**Időjárás**

const int maxN = 100;

struct homerseklet

{

int minHom;

int maxHom;

};

int main()

{

homerseklet napok[maxN];

//beolvasás

int n;

cin >> n;

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

{

cin >> napok[i].minHom >> napok[i].maxHom;

}

//a) azon napok száma, amikor fagyni fog

int db = 0;

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

{

if (napok[i].minHom <= 0)

{

db++;

}

}

cout << db << endl;

//b) legnagyobb kül min és max között (ha több van -> legkisebb sorszámút)

int maxKul = 0;

for (int i = 1; i < n; ++i)

{

if (napok[i].maxHom - napok[i].minHom > napok[maxKul].maxHom - napok[maxKul].minHom)

{

maxKul = i;

}

}

cout << maxKul + 1 << endl;

//c) 1 nap, amikor max kisebb előző min-nál (több -> legkisebb sorszámú; nincs -> -1)

int ind = 1;

while (!(napok[ind].maxHom < napok[ind-1].minHom))

{

ind++;

}

bool van = (ind < n);

if (van)

{

cout << ind + 1 << endl;

}

else

{

cout << "-1" << endl;

}

//d) napok száma és sorszámai, ahol fagyni és olvadni is fog

int d[maxN];

int d\_db = 0;

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

{

if (napok[i].minHom <= 0 && napok[i].maxHom > 0)

{

d[d\_db] = i + 1;

d\_db++;

}

}

cout << d\_db << " ";

for (int i = 0; i < d\_db; ++i)

{

cout << d[i] << " ";

}

return 0;

}

**Lakás**

const int maxN=100;

struct Lakas

{

int ter, ar;

};

int A\_fel\_max(Lakas T[], const int N);

int B\_fel\_megsz(Lakas T[], const int N);

int C\_fel\_megsz(Lakas T[], const int N);

void D\_fel(Lakas T[], const int N);

int main()

{

int N;

Lakas T[maxN];

cin >> N;

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

{

cin >> T[i].ter >> T[i].ar;

}

/\*//beolasás ellenőrzése

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

{

clog << T[i].ter << " " << T[i].ar << endl;

}\*/

cout << A\_fel\_max(T, N) << endl;

cout << B\_fel\_megsz(T, N) << endl;

cout << C\_fel\_megsz(T, N) << endl;

D\_fel(T, N);

return 0;

}

int A\_fel\_max(Lakas T[], const int N)

{

int index = 0;

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

{

if (T[i].ar > T[index].ar)

{

index = i;

}

}

return index + 1;

}

int B\_fel\_megsz(Lakas T[], const int N)

{

int s = 0;

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

{

if(T[i].ter > 100 && T[i].ar < 40)

{

s++;

}

}

return s;

}

int C\_fel\_megsz(Lakas T[], const int N)

{

int s = 0;

int j;

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

{

j = 0;

while (j < i && T[i].ter != T[j].ter)

{

j++;

}

if (j == i)

{

s++;

}

}

return s;

}

void D\_fel(Lakas T[], const int N)

{

int seged[maxN];

int db = 0;

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

{

if (T[i].ar > 100)

{

seged[db] = i + 1;

db++;

}

}

cout << db << " ";

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

{

cout << seged[i] << " ";

}

}

**Utazás**

const int maxN = 100;

struct utazas

{

int tav;

int ar;

};

int main()

{

utazas ut[maxN];

//beolvasás

int n;

cin >> n;

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

{

cin >> ut[i].tav >> ut[i].ar;

}

//a) legtávolabbi város sorszáma (több -> legkisebb sorszámú)

int legtavolabbi = 0;

for (int i = 1; i < n; ++i)

{

if (ut[i].tav > ut[legtavolabbi].tav)

{

legtavolabbi = i;

}

}

cout << legtavolabbi + 1 << endl;

//b) 1000 km-nél közelebbiek közül a legdrágább

int legdragabb = 0;

bool ezer = false;

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

{

if (ut[i].tav < 1000)

{

if (ezer)

{

if (ut[i].ar > ut[legdragabb].ar)

