Drumuri minime in graf

Olteanu Eduard-Florin

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București, România
ediolteanu99@yahoo.com

Rezumat. Drumul minim într-un graf reprezintă costul minim de la un nod la toate celelalte sau costul minim între oricare două noduri. Un graf este o structură care corespunde unui grup de obiecte, în care unele perechi de obiecte sunt "legate" reciproc de muchii orientate sau nu. Muchiile din graf pot avea sau nu cost. Pentru drumuri minime în graf există diferiți algoritmi, cum ar fi: Dijkstra, Bellman-Ford și Dijkstra adaptat (cunoscut și sub numele de Algoritmul lui Dial), pentru costul minim de la un nod la toate celelalte, și Floyd-Warshall, Johnson și Dijkstra sau Bellman-Ford (aplicate intre oricare două noduri) pentru costul minim între oricare două noduri. În acest studiu vom analiza algoritmii pentru costul minim de la un nod la toate celelalte.

Cuvinte-cheie: graf, cost, Dijkstra, Bellman-Ford, Dial

1 Introducere

1.1 Descrierea problemei rezolvate

Fie un graf ponderat orientat, G (V, E), cu o funcție pentru costurile muchiilor, w definită pe E cu valori în R. Problema își propune să găsească, pentru un nod u, drumul minim de la u la toate celelalte noduri, unde costul unui drum este suma tuturor costurilor muchiilor ce alcătuiesc drumul. Deși, in cele mai multe cazuri, costul este o funcție cu valori pozitive, există situații in care graful cu muchii de cost negativ are relevanță practică. O parte dintre algoritmi pot determina drumul corect de cost minim, inclusive pe aceste grafuri. Totuși, căutarea drumului minim nu are sens în cazul în care graful conține cicluri de cost negativ - un drum minim va avea lungime infinită, întrucât costul său s-ar reduce la fiecare reparcurgere a ciclului.

1.2 Aplicații practice

Algoritmii pentru determinarea drumurilor minime au multiple aplicații practice:

- Găsirea drumului minim dintre doua locații (Google Maps, GPS etc.)
- Stabilirea unei agende de zbor, în vederea asigurării unor conexiuni optime
- Rutare în cadrul unei rețele (telefonice, de calculatoare etc.)

1.3 Specificarea algoritmilor aleși

- **Algoritmul lui Dijkstra**

Algoritmul lui Dijkstra determină pentru un nod dat într-un graf orientat cu costuri costurile minime ale drumurilor care au acel nod ca extremitate inițială.
Mai precis, pentru un nod u – *sursă*, algoritmul determină pentru orice nod v costul minim al unui drum de la s la x.[1]

- **Algoritmul Bellman-Ford**

Algoritmul Bellman Ford poate fi folosit și pentru grafuri ce conțin muchii de cost negativ, dar nu poate fi folosit pentru grafuri ce conțin cicluri de cost negativ (când căutarea unui drum minim nu are sens). Cu ajutorul său putem afla dacă un graf conține cicluri.
Algoritmul folosește același mecanism de relaxare ca si Dijkstra, dar, spre deosebire de acesta, nu optimizează o soluție folosind un criteriu de optim local, ci parcurge fiecare muchie de un număr de ori egal cu numărul de noduri si încearcă sa o relaxeze de fiecare data, pentru a îmbunătăți distanta până la nodul destinație al muchiei curente.

- **Algoritmul lui Dial – Dijkstra adaptat**

Algoritmul lui Dial este un algoritm Dijkstra adaptat pentru costuri mici. Astfel, algoritmul lui Dijkstra este modificat, folosind altă structură de date numită buckets, iar algoritmul alege diferit ți nodul cu costul minim la fiecare pas. Fie C lungimea maximă a unei muchii in G. Algoritmul menține nC + 1 buckets, in bucket-ul k se află toate nodurile cu costul egal cu k (orice cost finit fiind cel mult nC).

1.4 Criteriile de evaluare pentru soluția propusă

Am implementat un algoritm care alocă o valoare random pentru fiecare muchie dintr-un graf complet orientat cu n noduri. Atunci când costul muchiei este 0 am considerat că acolo nu se află muchie. Am creat un set de 10 teste cu costuri între 1 și 100 pentru o testare normală a tuturor celor 3 algoritmi. Am creat încă un set de 10 teste cu costuri între 1 și 10 pentru testarea eficienței algoritmului lui Dial care este mai eficient decât Dijkstra pe grafuri cu cost mic. In final am mai creat un set de 10 teste ce conțin și muchii negative pentru a arăta corectitudinea algoritmului lui Bellman-Ford pe grafuri cu costuri negative. Eficiența va fi testată folosind un număr mare de noduri. ($n \in [100, 1000]$).

