**Worksheet-3.3**

**Student Name:-** Pushpraj Roy **UID:-** 20BCS9866

**Branch:-** BE- CSE **Section/Group:-** WM\_617 “A”

**Subjetct Code:-** 20CSP-312 **Semester:-** 5th

**Subject Name:-** DAA Lab

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 find all occurrences of a pattern P in a given string S.

1. **Algorithm/Flowchart :-**
2. We start comparison of pat[j] with j = 0 with characters of current window of text.
3. We keep matching characters txt[i] and pat[j] and keep incrementing i and j while pat[j] and txt[i] keep matching.
4. When we see a mismatch –

* We know that characters pat[0..j-1] match with txt[i-j…i-1] (Note that j starts with 0 and increment it only when there is a match).
* We also know (from above definition) that lps[j-1] is count of characters of pat[0…j-1] that are both proper prefix and suffix.
* From above two points, we can conclude that we do not need to match these lps[j-1] characters with txt[i-j…i-1] because we know that these characters will anyway match.

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

#include<iostream>

using namespace std;

void findPrefix(string pattern, int m, int prefArray[]) {

int length = 0;

prefArray[0] = 0;

for(int i = 1; i<m; i++)

{

if(pattern[i] == pattern[length])

{

length++;

prefArray[i] = length;

}

else

{

if(length != 0)

{

Length = prefArray[length - 1];

i--;

}

else

prefArray[i] = 0;

}

}

}

void kmpPattSearch(string mainString, string pattern, int \*locArray, int &loc)

{

int n, m, i = 0, j = 0;

n = mainString.size();

m = pattern.size();

int prefixArray[m];

findPrefix(pattern, m, prefixArray);

loc = 0;

while(i < n)

{

if(mainString[i] == pattern[j]) {

i++; j++;

}

if(j == m)

{

locArray[loc] = i-j;

loc++;

j = prefixArray[j-1];

}

else if(i < n && pattern[j] != mainString[i])

{

if(j != 0)

j = prefixArray[j-1];

else

i++;

}

}

}

int main()

{

string str = "AAAABAAAAABBBAAAAB";

string patt = "AAAB";

int locationArray[str.size()];

int index;

kmpPattSearch(str, patt, locationArray, index);

for(int i = 0; i<index; i++)

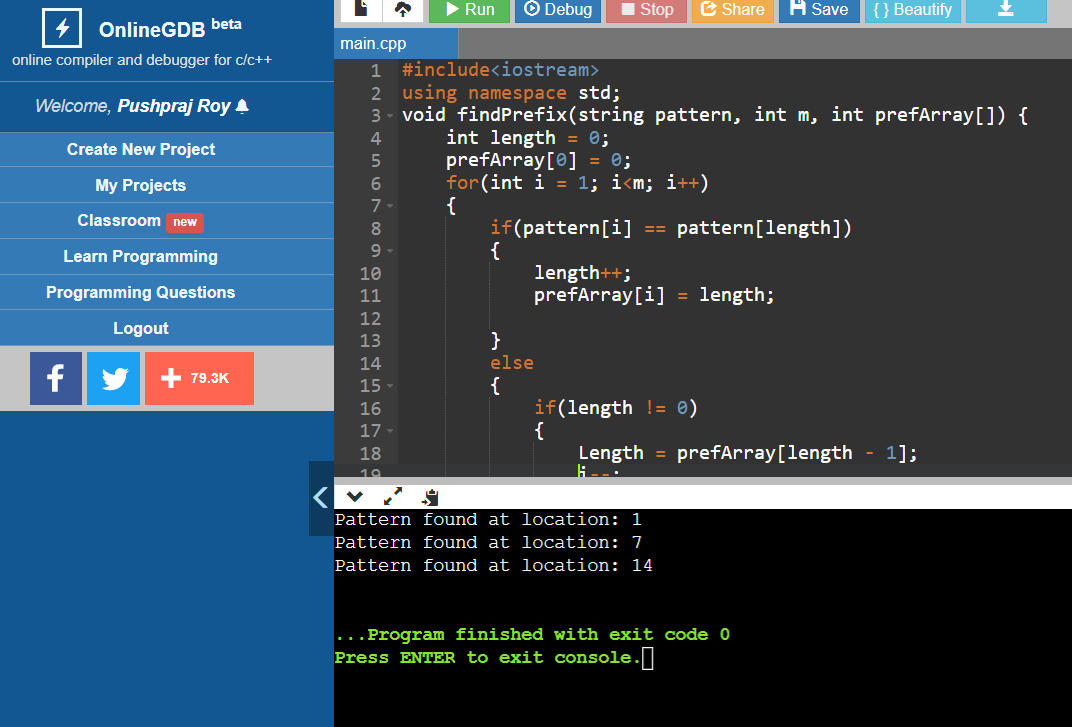
{

cout << "Pattern found at location: " <<locationArray[i] << endl;

}

}

1. **Result/Output :-**

****

1. **Learning outcomes (What I have learnt) –**
   * The KMP matching algorithm uses degenerating property (pattern

having same sub-patterns appearing more than once in the pattern) of the

pattern and improves the worst-case complexity to O(n).

* + The basic idea behind KMP’s algorithm is whenever we detect a

mismatch (after some matches), we already know some of the characters

in the text of the next window.

* + To learn the importance of designing an algorithm in an effective way by

considering space and time complexity.