Télégrenouilles (Telefrogs)

Fichier d'entrée	Fichier de sortie	Limite de temps	Limite de mémoire
entrée standard	sortie standard	1 seconde	256 MiB

Opale est une scientifique qui étudie les mouvement d'une certaine espèce d'amphibiens supernaturels, les *télégrenouilles*. Une télégrenouille ressemble à une grenouille normale, mais au lieu de sauter, elle se *téléporte*[1].

Une colonie de K télégrenouilles vivent dans un étang qu'Opale observe depuis maintenant D jours. L'étang contient N nénuphars numérotés de 1 à N sur lesquels les télégrenouilles peuvent s'asseoir. Avant qu'Opale commence ne son étude, toutes les grenouilles étaient assises sur le nénuphar 1.

- Au début de chaque jour, chaque télégrenouille *peut* décider de se téléporter vers un autre nénuphar.
- À la fin de chaque jour, Opale enregistre le nombre de grenouilles sur chaque nénuphar. Plus particulièrement, il y a exactement c_{ij} grenouilles sur le j-ème nénuphar le i-ème jour.

Aucune grenouille supplémentaire ne rejoint la colonie et aucune grenouille ne disparaît lors de l'étude.

À la fin de son étude, Opale réalise que certaines des K grenouilles sont peut-être des *grenouillesimposteur*, qui ne peuvent pas se téléporter ! Elle a découvert $N - 1$ tunnels cachés, unidirectionels, entre certaines paires de nénuphars. Le i-ème tunnel permet aux imposteurs sur le nénuphar a_i de se rendre au nénuphar b_i ($a_i < b_i$). Il est possible de se rendre du nénuphar 1 à n'importe quel autre nénuphar à l'aide d'une séquence de tunnels.

Chaque nuit, les imposteurs, qui ne peuvent pas se téléporter, peuvent se déplacer d'un nénuphar à un autre en empruntant une série de tunnels.

Aidez Opale à déterminer le nombre *maximal* d'imposteurs qui peuvent se trouver parmi les télégrenouilles.

Sous-tâches et Contraintes

Dans chaque sous-tâche, il est garanti que :

- $2 \leq N \leq 1000$, $1 \leq K \leq 10^9$ et $2 \leq D \leq 200$
- $0 \leq c_{ij} \leq K$, pour tous i and j.
- $c_{i1} + c_{i2} + \ldots + c_{iN} = K$, pour tout i. C'est à dire, le nombre de grenouilles observé chaque jour est égal à K.
- $1 \leq a_i < b_i \leq N$, pour tout i.
- Il est possible de voyager du nénuphar 1 à n'importe quel autre nénuphar en empruntant une séquence de tunnels.

Les contraintes supplémentaires pour chaque sous-tâche sont données ci-dessous.

Sous-tâche	Points	Contraintes supplémentaires
1	14	$D = 2$ et $a_i + 1 = b_i$, pout tout i.
2	26	$a_i + 1 = b_i$, pour tout i.
3	16	$D = 2$
4	13	$a_i = 1$, pour tout i.
5	31	Aucune contrainte supplémentaire.

[1]Personne ne sait comment les grenouilles se téléportent. Ça a l'air étrange

Entrée

- La première ligne de l'entrée contient les entiers N, K et D.
- Les $N-1$ lignes suivantes décrivent les tunnels unidirectionnels. La i-ème ligne contient a_i et b_i.
- Les D lignes suivantes contiennent N entiers chacune. Le j-ème entier de la i-ème ligne est c_{ij}.

Sortie

Affichez un seul entier, le nombre maximal d'imposteur qui peuvent se trouver parmi les télé-grenouilles.

Entrée d'exemple 1

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

Sortie d'exemple 1

```
2
```

Entrée d'exemple 2

```
4 3 2
2 3
3 4
1 2
0 0 2 1
3 0 0 0
```

Sortie d'exemple 2

```
0
```

Explications

Dans l'exemple 1, il peut y avoir deux imposteurs :

- Le premier imposteur se déplace au nénuphar 2 le premier jour, ne fait rien le second jour, et ne fait rien le troisième jour.
- Le deuxième imposteur ne fait rien le premier jour, ne fait rien le deuxième jour, et se déplace au nénuphar 6 en passant par le nénuphar 3 le troisième jour.
- La première télégrenouille ne fait rien le premier jour, se téléporte au nénuphar 2 le deuxième jour, et se téléporte au nénuphar 1 le troisième jour.
- La deuxième télégrenouille se téléporte au nénuphar 6 le premier jour, se téléporte au nénuphar 2 le deuxième jour, et se téléporte au nénuphar 4 le troisième jour.

