ماشین تحریر خرچنگ

بعضی از مردم معتقد هستند که «لئوناردو» یکی از طرفداران پر و پا قرص «یوهان گوتنبرگ» - آهنگر آلمانی که ماشین چاپ متحرک را اختراع کرد - بوده است، تا حدی که یک دستگاه که آن را «ماشین تحریر خرچنگ» می نامند طراحی کرد که در حقیقت یک ابزار تایپ بسیار ساده است. این دستگاه شبیه یک ماشین تایپ مدرن است و فقط دو دستور را می پذیرد: یکی برای تایپ حرف بعدی، و دیگری برای باطل (Undo) کردن دستورهای اخیر. خصوصیت قابل توجه ماشین تحریر خرچنگ آن است که دستور Undo خیلی قدرتمند است: یک Undo خودش هم به عنوان یک دستور تلقی میشود، و میتواند با دستور Undo دیگری ابطال گردد.

شرح مساله

شما میبایست یک نسخه نرم افزاری از ماشین تحریر خرچنگ را ایجاد کنید که با یک متن خالی شروع میکند، و به عنوان ورودی این موارد را دریافت میکند: یک توالی دستور که توسط کاربر وارد میشود، و همچنین جستار(Query) هایی برای تعیین حرف قرار گرفته در موقعیت های مشخصی از نسخه فعلی متن، به شرحی که گفته میشود.

- `Init()` — یک بار در شروع اجرای برنامه بدون هیچ پارامتری فراخوانی میشود. این دستور میتواند برای مقداردهی اولیه داده ساختارها مورد استفاده قرار گیرد. این دستور هرگز مورد ابطال (Undo) قرار نمیگیرد.
- `TypeLetter(L)` — یک حرف کوچک L را که از بین حرفهای z ،... ،a انتخاب میشود به انتهای متن اضافه میکند.
- `UndoCommands(U)` — به ازای عدد صحیح مثبت U، آخرین U دستور را ابطال میکند.
- `GetLetter(P)` — به ازای اندیس نامنفی P، حرف قرار گرفته در موقعیت P از نسخه کنونی متن را برمیگرداند. اولین حرف از متن اندیس صفر دارد. این جستار یک دستور نیست، لذا دستور ابطال (Undo) آن را نادیده می گیرد.

بعد از فراخوانی اولیه دستور `Init()`، سایر تابع ها میتوانند صفر مرتبه یا بیشتر به هر ترتیب دلخواهی فراخوانی شوند. تضمین میگردد که U هیچ وقت از تعداد دستورهایی که قبلا دریافت شده اند بیشتر نخواهد بود، و P همواره کمتر از طول (تعداد حروف) متن کنونی خواهد بود.

دستور `UndoCommands(U)`، تعداد U دستور قبل را به ترتیب *برعکس* ابطال میکند: اگر دستوری که قرار است باطل شود `TypeLetter(L)` باشد، این دستور حرف L را از انتهای متن کنونی حذف میکند. اگر دستوری که قرار است باطل شود `UndoCommands(X)` به ازای مقداری از X باشد، تعداد X دستور قبل از آن به ترتیب *اصلی*-شان بازیابی (Redo) میشوند.

مثال

ما یک توالی فراخوانی ها را به همراه وضعیت متن بعد از هر فراخوانی نمایش میدهیم.

فراخوانی	مقدار بازگشتی	متن کنونی
Init()		
TypeLetter(a)		a
TypeLetter(b)		ab
GetLetter(1)	b	ab
TypeLetter(d)		abd
UndoCommands(2)		a
UndoCommands(1)		abd
GetLetter(2)	d	abd
TypeLetter(e)		abde
UndoCommands(1)		abd
UndoCommands(5)		ab
TypeLetter(c)		abc
GetLetter(2)	c	abc
UndoCommands(2)		abd
GetLetter(2)	d	abd

زیر مساله ۱ [۵ نمره]

تعداد کل دستورها و جستارها از ۱ تا ۱۰۰ میباشد و دستور UndoCommands فراخوانی نمیگردد.

