Pattern Searching algorithms are used to find a pattern or substring from another bigger string. There are different algorithms. The main goal to design these type of algorithms to reduce the time complexity. The traditional approach may take lots of time to complete the pattern searching task for a longer text.

An important subtask of the pattern discovery process is pattern matching, where the pattern sought is already known and we want to determine how often and where it occurs in a sequence.

A central sub problem in pattern discovery is to determine how often a candidate pattern occurs, as well as possibly some information on its frequency distribution across the text. In general, a pattern will be a description of a (finite or infinite) set of strings, each string being a sequence of symbols. Usually, a good pattern must be as specific (that is, denote a small set of strings) and as frequent as possible. Hence, given a candidate pattern, it is usual to ask for its frequency, as well as to examine its occurrences looking for more specific instances of the pattern that are frequent enough.

Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]. You may assume that n > m.

**Examples:**

Input: txt[] = "THIS IS A TEST TEXT"

pat[] = "TEST"

Output: Pattern found at index 10

Input: txt[] = "AABAACAADAABAABA"

pat[] = "AABA"

Output: Pattern found at index 0

Pattern found at index 9

Pattern found at index 12

Pattern searching is an important problem in computer science. When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

**Naive Pattern Searching:**  
Slide the pattern over text one by one and check for a match. If a match is found, then slides by 1 again to check for subsequent matches.

// C program for Naive Pattern Searching algorithm

#include <stdio.h>

#include <string.h>

void search(char\* pat, char\* txt)

{

    int M = strlen(pat);

    int N = strlen(txt);

    /\* A loop to slide pat[] one by one \*/

    for (int i = 0; i <= N - M; i++) {

        int j;

        /\* For current index i, check for pattern match \*/

        for (j = 0; j < M; j++)

            if (txt[i + j] != pat[j])

                break;

        if (j == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]

            printf("Pattern found at index %d \n", i);

    }

}

/\* Driver program to test above function \*/

int main()

{

    char txt[] = "AABAACAADAABAAABAA";

    char pat[] = "AABA";

    search(pat, txt);

    return 0;

}

**Knuth Morris Pratt (KMP)** is an algorithm, which checks the characters from left to right. When a pattern has a sub-pattern appears more than one in the sub-pattern, it uses that property to improve the time complexity, also for in the worst case.

The time complexity of KMP is O(n).

Input and Output

Input:

Main String: “AAAABAAAAABBBAAAAB”, The pattern “AAAB”

Output:

Pattern found at location: 1

Pattern found at location: 7

Pattern found at location: 14