Zadanie: Odometr na kamyki

Polish version, v. 1.2

Jednym z wynalazków Leonarda da Vinci jest *odometr* (drogomierz). Stanowi on rodzaj pojazdu służący do mierzenia odległości za pomocą kamyków, które wypadają w miarę kręcenia się kółek. Liczba upuszczonych kamyków jest równa liczbie obrotów kółek, co pozwala wyznaczyć odległość przebytą przez odometr. Jako informatycy będący fanami Leonarda, postanowiliśmy dodać do jego odometru oprogramowanie sterujące oraz nieco poszerzyć jego możliwości. Twoim zadaniem jest oprogramowanie odometru zgodnie z poniższą specyfikacją.

Opis planszy

Nasz odometr będzie poruszał się po planszy złożonej z 256 × 256 pól. Na każdym z pól może znajdować się maksymalnie 15 kamyków. Położenie pola określamy za pomocą pary współrzędnych (wiersz, kolumna), przy czym każda współrzędna jest z zakresu 0, …, 255. Pola sąsiadujące z polem (i, j) to (jeśli istnieją): (i - 1, j), (i + 1, j), (i, j - 1) oraz (i, j + 1). Pola znajdujące się w pierwszym lub ostatnim wierszu planszy, a także pola w pierwszej i ostatniej kolumnie planszy, nazywamy polami *brzegowymi*. Na początku odometr znajduje się na polu (0, 0) (czyli w północno-zachodnim narożniku planszy) i jest zwrócony na północ.

Podstawowe komendy

Odometr wykonuje program składający się z następujących komend:

- `left` — skręć o 90 stopni w lewo (przeciwnie do kierunku ruchu wskazówek zegara) i pozostań na obecnie zajmowanym polu (np. jeśli przed wykonaniem tej komendy odometr był zwrócony na południe, to po jej wykonaniu będzie zwrócony na wschód).
- `right` — skręć o 90 stopni w prawo (zgodnie z kierunkiem ruchu wskazówek zegara) i pozostań na obecnie zajmowanym polu (np. jeśli przed wykonaniem tej komendy odometr był zwrócony na zachód, to po jej wykonaniu będzie zwrócony na północ).
- `move` — wykonaj ruch o jedno pole naprzód (czyli w kierunku, w którym jest obecnie zwrócony odometr). Jeśli docelowe pole nie istnieje (tzn. odometr dotarł już do brzegu planszy w rozważanym kierunku), to ta komenda nie ma żadnego efektu.
- `get` — zabierz jeden kamyk z obecnie zajmowanego pola. Jeśli na polu nie ma żadnego kamyka, ta komenda nie ma żadnego efektu.
- `put` — upuść jeden kamyk na obecnie zajmowane pole. Jeśli pole zawiera już 15 kamyków, ta komenda nie ma żadnego efektu. Odometr ma zawsze w zapasie dostateczną liczbę kamyków.
- `halt` — zakończ działanie odometru.

Odometr wykonuje komendy w kolejności, w której występują one w programie. Każdy wiersz programu powinien zawierać co najwyżej jedną komendę. Puste wiersze programu są ignorowane. Symbol `#` oznacza komentarz; wszystko, co znajduje w wierszu za znakiem `#`, jest ignorowane. Jeśli odometr dojdzie do końca programu, kończy swoje działanie.

Przykład 1. Rozważmy poniższy, bardzo prosty program. Na końcu działania programu odometr znajduje się na polu (0, 2) i jest zwrócony na wschód. Zauważ, że w poniższym programie pierwsza komenda `move` zostanie zignorowana, ponieważ na początku odometr znajduje się w północno-zachodnim narożniku planszy i jest zwrócony na północ.

```
move  # nie ma żadnego efektu
right
# teraz odometr jest zwrócony na wschód
move
move
```

Etykiety

Aby modyfikować działanie programu zależnie od chwilowego stanu wykonania, będziemy używać etykiet. Etykiety to napisy złożone z co najwyżej 128 symboli a, …, z, A, …, Z, 0, …, 9 (w etykietach rozróżniamy wielkie i małe litery). Poniżej znajduje się lista komend związanych z etykietami. W poniższym opisie *L* oznacza jakąkolwiek poprawną etykietę.

- `L:` (tj. *L* plus dwukropek '`:`') — deklaruje w danym miejscu programu etykietę *L*. Wszystkie zadeklarowane etykiety muszą być różne. Deklaracja etykiety nie zmienia w żaden sposób ruchu odometru.
- `jump` *L* — kontynuuj działanie programu od miejsca, w którym zadeklarowano etykietę *L*.
- `border` *L* — przeskocz do miejsca w programie, w którym zadeklarowano etykietę *L*, pod warunkiem, że w chwili wykonywania instrukcji odometr znajduje się na brzegu i jest zwrócony ku krawędzi planszy (tzn. w tym położeniu komenda `move` nie miałaby żadnego efektu); w przeciwnym razie komenda `border` *L* nie ma żadnego efektu, a program kontynuuje działanie od następnej komendy.
- `pebble` *L* — przeskocz do miejsca w programie, w którym zadeklarowano etykietę *L*, pod warunkiem, że w chwili wykonywania instrukcji na polu zajmowanym przez odometr znajduje się co najmniej jeden kamyk; w przeciwnym razie ta komenda nie ma żadnego efektu, a program kontynuuje działanie od następnej komendy.

