Analiza Algoritmilor
Cel mai scurt drum (2)

Alexandru Licuriceanu
alicuriceanu@stud.acs.upb.ro

Universitatea POLITEHNICA din Bucureşti
Facultatea de Automatică şi Calculatoare

Abstract. Această temă presupune analiza din punct de vedere al complexităţii a trei algoritmi de căutare al costului minim între oricare două noduri ale unui graf ponderat.

1 Introducere

1.1 Descrierea problemei

În teoria grafurilor, un drum într-un graf este un şir de muchii prin care se unesc o succesiune de noduri, în majoritatea cazurilor distincte. Un graf ponderat se defineşte ca fiind un graf în care muchiilor li se atribuie o pondere, numită şi cost.

Problema constă în determinarea drumului cu costul minim între oricare două noduri dintr-un graf, sau cu alte cuvinte, găsirea drumului dintre oricare două noduri astfel încât suma costurilor muchiilor parcurse să fie minimă.

În cadrul acestei teme, am analizat trei algoritmi de căutare al celui mai scurt drum. De reţinut este faptul că pentru această problemă, particularităţile grafurilor precum pozitivitatea costului unei muchii schimbă soluţia de rezolvare.

1.2 Aplicaţii practice

Algoritmii de căutare al celui mai scurt drum dintr-un graf au suficiente aplicaţii reale, iar un exemplu bun de utilizare sunt aplicaţiile care lucrează cu hărţi digitale, cum ar fi Google Maps sau Waze.

De exemplu, putem considera o hartă ca fiind un graf, unde nodurile grafului sunt locuri de pe hartă, muchiile sunt drumurile care le leagă pe acestea, iar costul fiecărei muchii reprezintă distanţa dintre două locuri. Dacă alegem oricare două locuri şi vrem să aflăm distanţa dintre ele, se poate folosi unul dintre algoritmii menţionaţi.

1.3 Soluţiile alese pentru rezolvarea problemei

În continuare, am să prezint succint ideea din spatele fiecărui algoritm ales. De precizat este că niciunul dintre algoritmi nu funcţionează corect pentru un graf cu cicluri cu cost negativ.

Algoritmul Dijkstra [1]

Iniţial, Dijkstra atribuie fiecărui nod, în afară de cel sursă, valoarea infinit, pe care o voi numi estimare. Această valoare reprezintă costul celui mai scurt drum din nodul de origine până la nodul respectiv. De asemenea, toate nodurile în afară de origine sunt marcate ca fiind nevizitate. Dupa aceşti paşi, se repetă următoarele:

– Pentru fiecare vecin nevizitat al nodului curent, se calculează costul drumului. Dacă valoarea aceasta este mai mică decât estimarea nodului vecin, estimarea este actualizată cu valoarea calculată anterior.

– Nodul curent devine nodul cu estimarea minimă şi este marcat ca fiind vizitat.

De menţionat este faptul că Dijkstra nu poate fi aplicat pe grafuri care au muchii cu cost negativ, iar algoritmul calculează doar costul minim de la un nod la toate celelalte, nu costurile drumurilor dintre toate perechile de noduri. Această problemă se rezolvă rulând algoritmul pentru fiecare nod, însă cu o creştere în complexitate.

Algoritmul Bellman-Ford [2]

Spre deosebire de algoritmul Dijkstra, Bellman-Ford poate fi aplicat şi pe grafuri cu muchii ponderate negativ.

În primul rând, pentru fiecare nod este setat costul drumului din sursă până la acesta cu infinit. Costul de a ajunge din nodul sursă în nodul sursă este setat la 0. Algoritmul execută următorii paşi de $|Noduri|$ - 1 ori:

– Iterează toate nodurile şi calculeaza pentru fiecare, costul drumului pentru a ajunge la el.
– Dacă noul cost este mai mic decât cel vechi, valoarea costului drumului este actualizată.

La fel ca la Dijkstra, problema este că se calculează doar costul minim al drumului de la un nod la restul nodurilor, însă se rezolvă asemănător, rulând algoritmul pentru fiecare nod.

