Olympiades Régionales d'Informatique Franco-Australiennes

17 mars 2013

Durée : 4 heures

3 problèmes

Problème 1
Algo-fu

Limites de temps et de mémoire : 3 secondes, 128 Mo

Fichier d'entrée : *entrée standard*
Fichier de sortie : *sortie standard*

Vous êtes en chemin pour le temple d'Algo-fu, caché dans les montagnes, où vous prévoyez de devenir un grand maître de l'algo-fu. Avant de recevoir l'enseignement des moines, vous devez d'abord prouver que votre cœur est pur.

Vous avez accès à une carte représentant les montagnes comme une grille rectangulaire, chaque case ayant une certaine hauteur. Vous pouvez vous déplacer d'une case à une case adjacente (est, ouest, nord ou sud), du moment que la hauteur de la nouvelle case est inférieure ou égale à la pureté de votre cœur.

Lorsque vous arrivez sur la case en haut à gauche de la grille (sur la ligne 1 et la colonne 1 et toujours à une hauteur de 0), votre cœur est entaché de l'impureté de la civilisation moderne et a donc une pureté de 0. Heureusement certaines cases vous donnent l'opportunité d'accomplir une tâche qui augmente la pureté de votre cœur d'une certaine valeur, et vous permettent donc d'atteindre de nouveaux sommets. Votre but est de minimiser le nombre de tâches à effectuer afin d'atteindre le temple.

Entrée

- La première ligne contient trois entiers : R, C et T : le nombre de lignes et de colonnes de la carte, et le nombre de tâches disponibles.
- Chacune des R lignes suivantes contient C entiers : les hauteurs des cases correspondantes sur la carte.
- Chacune des T lignes suivantes contient trois entiers : la ligne et la colonne de la tâche sur la carte, et la pureté supplémentaire qu'elle vous apporte si vous l'accomplissez. Il y a au plus une tâche sur chaque case et chaque tâche ne peut être accomplie qu'une seule fois.
- La dernière ligne contient deux entiers : la ligne et la colonne du temple sur la carte.

Sortie

La sortie doit consister en une seule ligne contenant un entier : le nombre minimum de tâches que vous devez accomplir afin de pouvoir vous déplacer sur la case du temple. S'il est impossible d'atteindre le temple, vous devez afficher `-1`.

Exemple d'entrée

```
5 4 4
0 0 5 3
1 2 3 4
0 1 6 7
2 3 9 5
1 3 8 3
1 2 1
3 1 1
3 2 2
5 2 4
4 4
```

Exemple de sortie

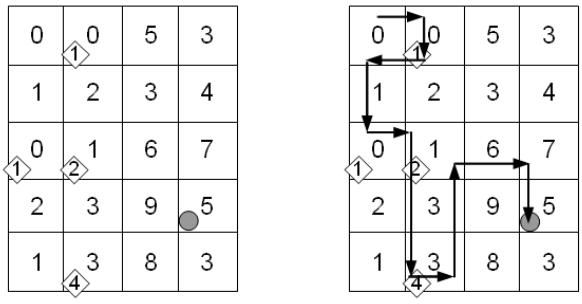
```
3
```

Explication

La carte consiste en 5 lignes et 4 colonnes, avec des hauteurs allant de 0 à 9.

Il y a 4 tâches disponibles, qui se trouvent toutes sur les deux premières colonnes de la carte, comme illustré à gauche ci-dessous. Le temple se trouve sur la dernière colonne à la quatrième ligne. Il peut être atteint en accomplissant les trois tâches de la deuxième colonne, en suivant le chemin illustré à droite ci-dessous.

La première tâche augmentre votre pureté de 1. Cela vous permet d'atteindre la deuxième tâche sur la deuxième colonne, qui vous ajoutera 2 unités de pureté, pour un total de 3. Enfin vous pouvez atteindre la troisième tâche sur la deuxième colonne, qui ajoute 4 à la pureté de votre cœur, soit un total de 7, suffisant pour atteindre le temple.



Sous-tâches

Pour toutes les sous-tâches, $R \geq 1$, $C \geq 1$, $T \geq 0$, toutes les hauteurs valent au moins 0 et la pureté des tâches au moins 1, et les tâches et le temple se trouvent sur une ligne entre 1 et R inclus, et sur une colonne entre 1 et C inclus.

