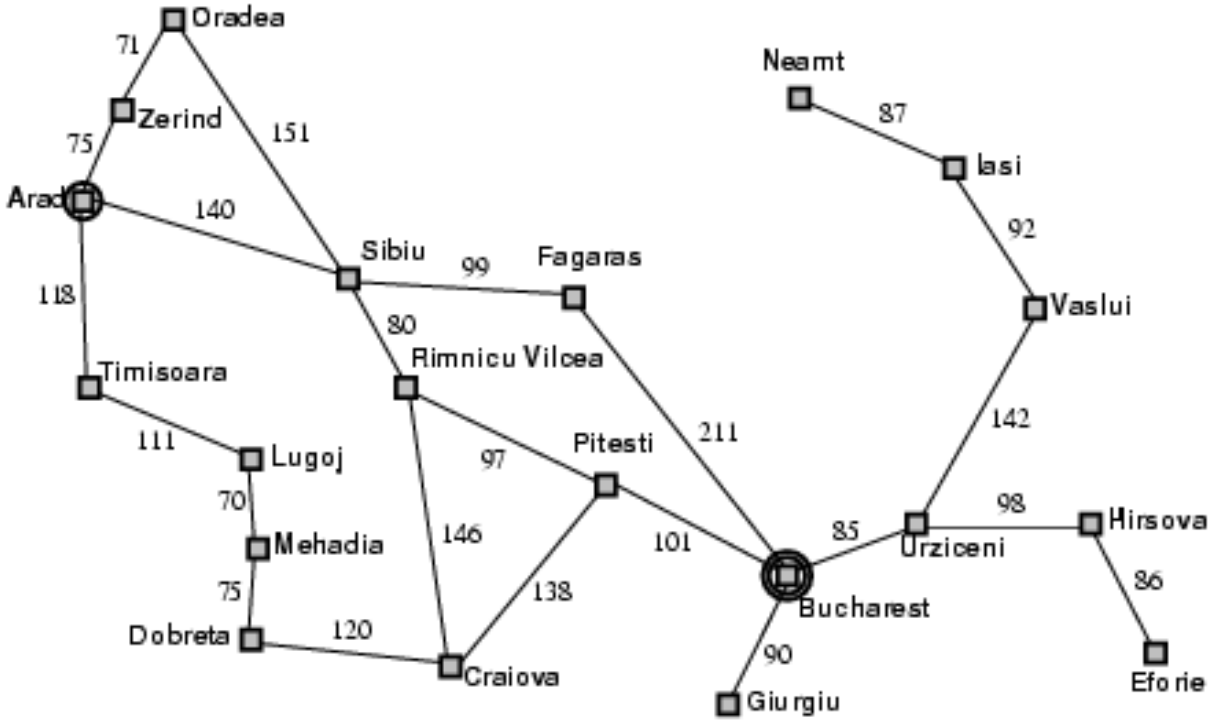
Algoritmii de în lățime, adâncime și cost uniform care cuprind costurile în km și timpii rutelor în ore/minute

Fie harta de mai jos:



Știind ca ne deplasam in medie cu 80 km/h si folsind numarul de km intre orase, scrieti un program care sa calculeze timpul de deplasare pe ruta gasita pentru fiecare din cautarile in latime, adancime si cost uniform.

Soluție

Iată câteva rezolvări primite, funcționale.

