Universitatea Tehnică a Moldovei

Facultatea - Calculatoare Informatică și Microelectronică

Disciplina: *Analiza și proiectarea algoritmilor*

**Raport**

Lucrarea de laborator Nr.4

Tema: Algoritmii Dijkstra și Floyd

A efectuat: st.gr. TI-207 Bunescu Gabriel

A verificat: lect.univ. Bîtca Ernest

Chișinău 2021

**Cuprins**

[I. Scopul lucrării 2](#_Toc88582074)

[II. Sarcina lucrării 2](#_Toc88582075)

[III. Considerații teoretice 3](#_Toc88582076)

[IV. Analiza empirică a algortimilor 4](#_Toc88582077)

[V. Concluzie: 5](#_Toc88582078)

# I. Scopul lucrării

1. Studierea programării dinamice, implementarea algoritmilor Dijkstra și Floydю

# II. Sarcina lucrării

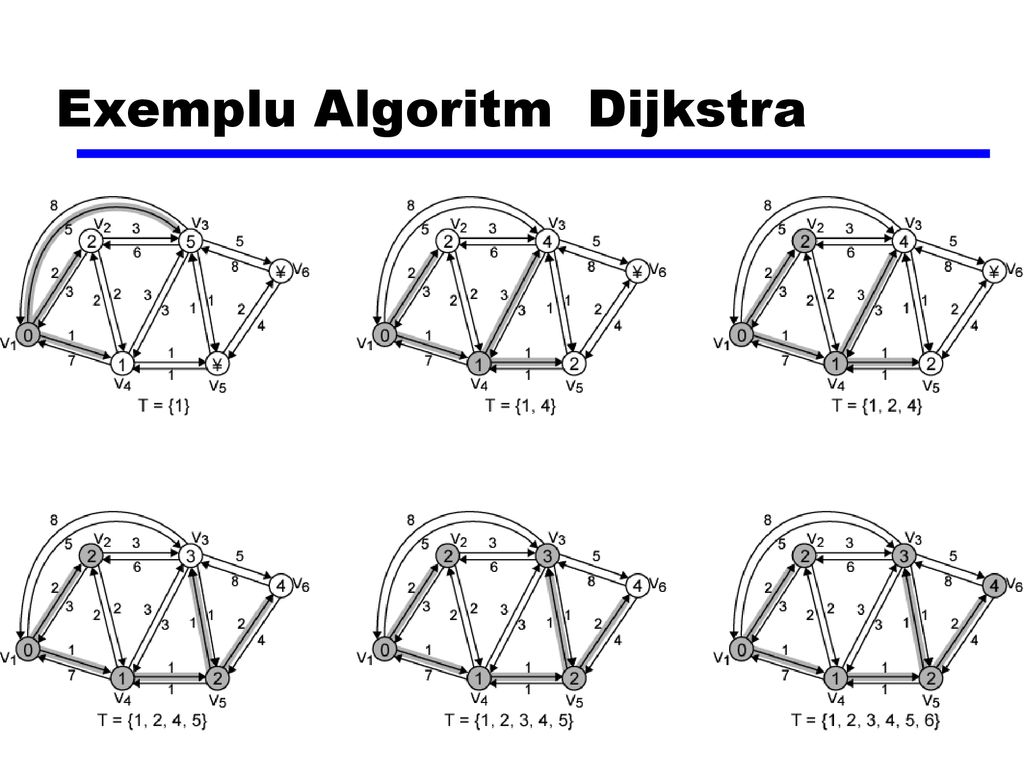
1. Să se învețe algoritmii
2. Să implementați algoritmii
3. Comparații:
4. Cazul din toate nodurile în toate nodurile;
5. Cazul din nod sursă (1 singur nod) în toate nodurile.

