Odometer s kamenčki

Leonardo je izumil originalni *odometer*: voziček, ki je ob vsakem obratu koles odvrgel kamenček in s tem omogočil merjenje razdalj. Preštevanje odvrženih kamenčkov je vrlega uporabnika pripeljalo do števila obratov vozičkovih koles in s tem do prevožene razdalje. Kot računalničarji se seveda nismo mogli upreti skušnjavi in smo zategadelj odometru obogatili uporabniško izkušnjo. Tvoja naloga je omenjeno napravo sprogramirati v skladu s spodaj navedenimi pravili.

Delovno območje

Odometer se premika po namišljeni kvadratni mreži, sestavljeni iz 256 × 256 enotskih polj, pri čemer je na vsakem polju lahko kvečjemu 15 kamenčkov. Polja so označena s parom koordinat (vrstica, stolpec), vsaka je v razponu 0, …,255. Danemu polju (i, j) so polja (i - 1, j), (i + 1, j), (i, j - 1) in (i, j + 1) (če obstajajo) sosednja. Polju, ki leži na prvi ali zadnji vrstici oziroma na prvem ali zadnjem stolpcu, pravimo *rob*. Odometer vedno začne na polju (0, 0) (t.j. v severno-zahodnem vogalu), in je obrnjen proti severu.

Osnovni ukazi

Voziček je moč upravljati z naslednjimi ukazi.

- `left` — obrni se za 90 stopinj v levo (v nasprotni smeri urinega kazalca) in ostani na istem polju (npr. če je bil prej obrnjen proti jugu, bo po tem obrnjen proti vzhodu).
- `right` — obrni se za 90 stopinj v desno (v smeri urinega kazalca) in ostani na istem polju (npr. če je bil prej obrnjen proti zahodu, bo po tem obrnjen proti severu).
- `move` — premakni se za eno enoto naprej v sosednje polje (v smeri, v katero je odometer trenutno obrnjen). Če tako polje ne obstaja (npr. če je voziček v tej smeri že na robu), potem ukaz nima nobenega učinka.
- `get` — odstrani en kamenček s trenutnega polja. Če je le-to prazno, potem ukaz nima nobenega učinka.
- `put` — odloži en kamenček na trenutno polje. Če je na polju že bilo 15 kamenčkov, potem ukaz nima nobenega učinka. Odometer ima neskončno zalogo kamenčkov.
- `halt` — prekini izvajanje.

Pretanjeni merilnik izvaja ukaze natanko v vrstnem redu, kot so napisani v programu, vsak v svoji vrstici — v vsaki je kvečjemu en ukaz. Prazne vrstice se ignorirajo. Znak `#` označuje začetek komentarja; karkoli mu sledi do konca vrstice, se ignorira. Če odometer doseže konec programa, se izvajanje prekine.

1. primer

Posvetimo kanček pozornosti naslednjemu primeru programa. Opevani voziček je popeljan do polja (0, 2) tako, da je na koncu obrnjen proti vzhodu. (Upoštevaj, da se prvi `move` ignorira, saj je odometer takrat obrnjen proti severu, na severno-zahodnem voglu.)

```
move  # ni učinka
right
# sedaj je odometer obrnjen proti vzhodu
move
move
```

Oznake, robovi in kamenčki

Za spreminjanje poteka programa glede na trenutno stanje lahko uporabljate oznake — nize največ 128. črk angleške abecede in cifer, razlikujoč med malimi in velikimi črkami (torej `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`). Dodatni ukazi za delo z oznakami so našteti spodaj, kjer *`L`* označuje veljavne oznake.

- *`L`*`:` (t.j. *`L`*, ki mu sledi '`:`') — za oznako *`L`* deklarira njeno mesto v programu. Vse deklarirane oznake morajo biti edinstvene. Deklaracija nima nobenega vpliva na naš dragi voziček.
- `jump` *`L`* — izvajanje brezpogojno nadaljuj s skokom na vrstico z oznako *`L`*.
- `border` *`L`* — z izvajanjem nadaljuj v vrstici z oznako *`L`*, če je odometer na robnem polju in obrnjen proti robu območja (t.j. ukaz `move` ne bi imel učinka). Sicer ukaz nima učinka in se izvajanje normalno nadaljuje naprej.
- `pebble` *`L`* — z izvajanjem nadaljuj v vrstici z oznako *`L`*, če je na trenutnem polju vsaj en kamenček. Sicer ukaz nima učinka in se izvajanje normalno nadaljuje naprej.

