Pebbling odometer

Leonardo menemukan *odometer* pertama: sebuah kereta yang dapat digunakan untuk mengukur jarak dengan melepas kerikil sembari roda kereta itu berputar. Dengan menghitung banyaknya kerikil, dapat diketahui banyak putaran roda, sehingga pengguna dapat menghitung jarak yang sudah ditempuh oleh odometer. Sebagai ilmuwan komputer, kita menambahkan perangkat lunak untuk mengendalikan odometer sehingga kegunaannya bertambah. Tugas Anda adalah memprogram odometer berdasarkan aturan yang dijelaskan di bawah ini.

Operation grid

Odometer ini bergerak pada sebuah grid persegi imajiner berukuran 256 x 256 satuan sel. Setiap sel dapat memiliki paling banyak 15 buah kerikil yang diidentifikasi dengan pasangan koordinat (baris, kolom), di mana setiap koordinat berada pada rentang 0, ..., 255. Untuk sebuah sel (i, j), sel-sel yang bersebelahan dengannya antara lain (apabila mereka ada) (i - 1, j), (i + 1, j), (i, j - 1) dan (i, j + 1). Sel manapun yang berada pada baris pertama atau terakhir, atau pada kolom pertama atau terakhir, disebut sebagai *border*. Odometer tersebut selalu mulai pada sel (0, 0) (pojok utara-barat/barat laut), menghadap ke arah utara.

Perintah-perintah dasar

Odometer ini dapat diprogram dengan perintah-perintah berikut.

- `left` — berputar 90 derajat ke kiri (berlawanan arah jarum jam) dan tetap berada pada sel yang sedang ditempati (sebagai contoh, apabila sebelumnya ia sedang menghadap ke selatan, ia akan menghadap ke arah timur setelah perintah ini).
- `right` — berputar 90 derajat ke kanan (searah jarum jam) dan tetap berada di sel yang sedang ditempati (sebagai contoh, apabila ia sedang menghadap ke arah barat sebelumnya, maka ia akan menghadap ke utara setelah perintah ini).
- `move` — berpindah sebanyak satu satuan ke depan (ke arah yang mana odometer sedang menghadap) ke sel yang bersebelahan. Apabila tidak ada sel yang demikian (yakni dia sudah mencapai border pada arah tersebut) maka perintah ini tidak memiliki efek.
- `get` — hapus sebuah kerikil dari sel yang sedang ditempati. Apabila sel yang sedang ditempati tidak memiliki kerikil, maka perintah ini tidak memiliki efek.
- `put` — tambahkan sebuah kerikil pada sel yang sedang ditempati. Apabila sel yang sedang ditempati sudah memiliki 15 kerikil, maka perintah ini tidak memiliki efek. Odometer tidak akan pernah kehabisan kerikil.

- `halt` — mengakhiri eksekusi.

Odometer tersebut mengeksekusi perintah-perintah berdasarkan urutan yang diberikan pada program. Program tersebut harus mengandung paling banyak sebuah perintah per baris. Baris-baris kosong diabaikan. Simbol `#` menandakan sebuah komentar; teks apapun yang mengikutinya, hingga akhir baris, diabaikan. Apabila odometer ini mencapai akhir program, eksekusi berakhir.

Contoh 1

Perhatikan program berikut untuk sebuah odometer. Program ini memindahkan odometer ke sel (0, 2), sehingga menghadap ke timur. (Perhatikan bahwa perintah `move` pertama diabaikan, karena odometer tersebut sedang berada pada pojok barat laut menghadap ke arah utara.)

```
move  # tidak ada efek
right
# sekarang odometer tersebut menghadap ke timur
move
move
```

Label, border, dan pebble

Untuk mengubah aliran program berdasarkan status saat ini, Anda dapat menggunakan label, yang merupakan untaian case-sensitive yang terdiri dari paling banyak 128 simbol yang dipilih dari `a`, ..., `z`, `A`, ..., `Z`, `0`, ..., `9`. Perintah-perintah baru mengenai label didaftarkan sebagai berikut. Pada deskripsi di bawah ini, *`L`* menyatakan label apapun yang valid.

- *`L`*`:` (i.e. *`L`* diikuti dengan sebuah titik dua '`:`') — menyatakan lokasi pada program yang berlabel *`L`*. Semua label yang terdeklarasi harus unik. Mendeklarasi sebuah label tidak memiliki efek pada odometer.
- `jump` *`L`* — melanjutkan eksekusi dengan melompat tanpa syarat ke baris berlabel *`L`*.
- `border` *`L`* — melanjutkan eksekusi dengan melompat ke baris berlabel *`L`*, apabila odometer berada pada border menghadap ke batas grid (yakni instruksi `move` tidak memiliki efek); selain itu, eksekusi berlanjut secara normal dan perintah tidak memiliki efek.
- `pebble` *`L`* — melanjutkan eksekusi dengan melompat ke baris berlabel *`L`*, apabila sel yang sedang ditempati mengandung setidaknya sebuah kerikil; selain itu, eksekusi berlanjut secara normal dan perintah ini tidak memiliki efek.