Soluția 1

Observație: timpii sunt calculați în minute. Un nou tablou „timp” de tip float e adăugat la fiecare metodă de căutare.

```
#include<iostream>
#include<string>
#define N 20
#include <math.h>
using namespace std;

string nume[N] = { "Arad","Bucuresti","Craiova", "Severin", "Eforie","Fagaras",
"Giurgiu","Harsova","Iasi","Lugoj","Mehadia","Neamt","Oradea","Pitesti","Vilcea","Si
biu","Timisoara","Urziceni","Vaslui","Zerind" };

int a[N][N];


void cautareCostUniform(int start, int stop) {
	int viz[N], noduri[N], parinte[N];
	float t;
	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 = false;//daca e fals nu mai afisam pasii intermediari
	float cost[N], timp[N];//cost[oras] va contine distanta de la start la oras
in km
	cost[start] = 0;//start e la 0 km de start
	timp[start] = 0;
	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]] << "
timp " << ((cost[noduri[i]] / 80) * 60);
			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
				{
					float costNou = a[curent][i] + cost[curent];//calculam intai costulde 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 elementuldupa 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
							noduri[q] = noduri[q - 1];
						noduri[j] = i; //j reperzinta pozitia pe care trebuie adaugat i
						nrNoduri++;
						cost[i] = costNou;
						t = (cost[i] / 80) * 60;
						timp[i] = t;
						parinte[i] = curent;
						viz[i] = 1;
					}//de la if ((viz[i] == 0) || (costNou < cost[i]))
				}//de la if (a[curent][i] != 0)
		} //acolada este de la while

		int solutie[N], temp = stop, i = 0;
		while (temp != start) {
			solutie[i++] = temp;
			temp = parinte[temp];
		}
		solutie[i++] = start;
		cout << endl << "Solutia folosind costul uniform este urmatoarea: " << endl;
```

```
	for (int j = i - 1; j >= 0; j--)

		cout << nume[solutie[j]] << " " << cost[solutie[j]] << " km, timp de
parcurgere " << timp[solutie[j]] << " minute; ";
	cout << endl;

	cout << endl;
}

void latime(int start, int stop)
{
	int viz[20], noduri[20], nrnoduri = 0, parinte[20], ok = 0;
	float timp[20], t, cost[20];
	for (int i = 0; i < 20; i++) {
		cost[i] = 0;
		timp[i] = 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] != 0) && (viz[i] == 0))
				{
					noduri[nrnoduri++] = i; //adaugam nodul i pe
ultima pozitie din noduri
					viz[i] = 1;
					parinte[i] = nod;
					cost[i] = cost[nod] + a[nod][i];

					t = (cost[i] / 80) * 60;
					timp[i] = t;
				}
	}
	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 " << nume[start] << " la "
<< nume[stop] << endl;
	for (int i = nrsol - 1; i >= 0; i--)
		cout << nume[solutie[i]] << " " << cost[solutie[i]] << " " << "km, timp
de parcurgere " << timp[solutie[i]] << " minute;";
	cout << endl;

}
void adancime(int start, int stop)
{
```

```
	int noduri[20], nod, viz[20], nrnoduri = 0, parinte[20], ok = 0;
	float timp[20], t, cost[20];
	for (int i = 0; i < 20; i++)
		viz[i] = 0;
	for (int i = 0; i < 20; i++) {

		cost[i] = 0;
		timp[i] = 0;
	}
	noduri[0] = start;
	nrnoduri++;
	viz[start] = 1;
	while ((ok == 0) && (nrnoduri > 0))
	{
		nod = noduri[0];
		for (int i = 0; i < 20; i++)
			noduri[i] = noduri[i + 1];
		nrnoduri--;
		if (nod == stop)
			ok = 1;
		else
		{
			for (int i = 0; i < 20; i++)
				if ((a[nod][i] != 0) && (viz[i] == 0))
				{
					for (int j = nrnoduri; j > 0; j--)
						noduri[j] = noduri[j - 1];
					noduri[0] = i;
					nrnoduri++;
					viz[i] = 1;
					parinte[i] = nod;
					cost[i] = cost[parinte[i]] + a[nod][i];
					t = (cost[i] / 80) * 60;
					timp[i] = t;
				}
		}
	}
	int solutie[20], nrsol = 0, final = stop;
	while (final != start)
	{
		solutie[nrsol++] = final;
		final = parinte[final];
	}
	solutie[nrsol++] = start;
	cout << endl << endl << "Cautare in adancime de la " << nume[start] << " la "
<< nume[stop] << endl;
	for (int i = nrsol - 1; i >= 0; i--)
		cout << nume[solutie[i]] << " " << cost[solutie[i]] << "km, timp de
parcugere " << timp[solutie[i]] << " minute; ";
	cout << endl;
}



int main() {



	for (int i = 0; i < N; i++)
		for (int j = 0; j < N; j++)
			a[i][j] = 0;
```

```
        a[0][19] = 75;
        a[0][15] = 140;
        a[0][16] = 118;
        a[1][6] = 90;
        a[1][13] = 101;
        a[1][5] = 211;
        a[1][17] = 85;
        a[2][13] = 138;
        a[2][14] = 146;
        a[2][3] = 120;
        a[3][10] = 75;
        a[4][7] = 80;
        a[5][15] = 99;
        a[7][17] = 98;
        a[8][11] = 87;
        a[8][18] = 92;
        a[9][10] = 70;
        a[9][16] = 111;
        a[12][19] = 71;
        a[12][15] = 151;
        a[13][14] = 97;
        a[14][15] = 80;
        a[17][18] = 142;

        for (int i = 0; i < N; i++)
        {
                for (int j = 0; j < N; j++)
                {
                        if (a[i][j] != 0)
                                a[j][i] = a[i][j];
                }
        }


        int start = 0, stop = 1;
        latime(start, stop);
        adancime(start, stop);
        cautareCostUniform(start, stop);
}
```

