`

Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №10-11

По дисциплине «Математические основы интеллектуальных систем»

Тема: «ТЕОРИЯ ГРАФОВ»

Выполнил:

Студент 2 курса

Группы ИИ-21

Литвинюк Т. В.

Проверил:

Козинский А. А.

Брест 2023

`

Цель: научиться решать задачи по графам.

Ход работы:
Вариант 7

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 = 2;
    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;
}
```

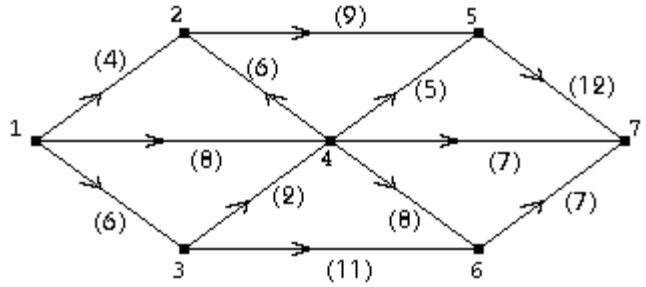
	1	2	3	4	5	6
1	∞	9	8	4	7	8
2	7	∞	5	4	8	2
3	9	8	∞	6	8	4
4	6	4	2	∞	5	6
5	4	2	1	8	∞	6
6	2	6	3	9	5	∞

```
path: 3 5 6 2 4 1 3   size_path: 38   price: 8 6 6 4 6 8
path: 3 5 6 4 1 2 3   size_path: 43   price: 8 6 9 6 9 5
path: 3 5 6 4 2 1 3   size_path: 42   price: 8 6 9 4 7 8
path: 3 6 1 2 4 5 3   size_path: 25   price: 4 2 9 4 5 1
path: 3 6 1 2 5 4 3   size_path: 33   price: 4 2 9 8 8 2
path: 3 6 1 4 2 5 3   size_path: 23   price: 4 2 4 4 8 1
path: 3 6 1 4 5 2 3   size_path: 22   price: 4 2 4 5 2 5
```

Задание 2

Найти максимальный поток в заданной транспортной сети, начиная с полученного полного потока.

`




```
#include <iostream>
#include <limits.h> // Библиотека для работы с числами
#include <queue> // Библиотека для работы с очередью
#include <vector> // Библиотека для работы с векторами
#include <string.h> // Библиотека для работы со строками
using namespace std;
// Функция для поиска пути по алгоритму BFS
bool bfs(vector<vector<int>> rGraph, int s, int t, vector<int>& parent){
    const int V = rGraph.size(); // Получаем количество вершин графа
    bool visited[V]; // Массив для хранения информации о посещенных вершинах
    memset(visited, 0, sizeof(visited)); // Инициализируем массив значением false
    queue<int> q; // Очередь для хранения вершин
    q.push(s); // Добавляем начальную вершину
    visited[s] = true; // Отмечаем начальную вершину как посещенную
    parent[s] = -1; // Родитель начальной вершины устанавливаем как -1 (так как начальная вершина является корнем)
    while (!q.empty()) { // Пока очередь не пуста
        int u = q.front(); // Извлекаем вершину из очереди
        q.pop(); // Удаляем вершину из очереди
        for (int v = 0; v < V; v++) { // Проходим по всем вершинам графа
            if (visited[v] == false && rGraph[u][v] > 0) { // Если вершина не была посещена и между вершинами есть ребро
                if (v == t) { // Если найден путь до конечной вершины
                    parent[v] = u; // Устанавливаем родителя конечной вершины
                    return true; // Возвращаем true, так как путь найден
                }
                q.push(v); // Добавляем вершину в очередь
                parent[v] = u; // Устанавливаем родителя вершины
                visited[v] = true; // Отмечаем вершину как посещенную
            }}}return false; // Если пути до конечной вершины не найдено, то возвращаем false
}

