

Podział

Temur i jego asystent Ulug'bek przygotowują sztuczkę magiczną na potrzeby programu *Mam Talent Uzbekistan*. Sztuczka polega na rozwiązaniu przez Temura następującego **problemu podziału**: podzieli dany multizbiór dodatnich liczb całkowitych na K niepustych grup tak, aby suma elementów w każdej grupie była taka sama. Innymi słowy, każda liczba całkowita z multizbioru musi być przypisana dokładnie do jednej z K grup, a suma liczb w obrębie każdej grupy musi być równa. Na przykład, jeśli dany multizbiór to $[2, 1, 6, 4, 5]$ i $K = 3$, prawidłowym podziałem będzie $[2, 4]$, $[1, 5]$ i $[6]$. W tym przypadku suma elementów w każdej grupie wynosi dokładnie 6.

Sztuczka wygląda następująco:

- Ulug'bek i Temur wchodzi do oddzielnych kabin i nie mogą się ze sobą komunikować.
- Członek jury wręcza Ulug'bekowi ciąg A zawierający N dodatnich liczb całkowitych $A[0], A[1], \dots, A[N-1]$, gdzie każda wartość mieści się w przedziale od 1 do M włącznie. Członek jury wręcza Ulug'bekowi również wartość K .
- Ulug'bek wybiera co najwyżej $K - 1$ (niekoniecznie różnych) liczb całkowitych, aby **dołożyć je jako nowe elementy** do ciągu A . Każda dodana liczba całkowita musi również mieścić się w przedziale od 1 do M włącznie.
- Jury dodaje liczby całkowite wybrane przez Ulug'beka do oryginalnego ciągu. Powiększony ciąg jest następnie **sortowany w kolejności niemalejącej** i przekazywany Temurowi wraz z wartością K .
- Temur musi następnie rozwiązać problem podziału dla tego rozszerzonego, posortowanego ciągu.

Twoim zadaniem jest opracowanie i wdrożenie strategii dla Temura i Ulug'beka. Można udowodnić, że przy danych ograniczeniach istnieje dla nich strategia, która pozwoli im pomyślnie rozwiązać problem partycji, niezależnie od ciągu A wskazanego przez jury.

Szczegóły implementacji

Powinieneś zaimplementować dwie procedury.

Procedura, którą powinieneś zaimplementować w przypadku **Ulug'beka**, jest następująca:

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

- A : tablica o długości N reprezentująca oryginalny ciąg przekazany Ulug'bekowi.

- K : docelowa liczba grup; należy pamiętać, że Ulug'bek może dodać do ciągu do $K - 1$ liczb całkowitych.
- M : maksymalna dozwolona wartość dla każdej oryginalnej i nowo dodanej liczby całkowitej.
- Ta procedura jest wywoływana dokładnie raz dla każdego przypadku testowego.

Procedura powinna zwrócić tablicę C zawierającą liczby całkowite, które Ulug'bek chce dodać do oryginalnego ciągu. Niech S oznacza długość tablicy C .

- S musi wynosić co najwyżej $K - 1$.
- Każdy element C musi mieścić się pomiędzy 1 i M włącznie.

Procedura, którą powinienes zaimplementować w przypadku **Temura**, jest następująca:

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

- B : tablice o długości $N + S$ zawierająca zarówno oryginalne liczby całkowite z A , jak i liczby całkowite dodane przez Ulug'beka. Elementy tablicy B są posortowane w kolejności niemalejącej.
- K : docelowa liczba grup.
- Ta procedura jest wywoływana dokładnie raz dla każdego przypadku testowego.

Procedura powinna zwrócić tablicę P reprezentującą podział B na K rozłącznych grup.

- Długość P powinna wynosić $N + S$.
- Dla każdego j takiego, że $0 \leq j < N + S$, $P[j]$ oznacza grupę, do której należy $B[j]$.
- Grupy powinny być ponumerowane od 0 do $K - 1$, co oznacza, że dla każdego $0 \leq j < N + S$, mamy $0 \leq P[j] < K$.
- Dla każdego i pomiędzy 0 a $K - 1$ włącznie musi istnieć co najmniej jeden element j ($0 \leq j < N + S$) taki, że $P[j] = i$.
- Suma elementów przypisanych do każdej z grup K musi być identyczna.

W rzeczywistym ocenianiu program wywołujący powyższe procedury jest uruchamiany **dokładnie dwa razy**.

- Podczas pierwszego uruchomienia programu:
 - `add_numbers` jest wywoływane dokładnie raz.
 - Zwrócony ciąg jest przetwarzany przez program oceniający w celu obliczenia ciągu B .
- Podczas drugiego uruchomienia programu:
 - `find_partition` jest wywoływane dokładnie raz.

Ograniczenia

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

- $1 \leq M \leq 10^9$
- $1 \leq A[i] \leq M$ dla każdego i takiego, że $0 \leq i < N$.

Podzadania

Podzadanie	Punkty	Dodatkowe ograniczenia
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	Brak dodatkowych ograniczeń.

Przykład

Rozważmy następujące wywołanie:

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

W tym przykładzie $A = [8, 2, 9, 6, 1, 5, 5]$. Chcemy podzielić liczby na $K = 3$ grup. Ulug'bek może dodawać liczby od 1 do $M = 9$.

Procedura może zwrócić $[5, 4]$, co oznacza, że Ulug'bek decyduje się dodać $S = 2$ nowe liczby całkowite: 5 i 4. Są one poprawne, ponieważ obie mieszczą się w przedziale od 1 do $M = 9$.

Rozszerzony ciąg jest następnie sortowany i przekazywany do Temura za pomocą następującego wywołania:

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

Możemy podzielić liczby całkowite z tego ciągu na trzy grupy: $[1, 5, 9]$, $[2, 5, 8]$ i $[4, 5, 6]$. Należy zauważyć, że suma każdej z tych grup jest równa 15. Aby wskazać ten podział, procedura powinna zwrócić tablicę $[0, 1, 2, 0, 1, 2, 2, 1, 0]$.

Przykładowy program oceniający

Format wejściowy:

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

Format wyjściowy:

Po zakończeniu wywołania `add_numbers` przykładowy program oceniający:

- Oblicza posortowaną tablicę B .
- Wypisuje tablice C i B w następującym formacie, **po którym następuje pusta linia**:

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

Po zakończeniu wywołania `find_partition` program oceniający wyświetla:

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

Wartość L jest długością tablicy P zwróconej przez `find_partition`.