عملیات ساده

- محدودیت زمان: ۴ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

در این سوال سعی داریم یکسری از عملیات های پرتکرار و نسبتا ساده که در داده ساختاری هایی مثل ArrayList و HashMap بکار میروند را پیاده سازی کنیم.

ArrayList

از معروف ترین لیست های مورد استفاده میباشد. در واقع یک ارایه با قابلیت تغییر سایز هست که در پکیج **java.util** پیاده سازی شده است. در این تمرین شما یکسری از توابع معروف آنرا استفاده میکنید و روی ورودی های داده شده اعمال میکنید.

در این تمرین اعضای **ArrayList** را از نوع **Integer** میگیریم.

HashMap

بر خلاف لیست ها که به اعضا توسط یک عدد ایندکس دسترسی داریم، در HashMap اعضا به شکل دوتایی های **(key/value)** ذخیره میشوند و میتوانیم از یک آبجکت و نه لزوما عدد، به عنوان یک کلید برای دسترسی به عضو دیگر استفاده کنیم.

در **HashMap** این تمرین، اعضا به شکل دوتایی هایی از **Integer** (کلید) و **String** (مقدار) هستند. (که میتوان برای مثال بشکل شماره دانشجویی و اسم شخص به آن ها نگاه کرد) و هر دو تایی (که میان کلید و مقدار فاصله وجود دارد) در یک خط جداگانه به عنوان ورودی داده میشوند.

خط اول ورودی در هر تست به شکل

```
collectionName inputSize command
```

میباشد که در آن **collectionName** یکی از ArrayList ویا HashMap میباشد. همچنین **inputSize** بیانگر اندازه سایز لیست ویا هش مپی است که قرار است در خط های بعدی ورودی های متناظرشان دریافت شود. **command** نیز بیانگر نوع دستوری هست که میخواهیم بر روی کالکشن مورد نظر اعمال کنیم. در خط های بعدی اعضای لیست و یا هش مپ مورد نظر داده میشود. و شما باید کامند مورد نظر را روی آن لیست اعمال کنید.

کامند های (دستورات) مربوط به ArrayListبه شرح زیر هستند:

- **اضافه کردن به لیست**: که در کد با **insert** متناظر شده و شما دو عدد را در ورودی دریافت میکنید. یکی ایندکس و یکی عددی که باید در آن ایندکس به لیست اضافه کنید و سپس لیست را چاپ کنید.
- **بدست آوردن یک عضو موجود در لیست**: که کامند متناظر با آن در ورودی **retrieve** هست و باید از وروردی یک عدد که نشان دهنده ایندکس مورد نظر هست را بگیرید و سپس عدد مورد نظر را از لیست چاپ کنید.
- **تغییر یک عضو**: کامند آن **modify** است و باید یک ایندکس و یک عدد بگیرید و عدد را در آن ایندکس قرار دهید و بعد لیست را چاپ کنید.
- **حذف عضو**: کامند **remove** یک ایندکس را از ورودی بگیرید و آنرا از لیست حذف میکند و بعد لیست را چاپ میکند.
- **بررسی وجود یک عضو**: کامند **contains** یک عدد از ورودی میگرد و چک میکند اگر در لیست وجود داشت، در یک خط خروجی "y" وگرنه "n" را چاپ کند.
- **مرتب کردن اعضای لیست از کوچک به بزرگ**: کامند **sort** لیست را مرتب کرده و سپس چاپ میکند.
- **معکوس کردن ترتیب اعضای لیست**: کامند **reverse** لیست را برعکس کرده و بعد آنرا چاپ میکند.
- **گرفتن ایندکس متناظر با یک عدد**: همان **getIndex** که یک عدد از ورودی میگیرد و ایندکس آنرا در لیست در خروجی چاپ میکند.
- **گرفتن سایز لیست**: کامند **getSize** که صرفا سایز را در خروجی چاپ میکند.
- **جا به جایی دو عضو**: کامند **swap** دو ایندکس را از روردی میگیرد و اعداد متناظر در آن دو ایندکس را با هم جا به جا میکند و سپس لیست را چاپ میکند.

- **تقسیم کردن به دو لیست و بعد انجام تغییرات روی یکی و بعد دوباره یکی کردن دو لیست** : کامند **extractReverseRejoin** یک ایندکس **i** از ورودی میگیرد و بعد **i** عضو اول را در یک لیست و بقیه را در یک لیست دیگر میریزد. سپس اعضای لیست اول را معکوس و بعد دو لیست را در یک لیست ریخته و همه اعضا را چاپ میکند.