زیر مساله ۲ [۷ نمره]

تعداد کل دستورها و جستارها از ۱ تا ۱۰۰ میباشد و هیچ دستور UndoCommands ای ابطال (Undo) نمیگردد.

زیر مساله ۳ [۲۲ نمره]

تعداد کل دستورها و جستارها از ۱ تا ۵۰۰۰ خواهد بود.

زیر مساله ۴ [۲۶ نمره]

تعداد کل دستورها و جستارها از ۱ تا ۱ ۰۰۰ ۰۰۰ میباشد. همه فراخوانیهای تابع GetLetter بعد از همه فراخوانیهای توابع TypeLetter و UndoCommands خواهد بود.

زیر مساله ۵ [۴۰ نمره]

تعداد کل دستورها و جستارها از ۱ تا ۱ ۰۰۰ ۰۰۰ میباشد.

جزئیات پیاده سازی

شما باید دقیقا یک فایل را ارسال کنید که scrivener.cpp، scrivener.c یا scrivener.pas نام دارد. این فایل میبایست حاوی پیاده سازی زیربرنامه های فوق با قالب زیر باشد.

برنامه های C و C++

```
void Init();
void TypeLetter(char L);
void UndoCommands(int U);
char GetLetter(int P);
```

برنامه های پاسکال

```
procedure Init;
procedure TypeLetter(L : Char);
procedure UndoCommands(U : LongInt);
function GetLetter(P : LongInt) : Char;
```

این زیر-برنامه ها باید طبق آنچه در بالا گفته شد رفتار کنند. البته شما میتوانید زیر-برنامه های دیگری برای استفاده داخلی بنویسید. ارسال های شما نباید هیچ تعاملی با ورودی/خروجی استاندارد یا هیچ فایل دیگری داشته باشند.

ارزیابی نمونه

سیستم ارزیابی نمونه، ورودی را طبق قالب زیر میخواند:

- سطر ۱: تعداد کل دستورها و جستارهای ورودی
- در هریک از سطرهای بعدی:
  - `T` و پس از آن یک فاصله و یک حرف کوچک برای دستور `TypeLetter`
  - `U` و پس از آن یک فاصله و یک عدد صحیح برای دستور `UndoCommands`
  - `P` و پس از آن یک فاصله و یک عدد صحیح برای دستور `GetLetter`

سیستم نمره دهی کاراکترهایی که توسط `GetLetter` برگردانده شوند را - هرکدام در یک خط - چاپ میکند

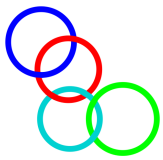
International Olympiad in Informatics 2012
September 2012 23-30
Sirmione - Montichiari, Italy
Competition tasks, day 1: Leonardo's inventions and projects

حلقه های نجات

یک نسخه بسیار پیشرفته از چیزی که ما الآن آن را چتر نجات می نامیم در کتاب Codex Atlantics لئوناردو شرح داده شده است. چتر نجات لئوناردو از یک پارچه کتانی بدون درز تشکیل گردیده که به یک سازه چوبی هرمی شکل بسته شده است.

حلقه های متصل به هم

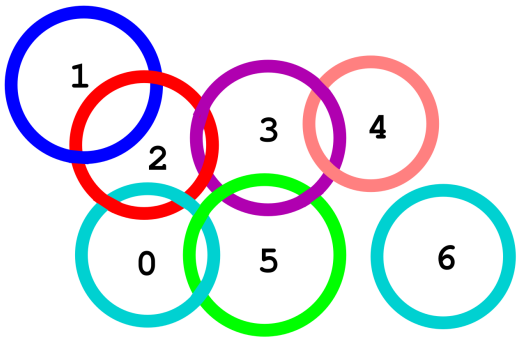
بیش از ۵۰۰ سال بعد، خلبان آدریان نیکلاس طراحی لئوناردو را آزمایش کرد. برای این کار یک سازه سبک پیشرفته چتر نجات لئوناردو را به بدن انسان گره زد. ما می خواهیم از حلقه های متصل به هم برای قلاب کردن پارچه های کتانی بدون درز استفاده کنیم. هر حلقه از ماده محکم قابل انعطافی ساخته شده است. با باز و بسته کردن حلقه ها می توان به سادگی آن ها را به یکدیگر متصل کرد. زنجیر یک شکل خاص از حلقه های متصل به هم است. همان طور که در شکل زیر نشان داده شده یک زنجیر شامل دنباله ای از حلقه هاست که در آن هر حلقه فقط به (حداکثر ۲) همسایه خود متصل است. دنباله باید یک حلقه شروع و یک حلقه پایان داشته باشد(حلقه هایی که حداکثر به یک حلقه دیگر متصل اند). یک حلقه تنها نیز یک زنجیر محسوب می شود.



