Последната вечера

Леонардо активно работел на сликата „Последната вечера“: една од првите работи кои требало да ги направи во денот е да одлучи кои темперни бои ќе ги користи во текот на денот. Нему му требале многу бои, но можел да чува само ограничен број од нив на неговото скеле. Неговиот асистент бил задолжен да се качува до скелето за да му ги донесе боите и обратно, да се симне, за да ги врати на соодветна полица на подот.

Во оваа задача, ќе треба да напишете две одделни програми за да му помогнете на асистентот. Првата програма ќе ги прима инструкциите на Леонардо (секвенца од бои кои ќе му требаат на Леонардо во текот на денот), и ќе креира еден *краток* стринг од битови, наречен *совет*. Додека ги процесира барањата на Леонардо во текот на денот, асистентот нема да има пристап до идните барања на Леонардо, туку само до советот продуциран од вашата прва програма. Втората програма ќе го прима советот, а потоа ќе ги прима и процесира барањата на Леонардо во онлајн стил (т.е. едно по едно). Оваа програма мора да биде способна да разбира што означува советот и да го користи за да донесува оптимални одлуки. Сето ова е објаснето подолу во повеќе детали.

Префрлање на бои помеѓу скеле и полица

Ќе разгледаме упростено сценарио. Претпостави дека има N бои нумерирани со 0 до N - 1, и дека секој ден Леонардо му побарува нова боја на асистентот точно N пати. Нека C е секвенцата од N-те побарувања на боја, од Леонардо. Значи, C е секвенца од N броеви, секој помеѓу 0 и N - 1, вклучително. Забележете дека некои од боите е можно воопшто да не се појават во C, а други да се појават повеќе пати.

Скелето е секогаш полно и содржи некои K од N-те бои, при што K < N. На почеток, скелето ги содржи боите од 0 до K - 1.

Асистентот ги процесира барањата на Леонардо, едно по едно. Кога бараната боја е *веќе на скелето*, асистентот си лежи. Во спротивно, ја зема бараната боја од полицата и ја префрла на скелето. Бидејќи тоа е веќе полно, тој мора да одлучи една од боите на скелето да ја врати на полицата.

Оптималната стратегија на Леонардо

Асистентот сака да лежи колку повеќе пати може. Бројот на побарувања на кои тој ќе лежи зависи од неговите претходни избори за тоа која боја ќе ја замени. Поточно, секогаш кога бира боја да замени, со различен избор во иднина ќе дојде до различни резултати. Леонардо му објаснил како да го постигне тоа ако го знае C: Најдобар избор која боја да се отстрани од скелето се добива со разгледување на постојните бои на скелето и

преостанатите побарувања од листата C. Се бира боја од тие на скелето според следните правила:

- Ако има боја на скелето која во иднина веќе нема да биде побарана, асистентот треба да отстрани таква боја.
- Инаку, треба да се отстрани онаа боја од скелето што ќе притреба најдоцна во иднината (Т.е., за секоја од боите на скелето го наоѓаме нејзиното прво идно појавување во C. Бојата која треба да се отстрани од скелето е онаа која ќе притреба последна.)

Може да се докаже дека со примена на леонардовата стратегија се добива најмногу пати лежење на асистентот.

Пример 1

Нека N = 4, значи имаме 4 бои (нумерирани од 0 до 3) и 4 побарувања. Нека низата побарувања е C = (2, 0, 3, 0). Исто, нека K = 2. Т.е., Леонардо има скеле кое може да држи 2 бои во даден момент. Според погоре кажаното, скелето на почеток ги има боите 0 и 1. Ќе ја запишеме содржината на скелето вака: [0, 1]. Еден можен начин, асистентот да ги спроведе барањата е следниот.

