EXPERIMENT 15

Experiment No: 15 **Date:** 13/06/2021

Aim: Implementation of KMP Pattern Matching Algorithm

Theory:

KMP Algorithm

- Knuth Morris Pratt (KMP) is an algorithm, which checks the characters from left to right.
- When a pattern has a sub-pattern appears more than one in the sub-pattern, it uses that property to improve the time complexity, also for in the worst case.

Pattern Matching

- The Pattern Searching algorithms are sometimes also referred to as String Searching Algorithms and are considered as a part of the String algorithms.
- These algorithms are useful in the case of searching a string within another string.
- Pattern searching is an important problem in computer science.
- When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

The Failure Function

- ***arr*** is initialized using the failure function ***f(i)***.
- This function is based on the fact that:
    - When a mismatch occurs, all the previous characters match correctly.
    - This implies that if a prefix of ***W*** occurred in this set of matching characters, then that prefix is also a suffix of ***W***.
- In other words, ***arr[i]*** will represent the length of the longest proper prefix of ***W***, which is also a proper suffix of ***W*** (considering ***W*** till the ***i*** th index only).

EXPERIMENT 15

Advantages of KMP Algorithm

- The most obvious advantage of KMP Algorithm – data is that it's guaranteed worst-case efficiency as discussed.
- The pre-processing and the always-on time are pre-defined.
- There are no worst-case or accidental inputs.
- Preferable where the search string in a larger space is easier and more efficiently searched due to it being a time linear algorithm.
- The algorithm needs to move backward in the input text, this is particularly favourable in processing large files.

Disadvantages of KMP Algorithm

- One of the glaring disadvantages of KMP Algorithm – data is that it doesn't work well when the size of the alphabets increases, due to this more and more error occurs.
- For processing very large files it also requires resources in the form of processors and that could be a problem for smaller organizations to adopt KMP Algorithm – data

Algorithm : KMPFailureFunction(P)

Algorithm KMPFailureFunction(P):

Input: String P (pattern) with m characters

Output: The failure function f for P. which maps j to the length of the

prefix of P that is a suffix of P[1,j]

i<-1

j<-0

f(0)<-0

while i<m do

EXPERIMENT 15

```
if P[i] = P[j]  then
        {we have matched J+1 characters}
        f(i) <- j+1
        j<-j+1
        i<-i+1
else if j>0 then
        {j indexes just after a prefix of P that must match}
        j<-f(j-1)
else
        {we have no match here}
        f(i)<- 0
        I<-i+1
```

Algorithm : KMPMatch(T, P)

```
Algorithm KMPMatch(T,P):
    Input: Strings T (text) with n characters and P (pattern) with m characters
    Output: Starting index of the first substring of 7 matching P, or an indicating
            that P is not a substring of T
    f-KMPFailureFunction(P)          { Construct the Failure Function f for P}
    i<-0
    j<-0
    while i<n do
      if Pj = T then
```

