SPOJ AMR12I

Saruman of Many Colours

Prof. Edson Alves – UnB/FGA

Problema

"For I am Saruman the Wise, Saruman Ring-maker, Saruman of Many Colours!"

'I looked then and saw that his robes, which had seemed white, were not so, but were woven of all colours. And if he moved they shimmered and changed hue so that the eye was bewildered.' - Gandalf the Grey.

And so it was that Saruman decided to brand his Uruk-hai army with the many colours that he fancied. His method of branding his army was as follows.

He straps his army of N Uruk-hai onto chairs on a conveyor belt. This conveyor belt passes through his colouring-room, and can be moved forward or backward. The Uruk-hai are numbered 0 to $N-1$ according to the order in which they are seated. Saruman wishes that the i'th Uruk-hai be coloured with the colour $c[i]$.

Problema

Further, his colouring-room has space for exactly K chairs. Once the chosen K consecutive Uruk-hai are put into the room, a colour jet sprays all K of them with any fixed colour. The conveyor belt is not circular (which means that the $(N-1)$'th and the 0'th Uruk-hai are not consecutive). Note that Uruk-hai can be recoloured in this process.

Saruman wants to find out what is the minimum number of times that the jet needs to be used in order to colour his army in the required fashion. If it is not possible to colour the army in the required fashion, output -1.

Entrada e saída

Input

The first line contains the number of test-cases T.

Each test case consists of 2 lines. The first line contains two space-separated integers, N and K.

This is followed by a single like containing a string of length N, describing the colours of the army. The i'th character of the string denotes the colour of the i'th Uruk-hai in the army.

Entrada e saída

Output

Output T lines, one for each test case containing the answer for the corresponding test case. Remember if it is not possible to colour the army as required, output -1.

Constraints

$1 \leq T \leq 50$

$1 \leq K \leq N \leq 20,000$

The string c has length exactly N and contains only the characters `'a'`, ..., `'z'`.

Exemplo de entradas e saídas

Sample Input

```
2
3 2
rgg
3 3
rgg
```

Sample Output

```
2
-1
```

Solução com complexidade $O(N)$

- Este problema pode ser resolvido por meio de um algoritmo guloso, embora a identificação das escolhas e das restrições não sejam triviais
- Como cada processo de pintura modifica a cor de K Uruk-hais, a última pintura gerará um grupo de K soldados de mesma cor
- Assim, se o arranjo desejado não tiver um grupo com, no mínimo, K soldados vizinhos de mesma cor, não há solução
- Existindo tal grupo, é possível modificar a cor dos demais de forma gulosa, em duas etapas: à esquerda do grupo e à direita do grupo de K soldados de mesma cor
- A partir da esquerda, modifica-se a cor do soldado na posição i até $K - 1$ vizinhos à sua direita, desde que tenham todos a mesma cor
- O mesmo processo pode ser feito a partir da direita, em sentido contrário
- Como cada soldados será avaliado, no máximo, duas vezes, este algoritmo tem complexidade $O(N)$

Solução AC com complexidade $O(N)$

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

using namespace std;

int solve(int N, int K, const string& c)
{
    int ans = 0, L = 0, len = 0;

    // Pinta de forma gulosa, em sequências de tamanho no máximo k
    while (L < N)
    {
        // Pinta na posição L
        ++ans;

        // No intervalo [L, R) todos tem mesma cor
        // Este intervalo deve ter no máximo K elementos
        int R = L + 1;

        while (R < N and R - L < K and c[R] == c[L])
            ++R;
```

Solução AC com complexidade $O(N)$

```
22          len = max(len, R - L);
23          L = R;
24      }
25
26      // Deve existir ao menos uma sequência de tamanho K
27      // (a última pincelada cria uma sequência de tamanho K)
28      if (len < K)
29          return -1;
30
31      return ans;
32  }
33
34  int main()
35  {
36      ios::sync_with_stdio(false);
37
38      int T;
39      cin >> T;
```

Solução AC com complexidade $O(N)$

```
41      while (T--)
42      {
43          int N, K;
44          cin >> N >> K;
45
46          string c;
47          cin >> c;
48
49          cout << solve(N, K, c) << '\n';
50      }
51
52      return 0;
53  }
```