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 Enunţul problemei

Implementaţi 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 Funcţii folosite

Este un proiect modular, care contine 3 fişiere cu extensia .c si două fişiere de tip header.

În matrix.c am implementat:

- O funcţie care să genereze în mod aleator numarul de vârfuri ale grafului
- O funcţie pentru generarea unui numar aleator pe care o folosesc la generarea vârfului start, dar şi a vărfului destinaţie
- O funcţie pentru generarea matricei de adiacenţă
- O funcţie pentru generarea matricei de cost pe baza celei de adiacenţă
- Două funcţii pentru afişarea matricelor

In shortest_length_path.c am implementat:

- Funcţia Dijkstra, de tip void
- Funcţia Floyd-Warshall, de tip void
- O funcţie push_beginning, pe care am folosit-o pentru a afişa calea în cazul algoritmului Floyd-Warshall, apelând-o pentru o stivă
- O funcţie pop_beginning_list pentru a extrage elementele din stivă

In main.c se află funcţia main.
Fişierele de tip header sunt matrix.h şi shortest_length_path.h

2 Algoritmi folosiţi

- Algoritmul lui Dijkstra
  La acest algoritm, se memorează elementele din matricea de cost într-un vector denumit distance şi se iniţializează 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 afişa doar calea minimă şi distanţa de la un vârf start până la unul destinaţie. Variabila locală count este folosită pentru a nu depăşi numărul de vârfuri ale grafului.
Cât timp nu este depăşit numărul de vârfuri, se calculează distanţa minimă şi se reţine 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 şi dacă răspunsul este afirmativ, atunci se actualizează vectorul distanţă şi vectorul predecessor reţine nextnode.Variabila count se incrementează pentru a arata că a mai fost parcurs un vârf. Apoi se afisează distanţa şi calea minimă dintre cele două vârfuri.

```
Dijkstra(adj,cost,n,start,dest)
      for i = 0, n - 1 execute
          distance[i] <- cost[start][i]
          predecessor[i] <- start
          visited[i] <- 0
      distance[start] <- 0
      visited[start] <- 1
      count <- 1
      while count < n - 1 execute
          min_dist <- inf
           for i = 0, n - 1 execute
               if distance[i] < min_dist and !visited[i] then
                   min_dist <- distance[i]
                   next_node <- i
           visited[next_node] <- 1
           for i = 0, n - 1 execute
             if !visited[i] then
               if min_dist + cost[next_node][i] < distance[i] then
                   distance[i] <- min_dist + cost[next_node][[i]
                   predecessor[i] <- next_node
       count <- count + 1
       for i = 0, n - 1 execute
            if i != start then
                if i = dest then
                   write distance[i]
                   write i
                   j <- i
                   repeat
                       j <- predecessor[j]
                       write j
                   while j != start
```

- Algoritmul Floyd-Warshall
  La acest algoritm, se iniţializează matricea distanţelor cu elementele matricei de cost. Apoi se adaugă toate vârfurile ca vârfuri intermediare, se

iau toate vârfurile ca vârfuri sursă şi apoi, se iau toate vârfurile ca vârfuri destinaţie. La fel ca şi Dijkstra, algoritmul Floyd-Warshall determină distanţa minimă pentru toete 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 distanţa de la acel moment de timp se adaugă distanţa parcursă prin nodul intermediar şi se reţine acest varf, altfel se parcurge calea directă şi se reţine vârful parcurs. Apoi urmează afişarea distanţei minime şi a căii minime.
La acest algoritm, pentru a afişa calea minimă am construit o stivă în care să se reţină vârfurile intermediare. Prima dată se va afişa primul vârf şi se va testa dacă path[iterator_1][destination_node] este diferit de -1, caz în care se afişează vârful destinaţie, -1 însemnând că nu există noduri intermediare.

```
Floyd-Warshall(cost_matrix, n, start, destination)
build the stack
for i = 0, n-1 execute
     for j = 0, n-1 execute
          path[i][j] <- -1
          dist[i][j] <- cost_matrix[i][j]
var <- 0
for i = 0, n-1 execute
     if dist[start][i] != inf
          var <- 1
if var = 0 then
     write "The start vertex has no connection"
     exit
for k = 0, n-1 execute
     for i = 0, n-1 execute
          for j = 0, n-1 execute
               for l = 0, n-1 execute
                    if dist[i][j] > dist[i][k]+dist[k][j] then
                         dist[i][j] <- dist[i][k]+ dist[k][j]
                         path[i][j] <- k
i <- start
j <- destination
write start
while i != destination and path[i][destination] != -1 execute
     push_begining(head_stack, path[i][destination])
     j <- path[i][destination]
     aux <- j
     while path[i][j] != -1
          push_begining(head_stack, path[i][j])
          j <- path[i][j]
     if path[i][j] = -1 then
          i <- aux
```

```
          while head_stack->next != NULL execute
              write pop_begining_list(head_stack)
          j <- destination
write destination
write "Do you want to print the path matrix?"
write "Answer with Yes or No"
read answer
if strcmp(answer, "Yes") = 0 then
    for i = 0, n-1 execute
        for j = 0, n-1 execute
              write path[i][j]
```

