

МИНОБРНАУКИ 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 и хеш-таблицы.

Студентка гр. 9381

Москаленко Е.М.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить и реализовать такую структуру данных, как случайное бинарное дерево поиска с рандомизацией. Написать функцию проверки вхождения в дерево определенного элемента.

Задание.

Вариант 10. БДП: случайное* БДП с рандомизацией; действие: 1+2a

1) По заданной последовательности элементов Elem построить структуру данных определённого типа – БДП или хеш-таблицу;

2) Выполнить одно из следующих действий:

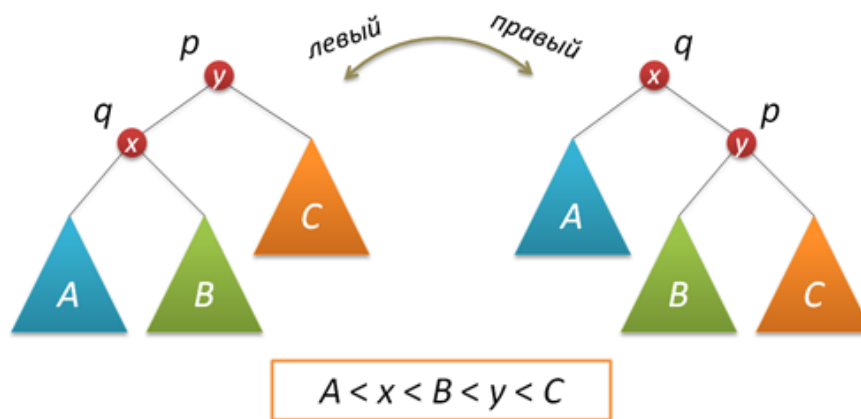
а) Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem, и если входит, то в скольких экземплярах. Добавить элемент e в структуру данных.

Предусмотреть возможность повторного выполнения с другим элементом.

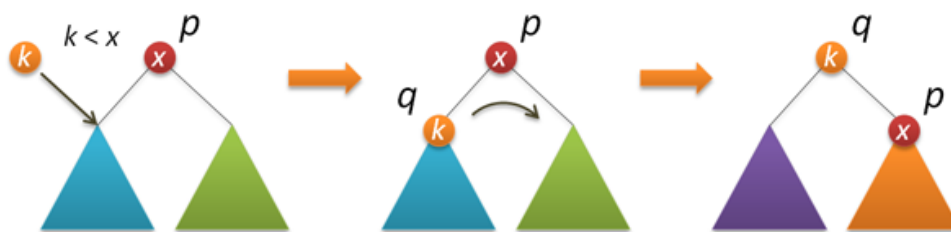
Основные теоретические положения.

Ключевая идея случайных БДП с рандомизацией состоит в чередовании обычной вставки в дерево поиска и вставки в корень. Чередование происходит случайным (рандомизированным) образом с использованием компьютерного генератора псевдослучайных чисел. Цель такого чередования – сохранить хорошие свойства случайного БДП в среднем и исключить (сделать маловероятным) появление «худшего случая» (поддеревьев большой высоты).

Рассмотрим операцию вставки в корень. Если дерево пусто, создаем новый узел со значением key , иначе, если $key(tree) > key$ то выполняем вставку в корень в левом поддереве $tree$ и выполняем правое вращение, иначе — вставку в корень в правом поддереве и левое вращение. Таким образом узел со значением key становится корнем дерева.



Левое и правое вращения



Вставка элемента в корень дерева с помощью вращений

Опишем теперь рандомизированную вставку значением `key` в дерево `tree`. Пусть в дереве имеется n узлов. Тогда будем считать, что после добавления еще одного узла любой узел с равной вероятностью может быть корнем дерева. Тогда, с вероятностью $1/(n+1)$ осуществим вставку в корень, иначе рекурсивно используем рандомизированную вставку в левое или правое поддерево в зависимости от значения ключа `key`.

Описание алгоритма.

Пользователю предлагается выбрать способ ввода данных: с консоли или с файла. В зависимости от выбора, вызывается функция `fillBdp` или `fillBdpFile`. В этих функциях дерево рекурсивно заполняется элементами с помощью метода рандомизированной вставки `insert` класса `Node`. С вероятностью $1/n$, где n – размер дерева до вставки, произойдет вставка элемента в корень дерева, и будет вызван метод `insertRoot`, использующий левое или правое вращение. Запись дерева выводится на консоль вызовом метода `recTreeprint`. Далее

пользователю необходимо выбрать действие: поиска элемента в дереве или выход из программы. При выборе первого действия, с консоли необходимо ввести искомый элемент. Вызывается метод класса *Node find*, который перебирает узлы дерева, сравнивая их значения с введенным элементом. Если равны – выводит информацию на консоль, значение узла меньше – переходит к правому поддереву, больше – к левому. Если такого элемента в дереве нет, то с помощью метода *insert* он добавляется в дерево.

