

## 고등부 1. 줄임말

풀이 작성자: 김준원

$S$ 의 길이는  $N$ ,  $T$ 의 길이는  $M$ 이라고 하자.

### 부분문제 1 ( $S$ 와 $T$ 가 'a'만으로 이루어져 있다)

모든 문자가 'a'이기 때문에,  $S$ 는 'a'가  $N$ 개,  $T^k$ 는 'a'가  $kM$ 개 있는 문자열이다. 언제  $S$ 가  $T^k$ 의 줄임말이 될까? 모든 문자가 'a'이므로,  $T^k$ 의 길이가  $S$ 보다 길거나 같으면 줄임말이 된다. 이제,  $kM \geq N$ 인 최소의  $k$ 를 찾는 문제가 된다. 답은  $\frac{N}{M}$ 을 소숫점 아래에서 올림한 값이 된다. C언어 코드로는  $(N + M - 1) / M$ 과 같이 쓸 수 있다.

### 부분문제 2 ( $N \leq 100, M \leq 100$ )

본 문제를 풀기 위해서 조금 더 쉬운 다음 문제를 생각한다. '두 문자열  $A$ 와  $B$ 가 주어졌을 때,  $B$ 가  $A$ 의 줄임말인가?' 아래는 이 문제를  $O(|A| + |B|)$ 에 해결하는 알고리즘이다. ( $|A|$ 는  $A$ 의 길이,  $|B|$ 는  $B$ 의 길이)

1.  $A$ 의 가장 왼쪽에서부터  $B$ 의 첫번째 글자를 찾는다.
2. 위에서 찾은 인덱스 다음부터  $B$ 의 두번째 글자를 찾는다.
3. 위에서 찾은 인덱스 다음부터  $B$ 의 세번째 글자를 찾는다.
4. ...
5. 위 과정을 끝까지 진행할 수 있으면(즉,  $B$ 의 마지막 글자까지 모든 글자를  $A$ 에서 찾을 수 있으면)  $B$ 는  $A$ 의 줄임말이다. 그렇지 않고 중간에 실패하면,  $B$ 는  $A$ 의 줄임말이 아니다.

위 알고리즘은 문자열  $A$ 에서의 위치와, 문자열  $B$ 에서의 위치를 동시에 관리한다는 의미에서 '투 포인터'(Two pointers) 알고리즘이라고 부르기도 한다. 이 알고리즘을 이해했다면, 바로 부분문제 2를 풀 수 있게 된다.  $S$ 가  $T$ 의 줄임말인지,  $T^2$ 의 줄임말인지,  $T^3$ 의 줄임말인지, ... 이를 차례대로 줄임말이 될 때까지 판별한다. 줄임말이 되는 것이 불가능하지 않은 이상 답은 언제나  $N$  이하이고, 따라서 만약  $T^1, \dots, T^N$ 까지 검사한 결과  $S$ 가 어떤 것의 줄임말도 아니라면, -1을 출력한다. 시간 복잡도는  $O(N^2M)$ 이다.

### 부분문제 3 ( $N \leq 10\,000, M \leq 100$ )

부분문제 2의 풀이는 동일한 작업을 공연히  $N$ 번 반복한다. 위 알고리즘을 단 한 번만 수행하는 최적화를 수행할 수 있다. 문자열  $T$ 의 처음과 끝을 이어붙이자. 즉,  $S$ 가  $T$ 의 줄임말인지 판별하는 도중  $T$ 의 끝에 도달하더라도 다시  $T$ 의 처음으로 돌아가서 알고리즘을 계속 진행한다. 문제의 답은 '새로운 알고리즘이 끝날 때까지 문자열  $T$ 를 몇 바퀴 돌았는가'로 쓸 수 있다. 또한, 만약  $S$ 의 어떤 글자에서  $T$ 를 한 바퀴 돌았음에도 답을 얻지 못했다면, -1을 출력해야 한다. 이 때의 시간 복잡도는  $O(NM)$ 이다.

### 부분문제 4 ( $M \leq 1\,000$ )

더 빠른 시간에 작동하는 알고리즘을 만들려면, 이제 'T를 한 칸씩 도는 시간'을 절약해야 한다.  $T$ 에서의 각 인덱스  $i$ 와 각 알파벳 문자  $c$ 에 대해, 다음과 같은 값을 기록(전처리)한 표(2차원 배열)을 만든다.

`next[i][c]`:  $T$ 에서  $i$ 번째 또는 그 이후에 첫번째로 등장하는 문자  $c$ 의 인덱스

위와 마찬가지로,  $next[i][c]$ 를 구할 때  $T$ 의 처음과 끝을 이어붙여 생각하자. 이와 같은 표를 한 번 만들고 나면, 앞선 알고리즘에서 ‘ $T$ 의 다음 인덱스를 찾는 작업’을 한 칸씩 이동하면서 하지 않아도 된다. 가야 할 ‘다음 인덱스’의 번호는 이미 우리가 만들어둔 표에 적혀 있기 때문이다. 그래서, 이제는 표를 만드는 시간을 제외하고 문제를  $O(N)$ 에 풀 수 있게 된다.

$next$  배열은 가장 쉬운 방법으로 만들면  $O(M^2)$ 에 만들 수 있고, 이 때에 부분문제 4를  $O(N + M^2)$ 의 시간 복잡도로 해결할 수 있게 된다.

## 부분문제 5

$next$  배열을 더 빠르게 채우면 부분문제 5를 풀 수 있다. 아이디어는 다음과 같다.

알파벳  $c$ 를 고정하고,  $next[1][c], next[2][c], \dots, next[N][c]$ 를 모두 채우는 방법을 생각한다. 먼저,  $T$ 에서 알파벳  $c$ 가 등장하는 인덱스들을 모두 기록해두고, 정렬된 배열로 만들어두자. 그러면  $next[i][c]$ 는 방금 만든 배열에서  $i$  이상인 최소 원소가 된다. 이 사실을 알면, 이분 탐색(lower\_bound 등의 함수를 이용할 수 있다)을 이용해  $next[1][c], \dots, next[N][c]$ 를  $O(M \log M)$ 에 채우거나, 투 포인터 알고리즘을 이용해  $O(M)$ 에 채울 수 있다.

위 방법을 이용해  $next$  배열을 채우면 전체 문제를  $O(N + \Omega M \log M)$  또는  $O(N + \Omega M)$ 의 시간에 풀 수 있으며, ( $\Omega \leq 26$ 는 알파벳의 개수) 두 복잡도 모두 만점을 받기 충분하다.

---