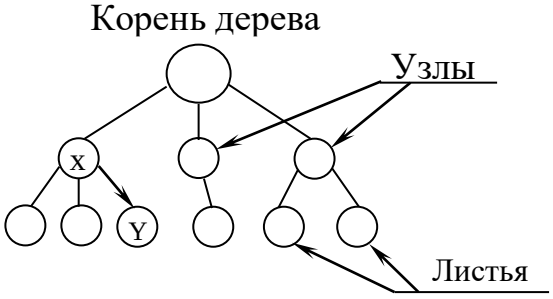
Дисциплина «Алгоритмы решения прикладных задач»
Рабочая тетрадь 1.
Структуры данных. Дерево. Двоичное дерево поиска.

Теоретический материал

Дерево

Дерево состоит из элементов, называемых узлами (вершинами). Узлы соединены между собой направленными дугами. В случае X→Y вершина X называется родителем, а Y – потомком.



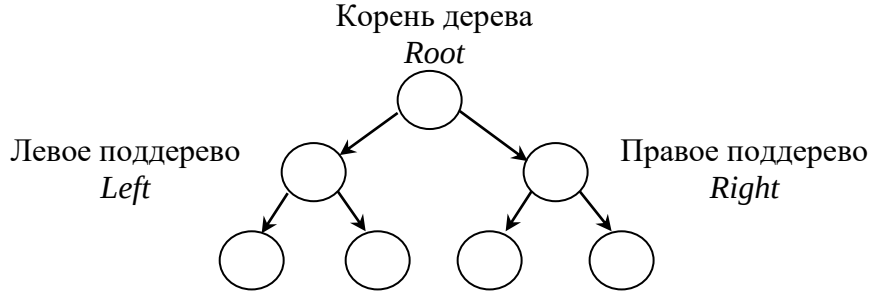
Дерево имеет единственный узел, не имеющий родителей (указателей на этот узел), который называется ***корнем***. Любой другой узел имеет ровно одного родителя, т.е. на каждый узел дерева имеется ровно один указатель.

Узел, не имеющий потомков, называется ***листом***.

Внутренний узел – это узел, не являющийся ни листом, ни корнем. ***Порядок узла*** равен количеству его узлов-потомков. ***Степень дерева*** – максимальный порядок его узлов. ***Высота (глубина) узла*** равна числу его родителей плюс один. ***Высота дерева*** – это наибольшая высота его узлов.

Двоичное дерево

Двоичное дерево – это иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками



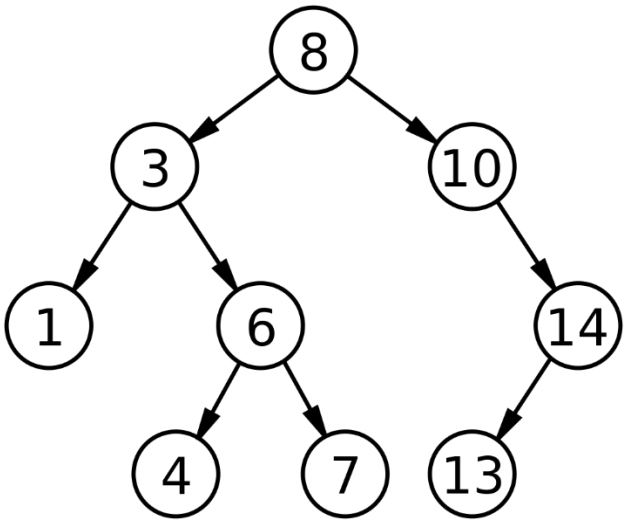
Дерево по своей организации является рекурсивной структурой данных, поскольку каждое его поддерево также является деревом. В связи с этим действия с такими структурами чаще всего описываются с помощью рекурсивных алгоритмов.

Если дерево организовано таким образом, что для каждого узла все ключи (значения узлов) его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева – больше, оно называется **двоичным деревом поиска**. Одинаковые ключи здесь не допускаются.

Двоичное дерево поиска (BST)

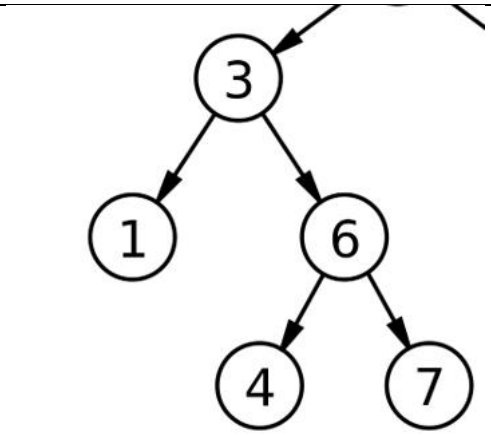
Двоичное дерево поиска (англ. *binary search tree*, BST) — двоичное дерево, для которого выполняются следующие дополнительные условия (*свойства дерева поиска*):

- оба поддерева — левое и правое — являются двоичными деревьями поиска;
- у всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше либо равны*, нежели значение ключа данных самого узла X;
- у всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше*, нежели значение ключа данных самого узла X.

