**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\_rand;

typedef std::vector<matrix\_rand> 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\_rand(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;}

1. Se dă un graf neorientat conex. Utilizând parcurgerea in lățime găsiți 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

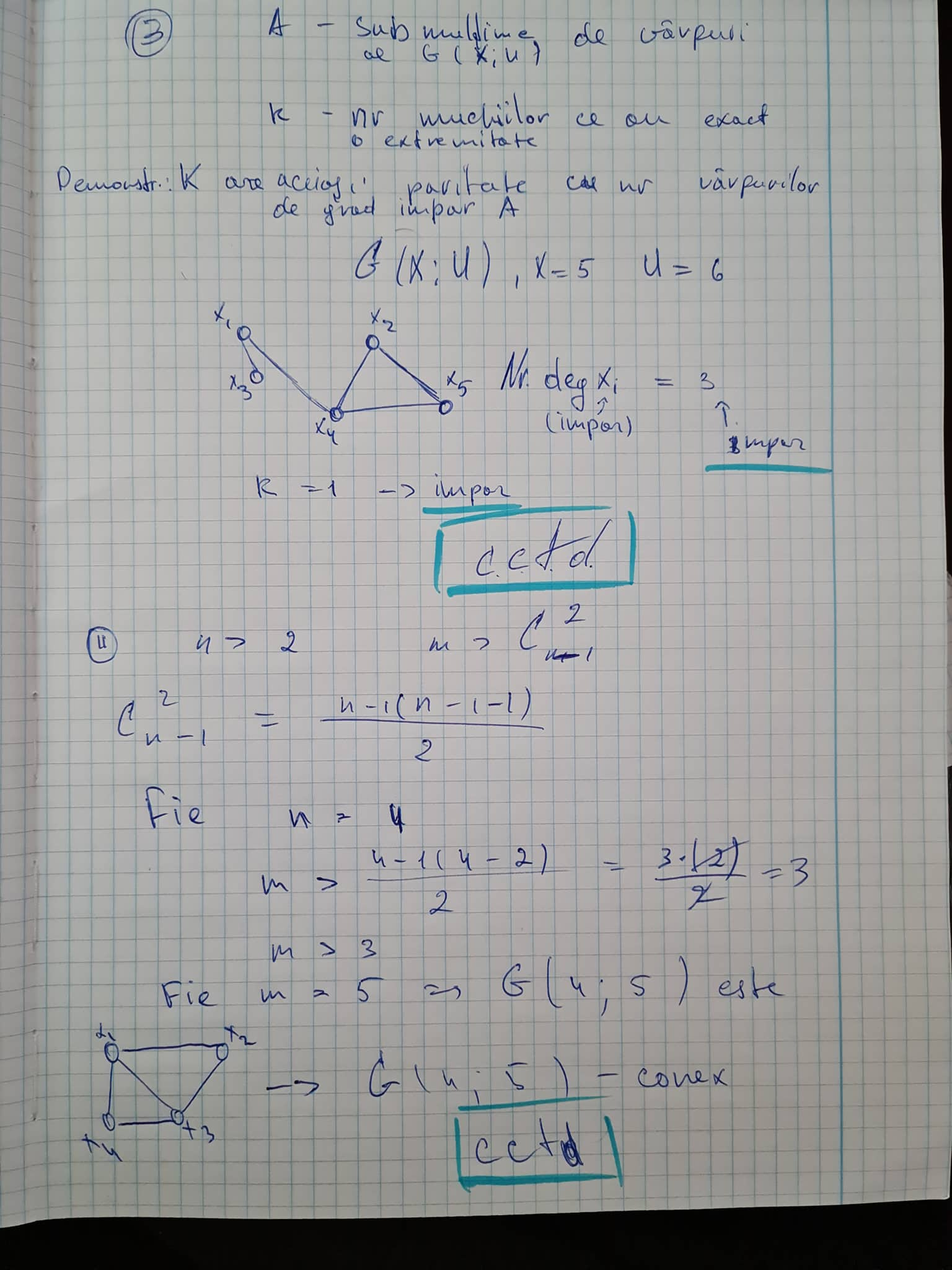
std::cout << "Diametrul= " << maxd << std::endl;

}

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

**3.** Fie *A* o submulţime de vârfuri ale grafului neorientat **, iar *k* – numărul muchiilor care au exact o extremitate în *A*. Să se demonstreze că numărul *k* este de aceeaşi paritate cu numărul vârfurilor de grad impar din *A*.

**11.** Să se demonstreze că dacă *G* este un graf neorientat cu  vârfuri şi  muchii, atunci *G* este conex.



Teorie

O matrice binara A = ||ai,j|| de dimensiunea n\*n se numeste ***matrice de adiacenta*** a grafului G cu multimea de vârfuri XG = {x1 , x2 ,..., xn }, daca:

Picture

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 x1 , x2 ,..., xn ale grafului.

În mod analog se defineste ***matricea de incidenta*** a grafului G = (X;U), cu multimile de vârfuri si muchii   
X = {x1 , x2 ,..., xn }, U = {u1 ,u2 ,...,um}. Aceasta este de asemenea o matrice binara ij B = ||bi,j|| de dimensiunea n\*m cu elementele:

Text

Description automatically generated with medium confidence

|  |
| --- |
| ***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 numește ***graf conex*** dacă pentru oricare două vârfuri x și y diferite ale sale, există cel puțin un lanț care le leagă, adică x este extremitatea inițială și y este extremitatea finală.