EXAMEN LABORATOR AF – ANGHELUȘ CRINA-COSMINA

#include <iostream>

#include <stdlib.h>

#include <fstream>

#include <algorithm>

struct Barca{

int nr\_pers;

int ID;

int kg\_provizii;

};

void zeroizare(int fr[1001])

{

for (int i = 0; i < 1001; i++)

{

fr[i] = 0;

}

}

void CitireDate(int& N, Barca barci[100])

{

int fr[1001];

zeroizare(fr);

std::cin >> N;

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

{

int x;

std::cin >> x;

while (x < 100 || x>1000 || fr[x]==1)

{

std::cout << "Introduceti un al ID." << std::endl;

std::cin >> x;

}

barci[i].ID = x;

fr[x] = 1;

}

}

void NumarPersCelMaiDesIntalnit(int N, Barca barci[100])

{

int frecventa\_pers[7] = { 0 };

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

{

frecventa\_pers[barci[i].nr\_pers]++;

}

int mx = 0,nr=0;

for (int i = 0; i < 7; i++)

{

if (frecventa\_pers[i] > mx)

{

mx = frecventa\_pers[i];

nr = i;

}

}

std::cout << "Numarul de persoane intr-o barca intalnit cel mai des este: " << nr << std::endl;

std::cout << "Aceste barci au id-urile: ";

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

{

if (frecventa\_pers[barci[i].nr\_pers]==mx)

{

std::cout << barci[i].ID << " ";

}

}

std::cout << std::endl;

}

void GenerareRandom(int N, Barca barci[100])

{

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

{

barci[i].nr\_pers = rand() % 6+1;

}

}

void ComponenteConsecutive(int N, Barca barci[100],int &st,int &dr,int &ok,int k)

{

int nr = 1;

for (int i = st; i < N - 1; i++)

{

for (int j = i + 1; j < N; j++)

{

if (barci[j - 1].nr\_pers < barci[j].nr\_pers)

{

nr++;

st = i;

dr = j;

}

else nr = 1;

}

if (nr == k)

ok++;

}

}

/\*Metoda aleasa cauta o secventa de lungime k de termeni ordonati crescatori incepand cu fiecare indice i prin parcurgerea vectorului cu

ajutorul a 2 structuri repetitive imbricate din care rezulta o complexitate O(n^2). \*/

void RezolvareB(int N, Barca barci[100])

{

int k;

std::cin >> k;

int st=0, dr = 0,ok=0;

ComponenteConsecutive(N, barci, st, dr,ok,k);

if (ok == 0)

{

std::cout << "NU exista k componente consecutive in ordine strict crescatoare." << std::endl;

}

else

{

std::ifstream f("provizii.txt");

for (int i = st; i <= dr; i++)

{

f >> barci[i].kg\_provizii;

}

int X, Y,nr=0;

std::cin >> X >> Y;

st = nr;

ComponenteConsecutive(N, barci, st, dr, ok,k);

bool verif = false;

for (int i = st; i <= dr&&verif==false; i++)

{

if (barci[i].nr\_pers == X && barci[i].kg\_provizii == Y)

{

std::cout << barci[i].ID << " " << barci[i].nr\_pers << " " << barci[i].kg\_provizii << std::endl;

verif = true;

ok = 0;

}

}

if (verif == false)

{

std::cout << "Nu a fost gasita o astfel de barca." << std::endl;

}

}

}

bool comparatorBarci(Barca x, Barca y)

{

return x.nr\_pers > y.nr\_pers;

}

/\*Metoda aleasa este Greedy doarece problema este una de optimizare care necesita ca barcile cu cele mai multe persoane sa

parcurga un numar minim de kilometrii, de aceea am ordonat crescator vectorul de kilometrii si descrescator vectorul de

barci in functie de nr de personae si am facut corespondenta intre acestia. Complexitatea astfel obtinuta este data de sortarile care au fost facute, ceea ce inseamna o complexitate

n\*logn.\*/

void RezolvareC(int N, Barca barci[100])

{

int distante[100];

std::ifstream g("distante.txt");

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

{

g >> distante[i];

}

std::sort(distante, distante + N);

std::sort(barci, barci + N, comparatorBarci);

int minim\_combustibil = 0;

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

{

minim\_combustibil += barci[i].nr\_pers \* distante[i];

std::cout << barci[i].ID << " ";

}

std::cout << std::endl;

std::cout << minim\_combustibil;

}

int main()

{

int N,k;

Barca barci[100];

CitireDate(N, barci);

GenerareRandom(N, barci);

std::cout << std::endl;

NumarPersCelMaiDesIntalnit(N, barci);

RezolvareB(N, barci);

RezolvareC(N, barci);

}