ММИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: БДП.

Студент гр. 9384 Мосин К.К.

Преподаватель Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Реализовать структуру бинарного дерева поиска, далее БДП, заполнить ее и записать в файл.

Задание.

ВАРИАНТ 11.

БДП: случайное БДП с рандомизацией.

По заданной последовательности элементов Elem построить структуру данных определенного типа - БДП или хеш-таблицу;

Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран.

Выполнение работы.

Для реализации БДП был разработан класс Tree со следующими методами:

public-методы:

Tree();

insert();

print();

write();

Private-методы:

insert_root();

rotate_right();

rotate_left();

get_size();

update_size();

Класс Tree тестировался на примере целочисленного типа int.

Каждый узел хранит значение, поданное на вход, количество узлов, входящих в данный узел, и указатели на левый и правый узлы. Для распределения данных использовались два метода вставок - бинарная вставка и вставка в корень. Бинарная вставка предполагает расположение поданного

значения от дерева: если текущее значение больше, то идет перенаправление на левого ребенка, иначе на правого. Вставка в корень предполагает бинарную с поворотом вокруг узла. Вставка в дерево определяется следующим образом: если rand%(size+1) равняется 0, то выбирается вставка в корень, иначе оставшаяся. Это означает, что вставка в корень подбирается тем реже, чем больше входных данных.

Тестирование.

Результаты тестирования представлены в табл. 2.

Таблица 2 – Результаты тестирования

Входные данные	Выходные данные
3 3 2 1	1,1 2,3 3,1 outfile: 1 2 3
5 1 2 3 4 5	1,2 2,1 3,5 4,1 5,2 outfile: 1 2 3 4 5
5 3 3 3 3 2	2,1 3,2 3,5 3,2 3,1 outfile: 2 3 3 3 3

Выводы.

В ходе выполнения лабораторный работы была реализовано БДП с рандомизацией.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include "tree.h"

int main(){
   srand(time(0));

   Tree<int> *tree;

   int count;
   std::cout << "Type count: ";
   std::cin >> count;
   if(count > 0){
      int temp;
      std::cout << "Type data: ";
      for(int i = 0; i < count; i++){
         std::cin >> temp;
         tree ? tree = tree->insert(tree, temp) : tree = new Tree<int>(temp);
      }
      tree->print();
      std::ofstream file("output.txt");
      file.is_open() ? tree->write(file) : throw std::runtime_error("file could not open");
      delete tree;
   }

   return 0;
}
```

Название файла: tree.h

```
#ifndef TREE_H
#define TREE_H

template <typename T>
class Tree{
public:
   Tree(T);
   Tree<T>* insert(Tree<T>*, T);
   void print(int = 0);
   void write(std::ofstream&);

private:
   Tree<T>* insert_root(Tree<T>* , T);
   Tree<T>* rotate_right(Tree<T>*);
   Tree<T>* rotate_left(Tree<T>*);
   int get_size(Tree<T>*);
```

```
    void update_size(Tree<T>*);

private:
    T data;
    int size;
    Tree<T>* left;
    Tree<T>* right;
};

template <typename T>
Tree<T>::Tree(T data) : data(data), size(1), left(nullptr), right(nullptr) {}

template <typename T>
Tree<T>* Tree<T>::insert(Tree<T>* ptr, T data){
    if(!ptr){
        return new Tree<T>(data);
    }
    if(rand()%(ptr->size+1) == 0){
        std::cout << data <<" random\n";
        return insert_root(ptr, data);
    }
    if(ptr->data > data){
        std::cout << data << " left\n";
        ptr->left = insert(ptr->left, data);
    }
    else{
        std::cout << data << " right\n";
        ptr->right = insert(ptr->right, data);
    }
    update_size(ptr);
    return ptr;
}

template <typename T>
void Tree<T>::print(int level){
    if(left){
        left->print(level+1);
    }
    for(int i = 0; i < level; i++){
        std::cout << '\t';
    }
    std::cout << data << ',' << size << std::endl;
    if(right){
        right->print(level+1);
    }
}

template <typename T>
void Tree<T>::write(std::ofstream& file){
    if(left){
        left->write(file);
    }
```

```
    file << data << " ";
    if(right){
        right->write(file);
    }
}

template <typename T>
Tree<T>* Tree<T>::insert_root(Tree<T>* ptr, T data){
    if(!ptr){
        std::cout << "create\n";
        return new Tree<T>(data);
    }
    if(ptr->data > data){
        ptr->left = insert_root(ptr->left, data);
        update_size(ptr);
        return rotate_right(ptr);
    }
    else{
        ptr->right = insert_root(ptr->right, data);
        update_size(ptr);
        return rotate_left(ptr);
    }
}

template <typename T>
Tree<T>* Tree<T>::rotate_right(Tree<T>* ptr){
    std::cout << "rotate_right\n";
    Tree<T>* temp = ptr->left;
    if(!temp){
        std::cout << "exit\n";
        return ptr;
    }
    ptr->left = temp->right;
    temp->right = ptr;
    temp->size = ptr->size;
    update_size(ptr);
    return temp;
}

template <typename T>
Tree<T>* Tree<T>::rotate_left(Tree<T>* ptr){
    std::cout << "rotate_left\n";
    Tree<T>* temp = ptr->right;
    if(!temp){
        return ptr;
    }
    ptr->right = temp->left;
    temp->left = ptr;
    temp->size = ptr->size;
    update_size(ptr);
    return temp;
}
```

```
template <typename T>
int Tree<T>::get_size(Tree<T>* ptr){
   if(!ptr){
      return 0;
   }
   return ptr->size;
}

template <typename T>
void Tree<T>::update_size(Tree<T>* ptr){
   ptr->size = get_size(ptr->left) + get_size(ptr->right) + 1;
}

#endif
```