

**UNIVERSITATEA DE STAT DIN MOLDOVA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ  
SPECIALITATEA INFORMATICA**

**Pavlovschi Cătălin**

**Dare de seama**

***Lucrare de laborator nr.1:***

***Algoritmica Grafurilor***

## Lucrare de laborator nr.1

***N=11***

### Varianta 3

1. Să se introducă un graf prin matricea de adiacență, formați din ea matricea de incidență, apoi, din ea, formați matricea de adiacență a grafului muchiilor.

```
#include <vector>
#include <cassert>
#include <iostream>

typedef std::vector<bool> matrix_row;
typedef std::vector<matrix_row> matrix;

matrix adj =
{
    { 0, 1, 1, 0 },
    { 1, 0, 0, 1 },
    { 1, 0, 0, 1 },
    { 0, 1, 1, 0 }
};

matrix adiacenta_to_incident(const matrix &adj)
{
    int cols = adj.size();
    assert(cols > 0);

    int rows = adj[0].size();
    assert(rows > 0);

    assert(rows == cols);

    int edge = 0;
    matrix incident;

    for (int col = 0; col < cols; ++col) {

        for (int row = 0; row <= col; ++row) {
            if (adj[col][row]) {
                incident.push_back(matrix_row(cols, 0));
                incident[edge][row] = incident[edge][col] = 1;
                ++edge;
            }
        }
    }

    return incident;
}
```

```

matrix incident_to_adiacenta(const matrix &inc)
{
    int edges = inc.size();
    assert(edges > 0);

    int vertices = inc[0].size();
    assert(vertices > 0);

    matrix adiacenta(vertices, matrix_row(vertices, 0));

    for (int edge = 0; edge < edges; ++edge) {
        int a = -1, b = -1, graf = 0;
        for (; graf < vertices && a == -1; ++graf) {
            if (inc[edge][graf]) a = graf;
        }
        for (; graf < vertices && b == -1; ++graf) {
            if (inc[edge][graf]) b = graf;
        }
        if (b == -1) b = a;
        adiacenta[a][b] = adiacenta[b][a] = 1;
    }

    return adiacenta;
}

void print_matrix(const matrix &m)
{
    int cols = m.size();
    if (cols == 0) return;
    int rows = m[0].size();
    if (rows == 0) return;

    for (int c = 0; c < cols; ++c) {
        for (int r = 0; r < rows; ++r) {
            std::cout << m[c][r] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

int main()
{
    matrix incident = adiacenta_to_incident(adj);
    print_matrix(incident);

    matrix adiacenta = incident_to_adiacenta(incident);
    print_matrix(adiacenta);

    return 0;}

```

2. Se dă un graf neorientat conex. Utilizând parcurgerea în lăţime găsiţi diametrul grafului.

```
#include <sstream>
#include <vector>
#include <string>
#include <algorithm>
#include <iostream>

int main() {
    std::string inps = R"(
        0 0 2 0
        4 0 3 0
        0 0 0 2
        0 1 0 0
    )";
    std::stringstream inp;
    inp.str(inps);

    size_t const inf = size_t(-1) >> 2;
    std::vector<std::vector<size_t>> d;
    // Input
    std::string line;
    while (std::getline(inp, line)) {
        if (line.find_last_not_of(" rnt") == std::string::npos)
            continue;
        d.resize(d.size() + 1);
        std::stringstream ss;
        ss.str(line);
        size_t x = 0;
        while (ss >> x)
            d.back().push_back(x);
    }
    // metoda Floyd-Warshall
    for (size_t i = 0; i < d.size(); ++i)
        for (size_t j = 0; j < d[i].size(); ++j)
            if (d[i][j] == 0 && i != j)
```

```
        d[i][j] = inf;
    for (size_t k = 0; k < d.size(); ++k)
        for (size_t i = 0; i < d.size(); ++i)
            for (size_t j = 0; j < d.size(); ++j)
                d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]);

    // diametrul
    size_t maxd = 0, maxi = 0, maxj = 0;
    for (size_t i = 0; i < d.size(); ++i)
        for (size_t j = 0; j < d.size(); ++j)
            if (maxd < d[i][j]) {
                maxd = d[i][j];
                maxi = i;
                maxj = j;
            }
    // Output
    std::cout << "Diametrul= " << maxd << std::endl;
}
```

3. Rezolvați problemele N și 14-N, unde N este numărul dvs de ordine.

3. Fie  $A$  o submulțime de vârfuri ale grafului neorientat  $G = (X; U)$ , iar  $k$  – numărul muchiilor care au exact o extremitate în  $A$ . Să se demonstreze că numărul  $k$  este de aceeași paritate cu numărul vârfurilor de grad impar din  $A$ .

