ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Департамент программной инженерии

Алгоритмы и структуры данных

Семинар №7. 2021-2022 учебный год

Нестеров Роман Александрович, *ДПИ ФКН и НУЛ ПОИС*

Бессмертный Александр Игоревич, *ДПИ ФКН*

Higher School of Economics

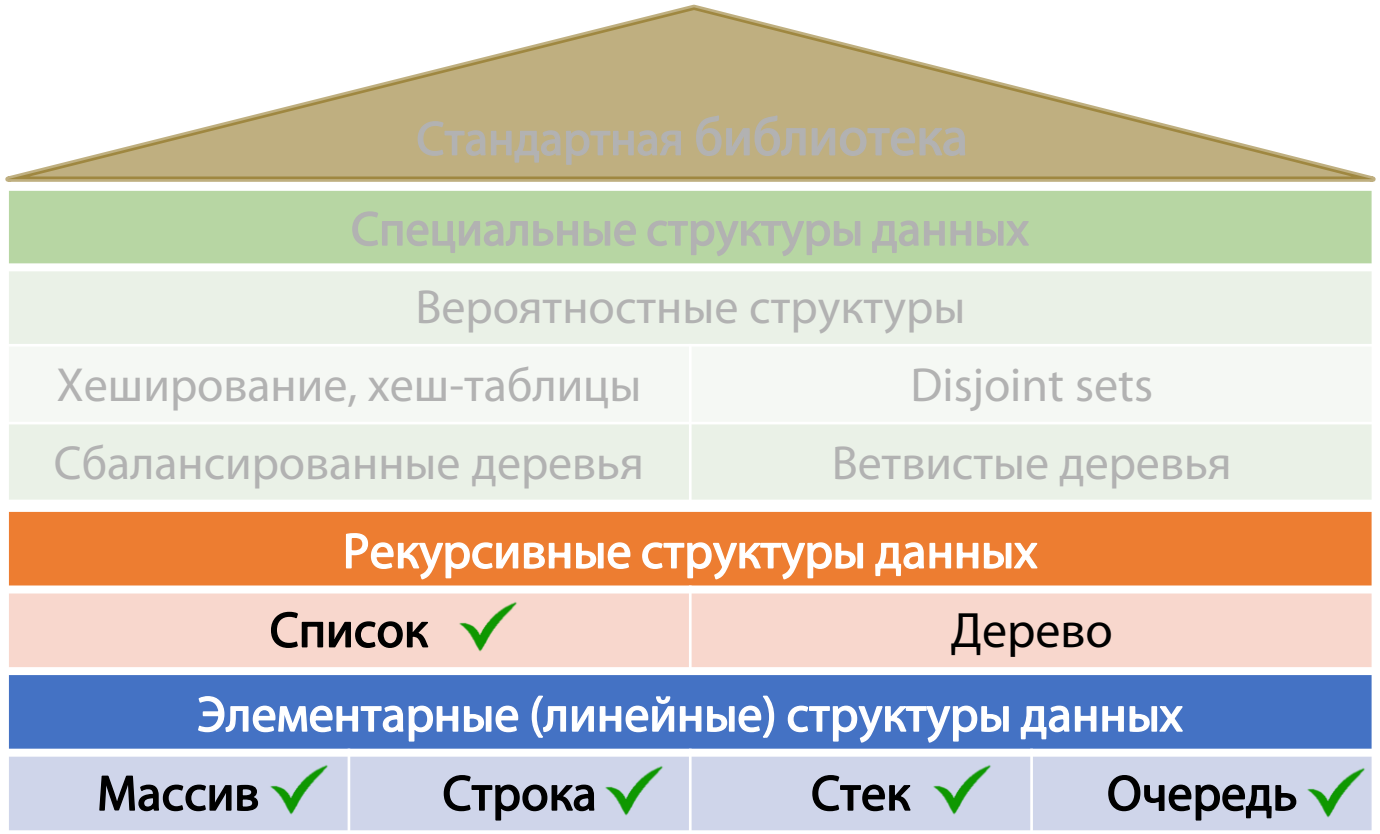
Faculty of Computer science

Chaque chose en son temps

Где мы?


Стандартная библиотека
Специальные структуры данных
Вероятностные структуры
Хеширование, хеш-таблицы
Disjoint sets
Сбалансированные деревья
Ветвистые деревья
Рекурсивные структуры данных
Список
Дерево
Элементарные (линейные) структуры данных
Массив
Строка
Стек
Очередь

Где мы?


Стандартная библиотека
Специальные структуры данных
Вероятностные структуры
Хеширование, хеш-таблицы
Disjoint sets
Сбалансированные деревья
Ветвистые деревья
Рекурсивные структуры данных
Список ✓
Дерево
Элементарные (линейные) структуры данных
Массив ✓
Строка ✓
Стек ✓
Очередь ✓

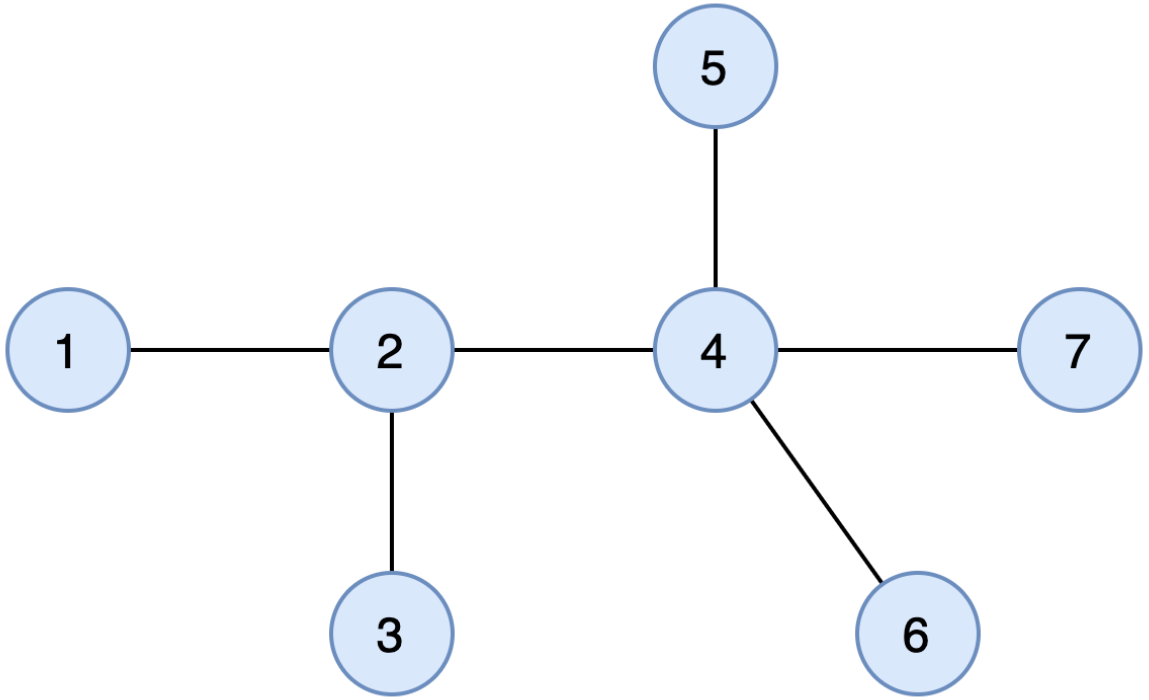
План

- Дерево и его свойства
- Бинарное дерево и синтаксический разбор
- Обход дерева: прямой, обратный, симметричный, в ширину
- Бинарное дерево поиска
- Балансировка: поворот дерева
- Реализация

