Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №10-11

По дисциплине «МОИС»

Тема: «Теория графов»

Выполнил:

Студент 2 курса

Группы ИИ-21(2)

Кирилович А. А.

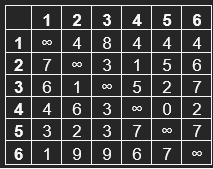
Проверил:

Козинский А.А.

Брест 2022

Вариант 5

Задание 1: Решить задачу коммивояжера.



#include <iostream>

#include <vector>

#include <sstream>

#include <string>

#include <algorithm>

#include <fstream>

using namespace std;

#define INF 9999999

void print\_traveling\_salesman(int arr[], vector<vector<int>> matrix, int start) {

        int size\_path = 0;

        vector<int> size;//вектор содержащий стоимость от точки до точки

        cout << "path: ";

        for(int j = 0;j<matrix.size(); j++){

            if(j != matrix.size() - 1){

                size\_path += matrix[ arr[j]-1 ][ arr[j+1]-1 ];

                size.push\_back(matrix[ arr[j]-1 ][ arr[j+1]-1 ]);

            }

            else{//возвращение в стартовую точку

                size\_path += matrix[ arr[j]-1 ][ arr[0] - 1];

                size.push\_back(matrix[ arr[j]-1 ][ arr[0] - 1 ]);

            }cout << arr[j] <<" ";

        }

        cout<<start + 1;

        cout<<"   size\_path: "<<size\_path;

        cout<<"   price: ";

        for(auto elem: size){cout<<elem<<" "; }

        cout<<endl;

    }

    void traveling\_salesman(int arr[], int n, vector<vector<int>> matrix,int start,int k = 0) {//перестановки для комивояжёра

        int temp = arr[start];

        for (int i = start; i > 0; i--) { arr[i] = arr[i-1]; }

        arr[0] = temp;

        while (true) {//генерация перестановок

            if(arr[0] == start + 1){//если стартовая точка стоит в начале перестановок, то переставляем следующие 5 элементов

                print\_traveling\_salesman(arr, matrix,start);

                int i = n - 2;// Ищем индекс первого элемента, который можно заменить

                while (i >= 0 && arr[i] >= arr[i+1]) { i--; }

                // Если такого элемента нет, завершаем цикл

                if (i < 0) { break; }

                // Ищем индекс первого элемента справа от arr[i], который меньше arr[i]

                int j = n - 1;

                while (arr[i] >= arr[j]) { j--; }

                swap(arr[i], arr[j]);// Меняем местами arr[i] и arr[j]

                reverse(arr + i + 1, arr + n);// Переворачиваем массив справа от arr[i]

            } else break;// чтобы не делать лишние расчеты перестановок, когда стартовая точка не в начале

        }}

int main() {

    string filename = "salesman.txt";

    vector<vector<int>> matrix

    ifstream file(filename);

    if (file.is\_open()) {

        string line;

        while (getline(file, line)) {

            vector<int> row;

            istringstream iss(line);

            string num;

            while (iss >> num) {

                if (num == "∞") { row.push\_back(INF); }

                else { row.push\_back(stod(num)); }

            }matrix.push\_back(row);

        }

        file.close();

        for (auto row : matrix) {

            for (auto num : row) { cout << num << " "; }

            cout << endl;

        }

    } else { cout << "Error opening file: " << filename << endl; }

    int start = 0;

    int num = matrix.size();

    int arr[num];

    for(int i = 1; i <=num;i++){ arr[i-1] = i; }

    traveling\_salesman(arr,num,matrix,start);

    return 0;

}



Задание 2: найти максимальный поток в заданной транспортной сети

#include <iostream>

#include <limits.h>

#include <queue>

#include <vector>

#include <string.h>

using namespace std;

bool bfs(vector<vector<int>> rGraph, int s, int t, vector<int>& parent)

