20. travnja 2023.

2023 Natjecanje iz informatike

Državna razina 2023/ Osnovna škola (5. i 6. i 7. i 8. razred)

Primjena algoritama OŠ

OPISI ALGORITAMA

Agencija za odgoj i obrazovanje
Education and Teacher Training Agency

HRVATSKI SAVEZ INFORMATIČARA

Ministarstvo znanosti, obrazovanja i sporta

5.1. Zadatak: ŠČ

Autor: Nikola Dmitrović

Za rješenje analiziraj programski kod te ga probaj pojednostaviti.

Programski kod (pisan u Python 3.x)

```
D = int(input())
DR = int(input())
MR = int(input())
dani_pon = ["ZETOR", "IMT", "FENDT", "URSUS", "DEERE", "TORPEDO", "TV"]
dani_uto = ["IMT", "FENDT", "URSUS", "DEERE", "TORPEDO", "TV", "ZETOR"]
dani_sri = ["FENDT", "URSUS", "DEERE", "TORPEDO", "TV", "ZETOR", "IMT"]
dani_cet = ["URSUS", "DEERE", "TORPEDO", "TV","ZETOR", "IMT", "FENDT"]
dani_pet = ["DEERE", "TORPEDO", "TV","ZETOR", "IMT", "FENDT", "URSUS"]
dani_sub = ["TORPEDO", "TV","ZETOR", "IMT", "FENDT", "URSUS", "DEERE"]
dani_ned = ["TV","ZETOR", "IMT", "FENDT", "URSUS", "DEERE", "TORPEDO"]
mjeseci = [31,28,31,30,31,30,31,31,30,31,30,31]
X = DR
for i in range(MR-1):
    X += mjeseci[i]
X = X % 7
if X == 0:
    X = 7
if D == 1:
    print(dani_pon[X - 1])
elif D == 2:
    print(dani_uto[X - 1])
elif D == 3:
    print(dani_sri[X - 1])
elif D == 4:
    print(dani_cet[X - 1])
elif D == 5:
    print(dani_pet[X - 1])
elif D == 6:
    print(dani_sub[X - 1])
elif D == 7:
    print(dani_ned[X - 1])
```

5.2. Zadatak: Hermiona

Autor: Nikola Dmitrović

Zadani broj N rastavimo na moguće slučajeve te provjerama odredimo koji je najveći mogući zbroj.

Programski kod (pisan u Python 3.x)

```
N = int(input())
zbroj = 0
for i in range(N):
    H = int(input())
    if H < 10:
        zbroj += H
    elif H < 100:
        zbroj += H // 10 + H % 10
    elif H < 1000:
        zbroj += H
    elif H < 10000:
        if (H // 10 + H % 10) > (H // 1000 + H % 1000):
            zbroj += H // 10 + H % 10
        else:
            zbroj += H // 1000 + H % 1000
    else:
        zbroj += H
print(zbroj)
```

5.3. Zadatak: Film

Autor: Nikola Dmitrović

Analizom programskog koda dođite do rješenja,

Programski kod (pisan u Python 3.x)

```
X = int(input())
N = int(input())
film = [0] * X
for i in range(N):
    PFs, PFm = map(int, input().split())
    PGs, PGm = map(int, input().split())
    KGs, KGm = map(int, input().split())
    P_filma = PFs * 60 + PFm
    K_filma = (P_filma + X) % 1440
```

```
    print(K_filma // 60, K_filma % 60)
#***************************
    P_gledanja = PGs * 60 + PGm
    K_gledanja = KGs * 60 + KGm
    if K_filma > P_filma:
        L = P_gledanja - P_filma
        D = K_gledanja - P_gledanja
        for i in range(L, L + D):
            film[i] = 1
        koliko = sum(film)
    else:
        if P_gledanja > P_filma:
            if K_gledanja > P_gledanja:
                L = P_gledanja - P_filma
                D = K_gledanja - P_gledanja
                for i in range(L, L + D):
                    film[i] = 1
                koliko = sum(film)
            else:
                L = P_gledanja - P_filma
                D = 1440 - P_gledanja + K_gledanja
                for i in range(L, L + D):
                    film[i] = 1
                koliko = sum(film)
        else:
            L = 1440 - P_filma + P_gledanja
            D = K_gledanja - P_gledanja
            for i in range(L, L + D):
                film[i] = 1
            koliko = sum(film)
print(koliko)
```