Algoritmul Floyd-Warshall [3]

Algoritmul Floyd-Warshall a fost conceput exact pentru a rezolva problema costului minim între oricare două noduri. Rezultatul algoritmului este o matrice $N \times N$ unde N este numărul de noduri, iar valoarea (X, Y) din matrice va fi costul minim pentru drumul de la nodul X la nodul Y.

Implementarea algoritmului constă în trei for-uri interioare ("nested") care ciclează de la 1 la N şi verifică pentru fiecare pereche de noduri dacă trecând printr-un nod intermediar K scade costul drumului pentru perechea de noduri iniţială.

1.4 Evaluarea soluţiilor

Din punct de vedere al corectitudinii rezultatelor, sunt două cazuri când nu se calculează corect costul minim:

- Când Dijkstra primeste un graf cu muchii cu cost negativ.
- Când exista cicluri negative în graf. (Valabil pentru toţi algoritmii)

O evaluare riguroasă trebuie facută pe seturi de date cât mai variate. Am pornit de la grafuri neorientate, cu puţine noduri, puţine muchii şi costuri mici, crescând treptat dimensiunile datelor de intrare. În acelaşi mod am generat şi teste pentru grafuri orientate.

Valorile din seturile de test se generează conform următoarelor constrângeri:

$$1 \leq Noduri \leq 10^4$$

$$1 \leq Muchii \leq min(Noduri^2, 10^6)$$

$$1 \leq X, Y \leq Noduri$$

$$-10^6 \leq C \leq 10^6$$

Unde tripletul (X, Y, C) semnifică faptul că există o muchie de la nodul X la nodul Y care are costul C.

Pentru a evalua performanţele algoritmilor pe seturi de test variate, am generat fişiere de test care sunt structurate conform tabelului 1. De asemenea, am generat două teste care au între 1000 şi 1500 de noduri, cu costuri cuprinse între 10^5 şi 10^6 şi două teste care au tot între 1000 şi 1500 de noduri, dar cu costuri între 0 şi 10^2.

2 Prezentarea soluţiilor

În această secţiune a documentului voi prezenta mai detaliat modul în care funcţionează cei trei algoritmi şi de asemenea o analiză a complexităţii şi principalele avantaje şi dezavantaje ale soluţiilor.

2.1 Modul de funcţionare al algoritmilor şi analiza complexităţii

După cum am precizat, algoritmii Dijkstra şi Bellman-Ford au fost proiectaţi pentru a găsi cel mai scurt drum de la un singur nod la restul, aşadar, aceştia trebuie rulaţi pentru fiecare dintre nodurile grafului. Pe de altă parte, algoritmul Floyd-Warshall a fost conceput pentru a calcula costul minim pentru toate perechile de noduri posibile din graf.

Algoritmul Dijkstra

Algorithm 1 Dijkstra, all to all version

```
1: for each vertex S in graph G do
2:     for each vertex V in G do
3:         distance[V] ← infinity
4:         queue ← V
5:     end for
6:     distance[S] ← 0
7:     while queue is not empty do
8:         U ← min(distance[U])
9:         dequeue(U)
10:        for each neighbor V of U do
11:            alt ← distance[U] + distance(U, V)
12:            if alt < distance[V] then
13:                distance[V] ← alt
14:            end if
15:        end for
16:    end while
17: end for
```

În primul rând, iniţializăm distanţele de la nodul curent la toate celelalte din graf cu infinit. Distanţa de la nodul sursă la el însuşi este 0.
Toate nodurile din lista de noduri vizitate sunt setate cu fals. Iterăm toate nodurile adiacente nodului curent şi îl aleg pe cel care are muchia cu costul minim. Dacă noul cost calculat pentru a ajunge în nodul respectiv este mai mic decât cel calculat anterior, acesta este actualizat. Nodul curent devine nodul care este conectat cu muchia cu cost minim şi procesul se repetă până când se vizitează toate nodurile.

