بازی

جیان-جیا پسر جوانی است که عاشق بازی‌های فکری است. وقتی از او سؤالی پرسیده می‌شود، ترجیح می‌دهد به‌جای دادن پاسخ مستقیم یک بازی انجام دهد. او دوستش می-یو را ملاقات کرد و در مورد شبکه‌ی پروازی در تایوان به او توضیح داد. در تایوان n شهر که با شماره‌های ۰ تا $n-۱$ شناخته می‌شوند وجود دارد که بعضی از آن‌ها با پرواز مستقیم به یک‌دیگر وصل شده‌اند. هر خط پروازی دو شهر را به صورت دو طرفه به یک‌دیگر وصل می‌کند.

می-یو از جیان-جیا می‌پرسد که آیا امکان رفتن بین هر دو شهر (با پرواز مستقیم یا غیرمستقیم) وجود دارد؟ جیان-جیا که نمی‌خواهد صریح جواب دهد، درخواست می‌کند که یک بازی انجام دهند. می-یو می‌تواند سؤال‌هایی از این نوع بپرسد: «آیا شهرهای x و y با یک پرواز مستقیم به یک‌دیگر وصل هستند؟». جیان-جیا این نوع سؤال‌ها را بلافاصله با «بله» یا «خیر» پاسخ می‌دهد. می-یو می‌تواند در مورد هر زوج شهر دقیقا یک سؤال و در مجموع $r = n(n-۱)/۲$ سؤال بپرسد. می-یو برنده‌ی بازی می‌شود اگر به ازای یک مقدار $i < r$ با دریافت پاسخ i سؤال متوجه شود که آن شبکه همبند است یا خیر؛ به این معنی که آیا امکان رفتن از هر شهری به هر شهر دیگر وجود دارد یا خیر. در غیر این صورت (حالتی که می-یو به r سؤال نیاز پیدا کند) برنده‌ی بازی جیان-جیا خواهد بود.

برای این که بازی برای جیان-جیا جذاب‌تر شود، دو دوست توافق کردند که جیان-جیا شبکه‌ی پروازی تایوان را فراموش کند و در حین بازی شبکه را متناسب با سؤال‌های می-یو (سؤال‌های که تا کنون پرسیده شده است) ترسیم کند. وظیفه‌ی شما آن است که به جیان-جیا کمک کنید تا به گونه‌ای به سؤال‌ها پاسخ دهد که برنده‌ی بازی شود.

مثال

قواعد بازی را با سه مثال توضیح می‌دهیم. هر مثال شامل $n = ۴$ شهر و $r = ۶$ مرحله سؤال و پاسخ است.

در اولین مثال (جدول زیر)، جیان-جیا بعد از چهار مرحله می‌بازد، چرا که می-یو بعد از مرحله‌ی ۴ام مطمئن خواهد بود که شبکه همبند است صرف‌نظر از آنکه جیان-جیا به سؤال‌های ۵ و ۶ چه پاسخی دهد.

round	question	answer
1	0, 1	yes
2	3, 0	yes
3	1, 2	no
4	0, 2	yes
—	—	—
5	3, 1	no
6	2, 3	no

در مثال بعدی، می-یو بعد از مرحله‌ی سوم می‌تواند اثبات کند هر طور جیان-جیا به سؤال‌های ۴، ۵ و ۶ پاسخ دهد نمی‌توان از شهر ۰ به شهر ۱ رفت. بنابراین جیان-جیا بازی را دوباره می‌بازد.

round	question	answer
1	0, 3	no
2	2, 0	no
3	0, 1	no
—	—	—
4	1, 2	yes
5	1, 3	yes
6	2, 3	yes

در مثال آخر، می-یو تا پاسخ هر ۶ سؤال را دریافت نکند نمی‌تواند مشخص کند که شبکه همبند است یا خیر؛ بنابراین جیان-جیا برنده‌ی بازی خواهد بود. به‌طور مشخص، چون جیان-جیا به سؤال ۶ پاسخ «بله» داده است (طبق جدول زیر) شبکه همبند است. اگر پاسخ به این سؤال را «خیر» داده بود شبکه ناهمبند می‌شد.