Описание структур данных и функций.

Для реализации узла бинарного дерева поиска создан класс *Node*.

Поля класса Node:

int key - значение элемента узла

int size - размер дерева с корнем в данном узле

int amount - количество попыток ввести элемент со значением данного узла

*Node *left* - левое поддерево

*Node *right* - правое поддерево

Методы класса Node:

1) *Node(int key, Node *left = nullptr, Node *right = nullptr)* – конструктор класса

int key – значение добавляемого элемента

*Node *left* - указатель на левое поддерево

*Node *right* - указатель на правое поддерево

2) *int find(Node *p, int k)* – метод поиска узла дерева с заданным пользователем значением. Возвращает количество вхождений в дерево.

*Node *p* – указатель на корень дерева

int k - значение искомого элемента

- 3) *int getSize(Node* p)* – возвращает размер дерева, корнем которого является переданный в аргументах узел
*Node *p* – указатель на корень дерева
- 4) *void fixSize(Node* p)* – метод, корректирующий размер дерева (размер левого поддерева + размер правого поддерева + 1)
*Node *p* – указатель на корень дерева
- 5) *Node* rotateLeft(Node* p)* – метод левого вращения дерева, вызывается при вставке элемента в корень (из правого поддерева). Возвращает указатель на корень измененного дерева.
*Node *p* – указатель на корень дерева
- 6) *Node* rotateRight(Node* p)* – метод правого вращения дерева, вызывается при вставке элемента в корень (из левого поддерева). Возвращает указатель на корень измененного дерева.
*Node *p* – указатель на корень дерева
- 7) *Node* insert(Node* p, int k)* – выполняет с вероятностью $1/(n+1)$, где n – размер дерева в узлах, вставку в корень, а с вероятностью $1-1/(n+1)$ — рекурсивную вставку в правое или левое поддерево в зависимости от значения ключа в корне. Возвращает указатель на корень измененного дерева.
*Node *p* – указатель на корень дерева
int k - значение искомого элемента
- 8) *Node* insertRoot(Node* p, int k)* – вставка в корень дерева. Сначала рекурсивно вставляется новый ключ в корень левого или правого поддеревьев (в зависимости от результата сравнения с корневым

ключом) и выполняется правый (левый) поворот, который поднимает нужный узел в корень дерева.

*Node *p* – указатель на корень дерева

int k - значение искомого элемента

9) *Node* getLeft()* – возвращает указатель на корень левого поддерева.

10) *Node* getRight()* - возвращает указатель на корень правого поддерева.

11) *void recTreePrint(Node* p)* – рекурсивно печатает значения узлов БДП в КЛП обходе.

*Node *p* – указатель на корень дерева

12) *void print2DUtil(Node *root, int space)* – функция вывода на консоль элементов дерева (дерево, «лежащее на боку»).

Node root* – указатель на корень дерева

int space – количество пробелов (расстояние между уровнями)

Также реализованы функции:

1) *Node* fillBdp(Node* p)* – рекурсивное заполнение БДП при вводе с консоли. Считывается строка, введенная пользователем, все числа из нее, игнорируя пробелы, добавляются в вектор *arr*. Если строка неправильная – выводится замечание, программа прекращает работу. Затем все значения из *arr* с помощью метода класса *Node insert* добавляются в дерево. Функция возвращает указатель на корень измененного дерева.

*Node *p* – указатель на корень дерева

2) *Node* fillBdpFile(Node* p, string name)* – одна из двух функций рекурсивного заполнения БДП при считывании с файла. Если файла с переданным именем не существует, то возвращается пустое дерево.

Иначе в файле посчитывается количество чисел в строке, и возвращается значение функции *fillBdpFile(Node* p, string name)*.

*Node *p* – указатель на корень дерева

string name – полный путь до файла

- 3) ***Node* fillMas(Node* p, string name, int count)*** - одна из двух функций рекурсивного заполнения БДП при считывании с файла. Создается целочисленный массив для *count* элементов, затем в него считываются числа из файла, а после этого все значения из массива *x* с помощью метода класса *Node insert* добавляются в дерево. Функция возвращает указатель на корень измененного дерева.

