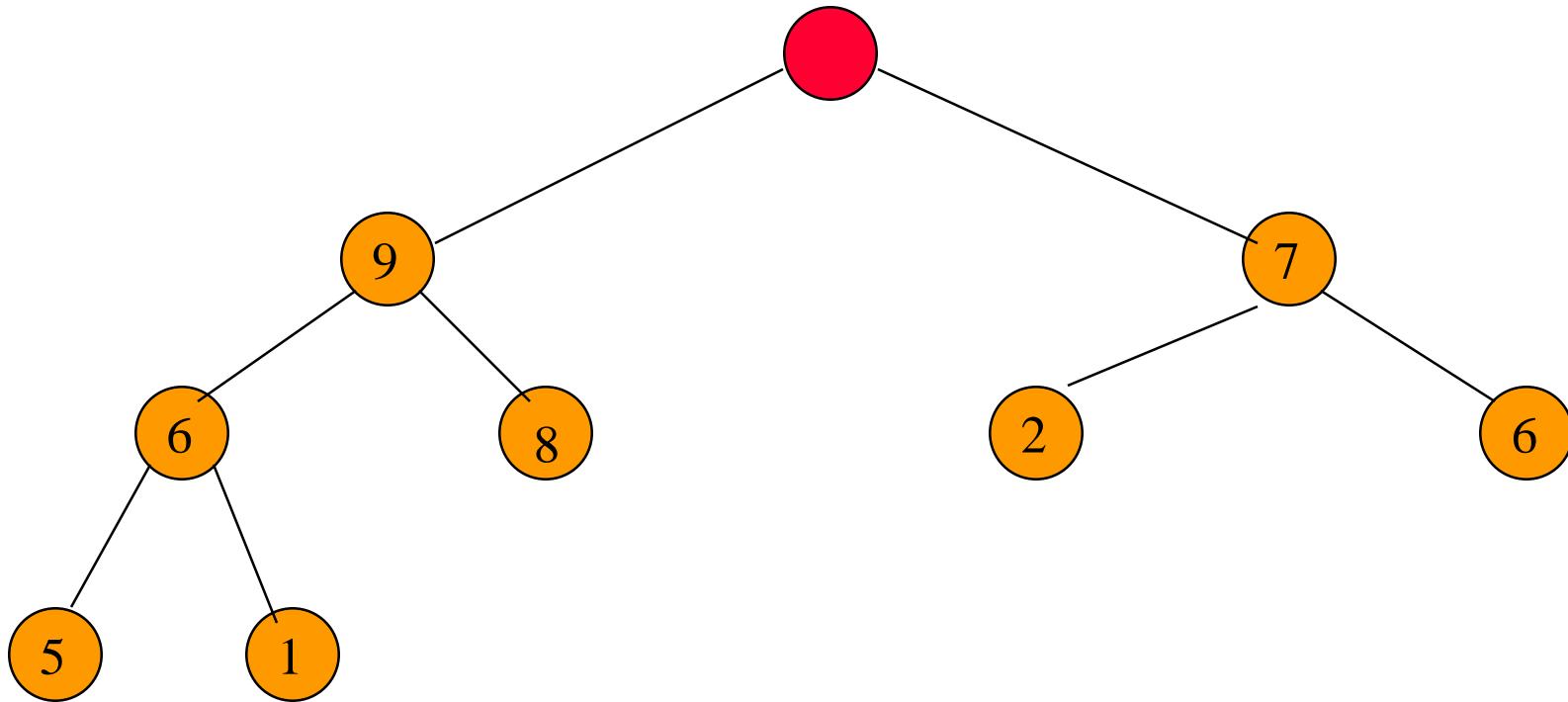
max элементийг устгасны дараа.

# Мах элементийн устгах



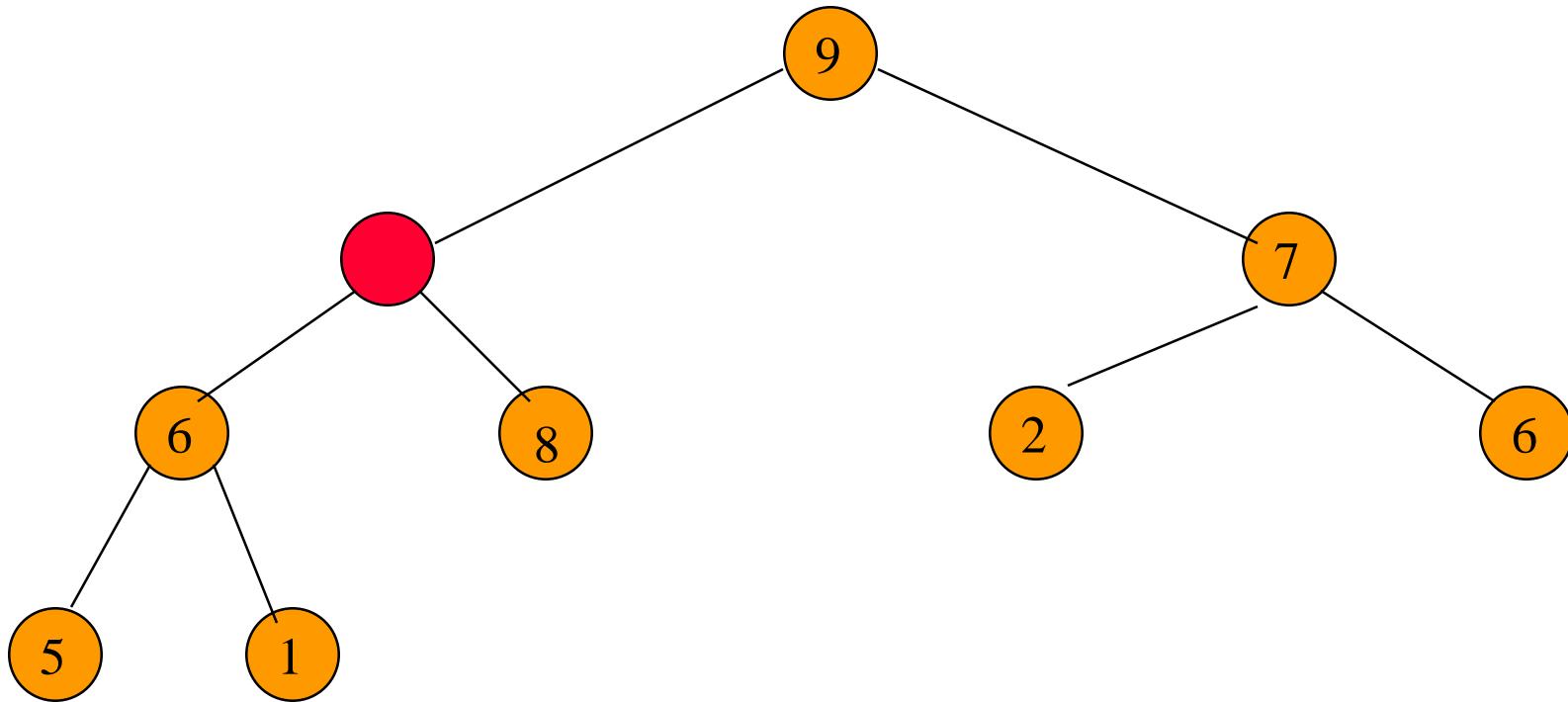
9 зангилаатай овоолго.

# Мах элементийн устгах



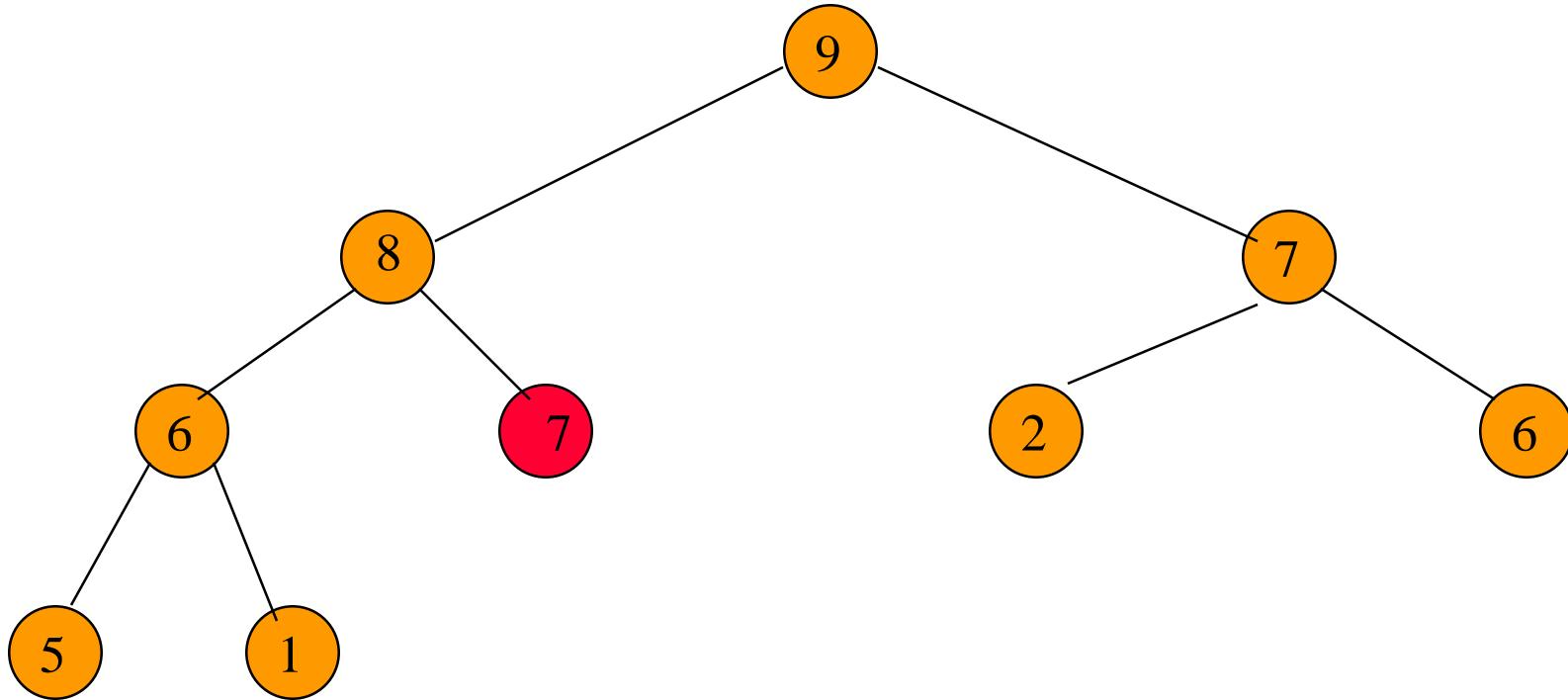
7 -г дахин хийх

# Мах элементийн устгах



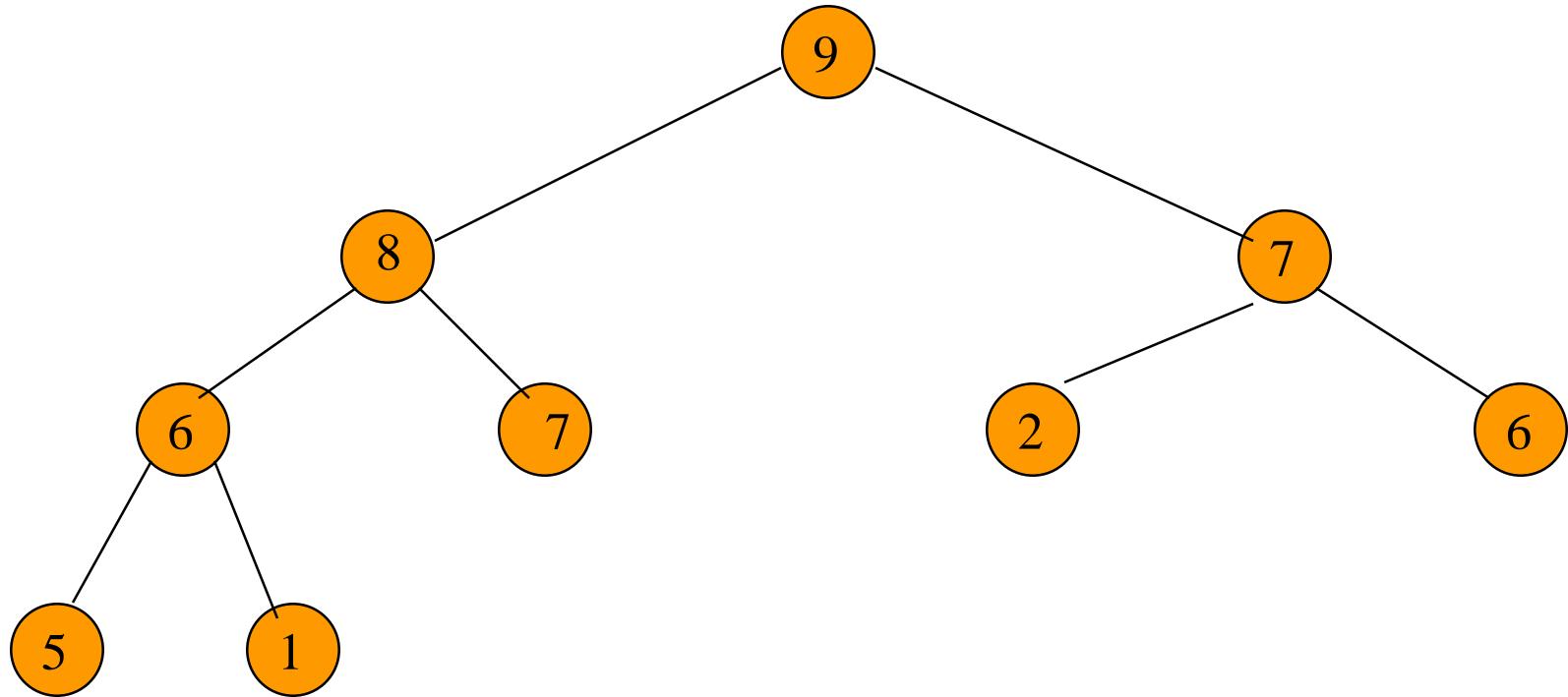
7 -г дахин хийх

# Мах элементийн устгах



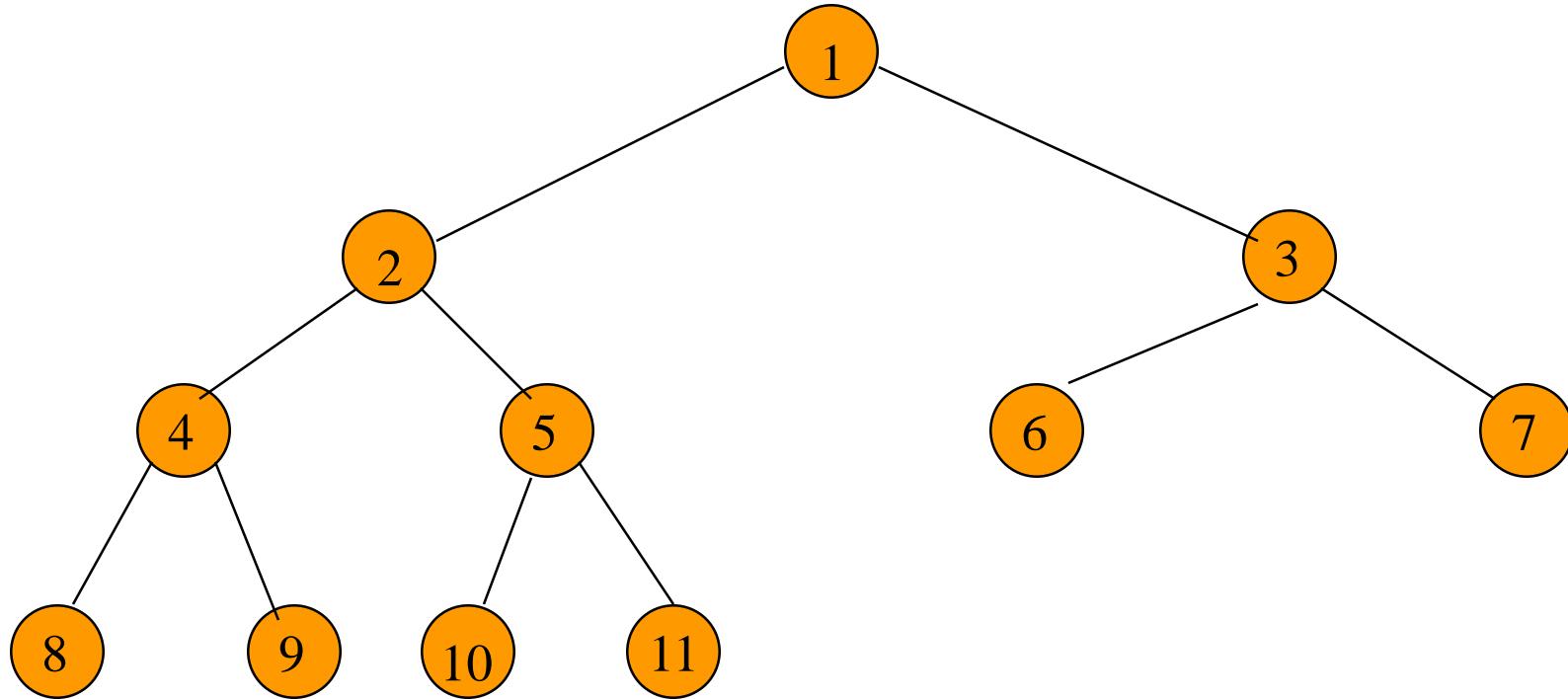
7 -г дахин хийх

# Remove Max үйлдлийн хугацаа



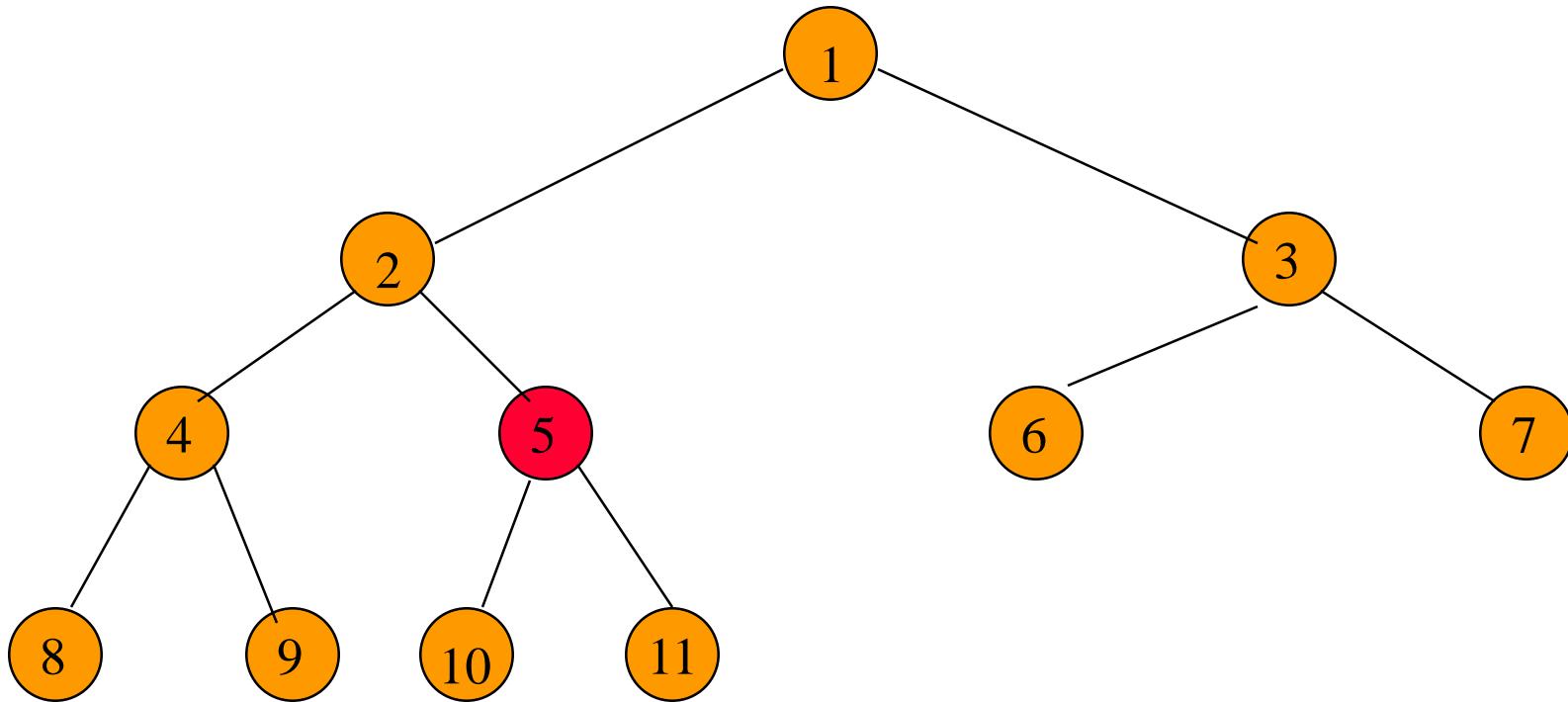
Хугацаа  $O(\log n)$ .

# Мах овоолгыг идэхижүүлэх



Оролтын массив = [-, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

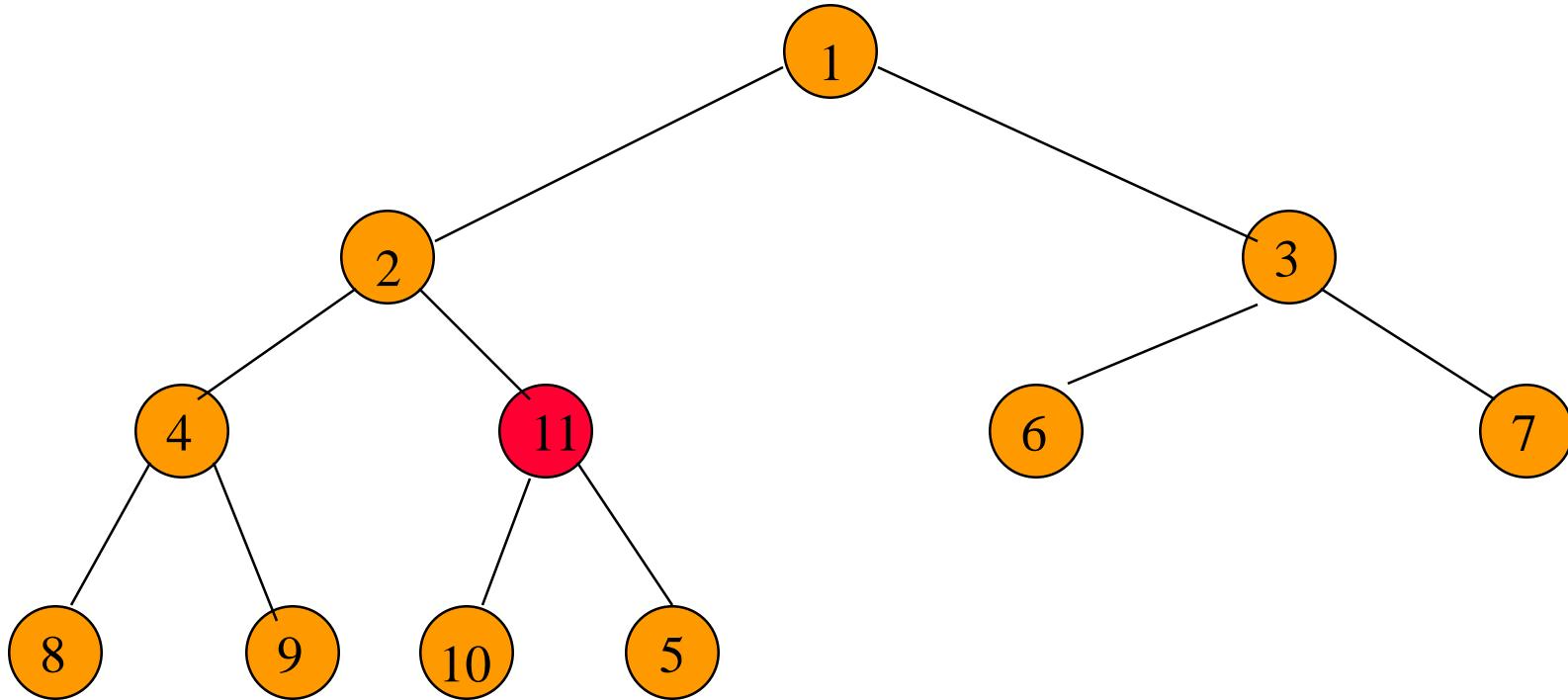
# Мах овоолгыг идэхижүүлэх



Массивын хамгийн баруун байршилтай хүүхэдтэй  
зангилаанаас эхэлнэ.

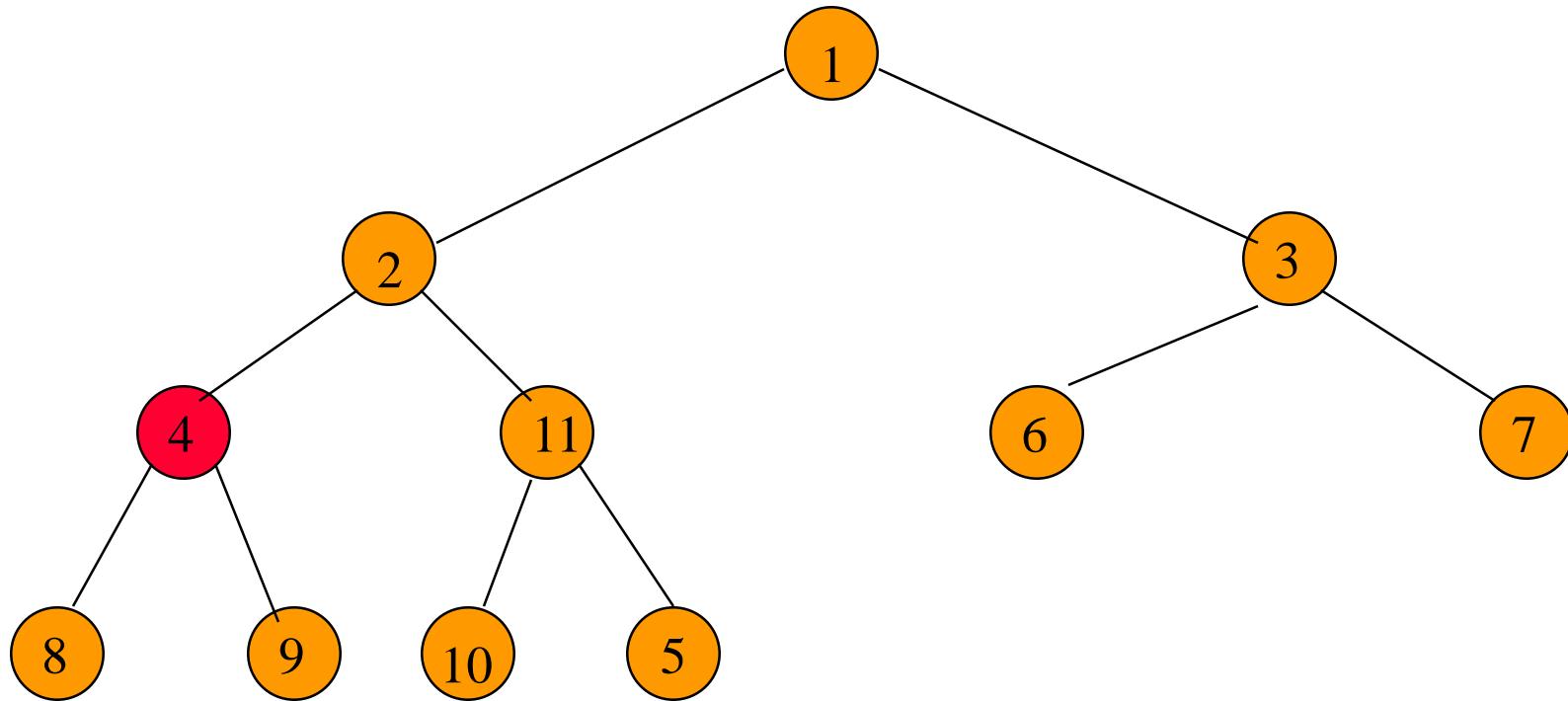
Индекс нь  $n/2$ .

# Мах овоолгыг идэхижүүлэх

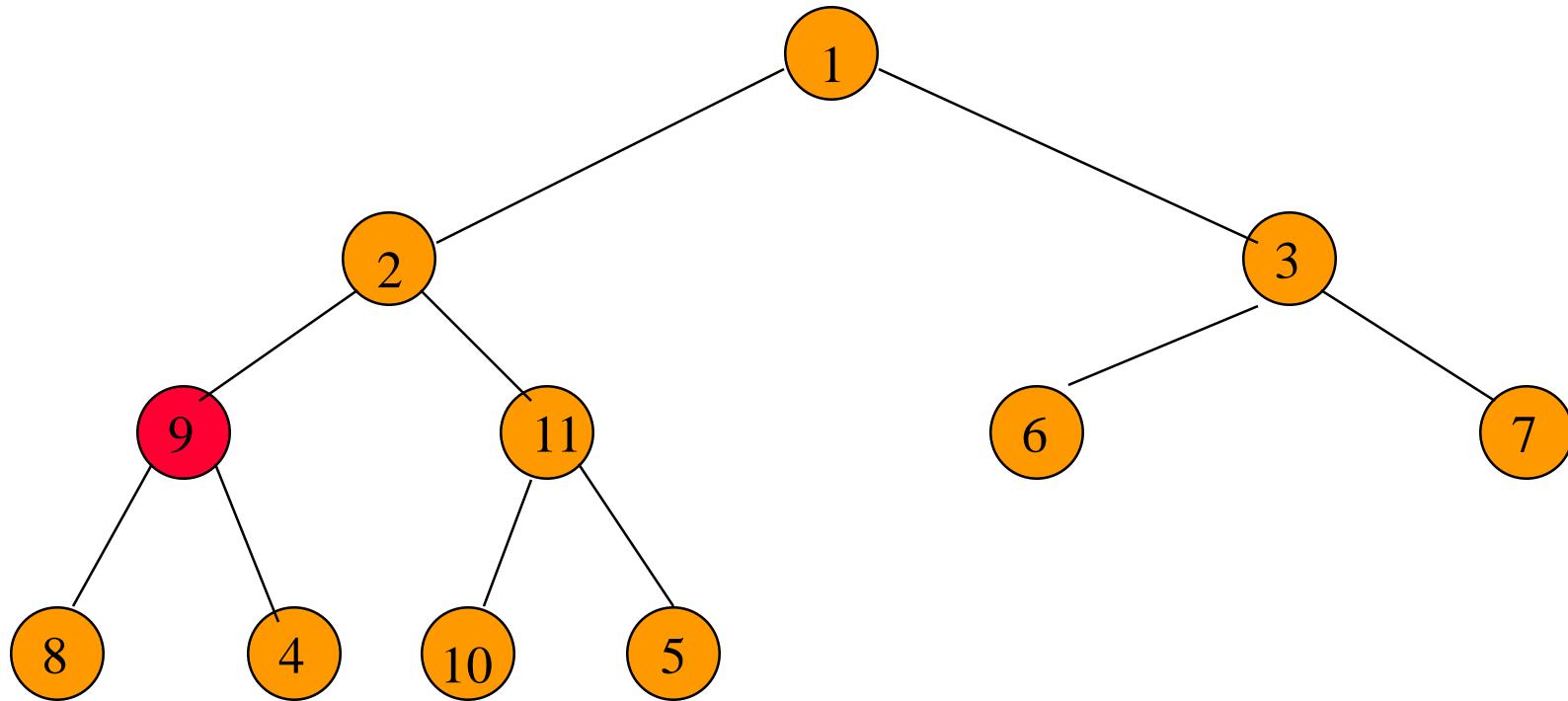


Массивын дараачийн доод байршил руу явна.

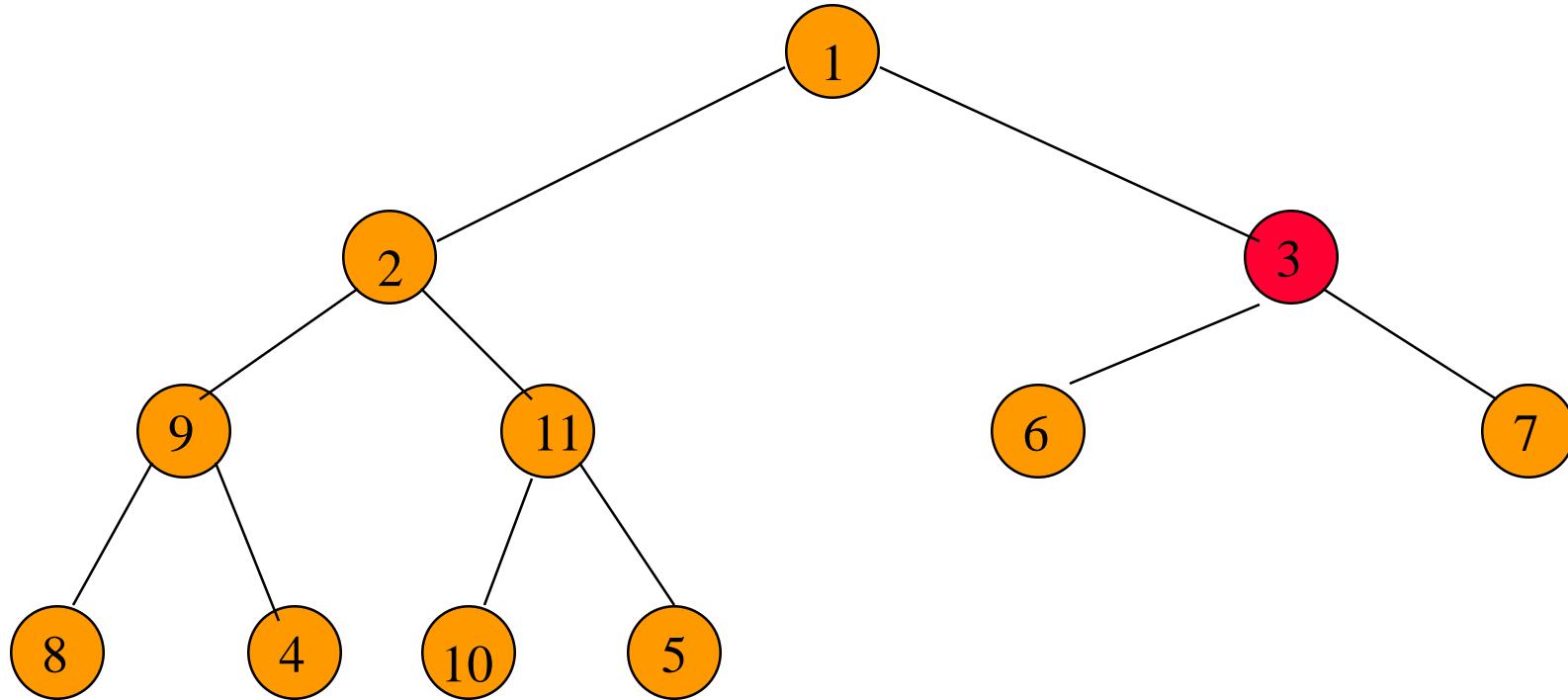
# Мах овоолгыг идэхижүүлэх



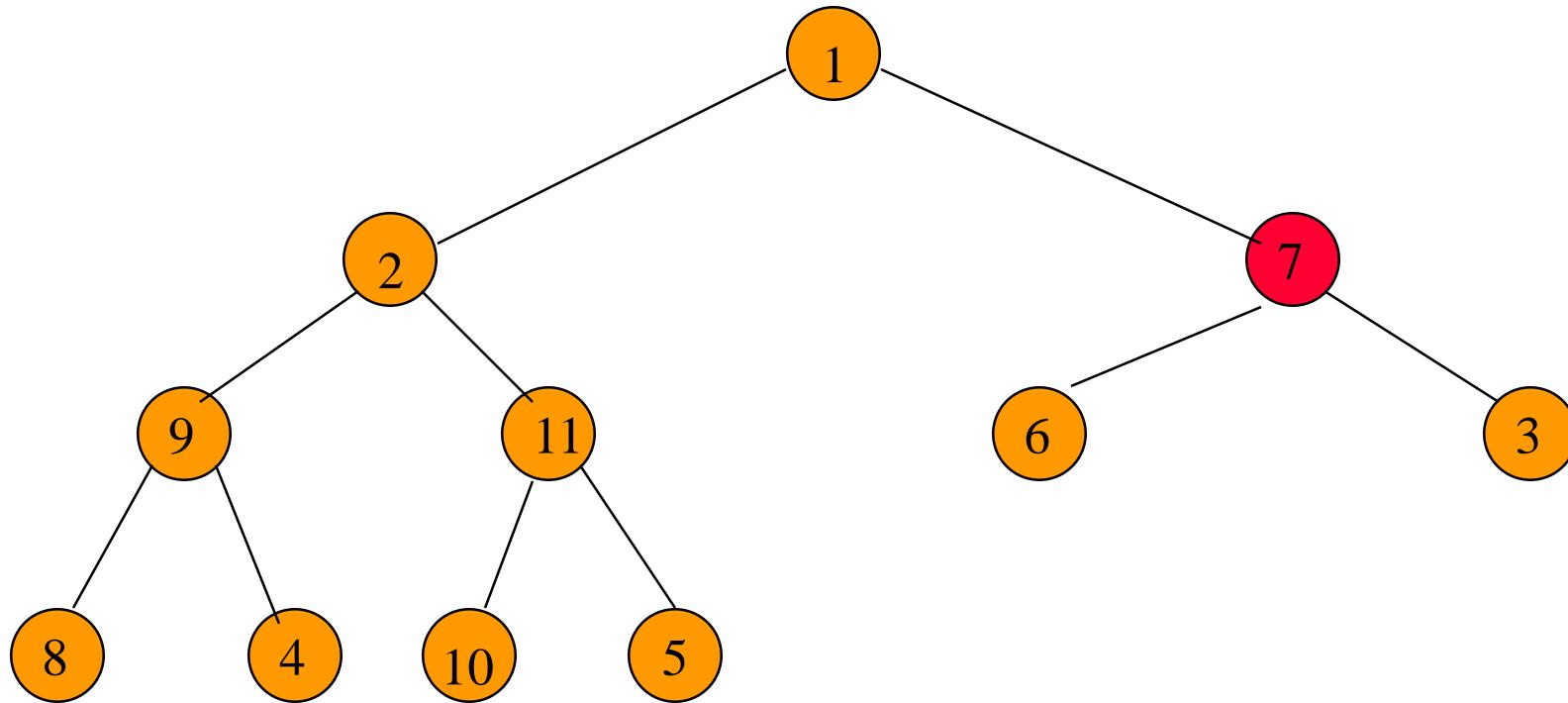
# Мах овоолгыг идэхижүүлэх



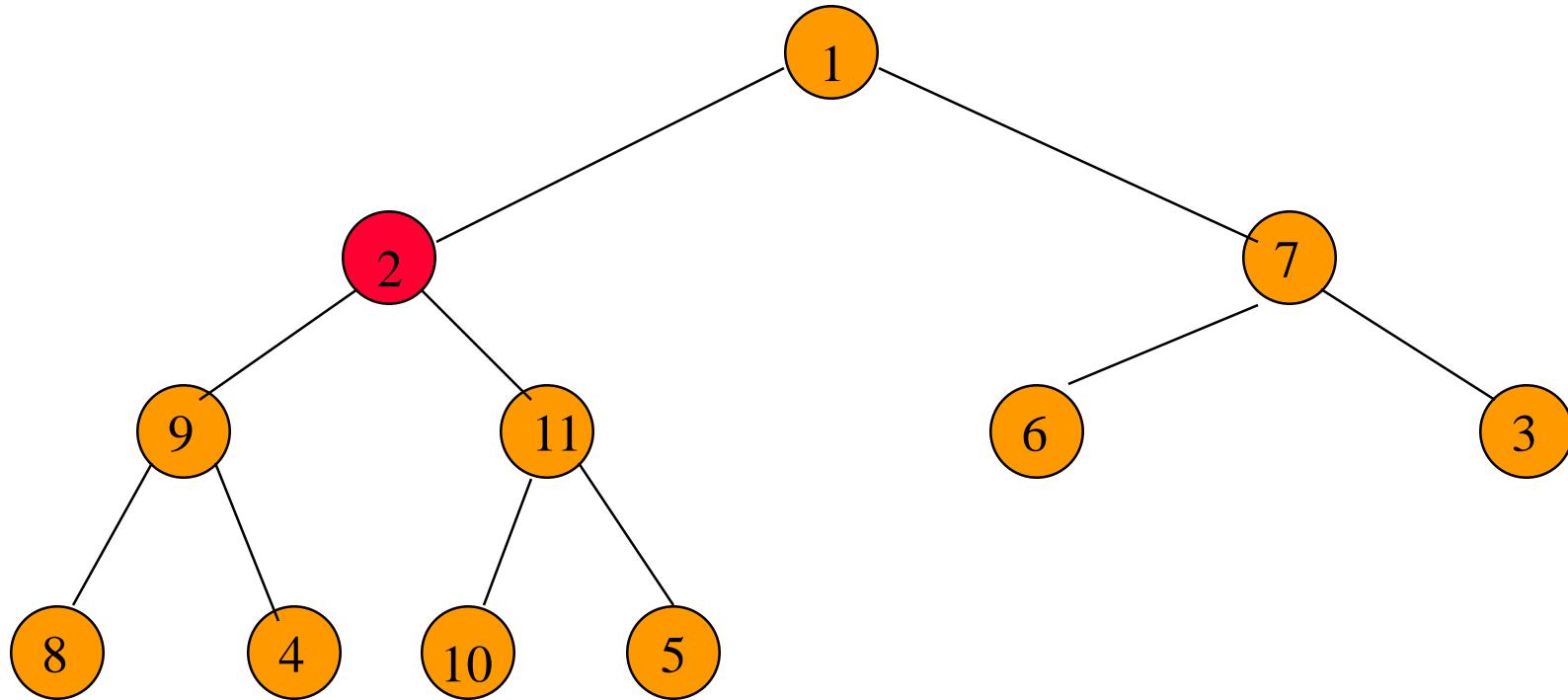
# Мах овоолгыг идэхижүүлэх



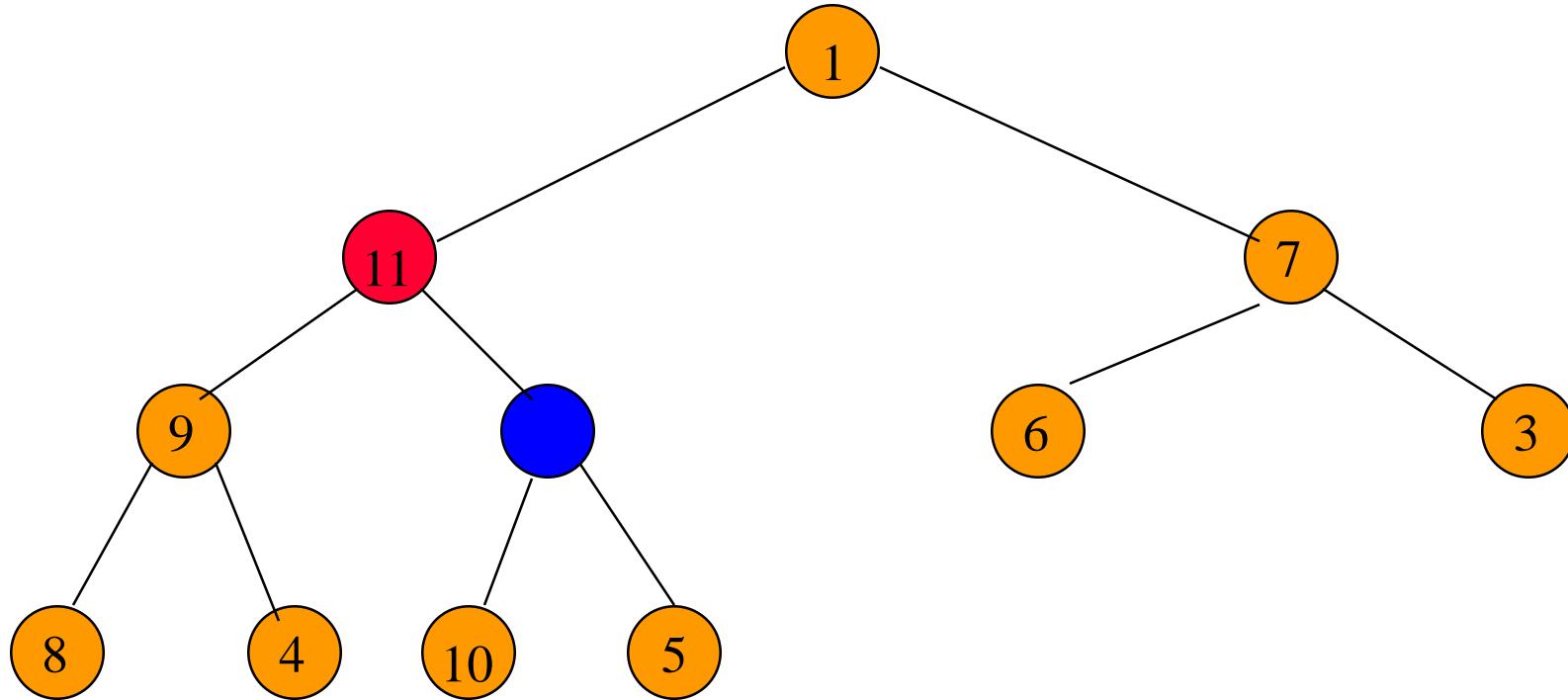
# Мах овоолгыг идэхижүүлэх



# Мах овоолгыг идэхижүүлэх

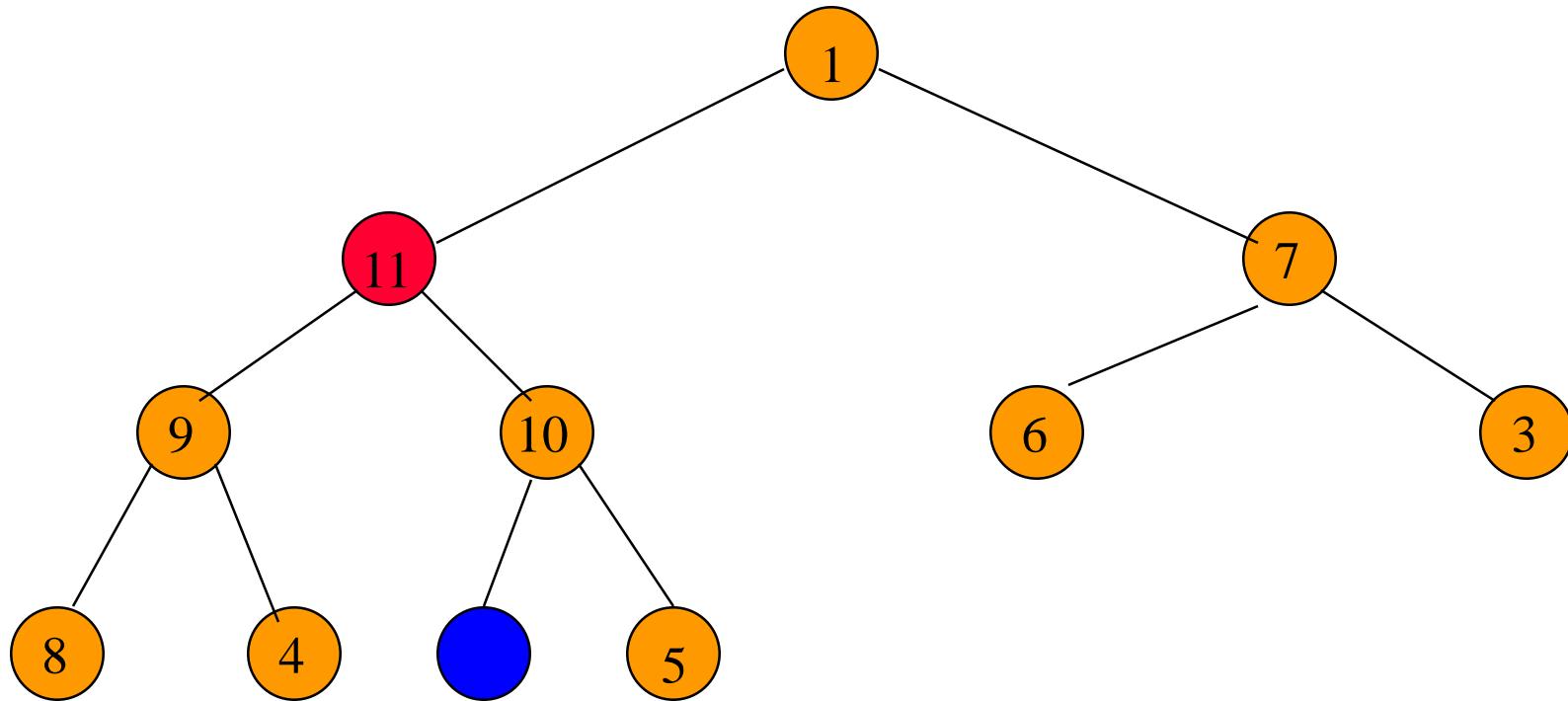


# Мах овоолгыг идэхижүүлэх



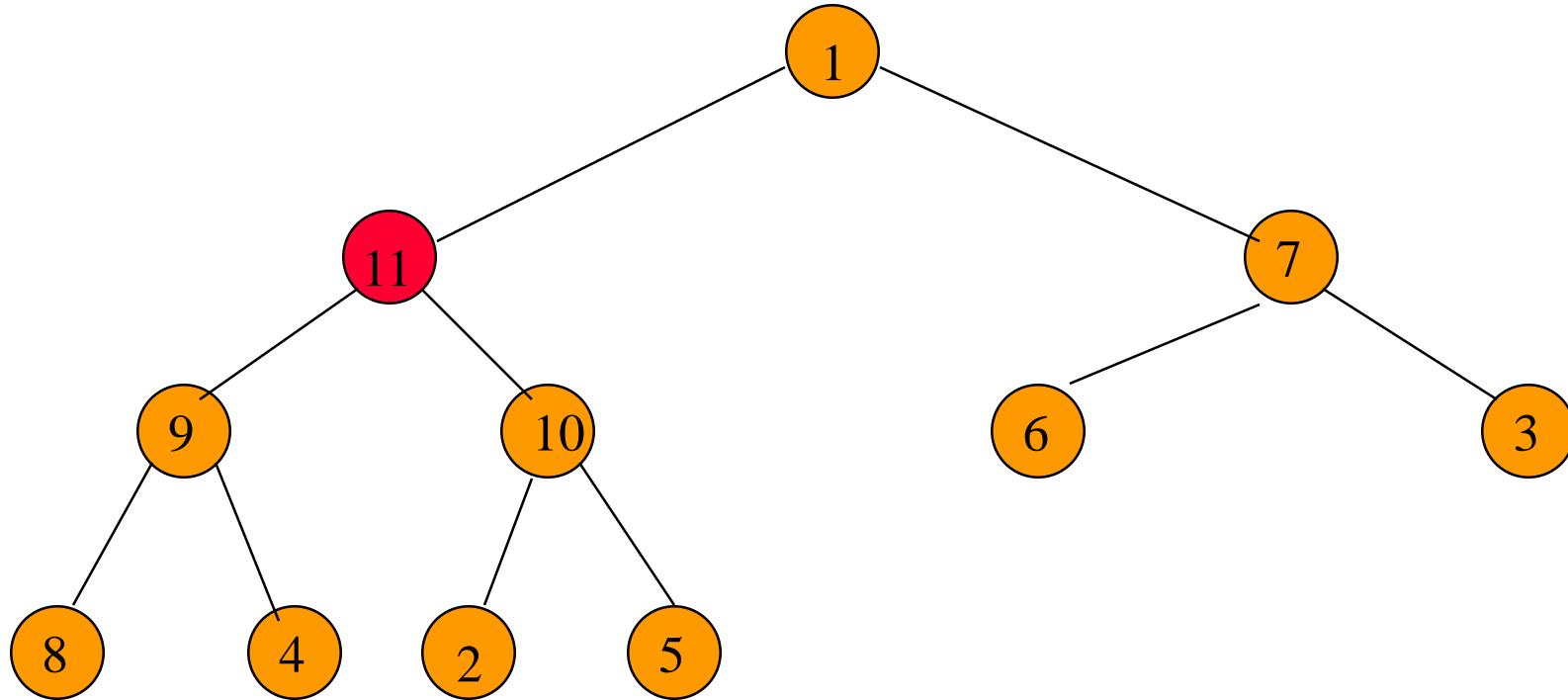
2 -н байрыг олох

# Мах овоолгыг идэхижүүлэх



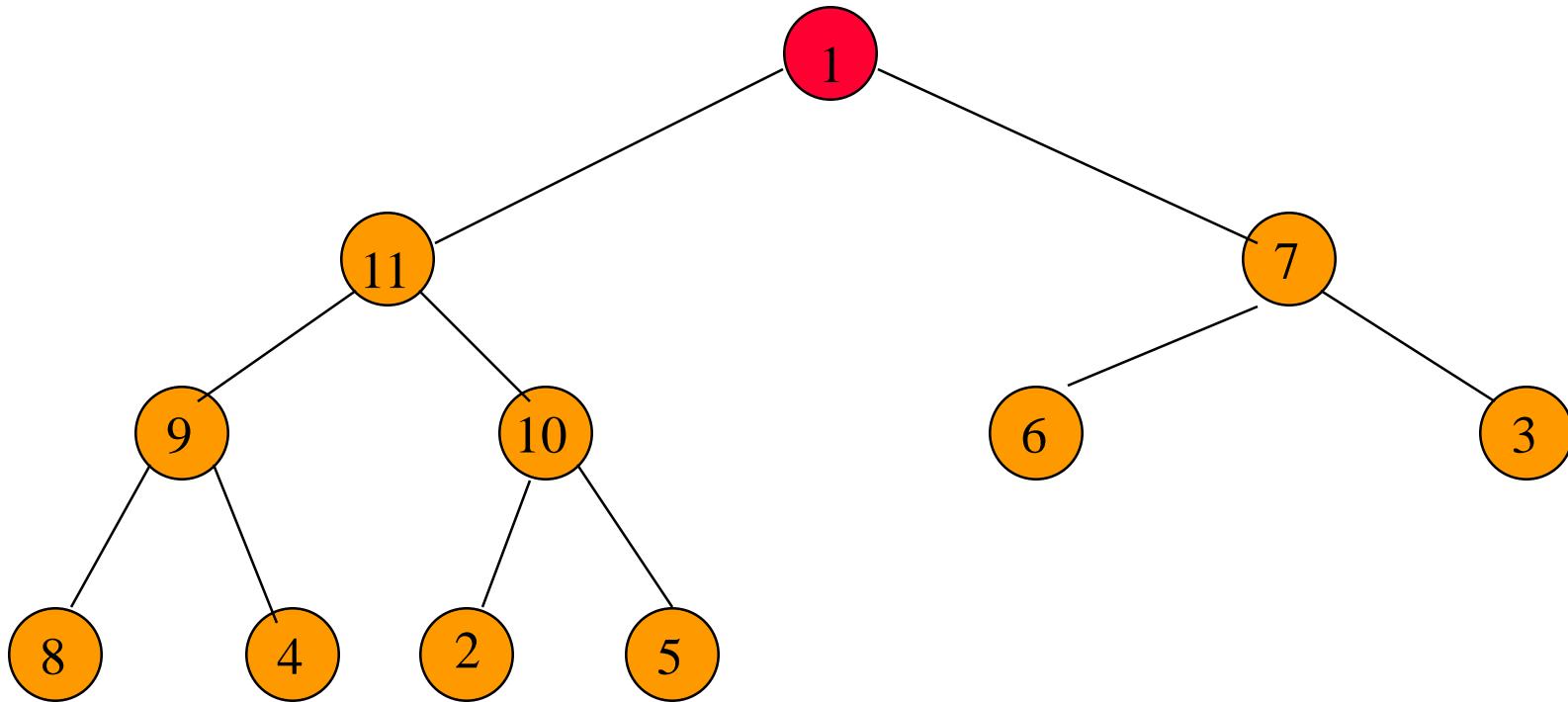
2 -н байрыг олох

# Мах овоолгыг идэхижүүлэх



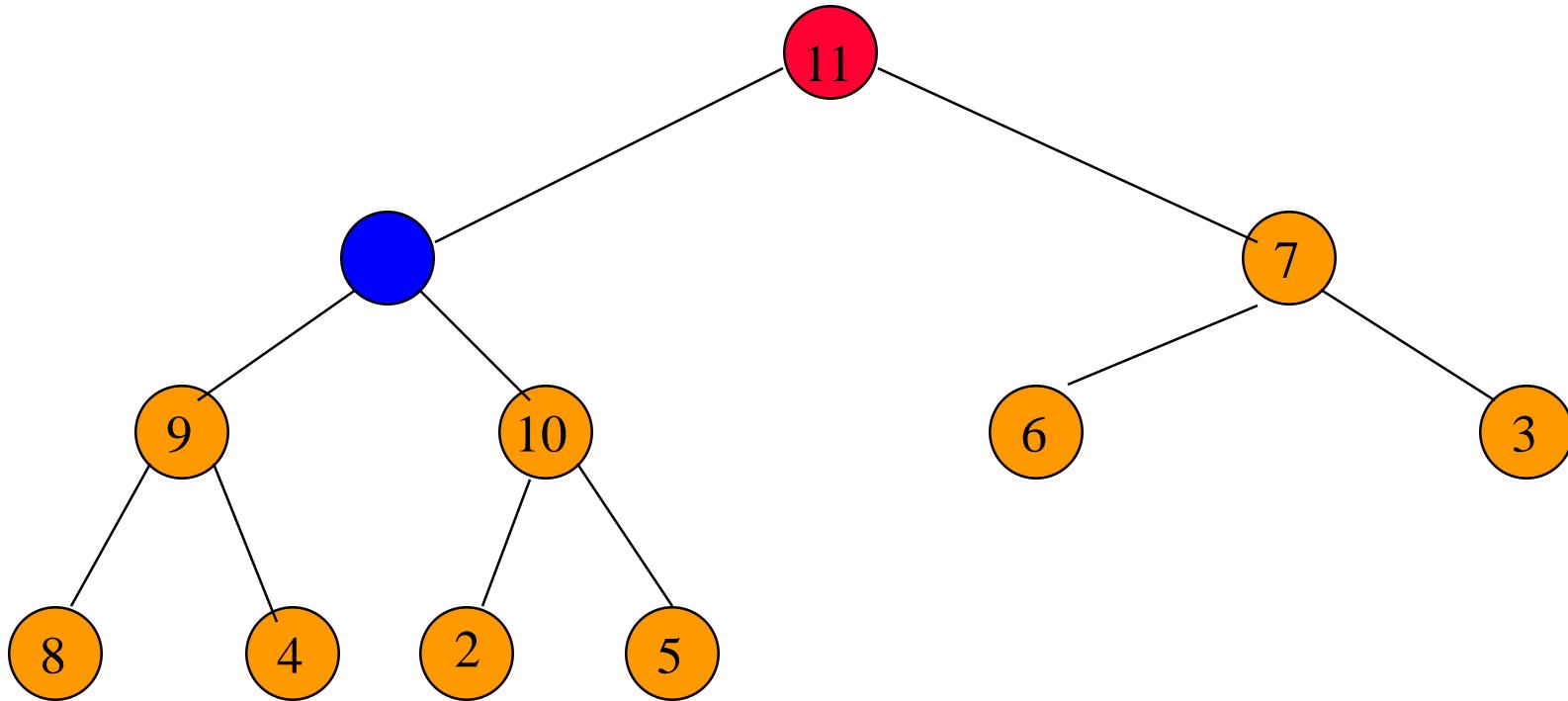
Гүйцлээ, массивын дараачийн доод байршил руу  
явна.

