

Calea minimă dintre două vârfuri la grafuri orientate

June 5, 2018

Student: Anghel Florina-Cătălina

Grupa: CR1.1

Specializarea: Calculatoare cu predare în limba română

Anul întâi

1 Introducere

1.1 Enunțul problemei

Implementați doi algoritmi care să determine calea minimă dintre două vârfuri dintr-un graf orientat.

Pentru a rezolva această problemă, am ales să implementez:

- Algoritmul lui Dijkstra
- Algoritmul Floyd-Warshall

1.2 Funcții folosite

Este un proiect modular, care conține 3 fișiere cu extensia .c și două fișiere de tip header.

În `matrix.c` am implementat:

- O funcție care să genereze în mod aleator numărul de vârfuri ale grafului
- O funcție pentru generarea unui număr aleator pe care o folosesc la generarea vârfului start, dar și a vârfului destinație
- O funcție pentru generarea matricei de adiacență
- O funcție pentru generarea matricei de cost pe baza celei de adiacență
- Două funcții pentru afișarea matricelor

În `shortest_length_path.c` am implementat:

- Funcția Dijkstra, de tip void
- Funcția Floyd-Warshall, de tip void
- O funcție `push_beginning`, pe care am folosit-o pentru a afișa calea în cazul algoritmului Floyd-Warshall, apelând-o pentru o stivă
- O funcție `pop_beginning_list` pentru a extrage elementele din stivă

În `main.c` se află funcția `main`.

Fișierele de tip header sunt `matrix.h` și `shortest_length_path.h`

2 Algoritmi folosiți

- Algoritmul lui Dijkstra
La acest algoritm, se memorează elementele din matricea de cost într-un vector denumit `distance` și se inițializează elementele vectorului `visited` cu 0, acest vector ilustrând dacă vârful respectiv a fost sau nu parcurs. Acest algoritm determină calea minimă pentru toate vârfurile, dar a fost

particularizat pentru a afișa doar calea minimă și distanța de la un vârf start până la unul destinație. Variabila locală count este folosită pentru a nu depăși numărul de vârfuri ale grafului.

Cât timp nu este depășit numărul de vârfuri, se calculează distanța minimă și se reține valoarea iteratorului în variabila next_node. Se testează dacă există o cale mai eficientă între vârfuri. Se testează dacă există o altă cale mai eficientă și dacă răspunsul este afirmativ, atunci se actualizează vectorul distanță și vectorul predecessor reține nextnode. Variabila count se incrementează pentru a arata că a mai fost parcurs un vârf. Apoi se afisează distanța și calea minimă dintre cele două vârfuri.

```

Dijkstra (adj , cost , n , start , dest)
1.   for i = 0, n - 1 execute
2.       distance[i] <- cost[start][i]
3.       predecessor[i] <- start
4.       visited[i] <- 0
5.   distance[start] <- 0
6.   visited[start] <- 1
7.   count <- 1
8.   while count < n - 1 execute
9.       min_dist <- inf
10.      for i = 0, n - 1 execute
11.          if distance[i] < min_dist and !visited[i] then
12.              min_dist <- distance[i]
13.              next_node <- i
14.          visited[next_node] <- 1
15.      for i = 0, n - 1 execute
16.          if !visited[i] then
17.              if min_dist + cost[next_node][i] < distance[i] then
18.                  distance[i] <- min_dist + cost[next_node][i]
19.                  predecessor[i] <- next_node
20.      count <- count + 1
21.      for i = 0, n - 1 execute
22.          if i != start then
23.              if i = dest then
24.                  write distance[i]
25.                  write i
26.                  j <- i
27.                  repeat
28.                      j <- predecessor[j]
29.                      write j
30.                  while j != start

```

- Algoritmul Floyd-Warshall

La acest algoritm, se inițializează matricea distanțelor cu elementele matricei de cost. Apoi se adaugă toate vârfurile ca vârfuri intermediare, se

iau toate vârfurile ca vârfuri sursă și apoi, se iau toate vârfurile ca vârfuri destinație. La fel ca și Dijkstra, algoritmul Floyd-Warshall determină distanța minimă pentru toate perechile de vârfuri din graf.

