Marian M. Kędzierski
Treść zadania, Opracowanie

Michał Pilipczuk
Program

OI, Etap I, 22.10–19.11.2007

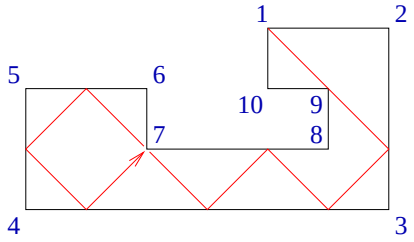
Szklana pułapka

W pałacu króla Bajtazara straszy duch. Złośliwi twierdzą, że jest to duch niedawno zmarłej (w podejrzanych okolicznościach) małżonki Bajtazara, gdyż tak jak ona, bardzo lubi przeglądać się w lustrach. Nie dziwi więc, że Bajtazar chętnie pozbyłby się ducha.

Bajtazar postanowił w jednej z komnat pałacu przygotować specjalną **szklaną pułapkę**. *Jest to zamknięta, dobrze oświetlona sala, której wszystkie ściany wewnętrzne pokryte są lustrami. Ponadto w każdym rogu sali może być umieszczony laser albo czujnik promieni laserowych. Gdy tylko duch przetnie wiązkę jednego z laserów, podniesie się alarm, a sprowadzeni przez Bajtazara pogromcy duchów rozprawią się z nim.*

Sala, w której powstaje szklana pułapka, ma kształt wielokąta o sąsiednich bokach prostopadłych do siebie. Ponadto długość każdej ściany (wyrażona w metrach) jest całkowita. Jeżeli w danym rogu sali ma być umieszczony laser, to jego promień musi być skierowany wzdłuż dwusiecznej kąta, jaki tworzą stykające się w tym rogu ściany (i zarazem równolegle do podłogi). Oczywiście, jeżeli promień lasera napotka na swojej drodze lustro, to odbija się od niego pod takim samym kątem, pod jakim pada, czyli 45 stopni. Promień wpadający wzdłuż dwusiecznej kąta w róg sali z czujnikiem zostaje całkowicie pochłonięty przez czujnik. Promień wpadający wzdłuż dwusiecznej kąta w róg sali bez czujnika (za to być może z innym laserem) jest odbijany o 180 stopni. W innych przypadkach promień kontynuuje wędrówkę po sali, nie zmieniając kierunku.

W sali należy rozmieścić maksymalnie dużo laserów i odpowiadających im czujników, w taki sposób, żeby promień każdego lasera wpadał do jakiegoś czujnika, promienie różnych laserów wpadały do różnych czujników oraz żeby do każdego czujnika wpadał promień pewnego lasera. Przy tym w każdym rogu można zainstalować tylko albo laser, albo czujnik, ale nie jedno i drugie.



Przykład szklanej pułapki z promieniem lasera biegnącym z rogu 1 do rogu 7.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opis kształtu szklanej pułapki,*
- *wyznaczy sposób rozmieszczenia maksymalnej liczby laserów i czujników spełniający warunki zadania,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba całkowita n ($4 \leqslant n \leqslant 100\,000$). Jest to liczba ścian szklanej pułapki. W każdym z następnych n wierszy znajdują się po dwie liczby całkowite x_i i y_i oznaczające współrzędne rogu nr i ($1 \leqslant i \leqslant n$, $-1\,000\,000 \leqslant x_i, y_i \leqslant 1\,000\,000$), przedzielone pojedynczą spacją. Każde dwa kolejne rogi połączone są ścianą równoległą do jednej z osi współrzędnych. Żadne dwie ściany nie przecinają się i nie mają punktów wspólnych, z wyjątkiem dwóch kolejnych ścian, które stykają się we wspólnym rogu. Kolejne rogi pomieszczenia podane są w kolejności zgodnej z ruchem wskazówek zegara (tzn. przy obchodzeniu pomieszczenia wzdłuż ścian wnętrze znajduje się zawsze po prawej stronie). Łączna długość wszystkich ścian nie przekracza 300 000.

Wyjście

Twój program powinien w pierwszym wierszu wyjścia wypisać jedną liczbę całkowitą m, oznaczającą maksymalną liczbę par laser-czujnik, jakie można umieścić w szklanej pułapce. W każdym z następnych m wierszy powinna znaleźć się para liczb a_i i b_i oddzielonych pojedynczą spacją, gdzie a_i oznacza numer rogu, w którym powinien znaleźć się laser, zaś b_i — numer rogu, w którym powinien znaleźć się odpowiadający mu czujnik, przy czym $1 \leqslant a_i, b_i \leqslant n$. W przypadku, gdy istnieje wiele rozwiązań, należy podać dowolne z nich.

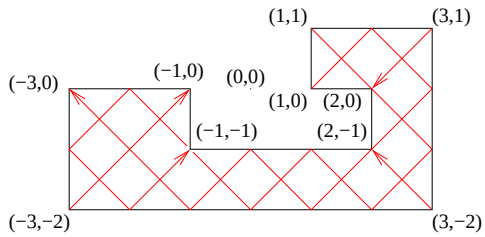
Przykład

Dla danych wejściowych:

```
10
1 1
3 1
3 -2
-3 -2
-3 0
-1 0
-1 -1
2 -1
2 0
1 0
```

poprawnym wynikiem jest:

```
5
10 5
1 7
2 9
3 8
4 6
```



Przykładowe rozmieszczenie systemu laserów i czujników w szklanej pułapce przedstawia rysunek.