# III. Considerații teoretice

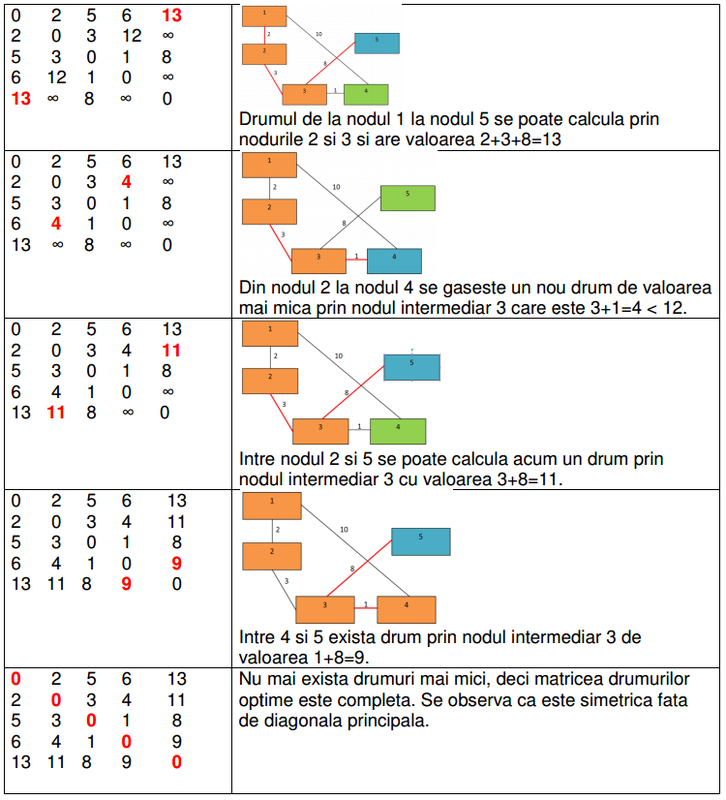
**Algoritmul lui Dijkstra** este o [metodă](https://ro.wikipedia.org/wiki/Metod%C4%83" \o "Metodă) de a stabili [drumul](https://ro.wikipedia.org/wiki/Drum" \o "Drum) de cost minim de la un nod de start la oricare altul dintr-un [graf](https://ro.wikipedia.org/wiki/Graf" \o "Graf). Numele este dat de [Edsger Dijkstra](https://ro.wikipedia.org/wiki/Edsger_Dijkstra" \o "Edsger Dijkstra), [savantul](https://ro.wikipedia.org/wiki/Savant" \o "Savant) care l-a descoperit.

Algoritm

1. Se creează o listă cu distanțe, o listă cu nodul anterior, o listă cu nodurile vizitate și un nod curent.
2. Toate valorile din lista cu distanțe sunt inițializate cu o valoare infinită, cu excepția nodului de start, care este setat cu 0.
3. Toate valorile din lista cu nodurile vizitate sunt setate cu fals.
4. Toate valorile din lista cu nodurile anterioare sunt inițializate cu -1.
5. Nodul de start este setat ca nodul curent.
6. Se marchează ca vizitat nodul curent.
7. Se actualizează distanțele, pe baza nodurilor care pot fi vizitate imediat din nodul curent.
8. Se actualizează nodul curent la nodul nevizitat care poate fi vizitat prin calea cea mai scurtă de la nodul de start.
9. Se repetă (de la punctul 6) până când toate nodurile sunt vizitate.



Algoritmul Roy-Floyd este folosit in diverse domeni des intalnite, de la controlul avioanelor de pe un anumit aeroport pana la jocuri informatice, rolul principal fiind acela de gasire a drumului de cost minim intre un obiect principal si o tinta anume.



# IV. Analiza empirică a algortimilor

Tabelul 1. Caz favorabil

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vârfuri | | 10 | 50 | 100 | 250 | 500 | 750 | 1000 |
| Floyd | Iteratii | 1036 | 126176 | 1004851 | 15655876 | 125124251 | 422015231 | 1000498501 |
| Timp | 0.000122 | 0.000870 | 0.007545 | 0.143780 | 0.7143 | 3.016323 | 5.804519 |
| Dijkstra | Iteratii | 1080 | 127400 | 1009800 | 156787000 | 125249000 | 424127250 | 1000998000 |
| Timp | 0.00206 | 0.002025 | 0.010310 | 0.208867 | 1.957478 | 4.626759 | 11.136161 |

