Бинарное дерево поиска

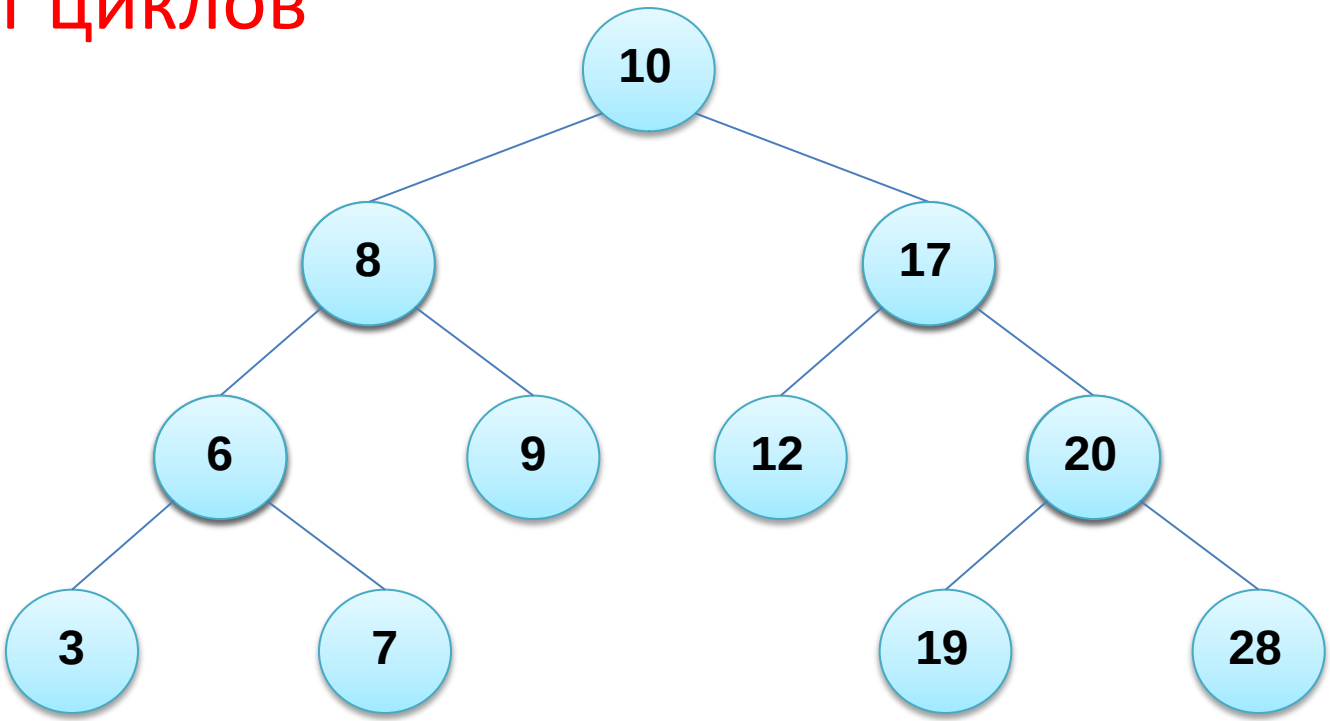
19 июля 2017 г.

Повторение

1. В чем отличия между односвязным и двусвязным списком?
2. Что такое голова и хвост списка?
3. Могут ли голова и хвост списка совпадать?
4. Какие поля обязательно имеет структура для узла в случае создания односвязного списка и в случае двусвязного списка?
5. Можно ли реализовать стек на основе односвязного списка? А очередь?

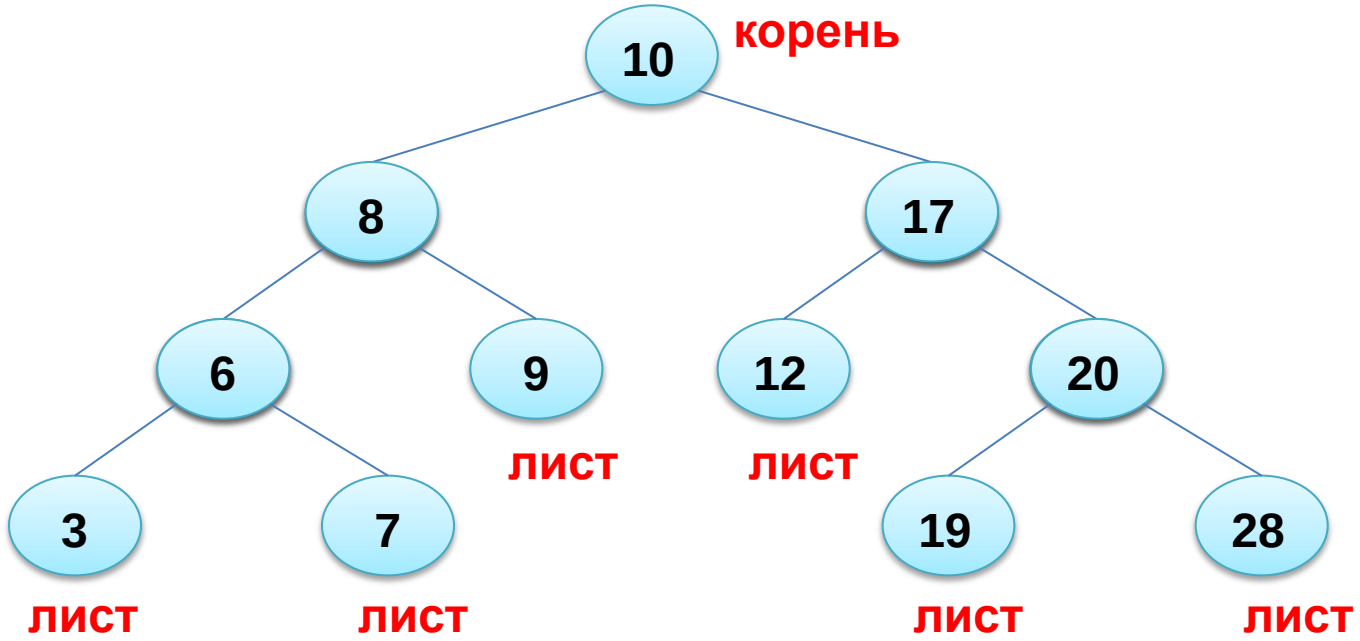
Бинарное (двоичное) дерево поиска

- упорядоченная совокупность узлов
- каждый узел имеет не более двух следующих
- нет циклов



