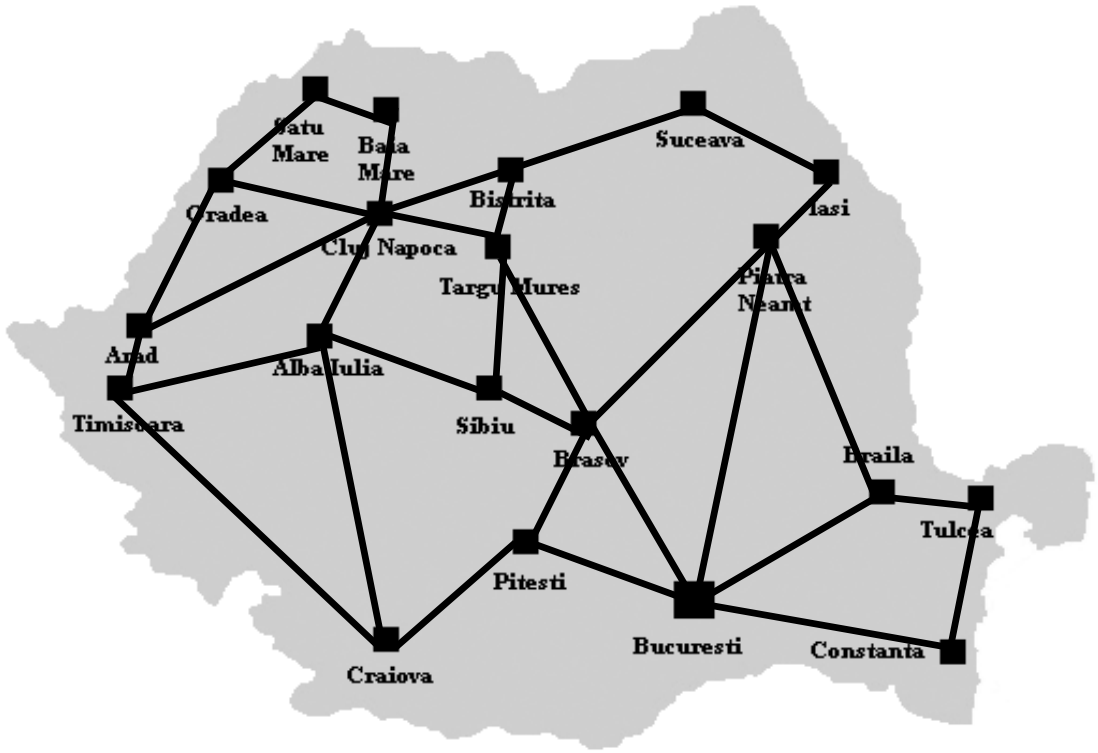
Algoritmul de căutare cu cost uniform-rezolvare



Găsiți distantele rutiere dintre orașele de pe harta din figură. Utilizați-le apoi pentru a implementa un algoritm de căutare cu cost uniform (folosind instrucțiunile de mai jos) pentru a ajunge de la un oraș la altul, de la Oradea la Tulcea.

Instrucțiuni

Algoritm de căutare cu cost uniform
Toate orașele sunt nevizitate.
Adăugăm in lista *noduri* orașul de plecare.
Marcam orașul de plecare ca vizitat.
Inițializăm costul orașului de plecare cu 0.
Cat timp soluție negăsită si *noduri* ≠ ∅ *executa*
    nod = scoate_din_față(*noduri*) //stocam primul element din *noduri* in variabila *nod*
    Eliminam primul element din *noduri*
    Daca testare_țintă[problema] se aplica la stare(nod) *atunci*
        Soluția este găsită //facem variabila booleana găsit adevărată
    Altfel
        Găsim orașele conectate de *nod* si calculam pentru ele costul ca fiind suma dintre costul lui *nod* + distanta de la *nod* la ele

Identificam *(*orașele nevizitate care sunt conectate de *nod*) si (orașele vizitate anterior daca au un cost mai mic decât aveau când au fost vizitate anterior)
Stabilim costul pentru orașe găsite (suma dintre costul lui *nod* + distanta de la *nod* la ele)
Adăugăm aceste orașe la *noduri* astfel încât orașele din noduri sunt ordonate după costul fiecăreia
Orașele adăugate sunt marcate ca vizitate
Se reține pentru oricare din orașele adăugate nodul *părinte* ca fiind *nod*

Sfârșit cat timp
Stocam soluția parcurgând orașele de la destinație către start utilizând *părinții* reținuți.
Afișăm soluția si costul (numărul de km al soluției).

Programe primite ca soluții

În continuare, pun soluțiile primite pentru a le avea cu toții ca exemplu. Nu toate sunt perfecte, însă și din greșeli putem învăța. Ordinea soluțiilor este aleatorie.
Vă amintesc că acest program a fost deja rezolvat la orele de laborator.

