Kick Start 2021 - Round B

Analysis: Increasing Substring

Test set 1

A string X of length L is strictly increasing, if for every pair of indices i and j such that $1 \leq i < j \leq L$ (1-based), the character at position i is smaller than the character at position j. Using this, we can say that $X_i < X_{i+1}$ for $1 \leq i < L$. We can check this in $O(L)$ time complexity for string X.

There are $O(\mathbf{N}^2)$ substrings of string $\mathbf{S}$ each of which can have length at most $\mathbf{N}$. We can iterate over each substring and check whether it is strictly increasing in $O(\mathbf{N})$ time. If a substring from i to j is strictly increasing, update the length of longest strictly increasing substring ending at index j to $j - i + 1$ if it is greater than the previous found length. The overall time complexity of the solution is $O(\mathbf{N}^3)$.

Consider the following example with string $bbcd$. There is only 1 substring (b) ending at index 1. Hence, the answer for index 1 is 1. There are 2 substrings (b and bb) which end at index 2. bb is not strictly increasing string. Hence, the answer for index 2 is 1. There are 3 substrings (bbc, bc, and c) ending at index 3. bbc is not strictly increasing as b is repeated twice. bc is the longest strictly increasing substring ending at index 3. Hence, answer for index 3 is 2. There are 4 substrings ($bbcd$, bcd, cd, and d) ending at index 4. $bbcd$ is not strictly increasing as b is repeated twice. bcd is the longest strictly increasing substring ending at index 4. Hence, answer for index 4 is 3.

Sample Code(C++)

```
vector<int> longestStrictlyIncreasingSubstring(string S) {
  vector<int> answer(S.size(), 0);
  for(int i = 0; i < S.size(); i++) {
    for(int j = i; j < S.size(); j++) {
      bool is_strictly_increasing = true;
      for(int k = i; k < j; k++) {
        is_strictly_increasing &= (S[k] < S[k+1]);
      }
      if(is_strictly_increasing) {
        answer[j] = max(answer[j], j - i + 1);
      }
    }
  }
  return answer;
}
```

Test set 2

We cannot check each substring of string $\mathbf{S}$ for this test set due to the large constraints. We already know that, for a string $\mathbf{S}$ to be strictly increasing, $\mathbf{S_i} < \mathbf{S_{i+1}}$ for $1 \leq i < \mathbf{N}$. Consider that we have already calculated the length of the longest strictly increasing substring that ends at position i. Let this length be $MaxLen(i)$. Now we need to compute the answer for position $i + 1$. There is no need to consider all substrings ending at position $i + 1$. We can simply check if $\mathbf{S_i} < \mathbf{S_{i+1}}$. If this condition is satisfied, we can simply append S(i+1) to the longest increasing

substring ending at i and it would still be increasing and update $MaxLen(i+1) = MaxLen(i) + 1$. Otherwise, $MaxLen(i) = 1$. This way, we can calculate the length of the longest strictly increasing substring that ends at position i in constant time. Hence, the overall time complexity of the solution is $O(\mathbf{N})$.

Consider the following example with string $bbcda$. For index 1, $MaxLen(1) = 1$. For index 2, we can see that index 1 and index 2 have equal values and thus do not satisfy the constraint $\mathbf{S_i} < \mathbf{S_{i+1}}$. In this case, $MaxLen(2) = 1$. For index 3, $\mathbf{S_2} < \mathbf{S_3}$, hence we can extend the longest strictly increasing substring ending at index $\mathbf{2}$. In this case, $MaxLen(3) = 2$. For index 4, $\mathbf{S_3} < \mathbf{S_4}$, hence we can extend the longest strictly increasing substring ending at index $\mathbf{3}$. In this case, $MaxLen(4) = 3$. For index 4, $\mathbf{S_4} > \mathbf{S_5}$ which violates the condition for strictly increasing substring. In this case, $MaxLen(5) = 1$.

Sample Code(C++)

```
vector<int> longestStrictlyIncreasingSubstring(string S) {
  vector<int> answer(S.size(), 0);
  answer[0] = 1;
  for(int i = 1; i < S.size(); i++) {
      if(S[i - 1] < S[i]) {
        answer[i] = answer[i - 1] + 1;
      }
      else {
        answer[i] = 1;
      }
  }
  return answer;
}
```