2 Prezentarea soluțiilor

2.1 Algoritmul lui Dijkstra

- Descrierea modului în care funcționează algoritmul

Algoritmul lui Dijkstra este un algoritm care funcționează numai pe grafuri orientate. De asemenea, algoritmul lui Dijkstra funcționează atât pe grafuri conexe cât și pe grafuri neconexe.

Algoritmul primește matricea de adiacență și inițializează un vector de distanțe minime care urmează să fie updatat cu valoarea inițială a costului dacă există muchie sau cu INF dacă nu. De asemenea este creat și un vector de vizitare. Algoritmul pornește de la nodul de start dat ca parametru și trece prin toate nodurile nevizitate pentru a găsi distanța minimă. Atunci vizitează nodul și parcurge din nou toate nodurile pentru a vedea dacă se poate updata distanta precedentă cu o nouă distanță mai mică, scăzând costul total. La final in vectorul output se trec distanțele minime de la nodul de start la celelalte noduri.

- Analiza complexității

În această evaluare am folosit algoritmul cu complexitatea $O(V^2)$, deși cu ajutorul unui heap complexitatea se poate reduce la O(E*logV) (V - numărul de noduri, E - numărul de muchii). Complexitatea $O(V^2)$ este dată de faptul că sunt folosite parcurgeri imbricate - while de la 0 la V - 2 și for de la 0 la V - 1.

- Avantaje și dezavantaje

Avantaje:

- Este un algoritm dinamic
- Este easy to debug
- Este o metodă de tip Greedy
- Este eficient atunci când este implementat folosind un heap, mai ales pentru grafuri mai puțin dense.

Dezavantaje:

- Nu poate fi aplicat pe grafuri care conțin muchii cu cost negativ
- Funcționează doar pentru grafuri orientate
- Este greu de implementat

2.2 Algoritmul Bellman-Ford

- Descrierea modului în care funcționează algoritmul

Algoritmul Bellman-Ford este un algoritm care, spre deosebire de Dijkstra funcționează și pe grafuri cu muchii cu cost negativ.

Mai întâi este inițializat un vector ce reprezintă distanțele, inițial acestea fiind INF.

Apoi este efectuat algoritmul trecând prin fiecare muchie de V - 1 ori. Pentru fiecare muchie se verifică dacă se poate actualiza distanța precedentă cu una mai mică, cea nou calculată.

La final, ținând cont ca este un algoritm care funcționează si cu grafuri cu muchii cu cost negativ, se verifică dacă există un ciclu negativ și se afișează.

- Analiza complexității

Spre deosebire de algoritmul lui Dijkstra care în implementarea cu matrice de adiacență și heap poate ajunge la o complexitate O(E*logV), algoritmul Bellman-Ford are o complexitate O(V*E) (V - numărul de noduri, E - numărul de muchii). Pe testele create de mine, E tinde la V^2, deci algoritmul are complexitatea $O(V^3)$.

De asemenea, varianta implementării algoritmului cu coadă este mai rapidă deși are aceeași complexitate deoarece nu trebuie calculate toate distanțele minime, acestea pot rămâne în coadă de la pasul anterior.

- Avantaje și dezavantaje

Avantaje:

- Unul din cele mai mare avantaje ale algoritmului este că funcționează pe grafuri cu muchii cu cost negativ
- Este easy to debug
- Este o metodă de tip Greedy
- Este ușor de implementat

Dezavantaje:

- Are complexitate mai mare decât algoritmul lui Dijkstra
- Funcționează doar pentru grafuri orientate
- Este mult mai puțin eficient din punct de vedere temporal, mai ales pentru grafuri foarte dense.

2.3 Algoritmul lui Dial - Dijkstra adaptat

- Descrierea modului în care funcționează algoritmul

Algoritmul lui Dial este o variantă îmbunătățită a algoritmului lui Dijkstra care este mai eficient pe grafuri cu costuri mici.