Деревья и их свойства

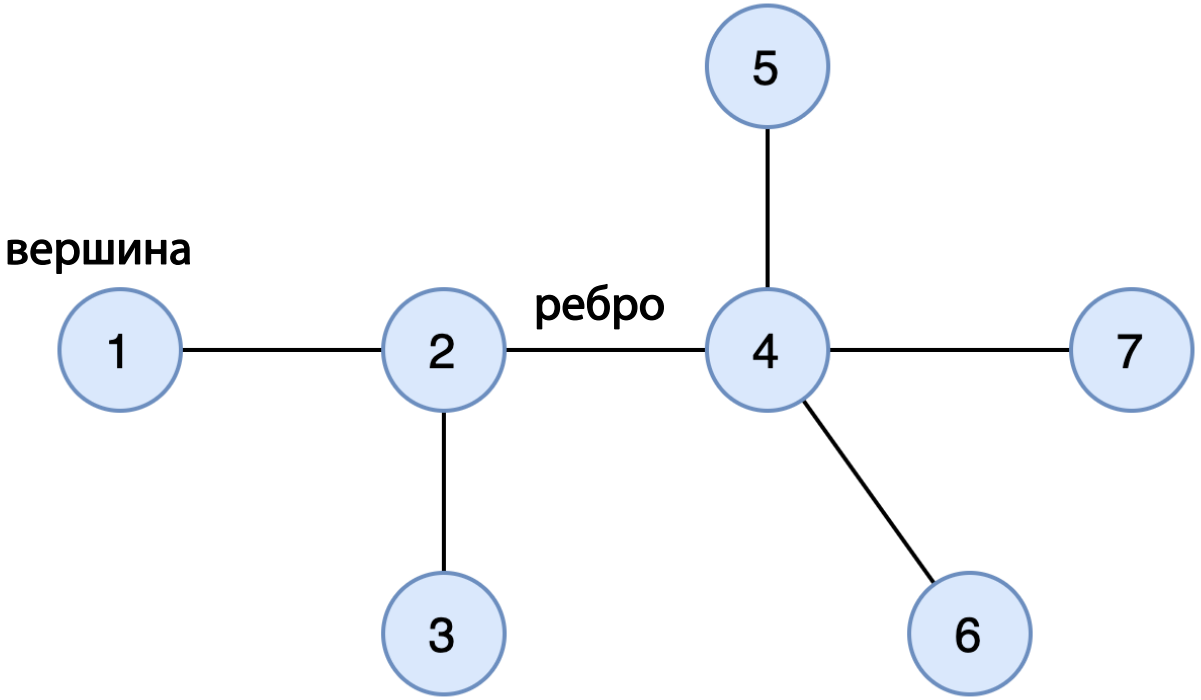
Дерево

Односвязный граф **без циклов**



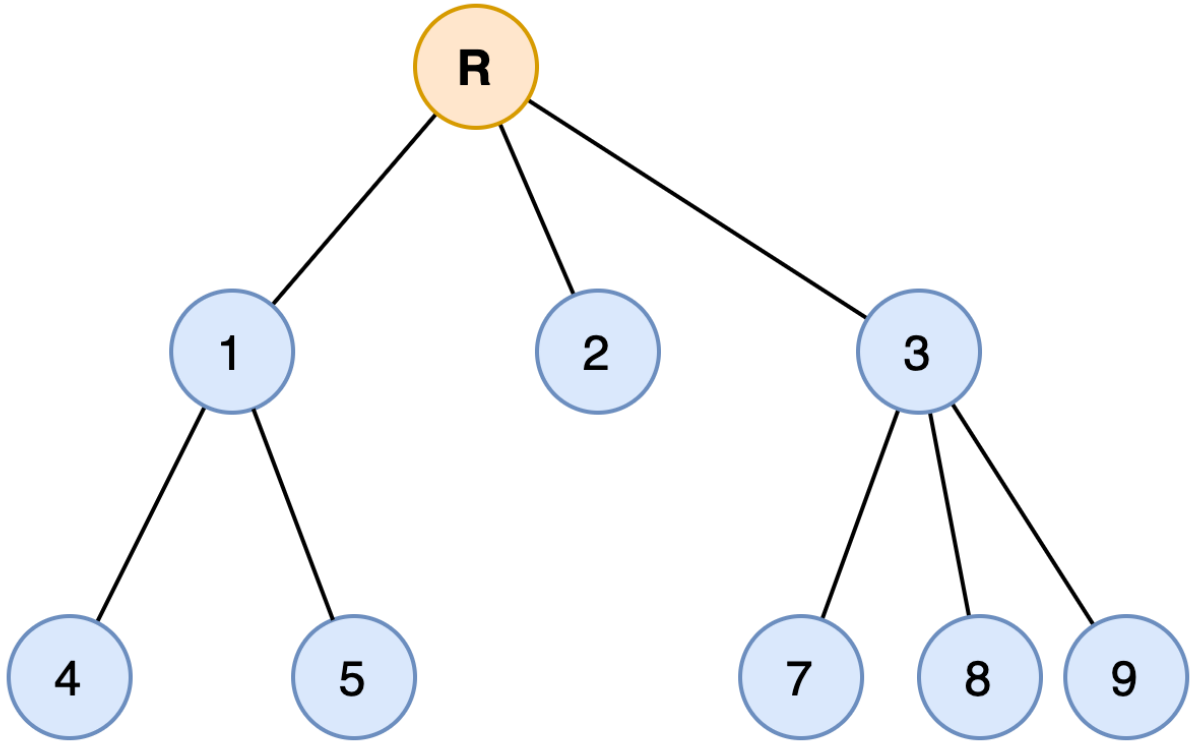
Дерево

Односвязный граф **без циклов**



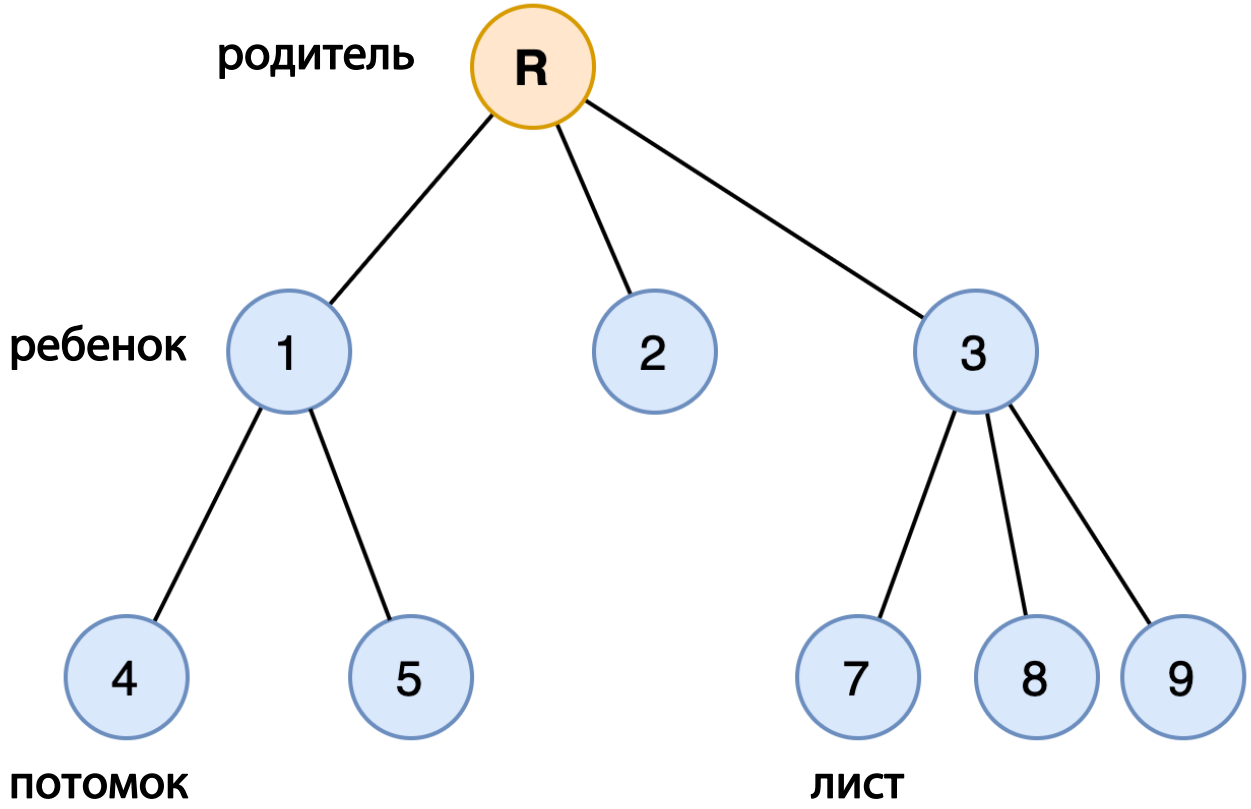
Корневое (rooted) дерево

Дерево с выделенной **корневой** вершиной



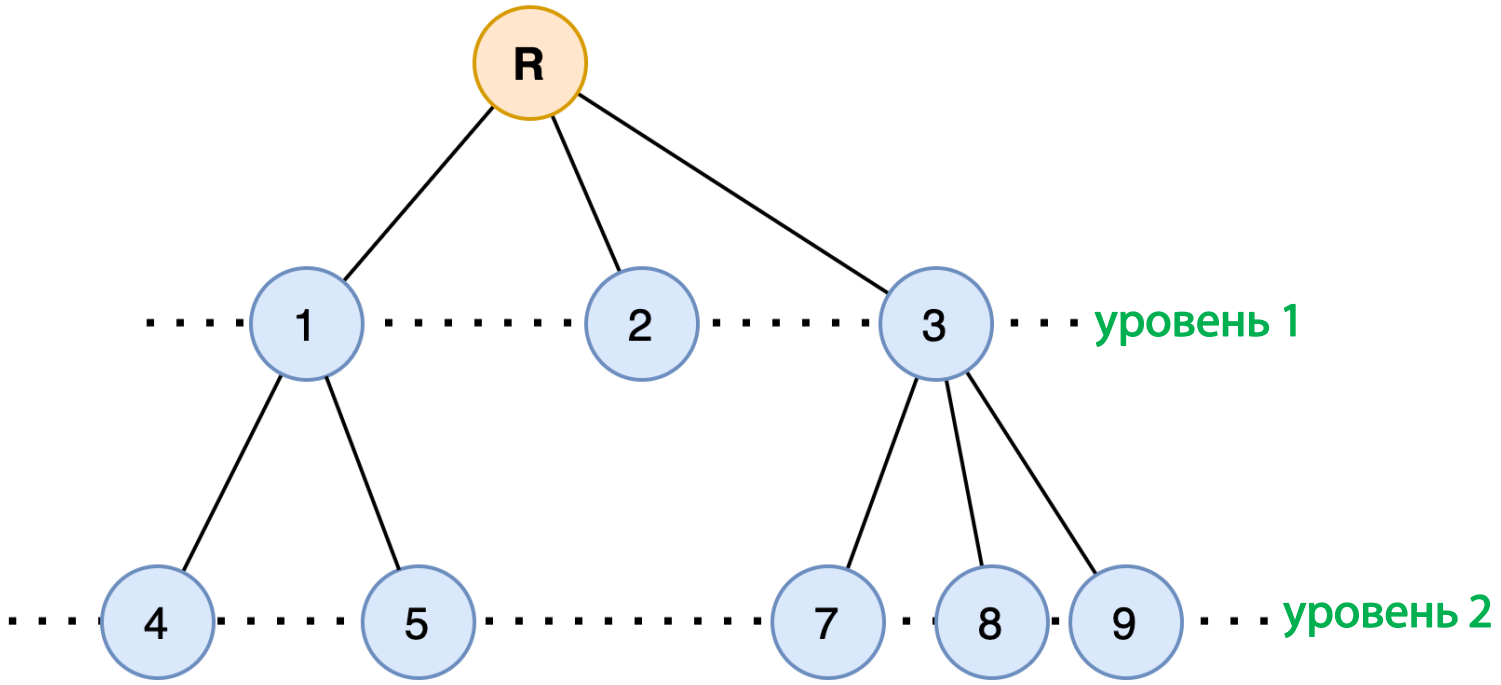
Корневое (rooted) дерево

Дерево с выделенной **корневой** вершиной



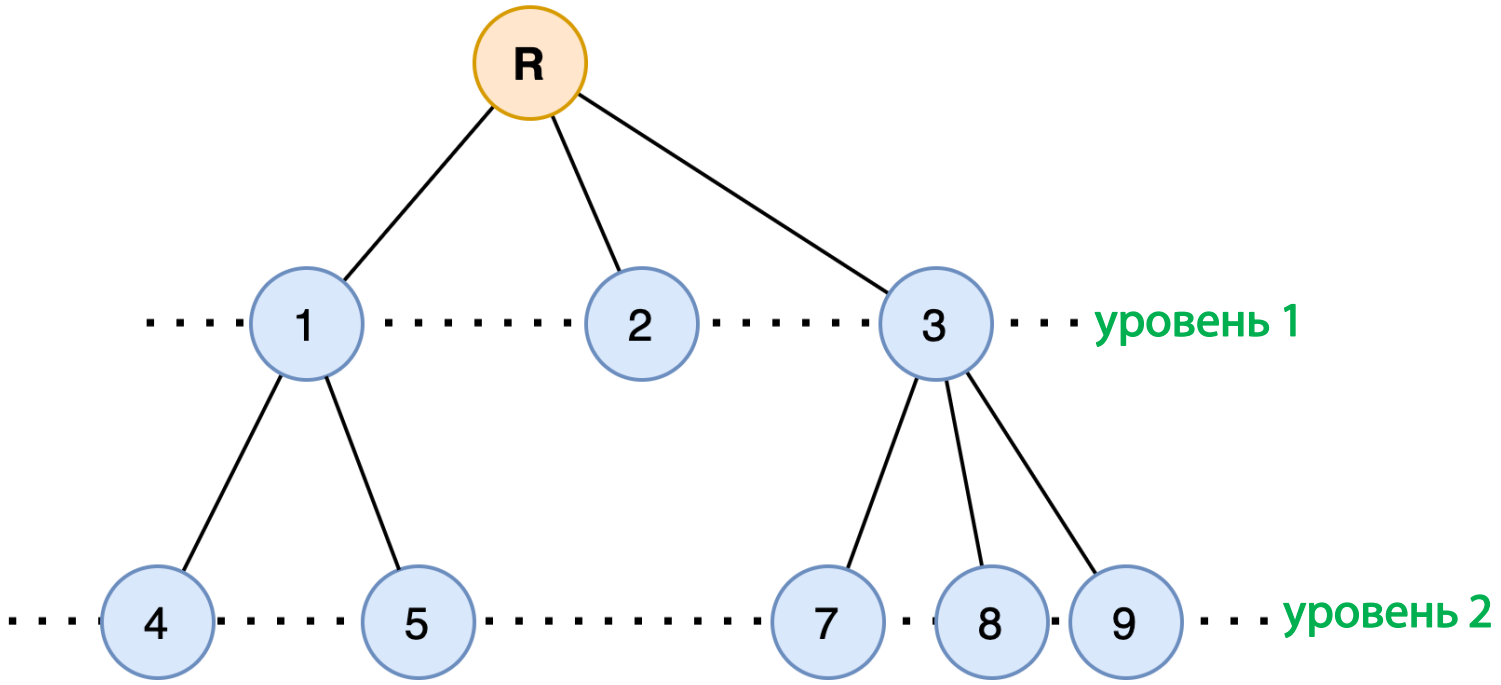
Корневое (rooted) дерево

Дерево с выделенной **корневой** вершиной



Корневое (rooted) дерево

Дерево с выделенной **корневой** вершиной

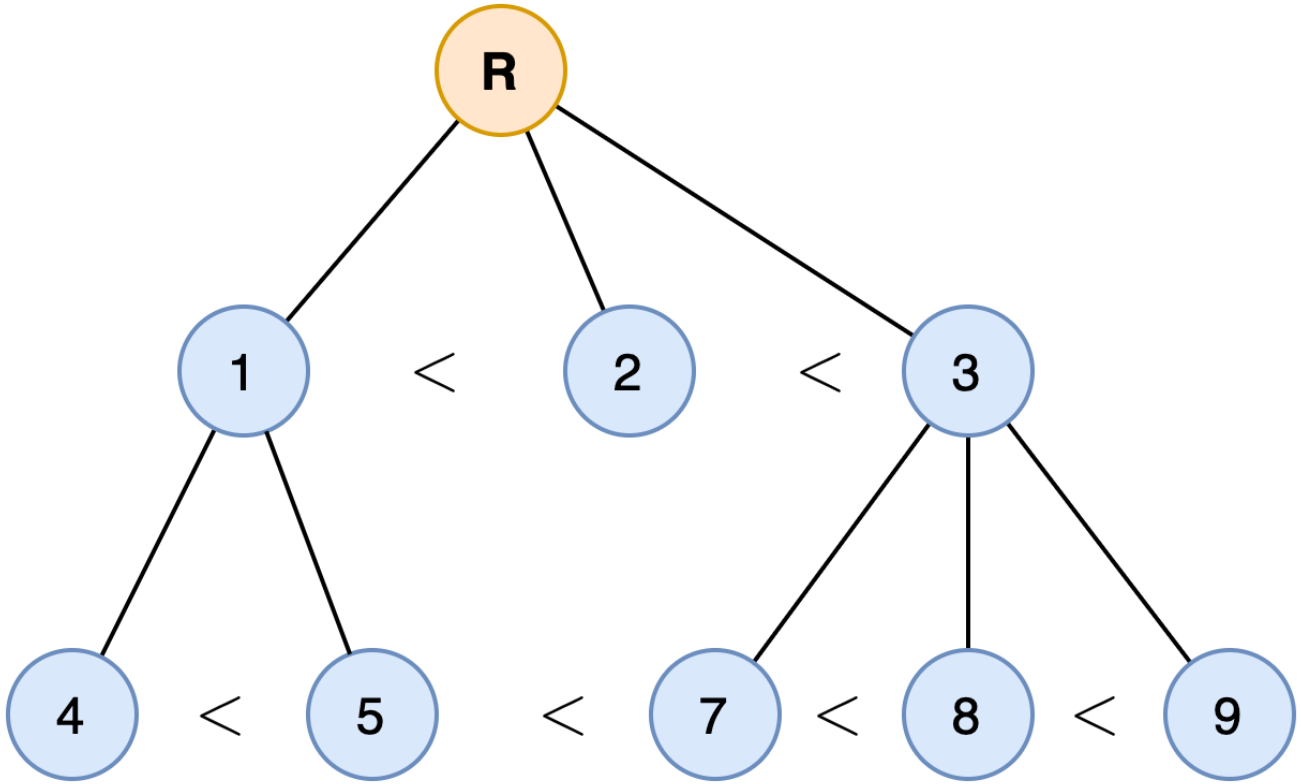


Высота дерева

- Самый длинный путь из корня
- Наибольший уровень

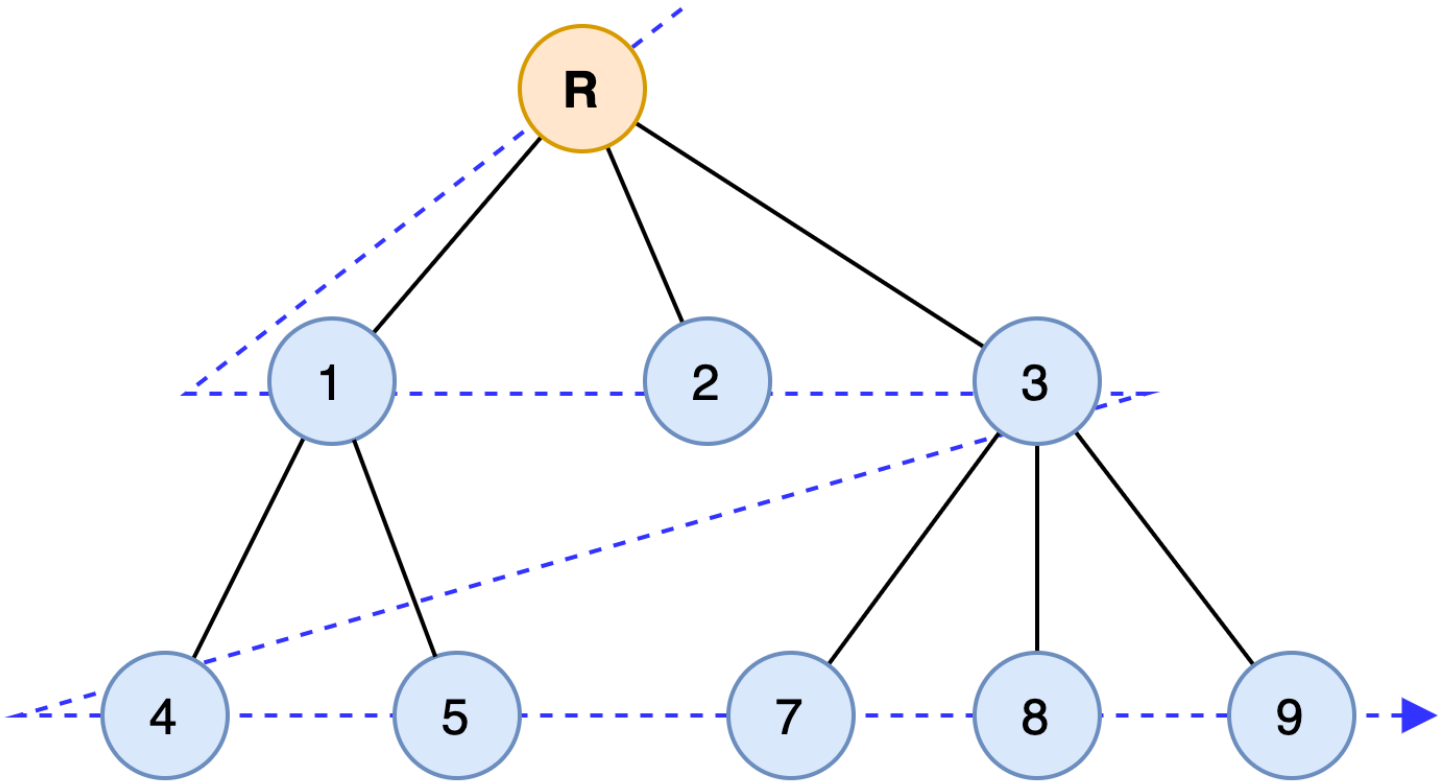
Упорядоченное дерево

На каждом уровне дерева задан некоторый порядок вершин



Упорядоченное дерево

На каждом уровне дерева задан некоторый порядок вершин

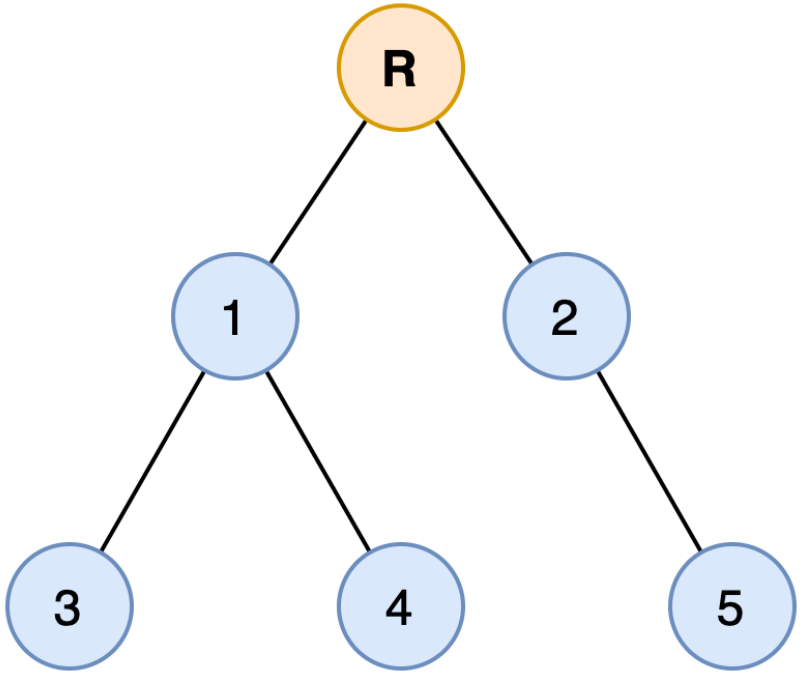


Обход дерева в ширину
(**Level order traversal**)

