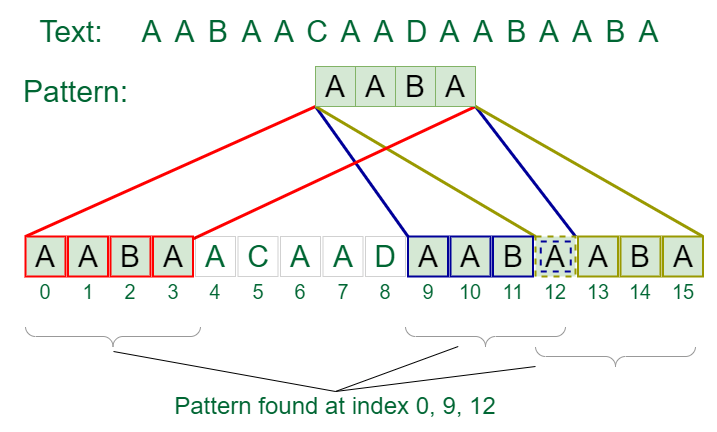
**Naive algorithm for Pattern Searching**

Given a text of length **N** txt[0..N-1]and a pattern of length M 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*

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

## **What is the best case of Naive algorithm for Pattern Searching?**

The best case occurs when the first character of the pattern is not present in the text at all.

|  |
| --- |
| txt[] = "AABCCAADDEE";  pat[] = "FAA"; |

The number of comparisons in the best case is O(N).

## **What is the worst caseof Naive algorithm for Pattern Searching?**

The worst case of Naive Pattern Searching occurs in the following scenarios.   
**1)** When all characters of the text and pattern are the same.

|  |
| --- |
| txt[] = "AAAAAAAAAAAAAAAAAA";  pat[] = "AAAAA"; |

**2)** Worst case also occurs when only the last character is different.

|  |
| --- |
| txt[] = "AAAAAAAAAAAAAAAAAB";  pat[] = "AAAAB"; |

The number of comparisons in the worst case is O(M \* (N – M + 1)). Although strings which have repeated characters are not likely to appear in English text, they may well occur in other applications (for example, in binary texts). The KMP matching algorithm improves the worst case to O(N). We will be covering KMP in the next post. Also, we will be writing more posts to cover all pattern searching algorithms and data structures.

Code in c++

#include <bits/stdc++.h>

using namespace std;

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]

            cout << "Pattern found at index " << i << endl;

    }

}

// Driver's Code

int main()

{

    char txt[] = "AABAACAADAABAAABAA";

    char pat[] = "AABA";

      // Function call

    search(pat, txt);

    return 0;

}

**Time Complexity:**O(N2)  
**Auxiliary Space:** O(1)