Rozwiązanie

Wprowadzenie

Na pierwszym etapie LV Olimpiady Matematycznej pojawiło się następujące zadanie:

> Dany jest wielokąt o bokach długości wymiernej, w którym wszystkie kąty wewnętrzne są równe 90° lub 270°. Z ustalonego wierzchołka wypuszczamy promień świetlny do wnętrza wielokąta w kierunku dwusiecznej kąta wewnętrznego przy tym wierzchołku. Promień odbija się zgodnie z zasadą: kąt padania jest równy kątowi odbicia. Udowodnić, że promień trafi w jeden z wierzchołków wielokąta.

W dowodzie tego twierdzenia okazywało się ponadto, że wierzchołek, do którego trafiał promień świetlny, był różny od tego, z którego promień został wypuszczony. Nie jest to jednak jedyny ciekawy wniosek, jaki z tego dowodu można było wysnuć.

Jeśli przyjrzymy się treści powyższego zadania z Olimpiady Matematycznej, to zauważymy zapewne łudzące podobieństwo do naszego problemu ze szklaną pułapką. Nie bez powodu...

Okazuje się, że jeśli przekształcimy wielokąt z tego zadania przez jednokładność o skali równej wspólnemu mianownikowi wszystkich ułamków określających długości poszczególnych boków, to otrzymamy wielokąt podobny do oryginalnego, w którym długości wszystkich boków będą liczbami naturalnymi. Następnie, korzystając z faktu, że wszystkie kąty wewnętrzne wielokąta są wielokrotnościami 90°, możemy obrócić go tak, aby jego boki stały się równoległe do osi układu współrzędnych. Wystarczy jeszcze tylko zastosować przesunięcie (uczenie mówiąc: *translację*), takie aby pewien wierzchołek wielokąta stał się punktem kratowym (tzn. aby jego obie współrzędne stały się liczbami całkowitymi), i wtedy okaże się, że wszystkie wierzchołki wielokąta będą punktami kratowymi. Okazuje się, że tak przekształcony wielokąt jest dokładnie tej samej postaci, co szklana pułapka występująca w naszym zadaniu!

Ponieważ wykonane przekształcenia nie zmieniają kształtu figury (ściślej mówiąc, powstały wielokąt jest *podobny* do pierwotnego), to teza zadania z Olimpiady Matematycznej zachodzi również dla tak przekształconego wielokąta. W szczególności jest ona także prawdziwa dla szklanej pułapki.

Wiemy więc już, że jeżeli wystrzelimy promień lasera z wierzchołka a (wzdłuż dwusiecznej kąta), to trafi on w pewien wierzchołek b. Co więcej, będzie to wierzchołek różny od a. Łatwo też zauważyć, że promień biegnie w sposób symetryczny, tzn. wystrzelony z wierzchołka b trafi z powrotem do wierzchołka a.

To spostrzeżenie pozwala zauważyć, że wierzchołki dowolnego wielokąta można połączyć w jednoznaczny sposób w pary (a,b) takie, że promień lasera wystrzelony z wierzchołka a trafia w wierzchołek b, zaś wystrzelony z b trafia w a. To z kolei prowadzi do wniosku, że jeżeli dla każdej takiej pary umieścimy laser w jednym wierzchołku, a czujnik w drugim, to otrzymamy $\frac{n}{2}$ par laser-czujnik spełniających warunki zadania. Z drugiej strony w każdym rogu pomieszczenia może znaleźć się tylko jedno urządzenie, więc rozmieszczenie większej liczby par niż $\frac{n}{2}$ jest niemożliwe.

Tak więc jesteśmy już w stanie udzielić odpowiedzi na pierwszą część zadania i to jedynie na podstawie liczby rogów pomieszczenia — można zawsze rozmieścić dokładnie $\frac{n}{2}$ par urządzeń[1].

Wszystkie powyższe wnioski można było łatwo wyciągnąć na podstawie dowodu tezy przedstawionej w zadaniu z Olimpiady Matematycznej. Zadanie to nasuwa jednak na myśl kolejny problem. Skoro wiemy, że dla dowolnego wielokąta o wierzchołkach kratowych i bokach równoległych do osi układu współrzędnych istnieje jednoznaczny rozkład zbioru wierzchołków na pary „widzące się wzajemnie", to dlaczego nie pokusić się o wyznaczenie tego rozkładu dla zadanego wielokąta?

I tak oto wkraczamy w drugą, ciekawszą część zadania...

Rozważania matematyczne

Lepsze zrozumienie dowodu twierdzenia z Olimpiady Matematycznej powinno pomóc podzielić wierzchołki wielokąta na żądane pary. Przyjrzyjmy się więc dokładniej temu rozumowaniu.

Lemat 1. Wielokąt o bokach długości wymiernej, w którym wszystkie kąty wewnętrzne są równe 90° lub 270°, jest podobny do pewnego wielokąta o wierzchołkach kratowych i bokach równoległych do osi układu współrzędnych.

Dowód: Ciąg przekształceń geometrycznych opisanych w rozdziale „Wprowadzenie" pokazuje, jak dokonać przeprowadzenia pomiędzy takimi wielokątami, zachowując podobieństwo. ■

W dalszych rozważaniach będziemy już odnosić się do przekształconego wielokąta, który odpowiada założeniom o szklanej pułapce. Ponieważ wciąż mamy na myśli zadanie z Olimpiady Matematycznej, to w tym rozdziale będziemy zakładać, że promień, który wpada do wierzchołka, zawsze zostaje przez niego pochłonięty.