EXPERIMENT 15

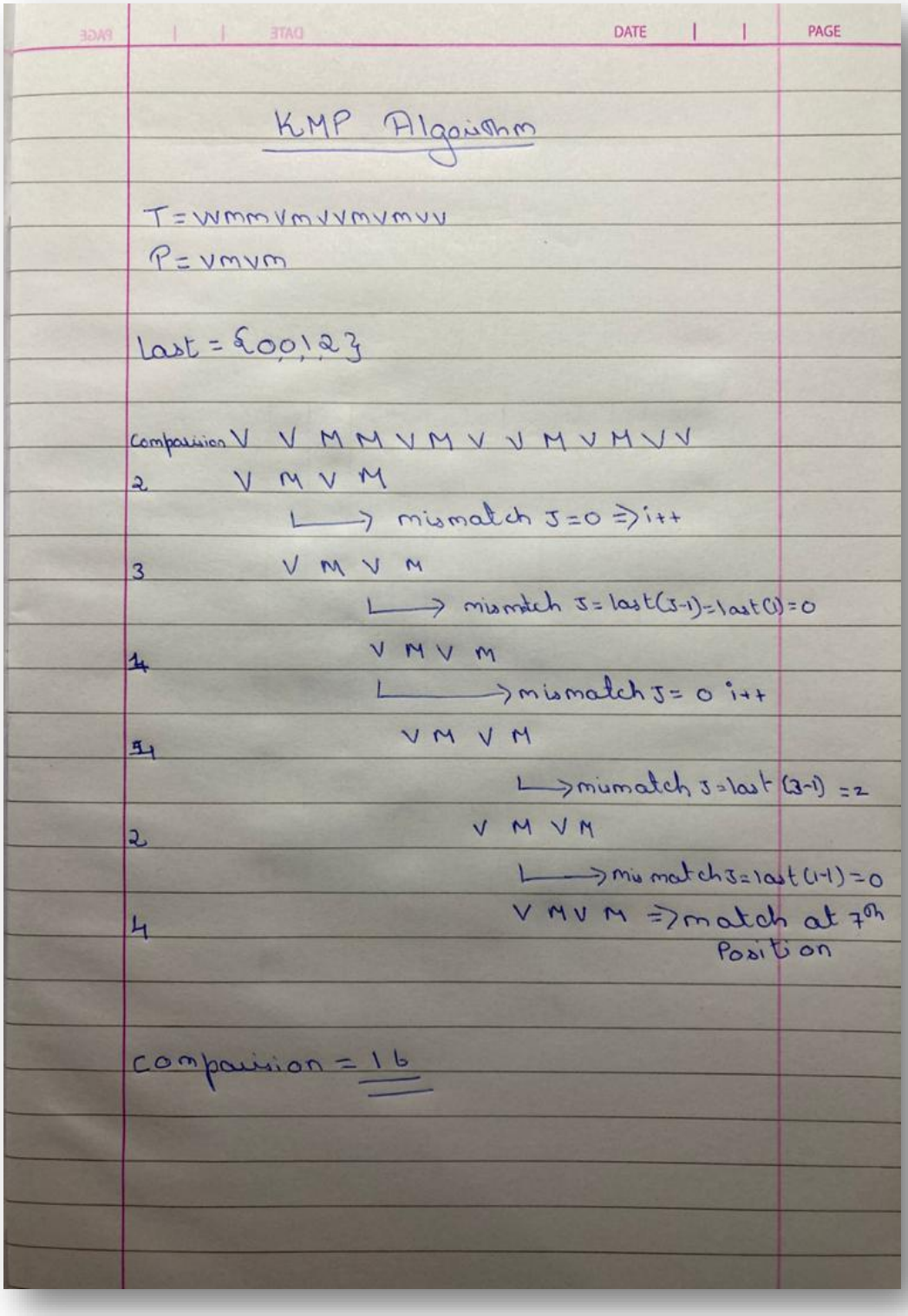
```
            if j=m-1 then
                    return i-m +1  {a match!}
            i<-i+1
            j<-j+1
    else if > 0 {no match, but we have advanced in P} then
            j <- f(j-1) {j indexes just after prefix of P that must match}
    else
        i <- i+1
    return "There is no substring of I matching P"
```

Time Complexity

- The prefix-function already tells us what needs to be done about the already matched parts, we do not need to check those parts again and hence, only one pass over the string is sufficient.
- So, the time complexity of the matching is ***O(n)***.
- What about the pre-computation? It turns out that, if correctly implemented, it can be done in O(m)O(m) time.
- It takes ***O(m + n)*** time.
- Since m ≤ n, the time is bounded by ***O(n)***.
- This is certainly better than the usual ***O(n^2)***.

EXPERIMENT 15

Tracing With Examples



KMP Algorithm

T = vmmvmvvmvmvv

P = vmvm

last = {0,1,2}

Comparision V V M M V M V V M V M V V

2 V M V M
↳ mismatch J=0 ⇒ i++

3 V M V M
↳ mismatch J = last(J-1) = last(1) = 0

4 V M V M
↳ mismatch J = 0 i++

5 V M V M
↳ mismatch J = last(J-1) = 2

2 V M V M
↳ mismatch J = last(1-1) = 0

4 V M V M ⇒ match at 7th Position

comparision = 16

EXPERIMENT 15

Program

```
#include<bits/stdc++.h>

using namespace std;

void kmpFailure(vector<int> &X,string b);

int main()

{

        string a,b;

        int y,z;

        cout<<"**************************"<<endl;

        cout<<"Enter The String:"<<endl;

        cin>>a;

        cout<<"**************************"<<endl;

        cout<<"Enter The Pattern:"<<endl;

        cin>>b;

        y=a.size();

        z=b.size();

        int found=0;

   vector<int> X(z,0);

   kmpFailure(X,b);

   int c=0,i=0,j=0;

   while(i<y)

   {
```

