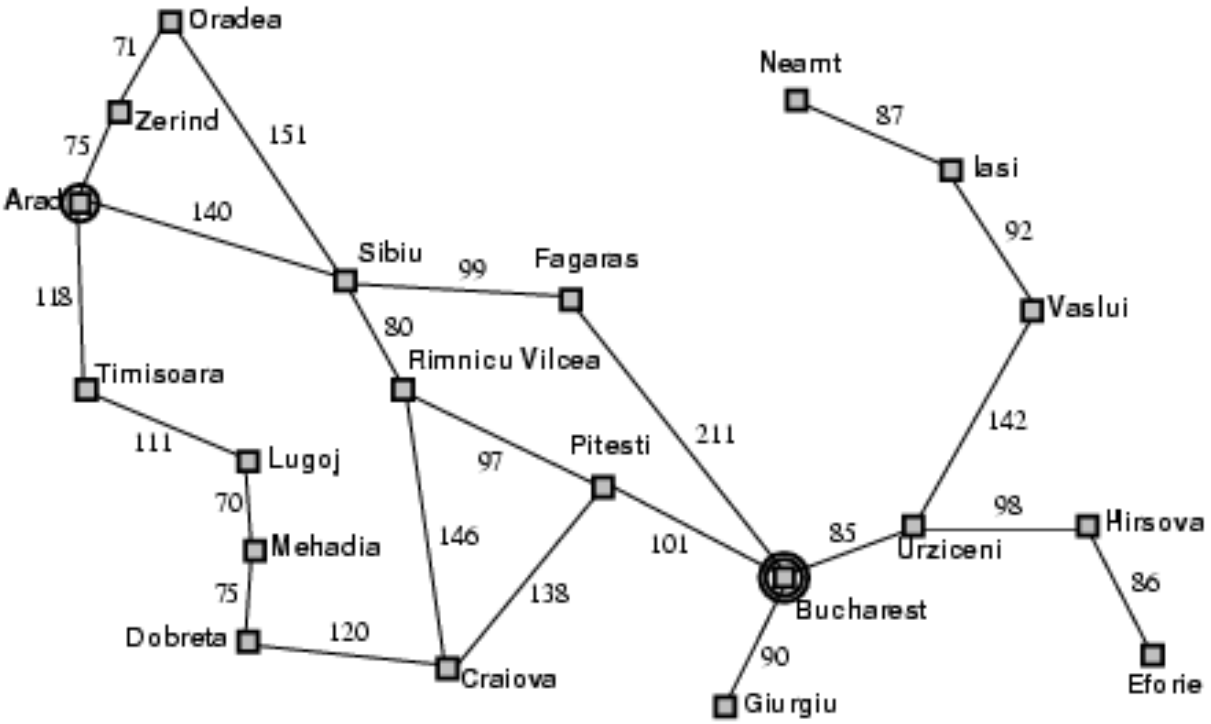
Algoritmul de căutare în adâncime - rezolvare

Fie harta de mai jos:



1. Implementați algoritmul de căutare in adâncime (adăugarea se face la început in lista *noduri* spre deosebire de căutarea in lățime).

Algoritm de căutare in adâncime
Toate orasele sunt nevizitate.
Adaugam in lista *noduri* orasul de plecare.
Marcam orasul de plecare ca vizitat.
Cat timp solutie negasita si *noduri* ≠ ∅ *executa*
    nod = scoate_din_fata(*noduri*) //stocam primul element din *noduri* in variabila *nod*
    Eliminam primul element din *noduri*
    Daca testare_tinta[problema] se aplica la stare(nod) *atunci*
        Solutia este gasita //facem variabila booleana gasit adevarata
    Altfel
        Adaugam la început in *noduri* orasele nevizitate care sunt conectate de *nod*
        Orasele adaugate sunt marcate ca vizitate
        Se retine pentru oricare din orasele adaugate nodul *parinte* ca fiind *nod*
Sfarsit cat timp
Stocam solutia parcurgand orasele de la destinatie catre start utilizand *parintii* retinuti.

Sugestie

Începeți pornind de la căutarea în lățime (faceți o copie funcției de căutare în lățime și o redenumiți adâncime) și modificați doar adăugarea nodului nou în *noduri*: în loc să îl puneți pe ultima poziție, va trebui să îl punem pe prima poziție prin mutarea tuturor elementelor la dreapta cu o poziție.

