توضیحات پیاده‌سازی

- شما باید تنها یک فایل ارسال نمایید (اسم فایل در صورت مسئله گفته شده است).
- این فایل باید زیربرنامه‌هایی که در صورت مسئله تشریح شده پیاده‌سازی کند طوری که منطبق بر الگوی داده‌شده در پیاده‌سازی نمونه باشد.
- این زیربرنامه‌ها باید همان‌گونه که در صورت مسئله آمده رفتار کنند.
- شما می‌توانید زیربرنامه‌های دیگر شامل توابع، روال‌ها و متدها نیز پیاده‌سازی کنید.
- برنامه‌ی شما نباید به هیچ طریقی با ورودی/خروجی استاندارد، یا هر فایل دیگری ارتباط برقرار کند.

به ویژه، اگر برنامه‌ی شما چیزی را به خروجی استاندارد بفرستد نتیجه‌ی ارزیابی آن در این تست «نقض امنیت - Security Violation, SV خواهد بود. شما می‌توانید هرچیزی را در Standard error بنویسید.

تعاریف

در صورت مسئله‌ها از کلمه‌ی **آرایه** استفاده شده و بخش‌های «جزئیات پیاده‌سازی» از نوع متغیر `int[]` استفاده می‌کنند. بسته به زبان برنامه‌سازی، ارزیاب‌ها از نوع متغیرهای زیر به‌جای `int[]` استفاده می‌کنند (که به خاطر سادگی، تمام آن‌ها را آرایه می‌نامیم):

- `std::vector<int>` در زبان C++
- `int*` در زبان C
- `array of longint` در زبان پاسکال
- `int[]` در جاوا

بخش‌های جزئیات پیاده‌سازی از نوع متغیر `int64` استفاده می‌کنند که به عنوان یک عدد صحیح ۶۴ بیتی با علامت است:

- `long long` در زبان C++
- `int64` در پاسکال
- `long` در جاوا

محدودیت‌ها

مسئله	محدودیت زمان	محدودیت حافظه
تشخیص مولکول‌ها	۱ ثانیه	۲ گیگابایت
ریل ترن هوایی	۲ ثانیه	۲ گیگابایت
میان‌بُر	۲ ثانیه	۲ گیگابایت

تشخیص مولکول‌ها

پطر برای شرکتی کار می‌کند که یک دستگاه برای تشخیص مولکول‌ها ساخته است. وزن هر مولکول یک عدد صحیح مثبت است. دستگاه یک **بازه‌ی تشخیص** $[l, u]$ دارد، که l و u اعداد صحیح مثبت هستند. دستگاه می‌تواند یک مجموعه از مولکول‌ها را تشخیص دهد، اگر و تنها اگر این مجموعه شامل زیرمجموعه‌ای از مولکول‌ها باشد که مجموع وزن آن‌ها در بازه‌ی تشخیص دستگاه قرار داشته باشد.

به عبارت دیگر، n مولکول با وزن‌های $w_0, \ldots, w_{n-1}$ را در نظر بگیرید. عملیات تشخیص موفقیت‌آمیز است اگر مجموعه‌ای از اندیس‌های متمایز $I = \{i_1, \ldots, i_m\}$ وجود داشته باشد طوری که $l \leq w_{i_1} + \cdots + w_{i_m} \leq u$.

با توجه به ویژگی‌های دستگاه، می‌دانیم که فاصله‌ی بین l و u قطعاً بزرگ‌تر یا مساوی فاصله‌ی بین وزن سنگین‌ترین و سبک‌ترین مولکول است. به عبارت دیگر، $u - l \geq w_{max} - w_{min}$، که در آن $w_{max} = \max(w_0, \ldots, w_{n-1})$ و $w_{min} = \min(w_0, \ldots, w_{n-1})$.

شما باید برنامه‌ای بنویسید که زیرمجموعه‌ای از مولکول‌ها را که مجموع وزن آن‌ها در بازه‌ی تشخیص قرار دارد بیابد، یا این که تعیین کند چنین زیرمجموعه‌ای وجود ندارد.

جزئیات پیاده‌سازی

شما باید تابع زیر را پیاده‌سازی کنید:

- `int[] solve(int l, int u, int[] w)`
  - `l` و `u`: ابتدا و انتهای بازه‌ی تشخیص هستند،
  - `w`: وزن مولکول‌ها است،
  - اگر زیرمجموعه‌ی موردنظر وجود داشته باشد، تابع باید یک آرایه از اندیس‌های مولکول‌هایی را برگرداند که چنین مجموعه‌ای را می‌سازد. اگر بیش از یک جواب درست وجود داشته باشد، می‌توانید یکی از آن‌ها را به دلخواه برگردانید.
  - اگر زیرمجموعه‌ی موردنظر وجود نداشته باشد، تابع باید یک آرایه‌ی خالی برگرداند.

برای زبان C، تعریف تابع کمی متفاوت است:

- `int solve(int l, int u, int[] w, int n, int[] result)`
  - `n`: تعداد اعضای `w` (یعنی تعداد مولکول‌ها) است،
  - بقیه‌ی پارامترها همانند بالا است.
  - به جای برگرداندن یک آرایه از m اندیس (همانند بالا)، تابع باید اندیس‌ها را در اولین m خانه‌ی آرایه‌ی `result` بنویسد و سپس m را برگرداند.

- اگر زیرمجموعه‌ی موردنظر وجود نداشت، تابع نباید چیزی را در آرایه‌ی `result` بنویسد و باید مقدار ۰ را برگرداند.

برنامه‌ی شما می‌تواند اندیس‌ها را به هر ترتیب دل‌خواه در آرایه‌ی بازگشتی (یا در آرایه‌ی `result` در C) قرار دهد.
برای جزئیات پیاده‌سازی در زبان برنامه‌سازی موردنظر خود، لطفا از فایل‌های قالب داده‌شده استفاده کنید.

مثال‌ها

مثال ۱

```
solve(15, 17, [6, 8, 8, 7])
```

در این مثال چهار مولکول با وزن‌های ۶، ۸، ۸ و ۷ داده شده است. دستگاه می‌تواند هر زیرمجموعه از مولکول‌ها را که مجموع وزن آن‌ها در بازه‌ی بسته‌ی ۱۵ تا ۱۷ قرار دارد تشخیص دهد. دقت کنید که $۶ - ۸ \geq ۱۵ - ۱۷$. به طور مثال، مجموع وزن مولکول‌های ۱ و ۳ برابر است با $w_۱ + w_۳ = ۸ + ۷ = ۱۵$، بنابراین تابع می‌تواند آرایه‌ی `[1, 3]` را برگرداند. جواب‌های ممکن دیگر عبارت‌اند از `[1, 2]` ($w_۱ + w_۲ = ۸ + ۸ = ۱۶$) و `[2, 3]` ($w_۲ + w_۳ = ۸ + ۷ = ۱۵$).

