Խճաքարերով օդոմետր

Լեոնարդոն հայտնագործեց յուրօրինակ "օդոմետր" ՝ սայլակառք, որը կարողանում է հեռավորություն չափել սայլակառի անիվի պտտվելու ընթացքում խճաքարեր գցելով: Խճաքարերը հաշվելով կարելի է իմանալ, թե անիվը քանի անգամ է պտտվել, որը թույլ է տալիս օգտվողին հաշվել օդոմետրի անցած ճանապարհը: Որպես ինֆորմատիկայի մասնագետներ, մեզ օդոմետրին ավելացրել եզ ծրագրային ապահովում, որն ընդլայնում է նրա ֆունկցիոնալությունը: Ձեր խնդիրն է ծրագրավորել օդոմետրի աշխատանքը ըստ ստորև բերված կանոնների:

" Գործողություններ ցանցում "

Օդոմետրը շարժվում է 256 × 256 չափի վանդակավոր ցանցում: Յուրաքանչյուր վանդակ կարող է պարունակել առավելագույնը 15 խճաքար և որոշվում է կոորդինատների զույգով (տող, սյուն), որտեղ յուրաքանչյուր կոորդինատ 0, …, 255 միջակայքից է: (i, j) վանդակի հարևան վանդակներն են (եթե դրանք գոյություն ունեն) (i - 1, j), (i + 1, j), (i, j - 1) and (i, j + 1): Առաջին կամ վերջին տողում, կամ առաջին կամ վերջին սյունում գտնվող վանդակները կոչվում են "սահման": Օդոմետրը միշտ սկսում է (0, 0) վանդակից (հյուսիս-արևմտյան անկյուն), ուղղված դեպի հյուսիս:

" Հիմնական հրամաններ "

Օդոմետրը կարելի է ծրագրավորել հետևյալ հրամանների միջոցով.

- `left` — պտտվել 90 աստիճանով դեպի ձախ (ժամացույցի սլաքի հակառակ ուղղությամբ) և մնալ ընթացիկ վանդակում (օրինակ, եթե նայում էր դեպի հարավ, այս հրամանը կատարելուց հետո կնայի դեպի արևելք):
- `left` — պտտվել 90 աստիճանով դեպի աջ (ժամացույցի սլաքի ուղղությամբ) և մնալ ընթացիկ վանդակում (օրինակ, եթե նայում էր դեպի արևմուտք, այս հրամանը կատարելուց հետո կնայի դեպի հյուսիս):
- `move` — շարժվել մեկ միավոր առաջ (որ ուղղությամբ որ նայում է օդոմետրը) հարևան վանդակը: Եթե այդպիսի վանդակ գոյություն չունի (այսինքն այդ ուղղությամբ արդեն հասել եզ սահմանին) ապա հրամանը դեր չի խաղում:

- `get` — հեռացնել մեկ խճաքար ընթացիկ վանդակից: Եթե ընթացիկ վանդակում խճաքար չկա, ապա այս հրամանը դեր չի խաղում:
- `put` — ավելացնել մեկ խճաքար ընթացիկ վանդակին: Եթե ընթացիկ վանդակում արդեն 15 խճաքար կա, ապա այս հրամանը չի կատարվում: Օդոմետրը երբեք խճաքարերի համար սահմանված չափը չի անցնում:
- `halt` — ավարտել աշխատանքը:

Օդոմետրը կատարում է հրամանները ծրագրում տրված հերթականությամբ, ամեն տողում մեկ հրաման: Դատարկ տողերը հաշվի չեն առնվում: `#` սիմվոլը նշանակում է մեկնաբանություն; նրան հաջորդող տեքստը մինչև տողի ավարտը հաշվի չեն առնվում: Եթե օդոմետրը հասնում է ծրագրի վերջին, նրա աշխատանքն ավարտվում է:.