round	question	answer
1	0, 3	no
2	1, 0	yes
3	0, 2	no
4	3, 1	yes
5	1, 2	no
6	2, 3	yes

مسئله

برنامه‌ای بنویسید که به جیان-جیا کمک کند تا برنده‌ی بازی شود. توجه کنید که می-یو و جیان-جیا از استراتژی یک‌دیگر مطلع نیستند. می-یو می‌تواند به هر ترتیب دل‌خواه در مورد زوج شهرها سؤال کند و جیان-جیا باید بلافاصله بعد از هر سؤال (بدون اطلاع از سؤال‌های بعدی می-یو) به او پاسخ «بله» یا «خیر» دهد. شما باید دو تابع زیر را پیاده‌سازی کنید.

- `initialize(n)`: ما ابتدا تابع `initialize` شما را فراخوانی می‌کنیم. پارامتر n تعداد شهرها است.
- `hasEdge(u, v)`: سپس ما تابع `hasEdge` را $r = n(n-۱)/۲$ بار فراخوانی می‌کنیم. این فراخوانی‌ها پاسخ سؤال‌های می-یو را به ترتیبی که پرسیده شده‌اند، مشخص می‌کند. شما باید مشخص کنید که بین شهرهای `u` و `v` پرواز مستقیم وجود دارد یا خیر. به طور مشخص، اگر پرواز مستقیم بین این دو شهر وجود داشت این تابع باید عدد ۱ و در غیر این‌صورت عدد ۰ را برگرداند.

زیرمسئله‌ها

هر زیرمسئله شامل چندین بازی است. شما امتیاز هر زیرمسئله را به شرطی دریافت می‌کنید که همه‌ی بازی‌ها را به نفع جیان-جیا تمام کنید.

subtask	point	n
1	15	$n = 4$
2	27	$4 \leq n \leq 80$
3	58	$4 \leq n \leq 1500$

جزئیات پیاده‌سازی

شما باید دقیقا یک فایل به نام `game.c`، `game.cpp` یا `game.pas` ارسال کنید. در این فایل زیربرنامه‌های فوق باید به شکل زیر پیاده‌سازی شوند.

برنامه‌ی C/C++

```
void initialize(int n);
int hasEdge(int u, int v);
```

برنامه‌ی پاسکال

```
procedure initialize(n: longint);
function hasEdge(u, v: longint): longint;
```

ارزیاب نمونه

ارزیاب نمونه ورودی را در قالب زیر می‌خواند.

- خط ۱: `n`
- در هر یک از `r` خط بعد: هر خط شامل دو عدد صحیح `u` و `v` است که به معنی سؤال در مورد شهرهای `u` و `v` است.

دیوار

جیان-جیا می‌خواهد با روی هم چیدن تعدادی آجر هم‌اندازه یک دیوار بسازد. این دیوار از n ستون آجری تشکیل شده است که به ترتیب از چپ به راست با شماره‌های ۰ تا $n-1$ شماره‌گذاری شده‌اند. ستون‌ها ممکن است ارتفاع‌های متفاوتی داشته باشند. ارتفاع یک ستون برابر تعداد آجرهای به‌کاررفته در آن است.

جیان-جیا دیوار را به این شکل می‌سازد. در ابتدا همه‌ی ستون‌ها بدون آجر هستند. سپس جیان-جیا k مرحله‌ی **افزودن** یا **حذف** آجرها را انجام می‌دهد، و پس از اتمام این k مرحله فرایند ساخت دیوار به پایان می‌رسد. در هر مرحله، یک بازه از ستون‌های متوالی و یک ارتفاع h به جیان-جیا داده می‌شود و او کارهای زیر را انجام می‌دهد:

- در مرحله‌ی **افزودن**، جیان-جیا به ستون‌هایی در بازه‌ی داده‌شده که کم‌تر از h آجر دارند آن‌قدر آجر اضافه می‌کند تا ارتفاع آن‌ها دقیقا h شود. او روی ستون‌هایی که بیش‌تر یا مساوی h آجر دارند کاری انجام نمی‌دهد.
- در مرحله‌ی **حذف**، جیان-جیا از ستون‌هایی در بازه‌ی داده‌شده که بیش‌تر از h آجر دارند آن‌قدر آجر حذف می‌کند تا ارتفاع آن‌ها دقیقا h شود. او روی ستون‌هایی که کم‌تر یا مساوی h آجر دارند کاری انجام نمی‌دهد.

