***1.Постановка задачи.***

Основные задачи работы:

* изучение различных форм представления бинарных деревьев;
* реализация бинарных деревьев посредством использования динамической памяти и указателей (ссылок);
* освоение и реализация алгоритмов построения деревьев бинарного поиска;
* реализация алгоритма поиска с включением на дереве бинарного поиска;
* реализация алгоритмов прямого, обратного и симметичного обхода дерева;
* оценка временной сложности алгоритма поиска с включением;
* определение временной сложности алгоритма поиска с включением.

***2.Теория***

O.1 **Дерево** - это связный граф без циклов.

O.2 **Бинарное (двоичное) дерево (binary tree)** - это упорядоченное дерево, каждая вершина которого имеет не более двух поддеревьев.

Бинарное дерево является рекурсивной структурой, поскольку каждое его поддерево само является бинарным деревом и, следовательно, каждый его узел в свою очередь является корнем дерева.   
Узел дерева, не имеющий потомков, называется листом.

Дерево двоичного поиска для множества чисел M - это размеченное бинарное дерево, каждой вершине которого сопоставлено число из множества M, причем все пометки удовлетворяют следующим условиям:

1. существует ровно одна вершина, помеченная любым числом из множества M;
2. все пометки левого поддерева строго меньше, чем пометка текущей вершины;
3. все пометки правого поддерева строго больше, чем пометка текущей вершины.

Если выражаться простым языком, то структура дерева двоичного поиска подчиняется простому правилу: "**если больше - направо, если меньше - налево**".

Говорят, что **бинарное дерево идеально- сбалансировано**, если для каждого

его узла количество узлов в левом и правом поддеревьях различается не более чем на 1.

**Создание идеально сбалансированного дерева:**

Если число узлов не известно и дана последовательность значений поля данных вершин а[0],а[1],...,а[n - 1], то можно использовать следующий рекурсивный алгоритм построения

идеально сбалансированного дерева:

1. Берем очередное значение а[n/2] в качестве значения корня дерева (поддерева).
2. Строим левое поддерево с n1 = n/2 узлами тем же cпособом.
3. Строим правое поддерево с nr = n - nl - 1 узлами.

Таким образом, значение a[n/2] окажется в корне дерева, а именно на него

будет ссылаться указатель дерева, а[0],...а[nl] значения попадут в левое поддерево,

остальные — в правое поддерево. Следовательно, распределение значений по узлам

дерева полностью определяется исходной последовательностью данных.

**Обход дерева** - это некоторая последовательность посещения всех его вершин.

Рассмотрим прямой, симметричный и обратный обходы деревьев бинарного поиска:

**Алгоритм прямого обхода:**

-Каждый узел посещается до того, как посещены его потомки.

Для корня дерева рекурсивно вызывается следующая процедура:

* -Посетить узел
* -Обойти левое поддерево
* -Обойти правое поддерево

**Алгоритм обратного обхода:**

-Узлы посещаются 'снизу вверх'.

Для корня дерева рекурсивно вызывается следующая процедура:

* -Обойти левое поддерево
* -Обойти правое поддерево
* -Посетить узел

**Алгоритм симметричного обхода:**

-Посещаем сначала левое поддерево, затем узел, затем - правое поддерево.

Для корня дерева рекурсивно вызывается следующая процедура:

* -Обойти левое поддерево
* -Посетить узел
* -Обойти правое поддерево

**Поиск (локализация) элемента в дереве:**

Поиск элемента может преследовать различные цели. Во-первых, определить, имеется ли искомый элемент в дереве или нет.

В этом случае результатом будет возврат признака о наличии или отсутствии элемента. Во-вторых, поиск с целью выборки и обработки данных элемента, тогда результат возвращается в виде адреса вершины. В-третьих, поиск с целью удаления элемента: такой поиск обычно является частью операции удаления, а не самостоятельной операцией.

