**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по курсовой работе**

**по дисциплине «Программирование»**

Тема: Бинарное дерево поиска

|  |  |  |
| --- | --- | --- |
| Студент гр. |  | Серебрякова А.К. |
| Преподаватель |  | Глущенко А.Г. |

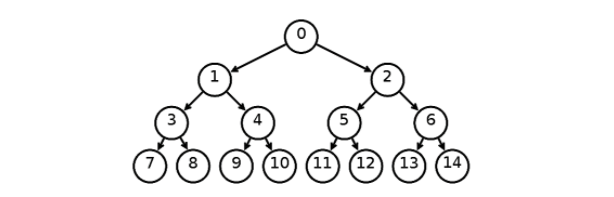
Санкт-Петербург

2021

## Понятие бинарного дерева

Бинарное дерево – это динамическая структура данных, состоящая из узлов, каждый из которых содержит кроме данных не более двух ссылок на различные бинарные деревья. На каждый узел имеется ровно одна ссылка. Начальный узел называется корнем дерева.

Узел, не имеющий поддеревьев, называется листом. Исходящие узлы называются предками, входящие – потомками. Высота дерева определяется количеством уровней, на которых располагаются его узлы.



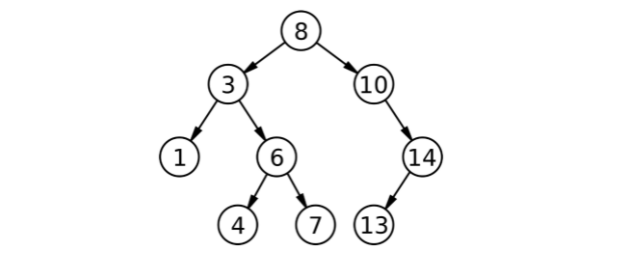
Дерево является рекурсивной структурой данных, поскольку каждое поддерево также является деревом. Действия с такими структурами лучше всего описывать с помощью рекурсивных алгоритмов.

Бинарные дерева нужны:

1. Если необходимо расположить информацию, связанную между собой некой иерархией. Примером является файловая система компьютера.
2. Если необходимо хранить данные, составленные в виде определённой структуры. Тогда хранение в виде бинарного дерева позволяет уменьшить скорость поиска данных и доступа к хранимой информации.
3. Если необходима высокая скорость добавления или удаления данных.
4. Если заранее неизвестен хранимый объем данных. Бинарные деревья, также, как и связанные списки, не имеют ограничения на количество узлов, поскольку узлы связаны указателями.

## Понятие бинарного дерева поиска

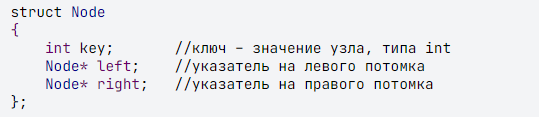
Бинарное дерево поиска – бинарное дерево, которое обладает дополнительными свойствами. Если дерево организовано таким образом, что для каждого узла все ключи левого поддерева меньше ключа этого узла, а все ключи его правого поддерева больше, то его можно называть бинарным деревом поиска.  
Иными словами, данные в бинарном дереве поиска хранятся в отсортированном виде. Одинаковые ключи не допускаются.



При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем.

Если искомое значение больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается.

Поскольку каждый корень дерева должен иметь три части, логичнее всего представить его в виде следующей рекурсивной структуры:



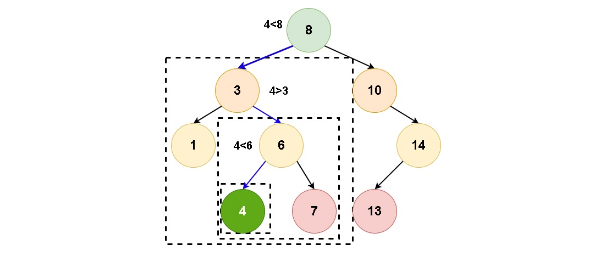
Сбалансированное бинарное дерево поиска – это бинарное дерево поиска с логарифмической высотой.