Przykład 2. Poniższy program znajduje pierwszy (skrajnie lewy) kamyk w wierszu 0 i w tym miejscu zatrzymuje odometr. Jeśli wiersz 0 nie zawiera żadnych kamyków, odometr zatrzymuje się na końcu tego wiersza. W programie używamy dwóch etykiet, `leonardo` i `davinci`.

```
right
leonardo:
pebble davinci  # znaleziono kamyk
border davinci  # koniec wiersza
move
jump leonardo
davinci:
halt
```

Na początku odometr wykonuje obrót w prawo. W programie znajduje się pętla, której początek stanowi deklaracja etykiety `leonardo`, a koniec stanowi komenda `jump leonardo`. Wewnątrz pętli odometr sprawdza, czy znajduje się na polu zawierającym kamyk albo na końcu wiersza, a jeśli nie, wykonuje komendę `move`, wskutek której przemieszcza się z bieżącego pola (0, j) na sąsiednie pole (0, j + 1) (wiadomo teraz, że to pole istnieje). Zauważ, że komenda `halt` nie jest tu konieczna, gdyż bez niej program i tak by się zakończył.

Zadanie

Napisz program sterujący odometrem (zgodnie z językiem programowania wyspecyfikowanym powyżej), który spowoduje, że odometr zrealizuje cele wyznaczone mu w poszczególnych podzadaniach. Każde podzadanie zawiera pewne ograniczenia, które Twój program musi spełniać. Ograniczenia te dotyczą dwóch następujących pojęć.

- *Długość programu* — jest to liczba komend zawartych w programie. Deklaracje etykiet, komentarze i puste wiersze w programie *nie są brane pod uwagę* przy obliczaniu długości programu.
- *Czas wykonania* — jest to liczba *kroków* wykonanych przez program. Krokiem nazywamy dowolne wykonanie komendy, niezależnie od tego, czy ma ono jakikolwiek efekt. Deklaracje etykiet, komentarze i puste wiersze w programie nie są liczone jako kroki.

W Przykładzie 1 długość programu to 4 i czas wykonania to 4. W Przykładzie 2 długość programu to 6, a czas wykonania dla planszy zawierającej tylko jeden kamyk, na polu (0, 10), to 43. Poszczególne kroki to: komenda `right`, następnie 10 obrotów pętli, z których każdy składa się z 4 kroków (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), a na końcu komendy `pebble davinci` oraz `halt`.

Podzadanie 1. [9 punktów]

Na początku na polu (0, 0) znajduje się x kamyków, a na polu (0, 1) znajduje się y kamyków. Wszystkie pozostałe pola są puste. Pamiętaj, że na każdym polu znajduje się co najwyżej 15 kamyków. Napisz program, po wykonaniu którego odometr zakończy działanie na polu (0, 0), jeśli $x \leq y$, a w przeciwnym przypadku na polu (0, 1). (Nie interesuje nas kierunek, w którym zwrócony jest odometr na końcu programu; nie interesuje nas również to, ile kamyków znajduje się na końcu na planszy ani gdzie się one znajdują.)

Ograniczenia: długość programu ≤ 100, czas wykonania ≤ 1 000.

Podzadanie 2. [12 punktów]

Twoje zadanie jest takie samo, jak poprzednio, tylko że na końcu programu na polu (0, 0) musi znajdować się x kamyków, a na polu (0, 1) y kamyków.

Ograniczenia: długość programu ≤ 200, czas wykonania ≤ 2 000.

Podzadanie 3. [19 punktów]

Gdzieś w wierszu 0 znajdują się dwa kamyki: jeden na polu (0, x), a drugi na polu (0, y). Liczby x i y są różne, a ich suma jest parzysta. Napisz program, który umieszcza odometr w polu o współrzędnych (0, (x + y) / 2), tj. w polu znajdującym się dokładnie w połowie drogi między rozważanymi polami zawierającymi kamyki. Końcowa zawartość pól planszy jest tu nieistotna.

Ograniczenia: długość programu ≤ 100, czas wykonania ≤ 200 000.

Podzadanie 4. [co najwyżej 32 punkty]

Na planszy znajduje się co najwyżej 15 kamyków, przy czym każde pole zawiera co najwyżej jeden kamyk. Napisz program, który zbierze je wszystkie w północno-zachodnim narożniku planszy. Dokładniej, jeśli na początku plansza zawierała x kamyków, to na końcu na polu (0, 0) musi znajdować się dokładnie x kamyków, a wszystkie pozostałe pola muszą być puste.

