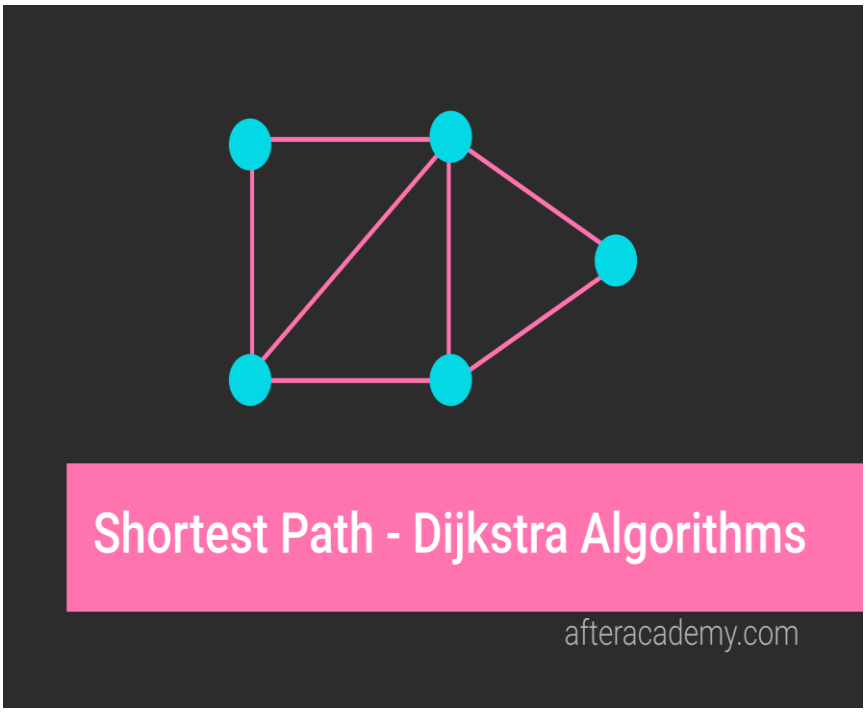
Dijkstra Implementation – Project

APD



Student: Bucă-Ghebaură Elizabetha Alexandrina

Grupa: 3.1 A

Specializarea: Calculatoare Romana

Pentru acest proiect am ales sa fac 3 implementari ale algoritmului Dijkstra, si anume:

- Implementare standard, in C++
- Implementare folosind MPI, in C++
- Implementare folosind Neo4j

1. Implementare Standard

Aici am implementat varianta clasica a algoritmului Dijkstra in C++.

Clasele pentru rezolvarea problemei sunt:

- EdgeNode
- Graph

Calcularea distantelor se face prin metoda *dijkstra_shorthest_path()*.

```
87  void dijkstra_shortest_path(Graph* g, vector<int> &parent, vector<int> &distance, int start) {
88
89      vector<bool> discovered;
90      EdgeNode* curr;
91      int v_curr;
92      int v_neighbor;
93      int weight;
94      int smallest_dist;
95
96      init_vars(discovered, distance, parent);
97
98      distance[start] = 0;
99      v_curr = start;
100
101     while (discovered[v_curr] == false) {
102
103         discovered[v_curr] = true;
104         curr = g->edges[v_curr];
105
106         while (curr != NULL) {
107
108             v_neighbor = curr->key;
109             weight = curr->weight;
110
111             if ((distance[v_curr] + weight) < distance[v_neighbor]) {
112                 distance[v_neighbor] = distance[v_curr] + weight;
113                 parent[v_neighbor] = v_curr;
114             }
115             curr = curr->next;
116         }
117
118         //set the next current vertex to the vertex with the smallest distance
119         smallest_dist = std::numeric_limits<int>::max();
120         for (int i = 0; i < (noNodes); i++) {
121             if (!discovered[i] && (distance[i] < smallest_dist)) {
122                 v_curr = i;
123                 smallest_dist = distance[i];
124             }
125         }
126     }
127 }
```