مثال ۲

```
solve(14, 15, [5, 5, 6, 6])
```

در این مثال چهار مولکول با وزن‌های ۵، ۵، ۶ و ۶ داده شده است و ما به دنبال زیرمجموعه‌ای از آن‌ها با مجموع وزن بین ۱۴ و ۱۵ هستیم. مجددا دقت کنید که $۵ - ۶ \geq ۱۴ - ۱۵$. هیچ زیرمجموعه‌ای از مولکول‌ها با مجموع وزن ۱۴ یا ۱۵ وجود ندارد، بنابراین تابع باید یک آرایه‌ی تهی برگرداند.

مثال ۳

```
solve(10, 20, [15, 17, 16, 18])
```

در این مثال چهار مولکول با وزن‌های ۱۵، ۱۷، ۱۶ و ۱۸ داده شده است و ما به دنبال زیرمجموعه‌ای از آن‌ها با مجموع وزن بین ۱۰ و ۲۰ هستیم. مجددا، دقت کنید که $۱۵ - ۱۸ \geq ۱۰ - ۲۰$. هر زیرمجموعه‌ی یک‌عضوی مجموع وزنی بین ۱۰ و ۲۰ دارد، بنابراین جواب‌های درست عبارت‌اند از `[0]`، `[1]`، `[2]` و `[3]`.

زیرمسئله‌ها

۱. (۹ امتیاز): $۱ \leq n \leq ۱۰۰$، $۱ \leq w_i \leq ۱۰۰$، $۱ \leq u, l \leq ۱۰۰۰$ و همه‌ی w_iها برابرند.

۲. (۱۰ امتیاز): $۱ \leq n \leq ۱۰۰$، $۱ \leq w_i, u, l \leq ۱۰۰۰$ و $\max(w_۰, \ldots, w_{n-۱}) - \min(w_۰, \ldots, w_{n-۱}) \leq ۱$.

۳. (۱۲ امتیاز): $۱ \leq n \leq ۱۰۰$ و $۱ \leq w_i, u, l \leq ۱۰۰۰$.

۴. (۱۵ امتیاز): $۱ \leq n \leq ۱۰\,۰۰۰$ و $۱ \leq w_i, u, l \leq ۱۰\,۰۰۰$.

۵. (۲۳ امتیاز): $۱ \leq n \leq ۱۰\,۰۰۰$ و $۱ \leq w_i, u, l \leq ۵۰۰\,۰۰۰$.

۶. (۳۱ امتیاز): $۱ \leq n \leq ۲۰۰\,۰۰۰$ و $۱ \leq w_i, u, l < ۲^{۳۱}$.

ارزیاب نمونه

ارزیاب نمونه، ورودی را در قالب زیر می‌خواند:

- خط ۱: اعداد صحیح n، l و u.
- خط ۲: n عدد صحیح: $w_0, \ldots, w_{n-1}$.

ریل ترن هوایی

آنا در یک شهربازی کار می‌کند و مسئول ساخت خط ریلی یک ترن هوایی جدید است. او پیش از این n قسمت ویژه طراحی کرده است که سرعت ترن هوایی را تحت تاثیر قرار می‌دهند و با شماره‌های ۰ تا $n-1$ شماره‌گذاری شده‌اند. حال او باید آن‌ها را در کنار یک‌دیگر قرار دهد و یک طراحی نهایی برای ترن هوایی پیشنهاد کند. می‌توانید طول قطار را در این مسئله صفر در نظر بگیرید.

به ازای هر i، $0 \leq i \leq n-1$ ، قسمت ویژه‌ی i دو خصوصیت دارد:

- هنگام ورود به این قسمت، محدودیت سرعت وجود دارد: سرعت قطار باید **کوچک‌تر یا مساوی** s_i کیلومتر بر ساعت باشد.
- هنگام خروج از این قسمت، سرعت قطار **دقیقاً** t_i کیلومتر بر ساعت می‌شود، مستقل از سرعتی که قطار با آن وارد این قسمت شده است.

طرح نهایی ترن هوایی، یک خط ریلی است که n قسمت ویژه به ترتیب مشخصی در آن قرار دارند. هر یک از این n قسمت باید دقیقاً یک‌بار استفاده شوند. قسمت‌های متوالی در این ترتیب، با استفاده از ریل‌های رابط به یک‌دیگر متصل شده‌اند. آنا باید ترتیب این n قسمت و طول ریل‌های بین قسمت‌های متوالی را مشخص کند. طول هر ریل رابط به متر اندازه‌گیری می‌شود و می‌تواند هر عدد صحیح نامنفی‌ای (شامل صفر) باشد.

عبور از هر متر از ریل بین دو قسمت ویژه، سرعت قطار را یک کیلومتر بر ساعت کاهش می‌دهد. در ابتدای حرکت، قطار با سرعت یک کیلومتر بر ساعت وارد اولین قسمت ویژه، طبق ترتیبی که آنا انتخاب کرده است، می‌شود.

طرح نهایی باید شرایط زیر را داشته باشد:

- هنگام ورود به قسمت‌های ویژه، قطار هیچ‌یک از محدودیت‌های سرعت را نقض نکند.
- سرعت قطار در هر لحظه مثبت باشد.

در تمامی زیرمسئله‌ها به جز زیرمسئله‌ی ۳، هدف شما پیدا کردن کمترین مجموع ممکن طول ریل‌های بین قسمت‌ها است. در زیرمسئله‌ی ۳، فقط باید مشخص کنید که آیا طرح معتبری وجود دارد که طول تمام ریل‌های رابط آن صفر باشد.

جزئیات پیاده‌سازی

شما باید تابع زیر را پیاده‌سازی کنید:

- `int64 plan_roller_coaster(int[] s, int[] t)`
  - `s`: آرایه‌ای به طول n شامل بیشترین سرعت‌های مجاز ورودی.
  - `t`: آرایه‌ای به طول n شامل سرعت‌های خروجی.

- در تمامی زیرمسئله‌ها به جز زیرمسئله‌ی ۳، تابع باید کمینه‌ی مجموع طول تمام ریل‌های بین قسمت‌های ویژه را برگرداند. در زیرمسئله‌ی ۳ اگر یک طراحی معتبر وجود داشته باشد که طول هر ریل در آن صفر است، تابع باید عدد ۰ و در غیراین‌صورت یک عدد صحیح مثبت دلخواه را برگرداند.

برای زبان C تعریف تابع کمی متفاوت است:

- `int64 plan_roller_coaster(int n, int[] s, int[] t)`
  - `n`: تعداد عناصر داخل `s` و `t` (یا همان تعداد قسمت‌های ویژه).
  - سایر پارامترها مانند بالا هستند.

