Codility_

CodeCheck Report: trainingX8CAJM-GDH

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task		Time spent	Score
PolygonConcavityIndex Python	⚠	6 min	100%

Total score

100%

Tasks Details

Hard

1. **PolygonConcavityIndex**
Check whether a given polygon in a 2D plane is convex; if not, return the index of a vertex that doesn't belong to the convex hull.

Task Score	Correctness	Performance
100%	100%	100%

Task description

An array A of points in a 2D plane is given. These points represent a polygon: every two consecutive points describe an edge of the polygon, and there is an edge connecting the last point and the first point in the array.

A set of points in a 2D plane, whose boundary is a straight line, is called a *semiplane*. More precisely, any set of the form {(x, y) : ax + by ≥ c} is a semiplane. The semiplane contains its boundary.

A polygon is *convex* if and only if, no line segment between two points on the boundary ever goes outside the polygon.

For example, the polygon consisting of vertices whose Cartesian coordinates are consecutively:

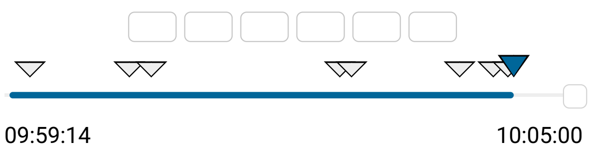
```
(-1, 3)   (3, 1)   (0, -1)   (-2, 1)
```

is convex.

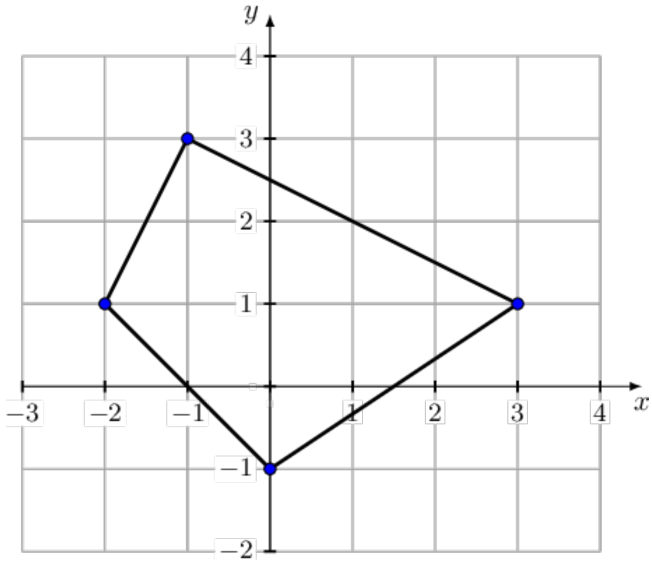
Solution

Programming language used:	Python	
Total time used:	6 minutes	?
Effective time used:	6 minutes	?
Notes:	*not defined yet*	

Task timeline ?



Code: 10:04:59 UTC, py, show code in pop-up

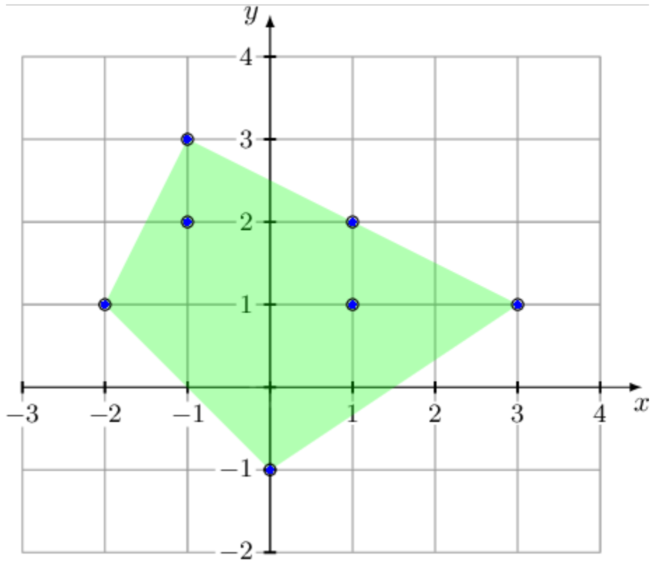


The *convex hull* of a finite set of points in a 2D plane is the smallest convex polygon that contains all points in this set. For example, the convex hull of a set consisting of seven points whose Cartesian coordinates are:

```
  (−1, 3)   (1, 2)   (3, 1)   (1, 1)   (0, −1)
(−2, 1)  (−1, 2)
```

is a polygon that has five vertices. When traversed clockwise, its vertices are:

```
  (−1, 3)   (1, 2)   (3, 1)   (0, −1)   (−2, 1)
```



If a polygon is concave (that is, it is not convex), it has a vertex which does not lie on its convex hull border. Your assignment is to find such a vertex.