Il est possible de montrer qu'il est impossible d'avoir plus que deux imposteurs.

Dans l'exemple 2, aucune grenouille ne peut être imposteur.

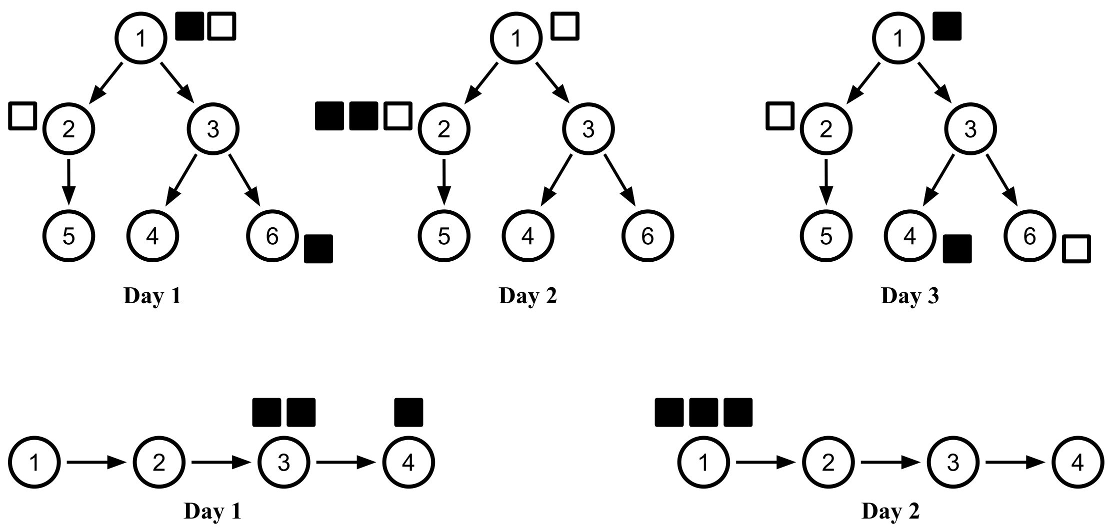


Figure 1: Dans chaque case, les carrés noirs représentent les télégrenouilles et les carré blancs représentent les imposteurs.

Rivière (River II)

Fichier d'entrée	Fichier de sortie	Limite de temps	Limite de mémoire
entrée standard	sortie standard	1.5 secondes	256 MiB

Les N habitants de Ragden habitent dans les caves souterraines rectangulaires creusées sous l'extravagant palace royal du Grand Arbre.

Fatigués par inondations répétitives causées par des événements comme la *Grande Tempête* ou *Lauren a Oublié d'Éteindre l'Arrosage Automatique*, les habitants ont demandé à ce que soit construit une rivière souterraine par laquelle l'eau peut s'écouler.

Le Souterrain peut être décrit par un rectangle de W mètres de large, et H mètres de haut. Le point situé à x mètres du bord gauche du Souterrain et y mètres sous la surface est noté (x, y).

La i-ème cave est définie par le rectangle avec un coin haut gauche (a_i, b_i) et un coin bas-droite (c_i, d_i). **Deux caves ne s'intersectent jamais**. Les caves peuvent se toucher au niveau d'un côté ou d'un coin.

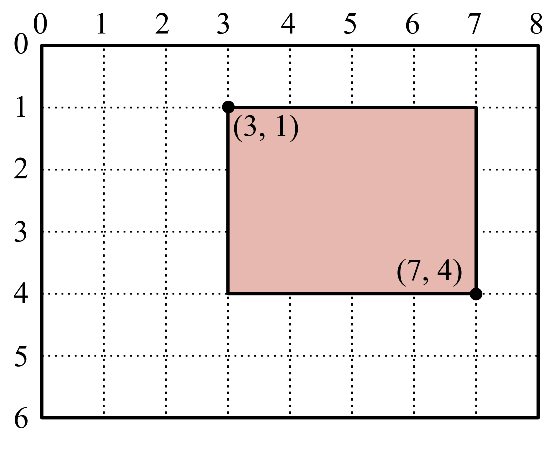


Figure 1: Un exemple avec $W = 8$, $H = 6$

La rivière peut être représentée par une séquence de points (x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_k, y_k), qui forme une ligne brisée.

