Odomètre à cailloux (Pebbling odometer)

Léonard a inventé le premier "odomètre" : un chariot qui pouvait mesurer des distances en déposant des cailloux au fur et à mesure que ses roues tournaient. En comptant le nombre de cailloux, on pouvait en déduire le nombre de tours de roue, ce qui permettait à l'utilisateur de mesurer la distance parcourue. En tant qu'informaticiens, nous avons ajouté une interface logicielle à l'odomètre, ce qui permet d'améliorer ses capacités. Votre tâche est de programmer l'odomètre selon les règles définies ci-dessous.

Grille des opérations

L'odomètre se déplace sur une grille carrée imaginaire de 256 × 256 cases. Chaque case peut contenir au plus 15 cailloux et est repérée par une paire de coordonnées (ligne, colonne) où chaque coordonnée est dans l'intervalle 0, ..., 255. Étant donné une case (i, j), les cases adjacentes sont (si elles existent) (i - 1, j), (i + 1, j), (i, j - 1), (i, j + 1). Les cases de la première et dernière ligne, et de la première et dernières colonne forment la *bordure*. L'odomètre commence toujours sur la case (0, 0) (le coin nord-ouest) et est dirigé vers le nord.

Commandes de base

L'odomètre peut être programmé en utilisant les commandes suivantes.

- `left` — tourne de 90° vers la gauche (dans le sens contraire des aiguilles d'une montre) et reste sur la même case (par exemple s'il était orienté vers le sud avant la commande, il sera orienté vers l'est après).
- `right` — tourne de 90° vers la droite (dans le sens des aiguilles d'une montre) et reste sur la même case (par exemple s'il était orienté vers l'ouest avant la commande, il sera orienté vers le nord après).
- `move` — déplace l'odomètre d'une case (dans la direction dans laquelle il est orienté) vers la case voisine. Si une telle case n'existe pas (i.e. parce que l'odomètre a atteint le bord dans cette direction) cette commande n'a aucun effet.
- `get` — enlève un caillou de la case en cours. Si la case en cours ne contient aucun caillou, cette commande n'a aucun effet.
- `put` — ajoute un caillou à la case en cours. Si la case contient déjà 15 cailloux, cette commande n'a aucun effet. L'odomètre n'est jamais à cours de cailloux.
- `halt` — termine l'exécution du programme.

L'odomètre exécute les instructions dans l'ordre du programme. Le programme doit contenir au plus une instruction par ligne. Les lignes vides sont ignorées. Le symbole `#` indique un commentaire; tout le texte qui suit sur la ligne est ignoré. Si l'odomètre atteint la fin du programme, il s'arrête.

Exemple 1

Considérez le programme suivant de l'odomètre. Il déplace l'odomètre à la case (0, 2) et l'oriente vers l'est. (Notez que le premier `move` est ignoré car l'odomètre est au bord nord-ouest et est orienté vers le nord).

```
move  # aucun effet
right
# maintenant l'odomètre est orienté vers l'est
move
move
```

Labels, bordures et cailloux

Pour modifier le flux d'exécution du programme, vous pouvez utiliser des labels, qui sont des chaînes de caractères sensibles à la casse constituées d'au plus 128 symboles choisis parmi `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Les nouvelles commandes utilisant des labels sont listées ci-dessous. Dans la descriptions, *`L`* dénote n'importe quel label valide.

- *`L`*`:` (i.e. *`L`* suivi de ':') — déclare la position dans le programme du label *`L`*. Tous les labels déclarés doivent être uniques. Déclarer un label n'a aucun effet sur l'odomètre.
- `jump` *`L`* — continue l'exécution du programme en sautant inconditionnellement à la ligne du label *`L`*.
- `border` *`L`* — continue l'exécution en sautant à la ligne du label *`L`* si l'odomètre se trouve sur le bord et est orienté vers le bord de la grille (c'est-à-dire, une instruction `move` n'aurait aucun effet) ; sinon l'exécution continue normalement et cette commande n'a aucun effet.
- `pebble` *`L`* — continue l'exécution en sautant à la ligne du label *`L`* si l'odomètre se trouve sur une case qui contient au moins un caillou ; sinon l'exécution continue normalement et cette commande n'a aucun effet.

