Last Supper

Leonardo era foarte activ când lucra la "Cina cea de taină", cea mai faimoasă pictură murală: una din primele sarcini ale zilei era să decidă ce vopsele să folosească în cursul zilei. El avea nevoie de multe culori, dar putea păstra pe schelă doar un număr limitat de culori. Pe lânga alte lucruri, asistentul său trebuia să urce pe schelă pentru a-i furniza culorile și pentru a le coborâ pe un anumit raft pe podea.

În această problemă, trebuie să scrieți două programe separate pentru a ajuta asistentul. Primul program va primi instrucțiunile lui Leonardo (secvența de culori de care Leonardo are nevoie în timpul zilei), și crează un *scurt* string de biți, numit *secvență ajutătoare*. În timpul procesării cerințelor lui Leonardo din cursul zilei, asistentul nu va avea acces la cerințele ce vor urma, el va avea acces doar la secvența ajutătoare produsă de primul program. Al doilea program va primi secvența ajutătoare, și apoi va primi și procesa cerințele lui Leonarda într-o manieră "online" (adică una câte una). Acest program trebuie să înțeleagă ce reprezintă secvența ajutătoare și să facă alegerile optime. Totul este explicat mai jos în detaliu.

Mutarea culorilor între raft și schelă

Vom considera un scenariu simplificat. Presupunem că există N culori numerotate de la 0 la N - 1, și în fiecare zi, Leonardo îi cere asistentului o nouă culoare de exact N ori. Fie C o secnvență de N culori cerute de Leonardo. Putem privi C ca o secvență de N numere, fiecare între 0 și N - 1, inclusiv. Unele culori ar putea să nu apară deloc în C, iar altele ar putea să apară de mai multe ori.

Schela este întotdeauna plină și conține K din cele N culori, cu K < N. Inițial, schela conține culorile de la 0 la K - 1, inclusiv.

Asistentul procesează cererile lui Leonardo una câte una. Oricând culoarea cerută este *deja pe schelă*, asistentul se poate odihni. Altfel, el trebuie să ia culoarea cerută de pe raft și să o ducă pe schelă. Desigur, nu va mai fi loc pentru o nouă culoare, deci asistentul trebuie să aleagă una din culorile existente pe schelă și să o ducă pe raft, astfel încât să facă loc pentru noua culoare.

Strategia optimă a lui Leonardo

Asistentul vrea să se odihnească de cât mai multe ori posibil. Numărul de cereri în care el se poate odihni depinde de alegerile din timpul procesului. Mai precis, de fiecare dată când asistentul trebuie să elimine o culoare de pe schelă, diferite alegeri pot condice la diferite situații viitoare. Leonardo îi explică asistentului cum poate să-și atingă scopul cunoscând secvența C. Cea mai bună alegere de a elimina o culoare de pe schelă se obține examinând culorile existente pe schelă, și ce cerințe au mai rămas în C. O culoare trebuie alese dintre cele de pe schelă respectând următoarele reguli:

- Dacă există o culoare pe schelă care nu va mai fi niciodată necesară în viitor, asistentul trebuie să elimine o astfel de culoare de pe schelă.
- Altfel, culoarea eliminată de pe schelă trebuie să fie aceea care *va fi necesară cel mai târziu*. (Adică, dintre toate culorile de pe schelă, găsim viitoarea sa apariție în cererile care urmează. Culoarea întoarsă pe raft va fi cea care este necesară ultima)

Se poate demonstra că utilizând strategia lui Leonardo, asistentul se va odhni de cele mai multe ori.

Exemplul 1

Fie N = 4, deci avem 4 culori (numerotate de la 0 la 3) și 4 cereri. Fie secvența de cereri C = (2, 0, 3, 0). Presupunem K = 2, adică Leonardo are o schelă capabilă să țină 2 culori în același timp. Cum s-a mai sus, inițial, schela conține culorile 0 și 1, deci conținutul schelei este: [0, 1]. Un posibil mod în care asistentul poate rezolva cererile este următorul.