در هر یک از کامند های بالا که نیاز به چاپ آرایه بود، باید اعضا را کنار هم از ایندکس صفر تا اخر و با یک فاصله (اسپیس) از هم چاپ کنید.

کامند های (دستورات) مربوط به HashMap:

- **چاپ اعضا**: یا همان کامند **print** که باید اعضا را بشکل زیر چاپ کنید.
(key1,value1) (key2,value2) (key3,value3) ... یعنی یک ویرگول بین اعضای دوتایی ها و یک فاصله بین هر دو دوتایی کنار هم.
- **بررسی وجود یک کلید**: کامند **containsKey** یک کلید از ورودی میگرد و چک میکند اگر در کلید ها وجود داشت، در یک خط خروجی "y" وگرنه "n" را چاپ کند.
- **گرفتن مقدار متناظر با یک کلید**: کامند **getValue** یک کلید از ورودی میگیرد و مقدار متناظر با آنرا در خروچی چاپ میکند.
- **حذف دوتایی**: کامند **remove** یک کلید از ورودی میگیرد و دوتایی متناظر با آنرا از هش مپ حذف میکند و بعد هش مپ را چاپ میکند.
- **مرتب سازی بر اساس کلید ها** کامند **sortByKeys** هش مپ مورد نظر را طبق کلید ها، از کوچک به بزرگ مرتب و سپس هش مپ را چاپ میکند.
- **مرتب سازی بر اساس مقادیر** کامند **sortByValues** هش مپ مورد نظر را طبق مقادیر و یا همان مولفه های دوم، از کوچک به بزرگ مرتب و سپس هش مپ را چاپ میکند.

در مرتب سازی با توجه به اینکه HashMap ترتیب خاصی را نگه نمیدارد، باید بعد از بدست اوردن ترتیب درست سورت، یا همانجا اعضا را چاپ کنید و یا از LinkedHashMap ویا TreeMap استفاده کنید.

ورودی

همانطور که اشاره شد، خط ابتدایی ورودی به شکل زیر است

```
collectionName inputSize command
```

و در خط های بعدی اعضای لیست و یا هش مپ میایند و سپس ورودی های مربوط به کامند موردنظر (اگر نیاز به ورودی بود)

خروجی

در خروجی با توجه به نوع کامند یا باید لیست یا هش مپ موردنظر را باتوجه به فرمت گفته شده چاپ کنید و یا خروجی خاصی مثل یک عدد یا موارد مشابه را چاپ کنید.

