ملاحظات پیاده‌سازی

- شما باید دقیقا یک فایل ارسال کنید (نام فایل در صورت سوال داده شده است).
- این فایل باید زیربرنامه‌هایی را که در صورت سوال توضیح داده شده در قالبی که در پیاده‌سازی نمونه داده شده پیاده‌سازی کند.
- رفتار این زیربرنامه‌ها باید مطابق آن چیزی باشد که در صورت سوال توضیح داده شده است.
- شما می‌توانید برای استفاده‌ی داخلی زیربرنامه‌های دیگری نیز پیاده‌سازی کنید.
- برنامه‌ی ارسالی شما نباید به هیچ طریقی با ورودی و خروجی استاندارد و یا هر فایل دیگری در تعامل باشد.

محدودیت‌ها

Task	Time Limit	Memory Limit
Boxes	2 seconds	1500 MB
Scales	1 second	1500 MB
Teams	4 seconds	1500 MB

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»[1] 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.

[1] 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.

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

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.

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.

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.

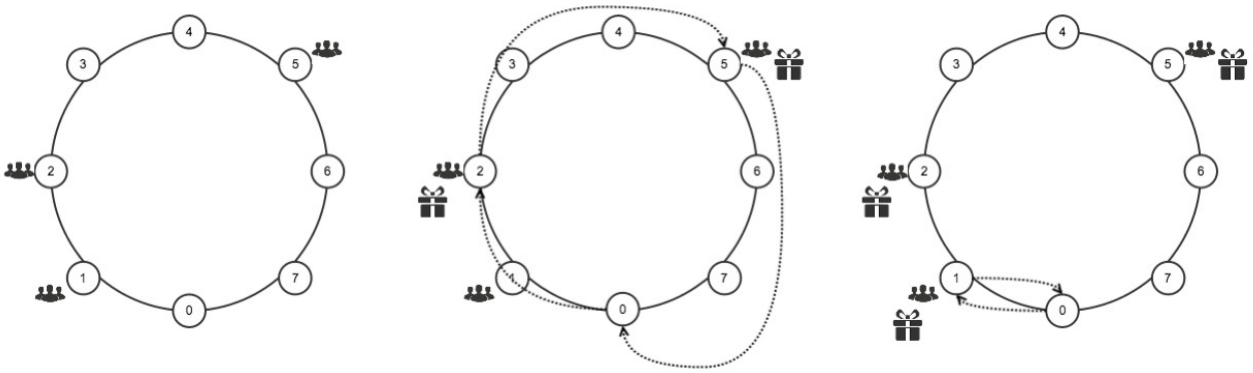
تعداد N سوغاتی کاملا مشابه وجود دارد. ابتدا امان و سوغاتی‌ها در قسمت ۰ قرار دارند. امان باید به هر تیم یک سوغاتی بدهد و بعد از تحویل آخرین سوغاتی به قسمت ۰ برگردد. دقت کنید که برخی تیم‌ها ممکن است در قسمت ۰ مستقر شده باشند.

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.

ملاحظات پیاده‌سازی

- شما باید دقیقاً یک فایل ارسال کنید.
- این فایل باید شامل پیاده‌سازی زیربرنامه‌هایی باشد که در صورت سوال توضیح داده شده است.
- رفتار این زیربرنامه‌ها باید مطابق آن چیزی باشد که در صورت سوال توضیح داده شده است.
- یک پیاده‌سازی نمونه از این فایل برای شما فراهم شده است.
- پیاده‌سازی نمونه شامل ساختار صحیح توابع و دیگر قواعد لازم برای کامپایل مناسب برنامه است. (مثلاً در C++، پیاده‌سازی نمونه همه‌ی headerهای لازم را include می‌کند.)
- برنامه‌ی ارسالی شما نباید به هیچ طریقی با ورودی و خروجی استاندارد و یا هر فایل دیگری در تعامل باشد.