*Node *p* – указатель на корень дерева

string name – полный путь до файла

int count – количество чисел в строке

- 4) ***void info()*** – вывод на консоль «меню» - действий, которые может выбрать пользователь

- 5) ***void choose()*** - вывод на консоль «меню вывода» - с консоли или из файла

- 6) ***Node* foo(Node* p, int k)*** – вызов функции считывания с консоли или из файла при помощи оператора *switch*. Функция возвращает указатель на корень измененного дерева

*Node *p* – указатель на корень дерева

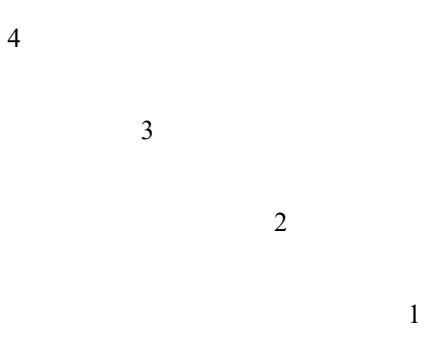
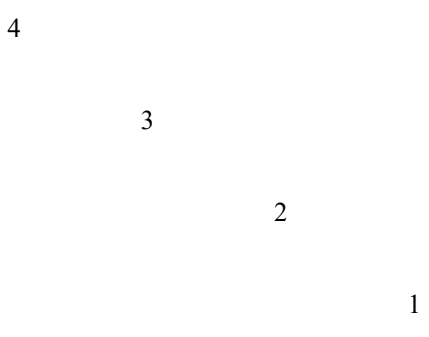
int k – переменная выбора, 1 или 2

- 7) ***void destroy(Node* p)*** – рекурсивная функция удаления дерева.

*Node *p* – указатель на корень дерева

Тестирование.

№	Входные данные	Вывод
1	<p>1</p> <p>1 2 3 4 5 6 7 8</p> <p>а</p> <p>Введите искомый элемент:</p> <p>3</p>	<p>Введите числа для построения</p> <p>дерева.</p> <p>1 2 3 4</p> <p>Добавлен элемент 1</p> <p>1 //</p> <p>Вставляем узел со значением 2 в корень дерева!</p> <p>2 > значения узла 1, поэтому переходим к правому поддереву</p> <p>Осуществляем левый поворот</p> <p>2</p> <p>1</p> <p>Добавлен элемент 2</p> <p>2 1 ///</p> <p>Вставляем узел со значением 3 в корень дерева!</p> <p>3 > значения узла 2, поэтому переходим к правому поддереву</p> <p>Осуществляем левый поворот</p> <p>3</p> <p>2</p> <p>1</p> <p>Добавлен элемент 3</p> <p>3 2 1 ////</p> <p>Вставляем узел со значением 4 в корень дерева!</p> <p>4 > значения узла 3, поэтому переходим к правому поддереву</p>

		<p>Осуществляем левый поворот</p>  <p>Добавлен элемент 4 4 3 2 1 // // //</p> <p>Получившееся дерево:</p>  <p>Введите:</p> <p>a - Для поиска элемента в БДП b - Для выхода из программы</p> <p>a</p> <p>Введите искомый элемент:</p> <p>3</p> <p>Значение искомого узла меньше 4, переходим к левому поддереву</p> <p>Найден узел с совпадающим значением!</p> <p>3 содержится в дереве в количестве 1</p> <p>Вставляем узел со значением 3 в корень дерева!</p> <p>3 < значения узла 4, поэтому переходим к левому поддереву</p> <p>Осуществляем правый поворот</p>
--	--	--

		<p>4</p> <p>3</p> <p>2</p> <p>1</p> <p>Добавлен элемент 3</p> <p>3 2 1 /// 4 //</p>
2	<p>1</p> <p>313 231 232 24 535 24 536</p> <p>Введите искомый элемент:</p> <p>24</p> <p>Введите искомый элемент:</p> <p>313</p>	<p>Найден узел с совпадающим значением!</p> <p>24 содержится в дереве в количестве 2</p> <p>24 < значения узла 536, поэтому переходим к левому поддереву</p> <p>Вставляем узел со значением 24 в корень дерева!</p> <p>Добавлен элемент 24</p> <p>536 24 / 535 231 / 313 232 // // //</p> <p>Найден узел с совпадающим значением!</p> <p>313 содержится в дереве в количестве 1</p> <p>Еще одна попытка вставить узел со значением 313</p> <p>Добавлен элемент 313</p> <p>536 24 / 535 231 / 313 232 // // //</p>
3	<p>2</p> <p>/Users/elizaveta/test.txt</p> <p>Добавлен элемент 1</p> <p>Введите искомый элемент:</p> <p>1</p>	<p>1 содержится в дереве в количестве 1</p>
4	<p>1</p> <p>adas 13 dfa</p>	<p>Строка имеет неправильный формат.</p>
5	<p>1</p> <p>131 24 142f2</p>	<p>Строка имеет неправильный формат.</p>

Выводы.