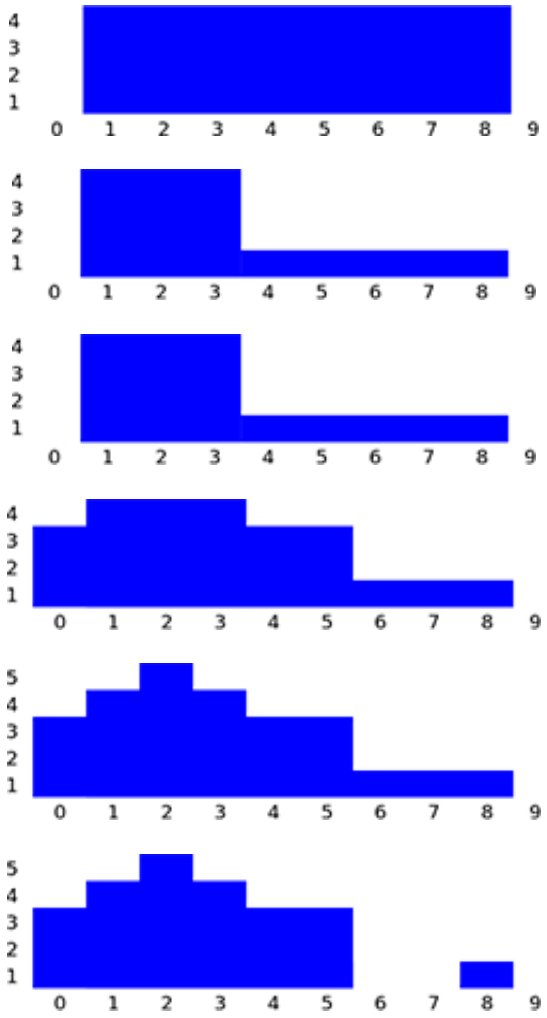
وظیفه‌ی شما تعیین شکل نهایی دیوار است.

مثال

فرض کنید ۱۰ ستون آجری و ۶ مرحله‌ی ساخت دیوار داریم. بازه‌های ذکرشده در جدول زیر بسته‌اند (یعنی شامل ابتدا و انتهای بازه‌ها می‌باشند). شکل دیوار پس از هر مرحله در صفحه‌ی بعد نشان داده شده است.

phase	type	range	height
0	add	columns 1 to 8	4
1	remove	columns 4 to 9	1
2	remove	columns 3 to 6	5
3	add	columns 0 to 5	3
4	add	column 2	5
5	remove	columns 6 to 7	0

از آن جایی که در ابتدا تمام ستون‌ها خالی هستند، پس از مرحله‌ی ۰، ستون‌های ۱ تا ۸ همگی ۴ آجر خواهند داشت و ستون‌های ۰ و ۹ خالی باقی می‌مانند. در مرحله‌ی ۱، از ستون‌های ۴ تا ۸ آن‌قدر آجر حذف می‌شود که هر یک از آن‌ها ۱ آجر داشته باشند، و ستون ۹ خالی باقی می‌ماند. ستون‌های ۰ تا ۳ که خارج از بازه‌ی داده‌شده‌اند، بدون تغییر باقی می‌مانند. مرحله‌ی ۲ تغییری ایجاد نمی‌کند زیرا ستون‌های ۳ تا ۶ بیش‌تر از ۵ آجر ندارند. پس از مرحله‌ی ۳، تعداد آجرها در ستون‌های ۰، ۴ و ۵ به ۳ افزایش می‌یابد. پس از مرحله‌ی ۴، تعداد آجرهای ستون ۲ برابر ۵ خواهد بود. مرحله‌ی ۵ تمام آجرها در ستون‌های ۶ و ۷ را حذف می‌کند.



