Utolsó vacsora

Leonardo nagyon aktívan dolgozott az Utolsó vacsora elkészítésekor. Minden reggel eldöntötte, hogy milyen festékeket fog használni aznap. Sokféle színre volt szüksége, de egyidejűleg csak adott számút tudott tárolni az állványán, amiről a freskót festette. A segédje gondoskodott arról, hogy az éppen szükséges szín mindig ott legyen az állványán.

Két programot kell írnod! Az első program Leonardo kérései alapján készít egy rövid bitsorozatot, amit kódsorozatnak hívunk. A nap folyamán a segéd nem ismeri Leonardo jövőbeli kéréseit, csak a kódsorozatot. A második program a kódsorozat és az aktuális kérés alapján megadja, hogy a segéd hogyan elégítse ki a kérést.

A festékek mozgatása

Tegyük fel, hogy N színünk van, 0-tól N-1-ig sorszámozva és Leonardo minden nap pontosan N-szer kér színt a segédjétől. Legyen C az igényelt N szín sorozata, azaz C N számot tartalmaz, mindegyik 0 és N-1 közötti. A C-ben egyes színek többször is lehetnek, mások egyszer sem.

Az állványon mindig pontosan K szín van (K<N), a nap kezdetén a 0..K-1 színek vannak ott.

Leonardo segédje a kéréseket egyesével teljesíti. Ha a kért szín már az állványon van, akkor a segéd pihenhet. Ha nincs az állványon, akkor a kívánt színt felviszi az állványra, de egy másik helyére kell raknia és azt visszavinni a polcra, ahol az összes festéket tárolja.

Leonardo optimális stratégiája

A segéd a lehető legtöbbet szeretne pihenni, de ez függ az általa választott stratégiától. Leonardo elmagyarázta, hogy hogyan pihenhet a legtöbbet, ha ismeri a C sorozatot. Ha a szükséges szín nincs az állványon, akkor olyan helyére rakja, hogy a lehető legkésőbb kelljen újat vinnie. Ehhez meg kell néznie

- az állványon levőket és
- a C-ben levő további igényeket.

A kicserélendő szín vagy olyan, amire már nem lesz szükség a jövőben, vagy olyan, amire később lesz szükség, mint bármely másikra, ami az állványon van.

1. példa

Legyen N=4, azaz 4 szín van (0-tól 3-ig sorszámozva) és 4 kérés. A kérések sorrendje: C = (2, 0, 3, 0), és tegyük fel, hogy K=2, tehát egyszerre 2 festék fér el az állványon. Kezdetben a 0 és 1 sorszámú van az állványon. Az állványon levőket így jelöljük: [0, 1]. A kérések egy lehetséges kiszolgálása a következő:

Az első kért szín (2) nincs az állványon. A segéd az 1-es helyére teszi, azaz a állvány tartalma: [0, 2].

A következő kérés (0) az állványon van, a segéd pihen.

A harmadik igény (3) nincs az állványon, a segéd a 0-t cseréli le vele, az állvány tartalma: [3, 2].

Az utolsó kért szín (0) nincs az állványon, a segéd a 2-t cseréli le vele, az állvány tartalma: [3, 0].

Vegyük észre, hogy a segéd nem követte Leonardo optimális stratégiáját. Az optimális az lett volna, ha a 2-es szín helyére rakja a harmadik kérést, és így pihenhetett volna az utolsó kérésnél.

A segéd stratégiája korlátozott memóriával

Reggel a segéd kérte Leonardot, hogy a C kéréssort írja le papírra, hogy követni tudja az optimális stratégiát. Mivel Leonardo titokban akarta tartani munkamódszerét, ezért a segédnek nem tarthatta meg a papírt, meg kellett tanulnia a sorozatot.

A segéd rossz memóriájú, legfeljebb M bitet tud megjegyezni. Ezért nem biztos, hogy a teljes C sorozatot helyre tudja állítani memóriájából. Okos módszert kell alkalmaznia az M bit előállítására. Ezt a sorozatot kódsorozatnak hívjuk és A-val jelöljük.