Liczba punktów, jaką zdobędziesz za to podzadanie, będzie zależeć od czasu wykonania Twojego programu. Niech L oznacza maksymalny czas wykonania ze wszystkich przypadków testowych w tym podzadaniu. Otrzymasz:

- 32 punkty, jeśli L ≤ 200 000;
- $32 - 32 \log_{10}(L / 200\,000)$ punktów, jeśli $200\,000 < L < 2\,000\,000$;

- 0 punktów, jeśli $L \geq 2\,000\,000$.

Ograniczenia: długość programu ≤ 200.

Podzadanie 5. [co najwyżej 28 punktów]

Każde pole planszy może zawierać dowolną liczbę kamyków (rzecz jasna, między 0 a 15). Napisz program, który wyznaczy minimum, tj. zakończy działanie odometru w polu (i, j), takim że każde z pozostałych pól zawiera co najmniej tyle kamyków, co pole (i, j). Po zakończeniu działania programu każde pole musi zawierać dokładnie tyle samo kamyków, ile zawierało przed wykonaniem programu.

Liczba punktów, jaką zdobędziesz za to podzadanie, będzie zależeć od długości P Twojego programu. Otrzymasz:

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

Ograniczenia: czas wykonania ≤ 44 400 000.

Szczegóły implementacyjne

Jako rozwiązanie każdego podzadania powinieneś wysłać dokładnie jeden plik, zawierający program zapisany zgodnie ze składnią języka sterowania odometru. Rozmiary poszczególnych plików nie mogą przekraczać 5 MiB. Dla każdego podzadania, Twój program zostanie sprawdzony za pomocą kilku przypadków testowych i otrzymasz pewną informację o tym, ile zasobów zużył. Jeśli składnia Twojego programu będzie niepoprawna, otrzymasz informację o tym, jaki błąd składni w nim występuje.

Nie musisz w każdym swoim zgłoszeniu dzielnie wyklikiwać programów dla wszystkich podzadań. Jeśli Twoje obecne zgłoszenie nie zawiera programu dla podzadania X, w sposób automatyczny zostanie dołączone do tego zgłoszenia Twoje najświeższe zgłoszenie do tego podzadania. Jeśli do podzadania X nie wysyłałeś jeszcze żadnych zgłoszeń, Twoje zgłoszenie nie otrzyma żadnych punktów za to podzadanie.

Jak zwykle, wynikiem zgłoszenia jest suma punktów uzyskanych za poszczególne podzadania, a Twoim ostatecznym wynikiem będzie maksimum z wyników zgłoszeń, na których dokonasz odsłonięcia pełnej punktacji, oraz z wyniku ostatniego zgłoszenia.

Symulator

Masz do dyspozycji symulator odometru, który potrafi uruchomić program sterujący odometrem na konkretnych planszach. Symulator wykonuje programy o składni takiej, jak opisana powyżej.

Plik z opisem planszy powinien mieć następujący format: każdy wiersz zawiera trzy liczby R, C oraz P, oznaczające, że pole (R, C) zawiera P kamyków. Pola niewystępujące w opisie planszy nie zawierają żadnych kamyków. Dla przykładu, rozważmy plik o następującej zawartości:

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

Plansza opisana przez ten plik zawiera 15 kamyków: 3 kamyki na polu (0, 10) i 12 kamyków na polu (4, 5).

Symulator uruchamia się przez wywołanie programu `simulator.py` znajdującego się w katalogu tego zadania. Jako argument należy podać nazwę pliku z programem sterującym odometrem. Symulator przyjmuje następujące parametry wiersza poleceń:

* `-h` symulator podaje krótki opis dostępnych parametrów wiersza poleceń;

* `-g GRID_FILE` ładuje opis planszy z pliku `GRID_FILE` (domyślnie plansza jest pusta);

* `-s GRID_SIDE` ustawia rozmiar planszy na `GRID_SIDE x GRID_SIDE` (domyślna wartość to 256, taka sama jak występująca wszędzie w treści zadania); używanie mniejszych plansz może Ci pomóc w usuwaniu usterek z programu;

* `-m STEPS` ustawia górny limit na czas wykonania programu na `STEPS` kroków;

* `-c` uruchamia tryb kompilacji. W trybie kompilacji symulator generuje i kompiluje program w języku C, który będzie przeprowadzał symulację. Wprawdzie przygotowanie programu zajmuje pewien czas, jednak gotowy program wykonuje symulację istotnie szybciej niż `simulator.py`. Polecamy Ci używać tej opcji, gdy Twój program wykonuje 10 000 000 lub więcej kroków.

Liczba zgłoszeń

Maksymalna liczba zgłoszeń w tym zadaniu to 128.