Poslední večeře

Leonardo je autorem slavné nástěnné malby Poslední večeře. Při jejím malování potřeboval používat mnoho různých barev, ale na lešení se mu jich vešlo pouze několik. Každé ráno proto určil, které barvy a v jakém pořadí bude potřebovat, a pověřil svého asistenta nošením těchto barev na lešení.

V této úloze budete muset napsat dva programy pro asistenta. První program dostane Leonardovy instrukce (posloupnost barev, které Leonardo bude potřebovat) a vytvoří z nich krátké shrnutí, kterému budeme říkat rada. Během dne nemá asistent přístup k Leonardovým instrukcím, pouze k radě vytvořené prvním programem. Druhý program dostane k dispozici tuto radu, a s její pomocí musí zpracovávat Leonardovy požadavky jeden po druhém, bez znalosti budoucích požadavků (kromě informace obsažené v radě). Tento program musí rozumět informaci zakódované v radě a umět ji využít k provedení optimálních voleb. Následuje přesný popis zadání.

Přenášení barev na lešení

Uvažujeme následující zjednodušenou situaci. Leonardo používá N různých barev očíslovaných od 0 do N - 1 a každý den požádá asistenta o novou barvu právě N-krát. Nechť C je posloupnost Leonardových požadavků, tedy posloupnost N čísel v rozsahu od 0 do N - 1 včetně. Čísla barev v této posloupnosti se mohou opakovat a naopak, některá z barev se v ní nemusí vůbec vyskytnout.

Lešení je vždy plné a je na něm nějakých K z barev, kde $K < N$. Na začátku jsou na lešení barvy 0 až K - 1 včetně.

Asistent vykonává Leonardovy požadavky jeden po druhém. Je-li požadovaná barva již na lešení, asistent nemusí nic dělat a odpočívá. Jinak vezme požadovanou barvu a vynese ji na lešení. Jelikož by tam pro ni nebylo místo, musí si také vybrat jednu z barev na lešení a tu snést z lešení dolů.

Leonardova optimální strategie

Asistent by chtěl odpočívat co nejvíc. Při kolika požadavcích bude moct odpočívat záleží na jeho volbách snášených barev. Leonardo mu poradil následující prokazatelně optimální strategii: vždy je třeba snést takovou barvu, aby situace, že požadovaná barva není na lešení, nastala co nejpozději. Snešenou barvu tedy určí z barev na lešení a zbytku posloupnosti C takto:

- Je-li na lešení barva, která v budoucnosti už nebude nikdy požadována, pak by asistent měl vybrat takovou barvu.
- Jinak by měl vybrat barvu, která bude požadována později než libovolná jiná z barev na lešení.

Příklad 1

Mějme N = 4 barvy očíslované od 0 do 3, a tedy také N = 4 požadavky. Posloupnost požadavků budiž C = (2, 0, 3, 0). Na lešení se vejdou K = 2 barvy. Jak je uvedeno výše, na začátku to jsou barvy 0 a 1. Počáteční stav barev na lešení zapíšeme takto: [0, 1]. Jedna (neoptimální) možnost, jak by asistent mohl vyřizovat požadavky, je tato:

- První požadovaná barva 2 není na lešení. Asistent ji vynese nahoru a rozhodne se snést barvu 1. Stav lešení je [0, 2].
- Další požadovaná barva 0 je na lešení, a proto asistent může odpočívat.
- Třetí požadovaná barva je 3 a asistent snese barvu 0, čímž změní stav lešení na [3, 2].
- Poslední požadovaná barva 0 pak musí být vynesena na lešení. Asistent místo ní sundá barvu 2 a stav lešení bude [3, 0].

V tomto příkladu asistent nepoužíval Leonardovu optimální strategii. Optimální strategie by ve třetím kroku snesla z lešení barvu 2 a asistent by mohl v posledním kroku odpočívat.

Strategie pro zapomnětlivého asistenta

Každé ráno Leonardo napíše asistentovi posloupnost C na kus papíru. Leonardo je ale paranoidní a nedovolí asistentovi si papír nechat. Asistent se musí pokusit zapamatovat si posloupnost C a pak papír spálí.

Asistent má ale špatnou pamět a dokáže si zapamatovat pouze M bitů. To většinou nestačí na zapamatování celé posloupnosti C. Asistent proto musí vymyslet nějakou jinou posloupnost bitů, kterou je schopen si zapamatovat a která mu bude stačit pro optimální vyřizování Leonardových požadavků. Této posloupnosti budeme říkat rada a budeme ji označovat A.

