Tomasz Idziaszek
Treść zadania, Opracowanie

Maciej Borsz
Program

Dostępna pamięć: 128 MB. *OI, etap III, dzień pierwszy, 13.03.2013*

Labirynt

Bajtazar przeczytał niedawno ciekawą historię. Jej bohaterem był jakiś grecki królewicz, który pokonał straszliwego potwora za pomocą kłębka wełny, lub coś w tym rodzaju. Ale to nie to tak zafascynowało Bajtazara. Najbardziej spodobało mu się, że kluczowe wydarzenia działy się w labiryncie. Od tej pory Bajtazar ma bzika na punkcie labiryntów.

Bajtazar rysuje plany labiryntów na kratkowanej kartce papieru. Każdy taki plan jest wielokątem, którego boki (reprezentujące ściany labiryntu) są równoległe do brzegów kartki (tj. osi prostokątnego układu współrzędnych) i każde dwa kolejne boki są do siebie prostopadłe. Bajtazar zauważył, że jeśli na jednej ze ścian takiego labiryntu umieścimy wejście, a następnie wejdziemy do niego i, idąc, cały czas będziemy trzymać się prawą ręką ściany, to na pewno obejdziemy cały labirynt, wracając na końcu z powrotem do wejścia.

Co więcej, podczas takiego obejścia możemy notować wykonywane przez nas zakręty. Zapisujemy literę `L`*, jeśli podczas przechodzenia na kolejną ścianę obracamy się w lewo, zaś* `P`*, jeśli obracamy się w prawo. Bajtazar zastanawia się, dla jakich słów złożonych z liter* `L` *i* `P` *istnieje labirynt, który spowoduje, że zanotujemy takie słowo podczas obchodzenia tego labiryntu.*

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedno n-literowe słowo ($1 \leqslant n \leqslant 100\,000$) złożone z liter `L` *i* `P`*, które opisuje ciąg kolejnych zakrętów napotykanych podczas obchodzenia labiryntu.*

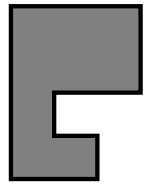
W testach wartych 50% punktów zachodzi dodatkowy warunek $n \leqslant 2500$.

Wyjście

Jeśli nie da się skonstruować labiryntu według opisu z wejścia, na standardowym wyjściu należy wypisać słowo `NIE`*. W przeciwnym wypadku na wyjście należy wypisać dokładnie n wierszy zawierających opis przykładowego labiryntu. W i-tym z nich powinny znaleźć się dwie liczby całkowite x_i i y_i ($-10^9 \leqslant x_i, y_i \leqslant 10^9$) oddzielone pojedynczym odstępem, oznaczające współrzędne i-tego wierzchołka na planie labiryntu. Wierzchołki powinny zostać wypisane zgodnie z kolejnością ich występowania na obwodzie wielokąta, przeciwnie do ruchu wskazówek zegara; można zacząć od dowolnego wierzchołka i nie trzeba zaznaczać umiejscowienia wejścia.*

Przykład

Dla danych wejściowych:
`LLLLPPLL`
poprawnym wynikiem jest:
```
0 0
2 0
2 2
-1 2
-1 -2
1 -2
1 -1
0 -1
```



Testy „ocen":

`1ocen`**:** *$n = 12$, słowo* `LLLLLLLLLLLL`*, odpowiedź* `NIE`*;*

`2ocen`**:** *$n = 100$, spirala zakręcająca w lewo;*

`3ocen`**:** *$n = 100\,000$, schodki.*

Wizualizator wyjść

Do tego zadania dołączony jest wizualizator wyjść, który dla pliku zgodnego z formatem wyjścia rysuje odpowiadający mu labirynt. Aby go uruchomić, wejdź do folderu `/home/zawodnik/labwiz` *i wykonaj polecenie:*

```
./labwiz
```

Dla plików, które nie zawierają opisu labiryntu zgodnego z formatem wyjścia, zachowanie wizualizatora jest nieokreślone.

Rozwiązanie

Dane jest n-literowe słowo składające się tylko z liter `L` i `P`. Zadanie polega na wygenerowaniu takiego labiryntu – wielokąta o bokach prostopadłych do osi układu współrzędnych – że, obchodząc jego obwód (zaczynając od wybranej ściany), wykonamy określoną sekwencję skrętów w lewo (litera `L` w słowie) i w prawo (`P`).

