კენჭებიანი ოდომეტრი

ლეონარდომ ორიგინალური ოდომეტრი გამოიგონა (ოდომეტრი არის ხელსაწყო, რომელიც ზომავს გავლილ მანძილს): ეს არის ურიკა, რომელსაც შეუძლია მანძილის გაზომვა ყოველ იმ ადგილზე თითო კენჭის ჩამოგდებით, სადაც იგი შემობრუნდება. კენჭების დათვლის შემდეგ შესაძლებელი ხდებოდა ურიკის შემობრუნებათა რაოდენობის დათვლა, რაც ოდომეტრის მიერ გავლილი მანძილის გამოთვლის საშუალებას იძლეოდა . როგორც ნამდვილმა პროგრამისტებმა, ჩვენ უნდა დავუმატოთ ოდომეტრს პროგრამული კონტროლი, რაც გაზრდის და შეცვლის მის ფუნქციონალურ შესაძლებლობებს. ჩვენი ამოცანაა დავწეროთ პროგრამა, რომელიც დააკმაყოფილებს ქვემოთ მოყვანილ წესებს.

სამუშაო ზედაპირი

ოდომეტრი გადაადგილდება 256x256 კვადრატული ფორმის უჯრიან ზედაპირზე. თითოეულ უჯრაში არ შეიძლება 15-ზე მეტი კენჭი იყოს. თითოეული უჯრა მოიცემა კოორდინატთა წყვილით (სტრიქონი,სვეტი), სადაც ყოველი კოორდინატი 0,...,255 დიაპაზონშია. (I,J) უჯრისათვის მეზობელი უჯრებია (თუკი ისინი არსებობენ) (I - 1,J), (I + 1,J), (I,J-1) და (I,J+1). უჯრებს, რომლებიც არიან პირველ ან ბოლო სტრიქონში, ანდა პირველ ან ბოლო სვეტში, ეწოდებათ საზღვრები. ოდომეტრი ყოველთვის იწყებს (0,0) უჯრიდან (უკიდურესი ჩრდილო-დასავლეთი უჯრა) და ორიენტირებულია ჩრდილოეთისკენ.

ძირითადი ბრძანებები

ოდომეტრი შეიძლება დაპროგრამდეს შემდეგი ბრძანებების გამოყენებით:

- `left` — შემობრუნდება 90 გრადუსით მარცხნივ (საათის ისრის საწინააღმდეგოდ) და რჩება იგივე უჯრაზე (მაგალითად, თუ ოდომეტრი მიმართული იყო სამხრეთით, მაშინ იგი ბრძანების შესრულების შემდეგ მიმართული გახდება აღმოსავლეთით).
- `right` — შემობრუნდება 90 გრადუსით მარჯვნივ (საათის ისრის მიმართულებით) და რჩება იგივე უჯრაზე (მაგალითად, თუ ოდომეტრი მიმართული იყო დასავლეთით, მაშინ იგი ბრძანების შესრულების შემდეგ მიმართული გახდება ჩრდილოეთით).

- `move` — გადააადგილებს წინა მეზობელ უჯრაზე (იმ მიმართულებით, საითკენაც ოდომეტრია ორიენტირებული). თუ ასეთი უჯრა არ არის (მაგალითად, თუ საზღვარია მიღწეული ოდომეტრის ორიენტაციის მიმართულებით) მაშინ ბრძანებას ეფექტი არ ექნება, ანუ იგნორირდება.
- `get` — მიმდინარე უჯრიდან ერთი კენჭის აღება. თუ მიმდინარე უჯრაში კენჭები არ არის, მაშინ ბრძანებას ეფექტი არ ექნება, ანუ იგნორირდება.
- `put` — მიმდინარე უჯრაში ერთი კენჭის დამატება. თუ მიმდინარე უჯრაში 15 კენჭია, მაშინ ბრძანებას ეფექტი არ ექნება, ანუ იგნორირდება.
- `halt` — პროგრამის დასრულება.

ოდომეტრი ბრძანებებს იმ თანმიმდევრობით ასრულებს, რა თანმიმდევრობითაც პროგრამაშია მოცემული (თითო ხაზში — თითო ბრძანება). ცარიელი სტრიქონები იგნორირდება. `#` სიმბოლო აღნიშნავს კომენტარების დასაწყისს - ნებისმიერი ტექსტი სტრიქონის ბოლომდე იგნორირდება. თუ ოდომეტრი მივიდა პროგრამის ბოლომდე — პროგრამა მთავრდება.

