Tehnici de Optimizare

Optimizarea consumului de combustibil al unui vehicul autonom

Proiect

Emma Ganciu

Victor Ungureanu

Liuta Adin-Robert

Dumitru Razvan-Marian

I. Scopul

Scopul proiectului este de a implementa algoritmi in MATLAB ref. la optimizarea consumului de combustibil al unui vehicul autonom urmand puncte de interes prestabilite. Acesti algoritmi sunt relevanti in cazul transportului, planificarea rutelor (atat industriale - roboti in fabrica – cat si cele ref. la transportul traditional), turism, etc. Cu scopul final de a scadea costul de energie a vehiculelor.

Pentru indeplinirea acestui obiectiv am urmat urmatorii pasi:

a) Am identificat structurile de date relevante pentru indeplinirea obiectivului, in acest caz am identificat ca relevante clasele: Graph.m(stabileste structura grafului si algoritmii de calculare relevanti) si Cost.m(clasa cu ajutorul careia se calculeaza costul de combustibil intre doua puncte de interes).
b) Algoritmii necesari au fost identificati si construiti in C++ - sunt algoritmi cu care ne-am familiarizat si in cadrul materiei „Algoritmica Grafurilor”. Au fost utilizati AStar(A*) cu un element de euristica – distanta Manhattan, Djikstra si Algoritmul de Cautare Bruta (ineficient).
c) Am adaptat codul algoritmilor in functii MATLAB specifice.
d) Fisierul de test a fost alcatuit, iar astfel am putut verifica algoritmii relevanti.

II. Structura de Date

Cost.m

Clasa Cost.m contine 4 proprietati: distanta, viteza, greutate si tip_drum. Acestea sunt necesare pentru a calcula un cost real de energie a vehiculului intre doua puncte de interes. Prin acest

mod se arata cum factorii de influenta (cele 4 proprietati) au impact asupra rezultatului cautarii drumului cu un cost optim.

$$E_{consum}=(\alpha D)+(\beta V)+(\gamma G)+(\delta_T D)$$

D – Distanta

V – Greutate

G – Greutate

TD – Tip_drum

Functia getConsumEnergie returneaza consumul de energie luand in considerare acesti factori de interes.

```
        % Calculeaza consumul de energie
        function consum_energie = getConsumEnergie(obj)
            % Formula simplificată pentru consumul de energie

            % Parametri de bază
            consum_baza = 0.1;  % kWh/km pentru un vehicul standard
            factor_greutate = obj.greutate / 1500.0;  % Raport față de greutatea standard

            % Factori care afectează consumul
            factor_viteza = 1.0 + 0.01 * abs(obj.viteza - 60);  % Consum optim la 60 km/h
            factor_drum = obj.tip_drum;  % Poate fi extins pentru diferite tipuri de drum

            % Calculul final
            consum_energie = obj.distanta * consum_baza * factor_greutate * factor_viteza * factor_drum;
        end
    end
end
```

De mentionat este ca la testare am introdus date diferite doar cu privire la distanta si tipul drumului specifice fiecarei muchii; viteza si greutatea vehiculului raman constante si pot fi ajustate la testare in functie de nevoie. A se vedea ultima sectiune.

Graph.m

Aceasta clasa permite crearea unui graf cu nodurile(vertexurile) si muchiile necesare intr-o lista de adiacenta.

```
classdef Graph < handle
    properties
        V  %nr de noduri din graf
        adj  %lista de adiacență (stocăm muchiile și greutățile lor)
    end
```

Functiile din aceasta clasa sunt apelari ale algoritmilor mentionati mai sus plus functia de addEdge relevanta in adaugarea unei muchii.

```
    %Funcție pentru a adăuga o muchie între nodul u(curent) și nodul v(vecin), cu greutatea weight
    function addEdge(obj, u, v, weight)
```

A) Functia A*(AStar)

```
  % Algoritm A* simplificat (similar cu Dijkstra)
function [dist, parent] = astarSimplu(obj, start, goal)
    fprintf('A* - Găsire drum de la nodul %d la nodul %d\n', start, goal);
```

AStar gaseste cel mai scurt drum intre un nod de start si un nod destinatie specific.

