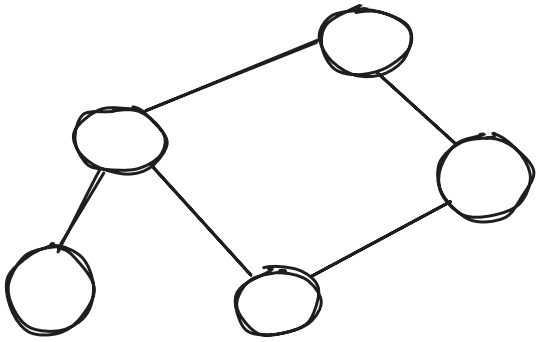
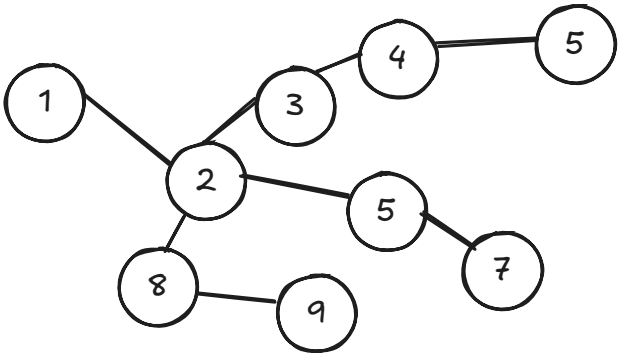
14. Definition of a tree and corresponding concepts (height, level, binary tree, etc.),Binary tree traversals: pre-order, post-order, in-order, Traversing a tree using BFS.

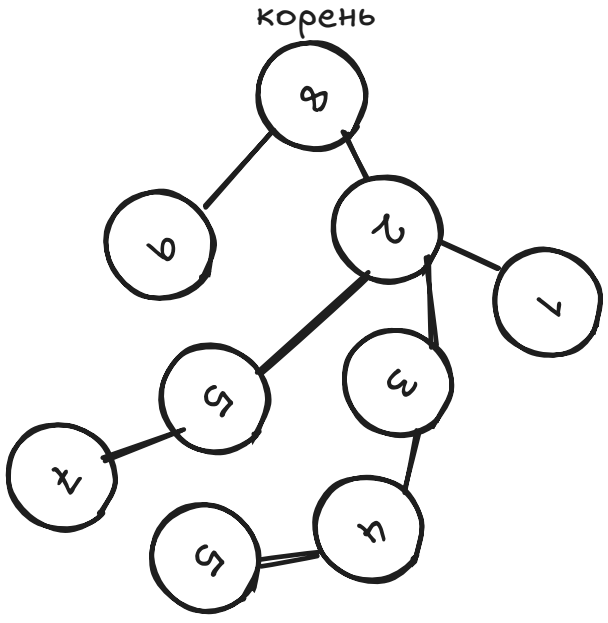


Это цикл.

Дерево - это связный граф без циклов.



В дереве отделим какую-то вершину/узел и назовем ее *корнем*:



Данное дерево *не является* бинарным.

Дерево называется бинарным, если из каждой вершины выходит не более 2 дочерних узла.

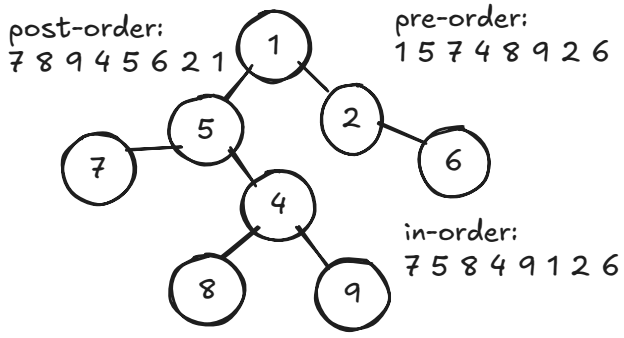
Определения бинарных деревьев:

1. пустое множество - это дерево
2. множество, состоящее из одного значения, это дерево
3. если `T1` и `T2` - деревья, то { `e` , { `T1` }, { `T2` }} - это дерево, где `e` - корень, `T1` - левое поддерево, `T2` - правое.

Узел дерева называется **листом** (leaf), если у него *нет* дочерних узлов. *Узел* дерева называется **корнем**, если у него *нет* родительного узла.

