Última Ceia

Leonardo era muito ativo quando trabalhava na Última Ceia, o seu mural mais famoso: uma das suas primeiras tarefas diárias era decidir quais os tons de cores a usar durante o resto do dia de trabalho. Ele precisava de muitas cores mas apenas podia manter um número limitado delas no seu andaime. O seu assistente estava encarregue, entre outras coisas, de subir no andaime para lhe entregar cores e depois descer para as colocar de volta na prateleira colocada no chão.

Nesta tarefa, tem de escrever dois programas separados para ajudar o assistente. O primeiro programa irá receber as instruções de Leonardo (uma sequência de cores que Leonardo necessitará durante o dia), e criará uma string de bits *curta*, chamada *conselho*. Enquanto processar os pedidos do Leonardo durante o dia, o assistente não terá acesso aos pedidos futuros do Leonardo, apenas ao conselho produzido pelo seu primeiro programa. O segundo programa receberá o conselho, e depois receberá e processará os pedidos do Leonardo numa forma online (i.e., um de cada vez). Este programa deve ser capaz de compreender o que o conselho significa e usá-lo para fazer escolhas ótimas. Tudo é explicado abaixo com maior detalhe.

Movendo cores entre a prateleira e o andaime

Iremos considerar um cenário simplificado. Suponha que existem N cores numeradas de 0 a N-1, e que em cada dia Leonardo pede ao seu assistente uma nova cor exatamente N vezes. Seja C a sequência dos N pedidos de cores feitos por Leonardo. Então, podemos pensar em C como sendo uma sequência de N números, cada um estando entre 0 e N-1, inclusive. Note que algumas cores podem não aparecer de todo em C, e que outras podem aparecer múltiplas vezes.

O andaime está sempre cheio e contém algumas K das N cores, com $K < N$. Inicialmente, o andaime contém as cores de 0 a K-1, inclusive.

O assistente processa os pedidos de Leonardo um de cada vez. Quando uma cor pedida *já está no andaime*, o assistente pode descansar. Caso contrário, ele tem de tirar a cor pedida da prateleira e movê-la para o andaime. Claro que não existe espaço no andaime para a nova cor, e por isso o assistente tem de escolher uma das cores no andaime para retirar e colocar de novo na prateleira.

A estratégia ótima de Leonardo

O assistente quer descansar o máximo possível. O número de pedidos em que ele pode descansar depende das suas escolhas durante o processo. Mais precisamente, cada vez que o assistente tem de remover uma cor do andaime, escolhas diferentes podem levar a resultados diferentes no futuro. Leonardo explica-lhe como pode conseguir o seu objetivo sabendo C. A melhor escolha para a cor a remover do andaime e obtida analisando as cores que estão presentemente no andaime, e os restantes pedidos em C. A cor deve ser escolhida, de entre aquelas no andaime, de acordo com as

seguintes regras:

- Se há uma cor no andaime que nunca será necessária no futuro, o assistente deve remover essa cor do andaime.
- Caso contrário, a cor removida do andaime deve ser aquela *que será necessária mais tarde no futuro*. (Isto é, para cada uma das cores no andaime, encontramos a primeira vez que será necessária no futuro. A cor que volta para a prateleira é a ultima a ser necessária.)

Pode ser provado que, quando usando a estratégia do Leonardo, o assistente vai descansar o máximo de vezes possível.

Exemplo 1

Seja N = 4, de modo que temos 4 cores (numeradas de 0 a 3) e 4 pedidos. Seja a sequência de pedidos C = (2, 0, 3, 0). Assuma também que K = 2, isto é, Leonardo tem um andaime capaz de conter 2 cores ao mesmo tempo. Tal como foi dito atrás, o andaime inicialmente contém as cores 0 e 1. Iremos escrever o conteúdo do andaime assim: [0, 1]. Uma possível maneira de o assistente lidar com os pedidos é a seguinte.

- A primeira cor pedida (número 2) não está no andaime. O assistente coloca-a lá e decide remover a cor 1. O andaime passa a conter [0, 2].
- A próxima cor pedida (número 0) já está no andaime, pelo que o assistente pode descansar.
- Para o terceiro pedido (número 3), o assistente remove a cor 0, mudando o andaime para [3, 2].
- Finalmente, a última cor pedida (número 0) tem de ser movida da prateleira para o andaime. O assistente decide remover a cor 2, e o andaime fica com [3, 0].

Note que no exemplo acima, o assistente não seguiu a estratégia ótima de Leonardo. A estratégia ótima removeria a cor 2 no terceiro passo, para que o assistente pudesse então descansar novamente no passo final.

Estratégia do assistente quando a sua memória é limitada

De manhã, o assistente pede ao Leonardo para escrever a sequência C num pedaço de papel, de forma a que ele possa encontrar e seguir a estratégia ótima. No entanto, Leonardo é obcecado em manter as suas técnicas de trabalho secretas, e então recusa-se a deixar o assistente manter o papel. Ele apenas permitiu ao assistente ler C e tentar lembrar-se da sequência.

