Drumuri minime în graf. Costul minim între oricare două noduri

Iordache Mădălina-Gabriela

Universitatea Politehnică din Bucureşti
Facultatea de Automatică şi Calculatoare
madaiorda@yahoo.com
Grupa: 323CA

Abstract. Această temă este dedicată găsirii costului minim în calea unui graf ponderat între nodul sursă și destinație. Au fost aleși trei algoritmi pentru a determina calea minimă: Dijkstra, Bellman-Ford și, nu în ultimul rând, Floyd-Warshall. În cele din urmă, va fi analizată performanța acestor algoritmi.

Keywords: Floyd-Warshall · Bellman-Ford · Dijkstra · graf ponderat · drum minim.

1 Introducere

1.1 Descrierea problemei rezolvate

În viața de zi cu zi există situații în care ne dorim să ajungem de la un punct la altul cât de repede putem, luând drumul cel mai scurt. În acest sens, există lucruri precum GPS-ul, dar cum funcționează?

În centrul GPS-ului, există algoritmi bazați pe grafuri, cum ar fi Floyd-Warshall sau Djikstra, care pot găsi cele mai scurte distanțe între fiecare pereche de puncte de pe hartă.

Un graf este o structură de date formată prin noduri și muchii. Muchiile sunt uneori denumite arce (sau linii), iar nodurile sunt numite și vârfuri. Mai simplu spus, un graf este o pereche ordonată G = (V, E) formată din: V care este un set de puncte și E un set de drepte.

Principala problemă pe care dorim să o rezolvăm prin intermediul acestui document este găsirea drumului minim în graf și a costului minim între oricare două noduri. Această problemă este specifică grafurilor ponderate, ce au costuri specifice între două vârfuri. Problema drumului minim poate fi privită din perspectiva atât a grafurilor orientate cât și a celor neorientate, scopul temei îl va face analiza pe grafuri orientate, aciclice, ponderate.

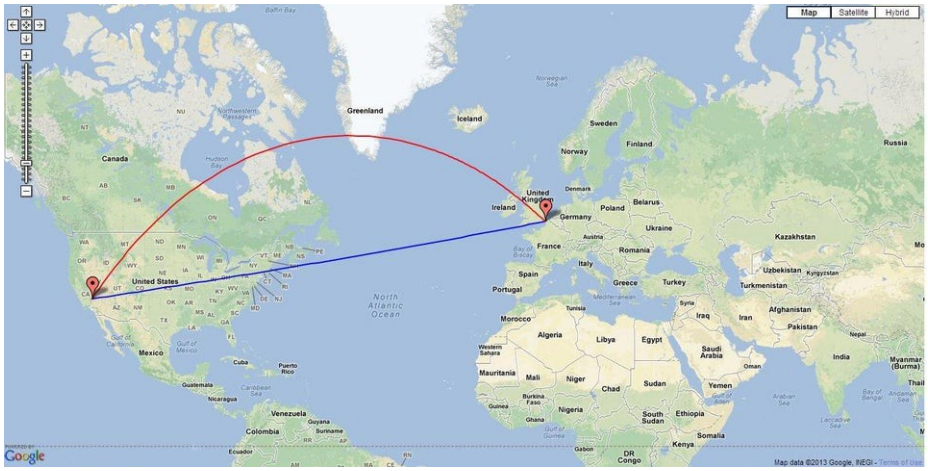


Fig. 1. Disanța minimă dintre două puncte

1.2 Exemple de aplicații practice pentru problema aleasă

Grafurile au numeroase aplicații[1] în diverse domenii: determinarea celui mai scurt drum dintre două localități (În aplicații precum Waze sau Google Maps această problemă trebuie rezolvată prin algoritmul cel mai optim din punct de vedere temporal. Redactarea celei mai bune soluții este esențială, deoarece, în ziua de azi, traficul suprateran din marile metropole ale lumii este din ce în ce mai aglomerat, fiind destul de importantă ghidarea șoferilor către rute alternative, mai rapide., rețelele sociale, structura substanțelor chimice, proiectarea circuitelor electrice, etc.

