Pebbling odometer

לאונרדו המציא אודומטר, אשר ננסה כאן לחקות את פעולותיו.

Operation grid

האודומטר נע על לוח משבצות בגודל 256 על 256 שהקואורדינטות של כל משבצת בו הן בין 0 ל- 255. בכל משבצת יש בין 0 ל- 15 גולות. בהתחלה האודומטר מוצב בתא (0,0), שהוא הפינה הצפון מערבית (צפון זה למעלה. מערב זה שמאלה). בהתחלה, פניו של האודומטר צפונה.

Basic commands

עליכם לתכנת את האודומטר בעזרת הפקודות הבאות.

- `left`

פקודה זו מסובבת את פניו של האודומטר 90 מעלות שמאלה (כלומר נגד כיוון השעון). לדוגמא: אם לפני קבלת הפקודה האודומטר פנה דרומה, לאחר ביצוע הפקודה הוא יפנה מזרחה.

- `right`

פקודה זו מסובבת את פניו של האודומטר 90 מעלות ימינה (כלומר עם כיוון השעון).

- `move`

פקודה זו מזיזה את האודומטר תא אחד בכיוון אליו הוא פונה באותו רגע. אם אין תא סמוך בכיוון זה (כי האודומטר נמצא על קצה הלוח) אז הפקודה הזו לא עושה שום דבר.

- `get`

פקודה זו מסירה גולה אחת מהמשבצת עליה נמצא האודומטר, כלומר, מקטינה את מספר הגולות במשבצת הזו ב - 1. אם במשבצת עליה נמצא האודומטר אין גולות אז הפקודה הזו לא עושה שום דבר.

- `put`

פקודה זו מוסיפה גולה אחת למשבצת עליה נמצא האודומטר. כלומר, מגדילה את מספר הגולות במשבצת הזו ב - 1. אם במשבצת הזו כבר יש 15 גולות אז הפקודה הזו לא עושה שום דבר.

- `halt`

פקודה זו מסיימת את ריצת התוכנית.

האודומטר מבצע את ההוראות שכתובות בתוכניות שניתנות לו בסדר שבו הן נתונות, שורה אחר שורה. בכל שורה יש

הוראה אחת לכל היותר. האודומטר מתעלם משורות ריקות ומשורות שמתחילות ב- # (שמסמן הערה). האודומטר מפסיק את פעולתו אם הוא מגיע לסוף התוכנית או שהוא מקבל פקודה מפורשת לסיום.

Example 1

Consider the following program for the odometer. It takes the odometer to the cell (0, 2), facing east. (Note that the first `move` is ignored, because the odometer is on the north-west corner facing north.)

```
move  # no effect
right
# now the odometer is facing east
move
move
```

Labels, borders and pebbles

בנוסף לפקודות המתוארות, יש פקודות נוספות שמקפיצות את מהלך הביצוע לשורה שמסומנת על-ידי תוית (Label).

- *L*`:`

מסמן שורה בתוך התוכנית בתוית L. אורך התוית הוא 128 תוים לכל היותר שכל אחד מהם הוא a..z או A..Z או 0..9.

- `jump` *L*

קפוץ לשורה בתוכנית שמסומנת בתוית L.

- `border` *L*

אם האודומטר נמצא על משבצת בקצה הלוח ופונה החוצה (כלומר, בדיוק במצב שבו פקודת move לא תעשה שום דבר) אז הפקודה הזו קופצת לשורה בתוכנית שמסומנת בתוית L. אחרת, הפקודה הזו לא עושה שום דבר.

- `pebble` *L*

אם האודומטר נמצא במשבצת שיש בה לפחות גולה אחת אז הפקודה הזו קופצת לשורה שמסומנת בתוית L. אחרת, הפקודה הזו לא עושה שום דבר.

Example 2

The following program locates the first (westmost) pebble in row 0 and stops there; if there are no pebbles in row 0, it stops on the border at the end of the row. It uses two labels `leonardo` and `davinci`.

```
right
leonardo:
pebble davinci  # pebble found
border davinci  # end of the row
move
jump leonardo
davinci:
halt
```

The odometer starts by turning to its right. The loop begins with the label declaration `leonardo:` and ends with the `jump leonardo` command. In the loop, the odometer checks for the presence of a pebble or the border at the end of the row; if not so, the odometer makes a `move` from the current cell (0, j) to the adjacent cell (0, j + 1) since the latter exists. (The `halt` command is not strictly necessary here as the program terminates anyway.)

Statement

לפניך סדרה של subtasks, חלקם קלים. בכל subtask מתוארת משימה שונה שהאודומטר צריך לבצע בעזרת תוכנית שתכתוב עבורו בשפה שתוארה. בכל אחת מהמשימות נתונות שתי מגבלות:

- *מגבלה על גודל התוכנית* - התוכנית חייבת להיות קצרה מספיק (תויות, הערות ושורות ריקות לא נספרות).
- *מגבלה על מספר הפעולות שיבוצעו* - מספר הפעולות שיבוצעו במהלך הריצה חייב להיות קטן (כולל הוראות שלא עשו שום דבר - פעולות כגון move, get, jump נספרות גם אם בפועל לא בוצע בהן שום דבר).

בדוגמא 1 גודל התוכנית הוא 4 ומספר הפעולות 4. בדוגמא 2 גודל התוכנית הוא 6, וכאשר היא מופעלת על לוח משבצות שבו גולה אחת בדיוק בתא (0,10), מספר הפעולות הוא 43 (בגרסה האנגלית יש פירוט של הפעולות שמבוצעות במקרה זה).