از آن جایی که یک حلقه می تواند به سه یا تعداد بیشتری حلقه دیگر متصل باشد، واضح است که حلقه ها می توانند حالات دیگری جز زنجیر نیز داشته باشند. ما یک حلقه را *بحرانی* می نامیم اگر بعد از باز کردن و حذف آن، همه حلقه های باقی مانده به صورت مجموعه ای از زنجیرها درآیند یا این که هیچ حلقه دیگری باقی نمانده باشد.

مثال

هفت حلقه شکل زیر که با شماره های ۰ تا ۶ نمایش داده شده اند را در نظر بگیرید. در این میان دو حلقه بحرانی وجود دارد. یکی از آن ها حلقه شماره ۲ است که بعد از حذف آن حلقه های باقی مانده زنجیر های [1],[0,5,3,4] و [6] را تشکیل می دهند. حلقه بحرانی بعدی حلقه شماره ۳ است که پس از حذف آن حلقه های باقی مانده زنجیر های [1,2,0,5] ، [4] و [6] را تشکیل می دهند. اگر ما هر حلقه دیگری را حذف کنیم شکل به دست آمده تشکیل شده از زنجیر های جدا از هم نخواهد بود. برای مثال بعد از حذف حلقه شماره ۵ هر چند زنجیر [6] وجود دارد اما حلقه های ۰،۱،۲،۳ و ۴ که به هم متصل اند به صورت یک زنجیر نیستند.



شرح مساله

شما باید به ازای هر وضعیتی از حلقه ها که در اختیار برنامه شما قرار می گیرد، تعداد حلقه های بحرانی آن را بشمارید.

در ابتدای کار تعداد مشخصی حلقه وجود دارند که هیچ دو تایی از آن ها به هم متصل نیستند. بعد از آن، حلقه ها به هم متصل می شوند. در هر لحظه امکان دارد از برنامه شما تعداد حلقه های بحرانی وضعیتی که به دست آمده است پرسیده شود. به عبارت دقیق تر، شما باید این سه تابع را پیاده سازی کنید.

- `Init(N)` - این تابع دقیقاً یک بار در ابتدای کار صدا زده می شود و تعیین می کند که دقیقاً N حلقه جدا از هم با شماره های ۰ تا N-1 (شامل ۰ و N-1) در شکل اولیه وجود دارد.
- `Link(A,B)` - حلقه های شماره A و B به یکدیگر متصل می شوند. در این حالت تضمین شده است که A و B متفاوت اند و قبلاً مستقیماً به هم متصل نشده اند. به جز این، هیچ شرط اضافی و شرایط ناشی از محدودیت های فیزیکی برای حلقه های A و B وجود ندارد. بدیهی است که Link(A,B) با Link(B,A) هیچ تفاوتی ندارد.
- `CountCritical()` - این تابع تعداد حلقه های بحرانی وضعیت کنونی را برمی گرداند.

مثال

شکل فوق که در آن N برابر با ۷ است را در نظر بگیرید و فرض کنید که حلقه ها در ابتدا به یکدیگر متصل نیستند. ما یک سری از بازخوانی توابع به صورت قابل قبول را نشان می دهیم. پس از آخرین بازخوانی وضعیت به دست آمده منطبق را تصویر فوق است.

