تیم‌ها

کلاسی را در نظر بگیرید شامل N دانش‌آموز که با اعداد ۰ تا $N-1$ شماره‌گذاری شده‌اند. معلم کلاس هر روز تعدادی پروژه به این دانش‌آموزان می‌دهد. هر کدام از این پروژه‌ها باید در همان روز توسط یک تیم از دانش‌آموزان انجام شود. میزان سختی پروژه‌ها متفاوت است و معلم می‌داند که تیم تخصیص‌داده شده برای هر کدام از پروژه‌ها باید دقیقاً شامل چند نفر باشد.

دانش‌آموزان مختلف ممکن است ترجیح دهند در تیم‌های با اندازه‌های متفاوتی باشند. به بیان دقیق‌تر، دانش‌آموز i باید به تیمی با اندازه‌ی حداقل $A[i]$ و حداکثر $B[i]$ تخصیص داده شود. هر دانش‌آموز هر روز می‌تواند حداکثر به یک تیم تخصیص داده شود. بعضی از دانش‌آموزان ممکن است به هیچ تیمی تخصیص داده نشوند. هر تیم روی دقیقاً یک پروژه کار می‌کند.

معلم برای هر یک از Q روز آینده تعدادی پروژه انتخاب کرده است. برای هر یک از این روزها، تعیین کنید که آیا می‌توان دانش‌آموزان را به پروژه‌های مربوط به آن روز به گونه‌ای تخصیص داد که هر پروژه دقیقاً یک تیم داشته باشد.

مثال

فرض کنید $N = ۴$ دانش‌آموز و $Q = ۲$ روز داریم. محدودیت اندازه‌ی تیم‌ها برای هر کدام از این دانش‌آموزان در جدول زیر داده شده است:

دانش‌آموز	۰	۱	۲	۳
A	۱	۲	۲	۲
B	۲	۳	۳	۴

در روز اول $M = ۲$ پروژه وجود دارند و اندازه‌ی تیم لازم برای این پروژه‌ها $K[۰] = ۱$ و $K[۱] = ۳$ است. یک راه‌حل ممکن برای این روز این است که دانش‌آموز ۰ به تیمی با اندازه‌ی ۱ و باقی دانش‌آموزان به تیمی با اندازه‌ی ۳ تخصیص داده شوند.

در روز دوم هم $M = ۲$ پروژه وجود دارند، ولی این بار اندازه‌ی تیم لازم برای این پروژه‌ها $K[۰] = ۱$ و $K[۱] = ۱$ است. چون فقط یکی از دانش‌آموزان می‌تواند در تیمی با اندازه‌ی یک باشد، مسئله برای این روز جوابی ندارد.

مسئله

اطلاعات مربوط به دانش‌آموزان (N، A و B) و همچنین Q سؤال (یکی برای هر روز) به شما داده شده است. هر سؤال شامل تعداد پروژه‌های آن روز (M) و همچنین دنباله‌ی K متشکل از M عدد است که اندازه‌ی تیم‌ها را مشخص می‌کنند. برای هر سؤال، برنامه‌ی شما باید تعیین کند که آیا شکل‌دهی این تیم‌ها امکان‌پذیر است یا خیر.

شما باید دو تابع `init` و `can` را پیاده‌سازی کنید:

- `init(N, A, B)` — این تابع، اولِ کار دقیقاً یک بار توسط ارزیاب فراخوانی می‌شود.
  - `N`: تعداد دانش‌آموزان
  - `A`: آرایه‌ای به طول N: $A[i]$ حداقل اندازه‌ی تیم برای دانش‌آموز i است.
  - `B`: آرایه‌ای به طول N: $B[i]$ حداکثر اندازه‌ی تیم برای دانش‌آموز i است.
  - این تابع چیزی برنمی‌گرداند.
  - می‌توانید فرض کنید که برای هر i از ۰ تا $N-1$ داریم $1 \le A[i] \le B[i] \le N$.
- `can(M,K)` — پس از این که `init` یک بار فراخوانی شد، ارزیاب این تابع را Q بار پشت سر هم (یک بار برای هر روز) فراخوانی می‌کند.
  - `M`: تعداد پروژه‌های این روز
  - `K`: آرایه‌ای به طول M شامل اندازه‌ی تیم‌های مورد نیاز برای هر کدام از این پروژه‌ها.
  - این تابع باید در صورتی که شکل‌دهی همه‌ی این تیم‌ها ممکن باشد ۱ و در غیر این صورت ۰ برگرداند.
  - می‌توانید فرض کنید که $1 \le M \le N$ و برای هر $i = 0, \ldots, M-1$ داریم $1 \le K[i] \le N$. توجه کنید که جمع همه‌ی $K[i]$ها ممکن است بیش‌تر از N باشد.