{

legdragabb = i;

}

}

else

{

legdragabb = i;

ezer = true;

}

}

}

if (ut[legdragabb].tav >= 1000)

{

legdragabb = -1;

}

else

{

legdragabb = ut[legdragabb].ar;

}

cout << legdragabb << endl;

//c) hányféle ár, amelyről egyértelmű város és ár (aka nincs több belőle)

int hanyfele = 0;

bool van = false;

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

{

for (int j = 0; j < n; ++j)

{

if (i != j && ut[i].ar == ut[j].ar)

{

van = true;

}

}

if (van == false) //true lenne ha hányféle kül ár kéne

{

hanyfele++;

}

van = false;

}

cout << hanyfele << endl;

//d) városok száma és sorszámai ahova >100ft/km

int d[maxN];

int db = 0;

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

{

if (ut[i].ar / ut[i].tav >= 100) //feladat 100nál nagyobbat ír

{

d[db] = i + 1;

db++;

}

}

cout << db << " ";

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

{

cout << d[i] << " ";

}

return 0;

}

**Benzinkút**

struct Benzinkut

{

int Tav;

int Tank;

};

int main()

{

int hossz;

int tankDb;

int kezdoBenzin;

int fogyLit;

cin >> hossz >> tankDb >> kezdoBenzin >> fogyLit;

Benzinkut adatok[tankDb];

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

{ cin >> adatok[i].Tav >> adatok[i].Tank; }

//a) mennyi benzin lesz az út végén

int szuksegesBenzin = hossz / 100 \* fogyLit;

int osszBenzin = kezdoBenzin;

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

{ osszBenzin += adatok[i].Tank; }

cout << osszBenzin - szuksegesBenzin << endl;

//aktuális benzinmennyiség kiszámítása a többi feladathoz

int aktBenzin = kezdoBenzin;

int aktBenz[tankDb];

int aktBenz\_ind = 0;

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

{

if (i == 0)

{

aktBenzin = kezdoBenzin - (adatok[i].Tav / 100 \* fogyLit) + adatok[i].Tank;

aktBenz[i] = aktBenzin;

aktBenz\_ind++;

}

else

{

aktBenzin += adatok[i].Tank - ((adatok[i].Tav - adatok[i-1].Tav) / 100 \* fogyLit);

aktBenz[i] = aktBenzin;

aktBenz\_ind++;

}

}

//b) hány literes az üzemanyagtartály [aktbenzin maximuma]

int tartaly = 0;

int maxAktBenz = 0;

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

{

if (aktBenz[i] > aktBenz[maxAktBenz])

{

tartaly = aktBenz[i];

maxAktBenz = i;

}

}

cout << tartaly << endl;

//c) legkorábbi benzinkút száma, amikor már nem kellett volna tankolni

int osszTank = kezdoBenzin;

int osszTankolas[tankDb];

int osszTankolas\_ind = 0;

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

{

osszTank += adatok[i].Tank;

osszTankolas[i] = osszTank;

osszTankolas\_ind++;

}

int j = 0;

if (kezdoBenzin >= szuksegesBenzin)

{ cout << 0 << endl; }

else

{

while (!(osszTankolas[j] - szuksegesBenzin >= 0))

{

j++;

}

bool van = (j < tankDb);

if (van)

{

cout << j + 1 << endl;

}

}

//d) mely benzinkutaknál lett több a benzin, mint az elõzõ tankolás után

int db = 0;

int tobbBenz[tankDb];

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

{

if (aktBenz[i] > aktBenz[i-1])

{

tobbBenz[db] = i + 1;

db++;

}

}

cout << db << " ";

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

{

cout << tobbBenz[i] << " ";

}

cout << endl;

//e) leghosszabb olyan 2 benzinkút közötti szakasz hossza, amikor a benzin folyamatosan a kezdeti benzinmennyiség, vagy annál több volt

int i = 0;

int k = 0;

int kezdFelett[tankDb];

int kezdFelett\_ind = 0;

while (i < tankDb - 1)

{

k = i + 1;

if (aktBenz[i] >= kezdoBenzin)

