# Olympiades Régionales d'Informatique Franco-Australiennes

## 12 mars 2017

**Durée : 4 heures**

**3 problèmes**

# Problème 1
# Programmation en binôme

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

**Limites de temps et de mémoire :** 3 secondes, 256 Mo

Vos élèves se crient dessus. Certains se cachent sous les tables, d'autres se tiennent debout dessus. Pendant que certains rejouent les fameux duels de sabres lasers avec leurs stylos, d'autres se balancent sur les ventilateurs accrochés au plafond... En rentrant dans la salle, vous êtes horrifiés de trouver votre classe de programmation en train de semer une telle zizanie. Les épaules baissées et la paume de votre main appuyée fermement contre votre visage, vous tentez de restaurer un semblant d'ordre en demandant à quelques élèves de travailler par binômes.

Il y a $N$ élèves dans votre classe. Chaque élève écrit dans un certain langage de programmation, et utilise un certain éditeur de texte. Vous avez remarqué qu'un binôme est *coopératif* si ses deux élèves programment dans le même langage ou bien utilisent le même éditeur, ou les deux.

Vous aimeriez écrire un programme qui trouve le plus grand nombre de binômes coopératifs que l'on puisse former, telles qu'aucun élève n'est présent dans plus d'un binôme. Votre programme doit aussi fournir un ensemble valide de tels binômes (voir les sections *Sortie* et *Score* plus bas pour plus de détails).

## Entrée

La première ligne de l'entrée contient un seul entier $N$ : le nombre d'élèves dans votre classe. Les élèves sont numérotés de 1 à $N$.

$N$ lignes s'en suivent, la $i$-ème contient deux entiers $l_i$ $e_i$ : l'identifiant du langage et l'identifiant de l'éditeur que l'élève $i$ utilise. Les langages et éditeurs sont représentés par des nombres compris entre 1 et $N$, inclus.

## Sortie

La première ligne de la sortie doit contenir un seul entier $A$ : le nombre maximum de binômes coopératifs qu'il est possible de former.

Pour obtenir la totalité des points, $A$ lignes doivent suivre. Chacune de ces lignes doit contenir deux entiers identifiant deux élèves qui constitueront un binôme de travail. Aucun identifiant d'élève ne doit apparaître plus d'une fois parmi ces $A$ lignes. L'ordre de ces lignes ainsi que l'ordre des élèves sur chaque ligne n'est pas important. S'il y a plus d'un plus grand ensemble de binômes valides, n'importe lequel d'entre eux sera accepté.

Voir la section *Score* ci-dessous pour plus de détails.

## Exemple d'entrée

```
10
3 6
8 2
4 4
4 7
5 2
5 4
```

```
2 1
7 4
9 10
9 10
```

## Exemple de sortie

```
4
2 5
8 6
3 4
9 10
```

## Explications

On peut mettre ensemble :

- L'élève 2 et l'élève 5 (car les deux utilisent l'éditeur 2) ;
- L'élève 8 et l'élève 6 (car les deux utilisent l'éditeur 4) ;
- L'élève 3 et l'élève 4 (car les deux programment dans le langage 4) ; and
- L'élève 9 et l'élève 10 (car les deux programment dans le langage 9 et utilisent l'éditeur 10)

pour un total de 4 binômes, ce qui est le maximum que l'on peut former dans ce cas. Notez que certains langages et certains éditeurs ne sont utilisés par aucun étudiant.

## Sous-tâches & Contraintes

Pour toutes les sous-tâches, $1 \leq N \leq 1\,000\,000$ et pour tout $1 \leq i \leq N$, $1 \leq l_i, e_i \leq N$.

- Pour la sous-tâche 1 (20 points), le langage dans lequel programme un étudiant a le même identifiant que l'éditeur qu'il utilise. Plus précisément, $l_i = e_i$ pour tout $i$.
- Pour la sous-tâche 2 (20 points), $N \leq 50$.
- Pour la sous-tâche 3 (20 points), $N \leq 1\,000$.
- Pour la sous-tâche 4 (40 points), il n'y a pas de contrainte supplémentaire.

