Tema la disciplina Analiza Algoritmilor - Cel mai scurt drum - Etapa 1

Preda Diana, grupa 324CA
dianapreda.1305@stud.acs.upb.ro

Facultatea de Automatica si Calculatoare
Universitatea Politehnica din Bucuresti

1 Introducere

1.1 Descrierea problemei rezolvate

intr-un graf, orice drum este definit de o succesiune de muchii cu proprietatea ca, pentru oricare doua muchii consecutive din succesiune, nodul destinaţie al primei muchii este acelaşi cu nodul sursa al celei de-a doua muchii. Costul unui drum va fi definit ca suma costurilor muchiilor ce compun acel drum. Fie un nod sursa (S) şi un nod destinaţie (D). Pot exista mai multe drumuri de la S la D, iar drumul de cost minim de la S la D va fi cel cu costul cel mai mic dintre acestea. De asemenea, pot exista mai multe drumuri de cost minim de la S la D.

Aplicatii practice: Algoritmul de drum minim este folosit in aplicatii precum cele GPS (Waze, Google Maps, Moovit), determinand cel mai avantajos drum din punct de vedere al distantei si aglomeratiei; in animatii (Flip book animation) si in cadrul retelelor de comunicatie (trimiterea unui email).

1.2 Specificarea solutiilor alese

Vom alege urmatorii algoritmi care cauta drumurile de cost minim intre oricare 2 noduri: Dijkstra, Floyd-Warshall si Johnson.

Dijkstra Algoritmul Dijkstra se foloseste de o coada de prioritate, nodul radacina fiind ales de noi. Acesta nu functioneaza pentru grafuri cu circuite negative (in cazul in care exista circuite de cost negativ, costum minim pentru orice varf care apartine circuitului, ar putea fi considerat $-\infty$, iar algoritmul nu detecteaza aceste circuite).

Floyd-Warshall Algoritmul Floyd-Marshall foloseste o matrice in care sunt stocate costurile dintre noduri, iar daca nu exista muchie intre doua noduri atunci costul este considerat ∞. Rezultatul algoritmului Floyd-Warshall este o matrice N x N numita dist, iar valorea din matrice de la poziţia [i][j] va fi costul minim pentru drumul i-j.

Johnson Algoritmul Johnson se foloseste atat Dijkstra, cat si de Bellman-Ford. Acesta functioneaza pe greutati negative, dar nu pe cicluri cu greutati negative si este utilizat pentru grafuri rare.

1.3 Evaluarea solutiilor

Vom implementa algoritmii si apoi ii vom compara pe acelasi set de date in care se va urmari eficienta fiecaruia din punct de vedere al timpului de executie, al memoriei si al complexitatii.

Pentru validarea celor trei algoritmi vom folosi 20 de teste. Vom introduce cazuri in care nu functioneaza unul sau chiar toti algoritmii propusi (atunci cand in grafuri se afla cicluri negative). Vom avea minim un caz cu costuri negative (care va da un rezultat incorect pentru Dijkstra). Vom genera grafuri simple orientate ponderate atat dense, cat si rare, neconexe, aciclice sau fara muchii.

Vom rula fiecare dintre cei trei algoritmi si vom verifica daca avem aceleasi rezultate.

Distantele minime intre 2 noduri vor fi reprezentate in output printr-o matrice, unde indicii liniei reprezinta nodul destinatie, iar indicii coloanelor reprezinta nodul sursa.

2 Prezentarea solutiilor

2.1 Descrierea modului in care functioneaza algoritmii alesi

Fie graful G = (V, E), unde V este numarul de noduri, iar E e numarul de muchii. In Input va fi matricea de adiacenta a grafului G. Liniile sunt indexate cu i, iar coloanele cu j. In Output vor fi dispuse distantele dintre nodurile grafului sub forma de matrice. Daca nu exista un drum intre nodurile i si j, distanta va fi marcat ca INF.