{

k = i;

while (k < tankDb - 1 && aktBenz[k] >= kezdoBenzin)

{

k++;

}

kezdFelett[kezdFelett\_ind] = adatok[k].Tav - adatok[i].Tav;

kezdFelett\_ind++;

}

i = k;

}

int maxKezdFelett = 0;

int maxHossz = 0;

for (int i = 0; i < kezdFelett\_ind; ++i)

{

if (kezdFelett[i] > kezdFelett[maxKezdFelett])

{

maxHossz = kezdFelett[i];

maxKezdFelett = i;

}

else if (kezdFelett\_ind == 1)

{

maxHossz = kezdFelett[i];

}

}

cout << maxHossz;

return 0;

}

**Árvíz (9)**

const int maxN = 10000; //maximum meresi pont

const int maxA = 3000; //maximum meresi eredmeny

struct Arviz

{

int kezd, veg;

};

struct Csucspont

{

int index, meres;

};

void beolvas\_konzol(int meresek[], int & N);

int intervallum\_keres(int meresek[], const int N, Arviz szakasz[]); //arvizi szakaszok szamaval ter vissza

void kiir\_Arviz(Arviz t[], int N);

void csucspontok(const int meresek[], const int N, const Arviz szakasz[], const int szakasz\_n, Csucspont csucsok[]); //ahany szakasz, annyi csucspont

int max\_intervallumban(const int meresek[], const int N, int tol, int ig); //maximum kereses egy tomb intervallumaban

void kiir\_Csucsok(Csucspont t[], int N);

int main()

{

int meresek[maxN];

int N, szakasz\_N;

Arviz szakasz[maxN+1];

Csucspont csucsok[maxN+1];

beolvas\_konzol(meresek, N);

szakasz\_N = intervallum\_keres(meresek, N, szakasz);

cout << szakasz\_N << endl;

kiir\_Arviz(szakasz, szakasz\_N);

csucspontok(meresek, N, szakasz, szakasz\_N, csucsok);

kiir\_Csucsok(csucsok, szakasz\_N);

}

void beolvas\_konzol(int meresek[], int & N)

{

//clog<< "Add meg N-t!\n"; //(cout helyett) clog=consol log, nem latja biro, nem standard kimenet

cin >> N;

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

{

//clog << "Add meg " << i+1 << ". meresi eredmenyt: ";

cin >> meresek[i];

}

}

int intervallum\_keres(int meresek[], const int N, Arviz szakasz[])

{

int elsofok = 800;

int db = 0;

int j;

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

{

if ((i == 0 && meresek[i] > elsofok) || (i > 0 && meresek[i] > elsofok && meresek[i-1] < elsofok))

{

szakasz[db].kezd = i;

j = i;

while(j < N && meresek[j] > elsofok)

{

j++;

}

szakasz[db].veg = j-1;

db++;

//i = j-i;

}

}

return db; //arvizi szakaszok szama

}

void kiir\_Arviz(Arviz t[], int N)

{

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

{

cout << t[i].kezd + 1 << " " << t[i].veg + 1 << " ";

}

if (N > 0)

{

cout << endl;

}

}

void csucspontok(const int meresek[], const int N, const Arviz szakasz[], const int szakasz\_n, Csucspont csucsok[])

{

int maxind, maxertek;

int db = 0;

for (int i = 0; i < szakasz\_n; i++)

{

maxind = max\_intervallumban(meresek, N, szakasz[i].kezd, szakasz[i].veg);

maxertek = meresek[maxind];

csucsok[db].index = maxind + 1;

csucsok[db].meres = maxertek;

db++;

}

}

int max\_intervallumban(const int meresek[], const int N, int tol, int ig)

{

if (tol < 0 || ig > N)

{

return -1;

}

int index = tol;

for (int i = tol + 1; i <= ig; i++)

{

if (meresek[index] < meresek[i])

{

index = i;

}

}

return index;

}

void kiir\_Csucsok(Csucspont t[], int N)

{

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

{

cout << t[i].index << " " << t[i].meres << " ";

}

if (N > 0)

{

cout << endl;

}

}