Příklad 2

Ráno si asistent může přečíst posloupnost C a na jejím základě určit všechny optimální volby snesených barev a stav lešení po každém požadavku. Pro (neoptimální) strategii z příkladu 1 je posloupnost stavů lešení [0, 2], [0, 2], [3, 2], [3, 0]. Vynechali jsme počáteční stav lešení, jelikož víme, že je [0, 1].

Mějme M = 16, takže asistent si může zapamatovat nejvýše 16 bitů informace. Jelikož N = 4, každou barvu můžeme reprezentovat dvěma bity. Do 16 bitů tedy můžeme uložit posloupnost stavů lešení, kterou jsme určili v minulém odstavci. Asistent si tak spočítá následující radu: A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

Během dne může asistent posloupnost tvořící radu dekódovat a použít ji k rozhodování, kterou barvu snést z lešení.

(Jelikož M = 16, asistent by si místo toho mohl zapamatovat posloupnost C a použil by pouze 8 bitů. V tomto příkladu jsme chtěli ukázat, že existují i jiné možnosti, ale nijak vám přitom nenapovědět.)

Zadání

Napište dva samostatné programy ve stejném programovacím jazyce. Tyto programy budou spuštěny jeden po druhém a nebudou spolu moci komunikovat.

První program spustí asistent ráno a zadá mu posloupnost C. Tento program vytvoří radu A.

Druhý program bude asistent používat během dne. Tento program dostane radu A a pak bude zpracovávat Leonardovy požadavky. Požadavky z posloupnosti C budou tomuto programu poskytovány jeden po druhém a program musí na každý z nich odpovědět, než dostane další.

V prvním programu implementujte proceduru `ComputeAdvice`(C, N, K, M). Ta dostane za vstup pole C obsahující N celých čísel v rozsahu od 0 do N - 1, celé číslo K udávající počet barev na lešení a celé číslo M udávající maximální počet bitů rady. Procedura `ComputeAdvice` určí radu A skládající se z nanejvýš M bitů a sdělí ji vyhodnocovacímu systému tak, že popořadě pro každý bit z A zavolá následující proceduru:

- `WriteAdvice`(B) — připojí bit B na konec aktuální rady A. Tuto proceduru smíte volat nejvýše M-krát.

Ve druhém programu implementujte proceduru `Assist`(A, N, K, R). Vstupem této funkce je rada A, celá čísla N a K s výše popsaným významem a celé číslo R udávající skutečnou délku rady A v bitech ($R \leq M$). Tato procedura vykonává vámi navrženou strategii pro asistenta a komunikuje s vyhodnocovacím systémem pomocí volání následujících funkcí:

- `GetRequest`() — vrátí číslo následující barvy, kterou bude Leonardo požadovat.
- `PutBack`(T) — snese barvu T z lešení. Tuto proceduru můžete volat pouze pro některou z barev T, které jsou aktuálně na lešení.

Procedura `Assist` musí během svého běhu zavolat `GetRequest` právě N-krát, čímž po řadě obdrží všechny Leonardovy požadavky (posloupnost C). Po každém volání `GetRequest`, jestliže vrácená barva není na lešení, musíte také zavolat `PutBack`(T) se svou volbou snesené barvy T. Je-li vrácená barva na lešení, `PutBack` volat nesmíte (takové volání by bylo považováno za chybu a vedlo by k ukončení vašeho programu). Připomeňme, že na začátku lešení obsahuje barvy 0 až K - 1 včetně.

Testovací vstup bude považován za správně vyřešený, jestliže obě vaše procedury dodrží požadovaná omezení a počet volání `PutBack` je přesně stejný jako v Leonardově optimální strategii. Váš program se nemusí nutně této strategie držet, t.j. existuje-li jiná strategie vykonávající stejný počet volání `PutBack`, může ji použít.

Příklad 3

Pokračujme v příkladu 2 a řekněme, že v `ComputeAdvice` jste určili A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Tuto posloupnost sdělíte vyhodnocovacímu systému následující posloupností volání: `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)`.

Vaše druhá procedura `Assist` bude poté spuštěna s touto posloupností A a s hodnotami N = 4, K = 2 a R = 16. Procedura `Assist` pak musí zavolat právě čtyřikrát `GetRequest`. Po některých z těchto volání musí `Assist` také zavolat `PutBack`(T) s vhodnou volbou T.

Následující tabulka ukazuje posloupnost volání odpovídající (neoptimálním) volbám z příkladu 1. Pomlčka označuje, že `PutBack` není voláno.