- La rivière doit commence à la surface, c'est à dire $y_0 = 0$.
- La rivière doit se terminer en bas du souterrain, c'est à dire $y_k = H$.
- La rivière ne doit jamais remonter, c'est à dire $y_i \leq y_{i+1}$ pour tout i.
- La rivière ne doit pas passer par l'intérieur d'une cave. La rivière peut toucher les côtés ou les coins des caves.

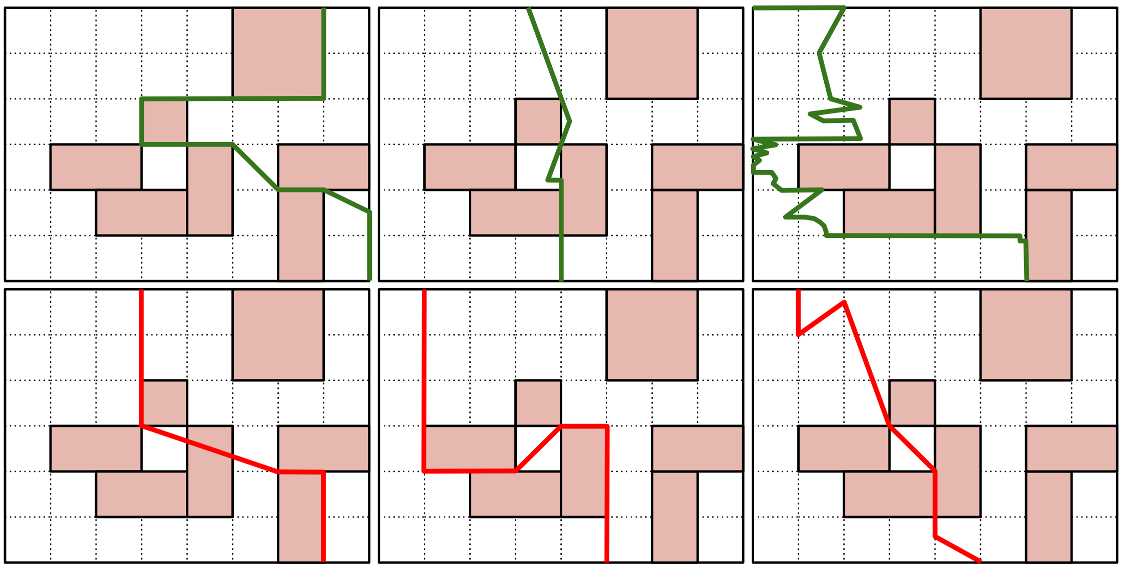


Figure 2: Les trois rivières du haut sont valides. Les trois rivières du bas sont invalides.

Observez que la rivière sépare le Sousterrain en un *côté gauche* et un *côté droit*. Si la i-ème cave est sur le côté gauche, les habitants généreront l_i points de bonheur. De même, si la i-ème cave est sur le côté gauche, les habitants généreront r_i points de bonheur. Notez que l_i et r_i peuvent être négatifs.

Quelle est la quantité maximale de bonheur atteignable ?

Sous-tâches et Contraintes

Dans chaque sous-tâche, il est garanti que :

- $1 \leq N \leq 100\,000$.
- $1 \leq W, H \leq 1\,000\,000$.
- $0 \leq a_i < c_i \leq W$, pour tout i.
- $0 \leq b_i < d_i \leq H$, pout tout i.
- $-10\,000 \leq l_i, r_i \leq 10\,000$, pour tout i.
- Deux cave ne s'intersectent jamais.

Les contraintes supplémentaires pour chaque sous-tâche sont données ci-dessous.