## Score

Pour chaque test, vous recevrez un score de 100% si l'entier $A$ sur la première ligne de la sortie est correct et que vous fournissez un ensemble valide de $A$ binômes coopératifs. Sinon, si l'entier $A$ est correct, vous recevrez 40% pour ce test *même si* :

- Vous ne fournissez aucun ensemble de binômes ; ou
- Vous tentez de fournir un ensemble de binômes, mais celui-ci n'est pas un ensemble valide de $A$ binômes coopératifs.

Autrement, vous recevrez un score de 0% pour ce test.

Votre score pour chaque sous-tâche sera le score **minimum** parmi tous les tests de cette sous-tâche. Votre score pour ce problème sera le score **maximum** parmi toutes les soumissions que vous aurez effectué pour ce problème.

# Problème 2
# Gâteau pyramidal

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

**Limites de temps et de mémoire :** 4 secondes, 256 Mo

C'est votre anniversaire ! Étant fervent égyptophile, vous organisez une soirée sur le thème de l'Égypte antique, et vous décidez de commander un gâteau pyramidal dans un célèbre café hipster de Melbourne. Dans la plus pure tradition hipster, les gâteaux sont vendus dans des boîtes de forme unique et complètement inadaptée. La base de la boite est une grille rectangulaire de $R$ lignes numérotées de 1 à $R$, de haut en bas, et de $C$ colonnes numérotées de 1 à $C$, de gauche à droite. Cependant, la hauteur de la boîte est différente en chaque cellule de la grille, et pour éviter que le gâteau arrive tout écrasé à votre soirée, la hauteur du gâteau en chaque cellule ne doit pas dépasser la hauteur de la boîte en cette cellule.

Un gâteau pyramidal se compose d'un ou plusieurs étages rectangulaires, dont les côtés sont parallèles aux côtés de la boîte. Chaque étage est ancré à la boîte par le coin en haut à gauche : il doit donc toujours couvrir la cellule $(1, 1)$. Pour garantir l'intégrité structurelle de votre gâteau, chaque étage doit reposer entièrement sur étage du dessous, de sorte que chaque cellule qui est couverte par un étage est également couverte par tous les étages situés en dessous.

Le volume d'un gâteau pyramidal est la somme du nombre de cellules couvertes par chaque étage du gâteau. Les gâteaux de ce café sont vraiment délicieux, et vous voulez en manger le plus possible. Vous aimeriez donc écrire un programme pour maximiser le volume de votre gâteau pyramidal. S'il n'est pas possible de faire tenir un gâteau valide dans la boîte, affichez 0.

### Entrée

La première ligne de l'entrée contient deux entiers $R$ et $C$, correspondant respectivement au nombre de lignes et de colonnes de la boîte. Chacune des $R$ lignes suivantes contient $C$ entiers séparés par des espaces. Le $j$-ième entier de la $i$-ième ligne $H_{ij}$, correspond au nombre maximal d'étages du gâteau à la position $(i, j)$ dans la grille.

### Sortie

Votre programme doit écrire un unique entier sur la sortie : le volume maximal d'un gâteau pyramidal qui tient dans la boîte. Comme ce nombre peut être assez grand, il est possible que vous ayez à utiliser des types d'entiers de 64 bits (comme `long long` en C/C++) dans votre programme.

