Proiect Analiza Algoritmilor

Chelcea Claudiu-Marian[322CA]

Universitatea Politehnica din Bucureşti
Facultatea de Automatică şi Calculatoare
claudiuchelcea01@gmail.com

Abstract. În această lucrare vom analiza, compara şi discuta principalii algoritmi ce abordează tematica lucrării mele şi anume: Drumuri minime in graf - Costul minim de la un nod la toate celelalte.

Fiecare algoritm abordat va avea propria explicaţie şi mod de prezentare unic, dar, pentru fiecare dintre acestea, se va încerca urmarea unei structuri şi anume:

- Prezentare
- Avantaje şi dezavantaje
- Complexităţi
- Alte detalii

Keywords: Dijkstra · Bellman-Ford · Graphs · Distanţe minime

1 Introducere

1.1 Descrierea problemei rezolvate

Grafurile nu au nevoie de o introducere. Teoria grafurilor s-a dezvoltat împreună cu algebra şi au multiple aplicaţii practice, fiind strâns legate de multe ramuri ale matematicii, cât şi ale informaticii.

Printre utilităţile lor se află, în special, modelarea de situaţii din viaţa reală: conexiuni, reţele de calculatoare, algoritmi de căutare (Page Rank), hărţi rutiere, etc...

Lucrarea curentă urmăreşte analiza algoritmilor principali folosiţi în determinarea drumurilor minime între noduri, acest lucru având multe utilităţi practice, câteva dintre acestea fiind enumerate mai jos.

1.2 Aplicaţie practică la problemă

Printre cele mai importante aplicaţii practice rezolvate folosind drumurile minime regăsim:

- Găsirea drumului minim dintre doua locații (Google Maps, GPS etc.)
- Rutare in cadrul unei rețele (telefonice, de calculatoare etc.)
- Găsirea de sugestii (de ex. de prietenie) pe reţelele sociale
- Roboţi inteligenţi

1.3 Specificarea soluţiilor alese

Pentru a rezolva problema drumurilor minime, voi aborda algoritmii Bellman-Ford şi Dijkstra (normal si optimizat), cât şi shortest path într-un graf aciclic orientat. Ei vor fi detaliaţi urmărind aceeaşi structură şi vor fi testaţi, pe cât posibil, cu seturi de date similare.

Algoritmul Dijkstra se bazează pe etichete aflate pe ramuri ce reprezintă distanţa dintre doua noduri. Acesta are o complexitate de $O(V^2)$, unde V reprezintă numărul de noduri, complexitate care este redusă la $O(V + E * \log V)$, unde E reprezintă numarul de muchii, atunci când folosim o coadă de prioritaţi.

Algoritmul Bellman Ford poate fi folosit doar atunci căndnu există niciun ciclu în graf. Acesta funcţionează după principiul de actualizare constantă a distanţei dintre noduri în timp ce sunt parcurse, ca în final să se atingă soluţia optimă. Complexitatea acestui algoritm este $O(V * E)$, unde V reprezintă numărul de noduri, iar E numărul de muchii.

1.4 Criteriile de evaluare pentru soluţia propusă

În evaluarea algoritmilor voi folosi în primă fază seturi de teste din surse externe, datorită faptului că acestea ofera şi raspunsul, astfel putând confirma atât corectitudinea algoritmului, cât şi compara cu timpii de rulare oferiţi de aceste surse.

De asemenea, îmi voi crea propriile seturi de date în încercarea de a exploata slabiciuni ale algoritmilor.

Voi folosi seturi variate de date şi cantitaţi diferite de date, iar, în final, voi face medii din răspunsurile pe care le-am obţinut, atât pe PC-ul meu, dar şi pe alte PC-uri şi compilatoare online.

2 Prezentarea soluţiilor

Aşa cum am menţionat anterior, algoritmii implementaţi sunt următorii:

- Dijkstra distanţe minime
- BellmanFord
- Dijkstra optimizat
- Cele mai scurte căi într-un graf orientat aciclic

2.1 Dijkstra distanţe minime

Algoritmul Dijkstra este folosit pentru găsirea celor mai scurte căi între noduri dintr-un graf. A fost conceput de informaticianul Edsger W. Dijkstra în 1956 și publicat trei ani mai târziu.

Algoritmul există în multe variante. Cel original a găsit calea cea mai scurtă între două noduri date, dar o variantă mai comună fixează un singur nod ca nod "sursă" și găsește cele mai scurte căi de la sursă la toate celelalte noduri din grafic, producând cea mai scurtă cale.

În implementarea mea, doi vectori memoreaza distanţele şi nodurile vizitate. Dupa aceea, distanţele sunt modificate progresiv pentru a ajunge la distanţele minime. În final, se afişează rezultatele.

Acesta are o complexitate de $O(V^2)$, unde V reprezintă numărul de noduri, complexitate care este redusă la $O(V + E * \log V)$, unde E reprezintă numarul de muchii, atunci când folosim o coadă de prioritaţi.

Avantaje:

- Complexitatea de $O(n^2)$ este una suficient de bună pentru utilizarea acestuia pe grafuri mari.
- Poate fi folosit pentru a calcula distanţa minimă de la un nod la celelalte, cât şi la un singur nod, oprind algoritmul după ce atinge nodul specificat.

Dezavantaje:

- Nu funcţionează pe muchii cu valori negative.
- Necesitatea de a urmări ce noduri au fost vizitate.

2.2 BellmanFord

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 grafice în care unele dintre greutățile muchiilor sunt numere negative.