6.1. Zadatak: Hermiona Autor:

Vidi 5.2

6.2. Zadatak: Film Autor:

Vidi 5.3

6.3. Zadatak: Igra Autor: Vito Anić

Ideja rješenja je simulirati igru, uzimajući u obzir sva pravila iz zadatka. Bilo je potrebno paziti na puno graničnih uvjeta poput: kada je neki igrač završio igru da se njega preskače, kada igrač prolazi kroz žuto polje da obavlja njegovu akciju, ako bi igrač sa svojim potezom prošao kroz crveno polje da se na njemu zaustavi. Za svakog igrača ćemo pamtiti njegovu poziciju na ploči i koliko novaca ima. Primjer implementacije rješenja slijedi u nastavku:

Programski kod (pisan u Python 3.x)

```
n, m, l = map(int, input().split())
akcija = [0]
boja = ["START"]

for i in range(m):
    a, b, c = input().split()
    c = int(c)

    if b == "KAZNA":
        c = -c
    akcija.append(c)
    boja.append(a)

novci = [l] * n
pos = [0] * n
trenutni = 0

k = int(input())
l = list(map(int, input().split()))

for i in l:
    while pos[trenutni] == m:
        trenutni = (trenutni + 1) % n

    for j in range(i):
        pos[trenutni] += 1
        if boja[pos[trenutni]] == "ZUTA":
            novci[trenutni] += akcija[pos[trenutni]]
        elif boja[pos[trenutni]] == "CRVENA":
            break

    if boja[pos[trenutni]] != "ZUTA":
        novci[trenutni] += akcija[pos[trenutni]]
    trenutni = (trenutni + 1) % n

for i in novci:
    print(i, end = ' ')
print()
```

7.1. Zadatak: Film Autor:

Vidi 5.3

7.2. Zadatak: Igra Autor: Vito Anić

Treći se podzadatak mogao riješiti simuliranjem igre. Svakog igrača možemo micati polje po polje, ako naiđemo na crveno polje, stanemo, ako prolazimo kroz žuto obavimo akciju. No za veće ograničenja, ovaj pristup nije dovoljno brz, tako da je potrebno razmišljati o bržoj ideji.

Za prvi podzadatak možemo primijetiti da jedine akcije koje radimo su plave (osim zadnje). Dakle za svakog igrača ćemo pamtiti na kojem se polju trenutno nalazi, kada zavrti kolo i dobije X, pomaknuti ćemo ga odjednom za toliko polja prema naprijed i obaviti akciju na polju na kojem smo završili. Ako smo kojim slučajem došli do kraja ploče ili ga prekoračili, vratiti ćemo igrača na kraj, obaviti zadnju akciju i u svakom idućem krugu ga preskočiti.

Rješenje cijelog zadatka se bazira na ideji navedenoj gore: kada igrač dođe na potez umjesto da ga mičemo polje po polje, želimo ga pomaknuti X ili manje koraka odjednom i odraditi sve potrebne akcije odjednom. Plave akcije smo riješili. Ako igrač stane na plavo polje obaviti ćemo tu akciju, inače nećemo. Pogledajmo sada žute akcije. Ako se nalazimo na nekom polju a, i cilj nam je na polju a+X, moramo obaviti sve žute akcije između. Pamtiti ćemo sumu žutih akcija od početka do svakog polja, takozvani prefiks sume. Na primjer ako imamo na prva tri polja žute akcije koje su redom NAGRADA 3, KAZNA 2, NAGRADA 5, niz će izgledati ovako: [3, 3-2, 3-2+5] = [3, 1, 6]. Ako želimo dobiti sumu svih akcija žutih akcija koje se nalaze na poljima od a do a+X, to je kao da smo uzeli sve žute akcije na prvih a+X polja i od njih oduzeli žute akcije na prvih a polja. Ostaju nam samo crvene akcije. Napravimo novi niz u kojem pamtimo indekse crvenih polja. Primijetimo da ako je igrač na nekom polju a da je nužno morao stati na svim crvenim poljima s manjim indeksima od a. Za svakog igrača pamtimo na koliko je crvenih polja zasad stao. Ako želimo s a doći do a+X, dodatno moramo provjeriti je li prvo crveno polje na kojem taj igrač nije bio negdje između. Ako je, umjesto da idemo na a+X ćemo stati na njemu, ako nije možemo slobodno otići na a+X. Tako smo riješili i crvena polja.