Infelizmente, a memória do assistente é muito má. Ele é apenas capaz de se lembrar até M bits. No geral, isto pode evitar que ele seja capaz de reconstruir a sequência C inteira. Por isso, o assistente tem de chegar a uma forma inteligente de computar a sequência de bits que irá recordar. Chamaremos esta sequência de *sequência de conselho* e iremos denotá-la por A.

Exemplo 2

Durante a manhã, o assistente pode receber o papel do Leonardo com a sequência C, ler a sequência, e fazer todas as escolhas necessárias. Uma coisa que ele pode escolher fazer é examinar

o estado do andaime depois de cada pedido. Por exemplo, quando usando a estratégia (sub-ótima) dada no Exemplo 1, a sequência de estados do andaime seria [0, 2], [0, 2], [3, 2], [3, 0]. (Relembre-se que ele sabe que o estado inicial do andaime é [0, 1].)

Agora assuma que temos M = 16, então o assistente é capaz de se lembrar até 16 bits de informação. Como N = 4, podemos guardar cada cor usando 2 bits. Por isso 16 bits são suficientes para guardar a sequência acima de estados do andaime. Assim o assistente computa a sequência de conselho seguinte: A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0).

Mais tarde no dia, o assistente pode descodificar esta sequência de conselho e usá-la para fazer as suas próprias escolhas.

(Claro, com M = 16 o assistente também pode escolher lembrar toda a sequência C, usando apenas 8 dos 16 bits disponíveis. Neste exemplo só queríamos ilustrar que ele tem outras opcões, sem revelar uma boa solucão.)

Enunciado

Tem de escrever *dois programas separados* na mesma linguagem de programação. Estes programas serão executados sequencialmente, sem a capacidade de comunicar um com o outro durante a execução.

O primeiro programa será o usado pelo assistente durante a manhã. A este programa será dada a sequência C, e tem de computar uma sequência de conselho A.

O segundo programa será o usado pelo assistente durante o dia. Este programa receberá a sequência de conselho A, e então terá de processar a sequência C dos pedidos do Leonardo. Note que a sequência C só será revelada a este programa um pedido de cada vez, e cada pedido tem de ser processado antes de receber o próximo.

Mais precisamente, no primeiro programa, tem de implementar uma única rotina `ComputeAdvice`(C, N, K, M) tendo como input o array C de N inteiros (todos os inteiros estão em 0, ..., N-1), o número K de cores que o andaime suporta, e o numero M de bits disponíveis para o conselho. Este programa tem de computar uma sequência de conselho A que consiste, no máximo de M bits. O programa tem então de comunicar a sequência A ao sistema chamando, para cada bit em A por ordem, a seguinte rotina:

- `WriteAdvice`(B) — acrescenta o bit B à sequência de conselho A atual. (Pode chamar esta rotina um máximo de M vezes.)

No segundo programa tem de implementar apenas a rotina `Assist`(A, N, K, R). O input desta rotina é a sequência de conselho A, os inteiros N e K como definidos acima, e o comprimento R da sequência A em bits (R ≤ M). Esta rotina deve executar a sua estratégia proposta para o assistente, usando as seguintes rotinas que lhe são fornecidas:

- `GetRequest`() — devolve a próxima cor pedida pelo Leonardo. (Nenhuma informação sobre pedidos futuros é revelada.)

- `PutBack`(T) — retira a cor T do andaime e coloca-a na prateleira. Só pode chamar esta rotina com T sendo uma das cores atualmente no andaime.

Quando executada, a sua rotina `Assist` deve chamar `GetRequest` exatamente N vezes, cada vez recebendo um dos pedidos do Leonardo, por ordem. Depois de cada chamada a `GetRequest`, se a cor devolvida *não* está no andaime, *tem* de também chamar `PutBack`(T) com a sua escolha de T. Caso contrário, *não pode* chamar `PutBack`. Caso não siga estas instruções, dará origem a um erro que causará a terminação do seu programa. Lembre-se que no início o andaime contém as cores de 0 a K-1, inclusive.

Um caso de teste particular será considerado resolvido se as suas duas rotinas seguirem todas as restrições impostas, e o número total de chamadas a `PutBack` é *exatamente igual* ao da estratégia ótima de Leonardo. Note que se existem varias estratégias que usem o mesmo numero de chamadas a `PutBack`, o seu programa pode usar qualquer uma delas. (I.e., não é necessário seguir a estratégia de Leonardo, se existir outra estratégia que funcione tão bem como ela).

Exemplo 3

Continuando o Exemplo 3, assuma que em `ComputeAdvice` computou A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). De forma a comunicá-lo ao sistema, deve fazer a seguinte sequência de chamadas: `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`).

A sua segunda rotina `Assist` seria então executada, recebendo a sequência A acima, e os valores N = 4, K = 2 e R = 16. A rotina `Assist` tem então de chamar `GetRequest` exatamente N = 4 vezes. Também, depois de alguns destes pedidos, `Assist` terá de chamar `PutBack`(T) com uma escolha apropriada de T.