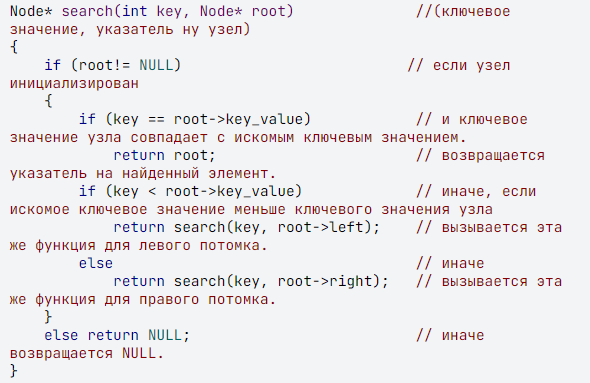
Для данного случая это правило необязательно, в отличие от AVL или красно-черных деревьев, где строгое определение оперирует разницей глубины самого глубоко и самого неглубокого листа и отношением глубины самого глубоко и самого неглубокого листа соответственно.

Сбалансированное бинарное дерево поиска применяется, когда необходимо осуществлять быстрый поиск элементов, чередующийся со вставками новых элементов и удалением существующих.

## Поиск элемента с заданным ключом

Для поиска элемента в бинарном дереве поиска можно воспользоваться функцией, которая принимает в качестве параметра заданное значение ключа. Для каждого узла функция сравнивает значение его ключа со значением заданного ключа. Если значения ключей одинаковы, то функция возвращает текущий узел. В противном случае, функция вызывается рекурсивно: для левого поддерева, если заданный ключ меньше ключа узла, для правого поддерева, если заданный ключ больше ключа узла.





## Вставка нового узла

При вставке элемента в бинарное дерево поиска необходимо учитывать два случая: Дерево пустое или

Дерево непустое.

Если дерево пустое, то создается новый узел, который добавляется в дерево. Если дерево не пустое, то значение ключа добавляемого элемента сравнивается со значением ключа в узле, начиная от корня.

Так как одинаковые ключи не допускаются, перед вставкой нового элемент нужно проверить, нет ли уже в бинарном дереве такого элемента. Для этого необходимо начать обход дерева с корневого узла и проверить, не превосходит ли значение корневого узла добавляемого значения.

Если корневой узел больше добавляемого элемента, то необходимо переместиться в левое дочернее дерево. В противном случае – в правое. После выполнения данной проверки можно переходить к добавлению узла в бинарное дерево.

## Удаление узла из дерева

С удалением узла дело обстоит немного иначе. Существует несколько возможных ситуаций:

1)      у узла нет наследников (узел является листом);  
2)      у узла имеется наследник, притом только один;  
3)      у узла есть оба наследника.

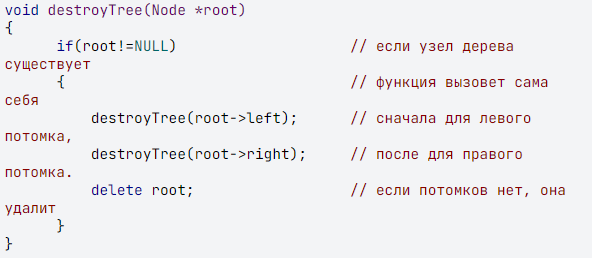
Если у узла нет наследника, то необходимо просто удалить этот узел, а у его родителя обнулить указатель на него. Это самая простая ситуация.

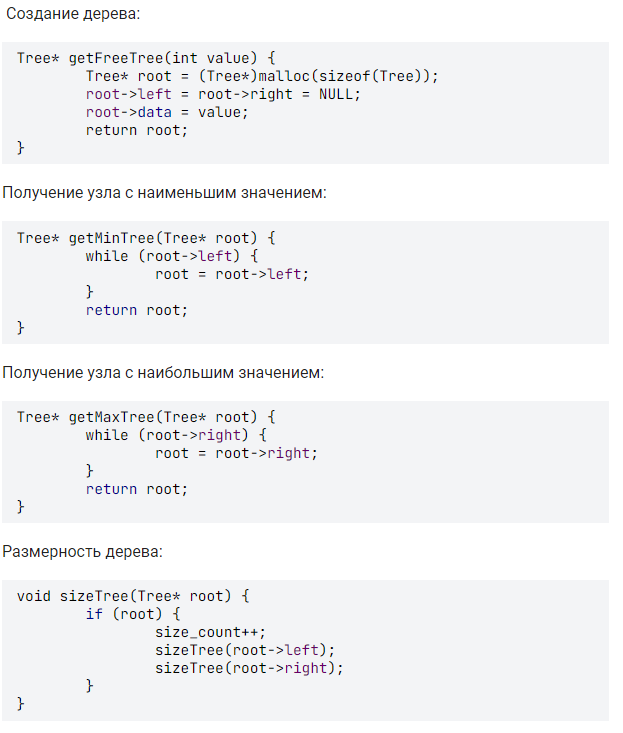
Все слегка усложняется, когда у удаляемого узла имеются наследники.