2. példa

Reggel a segéd elolvassa Leonardo C sorozatát. Egy lehetőség, hogy leírja az állvány tartalmat minden kérés után. Például az első példa esetén: [0, 2], [0, 2], [3, 2], [3, 0]. A kezdőállapotot [0, 1] nem kell leírni.

Tegyük fel, hogy M = 16, azaz a segéd 16 bitet tud megjegyezni. Mivel N = 4, minden színt 2 bittel tudunk tárolni. Azaz 16 bit elég az összes állvány állapot tárolására. Így a segéd ezt a kódsorozatot tárolja: A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

Később a segédnek dekódolnia kell a kódsorozatot, hogy a kéréseket teljesíthesse.

Mivel M=16, a segéd a teljes C sorozatra emlékezhet, csak 8 bitet használva. A példa azt illusztrálja, hogy kevesebb bittel is tud elegendő információt tárolni.

Feladat

Két programot kel írnod ugyanazon a programozási nyelven. Ezek egymás után lesznek végrehajtva, de nem kommunikálhatnak egymással.

Az első programot reggel használja a segéd. Adott C sorozatra ki kell számítania az A kódsorozatot!

A második programot később használja. A kapott A kódsorozat alapján kell teljesíteni az egyes kéréseket. Tehát meg kell tudni mondania, hogy adott kérés esetén melyik színt cseréli le az állványon, ha szükséges.

Az első programban egy `ComputeAdvice`(C, N, K, M) eljárást kell megvalósítanod, amely C, N, K és M paraméterek alapján megadja az A kódsorozatot. Az A-t bitenként kell elküldeni a következő eljárással:

- `WriteAdvice`(B) — a B bitet írja az aktuális A kódsorozat végére. Ezt legfeljebb M-szer szabad meghívni.

A második programban az `Assist`(A, N, K, R) eljárást kell megírnod. Bemenő paramétere A, N, K és az A kódsorozat R hossza ($R \leq M$). Ez az eljárás hajtja végre a segédnek javasolt stratégiát, a következő eljárások használatával:

- `GetRequest`() — megadja Leonardo következő szín kérését. (Nem tudjuk a C kérés sorozat további elemeit.)
- `PutBack`(T) — a T színt kell levenni az állványról és ennek helyébe rakni az aktuális kérést. Csak olyan T-re hívhatod, ami az állványon van.

Az `Assist` végrehajtásakor pontosan N-szer kell hívni a `GetRequest` eljárást, mindegyik egy kérést ad a kérések sorrendjében. Ha az adott kérés színe nincs az állványon, akkor meg kell hívni a `PutBack`(T) eljárást azzal a T-vel, aminek a helyébe kell tenni a kért színt, egyébként nem szabad meghívni ezt az eljárást. Ha ezt nem tartod be, akkor programod hibás futással befejeződik.

Egy teszteset akkor fogadnak el, ha a fenti feltételeket teljesíted és a `PutBack` hívásai száma pontosan megegyezik Leonardo optimális stratégiájával. Megjegyezzük, hogy ugyanilyen hívás számot más stratégiával is el lehet érni, azaz más stratégiát is használhatsz.

3. példa

A 2. példát folytatva feltesszük, hogy a `ComputeAdvice` eljárással kiszámoltad A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`) értékét. A következő eljáráshívásokkal közölted ezt: `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`.

A második programban az `Assist` paramétere az A kódsorozat, N = 4, K = 2, és R = 16. Az `Assist` pontosan N = 4-szer hívja a `GetRequest` eljárást. Az egyes kérések után `Assistaz/code>, ha kell, meghívja a <code>PutBack`(T)-ta megfelelő T-vel.

Az alábbi tábla mutatja az 1. példa nem optimális algoritmusa esetén a `PutBack` hívásokat.

GetRequest()	Hívás
2	PutBack(1)
0	-
3	PutBack(0)
0	PutBack(2)

1. részfeladat [8 pont]

- $N \leq 5\ 000$.

Legfeljebb M=65 000 bitet használhatsz.

2. részfeladat [9 pont]

- $N \leq 100\ 000$.

Legfeljebb M=2 000 000 bitet használhatsz.

3. részfeladat [9 pont] =

- $N \leq 100\ 000$.
- $K \leq 25\ 000$.