მაგალითი 1

განვიხილოთ შემდეგი პროგრამა ოდომეტრისათვის. მისი შესრულების შემდეგ ოდომეტრი აღმოჩნდება (0,2) უჯრაში და ორიენტირებული იქნება აღმოსავლეთით (მიაქციეთ ყურადღება, რომ პირველი `move` იგნორირებული იქნება, რადგან ოდომეტრი თავიდან (0,0) უჯრაშია და ჩრდილოეთისკენაა ორიენტირებული).

```
move  # არაფერს აკეთებს
right
# ახლა ოდომეტრი მიმართულია აღმოსავლეთისაკენ.
move
move
```

ჭდეები,საზღვრები და კენჭები.

იმისათვის, რომ შევცვალოთ პროგრამის მსვლელობა, ჩვენ შეგვიძლია გამოვიყენოთ ჭდეები, რომელიც არის მაქსიმუმ 128 სიმბოლო. სიმბოლოები შეიძლება იყოს `a`, ..., `z`, `A`, ..., `Z`, `0`, ..., `9`. ჭდეებთან დაკავშირებული ახალი ბრძანებები ქვემოთაა აღწერილი. *L* სიმბოლო ქვემოთ ყველგან აღწერს კორექტულ ჭდეს.

- *L*`:` (ე.ი. *L*, რომლის შემდეგაც არის ორწერტილი ‘ `:`’) — განსაზღვრავს *L* ჭდის მდებარეობას პროგრამაში. ყველა ჭდე უნდა იყოს უნიკალური. ჭდის გამოცხადება ოდომეტრზე გავლენას არ ახდენს.
- `jump` *L* — განაგრძოთ შესრულება ნებისმიერ შემთხვევაში *L* ჭდეზე გადასვლით.

- `border L` — განვაგრძოთ შესრულება `L` ჭდეზე გადასვლით იმ შემთხვევაში, თუ ოდომეტრი იმყოფება საზღვარზე და მიმართულია გარეთ (ანუ `move` ინსტრუქცია არაფერს არ აკეთებს). წინააღმდეგ შემთხვევაში, შესრულება გაგრძელდება ჩვეულებრივი რიგით და ბრძანება არაფერს შეცვლის.

- `pebble L` — განვაგრძოთ შესრულება `L` ჭდეზე გადასვლით იმ შემთხვევაში, თუ მიმდინარე უჯრაში არის ერთი მაინც კენჭი. წინააღმდეგ შემთხვევაში შესრულება გაგრძელდება ჩვეულებრივი რიგით და ბრძანება არაფერს შეცვლის.

მაგალითი 2

ქვემოთ მოყვანილი პროგრამის შესრულების შედეგად ოდომეტრი განსაზღვრავს 0-ვან სტრიქონში პირველი კენჭის (ყველაზე დასავლეთით) მდებარეობას და გაჩერდება იქ. თუ 0-ვან სტრიქონში კენჭები არ არის, მაშინ ოდომეტრი გაჩერდება საზღვარზე - სტრიქონის ბოლოში. ამ პროგრამაში გამოყენებულია ორი ჭდე: `leonardo` და `davinci`.

```
right
leonardo:
pebble davinci  # კენჭი ნაპოვნია
border davinci  # სტრიქონის ბოლო
move
jump leonardo
davinci:
halt
```

თავიდან ოდომეტრი მობრუნდება მარჯვნივ. ციკლი იწყება `leonardo:` ჭდიდან და მთავრდება ბრძანებით `jump leonardo`. ამ ციკლში ოდომეტრი ამოწმებს არის თუ არა კენჭები მიმდინარე უჯრაში, ან მიაღწია თუ არა საზღვარს. თუ ეს ასე არ არის, მაშინ ოდომეტრი ასრულებს `move` ბრძანებას და გადაადგილდება (0, j) წერტილიდან (0, j + 1) წერტილში. აქ შეიძლება `halt` ბრძანების გარეშეც მოქმედება, რადგანაც პროგრამის ბოლო სტრიქონზე მიღწევის გამო, პროგრამა მაინც დაამთავრებს შესრულებას.

ამოცანა

თქვენ უნდა გააგზავნოთ ოდომეტრ-ენაზე დაწერილი პროგრამა, რომელიც აღწერილია ზემოთ. ქვემოთ აღწერილ ყოველ ქვეამოცანაში მოითხოვება ოდომეტრის მოქმედებათა რაოდენობა ისე, რომ დაცული იქნას ორი შემდეგი შეზღუდვა:

- *პროგრამის ზომა* — პროგრამა უნდა იყოს საკმარისად მოკლე. პროგრამის ზომა - ეს არის ბრძანებების რაოდენობა მასში. ჭდეების განსაზღვრა, კომენტარები და ცარიელი სტრიქონები არ გაითვალისწინება პროგრამის ზომის გამოთვლის დროს.