محدودیت‌ها

Task	Time Limit	Memory Limit
horses	1.5 second	1500 MB
sorting	1 second	1500 MB
towns	1 second	1500 MB

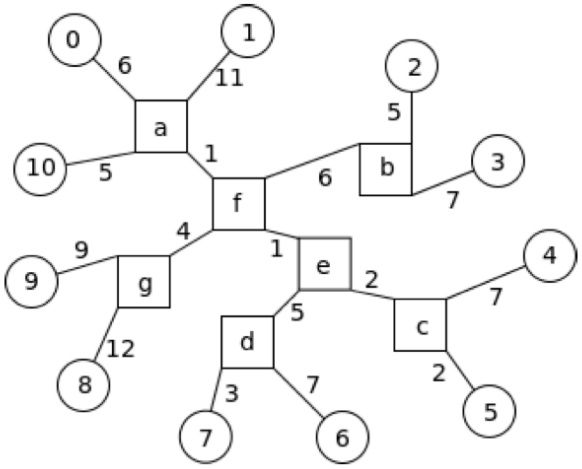
شهرها

در قزاقستان N شهر کوچک وجود دارد که از ۰ تا $N-1$ شماره‌گذاری شده‌اند. همچنین تعداد نامشخصی کلان‌شهر نیز وجود دارند. ما به شهرهای کوچک و کلان‌شهرها به طور کلی «شهر» اطلاق می‌کنیم.

تمام شهرها در قزاقستان توسط شبکه‌ای از بزرگراه‌های دوطرفه به هم متصل شده‌اند. هر بزرگراه دو شهر متمایز را به هم متصل می‌کند و هر دو شهری توسط حداکثر یک بزرگراه به یکدیگر متصل‌اند. برای هر دو شهر a و b، یک مسیر یکتا برای رسیدن از a به b وجود دارد، به شرط آن‌که از هیچ بزرگراهی در این مسیر دو بار استفاده نشود.

می‌دانیم که هر شهر کوچک به دقیقا یک شهر دیگر متصل است. همچنین هر کلان‌شهر به حداقل سه شهر متصل است.

شکل زیر شبکه‌ای شامل ۱۱ شهر کوچک و ۷ کلان‌شهر را نشان می‌دهد. شهرهای کوچک به صورت دایره با برچسب‌های عددی، و کلان‌شهرها به صورت مربع با برچسب‌های حرفی نشان داده شده‌اند.



طول هر بزرگراه یک عدد صحیح مثبت است. فاصله‌ی بین هر دو شهر برابر است با مجموع طول بزرگراه‌ها روی کوتاه‌ترین مسیری که آن دو شهر را به هم متصل می‌کند.

برای هر کلان‌شهر C، مقدار $r(C)$ را برابر با فاصله‌ی دورترین شهر کوچک از C تعریف می‌کنیم. یک کلان‌شهر C را «مرکز» می‌نامیم اگر $r(C)$ بین تمام کلان‌شهرها کم‌ترین باشد. فاصله‌ی یک مرکز تا دورترین شهر کوچک از آن را با R نشان می‌دهیم. بنابراین R کوچک‌ترین مقدار از بین تمام مقادیر $r(C)$ است.

در مثال بالا دورترین شهر کوچک از کلان‌شهر a شهر شماره‌ی ۸ است، و فاصله‌ی بین این دو شهر برابر است با $r(a) = ۱۷ = ۱ + ۴ + ۱۲$. برای کلان‌شهر g نیز $r(g) = ۱۷$ است (یکی از شهرهای کوچکی که از g دورترین است، شهر شماره‌ی ۶ است.) در مثال بالا تنها مرکز، کلان‌شهر f با مقدار $r(f) = ۱۶$ است. بنابراین در مثال بالا مقدار R برابر ۱۶ است.

با حذف یک مرکز، شبکه به چند مؤلفه‌ی همبند تقسیم می‌شود. یک مرکز را «متوازن» می‌نامیم، اگر هر یک از این مؤلفه‌ها حداکثر شامل $\lfloor \frac{N}{۲} \rfloor$ شهر کوچک باشد. (دقت کنید که ما در این تعریف فقط شهرهای کوچک را می‌شمریم.)