### Example d'entrée 1

```
4 4
1 1 1 0
1 1 0 1
1 1 1 0
1 0 0 1
```

### Example d'entrée 2

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

### Example de sortie 1

```
6
```

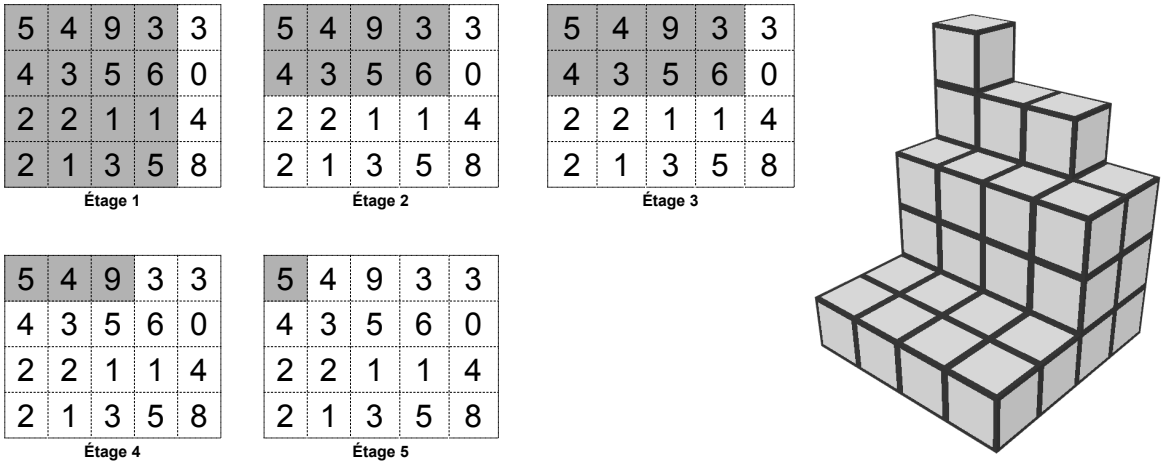
### Example de sortie 2

```
36
```

## Explication

Dans le premier exemple, vous pouvez commander un gâteau à un seul étage, commençant en $(1, 1)$ et finissant en $(3, 2)$. Cela donne un volume de 6, et c'est le volume maximal parmi tous les gâteaux possibles.

Pour le deuxième exemple, vous pouvez commander le gâteau décrit ci-dessous :




Remarquez que la case de coordonnées $(1, 1)$ est toujours couverte par chacun de ces étages. Le volume de ce gâteau est de $16 + 8 + 8 + 3 + 1 = 36$, et c'est le volume maximal atteignable.

## Sous-tâches & Contraintes

Pour toutes les sous-tâches, $1 \leq R, C \leq 1\,000$, et $0 \leq H_{ij} \leq 1\,000\,000$ pour tout $1 \leq i \leq R$ et $1 \leq j \leq C$.

- Pour la sous-tâche 1 (20 points), $H_{ij} \leq 1$ pour tous $i$ et $j$.
- Pour la sous-tâche 2 (25 points), $H_{ij} \leq 25$ pour tous $i$ et $j$.
- Pour la sous-tâche 3 (25 points), $R, C \leq 80$.
- Pour la sous-tâche 4 (30 points), aucune contrainte supplémentaire ne s'applique.

# Problème 3
# Livraison Optimale

**Limites de temps et de mémoire :** 4 seconds, 256 MB

Vous venez d'emménager dans une nouvelle ville, et vous avez trouvé un travail de coursier chez la compagnie Livraison optimale. Cette ville est une grille de $R$ lignes par $C$ colonnes, où les cellules sont numérotées de $(0, 0)$ (coin nord-ouest) à $(R-1, C-1)$ (coin sud-ouest). Il n'est possible de passer d'une cellule à une autre que si elles sont adjacentes horizontalement ou verticalement.

Un de vos collègues, qui exerce ce boulot depuis un certain temps, vous a dit que tous les temps de parcours entre 2 cellules ont été choisis aléatoirement selon une loi uniforme entre 0 et 1 minute à la construction de la ville, et ont été conservés depuis pour d'obscures raisons fiscales.

Pour éviter de ternir la réputation de votre employeur, vous devez au préalable suivre un programme d'entraînement obligatoire. Vous pourrez proposer jusqu'à $Q$ trajets les uns après les autres, chacun entre deux cellules de votre choix, pour lesquels on vous donnera **la différence** entre le temps de votre trajet et le temps optimal entre ces deux cellules.

