

## Partition

Temur und sein Assistent Ulug'bek bereiten einen Zaubertrick für die Show *Uzbekistan Got Talent* vor. Der Trick besteht darin, dass Temur das folgende **Partitionsproblem** löst: eine gegebene Menge positiver ganzer Zahlen in  $K$  nichtleere Gruppen aufzuteilen, sodass die Summe der Elemente in jeder Gruppe gleich ist. Mit anderen Worten: Jede ganze Zahl aus der Menge muss genau einer der  $K$  Gruppen zugeordnet werden und die Summe der Elemente in jeder Gruppe muss gleich sein. Wenn die gegebene Menge beispielsweise  $[2, 1, 6, 4, 5]$  ist und  $K = 3$ , dann wäre eine gültige Partition  $[2, 4]$ ,  $[1, 5]$  und  $[6]$ . In diesem Fall ist die Summe der Elemente in jeder Gruppe genau 6.

Der Zaubertrick wird folgendermaßen durchgeführt:

- Ulug'bek und Temur gehen in verschiedene Kabinen und können nicht miteinander kommunizieren.
- Ein Jurymitglied gibt Ulug'bek ein Array  $A$  mit  $N$  positiven ganzen Zahlen,  $A[0], A[1], \dots, A[N-1]$ , wobei jeder Wert zwischen 1 und  $M$  einschließlich liegt. Das Jurymitglied gibt Ulug'bek auch den Wert von  $K$ .
- Ulug'bek wählt dann höchstens  $K - 1$  (nicht notwendigerweise verschiedene) ganze Zahlen aus, die **als neue Elemente** zum Array  $A$  hinzugefügt werden. Jede hinzugefügte ganze Zahl muss außerdem zwischen 1 und  $M$  (einschließlich) liegen.
- Die Jury fügt die von Ulug'bek gewählten ganzen Zahlen dem ursprünglichen Array hinzu. Dieses erweiterte Array wird dann in **nicht-absteigender** Reihenfolge sortiert und zusammen mit dem Wert von  $K$  an Temur übergeben.
- Temur muss dann das Partitionsproblem für dieses erweiterte und sortierte Array lösen.

Deine Aufgabe ist es, eine Strategie für Temur und Ulug'bek zu finden und diese zu implementieren. Es lässt sich beweisen, dass unter den gegebenen Beschränkungen eine Strategie existiert, die es ermöglicht, das Partitionsproblem unabhängig vom von der Jury vorgegebenen Array  $A$  erfolgreich zu lösen.

## Implementierungsdetails

Du musst zwei Funktionen implementieren.

Die Funktion, die du für **Ulug'bek** implementieren musst, ist:

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

- $A$ : ein Array der Länge  $N$ , das ursprüngliche Array das Ulug'bek übergeben wird.
- $K$ : die Zielanzahl der Gruppen. Beachte, dass Ulug'bek bis zu  $K - 1$  ganze Zahlen zum Array hinzufügen kann.
- $M$ : der maximale Wert für jede ursprüngliche und neu hinzugefügte ganze Zahl.
- Diese Funktion wird für jeden Testfall genau einmal aufgerufen.

Die Funktion muss ein Array  $C$  zurückgeben, das die ganzen Zahlen enthält, die Ulug'bek dem ursprünglichen Array hinzufügen möchte. Sei  $S$  die Länge des Arrays  $C$ .

- $S$  darf höchstens  $K - 1$  sein.
- Jedes Element von  $C$  muss zwischen 1 und  $M$  einschließlich liegen.

Die Funktion, die du für **Temur** implementieren musst, ist:

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

- $B$ : ein Array der Länge  $N + S$  das sowohl die ursprünglichen ganzen Zahlen aus  $A$  als auch die von Ulug'bek hinzugefügten ganzen Zahlen enthält. Die Elemente des Arrays  $B$  sind in aufsteigender Reihenfolge.
- $K$ : die Zielanzahl der Gruppen.
- Diese Funktion wird für jeden Testfall genau einmal aufgerufen.

Die Funktion muss ein Array  $P$  zurückgeben, das die Partition von  $B$  in  $K$  disjunkte Gruppen darstellt.

- Die Länge von  $P$  muss  $N + S$  sein.
- Für jedes  $j$  mit  $0 \leq j < N + S$  bezeichnet  $P[j]$  die Gruppe zu der  $B[j]$  gehört.
- Die Gruppen müssen von 0 bis  $K - 1$  durchnummeriert sein, das heißt es gilt  $0 \leq P[j] < K$  für jedes  $0 \leq j < N + S$ .
- Für jedes  $i$  zwischen 0 und  $K - 1$  einschließlich muss es mindestens ein Element  $j$  ( $0 \leq j < N + S$ ) geben, sodass  $P[j] = i$ .
- Die Summe der Elemente, die jeder der  $K$  Gruppen zugeordnet sind, muss identisch sein.

Beim eigentlichen Grading wird ein Programm, das die oben genannten Funktion implementiert, **genau zweimal** ausgeführt.

- Während des ersten Programmlaufs:
  - `add_numbers` wird genau einmal aufgerufen.
  - Das zurückgegebene Array wird vom Bewertungssystem verarbeitet, um das Array  $B$  zu berechnen.
- Während des zweiten Programmdurchlaufs:
  - `find_partition` wird genau einmal aufgerufen.

## Beschränkungen

- $3 \leq N \leq 100\,000$
- $2 \leq K \leq 100\,000$
- $K \leq N$
- $1 \leq M \leq 10^9$
- $1 \leq A[i] \leq M$  für jedes  $i$ , sodass  $0 \leq i < N$ .

## Teilaufgaben

| Teilaufgabe | Punktzahl | Weitere Beschränkungen         |
|-------------|-----------|--------------------------------|
| 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        | Keine weiteren Beschränkungen. |

## Beispiele

Betrachte folgenden Funktionsaufruf:

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

In diesem Beispiel ist  $A = [8, 2, 9, 6, 1, 5, 5]$ . Wir möchten dieses Array in  $K = 3$  Gruppen aufteilen. Ulug'bek kann ganze Zahlen zwischen 1 und  $M = 9$  hinzufügen.

Die Funktion kann beispielsweise  $[5, 4]$  zurückgeben, was bedeutet, dass Ulug'bek beschließt,  $S = 2$  neue ganze Zahlen hinzuzufügen: 5 und 4. Diese sind gültig, da beide Elemente zwischen 1 und  $M = 9$  liegen.

Das erweiterte Array wird anschließend sortiert und mit folgendem Aufruf an Temur übergeben:

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

Wir könnten die ganzen Zahlen aus diesem Array in diese drei Gruppen aufteilen:  $[1, 5, 9]$ ,  $[2, 5, 8]$ , und  $[4, 5, 6]$ . Beachte, dass die Summe jeder dieser Gruppen gleich 15 ist. Um diese Partition zu melden, sollte die Funktion das Array  $[0, 1, 2, 0, 1, 2, 2, 1, 0]$  zurückgeben.

## Sample-Grader

Eingabeformat:

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

Ausgabeformat:

Nach Abschluss des Aufrufs von `add_numbers` wird der Sample-Grader ausgeführt:

- Er berechnet das sortierte Array  $B$ .
- Er gibt die Arrays  $C$  und  $B$  im folgenden Format aus, **gefolgt von einer Leerzeile**:

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

Nachdem der Aufruf von `find_partition` abgeschlossen ist, gibt der Grader Folgendes aus:

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

Hier ist  $L$  die Länge des von `find_partition` zurückgegebenen Arrays  $P$ .