مثال

ورودی نمونه ۱

```
ArrayList 6 swap
7 3 8 5 9 2
2 3
```

خروجی نمونه ۱

```
7 3 5 8 9 2
```

ورودی نمونه ۲

```
ArrayList 6 extractReverseRejoin
7 5 8 1 2 9
2
```

خروجی نمونه ۲

```
5 7 8 1 2 9
```

ورودی نمونه ۳

```
HashMap 6 sortByValues
3 a
4 c
2 v
7 b
1 w
6 q
```

خروجی نمونه ۳

```
(3,a) (7,b) (4,c) (6,q) (2,v) (1,w)
```

ورودی نمونه ۴

```
HashMap 5 remove
1 a
3 s
2 v
7 w
6 t
2
```

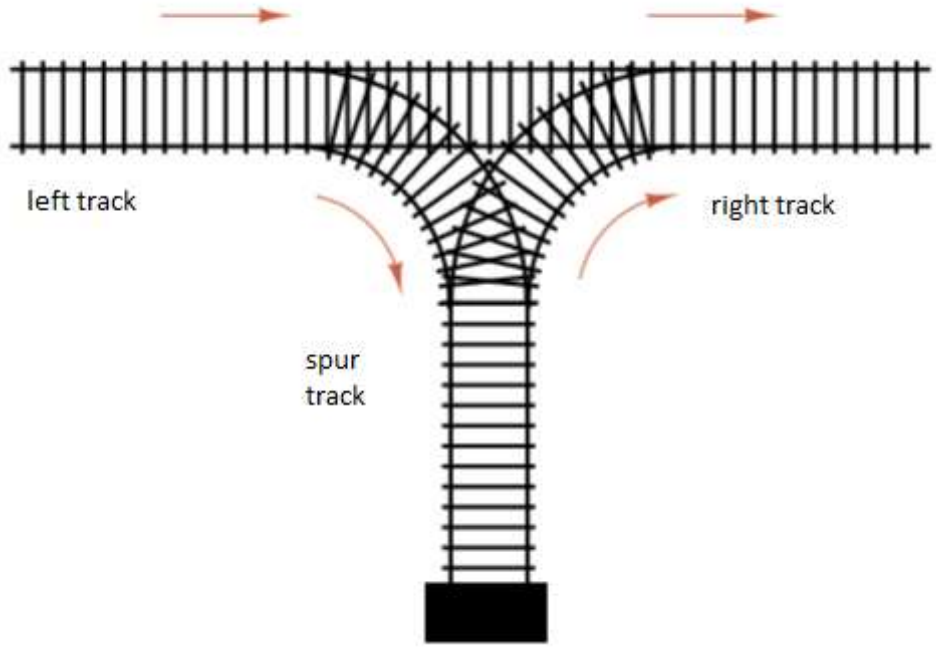
خروجی نمونه ۴

```
(1,a) (3,s) (6,t) (7,w)
```

تری ریل

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

بچه های درس برنامه نویسی پیشرفته علاقه زیادی به شهربازی داشتند و روزی تصمیم میگیرند به پارک ارم بروند. از آنجایی که یکی از بچه ها علاقه زیادی به هیجان افراطی نداشت، تصمیم میگیرند سوار مونو ریل شوند که متاسفانه یا خوشبختانه چون ایمان با آن ها بود، به پیشنهاد او بازی تری ریل را امتحان می کنند. بازی به صورت زیر می باشد و هر کدام از بچه ها باید سوار یک گاری که روی ریل سوار است بشوند به طوری که نفر اول در سمت چپ بقیه قرار دارد و در مجموع n گاری داریم که هرکدام سوار یکی می شوند و آن را هدایت می کنند در ابتدا ایمان نقشه ریل که به صورت استک است و قوانین را به بچه ها توضیح می دهد:



هدف بازی این است که دوستان گرامی گاری ها را با جایگشت P در ریل سمت راست قرار بدهند و تا زمانی

که بتوانند این کار را بکنند در هر مرحله می توانند یکی از کار های زیر را انجام دهند:

۱. آخرین راننده گاری ریل سمت چپ، گاری اش را وارد ریل وسط کند.
۲. آخرین راننده گاری ریل سمت چپ، گاری اش را وارد ریل سمت راست کند.
۳. بالاترین راننده ای که در ریل وسط است گاری را وارد ریل سمت راست کند.

برای مثال، اگر ۳ گاری روی ریل سمت چپ قرار داشته باشند، پس از دنباله‌ی عملیات $1, 2, 1^*, 3^*, 3$ (از چپ به راست) به جایگشت $3\ 1\ 2$ می‌رسد که 2 اولین گاریست که وارد ریل سمت راست می‌شود. (البته دقت کنید که این مثال بهینه نیست، چرا که می‌توان دنباله‌ی $1^*, 3^*$ را با 2 جایگزین کرد و دنباله‌ی بهینه‌ی $1, 2, 2, 3$ را بدست آورد.)

یا مثلا با دنباله‌ی عملیات $2, 2, 2$ به جایگشت $1\ 2\ 3$ می‌رسیم. ولی با هیچ دنباله‌ای از این اعمال نمی‌توان به جایگشت $2\ 3\ 1$ رسید. در این صورت باید عدد -1 فقط در یک خط در خروجی گزارش شود.

حالا به دوستان عزیز کمک کنید تا برای به دست آوردن گاری ها با جایگشت P در ریل سمت راست، در صورت امکان ***کوتاه‌ترین*** دنباله‌ی ممکن از ۳ عملیات بالا را برای رسیدن به این جایگشت را اعلام کنند و خروجی دهند.

ورودی

در خط اول تعداد گاری ها به شما داده می شوند

$$1 \leq n \leq 100000$$

در خط دوم نیز n عدد که جایگشت مطلوب، P، را توصیف می‌کنند.

خروجی

در اولین خط از خروجی، ***کمترین تعداد*** عملیات مورد نیاز (m) برای رسیدن به جایگشت هدف را چاپ کنید.

