**Experiment 2.3**

|  |  |
| --- | --- |
| **Student Name: SATYAM AWASTHI** | **UID: 20BCS9145** |
| **Branch: CSE** | **Section/Group: 615-B** |
| **Semester: 5th** | **Date of Performance: 03/11/2022** |
| **Subject Name: DAA LAB** | **Subject Code: 21CSP-312** |

1. **Aim/Overview of the practical:**

Knuthh morris prat to search patter matching

# Task to be done/which logistics used:

# Knuthh morris prat to search patter matching

# 3.Steps for experiment/practical/Code:

#include <bits/stdc++.h>

void computeLPSArray(char\* pat, int M, int\* lps);

// Prints occurrences of txt[] in pat[]

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

{

int M = strlen(pat);

int N = strlen(txt);

// create lps[] that will hold the longest prefix suffix

// values for pattern

int lps[M];

// Preprocess the pattern (calculate lps[] array)

computeLPSArray(pat, M, lps);

int i = 0; // index for txt[]

int j = 0; // index for pat[]

while ((N - i) >= (M - j)) {

if (pat[j] == txt[i]) {

j++;

i++;

}

if (j == M) {

printf("Found pattern at index %d ", i - j);

j = lps[j - 1];

}

// mismatch after j matches

else if (i < N && pat[j] != txt[i]) {

// Do not match lps[0..lps[j-1]] characters,

// they will match anyway

if (j != 0)

j = lps[j - 1];

else

i = i + 1;

}

}

}

// Fills lps[] for given pattern pat[0..M-1]

void computeLPSArray(char\* pat, int M, int\* lps)

{

// length of the previous longest prefix suffix

int len = 0;

lps[0] = 0; // lps[0] is always 0

// the loop calculates lps[i] for i = 1 to M-1

int i = 1;

while (i < M) {

if (pat[i] == pat[len]) {

len++;

lps[i] = len;

i++;

}

else // (pat[i] != pat[len])

{

// This is tricky. Consider the example.

// AAACAAAA and i = 7. The idea is similar

// to search step.

if (len != 0) {

len = lps[len - 1];

// Also, note that we do not increment

// i here

}

else // if (len == 0)

{

lps[i] = 0;

i++;

}

}

}

}

// Driver program to test above function

int main()

{

char txt[] = "ABABDABACDABABCABAB";

char pat[] = "ABABCABAB";

KMPSearch(pat, txt);

return 0;

}

# Output:

# WhatsApp Image 2022-11-06 at 22.14.01

# Learning Outcomes:

a) Create a program keeping in mind the time complexity.

b) Create a program keeping in mind the space complexity.

c) Steps to make optimal algorithm.

d) Learnt about how to implement 0-1 Knapsack problem using dynamic programming.