Алгоритм поиска в бинарном дереве поиска, как простом, так и сбалансированном, достаточно прост. Поиск осуществляется целенаправленным движением по ветвям дерева. Если ключ поиска равен ключу в вершине, значит, ключ найден и его адрес возвращается через параметр-указатель. Если ключ поиска меньше ключа в вершине, то осуществляется движение вниз влево, в противном случае — вниз вправо.

Если в очередной вершине дальнейшее движение вниз невозможно (указатель равен NULL),

то это означает, что искомого ключа нет в дереве. Максимальная длина поиска равна высоте дерева.

**Включение элемента в бинарное дерево поиска:**

Включение элемента состоит из трех действий: поиска места включения; получения

динамической памяти для элемента; занесения данных в новый узел.

Сразу же нужно определиться, что делать, если в дереве уже есть элемент с включаемым ключом.

В одних случаях такой элемент может просто отвергаться, в других случаях включаться например, если дерево используется для сортировки по ключа Возможно использование такого элемента также для обновления данных в узле дерева.

Рассмотрим вариант, когда элемент с дублирующим ключе отвергается. Порядок поиска места включения такой же, как и при поиске элемента в дереве. Однако положительным исходом считается отсутствие элемента с заданным ключом, т.е. приход в результате движения к пустому поддереву, обозначенному указателем с нулевым значением NULL. Тогда получаем динамическую память для элемента и его адрес заносим в этот пустой указатель, затем помещаем в новую вершину ключ и данные, а указатели в ней обнуляем.

Если же элемент с дублирующим ключом включается, то в каждой вершине осуществляется сравнение только на «меньше» на «больше или равно». Если ключ включаемого элемента меньше ключа узла, то движение — вниз влево, в противном случае вниз вправо. Применение такого дерева для сортировки приведёт к «устойчивой» сортировке, т.е. элементы с одинаковыми ключами отсортируются в порядке их поступления в дерево.

**Удаление элемента из бинарного дерева поиска:**

Операция удаления элемента из дерева поиска сложнее, чем операция включения элемента. После удаления элемента дерево поиска должно сохранить свое свойство — обеспечение поиска элементов. Действия по удалению определяются тем, в каком месте дерева находится удаляемый элемент. В процедуре удаления различают следующие ситуации.

1. Элемента с искомым ключом нет, дерево не изменяется, возврат с признаком отсутствия ключа.
2. Удаляемый элемент — лист, в родительском узле соответствующий указатель на удаляемый элемент обнуляется, память из-под элемента освобождается.
3. Удаляемый элемент только с одним поддеревом. Корректируются указатели, память освобождается.
4. Удаляемый элемент имеет два поддерева. В этом случае необходимо спуститься вдоль самой правой ветви левого поддерева до конца и заменить значение удаляемого элемента значением конечного элемента и удаляется конечный элемент.

**Оценка временной сложности алгоритмов:**

Временная сложность операции поиска с включением и операции поиска с удалением дерева равны высоте этого дерева( т.е. O(logN))(лучший случай).В случае изменения структуры идеальной сбалансированности дерева пользователем, оно может выродиться в список, где максимальная сложность операций поиска, включения и удаления элементов составит Tmax(n)=*O*(n)(худший случай).

***3.Руководство программиста***

Это программа разработана на языке С++ и содержит следующие функции.

***void outxy(int x, int y, int var) -*** *функция, которая преобразует элементы массива, в текстовую форму и выводит по заданным координатам.* ***Параметры****: координаты места вывода на экран, элемент дерева.*

***node \*Create(int \*mas, int n)****– функция построения идеально сбалансированного дерева.*

***Параметры:*** *отсортированный массив чисел и количество элементов в массиве.*

***void graphTravers(node \*t) –*** *графическое отображения обхода дерева.*

***Параметр:*** *указатель на дерево.*

***void inOrderTravers(node \*t) –*** *функция , осуществляющая симметричный обход дерева.*

***Параметр:*** *указатель на дерево.*

***void preOrderTravers(node \*t) –*** *функция прямого обхода дерева.* ***Параметр****: указатель на дерево.*

***void postOrderTravers(node \*t)*** *– функция, осуществляющая обратный обход дерева.*

***Параметр****: указатель на дерево.*

***void graph()-****инициирует графический режим*

***void fillmas(int \*mas,int n) -*** *функция заполнения массива случайными числами.* ***Параметры****: указатель на массив, количество элементов.*