Iar aici afisam cel mai scurt drum si distantele de la un varf de start la un altul.

```
129  void print_shortest_path(int v, vector<int> &parent) {
130      if (v >= 0 && v < (noNodes) && parent[v] != -1) {
131          cout << parent[v] << " ";
132          print_shortest_path(parent[v], parent);
133      }
134  }
135
136  void print_distances(int start, vector<int> &distance) {
137      for (int i = 0; i < (noNodes); i++) {
138              cout << "Shortest distance from " << start << " to " << i << " is: " << distance[i] << endl;
139
140      }
141  }
```

In program, aici se face inserarea muchiilor (insert_edge) si afisarea vecinilor varfului respectiv.

```
55  void Graph::insert_edge(int x, int y, int weight, bool directed) {
56      if (x > 0 && x < (noNodes) && y > 0 && y < (noNodes)) {
57          EdgeNode* edge = new EdgeNode(y, weight);
58          edge->next = this->edges[x];
59          this->edges[x] = edge;
60          if (!directed) {
61              insert_edge(y, x, weight, true);
62          }
63      }
64  }
65
66  void Graph::print() {
67      for (int v = 0; v < (noNodes); v++) {
68          if (this->edges[v] != NULL) {
69              cout << "Vertex " << v << " has neighbors: " << endl;
70              EdgeNode* curr = this->edges[v];
71              while (curr != NULL) {
72                  cout << curr->key << endl;
73                  curr = curr->next;
74              }
75          }
76      }
77  }
```

Algoritmul poate fi rulat pe mai multe teste, iar aici am facut implementarea acestei parti:

```
149  int main() {
150
151      string testFile = "test{i}.in";
152      for (int i = 0; i <= 9; i++) {
153
154          in_file.open("test" + int_to_string(i + 1));
155
156          int nrNoduri, nrMuchii;
157          in_file >> nrNoduri >> nrMuchii;
158
159          noNodes = nrNoduri;
160
161          Graph* g = new Graph(false);
162          vector<int> parent;
163          vector<int> distance;
164
165          int start = 1;
166
167          int a, b, c;
168          for (int muc = 0; muc < nrMuchii; muc++) {
169              in_file >> a >> b >> c;
170              g->insert_edge(a, b, c, false);
171          }
172
173          dijkstra_shortest_path(g, parent, distance, start);
174
175          print_distances(start, distance);
176
177          cout << endl << endl << "------------------------------ TEST: " << i + 1 << " finished----------------------" << endl << endl;
178          delete g;
179          in_file.close();
180      }
181
182      return 0;
183  }
```