- Првата барана боја не е на скелето (бојата 2). Асистентот ја става таму и одлучува да ја отстрани бојата 1 од скелето. Состојбата на скелето е [0, 2].
- Следната барана боја (број 0) е на скелето, па асистентот лежи.
- За третото барање (број 3), асистентот ја трга бојата 0, менувајќи го скелето во [3, 2].
- Конечно, последната барана боја (број 0) мора да се однесе од полицата до скелето. Асистентот ја трга боја 2, и скелето сега е [3, 0].

Забележете дека во горниот пример асистентот не ја следел оптималната стратегија на Леонардо. Според оптималната стратегија, во третиот чекор ќе се тргнеше боја 2, па асистентот ќе лежеше во последниот чекор.

Стратегија на асистентот кога меморијата му е ограничена

Наутро, асистентот бара од Леонардо да му ја напише C на ливче, за да ја следи оптималната стратегија. Но, Леонардо сака да си ги скрие техниките на работа и не му дозволува на асистентот да користи хартија за да го запишува C, туку му дозволува само да го прочита C и да се обиде да го запамети.

За жал, асистентот има слаба меморија. Може да памти само до M бита. Во општ случај, тоа може да го спречи да може да ја запамети целата секвенца C. Затоа, асистентот мора да најде некој интелигентен начин за обработка на секвенцата од битови која ќе ја запамети. Оваа секвенца ќе ја нарекуваме *советодавна секвенца* и ќе ја означиме со A.

Пример 2

Наутро асистентот може да го земе ливчето со секвенцата C, да ја прочита истата и да ги

направи сите неопходни избори. Потоа може, на пример, да избере да ги разгледа состојбите на скелето по секое побарување. На пример, кога се користи (полу-оптималната) стратегија дадена во Пример 1, секвенцата од состојби на скелето ќе биде [0, 2], [0, 2], [3, 2], [3, 0]. (Сетете се дека тој знае дека почетна состојба на скелето е секогаш [0, 1].)

Нека M = 16, па асистентот памти до 16 бита информација. Бидејќи N = 4, може да ја запишеме (мемориграме) секоја боја со користење на 2 бита. Затоа 16 бита се доволни да се запише горната секвенца од состојби на скелето. Така, асистентот ја пресметува следната советодавна секвенца: A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

Во текот на денот, асистентот може да ја декодира оваа советодавна секвенца и да ја искористи за да ги прави изборите.

(Се разбира, со M = 16 асистентот наместо тоа, може да одбере да си ја запамти целата C, со само 8 од расположивите 16 бита. Во примерот, сакаме да илустрираме дека постојат и други можности, без да оддадеме некое добро решение.)

Задачата

Треба да напишеш *две засебни програми* во ист програмски јазик. Овие програми ќе бидат извршени по редослед, без да може да комуницираат една со друга за време на извршувањето.

Првата програма ќе биде онаа која утрината ќе ја искористи асистентот. На оваа програма ќе и биде дадена секвенца C, и таа ќе треба да пресмета советодавна секвенца A.

Втората програма ќе е онаа која асистентот ќе ја користи во текот на денот. Оваа програма ќе ја прими советодавната секвенца A, и потоа мора да ја процесира секвенцата C од леонардови барања. Забележи дека секвенцата C ќе биде откриена на оваа програма, едно по едно барање, и секое барање мора да биде процесирано пред да се прими следното.

Поточно, во првата програма треба да имплементираш едноставна рутина `ComputeAdvice`(C, N, K, M) имајќи ја како влез низата C од N цели броеви (секој од 0, …, N - 1), бројот K од бои на скелето, и бројот M на битови достапни за „советот“. Програмава треба да пресмета советодавна секвенца A која ќе се состои од најмногу M битови. Програмата потоа мора да ја предаде секвенцата A преку повикување на следната рутина (и тоа за секој бит во A):

- `WriteAdvice`(B) — надоврзи го битот B кон моментлната советодавна секвенца A. (Може да ја повикуваш оваа секвенца најмногу M пати.)

