

Ascenseur

Limite de temps: 10 seconds

Limite de mémoire: 64 Mio (par processus)

Ceci est un problème de communication.

Considérons un immeuble avec des étages numérotés de 0 à N . L'étage 0 est le rez-de-chaussée, et au-dessus de l'étage N se trouve le toit. Ainsi, en incluant le toit, l'immeuble a $N + 2$ niveaux.

Chaque étage f ($0 \leq f \leq N$) est occupé par exactement un résident, qui est le seul à connaître a un entier secret v_f ($0 \leq v_f \leq 3$). Il y a aussi Karlsson qui vit sur le toit de cet immeuble. L'objectif est que Karlsson détermine autant de valeurs $v_0, v_1, \dots, v_N$ que possible, tandis que chaque résident **coopère** pour maximiser le nombre de valeurs qu'il peut retrouver.

Pour communiquer, les résidents utilisent l'ascenseur de l'immeuble de la manière suivante.

L'ascenseur commence à l'étage 0 et se déplace uniquement vers le haut. À l'intérieur de l'ascenseur, il y a des boutons étiquetés avec les étages 1 à N . Initialement, aucun bouton n'est pressé. Une fois qu'un bouton est pressé, il reste pressé de manière permanente. L'ascenseur s'arrête et s'ouvre à l'étage f si soit $f = 0$, soit le bouton correspondant à l'étage f a été pressé à un arrêt antérieur. Après avoir visité l'étage le plus élevé dont le bouton a été pressé (ou l'étage 0 si aucun bouton n'a jamais été pressé), l'ascenseur continue directement vers le toit sans s'arrêter à aucun des étages restants.

Quand l'ascenseur s'arrête à l'étage f , les événements suivants se produisent :

1. Le résident voit l'ensemble complet des boutons qui sont actuellement pressés.
2. En se basant uniquement sur cette information et sur la valeur v_f , il peut choisir n'importe quel sous-ensemble des boutons actuellement non pressés correspondant aux étages **strictement au-dessus** de son propre étage. Le résident presse ces boutons.
3. L'ascenseur monte vers le prochain étage le plus bas pour lequel le bouton correspondant a été pressé (ou vers le toit si tous les étages correspondant aux boutons pressés ont été visités).

Si l'ascenseur ne s'arrête pas à un certain étage, le résident correspondant ne peut pas presser de boutons.

Quand l'ascenseur atteint le toit, Karlsson voit uniquement l'ensemble final des boutons pressés. En utilisant cette information, il tente de retrouver autant de valeurs $v_0, v_1, \dots, v_N$ que possible.

Les valeurs de v sont fixées avant le démarrage de votre programme et ne changent pas pendant le processus.

Détails d'implémentation

Il y aura T sous-tests.

Soumettez un fichier implémentant ces deux fonctions.

La première fonction que vous devez implémenter est la suivante :

```
std::vector<int> press_buttons(int subtask, int N,
                               int f, int v, std::vector<int> p)
```

- $subtask$: la sous-tâche pour laquelle cette fonction est appelée, où $0 \leq subtask \leq 4$;
- N : le numéro du dernier étage ;
- f : l'étage actuel, où $0 \leq f \leq N$;
- v : la valeur v_f de l'étage f ;
- p : les boutons actuellement pressés dans l'ordre croissant.

Cette fonction est appelée lorsque l'ascenseur s'ouvre à l'étage f (si soit $f = 0$, soit le bouton correspondant à l'étage f a été pressé lors d'un arrêt antérieur). Cette fonction ne sera pas appelée pour un étage spécifique si le bouton correspondant n'a pas été pressé auparavant.

La valeur renvoyée est la liste des nouveaux boutons à presser. L'ordre des valeurs dans le tableau renvoyé n'a pas d'importance. Chaque bouton pressé x doit satisfaire les conditions suivantes :

- $f < x \leq N$;
- x n'est pas déjà dans p et x apparaît exactement une fois dans le tableau renvoyé.

La deuxième fonction que vous devez implémenter est la suivante :

```
std::vector<int> answer(int subtask, int N, std::vector<int> p)
```

- $subtask$: la sous-tâche pour laquelle cette fonction est appelée, où $0 \leq subtask \leq 4$;
- N : le numéro du dernier étage ;
- p : l'ensemble final des boutons pressés en ordre croissant.

Cette fonction est appelée exactement une fois par sous-test, quand l'ascenseur atteint le toit.

Le tableau renvoyé doit avoir une longueur de $N + 1$, le i -ème élément de celui-ci doit être égal à v_i , tandis que les valeurs que Karlsson **ne peut pas retrouver** doivent être remplacées par -1 .

