Odômetro de Pedrinhas

Leonardo inventou o *odômetro* original: uma carroça que podia medir distâncias deixando cair pedrinhas à medida que as rodas giravam. A contagem das pedras fornecia o número de voltas da roda, o que permitia ao usuário computar a distância percorrida pelo odômetro. Como cientistas da computação, adicionamos controle por software ao odômetro, extendendo suas funcionalidades. Sua tarefa é programar o odômetro de acordo com as regras especificadas abaixo.

Operação do quadriculado

O odômetro move-se em um quadriculado imaginário de 256 x 256 células. Cada célula contém no máximo 15 pedrinhas e é identificada por um par de coordenadas (linha, coluna), em que cada coordenada está no intervalo 0, ..., 255. Dada uma célula (i, j), as células adjacentes são (se existem) (i - 1, j), (i + 1, j), (i, j - 1) e (i, j + 1). Qualquer célula da primeira ou da última linha, ou da primeira ou da última coluna, é chamada de *borda*. O odômetro sempre inicia na célula (0, 0) (o canto noroeste), apontando para o norte.

Comandos Básicos

O odômetro pode ser programado usando os seguintes comandos.

- `left` — gire 90 graus para a esquerda (sentido anti-horário) e permaneça na célula corrente (por exemplo, se estava apontando para o sul antes do comando, então estará apontando para o leste após o comando).
- `right` — gire 90 graus para a direita (sentido horário) e permaneça na célula corrente (por exemplo, se estava apontando para o oeste antes do comando, então estará apontando para o norte após o comando).
- `move` — mova uma unidade para a frente (na direção em que o odômetro está apontando) para uma célula adjacente. Se essa célula não existe (i. e. a borda nessa direção já foi alcançada), então o comando não tem efeito.
- `get` — remova uma pedrinha da célula corrente. Se a célula corrente não contém pedrinhas, então o comando não tem efeito.
- `put` — adicione uma pedrinha à célula corrente. Se a célula corrente já contém 15 pedrinhas, então o comando não tem efeito. O odômetro nunca fica sem pedrinhas.
- `halt` — termina a execução.

O odômetro executa os comandos na ordem dada no programa. O programa deve ter no máximo um comando por linha. Linhas em branco são ignoradas. O símbolo `#` indica um comentário; qualquer texto que segue esse símbolo, até o final da linha, é ignorado. Se o odômetro chega ao final do programa, a execução termina.

Exemplo 1

Considere o seguinte programa para o odômetro. Ele coloca o odômetro na célula (0, 2), apontando para o leste. (Note que o primeiro `move` é ignorado, porque o odômetro está no canto noroeste, apontando para o norte.)

```
move  # sem efeito
right
# agora o odômetro está apontando para o leste
move
move
```

Rótulos, bordas e pedrinhas

Para alterar o fluxo do programa dependendo do estado corrente, você pode utilizar rótulos, que são cadeias de caracteres (sensíveis a maiúsculas/minúsculas), com no máximo 128 símbolos entre `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Os comandos que utilizam rótulos são listados abaixo. Na descrição, *`L`* denota um rótulo válido.

- *`L`*`:` (i.e. *`L`* seguido de dois pontos '`:`') — declara a posição dentro do programa com rótulo *`L`*. Todos os rótulos declarados devem ser únicos. A declaração de um rótulo não tem efeito no odômetro.
- `jump` *`L`* — continua a execução saltando incondicionalmente para a linha de rótulo *`L`*.
- `border` *`L`* — continua a execução saltando para a linha de rótulo *`L`*, se o odômetro está em uma borda apontando para uma fora do grid (i.e. `move` não teria efeito); caso contrário, a execução continua normalmente e este comando não tem efeito.
- `pebble` *`L`* — continua a execução saltando para a linha de rótulo *`L`* se a célula corrente contém ao menos uma pedrinha; caso contrário a execução continua normalmente e este comando não tem efeito.

