

Aesthetic Islands

Les organisateurs de PAIO prévoient une excursion au lac Kivu, réputé pour ses nombreuses îles magnifiques. Afin de bien organiser cette sortie, ils ont besoin de connaître le nombre exact d'îles que compte le lac. Heureusement, ils viennent de recevoir une image satellite de la carte du lac.

La carte est représentée sous la forme d'une grille de $N \times M$. Les lignes de la carte sont numérotées de 0 à $N - 1$ de haut en bas, et les colonnes de la carte sont numérotées de 0 à $M - 1$ de gauche à droite. La cellule située à la ligne i et à la colonne j ($0 \leq i < N, 0 \leq j < M$) est désignée par (i, j) . Chaque cellule contient 1 s'il s'agit de terre et 0 s'il s'agit d'eau.

Deux cellules de terre sont considérées comme adjacentes si elles partagent un côté. En d'autres termes, deux cellules de terre sont adjacentes si l'une se trouve directement au-dessus, en dessous, à gauche ou à droite de l'autre. Les cellules qui ne se touchent qu'à un coin ne sont pas adjacentes.

Une île est un ensemble maximal de cellules de terre tel que chaque paire de cellules de l'ensemble puisse être reliée par une suite de cellules de terre adjacentes. Ici, « maximal » signifie qu'aucune autre cellule de terre ne peut être ajoutée à l'ensemble tout en conservant cette propriété.

Par exemple, sur la carte suivante :

1	0	1
0	1	0
1	0	1

Il y a 5 îles, car les contacts en diagonale ne relient pas les cases de terre.

Sur la carte suivante :

1	1	0	1
0	1	0	1
0	0	0	0
1	1	1	0

Il y a 3 îles.

Nous savons également que, dans le lac Kivu, toutes les îles sont « esthétiques ». Une île est dite « esthétique » si, pour toutes les cellules de terre appartenant à l'île et situées sur la même ligne ou la même colonne, chaque cellule entre elles est également de la terre. En d'autres termes, au sein

d'une même île, chaque ligne et chaque colonne occupées par l'île contiennent un bloc continu de cellules de terre.

Par exemple, voici des cartes possibles pour le lac Kivu (c'est-à-dire ne contenant que des îles esthétiques) :

0	0	1	0	0	1	1	1	1	1
0	1	1	1	0	1	0	0	0	0
1	1	1	1	1	0	1	1	1	0
0	1	1	0	0	1	0	1	0	1
0	1	0	0	0	1	1	0	1	1

La deuxième carte comporte 4 îles, toutes aussi pittoresques les unes que les autres.

En revanche, les cartes suivantes ne correspondent pas au lac Kivu :

					1	1	1	0	1
1	1	1			1	0	1	0	1
0	1	0			1	0	1	0	0
1	1	1			1	0	1	0	1
					1	1	1	0	1

Les organisateurs ont accès à un ordinateur spécial doté d'une puce mémoire de K bits, numérotés de 0 à $K - 1$. La carte est déjà stockée dans cette mémoire. Plus précisément, la valeur de la cellule (i, j) est stockée dans le bit $i \cdot M + j$ de la mémoire.

Les organisateurs ne peuvent ni lire ni modifier la mémoire directement ; ils ne savent donc pas à quoi ressemble la carte. En revanche, avec votre aide, ils peuvent soumettre un programme à l'ordinateur.

Votre programme ne connaît pas les valeurs stockées en mémoire. Il ne connaît que N , M et K . Sa tâche consiste à décrire une séquence fixe d'opérations binaires que l'ordinateur effectuera ultérieurement sur la mémoire contenant la carte.

Une fois votre programme terminé, les opérations décrites sont exécutées sur le contenu caché de la mémoire. À la fin, votre programme doit sélectionner certains bits occupés de la mémoire. Ces bits, lus du moins significatif au plus significatif, doivent représenter le nombre d'îles en binaire après l'exécution de votre programme sur la carte cachée stockée en mémoire.

Vous devez implémenter un programme qui décrit les opérations que l'ordinateur doit effectuer sur la mémoire.

Au départ, les bits $0, 1, \dots, N \cdot M - 1$ contiennent la carte. Ces bits sont déjà occupés et ne peuvent pas être écrasés. Tous les bits restants sont initialement vides.