فراخوانی	مقدار بازگشتی
`Init(7)`	
`CountCritical()`	7
`Link(1, 2)`	
`CountCritical()`	7
`Link(0, 5)`	
`CountCritical()`	7
`Link(2, 0)`	
`CountCritical()`	7
`Link(3, 2)`	
`CountCritical()`	4
`Link(3, 5)`	
`CountCritical()`	3
`Link(4, 3)`	
`CountCritical()`	2

زیر مساله 1 [۲۰ نمره]

- $N \leq 5\,000$.
- تابع `CountCritical` تنها یک بار پس از دیگر فراخوانی ها، فراخوانی می شود؛ تابع `Link` حداکثر ۵۰۰۰ بار فراخوانی می شود.

زیر مساله ۲ [۱۷ نمره]

- $N \leq 1\,000\,000$.
- تابع `CountCritical` تنها یک بار پس از دیگر فراخوانی ها فراخوانی می شود؛ تابع `Link` حداکثر ۱ ۰۰۰ ۰۰۰ بار فراخوانی می شود.

زیر مساله ۳ [۱۸ نمره]

- $N \leq 20\,000$.
- تابع `CountCritical` حداکثر ۱۰۰ بار فراخوانی می شود؛ تابع `Link` حداکثر ۱۰ ۰۰۰ بار فراخوانی می شود.

زیر مساله ۴ [۱۴ نمره]

- $N \leq 100\,000$.
- تعداد فراخوانی های تابع `CountCritical` و `Link` در مجموع حداکثر ۱۰۰ ۰۰۰ است.

زیر مساله ۵ [۳۱ نمره]

- $N \leq 1\,000\,000$.
- توابع `CountCritical` و `Link` در مجموع حداکثر ۱ ۰۰۰ ۰۰۰ بار فراخوانی می شوند.

جزئیات پیاده سازی

شما باید دقیقاً یک فایل را ارسال کنید که `rings.cpp` ، `rings.c` یا `rings.pas` نام دارد. در این فایل باید زیر برنامه ای که در بالا شرح داده شد، با قالب زیر پیاده سازی شده باشد.

برنامه های C و C++

```
void Init(int N);
void Link(int A, int B);
int CountCritical();
```

برنامه های پاسکال

```
procedure Init(N : LongInt);
procedure Link(A, B : LongInt);
function CountCritical() : LongInt;
```

این سه زیر برنـامه باید به همـان صورت که توضـیح داده شد رفتار کنند. البته شما می توانید برای استفاده داخلی خود، زیر برنـامه هـای دیگری نیز تعریف کنید. برنـامه ارسـالی شما نباید به هیـچ وجه با ورودی و خروجی استاندارد و یا هیـچ فایلی تعامل داشته باشد.

سیستم ارزیابی نمونه

سیستم ارزیابی نمونه ورودی را طبق قالب زیر میخواند:

- خط ۱: N, L (ابتدا N)
- خطوط ۲, ..., L + 1:
  - 1- برای فراخوانی `CountCritical`
  - پارامتر های A و B برای `Link`.

سیستم نمره دهی تمام نتایج مربوط به `CountCritical` را در خروجی چاپ می کند.

کیلومترشمار سنگ‌ریزه‌ای

دستگاه *کیلومترشمار* بوسیله لئوناردو اختراع شده است: این دستگاه به ازای هر دور کامل چرخ ماشین یک سنگ‌ریزه بیرون می‌اندازد. تعداد سنگ‌ریزه‌ها مشخص می‌کند که چرخ چند دور کامل زده‌است، که بر این اساس کاربر امکان تعیین مسافت پیموده شده را پیدا میکند. ما بعنوان دانشمندان کامپیوتر، یک واحد کنترل نرم‌افزاری به دستگاه کیلومترشمار اضافه کرده ایم تا قابلیت‌های ان را افزایش دهیم. وظیفه شما برنامه‌ریزی دستگاه کیلومترشمار تحت قوانین زیر است.

