Lista programe de rezolvat

1. Se da o lista de numere. Sa se calculeze suma acestora.
2. Se da o lista de numere. Sa se verifice daca un anumit numar exista sau nu in aceasta lista.
3. Se da o lista de numere. Sa se schimbe valorile dintre doua pozitii date din aceasta lista.
4. Gasiti valoarea minima dintr-o lista de numere.
5. Sa se elimine un numar aflat pe o anumita pozitie intr-o lista.
6. Sa se elimine un numar dat dintr-o lista data.
7. Sa se adauge un numar dat pe o pozitie data intr-o lista.
8. Avem trei vectori cu aceeasi lungime a, b si c. Scrieti un program care sa verifice daca pentru fiecare pozitie i, avem a[i] < b[i] < c[i]. Daca inegalitatea se verifica pentru toate elementele, intoarceti adevarat, altfel fals.
9. Ordonarea elementelor dintr-o lista.
10. Dandu-se o lista de numere ordonate crescator, inserati un numar dat in aceasta lista astfel incat ea sa ramana ordonata.
11. Sa se genereze un numar intre 0 si 100 in C.
12. Afisati cu o probabilitate de 0.5 (50% sanse) textul „Inteligenta Artificiala”. In celalalt caz afisati „Limbajul C”.
13. Oferiti sanse egale afisarii unuia din cele trei texte in C:
    a. Inteligenta Artificiala
    b. Limbajul C
    c. Programare Web
14. Dintr-o lista de *n* numere alegeti in mod aleatoriu *m* (*m* < *n*) numere nu neaparat unice (adica pot fi numere din lista initiala ce pot aparea de mai multe ori), le afisati, apoi gasiti maximul dintre aceste *n* numere.
15. Dintr-o lista de numere *n* numere alegeti in mod aleatoriu *m* numere neaparat unice (*m* < *n*).
16. Avand doua liste de numere ca mai jos, se alege un punct de taietura (3 in exemplul de mai jos) si construiti doua liste noi astfel:
    a. Se copiaza primele doua parti din listele initiale
    b. A doua parte se completeazaprin inserarea de elemente de la cealalta lista in ordinea in care apar, incepand cu punctul de dupa taietura, sarind valorile care apar deja in lista.

1	3	5	2	6	4	7	8
8	5	6	7	4	3	2	1

1	3	5	7	4	2	8	6
8	5	6	2	4	7	1	3

17. Reprezentati tabela de mai jos. Stiind ca posibilitatile de modificare ale tabelei sunt sa se mute casuta goala sus, jos, la stanga sau la dreapta, alegeti in mod aleatoriu una din aceste actiuni si afisati ce mutare are loc si ce stare se obtine.