L'ordinateur dispose d'6s types d'instructions pouvant être utilisées dans votre programme pour effectuer des opérations sur les bits de mémoire. Chaque instruction est une opération bit à bit sur

un ou deux bits d'entrée, et le résultat est stocké dans un nouveau bit de mémoire.

Chaque fois qu'une instruction génère un nouveau bit, l'ordinateur le stocke dans le premier bit libre disponible de la mémoire. Ce bit devient alors occupé et peut être utilisé comme bit d'entrée dans des instructions ultérieures. Chaque bit de mémoire ne peut être écrit qu'une seule fois au maximum.

Les instructions disponibles sont décrites ci-dessous. Après avoir reçu chaque instruction, l'ordinateur crée d'abord un nouvel $\text{ind}C$ ($0 \leq C < K$), qui correspond au premier bit libre de la mémoire. Pour un $\text{ind}X$, soit $\text{memory}[X]$ la valeur stockée dans le bit X de la mémoire.

- $\text{and}(A, B)$: effectue l'opération AND bit à bit sur les bits d'indices A et B et stocke le résultat dans le bit C de la mémoire. En d'autres termes, elle définit $\text{memory}[C] := 1$ si $\text{memory}[A]$ et $\text{memory}[B]$ sont tous deux 1, et définit $\text{memory}[C] := 0$ dans le cas contraire. La fonction renvoie C .
- $\text{or}(A, B)$: effectue l'opération OR binaire sur les bits dont les indices sont A et B , puis stocke le résultat dans le bit C de la mémoire. En d'autres termes, elle définit $\text{memory}[C] := 1$ si au moins l'un des bits $\text{memory}[A]$ et $\text{memory}[B]$ est 1, et définit $\text{memory}[C] := 0$ dans le cas contraire. La fonction renvoie C .
- $\text{xor}(A, B)$: effectue l'opération XOR bit à bit sur les bits dont les indices sont A et B , et stocke le résultat dans le bit C de la mémoire. En d'autres termes, elle définit $\text{memory}[C] := 1$ si exactement l'un des bits $\text{memory}[A]$ et $\text{memory}[B]$ vaut 1, et définit $\text{memory}[C] := 0$ dans le cas contraire. La fonction renvoie C .
- $\text{not}(A)$: effectue l'opération NOT bit à bit sur le bit dont l'indice est A , et stocke le résultat dans le bit C de la mémoire. En d'autres termes, elle définit $\text{memory}[C] := 1 - \text{memory}[A]$. La fonction renvoie C .
- $\text{nand}(A, B)$: calcule le NAND binaire des bits d'indices A et B et stocke le résultat dans le bit C de la mémoire. En d'autres termes, elle active $\text{memory}[C] := 0$ si $\text{memory}[A]$ et $\text{memory}[B]$ sont tous deux 1, et active $\text{memory}[C] := 1$ dans le cas contraire. La fonction renvoie C .
- $\text{nor}(A, B)$: calcule le NOR binaire des bits d'indices A et B et stocke le résultat dans le bit C de la mémoire. En d'autres termes, elle définit $\text{memory}[C] := 0$ si au moins l'un des bits $\text{memory}[A]$ et $\text{memory}[B]$ est 1, et définit $\text{memory}[C] := 1$ dans le cas contraire. La fonction renvoie C .

L'ordinateur exécutera les opérations décrites dans l'ordre. À la fin, certains bits de mémoire occupés, choisis par votre programme, doivent contenir la représentation binaire du nombre d'îles présentes sur la carte.

Implementation Details

C++/Python Interface

Pour les soumissions C++/Python, vous devez implémenter la procédure suivante :

```
int32[] construct_instructions(int32 N, int32 M, int32 K)
```

- N : nombre de lignes
- M : nombre de colonnes
- K : nombre de bits dans la puce mémoire
- Cette procédure est appelée exactement une fois et doit construire une séquence d'instructions permettant d'effectuer la tâche requise.
- La procédure doit renvoyer un tableau V contenant les indices mémoire où est stockée la réponse, du bit le moins significatif au bit le plus significatif. En d'autres termes, soit l la longueur de V . Le nombre d'îles sur la carte doit être égal à $2^0 \cdot \text{memory}[V[0]] + 2^1 \cdot \text{memory}[V[1]] + \dots + 2^{l-1} \cdot \text{memory}[V[l-1]]$.
- Chaque index de bit dans V doit faire référence à un bit occupé.
- La longueur de V ne doit pas dépasser 20.