# Мах овоолгыг идэхижүүлэх



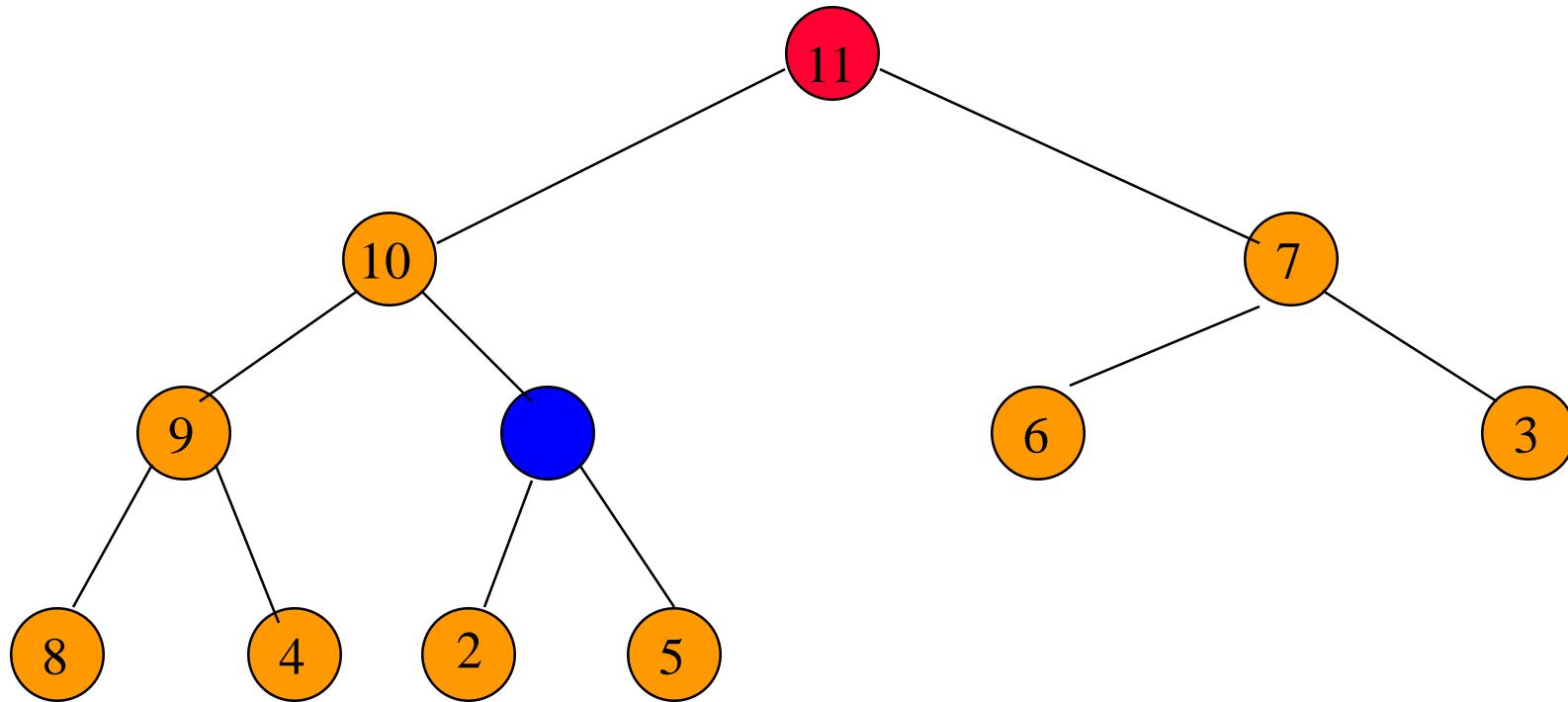
1 -н байрыг олох

# Мах овоолгыг идэхижүүлэх



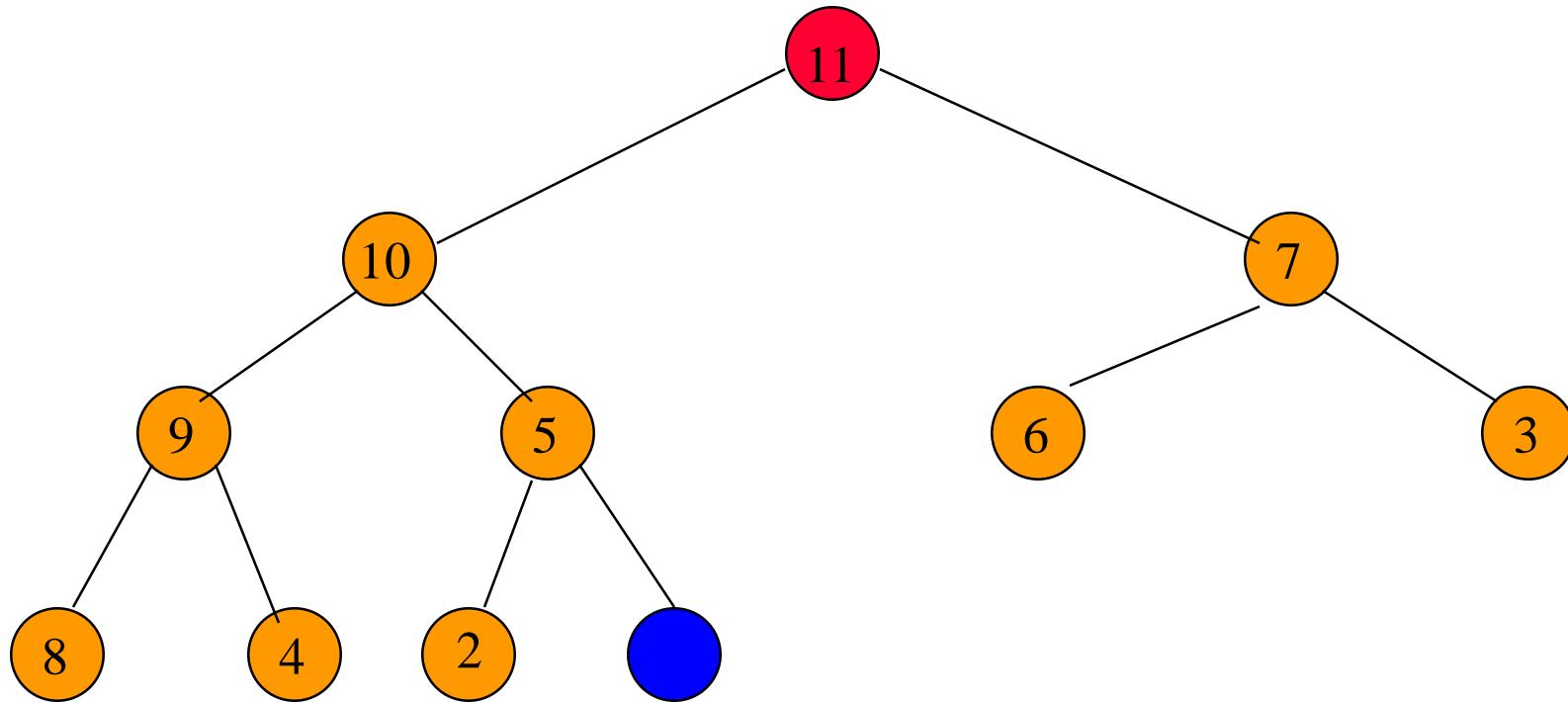
1 -н байрыг олох

# Мах овоолгыг идэхижүүлэх



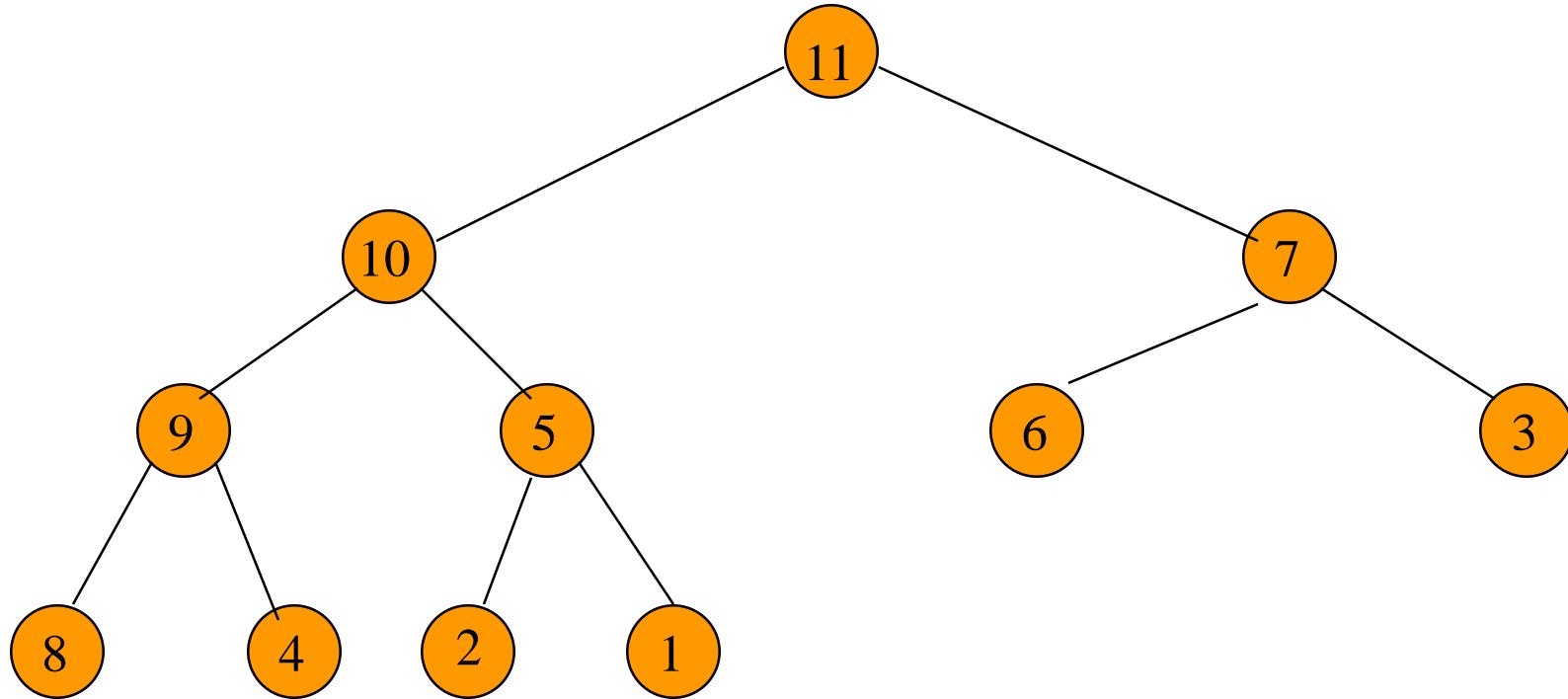
1 -н байрыг олох

# Мах овоолгыг идэхижүүлэх



1 -н байрыг олох

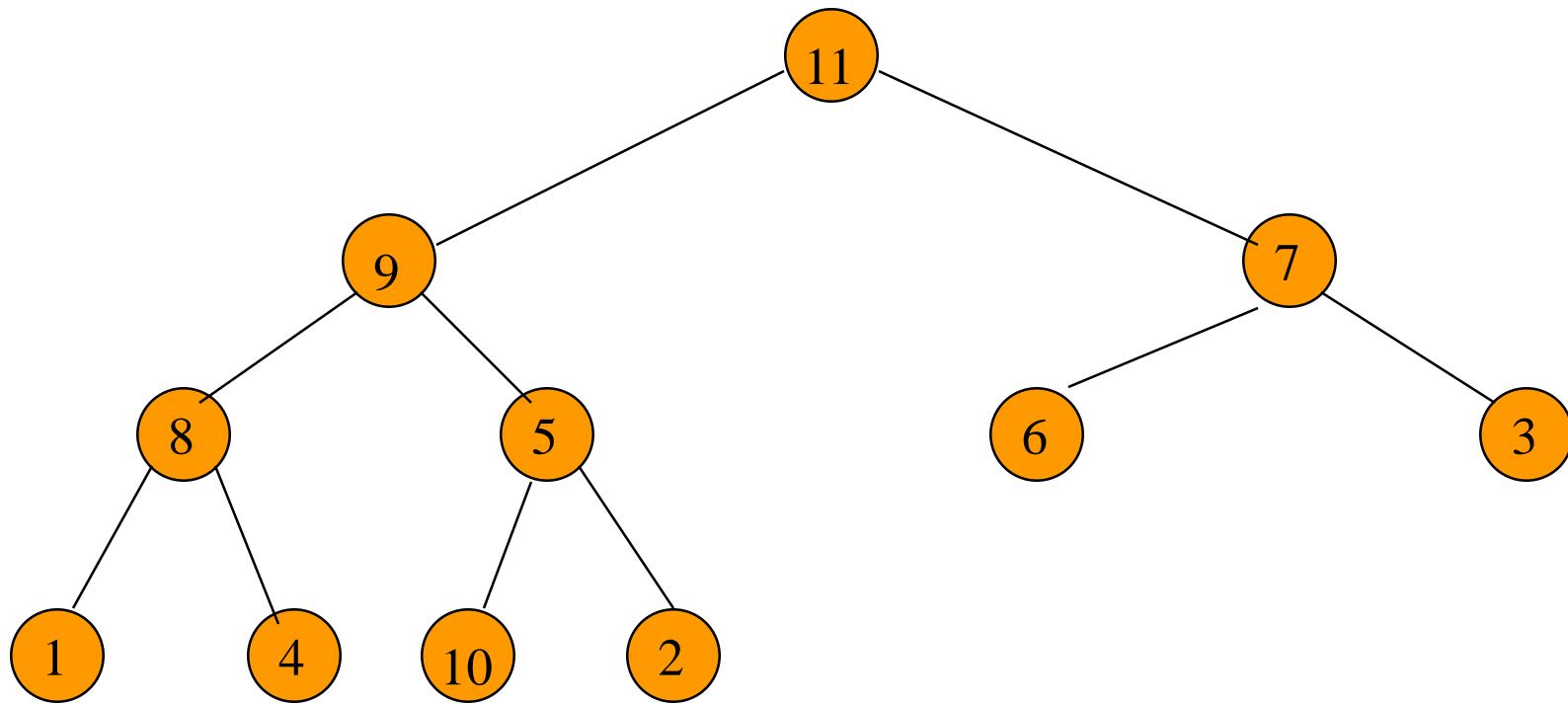
# Мах овоолгыг идэхижүүлэх



Гүйцлээ.



# Хугацаа



Овоолгын өндөр  $= h$ .

$j$  түвшинд үндэстэй дэд моднуудын тоо  $\leq 2^{j-1}$ .

Дэд мод бурийн хугацаа  $O(h-j+1)$ .



# Хугацаа

$j$  түвшний моднуудын хугацаа  $\leq 2^{j-1}(h-j+1) = t(j)$ .

Нийт хугацаа  $t(1) + t(2) + \dots + t(h-1) = O(n)$ .

# Зүүний-Leftist мод

Холбоостой хоёртын мод.

Овоолгын хийж чадах бүхнийг ойролцоо  
хугацаанд хийж чадна.

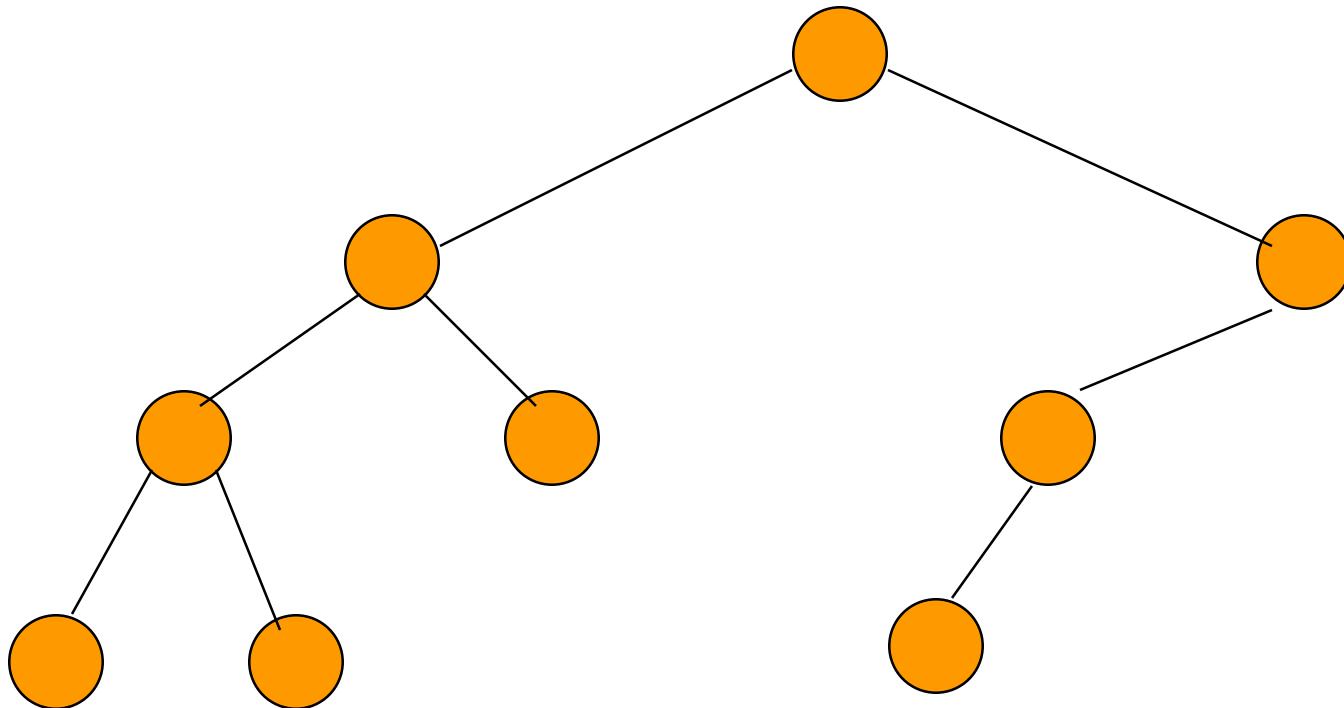
Хоёр зүүний модны эрэмбэтэй дарааллыг  
нэгтгэхэд  $O(\log n)$  хугацаа шаардана.

# Өргөтгөсөн хоёртын мод

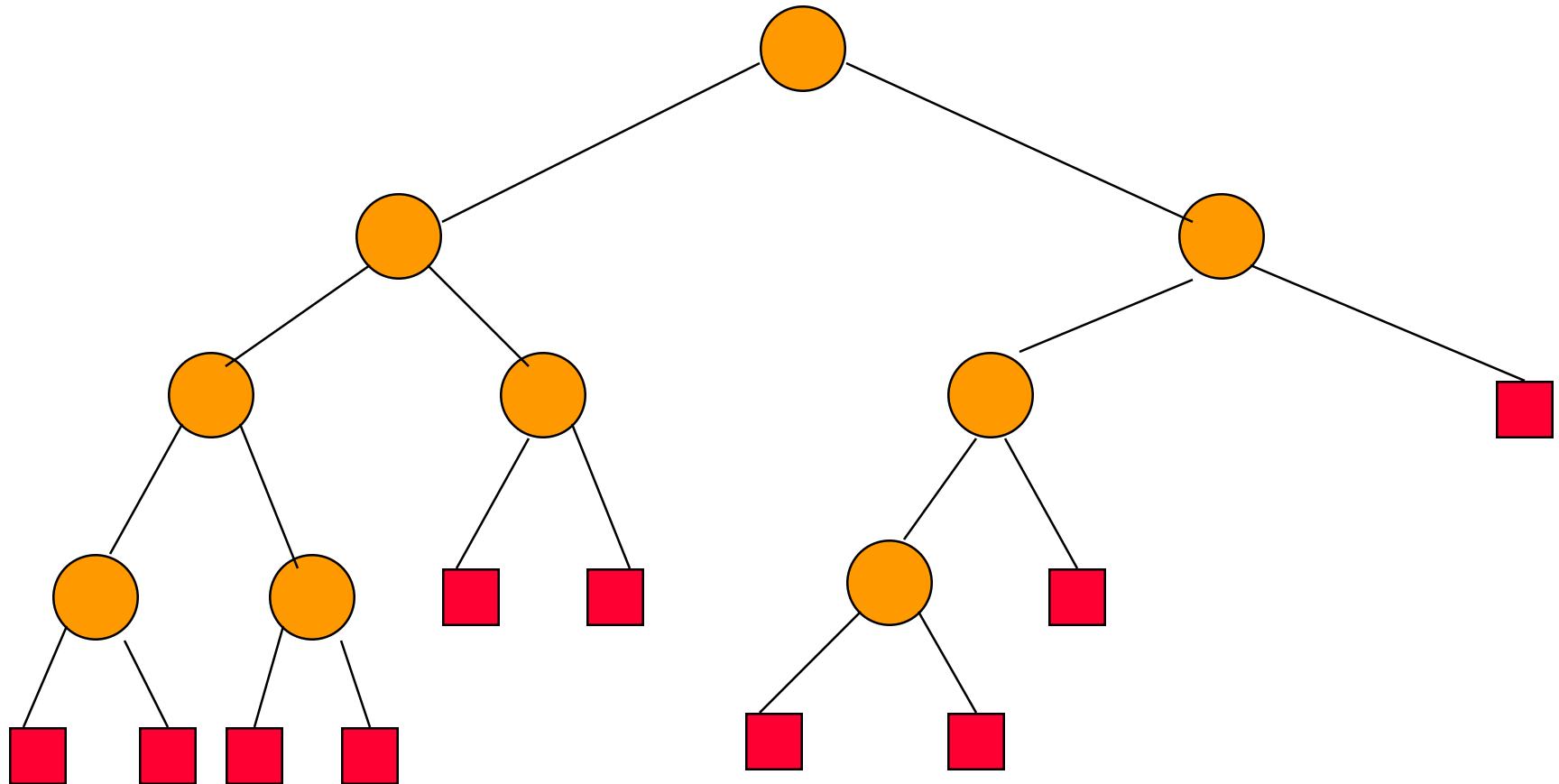
Дурын хоёртын модноос эхлээд  
хоосон дэд мод үүсгэж гадны  
зангилааг нэмнэ

Үр дүнд нь **өргөтгөсөн** хоёртын  
мод үүснэ.

# Хоёртын мод



# Өргөтгөсөн хоёртын мод



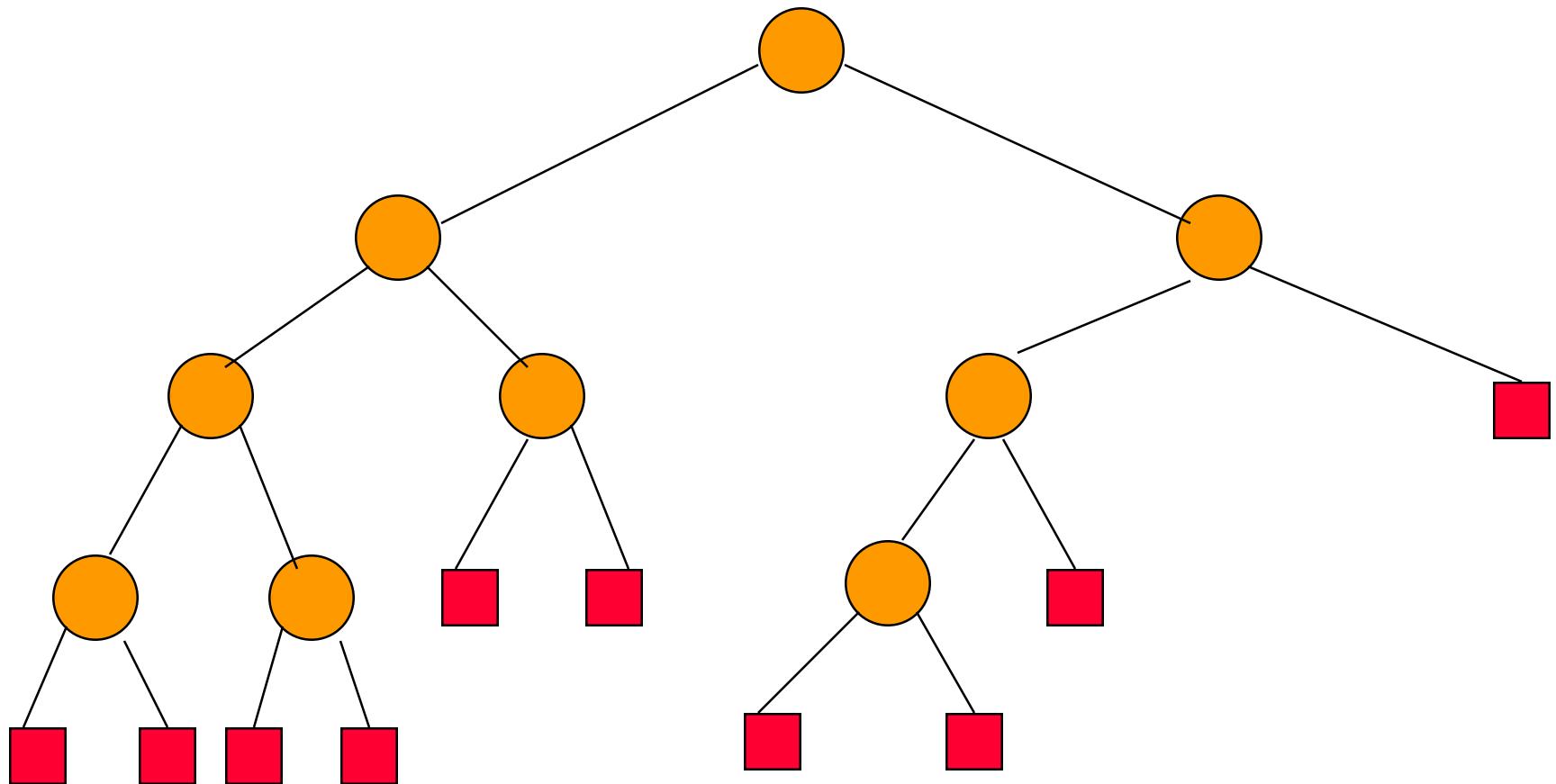
Гадны зангилааны тоо  $n+1$

# $s()$ функц

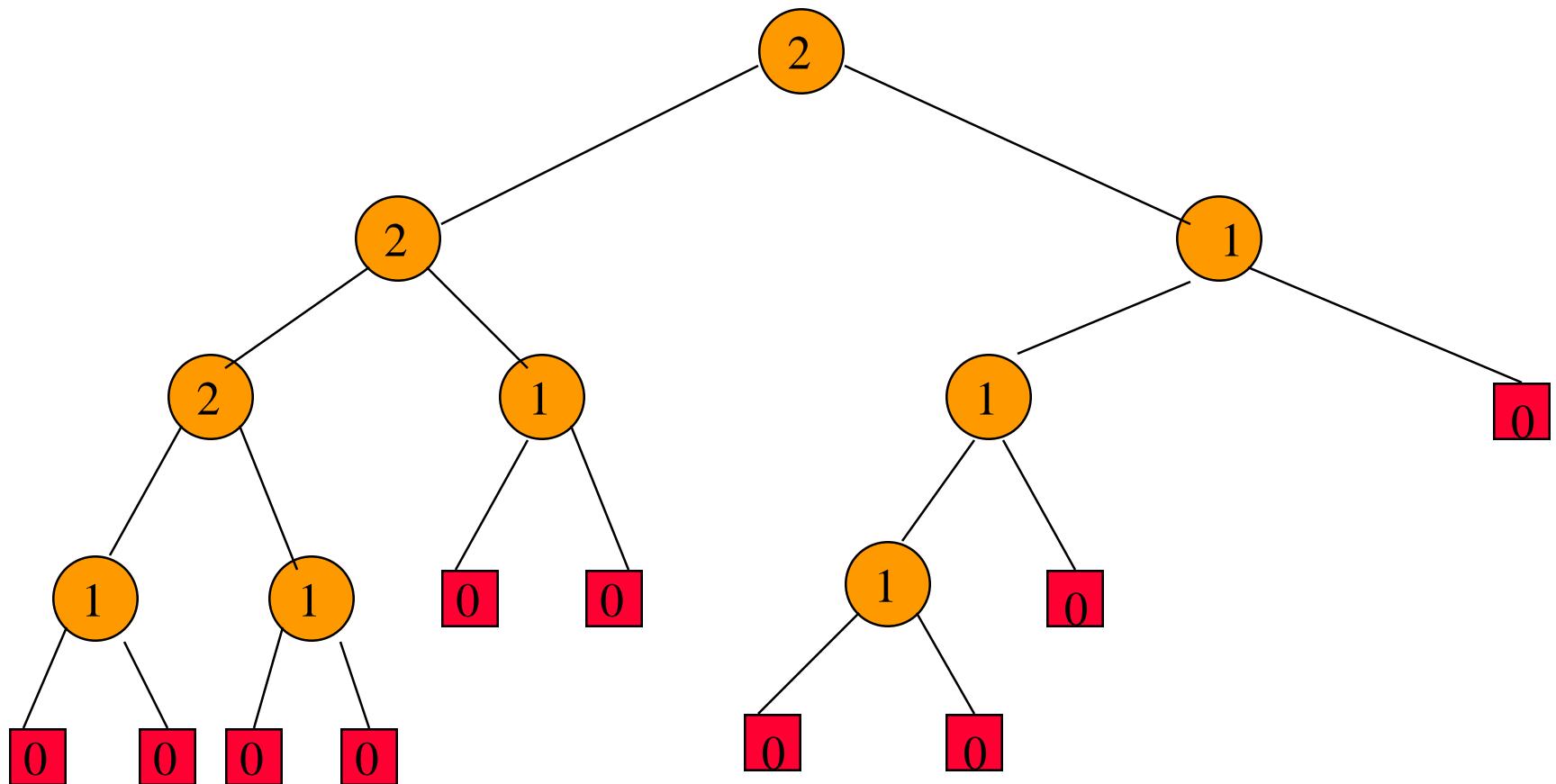
Өргөтгөсөн хоёртын модны дурын  $x$   
зангилааны хувьд,

$s(x)$  нь  $x$  –ээс түүнийг үндэс болгосон  
дэд модны гадны зангилаа хүртэлх  
хамгийн богино зам.

# s() утга - жишээ



# s() утга - жишээ



## $s()$ –Н ШИНЖ

Хэрвээ  $x$  гадны зангилаа бол  $s(x) = 0$ .

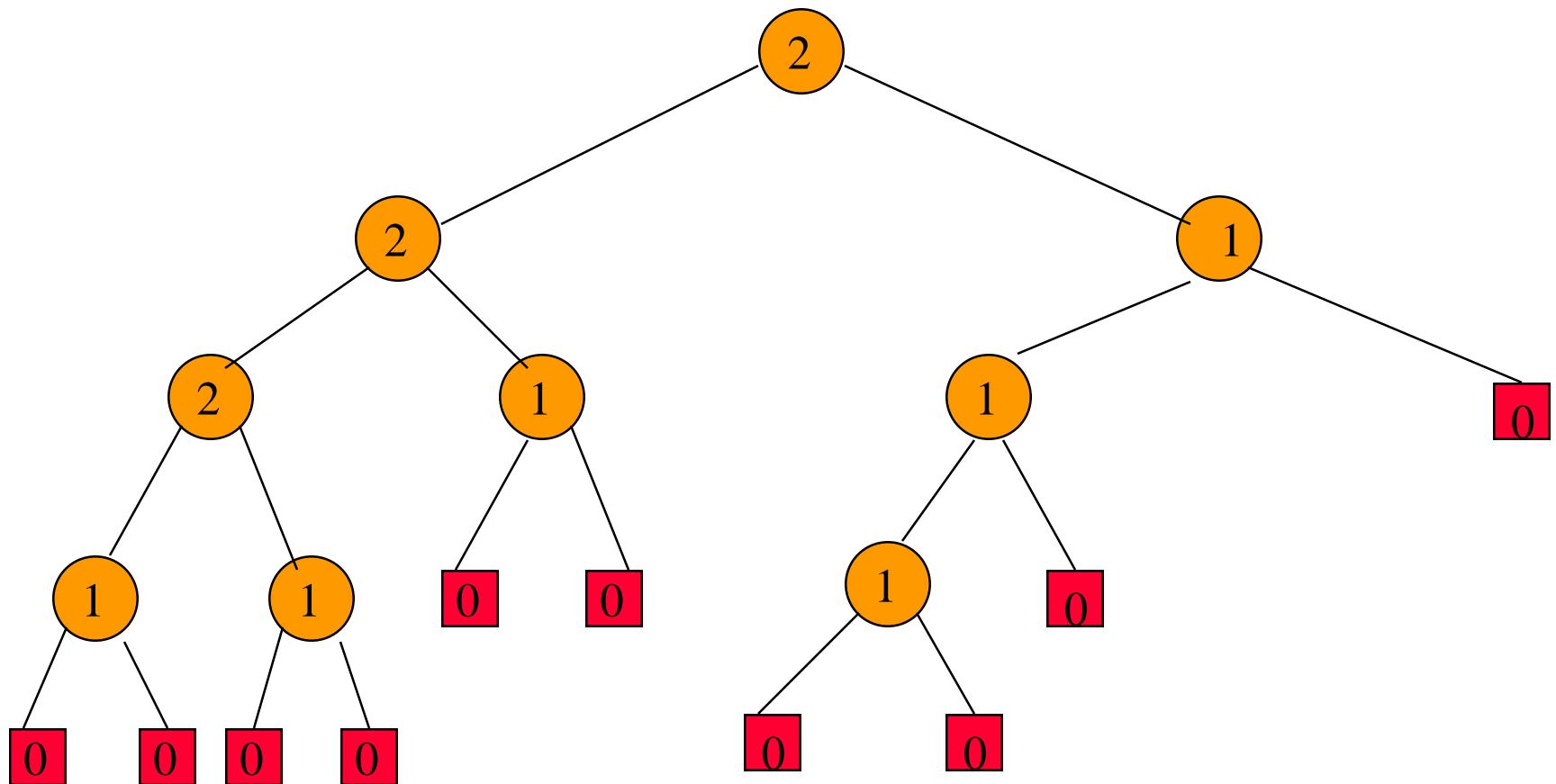
Бусад тохиолдолд,

$$s(x) = \min \{ s(\text{leftChild}(x)), \\ s(\text{rightChild}(x)) \} + 1$$

# Өндрөөр налсан зүүний мод

Хоёртын мод (өндрөөр налсан) зүүний  
мод болохын тулд бүх дотоод  
зангилаа  $x$  -н хульд  $s(\text{leftChild}(x)) \geq$   
 $s(\text{rightChild}(x))$

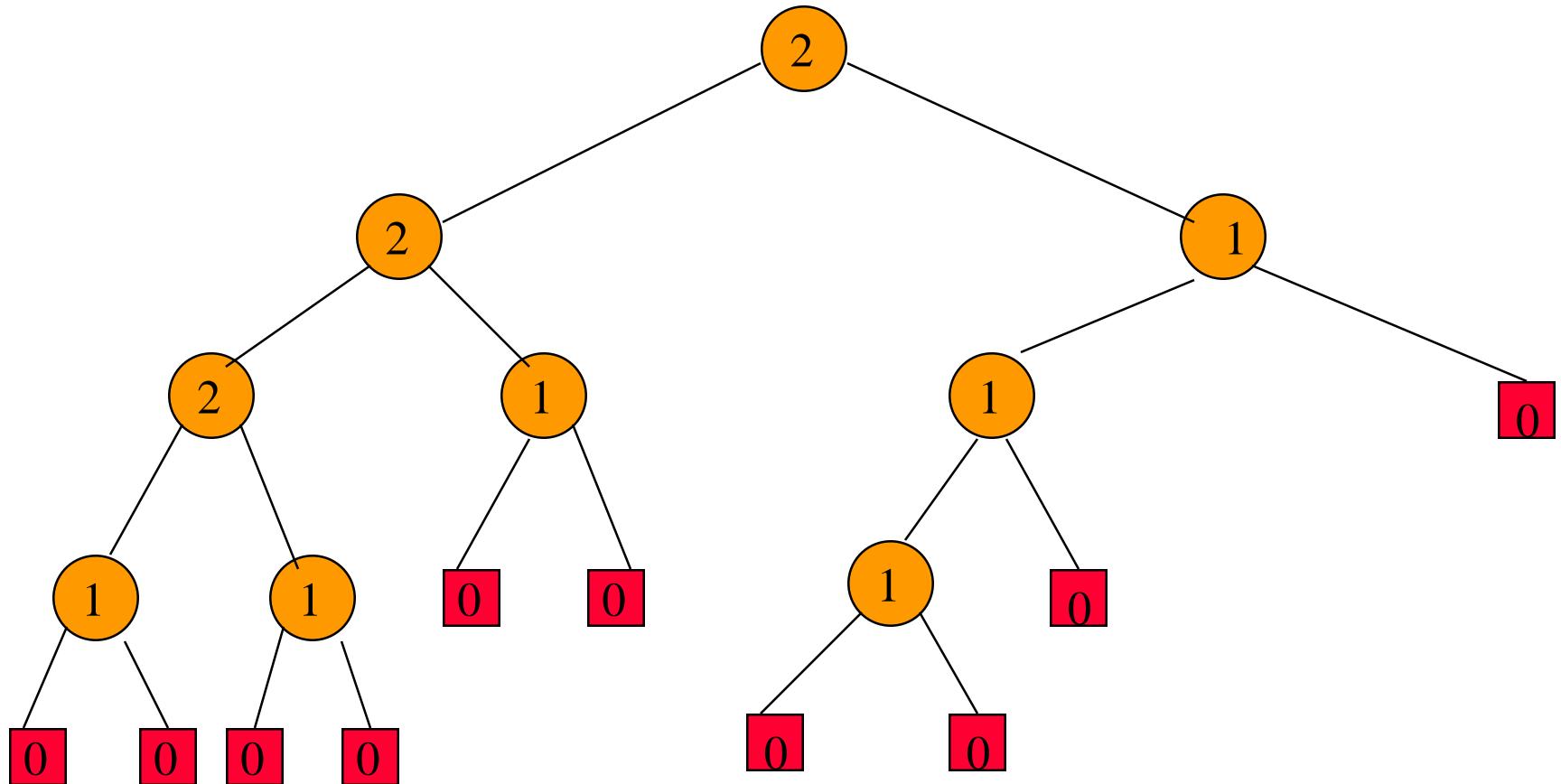
# ЗҮҮНИЙ МОД



## ЗҮҮНИЙ МОДНЫ ШИНЖ 1

Зүүний модонд, хамгийн баруун зам нь  
үндсээс гадны зангилаа хүртэлх  
хамгийн богино зам болдог бөгөөд урт  
нь **s(root)**.

# Зүүний мод



Хамгийн баруун замын урт **2**.

## ЗҮҮНИЙ МОДНЫ ШИНЖ 2

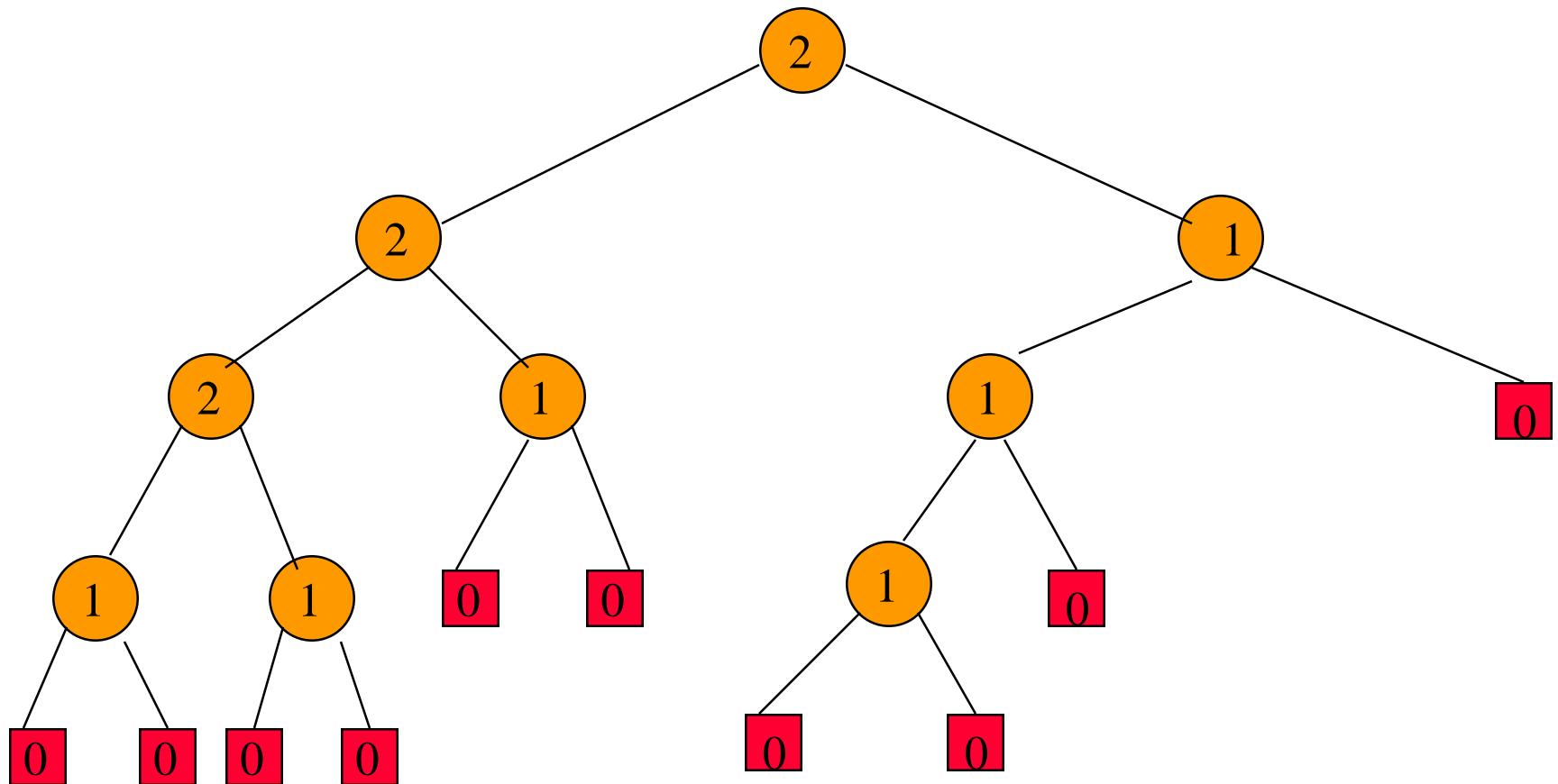
Дотоод зангилааны тоо дор хаяж

$$2^{s(\text{root})} - 1$$

1 -ээс  $s(\text{root})$  түвшингүүдэд гадны  
зангилаа байхгүй болохоор

$$s(\text{root}) \leq \log(n+1)$$

# Зүүний мод



1 , 2 -р түвшинд гадны зангилаа алга.

## ЗҮҮНИЙ МОДНЫ ШИНЖ З

Хамгийн баруун замын урт нь  $O(\log n)$ ,  
Үүнд  $n$  – зүүний модны зангилааны  
тоо.

1 , 2 –р шинжээс гарч ирсэн.

# Зүүний мод эрэмбэтэй дараалал болох

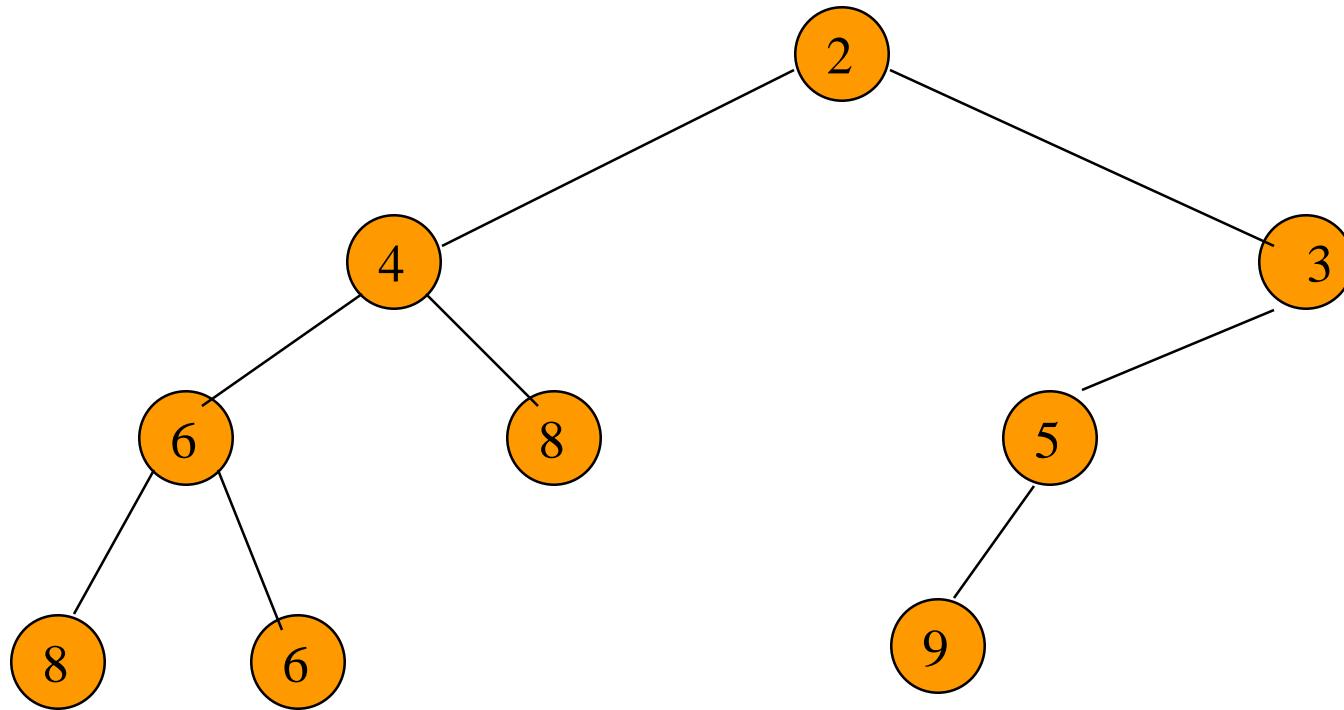
Min зүүний мод бол min мод.

Тэгэхээр min эрэмбэтэй модны байдаар ашиглана.

Max зүүний мод бол max мод.

Тэгэхээр max эрэмбэтэй модны байдаар ашиглана.

# Min зүүний мод



# Min зүүний модны зарим үйлдлүүд

put()

remove()

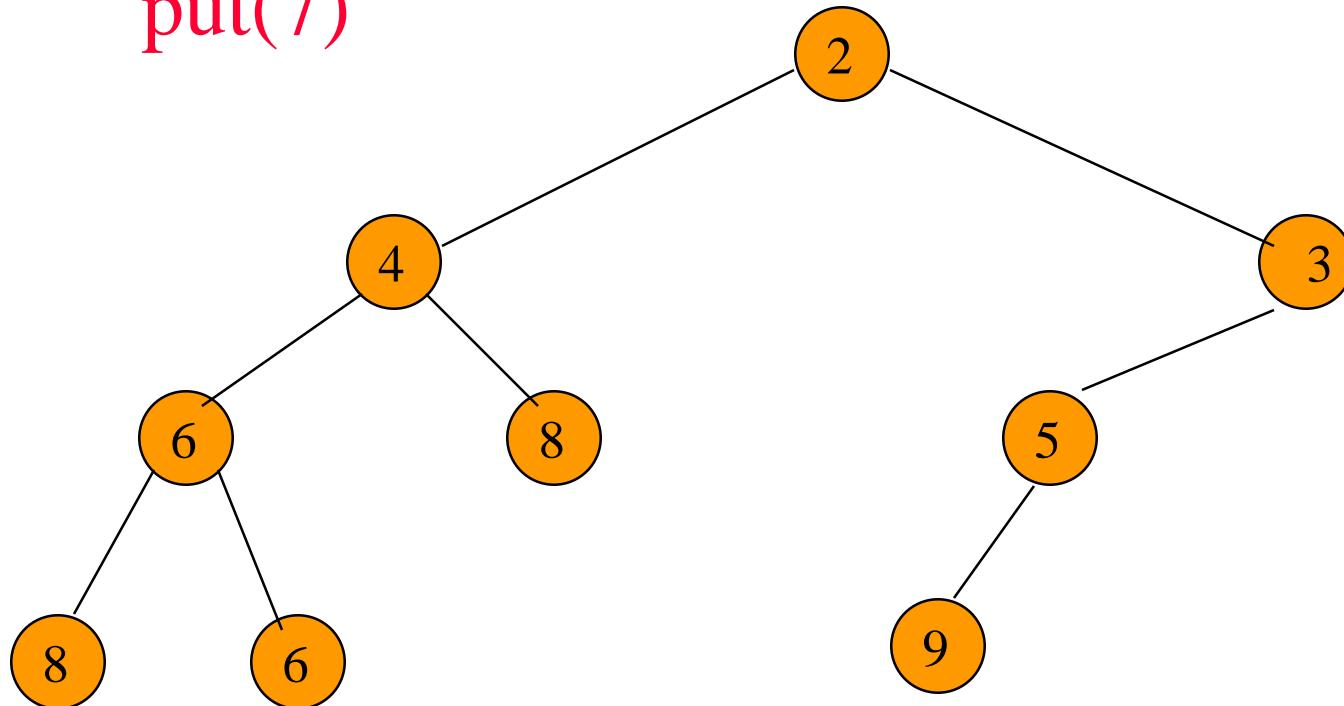
meld()

initialize()

put() , remove() -г meld() -д ашиглана

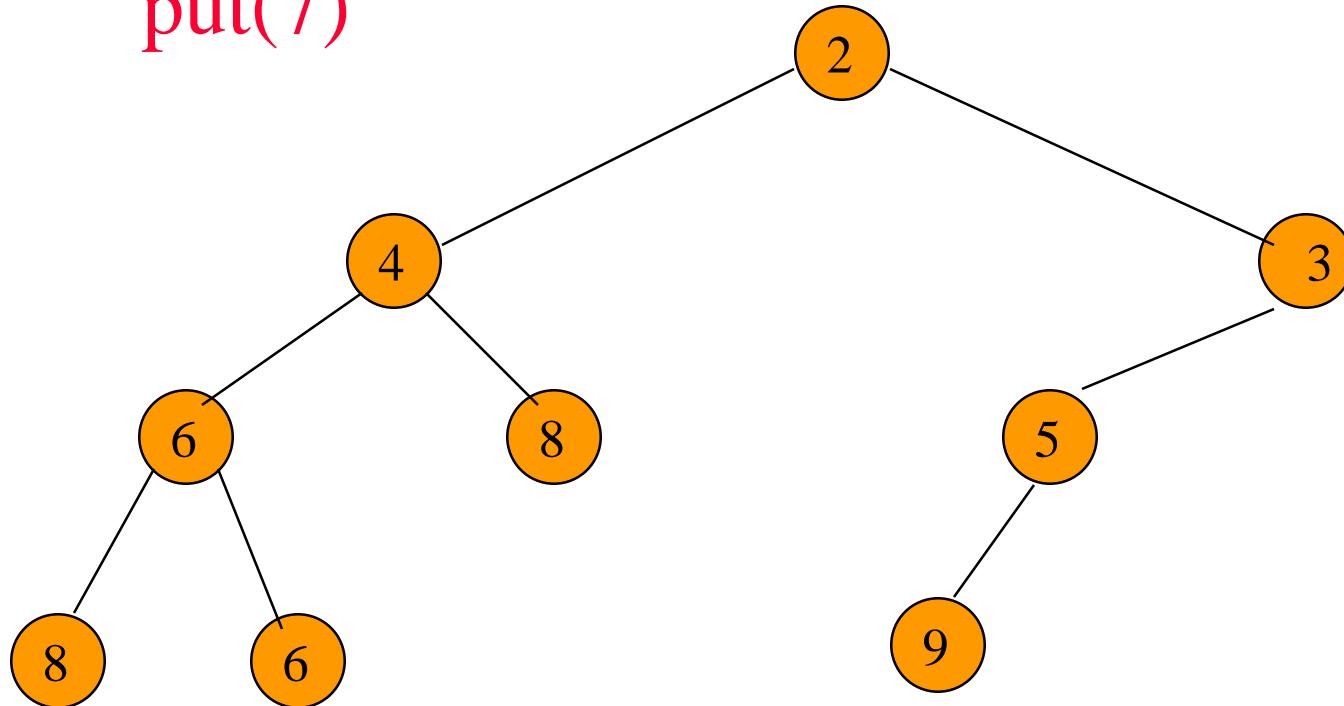
# Put Үйлдэл

put(7)



# Put Үйлдэл

put(7)

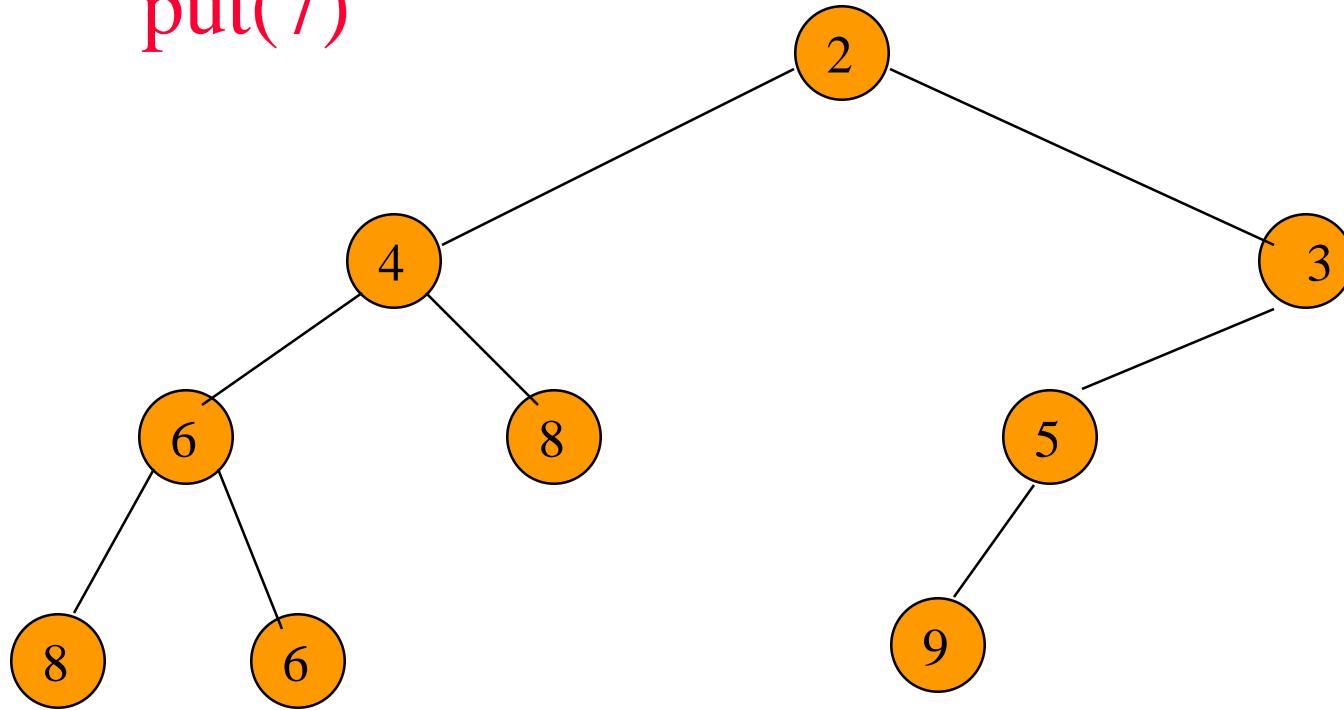


Дан зангилаатай min зүүний мод үүсгэх.

7

# Put Үйлдэл

put(7)

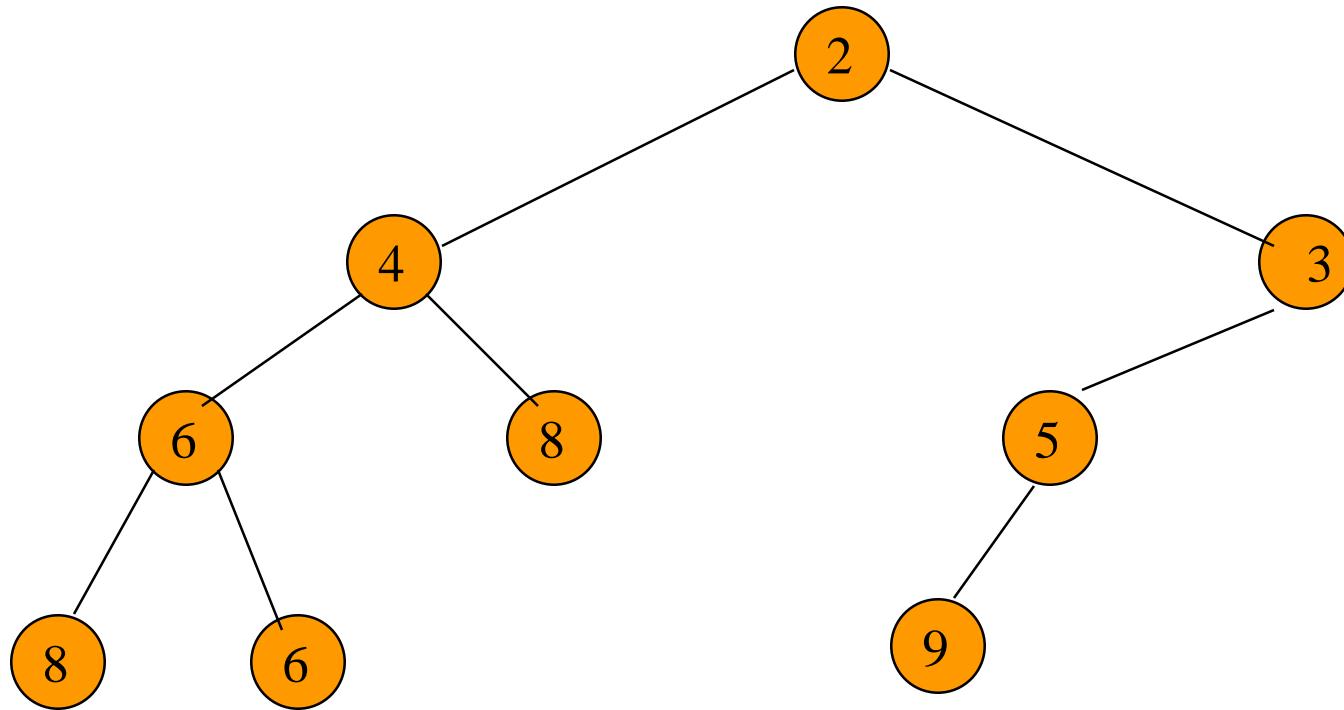


Дан зангилаатай min зүүний мод Үүсгэх.

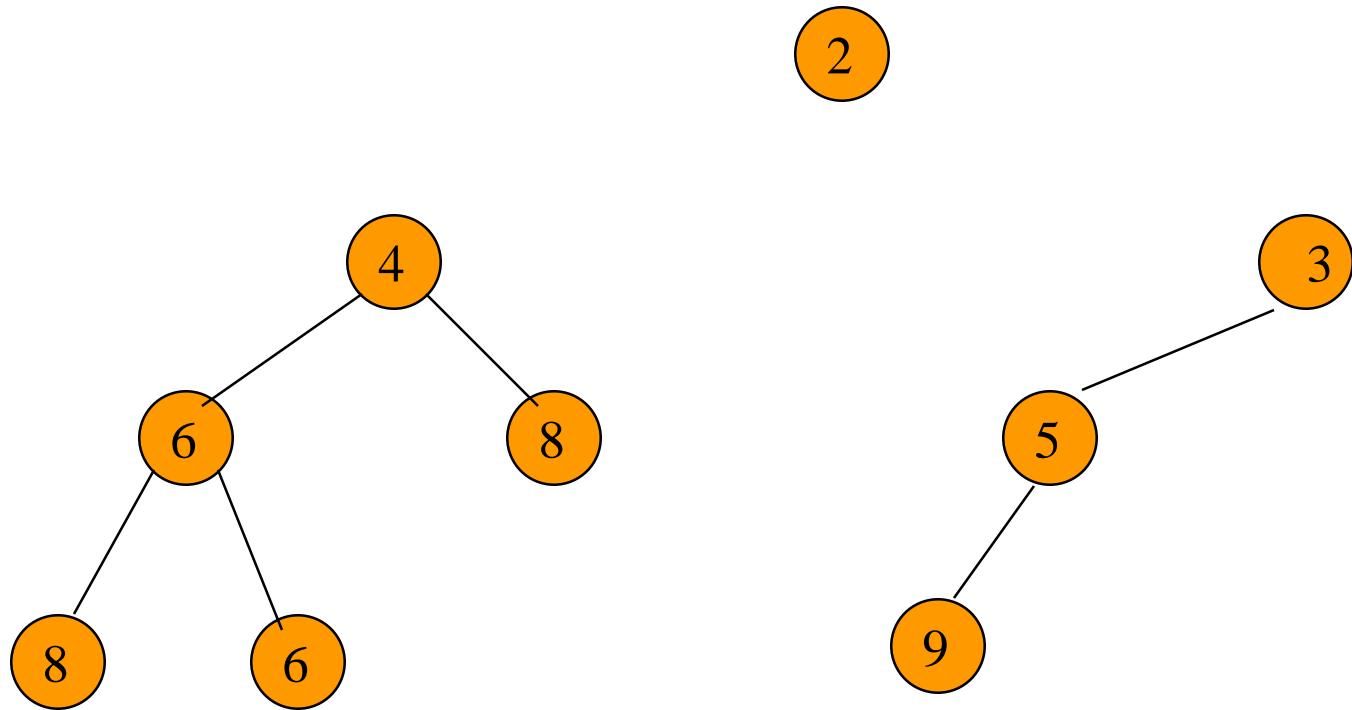
7

Хоёр min зүүний модыг нэгтгэх.

# Remove Min

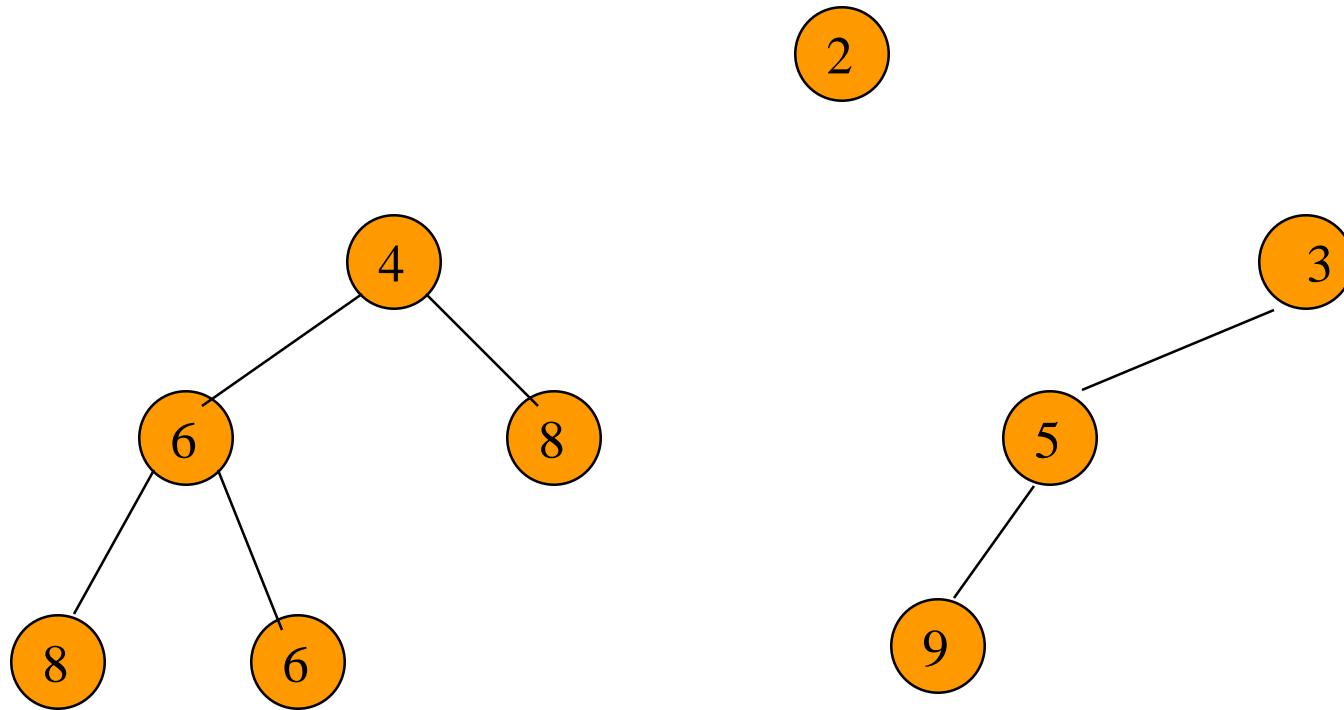


# Remove Min



Үндсийг устгах.

