Kivikestega odomeeter

Leonardo leiutas esialgse *odomeetri*: käru, millega sai mõõta läbitud vahemaad, kuna see pillas maha ühe kivikese iga kord, kui käru ratas tegi ühe täistiiru. Kivikeste loendamine andis käru ratta täistiirude arvu, mis võimaldas kasutajal arvutada odomeetri poolt läbitud vahemaa pikkuse. Arvutiteadlastena oleme me lisanud odomeetrile programmjuhtimise, laiendades nii selle funktsionaalsust. Teie ülesanne on programmeerida odomeetrit allpool kirjeldatud reeglite järgi.

Väljak

Odomeeter liigub kujutletaval ruudukujulisel väljakul mõõtmetega 256 × 256 ühikruutu. Igas ruudus võib olla ülimalt 15 kivikest ja iga ruut on identifitseeritav tema koordinaatide (rida, veerg) järgi, kus iga koordinaat on täisarv 0, …, 255. Kui vaadata ruutu (i, j), siis ruudud selle kõrval (kui need eksisteerivad) on (i-1, j), (i+1, j), (i, j-1) ja (i, j+1). Igat ruutu, mis asetseb esimeses või viimases reas või veerus, nimetatakse *piiriks*. Odomeeter alustab alati ruudult (0, 0) (loodenurgast), olles suunaga põhja.

Peamised käsud

Odomeetrit saab programmeerida, kasutades järgmisi käske.

- `left` — pööra 90 kraadi vasakule (vastupäeva) ja jää samasse ruutu (näiteks kui odomeeter oli suunaga lõunasse, siis pärast käsku on see suunaga itta).
- `right` — pööra 90 kraadi paremale (päripäeva) ja jää samasse ruutu (näiteks kui odomeeter oli suunaga läände, siis pärast käsku on see suunaga põhja).
- `move` — liigu ühe sammu võrra edasi (odomeetri paiknemise suunas) kõrvalasuvasse ruutu. Kui sellist ruutu ei ole (s.t. selles suunas liikumisel on juba jõutud piirile), siis ei tehta midagi.
- `get` — korja käesolevast ruudust üles üks kivikene. Kui käesolevas ruudus pole ühtegi kivikest, siis ei tehta midagi.
- `put` — pane üks kivikene käesolevasse ruutu. Kui käesolevas ruudus on juba 15 kivikest, siis ei tehta midagi. Odomeetrist ei saa kivikesed kunagi otsa.
- `halt` — lõpeta programmi täitmine.

Odomeeter täidab käske programmi poolt etteantud järjekorras. Programmi ühel real võib olla ülimalt üks käsk. Tühje ridu ignoreeritakse. Sümbol # tähistab kommentaari; odomeeter ignoreerib igasugust sellele järgnevat teksti kuni rea lõpuni. Kui odomeeter jõuab programmi lõpuni, siis töö lõpetatakse.

Näide 1

Vaatleme järgmist odomeetri jaoks koostatud programmi. See viib odomeetri ruudule (0, 2), suunaga itta. (Pane tähele, et esimest `move`-käsku ignoreeritakse, kuna odomeeter seisab loodenurgas suunaga põhja.)

```
move  # midagi ei juhtu
right
# odomeeter on nüüd suunaga itta
move
move
```

Märgendid, piirid ja kivikesed

Selleks, et muuta programmi täitmise järjekorda sõltuvalt hetkeolekust, saad sa kasutada märgendeid, mis on ülimalt 128 kohalised tõstutundlikud stringid ja koosnevad sümbolitest `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Uued käsud, mis kasutavad märgendeid, on loetletud allpool. Kirjelduses tähistab *L* igasugust reeglitele vastavat märgendit.

- *L*`:` (s.t. *L*, millele järgneb koolon '`:`') — tähistab programmis kohta märgendiga *L*. Kõik deklareeritud märgendid peavad olema unikaalsed. Märgendi deklareerimine ei oma odomeetrile mingisugust mõju.
- `jump` *L* — jätka programmi täitmist, hüpates tingimusteta reale, kus asub märgend *L*.
- `border` *L* — jätka programmi täitmist realt märgendiga *L*, kui odomeeter asub piiril suunaga väljakult väljapoole (s.t. käsk `move` ei teeks midagi); vastasel juhul see käsk ei tee midagi ja programmi täitmine jätkub normaalselt.
- `pebble` *L* — jätka programmi täitmist realt märgendiga *L*, kui käesolevas ruudus on vähemalt üks kivike; vastasel juhul see käsk ei tee midagi ja programmi täitmine jätkub normaalselt.