Definicja 1. Wszystkie sytuacje powstające w chwili zetknięcia promienia z obwodem wielokąta podzielimy na *zdarzenia* i *otarcia*. *Zdarzeniem* będziemy nazywali jedną z następujących sytuacji:

1. *wystrzelenie* promienia z wierzchołka (zgodnie z zasadami),
2. *odbicie* promienia od obwodu wielokąta,
3. *wpadnięcie* promienia do wierzchołka.

Otarciem będziemy nazywali sytuację, w której promień dotyka obwodu wielokąta, jednak nie zmienia to jego biegu.

Zauważmy, że terminy *zdarzenie* i *otarcie* nigdy nie opisują tej samej sytuacji i termin *zdarzenie* wyczerpuje wszystkie sytuacje, w których promień zmienia swój bieg.

Definicja 2. *Krokiem* biegu promienia będziemy nazywali odcinek jego toru zawarty pomiędzy dwoma zdarzeniami.

[1] Ciekawostka: przy okazji całe to rozumowanie uzasadnia, że liczba wierzchołków wyjściowego wielokąta musi być parzysta!

Lemat 2. Kierunek biegu promienia jest zawsze równoległy do jednej z prostych: $y = x$ lub $y = -x$.

Dowód: Dowiedziemy tego przez indukcję ze względu na kolejne kroki biegu promienia.

Baza indukcji. Dwusieczna kąta w każdym wierzchołku wielokąta jest zawsze równoległa do jednej z podanych prostych, więc promień wystrzelony z wierzchołka również.

Krok indukcyjny. Załóżmy, że promień spełnia tezę lematu i jest równoległy do prostej $y = x$. Po kolejnym zdarzeniu:

- jeżeli jest to odbicie, to kierunek po nim zmienia się na prostopadły do dotychczasowego, czyli będzie równoległy do prostej $y = -x$;
- jeżeli zaś jest to wpadnięcie, to po nim nie następuje żaden krok biegu promienia.

W przypadku, gdy przed zdarzeniem promień biegnie równolegle do prostej $y = -x$, dowód jest analogiczny.

Na mocy Twierdzenia o Indukcji Matematycznej, teza lematu jest więc prawdziwa w dowolnym kroku biegu promienia. ■

Oznaczmy przez B — brzeg wielokąta, przez W — jego wnętrze wraz z brzegiem, zaś przez $\mathbb{Z}$ — zbiór liczb całkowitych.

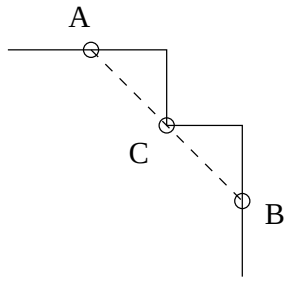
Z Lematu 2 wynika, że wszystkie możliwe trasy promieni są całkowicie zawarte w zbiorze:

$$X = W \cap \left(\bigcup_{a \in \mathbb{Z}} \{(x,y) : y = a - x\} \cup \bigcup_{a \in \mathbb{Z}} \{(x,y) : y = a + x\} \right)$$

Zauważmy, że $X \cap B$ składa się wyłącznie z punktów kratowych. Ponadto, zbiór X jest sumą odcinków, których końce są właśnie punktami kratowymi obwodu B.

Te spostrzeżenia przywodzą na myśl pewien pomysł, który jest istotą niniejszych rozważań i — jak się okaże później — także istotą rozwiązania całego problemu szklanej pułapki. Zbiór X można przedstawić jako graf, w którym wierzchołkami są punkty kratowe na obwodzie B rozpatrywanego wielokąta, zaś krawędzie łączą końce odcinków ze zbioru X.

Przy takiej konstrukcji grafu powstaje jednak pewien problem: nie jest jednoznacznie powiedziane, jak traktować otarcia. Na rys. 1 widać, że nie jest oczywiste, czy punkty A i B należy połączyć krawędzią, ignorując tym samym, że przechodzi ona przez punkt C, czy utworzyć dwie krawędzie: pomiędzy wierzchołkami A i C oraz B i C? Chociaż na tym etapie oba rozwiązania są równoważne, to lepiej jest przyjąć pierwsze z nich, gdyż uprości to dalsze rozważania.



Rys.1: Otarcie promienia o brzeg wielokąta.

Skonstruujmy więc graf, ignorując wszystkie otarcia. Nazwijmy go G. Łatwo zauważyć, że każdy wierzchołek odpowiadający narożnikowi wielokąta ma w grafie G stopień 1 (tzn. wychodzi z niego jedna krawędź). Natomiast wierzchołek odpowiadający dowolnemu innemu punktowi kratowemu na obwodzie ma w G stopień 2 (wychodzą z niego dwie krawędzie).

Okazuje się, że grafy o powyższej własności mają bardzo prostą postać.

Lemat 3. Graf, w którym każdy wierzchołek ma stopień 1 lub 2, jest zbiorem rozłącznych wierzchołkowo ścieżek i cykli.

Dowód: Udowodnimy ten fakt przez indukcję ze względu na liczbę wierzchołków grafu.

Baza indukcji. Najmniejszy graf spełniający założenia lematu to graf dwuwierzchołkowy mający jedną krawędź. Jest on oczywiście ścieżką.

Krok indukcyjny. Załóżmy, że teza zachodzi dla wszystkich grafów o $k < n$ wierzchołkach. Rozważmy graf n-wierzchołkowy. Jeśli graf ten nie ma wierzchołka stopnia 1, to wybierzmy dowolną krawędź e i usuńmy ją — w grafie mamy teraz dokładnie dwa wierzchołki stopnia 1.