- Prima culoare cerută (culoarea 2) nu este pe schelă. Asistentul o pune pe schelă și decide să elimine culoarea 1. Acum schela conține culorile [0, 2].
- Următoarea culoare cerută (culoarea 0) se află deja pe schelă, deci asistentul se poate odihni.
- Pentru a treia cerere (culoarea 3), asistentul scoate culoarea 0, schimbând schela în configurația [3, 2].
- În final, a patra cerere (culoarea 0) trebuie să fie mutată de pe raft pe schelă. Asistentul decide să elimine culoarea 2, deci schela devine acum [3, 0].

Se observă că în exemplul de mai sus, asistentul nu a urmat strategia optimă a lui Leonardo. Strategia optimă ar fi eliminat culoarea 2 la pasul 3, deci asistentul s-ar fi putut odihni din nou în pasul final.

Strategia asistentului când memoria sa este limitată

Dimineața, asistentul îi cere lui Leonardă să scrie secvența C pe o foaie de hârtie, astfel încât să găsească și să urmeze strategia optimă. Cu toate acestea, Leonardo este obsedat să-și păstreze tehnicile de lucru secrete, deci refuză să-i permită asistentului să folosească hârtia. El îl lasă pe asistent doar să citească secvența C și să încerce să o memoreze.

Din păcate, asistentul nu are o memorie bună. El este capabil să memoreze cel mult M biți. Din acest motiv, este posibil să nu poate reconstrui întreaga secvență C. Totuși, asistentul trebuie să găsească o idee inteligentă pentru reconcstruiea secvenței folosind biții pe care îi va memora. Vom numi această secvență *secvență ajutătoare* și o vom nota cu A.

Exemplul 2

De dimineață, asistentul poate lua hârtia lui Leonardo cu secvența C, să o citească, și să facă toate alegerile necesare. Un lucru pe care ar putea să-l facă ar fi să examineze starea schelei după fiecare cerere. De exemplu, când folosește strategia (neoptimă) dată în exemplul 1, secvența stărilor schelei ar fi [0, 2], [0, 2], [3, 2], [3, 0]. (Reamintim că el știe că starea inițială a schelei este [0, 1]).

Acum presupunem că M = 16, deci asistentul este capabil să rețină până la 16 biți de informații.

Cum N = 4, se pot memora culorile utilizând 2 biți. De aceea, 16 biți sunt suficienți pentru a stoca secvența de mai sus cu stările schelei. Deci, asistentul poate construi următoarea secvență ajutătoare: A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0).

În cursul zilei, asistentul poate decodifica această secvență ajutătoare și o poate utiliza în alegerile sale.

(Desigur, cu M = 16, asistentul poate să memoreze întreagă secvență C, utilizând doar 8 din cei 16 biți disponibili. In acest exemplu am vrut doar să arătăm că există mai multe opțiuni, fără a da orice solutie bună)

Enunț

Trebuie să scrii *două programe separate* în același limbaj de programare. Aceste programe vor fi executate pe rând, fără ca acestea să poată comunica între ele în timpul execuției.

Primul program va fi cel utilizat de asistent dimineața. Acest program va primi secvența C, iar el trebuie să calculeze secvența de ajutor A.

Al doilea program va fi cel utilizat de asistent în cursul zilei. Acest program va primi secvența ajutătoare A, iar el trebuie să proceseze șirul C cu cererile lui Leonardo. De remarcat că acest program va primi secvența C element cu element, iar fiecare element curent trebuie să fie procesat înainte de a fi primit următorul.

Mai exact, în primul program trebuie implementată o singură rutină `ComputeAdvice`(C, N, K, M) având ca input vectorul C cu N elmente întregi (fiecare în 0, …, N - 1), numărul K de culori care pot încăpea pe schelă, și numărm M de biți disponibili pentru secvența ajutătoare. Acest program trebuie să calculeze secvența ajutătoare A, care poate conține cel mult M biți. După aceasta, programul trebuie să transmită secvența A sistemului, apelând pentru fiecare bit din A (în ordine), următoarea rutină:

- `WriteAdvice`(B) — adaugă bitul B secvenței ajutătoare A (poți apela această rutină de cel mult M ori).

