codility

Check out Codility training tasks

Candidate Report: Anonymous

Test Name:

Summary Timeline

Test Score

62 out of 100 points

62%

Tasks in Test

	Time Spent	Task Score
GenomicRangeQuery Submitted in: JavaScript	1 min	62%

TASKS DETAILS

MEDIUM

1. GenomicRangeQuery
Find the minimal nucleotide from a range of sequence DNA.

Task Score	Correctness	Performance
62%	100%	0%

Task description

Solution

A DNA sequence can be represented as a string consisting of the letters A, C, G and T, which correspond to the types of successive nucleotides in the sequence. Each nucleotide has an *impact factor*, which is an integer. Nucleotides of types A, C, G and T have impact factors of 1, 2, 3 and 4, respectively. You are going to answer several queries of the form: What is the minimal impact factor of nucleotides contained in a particular part of the given DNA sequence?

The DNA sequence is given as a non-empty string S = S[0]S[1]...S[N-1] consisting of N characters. There are M queries, which are given in non-empty arrays P and Q, each consisting of M integers. The K-th query (0 ≤ K < M) requires you to find the minimal impact factor of nucleotides contained in the DNA sequence between positions P[K] and Q[K] (inclusive).

For example, consider string S = CAGCCTA and arrays P, Q such that:

```
P[0] = 2    Q[0] = 4
P[1] = 5    Q[1] = 5
P[2] = 0    Q[2] = 6
```

The answers to these M = 3 queries are as follows:

- The part of the DNA between positions 2 and 4 contains nucleotides G and C (twice), whose impact factors are 3 and 2 respectively, so the answer is 2.
- The part between positions 5 and 5 contains a single nucleotide T, whose impact factor is 4, so the answer is 4.
- The part between positions 0 and 6 (the whole string) contains all nucleotides, in particular nucleotide A whose impact factor is 1, so the answer is 1.

Write a function:

```
function solution(S, P, Q);
```

that, given a non-empty string S consisting of N characters and two non-empty arrays P and Q consisting of M integers, returns an array consisting of M integers specifying the consecutive answers to all queries.

Result array should be returned as an array of integers.

For example, given the string S = CAGCCTA and arrays P, Q such that:

Programming language used:	JavaScript
Total time used:	1 minutes
Effective time used:	1 minutes
Notes:	*not defined yet*

Task timeline

22:12:08 22:12:33

Code: 22:12:32 UTC, js, final, score: **62** show code in pop-up

```
1  const ensureValid = (S, P, Q) => {
2      if (!S || !P || !Q || P.length != Q.length) {
3          throw error("Data is invalid");
4      }
5  }
6
7  const getImpactFactor = (s) => {
8      if (s === 'A') {
9          return 1;
10     } else if (s === 'C') {
11         return 2;
12     } else if (s === 'G') {
13         return 3;
14     } else if (s === "T") {
15         return 4
16     } else {
17         return null;
18     }
```

```
P[0] = 2    Q[0] = 4
P[1] = 5    Q[1] = 5
P[2] = 0    Q[2] = 6
```

the function should return the values [2, 4, 1], as explained above.

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

- N is an integer within the range [1..100,000];
- M is an integer within the range [1..50,000];
- each element of arrays P, Q is an integer within the range [0..N − 1];
- P[K] ≤ Q[K], where 0 ≤ K < M;
- string S consists only of upper-case English letters `A, C, G, T`.


```
}

var solution = function solution(S, P, Q) {

    const result = [];
    const minImpactFactor = 1;

    ensureValid(S, P, Q);

    for (var i = 0; i < P.length; i++) {
        var substr = S.substring(P[i], Q[i] + 1);

        let impactFactor = null;

        for (let j = 0; j < substr.length; j++) {
            const element = substr.substring(j, j+1);

            const currentImpactFactor = getImpactFactor(element);
            if (currentImpactFactor === null) {
                throw error("Data is invalid");
            }

            if (impactFactor == null || impactFactor > currentImpactF
                impactFactor = currentImpactFactor;
            }

            if (impactFactor === minImpactFactor) {
                break;
            }
        }

        result.push(impactFactor);
    }


    return result;
}
```

Analysis summary

The following issues have been detected: timeout errors.

Analysis

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

expand all	Example tests	
▶	example example test	✓ OK
expand all	Correctness tests	
▶	extreme_sinlge single character string	✓ OK
▶	extreme_double double character string	✓ OK
▶	simple simple tests	✓ OK
▶	small_length_string small length simple string	✓ OK
▶	small_random small random string, length = ~300	✓ OK
collapse all	Performance tests	
▼	almost_all_same_letters GGGGGG..??..GGGGGG..??..GGGGGG	✗ TIMEOUT ERROR Killed. Hard limit reached: 6.000 sec.
1.	6.000 s	**TIMEOUT ERROR**, Killed. Hard limit reached: 6.000 sec.
2.	0.084 s	**OK**
▼	large_random large random string, length	✗ TIMEOUT ERROR Killed. Hard limit reached: 6.000 sec.

1.	6.000 s	**TIMEOUT ERROR**, Killed. Hard limit reached: 6.000 sec.
▼	extreme_large all max ranges	✗ **TIMEOUT ERROR** Killed. Hard limit reached: 6.000 sec.
1.	6.000 s	**TIMEOUT ERROR**, Killed. Hard limit reached: 6.000 sec.
