Последната вечеря

Леонардо работи много активно върху "Последната вечеря" - неговия известен стенопис. Той трябва да реши кои бои да използва. Има нужда от много цветове, но може да запазва ограничен брой на своето скеле, откъдето рисува. Асистентът му има задължението да се изкачва по скелето и да му доставя боите, както и да връща други бои, които да съхранява на определен рафт.

При тази задача ще трябва напишете две отделни програми, за да помогнете на асистента. Първата програма ще получи инструкциите на Леонардо (последователност от цветове, необходими на Леонардо през деня), и трябва да създаде *къс* низ от битове, наречен "препоръчителната последователност". По време на обработката на заявките за деня на Леонардо асистентът няма достъп до следващите заявки на Леонардо, а само към "препоръчителната последователност", представена от първата си програма. Втората програма ще получи "препоръчителната последователност" и заявките на Леонардо.Тази програма трябва да бъде в състояние да разбере какво означава "препоръчителната последователност" и да я използва, за да се направи оптимален избор. Всичко е обяснено по-долу в по-голяма детайлност.

Пренасяне на боите между скелето и рафтовете

Ние ще разгледаме опростена сценарий. Да предположим, че има N цветове, номерирани от 0 до N - 1, и че всеки ден Леонардо изисква от асистента нов цвят точно N пъти. Нека C е последователността от N цвята, изисквани от Леонардо. По този начин ние представяме C като последователност от N числа, всяко от които е между 0 и N - 1, включително. Имайте предвид, че някои цветове може да не се срещат в C, а други могат да се появят няколко пъти.

Скелето е винаги пълно и съдържа някои K от N цветове, K < N. Първоначално скелето съдържа цветове от 0 до K - 1, включително.

Асистентът обработва заявките на Леонардо една по една. Когато поисканият цвят е вече на скелето, асистентът може да си почине. В противен случай, той трябва да вземе искания цвят от рафта и да го премести на скелето. Разбира се, когато няма място на скелето за нови цветове, асистентът трябва да изберете един от цветовете от скелето и да го върне на рафта.

Оптимална стратегия на Леонардо

Асистентът иска да си почива, колкото може повече пъти. Броят на заявките, за който не е нужно да работи, зависи от неговия избор по време на процеса. По-точно, всеки път,

когато асистентът трябва да махне цвят от скелето, различните избори могат да доведат до различни ситуации в бъдеще. Леонардо му обяснява как да постигне целта си, ако знае C. Най-добрият избор на цвят за премахване от скелето е да се прегледат цветовете, намиращи се в момента там и оставащите в C заявки. Цветът трябва да бъде избран измежду тези намиращи се на скелето по следните правила:

- ако на скелето има цвят, който никога няма да е нужен в бъдеще, асистентът трябва да го премахне от там.
- в противен случай, цветът премахнат от скелето трябва да е *е този, който ще бъде използван най-късно в бъдещето*. (Т.е., за всеки цвят на скелето намираме момента на първото му използване в бъдеще. Цветът който трябва да се върне на полицата е този, който най-късно ще е необходим.)

Може да се докаже, че ако се използва стратегията на Леонардо, асистентът ще почива най-много пъти.

Пример 1

Нека N = 4, така че да имаме 4 цвята (номерирани от 0 до 3) и 4 заявки. Нека последователността на заявките е C = (2, 0, 3, 0). Нека K = 2. Т.е., Леонардо има скеле, което може да съдържа 2 цвята във всеки момент. Както бе посочено по-горе, скелето първоначално съдържа цветове 0 и 1. Записваме съдържанието на скелето както следва: [0, 1]. Един възможен начин за асистента, да се справи със заявките е както следва.

- Първата заявка е за цвят (номер 2), който не е на скелето. Асистентът го поставя там и решава да отстрани от скелето цвят 1. Текущото съдържание на скелето е [0, 2].

- Следващият поискан цвят (номер 0) вече е на скелетни конструкции, така че асистентът може да си почива.

- За третата заявка (цвят номер 3), асистентът премахва цвят 0 и така съдържанието на скелето става [3, 2].

- Най-накрая, цветът на последната заявка (цвят номер 0) трябва да бъде преместен от рафта на скелето. Асистентът решава да отстрани цвят 2 и съдържанието на скелето сега става [3, 0].

Имайте предвид, че в по-горния пример асистентът не следва оптималната стратегия на Леонардо. Оптималната стратегия ще премахне цвят 2 в третата стъпка, така че асистентът би могъл да си почива в последната стъпка.

''' Стратегия на асистента, когато паметта му е ограничена '''

На сутринта, асистентът иска от Леонардо да напише C на хартия, така че той да може да намери и да следва оптималната стратегия. Въпреки това, понеже Леонардо е обсебен да запази техниката на работата си в тайна, той отказа да запише на хартия. Той обаче само разрешава на асистента си да прочете C и да се опита да го запомни.