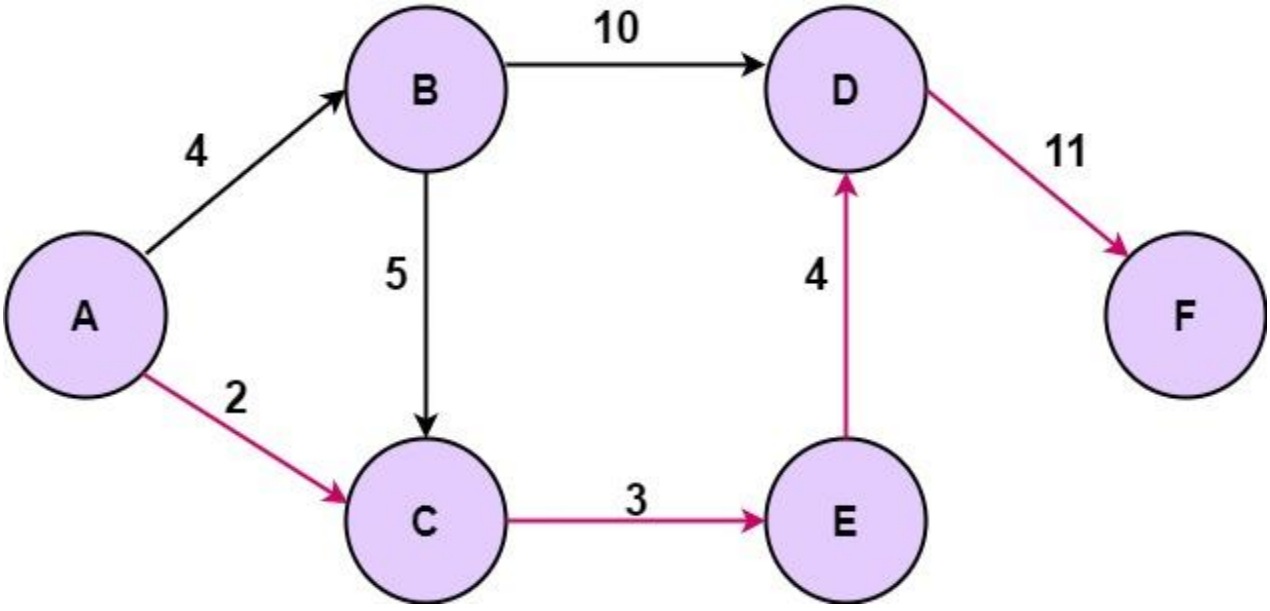
Exemplu de rulare a programului:

```
Shortest distance from 1 to 51 is: 7
Shortest distance from 1 to 52 is: 4
Shortest distance from 1 to 53 is: 16
Shortest distance from 1 to 54 is: 10
Shortest distance from 1 to 55 is: 10
Shortest distance from 1 to 56 is: 21


-------------------------------- TEST: 2 finished-----------------------

Shortest distance from 1 to 0 is: 2147483647
Shortest distance from 1 to 1 is: 0
Shortest distance from 1 to 2 is: 15
Shortest distance from 1 to 3 is: 19
Shortest distance from 1 to 4 is: 15
Shortest distance from 1 to 5 is: 9
Shortest distance from 1 to 6 is: 10
Shortest distance from 1 to 7 is: 22
Shortest distance from 1 to 8 is: 6
Shortest distance from 1 to 9 is: 25
Shortest distance from 1 to 10 is: 14
Shortest distance from 1 to 11 is: 2
Shortest distance from 1 to 12 is: 20
Shortest distance from 1 to 13 is: 19
Shortest distance from 1 to 14 is: 19
Shortest distance from 1 to 15 is: 34
Shortest distance from 1 to 16 is: 9
Shortest distance from 1 to 17 is: 15
Shortest distance from 1 to 18 is: 21
Shortest distance from 1 to 19 is: 10
Shortest distance from 1 to 20 is: 9
Shortest distance from 1 to 21 is: 12
Shortest distance from 1 to 22 is: 9
Shortest distance from 1 to 23 is: 20


-------------------------------- TEST: 3 finished-----------------------

Shortest distance from 1 to 0 is: 2147483647
Shortest distance from 1 to 1 is: 0
Shortest distance from 1 to 2 is: 11
Shortest distance from 1 to 3 is: 16
Shortest distance from 1 to 4 is: 15
Shortest distance from 1 to 5 is: 15
Shortest distance from 1 to 6 is: 12
```

2. Implementare cu MPI – C++

Aceasta este o implementare paralela a algoritmului Dijkstra, cu cea mai scurta cale, pentru un graf directionat ponderat dat ca o matrice adiacenta.

Algoritmul lui Dijkstra gaseste calea cea mai scurta de la o sursa la orice alt varf.



In sursa MpiDijkstra.cpp am introdus un exemplu de rulare a celei mai scurte cai de la A la orice alt varf din graful de mai sus.

Aici, sagetile colorate reprezinta calea cea mai scurta de la A la F. In program, A, B, C, D, E, F corespund lui 0, 1, 2, 3, 4 si 5.

$0 \longrightarrow v$ arata ca calea calculata catre F(5) este de lungime 20.

In program, avem urmatoarele conditii:

- n = numarul de varfuri
- p = numarul de procese
- p se imparte egal la n (p evenly divides n)

Proces 0: Varfurile 0, 1, ..., (n/p) – 1

Proces 1: Varfurile (n/p), (n/p) + 1, ..., 2 * (n/p) – 1

.....

