Dernier souper (Last Supper)

Léonard était très actif lors de la réalisation du Dernier Souper, sa peinture murale la plus connue : l'une de ses premières activités de la journée était de décider quelles couleurs de tempera il allait utiliser pendant le reste de sa journée de travail. Il avait besoin de nombreuses couleurs, mais ne pouvait garder qu'un petit nombre d'entre elles sur son échafaudage. Son assistant était responsable, entre autres, d'escalader l'échafaudage pour lui livrer les couleurs puis de redescendre pour les remettre sur une étagère placée au sol.

Dans ce sujet, vous devrez écrire deux programmes séparés pour aider l'assistant. Le premier programme recevra les instructions de Léonard (une séquence de couleurs dont Léonard aura besoin au cours de la journée), et créer une *courte* chaîne de bits appelée *conseil*. Alors qu'il traitera les requêtes de Léonard au cours de la journée, l'assistant n'aura pas accès aux requêtes futures de Léonard, mais uniquement au conseil produit par votre premier programme. Le deuxième programme recevra le conseil, puis recevra et traitera les requêtes de Léonard en ligne (i.e. une par une). Ce programme doit être capable de comprendre ce que signifie le conseil et l'utiliser pour effectuer des choix optimaux. Tout est expliqué ci-dessous en détail.

Déplacement des couleurs entre l'étagère et l'échafaudage

Nous considérons ici un scénario simplifié. Supposons qu'il y a N couleurs numérotées de 0 à N-1, et que chaque jour, Léonard demande une nouvelle couleur à son assistant exactement N fois. Appelons C la séquence des N requêtes de couleurs demandées par Léonard. On peut voir C comme une séquence de N nombres, où chacun est compris entre 0 et N - 1 inclus. Remarquez que certaines couleurs peuvent ne pas apparaître du tout dans C, et que d'autres peuvent s'y trouver plusieurs fois.

L'échafaudage est toujours plein et contient un sous-ensemble de K couleurs parmi N, où $K < N$. Au départ, l'échafaudage contient les couleurs de 0 à K - 1 inclus.

L'assistant traite les requêtes de Léonard une par une. Lorsque la couleur demandée est *déjà sur l'échafaudage*, l'assistant peut se reposer. Sinon, il doit prendre la couleur demandée sur l'étagère et la placer sur l'échafaudage. Bien sûr, il n'y a pas de place pour la nouvelle couleur sur l'échafaudage, donc l'assistant doit choisir l'une des couleurs se trouvant sur l'échafaudage et la déplacer de l'échafaudage vers l'étagère.

La stratégie optimale de Léonard

L'assistant veut se reposer autant de fois que possible. Le nombre de requêtes pour lesquelles il peut se reposer dépend de ses choix au cours de ce processus. Plus précisément, chaque fois que l'assistant doit retirer une couleur de l'échafaudage, des choix différents peuvent avoir des

conséquences futures différentes. Léonard lui explique comment il peut atteindre son objectif s'il connaît le contenu de C. Le meilleur choix de couleur à retirer de l'échafaudage est obtenu en examinant les couleurs actuellement sur l'échafaudage et les requêtes de couleurs restantes dans C. Une couleur doit être choisie parmi celles se trouvant sur l'échafaudage selon les règles suivantes :

- S'il y a une couleur sur l'échafaudage qui ne sera jamais nécessaire à l'avenir, l'assistant doit retirer cette couleur de l'échafaudage.
- Sinon, la couleur retirée de l'échafaudage doit être celle *dont la prochaine utilisation sera la plus tardive*. (C'est à dire que pour chacune des couleurs sur l'échafaudage, on trouve sa première occurrence future. La couleur que l'on replace sur l'étagère est celle qui sera demandée le plus tard.)
- On peut prouver qu'en utilisant la stratégie de Léonard, l'assistant se reposera aussi souvent que possible.

Exemple 1

Soit N = 4, nous avons donc 4 couleurs (numérotées de 0 à 3) et 4 requêtes. Prenons la séquence de requêtes C = (2, 0, 3, 0). De plus, supposons que K = 2. Ainsi, Léonard dispose d'un échafaudage capable de contenir 2 couleurs à tout moment. Comme indiqué précédemment, l'échafaudage contient initialement les couleurs 0 et 1. On décrira le contenu de l'échafaudage comme ceci : [0, 1]. Une manière dont l'assistant peut traiter les requêtes est la suivante.

