Pebbling odometer

Leonardo a inventat *odometrul*: o căruță care putea măsura distanțele aruncând pietricele după cum se roteau roțile acesteia. Numărând pietricelele aflam numărul de rotiri al roții, ceea ce permitea utilizatorului să calculeze distanță parcursa de odometru. Ca programatori, noi am adăugat un soft de control odometrului, extinzându-i funcționalitățile. Sarcina ta este să programezi odometrul respectând regulile specificate mai jos.

Zona de funcționare

Odometrul se mișcă pe o matrice pătratică imaginară având 256 x 256 celule. Fiecare celulă poate conține cel mult 15 pietricele și se identifică printr-o pereche de coordonate (rând, coloană), unde fiecare coordonată se află în intervalul 0, ..., 255. Pentru o celuă (i, j), celulele adiacente ei sunt (dacă există) (i - 1, j), (i + 1, j), (i, j - 1) și (i, j + 1). Oricare celulă aflată pe primul sau ultimul rând, sau pe prima sau ultima coloană, este numită *frontieră*. Odometrul începe întotdeauna din celula (0, 0) (colțul nord-vest), orientat spre nord.

Comenzile de bază

Odometrul poate fi programat folosind următoarele comenzi.

- `left` — se rotește cu 90 de grade spre stânga (invers acelor de ceasornic) și rămâne în celula curentă (de exemplu: dacă înainte a fost orientat spre sud, după executarea comenzii va fi orientat spre est).
- `right` — se rotește cu 90 de grade spre dreapta (în sensul acelor de ceasornic) și rămâne în celula curentă (de exemplu: dacă înaintă a fost orientat spre vest, după executarea comenzii va fi orientat spre nord).
- `move` — se mută exact o unitate înainte (în direcția în care este orientat odometrul) în celula adiacentă. Dacă o astfel de celulă nu există (adică frontiera în această direcție a fost deja atinsă) atunci comanda nu are niciun efect.
- `get` — elimină o pietricică din celula curentă. Dacă celula curentă nu are nicio pietricică, atunci comanda nu are niciun efect.
- `put` — adaugă o pietricică în celula curentă. Dacă celula curentă are deja 15 pietricele, atunci comanda nu are niciun efect. Odometrul nu rămâne niciodată fără pietricele.
- `halt` — încheie execuția programului.

Odometrul execută comenzile în ordinea în care sunt date în program. Programul trebuie să conțină cel mult o comandă pe linie. Liniile goale vor fi ignorate. Simbolul `#` indică un comentariu; orice text care urmează până la sfârșitul liniei, este ignorat. Dacă odometrul ajunge la sfârșitul programului, execuția se încheie.

Examplul 1

Fie următorul program pentru odometru. El duce odometrul în celula (0, 2), orientat spre est. (remarcați că primul `move` este ignorat, deoarece odometrul este în colțul nord-vest orientat spre nord).

```
move  # fără efect
right
# acum odometrul este orientat spre est
move
move
```

Etichete, frontiere și pietricele

Pentru a modifica funcționarea programului în funcție de starea curentă, poți utiliza etichete, care sunt stringuri case-sensitive având cel mult 128 de simboluri din `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Noile comenzi referitoare etichetelor sunt listate mai jos. În descrierea de mai jos, *L* reprezintă orice etichetă validă.

- *L*`:` (adică *L* urmat de două puncte '`:`') — declară locația în interiorul programului a etichetei *L*. Toate declarațiile etichetelor trebuie să fie unice. Declararea unei etichete nu are niciun efect asupra odometrului.
- `jump` *L* — continuă execuția prin salt necondiționat la eticheta *L*.
- `border` *L* — continuă execuția sărind la linia cu eticheta *L*, doar dacă odometrul se află pe o celulă frontieră orientat spre exteriorul matricei (adică o instrucțiune `move` nu ar avea efect); altfel, execuția continuă normal iar această comandă nu are niciun efect.
- `pebble` *L* — continuă execuția sărind la linia cu eticheta *L*, doar dacă odometrul se află pe o celulă care conține măcar o pietricică; altfel, execuția continuă normal iar această comandă nu are niciun efect.