***void sort(int \*p, long size) -*** *функция сортировки массива.* ***Параметры:*** *указатель на массив, размер.*

***void printTree(node \*p, int n,int left, int right )*** *- функция графического представления дерева.* ***Параметры:*** *указатель на дерево, уровень дерева, левая и правая граница расположения текущего элемента дерева.*

***node \*searchIn(node \*&root, int n) –*** *функция , осуществляющая поиск с включением.*

***Параметры:*** *ссылка на указатель дерева и элемент, который необходимо найти.*

***void deleteTree(node\*)-****функция удаления дерева, параметром является указатель на дерево.*

***node \*& rightmost(node \*&root) -*** *функция смещения к самому правому элементу левого поддерева.*

***node\* searchDel(node \*root, int val) –*** *функция, осуществляющая поиск с удалением.*

***Параметры:*** *указатель на дерево и искомый элемент, вводимый пользователем.*

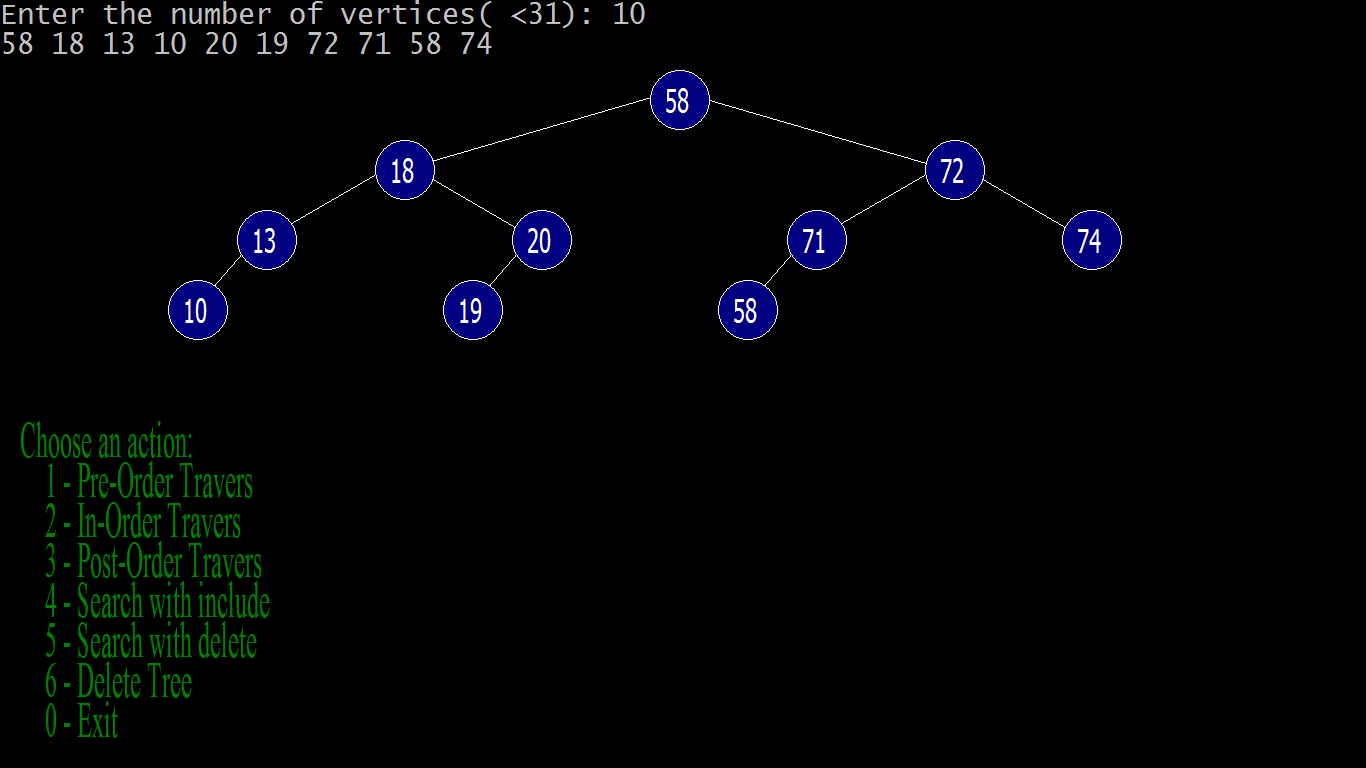
***void deleteTree(node\*)-****функция удаления дерева.* ***Параметр:***  *указатель на дерево.*