- La première couleur demandée (numéro 2) ne se trouve pas sur l'échafaudage. L'assistant l'y place et décide de retirer la couleur 1 de l'échafaudage. L'échafaudage contient donc maintenant [0, 2].

- La couleur demandée suivante (numéro 0) se trouve déjà sur l'échafaudage, l'assistant peut donc se reposer.

- Pour la troisième requête (numéro 3), l'assistant retire la couleur 0, et change l'échafaudage en [3, 2].

- Enfin, la dernière couleur demandée (numéro 0) doit être déplacée de l'étagère vers l'échafaudage. L'assistant décide de retirer la couleur 2, et l'échafaudage contient maintenant [3, 0].

Remarquez que dans l'exemple ci-dessus, l'assistant n'a pas suivi la stratégie optimale de Léonard. La stratégie optimale serait de retirer la couleur 2 à la troisième étape, ce qui permettrait à l'assistant de se reposer lors de la dernière étape.

Stratégie de l'assistant lorsque sa mémoire est limitée

Le matin, l'assistant demande à Léonard d'écrire le contenu de C sur une feuille de papier, de telle sorte qu'il puisse trouver et suivre la stratégie optimale. Cependant, Léonard tient absolument à garder ses techniques de travail secrètes, et refuse de laisser l'assistant utiliser une feuille de papier. Il ne lui permet que de lire C et d'essayer de s'en souvenir.

Malheureusement, l'assistant a une très mauvaise mémoire. Il n'est capable de mémoriser que jusqu'à M bits. En général, cela l'empêche de reconstruire la séquence C entièrement. L'assistant doit donc trouver une manière intelligente de calculer une séquence de bits qu'il mémorisera. Nous appellerons cette séquence la *séquence conseil* et la noterons A.

Exemple 2

Le matin, l'assistant peut prendre la feuille de Léonard contenant la séquence C, lire cette séquence, et faire tous les choix nécessaires. Il peut par exemple choisir d'examiner l'état de l'échafaudage après chacune des requêtes. Par exemple, en utilisant la stratégie (sous-optimale) donnée dans l'exemple 1, la séquence des états de l'échafaudage serait [0, 2], [0, 2], [3, 2], [3, 0]. (Rappelez-vous qu'il sait que l'état initial de l'échafaudage est [0, 1].)

Supposez maintenant que nous avons M = 16, donc que l'assistant peut se souvenir d'au maximum 16 bits d'information. Comme N = 4, on peut stocker chaque couleur en utilisant 2 bits. Ainsi, 16 bits sont suffisants pour stocker la séquence d'états de l'échafaudage ci-dessus. L'assistant calcule ainsi la séquence conseil suivante : A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

Plus tard dans la journée, l'assistant peut décoder cette séquence conseil et l'utiliser pour faire ses choix.

(Bien sûr, avec M = 16, l'assistant peut aussi choisir de se souvenir de l'intégralité de la séquence C, en n'utilisant que 8 des 16 bits disponibles. Dans cet exemple, on souhaitait simplement illustrer qu'il existe d'autres possibilités, sans vous donner une bonne solution.)

Problème

Vous devez écrire *deux programmes séparés* dans le même langage de programmation. Ces programmes seront exécutés en séquence, sans être capables de communiquer l'un avec l'autre au cours de l'exécution.

Le premier programme sera celui utilisé par l'assistant le matin. Ce programme reçoit la séquence C et doit calculer une séquence conseil A.

Le deuxième programme sera celui utilisé par l'assistant pendant la journée. Ce programme reçoit la séquence conseil A et doit ensuite traiter la séquence de requêtes C de Léonard. Remarquez que la séquence C ne sera révélée à son programme qu'une requête à la fois et que chaque requête doit être traitée avant de recevoir la suivante.

