Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul ISA

# RAPORT

Lucrarea de laborator nr.4
*Analiza si proiectarea algoritmilor*

A efectuat:
st. gr. TI-173 Rosca Florin

A verificat:
lect., univ. Andrievschi-Bagrin Veronica

Chisinau 2018

## LUCRAREA DE LABORATOR NR.4

**Tema:** Metoda programarii dinamice

**Scopul lucrării:**
1. Studierea metodei programării dinamice.
2. Analiza şi implementarea algoritmilor de programare dinamică.
3. Compararea tehnicii greedy cu metoda de programare dinamică.

**Sarcina lucrării**

Sa se realizeze analiza empirica a algoritmilor Dijkstra modificat si Floyd pentru determinarea celor mai scurte drumuri intr-un graf.

```
function Floyd(L[1 .. n, 1 .. n])
1:  array D[1 .. n, 1 .. n]
2:  D ← L
3:  for k ← 1 to n do
4:        for i ←1 to n do
5:              for j ← 1 to n do
6:                    D[i, j] ← min(D[i, j], D[i, k D[k, j])
 return D
```

```
function Dijkstra(L[1 .. n, 1 .. n])
1: C ← {2, 3, ..., n}       {S = V \C există doar implicit}
2: for i ←  2 to n do D[i] ←  L[1, i]
3: repeat n–2 times
4:        v ←  vârful din C care minimizează D[v]
5:       C ←  C \ {v}         {si, implicit, S ←  S ∪ {v}}
6:        for fiecare w ∈C do
7:              D[w] ← min(D[w], D[v]+L[v, w])
return D
```

**Notiuni teoretice**

Dezvoltarea unui algoritm bazat pe programarea dinamică poate fi împărţită într-o secvenţă de patru paşi:
1. Caracterizarea structurii unei soluţii optime.
2. Definirea recursivă a valorii unei soluţii optime.
3. Calculul valorii unei soluţii optime într-o manieră de tip "bottom-up".
4. Construirea unei soluţii optime din informaţia calculată.

0

Paşii 1-3 sunt baza unei abordări de tip programare dinamică. Pasul 4 poate fi omis dacă se doreşte doar calculul unei singure soluţii optime. In vederea realizării pasului 4, deseori se

păstrează informaţie suplimentară de la execuţia pasului 3, pentru a uşura construcţia unei soluţii optimale.

Fie $G = <V, A>$ un graf orientat, unde $V$ este mulţimea vârfurilor şi $A$ este mulţimea arcelo. Fiecare arc are o lungime nenegativa. Unul din vârfuri este ales că vârf *sursă*. Problema este de a determina lungimea celui mai scurt drum de la sursă către fiecare vârf din graf.

Se va folosi un algoritm greedy, datorat lui Dijkstra (1959). Notăm cu $C$ mulţimea vârfurilor disponibile (candidaţii) şi cu $S$ mulţimea vârfurilor deja selectate. În fiecare moment, $S$ conţine acele vârfuri a căror distanţă minimă de la sursă este deja cunoscută, în timp ce mulţimea $C$ conţine toate celelalte vârfuri. La început, $S$ conţine doar vârful sursă, iar în final $S$ conţine toate vârfurile grafului. La fiecare pas, adăugam în $S$ acel vârf din $C$ a cărui distanţă de la sursă este cea mai mică.

Se spune, că un drum de la sursă către un alt vârf este *special*, dacă toate vârfurile intermediare de-a lungul drumului aparţin lui $S$. Algoritmul lui Dijkstra lucrează în felul următor. La fiecare pas al algoritmului, un tablou $D$ conţine lungimea celui mai scurt drum special către fiecare vârf al grafului. După ce se adaugă un nou vârf $v$ la $S$, cel mai scurt drum special către $v$ va fi, de asemenea, cel mai scurt dintre toate drumurile către $v$. Când algoritmul se termină, toate vârfurile din graf sunt în $S$, deci toate drumurile de la sursă către
celelalte vârfuri sunt speciale şi valorile din $D$ reprezintă soluţia problemei.