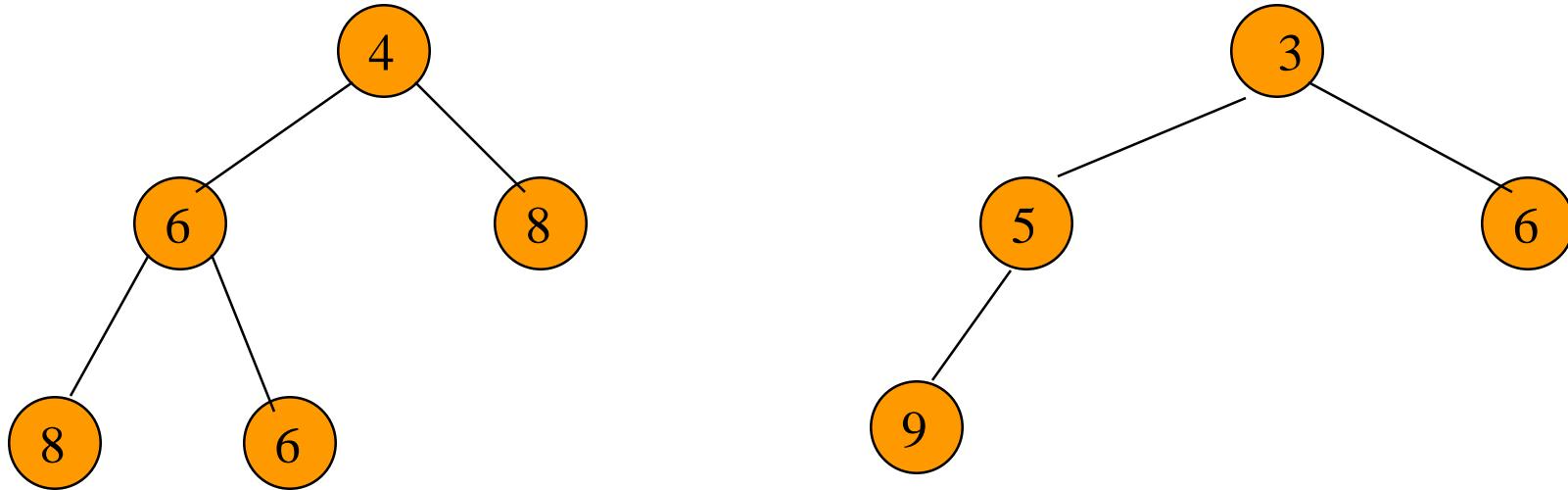
# Remove Min



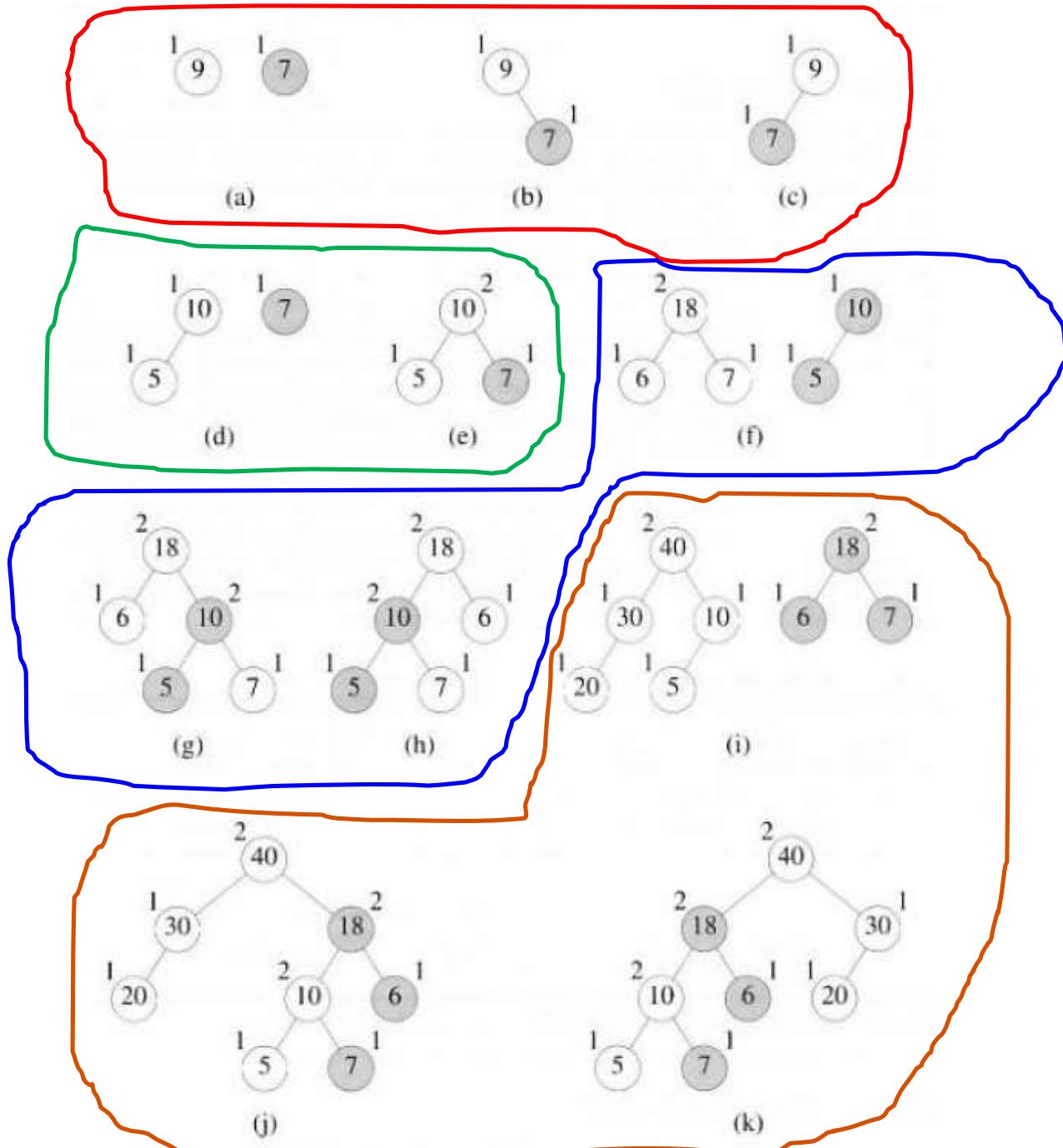
Үндсийг устгах.

Хоёр дэд модыг нийлүүлэх.

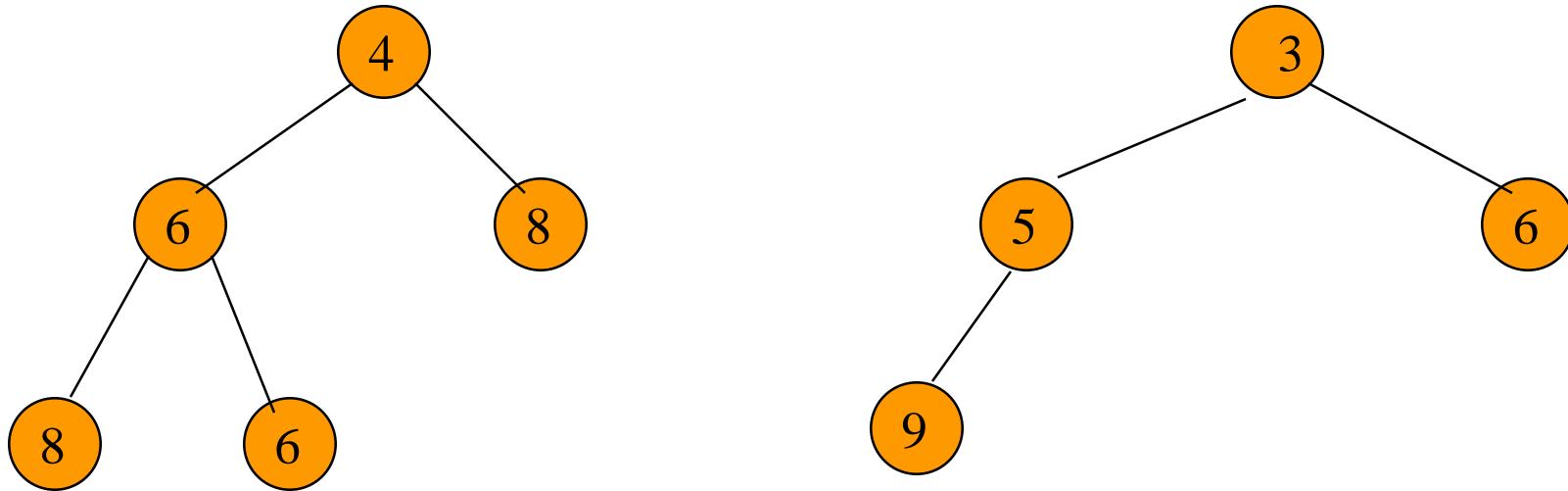
# Хоёр Min зүүний модыг нийлүүлэх



Хамгийн баруун талын замаар нэвтрэхэд хугацаа логарифмын хамаарлаар тооцогдоно.

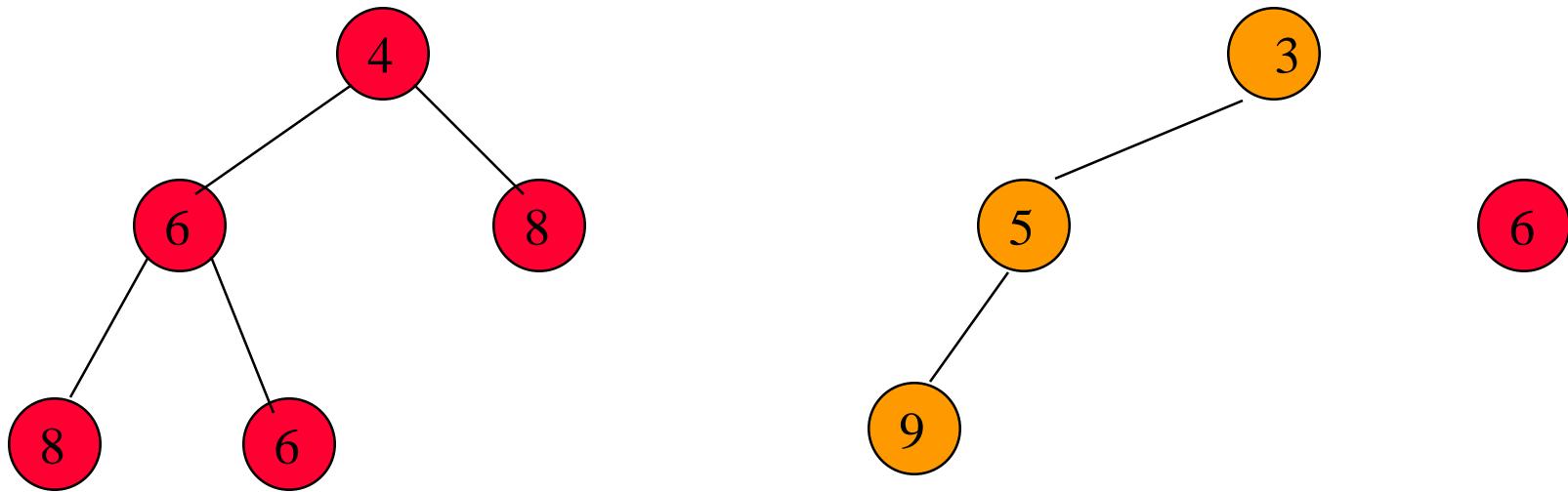


# Хоёр Min зүүний модыг нийлүүлэх



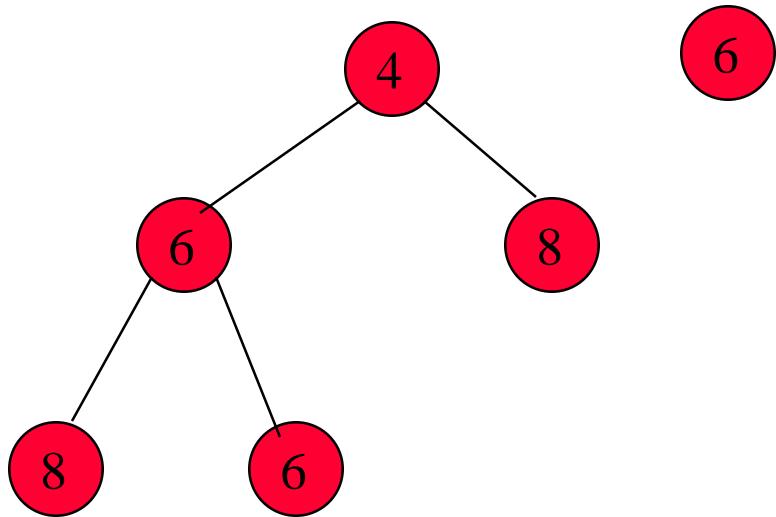
Хамгийн бага үндэстэй модны баруун дэд мод болон бусад модыг нэгтгэх.

# Хоёр Min зүүний модыг нийлүүлэх



Хамгийн бага үндэстэй модны баруун дэд мод болон бусад модыг нэгтгэх.

# Хоёр Min зүүний модыг нийлүүлэх



Хамгийн бага үндэстэй модны баруун дэд мод болон бусад модыг нэгтгэх.

# Хоёр Min зүүний модыг нийлүүлэх

8

6

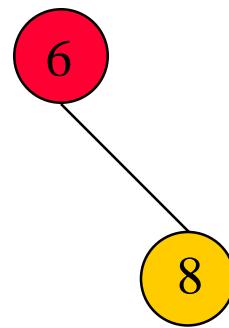
Хамгийн бага үндэстэй модны баруун дэд мод болон бусад модыг нэгтгэх.

6 –н баруун дэд мод хоосон байна. Иймд хамгийн бага үндэстэй модны баруун дэд мод болон бусад модыг нэгтгэсний үр дүн нь бусад мод болно.

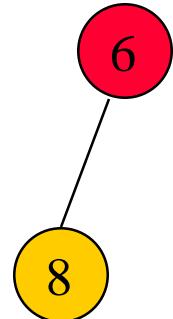
# Хоёр Min зүүний модыг нийлүүлэх



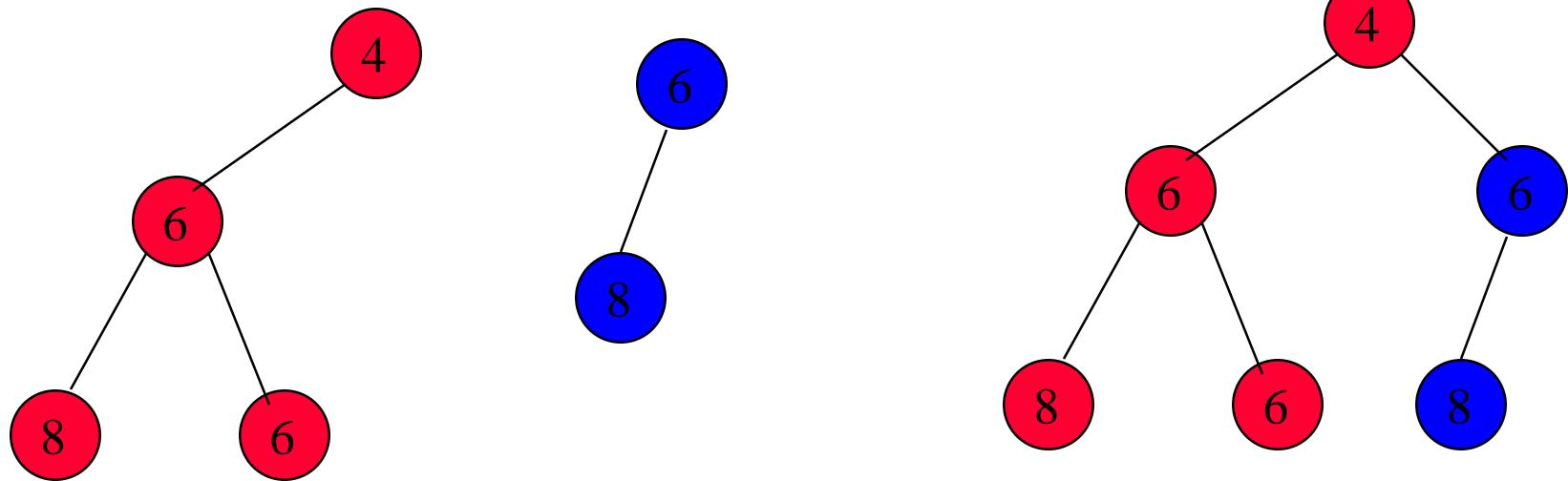
Бага үндэстэй баруун дэд модноос нэгтгэсэн дэд мод гаргах.



Хэрвээ  $s(\text{left}) < s(\text{right})$  бол зүүн, баруун дэд моднуудын байрыг солих



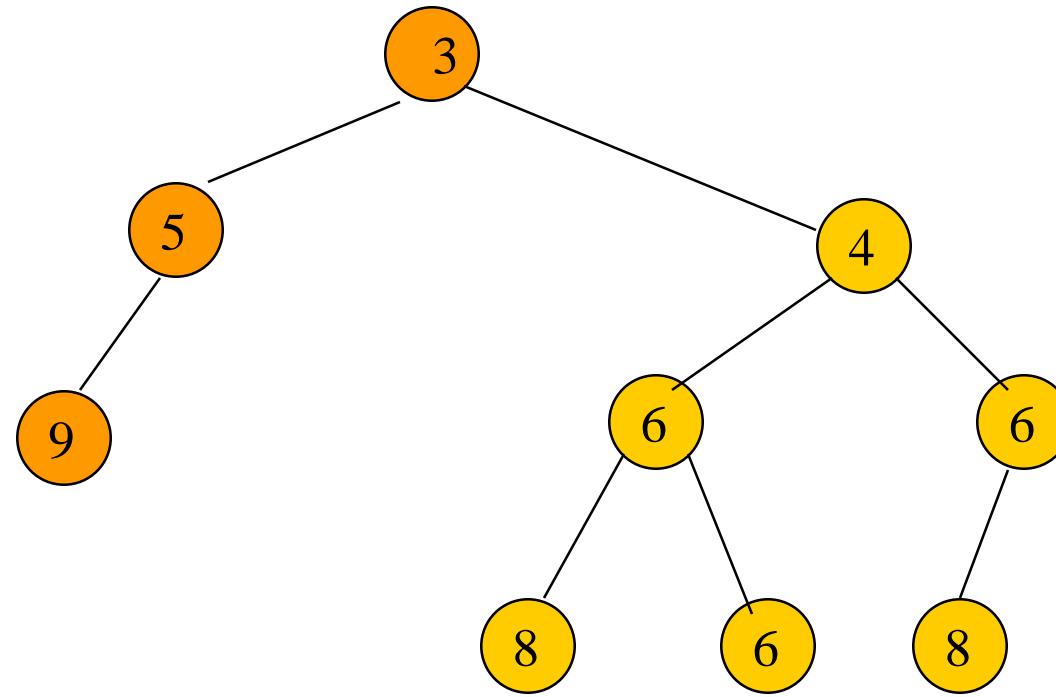
# Хоёр Min зүүний модыг нийлүүлэх



Бага үндэстэй баруун дэд модноос нэгтгэсэн дэд мод гаргах.

Хэрвээ  $s(left) < s(right)$  бол зүүн, баруун дэд моднуудын байрыг солих

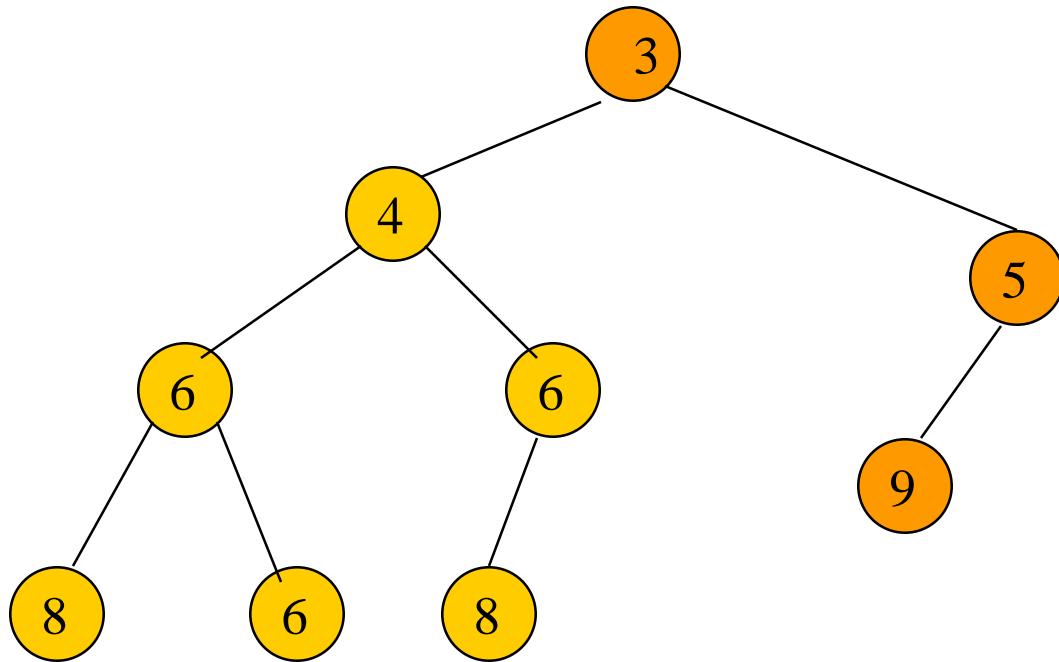
# Хоёр Min зүүний модыг нийлүүлэх



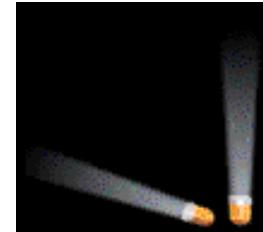
Бага үндэстэй баруун дэд модноос нэгтгэсэн дэд мод гарах.

Хэрвээ  $s(left) < s(right)$  бол зүүн, баруун дэд моднуудын байрыг солих

# Хоёр Min зүүний модыг нийлүүлэх



# ТЭМЦЭЭНИЙ МОД



Хожлын мод.

Хожигдлын мод.

# Хожлын мод

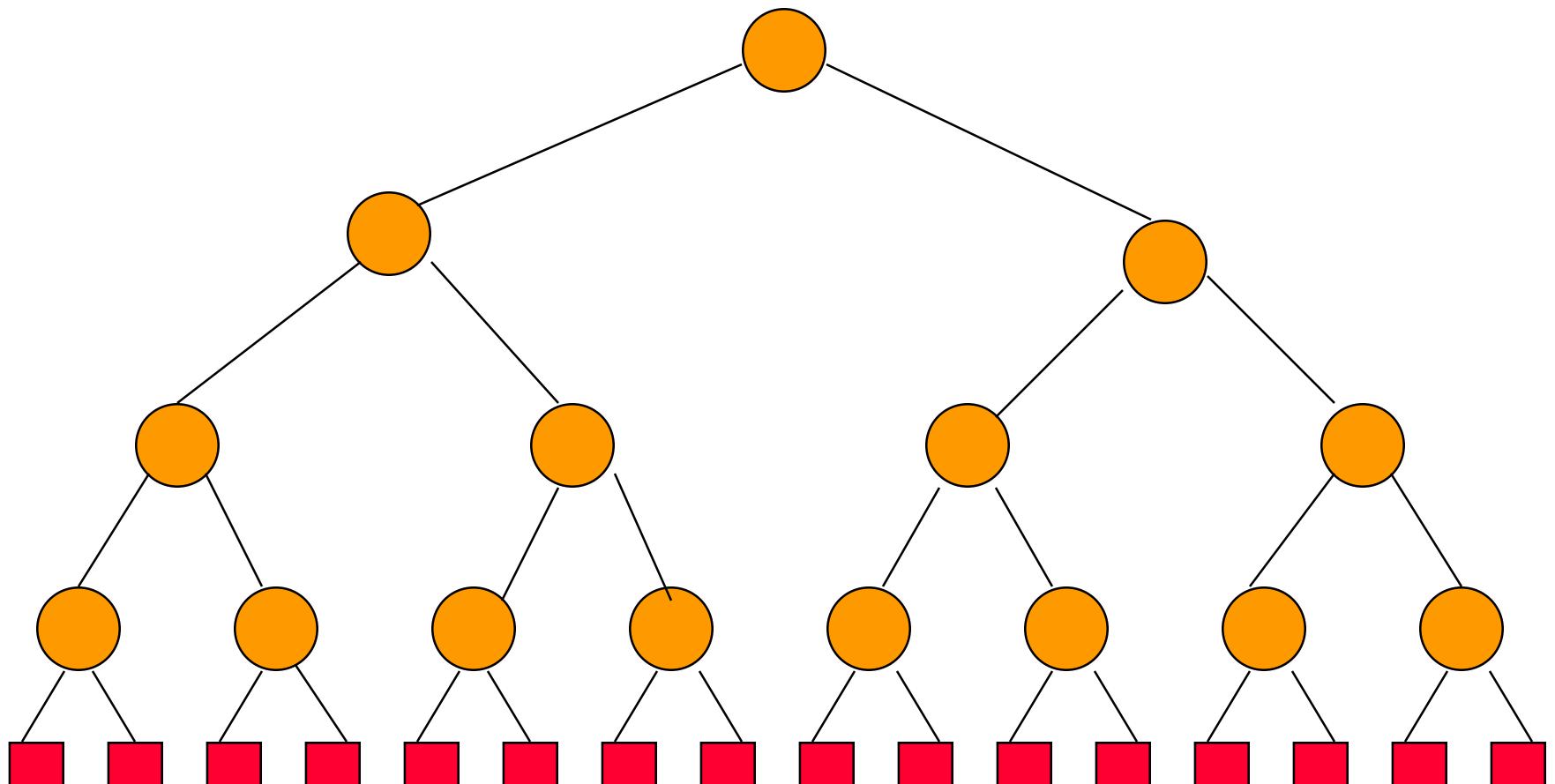
**n** гадны, **n - 1** дотоод зангилаатай төгс хоёртын мод.

Гадны зангилаа нь тэмцээний тоглогчийг илэрхийлнэ.

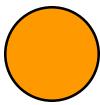
Дотоод зангилаа бүр өөрийн хүүхдүүдийн хоорондын тоглолтыг илэрхийлнэ; тоглолтын ялагч дотоод зангилаанд хадгалагдана.

Үндэс нь ялагч болно.

# 16 тоглогчийн хожлын мод

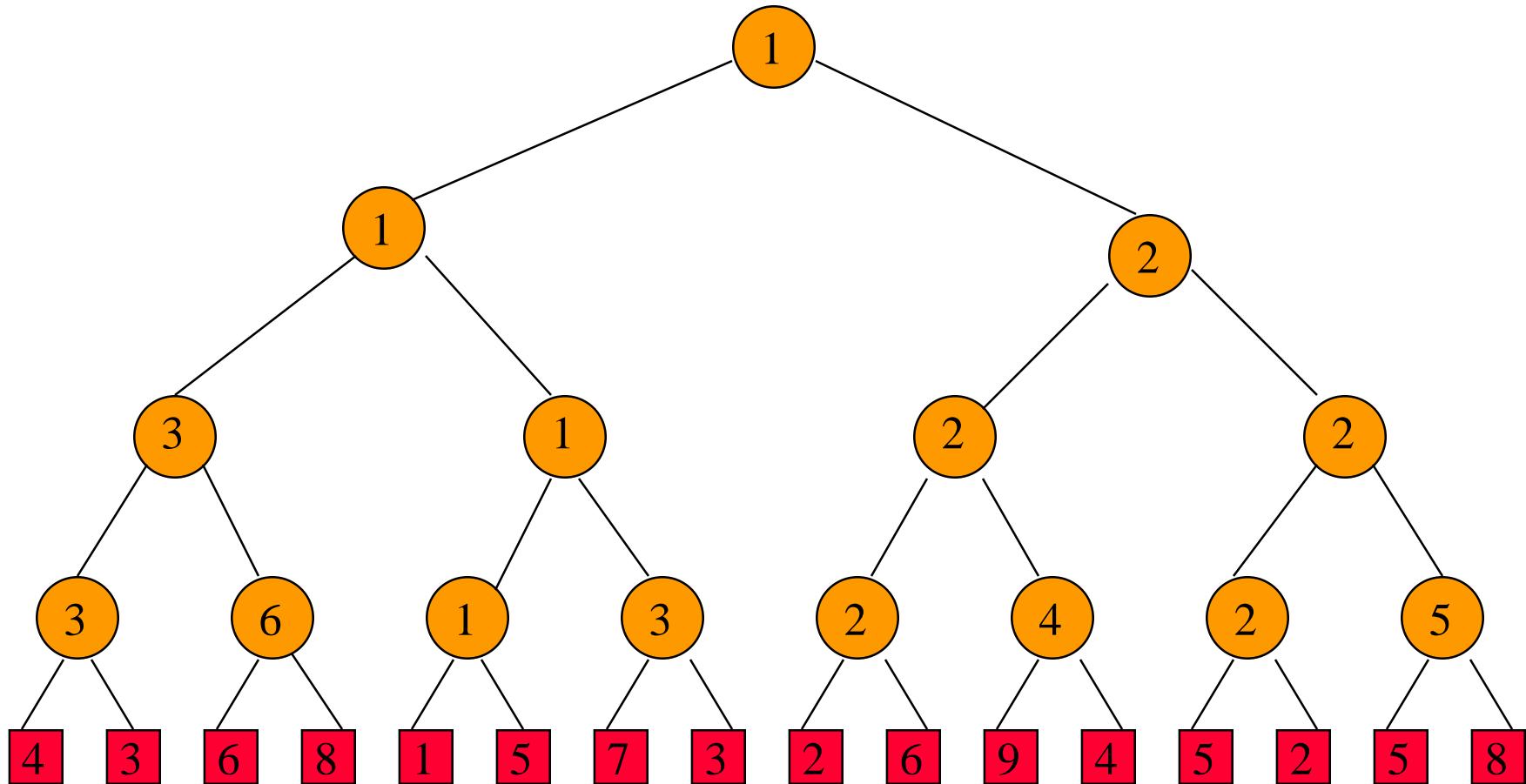


ТОГЛОГЧ



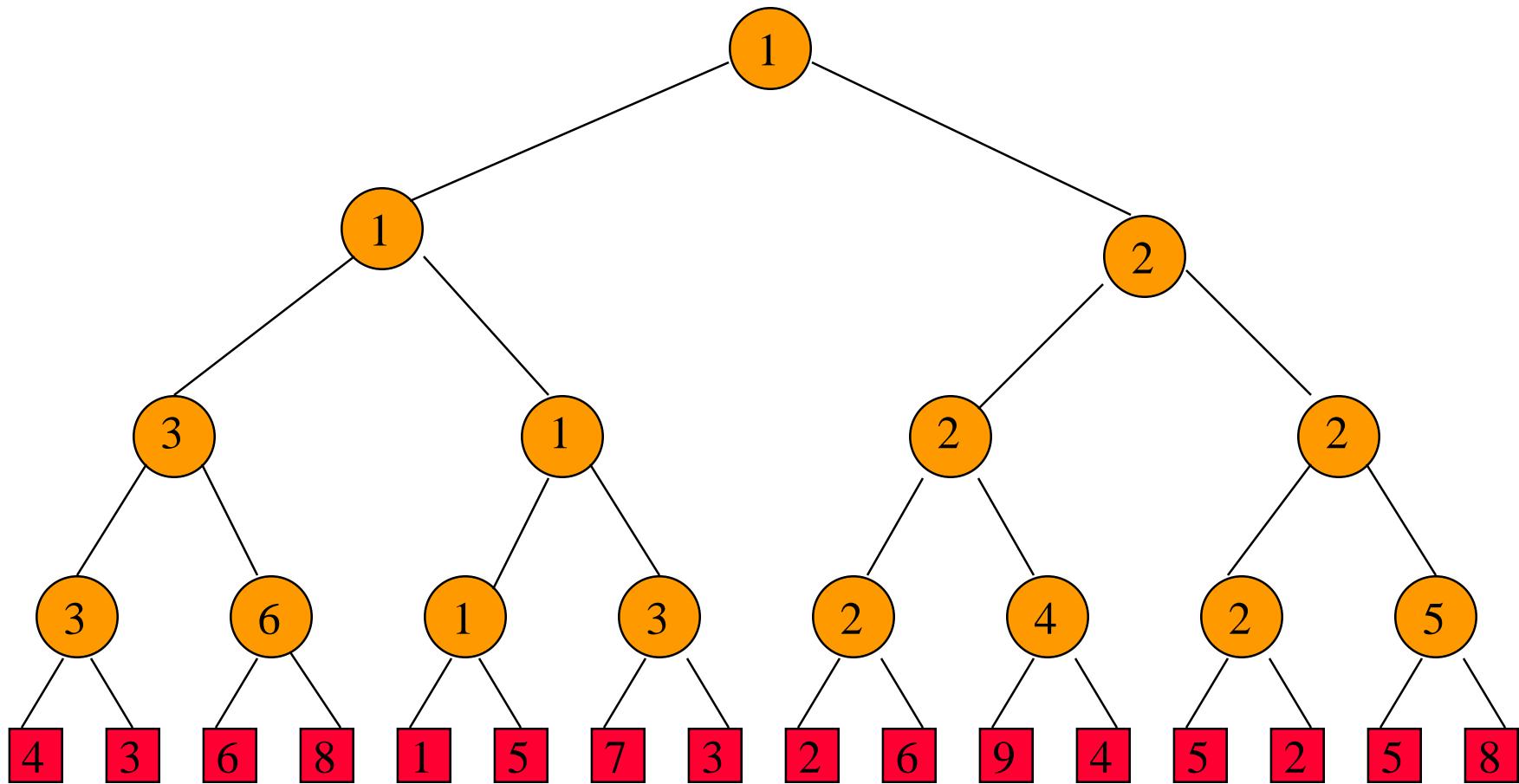
ТОГЛОЛТ

# 16 тоглогчтой хожлын мод



Бага элемент ялбал => min хожлын мод.

# 16 тоглогчтой хожлын мод



Өндөр  $\log_2 n$  (тоглогчдын түвшин ороогүй)

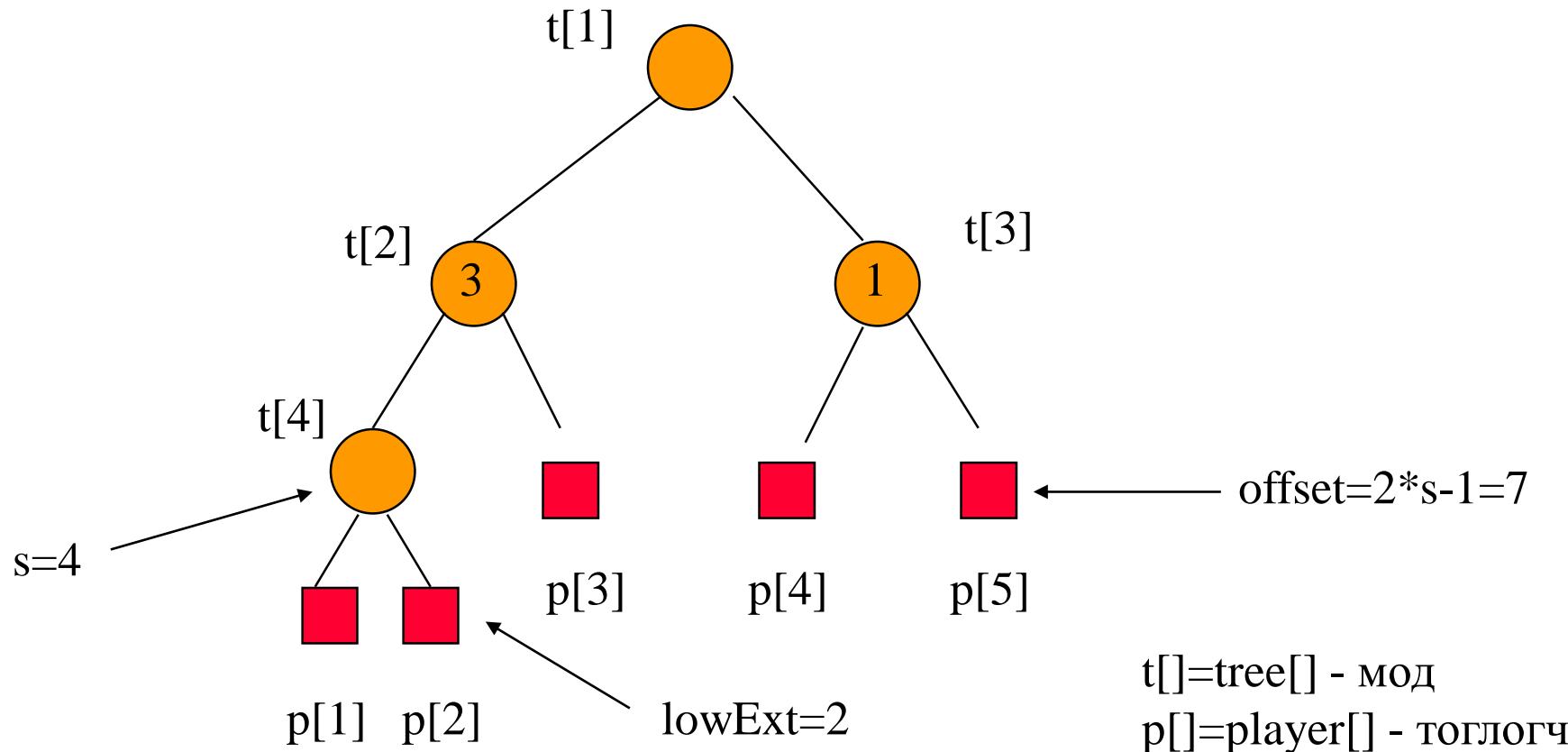
# Интерфейс WinnerTree

```
public interface WinnerTree
{
    public void initialize(Playable []
thePlayer);
    public int getWinner();
    public void rePlay(int i);
}
```

# Хожлын модны шинж

- Хэрвээ гадны зангилааны тоо **n** бол дотоод зангилааны тоо **n-1**
- Хамгийн доод түвшингийн хамгийн зүүн зангилааны дугаар  $s = 2^{\log_2(n-1)}$
- Хамгийн доод түвшиний дотоод зангилааны тоо **n-s**
- Хамгийн доод түвшний гадаад зангилааны тоо **lowExt=2(n-s)**

# Хожлын модны шинж



Тоглогч  $i$ -ийн эцэг:

$$p = (i + offset) / 2 ; \quad i \leq lowExt$$

$$p = (i - lowExt + n - 1) / 2; \quad i > lowExt$$

# Идэвхижүүлэх хугацаа

- $O(1)$  тоглолтын зангилаа бүрт шаардагдах хугацаа.
- $n - 1$  тоглолтын зангилаа.
- $O(n)$   $n$  тоглогчтой хожлын модыг идэвхижүүлэх хугацаа.

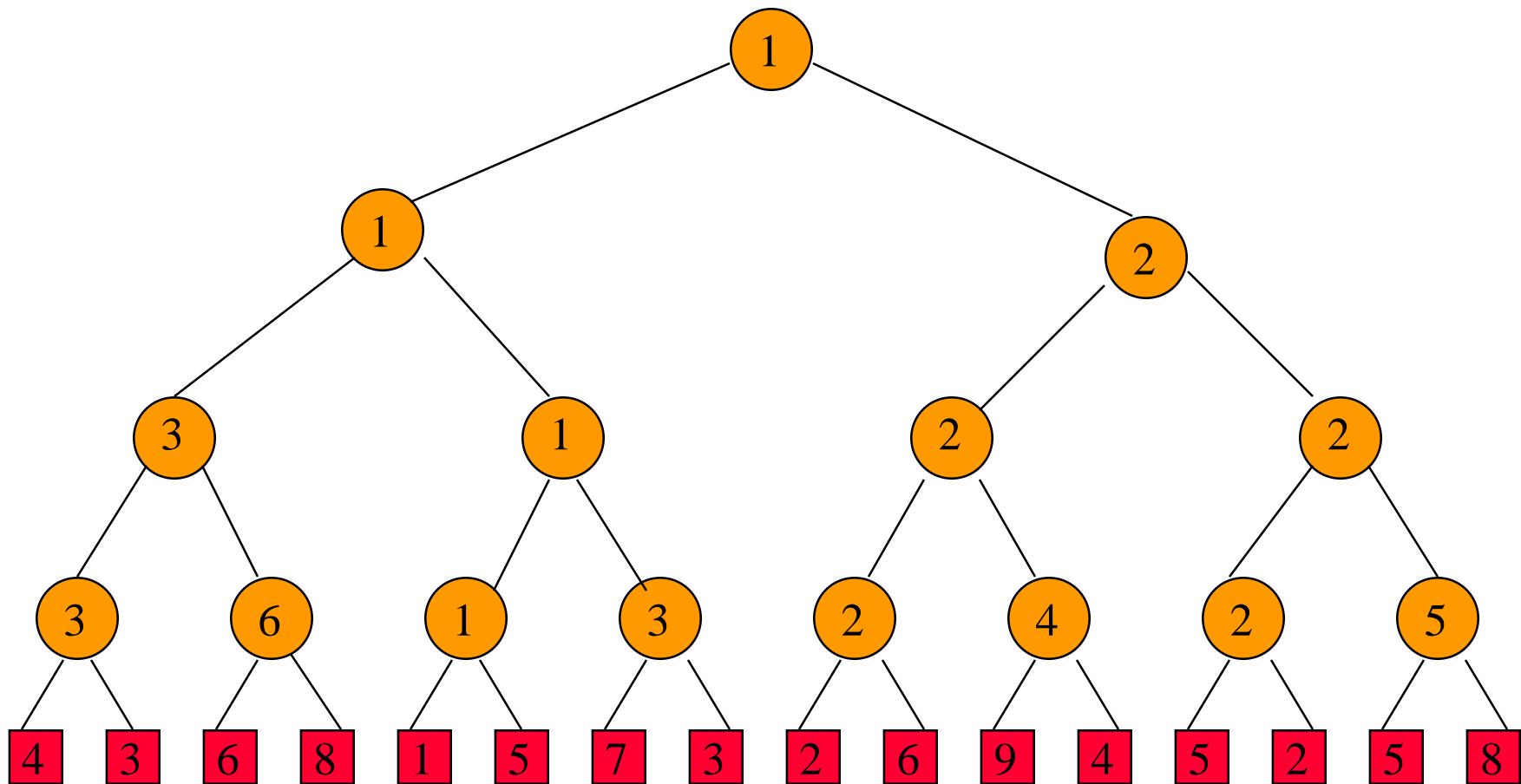
# Хэрэглээ

Эрэмбэлэлт.

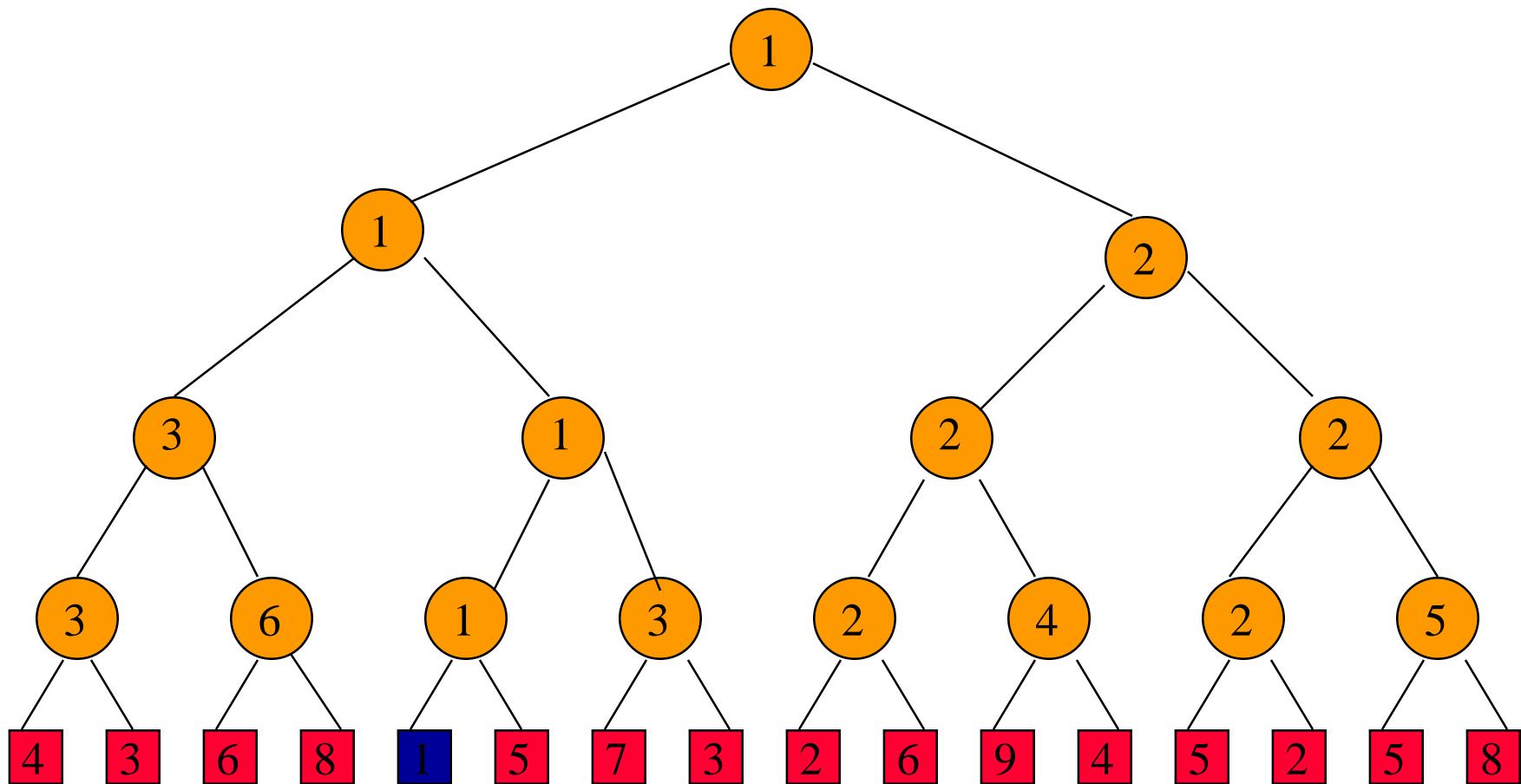
Эрэмбэлэгдэх элементүүдийг  
хожлын модонд оруулах.

Давталтаар ялагчийг тодруулж, том  
утгаар солино.

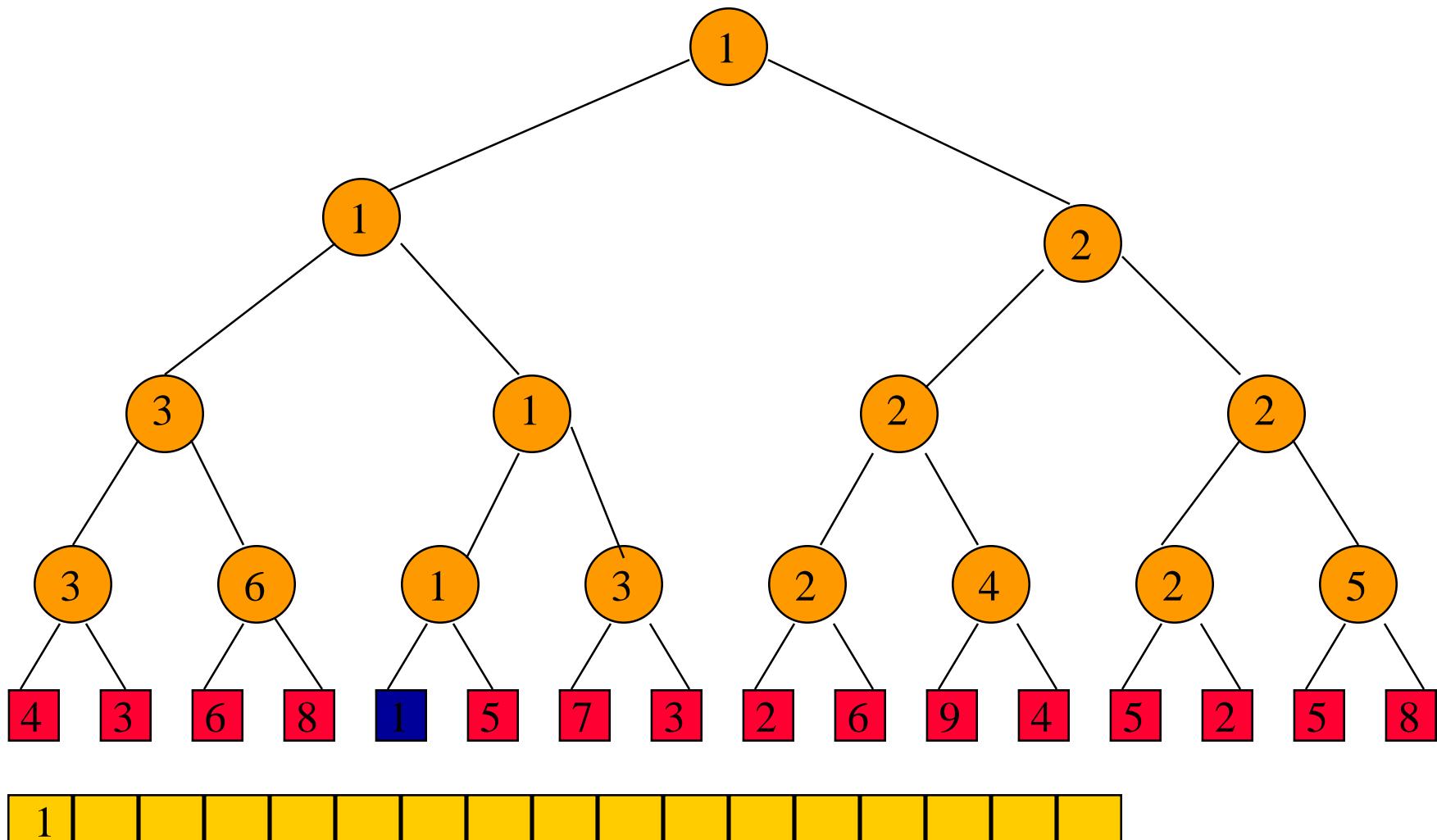
# 16 тоог эрэмбэлэх



# 16 тоог эрэмбэлэх

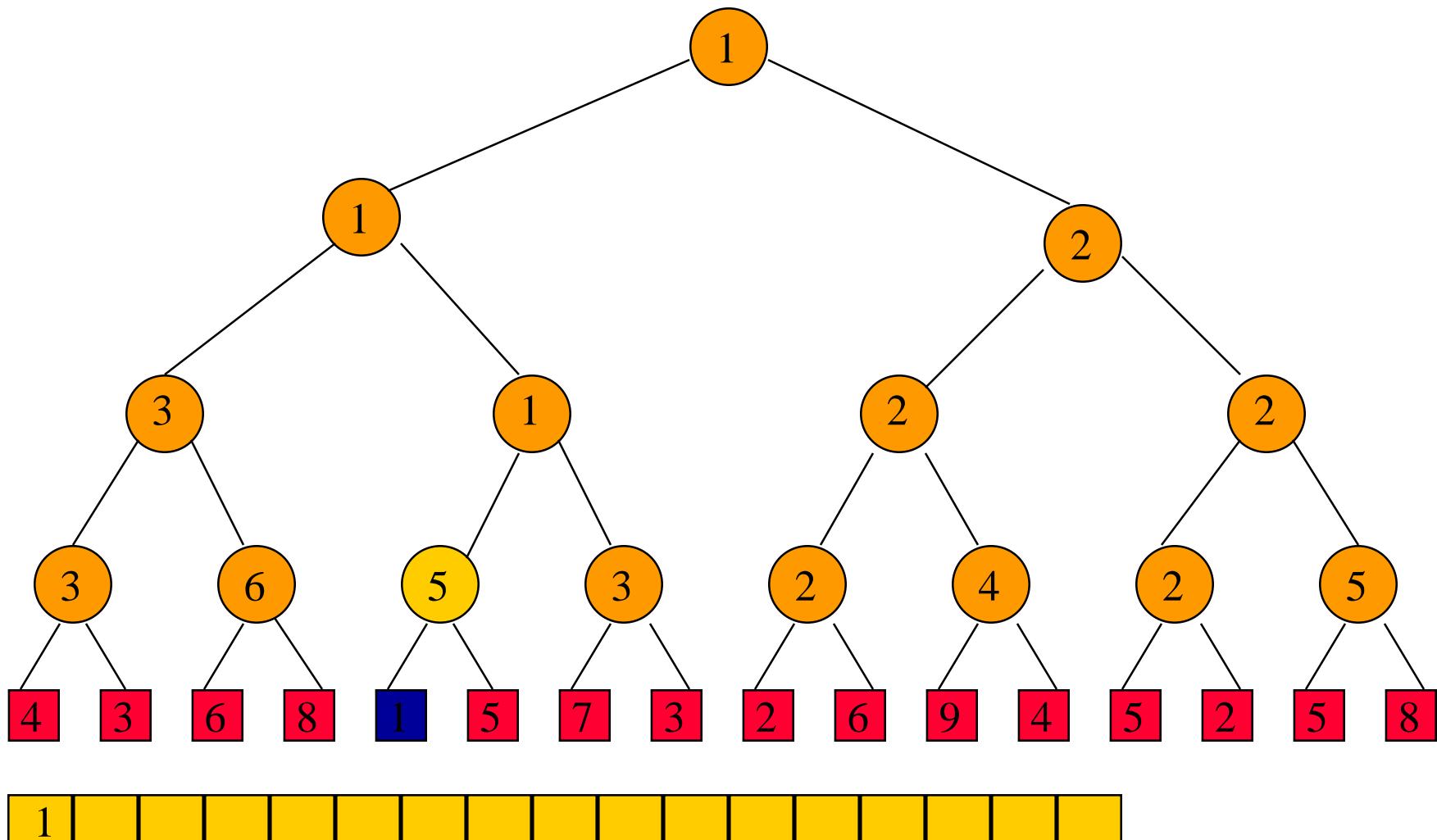


# 16 тоог эрэмбэлэх



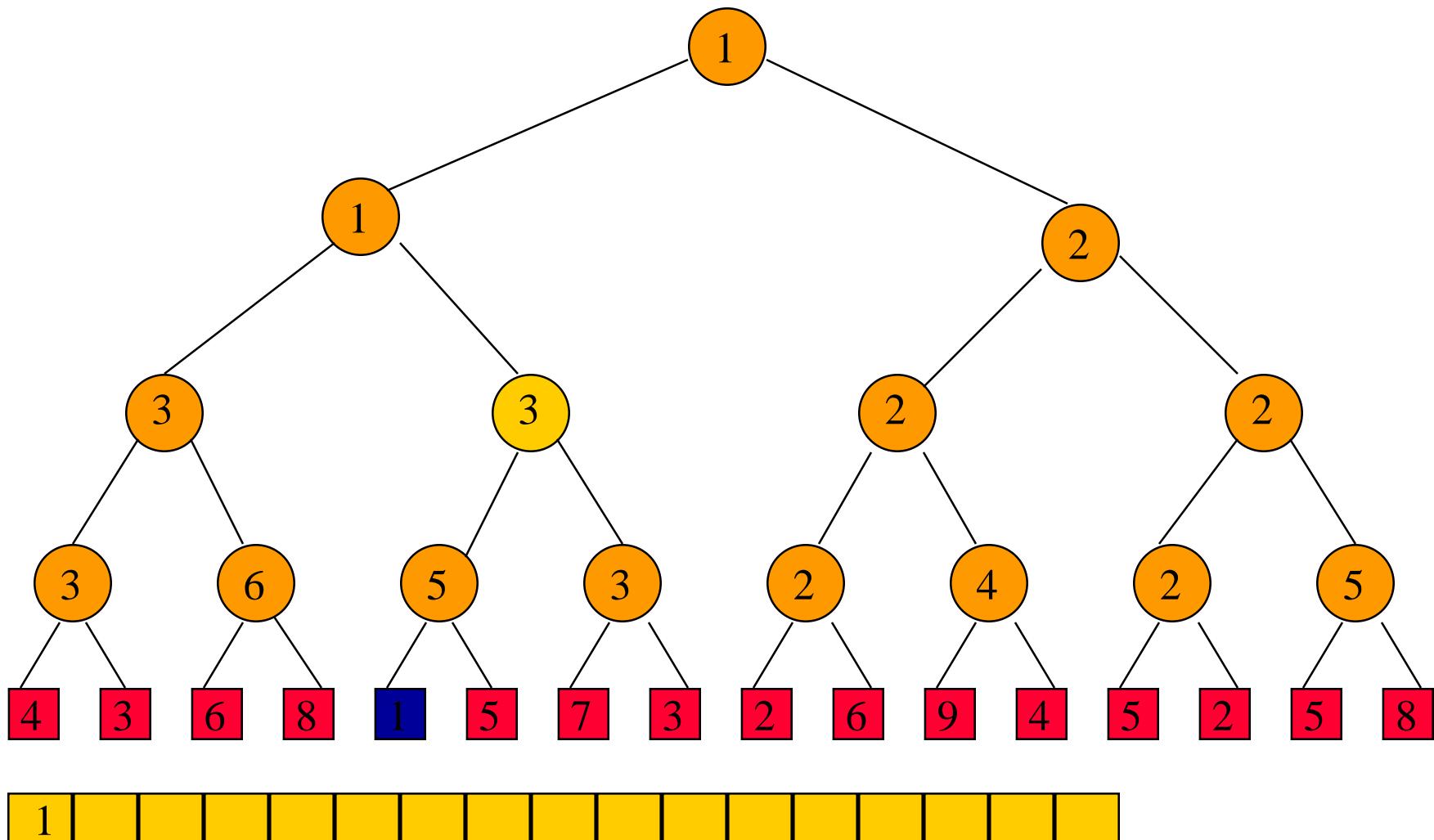
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



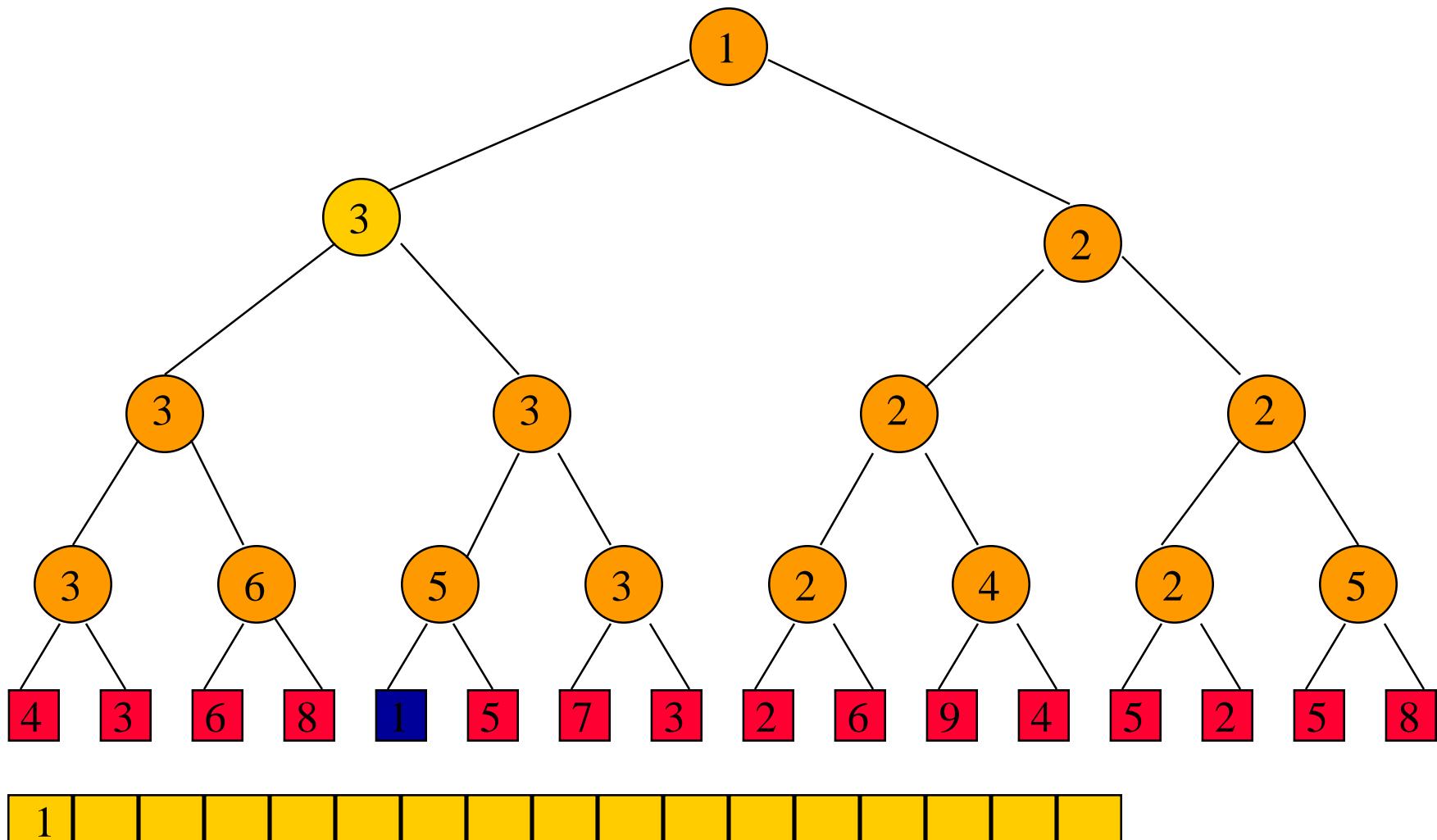
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