Exemplo 2

O programa a seguir localiza a primeira (mais a oeste) pedrinha na linha 0 e para; se não há pedrinhas na linha 0, ele para na borda ao final da linha. O programa usa dois rótulos `leonardo` e `davinci`.

```
right
leonardo:
pebble davinci  # pedrinha encontrada
border davinci  # final da linha
move
jump leonardo
davinci:
halt
```

O odômetro inicia girando para a sua direita. O loop inicia com uma declaração do rótulo `leonardo:` e termina com o comando `jump leonardo`. No loop, o odômetro verifica a presença de uma pedrinha ou a chegada à borda no final da linha; caso contrário o odômetro executa um `move` da célula corrente (0, j) para a célula adjacente (0, j + 1) já que esta existe. (O comando `halt` não é estritamente necessário aqui, já que o programa termina de qualquer forma.)

Enunciado

Você deve submeter um programa na linguagem do odômetro, como descrito acima, para fazer o odômetro comportar-se como esperado. Cada sub-tarefa (veja abaixo) especifica o comportamento que o odômetro deve ter e as restrições que a solução submetida deve satisfazer. As restrições se referem aos seguintes pontos.

- *Tamanho do programa* — o programa deve ser curto o suficiente. O tamanho de um programa é o seu número de comandos. Declarações de rótulos, comentários e linhas em brancos *não são contados* no tamanho.
- *Comprimento da execução* — o programa deve terminar suficientemente rápido. O comprimento da execução é o número de *passos* executados: cada execução de um comando conta como um passo, independentemente se o comando teve efeito ou não; declarações de rótulos, comentários e linhas em branco não contam como passos.

No Exemplo 1, o tamanho do programa é 4 e o comprimento da execução é 4. No Exemplo 2, o tamanho do programa é 6 e, quando executado em um quadriculado com uma única pedrinha na célula (0, 10), o comprimento da execução é 43 passos: `right`, 10 iterações do loop, cada iteração tomando 4 passos (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), e finalmente, `pebble davinci` e `halt`.

Subtarefa 1 [9 pontos]

No início há x pedrinhas na célula (0, 0) e y na célula (0, 1); todas as outras células estão vazias. Escreva um programa que termina com o odômetro na célula (0, 0) se $x \leq y$, e na célula (0, 1) caso contrário. (Não nos importamos quanto à direção para a qual o odômetro aponta ao final; também não nos importamos com quantas pedrinhas estão presentes no quadriculado ao final, nem onde estão localizadas.)

Limites: tamanho do programa ≤ 100, comprimento da execução ≤ 1 000.

Subtarefa 2 [12 pontos]

Mesma tarefa acima, mas quando o programa termina a célula (0, 0) deve conter exatamente x pedrinhas e a célula (0, 1) deve conter exatamente y pedrinhas.

Limites: tamanho do programa ≤ 200, comprimento da execução ≤ 2 000.

Subtarefa 3 [19 pontos]

Há exatamente duas pedrinhas em algum lugar da linha 0: uma está na célula (0, x), a outra na célula (0, y); x e y são distintos, e x + y é par. Escreva um programa que deixe o odômetro na célula (0, (x + y) / 2), i.e., exatamente na célula no meio entre as duas células contendo as pedrinhas. O estado final do quadriculado não é relevante.

Limites: tamanho do programa ≤ 100, comprimento da execução ≤ 200 000.

Sub-tarefa 4 [até 32 pontos]

Há no máximo 15 pedrinhas no quadriculado, nenhuma célula contém mais do que uma pedrinha. Escreva um programa que coleta todas as pedrinhas no canto noroeste do quadriculado. Mais precisamente, se há x pedrinhas no quadriculado no início, ao final deve haver exatamente x pedrinhas na célula (0, 0) e nenhuma pedrinha em outra célula.