Näide 2

Järgmine programm teeb kindlaks esimese (vasakpoolseima) kivikese reas 0 ja peatub seal; kui real 0 ei ole mitte ühtki kivikest, siis peatub rea lõpus piiril. Programm kasutab kaht märgendit: `leonardo` ja `davinci`.

```
right
leonardo:
pebble davinci  # kivike leitud
border davinci  # rea lõpp
move
jump leonardo
davinci:
halt
```

Odomeeter alustab pöördega paremale. Tsükkel algab märgendi `leonardo:` deklareerimisega ja lõpeb käsuga `jump leonardo`. Tsüklis kontrollib odomeeter kivikese olemasolu või piirile jõudmist rea lõpus; kui need tingimused pole täidetud, siis odomeeter täidab käsu `move` ja liigub ruudult(0, j) kõrvalasuvale ruudule (0, j+1), kuna see kindlasti eksisteerib. (Käsk `halt` on

tegelikult mittevajalik, kuna programm peatuks selles kohas nii-kui-nii.)

Ülesanne

Sa pead esitama odomeetri enda keeles koostatud programmi, nagu eespool kirjeldatud ning see peab panema odomeetri käituma nii, nagu oodatakse. Iga alamülesanne (vaata allpool) kirjeldab käitumist, mida odomeeter peab täitma ja piiranguid, mida esitatud lahendus peab rahuldama. Piiranguid esitatakse kahe järgneva omaduse kohta.

- *Programmi suurus* — programm peab olema piisavalt lühike. Programmi suuruse mõõdikuks on käskude arv programmis. Märgendite deklaratsioonid, kommentaarid ja tühjad read *ei lähe* programmi suuruse leidmisel arvesse.
- *Täitmise kestvus* — programm peab lõpetama oma töö piisavalt kiiresti. Täitmise kestvuse mõõdikuks on teostatud *sammude* arv: iga üksik käsu täitmine läheb arvesse kui samm, hoolimata sellest, kas see käsk midagi tegi või mitte; märgendite deklaratsioonid, kommentaarid ja tühjad read sammudena arvesse ei lähe.

Näites 1 on programmi suuruseks 4 ja täitmise kestvuseks ka 4. Näites 2 on programmi suuruseks 6 ja kui programm käivitatakse väljakul, millel on üksainus kivike ruudus (0, 10), siis täitmise kestvuseks on 43 sammu: `right`, 10 tsükli iteratsiooni, igas iteratsioonis 4 sammu (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), ja lõpuks `pebble davinci` ja `halt`.

Alamülesanne 1 [9 punkti]

Alguses on x kivikest ruudus (0, 0) ja y kivikest ruudus (0, 1), kusjuures kõik ülejäänud ruudud on tühjad. Meenuta, et igas ruudus võib olla ülimalt 15 kivikest. Kirjuta programm, mis peataks odomeetri ruudus (0, 0), kui $x \leq y$, ja ruudus (0, 1) vastupidisel juhul. (Me ei hooli, millises suunas odomeeter lõpuks on; samuti ei huvita meid, kui mitu kivikest on väljakul ja kus need asuvad.)

Piirangud: programmi suurus ≤ 100, täitmise kestvus ≤ 1 000.

Alamülesanne 2 [12 punkti]

Sama ülesanne nagu eelmine, kuid kui programm lõpetab, siis ruudul (0, 0) peab olema täpselt x kivikest ja ruudus (0, 1) peab olema täpselt y kivikest.

Piirangud: programmi suurus ≤ 200, täitmise kestvus ≤ 2 000.

Alamülesanne 3 [19 punkti]

Real 0 on kuskil täpselt kaks kivikest: üks on ruudus (0, x) ja teine ruudus (0, y); x ja y on erinevad ning x + y on paarisarv. Kirjuta programm, mis jätab odomeetri ruudule (0, (x + y) / 2), s.t. täpselt kivikesi sisaldavate ruutude vahele. Väljaku lõplik seis ei ole oluline.

Piirangud: programmi suurus ≤ 100, täitmise kestvus ≤ 200 000.

Alamülesanne 4 [kuni 32 punkti]

Väljakul on ülimalt 15 kivikest, ükski neist ei ole mõne teisega samas ruudus. Kirjuta programm, mis kogub need kõik loodenurka; täpsemalt öeldes, kui alguses on väljakul x kivikest, siis lõpuks peab olema täpselt x kivikest ruudus (0, 0) ja mitte ühtegi kivikest kusagil mujal.

