Megoldások

A feladatsor létrejöttében közreműködtek: Bakonyi Viktória, Busa Máté, Csertán András, Deák Bence, Gáspár Attila, Horváth Gyula, Németh Zsolt, Noszály Áron, Sárközi Gergely, Szente Péter, Zsakó László

31

Ötlet: Zsakó László

Kidolgozó: Zsakó László

Témák: implementáció (elágazás, ciklus, tömb)

A feladat erősen implementációs jellegű, a megoldáshoz elegendő a feladatszövegben definiált algoritmus lekódolása. Gyakori buktatók voltak a második üres sor hiánya vagy az algoritmus nem pontos megvalósítása (pl. `N>17`-ig ismételni `N>0` helyett). Érdekeségképpen meggondolhatjuk, hogy az algoritmus `O(log(N))` futási idejű, mivel lépésenként legalább `1`-el csökken az `N` tízes számrendszerbeli hossza.

Torony2

Ötlet: Zsakó László

Kidolgozó: Zsakó László

Témák: rekurzív sorozatok

A válasz a kérdésre legyen `F(N)`! Ekkor:

$$F(N) = \begin{cases} 1 & ha \quad N = 0 \\ 3 & ha \quad N = 1 \\ 2 * F(N-1) + 2 * F(N-3) & ha \quad N > 1 \end{cases}$$

Miért pont `2*F(N-1)+2*F(N-3)`? Osztályozzuk az `N` magas tornyokat! Tetejükön piros, zöld, fehér vagy sárga építőelem van, ezeket levéve `N-1`, `N-1`, `N-3` vagy `N-3` nagyságú tornyokhoz jutunk, amikből összesen `F(N-1)+F(N-1)+F(N-3)+F(N-3)=2*F(N-1)+2*F(N-3)` van. Lépésenkénti maradékszámítás kihagyásával a nagy értékű eredmények hibásak lesznek (ha nincs tetszőlegesen nagy szám ábrázolására alkalmas típusunk). Az `F` értékek kiszámítására nem célszerű nyelvbe épített rekurziót használnunk, mivel naívan (ún. memorizáció nélkül) nagyon lassú, exponenciális futási idejű megoldást kapunk.

Érdemes lehet ciklust használni, két különböző módon is gondolkodhatunk (F(i)-t miből számíthatjuk ki, illetve F(i)-ből mit számíthatunk ki):

```
F[0]=1; F(1):=2; F[2]:=2*F[1]
Ciklus i=3-tól N-ig
  F[i]=(3*F[i-1]+F[i-2]) mod 20210108
Ciklus vége
```

```
F[0]=1;
Ciklus i=0-tól N-1-ig
  F[i+1]=(F[i+1]+2*F[i]) mod 20210108
  F[i+3]=(F[i+3]+2*F[i]) mod 20210108
Ciklus vége
```

Az `F` kezdetben mindkét esetben `0`-kal van feltöltve, az eredmény mindkettőben `F(N)`, bár a második még F(N+2)-be is számol.

Tehát összességében `O(N)` időben megoldhatjuk a feladatot. Megjegyzendő, `O(log(N))` idejű megoldás is adható, például a rekurzió mátrixának gyorshatványozásával.

Inverzió

Ötlet: Noszály Áron

Kidolgozó: Noszály Áron

Témák: kombinatorika, 2 mutató technika

Egy $O(N^2)$-es megoldás: vizsgáljuk az összes indexpárt, egy indexpárra $O(1)$ időben eldönthető, hogy inverziót alkotnak-e. Ezek közül a legtávolabbi párt kell kiválasztanunk.

Észrevehetjük viszont, hogy ez sok felesleges vizsgálattal jár, mivel minden i számra elég lenne az i-nél kisebb számok közül csak a legjobboldalibbat megtalálni. Ezt sokféleképpen megtehetjük, de az egyik legegyszerűbb eljárás az, hogy n-től 2-ig végighaladunk az i értékeken és megkeressük a legjobboldalibb, i-nél kisebb szám indexét: ekkor kisebb i-t vizsgálva ez az index vagy ugyanaz lesz, mint korábban, vagy balra tolódik, tehát összességében $O(N)$ időben megoldhatjuk a feladatot.

Például a 4,1,2,3,6,5 sorozatra, a piros szám jelöli az i számot, míg a zöld a legjobboldalibb i-nél kisebb számot:

. 4 1 2 3 6 5

. 4 1 2 3 6 5

. 4 1 2 3 6 5

. 4 1 2 3 6 5

. 4 1 2 3 6 5

Késés

Ötlet: Noszály Áron

Kidolgozó: Noszály Áron

Témák: rendezés, koordináta kompresszió

Ahhoz, hogy minden évszámról halljunk, megérkezésünknek nem szabad semelyik évszám utolsó elhangzása után történnie. Tehát azt az évszámot keressük, amelyik utolsó elhangzása a leghamarabb történik: ez az időpont lesz a válaszunk.