Programe primite ca soluții

În continuare, pun câteva soluții primite. Am solicitat ca la temă să fie incluse ambele căutări ca să le putem compara. Nu pun toate soluțiile primite fiindcă sunt foarte asemănătoare (nu era prea mult de modificat de la căutarea în lățime pentru a obține căutarea în adâncime).

Soluția 1

Observație: lățime și adâncime

```
#include <iostream>
#include <string>

using namespace std;

string v[20] = { "Arad","Bucuresti","Craiova", "Drobeta", "Eforie","Fagaras", "Giurgiu","Harsova","Iasi","Lugoj","Mehadia","Neamt","Oradea","Pitesti","RValcea","Sibiu","Timisoara","Urziceni","Vaslui","Zerind" };
int a[20][20];

void latime(int start, int stop) {
    int viz[20], noduri[20], nrnoduri = 0, parinte[20], ok = 0;
    noduri[nrnoduri++] = start;
    for (int i = 0;i < 20;i++)
     viz[i] = 0;
    viz[start] = 1;
    int pas = 0;
    while ((ok == 0) && (nrnoduri > 0)) {
     int nod = noduri[0];
     for (int i = 0;i < nrnoduri;i++)
         noduri[i] = noduri[i + 1];  //stergem elem de pe prima pozitie
     nrnoduri--;
     if (nod == stop) ok = 1;
     else
         for (int i = 0;i < 20;i++)
             if ((a[nod][i] == 1) && (viz[i] == 0))
             {
                 noduri[nrnoduri++] = i;  //adaugam nodul i pe ultima pozitie din noduri
                 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 << 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;
}

void adancime(int start, int stop)
{
    int noduri[20], nod, viz[20], nrnoduri = 0, parinte[20], ok = 0;
```

```
46.     for (int i = 0; i < 20; i++)
47.         viz[i] = 0;
48.     noduri[0] = start;
49.     nrnoduri++;
50.     viz[start] = 1;
51.     while ((ok== 0) && (nrnoduri > 0))
52.     {
53.         nod = noduri[0];
54.         for (int i = 0; i < 20; i++)
55.             noduri[i] = noduri[i + 1];
56.         nrnoduri--;
57.         if (nod == stop)
58.             ok=1;
59.         else
60.         {
61.             for (int i = 0; i < 20; i++)
62.                 if ((a[nod][i] == 1) && (viz[i]==0))
63.                 {
64.                     for (int j = nrnoduri; j > 0; j--)
65.                         noduri[j] = noduri[j - 1];
66.                     noduri[0] = i;
67.                     nrnoduri++;
68.                     viz[i] = 1;
69.                     parinte[i] = nod;
70.                 }
71.         }
72.     }
73.     int solutie[20], nrsol = 0, final = stop;
74.     while (final != start) {
75.      solutie[nrsol++] = final;
76.      final = parinte[final];
77.     }
78.     solutie[nrsol++] = start;
79.     cout << endl << endl << "Cautare in adancime de la " << v[start] << " la " <<
    v[stop] << endl;
80.     for (int i = nrsol - 1;i >= 0;i--)
81.      cout << v[solutie[i]] << " ";
82.     cout << endl;
83.
84. }
85.
86. int main()
87. {
88.     a[0][15] = 1;
89.     a[0][16] = 1;
90.     a[0][19] = 1;
91.     a[1][6] = 1;
92.     a[1][13] = 1;
93.     a[1][5] = 1;
94.     a[1][17] = 1;
95.     a[2][3] = 1;
96.     a[2][13] = 1;
97.     a[2][14] = 1;
98.     a[3][10] = 1;
99.     a[4][7] = 1;
100.          a[5][15] = 1;
101.          a[7][17] = 1;
102.          a[8][18] = 1;
103.          a[8][11] = 1;
104.          a[9][10] = 1;
105.          a[9][16] = 1;
106.          a[12][19] = 1;
107.          a[12][15] = 1;
108.          a[13][14] = 1;
109.          a[14][15] = 1;
110.          a[17][18] = 1;
```

```
            for (int i = 0; i < 20; i++)
                for (int j = 0; j < 20; j++)
                    if (a[i][j] == 1)
                        a[j][i] = 1;
            int b,c;
            cout<<"Introduceti nr oraselor separate printr-un spatiu: ";
            cin>>b>>c;
            cout<<"Orasele alese sunt: "<<v[b]<<" si "<<v[c]<<endl;
            adancime(b,c);
            for(int i=0;i<5;i++)
            cout<<endl;
            latime(b,c);
            for(int i=0;i<5;i++)
            cout<<endl;
        }
```