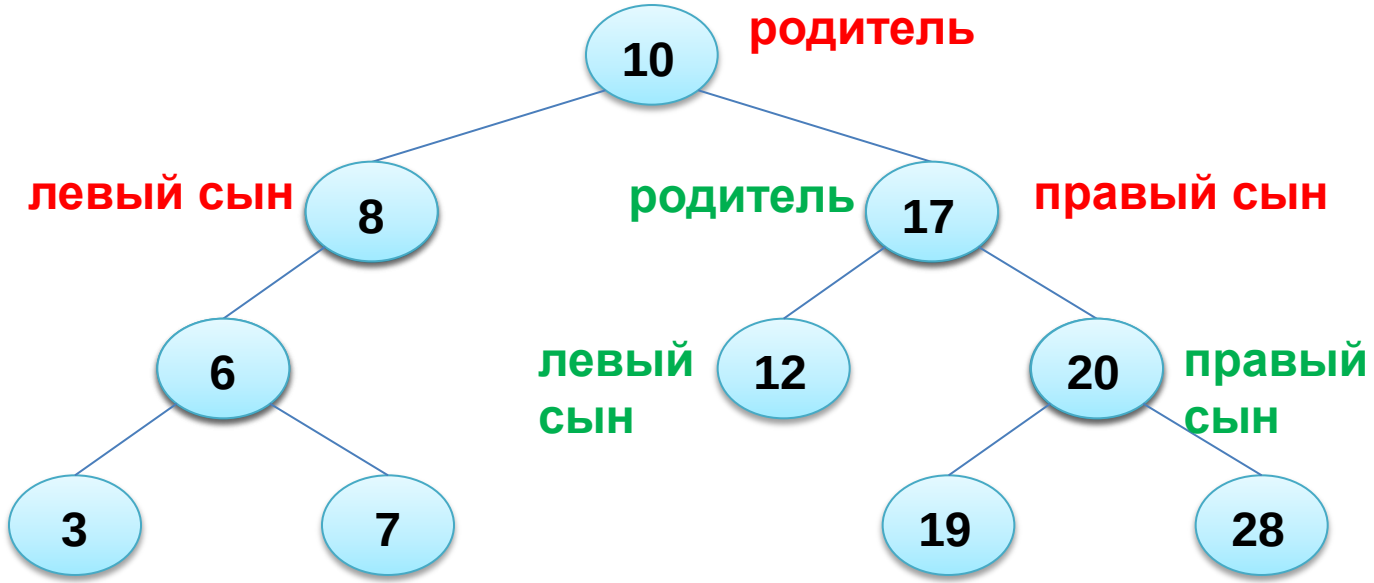
Строение дерева

- корень – вершина дерева (всегда один)
- родительские узлы (родители)
- дочерние узлы (сыновья)
- листья – узлы без сыновей



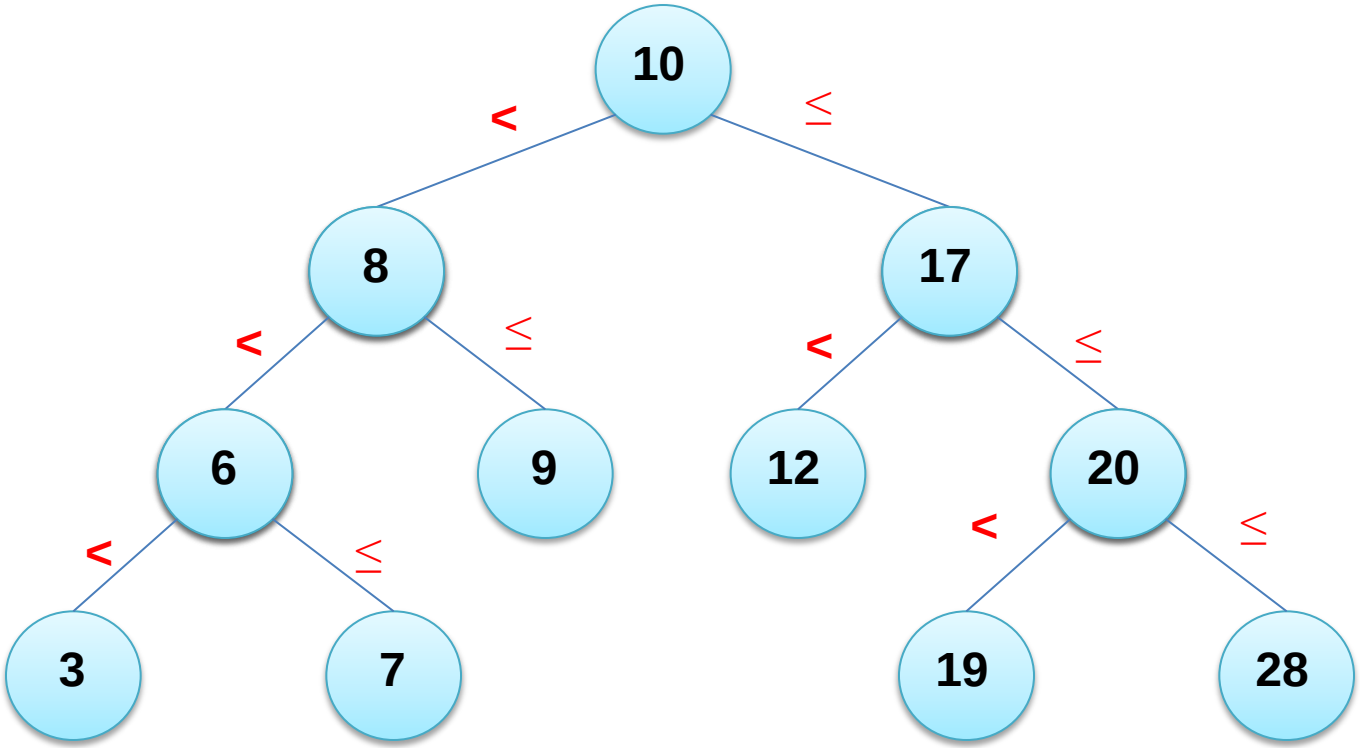
Строение дерева

- левый сын – дочерний узел слева
- правый сын – дочерний узел справа
- каждый узел является родителем для своих сыновей и сыном для своего родителя



Неравенства дерева

- для каждого узла верно, что значение в его левом сыне меньше, а значение в правом сыне больше (*или равно*), чем в нем самом