Exemple 2

Le programme suivant trouve la position du premier caillou sur la ligne 0 (le plus à l'ouest) et s'arrête là ; si la ligne 0 ne contient aucun caillou, il s'arrête à la fin de la ligne. Ce programme utilise deux labels `leonardo` et `davinci`.

```
right
leonardo:
pebble davinci  # caillou trouvé
border davinci  # fin de la ligne
move
jump leonardo
davinci:
halt
```

L'odomètre commence par tourner à droite. La boucle commence par la déclaration du label `leonardo:` et se termine par l'instruction `jump leonardo`. Dans la boucle, l'odomètre teste la présence d'un caillou ou du bord à la fin de la ligne ; dans le cas contraire, l'odomètre passe de la case (0, j) à la case (0, j + 1) par un `move` puisque cette case existe. La commande `halt` n'est pas strictement nécessaire à cet endroit car le programme se termine à la fin de toute façon.

Problème

Vous devez soumettre un programme dans le langage de l'odomètre comme décrit précédemment, qui se comporte comme demandé. Chaque sous-tâche (voir ci-dessous) spécifie le comportement attendu de l'odomètre et les contraintes à respecter afin de la valider. Les contraintes concernent les deux mesures suivantes.

- *Taille du programme* — le programme doit être suffisamment court. La taille d'un programme est le nombre de commandes qu'il contient. Les déclarations de labels, les commentaires et les lignes vides *ne sont pas comptés* dans la taille.
- *Durée d'exécution* — le programme doit se terminer suffisamment vite. La durée d'exécution est le nombre de *pas* exécutés: chaque commande compte pour un pas, indépendamment de si la commande à un effet ou non ; Les déclarations de labels, les commentaires et les lignes vides ne comptent pas pour un pas.

Dans l'exemple 1, la taille du programme est de 4 et sa durée d'exécution est de 4. Dans l'exemple 2, la taille du programme est de 6 et, lorsqu'il est exécuté sur une grille avec un sur caillou sur la case (0, 10), sa durée d'exécution est de 43 pas: `right`, 10 itérations de la boucle, chaque itération comptant 4 pas (`pebble`

`davinci`; `border davinci`; `move`; `jump leonardo`), et enfin, `pebble davinci` et `halt`.

Sous-tâche 1 [9 points]

Il y a au début x cailloux sur la case (0, 0) et y sur la case (0, 1) tandis que toutes les autres cases sont vides. Rappelez-vous qu'il ne peut pas y avoir plus de 15 cailloux dans chaque case. Écrivez un programme qui termine avec l'odomètre sur la case (0, 0) si x ≤ y, et qui termine sur la case (0, 1) sinon. L'orientation de l'odomètre à la fin n'est pas importante ; de même, le nombre de cailloux présents à la fin sur la grille ainsi que leur localisation ne sont pas importants.

Limites: taille du programme ≤ 100, durée d'exécution ≤ 1 000.

Sous-tâche 2 [12 points]

Même tâche que précédemment mais lorsque le programme se termine, la case (0, 0) doit contenir exactement x cailloux et la case (0, 1) exactement y cailloux.

Limites: taille du programme ≤ 200, durée d'exécution ≤ 2 000.

Sous-tâche 3 [19 points]

Il y a exactement deux cailloux sur la ligne 0 : un sur la case (0, x) et l'autre sur la case (0, y) ; x et y étant distincts et x + y étant pair. Écrivez un programme qui déplace l'odomètre sur la case (0, (x + y) / 2), i.e. exactement à mi-distance entre les deux cases contenant les cailloux. L'état final de la grille n'a aucune importance.

Limites: taille du programme ≤ 100, durée d'exécution ≤ 200 000.

Sous-tâche 4 [jusqu'à 32 points]

Il y a au plus 15 cailloux sur la grille et deux cailloux ne se trouvent jamais sur la même case. Écrivez un programme qui rassemble tous les cailloux dans le coin nord-ouest ; plus précisément, s'il y a x cailloux sur le grille au départ, à la fin il doit y avoir exactement x cailloux sur la case (0, 0) et aucun caillou ailleurs.