Algoritmul creează o listă de buckets cu W * V + 1 elemente (V - numărul de noduri, W - costul maxim al unei muchii). Bucket-ul k conține toate nodurile cu distanța față de nodul de start egală k. Nodurile din fiecare bucket sunt salvate printr-o listă.

Sunt parcurse toate bucket-urile până când se găsește unul care nu este gol. Fiecare nod din acest bucket este la distanță minimă. Atunci când distanța relativă față de un nod se modifică acesta este șters din bucket-ul precedent și adăugat în bucket-ul corespunzător pentru costul curent. Algoritmul se execută până când toate nodurile au fost puse în bucket-urile finale sau până când au fost parcurse toate distanțele.

- Analiza complexității

Spre deosebire de ceilalți doi algoritmi care folosesc doar numărul de muchii respectiv de noduri pentru implementare respectiv complexitate, complexitatea algoritmului lui Dial corespunde direct cu costul maxim al muchiilor. Algoritmul lui Dial are complexitatea O(E + W * V) (V - numărul de noduri, E- numărul de muchii, W - costul maxim al unei muchii). Astfel, cu cât costul maxim al muchiilor este mai mic cu atât este mai eficient pentru același număr de noduri și muchii.

- Avantaje și dezavantaje

Avantaje:

- Este mult mai eficient decât algoritmul lui Dijkstra pentru muchii cu costuri mici.

Dezavantaje:

- Este greu să faci o comparație între complexitatea lui față de celelalte din cauza faptului că se calculează în 3 dimensiuni.
- Funcționează doar pentru grafuri orientate
- Este greu de implementat

3 Evaluare

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

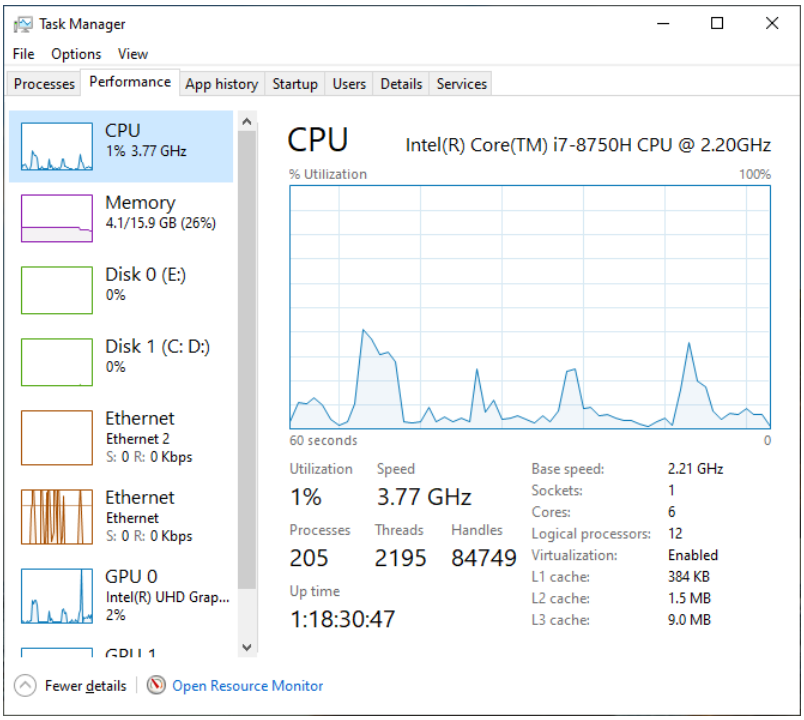
Pentru generarea testelor am implementat un program C++ care generează toate muchiile. Atunci când este generată o muchie cu costul 0 am considerat că acea muchie nu există. Costurile muchiilor au fost generate random folosind device-ul random din C++. Numărul de noduri este dat de la tastatura, fiind valori de la 5 la 1000.

Am generat 3 seturi de câte 10 teste pentru a verifica eficiența și corectitudinea algoritmilor. Astfel, 10 teste au muchii cu costuri între 1 și 100, 10 teste au muchii cu costuri între 1 și 10, iar ultimele 10 teste conțin muchii cu costuri negative, acestea fiind folosite pentru testarea corectitudinii algoritmului Bellman-Ford pentru cazul cu muchii negative.