- Pour la sous-tâche 1 (25 pts), $R, C \leq 100$, hauteurs ≤ 100, $T \leq 1\,000$, puretés des tâches ≤ 100.
- Pour la sous-tâche 2 (25 pts), $R, C \leq 1\,000$, hauteurs $\leq 1\,000$, $T \leq 1\,000$, puretés des tâches ≤ 100.
- Pour la sous-tâche 3 (50 pts), $R, C \leq 1\,000$, hauteurs $\leq 100\,000$, $T \leq 100\,000$, puretés des tâches $\leq 1\,000$.

Problème 2
Super Maria

Limites de temps et de mémoire : 1 seconde, 128 Mo

Maria est électricienne et vit dans la République Champignonne, où un désastre se produit toutes les quelques années. Maria finit toujours par être celle qui doit rétablir l'ordre. Cette année sa soeur diabolique Waria a kidnappé son Prince Nectarine, et se retrouve donc de nouveau lancée dans une longue quête étrange pour le sauver.

Les amies de Maria l'appellent Super Maria car elle possède trois super-pouvoirs :

- La capacité épatante consistant à courir vers la gauche.
- La capacité épatante consistant à courir vers la droite.
- La capacité ordinaire consistant à se téléporter **de n'importe quelle position** vers un *récepteur de téléporteur*. Une fois que Maria s'est téléportée vers un récepteur, il est vidé de son énergie et ne peut pas être réutilisé. Ainsi, Maria peut se téléporter vers chaque récepteur au maximum une fois.

Pour une raison inexpliquée, la quête de Maria nécessite qu'elle récolte une séquence de pièces d'or posées sur une line droite (une *plate-forme*). Il y a aussi au moins un récepteur de téléporteur sur cette plate-forme. Maria doit **d'abord se téléporter vers l'un des récepteurs** car elle ne se trouve pas initialement sur la plate-forme, puis utiliser ses trois super-pouvoirs pour se déplacer et récolter toutes les pièces. Maria n'aime pas courir dans tous les sens sur une plate-forme comme un ridicule plombier en récoltant des objets divers, et elle vous a donc demandé d'écrire un programme qui calcule la distance totale minimale qu'elle doit courir.

Maria se moque du choix du récepteur par lequel elle devra commencer, et de l'endroit où elle se trouvera sur la plate-forme après avoir récolté toutes les pièces. Deux pièces ne seront jamais à la même position et deux récepteurs ne seront jamais à la même position (par contre, une pièce et un récepteur peuvent partager la même position).

Entrée

- La première ligne de l'entrée contiendra deux entiers P et R : le nombre de pièces et le nombre de récepteurs de la plate-forme.
- La deuxième ligne de l'entrée contiendra P entiers, dans l'ordre croissant. Le $i$ème entier est la distance (en mètres) de la $i$ème pièce depuis le bord gauche de la plate-forme.
- La troisième ligne de l'entrée contiendra R entiers, dans l'ordre croissant. Le $i$ème entier est la distance (en mètres) du $i$ème récepteur depuis le bord gauche de la plate-forme.

Sortie

La sortie doit consister en une ligne contenant un entier : la distance minimale que Maria doit courir pour récolter toutes les pièces.

Exemple d'entrée

```
4 2
5 10 18 41
14 32
```

Exemple de sortie

```
26
```

Explications

Maria peut faire ce qui suit :

1. Se téléporter vers le récepteur à 32m
2. Courir vers la droite et récolter la pièce à 41m
3. Se téléporter vers le récepteur à 14m
4. Courir vers la droite et récolter la pièce à 18m
5. Courir vers la gauche et récolter les pièces à 10m puis à 5m

Elle aura alors récolté toutes les pièces et couru une distance totale de $9 + 4 + 13 = 26$ mètres.

Sous-tâches

Pour toutes les sous-tâches, $P \geq 1$, $R \geq 1$ et toutes les positions des récepteurs et pièces sont des entiers entre 0 et L (la longueur de la plate-forme) inclus.

- Pour la sous-tâche 1 (15 pts), $P \leq 100$, $R \leq 2$, $L \leq 1\,000\,000$.
- Pour la sous-tâche 2 (25 pts), $P \leq 100$, $R \leq 100$, $L \leq 1\,000\,000$.
- Pour la sous-tâche 3 (25 pts), $P \leq 1\,000$, $R \leq 1\,000$, $L \leq 1\,000\,000\,000$.
- Pour la sous-tâche 4 (35 pts), $P \leq 100\,000$, $R \leq 100\,000$, $L \leq 1\,000\,000\,000$.