Contoh 2

Program di bawah ini melacak kerikil pertama (paling barat) di baris 0 dan berhenti di sana; apabila tidak ada kerikil di baris 0, ia berhenti pada *border* pada akhir baris tersebut. Program ini menggunakan dua label `leonardo` dan `davinci`.

```
right
leonardo:
pebble davinci  # pebble ditemukan
border davinci  # akhir dari baris
move
jump leonardo
davinci:
halt
```

Odometer ini mulai dengan berbelok ke kanan. Loop dimulai dengan deklarasi label `leonardo:` dan berakhir dengan perintah `jump leonardo`. Di dalam loop tersebut, odometer mengecek keberadaan sebuah kerikil atau *border* pada akhir dari baris; apabila tidak demikian, odometer tersebut melakukan sebuah `move` dari sel saat ini (0, j) ke sel sebelah (0, j + 1) jika ada. (perintah `halt` tidak harus ada di sini karena bagaimana pun tersebut tetap akan berhenti.)

Statement

Anda harus mengumpulkan sebuah program dalam bahasa odometer, seperti yang dijelaskan di atas, yang menyebabkan odometer berperilaku seperti yang diharapkan. Setiap subtask (perhatikan di bawah ini) menspesifikasikan sebuah perilaku odometer yang diperlukan dan batasan yang harus dipenuhi oleh solusi yang dikumpulkan. Batasan menyangkut dua hal berikut.

- *Ukuran program* — program tersebut harus cukup singkat. Ukuran dari sebuah program adalah banyaknya perintah pada program tersebut. Deklarasi label, komentar, dan baris kosong "tidak diperhitungkan" pada ukuran.
- *Lama eksekusi* — program harus berhenti cukup cepat. Lama eksekusi adalah banyaknya "langkah" yang dilakukan: setiap eksekusi sebuah perintah dihitung sebagai satu langkah, tanpa memperhitungkan apakah perintah tersebut memiliki efek atau tidak; deklarasi label, komentar dan baris kosong tidak dihitung sebagai langkah.

Pada Contoh 1, ukuran program adalah 4 dan lama eksekusi adalah 4. Pada Contoh 2, ukuran program adalah 6 dan, ketika dieksekusi pada sebuah grid dengan sebuah kerikil pada sel (0, 10), lama eksekusi adalah 43 langkah: `right`, 10 iterasi pada loop, setiap iterasi terdiri dari 4 langkah (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), dan terakhir, `pebble davinci` dan `halt`.

Subtask 1 [9 points]

Pada mulanya terdapat x kerikil pada sel (0, 0) dan y pada sel (0, 1), sedangkan semua sel lainnya kosong. Ingat bahwa bisa ada maksimum 15 kerikil pada sel manapun. Tulis sebuah program yang berhenti dengan odometer berada pada sel (0, 0) jika $x \leq y$, dan pada sel (0, 1) jika sebaliknya. (Kita tidak mempedulikan ke mana arah odometer pada akhirnya; kita juga tidak peduli tentang banyaknya kerikil pada saat itu di akhir grid, atau di mana mereka ditemukan.)

Batasan: ukuran program ≤ 100, lama eksekusi $\leq 1\,000$.

Subtask 2 [12 points]

Sama dengan subtask 1 tetapi saat program berakhir, sel (0, 0) harus berisi tepat x kerikil dan sel (0, 1) harus berisi tepat y kerikil.

Batasan: ukuran program ≤ 200, lama eksekusi ≤ 2 000.

Subtask 3 [19 points]

Ada tepat dua kerikil pada baris 0: salah satu berada pada sel (0, x), yang lain pada sel (0, y); x tidak sama dengan y, dan x + y genap. Tulislah sebuah program yang menyebabkan odometer berada pada sel (0, (x + y) / 2), yaitu, tepat di titik tengah antara kedua sel yang berisi kerikil. State akhir dari grid tidak penting.

Batasan: ukuran program ≤ 100, lama eksekusi ≤ 200 000.

Subtask 4 [up to 32 points]

Ada paling banyak 15 kerikil pada grid, tanpa ada dua yang berada di sel yang sama. Tulislah sebuah program yang mengumpulkan semua kerikil itu ke ujung barat laut; lebih tepatnya, jika ada x buah kerikil di grid pada awalnya, pada akhirnya harus ada tepat x kerikil pada sel (0, 0) dan tidak ada kerikil di tempat lain.

