Drumuri Minime in Graf

Cristescu Diana-Andeea

[1] Universitatea Politehnica din București
[2] Facultatea de Automatică și Calculatoare

Abstract. Algoritmii de calculare a drumurilor minime în graf sunt folosiți pentru a rezolva probleme legate de distanțe și costuri în domenii precum cartografie, robotică și telecomunicații. În lucrare se vor prezenta atat avantajele cat și dezavantajele algoritmilor Dijkstra, Dijkstra adaptat și cea mai eficientă metodă pentru grafuri orientate aciclice. Se vor implementa algoritmii, se vor verifica pe un set de teste și se vor compara, stabilindu-se care este mai eficient și în ce cazuri.

Keywords: Dijkstra · Dial's · sortare topografica · tip de date Bucket.

1 Introducere

1.1 Descrierea Problemei

Algoritmii de drumuri minime sunt o familie de algoritmi concepuți pentru a rezolva problema drumurilor minime în graf. Acestea funcționează pe un graf dat care este compus dintr-un set de noduri și laturi. Dacă laturile au costuri asociate lor graful se numește ponderat. Grafurile mai pot fi și orientate, în cazul în care laturile au o direcție în care pot fi traversate sau neorientate în caz contrar. Din cauza acestor diferențe este necesară existența a mai mulți algoritmi pentru rezolvarea problemei drumurilor minime.

Algoritmii de drumuri minime sunt folosiți în practică în multe domenii cum ar fi: serviciul de hărți digitale Google Maps unde calculează distanța minimă dintre două puncte pe hartă, aplicații social media pentru a-ți sugera profilurile oamenilor pe care este posibil să îi recunoști în viața reală, rețeaua de telefonie, IP routing, calcularea drumurilor cu avionul pentru corporații sau în roboți pentru a calcula cel mai scurt drum până la destinație.

1.2 Prezentarea Algoritmilor Aleși

Dijkstra Se poate aplica numai pe grafuri cu costuri pozitive din cauza faptului că pentru a calcula distanța dintre două noduri adună costurile laturilor intermediare. Dacă graful are costuri negative pe unele laturi suma nu va reprezenta distanta reala.

Dijkstra adaptat (Dial's Algorithm) Acest algoritm este o versiune modificată a algoritmului Dijkstra, ceea ce îl face sa aibă aceleași dezavantaje. Pe lângă faptul ca nu poate fi folosit pe grafuri cu costuri negative implementarea lui necesită alegerea unui cost maxim. În funcție de costul ales va fi nevoie de creerea a C*N tipuri de date Bucket, unde N este numărul nodurilor din graf, C costul maxim, iar produsul lor distanța maximă.

Cea mai eficienta metoda pentru grafuri orientate aciclice (DAG) DAG este un algoritm care se bazează pe principiul sortarii topologice. Precum Dijkstra, acest algoritm este optimizat pentru grafuri orientate aciclice, ceea ce înseamnă că nu poate fi folosit pe grafuri ciclice sau neorientate. Un avantaj pe care îl are comparativ cu Dijkstra este faptul că graful nu trebuie sa aibă numai costuri pozitive.

2 Analiza complexitatii algoritmilor

2.1 Dijkstra

Implementarea aleasa consta in folosirea unei liste sortate crescator(pq) in functie de costurile asociate fiecarui nod si o lista in care vom salva costurile drumurilor deja calculate(shortestPaths). Atat pq cat si shortestPaths vor fi initializate cu 0 pentru nodul de la care vom calcula costul drumurilor si myInf(valoare maxima pe care o poate lua un integer) in rest. Cat timp pq nu este gol se va extrage din el nodul cu cel mai mic cost asociat(N) si se va pune in shortestPaths, dupa care se va calcula distanta de la sursa prin N pentru toti vecinii lui N si se va actualiza valoarea din pq in caz ca distanta noua este mai mica decat cea salvata. Acest proces se va repeta pana cand se vor gasii distantele cele mai mici de la sursa pana la toate celalalte noduri. Rezultatul va fi salvat in array-ul shortestPaths.

Complexitatea

$$T(V) = c_0 + V * c_1 + V * (c_2 + V * c_3 + c_4 * (E - x) + c_5 * (E - y)) \quad (1)$$

$$=> O(V^2) \quad (2)$$

(V este numarul de noduri, iar E numarul nodurilor alaturate nodului curent)

Avantaje Este cea mai generala si usor de implementat solutie. Atat Dial's cat si DAG sunt variante optimizate ale acestui algoritm.

