[Testing DeepestPit]

Candidate

E-mail:

Session

ID: try7TB85B-P4W
Time limit: 60 min.
Report recipients: george@pligor.com
Accessed from: 79.107.114.195

Status: closed

Created on: 2016-06-01 21:46 UTC
Started on: 2016-06-01 21:46 UTC
Finished on: 2016-06-01 21:48 UTC

Notes:
N/A

Similarity Check

For paid accounts, all submissions are checked against improper behavior (similarity against other solutions).
To activate this feature upgrade your account.

Tasks in test	Correctness	Performance	Task score
1 DeepestPit Submitted in: JavaScript	66%	100%	80%

Test score

80%

80 out of 100 points

MEDIUM

1. DeepestPit

score: 80 of 100

Given an array of integers, find a bitonic sequence with maximal difference between the middle term and the first and the last terms.

Task description

A non-empty zero-indexed array A consisting of N integers is given. A *pit* in this array is any triplet of integers (P, Q, R) such that:

- 0 ≤ P < Q < R < N;
- sequence [A[P], A[P+1], ..., A[Q]] is strictly decreasing,
  i.e. A[P] > A[P+1] > ... > A[Q];
- sequence A[Q], A[Q+1], ..., A[R] is strictly increasing,
  i.e. A[Q] < A[Q+1] < ... < A[R].

The *depth* of a pit (P, Q, R) is the number min{A[P] − A[Q], A[R] − A[Q]}.

For example, consider array A consisting of 10 elements such that:

```
  A[0] =  0
  A[1] =  1
  A[2] =  3
  A[3] = -2
  A[4] =  0
  A[5] =  1
  A[6] =  0
  A[7] = -3
  A[8] =  2
  A[9] =  3
```

Triplet (2, 3, 4) is one of pits in this array, because sequence [A[2], A[3]] is strictly decreasing (3 > −2) and sequence [A[3], A[4]] is strictly increasing (−2 < 0). Its depth is min{A[2] − A[3], A[4] − A[3]} = 2. Triplet (2, 3, 5) is another pit with depth 3. Triplet (5, 7, 8) is yet another pit with depth 4. There is no pit in this array deeper (i.e. having depth greater) than 4.

Write a function:

```
function solution(A);
```

that, given a non-empty zero-indexed array A consisting of N integers, returns the depth of the deepest pit in array A. The function should return −1 if there are no pits in array A.

For example, consider array A consisting of 10 elements such that:

```
  A[0] =  0
  A[1] =  1
  A[2] =  3
  A[3] = -2
  A[4] =  0
  A[5] =  1
  A[6] =  0
  A[7] = -3
  A[8] =  2
  A[9] =  3
```

the function should return 4, as explained above.

Assume that:

- N is an integer within the range [1..1,000,000];
- each element of array A is an integer within the range [−100,000,000..100,000,000].

Complexity:

Solution

Programming language used: JavaScript

Total time used: 2 minutes

Effective time used: 2 minutes

Notes: *not defined yet*

Source code

Code: 21:48:38 UTC, js, final,
score: 80

```javascript
// you can write to stdout for debugging purposes, e.g.
// console.log('this is a debug message');

function solution(array) {
    var curDepth = -1
    var goDeepDepth = -1

    //var curTriplet: [number, number, number] = [-1,-1,-1]  //invalid
    var curP = -1
    var curQ = -1
    var curR = -1
    var curStep = -1

    var bestDepth = -1
    var bestP = -1
    var bestQ = -1
    var bestR = -1
    var bestStep = -1

    function makeCurrentPitBest() {
        //console.log("makeCurrentPitBest")
        //console.log(curR)

        bestP = curP
        bestQ = curQ
        bestR = curR
    }

    var i = 0;

    var current = function () {
        return i in array ? array[i] : null
    }

    var next = function () {
        return (i + 1) in array ? array[i + 1] : null
    }

    function check() {
        return (current() !== null) && (next() !== null) && (typeof current() != "undefined") && (typeof next() != "undefined")
    }


    function skipRemainingInClimbing() {
        while (check() && (current() < next())) {
            i = i + 1
        }
    }

    function goDeep() {
        curP = i

        while (check() && (current() > next())) {
```

- expected worst-case time complexity is O(N);
- expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

```
            i = i + 1
        }

        curQ = i
        //console.log(curQ)

        //console.log(array[curP])
        //console.log(array[curQ])

        return curP === curQ ? -1 : array[curP] - array[curQ]
    }

    //suppose you have the go deep depth
    function climb() {
        curDepth = -1

        //console.log("curP: " + curP)
        //console.log("curQ: " + curQ)
        //console.log("current: " + current())
        //console.log("next: " + next())

        var qValue = array[curQ]

        while (check() && (current() < next())) {
            //console.log("mpainw")
            //console.log("curDepth")
            //console.log(curDepth)

            curDepth = next() - qValue

            i = i + 1

            if (curDepth >= goDeepDepth) {
                curDepth = goDeepDepth
                break;
            }
        }

        skipRemainingInClimbing()

        curR = i

        return curQ === curR ? -1 : curDepth
    }

    var len = array.length

    while (true) {
        goDeepDepth = goDeep()

        curDepth = climb()

        /*if (goDeepDepth > bestDepth) {
            bestDepth = goDeepDepth
            makeCurrentPitBest()

            skipRemainingInClimbing()
        } else {
            curDepth = climb()

            if (curDepth > bestDepth) {
                bestDepth = curDepth
                makeCurrentPitBest()
            }
        }*/

        if (curDepth > bestDepth) {
            bestDepth = curDepth
            makeCurrentPitBest()
        }

        curStep = Math.min(curQ - curP, curR - curQ)
        if(curStep > bestStep) {
            bestStep = curStep
        }

        //console.log("curR: "+ curR)

        if (((bestStep * 2) >= (len - curR + 1)) ||    //it is not the best depth it count something else
            (i + 1 == len)) {
            break;
        }

        //i = i + 1

        //break;
    }

    //console.log(goDeepDepth)

    //console.log(bestP)
    //console.log(bestQ)
    //console.log(bestR)

    return bestDepth;
    /*//goDeep()
     console.log(goDeepDepth)
     console.log(i)

     climb()
     console.log(i)

     console.log(curP)
     console.log(curQ)
     console.log(curR)
     console.log(curDepth)*/
}
```

Analysis summary

The following issues have been detected: timeout errors.

Analysis

Detected time complexity:
O(N)

Example tests	
example example test	✔ OK
Correctness tests	
extreme_no_pit small test cases	✔ OK
extreme_depth TIMEOUT ERROR running time: >11.00 sec., time limit: 8.00 sec.	✘
simple1 no pit	✘ TIMEOUT ERROR running time: >11.00 sec., time limit: 8.00 sec.
simple2 one pit	✔ OK
user user-defined test case	✔ OK
simple3 `vulcano' shape	✔ OK
retries retries	✘ TIMEOUT ERROR running time: >11.00 sec., time limit: 8.00 sec.
medium1 medium correctness test	✔ OK
medium_pit medium test one pit	✔ OK
Performance tests	
large_pit_1 large test one pit 1	✔ OK
large_pit_2 large test one pit 2	✔ OK
big_pit_1	✔

big test one pit 1	OK
big_pit_2 big test one pit 1	✓ OK
big3_1 large random test	✓ OK
big3_2 big random test	✓ OK