Pe lângă GPS, există un exemplu și mai cunoscut de utilitazare a unor algoritmi bazați pe grafuri: conectarea cu prietenii pe rețelele sociale, unde fiecare utilizator este un vârf și când se conectează creează o margine(ex. Facebook, unde nodurile sunt utilizatorii, iar o muchie reprezintă relația de prietenie între doi utilizatori).

Mai mult, acești algoritmi sunt utilizați pentru a căuta pagini web. Putem compara o pagină web cu un vârf și legătura dintre două pagini web cu o margine.

1.3 Specificarea soluţiilor alese

Algoritmii aleşi sunt: Floyd-Warshall, Bellman-Ford si Dijkstra.

În informatică, algoritmul Floyd-Warshall este un algoritm pentru găsirea celor mai scurte căi într-un graf ponderat cu greutăți de margini pozitive sau negative (dar fără cicluri negative). O singură execuție a algoritmului va găsi lungimile (greutățile însumate) ale celor mai scurte căi între toate perechile de vârfuri.

Algoritmul Bellman-Ford este un algoritm care calculează cele mai scurte căi de la un singur vârf sursă la toate celelalte vârfuri dintr-un digraf ponderat. Este mai lent decât algoritmul lui Dijkstra pentru aceeași problemă, dar mai versatil, deoarece este capabil să gestioneze grafuri în care unele dintre greutățile marginilor sunt numere negative.

Algoritmul lui Dijkstra este un algoritm pentru găsirea celor mai scurte căi între nodurile dintr-un graf. Pentru un nod sursă dat din graf, algoritmul găsește calea cea mai scurtă între acel nod și oricare altul. Acesta are o performanță mai bună decât algoritmii anteriori, dar nu poate funcționa pe grafuri cu margini negative.

1.4 Specificarea criteriilor de evaluare alese pentru validarea soluțiilor

Pentru a testa corectitudinea, eficiența și performanța algoritmilor aleși, am creat un generator care construiește intrări aleatorii pentru a vedea dacă calea generată de algoritm este cea mai scurtă.

Există mai multe tipuri de teste de intrare și acestea sunt împărțite în două categorii: teste de performanță și teste de acceptare.

Când vorbim despre teste de performanță, putem compara implementările noastre cu altele pentru a vedea care este cea mai rapidă, de preferință intrarea ar trebui să fie una mai mare.

În cazul testelor de acceptare vrem să vedem dacă algoritmul are o implementare bună. Facem asta testând pe teste mai mari sau de tip margine. La final, verificăm dacă rezultatul este cel așteptat.

2 Prezentarea soluțiilor

2.1 Descrierea modului in care funcționează algoritmii aleși

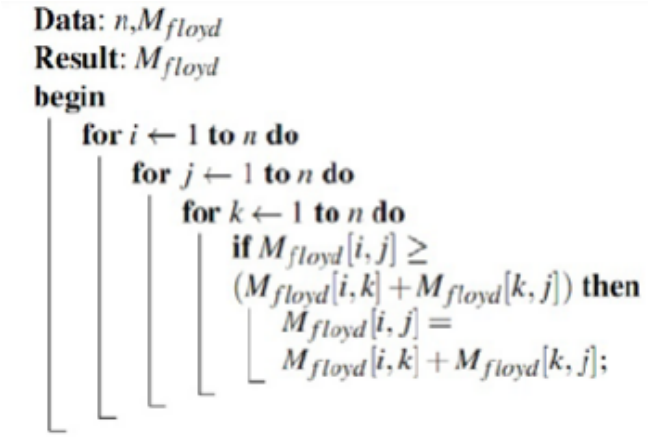


Fig. 2. Pseudocodul algoritmului Floyd-Warshall

Algoritmul Floyd-Warshall

Inițializăm matricea rezultată cu costul muchiilor dintre nodurile date de intrare. După aceasta, actualizăm matricea rezultată calculând cele mai scurte căi și utilizând toate nodurile ca intermediare (unul câte unul). Când ajungem la vârful k, știm deja că vârfurile de la 1 la k - 1 au fost verificate. Pentru fiecare pereche (x, y) (sursă și destinație) există două posibile cazuri: dacă k nu este un nod intermediar, nu actualizăm rezultatul, altfel, noi actualizam rezultatul dacă distanța[x][k] + distanța[k][y] este mai mică decât distanța[x][y].

