# 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.