زیرمسئله‌ها

جمع همه‌ی مقادیر M در تمام دفعاتی که تابع `can(M,K)` فراخوانی می‌شود را با S نشان می‌دهیم.

زیرمسئله	امتیاز	N	Q	محدودیت‌های دیگر
1	21	$1 \le N \le 100$	$1 \le Q \le 100$	none
2	13	$1 \le N \le 100,000$	$Q = 1$	none
3	43	$1 \le N \le 100,000$	$1 \le Q \le 100,000$	$S \le 100,000$
4	23	$1 \le N \le 500,000$	$1 \le Q \le 200,000$	$S \le 200,000$

ارزیاب نمونه

ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: `N`
- خطوط ۲ تا `N+1`: `A[i]` و سپس `B[i]`
- خط `N+2`: `Q`
- خطوط `N+3` تا `N+Q+2`: مقادیر `M` و سپس `K[0]` تا `K[M-1]`

برای هر سؤال، ارزیاب نمونه مقدار خروجی تابع `can` را چاپ می‌کند.

ترازو

«امینه» شش سکه دارد که از ۱ تا ۶ شماره‌گذاری شده‌اند. او می‌داند که وزن تمامی سکه‌ها متفاوت است. او دوست دارد که سکه‌ها را برحسب وزن‌شان مرتب کند. به همین منظور، او یک ترازوی جدید طراحی کرده است.

ترازوهای سنتی دارای دو کفه‌اند. برای استفاده از این ترازوها، روی هر کفه یک سکه قرار می‌دهیم و ترازو سکه‌ی سنگین‌تر را مشخص می‌کند.

ترازوی جدید امینه پیچیده‌تر است. این ترازو چهار کفه دارد که با برچسب‌های A، B، C و D مشخص شده‌اند. ترازو چهار «تنظیم» متفاوت دارد، و هر کدام از این تنظیم‌ها، سؤال متفاوتی را در مورد سکه‌ها پاسخ می‌دهد. برای استفاده از ترازو، امینه باید روی هر یک از کفه‌های A، B و C دقیقاً یک سکه قرار دهد. علاوه بر این، در تنظیم چهارم، او باید روی کفه‌ی D هم دقیقاً یک سکه قرار دهد.

ترازو بر اساس چهار تنظیم مذکور به یکی از چهار سؤال زیر پاسخ می‌دهد:

۱. سنگین‌ترین سکه از بین سکه‌های کفه‌های A، B و C کدام است؟

۲. سبک‌ترین سکه از بین سکه‌های کفه‌های A، B و C کدام است؟

۳. کدام‌یک از سکه‌های کفه‌های A، B و C دارای وزن میانه است؟ (منظور سکه‌ای است که نه سنگین‌ترین سکه است و نه سبک‌ترین سکه.)

۴. از میان سکه‌هایی که در کفه‌های A، B و C قرار دارند، تنها سکه‌هایی را در نظر بگیرید که از سکه‌ی کفه‌ی D سنگین‌ترند. اگر چنین سکه‌هایی وجود داشته باشند، سبک‌ترین سکه از بین این سکه‌ها کدام است؟ اگر چنین سکه‌هایی وجود نداشته باشند، سبک‌ترین سکه از بین سکه‌های کفه‌های A، B و C کدام است؟

مسئله

برنامه‌ای بنویسید که سکه‌های امینه را بر حسب وزن‌شان مرتب کند. برنامه‌ی شما می‌تواند برای مقایسه‌ی سکه‌ها از ترازوی امینه استفاده کند. به برنامه‌ی شما چند «مورد آزمون»[۱] داده خواهد شد که هر یک، متناظر با مجموعه‌ای جدید از شش سکه است.