Un ciclu negativ este un ciclu ale cărui muchii însumate se concretizează într-un rezultat negativ. Nu există o cale cea mai scurtă între oricare două noduri (sau vârfuri) i, j care fac parte dintr-un ciclu negativ, deoarece lungimile căilor de la i la j pot fi arbitrar mici (negative). Pentru rezultate semnificative din punct de vedere numeric, algoritmul Floyd-Warshall presupune că nu există cicluri negative. Cu toate acestea, dacă există cicluri negative, algoritmul Floyd-Warshall[**?**] poate fi utilizat pentru a le detecta.

```
procedure dijkstra(G, l, s)
Input:  Graph G = (V, E), directed or undirected;
        positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output: For all vertices u reachable from s, dist(u) is set
        to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil
dist(s) = 0


H = makequeue(V)  (using dist-values as keys)
while H is not empty:
   u = deletemin(H)
   for all edges (u, v) ∈ E:
       if dist(v) > dist(u) + l(u, v):
          dist(v) = dist(u) + l(u, v)
          prev(v) = u
          decreasekey(H, v)
```

Fig. 3. Pseudocodul algoritmului Djikstra

Algoritmul Djikstra

Pentru a obține calea cea mai scurtă de la un singur vârf sursă la toate celelalte, trebuie să luăm fiecare nod ca nod sursă și apoi să aplicăm algoritmul. Algoritmul are mai mulți pași[5].

În primul rând, creăm un set de arbore cu cea mai scurtă cale care păstrează evidența nodurilor incluse în arborele cu cea mai scurtă cale. Acesta este un tablou care are toate elementele setate ca false în mod implicit.

În al doilea rând, atribuim o valoare distanță tuturor vârfurilor din graf (de preferat să inițializam toate distanțele la infinit). Important este să alocam distanței valoarea 0 pentru sursă, astfel încât aceasta să fie selectată prima.

În ultimul pas, trebuie să actualizăm valoarea distanței tuturor vârfurilor adiacente lui u. Pentru a actualiza valorile distanței, iteram prin toate vârfurile adiacente. Pentru fiecare vârf adiacent v, dacă suma valorilor distanței lui u (de la sursă) și greutatea muchiei u-v, este mai mică decât valoarea distanței lui v, atunci actualizăm valoarea distanței lui v.

```
BellmanFord(graph, start, goal)
    foreach v in graph.vertices
        v.distance = infinity
        v.previous = null

    start.distance = 0

    for i = 1 to (graph.nvertices - 1)
        foreach v in graph.vertices
            foreach u in graph.vertices
            if there is an edge from v to u

                if (v.distance + distance(v, u)) < u.distance
                    u.distance = v.distance + distance(v, u)
                    u.previous = v

    return (goal.distance != infinity)
```

Fig. 4. Pseudocodul algoritmului Bellman-Ford

Algoritmul Bellman-Ford

Pentru a obține calea cea mai scurtă de la un singur vârf sursă la toate celelalte, trebuie să luăm fiecare nod ca nod sursă și apoi să aplicăm algoritmul.

În primul pas, inițializăm distanțele de la sursă la toate vârfurile ca infinite iar distanța până la sursă ca fiind 0. Creăm o matrice dist[] de dimensiunea $|V|$ cu toate valorile infinite, cu excepția dist[src], unde src este vârful sursă.

În al doilea pas, calculăm cele mai scurte distanțe. Executăm următoarea operație de $|V| - 1$ ori (numărul de vârfuri minus unu). Pentru muchiile de la u la v face: dacă $dist[v]$ mai mare decât $dist[u]$ plus greutatea marginii uv, apoi actualizăm $dist[v]$, $dist[v]$ este egal cu $dist[u]$ plus greutatea muchiei uv.

În al treilea și ultimul pas algoritmul raportează dacă există un ciclu de pondere negativă în graf. Pentru muchiile de la u la v face: dacă $dist[v]$ este mai mare decât $dist[u]$ plus greutatea marginii uv, atunci Graful conține un ciclu de greutate negativă". În unele domenii, inteligența artificială în special, algoritmul lui Dijkstra sau o variație a acestuia este cunoscută sub numele de căutare uniformă a costurilor și formulată ca o instanță pentru o idee mult mai generală a best first search.