Une fois votre entraînement fini, vous devrez effectuer $M$ livraisons. Pour chacune d'entre elles, on vous donnera la cellule où vous récupérerez le colis, et celle où vous devrez le livrer. Vous pouvez considérer que pour chaque livraison, la cellule de départ est choisie uniformément au hasard parmi les $R \cdot C$ cellules, et la cellule d'arrivée est choisie uniformément au hasard parmi les $R \cdot C - 1$ cellules restantes. Vous devez proposer un trajet pour livrer le colis le plus rapidement possible. Les points que vous obtiendrez dépendront de la différence entre le temps de votre trajet et le temps de trajet optimal entre ces deux cellules.

Attention cependant, Livraison optimale est intransigeante avec ses nouvelles recrues, et vous licenciera immédiatement si vous proposez un trajet qui passe plus d'une fois par la même cellule, ou passe en dehors de la grille, que ce soit pendant l'entraînement ou une livraison.

## Entrée / Sortie

Cet exercice n'a pas de fichiers d'entrée ou de sortie. À la place, votre programme doit interagir avec les fonctions contenues dans le header `"courier.h"`. Les fonctions fournies sont décrites en détails dans le paragraphe suivant.

**N'affichez *rien* sur la sortie standard, ou vous recevrez 0 sur ce test**

## Module

Votre programme doit interagir avec les fonctions de `"courier.h"` comme suit :

- *N'implémentez pas* de fonction `main` : elle est fournie par le module. À la place, implémentez la fonction
  `void init(int R, int C, int M, int Q)`
  Cette fonction sera appelée *exactement une fois* au lancement de votre programme. Depuis cette fonction, vous pouvez appeler jusqu'à $Q$ fois la fonction
  `double query(int rStart, int cStart, int length, const char* path)`
  où :
  — `rStart` et `cStart` sont la ligne et la colonne où commence le trajet que vous proposez
  — `length` est le nombre d'étapes du trajet (nombre de fois où vous passez d'une case à l'autre), et
  — `path` est une chaîne de caractères composée uniquement des caractères `N`, `E`, `S`, et `W`, qui représentent respectivement un mouvement vers la case située en haut, à droite, en bas et à gauche.

La fonction `query` renvoie la différence, en minutes, entre le trajet proposé et le trajet optimal. Chaque appel à la fonction `query` prend un temps proportionnel à `length` [1].

- Vous devez également implémenter la fonction
  `void solve(int rStart, int cStart, int rEnd, int cEnd, int* length, char* path)`
  Après l'appel à votre fonction init(), cette fonction sera appelée $M$ fois, une fois pour chaque livraison que vous devez effectuer.
  — `rStart` et `cStart` sont la ligne et la colonne de la cellule où vous devez récuperer le colis,
  — et `rEnd` et `cEnd` sont la ligne et la colonne où vous devez le livrer.
  **Remarque :** les cellules (`rStart`, `cStart`) et (`rEnd`, `cEnd`) seront distinctes.
  **Note :** Pour chaque appel de la fonction `solve`, il est garanti que (`rStart`, `cStart`) est choisie uniformément au hasard parmi les $R \cdot C$ cellules, et que (`rEnd`, `cEnd`) est choisie uniformément au hasard parmi les $R \cdot C - 1$ cellules restantes.

  Vous devez renvoyer un chemin valide entre ces deux points en fixant `*length` $= L$, où $L$ est la longueur du trajet que vous proposez, ainsi que remplir le tableau `path` avec exactement $L$ caractères représentant les mouvements requis entre les cellules pour arriver à destination, dans le même format que décrit ci-dessus. $L$ ne doit *pas* dépasser $RC - 1$.
  Votre solution recevra un certain nombre de points, selon à quel point les trajets que vous proposez sont plus long que les trajets les plus rapides. Voir la section *Scores* plus bas pour de plus amples détails.

