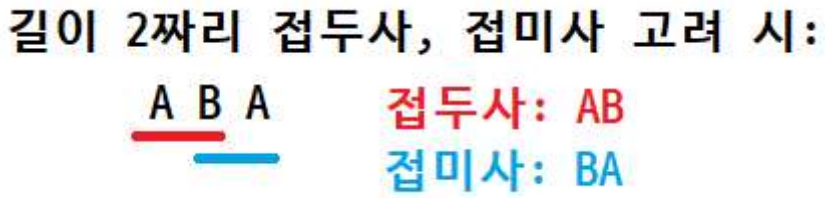
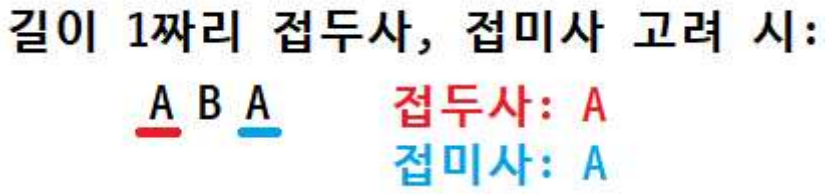
문자열 알고리즘 KMP – (2)

sianke1991

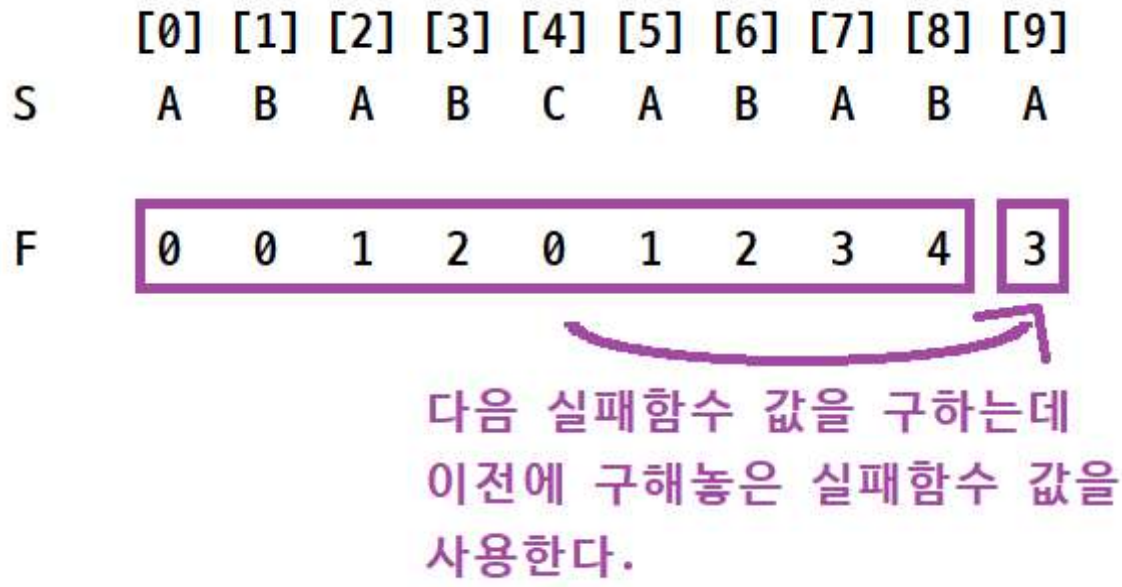
실패함수 F(x) := 문자열 s[0:x+1]에서 접두사와 접미사가 일치하는 최대 길이 (0 이상 x 이하의 정수가 된다.)
e.g. s = “ABABCABABA” 에 대해 F(2)는 s[0:3] = “ABA”의 접두사와 접미사가 일치하는 최대 길이





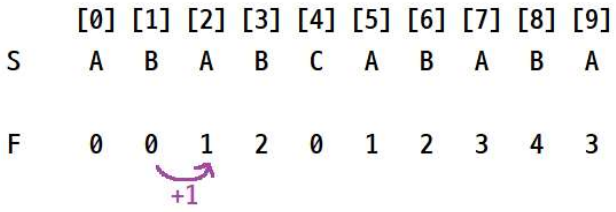
따라서 F(2)는 1이다. (만약 길이 1짜리 접두사, 접미사 역시 서로 다르다면 F(2)는 0이 된다.