Dezavantaje Fiind varianta neoptimizata in cazul in care graful are costuri mici sau este orientat aciclic se pot optine rezultate mai bune folosind ceilalti algoritmi.

2.2 Dijkstra adaptat (Dial's Algorithm)

Dial's este implementat folosind o structura de date numita bucket care consta intr-un vector de liste inlantuite. Se va creea cate o lista inlantuit pentru fiecare valoare posibila pe care o poate lua costul unui drum (maximul fiind costul maxim dintre oricare doua noduri vecine * numarul de noduri din graf). Pe langa structura bucket(bucket) se va folosii si o lista in care vom salva costurile drumurilor deja calculate(shortestPaths). ShortestPaths va fi initializat cu 0 pentru nodul de la care vom calcula costul drumurilor si myInf(valoare maxima pe care o poate lua un integer) in rest, iar in bucket se va adauga indicele nodului sursa in lista asociata costului 0. Calculul costurilor drumurilor se va face trecand prin listele din bucket incepand de la cea asociata costului 0, sarind peste cele goale. Pentru fiecare lista se vor scoate pe rand indexurile nodurilor salvate(N) si se vor recalcula distantele de la sursa prin N pentru toti vecinii lor, daca distanta calculata este mai mica decat cea din shortestPaths aceasta se va actualiza. Procesul se repeta pana cand nu mai exista liste neaccesate in bucket, iar rezultatul se va afla in shortestPaths.

Complexitatea

$$T(V) = c_0 + V * c_1 + W * (c_2 + V * c_3 + c_4 * (E - x) + c_5 * (E - y)) \quad (3)$$

$$=> O(W * V) \quad (4)$$

(V este numarul de noduri, E numarul nodurilor alaturate nodului curent, iar W costul maxim al drumurilor)

Avantaje In cazul in care graful are costuri foarte mici iar W este mai mic decat V algoritmul va fi mult mai eficient decat celalalte doua optiuni.

Dezavantaje Faptul ca depinde atat de costul maxim cat si de numarul de noduri vecine din graf face acest algoritm extrem de ineficient in cazul in care fie costul, fie numarul vecinilor e mare.

2.3 Cea mai eficienta metoda pentru grafuri orientate aciclice (DAG)

Implementarea utilizata foloseste trei liste: sortarea topologica a grafului(sortedGraph), lista nodurilor vizitate(visited) si cea in care se va afla rezultatul calculelor(shortestPaths). SortedGraph consta intr-un array al indexurilor in ordine topologica astfel incat prin parcurgerea lui putem fii siguri ca nu o sa accesam un nod pentru care nu putem calcula drumul de la sursa folosind drumurile deja calculate si salvate. ShortestPaths va fi initializat cu 0 pentru nodul de la care vom calcula costul drumurilor si myInf(valoare maxima pe care o poate lua un integer) in rest, visited va fi initializat cu FALSE pentru toate nodurile, iar sortedGraph va

fi calculat folosind functia topologicalSortRecursive. TopologicalSortRecursive primeste ca parametrii matricea grafului, array-ul visited, indexul nodului sursa si array-ul sortedGraph si are ca obiectiv popularea acestuia. Algoritmul consta in parcurgerea listei sortedGraph, care ne va da nodul curent(N) si calcularea pentru toti vecinii lui drumul de la sursa prin N. In cazul in care costul gasit este mai mic decat cel din shortestPaths i se va actualiza valoarea. Procesul se repeta pana cand avem costul drumului cel mai mic de la sursa pana la toate nodurile din graf.

Complexitatea

$$T(V) = c_0 + V * c_1 + V * (c_2 + (E - z) * c_3) + V * (c_4 + V * c_5 + (E - x) * c_6 + (E - y) * c_7) \tag{5}$$

$$=> O(V^2) \tag{6}$$

(V este numarul de noduri, iar E numarul nodurilor alaturate nodului curent)

Avantaje Este cea mai buna varianta in cazul in care graful este orientat aciclic.

Dezavantaje Graful trebuie sa fie aciclic, in caz contrar nu va rula.

