Căutare bidirecțională

Implementați un algoritm de căutare bidirecțională astfel încât să avem combinații din următoarele căutări:

- Lățime
- Adâncime.

Instrucțiuni legate de căutarea bidirecțională:

1. Programul va porni concomitent de la nodul de start cu căutare în lățime și de la cel de stop cu căutare în adâncime.
2. Va trebui să rețineți câte două tablouri distincte de *noduri*, *vizitate, parinte,* o serie pentru căutarea de la *start* și una pentru cea de la *stop*.
3. Vă opriți când un nod din lista de *vizitate1* se găsește și în lista de *vizitate2*. Atunci se poate construi soluția finală concatenând lista de la *start* la nodul comun (dintre *vizitate1* și *vizitate2*) cu cea de la nodul comun la stop.

Soluție

Iată una din soluțiile primite.

Soluție

Observații: programul conține doar căutarea bidirecțională și afișează pașii intermediari pentru fiecare dintre cele două căutări.

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

using namespace std;

string nume[20] = {
"Oradea","Zerind","Arad","Timisoara","Lugoj","Mehadia","Drobeta","Craiova
","Sibiu","Ramnicu
Valcea","Fagaras","Pitesti","Bucuresti","Giurgiu","Urziceni","Hrisova","E
forie","Vaslui","Iasi","Neamt" };

int distante[20][20], i, j;

void cautareBidirectionala(int start, int stop) {
        int viz1[20], noduri1[20], nrNod1 = 0, parinte1[20], gasit1 = 0,
pas1 = 0, solutie1[20], nrSol1 = 0, final1;
        int viz2[20], noduri2[20], nrNod2 = 0, parinte2[20], gasit2 = 0,
pas2 = 0, solutie2[20], nrSol2 = 0, final2;
        final1 = stop;
```

```
	final2 = start;
	for (i = 0; i < 20; i++)
	{
		viz1[i] = 0;
		viz2[i] = 0;
	}

	noduri1[nrNod1++] = start;
	viz1[start] = 1;
	noduri2[nrNod2++] = stop;
	viz2[stop] = 1;

	int okaius = 0;
	while (okaius != 1)
	{
		int nod1 = noduri1[0];

		if (gasit1 == 0)
		{
			cout << "Pasul[latime] " << ++pas1 << ": ";
			for (i = 0; i < nrNod1; i++)
			{
				cout << nume[noduri1[i]] << " ";
			}
			cout << endl;
		}
		for (i = 0; i < nrNod1 - 1; i++)
		{
			noduri1[i] = noduri1[i + 1];
		}
		nrNod1--;
		if (nod1 == stop)
		{
			gasit1 = 1;
		}
		else
		{
			for (i = 0; i < 20; i++)
			{
				if ((distante[nod1][i] != 0) && (viz1[i] ==
0))
				{
					noduri1[nrNod1++] = i;
					viz1[i] = 1;
					parinte1[i] = nod1;
				}
			}
		}
		int nod2 = noduri2[0];
		cout << "Pasul[adancime] " << ++pas2 << ": ";
		for (i = 0; i < nrNod2; i++)
		{
			cout << nume[noduri2[i]] << " ";
		}
		cout << endl;
		for (i = 0; i < nrNod2 - 1; i++)
		{
```

```
                        noduri2[i] = noduri2[i + 1];
                }
                nrNod2--;
                if (nod2 == start)
                {
                        gasit2 = 1;
                }
                else
                {
                        for (i = 0; i < 20; i++)
                        {
                                if ((distante[nod2][i] != 0) && (viz2[i] ==
0))
                                {
                                        for (j = nrNod2 - 1; j >= 0; j--)
                                        {
                                                noduri2[j + 1] =
noduri2[j];
                                        }
                                        noduri2[0] = i;
                                        nrNod2++;
                                        viz2[i] = 1;
                                        parinte2[i] = nod2;
                                }
                        }
                }
                for (i = 0; i < 20; i++)
                {
                        if (viz1[i] == 1 && viz2[i] == 1)
                        {
                                okaius = 1;
                                final1 = i;
                                final2 = i;
                        }
                }
        }

        cout << endl;
        cout << "Rezultate cautare latime:" << endl;

        while (final1 != start)
        {
                solutie1[nrSol1++] = final1;
                final1 = parinte1[final1];
        }
        solutie1[nrSol1++] = start;
        for (i = nrSol1 - 1; i >= 0; i--)
        {
                cout << nume[solutie1[i]] << " ";
        }
        cout << endl;
        cout << endl;

        cout << "Rezultate cautare adancime:" << endl;

        while (final2 != stop)
        {
```

```
                solutie2[nrSol2++] = final2;
                final2 = parinte2[final2];
        }
        solutie2[nrSol2++] = stop;

        for (i = nrSol2 - 1; i >= 0; i--)
        {
                cout << nume[solutie2[i]] << " ";
        }
        cout << endl;

        int solutie3[20], nrSol3 = 0;
        cout << endl;

        for (i = nrSol1 - 1; i >= 1; i--)
        {
                solutie3[nrSol3++] = solutie1[i];
        }

        for (i = 0; i <= nrSol2 - 1; i++)
        {
                solutie3[nrSol3++] = solutie2[i];
        }

        cout << "Rezultatul concatenarii celor doua cautari:" << endl;
        for (i = 0; i <= nrSol3 - 1; i++)
        {
                cout << nume[solutie3[i]] << " ";
        }
}

int main() {

        int start = 2, stop = 7;

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

        distante[0][1] = 1;
        distante[0][8] = 1;
        distante[1][2] = 1;
        distante[2][3] = 1;
        distante[2][8] = 1;
        distante[3][4] = 1;
        distante[4][5] = 1;
        distante[5][6] = 1;
        distante[6][7] = 1;
        distante[7][9] = 1;
        distante[7][11] = 1;
        distante[8][9] = 1;
        distante[8][10] = 1;
        distante[9][11] = 1;
```

```
        distante[10][12] = 1;
        distante[11][12] = 1;
        distante[12][13] = 1;
        distante[12][14] = 1;
        distante[14][15] = 1;
        distante[14][17] = 1;
        distante[15][16] = 1;
        distante[17][18] = 1;
        distante[18][19] = 1;

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

        cautareBidirectionala(start, stop);

        _getch();
        return 0;
}
```

Programul rulat

```
Pasul[latime] 1: Arad
Pasul[adancime] 1: Craiova
Pasul[latime] 2: Zerind Timisoara Sibiu
Pasul[adancime] 2: Pitesti Ramnicu Valcea Drobeta
Pasul[latime] 3: Timisoara Sibiu Oradea
Pasul[adancime] 3: Bucuresti Ramnicu Valcea Drobeta
Pasul[latime] 4: Sibiu Oradea Lugoj
Pasul[adancime] 4: Urziceni Giurgiu Fagaras Ramnicu Valcea Drobeta

Rezultate cautare latime:
Arad Sibiu Fagaras

Rezultate cautare adancime:
Craiova Pitesti Bucuresti Fagaras

Rezultatul concatenarii celor doua cautari:
Arad Sibiu Fagaras Bucuresti Pitesti Craiova
```