Если же у удаляемого два наследника, то узел не удаляется, а его значение заменяется на максимум левого поддерева. После этого удаляется максимум левого поддерева.

Максимум левого поддерева имеет не более одного наследника, так что он удаляется просто, аналогично ситуации, рассмотренной выше.Функция вставки аналогична функции поиска: необходимо пройти по дереву и вставить узел в нужное место. Для этого следует выбрать левое или правое поддерево корневого узла, а затем рекурсивно перемещаться по выбранному поддереву до тех пор, пока не будет найдена позиция для вставки узла.

 Если же необходимо удалить дерево, то необходимо реализовать рекурсивную функцию удаления:





## Обходы дерева

Граф – это множество вершин и ребер. Ребро – это связь между двумя вершинами. Количество возможных ребер в графе квадратично зависит от количества вершин (для понимания можно представить турнирную таблицу сыгранных матчей).

Дерево – это связный граф без циклов. Связность означает, что из любой вершины в любую другую существует путь по ребрам. Отсутствие циклов означает, что данный путь – единственный.

Обход графа – это систематическое посещение всех его вершин по одному разу каждой. Существует два вида обхода графа:

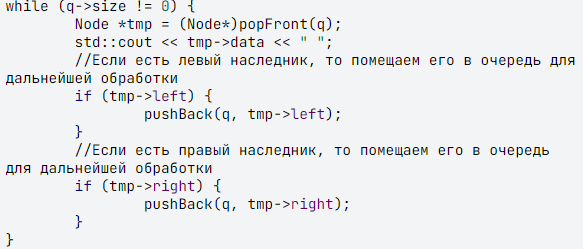
 1) поиск в глубину;

 2) поиск в ширину.

Существует множество задач, которые выполняются на структурах данных, представленных в виде бинарного дерева. Одна из самых распространенных – выполнение заданной операции с каждым элементом дерева. Чтобы отредактировать какие-либо элементы дерева, нужно совершить его обход.

Поиск в глубину идет из начальной вершины, посещая еще не посещенные вершины без оглядки на удаленность от начальной вершины. Алгоритм поиска в глубину по своей природе является рекурсивным. Для эмуляции рекурсии в итеративном варианте алгоритма применяется структура данных «стек».

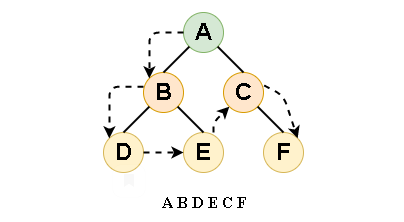
Поиск в ширину идет из начальной вершины, посещает сначала все вершины, находящиеся на расстоянии одного ребра от начальной, потом посещает все вершины на расстоянии двух ребер от начальной и так далее. Алгоритм поиска в ширину является по своей природе нерекурсивным (итеративным). Для его реализации применяется структура данных «очередь» (FIFO).



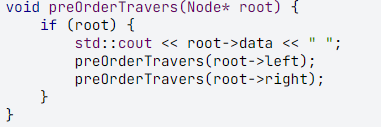
Обходу в ширину в графе соответствует обход по уровням бинарного дерева. При данном обходе идет посещение узлов по принципам сверху вниз и слева направо. Обходу в глубину в графе соответствуют три вида обходов бинарного дерева: прямой, симметричный и обратный.

## Прямой обход - сверху внизу

Прямой обход идет в следующем порядке: корень, левый потомок, правый потомок.

