

Partition

Temur và trợ lý của anh, Ulug'bek, đang chuẩn bị một trò ảo thuật cho chương trình *Uzbekistan Got Talent*. Trò ảo thuật này xoay quanh việc Temur giải quyết **một bài toán phân hoạch** sau đây: phân chia một bộ các số nguyên dương đã cho thành K nhóm không rỗng sao cho tổng các phần tử trong mỗi nhóm đều bằng nhau. Nói cách khác, mỗi số nguyên trong bộ phải được gán vào đúng một trong K nhóm, và tổng các phần tử trong mỗi nhóm phải bằng nhau. Ví dụ, nếu bộ đã cho là $[2, 1, 6, 4, 5]$ và $K = 3$, một cách phân chia hợp lệ sẽ là $[2, 4]$, $[1, 5]$ và $[6]$. Trong trường hợp này, tổng các phần tử trong mỗi nhóm chính xác là 6.

Trò ảo thuật được thực hiện như sau:

- Ulug'bek và Temur vào các phòng riêng biệt và không thể giao tiếp với nhau.
- Một thành viên ban giám khảo đưa cho Ulug'bek một mảng A gồm N số nguyên dương, $A[0], A[1], \dots, A[N-1]$, trong đó mỗi giá trị nằm trong khoảng từ 1 đến M (tính cả hai đầu). Thành viên ban giám khảo cũng đưa cho Ulug'bek giá trị K .
- Ulug'bek chọn tối đa $K - 1$ (không nhất thiết phải khác nhau) số nguyên để **thêm như các phần tử mới** vào mảng A . Mỗi số nguyên được thêm vào cũng phải nằm trong khoảng từ 1 đến M (tính cả hai đầu).
- Ban giám khảo thêm các số nguyên do Ulug'bek chọn vào mảng ban đầu. Mảng mở rộng này sau đó được **sắp xếp theo thứ tự không giảm** và được chuyển cho Temur, kèm theo giá trị của K .
- Temur phải giải quyết bài toán phân hoạch cho mảng mở rộng và đã được sắp xếp này.

Nhiệm vụ của bạn là thiết kế và cài đặt một chiến lược cho Temur và Ulug'bek. Có thể chứng minh rằng, với các điều kiện ràng buộc đã cho, luôn tồn tại một chiến lược cho phép họ giải quyết thành công bài toán phân hoạch, bất kể mảng A nào do ban giám khảo cung cấp.

Chi tiết cài đặt

Bạn cần thực hiện hai hàm.

Hàm mà bạn cần cài đặt cho **Ulug'bek** là:

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

- A : một mảng có kích thước N , biểu diễn mảng ban đầu được cung cấp cho Ulug'bek.

- K : số nhóm mong muốn; lưu ý rằng Ulug'bek có thể thêm tối đa $K - 1$ số nguyên vào mảng.
- M : giá trị lớn nhất cho phép đối với mỗi số nguyên ban đầu và số nguyên mới được thêm vào.
- Hàm này được gọi đúng một lần cho mỗi trường hợp test.

Hàm này cần trả về một mảng C chứa các số nguyên mà Ulug'bek muốn thêm vào mảng ban đầu. Gọi S là kích thước của mảng C .

- S phải nhỏ hơn hoặc bằng $K - 1$.
- Mỗi phần tử của C phải nằm trong khoảng từ 1 đến M (tính cả hai đầu).

Hàm mà bạn cần cài đặt cho **Temur** là:

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

- B : một mảng có kích thước $N + S$ chứa cả các số nguyên ban đầu từ A và các số nguyên mà Ulug'bek đã thêm vào. Các phần tử của mảng B được sắp xếp theo thứ tự không giảm.
- K : số lượng nhóm mục tiêu.
- Hàm này được gọi đúng một lần cho mỗi trường hợp test.

Hàm này cần trả về một mảng P biểu thị sự phân hoạch của B thành K nhóm không giao nhau.

- Kích thước của P phải bằng $N + S$.
- Với mỗi j sao cho $0 \leq j < N + S$, $P[j]$ biểu thị nhóm mà $B[j]$ thuộc về.
- Các nhóm cần được đánh số từ 0 đến $K - 1$, nghĩa là $0 \leq P[j] < K$ phải đúng với mọi $0 \leq j < N + S$.
- Với mỗi i nằm giữa 0 và $K - 1$ (tính cả hai đầu), phải có ít nhất một phần tử j ($0 \leq j < N + S$) sao cho $P[j] = i$.
- Tổng các phần tử được gán cho mỗi nhóm trong số K nhóm phải giống nhau.

Trong quá trình chấm thực tế, một chương trình gọi các hàm nêu trên sẽ được chạy **đúng hai lần**.

- Trong lần chạy đầu tiên của chương trình:
 - `add_numbers` được gọi một lần duy nhất.
 - Mảng trả về là mảng được hệ thống chấm xử lý để tính ra mảng B .
- Trong lần chạy thứ hai của chương trình:
 - `find_partition` được gọi một lần duy nhất.

Các ràng buộc

- $3 \leq N \leq 100\,000$
- $2 \leq K \leq 100\,000$
- $K \leq N$
- $1 \leq M \leq 10^9$

- $1 \leq A[i] \leq M$ với mỗi i sao cho $0 \leq i < N$.

Các subtask

Subtask	Điểm	Các ràng buộc thêm
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	Không có ràng buộc nào thêm.

Ví dụ

Xét lời gọi hàm sau:

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

Trong ví dụ này $A = [8, 2, 9, 6, 1, 5, 5]$. Ta muốn phân hoạch các số thành $K = 3$ nhóm. Ulug'bek có thể thêm các số từ 1 đến $M = 9$.

Hàm có thể trả về $[5, 4]$, có nghĩa là Ulug'bek quyết định thêm $S = 2$ số nguyên mới: 5 và 4. Điều này hợp lệ vì cả hai phần tử đều nằm giữa 1 và $M = 9$.

Mảng mở rộng sau đó được sắp xếp và truyền cho Temur qua lời gọi sau:

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

Ta có thể chia các số nguyên từ mảng này thành ba nhóm sau: $[1, 5, 9]$, $[2, 5, 8]$, và $[4, 5, 6]$. Lưu ý rằng tổng của mỗi nhóm này bằng 15. Để chỉ ra phân hoạch này, hàm cần trả về mảng $[0, 1, 2, 0, 1, 2, 2, 1, 0]$.

Trình chấm mẫu

Định dạng dữ liệu vào:

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

Định dạng kết quả ra:

Sau lời gọi hàm `add_numbers` hoàn tất, trình chấm mẫu sẽ hiển thị như sau:

- Tính ra mảng đã được sắp xếp B .
- In ra các mảng C và B theo định dạng sau (**kết thúc bởi một dòng trống**):

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

Sau khi lời gọi `find_partition` hoàn tất, trình chấm sẽ đưa ra:

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

Trong đó, L là kích thước của mảng P được trả về bởi `find_partition`.