

Partición

Abdurahmon y su asistente, Ulug'Saúl, están preparando un truco de magia para el programa de *Pequeños Gigantes*. El truco consiste en que Abdurahmon resuelva el siguiente **problema de partición**: hay que partir una colección de enteros positivos en K grupos no vacíos tal que la suma de los elementos de cada grupo sea la misma. En otras palabras, cada entero de la colección original debe de ser asignado a exactamente uno de los K grupos, y la suma de cada grupo debe de ser la misma. Por ejemplo, si la colección original es $[2, 1, 6, 4, 5]$ y $K = 3$, una partición válida sería $[2, 4]$, $[1, 5]$, y $[6]$. En este caso, la suma de los elementos en cada grupo es exactamente 6.

¡Pero ese no es el truco! El truco real es el siguiente:

- Ulug'Saúl y Abdurahmon se meten a diferentes cabinas y NO se pueden comunicar uno con el otro una vez dentro.
- Un miembro del jurado le da a Ulug'Saúl un arreglo A con N enteros, $A[0], A[1], \dots, A[N-1]$, donde cada valor está entre 1 y M inclusive. Ulug'Saúl también recibe el valor de K .
- Ulug'Saúl elige a lo más $K - 1$ enteros (no necesariamente distintos) para **agregar como nuevos elementos** al arreglo A . Cada entero agregado por Ulug'Saúl también debe de estar entre 1 y M inclusive.
- El jurado entonces agrega los enteros elegidos por Ulug'Saúl al arreglo original. Este nuevo arreglo extendido es **ordenado en orden no decreciente** y se le da a Abdurahmon. Abdurahmon también recibe el valor de K .
- Abdurahmon debe entonces resolver el problema de partición para este arreglo extendido y ordenado.

Tu tarea consiste en diseñar e implementar una estrategia para Abdurahmon y Ulug'Saúl. Se puede demostrar que, bajo las condiciones del problema, existe una estrategia óptima para Abdurahmon y Ulug'Saúl que les permita ganar sin importar qué arreglo A elegido el juez.

Detalles de implementación

Debes de implementar dos funciones.

La función que debes de implementar para **Ulug'Saúl** es:

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

- A : un arreglo de longitud N representando el arreglo original que se le da a Ulug'Saúl.
- K : el número deseado de grupos. Importante notar que Ulug'Saúl puede agregar a lo más $K - 1$ elementos al arreglo.
- M : el máximo valor permitido para cada entero agregado.
- Esta función se va a llamar exactamente una vez por cada caso de prueba.

Esta función debe de regresar un arreglo C con los enteros que Ulug'Saúl quiere agregar al arreglo original. Llamémosle S a la longitud del arreglo C .

- S debe de ser a lo más $K - 1$.
- El valor de cada entero de C debe de estar entre 1 y M inclusive.

La función que debes de implementar para **Abdurahmon** es:

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

- B : un arreglo de longitud $N + S$ conteniendo tanto los enteros originales en A como los enteros que Ulug'Saúl agregó (los enteros del arreglo C). Los elementos del arreglo B están ordenados en orden no decreciente.
- K : el número deseado de grupos.
- Esta función se va a llamar exactamente una vez por cada caso de prueba.

Esta función debe de regresar un arreglo P que represente la partición de B en K grupos disjuntos.

- La longitud de P debe de ser $N + S$.
- Para cada j tal que $0 \leq j < N + S$, $P[j]$ indica el grupo en el que está $B[j]$.
- Los grupos deben de ser numerados de 0 a $K - 1$, o sea que $0 \leq P[j] < K$ debe de ser verdad para cada $0 \leq j < N + S$.
- Para cada i entre 0 y $K - 1$ inclusive, debe de haber almenos un elemento j ($0 \leq j < N + S$) tal que $P[j] = i$.
- La suma de los elementos asignados a cada uno de los K grupos debe de ser idéntica.

Durante el proceso de evaluación, el programa que llama a las funciones descritas arriba se va a llamar **exactamente dos veces**.

- Durante la primera ejecución:
 - `add_numbers` se llama exactamente una vez.
 - El arreglo devuelto por tu función es procesado por el evaluador para computar el arreglo B .
- Durante la segunda ejecución:
 - `find_partition` se llama exactamente una vez.

Límites

- $3 \leq N \leq 100\,000$
- $2 \leq K \leq 100\,000$
- $K \leq N$
- $1 \leq M \leq 10^9$
- $1 \leq A[i] \leq M$ para cada i tal que $0 \leq i < N$.

Subtareas

Subtarea	Puntos	Condiciones adicionales
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	Sin condiciones adicionales

Ejemplo

Considera la siguiente llamada:

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

En este ejemplo $A = [8, 2, 9, 6, 1, 5, 5]$. Queremos partir los números en $K = 3$ grupos. Ulug'Saúl puede agregar enteros entre 1 y $M = 9$.

La primera función regresa $[5, 4]$, lo que quiere decir que Ulug'Saúl decidió agregar $S = 2$ enteros nuevos: 5 y 4. Éstos son válidos ya que ambos elementos están entre 1 y $M = 9$.

El arreglo extendido es entonces ordenado y se pasa a Abdurahmon con la siguiente llamada:

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

Podemos entonces dividir los enteros de este arreglo en tres grupos: $[1, 5, 9]$, $[2, 5, 8]$, y $[4, 5, 6]$. Nota que la suma de cada uno de estos grupos es igual a 15. Para denotar esta partición, la segunda función debe de regresar: $[0, 1, 2, 0, 1, 2, 2, 1, 0]$.

Evaluador de ejemplo

Formato de entrada:

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

Formato de salida:

Una vez que la llamada a `add_numbers` termine, el evaluador de ejemplo:

- Computa el arreglo ordenado B .
- Imprime el arreglo C y B en el siguiente formato, **seguido de una línea vacía**:

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

Una vez que la llamada a `find_partition` termine, el evaluador de ejemplo imprime:

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

Aquí, L es la longitud del arreglo P regresado por `find_partition`.