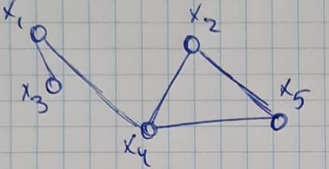
11. Să se demonstreze că dacă  $G$  este un graf neorientat cu  $n > 2$  vârfuri și  $m > C_{n-1}^2$  muchii, atunci  $G$  este conex.

③  $A$  – submulțime de vârfuri ale  $G(X; U)$

$k$  – nr muchiilor ce au exact o extremitate

Demonstr.:  $k$  are aceeași paritate cu nr vârfurilor de grad impar din  $A$

$G(X; U)$ ,  $X=5$   $U=6$



Nr.  $\deg x_i = 3$  (impar) impar

$k = 1 \rightarrow$  impar

c.e.t.d.

⑪  $n > 2$   $m > C_{n-1}^2$

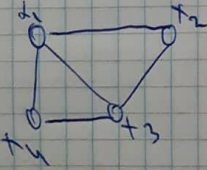
$C_{n-1}^2 = \frac{n-1(n-1-1)}{2}$

Fie  $n = 4$

$m > \frac{4-1(4-2)}{2} = \frac{3 \cdot 2}{2} = 3$

$m > 3$

Fie  $m = 5 \Rightarrow G(4; 5)$  este



$\rightarrow G(4; 5) -$  conex

c.e.t.d.

# Teorie

O matrice binara  $A = ||a_{ij}||$  de dimensiunea  $n \times n$  se numeste **matrice de adiacenta** a grafului  $G$  cu multimea de vârfuri  $X_G = \{x_1, x_2, \dots, x_n\}$ , daca:

$$a_{ij} = \begin{cases} 1, & \text{dacă } [i, j] \in U \\ 0, & \text{dacă } [i, j] \notin U \end{cases}$$

Matricea de adiacenta a grafului este o matrice simetrica cu elementele de pe diagonala principala egale cu zero. Liniile si coloanele acestei matrici corespund vârfurilor  $x_1, x_2, \dots, x_n$  ale grafului.

În mod analog se defineste **matricea de incidenta** a grafului  $G = (X; U)$ , cu multimile de vârfuri si muchii  $X = \{x_1, x_2, \dots, x_n\}$ ,  $U = \{u_1, u_2, \dots, u_m\}$ . Aceasta este de asemenea o matrice binara  $B = ||b_{ij}||$  de dimensiunea  $n \times m$  cu elementele:

$$b_{ij} = \begin{cases} 1, & \text{daca } x_i \text{ este incident muchiei } u_j, \\ 0, & \text{în caz contrar.} \end{cases}$$

**Traversarea grafurilor în lăţime sau Breadth-First** se bazează pe următoarea tehnică:

- fie un graf  $G = (X, U)$  cu  $n$  noduri şi un nod de plecare **ns** numit şi nod sursă
- căutarea în lăţime explorează sistematic muchiile grafului  $G$  pentru a "**descoperi**" fiecare nod accesibil din **ns**. Algoritmul calculează distanţa (cel mai mic număr de muchii) de la **ns** la toate vârfurile accesibile lui. El produce un "arbore de lăţime" cu rădăcina în **ns**, care conţine toate nodurile accesibile. Pentru fiecare nod **v** accesibil din **ns**, calea din arborele de lăţime de la **ns** la **v** corespunde "**celui mai scurt drum**" de la **ns** la **v**, adică conţine un număr minim de muchii.

Un graf neorientat se numeşte **graf conex** dacă pentru oricare două vârfuri  $x$  şi  $y$  diferite ale sale, există cel puţin un lanţ care le leagă, adică  $x$  este extremitatea iniţială şi  $y$  este extremitatea finală.