Fig1.1.Cazul favorabil,Iterații

Fig1.2.Cazul favorabil,Timpul

Tabelul 2. Caz mediu

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vârfuri | | 10 | 50 | 100 | 250 | 500 | 750 | 1000 |
| Floyd | Iteratii | 1130 | 135470 | 1061769 | 16199641 | 128030545 | 429558611 | 1014805179 |
| Timp | 0.000109 | 0.00101 | 0.11328 | 0.13747 | 0.790954 | 3.36433 | 8.220425 |
| Dijkstra | Iteratii | 1150 | 138700 | 1074300 | 16248750 | 129617000 | 438321750 | 1043641000 |
| Timp | 0.002088 | 0.001448 | 0.011797 | 0.185329 | 1.21134 | 4.229998 | 10.184295 |

Fig2.1.Cazul mediu,Iterații

Fig2.2.Cazul mediu,Timpul

Tabelul 3.Caz defavorabil

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vârfuri | | 10 | 50 | 100 | 250 | 500 | 750 | 1000 |
| Floyd | Iteratii | 1084 | 135156 | 1058732 | 16172400 | 127911506 | 429130764 | 1013838398 |
| Timp | 0.00016 | 0.001239 | 0.010198 | 0.197156 | 1.304249 | 4.756564 | 11.145589 |
| Dijkstra | Iteratii | 1260 | 139400 | 1070300 | 16403500 | 131849500 | 447167250 | 1076346000 |
| Timp | 0.000348 | 0.001349 | 0.012779 | 0.211794 | 1.645894 | 6.03904 | 15.445723 |

Fig3.1.Cazul defavorabil,Iterații

Fig3.2.Cazul defavorabil,Timpul

# V. Concluzie:

În această lucrare de laborator s-au studiat Algoritmii Dijkstra și Floyd, conform timpului de execuție și a numărului de iterații. Algoritmul Floyd verifică toate drupurile posibile din graf de la un nod (începul) anumit spre alt nod (sfîrșit), el parcurge pe rînd fiecare din drumurile posibile și îl alege doar pe cel minim de parcurgere, iar Algoritmii Dijkstra creează o listă cu distanțe, o listă cu nodul anterior, o listă cu nodurile vizitate și un nod current, toate valorile din lista cu distanțe sunt inițializate cu o valoare infinită, cu excepția nodului de start, care este setat cu 0, apoi toate valorile din lista cu nodurile vizitate sunt setate cu fals și toate valorile din lista cu nodurile anterioare sunt inițializate cu -1.Nodul de start este setat ca nodul current, se marchează ca vizitat nodul current, se actualizează distanțele, pe baza nodurilor care pot fi vizitate imediat din nodul current și se actualizează nodul curent la nodul nevizitat care poate fi vizitat prin calea cea mai scurtă de la nodul de start. Apoi se repetă până când toate nodurile sunt vizitate.

În cazul meu eu ma verificat trei cazurii de complexitare: favorabil, mediu și defavorabil, pentru diferite valori ale vîrfurilor de la 10 pînă la 1000. Și pot spune că algoritmii Dijkstra este mai lent și are nevoie de mai multe iterații și timp decânt algoritmii Floyd.

**VI Codul sursă:**

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#define max 1000

#define INF 99999999

using namespace std;

unsigned long long MS[max][max], MV[max][max], MVdij[max][max], MVfloid[max][max], n, parent[max], visited[max], path[max], disvec[max][max];

int count1 = 0, count2 = 0, tmpi, u, possition = 0, start, finish; void RESET() {

count1 = 0;

count2 = 0;

for (int i = 0; i<n; i++) {

parent[i] = 0;

visited[i] = 0;

for (int j = 0; j<n; j++) {

MVdij[i][j] = MV[i][j];

MVfloid[i][j] = MV[i][j];

}

}

}

void nr\_virf\_defavorabil() {

int j;

cout << "Nr. de varfuri: ";

cin >> n;

//cout << "Matricea de adiacenta\n";