Soluția 1

Observație: programul este incomplet, calculeaza costurile, verifica si daca noul oras de adaugat a mai fost vizitat si are acum un cost mai mic, dar nu ordoneaza orasele in noduri dupa cost, ci doar adauga orasul pe ultima pozitie din noduri.

```
1.  #include<string>
2.  #include<iostream>
3.
4.  using namespace std;
5.
6.  string v[20] = { "Satu Mare", "Baia Mare", "Oradea", "Cluj Napoca", "Bistrita", "S
    uceava", "Iasi", "Piatra Neamt", "Targu Mures", "Arad", "Timisoara", "Alba Iulia",
     "Sibiu", "Brasov", "Braila", "Tulcea", "Craiova", "Pitesti", "Bucuresti", "Consta
    nta" };
7.  int q, a[20][20];
8.
9.  void latime(int start, int stop)
10. {
11.     int viz[20], noduri[20], nrnoduri = 0, parinte[20], ok = 0, cost[20], cost2[20
    ];
12.     for (int i = 0; i < 20; i++)
13.     {
14.         cost[i]=0;
15.         cost2[i]=0;
16.     }
17.     noduri[nrnoduri++] = start;
18.     for (int i = 0; i < 20; i++)
19.         viz[i] = 0;
20.     viz[start] = 1;
21.     int pas = 0;
22.     while ((ok == 0) && (nrnoduri > 0))
23.     {
24.         int nod = noduri[0];
25.         for (int i = 0; i < nrnoduri; i++)
26.             noduri[i] = noduri[i + 1];  //stergem elem de pe prima pozitie
27.         nrnoduri--;
28.         if (nod == stop) ok = 1;
29.         else
30.             for (int i = 0; i < 20; i++)
31.             {
32.                 if (a[nod][i]!=0)
33.                     cost2[i]=cost2[parinte[i]]+a[nod][i];
34.                 if(((a[nod][i]!=0)&&(viz[i]==0))||((viz[i]!=0)&&(cost2[i]<cost[par
    inte[i]]+a[nod][i])))
35.                 {
36.                     cost[i]=cost[parinte[i]]+a[nod][i];
37.                     noduri[nrnoduri++] = i;  //adaugam nodul i pe ultima pozitie d
    in noduri
38.                     viz[i] = 1;
39.                     parinte[i] = nod;
40.                 }
41.             }
```

```
    }
    int solutie[20], nrsol = 0, final = stop;
    while (final != start)
    {
        solutie[nrsol++] = final;
        final = parinte[final];
    }
    solutie[nrsol++] = start;
    cout << endl << endl << "Cautare in latime de la " << v[start] << " la " << v[
stop] << endl;
    for (int i = nrsol - 1; i >= 0; i--)
        cout << v[solutie[i]] << " ";
    cout << endl<<endl;
    cout<<"Costul este "<<cost[stop];
    cout<<" de km";
}


int main()
{
    a[0][1]=62;
    a[0][2]=133;
    a[1][3]=148;
    a[2][3]=152;
    a[2][9]=118;
    a[3][9]=268;
    a[3][11]=98;
    a[3][8]=111;
    a[3][4]=116;
    a[4][5]=190;
    a[4][8]=97;
    a[5][6]=144;
    a[6][7]=130;
    a[7][13]=238;
    a[7][14]=255;
    a[7][18]=347;
    a[8][12]=115;
    a[8][13]=170;
    a[9][10]=54;
    a[10][11]=217;
    a[10][16]=341;
    a[11][16]=301;
    a[11][12]=77;
    a[12][13]=142;
    a[13][17]=137;
    a[13][18]=184;
    a[14][18]=218;
    a[14][15]=94;
    a[15][19]=125;
    a[16][17]=121;
    a[17][18]=117;
    a[18][19]=224;
    for (int i = 0; i < 20; i++)
        for (int j = 0; j < 20; j++)
            a[j][i] = a[i][j];
    int b=2, c=15;
    cout<<"Parcurgere: "<<endl;
    cout<<endl;
    latime(b,c);
       }
```

Soluția 2

Observație: programul realizat in C#, nu in C/C++. Îl postez, poate se găsește cineva interesat de soluție.

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using Wintellect.PowerCollections;
5.
6.  // NOTE: Aceasta abordare este relativ orientata pe obiect
7.  // NOTE: De retinut, am folosit anumite extensii din niste librarii ale limbajului

8.  //       Dar algoritmul de cautare cu cost uniform, precum si alte functii(cele de
     afisare, de determinare a drumulu optim)
9.  // NOTE: Graful este format din mai multe noduri, fiecare dintre ele avand o lista
     cu nodurile adiacente
10. namespace CautareCuCostUniform
11. {
12.     /// <summary>
13.     /// Clasa nod ce reprezinta fiecare oras
14.     /// </summary>
15.     public class Nod
16.     {
17.         /// <summary>
18.         /// Numele orasului
19.         /// </summary>
20.         public string Nume { get; set; }
21.
22.         /// <summary>
23.         /// Costul pe care il vei face pana la acest nod, daca il vei parcurge
24.         /// </summary>
25.         public int CostulDrumului { get; set; }
26.
27.         /// <summary>
28.         /// Lista de noduri adiacente ale acestui nod
29.         /// </summary>
30.         public IList<Muchie> ListaDeAdiacenta { get; set; }
31.
32.         /// <summary>
33.         /// Parintele acestui nod
34.         /// </summary>
35.         public Nod Parinte { get; set; }
36.
37.         /// <summary>
38.         /// Constructor parametrizat
39.         /// </summary>
40.         /// <param name="nume"> Numele nodului </param>
41.         public Nod(string nume)
42.         {
43.             Nume = nume;
44.         }
45.     }
46.
47.     /// <summary>
48.     /// Clasa muchie ce reprezinta legatura dintre doua orase( drumul in sine )
49.     /// </summary>
50.     public class Muchie
51.     {
52.         /// <summary>
53.         /// Costul muchies/drumului
54.         /// </summary>
55.         public int Cost { get; set; }
56.
57.         /// <summary>
```