Ponieważ dopuszczalną odpowiedzią jest też stwierdzenie, że żądany labirynt nie istnieje, zastanówmy się najpierw, kiedy taka sytuacja będzie miała miejsce. Niech l oznacza liczbę skrętów w lewo w słowie, a $p = n - l$ liczbę skrętów w prawo. Dość łatwo się przekonać, że warunkiem koniecznym na istnienie labiryntu jest to, żeby skrętów w lewo było dokładnie o cztery więcej niż skrętów w prawo, czyli

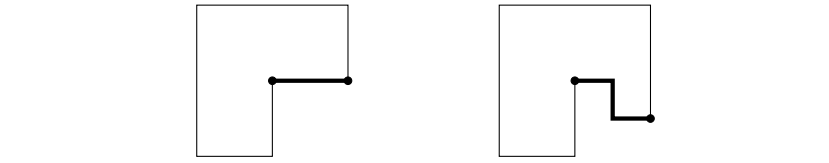
$$l = p + 4. \qquad (\star)$$

Istotnie, aby mieć szansę skończyć wędrówkę w tym samym miejscu, w którym ją zaczęliśmy, musimy (poruszając się przeciwnie do ruchu wskazówek zegara) sumarycznie wykonać obrót o dokładnie $4 \cdot 90^\circ = 360^\circ$ w lewo. (W szczególności oznacza to, że długość słowa opisującego labirynt musi być parzysta.)

Mniej oczywiste jest to, że jest to również warunek wystarczający na istnienie labiryntu. Dowód tego faktu przeprowadzimy przez indukcję po długości słowa opisującego labirynt. Najmniejszy wielokąt ma co najmniej 4 boki. Dla $n = 4$ daje to $l = 4$ i $p = 0$, a rozwiązaniem jest dowolny prostokąt o współrzędnych całkowitoliczbowych.

Załóżmy zatem, że labirynt istnieje dla wszystkich słów długości $n-2$ spełniających warunek $(\star)$, i rozważmy dowolne słowo w długości n spełniające ten warunek. Skoro $n \geqslant 6$, to w słowie w znajduje się co najmniej jedna litera `P` i co najmniej jedna litera `L`. Co więcej, możemy bez straty ogólności założyć, że pierwsze dwie litery słowa w to `P` i `L`, czyli $w = \texttt{PL}w'$ dla pewnego słowa w' o długości $n-2$. Jeżeli tak nie jest, to możemy zastąpić w jego obrotem cyklicznym, który zaczyna się od `PL` (zauważmy, że labirynt będzie taki sam dla każdego obrotu cyklicznego słowa w).

Zauważmy, że słowo w' również spełnia warunek $(\star)$, zatem z założenia indukcyjnego istnieje labirynt opisywany tym słowem. Możemy teraz zmodyfikować ten labirynt, aby dostać labirynt opisywany słowem $\texttt{PL}w'$, poprzez dodanie dwóch zakrętów na ścianie labiryntu zawierającej wejście. Przykładowo, jeśli jest to ściana łącząca wierzchołki (x, y) i $(x + 1, y)$, tak jak na rys. 1, to zastępujemy ją trzema ścianami łączącymi kolejno wierzchołki (x, y), $(x + \frac{1}{2}, y)$, $(x + \frac{1}{2}, y - \frac{1}{2})$, $(x + 1, y - \frac{1}{2})$ i na pierwszej z tych ścian umieszczamy wejście. Następnie mnożymy wszystkie współrzędne wierzchołków przez 2, aby z powrotem stały się całkowite.



Rys. 1: Labirynt dla słowa $w' = \texttt{LLLLLP}$ oraz dla słowa $w = \texttt{PL}w'$.

Rozwiązanie kwadratowe

Powyższy dowód istnienia labiryntu jest konstruktywny i może stać się podstawą do zapisania rekurencyjnego algorytmu wyznaczającego labirynt. Każda z $O(n)$ faz rekurencji będzie znajdować odpowiedni obrót cykliczny słowa w, wyznaczać rekurencyjnie labirynt dla słowa w' i poprawiać go zgodnie z powyższym opisem. Pojedyncza faza rekurencji trwa $O(n)$, zatem cały algorytm będzie działał w czasie $O(n^2)$.