در مثال قبل، کلان‌شهر f یک مرکز است. اگر f را حذف کنیم، شبکه به چهار مؤلفه‌ی همبند افراز می‌شود. این چهار مؤلفه شامل مجموعه شهرهای کوچک زیر خواهند بود: $\{۰, ۱, ۱۰\}$، $\{۲, ۳\}$، $\{۴, ۵, ۶, ۷\}$، و $\{۸, ۹\}$. هیچ یک از این مؤلفه‌ها بیش‌تر از $\lfloor \frac{۱۱}{۲} \rfloor = ۵$ شهر کوچک ندارد، بنابراین f یک مرکز متوازن است.

مسئله

تنها اطلاعاتی که در آغاز در مورد شبکه‌ی شهرها و بزرگراه‌ها دارید، تعداد شهرهای کوچک یعنی N است. شما اطلاعی از تعداد کلان‌شهرها و همچنین چینش بزرگراه‌های کشور ندارید. شما تنها می‌توانید با پرس‌وجو درباره‌ی فاصله‌ی بین جفت شهرهای کوچک در مورد شبکه اطلاعات جدید به دست بیاورید.

شما قرار است موارد زیر را انجام دهید:

- در تمام زیرمسئله‌ها: تعیین مقدار R
- در زیرمسئله‌های ۳ تا ۶: تعیین این که آیا شبکه دارای یک مرکز متوازن است.

شما باید تابع `hubDistance` را پیاده‌سازی کنید. ارزیاب در هر بار اجرا تعدادی مورد آزمون را ارزیابی می‌کند. تعداد موارد آزمون در هر اجرا حداکثر ۴۰ است. برای هر مورد آزمون، ارزیاب تابع `hubDistance` را دقیقا یک‌بار فراخوانی می‌کند. مطمئن شوید که تابع شما متغیرهای موردنیاز را در هر بار فراخوانی مقداردهی اولیه می‌کند.

- `deliveryhubDistance(N, sub)`
  - `N`: تعداد شهرهای کوچک.
  - `sub`: شماره‌ی زیرمسئله (در قسمت زیرمسئله‌ها توضیح داده شده است).
  - اگر `sub` برابر ۱ یا ۲ باشد، تابع می‌تواند مقدار R یا $-R$ را برگرداند.
  - اگر `sub` بزرگ‌تر از ۲ باشد، اگر یک مرکز متوازن وجود داشت، تابع باید مقدار R و در غیر این صورت مقدار $-R$ را برگرداند.

شما می‌توانید درون تابع `hubDistance` با فراخوانی تابع ارزیاب `getDistance(i, j)` در مورد شبکه‌ی بزرگراه‌ها اطلاعات کسب کنید. این تابع فاصله‌ی بین دو شهر کوچک i و j را برمی‌گرداند. اگر i و j برابر باشند، تابع مقدار ۰ را برمی‌گرداند. همچنین وقتی که آرگومان‌ها غیرمعتبر باشند، تابع مقدار ۰ برمی‌گرداند.

زیرمسئله‌ها

در هر مورد آزمون:

- N عدد صحیحی بین ۶ و ۱۱۰ است.
- فاصله‌ی بین هر دو شهر کوچک متمایز بین ۱ و ۱,۰۰۰,۰۰۰ است.

تعداد پرس‌وجوهایی که برنامه‌ی شما می‌تواند انجام دهد محدود است. این محدودیت بسته به زیرمسئله متفاوت است و برای هر زیرمسئله در جدول زیر مشخص شده است. اگر برنامه‌ی شما از محدودیت تعداد پرس‌وجوها تجاوز کند، برنامه خاتمه یافته و فرض می‌شود که برنامه جواب غلط داده است.

