Rozbitá klávesnica

Usámec veľmi aktívne programoval a rozbil si klávesnicu tak, že mu ostalo už len niekoľko funkčných tlačidiel (našťastie sa dajú presúvať z jedného klávesu na iný). Ale keďže je najlepší programátor tak už ráno presne dopísmenka vie, čo bude programovať. A má aj asistentku Moniku, ktorá sa okrem iného stará o to, aby tlačidla boli práve na tých miestach, kde treba.

V tejto úlohe bude programovať dva osobitné programy, ktoré pomôžu Monike. Prvý program dostane kód, ktorý sa chystá Usámec napísať (postupnosť stlačení klávesov) a vytvorí *krátky* string nazývaný *pomôcka*. Počas zvyšku dňa Monika nebude mať prístup k Usámcovmu kódu, iba k pomôcke, ktorú vyrobil prvý program. Druhý program dostane pomôcku a potom bude spracovať Usámcov kód online spôsobom (znak po znaku). Tento program musí byť schopný pochopiť, čo pomôcka znamená a použiť ju na vykonanie *optimálnych* rozhodnutí. Všetko je detailne vysvetlené nižšie. Nebojte sa.

Presúvanie tlačidiel po klávesnici

Usámcova klávesnica v nerozbitom stave mala práve N klávesov, očíslovaných od 0 do N - 1. Na napísanie dnešného programu potrebuje práve N stlačení. Označme ako C postupnosť klávesov, ktorú potrebuje stlačiť. C je teda postupnosť N čísel od 0 do N - 1 (vrátane). Niektoré klávesy sa môžu v C vyskytnúť viackrát, niektoré ani raz.

Klávesnica má vždy funkčných práve K z N klávesov (sú na nich tlačidlá), kde $K < N$. Na začiatku fungujú klávesy do 0 do K - 1 (vrátane).

Programovanie vyzerá nasledovne: Usámec vykríkne kláves, ktorý plánuje stlačiť. Pokiaľ je na danom klávese tlačidlo, tak Monika má pokoj. Pokiaľ nie, tak musí z nejakého klávesu tlačidlo vyrvať a presunúť ho na požadované miesto.

Optimálna stratégia

Presúvanie tlačidiel nie je príjemná činnosť, preto chce Monika spraviť najmenší možný počet presunov. Množstvo presúvania závisi hlavne od jej rozhodnutí, kedy odkiaľ tlačidlo presunie. Monika je šikovná. Ak by poznala postupnosť C vopred, vedela by s istotou dosiahnuť svoj cieľ. Spraví to takto: Vždy, keď potrebuje presunúť tlačidlo, pozrie sa na dve veci - na aktuálne rozmiestnenie tlačidiel a tiež na tie stlačenia v postupnosti C, ktoré ešte len budú nasledovať. Výber vhodného tlačidla na presun sa dá popísať nasledovne:

- Pokiaľ je nejaké tlačidlo na klávese, ktorý už v budúcnosti Usámec nikdy nebude potrebovať, zoberieme ho odtiaľ.

- Inak zoberieme tlačidlo z klávesu, ktorý budeme potrebovať čo najneskôr. (T. j. pre každý kláves, ktorý má teraz tlačidlo, nájdeme jeho prvé použitie v budúcnosti. Na odstránenie vyberieme ten, ktorý budeme potrebovať najneskôr.)

Dá sa dokázať, že pri použití tejto stratégie bude vždy počet presunov klávesy najmenší možný.

Príklad 1

Nech N = 4, takže máme 4 klávesy (očíslované 0, 1, 2, 3) a postupnosť C má dlžku 4. Nech C = (2, 0, 3, 0). A tiež nech K = 2. To znamená, že máme 2 fungujúce tlačidlá. A ako bolo povedané vyššie, na začiatku sú na klávesoch 0 a 1. Polohu tlačidiel budeme zapisovať ako: [0, 1]. Jedna možnosť ako bude Monika presúvať klávesy, je nasledovná.

