МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: Случайное БДП с рандомизацией

Студентка гр. 9303 _____ Булыно Д. А.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Формирование практических навыков программирования такой структуры данных, как случайное БДП с рандомизацией, на языке программирования C++ путём решения поставленной задачи.

Основные теоретические положения.

Случайные бинарные деревья поиска с рандомизацией

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).

Вставка элемента со значением key в дерево tree. 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.

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.

Задание.

Вариант 11

Случайное БДП с рандомизацией.

1) По заданной последовательности элементов *Elem* построить структуру данных определённого типа – БДП или хеш-таблицу;
2) в) Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран в наглядном виде.

Выполнение работы.

Для выполнения программы была написана структура `Node` для представления узлов дерева и несколько функций:

1. `int getsize(Node* btree)` – функция для хранения размера дерева;
2. `void fixsize(Node* btree)` – функция для корректировки размера дерева;
3. `Node* rotateright(Node* btree)` – функция, осуществляющая правый поворот вокруг узла `btree`;
4. `Node* rotateleft(Node* btree)` – функция, осуществляющая левый поворот вокруг узла `btree`;
5. `Node* insertroot(Node* btree, int k)` – функция вставки нового узла с ключом в корень;
6. `Node* insert(Node* btree, int k)` – функция рандомизированной вставки нового узла с ключом в дерево;
7. `Node* Create(int key, Node* left, Node* right)` – функция создания дерева;
8. `void Print(Node* btree, int level)` – функция вывода дерева;
9. `void LTR(Node* btree, ofstream & out)` – функция записи элементов в файл в порядке возрастания.

Код программы см. в приложении А.

Результаты работы программы см. в файле «result.txt».

Выводы.

В ходе выполнения лабораторной работы было изучено случайное БДП с рандомизацией. В результате выполнения лабораторной работы была написана программа, которая строит данное дерево, записывает в файл элементы этого дерева в порядке возрастания, которая выводит построенное дерево на экран.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <fstream>
using namespace std;

struct Node {
	int key;
	int size;
	Node* left;
	Node* right;
	Node(int k) {
		key = k;
		left = right = nullptr;
		size = 1;
	}
};

int getsize(Node* btree){
	if (!btree) return 0;
	return btree->size;
}

void fixsize(Node* btree){
	btree->size = getsize(btree->left) + getsize(btree->right) + 1;
}

Node* rotateright(Node* btree){
	Node* newbtree = btree->left;
	if (!newbtree) return btree;
	btree->left = newbtree->right;
	newbtree->right = btree;
	newbtree->size = btree->size;
	fixsize(btree);
	return newbtree;
}

Node* rotateleft(Node* btree){
	Node* newbtree = btree->right;
	if (!newbtree) return btree;
	btree->right = newbtree->left;
	newbtree->left = btree;
	newbtree->size = btree->size;
	fixsize(btree);
	return newbtree;
}

Node* insertroot(Node* btree, int k){
	if (!btree) return new Node(k);
```

```
      if (k < btree->key){
            btree->left = insertroot(btree->left, k);
            return rotateright(btree);
      }
      else{
            btree->right = insertroot(btree->right, k);
            return rotateleft(btree);
      }
}

Node* insert(Node* btree, int k){
      if (!btree) return new Node(k);
      if (rand() % (btree->size + 1) == 0)
      return insertroot(btree, k);

      if (btree->key > k)
            btree->left = insert(btree->left, k);
      else
            btree->right = insert(btree->right, k);

      fixsize(btree);
      return btree;
}

Node* Create(int key, Node* left, Node* right){

      Node* res = new Node(key);
      res->left = left;
      res->right = right;

      return res;
}


void Print(Node* btree, int level){

      if (btree->right != nullptr){
            Print(btree->right, level + 1);
      }
      for (int i = 0; i < level; i++) {
            cout << "    ";
      }

      cout << btree->key << "\n";

      if (btree->left != nullptr) {
            Print(btree->left, level + 1);
      }
}

void LTR(Node* btree, ofstream & out){
```

```
	if(btree == nullptr) return;
	LTR(btree->left, out);
	out << btree->key << " ";
	LTR(btree->right, out);
}

int main() {
	char c = 'y';
	int input = 0;

	int size;
	cout << "Введите количество элементов: ";
	cin >> size;
	while (size<=0){
		cout << "Неправильно указано количество элементов.\nВведите
количество элементов: ";
		cin >> size;
	}
	Node* tree = nullptr;
	srand(time(0));

	for (int i = 0; i < size; i++) {
		int in;
		cin >> in;
		tree = insert(tree, in);
	}

	cout << endl << "Случайное БДП с рандомизацией:" << endl;
	Print(tree, 0);
	ofstream file;
	file.open("result.txt");
	LTR(tree, file);
	file.close();
	return 0;
}
```