

Бинарные деревья поиска.

Бинарные деревья поиска (БДП) являются одним из вариантов корневых деревьев, которые, в свою очередь являются одним из видов графов.

С точки зрения алгоритмизации БДП являются связанными структурами данных, которые используются для разных целей.

Каждый узел представляет собой отдельный объект, одним из элементов которого является **key**.

Элементы объекта:

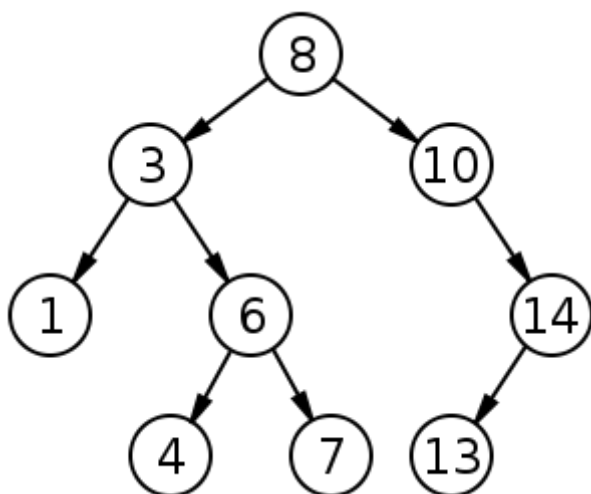
Key – ключ элемента

P – указатель на родительский элемент

Left – указатель на левый дочерний элемент

Right – указатель на правый дочерний элемент

Value – значение, которое должно храниться в дереве, по аналогии со значением элемента массива.



Условие существования бинарного дерева поиска:

Пусть X – узел БДП, если Y – является узлом в левом поддереве X , то $Y.key \leq X.key$. Если Y является узлом в правом поддереве X , то $Y.key > X.key$.

Операции с БДП

Вставка

Алгоритм добавления элемента в БДП:

Пусть необходимо добавить в дерево узел Z, элементы которого равны:

Z.key = V (заданное значение)

Z.right = null

Z.left = null

Z.Value = val (Заданное значение)

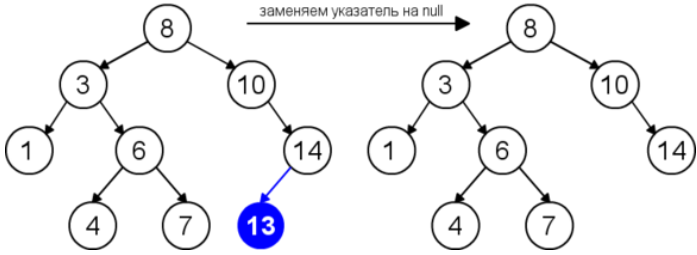
Тогда последовательность добавления выглядит следующим образом:

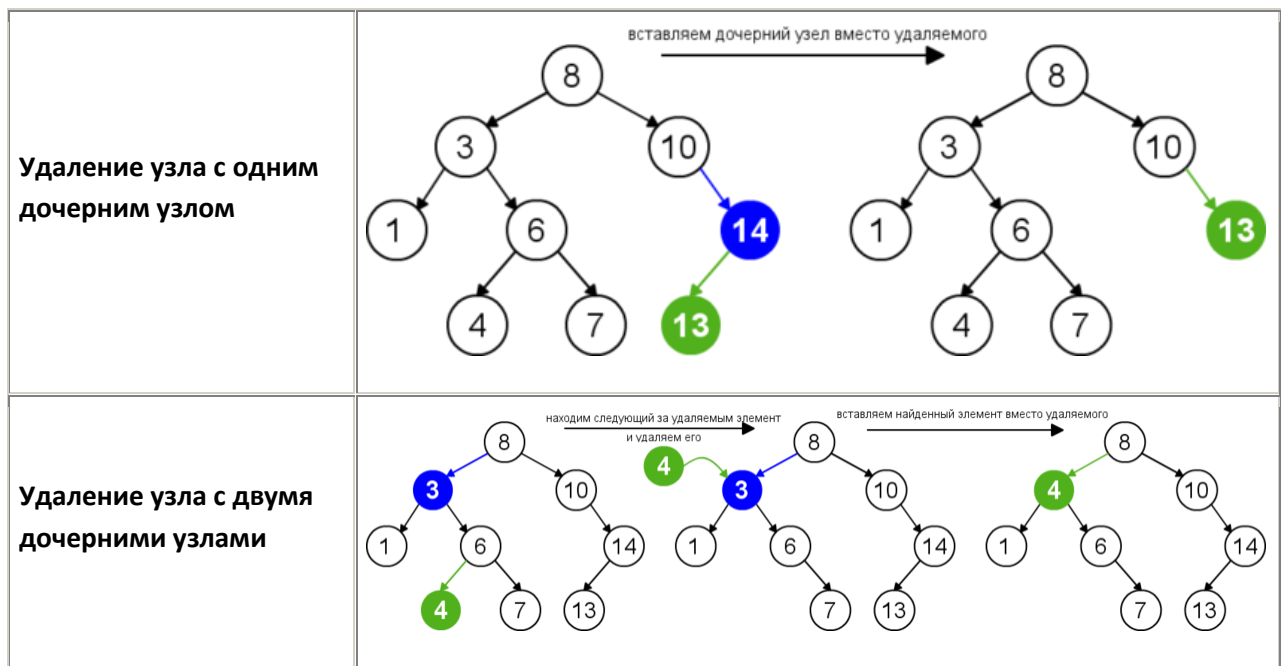
```
Y = null    // создаем пустой узел
X = Root    // Получаем корневой элемент дерева
While (X != null ) // пока не найдем «край» дерева
{
    Y = X    //текущий узел равен проверяемому
    If Z.key < X.key
        X = X.left
    Else
        X = X.right
}
Z.p = Y
If y == null
    Root = Z //Дерево было пустым
Else if Z.key < Y.key
    Y.left = Z
Else
    Y.right.Z
```

Удаление узла

Нерекурсивная реализация

Для удаления узла из бинарного дерева поиска нужно рассмотреть три возможные ситуации. Если 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. Наконец, 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. Таким образом, свойство бинарного дерева поиска не будет нарушено. Данная реализация удаления не увеличивает высоту дерева. Время работы алгоритма - **O(h)**.