Ovo je rješenje bilo dovoljno za veliku većinu bodova. No za sve bodove bilo je potrebno primijetiti još jedan detalj. Kada neki igrač završi igru, bilo ga je potrebno efikasno ukloniti iz popisa igrača. Pristup da se on svaki idući puta samo preskače i traži prvi idući igrač koji još može napraviti potez nije dovoljno brz, za određene primjere on ispada vremenske složenosti O(N*K). Za efikasniji pristup bilo je moguće koristiti strukturu set, koja podržava izbacivanje i pretraživanje u logaritamskoj složenosti. U službenom rješenju korištena su dva niza iduci i prosli, koji za svakog igrača i, određuje tko će idući biti na potezu i tko je prije njega bio na potezu. Ukupna vremenska složenost O(K).

7.3. Zadatak: Zmija Autor: Stjepan Požgaj

Prvo ćemo objasniti kako riješiti problem za K=0, na temelju tog rješenja izgraditi rješenje za K=1 i isto tako iz njega izgraditi rješenje za K=2.

Za početak bez smanjenja općenitosti možemo pretpostaviti da je zmija svoje kretanje započela slijeva nadesno i rješavati taj problem. Taj ćemo problem, naime, rješavati za 4 različite orijentacije matrice i na kraju uzeti najveće od 4 pronađenih rješenja. U slučaju kad nema skretanja za svaki redak matrice želimo pronaći uzastopni podniz s najvećim zbrojem. To možemo tako for petljom prođemo po tom retku i u svakom trenutku održavamo dvije vrijednosti: ukupni zbroj elemenata niza iz ovog retka do

trenutnog elementa niza i najmanji takav zbroj do sad. Razlika između te dvije vrijednosti je upravo najveći mogući zbroj elemenata uzastopnog podniza koji završava na trenutnom elementu.

Slučaj s jednim skretanjem ćemo riješiti na sličan način. Opet ćemo for petljom prolaziti po svakom od redaka, no u ovom slučaju moramo prije tih prolazaka za svaki polje matrice (i, j) izračunati koji je najveći zbroj elemenata uzastopnog podniza j-tog stupca koji završava u (i-1)-vom retku i koji je najveći zbroj elemenata uzastopnog podniza j-tog stupca koji počinje u (i+1)-vom retku. Za dobivanje rješenja u ovom slučaju treba, uz trenutno vrijednost dobivenu na isti način kao u K=0, uzeti i veću od ovih prije izračunatih vrijednosti.

U slučaju u kojem su dozvoljena dva skretanja ćemo bez smanjenja općenitosti pretpostaviti sličnu stvar kao u prva dva slučaja. Sad pretpostavljamo da je zmija krenula odozgo prema dolje ili odozgo prema dolje. Rješenje u ovom slučaju je nadogradnja rješenja iz K=1 na isti način kao što smo iz K=0 dobili rješenja za K=2. Za više detalja čitatelju preporučamo čitanje izvornog koda službenog rješenja.

Vremenska složenost ovog rješenja je O(NM).

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

#define TRACE(x) cerr << #x << " = " << x << endl

#define FOR(i, a, b) for(int i = (a); i < (b); i++)
#define REP(i, n) FOR(i, 0, n)

typedef long long int llint;

typedef pair<int, int> par;

#define X first
#define Y second

int n, m;
vector<vector<int> > mat;
int rj[3];

void unos() {
  scanf("%d %d", &n, &m);
  REP(i, n) {
    vector<int> v;
    REP(j, m) {
      int x;
      scanf("%d", &x);
      v.push_back(x);
    }
    mat.push_back(v);
  }
}

void rotiraj() {
  vector<vector<int> > nova(m, vector<int> (n));
  REP(i, n)
    REP(j, m)
      nova[j][n - 1 - i] = mat[i][j];
  mat = nova;
  swap(n, m);
```