زیرمسئله	امتیاز	تعداد پرس‌وجو	یافتن مرکز متوازن	محدودیت‌های دیگر
۱	۱۳	$\frac{N(N-۱)}{۲}$	خیر	ندارد
۲	۱۲	$\lceil \frac{۷N}{۲} \rceil$	خیر	ندارد
۳	۱۳	$\frac{N(N-۱)}{۲}$	بله	ندارد
۴	۱۰	$\lceil \frac{۷N}{۲} \rceil$	بله	هر کلان‌شهر دقیقا به سه شهر دیگر متصل است.
۵	۱۳	$۵N$	بله	ندارد
۶	۳۹	$\lceil \frac{۷N}{۲} \rceil$	بله	ندارد

ارزیاب نمونه

توجه کنید که شماره‌ی زیرمسئله بخشی از ورودی است. ارزیاب نمونه رفتارش را برحسب شماره‌ی زیرمسئله تغییر می‌دهد. ارزیاب نمونه ورودی را از فایل `towns.in` با قالب زیر می‌خواند:

- خط ۱: شماره‌ی زیرمسئله و تعداد موارد آزمون
- خط ۲: $N_۱$، تعداد شهرهای کوچک در اولین مورد آزمون.
- $N_۱$ خط بعد: عدد j اُم ($۱ \le j \le N_۱$) در خط i اُم از این خطها ($۱ \le i \le N_۱$) فاصله بین شهرهای کوچکِ $i-۱$ و $j-۱$ است.
- موارد آزمون بعدی در ادامه می‌آیند. آن‌ها هم به همان قالب اولین مورد آزمون داده می‌شوند.

ارزیاب نمونه برای هر مورد آزمون، مقدار خروجی تابع `hubDistance` و تعداد فراخوانی‌های انجام‌شده را در سطرهای جدا چاپ می‌کند .

فایل ورودی مربوط به مثال بالا به صورت زیر است:

```
1 1
11
0 17 18 20 17 12 20 16 23 20 11
17 0 23 25 22 17 25 21 28 25 16
18 23 0 12 21 16 24 20 27 24 17
20 25 12 0 23 18 26 22 29 26 19
17 22 21 23 0 9 21 17 26 23 16
12 17 16 18 9 0 16 12 21 18 11
20 25 24 26 21 16 0 10 29 26 19
16 21 20 22 17 12 10 0 25 22 15
23 28 27 29 26 21 29 25 0 21 22
20 25 24 26 23 18 26 22 21 0 19
11 16 17 19 16 11 19 15 22 19 0
```

این قالب با مشخص کردن لیست بزرگراه‌ها کاملاً متفاوت است. مسلماً شما می‌توانید ارزیاب نمونه را ویرایش کنید تا از قالب ورودی دیگری استفاده کند.

مرتب‌سازی

«آیژان» یک دنباله شامل N عدد صحیح $S[۰]$، $S[۱]$، ...، $S[N-۱]$ در اختیار دارد. این دنباله شامل اعداد متمایز از ۰ تا $N-۱$ است. او می‌خواهد این دنباله را با استفاده از جابه‌جایی جفت عناصر دنباله (swap) به ترتیب صعودی مرتب کند. دوستش «اِرمِک» نیز قصد دارد برخی از جفت عناصر دنباله را (نه لزوما در راستای کمک به آیژان) جابه‌جا کند.

ارمک و آیژان قصد دارند دنباله را در چندین مرحله تغییر دهند. در هر مرحله، ابتدا ارمک یک جابه‌جایی انجام می‌دهد و سپس آیژان یک جابه‌جایی دیگر انجام می‌دهد. به طور دقیق‌تر، شخصی که جابه‌جایی را انجام می‌دهد، دو اندیس معتبر انتخاب می‌کند و عناصر قرار گرفته در آن دو اندیس را جابه‌جا می‌کند. توجه کنید که این دو اندیس لزوما متمایز نیستند. در صورت برابری اندیس‌ها، یک عنصر با خودش جابه‌جا می‌شود و این کار دنباله را تغییر نمی‌دهد.

آیژان می‌داند که مرتب‌سازی دنباله‌ی S برای ارمک مهم نیست. او همچنین اندیس‌های انتخابی ارمک را از قبل می‌داند. ارمک قصد دارد در M مرحله شرکت کند. این مراحل را از ۰ تا $M-۱$ شماره‌گذاری می‌کنیم. به ازای هر i بین ۰ تا $M-۱$ (شامل هر دو)، ارمک اندیس‌های $X[i]$ و $Y[i]$ را در مرحله‌ی i انتخاب خواهد کرد.