Niech v będzie wierzchołkiem stopnia 1. Utwórzmy najdłuższy możliwy ciąg (v_i) różnych wierzchołków, takich że $v_1 = v$ oraz v_i jest połączony krawędzią z v_{i-1} dla $i > 1$. Powstały ciąg jest ścieżką i kończy się wierzchołkiem stopnia 1 innym niż v (dlaczego?) — oznaczmy go u. Po usunięciu z grafu ścieżki $v-u$ otrzymujemy graf o mniejszej liczbie wierzchołków, w którym również każdy wierzchołek ma stopień 1 lub 2, więc z założenia indukcyjnego jest także zbiorem rozłącznych ścieżek i cykli.

Graf wyjściowy zawiera dodatkowo ścieżkę $v-u$ i, ewentualnie, krawędź e. Zauważmy, że jeśli usunęliśmy z wyjściowego grafu krawędź e, to $e = (v,u)$ i zamyka ona ścieżkę $v-u$ w cykl. Stąd w obu przypadkach graf wyjściowy również jest zbiorem rozłącznych ścieżek oraz cykli. Z Twierdzenia o Indukcji Matematycznej teza lematu jest więc prawdziwa. ■

Teraz pozostaje już tylko zauważyć, że promień wypuszczony z dowolnego wierzchołka wielokąta biegnie dokładnie wzdłuż ścieżki grafu rozpoczynającej się w tym wierzchołku. Stąd już bezpośrednio wynika następujące Twierdzenie.

Twierdzenie 1 (Rozwiązanie zadania z LV OM). *Promień wypuszczony z dowolnego narożnika trafia w inny narożnik wielokąta.*

Dowód: Narożnikowi, z którego jest wypuszczany promień, odpowiada w grafie G wierzchołek v stopnia 1. Promień musi podążać po krawędziach grafu G ścieżką rozpoczynającą się w v, która kończy się w u — innym wierzchołku stopnia 1. Wierzchołkowi u także odpowiada narożnik wielokąta (oczywiście inny niż ten, z którego został wypuszczony promień). ■

Tak więc twierdzenie zostało ostatecznie udowodnione. Można je wyprowadzić krócej, jednak ten dowód jest lepszy z punktu widzenia zadania o szklanej pułapce. Jest w nim bowiem zakodowany algorytm pozwalający znaleźć żądane pary wierzchołków…

Wyznaczanie podziału wierzchołków na pary

Jak Czytelnik mógł się już domyślić, zamierzamy jawnie zbudować graf G, a następnie wyszukać w nim wszystkie ścieżki i połączyć w pary ich końce. Pozostaje jeszcze tylko pytanie, jak zbudować graf G?

Popatrzmy na konstrukcję zbioru X z poprzedniego rozdziału. Wynika z niej, że przecinając z wnętrzem wielokąta wszystkie proste równoległe do prostej $y = x$ przechodzące przez punkty kratowe, otrzymamy wszystkie krawędzie grafu G równoległe do tej prostej. Aby wyznaczyć resztę krawędzi, wystarczy tę samą procedurę powtórzyć dla wszystkich prostych równoległych do prostej $y = -x$.

Rozważmy jedną z interesujących nas prostych — równoległą do $y = x$ lub $y = -x$ — i oznaczmy ją przez ℓ. Powinniśmy teraz znaleźć wszystkie punkty, w których prosta ℓ przecina brzeg wielokąta, i pomiędzy nimi poprowadzić krawędzie grafu G. Zanim do tego przystąpimy, uściślijmy, które przecięcia nas interesują.

Definicja 3. Powiemy, że prosta ℓ *przebija* brzeg B wielokąta w punkcie x, jeżeli przechodzi ona przez punkt x i w każdym jego otoczeniu[2] ma punkty wspólne z wnętrzem oraz z zewnętrzem wielokąta.

Prawdziwa jest następująca, ciekawa własność.

Lemat 4. Dowolna prosta przebija brzeg wielokąta parzystą liczbę razy.

Dowód: Niech $V = \mathbb{R}^2 \setminus W$. Podział płaszczyzny $\mathbb{R}^2$ na zbiory W i V odpowiada jej podziałowi na wnętrze wielokąta i pozostały obszar. Brzeg wielokąta jest granicą pomiędzy zbiorami V i W (przypomnijmy, że brzeg wielokąta zaliczamy do zbioru W).

Obszar W jest ograniczony, więc rozważana prosta odpowiednio daleko zarówno z jednej, jak i z drugiej strony, leży w obszarze V. Prześledźmy jej wędrówkę, zaczynając odpowiednio daleko z jednej, a kończąc odpowiednio daleko z drugiej strony. Każdy punkt, w którym prosta przebija brzeg wielokąta, powoduje jej przejście do innego obszaru: z W do V lub z V do W. Ponieważ zaś „wędrówka" zaczyna i kończy się w obszarze V, to w jej trakcie prosta musiała przebić brzeg parzystą liczbę razy. ■

Załóżmy, że uda nam się wyznaczyć wszystkie punkty przebicia obwodu B przez prostą ℓ. Wówczas wystarczy je posortować w kolejności występowania na prostej ℓ, a następnie połączyć: pierwszy z drugim, trzeci z czwartym itd., tworząc kolejne krawędzie grafu G.

Takiego postępowania nie możemy oczywiście przeprowadzić dla wszystkich możliwych prostych, gdyż jest ich nieskończenie wiele. Ale możemy ograniczyć się tylko do tych prostych, które przecinają wielokąt. Jak je wyznaczyć?