Algoritmul este similar cu Dijkstra, dar foloseste o functie euristica pentru a ghida cautarea h_score spre exemplu.

Evalueaza nodurile pe baza costului real plus o estimare pana la destinatie

Prioritizeaza explorarea nodurilor care par mai promitatoare

Mai rapid decat Dijkstra pentru gasirea unui drum specific intre doua noduri doar in situatia in care numarul de noduri este mai mare. A se vedea la final.

Complexitate: variabila, depinde de euristica folosita. In cazul nostru folosim distanta Manhattan.

```
            % Folosim o euristică reală care direcționează către țintă
            h_score = abs(neighbor - goal);
```

Distanta Manhattan este diferenta absoluta dintre indicele nodului curent si al destinatiei (abs(neighbor - goal)), fiind relevanta doar daca numerele nodurilor au o legatura cu pozitia lor fizica in graf sau sunt aranjate intr-o ordine logica.

B) Functia Djikstra

```
%Algoritmul Dijkstra
function [dist, parent] = dijkstra(obj, start)
```

Dijkstra gaseste cel mai scurt drum de la un nod de start la toate celelalte noduri.

Initializeaza toate distantele cu infinit, nodul start cu 0.

La fiecare pas, alege nodul nevizitat cu distanta minima.

Actualizeaza distantele vecinilor daca gaseste drumuri mai scurte.

Marcheaza nodul ca vizitat si repeta.

C) Cautarea Bruta

```
% Funcția principală pentru căutarea brută
function bruteForceSearch(obj, start, sfarsit)
```

Cautarea Bruta exploreaza toate drumurile posibile pentru a gasi drumul optim intre doua noduri.

Foloseste backtracking recursiv pentru a explora toate caile posibile.

Tine evidenta celui mai bun drum gasit pana in prezent.

Elimina caile care depasesc deja costul minim gasit.

III. <u>Testare</u>

In mod specific sunt doua teste pe care le dorim a le exemplifica. Primul test contine 5 noduri si un numar de 10 muchii al doilea test contine 100 de noduri si 200 de muchii.

In ambele cazuri am construit tabele pentru muchii, tipul de drum si distanta muchilor. In ambele cazuri viteza si greutatea sunt constante.

Date Testul 1:

```
muchii =  [1, 3; 2, 3; 2, 4; 3, 4; 3, 5; 4, 5; 1, 4; 1, 5; 2, 5; 4, 3];
distante = [5, 2, 1, 9, 2, 4, 15, 20, 10, 3];
tipuri_drum = [1, 1, 1.2, 1, 1.1, 0.9, 0.8, 1, 1.2, 1];

% Viteza si greutatea masinii pot fi ajustate
greutate = 1500;
viteza = 60;
```