```
58.         /// Nodul/Orasul de la capatul drumului
59.         /// </summary>
60.         public Nod NodTinta { get; set; }
61.
62.         /// <summary>
63.         /// Constructor parametrizat
64.         /// </summary>
65.         /// <param name="cost">Costul parcurgerii drumului</param>
66.         /// <param name="nodTinta">Nodul/Orasul de la capatul drumului</param>
67.         public Muchie(Nod nodTinta, int cost)
68.         {
69.             NodTinta = nodTinta;
70.             Cost = cost;
71.         }
72.     }
73.
74.     class Program
75.     {
76.         /// <summary>
77.         /// Functia ce efectueaza cautarea cu cost uniform
78.         /// </summary>
79.         /// <param name="start">Nodul de plecare</param>
80.         /// <param name="sosire">Nodul de sosire</param>
81.         public static void CautareCuCostUniform(Nod start, Nod sosire)
82.         {
83.             // Marcam costul drumului din punctul de plecare ca fiind 0
84.             start.CostulDrumului = 0;
85.
86.             // Creeam o coada(o coada cu prioritati mai exact)
87.             // Care ne va ajuta sa ordonam nodurile(orasele) dupa un criteriu de c
    omparatie
88.             // In cazul nostru vom compara dupa costul drumului a doua noduri, si
    astfel
89.             // vom ordona corespunzator orasele
90.             OrderedSet<Nod> coada = new OrderedSet<Nod>((i, j) =>
91.             {
92.                 if (i.CostulDrumului > j.CostulDrumului)
93.                     return 1;
94.
95.                 else if (i.CostulDrumului < j.CostulDrumului)
96.                     return -1;
97.
98.                 else
99.                 {
100.                        return 0;
101.                    }
102.                });
103.
104.                // Adaugam nodul de start in coada
105.                coada.Add(start);
106.
107.                // Creeam o structura de tipul multime, care tine evidenta nodu
    rilor deja vizitate
108.                // Aceasta nu poate contine elemente duplicat
109.                HashSet<Nod> noduriVizitate = new HashSet<Nod>();
110.
111.                // O variabila booleana care determina daca am ajuns la nodul d
    e sosire
112.                bool gasit = false;
113.
114.                do
115.                {
116.                    // Gasim nodul actual eliminandu-l de la inceputul cozii
117.                    Nod nodActual = coada.RemoveFirst();
118.
119.                    // Marcam acest nod ca vizitat
```

```
120.                         noduriVizitate.Add(nodActual);
121.
122.                         // Daca numele coincid...
123.                         if (nodActual.Nume == sosire.Nume)
124.                             // Atunci am ajuns la destinatie
125.                             gasit = true;
126.
127.                         // Parcurgem lista de adiacenta a nodului actual
128.                         foreach (Muchie muchie in nodActual.ListaDeAdiacenta)
129.                         {
130.                             // Cautam nodul de la celalalt capat al muchiei
131.                             // Adica nodul copil, care va fi initializat intr-
    o variabila noua
132.                             Nod nodCopil = muchie.NodTinta;
133.
134.                             // Preluam costul muchiei (lungimea drumului de la nodu
    l actual la nodul copil)
135.                             int cost = muchie.Cost;
136.
137.                             // Marcam costul drumului ca fiind costul drumului nodu
    lui actual + costul muchiei
138.                             // Aceasta formula ne ajuta va ajuta sa acumulam de fie
    care data costul drumului pentru a ajunge
139.                             // la un cost optim
140.                             nodCopil.CostulDrumului = nodActual.CostulDrumului + co
    st;
141.
142.                             // Daca nodCopil nu a mai fost vizitat si nu apartine c
    ozii...
143.                             if(!noduriVizitate.Contains(nodCopil) && !coada.Contain
    s(nodCopil))
144.                             {
145.                                 // Marcam parintele lui nodCopil ca fiind nodActual

146.                                 nodCopil.Parinte = nodActual;
147.
148.                                 // Adaugam nodCopil in coada
149.                                 coada.Add(nodCopil);
150.
151.                                 // Afisam numele nodului copil
152.                                 Console.WriteLine(nodCopil.Nume);
153.
154.                                 // Afisam coada actuala
155.                                 AfisareListaDeNoduri<Nod>(coada);
156.                                 Console.WriteLine();
157.                             }
158.                             // Altfel daca nodCopil apartine cozii si costul drumul
    ui din nodul copil
159.                             // este mai mare mare decat costul drumului din nodul a
    ctual + costul muchiei
160.                             else if ((coada.Contains(nodCopil)) && (nodCopil.Costul
    Drumului > nodActual.CostulDrumului + cost))
161.                             {
162.                                 // Marcam parintele lui nodCopil ca fiind nodActual