آیژان قصد دارد که دنباله‌ی S را مرتب کند. قبل از هر مرحله، اگر آیژان متوجه شود که دنباله به صورت صعودی مرتب شده است، او به فرآیند مرتب‌سازی خاتمه خواهد داد. با فرض این‌که دنباله‌ی S و اندیس‌هایی که ارمک قصد انتخابشان را دارد به شما داده می‌شود، وظیفه‌ی شما پیدا کردن دنباله‌ای از جابه‌جایی‌ها است که آیژان می‌تواند با استفاده از آن‌ها دنباله‌ی S را مرتب کند. علاوه بر این، در برخی از زیرمسئله‌ها شما باید کوتاهترین دنباله‌ی جابه‌جایی‌ها را پیدا کنید. شما می‌توانید فرض کنید که دنباله‌ی S با M مرحله یا کمتر، قابل مرتب‌سازی است.

توجه کنید که اگر آیژان متوجه شود که بعد از جابه‌جایی ارمک دنباله‌ی S مرتب شده است، او می‌تواند دو اندیس یکسان را جابه‌جا کند (برای نمونه، ۰ و ۰). در نتیجه دنباله‌ی S بعد از پایان این مرحله مرتب شده است و در نتیجه آیژان به هدفش می‌رسد. هم‌چنین توجه کنید در صورتی که دنباله‌ی S در ابتدا مرتب باشد، کمترین تعداد مرحله‌ی مورد نیاز ۰ خواهد بود.

مثال ۱

فرض کنید:

- دنباله‌ی اولیه $S = ۴, ۳, ۲, ۱, ۰$ است.
- ارمک تمایل دارد $M = ۶$ جابه‌جایی انجام دهد.
- دنباله‌های X و Y ای که اندیس‌های انتخابی ارمک را نشان می‌دهند عبارتند از: $X = ۰, ۱, ۲, ۳, ۰, ۱$ و $Y = ۱, ۲, ۳, ۴, ۱, ۲$. به عبارت دیگر، جفت اندیس‌هایی که ارمک قصد انتخاب آن‌ها را دارد عبارتند از (۰, ۱)، (۱, ۲)، (۲, ۳)، (۳, ۴)، (۰, ۱) و (۱, ۲).

در این سناریو آیژان می‌تواند دنباله‌ی S را در سه مرحله به دنباله $۰, ۱, ۲, ۳, ۴$ تبدیل کند. او می‌تواند این کار با انتخاب اندیس‌های (۰, ۴)، (۱, ۳) و سپس (۳, ۴) انجام دهد.

جدول زیر نشان می‌دهد که ارمک و آیژان چگونه دنباله را تغییر داده‌اند.

مرحله	بازیکن	جابه‌جایی	دنباله (از چپ به راست)
شروع			۳, ۰, ۴, ۲, ۱
۰	ارمک	(۱, ۱)	۳, ۰, ۴, ۲, ۱
۰	آیژان	(۱, ۴)	۳, ۱, ۴, ۲, ۰
۱	ارمک	(۴, ۰)	۰, ۱, ۴, ۲, ۳
۱	آیژان	(۴, ۲)	۰, ۱, ۳, ۲, ۴
۲	ارمک	(۲, ۳)	۰, ۱, ۲, ۳, ۴
۲	آیژان	(۲, ۲)	۰, ۱, ۲, ۳, ۴

مسئله

دنباله‌ی S، عدد M، و دنباله‌های X و Y به شما داده شده است. دنباله‌ای از جابه‌جایی‌ها را پیدا کنید که به کمک آن‌ها آیژان می‌تواند دنباله‌ی S را مرتب کند. در زیرمسئله‌های ۵ و ۶ شما باید کوتاه‌ترین دنباله‌ی ممکن از جابه‌جایی‌ها را پیدا کنید.