2. primer

Sledeči program najde prvi (najbolj zahoden) kamenček v 0. vrstici ter se tam ustavi; če v tej vrstici ni kamenčkov, se ustavi na robu, na koncu vrstice. Uporablja oznaki `leonardo` in `davinci`.

```
right
leonardo:
pebble davinci  # najden kamenček
border davinci  # konec vrstice
move
jump leonardo
davinci:
halt
```

Odometer začne z obratom na desno. Zanka se prične z deklaracijo oznake `leonardo:` in konča z ukazom `jump leonardo`. V zanki voziček najprej preveri ali lahko pobere kamenček in ali se

nahaja na robu. Če oba testa klavrno propadeta, se z `move` premakne s trenutne celice (0, j) naprej na sosednjo (0, j + 1) (ta seveda obstaja, saj preverjanje roba ni bilo uspešno). (Ukaz `halt` tu ni neobhodno potreben, saj je programa tako ali tako konec.)

Naloga

Oddati moraš program (napisan v zgoraj opisanem jeziku) ki povzroči zahtevano obnašanje odometra. Vsaka podnaloga (glej spodaj) opisuje neko obnašanje, ki mu mora voziček slediti, in kakopak tudi omejitve, ki jim mora oddani program zadoščati. Omejitve so omejene na sledeče lastnosti.

- *Velikost programa* — program mora biti dovolj kratek. Velikost programa je število ukazov; deklaracije oznak, komentarji in prazne vrstice *ne štejejo* k velikosti.
- *Trajanje izvajanja* — program se mora končati dovolj hitro. Trajanje izvajanja je število izvedenih *korakov*: vsak ukaz šteje kot en korak, ne glede na to, ali je imel učinek ali ne; deklaracije oznak, komentarji in prazne vrstice ne štejejo kot koraki.

V 1. primeru je velikost programa 4, trajanje pa takisto 4. V 2. primeru je velikost programa 6, trajanje pa je malce bolj zamotan oreh. Če je na delovnem območju samo en kamenček, na polju (0, 10), je trajanje 43 korakov: `right`, 10 ponovitev zanke, kjer je vsaka ponovitev dolga 4 korake (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), in na koncu `pebble davinci` in `halt`.

1. podnaloga [9 točk]

Sprva je na polju (0, 0) x kamenčkov, na polju (0, 1) jih je y, vsa ostala polja pa so prazna. Pomni, da je na vsakem polju lahko največ 15 kamenčkov. Napiši program, ki se konča z odometrom na polju (0, 0), če je $x \leq y$, sicer pa na polju (0, 1). (Ni važno, v katero smer je voziček obrnjen na koncu; po koncu izvajanja takisto ni važno, koliko kamenčkov še ostane na delovnem območju in kje se nahajajo.)

Omejitve: velikost programa ≤ 100, trajanje izvajanja ≤ 1 000.

2. podnaloga [12 točk]

Enaka naloga kot zgoraj, vendar mora ob koncu programa polje (0, 0) vsebovati natanko x kamenčkov, polje (0, 1) pa natanko y kamenčkov.

Omejitve: velikost programa ≤ 200, trajanje izvajanja ≤ 2 000.

3. podnaloga [19 točk]

Nekje v vrstici 0 sta natanko dva kamenčka: en na polju (0, x), drugi pa na polju (0, y); x in y sta različna, $x + y$ je sodo. Napiši program, ki odometer pusti na polju (0, ($x + y$) / 2), t.j. točno na sredini med kamenčkoma. Končno stanje na delovnem območju ni važno.

Omejitve: velikost programa ≤ 100, trajanje izvajanja ≤ 200 000.

4. podnaloga [do 32 točk]

Na delovnem območju je največ 15 kamenčkov, v vsakem polju največ en. Napiši program, ki vse kamenčke zbere v severo-zahodni vogal; natančneje, če je bilo na začetku na delovnem območju x kamenčkov, mora na koncu biti na polju (0, 0) natanko x kamenčkov, drugje pa nobenega.

