L06. GRAFURI

Header.h

```
#pragma once

#include <iostream>
#include <stack>
#include <queue>

using namespace std;

void DFS(int** a, int n, int i, int*viz, queue<int>& L);
void DFS_it(int** a, int n, int i, int*viz, queue<int>& L);

void BFS_it(int** a, int n, int i, int* viz, queue<int>& L);
void BFS(int** a, int n, int* viz, queue<int>& q, queue<int>& L);

void ListNodes(int* pred, int s, int d, stack<int> &sp);

int Sort(int* q, int* dist, int& n);
void Dijkstra(int** a, int n, int s, int* prev, int* dist);

int minDistNode(int* q, int* dist, int* viz, int n);
void Dijkstra_Updated(int** a, int n, int s, int* prev, int* dist);

typedef pair<int, int> dist2node;
struct Order
{
	bool operator()(const dist2node& a, const dist2node& b);
};
void Dijkstra_PQ(int** a, int n, int s, int* prev, int* dist);
```

Functii.cpp

```
#include "Header.h"

void DFS(int** a, int n, int i, int*viz, queue<int>& L)
{
	viz[i] = 1;
	L.push(i);
	for (int k = 0;k < n;k++)
	{
		if (a[i][k] && !viz[k])
			DFS(a, n, k, viz, L);
```

```
	}
}

void DFS_it(int** a, int n, int i, int*viz, queue<int>& L)
{
	stack<int> s;
	int val;
	s.push(i);
	while (!s.empty())
	{
		val = s.top();
		s.pop();
		if (!viz[val])
		{
			viz[val] = 1;
			L.push(val);
			for (int j = n - 1;j >= 0;j--)
				if (!viz[j] && a[val][j])
					s.push(j);
		}
	}
}

void BFS_it(int** a, int n, int i, int* viz, queue<int>& L)
{
	queue<int> q;
	int val;
	q.push(i);
	while (!q.empty())
	{
		val = q.front();
		q.pop();
		if (!viz[val])
		{
			viz[val] = 1;
			L.push(val);
			for (int j = 0;j < n;j++)
				if (!viz[j] && a[val][j])
					q.push(j);
		}
	}
}

void BFS(int** a, int n, int* viz, queue<int>& q, queue<int>& L)
{
	int val;
	if (!q.empty())
	{
		val = q.front();
		q.pop();
		L.push(val);
```

```
            for (int j = 0;j < n;j++)
                  if (!viz[j] && a[val][j])
                  {
                        q.push(j);
                        viz[j] = 1;
                  }
            BFS(a, n, viz, q, L);
      }
}

void Dijkstra(int** a, int n, int s, int* prev, int* dist)
{
      int v, u, n1;
      int* q;

      q = new int[n];

      for (v = 0;v < n;v++)
      {
            dist[v] = std::numeric_limits<int>::max(); q[v] = v;
prev[v] = -2;
      }
      dist[s] = 0;

      n1 = n;

      while (n1 != 0)
      {
            u = Sort(q, dist, n1);
            for (v = 0;v < n;v++)
            {
                  if (a[u][v] && dist[u] + a[u][v] <= dist[v])
                  {
                        dist[v] = dist[u] + a[u][v];
                        prev[v] = u;
                  }
            }
      }
}

int Sort(int* q, int* dist, int& n)
{
      int aux, i, lim = n - 1, ok = 1;
      do {
            ok = 0;
            for (i = 0;i < lim;i++)
                  if (dist[i]>dist[i + 1])
                  {
                        aux = q[i]; q[i] = q[i + 1]; q[i + 1] = aux;
                        aux = dist[i]; dist[i] = dist[i + 1]; dist[i +
1] = aux;
```

```
                }
            lim--;
      } while (ok);
      aux = q[0];
      for (i = 1;i <= n - 1;i++)
            q[i - 1] = q[i];
      n--;
      return aux;
}

int minDistNode(int* q, int* dist, int* viz, int n)
{
      int min = std::numeric_limits<int>::max();
      int poz = -1;
      for (int i = 0;i < n;i++)
            if (min >= dist[i] && !viz[i])
            {
                  min = dist[i]; poz = i;
            }
      return poz;
}

void Dijkstra_Updated(int** a, int n, int s, int* prev, int* dist)
{
      int v, u;
      int* q;
      int* viz;

      q = new int[n];
      viz = new int[n];

      for (v = 0;v < n;v++)
      {
            dist[v] = std::numeric_limits<int>::max();
            q[v] = v; prev[v] = -2; viz[v] = 0;
      }
      dist[s] = 0;

      do {
            u = minDistNode(q, dist, viz, n); viz[u] = 1;
            if (u != -1)
                  for (v = 0;v < n;v++)
                        if (a[u][v] && dist[u] + a[u][v] <= dist[v])
                        {
                              dist[v] = dist[u] + a[u][v];
                              prev[v] = u;
                        }
      } while (u != -1);
}

bool Order::operator()(const dist2node& a, const dist2node& b)
```

```
{
	return a.second > b.second;
}

void Dijkstra_PQ(int** a, int n, int s, int* prev, int* dist)
{
	int v, u;
	int* viz = new int[n];
	priority_queue<dist2node, vector<dist2node>, Order> q;

	for (v = 0;v < n;v++)
	{
		dist[v] = std::numeric_limits<int>::max();
		prev[v] = -2; viz[v] = 0;
	}
	dist[s] = 0;
	q.push(make_pair(s, dist[s]));
	viz[s] = 1;
	while (!q.empty())
	{
		u = q.top().first;
		q.pop();
		for (v = 0;v < n;v++)
			if (a[u][v] && dist[u] + a[u][v] <= dist[v])
			{
				dist[v] = dist[u] + a[u][v];
				prev[v] = u;
				if (!viz[v])
					q.push(make_pair(v, dist[v]));
			}
	}
}

void ListNodes(int* prev, int s, int d, stack<int>& sp)
{
	int k = d;
	sp.push(k);
	while (k != s)
	{
		sp.push(prev[k]);
		k = prev[k];
	}
}
```