Dijkstra Ne vom folosi de 2 vectori, unul de tip boolean pentru a vedea ce noduri mai avem de vizita si unul de tip int pentru a retine distantele gasite. Algoritmul se parcurge pentru fiecare nod din graf. Luam un nod i care nu a fost deja vizitat si il adaugam in lista de noduri vizitate. Parcurgem toate nodurile adiacente in adancime. Daca suma dintre costul muchiei dintre nodul i in care ne aflam si nodul adiacent j si distanta dintre nodul i in care ne aflam si sursa este mai mica decat valoarea determinata anterior ca fiind distanta minima, actualizam distanta minima. $dist[i] + graf[i][j] < dist[j]$

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

.

Figure 1: Pseudocod algoritm Dijkstra

Floyd-Warshall Vom initializa matricea solutiei la fel ca matricea graficului de intrare. Apoi vom actualiza matricea solutiei luand in considerare toate nodurile ca un varf intermediar. Alege unul cate unul toate nodurile si actualizeazam toate caile cele mai scurte, care includ varful ales ca un varf intermediar in calea cea mai scurta. Cand alegem nodul k ca nod intermediar, consideram deja nodurile 0, 1, 2, .. k-1 ca noduri intermediare. Pentru fiecare pereche (i, j) a nodurilor sursa si respectiv destinatie, exista doua cazuri posibile: k nu este un varf intermediar in cel mai scurt drum de la i la j. Pastram valoarea dist[i][j] asa cum este sau k este un varf intermediar in cel mai scurt drum de la i la j. Actualizam valoarea dist[i][j] ca dist[i][k] + dist[k][j] daca $dist[i][j] > dist[i][k] + dist[k][j]$

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

Figure 2: Pseudocod algoritm Floyd-Warshall

Johnson Fie graful dat G. Se adauga un nod s si se uneste cu toate nodurile grafului prin muchii de cost 0. Noul graf creat va fi G'. Rulam algoritmul Bellman-Ford pe G' cu s ca sursa. Distantele calculate de Bellman-Ford le denumim h[0], h[1],..., h[n - 1]. Daca gasim cicluri negative ne vom intoarce deoarece ciclul de greutate negativa nu poate fi create de noi noduri s, intrucat nu exista nici o margine la s. Recalculam costul muchiilor folosit formula: $cost_{original} + h[i]–h[j]$. Scoatem nodurile adaugate si rulam algoritmul lui Dijkstra pentru fiecare nod.

```
JOHNSON(G, w)
1  compute G', where G'.V = G.V ∪ {s},
       G'.E = G.E ∪ {(s, v) : v ∈ G.V}, and
       w(s, v) = 0 for all v ∈ G.V
2  if BELLMAN-FORD(G', w, s) == FALSE
3      print "the input graph contains a negative-weight cycle"
4  else for each vertex v ∈ G'.V
5          set h(v) to the value of δ(s, v)
               computed by the Bellman-Ford algorithm
6      for each edge (u, v) ∈ G'.E
7          ŵ(u, v) = w(u, v) + h(u) − h(v)
8      let D = (d_uv) be a new n × n matrix
9      for each vertex u ∈ G.V
10         run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
11         for each vertex v ∈ G.V
12             d_uv = δ̂(u, v) + h(v) − h(u)
13     return D
```

Figure 3: Pseudocod algoritm Johnson

2.2 Analiza complexitatii solutiilor.

Dijkstra Complexitatea algoritmului lui Dijkstra este data de implemntarea cozii de prioritati minima. Complexitatea algoritmului lui Dijkstra este O(V * $(E + V^2)$) = O(V^3).
Folosind un min-heap binar, complexitatea algoritmului poate fi redusa la O(V * (E + VlogV)) = O(VE + V^2logV).
Folosind Fibonacci heap, complexitatea algoritmului poate fi redusa la O(V * ((E + V) * logV)) = O(V^2logV + EV).

Floyd-Warshall Algoritmul Floyd-Warshall rezolva problema prin programare dinamica in complexitate $\theta(V^3)$.

Johnson Complexitatea algoritmului Johnson se bazeaza pe complexitatile algoritmilor Dijkstra si Bellman-Ford. Pasii principali ai algoritmului sunt algoritmul

Bellman-Ford apelat o singura data si algoritmul Dijkstra apelat de V ori. Complexitatea generala a algoritmului Bellman-Ford este O(VE), iar complexitatea generala a algoritmului Dijkstra este O(VLogV). Deci, complexitatea generala a algoritmului Johnson este O(V^2log V + VE).

