PRACTICĂ ALGORITMI FUNDAMENTALI

ANGHELUȘ CRINA-COSMINA

1.

include <iostream>

int VerificareNumar(int s)

{

int number = 1,suma=0;

while (s>suma)

{

suma += number;

number++;

}

number--;

if (suma == s)

return number;

else return -1;

}

int main()

{

int s;

std::cin >> s;

std::cout << VerificareNumar(s);

}

Complexitatea algoritmului propus este data de structura repetitive while care parcurge numerele incepand cu 1 pana cand suma construita este egala sau depaseste numarul dat. De aici rezulta o complexiatate de O(n).

2.

#include <iostream>

#include <algorithm>

using namespace std;

void AfisareDescrescatoare(int a, int b, int c)

{

int x, y;

x = min(a, min(b, c));

y = max(a, max(b, c));

cout << x << " " << a + b + c - x - y << " " << y;

}

int main()

{

int a, b, c;

std::cin >> a >> b >> c;

AfisareDescrescatoare(a, b, c);

}

3.

#include <iostream>

int Baza10InBaza2(int nrbaza10)

{

int nrbaza2 = 0, p = 1;

while (nrbaza10 != 0)

{

nrbaza2 = nrbaza2 + nrbaza10 % 2\* p;

p \*= 10;

nrbaza10 /= 2;

}

return nrbaza2;

}

int DeterminarePutere(int x)

{

x = Baza10InBaza2(x);

int k = 0;

while (x)

{

if (x % 10 == 1)

{

k++;

}

x = x / 10;

}

return k;

}

void CitireVector(int v[100], int &dim)

{

std::cout << "b) Dimensiune vector=";

std::cin >> dim;

std::cout << "Introduceti elementele vectorului:";

for (int i = 0; i < dim; i++)

{

int x;

std::cin >> x;

while (x < 0 || x > 1023)

{

std::cout << "Introduceti alt numar!";

std::cin >> x;

}

v[i] = x;

}

}

void ConstruireVectorGrupuri(int dim, int fr[11], int v[100])

{

for (int i = 0; i < dim; i++)

{

int x = DeterminarePutere(v[i]);

fr[x]++;

}

}

int GrupuriPuteri(int v[100], int fr[11], int dim)

{

fr[11] = { 0 };

int nr\_grupuri = 0;

ConstruireVectorGrupuri(dim, fr, v);

for (int i = 0; i <= 10; i++)

{

if (fr[i] != 0)

{

nr\_grupuri++;

}

}

return nr\_grupuri;

}

int main()

{

int valoare;

std::cout << "a) valoare= ";

std::cin >> valoare;

std::cout << valoare << " are puterea " << DeterminarePutere(valoare)<<std::endl;

int v[100] = { 0 }, dim, fr[11] = { 0 };

CitireVector(v, dim);

std::cout << "In acest vector sunt " << GrupuriPuteri(v,fr,dim)<< " grupuri de puteri.";

}

4.

#include <iostream>

#include <stdlib.h>

#include<time.h>

void GenerareRandom(int v[100], int dim)

{

srand(time(0));

for (int i = 0; i < dim; i++)

v[i] = rand();

}

int VerfificareSirDescrescator(int n, int v[100])

{

int ok = 1;

for (int i = 0; i < n-1&&ok==1; i++)

{

if (v[i] < v[i + 1])

{

ok = 0;

}

}

return ok;

}

int CautareBinara(int dim, int v[100], int element)

{

int st = 0, dr = dim - 1;

while (st <= dr)

{

int mij = (st + dr) / 2;

if (v[mij] == element)

return mij;

else

if (v[mij] > element)

st = mij + 1;

else dr = mij - 1;

}

return -1;

}

int CautareBinara2(int dim, int v[100], int element)

{

int st = 0, dr = dim - 1;

while (st <= dr)

{

int mij = (st + dr) / 2;

if (v[mij] < element&&v[mij-1]>element)

return mij;

else

if (v[mij] > element)

{

if (v[mij + 1] < element)

return mij;

else

st = mij + 1;

}

else dr = mij - 1;

}

return -1;

}

void InserareElement(int &dim, int v[100],int element)

{

int poz = CautareBinara2(dim, v, element);

dim++;

for (int i = dim - 1; i >= poz; i--)

{

v[i + 1] = v[i];

}

v[poz] = element;

}

void StergereElement(int& dim, int v[100], int element)

{

int poz = CautareBinara(dim, v, element);

if (poz != -1)

{

for (int i = poz; i < dim - 1; i++)

v[i] = v[i + 1];

dim--;

}

}

void AfisareVector(int dim, int v[100])

{

for (int i = 0; i < dim; i++)

{

std::cout << v[i] << " ";

}

}

int main()

{

int dim, v[100] = { 0 }, element\_i, element\_s;

std::cin >> dim;

GenerareRandom(v, dim);

if (VerfificareSirDescrescator(dim, v) == 0)

{

std::cout << "Vectorul nu este sortat strict descrescator.";

}

else

{

std::cout << "Vectorul este sortat strict descrescator."<<std::endl;

std::cin >> element\_i;

InserareElement(dim, v, element\_i);

AfisareVector(dim, v);

std::cout << std::endl;

std::cin >> element\_s;

StergereElement(dim, v, element\_s);

AfisareVector(dim, v);

}

}

Metodele de inserare si de stergere folosesc cate un algoritm de cautare binara pentru a determina pozitia pe care se insereaza sau pozitia de pe care se sterge un element. Complexitate algoritmilor de cautare este de O(logn), iar parcurgerea vectorului pana la pozitia cautata are o complexiatate de O(n). De unde rezulta o complexitate de O(n).

5.

#include <iostream>

void CitireVectori(int pompieri[100], int incendii[100],int &k,int &j)

{

int dim;

std::cin >> dim;

for (int i = 0; i < dim; i++)

{

char c;

std::cin >> c;

if (c == 'I')

{

incendii[k] = i;

k++;

}

else

{

pompieri[j] = i;

j++;

}

}

}

int DeterminareNumarIncendii(int pompieri[100], int incendii[100], int nr\_incendii, int nr\_pompieri,int k)

{

int nr\_incendii\_stinse = 0;

for (int i = 0; i < nr\_pompieri; i++)

{

int j = 0;

while (j<nr\_incendii)

{

if (abs(pompieri[i] - incendii[j]) <= k&&incendii[j]!=-1)

{

incendii[j] = -1;

nr\_incendii\_stinse++;

break;

}

j++;

}

}

return nr\_incendii\_stinse;

}

int main()

{

int incendii[100], pompieri[100],k,nr\_incendii=0,nr\_pompieri=0;

CitireVectori(pompieri, incendii, nr\_incendii, nr\_pompieri);

std::cin >> k;

std::cout << "Maximum "<< DeterminareNumarIncendii(pompieri, incendii, nr\_incendii, nr\_pompieri, k)<<" pot fi stinse.";

}

Metoda implementata parcurge vectorul care contine poziitile incendiilor si vectorul de pozitii ale pompierilor si asociaza fiecarui incendiu un pompier care se afla la o pozitie mai mica sau egala decat k, iar daca un incendiu este stins primeste valoarea -1. Metoda folosita este Greedy deoarece dorim sa determinam nr maxim de incendii care pot fi stinse. Complexitatea algoritmului este data de parcurgerea incendiilor si a pompierilor. Fiind vorba despre un while care parcurge vectorul de incendii, while aflat intr-un for rezulta o complexitate de O(n^2).