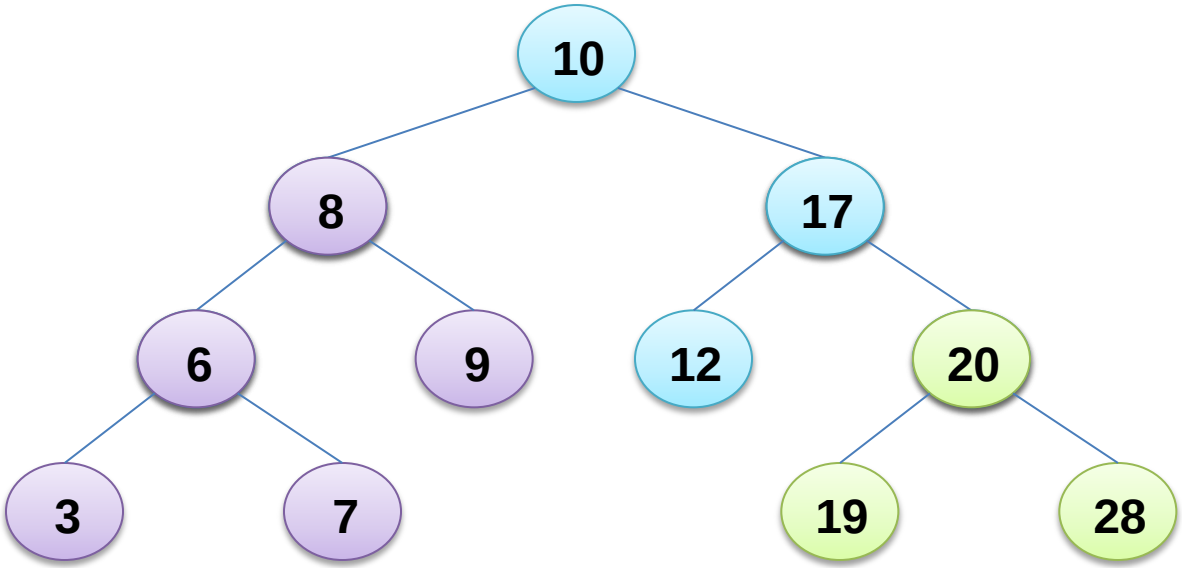


Особенности

- значения всех узлов левее данного узла меньше значения этого узла, а значения всех узлов правее данного узла больше или равны значению в этом узле
- основное назначение бинарного дерева – быстрый поиск элемента с указанным значением
- доступны быстрая вставка и удаление элементов

Деревья и рекурсия

- бинарное дерево – рекурсивная структура:
  - пустая структура является бинарным деревом;
  - дерево – это корень и два связанных с ним бинарных дерева, которые называют левым и правым поддеревьями
- используются рекурсивные алгоритмы



Задание

Даны последовательности чисел:

а) 2, 12, 8, 5, 0, 6

б) 26, 8, 9, 11, 15, 19, 20, 21, 7

Постройте для этих чисел бинарные деревья (какие-нибудь из возможных, чтобы в них соблюдались все необходимые свойства).

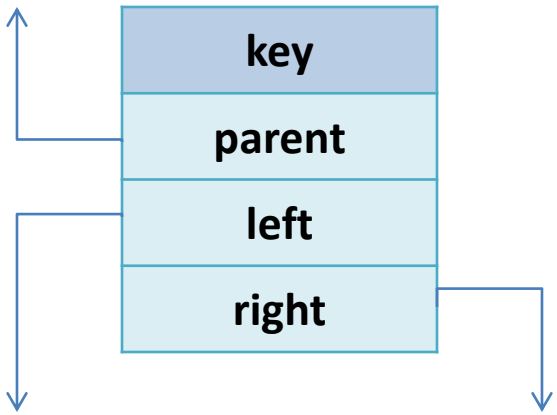
Основные операции

- обойти дерево
- найти элемент (find)
- получить минимальный элемент (min)
- получить максимальный элемент (max)
- вставить элемент в дерево (insert)
- удалить элемент из дерева (erase)

Реализация: узлы

- узел представляет собой структуру
- узел содержит поле с данными и указатели:
  - на левый дочерний узел
  - на правый дочерний узел
  - на родительский узел (этого поля может не быть)

```
template <typename T>
struct node {
        T key;
        node* parent;
        node* left;
        node* right;
};
```



Объявление класса

- создается класс для реализации дерева
- в классе дополнительно создается поле-указатель на корень дерева (root)

```
template <typename T>
struct node { … };

template <typename T>
class BinaryTree{
        node* root;
};
```

Обход дерева

- каждая вершина посещается по одному разу, выводится информация из этой вершины
- используется рекурсия

прямой (корень – левое поддерево – правое поддерево)

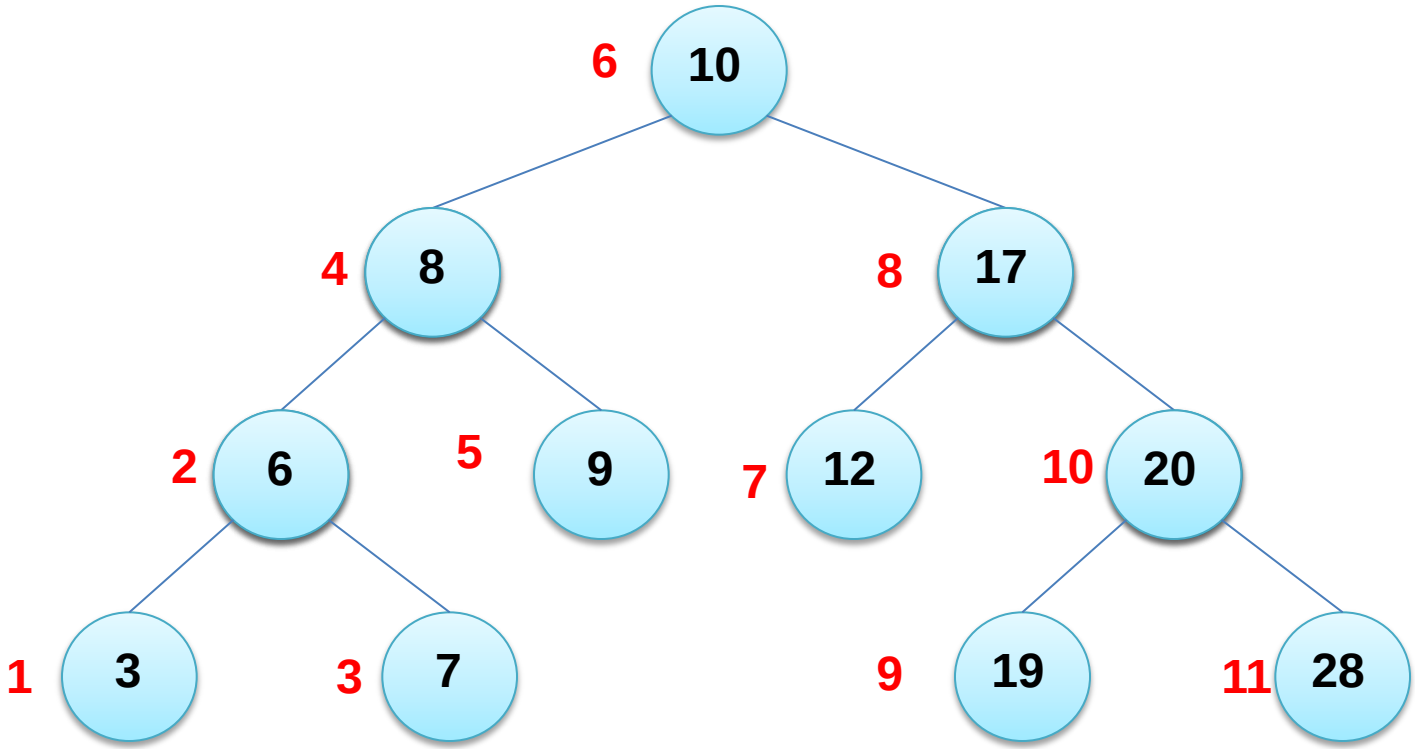
симметричный (левое поддерево – корень – правое поддерево)