عملیات‌های مجاز بر روی گرید

دستگاه کیلومترشمار بر روی یک گرید (صفحه) مربعی شکل شامل 256 × 256 خانه حرکت می‌کند. هر خانه می‌تواند حداکثر 15 سنگ ریزه را در خود جا دهد، و بوسیله یک زوج مختصات (Row, Column) - (ستون، سطر) مشخص می‌شود که هر مختصات عددی صحیح در بازه 0, ..., 255 است. همسایه‌های خانه (i, j) در صورت وجود عبارتند از (i - 1, j)، (i, j - 1)، (i + 1, j) و (i, j + 1). هر خانه‌ای که در سطر اول یا آخر، یا در ستون اول یا آخر قرار دارد، خانه مرزی نامیده می‌شود. دستگاه کیلومترشمار همیشه از خانه (0, 0) (گوشه شمالی-غربی) شروع به حرکت می‌کند و در آغاز حرکت رو به شمال ایستاده‌است.

دستورهای اولیه

دستگاه کیلومترشمار بوسیله یکی از دستورهای زیر قابل برنامه‌ریزی است.

- `left` — نود درجه به چپ بچرخ (در خلاف عقربه‌های ساعت) و در خانه جاری باقی بمان (بعنوان مثال، اگر دستگاه کیلومتر شمار قبل از اجرای دستور به سمت جنوب ایستاده باشد با اجرای این دستور رو به شرق خواهد ایستاد).
- `right` — نود درجه به راست بچرخ (در جهت عقربه‌های ساعت) و در خانه جاری باقی بمان (بعنوان مثال، اگر دستگاه کیلومترشمار قبل از اجرای دستور رو به سمت غرب ایستاده است، بعد از اجرای دستور رو به سمت شمال خواهد ایستاد).
- `move` — یک خانه رو به جلو (در جهتی که کیلومترشمار ایستاده است) حرکت کن و به خانه مجاور برو. اگر چنین خانه‌ای موجود نباشد (دستگاه کیلومترشمار قبلا در خانه مرزی همان جهت قرار گرفته باشد) این دستور بی تاثیر است
- `get` — یک سنگ‌ریزه را از خانه کنونی حذف کن. اگر خانه کنونی سنگ‌ریزه نداشته باشد، این دستور بی‌تاثیر خواهد بود.
- `put` — یک سنگ‌ریزه به خانه کنونی اضافه کن. اگر خانه جاری در حال حاضر 15 سنگ‌ریزه داشته باشد این دستور بی‌تاثیر خواهد بود. هیچ گاه سنگ‌ریزه‌های کیلومترشمار تمام نمی‌شود.
- `halt` — به اجرا خاتمه بده.

دستگاه کیلومترشمار دستورها را به ترتیب داده شده در برنامه اجرا می‌کند. هر سطر برنامه باید حداکثر شامل یک دستور باشد. خطهای خالی نادیده گرفته می‌شود. علامت # نشاندهنده شروع یک کامنت است؛ هر متنی بعد از این علامت تا انتهای

خط نادیده گرفته خواهد شد. اگر دستگاه کیلومترشمار به انتهای برنامه برسد، اجرا خاتمه پیدا خواهد کرد.

مثال ۱

برنامه زیر را که برای دستگاه کیلومترشمار نوشته شده در نظر بگیرید. این برنامه دستگاه کیلومترشمار را به خانه $(0, 2)$ هدایت می‌کند و جهت قرارگیری ان نهایتا به سمت شرق خواهد بود. (توجه کنید دستور `move` اجرا نخواهد شد چرا که در حالت اغازین کیلومترشمار در گوشه شمالی-غربی رو به شمال ایستاده است.)

```
move  # بدون تاثیر
right
# حالا دستگاه کیلومترشمار رو به شرق ایستاده است
move
move
```

برچسب‌ها، خانه‌های مرزی و سنگ‌ریزه‌ها

برای آنکه بتوان روند اجرای دستورها را براساس شرایط تغییر داد، می‌توانید از برچسب‌ها که رشته‌هایی حساس به حروف کوچک-بزرگ و به طول حداکثر 128 حرف شامل `a,...,z, A,...,Z, 0,...,9` هستند، استفاده کنید. دستورهای پرشی که بر اساس برچسب ها کار میکنند در زیر فهرست شده‌اند. در تمامی توضیحات زیر L یک برچسب معتبر را نشان می‌دهد.