مسئله

با داشتن توصیف k مرحله، تعداد آجرهای هر یک از ستون‌ها را پس از آن که تمامی مراحل خاتمه یافتند، محاسبه کنید. شما باید تابع `buildWall` را پیاده‌سازی کنید.

- `buildWall(n, k, op, left, right, height, finalHeight)`

- `n`: تعداد ستون‌های دیوار.
- `k`: تعداد مرحله‌ها.
- `op`: آرایه‌ای به طول k؛ `op[i]` نشان‌دهنده‌ی نوع مرحله‌ی i است: 1 برای مرحله‌ی افزودن و 2 برای مرحله‌ی حذف (به ازای $0 \leq i \leq k-1$).
- `left` و `right`: آرایه‌هایی به طول k؛ بازه‌ی ستون‌ها در مرحله‌ی i از ستون `left[i]` شروع شده و به ستون `right[i]` خاتمه می‌یابد ($0 \leq i \leq k-1$). این بازه شامل هر دو نقطه‌ی پایانی `left[i]` و `right[i]` می‌شود. همچنین همیشه داریم `left[i]` $\leq$ `right[i]`.
- `height`: آرایه‌ای به طول k؛ `height[i]` پارامتر ارتفاع مرحله‌ی i است ($0 \leq i \leq k-1$).
- `finalHeight`: آرایه‌ای به طول n برای برگرداندن نتایج؛ شما باید تعداد نهایی آجرها در ستون i را در درایه‌ی `finalHeight[i]` قرار دهید ($0 \leq i \leq n-1$).

زیرمسئله‌ها

برای تمام زیرمسئله‌ها، پارامتر ارتفاع در تمامی مرحله‌ها یک عدد صحیح نامنفی کم‌تر یا مساوی ۱۰۰,۰۰۰ است.

subtask	points	n	k	note
1	8	$1 \leq n \leq 10,000$	$1 \leq k \leq 5,000$	بدون هیچ محدودیت اضافی
2	24	$1 \leq n \leq 100,000$	$1 \leq k \leq 500,000$	تمامی مراحل افزودن پیش از تمامی مراحل حذف هستند
3	29	$1 \leq n \leq 100,000$	$1 \leq k \leq 500,000$	بدون هیچ محدودیت اضافی
4	39	$1 \leq n \leq 2,000,000$	$1 \leq k \leq 500,000$	بدون هیچ محدودیت اضافی

جزئیات پیاده‌سازی

شما باید دقیقا یک فایل با نام `wall.c`، `wall.cpp` یا `wall.pas` را ارسال کنید. این فایل باید تابعی که در بالا توضیح داده شد را به شکل زیر پیاده‌سازی کند. همچنین شما باید هدرفایل `wall.h` را در برنامه‌ی C/C++ خود درج کنید.

برنامه‌ی C/C++

```
void buildWall(int n, int k, int op[], int left[], int right[],
int height[], int finalHeight[]);
```

برنامه‌ی پاسکال

```
procedure buildWall(n, k : longint; op, left, right, height :
array of longint; var finalHeight : array of longint);
```

ارزیاب نمونه

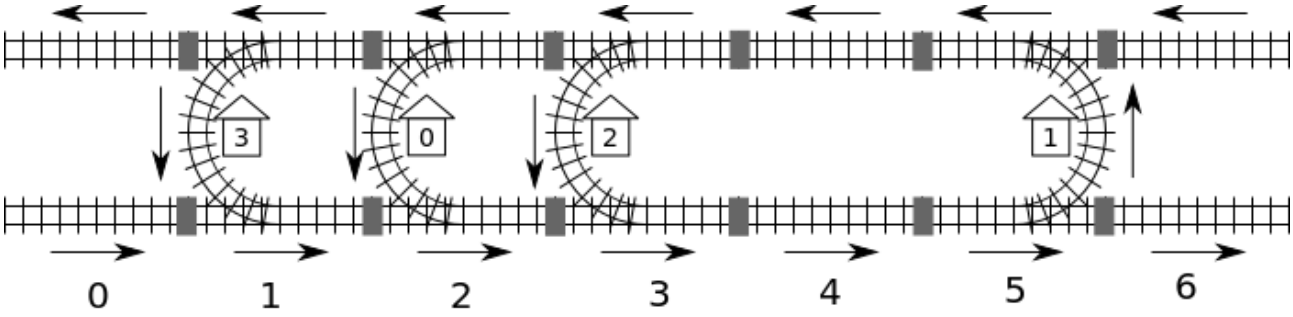
ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: `n`، `k`.
- خط $i + ۲$ ($۰ \leq i \leq k - ۱$): `op[i]`، `left[i]`، `right[i]`، `height[i]`.