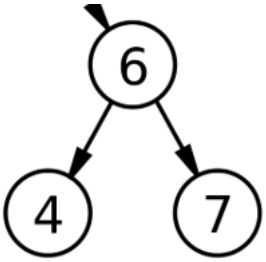


Например, чтобы добавить узел со значением «5», процедура будет следующей:

1. Сравниваем 5 и 8. 5 меньше 8, поэтому идем в левое поддерево.

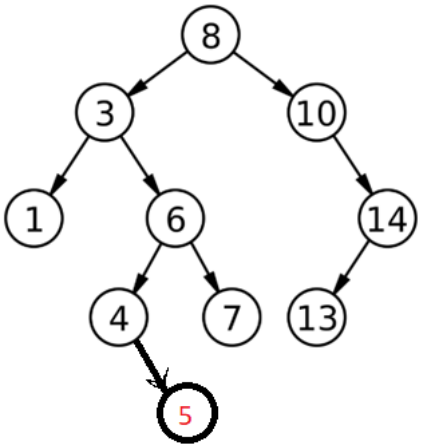


2. Сравниваем 5 и 3. 5 больше, чем 3, поэтому идем в правое поддерево.



3. Сравниваем 5 и 6. 5 меньше, чем 6, поэтому идем в левое поддерево

4. 4 – это лист. Сравниваем 5 и 4. 5 больше, чем 4. Поэтому 5 делаем правым потомком относительно 4. Итоговое дерево имеет вид:



Если в двоичное дерево поиска необходимо добавить данное, которое там уже есть, новый узел не добавляют, а увеличивают значение специальной служебной переменной – количество экземпляров узлов с определенным значением ключа (count)

Структура бинарного дерева построена из узлов. Как и в связанном списке эти узлы содержат поля данных и указатели на другие узлы в коллекции. Узел дерева

содержит поле данных и два поля с указателями, которые называются левым и правым указателями. Значение nullptr является признаком пустого поддерева.

Дерево, по сути, является рекурсивной структурой данных. В результате множество операций будут реализованы через рекурсивные функции. У таких функций будет обязательный служебный параметр - адрес узла текущего уровня.

Структура данных, описывающих дерево, имеет вид:

```
struct Node
{
int value; //Значение узла (ключ), данные любого типа
int count; //Количество экземпляров узла с данным значением (в дереве)
Node * left; //
Node * right; //
};
```

Перечислим основные рекурсивные функции для работы с деревом (названия являются условными):

• addNode() - добавление нового узла в дерево.

• printTree() - обход и печать данных дерева.

• depthTree() - вычисление глубины (высоты) дерева.

• searchNode() - поиск узла в дереве.

• delTree() - удаление дерева.

• delNode() - удаление определенного узла в дереве.

Добавление нового узла в двоичное дерево поиска

Наиболее ответственная операция: добавление в дерево нового узла. Суть алгоритма добавления в следующем: мы начинаем работу с корня всего дерева. Если дерево пустое (корень нулевой), то тогда сразу создается новый узел и его адрес возвращается в качестве корня дерева. Если корень не пустой, то по результату сравнения вставляемых данных и данных в узле дерева мы идем либо в левое, либо в правое поддерево. Далее, либо мы достигаем листа и добавляем новый узел в качестве его потомка, либо находим узел, значение данных в котором совпадает с добавляемым значением. В таком случае добавлять узел не надо, а только увеличить счетчик в найденном узле.

Обход и печать данных дерева

Существует несколько методов прохождения дерева для доступа к его элементам. К ним относятся **прямой, обратный и симметричный**. При прохождении дерева используется рекурсия, поскольку каждый узел является корнем своего поддерева. Каждый алгоритм выполняет в узле три действия: заходит в узел, рекурсивно спускается по левому и по правому поддереву. Спуск прекращается при достижении пустого поддерева (нулевой указатель).

- Порядок действий при прямом обходе:
  1. Обработка данных узла.
  2. Прохождение левого поддерева.
  3. Прохождение правого поддерева.
- Порядок действий при обратном обходе:
  1. Прохождение левого поддерева.
  2. Прохождение правого поддерева.
  3. Обработка данных узла.
- Порядок действий при симметричном обходе:
  1. Прохождение левого поддерева.
  2. Обработка данных узла.
  3. Прохождение правого поддерева.