În cazul în care costul parcurgerii printr-un vârf intermediar e mai mic decât cel al parcurgerii directe, atunci la distanța de la acel moment de timp se adaugă distanța parcursă prin nodul intermediar și se reține acest varf, altfel se parcurge calea directă și se reține vârful parcurs. Apoi urmează afișarea distanței minime și a căii minime.

La acest algoritm, pentru a afișa calea minimă am construit o stivă în care să se rețină vârfurile intermediare. Prima dată se va afișa primul vârf și se va testa dacă `path[iterator_1][destination_node]` este diferit de -1, caz în care se afișează vârful destinație, -1 însemnând că nu există noduri intermediare.

```
Floyd-Warshall(cost_matrix, n, start, destination)
1. build the stack
2. for i = 0, n-1 execute
3.     for j = 0, n-1 execute
4.         path[i][j] <- -1
5.         dist[i][j] <- cost_matrix[i][j]
6. var <- 0
7. for i = 0, n-1 execute
8.     if dist[start][i] != inf
9.         var <- 1
10. if var = 0 then
11.     write "The start vertex has no connection"
12.     exit
13. for k = 0, n-1 execute
14.     for i = 0, n-1 execute
15.         for j = 0, n-1 execute
16.             for l = 0, n-1 execute
17.                 if dist[i][j] > dist[i][k] + dist[k][j] then
18.                     dist[i][j] <- dist[i][k] + dist[k][j]
19.                     path[i][j] <- k
20. i <- start
21. j <- destination
22. write start
23. while i != destination and path[i][destination] != -1 execute
24.     push.begining(head_stack, path[i][destination])
25.     j <- path[i][destination]
26.     aux <- j
27.     while path[i][j] != -1
28.         push.begining(head_stack, path[i][j])
29.         j <- path[i][j]
30.     if path[i][j] = -1 then
31.         i <- aux
```

```

32.         while head_stack->next != NULL execute
33.             write pop_begining_list(head_stack)
34.         j <- destination
35. write destination
36. write "Do you want to print the path matrix?"
37. write "Answer with Yes or No"
38. read answer
39. if strcmp(answer, "Yes") = 0 then
40.     for i = 0, n-1 execute
41.         for j = 0, n-1 execute
42.             write path[i][j]

```

3 Date experimentale (Generarea matricelor și a vârfurilor)

Pentru a genera numărul de noduri, am folosit funcția `rand()%`, apelând-o la început prin `srand((unsigned) (t))`. În funcția `main`, pentru a acoperi cazurile în care numărul de vârfuri generat aleator este 0 sau 1, am introdus anumite condiții, care determină ieșirea din program. Din cauza faptului că de cele mai multe ori vârful sursă și vârful destinație generate în mod aleator coincideau, am introdus o etichetă care îmi permite reluarea generării vârfului destinație și modificarea acestuia astfel încat el să fie diferit de cel sursă și să nu depășească numărul de vârfuri.

Pentru utilizarea eficientă a memoriei, matricele au fost alocate dinamic și inițializate cu 0 prin utilizarea funcției `calloc` pentru a scurta timpul de execuție. Tot cu acest scop, dar și datorită corectitudinii teoretice, indicii matricelor și număratoarea vârfurilor încep de la zero.

Matricea de adiacență are pe diagonala principală zero, ea nu este simetrică față de diagonala principală deoarece se tratează cazul în care graful este orientat. Dacă elementul de deasupra diagonalei principale este 1, atunci cel de pe poziția simetrică în funcție de diagonala principală va avea valoarea zero deoarece se dorește eliminarea intrării într-o buclă infinită. Însă dacă elementul de deasupra diagonalei principale este zero, atunci pentru elementul de pe poziția simetrică lui se va genera aleator. Matricea de cost este generată pe baza matricei de adiacență, având pe diagonala principală zero, iar acolo unde în matricea de adiacență este unu, se generează aleator un număr. În cazul în care în matricea de adiacență elementul are valoarea zero și nu se află pe diagonala principală, atunci în matricea de cost se pune infinit, pentru că nu s-a calculat încă distanța până la acel vârf prin vârfuri intermediare.