- Prvý požadovaný kláves (2) nemá tlačidlo. Monika ho vyrve z klávesu 1 a presunie ho na kláves 2. Aktuálna poloha tlačidiel je [0, 2].
- Ďalší požadovaný kláves (0) už tlačidlo má, takže Monika nemusí robiť nič.
- Ďalej chce Usámec stlačíť kláves 3, Monika vyrve tlačidlo z klávesu 0 a aktuálna poloha tlačidiel bude [3, 2].
- Nakoniec, ešte treba presunúť tlačidlo na klávesu 0. Monika ho zoberie z klávesu 2 a poloha tlačidiel bude [3, 0].

Všimnite si, že v príklade vyššie Monika nepoužila optimálnu stratégiu. Optimálna stratégia by zobrala v treťom kroku tlačidlo z klávesu 2. Potom by Monika v poslednom kroku nemusela presúvať tlačidlo.

Monikina stratégia, keď má slabú pamäť

Ráno Monika požiadala Usámca, aby si postupnosť C mohla od neho odpísať na svoj papier (a ona mohla klávesy presúvať optimálne). Usámec sa ale bojí, že by sa jeho kód Fínskeho stromu 4.7 rozšíril do sveta a stratil by konkurenčnú výhodu. A tak Monike dovolil len prečítať si postupnosť C a zapamätať si ju.

Nanešťastie, Monikina pamäť je veľmi slabá. Je schopná pamätať si iba M bitov. Vo všeobecnosti jej to bráni zrekonštruovať celú postupnosť C. Preto musí vymyslieť nejaký šikovný spôsob ako vypočítať, aké bity si zapamätá. Túto postupnosť bitov budeme nazývať *pomocná postupnosť* a budeme ju označovať A.

Príklad 2

Ráno sa Monika pozrie na Usámcov papier s postupnosťou C, prečíta si postupnosť a vykoná potrebné rozhodnutia. Napríklad sa môže pozrieť na postupnosť polôh tlačidiel po každej požiadavke. Napríklad, keď používa (neoptimálnu) stratégiu z príkladu 1, tak postupnosť polôh tlačidiel bude [0, 2], [0, 2], [3, 2], [3, 0]. (Nezabudnite, že úvodná poloha tlačidiel je [0, 1]).

Predpokladme, že M = 16, takže Monikina pamäť si je schopná uchovať 16 bitov. Keďže N = 4, tak si každý kláves vieme zapísať pomocou 2 bitov. Preto nám 16 bitov stačí na uloženie postupnosti polôh tlačidiel. Monika si preto zapamätá nasledovnú pomocnú postupnosť: A = (`0`, `0`,

1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0).

Počas dňa potom Monika môže dekódovať svoju poradnú postupnosť a na jej základe vykonať potrebné rozhodnutia.

(Samozrejme, pri M = 16 si môže Monika zapamätať celú postupnosť C, s použitím iba 8 bitov zo 16. V tomto príklade sme len chceli ilustrovať, že existujú aj iné môžnosti. A zároveň sme vám nechceli ukázať žiadne dobré riešenie.)

Úloha

Vašou úlohou je napísať *dva osobitné programy* v jednom programovacom jazyku. Tieto programy budú pustené postupne, bez možnosti komunikovať medzi sebou počas svojho behu.

Prvý program použije Monika ráno. Tento program dostane postupnosť C a má spočítať postuponosť A.

Druhý program použije Monika neskôr. Tento program dostane postupnosť A a má spracúvať postupnosť C - Usámcove požiadavky. Táto postupnosť bude programu ukazovaná postupne po jednej požiadavke. Každá požiadavka musí byť spracovaná skôr ako si program vypýta nasledujúcu.

Presnejšie, v prvom programe treba implementovať jednu funkciu `ComputeAdvice`(C, N, K, M), ktorá dostane na vstup pole C pozostávajúce z N čísel (každé z rozsahu 0, ..., N - 1), číslo K - počet funkčných tlačidiel na klávesnici a číslo M - počet bitov, ktoré si vie Monika zapamätať. Táto funkcia musí spočitať postupnosť A, ktorá má najviac M bitov. Bity postuponosť postupne po jednom odovzdá systému pomocou volaní funkcie:

- `WriteAdvice`(B) — pripojí bit B na koniec pomocnej postupnosti A (túto funkciu môžete zavolať maximálne M-krát).