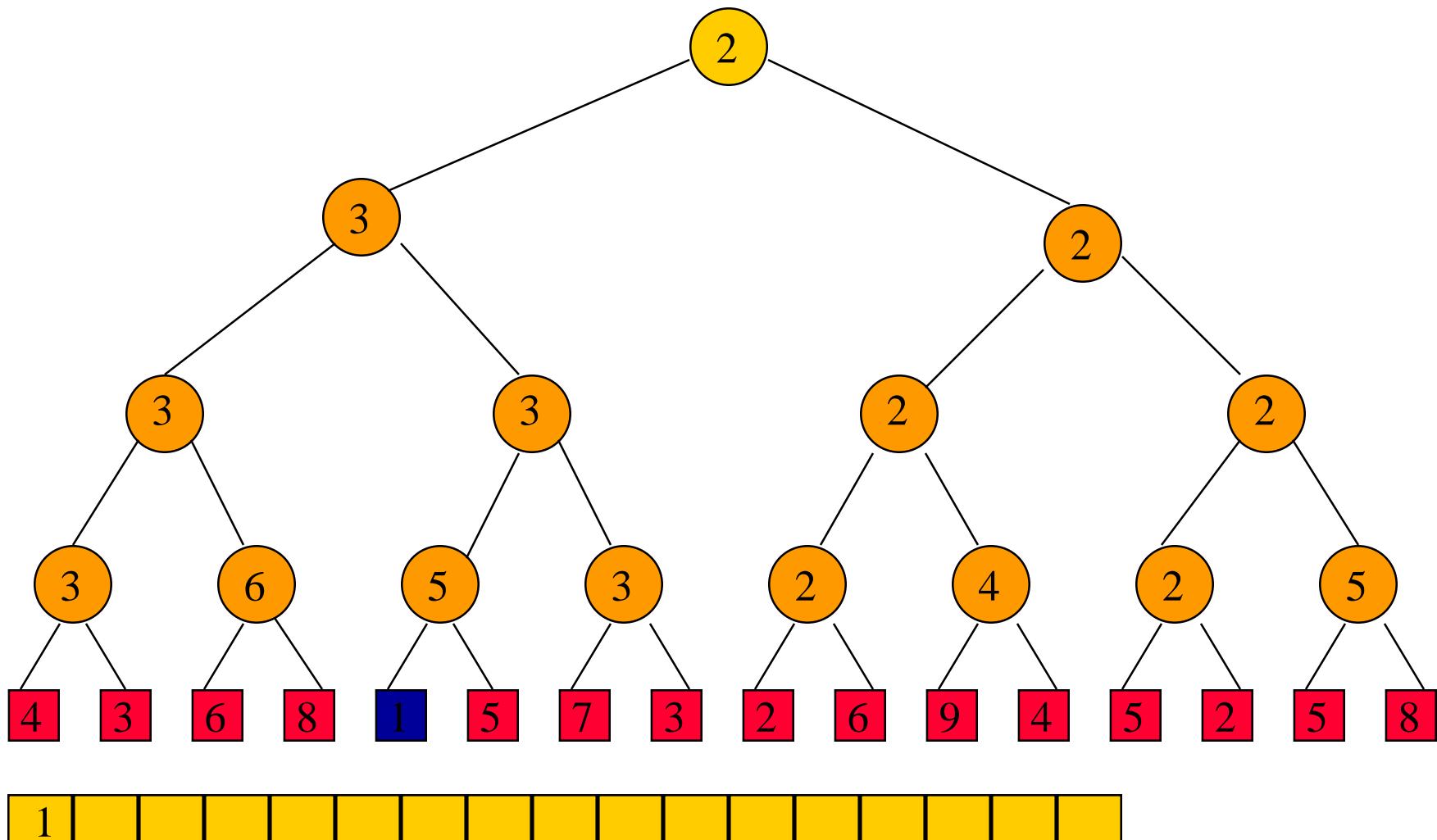
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



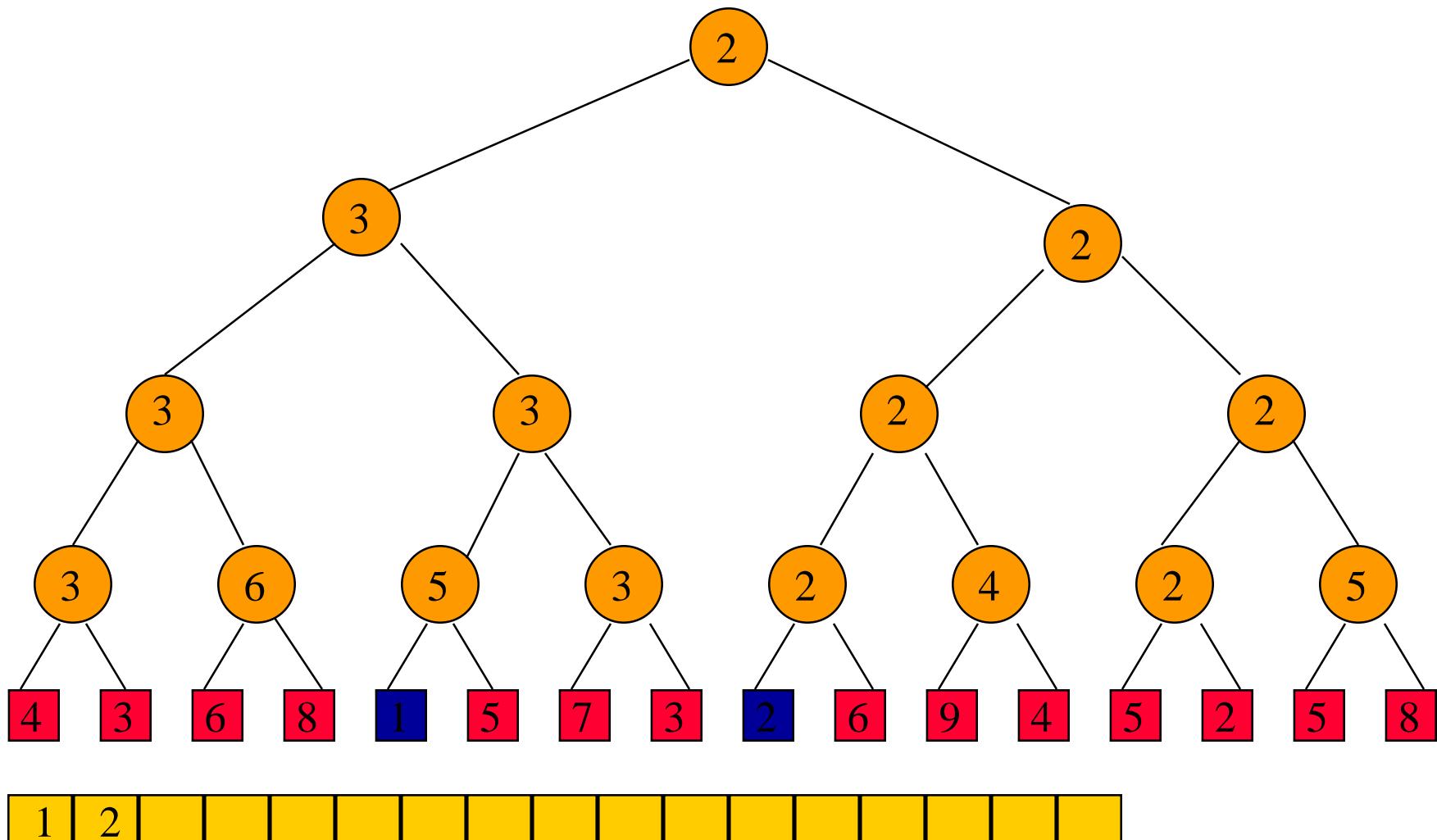
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



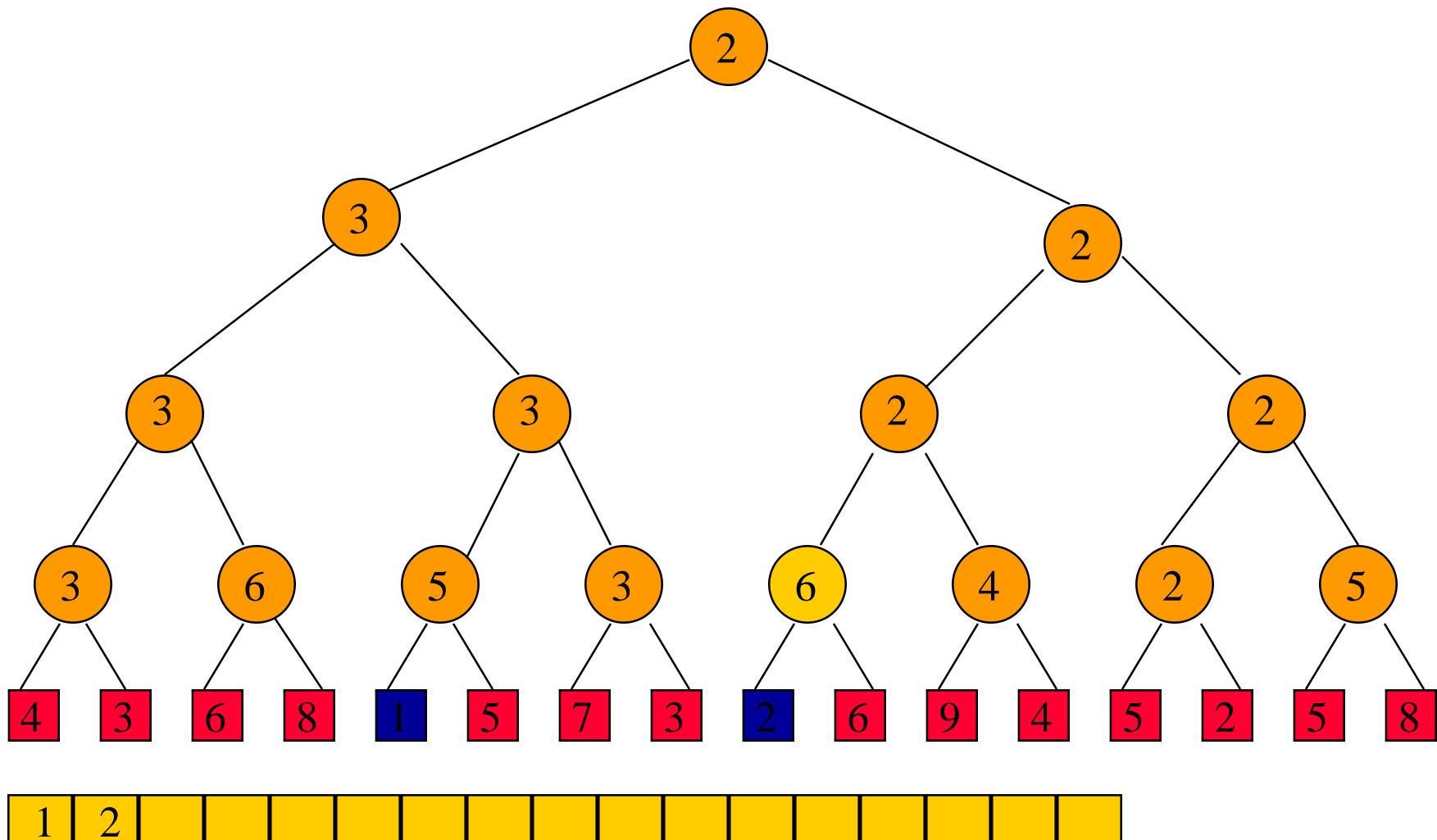
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



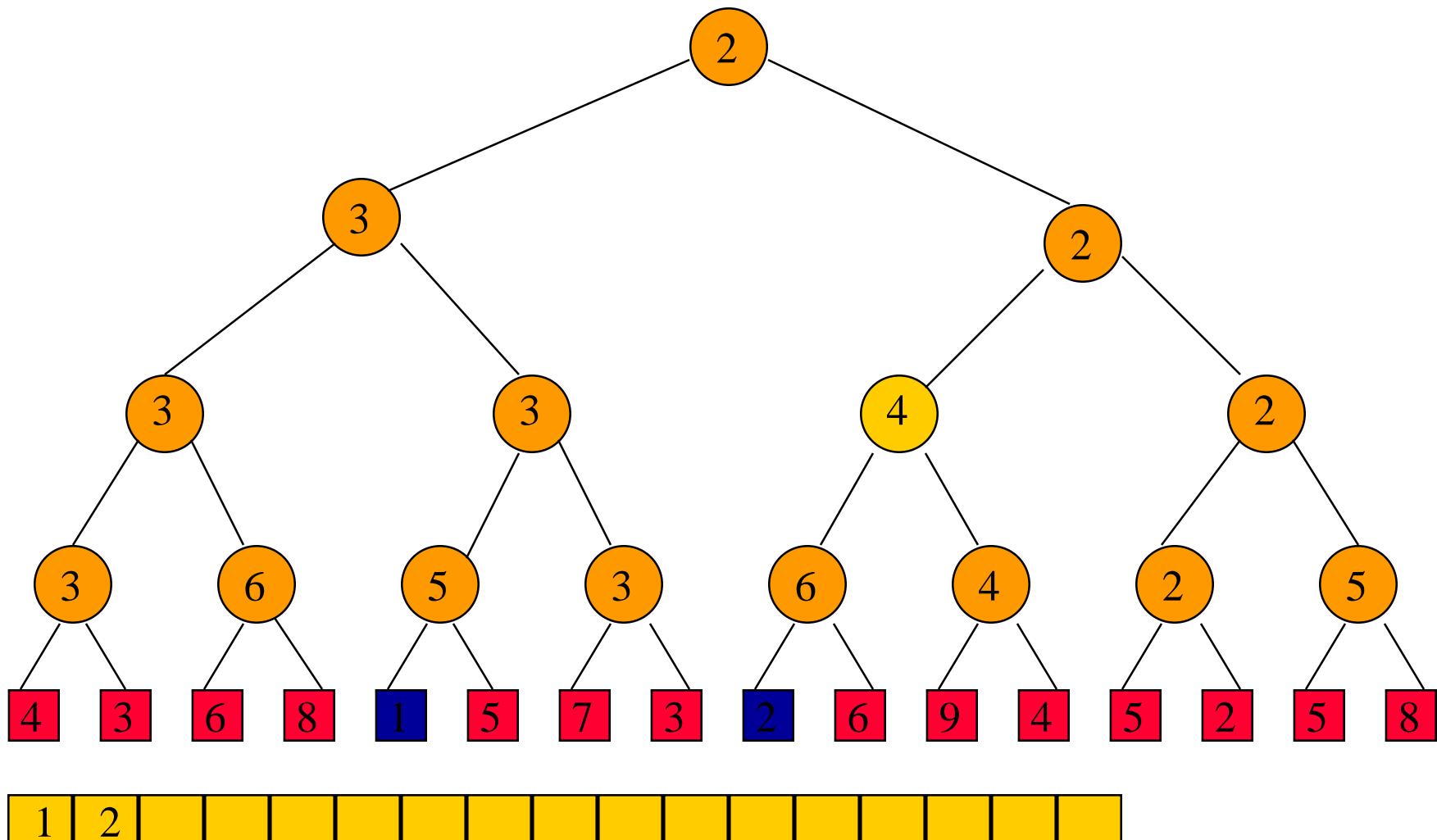
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



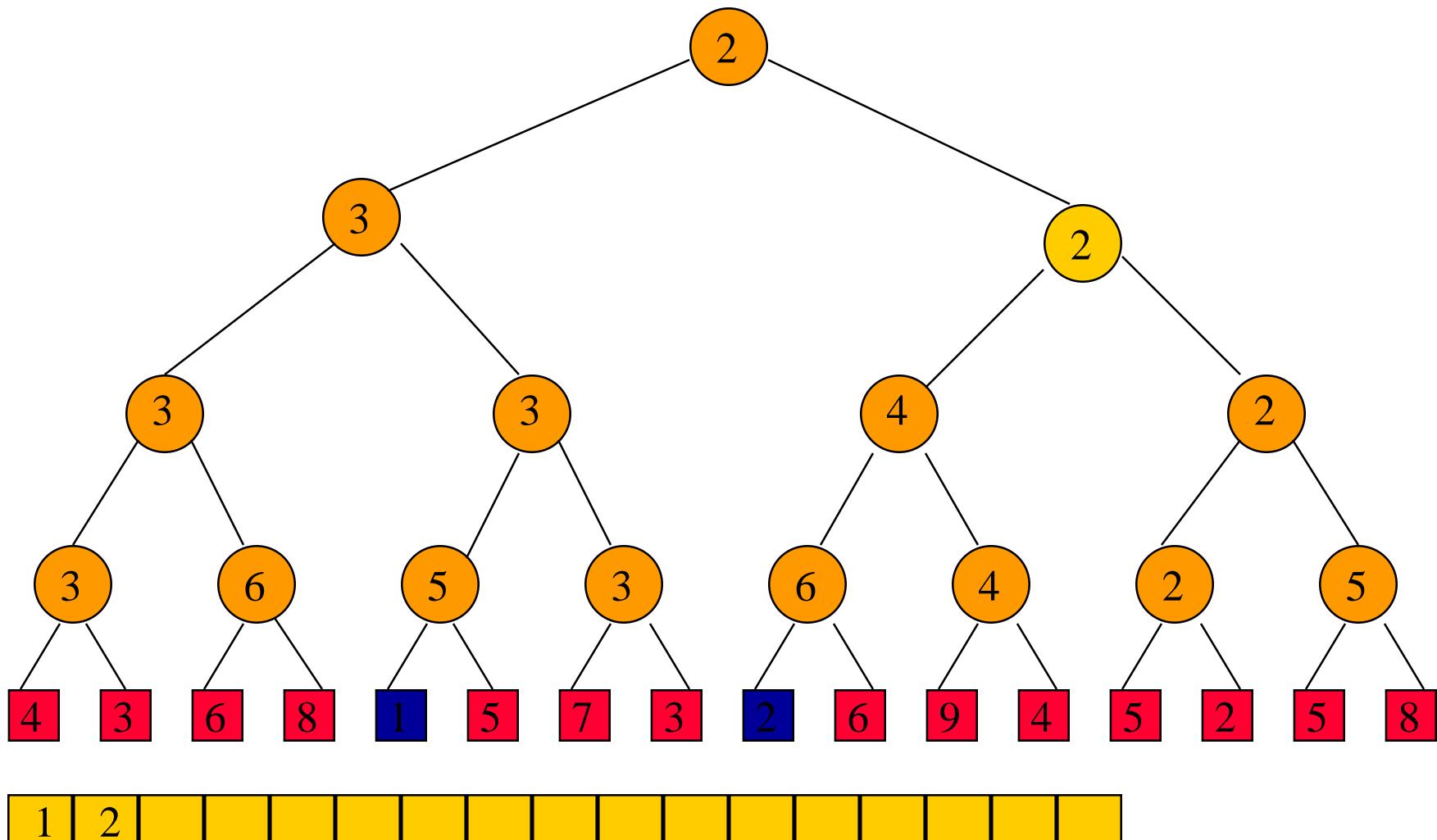
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



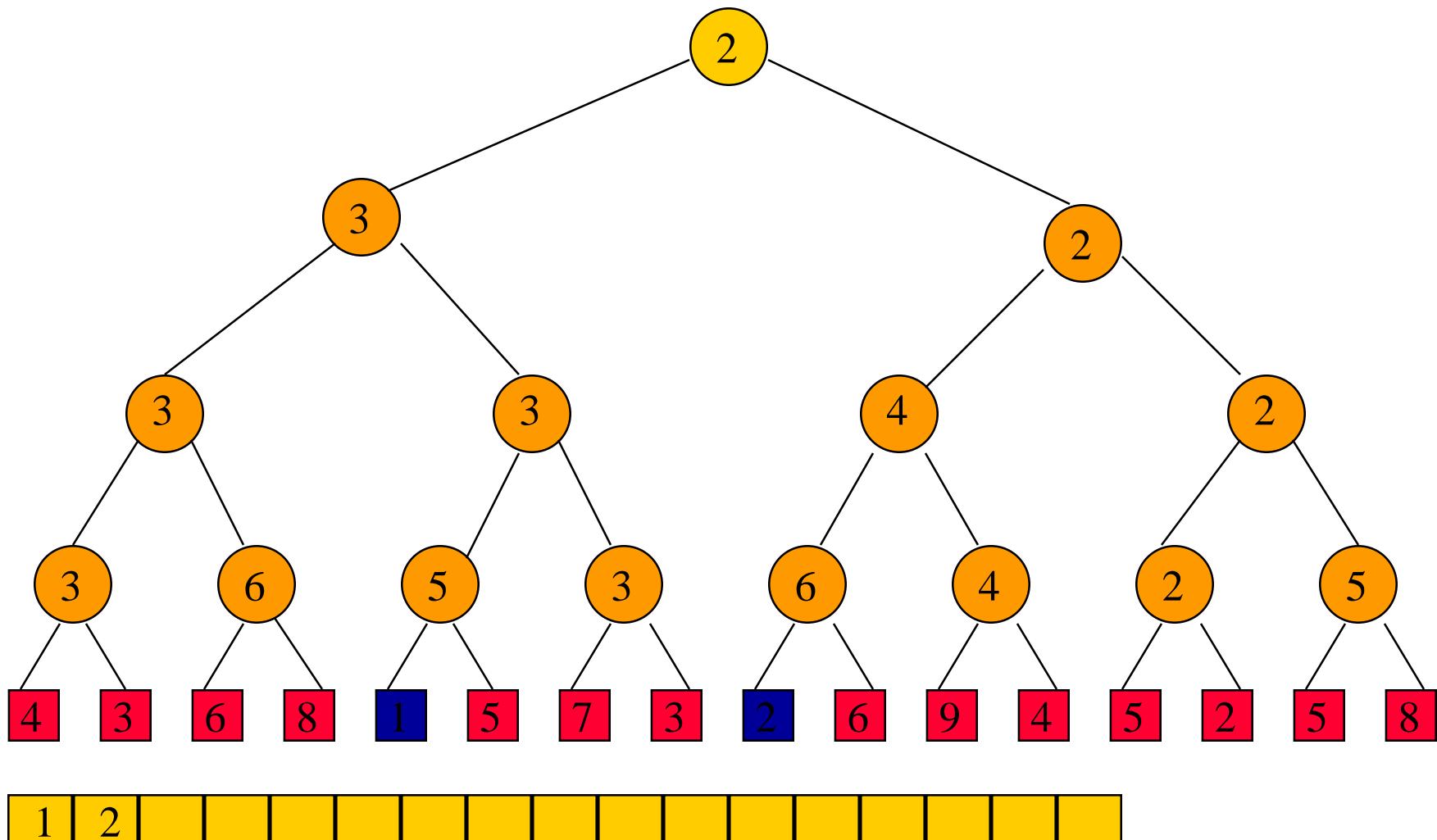
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



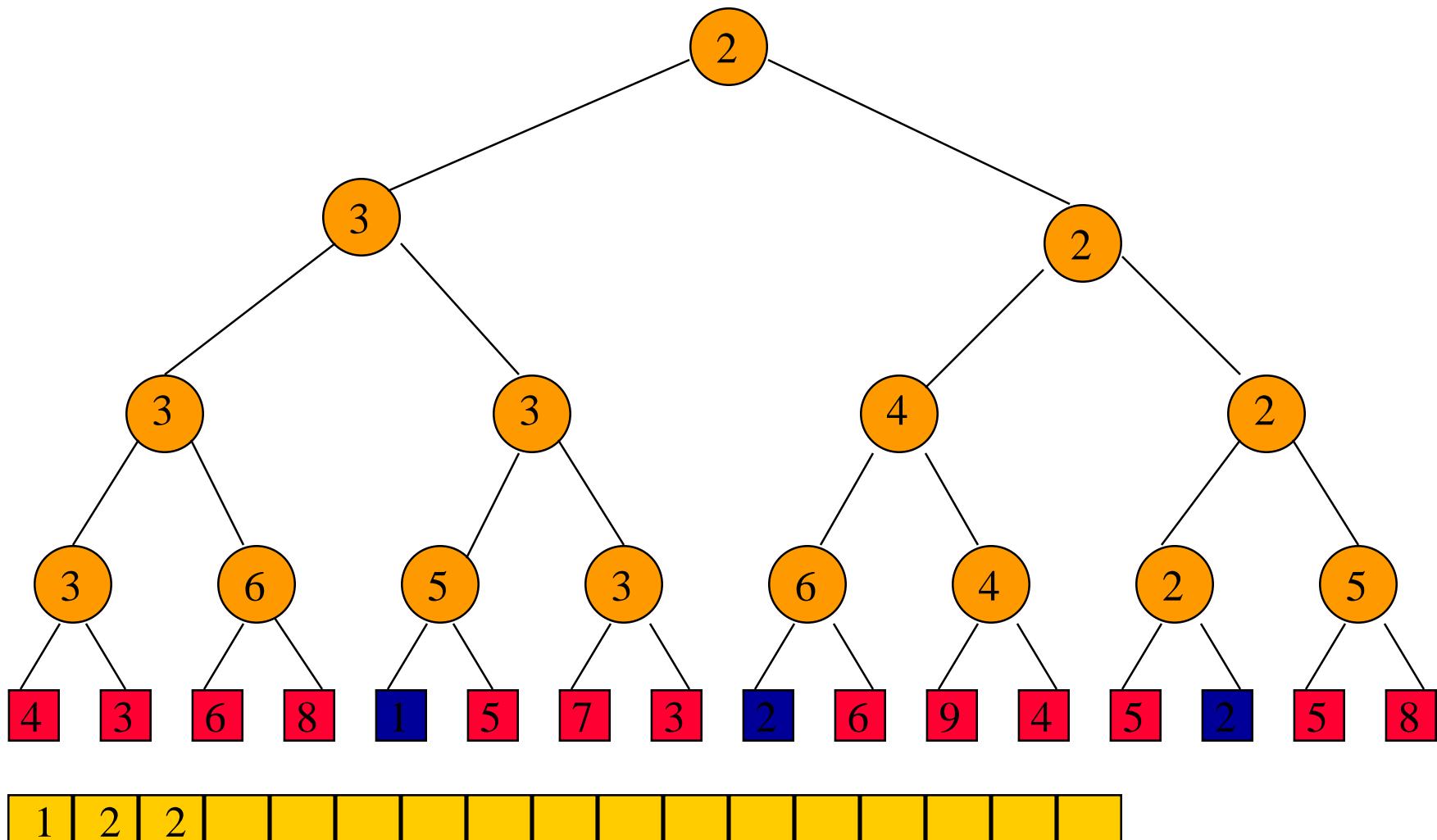
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



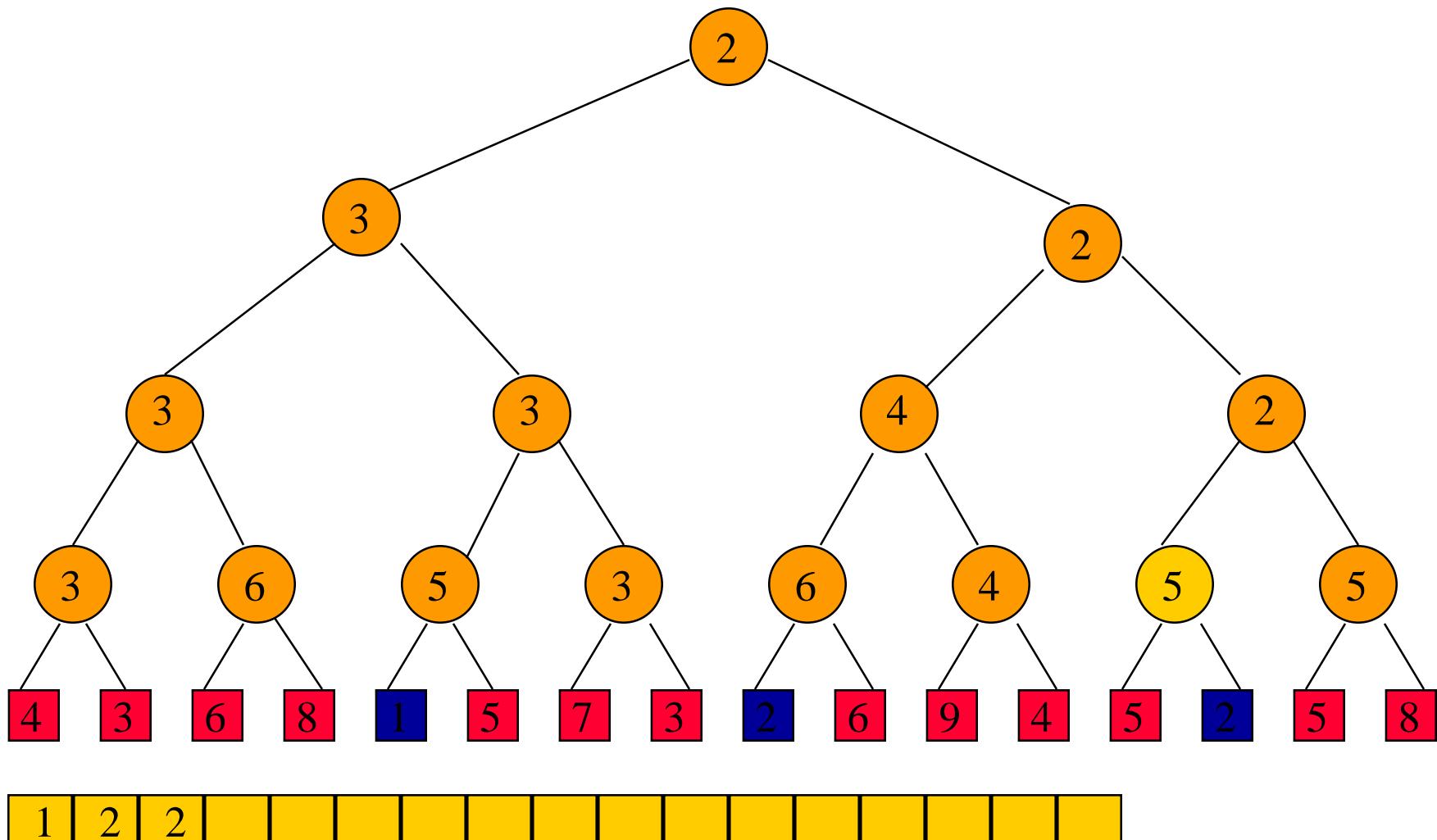
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



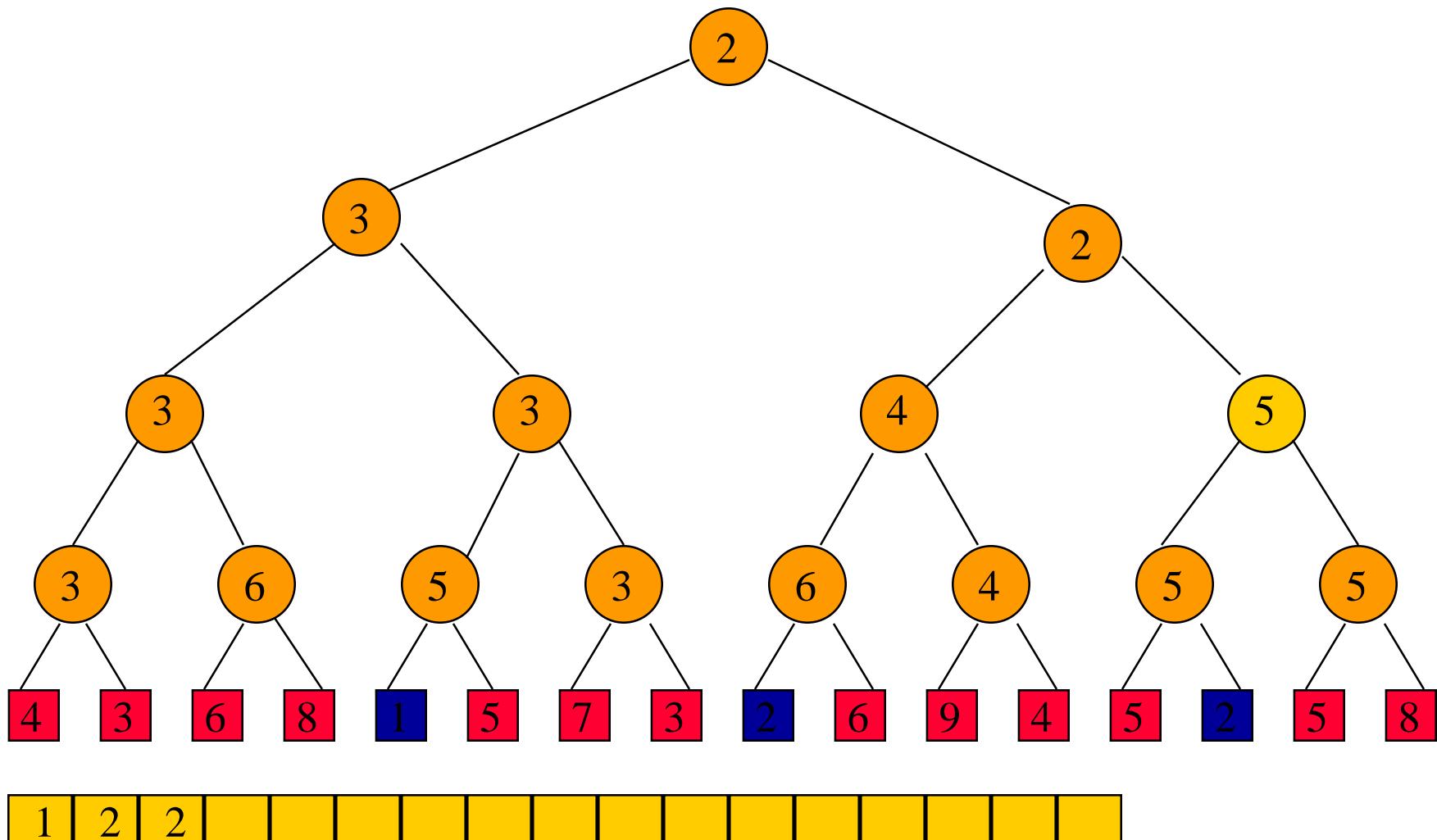
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



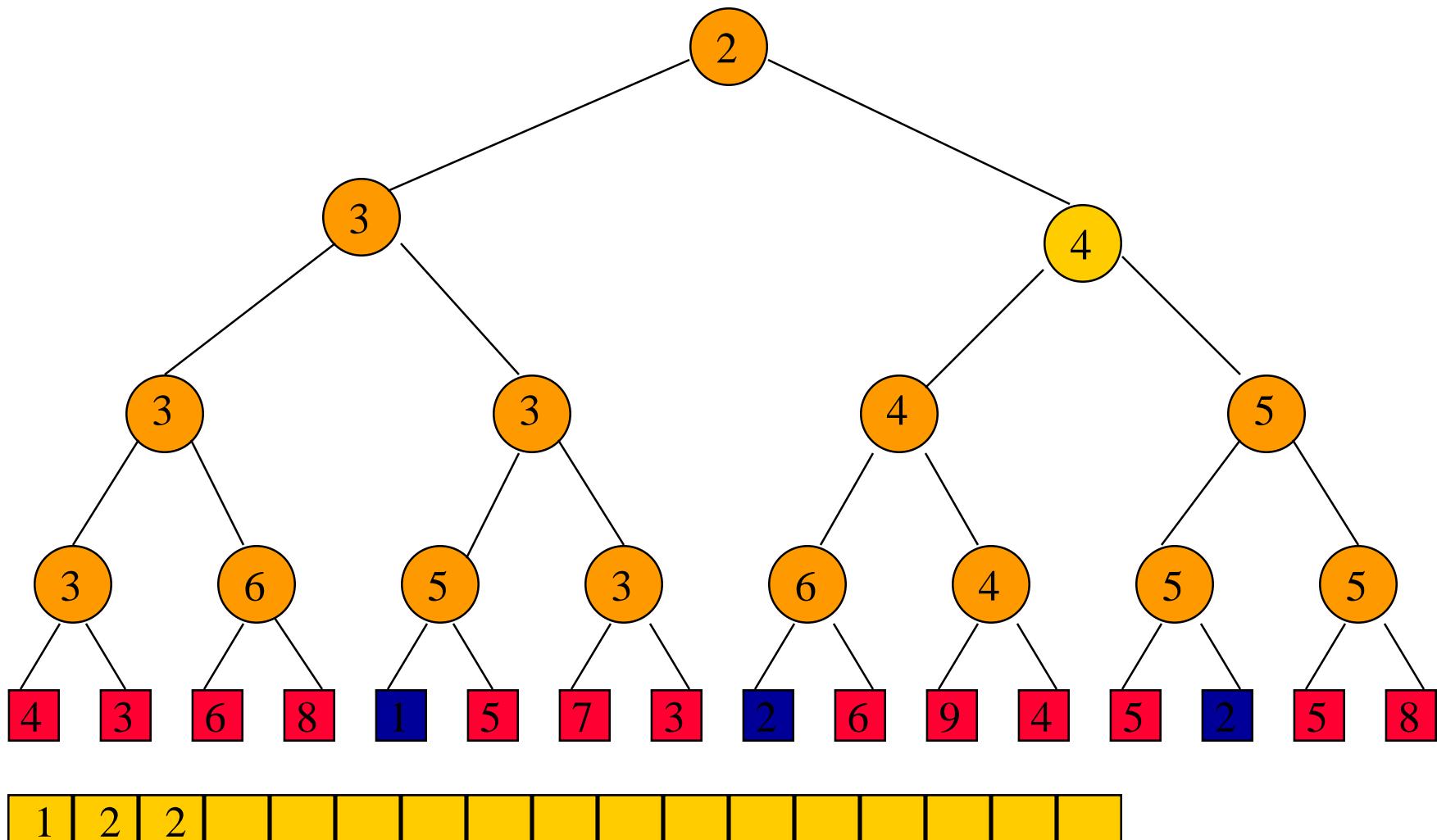
Эррэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



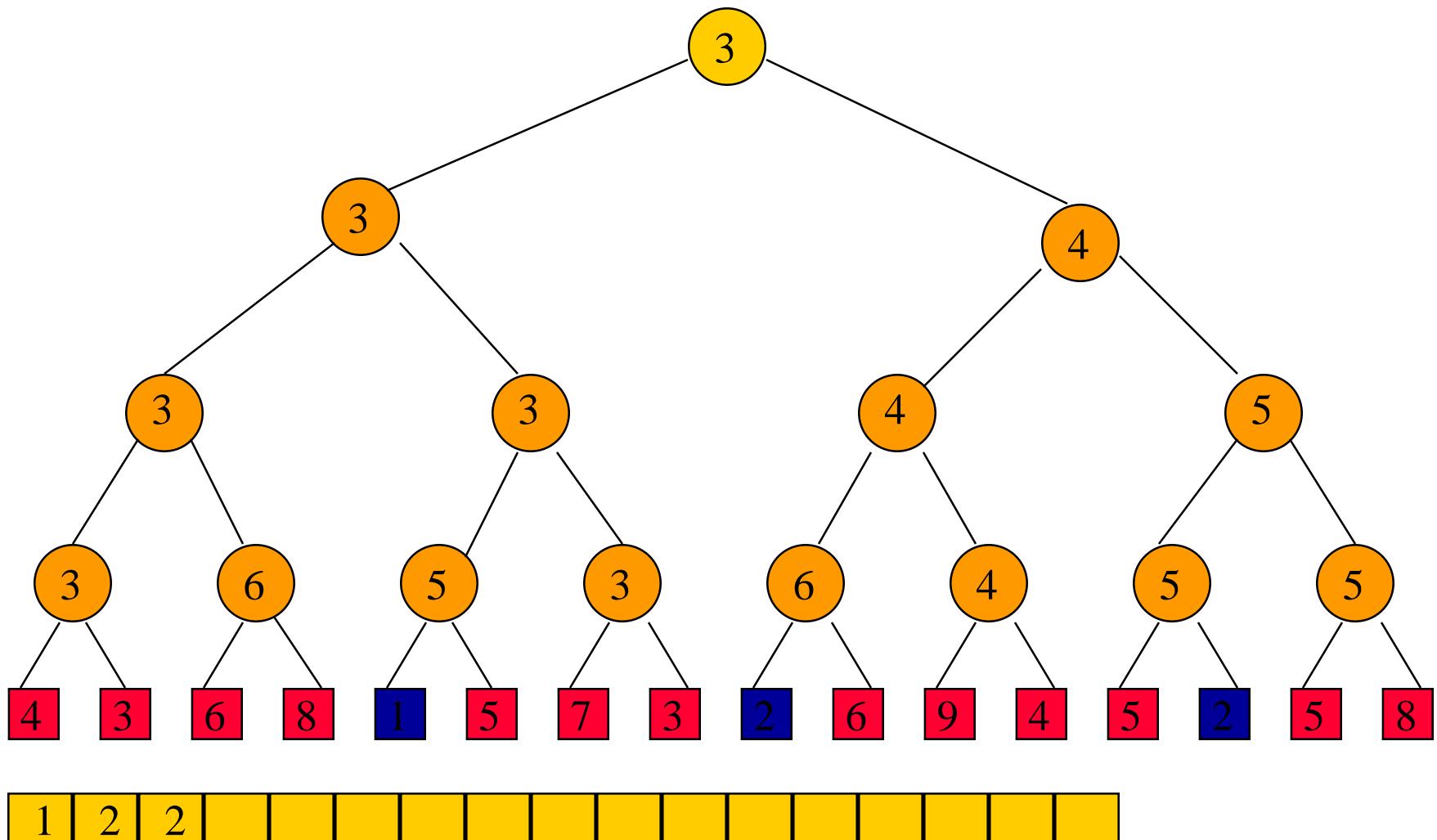
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



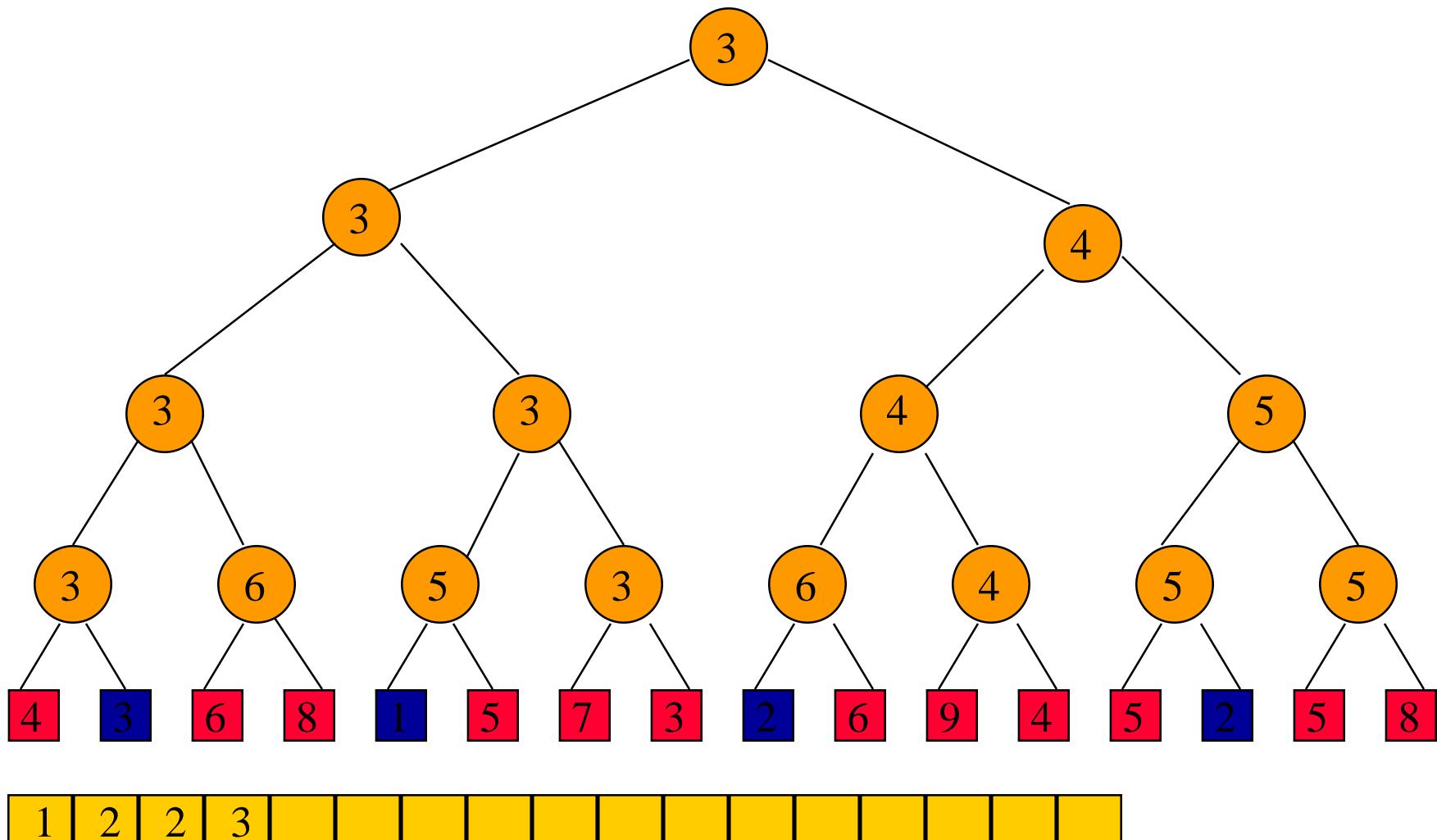
Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



Эрэмбэлэгдсэн массив.

# 16 тоог эрэмбэлэх



Эрэмбэлэгдсэн массив.



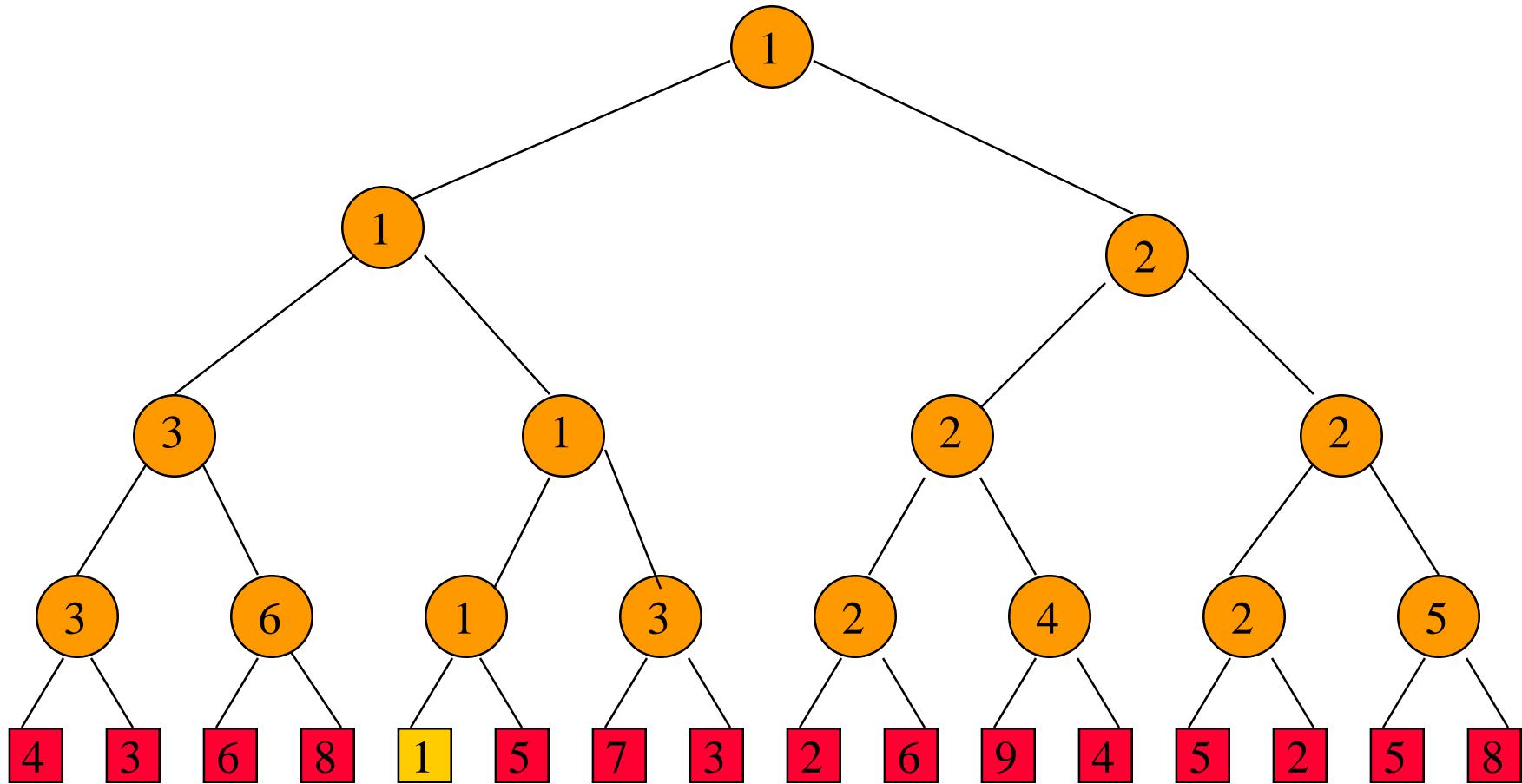
# Эрэмбэлэх хугацаа

- Хожлын модыг идэвхижүүлэх хугацаа.
  - $O(n)$
- Ялагчийг устгаж, дахин тоглох хугацаа.
  - $O(\log n)$
- Ялагчийг устгаж,  $n$  дахин тоглох.
  - $O(n \log n)$
- Эрэмбэлэх хугацаа  $O(n \log n)$ .
- Жинхэнэдээ  $\Theta(n \log n)$ .

# Хожлын модны үйлдлүүд

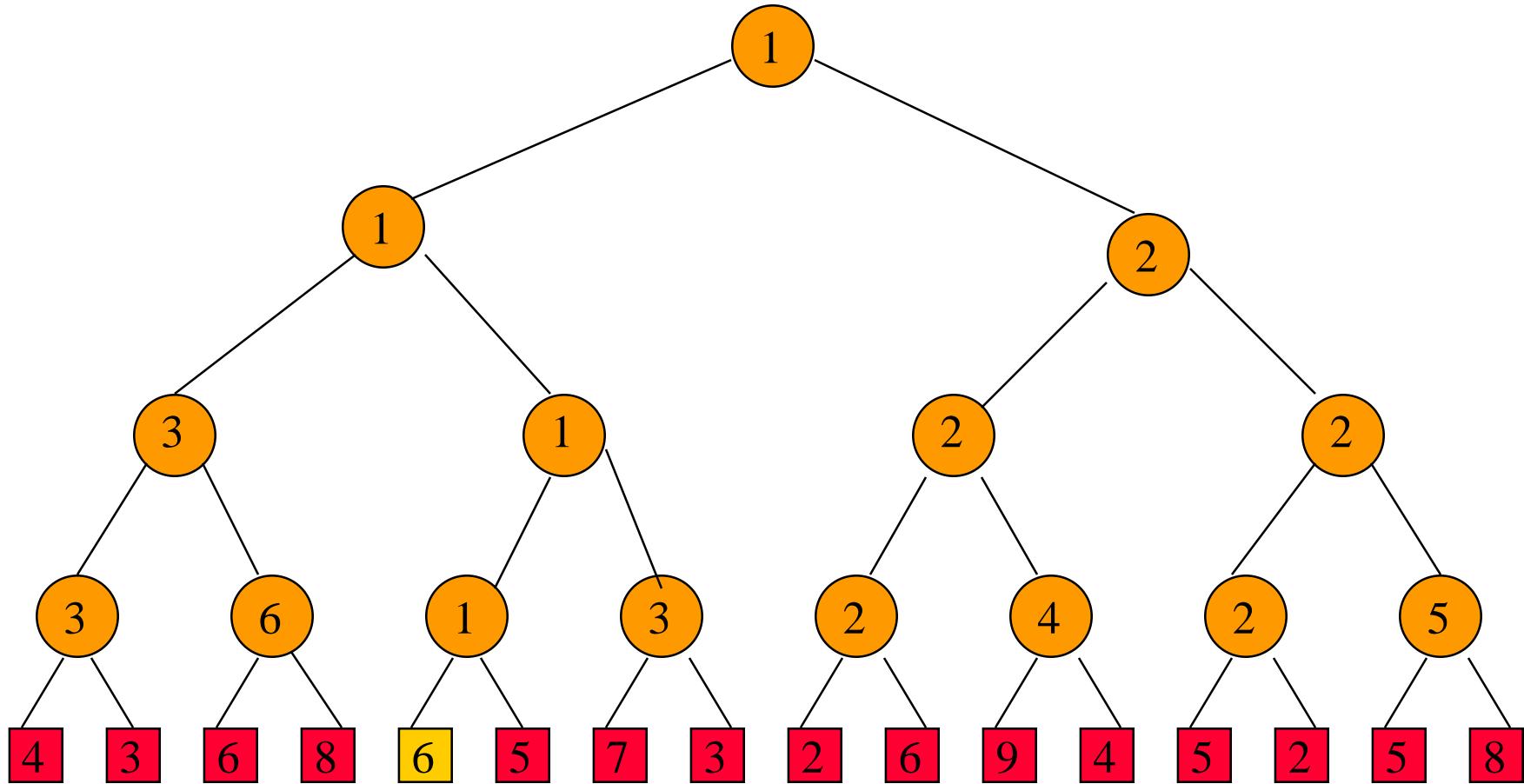
- Идэвхижүүлэх
  - $O(n)$
- Ялагчийг гаргах
  - $O(1)$
- Ялагчийг устгах/солих, дахин тоглох
  - $O(\log n)$
  - яг нарийндаа  $\Theta(\log n)$

# Ялагчийг солиод дахин тоглох



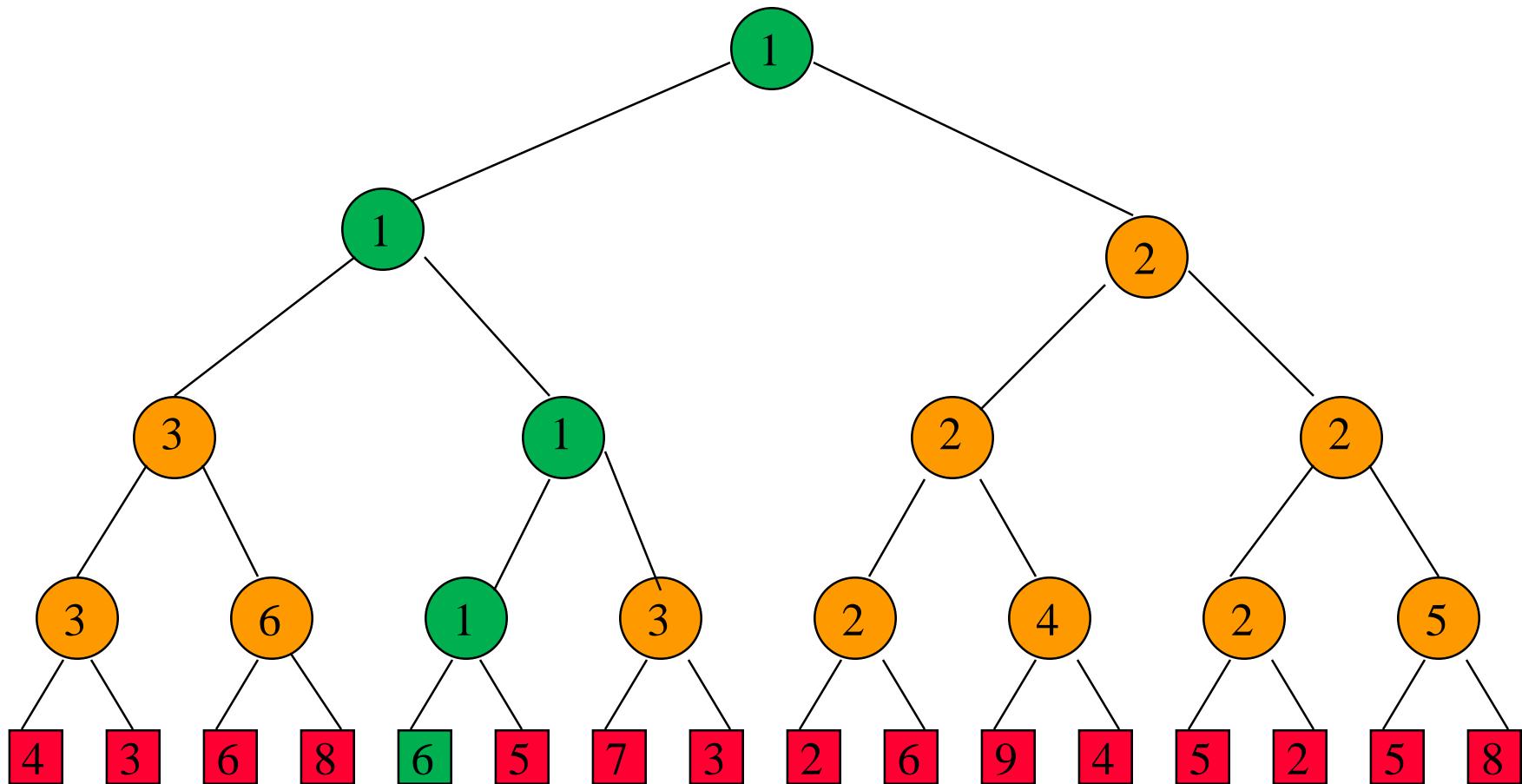
Ялагчийг **6** –аар солих.

# Ялагчийг солиод дахин тоглох



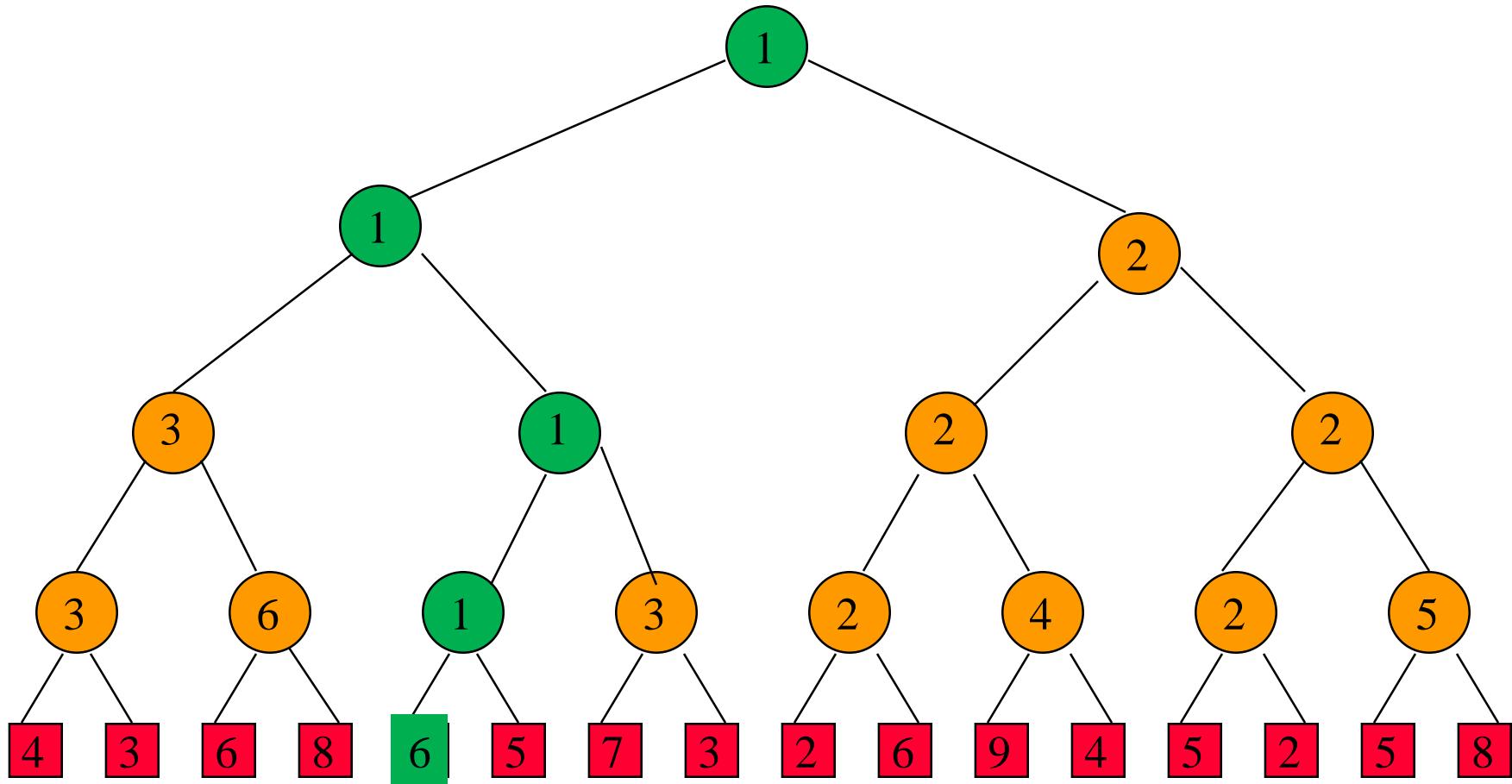
Үндэс хүртэлх замын тоглолтуудыг дахин хийх.

# Ялагчийг солиод дахин тоглох



Үндэс хүртэлх замын тоглолтуудыг дахин хийх.