مثال

```
plan_roller_coaster([1, 4, 5, 6], [7, 3, 8, 6])
```

در این مثال، چهار قسمت ویژه وجود دارد. بهترین طرح، قسمت‌ها را به ترتیب ۰، ۳، ۱ و ۲ قرار می‌دهد و آن‌ها را به ترتیب با ریل‌هایی به طول‌های ۱، ۲ و ۰ به یکدیگر متصل می‌کند. قطار این‌گونه در این طرح حرکت می‌کند:

- در ابتدا سرعت قطار یک کیلومتر بر ساعت است.
- قطار حرکت را با ورود به قسمت ویژه‌ی ۰ آغاز می‌کند.
- قطار قسمت ۰ را با سرعت ۷ کیلومتر بر ساعت ترک می‌کند.
- پس از آن ریلی به طول یک متر قرار دارد. سرعت قطار در انتهای این ریل ۶ کیلومتر بر ساعت است.
- قطار با سرعت ۶ کیلومتر بر ساعت وارد قسمت ویژه‌ی ۳ می‌شود و آن را با همان سرعت ترک می‌کند.
- پس از ترک قسمت ۳، قطار یک ریل ۲ متری را طی می‌کند و سرعت آن به ۴ کیلومتر بر ساعت کاهش می‌یابد.
- قطار با سرعت ۴ کیلومتر بر ساعت وارد قسمت ویژه‌ی ۱ می‌شود و با سرعت ۳ کیلومتر بر ساعت آن را ترک می‌کند.
- بلافاصله پس از قسمت ویژه‌ی ۱، قطار وارد قسمت ویژه‌ی ۲ می‌شود.
- قطار قسمت ویژه‌ی ۲ را ترک می‌کند. سرعت نهایی آن ۸ کیلومتر بر ساعت است.

تابع باید مجموع طول ریل‌های بین قسمت‌های ویژه را برگرداند که برابر $۳ = ۰ + ۲ + ۱$ است.

زیرمسئله‌ها

در تمامی زیرمسئله‌ها، $۱ \le s_i \le ۱۰^۹$ و $۱ \le t_i \le ۱۰^۹$

۱. (۱۱ امتیاز): $۲ \le n \le ۸$

۲. (۲۳ امتیاز): $۲ \le n \le ۱۶$

۳. (۳۰ امتیاز): $۲ \le n \le ۲۰۰\,۰۰۰$. در این زیرمسئله برنامه‌ی شما تنها باید صفر بودن جواب را بررسی کند. اگر پاسخ صفر نیست، هر عدد صحیح مثبتی درست در نظر گرفته می‌شود.

۴. (۳۶ امتیاز): $۲ \le n \le ۲۰۰\,۰۰۰$

ارزیاب نمونه

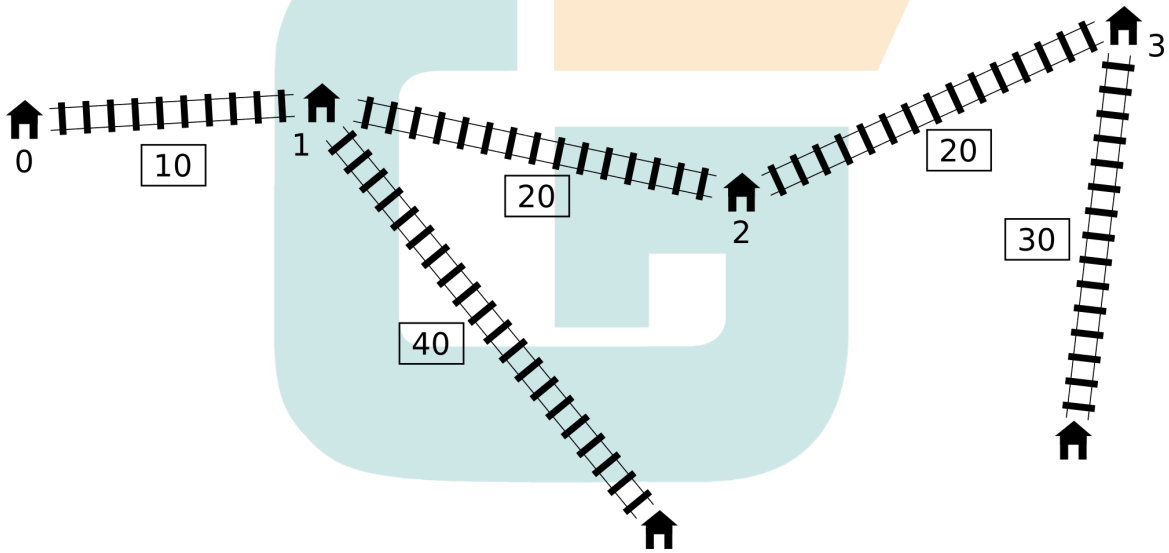
ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: عدد صحیح n.
- خط $i + ۲$، به ازای هر i بین ۰ و $n - ۱$: اعداد صحیح s_i و t_i.

میان‌بُر

پاول یک راه‌آهن اسباب‌بازی دارد که خیلی ساده است: یک خط اصلی شامل n ایستگاه، که در طول خط به ترتیب از صفر تا $n-1$ شماره‌گذاری شده‌اند. فاصله‌ی بین ایستگاه‌های i و $i+1$ برابر l_i سانتی‌متر است ($0 \leq i < n-1$).

علاوه‌بر خط اصلی، ممکن است چند خط جانبی هم وجود داشته باشد. هر خط جانبی یک خط آهن بین یک ایستگاه در خط اصلی و یک ایستگاه جدید است که روی خط اصلی قرار ندارد (ایستگاه‌های جدید شماره‌گذاری نشده‌اند). از هر ایستگاه خط اصلی، حداکثر یک خط جانبی می‌تواند آغاز شود. طول خط جانبی که از ایستگاه i شروع می‌شود d_i سانتی‌متر است. اگر هیچ خط جانبی از ایستگاه i ام آغاز نشود $d_i = 0$ خواهد بود.



اکنون، پاول در نظر دارد یک میان‌بُر بسازد: یک خط سریع‌السیر بین دو ایستگاه مختلف **خط اصلی** (که می‌توانند همسایه هم باشند). طول خط سریع‌السیر دقیقا c سانتی‌متر است، مستقل از آن‌که کدام دو ایستگاه را به هم وصل می‌کند.

هر قسمت از راه‌آهن، شامل خط سریع‌السیر جدید، دوطرفه است. *فاصله‌ی* بین دو ایستگاه، کم‌ترین طول مسیری است که روی راه‌آهن از یکی به دیگری می‌رود. *قطر* کل شبکه‌ی راه‌آهن، بیشینه فاصله‌ی میان همه‌ی جفت ایستگاه‌ها است؛ به عبارت دیگر، کوچک‌ترین عدد t است طوری‌که فاصله‌ی بین هر جفت ایستگاه حداکثر t باشد.

پاول می‌خواهد خط سریع‌السیر را طوری بسازد که قطر شبکه‌ی حاصل کمینه باشد.

جزئیات پیاده‌سازی

شما باید تابع زیر را پیاده‌سازی کنید:

```
int64 find_shortcut(int n, int[] l, int[] d, int c)
```

- n: تعداد ایستگاه‌های خط اصلی
- l: فاصله‌ی بین ایستگاه‌های خط اصلی (آرایه به طول $n - ۱$)
- d: طول خط‌های جانبی (آرایه به طول n)
- c: طول خط سریع‌السیر جدید
- تابع باید کمترین قطر ممکن شبکه‌ی راه‌آهن را بعد از افزودن خط سریع‌السیر برگرداند.

