Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

По дисциплине: “Управление информационными ресурсами”

Лабораторная работа №1

“Исследование нелинейных структур данных. AVL-деревья”

Выполнил:

ст.гр. ИС/б-17-2

Долженко И.А.

Проверил:

Дрозин А.Ю.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

Исследовать возможности применения AVL-деревьев – для хранения, поиска и обработки информации. Приобрести практические знания использования классов, реализующих AVL-деревья. Оценить эффективность использования AVL-деревья по сравнению с бинарными деревьями поиска.

2 ПОСТАНОВКА ЗАДАЧИ

1. На языке программирования С# в среде программирования Visual Studio, с использованием классов, реализующих AVL-деревья реализовать Windows-приложение, обеспечивающее выполнения следующих функций:

* Построение AVL-дерева на основании данных, расположенных в файле, содержащем построчно записи разделенные символом табуляции;
* Отображение на визуальной форме AVL-дерева;
* Предоставление интерфейса пользователю для выполнения операций добавления, удаления, изменения и поиска (по ключевому полю) элементов AVL-дерева, с отображением результатов выполнения операций на визуальной форме;
* Отображение времени выполнения операций добавления, удаления, изменения и поиска данных по заданному пользователем значению ключевого поля.

2. С использованием разработанной программы выполнить исследования структур данных AVL-дерева:

* Построить AVL- дерево для первых N1 строк таблицы;
* Выполнить по 5 раз операции добавления, удаления и поиска информации (по случайным значениям ключевого поля), фиксируя в отчете время выполнения операций;
* Вычислить среднее время выполнения операций добавления, удаления и поиска информации (по ключевому полю).

Вариант 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ключевое поле | N1 | N2 | N3 | N4 | N5 |
| Счёт | 10 | 200 | 1400 | 5000 | 9000 |

3 ТЕКСТ ПРОГРАММЫ

using System.Collections.Generic;

namespace WindowsFormsApp1

{

class AVL

{

class Node

{

public int data;

public Node left;

public Node right;

public Node(int data)

{

this.data = data;

}

}

Node root;

public AVL()

{

}

public void Add(int data)

{

Node newItem = new Node(data);

if (root == null)

{

root = newItem;

}

else

{

root = RecursiveInsert(root, newItem);

}

}

private Node RecursiveInsert(Node current, Node n)

{

if (current == null)

{

current = n;

return current;

}

else if (n.data < current.data)

{

current.left = RecursiveInsert(current.left, n);

current = balance\_tree(current);

}

else if (n.data > current.data)

{

current.right = RecursiveInsert(current.right, n);

current = balance\_tree(current);

}

return current;

}

private Node balance\_tree(Node current)

{

int b\_factor = balance\_factor(current);

if (b\_factor > 1)

{

if (balance\_factor(current.left) > 0)

current = RotateLL(current);

else

current = RotateLR(current);

}

else if (b\_factor < -1)

{

if (balance\_factor(current.right) > 0)

current = RotateRL(current);

else

current = RotateRR(current);

}

return current;

}

public void Delete(int target)

{

root = Delete(root, target);

}

private Node Delete(Node current, int target)

{

Node parent;

if (current == null) return null;

else

{

if (target < current.data)

{

current.left = Delete(current.left, target);

if (balance\_factor(current) == -2)

{

if (balance\_factor(current.right) <= 0)

current = RotateRR(current);

else

current = RotateRL(current);

}

}

else if (target > current.data)

{

current.right = Delete(current.right, target);

if (balance\_factor(current) == 2)

{

if (balance\_factor(current.left) >= 0)

current = RotateLL(current);

else

current = RotateLR(current);

}

}

else

{

if (current.right != null)

{

parent = current.right;

while (parent.left != null)

{

parent = parent.left;

}

current.data = parent.data;

current.right = Delete(current.right, parent.data);

if (balance\_factor(current) == 2)

{

if (balance\_factor(current.left) >= 0)

current = RotateLL(current);

else

current = RotateLR(current);

}

}

else

{

return current.left;

}

}

}

return current;

}

public bool Find(int key)

{

if (Find(key, root).data == key) return true;

else return false;

}

private Node Find(int target, Node current)

{

if (target < current.data)

{

if (target == current.data) return current;

else return Find(target, current.left);

}

else

{

if (target == current.data) return current;

else return Find(target, current.right);

}

}

private int max(int l, int r)

{

return l > r ? l : r;

}

private int getHeight(Node current)

{

int height = 0;

if (current != null)

{

int l = getHeight(current.left);

int r = getHeight(current.right);

int m = max(l, r);

height = m + 1;

}

return height;

}

private int balance\_factor(Node current)

{

int l = getHeight(current.left);

int r = getHeight(current.right);

int b\_factor = l - r;

return b\_factor;

}

private Node RotateRR(Node parent)

{

Node pivot = parent.right;

parent.right = pivot.left;

pivot.left = parent;

return pivot;

}

private Node RotateLL(Node parent)

{

Node pivot = parent.left;

parent.left = pivot.right;

pivot.right = parent;

return pivot;

}

private Node RotateLR(Node parent)

{

Node pivot = parent.left;

parent.left = RotateRR(pivot);

return RotateLL(parent);

}

private Node RotateRL(Node parent)

{

Node pivot = parent.right;

parent.right = RotateLL(pivot);

return RotateRR(parent);

}

}

}

4 РЕЗУЛЬТАТЫ

1. Загрузим исходные данные и выполним построение графа.