Algoritmul Bellman-Ford[2] poate fi îmbunătățit în practică (deși nu în cel mai rău caz) prin observația că, dacă o iterație a buclei principale a algoritmului se termină fără a face modificări, algoritmul poate fi terminat imediat, deoarece iterațiile ulterioare nu vor mai face modificări.

Cu această condiție de terminare timpurie, bucla principală poate fi utilizată în unele cazuri cu mult mai puțin decât $|V|1$ iterații, chiar dacă cel mai rău caz al algoritmului rămâne neschimbat.

2.2 Analiza complexității soluțiilor

Algoritmul Floyd-Warshall face o comparație între toate posibilele margini ale grafului dintre fiecare două noduri. Performanța algoritmului este $O(V^3)$ (comparații într-un graf), deși ar putea exista până la $O(V^2)$ căi într-un graf și fiecare combinație de muchii este testată.

Complexitatea de timp a algoritmului lui Dijkstra este $O(V2)$, dar cu coadă cu prioritate minimă scade la $O(V + ElogV)$. Totuși, dacă trebuie să găsim calea cea mai scurtă între toate perechile de vârfuri, ambele metode de mai sus ar fi costisitoare în termeni de timp.

Complexitatea de timp a algoritmului Bellman Ford este relativ mare $O(VE)$, în cazul in care E este egal cu V^2, $O(E^3)$.

2.3 Prezentarea principalelor avantaje și dezavantaje pentru soluțiile luate în considerare

Avantajele algoritmului Floyd-Warshall sunt: este ușor de scris cod pentru acest algoritm într-un program și evidențiază toate perechile de drumuri cele mai scurte dintr-un graf.

Un dezavantaj al acestui algoritm este că rulează mai lent decât alți algoritmi concepute pentru a îndeplini aceeași sarcină.

Algoritmul Djikstra funcționează similar cu Bellman-Ford. Avantajul Djikstra față de Bellman-Ford este că este mai rapid.

Un dezavantaj al algoritmului Djikstra este că nu va funcționa pe grafuri cu cost negativ pe margini.

Principalul avantaj al algoritmului Bellman-Ford este că funcționează bine pe un graf având greutăți pe marginile negative.

Principalul dezavantaj al lui Bellman-Ford este că nu funcționează dacă graful conține cicluri de greutăți negative. În plus, este mai lent decât Djikstra.

Deși fiecare algoritm are avantajele și dezavantajele sale, depinde de situație să decidemm ce tip de graf și algoritm să folosim. De exemplu, vom folosi un graf ponderat pozitiv pentru o hartă GPS, unul negativ pentru afaceri în vânzări, plus multe alte exemple.

3 Evaluare

3.1 Descrierea modalității de construire a setului de teste folosite pentru validare

Am creat un generator care preia numărul de vârfuri și numărul de muchii și generează o intrare aleatorie stocată în fișierul ".in".

Primele teste de validare au fost relativ mici și au fost destinate verificării dacă algoritmii pot rezolva o situație de bază a unui graf.

Treptat, dimensiunile testului de intrare au crescut și testele au ajuns la dimensiuni considerabile. Aceste dimensiuni reprezentau grafuri care nu puteau fi desenate pe hârtie.

Mai mult, am generat câteva cazuri marginale, cum ar fi costuri ponderate negative pentru Djikstra și cicluri ponderate negative pentru Floyd-Warshall și Bellman-Ford. Dacă intrarea aduce algoritmul într-un caz limită, afișează un mesaj sugestiv.

Indiferent de test, am analizat rezultatul și l-am comparat cu cel așteptat, ajungând la concluzia că algoritmii au fost proiectați și construiti corect.