163.                                 nodCopil.Parinte = nodActual;
164.
165.
166.                                 nodCopil.CostulDrumului = nodActual.CostulDrumului
    + cost;
167.
168.                                 // Scoatem nodul copil din coada
169.                                 coada.Remove(nodCopil);
170.
171.                                 // Si il adaugam iar
172.                                 coada.Add(nodCopil);
```

```
                            }
                        }

                    // Parcurgem pana cand coada este goala
                    } while (!(coada.Count == 0));

                }

                /// <summary>
                /// Functia care ne returneaza drumul optim
                /// </summary>
                /// <param name="sosire">Orasul de sosire</param>
                /// <returns></returns>
                private static IList<Nod> DrumOptim(Nod sosire)
                {
                    IList<Nod> drum = new List<Nod>();

                    for(Nod nod = sosire; nod != null; nod = nod.Parinte)
                    {
                        drum.Add(nod);
                    }

                    return drum.Reverse().ToList();
                }

                /// <summary>
                /// Functia care afiseaza drumul optim
                /// </summary>
                /// <param name="drum">Drumul optim parcurs</param>
                public static void AfisareDrumOptim(IList<Nod> drum)
                {
                    foreach (var nod in drum)
                    {
                        Console.Write(nod.Nume + "-> ");
                    }
                }

                /// <summary>
                /// Functia care afiseaza o lista oarecare de noduri
                /// </summary>
                /// <typeparam name="T"></typeparam>
                /// <param name="lista"></param>
                public static void AfisareListaDeNoduri<T>(IEnumerable<T> lista) wh
    ere T : Nod
                {
                    foreach (var nod in lista)
                    {
                        Console.Write(nod.Nume);
                    }
                }

                static void Main(string[] args)
                {
                    //initializam orasele de pe harta ca fiind noduri
                    Nod n1 = new Nod("Arad");
                    Nod n2 = new Nod("Zerind");
                    Nod n3 = new Nod("Oradea");
                    Nod n4 = new Nod("Sibiu");
                    Nod n5 = new Nod("Fagaras");
                    Nod n6 = new Nod("Rimnicu Vilcea");
                    Nod n7 = new Nod("Pitesti");
                    Nod n8 = new Nod("Timisoara");
                    Nod n9 = new Nod("Lugoj");
                    Nod n10 = new Nod("Mehadia");
                    Nod n11 = new Nod("Drobeta");
                    Nod n12 = new Nod("Craiova");
```

```
238.                    Nod n13 = new Nod("Bucharest");
239.                    Nod n14 = new Nod("Giurgiu");
240.                    Nod n15 = new Nod("Urziceni");
241.                    Nod n16 = new Nod("Hirsova");
242.                    Nod n17 = new Nod("Eforie");
243.                    Nod n18 = new Nod("Vaslui");
244.                    Nod n19 = new Nod("Iasi");
245.                    Nod n20 = new Nod("Neamt");
246.
247.                    //initializam muchiile
248.                    n1.ListaDeAdiacenta = new List<Muchie>
249.                    {
250.                        new Muchie(n2, 75),
251.                        new Muchie(n4, 140),
252.                        new Muchie(n8, 118)
253.                    };
254.
255.                    n2.ListaDeAdiacenta = new List<Muchie>
256.                    {
257.                        new Muchie(n1, 75),
258.                        new Muchie(n3, 71)
259.                    };
260.
261.                    n3.ListaDeAdiacenta = new List<Muchie>
262.                    {
263.                        new Muchie(n2, 71),
264.                        new Muchie(n4, 151)
265.                    };
266.
267.                    n4.ListaDeAdiacenta = new List<Muchie>
268.                    {
269.                        new Muchie(n1, 140),
270.                        new Muchie(n5, 99),
271.                        new Muchie(n3, 151),
272.                        new Muchie(n6, 80),
273.                    };
274.
275.                    n5.ListaDeAdiacenta = new List<Muchie>
276.                    {
277.                        new Muchie(n4, 99),
278.                        new Muchie(n13, 211)
279.                    };
280.
281.                    n6.ListaDeAdiacenta = new List<Muchie>
282.                    {
283.                        new Muchie(n4, 80),
284.                        new Muchie(n7, 97),
285.                        new Muchie(n12, 146)
286.                    };
287.
288.                    n7.ListaDeAdiacenta = new List<Muchie>
289.                    {
290.                        new Muchie(n6, 97),
291.                        new Muchie(n13, 101),
292.                        new Muchie(n12, 138)
293.                    };
294.
295.                    n8.ListaDeAdiacenta = new List<Muchie>
296.                    {
297.                        new Muchie(n1, 118),
298.                        new Muchie(n9, 111)
299.                    };
300.
301.                    n9.ListaDeAdiacenta = new List<Muchie>
302.                    {
303.                        new Muchie(n8, 111),
```