- L **:** (یعنی L که بعد از آن دونقطه آمده است) — مکان برچسب را در برنامه مشخص می‌کند. برچسب‌ها باید یکتا باشند. تعریف یک برچسب تاثیری بر دستگاه کیلومترشمار نخواهد گذاشت.
- `jump` L — اجرا را با پرش بدون شرط به مکانی از برنامه که برچسب L قرار گرفته ادامه می‌دهد.
- `border` L — اگر دستگاه کیلومترشمار در خانه مرزی و رو به لبه صفحه ایستاده باشد (یعنی اجرای دستور `move` بی‌تاثیر باشد) اجرای برنامه با پرش به برچسب L ادامه پیدا می‌کند. در غیر این صورت اجرای برنامه به صورت عادی ادامه می یابد و این دستور بی تاثیر است.
- `pebble` L — اگر خانه کنونی حداقل یک سنگریزه داشته باشد، اجرای برنامه با پرش به برچسب L ادامه می یابد. در غیر این صورت برنامه روال عادی اجرای خود را خواهد داشت و این دستور بی تاثیر خواهد بود.

مثال 2

برنامه زیر نخستین (غربی ترین) سنگریزه را از ردیف 0 پیدا میکند و در آن خانه متوقف میشود. اگر هیچ سنگریزه ای در سطر 0 نباشد، محل توقف دستگاه روی مرز انتهای این ردیف خواهد بود. برنامه از دو برچسب `leonardo` و `davinci` استفاده می‌کند.

```
right
leonardo:
pebble davinci  # سنگریزه پیدا شد
border davinci  # انتهای ردیف
move
jump leonardo
davinci:
halt
```

دستگاه کیلومترشمار حرکتش را با چرخش به راست اغاز می‌کند. حلقه با تعریف برچسب `leonardo:` شروع و با دستور پرش `jump leonardo` پایان می‌پذیرد. در این حلقه دستگاه کیلومترشمار موجود بودن سنگریزه یا قرارگیری در مرز انتهای ردیف را بررسی می‌کند؛ اگر این گونه نبود، دستگاه کیلومترشمار با اجرای دستور `move` از خانه جاری $(0, j)$ به خانه مجاور $(0, j + 1)$ می‌رود. (اجرای دستور `halt` در این مثال لزوما مورد نیاز نیست چرا که به هرحال برنامه متوقف خواهد شد.)

شرح مساله

شما باید یک برنامه به زبان دستگاه کیلومترشمار همانطور که در بالا توضیح داده شد ارسال کنید که اجرای آن باعث شود دستگاه طوری رفتار کند که مورد انتظار است. هر زیرمساله داده شده، رفتار خواسته شده از دستگاه کیلومترشمار را به همراه محدودیت‌هایی که راه حل ارسالی میبایست برآورده کند دربر دارد. محدودیت‌ها میتواند شامل دو موضوع زیر باشد:

- *اندازه برنامه* — برنامه باید به اندازه کافی کوتاه باشد. اندازه یک برنامه، تعداد دستورهای آن است. تعریف برچسب‌ها، کامنت‌ها و خطهای خالی در محاسبه اندازه برنامه لحاظ *نخواهند شد*.
- *طول اجرا* — برنامه باید به اندازه کافی سریع خاتمه پیدا کند. طول اجرا برابر تعداد *گام‌های* برنامه می‌باشد: هر اجرای یک دستور یک گام شمرده می‌شود صرفنظر از انکه ان دستور تاثیر داشته یا نداشته باشد. تعریف برچسب‌ها، کامنت‌ها و خطوط خالی به عنوان گام لحاظ نخواهند شد.