Sous-tâche	Points	Contraintes supplémentaires
1	6	Toutes les caves ont une hauteur de 1, c'est à dire $b_i + 1 = d_i$ pour tout i.
2	23	$W, H, N \leq 100$.
3	14	$W, H \leq 1000$.
4	25	$N \leq 5000$.
5	28	Toutes les caves ont une largeur de 1, c'est à dire $a_i + 1 = c_i$ pour tout i.
6	4	Aucune contrainte supplémentaire.

Entrée

- La première ligne de l'entrée contient les trois entiers N, W et H.
- Les N lignes suivantes décrivent les caves. La i-ème ligne contient a_i, b_i, c_i, d_i, l_i et r_i.

Sortie

Affichez un seul entier, la quantité maximale de bonheur atteignable.

Entrée d'exemple

```
7 8 6
5 0 7 2 -30 -9
3 2 4 3 1 9
1 3 3 4 -3 -8
2 4 4 5 -10 10
4 3 5 5 0 2
6 3 8 4 40 6
6 4 7 6 1 3
```

Sortie d'exemple

```
30
```

Explication

En dessinant la rivière comme ci-dessous, il est possible d'atteindre une quantité de bonheur de $-9 + 9 + -3 + -10 + 0 + 40 + 3 = 30$.

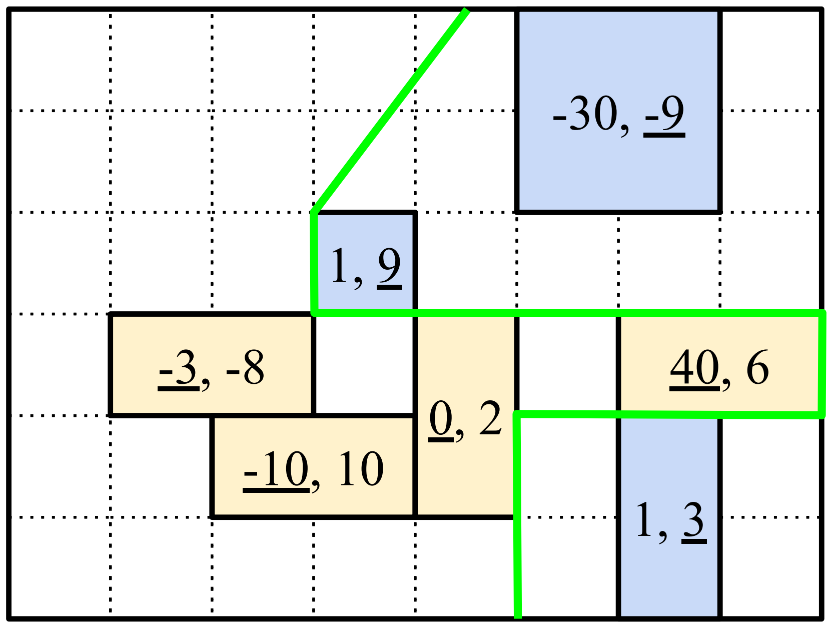


Figure 3: Représentation de l'exemple. Les caves sur la gauche sont dessinées en jaune, les caves sur la droite sont dessinées en bleu.

Symboles Chanceux (Lucky Symbols)

Fichier d'entrée	Fichier de sortie	Limite de temps	Limite de mémoire
N/A	N/A	2 secondes	512 MiB

Symboles Chanceux est un jeu qui se joue sur une grille de R par C cases. Les lignes sont numérotées de 0 à $R-1$ de haut en bas, et les colonnes sont numérotées de 0 à $C-1$ de gauche à droite.

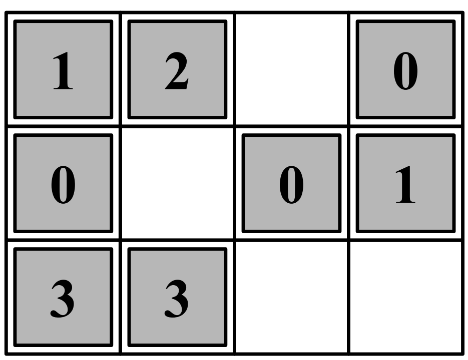
Chaque jeu se joue en 100 *parties* (*rounds*). Le but lors de chaque partie est de placer des symboles sur la grille afin de maximiser le score à la fin de la partie.

Il y a N types de symboles, numérotés de 0 à $N-1$. Le i-ème symbole a une valeur a_i.