لطفا از فایل‌های قالب ارائه‌شده برای جزئیات پیاده‌سازی در زبان برنامه‌نویسی خودتان استفاده کنید.

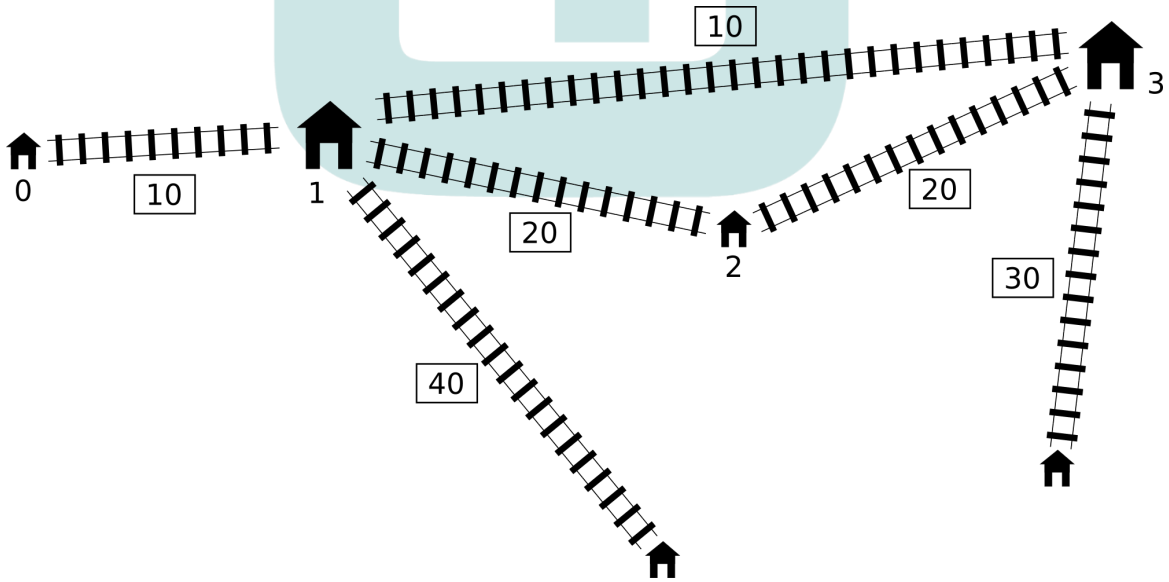
مثال‌ها

مثال ۱

برای شبکه‌ی راه‌آهنی که در بالا نمایش داده‌شد، ارزیاب فراخوانی تابع را به صورت زیر انجام می‌دهد:

```
find_shortcut(4, [10, 20, 20],
    [0, 40, 0, 30], 10)
```

راه‌حل بهینه، ساختن خط سریع‌السیر بین ایستگاه‌های ۱ و ۳ است، به صورتی که در شکل زیر نمایش‌داده شده.



قطر شبکه‌ی راه‌آهن جدید ۸۰ سانتی‌متر است، بنابراین تابع باید عدد ۸۰ را برگرداند.

مثال ۲

ارزیاب، فراخوانی تابع زیر را انجام می‌دهد:

```
find_shortcut(9, [10, 10, 10, 10, 10, 10, 10, 10],
    [20, 0, 30, 0, 0, 40, 0, 40, 0], 30)
```

راه‌حل بهینه، متصل کردن ایستگاه‌های ۲ و ۷ است؛ که در این حالت قطر ۱۱۰ خواهد بود.

مثال ۳

ارزیاب، فراخوانی تابع زیر را انجام می‌دهد:

```
find_shortcut(4, [2, 2, 2],
    [1, 10, 10, 1], 1)
```

راه‌حل بهینه اتصال ایستگاه‌های ۱ و ۲ است که قطر را به ۲۱ کاهش می‌دهد.

مثال ۴

ارزیاب، فراخوانی تابع زیر را انجام می‌دهد:

```
find_shortcut(3, [1, 1],
    [1, 1, 1], 3)
```

متصل کردن هر جفت ایستگاه با خط سریع‌السیر به طول ۳ قطر اولیه‌ی شبکه‌ی راه‌آهن را که ۴ است بهبود نمی‌دهد.

زیرمسئله‌ها

در همه‌ی زیرمسئله‌ها $2 \le n \le 1\,000\,000$، $1 \le l_i \le 10^9$، $0 \le d_i \le 10^9$ و $1 \le c \le 10^9$ است.

۱. (۹ امتیاز) $2 \le n \le 10$
۲. (۱۴ امتیاز) $2 \le n \le 100$
۳. (۸ امتیاز) $2 \le n \le 250$
۴. (۷ امتیاز) $2 \le n \le 500$
۵. (۳۳ امتیاز) $2 \le n \le 3000$
۶. (۲۲ امتیاز) $2 \le n \le 100\,000$
۷. (۴ امتیاز) $2 \le n \le 300\,000$
۸. (۳ امتیاز) $2 \le n \le 1\,000\,000$

ارزیاب نمونه

ارزیاب نمونه ورودی را به صورت زیر می‌خواند:

- سطر ۱: اعداد صحیح n و c
- سطر ۲: اعداد صحیح $l_0, l_1, \ldots, l_{n-2}$
- سطر ۳: اعداد صحیح $d_0, d_1, \ldots, d_{n-1}$

توضیحات پیاده‌سازی

- شما باید تنها یک فایل ارسال نمایید (اسم فایل در صورت مسئله گفته شده است).
- این فایل باید زیربرنامه‌هایی که در صورت مسئله تشریح شده پیاده‌سازی کند طوری که منطبق بر الگوی داده‌شده در پیاده‌سازی نمونه باشد.
- این زیربرنامه‌ها باید همان‌گونه که در صورت مسئله آمده رفتار کنند.
- شما می‌توانید زیربرنامه‌های دیگر شامل توابع، روال‌ها و متدها نیز پیاده‌سازی کنید.
- برنامه‌ی شما نباید به هیچ طریقی با ورودی/خروجی استاندارد، یا هر فایل دیگری ارتباط برقرار کند. به ویژه، اگر برنامه‌ی شما چیزی را به خروجی استاندارد بفرستد نتیجه‌ی ارزیابی آن در این تست «نقض امنیت – SV (Security Violation) » خواهد بود. شما می‌توانید هرچیزی را در Standard error بنویسید.