обратный (левое поддерево – правое поддерево – корень)

Обход дерева (пример)

- создаем указатель tmp на переданную вершину
- если указатель не равен nullptr, то:
  - если у вершины есть левое поддерево, рекурсивно вызываем функцию обхода для вершины левого поддерева
  - обрабатываем значение в текущей вершине
  - если у вершины есть правое поддерево, рекурсивно вызываем функцию обхода для вершины правого поддерева

Обход дерева (схема)



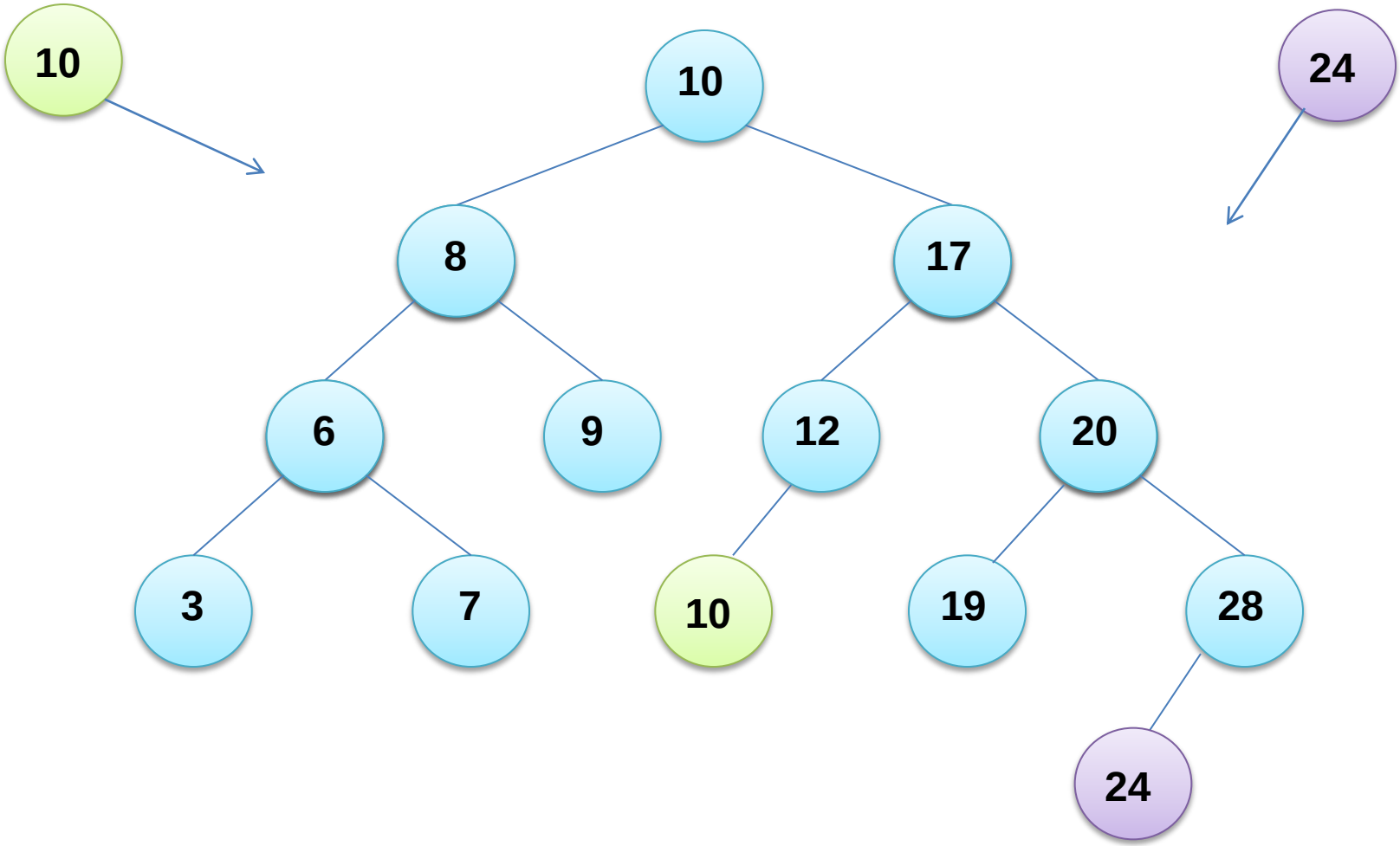
Вставка элемента k

1) если дерево пустое, то создать узел и сделать его корнем дерева
2) если в дереве уже есть вершины, то:
   - создать новый узел, проинициализировать его поля; все указатели установить в nullptr
   - создать указатель currentNode на текущий узел, который сразу будет указывать на корень

Вставка элемента k

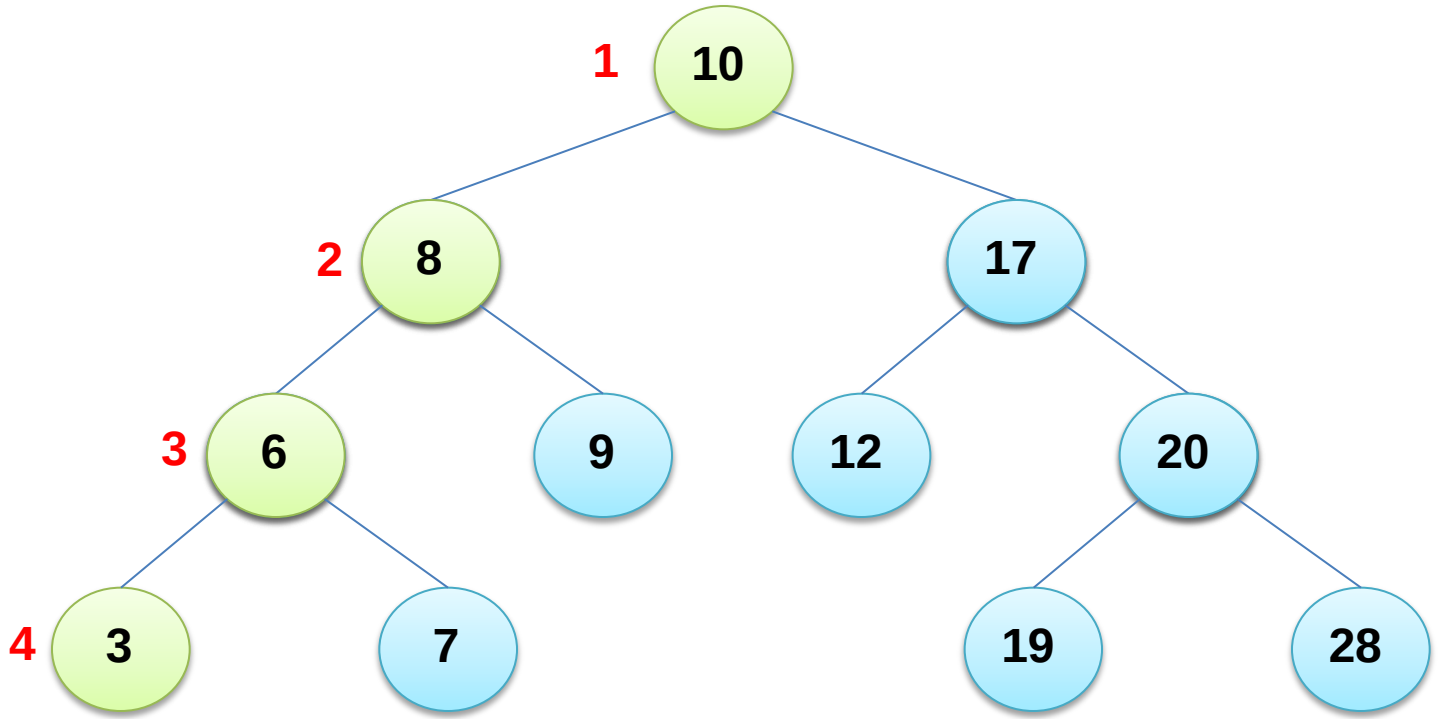
3) пока указатель на текущий узел не nullptr:
   - если k меньше значения в currentNode
     - если у текущего узла есть левый сын, то перейти в левого сына
     - иначе вставить созданный узел (установить его родителя в текущий узел, указатель на левого сына в текущем узле установить на новый узел)
   - если k больше или равно значению в currentNode
     - если у текущего узла есть правый сын, то перейти в правого сына
     - иначе вставить созданный узел (установить его родителя в текущий узел, указатель на правого сына в текущем узле установить на новый узел)

Вставка элемента (схема)



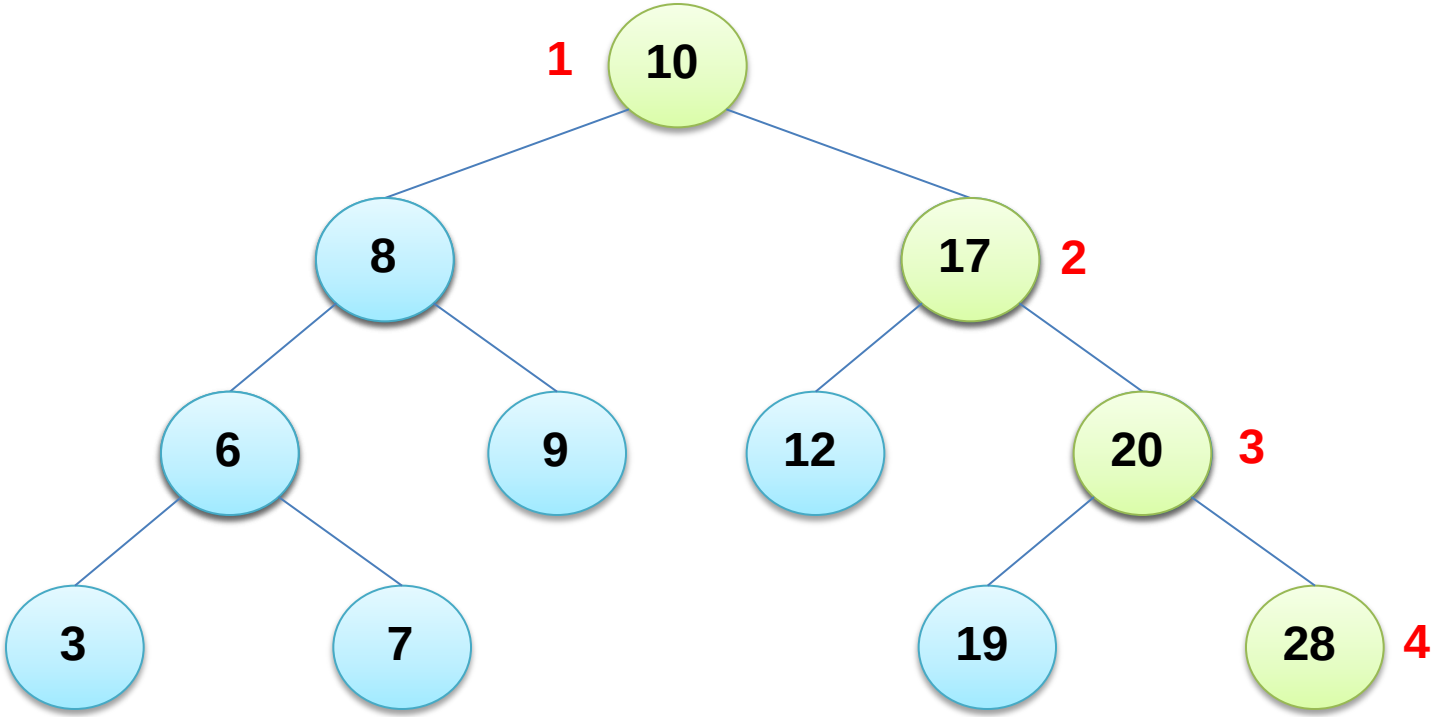
Поиск минимума

- создать указатель на корень дерева
- переходить в левого сына до тех пор, пока указатель на левого сына не nullptr