Pozostał jeszcze jeden szczegół techniczny do dopracowania. Mnożąc w każdej fazie współrzędne wierzchołków przez 2, dość szybko wyjdziemy poza dopuszczalny prostokąt $|x|, |y| \leqslant 10^9$, w którym ma się zmieścić labirynt. Aby temu zapobiec, wystarczy na koniec każdej fazy algorytmu *skompresować* labirynt, przenumerowując wszystkie współrzędne występujące w wyniku na pewne liczby z zakresu od 1 do $\frac{n}{2}$. Taki zakres współrzędnych nam wystarczy, gdyż każda z $\frac{n}{2}$ ścian pionowych łączy dwa wierzchołki

o równej współrzędnej x, podobnie dla ścian poziomych i współrzędnych y. Ponadto dowolne przekształcenie „ściskające" labirynt wzdłuż osi układu współrzędnych daje nadal poprawny labirynt. Jeśli chodzi o implementację, to używamy standardowej sztuczki polegającej na zmapowaniu każdej współrzędnej na liczbę współrzędnych mniejszych od niej przed przemapowaniem. Takie przemapowanie wymaga sortowania i zajmuje czas $O(n \log n)$. W przypadku naszego zadania, przemapowanie będziemy musieli robić po każdej fazie, będzie ono jednak kosztować $O(n)$, gdyż możemy zastosować sortowanie przez zliczanie.

Rozwiązanie wzorcowe

Słowo złożone z takiej samej liczby liter `L` i `P`, w którego każdym prefiksie jest co najmniej tyle samo liter `L`, co liter `P`, nazwiemy *zbalansowanym*. Rozwiązanie wzorcowe działa w oparciu o obserwację, że ze słowa opisującego labirynt możemy usunąć takie cztery skręty w lewo, aby pozostałe cztery fragmenty były słowami zbalansowanymi (przyjmujemy przy tym, że pierwsza litera słowa sąsiaduje z ostatnią). Na przykład ze słowa `PLLPLLLPLPPLLLLP` można usunąć podkreślone litery `L`, aby pozostałe fragmenty `LP`, `LLPLPP`, ϵ (słowo puste) i `LLPP` były zbalansowane. Każdy z tych fragmentów będzie rozpatrywany osobno, a wynikowy labirynt powstanie przez sklejenie labiryntów odpowiadających tym fragmentom. (Formalnie, zbalansowanemu słowu odpowiada pewna łamana niebędąca wielokątem, ale dla prostoty opisu będziemy ją również nazywać labiryntem.)

Labirynt dla zbalansowanego słowa będziemy tworzyć rekurencyjnie. Słowu pustemu (brak skrętów) odpowiada jedna ściana. Jeśli słowo w jest niepuste, to rozważymy dwa przypadki:

(1) Pewien właściwy prefiks w też jest zbalansowany, tzn. $w = w_1 w_2$ dla niepustych słów w_1 i w_2, takich że w_1 jest zbalansowane. Wówczas w_2 również jest zbalansowane i labirynt dla w powstaje z połączenia labiryntów dla w_1 i w_2.

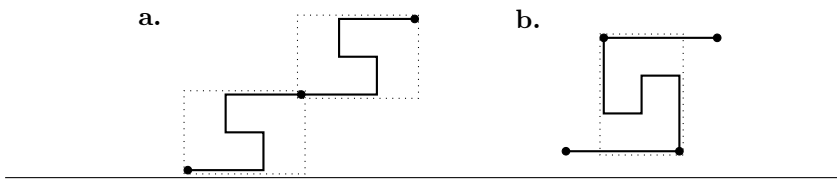
(2) W przeciwnym wypadku $w = \texttt{L}w'\texttt{P}$ i słowo w' jest zbalansowane. Labirynt dla w powstaje z obrotu labiryntu dla w' w lewo i dołożeniu dwóch ścian.

A konkretniej: zakładamy, że konstruowany przez nas labirynt dla zbalansowanego słowa długości n ma mieć $n+1$ ścian, ma rozpoczynać się w wierzchołku $(0, 0)$, kończyć w wierzchołku (x, y) i ma mieścić się w prostokącie o rozmiarze $x \times y$, tak że poza końcami żadne inne punkty labiryntu nie dotykają prawej i lewej krawędzi prostokąta. Wtedy opisane powyżej przypadki rozpatrujemy następująco (patrz rys. 2):

(0) Labirynt dla słowa pustego składa się ze ściany łączącej punkty $(0, 0)$ i $(1, 0)$.