## Expérimentation

Afin de pouvoir tester votre code sur votre propre machine, commencez par **télécharger** les fichiers `courier.h` et `courier.c` fournis, et placez-les **dans le même dossier que le fichier de votre code**, dans l'onglet *Enoncé* du problème *Delivery*, et ajoutez `#include "courier.h"` en haut de votre code. Un exemple de solution est également disponible pour vous aider. Compilez votre solution en C avec la commande

```
gcc -O2 -Wall -static votrecode.c courier.c -o courier -lm
```

ou bien, si vous utilisez C++,

```
g++ -std=c++11 -O2 -Wall -static yourcode.cpp courier.c -o courier
```

où `votrecode.c/cpp` est le nom du fichier qui contient votre code. Notez que vous devrez toujours utiliser `courier.c` même si votre solution est en C++. L'exécutable compilé `courier` lira 4 entiers séparés par des espaces `R C M Q` sur l'entrée standard, suivis par $2R - 1$ lignes décrivant les temps de trajet, où les lignes avec des numéros pairs contiennent $C - 1$ nombres décimaux (séparés par des espaces), compris entre 0 et 1 (inclus), et les lignes avec des numéros impairs contiennent $C$ nombres décimaux (séparés par des espaces), compris entre 0 et 1 (inclus). Ces lignes doivent être suivies de $M$ lignes de la forme $R_{D_i}\ C_{D_i}\ R_{F_i}\ C_{F_i}$, où $(R_{D_i}, C_{D_i})$ est la cellule de départ et $(R_{F_i}, C_{F_i})$ est la cellule finale pour le $i$-ième appel que le module fera à la fonction `solve`. Un tel fichier d'entrée ressemblera à quelque chose comme ceci :

1. Cela est vrai pour le module utilisé pour tester votre programme sur le serveur de l'épreuve, mais la fonction `query` du fichier `courier.h` qui vous est mis à disposition à des fins de tests peut être plus lente.

$$
\begin{array}{cccccccccc}
R & C & M & Q & & & & & & \\
 & h_{0,0} & & h_{0,1} & & h_{0,2} & \cdots & h_{0,C-2} & & \\
v_{0,0} & & v_{0,1} & & v_{0,2} & & \cdots & & v_{0,C-1} & \\
 & h_{1,0} & & h_{1,1} & & h_{1,2} & \cdots & h_{1,C-2} & & \\
v_{1,0} & & v_{1,1} & & v_{1,2} & & \cdots & & v_{1,C-1} & \\
 & & & & \vdots & & & & & \\
v_{R-2,0} & & v_{R-2,1} & & v_{R-2,2} & & \cdots & & v_{R-2,C-1} & \\
 & h_{R-1,0} & & h_{R-1,1} & & h_{R-1,2} & \cdots & h_{R-1,C-2} & & \\
R_{D_1} & C_{D_1} & R_{F_1} & C_{F_1} & & & & & & \\
 & & & \vdots & & & & & & \\
R_{D_M} & C_{D_M} & R_{F_M} & C_{F_M} & & & & & &
\end{array}
$$

où $h_{i,j}$ est le temps de trajet entre la cellule $(i, j)$ et la cellule $(i, j+1)$, et $v_{i,j}$ est le temps de trajet entre la cellule $(i, j)$ et la cellule $(i+1, j)$.

**Note :** l'entrée telle qu'illustrée ici contient des espaces supplémentaires pour la clarté. Ceci n'est pas requis par le module, mais d'éventuels espaces supplémentaires seront ignorés en toute sécurité.

L'exécutable affichera $M$ lignes sur la sortie standard, où la $i$-ième ligne contient un nombre décimal $s_i$, le score que votre programme reçoit pour la requête $i$. Ces lignes seront suivies d'une ligne contenant `Total score :` S, où $S$ est le score total pour l'ensemble des requêtes de ce fichier test. Voir la section *Scores* pour plus de détails.