Поиск конкретного элемента в дереве

Известно, что слева от узла располагается элемент, который меньше чем текущий узел. Из чего следует, что если у узла нет левого наследника, то он является минимумом в дереве. Таким образом, можно найти минимальный элемент дерева.

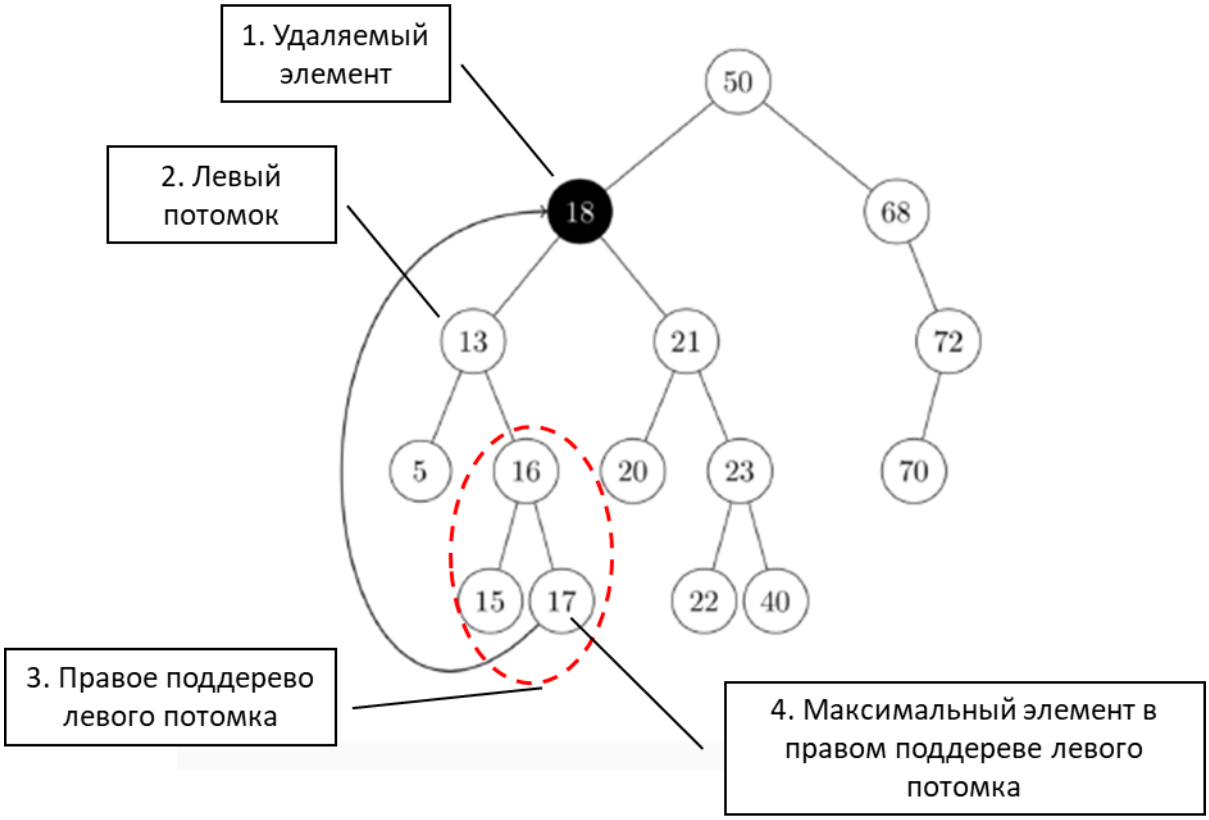
Поиск нужного узла по значению выполняется следующим образом. Если искомое значение больше узла, то продолжаем поиск в правом поддереве, если меньше, то продолжаем в левом. Если узлов уже нет, то элемент не содержится в дереве.