Бинарное дерево

Бинарное дерево

Дерево, в котором каждая вершина имеет не более **двух потомков**

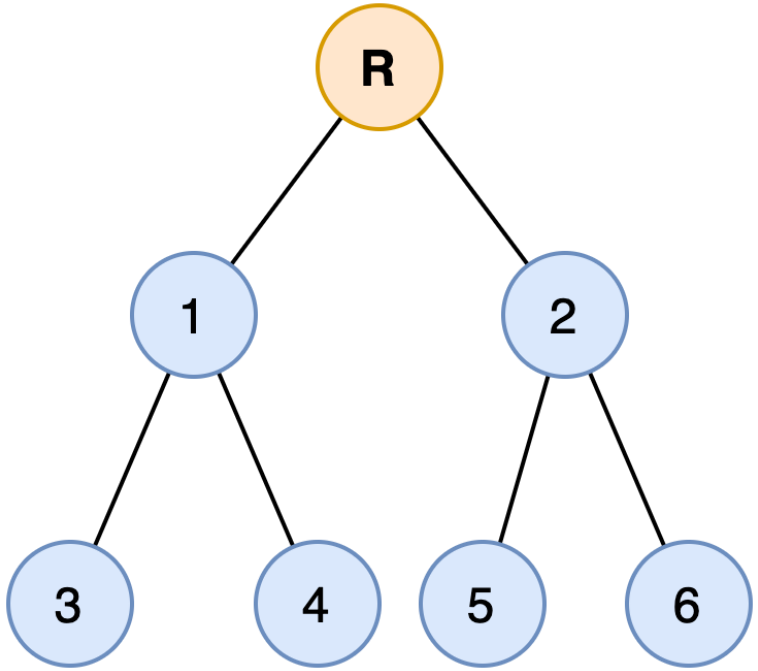



```
template <class T>
class Tree {
    T data;
    Tree *left;
    Tree *right;

    Tree();
    …
}
```

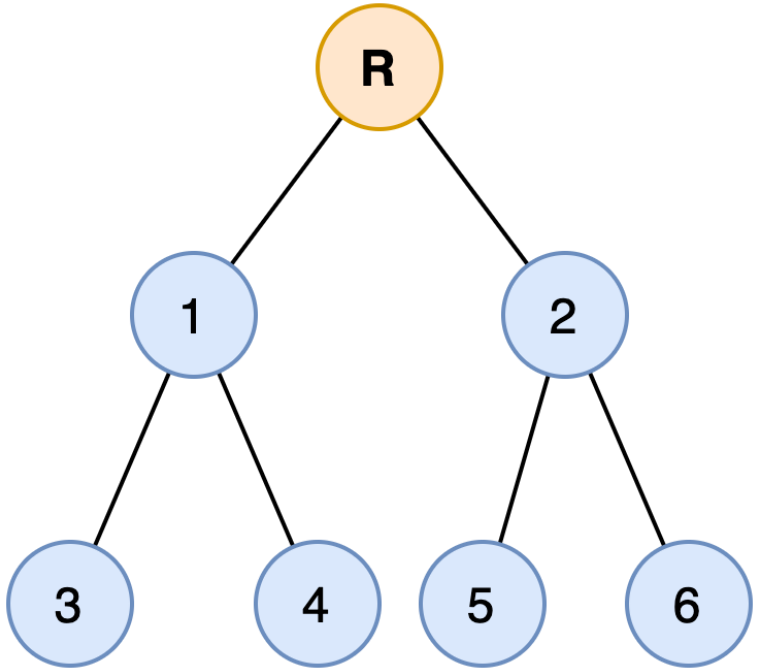
Идеальное (perfect) бинарное дерево

- Бинарное дерево, в котором каждая вершина (кроме листьев) имеет двух потомков.
- Все листья находятся на **одном уровне.**



Идеальное (perfect) бинарное дерево

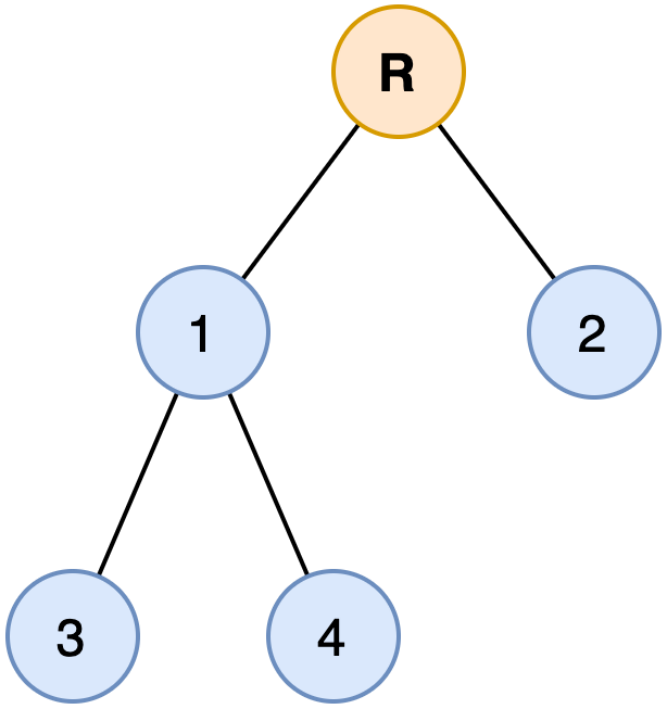
- Бинарное дерево, в котором каждая вершина (кроме листьев) имеет двух потомков.
- Все листья находятся на **одном уровне.**



Сколько вершин имеет идеальное дерево высоты h?

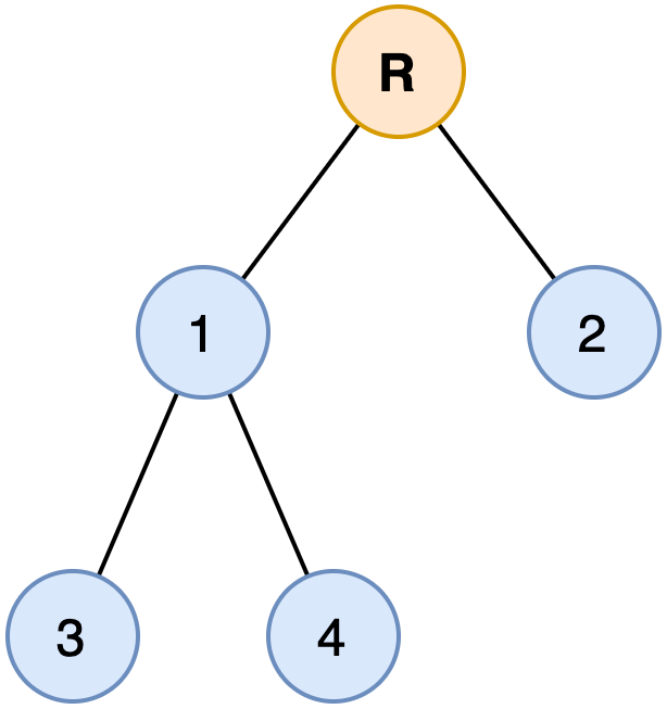
Строгое (full) бинарное дерево

Бинарное дерево, в котором каждая вершина (кроме листьев) имеет двух потомков.



Строгое (full) бинарное дерево

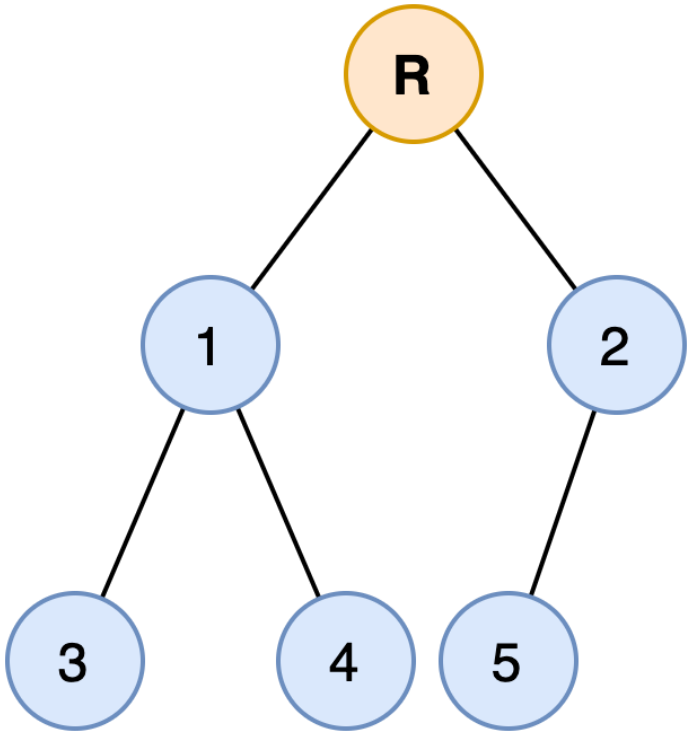
Бинарное дерево, в котором каждая вершина (кроме листьев) имеет двух потомков.



Сколько листьев в строгом бинарном дереве, если известно количество вершин с потомками?

Полное (complete) бинарное дерево

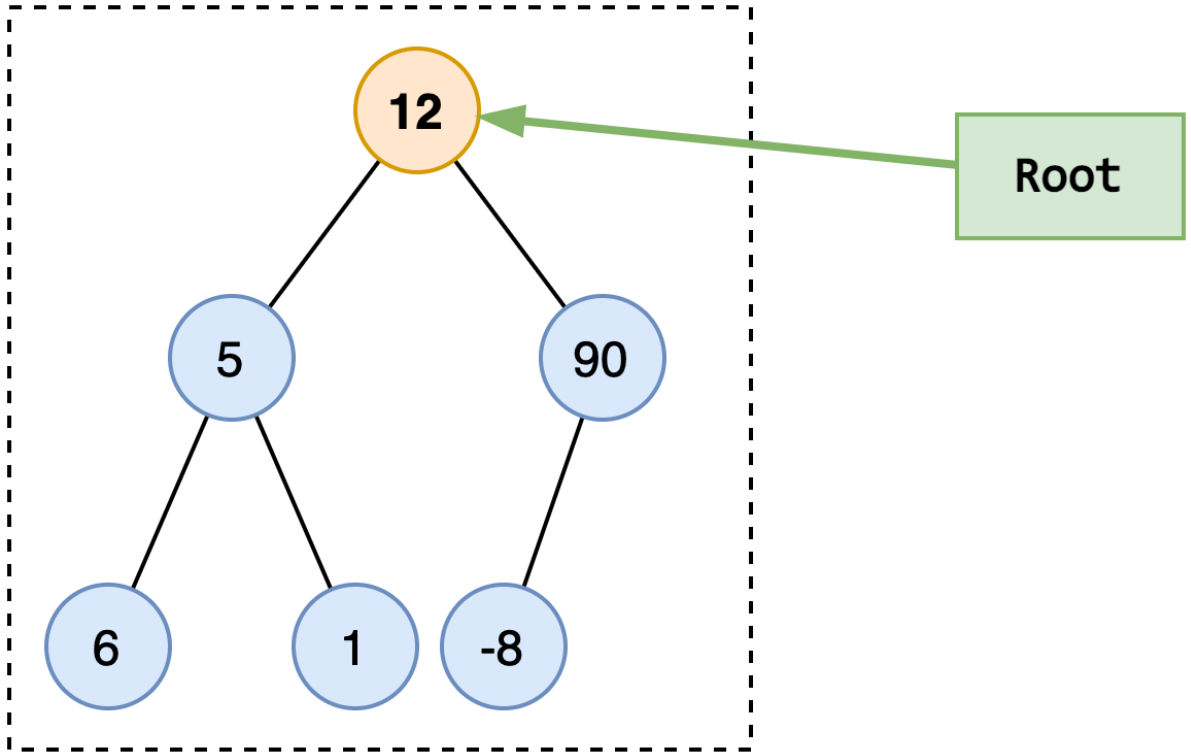
- Бинарное дерево, в котором все уровни (кроме, м.б., последнего) заполнены
- Последний уровень заполнен **слева направо**



Бинарное дерево. Вставка узла

Вершины упорядочены на каждом уровне слева направо.

Вставить узел на первое свободное место.

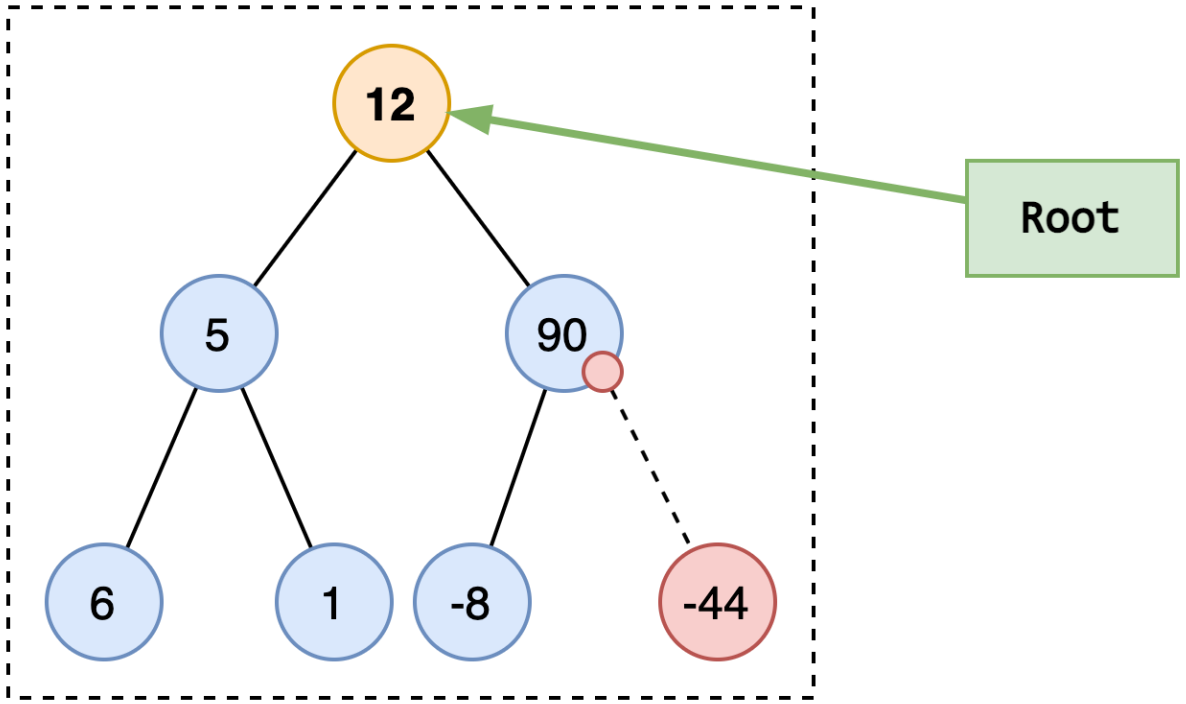



```
insert -44
```

Бинарное дерево. Вставка узла

Вершины упорядочены на каждом уровне **слева направо**.

Вставить узел на первое свободное место.



insert -44

Обходим дерево в **ширину** пока:

- Левый потомок не **nullptr** или
- Правый потомок не **nullptr**

Бинарное дерево. Обход в ширину

Потомки вершин помещаются в конец очереди, которая хранит указатели.