ریل

تایوان یک خط آهن بزرگ دارد که سواحل غربی و شرقی جزیره را به هم متصل می‌کند. این خط شامل m بلوک است. این بلوک‌های متوالی به ترتیب از غرب به شرق با شماره‌های ۰ تا $m-1$ شماره گذاری شده‌اند. هر بلوک شامل یک ریل یک‌طرفه به سمت غرب در شمال و یک ریل یک‌طرفه به سمت شرق در جنوب است. همچنین در برخی از بلوک‌ها بین ریل‌های شمالی و جنوبی یک ایستگاه قطار واقع شده است.

سه نوع مختلف از بلوک‌ها وجود دارد. بلوک نوع C یک ایستگاه قطار دارد که باید از ریل شمالی وارد آن شده و از ریل جنوبی از آن خارج شد. بلوک نوع D یک ایستگاه قطار دارد که باید از ریل جنوبی وارد آن شده و از ریل شمالی از آن خارج شد. در بلوک‌های **خالی** ایستگاه قطاری وجود ندارد. به عنوان مثال، در شکل زیر بلوک‌های ۰، ۴ و ۶ از نوع خالی، بلوک‌های ۱، ۲ و ۳ از نوع C، و بلوک ۵ از نوع D می‌باشد. ریل‌های بلوک‌های مجاور از طریق بست‌هایی که در شکل به صورت مستطیل‌های خاکستری مشخص شده‌اند به یک‌دیگر وصل می‌شوند.



سیستم ریلی شامل n ایستگاه است که با شماره‌های ۰ تا $n-1$ شماره‌گذاری شده‌اند. فرض می‌کنیم که با استفاده از ریل‌ها از هر ایستگاهی به هر ایستگاه دیگری می‌توان رفت. به عنوان مثال، به این طریق می‌توانیم از ایستگاه ۰ به ایستگاه ۲ برویم: از بلوک شماره ۲ شروع کرده، با گذشتن از بلوک‌های ۳ و ۴ در ریل جنوبی، از طریق ایستگاه ۱ به ریل شمالی می‌رویم. سپس با ریل شمالی از بلوک ۴ عبور می‌کنیم و در نهایت به ایستگاه ۲ واقع در بلوک ۳ می‌رسیم.

از آن جایی که مسیرهای مختلفی بین دو ایستگاه وجود دارد، فاصله‌ی یک ایستگاه تا ایستگاه دیگر را برابر با کم‌ترین تعداد بست‌هایی که مسیر از آن‌ها می‌گذرد تعریف می‌کنیم. به عنوان مثال، کوتاه‌ترین مسیر از ایستگاه ۰ به ایستگاه ۲ به ترتیب (از چپ به راست) از بلوک‌های $3-4-5-4-3-2$ و از تعداد ۵ بست عبور می‌کند. در نتیجه فاصله‌ی ایستگاه ۰ تا ایستگاه ۲ برابر ۵ است.

یک سامانه‌ی کامپیوتری سیستم ریلی را مدیریت می‌کند. متأسفانه در اثر قطع برق، کامپیوتر دیگر نمی‌داند که ایستگاه‌ها در کدام بلوک‌ها واقع شده‌اند و بلوک مربوط به هر ایستگاه از چه نوعی است. تنها اطلاعاتی که کامپیوتر دارد شماره‌ی بلوک ایستگاه ۰ است که همیشه از نوع C می‌باشد. خوش‌بختانه کامپیوتر می‌تواند فاصله‌ی بین هر دو ایستگاه را بپرسد. به عنوان مثال کامپیوتر می‌تواند بپرسد: «فاصله‌ی ایستگاه ۰ تا ایستگاه ۲ چقدر است؟» و جواب بگیرد که این مقدار برابر ۵ است.