Zauważmy, że proste równoległe do $y = x$ lub $y = -x$ mają odpowiednio równania $x - y = c$ oraz $x + y = d$ dla pewnych stałych c, d. W równaniach „skrajnych" prostych przecinających wielokąt, wartości stałych wynoszą $c_1 = \min_{(x,y)\in B}(x - y)$, $c_2 = \max_{(x,y)\in B}(x-y)$, $d_1 = \min_{(x,y)\in B}(x+y)$, $d_2 = \max_{(x,y)\in B}(x+y)$. Można zauważyć, że są to minimalne i maksymalne współrzędne wierzchołków wielokąta w nieco innym układzie współrzędnych, a mianowicie w układzie $(x-y, x+y)$ [3]. Po przedstawieniu wielokąta w tym układzie współrzędnych, łatwo znaleźć skrajne wartości współrzędnych wierzchołków. Gdy znamy wartości c_1, c_2, d_1 oraz d_2, wówczas trzeba wyznaczyć przecięcia z wielokątem tylko prostych o parametrach $c_1 \leqslant c \leqslant c_2$ oraz $d_1 \leqslant d \leqslant d_2$ $(c, d \in \mathbb{Z})$.

Pozostaje jeszcze kwestia sortowania punktów przecięcia na jednej prostej. Tutaj także warto zastosować układ współrzędnych $(x-y, x+y)$. Okazuje się, że jeśli rozważana prosta

[2] Czytelnik niezaznajomiony z pojęciem *otoczenia* może sobie zastąpić ten termin w niniejszej definicji przez *koło bez brzegu*.

[3] Ćwiczenie dla Czytelnika: czym jest przekształcenie płaszczyzny, które punktowi o współrzędnych (x, y) przyporządkowuje właśnie punkt $(x-y, x+y)$?

jest postaci $x - y = c$, to punkty należy posortować względem wartości $x + y$. W nowym układzie oznacza to, że punkty na prostych pionowych należy posortować względem rzędnych. Dla prostych postaci $x + y = d$ punkty przecięcia należy posortować względem wartości $x - y$. W nowym układzie przenosi się to na sortowanie punktów na prostych poziomych względem odciętych.

Otrzymaliśmy więc pierwszy algorytm, który pozwala na rozwiązanie zadania.

Algorytm I

Przyjmijmy, że współrzędne wierzchołków są przechowywane odpowiednio w tablicach $x[1..n]$ oraz $y[1..n]$. Graf będziemy budowali w słowniku *sąsiedzi*, który współrzędnym punktu kratowego obwodu będzie przypisywał jego listę sąsiadów w grafie G (tzn. listę współrzędnych punktów kratowych obwodu, z którymi dany punkt jest połączony krawędzią). Aby wykonać sortowanie punktów na prostej, będziemy potrzebowali jeszcze pomocniczej tablicy $srt[1..MAXL]$, gdzie $MAXL$ oznacza maksymalną liczbę punktów kratowych na krawędzi (czyli 300 000). W tablicy tej będziemy zapisywali współrzędne punktów przecięcia z obwodem B prostych skośnych w każdym z dwóch kierunków.

Przyjmijmy, że funkcje pomocnicze sortuj_względem_sumy(L : *integer*) oraz sortuj_względem_różnicy(L : *integer*) sortują pierwsze L rekordów w tablicy $srt[1..MAXL]$, posługując się przy porównaniach wartościami odpowiednio $x + y$ lub $x - y$.

Otrzymujemy zatem następujący algorytm budowania grafu:

```
{ Znajdujemy ekstremalne parametry prostych c1, c2, d1 i d2. }
c1 := d1 := +∞;
c2 := d2 := −∞;
for i := 1 to n do
begin
  if x[i] − y[i] < c1 then c1 := x[i] − y[i];
  if x[i] − y[i] > c2 then c2 := x[i] − y[i];
  if x[i] + y[i] < d1 then d1 := x[i] + y[i];
  if x[i] + y[i] > d2 then d2 := x[i] + y[i];
end
{ Dla każdej prostej przecinającej wielokąt }
{ wyznaczamy zawarte w niej krawędzie grafu G. }
for c := c1 to c2 do
begin
  { L oznacza liczbę znalezionych w danym kroku przecięć. }
  L := 0;
  { Rozważamy prostą x − y = c. }
  { Znajdujemy wszystkie jej punkty przecięcia z obwodem wielokąta. }
  for i := 1 to n do
  begin
    { Badamy bok między wierzchołkami i oraz (i mod n) + 1. }
    next_i := (i mod n) + 1;
    if bok (i, next_i) przecina się z prostą x − y = c then
    begin
      L := L + 1;
```

```
26:          srt[L] := punkt_przecięcia(bok (i, next_i), prosta x − y = c);[4]
27:        end
28:      end
29:      { Sortujemy przecięcia srt[1..L] względem wartości x + y. }
30:      sortuj_względem_sumy(L);
31:      { Usuwamy wszystkie pozostałe punkty otarcia, }
32:      { czyli te punkty, które występują dwa razy obok siebie. }
33:      L2 := 1;
34:      i := 1;
35:      while i ⩽ L do
36:        if i < L and srt[i] = srt[i + 1] then i := i + 2
37:        else
38:        begin
39:          srt[L2] := srt[i];
40:          i := i + 1;
41:          L2 := L2 + 1;
42:        end
43:      L := L2;
44:      { Wyznaczamy krawędzie. }
45:      i := 1;
46:      while i ⩽ L do
47:      begin
48:        dodaj_do_listy(sąsiedzi[srt[i]], srt[i + 1]);
49:        dodaj_do_listy(sąsiedzi[srt[i + 1]], srt[i]);
50:        i := i + 2;
51:      end
52:  end
53:  for d := d1 to d2 do
54:  begin
55:      { Powtarzamy wszystko analogicznie dla prostej x + y = d, }
56:      { sortując względem różnicy x − y zamiast względem sumy. }
57:      ...
58:  end
```

Powyższa procedura konstruuje graf G, który wystarczy tylko przeszukać, aby znaleźć pary wierzchołków odpowiadające końcom ścieżek.