Problème 3
Torusia

Limites de temps et de mémoire : 1 seconde, 128 Mo

Les scientifiques Alison et Bill participent à une expédition scientifique sur Torusia, une planète récemment découverte en forme de donut (les cosmologistes se creusent toujours la tête). Lors d'une expérience de routine, les deux scientifiques ont été séparés et des vents solaires intenses ont détruit leur équipement de communication. Chacun n'a aucune idée de la position de l'autre, mais ils disposent chacun d'une machine conçue pour une expérience scientifique, et peuvent utiliser pour se retrouver.

Alison et Bill voient tous deux Torusia comme une grille de 4096 x 4096 cases, où (0, 0) est le coin **haut-gauche**. La grille "fait le tour" aux bords de telle sorte que les lignes horizontales sont en fait des cercles horizontaux, et que les lignes verticales sont des cercles verticaux. Ainsi, le point à l'est de (4095, 34) est (0, 34) et le point au nord de (17, 0) est (17, 4095).

Alison voit sa propre position comme ayant pour coordonnées (0, 0), et Bill voit également sa propre position comme (0, 0) mais leurs positions absolues sont différentes, car leurs systèmes de coordonnées ne sont pas alignés. Notez cependant qu'ils ont la même orientation (les directions du nord, sud, est et ouest), donc si Alison est à x_A mètres à l'est et à y_A mètres au sud de Bill, alors un point qu'Alison voit comme (x, y) correspond au point de la grille de Bill identifié comme $((x + x_A) \mod 4096, (y + y_A) \mod 4096)$.

Alison dispose d'une machine qui peut effectuer une opération :

- **mark(x, y)** : crée un marqueur électromagnétique x mètres à l'est et y mètres au sud de sa position (la case (x, y) de sa grille). Appeler **mark(x, y)** sur une case qui contient déjà un marqueur ne crée pas d'autre marqueur. Alison doit s'assurer que $0 \leq x, y \leq 4095$.

Bill dispose d'une machine qui peut effectuer deux opérations :

- **numRow(y)** : détermine le nombre de marqueurs électromagnétiques qui se trouvent sur la rangée située y mètres au sud de sa position. Bill doit s'assurer que $0 \leq y \leq 4095$.
- **numColumn(x)** : trouve le nombre de marqueurs électromagnétiques qui se trouvent sur la colonne située x mètres à l'est de sa position. Bill doit s'assurer que $0 \leq x \leq 4095$.

Chaque minute, Alison et Bill peuvent utiliser leurs machines respectives pour effectuer une fonction. La machine d'Alison démarre un peu plus vite que celle de Bill, donc à chaque minute, vous pouvez considérer que son marqueur est placé *avant* que Bill n'effectue sa requête.

Vous êtes capable d'envoyer un programme à Alison et Bill pour aider Alison à placer des marqueurs et Bill à effectuer des mesures pour que Bill puisse déterminer la position d'Alison le plus vite possible.

Bibliothèque

Votre fichier source va interagir avec des fonctions fournies dans le fichier source `science.h`. Vous devez implémenter les deux fonctions suivantes :

- `void alison();` qui peut effectuer des appels à `mark(x, y)`. Notez que chaque fois que vous appelez `mark`, le temps écoulé est augmenté d'une minute. Vous pouvez appeler `mark` au maximum 10 000 fois avant de sortir de votre fonction, au delà de quoi l'exécution de votre programme sera interrompue.
- `void bill();` qui peut effectuer des appels à `numRow(y)` et `numColumn(x)`. Notez que chaque fois que l'une de ces fonctions est appelée, le temps écoulé est augmenté d'une minute. Bill doit finalement appeler `found(x, y)` pour indiquer qu'il pense qu'Alison se trouve à x mètres à l'est et y mètres au sud de Bill. Appeler `found` termine l'exécution de votre programme.

Votre code ne doit pas contenir de fonction "main", celle-ci étant fournie par `science.h`, de même que les fonctions suivantes :

`void mark(int x, int y);` qui ne peut être appelée (directement ou indirectement) que par Alison.
`int numRow(int y);` qui ne peut être appelée (directement ou indirectement) que par Bill.
`int numColumn(int x);` qui ne peut être appelée (directement ou indirectement) que par Bill.
`void found(int x, int y);` qui ne peut être appelée (directement ou indirectement) que par Bill.