Complexitate
Pentru a găsi nodul adiacent celui curent care are drumul cu cel mai mic cost, trebuie să iterăm toate nodurile adiacente, care se face în $O(N)$, unde N este numărul nodurilor adiacente. În cel mai rău caz, un nod poate avea toate nodurile grafului adiacente.

Pentru fiecare nod adiacent, trebuie verificat şi actualizat costul drumului până la acesta, operaţie care este de ordinul lui $O(1) \cdot O(N)$ unde N este numărul de noduri adiacente. Tot procesul trebuie aplicat tuturor nodurilor, de unde rezultă complexitatea $O(N)$ unde N este numărul de noduri ale grafului.

Complexitatea finală a algoritmului Dijkstra pentru toate perechile de noduri este de $O(N) \cdot O(N) \cdot O(1) \cdot O(N) = O(N^3)$ unde N este numărul de noduri pe care le are graful.

Algoritmul Bellman-Ford

Algorithm 2 Bellman-Ford, all to all version

```
1: for each vertex S in graph G do
2:     for each vertex V in G do
3:         distance[V] ← infinity
4:     end for
5:     distance[S] ← 0
6:     for i = 1, 2, ..., G.vertex_count − 1 do
7:         for each vertex V in G do
8:             for each vertex U in V.neighbors do
9:                 if distance[V] + distance(V, U) < distance[U] then
10:                    distance[U] = distance[V] + distance(V, U)
11:                end if
12:            end for
13:        end for
14:    end for
15: end for
```

La primul pas, iniţializăm distanţele de la nodul sursă la restul cu infinit, iar distanţa de la nodul sursă la el însuşi cu 0.
Algoritmul face $Noduri - 1$ iteraţii, iar la fiecare, iterează toate nodurile grafului şi verifică dacă costul minim al drumului de la sursă prin nodul curent către fiecare dintre nodurile adiacente este mai mic decât cel calculat anterior, actualizându-l dacă este cazul.

Complexitate
Algoritmul Bellman-Ford se bazează pe faptul că cel mai lung drum intr-un graf fără cicluri are $Noduri - 1$ muchii. Aşadar, se fac $Noduri - 1$ executări de unde reiese o complexitate de $O(N-1) = O(N)$ unde N este numărul de noduri.

Apoi, toate muchiile nodului curent sunt "relaxate" operaţiune care are complexitatea $O(M)$ unde M este numărul de muchii conectate la nodul curent. De asemenea, aplicăm Bellman-Ford pentru toate perechile de noduri, de unde reiese încă o complexitate de $O(N)$ cu N fiind numărul de noduri ale grafului.

Complexitatea finală a algoritmului Bellman-Ford pentru toate perechile de noduri este de $O(N) \cdot O(M) \cdot O(N) = O(M \cdot N^2)$ unde N este numărul de noduri pe care le are graful, iar M este numărul de muchii ale grafului. În cel mai rău caz, pe un graf complet sau când M se apropie de V^2, complexitatea devine $O(N^4)$ pentru găsirea costurilor minime tuturor perechilor de noduri.

Algoritmul Floyd-Warshall

Algorithm 3 Floyd-Warshall

```
1: for i = 1, 2, ..., G.vertex_count do
2:     distance[i][j] ← infinity
3:     if i = j then
4:         distance[i][j] ← 0
5:     end if
6: end for
7: for k = 1, 2, ..., G.vertex_count do
8:     for i = 1, 2, ..., G.vertex_count do
9:         for j = 1, 2, ..., G.vertex_count do
10:            if distance[i][j] > distance[i][k] + distance[k][j] then
11:                distance[i][j] ← distance[i][k] + distance[k][j]
12:            end if
13:        end for
14:    end for
15: end for
```

Algoritmul Floyd-Warshall compară toate căile posibile dintre oricare două perechi de noduri din graf. Algoritmul calculează cel mai scurt drum utilizând toate nodurile ca intermediare, unul câte unul şi actualizează costul drumului pentru perechea de noduri curentă dacă a fost găsit un drum cu un cost mai mic.