EXPERIMENT 15

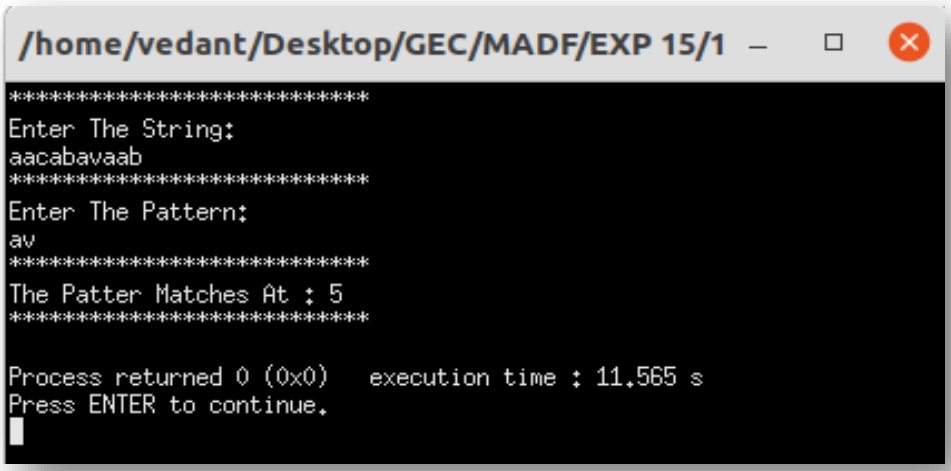
```
        if(b[j]==a[i])
        {

                if(j==z-1)
                {
                        cout<<"**************************"<<endl;
                                cout<<"The Patter Matches At : "<<i-z+1<<endl;

cout<<"**************************"<<endl;
                        found=1;
                        break;
                        }
                        i++;
                        j++;
                }
                else if(j>0)
                j=X[j-1];
                else i++;

        }
        if(found==0)
  cout<<"**************************"<<endl;
```

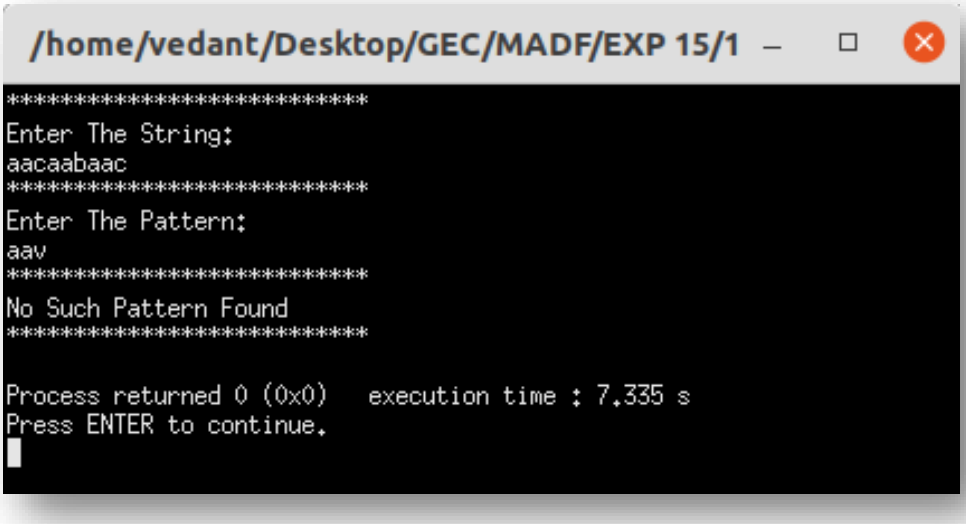
EXPERIMENT 15

```
        cout<<"No Such Pattern Found"<<endl;
        cout<<"**************************"<<endl;
}
void kmpFailure(vector<int> &X,string b)
{
        int i=1,j=0;
        X[0] = 0;
        while(i<b.size())
        {
                if(b[i]==b[j])
                {
                        X[i] = j+1;
                        i++;
                        j++;
                }else if(j>0){
                        j=X[j-1];
                }else{
                        X[i] = 0;
                        i++;
                }
        }
}
```

EXPERIMENT 15

Output





Conclusion

- Detailed concept of KMP Pattern Matching Approach was studied successfully.
- Program using KMP Pattern Matching Algorithm was executed successfully.