Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №7**

По дисциплине «Математические основы интеллектуальных систем»

Тема: «Проверка бинарных деревьев на изоморфность»

**Выполнил:**

Студент 2 курса

Группы ИИ-21

Литвинюк Т. В.

**Проверил:**

Козинский А. А.

Брест 2023

**Цель:** научиться делать проверку деревьев на изоморфность, с помощью обходов.

**Ход работы:**

**Вариант 7**

1. Написать программу проверки бинарных деревьев на изоморфность, используя обходы бинарных деревьев. Варианты заданий указаны в таблице 1. Дерево задано списком ребер. Корнем дерева является вершина ***a.***
2. Добавлением и удалением вершин в деревьях привести деревья к изоморфному виду.
3. Определить высоту каждого дерева и указать является ли оно сбалансированным.
4. Изобразить деревья.

|  |  |
| --- | --- |
| (a,b),(a,c),(b,d).(b,e)  (c,f),(c,g) | a,b),(a,c),(b,d).  (c,f),(c,e) |

#include "..\graphs.h"

#include <iostream>

using namespace std;

int find(int el, int\* arr, int len){

for (int i = 0; i < len; i++){

if (arr[i] == el)

return i;

}

return -1;

}

void dfs(int v, vector<vector<int>>& adj\_matrix, vector<bool>& visited, List& res) {

visited[v] = true; // помечаем вершину как посещенную

res.append(v);

for (int u = 0; u < adj\_matrix.size(); u++) {

if (adj\_matrix[v][u] == 1 && !visited[u]) { // если есть ребро и вершина не посещена

dfs(u, adj\_matrix, visited, res); // рекурсивно вызываем dfs для смежной вершины

res.append(v);

}

}

}

List pref\_detour(List dfs\_res){

List pref\_res;

for (int i = 0; i < dfs\_res.length; i++){

if (pref\_res.find(dfs\_res.list[i]) != -1)

continue;

pref\_res.append(dfs\_res.list[i]);

}

return pref\_res;

}

List inf\_detour(List dfs\_res){

List inf\_res, meets; // в meet будут записаны вершины, которые встречались раньше

int temp;

for (int i = 0; i < dfs\_res.length; i++){

temp = dfs\_res.list[i];

if (meets.find(temp) != -1 && inf\_res.find(temp) == -1)

inf\_res.append(temp);

meets.append(dfs\_res.list[i]);

dfs\_res.list[i] = -1;

if (dfs\_res.find(temp) == -1 && inf\_res.find(temp) == -1)

inf\_res.append(temp);

}

return inf\_res;

}

List postf\_detour(List dfs\_res){

List postf\_res; // в meet будут записаны вершины, которые встречались раньше

int temp;

for (int i = 0; i < dfs\_res.length; i++){

temp = dfs\_res.list[i];

dfs\_res.list[i] = -1;

if (dfs\_res.find(temp) == -1)

postf\_res.append(temp);

}

return postf\_res;

}

int main(){

char path[] = "E:\\Studing\\MOIS\\Lab. 7\\cons", path2[] = "E:\\Studing\\MOIS\\Lab. 7\\cons2";

int tops = getTopsCount(path), edges = getEdgesCount(path), tops2 = getTopsCount(path2), edges2 = getEdgesCount(path2);

int\*\* matr = adjacencyMatrixFromConnections(path), \*\*matr2 = adjacencyMatrixFromConnections(path2);

vector<vector<int>> adj(tops, vector<int> (tops)); vector<vector<int>> adj2(tops2, vector<int> (tops2));

vector<bool> vis(tops, false); vector<bool> vis2(tops2, false);

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

for (int j = 0; j < tops; j++)

adj[i][j] = matr[i][j];

for (int i = 0; i < tops2; i++)

for (int j = 0; j < tops2; j++)

adj2[i][j] = matr2[i][j];

List dfs\_res, dfs\_res2;

dfs(0, adj, vis, dfs\_res); dfs(0, adj2, vis2, dfs\_res2);

// префиксный(прямой) обход

List pref\_res = pref\_detour(dfs\_res), pref\_res2 = pref\_detour(dfs\_res2);

cout << "1-ое дерево, прямой обход: ";

for (int i = 0; i < pref\_res.length; i++)

cout << pref\_res.list[i] + 1 << " ";

cout << endl << "2-ое дерево, прямой обход: ";

for (int i = 0; i < pref\_res2.length; i++)

cout << pref\_res2.list[i] + 1 << " ";

// инфиксный(симметричный) обход

List inf\_res = inf\_detour(dfs\_res), inf\_res2 = inf\_detour(dfs\_res2);

cout << endl << endl << "1-ое дерево, симметричный обход: ";

for (int i = 0; i < inf\_res.length; i++)

cout << inf\_res.list[i] + 1 << " ";

cout << endl << "2-ое дерево, симметричный обход: ";

for (int i = 0; i < inf\_res2.length; i++)

cout << inf\_res2.list[i] + 1 << " ";

// постфиксный(обратный) обход

List postf\_res = postf\_detour(dfs\_res), postf\_res2 = postf\_detour(dfs\_res2);

cout << endl << endl << "1-ое дерево, обратный обход: ";

for (int i = 0; i < inf\_res.length; i++)

cout << postf\_res.list[i] + 1 << " ";

cout << endl << "2-ое дерево, обратный обход: ";

for (int i = 0; i < inf\_res2.length; i++)

cout << postf\_res2.list[i] + 1 << " ";

cout << endl << endl << "Деревья";

if(pref\_res == pref\_res2 and inf\_res == inf\_res2 and postf\_res == postf\_res2)

cout << " ";

else cout << " не ";

cout << "изоморфны.";

}

**Вывод:** в ходе лабораторной работы я научился находить кратчайшие пути в графе.