




Древовидные структуры

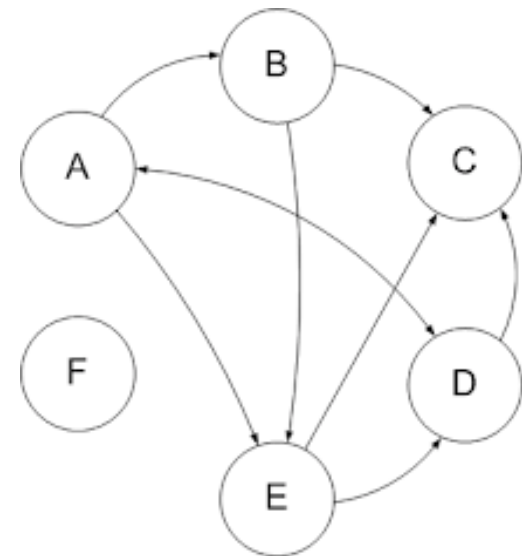
Составитель: Рощупкин Александр



Древовидные структуры данных

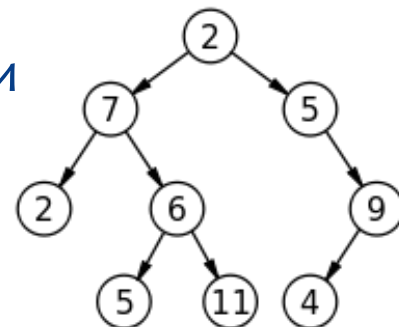
Графы

- * Структура данных состоящая из множества вершин и рёбер
- * **Вершины** – это основные объекты графа, служащие для хранения данных
- * **Рёбра** – это связи между вершинами
- * **Дуга** – ориентированное ребро
- * Граф с рёбрами – ориентированный
- * Цикл – дуга, исходящая и входящая в тот же узел



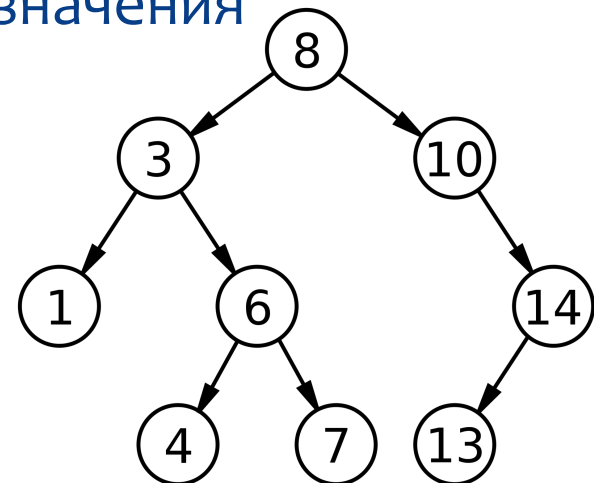
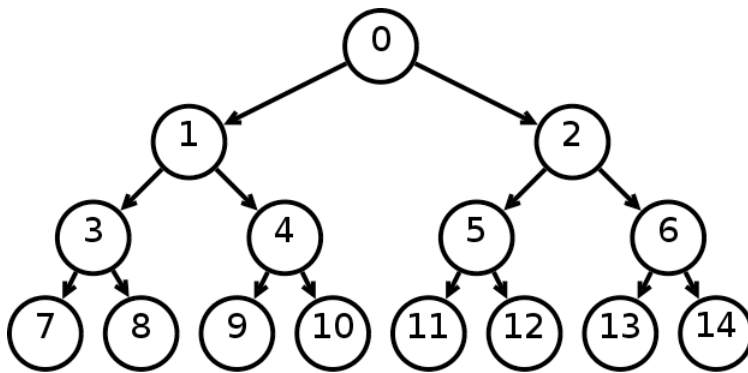
Понятие дерева

- * Дерево – связанный ациклический граф
- * **Связность** означает наличие путей между любой парой вершин
- * **Ациклическость** — отсутствие циклов и то, что между парами вершин имеется только по одному пути
- * Взвешенное дерево – это дерево, вершинам которого задан определённый вес
- * **Корень** – узел, в который не входят дуги
- * **Лист** – концевой узел
- * **Узел ветвления** – не концевой узел



