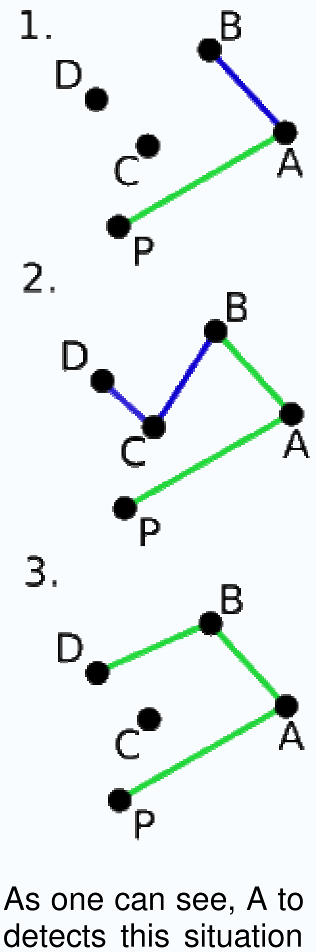
Задача 1. Овце

Лесно се забелязва, че за да намерим най-голямата поляна, оградена от дървета това е все едно да намерим полигона с най-голямо лице – ами това всъщност е изпъкналата обвивка на всички дървета. Т.е. задачата е сведена до намиране на изпъкнала обвивка и след това намиране на лицето на изпъкналата обвивка.

Тук ще копирам материали от Wikipedia и MathWorld, които описват алгоритмите за тези два проблема.

Graham Scan Algorithm (за изпъкнала обвивка)



As one can see, A to B and B to C are counterclockwise, but C to D isn't. The algorithm detects this situation and discards previously chosen segments until the turn taken is counterclockwise (B to D in this case.)

The first step in this algorithm is to find the point with the lowest y-coordinate. If the lowest y-coordinate exists in more than one point in the set, the point with the lowest x-coordinate out of the candidates should be chosen. Call this point *P*. This step takes O(*n*), where *n* is the number of points in question.

Next, the set of points must be sorted in increasing order of the angle they and the point *P* make with the x-axis. Any general-purpose sorting algorithm is appropriate for this, for example heapsort (which is O(*n* log *n*)). In order to speed up the calculations, it is not actually necessary to calculate the actual angle these points make with the x-axis; instead, it suffices to calculate the cotangent of this angle: it is a monotonically decreasing function in the domain in question (which is 0 to 180 degrees, due to the first step) and may be calculated with simple arithmetic.

The algorithm proceeds by considering each of the points in the sorted array in sequence. For each point, it is determined whether moving from the two previously considered points to this point is a "left turn" or a "right turn". If it is a "right turn", this means that the second-to-last point is not part of the convex hull and should be removed from consideration. This process is continued for as long as the set of the last three points is a "right turn". As soon as a "left turn" is encountered, the algorithm moves on to the next point in the sorted array. (If at any stage the three points are collinear, one may opt either to discard or to report it, since in some applications it is required to find all points on the boundary of the convex hull.)

Again, determining whether three points constitute a "left turn" or a "right turn" does not require computing the actual angle between the two line segments, and can actually be achieved with simple arithmetic only. For three points (x_1,y_1), (x_2,y_2) and (x_3,y_3), simply compute the direction of the cross product of the two vectors defined by points (x_1,y_1), (x_2,y_2) and (x_1,y_1), (x_3,y_3), characterized by the sign of the expression $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$. If the result is 0, the points are collinear; if it is positive, the three points constitute a "left turn", otherwise a "right turn".

This process will eventually return to the point at which it started, at which point the algorithm is completed and the stack now contains the points on the convex hull in counterclockwise order.

Time complexity

Sorting the points has time complexity O(*n* log *n*). While it may seem that the time complexity of the loop is O(n^2), because for each point it goes back to check if any of the previous points make a "right turn", it is actually O(*n*), because each point is considered at most twice in some sense. Each point can appear only once as a point (x3,y3) in a "left turn" (because the algorithm advances to the next point (x3,y3) after that), and as a point (x2,y2) in a "right turn" (because the point (x2,y2) is removed). The overall time complexity is therefore O(*n* log *n*), since the time to sort dominates the time to actually compute the convex hull.

Pseudocode

First, define

```
# Three points are a counter-clockwise turn if ccw > 0, clockwise if
# ccw < 0, and collinear if ccw = 0 because ccw is a determinant that
# gives the signed area of the triangle formed by p1, p2, and p3.
function ccw(p1, p2, p3):
    return (p2.x - p1.x)*(p3.y - p1.y) - (p2.y - p1.y)*(p3.x - p1.x)
```

Then let the result be stored in the array `points`.

```
let N           = number of points
let points[N+1] = the array of points
swap points[1] with the point with the lowest y-coordinate
sort points by polar angle with points[1]

# We want points[0] to be a sentinel point that will stop the loop.
let points[0] = points[N]

# M will denote the number of points on the convex hull.
let M = 2
for i = 3 to N:
    # Find next valid point on convex hull.
    while ccw(points[M-1], points[M], points[i]) <= 0:
       M -= 1

    # Swap points[i] to the correct place and update M.
    M += 1
    swap points[M] with points[i]
```

This pseudocode is adapted from Sedgewick and Wayne's *Algorithms, 4th edition*.

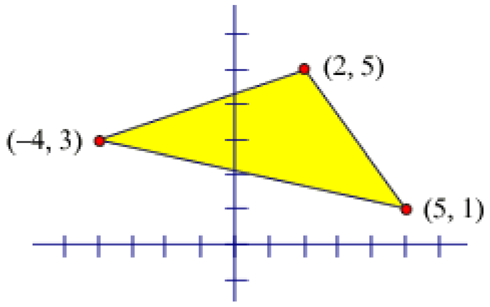
Лице на полигон.

Area of a Convex Polygon

The coordinates (x_1, y_1), (x_2, y_2), (x_3, y_3), . . . , (x_n, y_n) of a convex polygon are arranged in the "determinant" below. The coordinates must be taken in counterclockwise order around the polygon, beginning and ending at the same point.

$$\text{Area} = \frac{1}{2}\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \vdots & \vdots \\ x_n & y_n \\ x_1 & y_1 \end{vmatrix} = \frac{1}{2}\left[(x_1y_2 + x_2y_3 + x_3y_4 + \cdots + x_ny_1) - (y_1x_2 + y_2x_3 + y_3x_4 + \cdot\right.$$

Example:



Find the area of this polygon:

$$\text{Area} = \frac{1}{2}\begin{vmatrix} 2 & 5 \\ -4 & 3 \\ 5 & 1 \\ 2 & 5 \end{vmatrix} = \frac{1}{2}\left[(6 + -4 + 25) - (-20 + 15 + 2)\right] = 15$$