За съжаление, паметта на асистента е много лоша. Той е в състояние да запомни най-много M бита. Като цяло, това може да му попречи да е в състояние да възстанови цялата

последователност C. Следователно, асистетът трябва да използва някой интелигентен начин за изчисляването на последователността, използвайки битовете, които е запомнил. Ние ще наричаме тази последователност "препоръчана последователност" и ние ще я означаваме с A.

Пример 2

На сутринта, асистентът може да вземе от Леонардо хартията със записаната последователност C, да я прочете, и да направи всички необходими избори. Едно нещо, което той може да направи е да разгледа състоянието на скелето след всяка заявка. Например, при използване на суб-оптималната стратегия като в пример 1, последователностите от съдържанията на скелето ще бъдат [0, 2], [0, 2], [3, 2], [3, 0]. (Да напомним, че той знае, че първоначално състояние на скелето е [0, 1].)

Нека да предположим, че имаме M = 16, така че асистентът е в състояние да си спомним до 16 бита на информация. Понеже N = 4, ние можем да съхраним всеки цвят, използвайки 2 бита. Следователно 16 бита са достатъчни, за да се съхрани дадената по-горе последователност от състояния на скелето. По този асистентът изчислява следните "препоръчителни последователности": A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0).

По-късно през деня, асистентът може да декодира препоръчаната последователност и да я използва, за да прави своите избори.

(Разбира се, при M = 16 асистентът може да избере да запомни цялата последователност C, вместо да използва само 8 от разполагаемите 16 бита. В този пример ние само илюстрираме, че той може да има други опции, без те да дадат добро решение.)

Задача

Трябва да напишете "две отделни програми" на един и същ език за програмиране. Тези програми ще се изпълняват последователно, без да са в състояние да общуват помежду си по време на изпълнение.

Първата програма ще се използва сутрин от асистента. На тази програма ще бъде дадена последователността C, и тя трябва да изчисли на "препоръчаната последователност" A.

Втората програма ще се използва през деня от асистента. На тази програма ще се подаде "препоръчаната последователност" A, и след това програмата ще обработи последователността C от заявките на Леонардо. Имайте предвид, че последователността C ще се подава разкрива на тази програма само с по една заявка в момента и тази заявка трябва да бъде обработена преди получаването на следващата.

По-точно, в първата програма трябва да се имплементира една подпрограма `ComputeAdvice`(C, N, K, M), която служи за въвеждане на масива C от N цели числа (всяко измежду 0, …, N - 1), броя K на цветовете, които се съхраняват на скелето, и броя M на битовете за "препоръчана последователност". Вашата програма трябва след това да изпрати на системата последователността A, извиквайки за всеки бит, в реда в който се срещат, следните подпрограми:

- `WriteAdvice`(B) — долепва в края към текущата "препоръчана последователност" A един бит B. (Може да извиквате тази подпрограма най-много M пъти.)

Във втората програма вие трябва да имплементирате единствена подпрограма `Assist`(A, N, K, R). На входа на тази процедура се подава "препоръчана последоватеност" A, целите числа N и K, дефинирани по-горе, и на действителната дължина R на последователността A в бита ($R \leq M$). Тази процедура следва да изпълнява вашата стратегия за асистента, чрез използване на следните процедури, които са осигурени за вас:

- `GetRequest`() — връща следващия цвят, поискан от Леонардо. (Не се получава информация за следващи заявки.)
- `PutBack`(T) — премества цвят T от скелето обратно на рафта. Можете да извиката тази процедура с такова T, което означава цвят, наличен на скелето.

Когато се изпълнява, вашата процедура `Assist`, трябва да извика `GetRequest` точно N пъти и така всеки път да получава поредната заявка на Леонардо. След всяко извикване на `GetRequest`, ако върнатият цвят *не е* на скелето, вие *трябва* да извикате `PutBack`(T) с вашия избор на T. В противен случай, *не трябва* да извиквате `PutBack`. Нарушаването на това изискване ще се счита за грешка и ще предизвика спиране на програмата ви. напомняме, че в началота скелето съдържа цветовете от 0 to K - 1, включително.

Даден тест се счита решен, ако вашите две програми следват всички наложени ограничения, и общият брой извиквания към `PutBack` е "точно равен" на този от оптималнота стратегия на Леонардо. Имайте предвид, че ако са налице няколко стратегии, всяка от които постига същия брой извиквания към `PutBack`, за вашата програма е разрешено да изпълнява всяка от тях. (т.е., не е необходимо да се следва стратегията на Леонардо, ако има друга еднакво добра стратегия.)

Пример 3

Продължавайки Example 2, приемаме че в `ComputeAdvice` вие пресмятате A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). In order to communicate it to the system, you would have to make the following sequence of calls: `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`).