3.2 Specificațiile sistemului de calcul pe care am rulat testele

Sistem de operare: Ubuntu 20.04 LTS

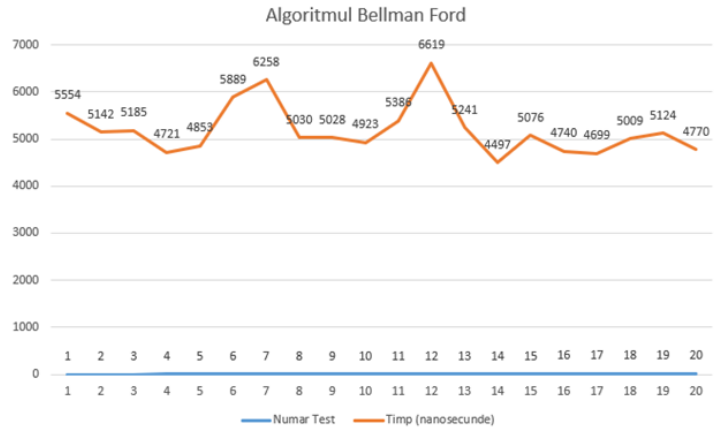
Tipul sistemului: 64-bit operating system, x64-based processor

Procesor: 11th Gen Intel(R) Core(TM) i7-1165G7 2.80GHz

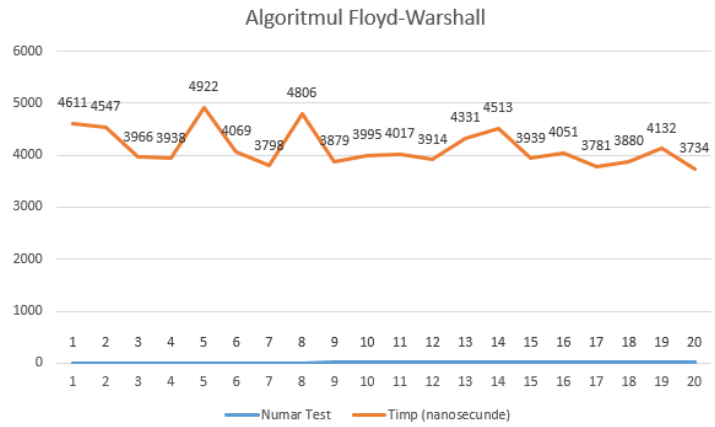
Memorie RAM: 16.0 GB (15.7 GB usable)

3.3 Ilustrarea, folosind grafice/tabele, a rezultatelor evaluării soluțiilor pe setul de teste

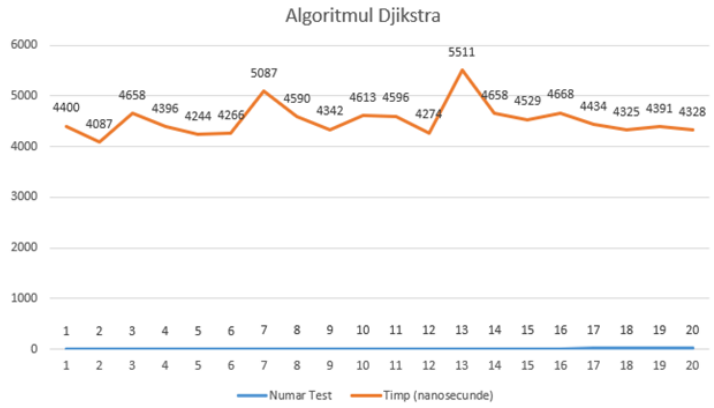
Algoritmul Bellman Ford	
Numar Test	Timp (nanosecunde)
1	5554
2	5142
3	5185
4	4721
5	4853
6	5889
7	6258
8	5030
9	5028
10	4923
11	5386
12	6619
13	5241
14	4497
15	5076
16	4740
17	4699
18	5009
19	5124
20	4770