A tabela abaixo mostra uma sequência de chamadas que correspondem as escolhas (sub-ótimas) do Exemplo 1. O hífen indica que `PutBack` não foi chamado.

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

Subtarefa 1 [8 points]

- $N \leq 5\,000$.
- No máximo pode usar M = 65 0000 bits

Subtarefa 2 [9 points]

- $N \leq 100\,000$.
- No máximo pode usar $M = 2\,000\,000$ bits.

Subtarefa 3 [9 points]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- No máximo pode usar $M = 1\,500\,000$ bits.

Subtarefa 4 [35 points]

- $N \leq 5\,000$.
- No maximo pode usar $M = 10\,000$ bits.

Subtarefa 5 [up to 39 points]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- No máximo pode usar $M = 1\,800\,000$ bits.

A pontuação desta subtarefa depende do comprimento R do conselho que o seu programa comunica. Mais precisamente, se R_{max} é o máximo (de todos os casos de testes) do comprimento da sequência de conselho produzida pela sua rotina `ComputeAdvice`, a sua pontuação será:

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

Detalhes da implementação

Deve submeter exatamente dois ficheiros *na mesma linguagem de programação*.

O primeiro ficheiro chama-se `advisor.c`, `advisor.cpp` ou `advisor.pas`. Neste ficheiro a rotina `ComputeAdvice` deve ser implementada como descrito acima e pode chamar a rotina `WriteAdvice`. O segundo ficheiro chama-se `assistant.c`, `assistant.cpp` ou `assistant.pas`. Neste ficheiro a rotina `Assist` deve ser implementada como descrito acima e pode chamar as rotinas `GetRequest` e `PutBack`.

Seguem-se as assinaturas para todas as rotinas.

Programas em 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();
```

Programas em 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;
```

Estas rotinas devem comportar-se como descrito acima. Claro que é livre de implementar outras rotinas para uso interno. Para programas em C/C++, as suas rotinas devem ser declaradas como `static`, visto que o avaliador fornecido irá ligá-las. Alternativamente, apenas evite ter duas rotinas (uma em cada programa) com o mesmo nome. As suas submissões não devem interagir de qualquer forma com o standard input/output, nem com qualquer outro ficheiro.

Quando estiver a programar a sua solução, também tem de ter em conta as seguintes instruções (os templates que pode encontrar no seu ambiente de concurso já satisfazem os requerimentos listados abaixo).

Programas em C/C++

No inicio da sua solução, tem de incluir o ficheiro `advisor.h` e `assistant.h`, respetivamente, no conselheiro e no assistente. Isto é feito incluindo no seu código a linha:

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

ou

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

Os dois ficheiros `advisor.h` e `assistant.h` serão fornecidos num diretório dentro do seu ambiente de concurso e também serão oferecidos pela interface online do concurso. Também lhe será dado (através dos mesmos meios) código e scripts para compilar e testar a sua solução. Especificamente, depois de copiar a sua solução para o diretório com estes scripts, terá de executar `compile_c.sh` ou `compile_cpp.sh` (dependendo da linguagem do seu código).

Programas em Pascal

Tem de usar as unidades `advisorlib` e `assistantlib`, respetivamente, no conselheiro e no assistente. Isto é feito incluindo no seu código a seguinte linha:

```
uses advisorlib;
```

ou

```
uses assistantlib;
```

Os dois ficheiros `advisorlib.pas` e `assistantlib.pas` serão fornecidos num diretório dentro do seu ambiente de concurso e também através da interface online do concurso. Também lhe será dado (através dos mesmos meios) código e scripts para compilar e testar a sua solução. Especificamente, depois de copiar a sua solução para o diretório com estes scripts, terá de executar `compile_pas.sh`

Avaliador fornecido

O avaliador fornecido aceita input formatado da seguinte forma:

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

O avaliador irá primeiro executar a rotina `ComputeAdvice`. Isto gerará um ficheiro `advice.txt`, contendo os bits individuais da sequência de conselho, separados por espaços e terminados por um 2.

Depois prosseguirá executando a sua rotina `Assist`, e gerará output em que cada linha ou é da forma "`R [number]`", ou da forma "`P [number]`". Linhas do primeiro tipo indicam chamadas a `GetRequest()` e as respostas recebidas. Linhas do segundo tipo representam chamadas a `PutBack`() e as cores escolhidas para remover. O output é terminado por uma linha da forma "`E`".

Note que no avaliador oficial o tempo de execução do seu programa poderá ser ligeiramente diferente do que no seu computador local. Esta diferença não deve ser significativa. Apesar disso, convidámo-lo a usar a interface de teste de forma a verificar se a sua solução corre dentro do limite de tempo.

Limites de Memória e Tempo

- Limite de tempo: 7 segundos.
- Limite de memória: 256 MiB.