3 Date experimentale (Generarea matricelor şi a vârfurilor)

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

Pentru utilizarea eficientă a memoriei, matricele au fost alocate dinamic si iniţializate cu 0 prin utilizarea funcţiei calloc pentru a scurta timpul de execuţie. Tot cu acest scop, dar şi datorită corectitudinii teoretice, indicii matricelor şi număratoarea vârfurilor încep de la zero.

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

4 Complexitatea algoritmilor

Pentru a evidenţia eficienţa algoritmilor, am ales doi algoritmi care au complexităti diferite. Primul algoritm, algoritmul lui Dijkstra este foarte eficient

din punctul de vedere al timpului necesar execuţiei sale.
Pentru algoritmul Dijkstra, timpul de execuţie este egal cu $O(|E| + |V|^2)$, adică $O(|V|^2)$, unde E este numărul de muchii şi 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 şi una while. Deci timpul de execuţie devine $O(|V|^2)$.
La algoritmul Floyd-Warshall, timpul de execuţie este dat de cele patru bucle for combinate, deci timpul este $\Theta(V^4)$.
Lăsând la o parte aceşti algoritmi, o altă secvenă care poate încetini programul este cea de regenerare a vârfului destinaţie deoarece se verifică dacă este diferit de cel start şi dacă se află în intervalul [0, V - 1], unde V este numărul de vârfuri.

5 Rezultate şi concluzii

După mai multe teste, am constatat că în puţine cazuri, cei doi algoritmi afişează căi diferite deoarece există mai multe căi care au acelaşi cost şi leagă aceleaşi vârfuri. Însă există câteva teste negative, adică teste în care cele două căi afişate diferă, problema fiind la cel de-al doilea algoritm pentru că exclude vârfuri (de exemplu, figura a patra ilustrează un astfel de test). După mai multe teste, am constatat că timpul de execuţie este de câteva secunde, dar se poate extinde şi la câteva minute deoarece, atunci când se generează vârful destinaţie, este testat dacă este diferit de cel de start şi dacă se află în intervalul de definiţie şi existenţă al vârfurilor, în caz contrar, fiind regenerat.
În fig. 1, este prezentat un caz de test pozitiv, adică un caz în care rezultatele de la cei doi algoritmi coincid.

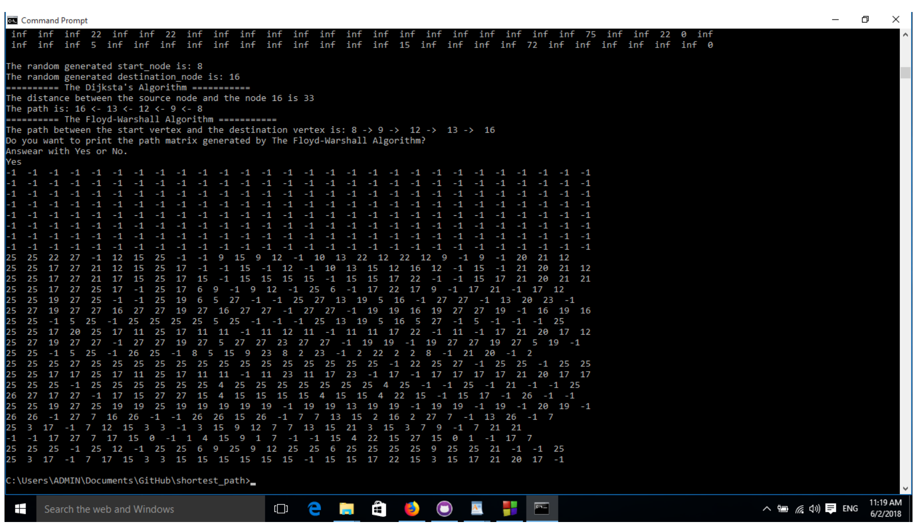

Figure 1: Acesta este un caz de test.

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 şi să aibă acelaşi cost.