Complexitate
Complexitatea algoritmului Floyd-Warshall se calculează uşor, avem trei bucle for, fiecare mergând până la numărul de noduri din graf, de unde rezultă complexitatea finală de $O(N) \cdot O(N) \cdot O(N) = O(N^3)$ cu N fiind numărul de noduri ale grafului.

2.2 Principalele avantaje şi dezavantaje ale algoritmilor

Avantaje

- Dijkstra: Este cel mai rapid dintre toţi trei algoritmii, mai ales dacă este implementat cu o coadă de priorităţi minime.

- Bellman-Ford: Funcţionează bine pe un graf care are muchii cu greutăţi negative şi poate detecta ciculuri negative.

- Floyd-Warshall: Găseşte din start drumurile cu cost minim pentru toate perechile de noduri şi este uşor de scris şi reţinut codul pentru acest algoritm.

Dezavantaje

- Dijkstra: Marele dezavantaj al algoritmului Dijkstra este că nu funcţionează pe grafuri care au muchii cu cost negativ.
- Bellman-Ford: Nu funcţionează corect pentru grafuri care conţin cicluri cu cost negativ.
- Floyd-Warshall: În majoritatea cazurilor, acesta este cel mai lent algoritm dintre toţi trei.

3 Evaluare

3.1 Construirea seturilor de teste

Pentru teste am creat un program care îmi permite să generez grafuri cu anumite proprietăti. Pot să specific proprietăţi cum ar fi numărul minim sau maxim de noduri, intervalul în care se află costurile sau dacă este un graf orientat sau neorientat. Am încercat sa fac astfel încât să existe o variaţie mare între datele de intrare.

Atât pentru grafuri orientate, cât şi pentru cele neorientate, testele au avut în esenţă aceeasi structură. Primele cinci teste au numărul de noduri cuprins între 1 şi 10 cu costuri între 0 şi 100. Testele de la şase la zece au între 10 şi 100 de noduri, cu costuri cuprinse între 100 şi 1000. Următoarele zece seturi au între 100 şi 500 de noduri şi cu costuri între 10^3 şi 10^4 pentru a putea observa cum se comportă fiecare algoritm cu date mari de intrare. De asemenea am generat patru teste cu numărul de noduri cuprins între 1000 şi 1500, însă două dintre ele au costuri mici, între 0 şi 100, iar două au costuri mai mari, între 10^5 şi 10^6.

Organizarea fişierelor de test este detaliată în tabelul 1

Tabel 1: Intervale de valori pentru fiecare test.

Grafuri orientate			Grafuri neorientate		
Număr test	Număr noduri	Costuri	Număr test	Număr noduri	Costuri
1-5	1-10	0-10^2	21-25	1-10	0-10^2
6-10	10-10^2	10^2-10^3	26-30	10-10^2	10^2-10^3
11-15	10^2-500	10^3-10^4	31-35	10^2-500	10^3-10^4
16-20	10^2-500	10^3-10^4	36-40	10^2-500	10^3-10^4
41	10^3-1500	10^4-10^5	42	10^3-1500	10^4-10^5
43	10^3-1500	0-10^2	44	10^3-1500	0-10^2

3.2 Specificaţiile sistemului de calcul pe care am rulat testele

Bineînţeles, timpul de execuţie al fiecărui program diferă de la un sistem la altul, astfel am rulat testele pe următorul sistem de calcul:

- Procesor: Intel Core i7-6700K 4.0GHz
- Memorie RAM: 32.0 GB DDR4
- Sistem de operare: Windows 10 x64 cu subsistem Ubuntu 20.04 LTS

3.3 Ilustrarea, folosind grafice şi tabele, a rezultatelor evaluării soluţiilor pe seturile de teste

Ca să măsor timpul de execuţie al algoritmilor, am construit un alt program care face acest lucru, folosind biblioteca std::chrono, măsurând în nanosecunde şi am trecut rezultatele finale în milisecunde. Datele despre timpii de execuţie sunt trecute în tabelele 2 şi 3

Tabel 2: Timpul de execuţie pentru grafurile orientate.