Soluția 2

Observație: soluție în C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

/// <summary>
/// NOTE: Explicatie OUTPUT plus observatie la linia 333
/// </summary>
namespace DFS
{
    /// <summary>
    /// Clasa nod ce reprezinta fiecare oras
    /// </summary>
    public class Nod
    {
        /// <summary>
        /// Numele orasului
        /// </summary>
        public string Nume { get; set; }

        /// <summary>
        /// Lista de noduri adiacente ale acestui nod
        /// </summary>
        public IList<Muchie> ListaDeAdiacenta { get; set; }

        /// <summary>
        /// Parintele acestui nod
        /// </summary>
        public Nod Parinte { get; set; }

        /// <summary>
        /// Constructor parametrizat
        /// </summary>
        /// <param name="nume"> Numele nodului </param>
        public Nod(string nume)
        {
            Nume = nume;
        }
    }

```

```
42.     /// <summary>
43.     /// Clasa muchie ce reprezinta legatura dintre doua orase( drumul in sine )
44.     /// </summary>
45.     public class Muchie
46.     {
47.         /// <summary>
48.         /// Nodul/Orasul de la capatul drumului
49.         /// </summary>
50.         public Nod NodTinta { get; set; }
51.
52.         /// <summary>
53.         /// Constructor parametrizat
54.         /// </summary>
55.         /// <param name="cost">Costul parcurgerii drumului</param>
56.         /// <param name="nodTinta">Nodul/Orasul de la capatul drumului</param>
57.         public Muchie(Nod nodTinta)
58.         {
59.             NodTinta = nodTinta;
60.         }
61.     }
62.
63.     class Program
64.     {
65.         /// <summary>
66.         /// Functia ce efectueaza cautarea in adancime
67.         /// </summary>
68.         /// <param name="start">Nodul de plecare</param>
69.         /// <param name="sosire">Nodul de sosire</param>
70.         public static void CautareInAdancime(Nod start, Nod sosire)
71.         {
72.             // O structura de date de tipul Stack
73.             // In care vom adauga orasele
74.             // Observatie: Natura structurii Stack ne perimite
75.             //             efectuarea cautarii in adancime intr-
    un mod sa zicem natural
76.             Stack<Nod> stiva = new Stack<Nod>();
77.
78.             // Adaugam nodul de start in capatul stivei
79.             stiva.Push(start);
80.
81.             // Creeam o structura de tipul multime, care tine evidenta nodurilor d
    eja vizitate
82.             // Aceasta nu poate contine elemente duplicat
83.             HashSet<Nod> noduriVizitate = new HashSet<Nod>();
84.
85.             // O variabila booleana care determina daca am ajuns la nodul de sosir
    e
86.             bool gasit = false;
87.
88.             do
89.             {
90.                 // Gasim nodul actual eliminandu-l de la capatul stivei
91.                 Nod nodActual = stiva.Pop();
92.
93.                 // Marcam acest nod ca vizitat
94.                 noduriVizitate.Add(nodActual);
95.
96.                 // Daca numele coincid...
97.                 if (nodActual.Nume == sosire.Nume)
98.                     // Atunci am ajuns la destinatie
99.                     gasit = true;
100.
101.                    // Parcurgem lista de adiacenta a nodului actual
102.                    foreach (Muchie muchie in nodActual.ListaDeAdiacenta)
103.                    {
104.                        // Cautam nodul de la celalalt capat al muchiei
```