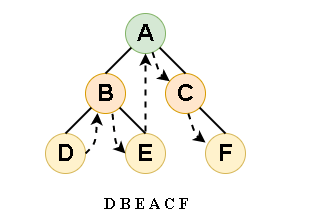


1. Обход начинается с корня дерева – узел А;
2. Далее выполняется переход вниз, сначала к левому поддереву;
3. Выполняется переход к левому потомку узла А – узел B;
4. Выполняется переход к левому потомку узла В – узел D;
5. Узел D является листом, то есть ниже двигаться нет возможности;
6. Выполняется переход к правому потомку узла В – узел E;
7. Левое поддерево узла А полностью пройдено, начинается обход правого поддерева;
8. Выполняется переход к правому потомку узла А – узел С;
9. Выполняется переход к правому потомку узла С – узел F, достигнут последний узел дерева, обход завершен.

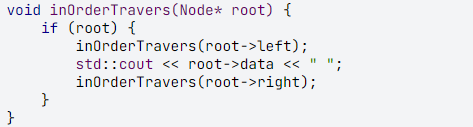


## Симметричный обход - слева направо

Симметричный обход — левый потомок, корень, правый потомок. Для бинарных деревьев поиска симметричный обход проходит все узлы в отсортированном порядке.



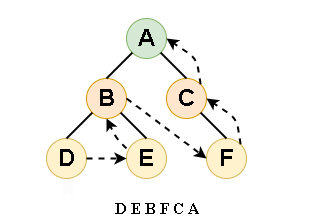
1. Обход начинается с самого нижнего левого листа – узел D;
2. Выполняется переход к родителю листа D – узел B;
3. Выполняется переход к правому потомку узла В – узел Е;
4. Узел В и все его потомки пройдены, поэтому необходимо двигаться вверх;
5. Выполняется переход к корню дерева – узел А;
6. Выполняется переход к правому потомку узла А – узел С;
7. Выполняется переход к правому потомку узла С – узел F, достигнут последний узел дерева, обход завершен.



Если мы хотим посетить узлы в обратно отсортированном порядке, то в коде рекурсивной функции симметричного обхода следует поменять местами правого и левого потомка.

## ****Обратный**** ****обход**** – ****снизу-вверх****

Обратный – левый потомок, правый потомок, корень.



1. Обход начинается с самого нижнего левого листа – узел D;
2. Выполняется переход к узлу Е;
3. Выполняется переход к родителю узлов D и E – узел B;
4. Левое поддерево полностью пройдено. Дальнейший обход через корень дерева игнорируется, выполняется переход к узлу F;
5. Выполняется переход к родителю узла F – узел С;
6. Выполняется переход к родителю узла С – узел А;
7. Узел А является корнем дерева, обход завершен.

## Курсовая работа вариант №2 - Бинарное дерево поиска

**Цели работы**: изучение свойств и организации деревьев как структуры данных; получение практических навыков в работе с бинарным деревом поиска; определение преимуществ и недостатков структуры данных вида дерева; проведение сравнительной характеристики скорости вставки, удаления и поиска элемента в различных структурах данных.

Необходимо реализовать программу, которая выполняет следующие действия.

1.   Формирование бинарного дерева из N элементов:

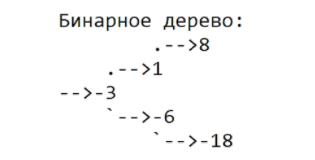
a) пользователь вводит количество элементов N бинарного дерева, которое автоматически заполняется случайными числами (–99 до 99);

б) пользователь вводит в консоль элементы массива, N определяется автоматически по количеству введенных элементов;

в) элементы считываются с файла, в котором хранится массив из чисел, N – количество элементов в файле.

Определение скорости формирования бинарного дерева.

2.   Вывод в консоль и файл tree бинарного дерева. Бинарное дерево должно иметь подобный вид:



3.   Определение скорости вставки, удаления и получения элемента дерева. В отчете сравните скорость работы бинарного дерева с линейной структурой (двусвязным списком или динамическим массивом) и сделайте выводы.