Ha $E_i \leq N$, akkor ezt könnyen megtehetjük egy tömb segítségével: minden évre feljegyezzük a legkésőbbi előfordulás indexét és ezek közül a legkisebbet kiválasztjuk lineáris időben. Ha E_i már 10^9-es nagyságrendű, akkor óvatosabban kell eljárnunk, mivel egy 10^9 méretű tömbbel való munka sem időben, sem memóriában nem fér bele a korlátokba.

Három különböző módszert adunk ezen probléma feloldására:

- Észrevehetjük, hogy maguk az évszámok értékei nem számítanak, csak az egymáshoz viszonyított sorrendjük, így helyettesíthetjük őket legfeljebb `N` értékű számokkal, pl. a `(3,4,3,4,1,2)` sorozatra ugyanaz a megoldás, mint a példában szereplő eredetire. Ez a technika koordináta kompresszió néven is ismert.
- Az eredeti megoldásban tömbünket egy bonyolultabb adatstruktúrával helyettesítjük, pl. hash tábla vagy c++-ban map.
- `(évszám,index)` párokat képzünk, és azokat rendezzük év, azon belül index szerint. Pl. a példában `(895,5); (1516,6); (1848,1); (1848,3); (1849,2); (1849,4)` sorozatot kapjuk, láthatjuk, hogy ebből könnyen megkaphatjuk a legutolsó előfordulási helyeket (`5,6,3,4`).

Színezés

Ötlet: Noszály Áron

Kidolgozó: Noszály Áron

Témák: észrevételek

Belátható hogy a minimális színező műveletek száma $min(p,k) = \left\lfloor \frac{p+k}{2} \right\rfloor$, ahol `p` az egybefüggő piros, `k` az egybefüggő kék részek számát jelöli a sorozatban (hiszen `p` és `k` maximum eggyel térhet el). Pl. a példában kezdetben `p=k=2`, és tényleg 2 műveletre van szükségünk.

Ezzel már `N*Q` művelettel megoldhatjuk a feladatot, ha minden változtatás után kiszámítjuk `p`-t és `k`-t. Ahhoz, hogy ezt javítsuk, azt kell észre vennünk, hogy egy művelet után `p` és `k` értéke nagyon kicsit változhat csak és a már meglévő értékekből `O(1)` időben kiszámíthatjuk az újakat. Tehát összességében `O(N+Q)` időben megoldottuk a feladatot.

Az `i`. elem változtatása:

```
Ha t[i-1]=0 és t[i]=0 és t[i+1]=0 akkor p:=p+1; k:=k+1
Ha t[i-1]=1 és t[i]=0 és t[i+1]=1 akkor p:=p-1; k:=k-1
Ha t[i-1]=0 és t[i]=1 és t[i+1]=0 akkor p:=p-1; k:=k-1
Ha t[i-1]=1 és t[i]=1 és t[i+1]=1 akkor p:=p+1; k:=k+1
t[i]:=1-t[i]
```

Zene

Ötlet: Nikházy László

Kidolgozó: Deák Bence

Témák: prefix összegek, bináris keresés

Világos, hogy az $[1,T_1]$ időpontokban az első, $[T_1+1,T_1+T_2]$-ben a második, $[T_1+T_2+1, T_1+T_2+T_3]$-ban a harmadik zene szól, és így tovább.

Általánosan: $[T_1+...+T_{i-1}+1,T_1+...+T_i]$-re a válasz i.

Állítsuk elő T prefix összegeit: $S_i=T_1+...+T_i$! Ezek szerint az $[S_{i-1}+1,S_i]$ időpontokban az i-edik zeneszám szól. A feladat szerint a lejátszás periódusa S_N, vagyis bármely $p>S_N$ időpontban ugyanaz a dal szól, mint a $p-S_N$ időpontban.

Először egy adott P_i időponthoz határozzuk meg azt a $1 \le p \le S_N$ értéket, amire $P_i \equiv p \pmod{S_N}$! Ekkor a válasz az a legnagyobb j index lesz, amire $p \le S_j$. Megkereséséhez használhatunk bináris keresést. A bináris keresés komplexitása $O(\log(N))$, tehát a teljes algoritmus aszimptotikus futási ideje $O(N+K*\log(N))$ lesz.