Dostępna pamięć: 64MB

Poszukiwanie skarbu

Dana jest kostka sześcienna $n \times n \times n$ ($3 \leqslant n \leqslant 300$). Powierzchnia każdej ze ścian składa się z $n \times n$ kwadratowych płytek (w sumie takich płytek na powierzchni kostki jest $6n^2$). Pod jedną z płytek ukryty jest skarb. Naszym zadaniem jest go znaleźć. Na powierzchni kostki znajduje się robot, którego kontrolujemy. Niestety nie znamy jego położenia ani kierunku, w którym patrzy. Wiemy za to, że robot stoi na środku pewnej płytki oraz patrzy w stronę środka jednej z jej czterech boków. Możemy sterować robotem wydając mu następujące polecenia:

1. `naprzod` – robot przemieszcza się na sąsiednią płytkę w kierunku, w którym patrzy. Jeśli patrzy w kierunku krawędzi kostki, w naturalny sposób przejeżdża przez tę krawędź i obraca się tak, by stać na sąsiedniej płytce na sąsiedniej ścianie kostki. Po wykonaniu ruchu robot patrzy w przeciwną stronę niż krawędź, przez którą przejechał.
2. `prawo` – robot obraca się o 90° w prawo, pozostając w miejscu.
3. `lewo` – robot obraca się o 90° w lewo, pozostając w miejscu.
4. `szukaj` – robot rozkopuje płytkę, na której się znajduje, próbując znaleźć zakopany pod nią skarb.

Warto wiedzieć, że kopanie skarbu jest bardzo energochłonne i robotowi wystarczy energii na tylko jedną taką akcję. W związku z tym, znalezienie skarbu byłoby mało prawdopodobne, gdyby nie wykrywacz metalu, który posiada robot. Jako że skarb zawiera duże ilości metalu, robot jest w stanie użyć wykrywacza by oszacować odległość od skarbu. Niestety wykrywacz jest bardzo ułomny i jest w stanie jedynie stwierdzić, czy skarb znajduje się bliżej czy dalej, niż ruch wcześniej. Zatem przy każdej komendzie `naprzod`, robot wysyła dwa sygnały wykrywaczem metalu (przed i po wykonaniu ruchu), po czym porównuje ich wyniki i stwierdza czy skarb po wykonaniu ruchu znajduje się bliżej, dalej czy w tej samej odległości od robota. Wykrywacz metalu liczy odległość w **metryce euklidesowej** od środka płytki ze skarbem do środka płytki z robotem. Twoim zadaniem jest napisać program, który będzie sterował robotem i pomoże mu znaleźć skarb. A, byłbym zapomniał: wspomniałem o ograniczonej baterii robota? Zwykłe operacje (`naprzod`, `prawo`, `lewo`) też ją zużywają, tylko że wolniej od kopania. Robotowi wystarczy baterii na $20n$ ruchów... nooo, może $100n$ ruchów jak będzie jeździł oszczędnie. Upewnij się, że odnajdzie i wykopie skarb, zanim skończy się bateria. Powodzenia!

Biblioteka

Jest to zadanie interaktywne, to znaczy Twój program będzie porozumiewał się z biblioteką. Aby użyć biblioteki, należy załączyć nagłówek `#include "pos.h"`. Biblioteka udostępnia następujące funkcje:

- `void init()` – funkcja ta powinna zostać wywoływana tylko raz, na początku działania programu. Włącza ona robota i bez jej wywołania nie będzie on reagował na inne polecenia.
- `char naprzod()` – funkcja ta wysyła robotowi polecenie przemieszczenia się naprzód. Zwraca `'b'`, `'d'` lub `'r'` w zależności od tego czy po wykonaniu ruchu robot znajduje się bliżej, dalej czy tak samo daleko od skarbu.
- `void prawo()` – funkcja ta wysyła robotowi polecenie obrotu o 90° w prawo.
- `void lewo()` – funkcja ta wysyła robotowi polecenie obrotu o 90° w lewo.
- `void szukaj()` – funkcja ta wysyła robotowi polecenie szukania skarbu. Powinna być ona wywołana tylko raz, a po jej wykonaniu program powinien się zakończyć (w końcu jaki jest cel wysyłania poleceń rozładowanemu robotowi?).

Kompilacja na swoim komputerze

Pliki z archiwum `pos_dla_zaw.zip` dostepnego w zakładce "Pliki" należy wypakować do folderu z kodem źródłowym programu.

Aby program się skompilował należy załączyć nagłówek `#include "pos.h"`.

Program należy skompilować razem z biblioteką `pos_lib.cc`. Można to zrobić za pomocą polecnia: `g++ twoj_program.cpp pos_lib.cc -o twoj_program`.

Wyjście

Twój program nie powinien pisać na standardowe wyjście (`stdout`) ani czytać ze standardowego wejścia (`stdin`). Dozwolone jest pisanie na standardowe wyjście diagnostyczne (`stderr`), lecz pamiętaj, że zabiera to cenny czas.

Przykład

Funkcja	Wynik	Opis
`init()`	-	Robot zostaje uruchomiony. Sytuacja odbywa się na kostce $3 \times 3 \times 3$, skarb jest zakopany pośrodku górnej ścianki, a robot stoi na płytce obok i patrzy w przeciwnym kierunku. Zarówno robot jak i program nie są tego świadome.
`naprzod()`	`'d'`	Robot przemieszcza się naprzód i dowiaduje się, że oddalił się od skarbu. Robot znajduje się teraz na bocznej ścianie kostki.
`prawo()`	-	Robot obraca się o 90° w prawo.
`prawo()`	-	Robot obraca się o 90° w prawo.
`naprzod()`	`'b'`	Robot przemieszcza się naprzód i dowiaduje się, że zbliżył się do skarbu. Robot znajduje się z powrotem na górnej ścianie kostki.
`naprzod()`	`'b'`	Robot przemieszcza się naprzód i dowiaduje się, że zbliżył się do skarbu.
`szukaj()`	-	Robot kopie w poszukiwaniu skarbu. Miejmy nadzieję, że mu się udało.

Ocenianie

Żeby program dostał jakiekolwiek punkty, musi on działać zgodnie z wymaganiami, czyli jako pierwszą wywołać funkcję `init`, jako ostatnią `szukaj` i każdą z nich dokładnie raz. Musi on również wywołać funkcję `szukaj` stojąc na płytce ze skarbem. Jeśli powyższe warunki zostaną spełnione, to program oceniany jest w następujący sposób. Niech p oznacza liczbę punktów za dany test, a k – liczbę komend `naprzod`/`prawo`/`lewo` wysłanych robotowi.

1. Jeśli $k \leqslant 20n$, program dostanie p punktów.
2. Jeśli $k > 100n$, program dostanie 0 punktów.
3. Jeśli żaden z dwóch powyższych nie zachodzi, to program dostanie $\lfloor p \cdot 20n/k \rfloor$ punktów. Oznacza to, że jeśli program spełni warunki i zmieści się w limicie ruchów, otrzyma on co najmniej 20% możliwych punktów.

Podzadanie	Ograniczenia	Limity czasowe	Punkty
1	$n \leqslant 20$	0.1 s	50
2	brak dodatkowych ograniczeń	0.5 s	50