Nokiathon 2016/2017 - przykładowe rozwiązania zadań.

Contents

Zad. 1 MinDist

```
//Author: Łukasz Ogan


import java.util.Arrays;

// you can write to stdout for debugging purposes, e.g.

// System.out.println("this is a debug message");


class Solution {

    public int solution(int[] A) {

        // write your code in Java SE 8

        int len = A.length;



        Arrays.sort(A);

        int diff = A[len-1];



        if(len>=2) {



            for (int i = 0; i < len - 1; i++) {

                if (Math.abs(A[i + 1]) - Math.abs(A[i]) < diff) {

                    diff = Math.abs(A[i + 1] - A[i]);

                }

            }

        }else{

            diff = -1;

        }
```

```
        return diff;



    }

}
```

Zad. 2 Pow11

// Author: Arkadiusz Karwasz

```
import java.math.BigInteger;

class Solution {

    public int solution(int N) {

        // write your code in Java SE 8

        BigInteger tmp;



        tmp = new BigInteger("11");

        tmp = tmp.pow(N);



        int result = 0;

        char ch;



        String number = tmp.toString();



        for(int i = 0; i < number.length(); i++){

            ch = number.charAt(i);

            if(ch == '1'){

                result++;

            }

        }



        return result;

    }

}
```

Zad. 3 MonkeyRiver

```
// Author: Szymon Bogusławski


#include <algorithm>

#include <vector>

#include <set>

// you can write to stdout for debugging purposes, e.g.

// cout << "this is a debug message" << endl;



struct Stone {

    int time;

    int index;

    Stone(int t, int i) : time(t), index(i) {}

};



bool searchPath(set<int>& s, int& startIndex, int maxHop) {

    auto it = s.upper_bound(startIndex + maxHop);

    if (it == s.end()) return true;

    it = prev(it);

    if (*it == startIndex) return false;

    startIndex = *it;

    return searchPath(s, startIndex, maxHop);

}



int solution(vector<int> &A, int D) {

    // write your code in C++14 (g++ 6.2.0)
```

```
    set<int> activeStones { -1, (int) A.size() };
    vector<Stone> stones;
    for (int i = A.size() -1; i >= 0; i--) {
        if (A[i] != -1) stones.push_back(Stone(A[i], i));
    }
    sort(stones.begin(), stones.end(), [](const Stone& a, const Stone& b) -> bool {
        return a.time < b.time;
    });
    auto nextStone = stones.begin();
    int startIndex = -1;
    while (!searchPath(activeStones, startIndex, D)) {
        while (nextStone != stones.end() && (*nextStone).index < startIndex) {
            nextStone++;
        }
        if (nextStone != stones.end()) {
            activeStones.insert((*nextStone++).index);
        }
        else return -1;
    }
    return (nextStone == stones.begin() ? 0 : (*prev(nextStone)).time);
}
```