Cette procédure doit appeler les procédures suivantes pour construire la séquence d'instructions :

```
int32 append_and(int32 A, int32 B)
int32 append_or(int32 A, int32 B)
int32 append_xor(int32 A, int32 B)
int32 append_not(int32 A)
int32 append_nand(int32 A, int32 B)
int32 append_nor(int32 A, int32 B)
```

- Chaque procédure ajoute respectivement au programme une instruction « *and* », « *or* », « *xor* », « *not* », « *nand* » ou « *nor* ».
- Pour la procédure « `append_not` », le bit « A » doit être activé. Pour chacune des cinq autres procédures, les bits « A » et « B » doivent tous deux être activés.
- Chaque procédure renvoie l'index mémoire à l'emplacement duquel le bit généré sera stocké.

Vous pouvez également appeler la procédure suivante pour vous aider à tester votre solution :

```
void append_print(string label, int32[] A)
```

- Chaque index transmis dans A à `append_print` doit faire référence à un bit occupé.
- Tout appel à cette procédure sera ignoré lors de l'évaluation de votre solution et ne sera pas comptabilisé comme une instruction.

- Dans l'évaluateur d'exemple, cette procédure ajoute une opération « *print(label, A)* » au programme.
- Lorsque l'évaluateur d'exemple rencontre une opération « *print(label, A)* » pendant l'exécution d'un programme, il affiche « *label* » suivi du contenu des indices de bits de « *A* » sur une seule ligne, dans le même ordre que celui dans lequel ils apparaissent dans « *A* » (voir la section « Évaluateur d'exemple » pour plus de détails).

C Interface

For C submissions, include the provided `islands.h` header and implement:

```
int construct_instructions(int N, int M, int K, int answer_bits[]);
```

The grader supplies `answer_bits`, an array with capacity `ISLANDS_MAX_ANSWER_BITS` (20). Write the indices of V , from least significant bit to most significant bit, into `answer_bits[0]`, `answer_bits[1]`, and so on, and return the number l of indices written. Thus, $V[i] = \text{answer_bits}[i]$ for $0 \leq i < l$. The returned value must satisfy $0 \leq l \leq 20$. Do not write beyond the capacity of `answer_bits` and do not free this grader-owned array.

In C, the exact signature of the debugging procedure is:

```
void append_print(const char *label, const int A[], int length);
```

Here, `label` must point to a null-terminated string, and `length` is the number of elements in `A` and must be nonnegative. `A` may be `NULL` only when `length` is 0.

L'évaluation de votre solution peut générer l'un des messages d'erreur suivants :

- `Invalid index` : un index de bit incorrect (éventuellement négatif) a été fourni comme paramètre A ou B lors d'un appel à l'une des procédures, ou a été inclus dans la réponse générée par `construct_instructions(N, M, K)`.
- `Too many instructions` : il n'y a pas suffisamment de bits de mémoire pour stocker les résultats des instructions. En d'autres termes, vous avez utilisé plus de $K - N \cdot M$ instructions.
- `Too many bits in the answer` : `construct_instructions` a produit plus de 20 bits pour la réponse.

Si le programme ne génère aucun message d'erreur et calcule le nombre correct d'îles, votre programme sera jugé « Accepté » et votre score sera calculé en fonction du nombre d'instructions utilisées (voir Sous-tâches). Dans le cas contraire, il sera jugé « Réponse incorrecte ».

Exemples

Exemple 1

Supposons que $N = 1$, $M = 2$ et $K = 10^7$.

Il n'existe que quatre cas de figure possibles pour les données d'entrée du programme :

- Cas 1 : 0 0
- Cas 2 : 0 1
- Cas 3 : 1 0
- Cas 4 : 1 1

On constate que, dans tous les cas, le nombre d'îles est égal à 1 si la cellule (0,0) ou (0,1) est une cellule de terre. Par conséquent, une solution possible pour $N = 1$ et $M = 2$ consiste à construire un programme à l'aide d'une seule instruction *or* :

- `append_or(0, 1)`, qui calculerait le OU binaire des cellules (0,0) et (0,1) de la carte stockées respectivement dans les bits 0 et 1 de la mémoire, et le stockerait dans le bit 2 de la mémoire.

La séquence de réponse générée par `construct_instructions` est alors [2], puisque le bit 2 de la mémoire contiendrait 1 s'il y a exactement 1 île sur la carte et 0 dans le cas contraire.