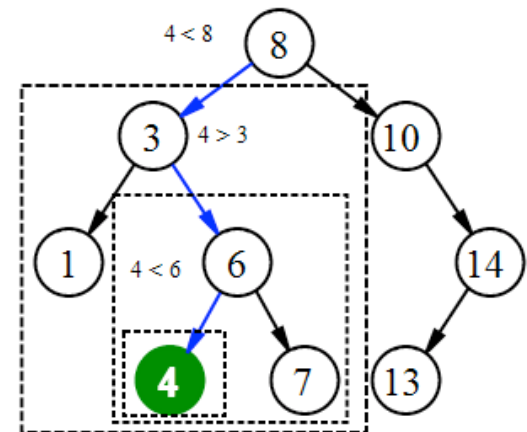
Бинарное дерево поиска

- * Это бинарное дерево, обладающее дополнительными свойствами:
- * Значение левого потомка меньше значения родителя,
- * Значение правого потомка больше значения родителя для каждого узла дерева



Поиск элемента

- * Для каждого узла метод сравнивает значение его ключа с искомым ключом
- * Если ключи одинаковы, то метод возвращает текущий узел, в противном случае метод вызывается рекурсивно для левого или правого поддерева
- * Узлы, которые посещает функция образуют нисходящий путь от корня, так что время ее работы $O(h)$, где h — высота дерева



Поиск максимума и минимума

- * Чтобы найти минимальный элемент в бинарном дереве поиска, необходимо просто следовать указателям left от корня дерева, пока не встретится значение null
- * Если у вершины есть левое поддерево, то по свойству бинарного дерева поиска в нем хранятся все элементы с меньшим ключом
- * Если его нет, значит эта вершина и есть минимальная
- * Максимальный элемент находится движением вправо