***void reDrawTree(node \*z, int c) -*** *функция перерисовывает дерево.* ***Параметр:***  *указатель на дерево.*

***void menu() -*** *функция выводит список действий для работы с деревом.*

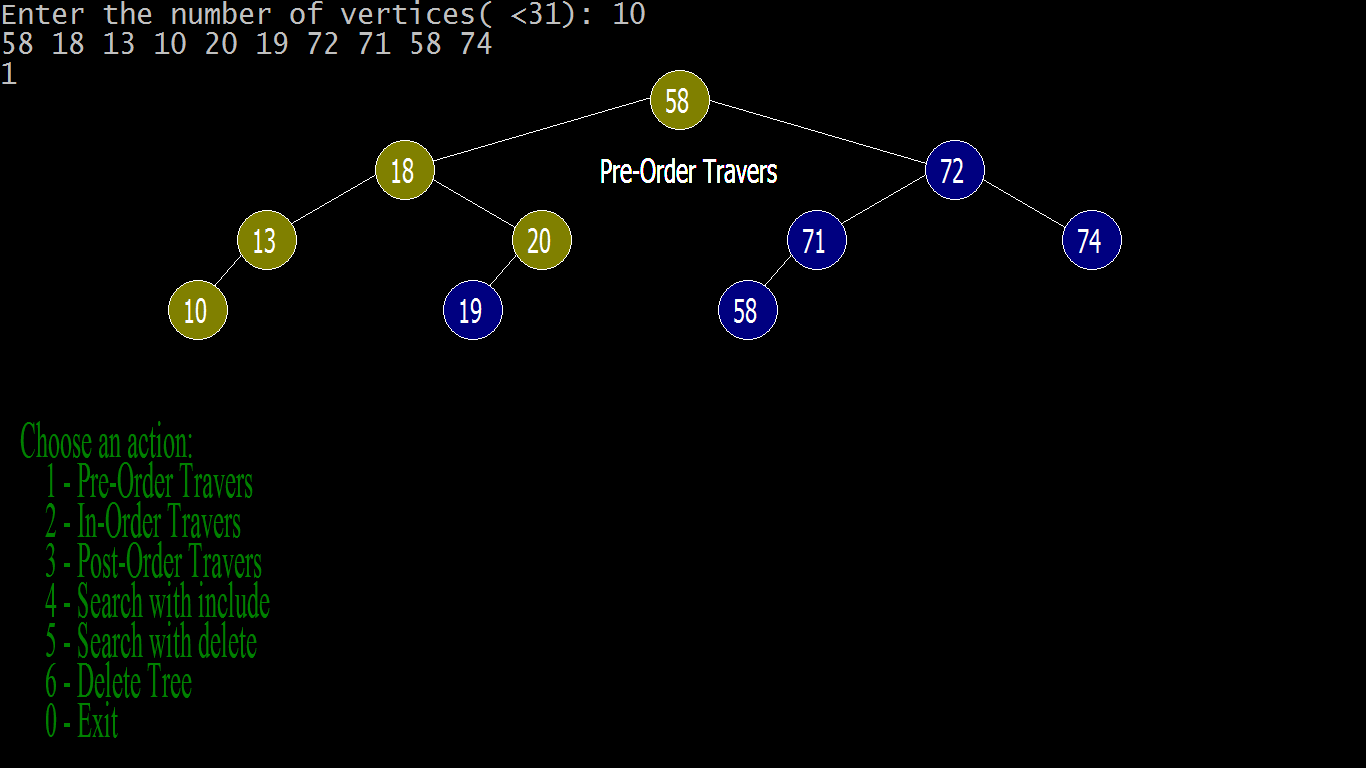
***int main()-****основная функция программы, запрашивающая данные от пользователя; обращаясь к нужным функциям, укомплектовывает и записывает данные в соответствующие переменные, в результате чего, реализует и отображает полную картину алгоритмов(перечисленных в постановке задачи) для работы с деревом.*

