Metody pomocnicze

Przypuśćmy, że będziemy pracować na punktach i wektorach w przestrzeni dwuwymiarowej, reprezentowanych przez następującą strukturę.

```
[Serializable]
public struct Point
{
    public double x;
    public double y;
    public Point(double px, double py) { x = px; y = py; }
}
```

Iloczyn wektorowy (a ściślej: opatrzony znakiem moduł iloczynu wektorowego) wyznaczamy w następujący sposób.

```
static double VectorProduct(Point p1, Point p2)
{
    return p1.x * p2.y - p2.x * p1.y;
}
```

Posługując się iloczynem wektorowym możemy sprawdzić, czy przechodząc kolejno przez trzy punkty p_0, p_1, p_2 wykonujemy skręt w lewo.

```
static bool IsLeftTurn(Point p0, Point p1, Point p2)
{
    Point p0p1 = new Point(p1.x - p0.x, p1.y - p0.y);
    Point p0p2 = new Point(p2.x - p0.x, p2.y - p0.y);
    return VectorProduct(p0p1, p0p2)>0;
}
```

Sprawdzenie, czy dwa odcinki przecinają się ze soba, wykonujemy następująco.

```
static bool SegmentIntersection(Point p1, Point p2, Point p3, Point p4)
{
    Point p34 = new Point(p4.x - p3.x, p4.y - p3.y);
    Point p31 = new Point(p1.x - p3.x, p1.y - p3.y);
    Point p32 = new Point(p2.x - p3.x, p2.y - p3.y);
    Point p12 = new Point(p2.x - p1.x, p2.y - p1.y);
    Point p13 = new Point(p3.x - p1.x, p3.y - p1.y);
    Point p14 = new Point(p4.x - p1.x, p4.y - p1.y);
    double d1 = VectorProduct(p34, p31);
    double d2 = VectorProduct(p34, p32);
    double d3 = VectorProduct(p12, p13);
    double d4 = VectorProduct(p12, p14);
    double d12 = d1 * d2;
    double d34 = d3 * d4;

    // Jeden odcinek leży w całości na prawo lub w całości na lewo od drugiego
    if (d12 > 0 || d34 > 0)
        return false;

    // Jeden z odcinków ma końce po obu stronach drugiego
    if (d12 < 0 && d34 < 0)
        return true;

    // Odcinki mają wspólny koniec
    if ((p1.x == p3.x && p1.y == p3.y) || (p1.x == p4.x && p1.y == p4.y)
        || (p2.x == p3.x && p2.y == p3.y) || (p2.x == p4.x && p2.y == p4.y))
        return true;

    // Odcinki są współliniowe - sprawdzamy w jednym wymiarze
    if (p1.x != p3.x)
        return (Math.Max(p1.x, p2.x) >= Math.Min(p3.x, p4.x))
                                    && (Math.Max(p3.x, p4.x) >= Math.Min(p1.x, p2.x));
    else
        return (Math.Max(p1.y, p2.y) >= Math.Min(p3.y, p4.y))
                                && (Math.Max(p3.y, p4.y) >= Math.Min(p1.y, p2.y));
}
```

Zadanie: łączenie pól

Warszawa, 2400

Zawód tradycyjnego rolnika zanikł wiele lat temu. Dziś farmer jest programistą i wydaje polecenia robotom pracującym na roli. W ramach ostatniej reformy rządu wszystkie pola uprawne danego właściciela zostaną połączone w jedno supergospodarstwo. Do tej pory były one wielokątami wypukłymi, których otoczka zawierała pola tylko jednego właściciela. Od przyszłego tygodnia właśnie ta otoczka będzie stanowić pole rolnika. Twoja firma dostarcza oprogramowanie do robotów pracujących na roli. Pomóż rolnikom dostosować się do nowych regulacji.

Część 1 - 1 pkt

Do tej pory rolicy nie przywiązywali uwagi do dokładnego określenia współrzędnych swoich pól. Na ten moment pole określone jest jako nieuporządkowana lista współrzędnych wierzchołków. Niektóre z nich mogą się powtarzać. Współrzędne mogą też tworzyć krawędzie przedłużające się. Zaimplementuj znany Ci algorytm wyznaczający otoczkę wypukłą i na jego podstawie zwróć posortowaną przeciwnie do ruchu wskazówek zegara listę współrzędnych otoczki wypukłej ograniczającej podany na wejściu zbiór punktów.