Plus précisément, dans le premier programme, vous devez implémenter une fonction `ComputeAdvice`(C, N, K, M) qui prend en entrée le tableau C de N entiers (chacun dans l'intervalle 0, …, N-1), le nombre K de couleurs sur l'échafaudage, et le nombre M de bits disponibles pour la séquence conseil. Ce programme doit calculer une séquence conseil A qui consiste en jusqu'à M bits. Le programme doit alors transmettre la séquence A au système en appelant, pour chaque bit de A dans l'ordre, la fonction suivante :

- `WriteAdvice`(B) — ajoute le bit B à la séquence conseil A. (Vous pouvez appeler cette fonction au maximum M fois).

Dans le deuxième programme, vous devez implémenter une fonction `Assist`(A, N, K, R). L'entrée de cette fonction est la séquence conseil A, les entiers N et K tels que définis ci-dessus, et la longueur R de la séquence A en bits ($R \leq M$). Cette fonction doit exécuter la stratégie que vous proposez pour l'assistant, en utilisant les fonctions suivantes qui vous sont fournies :

- `GetRequest`() — retourne la couleur suivante demandée par Léonard. (Aucune information sur les requêtes suivantes n'est révélée.)
- `PutBack`(T) — déplace la couleur T de l'échafaudage vers l'étagère. Vous ne pouvez appeler cette fonction que si T est l'une des couleurs actuellement présentes sur l'échafaudage.

Lorsqu'elle est exécutée, votre fonction `Assist` doit appeler `GetRequest` exactement N fois, afin de recevoir à chaque fois, dans l'ordre, une requête de Léonard. Après chaque appel à `GetRequest`, si la couleur qu'elle retourne n'est *pas* sur l'échafaudage, vous *devez* appeler également `PutBack`(T) avec votre choix T. Sinon, vous *ne devez pas* appeler `PutBack`. Ne pas respecter ce principe est considéré comme une erreur et entraînera la fermeture de votre programme. N'oubliez pas qu'au début, l'échafaudage contient les couleurs de 0 à K - 1 inclus.

Un cas de test donné sera considéré comme résolu si vos deux fonctions respectent toutes les contraintes imposées et si le nombre total d'appels à `PutBack` est *exactement égal* à celui de la stratégie optimale de Léonard. Remarquez que s'il existe plusieurs stratégies qui permettent d'obtenir le même nombre d'appels à `PutBack`, votre programme peut suivre n'importe laquelle d'entre elles. (i.e. il n'est pas obligatoire de suivre la stratégie de Léonard, s'il existe une autre stratégie aussi bonne.)

Exemple 3

En continuant l'exemple 2, supposons que dans `ComputeAdvice` vous avez calculé A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Pour la transmettre au système, vous auriez à effectuer la séquence d'appels suivante : `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)`.

Votre deuxième fonction `Assist` serait alors exécutée, recevant la séquence A ci-dessus et les valeurs N = 4, K = 2, et R = 16. La fonction `Assist` doit alors effectuer exactement N = 4 appels à `GetRequest`. De plus, après certaines de ces requêtes, `Assist` doit appeler `PutBack`(T) avec une valeur de T qui convient.

Le tableau ci-dessous présente une séquence d'appels qui correspond aux choix (sous-optimaux) de l'exemple 1. Le tiret indique une absence d'appel à `PutBack`.

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

Sous-tâche 1 [8 points]

- $N \leq 5\,000$.
- Vous pouvez utiliser au maximum M = 65 000 bits.

Sous-tâche 2 [9 points]

- $N \leq 100\,000$.
- Vous pouvez utiliser au maximum M = 2 000 000 bits.

Sous-tâche 3 [9 points]

- $N \leq 100\,000$
- $K \leq 25\,000$.
- Vous pouvez utiliser au maximum M = 1 500 000 bits.

Sous-tâche 4 [35 points]

- $N \leq 5\,000$.
- Vous pouvez utiliser au maximum M = 10 000 bits.

Sous-tâche 5 [jusqu'à 39 points]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Vous pouvez utiliser au maximum M = 1 800 000 bits.

Le score pour cette sous-tâche dépend de la longueur R de la séquence conseil transmise par votre programme. Plus précisément, si R_{max} est le maximum (parmi tous les cas de test) de la longueur de la séquence conseil produite par votre fonction `ComputeAdvice`, votre score sera de :

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

- 0 point si $R_{max} \geq 1\ 800\ 000$.