Număr Test	Timp (ms)		
	Dijkstra	Bellman-Ford	Floyd-Warshall
1	0.0113	0.0257	**0.011**
2	0.0094	0.0099	**0.0054**
3	0.0047	0.004	**0.0035**
4	0.0268	0.1039	**0.0185**
5	0.0134	0.0248	**0.0087**
6	**0.27**	55.68	3.18
7	1.27	24.41	**1.25**
8	0.54	2.26	**0.51**
9	0.059	0.444	**0.051**
10	**1.37**	49.46	1.52
11	**662.98**	117175.62	812.36
12	**537.13**	43790.47	639.53
13	**9.66**	927.34	12.05
14	**198.77**	8610.47	255.45
15	**293.54**	16399.75	365.08
16	**1014.04**	462912.16	1233.55
17	**15.18**	913.06	17.50
18	**158.96**	33644.86	205.02
19	**329.10**	7371.75	406.43
20	**66.07**	8001.32	77.12
41	**22995.47**	201147.55	25555.32
43	**12845.88**	978603.91	14894.75

Tabel 3: Timpul de execuţie pentru grafurile neorientate.

Număr Test	Timp (ms)		
	Dijkstra	Bellman-Ford	Floyd-Warshall
21	0.019	0.117	**0.016**
22	0.004	0.003	**0.002**
23	0.019	0.052	**0.015**
24	0.0038	0.0038	**0.0033**
25	0.025	0.18	**0.020**
26	**0.48**	20.76	0.50
27	0.20	1.71	**0.18**
28	0.59	5.05	**0.56**
29	**2.46**	84.43	2.84
30	**8.54**	294.37	9.75
30	**177.20**	17391.45	210.66
32	**741.52**	559032.67	875.76
33	**25.89**	998.53	33.58
34	**91.08**	8842.01	118.20
35	**542.14**	253303.24	664.96
36	**103.26**	6506.47	134.30
37	**252.89**	127695.16	320.54
38	**15.09**	301.61	18.69
39	**849.77**	494099.81	1015.09
40	**10.86**	1600.64	12.78
42	0.28	2406971.64	**0.02**
44	**15386.70**	34983799.22	19410.09

3.4 Prezentarea valorilor obţinute pe teste

În total, 44 de teste rulate, dintre care cele mai rapid executate au fost: 28 cu Dijkstra, 0 cu Bellman-Ford şi 16 cu Floyd-Warshall. Orientarea grafului nu pare să influenţeze timpul de execuţie.

Iniţial, toţi trei algoritmii au avut performanţe similare pe seturi de date mici, însă pe măsură ce au crescut valorile din fişierele de test, algoritmul Dijkstra a fost cel mai rapid.

Este de remarcat faptul că pentru grafuri care au numărul de muchii M apropiat de N^2 (N este numărul de noduri din graf), algoritmul Bellman-Ford începe să fie cu mult mai lent decât Dijkstra sau Floyd-Warshall. După cum am precizat anterior, în aceste cazuri cu grafuri complete sau aproape complete, Bellman-Ford are complexitatea $O(N^4)$, pe când celelalte două au $O(N^3)$, N fiind numărul de noduri din graf.

Timpul de rulare este reprezentat sub formă de grafice în figurile 1, 2 şi 3.

Fig. 1. Timpul de executare pentru algoritmul Dijkstra.

