Knuth-Morris-Prat Algorithm (KMP Algorithm)

Problem: Search for word W in text string S.
(Return starting index)

Example: S: ABC_ABCDAB_ABCDABCDABDE

$$\underbrace{\text{ABCDABD}}_{15 \sim 21}$$

W: ABCDABD

Let length of S be n, length of W be k

Naive approach (Brute force): For $\forall m$ $\quad 0 \le m < n$

$(m = 0, 1, 2, \dots, n-1)$

, Compare $S[m], S[m+1], \dots, S[m+k-1]$

$W[0], W[1], \dots, W[k-1]$

---

ex) S: ABCABCD
W: ABCD

```
m=0           m=1           m=2
ABCABCD       ABCABCD       ABCABCD
ABCD           ABCD           ABCD
↑ m=0          ↑ m=1          ↑ m=2
```

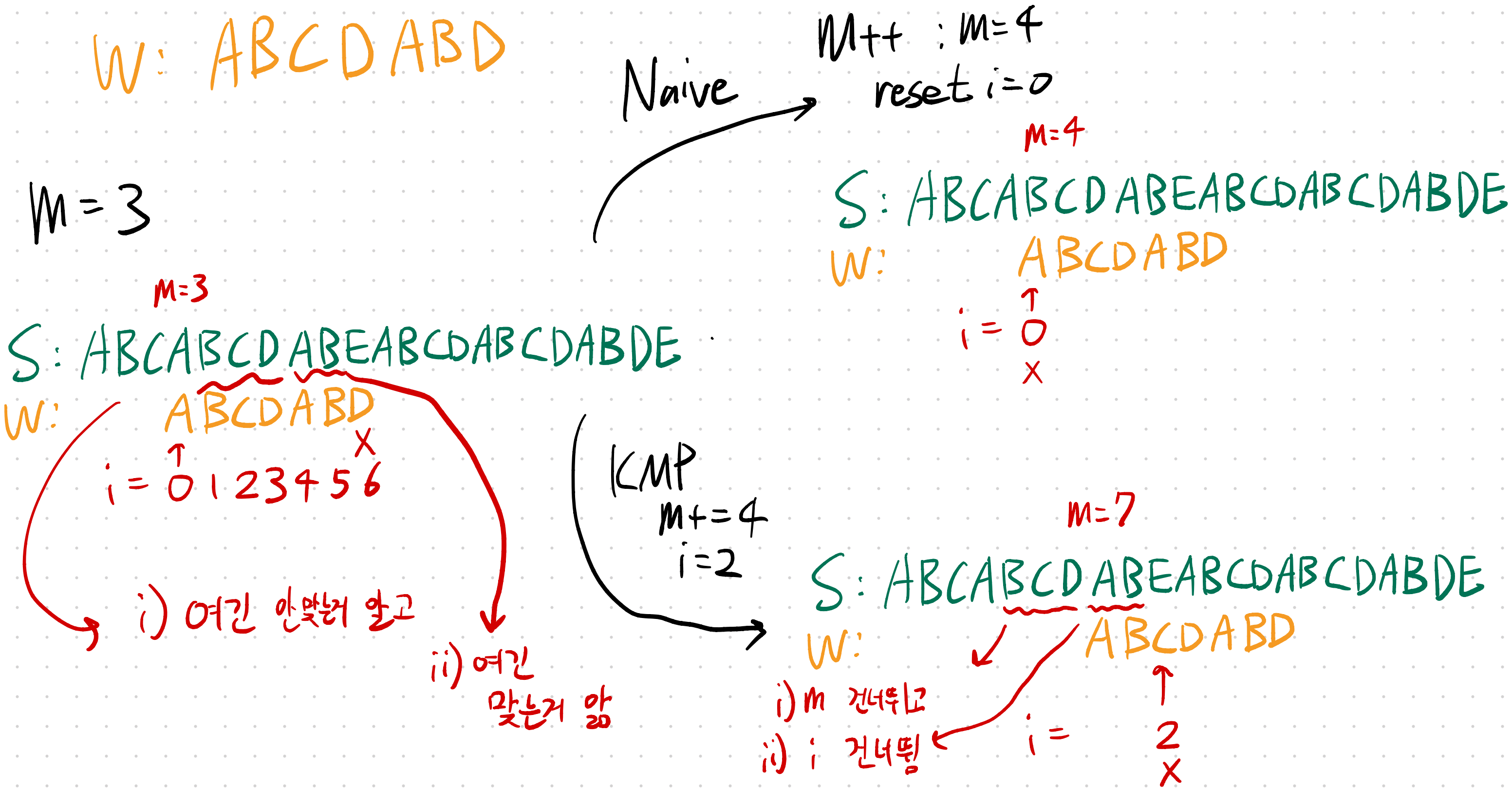
$O(nk)$

ex) S: AAAAAAAAAA...
W: AAAAB