Chaque partie se déroule en 1000 *tours* (*turns*). Au début de chaque tour, on vous donne un symbole d'un des N types, choisi uniformément aléatoirement (chaque symbole apparaît avec la même probabilité). Vous devez ensuite choisir entre :

- *Placer* le symbole sur une des cases vides de la grille.
- *Jeter* le symbole sans le placer sur la grille.

À la fin de chaque tour, vous recevez un nombre de points égal à la somme des valeurs des symboles sur la grille. Par exemple, si vous terminez un tour avec la grille ci-dessous, vous recevrez $50 + 10 + 20 + 20 + 20 + 50 + 100 + 100 = 370$ points pour ce tour.



Tile	Value
0	20
1	50
2	10
3	100

Bonus Pair	Points
0, 1	90
0, 2	70
0, 3	30
3, 3	100

Figure 1: Un exemple de grille avec $R = 3$, $C = 4$ et $N = 4$

Après 1000 tours la partie prend fin et vous gagnez des points bonus ! Il y a E paires de symboles bonus. La i-ème paire bonus donne b_i points pour chaque paire de cases adjacentes (qui se touchent par un côté), telle qu'une des case contient le symbole x_i et l'autre contient le symbole y_i.

Par exemple, si vous terminez une partie avec la grille ci-dessus, vous recevrez 400 points bonus pour cette partie

- Trois paires bonus $\{0, 1\}$: $3 \times 90 = 270$
- Aucune paire bonus $\{0, 2\}$.
- Une paire bonus $\{0, 3\}$: $1 \times 30 = 30$.
- Une paire bonus $\{3, 3\}$: $1 \times 100 = 100$.

Après avoir calculé votre score, les symboles sont retirés de la grille et la partie suivante commence. Écrivez un programme qui maximise votre score moyen sur 100 parties.

Sous-tâches et Contraintes

Dans chaque sous-tâche, il est garantit que :

- $0 \le a_i$, pour tout i.
- $1 \le b_i$, pour tout i.
- $0 \le x_i \le y_i < N$, pour tout i. Notez qu'il est possible que $x_i = y_i$.
- Aucune paire de symboles apparaît plus d'une fois en tant que paire bonus. C'est à dire, soit $x_i \neq x_j$ soit $y_i \neq y_j$ pour tous i et j avec $i \neq j$.

Sous-tâche	Points	R	C	N	E	Max a_i	Max b_i	Contraintes supplémentaires
1	8	1	1	1000	0	1000	N/A	
2	8	3	3	1000	0	1000	N/A	
3	18	50	50	10	10	0	1	$x_i = y_i$ pour tout i
4	18	15	15	2000	2000	0	1	$x_i = y_i$ pour tout i
5	16	20	20	10	50	10	2000	
6	16	20	20	100	500	10	2000	
7	16	20	20	1000	5000	10	2000	

Dans ce problème, chaque sous-tâche a **un seul fichier test**. Ces fichiers tests sont téléchargeables depuis la page "Attachements". Notez que dans chaque fichier, les valeurs de a_i et b_i ont été générées uniformément aléatoirement. De même, les E paires bonus ont été générées de telle manière que chaque paire a la même probabilité d'être choisie.

On vous rappelle que votre score sur ce problème est la somme des scores sur les différentes sous-tâches. Votre score sur une sous-tâche est le score maximum obtenu par l'une de vos soumissions. Par conséquent, il peut être intéressant de résoudre les différentes sous-tâches avec des soumissions différentes.

Score

Si votre programme ne joue pas correctement au jeu décrit dans la section *Implémentation*, vous recevrez O% des points de la sous-tâche.

Sinon, votre score sera calculé à l'aide d'une fonction linéaire entre deux scores limites O_{min} et O_{max} ($O_{min} < O_{max}$). Votre score passe linéairement de 0% à 100% entre O_{min} et O_{max}. Plus précisément, si S est le nombre moyen de points obtenus par votre programme, alors :

- Si $S > O_{max}$, vous marquez 100% des points.
- Si $O_{min} \le S \le O_{max}$, vous marquez $\lfloor (S - O_{min})/(O_{max} - O_{min}) \times 100 \rfloor$% des points.
- Si $S < O_{min}$, vous ne marquez aucun point.