Удаление узла с определенными данными

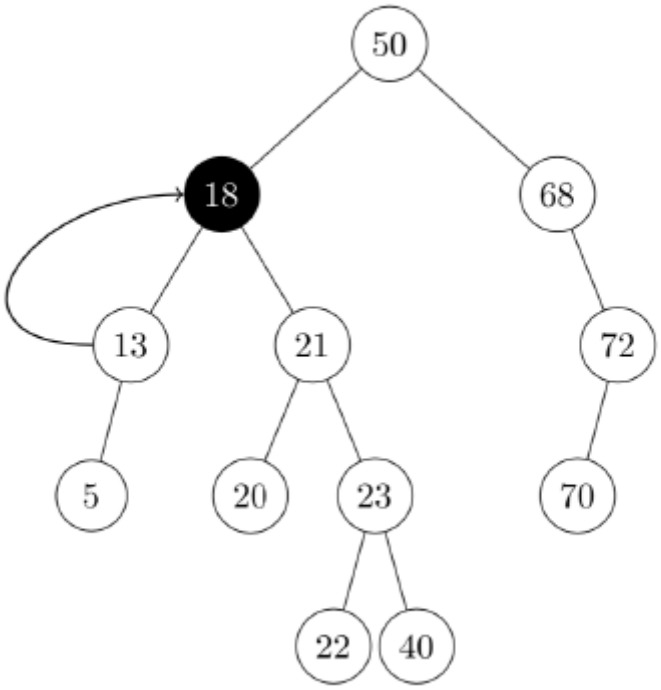
Данная операция – наиболее сложная, поскольку при удалении произвольного узла должна сохраняться упорядоченность оставшихся элементов дерева. Существует 4 возможных ситуации при удалении узла дерева:

1. У удаляемого узла нет потомков. Тогда мы можем освободить память, занимаемую узлом, а у его родителя выставить nullptr в указателе на потомка.

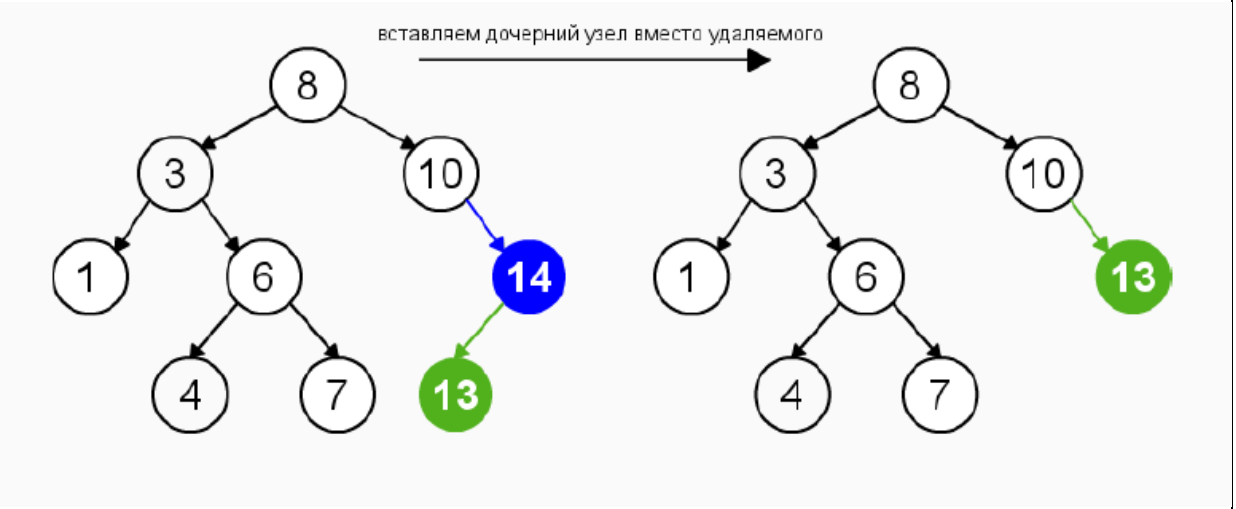
2. Удаляемый узел имеет двух потомков, причем у левого потомка есть свое правое поддерево. В этом случае нужно найти в этом правом поддереве наибольший элемент и вставить его вместо удаляемого узла.



3. Удаляемый узел имеет двух потомков, причем у левого потомка нет правого поддерева. В этом случае элемент заменяется на корень левого поддерева:



4. Удаляемый узел имеет одного потомка (левого или правого). В этом случае мы присваиваем адрес потомка указателю нашего родителя, вместо адреса текущего узла:



Задание 1

Задача:

Создать двоичное дерево поиска (в узлах хранятся целые положительные числа). Программа должна запрашивать количество элементов дерева, далее значения, хранящиеся в элементах, создаются генератором случайных чисел. Для готового дерева реализовать операции:

а) добавление нового узла в дерево;

б) обход дерева (прямой, обратный или симметричный – по выбору) и печать элементов дерева на экран;

в) вычисление глубины (высоты) дерева;

г) поиск конкретного элемента в дереве

д*) удаление определенного узла в дереве;

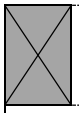
Решение:

Ответ:

Задание 2*

Задача:

На основе двоичного дерева поиска реализовать консольное приложение «Телефонная книга». Двоичное дерево поиска в данном случае – это хранилище записей (имя человека, его телефон) с операциями поиска и удаления записей по имени человека и операцией добавления новой записи.

	При этом у одного и того же человека может быть несколько номеров телефона.
Решение:	
	
Ответ: