Ministerul Educaţiei, Culturii şi Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică şi Microelectronică
Departamentul Ingineria Software şi Automatică

Disciplina: Analiza și Proiectarea Algoritmilor

Lucrarea de laborator nr. 4

Tema: Algoritmii Dijkstra și Floyd

A efectuat student: **Borta Sergiu, gr. TI-183**

A controlat: **Buldumac Oleg, l. univ**

Chişinău 2019

Scopul lucrării: Studierea programării dinamice, implementarea algoritmilor Dijkstra şi Floyd în limbajul de programare C++.

Consideraţii teoretice:

Dezvoltarea unui algoritm bazat pe programarea dinamică poate fi împărţit 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.

Algoritmul Dijkstra:

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

Algoritmul Floyd:

```
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
```

Codul programului în C++:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define max 1000
#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];
      }
   }
}

void nr_virf_defavorabil() {
cout << "Nr. de varfuri: ";
cin >> 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() {
   for (int i = 0; i<n; i++) {
      for (int j = 0; j<n; j++) {
         if (MS[i][j] && i != j) {
            MV[i][j] = MS[i][j];}
```

```
            else
                MV[i][j] = INF;}
    }
    RESET();
}
void nr_virf_favorabil() {
    cout << "Nr. de varfuri: ";
    cin >> n;
    for (int i = 0; i<n; i++) {
        for (int j = 0; j<n; j++)
            MS[i][i + 1] = rand() % 1000;
    }
}
void costurile_favorabil() {
for (int i = 0; i<n; i++) {
    for (int j = 0; j<n; j++) {
        if (MS[i][j] && i != j) {
                MV[i][j] = MS[i][j];}
            else
                MV[i][j] = INF;
        }
    }
    RESET();}

void nr_virf_mediu() {
    cout << "Nr. de virfuri: ";
    cin >> 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() {
   for (int i = 0; i<n; i++) {
      for (int j = 0; j<n; j++) {
         if (MS[i][j] && i != j) {
            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;
   }}}}}}}
```

Rezultate:

Cazul favorabil:

Algoritm	Vârfuri	Iterații	Timp (s)
Dijkstra	10	1080	0.000283
	100	1009800	0.012973
	1000	997999002	10.498716
Floyd	10	1036	0.000187
	100	1004851	0.007878
	1000	1000243044	5.069551

Cazul mediu:

Algoritm	Vârfuri	Iterații	Timp (s)
Dijkstra	10	1220	0.000207
	100	1071500	0.014247
	1000	104594000	10.54811
Floyd	10	1135	0.000210
	100	1059931	0.008373
	1000	1014553208	5.273223

Cazul defavorabil:

Algoritm	Vârfuri	Iterații	Timp (s)
Dijkstra	10	1280	0.000302
	100	1084500	0.013798
	1000	1071942000	10.348713
Floyd	10	1096	0.000162
	100	1061974	0.008597
	1000	1013734576	5.315031

Caz favorabil – Grafice:

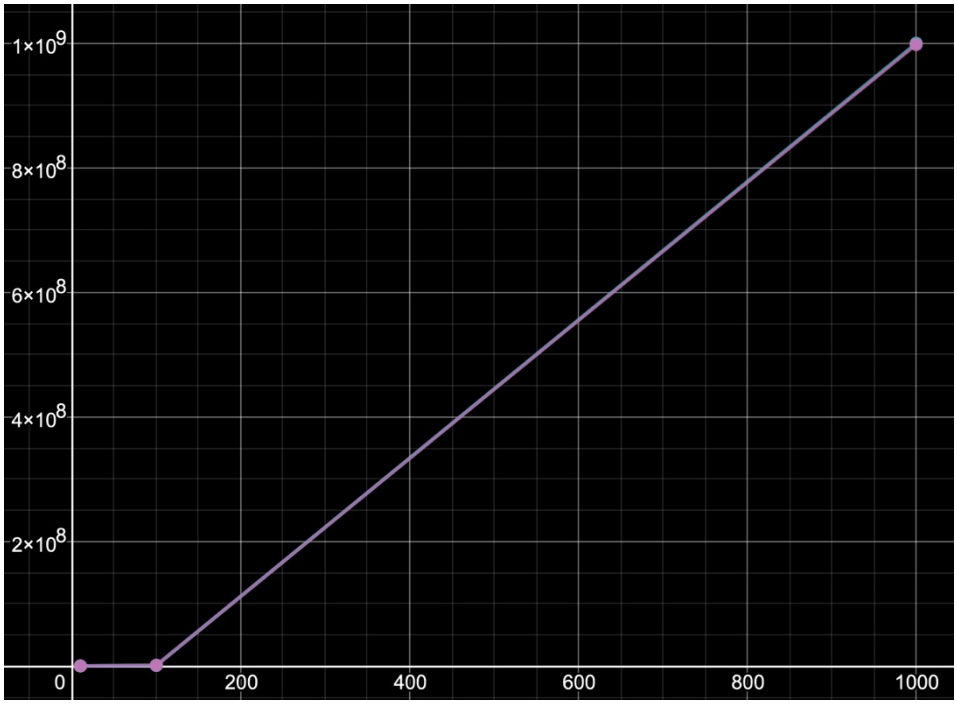

Fig. 1.1 – Nr. de vârfuri în raport cu iterațiile

Violet: **Dijkstra**; Albastru: **Floyd**.

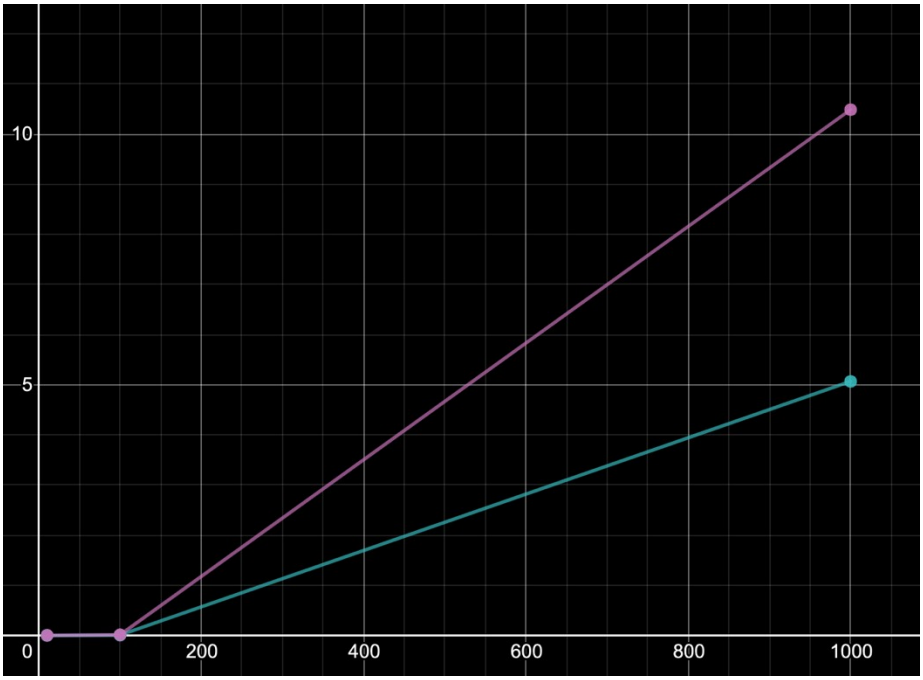

Fig. 1.2 – Nr. de vârfuri în raport cu timpul

Violet: **Dijkstra**; Albastru: **Floyd**.

Caz mediu – Grafice:

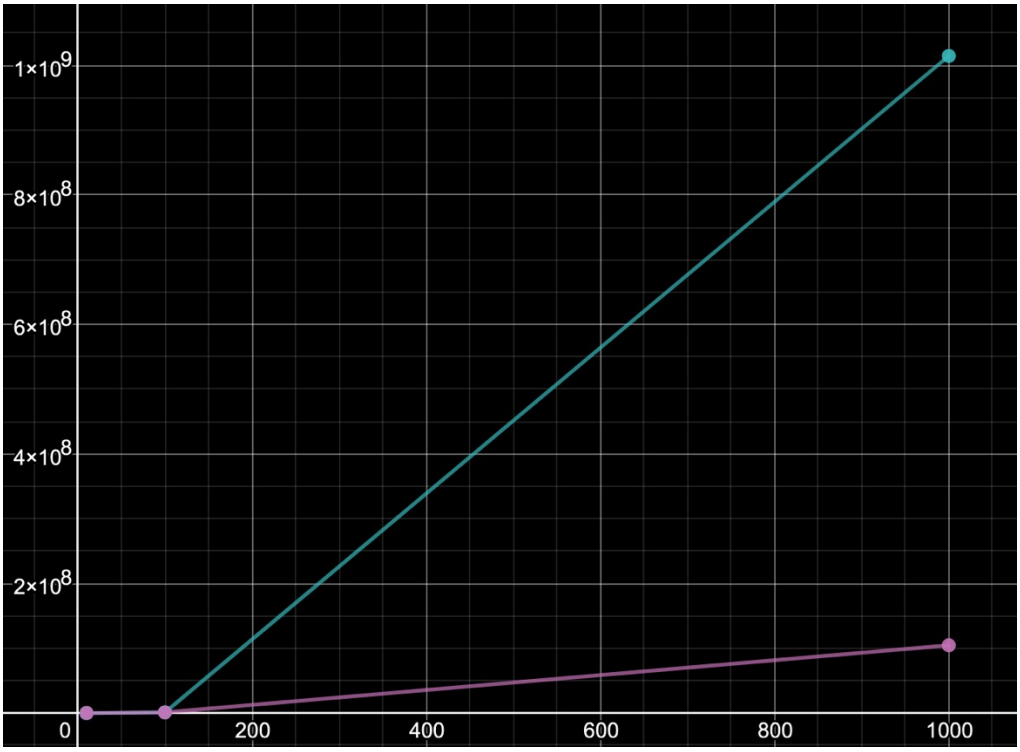

Fig. 2.1 – Nr. de vârfuri în raport cu iterațiile

Violet: **Dijkstra**; Albastru: **Floyd**.

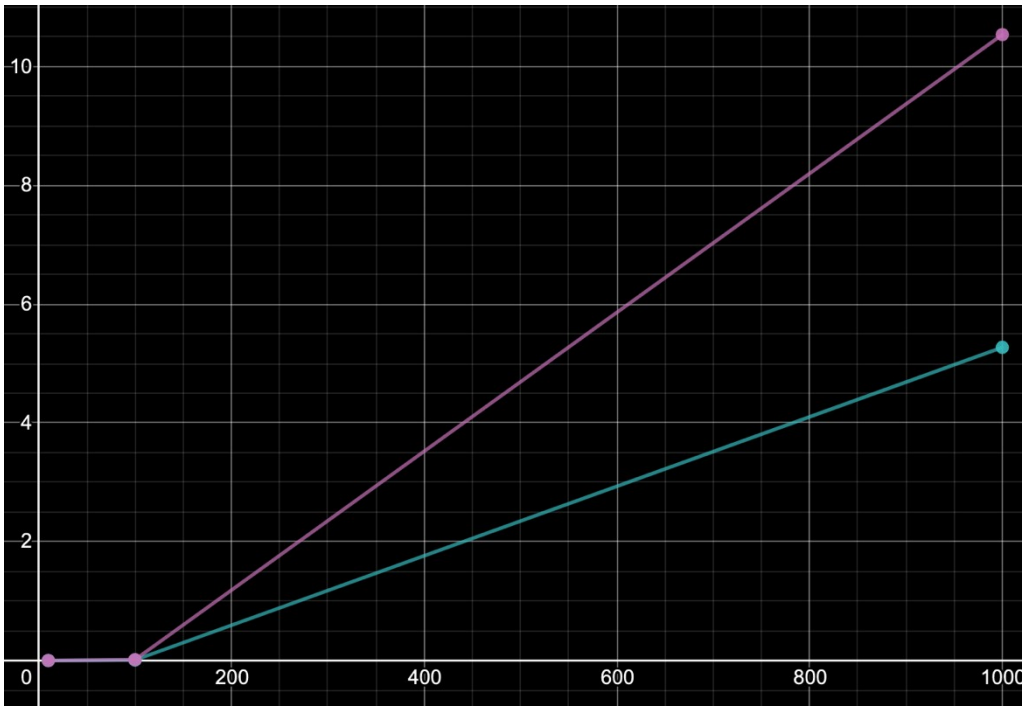

Fig. 2.2 – Nr. de vârfuri în raport cu timpul

Violet: **Dijkstra**; Albastru: **Floyd**.

Caz defavorabil – Grafice:

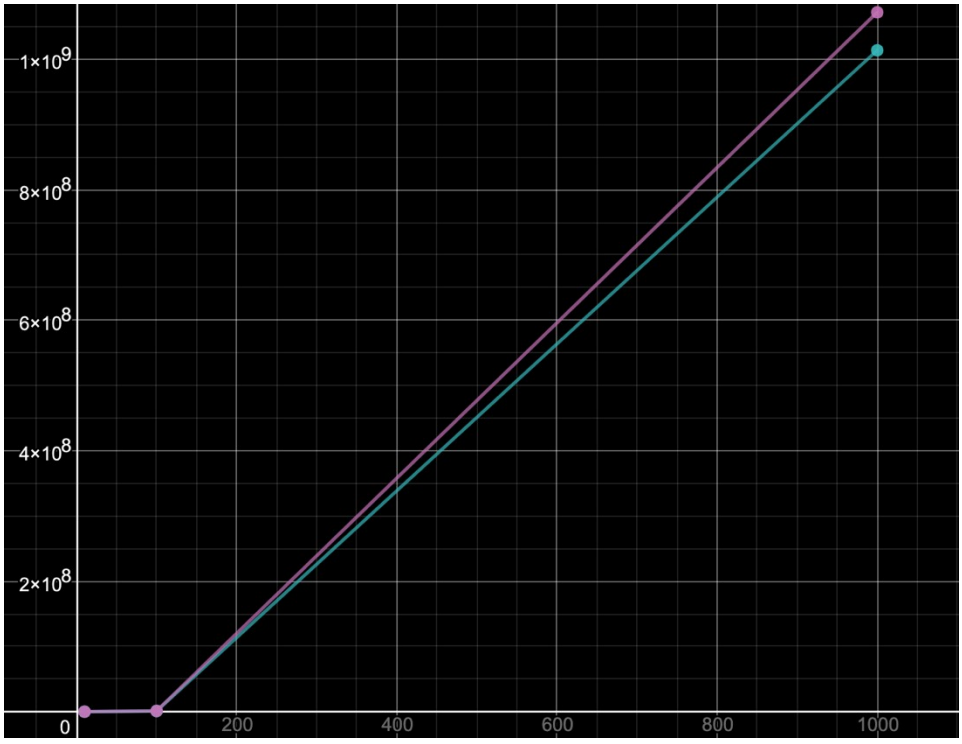

Fig. 3.1 - Nr. de vârfuri în raport cu iterațiile

Violet: **Dijkstra**; Albastru: **Floyd**.

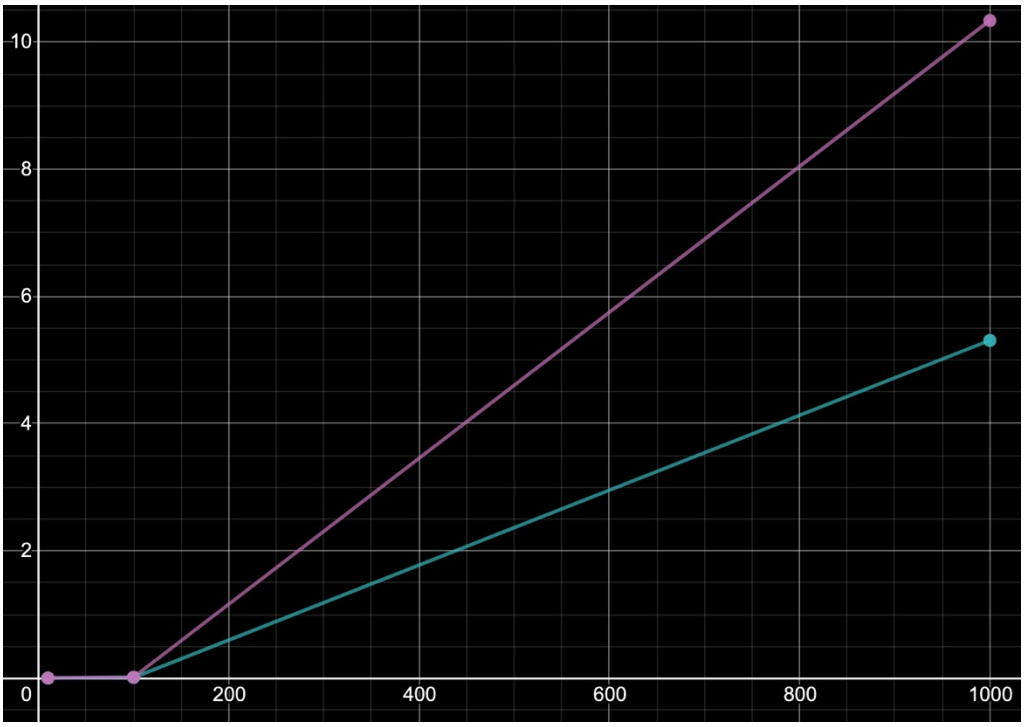

Fig. 3.2 - Nr. de vârfuri în raport cu timpul

Concluzie: În această lucrare de laborator am lucrat cu algoritmii Dijkstra și Floyd și i-am implementat în limbajul de programare C++. În practică, algoritmul Floyd se dovedește a fi mai eficient, deoarece se implementează mai ușor în programare și folosesc mai puține cicluri. De asemenea el este mai eficient din punct de vedere al timpului și al iterațiilor. Se poate deduce că algoritmul lui Floyd necesită un timp în $O(n^3)$. Acesta, datorită simplităţii lui, are constanta multiplicativă mai mică, de aia fiind probabil mai rapid în practică. Am făcut analiza empirică a acestor algoritmi pentru un graf rar şi pentru un graf dens și am determinat comportamentul acestora în ambele cazuri.