# Ялагчийг солиод дахин тоглох

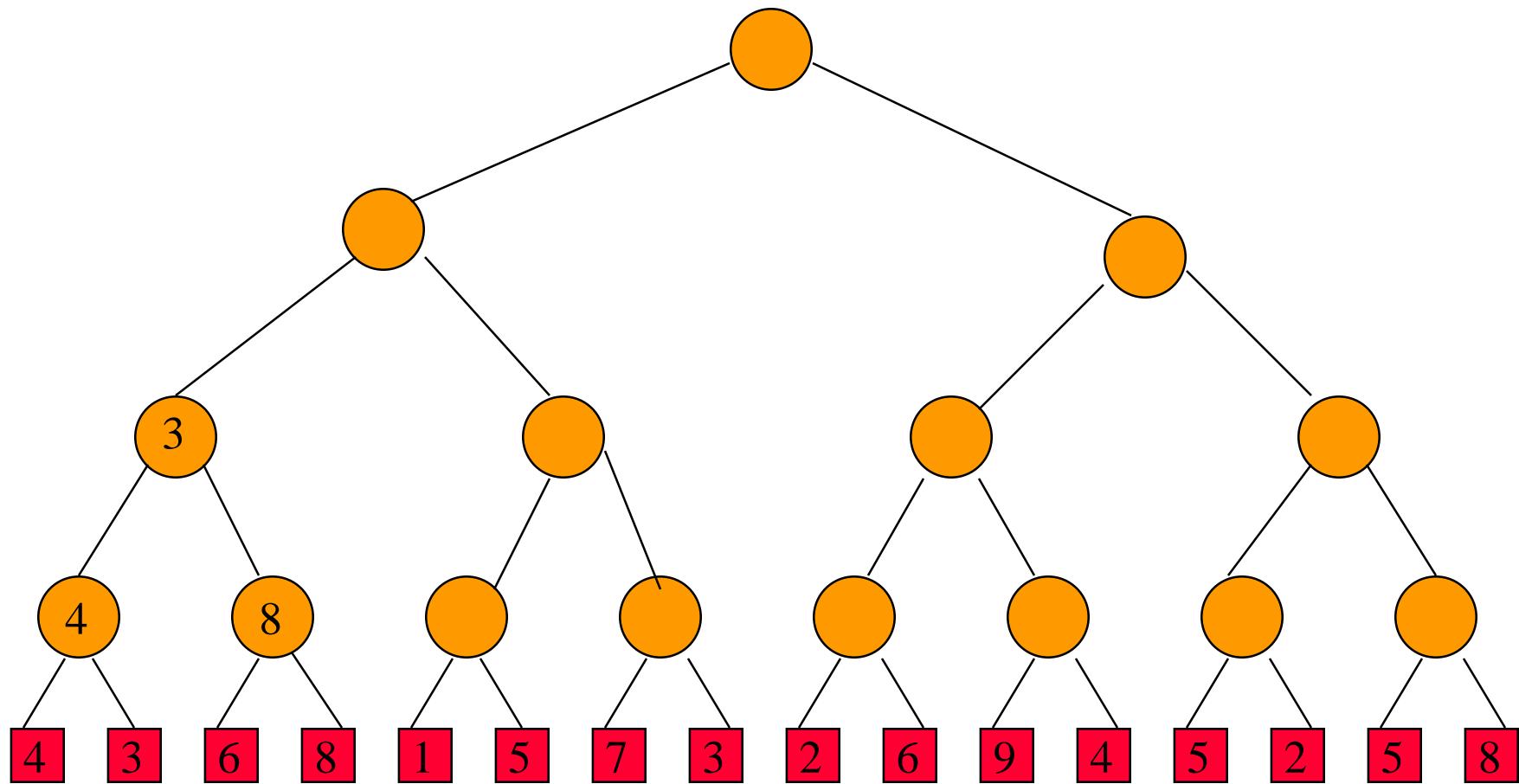


Өрсөлдөгч нь энэ зангилааны сүүлийн тоглолтонд ялагдсан  
тоглогч байна.

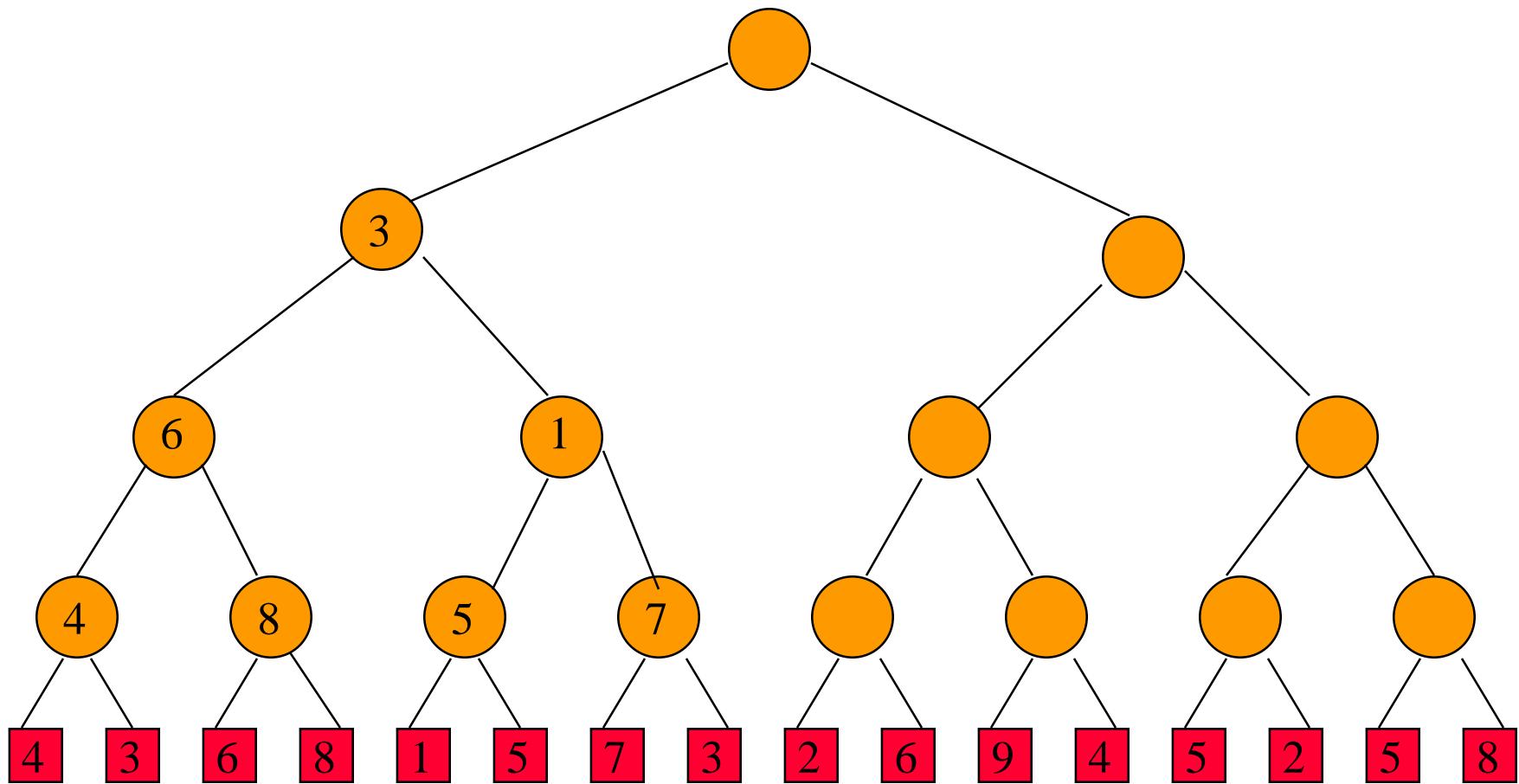
# Хожигдлын мод

Тоглолтын зангилаа бүрт ялсан  
биш, ялагдсан тоглогчийг  
хадгална.

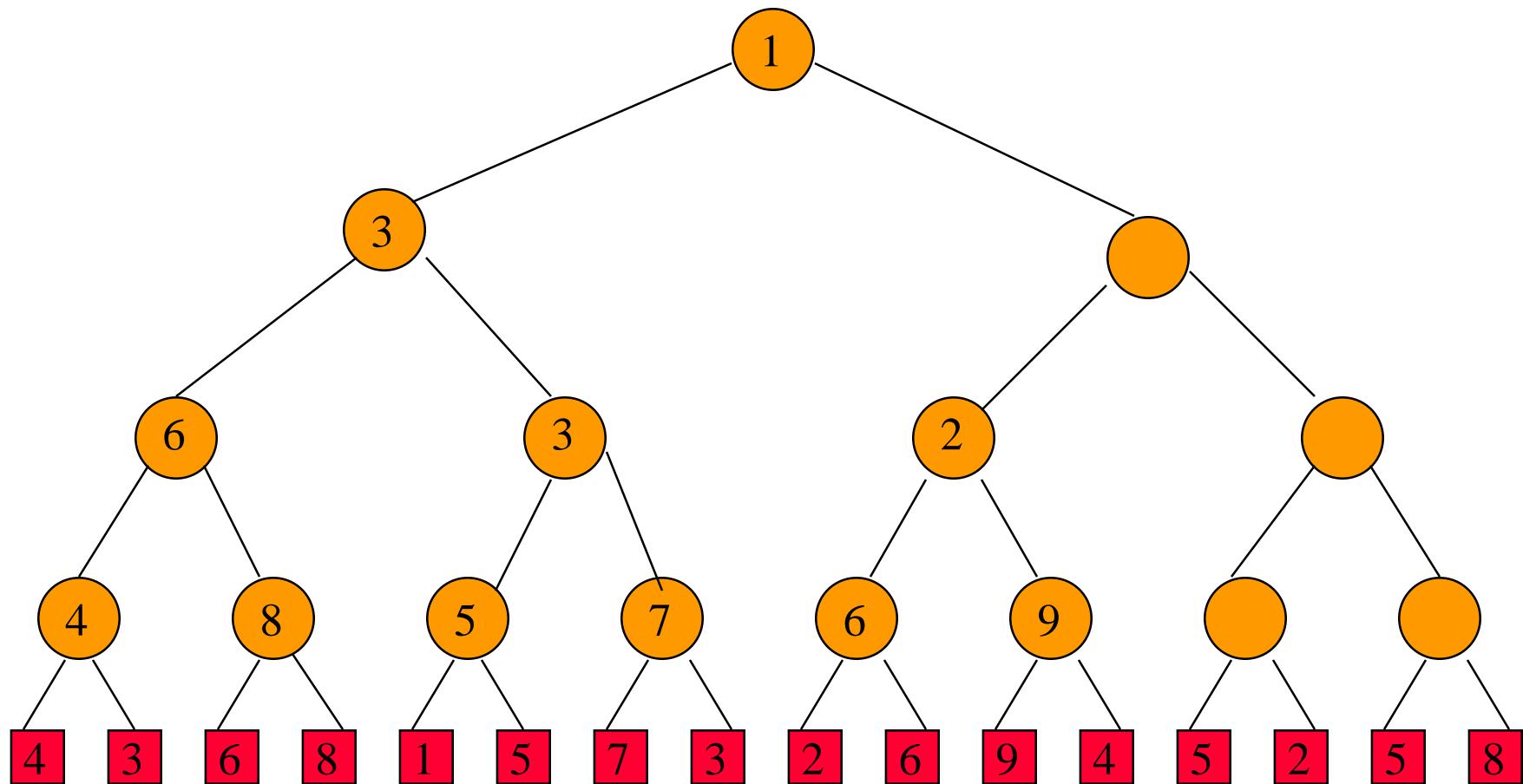
# 16 тоглогчтой min хожигдлын мод



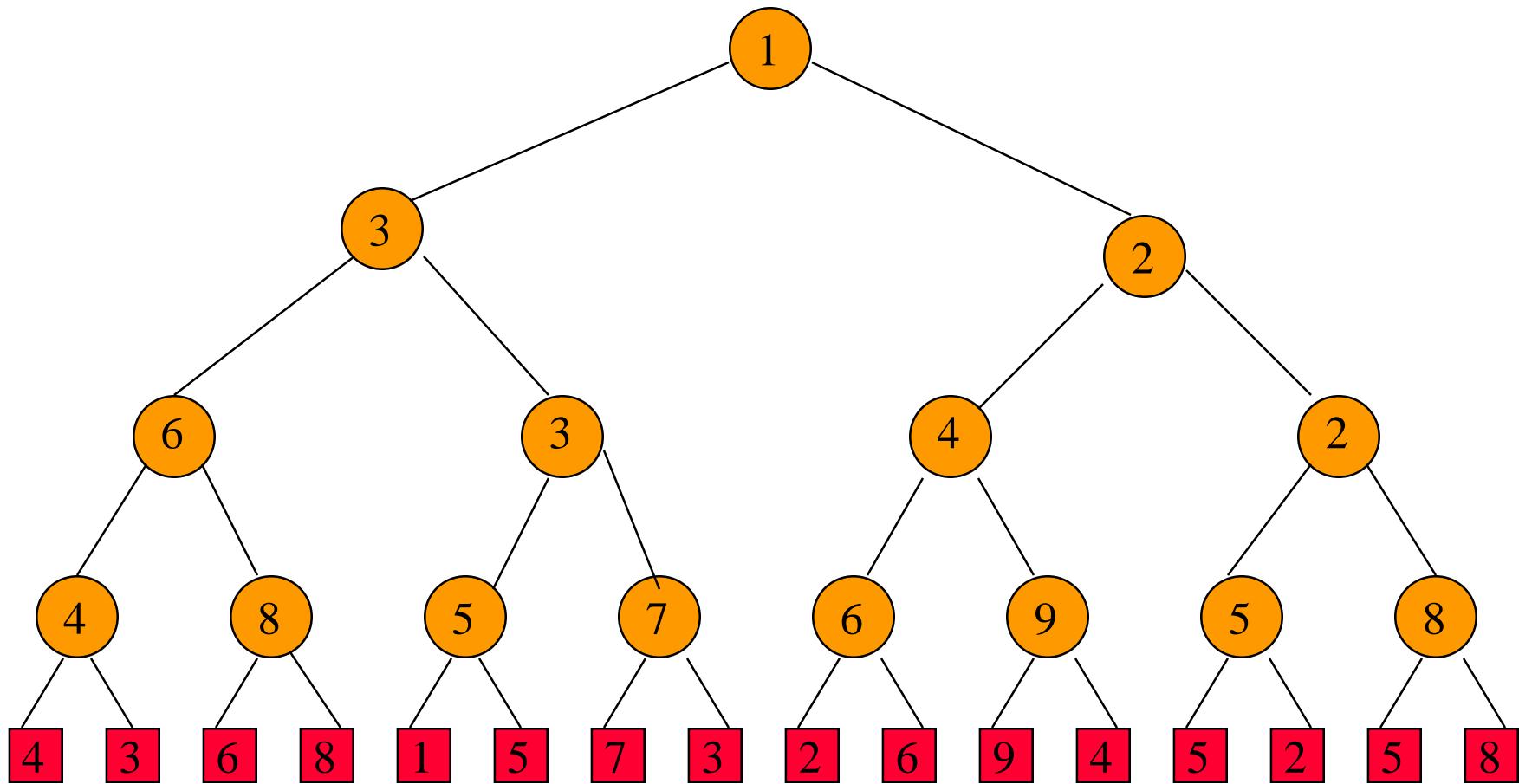
# 16 тоглогчтой min хожигдлын мод



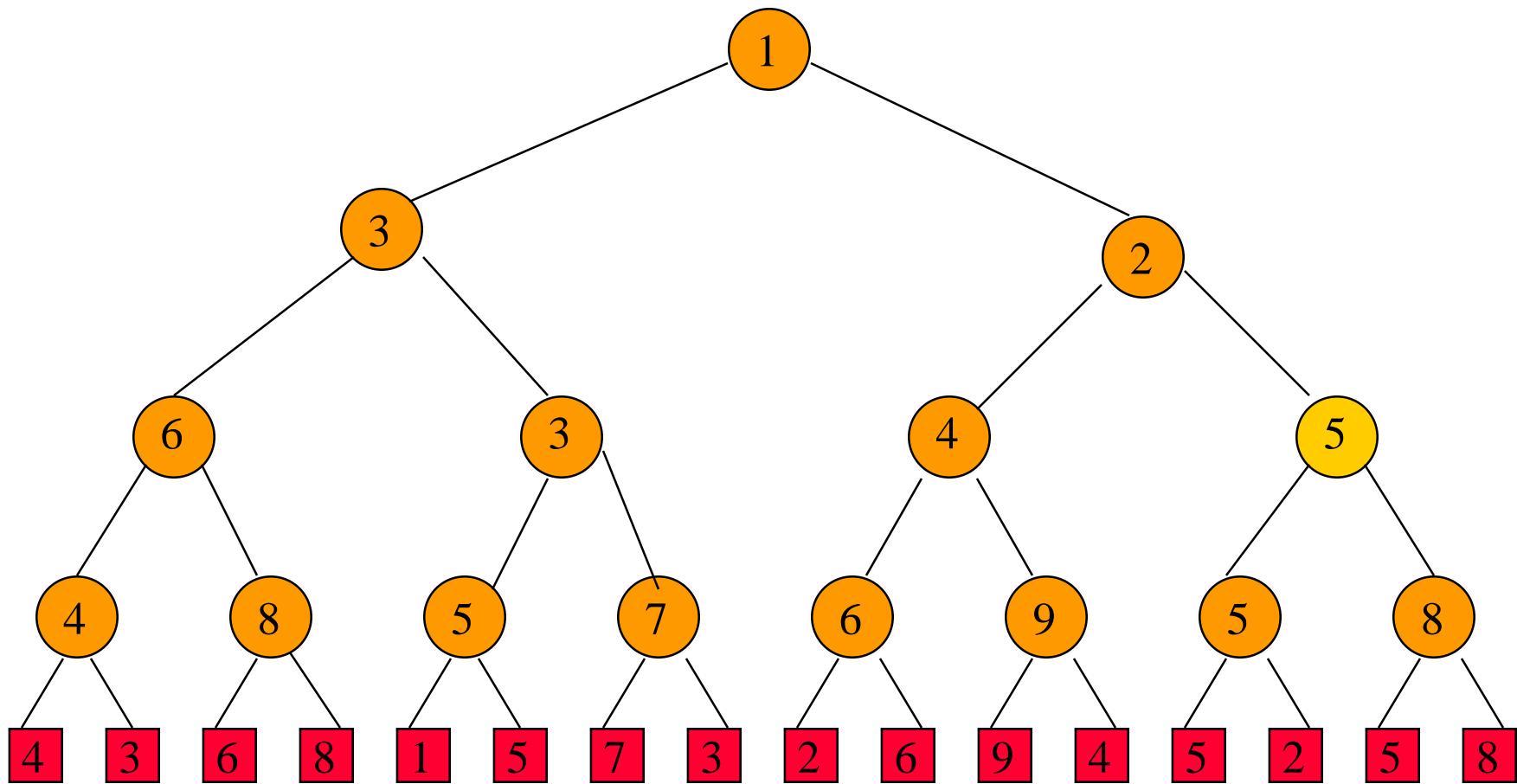
# 16 тоглогчтой min хожигдлын мод



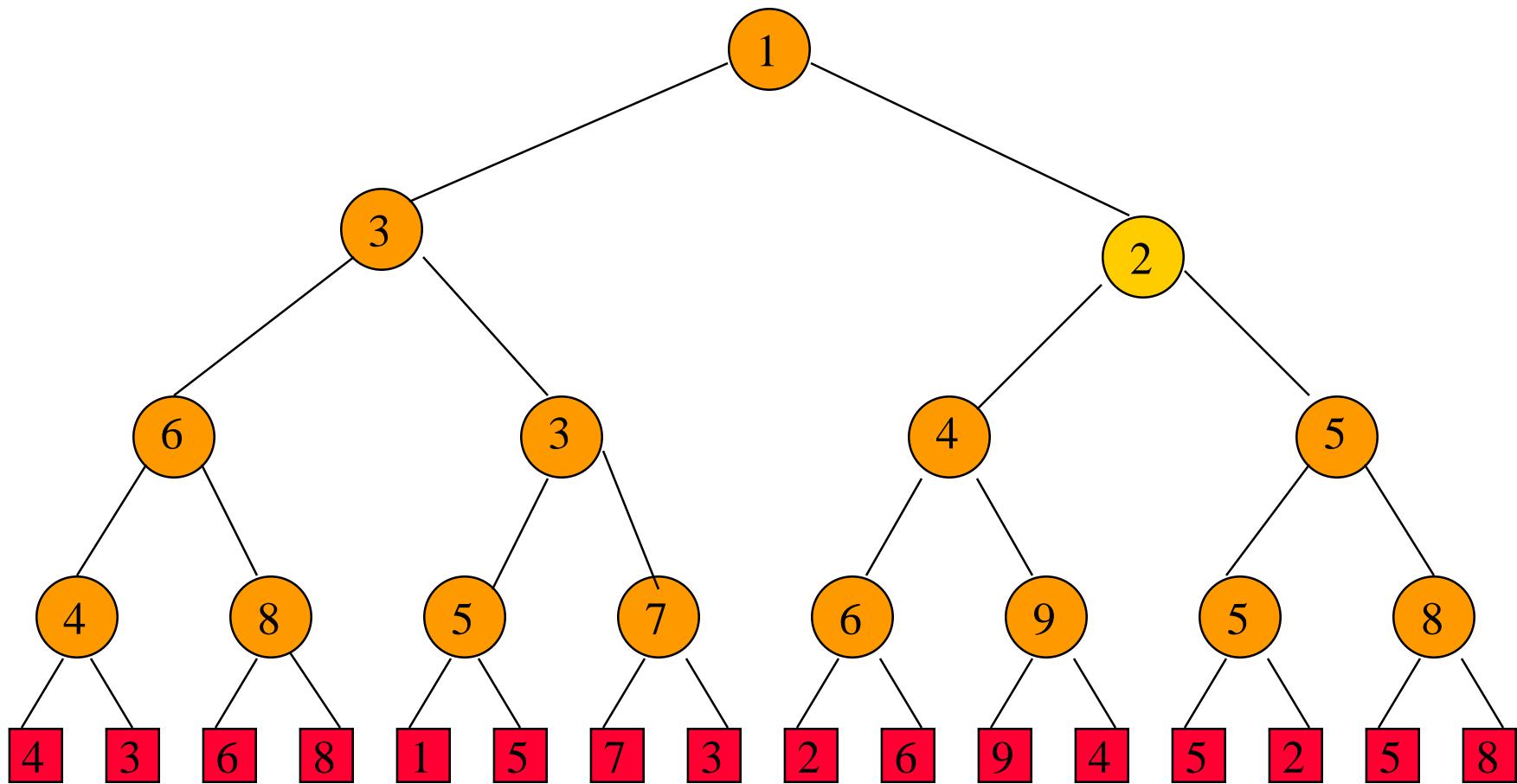
# 16 тоглогчтой min хожигдлын мод



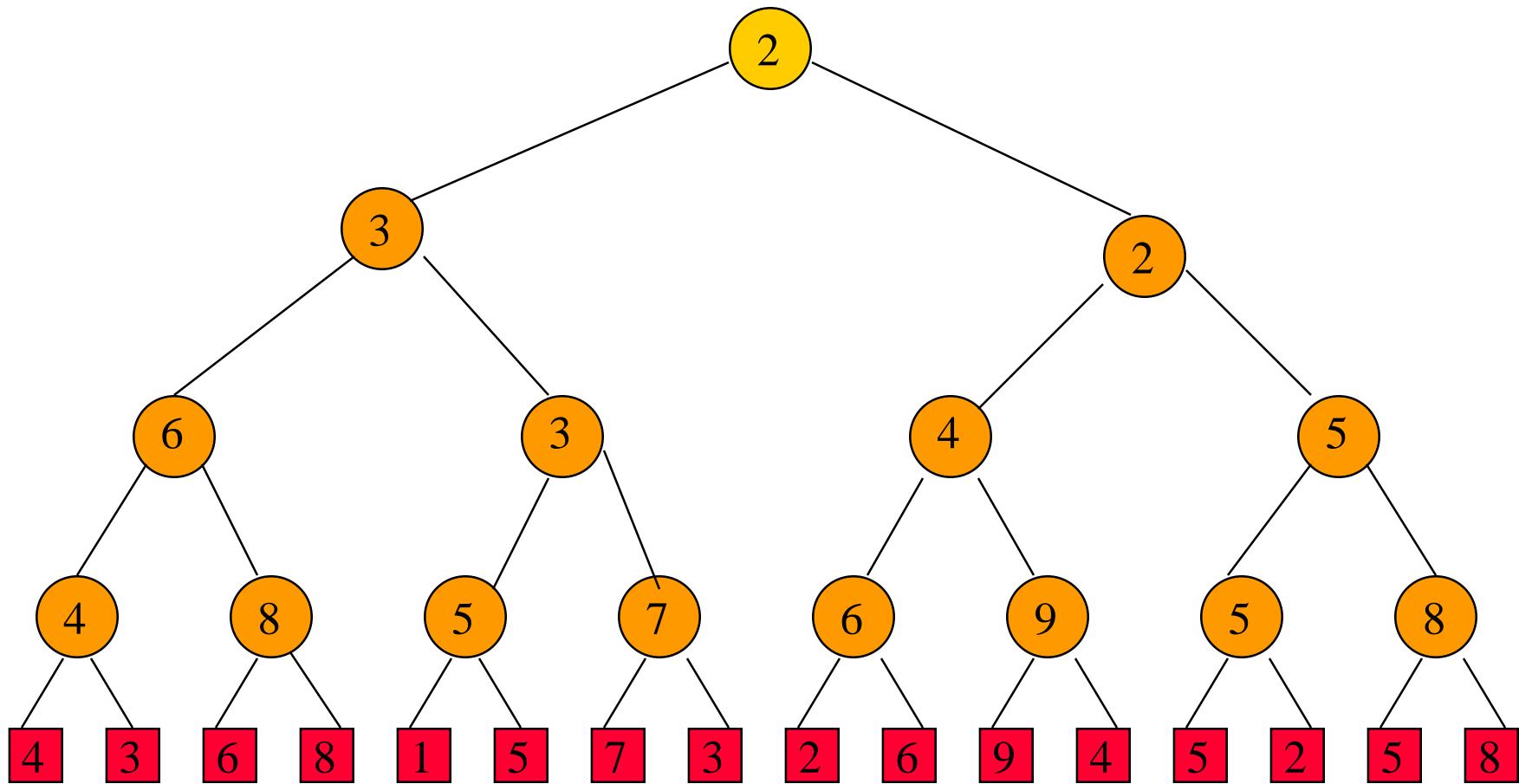
# 16 тоглогчтой min хожигдлын мод

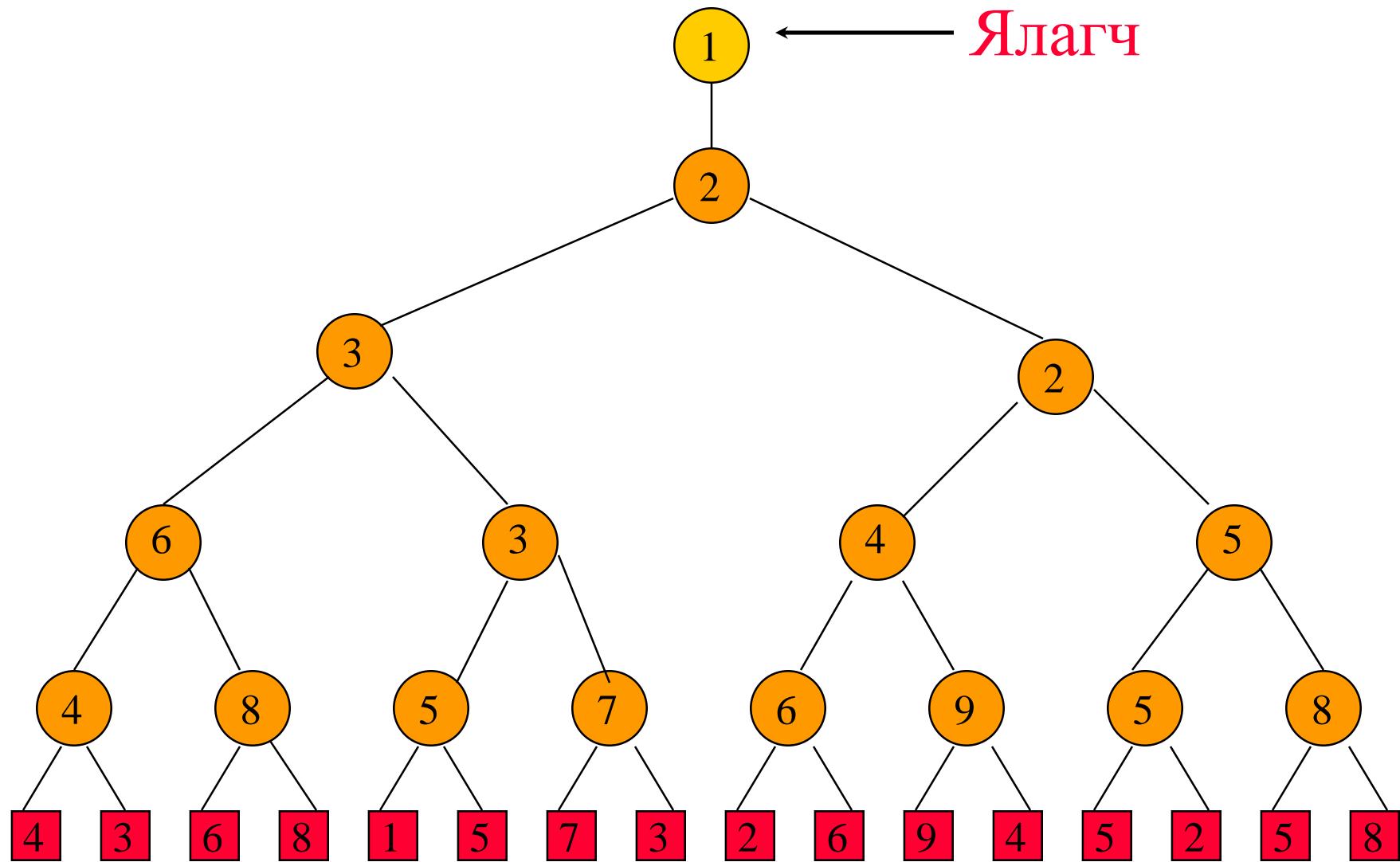


# 16 тоглогчтой min хожигдлын мод



# 16 тоглогчтой min хожигдлын мод

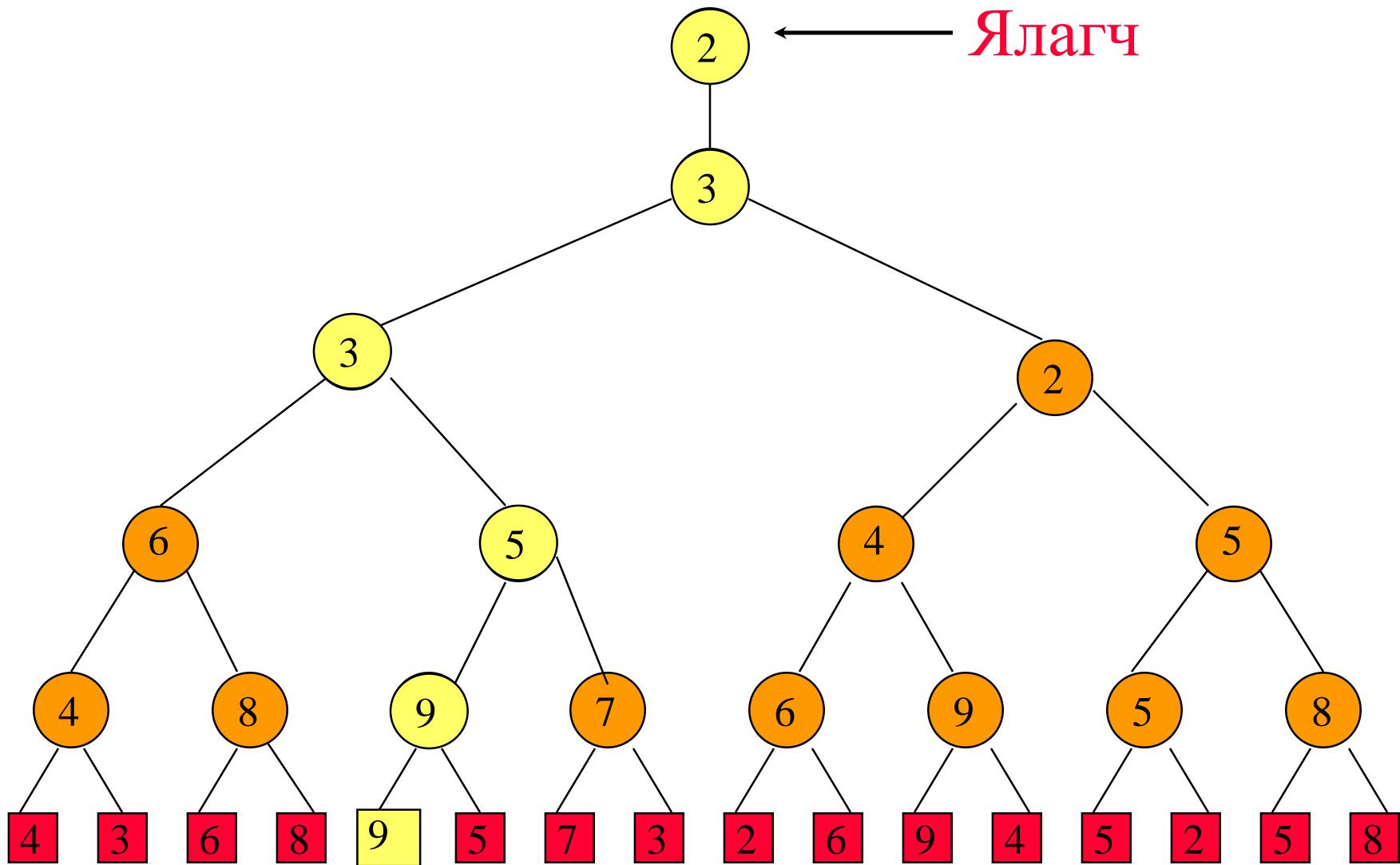




# Хожигдлын модыг идэвхижүүлэх хугацаа



- Тоглолтын зангилаа бүрт нэг тоглолт.
- Зүүн хүүхэд ялахад нэг хадгалалт.
- Нийт хугацаа  $O(n)$ .
- Яг нарийндаа  $\Theta(n)$ .



Ялагчийг 9 -өөр сольж, тогтолтуудыг дахин хийх.

# Дахин тоглох хугацаа

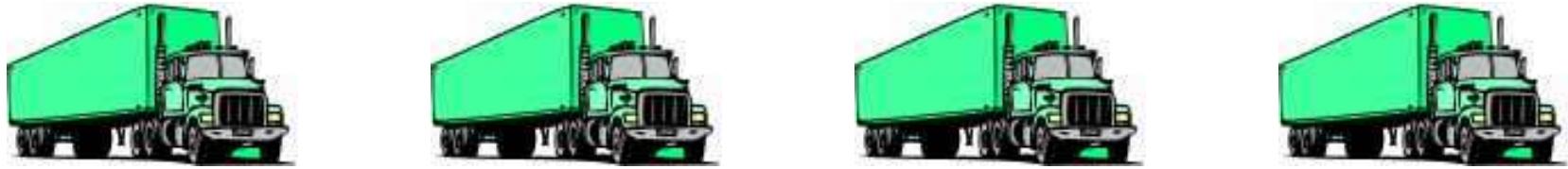


- Тоглолтын зангилаатай түвшин бүрт нэг тоглолт.
- $O(\log n)$
- Яг нарийндаа  $\Theta(\log n)$ .

# ТЭМЦЭЭНИЙ МОДНЫ ӨӨР ХЭРЭГЛЭЭ

- Гадаад эрэмбэлэлтийн үед **k**-замтай холилт хэрэглэх
- Машинд ачаа ачих

# Машинд ачаа ачих



- **n** ачааг машинуудад ачих
- Ачаа бүр тодорхой жинтэй
- Машин бүрийн даац **c** тонн
- Машини тоог багасгах

# Машинд ачаа ачих

$n = 5$  ачаа

жин  $[2, 5, 6, 3, 4]$

Машины даац  $c = 10$

Ачааг зүүнээс баруун тийш ачна. Хэрвээ ачаа машинд багтахгүй бол дараачийн машинд ачна.

# Машинд ачаа ачих

$n = 5$  ачаа

жин  $[2, 5, 6, 3, 4]$

Машины даац  $c = 10$

машин1 =  $[2, 5]$

машин2 =  $[6, 3]$

машин3 =  $[4]$

2 машин хэрэгтэй ч 3 машин ашиглалаа

# Машинд ачаа ачих

$n = 5$  ачаа

жин  $[2, 5, 6, 3, 4]$

Машины даац  $c = 10$

машин1  $= [2, 5, 3]$

машин2  $= [6, 4]$

# Хайрцганд савлах

- **n** зүйлийг хайрцганд савлах
- Зүйл бүхэн хэмжээтэй
- Хайрцааг бүрийн хэмжээ **c**
- Хайрцгийн тоог багасгах

# Хайрцганд савлах

Машинд ачаа ачих нь хайрцганд савлахтай адил.

Машин бол савлах (ачих) хайрцаг.

Ачаа бол зүйл/элемент.

Савлах хайрцгийн тоог багасгах бодлого бол NP-hard төрлийн бодлого.

Хэд хэдэн хурдан эвристик санааг дэвшүүлж болно.

# Хайрцгийг савлах эвристик

- Эхлэн дүүргэлт(First Fit).
  - Хайрцуудыг зүүнээс баруун тийш дараалуулна.
  - Юмс/зүйлсийг өгөгдсөн дарааллаар нэг нэгээр нь савлана.
  - Тухайн зүйлийг багтах хамгийн зүүн хайрцагт хийнэ.
  - Тухайн зүйлийг багтаах хайрцаг байхгүй бол шинэ хайрцгаас эхэлнэ.

# Эхлэн дүүргэлт

$n = 4$

жин = [4, 7, 3, 6]

багтаамж = 10



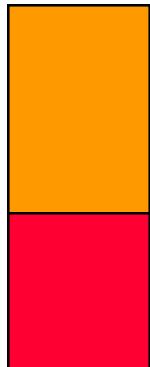
Улааныг эхний хайрцагт.

# Эхлэн дүүргэлт

$n = 4$

жин = [4, 7, 3, 6]

багтаамж = 10



Дараа нь хөхийг хийе.

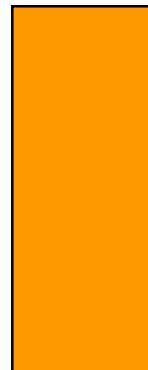
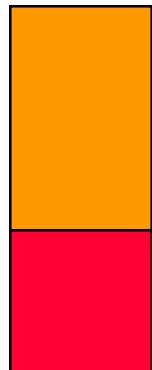
Багтахгүй тул шинэ хайрцагт хийе.

# Эхлэн дүүргэлт

$n = 4$

жин = [4, 7, 3, 6]

багтаамж = 10

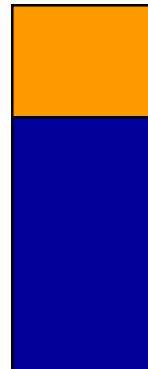
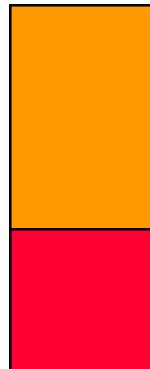


# Эхлэн дүүргэлт

$n = 4$

жин = [4, 7, 3, 6]

багтаамж = 10



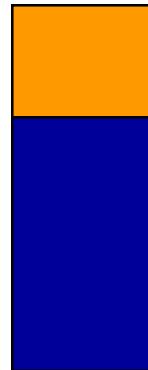
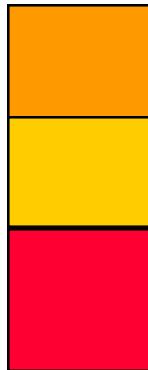
Шарыг эхний хайрцагт  
хийе.

# Эхлэн дүүргэлт

$n = 4$

жин = [4, 7, 3, 6]

багтаамж = 10



Ногооныг савлая.

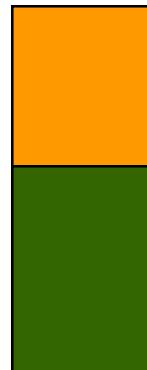
Шинэ хайрцаг хэрэгтэй.

# Эхлэн дүүргэлт

$n = 4$

жин = [4, 7, 3, 6]

багтаамж = 10



Оновчтой биш.  
2 хайрцаг  
хангалттай.

# Хайрцгийг савлах эвристик

- Бууруулсан эхлэн дүүргэлт(FF Decreasing).
  - Юмсыг буурах дарааллаар эрэмблэнэ.
  - Эхлээд дүүргэх аргыг хэрэглэнэ.

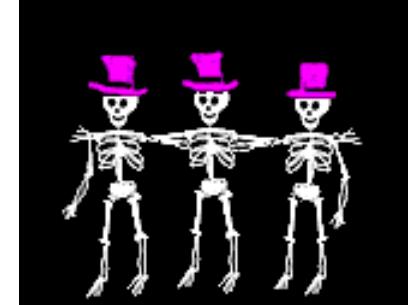
# Хайрцгийг савлах эвристик

- Сайн дүүргэлт(Best Fit).
  - Юмыг өгөгдсөн дарааллаар нэг нэгээр нь савлана.
  - Тухайн юмыг савлах хайрцгийг тогтоохын тулд эхлээд түүнийг багтаах хайрцгуудын **S** олонлогийг олно.
  - Хэрвээ **S хоосон бол**, тухайн зүйлийг шинэ хайрцагт хийнэ.
  - Үгүй бол, тухайн юмыг **S** –ийн хамгийн бага чөлөөт багтаамжтай хайрцагт хийнэ.

# Хайрцгийг савлах эвристик

- Бууруулсан сайн дүүргэлт(BF Decreasing).
  - Юмсыг буурах дарааллаар нь эрэмбэлнэ.
  - Сайн дүүргэлтийн аргыг хэрэглэнэ.

# Чанарын ҮЗҮҮЛЭЛТ



- Эхлэн дүүргэлт, Сайн дүүргэлт:  
$$\text{Эвристик хайрцгийн тоо} \leq (17/10)(\text{Min хайрцгийн тоо}) + 2$$

Бууруулсан эхлэн дүүргэлт, Бууруулсан сайн дүүргэлт:

$$\text{Эвристик хайрцгийн тоо} \leq (11/9)(\text{Min хайрцгийн тоо}) + 4$$

# Эхлэн дүүргэлтийн хугацаа



max тэмцээний модыг ашиглана.

Тоглогчийн үүргийг **n** хайрцаг,  
тоглогчийн утгыг хайрцгийн  
чөлөөт багтаамж илэрхийлнэ.

$O(n \log n)$ , Үүнд **n** юмсын тоо.

# Граф

- $G = (V, E)$
- $V$  – оройн олонлог.
- Оройг бас зангилаа, цэг(пункт) гэж нэрлэдэг.
- $E$  – ирмэг(нуруу)-ийн олонлог.
- Ирмэг хоёр өөр оройг холбодог.
- Ирмэгийг бас нум, шулуун гэж нэрлэдэг.
- Чиглэлтэй ирмэг зүг чигийг заадаг  $(u, v)$ .

$u \longrightarrow v$

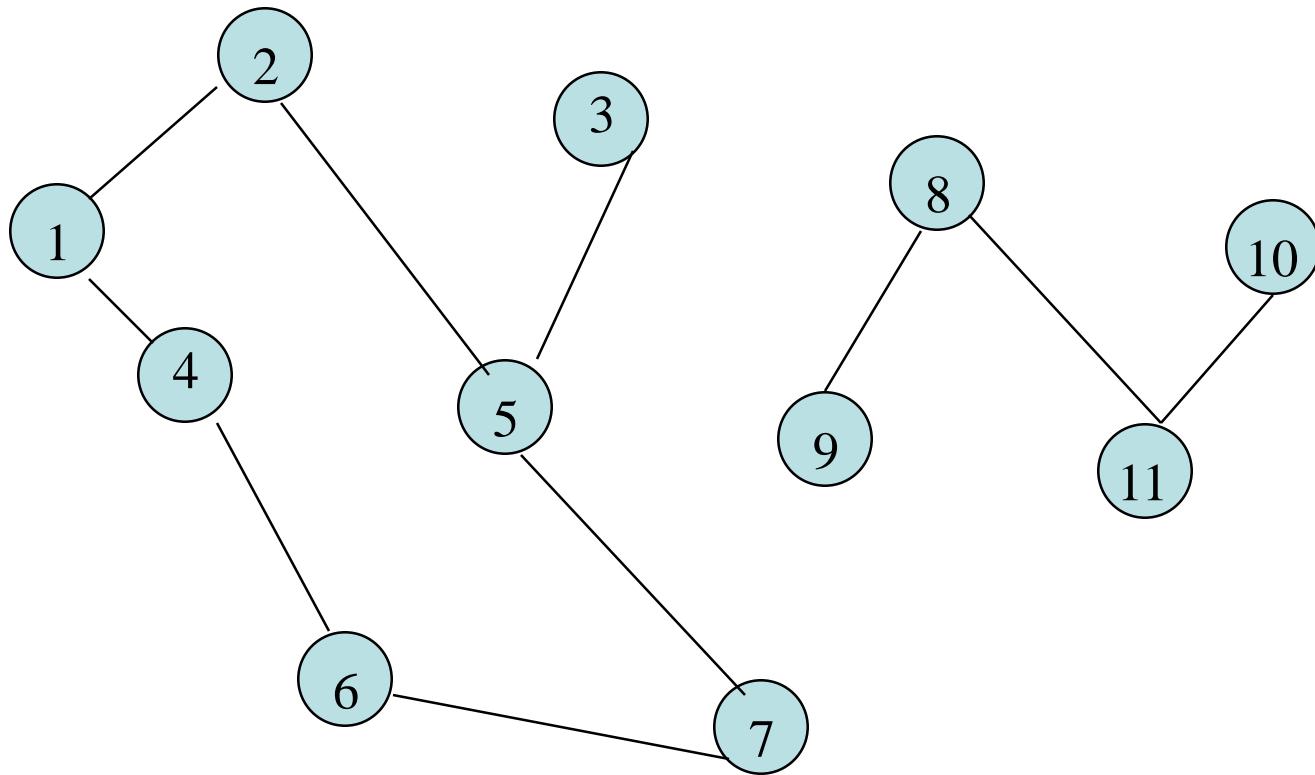
# Граф

- Чиглэлгүй ирмэг зүг чиг заахгүй  $(u, v)$ .

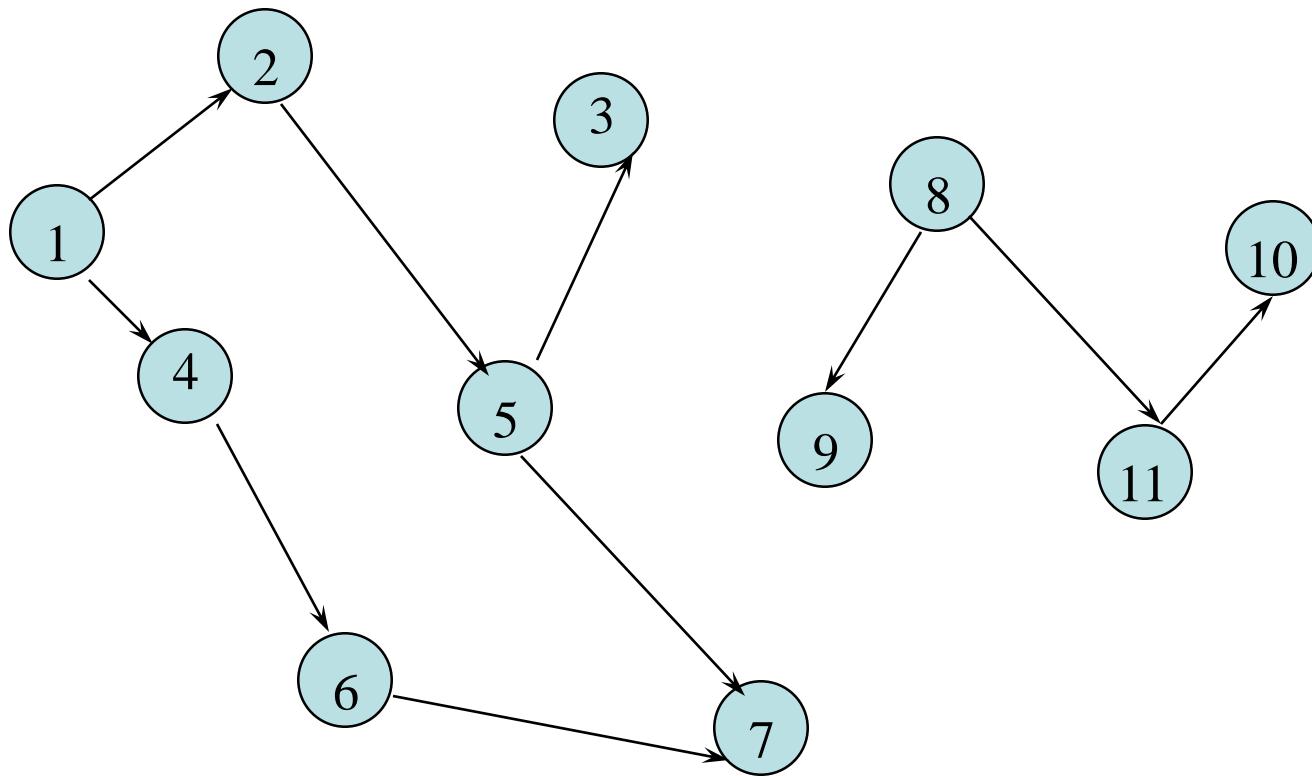
$u$  —  $v$

- Чиглэлгүй граф  $\Rightarrow$  зүг чигийг заасан ирмэггүй.
- Чиглэлтэй граф  $\Rightarrow$  ирмэг бүр зүг чигийг заана.

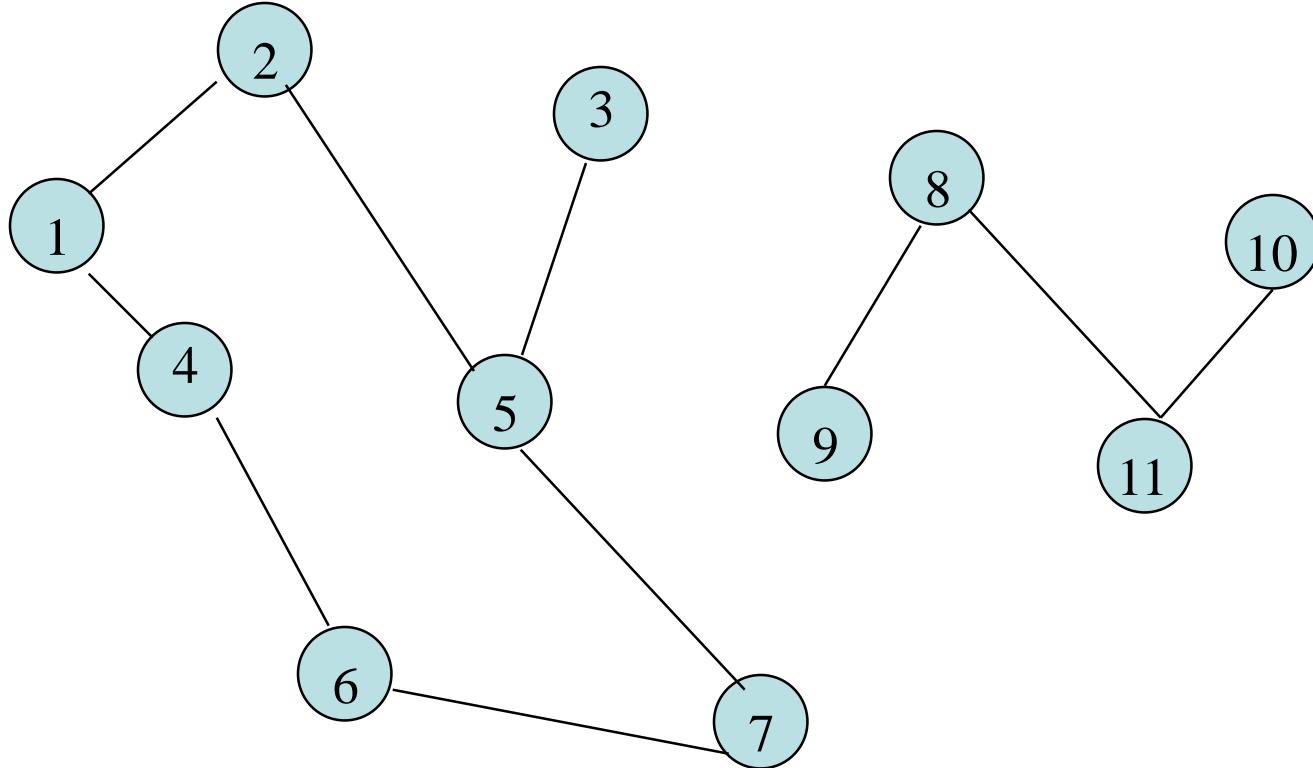
# Чиглэлгүй граф



# Чиглэлтэй граф (Digraph)

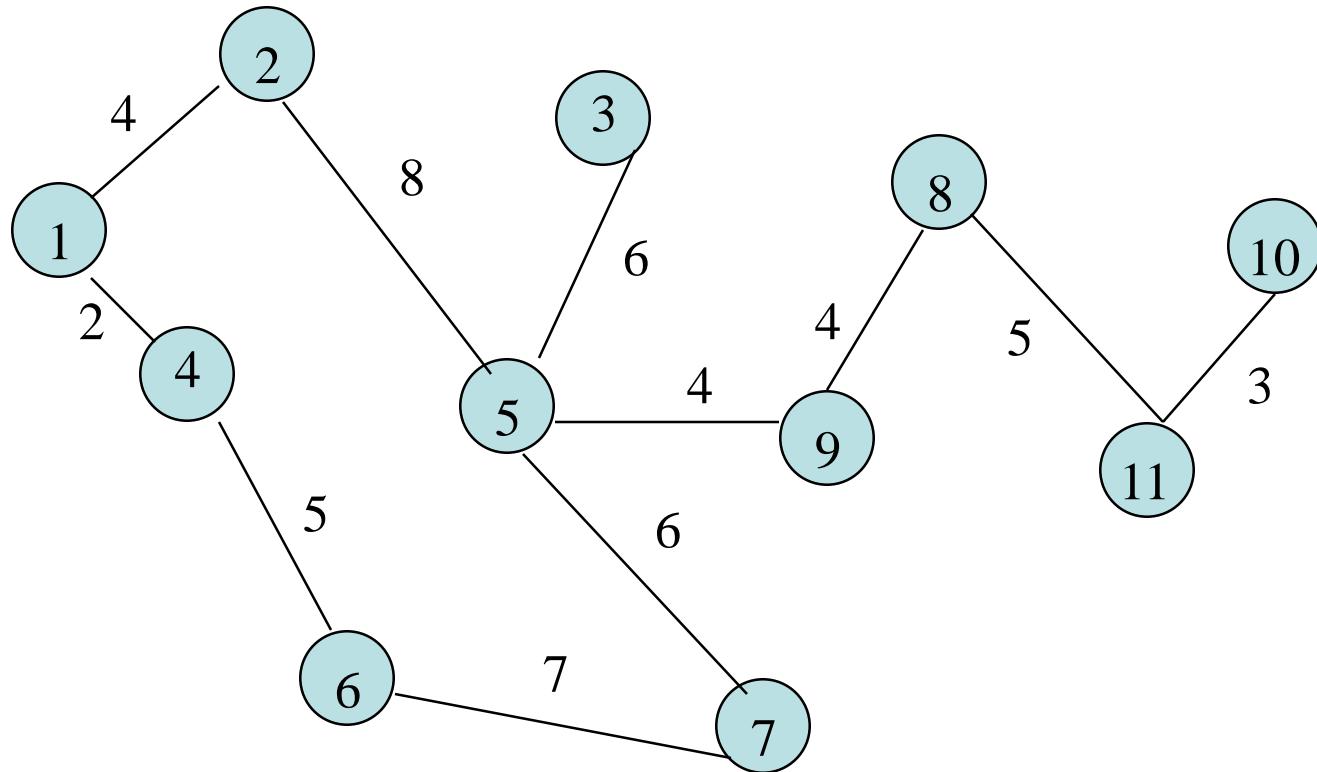


# Хэрэглээ—Холбооний Сүлжээ



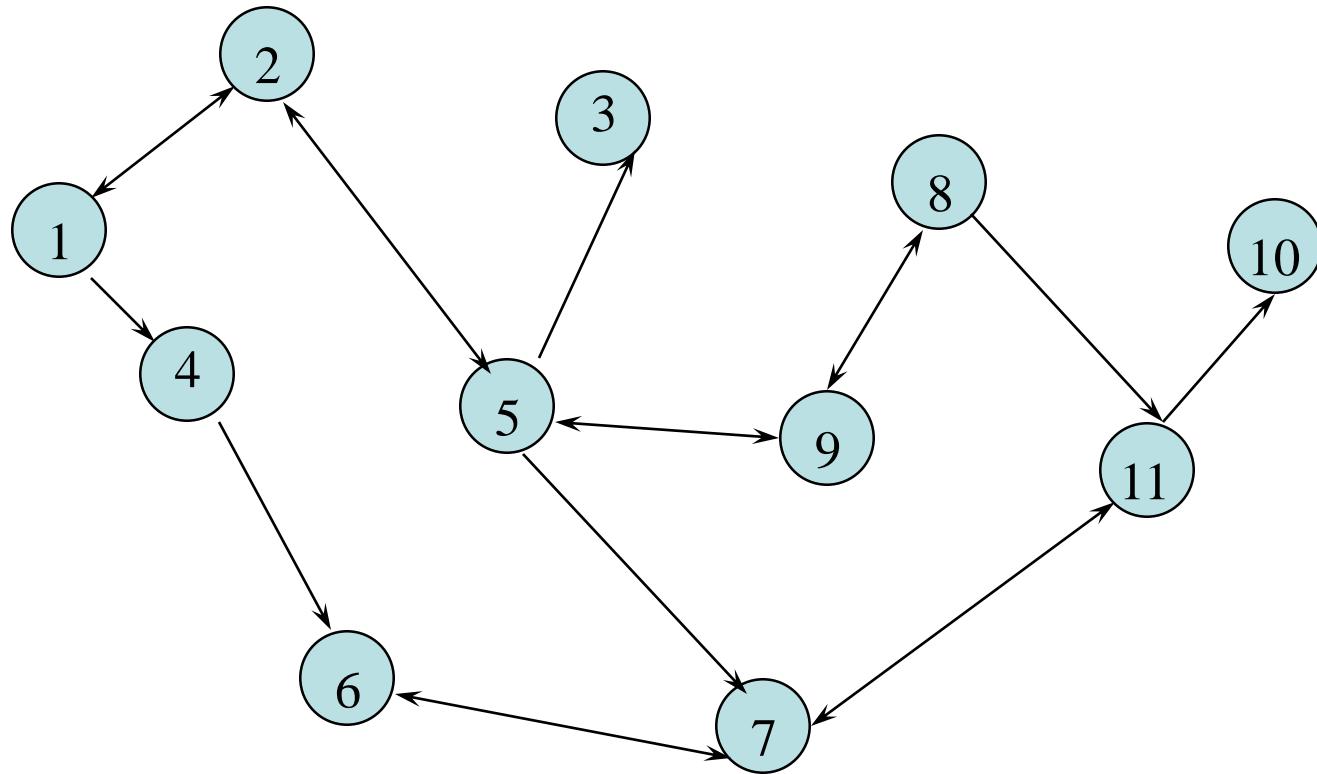
- Орой = хот, ирмэг = холбоос.

# Явах зайд/Хугацааны зураглал



- Орой = хот, ирмэгийн жин/ачаа = явах зайд/хугацаа.

# Гудамжны зураглал



- Зарим гудамж нэг чиглэлтэй.

# Төгс чиглэлгүй граф

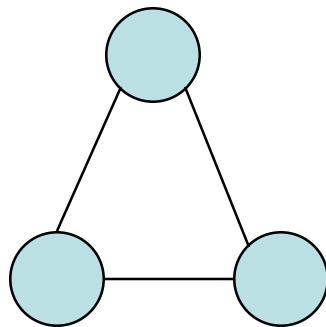
Байж болох бүх ирмэгтэй.



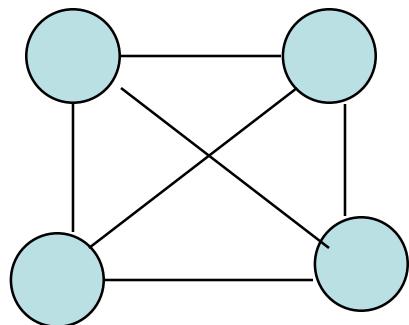
$$n = 1$$



$$n = 2$$



$$n = 3$$



$$n = 4$$

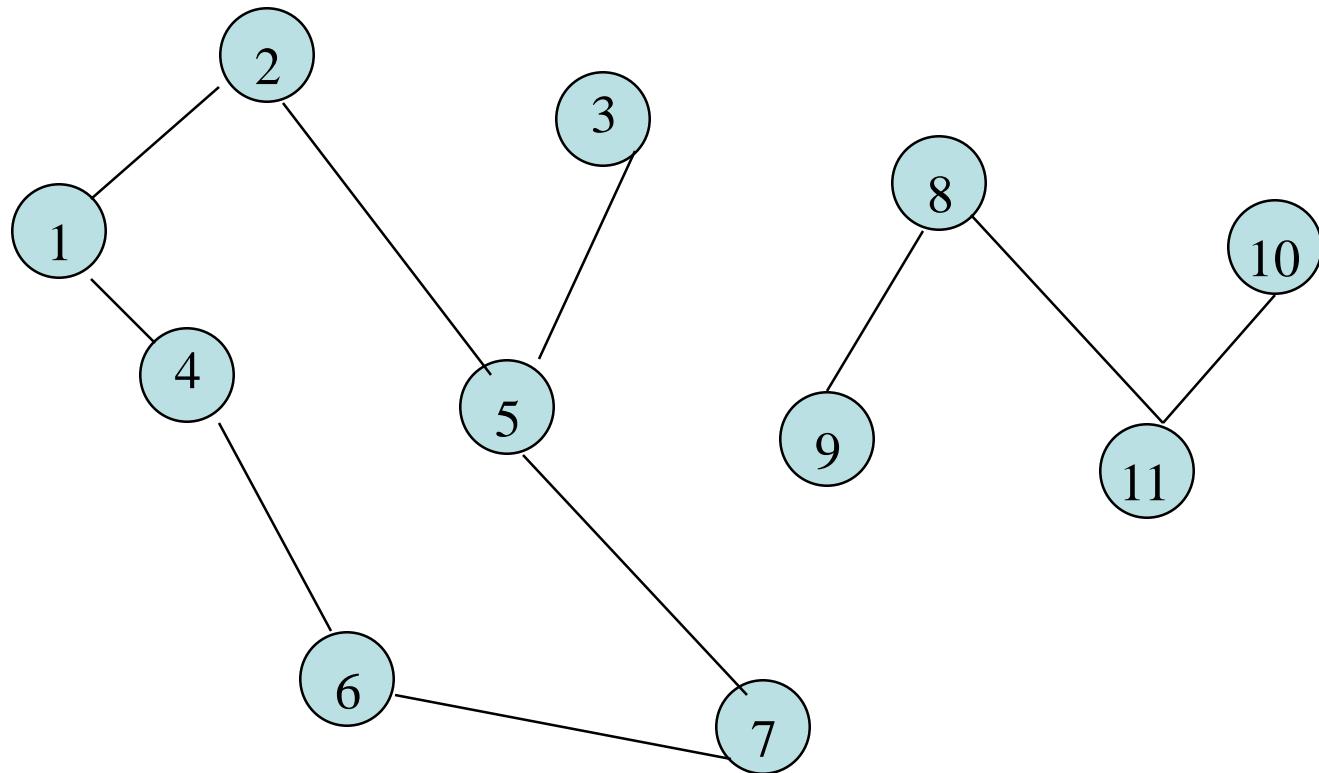
# Ирмэгийн тоо—чиглэлгүй граф

- Ирмэг бүр  $(u,v)$ ,  $u \neq v$  хэлбэртэй
- $n$  оройтой графын ийм хосын тоо  $n(n-1)$ .
- Нэгэнт  $(u,v)$  ирмэг  $(v,u)$  ирмэгтэй адил болохоор, төгс чиглэлгүй графын ирмэгийн тоо  $n(n-1)/2$ .
- Чиглэлгүй графын ирмэгийн тоо  $\leq n(n-1)/2$ .