2	3	5
4		1
8	6	7

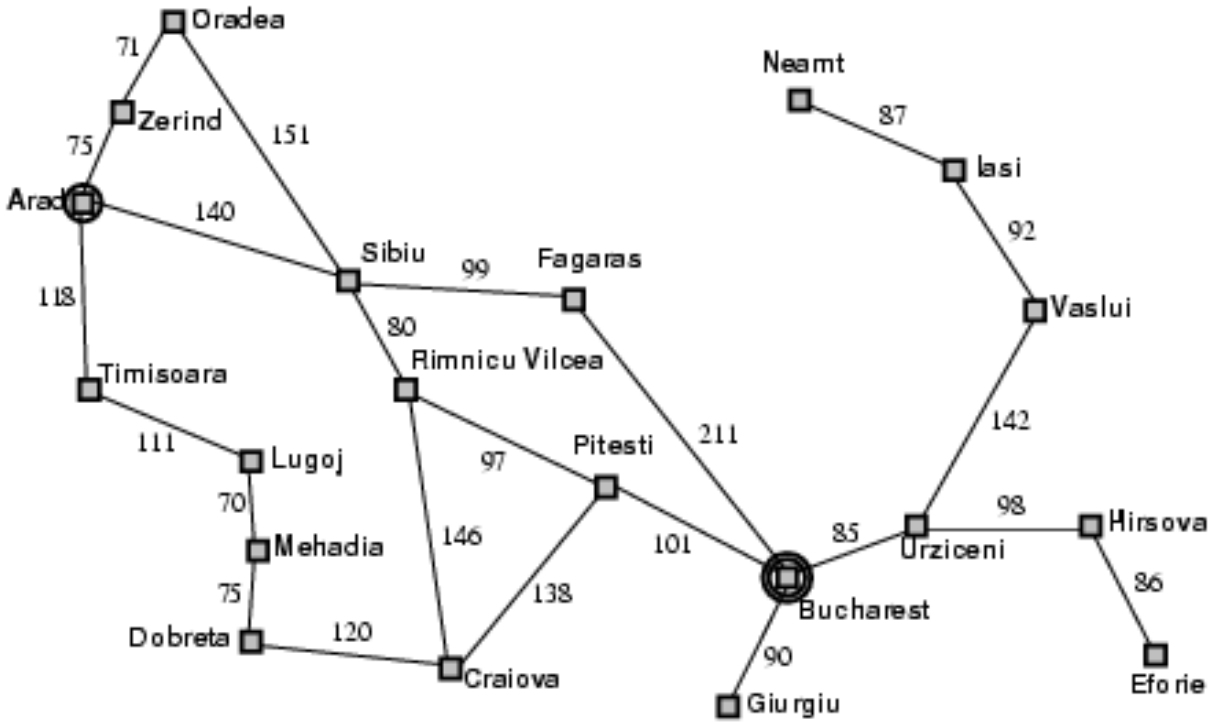
18. Reprezentati o tabla de X si 0 goala. Puneti in mod aleatoriu, alternativ, X, apoi 0, apoi iar X, iar 0 pana cand ori invinge X, ori invinge 0, ori tabla este completata.
19. Fie urmatorul joc ce se poate juca intre doua persoane:
    Primul jucator spune numar intre 1 si 10. Cel de-al doilea jucator adauga la numarul dat un numar intre 1 si 10. Primul jucator adauga la randul lui un numar intre 1 si 10 la numarul rezultat. Jocul se continua astfel, iar castigator este jucatorul care ajunge primul la 100.
    Realizati un program care sa poata juca acest joc cu utilizatorul.

Generarea unui numar aleatoriu intre 0 si n-1 in C

```
#include <time.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream>
using namespace std;
int main(){
      //orice altceva
      int n = 10;
      srand(time(NULL));
      int g = rand() % n;
      cout << g;
      //orice altceva
      _getch();
}
```

Lista programe de rezolvat

Fie harta de mai jos:



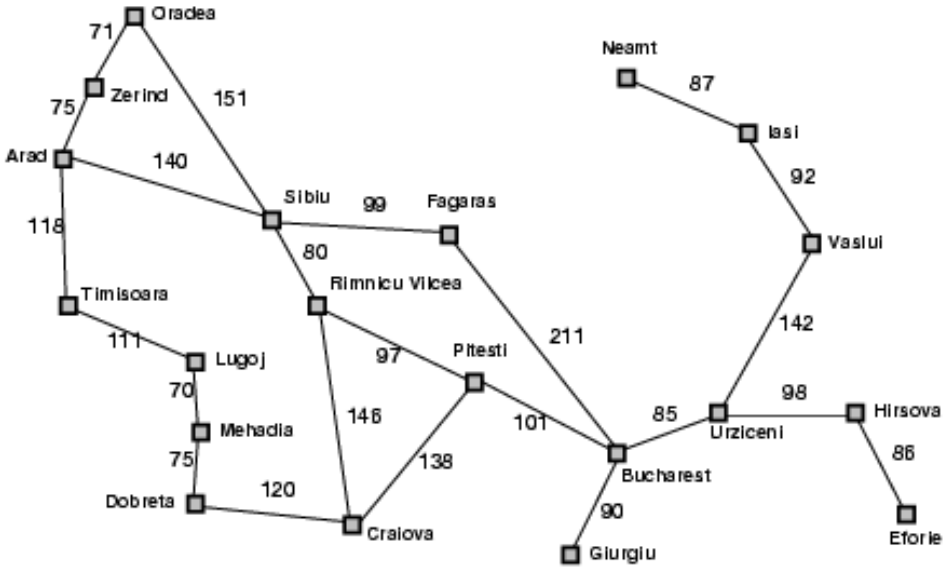
1. Scrieti un program in care sa introduceti toate orasele din aceasta harta si legaturile dintre ele sub forma unei matrice de adiacenta.
2. Utilizand programul de la punctul 1, pentru un oras stabilit, afisati toate orasele care sunt conectate de acesta.
3. Aplicați algoritmul de căutare in lățime pentru a găsi ruta intre doua orase oarecare. Algoritmul este descris la finalul acestui laborator.
4. Algoritmul de căutare in adâncime (adăugarea se face la început in lista *noduri* spre deosebire de căutarea in lățime).
5. Modificați algoritmii de mai sus astfel încât sa includeți si numărul de km dintre orașe si calculați distantele de la orașul de start pana la orașul de destinație.
6. Realizați o comparație intre cei doi algoritmi relativ la:
    a. Numărul de orașe din soluție de la start la destinație
    b. Numărul de orașe distincte care au fost vizitate
    c. Numărul de km pentru soluțiile găsite
7. Calculați o noua matrice de adiacenta relativa la timpul care se realizează intre oricare doua orașe conectate. Considerați o viteza medie de 80 km/h. Redați apoi si informațiile legate de timpi pentru fiecare soluție in parte.
8. Adăugați o noua matrice de adiacenta care sa conțină informații legate de numărul de mașini de pe fiecare ruta. Cu cat acestea sunt mai multe, cu atât cresc timpii de deplasare de la punctul 7. Folosiți matricea de adiacenta relativa la timp pentru a găsi soluțiile date de algoritmii de căutare in lățime si adâncime.