```
105.                            // Adica nodul copil, care va fi initializat intr-
    o variabila noua
106.                            Nod nodCopil = muchie.NodTinta;
107.
108.                            // Daca nodCopil nu a mai fost vizitat si nu apartine c
    ozii...
109.                            if (!noduriVizitate.Contains(nodCopil) && !stiva.Contai
    ns(nodCopil))
110.                            {
111.                                // Marcam parintele lui nodCopil ca fiind nodActual
112.                                nodCopil.Parinte = nodActual;
113.
114.                                // Adaugam nodCopil in stiva
115.                                stiva.Push(nodCopil);
116.
117.                                // Afisam numele nodului copil
118.                                Console.WriteLine(nodCopil.Nume);
119.
120.                                // Afisam coada actuala
121.                                AfisareListaDeNoduri<Nod>(stiva);
122.
123.                                Console.WriteLine();
124.                            }
125.                        }
126.
127.                        // Parcurgem pana cand stiva este goala
128.                    } while (!(stiva.Count == 0));
129.
130.                }
131.
132.                /// <summary>
133.                /// Functia care ne returneaza drumul
134.                /// </summary>
135.                /// <param name="sosire">Orasul de sosire</param>
136.                /// <returns></returns>
137.                private static IList<Nod> Drum(Nod sosire)
138.                {
139.                    IList<Nod> drum = new List<Nod>();
140.
141.                    for (Nod nod = sosire; nod != null; nod = nod.Parinte)
142.                    {
143.                        drum.Add(nod);
144.                    }
145.
146.                    return drum.Reverse().ToList();
147.                }
148.
149.                /// <summary>
150.                /// Functia care afiseaza drumul
151.                /// </summary>
152.                /// <param name="drum">Drumul parcurs</param>
153.                public static void AfisareDrum(IList<Nod> drum)
154.                {
155.                    foreach (var nod in drum)
156.                    {
157.                        Console.Write(nod.Nume + "-> ");
158.                    }
159.                }
160.
161.                /// <summary>
162.                /// Functia care afiseaza o lista oarecare de noduri
163.                /// </summary>
164.                /// <typeparam name="T"></typeparam>
165.                /// <param name="lista"></param>
```

```
166.                public static void AfisareListaDeNoduri<T>(IEnumerable<T> lista) wh
    ere T : Nod
167.                {
168.                    foreach (var nod in lista)
169.                    {
170.                        Console.Write(nod.Nume);
171.                    }
172.                }
173.
174.                static void Main(string[] args)
175.                {
176.                    //initializam orasele de pe harta ca fiind noduri
177.                    Nod n1 = new Nod("Arad");
178.                    Nod n2 = new Nod("Zerind");
179.                    Nod n3 = new Nod("Oradea");
180.                    Nod n4 = new Nod("Sibiu");
181.                    Nod n5 = new Nod("Fagaras");
182.                    Nod n6 = new Nod("Rimnicu Vilcea");
183.                    Nod n7 = new Nod("Pitesti");
184.                    Nod n8 = new Nod("Timisoara");
185.                    Nod n9 = new Nod("Lugoj");
186.                    Nod n10 = new Nod("Mehadia");
187.                    Nod n11 = new Nod("Drobeta");
188.                    Nod n12 = new Nod("Craiova");
189.                    Nod n13 = new Nod("Bucharest");
190.                    Nod n14 = new Nod("Giurgiu");
191.                    Nod n15 = new Nod("Urziceni");
192.                    Nod n16 = new Nod("Hirsova");
193.                    Nod n17 = new Nod("Eforie");
194.                    Nod n18 = new Nod("Vaslui");
195.                    Nod n19 = new Nod("Iasi");
196.                    Nod n20 = new Nod("Neamt");
197.
198.                    //initializam muchiile
199.                    n1.ListaDeAdiacenta = new List<Muchie>
200.                    {
201.                        new Muchie(n2),
202.                        new Muchie(n4),
203.                        new Muchie(n8)
204.                    };
205.
206.                    n2.ListaDeAdiacenta = new List<Muchie>
207.                    {
208.                        new Muchie(n1),
209.                        new Muchie(n3)
210.                    };
211.
212.                    n3.ListaDeAdiacenta = new List<Muchie>
213.                    {
214.                        new Muchie(n2),
215.                        new Muchie(n4)
216.                    };
217.
218.                    n4.ListaDeAdiacenta = new List<Muchie>
219.                    {
220.                        new Muchie(n1),
221.                        new Muchie(n5),
222.                        new Muchie(n3),
223.                        new Muchie(n6),
224.                    };
225.
226.                    n5.ListaDeAdiacenta = new List<Muchie>
227.                    {
228.                        new Muchie(n4),
229.                        new Muchie(n13)
230.                    };
```