În al doilea program trebuie să implementezi o singură rutină `Assist`(A, N, K, R). Inputul acestei rutine este secvența ajutătoare A, intregii N și K definiți mai sus, și numărul curent de biți R al secvenței A (R ≤ M). Această rutină trebuie să execute strategia propusă de voi asistentului, folosind următoarele rutine care îți sunt oferite:

- `GetRequest`() — returnează următoarea culoare cerută de către Leonardo. (Nu este oferită nicio altă informație referitoare la cererile care urmează).
- `PutBack`(T) — mută culoarea T din schelă înapoi pe raft. Poți apela această rutină doar dacă culoarea T este una din culorile de pe schelă.

În timpul execuției rutina `Assist` trebuie să apeleze `GetRequest` de exact N ori, de fiecare dată primind una din cele N cereri ale lui Leonardo, în ordine. După fiecare apel `GetRequest`, dacă culoarea returnată nu se află pe schelă, trebuie apelat `PutBack`(T) cu T ales de tine. Dacă culoarea se află pe schelă *nu trebuie* apelat `PutBack` deloc. Dacă nu se procesează așa, se

consideră eroare și aceasta va cauza întreruperea execuției programului. Vă amintim că la începutschela conține toate culorile de la 0 la K - 1, inclusiv.

Un test va fi considerat rezolvat doar dacă cele două rutine respectă toate constrângerile, iar numărul total de apeluri `PutBack` este *același* cu cel din strategia optimă a lui Leonardo. De remarcat că dacă există mai multe strategii de a obține același număr de apeluri `PutBack`, se poate utiliza oricare dintre ele. (Adică nu este obligatoriu să fie urmată strategia lui Leonardo, dacă există o strategie la fel de bună).

Exemplul 3

În continuarea exemplului 2, presupunem că în `ComputeAdvice` ai calculat A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Pentru a comunica aceasta sistemului, trebuie să realizezi următoarea secvență de apeluri: `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`.

A doua rutină `Assist` va fi apoi executată, primind secvența A de mai sus, și valorile N = 4, K = 2 și R = 16. Apoi, rutina `Assist` trebuie să facă exact N = 4 apeluri `GetRequest`. De asemenea, după unele dintre aceste cereri, `Assist` trebuie apelat `PutBack`(T) cu alegeri potrivite pentru T.

Tabelul de mai jos arată secvența de apeluri corespunzătoare alegerilor (neoptime) pentru exemplul 1. Liniuța "-" din tabel arată că nu s-a apelat `PutBack`.

`GetRequest()`	Acțiune
2	`PutBack`(1)
0	-
3	`PutBack`(0)
0	`PutBack`(2)

Subtask 1 [8 puncte]

- $N \leq 5\,000$.
- Poți folosi cel mult M = 65 000 biți.

Subtask 2 [9 puncte]

- $N \leq 100\,000$.
- Poți folosi cel mult M = 2 000 000 biți.

Subtask 3 [9 puncte]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Poti folosi cel mult M = 1 500 000 biți.

Subtask 4 [35 de puncte]

- $N \leq 5\,000$.
- Poți folosi cel mult M = 10 000 biți.

Subtask 5 [până la 39 de puncte]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Poți folosi cel mult M = 1 800 000 biți.

Scorul pentru acest subtask depinde de lungimea R a secvenței ajutătoare comunicată de programul tău. Mai precis, dacă R_{max} este cea mai mare (dintre toate testele) lungime a secvenței ajutătoare produsă de rutina `ComputeAdvice`, scorul tău va fi:

- 39 de puncte dacă $R_{max} \leq 200\,000$;
- 39 $(1\,800\,000 - R_{max}) / 1\,600\,000$ puncte dacă $200\,000 < R_{max} < 1\,800\,000$;
- 0 puncte dacă $R_{max} \geq 1\,800\,000$.