Date Testul 2:

```
muchii2 = [
    1, 2; 2, 3; 3, 4; 4, 5; 5, 6; 6, 7; 7, 8; 8, 9; 9, 10; 10, 11;
    11, 12; 12, 13; 13, 14; 14, 15; 15, 16; 16, 17; 17, 18; 18, 19; 19, 20; 20, 21;
    21, 22; 22, 23; 23, 24; 24, 25; 25, 26; 26, 27; 27, 28; 28, 29; 29, 30; 30, 31;
    31, 32; 32, 33; 33, 34; 34, 35; 35, 36; 36, 37; 37, 38; 38, 39; 39, 40; 40, 41;
    41, 42; 42, 43; 43, 44; 44, 45; 45, 46; 46, 47; 47, 48; 48, 49; 49, 50; 50, 51;
    51, 52; 52, 53; 53, 54; 54, 55; 55, 56; 56, 57; 57, 58; 58, 59; 59, 60; 60, 61;
    61, 62; 62, 63; 63, 64; 64, 65; 65, 66; 66, 67; 67, 68; 68, 69; 69, 70; 70, 71;
    71, 72; 72, 73; 73, 74; 74, 75; 75, 76; 76, 77; 77, 78; 78, 79; 79, 80; 80, 81;
    81, 82; 82, 83; 83, 84; 84, 85; 85, 86; 86, 87; 87, 88; 88, 89; 89, 90; 90, 91;
    91, 92; 92, 93; 93, 94; 94, 95; 95, 96; 96, 97; 97, 98; 98, 99; 99, 100;
    17, 42; 73, 11; 26, 82; 53, 15; 39, 67; 91, 8; 64, 31; 5, 88; 47, 22; 79, 60;
    34, 95; 56, 13; 87, 28; 10, 69; 44, 99; 72, 23; 36, 54; 77, 16; 45, 89; 61, 30;
    2, 93; 58, 25; 81, 49; 20, 75; 63, 33; 14, 85; 70, 48; 37, 98; 52, 24; 96, 7;
    29, 66; 83, 41; 12, 57; 46, 94; 21, 78; 68, 35; 90, 4; 27, 51; 74, 38; 19, 86;
    59, 32; 43, 97; 65, 9; 40, 84; 71, 18; 1, 76; 55, 92; 28, 62; 80, 50; 6, 100;
    48, 97; 14, 82; 34, 59; 71, 26; 3, 89; 56, 22; 76, 42; 29, 68; 91, 15; 45, 96;
    20, 65; 74, 37; 12, 85; 53, 33; 27, 78; 94, 16; 49, 99; 7, 61; 81, 40; 23, 67;
    95, 30; 44, 10; 72, 54; 38, 86; 60, 25; 5, 93; 52, 35; 77, 19; 31, 64; 87, 43;
    17, 66; 58, 39; 83, 11; 47, 70; 21, 92; 63, 36; 9, 79; 50, 24; 75, 2; 32, 98;
    57, 46; 13, 69; 84, 28; 41, 100; 73, 18; 25, 90; 55, 51; 8, 80; 62, 29; 4, 88
    ];
```

```
distante2 = [
    12, 5, 18, 7, 14, 3, 11, 16, 9, 2, 19, 8, 15, 4, 13, 6, 17, 10, 1, 5,...
    15, 8, 12, 3, 19, 7, 14, 9, 2, 16, 4, 11, 18, 5, 13, 6, 20, 10, 1, 17,...
    8, 14, 3, 19, 7, 15, 6, 11, 2, 18, 9, 4, 16, 5, 12, 1, 19, 7, 13, 8,...
    14, 3, 17, 10, 5, 15, 9, 2, 18, 6, 12, 4, 16, 7, 11, 1, 19, 8, 13, 2,...
    17, 5, 14, 9, 3, 18, 10, 6, 15, 4, 11, 7, 19, 8, 12, 2, 16, 5, 9, 10,...
    9, 16, 4, 12, 7, 18, 3, 14, 6, 11, 2, 17, 8, 15, 5, 19, 7, 13, 4, 16,...
    9, 1, 18, 6, 12, 3, 15, 8, 14, 2, 17, 5, 19, 10, 1, 16, 7, 13, 9, 3,...
    18, 6, 11, 4, 20, 8, 15, 7, 13, 2, 17, 5, 19, 8, 14, 6, 11, 3, 16, 9,...
    1, 18, 7, 12, 4, 15, 8, 9, 3, 18, 6, 17, 4, 20, 8, 15, 7, 13, 3, 18,...
    6, 11, 9, 1, 16, 7, 13, 9, 3, 16, 5, 17, 9, 1, 11, 2, 19, 8, 7, 4
    ];
tipuri_drum2 = [

1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2,
1.0, 0.9, 1.1, 0.8, 1.2, 1.0, 0.9, 1.1, 0.8, 1.2
];

% Viteza si greutatea masinii pot fi ajustate
greutate2 = 1500;
viteza2 = 60;

```

In ambele cazuri am utilizat un for loop pentru popularea grafurilor. De exemplu pentru testul 2 am creat un graf cu 100 de noduri si am populat graful cu datele de mai sus:

```
% Creează un graf cu 100 de noduri
f = Graph(100);


%Populam graful
for i = 1:length(muchii2)
    cost = Cost(distante2(i), tipuri_drum2(i), greutate2, viteza2);
    consum_energie = cost.getConsumEnergie();
    f.addEdge(muchii2(i, 1), muchii2(i, 2), consum_energie);
end