- *ოპერაციების რაოდენობა' ' — ეს პროგრამა უნდა იყოს საკმარისად სწრაფი. ოპერაციათა რაოდენობა — ეს არის შესრულებული ბიჯების რაოდენობა: ყოველი ცალკეული ბრძანება ითვლება როგორც ერთი ბიჯი, მიუხედავად იმისა, ჰქონდა თუ არა ამ ბრძანებას ეფექტი. ჭდეები, კომენტარები და ცარიელი სტრიქონები ბიჯად არ ითვლება.*

მაგალითი 1-ში პროგრამის ზომა არის 4 და ოპერაციათა რაოდენობაც არის 4. მაგალითი 2-ში პროგრამის ზომა არის 6. თუ პროგრამას გავუშვებთ (0,10) უჯრაში 1 ქვით, მაშინ ოპერაციათა რაოდენობა იქნება 43: მართლაც, `right`, ციკლის 10 იტერაციიდან თითოეული შედგება 4 ბიჯისაგან (`pebble davinci`; `border davinci`; `move`; `jump leonardo`),და ბოლოს ბრძანებები `pebble davinci` and `halt`..

ქვეამოცანა 1 [9 ქულა]

დასაწყისში (0, 0) უჯრაში არის x ცალი კენჭი და (0, 1) უჯრაში y ცალი. ამასთანავე, დანარჩენი უჯრები ცარიელია. მიაქციეთ ყურადღება, რომ ნებისმიერ უჯრაში არ შეიძლება 15 კენჭზე მეტი იყოს. საჭიროა დავწეროთ პროგრამა, რომლის შესრულების შემდეგ ოდომეტრი იქნება (0, 0) უჯრაში, თუ x ≤ y და (0, 1) უჯრაში - წინააღმდეგ შემთხვევაში. ჩვენ არ გვაინტერესებს ოდომეტრის ორიენტაცია, კენჭების რაოდენობა და მათი განლაგება.

შეზღუდვები':' პროგრამის ზომა ≤ 100, ოპერაციათა რაოდენობა ≤ 1 000.

```
== ქვეამოცანა 2 [12 ქულა] ==
```

ისეთივე ქვეამოცანაა, როგორც ზემოთ აღწერილი, მხოლოდ პროგრამის დამთავრების შემდეგ (0,0) უჯრაში უნდა იყოს x ცალი კენჭი, ხოლო (0,1) უჯრაში - ზუსტად y ცალი კენჭი.

შეზღუდვები':' პროგრამის ზომა ≤ 200, ოპერაციათა რაოდენობა ≤ 2 000.

```
==  ქვეამოცანა 3 [19ქულა] ==
```

0 ინდექსის მქონე სტრიქონში არის ზუსტად 2 კენჭი: ერთი (0, x) უჯრაში, ხოლო მეორე - (0, y) წერტილში (x და y განსხვავებულია; x + y ლუწია). საჭიროა დავწეროთ პროგრამა, რომლის შესრულების შემდეგ ოდომეტრი აღმოჩნდება (0, (x + y) / 2) უჯრაში, ანუ ზუსტად ორი მოცემული უჯრის შუაში. უჯრების განლაგებას მნიშვნელობა არ აქვს.

შეზღუდვები':' პროგრამის ზომა ≤ 100, ოპერაციათა რაოდენობა ≤ 200 000.

```
==  ქვეამოცანა 4 [32 ქულამდე] ==
```

დაფაზე არის არაუმეტეს 15 კენჭისა, არცერთი წყვილი არ არის ერთ და იმავე უჯრაში. საჭიროა დავწეროთ პროგრამა, რომელიც შეაგროვებს ყველა კენჭს ჩრდილო-დასავლეთ კუთხეში. უფრო ზუსტად, თუ დასაწყისში დაფაზე

იყო x კენჭი, მაშინ პროგრამის დამთავრების შემდეგ (0,0) უჯრაში უნდა იყოს ზუსტად x კენჭი, ხოლო დანარჩენ უჯრაში არ უნდა იყოს.

ქულების რაოდენობა ამ ქვეამოცანაში დამოკიდებულია ოპერაციათა რაოდენობაზე, რომელიც შესრულდება შემოწმებაზე გაგზავნის შემდეგ. კერძოდ, თუ L არის მაქსიმალური რაოდენობა ოპერაციებისა სხვადასხვა შემავალი მონაცემებისთვის, რომლებიც შეესაბამებიან ამ ქვეჯგუფს, მაშინ თქვენ მიიღებთ ქულათა შემდეგ რაოდენობას:

- 32 ქულა, თუ $L \leq 200\,000$;

$32 - 32 \log_{10}(L / 200\,000)$ ქულა, თუ $200\,000 < L < 2\,000\,000$;

- 0 ქულა, თუ $L \geq 2\,000\,000$.

შეზღუდვა: პროგრამის ზომა ≤ 200.