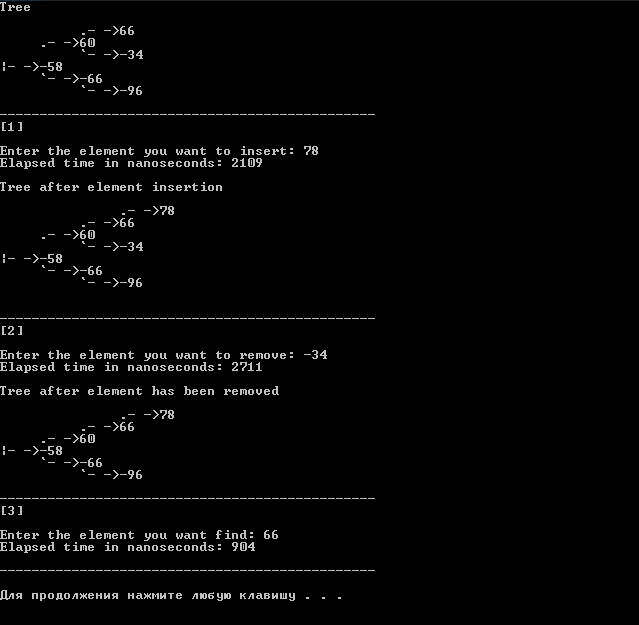
4. Прямой обход, обратный обход и обход в ширину бинарного дерева.

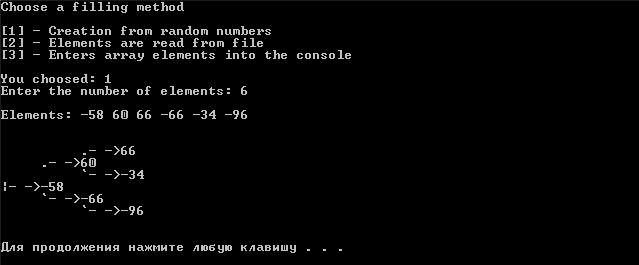
5.  Генерация заданий к практической работе по бинарным деревьям. Необходимо сгенерировать задания в файл output\_task в количестве вариантов, которые введет пользователь. В файл output\_key необходимо вывести короткие ответы к заданиям. В файл output\_ans необходимо вывести развернутые ответы к заданиям. Должны быть представлены задания на создание бинарного дерева поиска из массива значений, удаление и вставка элементов в бинарное дерево поиска.

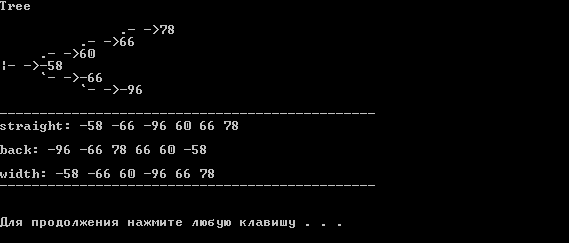
**Выполнение работы.**

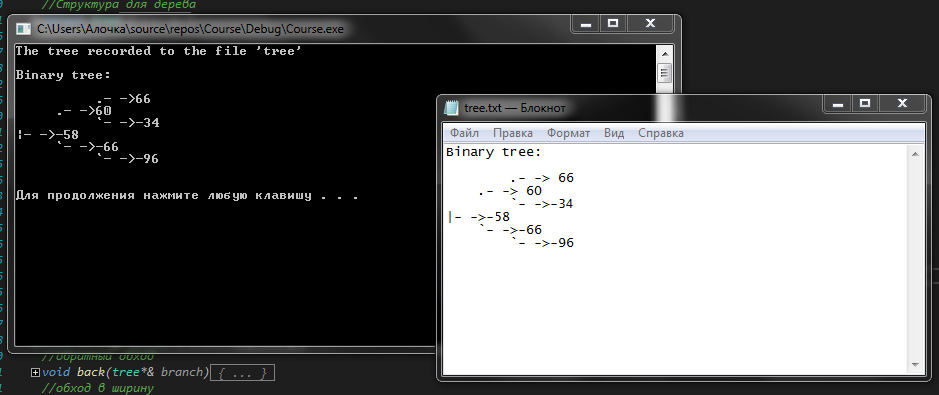
Исходный код программы представлен в приложении А

Примеры работы программы









Приложение А

**КОД ПРОГРАММЫ**

**Ссылка на github**

https://github.com/xXchamomileteaXx/\_NAME\_/blob/main/Course.cpp