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

ОТЧЁТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья поиска

Студентка гр. 7383 _____ Иолшина В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2018

Содержание

Цель работы.

Познакомиться с рандомизированными деревьями поиска и научиться реализовывать их на языке программирования C++. По заданному файлу F (типа file of Elem), все элементы которого различны, построить дерево поиска. Для построенного БДП проверить, входит ли в него элемент e типа Elem, и если не входит, то добавить элемент e в дерево поиска.

Реализация задачи.

Бинарное дерево поиска в данной работе реализовано с помощью массива структур. Используются следующие функции:

- base Root(BST* tree);
- BST* Left(BST* tree);
- BST* Right(BST* tree);
- BST* createBST(BST* tree, int el);
- void printBST(BST* tree, int n);
- bool find_add (BST* tree, base &x);
- BST* destroy(BST* tree);

В функции main выводится приглашение выбрать способ ввода входных данных или закончить работу. В случае выбора ввода данных из файла, программа считывает строку из файла text.txt и записывает её в поток ввода, после этого закрывает файл. В случае выбора ввода информации с консоли, функция main считывает строку и записывает её в этот же поток ввода. Если файл пустой, то main выводит ошибку и начинает выполнение программы заново. Если все было введено верно, происходит вызов функции createBST. Функция createBST строит дерево с помощью функции ConsBST. Далее происходит вызов функции find_add, которая либо находит искомый элемент и выводится сообщение, что элемент в дереве уже есть, либо, не найдя элемент, добавляет этот элемент в дерево. Далее при помощи функции printBST полученное дерево выводится на консоль.

Тестирование.

Процесс тестирования.

Программа собрана в операционной системе Ubuntu 16.04.2 LTS", с использованием компилятора g++ (Ubuntu 5.4.0-6ubuntu1~16.04.5). В других ОС и компиляторах тестирование не проводилось.

Результаты тестирования

Тестовые случаи представлены в Приложении А.

Во время тестирования ошибок обнаружено не было, что свидетельствует о том, что поставленная программа была выполнена.

Выводы.

В ходе выполнения лабораторной работы была изучена такая нелинейная структура данных, как бинарное дерево поиска, способы её представления и реализации, получены навыки решения задач и обработки бинарных деревьев поиска на языке C++. Была реализована программа, создающее бинарное дерево поиска определенного типа, проверяющая дерево на наличие элемента и вставляющая его, если элемента обнаружено не было.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ

bt.h :

```
#pragma once
typedef int base;
#define N 100

struct BST
{
  base key;
  BST* left;
  BST* right;
  BST(int k)
  {
    key = k;
    left = nullptr;
    right = nullptr;
  }
};
base Root(BST* tree);
BST* Left(BST* tree);
BST* Right(BST* tree);
BST* createBST(BST* tree, int el);
void printBST(BST* tree, int n);
bool find_add (BST* tree, base &x);
BST* destroy(BST* tree);
```

bt.cpp :

```
#include "bst.h"
#include <iostream>
using namespace std;

BST* destroy(BST* tree)
{
  if (left)
    delete tree->left;
  if (right)
    delete tree->right;
  delete tree;
```

```
  return tree = NULL;
            }

base Root(BST* tree)
{
  if (tree == NULL)
    exit(1);
  else
    return tree->key;
}

BST* Left(BST* tree)
{
  if (tree == NULL)
    exit(1);
  else
    return tree->left;
}

BST* Right(BST* tree)
{
  if (tree == NULL)
    exit(1);
  else
    return tree->right;
}

BST* createBST(BST* tree, int el)
{
  if (!tree)
    return new BST(el);
            if(tree->key > el)
    tree->left = createBST(Left(tree), el);
  else
    tree->right = createBST(Right(tree), el);
            return tree;
}

void printBST(BST* tree, int l)
```

```
{
  if(tree==NULL)
  {
     for(int i=0; i<l; i++)
        cout << "\t";
     cout << '#' << endl;
     return;
  }
  printBST(Right(tree), l+1);
  for(int i=0; i<l; i++)
     cout << "\t";
  cout << Root(tree) << endl;
  printBST(Left(tree),l+1);
}

bool find_add (BST* tree, base &k)
{
 if (Root(tree) > k)
 {
  if (Left(tree) == NULL)
  {
    tree->left = new BST(k);
    return false;
  }
    return find_add(Left(tree), k);
 }
 if (Root(tree) < k)
 {
  if (Right(tree) == NULL)
  {
    tree->right = new BST(k);
    return false;
  }
  return find_add(Right(tree), k);
 }
 if (Root(tree) == k)
  return true;
}
```

Main.cpp :

```
#include <iostream>
#include <fstream>
#include "bst.h"
#include <sstream>
#include <ctype.h>
#include <cstring>

using namespace std;

int main()
{
 bool b = 0;
 int count=0;
 int k;
 base el;
 int run = 0;
 base arr[N];
  while(run!=3)
 {
  cout << "Введите 1, если хотите ввести выражение с клавиатуры.\n"
       "Введите 2, если хотите использовать выражение из файла test.txt.\n"
       "Введите 3, если хотите закончить работу." << endl;
  cin >> run;
  BST* tree = NULL;
  count=0;
  switch(run)
  {
   case 1:
    {
     cout << "Введите последовательность различных элементов: \n";
     cin.get();
     char c=getchar();
     while(c!='\n')
     {
      cin >> el;
      arr[count] = el;
      c=getchar();
      count++;
     }
```

```
      cout << "Введите искомый элемент: \n";
      cin >> k;
      b=1;
      break;
    }
  case 2:
    {
      ifstream outfile;
      outfile.open("test.txt");
      if (!outfile)
      {
        cout << "Входной файл не открыт!\n";
        b = 0;
        break;
      }
      while(outfile >> arr[count])
     {
       count++;
     }
      outfile.close();
      b=1;
      cout << "Введите искомый элемент: \n";
      cin >> k;
      break;
    }
  case 3:
    {
      b=0;
      break;
    }
  default:
    {
      cout << "Введите верное число\n";
      b=0;
      break;
    }
}
if(b)
{
```

```
  for (int i=0; i<count; i++)
  tree = createBST(tree, arr[i]);
  if(find_add(tree, k))
    cout << "Элемент в дереве уже есть\n";
  else
    cout << "Элемент вставлен в дерево\n";
  cout << "Дерево: \n";
  printBST(tree, 1);
  tree = destroy(tree);
  cout << endl;
}   }return 0;}
```

ПРИЛОЖЕНИЕ Б.
ТЕСТОВЫЕ СЛУЧАИ

Ввод	Вывод	Верно
123 788 1324 12 48 91 91	Результат работы программы 1324 788 123 48 12 9 Элемент в дереве уже есть	Да
6786 427 472 42 92 47	Результат работы программы 6786 472 427 92 47 42 Элемент вставлен в дерево	Да
7 91 3 4 1 2 4 8 12 9	Реузльтат работы программы 91 12 9 8 7 4 3 2	Да

	1 Элемент вставлен в дерево	