KMP Algorithm: Idea

ex) S: ABCABCDABEABCDABCDABDE

W: ABCDABD



m, i 얼마나 건너뛸지 결정하기

→ 비교중 W[i] 에서 어긋났을 때 얼마나 건너 뛸지가 i에 의해 결정됨

W : ABCDABD

i = 0, 1, 2, 3, 4, 5, 6, (7)

T[i] = -1, 0, 0, 0, -1, 0, 2, 0

W 의 i 번째 에서 틀리면 다음번엔 T[i] 부터 시작하면 됨
(T[i]=-1 이면 0)

$m \to m+i-T[i]$

$i \to \max(T[i], 0)$

ex) $m=0$ $i=0,1,2,3$

S: ABCABCDABEABCDABCDABDE
W: ABCD
X

$m = m+i-T[i] = m+3 = 3$ $i = T[i] = 0$

S: ABCABCDABEABCDABCDABDE
W: A....
i=0

ex) $m=0$ $i=0,1,2,3$

```
S: ABCABCDABEABCDABCDABDE
W: ABCD
      X
```

$\longrightarrow$ $m = m+i-T[i] = m+3-0 = 3$ $i = T[i] = 0$

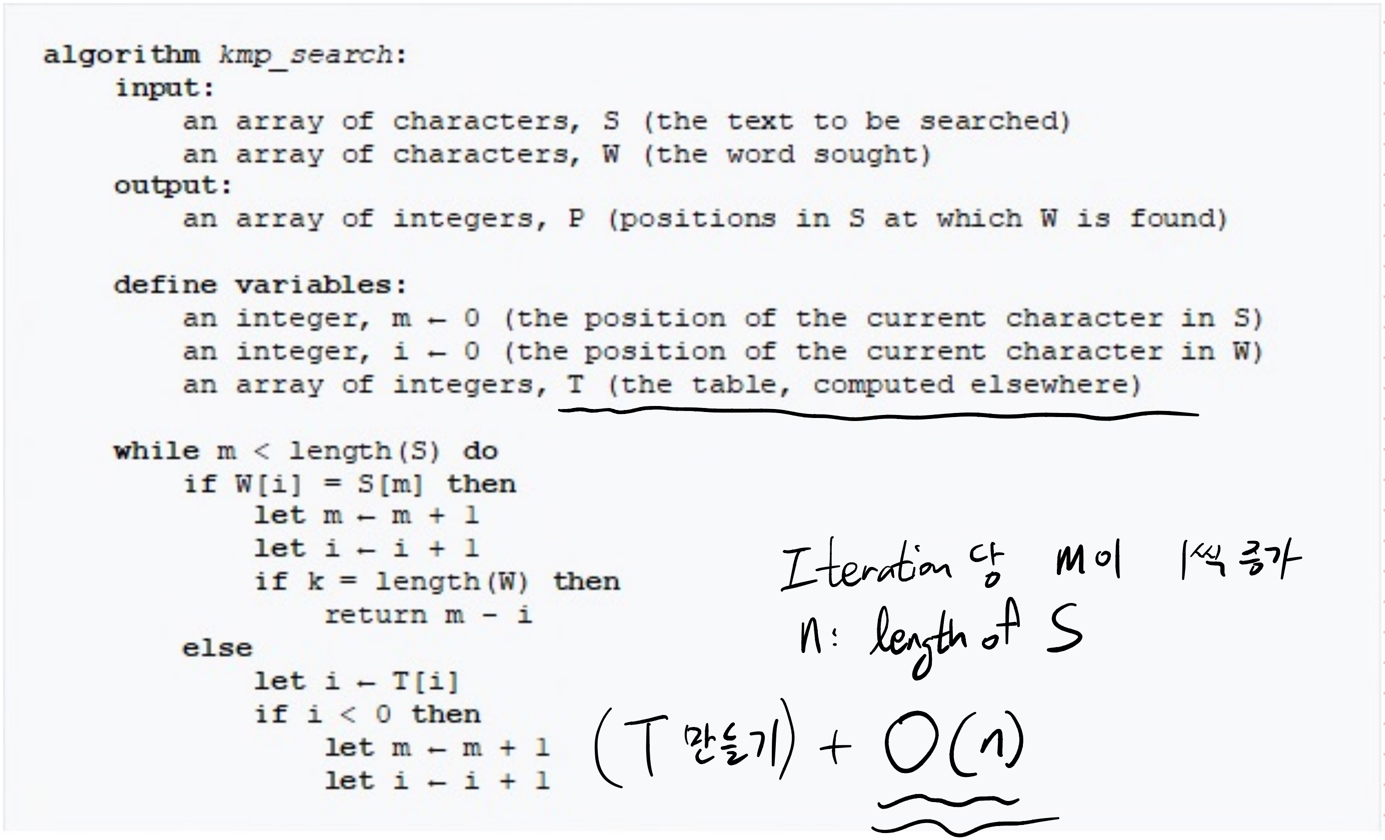
```
S: ABCABCDABEABCDABCDABDE
W:    A....
      i=0
```

