افراز

تیمور و دستیارش اولوغ‌بیگ در حال آماده‌سازی یک شعبده‌بازی برای مسابقه‌ی استعدادیابی ازبکستان هستند. این شعبده حول حل این **مسئله‌ی افراز** توسط تیمور می‌گردد: یک لیست از اعداد صحیح مثبت را به K دسته‌ی ناتهی به گونه‌ای افراز کنید که مجموع اعداد دسته‌ها با هم برابر باشند. به زبان دیگر، هر عدد درون لیست باید دقیقا به یکی از دسته‌ها نسبت داده شود، و جمع اعداد درون هر کدام از دسته‌ها باید برابر باشند. برای مثال، اگر لیست اعداد داده‌شده $[2, 1, 6, 4, 5]$ باشد و $K = 3$، یک دسته‌بندی می‌تواند $[2, 4]$، $[1, 5]$، و $[6]$ باشد. در این صورت، جمع اعداد هر دسته دقیقا برابر 6 است.

شعبده به صورت زیر اجرا می‌شود:

- اولوغ‌بیگ و تیمور وارد اتاقک‌های مختلفی می‌شوند و نمی‌توانند با هم ارتباط بگیرند.
- یک عضو هیئت داوری یک آرایه‌ی A شامل N عدد صحیح مثبت به صورت $A[0], A[1], \ldots, A[N-1]$ را به اولوغ‌بیگ می‌دهد که هر عدد در این آرایه بین 1 تا M (شامل هر دو) است، همچنین مقدار K را در اختیار اولوغ‌بیگ قرار می‌دهد.
- اولوغ‌بیگ حداکثر $K - 1$ عدد صحیح (نه لزوما متمایز) انتخاب می‌کند که **به عنوان اعداد جدید** به دنباله‌ی A **اضافه** کند. هر عدد اضافه‌شده هم باید بین 1 تا M (شامل هردو) باشد.
- هیئت داوری اعداد انتخاب‌شده توسط اولوغ‌بیگ را به آرایه‌ی اولیه اضافه می‌کند. سپس آرایه‌ی بسط‌یافته را به **ترتیب غیرنزولی مرتب می‌کند** و همراه با عدد K به تیمور می‌دهد.
- سپس تیمور باید مسئله‌ی افراز را برای این آرایه‌ی بسط‌داده و مرتب‌شده حل کند.

وظیفه‌ی شما این است که برای تیمور و اولوغ‌بیگ یک استراتژی طراحی و پیاده‌سازی کنید. می‌توان اثبات کرد که تحت شرایط داده‌شده، یک استراتژی برای آن‌ها وجود دارد که مستقل از دنباله‌ی A که توسط هیئت داوری داده می‌شود، قادر به حل موفقیت‌آمیز مسئله‌ی افراز باشند.

جزئیات پیاده‌سازی

شما باید دو تابع زیر را پیاده‌سازی کنید.

تابعی که باید برای **اولوغ‌بیگ** پیاده‌سازی کنید به شرح زیر است:

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

- A: یک آرایه به طول N که نشان‌دهنده آرایه‌ی اولیه‌ای است که به اولوغ‌بیگ داده می‌شود.
- K: تعداد دسته‌های موردنظر; توجه کنید که اولوغ‌بیگ می‌تواند حداکثر $K - 1$ عدد به آرایه اضافه کند.
- M: بیشترین مقدار مجاز برای هر عدد در آرایه‌ی اولیه و اعداد اضافه‌شده
- این تابع دقیقا یک بار در هر نمونه فراخوانی می‌شود.

این تابع باید یک آرایه C برگرداند که شامل اعدادی است که اولوغ‌بیگ می‌خواهد به آرایه‌ی اولیه اضافه کند. عدد S را برابر با طول آرایه‌ی C در نظر بگیرید.

- S باید حداکثر $K-1$ باشد.
- هر عضو C باید بین 1 تا M (شامل هر دو) باشد.