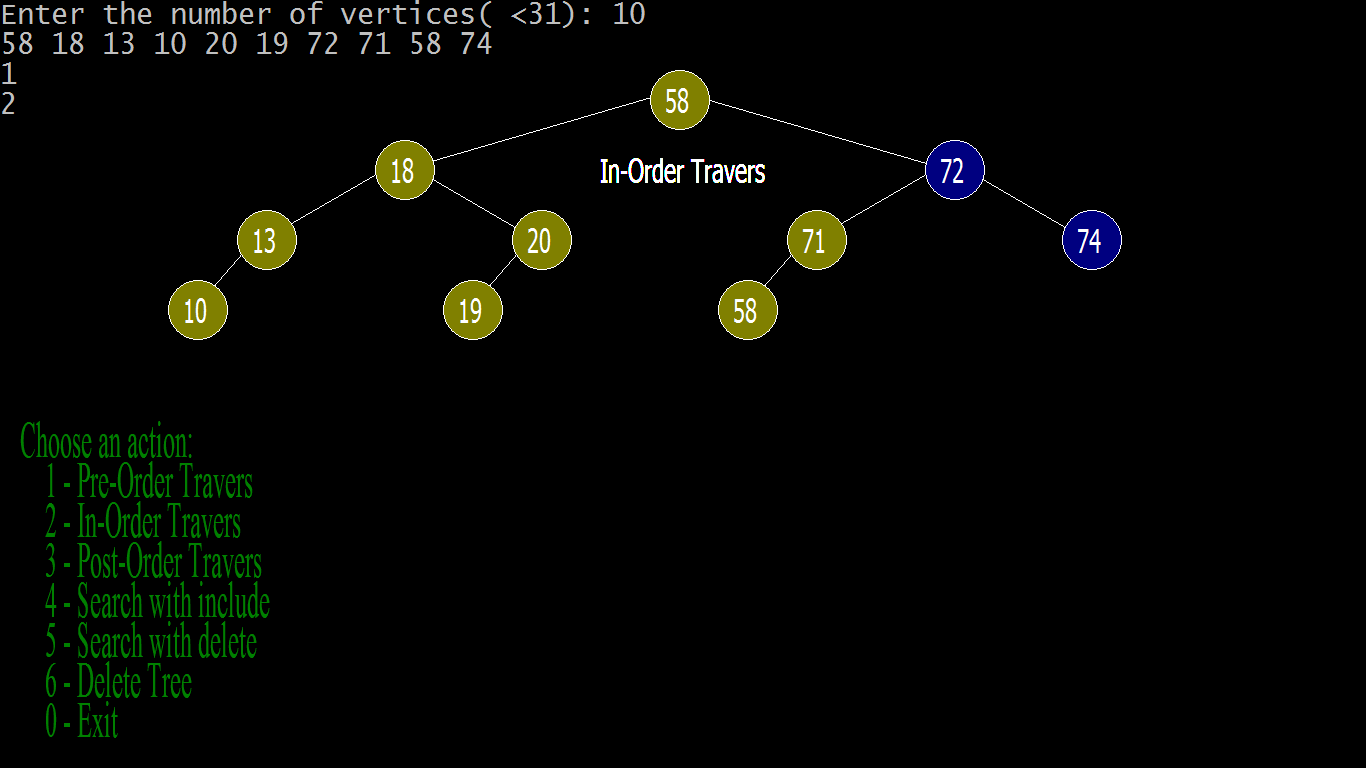