// Функция для реализации алгоритма Форда-Фалкерсона
int fordFulkerson(vector<vector<int>> graph, int s, int t){
    int u, v;
    const int V = graph.size(); // Получаем количество вершин графа
    vector<vector<int>> rGraph(V , vector<int>(V , 0)); // Создаем резидуальный граф
    for (u = 0; u < V ; u++)
        for (v = 0; v < V ; v++)
            rGraph[u][v] = graph[u][v]; // Инициализируем ребра резидуального графа
    vector<int> parent(V); // Создаем массив для хранения родителей каждой вершины в bfs
    int max_flow = 0; // Изначально максимальный поток равен 0
    while (bfs(rGraph, s, t, parent)) { // Пока есть увеличивающий путь в резидуальном графе
        int path_flow = INT_MAX; // Изначально значение потока на пути равно максимальному возможному значению
        for (v = t; v != s; v = parent[v]) { // Находим минимальное значение потока на увеличивающем пути
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }
        for (v = t; v != s; v = parent[v]) { // Обновляем значения ребер на увеличивающем пути и обратных ребер
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }max_flow += path_flow; // Добавляем значение потока на увеличивающем пути к общему максимальному потоку
    }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; // Возвращаем максимальный поток
}
int main(){
    vector<vector<int>> graph = { { 0, 4, 6, 0, 0, 0,7 },
                      { 0, 0, 0, 0, 9, 0,0 },
                      { 0, 0, 0, 2, 0, 11,0 },
                      { 0, 6, 0, 0, 5,8,7 },
                      { 0, 0, 0, 0, 0, 0,12 },
                      { 0, 0, 0, 0, 0, 0,7 },
                      { 0, 0, 0, 0, 0, 0, 0}};
    cout << "The maximum possible flow is "
             << fordFulkerson(graph, 0, 5); // Вызываем функцию для
          нахождения максимального потока
}
```

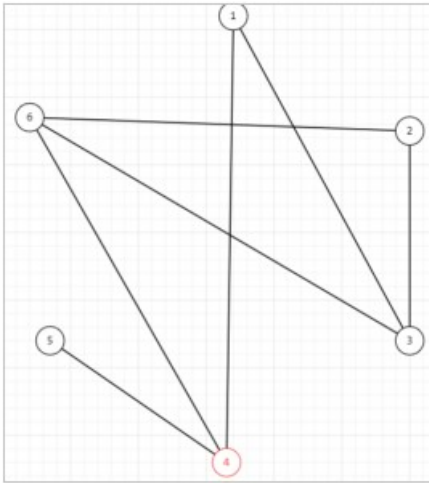
```
The maximum possible flow is
0 4 0 0 0 0 7
0 0 0 0 9 0 0
6 0 0 2 0 5 0
0 6 0 0 5 8 7
0 0 0 0 0 0 12
0 0 6 0 0 0 7
0 0 0 0 0 0 0
```

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.




```
#include <iostream>
#include <vector>
#include <algorithm>
#include <fstream>
#include <string>
#include <stack>
using namespace std;
const int INF = 1e9;
vector<int> reading_file(string path){
    ifstream input_file("task3.txt");
    vector<int> numbers;

    if (input_file.is_open()) {
        string line;
        while (getline(input_file, line)) {
            // Ищем все числа в строке
            for (int i = 0; i < line.length(); i++) {
                if (isdigit(line[i])) {
                    string number_str = "";
                    while (isdigit(line[i])) {
                        number_str += line[i];
                        i++;
                    }
                    int number = stoi(number_str);
                    numbers.push_back(number);
                }
            }
        }
        input_file.close();
    }
    else {
        cout << "Не удалось открыть файл" << endl;

    }

    return numbers;
}
std::vector<std::vector<int>> floyd_warshall(const std::vector<std::vector<int>>& graph) {
    int n = graph.size();
    std::vector<std::vector<int>> dist(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            dist[i][j] = graph[i][j];
        }
    }
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                dist[i][j] = std::min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
    return dist;
}

int diameter(const std::vector<std::vector<int>>& graph) {
    auto dist = floyd_warshall(graph);
    int diameter = 0;
    for (int i = 0; i < graph.size(); ++i) {
        for (int j = 0; j < graph.size(); ++j) {
            if (dist[i][j] != INF) {
                diameter = std::max(diameter, dist[i][j]);
            }
        }
    }
    return diameter;
}

std::vector<std::vector<int>> diameter_paths(const std::vector<std::vector<int>>& graph,int diameter) {
    auto dist = floyd_warshall(graph);
    std::vector<std::vector<int>> paths;
    for (int i = 0; i < graph.size(); ++i) {
        for (int j = i + 1; j < graph.size(); ++j) {
```