Вашата втора програма `Assist` когата бъде изпълнявана, получавайки горната последователност A, и стойностите N = 4, K = 2, и R = 16. Програмата `Assist` тогава трябва да изпълни точно N = 4 извиквания на `GetRequest`. След някои от тези заявки, `Assist` трябва да извика `PutBack`(T) с подходящо T.

Таблицата по-долу показва поредица от извиквания, които съответстват на (суб-оптимални) избори от Пример 1. Тирето обозначава, че няма извикване към `PutBack`.

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

Subtask 1 [8 points]

- $N \leq 5\,000$.
- Може да използвате най-много M = 65 000 bits.

Subtask 2 [9 points]

- $N \leq 100\,000$.
- Може да използвате най-много M = 2 000 000 bits.

Subtask 3 [9 points]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Може да използвате на-много M = 1 500 000 bits.

Subtask 4 [35 points]

- $N \leq 5\,000$.
- Може да използвате най-много M = 10 000 bits.

Subtask 5 [up to 39 points]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Може да използвате най-много M = 1 800 000 bits.

Точките за тази подзадача зависят от дължината R на "препоръчаната последователност" , с която програмата си комуникира. По-точно, ако R_{max} е максималната (при всички тестови случаи) дължина на "препоръчаната последователност" произведени от вашата процедура `ComputeAdvice`, резултатът ви ще бъде:

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

- 0 точки, ако $R_{max} \geq 1\ 800\ 000$.

Implementation details

Вие трябва да събмитнете точно два файла "в един и същ език за програмиране".

Първият файл се нарича `advisor.c`, `advisor.cpp` или `advisor.pas`. Този файл трябва да имплементира `ComputeAdvice` и да извиква `WriteAdvice`. Вторият файл се нарича `assistant.c`, `assistant.cpp` или `assistant.pas`. Този файл трябва да имплементира `Assist` и може да извиква `GetRequest` и `PutBack`.

Следват прототипите на всички процедури.

C/C++ programs

```
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 programs

```
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;
```

Тези процедури трябва да функционират както е описано по-горе. Разбира се, вие можете да напишете и други процедури за ваша вътрешна употреба. За C/C++ програми, вътрешните ви процедури трябва да бъдат декларирани статично, тъй като пробният грейдер ще ги свърже. Не използвайте две процедури с еднакви имена. Вашите събмити не трябва да взаимодейства по какъвто и да е начин със стандартен вход/изход, нито с друг файл.

При програмиране на решенията, трябва да се погрижите за следните инструкции (шаблоните, които може да намерите в конкурсната среда отговарят на изискванията, посочени по-долу).

C/C++ programs

В началото на вашето решение трябва да включите файла `advisor.h` и `assistant.h`, съответно в advisor и във assistant. Това става с включване във вашия сорс на:

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

или

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

Двата файла `advisor.h` и `assistant.h` ще ви бъдат предоставени в директорията, която е във вашата състезателна среда и също чрез уеб интерфейс. Ще ви бъдат предоставени кодове и скриптове, за да компилирате и тествате вашите решения. След копиране на вашето решение в директорията със скриптовете трябва да извикате `compile_c.sh` или `compile_cpp.sh`.

Pascal програма

Използвайте юнитите `advisorlib` и `assistantlib` в съветника и асистента, съответно. За целта добавете във вашата програма реда:

```
uses advisorlib;
```

или

```
uses assistantlib;
```

Двата файла `advisorlib.pas` и `assistantlib.pas` ще ви бъдат предоставени в папка, намираща се в състезателната среда, както и в Web интерфейса на състезанието. Ще получите също (от същите източници) код и скрипт, необходими да завършите вашето решение. По-специално, след копиране на вашето решение в папката с тези скриптове, трябва да изпълните `compile_pas.sh`.

Примерен грейдър

Пробният грейдер ще приеме вход, форматиран както следва:

- ред 1: N, K, M;
- редове i = 2, …, N + 1: C[i-2].

Грейдерът най-напред изпълнява функцията `ComputeAdvice`. В резултат се генерира файлът `advice.txt`, съдържащ индивидуалните битове на препоръчаната последователност, разделени с интервали и завършващи с 2.

След това грейдерът ще се заеме с изпълнението на вашата функция `Assist` и генериране на изхода, всеки ред на който е или от вида "`R [number]`", или от вида "`P [number]`". Редовете от първия вид означават извиквания на функцията `GetRequest()` и получените отговори. Редовете от втория вид означават извиквания на функцията `PutBack`() и цветовете които избрани за връщане. Изходът завършва с ред от вида "`E`".

Моля, обърнете внимание, че при официалния грейдер времето за работа на вашата

програма може да се различава незначително от това на локалния ви компютър. Тази разлика няма да бъде съществена. Все едно, съветваме ви да използвате тестовия интерфейс, за да проверите дали вашето решение работи в рамките на необходимото време.

Ограничение по време и памет

- Ограничение по време: 7 секунди.
- Ограничение по памет: 256 MiB.