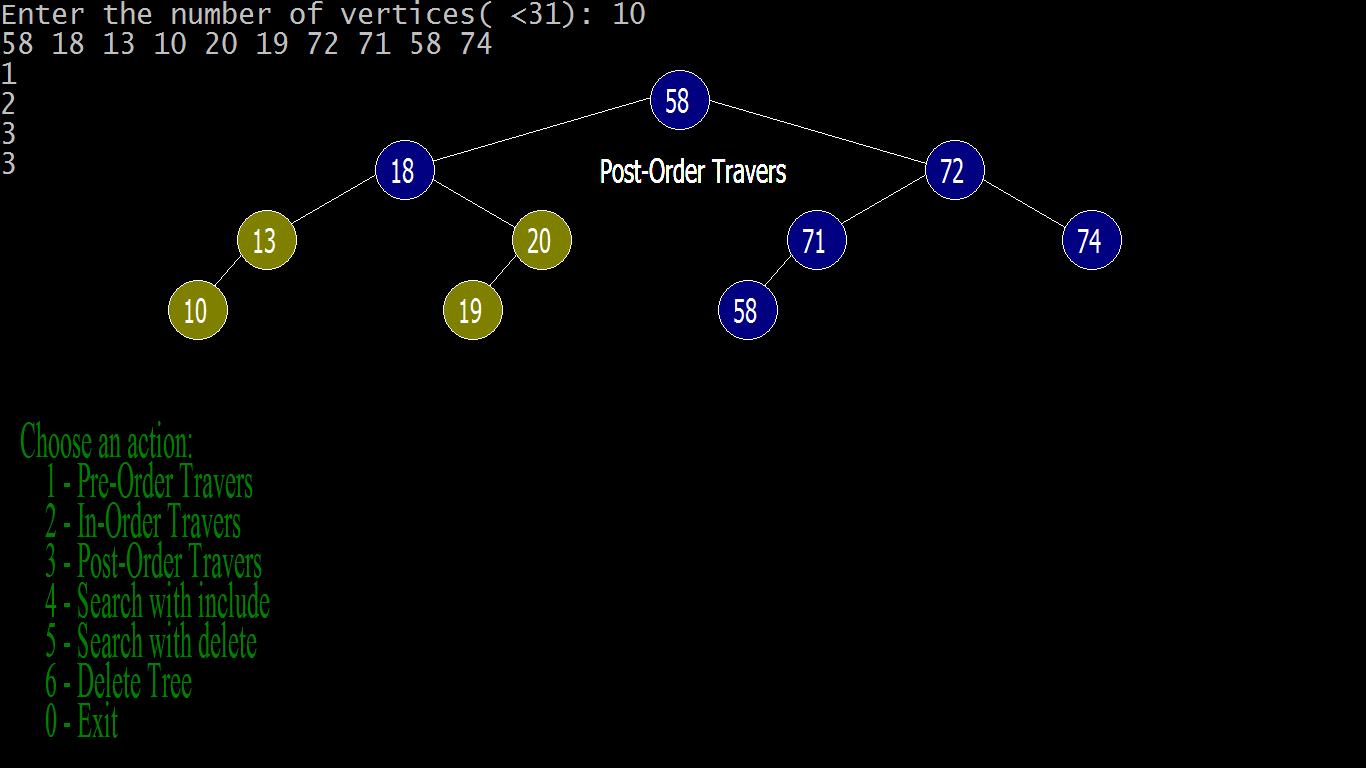
***4.Тестирование***