4 Complexitatea algoritmilor

Pentru a evidenția eficiența algoritmilor, am ales doi algoritmi care au complexități diferite. Primul algoritm, algoritmul lui Dijkstra este foarte eficient

Pentru algoritmul Dijkstra, timpul de execuție este egal cu $O(|E| + |V|^2)$, adică $O(|V|^2)$, unde E este numărul de muchii și V este numărul de vârfuri. Timpul este acesta pentru că este timpul maxim al unor bucle combinate, în acest caz, fiind vorba de o buclă for și una while. Deci timpul de execuție devine $O(|V|^2)$. La algoritmul Floyd-Warshall, timpul de execuție este dat de cele patru bucle for combinate, deci timpul este $\Theta(V^4)$.

5 Rezultate și concluzii

În fig. 1, este prezentat un caz de test pozitiv, adică un caz în care rezultatele de la cei doi algoritmi coincid.

[illegible]

6

În figura a doua, este un caz de test în care cele două căi nu coincid, dar acest test este pozitiv deoarece calea nu este unică, adică pot exista mai multe căi diferite care să unească două vârfuri și să aibă același cost.

```

Command Prompt - project
88 23 inf inf inf inf 18 inf inf inf inf inf inf inf inf 82 95 39 24 inf inf 34 inf 55 inf 69 inf inf 58 inf 38 inf inf inf inf inf inf
inf inf inf 0 inf 80 44 inf 39 inf 5 18 inf inf inf 48 inf inf inf
54 inf inf 47 inf inf 84 inf inf 32 33 inf 53 inf 92 inf inf inf inf 69 inf inf 30 inf inf 36 94 39 inf inf inf inf inf inf inf inf
inf inf 93 24 0 inf inf 33 inf inf inf inf inf inf inf inf inf inf 43
59 40 50 inf inf inf 55 inf inf inf 77 inf inf inf 73 inf inf inf 75 inf inf 6 inf inf inf inf inf inf 23 inf inf inf 4 inf inf
inf inf 32 inf inf 0 inf inf 35 inf 70 inf inf 47 inf inf inf 97 inf
inf 70 inf inf 4 inf inf inf inf inf inf inf inf inf inf 98 69 inf 80 inf 87 14 23 inf 33 inf inf inf 70 inf inf 39 inf 44 39 1
inf inf inf inf inf 0 63 39 inf inf inf inf 30 inf 43 inf 16 0
inf 48 81 inf 3 inf inf inf 3 inf inf inf inf inf inf inf inf inf 71 inf inf inf inf inf inf 95 inf 51 86 83 inf inf inf inf inf inf
19 69 76 inf inf inf 0 inf inf inf 4 inf 62 15 74 48 96 inf
inf inf inf inf inf inf inf inf 87 14 25 inf inf inf 91 inf 63 inf inf 98 inf inf inf inf 97 inf inf inf 56 63 inf inf inf inf 2
9 inf inf inf inf inf 48 inf inf inf 0 inf 57 inf inf 66 84 inf inf inf 60
42 85 inf inf inf inf 50 73 inf inf inf inf inf 24 94 inf 70 inf inf inf inf inf inf 52 23 50 inf 81 inf inf inf inf inf inf inf
85 inf 0 inf inf 61 inf 79 12 0 inf 93 0 2 51 64 inf inf 73
inf inf 48 inf 25 inf inf 34 inf inf inf 96 inf inf 73 inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf 42 92 inf inf inf
inf inf inf inf inf 50 inf 35 31 inf 42 0 83 72 99 67 19 80 inf inf
15 inf inf 22 inf inf inf inf inf 19 inf 39 inf inf 78 inf inf inf inf 99 53 inf 91 62 inf inf inf inf 3 inf inf inf 59 inf inf inf
inf 40 inf inf 2 inf inf inf 61 inf inf 0 49 85 inf 35 inf 8 inf
inf inf 40 inf inf 56 17 inf inf 70 7 13 66 inf inf 25 41 inf inf inf inf inf inf inf inf inf inf inf 66 8 inf inf inf 44 inf inf 1
inf 88 inf inf 21 inf inf 62 inf inf inf inf 0 inf inf 64 inf 96 75
inf inf inf inf inf 33 inf inf inf inf 23 40 78 inf inf inf inf inf 16 93 inf inf inf inf inf inf 38 inf 97 30 inf inf inf inf 1
inf inf inf inf 17 inf inf inf inf inf inf inf 29 0 76 inf 65 91 inf
inf inf inf inf inf inf inf inf 21 75 inf 11 inf inf inf 27 8 inf inf inf 72 inf 64 32 9 inf inf inf inf inf inf inf 27 inf inf
95 inf inf 14 54 81 inf inf inf inf inf 98 48 inf 0 inf 48 inf inf
inf 34 22 inf inf inf inf inf 7 66 inf 83 inf 85 99 inf 12 52 inf inf inf inf inf 70 71 32 inf inf inf inf inf inf inf inf inf
inf inf 67 inf 42 inf inf inf inf inf inf inf inf inf 0 87 69 92
inf inf inf inf inf inf inf inf 71 inf inf inf inf inf inf inf inf 35 85 inf inf inf 1 inf inf 51 51 inf 49 inf inf inf inf inf
inf inf 14 95 inf 21 inf 0 inf 64 inf inf 17 inf inf inf inf 0 74 inf
inf 70 inf 62 inf inf 44 inf inf inf 61 35 inf inf 10 19 inf inf inf inf inf inf inf inf inf inf 1 inf inf inf inf 3 inf inf
17 inf inf inf 7 inf inf inf inf inf inf inf inf inf inf inf 0 inf
inf inf 21 inf inf inf inf inf inf inf inf inf 85 25 inf inf inf inf inf inf inf inf inf inf 97 61 inf inf inf inf 51 inf inf
inf inf inf 16 15 inf inf 74 inf inf inf 65 inf inf inf inf 51 inf 0

The random generated start node is: 43
The random generated destination node is: 46
===== The Dijkstra's Algorithm =====
The distance between the source node and the node 46 is 28
The path is: 46 < 39 < 21 < 28 < 53 < 4 < 43
===== The Floyd-Marshall Algorithm =====
The path between the start vertex and the destination vertex is: 43 -> 8 -> 54 -> 39 -> 46
Do you want to print the path matrix generated by The Floyd-Marshall Algorithm?
Answer with Yes or No.