# Ирмэгийн тоо—Чиглэлтэй граф

- Ирмэг бүр  $(u,v)$ ,  $u \neq v$  хэлбэртэй
- $n$  оройтой графын ийм хосын тоо  $n(n-1)$ .
- Нэгэнт  $(u,v)$  ирмэг  $(v,u)$  ирмэгтэй адилгүй тул төгс чиглэлтэй графын ирмэгийн тоо  $n(n-1)$ .
- Чиглэлтэй графын ирмэгийн тоо  $\leq n(n-1)$ .

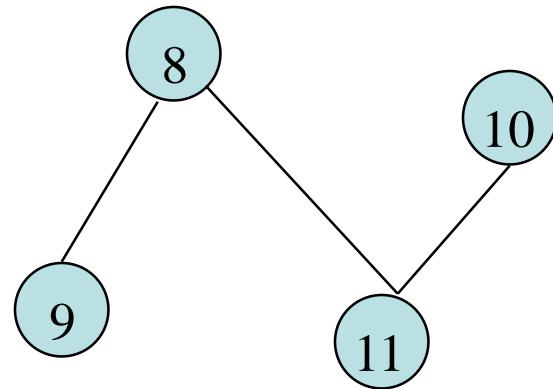
# Оройн зэрэг(Vertex degree)



Орой руу явсан ирмэгийн тоо.

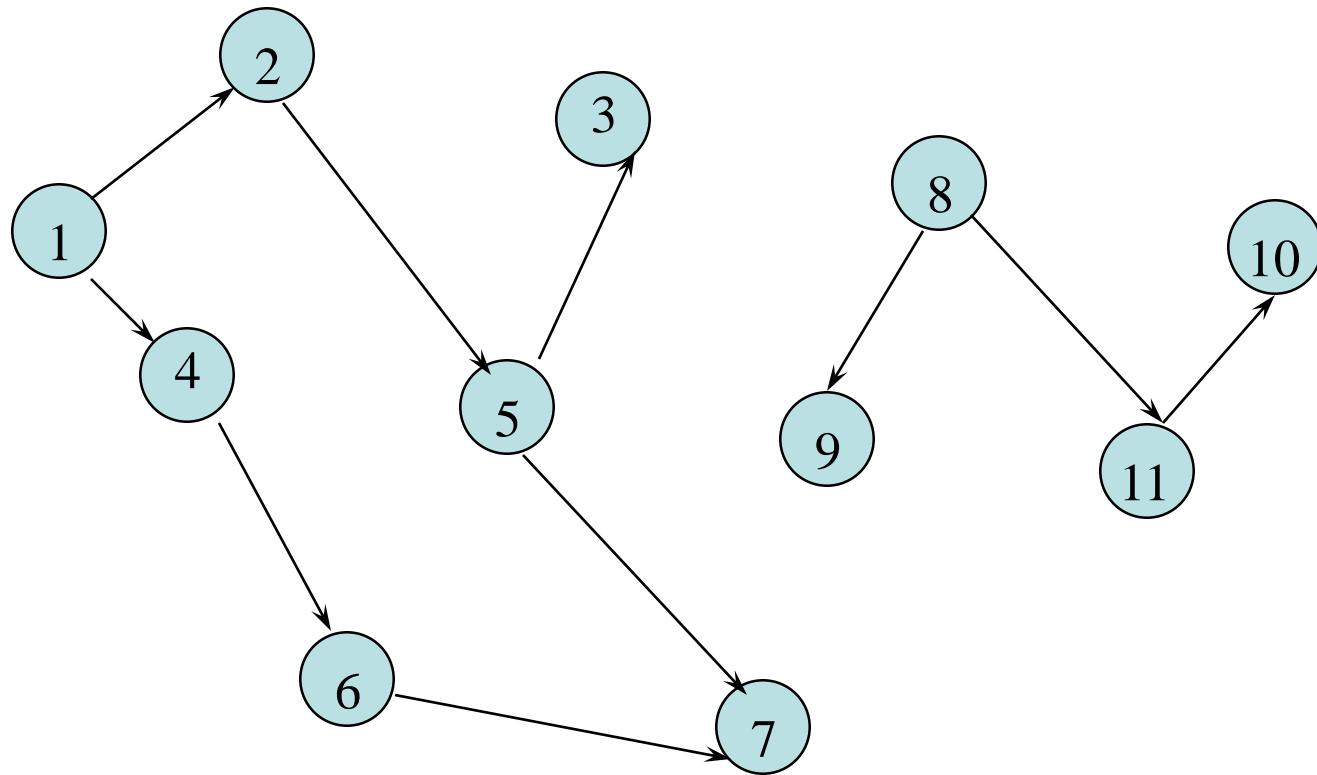
$\text{degree}(2) = 2$ ,  $\text{degree}(5) = 3$ ,  $\text{degree}(3) = 1$

# Оройн зэргийн нийлбэр



Зэргийн нийлбэр =  $2e$  (е ирмэгийн тоо)

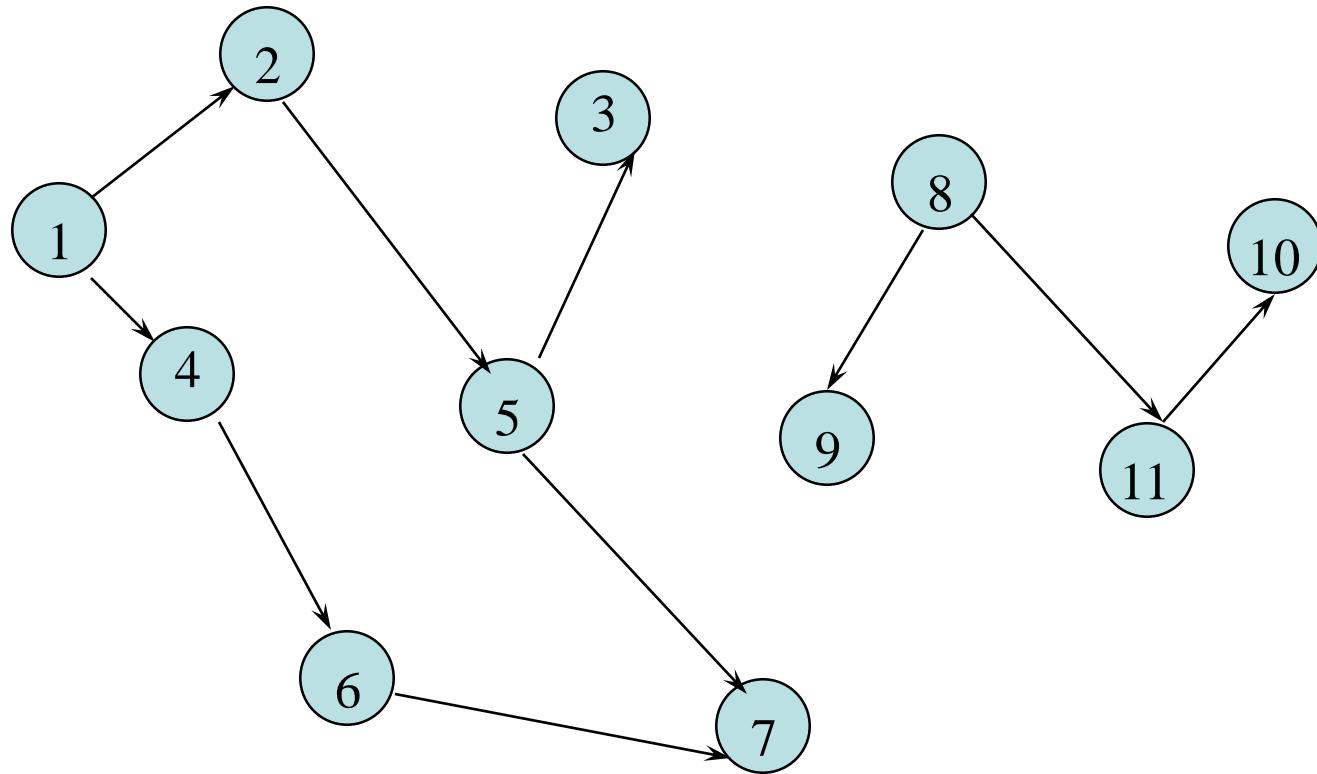
# Оройн орох зэрэг(In-Degree)



in-degree – орсон ирмэгийн тоо

$\text{indegree}(2) = 1$ ,  $\text{indegree}(8) = 0$

# Оройн гарах зэрэг(Out-Degree)



out-degree - гарсан ирмэгийн тоо

$\text{outdegree}(2) = 1$ ,  $\text{outdegree}(8) = 2$

# Орох ба гарах зэргийн нийлбэр

Ирмэг бүр **1-г** ямар нэг оройн орох зэрэгт, **1-г** нөгөө оройн гарах зэрэгт нэмэрлэдэг

Орох зэргийн нийлбэр = гарах зэргийн нийлбэр  
= **e**,  
үүнд **e** – digraph ирмэгийн тоо

# Графын үйлдлүүд ба дүрслэл

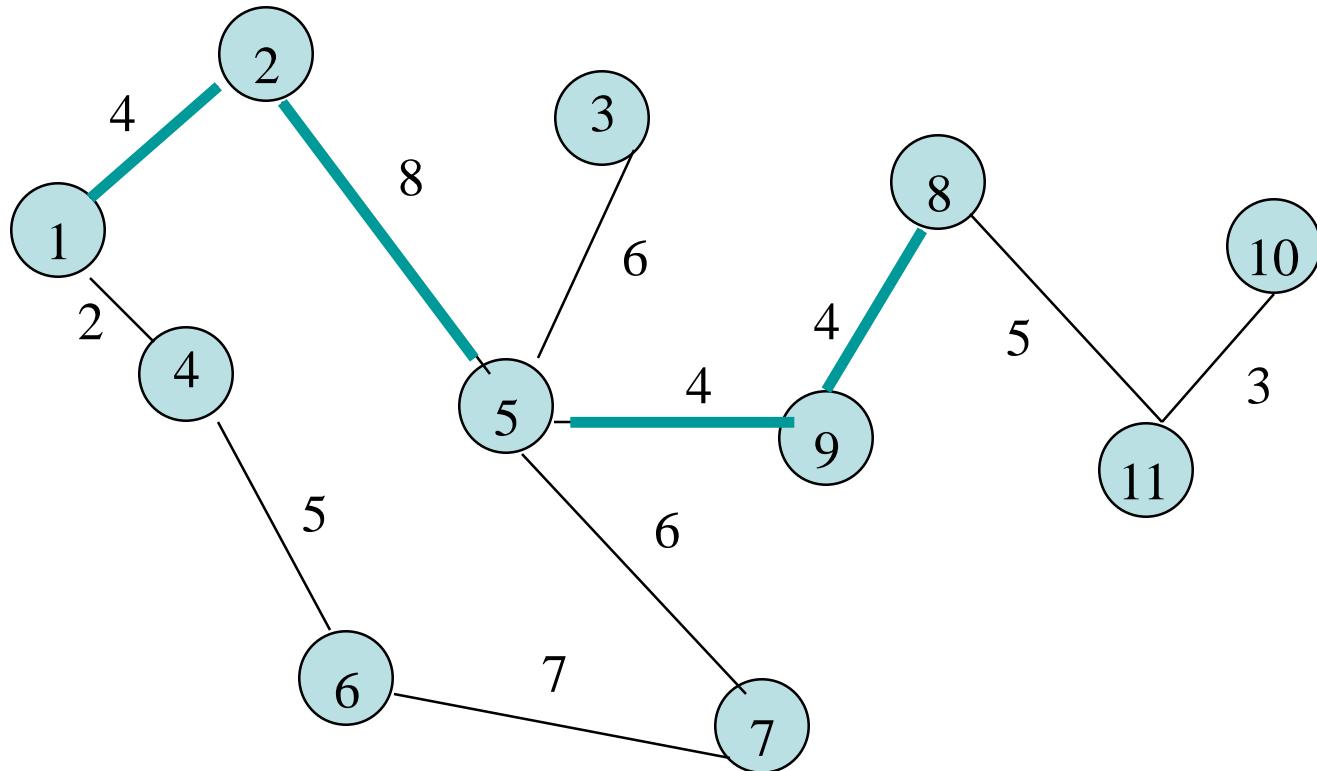


# Графын зарим бодлогууд

- Замын бодлого.
- Холболтын бодлого.
- Бүрхэгч модны (Spanning tree) бодлого.

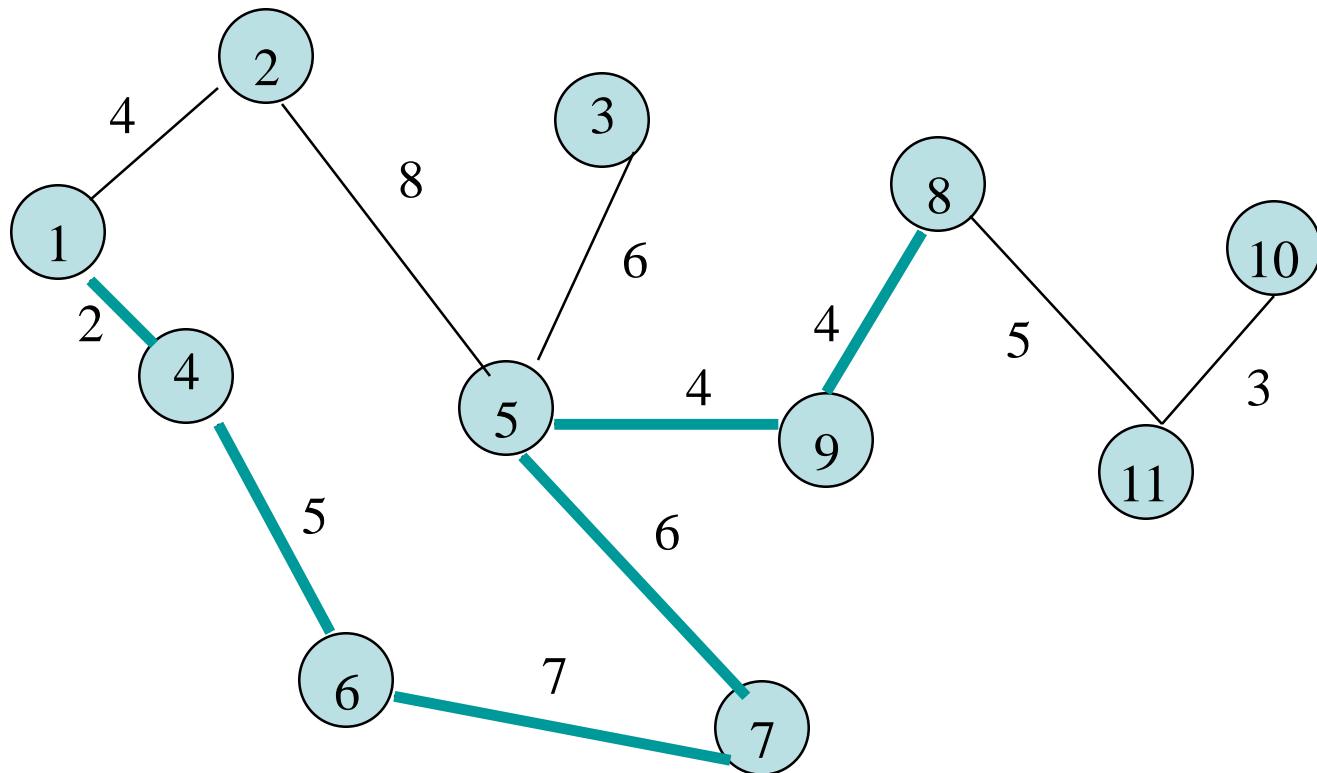
# Зам олох

1 – 8 хүрэх зам



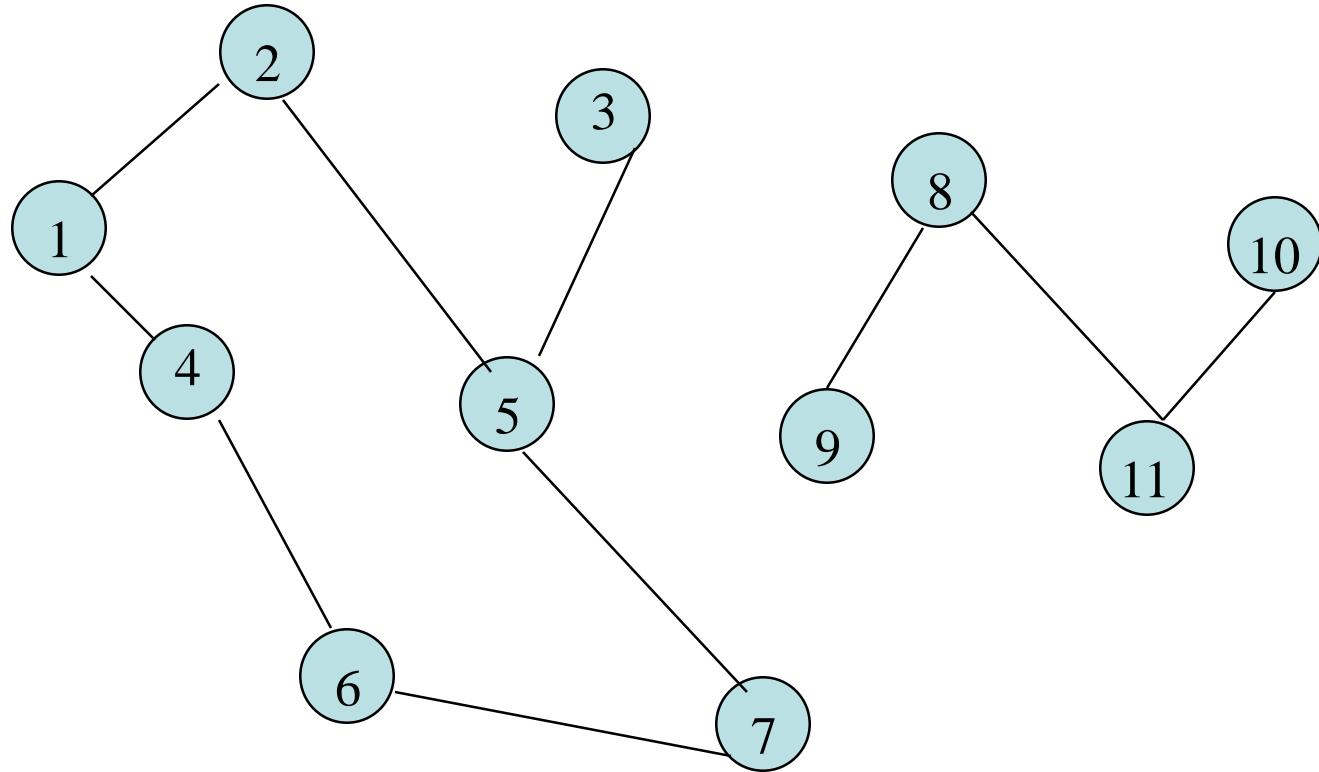
Замын урт **20.**

# 1 – 8 хүрэх өөр зам



Замын урт **28.**

# Замгүй байх жишээ

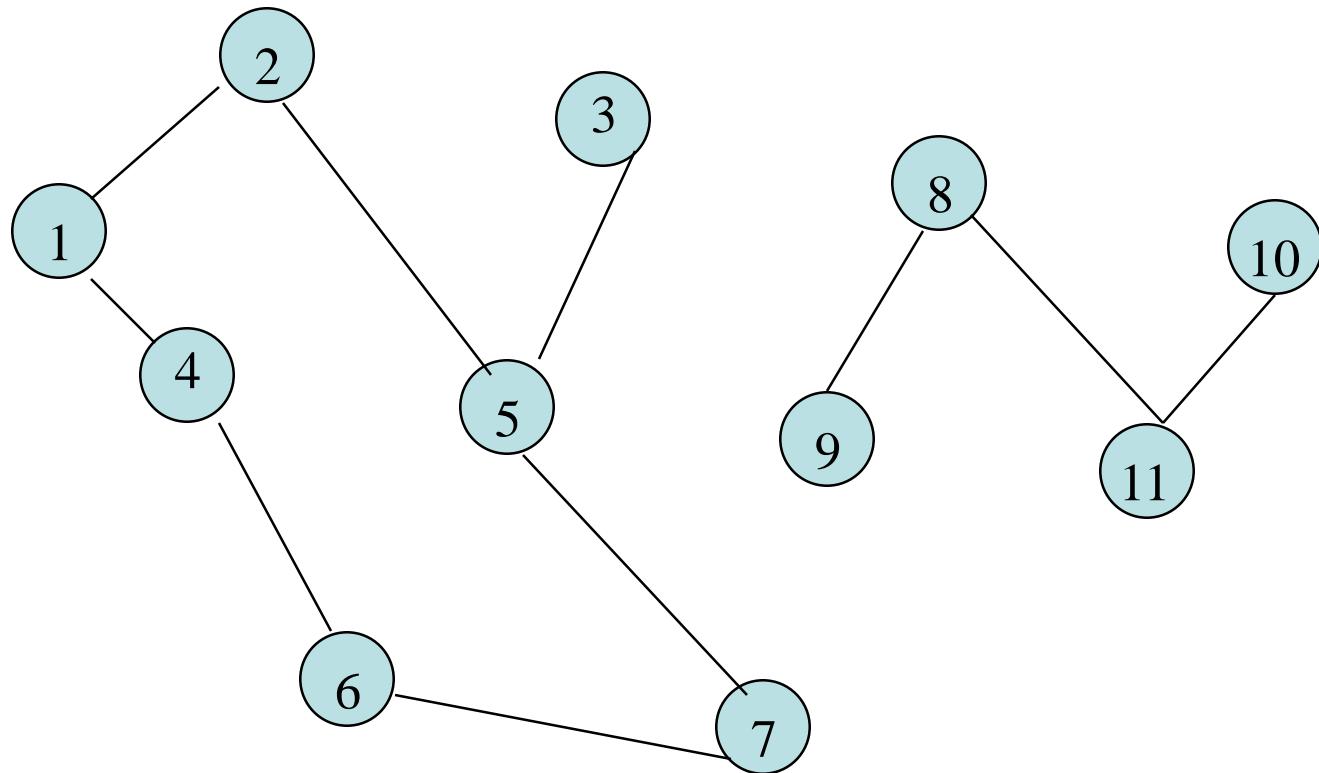


2 , 9 -ийн хооронд замгүй

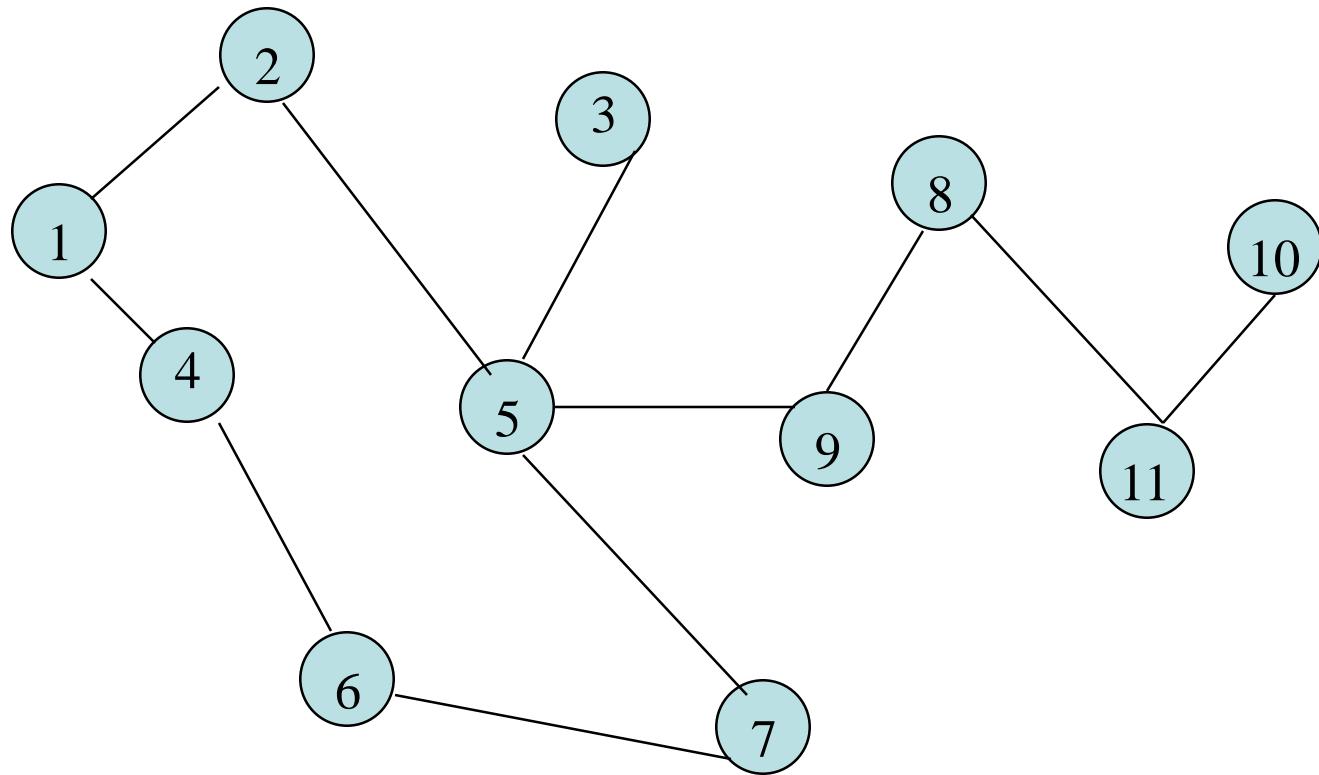
# Холбогдсон граф

- Чиглэлгүй граф.
- Хос орой бүрийг хооронд замтай.

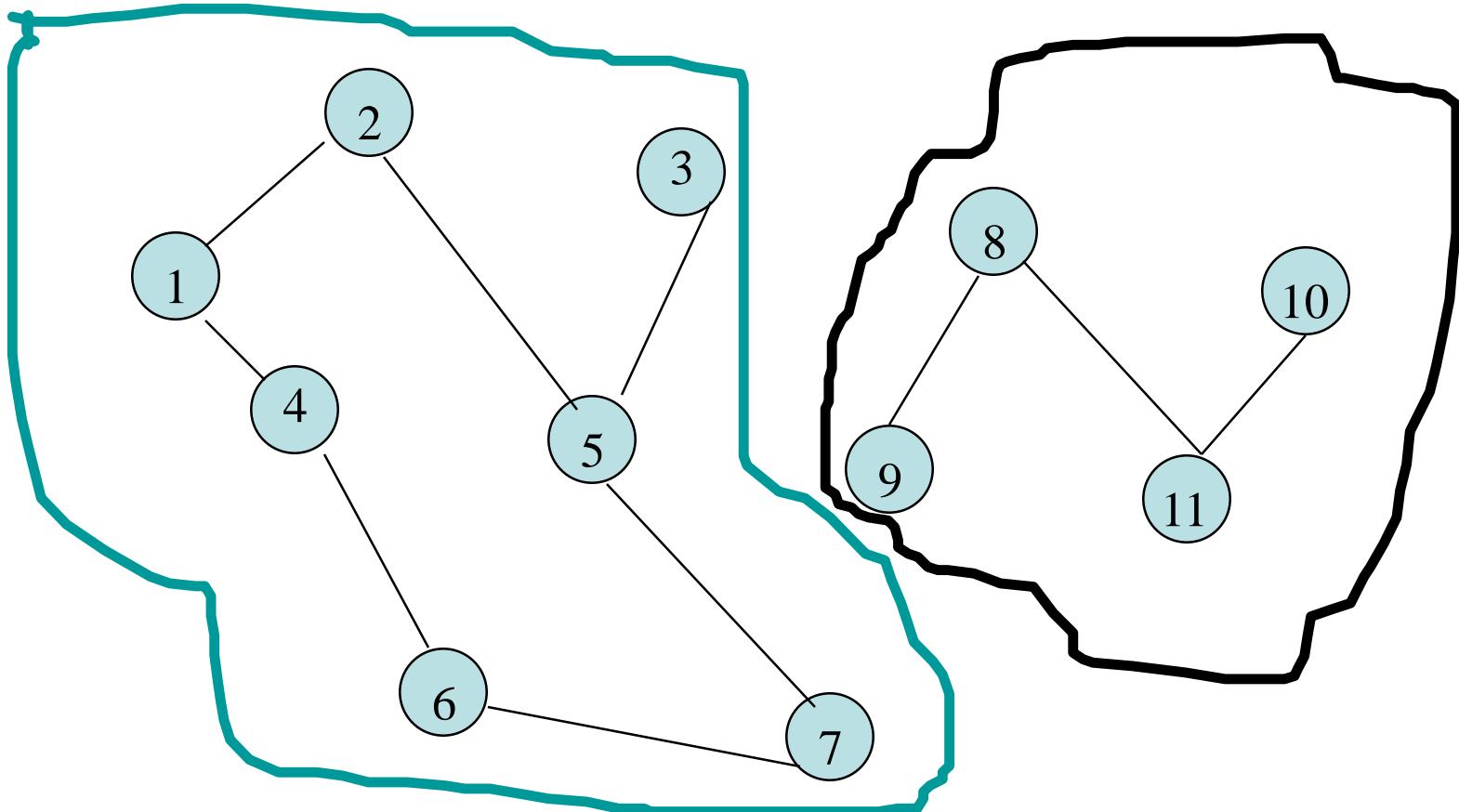
# Холбогдоогүй графын жишээ



# Холбогдсон графын жишээ



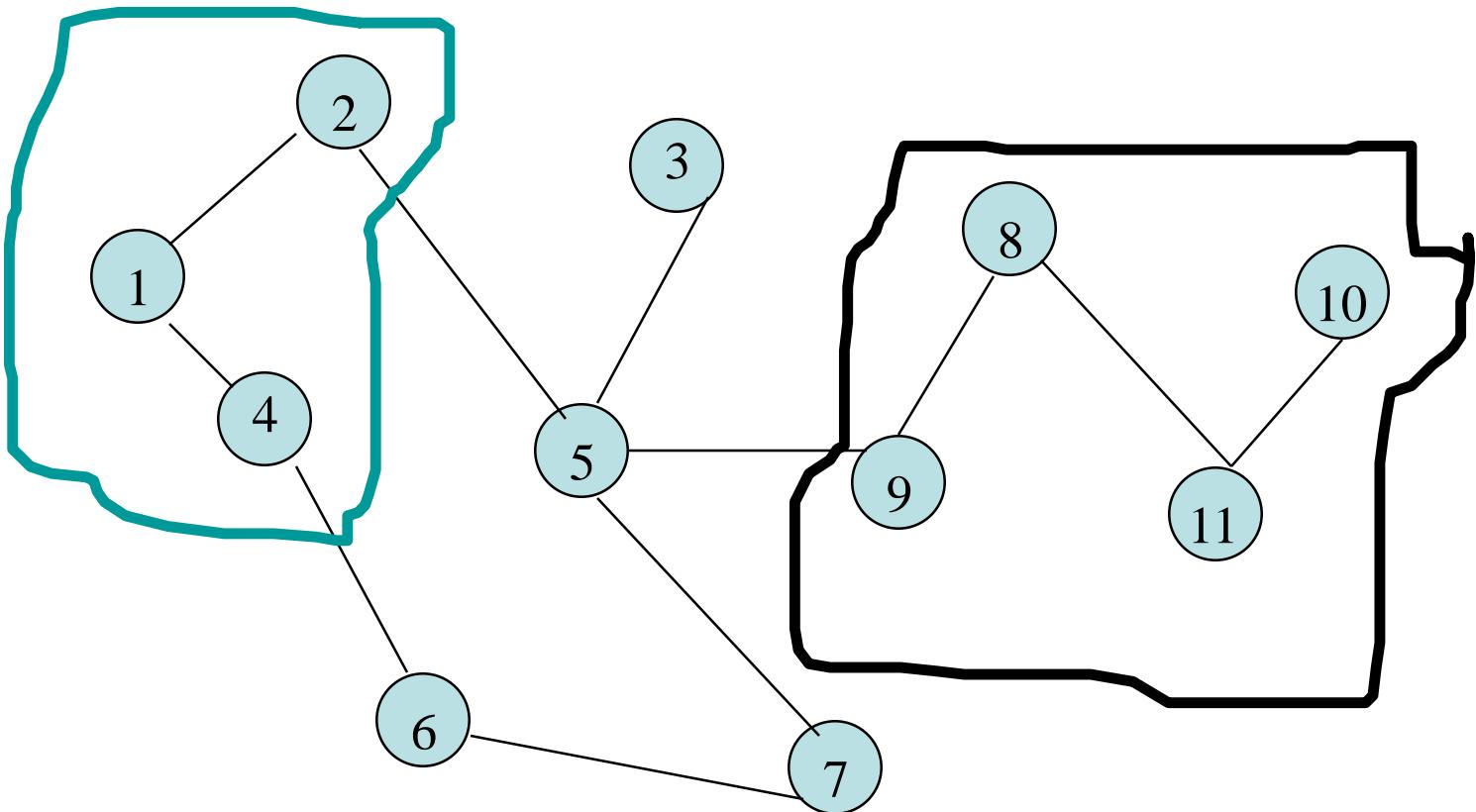
# Холбогдсон бүрдүүлбэрүүд



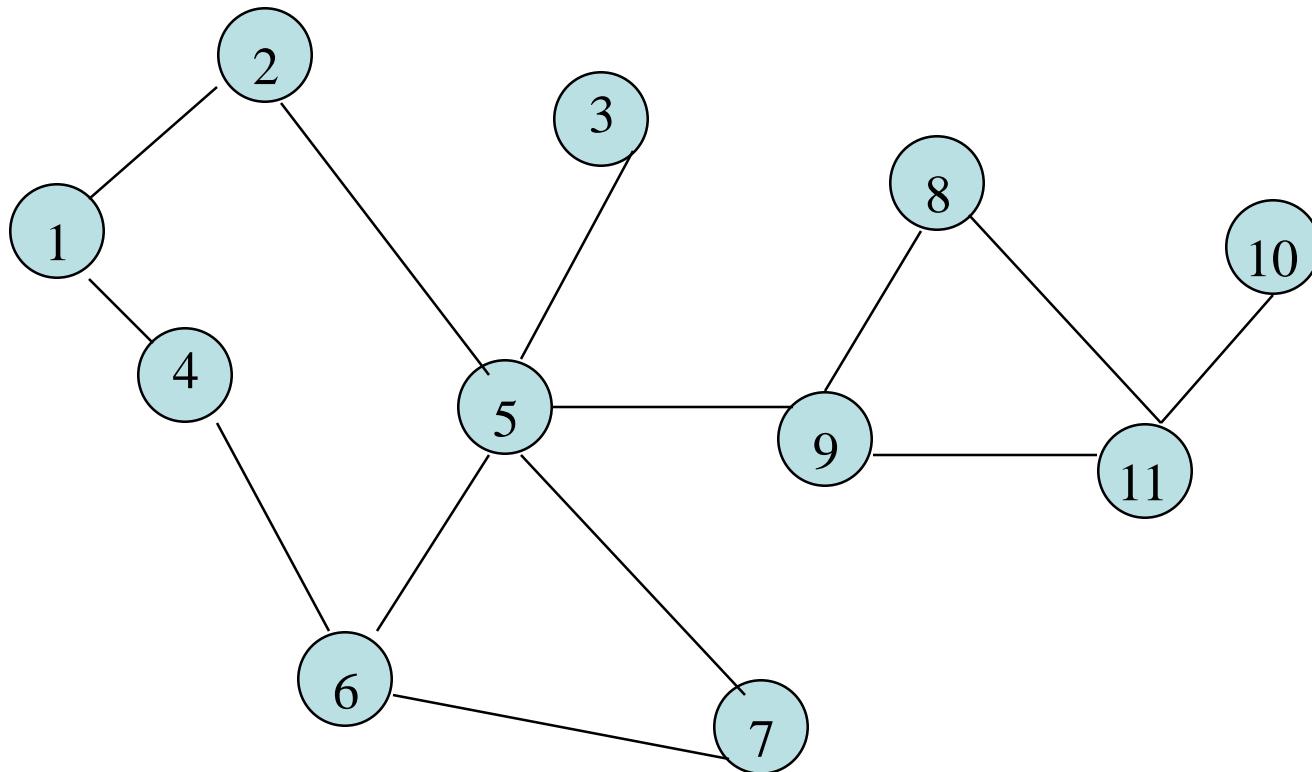
# Холбогдсон бүрдүүлбэрүүд

- Холбогдсон дэд граф.
  - Оригинал граф дээр орой, ирмэг нэмж болохгүй, салангид байдлаа хадгалдаг.
- Холбогдсон граф яг 1 бүрдүүлбэртэй.

# Бүрдүүлбэр биш



# Холбооны сүлжээ

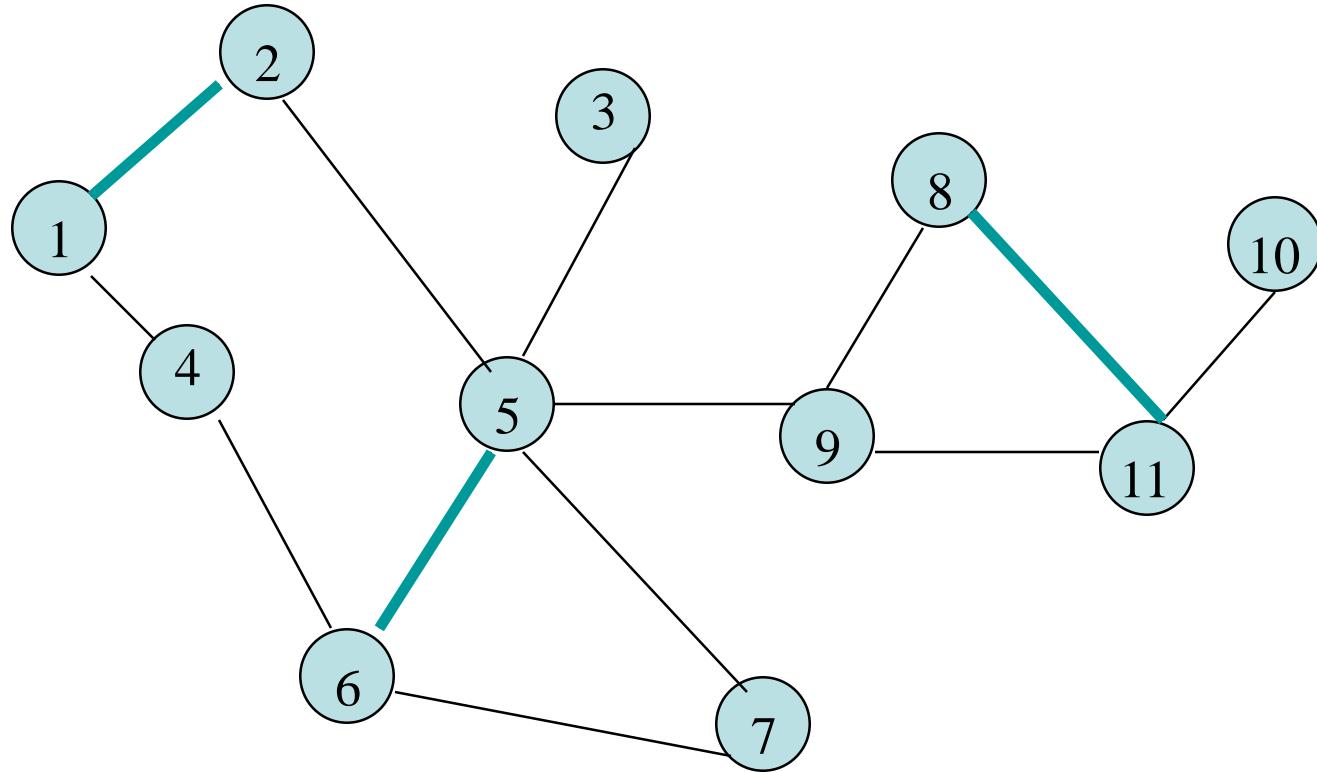


Ирмэг бүр байгуулж болох холбоос (ө.х., а боломжит холбоос).

# Холбооны сүлжээний бодлогууд

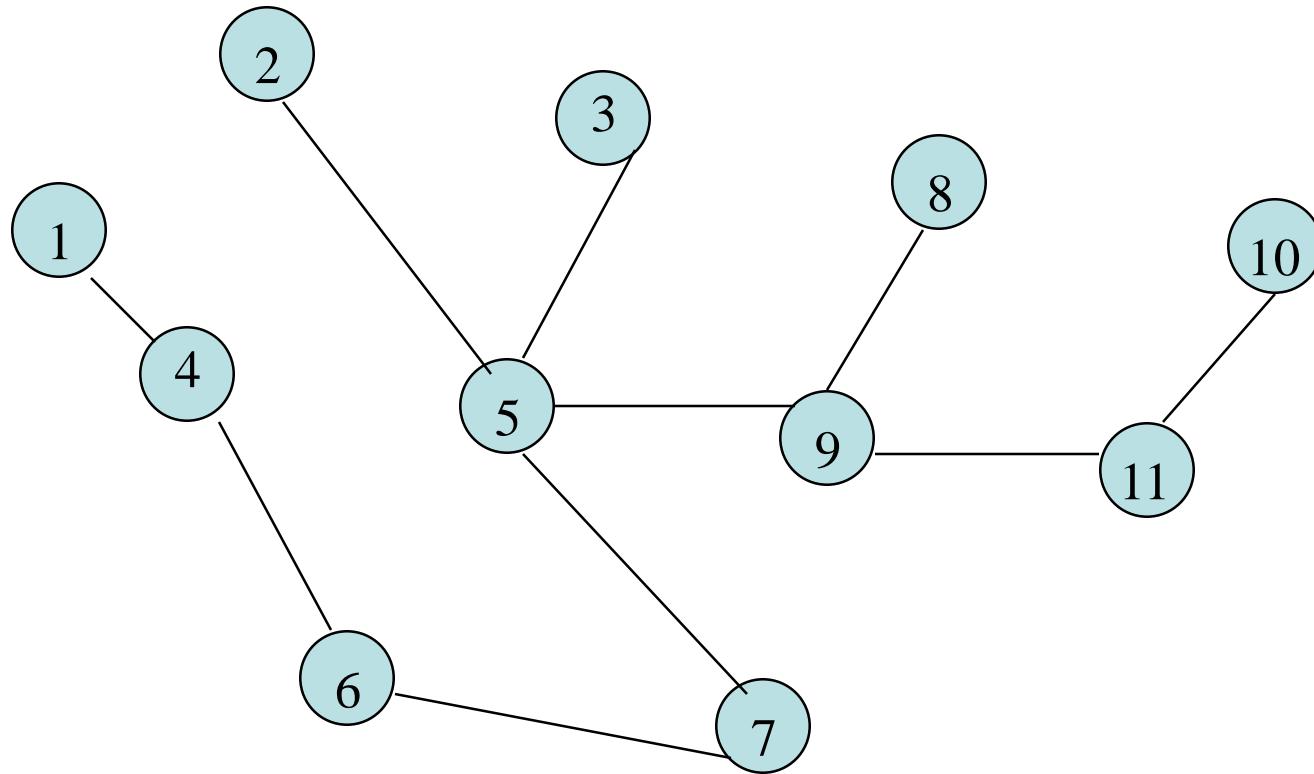
- Сүлжээ холбогдсон уу?
  - Дурын хоёр хот харьцаж чадах уу?
- Бүрдүүлбэрийг олох.
- Сүлжээ холбогдсон байхын тулд шаардлагатай хамгийн цөөн боломжит холбоосыг байгуулах.

# Цикл ба холболт



Цагирагнаас ирмэг устгахад холболтонд нөлөөлөхгүй.

# Цикл ба холболт



Бүх орой, цикл үүсгэхгүй минимум тооны  
ирмэгтэй холбогдсон дэд граф.



# Мод

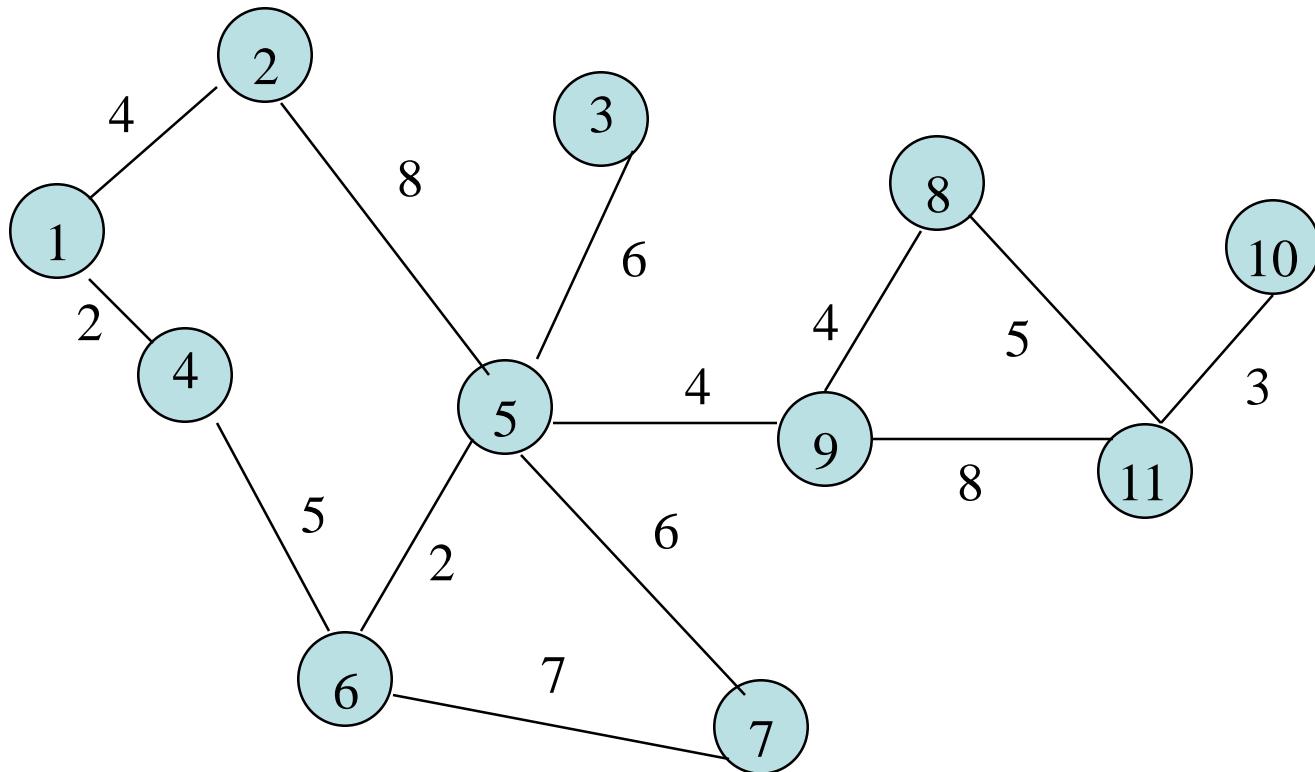


- Циклгүй, холбогдсон граф.
- $n-1$  ирмэгтэй,  $n$  оройтой холбогдсон граф.

# Бүрхэгч мод

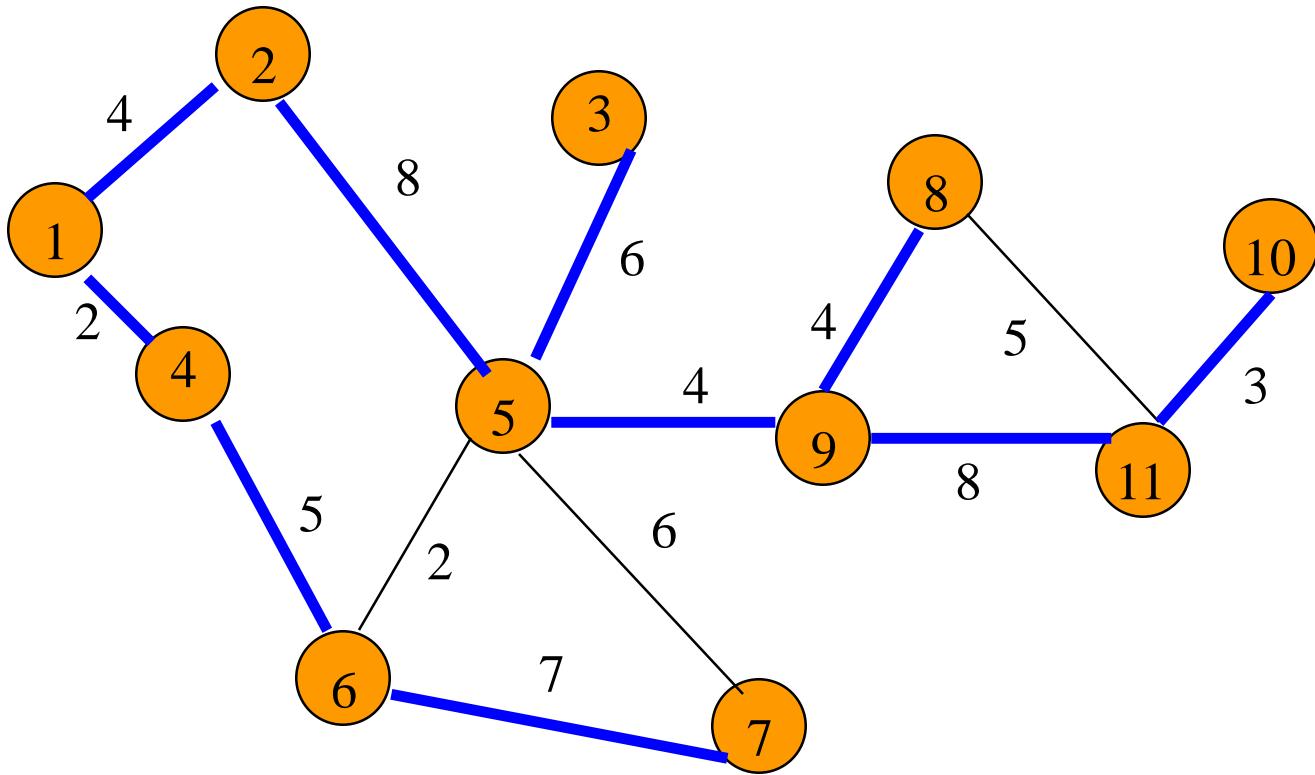
- Оригинал графын бүх оройг агуулсан дэд граф.
- Дэд граф нь мод.
  - Хэрвээ оригинал граф  $n$  оройтой бол, бүрхэлтийн мод  $n$  орой ,  $n-1$  ирмэгтэй.

# Min өртөгтэй бүрхэгч мод



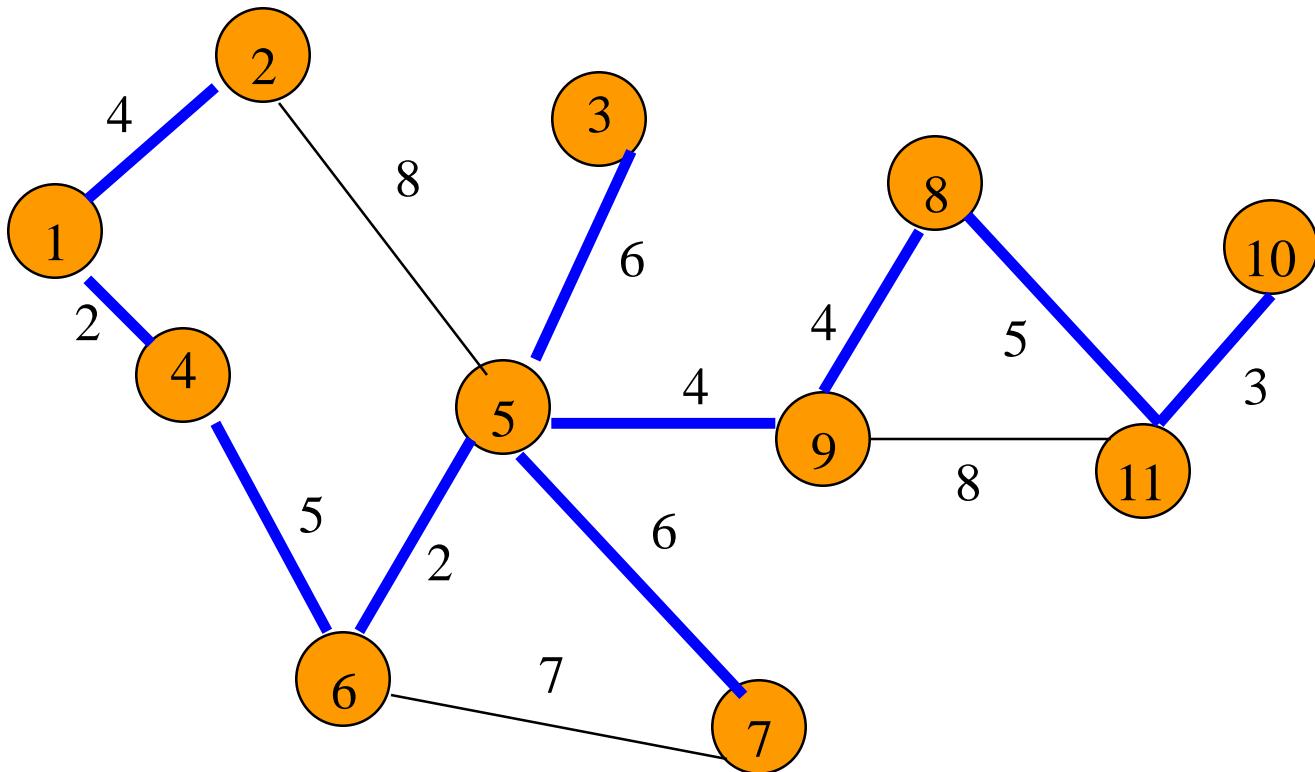
- Модны өртөг/зардал нь ирмэгүүдийн жин/өртгийн нийлбэр байна.

# Бүрхэгч мөд



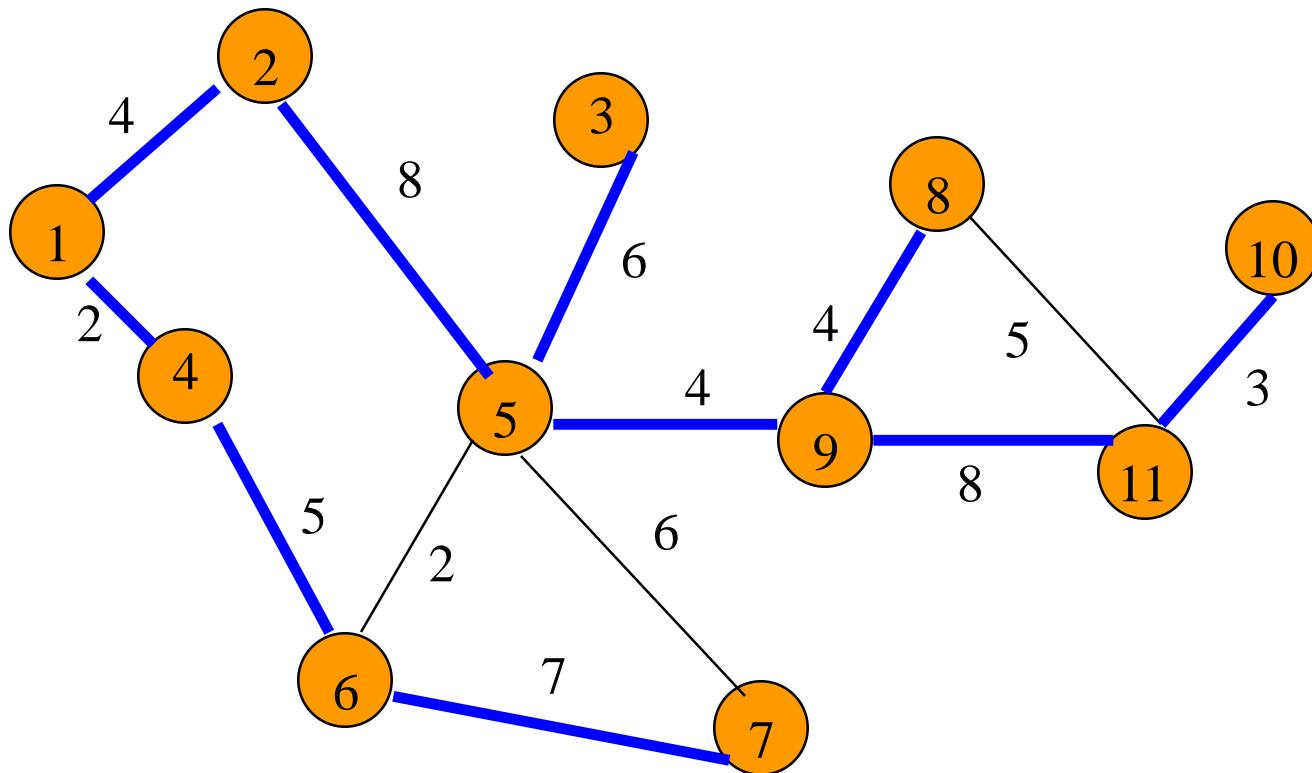
Бүрхэгч мөдны өртөг = 51.

# Min өртгийн Бурхэгч мод



Бурхэгч модны өртөг = 41.

# Утасгүй Дамжуулалтын мод



Төв = 1, жин = шаардлагатай чадал.

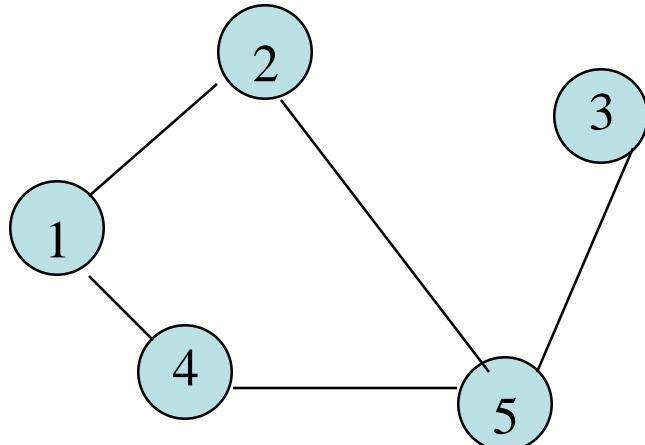
Зардал =  $4 + 8 + 5 + 6 + 7 + 8 + 3 = 41$ .

# Графын дүрслэл

- Холболтын матриц(Adjacency matrix)
- Холболтын жагсаалт
  - Холбоост холболтын жагсаалт
  - Массивт холболтын жагсаалт

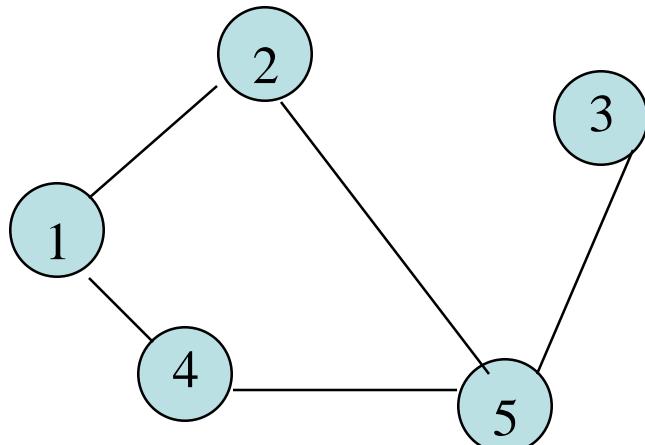
# Холболтын матриц

- 0/1  $n \times n$  матриц, үүнд  $n$  = оройн тоо
- $A(i,j) = 1$  хэрвээ  $(i,j)$  гэсэн ирмэгтэй бол



|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 1 | 1 | 0 |

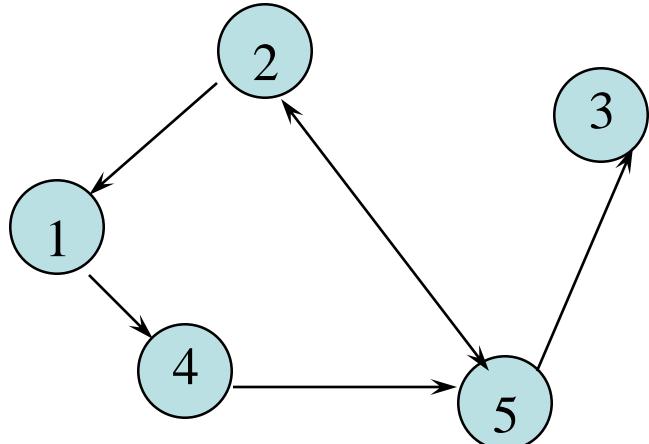
# Холболтын матрицын шинж



|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 1 | 1 | 0 |

- Диагональ 0.
- Чиглэлгүй графын холболтын матриц симметр.
  - $A(i,j) = A(j,i)$  бүх  $i, j$ -ийн хувьд

# Холболтын матриц (Digraph)



|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 1 | 0 | 0 |

- Диагналь 0.
- Чиглэлтэй графын холболтын матриц симметр бус.