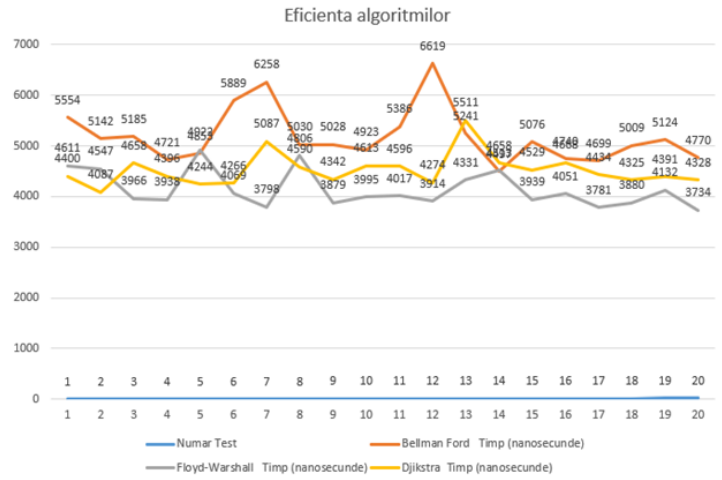
Algoritmul Floyd-Warshall	
Numar Test	Timp (nanosecunde)
1	4611
2	4547
3	3966
4	3938
5	4922
6	4069
7	3798
8	4806
9	3879
10	3995
11	4017
12	3914
13	4331
14	4513
15	3939
16	4051
17	3781
18	3880
19	4132
20	3734



Algoritmul Djikstra	
Numar Test	Timp (nanosecunde)
1	4400
2	4087
3	4658
4	4396
5	4244
6	4266
7	5087
8	4590
9	4342
10	4613
11	4596
12	4274
13	5511
14	4658
15	4529
16	4668
17	4434
18	4325
19	4391
20	4328



Numar Test	Bellman Ford Timp (nanosecunde)	Floyd-Warshall Timp (nanosecunde)	Djikstra Timp (nanosecunde)
1	5554	4611	4400
2	5142	4547	4087
3	5185	3966	4658
4	4721	3938	4396
5	4853	4922	4244
6	5889	4069	4266
7	6258	3798	5087
8	5030	4806	4590
9	5028	3879	4342
10	4923	3995	4613
11	5386	4017	4596
12	6619	3914	4274
13	5241	4331	5511
14	4497	4513	4658
15	5076	3939	4529
16	4740	4051	4668
17	4699	3781	4434
18	5009	3880	4325
19	5124	4132	4391
20	4770	3734	4328



3.4 Prezentarea valorilor obținute pe teste

Chiar dacă Djikstra și Bellman-Ford au o performanță mai bună, natura testului este de asemenea un factor foarte important[5]. După cum putem vedea în graficul de mai sus, graficele tind să semene mult între ele la început, deși în final, când intrarea este mai mare, încep să difere și Dijkstra pare să fie cel mai eficient la testele mai mari.

Un alt factor important este specificația sistemului pe care se află programul lansat. Chiar dacă intrarea este aceeași, nu va exista niciodată aceeași ieșire din cauza diferenţelor date de hardware.

4 Concluzii

Într-o situație din viață reală, una dintre cele mai comune utilizări ale grafurilor și traseelor minime dintr-un graf este, după cum am spus, GPS-ul. Dacă aș dezvolta un GPS, aș face cea mai mare parte a algoritmului Dijkstra datorită performanței sale relativ bune și pentru că nu există margini de greutati negative pe o hartă.

O altă utilizare practică a unui algoritm bazat pe grafuri ar fi în domeniul afacerilor. Greutatea unei margini ar putea reprezenta date utilizate pentru a evalua eficacitatea campaniilor de vânzări. În aceste situații, aș utiliza Bellman-Ford pentru că este mai eficient decât Floyd-Warshall și funcționează pe margini de greutăți negative.

În concluzie, alegerea pe care o faceți pentru algoritm depinde într-adevăr de situație, fiecare are propriile avantaje și dezavantaje și poate fi folosit în diferite situații și în diferite scopuri.

Bibliografie

1. Noțiuni elementare despre grafuri
   `https://infogenius.ro/introducere-teoria-grafurilor/`
   Ultima accesare: 22 Oct 2021.

2. `https://tutoriale-pe.net/algoritmul-bellman-ford-in-c-infoarena/`
   Ultima accesare: 22 Oct 2021.

3. `https://www.programiz.com/dsa/bellman-ford-algorithm`
   Ultima accesare: 22 Oct 2021.

4. `https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/`
   Ultima accesare: 22 Oct 2021.

5. Algoritmi fundamentali. O perspectiva C++ - Razvan Andonie, Ilie Garbacea