Exemplul 2

Următorul program localizează prima (cea mai din vest) pietricică din rândul 0 și se oprește acolo; dacă nu există nicio pietricică în rândul 0, programul se oprește pe frontieră la sfârșitul rândului. El folosește două etichete `leonardo` și `davinci`.

```
right
leonardo:
pebble davinci  # pietricică găsită
border davinci  # sfârșitul rândului
move
jump leonardo
davinci:
halt
```

Odometrul pornește rotindu-se spre dreapta. Bucla începe cu declararea etichetei `leonardo` și se termină cu comanda `jump leonardo`. În buclă, odometrul verifică dacă se află pietricele sau dacă se află pe frontieră la sfârșitul rândului; dacă nu, odometrul execută comanda `move` din celula curentă (0, j) în celula adiacenă (0, j+1) cât timp cea din urmă există. (Comanda `halt` nu este strict necesara aici deoarece programul se încheie oricum).

Enunț

Tu trebuie să trimiți un program în limbajul odometrului, cum a fost descris mai sus, care să facă odometrul să se comporte conform așteptărilor. Fiecare subtask (vezi mai jos) specifică o comportare pe care odometrul trebuie să o îndeplinească și restricțiile pe care soluția trimisă trebuie să le îndeplinească. Restricțiile se refer urătoarele două aspecte.

- *Dimensiunea programului* - programul trebuie să fie suficient de scurt. Dimensiunea programului este numărul de comenzi pe care îl conține. Declarațiile de etichete, comentariile și liniile libere *nu se contorizează" în dimensiune.*
- *Lungimea execuției* - programul trebuie să se termine suficient de repede. Lungimea execuției este numărul de *pași*: fiecare execuție a unei comenzi este numărată ca un pas, chiar dacă comanda are efect sau nu; declararea etichetelor, comentariile și liniile libere nu se numără ca pași.

În primul exemplu, programul are dimensiunea 4 și lungimea execuției 4. În al doilea exemplu, programul are dimensiunea 6 și, când este executat pe o matrice având o singură pietricică în celula (0, 10), lungimea execuției este de 43 de pași: `right`, 10 iterații prin buclă, fiecare conținând câte 4 pași (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), și la final, `pebble davinci` și `halt`.

Subtask 1 [9 puncte]

La început se află x pietricele în celula (0, 0) și y pietricele în celula (0, 1), în timp ce toate celalte celule sunt libere. Rețineți că pot fi cel mult 15 pietricele într-o celulă. Scrie un program care se încheie cu odometrul în celula (0, 0) dacă $x \leq y$, sau în celula (0, 1) în caz contrar (Nu contează cum este orientat odometrul la sfârșit sau câte pietricele există, în final, în matrice și nici unde sunt așezate).

Limite: Dimensiunea programului ≤ 100, lungimea execuției ≤ 1 000.

Subtask 2 [12 puncte]

La fel ca subtask-ul anterior, dar când programul se încheie, celula (0, 0) trebuie să conțină exact x pietricele iar celula (0, 1) trebuie să conțină exact y pietricele.

Limite: Dimensiunea programului ≤ 200, lungimea execuției ≤ 2 000.

Subtask 3 [19 puncte]

Există exact două pietricele pe rândul 0: una în celula (0, x), iar cealalta în celula (0, y); x și y sunt distincte, iar x + y este par. Scrie un program care lasă odometrul în celula (0, (x + y) / 2), adică exact la mijlocul dintre cele două celule care conțin pietricele. Starea finală a matricii este irelevantă.

Limite: Dimensiunea programului ≤ 100, lungimea execuției ≤ 200 000.

Subtask 4 [pana la 32 de puncte]

Se află cel mult 15 pietricele în matrice, oricare două în celule diferite. Scrie un program care le adună pe toate în colțul nord-vest; mai exact, dacă la început existau x pietricele în matrice, atunci la sfârșit trebuie să existe exact x pietricele în celula (0, 0) și nici o altă pietricică în celelalte celule.