Օրինակ 1

Դիտարկենք օդոմետրի համար նախատեսված հետևյալ ծրագիրը: Այն օդոմետրին տանում է (0, 2) վանդակը, ուղղությունը դեպի արևելք: (Նկատենք, որ առաջին `move` հրամանը անտեսվում է, քանի որ օդոմետրը գտնվում է հյուսիս-արևմտյան անկյունում և նայում է դեպի հյուսիս):

```
move  # դեր չի խաղում
right
# այժմ օդոմետրը նայում է դեպի արևելք
move
move
```

Նշիչներ, սահմաններ և խճաքարեր

Ընթացիկ իրավիճակից կախված ծրագրի աշխատանքի ընթացքը փոխելու համար դուք կարող եք օգտագործել նշիչներ, դրանք առավելագույնը 128 սիմվոլներից բաղկացած տողերը են, որոնցում մեծատառ-փոքրատառը տարբերելի է, տառերը պետք է լինեն հետևյալ տիրույթներից. `a`, ..., `z`, `A`, ..., `Z`, `0`, ..., `9`: Ստորև բերված են նշիչներին վերաբերող հրամանները: Ստորև բերված նկարագրություններում *L*-ը նշանակում է ցանկացած թույլատրելի նշիչ:

- *L*`:` (այսինքն *L*-ին հաջորդում է երկու կետ ‘`:`’) — ծրագրի տվյալ հատվածում հայտարարում է *L* նշիչը: Նշիչների բոլոր հայտարարությունները պետք է տարբերվեն: Նշիչի հայտարարությունը օդոմետրի աշխատանքի վրա չի ազդում:
- `jump` *L* — շարունակել աշխատանքը անցնելով *L* նշիչով տողին:

- `border` *L* — Շարունակել աշխատանքը անցնելով *L* նշինով տողին, եթե օդոմետրը գտնվում է սահմանի վրա և ուղղված է դեպի ցանցի եզրը (այսինքն `move` հրամանը դեր չի խաղում); հակառակ դեպքում, աշխատանքը նորմալ շարունակվում է և այս հրամանը ոչնչի վրա չի ազդում:

- `pebble` *L* — Շարունակել աշխատանքը անցնելով *L* նշինով տողին, եթե ընթացիկ վանդակում գոնե մեկ խճաքար կա; հակառակ դեպքում աշխատանքը նորմալ շարունակվում է և այս հրամանը ոչինչի վրա չի ազդում:

Օրինակ 2

Հետևյալ ծրագիրը տեղադրում է առաջին խճաքարը 0 տողում և կանգնում է այնտեղ; եթե 0 տողում խճաքարեր չկան, այն կանգնում է տողի վերջում սահմանի վրա: Ծրագիրն օգտագործում է երկու նշիչ՝ `leonardo` և `davinci`:

```
right
leonardo:
pebble davinci  # խճաքար է գտնվել
border davinci  # տողի վերջը
move
jump leonardo
davinci:
halt
```

Օդոմետրը սկսում է շրջելով դեպի աջ: Ցիկլը սկսում է `leonardo:` նշինի հայտարարությունից և ավարտվում է `jump leonardo` հրամանով: Ցիկլի մեջ օդոմետրը ստուգում է տողի վերջում խճաքարի առկայությունը: Եթե դա այդպես չէ, օդոմետրը `move` հրամանի միջոցով ընթացիկ (0, j) վանդակից անցնում է հարևան (0, j + 1) վանդակին, ճանի վերջինս գոյություն ունի: (`halt` հրամանը պարտադիր չէ որ լինի այստեղ, ճանի որ ծրագիրը ամեն դեպքում կավարտվի):

Խնդիրը

Դուք պետք է ուղարկեք ծրագիր օդոմետրի լեզվով, որը նկարագրված է վերևում: Այդ ծրագրի կատարման արդյունքում օդոմետրը պետք է իրեն պահի ինչպես սպասվում է: Ցուրաքանչյուր ենթախնդիր (տե՛ս ներքևում) նկարագրում է, թե օդոմետրը ինչպես պետք է գործի և, թե ձեր ծրագիրը ինչ սահմանափակումների պիտի բավարարի: Սահմանափակումները վերաբերում են հետևյալ երկու պահերին:

- *Ծրագրի չափը* — ծրագիրը պիտի բավականաչափ կարճ լինի: Ծրագրի չափը նրա մեջ եղած հրամանների ճանակն է: Նշինների հայտարարությունները, մեկնաբանությունները և դատարկ տողերը ծրագրի չափի մեջ "չեն հաշվվում":

- 'Կատարման երկարություն — *ծրագիրը պետք է բավականին շուտ ավարտվի: Կատարման երկարությունը կատարված հայլերի համակց է. յուրաքանչյուր հրամանի կատարում համարվում է մեկ հայլ անկախ նրանից այդ հրամանը ոնց խաղում է, թե ոչ: Նշիչների հայտարարությունները, մեկնաբանությունները և դատարկ տողերը քայլ չեն համարվում:*