Przeanalizujmy złożoność tego algorytmu. Niech l oznacza liczbę punktów kratowych na obwodzie wielokąta. Dla każdej prostej przecinającej wielokąt (może być ich potencjalnie $O(l)$) sprawdzamy wszystkie boki (jest ich $O(n)$) w celu wykrycia możliwych przecięć. Tak więc samo wyznaczanie przecięć ma w tej metodzie koszt $O(n \cdot l)$. Koszt wykonywanych dwóch sortowań to $O(l \cdot \log l)$, jeżeli użyjemy jednego z szybkich algorytmów sortowania[5]. Trzeba jeszcze uwzględnić operacje na słowniku *sąsiedzi*, które są wykonywane $O(l)$ razy.

[4] Przy wyznaczaniu przecięć zakładamy, że odcinki zawierają ten z dwóch końców, który ma mniejszą wartość $x - y$, zaś drugiego nie zawierają. Dzięki temu łatwo jest odsiać punkty otarcia, gdyż są one albo wykrywane jako przecięcia dwukrotnie, albo w ogóle, podczas gdy każde inne przecięcie jest wykrywane dokładnie raz.

[5] Np. MergeSort bądź HeapSort — oba algorytmy są opisane w [17] i w [20].

Struktury słownikowe można tak zaimplementować, aby każda operacja na nich miała koszt $O(\log l)$[6]. Jednak nawet tak finezyjna ich realizacja niewiele nam pomoże, gdyż sumaryczna złożoność algorytmu już i tak wynosi $O(n \cdot l)$, co w naszym wypadku jest kosztem zbyt dużym.

Jeżeli jednak powrócimy do wspomnianego wcześniej przekształcenia układu współrzędnych (x,y) w układ $(x-y, x+y)$, to możemy znaleźć sposób na przyspieszenie algorytmu...

Rozwiązanie wzorcowe

Zamiast przykładać do każdej prostej każdy bok wielokąta, żeby sprawdzić, czy aby przypadkiem się nie przetną, możemy przetwarzać zarówno proste, jak i możliwe punkty przecięcia w pewnych analogicznych do siebie porządkach. Przecież każdy z możliwych punktów przecięcia, czyli punktów kratowych na obwodzie wielokąta, jest punktem przecięcia dla pewnej prostej. Tak więc zamiast szukać przecięć prostych z poszczególnymi bokami, możemy wziąć wszystkie punkty kratowe pojawiające się na obwodzie i przetwarzać je w takiej kolejności, w jakiej pojawiałyby się w poprzednim algorytmie. Jaka to kolejność?

Skupmy się na pierwszym kierunku prostych ($x - y = c$). Punkty kratowe obwodu są tam przetwarzane w blokach odpowiadających kolejnym wartościom c (dla kolejnych prostych). W każdym zaś z tych bloków są one sortowane względem wartości $x+y$. Tak więc szukany porządek to punkty posortowane w pierwszej kolejności względem „współrzędnej" $x-y$, zaś w drugiej kolejności (tzn. w przypadku takich samych wartości $x-y$) względem „współrzędnej" $x+y$.

Analogicznie postępujemy dla prostych biegnących w kierunku prostopadłym do poprzedniego (czyli prostych postaci $x+y=d$) — punkty kratowe obwodu są przetwarzane w porządku posortowanym w pierwszej kolejności względem $x+y$, zaś w drugiej kolejności względem $x-y$.

Możemy więc na samym początku analizy każdego z dwóch kierunków prostych posortować wszystkie punkty kratowe na obwodzie w odpowiedni sposób, a następnie przetwarzać je w tak otrzymanym porządku dokładnie taką samą metodą jak poprzednio. Trochę więcej uwagi trzeba jednak w tej sytuacji poświęcić odróżnianiu otarć od przebić. Zauważmy jednak, że jeżeli analizujemy punkty posortowane w pierwszej kolejności względem jednej z naszych przekształconych „współrzędnych" ($x-y$ lub $x+y$), a następnie względem drugiej, to punkt kratowy obwodu jest punktem otarcia wtedy i tylko wtedy, gdy jego sąsiedzi na obwodzie wielokąta (tj. poprzedni i następny punkt kratowy) mają takie same wartości pierwszej „współrzędnej". Analiza kilku możliwych przypadków potwierdza prawdziwość tego spostrzeżenia, a jego prostota bardzo ułatwi nam implementację.

Algorytm II — wzorcowy

W tablicach *brzeg_x*$[1..MAXL]$, *brzeg_y*$[1..MAXL]$ zapisujemy wszystkie punkty kratowe obwodu, które następnie będziemy sortować. Funkcje sortujące w tym algorytmie działają podobnie jak poprzednio na tablicy *srt*$[1..MAXL]$, która jednak tym razem zawiera tylko indeksy punktów na obwodzie w odpowiednio posortowanej kolejności.