در مثال ۱، اندازه برنامه و طول اجرا هر کدام ۴ می‌باشد. در مثال ۲، اندازه برنامه ۶ می‌باشد و طول اجرا روی یک صفحه با یک سنگ ریزه در خانه (0, 10) برابر با ۴۳ گام بدین ترتیب می‌باشد: `right` ، ده بار تکرار حلقه که هر تکرار شامل ۴ گام است (`pebble davinci; border davinci; move; jump leonardo`) و نهایتا `pebble davinci`و `halt`

زیر مساله ۱ [۹ نمره]

در آغاز x سنگ‌ریزه در خانه (0, 0) و y سنگ‌ریزه در خانه (0, 1) قرار دارد و بقیه خانه‌ها خالی می‌باشند. دقت کنید که در هر خانه حداکثر ۱۵سنگ‌ریزه می‌تواند وجود داشته باشد. برنامه‌ای بنویسید که به شرط انکه $x \leq y$ است در خانه (0, 0) متوقف شود و در غیر اینصورت در خانه (0, 1) متوقف شود. (جهت ایستادن دستگاه کیلومترشمار در پایان حرکت اهمیتی ندارد. همچنین اهمیتی ندارد که چند تا سنگ‌ریزه در صفحه و در کجا باقی مانده است.)

محدودیت ها: اندازه برنامه میبایست حداکثر 100 و طول اجرا حداکثر 1000 باشد.

زیر مساله ۲ [۱۲ نمره]

مشابه زیرمساله بالا، فقط در خاتمه برنامه خانه های (0, 0) و (0, 1) باید به ترتیب دقیقا x و y سنگ‌ریزه داشته باشند.

محدودیت‌ها: اندازه برنامه حداکثر 200 و طول اجرا حداکثر 2 000 باشد.

زیر مساله ۳ [۱۹ نمره]

دقیقا دو سنگ‌ریزی در ردیف 0 موجود است: یکی در خانه (0, x) و دیگری در خانه (0, y)؛ x و y متمایز هستند و $x + y$ عددی زوج است. برنامه‌ای بنویسید که دستگاه کیلومترشمار را در خانه $(0, (x + y) / 2)$ متوقف کند (یعنی در خانه وسط بین دو خانه‌ای که سنگ‌ریزه دارند). حالت نهایی صفحه اهمیتی ندارد.

محدودیت‌ها: اندازه برنامه حداکثر 100 و طول اجرا حداکثر 200 000 باشد.

زیرمساله ۴ [حداکثر ۳۲ نمره]

حداکثر 15 سنگ‌ریزه در صفحه موجود است که هیچ دوتایی در یک خانه قرار ندارند. برنامه‌ای بنویسید که همه آنها را در گوشه شمالی-غربی جمع کند. به عبارت دقیق‌تر، اگر در ابتدا x سنگ‌ریزه در صفحه موجود است، در انتها باید x سنگریزه در خانه (0, 0) قرار داشته باشد و بقیه خانه‌ها خالی باشند.

امتیاز این زیرمساله بستگی به طول اجرای برنامه ارسال شده دارد. به عبارت دقیق‌تر، اگر L طول اجرای بیشینه روی ورودی های مختلف باشد، امتیاز شما اینگونه محاسبه خواهد شد:

- 32 امتیاز اگر $L \leq 200\,000$
- $32 - 32 \log_{10} (L / 200\,000)$ نمره اگر $200\,000 < L < 2\,000\,000$
- 0 نمره اگر $L \geq 2\,000\,000$

محدودیت‌ها: اندازه برنامه حداکثر 200 باشد.

زیرمساله ۵ [حداکثر ۲۸ نمره]

هرتعدادی سنگ ریزه می‌توانند در هر خانه موجود باشد (البته بین 0 تا 15). برنامه ای بنویسید که خانه‌ای که کمترین تعداد سنگ‌ریزه را دارد پیدا کند، به این معنی که با خاتمه اجرای برنامه، دستگاه کیلومترشمار در خانه ای قرار بگیرد که تعداد سنگ‌ریزه‌هایش کوچکتر یا مساوی تعداد سنگ‌ریزه‌های بقیه خانه‌هاست. تعداد سنگ‌ریزه‌ها در هر خانه قبل و بعد از اجرای برنامه باید یکسان باشد.

نمره این زیرمساله بستگی به اندازه برنامه ارسالی P دارد. به عبارت دقیق‌تر، نمره شما اینگونه محاسبه خواهد شد:

- 28 نمره اگر $P \leq 444$
- $28 - 28 \log_{10} (P / 444)$ نمره اگر $444 < P < 4\,440$
- 0 نمره اگر $P \geq 4\,440$

محدودیت‌ها: طول اجرا حداکثر 44 400 000 باشد.

