

Îlots esthétiques

Les organisateurs du PAIO préparent une excursion au Lac Kivu, connu pour ses îlots sublimes. Ils ont besoin, à cette fin, au nombre d'îlots sur le lac. Ils ont alors reçu une image satellite du lac.

Cette carte une grille $N \times M$; les lignes sont numérotés de 0 à $N - 1$ et les colonnes de 0 à $M - 1$. La case en ligne i et colonne j est notée (i, j) . Elle contient le chiffre 1 si c'est une cellule de terre, sinon elle contient le chiffre 0 pour indiquer une cellule de mer. Deux cellules de terre sont adjacent s'ils ont un même coté. Un îlot est un ensemble de cellules de terre. Deux cellules appartiennent au même îlot s'ils sont adjacents.

Par exemple, dans cette carte, il y a 5 îlots distincts:

1	0	1
0	1	0
1	0	1

Dans cette carte, il y en a 3:

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

On sait également que les îlots sont dits 'esthétiques', c'est à dire que si deux cellules appartiennent au même îlot et à la même ligne ou la même colonne, cela signifie qu'il n'y a que des cellules de terre entre eux sur cette ligne/colonne.

Par exemple, les cartes suivantes sont possibles:

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

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

La deuxième carte contient 4 îlots qui sont tous esthétiques.

Pourtant, les cartes suivantes ne sont pas possibles:

			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 également accès à un ordinateur doté d'une mémoire de K bits, numérotés de 0 à $K - 1$. Cette mémoire contient déjà les données de la carte. La valeur de la case (i, j) est stockée dans le bit $i \cdot M + j$.

Les organisateurs n'ont, par contre, aucun moyen de lire directement la mémoire, mais ils peuvent soumettre un programme à l'ordinateur.

Ce programme ne peut non plus lire directement la mémoire. Il ne peut que recevoir les valeurs de N , M and K . Il doit décrire une suite d'opérations bit à bit qu'effectue l'ordinateur sur la mémoire. Lorsque ces opérations sont exécutées, le programme doit ensuite choisir un ensemble de bits occupés qui représentent en binaire, allant du bit de poids faible au bit de poids fort, le nombre d'îlots dans le lac.

Initialement, les octets de $0, 1, \dots, N \cdot M - 1$ contiennent les données de la carte. Ces octets sont occupés et ne peuvent pas être modifiés. Les octets restants sont vides.

L'ordinateur peut effectuer 6 types d'instructions bit à bit. Ces instructions prennent deux indices A, B comme paramètres et écrivent le résultat dans l'adresse C du premier bit vide. Ces instructions retournent tous l'adresse C . Soit $memory[i]$ la valeur du i ème bit. Les instructions sont comme suit:

- $and(A, B)$: Le bit $memory[C]$ devient 1 si $memory[A] = memory[B] = 1$, sinon il devient 0.
- $or(A, B)$: Le bit $memory[C]$ devient 1 si $memory[A] = 1$ ou $memory[B] = 1$, sinon il devient 0.
- $xor(A, B)$: Le bit $memory[C]$ devient 1 si $memory[A] \neq memory[B]$, sinon il devient 0.
- $not(A)$: Le bit $memory[C]$ devient 1 si $memory[A] = 0$, sinon il devient 0.
- $nand(A, B)$: Le bit $memory[C]$ devient 1 si $memory[A] = 0$ ou $memory[B] = 0$, sinon il devient 0.
- $nor(A, B)$: Le bit $memory[C]$ devient 1 si $memory[A] = memory[B] = 0$, sinon il devient 0.

Dès que le ordinateur finit d'exécuter l'ensemble du programme des instructions, ce programme doit indiquer les bits qui représentent le nombre d'îlots.

Détails d'implémentation

Interface C++/Python

Si vous utilisez 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 mémoire
- Cette procédure est appelée exactement une fois par cas de test.
- Cette procédure doit retourner un tableau V qui contient les indices des bits qui composent la représentation binaire du nombre d'îlots: $2^0 \cdot \text{memory}[V[0]] + 2^1 \cdot \text{memory}[V[1]] + \dots + 2^{l-1} \cdot \text{memory}[V[l-1]]$ pour l longueur du tableau V .
- Tous les indices écrits dans V doivent être occupés.
- La longueur V doit être inférieure à 20.

Cette procédure doit appeler les procédures suivantes pour exécuter les instructions sur l'ordinateur:

```
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)
```

- Les paramètres A et B de chaque procédure doivent être indices de bit occupés.
- Chaque procédure retourne l'adresse du bit C modifié.