شما باید تابع `findSwapPairs` را پیاده‌سازی کنید:

- `findSwapPairs(N, S, M, X, Y, P, Q)` — این تابع دقیقا یک بار از طرف ارزیاب فراخوانی می‌شود.

- N: طول دنباله‌ی S
- S: آرایه‌ای از اعداد صحیح شامل دنباله‌ی اولیه‌ی S
- M: تعداد جابه‌جاهایی که ارمک قصد دارد انجام دهد.
- X، Y: آرایه‌هایی به طول M از اعداد صحیح. به ازای هر $0 \leq i \leq M - 1$، ارمک قصد دارد در مرحله‌ی i اندیس‌های $X[i]$ و $Y[i]$ را جابه‌جا کند.
- P، Q: آرایه‌ای از اعداد صحیح. از این آرایه‌ها، برای گزارش دنباله‌ی جابه‌جایی‌هایی که آیژان توسط آن‌ها می‌تواند دنباله‌ی S را مرتب کند استفاده کنید. مقدار R را برابر با طول دنباله‌ی جابه‌جایی‌هایی که برنامه‌ی شما پیدا کرده است، در نظر بگیرید. برای هر i بین 0 تا $R - 1$ (شامل هر دو)، اندیس‌هایی که آیژان در مرحله‌ی i انتخاب می‌کند باید در $P[i]$ و $Q[i]$ ذخیره شوند. شما می‌توانید فرض کنید که برای هر یک از آرایه‌های P و Q، تعداد M عنصر در حافظه اختصاص داده شده است.
- این تابع باید مقدار R (که در بالا تعریف شده) را به عنوان خروجی برگرداند.

زیرمسئله‌ها

زیرمسئله	امتیاز	N	M	محدودیت‌های دیگر روی X و Y	محدودیت روی R
1	8	$1 \leq N \leq 5$	$M = N^2$	$X[i] = Y[i] = 0$ for all i	$R \leq M$
2	12	$1 \leq N \leq 100$	$M = 30N$	$X[i] = Y[i] = 0$ for all i	$R \leq M$
3	16	$1 \leq N \leq 100$	$M = 30N$	$X[i] = 0, Y[i] = 1$ for all i	$R \leq M$
4	18	$1 \leq N \leq 500$	$M = 30N$	none	$R \leq M$
5	20	$6 \leq N \leq 2,000$	$M = 3N$	none	minimum possible
6	26	$6 \leq N \leq 200,000$	$M = 3N$	none	minimum possible

شما می‌توانید فرض کنید که جوابی با M مرحله و یا کمتر وجود دارد.

ارزیاب نمونه

ارزیاب نمونه ورودی را از فایل sorting.in با فرمتی که در زیر آمده است، می‌خواند:

- سطر ۱: N
- سطر ۲: S[0] تا S[N-1]
- سطر ۳: M
- سطر ۴ تا $M + 3$: X[i] سپس Y[i]

ارزیاب نمونه خروجی‌های زیر را چاپ می‌کند:

- سطر ۱: مقدار R، خروجی تابع findSwapPairs
- سطر $i + 2$، برای $0 \leq i < R$: P[i] سپس Q[i]

اسب‌ها

منصور مانند نیاکانش عاشق پرورش اسب است. او در حال حاضر مالک بزرگ‌ترین گله‌ی اسب در قزاقستان است. ولی همیشه این طور نبوده است. N سال پیش، منصور جوانی بود با تنها یک اسب و آرزو داشت که روزی پولدار شود.

فرض کنید سال‌ها را به ترتیب زمانی از ۰ تا $N-1$ شماره‌گذاری می‌کنیم (یعنی سال $N-1$ آخرین سال است). آب و هوای سال‌های مختلف بر میزان رشد گله اثر می‌گذارد. برای هر سال i، منصور عدد صحیح مثبت $X[i]$ را به عنوان ضریب رشد آن سال به یاد می‌آورد. این به این معنی است که اگر در ابتدای سال i گله شامل h اسب باشد، در انتهای این سال $h \times X[i]$ اسب در گله خواهد بود.