2.3 Prezentarea principalelor avantaje si dezavantaje pentru solutiile luate in considerare.

Dijkstra

Avantaje Algoritmul Dijkstra este eficient din punct de vedere al complexitatii si este folosit in pratica.

Dezavantaje Algoritmul Dijkstra nu functioneaza pentru muchii de cost negativ, acesta conducand la grafuri aciclice si neputand uneori sa determine corect cel mai scurt drum. Algoritmul Dijkstra consuma inutil resurse in procesare.

Floyd-Warshall

Avantaje Algoritmul Floys-Warshall este usor de implementat si functioneaza pentru muchii de cost negativ.

Dezavantaje Algoritmul Floys-Warshall nu functioneaza pentru grafuri cu cicluri negative si are timpul de executie mai mare decat algoritmul Dijkstra.

Johnson

Avantaje Algoritmul Johnson este mai eficient pentru grafuri rare deoarece complexitatea sa depinde de numarul de muchii din grafic, in timp ce complexitatea algoritmul Floyd-Warshall nu. Algoritmul lui Johnson ruleaza in O(V^2log V + VE). Asadar, acesta va rula mai repede decat $O(V^3)$ care este timpul de executie pentru Floyd-Warshall.

Dezavantaje Algoritmul Johnson nu functioneaza pentru grafuri cu cicluri negative.

3 Evaluare

3.1 Descrierea modalitatii de construire a setului de teste folosite pentru validare.

Pentru a genera testele, am realizat un cod in Python ce genereaza in mod aleator liste de adiacenta.

Input Testele de input au urmatoarea structura: - prima linie va fi de forma: V E, unde V este numarul de noduri, iar E este numarul de muchii - urmatoarele E linii vor fi de forma: src dest weight, unde src e nodul sursa, dest este nodul destinatie si weight este costul muchiei

Output Distantele minime intre oricare doua noduri vor fi reprezentate sub forma unei matrice in care indicii liniei reprezinta nodul 'sursa', iar indicii coloanelor reprezinta nodul 'destinatie'.

Au fost generate 20 de teste cu diferite specificatii pentru a vedea cum se comporta cei 3 algoritmi.

Specificatii teste:
- testul 0: graf neconex
- testul 1: graf conex
- testul 2: graf ciclic
- testul 3: graf aciclic
- testul 4: graf fara muchii
- testul 5: graf rar
- testul 6: graf cu toate costurile egale
- testul 7: graf cu toate costurile negative
- testul 8: graf simplu complet
- testul 9: graf cu cicluri negative
- testul 10: graf cu costuri negative
- testele 11 - 13: graf cu un ciclu negativ
- testele 14 - 18: grafuri generate aleator
- testul 19: graf fara noduri si fara muchii

3.2 Mentionati specificatiile sistemului de calcul pe care ati rulat testele

Vom rula testele pe un procesor 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz cu 16.0 GB (15.8 GB utilizabili).

3.3 Ilustrarea, folosind grafice/tabele, a rezultatelor evaluarii solutiilor pe setul de teste.

Test	V	E	Dijkstra(s)	Floyd-Warshall(s)	Johnson(s)
0	7	10	0.000074	0.000071	0.000071
1	7	6	0.000091	0.00538	0.001134
2	10	10	0.000052	0.000052	0.000056
3	6	8	0.000070	0.000107	0.000082
4	453	0	0.000063	0.000063	0.000102
5	23	22	0.000068	0.000073	0.000067
6	30	56	0.000057	0.000082	0.000058
7	13	10	0.000069	0.000196	0.000349
8	5	6	0.000089	0.000067	0.000075
9	20	23	0.000058	0.000056	0.000054
10	60	17	0.000122	0.001976	0.004402
11	4	4	0.000076	0.000068	0.000103
12	29	36	0.000091	0.000089	0.000077
13	8	49	0.000117	0.000062	0.000065
14	20	23	0.350230	0.149869	0.353566
15	4	4	0.000086	0.000165	0.000175
16	35	42	0.000263	0.000161	0.000279
17	7	42	0.000060	0.000066	0.000085
18	5	10	0.000057	0.000063	0.000081
19	99	10	0.000069	0.000161	0.000154