```

Figure 2: Acesta este un alt caz de test.

În figura 3 este prezentat un alt caz pozitiv.

În figura 4 este prezentat un caz de test negativ, în care cei doi algoritmi dau căi diferite. Mai exact, în această situație, algoritmul Floyd-Warshall afișează cu un vârf intermediar în minus.

```
Command Prompt
===== The Floyd-Marshall Algorithm =====
The path between the start vertex and the destination vertex is: 43 -> 8 -> 54 -> 39 -> 46
Do you want to print the path matrix generated by The Floyd-Marshall Algorithm?
Answer with Yes or No.
No

C:\Users\ADMIN\Documents\GitHub\shortest_path\project
The number of vertices is: 6

The adjacency matrix is:
0 1 0 1 0 1
0 0 1 1 0 1
0 0 0 1 1 0
0 0 0 0 0 1
0 0 0 1 0 1
0 0 1 0 0 0

The cost matrix is:
0 83 inf 58 inf 85
inf 0 73 31 inf 41
inf inf 0 52 57 inf
inf inf inf 0 inf 62
inf inf inf 48 0 78
inf inf 85 inf inf 0

The random generated start_node is: 4
The random generated destination_node is: 5
===== The Dijkstra's Algorithm =====
The distance between the source node and the node 5 is 78
The path is: 5 <- 4

===== The Floyd-Marshall Algorithm =====
The path between the start vertex and the destination vertex is: 4 -> 5
Do you want to print the path matrix generated by The Floyd-Marshall Algorithm?
Answer with Yes or No.
Yes
-1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1
-1 5 -1 -1 -1 -1
-1 -1 -1 2 2 -1

C:\Users\ADMIN\Documents\GitHub\shortest_path\project
```