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++)

if (j > i) {

MS[i][j] = rand() % 1000;}

else if (i > j) {

MS[i][j] = MS[j][i];}}}

void costurile\_defavorabil() {

//cout << "Costurile muchiilor \n";

for (int i = 0; i<n; i++) {

for (int j = 0; j<n; j++) {

if (MS[i][j] && i != j) {

//cout << i + 1 << " -> " << j + 1 << " : " << MS[i][j]<< endl;

MV[i][j] = MS[i][j];}

else

MV[i][j] = INF;}

}

RESET();

}

void nr\_virf\_favorabil() {

cout << "Nr. de varfuri: ";

cin >> n;

//cout << "Matricea de adiacenta\n";

for (int i = 0; i<n; i++) {

for (int j = 0; j<n; j++)

MS[i][i + 1] = rand() % 1000;

}

}

void costurile\_favorabil() {

// cout << "Costurile muchiilor \n";

for (int i = 0; i<n; i++) {

for (int j = 0; j<n; j++) {

if (MS[i][j] && i != j) {

// cout << i + 1 << " -> " << j + 1 << " : " << MS[i][j]<< endl;//

MV[i][j] = MS[i][j];}

else

MV[i][j] = INF;

}

}

RESET();}

void nr\_virf\_mediu() {

cout << "Nr. de virfuri: ";

cin >> n;

//cout << "Introduceti matricea de adiacenta\n";

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++)

if (i % 2 == 0 && j % 2 == 0) {

j = j + 1;

MS[i][j] = rand() % 1000;

}

else if (i % 2 != 0 && j % 2 != 0) {

j = j + 1;

MS[i][j] = rand() % 1000;

}

else if (i > j) {

MS[i][j] = MS[j][i];

}

}

}

void costurile\_mediu() {

//cout << "Costurile muchiilor \n";

for (int i = 0; i<n; i++) {

for (int j = 0; j<n; j++) {

if (MS[i][j] && i != j) {

//cout << i + 1 << " -> " << j + 1 << " : " << MS[i][j] <<endl;

MV[i][j] = MS[i][j];

}

else

MV[i][j] = INF;

}

}

RESET();

}

void DIJKSTRA() {

int distance[max],disvec[max][max],visited[max];

cout << "Introduceti 2 varfuri (de la - pana la): " << endl;

cout << "Din ";

cin >> start;

cout << "In ";

cin >> finish;

for (int ji = 0; ji < n; ji++) {

for (int i = 0; i < n; i++) {

distance[i] = INF;

visited[i] = 0;

}

distance[start - 1] = 0;

int st = 0;

for (int i = 0; i < n; i++) {

disvec[st][i] = distance[i];

}

for (int i = 0; i < n - 1; i++) {

int min = INF;

for (int i = 0; i < n; i++) {

if (!visited[i] && distance[i] <= min) {

min = distance[i];

tmpi = i;

count1++;

}

}

u = tmpi;

visited[u] = 1;

for (int i = 0; i < n; i++) {

if (!visited[i] && MVdij[u][i] && distance[u] != INF && distance[u] + MVdij[u] [i] < distance[i]) {

distance[i] = distance[u] + MVdij[u][i];

count1++;

}

}

st++;

for (int i = 0; i < n; i++) {

disvec[st][i] = distance[i];

count1++;

}

}

int k = finish - 1;

path[possition] = finish - 1;

while (path[possition] != start - 1) {

if (disvec[st][k] == disvec[st - 1][k]) {

st--;

}

else {

for (int j = 0; j < n; j++) {

if (disvec[st][k] == disvec[st - 1][j] + MVdij[j][k]) { path[++possition] = j;

k = j;

st--;

break;

}

}

}

}

}

cout<<"===========================================\n\n";

cout << "Drumul minin de la " << start << " pina la " << finish << " este " << distance[finish - 1] << "." << endl;

cout << "Nr. de iteratii: " << count1 << endl;