# Холболтын матриц

- $n^2$  бит орон зай
- Чиглэлгүй графын хувьд, дээд, доод гурвалжны аль нэг (диагональ орохгүй).
  - $(n-1)n/2$  бит
- $O(n)$  - оройн зэрэг, өгөгдсөн оройтой хөршлөх оройнуудыг олоход.

# Холболтын жагсаалт

- *i* оройн холболтын жагсаалт нь тухайн оройтой хөршлөх оройнуудын шугаман жагсаалт байна.
- *n* холболтын жагсаалтын массив.

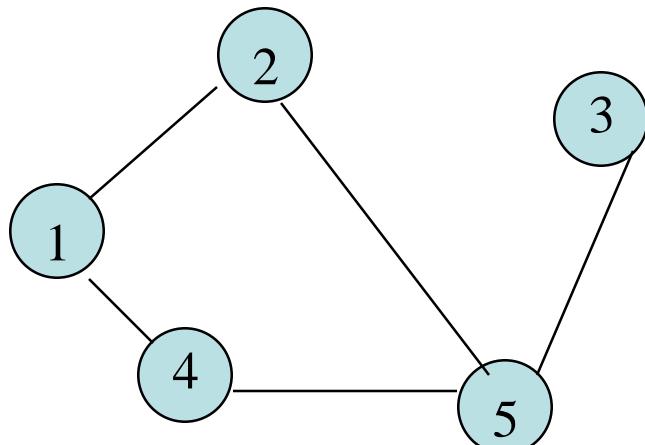
$$aList[1] = (2,4)$$

$$aList[2] = (1,5)$$

$$aList[3] = (5)$$

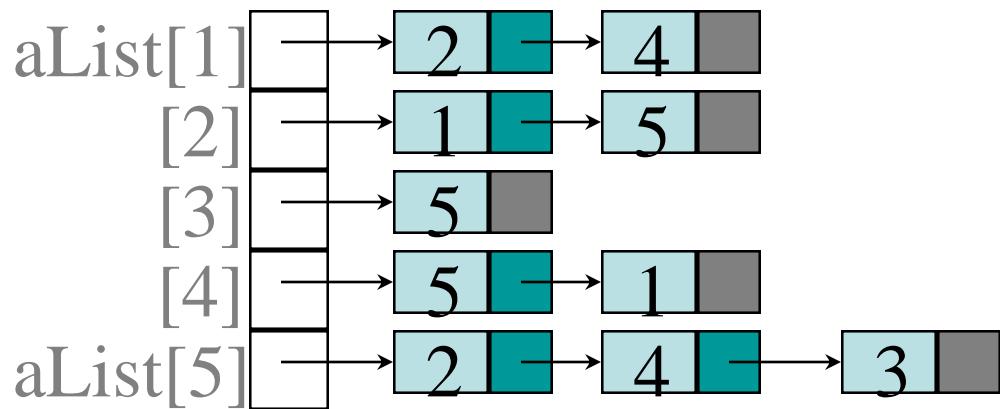
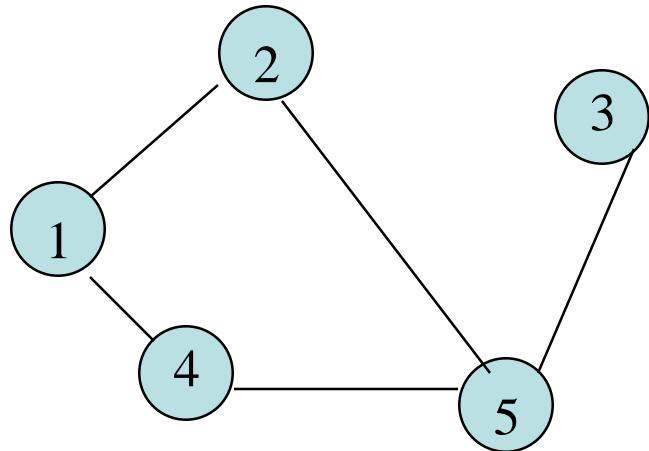
$$aList[4] = (5,1)$$

$$aList[5] = (2,4,3)$$



# Холбоост холболтын жагсаалт

- Холболтын жагсаалт бүр гинж болно.



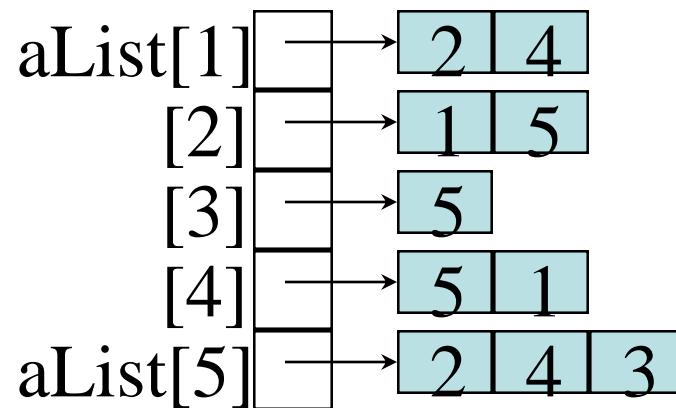
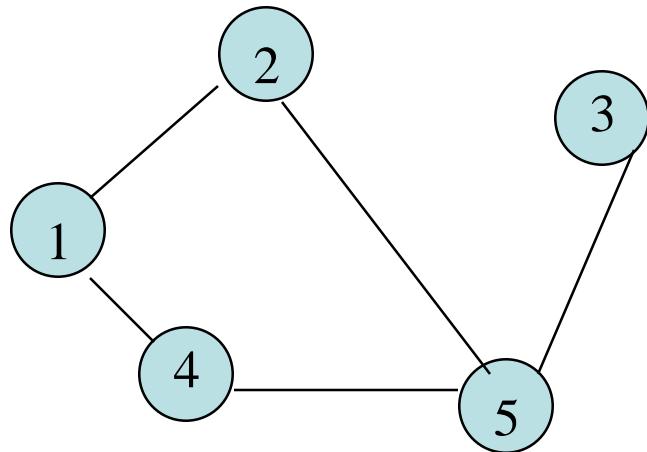
Массивын урт =  $n$

Гинжин зангилааны тоо =  $2e$  (чиглэлгүй граф)

Гинжин зангилааны тоо =  $e$  (digraph)

# Массивт холболтын жагсаалт

- Холболтын жагсаалт бүр массив.



Массивын урт = n

Жагсаалтын элементийн тоо = 2e (чиглэлгүй граф)

Жагсаалтын элементийн тоо = e (digraph)

# Жин/ачаатай граф

- Өртөгт холболтын матриц.
  - $C(i,j) = (i,j)$  ирмэгийн өртөг
- Холболтын жагсаалт => жагсаалтын элемент бүр хос (хөрш орой, ирмэгийн жин)

# Шаардлагатай Java класс

- Графын дүрслэл
  - Холболтын матриц
  - Холболтын жагсаалт
    - Холбоост холболтын жагсаалт
    - Массивт холболтын жагсаалт
  - $3$  дүрслэл
- Графын төрөл
  - Чиглэлтэй, чиглэлгүй.
  - Жинтэй, жингүй.
  - $2 \times 2 = 4$  графын төрөл
- $3 \times 4 = 12$  Java класс

# Хийсвэр класс Graph

```
package dataStructures;  
import java.util.*;  
public abstract class Graph  
{  
    // Хийсвэр өгөгдлийн төрлийн аргууд энд бичигдэнэ  
  
    // i оройн iterator арга  
    public abstract Iterator iterator(int i);  
  
    // хэрэгжүүлэлтээс хамааралгүй аргуудын хэрэгжүүлэлт  
    // энд бичигдэнэ  
}
```

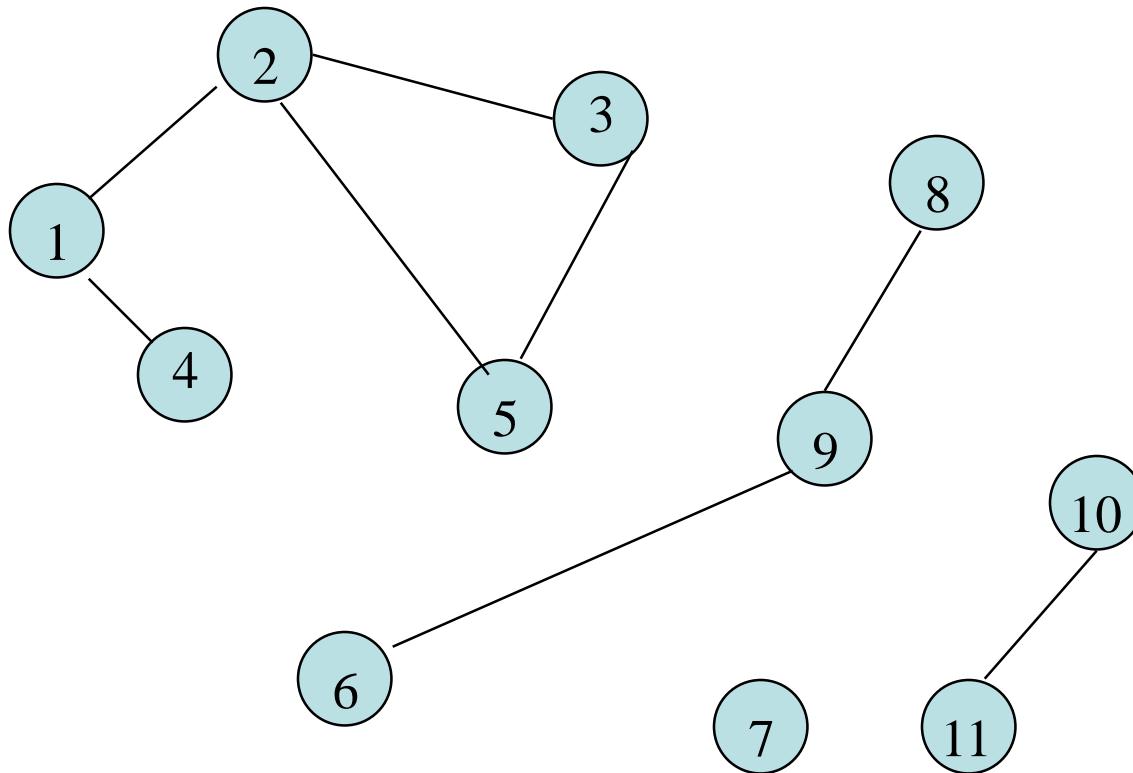
# Graph –н хийсвэр аргууд

// ADT methods

```
public abstract int vertices();  
public abstract int edges();  
public abstract boolean existsEdge(int i, int j);  
public abstract void putEdge(Object theEdge);  
public abstract void removeEdge(int i, int j);  
public abstract int degree(int i);  
public abstract int inDegree(int i);  
public abstract int outDegree(int i);
```

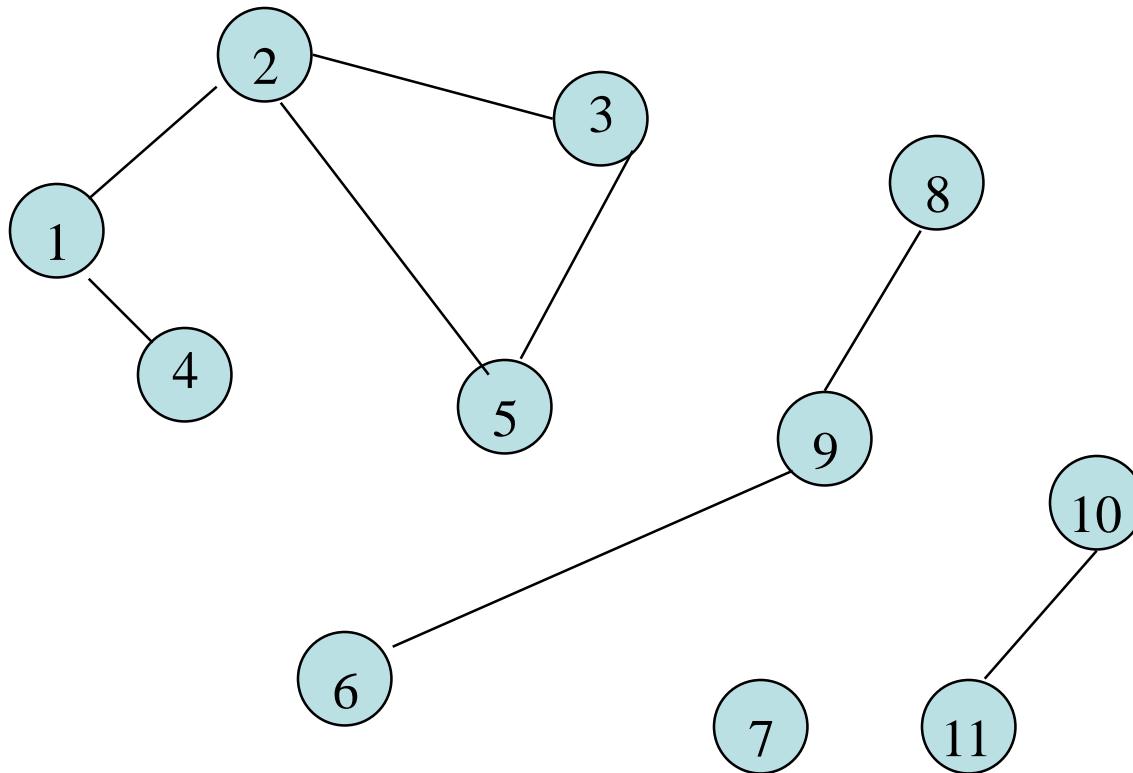
# Графын хайлтын аргууд

- Хэрвээ  $v$  -с  $u$  –н хооронд зам байгаа бол  $v$  оройгоос  $u$  оройд **хүрч болно**



# Графын хайлтын аргууд

- Хайлтын арга өгөгдсөн  $v$  оройгоос эхэлж  $v$  оройгоос хүрч болох бүх оройгоор зочлож/хаяглаж/тэмдэглэж хайна.



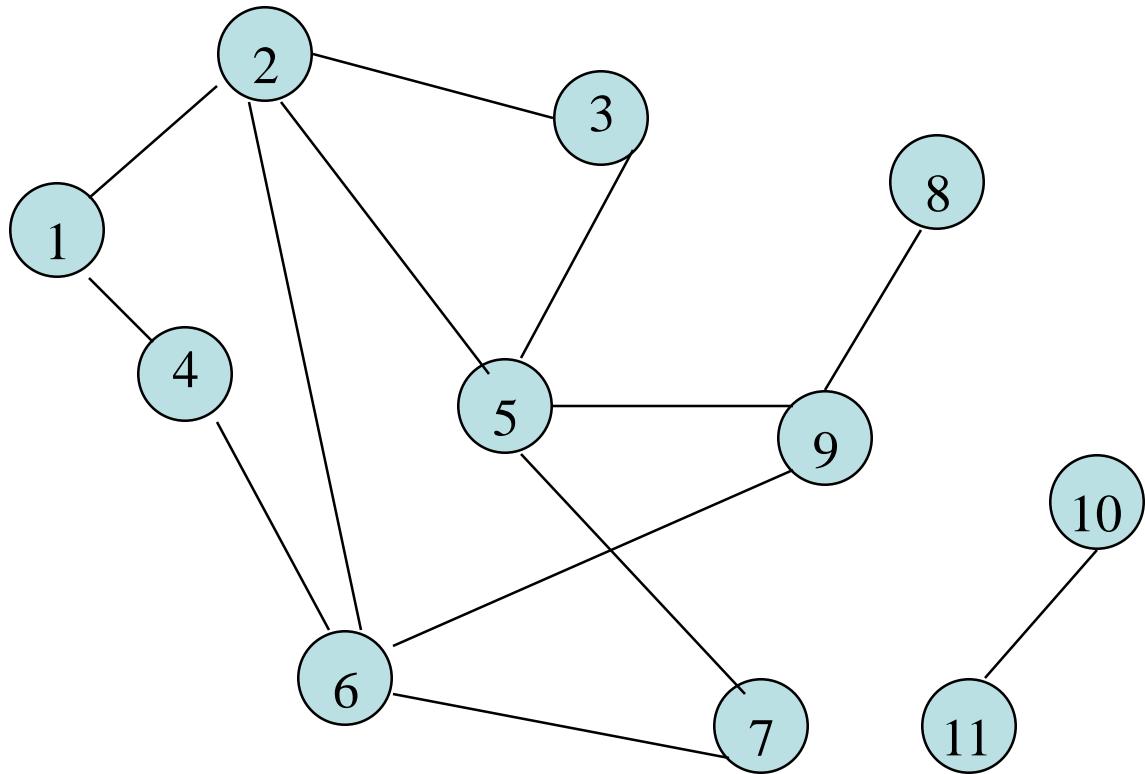
# Графын хайлтын аргууд

- Ихэнх графын бодлогуудыг хайлтын аргаар шийдвэрлэдэг.
  - Нэг оройгоос нөгөө оройд хүрэх зам.
  - Граф холбогдсон эсэх?
  - Бүрхсэн модыг олох.
  - Г.М.
- Түгээмэл ашигладаг хайлтын аргууд:
  - Түвшнээр-Эхэлж хайх.(Depth-First-Search) түвшний хайлт
  - Гүнээр-Эхэлж хайх.(Breadth-First-Search) гүний хайлт

# Түвшний хайлт (Breadth-First Search)

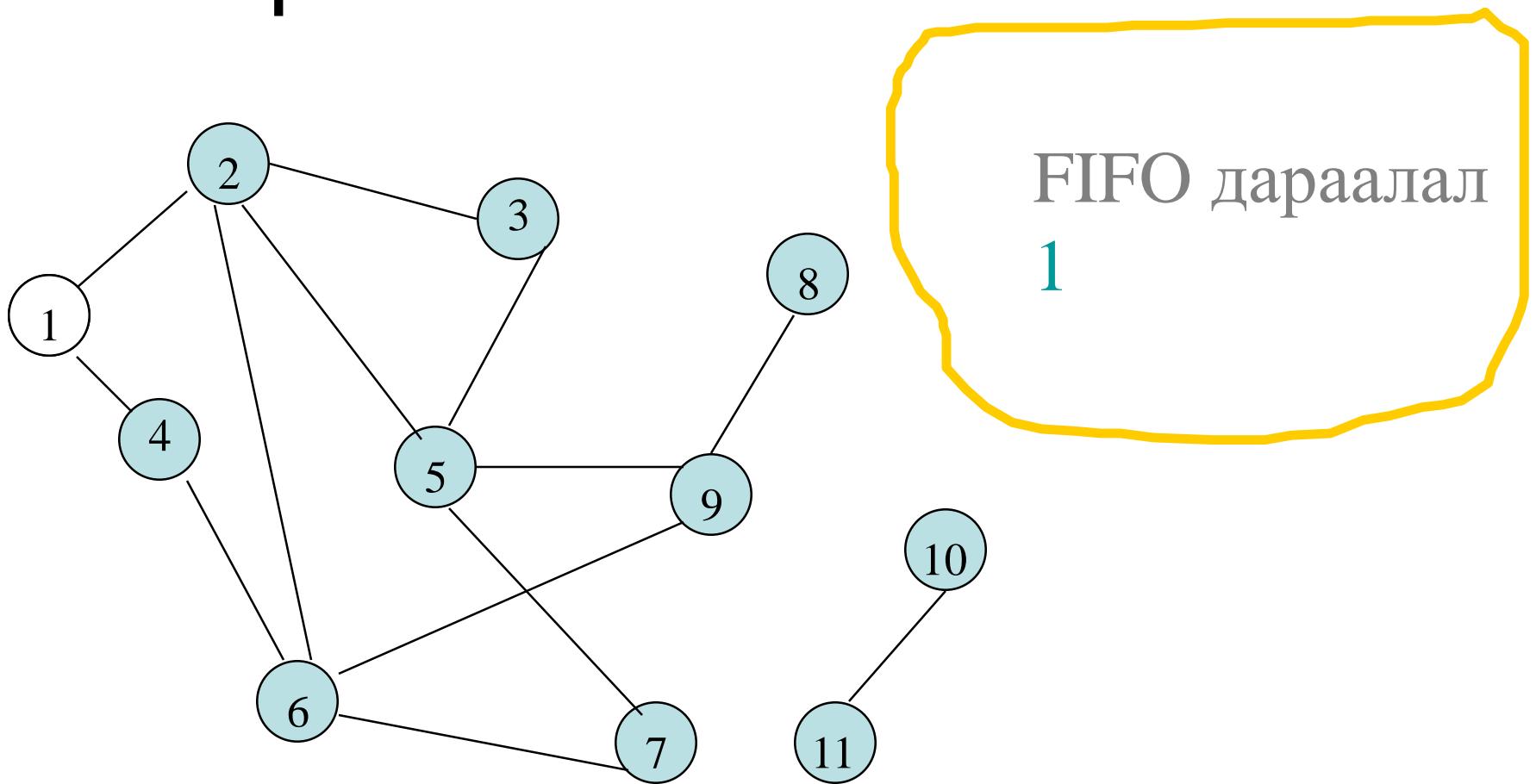
- Энхний оройд зочлоод түүнийг FIFO дараалалд хийнэ.
- Дараалаас оройг устгаж, уг оройн зочлоогүй хөрш оройнуудаар зочлох, шинэ зочилсон оройнуудаа дараалалд нэмэх үйлдлийг давтан гүйцэтгэнэ.

# Түвшний хайлтын жишээ



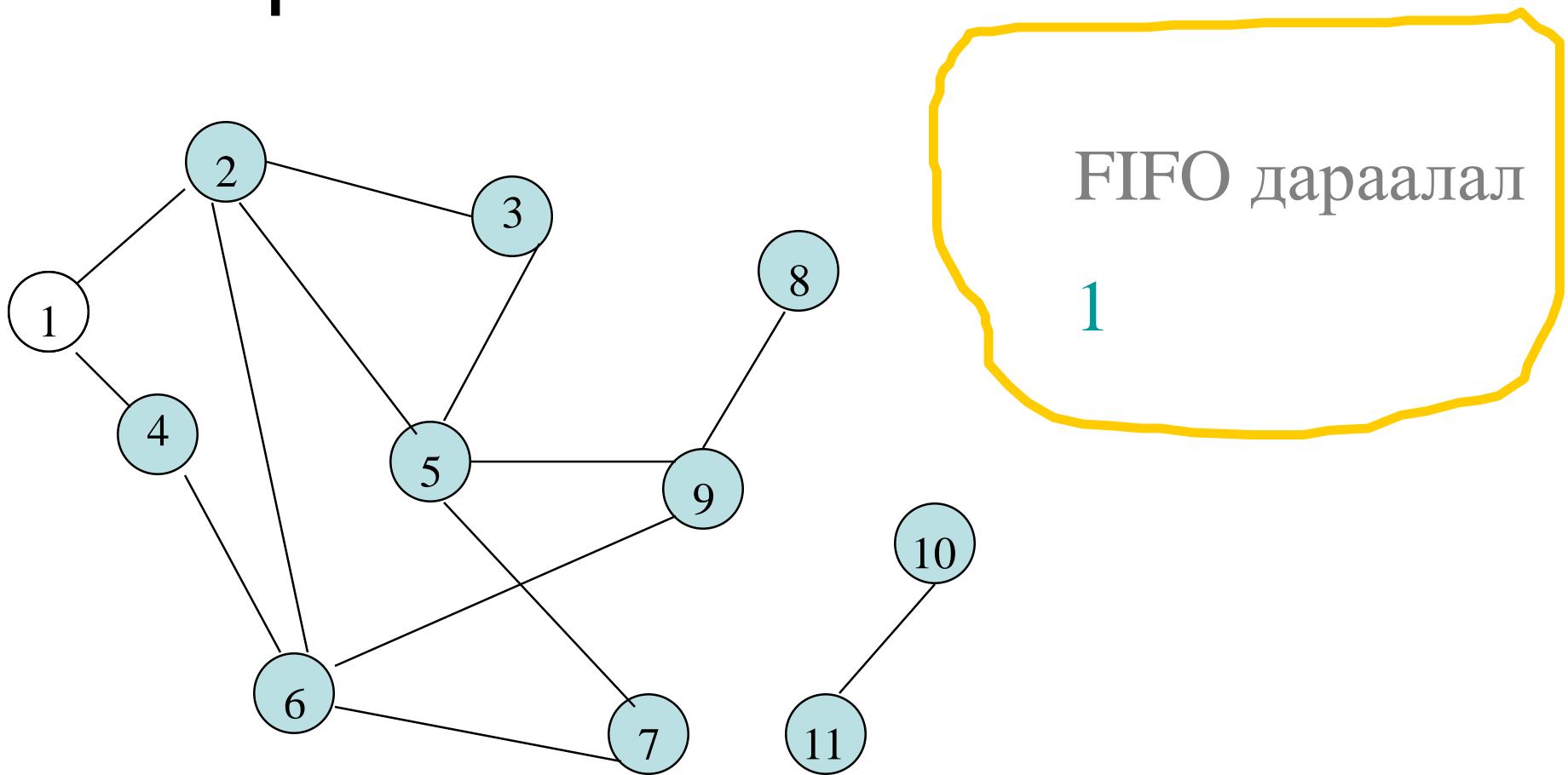
Хайлт эхлэх орой - 1.

# Түвшний хайлтын жишээ



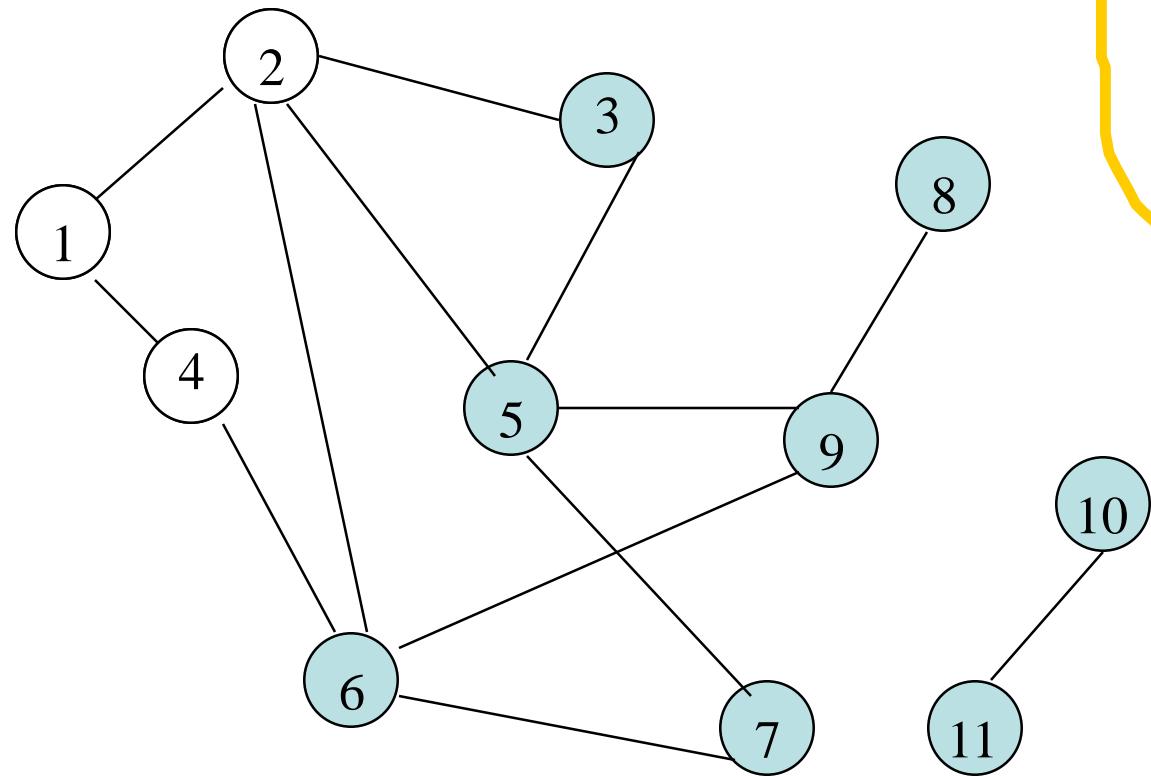
Эхний оройд зочилж/тэмдэглэж/хаяглаад, FIFO дараалалд хийнэ.

# Түвшний хайлтын жишээ



Q -с 1 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

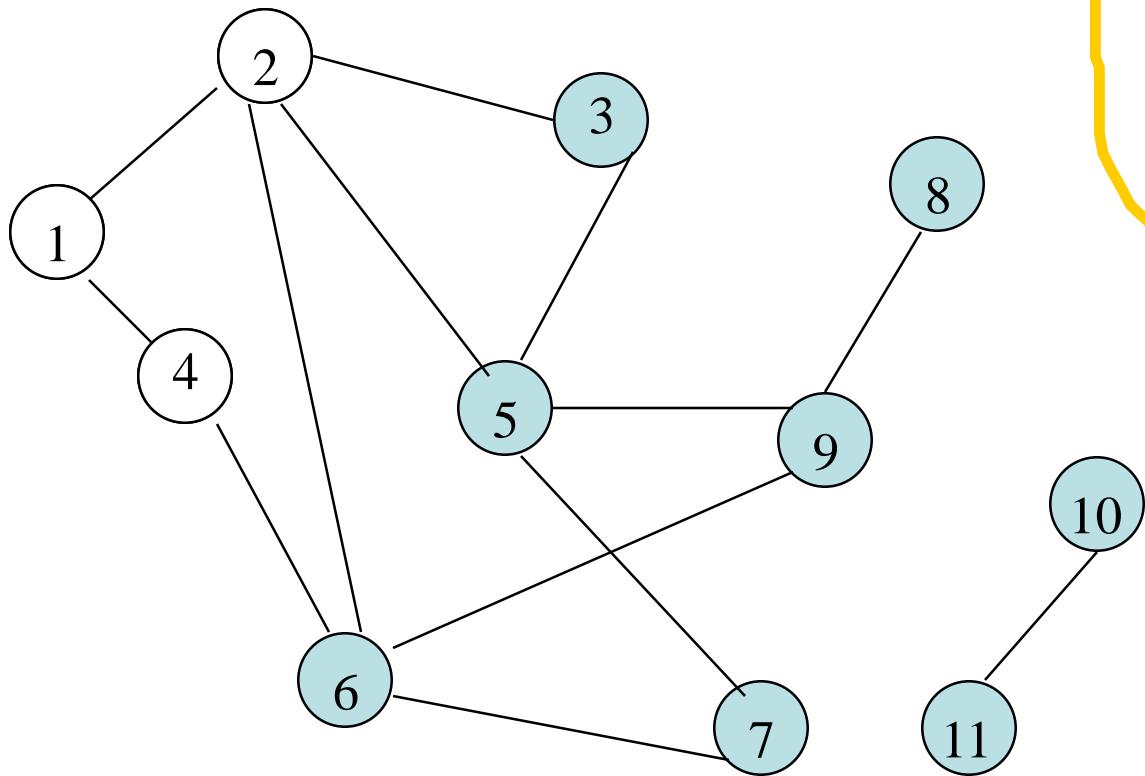
# Түвшний хайлтын жишээ



FIFO дараалал  
2 4

Q -с 1 –г устгаж, айлчлаагүй хөршүүдээр нь айлчилж  
Q -д хийнэ.

