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: Бинарные деревья поиска

Студент гр. 9382 _____ Кузьмин Д. И.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить принципы бинарных деревьев поиска. Освоить навыки разработки программ, реализующих бинарные деревья поиска.

Основные теоретические положения.

Двоичное дерево поиска — это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

•Оба поддерева — левое и правое — являются двоичными деревьями поиска.
•У всех узлов левого поддерева произвольного узла X значения ключей данных меньше либо равны, нежели значение ключа данных самого узла X.
•У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

Задание.

Вариант 12. БДП: Рандомизированная дерамида поиска (treap); действие: 1+2a

1) По заданной последовательности элементов *Elem* построить структуру данных определённого типа – БДП или хеш-таблицу;
2) 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.

Описание функций и структур данных.

1) В качестве БДП в работе используется рандомизированная дерамида поиска. Это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу. Каждый узел представляет собой пару (ключ, приоритет) и при этом для всех ключей выполнены свойства бинарного дерева(для каждого узла значение ключа левого потомка меньше, а правого потомка — больше(или равно)). И для всех приоритетов выполнены свойства кучи(для каждого узла приоритет обоих потомков меньше). Данная структура данных, в отличие от обычного бинарного дерева поиска определяется однозначно, то есть по известному набору элементов можно построить единственную дерамиду.

2) Для реализации элементов дерамиды был использован класс TreapElem. Среди его полей: int key – ключ, int prior - приоритет, TreapElem* right – указатель на правого потомка, TreapElem* left – указатель на левого потомка.

3) Для реализации дерамиды был использован класс Treap. Среди его полей: TreapElem* root – указатель на корневой узел типа TreapElem.

4) Среди функций дерамиды Treap:

void read(std::istream& is) – считывание и построение дерамиды, на вход принимает ссылку на поток ввода, ничего не возвращает.

void insert(TreapElem*& root, TreapElem a) – рекурсивно реализует алгоритм вставки(см. описание алгоритма) в дерамиду. Параметры: ссылка на указатель на узел, и сам элемент типа TreapElem. Ничего не возвращает.

void rotRight(TreapElem*& root) – осуществляет поворот данного узла (см. описание алгоритма) направо. Принимает ссылку на указатель узла, который нужно повернуть. Возвращает void.

void rotRight(TreapElem*& root) – аналогично, только осуществляет поворот налево.

int find(TreapElem* root, TreapElem e) – рекурсивно реализует алгоритм поиска заданного элемента e и возвращает количество его вхождений. На вход принимает указатель на текущий узел и сам элемент типа TreapElem.

void printLKP(TreapElem* root) – реализует вывод элементов дерамиды в порядке ЛКП. На вход принимает указатель на текущий узел. Ничего не возвращает.

Описание алгоритма

1) Алгоритм вставки. Просматривается текущий узел(начиная с корневого) и если он пустой, то на его месте создается новый, равный вставляемому элементу. Если же узел не пуст, то сравниваются значения его ключа со ключом вставляемого элемента. Далее аналогичные действия проделывается для левого потомка, если ключ узла меньше и для правого, если больше. Этим сравнением реализуются свойства бинарного дерева поиска.

2) Далее проверяется выполнены ли свойства кучи, то есть в некотором узле значения приоритета больше, чем в потомках. Если нет, то с помощью преобразования БДП, называемого «поворот» достигается соблюдения этих свойств. «Поворот» сохраняет инвариант БДП, но переставляет элементы таким

образом, что значения приоритетов может измениться. Если в текущем узле приоритет меньше чем у правого потомка, то осуществляется поворот налево. Аналогично если меньше, чем у левого потомка, то направо. В среднем сложность вставки - O(logn).

3) Алгоритм построения представляет из себя поочередные вставки элементов из данной последовательности. В среднем сложность - O(nlogn).

4) Алгоритм поиска. В его основе лежит инвариант БДП. Проверяется текущий узел на равенство с искомым элементом. Если равенство имеет место, то количество вхождений увеличивается на один и далее следует переход к правому потомку этого узла(в данной реализации элемент равный некоторому узлу располагается является правым потомком этого узла). Если узел и искомый элемент не равны, то сравниваются их ключи. В случае если ключ искомого элемента больше(или равен), то далее аналогичные действия осуществляется для правого потомка. Если меньше, то для левого. В среднем сложность - O(logn).

Исходный код см. в приложении А.

Тестирование.

Результаты тестирование представлены в табл. 1.