```
304.                        new Muchie(n10, 70)
305.                    };
306.
307.                    n10.ListaDeAdiacenta = new List<Muchie>
308.                    {
309.                        new Muchie(n9, 70),
310.                        new Muchie(n11, 75)
311.                    };
312.
313.                    n11.ListaDeAdiacenta = new List<Muchie>
314.                    {
315.                        new Muchie(n10, 75),
316.                        new Muchie(n12, 120)
317.                    };
318.
319.                    n12.ListaDeAdiacenta = new List<Muchie>
320.                    {
321.                        new Muchie(n11, 120),
322.                        new Muchie(n6, 146),
323.                        new Muchie(n7, 138)
324.                    };
325.
326.                    n13.ListaDeAdiacenta = new List<Muchie>
327.                    {
328.                        new Muchie(n7, 101),
329.                        new Muchie(n14, 90),
330.                        new Muchie(n5, 211),
331.                        new Muchie(n15, 85)
332.                    };
333.
334.                    n14.ListaDeAdiacenta = new List<Muchie>
335.                    {
336.                        new Muchie(n13, 90)
337.                    };
338.
339.                    n15.ListaDeAdiacenta = new List<Muchie>
340.                    {
341.                        new Muchie(n13, 85),
342.                        new Muchie(n16, 98),
343.                        new Muchie(n18, 142)
344.                    };
345.
346.                    n16.ListaDeAdiacenta = new List<Muchie>
347.                    {
348.                        new Muchie(n15, 98),
349.                        new Muchie(n17, 86)
350.                    };
351.
352.                    n17.ListaDeAdiacenta = new List<Muchie>
353.                    {
354.                        new Muchie(n16, 86)
355.                    };
356.
357.                    n18.ListaDeAdiacenta = new List<Muchie>
358.                    {
359.                        new Muchie(n15, 142),
360.                        new Muchie(n19, 92)
361.                    };
362.
363.                    n19.ListaDeAdiacenta = new List<Muchie>
364.                    {
365.                        new Muchie(n18, 92),
366.                        new Muchie(n20, 87)
367.                    };
368.
369.                    n20.ListaDeAdiacenta = new List<Muchie>
```

```
                    {
                        new Muchie(n19, 87)
                    };

                    // Efectuam cautarea cu cost uniforma
                    CautareCuCostUniform(n1, n13);

                    // Gasim drumul optim
                    IList<Nod> drum = DrumOptim(n13);

                    // Afisam acel drum ( OUTPUT: Arad-> Sibiu-> Fagaras-
    > Bucharest->
                    // prin OUTPUT ma refer la ce imi afiseaza la final)

                    AfisareDrumOptim(drum);

                    Console.ReadKey();
                }
            }
        }
```

Soluția 3

Observație: programul reprezintă tot căutarea în lățime, doar au fost adăugați km între orașe în loc să fie valori de 0 și 1. Nu sunt calculate nici costurile. Am apreciat efortul de a culege datele despre distanțele dintre orașe. ☺

```
#include<conio.h>
#include<string>
#include<iostream>

using namespace std;

string nume[20] = { "Satu Mare", "Baia Mare", "Oradea", "Cluj Napoca", "Bistrita",
 "Suceava", "Iasi", "Piatra Neamt",
 "Targu Mures", "Arad", "Timisoara", "Alba Iulia", "Sibiu", "Brasov", "Braila", "T
ulcea", "Craiova", "Pitesti",
 "Bucuresti", "Constanta" };
int a[20][20];

void cost(int start, int stop)
{
    int viz[20], noduri[20], nrnoduri = 0, parinte[20], gasit= 0;

    for (int i = 0; i < 20; i++)
        viz[i] = 0;
    noduri[nrnoduri++] = start;
    viz[start] = 1;
    int pas = 0;

    while ((gasit == 0) && (nrnoduri > 0))
    {
        int nod = noduri[0];

        cout << "Pasul " << ++pas << ": ";
        for (int i = 0; i < nrnoduri; i++)
            cout << nume[noduri[i]] << " - ";
        cout << endl;


        for (int i = 0; i < nrnoduri; i++)
```

```
            noduri[i] = noduri[i + 1];
        nrnoduri--;
        if (nod == stop) gasit= 1;
        else
            for (int i = 0; i < 20; i++)
                if ((a[nod][i]!=0) && (viz[i] == 0))
                {
                    noduri[nrnoduri++] = i;
                    viz[i] = 1;
                    parinte[i] = nod;
                }
    }

    int solutie[20], nrsol = 0, final = stop;

    while (final != start)
    {
        solutie[nrsol++] = final;
        final = parinte[final];
    }
    solutie[nrsol++] = start;
    cout<<endl;
    cout << "Cautare de la " << nume[start] << " la " << nume[stop] << endl;
    cout<<endl;
    for (int i = nrsol - 1; i >= 0; i--)
        cout << nume[solutie[i]] << " "<<endl;;
        cout<<endl;
}


int main()
{
     a[0][1]=62;
    a[0][2]=139;
    a[1][3]=151;
    a[2][3]=159;
    a[2][9]=114;
    a[3][9]=109;
    a[3][11]=107;
    a[3][8]=350;
    a[3][4]=97;
    a[4][5]=196;
    a[4][8]=94;
    a[5][6]=148;
    a[6][7]=130;
    a[7][13]=226;
    a[7][14]=256;
    a[7][18]=354;
    a[8][12]=121;
    a[8][13]=169;
    a[9][10]=61;
    a[10][11]=218;
    a[10][16]=341;
    a[11][16]=75;
    a[11][12]=296;
    a[12][13]=144;
    a[13][17]=140;
    a[13][18]=183;
    a[14][18]=218;
    a[14][15]=96;
    a[15][19]=128;
    a[16][17]=124;
    a[17][18]=119;
    a[18][19]=225;