Figure 3: Acesta este un alt caz de test.

```
Command Prompt - project
1 10 inf inf inf inf inf inf inf inf inf 92 inf inf inf 58 inf inf inf inf 98 inf 0 inf inf inf inf 26 inf inf inf 28
inf inf 53 inf 93 43 inf inf inf inf inf inf inf 95 inf 8 inf inf inf 16 inf 92 03 inf 58 inf inf inf inf 58 94 inf inf
inf inf inf 54 inf inf inf inf inf inf inf 49 inf inf inf inf inf inf 46 inf inf inf 6 0 56 inf 84 inf 7 96 inf 28
inf inf 11 70 inf inf 3 80 inf inf 51 inf inf 45 inf inf inf inf inf inf 11 97 inf inf 91 inf inf inf inf 26 inf inf 65 inf inf
inf inf inf inf inf inf inf 51 64 inf inf inf 5 inf inf inf inf inf inf inf 98 inf inf inf 0 inf 99 inf 79 10 65 inf
inf inf inf inf inf 39 inf 12 inf inf inf 6 inf 91 inf inf inf 20 inf inf inf inf inf inf inf inf inf inf 39 80 inf inf
inf inf inf inf inf inf inf inf inf inf inf inf 24 inf inf inf 48 inf 92 inf inf inf inf 67 0 inf 59 inf 93 65 42
inf inf 33 inf inf inf inf inf inf inf inf inf 90 inf 66 inf 79 6 inf inf inf inf inf inf inf inf 49 inf 20 51 12 inf inf inf inf
inf 53 inf inf inf inf inf 11 inf inf inf inf inf inf inf inf inf 82 inf 28 inf inf inf inf 0 0 81 inf 94 14
inf inf inf inf inf inf inf 72 inf 32 inf 99 1 inf inf inf 89 inf inf inf inf 37 inf inf inf 0 inf inf inf inf 79 inf 74 1 inf inf
inf inf 28 inf 62 inf inf 25 inf 24 inf 77 inf 63 inf inf 84 inf inf inf inf 73 inf inf inf inf inf 0 11 inf inf inf
inf inf inf inf inf inf inf inf inf 26 98 89 inf 69 50 inf inf inf inf inf inf inf inf inf inf inf inf 8 81 inf inf 86 inf 0 inf
inf 76 45 0 inf 28 inf inf inf 75 inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf 0 29 inf inf
inf inf inf 35 11 41 89 inf inf inf inf 81 inf inf inf inf inf inf 7 inf inf inf inf inf inf inf inf 95 88 inf inf inf inf inf
inf 93 inf 90 inf inf inf 14 inf inf 00 inf inf 62 inf inf inf inf inf 55 inf inf inf inf inf inf 74 inf 0 inf 94
inf 47 inf 30 inf inf 81 1 50 inf inf inf inf inf inf inf inf inf inf 87 inf inf inf inf inf inf inf inf inf 14 inf 77 1
inf inf 11 inf inf inf inf 18 inf 7 inf 37 inf inf inf inf 73 74 inf 48 inf inf inf inf 52 inf inf inf 8 41 45 0 10
inf 95 18 inf 3 inf inf inf inf inf inf inf 34 inf inf inf inf 49 inf inf inf inf 20 inf inf inf inf 99 inf 16 inf inf inf 10 inf
27 inf 19 inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf 62 inf inf 16 78 inf inf 0

The random generated start_node is: 11
The random generated destination_node is: 37
===== The Dijkstra's Algorithm =====
The distance between the source node and the node 37 is 19
The path is: 37 <- 16 <- 61 <- 47 <- 59 <- 11
===== The Floyd-Marshall Algorithm =====
The path between the start vertex and the destination vertex is: 11 -> 59 -> 61 -> 16 -> 37
Do you want to print the path matrix generated by The Floyd-Marshall Algorithm?
Answer with Yes or No.
No
```

Figure 4: Acesta este un caz de test negativ.

În concluzie, în urma realizării acestui proiect, mi-am îmbogățit cunoștințele legate de proiectarea algoritmilor, de compilarea în linie de comandă, de utilizarea unui Makefile și am exersat lucrul cu aplicațiile Github, Doxygen și ShareLaTeX.

6 Referințe

- Wikipedia: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
- <https://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/>
- <https://www.geeksforgeeks.org/dynamic-programming-set-16-floyd-warshall-algorithm/>
- Wikipedia: https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm