**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Деревья»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. 7381 | |  | Вологдин М.Д. |
| Преподаватель |  |  | Фирсов М.А. |

Санкт-Петербург

2018

**Задание:**

2д-в) Для заданного бинарного дерева *b* типа *BT* с произвольным типом элементов определить, есть ли в дереве *b* хотя бы два одинаковых элемента.

**Пояснение задания:**

На вход программе подаётся скобочная запись дерева. Требуется вывести True или False, в зависимости от того, есть ли в дереве два одинаковых элемента.

Тип узлов дерева в программе – char, максимальная глубина дерева – 5.

**Описание алгоритма:**

1. Создаём массив, в котором будут храниться элементы дерева.
2. Проходим по всей скобочной записи дерева, игнорируя пробелы и скобки, записываем каждый элемент дерева в массив следующим образом:
   1. Корень кладется в n-ную (n=0 для корня главного дерева) ячейку
   2. Корень левого поддерева кладется в 2\*n+1 ячейку, правого в 2\*n+2 ячейку.
   3. Действия a и b повторяются для каждого поддерева.
3. Ищем повторы в дереве, для этого сравниваем каждый узел дерева, начиная с корня, со всеми остальными узлами. При первой встрече одинаковых элементов, выходим из программы.

**Описание функций**:

* Template <typename T>int enterBT(T \* array, int cur)  
  – Основная рекурсивная функция для считывания скобочной записи дерева. Функция принимает массив любого типа и его текущую длину, возвращает новую длину массива.
* int Left (int x)

– Вычисление корня левого поддерева. Принимает номер элемента в массиве и возвращает номер корня левого поддерева.

* int Right (int x)

– Аналогично Left для правого поддерева.

* template <typename T>  
  bool isNull(T x)  
  – Проверка узла на пустоту. Принимает значение узла и возвращает false, если в корне есть значение и true в ином случае.
* Template <typename T>  
  bool sravn(T \* arr, int d, T x, int cur\_pos)  
  – Сравнение корня дерева со всеми его узлами. Возвращает true, если нашлось повторение и false в ином случае.
* Template <typename T>  
  bool isduplicate(T \* arr, int d)  
  – Поиск повторов в дереве. Принимает дерево и номер корня. Возвращает true, если нашелся хотя бы 1 повтор, false в ином случае.

**Тестирование** **программы**:

Для проверки программы были созданы тесты с корректными и некорректными данными.

Рассмотрим первый тест подробнее. На вход программе подаётся строка «(a(b(d^(h^^))(e^^))(c(f(i^^)(j^^))(g^(k(l^^)^)))))». Программа кладет заданное дерево в массив. Затем идёт по каждому узлу дерева, и проверяет, есть ли в дереве еще 1 с таким же значением. Так как таких узлов не оказалось, то вывелся «False».

В табл. 1 приведены результаты тестирования.

Таблица 1 – Результаты тестирования

|  |  |
| --- | --- |
| (a(b(d^(h^^))(e^^))(c(f(i^^)(j^^))(g^(k(l^^)^)))) | 0 a  1 b  2 c  3 d  4 e  5 f  6 g  7 ^  8 h  9 ^  10 ^  11 i  12 j  13 ^  14 k  15 0  16 0  17 ^  18 ^  19 0  20 0  21 0  22 0  23 ^  24 ^  25 ^  26 ^  27 0  28 0  29 l  30 ^  31 0  32 0  33 0  34 0  35 0  36 0  37 0  38 0  39 0  40 0  41 0  42 0  43 0  44 0  45 0  46 0  47 0  48 0  49 0  50 0  51 0  52 0  53 0  54 0  55 0  56 0  57 0  58 0  59 ^  60 ^  61 0  62 0  False |
| (a(b(d^(h^^))(e^^))(c(f(b^^)(j^^))(g^(k(l^^)^)))) | Duplicate is b  True |
| (a^^) | False |
| (a(b)) | Error in the tree view in 1 |

**Выводы.**

В результате работы была освоена тема «Деревья» на примере представления дерева в виде массива и поиске в нём двух одинаковых элементов.

**Приложение А**

**Исходный код**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <cctype>

//max depth is 5

#define N 63

typedef char type;

int Left (int x)

{

return x\*2+1;

}

int Right (int x)

{

return x\*2+2;

}

template <typename T>

bool isNull(T x)

{

if (x == '0' || x == '^')

return true;

else

return false;

}

template <typename T>

bool enterBT(T \* array, int cur)

{

char bkt;

std::cin >> array[cur]; // root reading

std::cin >> bkt; // reading '('

if (bkt != '(' && bkt != '^')

{

std::cout << "Error in the tree view in " << cur << std::endl;

return false;

}

if (bkt =='(')

{

if (!enterBT(array,Left(cur))) // reading the left subtree

{

return false;

}

std::cin >> bkt; // reading ')'

if (bkt != ')')

{

std::cout << "Error in the tree view in " << cur << std::endl;

return false;

}

}

if (bkt == '^')

array[Left(cur)] = '^';

std::cin >> bkt; // reading '('

if (bkt != '(' && bkt != '^')

{

std::cout << "Error in the tree view in " << cur << std::endl;

return false;

}

if (bkt =='(' )

{

if (!enterBT(array,Right(cur))) // reading the right subtree

{

return false;

}

std::cin >> bkt; // reading ')'

if (bkt != ')')

{

std::cout << "Error in the tree view in " << cur << std::endl;

return false;

}

}

if (bkt == '^')

array[Right(cur)] = '^';

return true;

}

template <typename T>

bool sravn(T \* arr, int d, T x, int cur\_pos)

{

if (!isNull(arr[d]))

{

if (x == arr[d] && d!=cur\_pos) //comparison of the root in the tree

{

std::cout << "Duplicate is " << x << std::endl;

return true;

}

if(sravn(arr,Left(d),x,cur\_pos)) //comparing the root of the left subtree in the tree.

return true;

if (sravn(arr,Right(d),x,cur\_pos)) //comparing the root of the right subtree in the tree.

return true;

}

return false;

}

template <typename T>

bool isduplicate(T \* arr, int d)

{

if (!isNull(arr[d]))

{

if (sravn(arr, 0,arr[d],d)) //comparison of all elements of the tree with the root

return true;

if(isduplicate(arr,Left(d))) //search in the left subtree

return true;

if(isduplicate(arr,Right(d))) //search in the left subtree

return true;

}

return false;

}

int main()

{

char bkt;

std::cin >> bkt; // reading '('

if (bkt != '(')

{

std::cout << "Error in the tree view in 1" << std::endl;

return (0);

}

type mas[N];

for (int i =0;i<N;i++)

mas[i]='0';

if(!enterBT(mas, 0)) // reading tree

{

return (0);

}

std::cin >> bkt; // reading ')'

if (bkt != ')')

{

std::cout << "Error in the tree view in the end" << std::endl;

return (0);

}

for (int i=0;i<N;i++)

std::cout << i << ' ' << mas[i] <<std::endl;

if (isduplicate(mas,0)) // search for repetitions

std::cout << "True" << std::endl;

else

std::cout << "False" << std::endl;

return (0);

}