PROJECT REPORT: Brute Force Search

Course ID: CSE211

Section: 03

Submitted To:

Md. Asif Bin Khaled

Department of Computer Science & Engineering

Independent University, Bangladesh

Submitted By:

Mahrina Mohsin Raisa

ID: 1810265

Md. Tahmid Hossain

ID: 1820229

Farihan Farabi

ID: 1820807

1. Introduction of the Paradigm:

Brute force problem solving paradigm is a very simple algorithm that solves problems by checking all the possible ways to solve it. It does not contain any approach that won’t help in the outcome of the solution. As a result, using Brute-force search ensures the best solution to a given problem. Efficiency is assured since this paradigm relying absolutely on computer power, disregards complex techniques and sticks to a straightforward practice. However in some cases this approach may not be considered as the most efficient approach since it tests all the possible outcomes before providing the solution. This paradigm is also known as Exhaustive Search.

2. Fundamentals of the Given Paradigm:

Brute force search algorithmic paradigm is very easy to implement and it always provides a solution, if it exists. It is quite simpler than most other algorithmic paradigms. In Brute-force search, the cost depends on the number of possible solutions and so brute-force search is best used when the problem size is limited or when it is possible to apply problem-specific heuristics that can reduce the set of candidate solutions to an acceptable amount.

Implementation of Brute-Force Search-
Algorithm-

```
$c \leftarrow first(P)$
while $c \neq \Lambda$ do
    if $valid(P,c)$ then
        $output(P, c)$
    $c \leftarrow next(P, c)$
end while
```

Here, valid(P,c) checks whether candidate c is a solution for P
output(P,c) states solution c of P to be an appropriate solution.

The algorithm will keep on incrementing for every possible outcome that is a solution to the given instance P and return “null candidate” otherwise.

Brute force approach for pattern matching is a process where a string n is attempted to be matched with substring m within a given text string. Matching is done thoroughly throughout the main text. If a wrong match occurs, it leads to text backtracking. Worst case for string search algorithms may resort to O(mn) time.

The main disadvantage of brute-force method is that when the data size increases, the number of possible outcomes undergoes a steep growth making the algorithm an undesirable approach to problem solving. Therefore, it is very important to know the use cases of this paradigm.

3. Notable Algorithms in the Given Paradigm:

The brute force paradigm has many algorithms. Some of the notable algorithms are:

- ➔ Rabin-Karp algorithm
- ➔ Floyd’s cycle detection algorithm
- ➔ Boyer-Moore algorithm
- ➔ Lowest Common Ancestor
- ➔ Convex Hull
- ➔ Naive string search algorithm

4. Introduction of the Selected Algorithm

KMP Algorithm or Knuth-Morris-Pratt Algorithm is a pattern matching algorithm. It was the first Linear time complexity algorithm for string matching.The KMP algorithm is an efficient string matching algorithm due to Donald Knuth, Vaughan Pratt, and James H. Morris who wrote the paper on KMP Algorithm in 1977 although James Morris had independently invented the algorithm in 1970. It is a linear time algorithm that exploits the observation that every time a match or a mismatch A String

Matching or String Algorithm in Computer Science is string or pattern recognition in a larger space finding similar strings. KMP Algorithm- data thus uses such string algorithms to improve the time taken to find and eliminate such patterns when searching and therefore called linear time complexity algorithm happens,the pattern itself contains enough information to dictate where the new examination should begin from.