سپس در خط بعدی، m عدد با فاصله چاپ کنید که هر کدام بین 1 تا 3 و متناظر با ۳ عملیاتی هستند که در این سوال مطرح شده‌اند. از آنجا که باید این کار را با کم‌ترین تعداد عملیات ممکن انجام دهید، به صورت حریصانه عمل کنید و اگر چند دنباله با m عملیات برای رسیدن به جایگشت هدف وجود داشت، می‌توانید هرکدام از آن‌ها را به دلخواه چاپ کنید.

مثال

ورودی نمونه ۱

```
3

3 1 2
```

خروجی نمونه ۱

```
4

1 2 2 3
```

در این مثال، راننده گاری شماره 2 اولین نفری است که گاری را به ریل سمت راست وارد می‌کند. برای این منظور ابتدا عمل شماره ۱ را انجام می‌دهیم تا گاری شماره 3 از ریل سمت چپ به ریل وسط منتقل شود. سپس عمل شماره ۲ را انجام می‌دهیم تا گاری شماره 2 به ریل سمت راست منتقل شود. بعد همین حرکت را با گاری شماره 1 نیز انجام می‌دهیم و در نهایت با انجام عمل ۳ ، گاری شماره 3 به ریل سمت راست منتقل می‌شود.

ورودی نمونه ۲

```
4
```

```
1 3 4 2
```

خروجی نمونه ۲

```
-1
```

ماتریس تنک

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

در بسیاری از مسئله های مربوط به داده ساختارها نیاز داریم تا با ماتریس هایی با ابعاد بسیار بزرگ (مثلا یک میلیون در یک میلیون) کار کنیم که می دانیم مقدار بیشتر خانه های آن برابر صفر است. مثلا یک شبکه ی اجتماعی شامل n کاربر را در نظر بگیرید در آن میزان ارتباط دو کاربر با یک عدد صحیح بیان می شود (عدد صفر نشان دهنده ی ارتباط نداشتن دو کاربر است) . برای نگه داری این داده ها نیاز داریم تا یک ماتریس با ابعاد n در n را ذخیره کنیم که در سطر i ام و ستون j ام آن میزان ارتباط دو کاربر i و j قرار دارد. برای یک شبکه ی اجتماعی نسبتا پر مخاطب، n یک عدد بسیار بزرگ بوده و به وضوح بیشتر خانه های این ماتریس برابر صفر هستند، زیرا هر کاربر احتمالا با تعداد محدودی از کل کاربران شبکه ارتباط دارد. اگر هر کاربر با یک درصد از کل کاربران دیگر نیز ارتباط داشته باشد 99 درصد خانه های این ماتریس برابر صفر هستند. به چنین ماتریس هایی ماتریس تنک میگوییم. ذخیره سازی ماتریس های تنک با ابعاد بزرگ به صورت آرایه n در n ممکن نیست، زیرا هر خانه از آرایه حتی اگر مقداری نداشته باشد در حافظه فضا اشغال می کند که یعنی به n^2 واحد حافظه نیاز خواهیم داشت که در دسترس نیست.

برای اینکه بتوانیم ماتریس های تنک را در حافظه ذخیره کنیم داده ساختاری طراحی میکنیم که در آن صرفا خانه هایی از ماتریس ذخیره شوند که مقدار غیر صفر دارند. برای این منظور، کل ماتریس را با یک لیست مدل سازی میکنیم. اعضای این لیست، خودشان نیز لیست هستند و بیانگر سطرهای ماتریس اصلی. به عبارتی ماتریس را با یک لیست شامل n لیست دیگر نمایش می دهیم که هر کدام مقادیر غیر صفر یکی از سطرهای ماتریس اصلی (همراه با اندیسشان در این سطر) را در خود دارند. اعضای این لیست ها به صورت دوتایی های (a, b) هستند که a بیانگر شماره ی ستون و b بیانگر مقدار ذخیر شده در این خانه است. پس اگر در لیست i ام زوج (a, b) را داشته باشیم به این معنی است که در خانه ی (i, a) در ماتریس اصلی مقدار b قرار دارد.

```
matrix :=
[
[(a1,b1),(a2,b2),...,(ak,bk)],   \\ first row
[(c1,d1),(c2,d2),...,(cl,dl)],   \\ second row
...
]
```

در این تمرین میخواهیم ماتریس تنک را با `LinkedList` پیاده‌سازی کنیم و عمل جمع بر روی دو ماتریس تنک را انجام دهیم.