فروش اسب‌ها تنها در پایان سال ممکن است. برای هر سال i، منصور یک عدد صحیح مثبت $Y[i]$ را به عنوان قیمت فروش اسب در انتهای سال i به یاد می‌آورد. این به این معنی است که در پایان سال i، منصور می‌توانسته است هر تعداد اسبی را به مبلغ $Y[i]$ به ازای هر اسب بفروشد.

منصور دوست دارد بداند که اگر در طی این N سال اسب‌هایش را در بهترین زمان‌های ممکن فروخته بود، بیشترین مقدار پولی که می‌توانست داشته باشد چه‌قدر است. شما این افتخار را داشته‌اید که در طی تعطیلات مهمان منصور باشید، و او از شما خواسته است که به این سؤال پاسخ دهید.

هر چه از شب می‌گذرد، حافظه‌ی منصور بهتر می‌شود و او M بار خاطراتش را به‌روزرسانی می‌کند. هر به‌روزرسانی مقدار یکی از $X[i]$ها یا یکی از $Y[i]$ها را تغییر می‌دهد. پس از هر به‌روزرسانی، منصور دوباره از شما می‌پرسد که بیشترین مقدار پولی که می‌توانست داشته باشد چه‌قدر است. این به‌روزرسانی‌ها «افزاینده» هستند، یعنی هر بار که می‌خواهید به سؤال جواب دهید، باید همه‌ی به‌روزرسانی‌های قبلی را اعمال کرده باشید. توجه کنید که مقدار یک $X[i]$ یا یک $Y[i]$ ممکن است بیش از یک بار تغییر کند.

جواب درست سؤال منصور ممکن است عدد بسیار بزرگی باشد. برای این که لازم نباشد با عددهای بزرگ کار کنید، از شما خواسته شده است که جواب را تنها در پیمانه‌ی $۱۰^۹ + ۷$ محاسبه کنید.

مثال

فرض کنید $N = ۳$ سال با اطلاعات زیر داریم:

سال	۰	۱	۲
X	۲	۱	۳
Y	۳	۴	۱

برای مقادیر اولیه‌ی فوق، منصور بیشترین مقدار پول را با فروختن هر دو اسبش در پایان سال ۱ به دست می‌آورد. کل فرایند به صورت زیر است:

- در ابتدا، منصور یک اسب دارد.

- پس از پایان سال ۰، او ۲ $= [۰]X \times$ ۱ اسب خواهد داشت.
- پس از پایان سال ۱، او ۲ $= [۱]X \times$ ۲ اسب خواهد داشت.
- او می‌تواند هر دوی این اسب‌ها را در پایان سال ۱ بفروشد. میزان درآمد او ۸ $= [۱]Y \times$ ۲ خواهد بود.

حالا فرض کنید که $M =$ ۱ به‌روزرسانی داریم: مقدار $Y[۱]$ باید به ۲ تغییر کند. پس از این به‌روزرسانی، مقادیر X و Y به این صورت هستند:

سال	۰	۱	۲
X	۲	۱	۳
Y	۳	۲	۱

پس از این تغییر، یک جواب بهینه این است که ۱ اسب را در پایان سال ۰ و پس از آن ۳ اسب را در پایان سال ۲ بفروشیم. کل فرایند به صورت زیر خواهد بود:

- در ابتدا، منصور یک اسب دارد.
- پس از پایان سال ۰، او ۲ $= [۰]X \times$ ۱ اسب خواهد داشت.
- او می‌تواند یکی از این دو اسب را به قیمت ۳ $= [۰]Y$ بفروشد و یک اسب را نگه دارد.
- پس از پایان سال ۱، او ۱ $= [۱]X \times$ ۱ اسب خواهد داشت.
- پس از پایان سال ۲، او ۳ $= [۲]X \times$ ۱ اسب خواهد داشت.
- او می‌تواند این سه اسب را در پایان سال ۲ به قیمت ۳ $= [۲]Y \times$ ۳ بفروشد. میزان کل درآمد او ۶ $=$ ۳ $+$ ۳ خواهد بود.