Proces p – 1: Varfurile (p – 1) * (n/p), (p – 1) * (n/p) + 1, ..., p * (n/p) – 1 = n – 1

Daca luam in considerare graful sub forma de matrice de adiacenta, unde indicele [u][v] are distanta de la u la v care este „infinit" daca nu exista muchie intre u si v, atunci am dori sa calculam calea cea mai scurta de la A la F cu 3 procese, iar matricea de adiacenta devine:

Proces 0: va avea coloana 0 si 1 (A si B)

Proces 1: va avea coloana 2 si 3 (C si D)

Proces 2: va avea coloana 4 si 5 (E si F)

Matricea de adiacenta:

u, v	A	B	C	D	E	F
A	0	4	2	Infinit	Infinit	infinit
B	Infinit	0	5	10	Infinit	Infinit
C	Infinit	Infinit	0	Infinit	3	Infinit
D	Infinit	Infinit	Infinit	0	Infinit	11
E	Infinit	Infinit	Infinit	4	0	Infinit
F	Infinit	Infinit	Infinit	Infinit	Infinit	0

Facand o partitie pe coloana a matricei, fiecare proces devine responsabil pentru toate muchiile care vin la varfurile atribuite lor.

Fiecare proces verifica apoi distanta minima locala de la varful sursa globala la varfurile atribuite. Acum, unul dintre procese a gasit o noua distanta minima globala fata de varful sursa. Acest varf nu va mai fi vizitat.

Toate procesele isi actualizeaza distanta locala de la sursa la acest varf global nou gasit verificand daca distanta caii de la sursa la varful global pana la varful atribuit este mai mica decat distanta directa la varful atribuit de la sursa.

Aceasta procedura se repeta de n – 1 ori si dupa finalizare va fi calculata distanta minima de la varful sursa la toate celelalte varfuri.

Exemplu a primei iteratii:

Proces 0 isi gaseste distanta minima locala 4 de la 0 la (local) 1 ⟶ (global) 1

Proces 1 isi gaseste distanta minima locala 2 de la 0 la (local) 2 ⟶ (global) 2

Proces 2 isi gaseste distanta minima locala *Infinit* deoarece nu exista muchie de la 0 la vreunul dintre varfurile lui atribuite (0 la (global) 4) si (0 la (global) 5).

```
285  void Print_dists(int global_dist[], int n) { //afiseaza lungimea celei mai scurte cai de la 0 la fiecare varf
286      int v;
287      //n -> nr de varfuri
288      //dist -> distantele de la 0 la fiecare varf v
289      //dist[v] -> lungimea celui mai scurt drum de la 0 -> v
290      printf("  v    dist 0->v\n");
291      printf("----   ---------\n");
292
293      for (v = 1; v < n; v++) {
294          if (global_dist[v] == INFINITY) {
295              printf("%3d       %5s\n", v, "inf");
296          }
297          else
298              printf("%3d       %4d\n", v, global_dist[v]);
299      }
300      printf("\n");
301  }
```

Algoritmul determina ca procesul 1 a gasit varful minim global, astfel incat acesta este marcat ca si vizitat. Acum, fiecare proces verifica daca distanta pana la varfurile atribuite poate fi actualizata.

Procesul 2 observa ca exista o noua cale mai scurta catre varful 4 (global) alocat (adica varful E din graful de mai sus) de la sursa 0. Aceasta noua distanta este calculata prin adaugarea distantei caii la minimul global si adaugand costul caii de la acel minim global la varful 4 alocat.

Distanta de la minimul global la varful 4 este 3, asa ca distanta totala de la 0 la 4 este acum 2 + 3 adica 5.

Distanta este stocata local la procesul 2 si va fi utilizata in descoperirile ulterioare ale distantelor minime globale.