Таблица 1 — результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарий
1	Последовательность: (5, 3) (2, 10) (1, 2) Элемент: (5, 3)	построенная дерамида - (1,2)(2,10)(5,3) количество вхождений:1	Элемент (5, 3) встречается один раз.
2	Последовательность: (0, 10) (5, 7) (-5, 20) (3, 4) Элемент: (4, 3)	Построенная дерамида: (null)(-5,20)(null)(0,10)(3,4) (5,7)(null) количество вхождений: 0	Элемент (4, 3) не встречается в дерамиде.
3	Последовательность: (7, 2) (2, 2) (13, 4) Элемент: (3, 4)	Построенная дерамида: (2,2)(7,2)(13,4)(null) количество вхождений: 0	Элемент (3, 4) не встречается в дерамиде.
4	Последовательность: (20, 20) (-20, 20) (13, 4) (5, 7) Элемент:(13, 4)	Построенная дерамида: (null)(-20,20)(null)(5,7) (13,4)(20,20)(null) Количество вхождений: 1	Элемент (13, 4) встречается в дерамиде 1 раз.
5	Последовательность: (0,0)(0,1)(null) Элемент: (10, 10)	Построенная дерамида: (0,0)(0,1)(null) Количество вхождений: 0	Элемент (10, 10) не встречается в дерамиде.
6	Последовательность: (1,1) (1, 1) (1, 1) Элемент: (1, 1)	Построенная дерамида: (null)(1,1)(null)(1,1)(1,1) Количество вхождений: 3	Элемент (1, 1) встречается три раза.
7	Последовательность: (0, 25) (25, 0) (1, 11) (11, 11) Элемент: (1, 11)	Построенная дерамида: (null)(0,25)(null)(1,11)(null) (11,11)(25,0) Количество вхождений: 1	Элемент (1, 11) встречается 1 раз.

Выводы.

Был изучен принцип бинарных деревьев поиска, в частности рандомизированной дерамиды поиска. Получены навыки разработки программ, релизующих некоторые операции по работе с этими структурами данных.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <cstring

//элемент дерамиды
class TreapElem {
public:
	int key;
	int prior;
	TreapElem* right = nullptr;
	TreapElem* left = nullptr;

	TreapElem(int key, char prior): key(key), prior(prior) {}
	TreapElem(int key):key(key){
		prior = rand() % 100;
	}
	TreapElem() {}
	friend std::istream& operator>>(std::istream& in, TreapElem& e) {
//оператор ввода из потока. каждый элемент представляется

				//в виде(x, y), где x - ключ, y - приоритет
		while (in.get() != '(' && !in.eof()) {}
		in >> e.key;
		while (in.get() != ',' && !in.eof()) {}
		in >> e.prior;
		while (in.get() != ')' && !in.eof()) {}
		return in;
	}

	friend std::ostream& operator<<(std::ostream& out, TreapElem* e) {
//оператор вывода в поток

				//пустые элементы выводятся как (null)
		if(e != nullptr)
		out << '(' << e->key << "," << e->prior << ")";
		else out << "(null)";
		return out;
	}
};

class Treap {

public:

	TreapElem* root;

	Treap() {
		root = nullptr;
	}
	//вставка элемента
	void insert(TreapElem*& root, TreapElem a)
	{
```

```
            //создание нового элемента на месте некоторого пустого, выход
из функции
            if (root == nullptr)
            {
                  std::cout << "Встречен пустой узел. Создание нового
элемента " << &a << " на его месте.\n";
                  root = new TreapElem(a);
                  return;
            }
            //сравнения с ключом текущего узла
            if (a.key < root->key) {
                  std::cout << "Значение ключа вставляемого элемента -" <<
&a << " меньше значения ключа текущего элемента - " << root << "\n";
                  std::cout << "Переход к левому потомку\n";
                  insert(root->left, a);
            }

            else if (a.key >= root->key) {
                  std::cout << "Значение ключа вставляемого элемента -" <<
&a << " не меньше значения ключа текущего элемента - " << root<< "\n";
                  std::cout << "Переход к правому потомку\n";
                  insert(root->right, a);
            }

            //повороты (если требуется)
            if (root->left && root->left->prior > root->prior) {
                  std::cout << "Значение приоритета элемента " << root-
>left << " больше чем значение приоритета элемента " << root << "\n";
                  std::cout << "Осуществляется поворот направо\n";
                  rotRight(root);
            }

            if (root->right && root->right->prior > root->prior) {
                  std::cout << "Значение приоритета элемента " << root-
>right << " больше чем значение приоритета элемента " << root << "\n";
                  std::cout << "Осуществляется поворот налево\n";
                  rotLeft(root);
            }
      }

      //поворот налево
      void rotLeft(TreapElem*& el) {

            std::cout<<"Поворот налево узла " << el <<":\n";
            TreapElem* newEl = el->right;
            std::cout << "Корень нового узла равен " << el->right;
            el->right = newEl->left;
            std::cout << "\nЛевый потомок нового узла равен " << el <<
"\n";
            newEl->left = el;
            el = newEl;
      }
      //поворот направо
      void rotRight(TreapElem*& el) {
            std::cout << "Поворот направово узла " << el << ":\n";
            TreapElem* newEl = el->left;
            std::cout << "Корень нового узла равен " << newEl;
            el->left = newEl->right;
```