مسئله

مقادیر N، X، Y و لیست به‌روزرسانی‌ها به شما داده شده است. قبل از اولین به‌روزرسانی و پس از هر یک از به‌روزرسانی‌ها، بیشترین مقدار پولی که منصور می‌توانست از فروش اسب‌هایش به دست آورد را در پیمانه‌ی ۷ $+$ ۱۰^۹ محاسبه کنید. شما باید سه تابع `init`، `updateX` و `updateY` به شرح زیر را پیاده‌سازی کنید:

- `init(N, X, Y)` — ارزیاب این تابع را در ابتدا فقط یک بار فراخوانی می‌کند.
  - `N`: تعداد سال‌ها
  - `X`: آرایه‌ای به طول N. برای هر ۰ $\le i \le N -$ ۱، $X[i]$ ضریب رشد گله در سال i است.
  - `Y`: آرایه‌ای به طول N. برای هر ۰ $\le i \le N -$ ۱، $Y[i]$ قیمت یک اسب در پایان سال i است.
  - دقت کنید که `X` و `Y` مقادیر اولیه‌ای که منصور به شما داده (قبل از هر گونه به‌روزرسانی) هستند.
  - پس از خاتمه‌ی تابع `init`، آرایه‌های `X` و `Y` معتبر باقی می‌مانند و شما اگر بخواهید می‌توانید مقدارشان را تغییر دهید.
  - این تابع باید بیشترین مقدار درآمد ممکن برای منصور به ازای این مقادیر اولیه را در پیمانه‌ی ۷ $+$ ۱۰^۹ برگرداند.

- `updateX(pos, val)`
  - `pos`: عددی صحیح در محدوده‌ی ۰ تا $N-1$.
  - `val`: مقدار جدید $X[\texttt{pos}]$.
  - این تابع باید بیشترین مقدار درآمد ممکن برای منصور پس از این به‌روزرسانی را در پیمانه‌ی 10^9+7 برگرداند.
- `updateY(pos, val)`
  - `pos`: عددی صحیح در محدوده‌ی ۰ تا $N-1$.
  - `val`: مقدار جدید $Y[\texttt{pos}]$.
  - این تابع باید بیشترین مقدار درآمد ممکن برای منصور پس از این به‌روزرسانی را در پیمانه‌ی 10^9+7 برگرداند.

می‌توانید فرض کنید که مقادیر اولیه، و همچنین مقادیر به‌روزرسانی شده‌ی $X[i]$ و $Y[i]$، همگی بین ۱ و 10^9 هستند. پس از فراخوانی تابع `init`، ارزیاب توابع `updateX` و `updateY` را چندین بار فراخوانی می‌کند. تعداد کل فراخوانی‌های این دو تابع M است.

زیرمسئله‌ها

زیرمسئله	امتیاز	N	M	محدودیت‌های دیگر
1	17	$1 \le N \le 10$	$M = 0$	$X[i], Y[i] \le 10$, $X[0] \times X[1] \times \cdots \times X[N-1] \le 1,000$
2	17	$1 \le N \le 1,000$	$0 \le M \le 1,000$	none
3	20	$1 \le N \le 500,000$	$0 \le M \le 100,000$	$X[i] \ge 2$ and `val` ≥ 2 for `init` and `updateX` correspondingly
4	23	$1 \le N \le 500,000$	$0 \le M \le 10,000$	none
5	23	$1 \le N \le 500,000$	$0 \le M \le 100,000$	none

ارزیاب نمونه

ارزیاب نمونه ورودی را از فایل `horses.in` در قالب زیر می‌خواند:

- خط ۱: `N`
- خط ۲: `X[0]` تا `X[N - 1]`
- خط ۳: `Y[0]` تا `Y[N - 1]`
- خط ۴: `M`
- خطوط ۵ تا $M+4$: سه عدد `type`، `pos` و سپس `val` (`type=1` به معنای فراخوانی `updateX` و `type=2` به معنای `updateY` است).

ارزیاب نمونه مقدار خروجی تابع `init` و پس از آن مقادیر خروجی توابع `updateX` و `updateY` را چاپ می‌کند.