Օրինակ 1-ում ծրագրի չափը 4 է, կատարման երկարությունը 4 է: Օրինակ 2-ում ծրագրի չափը 6 է, եթե այն կատարվում է մի ցանցում, որտեղ միայն մեկ խճաքար կա (0, 10) վանդակում, կատարման երկարությունը 43 հայլ է. `right` հայլը, ցիկլի 10 իտերացիա, յուրաքանչյուրը 4 հայլ (`pebble davinci; border davinci; move; jump leonardo`),և վերջում , `pebble davinci` և `halt`.հրամանների կատարումը:

Ենթախնդիր 1 [9 միավոր]

Սկզբում x խճաքար կա (0, 0) վանդակում և y խճաքար (0, 1) վանդակում, իսկ մնացած վանդակները դատարկ են: Հիշենք, որ մի վանդակում կարող է լինել առավելագույնը 15 խճաքար: Գրեք ծրագիր, որի կատարման արդյունքում օդոմետրը գտնվում է (0, 0) վանդակում, եթե $x \le y$, և (0, 1) վանդակում հակառակ դեպքում: (Կարևոր չէ, թե վերջում օդոմետրը ինչ ուղղությամբ է նայում: Նաև կարևոր չէ, թե ցանցում քանի խճաքար կլինի վերջում, կամ որտեղ նրանք տեղադրված կլինեն):

Սահմանափակումները. ծրագրի չափը ≤ 100, կատարման երկարությունը $\le 1\,000$.

Ենթախնդիր 2 [12 միավոր]

Նույն խնդիրը ինչ վերևում, բայց երբ ծրագիրն ավարտվում է, (0, 0) վանդակում պիտի լինի ճիշտ x խճաքար, իսկ (0, 1) վանդակում պիտի լինի ճիշտ y խճաքար:

Սահմանափակումները. ծրագրի չափը ≤ 200, կատարման երկարությունը $\le 2\,000$.

Ենթախնդիր 3 [19 միավոր]

Ճիշտ երկու խճաքար կա ինչ որ տեղ 0 տողում. մեկը գտնվում է (0, x) վանդակում, մյուսը՝ (0, y) վանդակում; x-ը և y-ը տարբեր են, և x + y-ը զույգ է: Գրեք ծրագիր, որը օդոմետրին հասցնում է (0, (x + y) / 2) վանդակը, այսինքն, խճաքարար պարունակող երկու վանդակների ճիշտ մեջտեղում: Ցանցի վերջնական վիճակը կարևոր չէ:

Սահմանափակումները. ծրագրի չափը ≤ 100, կատարման երկարությունը $\le 200\,000$:

Ենթախնդիր 4 [մինչև 32 միավոր]

Ցանցում կա առավելագույնը 15 խճաքար, յուրաքանչյուր վանդակում կա առավելագույնը մեկ խճաքար: Գրեք ծրագիր, որը հավաքում է բոլոր խճաքարերը հյուսիս-արևմտյան անկյունում: Ավելի ստույգ, եթե սկզբում ցանցում կար x խճաքար, վերջում (0, 0) վանդակում պիտի լինի x խճաքար, իսկ մնացած վանդակները պիտի դատարկ լինեն:

Այս ենթախնդրի համար միավորի չափը կախված է submit արած ծրագրի կատարման երկարությունից: Ավելի ստույգ, եթե տարբեր test case-երում կատարման երկարության մաքսիմումը L է, ձեր միավորը կլինի.

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

Սահմանափակումները. ծրագրի չափը ≤ 200.

Ենթախնդիր 5 [մինչև 28 միավոր]

Ցանցի վանդակներից յուրաքանչյուրում կարող է լինել կամայական ֆանակով (հարկե, 0-ից 15 սահմանափակում) խճաքար: Գրեք ծրագիր, որը գտնում է մինիմումը, այսինքն նրա աշխատանքի արդյունքում օդոմետրը գտնվում է այնպիսի (i, j) վանդակում, որ մնացած վանդակներից յուրաքանչյուրում առնվազն նույնքան խճաքար կա, քան (i, j) վանդակում: Ծրագրի կատարումից հետո յուրաքանչյուր վանդակում պիտի նույնքան խճաքար լինի, որքան կար նախքան ծրագիրն աշխատացնելը:

Այս ենթախնդրի միավորը կախված է submit արած ծրագրի P չափից: Ավելի ստույգ, ձեր միավորը կլինի.

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

Սահմանափակումներ. կատարման երկարությունը ≤ 44 400 000.