Detalii de implementare

Trebuie să trimiți exact două fișiere *în același limbaj de programare*.

Primul fișier trebuie numit `advisor.c`, `advisor.cpp` sau `advisor.pas`. Acest fișier trebuie să implementeze rutina `ComputeAdvice` după cum a fost descrisă mai sus și trebuie să apeleze `WriteAdvice`. Al doilea fișier trebuie numit `assistant.c`, `assistant.cpp` sau `assistant.pas`. Acest fișier trebuie să implementeze rutina `Assist` după cum a fost descrisă mai sus și poate apela `GetRequest` și `PutBack`.

Semnăturile celor două rutine sunt următoarele.

Programe C/C++

```
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();
```

Programe Pascal

```
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;
```

Aceste rutine trebuie să se comporte așa cum s-a descris mai sus. Desicug, sunteți liberi să implementați alte rutine pentru uzul intern. Pentru programele C/C++, rutinele interne trebuie declarate `static`, ca evaluatorul să le poată lega. Alternativ, evidați să aveți două rutine (una în fiecare program) cu același nume. Submisiile voastre nu au voie să interacționeze cu intrarea/ieșirea standard, sau cu orice alt fișier.

Când scrieți soluțiile, trebuie să aveți grije la următoarele instructiuni (modelele pe care le puteți găsi pe calculator, respectă cerințele listate mai jos).

Programe C/C++

La începutul sursei, trebuie să includeți fișierul `advisor.h` în programul advisor și `assistant.h` în programul assistant. Acest lucru poate fi realizat inserând în sursă linia:

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

sau

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

Cele două fișiere `advisor.h` și `assistant.h` vor fi furnizate atât pe calculator (într-un director) cât și pe interfața web a concursului. De altfe, vei avea acces (prin aceleași canale) la cod și scripturi pentru a compila și testa soluția pe calculator. Mai precis, după copierea soluției în directorul cu aceste scripturi, trebuie să rulați `compile_c.sh` or `compile_cpp.sh` (în funcție de limbajul de programare al codului).

Programe Pascal

Trebuie să utilizați uniturile `advisorlib` în programul advisor și `assistantlib` în programul assistant. Acest lucru poate fi realizat inserând în sursă linia:

```
uses advisorlib;
```

sau

```
uses assistantlib;
```

Cele două fișiere `advisorlib.pas` și `assistantlib.pas` vor fi furnizate atât pe calculator (într-un director) cât și pe interfața web a concursului. De altfe, vei avea acces (prin aceleași canale) la cod și scripturi pentru a compila și testa soluția pe calculator. Mai precis, după copierea soluției în directorul cu aceste scripturi, trebuie să rulați `compile_pas.sh`

Exemplu de evaluator

Exemplul de evaluator va accepta inputul în următorul format:

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

Prima dată, evaluatorul va executa rutina `ComputeAdvice`. Acesta va genera un fișier `advice.txt` conținând biții secvenței ajutătoare. Biții vor fi separați prin spații, iar fișierul se va termina cu cifra 2.

Apoi se va trece la execuția rutinei `Assist`, și se va genera output-ul în care, fiecare linie este fie de forma "`R [number]`", fie de forma "`P [number]`. Liniile de primul tip indica apeluri `GetRequest()` împreună cu răspunsul prmit. Liniile de al doulea tip reprezintă apeluri `PutBack`() împreună cu culoarea aleasă pentru a fi pusă înapoi pe raft. Output-ul este terminat cu o linie de forma "`E`".

Precizăm că timpul de rulare al evaluatorului oficial poate să difere puțin față de cel de pe calculatorul local. Această diferență nu ar trebui să fie semnificativă. Totuși, vă sfătuim să utilizați interfața de testare pentru a verifica dacă soluția voastră se încadrează în timp.

Limtele de timp și memorie

- Limita de timp: 7 secunde.
- Limita de memorie: 256 MB.