Presupunem că vârfurile sunt numerotate, $V = \{1, 2, ..., n\}$, vârful 1 fiind sursa, şi că matricea $L$ dă lungimea fiecărui arc, cu $L[i, j] = \propto$, dacă arcul $(i, j)$ nu există. Soluţia se va construi în tabloul $D[2 .. n]$. Algoritmul este:

**function** *Dijkstra*($L[1 .. n, 1 .. n]$)
1: $C \leftarrow \{2, 3, ..., n\}$ {$S = V \setminus C$ există doar implicit}
2: **for** $i \leftarrow 2$ **to** $n$ **do** $D[i] \leftarrow L[1, i]$
3: **repeat** $n–2$ **times**
4: $v \leftarrow$ vârful din $C$ care minimizează $D[v]$
5: $C \leftarrow C \setminus \{v\}$ {si, implicit, $S \leftarrow S \cup \{v\}$}
6: **for** fiecare $w \in C$ **do**
7: $D[w] \leftarrow \min(D[w], D[v]+L[v, w])$
**return** $D$

**Proprietatea 1.** În algoritmul lui Dijkstra, dacă un vârf $i$
*a*) este în $S$, atunci $D[i]$ dă lungimea celui mai scurt drum de la sursă către $i$;
*b)* nu este în $S$, atunci $D[i]$ dă lungimea celui mai scurt drum special de la sursă către $i$.

La terminarea algoritmului, toate vârfurile grafului, cu excepţia unuia, sunt în $S$. Din proprietatea precedenta, rezulta că algoritmul lui Dijkstra funcţionează corect.

Fie $G = <V, A>$ un graf orientat, unde $V$ este mulţimea vârfurilor şi $A$ este mulţimea arcelor. Fiecărui arc i se asociază o lungime nenegativă. Să se calculeze lungimea celui mai scurt drum între fiecare pereche
de varfuri.

Vom presupune că vârfurile sunt numerotate de la 1 la $n$ şi că matricea $L$ dă lungimea fiecărui arc:
$L[i, i] = 0$, $L[i, j] \geq 0$ pentru $i \neq j$, $L[i, j] = \propto$ dacă arcul $(i, j)$ nu există.

Principiul optimalităţii este valabil: dacă cel mai scurt drum de la $i$ la $j$ trece prin varful $k$, atunci porţiunea de drum de la $i$ la $k$, cât şi cea de la $k$ la $j$, trebuie să fie, de asemenea, optime.

Construim o matrice $D$ care să conţină lungimea celui mai scurt drum între fiecare pereche de vârfuri. Algoritmul de programare dinamică iniţializează pe $D$ cu $L$.

$$D_k[i, j] = \min(D_{k-1}[i, j], D_{k-1}[i, k]\ D_{k-1}[k, j])$$

unde s-a facut uz de principiul optimalităţii pentru a calcula lungimea celui mai scurt drum faţă de $k$. Implicit, s-a considerat că un drum optim care trece prin $k$ nu poate trece de două ori prin $k$. Acest algoritm simplu este datorat lui Floyd (1962):

```
1: array D[1 .. n, 1 .. n]
2: D ← L
3: for k ← 1 to n do
4:     for i ←1 to n do
5:         for j ← 1 to n do
6: D[i, j] ← min(D[i, j], D[i, k D[k, j]) return D
```

Se poate deduce că algoritmul lui Floyd necesită un timp în $O(n^3)$. Un alt mod de a rezolva această problemă este să se aplice algoritmul *Dijkstra* prezentat mai sus de $n$ ori, alegând mereu un alt vârf sursă. Se obtine un timp în $n\ O\ (n^2)$, adică tot în $O\ (n^3)$. Algoritmul lui Floyd, datorită simplităţii lui, are însă constanta multiplicativă mai mică, fiind probabil mai rapid în practică.

**SARCINA DE BAZĂ:**

1. De studiat metoda programării dinamice de proiectare a algoritmilor.
2. De implementat într-un limbaj de programare algoritmii prezentaţi mai sus.
3. De făcut analiza empirică a acestor algoritmi pentru un graf rar şi pentru un graf dens.
4. De alcătuit un raport.

**Codul programului in C++**

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define max 500
#define INF INT_MAX
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];
                }
        }

}
//cazul defavorabil
void nr_virf_defavorabil() {
        cout << "Numarul de virfuri: ";
        cin >> n;
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++)
                        //cin >> MS[i][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();
}
//cazul favorabil
void nr_virf_favorabil() {
        cout << "Numarul de virfuri: ";
        cin >> n;
        //cout << "Matricea de adiacenta\n";
        for (int i = 0; i<n; i++) {
                for (int j = 0; j<n; j++)
                        //cin >> MS[i][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();
}
//cazul mediu
void nr_virf_mediu() {
	cout << "Numarul de virfuri: ";
	cin >> n;
	//cout << "Introduceti matricea de adiacenta\n";
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++)
			//cin >> MS[i][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 virfuri unde doriti sa cautati" << 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 << "Drumul minin de la " << start << " pina la " << finish << " este " <<
distance[finish - 1] << "." << endl;
		cout << "Numarul de iteratii : " << count1 << endl;

}
void FLOYD() {
		int start, finish;
		cout << "Introduceti 2 virfuri pentru a afla drumul 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 << "Drumul minim de la " << start << " pina la " << finish << " este " <<
MVfloid[start - 1][finish - 1] << "." << endl;
		cout << "Numarul de iteratii : " << count2 << endl;
}

int main() {
		double t1, t2;
		int ChooseMenu;
		int x;
x: while (true) {
		system("cls");
		cout << "1. Cazul favorabil." << endl
			<< "2. Cazul mediu." << endl
			<< "3. Cazul defavorabil." << endl
			<< "0. Exit." << endl;
		cout << endl << ">>> ";
		cin >> ChooseMenu;
		system("cls");
		switch (ChooseMenu) {
		case 1: {
			while (true) {
				system("cls");
				cout << "1. Introduceti numarul de virfuri." << endl
					<< "2. Algoritmul Floyd." << endl
					<< "3. Algoritmul Dijkstra." << endl
					<< "0. Meniul principal." << endl;
				cout << endl << ">>> ";
				cin >> ChooseMenu;
				system("cls");
```

```
                        switch (ChooseMenu) {
                        case 1: {
                                nr_virf_favorabil();
                                costurile_favorabil();
                                break;
                        }
                        case 2: {
                                t1 = clock();
                                FLOYD();
                                t2 = clock();
                                cout << "Timpul de lucru al algoritmului : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                                break;
                        }
                        case 3: {
                                t1 = clock();
                                DIJKSTRA();
                                t2 = clock();
                                cout << "Timpul de lucru al algoritmului : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                                break;
                        }
                        case 0: {
                                RESET();
                                goto x;
                        }
                        default: {
                                cout << "Optiune gresita" << endl;
                                break;
                        }
                        }
                        system("pause");
                        system("cls");
                }
        case 2: {
                while (true) {
                        system("cls");
                        cout << "1. Introduceti numarul de virfuri." << endl
                                << "2. Algoritmul Floyd." << endl
                                << "3. Algoritmul Dijkstra." << endl
                                << "0. Meniul principal." << endl;
                        cout << endl << ">>> ";
                        cin >> ChooseMenu;
                        system("cls");
                        switch (ChooseMenu) {
                        case 1: {
                                nr_virf_mediu();
                                costurile_mediu();
                                break;
                        }
                        case 2: {
```

```
                        t1 = clock();
                        FLOYD();
                        t2 = clock();
                        cout << "Timpul de lucru al algoritmului : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                        break;
                }
                case 3: {
                        t1 = clock();
                        DIJKSTRA();
                        t2 = clock();
                        cout << "Timpul de lucru al algoritmului : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                        break;
                }
                case 0: {
                        RESET();
                        goto x;
                }
                default: {
                        cout << "Optiune gresita" << endl;
                        break;
                }
                }
                system("pause");
                system("cls");
        }
    case 3: {
        while (true) {
                system("cls");
                cout << "1. Introduceti numarul de virfuri." << endl
                        << "2. Algoritmul Floyd." << endl
                        << "3. Algoritmul Dijkstra." << endl
                        << "0. Meniul principal." << endl;
                cout << endl << ">>> ";
                cin >> ChooseMenu;
                system("cls");
                switch (ChooseMenu) {
                case 1: {
                        nr_virf_defavorabil();
                        costurile_defavorabil();
                        break;
                }
                case 2: {
                        t1 = clock();
                        FLOYD();
                        t2 = clock();
                        cout << "Timpul de lucru al algoritmului : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                        break;
                }
```

```
                        case 3: {
                                t1 = clock();
                                DIJKSTRA();
                                t2 = clock();
                                cout << "Timpul de lucru al algoritmului : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                                break;
                        }
                        case 0: {
                                RESET();
                                goto x;
                        }
                        default: {
                                cout << "Optiune gresita" << endl;
                                break;
                        }
                        }
                        system("pause");
                        system("cls");
                }
        case 0: {
                return 0;
        }
        default: {
                cout << "Optiune gresita" << endl;
                break;
        }}}}}
        system("pause");
        system("cls");}}
```

**Rezultatele obtinute**

**Cazul favorabil**

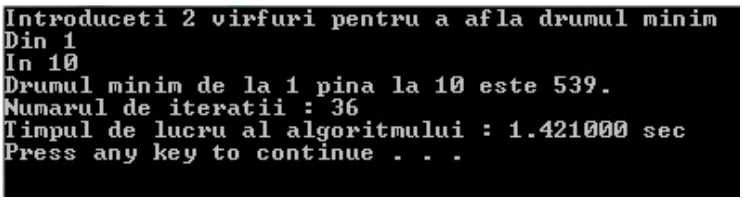



***Fig 1.*** Alg.Floyd 10 noduri

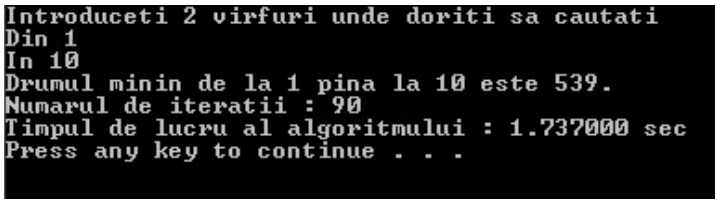



***Fig 2.*** Alg.Dijkstra 10 noduri

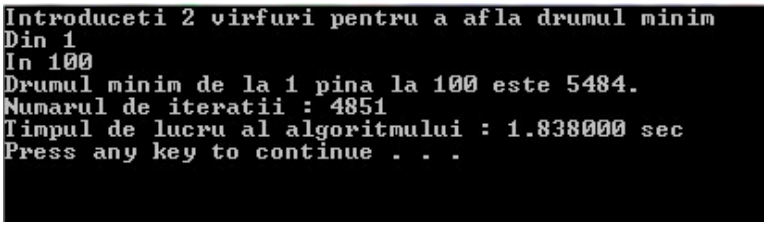



***Fig 3.*** Alg.Floyd 100 noduri

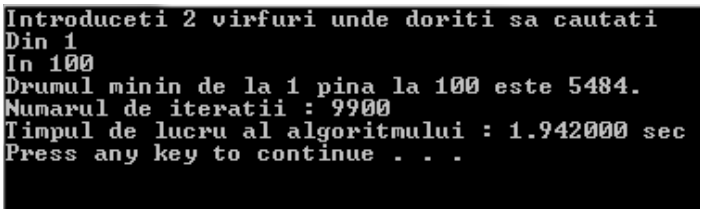



***Fig 4.*** Alg.Dijkstra 100 noduri

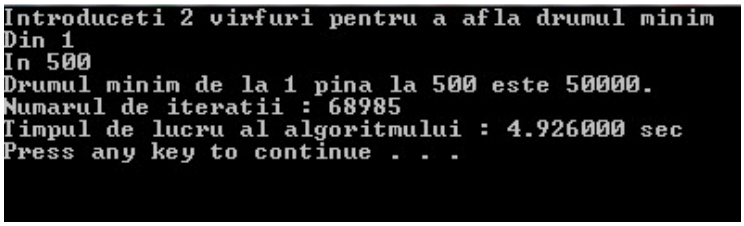


***Fig 5.*** Alg.Floyd 500 noduri

**Cazul defavorabil**

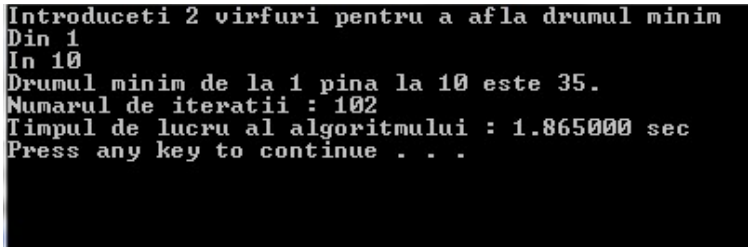



***Fig 6.*** Alg.Floyd 10 noduri

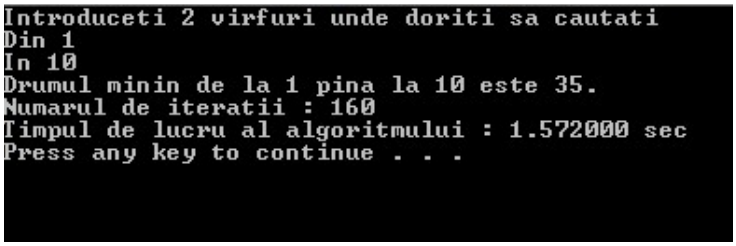



***Fig 7.*** Alg.Dijkstra 10 noduri

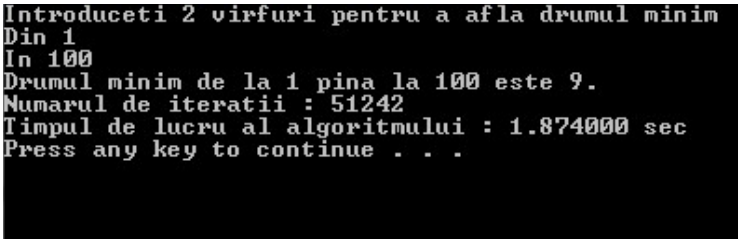



***Fig 8.*** Alg.Floyd 100 noduri

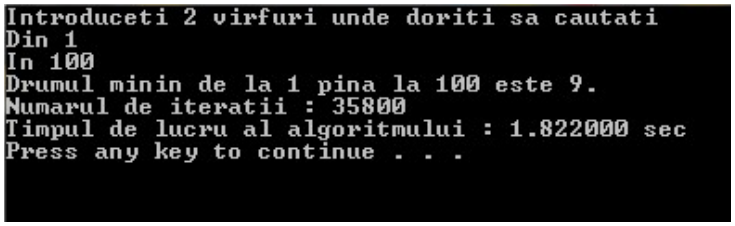



***Fig 9.*** Alg.Dijkstra 100 noduri

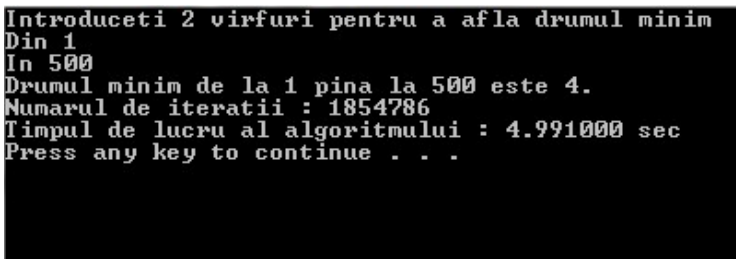



***Fig 10.*** Alg.Floyd 500 noduri

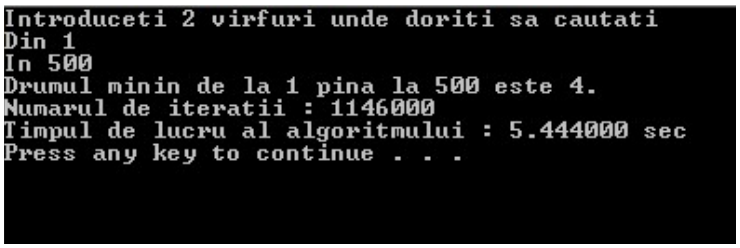



***Fig 11.*** Alg.Dijkstra 500 noduri

**Cazul mediu**

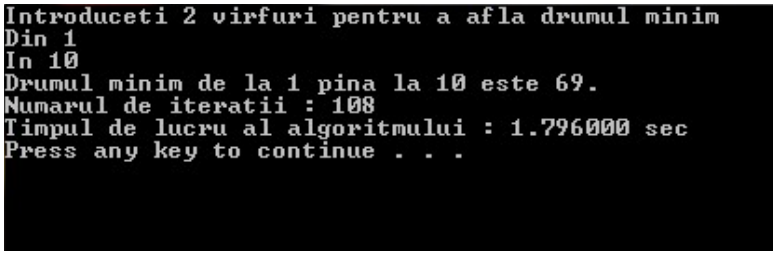



***Fig 12.*** Alg.Floyd 10 noduri

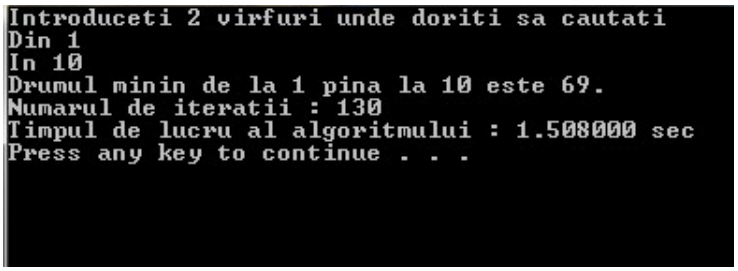



***Fig 13.*** Alg.Dijkstra 10 noduri

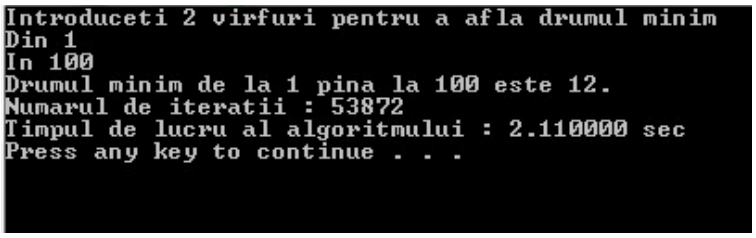



***Fig 14.*** Alg.Floyd 100 noduri

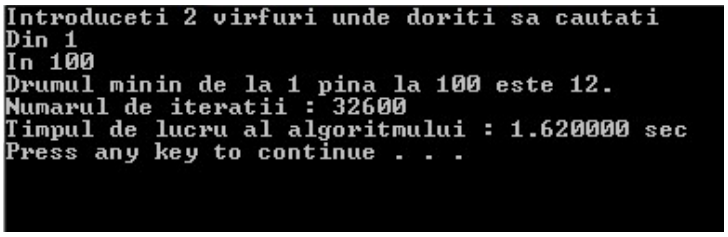



***Fig 15.*** Alg.Dijkstra 100 noduri

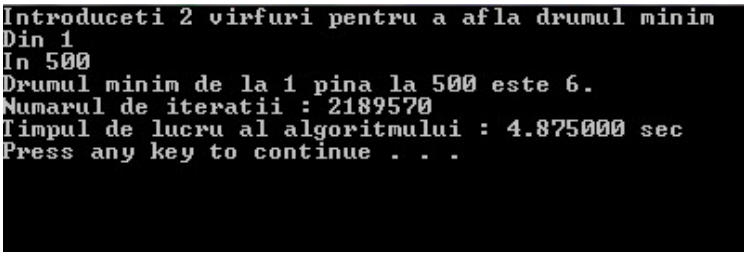



***Fig 16.*** Alg.Floyd 500 noduri

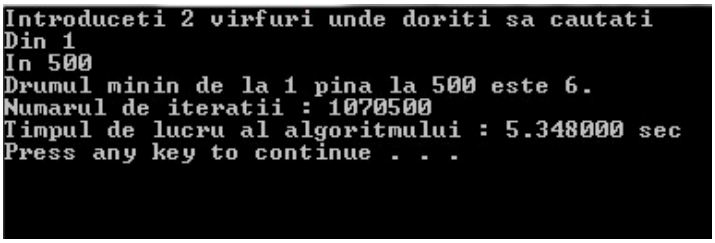