تعاریف

در صورت مسئله‌ها و توضیحات پیاده‌سازی موارد زیر به عنوان نام‌های کلی برای انواع داده‌های به کار رفته است:

- نام *آرایه* و نوع متناظر `int[]`
- نوع `int64`
- نوع `string`
- نوع `boolean`

در هر یک از زبان‌های برنامه‌سازی پشتیبانی شده برنامه ارزیاب نوع داده متناسب با آن زبان را مطابق جدول زیر استفاده خواهد کرد:

boolean	string	int64	array	زبان
`bool`	`std::string`	`long long`	`std::vector`	C++
`int`	`char*`	`long long`	`int*`	C
`boolean`	`string`	`int64`	`array of longint`	Pascal
`boolean`	`String`	`long`	`int[]`	Java

محدودیت‌ها

مسئله	محدودیت زمان	محدودیت حافظه
رنگ‌آمیزی با اعداد	۲ ثانیه	۲ گیگابایت
رمزگشایی یک اشتباه دردسرساز	۲ ثانیه	۲ گیگابایت
فضایی‌ها	۲ ثانیه	۲ گیگابایت

رنگ‌آمیزی با اعداد

«رنگ‌آمیزی با اعداد» یک بازی معروف است. ما یک نسخه‌ی ساده‌ی یک‌بعدی از این بازی را در نظر می‌گیریم. در این بازی یک سطر شامل n خانه به بازیکن داده می‌شود. خانه‌ها از چپ به راست به ترتیب با ۰ تا $n-1$ شماره‌گذاری شده‌اند. بازیکن باید هر خانه را با رنگ سفید یا سیاه رنگ‌آمیزی کند. خانه‌های سیاه را با 'X' و خانه‌های سفید را با '_' نشان می‌دهیم.

دنباله‌ی $c = [c_0, \ldots, c_{k-1}]$ از k عدد صحیح مثبت که آن‌ها را سرنخ می‌نامیم به بازیکن داده شده است. او باید خانه‌ها را به نحوی رنگ‌آمیزی کند که خانه‌های سیاه در این سطر دقیقاً k بلوک از خانه‌های مجاور تشکیل دهند. هم‌چنین تعداد خانه‌های سیاه در بلوک iاُم (با شروع از ۰) از سمت چپ باید برابر c_i باشد. مثلا اگر سرنخ‌ها برابر $c = [3, 4]$ باشند، جواب بازی شامل دقیقاً دو بلوک از خانه‌های سیاه مجاور است که یکی به طول ۳ و دیگری به طول ۴ است. در نتیجه اگر $n = 10$ و $c = [3, 4]$، یک جواب سازگار با سرنخ برابر "_XXX__XXXX" است. توجه داشته باشید که "XXXX_XXX__" یک جواب سازگار با سرنخ نیست، چون بلوک‌های خانه‌های سیاه در ترتیب درستی قرار ندارند. هم‌چنین "_XXXXXXX__" یک جواب سازگار با سرنخ نیست، چون به جای دو بلوک مجزا فقط یک بلوک از خانه‌های سیاه وجود دارد.

یک سناریوی نیمه‌تمام از بازی «رنگ‌آمیزی با اعداد» به شما داده شده است. به این معنی که شما n و c را می‌دانید، و علاوه بر آن می‌دانید که بعضی از خانه‌ها باید سیاه و بعضی از خانه‌ها باید سفید شوند. وظیفه‌ی شما این است که اطلاعات بیشتری در مورد خانه‌ها به‌دست آورید.

به طور دقیق‌تر، یک «جواب معتبر» جوابی است که با سرنخ‌ها سازگاری داشته، و هم‌چنین با خانه‌هایی که رنگ آن‌ها از قبل مشخص شده، مطابقت داشته باشد. برنامه‌ی شما باید خانه‌هایی را که در همه‌ی جواب‌های معتبر به رنگ سیاه رنگ‌آمیزی می‌شوند، و نیز خانه‌هایی را که در همه‌ی جواب‌های معتبر به رنگ سفید رنگ‌آمیزی می‌شوند پیدا کند.

می‌توانید فرض کنید که ورودی به شکلی است که حداقل یک جواب معتبر وجود دارد.

جزئیات پیاده‌سازی

شما باید تابع زیر را پیاده‌سازی کنید:

- `string solve_puzzle(string s, int[] c)`
- s: رشته‌ای به طول n. برای هر i ($0 \le i \le n-1$) کارکتر i برابر است با:
  - 'X'، اگر خانه‌ی i باید سیاه باشد،
  - '_'، اگر خانه‌ی i باید سفید باشد،
  - '.'، اگر اطلاعاتی در مورد خانه‌ی i ارائه نشده باشد،
- c: آرایه‌ای به طول k از سرنخ‌ها، مطابق آن چه بالا تعریف شد،
- تابع باید رشته‌ای به طول n به عنوان خروجی برگرداند. برای هر i ($0 \le i \le n-1$) کارکتر i خروجی باید به صورت زیر باشد:

- ’X‘، اگر خانه‌ی i در هر جواب معتبر باید سیاه باشد،
- ’_‘، اگر خانه‌ی i در هر جواب معتبر باید سفید باشد،
- ’?‘، در غیر این صورت (یعنی اگر دو جواب معتبر وجود داشته باشد طوری که خانه‌ی i در یکی از آن‌ها سیاه و در دیگری سفید باشد.)

برای زبان C، تعریف تابع کمی متفاوت است:

- `void solve_puzzle(int n, char* s, int k, int* c, char* result)`
  - `n`: طول رشته‌ی `s` (تعداد خانه‌ها)،
  - `k`: طول آرایه‌ی `c` (تعداد سرنخ‌ها)،
  - بقیه‌ی پارامترها همانند بالا است.
  - به جای برگرداندن یک رشته شامل n کاراکتر، تابع باید جواب را در رشته‌ی `result` بنویسد.

کد ASCII کارکترهای به‌کاررفته در این مسئله به شرح زیر است:

- ’X‘: ۸۸،
- ’_‘: ۹۵،
- ’.‘: ۴۶،
- ’?‘: ۶۳.

برای جزئیات پیاده‌سازی در زبان برنامه‌سازی موردنظر خود، لطفا از فایل‌های قالب داده‌شده استفاده کنید.

مثال‌ها

مثال ۱

`solve_puzzle(".........", [3, 4])`
تمام جواب‌های معتبر برای این بازی در زیر آمده است:

- “XXX_XXXX__”،
- “XXX__XXXX_”،
- “XXX___XXXX”،
- “_XXX_XXXX_”،
- “_XXX__XXXX”،
- “__XXX_XXXX”.

از این جواب‌ها می‌توان تشخیص داد که خانه‌های با اندیس ۲، ۶ و ۷ (با شروع از اندیس ۰) در همه‌ی جواب‌های معتبر سیاه است. بقیه‌ی خانه‌ها می‌توانند سیاه باشند، ولی لزوماً در تمام جواب‌ها سیاه نیستند. در نتیجه، جواب صحیح عبارت است از “??X???XX??”.

مثال ۲

```
solve_puzzle("........", [3, 4])
```

در این مثال، جواب به صورت یکتا قابل تشخیص است و جواب صحیح عبارت است از "XXX_XXXX".

مثال ۳

```
solve_puzzle("..._._....", [3])
```

در این مثال، می‌توانیم نتیجه بگیریم خانه‌ی ۴ نیز باید سفید باشد، چون هیچ سه خانه‌ی متوالی سیاهی را نمی‌توان بین دو خانه‌ی سفید با اندیس‌های ۳ و ۵ قرار داد. در نتیجه، جواب درست برابر "????___????" است.