Представление дерева в коде

Пусть имеем следующее дерево:




```
#include <iostream>

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int value) : val(value), left(nullptr), right(nullptr) {}
};

// Pre-order traversal to print the tree nodes
void print(TreeNode* node) {
    if (node == nullptr) {
        return;
    }
    std::cout << node->val << " ";
    print(node->left);
    print(node->right);
}

void deleteTree(TreeNode* node) {
    if (node == nullptr) {
        return;
    }
    deleteTree(node->left);
    deleteTree(node->right);
    delete node;
}

int main() {
```

```
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(5);
    root->right = new TreeNode(2);
    root->left->right = new TreeNode(4);
    root->left->left = new TreeNode(7);
    root->right->right = new TreeNode(6);
    root->left->right->left = new TreeNode(8);
    root->left->right->right = new TreeNode(9);

    std::cout << "Tree nodes in pre-order traversal:" << std::endl;
    print(root);
    std::cout << std::endl;

    deleteTree(root);

    return 0;
}
```

в зависимости от того, где расположена строка вывода в функции `print` мы будем иметь

- *pre-order* если она написана до вызовов `print`
- *in-order*, если между вызовами
- *post-order*, если после вызовов.

Pre-order Traversal

1. Посетить узел
2. Посетить левое поддерево узла
3. Посетить правое поддерево узла

Post-order Traversal

1. Посетить левое поддерево узла
2. Посетить правое поддерево узла
3. Посетить узел

In-order Traversal

1. Посетить левое поддерево узла
2. Посетить узел
3. Посетить правое поддерево узла

BFS (Обход в ширину):

Обход в ширину использует *очередь* `std::queue` для обработки узлов уровня за уровнем. Узлы обрабатываются сначала на одном уровне, а затем на следующем.

Алгоритм:

1. Создать пустую очередь.
2. Поместить корень дерева в очередь и объявить его посещённым.
3. Пока очередь не пуста:
    - Извлечь текущую вершину из очереди.
    - Обработать текущую вершину.
    - Добавить всех её дочерних узлов (если они существуют) в очередь.

```cpp
#include <iostream>
#include <queue>

// Структура узла дерева
struct TreeNode {
    int value;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int val) : value(val), left(nullptr), right(nullptr) {}
};

// BFS для дерева
void BFS(TreeNode* root) {
    if (root == nullptr) return; // Если дерево пустое, ничего не делаем

    std::queue<TreeNode*> q;
    q.push(root); // Добавляем корень в очередь

    while (!q.empty()) {
        TreeNode* current = q.front(); // Извлекаем элемент из очереди
        q.pop();

        // Обработка текущего узла
        std::cout << current->value << " ";

        // Добавляем детей узла в очередь
        if (current->left != nullptr) q.push(current->left);
        if (current->right != nullptr) q.push(current->right);
    }
}

// Пример дерева и вызов BFS
int main() {
    // Создаем пример дерева
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);

    std::cout << "BFS traversal of the tree:\n";
    BFS(root);

    return 0;
}
```

Длина пути и высота дерева

> *Длина самого длинного пути из корня до листа дерева* называется **высотой** дерева (длина пути — это количество рёбер).

> В худшем случае **сложность BFS** $O(h)$, где h — высота дерева.

Утверждение:

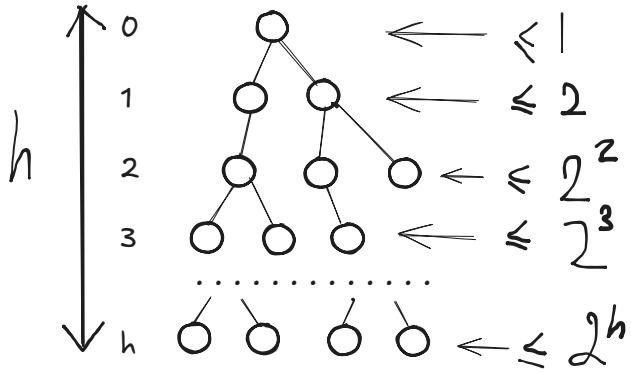
Если дерево содержит n элементов и высота этого дерева h, то:

$$\log_2 n - 1 \leq h < n$$

Доказательство:

① $h < n$, так как в дереве с n элементами всегда $n - 1$ рёбер.

② Докажем, что $n \leq 2^{h+1}$.



$$n \leq 1 + 2 + 2^2 + 2^3 + \cdots + 2^h = 2^{h+1} - 1 \leq 2^{h+1}$$

Так как $n \leq 2^{h+1}$, то:

$$\log_2 n \leq h + 1$$

$$\log_2 n - 1 \leq h$$

Оценки высоты дерева:

- **В лучшем случае**:

$$h = O(\log n)$$

- **В худшем случае**:

$$h = O(n)$$

Итог:

$$h = O(n) \text{ и } h = \Omega(\log_2 n)$$