V druhom programe treba implementovať jednu funkciu `Assist`(A, N, K, R). Vstupom tejto funkcie je pomocná postupnosť A, čísla N a K definované vyššie a dĺžka pomocnej postupnosti R v bitoch (R $\leq$ M). Táto funkcia vykonáva vašu navrhovanú stratégiu pre Moniku, volaním nasledovných funkcií:

- `GetRequest`() — vráti nasledovný kláves, ktorú plánuje stlačiť Usámec. (Vždy sa dozviete len jeden kláves, nedostanete žiadnu informáciu o stlačeniach v budúcnosti.)

- `PutBack`(T) — vyberie tlačidlo z klávesu T a dá ho na kláves, kde ho práve Usámec potrebuje. Túto funkciu môžete zavolať len keď má kláves T na sebe tlačidlo.

Počas svojho behu musí vaša funkcia `Assist` zavolať funkciu `GetRequest` presne N-krát. Po každom zavolaní dostane jednu Usámcovu požiadavku. Pokiaľ vrátený kláves po volaní `GetRequest` práve nemá tlačidlo, *musíte* zavolať `PutBack`(T) s vašou voľbou T. Ináč *nesmiete* volať `PutBack`. Porušenie týchto pravidiel je považované za chybu a spôsobí ukončenie vášho programu. Pripomíname, že na začiatku sú tlačidlá na klávesoch 0 až K-1 (vrátane).

Príslušný test case je považovaný za vyriešený, ak vaše dva programy splnia všetky požiadavky a

navyše bude platiť, že celkový počet volaní `PutBack` je *presne rovnaký* ako v optimálnej stratégii. Všimnite si, že pokiaľ existuje viac možností ako dosiahnúť optimálny počet volaní `PutBack`, váš program môže vykonať ľubovoľnú z nich. (Teda nie je nutne sa držať vyššie popísanej optimálnej stratégie, ak existuje nejaká iná rovnako dobrá.)

Príklad 3

Pokračujme, kde sme prestali v príklade 2. Predpokladajme, že v `ComputeAdvice` ste spočítali A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Pre jej zapísanie do systému treba vykonať nasledovnú postupnosť volaní: `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)`.

Potom bude zavolaná vaša funkcia `Assist`. Tá dostane ako parametre hore uvedenú postupnosť A a hodnoty N = 4, K = 2, a R = 16. Funkcia `Assist` potom vykoná presne N = 4 volaní funkcie `GetRequest`. Po niekorých požiadavkách bude tiež musieť zavolať funkciu `PutBack`(T) s vhodnou voľbou T.

Tabuľka nižšie ukazuje postupnosť volaní, ktorá zodpovedá (neoptimálnym) rozhodnutiam z príkladu 1. Pomlčka označuje nezavolanie funkcie `PutBack`.

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

Podúloha 1 [8 bodov]

- N ≤ 5 000.
- Môžete použiť najviac M = 65 000 bitov

Podúloha 2 [9 bodov]

- N ≤ 100 000.
- Môžete použiť najviac M = 2 000 000 bitov.

Podúloha 3 [9 bodov]

- N ≤ 100 000.
- K ≤ 25 000.

- Môžete použiť najviac M = 1 500 000 bitov.

Podúloha 4 [35 bodov]

- $N \leq 5\,000$.
- Môžete použiť najviac M = 10 000 bitov.

Podúloha 5 [najviac 39 bodov]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Môžete použiť najviac M = 1 800 000 bitov.

Počet bodov za túto podúlohu závisí na dĺžke R pomocnej postupnosti, ktorú váš program pošle. Presnejšie, nech R_{max} je maximálna dĺžka (cez všetky testovacie vstupy) pomocnej postupnosti vyrobenej vašou funkciou `ComputeAdvice`. Potom získate:

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

Implementačné detaily

Odovzdajte presne dva súbory napísané v *rovnakom programovacom jazyku*.