مسئله

شما باید تابع `findLocation` را پیاده سازی کنید که به ازای هر ایستگاه شماره‌ی بلوک و نوع آن را مشخص می‌کند.

- `findLocation(n, first, location, stype)`
  - `n`: تعداد ایستگاه‌ها.

- `first`: شماره‌ی بلوک ایستگاه ۰.
- `location`: آرایه‌ای به طول n؛ شما باید شماره‌ی بلوک ایستگاه i را در `location[i]` قرار دهید.
- `stype`: آرایه‌ای به طول n؛ شما باید نوع بلوک ایستگاه i را در `stype[i]` قرار دهید: برای نوع C از مقدار ۱ و برای نوع D از مقدار ۲ استفاده کنید.

شما می‌توانید یک تابع `getDistance` را صدا بزنید که به شما کمک می‌کند تا محل و نوع ایستگاه‌ها را بیابید.

- `getDistance(i, j)`: فاصله‌ی ایستگاه i تا ایستگاه j را بر می‌گرداند. `getDistance(i, i)` مقدار ۰ را برمی‌گرداند. در صورتی که i و j خارج از محدوده‌ی $0 \leq i, j \leq n-1$ باشند، `getDistance(i, j)` مقدار -1 را برمی‌گرداند.

زیرمسئله‌ها

در تمام زیرمسئله‌ها تعداد بلوک‌ها از ۱,۰۰۰,۰۰۰ بیش‌تر نیست. در بعضی از زیرمسئله‌ها تعداد دفعات فراخوانی تابع `getDistance` محدود است که بر اساس زیرمسئله تغییر می‌کند. در صورتی که برنامه‌ی شما از این محدودیت عبور کند، پیغام 'wrong answer' دریافت می‌کنید.

subtask	points	n	getDistance calls	note
1	8	$1 \leq n \leq 100$	unlimited	همه‌ی ایستگاه‌ها به جز ایستگاه ۰ در بلوک‌های نوع D هستند.
2	22	$1 \leq n \leq 100$	unlimited	همه‌ی ایستگاه‌های سمت راست ایستگاه ۰ در بلوک‌های نوع D هستند و همه‌ی بلوک‌های سمت چپ ایستگاه ۰ در بلوک‌های نوع C هستند.
3	26	$1 \leq n \leq 5,000$	$n(n-1)/2$	بدون هیچ محدودیت اضافی
4	44	$1 \leq n \leq 5,000$	$3(n-1)$	بدون هیچ محدودیت اضافی

جزئیات پیاده‌سازی

شما باید دقیقا یک فایل با نام `rail.c`، `rail.cpp` یا `rail.pas` را ارسال کنید. این فایل تابع `findLocation` را آن‌گونه که در بالا توضیح داده شد به شکل زیر پیاده‌سازی می‌کند. شما باید هدرفایل `rail.h` را در برنامه‌های C/C++ خود درج کنید.

برنامه‌ی C/C++

```
void findLocation(int n, int first, int location[], int stype[]);
```

برنامه‌ی پاسکال

```
procedure findLocation(n, first : longint; var location,
stype : array of longint);
```

تعریف تابع `getDistance` به شکل زیر است.

برنامه‌ی C/C++

```
int getDistance(int i, int j);
```

برنامه‌ی پاسکال

```
function getDistance(i, j: longint): longint;
```

ارزیاب نمونه

ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: شماره‌ی زیرمسئله.
- خط ۲: n
- خط $i + ۳$ ($۰ \leq i \leq n - ۱$): `stype[i]` (۱ برای نوع C و ۲ برای نوع D)، `location[i]`.

ارزیاب نمونه درصورتی که `location[0]` ... `location[n-1]` و `stype[0]` ... `stype[n-1]` که توسط برنامه‌ی شما محاسبه شده‌اند با ورودی مطابقت داشته باشد، عبارت `Correct` و در غیر این صورت عبارت `Incorrect` را چاپ می‌کند.