Le score pour cette tâche dépend de la durée d'exécution du programme soumis. Plus précisément, si L est la durée maximale d'exécution sur les fichiers tests, votre score sera de:

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

Limites : taille du programme ≤ 200.

Sous-tâche 5 [jusqu'à 28 points]

Il peut y avoir un nombre quelconque de cailloux sur chaque case (compris entre 0 et 15 inclus bien sûr). Écrivez un programme qui trouve le minimum, i.e. qui termine avec l'odomètre sur une case (i, j) telle que toutes les autres cases contiennent au moins autant de cailloux que la case (i, j). Après avoir lancé le programme, le nombre de cailloux sur chaque case doit être le même qu'au départ.

Le score pour cette sous-tâche dépend de la taille P du programme soumis. Plus précisément, votre score sera de :

- 28 points si P ≤ 444;
- 28 - 28 $\log_{10}$ (P / 444) points si 444 < P < 4 440;

- 0 points si P ≥ 4 440.

Limites : durée d'exécution ≤ 44 400 000.

Détails d'implémentation

Vous devez soumettre exactement un fichier par sous-tâche, écrit en respectant les règles de syntaxe énoncées plus haut. Chaque fichier soumis peut avoir une taille d'au plus 5 Mio. Pour chaque sous-tâche, votre code sera testé sur quelques fichiers tests et vous recevrez des informations sur les ressources utilisées par votre code. Dans le cas où votre code ne serait pas syntaxiquement correct et donc impossible à tester, vous recevrez des informations sur l'erreur de syntaxe.

Il n'est pas nécessaire que vos soumissions contiennent les programmes pour toutes les sous-tâches. Si votre soumission ne contient pas le programme pour la sous-tâche X, votre soumission la plus récente pour la sous-tâche X sera automatiquement inclue ; si aucun programme n'a été soumis pour cette sous-tâche, votre score sera de 0 pour cette soumission.

Comme d'habitude, le score d'une soumission sera la somme des scores pour chaque sous-tâche, et le score final de la tâche sera le maximum des scores parmi les soumissions utilisant un release-token et la dernière soumission.

Simulateur

À des fins de test, on vous fournit un simulateur d'odomètre auquel vous pouvez donner un programme et une grille d'entrée. Les programmes sont écrits dans le même format que pour la soumission (i.e. dans le format décrit ci-dessus).

La description des grilles doit être donnée en utilisant le format suivant : chaque ligne contient exactement trois entiers R, C et P signifiant que la case à la ligne R et la colonne C contient P cailloux. Les cases non-spécifiées dans la description sont considérées comme vides. Par exemple, considérez le fichier suivant:

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

La grille décrite par ce fichier contient 15 cailloux : 3 sur la case (0, 10) et 12 sur la case (4, 5).

Vous pouvez lancer le simulateur en lançant le programme `simulator.py` dans le dossier de la tâche et en donnant au programme le nom du fichier en argument. Le simulateur accepte aussi les options suivantes sur la ligne de commande :

- `-h` affiche une brève description des options disponibles ;
- `-g GRID_FILE` charge une description de la grille depuis le fichier `GRID_FILE` (par défaut la grille est vide) ;
- `-s GRID_SIDE` définit la taille de la grille à `GRID_SIDE x GRID_SIDE` (par défaut 256 est utilisé, comme dans les spécifications du problème) ; l'utilisation de grilles plus petites est utile uniquement à des fins de déboggage ;
- `-m STEPS` limite le nombre de pas de simulation à `STEPS`;
- `-c` utilise le mode compilation ; en mode compilation le simulateur affiche la même sortie mais génère et compile un petit programme C au lieu de faire la simulation en Python. Cela induit un temps d'attente au lancement mais la simulation est nettement plus rapide ; nous vous conseillons d'utiliser ce mode si la simulation nécessite plus de 10 000 000 pas.

Nombre de soumissions

Le nombre maximal de soumissions autorisées pour cette tâche est de 128.