5. Pseudocode:

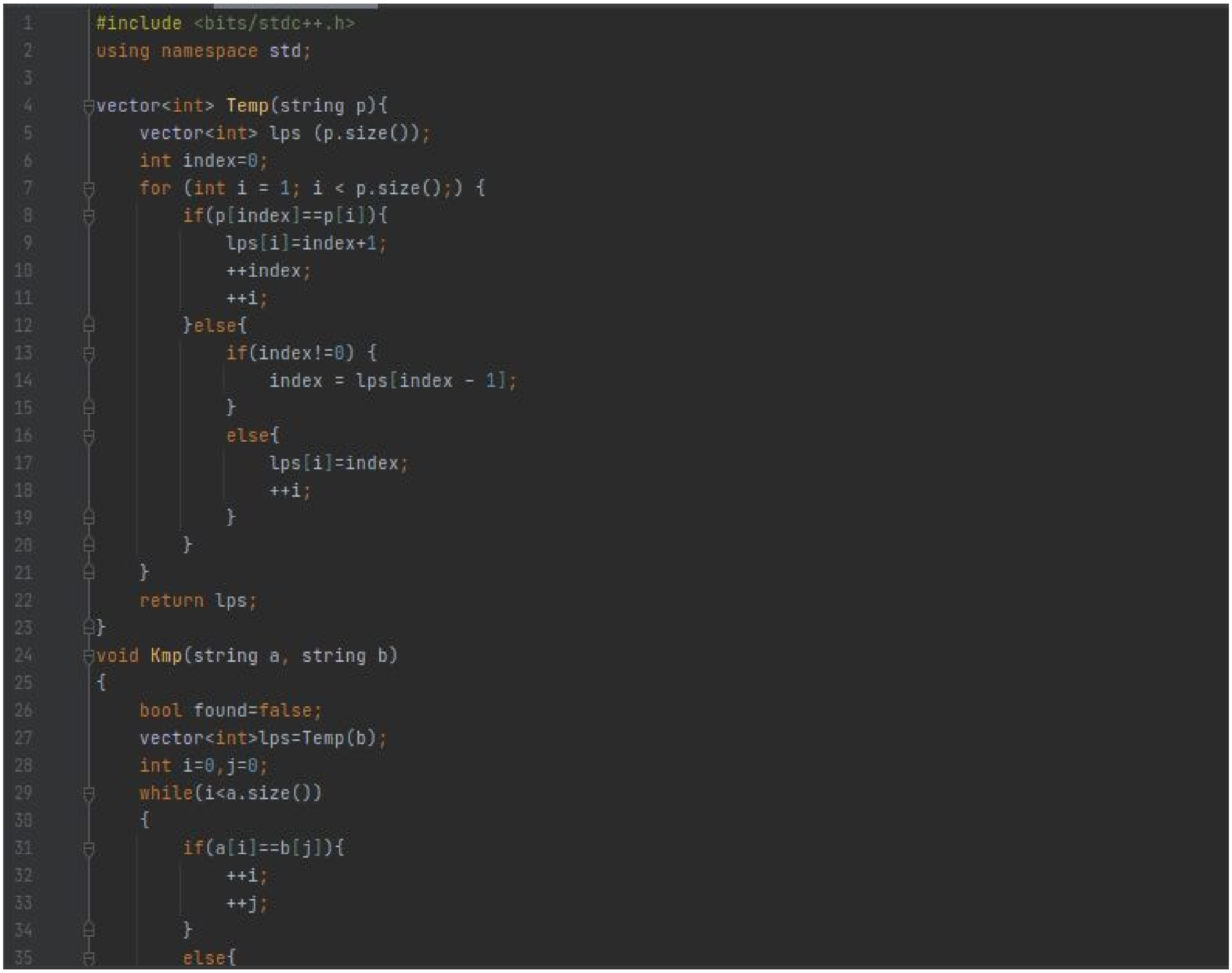
```
n= T.length
m= P.length
π= ComputePrefixFunction(P)
q= 0
for i=1 to n
    while q and P[q+1]≠T[i]
        q=π[q]
    if P[q+1] == T[i]
        q++
    if q==m
        print “Pattern Occurs with Shift” i-m
        q=π[q]
```

ComputePrefixFunction(P)