    for (int i = 0; i < 20; i++)
```

```
99.         for (int j = 0; j < 20; j++)
100.                    a[j][i] = a[i][j];
101.
102.
103.           int start=2, stop=15;
104.
105.           cost(start,stop);
106.
107.       }
```

Soluția 4

Observație: programul păstrează matricea de adiacență a și adaugă încă o matrice b pentru distanțe. Distanțele puteau fi incluse tot în a, dar important este că funcționează. Adăugarea în noduri se face pe ultima poziție, însă apoi se realizează o sortare a tabloului în funcție de costuri.

```
1.  #include<iostream>
2.  #include<conio.h>
3.  #include<string>
4.
5.  using namespace std;
6.
7.  int main() {
8.
9.      string nume[20] = { "Timisoara","Arad","Oradea","Satu Mare", "Baia Mare","Cluj
     Napoca","Alba Iulia","Craiova","Bistrita","Targu Mures","Sibiu","Pitesti","Brasov
    ","Suceava","Iasi","Piatra Neamt","Bucuresti","Braila","Tulcea","Constanta" };
10.
11.     int a[20][20];
12.     int i, j;
13.
14.     for (i = 0; i < 20; i++)
15.     {
16.         for (j = 0; j < 20; j++)
17.         {
18.             a[i][j] = 0;
19.         }
20.     }
21.
22.     a[0][1] = 1;
23.     a[0][6] = 1;
24.     a[0][7] = 1;
25.     a[1][2] = 1;
26.     a[1][5] = 1;
27.     a[2][3] = 1;
28.     a[2][5] = 1;
29.     a[3][4] = 1;
30.     a[4][5] = 1;
31.     a[5][6] = 1;
32.     a[5][8] = 1;
33.     a[5][9] = 1;
34.     a[6][10] = 1;
35.     a[6][7] = 1;
36.     a[7][11] = 1;
37.     a[8][9] = 1;
38.     a[8][13] = 1;
39.     a[9][10] = 1;
40.     a[9][12] = 1;
41.     a[10][12] = 1;
```

```
    a[11][12] = 1;
    a[11][16] = 1;
    a[12][16] = 1;
    a[12][15] = 1;
    a[13][14] = 1;
    a[14][15] = 1;
    a[15][17] = 1;
    a[16][17] = 1;
    a[16][19] = 1;
    a[17][18] = 1;
    a[18][19] = 1;

    for (i = 0; i < 20; i++)
    {
        for (j = 0; j < 20; j++)
        {
            if (a[i][j] == 1)
                a[j][i] = 1;
        }
    }

    int b[20][20];

    for (i = 0; i < 20; i++)
    {
        for (j = 0; j < 20; j++)
        {
            b[i][j] = 0;
        }
    }

    b[0][1] = 53;
    b[0][6] = 219;
    b[0][7] = 340;
    b[1][2] = 118;
    b[1][5] = 268;
    b[2][3] = 113;
    b[2][5] = 152;
    b[3][4] = 62;
    b[4][5] = 148;
    b[5][6] = 98;
    b[5][8] = 110;
    b[5][9] = 111;
    b[6][10] = 77;
    b[6][7] = 301;
    b[7][11] = 121;
    b[8][9] = 91;
    b[8][13] = 192;
    b[9][10] = 115;
    b[9][12] = 170;
    b[10][12] = 142;
    b[11][12] = 137;
    b[11][16] = 118;
    b[12][16] = 184;
    b[12][15] = 238;
    b[13][14] = 144;
    b[14][15] = 129;
    b[15][17] = 255;
        b[16][17] = 218;
        b[16][19] = 224;
        b[17][18] = 95;
        b[18][19] = 125;