## Exemple d'entrée

```
3 4 1 2
    0.5     1.0     0.9
0.1     0.6     0.7     0.4
    0.0     1.0     0.3
0.3     0.2     0.4     0.6
    0.0     1.0     0.5

2 1 0 2
```

## Exemple de session

1. Le module appelle `init(3, 4, 1, 2)`.
2. Vous appelez `query(0, 0, 3, "EEE")`. La fonction retourne 0.6 puisque le trajet proposé prend 2.4 minutes, tandis que le plus court trajet possible `"SEEEN"` prend 1.8 minutes.
3. Vous appelez `query(1, 0, 2, "ES")`. Ce chemin prend 0.2 minutes et est le trajet le plus court, donc la fonction renvoie 0.
4. Votre fonction `init` se termine.
5. Le module appelle `solve(2, 1, 0, 2, &length, path)`, comme spécifié dans l'entrée, vous demandant de trouver un trajet le plus court possible entre $(2, 1)$ et $(0, 2)$.

   Vous fixez `*length = 3` et `path[0] = 'N', path[1] = 'E', path[2] = 'N'`. Le module affiche `33` sur la sortie standard, le score pour cette requête.
6. Le module affiche `Total score: 33` sur la sortie standard, le score total pour ce fichier test.

## Sous-tâches & Contraintes

Pour tous les appels à `solve`, $0 \leq$ `rStart`, `rEnd` $< R$ and $0 \leq$ `cStart`, `cEnd` $< C$, et `length` et `path` seront toujours des pointeurs différents de `NULL`. Veuillez également noter que pour tous les tests au sein d'une sous-tâche, les valeurs de $R, C, M$ et $Q$ sont fixées, comme spécifié ci-dessous :

- Pour la sous-tâche 1 (30 points), $R = 2$, $C = 10$, $M = 1000$, $Q = 5000$.
- Pour la sous-tâche 2 (15 points), $R = C = 10$, $M = 1000$, $Q = 25000$.
- Pour la sous-tâche 3 (15 points), $R = C = 50$, $M = 1000$, $Q = 25000$.
- Pour la sous-tâche 4 (20 points), $R = C = 50$, $M = 1000$, $Q = 4999$, et tous les temps de trajet sont soit 0, soit 1.
- Pour la sous-tâche 5 (20 points), $R = C = 50$, $M = 1000$, $Q = 5000$.

## Scores

Votre score pour ce problème sera le score **maximum** parmi toutes vos soumissions pour ce problème. Pour chaque soumission, votre score pour chaque sous-tâche sera le score **minimum** parmi tous les tests de cette sous-tâche. Votre score pour chaque test sera

$$\max\left(0, \frac{\max\left(t_M, t_R + 10^{-6}\right) - t}{\max\left(t_M, t_R + 10^{-6}\right) - t_R} \times 100\right)$$

où $t$, $t_R$, et $t_M$ sont les sommes de tous les $t_i$, $t_{Ri}$, et $t_{Mi}$, respectivement. Pour le $i$-ième appel à `solve` :

- $t_i$ est le temps pris par le trajet que vous avez proposé,
- $t_{Ri}$ est le temps pris par le plus court trajet possible, et
- $t_{Mi}$ est la *moyenne* des temps pris par les deux trajets suivants : celui qui commence d'abord par atteindre la ligne de la destination en se déplaçant uniquement dans les directions nord-sud, puis atteint la colonne de la destination en se déplaçant uniquement dans les directions est-ouest ; ainsi que celui qui commence d'abord par atteindre la colonne de la destination en se déplaçant uniquement dans les directions est-ouest, puis atteint la ligne de la destination en se déplaçant uniquement dans les directions nord-sud.

Si vous proposez un trajet qui aurait pour conséquence votre licenciement, vous recevrez 0 point pour ce test.

**Si vous affichez quoi que ce soit sur la sortie standard `stdout`, vous recevrez 0 point pour ce test.**