Было реализовано случайное БДП с рандомизацией и написаны все необходимые функции для работы с ним. Написана функция проверки вхождения элемента в дерево, предусмотрена возможность повторного выполнения с другим элементом.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Файл node.h

```
#ifndef BDP_NODE_H
#define BDP_NODE_H
#include <iostream>
using namespace std;
#define COUNT 5

class Node {
    int key; // значение элемента узла
    int size; //размер дерева с корнем в данном узле
    int amount; //количество попыток ввести элемент со значением
данного узла
    Node *left; //левое поддерево
    Node *right; // правое поддерево
public:
    Node(int key, Node *left = nullptr, Node *right = nullptr);
    int find(Node *p, int k);
    int getSize(Node* p);
    void fixSize(Node* p);
    Node* rotateLeft(Node* p);
    Node* rotateRight(Node* p);
    Node* insert(Node* p, int k);
    Node* insertRoot(Node* p, int k);
    Node* getLeft();
    Node* getRight();
    void print2DUtil(Node *root, int space);
    void recTreePrint(Node* p);
};

#endif //BDP_NODE_H
```

Файл node.cpp

```
#include "node.h"

Node* Node::getLeft() { //возвращает левое поддерево
    return left;
}

Node* Node::getRight() { //возвращает правое поддерево
    return left;
}

Node::Node(int key, Node* left, Node* right):key(key),
//конструктор
left(left),
right(right){
    this->size = (left ? left->size : 0) + (right ? right->size :
0) + 1; //размер = размер левого поддерева + правого + 1
    this->amount = 1; //узел новый. количество = 1
```

```

    }

    int Node::find(Node* p, int k) // поиск ключа k в дереве p
    {
        if(!p)
            return false; // в пустом дереве можно не искать
        if( k == p->key )
            return p->amount; //если нашли, возвращаем поле amount -
количество вхождений
        if( k < p->key ) //если меньше, ищем в левом поддереве
            return find(p->left,k);
        else //если больше, ищем в правом поддереве
            return find(p->right,k);
    }

    int Node::getSize(Node* p) // возвращает размер дерева
    {
        return (p)? p->size : 0;
    }

    void Node::fixSize(Node* p) // установление корректного размера
дерева
    {
        p->size = getSize(p->left) + getSize(p->right) + 1; //размер =
размер левого поддерева + правого + 1
    }

    Node* Node::rotateLeft(Node* p) // левый поворот вокруг узла p
    {
        if (!p || !p->right) //если узла нет, или правого поддерева,
то поворот не происходит
            return p;

        Node* q = p->right;
        p->right = q->left;
        q->left = p;
        q->size = p->size;
        fixSize(p);
        return q;
    }

    Node* Node::rotateRight(Node* p) // правый поворот вокруг узла p
    {
        if (!p || !p->left) //если узла нет, или левого поддерева, то
поворот не происходит
            return p;

        Node* q = p->left;
        p->left = q->right;
        q->right = p;
        q->size = p->size;
        fixSize(p);
        return q;
    }

    Node* Node::insertRoot(Node* p, int k) // вставка нового узла с
ключом k в корень дерева p

```

```

{
    if(!p) {
        return new Node(k);
    }

    if( k < p->key )           // если значение k < значения узла, то
переходим в левое поддерево
    {
        p->left = insertRoot(p->left,k);
        p = rotateRight(p);    //осуществляем правый поворот
    }

    else if (k > p->key)         // если значение k > значения узла,
то переходим в правое поддерево
    {
        p->right = insertRoot(p->right,k);
        p = rotateLeft(p);     //осуществляем левый поворот
    }
    return p;
}

Node* Node::insert(Node* p, int k) // рандомизированная вставка
нового узла с ключом k в дерево p
{
    if(!p){
        return new Node(k); //если корня дерева не существует, то
создаем его в конструкторе со значением k
    }

    if(k == p->key){           //если узел с таким значением уже есть в
дереве, увеличиваем значение поля amount
        p->amount++;
        return p;
    }

    if(rand() % (p->size+1) == 0) { // вставка в корень происходит
с вероятностью 1/(n+1),
        // где n - размер дерева до вставки
        return insertRoot(p, k);
    }

    if(p->key > k) // иначе происходит обычная вставка в правое
или левое поддерево в зависимости от значения k
        p->left = insert(p->left, k);
    else
        p->right = insert(p->right,k);
    fixSize(p); //регулируется размер дерева
    return p;
}

void Node::recTreePrint(Node* p) { //печать дерева в обходе КЛП
    if (!p) {
        cout << "/ ";
        return;
    }
    cout << p->key << ' ';
    recTreePrint(p->left); //печать левого

