`

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

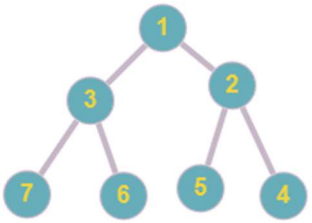
1. Написать программу проверки бинарных деревьев на изоморфность, используя обходы бинарных деревьев. Варианты заданий указаны в таблице 1. Дерево задано списком ребер. Корнем дерева является вершина ***a.***
2. Добавлением и удалением вершин в деревьях привести деревья к изоморфному виду.
3. Определить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

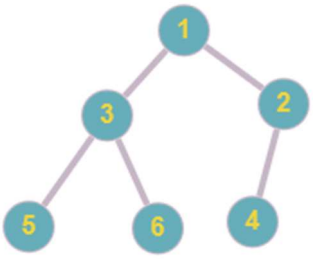
(a,b),(a,c),(b,d).(b,e) (c,f),(c,g)	a,b),(a,c),(b,d). (c,f),(c,e)

```
#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.";
}
```

```
1-ое дерево, прямой обход: 1 2 4 5 3 6 7
2-ое дерево, прямой обход: 1 2 4 3 5 6

1-ое дерево, симметричный обход: 4 2 5 1 6 3 7
2-ое дерево, симметричный обход: 4 2 1 5 3 6

1-ое дерево, обратный обход: 4 5 2 6 7 3 1
2-ое дерево, обратный обход: 4 2 5 6 3 1

Деревья не изоморфны.
```





Высота обоих деревьев: 3.

Оба дерева являются сбалансированными.

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