Случай	Иллюстрация
Удаление листа	



```

func delete(t : Node, v : Node):           // t – дерево, v – удаляемый элемент
    p = v.p                                // предок удаляемого элемента
    if v.left == null and v.right == null // первый случай: удаляемый элемент
- лист
        if p.left == v
            p.left = null
        if p.right == v
            p.right = null
    else if v.left == null or v.right == null // второй случай: удаляемый
элемент имеет одного потомка
        if v.left == null
            if p.left == v
                p.left = v.right
            else
                p.right = v.right
                v.right.p = p
        else
            if p.left == v
                p.left = v.left
            else
                p.right = v.left
                v.left.parent = p
    else // третий случай: удаляемый элемент имеет двух потомков
        successor = next(v, t)
        v.key = successor.key
        if successor.p.left == successor
            successor.p.left = successor.right
            if successor.right != null
                successor.right.parent = successor.parent
        else
            successor.p.right = successor.right
            if successor.right != null
                successor.right.p = successor.parent

```

В этом примере метод **next ()** – поиск следующего элемента.

Вывод данных из дерева

Операция вывода данных из дерева является рекурсивной, т.е. использует вызов операции из собственного тела.

Пример вывода дерева:

```
Walk (X) // Аргумент - узел дерева
If X != null // Если элемент существует, то
{
    Walk (X.left) //Пытаемся идти в левое поддерево
    Print X.Key //Выводим значение ключа элемента
    Print X.Value //Выводим значение элемента
    Walk (X.right) //Пытаемся идти в правое поддерево
}
```

Для вывода полного содержимого дерева нужно вызвать метод Walk (Root), где Root – корневой элемент дерева.

Поиск элемента в дереве

Операция поиска так же рекурсивная.

```
Search (X, k) // X - узел дерева, k - ключ, который ищем в дереве
If x == null или k == x.key
    Return x;
If k < x.key
    Return Search (x.left, k)
Else
    Return Search (x.right, k)
```

Поиск следующего элемента

```
Node next(x )
current = Root // root - корень дерева
successor = null //Следующий элемент
while current != null
    if current.key > x
        successor = current
        current = current.left
    else
        current = current.right
return successor
```

Поиск максимума и минимума

Для поиска минимального или максимального элемента необходимо рекурсивно пройти в левое или правое поддерево соответственно. Алгоритм и код придумайте самостоятельно.

Альтернативное представление БДП.

В качестве механизма хранения данных дерева, можно использовать несколько массивов для представления каждого из элементов узла дерева.

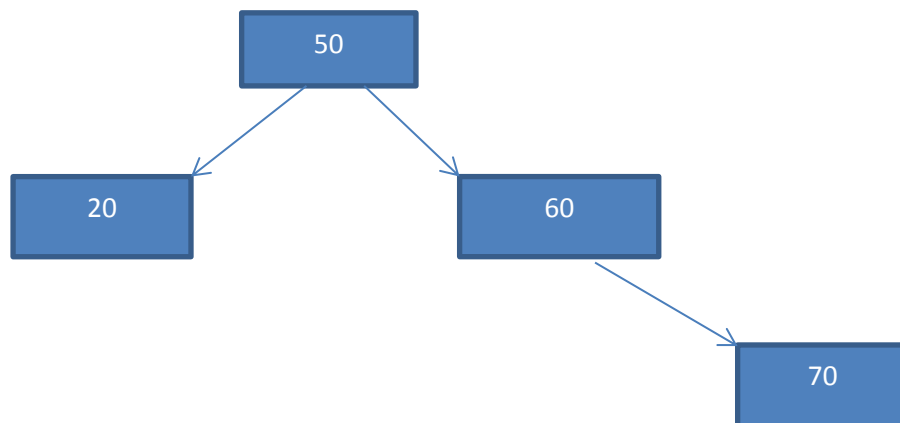
Например, массивы Key[], Left[], Right[], Value[], P[].

В таком случае элементы этих массивов с одинаковым индексом являются описанием одного элемента дерева.

Пример:

Индекс	Key[]	Left[]	Right[]	P[]	Комментарий
0	50	1	2	null	Ключевой элемент – 50, P = null, значит, это корень дерева Слева от элемента находится элемент, описываемый значениями массивов с индексом 1, справа – 2.
1	20	null	Null	0	
2	60	null	3	0	
3	70	null	null	2	

Вид дерева, описанный этими массивами:



Справочная информация

Для добавления в дерево нового элемента необходимо изменить размер каждого из массивов, увеличив его на единицу. На языке C# эта операция выполняется следующим образом:

```
Array.Resize(ref <имя массива>, <новая длина>);
```

Получить размер массива можно при помощи использования свойства Length массива:

`<имя массива>.Length`

Задание.

Написать программу, которая будет реализовывать основные операции с бинарным деревом поиска: добавление и удаление узлов, вывод дерева в консоль, поиск элементов.

Рекомендуется работу с деревом оформить в виде отдельного класса на языке C# с публичными методами, обеспечивающими работу с ним.

Источник данных для заполнения дерева – генератор случайных чисел, в качестве типа хранимых данных в массиве Value можно взять String.