***Fig 17.*** Alg.Dijkstra 500 noduri

**Cazul Favorabil**

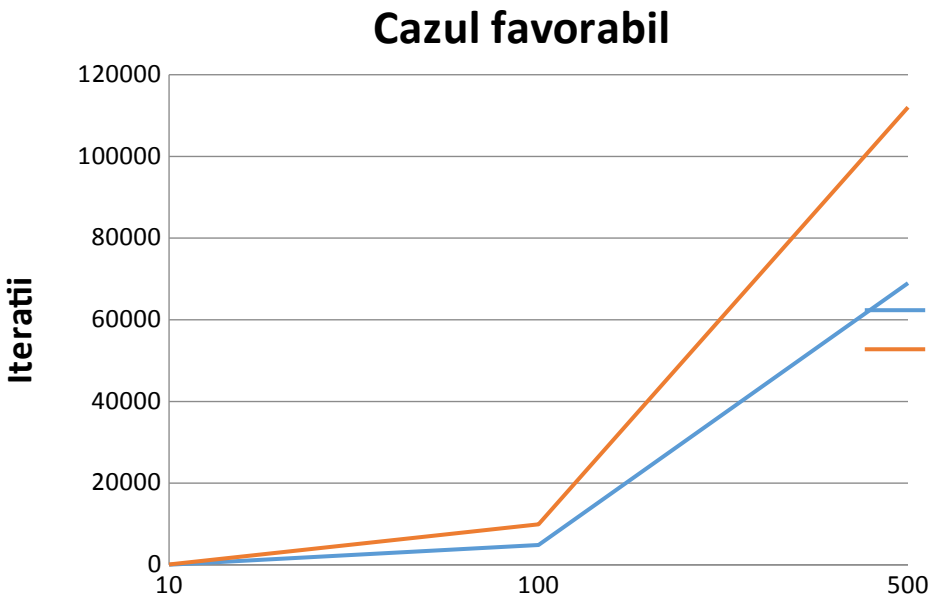
| Algoritmul | Nr.de virfuri | Iteratii |
|---|---|---|
| Floyd | 500 | 68985 |
| Floyd | 100 | 4851 |
| Floyd | 10 | 36 |
| Dijkstra | 500 | 112032 |
| Dijkstra | 100 | 9900 |
| Dijkstra | 10 | 90 |

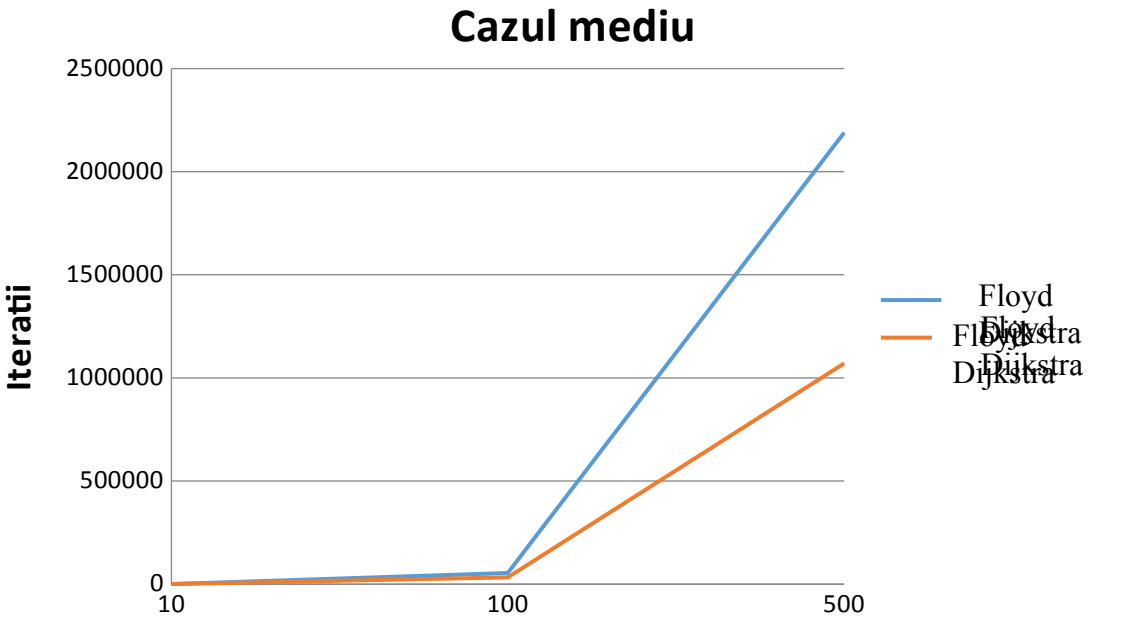
**Cazul mediu**

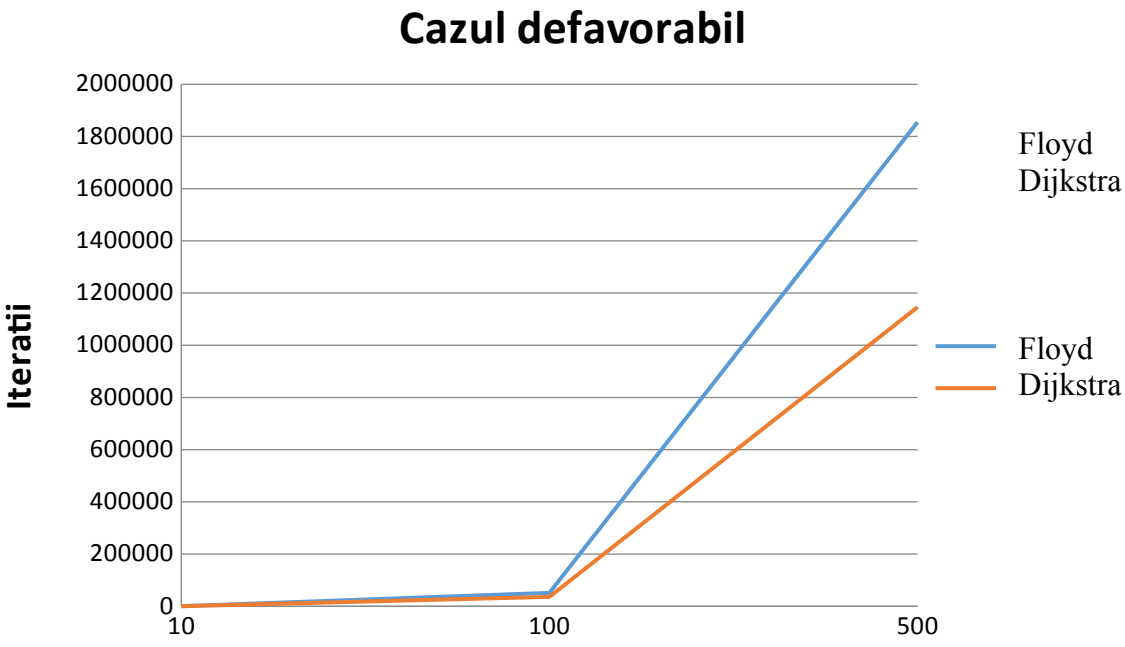
| Algoritmul | Nr.de virfuri | Iteratii |
|---|---|---|
| Floyd | 500 | 2189570 |
| Floyd | 100 | 53872 |
| Floyd | 10 | 108 |
| Dijkstra | 500 | 1070500 |
| Dijkstra | 100 | 32600 |
| Dijkstra | 10 | 130 |

**Caz defavorabil**

| Algoritmul | Nr.de virfuri | Iteratii |
|---|---|---|
| Floyd | 500 | 1854786 |
| Floyd | 100 | 51242 |
| Floyd | 10 | 102 |
| Dijkstra | 500 | 1146000 |
| Dijkstra | 100 | 35800 |
| Dijkstra | 10 | 160 |




Cazul mediu
Iteratii
2500000
2000000
1500000
1000000
500000
0
10
100
500
Floyd
Dijkstra


Cazul defavorabil
Iteratii
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
10
100
500
Floyd
Dijkstra
Floyd
Dijkstra

**Concluzie**

În urma elaborării lucrării de laborator  am facut cunostinta cu metoda programarii dinamice, care reprezinta o determinare a solutiei prin etape. Am realizat analiza empirica a algoritmilor de determinare a celor mai scurte drumuri intr-un graf si anume algoritmii Dijkstra si Floyd. Deasemenea algoritmul Floyd este mai simplu de implementat intr-un limbaj de programare, constituind  3 cicluri . In urma analizei s-a dovedit ca algoritmul Dijkstra ar fi cu foarte putin mai eficient ca Floyd, insa practica ne arata ca algoritmul Floyd este cu mult mai eficient decit algoritmul Dijkstra in cazul grafurilor orientate dense.