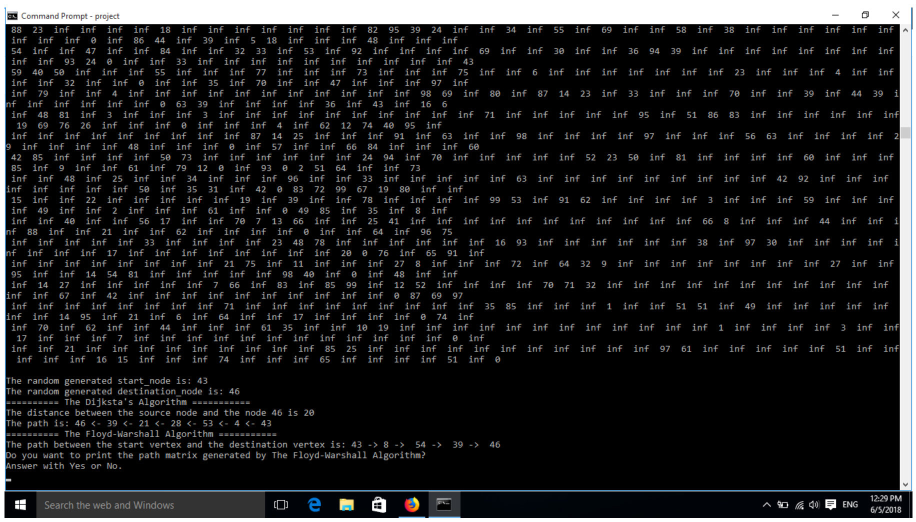

Figure 2: Acesta este un alt caz de test.

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 situaţie, algoritmul Floyd-Warshall afişează cu un vârf intermediar în minus.

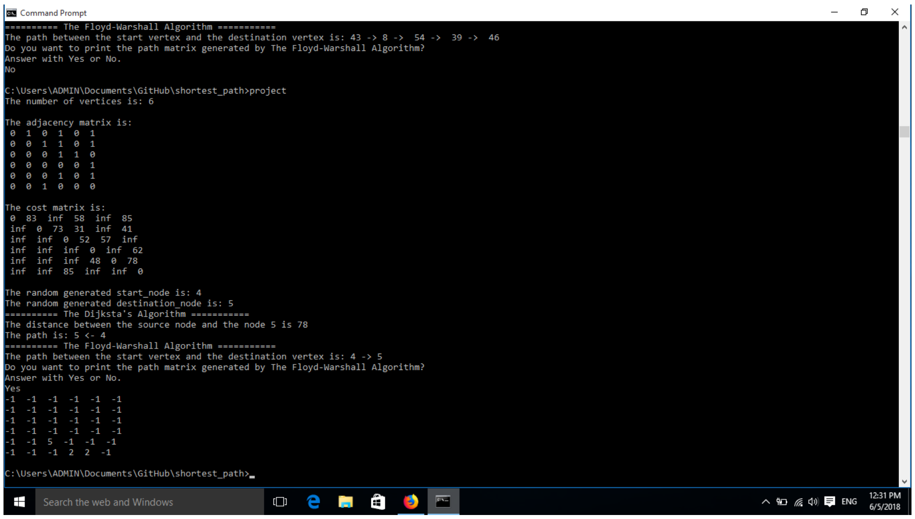

Figure 3: Acesta este un alt caz de test.

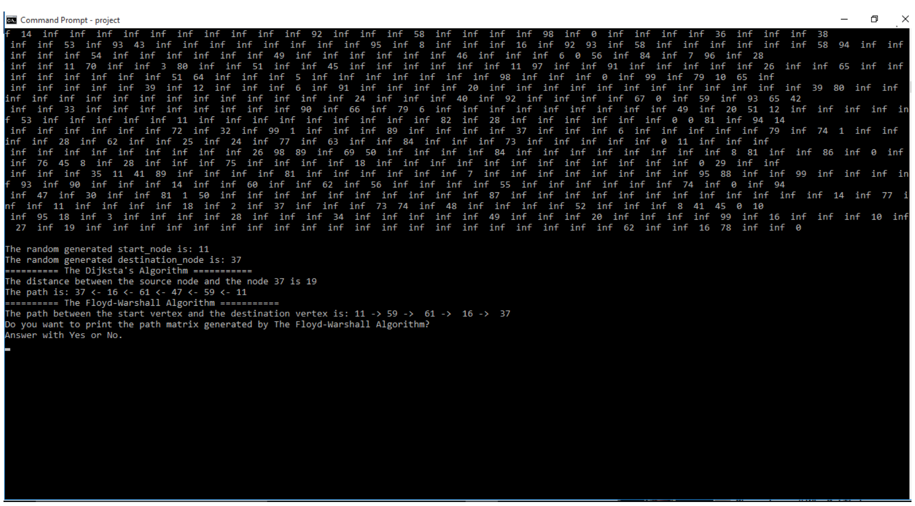

Figure 4: Acesta este un caz de test negativ.

În concluzie, în urma realizării acestui proiect, mi-am îmbogăţit cunoştinţele legate de proiectarea algoritmilor, de compilarea în linie de comandă, de utilizarea unui Makefile şi am exersat lucrul cu aplicaţiile Github, Doxygen şi ShareLaTeX.

6 Referinţe

- 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