МГТУ им. Н. Э. Баумана

ИУ 7 – 32

Отчет о лабораторной работе №7

«**Деревья**»

Вариант №8

Исаев Д.С.

# Цель работы

Получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов.

# Условие задачи

Построить дерево в соответствии со своим вариантом задания. Вывести его на экран в виде дерева. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов. Сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления.

В текстовом файле содержатся целые числа. Построить двоичное дерево из чисел файла. Вывести его на экран в виде дерева. Используя процедуру, определить количество узлов дерева на каждом уровне. Добавить число в дерево и файл. Сравнить время добавление в указанные структуры.

**Описание**

**Дерево**

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим».

Дерево с базовым типом *Т* определяется рекурсивно либо как пустая структура (пустое дерево), либо как узел типа *Т* с конечным числом древовидных структур этого же типа, называемых поддеревьями.

Деревья используются при построении организационных диаграмм, анализе электрических цепей, для представления синтаксических структур в компиляторах программ, для представления структур математических формул, организации информации в СУБД и, кроме того, для более эффективного извлечения данных.

Деревья бывают двоичными, B-деревьями, AVL, красно-черными, идеально сбалансированными и другими.

Основные операции с деревьями: обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

Обход (посещение) вершин дерева можно осуществить

следующим образом (рис):

* сверху вниз: R,A,B (префиксный обход)
* слева направо: A,R,B (инфиксный обход)
* снизу вверх: A,B,R (постфиксный обход)

Для реализации алгоритмов поиска используются деревья двоичного поиска. Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше. Это свойство называется характеристическим свойством дерева двоичного поиска и выполняется для любого узла, включая корень. С учетом этого свойства поиск узла в двоичном дереве поиска можно осуществить, двигаясь от корня в левое или правое поддерево в зависимости от значения ключа поддерева.

**Реализация**

Представление дерева в виде связей с предками:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № вершина | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Родитель | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 6 | 6 | 6 |

Пример представления этого же дерева в виде связного списка сыновей приведен на рис 2:

1

2

3

4

5

6

7

8

9

100

2

4

3

5

6

7

8

9

10

Описание структуры разреженной матрицы в программе:

**Узел дерева:**

class Node<T>

    {

        private Node<T> left, right;

        private Node<T> parent;

        private T key;

}

**Само дерево:**

class BTree<T> where T : IComparable

    {

        protected Node<T> Root;

…

}

**Результаты**

В результате работы программы были получены следующие данные:

*Сравнение в зависимости от глубины дерева:*

**Глубина 10:**

* Время поиска = 0,01
* Время сортировки = 0,04

**Глубина 20:**

* Время поиска = 0,01
* Время сортировки = 0,36

**Глубина 50:**

* Время поиска = 0,01
* Время сортировки = 2,72.

*Сравнение в зависимости от степени ветвления дерева:*

**Плохая степень ветвления:**

* Время поиска = 0,05
* Время сортировки = 0,10

**Отличная степень ветвления:**

* Время поиска = 0,01
* Время сортировки = 6,69

**Сравнение времени добавления:**

* В файл – 9,54 с
* В бинарное дерево – 1,32 с

Из приведённых данных видно, что время хорошо сбалансированном дереве значительно меньше, чем в несбалансированном.

Также видно, что время работы с внешней памятью (запись в файл) в 10 раз медленнее работы с ОЗУ.

Со временем при наполнении дерева, если элементы поступают не в случайном порядке, дерево «дестабилизируется» и время поиска в нем стремится к времени поиска в обычном односвязном списке.

**Вывод:**

Двоичные деревья поиска позволяют осуществить быстрый поиск, однако необходимо как-то осуществлять их балансировку, иначе время поиска заметно ухудшается. Запись во внешнюю память значительно медленнее работы с ОЗУ.