        for (i = 0; i < 20; i++)
        {
            for (j = 0; j < 20; j++)
```

```
108.                {
109.                    if (b[i][j] != 0)
110.                        b[j][i] = b[i][j];
111.                }
112.            }
113.
114.            int start = 2, stop = 18, viz[20], noduri[20], nrNod = 0, parinte[20],
    gasit = 0, pas = 0, solutie[20], nrSol = 0, final, sume[20];
115.            final = stop;
116.
117.            //in vectorul sume retinem costul pentru fiecare oras
118.
119.            for (i = 0; i < 20; i++)
120.            {
121.                viz[i] = 0;
122.            }
123.
124.            //costul initial pt. fiecare oras este 0
125.            for (i = 0; i < 20; i++)
126.            {
127.                sume[i] = 0;
128.            }
129.
130.            noduri[nrNod++] = start;
131.            viz[start] = 1;
132.
133.            while ((gasit == 0) && (nrNod > 0))
134.            {
135.                int nod = noduri[0];
136.                //retiem costul pentru orasul care urmeaza sa fie eliminat
137.                int sum = sume[noduri[0]];
138.
139.                cout << "Pasul " << ++pas << ": ";
140.                for (i = 0; i < nrNod; i++)
141.                {
142.                    cout << nume[noduri[i]] << " [" << sume[noduri[i]] << "] ";
143.                }
144.                cout << endl;
145.                for (i = 0; i < nrNod - 1; i++)
146.                {
147.                    noduri[i] = noduri[i + 1];
148.                }
149.                nrNod--;
150.                if (nod == stop)
151.                {
152.                    gasit = 1;
153.                }
154.                else
155.                {
156.                    for (i = 0; i < 20; i++)
157.                    {
158.                        if ((a[nod][i] == 1) && (viz[i] == 0) || ((a[nod][i] == 1)
    && (viz[i] == 1) && (b[nod][i] > 0) && (b[nod][i] + sum < sume[i])))
159.                        {
160.                            noduri[nrNod++] = i;
161.                            //calculam costul pentru orasul i (suma orasului care a
    fost eliminat + distanta dintre orasul eliminat si cel gasit)
162.                            sume[i] = sum + b[nod][i];
163.                            viz[i] = 1;
164.                            parinte[i] = nod;
165.                        }
166.                    }
167.                    //ordonam vectorul noduri crescator dupa suma oraselor
168.                    for (i = 0; i < nrNod - 1; i++)
169.                    {
170.                        for (j = i + 1; j < nrNod; j++)
```

```
                        {
                            if (sume[noduri[i]] > sume[noduri[j]])
                            {
                                int aux = noduri[i];
                                noduri[i] = noduri[j];
                                noduri[j] = aux;
                            }
                        }
                    }
                }
            }
            while (final != start)
            {
                solutie[nrSol++] = final;
                final = parinte[final];
            }
            solutie[nrSol++] = start;
            cout << "Cautati drum de la " << nume[start] << " la " << nume[stop] << endl;

            for (i = nrSol - 1; i >= 0; i--)
            {
                cout << nume[solutie[i]] << " ";
            }
            cout << endl;
            //afisam suma orasului la care trebuie sa ajungem
            cout << "Costul cel mai mic este " << sume[stop] << ".";

            _getch();
            return 0;
        }
```

Program propus pentru căutarea cu cost uniform

Soluția propusă

Observație: am plecat de la programele primite și am făcut mici modificări și completări pentru a putea fi ușor de urmărit.

```
#include<iostream>
#include<string>
#define N 20

using namespace std;

string nume[N] = { "Satu Mare", "Baia Mare", "Oradea", "Cluj Napoca", "Bistrita", "Suceava", "Iasi", "Piatra Neamt", "Targu Mures", "Arad", "Timisoara", "Alba Iulia", "Sibiu", "Brasov", "Braila", "Tulcea", "Craiova", "Pitesti", "Bucuresti", "Constanta" };

int a[N][N];


void cautareCostUniform(int start, int stop){
    int viz[N], noduri[N], parinte[N];
    int nrNoduri = 0;
```

```
    for (int i = 0; i < N; i++)
        viz[i] = 0;   //marcam orasele ca nevizitate
    noduri[nrNoduri++] = start; //punem orasul de pornire in lista oraselor de vizitat
    viz[start] = 1;//marcam orasul de start ca vizitat
    bool gasit = false;//cand avem stop pe prima pozitie in noduri gasit devine adevarat
    int k = 1;//contor pentru a afisa pasii din popularea lui "noduri"
    bool afisPasi = true;//daca e fals nu mai afisam pasii intermediari
    int cost[N];//cost[oras] va contine distanta de la start la oras in km
    cost[start] = 0;//start e la 0 km de start

    while ((!gasit) && (nrNoduri > 0))
    {
        if (afisPasi)
        {//afisam componenta tabloului noduri doar daca afisPasi este adevarat
            cout << "Pasul " << k++ << " ";
            for (int i = 0; i < nrNoduri; i++)
                cout << nume[noduri[i]] << " "<<cost[noduri[i]]<<"  ";
            cout << endl;
        }
        int curent = noduri[0];    //primul element din noduri

        for (int i = 0; i < nrNoduri - 1; i++)  //eliminam elementul dupa prima pozitie din noduri
            noduri[i] = noduri[i + 1];
        nrNoduri--;

        if (curent == stop)
            gasit = true;
        else
            for (int i = 0; i < N; i++)
            if (a[curent][i] != 0)     //orasul i este conectat de orasul curent
            {
                int costNou = a[curent][i] + cost[curent];//calculam intai costul de la start la curent
                if ((viz[i] == 0) || (costNou < cost[i]))//(nu a fost vizitat) ori (a fost si in acest caz avem pentru el un cost calculat anterior, adica cost[i], ce se compara cu costul nou)
                {
                    if (viz[i] == 1)//daca a fost vizitat anterior, ar trebui sa eliminam vechea aparitie a lui i din noduri
                    {
                        int j = 0;
                        while ((j < nrNoduri) && (noduri[j] != i))
                            j++; //j reprezinta pozitia pe care se gaseste i in noduri
                        if (j < nrNoduri)//adica j e inca in noduri (fiindca e posibil sa fi fost scos anterior)
                        {
                            for (int q = j; q < nrNoduri - 1; q++)
                                noduri[q] = noduri[q + 1]; //am eliminat elementul dupa pozitia j din noduri
                            nrNoduri--;
                        }
                    }
                    //cautam pozitia pe care trebuie sa introducem j in "noduri"
                    int j = 0;
                    while ((j<nrNoduri) && (costNou>cost[noduri[j]]))//cautam in noduri pozitia pana la care costNou e mai mare decat costul elementului de pe pozitia curenta j
                        j++;    //j reprezinta pozitia pe care adaugam orasul i