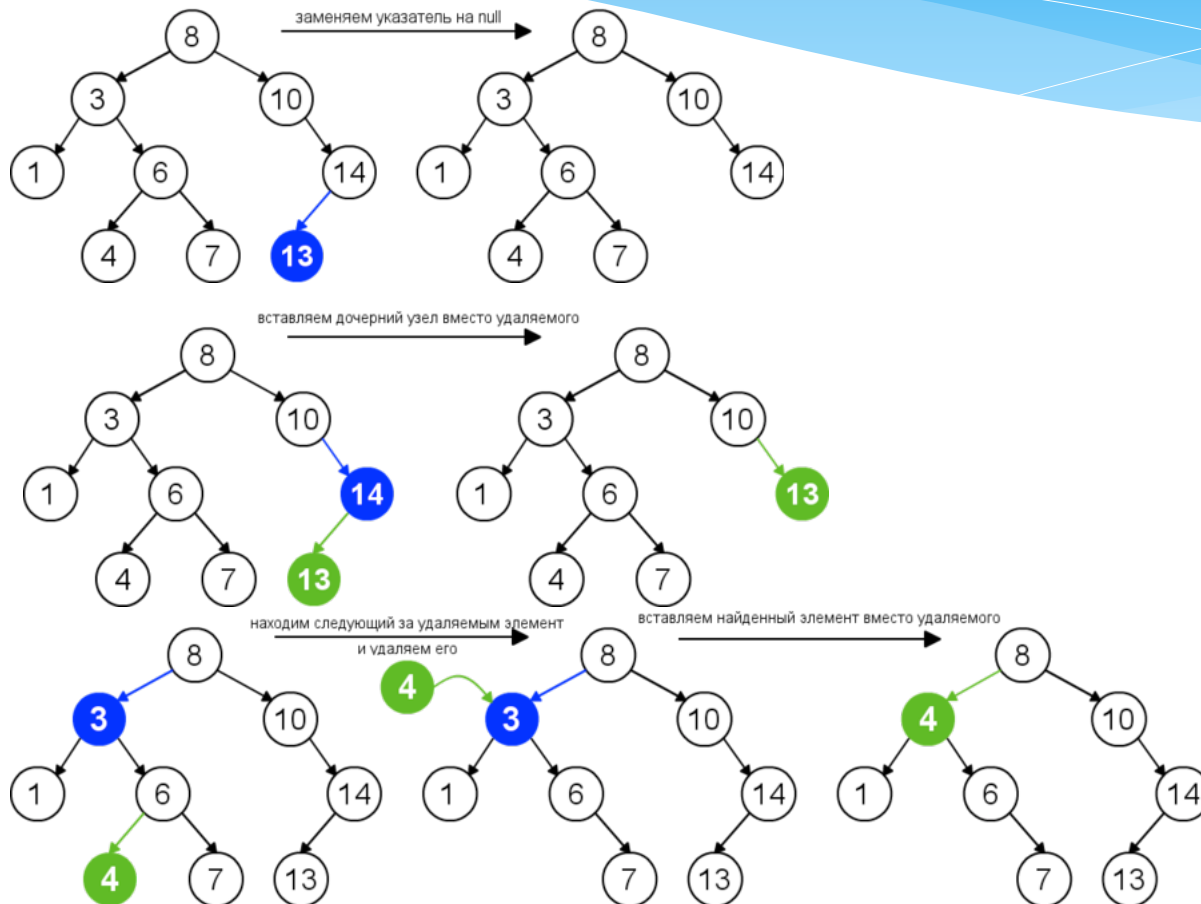
Добавление нового узла

- * Для каждого узла метод сравнивает значение его ключа с искомым ключом
- * Необходимо просто следовать указателям right или left от корня дерева, пока не встретится значение null
- * Вставить узел вместо значения null

Удаление узла

- * При удалении узла есть 3 основных случая:
 - * У узла нет дочерних узлов, то у его родителя нужно просто заменить указатель на null
 - * Если у узла есть только один дочерний узел, то нужно создать новую связь между родителем удаляемого узла и его дочерним узлом
 - * Если у узла два дочерних узла, то нужно найти следующий за ним элемент (у этого элемента не будет левого потомка), его правого потомка подвесить на место найденного элемента, а удаляемый узел заменить найденным узлом

Удаление узла

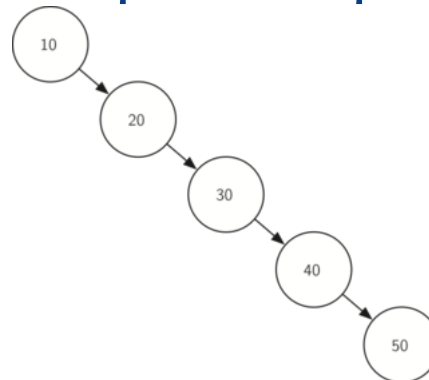


Обход дерева

- * Обход всех вершин дерева можно совершать несколькими способами:
 - * **Обход в ширину** - идет из начальной вершины, посещает сначала все вершины находящиеся на расстоянии одного ребра от начальной, потом посещает все вершины на расстоянии два ребра от начальной и так далее
 - * **Обход в глубину** - идет из начальной вершины, посещая еще не посещенные вершины без оглядки на удаленность от начальной вершины

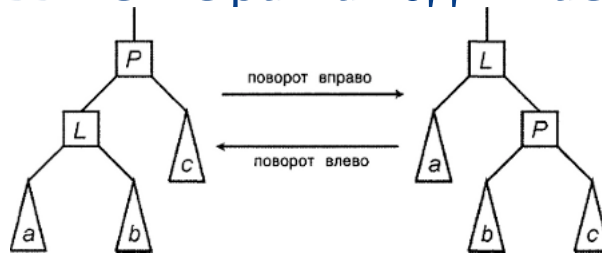
Вырожденное бинарное дерева

- * Всегда желательно, чтобы все пути в дереве от корня до листьев имели примерно одинаковую длину, то есть чтобы глубина и левого, и правого поддеревьев была примерно одинакова в любом узле
- * В противном случае теряется производительность



Баланс бинарного дерева

- * Для балансировки дерева применяется операция «поворот дерева», при этом повышается ранг того узла, который становится выше, т.е. повышение ранга поднимает узел на один уровень вверх



- * Левый дочерний узел в позицию его родительского узла, правое дочернее дерево становится новым левым дочерним поддеревом родительского узла
- * Для повышения ранга на 2 уровня используются спаренный двусторонний поворот

Пример бинарного дерева

- * Бинарное дерево представляет собой класс, в котором есть вложенный класс узла дерева
- * В классе узла находится две ссылки, на левое и правое поддерево

```
public class BinaryTree {  
  
    private Node root;  
  
    private static class Node {  
        int value;  
        Node left;  
        Node right;  
    }  
}
```

Домашнее задание

- * Сделать поиск, вставку, удаление элемента в бинарное дерево поиска
- * Определить, является ли заданное двоичное дерево деревом поиска
- * Найти в данном дереве такую вершину, что она будет корнем поддерева поиска с наибольшим количеством вершин

Балансирующие деревья

- * AVL дерево
- * Красно-чёрное дерево

AVL дерево

Красно-чёрное дерево