При запуске программы пользователю предлагается ввести количество вершин в дереве, после чего на экране отобразится идеально-сбалансированное дерево и список возможных действий для работы с ним:

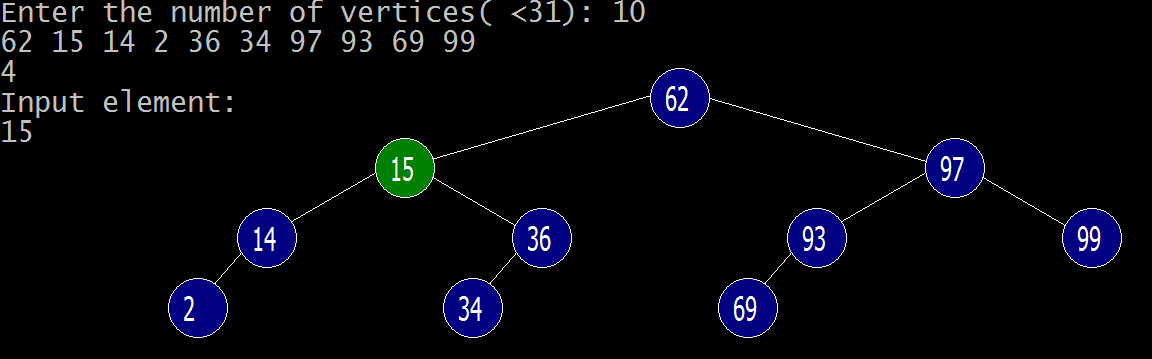
При выборе обхода дерева, будет графически «подсвечиваться» каждая вершина дерева, следуя порядку соответствующего обхода.

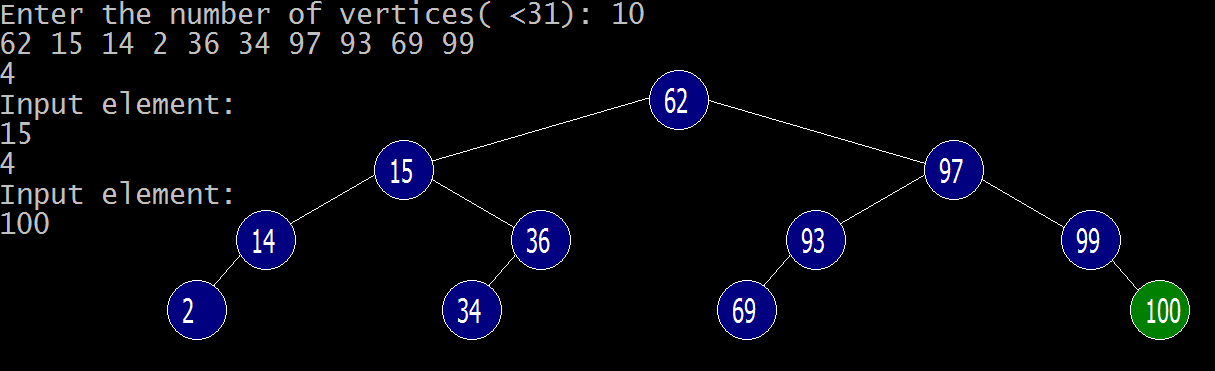
1. Прямой



1. Симметричный
2. Обратный

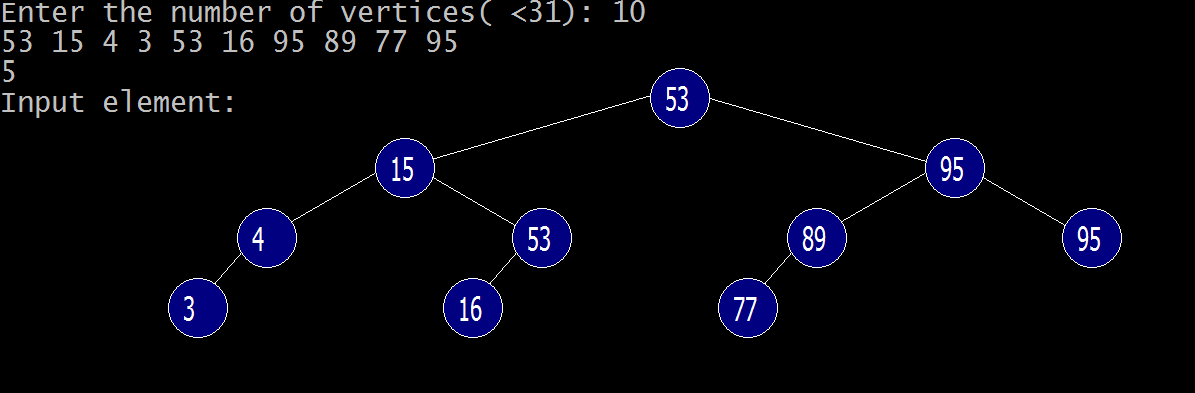
При выборе действия поиска с включением или с удалением на экране отобразится запрос о вводе вершины, после чего будет произведен поиск значения узла в дереве(если вершина есть в дереве, она «подсветится»).