Przy konstrukcji grafu od razu określamy dla każdego punktu, czy jest on punktem otarcia w każdym z kierunków. W tablicach wartości logicznych *otarcie_dla_sumy*$[1..MAXL]$ oraz

[6] O tym, jak to zrobić, można przeczytać w [17] lub w [20].

otarcie_dla_różnicy[1..*MAXL*] zapisujemy dla każdego punktu, czy jest on otarciem dla prostych w kierunku $x+y=d$ oraz w kierunku $x-y=c$.

W tablicy wartości logicznych *wierzchołek*[1..*MAXL*] pamiętamy, czy dany punkt kratowy obwodu jest wierzchołkiem wielokąta czy nie. Natomiast w tablicy wartości logicznych *odwiedzony*[1..*MAXL*], wypełnionej początkowo wartościami **false**, będziemy zaznaczać, które z punktów kratowych obwodu wielokąta zostały odwiedzone przy przeszukiwaniu grafu — ostatniej fazie algorytmu, mającej na celu wypisanie wszystkich żądanych par wierzchołków. Tak naprawdę wystarczy jako odwiedzone zaznaczać tylko wierzchołki wielokąta, czyli końce ścieżek, i tak też będziemy robić w poniższym algorytmie.

Zmienia się także reprezentacja grafu. Nie potrzebujemy już list sąsiadów w postaci słowników — dla każdego wierzchołka wystarczy nam zwykła tablica indeksowana numerami punktów na obwodzie, ponieważ zbiór tych punktów jest określany wspólnie dla obu kierunków prostych.

```
{ Znajdujemy wszystkie punkty kratowe na obwodzie. }
L := 0;
for i := 1 to n do
begin
  { Punkty kratowe na boku (i, (i mod n)+1). }
  next_i := (i mod n)+1;
  k := max(abs(x[i]-x[next_i]), abs(y[i]-y[next_i]));
  for j := 1 to k do
  begin
    L := L+1;
    brzeg_x[L] := (x[i]·(k-j)+x[next_i]·j)/k;
    brzeg_y[L] := (y[i]·(k-j)+y[next_i]·j)/k;
    wierzchołek[L] := (j=k);
  end
end
{ Zaznaczamy wszystkie punkty otarć. }
for i := 1 to L do
begin
  next_i := i+1;
  if next_i>n then next_i := 1;
  prev_i := i-1;
  if prev_i=0 then prev_i := n;
  otarcie_dla_różnicy[i] :=
    (brzeg_x[prev_i]-brzeg_y[prev_i]=brzeg_x[next_i]-brzeg_y[next_i]);
  otarcie_dla_sumy[i] :=
    (brzeg_x[prev_i]+brzeg_y[prev_i]=brzeg_x[next_i]+brzeg_y[next_i]);
end
{ Analizujemy proste postaci x-y=c. }
{ Sortujemy w pierwszej kolejności po x-y, a w drugiej po x+y. }
{ Wykonujemy to przez dwukrotne sortowanie — drugie musi być stabilne! }
```

[7] Sortowanie stabilne nie zmienia względnej kolejności elementów równych. Jeśli drugie z wykonanych sortowań jest stabilne, to posortowanie najpierw względem $x+y$, a później względem $x-y$ daje ciąg posortowany w pierwszej kolejności względem $x-y$, a w drugiej względem $x+y$. Naturalne, stabilne algorytmy sortowania to: sortowanie przez zliczanie, kubełkowe czy MergeSort. Jednak także

```
31: for i := 1 to n do srt[i] := i;
32: sortuj_względem_sumy(L);
33: sortuj_względem_różnicy(L);
34: { Łączymy punkty niebędące otarciami w pary, tworząc krawędzie grafu G. }
35: i := 0;
36: while i < L do
37: begin
38:   do i := i+1 while i < L and otarcie_dla_różnicy[srt[i]];
39:   początek := i;
40:   do i := i+1 while i < L and otarcie_dla_różnicy[srt[i]];
41:   koniec := i;
42:   if koniec ⩽ L then
43:   begin
44:     dodaj_do_listy(sąsiedzi[srt[początek]], srt[koniec]);
45:     dodaj_do_listy(sąsiedzi[srt[koniec]], srt[początek]);
46:   end
47: end
48: { Powtarzamy wszystko analogicznie dla prostych postaci x+y=d. }
49: ...
50: { Możemy umieścić n/2 par urządzeń. }
51: wypisz(n/2);
52: { Przeszukujemy wszystkie ścieżki w grafie i wypisujemy żądane pary. }
53: for i := 1 to L do
54:   if wierzchołek[i] and not odwiedzony[i] then
55:   begin
56:     last_j := i;
57:     j := początek_listy(sąsiedzi[i]);
58:     while rozmiar_listy(sąsiedzi[j]) = 2 do
59:     begin
60:       if początek_listy(sąsiedzi[j]) = last_j then
61:       begin
62:         last_j := j; j := koniec_listy(sąsiedzi[j]);
63:       end else
64:       begin
65:         last_j := j; j := początek_listy(sąsiedzi[j]);
66:       end
67:     end
68:     odwiedzone[i] := true;
69:     odwiedzone[j] := true;
70:     wypisz(i, j);
71:   end
```

Przeanalizujmy złożoność tego algorytmu. Wyznaczenie wszystkich punktów kratowych obwodu i oznaczenie, które z nich są otarciami, ma koszt $O(l)$. Sortowania wymagają

każdy z innych algorytmów sortowania, w tym QuickSort i HeapSort, można łatwo przekształcić w stabilny.

$O(l \cdot \log l)$ operacji. Następnie łączenie punktów obwodu w pary ma koszt liniowy ze względu na ich liczbę, czyli $O(l)$. Tak więc sumaryczna złożoność tego algorytmu to $O(l \cdot \log l)$, co przy ograniczeniach z zadania w zupełności wystarcza.

Rozwiązanie wzorcowe zostało zaimplementowane w plikach `szk.cpp` (bez użycia STL), `szk1.pas`, `szk2.cpp` (z użyciem STL) oraz `szk4.java`.

Inne rozwiązania

Oprócz wyżej podanych rozwiązań istnieje wiele innych wolniejszych. Tutaj wspomnimy o dwóch z nich.

W bardzo bezpośrednim podejściu do rozwiązania zadania można wykonać symulację biegu promienia z każdego wierzchołka „krok po kroku". Można ją wykonywać na dwa sposoby.

Symulacja I

Pierwszy z nich polega na symulowaniu ruchu promienia pomiędzy kolejnymi punktami kratowymi obwodu. W tym podejściu musimy jednak przy każdym kroku przeglądać wszystkie boki wielokąta, aby sprawdzić, z którym z nich następuje kolejne przecięcie promienia. Ponieważ takie sprawdzenie potencjalnie może być wykonywane dla każdego punktu kratowego obwodu, to złożonością takiego rozwiązania jest $O(n \cdot l)$.

Tego typu rozwiązania otrzymywały około 18 punktów. Przykład realizacji tego podejścia znajduje się w pliku `szks3.cpp`.

Symulacja II

Drugie podejście polega na symulacji ruchu promienia pomiędzy wszystkimi punktami kratowymi wewnątrz wielokąta, a nie tylko na obwodzie. Po starcie z danego wierzchołka przesuwamy pozycję promienia świetlnego w każdym kroku o jeden na każdej ze współrzędnych (w odpowiednim kierunku).

W każdym kroku musimy wtedy badać, czy nie trafiliśmy w wierzchołek lub obwód wielokąta. Możemy robić to bezpośrednio, przeglądając wszystkie wierzchołki. Ponieważ każdy punkt kratowy wewnątrz wielokąta zostanie w tym podejściu odwiedzony co najwyżej 4 razy (w czterech kierunkach), to metoda ta ma złożoność $O(S \cdot l)$, gdzie S — pole wielokąta. Rozwiązanie to znajduje się w pliku `szks2.cpp`.

Aby usprawnić sprawdzanie przecięć promienia z obwodem, możemy na początku wykonywania algorytmu umieścić wszystkie punkty kratowe obwodu oraz wszystkie wierzchołki w dwóch zrównoważonych drzewach BST (na przykład drzewach czerwono-czarnych zaimplementowanych w C++ w bibliotece STL pod nazwą *set*). Dzięki temu sprawdzenie przynależności punktu do obwodu oraz sprawdzenie, czy jest on wierzchołkiem, możemy wykonywać w czasie $O(\log l)$. więc otrzymamy algorytm o złożoności $O(S \cdot \log l)$. Został on zaimplementowany w pliku `szks1.cpp`.

Rozwiązanie to jest bardzo wydajne dla małych testów, jednak dla testów z dodanym do wielokąta fragmentem o dużym (rzędu kilkuset milionów) polu jest bardzo nieefektywne. Programy zaimplementowane w ten sposób otrzymywały około 30 punktów.

Testy

Programy zawodników były testowane na zestawie 15 testów.

Nazwa	n	l	Opis
szk1.in	380	796	mały test poprawnościowy
szk2.in	840	2 016	mały test poprawnościowy
szk3.in	4 310	9 540	mały test poprawnościowy
szk4.in	7 302	17 026	mały test poprawnościowo-wydajnościowy
szk5.in	1 512	10 188	mały test poprawnościowy
szk6.in	14 404	92 714	średni test poprawnościowy
szk7.in	16 002	107 762	średni test poprawnościowy
szk8.in	24 748	120 354	średni test poprawnościowy
szk9.in	32 610	225 910	duży test poprawnościowy
szk10.in	12 008	250 558	duży test poprawnościowy
szk11.in	43 666	295 342	duży test poprawnościowo-wydajnościowy
szk12.in	93 004	289 242	duży test wydajnościowy
szk13.in	97 328	290 010	duży test wydajnościowy
szk14.in	84 722	261 730	duży test poprawnościowo-wydajnościowy
szk15.in	28 654	285 884	duży test poprawnościowo-wydajnościowy

Testy 1–4 są obrazami narysowanymi za pomocą programu gimp przekształconymi do formatu z treści zadania. Pozostałe testy były tworzone przez kombinację różnego rodzaju sposobów generowania fragmentów obwodu wielokąta spełniającego warunki zadania i dodawanie do niego artefaktów. Wśród artefaktów były:

- „schodki” — struktura polegająca na naprzemiennych krokach losowej długości w określoną stronę, za pomocą której można budować długie fragmenty obwodu;
- „krzyże” — rekurencyjnie zagłębiająca się struktura fraktalna;
- „spirale” — rekurencyjna struktura w pełni oddająca kształtem swoją nazwę;
- „falbanki” — losowo „poszarpany” fragment obwodu;
- „losowa tuleja” — struktura polegająca na wytworzeniu wąskiego korytarza o losowym brzegu;
- „tuleja” — struktura polegająca na powtórzeniu pewnego fragmentu figury wiele razy i wytworzeniu w ten sposób długiego korytarza, w którym kilka promieni odbija się wielokrotnie w nietrywialny sposób.

W testach 6–15 dodatkowo zadbano, aby figura miała duże pole, co umożliwiło odsianie rozwiązań nieefektywnych o złożoności zależnej od tej wielkości.

Zawody II stopnia

opracowania zadań