Prvý súbor sa volá `advisor.c`, `advisor.cpp` alebo `advisor.pas`. V tomto súbore musí byť implementovaná funkcia `ComputeAdvice` tak ako je popisané vyššie a môže volať funkciu `WriteAdvice`. Druhý súbor sa volá `assistant.c`, `assistant.cpp` alebo `assistant.pas`. V tomto súbore musí byť implementovaná funkcia `Assist` ako je popisané vyššie a môže volať funkcie `GetRequest` a `PutBack`.

Nasledujú definície funkcií v jednotlivých jazykoch.

C/C++ programy

```
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 programy

```
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;
```

Tieto funkcie sa musia správať ako je popísané vyššie. Samozrejme môžete si implementovať ďalšie vlastné funkcie na vlastné použitie. V C/C++ by tieto pomocné funkcie mali byť deklarované ako statické (použitím kľúčového slova `static`), keďže ukážkový testovač linkuje oba vaše programy do jednej binárky. Alebo sa aspoň vyhnite tomu, že budete mať v oboch programoch funkciu s tým istým názvom. Vaše programy nesmú interagovať so štandardným vstupom/výstupom a so žiadnym iným súborom.

Pri programovaní vašich riešení dodržte nasledovné pokyny. (Šablóny vo vašom prostredí už spĺňajú tieto požiadavky.)

C/C++ programy

Na žačiatku `advisor.cpp` includnite súbor `advisor.h`, resp.na začiatku `assistant.cpp` súbor `assistant.h`. Toto sa robí nasledovným spôsobom vo vašom zdrojovom kóde:

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

alebo

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

Súbory `advisor.h` a `assistant.h` vám budú dodané vo adresári vo vašom prostredí a tiež sa budú dať stiahnuť z webového rozhrania. Tiež dostanete (rovnakým spôsobom) kódy na skompilovanie a testovanie vášho riešenia. Presnejšie, po nakopírovaní vášho riešenia do adresára z týmito skriptami, budeme musieť spustiť príkazy: `compile_c.sh` alebo `compile_cpp.sh` (v závislosti na jazyku vášho kódu).

Pascal programy

Musíte použiť unity `advisorlib` resp. `assistantlib` v súboroch `advisor.pas` resp. `assistant.pas`. Toto sa robí pomocou riadku:

```
uses advisorlib;
```

alebo

```
uses assistantlib;
```

Súbory `advisorlib.pas` a `assistantlib.pas` vám budú dodané vo adresári vo vašom prostredí a tiež sa budú dať stiahnuť z webového rozhrania. Tiež dostanete (rovnakým spôsobom) kódy na skompilovanie a testovanie vášho riešenia. Presnejšie, po nakopírovaní vášho riešenia do adresára z týmito skriptami, budeme musieť spustiť príkaz: `compile_pas.sh`.

Ukážkový testovač

Ukážkový testovač akceptuje nasledujúci formát vstupu:

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

Testovač najprv pustí funkciu `ComputeAdvice`. Toto vyrobí súbor `advice.txt`, obsahujúci jednotlivé bity pomocnej postupnosti oddelené medzerami a ukončené znakom 2.

Potom pokračuje pustením vašej funkcie `Assist` a generuje výstup v ktorom každý riadok je tvaru "`R [číslo]`", alebo "`P [čislo]`". Riadky prvého typu označujú volania `GetRequest()` a vrátené odpovede. Riadky druhého typu označujú volania `PutBack`() a klávesy z ktorých bolo vybrané tlačidlo. Výstup je ukončený riadkom tvaru "`E`".

Dajte si pozor na to, že čas behu programu na oficiálnom testovači môže byť jemne odlišný ako ten na vašom lokálnom počítači. Tento rozdiel by nemal byť významný. Stále však smelo môžete použiť testovacie rozhranie na overenie toho, či váš program beží v stanovenom limite.

Časové a pamäťové limity

- Časový limit: 7 sekúnd.
- Pamäťový limit: 256 MiB.