3.2 Specificațiile sistemului de calcul

Sistemul de calcul pe care am rulat algoritmii are specificațiile:

- Procesor: Intel® Core™ i7-8750H CPU @ 2.2GHz până la 4.1GHz
- Memorie disponibilă: aproximativ 12GB din 16GB



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

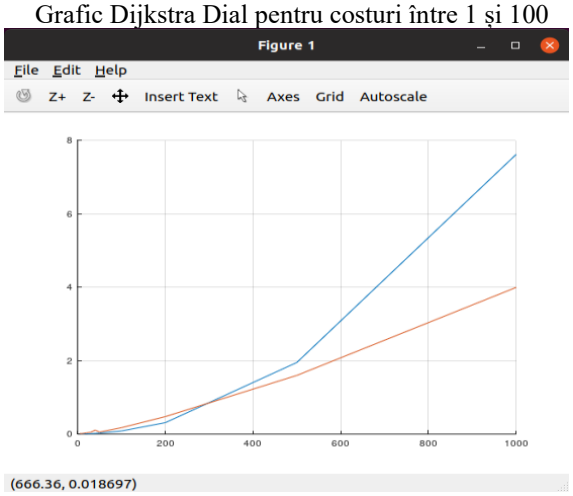
Timpul in milisecunde pentru algoritmii aplicați pe testele cu costul muchiilor între 1 și 100

Nr. noduri	Dijkstra	Dial	Bellman-Ford
5	0.0017	0.0167	0.0012
10	0.0102	0.0132	0.0021
20	0.0047	0.0333	0.0084
30	0.0113	0.0502	0.0257
40	0.0185	0.1086	0.058
50	0.0272	0.0554	0.1512
100	0.085	0.1746	0.9009
200	0.3116	0.4764	7.6967
500	1.9517	1.5992	116.5036
1000	7.6159	3.9948	935.8671

Timpul in milisecunde pentru algoritmii aplicați pe testele cu costul muchiilor între 1 și 10

Nr. noduri	Dijkstra	Dial	Bellman-Ford
5	0.0018	0.0017	0.0013
10	0.0024	0.0029	0.002
20	0.0043	0.0038	0.0075
30	0.0075	0.0057	0.0232
40	0.0106	0.0078	0.0518
50	0.0141	0.0098	0.1015
100	0.0359	0.0413	0.8495
200	0.1141	0.086	6.7489
500	0.6741	0.6245	104.009
1000	6.7247	2.3303	846.0658

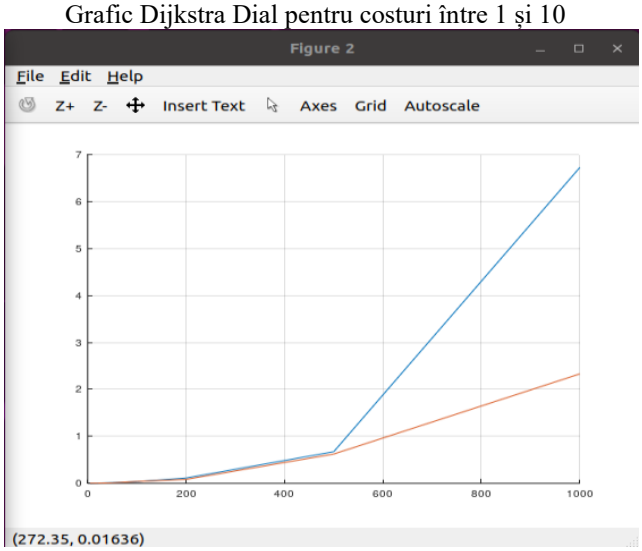
Grafic Dijkstra Dial pentru costuri între 1 și 100



Dijkstra - albastru

Dial - portocaliu

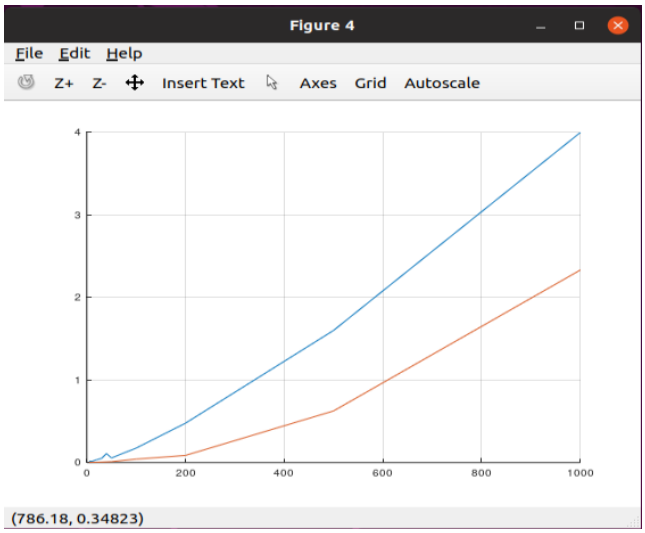
Grafic Dijkstra Dial pentru costuri între 1 și 10



