Codility

# CodeCheck Report: trainingFS9J46-2AN

Test Name:

Check out Codility training tasks

Summary | Timeline

## Tasks summary

| Task | Time spent | Score |
|---|---|---|
| GenomicRangeQuery<br>Python | 6 min | 100% |

## Total score

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:

```
def 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:

```
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 and 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: | Python |
| Total time used: | 6 minutes |
| Effective time used: | 6 minutes |
| Notes: | *not defined yet* |

### Task timeline

01:47:43 01:53:34

Code: 01:53:33 UTC, py, final, score: 100

show code in pop-up

```python
# you can write to stdout for debugging purposes, e.g.
# print("this is a debug message")

def solution(S, P, Q):
    # write your code in Python 3.6
    M = len(P)
    if M != len(Q):
        return -1
    N = len(S)
    dict_ = {
        'A' : 1,
        'C' : 2,
        'G' : 3,
        'T' : 4
        }

    arr=[]
    for el in S:
        arr.append(dict_[el])
    global_minimum = min(arr)

    if N == 1:
        return ([arr[0]] * M)
    else:
        dict1 = {}
        i = 0
        for el in P:
            if el not in dict1.keys():
                dict1[el] = []
            dict1[el].append(i)
            i += 1
        dict2 = {}
        i = 0
        for el in Q:
            if el not in dict2.keys():
                dict2[el] = []
            dict2[el].append(i)
            i += 1

        min_ = max(arr)+1
        min_before_reset = min_
        dict_ = {}
        queries = {}
        for i in range(N):
            min_ = min(min_, arr[i])

            if i in dict1.keys() and i in dict2.keys():
                list_ = []
                for el1 in dict1[i]:
                    for el2 in dict2[i]:
                        if el1 == el2:
                            queries[el1] = arr[i]
                            list_.append(el1)
                list1 = []
                for el1 in dict1[i]:
                    if el1 not in list_:
                        list1.append(el1)
                dict1[i] = list1.copy()
                list2 = []
                for el2 in dict2[i]:
                    if el2 not in list_:
                        list2.append(el2)
                dict2[i] = list2.copy()

            if i in dict2.keys():
                list_ = []
                for el in dict2[i]:
                    if el in dict_.keys():
                        dict_[el] = min(min_, dict_[el])
                        queries[el] = dict_[el]
                        list_.append(el)
                for el in list_:
                    dict_.pop(el)

            if i in dict1.keys():
                if min_before_reset != min_:
                    for key in dict_.keys():
                        dict_[key] = min(min_, dict_[key])
                for el in dict1[i]:
                    if arr[i] == global_minimum:
                        queries[el] = global_minimum
                    else:
                        dict_[el] = arr[i]
                        min_before_reset = min_
                        min_ = arr[i]

            if min_ == global_minimum:
                for key in dict_.keys():
                    queries[key] = global_minimum
                dict_ = ({}).copy()

        result = [ queries[i] for i in range(M)]

        return result
```

### Analysis summary

The solution obtained perfect score.

### Analysis

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

| expand all | Example tests | |
|---|---|---|
| ▶ | example<br>example test | ✔ OK |
| **expand all** | **Correctness tests** | |
| ▶ | extreme_sinlge<br>single character string | ✔ OK |
| ▶ | extreme_double<br>double character string | ✔ OK |
| ▶ | simple<br>simple tests | ✔ OK |
| ▶ | small_length_string<br>small length simple string | ✔ OK |
| ▶ | small_random<br>small random string, length = ~300 | ✔ OK |
| **expand all** | **Performance tests** | |
| ▶ | almost_all_same_letters<br>GGGGGG..??..GGGGGG..??..GGGGGG | ✔ OK |
| ▶ | large_random<br>large random string, length | ✔ OK |
| ▶ | extreme_large<br>all max ranges | ✔ OK |