                    //adaugam orasul i pe pozita j
                    for (int q = nrNoduri; q > j; q--
) //le deplasam pe toate de la j pana la capat la dreapta cu o pozitie
```

```
68.                         noduri[q] = noduri[q - 1];
69.                     noduri[j] = i;                          //j reperzinta poziti
    a pe care trebuie adaugat i
70.                     nrNoduri++;
71.                     cost[i] = costNou;
72.                     parinte[i] = curent;
73.                     viz[i] = 1;
74.                 }//de la if ((viz[i] == 0) || (costNou < cost[i]))
75.             }//de la if (a[curent][i] != 0)
76.     }   //acolada este de la while
77.
78.     cout << endl;
79.     int solutie[N], temp = stop, i = 0;
80.     while (temp != start){
81.         solutie[i++] = temp;
82.         temp = parinte[temp];
83.     }
84.     solutie[i++] = start;
85.     cout << endl << "Solutia folosind costul uniform este urmatoarea: ";
86.
87.     for (int j = i - 1; j >= 0; j--)
88.
89.         cout << nume[solutie[j]] << " " << cost[solutie[j]] << "  ";
90.     cout << endl;
91.
92.     cout << endl;
93. }
94.
95.
96. int main(){
97.
98.
99.
100.        for (int i = 0;i < N;i++)
101.        {
102.            for (int j = 0;j < N;j++) {
103.                a[i][j] = 0;
104.            }
105.        }
106.
107.            a[0][1]=62;
108.            a[0][2]=133;
109.            a[1][3]=148;
110.            a[2][3]=152;
111.            a[2][9]=118;
112.            a[3][9]=268;
113.            a[3][11]=98;
114.            a[3][8]=111;
115.            a[3][4]=116;
116.            a[4][5]=190;
117.            a[4][8]=97;
118.            a[5][6]=144;
119.            a[6][7]=130;
120.            a[7][13]=238;
121.            a[7][14]=255;
122.            a[7][18]=347;
123.            a[8][12]=115;
124.            a[8][13]=170;
125.            a[9][10]=54;
126.            a[10][11]=217;
127.            a[10][16]=341;
128.            a[11][16]=301;
129.            a[11][12]=77;
130.            a[12][13]=142;
131.            a[13][17]=137;
132.            a[13][18]=184;
```

```
133.            a[14][18]=218;
134.            a[14][15]=94;
135.            a[15][19]=125;
136.            a[16][17]=121;
137.            a[17][18]=117;
138.            a[18][19]=224;
139.
140.
141.            for (int i = 0;i < N;i++)
142.            {
143.                for (int j = 0;j < N;j++)
144.                {
145.                    if (a[i][j] == 1)
146.                        a[j][i] = 1;
147.                }
148.            }
149.
150.
151.            int start = 0, stop = 19;
152.
153.            cautareCostUniform(start, stop);
154.
155.        }
```

Exemplu de execuție a programului

start = Satu Mare, stop = Constanta

```
C:\Users\Cata\Documents\Classroom\CostUniform.exe
Pasul 1 Satu Mare 0
Pasul 2 Baia Mare 62  Oradea 133
Pasul 3 Oradea 133  Cluj Napoca 210
Pasul 4 Cluj Napoca 210  Arad 251
Pasul 5 Arad 251  Alba Iulia 308  Targu Mures 321  Bistrita 326
Pasul 6 Timisoara 305  Alba Iulia 308  Targu Mures 321  Bistrita 326
Pasul 7 Alba Iulia 308  Targu Mures 321  Bistrita 326  Craiova 646
Pasul 8 Targu Mures 321  Bistrita 326  Sibiu 385  Craiova 609
Pasul 9 Bistrita 326  Sibiu 385  Brasov 491  Craiova 609
Pasul 10 Sibiu 385  Brasov 491  Suceava 516  Craiova 609
Pasul 11 Brasov 491  Suceava 516  Craiova 609
Pasul 12 Suceava 516  Craiova 609  Pitesti 628  Bucuresti 675
Pasul 13 Craiova 609  Pitesti 628  Iasi 660  Bucuresti 675
Pasul 14 Pitesti 628  Iasi 660  Bucuresti 675
Pasul 15 Iasi 660  Bucuresti 675
Pasul 16 Bucuresti 675  Piatra Neamt 790
Pasul 17 Piatra Neamt 790  Constanta 899
Pasul 18 Constanta 899  Braila 1045


Solutia folosind costul uniform este urmatoarea: Satu Mare 0  Baia Mare 62
Cluj Napoca 210  Targu Mures 321  Brasov 491  Bucuresti 675  Constanta 899
```