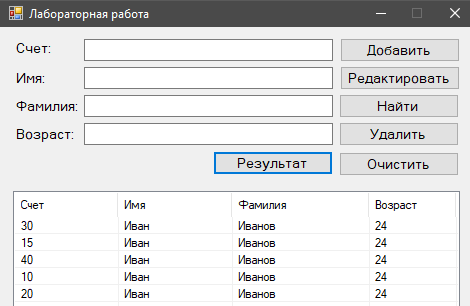


Рисунок 1 – Исходные данные

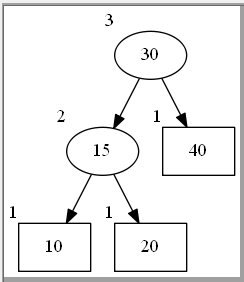


Рисунок 2 – Полученный граф в программе GraphViz

2. Добавим узел 5, который приведет к одиночному повороту вправо текущего дерева.

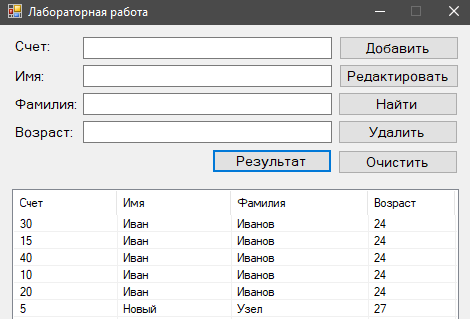


Рисунок 3 – Добавление нового узла

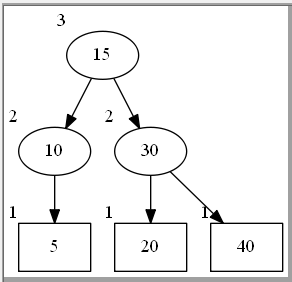


Рисунок 4 – Дерево после одиночного поворота вправо

3. Добавим новые узлы 13, 2, 14, 12, 11, которые приведут к двойному повороту вправо текущего дерева.

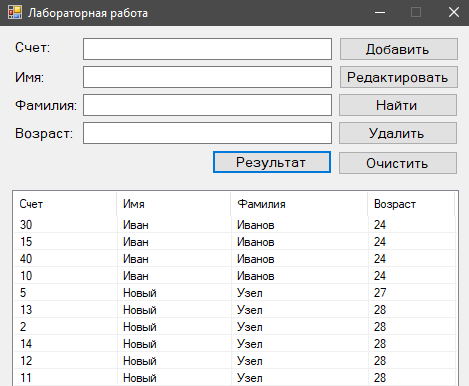


Рисунок 5 – Добавление новых узлов

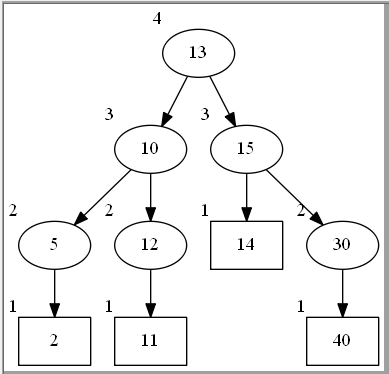


Рисунок 6 – Дерево после двойного поворота вправо

4. Измерим показатели времени добавления, удаления, поиска и занесем их в таблицу.

Таблица 1. Результаты проделанной работы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество записей | № | AVL – дерево | | |
| Добавление | Удаление | Поиск |
| N1=10 | 1 | 11 | 13 | 10 |
| 2 | 18 | 15 | 11 |
| 3 | 14 | 14 | 10 |
| 4 | 13 | 20 | 10 |
| 5 | 18 | 13 | 9 |
| **Ср.** | **15** | **15** | **10** |
| N2=200 | 1 | 86 | 135 | 11 |
| 2 | 113 | 84 | 21 |
| 3 | 77 | 111 | 15 |
| 4 | 122 | 74 | 11 |
| 5 | 86 | 85 | 11 |
| **Ср.** | **97** | **98** | **14** |
| N3=1400 | 1 | 703 | 788 | 12 |
| 2 | 1093 | 733 | 12 |
| 3 | 1054 | 1100 | 12 |
| 4 | 727 | 524 | 35 |
| 5 | 555 | 493 | 13 |
| **Ср.** | **826** | **727** | **17** |
| N4=5000 | 1 | 2934 | 3270 | 36 |
| 2 | 3188 | 2540 | 31 |
| 3 | 2400 | 2347 | 24 |
| 4 | 2373 | 1964 | 20 |
| 5 | 2435 | 2385 | 17 |
| **Ср.** | **2666** | **2501** | **27** |
| N5=9000 | 1 | 4830 | 3777 | 21 |
| 2 | 8288 | 4195 | 48 |
| 3 | 5234 | 9705 | 14 |
| 4 | 6938 | 5039 | 22 |
| 5 | 7403 | 5477 | 15 |
| **Ср.** | **6538** | **5638** | **24** |

ВЫВОД

В ходе выполнения лабораторной работы была написана программа, реализующая построение AVL деревьев на основании данных, расположенных в файле. Был создан интерфейс пользователя для выполнения операций добавления, удаления, изменения и поиска (по ключевому полю) элементов AVL-дерева, с отображением результатов выполнения операций на визуальной форме в программе GraphViz. Было выполнено исследование структур данных AVL дерева и вычислено среднее время выполнения операций добавления, удаления и поиска информации.