جزئیات پیاده‌سازی

شما باید به ازای هر زیرمساله یک فایل که براساس قواعد بالا نوشته شده ارسال کنید. اندازه هر فایل حداکثر می تواند ۵ میلیون بایت باشد. برای هر زیرمساله، برنامه شما روی چندین تست‌دیتا اجرا خواهد شد و نتیجه برنامه را بر اساس منابعی که کد شما استفاده کرده دریافت خواهید کرد. در حالتی که برنامه شما اشکال گرامری داشته باشد، اطلاعات خطای گرامری مربوطه به شما داده خواهد شد.

نیازی نیست که ارسال شما شامل رامحل تمام زیرمساله‌ها باشد. اگر ارسال کنونی شما حاوی برنامه برای زیرمساله X نباشد، نتیجه اخرین ارسال شما برای زیرمساله X به صورت خودکار منظور خواهد شد. اگر چنین ارسالی موجود نباشد، نمره اختصاص یافته به این زیرمساله صفر خواهد بود.

بطور معمول، نمره هر ارسال برابر مجموع نمرات زیرمساله‌ها خواهد بود و نمره نهایی برابر بیشینه نمره بین تمام ارسالهای تست شده و همچنین اخرین ارسال خواهد بود.

شبیه‌ساز

به منظور تست برنامه، یک شبیه‌ساز برای شما در نظر گرفته شده است که شما می‌توانید برنامه خود همراه با صفحه ورودی را به آن بدهید. برنامه دستگاه کیلومترشمار به همان قالبی که برای ارسال مورد استفاده قرار میگیرد نوشته خواهد شد.

توصیف صفحه ورودی به صورت زیر خواهد بود: هر خط فایل باید شامل سه عدد R, C, P باشد (R نخستین عدد) به این

معنی که خانه با ردیف R و ستون C دارای P سنگ‌ریزه است. بقیه خانه‌ها که در فایل نیامده‌اند، سنگ‌ریزه‌ای نخواهند داشت. برای مثال، فایل زیر را در نظر بگیرید:

```
0 10 3
4 5 12
```

این صفحه شامل 15 سنگ‌ریزه است: سه تا در خانه (0, 10) و 12 تا در خانه (4, 5).

شما می‌توانید شبیه‌ساز را با فراخوانی برنامه `simulator.py` که در فولدر مسایل قرار دارد، و دادن نام فایل به عنوان ارگومان ورودی اجرا کنید. برنامه شبیه‌ساز گزینه‌های زیر را قبول می‌کند:

- `-h` – توضیح مختصری در مورد گزینه‌ها می‌دهد.
- `-g GRID_FILE` – توصیف صفحه را از فایل `GRID_FILE` بار گذاری می‌کند. (پیش‌فرض: صفحه خالی)
- `-s GRID_SIDE` – اندازه صفحه را `GRID_SIDE x GRID_SIDE` می‌گذارد. (پیش‌فرض: 256 همانطور که در مساله گفته شده است)؛ استفاده از اندازه های کوچکتر به رفع اشکال برنامه کمک می‌کند.
- `-m STEPS` – تعداد گام‌های اجرا در شبیه‌سازی را به حداکثر `STEPS` محدود می‌کند.
- `-c` – وارد حالت کامپایل کردن می‌شود. در این حالت، شبیه‌ساز دقیقا خروجی مشابه برمی‌گرداند با این تفاوت که به جای انجام شبیه‌سازی با Python، یک برنامه کوچک C تولید و کمپایل می‌کند. این باعث سربار بیشتر در هنگام شروع به اجرا شده اما در ادامه نتایج را بسیار سریعتر ارایه می‌کند. پیشنهاد می شود زمانی از این حالت استفاده کنید که طول اجرای برنامه شما بیش از حدود ده میلیون گام است.

تعداد ارسال‌ها

تعداد ارسال‌های مجاز برای این مساله حداکثر 128 می‌باشد.