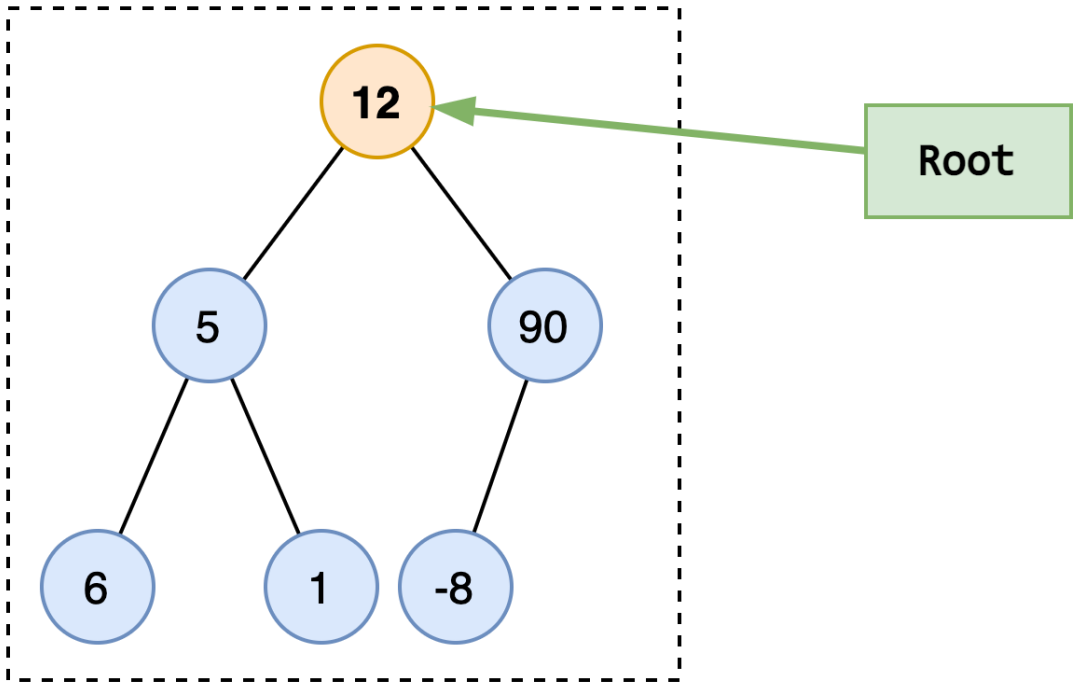
```
Queue q;
q.push(root);
Tree *curr;

while (!q.isEmpty()) {
    curr = q.front();
    q.pop();

    if (curr->left) { q.push(curr->left) }
    if (curr->right) { q.push(curr->right) }
}
```

Бинарное дерево. Удаление узла

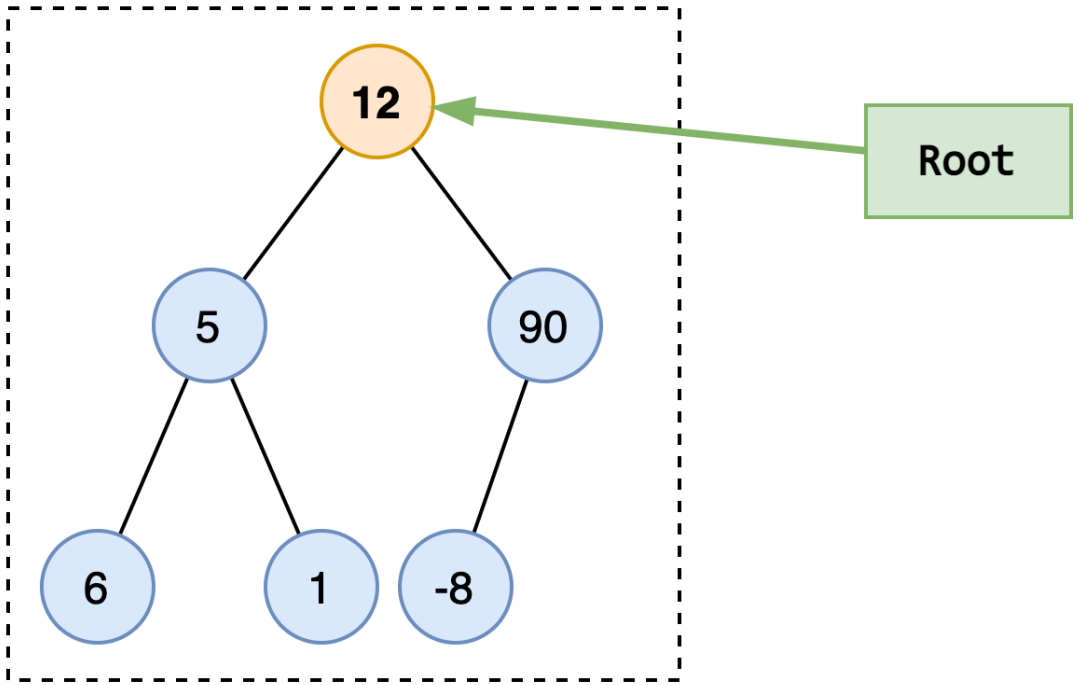
Удалить узел из заданного бинарного дерева по ключу.
Удаляемое значение заменить **самым правым значением последнего уровня**.




```
remove 5
```

Бинарное дерево. Удаление узла

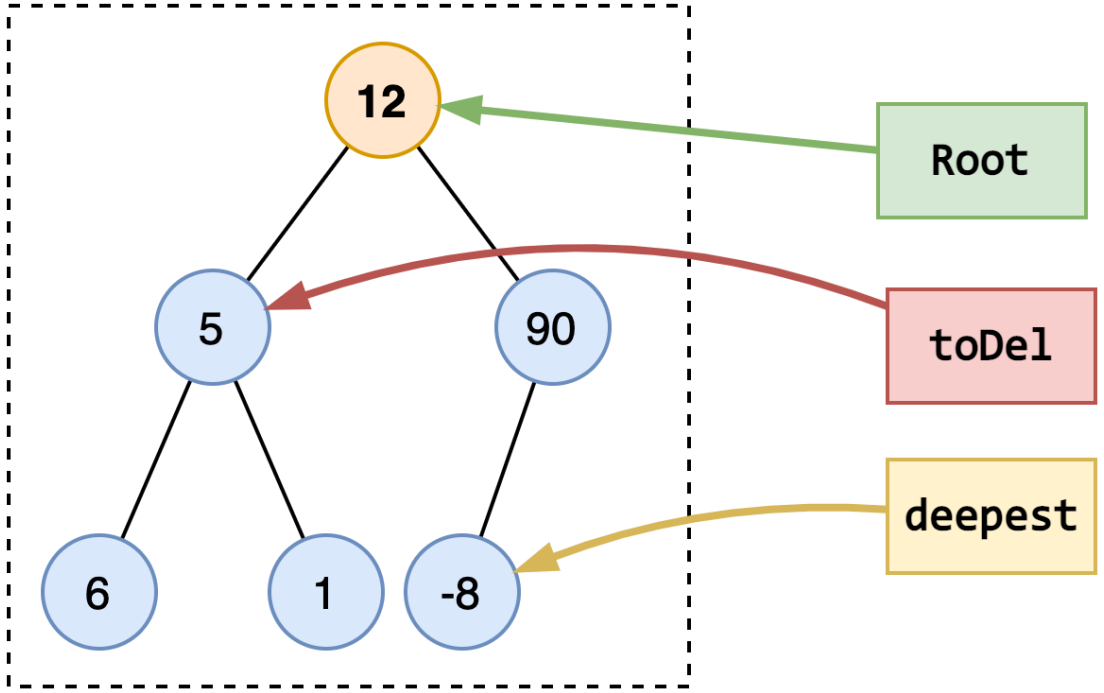
Удалить узел из заданного бинарного дерева по ключу.
Удаляемое значение заменить **самым правым значением последнего уровня**.




```
remove 5
```

Бинарное дерево. Удаление узла

Удалить узел из заданного бинарного дерева по ключу.
Удаляемое значение заменить **самым правым значением последнего уровня**.

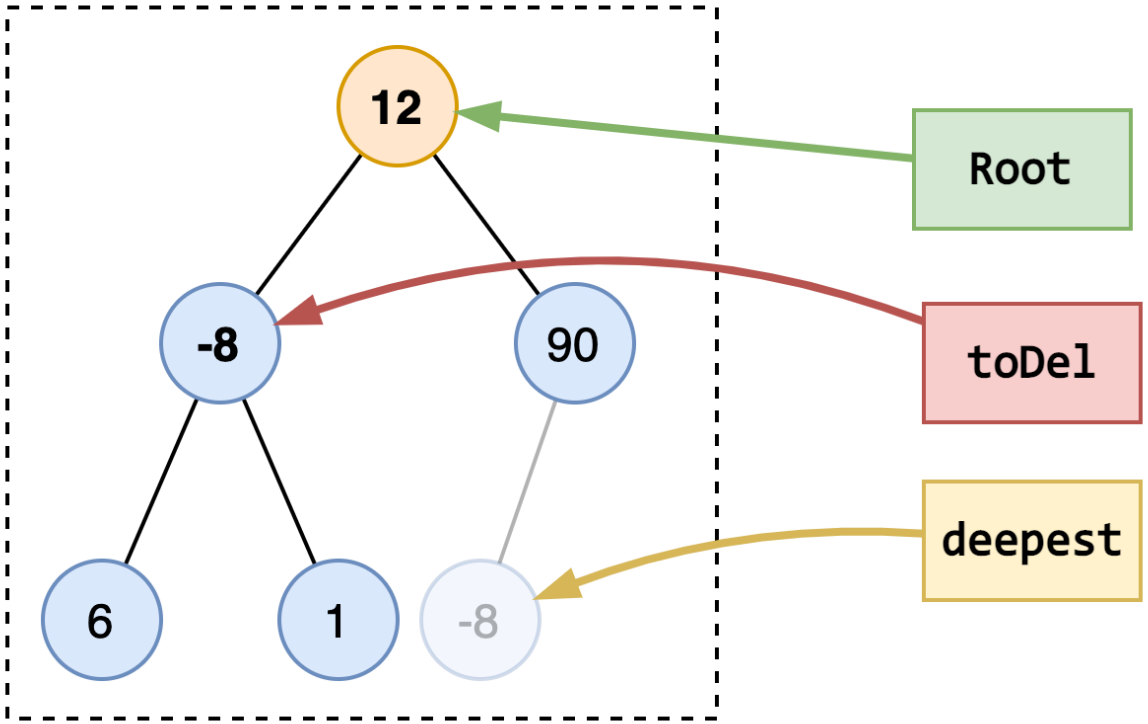


remove 5

```
toDel -> data = deepest -> data;
delete deepest;
обнулить указатель у предка -8;
```

Бинарное дерево. Удаление узла

Удалить узел из заданного бинарного дерева по ключу.
Удаляемое значение заменить **самым правым значением последнего уровня**.



remove 5

```
toDel -> data = deepest -> data;
delete deepest;
обнулить указатель у предка -8;
```

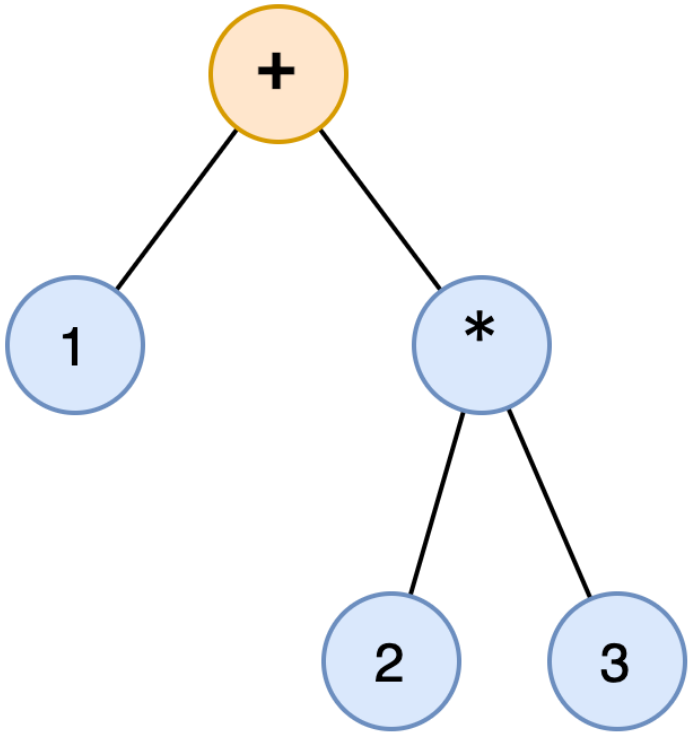
Синтаксический разбор выражения

Дерево синтаксического разбора

Задано арифметическое выражение.

Построить дерево его синтаксического разбора.

```
1 + 2 * 3
```



Дерево синтаксического разбора

Задано арифметическое выражение.

Построить дерево его синтаксического разбора.

(	3	+	(	4	*	5	)	)

- **Скобка ‘(’** – создать левый потомок у текущего узла
- **Оператор** – записать его в поле `data` текущего узла и добавить правого потомка
- **Число** – записать его в поле `data` текущего узла и вернуться к родителю
- **Скобка ‘)’** – вернуться к родителю текущего узла

Дерево синтаксического разбора

Задано арифметическое выражение.

Построить дерево его синтаксического разбора.

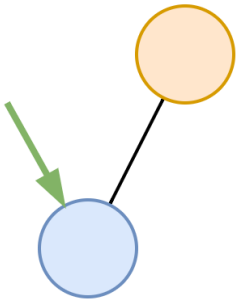
(	3	+	(	4	*	5	)	)

Дерево синтаксического разбора

Задано арифметическое выражение.

Построить дерево его синтаксического разбора.

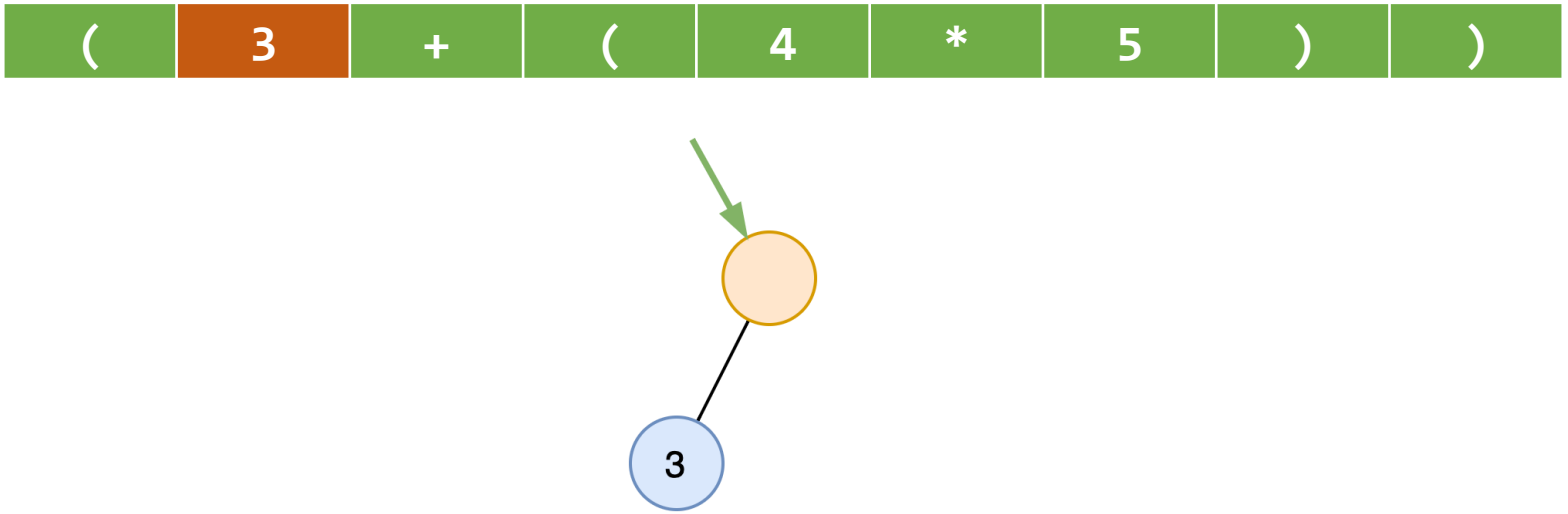
(	3	+	(	4	*	5	)	)



Дерево синтаксического разбора

Задано арифметическое выражение.

Построить дерево его синтаксического разбора.

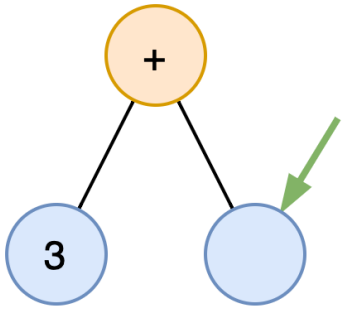


Дерево синтаксического разбора

Задано арифметическое выражение.

Построить дерево его синтаксического разбора.

(	3	+	(	4	*	5	)	)

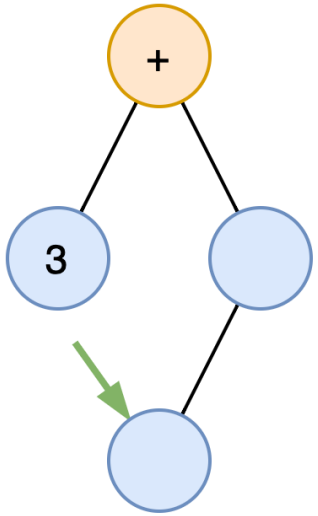


Дерево синтаксического разбора

Задано арифметическое выражение.