Selle alamülesande skoor sõltub esitatud programmi täitmise kestvusest. Täpsemalt väljendudes, kui L on maksimaalne täitmise kestvus mitmesuguste testjuhtude korral, siis sinu skoor on:

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

Piirangud: programmi suurus ≤ 200.

Alamülesanne 5 [kuni 28 punkti]

Väljakul võib olla suvaline number kivikesi igas ruudus (muidugi 0 ja 15 vahel). Kirjuta programm, mis leiaks miinimumi, s.t. peataks programmi töö nii, et odomeeter asuks ruudus (i, j) ja iga muu ruut sisaldaks vähemalt sama palju kivikesi kui ruut (i, j). Pärast programmi töö lõppu peab kivikeste arv igas ruudus olema täpselt sama kui programmi töö alguses.

Selle alamülesande skoor sõltub esitatava programmi suurusest P. Täpsemalt väljendudes, sinu skoor on:

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

Piirang: täitmise kestvus ≤ 44 400 000.

Realisatsioon

Sa pead esitama iga alamülesande kohta täpselt ühe faili, mis on koostatud eelkirjeldatud süntaksireegleid silmas pidades. Iga esitatud faili suuruseks võib olla ülimalt 5 MiB. Iga alamülesande korral testitakse sinu esitatud odomeetri koodi mõne testjuhuga ja sa saad mingisugust tagasisidet sinu koodi poolt kasutatavate ressursside kohta. Juhul, kui kood ei ole süntaktiliselt korrektne ja seetõttu mittetestitav, siis saad sa vastava vea kohta infot.

Esitatavad lahendused ei pea sisaldama odomeetri programme kõigi alamülesannete jaoks. Kui lahendus ei sisalda programmi mõne alamülesande X jaoks, siis lisatakse komplektile automaatselt viimane seni esitatud alamülesande X programm; kui sellist programmi ei ole, siis teenib lahendus selle alamülesande eest 0 punkti.

Nagu ikka, on lahenduse skooriks iga alamülesande eest saadud punktide summa ja ülesande lõplikuks skooriks on maksimaalne skoor testitud esituste skooride ja viimati üleslaaditud lahenduse skoori hulgast.

Simulaator

Testimise eesmärgil on sul kasutada odomeetri simulaator, millele saad sa ette anda oma programmi ja väljaku kirjelduse. Odomeetri programmid peavad olema kirjutatud samas formaadis, mida kasutad lahenduse esitamiseks (s.t. sama, millest eespool oli juttu).

Väljaku kirjeldus tuleb ette anda kasutades järgmist formaati: iga rida failis peab koosnema kolmest arvust, R, C ja P, mis tähendab, et ruut reas R ja veerus C sisaldab P kivikest. Kõik need ruudud, mille kohta väljaku kirjelduses infot ei ole, ei sisalda kivikesi. Näiteks vaata faili:

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

Selles failis kirjeldatud väljak sisaldab 15 kivikest: 3 ruudus (0, 10) ja 12 ruudus (4, 5).

Sa saad käivitada simulaatori, pannes käima programmi `simulator.py` oma ülesandekataloogist ja andes oma programmi nime argumendiks. Simulaatori programm võtab arvesse järgmiseid käsurea parameetreid:

- `-h` annab lühikese ülevaate võimalikest parameetritest;
- `-g GRID_FILE` loeb sisse väljaku kirjelduse failist `GRID_FILE` (vaikimisi: tühi väljak);
- `-s GRID_SIDE` seab väljaku suuruseks `GRID_SIDE x GRID_SIDE` (vaikimisi: 256, nagu on ka kirjas ülesandepüstituses); väiksema väljaku kasutamine on kasulik programmi silumiseks;
- `-m STEPS` piirab simulatsioonis tehtavate sammude arvu väärtusega `STEPS`;
- `-c` viib simulaatori kompileerimise režiimi; kompileerimise režiimis tagastab simulaator täpselt samasuguse töötulemuse, kuid Pythonis simuleerimise asemel genereerib ja kompileerib see väikese C-programmi. See põhjustab suurema ajakao käivitamise alguses, kuid seejärel leiab tulemused märgatavalt kiiremini; on soovitatav kasutada seda režiimi siis, kui sinu programmi oodatav kestvus ületab 10 000 000 sammu piiri.

Esituste (üleslaadimiste) arv

Selle ülesande korral on esituste maksimaalseks arvuks 128.