Les paramètres O_{min} et O_{max} pour chaque sous-tâche sont donnés dans le tableau ci-dessous. O_{max} est le meilleur score parmi les solutions des juges sur la sous-tâche.

Sous-tâche	O_{min}	O_{max}
1	450000	953000
2	4500000	8120000
3	3000	3590
4	0	104
5	2500000	3620000
6	1500000	3260000
7	1500000	2580000

Implémentation

Entrée / Sortie

Dans ce sujet, vous ne devez ni lire sur l'entrée standard ni écrire sur la sortie standard. À la place, votre solution doit interagir avec les fonctions définies dans le fichier `lucky.h`.

Ne *rien* écrire sur `stdout`, ou vous obtiendrez 0% **des points sur le fichier test.**

Fonctions

Vous **ne devez pas** implémenter une fontion `main`. À la place vous devez inclure `#include "lucky.h"` et implémenter les fonctions `init`, `newRound` et `playTurn` décrites ci-dessous :

```
void init(int R, int C, int N, int E,
    std::vector<int> a, std::vector<int> b, std::vector<int> x, std::vector<int> y)
```

avec:

- `R`, `C`, `N` et `E` correspondent aux variables définies ci-dessus.
- Pour tout $0 \leq i < N$, `a[i] =` a_i.
- Pour tout $0 \leq j < E$, `b[j] =` b_j, `x[j] =` x_j, et `y[j] =` y_j.
- Cette fonction est appelée en premier pour initialiser votre programme et commencer le jeu.

```
void newRound()
```

Cette fonction est appelée au début de chaque partie. Elle sera appelée 100 fois, une pour chaque partie du jeu.

```
void playTurn(int symbol)
```

Après chaque appel à `newRound`, cette fonction sera appelée 1000 fois, une fois pour chaque tour de la partie. `symbol` est le type du symbole qui vous est donné à ce tour.

Votre implémentation de `playTurn` doit appeler `place` ou `discard` comme décrit ci-dessous :

- `void place(int r, int c);` – appelez cette fonction pour placer un symbole dans la case située sur la `r`-ème ligne et la `c`-ème colone.
- `void discard();` – appelez cette fonction pour jeter le symbole.

L'évaluateur (grader) choisira chaque tour le symbol qui vous est donné, uniformément aléatoirement. L'évaluateur n'adapte pas les symboles à vos choix de placer ou jeter les symboles. La séquence de symboles choisie dans chaque fichier test est fixée (c'est à dire, les symboles donnés à la fonction `playTurn` seront les même lors de chaque soumission sur ce fichier test).

Conditions d'échec

Votre programme :

- Ne doit pas appeler `place` ou `discard` depuis `init` ou `newRound`.
- Doit appeler soit `place` soit `discard`, exactement une fois lors de chaque appel à `playTurn`.
- En appelant `place(r, c)`, les paramètres doivent vérifier $0 \leq$ `r` $< R$ et $0 \leq$ `c` $< C$
- En appelant `place(r, c)`, la case sur la `r`-ème ligne et la `c`-ème colonne ne doit pas encore contenir un symbole.

Si votre programme ne respecte pas une de ces conditions, il sera jugé `incorrect` et obtiendra un score de 0% sur le fichier test.

Expérimentation

Afin de pouvoir tester votre code sur votre propre machine, vous devez télécharger les fichiers `lucky.cpp`, `lucky.h` et `grader.cpp`, et les placer dans le même dossier que votre code. Notez que l'évaluateur officiel peut avoir un comportement différent de l'évaluateur d'exemple qui vous est fourni. Vous devez modifier `lucky.cpp`, qui contient des implémentation par défaut des fonctions `init`, `newRound` et `playTurn`.

Compilez votre programme avec :

```
g++ -std=c++11 -O2 -Wall lucky.cpp grader.cpp -o lucky
```

Cette commande créera un exécutable `lucky`, que vous pourrez lancer avec `./lucky`. Si vous avez des problème lors de la compilation, envoyez un message dans la section Communication de la plateforme.

L'évaluateur compilé lit l'entrée sur l'entrée standard dans le format suivant :

- La première ligne contient les quatres entiers R, C, N et E.
- La deuxième ligne contient N entiers. Le i-ème entier est a_i.
- Les E lignes suivantes décrivent les paires bonus. La i-ème de ces lignes contient b_i, x_i and y_i.