Algoritm de cautare in latime

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 final 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.

Lista programe de rezolvat

Fie harta de mai jos pentru care avem introdusa matricea de adiacenta (din laboratorul precedent):



Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

1. Aplicati algoritmul de cautare in adancime limitata (pentru fiecare nod se retine si adancimea la care se afla, incepand cu radacina care este la 0, iar la *while* se adauga si conditia sa nu se depaseasca limita prestabilita).
2. Cautarea in adancime iterativa (se cauta solutia la adancimea 0, apoi la 1, apoi la 2 s.a.m.d. pana cand ori se gaseste solutia ori nu mai exista alternative de incercat).
3. Cautarea cu cost uniform
    - matricea de adiacenta va contine intre doua orase conectate numarul de km dintre acestea si se stocheaza pentru fiecare nod si costul sau – numarul de km de la radacina pana la el.
    - Elementele in *noduri* sunt ordonate crescator in functie de costul lor.
    - Se adauga in *noduri* inclusiv elemente vizitate anterior care au un cost mai mic decat cel precedent.
4. Considerand valorile din dreapta figurii de mai sus, scrieti un program care sa foloseasca o cautare Greedy care sa ajunga la Bucuresti. Se adauga in lista *noduri* orasele astfel incat sa fie ordonate crescator in functie de aproximarile din dreapta figurii.
5. Pornind de la algoritmul de la exercitiul anterior si de la cautarea cu cost uniform (exercitiul 3), scrieti un algoritm care sa implementeze cautarea A*. Pentru aceasta, se ordoneaza in *noduri* orasele in functie de cost + aproximare.

Lista de programe

1. Implementati algoritmul de hill climbing pentru problema aranjarii a n dame pe o tabla de n x n (n poate fi considerat de la 4 in sus, cat permite calculatorul pentru a gasi solutia intr-un timp decent).
    - Pentru implementare, utilizati informatiile din prezentarea „Caracteristici si constrangeri”, slide-urile 49-56.
2. Transformati algoritmul de la exercitiul 1 intr-unul cu restart aleatoriu.
    - Pentru aceasta, cuprindeti intregul ciclu intr-o bucla while in care reinitilizati in mod aleatoriu solutia si stabiliti ca si conditie de oprire situatia in care solutia este gasita.
3. Implementati algoritmul de hill climbing pentru problema comis-voiajorului avand harta Romaniei din prezentarea „Caracteristici si constrangeri”, slide-urile 57-63.
4. Acelasi algoritm pentru rezolvarea urmatorului puzzle de criptaritmetica: trei+doi=cinci
5. Implementati algoritmul de hill climbing pentru o instanta a problemei rucsacului. Pentru aceasta folositi o codificare binara si numarul de elemente din vector este egal cu numarul de obiecte: o valoare de 1 reprezinta prezenta obiectului in rucsac, iar una de 0 absenta sa.
6. Implementati algoritmul simmuated annealing pentru toate problemele enuntate mai sus.