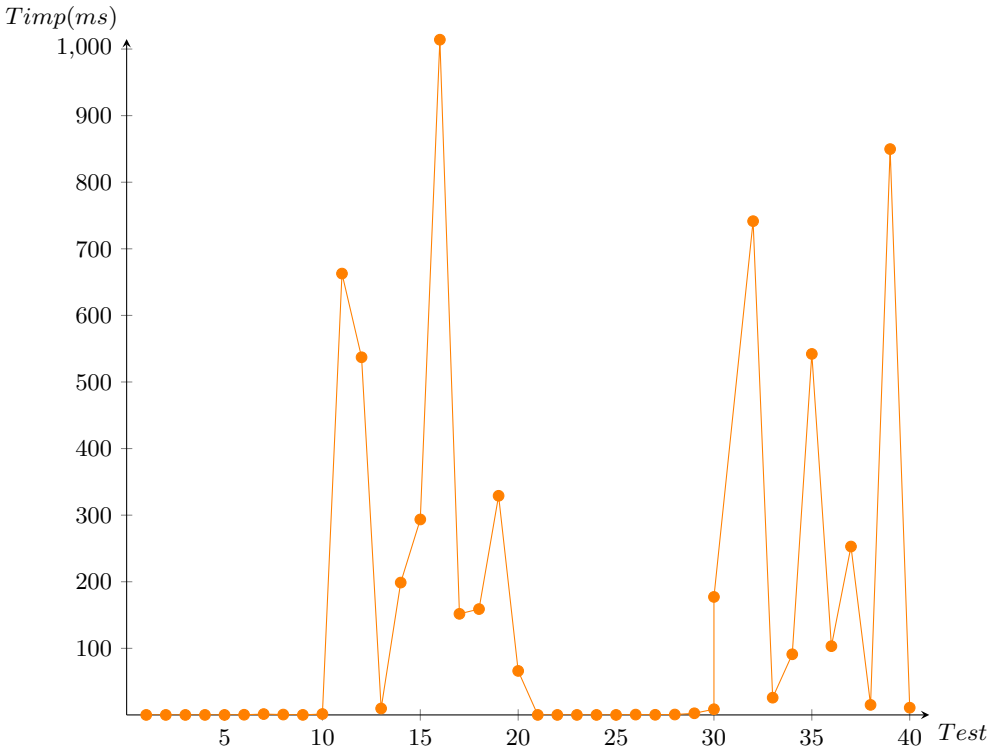


Fig. 2. Timpul de executare pentru algoritmul Bellman-Ford.