Nilai dari subtask ini tergantung pada lama eksekusi dari program yang dikumpulkan. Lebih tepatnya, jika L adalah maksimum dari lama eksekusi pada berbagai test case, nilai Anda adalah:

- 32 poin jika L ≤ 200 000;
- 32 - 32 $\log_{10}$ (L / 200 000) poin jika 200 000 < L < 2 000 000;
- 0 poin jika L ≥ 2 000 000.

Batasan: ukuran program ≤ 200.

Subtask 5 [up to 28 points]

Mungkin ada berapapun kerikil pada setiap sel pada grid (tentu saja, antara 0 dan 15). Tulis sebuah program yang mencari minimum, yaitu yang berakhir dengan odometer berada pada sel (i, j) sedemikian sehingga setiap sel lain memiliki setidaknya kerikil sebanyak yang terdapat pada (i, j). Setelah menjalankan program, banyaknya kerikil pada setiap sel harus sama seperti sebelum menjalankan program.

Nilai dari subtask ini tergantung dari ukuran program P pada program yang dikumpulkan. Lebih tepatnya, nilai Anda adalah:

- 28 poin jika P ≤ 444;

- 0 poin jika $P \geq 4\,440$.

Batasan: lama eksekusi ≤ 44 400 000.

Implementation details

Anda harus mengumpulkan tepat sebuah file per subtask, ditulis sesuai spesifikasi di atas. Setiap file yang dikumpulkan berukuran maksimal 5 MiB. Untuk setiap subtask, kode odometer Anda akan dites dengan beberapa test case, dan Anda akan menerima umpan balik mengenai resource yang digunakan oleh code Anda. Jika kode tersebut mengandung kesalahan sintaks sehingga tidak bisa dites, Anda akan menerima informasi mengenai syntax error tersebut.

Program yang Anda kumpulkan tidak harus mengandung program odometer untuk semua subtask. Jika submission Anda pada suatu waktu tidak mengandung program odometer untuk subtask X, pengumpulan Anda yang paling akhir untuk subtask X akan dimasukkan secara otomatis; jika tidak ada program yang demikian, subtask tersebut akan dinilai nol.

Seperti biasanya, nilai dari sebuah pengumpulan adalah jumlah dari nilai-nilai yang diperoleh pada setiap subtask, dan nilai akhir pada soal adalah nilai maksimum di antara pengumpulan yang sudah mengalami release-tested dan pengumpulan terakhir.

Simulator

Untuk keperluan pengujian, disediakan sebuah simulator odometer untuk Anda, yang mana Anda dapat mengumpaninya dengan program Anda dan grid masukan. Program odometer akan ditulis dengan format yang sama dengan yang untuk pengumpulan (yakni, yang telah dijelaskan di atas).

Deskripsi grid akan diberikan menggunakan format sebagai berikut: setiap baris pada file harus berisi tiga bilangan, R, C, dan P, yang berarti sel pada baris R dan kolom C memiliki P kerikil. Semua sel yang tidak dispesifikasikan pada deskripsi grid, diasumsikan tidak mengandung kerikil. Sebagai contoh, perhatikan file berikut:

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

Grid yang dideskripsikan file ini mengandung 15 kerikil: 3 pada sel (0, 10) dan 12 pada sel (4, 5).

Anda dapat meminta simulator untuk melakukan tes dengan memanggil program `simulator.py` pada direktori soal Anda, dan mem-passing nama file program sebagai argumen. Program simulator akan menerima command line options sebagai berikut:

- `-h` akan memberikan penjelasan singkat mengenai options yang tersedia;
- `-g GRID_FILE` memuat deskripsi grid dari file `GRID_FILE` (default: grid kosong);
- `-s GRID_SIDE` mengeset ukuran dari grid menjadi `GRID_SIDE x GRID_SIDE` (default: 256, seperti yang digunakan pada spesifikasi soal); penggunaan grid yang lebih kecil dapat berguna untuk mendebug program;

- `-m STEPS` membatasi banyaknya langkah eksekusi pada simulasi menjadi paling banyak `STEPS`;

- `-c` memasukkan modus kompilasi; pada modus kompilasi, simulator ini mengembalikan keluaran yang persis sama, tetapi selain melakukan simulasi dengan Python, ia men-generate dan melakukan kompilasi sebuah program kecil dalam bahasa C. Ini menyebabkan overhead yang lebih besar pada saat dimulai, tetapi kemudian akan memberikan hasil yang jauh lebih cepat; Anda disarankan menggunakan ini ketika program Anda diharapkan berjalan untuk lebih dari 10 000 000 langkah.

Number of submissions

Banyak maksimum pengumpulan yang diperbolehkan untuk soal ini adalah 128.