3 Evaluarea Algoritmilor

3.1 Criterii

Pentru compararea algoritmilor descriși vom testa atât eficiența lor de a calcula drumurile minime de la un nod la toate celelalte pe grafuri orientate aciclice, cât și corectitudinea listei de costuri rezultate prin compararea cu soluțiile calculate în prealabil. Tipul grafurilor folosite sunt:normal, rar, digraf complet și cu porțiuni izolate; toate opțiunile având o variantă cu costuri mici și normale. În cazul în care există noduri inaccesibile se va accepta ca rezultat corect infinit. Testele vor pune în evidență atât timpul de execuție cât și spațiul necesar rulării. Rezultatele vor fi comparate folosind un tabel din care vor reieși cazurile în care este eficientă folosirea fiecărui algoritm.

3.2 Teste

Testele au fost rulate pe un MacBook Pro Retina Early 2013: procesor 2,6 GHz Dual-Core Intel Core i5, memorie 8 GB 1600 MHz DDR3, graphics Intel HD Graphics 4000 1536 MB, iar codul a fost scris in Java.

	Dijkstra				Dial's				DAG			
1	95ms	44ms	44ms	49ms	50ms	62ms	45ms	44ms	65ms	53ms	43ms	45ms
2	53ms	45ms	46ms	46ms	48ms	48ms	51ms	43ms	50ms	61ms	47ms	45ms
3	47ms	44ms	43ms	41ms	50ms	46ms	45ms	47ms	42ms	55ms	46ms	48ms
4	46ms	44ms	45ms	46ms	49ms	46ms	51ms	49ms	46ms	47ms	43ms	47ms
5	48ms	45ms	49ms	47ms	54ms	51ms	52ms	47ms	48ms	44ms	45ms	47ms
6	66ms	51ms	43ms	45ms	48ms	48ms	51ms	46ms	54ms	55ms	56ms	49ms
7	40ms	44ms	43ms	51ms	51ms	56ms	47ms	52ms	42ms	56ms	50ms	49ms
8	47ms	45ms	48ms	45ms	45ms	80ms	49ms	46ms	54ms	47ms	45ms	43ms
9	48ms	48ms	44ms	46ms	49ms	45ms	59ms	49ms	52ms	43ms	46ms	47ms
10	48ms	46ms	46ms	88ms	48ms	48ms	48ms	50ms	50ms	48ms	47ms	44ms
11	49ms	45ms	44ms	44ms	52ms	49ms	48ms	57ms	54ms	44ms	70ms	65ms
12	51ms	49ms	48ms	54ms	88ms	72ms	58ms	56ms	59ms	50ms	48ms	46ms
13	51ms	45ms	51ms	43ms	56ms	50ms	47ms	47ms	90ms	47ms	49ms	70ms
14	43ms	50ms	58ms	44ms	44ms	48ms	51ms	44ms	56ms	48ms	60ms	69ms
15	62ms	48ms	44ms	46ms	48ms	51ms	48ms	46ms	42ms	54ms	42ms	47ms
16	45ms	51ms	48ms	47ms	52ms	45ms	51ms	45ms	41ms	50ms	45ms	43ms
17	47ms	45ms	46ms	46ms	46ms	46ms	47ms	45ms	42ms	50ms	45ms	43ms
18	43ms	46ms	43ms	42ms	48ms	48ms	46ms	49ms	47ms	47ms	45ms	44ms
19	43ms	46ms	59ms	49ms	46ms	48ms	46ms	55ms	49ms	47ms	42ms	47ms
20	43ms	54ms	48ms	43ms	44ms	47ms	48ms	46ms	48ms	45ms	42ms	42ms

Testele 1-10 au costuri mici pe laturi, iar 11-20 aceleasi costuri insa de 10 ori mai mari.

4 Concluzie

Dupa analizarea algoritmilor putem observa ca in general este preferata folosirea Dijkstra, insa daca avem un caz specific am putea opta pentru una dintre cele doua optimizari prezentate.

References

1. Brilliant, https://brilliant.org/wiki/shortest-path-algorithms/. Accesat ultima dată pe 2 Noiembrie 2021
2. Geeksforgeeks, https://www.geeksforgeeks.org/applications-of-dijkstras-shortest-path-algorithm/. Accesat ultima dată pe 2 Noiembrie 2021
3. Geeksforgeeks, https://www.geeksforgeeks.org/dials-algorithm-optimized-dijkstra-for-small-range-weights/. Accesat ultima dată pe 2 Noiembrie 2021
4. Geeksforgeeks, https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/. Accesat ultima dată pe 2 Noiembrie 2021
5. DZone, https://dzone.com/articles/algorithm-week-shorted-path. Accesat ultima dată pe 2 Noiembrie 2021