**Experiment 3.3**

**Student Name: Sahul Kumar Parida UID: 20BCS4919**

**Branch: CSE Section/Group: WM-904/B**

**Semester: 5th Date of Performance: 08/11/2022**

**Subject Name: Design and Analysis Algorithm Lab**

**Subject Code: 20CSP-312**

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

Code and Analyze to find all occurrences of a pattern P in a given string S.

1. **Task to be done/which logistics used:**

To solve this problem with the KMP(Knuth–Morris–Pratt) algorithm.

The idea is to use the KMP string matching algorithm to find all occurrences of the pattern in the string efficiently. The main logic of the KMP algorithm is that whenever we detect a mismatch, we already know that some of the characters of the pattern have matched with the string. We can use this information to avoid matching the characters we know have already matched.

We solve this problem with the KMP algorithm, we first need to find the LPS array of the pattern used in KMP. Here, LPS[i] is the longest proper prefix which is also a suffix for the substring pat[0:i].

1. **Algorithm/Flowchart:**
2. We will first create the LPS array.
3. Initialize two variables - ‘strIdx’ and ‘patIdx’ to iterate over the string and the pattern, respectively.
4. If ‘pat[patIdx]’ equals ‘str[strIdx]’, we will increment both the indexes.
5. When ‘patIdx’ equals the length of the pattern, this means that the pattern is found in the string. Therefore, we print the index and set ‘patIdx’ = LPS[patIdx-1].
6. If ‘pat[patIdx]’ is not equal to ‘str[strIdx]’, we update the patIdx with the last index that matches with ‘str[strIdx]’ using the LPS array.

1. **Steps for experiment/practical/Code:**

#include<bits/stdc++.h>

using namespace std;

vector<int> getLps(string pat){

int m = pat.size();

// Vector to store the LPS array.

vector<int>lps(m);

/\*

Prev is the last longest proper prefix

which is also a suffix

\*/

int prev = 0;

int ind = 1;

while (ind < m){

// If both are equal

if (pat[ind]==pat[prev]){

prev++;

lps[ind]=prev;

ind++;

}

/\*

If the current elements are unequal

And LPS is 0

\*/

else if (prev==0){

lps[ind]=0;

ind++;

}

/\*

If the current elements are unequal

But LPS is not 0

\*/

else{

prev = lps[prev-1];

}

}

return lps;

}

/\*

Function to find

All occurrences of the pattern in the string

\*/

void solve(string str, string pat){

vector<int> lps = getLps(pat);

// Initializing variables

int n = str.size();

int m = pat.size();

int patIdx = 0;

int strIdx = 0;

while (strIdx < n){

// If both the elements match

if (str[strIdx] == pat[patIdx]){

patIdx++;

strIdx++;

}

if (patIdx == m){

/\*

This means that the entire pattern has matched

Printing all occurrences of the pattern in the string

\*/

cout<<strIdx - m<<' ';

// Updating patIdx to the last element matched.

patIdx = lps[patIdx-1];

}

else if (strIdx < n){

if (str[strIdx]!=pat[patIdx]){

if (patIdx != 0)

// Updating lps to the last element matched

patIdx = lps[patIdx-1];

else

// If no element matched, we move to next index

strIdx++;

}

}

}

cout<<endl;

}

int main()

{

cout << "Sahul Kumar Parida" << endl;

cout << "20BCS4919" << endl;

string str = "heyhihey";

string pat = "hey";

cout<<"Pattern found at: "<<endl;

solve(str, pat);

}

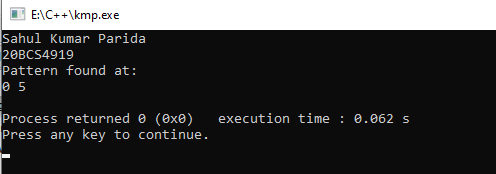
1. **Observations/Discussions/ Complexity Analysis:**

Computing the LPS array takes O(M) time. The time complexity of the KMP algorithm is O(N + M). Here ‘N’ is the length of the string, and ‘M’ is the length of the pattern.

Therefore, the total time complexity of the above approach is O(N + M).

We are creating a new LPS array of size O(M). Therefore, the space complexity of the above approach is O(M).

1. **Output:**



**Learning outcomes (What I have learnt):**

1. KMP algorithm is used where we have to find patterns in long strings. It can be used to search a substring in a string, find the number of unique substrings in a string, find all occurrences of the pattern in the string, etc.
2. LPS stands for Longest Proper Prefix, which is also a Suffix. As the name suggests, LPS[i] stores the longest proper prefix, also a suffix for the substring pat[0:i]. The proper prefix of a string is any prefix other than the entire string itself.