백준 16472번 - 고냥이
문제
입력
출력
예제 입력1
예제 출력1
출처
알고리즘 분류
접근 방법
소스코드

백준 16472번 - 고냥이

시간제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1초	512MB	1285	553	417	42.464%

문제

고양이는 너무 귀엽다. 사람들은 고양이를 너무 귀여워했고, 결국 고양이와 더욱 가까워지고 싶어 고양이와의 소통을 위한 고양이 말 번역기를 발명하기로 했다. 이 번역기는 사람의 언어를 고양이의 언어로, 고양이의 언어를 사람의 언어로 바꾸어 주는 희대의 발명품이 될 것이다.

현재 고양이말 번역기의 베타버전이 나왔다. 그러나 이 베타버전은 완전 엉망진창이다. 베타버전의 번역기는 문자열을 주면 그 중에서 최대 N개의 종류의 알파벳을 가진 연속된 문자열밖에 인식하지 못한다. 굉장히 별로지만 그나마 이게 최선이라고 사람들은 생각했다. 그리고 문자열이 주어졌을 때 이 번역기가 인식할 수 있는 최대 문자열의 길이는 얼마인지가 궁금해졌다.

고양이와 소통할 수 있도록 우리도 함께 노력해보자.

입력

첫째 줄에는 인식할 수 있는 알파벳의 종류의 최대 개수 N이 입력된다. (1 < N ≤ 26)

둘째 줄에는 문자열이 주어진다. (1 ≤ 문자열의 길이 ≤ 100,000) 단, 문자열에는 알파벳 소문자만이 포함된다.

출력

첫째 줄에 번역기가 인식할 수 있는 문자열의 최대길이를 출력한다.

예제 입력1

```
2
abbcaccba
```

예제 출력1

```
4
```

출처

출처

알고리즘 분류

- 이분 탐색
- 투 포인터

접근 방법

가장 먼저 생각할 수 있는 Naive한 방식은 모든 substring에 대하여 조사를 하는 방법이다. 해당 방식은 $O(N^2)$의 시간복잡도를 갖게 될 것이라는 것을 어렵지 않게 알 수 있다. N이 최대 10^5의 값을 가지므로 이 방법으로는 문제를 해결할 수 없다는 것을 알 수 있다.

불필요한 탐색을 제거해보자.

설명의 편의성을 위해 문자열에 포함된 각각의 문자를 $c_1c_2c_3...c_n$으로 표현해보자.

현재 탐색에서 $P_1 : [s_i, s_{i+l})$까지의 문자열이 현재까지의 최적해라면, 다음 탐색에서는 길이 l 이하의 substring에 대한 불필요한 조사를 할 필요가 없다.

즉, 다음 탐색 구간 $P_2 : [s_j, s_{j+l'})$에 대하여 $l \leq l'$인 l' 에 대한 조사를 진행하면 된다.

ex)

문제에서 탐색 순서는 중요하지 않으므로, 탐색 구간 $[s_i, s_{i+l})$ 에 대하여 i 를 1씩 증가시키면서 탐색한다.

이 후, **N개 이하의 종류의 알파벳을 가진 연속된 문자열**을 만날 때 마다 길이 l 을 증가시킨다.

$0 \leq i < k$ 범위에서 i가 증가하므로, 문제의 최적해를 무조건 찾을 수 있다.

+ 시간복잡도

반복문의 총 반복 횟수는 최악의 경우 문자열의 길이 K 이며, 내부에서 substring을 구하기 위한 반복 횟수는 반복문의 총 반복 횟수와 반비례 관계를 가지는 Weighted Sum 으로 표현된다.

수행 시간 $T(n)$ 은 $N \leq T(n) < N + \alpha$ 관계를 만족하므로 시간복잡도는 $O(N)$으로 볼 수 있다.

소스코드

```
#define FASTIO cin.tie(0)->sync_with_stdio(false), cout.tie(0)
#include <bits/stdc++.h>
```

```cpp
using namespace std;
int main(void){
    FASTIO;
////////////////////////////////////////////////////////////////
    int N;
    cin >> N;
    string s;
    cin >> s;

    int l = 1; // [i, i+l)
    int size = s.size();
    for(int i=0; i+l <= size; i++){
        string sub = s.substr(i, l);
        set<char> set1;
        for(auto item : sub) set1.insert(item);
        int j = i+l;
        while(j < size){
            set1.insert(s[j]);
            if(set1.size() > N)
                break;
            j++, l++;
        }
    }
    cout << l ;
    return 0;
}
```