Pour chaque fichier test, l'évaluateur d'exemple jouera 100 parties. Chaque partie commence par un appel à la fonction `newRound`, et est suivie par 1000 appels à `playTurn`.

À la fin du jeu, l'évaluateur d'exemple affiche le nombre moyen de points obtenus sur la sortie standard.

Notez que l'évaluateur d'exemple *peut ne pas* être aussi stricte que l'évaluateur utilisé pour évaluer vos solutions. En particulier le l'évaluateur d'exemple peut ne pas vérifier toutes les conditions d'échec énoncées ci-dessus.

Entrée de l'évaluateur d'exemple

```
3 4 4 4
20 50 10 100
90 0 1
70 0 2
30 0 3
100 3 3
```

Une interaction possible est détaillée ci-dessous :

Évaluateur	Participant	Description
`init(3, 4, 4, 4, [20, 50, 10, 100], [90, 70, 30, 100], [0, 0, 0, 3], [1, 2, 3, 3])`		L'évaluateur initialise votre programme. program.
`newRound()`		L'évaluateur commence une nouvelle partie.
`playTurn(2)`		Vous recevez un symbole de type 2.
	`place(0, 1)`	Vous le placez en ligne 0, colonne 1.
`playTurn(3)`		Vous recevez un symbole de type 3.
	`discard()`	Vous le jetez.
`playTurn(0)`		Vous recevez un symbole de type 0.
	`place(0, 2)`	Vous le placez en ligne 0, colonne 2.
`playTurn(2)`		Vous recevez un symbole de type 2.
	`place(2, 3)`	Vous le placez en ligne 2, colonne 3.
Fin de la partie		Votre score pour cette partie est de 200 points.
`newRound()`		L'évaluateur commence une nouvelle partie.
`playTurn(2)`		Vous recevez un symbole de type 2.
	`discard()`	Vous le jetez.
`playTurn(0)`		Vous recevez un symbole de type 0.
	`discard()`	Vous le jetez.
`playTurn(2)`		Vous recevez un symbole de type 2.
	`discard()`	Vous le jetez.
`playTurn(1)`		Vous recevez un symbole de type 1.
	`discard()`	Vous le jetez.
Fin de la partie		Votre score pour cette partie est de 0 points.
Fin du jeu		L'évaluateur affiche votre score moyen.

D'ordinaire, l'évaluateur jouera 100 parties de 1000 tours chacune. Cependant, par soucis de concision la session décrite consiste en 2 parties, de 4 tour chacune.

Votre score sur la première partie est de 200 points :

- À la fin du premier tour, vous recevez 10 points.
- À la fin du deuxième tour, vous recevez 10 points.
- À la fin du troisième tour, vous recevez 30 points.
- À la fin du quatrième tour, vous recevez 80 points.
- À la fin de la partie, vous recevez 70 points bonus.

Votre score sur la deuxième partie est de 0 point (car vous avez jeté tous les symboles).

Votre score moyen à la fin du jeu est donc de 100 points.

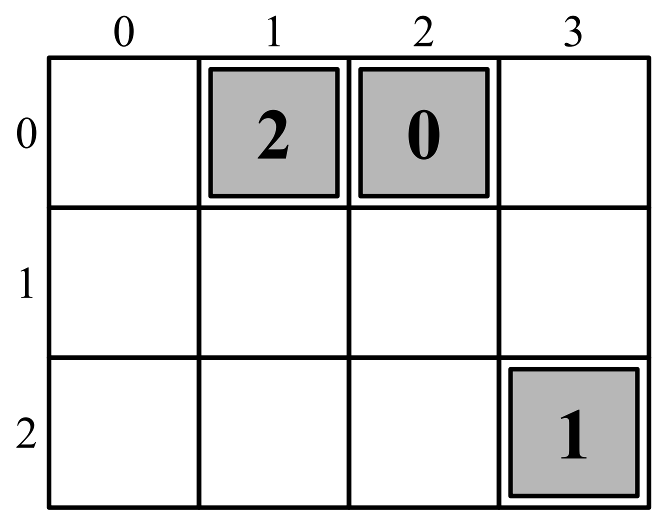


Figure 2: État de la grille à la fin de la première partie donnée en exemple