```

```

        recTreePrint(p->right);    //печать правого
    }

```

Файл main.cpp

```

#include <cstdlib>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include "node.h"

Node* fillBdp(Node* p){    // заполнение БДП с консоли
    cin.get();
    string str;
    getline(cin, str); // считываем строку чисел через пробел
    istringstream ss(str);
    vector<int>arr;    //вектор введенных чисел
    int x;
    while(ss >> x){
        arr.push_back(x);
        if(ss.peek() == ' ')    //игнорируем пробелы
            ss.ignore();
    }
    if (!ss.eof())    //если были введены символы кроме цифер
    {
        std::cout << "Строка имеет неправильный формат.\n";
        exit(1);
    }
    for(int val : arr) {    //добавляем элементы из вектора в БДП
        p = p->insert(p, val);
        cout << "\n\033[31m Добавлен элемент " << val << "\033[0m\n";
        p->recTreePrint(p);
    }
    return p;
}

Node* fillMas(Node* p, string name, int count){    // заполнение БДП с файла
    ifstream in(name);    //открываем файл
    if (!in.is_open()) {
        cout << "Файл не может быть открыт!\n";
        return p;
    }
    int x[count];
    for (int i = 0; i < count - 1; i++)
        in >> x[i];
    for (int i = 0; i < count - 1; i++) {
        p = p->insert(p, x[i]);
        cout << "Добавлен элемент " << x[i] << '\n';
        p->recTreePrint(p);
    }
    in.close();    //под конец закроем файла
    return p;
}

Node* fillBdpFile(Node* p, string name){    //подсчет количества чисел в файле
    //Создаем файловый поток и связываем его с файлом

```

```

    ifstream in(name);
    if (!in.is_open()) {
        cout << "Файл не может быть открыт!\n";
        return p;
    }

    int count = 0; // количество чисел в файле
    int temp; //Временная переменная

    while (!in.eof()) // пробегаем пока не встретим конец файла eof
    {
        in >> temp; //в пустоту считываем из файла числа
        count++; // увеличиваем счетчик количества чисел
    }

    in.close();
    return fillMas(p, name, count);
}

void info(){
    cout << "\n\t\tВведите:\n"
           "a - Для поиска элемента в БДП\n"
           "b - Для выхода из программы\n";
}

void choose(){
    cout << "\n\t\tДля выбора введите:\n"
           "1 - Считывание элементов с консоли\n"
           "2 - Считывание из файла\n";
}

Node* foo(Node* p, int k){
    switch(k){
        case 1:{
            cout << "\t\tВведите числа для построения дерева. \n";
            p = fillBdp(p); //заполнение с консоли
            cout << "\nПолучившееся дерево:\n";
            p->print2DUtil(p, 0);
            break;
        }

        case 2:{
            string name;
            cout << "Введите путь до файла.\n";
            cin >> name;
            p = fillBdpFile(p, name); //заполнение с файла
            cout << "\nПолучившееся дерево:\n";
            p->print2DUtil(p, 0);
            break;
        }

        default:
            cout << "Попробуйте еще раз.\n";
            break;
    }
    return p;
}

void destroy(Node* p) //удаление дерева
{
    if(!p)

```



```

        return;
        destroy(p->getLeft());
        destroy(p->getRight());
        p = nullptr;
    }

int main() {
    srand(time(0));
    Node *p = nullptr;
    choose();
    char choice;
    int k;
    cin >> k;
    p = foo(p, k);
    while (choice != 'b') { //цикл, пока пользователь не выйдет из
программы
        info();
        cin >> choice;
        switch (choice) {
            case 'a':{
                cout << "Введите искомый элемент:\n";
                cin >> k;
                int count = p->find(p, k);
                if (count)
                    cout << k << "\033[34m содержится в дереве в количестве
" << count << "\033[0m\n";
                else
                    cout << k << "\033[34m не входит в дерево." <<
"\033[0m\n";
                p = p->insert(p, k);
                cout << "\033[31m Добавлен элемент " << k << "\033[0m\n";
                p->recTreePrint(p);
                break;
            }
            case 'b':
                cout << "Конец выполнения программы";
                break;
            default:
                cout << "Попробуйте ещё раз\n";
                break;
        }
    }
    cout << endl;
    destroy(p); //очистка дерева
    return 0;
}

```