Exemple 2

Pour $N = 5$, $M = 5$, $K = 10^7$ et la carte suivante :

1	1	1	1	1
1	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	0	1	1

Le nombre d'îles est 4. Par conséquent, lorsque votre programme fixe est exécuté avec ces valeurs, les bits dont les indices sont générés par `construct_instructions` doivent contenir 0, 0 et 1 dans cet ordre, puisque la représentation binaire de 4 est 100_2 .

Constraints

- $1 \leq N, M \leq 32$
- $K = 10^7$
- Chaque case de la carte est soit 0, soit 1
- Pour toute paire de cases « terre » appartenant à la même île et situées sur la même ligne ou la même colonne, toutes les cases situées entre elles sont également des cases « terre ».

Sous-problèmes

1. (40 points) Chaque île est constituée d'exactly une case « terre ».
2. (20 points) Pour chaque île, il existe des lignes r_1, r_2 et des colonnes c_1, c_2 telles que l'île se compose exactement de toutes les cellules (i, j) avec $r_1 \leq i \leq r_2$ et $c_1 \leq j \leq c_2$. En d'autres termes, chaque île forme un rectangle rempli de cellules terrestres.
3. (40 points) Aucune contrainte supplémentaire.

Supposons que votre programme soit jugé « Accepté » et qu'il utilise les instructions I . Dans ce cas, votre score P pour le cas de test, en fonction du numéro de la sous-tâche, est calculé comme suit :

- Sous-tâche 1 :
 - Si $I \leq 5\,030$, $P = 40$.
 - Si $5\,030 < I \leq 10\,760$, $P = 22 + 18 \cdot \frac{10\,760 - I}{5\,730}$.
 - Si $10\,760 < I \leq 22\,530$, $P = 12 + 10 \cdot \frac{22\,530 - I}{11\,770}$.
 - Sinon, $P = 8$.
- Sous-tâche 2 :
 - Si $I \leq 62\,000$, $P = 20$.
 - Sinon, $P = 8$.
- Sous-tâche 3 :
 - Si $I \leq 9\,100$, $P = 40$.
 - Si $9\,100 < I \leq 15\,880$, $P = 22 + 18 \cdot \frac{15\,880 - I}{6\,780}$.
 - Si $15\,880 < I \leq 27\,650$, $P = 13 + 9 \cdot \frac{27\,650 - I}{11\,770}$.
 - Sinon, $P = 10$.

Sample Grader

Le grader lit les données d'entrée au format suivant :

- ligne 1 : $N \ M \ K$

Cette ligne est suivie de plusieurs cas de test. Chaque cas de test se compose d' N lignes :

- ligne i ($0 \leq i < N$) : $g[i][0] \ g[i][1] \ \dots \ g[i][M-1]$ (séparés par un espace)

où $g[i][j]$ ($0 \leq i < N, 0 \leq j < M$) désigne le contenu de la cellule (i, j) de la table.

La description de tous les cas de test est suivie d'une seule ligne contenant uniquement -1 .

Le correcteur type appelle d'abord `construct_instructions` en utilisant la signature correspondant au langage de soumission. Si cet appel enfreint une contrainte décrite dans

l'énoncé du problème, le correcteur type affiche l'un des messages d'erreur répertoriés à la fin de la section « Détails de l'implémentation » et se termine.

Sinon, le correcteur type traite les cas de test dans l'ordre. Pour chaque cas de test, il exécute le programme construit par `construct_instructions` sur le cas de test d'entrée.

Pour chaque instruction $print(label, A)$, soit l la longueur de A , le correcteur affiche le contenu des indices de bits dans A au format suivant :

- $label : memory[A[0]] \ memory[A[1]] \ \dots \ memory[A[l - 1]]$

Une fois toutes les instructions exécutées, le correcteur d'échantillons affiche le nombre binaire construit à partir de la séquence de réponses V générée par votre programme pour chaque cas de test. En d'autres termes, pour chaque cas de test, soit l la longueur de V , le correcteur affiche $2^0 \cdot memory[V[0]] + 2^1 \cdot memory[V[1]] + \dots + 2^{l-1} \cdot memory[V[l - 1]]$.

Après avoir exécuté tous les cas de test, le correcteur affiche `number of instructions : X` où X est le nombre d'instructions de votre programme.