[Subtask 1 [9 points

בהתחלה יש x גולות בתא (0,0) ו - y גולות בתא (0,1). כל שאר התאים ריקים (זכור שיש לכל היותר 15 גולות בתא). כתוב תוכנית שבסיומה - אם $x \le y$ אז האודומטר יהיה בתא (0,0) ואחרת יהיה בתא (0,1). לא חשוב לאן פונה האודומטר בסיום התוכנית והיכן נמצאות הגולות.

Limits: program size ≤ 100, execution length ≤ 1 000.

[Subtask 2 [12 points

Same task as above but when the program ends, the cell (0, 0) must contain exactly x pebbles and cell (0, 1) must contain exactly y pebbles.

Limits: program size ≤ 200, execution length ≤ 2 000.

[Subtask 3 [19 points

There are exactly two pebbles somewhere in row 0: one is in cell (0, x), the other in cell (0, y); x and y are distinct, and x + y is even. Write a program that leaves the odometer in cell (0, (x + y) / 2), i.e., exactly in the midpoint between the two cells containing the pebbles. The final state of the grid is not relevant.

Limits: program size ≤ 100, execution length ≤ 200 000.

[Subtask 4 [up to 32 points

There are at most 15 pebbles in the grid, no two of them in the same cell. Write a program that collects them all in the north-west corner; more precisely, if there were x pebbles in the grid at the beginning, at the end there must be exactly x pebbles in cell (0, 0) and no pebbles elsewhere.

The score for this subtask depends on the execution length of the submitted program. More precisely, if L is the maximum of the execution lengths on the various test cases, your score will be:

- 32 points if $L \leq 200\,000$;
- $32 - 32 \cdot \log_{10}(L / 200\,000)$ points if $200\,000 < L < 2\,000\,000$;
- 0 points if $L \geq 2\,000\,000$.

Limits: program size ≤ 200.

[Subtask 5 [up to 28 points

צריך להביא את האודומטר לתא שבו יש הכי מעט גולות. בנוסף, בסוף הריצה, בכל תא צריך להיות אותו מספר גולות שהיה בו בהתחלה. There may be any number of pebbles in each cell of the grid (of course, between 0 and 15). Write a program that finds the minimum, i.e., that terminates with the odometer in a cell (i, j) such that every other cell contains at least as many pebbles as (i, j). After running the program, the number of pebbles in each cell must be the same as before running the program.

The score for this subtask depends on the program size P of the submitted program. More precisely, your score will be:

- 28 points if $P \leq 444$;
- $28 - 28 \cdot \log_{10}(P / 444)$ points if $444 < P < 4\,440$;
- 0 points if $P \geq 4\,440$.

Limits: execution length ≤ 44 400 000.

פרטי מימוש

(כדאי לשקול לקרוא את החלקים הלא מתורגמים בגרסה האנגלית כי שם הם מיושרים לשמאל) בחלק הזה יש הוראות על ההגשה של שאלת OUTPUT זו ועל הניקוד שלה. שימו לב - בהגשה לא חייבים לכלול את כל ה - subtasks, אבל כל subtask שלא כוללים בהגשה מסוימת יקבל את הניקוד של ההגשה האחרונה שבה הוא הוגש ולאו דווקא של ההגשה הכי טובה שלו. הניקוד יינתן כרגיל על המקסימום מבין ההגשות שעשו להן release וההגשה האחרונה. בפועל זה אומר שכדאי לכלול בכל הגשה את הגרסה הכי טובה של הפתרון לכל subtask שיש לכם עד עכשיו.

You have to submit exactly one file per subtask, written according to the syntax rules specified above. Each submitted file can have a maximum size of 5 MiB. For each subtask, your odometer code will be tested on a few test cases, and you will receive some feedback on the resources used by your code. In the case the code is not syntactically correct and thus impossible to test, you will

.receive information on the specific syntax error

It is not necessary that your submissions contain odometer programs for all the subtasks. If your current submission does not contain the odometer program for subtask X, your most recent submission for subtask X is automatically included; if there is no such program, the subtask will .score zero for that submission

As usual, the score of a submission is the sum of the scores obtained in each subtask, and the final score of the task is the maximum score among the release-tested submissions and the last .submission

Simulator כאן יש פירוט על אופן השימוש בסימולטור שמגיע עם השאלה ויכול לעזור עם בדיקת התוכניות שלכם.

For testing purposes, you are provided with an odometer simulator, which you can feed with your programs and input grids. Odometer programs will be written in the same format used for .(submission (i.e., the one described above

Grid descriptions will be given using the following format: each line of the file must contain three numbers, R, C and P, meaning that the cell at row R and column C contains P pebbles. All cells not :specified in the grid description are assumed to contain no pebbles. For example, consider the file

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

The grid described by this file would contain 15 pebbles: 3 in the cell (0, 10) and 12 in the cell (4, .(5

You can invoke the test simulator by calling the program `simulator.py` in your task directory, passing the program file name as argument. The simulator program will accept the following :command line options

- `-h` will give a brief overview of the available options;
- `-g GRID_FILE` loads the grid description from file `GRID_FILE` (default: empty grid);
- `-s GRID_SIDE` sets the size of the grid to `GRID_SIDE x GRID_SIDE` (default: 256, as used in the problem specification); usage of smaller grids can be useful for program ;debugging
- `-m STEPS` limits the number of execution steps in the simulation to at most `STEPS`;
- `-c` enters compilation mode; in compilation mode, the simulator returns exactly the same output, but instead of doing the simulation with Python, it generates and compiles a small C program. This causes a larger overhead when starting, but then gives significantly faster results; you are advised to use it when your program is expected to run for more than about .10 000 000 steps

Number of submissions

.The maximum number of submissions allowed for this task is 128