Candidate Report: Anonymous

Test Name:

Summary | Timeline

Test Score

100 out of 100 points

100%

Tasks in Test

	Time Spent	Task Score
GenomicRangeQuery Submitted in: Java	17 min	100%

TASKS DETAILS

MEDIUM

1. GenomicRangeQuery

Find the minimal nucleotide from a range of sequence DNA.

Task Score	Correctness	Performance
100%	100%	100%

Task description

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:

```
class Solution { public int[] solution(String S, int[] P, int[]
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:

```
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`.

Solution

Programming language used:	Java
Total time used:	17 minutes
Effective time used:	17 minutes
Notes:	*not defined yet*

Task timeline

21:39:16 21:55:36

Code: 21:55:36 UTC, java, final, score: 100

show code in pop-up

```java
import java.util.*;

class Solution {
    public int[] solution(String S, int[] P, int[] Q) {
        char[] cA = S.toCharArray();
        int[] pSumA = getPSum('A', cA);
        int[] pSumC = getPSum('C', cA);
        int[] pSumG = getPSum('G', cA);

        int start, end;
        int[] result = new int[P.length];
        for (int i = 0; i < P.length; i++) {
            start = P[i];
            end = Q[i] + 1;

            if (pSumA[end] - pSumA[start] > 0) result[i] = 1;
            else if (pSumC[end] - pSumC[start] > 0) result[i] = 2;
            else if (pSumG[end] - pSumG[start] > 0) result[i] = 3;
            else result[i] = 4;
        }

        return result;
    }

    private int[] getPSum(char c, char[] cA) {
        int[] pSum = new int[cA.length + 1];

        if (cA[0] == c) {
            pSum[0] = 1;
        } else {
            pSum[0] = 0;
        }

        for (int i = 0; i < cA.length; i++) {
            if (cA[i] == c) {
                pSum[i + 1] = pSum[i] + 1;
            } else {
                pSum[i + 1] = pSum[i];
            }
        }

        return pSum;
    }
}
```

Analysis summary

The solution obtained perfect score.

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
expand all	**Performance tests**	
▶	almost_all_same_letters GGGGGG..??..GGGGGG..??..GGGGGG	✔ OK
▶	large_random large random string, length	✔ OK

▶ extreme_large all max ranges	✔ **OK**