Поиск максимума

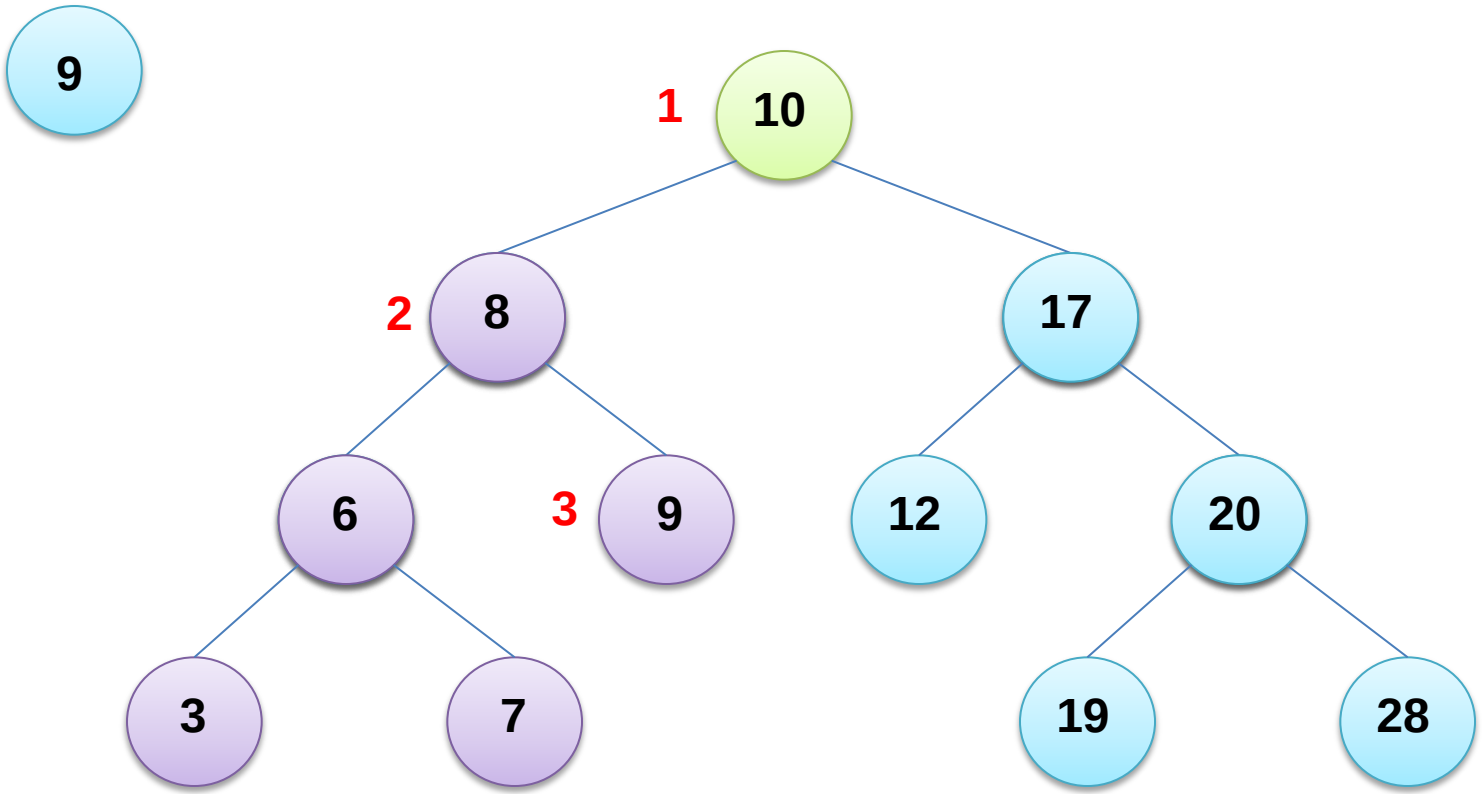
- создать указатель на корень дерева
- переходить в правого сына до тех пор, пока указатель на правого сына не nullptr



Поиск элемента (рекурсивный)

- сравнить переданное значение k со значением в корне дерева
- если значение k равно значению в корне, то вернуть корень
- если значение k меньше, чем значение в корне, то вызвать функцию поиска для левого поддерева
- если значение k больше, чем значение в корне, то вызвать функцию поиска для правого поддерева

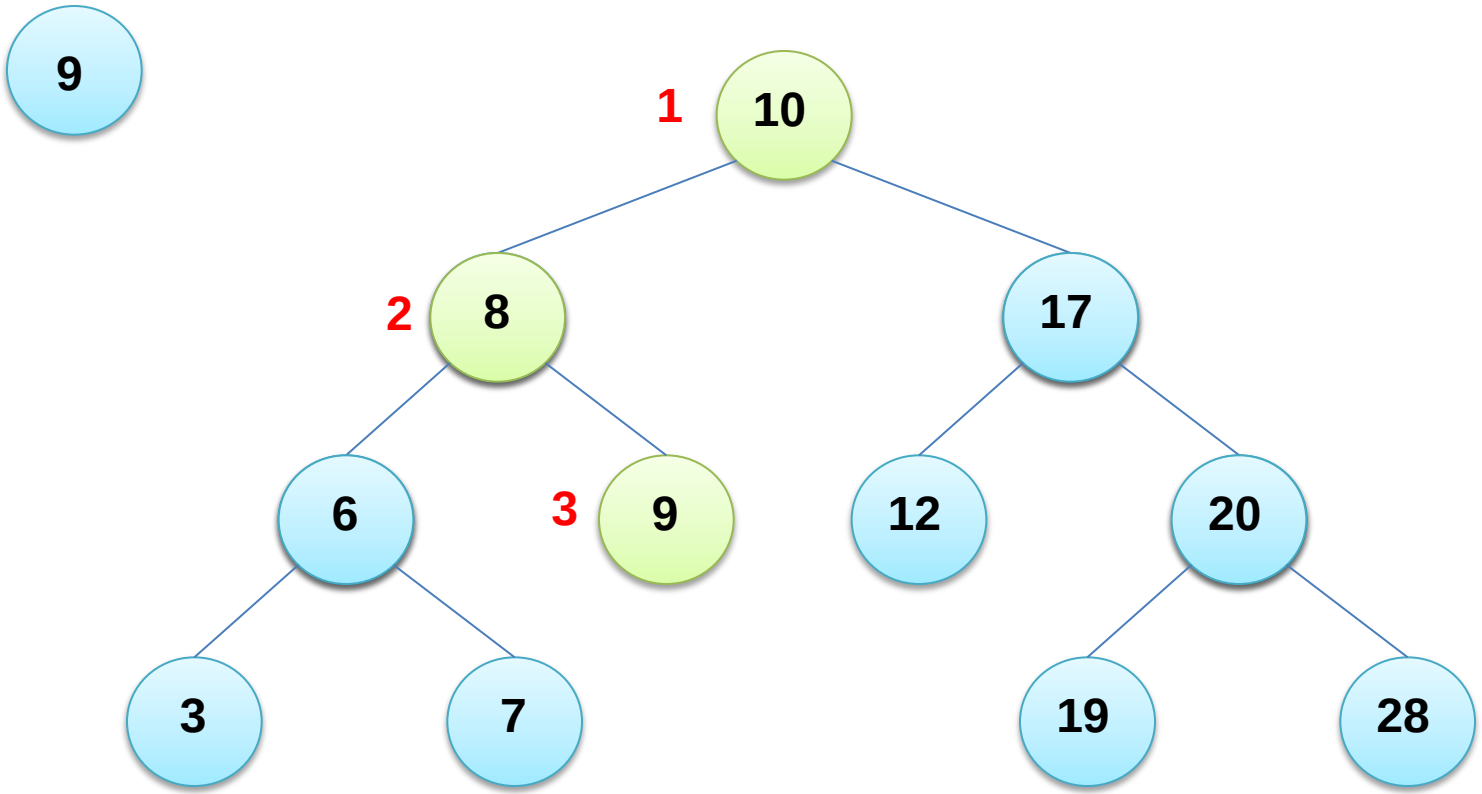
Поиск элемента (схема)



Поиск элемента (нерекурсивный)

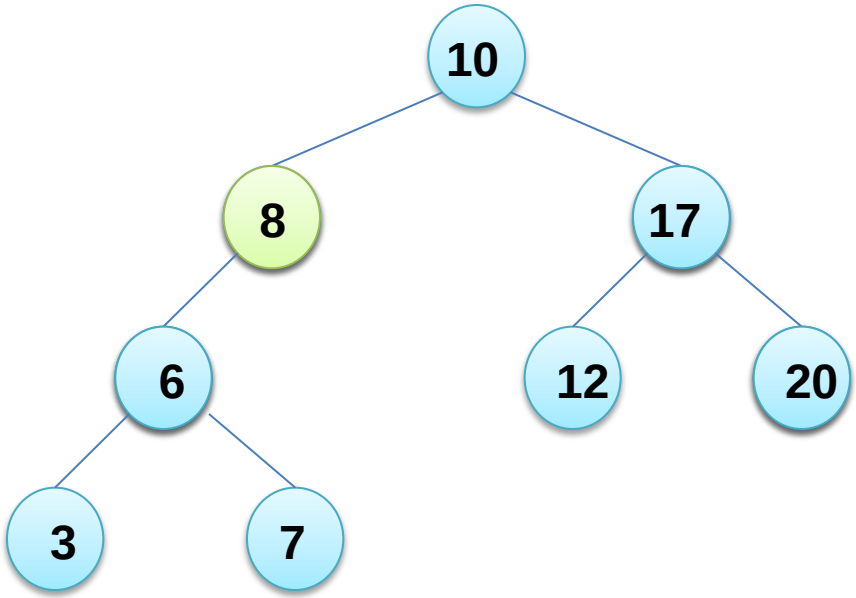
- создать указатель на текущий узел, проинициализировать его корнем дерева
- пока текущий узел не nullptr
  - сравнить переданное значение k со значением в текущем узле
  - если значение k равно значению в текущем узле, то вернуть текущий узел
  - если значение k меньше, чем значение в текущем узле, то перейти в левого сына
  - если значение k больше, чем значение в текущем узле, то перейти в правого сына

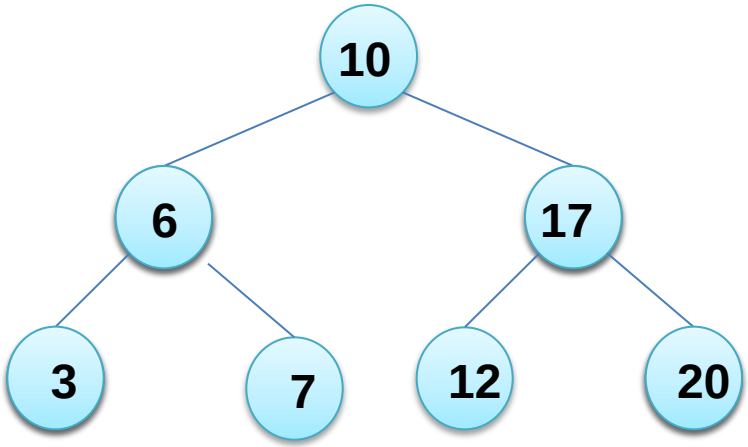
Поиск элемента (схема)



Удаление элемента (1)

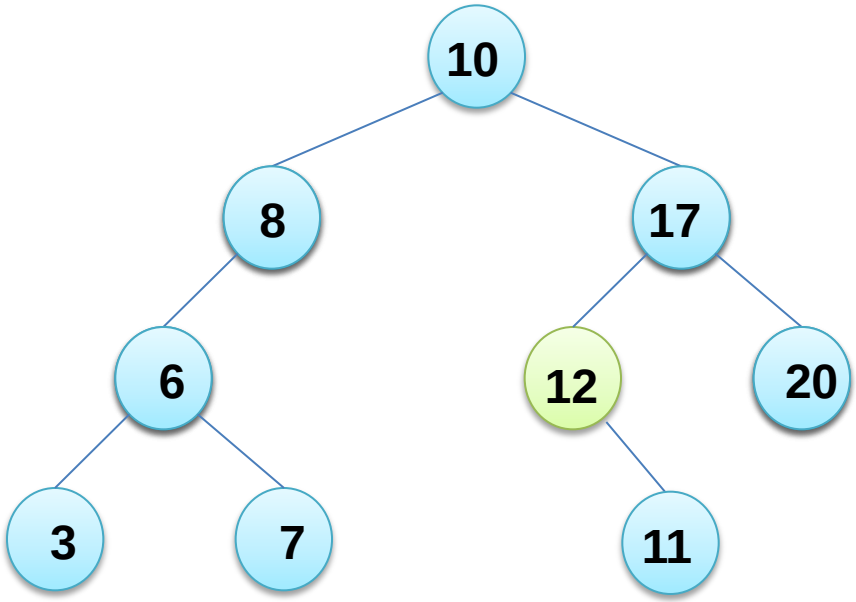
- найти узел с удаляемым значением
- если у удаляемого узла нет правого поддерева, перенести левое поддерево выше

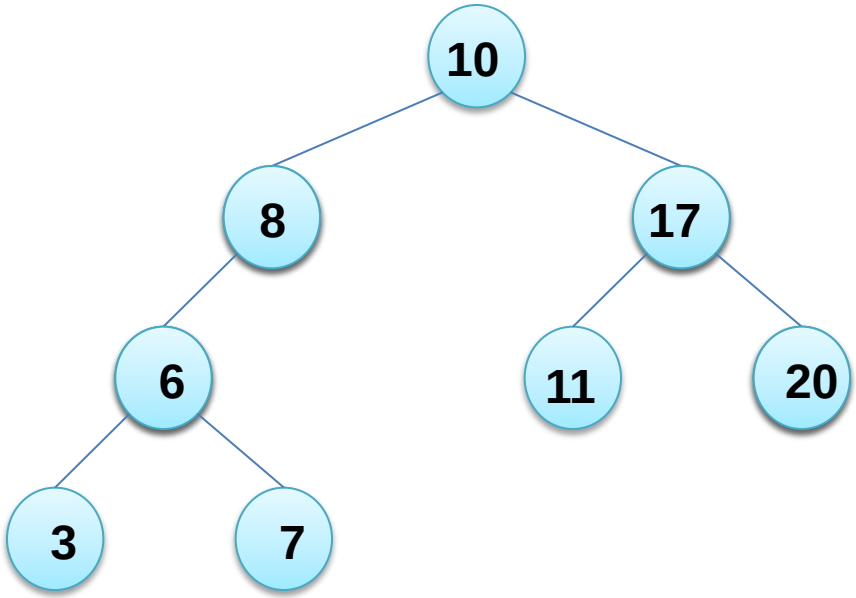




Удаление элемента (2)

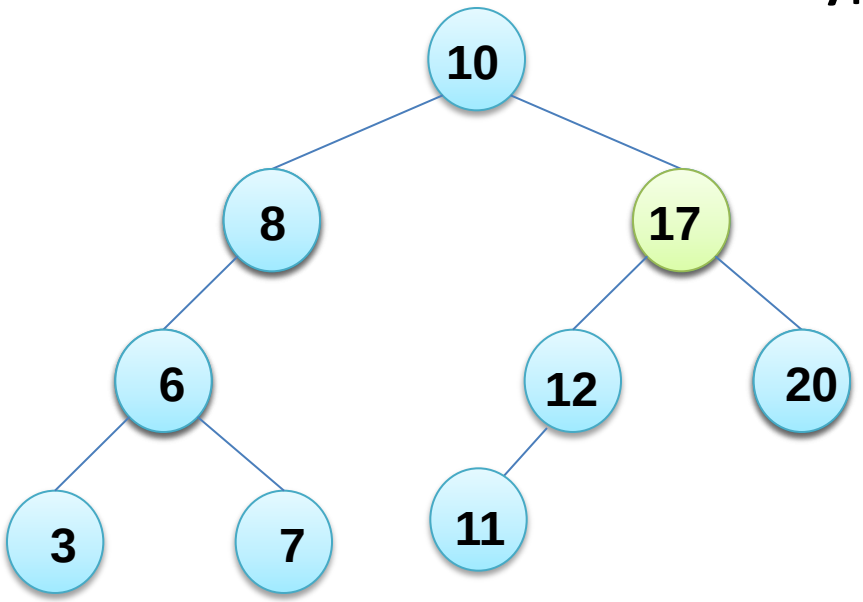
- найти узел с удаляемым значением
- если у удаляемого узла нет левого поддерева, перенести правое поддерево выше

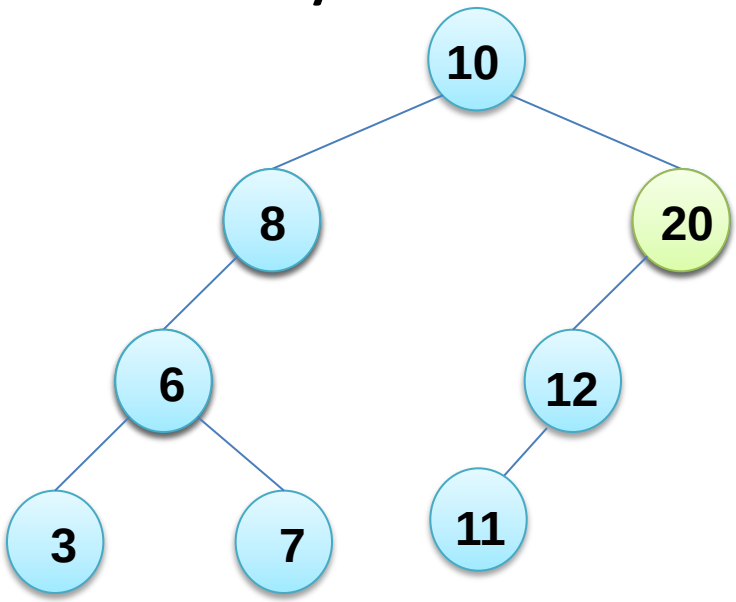




Удаление элемента (3)

- найти узел с удаляемым значением
- если у удаляемого узла есть и левый, и правый сын, то найти в правом поддереве минимум и вставить на место удаляемого узла





Вопросы?