Knuth-Morris-Pratt (KMP) Algorithm
(String-matching algorithm)

Prefix: A prefix of a string $T = t_0....t_{n-1}$ is a string $P = t_0....t_m$, where n is length of the string T and $m \leq n-1$. A proper prefix of a string is not equal to the string itself.

Suffix: A suffix of a string is any substring of the string which includes its last letter, including itself. A proper suffix of a string is not equal to the string itself.

Let us assume that $T = t_0....t_{n-1}$ (n is length of the string) is the string to be searched and $S = s_0....s_{m-1}$ (m is length of the string) is the pattern to be matched.

Prefix Table:

The KMP algorithm uses a table and in general we call it as Prefix Table or Prefix Function (F). This table is constructed for the pattern to be matched(S). The value at the i^{th} index of this table i.e. $F[i]$, stores the **length of the longest proper prefix that is also a suffix** of the substring $s_0....s_i$, where $i \leq m-1$. This table is used to avoid backtracking on the string T.

Prefix Table for the string S = *"ababaca"*

	0	1	2	3	4	5	6
S	a	b	a	b	a	c	a
F	0	0	1	2	3	0	1

Prefix Table for the string S = *"abcdabca"*

	0	1	2	3	4	5	6	7
S	a	b	c	d	a	b	c	a
F	0	0	0	0	1	2	3	1

Function for Constructing the Prefix Table:

```
int F[]; //Assume F is a global array
void Prefix-Table(char S[], int m)
{
    int i=1, j=0, F[0]=0;
    while(i < m)
    {
        if (S[i]==S[j])
        {
            F[i] = j+1;
            i++;
            j++;
        }
        else if (j > 0)
            j = F[j-1];
        else
        {
            F[i] = 0;
            i++;
        }
    }
}
```

Complexity of the Prefix Table Function:

In the while loop, each time the 'if' block or the 'else' block is executed, value of i increments by 1. Since the initial value of i is 1 and the final value is m, the two blocks execute a total of m times.

The 'else if' block decreases the value of j by at least 1, since $F[j-1]<j$. The value of j can be decreased at most as often as it has been increased previously by j++. Since j++ is present in the 'if' block which is executed at most m times, the overall number of executions of the 'else if' block is limited to m.
The complexity of the function is therefore O(m).

Matching Algorithm:

The basic idea behind KMP's algorithm is: whenever we detect a mismatch (after some matches), we already know some of the characters in the text (since they matched the pattern characters prior to the mismatch). We take advantage of this information to avoid matching the characters that we know will anyway match.

Suppose at any particular step of the algorithm we are performing the comparison of the i^{th} and j^{th} characters of the string T and S respectively.

- If the two **characters match**, we simply proceed to **compare the next character** in both the strings i.e. the $(i+1)^{th}$ and $(j+1)^{th}$ character of the string T and S respectively.
- If the two **characters do not match and j is greater than 0**, then we look at the $(j-1)^{th}$ index of the prefix table that we have constructed and proceed to **compare the i^{th} character of T with the $F[j-1]^{th}$ character of S** i.e. our new j is equal to F[j-1].
- If the two **characters do not match and j is equal to 0**, we proceed to **compare the $(i+1)^{th}$ character of T with the j^{th} character of S.**

We start the comparison with the first character of both the strings ($i = 0, j = 0$) and repeat the above steps until we reach the end of the string T or we reach the end of the string S (In which case, we have found a match).

Function for the Matching Algorithm:

```
int KMP(char T[], int n, char S[], int m)
{
    int i = 0, j = 0;
    Prefix-Table(S, m);
    while(i < n)
    {
        if (T[i]==S[j])
        {
            if (j==m-1)          //match found
                return i-j;
            else
            {
                i++;
                j++;
            }
        }
        else if (j > 0)
            j = F[j-1];
        else
            i++;
    }
    return -1;   //no match found
}
```

Complexity of the Matching Algorithm:

The KMP function is similar to the Prefix-Table function. Therefore, on similar arguments it follows that the KMP function has a complexity of **$O(n)$.**

Overall Time Complexity: Since the two portions of the algorithm have, respectively, complexities of $O(m)$ and $O(n)$, the time complexity of the overall algorithm is $O(m + n)$.

A few problems to try:

- https://www.codechef.com/problems/KAN13C
- https://www.codechef.com/problems/TASHIFT
- http://www.spoj.com/problems/ARDA1/