Assume that the following declarations are given:

```
from dataclasses import dataclass, field

@dataclass
class Point2D:
    x: int
    y: int
```

Write a function:

```
def solution(A)
```

that, given a non-empty array A consisting of N elements describing a polygon, returns −1 if the polygon is convex.

final, score: **100**

```
from extratypes import Point2D  # library wit

def solution(A):
    result = 0
    is_convex = True

    for p0, p1, p2 in zip(A, A[1:] + A[:1], A
        p0p1_x, p0p1_y = p1.x − p0.x, p1.y −
        p0p2_x, p0p2_y = p2.x − p0.x, p2.y −
        p0p1_p0p2 = p0p1_x * p0p2_y − p0p1_y

        if p0p1_p0p2 < 0:
            if result > 0:
                is_convex = False
                break
            result −= 1
        if p0p1_p0p2 > 0:
            if result < 0:
                is_convex = False
                break
            result += 1

    if is_convex:
        return −1

    min_point_i = 0
    min_x, min_y = A[0].x, A[0].y

    for i, a in enumerate(A[1:]):
        if a.y < min_y:
            min_x, min_y = a.x, a.y
            min_point_i = i + 1
        elif a.y == min_y:
            if a.x < min_x:
                min_x = a.x
                min_point_i = i + 1

    is_clockwise = True
    tmp_A = [A[−1]] + A + [A[0]]
    p0, p1, p2 = tmp_A[min_point_i], tmp_A[mi

    p0p1_x, p0p1_y = p1.x − p0.x, p1.y − p0.y
    p1p2_x, p1p2_y = p2.x − p1.x, p2.y − p1.y
    p0p1_p0p2 = p0p1_x * p1p2_y − p0p1_y * p1

    if p0p1_p0p2 > 0:
        is_clockwise = False

    for i, (p0, p1, p2) in enumerate(zip(A[−1
        p0p1_x, p0p1_y = p1.x − p0.x, p1.y −
        p1p2_x, p1p2_y = p2.x − p1.x, p2.y −
        p0p1_p0p2 = p0p1_x * p1p2_y − p0p1_y

        if is_clockwise and p0p1_p0p2 > 0:
            return i
        if not is_clockwise and p0p1_p0p2 < 0
            return i
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

expand all	Example tests	
▶	example1	✔ OK

Otherwise, the function should return the index of any point that doesn't belong to the convex hull border. Note that consecutive edges of the polygon may be collinear (that is, the polygon might have 180−degrees angles).

To access the coordinates of the K-th point (where 0 ≤ K < N), use the following syntax:

- `A[K].x` to access the x-coordinate,
- `A[K].y` to access the y-coordinate.

For example, given array A such that:

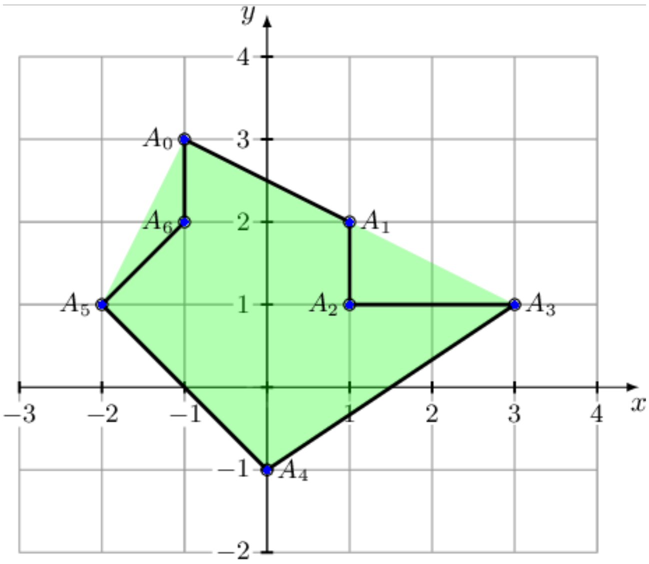
```
A[0].x = −1  A[0].y =  3
A[1].x =  1  A[1].y =  2
A[2].x =  3  A[2].y =  1
A[3].x =  0  A[3].y = −1
A[4].x = −2  A[4].y =  1
```

the function should return −1, as explained in the example above.

However, given array A such that:

```
A[0].x = −1  A[0].y =  3
A[1].x =  1  A[1].y =  2
A[2].x =  1  A[2].y =  1
A[3].x =  3  A[3].y =  1
A[4].x =  0  A[4].y = −1
A[5].x = −2  A[5].y =  1
A[6].x = −1  A[6].y =  2
```

the function should return either 2 or 6. These are the indices of the polygon lying strictly in its convex hull (that is, not on the convex hull border).



Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [3..10,000];
- the coordinates of each point in array A are integers within the range [−1,000,000,000..1,000,000,000];
- no two edges of the polygon A intersect, other than meeting at their endpoints;
- array A does not contain duplicate points.

first example test

▶ example2 second example test	✔ OK

expand all **Correctness tests**

▶ simple0 boomerang	✔ OK
▶ simple1 star	✔ OK
▶ simple2	✔ OK
▶ simple3 the polygon has exactly one angle equals to (90 + epsilon) degrees	✔ OK
▶ corner_cases corner cases	✔ OK
▶ cyclic all possible representations of a simple case	✔ OK
▶ collinear_vertices tests with many collinear triples of vertices	✔ OK
▶ medium1	✔ OK
▶ medium2	✔ OK

expand all **Performance tests**

▶ big1 almost diamond	✔ OK
▶ big2	✔ OK
▶ big3	✔ OK