تابعی که باید برای **تیمور** پیاده‌سازی کنید به شرح زیر است:

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

- B: یک آرایه به طول $N+S$ که شامل هم اعداد اولیه‌ی آرایه‌ی A و هم اعدادی است که اولوغ‌بیگ اضافه کرده است. اعضای آرایه‌ی B به صورت غیرنزولی مرتب شده‌اند.
- K: تعداد دسته‌های موردنظر.
- این تابع دقیقا یک بار در هر تست‌کیس فراخوانی می‌شود.

این تابع باید یک آرایه‌ی P برگرداند که نشان‌دهنده‌ی افراز آرایه‌ی B به K دسته‌ی مجزا است.

- طول P باید برابر با $N+S$ باشد.
- برای هر j که $0 \le j < N+S$، مقدار $P[j]$ نشان می‌دهد که $B[j]$ به کدام دسته تعلق دارد.
- دسته‌ها باید از 0 تا $K-1$ شماره‌گذاری شوند، یعنی برای هر $0 \le j < N+S$ باید $0 \le P[j] < K$ برقرار باشد.
- برای هر i بین 0 و $K-1$، باید حداقل یک عضو j ($0 \le j < N+S$) وجود داشته باشد که $P[j] = i$.
- مجموع اعضای قرارگرفته در هر یک از K دسته باید با یکدیگر برابر باشد.

در ارزیابی اصلی، برنامه‌ای که توابع بالا را فراخوانی می‌کند **دقیقا دو بار** اجرا می‌شود.

- در اجرای اول برنامه:
  - تابع `add_numbers` دقیقا یک بار فراخوانی می‌شود.
  - آرایه‌ی برگردانده‌شده توسط سیستم ارزیابی پردازش می‌شود تا آرایه‌ی B محاسبه شود.
- در اجرای دوم برنامه:
  - تابع `find_partition` دقیقا یک بار فراخوانی می‌شود.

محدودیت‌ها

- $3 \le N \le 100,000$
- $2 \le K \le 100,000$
- $K \le N$
- $1 \le M \le 10^9$
- برای هر i که $0 \le i < N$، داریم $1 \le A[i] \le M$.

زیرمسئله‌ها

Subtask	Score	Additional Constraints
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	No additional constraints.

مثال

فراخوانی زیر را در نظر بگیرید:

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

در این مثال $A = [8, 2, 9, 6, 1, 5, 5]$ است. می‌خواهیم اعداد را به $K = 3$ دسته تقسیم کنیم. اولوغ‌بیگ می‌تواند اعدادی بین 1 و $M = 9$ اضافه کند.

تابع می‌تواند آرایه‌ی $[5, 4]$ را برگرداند، یعنی اولوغ‌بیگ تصمیم می‌گیرد $S = 2$ عدد جدید، یعنی 5 و 4 را اضافه کند. این اعداد معتبرند، چون هر دو بین 1 و $M = 9$ قرار دارند.

سپس آرایه‌ی بسط‌یافته مرتب می‌شود و با فراخوانی زیر به تیمور داده می‌شود:

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

می‌توانیم اعداد این آرایه را به سه دسته‌ی $[1, 5, 9]$، $[2, 5, 8]$ و $[4, 5, 6]$ تقسیم کنیم. توجه کنید که مجموع اعضای هر یک از این دسته‌ها برابر با 15 است. برای گزارش این افراز، تابع باید آرایه‌ی $[0, 1, 2, 0, 1, 2, 2, 1, 0]$ را برگرداند.

ارزیاب نمونه

قالب ورودی:

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

قالب خروجی:

پس از اتمام فراخوانی `add_numbers`، ارزیاب نمونه:

- آرایه‌ی مرتب‌شده‌ی B را محاسبه می‌کند.
- آرایه‌های C و B را با قالب زیر چاپ می‌کند و **پس از آن یک خط خالی** چاپ می‌شود:

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

پس از اتمام فراخوانی `find_partition`، ارزیاب خروجی زیر را چاپ می‌کند:

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

در اینجا L طول آرایه‌ی P است که توسط `find_partition` برگردانده می‌شود.