Results from UVA

0.1 Problem A

17246735	1709 Amalgamated Artichokes	Accepted	JAVA	4.410	2016-04-22 19:22:25

0.2 Problem D

17274629	1712 Cutting Cheese	Accepted	JAVA	1.700	2016-04-27 22:19:28

1 Problem A: Amalgamated Artichokes

Results:

17246735	1709 Amalgamated Artichokes	Accepted	JAVA	4.410	2016-04-22 19:22:25

Background:

Fatima Cynara is an analyst at Amalgamated Artichokes (AA). As with any company, AA has had some very good times as well as some bad ones. Fatima does trending analysis of the stock prices for AA, and she wants to determine the largest decline in stock prices over various time spans. For example, if over a span of time the stock prices were 19, 12, 13, 11, 20 and 14, then the largest decline would be 8 between the first and fourth price. If the last price had been 10 instead of 14, then the largest decline would have been 10 between the last two prices.

Fatima has done some previous analyses and has found that the stock price over any period of time can be modelled reasonably accurately with the following equation:

$$price(x) = p \cdot (sin(a \cdot x + b) + cos(c \cdot x + d) + 2)$$

where $p, a, b, c,$ *and* d are constants.Fatima would like you to write a program to determine the largest price decline over a given sequence of prices. You have to consider the prices only for integer values of x.

Input:

The input file contains several test cases. Each test case is on a single line containing 6 integers, p $(1 \leq p \leq 1000)$, a, b, c, d $(0 \leq a, b, c, d \leq 1000)$, *and* n $(1 \leq n \leq 10^6)$. The first 5 integers are described above. The sequence of stock prices to consider are $price(1), price(2), ..., price(n)$.

Output:

For each test case, display the maximum decline in stock prices. If there is no decline, display the number '0'. Your output should have an absolute or relative error of at most 10^{-6}.

Sample Input:

42 1 23 4 8 10
100 7 615 998 801 3
100 432 406 867 60 1000

Sample Output:

104.855110477
0.00
399.303813

1.1 Mathematical Formulation

Given an input of integers $p, a, b, c, d,$ *and* n, the formula $f(x) = p \cdot (sin(a \cdot x + b) + cos(c \cdot x + d) + 2)$ where $x \in [1, n]$, determine the largest decrease between the integer values x_i, x_j where $i<j$ and $x_i \geq x_j$ and there does not exist another pair x_k, x_l where $k<l$ and $x_k \geq x_l$ but $x_k - x_l > x_i - x_j$.

1.2 Solution

The main functionality of this algorithm is to plug in each point keeping track of the highest seen point, h, the lowest seen point occuring after l, and the largest difference, $d = h - l$. It should be noted that since we are always taking the difference between the two values, we can factor out the $\cdot p$ as well as neglect the $+2$ portions of the formula. Also, to cut down on run time, it works in the java system if you % pi each of the entries before putting them into the sine and cosine functions. For whatever reason the larger the input, the more costly the operation is.

Algorithm 1 Main

```
procedure F(x)
    ab ← (a*x+b) % pi,
    cd ← (c*x+d) % pi;
    return (Math.sin(ab) + Math.cos(cd))
procedure SOLVE(p, a, b, c, d, n)
    val, h, l ← f(1); diff ← 0
    for x ∈ [2, n] do // if n = 1, do not execute
        val ← f(x)
        if val>h then // higher than current highest
            h, l ← val;
        else if val<l then // lower than current lowest
            l ← val; curDiff ← h - l;
            if curDiff>diff then diff ← curDiff;
    PRINT(p · diff)
```

1.3 Correctness

Proposition 1.
We will determine the value of largest price decline over the interval $[1, n]$*, only considering* $f(1), f(2), ..., f(n)$.

Proof.
We do this by keeping track of the largest price decline seen thus far, $diff$, the current highest point seen, $h = f(x_i)$, and the current lowest point seen, $l = f(x_j)$, such that $x_i \leq x_j$, *and* $f(x_i) \geq f(x_j)$. Therefore whenever we see a higher point,

$f(x_k)>f(x_i), x_k>x_i$, we update our $h = f(x_k)$ and reset our lowest point to be $l = h$ since we are searching for the largest decline $\implies$ l must occur after h. Now every time that we see a number $f(x_m) \leq l$, we update l and check to see if our $h-l \geq diff$, if so we update diff, else we continue to the next point. If we see a higher point than h we will repeat this process. Therefore we will be looking at each subsequent highest correspoing following lowest points $\implies$ we will see this largest price decline. □

1.4 Analysis

Proposition 2. *The space complexity of this algorithm is* ***O(1)***

Proof.

This is due to the fact that we will only store the values $p, a, b, c, d, n, anddiff$ as integer variables O(1):

Giving us a space complexity of **O(1)**

□

Proposition 3. *The time complexity of this algorithm is* ***O(N)***

Proof. This is the case because our algorithm goes through the points $1, 2, ..., n$ once and only calculates each value one time.

Giving us a time complexity of **O(N)**

□

1.5 An Example

Given the input of: 42 1 23 4 8 10, we will read this in as $p = 42, a = 1, b = 23, c = 4, d = 8, and\ n = 10$. Then we will initialize our $h = l = f(1)$ and $diff = 0$. Then starting with the second point until the 10th we will read through and record the values of what h, l, curDiff, and diff are:

x =	1	2	3	4	5	6	7	8	9	10
f(x)	-0.061724	-1.090011	1.170641	1.380555	-0.691700	0.170589	-1.115995	-1.070976	1.551270	0.359768
h	-0.061724	-0.061724	1.170641	1.380555	1.380555	1.380555	1.380555	1.380555	1.551270	1.551270
l	-0.061724	-1.090011	1.170641	1.380555	-0.691700	-0.691700	-1.115995	-1.115995	1.551270	0.359768
curDiff	–	1.028286	–	–	2.072255	–	2.496550	–	–	1.191502
diff	0	1.028286	1.028286	1.028286	2.072255	2.072255	2.496550	2.496550	2.496550	2.496550

Now multiplying our $diff$ by p $\implies$ $2.496550 \cdot 42 = 104.855110$ which is our solution.