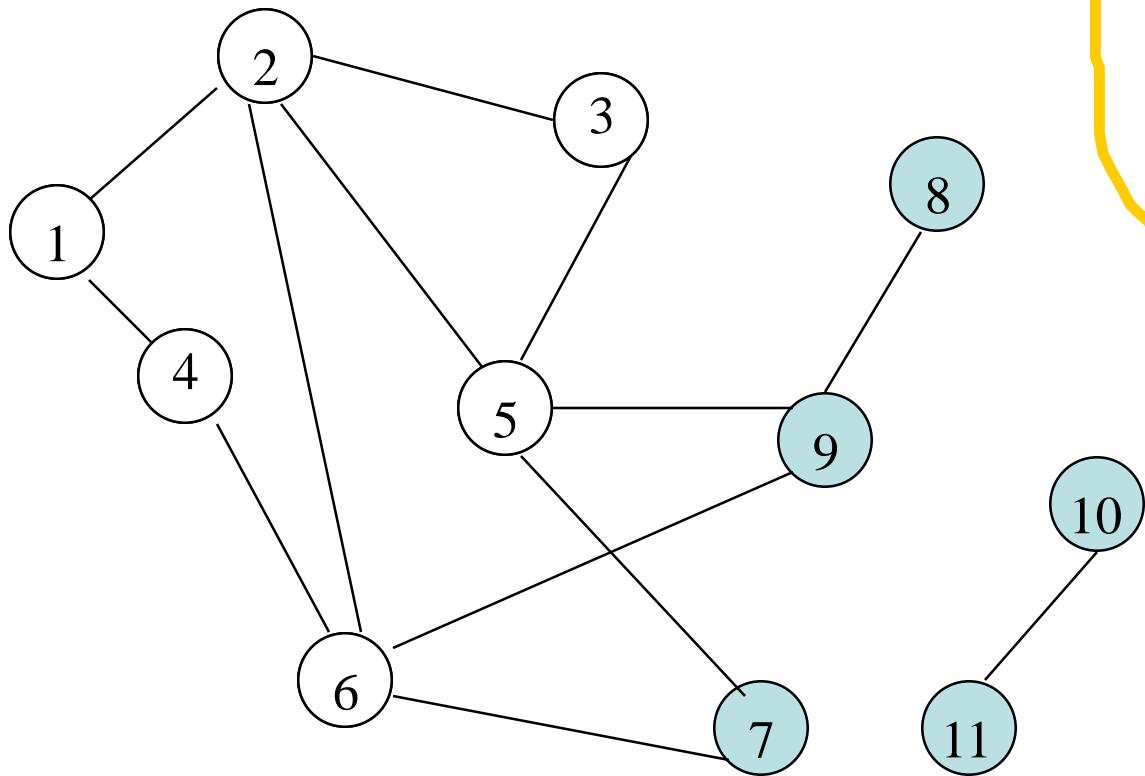
# Түвшний хайлтын жишээ



FIFO дараалал  
2 4

Q -с 2 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

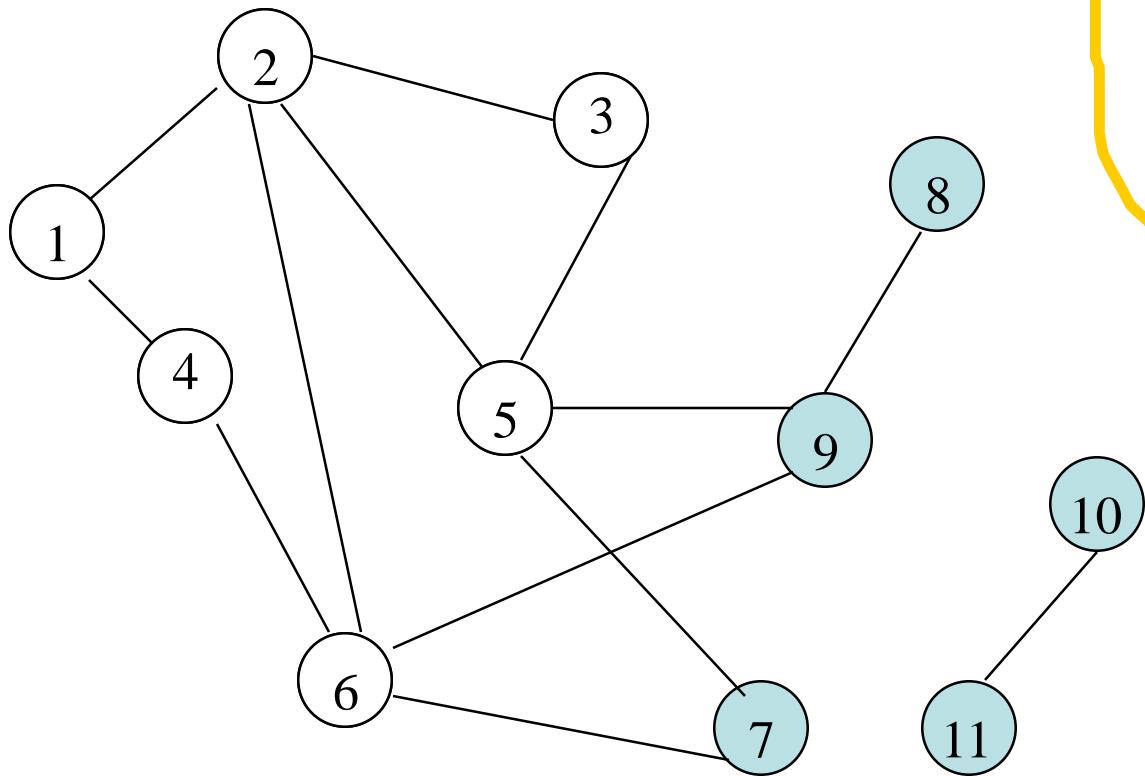


FIFO дараалал

4 5 3 6

Q -с 2 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

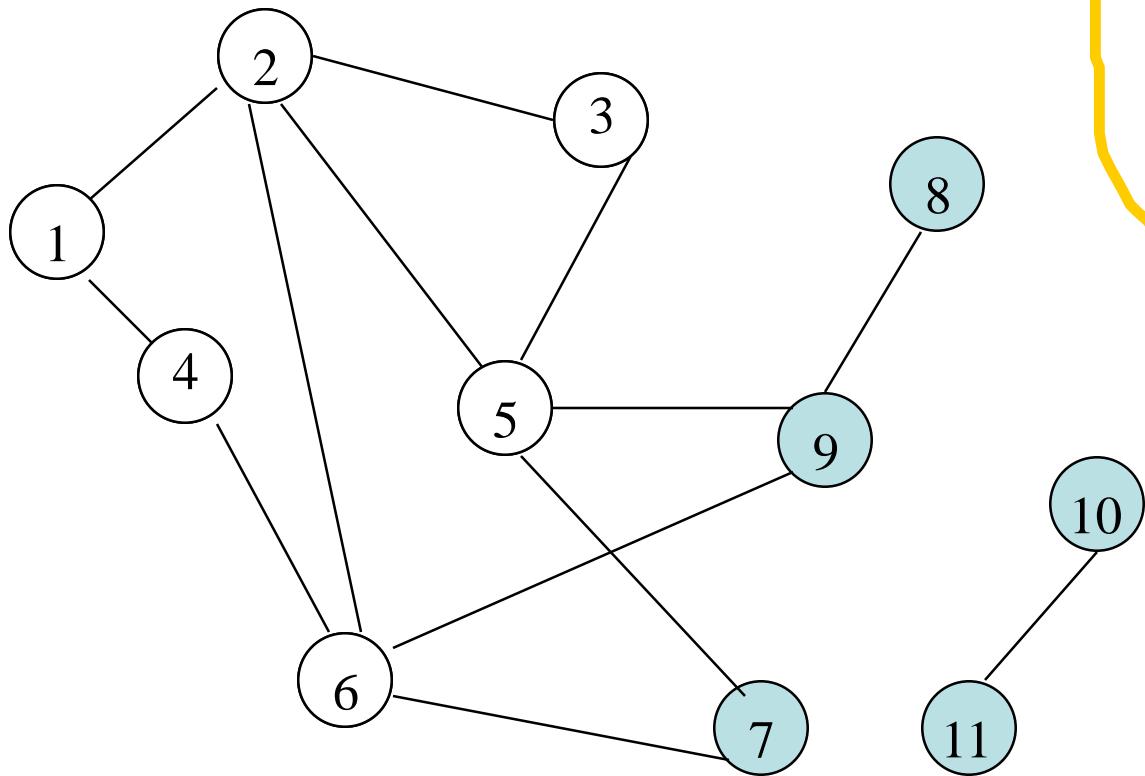


FIFO дараалал

4 5 3 6

- Q -с 4 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

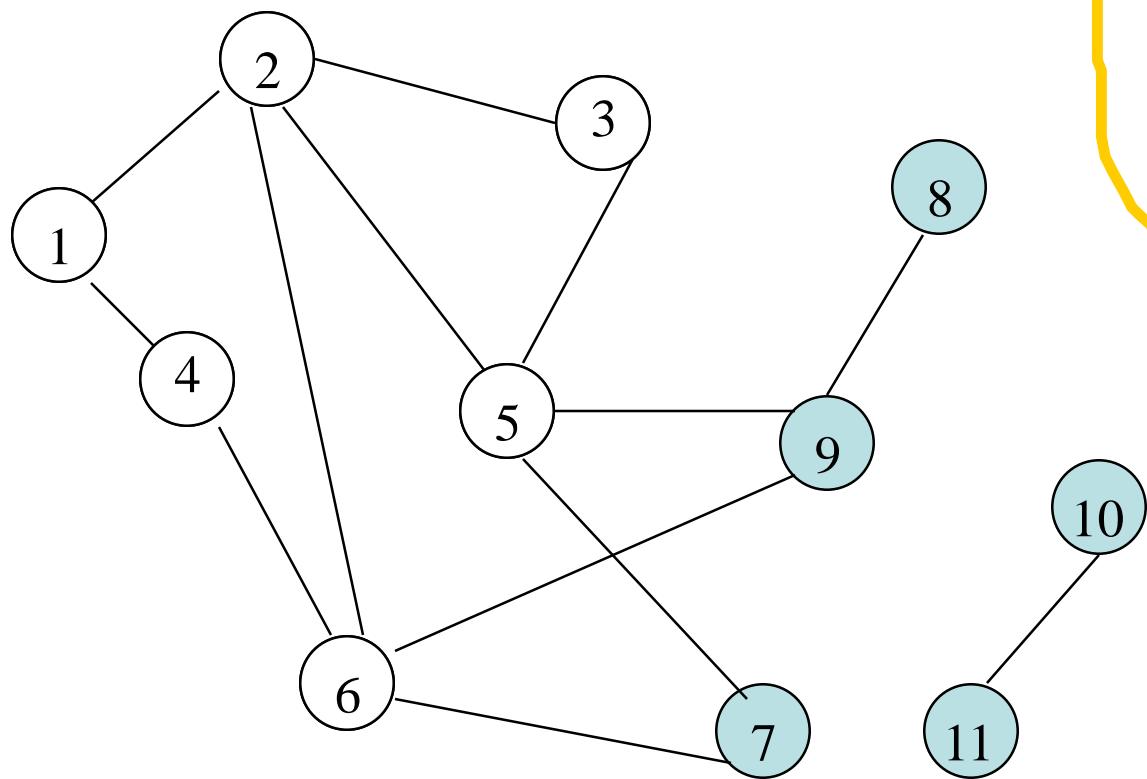


FIFO дараалал

5 3 6

Q -с 4 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

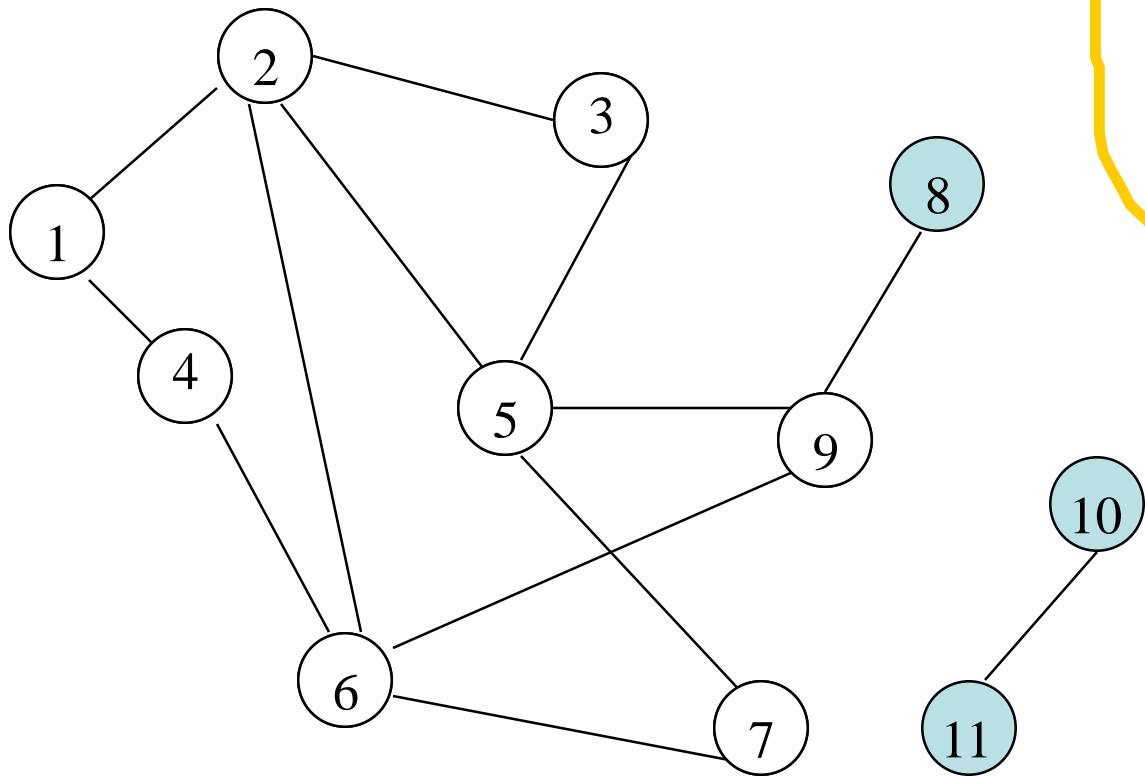


FIFO дараалал

5 3 6

Q -с 5 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

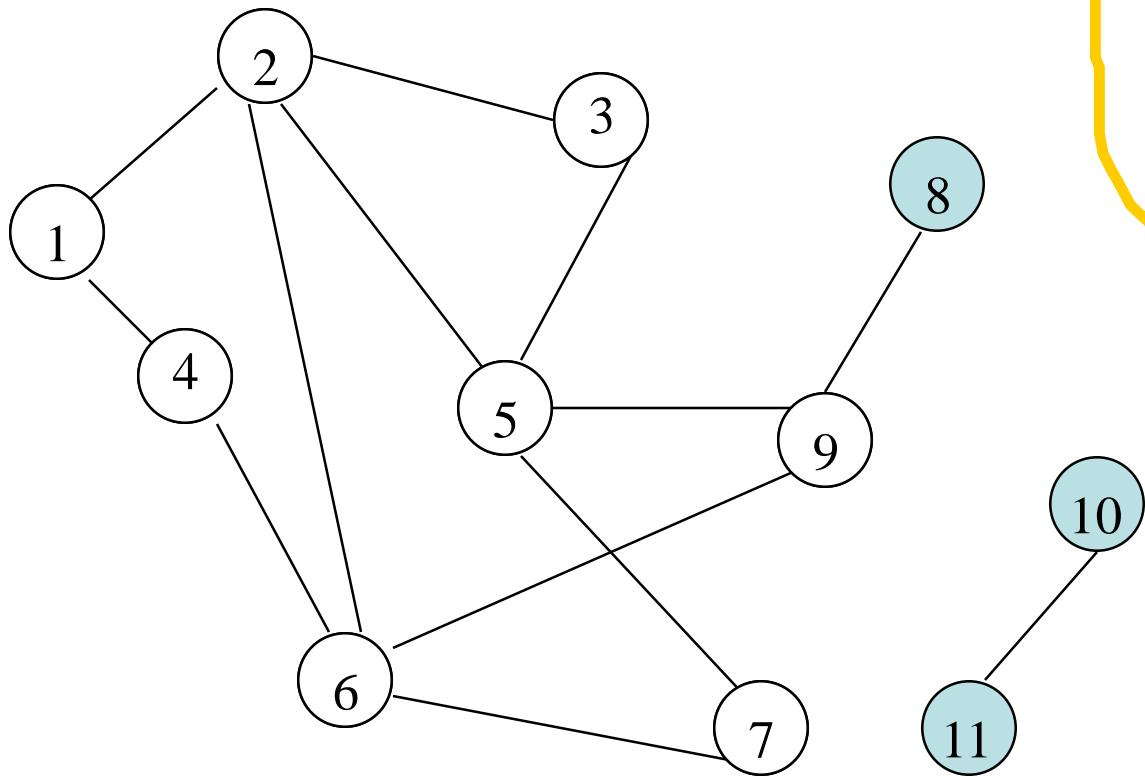


FIFO дараалал

3 6 9 7

Q -с 5 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

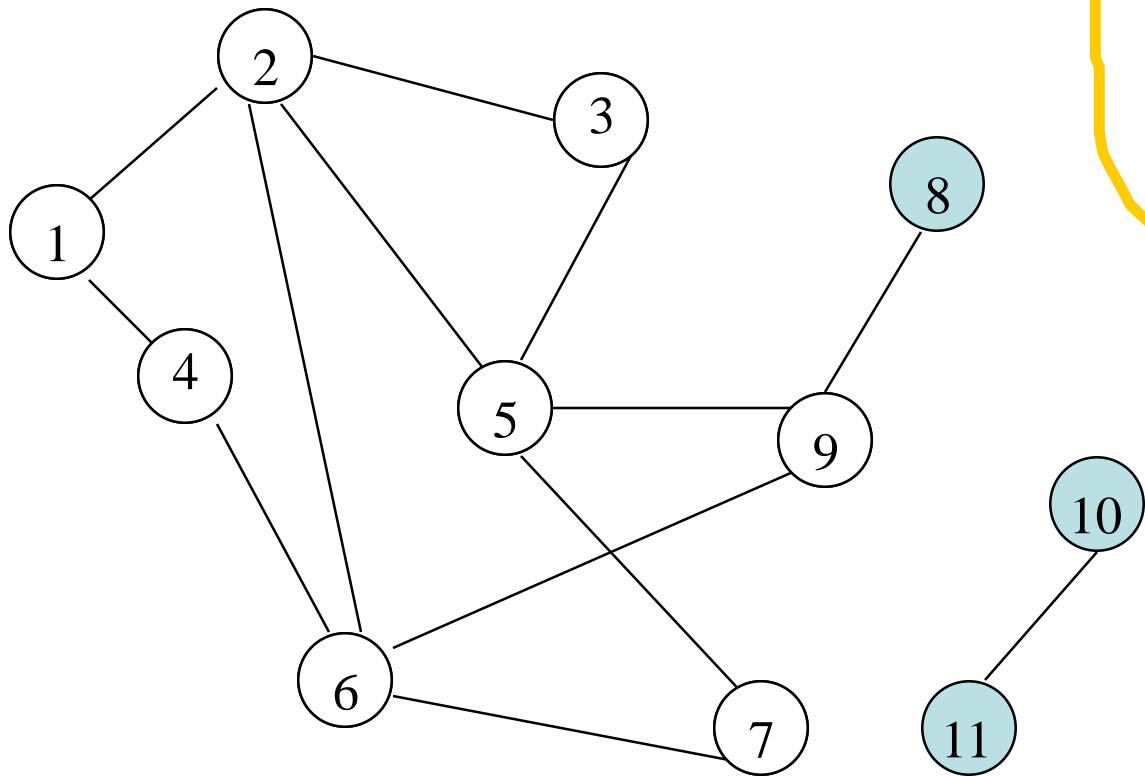


FIFO дараалал

3 6 9 7

Q -с 3 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

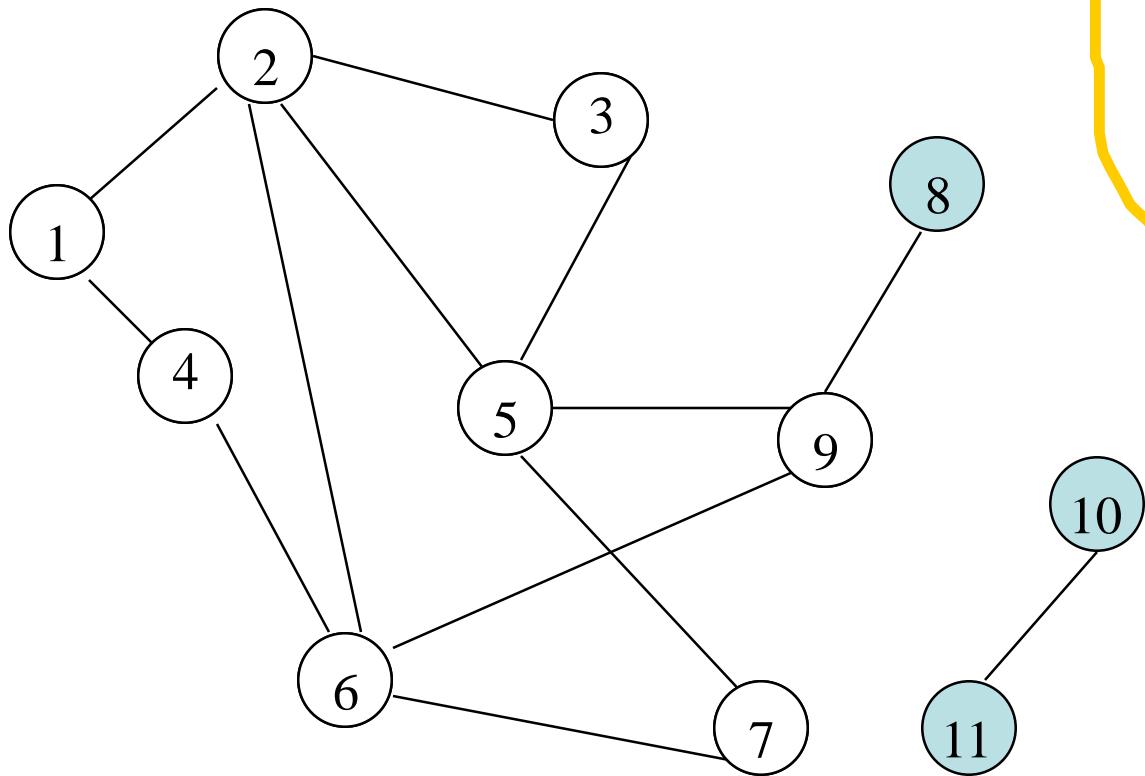


FIFO дараалал

6 9 7

Q -с 3 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

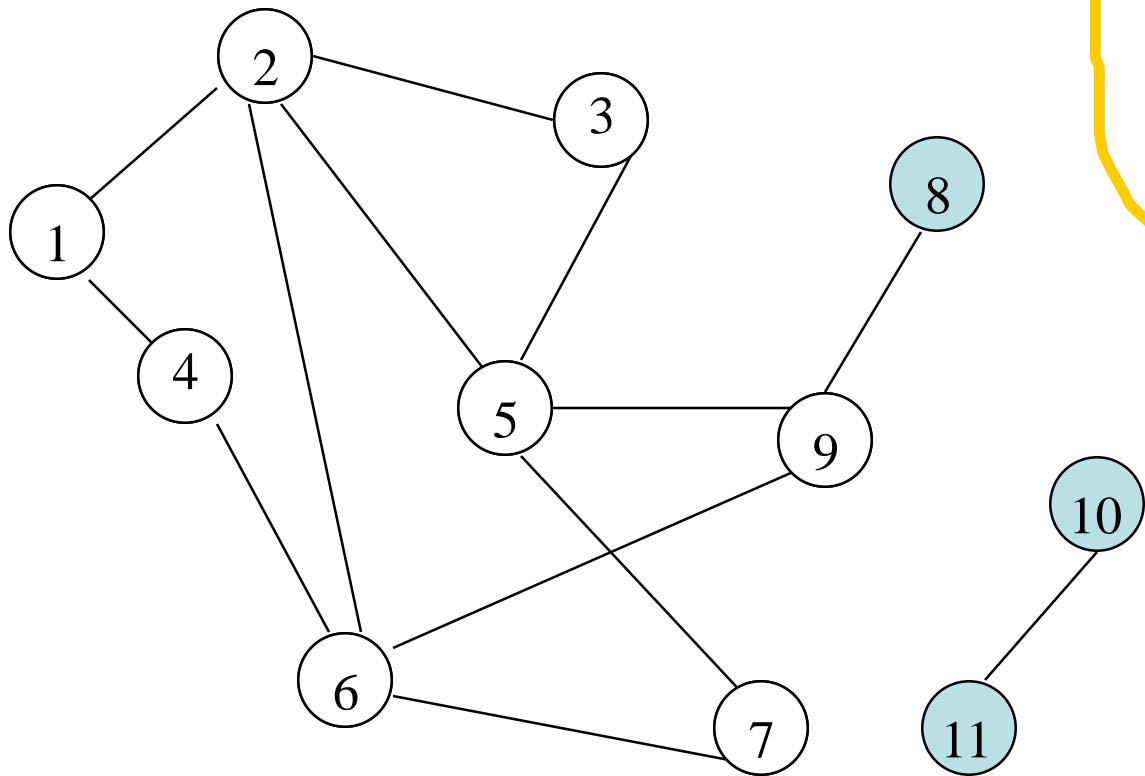


FIFO дараалал

6 9 7

Q -с 6 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

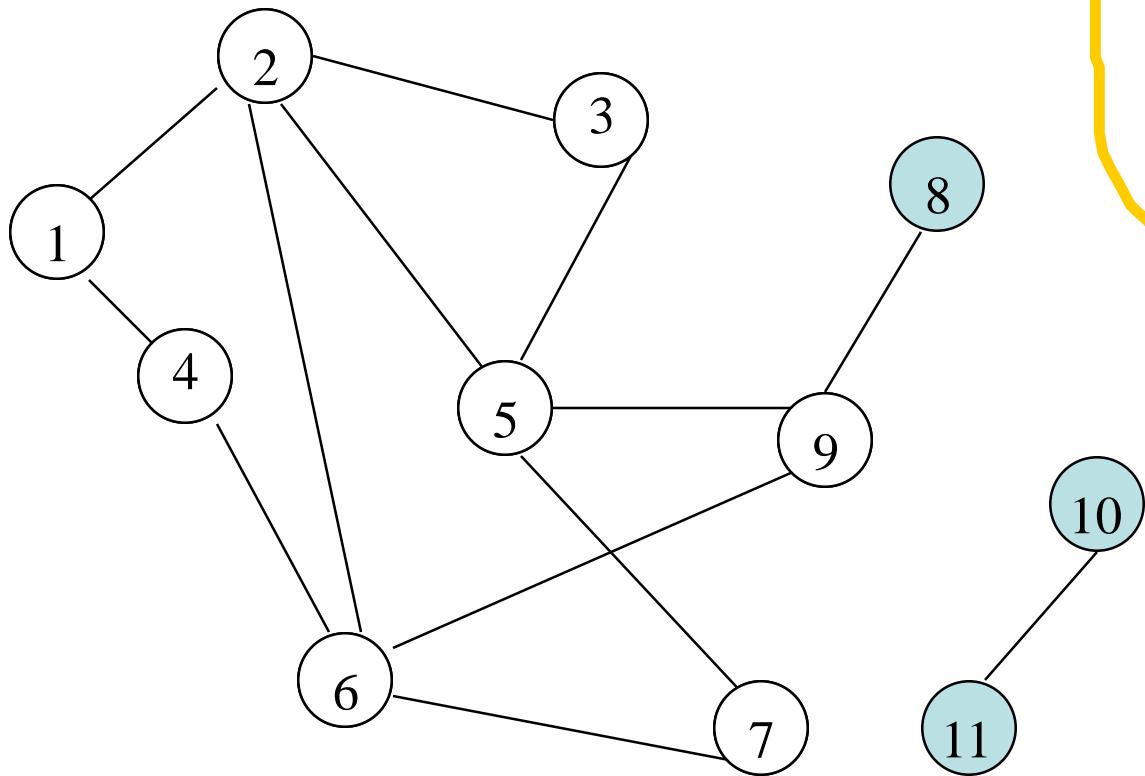


FIFO дараалал

9 7

Q -с 6 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

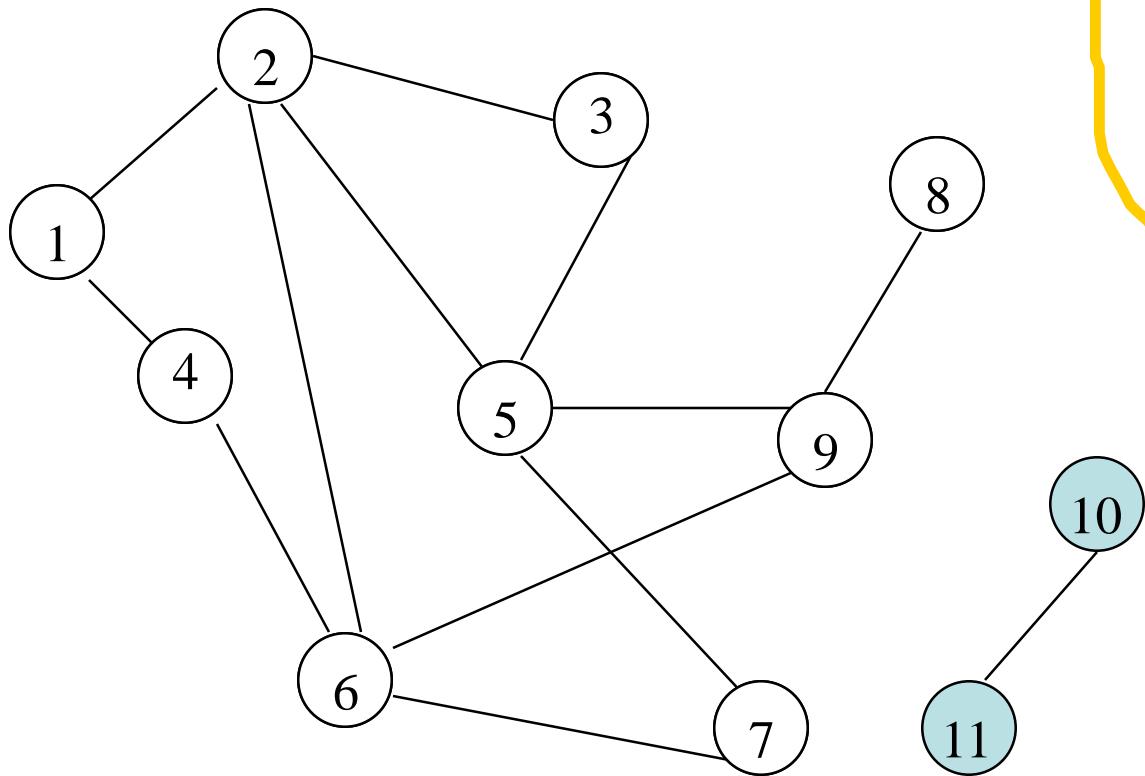


FIFO дараалал

9 7

Q -с 9 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

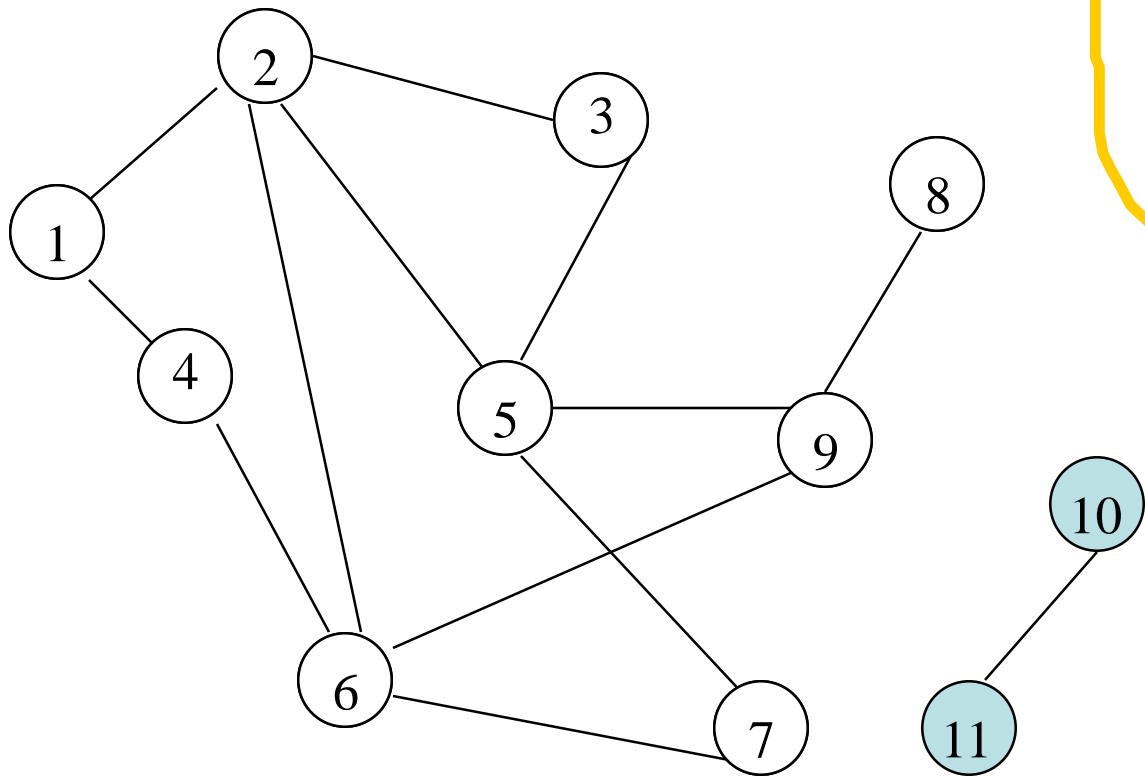


FIFO дараалал

7 8

Q -с 9 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

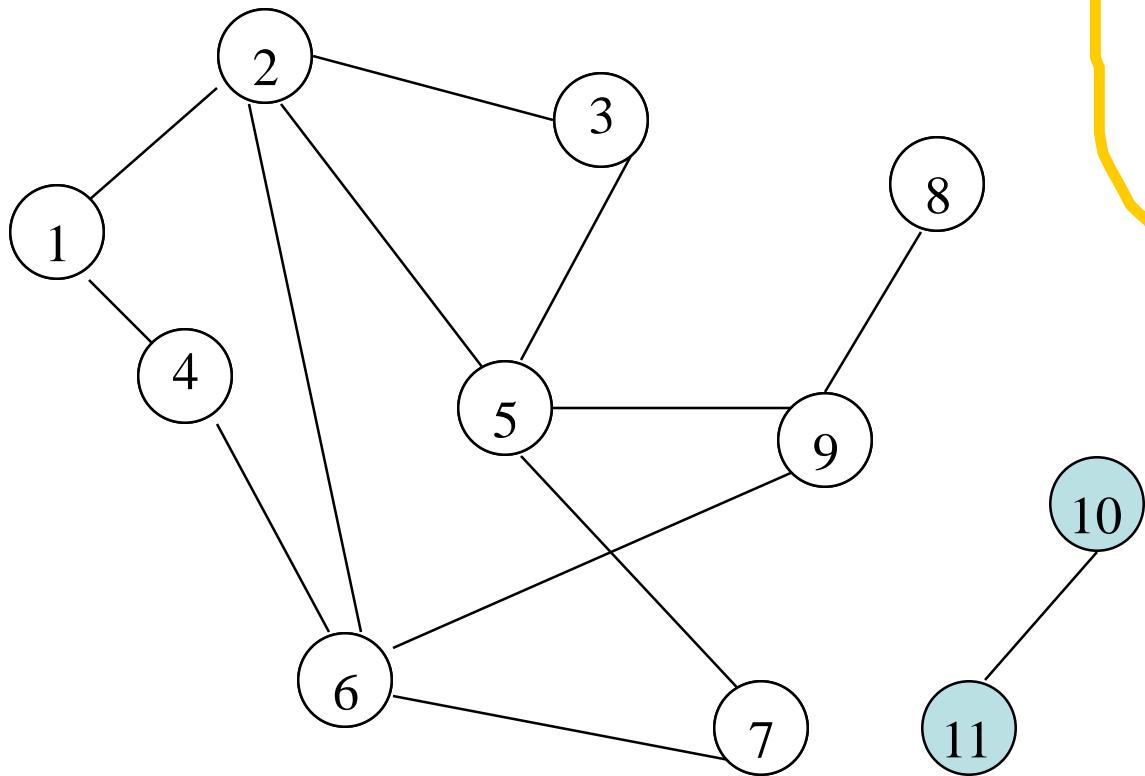


FIFO дараалал

7 8

Q -с 7 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

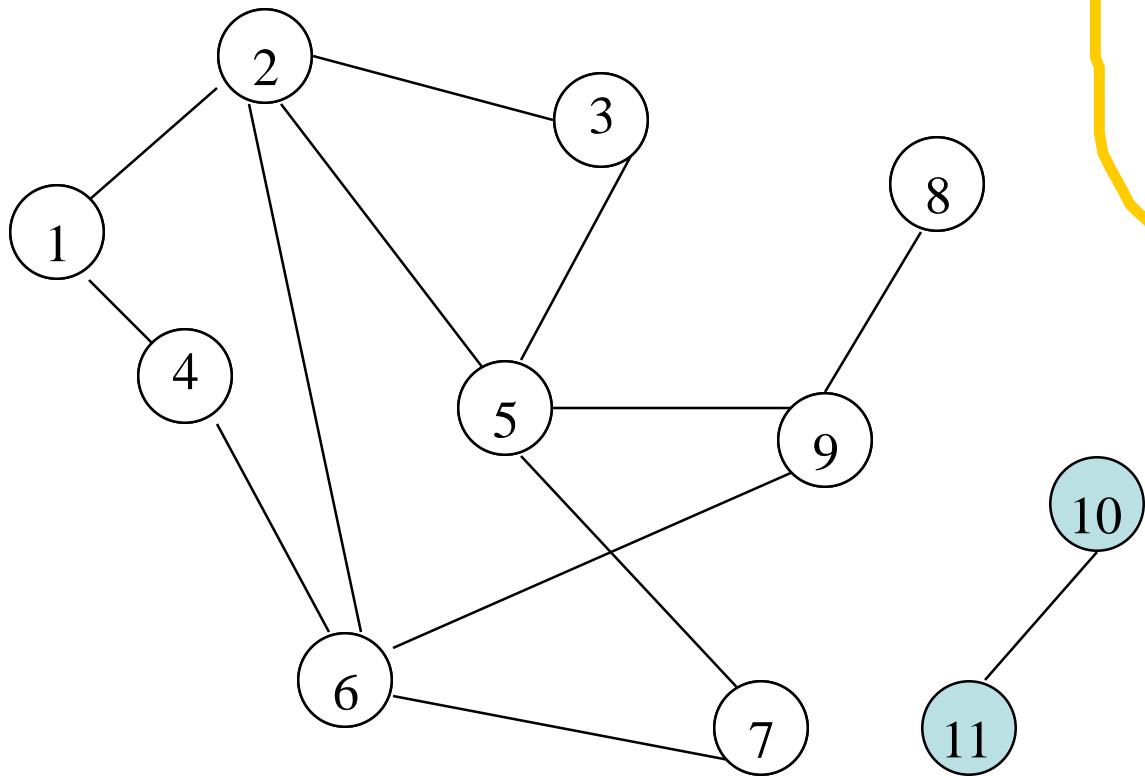


FIFO дараалал

8

Q -с 7 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ

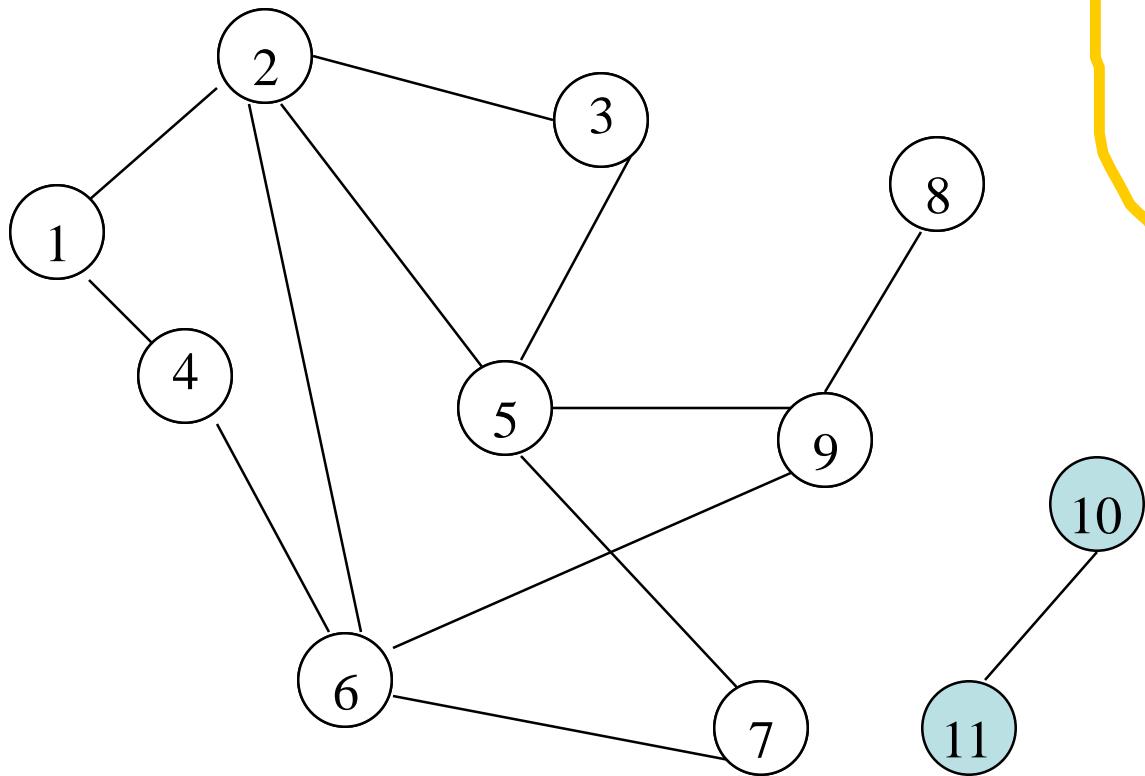


FIFO дараалал

8

Q -с 8 -г устгаж, айлчлаагүй хөршүүдээр нь айлчилж;  
Q -д хийнэ.

# Түвшний хайлтын жишээ



FIFO дараалал

Дараалал хоосон. Хайлт төгслөө.

# Түвшний хайлтын шинж

- Эхний оройгоос хүрч болох бүх оройгоор (эхний оройг оролцуулаад) зочилдог.



# Хугацаа

- Зочилсон орой бүрдараалалд яг нэг удаа орно (мөн тэндээс устгагдана) .
- Дарааллаас оройг устгахдаа, бид хөрш оройнуудыг нь шалгана.
  - $O(n)$  холболтын матриц ашиглахад
  - $O(\text{оройн зэрэг})$  холболтын жагсаалт ашиглахад
- Нийт хугацаа
  - $O(mn)$ , үүнд  $m$  –хайлт хийж байгаа бүрдүүлбэр дэх оройн тоо (холболтын матриц)



# Хугацаа

- $O(n + \text{бүрдүүлбэрийн оройн зэргийн нийлбэр})$  (холболтын жагсаалт)  
 $= O(n + \text{бүрдүүлбэрийн ирмэгийн тоо})$

# Орой $v$ –с орой и хүрэх зам

- Түвшний хайлтыг  $v$  оройгоос эхэлнэ
- Орой  $u$  –р зочилсон, эсхүл  $Q$  хоосорсон тохиолдолд зогсоно (Алин ч эхэлж тохиолдож болно).
- Хугацаа
  - $O(n^2)$  холболтын матриц ашиглахад
  - $O(n+e)$  холболтын жагсаалт ашиглахад ( $e$  –ирмэгийн тоо)

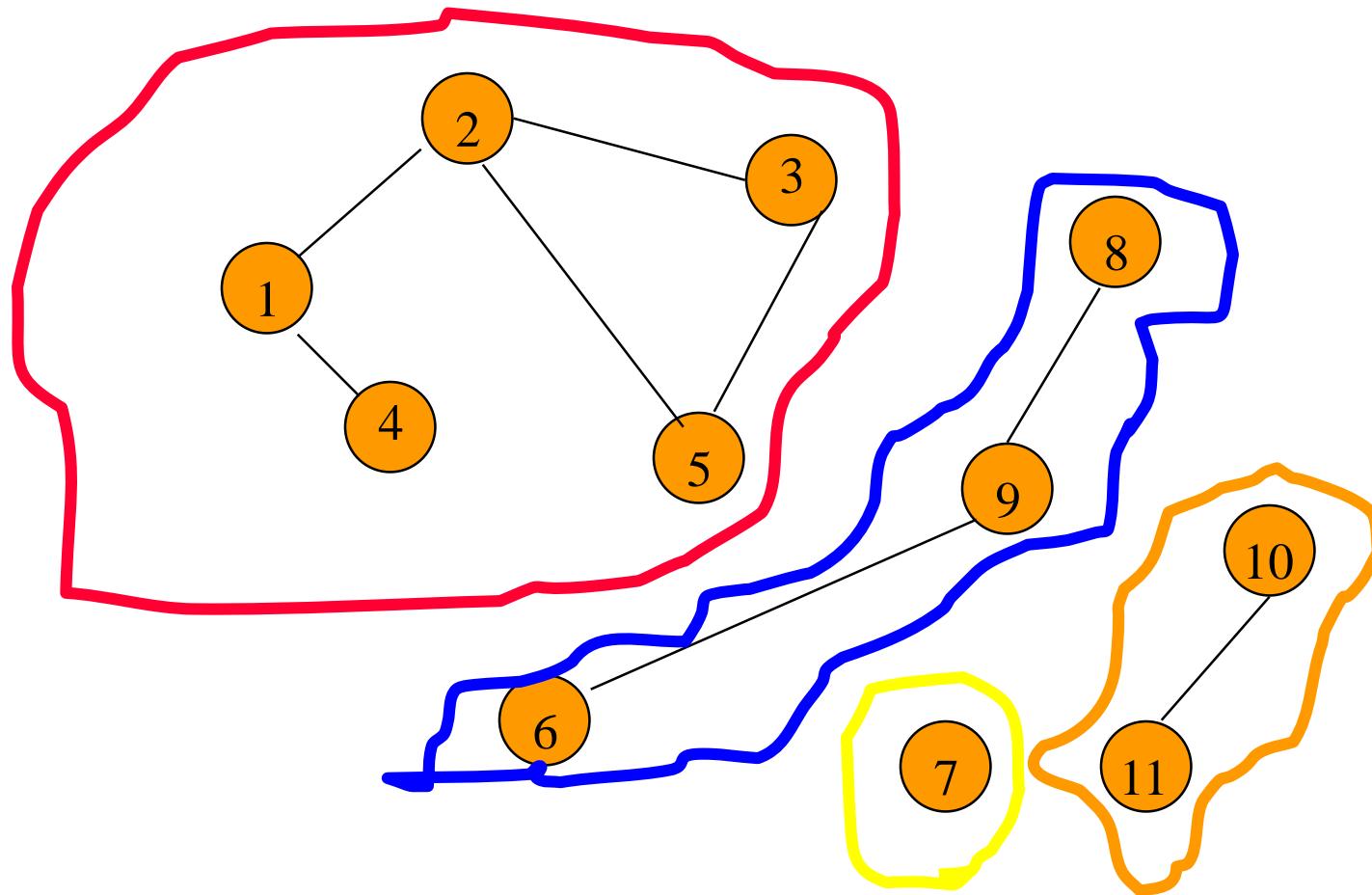
# Граф холбогдсон уу?

- Графын дурын оройгоос Түвшний хайлтыг эхэлнэ.
- Бүх  $n$  оройгоор зочилсон бол граф холбогдсон.
- Хугацаа
  - $O(n^2)$  холболтын матриц ашиглахад
  - $O(n+e)$  холболтын жагсаалт ашиглахад ( $e$  –ирмэгийн тоо)

# Холбогдсон бүрдүүлбэр

- Графын дурын зочлоогүй оройгоос Түвшнгий хайлт эхэлнэ.
- Шинээр айлчилсан оройнууд (тэдний хоорондох ирмэгүүдийг нэмээд) бүрдүүлбэрийг тодорхойлно.
- Бүх оройгоор зочилтол давтана.

# Холбогдсон бурдүүлбэрүүд

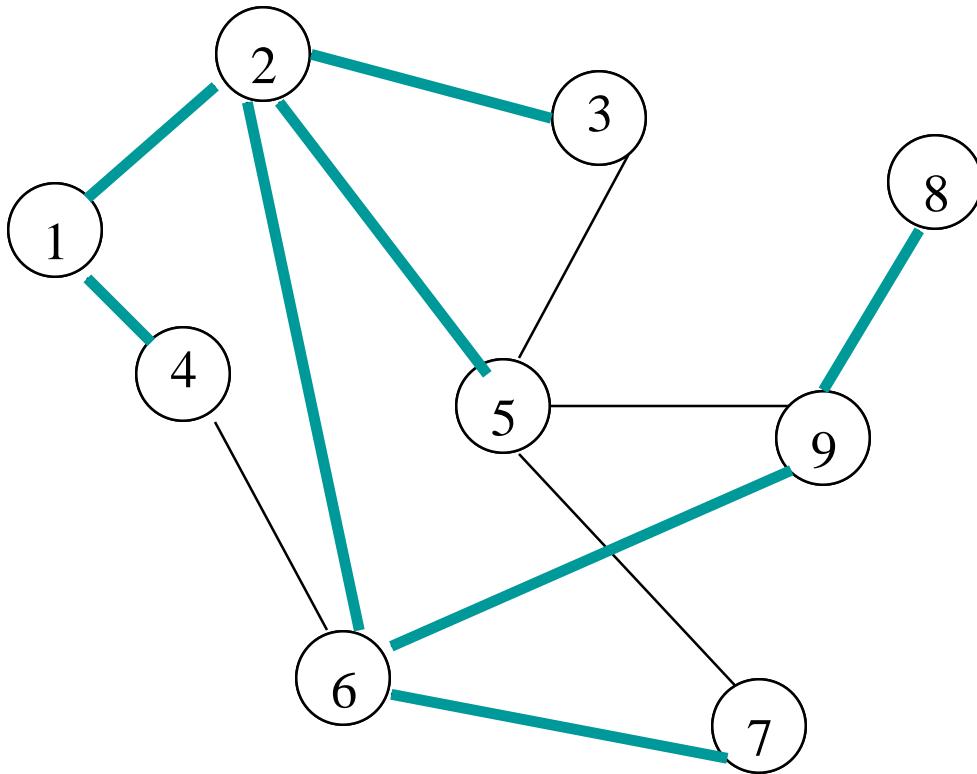




# Хугацаа

- $O(n^2)$  холболтын матриц ашиглахад
- $O(n+e)$  холболтын жагсаалт ашиглахад ( $e$  –ирмэгийн тоо)

# Бүрхэгч мод



1 –р оройгоос Түвшний хайлт хийнэ.

Түвшний бүрхэгч мод.

# Бүрхэгч мод

- Графын дурын оройгоос Түвшний хайлт хийж эхэлнэ.
- Хэрвээ граф холбогдсон бол,  $n-1$  ирмэгийг зочлоогүй оройнууд бүрхэгч модыг тодорхойлоход ашиглана (түвшний хайлтын бүрхэгч мод).
- Хугацаа
  - $O(n^2)$  холболтын матриц ашиглахад
  - $O(n+e)$  холболтын жагсаалт ашиглахад ( $e$  – ирмэгийн тоо)

# Гүний хайлт (Depth-First Search)

depthFirstSearch(**v**)

{

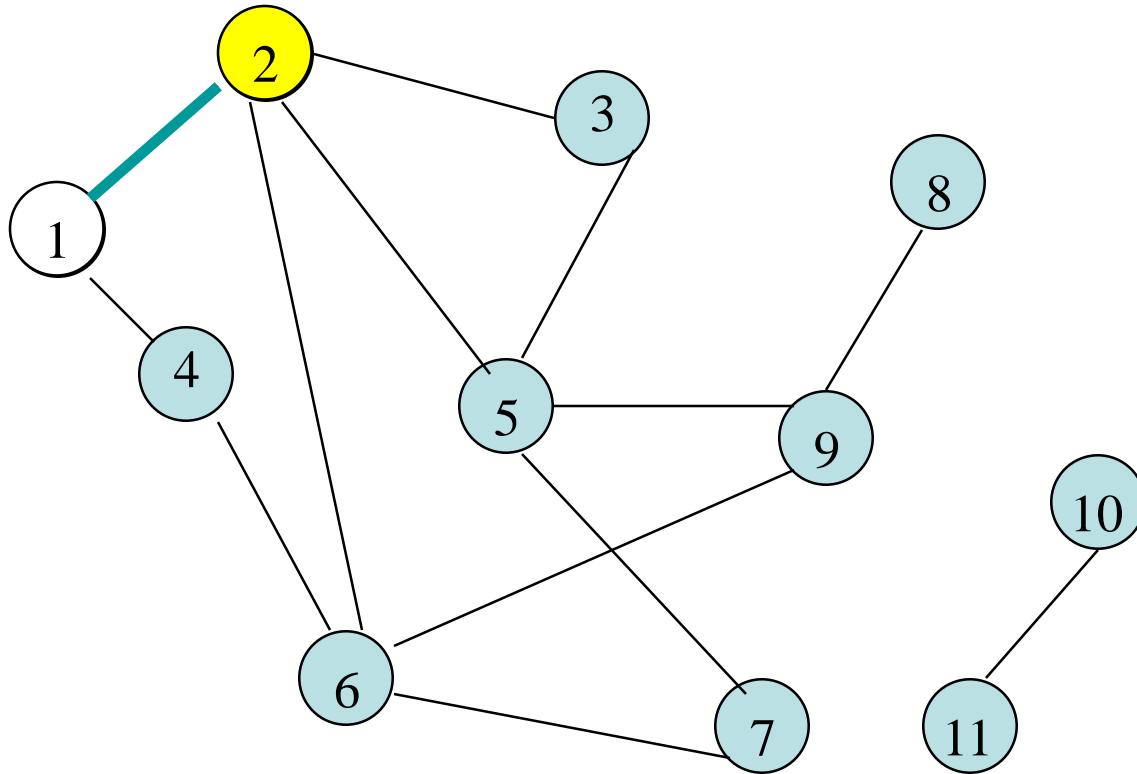
Орой **v** –г зочилсон болгох.

for (**v** -тэй хөрш зочлоогүй орой **u**  
-н хувьд)

    depthFirstSearch(**u**);

}

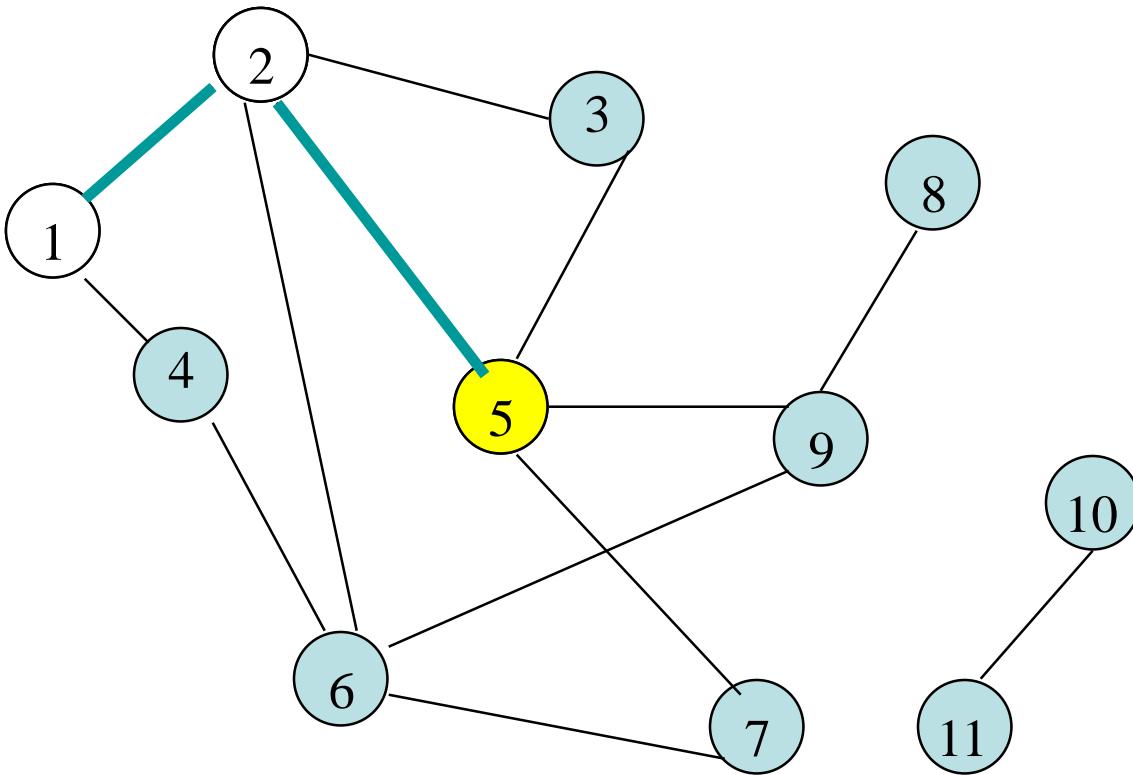
# Гүний хайлтын жишээ



Хайж эхлэх орой - 1.

Орой 1 –г тэмдэглээд 2 эсхүл 4 –с гүний хайлт  
эхлүүлнэ.  
Орой 2 –г сонгосон гэж үзье.

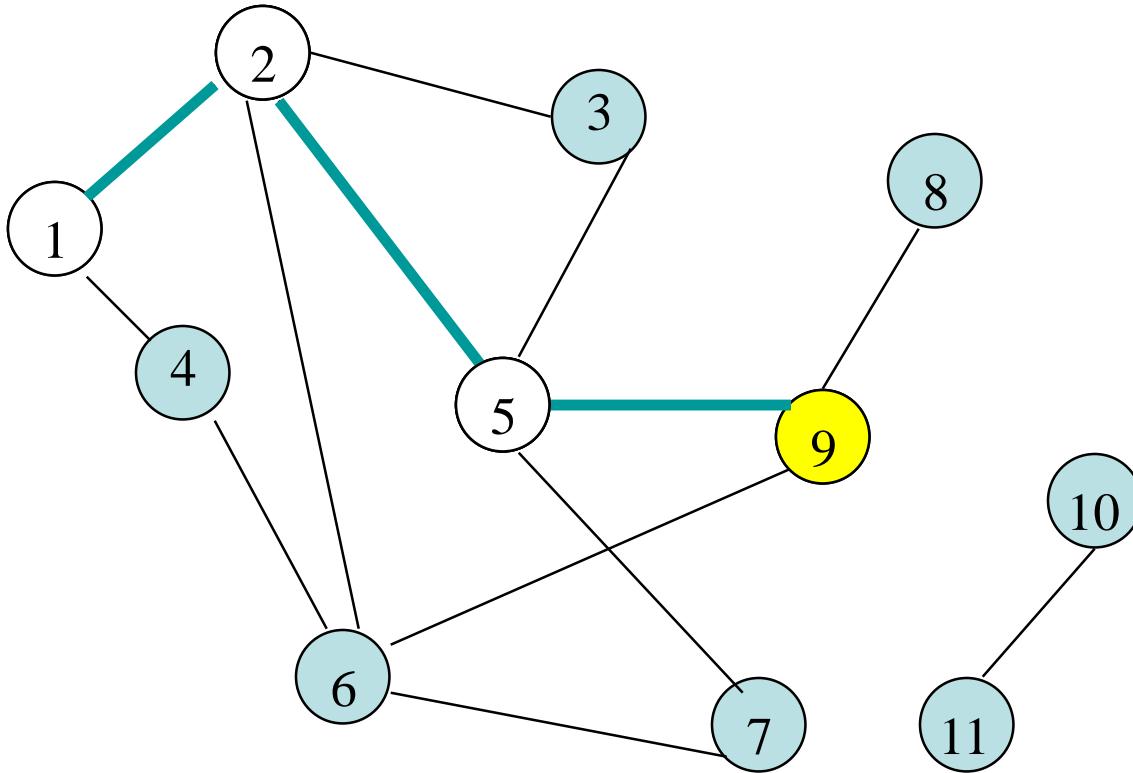
# Гүний хайлтын жишээ



Орой 2 –г тэмдэглээд 3, 5, эсхүл 6 -с гүний хайлт эхлүүлнэ

Орой 5 –г сонгосон гэж үзье.

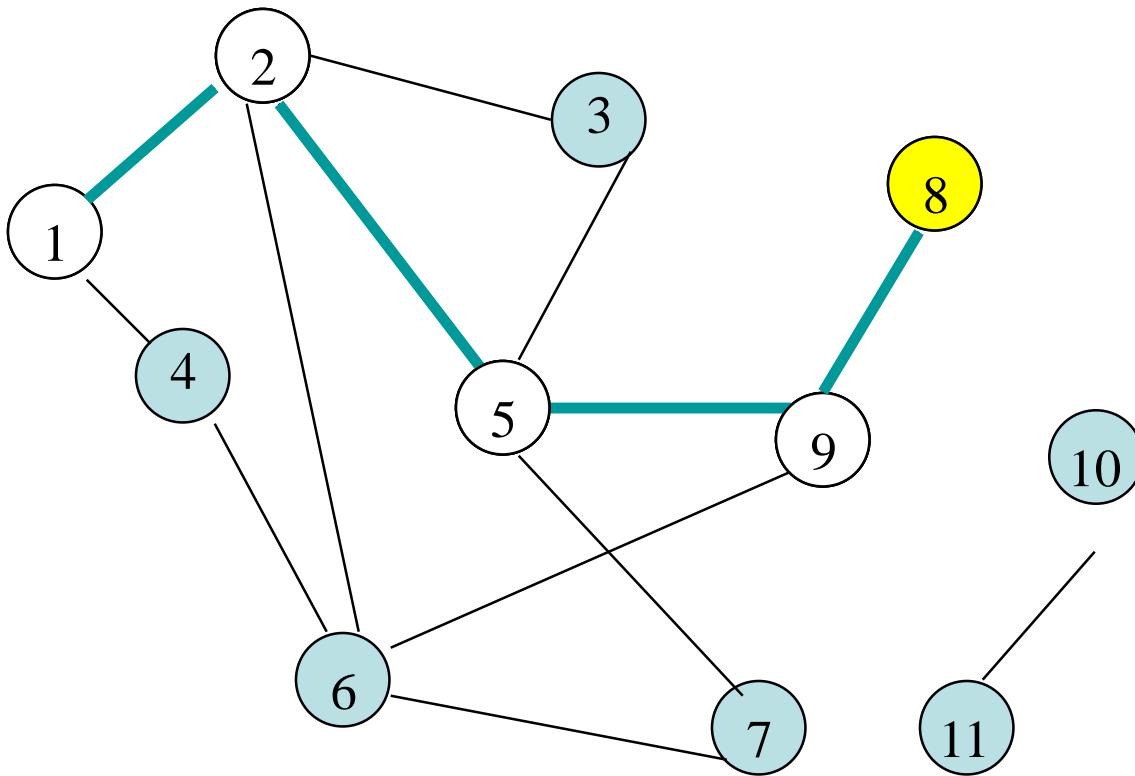
# Гүний хайлтын жишээ



Орой 5 –г тэмдэглэж 3, 7, эсхүл 9 -с гүний хайлт эхлүүлнэ

Орой 9 –г сонгосон гэж үзье.

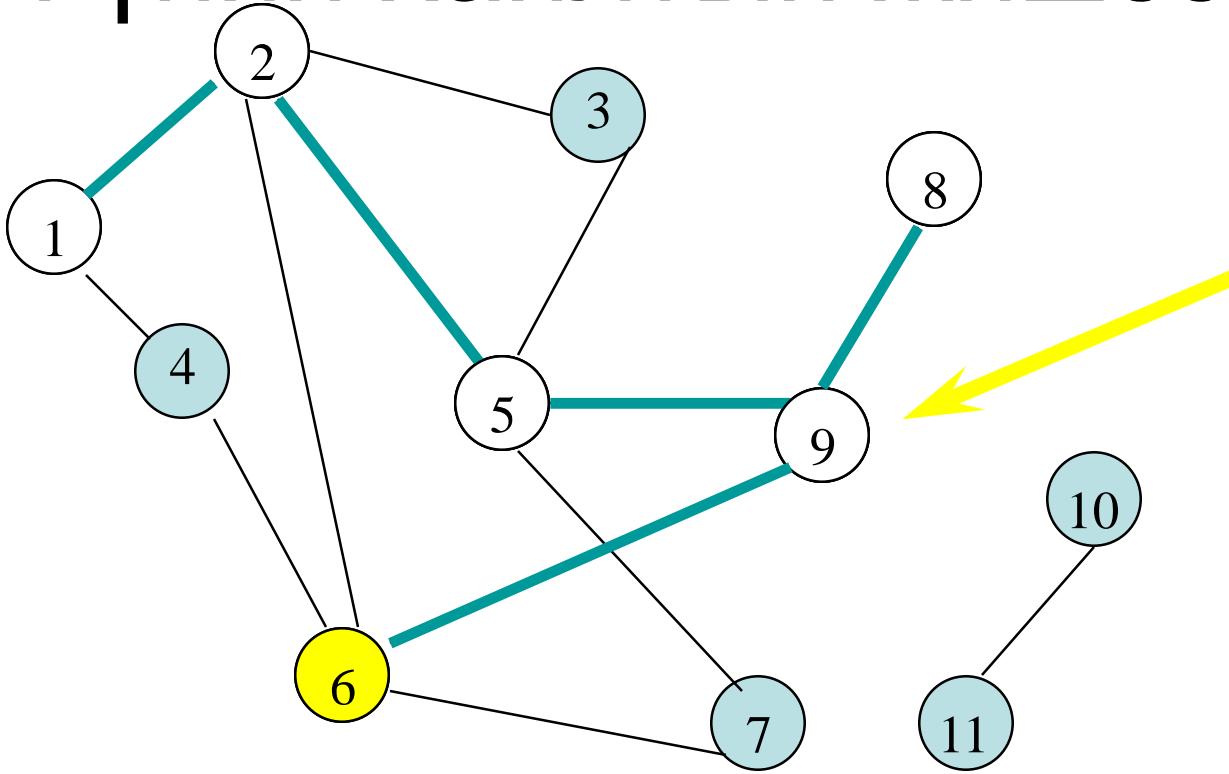
# Гүний хайлтын жишээ



Орой 9 –г тэмдэглээд 6 эсхүл 8 -с гүний хайлт эхлүүлнэ

Орой 8 –г сонгосон гэж үзье.

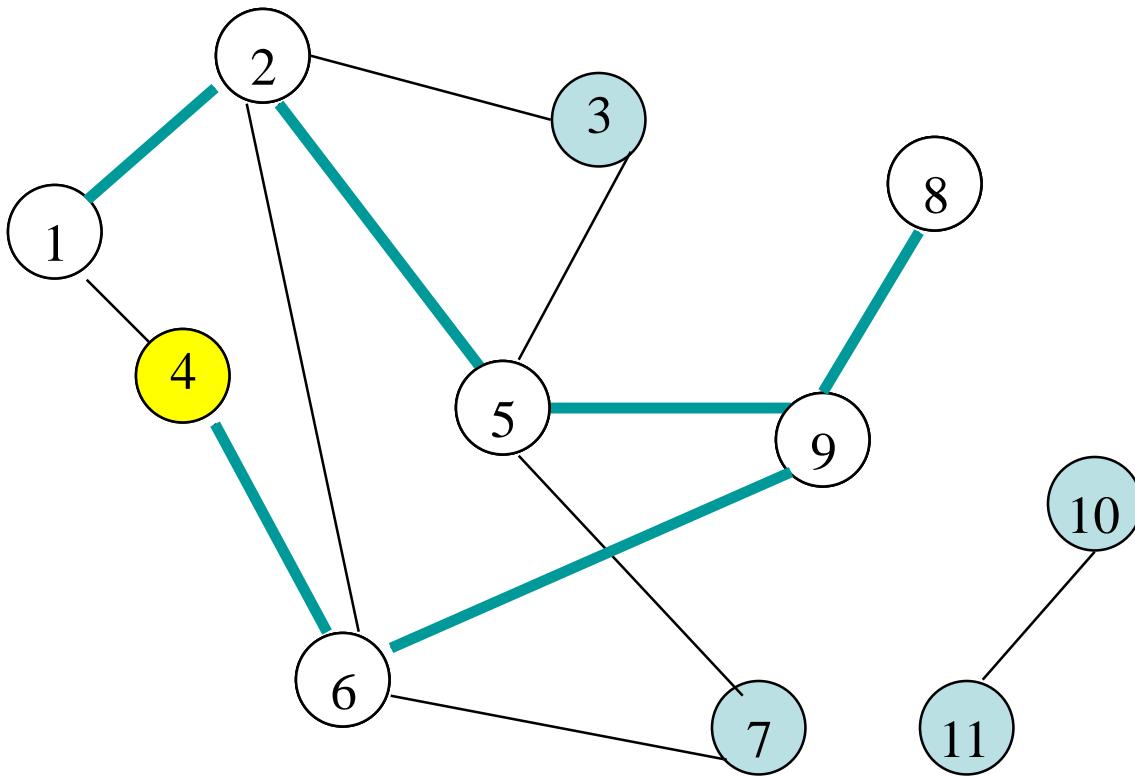
# Гүний хайлтын жишээ



Орой 8 –г тэмдэглээд орой 9 рүү буцна.

Орой 9 -с  $\text{dfs}(6)$ .

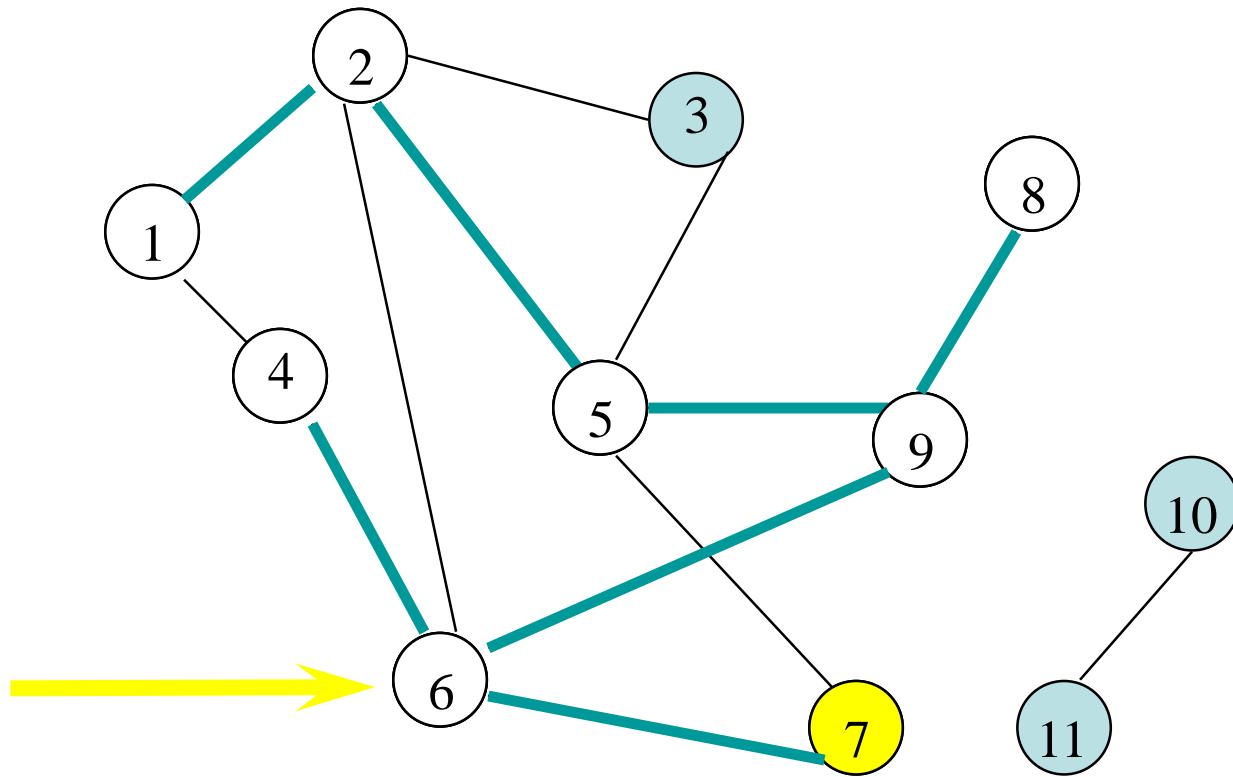
# Гүний хайлтын жишээ



Орой 6 –г тэмдэглээд 4 эсхүл 7 -с гүний хайлт  
эхлүүлнэ

Орой 4 –г сонгосон гэж үзье.

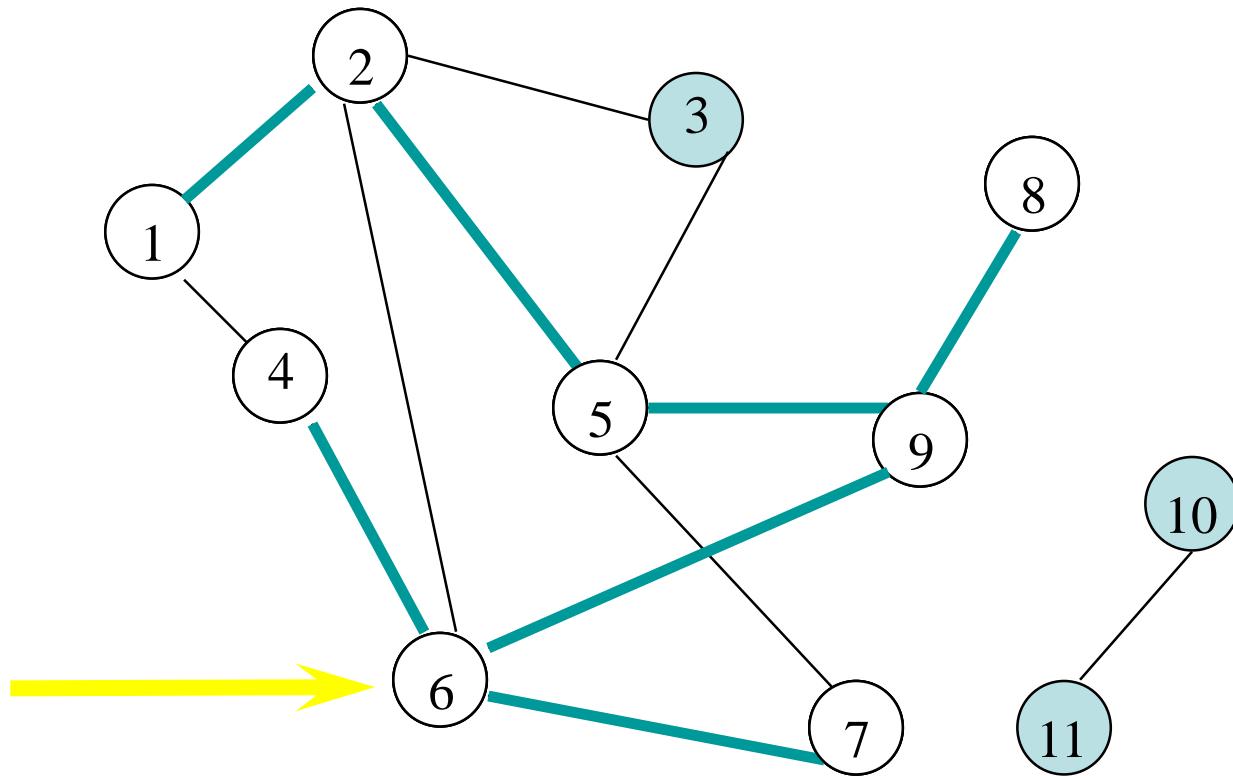
# Гүний хайлтын жишээ



Орой 4 –г тэмтэглээд орой 6 -рүү буцна

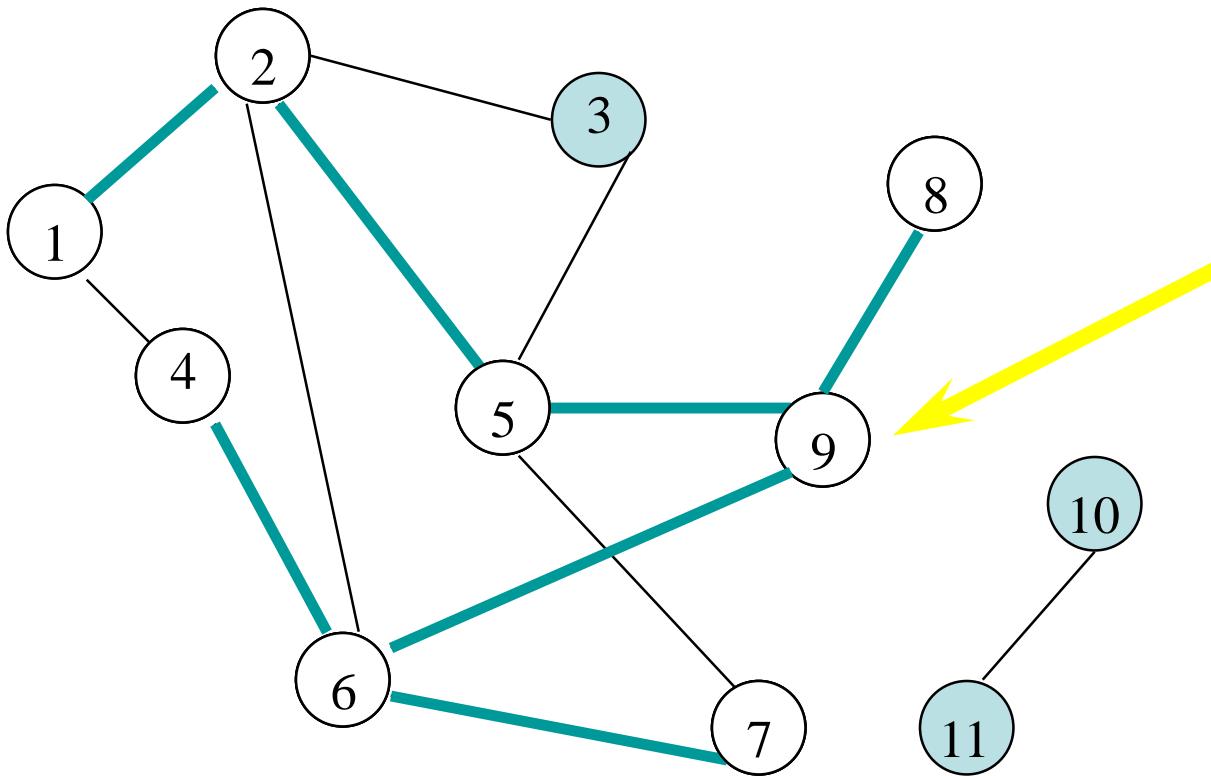
Орой 6 -с  $\text{dfs}(7)$ .

# Гүний хайлтын жишээ



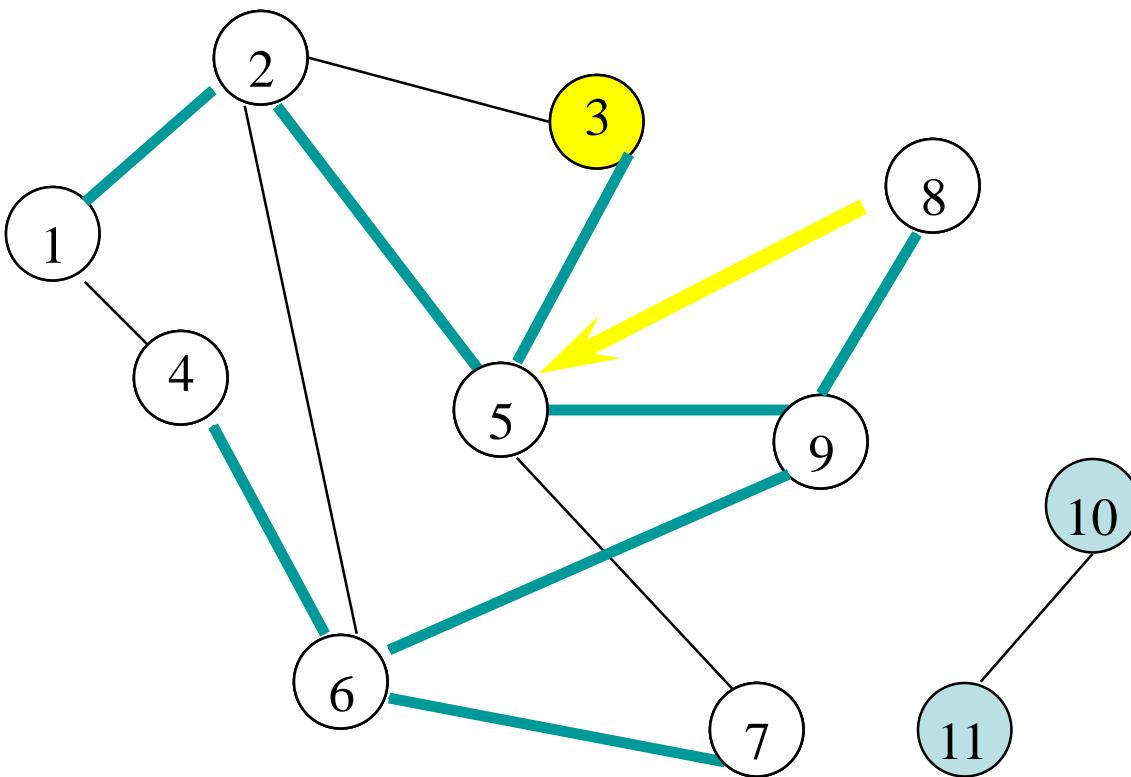
Орой 7 –г тэмдэглээд орой 6 -рүү буцна  
9 рүү буцна

# Гүний хайлтын жишээ



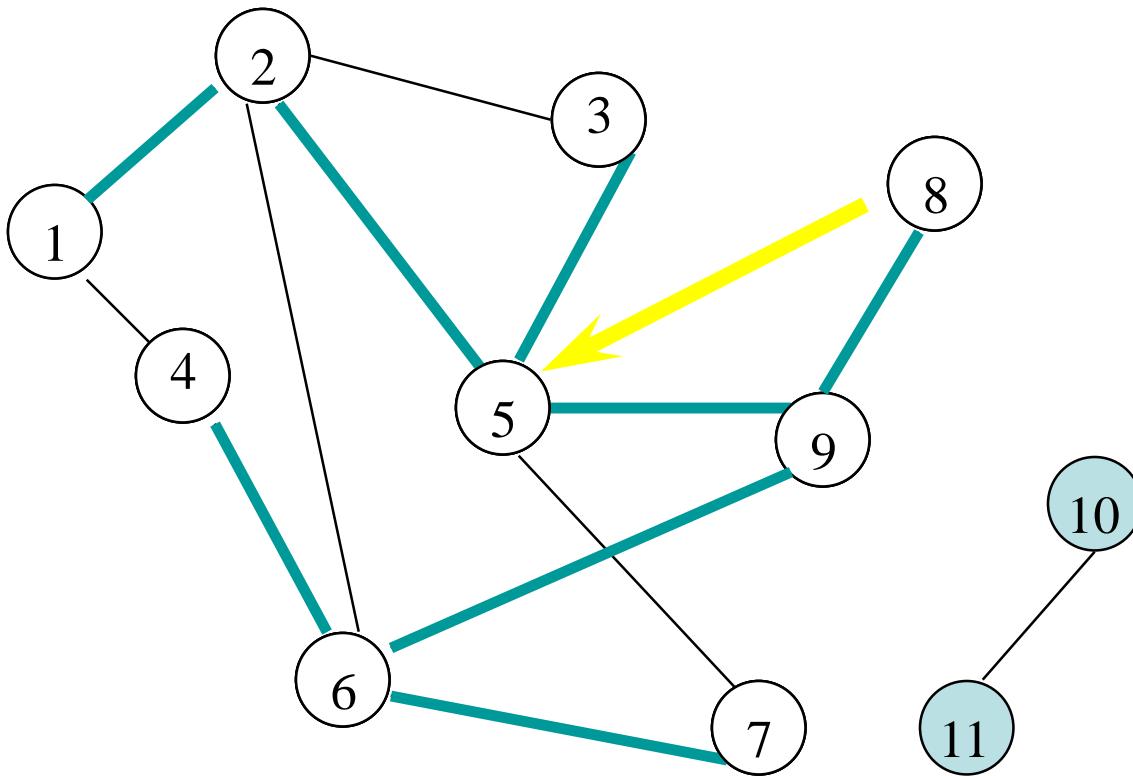
5 -рүү буцна

# Гүний хайлтын жишээ



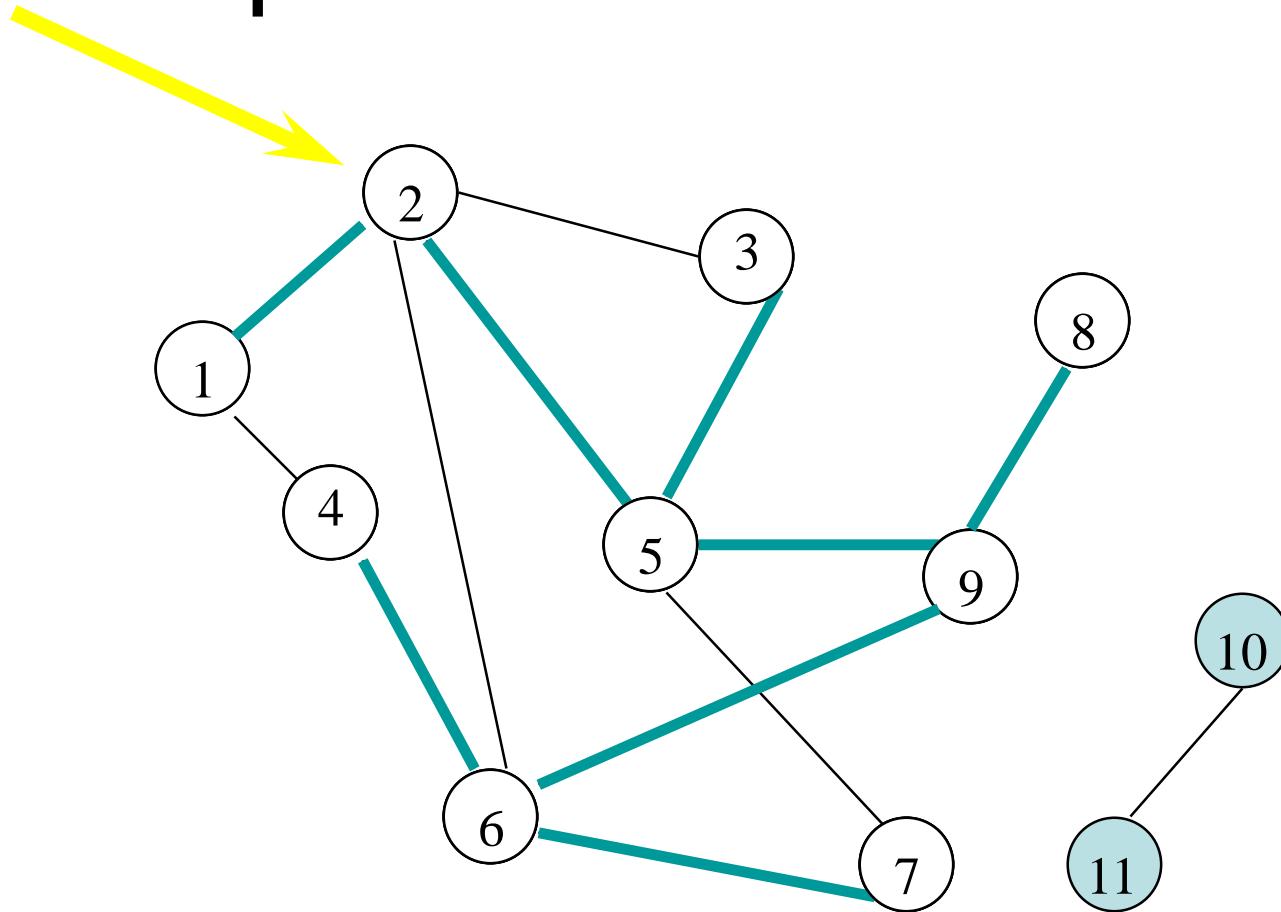
dfs(3).

# Гүний хайлтын жишээ



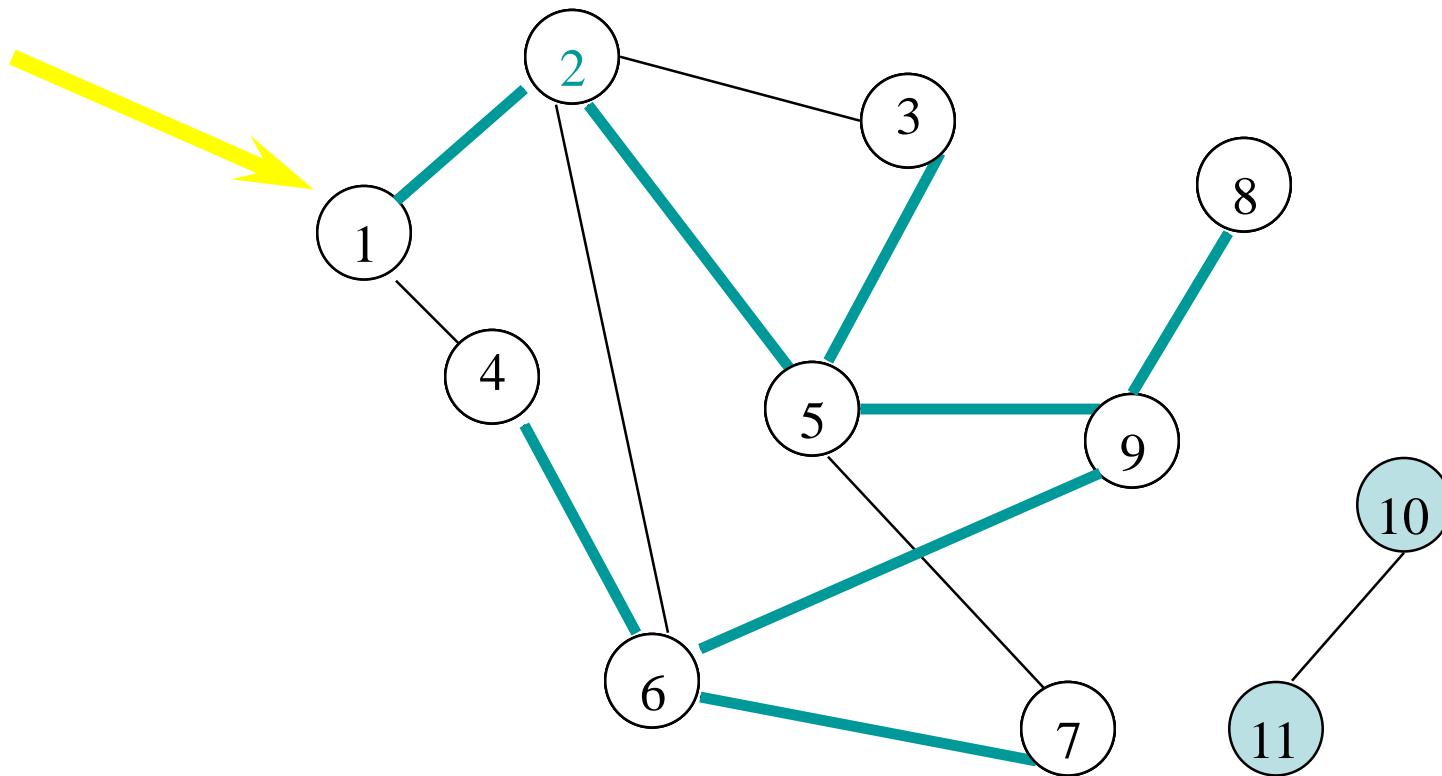
Орой 3 –г тэмдэглээд орой 5 -рүү буцна  
2 -рүү буцна

# Гүний хайлтын жишээ



1 -рүү буцна

# Гүний хайлтын жишээ



Дуудсан функц руу буцна.

# Гүний хайлтын шинж

- Хугацааны хувьд BFS -тай адил.
- Зам хайх, холбогдсон бүрдүүлбэр, бүрхэгч модны хувьд адилхан шинжтэй.
- Граф холбогдсон бол тэмдэглээгүй оройнуудад хүрэх ирмэгүүд гүний хайлтын бүрхэгч модыг тодорхойлно.
- Зарим бодлогод bfs, заримд нь dfs илүү тохирдог.