ex) $m=3$ $i=0,1,2,3,4,5,6$

```
S: ABCABCDABEABCDABCDABDE
W:    ABCDABD
            X
```

$\longrightarrow$ $m = m+i-T[i] = m+6-2 = 7$ $i = T[i] = 2$

```
S: ABCABCDABEABCDABCDABDE
W:        ABC....
            i=2
```



```
algorithm kmp_search:
    input:
        an array of characters, S (the text to be searched)
        an array of characters, W (the word sought)
    output:
        an array of integers, P (positions in S at which W is found)

    define variables:
        an integer, m ← 0 (the position of the current character in S)
        an integer, i ← 0 (the position of the current character in W)
        an array of integers, T (the table, computed elsewhere)

    while m < length(S) do
        if W[i] = S[m] then
            let m ← m + 1
            let i ← i + 1
            if k = length(W) then
                return m - i
        else
            let i ← T[i]
            if i < 0 then
                let m ← m + 1
                let i ← i + 1
```

Iteration 당 m이 1씩 증가

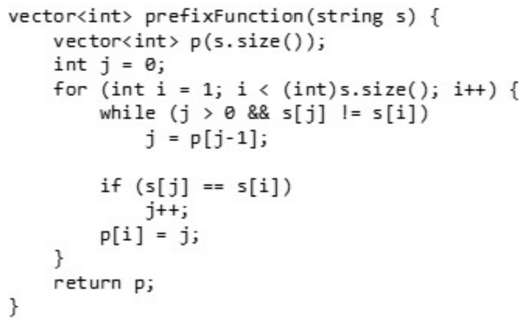
n: length of S

(T 만들기) + $\underline{\underline{O(n)}}$

T[i] 구하기: W[:i] 에서 prefix랑 suffix랑 일치하는 길이.

ABCDABD

i	W[:i]	T[i]
0	X	0
1	A	0
2	AB	0
3	ABC	0
4	ABCD	0
5	ABCDA	1
6	ABCDAB.	2



```
vector<int> prefixFunction(string s) {
    vector<int> p(s.size());
    int j = 0;
    for (int i = 1; i < (int)s.size(); i++) {
        while (j > 0 && s[j] != s[i])
            j = p[j-1];

        if (s[j] == s[i])
            j++;
        p[i] = j;
    }
    return p;
}
```