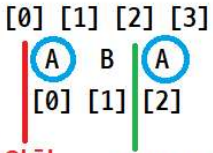
위와 같이 문자열 s에 대해 F(x)를 계산하기 위해서는 길이 x부터 시작하여 접두사, 접미사를 구하면 된다. 하지만 이 방법은 시간이 오래 걸린다. (길이 l 짜리 문자열에 대해 필요로 하는 실패함수 값은 F(l-1) 뿐만 아니라 F(0)부터 F(l-1)까지 총 l개의 값이다.) 이 실패함수 값들을 빠르게 구하기 위해 이전에 구해둔 실패함수 값을 사용할 수 있다.



S = “ABABCABABA”에 대해 F(3)을 구할 때 바로 전 단계에서 구한 접두사, 접미사는 “A”이다. (∵ F(2)는 1이므로 이 단계에서 구한 접두사, 접미사는 길이 1 짜리 문자열이다.) 이 접두사, 접미사에 대해 바로 다음에 위치하는 문자는 'B'로 서로 동일하다. (접두사 바로 다음에 나오는 문자는 S[1] = S[F(2)]이고 접미사 바로 다음에 나오는 문자는 S[3]이다.) 이 경우 F(3)에 해당하는 접두사, 접미사는 F(2)에 해당하는 접두사, 접미사 뒤에 'B'를 붙인 것이다. 따라서 F(3) = F(2)+1 이다.

실패함수 값 F(2) 구하기





접두사의 끝 위치 = F(1)

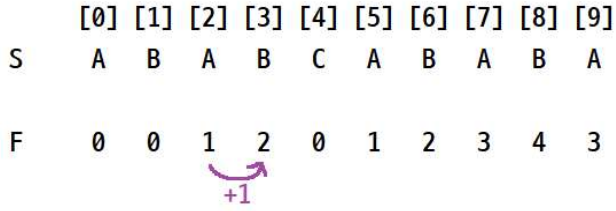
접미사의 끝 위치 = 2

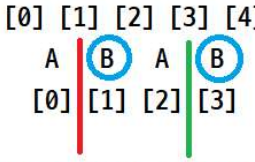
이 경우는 (최근에 구한) 접두사의 끝 위치 다음에 나오는 문자와, (최근에 구한) 접미사의 끝 위치 다음에 나오는 문자가 서로 같다.

(i.e. S[F(1)] = S[2])

이 경우 F(2) = F(1)+1 이다.

실패함수 값 F(3) 구하기





접두사의 끝 위치 = F(2)

접미사의 끝 위치 = 3

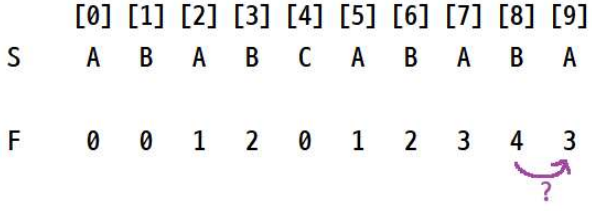
이 경우는 (최근에 구한) 접두사의 끝 위치 다음에 나오는 문자와, (최근에 구한) 접미사의 끝 위치 다음에 나오는 문자가 서로 같다.

(i.e. S[F(2)] = S[3])

이 경우 F(3) = F(2)+1 이다.

이렇게 바로 직전 단계에서 구한 접두사, 접미사 다음에 나오는 문자가 서로 같으면 실패함수 값은 바로 직전 값에 1을 더한 값이 된다. 하지만 바로 직전 단계에서 구한 접두사, 접미사 다음에 나오는 문자가 서로 다르면 어떻게 실패함수 값을 구할 수 있을까? 예를 들어 S = “ABABCABABA”에 대해 F(9)를 구할 때 바로 전 단계에서 구한 접두사, 접미사는 “ABAB”이다.

실패함수 값 F(9) 구하기



이 경우는 (최근에 구한) 접두사의 끝 위치 다음에 나오는 문자와, (최근에 구한) 접미사의 끝 위치 다음에 나오는 문자가 서로 다르다.