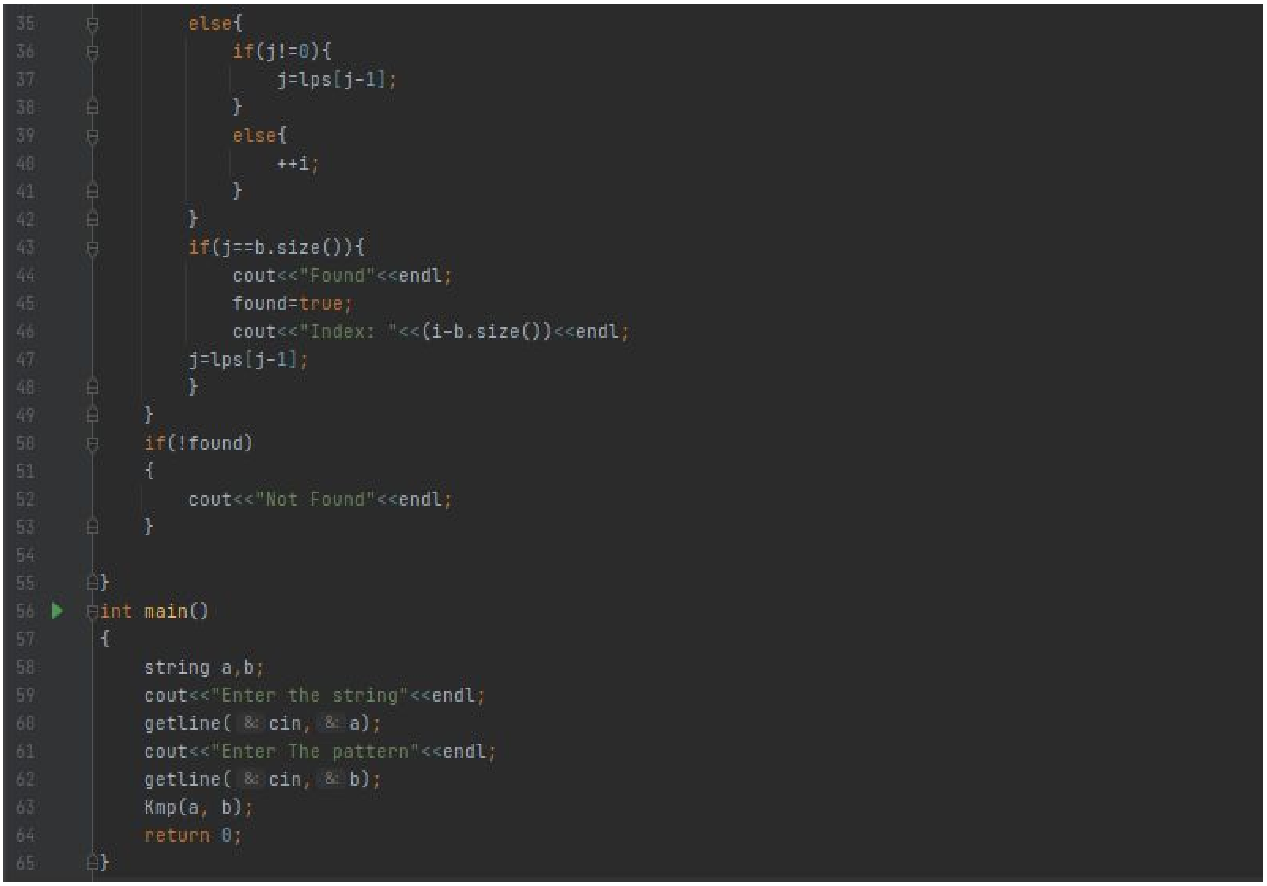
```
m=P.length
Let π[1...m] be a new array
π[1]=0
k=0
for q=2 to m
    while k>0 and P[k+1]≠P[q]
        k=π[k]
    if P[k+1]== P[q]
        k++
    π[q]=k
return π
```

6. Implementation:

Here is the implementation of the KMP algorithm:

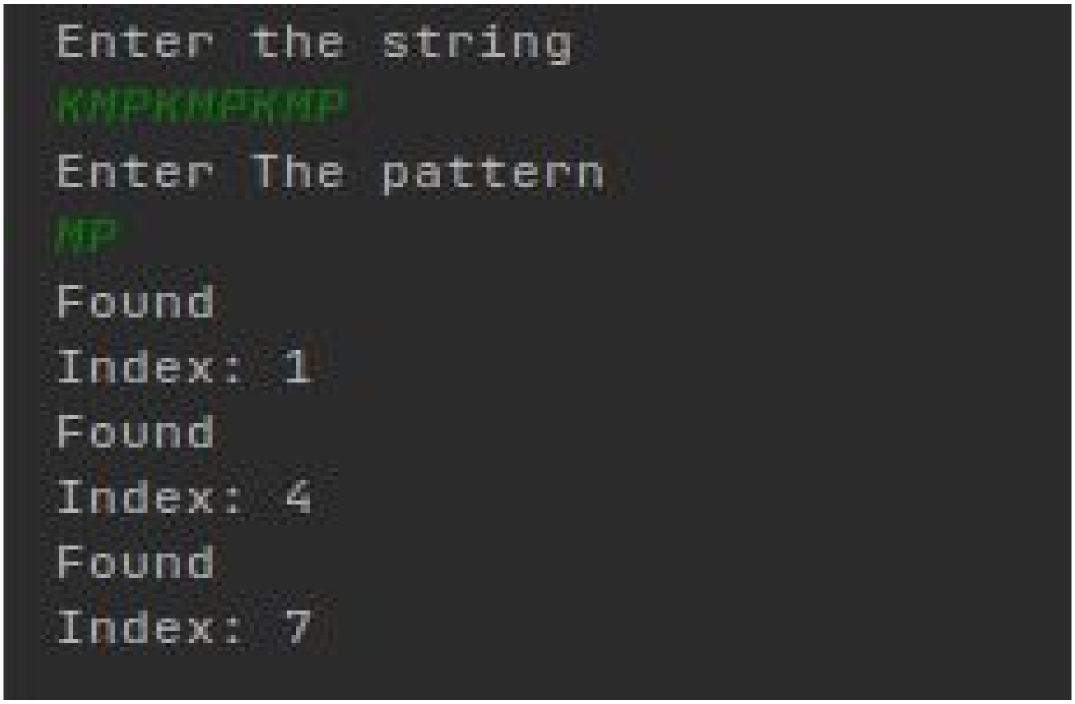


```
#include <bits/stdc++.h>
using namespace std;

vector<int> Temp(string p){
    vector<int> lps (p.size());
    int index=0;
    for (int i = 1; i < p.size();) {
        if(p[index]==p[i]){
            lps[i]=index+1;
            ++index;
            ++i;
        }else{
            if(index!=0) {
                index = lps[index - 1];
            }
            else{
                lps[i]=index;
                ++i;
            }
        }
    }
    return lps;
}
void Kmp(string a, string b)
{
    bool found=false;
    vector<int>lps=Temp(b);
    int i=0,j=0;
    while(i<a.size())
    {
        if(a[i]==b[j]){
            ++i;
            ++j;
        }
        else{
```



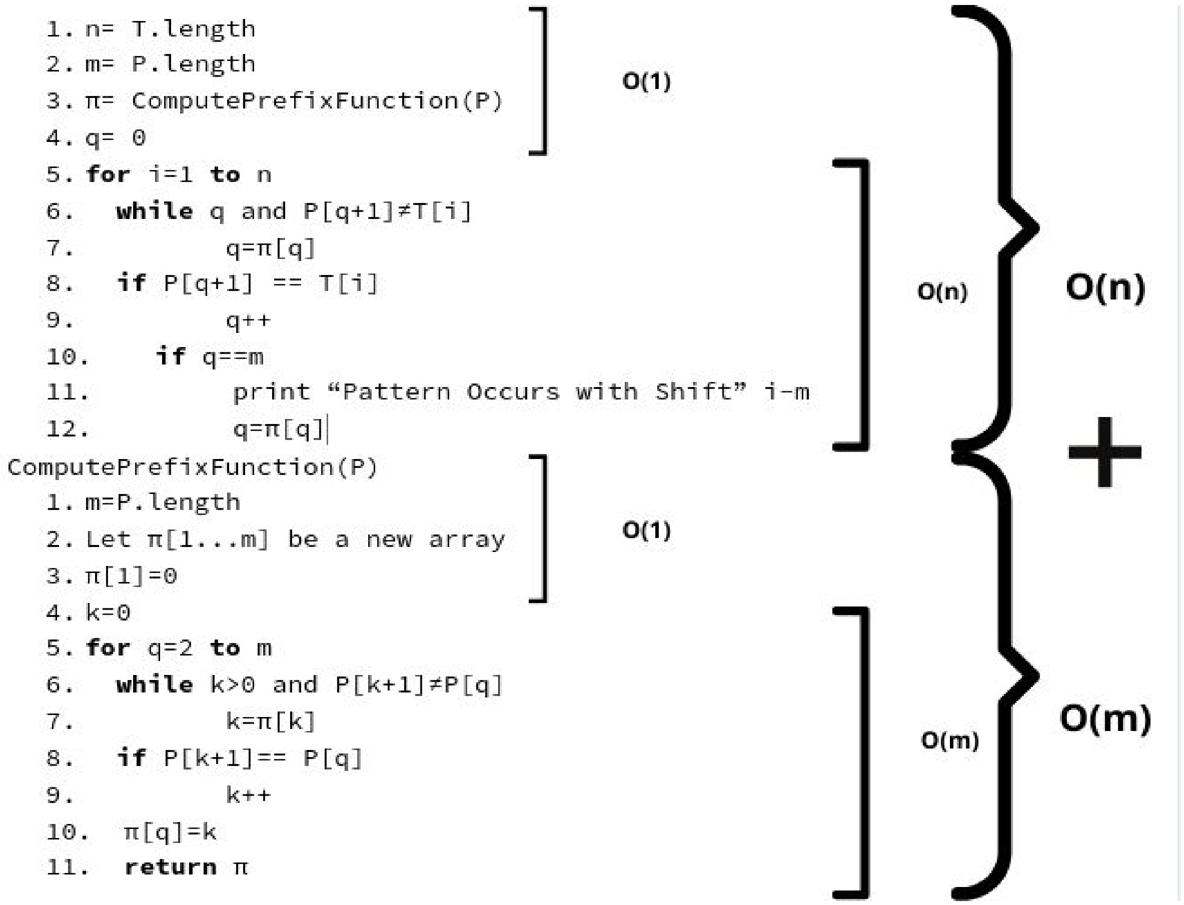
```
        else{
            if(j!=0){
                j=lps[j-1];
            }
            else{
                ++i;
            }
        }
        if(j==b.size()){
            cout<<"Found"<<endl;
            found=true;
            cout<<"Index: "<<(i-b.size())<<endl;
        j=lps[j-1];
        }
    }
    if(!found)
    {
        cout<<"Not Found"<<endl;
    }

}
int main()
{
    string a,b;
    cout<<"Enter the string"<<endl;
    getline( & cin, & a);
    cout<<"Enter The pattern"<<endl;
    getline( & cin, & b);
    Kmp(a, b);
    return 0;
}
```