```

Ulterior, pentru fiecare algoritm am salvat timpul la care a inceput rularea algoritmului, am rulat algoritmul si am salvat timpul la care rularea algoritmului s-a terminat. Diferenta reprezinta timpul de rulare al algoritmului si este afisat in milisecunde.

```
% A*
start_t0 = datetime('now');
fprintf('_____\n');
fprintf('Testare A*\n');
fprintf('_____\n');
[dist_astar, parent_astar] = f.astarSimplu(start_node, goal_node);
endt_t1 = datetime('now');
timpAStar = milliseconds(endt_t1 - start_t0);
fprintf('Algoritmul A* a durat:%.2f milisecunde\n',timpAStar);


% Dijkstra
start_t0 = datetime('now');
fprintf('_____\n');
fprintf('Testare Djikstra\n');
fprintf('_____\n');
[dist_dijkstra, parent_dijkstra] = f.dijkstra(start_node);
endt_t1 = datetime('now');
timpDijkstra = milliseconds(endt_t1 - start_t0);
fprintf('Algoritmul Dijkstra a durat:%.2f milisecunde\n',timpDijkstra);

% Cautare Bruta
start_t0 = datetime('now');
fprintf('_____\n');
fprintf('Testare Cautare Bruta\n');
fprintf('_____\n');
f.bruteForceSearch(start_node, goal_node);
endt_t1 = datetime('now');
timpCautareBruta = milliseconds(endt_t1 - start_t0);
fprintf('Algoritmul de Cautare Bruta a durat:%.2f milisecunde\n',timpCautareBruta);
```

Ce am observat?

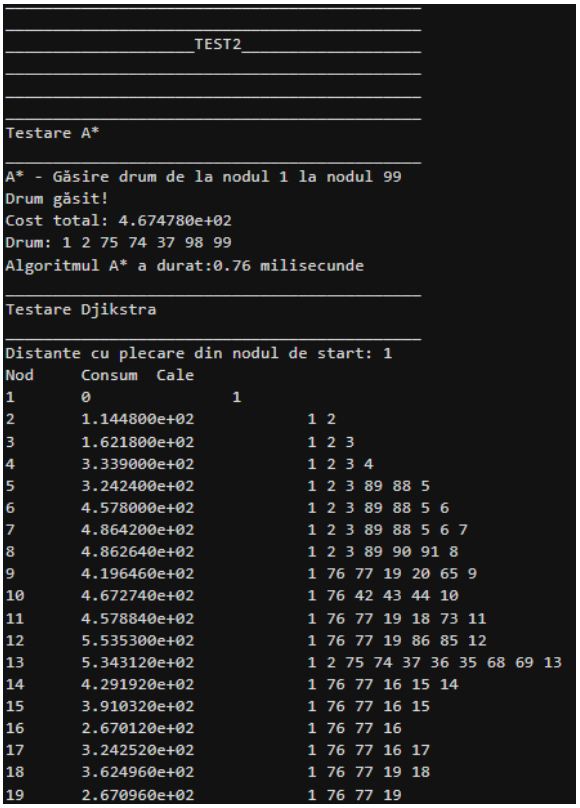
In cazul rularii Testului 1 (cu un dataset mai restrans) in mod neintuitiv A* pare a fi mai incet chiar fata de cautarea bruta, cu toate acestea, eficienta acestuia se poate observa in cazul rularii Testului 2 unde avem un dataset mult mai larg.

