

Partition

Temur et son assistant, Ulug'bek, préparent un tour de magie pour l'émission *L'Ouzbékistan a un incroyable talent*. Le tour consiste à résoudre le **problème de partition** suivant : partitionner une collection donnée d'entiers strictement positifs en K groupes non vides de sorte que la somme des éléments de chaque groupe soit la même. En d'autres termes, chaque entier de la collection doit être affecté à exactement un des K groupes, et la somme au sein de chaque groupe doit être égale. Par exemple, si la collection donnée est $[2, 1, 6, 4, 5]$ et $K = 3$, une partition valide serait $[2, 4]$, $[1, 5]$ et $[6]$. Dans ce cas, la somme des éléments de chaque groupe est exactement 6.

Le tour se déroule comme suit :

- Ulug'bek et Temur entrent dans des cabines séparées et ne peuvent pas communiquer entre eux.
- Un membre du jury donne à Ulug'bek un tableau A de N entiers strictement positifs, $A[0], A[1], \dots, A[N - 1]$, où chaque valeur est comprise entre 1 et M inclus. Le membre du jury donne également à Ulug'bek la valeur de K .
- Ulug'bek choisit au plus $K - 1$ entiers (pas nécessairement distincts) à **ajouter comme éléments supplémentaires** au tableau A . Chaque entier supplémentaire doit également être compris entre 1 et M inclus.
- Le jury ajoute les entiers choisis par Ulug'bek au tableau d'origine. Ce tableau étendu est ensuite **trié par ordre croissant** et remis à Temur, avec la valeur de K .
- Temur doit alors résoudre le problème de partition pour ce tableau étendu et trié.

Votre tâche consiste à implémenter une stratégie pour Temur et Ulug'bek. Sous ces contraintes, on peut prouver qu'il existe une stratégie qui leur permet de résoudre avec succès le problème de partition, quelle que soit le tableau A fourni par le jury.

Détails d'implémentation

Vous devez implémenter deux fonctions.

La fonction à implémenter pour **Ulug'bek** est :

```
std::vector<int> add_numbers(std::vector<int> A, int K, int M)
```

- A : un tableau de longueur N représentant le tableau original donné à Ulug'bek.

- K : le nombre de groupes à créer ; notez qu'Ulug'bek peut ajouter jusqu'à $K - 1$ entiers au tableau.
- M : la valeur maximale autorisée pour chaque entier d'origine et chaque entier supplémentaire.
- Cette procédure est appelée exactement une fois pour chaque fichier test.

La procédure doit renvoyer un tableau C contenant les entiers qu'Ulug'bek souhaite ajouter au tableau d'origine. Soit S la longueur du tableau C .

- S doit être au plus $K - 1$.
- Chaque élément de C doit être compris entre 1 et M inclus.

La fonction à implémenter pour **Temur** est :

```
std::vector<int> find_partition(std::vector<int> B, int K)
```

- B : un tableau de longueur $N + S$ contenant à la fois les entiers d'origine de A et les entiers ajoutés par Ulug'bek. Les éléments du tableau B sont triés par ordre croissant.
- K : le nombre de groupes à créer.
- Cette procédure est appelée exactement une fois pour chaque fichier test.

La procédure doit renvoyer un tableau P qui représente la partition de B en K groupes disjoints.

- La longueur de P doit être $N + S$.
- Pour chaque j tel que $0 \leq j < N + S$, $P[j]$ désigne le groupe auquel $B[j]$ appartient.
- Les groupes doivent être numérotés de 0 à $K - 1$, ce qui signifie que $0 \leq P[j] < K$ doit être vérifié pour chaque $0 \leq j < N + S$.
- Pour chaque i entre 0 et $K - 1$ inclus, il doit y avoir au moins un élément j ($0 \leq j < N + S$) tel que $P[j] = i$.
- La somme des éléments affectés à chacun des K groupes doit être identique.

Lors du calcul du score, un programme qui appelle les procédures ci-dessus est exécuté **exactement deux fois**.

- Lors de la première exécution du programme :
 - `add_numbers` est appelé exactement une fois.
 - Le tableau renvoyé est traité par l'évaluateur (grader) pour calculer le tableau B .
- Lors de la deuxième exécution du programme :
 - `find_partition` est appelé exactement une fois.

Contraintes

- $3 \leq N \leq 100\,000$
- $2 \leq K \leq 100\,000$
- $K \leq N$

- $1 \leq M \leq 10^9$
- $1 \leq A[i] \leq M$ pour chaque i tel que $0 \leq i < N$.

Sous-tâches

Sous-tâche	Score	Contraintes supplémentaires
1	5	$N = 3$
2	4	$M = 1$
3	7	$M \leq 2$
4	10	$N \leq 10$
5	7	$K = 2$
6	12	$K \leq 3$
7	17	$K \leq 10$
8	20	$K \leq 100$
9	18	Aucune contrainte supplémentaire.

Exemple

Considérons l'appel suivant :

```
add_numbers([8, 2, 9, 6, 1, 5, 5], 3, 9)
```

Dans cet exemple $A = [8, 2, 9, 6, 1, 5, 5]$. On veut répartir les nombres en $K = 3$ groupes. Ulug'bek peut ajouter des nombres entre 1 et $M = 9$.

La procédure peut renvoyer $[5, 4]$, ce qui signifie qu'Ulug'bek décide d'ajouter $S = 2$ entiers supplémentaires : 5 et 4. Ces valeurs sont valides puisque les deux éléments sont compris entre 1 et $M = 9$.

Le tableau étendu est ensuite trié et transmis à Temur avec l'appel suivant :

```
find_partition([1, 2, 4, 5, 5, 5, 6, 8, 9], 3)
```

On peut diviser les entiers de ce tableau en ces trois groupes : $[1, 5, 9]$, $[2, 5, 8]$, et $[4, 5, 6]$. Notez que la somme de chaque groupe est égale à 15. Pour encoder cette partition, la procédure doit renvoyer le tableau $[0, 1, 2, 0, 1, 2, 2, 1, 0]$.

Évaluateur d'exemple (grader)

Format d'entrée :

```
N K M
A[0] A[1] ... A[N-1]
```

Format de sortie :

Une fois l'appel à `add_numbers` terminé, l'évaluateur (grader) :

- Calcule le tableau trié B .
- Affiche les tableaux C et B au format suivant, **suivi d'une ligne vide** :

```
S
C[0] C[1] ... C[S-1]
B[0] B[1] ... B[N+S-1]
```

Une fois l'appel à `find_partition` terminé, l'évaluateur (grader) affiche :

```
L
P[0] P[1] ... P[L-1]
```

Ici, L est la longueur du tableau P renvoyé par `find_partition`.