Scorul acestui subtask depinde de lungimea execuției a programului trimis. Mai exact, dacă L este lungimea de execuție maximă a unui test, scorul tău va fi:

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

Limite: dimensiunea programului ≤ 200.

Subtask 5 [până la 28 de puncte]

Pot exista oricâte pietricele în fiecare celulă a matrice (desigur, între 0 și 15). Scrie un program care găsește minimul, adică, care se termină cu odometrul în celula (i, j) astfel încât oricare altă celulă conține cel puțin la fel de multe pietricele ca și celula (i, j). După rularea programului, numărul pietricelelor din fiecare celulă trebuie să fie același ca înainte de rularea programului.

Scorul acestui subtask depinde de dimensiunea programului P a programului trimis. Mai exact, scorul tău va fi:

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

Limite: lungimea execuției ≤ 44 400 000.

Detalii de implementare

Trebuie să trimiți exact un fișier pentru fiecare subtask, respectând regulile de sintaxă specificate mai sus. Fiecare fișier trimis poate avea cel mult 5 MB. Pentru fiecare subtask, codul odometrului va fi testat pe câteva teste și vei primi feedback în legătură cu resursele utilizate de codul tău. În

cazul în care codul nu respectă sintaxa și astfel este imposibil de testat, tu vei primi informații despre sintaxa de eroare.

Nu este necesar ca submisiile să conțină programe pentru odometru la toate subtask-urile. Dacă submisia curentă nu conține programul odometrului pentru subtask-ul x, cea mai recentă submisie pentru task-ul x va fi automat inclusă; dacă nu există deja trimis un astfel de program, acest subtask va primi 0 puncte pentru acea submisie.

Ca de obicei, punctajul unui submit se calculează însumand scorurile subtask-urilor, iar scorul final al sarcinii este punctajul maxim din rândul submit-urilor precedente şi al ultimului.

Simulator

În scopuri de testare, vi se oferă un simulator de odometru, căruia ii poți furniza programele tale și matricele de intrare. Programele odometrului vor fi scrise în formatul folosit pentru submit-uri (de exemplu, unul din cele descrise mai sus).

Descrierea matricei va fi dată cu ajutorul următorului format: fiecare linie a fișierului trebuie să conțină trei numere R, C și P, semnificand faptul că celula din randul R și coloana C conține P pietricele. Se presupune că celulele ce nu sunt specificate în descrierea matriceise nu conțin pietricele. Pentru exemplificare se consideră următorul fișier:

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

Matricea descrisă de acest fișier trebuie să conțină 15 pietricele: 3 în celulă (0, 10) și 12 în celulă (4, 5).

Poți invoca simulatorul de testare prin apelarea programului `simulator.py` în directorul sarcinei tale, transmițandui ca argument numele fişierului de program ca argument. Programul de simulare va accepta în linia de comandă următoarele opțiuni:

- `-h` va oferi o scurtă trecere în revistă a opțiunilor disponibile;
- `-g GRID_FILE` încarcă descrierea matricei din fişierul `GRID_FILE` (implicit: matricrea vidă);
- `-s GRID_SIDE` setează dimensiunea matricei ca fiind egală cu `GRID_SIDE x GRID_SIDE` (implicit 256, aşa cum este specificat în problemă); utilizarea unei matrice mai mici ar putea fi utilă în cazul depănarii programului;
- `-m STEPS` limitează numărul de paşi executați în procesul de simulare la cel mult `STEPS`;
- `-c` intră în modul de compilare; în modul de compilare, simulatorul returnează exact aceiaşi ieşire, dar în loc de a face o simulare cu Python, el generează și compilează un mic program în limbajul C. Acest lucru duce la cheltuieli mai mari la start, dar dă rezultate mult mai rapid; se recomandă să-l foşoseşti atunci când se preconizează ca programul tau va rula mai mult de circa 10 000 000 de paşi.

Numărul de submit-uri

Numărul maxim de submit-uri permise pentru această sarcină este de 128.