Rezultate Test 1

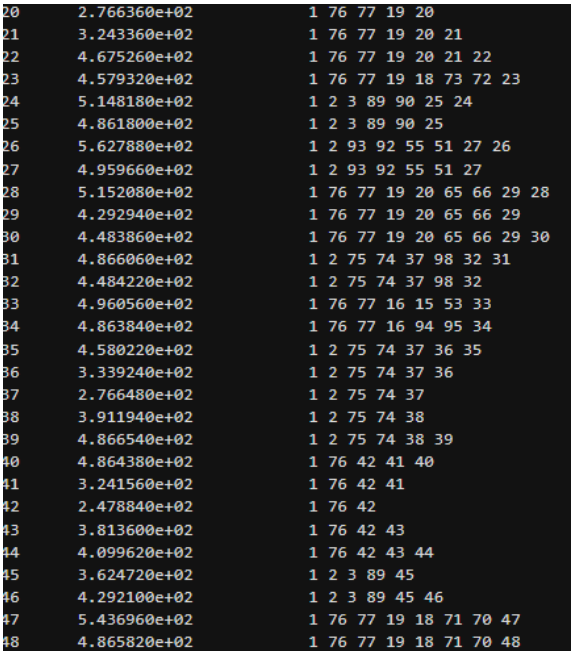
```
>> DijkstraTestScript
_____________________________________________
_____________________________________________
____________________TEST1____________________
_____________________________________________
_____________________________________________
_____________________________________________
Testare A*
_____________________________________________
A* - Găsire drum de la nodul 1 la nodul 5
Drum găsit!
Cost total: 6.676800e+01
Drum: 1 3 5
Algoritmul A* a durat:0.73 milisecunde
_____________________________________________
Testare Djikstra
_____________________________________________
Distante cu plecare din nodul de start: 1
Nod     Consum  Cale
1       0               1
2       6.678000e+01            1 3 2
3       4.770000e+01            1 3
4       7.630800e+01            1 3 2 4
5       6.676800e+01            1 3 5
Algoritmul Dijkstra a durat:0.82 milisecunde
_____________________________________________
Testare Cautare Bruta
_____________________________________________
Cel mai scurt drum gasit:
Cost total: 6.676800e+01
Drum: 1 3 5
Algoritmul de Cautare Bruta a durat:0.50 milisecunde
```

