- 문제.

· Input : 문자열의 길이 n, 찾을 부분 문자열의 길이 K.
($1 \le K \le n \le 2 \cdot 10^5$).
원래 문자열 S (길이 n).

· Output : K의 최대 개수.

· S의 부분문자열 sub가 하나의 문자 (소문자)로만 구성될 때
($S[i \cdots j]$ ($1 \le i \le j \le n$).)
길이가 K인 sub가 최대 몇개 존재하는지 찾는 문제.

- 접근

· 처음에는 S에 대해 $S.substr(i, K)$ (선형의 인덱스에서 시작하는)가 같은 문자로만 구성되었는지 검사하고 (애초에 두개씩 비교하기 때문에 K가 1이면 같은문자인걸로 됨).

맞다면 $[i, n-K]$ 범위에 몇개의 sub가 존재하는지 무식하게 count.

⇒ 시간초과. ($O(n^2)$)

ⓛ 20000 이고
nnn.... i..... K..... 에서
이미 $i=0$ 에서 X이면 $i=1$ ... 에서도 X인데
중복이 너무 많음.

② 하나의 문자로만 이루어진 S에 대해서도
일치하더라도 $O(n^2)$의 시간이 걸림.

· 그러면 어떻게?

① 이미 길이 K인 1개의 sub조차 발견되지 않았다면
달라진 알파벳의 인덱스로 건너뛸 필요.

② 각 알파벳만으로 이루어진 길이 K의 sub가 몇개인지
cnt[26]에 ++하면서 세어주면 됨. → 그냥 연속된 같은문자 수 세어줄다음 K로 나누면 되고 그값을 cnt[a]에 +=해

수번 되었네

⇒ O(n)으로 구현가능.

— 구현

① 문자열 ㅈㅈㅈㅈ가 입력되어도 char형 변수에 cin으로 받으면 1문자씩 받을 수 있다. scanf("%c") 처럼 엔터키 안먹음 ㄱㅊ.

② 문자열 s[0]을 비워두면 (s[i]==s[i-1]) 비교하기 편하다.

③ 알파벳 별로 다뤄야하기 때문에 int 배열에 정수로 저장하면 연산이 빠르다.

④ 직전 인덱스와 같으면 (연속된 알파벳 수) now++ (처음 1개는 무조건 존재하기에 now=1).

→ 직전인덱스와 다르다면 (최초에는 여기서 시작하겠지). (달라졌다면).

now(연속된 수) / k(부분 길이)을 ~~해당 알파벳~~ 직전 알파벳 cnt에 더해줌.

ex) 8 2

aa z aaaa z → 3 출력.

2 ↓ +1 4 ↓ +2.

※ 달라졌을 때만 cnt에 더해주므로 직전 인덱스에 위치하는

반복문 끝나고 or 반복문 내에서

↳ s[i]에 대해 cnt에 now/k 더해줌

↳ n까지 돌림.

⑤ 마지막으로 cnt[i] 최대값 찾음.