(i.e. S[F(8)] ≠ S[9])

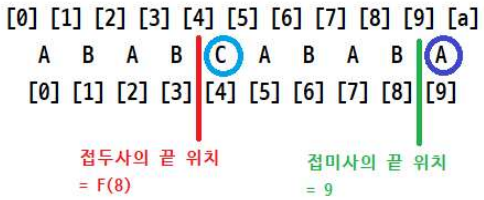
이 경우 F(9) ≠ F(8)+1 이다.

바로 전 단계에서 구한 (최장) 접두사, 접미사인 “ABAB” 다음에 오는 문자가 서로 다르므로 F(9) ≠ F(8)+1 이다. 이 경우에는 (비록 가장 긴 것은 아니더라도) 바로 전 단계의 접두사, 접미사 중 다음으로 긴 것을 고려하는 것이다.

S[0:8] 에 대해 접두사, 접미사가 같아지는 경우는
길이 4 짜리 문자열을 선택할 수도 있지만,
길이 2 짜리 문자열을 선택할 수도 있다.
(여기서 길이 2는 F(3)의 값이다.
길이 4 짜리 문자열의 부분 문자열로써 다음으로 긴 접두사, 접미사를 구해야 하므로
S[0:4]에서 접두사, 접미사의 길이를 의미하는 F(3)이 (다음으로 긴)
접두사, 접미사의 길이가 된다.)

실패함수 값 F(9) 구하기

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
S	A	B	A	B	C	A	B	A	B	A
F	0	0	1	2	0	1	2	3	4	3



이 경우는 (최근에 구한) 접두사의 끝 위치 다음에 나오는 문자와,
(최근에 구한) 접미사의 끝 위치 다음에 나오는 문자가 서로 다르다.
(i.e. S[F(8)] ≠ S[9])
이 경우 F(9) ≠ F(8)+1 이다.

다음으로 긴 접두사, 접미사를 고려한다.

이 경우는 (다음으로 긴) 접두사의 끝 위치 다음에 나오는 문자와,
(다음으로 긴) 접미사의 끝 위치 다음에 나오는 문자가 서로 같다.
(i.e. S[F(F(8)-1)] = S[9])
이 경우 F(9) = F(F(8)-1)+1 이다.

비록 길이 4짜리 접두사와 접미사 다음에 나오는 문자는 서로 다르지만 길이 2짜리 접두사와 접미사 다음에 나오는 문자가 서로 같다. 따라서 F(9) = 3 이다. (만약 길이 2짜리 접두사와 접미사 다음에 나오는 문자도 서로 다를 경우 더 짧은 접두사와 접미사를 찾아 내려간다.)

소스 코드

```
vector<int> failure(string& s) {
    vector<int> f(s.size()); //문자열 s의 길이만큼의 벡터를 생성한다.
    int j = 0; //j: 이전 접두사 바로 다음에 위치하는 문자를 가리키는 인덱스
    for (int i=1; i<s.size(); i++) { //i: 이전 접미사 바로 다음에 위치하는 문자를 가리키는 인덱스
                                     //즉, i는 문자열에 추가되는 문자를 가리키는 인덱스
        while (j>0 && s[i]!=s[j]) j = f[j-1]; //접두사 바로 다음에 위치하는 문자와 접미사 바로 다음에 위치하는 문자가
                                              //서로 다를 경우 더 짧은 접두사, 접미사를 선택해야 한다. (그 두 문자가
                                              //서로 같아질 때 까지-)
        if (s[i]==s[j]) f[i] = ++j; //접두사 바로 다음에 위치하는 문자와 접미사 바로 다음에 위치하는 문자가
                                    //서로 같은 경우 실패함수 값은 (접두사 길이+1)이 된다.
        /* while loop를 빠져나가는 조건은
         * 1. j가 0 이하가 되거나 - 또는 -
         * 2. s[i]==s[j]를 만족하는 것이다.
         * s[i]!=s[j]인 채로 j가 0 이하가 된다면 while loop는 빠져나가지만 if (s[i]==s[j])를 실행하지 않는다.
         * 즉, 접두사를 점점 줄여 나가다가 (접두사, 접미사 다음 문자가 일치하지 못한 채) 접두사의 길이가
         * 0이 경우 if 문은 실행하지 않는다. vector 내 해당 위치의 값은 0인 상태로 그대로 남는다.
         */
    } //i loop
    return f;
}
```