پیاده سازی

ابتدا یک کلاس `Tuple` به صورت زیر پیاده‌سازی کنید :

```
class Tuple{
int a,b;
}
```

وظیفه‌ی این کلاس نگه‌داری از یک زوج از اعداد صحیح می‌باشد.سپس داده‌های ماتریس تنک را به صورت زیر پیاده‌سازی کنید

```
data: LinkedList<LinkedList<Tuple>>
```

برای ساده‌تر شدن کارتان می‌توانید از شی‌گرایی برای پیاده‌سازی ماتریس تنک استفاده کنید. مثلا یک کلاس برای ماتریس تنک داشته باشید که در آن فیلدها و متدهای مربوط به آن را پیاده‌سازی کنید.

ورودی

ورودی کد شما، دو ماتریس تنک با ابعاد یکسان خواهد بود و در خروجی می‌بایست جمع دو ماتریس را چاپ کنید.

در خط اول ورودی، ابعاد دو ماتریس به صورت دو عدد صحیح m, n با فاصله داده می‌شوند که m برابر تعداد سطرها و n برابر تعداد ستون‌ها است. سپس عدد صحیح $t1$ که برابر تعداد خانه‌های غیر صفر ماتریس اول است داده می‌شود و در $t1$ خط بعدی ورودی، در هر خط سه عدد i, j, k با فاصله که به ترتیب بیانگر شماره‌ی سطر، شماره‌ی ستون و مقدار در ماتریس هستند داده می‌شوند:

$$0 \leq i \leq m-1, 0 \leq j \leq n-1$$

سپس عدد صحیح $t2$ که برابر تعداد خانه‌های غیر صفر ماتریس دوم است داده می‌شود و در $t2$ خط بعدی ورودی، در هر خط سه عدد i, j, k با فاصله که به ترتیب بیانگر شماره‌ی سطر، شماره‌ی ستون و مقدار در ماتریس هستند داده می‌شوند:

$$0 \leq i \leq m-1, 0 \leq j \leq n-1$$

خروجی

در خروجی ماتریس تنک حاصل جمع دو ماتریس داده شده را چاپ کنید.

```
[[(a1,b1), (a2,b2), ..,(ak,bk)], [(c1,d1), ..,(cl,dl)], ..]
```

مثال

ورودی نمونه 1:

```
3 2
4
0 1 1
1 0 3
1 1 4
2 1 1
4
0 0 2
```

```
0 1 2
1 0 1
2 1 3
```

خروجی نمونه 1:

```
[[(0,2), (1,3)], [(0,4), (1,4)], [(1,4)]]
```

ورودی نمونه 2:

```
3 3
4
0 2 1
1 0 2
2 0 1
2 1 2
3
0 0 1
2 0 3
2 2 1
```

خروجی نمونه 2:

```
[[(0,1), (2,1)], [(0,2)], [(0,4), (1,2), (2,1)]]
```

نکات

۱. تضمین می‌شود که مقادیر ماتریس اعداد صحیح و نامنفی هستند.

۲. زوج‌های مرتبی که برای هر سطر ماتریس نگه‌داری می‌شوند باید بر حسب مولفه‌ی اول (شماره‌ی ستون) مرتب شده باشند. یعنی اگر

$$[(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)]$$

مقادیر یکی از سطرها باشند باید داشته باشیم

$$a_1 < a_2 < \cdots < a_k$$

سطرهای خالی را نیز با لیست خالی نمایش دهید.

۳. برای ساده‌تر شدن چاپ خروجی می‌توانید از متد `toString` برای کلاس `LinkedList` استفاده کنید. همچنین همین تابع را برای کلاس `Tuple` نیز اورراید کنید.

ماتریس‌های بی‌دردسر!

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

در این سوال قرار است 4 عملیات ماتریسی ساده را پیاده سازی کنید. توجه کنید که نمی‌توانید از کلاس ماتریس خود جاوا استفاده کنید.

بلکه باید یک کلاس ماتریس بسازید که با استفاده از انواع لیست (ArrayList/LinkedList/etc.) ماتریس را شبیه سازی می‌کند.