(1) Jeśli labirynt dla zbalansowanego słowa w_i mieści się w prostokącie rozmiaru $x_i \times y_i$, to labirynt dla słowa $w = w_1 w_2$ powstaje przez przesunięcie labiryntu dla w_2 do punktu (x_1, y_1) i mieści się w prostokącie rozmiaru $(x_1 + x_2) \times (y_1 + y_2)$.

(2) Jeśli labirynt dla zbalansowanego słowa w' mieści się w prostokącie rozmiaru $x \times y$, to labirynt dla słowa $w = \texttt{L}w'\texttt{P}$ powstaje przez obrót tego labiryntu, by mieścił się w prostokącie wyznaczonym przez punkty $(y + 1, 0)$ i $(1, x)$, oraz



Rys. 2: Konstrukcja labiryntu **a.** dla słowa ww i **b.** słowa LwP, gdzie w = LLPP.

dodanie ścian łączących punkty $(0,0)$ i $(y+1,0)$ oraz punkty $(1,x)$ i $(y+2,x)$. Cały labirynt mieści się więc w prostokącie rozmiaru $(y+2) \times x$.

Poniższe dwie funkcje wzajemnie rekurencyjne pozwalają na wyznaczenie rozmiarów prostokąta otaczającego labirynt dla niepustego słowa zbalansowanego w. Zakładamy, że i jest globalną zmienną oznaczającą aktualnie podglądaną literę w słowie w, początkowo zainicjowaną na 1. Funkcja *prefiksy*, dopóki może, odcina zbalansowane prefiksy. Ponieważ są one najkrótsze możliwe, więc funkcja *rek* zakłada, że ma do czynienia z przypadkiem (2). Przeskakuje ona końcowe litery fragmentu, a środek rozdziela na zbalansowane fragmenty znów z wykorzystaniem funkcji *prefiksy*.

```
1:  function prefiksy()
2:  begin
3:    if w_i = P then return (1,0);
4:    x := 0; y := 0;
5:    while i < n and w_i = L do begin
6:      (x',y') := rek();
7:      x := x + x'; y := y + y';
8:    end
9:    return (x,y);
10: end
```

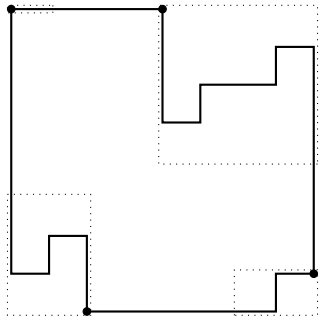
```
1: function rek()
2: begin
3:   { w_i = L }
4:   i := i + 1;
5:   (x,y) := prefiksy();
6:   { w_i = P }
7:   i := i + 1;
8:   return (y+2,x);
9: end
```

Jeśli spojrzymy na zbiór zbalansowanych fragmentów słowa w jak na drzewo (iterujemy po kolejnych literach i interpretujemy literę `L` jako początek opisu nowego poddrzewa, a literę `P` jako jego koniec – analogicznie do interpretacji nawiasowań jako drzew), to powyższe funkcje obliczają dla każdego poddrzewa prostokąt, jaki jest potrzebny do jego narysowania.

W następnym kroku możemy (korzystając z obliczonych rozmiarów prostokątów) rekurencyjnie wyznaczyć ciągi ścian odpowiadające zbalansowanym fragmentom słowa w.

Zauważmy, że nie potrzebujemy kompresować uzyskanych labiryntów; łatwo wykazać, że labirynt dla n-literowego słowa będzie miał współrzędne wierzchołków nie większe niż n. Na końcu łączymy cztery labirynty w jeden. Wystarczy umieścić prostokąty ograniczające te labirynty w wierzchołkach dużego kwadratu i odpowiednio przedłużyć ich początkowe ściany (patrz rys. 3).

Powyższe rozwiązanie działa w optymalnym czasie $O(n)$ i zostało zaimplementowane w plikach `lab.cpp` i `lab3.pas`.



Rys. 3: Łączenie czterech labiryntów dla słowa PLLPLLLPLPPLLLLP.

Aby uzasadnić poprawność rozwiązania, pozostaje tylko udowodnić fakt, że dla słowa spełniającego warunek ($\star$) zawsze istnieje podział zbalansowany. Pozostawimy to jako ćwiczenie dla Czytelnika, gdyż jego dowód jest bardzo podobny do dowodu znanego faktu, który mówi, że dla każdego słowa zawierającego taką samą liczbę liter `L` i `P` istnieje obrót cykliczny będący słowem zbalansowanym.