Построить дерево его синтаксического разбора.

(	3	+	(	4	*	5	)	)

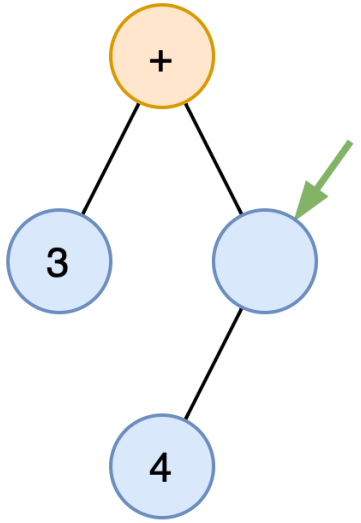


Дерево синтаксического разбора

Задано арифметическое выражение.

Построить дерево его синтаксического разбора.

(	3	+	(	4	*	5	)	)

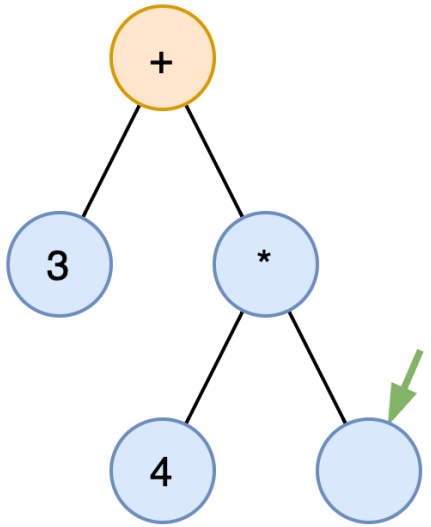


Дерево синтаксического разбора

Задано арифметическое выражение.

Построить дерево его синтаксического разбора.

(	3	+	(	4	*	5	)	)

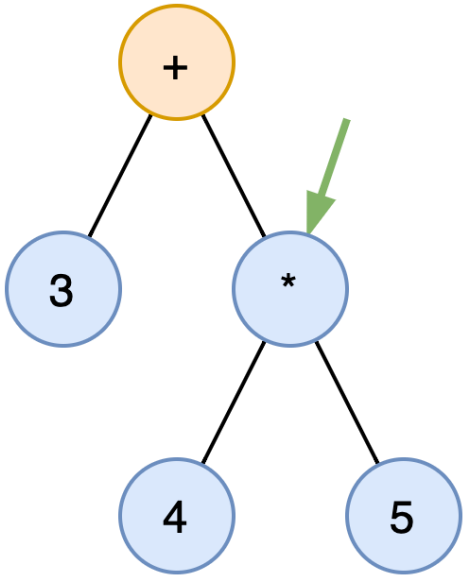


Дерево синтаксического разбора

Задано арифметическое выражение.

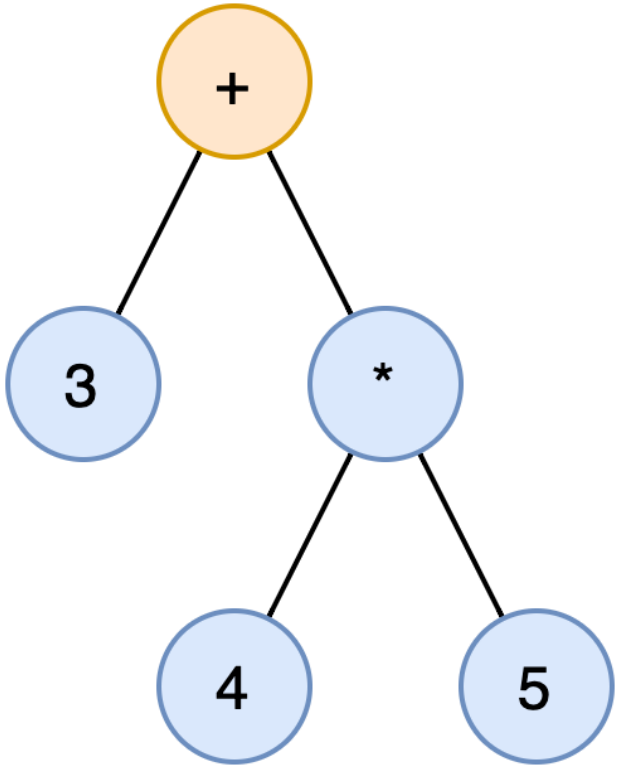
Построить дерево его синтаксического разбора.

(	3	+	(	4	*	5	)	)



Обход дерева разбора

Разные обходы дерева синтаксического разбора дают разные **формы записи** выражения.

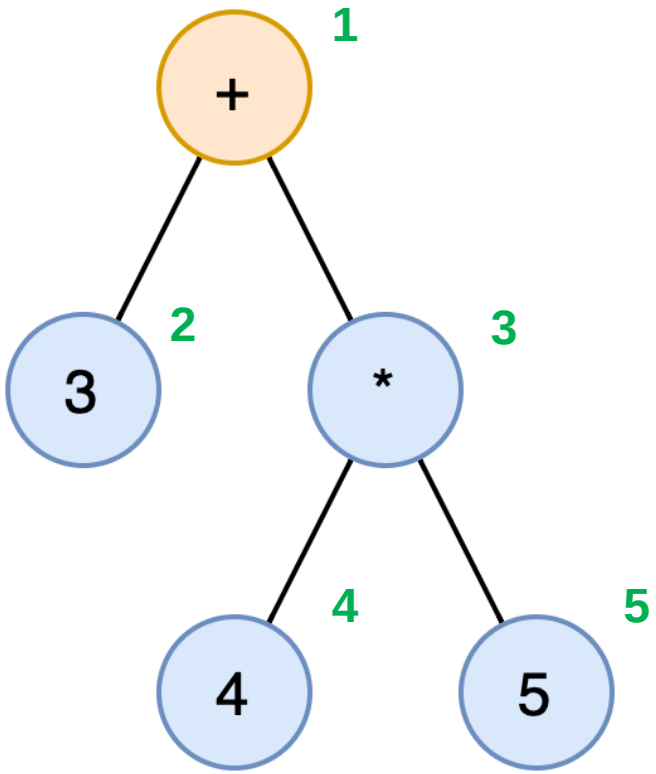


Прямой обход (**pre-order traversal**)

```
preOrder(Tree *root) {
    if (root) {
        std::cout << root -> data;
        preOrder(root -> left);
        preOrder(root -> right);
    }
}
```

Обход дерева разбора

Разные обходы дерева синтаксического разбора дают разные **формы записи** выражения.

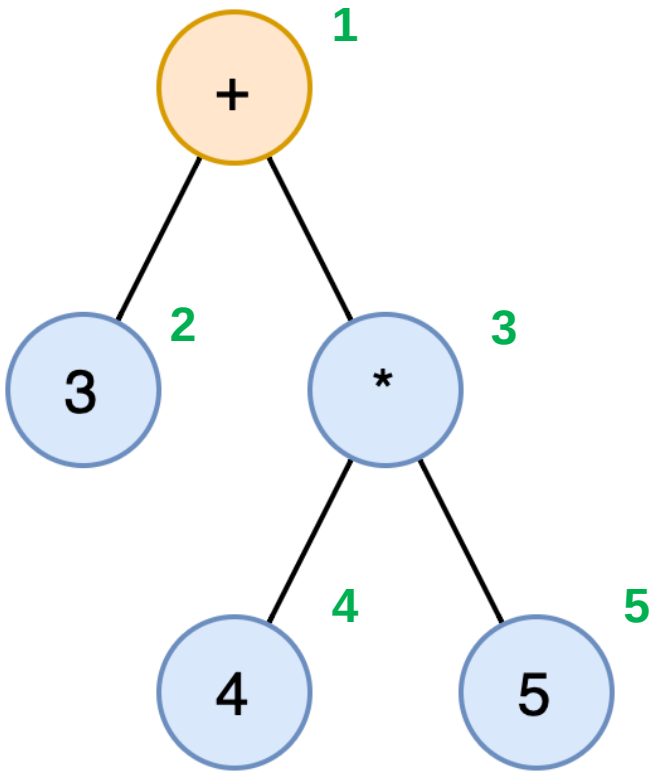


Прямой обход (**pre-order traversal**)

```
preOrder(Tree *root) {
    if (root) {
        std::cout << root -> data;
        preOrder(root -> left);
        preOrder(root -> right);
    }
}
```

Обход дерева разбора

Разные обходы дерева синтаксического разбора дают разные **формы записи** выражения.



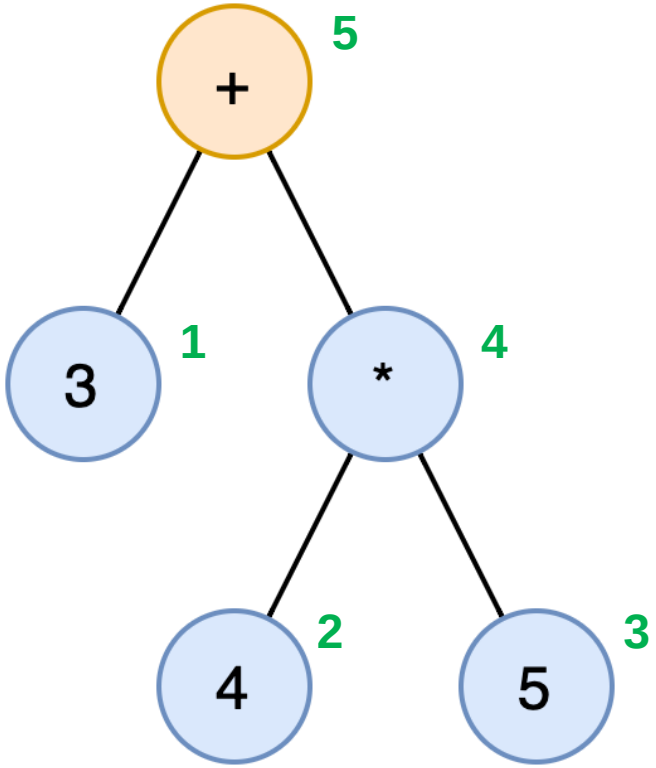
Прямой обход (**pre-order traversal**)

```
preOrder(Tree *root) {
    if (root) {
        std::cout << root -> data;
        preOrder(root -> left);
        preOrder(root -> right);
    }
}
```

+ 3 * 4 5

Обход дерева разбора

Разные обходы дерева синтаксического разбора дают разные **формы записи** выражения.



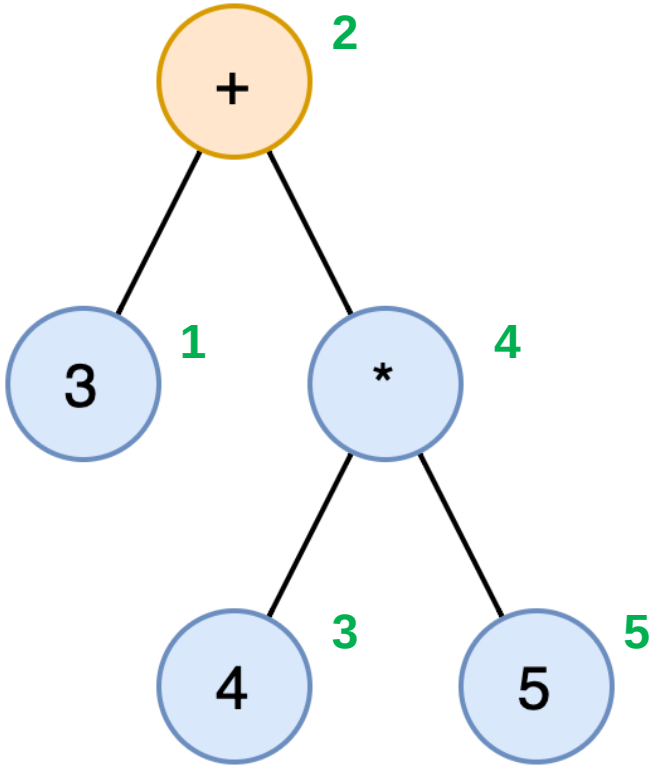
Обратный обход (**post-order traversal**)

```
postOrder(Tree *root) {
    if (root) {
        postOrder(root -> left);
        postOrder(root -> right);
        std::cout << root -> data;
    }
}
```

3 4 5 * +

Обход дерева разбора

Разные обходы дерева синтаксического разбора дают разные **формы записи** выражения.



Симметричный обход
(**in-order traversal**)

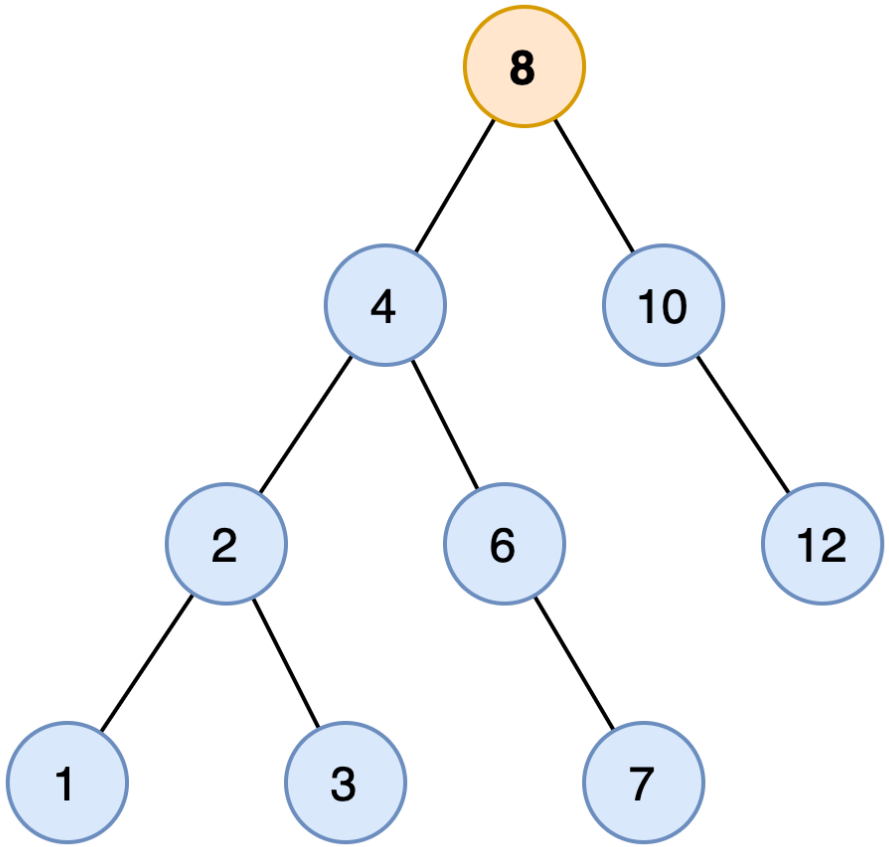
```
inOrder(Tree *root) {
    if (root) {
        inOrder(root -> left);
        std::cout << root -> data;
        inOrder(root -> right);
    }
}
```