Test	V	E	Dijkstra	Floyd-Warshall	Johnson
0	7	10	matrice de output	matrice de output	matrice de output
1	7	6	matrice de output	matrice de output	matrice de output
2	10	10	matrice de output	matrice de output	matrice de output
3	6	8	matrice de output	matrice de output	matrice de output
4	453	0	matrice de output	matrice de output	matrice de output
5	23	22	eroare	matrice de output	matrice de output
6	30	56	matrice de output	matrice de output	matrice de output
7	13	10	eroare	matrice de output	matrice de output
8	5	6	matrice de output	matrice de output	matrice de output
9	20	23	eroare	matrice de output	matrice de output
10	60	17	eroare	matrice de output	matrice de output
11	4	4	eroare	eroare	eroare
12	29	36	eroare	eroare	eroare
13	8	49	eroare	eroare	eroare
14	20	23	eroare	eroare	eroare
15	4	4	eroare	matrice de output	matrice de output
16	35	42	eroare	matrice de output	matrice de output
17	7	42	eroare	matrice de output	matrice de output
18	5	10	matrice de output	matrice de output	matrice de output
19	99	10	gol	gol	gol

.

3.4 Interpretarea, succinta, a valorilor obtinute pe teste.

Dupa cum se poate observa in tabelul de mai sus, valorile obtinute sunt de asteptat

4 Concluzii

4.1 Precizati, in urma analizei facute, cum ati aborda problema in practica; in ce situatii ati opta pentru una din solutiile alese.

Problema pe care incercam sa o rezolvam este gasirea celui mai mic drum intre doua noduri dintr-un graf ponderat dat in care costul poate sa fie negativ. Un factor important de diferentiere intre algoritmi este aspectul lor practic.
Algoritmul Dijkstra este utilizat in practica pentru rutarea protocolului IP, pentru a desemna un server de fisiere in RAM si pentru retelele de telefonice.
Algoritmul Floyd-Warshall poate fi implementat intr-un sistem distribuit, ceea ce il face potrivit pentru structuri de date, cum ar fi 'Graph of Graphs' (utilizat pentru harti).
Cea mai utilizata aplicatie a algoritmului Johnsons este 'Networking of Roads'. Utilizat pentru alegerea caii optime de trimitere a pachetelor de date.
De asemenea, este utilizat in aplicatiile destinate soferilor auto pentru a gasi un traseu optim din punct de vedere al distantei si al traficului. Acest algoritm este utilizat pe scara larga in scopuri logistice pentru a minimiza costurile de transport.

5 O selectie de referinte

[1] https://www.geeksforgeeks.org/johnsons-algorithm/ - *19/11/2022*
[2] https://brilliant.org/wiki/johnsons-algorithm/ - *19/11/2022*
[3] https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/ *19/11/2022*
[4] https://brilliant.org/wiki/johnsons-algorithm/ - *12/12/2022*
[5] https://ocw.cs.pub.ro/courses/sda-aa/laboratoare/09 - *19/11/2022*
[6] http://elf.cs.pub.ro/sda-ab/wiki/laboratoare/laborator-08 - 1*19/11/2022*
[7] https://tutoriale-pe.net/algoritmul-lui-dijkstra-c/ - *19/11/2022*
[8] https://www.myassignmenthelp.net/dijkstra-shortest-path-algorithm - *11/12/2022*
[9] https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/ - *11/12/2022*
[10] https://www.geeksforgeeks.org/johnsons-algorithm/ - *11/12/2022*
[11] https://www.scaler.com/topics/data-structures/johnsons-algorithm/

Sursa Figura 1
[12] https://stackoverflow.com/questions/50757074/does-the-dijkstra-algorythm-need-to-check-all-vertices -*11/12/2022*

Sursa Figura 2, Figura 3, Figura 4
[13] https://www.chegg.com/homework-help/questions-and-answers/code-johnsons-algorithm-using-pseudo-code–input-integer-v-100-designating-number-vertice-q6283981question-transcript - *11/12/2022*