```
            newEl->right = el;
            el = newEl;
            std::cout << "\nПравый потомок нового узла равен " << newEl-
>right <<"\n";
      }
      //считывание и построение дерамиды
      void read(std::istream& is) {
            std::vector<TreapElem> treapVector;
            while (1) {
                  TreapElem a;
                  is >> a;
                  treapVector.push_back(a);
                  char tmp = is.get();
                  if (tmp == '\n' || tmp == EOF) break;
            }
            std::cout << "\nПостроение дерамиды:\n";
            for (auto it : treapVector) {
                  std::cout << "Вставка элемента " << &it <<":\n";
                  insert(root, it);
                  std::cout << "\n";
            }
      }

      //проверка является ли элемент листом
      bool isLeaf(TreapElem* e) {
            return (!e->left && !e->right);
      }
      //поиск элемента в дерамиде, функция возвращает количество
вхождений
      int find(TreapElem* root, TreapElem e){

            if (root) {
                  std::cout << "Просмотр текущего узла " << root << "\n";
                  if (root->key == e.key && root->prior == e.prior  &&
isLeaf(root)){
                        std::cout << "Узел является листом и совпадает с
искомым элементом. Количество вхождением увеличено на 1.\n";
                        return 1;
                  }

                  else if (root->key == e.key && root->prior == e.prior
&& !isLeaf(root)) {

                        std::cout << "Узел совпадает с искомым элементом и
не является листом. Количество вхождений увеличено на 1. Переход к
правому потомку.\n";
                        return 1 + find(root->right, e);
                  }

                  else {
                        std::cout << "Текущий узел не совпадает с
искомым.\n";
                        if (e.key >= root->key) {
                              std::cout << "Ключ искомого элемента больше
или равен ключу текущего узла " << root;
                              std::cout << "\nПереход к правому потомку.\n";
                              return find(root->right, e);
                        }
```

```
                    else {
                        std::cout << "Ключ искомого элемента меньше
ключа текущего узла " << root;
                        std::cout << "\nПереход к левому потомку.\n";
                        return find(root->left, e);
                    }
                }
            }
            else {
                std::cout << "Встречен пустой узел.\n";
                return 0;
            }
        }
        //вывод узлов в формате ЛКП
        void printLKP(TreapElem* root) {
            if (root) {
                if (isLeaf(root)) std::cout << root;
                else {
                    printLKP(root->left);
                    std::cout << root;
                    printLKP(root->right);
                }
            }
            else std::cout << root;
        }
};
//интерфейс для вызова функции поиска
void findInterface(Treap& treap) {

            std::cout << "\nВведите элемент в формате (ключ, приоритет),
который нужно найти либо пустую строку для выхода из программы.\n";
            char* n = new char[20];
            fgets(n, 20, stdin);
            if (strcmp(n, "\n") == 0) return;
            else {
                std::stringstream ss(n);
                TreapElem b;
                ss >> b;

                std::cout << "\nПоиск элемента " << &b << "\n";
                int count = treap.find(treap.root, b);
                std::cout << "Количество вхождений элемента " << &b << "
- " << count << "\n\n";

                std::cout << "Вставка элемента " << &b << ":\n";

                treap.insert(treap.root, b);

                std::cout << "Элемент " << &b << " добавлен в
дерамиду\n\nПолученная дерамида(список узлов в порядке лкп):\n";

                treap.printLKP(treap.root);
                std::cout << "\n";

                findInterface(treap);
            }

}
```

```
int main() {

	Treap treap;
	setlocale(LC_ALL, "Russian");
	char* a = new char[20];
	while (1) {
		std::cout << "Введите 1 для ввода с консоли и 2 для ввода с
файла:\n";
		fgets(a, 20, stdin);
		if (a[1] != '\n' || (a[0] != '1' && a[0] != '2')) continue;
		else {
			switch (a[0]) {
			case '1':
				std::cout << "Введите последовательность элементов
дерамиды в виде (ключ, приоритет) через пробел\n";
				treap.read(std::cin);
				break;
			case '2':
				std::ifstream ifs("test.txt");
				char d;
				std::cout << "Содержимое файла: ";
				while (ifs.get(d)) std::cout << d;
				std::cout << "\n";
				ifs.close();
				ifs.open("test.txt");
				treap.read(ifs);
				ifs.close();
				break;
			}
				break;
		}
	}

	std::cout << "Построенная дерамида(узлы в порядке лкп):\n";

	treap.printLKP(treap.root);

	std::cout << "\n";

	findInterface(treap);

	return 0;

}
```