Output:



```
Enter the string
KMPKMPKMP
Enter The pattern
MP
Found
Index: 1
Found
Index: 4
Found
Index: 7
```

7. Complexity Analysis:



```
1. n= T.length
2. m= P.length
3. π= ComputePrefixFunction(P)
4. q= 0
5. for i=1 to n
6.   while q and P[q+1]≠T[i]
7.        q=π[q]
8.   if P[q+1] == T[i]
9.        q++
10.    if q==m
11.          print "Pattern Occurs with Shift" i-m
12.          q=π[q]
ComputePrefixFunction(P)
1. m=P.length
2. Let π[1...m] be a new array
3. π[1]=0
4. k=0
5. for q=2 to m
6.   while k>0 and P[k+1]≠P[q]
7.        k=π[k]
8.   if P[k+1]== P[q]
9.        k++
10.  π[q]=k
11.  return π
```

After a quick analysis of the pseudocode, we know, the ComputePrefixFunction runs in Θ(m) times and the for loop body runs Θ(n) times. Therefore, the KMP algorithm has a complexity of O(n+m) for the worst case.

Within the while loop of the ComputePrefixFunction, the variable k is set to 0 as seen in line 4. Line 9 increments k once for every execution of the for loop. However, line 7 decrements k with every execution as π[k]<k. Each time the for loop body is entered, q gets incremented and thus, k<q is persistent throughout the entire run time. The amortized cost for line 6-10 runs in constant time O(1).

8. Use Cases:

We know that the best advantage of KMP Algorithm is that it's guaranteed worst-case efficiency. The pre-processing and the always-on time is also pre-defined.

There is no occurrence of worst case or/accidental inputs in this algorithm.

It is used 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 favorable in processing large files.

9. Recommended Use:

The KMP algorithm plays an important role in bioinformatics, DNA sequence matching, disease detection etc. This algorithm doesn't need backtracking and shifting more than one position. Hence, KMP algorithm can be called as an intelligent string search algorithm. It is also used in large amounts in the field of competitive programming.

10. Comparison with Similar Algorithms

There are a few similar algorithms like Rabin-Karp string search algorithm, Boyer-Moore string search algorithm and the naive string search algorithm.

Name Of Algorithm	Best Case	Worst Case	Average Case	Space	Technique
KMP Algorithm	O(n)	O(n+m)	O(n)	O(m)	Checks string pattern within string text
Rabin-Karp	O(n+m)	O(n+m)	O(n+m)	O(m)	Uses hashing
Boyer-Moore	Ω(n)	O(mn)	O(n)	O(n+m)	Good suffix shift and bad character shift
Naive String Search Algorithm	O(n)	O(mn)	O(n+m)	O(1)	Characters of pattern compared to substring of text

11. Conclusion:

KMP Algorithm is the most widely used and most advantageous for organizations around the world due to its capacity of being the worst-case fault resistant algorithm. It’s beneficial for anyone wanting to make a career in Data Science in general and Search in particular.