{

    const int V = rGraph.size();

    bool visited[V];

    memset(visited, 0, sizeof(visited));

    queue<int> q;

    q.push(s);

    visited[s] = true;

    parent[s] = -1;

    while (!q.empty()) {

        int u = q.front();

        q.pop();

        for (int v = 0; v < V; v++) {

            if (visited[v] == false && rGraph[u][v] > 0) {

                if (v == t) {

                    parent[v] = u;

                    return true;

                }

                q.push(v);

                parent[v] = u;

                visited[v] = true;

            }

        }

    }

    return false;

}

int fordFulkerson(vector<vector<int>> graph, int s, int t)

{

    int u, v;

    const int V = graph.size();

    vector<vector<int>> rGraph(V , vector<int>(V , 0));

    for (u = 0; u < V ; u++)

        for (v = 0; v < V ; v++)

            rGraph[u][v] = graph[u][v];

    vector<int> parent(V);

    int max\_flow = 0;

    while (bfs(rGraph, s, t, parent)) {

        int path\_flow = INT\_MAX;

        for (v = t; v != s; v = parent[v]) {

            u = parent[v];

            path\_flow = min(path\_flow, rGraph[u][v]);

        }

        for (v = t; v != s; v = parent[v]) {

            u = parent[v];

            rGraph[u][v] -= path\_flow;

            rGraph[v][u] += path\_flow;

        }

        max\_flow += path\_flow;

    }

    cout<<endl;

    for(int i = 0;i<rGraph.size();i++){

        for(int j = 0;j<rGraph.size();j++){

            cout<<rGraph[i][j]<<" ";

        }

        cout<<endl;

    }cout<<endl;

    return max\_flow;

}

int main()

    vector<vector<int>> graph = {

{ 0, 7, 8, 0, 0, 0 },

                      { 0, 0, 2, 3, 6, 0 },

                      { 0, 0, 0, 2, 2, 0 },

                      { 0, 0, 0, 0, 0, 7 },

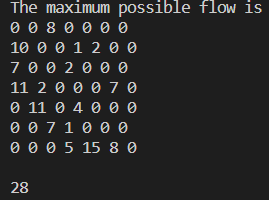
                      { 0, 0, 0, 0, 0, 9 },

                      { 0, 0, 0, 0, 0, 0 }};

    cout << "The maximum possible flow is "

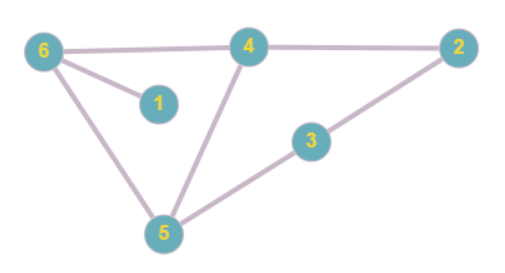
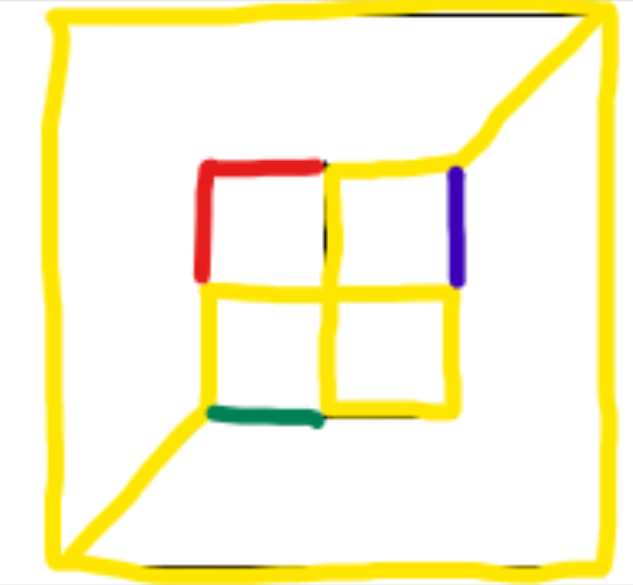
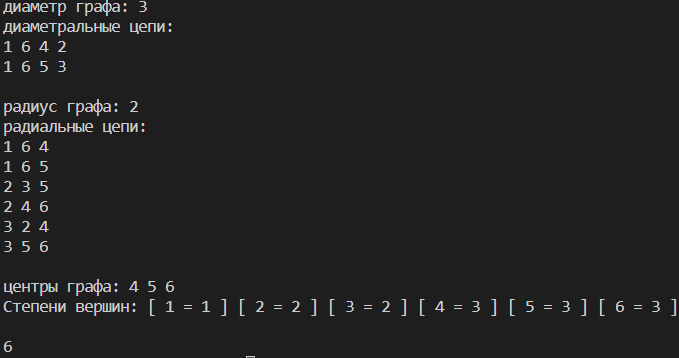
         << fordFulkerson(graph, 0, 5);

