International Olympiad in Informatics 2012
23-30 September 2012
Sirmione - Montichiari, Italy
Competition tasks, day 1: Leonardo's inventions and projects

odometer
Latvian — 1.2

Akmentiņu odometrs

Leonardo izgudroja oriģinālu *odometru*: ratiņus, ar kuriem varēja mērīt attālumu, izmetot akmentiņus ratiņu riteņu griešanās laikā. Izskaitot akmentiņus, varēja iegūt riteņu apgriezienu skaitu, kas ļāva lietotājam izskaitļot odometra veikto attālumu. Būdami datorzinātnieki, mēs odometram esam pievienojuši vadību ar programmatūru, tādējādi paplašinot tā funkcionalitāti. Jūsu uzdevums ir uzprogrammēt odometru ievērojot tālāk minētos nosacījumus.

Darbības laukums

Odometrs pārvietojas pa iedomātu 256x256 rūtiņu režģi. Katra šūna (rūtiņa) satur ne vairāk kā 15 akmentiņus un tiek identificēta ar koordināšu pāri (rinda, kolonna), kur katra koordināta atrodas intervālā 0, …, 255. Dotai šūnai (i, j) tās kaimiņu šūnas (ja tādas eksistē) ir (i - 1, j), (i + 1, j), (i, j - 1) un (i, j + 1). Jebkura šūna, kas atrodas pirmajā vai pēdējā rindā, vai pirmajā vai pēdējā kolonnā, tiek saukta par *robežas* šūnu. Odometrs vienmēr sākumā atrodas šūnā (0, 0) (ziemeļ-rietumu stūrī) un ir vērsts ziemeļu virzienā.

Pamatkomandas

Odometru var programmēt izmantojot sekojošās komandas.

- `left` — pagriezties par 90 grādiem pa kreisi (pretēji pulksteņrādītāja kustības virzienam), paliekot tajā pašā šūnā (piemēram, ja tas bija vērsts dienvidu virzienā, tad, pēc komandas izpildes, tas būs vērsts austrumu virzienā).
- `right` — pagriezties par 90 grādiem pa labi (pulksteņrādītāja kustības virzienā), paliekot tajā pašā šūnā (piemēram, ja tas bija vērsts rietumu virzienā, tad, pēc komandas izpildes, tas būs vērsts ziemeļu virzienā).
- `move` — pārvietoties vienu vienību uz priekšu (virzienā, kurā ir vērsts odometrs) uz blakus esošo šūnu. Ja šāda šūna neeksistē (t.i., robeža attiecīgajā virzienā jau ir sasniegta) tad šai komandai nav nekāda efekta.
- `get` — noņemt vienu akmentiņu no šūnas, kurā atrodas odometrs. Ja attiecīgajā šūnā nav neviena akmentiņa, šai komandai nav nekāda efekta.
- `put` — novieto vienu akmentiņu šūnā, kurā atrodas odometrs. Ja attiecīgajā šūna jau ir 15 akmentiņi, tad šai komandai nav nekāda efekta. Odometram nekad neizbeidzas akmentiņi, kurus var novietot šūnās.
- `halt` — pārtraukt programmas izpildi.

Odometrs izpilda komandas tādā secībā, kādā tās ir dotas programmā. Katrai programmas rindai jāsatur ne vairāk par vienu komandu. Tukšas rindas tiek ignorētas. Simbols # indicē komentāru; jebkāds teksts, kas seko pēc šī simbola, līdz pat rindas beigām, tiek ignorēts. Ja odometrs sasniedz programmas beigas, izpilde tiek izbeigta.

1.piemērs

Aplūkosim sekojošo odometra programmu. Tā liek odometram pārvietoties uz šūnu (0, 2), vērstam austrumu virzienā. (Ievērot, ka pirmā `move` komanda tiek ignorēta, jo odometrs atrodas ziemeļ-rietumu stūrī un ir vērsts uz ziemeļiem.)

```
move  # nav efekta
right
# tagad odometrs ir vērsts uz austrumiem
move
move
```

Iezīmes, robežas un akmentiņi

Lai mainītu programmas plūsmu atkarībā no pašreizējā stāvokļa, Jūs varat izmantot iezīmes, kas ir reģistrjūtīgas simbolu virknes ne garākas par 128 simboliem `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Ar iezīmēm saistītās komandas ir aprakstītas zemāk. Aprakstos *L* apzīmē jebkuru derīgu iezīmi.

- *L*`:` (t.i., *L* ar sekojošu kolu '`:`') — deklarē iezīmes *L* atrašanās vietu programmā. Visām deklarētajām iezīmēm jābūt unikālām. Iezīmes deklarācija nekādi neietekmē odometra darbību.
- `jump` *L* — turpināt programmas izpildi, pārlecot uz rindu ar iezīmi *L*.
- `border` *L* — turpināt programmas izpildi, pārlecot uz rindu ar iezīmi *L*, ja odometrs atrodas uz robežas un ir pavērsts uz režģa malu (t.i., komandai `move` nebūtu efekta); citādi programmas izpilde turpinās pa vecam un šai komandai nav efekta.
- `pebble` *L* — turpināt programmas izpildi, pārlecot un rindu ar iezīmi *L*, ja šūna, kurā atrodas odometrs, satur vismaz vienu akmentiņu; citādi programmas izpilde turpinās pa vecam un šai komandai nav efekta.

2.piemērs

Sekojošā programma liek odometram atrast pirmo (visvairāk uz rietumiem esošo) akmentiņu rindā 0 un tur arī apstāties; ja rindā 0 nav akmentiņu, odometrs apstāsies uz robežas rindas beigās. Programmā izmantotas divas iezīmes: `leonardo` un `davinci`.

```
right
leonardo:
pebble davinci  # atrasts akmentiņš
border davinci  # rindas beigas
move
jump leonardo
davinci:
halt
```

Odometrs sākumā pagriežas pa labi. Cikls sākas ar iezīmes deklarāciju `leonardo:` un beidzas ar komandu `jump leonardo`. Ciklā odometrs pārbauda, vai attiecīgajā šūnā ir akmentiņš, un to, vai šūna ir uz robežas; ja neviens no šiem nosacījumiem nav spēkā, odometrs izpilda komandu `move` un pārvietojas no šūnas (0, j) un blakus šūnu (0, j + 1), kas, kā zināms, eksistē. (Komanda `halt` šeit nav nepieciešama, jo programma beidzas jebkurā gadījumā.)

Uzdevums

Jums jāiesūta programma iepriekš aprakstītajā odometra valodā, kas liek odometram veikt atbilstošu darbību. Katram apakšuzdevumam (skatīt zemāk) tiek aprakstīts, kādu tieši darbību odometram ir jāveic un kādi ierobežojumi programmai ir jāievēro. Ir divu veidu ierobežojumi:

- *Programmas izmērs* — programmai jābūt pietiekami īsai. Programmas izmēru raksturo komandu

skaits tajā. Iezīmju deklarācijas, komentāri un tukšas rindas *netiek skaitītas* pie izmēra.

- *Izpildes garums* — programmai jābeidzas pietiekami ātri. Izpildes garumu raksturo izpildīto *soļu* skaits: katrs komandas izpildījums tiek skaitīts kā solis, neatkarīgi no tā, vai komandai bija kāds efekts vai ne; iezīmju deklarācijas, komentāri un tukšas rindas netiek skaitītas kā soļi.

Piemērā 1 programmas izmērs ir 4 un izpildes garums ir 4. Piemērā 2 programmas izmērs ir 6 un, kad izpildīta uz režģa ar vienu akmentiņu šūnā (0, 10), izpildes garums ir 43 soļi: `right`, 10 cikla iterācijas, kura katra ir 4 soļus gara (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), un beigās komandas `pebble davinci` un `halt`.

Apakšuzdevums 1 [9 punkti]

Sākumā šūnā (0, 0) ir x akmentiņi un šūnā (0, 1) ir y akmentiņi, visas citas šūnas ir tukšas. Atcerieties, ka katrā šūnā var atrasties ne vairāk par 15 akmentiņiem. Uzrakstiet programmu, kas tiek pārtraukta, kad odometrs atrodas šūnā (0, 0), ja $x \leq y$, vai šūnā (0, 1), ja $x > y$. (Virziens, kurā odometrs beigās ir vērsts, kā arī tas, cik akmentiņu pēc programmas izpildes beigām atrodas režģī un kā tie izkārtoti, nav svarīgi.)

Ierobežojumi: programmas izmērs ≤ 100, izpildes garums ≤ 1 000.

Apakšuzdevums 2 [12 punkti]

Tas pats uzdevums, kā iepriekšējais, bet, kad programma beidzas, šūnai (0, 0) jāsatur tieši x akmentiņi, bet šūnai (0, 1) jāsatur tieši y akmentiņi.

Ierobežojumi: programmas izmērs ≤ 200, izpildes garums ≤ 2 000.

Apakšuzdevums 3 [19 punkti]

Kaut kur rindā 0 ir tieši divi akmentiņi: viens šūnā (0, x), otrs šūnā (0, y); x un y ir dažādi un x + y ir pāra skaitlis. Uzrakstiet programmu, kas izpildes beigās atstāj odometru šūnā (0, (x + y) / 2), t.i., viduspunktā starp abām šūnām, kas satur akmentiņus. Režģa beigu stāvoklis nav svarīgs.

Ierobežojumi: programmas izmērs ≤ 100, izpildes garums ≤ 200 000.

Apakšuzdevums 4 [līdz 32 punktiem]

Režģis satur ne vairāk kā 15 akmentiņus, turklāt nekādi divi neatrodas vienā un tajā pašā šūnā. Uzrakstiet programmu, kas savāc visus akmentiņus ziemeļ-rietumu stūrī; precīzāk, ja režģis sākumā saturēja x akmentiņus, tad pēc programmas izpildes šūnai (0, 0) jāsatur tieši x akmentiņi un pārējām šūnām jābūt tukšām.