Programul rulat

```
Cautare in latime de la Arad la Bucuresti
Arad 0 km, timp de parcurgere 0 minute;Sibiu 140 km, timp de parcurgere 105 minute;Fagaras 239 km, timp
de parcurgere 179.25 minute;Bucuresti 450 km, timp de parcurgere 337.5 minute;


Cautare in adancime de la Arad la Bucuresti
Arad 0km, timp de parcugere 0 minute; Timisoara 118km, timp de parcugere 88.5 minute; Lugoj 229km, timp
de parcugere 171.75 minute; Mehadia 299km, timp de parcugere 224.25 minute; Severin 374km, timp de parcu
gere 280.5 minute; Craiova 494km, timp de parcugere 370.5 minute; Pitesti 632km, timp de parcugere 474 m
inute; Bucuresti 733km, timp de parcugere 549.75 minute;

Solutia folosind costul uniform este urmatoarea:
Arad 0 km, timp de parcurgere 0 minute; Sibiu 140 km, timp de parcurgere 105 minute; Vilcea 220 km, timp
 de parcurgere 165 minute; Pitesti 317 km, timp de parcurgere 237.75 minute; Bucuresti 418 km, timp de p
arcurgere 313.5 minute;
```

Soluția 2

Observație: afișare elegantă a timpilor rutelor în ore și secunde.

```
#include<iostream>
#include<string>
#define N 20
#include <math.h>
	using namespace std;

string nume[N] = { "Arad","Bucuresti","Craiova", "Severin", "Eforie","Fagaras",
"Giurgiu","Harsova","Iasi","Lugoj","Mehadia","Neamt","Oradea","Pitesti","Vilcea","Si
biu","Timisoara","Urziceni","Vaslui","Zerind" };

int a[N][N];


void cautareCostUniform(int start, int stop) {
	int viz[N], noduri[N], parinte[N];
	float t;
	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 = false;//daca e fals nu mai afisam pasii intermediari
	float cost[N], timp[N];//cost[oras] va contine distanta de la start la oras
in km
	cost[start] = 0;//start e la 0 km de start
	timp[start] = 0;
	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]] << "
timp " << ((cost[noduri[i]] / 80) * 60);
			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
				{
					float costNou = a[curent][i] +
cost[curent];//calculam intai costulde 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 elementuldupa 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
                                                noduri[q] = noduri[q - 1];
                                        noduri[j] = i; //j reperzinta pozitia pe
care trebuie adaugat i
                                        nrNoduri++;
                                        cost[i] = costNou;
                                        t = (cost[i] / 80) * 60;
                                        timp[i] = t;
                                        parinte[i] = curent;
                                        viz[i] = 1;
                                }//de la if ((viz[i] == 0) || (costNou < cost[i]))
                        }//de la if (a[curent][i] != 0)
        } //acolada este de la while

        int solutie[N], temp = stop, i = 0;
        while (temp != start) {
                solutie[i++] = temp;
                temp = parinte[temp];
        }
        solutie[i++] = start;
        cout << endl << "Solutia folosind costul uniform este urmatoarea: " << endl;

        for (int j = i - 1; j >= 0; j--)

