Stringonomics

by abizerl123

You are given a string S consisting of lowercase English letters.

You are given another string P, that may or may not exist as a substring in S.

Given Q queries, where each query is of the form X and C, where X is a non-negative integer and C a character. For every query, you are supposed to change the index X (Assume 0 based indexing) of the string S to the character C.

You are supposed to find the minimum number of queries, when sequentially executed from the start, after which the string P no longer exists as a substring in S.

If the string P exists in S even after executing all the queries, print -1.

It is guaranteed that each index is only changed once, and once the string P ceases to exist in S, it would not reappear again later (If P never existed in S, it wou

Input Format

First line contains T number of testcases. For each testcase:

- The first line contains the string S.
- Next line contains the string P.
- Next line contains number of queries Q.
- Following Q lines contains X and C, space seperated.

Constraints

- $1 \le T \le 50$
- $1 \le |S| \le 2 \cdot 10^5$
- $1 \le |P| \le |S|$
- $1 \le Q \le |S|$
- $0 \le X < |S|$
- C is a lowercase English letter
- Sum of $|S|$ and $|P|$ over all T $\le 7 \cdot 10^5$
- Sum of Q over all T $\le 7 \cdot 10^5$

Output Format

For each testcase T, output a single integer denoting the minimum number of queries after which the string P ceases to exist in string S.

Sample Input 0

```
2
abcde
bc
3
0 p
1 q
2 w
abcde
cde
2
0 t
1 z
```

Sample Output 0

```
2
-1
```

Explanation 0

First test case

- Initial - *a[bc]de*
- First update - *p[bc]de*
- Second update - *pqcde*
  So after second update the string `P` is not in string `S`, so the answer is *2.*

Second test case

- Initial - *ab[cde]*
- First update - *tb[cde]*
- Second update - *tz[cde]*
  Even, after all the updates the string `P` is in the string `S`, so the answer is *-1.*

Current Buffer (saved locally, editable)

C++14

```cpp
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

bool precomputed = false;
vector<int> preprocessed_info;
set<int> indexes;
void precompute(string &pattern)
{
    int i=1,len = 0;
    preprocessed_info[0] = 0;
    while(i<pattern.size())
    {
```

```
        if(pattern[i]==pattern[len])
        {
            preprocessed_info[i++] = len++;
        }
        else
        {
            if(len!=0)
                len = preprocessed_info[len-1];
            else
                preprocessed_info[i++]=0;
        }
    }

}

void KMP(string &txt,string &pat)
{
    preprocessed_info.resize(pat.length(),0);
    precomputed = true;
    precompute(pat);
    indexes.clear();
    int i=0,j=0;
    while(i<txt.size())
    {
        if(pat[j]==txt[i])
            i++,j++;
        if(j==pat.length())
        {
            indexes.insert(i-1);
            j = preprocessed_info[j-1];
        }
        else if(i<txt.length() && pat[j]!=txt[i])
        {
            if(j!=0)
                j = preprocessed_info[j-1];
            else
                i++;
        }
    }
}

int main()
{
    string t_temp;
    getline(cin, t_temp);

    int t = stoi(ltrim(rtrim(t_temp)));

    for (int t_itr = 0; t_itr < t; t_itr++) {
        string s;
        precomputed = false;
        getline(cin, s);

        string p;
        getline(cin, p);

        string q_temp;
        getline(cin, q_temp);

        KMP(s,p);

        int q = stoi(ltrim(rtrim(q_temp)));
        bool isPresent =  true;
        int count = 0;

        for (int q_itr = 0; q_itr < q; q_itr++) {

            string first_multiple_input_temp;
            getline(cin, first_multiple_input_temp);
            if(!isPresent)
            {
                count++;
                continue;
```

```
            }
            vector<string> first_multiple_input = split(rtrim(first_multiple_input_temp));

            int x = stoi(first_multiple_input[0]);

            char c = first_multiple_input[1][0];

            if(s[x]==c)
                continue;
            else
            {
                s[x] = c;
                auto it = indexes.lower_bound(x);
                if(it==indexes.end() || (*it)-p.length()+1>x)
                    continue;
                auto it2 = it;
                while((*it2)-p.length()+1<=x && it2!=indexes.end())
                {
                    it2++;
                }
                indexes.erase(it,it2);
                s[x] = c;
            }

            if(indexes.size()==0)
            {
                KMP(s,p);
                if(indexes.size()==0)
                    isPresent = false;
            }
            if(!isPresent)
            {
                count++;
                continue;
            }

        }
        cout<<((count>0)?count:-1)<<endl;
    }

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));
```

```

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}

```

Line: 58 Col: 1

Upload Code as File Test against custom input Run Code Submit Code