}

void FLOYD() {

int start, finish;

cout << "Introduceti 2 varfuri p/u aflarea drumului minim: " << endl;

cout << "Din ";

cin >> start;

cout << "In ";

cin >> finish;

for (int k = 0; k<n; k++)

for (int i = 0; i<n; i++)

for (int j = 0; j<n; j++) {

count2++;

if (MVfloid[i][k] && MVfloid[k][j] && i != j)

if (MVfloid[i][k] + MVfloid[k][j]<MVfloid[i][j] || MVfloid[i][j] == 0) { MVfloid[i][j] = MVfloid[i][k] + MVfloid[k][j];

count2++;

}

}

cout<<"=========================================\n\n";

cout << "Drumul minim de la " << start << " pina la " << finish << " este " << MVfloid[start - 1][finish - 1] << "." << endl;

cout << "Nr. de iteratii: " << count2 << endl;

}

int main() {

double t1, t2;

int ChooseMenu;

int x;

x: while (true) {

cout << "1. Caz FAVORABIL" << endl

<< "2. Caz MEDIU" << endl

<< "3. Caz DEFAVORABIL" << endl

<< "0. Iesire" << endl; cout<<"===========================================\n"; cout << endl << "Optiunea aleasa: ";

cin >> ChooseMenu; switch (ChooseMenu) { case 1: {

while (true) {

cout << "1. Introdu nr. de varfuri" << endl

<< "2. Algoritmul FLOYD" << endl

<< "3. Algoritmul DIJKSTRA" << endl

<< "0. Main menu" << endl; cout<<"===========================================\n"; cout << endl << "Optiunea aleasa: ";

cin >> ChooseMenu; switch (ChooseMenu) { case 1: {

nr\_virf\_favorabil(); costurile\_favorabil(); break;

}

case 2: {

t1 = clock();

FLOYD();

t2 = clock();

cout << "Timpul de functionare: " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC <<

" sec" << endl; cout<<"===========================================\n"; break;

}

case 3: {

t1 = clock();

DIJKSTRA();

t2 = clock();

cout << "Timpul de functionare: " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC <<

" sec" << endl; cout<<"===========================================\n";

break;

}

case 0: {

RESET();

goto x;

}

default: {

cout << "EROARE" << endl;

break;

}

}

}

case 2: {

while (true) {

cout << "1. Introdu nr. de varfuri" << endl

<< "2. Algoritmul FLOYD" << endl

<< "3. Algoritmul DIJKSTRA" << endl

<< "0. Main menu" << endl; cout<<"===========================================\n"; cout << endl << "Optiunea aleasa: ";

cin >> ChooseMenu; switch (ChooseMenu) { case 1: {

nr\_virf\_mediu(); costurile\_mediu(); break;

}

case 2: {

t1 = clock();

FLOYD();

t2 = clock();

cout << "Timpul de fucntionare: " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC <<

" sec" << endl; cout<<"===========================================\n";

break;

}

case 3: {

t1 = clock();

DIJKSTRA();

t2 = clock();

cout << "Timpul de functionare: " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC <<

" sec" << endl; break;

}

case 0: {

RESET();

goto x;

}

default: {

cout << "EROARE" << endl;

break;

}

}

}

case 3: {

while (true) {

cout << "1. Introdu nr. de varfuri" << endl

<< "2. Algoritmul FLOYD" << endl

<< "3. Algoritmul DIJKSTRA" << endl

<< "0. Main menu" << endl; cout<<"===========================================\n"; cout << endl << "Optiunea aleasa: ";

cin >> ChooseMenu; switch (ChooseMenu) { case 1: {

nr\_virf\_defavorabil(); costurile\_defavorabil(); break;

}

case 2: {

t1 = clock();

FLOYD();

t2 = clock();

cout << "Timpul functionare: " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC << " sec" << endl;

break;

}

case 3: {

t1 = clock();

DIJKSTRA();

t2 = clock();

cout << "Timpul de functionare: " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC <<

" sec" << endl; break;

}

case 0: {

RESET();

goto x;

}

default: {

cout << "EROARE" << endl;

break;

}

}

}

case 0: {

return 0;

}

default: {

cout << "EROARE" << endl;

break;

}}}}}}}