Détails d'implémentation

Vous devez soumettre exactement deux fichiers *dans le même langage de programmation*.

Le premier fichier est appelé `advisor.c`, `advisor.cpp` ou `advisor.pas`. Ce fichier doit implémenter la fonction `ComputeAdvice` comme décrit ci-dessus et peut appeler la fonction `WriteAdvice`. Le deuxième fichier est appelé `assistant.c`, `assistant.cpp` et `assistant.pas`. Le fichier doit implémenter la fonction `Assist` comme décrit ci-dessus et peut appeler les fonctions `GetRequest` et `PutBack`.

Voici les signatures de toutes les fonctions.

Programmes 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();
```

Programmes 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;
```

Ces fonctions doivent se comporter comme décrit plus haut. Bien sûr, vous êtes libre d'implémenter d'autres fonctions pour un usage interne. Pour les programmes C/C++, vos fonctions internes doivent être déclarées `static` car l'évaluateur d'exemple va les linker ensemble. Une autre possibilité est d'éviter de créer deux fonctions (une dans chaque programme) portant le même nom. Vos soumissions ne doivent en aucun cas interagir avec l'entrée/sortie standard, ni avec tout autre fichier.

Lorsque vous programmez votre solution, vous devez également suivre les instructions suivantes (les templates que vous pourrez trouver sur votre environnement de concours respectent déjà les instructions listées ci-dessous).

Programmes C/C++

Au début de votre solution, vous devez inclure les fichiers `advisor.h` et `assistant.h`, respectivement, dans le programme `advisor` et le programme `assistant`. Ceci se fait en incluant la ligne suivante dans votre source :

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

ou

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

Les deux fichiers `advisor.h` et `assistant.h` vous seront fournis dans un répertoire au sein de votre environnement de concours et seront également disponibles sur l'interface Web du concours. On vous fournira également (aux mêmes endroits) les codes et scripts permettant de compiler et tester votre solution. Plus précisément, après avoir copié votre solution dans le répertoire contenant ces scripts, vous devrez exécuter `compile_c.sh` ou `compile_cpp.sh` (suivant le langage de votre code).

Programmes Pascal

Vous devez utiliser les unités `advisorlib` et `assistantlib`, respectivement dans le programme `advisor` et dans le programme `assistant`. Ceci se fait en incluant cette ligne dans votre source :

```
uses advisorlib;
```

ou

```
uses assistantlib;
```

Les deux fichiers `advisorlib.pas` et `assistantlib.pas` vous seront fournies dans un répertoire à l'intérieur de votre environnement de concours et seront également disponibles sur l'interface Web du concours. On vous fournira également (aux mêmes endroits) le code et les scripts vous permettant de compiler et tester votre solution. Plus précisément, après avoir copié votre solution dans le répertoire contenant ces scripts, vous devrez exécuter `compile_pas.sh`.

Évaluateur d'exemple

L'évaluateur d'exemple accepte une entrée formatée comme suit :

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

L'évaluateur exécutera d'abord la fonction `ComputeAdvice`. Ceci génèrera un fichier `advice.txt` contenant les bits individuels de la séquence conseil séparés par des espaces et suivis d'un 2.

Il exécutera ensuite votre fonction `Assist` et générera une sortie dont chaque ligne est soit de la forme "`R [nombre]`", ou de la forme "`P [nombre]`". Les lignes du premier type représentent des appels à `GetRequest()` et les réponses reçues. Les lignes du deuxième type représentent tous les appels à `PutBack`() et les couleurs choisies comme étant à replacer. La sortie se termine par une ligne de la forme "`E`".

Veuillez noter que sur l'évaluateur officiel, le temps d'exécution peut être légèrement différent du temps sur votre ordinateur local. Cette différence ne devrait pas être significative. Malgré cela, vous êtes invités à utiliser l'interface de test vérifier si votre solution fonctionne dans la limite de temps.

Limites de temps et de mémoire

- Limite de temps : 7 secondes.
- Limite de mémoire : 256 Mio.