Część 2 - 1.5 pkt

Aby mieć pewność, że rolnicy nie oszukali władzy – pola będą łączone parami. Od setek lat roboty były niezawodne. Ich zbiór operacji był na tyle prosty i zoptymalizowany, że ich procesory to wybitnie niewydajne jednostki. Okazuje się, że robot będzie w stanie połączyć dwa pola tylko wtedy, gdy złożoność algorytmu łączenia tych pół będzie liniowa ze względu na sumę liczby ich wierzchołków. Zaproponuj algorytm, który to wykona. Załóż, że współrzędne wielokąta podane są jako lista wierzchołków w kolejności przeciwnej do ruchu wskazówek zegara.

Możesz do tego podejść następująco:

1. Podziel wielokąt na dwie częsci - otoczkę dolną (zawiera wierzchołki od tego o najmniejszej wartości współrzędnej x do tego o największej wartości współrzędnej x) oraz górną (pozostałe wierzchołki + ewentualnie pierwszy i ostatni wierzchołek otoczki dolnej).

2. Połącz w jedną listę otoczki górne obu wielokątów wykorzystując krok znany z algorytmu mergesort (tak aby punkty w połączonej liście były posowtowane względem wspołrzędnej x).

3. Analogicznie połącz otoczki dolne obu wielokątów.

4. Na podstawie listy utworzonej w pkt. 2 utwórz otoczkę górną obu wielokątów (wskazówka: możesz wykorzystać pomysł z algorytmu Grahama).

5. Analogicznie na podstawie listy utworzonej w pkt. 3 utwórz otoczkę dolną obu wielokątów.

6. Połącz obie „półotoczki" w wynikową otoczkę.

Uwaga 1: Obie części zadania są od siebie niezależne

Uwaga 2: Algorytm dla części 2 zadania musi mieć złożoność liniową, algorytm o gorszej złożoności nie będzie uznany.

Zadanie: Pole sumy prostokątów

Najpierw przeczytaj CAŁY niniejszy opis. Potem implementuj. Każdy inny sposób podejścia do tego zadania grozi niepowodzeniem.

Całe zadanie ma na celu przećwiczenie algorytmu z zamiataniem. Będziemy mierzyć się z następującym problemem: Mamy zbiór być może nakładających się prostokątów w dwuwymiarowej przestrzeni Euklidesowej, których boki są równoległe lub prostopadłe do osi układu współrzędnych. Naszym zadaniem jest policzyć pole tych prostokątów, ale w ten sposób, że jeśli jakiś fragment przestrzeni należy do więcej niż jednego prostokąta, to i tak liczymy go tylko raz (czyli liczymy pole sumy teoriomnogościowej tych prostokątów).

Aby rozwiązać to zadanie, najpierw zmierzymy się z zadaniem prostszym, czyli tym samym problemem, ale w jednowymiarowej przestrzeni czyli policzeniem długości być może nakładających się odcinków (dany fragment liczymy tylko raz). Po pierwsze jest to łatwiejsze do wyobrażenia sobie, a poza tym jest niezbędne, aby potem policzyć pola prostokątów.

UWAGA DO TESTÓW: aby każdy nie musiał sobie rozrysowywać przykładów testowych, są one narysowane w plikach pdf. Jeśli masz problem z konkretnym testem, po prostu otwórz plik o odpowiadającej mu nazwie, np. test1.pdf i spróbuj dojść, co jest nie tak w Twoim algorytmie. Dla pierwszego etapu, z odcinkami, w niektórych testach, dla czytelności, odcinki nie leżą na jednej linii, a na wielu równoległych liniach. Oczywiście tak naprawdę one wszystkie leżą na jednej prostej.

Etap 1 – długość odcinków (1p)

Mamy zbiór być może nakładających się odcinków w jednowymiarowej przestrzeni Euklidesowej, które siłą rzeczy znajdują się na jednej prostej. Naszym zadaniem jest policzyć długość tych odcinków, ale w ten sposób, że jeśli jakiś fragment przestrzeni należy do więcej niż jednego odcinka, to i tak liczymy go tylko raz (czyli liczymy pole sumy teoriomnogościowej tych odcinków).

UWAGA TECHNICZNA: będziemy dostawać na wejściu ogólną klasę Segment, który to odcinek może być w dowolnej pozycji w dwuwymiarowej przestrzeni. Jednak mamy zagwarantowane, że wszystkie one będą ustawione na jednej PIONOWEJ linii prostej (czyli x-owa współrzędna jest taka sama, różni się jedynie y-owa). Dlaczego pionowa? Bo nam się przyda w drugim etapie, przy prostokątach.