Во втората програма треба да имплементираш единствена рутина `Assist`(A, N, K, R). Влезот во оваа рутина е советодавната секвенца A, целите броеви N и K според горната дефиниција, и конкретната должина R на секвенцата A во битови ($R \leq M$). Оваа рутина треба да ја изврши твојата предложена стратегија за асистентот, користејќи ги следните рутини кои се обезбедени за тебе:

- `GetRequest`() — ја враќа следната боја што ја побарал Леонардо. (Не се откриваат никакви информации за идните барања.)
- `PutBack`(T) — врати ја бојата T од скелето назад на полицата. Оваа рутина може да ја повикаш само ако T е една од боите кои моментално се наоѓаат на скелето.

При извршување, твојата рутина `Assist` мора да ја повика `GetRequest` точно N пати, притоа секој пат примајќи по едно од барањата на Леонардо, во редослед. После секој повик до `GetRequest`, ако бојата што ја вратила таа *не* е на скелето, тогаш *мора* исто така да ја повикаш и `PutBack`(T) со твојот избор за T. Инаку, *не смееш* да ја повикаш `PutBack`. Доколку не постапиш вака, истото ќе се смета за грешка и ќе предизвика терминирање на твојата програма. Уште еднаш да нагласиме дека на почетокот скелето ги содржи боите од 0 до K - 1, вклучително.

За еден конкретен тест-пример ќе се смета дека е решен ако твоите две рутини ги задоволуваат сите наведени ограничувања, и вкупниот број на повици до `PutBack` е *идентичен* со оној на Леонардовата оптимална стратегија. Да забележиме дека ако постојат повеќе стратегии кои што го остваруваат истиот број на повици до `PutBack`, дозволено е твојата програма да ја спроведе која било од нив. (т.е. не се бара да се следи стратегијата на Леонардо во случај ако постои подеднакво добра друга стратегија.)

Пример 3

Продолжувајќи го пример 2, претпостави дека во `ComputeAdvice` си пресметал дека A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). За да го предадете тоа кон системот, мора да ја направиш следната секвенца на повици: `WriteAdvice`(`0`), `WriteAdvice`(`0`), `WriteAdvice`(`1`), `WriteAdvice`(`0`), `WriteAdvice`(`0`), `WriteAdvice`(`0`), `WriteAdvice`(`1`), `WriteAdvice`(`0`), `WriteAdvice`(`0`), `WriteAdvice`(`1`), `WriteAdvice`(`1`), `WriteAdvice`(`0`), `WriteAdvice`(`1`), `WriteAdvice`(`1`), `WriteAdvice`(`0`), `WriteAdvice`(`0`).

Твојата втора рутина `Assist` потоа ќе се изврши, ќе ја прими горната секвенца A, како и вредностите N = 4, K = 2, и R = 16. Рутината `Assist` мора да изврши N = 4 повици кон `GetRequest`. Исто така, по дел од овие побарувања, `Assist` ќе треба да ја повика `PutBack`(T) со соодветен избор за T.

Табелата покажува секвенца на повици што одговараат на (полу-оптималните) избори од Пример 1. Цртичката означува дека нема повик до `PutBack`.

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

Подзадача 1 [8 поени]

- $N \leq 5\,000$.

- Може да користиш до M = 65 000 битови.

Подзадача 2 [9 поени]

- $N \leq 100\,000$.
- Може да користиш до M = 2 000 000 битови.

Подзадача 3 [9 поени]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Може да користиш до M = 1 500 000 битови.

Подзадача 4 [35 поени]

- $N \leq 5\,000$.
- Може да користиш до M = 10 000 битови.

Подзадача 5 [до 39 поени]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Може да користиш до M = 1 800 000 битови.