Iegūto punktu skaits par šo apakšuzdevumu ir atkarīgs no iesūtītās programmas izpildes garuma. Precīzāk, ja L ir maksimālais izpildes garums no vairākiem testa piemēriem, Jūsu iegūtais punktu skaits būs:

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

Ierobežojumi: programmas izmērs ≤ 200.

Apakšuzdevums 5 [līdz 28 punktiem]

Katra režģa šūna var saturēt jebkādu akmentiņu daudzumu (protams, starp 0 un 15, ieskaitot). Uzrakstiet programmu, kas atrod minimumu, t.i., programma, kura tiek pārtraukta, kad odometrs atrodas tādā šūnā (i, j), ka jebkura cita režģa šūna satur vismaz tikpat daudz akmentiņu, cik šūna (i, j). Pēc programmas izpildes beigām akmentiņu skaitam katrā šūnā jābūt tādam pašam kā sākumā.

Iegūto punktu skaits par šo apakšuzdevumu ir atkarīgs no iesūtītās programmas izpildes garuma. Precīzāk, ja L ir maksimālais izpildes garums no vairākiem testa piemēriem, Jūsu iegūtais punktu skaits būs:

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

Ierobežojumi: izpildes garums ≤ 44 400 000.

Implementācijas detaļas

Jums jāiesūta tieši viens fails katram apakšuzdevumam, kurš ir uzrakstīts atbilstoši izklāstītajiem sintakses nosacījumiem. Katra iesūtītā faila izmēram jābūt ne lielākam par 5 MiB. Katrā apakšuzdevumā Jūsu odometra kods tiks testēts uz vairākiem testa piemēriem, un jūs saņemsiet atbildes informāciju par jūsu koda izmantotajiem resursiem. Gadījumā, ja Jūsu kods nav sintaktiski pareizs un tāpēc nav testējams, Jūs saņemsiet informāciju par atbilstošo sintakses kļūdu.

Jūsu iesūtījumiem nav obligāti jāsatur odometra programmas visiem apakšuzdevumiem. Ja Jūsu pašreizējais iesūtījums nesatur odometra programmu apakšuzdevumam X, Jūsu pēdējais apakšuzdevuma X programmas iesūtījums tiks automātiski pievienots iesūtījumam; ja tādas programmas nav, šajā iesūtījumā attiecīgais apakšuzdevums tiks novērtēts ar 0 punktiem.

Kā parasti, iesūtījuma rezultāts ir visu apakšuzdevumu rezultātu summa, un galīgais rezultāts par uzdevumu ir maksimālais rezultāts starp visiem relīzes (release-tested) iesūtījumiem un pēdējo iesūtījumu.

Simulators

Testēšanas nolūkiem jums tiek piedāvāts odometra simulators, kuram var padot jūsu programmas un režģus. Odometra programmām jābūt rakstītām iesūtīšanas formātā (tajā, kas aprakstīts iepriekš).

Režģis jāapraksta šādā formātā: katrā faila rindā jābūt trīs skaitļiem, R, C un P, kas nozīmē, ka šūnā, kas atrodas rindā R un kolonā C atrodas P akmentiņi. Tiek pieņemts, ka visās šūnās, kas netiek aprakstītas režģa aprakstā, neatrodas neviens akmentiņš. Piemēram, failā:

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

Šajā failā aprakstītajā režģī ir 15 akmentiņi: 3 šūnā (0, 10) un 12 šūnā (4, 5).

Jūs varat izsaukt simulatoru izsaucot programmu `simulator.py`, kas atrodas uzdevumu mapē, norādot programmas faila vārdu kā parametru. Simulatoram ir šādas komandrindas opcijas:

- `-h` izvadīs īsu pārskatu par pieejamajām parametru iespējām;
- `-g GRID_FILE` no faila `GRID_FILE` ielādē režģa aprakstu (noklusētā vērtība: tukšs režģis);

- `-s GRID_SIDE` uzstāda režģa izmēru `GRID_SIDE x GRID_SIDE` (noklusētā vērtība: 256, kā uzdevuma specifikācijā); mazāka režģa izmēru lietošana var būt noderīga programmu atkļūdošanai;
- `-m STEPS` ierobežo simulācijas soļu izpildes skaitu - simulācija izpildīsies ne vairāk kā `STEPS` soļus;
- `-c` simulators tiek laists compilācijas modē; kompilācijas modē simulators atgriež precīzi to pašu izvadu, bet simulācija netik veikta Python, tā ģenerē mazu C programmu. Tā rezultātā programmai ir lielāka aizture sākumā, bet tā strādā daudz ātrāk; to ir ieteicams lietot, ja jūsu programmai ir paredzēts strādāt vairāk kā 10 000 000 soļus.

Iesūtījumu skaits

Šim uzdevumam ir atļauti ne vairāk kā 128 iesūtījumi.