3 + 4 * 5

Бинарное дерево поиска

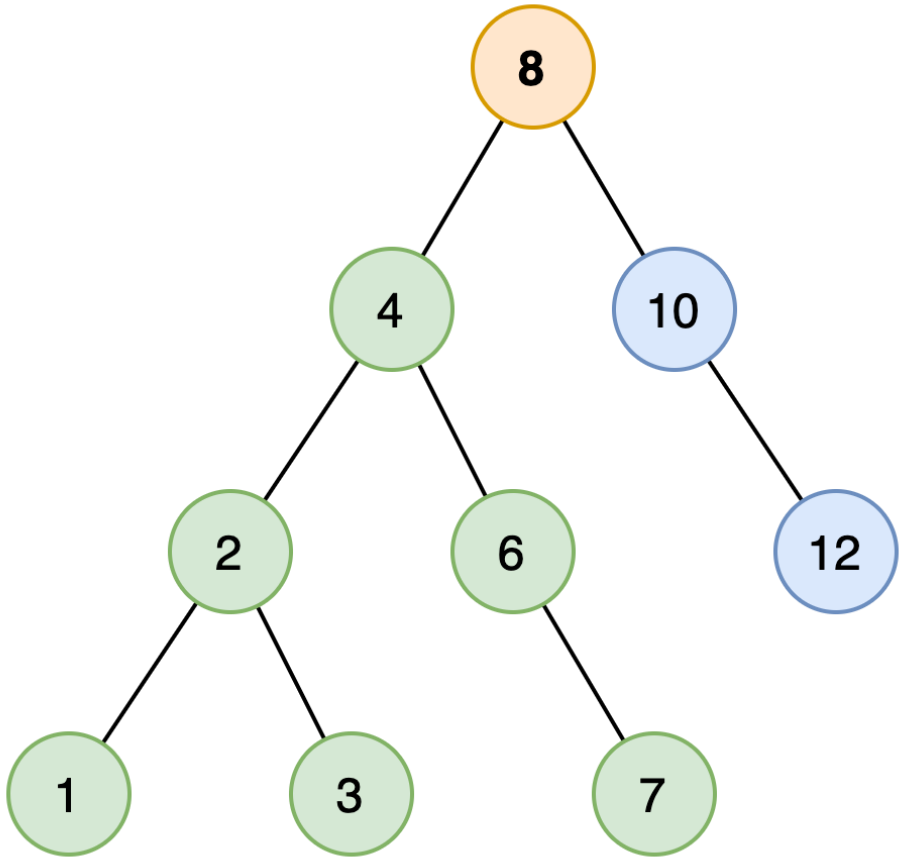
Бинарное дерево поиска (BST)

- Значение левого потомка < значение родителя
- Значение правого потомка >= значение родителя



Бинарное дерево поиска (BST)

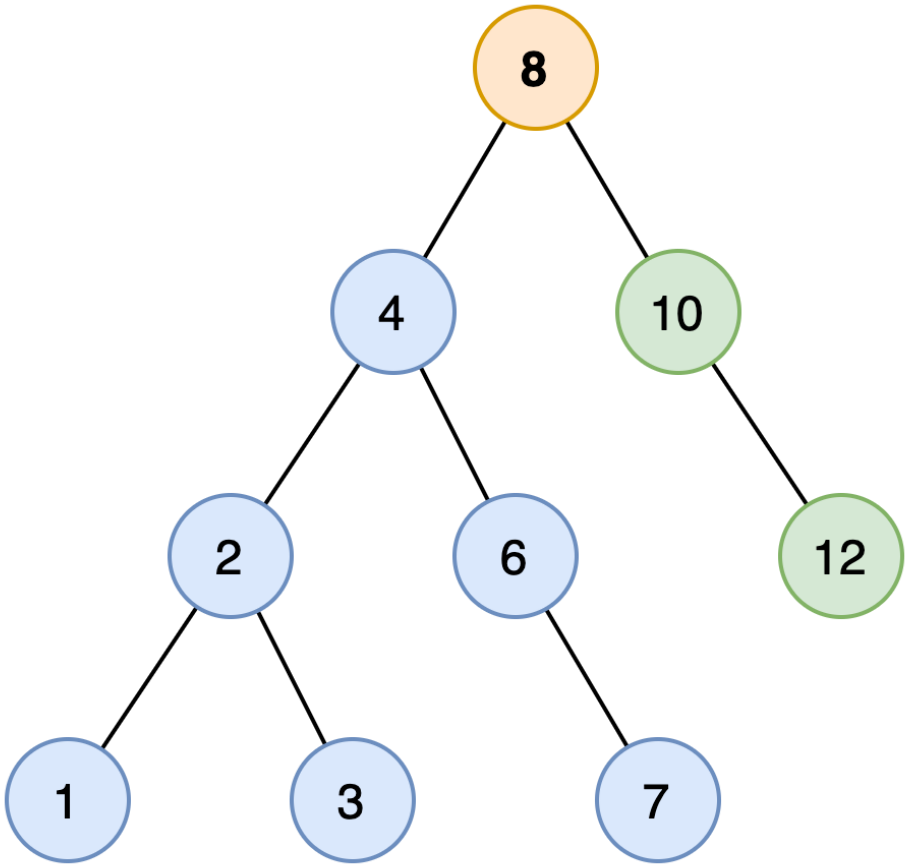
- Значение левого потомка < значение родителя
- Значение правого потомка >= значение родителя



1. Значения в **левом** поддереве некоторого узла **меньше** значения в этом узле

Бинарное дерево поиска (BST)

- Значение левого потомка < значение родителя
- Значение правого потомка >= значение родителя

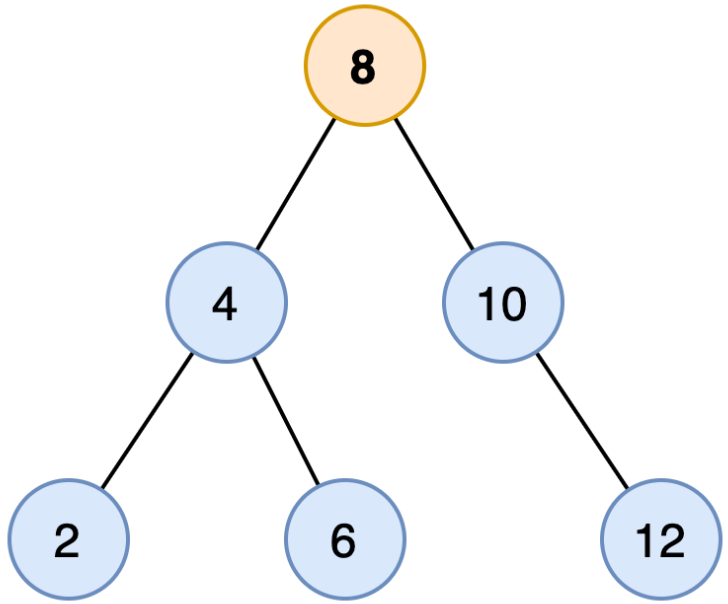


1. Значения в **левом** поддереве некоторого узла **меньше** значения в этом узле.

2. Значения в **правом** поддереве некоторого узла **больше** значения в этом узле.

Бинарное дерево поиска (BST). Вставка

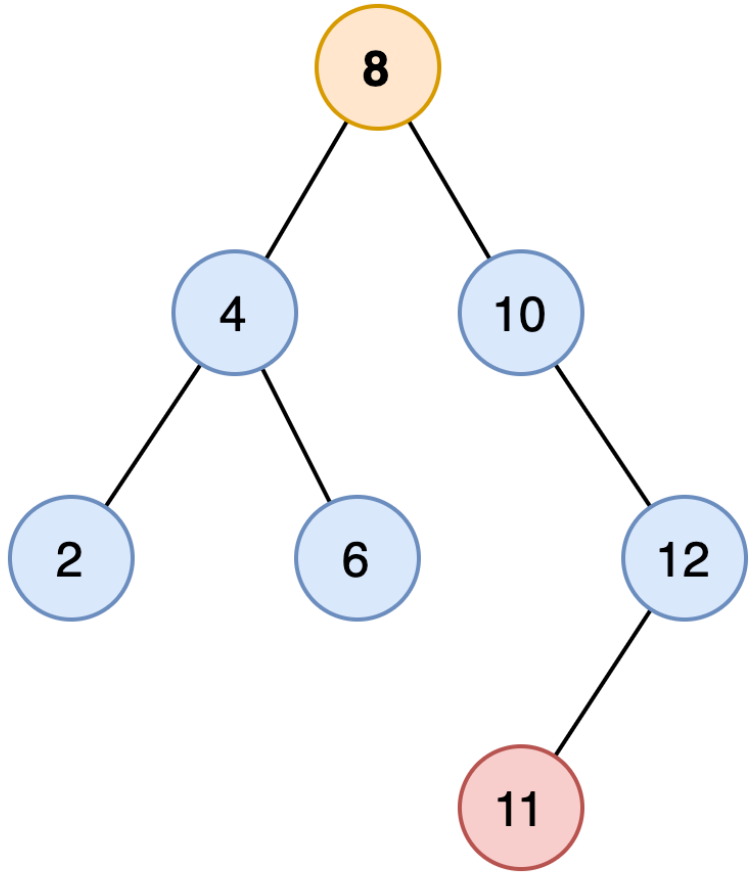
Ищем место в дереве для вставки в узел, каждый раз спускаясь в левое или правое поддерево (в зависимости от сравнения).




```
insert 11
```

Бинарное дерево поиска (BST). Вставка

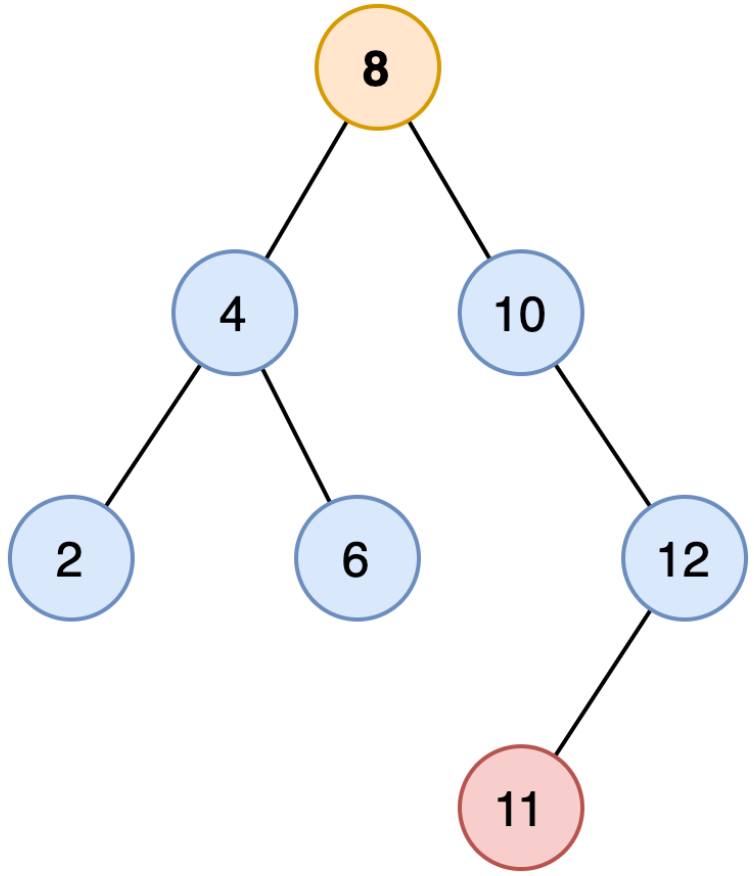
Ищем место в дереве для вставки в узел, каждый раз спускаясь в левое или правое поддерево (в зависимости от сравнения).




```
insert 11
```

Бинарное дерево поиска (BST). Вставка

Ищем место в дереве для вставки в узел, каждый раз спускаясь в левое или правое поддерево (в зависимости от результата сравнения).



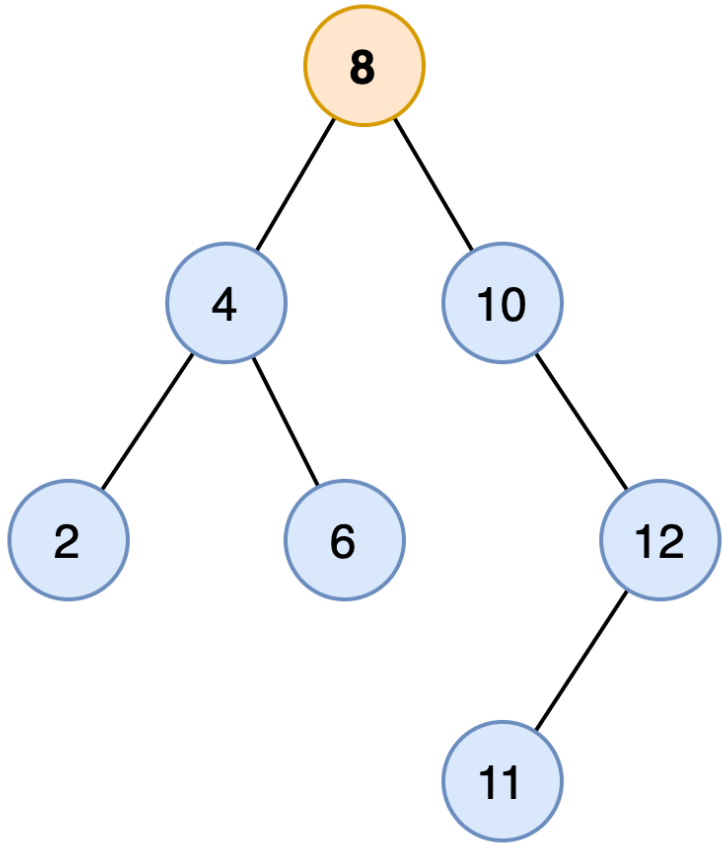
insert 11

```
Tree* ins(Tree *r, T key) {
    if (r == nullptr) {
        return Tree(key);
    } else if (key < r->key) {
        r->left = ins(r->left, key);
    } else {
        r->right = ins(r->right, key);
    }

    return r;
}
```

Бинарное дерево поиска (BST). Удаление

Удаление листа (вершины, у которой нет детей)

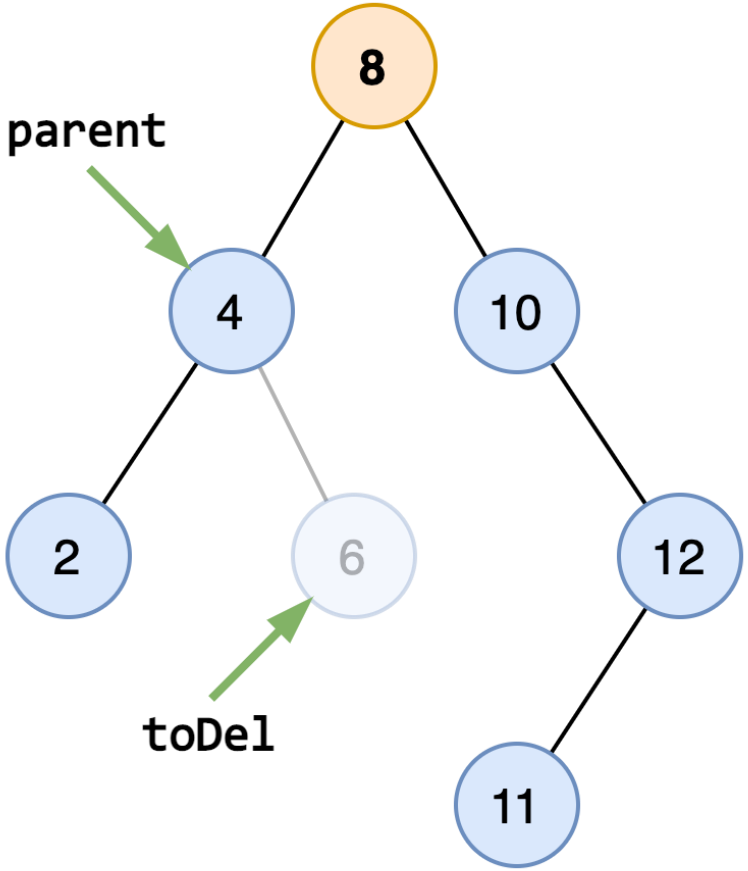


`remove 6`

- Находим 6 (**`toDel`**) и запоминаем родителя (**`parent`**)
- Освобождаем память **`toDel`**
- `parent->right = nullptr`

Бинарное дерево поиска (BST). Удаление

Удаление листа (вершины, у которой нет детей)