Поените на оваа подзадача зависат од должината R од советот кој твојата програма ќе го прати. Поточно, ако R_{max} е максимумот (за сите тест случаи) од должините на советодавни секцвенци продуцирани со твојата рутина `ComputeAdvice`, твојот резултат ќе биде:

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

Имплементациски детали

Треба да предадеш точно 2 фајла *во исти програмски јазик.*

Првиот фајл се вика `advisor.c`, `advisor.cpp` или `advisor.pas`. Фајлот мора да ја имплементира рутината `ComputeAdvice` како што е погоре опишано и може да ја повика рутината `WriteAdvice`. Вториот фајл се вика `assistant.c`,

`assistant.cpp` или `assistant.pas`. Овој фајл мора да ја имплементира рутината `Assist` како што е опишано погоре и може да ги повика рутините `GetRequest` и `PutBack`.

Прототиповите (потписите) за сите рутини следат.

C/C++ програми

```
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 програми

```
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;
```

Овие рутини мора да се однесуваат според објаснувањето погоре. Се разбира, може да имплементираш други рутини за внатрешно користење од споменатите рутини. За C/C++ програми, твоите внатрешни рутини треба да се декларирани `static`, бидејќи грејдерот ќе ги поврзе (излинка) сите нив заедно. Алтернативо, едноставно избегни да имаш 2 рутини (по една во секоја програма) со исто име. Твоите субмисии не смеат да имаат интеракција со стандарден input/output, ниту пак со некој друг фајл.

Кога ќе го програмираш твоето решение, мора да ги земеш предвид и следните инструкции (темплејтите кои ќе ги најдеш во твојата програмска околина веќе ги задоволуваат побарувањата излистани подолу).

C/C++ програми

На почетокот на твоето решение, треба да го вклучиш фајлот `advisor.h` и `assistant.h`, соодветно, во advisor и во assistant. Ова се прави, со тоа што во твојот код ќе ја вклучиш линијата:

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

или

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

Двата фајла `advisor.h` и `assistant.h` ќе ти бидат дадени во директориум во твојот

систем и ќе се понудени и преку Web interface-от на натпреварот. Преку истите канали ќе добиеш и код и скрипти за да го компајлираш и тестираш твоето решение. Поточно, по копирањето на твоето решение во директориумот кој ги има овие скрипти, ќе треба да го извршиш `compile_c.sh` или `compile_cpp.sh` (во зависност од јазикот на твојот код).

Pascal програми

Треба да ги користиш унитите `advisorlib` и `assistantlib`, соодветно, во advisor и во assistant. Тоа се прави со вклучување на линијата:

```
uses advisorlib;
```

или

```
uses assistantlib;
```

Двата фајла `advisorlib.pas` и `assistantlib.pas` ќе ти бидат дадени во директориум во твојот систем и ќе се понудени и преку Web interface-от на натпреварот. Преку истите канали ќе добиеш и код и скрипти за да го компајлираш и тестираш твоето решение. Поточно, по копирањето на твоето решение во директориумот кој ги има овие скрипти, ќе треба да го извршиш `compile_pas.sh`.

Пример-оценувач

Пример-оценувачот ќе прифаќа влез форматиран на следниот начин:

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

Оценувачот прво ќе ја изврши рутината `ComputeAdvice`. Ова ќе генерира документ `advice.txt`, со засебните битови од советодавната секвенца, издвоени со празни места и терминирана со 2.

Потоа ќе продолжи со извршување на твојата `Assist` рутина, и ќе генерира излез (аутпут) во кој секоја линија или има форма "`R [number]`", или форма "`P [number]`". Линиите од прв тип индицираат повици до `GetRequest()` и примените одговори. Линиите од втор тип претставуваат повици до `PutBack`() и боите избрани да се вратат назад. Излезот е терминиран со линија со форма "`E`".

Забележете дека на официјалниот оценувач, времето на извршување на твојата програма може малку да се разликува од времето на локалниот комплутер. Оваа разлика не треба да е значајна. Сепак, те поттикнуваме да го користиш тест интерфејсот за да утврдиш дали твоето решение се извршува во предвидениот временски лимит.

Временски и мемориски ограничувања

- Временско ограничување: 7 секунди.
- Мемориско ограничување: 256 MiB.