                cout << nume[solutie[j]] << " " << cost[solutie[j]] << " km, timp de
parcurgere " << timp[solutie[j]] << " minute; ";
        cout << endl;
```

```
	cout << endl;
}

void latime(int start, int stop)
{
	int viz[20], noduri[20], nrnoduri = 0, parinte[20], ok = 0;
	float timp[20], t, cost[20];
	for (int i = 0; i < 20; i++) {
		cost[i] = 0;
		timp[i] = 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] != 0) && (viz[i] == 0))
				{
					noduri[nrnoduri++] = i; //adaugam nodul i pe
ultima pozitie din noduri
					viz[i] = 1;
					parinte[i] = nod;
					cost[i] = cost[nod] + a[nod][i];

					t = (cost[i] / 80) * 60;
					timp[i] = t;
				}
	}
	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 " << nume[start] << " la "
<< nume[stop] << endl;
	for (int i = nrsol - 1; i >= 0; i--)
		cout << nume[solutie[i]] << " " << cost[solutie[i]] << " " << "km, timp
de parcurgere " << timp[solutie[i]] << " minute;";
	cout << endl;

}
void adancime(int start, int stop)
{
	int noduri[20], nod, viz[20], nrnoduri = 0, parinte[20], ok = 0;
	float timp[20], t, cost[20];
	for (int i = 0; i < 20; i++)
		viz[i] = 0;
	for (int i = 0; i < 20; i++) {

		cost[i] = 0;
```

```
		timp[i] = 0;
	}
	noduri[0] = start;
	nrnoduri++;
	viz[start] = 1;
	while ((ok == 0) && (nrnoduri > 0))
	{
		nod = noduri[0];
		for (int i = 0; i < 20; i++)
			noduri[i] = noduri[i + 1];
		nrnoduri--;
		if (nod == stop)
			ok = 1;
		else
		{
			for (int i = 0; i < 20; i++)
				if ((a[nod][i] != 0) && (viz[i] == 0))
				{
					for (int j = nrnoduri; j > 0; j--)
						noduri[j] = noduri[j - 1];
					noduri[0] = i;
					nrnoduri++;
					viz[i] = 1;
					parinte[i] = nod;
					cost[i] = cost[parinte[i]] + a[nod][i];
					t = (cost[i] / 80) * 60;
					timp[i] = t;
				}
		}
	}
	int solutie[20], nrsol = 0, final = stop;
	while (final != start)
	{
		solutie[nrsol++] = final;
		final = parinte[final];
	}
	solutie[nrsol++] = start;
	cout << endl << endl << "Cautare in adancime de la " << nume[start] << " la "
<< nume[stop] << endl;
	for (int i = nrsol - 1; i >= 0; i--)
		cout << nume[solutie[i]] << " " << cost[solutie[i]] << "km, timp de
parcugere " << timp[solutie[i]] << " minute; ";
	cout << endl;
}


int main() {



	for (int i = 0; i < N; i++)
		for (int j = 0; j < N; j++)
			a[i][j] = 0;

	a[0][19] = 75;
	a[0][15] = 140;
	a[0][16] = 118;
	a[1][6] = 90;
	a[1][13] = 101;
	a[1][5] = 211;
	a[1][17] = 85;
```

```
        a[2][13] = 138;
        a[2][14] = 146;
        a[2][3] = 120;
        a[3][10] = 75;
        a[4][7] = 80;
        a[5][15] = 99;
        a[7][17] = 98;
        a[8][11] = 87;
        a[8][18] = 92;
        a[9][10] = 70;
        a[9][16] = 111;
        a[12][19] = 71;
        a[12][15] = 151;
        a[13][14] = 97;
        a[14][15] = 80;
        a[17][18] = 142;

        for (int i = 0; i < N; i++)
        {
                for (int j = 0; j < N; j++)
                {
                        if (a[i][j] != 0)
                                a[j][i] = a[i][j];
                }
        }


        int start = 0, stop = 1;
        latime(start, stop);
        adancime(start, stop);
        cautareCostUniform(start, stop);
}
```