Իրականացման մանրամասներ

Դուք պետք է յուրաքանչյուր ենթախնդրի համար submit անեք ճիշտ մեկ ֆայլ, գրված վերը նկարագրված երականական կանոններին համաձայն: Յուրաքանչյուր submit արած ֆայլի երկարությունը պիտի լինի առավելագույնը 5 MiB: . Յուրաքանչյուր ենթախնդրի համար ձեր օդոմետրը կթեստավորվի մի քանի թեստերի խմբերի վրա, և դուք կստանաք ձեր կոդի կողմից օգտագործած ռեսուրսների վերաբերյալ տեղեկություն: Եթե ձեր կոդը երականական սխալներ պարունակի և այն թեստավորել հնարավոր

չլինի, դուք կստանաք ինֆորմացիա քերականական սխալների վերաբերյալ:

Պարտադիր չէ, որ ձեր յուրաքանչյուր submit ընդգրկի ծրագրեր բոլոր ենթախնդիրների համար: Եթե ձեր ընթացիկ submit-ը X ենթախնդրի համար ծրագիր չի պարունակում, ավտոմատ կերպով կվերցվի X ենթախնդրի համար ձեր այն ծրագիրը, որն ավելի ուշ է submit արվել: Եթե այդպիսի ծրագիր չկա, տվյալ ենթախնդրի համար կտրվի զրո միավոր:

Որպես կանոն, submit-ի համար տրված միավորը յուրաքանչյուր ենթախնդրից ստացած միավորների գումարն է, և վերջնական միավորը release-թեստավորումով submit-ներից և վերջին submit-ից ստացված միավորներից մաքսիմումն է:

Սիմուլյատոր

Թեստավորման համար ձեր տրամադրության տակ կա օդոմետրի սիմուլյատոր, որը կարող կատարել ձեր ծրագիրը մուտքայինը ցանցի վրա: Օդոմետրի ծրագիրը գրված կլինի նույն ֆորմատով, ինչը օգտագործվում է submit-ի համար (այսինքն, որը նկարագրված է վերևում):

Ցանցի նկարագրությունը տրված կլինի հետևյալ ֆորմատով. ֆայլի յուրաքանչյուր տողը պետք է պարունակի երեք ամբողջ թիվ՝ R, C և P, նշանակում է, որ R տողի C այդ վանդակում կա P խճաքար: Բոլոր այն վանդակներում, որոնք նկարագրված չեն, խճաքար չեն պարունակում: Օրինակ, դիտարկենք հետևյալ ֆայլը.

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

Այս ֆայլում նկարագրված ցանցը կպարունակի 15 խճաքար՝ 3-ը (0, 10) վանդակում և 12-ը (4, 5) վանդակում:

Դուք կարող եք թեստավորել ձեր ծրագիրը սիմուլյատորի միջոցով աշխատացնելով `simulator.py` ծրագիրը ձեր խնդրի ֆոլդերում թերմինալի միջոցով ձեր ծրագրի ֆայլի անունը սիմուլյատորին տալով որպես արգումենտ: Սիմուլյատորը կընդունի հրամայական տողի հետևյալ օպցիաները.

- `-h`-ը կտա հնարավոր օպցիաների համառոտ նկարագրությունը.
- `-g GRID_FILE` բեռնում է ցանցի նկարագրությունը `GRID_FILE` ֆայլից (լռությամբ ցանցը դատարկ է):
- `-s GRID_SIDE` ցանցի չափը դնում է հավասար `GRID_SIDE` x `GRID_SIDE` (լռությամբ 256, as used in the problem specification); փոքր ցանցերը կարող են օգտակար լինել ծրագիրը մշակելու համար:
- `-m STEPS` սահմանափակում է կատարման քայլերի քանակը սիմուլյացիայի ժամանակ առավելագույնը `STEPS`-ով:

- `-c` մտնում է compilation ռեժիմի: compilation ռեժիմում, սիմուլյատորը վերադարձնում է ֆիշտ նույն ելքը, բայց |Պիթոնով աշխատելու փոխարեն, այն գեներացնում է և կոմպիլացնում փոֆիկ C ծրագիր: Սա սկզբում բարդ է թվում, բայց եթե հարկ է լինում. բայց հետո էապես արագ արդյունք է տալիս: Խորհուրդ է տրվում այն օգտագործել, եթե ձեր ծրագիրը կատարելու է ավելի քան 10 000 000 քայլ:

submit-ների քանակը

Այս խնդրի համար առավելագույն submit-ների քանակը 128 է: