L06. GRAFURI – CU IMPLEM COADA, STIVĂ

Header.h

```
#pragma once
#include "QueueStack.h"

void DFS(int** a, int n, int i, int*viz, Elem*& L);
void DFS_it(int** a, int n, int i, int*viz, Elem*& L);

void BFS_it(int** a, int n, int i, int* viz, Elem*& L);

void ListNodes(int* pred, int s, int d, Elem* &sp);
int minDistNode(int* q, int* dist, int* viz, int n);
void Dijkstra_Updated(int** a, int n, int s, int* prev, int* dist);
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, Elem*& L)
{
	viz[i] = 1;
	Enqueue(L, 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, Elem*& L)
{
	Elem* s;
	Init(s);
	int val;
	Push(s, i);
	while (Pop(s,val))
	{
		if (!viz[val])
		{
			viz[val] = 1;
			Enqueue(L, val);
			for (int j = n - 1;j >= 0;j--)
```

```
                        if (!viz[j] && a[val][j])
                              Push(s, j);
            }
      }
}

void BFS_it(int** a, int n, int i, int* viz, Elem*& L)
{
      Elem* q;
      int val;
      Init(q);
      Enqueue(q, i);
      while (Dequeue(q,val))
      {
            if (!viz[val])
            {
                  viz[val] = 1;
                  Enqueue(L, val);
                  for (int j = 0;j < n;j++)
                        if (!viz[j] && a[val][j])
                              Enqueue(q, j);
            }
      }
}

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);
}


void Dijkstra_PQ(int** a, int n, int s, int* prev, int* dist)
{
	int v, u;
	int* viz = new int[n];
	Dist2Node* q;
	InitPQ(q);

	for (v = 0;v < n;v++)
	{
		dist[v] = std::numeric_limits<int>::max();
		prev[v] = -2; viz[v] = 0;
	}
	dist[s] = 0;
	EnqueuePQ(q, s, dist[s]);
	viz[s] = 1;
	while (DequeuePQ(q, u))
	{
		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])
					EnqueuePQ(q, v, dist[v]);
			}
	}
}

void ListNodes(int* prev, int s, int d, Elem*& sp)
{
	int k = d;
	Push(sp, k);
	while (k != s)
	{
		Push(sp, prev[k]);
		k = prev[k];
```

```
	}
}
```

Main.cpp

```
#include "Header.h"

int main()
{
	int** a;
	int* viz;
	Elem* L;
	Elem* sp;
	int n, m;
	int i, j, k;
	FILE* f;
	int* prev;
	int* dist;
	int s, d, val;

	Init(L);
	Init(sp);

	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 (Dequeue(L,val))
            cout << val << " ";
      cout << endl;

      cout << "\n DFS iter: ";
      for (k = 0;k < n;k++)
            viz[k] = 0;
      DFS_it(a, n, 0, viz, L);
      while (Dequeue(L, val))
            cout << val << " ";
      cout << endl;

      cout << "\n BFS iter: ";
      for (k = 0;k < n;k++)
            viz[k] = 0;
      BFS_it(a, n, 0, viz, L);
      while (Dequeue(L, val))
            cout << val << " ";
      cout << endl;

     //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 (Pop(sp,val))
	{
		cout << val + 1 << " ";
	}
	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
```

QueueStack.h

```
#pragma once
#include <iostream>
using namespace std;

struct Elem {
	int data;
	Elem* succ;
};

struct Dist2Node {
	int node;
	int dist;
	Dist2Node* succ;
};

void Init(Elem*& e);
bool isEmpty(Elem* e);
void List(Elem* e);
void List1(Elem* e);

bool Push(Elem*& stack, int val);
bool Pop(Elem*& stack, int& val);
bool Top(Elem* stack, int& val);

bool Enqueue(Elem*& queue, int val);
bool Dequeue(Elem*& queue, int& val);
bool Front(Elem* queue, int& val);

void InitPQ(Dist2Node*& e);
bool isEmptyPQ(Dist2Node* e);
bool EnqueuePQ(Dist2Node*& queue, int valN, int valD);
bool DequeuePQ(Dist2Node*& queue, int& valN);
```

QueueStack.cpp

```
#include "Header.h"

void Init(Elem*& e)
{
	e = NULL;
}

bool isEmpty(Elem* e)
{
	return e == NULL;
}
```

```
void List(Elem* e)
{
	if (e)
	{
		List(e->succ);
		cout << e->data << " ";
	}
	else
		cout << endl;
}

void List1(Elem* e)
{
	if (e)
	{
		cout << e->data << " ";
		List(e->succ);
	}
	else
		cout << endl;
}

bool Push(Elem*& stack, int val)
{
	//cout << "\n Insert " << val << " to stack.";
	Elem* p = new Elem;
	p->data = val;
	p->succ = stack;
	stack = p;
	return true;
}

bool Pop(Elem*& stack, int& val)
{
	if (isEmpty(stack))
		return false;
	else
	{
		Elem* p = stack;
		val = p->data;
		//cout << "\n Pop " << val << " from stack.";
		stack = stack->succ;
		delete p;
		p = NULL;
		return true;
	}
}

bool Top(Elem* stack, int& val)
{
	if (isEmpty(stack))
```

```
            return false;
      else
      {
            val = stack->data;
            //cout << "\n Top " << val << " in stack.";
            return true;
      }
}

bool Enqueue(Elem*& queue, int val)
{
      Elem* p = new Elem;
      p->data = val;
      //cout << "\n Push " << val << " to queue.";
      p->succ = queue;
      queue = p;
      return true;
}

bool Dequeue(Elem*& queue, int& val)
{
      if (isEmpty(queue))
            return false;
      else
      {
            Elem* p = queue;
            if (p->succ)
            {
                  Elem* q;
                  while (p->succ)
                  {
                        q = p;
                        p = p->succ;
                  }
                  q->succ = NULL;
            }
            else
                  queue = NULL;
            val = p->data;
            delete p;
            return true;
      }
}

bool Front(Elem* queue, int& val)
{
      if (isEmpty(queue))
            return false;
      else
      {
            Elem* p = queue;
```

```
            while (p->succ)
                  p = p->succ;
            val = p->data;
            return true;
      }
}


void InitPQ(Dist2Node*& e)
{
      e = NULL;
}

bool isEmptyPQ(Dist2Node* e)
{
      return e == NULL;
}

bool EnqueuePQ(Dist2Node*& queue, int valN, int valD)
{
      Dist2Node* p = new Dist2Node;
      p->node = valN;
      p->dist = valD;
      p->succ = NULL;

      if (isEmptyPQ(queue))
            queue = p;
      else
      {
            if (queue->dist >= valD)
            {
                  p->succ = queue;
                  queue = p;
            }
            else
            {
                  Dist2Node* n1=queue, *n2 = queue;
                  while (n2->succ)
                  {
                        n1 = n2;
                        if (n2->dist <= valD)
                              n2 = n2->succ;
                  }
                  n1->succ = p;
                  if (n1 != n2)
                        p->succ = n2;
            }
      }
      return true;
}
```

```
bool DequeuePQ(Dist2Node*& queue, int& valN)
{
	if (isEmptyPQ(queue))
		return false;
	else
	{
		Dist2Node* p = queue;
		if (p->succ)
		{
			Dist2Node* q;
			while (p->succ)
			{
				q = p;
				p = p->succ;
			}
			q->succ = NULL;
		}
		else
			queue = NULL;
		valN = p->node;
		delete p;
		return true;
	}
}
```