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

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

- L'indice A doit toujours être l'indice d'un bit occupé.
- L'évaluateur ignorera tout appel de cette procédure lors de la notation.
- L'évaluateur local, quant à lui, affichera l'étiquette `label` et la valeur des bits indexés par A juste à côté. Voir la section "évaluateur local pour plus de détails".

Interface C

Si vous utilisez C, il faut inclure l'entête `islands_c.h` et implémenter la procédure suivante:

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

L'évaluateur fournira `answer_bits`, un tableau de longueur `ISLANDS_MAX_ANSWER_BITS` (20). Il faut écrire les valeurs des indices dans ce tableau allant du bit de poids faible au bit de poids fort; il faut également retourner le nombre d'indices choisis. Ce nombre ne doit pas dépasser 20.

La signature de la fonction d'affichage est:

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

- `label` est une chaîne de caractères terminé par un octet nul.
- `A` est un tableau d'indices à afficher, qui peut être `NULL` lorsque `length` est de 0.
- `length` est le nombre d'indices renseignés dans `A`. Il ne peut pas être négatif.

Votre solution pourrait recevoir les messages d'erreur suivants:

- `Invalid index`: Dans les procédures précédents, vous avez fourni un indice non-valide, c'est à dire soit un indice négatif ou bien un indice non-occupé.
- `Too many instructions`: L'ordinateur a exécuté plus de $K - N \times Minstructions$, ce qui a saturé la mémoire.
- `Too many bits in the answer`: Un appel à `construct_instructions` a retourné plus de 20 indices de bit pour sa réponse finale.

Si le programme ne produit aucune erreur, et calcule le nombre correct d'îlots, il recevra un verdict `Accepted` et votre score sera calculé selon le tableau des sous-tâches. Sinon, il recevra un verdict de `Wrong Answer`.

Exemples

Exemple 1

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

Il y a quatre cas possibles

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

On remarque ici que, s'il existe une case contenant un 1, la réponse est forcément 1. On a donc un appel `append_or(0, 1)`, qui vérifie si une des deux cases est un 1, et stocke le résultat dans l'adresse 2. Notre procédure doit donc retourner le tableau `[2]`, car la réponse est donnée par $2^0 \times memory[2]$.

Exemple 2

Soit $N = 5$, $M = 5$, $K = 10^7$ avec 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'îlots est de 4. La réponse retournée par le programme doit avoir trois adresses qui contiennent les bits 0, 0 et 1 respectivement, car la réponse doit être $0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 = 4$

Contraintes

- $1 \leq N, M \leq 32$
- $K = 10^7$
- Chacune des cases contient soit 0 ou 1
- Tous les îlots sont esthétiques

Sous-tâches

1. (40 points) Chaque îlot est composé d'une seule cellule.
2. (20 points) Chaque îlot est un rectangle de cellules de terre.
3. (40 points) Aucune contrainte additionnelle.

Si votre programme est accepté et utilise I instructions, votre score P pour le cas de test est donné par:

- 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, $P = 10$.

L'Évaluateur Local

L'évaluateur local lit les paramètres de la procédure de la manière suivante:

- line 1: $N \ M \ K$

Ensuite il lit les cas de test de la manière suivante:

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

tel que $g[i][j]$ ($0 \leq i < N, 0 \leq j < M$) correspond à la valeur de la case (i, j) .

La description des cas de test est terminée par -1 .

L'évaluateur local appelle `construct_instructions` selon la signature correspondante à la langue de soumission. S'il encourt une erreur liée aux contraintes du problème, le message d'erreur correspondant sera affiché.

L'évaluateur local lira les cas de test selon l'ordre dans lequel ils ont été fournis et exécutera conformément la procédure `construct_instructions`.

Pour tout instruction `print(label, A)`, les valeurs seront affichés selon le format suivant:

- *label*: `memory[A[0]] memory[A[1]] ... memory[A[l-1]]`

Lorsque le programme termine l'exécution, l'évaluateur local affichera la valeur calculée à partir du tableau V retourné par la procédure `construct_instructions`, qui est donnée par:

$$2^0 \times \text{memory}[V[0]] + 2^1 \times \text{memory}[V[1]] + 2^2 \times \text{memory}[V[2]] + \dots$$

L'évaluateur affichera également le nombre X d'instructions exécutés par le programme comme suit:

Number of instructions: X