عملیات‌هایی که باید انجام دهید از این قرارند:

1- محاسبه ترانهاده یک ماتریس

2- ضرب دو ماتریس

3- محاسبه تریس یک ماتریس (trace)

4- محاسبه نرم فروبنیوس یک ماتریس (frobenius norm)

(هول نکنید! در ادامه گفتم هرکدوم چیه فرمولش)

فرمت ورودی

خط اول

در خط اول ورودی یک کاراکتر ورودی داده می‌شود که نشان دهنده نوع عملیات است:

t: ترانهاده

m: ضرب

تریس :r

نرم فروبنیوس :n

خط دوم

تعداد سطر و تعداد ستون‌های ماتریس که با فاصله از هم جدا شده‌اند.

خط سوم

مقادیر داخل ماتریس به ترتیب سطر که با فاصله از هم جدا شده‌اند. برای مثال اگر تعداد سطرها 2 و ستون‌ها 3 باشد، ورودی $1\ 2\ 3\ 4\ 5\ 6$ معادل ماتریس زیر است:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

اگر عملیات خواسته شده ضرب باشد:

خط چهارم: تعداد سطر و تعداد ستون‌های ماتریس دوم که با فاصله از هم جدا شده‌اند.

خط پنجم: مقادیر داخل ماتریس به ترتیب سطر که با فاصله از هم جدا شده‌اند.

(عین خط دوم و سوم که ماتریس اول ورودی داده میشه)

فرمت خروجی

اگر نتیجه نهایی عدد است (حاصل عملیات تریس و نرم) عدد را تا *دو رقم اعشار گرد کرده* (اگر اعشاریه!) و چاپ کنید. اگر عدد نهایی اعشار ندارد، بدون ممیز و به صورت int آن را چاپ کنید.

اگر نتیجه نهایی ماتریس است (حاصل عملیات ترانهاده و ضرب) به این شکل ماتریس را پرینت کنید:

کل ماتریس داخل یک کروشه کلی و هر سطر یک کروشه جدا که مقادیر ستون‌های آن با فاصله و کاما جدا شده‌اند. مثال:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
[[1, 2], [3, 4]]
```

راهنمایی:

چاپ یک شی `ArrayList<ArrayList<Integer>>` مستقیما با فرمت خواسته شده مطابق است!

نکته: درایه‌های ماتریس‌ها حتما عدد صحیح هستند.

در ادامه فرمول‌های محاسبه trace و نرم فروبنیوس توضیح داده شده‌اند:

تریس: جمع مقادیر روی قطر اصلی یک ماتریس.

$$trace(A) = \sum_{i=1}^{n} a_{ii}$$

بدیهتا این پارامتر تنها برای ماتریس‌های مربعی قابل تعریف است و در صورت فراخوانی روی ماتریس غیرمربعی باید در کنسول چاپ کنید:

```
Dimension Error!
```

نرم فروبنیوس: جذر جمع مربع درایه‌های ماتریس.در ادامه منظور از m تعداد سطرها و n تعداد ستون‌هاست.

$$||A||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}^2}$$

همچنین در مورد ضرب دو ماتریس، اگر ابعاد دو ماتریس داده شده طوری بود که قابل ضرب کردن در هم نبودند مجددا چاپ کنید:

```
Dimension Error!
```

توصیه اکید!: قبل از اینکه نرم فروبنیوس رو پیاده سازی کنید یه سرچ بکنیدش! فرمول‌های ساده‌تر برای محاسبه‌اش وجود داره.

نمونه ورودی 1

نرم

```
n
2 2
1 2 3 4
```

نمونه خروجی 1

```
5.48
```

نمونه ورودی 2

ترانهاده

```
t
4 5
0 -1 2 1 2 -3 2 2 2 13 13 13 -1 -1 -1 0 0 0 0 100
```

نمونه خروجی 2

```
[[0, -3, 13, 0], [-1, 2, 13, 0], [2, 2, -1, 0], [1, 2, -1, 0], [2, 13, -1, 100]]
```

نمونه ورودی 3

ضرب

```
m
2 2
1 2 3 4
2 4
0 10 100 1000 2000 3000 4000 5000
```

نمونه خروجی 3

```
[[4000, 6010, 8100, 11000], [8000, 12030, 16300, 23000]]
```

نمونه ورودی 4

ضرب

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

نمونه خروجی 4

```
Dimension Error!
```

نمونه ورودی 5

تریس

```
r
2 2
-1 2 -3 0
```

نمونه خروجی 5

```
-1
```

نمونه ورودی 6

تریس

```
r
2 3
-1 2 -3 0 7 8
```

نمونه خروجی 6

```
Dimension Error!
```