Programul rulat

```
Arad 0 Sibiu 140 Fagaras 239 Bucuresti 450
Folosind algoritmul de cautare in latime ,timpul de deplasare pe ruta gasita este de apoxima
tiv 5 ore si 37 minute.

Cautare in adancime de la Arad la Bucuresti
Arad 0 Timisoara 118 Lugoj 229 Mehadia 299 Severin 374 Craiova 494 Pitesti 632 Bucuresti 733

Folosind algoritmul de cautare in adancime ,timpul de deplasare pe ruta gasita este de aprox
imativ 9 ore si 9 minute.

Cautare in adancime cu descendantii in ordine alfabetica de la Arad la Bucuresti
Arad 0 Sibiu 140 Fagaras 239 Bucuresti 450
Folosind algoritmul de cautare in adancime cu descendentii in ordine alfabetica,timpul de de
plasare pe ruta gasita este de aproximativ 5 ore si 37 minute.

Solutia folosind costul uniform este urmatoarea:
Arad 0 Sibiu 140 Vilcea 220 Pitesti 317 Bucuresti 418
Folosind algoritmul pentru cautarea cu cost uniform, timpul de deplasare pe ruta gasita este
 de aproximativ 5 ore si 13 minute.
```

Soluția 3

Observație: Soluție cu obiecte. Are totuși o problemă la hartă cred, am observat că la căutarea în adâncime nu oferă ruta corectă pentru harta noastră (AR-SB-VL-Pitesti-B). Iar timpul este calculat în ore, cu zecimale, ceea ce este mai dificil de urmărit.

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

using namespace std;
class FindPath {
private:
	string names[20] = {
"Arad","Bucuresti","Craiova","Drobeta","Eforie","Fagaras","Giurgiu","Harsova",+
		"Iasi","Lugoj","Mehadia","Neamt","Oradea","Pitesti","Ramnicul
Valcea","Sibiu","Timisoara","Urziceni","Vaslui","Zerind" };
	int a[20][20];
	int parents[20];
	int startNode;
	int targetNode;

public:
	void initializeAdiacenceMatrix() {
		for (int i = 0; i < 20; i++) {
			for (int j = 0; j < 20; j++) {
				a[i][j] = 0;
			}
		}
		a[0][19] = 75;
		a[0][15] = 140;
		a[0][16] = 118;
		a[1][6] = 90;
		a[1][13] = 101;
		a[1][5] = 211;
		a[1][17] = 85;
		a[2][13] = 138;
		a[2][14] = 146;
		a[2][3] = 120;
		a[3][10] = 75;
		a[4][7] = 80;
		a[5][15] = 99;
		a[7][17] = 98;
		a[8][18] = 92;
		a[8][11] = 87;
		a[9][10] = 70;
		a[9][16] = 111;
		a[12][19] = 71;
		a[12][15] = 151;
		a[13][14] = 97;
		a[14][15] = 80;
		a[17][18] = 142;

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

	void readStartEndCities() {
		cout << "Introduceti numarul corespunzator orasului de plecare: " <<
endl;
		for (int i = 0; i < 20; i++) {
```

```
                        cout << i << " - " << names[i] << endl;
                }
                cin >> this->startNode;

                cout << "Introduceti numarul corespunzator orasului de sosire: " <<
endl;
                for (int i = 0; i < 20; i++) {
                        if (i != this->startNode) {
                                cout << i << " - " << names[i] << endl;
                        }
                }
                cin >> this->targetNode;
        }

        void breathSearch() {
                int visitedCities[20];
                int nodesTail[20];
                int nodesTailPos = 0;
                bool solutionFound = false;

                for (int i = 0; i < 20; i++) {
                        visitedCities[i] = 0;
                        parents[i] = -1;
                }

                nodesTail[nodesTailPos] = startNode;
                nodesTailPos++;
                visitedCities[startNode] = 1;

                while (!solutionFound && nodesTailPos != 0) {
                        int node = nodesTail[0];