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

Podúloha 1 [8 bodů]

- N ≤ 5 000.
- Smíte použít nejvýše M = 65 000 bitů.

Podúloha 2 [9 bodů]

- N ≤ 100 000.
- Smíte použít nejvýše M = 2 000 000 bitů.

Podúloha 3 [9 bodů]

- N ≤ 100 000.
- K ≤ 25 000.
- Smíte použít nejvýše M = 1 500 000 bitů.

Podúloha 4 [35 bodů]

- N ≤ 5 000.
- Smíte použít nejvýše M = 10 000 bitů.

Podúloha 5 [až 39 bodů]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Smíte použít nejvýše $M = 1\,800\,000$ bitů.

Počet bodů za tuto podúlohu závisí na délce R vaší rady. Je-li R_{max} maximum z délek posloupností vyprodukovaných vaší procedurou `ComputeAdvice` pro testovací vstupy, pak získáte následující počet bodů:

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

Implementace

Odevzdejte dva soubory ve stejném programovacím jazyce.

První soubor se jmenuje `advisor.c`, `advisor.cpp` nebo `advisor.pas`. Tento soubor implementuje výše popsanou proceduru `ComputeAdvice` a může volat proceduru `WriteAdvice`. Druhý soubor se jmenuje `assistant.c`, `assistant.cpp` nebo `assistant.pas`. Tento soubor implementuje výše popsanou proceduru `Assist` a smí volat funkci `GetRequest` a proceduru `PutBack`.

Zmiňované funkce mají následující deklarace.

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();
```

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;
```

Můžete samozřejmě implementovat i další pomocné procedury a funkce. Programujete-li v C/C++,

buď deklarujte takové pomocné procedury a funkce `static`, nebo nepoužívejte procedury a funkce pojmenované v obou souborech stejně, jelikož ukázkový vyhodnocovač linkuje oba soubory dohromady. Váš program nesmí vypisovat na standardní výstup, číst ze standardního vstupu ani jinak pracovat se soubory.

Musíte také dodržet následující instrukce (máte k dispozici vzorové soubory, které je splňuji).

C/C++

Na začátek vašeho řešení musíte vložit soubory `advisor.h` resp. `assistant.h`. Toho dosáhnete pomocí následujících příkazů zapsaných na začátek vašich programů:

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

resp.

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

Soubory `advisor.h` a `assistant.h` budou umístěny ve vašich domácích adresářích a budou také dostupné ve webovém rozhraní. Na stejných místech také naleznete kód a skripty pro kompilaci a testování vašeho řešení. Po zkopírování vašeho řešení do adresáře s těmito skripty ho zkompilujete spuštěním `compile_c.sh` či `compile_cpp.sh` v závislosti na vašem programovacím jazyce.

Pascal

Musíte použít unity `advisorlib` resp. `assistantlib`. Toho dosáhnete pomocí následujících příkazů zapsaných na začátek vašich programů:

```
uses advisorlib;
```

resp.

```
uses assistantlib;
```

Soubory `advisorlib.pas` a `assistantlib.pas` budou umístěny ve vašich domácích adresářích a budou také dostupné ve webovém rozhraní. Na stejných místech také naleznete kód a skripty pro kompilaci a testování vašeho řešení. Po zkopírování vašeho řešení do adresáře s těmito skripty ho zkompilujete spuštěním `compile_pas.sh`.

Ukázkový vyhodnocovač

Ukázkový vyhodnocovač očekává vstup v následujícím tvaru:

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

Vyhodnocovač nejprve spustí proceduru `ComputeAdvice`. Tím vytvoří soubor `advice.txt`, obsahující jednotlivé bity rady oddělené mezerami a ukončené číslem 2.

Poté spustí proceduru `Assist` a vygeneruje následující výstup. Na každou řádku vypíše buď "`R [number]`" nebo "`P [number]`". Řádky prvního typu odpovídají voláním `GetRequest()` a vráceným požadavkům. Řádky druhého typu odpovídají voláním `PutBack`() a barvám sneseným z lešení. Výstup je ukončen řádkou obsahující "`E`".

Čas běhu v oficiálním vyhodnocovači může být mírně odlišný od času ve vašem lokálním vyhodnocovači, tento rozdíl by ale neměl být podstatný. Přesto si pro jistotu můžete ověřit, zda vaše řešení běží v požadovaném časovém limitu, pomocí testovacího rozhraní.

Omezení na čas a paměť

- Čas: 7 sekund.
- Paměť: 256 MiB.