PATTERN MATCHING ALGORITHMS

Pattern matching is a problem of deciding whether or not a given string pattern P appears in a string text T. We assume that the length of P does not exceed the length of T.

Naive algorithm for Pattern Matching:

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

C program for Naive Pattern Matching algorithm

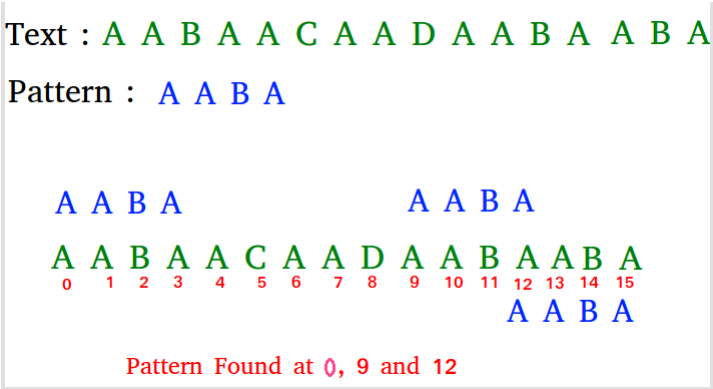
```
#include <stdio.h>
#include <string.h>
void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);
    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++) {
        int j;
        /* For current index i, check for pattern match */
```

```
        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;
}
```

KMP Algorithm for Pattern Matching

```
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
```



C program to implement Knuth–Morris–Pratt algorithm

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
//finds the position of the pattern in the given target string target - str, pattern-
//word
  int  kmpSearch(char *str, char *word, int *ptr) {
        int i = 0, j = 0;
        while ((i + j) < strlen(str)) {
              /* match found on the target and pattern string char */
              if (word[j] == str[i + j]) {
                    if (j == (strlen(word) - 1)) {
                          printf("%s located at the index %d\n",
                                word, i + 1);
                          return;
                    }
                    j = j + 1;
              } else {
                    /* manipulating next indices to compare */
                    i = i + j - ptr[j];
                    if (ptr[j] > -1) {
                          j = ptr[j];
```

```
                } else {
                        j = 0;
                }
        }
    }
}
/* find the overlap array for the given pattern */
void findOverlap(char *word, int *ptr) {
    int i = 2, j = 0, len = strlen(word);
    ptr[0] = -1;
    ptr[1] = 0;
    while (i < len) {
        if (word[i - 1] == word[j]) {
            j = j + 1;
            ptr[i] = j;
            i = i + 1;
        } else if (j > 0) {
            j = ptr[j];
        } else {
            ptr[i] = 0;
            i = i + 1;
        }
    }
    return;
}
int main() {
    char word[256], str[1024];;
    int *ptr, i;
    /* get the target string from the user */
```

```
    printf("Enter your target string:");
    fgets(str, 1024, stdin);
    str[strlen(str) - 1] = '\0';
    /* get the pattern string from the user */
    printf("Enter your pattern string:");
    fgets(word, 256, stdin);
    word[strlen(word) - 1] = '\0';
    /* dynamic memory allocation for overlap array */
    ptr = (int *)calloc(1, sizeof(int) * (strlen(word)));
    /* finds overlap array */
    findOverlap(word, ptr);
    /* find the index of the pattern in target string */
    kmpSearch(str, word, ptr);
    return 0;
}
```