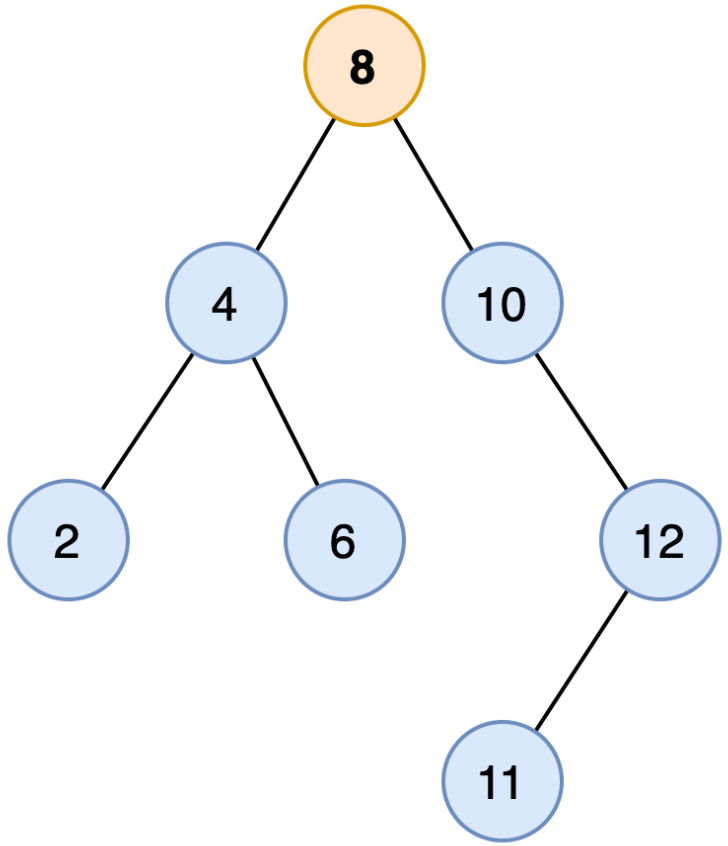


`remove 6`

- Находим 6 (**`toDel`**) и запоминаем родителя (**`parent`**)
- Освобождаем память **`toDel`**
- `parent->right = nullptr`

Бинарное дерево поиска (BST). Удаление

Удаление вершины с **одним ребенком**

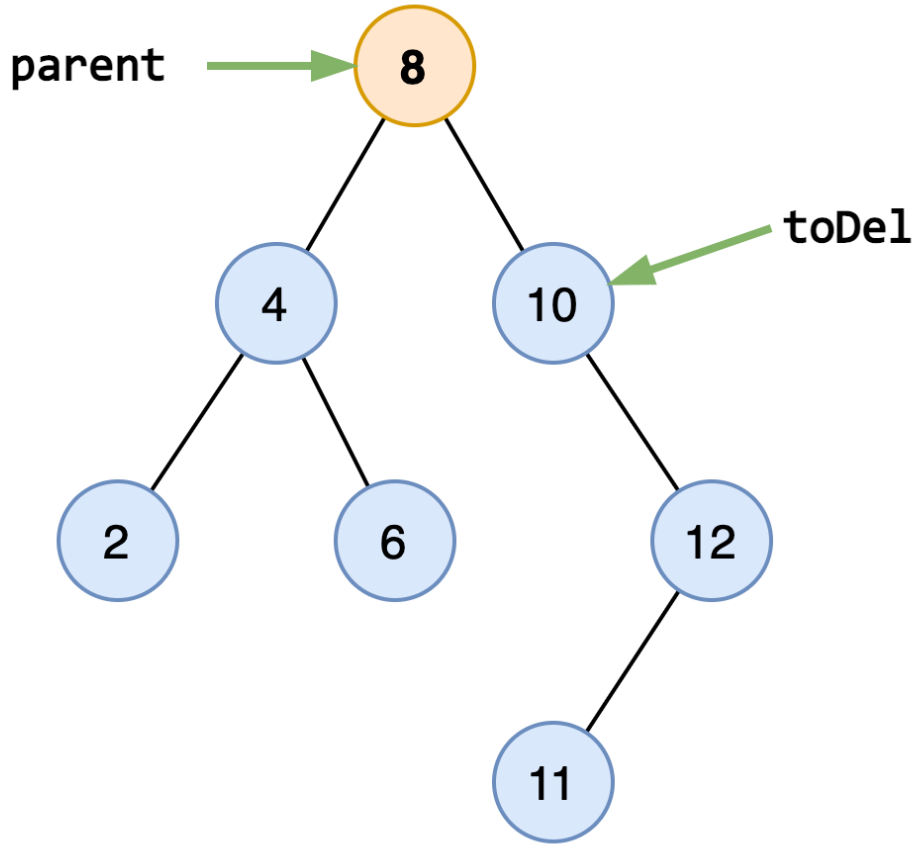


`remove 10`

- Находим `10` (**`toDel`**) и запоминаем родителя (**`parent`**)
- `parent->right = toDel->right`
- Освобождаем память **`toDel`**

Бинарное дерево поиска (BST). Удаление

Удаление вершины с **одним ребенком**

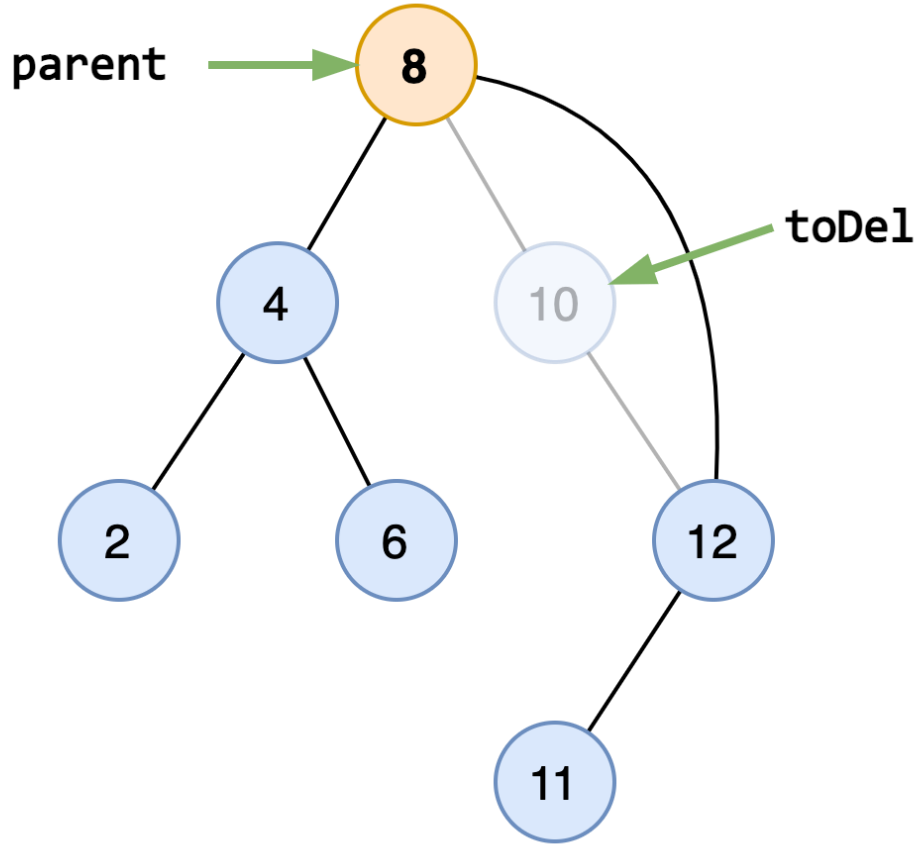


`remove 10`

- Находим `10` (**`toDel`**) и запоминаем родителя (**`parent`**)
- `parent->right = toDel->right`
- Освобождаем память **`toDel`**

Бинарное дерево поиска (BST). Удаление

Удаление вершины с **одним ребенком**

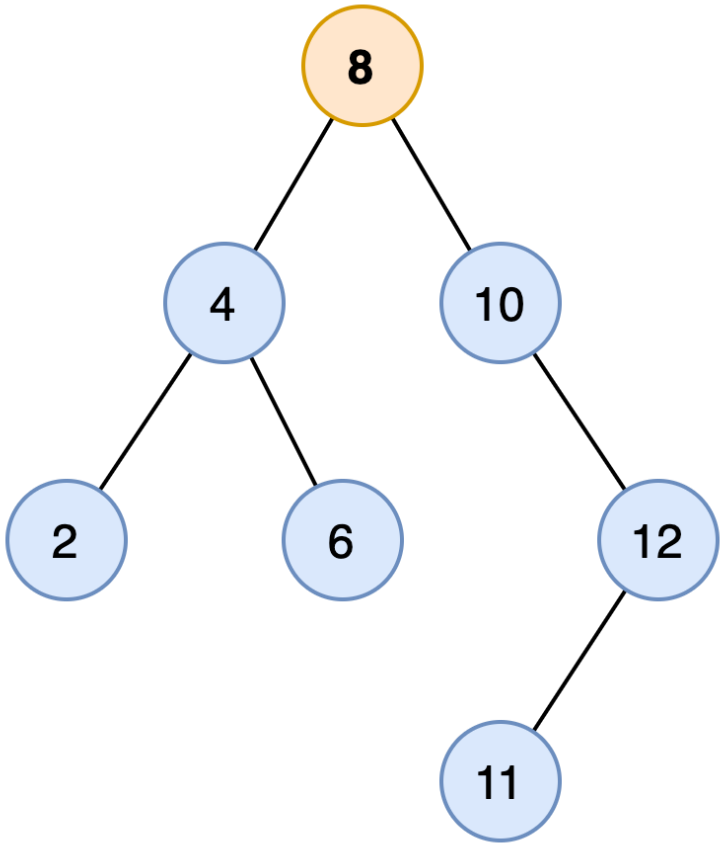


`remove 10`

- Находим `10` (**`toDel`**) и запоминаем родителя (**`parent`**)
- `parent->right = toDel->right`
- Освобождаем память **`toDel`**

Бинарное дерево поиска (BST). Удаление

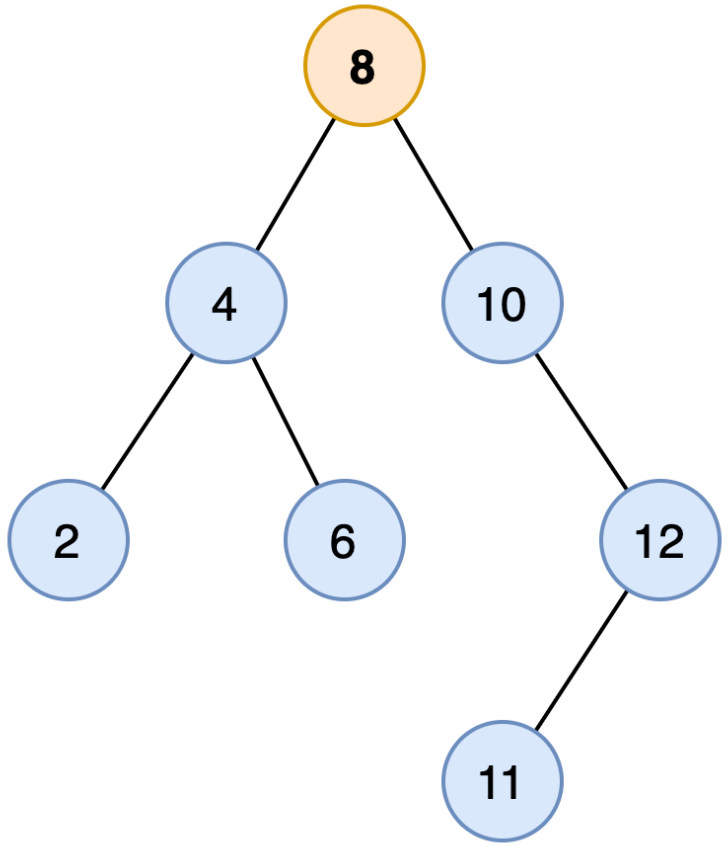
Удаление вершины с **двумя детьми**




```
remove 8
```

Бинарное дерево поиска (BST). Удаление

Удаление вершины с **двумя детьми**



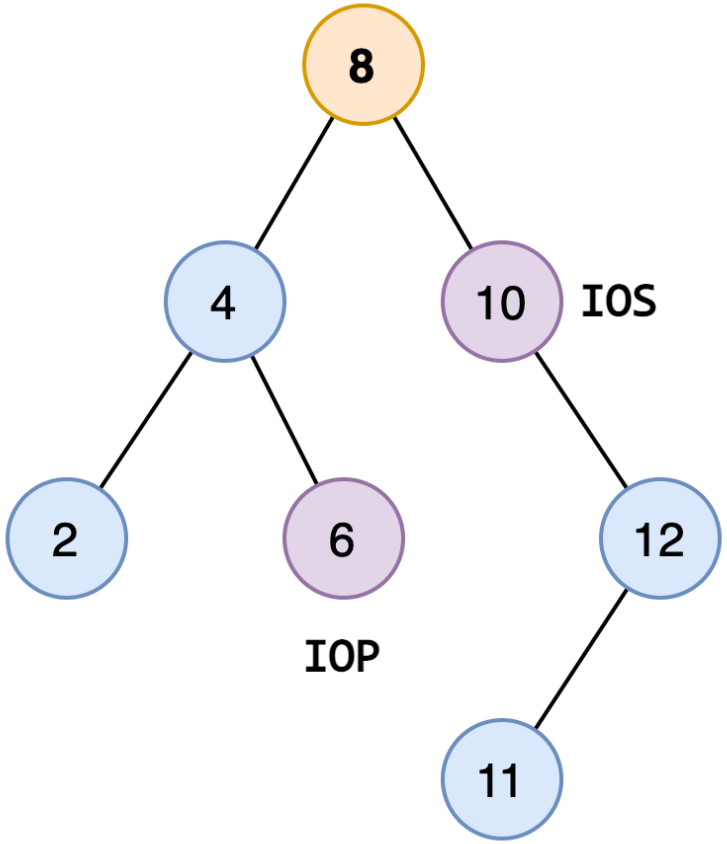
`remove 8`

Находим:

- Предыдущую вершину (**`in-order predecessor`**) для **`8`** **или**
- Следующую вершину (**`in-order successor`**) для **`8`**

Бинарное дерево поиска (BST). Удаление

Удаление вершины с **двумя детьми**



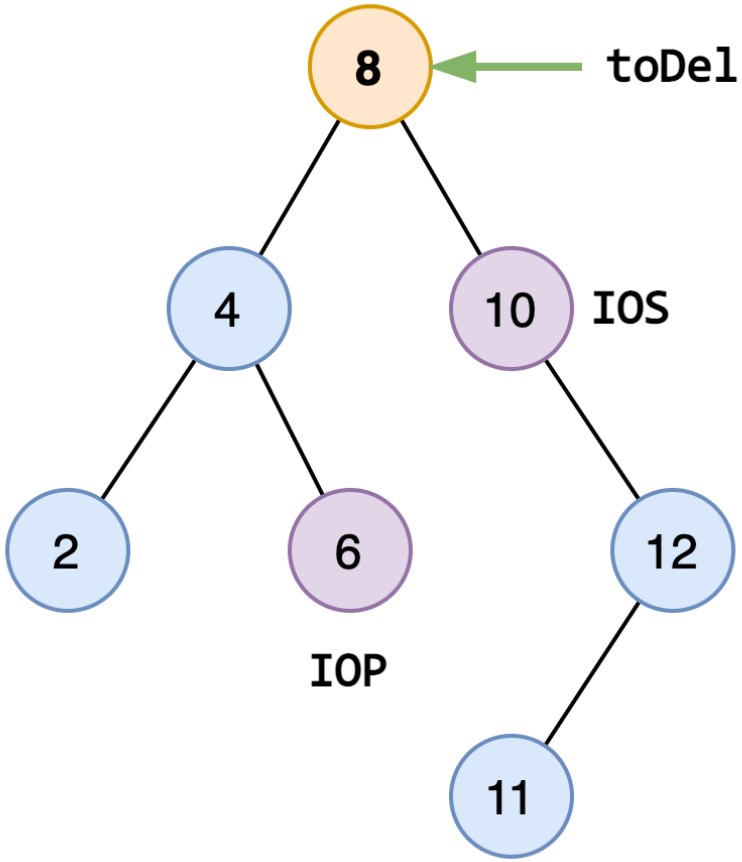
`remove 8`

Находим:

- Предыдущую вершину (**`in-order predecessor IOP`**) для **`8`** **или**
- Следующую вершину (**`in-order successor IOS`**) для **`8`**