مثال ۴

```
solve_puzzle(".X.......", [3])
```

فقط دو جواب معتبر در این مثال صدق می‌کنند:

- "XXX_____"،
- "_XXX_____".

بنابراین، جواب صحیح عبارت است از "?XX?_____".

زیرمسئله‌ها

در تمام زیرمسئله‌ها، $1 \le k \le n$، و برای هر $0 \le i \le k-1$ داریم $1 \le c_i \le n$.

۱. (۷ امتیاز) $n \le 20$، $k = 1$، s فقط شامل '.' است (بازی خالی)،

۲. (۳ امتیاز) $n \le 20$، s فقط شامل '.' است،

۳. (۲۲ امتیاز) $n \le 100$، s فقط شامل '.' است،

۴. (۲۷ امتیاز) $n \le 100$، s فقط شامل '.' و '_' است (فقط در مورد خانه‌های سفید اطلاعات داریم)،

۵. (۲۱ امتیاز) $n \le 100$،

۶. (۱۰ امتیاز) $n \le 5\,000$، $k \le 100$،

۷. (۱۰ امتیاز) $n \le 200\,000$، $k \le 100$.

ارزیاب نمونه

ارزیاب نمونه، ورودی را در قالب زیر می‌خواند:

- خط ۱: رشته‌ی s،
- خط ۲: عدد صحیح k و پس از آن k عدد صحیح $c_0, \ldots, c_{k-1}$.

رمزگشایی یک اشتباه دردسرساز

ایلشات یک مهندس نرم افزار است که روی موضوع داده‌ساختارهای بهینه کار می‌کند. یک روز، او داده ساختار جدیدی ابداع کرد. این داده‌ساختار می‌تواند مجموعه‌ای از اعداد صحیح n بیتی نامنفی را ذخیره کند که n توانی از دو است. یعنی عدد صحیح نامنفی b وجود دارد که $n = ۲^b$.

داده‌ساختار در ابتدا خالی است. برنامه‌ای که از این داده‌ساختار استفاده می‌کند باید قوانین زیر را رعایت کند:

- برنامه می‌تواند عناصر را که به شکل اعداد صحیح n بیتی هستند، یکی یکی، به کمک تابع `add_element(x)` به داده‌ساختار اضافه کند. اگر برنامه سعی کند عنصری را اضافه کند که پیش از این در داده‌ساختار وجود داشته است، هیچ اتفاقی رخ نمی‌دهد.
- پس از افزودن آخرین عنصر، برنامه باید تابع `compile_set()` را دقیقا یک‌بار صدا کند.
- در نهایت، برنامه می‌تواند با فراخوانی تابع `check_elemnt(x)` وجود عنصر x در داده‌ساختار را بررسی کند. این تابع می‌تواند چندین بار استفاده شود.

هنگامی که ایلشات برای اولین‌بار این داده‌ساختار را پیاده‌سازی کرد، در پیاده‌سازی تابع `compile_set()` مرتکب اشتباهی شد. بر اثر این اشتباه، ارقام دودویی هر عنصر درون مجموعه به ترتیب یکسانی جابه‌جا می‌شوند. ایلشات از شما می‌خواهد تا ترتیب دقیق جابه‌جایی ارقام ناشی از اشتباه را پیدا کنید.

به عبارت دقیق‌تر، دنباله‌ی $p = [p_۰, \ldots, p_{n-۱}]$ را در نظر بگیرید که هر یک از اعداد ۰ تا $n - ۱$ دقیقاً یک‌بار در آن آمده‌اند. به چنین دنباله‌ای یک جایگشت می‌گوییم. عنصری از مجموعه را در نظر بگیرید که ارقام آن در نمایش دودویی $a_۰, \ldots, a_{n-۱}$ است ($a_۰$ پرارزش‌ترین بیت است). هنگامی که تابع `compile_set()` صدا زده می‌شود، این عنصر با عنصر $a_{p_۰}, a_{p_۱}, \ldots, a_{p_{n-۱}}$ جایگزین می‌شود.

جایگشت یکسانی برای جابه‌جایی ارقام تمام عناصر استفاده می‌شود. هر جایگشتی امکان پذیر است، از جمله جایگشت $p_i = i$ به ازای هر $۰ \le i \le n - ۱$.

برای مثال، فرض کنید $n = ۴$، $p = [۲, ۱, ۳, ۰]$، و شما اعدادی را در مجموعه اضافه کرده‌اید که نمایش دودویی آن‌ها 0000، 1100 و 0111 است. فراخوانی تابع `compile_set` این عناصر را به ترتیب به 0000، 0101 و 1110 تبدیل می‌کند.

وظیفه‌ی شما نوشتن برنامه‌ای است که جایگشت p را به کمک تعامل با داده‌ساختار پیدا کند. برنامه باید (به ترتیب زیر):

۱. مجموعه‌ای از اعداد صحیح n بیتی را انتخاب کند،

۲. این اعداد را به داده‌ساختار اضافه کند،

۳. تابع `compile_set` را صدا کند تا موجب رخ دادن اشتباه شود،

۴. وجود برخی از عناصر را در مجموعه‌ی تغییر یافته بررسی کند،

۵. از این اطلاعات برای تعیین و بازگرداندن جایگشت p استفاده کند.

توجه کنید که برنامه‌ی شما تنها می‌تواند یک‌بار تابع `compile_set` را صدا کند.
به‌علاوه، در تعداد دفعات فراخوانی توابع کتابخانه توسط برنامه‌ی شما محدودیت وجود دارد. یعنی برنامه‌ی شما می‌تواند

- تابع `add_element` را حداکثر w بار صدا بزند (w ابتدای کلمه‌ی writes به معنای نوشتن‌ها است)
- تابع `check_element` را حداکثر r بار صدا بزند (r ابتدای کلمه‌ی reads به معنای خواندن‌ها است)

جزئیات پیاده‌سازی

شما باید یک تابع را پیاده‌سازی کنید:

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: تعداد بیت‌های نمایش دودویی هر عنصر درون مجموعه (و همچنین طول p)
  - `w`: بیشترین تعداد عملیات‌های `add_element` که برنامه‌ی شما می‌تواند انجام دهد.
  - `r`: بیشترین تعداد عملیات‌های `check_element` که برنامه‌ی شما می‌تواند انجام دهد.
  - تابع باید جایگشت بازیابی شده‌ی p را برگرداند.

برای زبان C تعریف تابع کمی متفاوت است:

- `void restore_permutation(int n, int w, int r, int* result)`
  - معنای `n`، `w` و `r` با بالا یکسان است.
  - تابع باید جایگشت بازیابی شده‌ی p را با ذخیره کردن آن در آرایه‌ی داده‌شده‌ی `result` برگرداند: به ازای هر i، باید مقدار p_i را در `result[i]` ذخیره کند.

توابع کتابخانه

برای تعامل با داده‌ساختار، برنامه‌ی شما باید از سه تابع زیر استفاده کند:

- `void add_element(string x)`

این تابع یک عنصر که با x نمایش داده می‌شود را به مجموعه اضافه می‌کند.