```
    `

            if (dist[i][j] == diameter) {
                std::vector<int> path = {i, j};
                while (true) {
                    int k = -1;
                    for (int l = 0; l < graph.size(); ++l) {
                        if (l != path[path.size() - 2] && graph[path[path.size() - 2]][l] != INF && dist[l][j] ==
dist[path[path.size() - 2]][j] - graph[path[path.size() - 2]][l]) {
                            k = l;
                            break;
                        }
                    }
                    if (k == -1) {
                        break;
                    }
                    path.insert(path.end() - 1, k);
                }
                paths.push_back(path);
            }
        }
    }
    return paths;
}
int radius_graph(const std::vector<std::vector<int>>& graph) {
    auto dist = floyd_warshall(graph);
    int radius = INF;
    for (const auto& row : dist) {
        radius = std::min(radius, *std::max_element(row.begin(), row.end()));
    }
    return radius;
}

std::vector<std::vector<int>> radial_paths(const std::vector<std::vector<int>>& graph,int radius) {
    auto dist = floyd_warshall(graph);
    std::vector<std::vector<int>> paths;

    for (int i = 0; i < graph.size(); i++) {
        for (int j = i + 1; j < graph.size(); j++) {
            if (dist[i][j] == radius) {
                std::vector<int> path = {i, j};
                while (true) {
                    int k = -1;
                    for (int l = 0; l < graph.size(); l++) {
                        if (l != path[path.size() - 2] && graph[path[path.size() - 2]][l] != INF && dist[l][j] ==
dist[path[path.size() - 2]][j] - graph[path[path.size() - 2]][l]) {
                            k = l;
                            break;
                        }
                    }
                    if (k == -1) {
                        break;
                    }
                    path.insert(path.end() - 1, k);
                }
                paths.push_back(path);
            }
        }
    }
    return paths;
}

std::vector<int> centers(std::vector<std::vector<int>> graph) {
    int n = graph.size();
    std::vector<std::vector<int>> dist = floyd_warshall(graph);
    std::vector<int> eccentricities(n);
    for (int i = 0; i < n; ++i)
        eccentricities[i] = *std::max_element(dist[i].begin(), dist[i].end());
    int min_eccentricity = *std::min_element(eccentricities.begin(), eccentricities.end());
    std::vector<int> centers;
    for (int i = 0; i < n; ++i)
        if (eccentricities[i] == min_eccentricity)
            centers.push_back(i + 1);
    return centers;
}

void degree(vector<vector<int>> graph){
    int n = graph.size();
    std::vector<int> degrees(n, 0);

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (graph[i][j] != INF)
                degrees[i]++;
    std::cout << "Степени вершин: ";
    for (int i = 0; i < n; i++)
        std::cout <<"[ "<<i+1<<" = "<< degrees[i] << " ] ";
    std::cout << std::endl;
}
void dfs(int u, int &time, vector<bool> &visited, vector<int> &discovery,
         vector<int> &low, vector<int> &parent, vector<int> &ap, vector<vector<int>> &graph) {
    int children = 0;
    visited[u] = true;
    discovery[u] = low[u] = time;
    time++;
```

```
    `

    for (int v = 0; v < graph.size(); v++) {
        if (graph[u][v] != INF) {
            if (!visited[v]) {
                children++;
                parent[v] = u;
                dfs(v, time, visited, discovery, low, parent, ap, graph);
                low[u] = min(low[u], low[v]);

                if (parent[u] == -1 && children > 1) {
                    ap.push_back(u + 1);
                }
                if (parent[u] != -1 && low[v] >= discovery[u]) {
                    ap.push_back(u + 1);
                }