Fiecare „runda” marcheaza un varf, ceea ce face ca algoritmul sa se incheie.

```
303  void Print_paths(int global_pred[], int n) { //afiseaza cel mai scurt drum de la 0 la fiecare varf
304      int v, w, * path, count, i;
305      //n -> nr de varfuri
306      //pred -> lista de predecesori
307      //pred[v] = u daca u il precede pe v pe cel mai mic drum de la 0 -> v
308      path = (int*)malloc(n * sizeof(int));
309
310      printf("  v     Path 0->v\n");
311      printf("----    ---------\n");
312      for (v = 1; v < n; v++) {
313          printf("%3d:    ", v);
314          count = 0;
315          w = v;
316          while (w != 0) {
317              path[count] = w;
318              count++;
319              w = global_pred[w];
320          }
321          printf("0 ");
322          for (i = count - 1; i >= 0; i--)
323              printf("%d ", path[i]);
324          printf("\n");
325      }
326
327      free(path);
328  }
```

Rezultatul din consola dupa rularea programului:

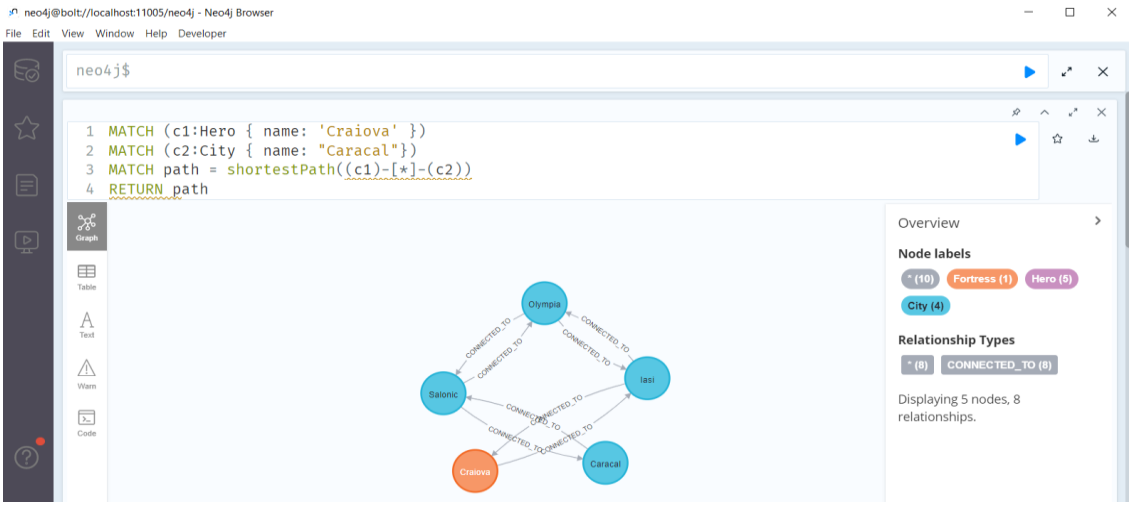
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Beti\Documents\GitHub\Dijkstra_APD\Mpi Dijkstra\MpiDijkstra\Debug>mpiexec -n 6 MpiDijkstra.exe
6
0 4 2 1000000 1000000 1000000
1000000 0 5 10 1000000 1000000
1000000 1000000 0 1000000 3 1000000
1000000 1000000 1000000 0 1000000 11
1000000 1000000 1000000 4 0 1000000
1000000 1000000 99 1000000 1000000 0
  v    dist 0->v
----   ---------
  1          4
  2          2
  3          9
  4          5
  5         20

  v     Path 0->v
----    ---------
  1:    0 1
  2:    0 2
  3:    0 2 4 3
  4:    0 2 4
  5:    0 2 4 3 5
```

3. Implementare folosind Neo4j

Am creat mai intai orase Hero cu conexiuni catre cele mai apropiate orase in ambele directii.

Nodurile reprezinta orasele situate.

Am gasit apoi calea cea mai scurta dupa nodurile conectate, si am introdus doua valori: Craiova si Caracal.



Am incercat apoi sa setez un nod de inceput si un nod de sfarsit, un tip de relatie si o proprietate pentru a le folosi ca pondere/cost a relatiilor. Astfel, am fi putut obtine si costul total al „calatoriei” si ar fi trebuit sa rezulte distanta totala intre cele 2 orase.

Nu am reusit sa rulez aceasta comanda din cauza unor erori.