Self Crossing

You are given an array *x* of `n` positive numbers. You start at point `(0,0)` and moves `x[0]` metres to the north, then `x[1]` metres to the west, `x[2]` metres to the south, `x[3]` metres to the east and so on. In other words, after each move your direction changes counter-clockwise.

Write a one-pass algorithm with `O(1)` extra space to determine, if your path crosses itself, or not.

Example 1:
Given *x* = `[2, 1, 1, 2]`
Return `true` (self crossing)
Example 2:
Given *x* = `[1, 2, 3, 4]`
Return `false` (not self crossing)
Example 3:
Given *x* = `[1, 1, 1, 1]`
Return `true` (self crossing)

Credits:
Special thanks to @dietpepsi for adding this problem and creating all test cases.

Solution 1

```
// Categorize the self-crossing scenarios, there are 3 of them:
// 1. Fourth line crosses first line and works for fifth line crosses second line
and so on...
// 2. Fifth line meets first line and works for the lines after
// 3. Sixth line crosses first line and works for the lines after
public class Solution {
    public boolean isSelfCrossing(int[] x) {
        int l = x.length;
        if(l <= 3) return false;

        for(int i = 3; i < l; i++){
            if(x[i] >= x[i-2] && x[i-1] <= x[i-3]) return true;  //Fourth line cr
osses first line and onward
            if(i >=4)
            {
                if(x[i-1] == x[i-3] && x[i] + x[i-4] >= x[i-2]) return true; // F
ifth line meets first line and onward
            }
            if(i >=5)
            {
                if(x[i-2] - x[i-4] >= 0 && x[i] >= x[i-2] - x[i-4] && x[i-1] >= x
[i-3] - x[i-5] && x[i-1] <= x[i-3]) return true;  // Sixth line crosses first lin
e and onward
            }
        }
        return false;
    }
}
```

written by KuangYuan original link here

Solution 2

After drawing a few crossing cases ourselves, we can simply find out there are two basic patterns:

- x[i-1]<=x[i-3] && x[i]>=x[i-2] the ending circle line cross the beginning circle line in one circle;
- i>=5 && x[i-1]<=x[i-3] && x[i]>=x[i-2]-x[i-4] the second line of the next circle cross the the beginning of the previous circle between two adjacent circles;

But still that is not over yet, how about some special cases? How about the first line of the next circle and the previous circle? Yeah, the beginning line of the next circle can overlap the the first line of the previous circle - another two adjacent circles case:

- i>=4 && x[i-1]==x[i-3] && x[i]>=x[i-2]-x[i-4]

Quite straightforward. Then we can test our patterns now, however soon we will find out that the second cases is not strong enough to cover all possible situations - the second line of the next circle crossing the previous circle at the its first line

- [3,3,3,2,1,1] is an example here, so x[i-2]>=x[i-4] then must be added to our conditions;
- [3,3,4,4,10,4,4,,3,3] is another typical example for x[i-3]<=x[i-1]+x[i-5] condition, which also should be added to make the constrained conditions stronger;

At last, we make it! Bang! End of story with a very terse, clean and efficient code as follows.

```
//AC - 0ms;
bool isSelfCrossing(int* x, int size)
{
    for(int i = 3; i < size; i++)
    {
        if(x[i]>=x[i-2] && x[i-1]<=x[i-3]) return true;
        if(i>=4 && x[i-1]==x[i-3] && x[i]+x[i-4]>=x[i-2]) return true;
        if(i>=5 && x[i-2]-x[i-4]>=0 && x[i]>=x[i-2]-x[i-4] && x[i-1]>=x[i-3]-x[i-
5] && x[i-1]<=x[i-3]) return true;
    }
    return false;
}
```

written by LHearen original link here

Solution 3

Checking out every six pack.

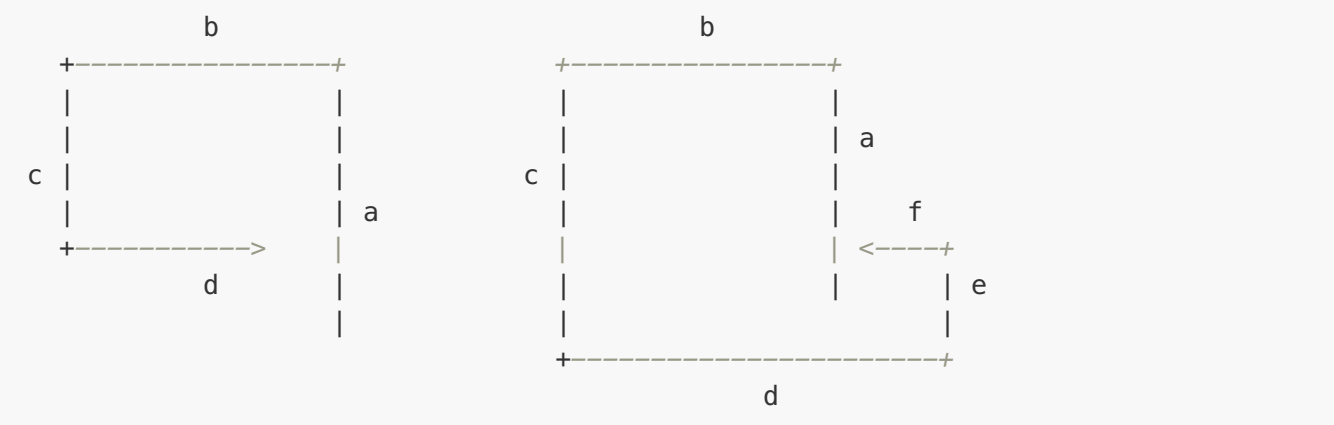
Solution 1

```
def isSelfCrossing(self, x):
    return any(d >= b > 0 and (a >= c or a >= c-e >= 0 and f >= d-b)
               for a, b, c, d, e, f in ((x[i:i+6] + [0] * 6)[:6]
                                        for i in xrange(len(x))))
```

Solution 2

```
def isSelfCrossing(self, x):
    b = c = d = e = 0
    for a in x:
        if d >= b > 0 and (a >= c or a >= c-e >= 0 and f >= d-b):
            return True
        b, c, d, e, f = a, b, c, d, e
    return False
```

Explanation



Draw a line of length `a`. Then draw further lines of lengths `b`, `c`, etc. How does the `a`-line get crossed? From the left by the `d`-line or from the right by the `f`-line, see the above picture. I just encoded the criteria for actually crossing it.

Two details:

- In both cases, `d` needs to be at least `b`. In the first case to cross the `a`-line directly, and in the second case to get behind it so that the `f`-line can cross it. So I factored out `d >= b`.
- The "special case" of the `e`-line stabbing the `a`-line from below is covered because in that case, the `f`-line "crosses" it (note that even if there is no actual `f`-line, my code uses `f = 0` and thus still finds that "crossing").

written by StefanPochmann original link here