Important : Les personnes dans le bâtiment ne peuvent communiquer d'aucune autre manière qu'en appuyant sur les boutons. Pour assurer cela, votre programme sera exécuté comme $N + 2$

processus séparés, un pour chaque étage et un pour le toit. Dans chaque processus pour les étages, il y aura au maximum T appels de la fonction `press_buttons` et dans le processus pour le toit il y aura exactement T appels de la fonction `answer`. Par conséquent, votre programme ne doit pas supposer un état partagé (par exemple des variables globales). Chaque processus aura 64 Mio de mémoire disponible séparée de la mémoire des autres processus, et tous ces appels (jusqu'à $(N + 1) \times T_{\text{press_buttons}}$ et exactement T_{answer}) doivent terminer en 10 secondes.

Contraintes

- $N = 60$
- $T \leq 10\,000$
- $0 \leq v_i \leq 3$ pour chaque $0 \leq i \leq N$

Exemple

Le test d'exemple a 1 sous-test avec les valeurs suivantes :

```
1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0 1 1 1
1 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1
```

Voici un exemple d'interaction :

Programme du participant	Programme du jury
	<code>press_buttons(0, 60, 0, 1, {})</code>
<code>return {2}</code>	
	<code>press_buttons(0, 60, 2, 0, {2})</code>
<code>return {13, 42}</code>	
	<code>press_buttons(0, 60, 13, 0, {2, 13, 42})</code>
<code>return {}</code>	
	<code>press_buttons(0, 60, 42, 1, {2, 13, 42})</code>
<code>return {}</code>	
	<code>answer(0, 60, {2, 13, 42})</code>
<code>return {1, -1, 0, -1, ..., -1}</code>	

Cette interaction retrouve correctement les 0-ème et 2-ème valeurs. Toutes les autres valeurs renvoyées sont égales à -1 car Karlsson ne les a pas retrouvées.

Sous-tâches

Sous-tâche	Points	Contraintes supplémentaires	Nombre d'étages à retrouver pour les points complets
0	0	Exemple du sujet.	–
1	15	$0 \leq v_i \leq 1$ $v_0 = v_N = 1$ Si $v_i = 0$, alors $v_{i+1} = 1$ pour chaque $0 \leq i < N$	60
2	35	$0 \leq v_i \leq 1$	40
3	15	$0 \leq v_i \leq 2$	30
4	35	$0 \leq v_i \leq 3$	25

Score

Si votre programme renvoie une valeur retrouvée incorrectement ou effectue une opération invalide (par exemple retourne un `vector` de longueur incorrecte) dans l'un des sous-tests d'une sous-tâche, votre soumission recevra zéro point pour cette sous-tâche.

Pour un sous-test le *nombre de positions retrouvées* de votre soumission est le nombre de positions dont la valeur renvoyée n'est pas -1 . Soit K le nombre minimum de positions retrouvées parmi tous les sous-tests d'une sous-tâche (chaque sous-tâche contient seulement 1 test contenant jusqu'à 10 000 sous-tests). Les points pour chaque sous-tâche sont calculés selon le tableau ci-dessous.

Sous-tâche	Plage de K	Points
0	-	0
1	$K < 60$	$0.25K$
	$K \geq 60$	15
2	$K < 30$	$0.5K$
	$30 \leq K < 40$	$2K - 45$
	$K \geq 40$	35
3	$K < 30$	$0.5K$
	$K \geq 30$	15
4	$K < 20$	$0.7K$
	$20 \leq K < 25$	$4K - 66$
	$K \geq 25$	35

Évaluateur d'exemple

L'évaluateur d'exemple effectue tous les appels de fonction pour tous les sous-tests en une seule exécution.

Le format d'entrée est le suivant :

- ligne 1 : trois entiers – le nombre de sous-tests T , le numéro du dernier étage N et le numéro de la sous-tâche S ;
- ligne $1 + i$: $N + 1$ entiers $v_0, v_1, \dots, v_N$, où v_f est la valeur de l'étage f pour le sous-test.

Le format de sortie est le suivant :

- ligne i : un entier – le nombre de valeurs de v correctement retrouvées dans le sous-test i ;
- ligne $T + 1$: les points finaux.

Pour des retours plus détaillés, changez la macro `DETAILED` de `false` à `true` dans la première ligne de l'évaluateur.

Vous pouvez laisser l'évaluateur générer les sous-tests (les valeurs de v_f) pour vous en changeant la macro `AUTO_GENERATE` de `false` à `true` dans la deuxième ligne de l'évaluateur.