Министерство образования Республики Беларусь

Учреждения образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №7

По дисциплине МОИС за 2 семестр

«Проверка бинарных деревьев на изоморфность»

Выполнил

Студент группы ИИ-23

Романюк А.П.

Проверил

Козинский А.А.

Брест 2024

Цель: изучить обходы бинарных деревьев, алгоритмы проверки на изморфность.

Задание:

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.

Вариант:

|  |  |  |
| --- | --- | --- |
| № | Дерево | Дерево |
| 4. | (a,b),(a,c),(b,d).(b,e)  (c,f),(c,g).(d,h),(f,k),(g,l) | (a,b),(a,c),(b,d).(c,f),  (c,g).(d,h),(f,k),(g,l) |

Выполнение:

task.cpp:

#include <iostream>

#include "Graph.h"

#include <fstream>

#include <string>

#include <unordered\_map>

using namespace std;

string convertStringToNumbers(const string& input, unordered\_map<char, int>& charMap) {

string result;

for (char ch : input) {

if (isalpha(ch)) {

char convertedChar = toupper(ch);

if (charMap.find(convertedChar) == charMap.end()) {

// Если буква встречается первый раз, присвоить ей новый номер

int number = charMap.size() ;

charMap[convertedChar] = number;

result += to\_string(number);

}

else {

// Если буква уже встречалась, присвоить ей тот же номер

result += to\_string(charMap[convertedChar]);

}

}

else {

result += ch;

}

}

return result;

}

int main() {

ifstream inputFile1("input1.txt");

if (!inputFile1.is\_open()) {

cerr << "Error opening input file." << endl;

return 1;

}

ifstream inputFile2("input2.txt");

if (!inputFile2.is\_open()) {

cerr << "Error opening input file." << endl;

return 1;

}

ofstream outputFile1("input1conv.txt");

if (!outputFile1.is\_open()) {

cerr << "Error opening output file." << endl;

inputFile1.close();

return 1;

}

ofstream outputFile2("input2conv.txt");

if (!outputFile2.is\_open()) {

cerr << "Error opening output file." << endl;

inputFile2.close();

return 1;

}

unordered\_map<char, int> charMap1;

string line;

while (getline(inputFile1, line)) {

string convertedLine = convertStringToNumbers(line, charMap1);

outputFile1 << convertedLine << endl;

}

unordered\_map<char, int> charMap2;

while (getline(inputFile2, line)) {

string convertedLine = convertStringToNumbers(line, charMap2);

outputFile2 << convertedLine << endl;

}

inputFile1.close();

outputFile1.close();

inputFile2.close();

outputFile2.close();

//cout << "Conversion complete." << endl;

ifstream in1("input1conv.txt");

ifstream in2("input2conv.txt");

vector<int> hash1;

vector<int> hash2;

int n;

in1 >> n;

Graph T1(n);

int a, b;

while (in1 >> a >> b) {

T1.addEdge(a, b);

}

T1.DFS(0);

hash1 = T1.getHash();

cout << "T1 tree\n";

T1.ShowTree();

in2 >> n;

Graph T2(n);

while (in2 >> a >> b) {

T2.addEdge(a, b);

}

T2.DFS(0);

hash2 = T2.getHash();

cout << "T2 tree\n";

T2.ShowTree();

if (hash1 == hash2) {

cout << "Изоморфны";

}

else {

cout << "Неизоморфны";

}

cout << "Неизоморфны";

return 0;

}

void Graph::DFS(int startVertex) //deep search

{

stack<int> stack;

stack.push(startVertex);

visited[startVertex] = true;

while (!stack.empty()) {

int currentVertex = stack.top();

stack.pop();

hash.push\_back(currentVertex);

for (int neighbor : adjLists[currentVertex]) {

stack.push(neighbor);

visited[neighbor] = true;

}

}

}

vector<int> Graph::getHash() {

return hash;

}

void Graph::ShowTree() {

std::cout << "Tree Structure:" << std::endl;

printTreeRecursive(0, 0, nodeMap, true);

}

void Graph::printTreeRecursive(int vertex, int level, std::unordered\_map<int, int>& nodeMap, bool isLastChild) {

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

if (i == level - 1 && !isLastChild) {

std::cout << "├─"; // Ветка

}

else if (i == level - 1 && isLastChild) {

std::cout << "└─"; // Последняя ветка

}

else {

std::cout << " "; // Промежуток до следующей вершины

}

}

std::cout << "Vertex " << vertex << std::endl;

int numChildren = adjLists[vertex].size();

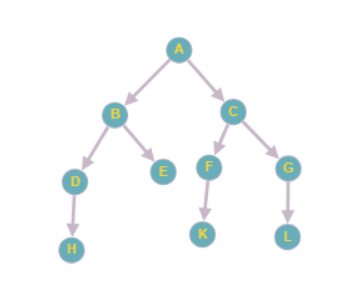
int currentChild = 0;

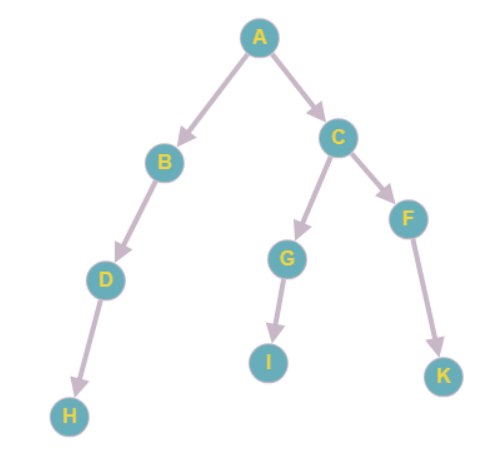
for (int neighbor : adjLists[vertex]) {

printTreeRecursive(neighbor, level + 1, nodeMap, ++currentChild == numChildren);

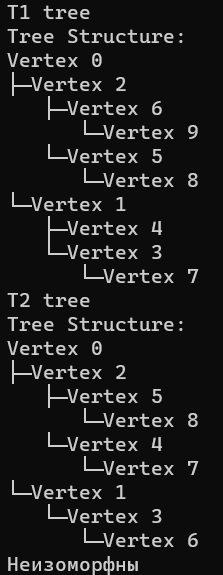
}

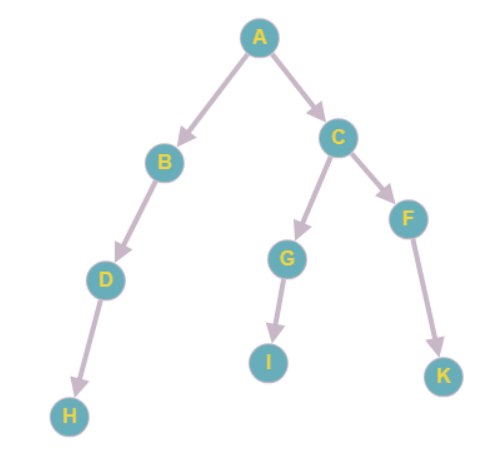
}





Оба дерева высоты 4. Дерево 2 является несбалансированным.

После приведения к изоморфному виду оба графа будут выглядеть так



Вывод: изучил обходы бинарных деревьев, алгоритмы проверки на изморфность.