Vous pouvez considérer que toutes ces fonctions tournent en temps constant.

Compilation

Vous devez ajouter `#include "science.h"` au début de votre code, et vous assurer que le fichier `science.h` se trouve dans le même répertoire que votre code.

Expérimentation

Tandis qu'Alison et Bill effectuent en théorie leurs actions simultanément, nous pouvons simuler cela en exécutant d'abord le programme "en tant qu'Alison", ce qui crée un fichier de log stockant les cases marquées chaque minute, puis en exécutant le programme "en tant que Bill". L'exécutable créé lira une ligne sur l'entrée standard, pour déterminer quel scientifique doit être considéré.

- Vous pouvez saisir une ligne contenant uniquement le caractère '`A`', ce qui exécutera le code d'Alison et génèrera un fichier `mark_log`. Conseil : ce fichier peut vous être utile pour débugger !
- Vous pouvez saisir le caractère '`B`' suivi de deux entiers séparés par une espace x_a et y_a, représentant la position d'Alison par rapport à celle de Bill. Ceci exécutera le code de Bill et affichera le résultat. Vous devez exécuter le code d'Alison avant celui de Bill, car celui de Bill nécessite la présence du fichier `mark_log`.

Il peut être utile de changer temporairement la valeur de la constante `SIZE` en haut du fichier `science.h`, pour tester votre programme sur des grilles plus petites.

Exemple de session

L'exemple de session qui suit est basé sur le code suivant :

```
#include "science.h"

void alison() {
    mark(1, 100);
    mark(2000, 100);
}

void bill() {
    int a = numRow(120);
    int b = numColumn(904);
    int c = numRow(120);
    found(3000, 20);
}
```

Tout d'abord, le programme est exécuté et la ligne suivante est saisie sur l'entrée standard (à l'écran) :

```
A
```

Le programme exécute `alison()`, qui effectue les appels de fonctions suivants :

Appel de fonction	Explication
`mark(1, 100)`	Lors de la première minute, Alison marque la case à 1m à l'est et 100m au sud de sa position.
`mark(2000, 100)`	Lors de la deuxième minute, Alison marque la case à 2000m à l'est et 100m au sud de sa position.

Le programme se termine avec succès et crée un fichier `mark_log`. Nous pouvons maintenant exécuter de nouveau le programme et fournir l'entrée suivante, qui place Alison à 3000m à l'est et à 20m au sud de Bill.

```
B 3000 20
```

Le programme exécutera `bill()`, qui effectue les appels de fonctions suivants :

Appel de fonction	Explication
`numRow(120)`	retourne 1, car lors de la première minute Alison a marqué une case à 100m au sud de sa position, et elle se trouve à 20m au sud de Bill, donc il y a une case marquée sur la rangée se trouvant à 120m au sud de la position de Bill.
`numColumn(904)`	retourne 1, car lors de la deuxième minute, Alison qui est à 3000m à l'est de Bill, a marqué une case se trouvant à 2000m à l'est, ce qui correspond à une case à 904m à l'est de Bill, car la grille fait 4096 de large.
`numRow(120)`	retourne 2, car il y a maintenant 2 marqueurs sur la rangée à 120m au sud de Bill
`found(3000, 20)`	Par une chance incroyable, Bill trouve la position exacte d'Alison après seulement 3 minutes.

Score

Votre programme obtiendra un score basé sur le temps qu'il faut à Bill pour trouver Alison. Plus précisément, si `found` est appelée avec les bons paramètres après M minutes :

$$\text{score} = \begin{cases} 100, & \text{si } M \leq 144 \\ \frac{13000}{M} + 10, & \text{si } 144 < M \leq 10\,000 \\ 0, & \text{si } 10\,000 < M \end{cases}$$

Voici un tableau qui montre dix exemples de scores et le nombre de minutes correspondant :

Score	11	20	30	40	50	60	70	80	90	100
M	10 000	1 300	650	433	325	260	216	185	162	144

Il n'y a pas de sous-tâche pour ce problème. Votre score total sera le score minimum que votre programme obtiendra parmi l'ensemble des fichiers tests.