            } else if (v != parent[u]) {
                low[u] = min(low[u], discovery[v]);
            }
        }
    }
}
vector<int> articulation_points(vector<vector<int>> &graph) {
    int n = graph.size();
    int time = 0;
    vector<bool> visited(n, false);
    vector<int> discovery(n, INF), low(n, INF), parent(n, -1), ap;

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i, time, visited, discovery, low, parent, ap, graph);
        }
    }
    return ap;
}

// Пример использования
int main() {
    int n = 0;  // количество вершин в графе
    vector<int> vertex = reading_file("data.txt");
    for( auto tow:vertex){
        if(tow > n){
            n = tow;
        }
    }
    vector<vector<int>> g(n, vector<int>(n, INF));
    for(int i=0;i<vertex.size();i+=2){
        int x = vertex[i]-1;
        int y = vertex[i+1]-1;
        g[x][y] = g[y][x] = 1;
    }
    cout<<endl;
    for(auto row:g){
        for(auto col:row){
            if(col == INF){
                cout<< "inf" <<" ";
            }
            else cout<<col<<" ";
        }cout<<endl;
    }

    int diametr = diameter(g);
    cout<<"диаметр графа: "<<diametr<<endl;
    vector<vector<int>> path_diametr = diameter_paths(g,diametr);

    cout<<"диаметральные цепи: "<<endl;
    for(auto row:path_diametr){
        for(auto col:row){
            cout<<col+1<<" ";
        }cout<<endl;
    }cout<<endl;


    int radius = radius_graph(g);
    cout<<"радиус графа: "<<radius<<endl;
    vector<vector<int>> path_radius =
radial_paths(g,radius);
    cout<<"радиальные цепи: "<<endl;
    for(auto row:path_radius){
        for(auto col:row){
            cout<<col+1<<" ";
        }cout<<endl;
    }cout<<endl;


    cout<<"центры графа: ";
    vector<int> centres_graph = centers(g);
    for(auto row:centres_graph){
        cout<<row<<" ";
    }cout<<endl;

    degree(g);

    cout<<endl;
```

```
диаметр графа: 3
диаметральные цепи:
2 6 4 5
3 1 4 5

радиус графа: 2
радиальные цепи:
1 3 2
1 4 5
1 3 6
2 6 4
3 1 4
5 4 6

центры графа: 1 4 6
Степени вершин: [ 1 = 2 ] [ 2 = 2 ] [ 3 = 3 ] [ 4 = 3 ] [ 5 = 1 ] [ 6 = 3 ]

точки сочленения: 4
```

```
`

    cout<<"точки сочленения: ";
    vector<int> articulation = articulation_points(g);
    for(auto row:articulation){
        cout<<row<<" ";
    }
}
```

Задание 4

Коробка скоростей – механизм для изменения частоты вращения ведомого вала при неизменной частоте вращения ведущего. Это изменение происходит за счет того, что находящиеся внутри коробки шестерни (зубчатые колеса) вводятся в зацепление специальным образом. Одна из задач, стоящая перед конструктором коробки, заключается в минимизации ее размеров, а это часто сводится к минимизации числа валов, на которых размещаются шестерни.

Некоторые шестерни не должны находиться на одном валу, например, они могут быть в зацеплении или их общий вес велик для одного вала и т.д. Для каждого варианта в таблице указаны такие пары шестерен. Найдите минимальное число валов, на которые можно поместить шестерни.

	1	2	3	4	5	6	7	8
1			+	+		+		+
2				+		+		+
3	+				+			
4	+	+				+	+	
5			+					
6	+	+		+			+	
7				+		+		+
8	+	+					+	

```
#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 = {
         {0,0,1,1,0,1,0,1},
         {0,0,0,1,0,1,0,1},
         {1,0,0,0,1,0,0,0},
         {1,1,0,0,0,1,1,0},
         {0,0,1,0,0,0,0,0},
         {1,1,0,1,0,0,1,0},
         {0,0,0,1,0,1,0,1},
         {1,1,0,0,0,0,1,0}};
    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 << "}";}
```

```
min domination set size: 3
min domination set: {0 2 3 }
```

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