Legfeljebb M=1 500 000 bitet használhatsz.

4. részfeladat [35 pont]

- $N \leq 5\ 000$.

Legfeljebb M=10 000 bitet használhatsz.

5. részfeladat [39 pont]

- $N \leq 100\ 000$.
- $K \leq 25\ 000$.

Legfeljebb M=1 800 000 bitet használhatsz.

A pontszám függ a kódsorozat R hosszától. Ha R_{max} a maximális hossza a kódsorozatnak az összes tesztesetre, akkor a pontszámod

- 39 pont, ha $R_{max} \leq 200\ 000$;
- 39 $(1\ 800\ 000 - R_{max}) / 1\ 600\ 000$ pont, ha $200\ 000 < R_{max} < 1\ 800\ 000$;

- 0 pont, ha $R_{max} \geq 1\ 800\ 000$.

Megvalósítás

Pontosan két programot kell beküldened, ugyanazon a programozási nyelven!

Az első neve `advisor.c`, `advisor.cpp` vagy `advisor.pas`. Ebben valósítsd meg a `ComputeAdvice` eljárást a `WriteAdvice` hívásokkal. A második neve `assistant.c`, `assistant.cpp` vagy `assistant.pas`. Ebben valósítsd meg az `Assist` eljárást a `GetRequest` és a `PutBack` eljárások hívásával.

Az eljárások definíciói:

C/C++ program

```
void ComputeAdvice(int *C, int N, int K, int M);
void WriteAdvice(unsigned char a);
```

```
void Assist(unsigned char *A, int N, int K, int R);
void PutBack(int T);
int GetRequest();
```

Pascal program

```
procedure ComputeAdvice(var C : array of LongInt; N, K, M : LongInt);
procedure WriteAdvice(a : Byte);
```

```
procedure Assist(var A : array of Byte; N, K, R : LongInt);
procedure PutBack(T : LongInt);
function GetRequest : LongInt;
```

Használhatsz más eljárásokat is. C/C++ esetén a saját eljárásokat `static`-nak kell definiálni, vagy ugyanolyan nevűt ne használj a két programban! A két programod másképp nem kommunikálhat, sem standard iput/outputon, sem file-ban.

A megoldás készítésekor tartsd be az alábbiakat (a megadott mintaprogramok teljesítik ezeket)

C/C++ program

A programod elejére tedd be a megfelelő sort: `advisor.h`, illetve `assistant.h`!

```
#include "advisor.h"
```

vagy

```
#include "assistant.h"
```

A két file (`advisor.h` és `assistant.h`) megtalálható a verseny könyvtárban, vagy le is tölthető. Fordításra és tesztelésre is megkapod scriptet, illetve programot: `compile_c.sh` vagy `compile_cpp.sh`

Pascal program

Használd a megfelelő unit-ot (`advisorlib`, illetve `assistantlib</code>)`!

```
uses advisorlib;
```

`vagy`

```
uses assistantlib;
```

`A két file (<code>advisorlib.pas` és `assistantlib.pas`) megtalálható a verseny könyvtárban, vagy le is tölthető. Fordításra és tesztelésre is megkapod scriptet, illetve programot: `compile_pas.sh`.

Mint aértékelő

A minta értékelő a bemenete az alábbi formában várja:

- 1. sor: N, K, M;
- 2, …, N + 1. sorok: C[i].

Az értékelő először a `ComputeAdvice` eljárást hajtja végre. Ez készít egy `advice.txt` file-t, ami a kódsorozatot tartalmazza, egy-egy szóközzel elválasztva, 2 van a végén.

Ezután indul az `Assist`, aminek eredménye sorai a következő formájúak: "`R [number]`", vagy "`P [number]`". Az első típusú egy `GetRequest()` hívás eredményét tartalmazza, a második típusú pedig egy `PutBack`() hívással megadott színt. A file utolsó sora: "`E`".

A hivatalos értékelő futása lassabb, de nem lényegesen. Időlimit ellenőrzésre a teszt interfészt használd!

Idő és memória limitek

- Időlimit: 7 másodperc.
- Memórialimit: 256 MiB.