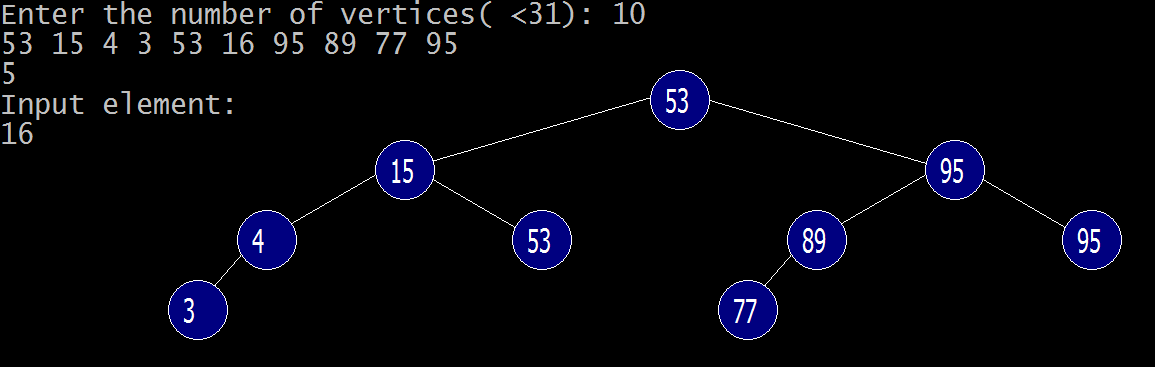
*1.Элемент существовал. При вводе подсветился.*

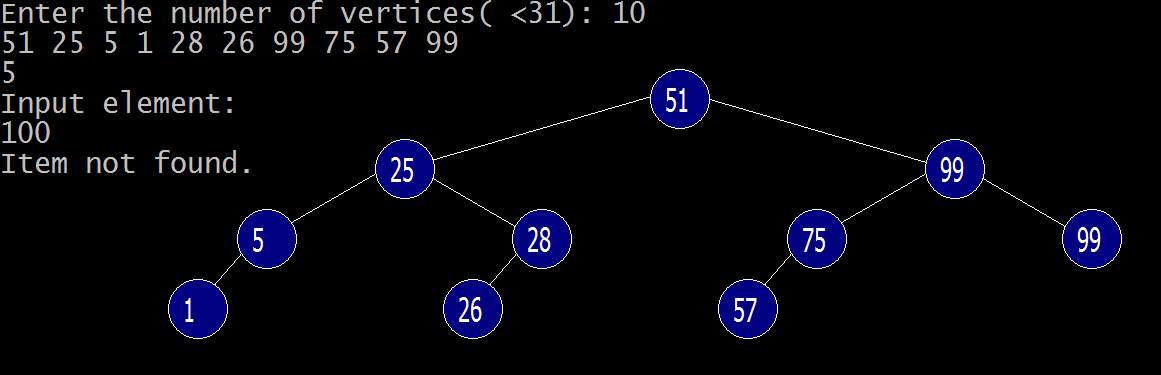
*2.Элемент был добавлен в дерево*

**Поиск с удалением.**

Если вершина с искомым значением будет найдена, то она удалится из дерева, в противном случае будет выведено сообщение о том, что данной вершины в дереве нет.

******

******

******

В результате тестирования программы, не было обнаружено отклонений от поставленной задачи, а это означает, что программа исправна и готова к использованию.

При выборе шестого действия дерево будет удалено, а при нажатие клавиши со значением ‘0’ программа завершит свою работу.