    return 0;



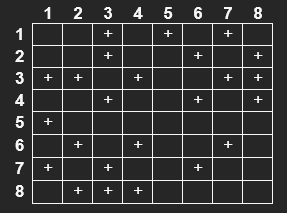
Задание 3: 1)Определите, из какого минимального числа кусков проволоки можно спаять данный каркас (толщина всех ребер каркаса должна быть одинаковой). Ответ обоснуйте. 2)Изобразите все реберно-непересекающиеся цепи, на которые можно разбить ребра графа, соответствующего данному каркасу (т.е. покажите, как спаять такие каркасы из минимального числа кусков проволоки). 3)Построить неориентированный граф G = <V,R> (множества V и R указаны для каждого варианта). Для графа G найти:

его диаметр и все диаметральные цепи; его радиус и все радиальные цепи; все центры графа; степень каждой его вершины; все разделяющие вершины.



Задание 4:

Найдите минимальное число валов, на которые можно поместить шестерни.



#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>

std::vector<std::vector<int>> getSubsets(std::vector<int>& nums) {

    int n = nums.size();

    int numSubsets = pow(2, n);

    std::vector<std::vector<int>> subsets(numSubsets, std::vector<int>());

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

        for (int j = 0; j < n; j++) if (i & (1 << j)) subsets[i].push\_back(nums[j]);

    return subsets;}

std::vector<int> min\_dominating\_set(std::vector<std::vector<int>> &adjMatrix) {

    std::vector<int> nums;

    for(int i = 0; i < adjMatrix.size(); i++) nums.push\_back(i);

    std::vector<std::vector<int>> subsets = getSubsets(nums);

    int n = adjMatrix.size();

    int m = subsets.size();

    std::vector<int> minSet;

    int minSize = n + 1;

    for (int i = 0; i < m; i++) {

        std::vector<int> subset = subsets[i];

        int subsetSize = subset.size();

        bool isDominating = true;

        for (int j = 0; j < n; j++) {

            if (find(subset.begin(), subset.end(), j) == subset.end()) {

                bool isAdjacent = false;

                for (int k = 0; k < subsetSize; k++) {

                    if (adjMatrix[j][subset[k]] == 1) {

                        isAdjacent = true;

                        break;

                    }}

                if (!isAdjacent) {

                    isDominating = false;

                    break;

                }}}

        if (isDominating && subsetSize < minSize) {

            minSet = subset;

            minSize = subsetSize;

        }}

    return minSet;}

int main() {

    std::vector<std::vector<int>> adjancy = {

        {0,0,1,0,1,0,1,0},

        {0,0,1,0,0,1,0,1},

        {1,1,0,1,0,0,1,1},

        {0,0,1,0,0,1,0,1},

        {1,0,0,0,0,0,0,0},

        {0,1,0,1,0,0,1,0},

        {1,0,1,0,0,1,0,0},

        {0,1,1,1,0,0,0,0}

    std::vector<int> min\_set = min\_dominating\_set(adjancy);

    std::cout << "min domination set size: " << min\_set.size() << std::endl;

    std::cout << "min domination set: {";

    for (auto elem : min\_set) std::cout << elem << " ";

    std::cout << "}";}