Main.cpp

```
#include "Header.h"

int main()
{
```

```
	int** a;
	int* viz;
	queue<int> L;
	queue<int> q;
	stack<int> sp;
	int n, m;
	int i, j, k;
	FILE* f;
	int* prev;
	int* dist;
	int s, d;

	if (fopen_s(&f, "data.txt", "r"))
	{
		fprintf(stderr, " \n Eroare la deschiderea fisierului
\n");
		exit(EXIT_FAILURE);
	}
	fscanf_s(f, "%d %d", &n, &m);

	a = new int*[n];
	viz = new int[n];
	prev = new int[n];
	dist = new int[n];

	for (k = 0;k < n;k++)
	{
		a[k] = new int[n];
	}
	for (i = 0;i < n;i++)
		for (j = 0;j < n;j++)
			a[i][j] = 0;
	for (k = 0;k < m;k++)
	{
		fscanf_s(f, "%d %d", &i, &j);
		fscanf_s(f, "%d", &a[i - 1][j - 1]);
	}
	if (fclose(f))
	{
		fprintf(stderr, " \n Eroare la deschiderea fisierului
\n");
		exit(EXIT_FAILURE);
	}

	cout << "\n Matricea de adiacenta:\n";
	for (i = 0;i < n;i++)
	{
		for (j = 0;j < n;j++)
			cout << a[i][j] << "\t";
		cout << endl;
	}
```

```
	cout << endl;

	cout << "\n Traversals: ";
	cout << "\n DFS rec: ";
	for (k = 0;k < n;k++)
		viz[k] = 0;
	DFS(a, n, 0, viz, L);
	while (!L.empty())
	{
		cout << L.front() << " ";
		L.pop();
	}
	cout << endl;

	cout << "\n DFS iter: ";
	for (k = 0;k < n;k++)
		viz[k] = 0;
	DFS_it(a, n, 0, viz, L);
	while (!L.empty())
	{
		cout << L.front() << " ";
		L.pop();
	}
	cout << endl;

	cout << "\n BFS iter: ";
	for (k = 0;k < n;k++)
		viz[k] = 0;
	BFS_it(a, n, 0, viz, L);
	while (!L.empty())
	{
		cout << L.front() << " ";
		L.pop();
	}
	cout << endl;

	cout << "\n BFS rec: ";
	for (k = 0;k < n;k++)
		viz[k] = 0;
	q.push(0);
	viz[0] = 1;
	BFS(a, n, viz, q, L);
	while (!L.empty())
	{
		cout << L.front() << " ";
		L.pop();
	}
	cout << endl;


	//Dijkstra(a, n, 0, prev, dist);
```

```
	//Dijkstra_Updated(a, n, 0, prev, dist);
	Dijkstra_PQ(a, n, 0, prev, dist);
	cout << endl << "i prev[i] dist[i]\n";
	for (int i = 0;i < n;i++)
	{
		cout << i + 1 << "\t" << prev[i] + 1 << "\t" << dist[i] << endl;
	}
	cout << endl;
	s = 0;
	d = 5;
	ListNodes(prev, s, d, sp);

	cout << "\n Dijkstra's alg: path(" << s + 1 << "; " << d + 1 << "): ";
	while (!sp.empty())
	{
		cout << sp.top()+1 << " ";
		sp.pop();
	}
	cout << endl << endl;
	return 0;
}
```

data.txt

```
6 9
1 2 2
1 3 4
2 3 1
2 4 4
2 5 2
3 5 3
4 6 2
5 4 3
5 6 2

5 7
1 2 10
1 4 30
1 5 100
2 3 50
3 5 10
4 3 20
4 5 60

7 12
1 2 1
```

```
1 3 1
2 4 1
2 5 1
3 5 1
4 1 1
4 7 1
5 4 1
5 6 1
5 7 1
6 7 1
7 5 1
```