Wyobraźmy sobie taki zbiór pionowych odcinków. Wyobraźmy sobie poziomą linię, która przemieszcza się od dołu do góry, zatrzymując się na punktach wyznaczających początek lub koniec jakiegoś odcinka. Każdy taki punkt i każde takie zatrzymanie się linii nazywamy źdarzeniem". Tak naprawdę w ogólności w przestrzeni mamy kilka serii nakładających się odcinków (zanim jeden odcinek się skończy, to już się zaczyna inny). Gdy poznamy długość każdej z takich serii, odpowiedzią będzie suma ich długości. Musimy wykryć, gdzie taka pojedyncza seria się zaczyna, a gdzie kończy, czyli mieć jej zakres. Zakres da nam długość.

Innymi słowy, musimy wykryć, w przy którym zdarzeniu weszliśmy w serię odcinków, a w którym zdarzeniu z niej wyszliśmy. Będzie to przypominać sprawdzanie, czy jesteśmy w nawiasie w tekście (z poprawnymi nawiasami). Będziemy zliczać otwierające się nawiasy (czyli punkty, które są początkami odcinków) i zamykające (punkty, które są końcami odcinków). Gdy różnica między tymi dwoma liczbami wyniesie zero, to właśnie zakończyła się seria.

A teraz szczegóły:

1. Stwórz strukturę, która ma indeks odcinka i informację, czy reprezentuje jego początek czy koniec. Struktura reprezentuje jeden z punktów krańcowych odcinka.
2. Stwórz listę tych struktur dla odcinków wejściowych.
3. Posortuj tę listę po y-owej współrzędnej.
4. Teraz zaimplementuj przechodzenie linii zamiatającej po zdarzeniach od dołu do góry. Pamiętaj, ile odcinków się zaczęło, a ile zakończyło.
5. Gdy zakończyła się seria nakładających się odcinków (czyli gdy liczba odcinków rozpoczętych i zakończonych jest taka sama) policz długość tej serii i dodaj do sumy ogólnej.
6. Wynikiem jest suma długości uzyskanych w poprzednim punkcie.

Etap 2 – pole prostokątów (1.5p)

Teraz, gdy mamy algorytm na długość odcinków, możemy przystąpić do prostokątów.

Wyobraź sobie prostokąty. Teraz wyobraź sobie pionową linię zamiatającą. Idzie ona od lewej do prawej. Tym razem zatrzymuje się ona za każdym razem, gdy natrafi na pionowy bok prostokąta (każdy prostokąt ma dwa takie boki). Takie zatrzymanie nazywamy zdarzeniem. W czasie zatrzymania obliczamy, jaka jest długość przecięcia wszystkich prostokątów z linią zamiatającą. Oznaczmy tę długość D. Nazwijmy ją wysokością. Teraz wiemy, że ta wysokość nie zmieni się do następnego zdarzenia. Dlatego jeśli jedno zdarzenie zdarzyło się na współrzędnej x1, a następne na x2, to pole prostokątów między tymi zdarzeniami wynosi (x2-x1)*D. Suma takich pól jest szukanym polem.

Szczegóły (zakładamy, że przechodzimy od lewej do prawej):

1. Użyj tej samej struktury, co w etapie 1. Stwórz listę zdarzeń odpowiadających pionowym bokom prostokąta. Tym razem wartość logiczna oznacza, czy to jest lewy bok prostokąta (bok otwierający prostokąt), czy prawy (zamykający prostokąt).

2. Posortuj zdarzenia (pionowe boki prostokąta) po x-owej współrzędnej.

3. Stwórz pomocniczą listę odcinków. Głównym problemem jest obliczenie przecięcia linii zamiatającej z prostokątami. Będziemy sobie radzić tak:

   - gdy natrafimy na odcinek, który otwiera prostokąt, będziemy go wrzucać na listę pomocniczą.
   - gdy natrafimy na odcinek, który kończy prostokąt, to będziemy z tej listy usuwać odpowiadający mu odcinek, (który ten prostokąt otwierał)

   W ten sposób zawsze na liście będziemy mieli odcinki wyznaczające zakresy pionowe prostokątów, z którymi linia zamiatająca się przecina w danym zdarzeniu. Oczywiście, one mogą się nakładać, tak jak prostokąty. Dlatego robiliśmy pierwszy etap.

4. Teraz przechodzimy linią zamiatającą. Wrzucamy i wyrzucamy odpowiednie odcinki z listy pomocniczej z poprzedniego punktu. Obliczamy (x2-x1)*D za ten fragment, który przebyliśmy od poprzedniego zdarzenia. Obliczamy, korzystając z etapu 1, łączną długość odcinków na liście pomocniczej. To jest nasze D dla następnej iteracji.

5. Zwracamy sumę pól.