Točkovanje pri tej nalogi je odvisno od trajanja izvajanja programa. Nadrobno, če je L največje trajanje izvajanja na posameznih testnih primerih, ti bodo dodeljene točke po tem sistemu:

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

Omejitve: velikost programa ≤ 200.

5. podnaloga [do 28 točk]

Na vsakem polju je lahko poljubno število kamenčkov (seveda med 0 in 15). Napiši program, ki najde minimum, t.j. se zaključi z odometrom na polju (i, j), za katerega velja, da ima vsako drugo polje vsaj toliko kamenčkov kot (i, j). Po koncu programa mora biti število kamenčkov na vsakem polju enako kot je bilo na začetku.

Točkovanje pri tej podnalogi je odvisno od velikosti oddanega programa P. Podrobneje, točke boš dobil po tem sistemu:

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

Omejitve: trajanje izvajanja ≤ 44 400 000.

Podrobnosti implementacije

Oddati moraš natanko eno datoteko za vsako podnalogo, spisano po zgoraj predpisanih pravilih. Vsaka oddana datoteka ima lahko največ 5 MiB. Za vsako podnalogo bo tvoj umotvor preverjen na nekaj testnih primerih, za katere boš dobil tudi nekaj povratnih informacij o porabi sredstev. Če nanese, da za kak testni primer oddaja ni sintaktično pravilna in jo je torej nemogoče testirati, boš dobil natančnejše podatke o napaki.

Ni potrebno vzeti za nujno, da tvoje oddaje vsebujejo odometrske programe za vse podnaloge. Če tvoja trenutna oddaja ne vsebuje programov za podnalogo X, je najnovejša oddaja za nalogo X avtomatično vključena; če taka oddaja ne obstaja, boš za podnalogo v trenutni oddaji prejel častnih 0 točk.

Kot je v navadi že stoletja, boš za svoje trdo kamenjavo delo ob vsaki oddaji nagrajen z vsoto točk po posameznih podnalogah, končna ocena zadane naloge pa bo največja ocena izmed zadnje oddaje in oddaj, testiranih na vseh testnih primerih.

Simulator

V svrho testiranja ti je dan na razpolago simulator slavljenega odometra, ki ga lahko nahraniš s svojimi programi in vhodnimi delovnimi območji. Programi naj bodo napisani po podobi programov za oddaje (t.j. kot opisano zgoraj).

Opisi delovnih območij naj bodo v takem formatu: vsaka vrstica mora vsebovati tri števila, R, C in P, kar pomeni, da je v vrstici R in stolpcu C P kamenčkov. Vsa polja, ki se jih na ta način ne dotakneš, bodo privzeto prazna. Za primer poglej to datoteko:

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

Območje, opisano v tej datoteki, ima 15 kamenčkov: 3 na polju (0, 10) in 12 na polju (4, 5).

Simulator lahko zaženeš s klicem programa `simulator.py` v mapi z nalogo, pri čemer za prvi argument napišeš ime datoteke z opisom. Simulator bo sprejel ta stikala:

- `-h` da kratek pregled vseh možnih stikal;
- `-g GRID_FILE` naloži opis območja iz datoteke `GRID_FILE` (privzeto: prazno območje);
- `-s GRID_SIDE` nastavi velikost območja na `GRID_SIDE x GRID_SIDE` (privzeto: 256, kot je uporabljeno v opisu naloge); uporaba manjših območij lahko pomaga pri razhroščevanju;
- `-m STEPS` omeji število korakov pri izvajanju v simulaciji na največ `STEPS`;
- `-c` vstopi v prevajalniški način; v tem načinu simulator vrne natanko enak rezultat, vendar namesto simulacije s pythonom ustvari in prevede majhen C program. To povzroči malce daljši zagon ampak vodi tudi v opazno hitrejše rezultate; priporočeno je, da ta način uporabljaš, če pričakuješ, da bo program izvedel več kot približno 10 000 000 korakov.

Število oddaj

Za to nalogo lahko oddaš največ 128 oddaj.