- `x`: رشته‌ای از `'0'` و `'1'` که نمایش دودویی عدد صحیحی است که باید به مجموعه اضافه شود. طول `x` باید n باشد.

- `void compile_set()`

این تابع باید دقیقا یک‌بار صدا زده شود. برنامه‌ی شما نمی‌تواند `add_element()` را بعد از فراخوانی این تابع صدا بزند. برنامه‌ی شما نمی‌تواند `check_element()` را پیش از فراخوانی این تابع صدا بزند.

- `boolean check_element(string x)`

این تابع وجود عنصر x در مجموعه‌ی تغییر یافته را بررسی می‌کند.

- `x`: رشته‌ای از `'0'` و `'1'`، نمایش دودویی عدد صحیحی که باید مورد بررسی قرار بگیرد. طول `x` باید n باشد.

- اگر `x` در مجموعه‌ی تغییر یافته باشد `true` برمی‌گرداند و درغیراین‌صورت، `false` برمی‌گرداند.

توجه کنید که اگر برنامه‌ی شما هر یک از محدودیت‌های بالا را رعایت نکند، نتیجه‌ی ارزیابی آن «پاسخ غلط ـ Wrong Answer» خواهد بود.

برای تمامی رشته‌ها، اولین کاراکتر پرارزش‌ترین بیت عدد متناظر رشته است.

ارزیاب نمونه جایگشت p را پیش از فراخوانی تابع `restore_permutation` تعیین می‌کند.

لطفا از فایل‌های قالب ارائه شده برای جزئیات پیاده‌سازی در زبان برنامه‌نویسی خود استفاده کنید.

مثال

ارزیاب تابع را به شکل زیر صدا می‌زند:

- `restore_permutation(4, 16, 16)`. در این حالت $n = ۴$ و برنامه می‌تواند حداکثر ۱۶ «نوشتن» و ۱۶ «خواندن» انجام دهد.

برنامه تابع‌ها را به ترتیب زیر فراخوانی می‌کند:

- `add_element("0001")`
- `add_element("0011")`
- `add_element("0100")`
- `compile_set()`
- `check_element("0001")` مقدار `false` برمی‌گرداند
- `check_element("0010")` مقدار `true` برمی‌گرداند
- `check_element("0100")` مقدار `true` برمی‌گرداند
- `check_element("1000")` مقدار `false` برمی‌گرداند
- `check_element("0011")` مقدار `false` برمی‌گرداند
- `check_element("0101")` مقدار `false` برمی‌گرداند
- `check_element("1001")` مقدار `false` برمی‌گرداند
- `check_element("0110")` مقدار `false` برمی‌گرداند
- `check_element("1010")` مقدار `true` برمی‌گرداند
- `check_element("1100")` مقدار `false` برمی‌گرداند

تنها یک جایگشت با مقادیر برگردانده‌شده توسط `check_element()` سازگار است: جایگشت $p = [۲, ۱, ۳, ۰]$. بنابراین، `restore_permutation` باید $[۲, ۱, ۳, ۰]$ را برگرداند.

زیرمسئله‌ها

۱. (۲۰ امتیاز): $n = ۸$، $w = ۲۵۶$، $r = ۲۵۶$، حداکثر برای ۲ اندیس i ، $p_i \neq i$ $(۰ \leq i \leq n - ۱)$

۲. (۱۸ امتیاز): $n = ۳۲$، $w = ۳۲۰$، $r = ۱۰۲۴$.

۳. (۱۱ امتیاز): $n = ۳۲$، $w = ۱۰۲۴$، $r = ۳۲۰$.

۴. (۲۱ امتیاز): $n = ۱۲۸$، $w = ۱۷۹۲$، $r = ۱۷۹۲$.

۵. (۳۰ امتیاز): $n = ۱۲۸$، $w = ۸۹۶$، $r = ۸۹۶$.

ارزیاب نمونه

ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: اعداد صحیح n، w، r،
- خط ۲: n عدد صحیح، عناصر p.

فضایی‌ها

ماهواره‌ی ما به تازگی یک تمدن فضایی را در سیاره‌ای دور کشف کرده‌است. ما قبلا یک عکس با وضوح کم از یک قسمت مربعی سیاره گرفته‌ایم. این عکس نشانه‌های زیادی از زندگی موجودات هوشمند نشان می‌دهد. متخصصان ما n نقطه‌ی جالب را در عکس شناسایی کرده‌اند که با شماره‌های صفر تا $n-1$ شماره‌گذاری شده‌اند. اکنون ما می‌خواهیم عکس‌هایی واضح بگیریم که همه‌ی آن n نقطه را شامل شوند.

ماهواره، سطح عکس با وضوح کم را به یک جدول $m \times m$ از خانه‌های با اندازه‌ی واحد تقسیم کرده‌است. سطرها و ستون‌های جدول پشت سرهم از صفر تا $m-1$ شماره‌گذاری شده‌اند (به ترتیب از بالا و چپ). ما خانه‌ی قرارگرفته در سطر s و ستون t را با (s,t) نمایش می‌دهیم. نقطه‌ی جالب i ام داخل خانه‌ی (r_i, c_i) است. هر خانه ممکن است تعداد دل‌خواهی از این نقطه‌ها داشته‌باشد.

ماهواره‌ی ما روی یک مدار پایدار است که مستقیما از روی قطر *اصلی* جدول می‌گذرد. قطر اصلی، پاره‌خطی است که گوشه‌ی بالا–چپ و پایین–راست جدول را وصل می‌کند. ماهواره می‌تواند از هر منطقه‌ای که همه‌ی شرایط زیر را برقرار می‌کند یک عکس واضح بگیرد:

- شکل منطقه مربعی باشد،
- دو گوشه‌ی مقابل مربع، هردو روی قطر اصلی جدول باشند،
- هر خانه از جدول یا کاملا داخل و یا کاملا خارج منطقه‌ی عکس‌برداری‌شده باشد.

ماهواره می‌تواند حداکثر k عکس واضح بگیرد.

هنگامی که ماهواره همه‌ی عکس‌ها را گرفت، از هر خانه‌ای که عکس‌برداری شده، یک عکس واضح به ایستگاه زمینی می‌فرستد (مستقل از آن‌که آن خانه شامل نقاط جالب باشد یا خیر). اطلاعات هر خانه‌ی عکس‌برداری‌شده *فقط یک بار* ارسال می‌شود، حتی اگر آن خانه چندین بار عکس‌برداری شده باشد.

بنابراین ما باید حداکثر k منطقه‌ی مربعی را برای عکس‌برداری انتخاب کنیم، با اطمینان از این‌که:

- هر خانه که دست‌کم یک نقطه‌ی جالب دارد، دست‌کم یک بار عکس‌برداری شود، و
- تعداد خانه‌هایی که دست‌کم یک بار عکس‌برداری‌شده‌اند کمینه شود.

شما باید کوچک‌ترین مجموع تعداد خانه‌های عکس‌برداری‌شده را پیدا کنید.