Greutățile marginilor negative se găsesc în diverse aplicații ale graficelor, de unde și utilitatea acestui algoritm. Dacă un grafic conține un "ciclu negativ" (adică un ciclu ale cărui margini însumează o valoare negativă) care este accesibil de la sursă, atunci nu există calea cea mai rapidă: orice cale care are un punct

pe ciclul negativ poate fi făcută mai rapidă prin încă o plimbare în jurul ciclului negativ. Într-un astfel de caz, algoritmul Bellman-Ford poate detecta și raporta ciclul negativ.

În implementarea mea, se setează vectorul de distanţe cu valori maxime. Acesta se actualizează cu ajutorul greutăţiilor de pe muchii, iar apoi se setează distanţele minime.

Complexitatea algoritmului este O(VE), unde V reprezintă numarul de noduri, iar E reprezintă numarul de muchii. Avantaje:

- Lucrul cu muchii negative.
- Poate fi folosit pentru a depista cicluri negative.

Dezavantaje:

- Complexitatea mare.
- Nu poate rezolva cicluri negative, ci doar muchii negative.

2.3 Dijkstra optimizat

Algoritmul lui Dijkstra poate fi modificat prin utilizarea diferitelor structuri de date, găleți (buckets), care se numește implementare de apelare a algoritmului lui Dijkstra. complexitatea temporala devenind O (E + WV) unde W este greutatea maximă pe orice margine a graficului, deci putem vedea că, dacă W este mic, atunci această implementare rulează mult mai rapid decât algoritmul tradițional.

În implementarea mea, se creează o listă de găleţi. Se merge prin fiecare bucket, şi, atât timp cât nu sunt goale toate, procesează nodul din vârf şi actualizează distanţele faţă de acel nod în funcţie de greutăţi.

Complexitatea algoritmului este O(E + WV), unde V reprezintă numarul de noduri, E reprezintă numarul de muchii, iar W reprezină greutatea maxima a unei muchii.

Avantaje:

- Mai rapid decat algoritmul normal dacă avem greutăti mici.

Dezavantaje:

- Utilizarea mai multă de memorie.

2.4 Cele mai scurte căi într-un graf orientat aciclic

Pentru un grafic ponderat general, putem calcula cele mai scurte distanțe de o singură sursă în timp O(VE) folosind algoritmul Bellman-Ford. Pentru un grafic fără ponderi negative, putem face mai bine și putem calcula cele mai scurte distanțe cu o singură sursă în timp O(E + VLogV) folosind algoritmul

lui Dijkstra. Putem calcula cele mai scurte distanțe de o singură sursă în timp O(V+E) pentru un grafic aciclic orientat folosind sortarea topologică.

În implementarea mea, sortăm topologic graful şi adaugăm în acest sens nodurile într-un stack din care scoatem progresiv nodurile şi actualizăm distanţele.

Complexitatea algoritmului este O(V+E), unde V reprezintă numarul de noduri, iar E reprezintă numarul de muchii.

Avantaje:

– Cel mai rapid algoritm dintre cele 4.

Dezavantaje:

– Deşi este cel mai rapid algoritm, acesta are dezavantajul de a funcţiona doar în cazul unui graf aciclic orientat,

3 Evaluare

3.1 Set de teste

Setul de teste a provenit din mai multe surse. În primul rând, am preluat teste din surse online precum GeeksForGeeks pentru a testa implementările mele cu teste valide realizate de alti programatori. În al doilea rănd, am generat manual teste pentru a testa funcţionalitatea generală a algoritmilor, cât şi funcţionarea în cazuri limită. Spre exemplu, am testat modul în care se execută algoritmii în cazul unor cicluri negative. Un alt exemplu este testarea plecării din diferite noduri, chiar şi din noduri inexistente, pentru a testa rezultatul.

3.2 Sistem

Sistemul pe care au rulat testele include un procesor AMD R3 2200G, o placă video RX 550 2 GB si RAM Kingston FURY Beast 8GB DDR4, 3200MHz CL16, Dual Channel Kit

3.3 Ilustrare rezultate

Nu pot folosi vreun grafic aici sau tabel, căci toate rezultatele se văd la rularea programului, fiecare test afişând output-ul programului, cât si output-ul aşteptat.

3.4 Prezentare valori

Din rularea testelor se afisează output-ul programului, cât si output-ul asteptat.

4 Concluzii

În urma analizei, fiecare algoritm excelează în anumite situaţii. Astfel, aş opta pentru BellmanFord când lucrăm pe grafuri care conţin muchii negative. Pentru grafuri orientate acicilice, cel mai eficient algoritm pe care îl voi folosi este Cele mai scurte căi într-un graf orientat aciclic. În restul cazurilor, voi folosi Dijkstra sau Dijkstra optimized, în funcţie de device-ul pe care lucrez. Dacă memoria nu este o problema, voi folosi algoritmul optimizat, iar, în caz contrar, pe cel normal.

Referinţe

1. OCW: https://ocw.cs.pub.ro/courses/pa/laboratoare/laborator-09
2. OCW: https://ocw.cs.pub.ro/courses/pa/laboratoare/laborator-07
3. RASFOIESC: https://www.rasfoiesc.com/educatie/matematica/Drumuri-minime-in-grafuri24.php
4. GEEKS FOR GEEKS: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
5. GEEKS FOR GEEKS: https://www.geeksforgeeks.org/shortest-path-in-a-directed-graph-by-dijkstras-algorithm/
6. GEEKS FOR GEEKS: https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/
7. HACKEREARTH: https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/