ქვეამოცანა 5 [28 ქულამდე]

ყოველ უჯრაში შეიძლება იყოს კენჭების ნებისმიერი რაოდენობა (0-დან 15-ის ჩათვლით). უნდა დავწეროთ პროგრამა, რომელიც იპოვის მინიმუმს, ანუ დამთავრების შემდეგ ოდომეტრი არის ისეთ (i, j) უჯრაში, რომ ნებისმიერი უჯრა შეიცავს როგორც მინიმუმ იმდენივე ქვას, რამდენიც არის (i, j) უჯრაში. პროგრამის შესრულების შემდეგ ყოველ უჯრაში კენჭების რაოდენობა ისეთივე უნდა იყოს, როგორც პროგრამის გაშვებამდე.

ქულების რაოდენობა ამ ქვეამოცანაში დამოკიდებულია პროგრამის P ზომაზე. კერძოდ, თქვენ მიიღებთ შემდეგ ქულათა რაოდენობას:

```
* 28 ქულა, თუ P ≤ 444;
```

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

შეზღუდვა: ოპერაციათა რაოდენობა ≤ 44 400 000.

რეალიზაციის დეტალები

თქვენ უნდა გააგზავნოთ შემოწმებაზე ზუსტად ერთი ფაილი თითოეული ქვეამოცანისათვის, რომელიც შედგენილია ზემოთ აღწერილი სინტაქსით. თითოეული ფაილის ზომა არ უნდა აღემატებოდეს 5 მბ-ს. ყოველი ქვეამოცანისთვის თქვენი პროგრამა გაიტესტება მონაცემთა რამდენიმე ნაკადისათვის და თქვენ მიიღებთ ინფორმაციას გამოყენებული რესურსების შესახებ. თუ თქვენი კოდი არ არის სინტაქსურად კორექტული, თქვენ მიიღებთ შეტყობინებას სინტაქსური შეცდომის შესახებ.

არაა სავალდებულო, ერთდროულად გააგზავნოთ ყველა ქვეამოცანის

ამოხსნები. თუ თქვენი მიმდინარე გზავნილი არ შეიცავს პროგრამას ქვეამოცანისათვის, მაშინ ყველაზე ბოლო გზავნილი ამ ქვეამოცანისათვის ჩართული იქნება ამ გზავნილში. თუ თქვენ არ გაგიგზავნიათ პროგრამა შესამოწმებლად ამ ამოცანის ამოსახსნელად, მაშინ ეს ქვეამოცანა ამ გზავნილში შეფასდება 0 ქულით. X

როგორც წესი, ქულების რაოდენობა გაგზავნისათვის უდრის ქულათა ჯამს ყველა ქვეამოცანაში და საბოლოო ქულა ამოცანისათვის ტოლი იქნება მაქსიმალური ჯამისა release-tested გაგზავნებსა და ბოლო გაგზავნას შორის.

სიმულატორი

პროგრამის გასამართად თქვენ მოგეცემათ ოდომეტრის სიმულატორი, რომლისთვისაც აუცილებელია პროგრამაც და ბადეც.

ბადე აღწერილი უნდა იყოს შემდეგი ფორმატით: ყოველი სტრიქონი შეიცავს 3 რიცხვს: R, C და P, რომლებიც აღნიშნავენ, რომ R სტრიქონის და C სვეტის გადაკვეთაზე მდებარე უჯრა შეიცავს P კენჭს. იგულისხმება, რომ თუ უჯრა აღწერილი არ არის - კენჭებს არ შეიცავს. მაგალითად, განვიხილოთ ფაილი

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

აქ ბადე შეიცავს 15 კენჭს: 3 - (0,10) უჯრაში და 12 - (4,5) უჯრაში

თქვენ შეგიძლიათ გაუშვათ `simulator.py` -ს საშუალებით თქვენს ფოლდერში, პროგრამული ფაილის არგუმენტად გადაეცემით. სიმულატორი მიიღებს პარამეტრებს ბრძანებათა სტრიქონიდან:

- `-h` დახმარება;
- `-g GRID_FILE` ჩატვირთავს ბადეს `GRID_FILE` ფაილიდან (default: ბადე ცარიელია);
- `-s GRID_SIDE` განსაზღვრავს ბადის ზომას `GRID_SIDE x GRID_SIDE` (ბადის default ზომა: 256); პროგრამის გამართვისას შეიძლება უმჯობესი იქნეს მცირე ზომის ბადე
- `-m STEPS` სიმულატორის ბიჯების შეზღუდვა `STEPS`;
- `-c` გადადის კომპილაციის რეჟიმში, ეს პარამეტრი გამოიყენეთ, თუ ბიჯების რაოდენობა 10 000 000-ზე მეტია.

გაშვებათა რაოდენობა

მაქსიმალური რაოდენობა - 128