برنامه‌ی شما باید توابع `init` و `orderCoins` را پیاده‌سازی کند. در هر اجرای برنامه‌ی شما، ارزیاب ابتدا تابع `init` را دقیقاً یک بار فراخوانی می‌کند. این فراخوانی تعداد موارد آزمون را به شما می‌دهد و به‌کمک آن می‌توانید متغیرهایتان را مقداردهی اولیه کنید. سپس، ارزیاب به‌ازای هر مورد آزمون، یک‌بار تابع `orderCoins()` را فراخوانی می‌کند.

[۱] Test case

- `init(T)`
  - `T`: تعداد موارد آزمونی است که برنامه‌ی شما باید در این اجرا پاسخ دهد. `T` یک عدد صحیح در محدوده‌ی ۱ تا ۱۸ است.
  - این تابع مقداری به عنوان خروجی برنمی‌گرداند.
- `orderCoins()`
  - این تابع به ازای هر مورد آزمون، دقیقاً یک‌بار فراخوانی می‌شود.
  - این تابع باید ترتیب درست سکه‌های امینه را با استفاده از فراخوانی توابع ارزیاب `getHeaviest()`، `getLightest()`، `getMedian()` و `getNextLightest()` مشخص کند.
  - زمانی که تابع، ترتیب درست سکه‌ها را به‌دست می‌آورد، باید آن را با فراخوانی تابع ارزیاب `answer()` گزارش دهد.
  - پس از فراخوانی `answer()`، تابع `orderCoins()` باید خاتمه یابد. این تابع مقداری برنمی‌گرداند.

شما می‌توانید از توابع ارزیاب زیر استفاده کنید:

- `answer(W)` — برنامه‌ی شما باید از این تابع برای گزارش جوابی که پیدا کرده است استفاده کند.
  - `W`: آرایه‌ای به طول ۶، شامل ترتیب درست سکه‌ها. `W[0]` تا `W[5]` باید شماره‌ی سکه‌ها (یعنی اعداد ۱ تا ۶) به ترتیب از سبک‌ترین به سنگین‌ترین سکه باشد.
  - برنامه‌ی شما باید این تابع را فقط از داخل `orderCoins()` یک بار به‌ازای هر مورد آزمون فراخوانی کند.
  - این تابع مقداری برنمی‌گرداند.
- `getHeaviest(A, B, C)`، `getLightest(A, B, C)`، `getMedian(A, B, C)` — این توابع به‌ترتیب متناظر با تنظیم‌های شماره‌ی ۱، ۲ و ۳ برای ترازوی امینه هستند.
  - `A`، `B`، `C`: سکه‌هایی که به ترتیب روی کفه‌های A، B و C قرار می‌گیرند. `A`، `B` و `C` باید سه عدد صحیح متمایز در محدوده‌ی ۱ تا ۶ باشند.
  - هر یک از توابع یکی از اعداد `A`، `B` و `C` را برمی‌گرداند که شماره‌ی سکه‌ی خواسته‌شده است. برای مثال، تابع `getHeaviest(A, B, C)` شماره‌ی سنگین‌ترین سکه از بین سه سکه‌ی داده شده را می‌دهد.
- `getNextLightest(A, B, C, D)` — این تابع متناظر با تنظیم شماره‌ی ۴ برای ترازوی امینه است.
  - `A`، `B`، `C`، `D`: سکه‌هایی که به ترتیب بر روی کفه‌های A، B، C و D قرار گرفته‌اند. `A`، `B`، `C`،و `D` باید چهار عدد صحیح متمایز در محدوده‌ی ۱ تا ۶ باشند.
  - این تابع یکی از اعداد `A`، `B` و `C` را برمی‌گرداند که شماره‌ی سکه‌ای است که توسط ترازو در تنظیم شماره‌ی ۴ مطابق توضیح فوق انتخاب می‌شود. بدین معنی که سکه‌ی برگردانده‌شده سبک‌ترین سکه از بین سکه‌های کفه‌های A، B و C است که از سکه‌ی کفه‌ی D سنگین‌تر هستند؛ و یا در صورتی که هیچ‌یک از این سکه‌ها از سکه‌ی کفه‌ی D سنگین‌تر نباشند، سکه‌ی برگردانده‌شده سبک‌ترین سکه از بین تمامی سکه‌های کفه‌های A، B و C است.

امتیازدهی

این سؤال هیچ زیرمسئله‌ای ندارد. در عوض، امتیاز شما برحسب تعداد توزین‌های برنامه (یعنی تعداد فراخوانی‌های توابع ارزیاب `getLightest()`، `getHeaviest()`، `getMedian()`، و `getNextLightest()`) مشخص می‌شود.

برنامه‌ی شما چند بار اجرا می‌شود و در هر اجرا، چند مورد آزمون به آن داده می‌شود. فرض کنید r تعداد اجراهای برنامه باشد. اگر برنامه‌ی شما حتی در یکی از موارد آزمونِ یکی از اجراها ترتیب درست را برنگرداند، امتیاز صفر به برنامه داده می‌شود. در غیر این‌صورت، امتیاز اجراها به طور جداگانه به صورت زیر محاسبه می‌شود.

فرض کنید Q کوچک‌ترین عددی است که مرتب‌کردن هر دنباله‌ای از شش سکه، با استفاده از Q بار توزین با ترازوی امینه ممکن باشد. برای این که مسئله چالشی‌تر شود، مقدار Q را در این‌جا مشخص نمی‌کنیم.

فرض کنید بیش‌ترین تعداد توزین‌ها در میان تمامی موارد آزمونِ تمامی اجراها برابر با $Q + y$ (به ازای یک عدد صحیح y) باشد. حال، یک اجرا از برنامه‌ی خود را در نظر بگیرید. فرض کنید بیش‌ترین تعداد توزین در بین T مورد آزمون این اجرا برابر $Q + x$ (به ازای یک عدد صحیح نامنفی x) باشد. (در صورتی که تعداد دفعات توزین برنامه‌ی شما برای تمام موارد آزمون کم‌تر از Q باشد، آن‌گاه $x = ۰$ است.) در این صورت، امتیاز شما برای این اجرا برابر $\frac{۱۰۰}{r(\frac{x+y}{۵}+۱)}$ است که این عدد تا دو رقم اعشار به پایین گرد می‌شود.

به طور خاص، اگر برنامه‌ی شما به ازای هر یک از موارد آزمونِ تمامی اجراها، حداکثر از Q مرتبه توزین استفاده کند، نمره‌ی شما ۱۰۰ می‌شود.

مثال

فرض کنید سکه‌ها به ترتیب (از چپ به راست) ۵ ۱ ۲ ۶ ۴ ۳ از سبک‌ترین تا سنگین‌ترین باشند.

فراخوانی تابع	خروجی	توضیح
`getMedian(4,5,6)`	6	سکه‌ی ۶ میانه‌ی سکه‌های ۴، ۵ و ۶ است.
`getHeaviest(3,1,2)`	1	سکه‌ی ۱ سنگین‌ترین سکه میان سکه‌های ۱، ۲ و ۳ است.
`getNextLightest(2,3,4,5)`	3	از میان سکه‌های ۲، ۳ و ۴ سبک‌ترین سکه‌ی سنگین‌تر از ۵، سکه‌ی ۳ است.
`getNextLightest(1,6,3,4)`	6	از میان سکه‌های ۱، ۶ و ۳ سبک‌ترین سکه‌ی سنگین‌تر از ۴، سکه‌ی ۶ است.
`getHeaviest(3,5,6)`	5	سکه‌ی ۵ سنگین‌ترین سکه میان سکه‌های ۳، ۵ و ۶ است.
`getMedian(1,5,6)`	1	سکه‌ی ۱ میانه‌ی سکه‌های ۱، ۵ و ۶ است.
`getMedian(2,4,6)`	6	سکه‌ی ۶ میانه‌ی سکه‌های ۲، ۴ و ۶ است.
`answer([3,4,6,2,1,5])`		برنامه پاسخ درست را برای این مورد آزمون پیدا کرده است.

ارزیاب نمونه

ارزیاب نمونه ورودی را با فرمت زیر می‌خواند:

- سطر ۱: مقدار T — تعداد موارد آزمون

- هر یک از سطرهای ۲ تا $T+1$: دنباله‌ای از ۶ عدد طبیعی متمایز در محدوده‌ی ۱ تا ۶ — ترتیب سکه‌ها از سبک‌ترین به سنگین‌ترین

برای مثال، ورودی زیر شامل دو مورد آزمون با ترتیب‌های ۱ ۲ ۳ ۴ ۵ ۶ و ۳ ۴ ۶ ۲ ۱ ۵ است:

```
2
1 2 3 4 5 6
3 4 6 2 1 5
```

ارزیاب نمونه آرایه‌ای را که به عنوان پارامتر به تابع `answer()` داده می‌شود، چاپ می‌کند.

جعبه‌های سوغاتی

آخرین بخش مراسم افتتاحیه‌ی IOI 2015 در حال برگزاری است. در حین مراسم افتتاحیه، قرار است هر تیم یک جعبه‌ی سوغاتی از میزبان دریافت کند. منتها داوطلبان آن‌قدر جذب مراسم شده‌اند که کاملا موضوع سوغاتی‌ها را فراموش کرده‌اند. تنها کسی که موضوع سوغاتی‌ها را فراموش نکرده، «اَمان» است. امان داوطلب پرشوری است و می‌خواهد که IOI به بهترین نحو برگزار شود. بنابراین او تصمیم می‌گیرد که تمام سوغاتی‌ها را به تنهایی و در کم‌ترین زمان ممکن توزیع کند.

محل برگزاری مراسم افتتاحیه به شکل یک دایره است که به L قسمت مساوی تقسیم شده است. این قسمت‌ها به ترتیب از ۰ تا $L-1$ شماره‌گذاری شده‌اند. یعنی به ازای هر $0 \leq i \leq L-2$، قسمت‌های i و $i+1$ مجاورند و همچنین قسمت‌های ۰ و $L-1$ مجاور هستند. در محل مراسم N تیم حضور دارند. هر تیم در یکی از قسمت‌ها مستقر شده است. یک قسمت می‌تواند شامل بیش‌تر از یک تیم باشد. برخی قسمت‌ها هم ممکن است خالی باشند.

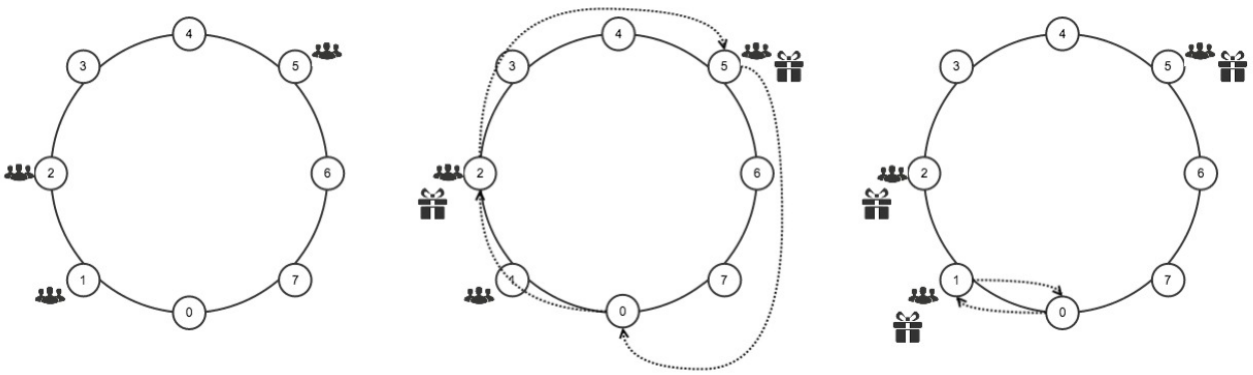
تعداد N سوغاتی کاملا مشابه وجود دارد. ابتدا امان و سوغاتی‌ها در قسمت ۰ قرار دارند. امان باید به هر تیم یک سوغاتی بدهد و بعد از تحویل آخرین سوغاتی به قسمت ۰ برگردد. دقت کنید که برخی تیم‌ها ممکن است در قسمت ۰ مستقر شده باشند.

امان در هر لحظه می‌تواند حداکثر K سوغاتی را با خود حمل کند. امان باید سوغاتی‌ها را از قسمت ۰ بردارد که این کار وقتی از او نمی‌گیرد. هر سوغاتی باید حمل شود تا به یکی از تیم‌ها تحویل داده شود. هر زمان که امان حداقل یک سوغاتی در دست دارد و به قسمتی می‌رسد که شامل یک تیم بدون سوغاتی است، می‌تواند یکی از سوغاتی‌های خود را به آن تیم بدهد. عمل تحویل سوغاتی هم زمانی نمی‌برد. تنها موضوعی که زمان‌بر است، حرکت است. امان می‌تواند به دور محل دایره‌ای‌شکل در هر دو جهت حرکت کند. حرکت امان از یک قسمت به قسمت مجاور (ساعت‌گرد یا پادساعت‌گرد) مستقل از تعداد سوغاتی‌هایی که در دست دارد، دقیقا یک ثانیه طول می‌کشد.

وظیفه‌ی شما یافتن کم‌ترین تعداد ثانیه‌هایی است که امان می‌تواند تمام سوغاتی‌ها را تحویل داده و به موقعیت اولش برگردد.

مثال

در این مثال، $N = ۳$ تیم داریم و امان در هر لحظه می‌تواند حداکثر $K = ۲$ سوغاتی حمل کند. تعداد قسمت‌ها نیز $L = ۸$ است. تیم‌ها در قسمت‌های ۱، ۲ و ۵ مستقر شده‌اند.



یکی از جواب‌های بهینه‌ی ممکن در شکل بالا نشان داده شده است. در دور اول، امان ابتدا دو سوغاتی را برداشته و به ترتیب به تیم‌های حاضر در قسمت‌های ۲ و ۵ تحویل می‌دهد و به قسمت ۰ برمی‌گردد. این رفت و برگشت ۸ ثانیه زمان می‌برد. در دور دوم، امان سوغاتی باقی‌مانده را برداشته و به تیم حاضر در قسمت ۱ تحویل می‌دهد و به قسمت ۰ برمی‌گردد. این رفت و برگشت ۲ ثانیه‌ی دیگر زمان می‌برد. در نتیجه کل زمان موردنیاز در این مثال ۱۰ ثانیه است.

مسئله

مقادیر N، K، L و موقعیت تمام تیم‌ها به شما داده شده است. شما باید کم‌ترین تعداد ثانیه‌هایی را که امان برای تحویل تمام سوغاتی‌ها و برگشتن به قسمت ۰ نیاز دارد محاسبه کنید. شما باید تابع `delivery` را پیاده‌سازی کنید.

- `delivery(N, K, L, positions)` — این تابع دقیقا یک بار توسط ارزیاب فراخوانی می‌شود.
  - `N`: تعداد تیم‌ها
  - `K`: حداکثر تعداد سوغاتی‌هایی که امان در هر لحظه می‌تواند حمل کند.
  - `L`: تعداد قسمت‌ها در محل برگزاری مراسم افتتاحیه
  - `positions`: یک آرایه به طول N. مقادیر `positions[0]` تا `positions[N-1]` شماره‌ی قسمت‌های تیم‌ها را نشان می‌دهند. مقادیر درون آرایه‌ی `positions` به ترتیب غیرنزولی هستند.

زیرمسئله‌ها

زیرمسئله	امتیاز	N	K	L
1	10	$1 \le N \le 1,000$	$K = 1$	$1 \le L \le 10^9$
2	10	$1 \le N \le 1,000$	$K = N$	$1 \le L \le 10^9$
3	15	$1 \le N \le 10$	$1 \le K \le N$	$1 \le L \le 10^9$
4	15	$1 \le N \le 1,000$	$1 \le K \le N$	$1 \le L \le 10^9$
5	20	$1 \le N \le 10^6$	$1 \le K \le 3,000$	$1 \le L \le 10^9$
6	30	$1 \le N \le 10^7$	$1 \le K \le N$	$1 \le L \le 10^9$

ارزیاب نمونه

ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: `N`، سپس `K` و `L`
- خط ۲: `positions[0]` تا `positions[N-1]`

ارزیاب نمونه مقدار خروجی تابع `delivery` را چاپ می‌کند.