Rezultate Test 2

Se poate observa rularea foarte rapida a algoritmului A* in acest caz – 0.76 milisecunde, mult mai rapid decat algoritmul Djikstra de peste 3 milisecunde si/sau cautarea bruta de peste 8 milisecunde.



```
_____TEST2_____
Testare A*
A* - Găsire drum de la nodul 1 la nodul 99
Drum găsit!
Cost total: 4.674780e+02
Drum: 1 2 75 74 37 98 99
Algoritmul A* a durat:0.76 milisecunde
Testare Djikstra
Distante cu plecare din nodul de start: 1
Nod     Consum  Cale
1       0               1
2       1.144800e+02            1 2
3       1.621800e+02            1 2 3
4       3.339000e+02            1 2 3 4
5       3.242400e+02            1 2 3 89 88 5
6       4.578000e+02            1 2 3 89 88 5 6
7       4.864200e+02            1 2 3 89 88 5 6 7
8       4.862640e+02            1 2 3 89 90 91 8
9       4.196460e+02            1 76 77 19 20 65 9
10      4.672740e+02            1 76 42 43 44 10
11      4.578840e+02            1 76 77 19 18 73 11
12      5.535300e+02            1 76 77 19 86 85 12
13      5.343120e+02            1 2 75 74 37 36 35 68 69 13
14      4.291920e+02            1 76 77 16 15 14
15      3.910320e+02            1 76 77 16 15
16      2.670120e+02            1 76 77 16
17      3.242520e+02            1 76 77 16 17
18      3.624960e+02            1 76 77 19 18
19      2.670960e+02            1 76 77 19
```



```
20      2.766360e+02            1 76 77 19 20
21      3.243360e+02            1 76 77 19 20 21
22      4.675260e+02            1 76 77 19 20 21 22
23      4.579320e+02            1 76 77 19 18 73 72 23
24      5.148180e+02            1 2 3 89 90 25 24
25      4.861800e+02            1 2 3 89 90 25
26      5.627880e+02            1 2 93 92 55 51 27 26
27      4.959660e+02            1 2 93 92 55 51 27
28      5.152080e+02            1 76 77 19 20 65 66 29 28
29      4.292940e+02            1 76 77 19 20 65 66 29
30      4.483860e+02            1 76 77 19 20 65 66 29 30
31      4.866060e+02            1 2 75 74 37 98 32 31
32      4.484220e+02            1 2 75 74 37 98 32
33      4.960560e+02            1 76 77 16 15 53 33
34      4.863840e+02            1 76 77 16 94 95 34
35      4.580220e+02            1 2 75 74 37 36 35
36      3.339240e+02            1 2 75 74 37 36
37      2.766480e+02            1 2 75 74 37
38      3.911940e+02            1 2 75 74 38
39      4.866540e+02            1 2 75 74 38 39
40      4.864380e+02            1 76 42 41 40
41      3.241560e+02            1 76 42 41
42      2.478840e+02            1 76 42
43      3.813600e+02            1 76 42 43
44      4.099620e+02            1 76 42 43 44
45      3.624720e+02            1 2 3 89 45
46      4.292100e+02            1 2 3 89 45 46
47      5.436960e+02            1 76 77 19 18 71 70 47
48      4.865820e+02            1 76 77 19 18 71 70 48
```

```
49      5.819520e+02            1 76 42 41 40 81 49
50      6.007560e+02            1 2 93 92 55 51 50
51      4.291440e+02            1 2 93 92 55 51
52      4.673280e+02            1 76 77 16 15 53 52
53      4.291920e+02            1 76 77 16 15 53
54      4.577580e+02            1 2 93 92 55 54
55      4.100880e+02            1 2 93 92 55
56      5.244960e+02            1 2 93 92 55 56
57      5.340300e+02            1 2 93 92 55 56 57
58      5.437980e+02            1 2 75 74 37 98 32 59 58
59      4.770600e+02            1 2 75 74 37 98 32 59
60      5.153760e+02            1 76 77 78 79 60
61      4.865460e+02            1 76 77 19 20 65 66 29 30 61
62      5.055180e+02            1 76 77 19 20 65 66 29 62
63      4.863720e+02            1 2 75 74 37 36 63
64      4.579620e+02            1 76 77 19 20 65 64
65      3.624420e+02            1 76 77 19 20 65
66      4.102020e+02            1 76 77 19 20 65 66
67      4.865880e+02            1 76 77 19 18 73 72 23 67
68      4.675680e+02            1 2 75 74 37 36 35 68
69      4.866720e+02            1 2 75 74 37 36 35 68 69
70      4.579440e+02            1 76 77 19 18 71 70
71      4.006320e+02            1 76 77 19 18 71
72      4.102320e+02            1 76 77 19 18 73 72
73      3.720240e+02            1 76 77 19 18 73
74      2.670960e+02            1 2 75 74
75      2.002320e+02            1 2 75
76      1.906800e+02            1 76
77      2.002320e+02            1 76 77
78      3.817200e+02            1 76 77 78
79      4.581360e+02            1 76 77 78 79
80      5.823120e+02            1 76 77 78 79 80
81      5.724060e+02            1 76 42 41 40 81
82      5.340780e+02            1 76 42 41 83 82
83      4.864380e+02            1 76 42 41 83
```

```
84      4.673460e+02            1 76 77 19 86 85 84
85      3.815940e+02            1 76 77 19 86 85
86      3.530100e+02            1 76 77 19 86
87      3.909000e+02            1 2 3 89 88 87
88      2.956200e+02            1 2 3 89 88
89      2.384520e+02            1 2 3 89
90      3.813720e+02            1 2 3 89 90
91      4.194840e+02            1 2 3 89 90 91
92      3.338160e+02            1 2 93 92
93      2.671200e+02            1 2 93
94      3.052200e+02            1 76 77 16 94
95      3.814440e+02            1 76 77 16 94 95
96      4.957800e+02            1 76 77 16 94 95 96
97      5.056500e+02            1 76 77 19 18 71 70 48 97
98      4.198380e+02            1 2 75 74 37 98
99      4.674780e+02            1 2 75 74 37 98 99
100     4.099080e+02            1 76 42 41 100
Algoritmul Dijkstra a durat:3.34 milisecunde
_____________________________________________
Testare Cautare Bruta
_____________________________________________
Cel mai scurt drum gasit:
Cost total: 4.674780e+02
Drum: 1 2 75 74 37 98 99
Algoritmul de Cautare Bruta a durat:8.63 milisecunde
>>
```