```
}

int pronadji_rjesenje() {
  vector<vector<int> > prema_dolje(n, vector<int> (m));
  vector<vector<int> > prema_gore(n, vector<int> (m));
  REP(j, m) {
    prema_dolje[0][j] = max(0, mat[0][j]);
    FOR(i, 1, n)
      prema_dolje[i][j] = max(0, mat[i][j] + prema_dolje[i - 1][j]);
    prema_gore[n - 1][j] = max(0, mat[n - 1][j]);
    for(int i = n - 2; i >= 0; i--)
      prema_gore[i][j] = max(0, mat[i][j] + prema_gore[i + 1][j]);
    REP(i, n)
      rj[0] = max(rj[0], max(prema_dolje[i][j], prema_gore[i][j]));
  }
  int ret = 0;
  REP(i, n) {
    int maksi = 0;
    REP(j, m) {
      int novi_krak = max(prema_dolje[i][j], prema_gore[i][j]);
      rj[2] = max(rj[2], maksi + novi_krak);
      maksi = max(maksi + mat[i][j], novi_krak);
      rj[1] = max(rj[1], maksi);
    }
  }
  return ret;
}

int main() {
  unos();
  int k;
  scanf("%d", &k);
  REP(i, 4) {
    pronadji_rjesenje();
    rotiraj();
  }
  printf("%d\n", rj[k]);
  return 0;
}
```

8.1. Zadatak: Igra Autor:

Vidi 7.2

8.2. Zadatak: Zmija Autor:

Vidi 7.3

8.3. Zadatak: Proizvodi Autor: Stjepan Požgaj

Ovaj zadatak ćemo riješiti dinamičkim programiranjem. U stanju dp[i][j] je zapisano koja je najveća moguća ukupna vrijednost proizvoda koju Mirko može imati ako se gleda prvih i vrsta proizvoda i ako je, da bi imao tih prvih i vrsta, napravio ukupno j preuzimanja. Ključno je primijetiti da je u prijelazu, ako odlučimo napraviti ukupno k preuzimanja proizvoda i-te vrste, optimalno pohlepno preuzeti proizvode od ljudi koji imaju najmanje proizvoda te vrste. Zbog toga je za svaku vrstu proizvoda, prije nego isprobamo sve kombinacije za broj proizvoda te vrste koje će Mirko preuzeti, potrebno sortirati ljude, odnosno brojeve proizvoda te vrste koje ljudi imaju kako bismo jednostavno i brzo mogli odrediti koja je cijena proizvoda i-te vrste ako smo na prethodno opisan pohlepan način preuzeli ukupno k takvih proizvoda. Ukupna vremenska složenost ovog rješenja je O(MK2).

```
#include <iostream>
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;

#define TRACE(x) cerr << #x << " = " << x << endl

#define FOR(i, a, b) for(int i = (a); i < (b); i++)
#define REP(i, n) FOR(i, 0, n)

typedef long long int llint;

typedef pair<int, int> par;

#define X first
#define Y second

const int MAXN = 1010, MAXM = 110, MAXK = 110;

int n, m, p;
int mat[MAXN][MAXM];
llint dp[MAXM][MAXK];
llint rj;

int main() {
  scanf("%d %d", &n, &m);
  REP(i, n)
    REP(j, m)
      scanf("%d", &mat[i][j]);
  scanf("%d", &p);
  FOR(i, 1, m + 1) {
    vector<int> v;
    FOR(j, 1, n)
      if (mat[j][i - 1] > 0)
        v.push_back(mat[j][i - 1]);
    sort(v.begin(), v.end());
```

```
    int cijena_prije = n - (int) v.size();
    int ima = mat[0][i - 1] > 0;
    cijena_prije -= ima;
    vector<bool> povecanja;
    REP(i, (int) v.size()) {
      REP(j, v[i] - 1)
        povecanja.push_back(false);
      povecanja.push_back(true);
    }
    REP(j, p + 1) {
      int trenutna_cijena = cijena_prije;
      REP(k, min(j + 1, (int) povecanja.size() + 1)) {
        if (k >= 1) {
          if (k == 1 && !ima)
            trenutna_cijena--;
          if (povecanja[k - 1])
            trenutna_cijena++;
        }
        dp[i][j] = max(dp[i][j], dp[i - 1][j - k] +
            (llint) trenutna_cijena * (mat[0][i - 1] + k));
      }
      if (i == m)
        rj = max(rj, dp[i][j]);
    }
  }
  printf("%lld\n", rj);
  return 0;
}
```