                        for (int i = 0; i < nodesTailPos; i++) {
                                nodesTail[i] = nodesTail[i + 1];
                        }
                        nodesTailPos--;

                        if (node == targetNode) {
                                solutionFound = true;
                        }
                        else {
                                for (int i = 0; i < 20; i++) {
                                        if (a[i][node] != 0 && visitedCities[i] == 0) {
                                                nodesTail[nodesTailPos] = i;
                                                nodesTailPos++;
                                                parents[i] = node;
                                                visitedCities[i] = 1;
                                        }
                                }
                        }
                }
        }

        void depthSearch() {
                int visitedCities[20];
                int nodesStack[20];
                int nodesStackPos = 0;
                bool solutionFound = false;

                for (int i = 0; i < 20; i++) {
                        visitedCities[i] = 0;
                        parents[i] = -1;
```

```
		}
		nodesStack[nodesStackPos] = startNode;
		nodesStackPos++;
		visitedCities[startNode] = 1;

		while (!solutionFound) {
			int node = nodesStack[nodesStackPos - 1];
			nodesStackPos--;

			if (node == targetNode) {
				solutionFound = true;
			}
			else
			{
				for (int i = 0; i < 20; i++) {
					if (a[i][node] != 0 && visitedCities[i] == 0) {
						nodesStack[nodesStackPos] = i;
						nodesStackPos++;
						parents[i] = node;
						visitedCities[i] = 1;
					}
				}
			}
		}
	}

	void costUniform() {
		int cost[20];
		int nodesTail[20];
		int nodesTailPos = 0;
		bool solutionFound = false;

		for (int i = 0; i < 20; i++) {
			cost[i] = 0;
			parents[i] = -1;
		}

		nodesTail[nodesTailPos] = startNode;
		nodesTailPos++;
		cost[startNode] = 0;

		while (!solutionFound && nodesTailPos != 0) {
			int node = nodesTail[0];

			for (int i = 0; i < nodesTailPos; i++) {
				nodesTail[i] = nodesTail[i + 1];
			}
			nodesTailPos--;

			if (node == targetNode) {
				solutionFound = true;
			}
			else {
				for (int i = 0; i < 20; i++) {
					if (a[i][node] != 0 && (cost[i] == 0 || cost[i] >
cost[node] + a[i][node])) {
						nodesTail[nodesTailPos] = i;
						nodesTailPos++;
						parents[i] = node;
						cost[i] = cost[node] + a[i][node];
					}
				}
```

```
                    }
                }
        }

        void printSolution() {
                this->printSolutionRec(targetNode);
                cout << endl;
        }

        void printTime() {
                int dist = 0;

                int node = targetNode;

                while (node != startNode) {
                        dist += a[node][parents[node]];
                        node = parents[node];
                }

                cout << "Time = " << dist / 80.0 << " hours" << endl;
        }

private:
        void printSolutionRec(int node) {
                if (node != startNode) {
                        printSolutionRec(parents[node]);
                }

                cout << names[node] << " ";
        }
};

int main()
{
        FindPath* findPath = new FindPath();
        findPath->readStartEndCities();
        findPath->initializeAdiacenceMatrix(); // initializeaza matricea de adiacenta

        cout << "Cautare in latime" << endl;
        findPath->breathSearch(); // cauta in latime
        findPath->printSolution(); // afiseaza solutia daca exista, altfel afiseaza
un mesaj ca solutia nu exista
        findPath->printTime();

        cout << "Cautare in adancime" << endl;
        findPath->depthSearch(); // cauta solutie in adancime
        findPath->printSolution();
        findPath->printTime();

        cout << "Cautare Cost Uniform" << endl;
        findPath->costUniform(); // cauta solutie cost uniform
        findPath->printSolution();
        findPath->printTime();
}
```

Programul rulat

```
Cautare in latime
Arad Sibiu Fagaras Bucuresti
Time = 5.625 hours
Cautare in adancime
Arad Timisoara Lugoj Mehadia Drobeta Craiova Pitesti Bucuresti
Time = 9.1625 hours
Cautare Cost Uniform
Arad Sibiu Fagaras Bucuresti
Time = 5.625 hours
```