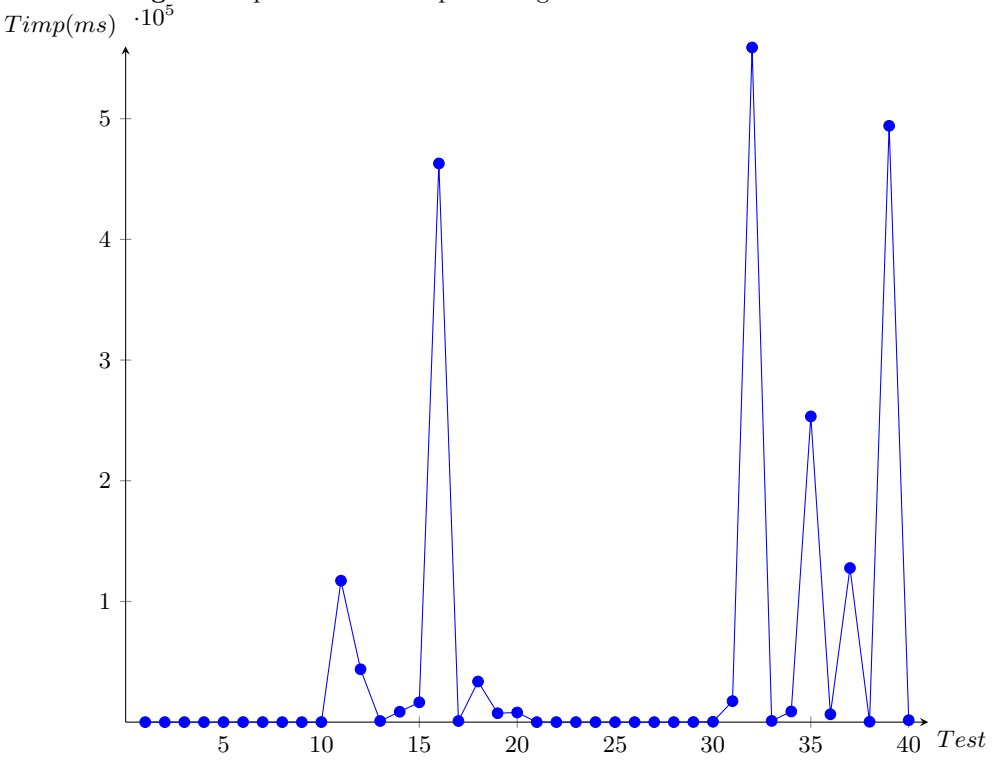
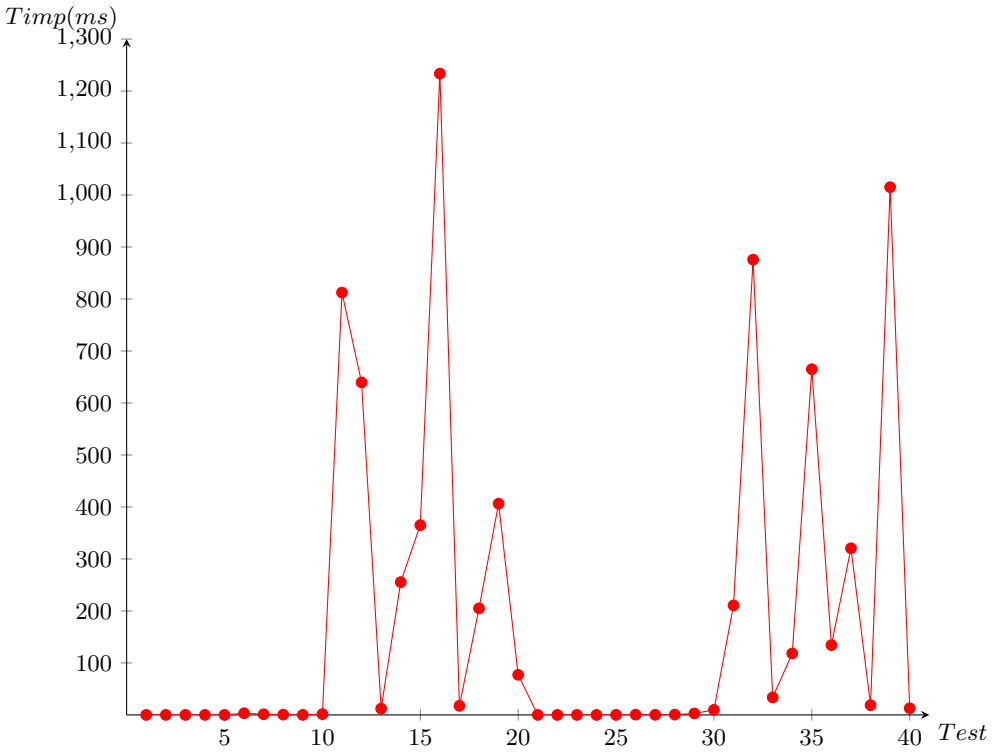


Fig. 3. Timpul de executare pentru algoritmul Floyd-Warshall.



4 Concluzii

Fiecare algoritm are avantajele şi dezavantajele sale, însă toate fac o treabă bună în a găsi cel mai scurt drum într-un graf, dar trebuie folosiţi în situaţii diferite.

În cele mai multe probleme din practică lucrezi cu grafuri care au costuri pozitive, cum ar fi aplicaţiile GPS, iar pentru asta aş alege algoritmul lui Dijkstra, datorită eficienţei acestuia. Dacă lucrez cu grafuri care au muchii ponderate negativ, aş opta pentru Floyd-Warshall deoarece funcţionează bine pe asemenea grafuri şi este uşor de ţinut minte.

Referinţe

1. Algoritmul Dijkstra, https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/.
   Ultima accesare 22 Dec 2022
2. Algoritmul Bellman-Ford, https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/.
   Ultima accesare 22 Dec 2022
3. Algoritmul Floyd-Warshall, https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/.
   Ultima accesare 22 Dec 2022