A pontuação para esta sub-tarefa depende do comprimento da execução do programa submetido. Mais precisamente, se L é o comprimento máximo de execução dos vários casos de teste, sua pontuação será:

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

Limites: tamanho do programa ≤ 200.

Sub-tarefa 5 [até 28 pontos]

Pode haver qualquer número de pedrinhas em cada célula do quadriculado (entre 0 e 15, claro). Escreva um protrama que encontre o mínimo, i.e., que termine com o odômetro em uma célula (i, j) tal que cada outra célula contém ao menos tantas pedras quanto (i, j). Após executar o programa, o número de pedrinhas em cada célula dever ser o mesmo que antes do início da execução.

A pontuação para esta sub-tarefa depende do tamanho de programa P do programa submetido. Mais precisamente, sua pontuação será:

- 28 pontos se P ≤ 444;
- 28 - 28 $\log_{10}$ (P / 444) pontos se 444 < P < 4 440;
- 0 pontos se P ≥ 4 440.

Limites: comprimento da execução ≤ 44 400 000.

Detalhes da Implementação

Você deve submeter exatamente um arquivo por sub-tarefa, escrito de acordo com as regras de sintaxe especificadas acima. Cada arquivo submetido deve ter um tamanho máximo de 5 MiB. Para cada sub-tarefa, seu código de odômetro será testado com alguns casos de teste, e você receberá feedback sobre os recursos utilizados pelo seu código. No caso de o código não estar sintaticamente correto e portanto impossível de ser testado, você receberá informação específica sobre o erro de sintaxe.

Não é necessário que suas submissões contenham programas de odômetro para todas as sub-tarefas. Se sua submissão corrente não contém o programa de odômetro para a sub-tarefa X, sua submissão mais recente para a sub-tarefa X é automaticamente incluída; se não há esse programa, a sub-tarefa receberá pontuação zero para essa submissão.

Como é usual, a pontuação de uma submissão é a soma das pontuações obtidas em cada sub-tarefa, e a pontuação final da tarefa é a pontuação máxima entre todas as sumbissões liberadas e a última submissão.

Simulador

Para propósitos de teste, você terá à disposição um simulador de odômetro, o qual você pode alimentar com seus programas e quadriculados de entrada. Os programas odômetros serão escritos no mesmo formato usado na submissão (i.e, conforme descrição acima).

A descrição do quadriculado será dada usando o seguinte formato: cada linha do arquivo contém três números, R, C e P, representando a célula na linha R e coluna C contém P pedrinhas. Todas as células não especificadas na descrição do quadriculado não possuem pedrinhas. Por exemplo, considere o arquivo:

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

O quadriculado descrito por este arquivo deve conter 15 pedrinhas: 3 na célula (0, 10) e 12 na célula (4, 5).

Você pode invocar o simulador de testes por chamadas do programa `simulator.py` no diretório da tarefa, passando o nome do programa como argumento. O programa simulador aceitará as seguintes opções na linha de comando:

- `-h` dará um breve apresentação das opções disponíveis;
- `-g GRID_FILE` carrega a descrição do quadriculado do arquivo `GRID_FILE` (padrão: quadriculado vazio);
- `-s GRID_SIDE` atribui o tamanho do quadriculado para `GRID_SIDE x GRID_SIDE` (padrão: 256, como usado na especificação do problema); uso de quadriculados menores pode ser útil para depurar o programa.
- `-m STEPS` limita o número de passos da execução em no máximo `STEPS`;

- `-c` entra em modo compilação; em modo compilação, o simulador devolve exatamente a mesma saída, mas ao invés de fazer a simulação com Python, este gera e compila um pequeno programa em C. Isto causa uma alta sobrecarga inicial, mas dá resultados significativamente mais rápidos; sugerimos que use isto quando for esperado que seu programa execute por mais de 10 000 000 passos.

Número de submissões

O número máximo de submissões permitidas para esta tarefa é 128.