Dijkstra - albastru

Dial - portocaliu

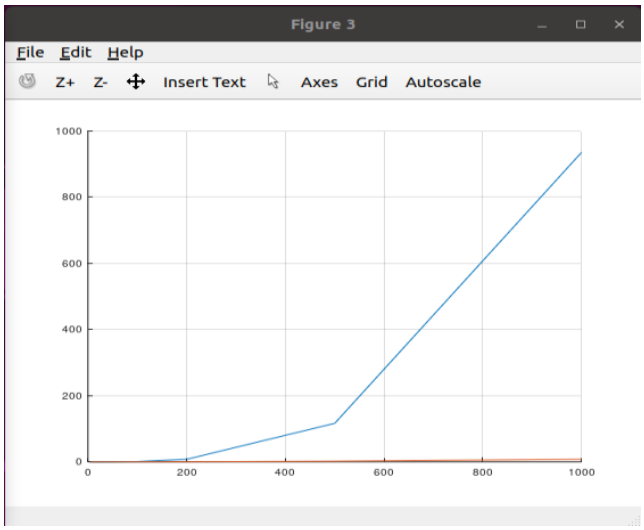
Grafic Dial vs Dial pentru ambele seturi de costuri



Costuri 1-100 - albastru

Costuri 1-10 - portocaliu

Grafic Dijkstra Bellman-Ford pentru costuri între 1-100



Bellman-Ford - albastru

Dijkstra - portocaliu

3.4 Prezentarea valorilor obținute pe teste

În principiu valorile rezultate de teste sunt conform așteptărilor. Toți cei trei algoritmi returnează aceleași distanțe, lucru verificat de checker-ul implementat în etapa precedentă. De asemenea, cu privire la grafice, rezultatele arată complexitățile diferite ale algoritmilor. Se observă că diferența de complexitate dintre Dijkstra și Dial nu este mare pe testele date, dar în ambele cazuri Dial este mai eficient. De asemenea se observă că există diferență de timp la algoritmul lui Dial în funcție de costul maxim al muchiilor. În cazul în care costul maxim este 10 se observă o îmbunătățire în eficiența algoritmului. În final, cel mai bine se observă diferența de complexitate dintre Dijkstra și Bellman-Ford.

De asemenea, apare un spike atunci când rulează Dijkstra pe testul cu 10 noduri cu costuri de la 100. Acesta poate fi datorat faptului că este posibil ca, în comparație cu celelalte teste, acesta să fie worst case possible, ajungând astfel să dureze mai mult pe testul cu 10 noduri decât pe cel cu 20 de noduri.

4 Concluzii

În concluzie, dintre cei trei algoritmi implementați, pe orice set de teste mai rulează cel mai eficient algoritmul lui Dial. Deși este mai greu de implementat și de înțeles acesta și-a demonstrat eficiența pe teste cu costuri mici. Probabil dacă costurile ar fi fost mai mari, eficiența acestuia ar fi scăzut drastic ținând cont de faptul ca el are complexitatea direct dependentă de costul maxim. De asemenea, se observă ca algoritmul Bellman-Ford este aproximativ la fel de eficient ca Dijkstra pentru testele de până la 200 de noduri, mai departe fiind foarte ineficient din cauza complexității sale de O(V*E).

5 Bibliografie

1. https://www.pbinfo.ro/?pagina=articole&subpagina=afisare&id=6135
2. https://www.geeksforgeeks.org/dials-algorithm-optimized-dijkstra-for-small-range-weights/
3. http://elf.cs.pub.ro/pa/wiki/laboratoare/laborator-08#algoritmul-bellman-ford
4. https://courses.csail.mit.edu/6.006/spring11/lectures/lec15.pdf
5. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/