جزئیات پیاده‌سازی

شما باید تابع (متد) زیر را پیاده‌سازی کنید:

```
int64 take_photos(int n, int m, int k, int[] r, int[] c)
```

- n: تعداد نقاط جالب،
- m: تعداد سطرها (و هم‌چنین ستون‌های) جدول،
- k: حداکثر تعداد عکس‌هایی که ماهواره می‌تواند بگیرد،
- r و c: دو آرایه به طول n که نشانگر مختصات خانه‌های حاوی نقاط جالب جدول است. به‌ازای $0 \leq i \leq n-1$ نقطه‌ی جالب i ام در خانه‌ی $(r[i], c[i])$ قرار گرفته‌است.
- تابع باید کمینه‌ی تعداد کل خانه‌هایی که دست‌کم یک بار عکس‌برداری شده‌اند را برگرداند (با درنظرگرفتن آن‌که عکس‌ها باید همه‌ی نقاط جالب را بپوشانند).

لطفا از فایل‌های قالب که در اختیارتان قرار گرفته برای جزئیات پیاده‌سازی در زبان برنامه‌نویسی خودتان استفاده کنید.

مثال‌ها

مثال ۱

```
take_photos(5, 7, 2, [0, 4, 4, 4, 4], [3, 4, 6, 5, 6])
```

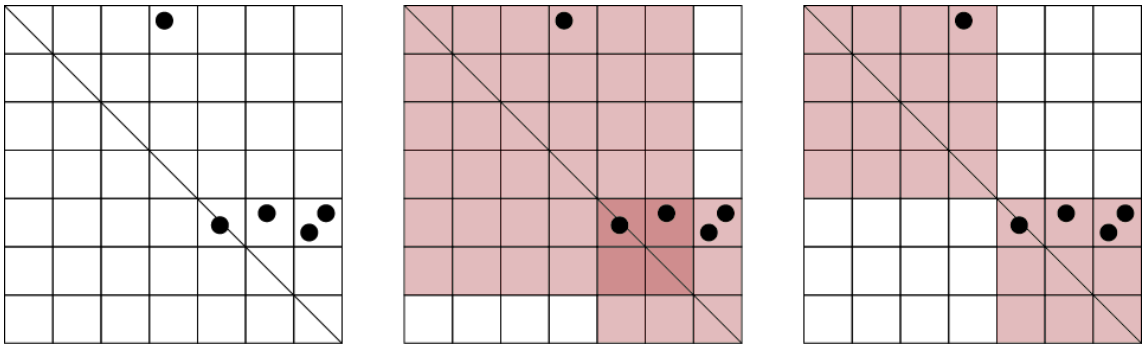
در این مثال ما یک جدول 7×7 با ۵ نقطه‌ی جالب داریم. نقاط جالب در ۴ خانه قرار دارند: $(0, 3)$، $(4, 4)$، $(4, 5)$ و $(4, 6)$. شما می‌توانید حداکثر ۲ عکس واضح بگیرید.

یک راه برای پوشش هر ۵ نقطه‌ی جالب، گرفتن ۲ عکس است: یک عکس از مربع 6×6 که شامل خانه‌های $(0, 0)$ و $(5, 5)$ است، و یک عکس از مربع 3×3 که شامل خانه‌های $(4, 4)$ و $(6, 6)$ است. اگر ماهواره همین دو عکس را بگیرد، داده‌های ۴۱ خانه را منتقل می‌کند. این مقدار بهینه نیست.

در راه‌حل بهینه، یک عکس از مربع 4×4 شامل خانه‌های $(0, 0)$ و $(3, 3)$ و یک عکس دیگر از مربع 3×3 شامل خانه‌های $(4, 4)$ و $(6, 6)$ وجود دارد. بدین‌ترتیب فقط ۲۵ خانه عکس‌برداری می‌شوند، که بهینه است. پس `take_photos` باید ۲۵ را برگرداند.

در نظر داشته باشید که کافی است از خانه‌ی $(4, 6)$ فقط یک بار عکس بگیریم، با آن‌که این خانه ۲ نقطه‌ی جالب دارد.

این مثال در شکل‌های ذیل نشان داده‌شده است. شکل سمت چپ جدولی را نشان می‌دهد که مربوط به این مثال است. شکل وسط نمایش‌گر راه حل غیر بهینه که در آن ۴۱ خانه عکس‌برداری می‌شوند است. شکل راست راه‌حل بهینه را نشان می‌دهد.

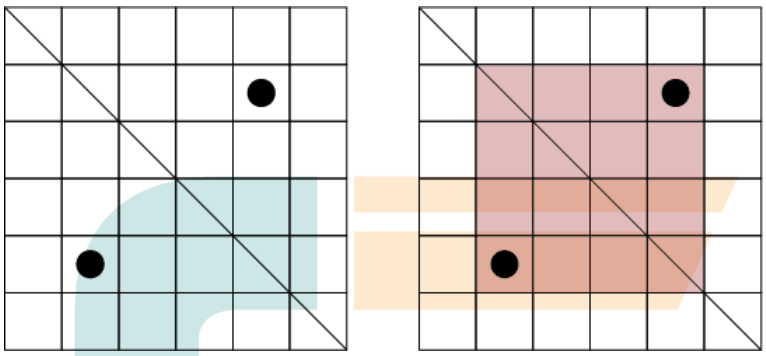


مثال ۲

```
take_photos(2, 6, 2, [1, 4], [4, 1])
```

این‌جا ما ۲ نقطه‌ی جالب داریم که به صورت متقارن در خانه‌های $(۱, ۴)$ و $(۴, ۱)$ قرار دارند. هر عکس معتبری که یکی از آن‌ها را شامل شود، شامل دیگری هم هست. بنابراین کافی است از یک عکس استفاده کنیم.

شکل زیر این مثال و راه‌حل بهینه‌ی آن را نشان می‌دهد. در این راه‌حل ماهواره یک عکس از ۱۶ خانه می‌گیرد.



زیرمسئله‌ها

در همه‌ی زیرمسئله‌ها $1 \le k \le n$ است.

۱. (۴ امتیاز) $1 \le n \le 50$، $1 \le m \le 100$، $k = n$،

۲. (۱۲ امتیاز) $1 \le n \le 500$، $1 \le m \le 1000$ و برای هر i $(0 \le i \le n-1)$ داریم $r_i = c_i$،

۳. (۹ امتیاز) $1 \le n \le 500$، $1 \le m \le 1000$،

۴. (۱۶ امتیاز) $1 \le n \le 4000$، $1 \le m \le 1\,000\,000$،

۵. (۱۹ امتیاز) $1 \le n \le 50\,000$، $1 \le k \le 100$، $1 \le m \le 1\,000\,000$،

۶. (۴۰ امتیاز) $1 \le n \le 100\,000$، $1 \le m \le 1\,000\,000$.

ارزیاب نمونه

ارزیاب نمونه ورودی را به صورت زیر می‌خواند:

- سطر ۱: اعداد صحیح n، m و k،
- سطر $2 + i$ $(0 \le i \le n-1)$: اعداد صحیح r_i و c_i.