```
231.
232.                    n6.ListaDeAdiacenta = new List<Muchie>
233.                    {
234.                        new Muchie(n4),
235.                        new Muchie(n7),
236.                        new Muchie(n12)
237.                    };
238.
239.                    n7.ListaDeAdiacenta = new List<Muchie>
240.                    {
241.                        new Muchie(n6),
242.                        new Muchie(n13),
243.                        new Muchie(n12)
244.                    };
245.
246.                    n8.ListaDeAdiacenta = new List<Muchie>
247.                    {
248.                        new Muchie(n1),
249.                        new Muchie(n9)
250.                    };
251.
252.                    n9.ListaDeAdiacenta = new List<Muchie>
253.                    {
254.                        new Muchie(n8),
255.                        new Muchie(n10)
256.                    };
257.
258.                    n10.ListaDeAdiacenta = new List<Muchie>
259.                    {
260.                        new Muchie(n9),
261.                        new Muchie(n11)
262.                    };
263.
264.                    n11.ListaDeAdiacenta = new List<Muchie>
265.                    {
266.                        new Muchie(n10),
267.                        new Muchie(n12)
268.                    };
269.
270.                    n12.ListaDeAdiacenta = new List<Muchie>
271.                    {
272.                        new Muchie(n11),
273.                        new Muchie(n6),
274.                        new Muchie(n7)
275.                    };
276.
277.                    n13.ListaDeAdiacenta = new List<Muchie>
278.                    {
279.                        new Muchie(n7),
280.                        new Muchie(n14),
281.                        new Muchie(n5),
282.                        new Muchie(n15)
283.                    };
284.
285.                    n14.ListaDeAdiacenta = new List<Muchie>
286.                    {
287.                        new Muchie(n13)
288.                    };
289.
290.                    n15.ListaDeAdiacenta = new List<Muchie>
291.                    {
292.                        new Muchie(n13),
293.                        new Muchie(n16),
294.                        new Muchie(n18)
295.                    };
296.
```

```
297.                    n16.ListaDeAdiacenta = new List<Muchie>
298.                    {
299.                        new Muchie(n15),
300.                        new Muchie(n17)
301.                    };
302.
303.                    n17.ListaDeAdiacenta = new List<Muchie>
304.                    {
305.                        new Muchie(n16)
306.                    };
307.
308.                    n18.ListaDeAdiacenta = new List<Muchie>
309.                    {
310.                        new Muchie(n15),
311.                        new Muchie(n19)
312.                    };
313.
314.                    n19.ListaDeAdiacenta = new List<Muchie>
315.                    {
316.                        new Muchie(n18),
317.                        new Muchie(n20)
318.                    };
319.
320.                    n20.ListaDeAdiacenta = new List<Muchie>
321.                    {
322.                        new Muchie(n19)
323.                    };
324.
325.                    // Efectuam cautarea in adancime
326.                    CautareInAdancime(n1, n13);
327.
328.                    // Gasim drumul
329.                    IList<Nod> drum = Drum(n13);
330.
331.                    // Afisam acel drum
332.                    // OUTPUT: Arad-> Timisoara-> Lugoj-> Mehadia-> Drobeta-
    > Craiova-> Pitesti-> Bucharest->
333.                    // Explicatie OUTPUT: Datorita faptului ca noi parcurgem in ada
    ncime(mereu in jos)
334.                    // Stiva in care am adaugat orasele ne permite sa parcurgem 1 c
    ate unul(omitand temporar celelalte noduri
335.                    // din lista de adiacenta). Ex Avem Arad are ca orase adiacente
     pe Zerind, Sibiu si Timisoara
336.                    // Deoarece Timisoara este ultimul din lista de adiacenta( in c
    azul aceste cel putin)
337.                    // Acesta va fi in capatul stivei, iar la urmatoarea iteratie d
    in stiva acesta va scos din ea
338.                    // Si astfel va devenii nod actual, iar procesul se va tot repe
    ta...
339.                    AfisareDrum(drum);
340.
341.                    Console.ReadKey();
342.                }
343.            }
344.        }
```