Бинарное дерево поиска (BST). Удаление

Удаление вершины с **двумя детьми**



`remove 8`

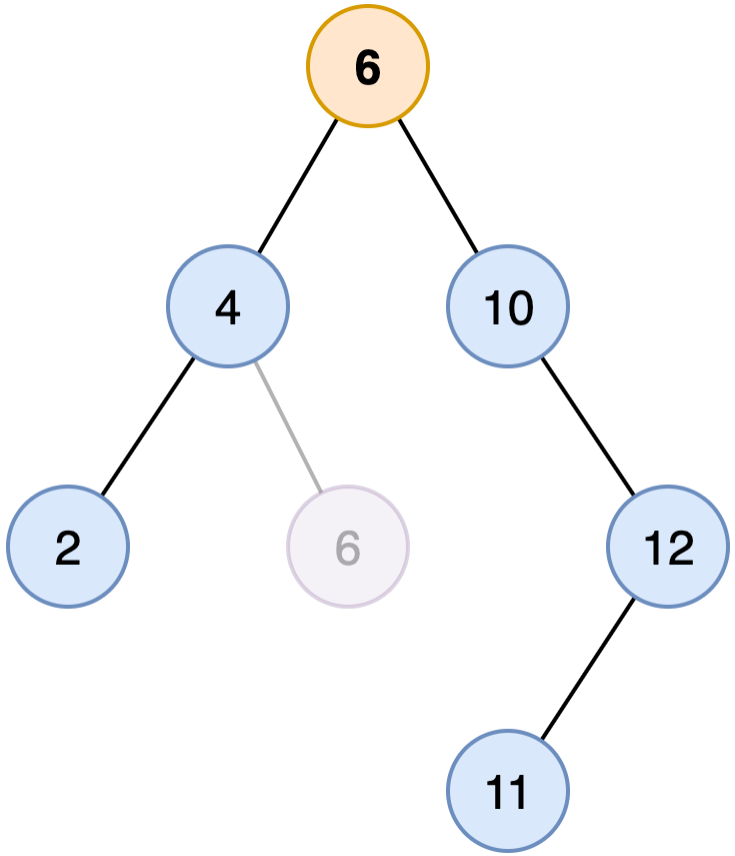
Находим:

- Предыдущую вершину (**`in-order predecessor IOP`**) для **`8`** **или**
- Следующую вершину (**`in-order successor IOS`**) для **`8`**

- `toDel->data = IOP/IOS->data`
- Снова вызываем удаление для **`IOP/IOS`**

Бинарное дерево поиска (BST). Удаление

Удаление вершины с **двумя детьми**



`remove 8`

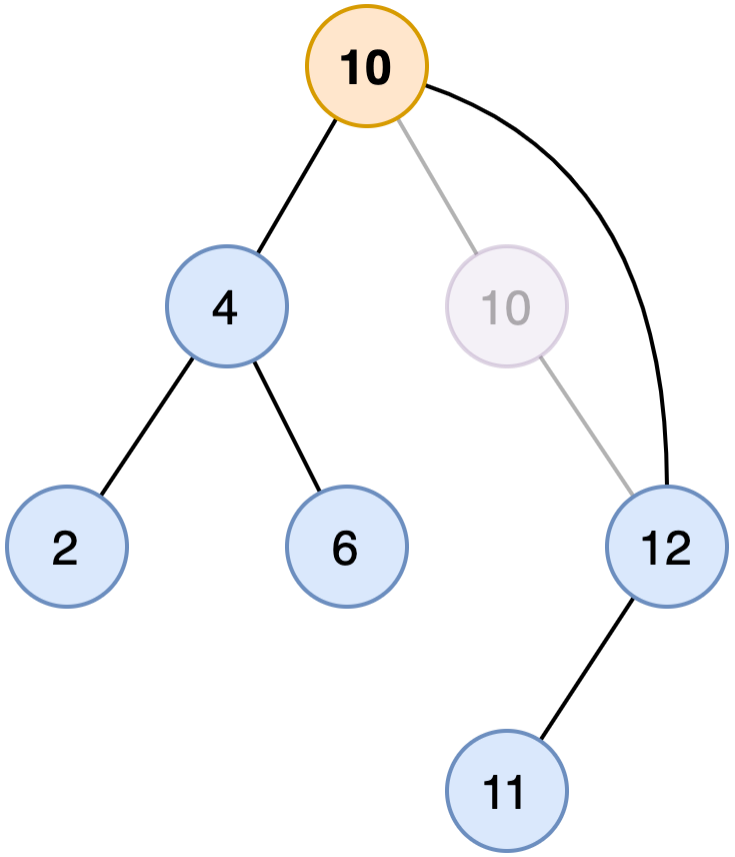
Находим:

- Предыдущую вершину (**`in-order predecessor IOP`**) для **`8`** **или**
- Следующую вершину (**`in-order successor IOS`**) для **`8`**

- `toDel->data = IOP->data`
- Снова вызываем удаление для **`IOP`**

Бинарное дерево поиска (BST). Удаление

Удаление вершины с **двумя детьми**



`remove 8`

Находим:

- Предыдущую вершину (**`in-order predecessor IOP`**) для **8 или**
- Следующую вершину (**in-order successor IOS**) для **8**

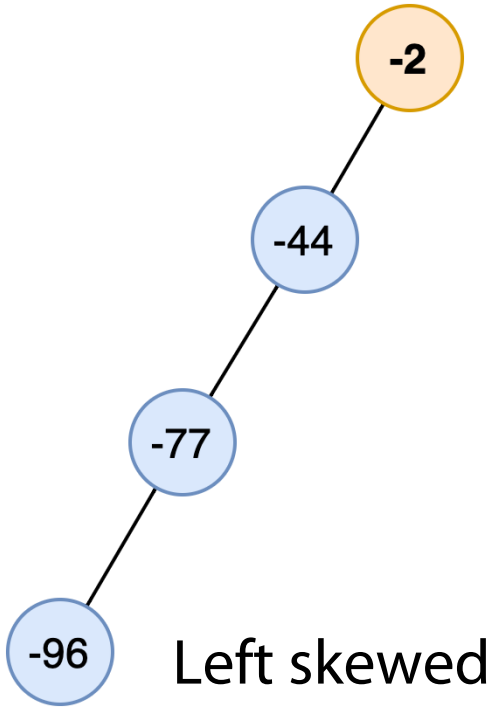
- `toDel->data = IOS->data`
- Снова вызываем удаление для **IOS**

Сбалансированные деревья поиска

Вырождение дерева в список

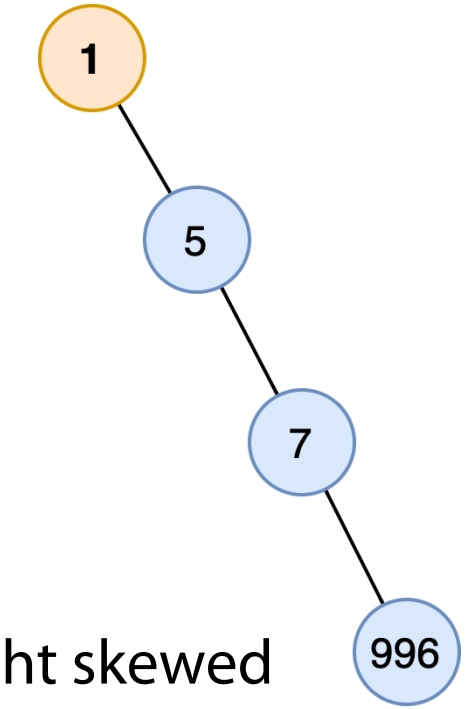
Создадим BST из заданной упорядоченной последовательности

-2 -44 -77 -96



Left skewed

1 5 7 996

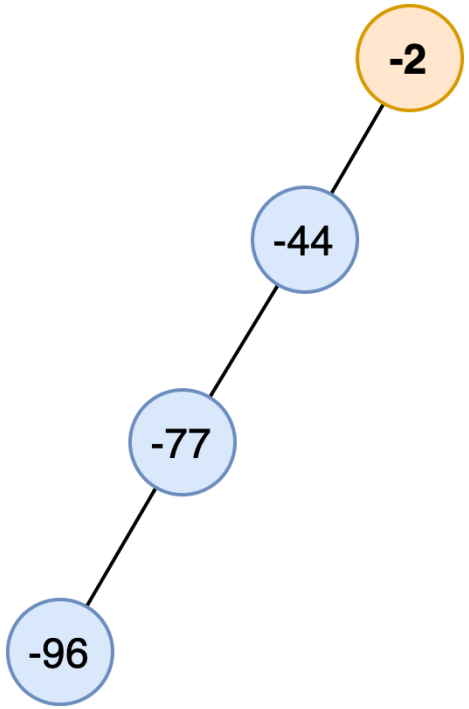


Right skewed

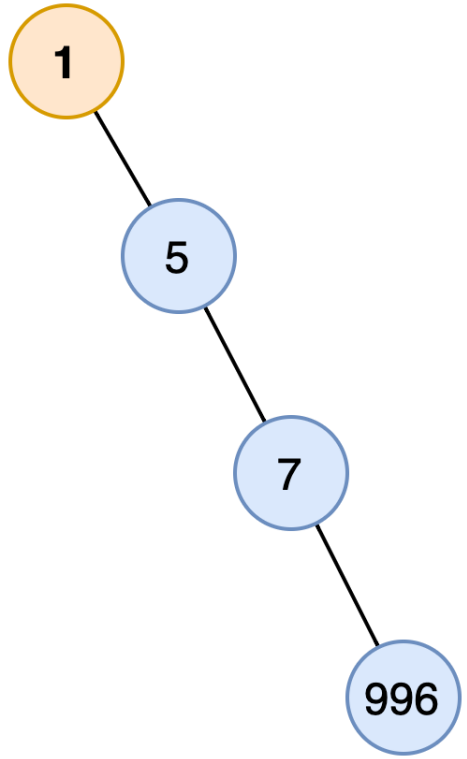
Вырождение дерева в список

Создадим BST из заданной упорядоченной последовательности

-2 -44 -77 -96



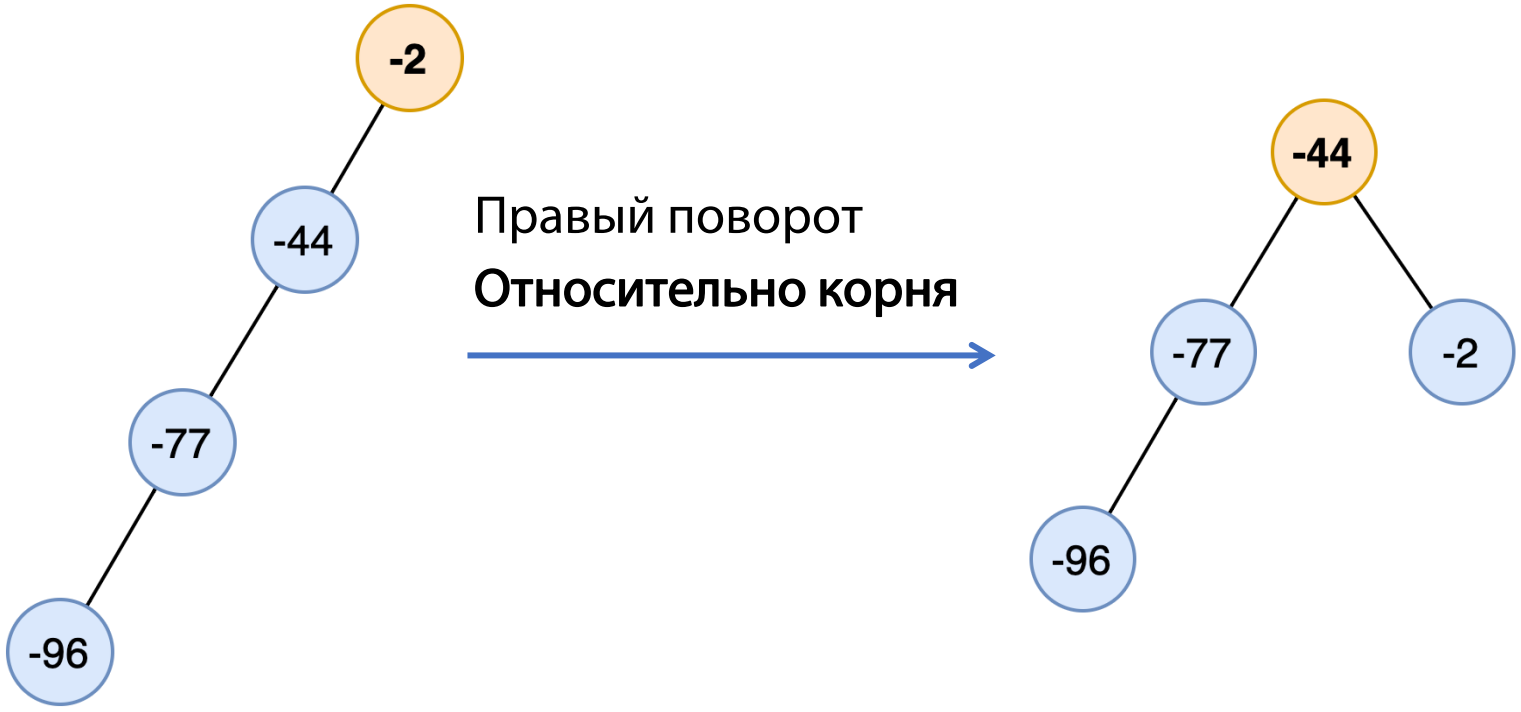
1 5 7 996



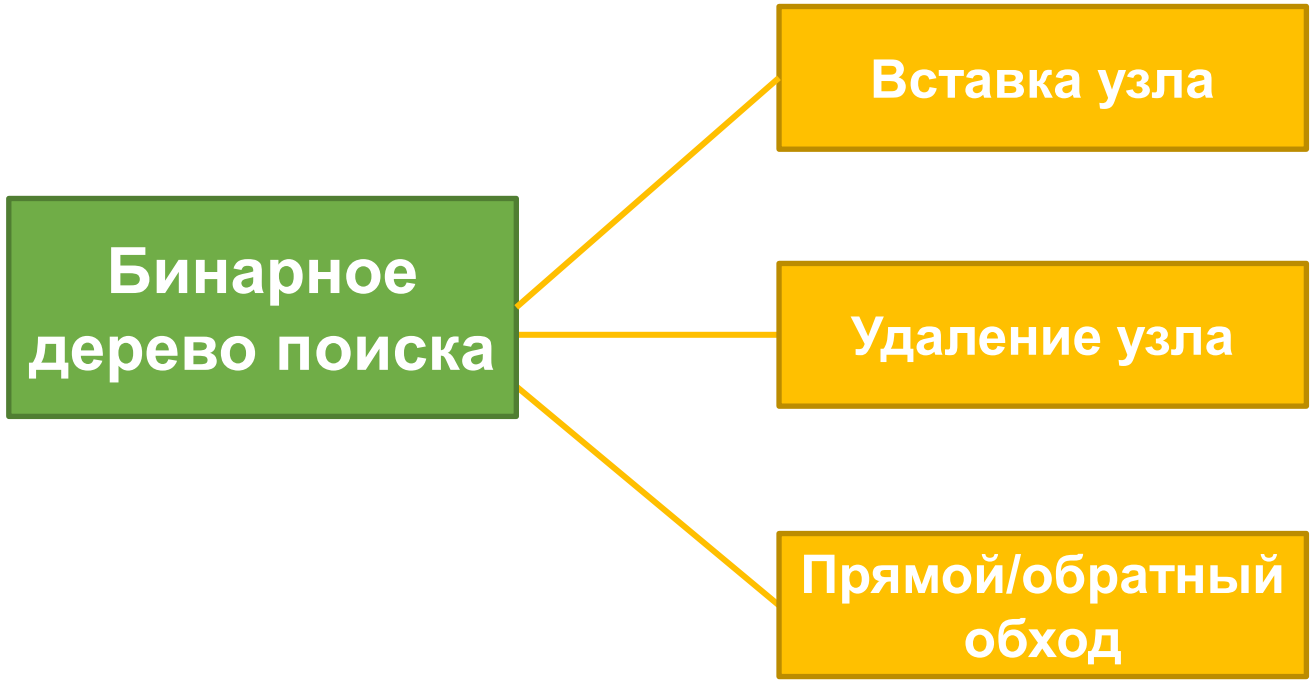
Преимущества дерева улетучиваются…

Поворот дерева

Изменяет структуру дерева с уменьшением его высоты



Реализация


Бинарное дерево поиска
Вставка узла
Удаление узла
Прямой/обратный обход