Assignment 02 | Advance Algorithms CE-092

Assignment submission for Advance Algorithms subject week 2.

nevilparmar24@gmail.com

Task 1:

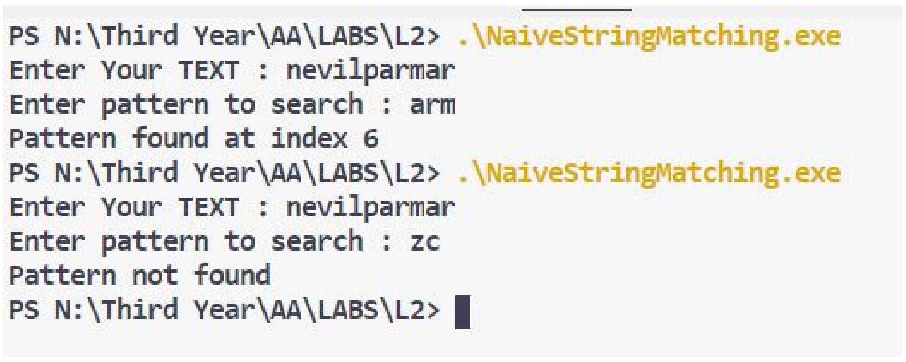
Implement naive string matching algorithm, brute force approach.

Code:

```
/*
* @Author: nevil
* @Date:   2020-07-17 15:35:27
* @Last Modified by:   nevil
* @Last Modified time: 2020-07-26 14:59:50
*/
#include <bits/stdc++.h>
using namespace std;

void naiveMatch(string pat, string txt)
{
    int M = pat.length();
    int N = txt.length();
    bool found = 0;

    for (int i = 0; i <= N - M; i++) {
        int j;

        for (j = 0; j < M; j++)
```

```
            if (txt[i + j] != pat[j])
                break;

        if (j == M) {
            cout << "Pattern found at index "
                 << i << endl;
            found = 1;
        }
    }

    if(!found)
        cout << "Pattern not found" << endl;
}

int main()
{
    string text, pattern;
    cout << "Enter Your TEXT : ";
    cin >> text;
    cout << "Enter pattern to search : ";
    cin >> pattern;

    naiveMatch(pattern, text);
    return 0;
}
```

Output:



```
PS N:\Third Year\AA\LABS\L2> .\NaiveStringMatching.exe
Enter Your TEXT : nevilparmar
Enter pattern to search : arm
Pattern found at index 6
PS N:\Third Year\AA\LABS\L2> .\NaiveStringMatching.exe
Enter Your TEXT : nevilparmar
Enter pattern to search : zc
Pattern not found
PS N:\Third Year\AA\LABS\L2>
```

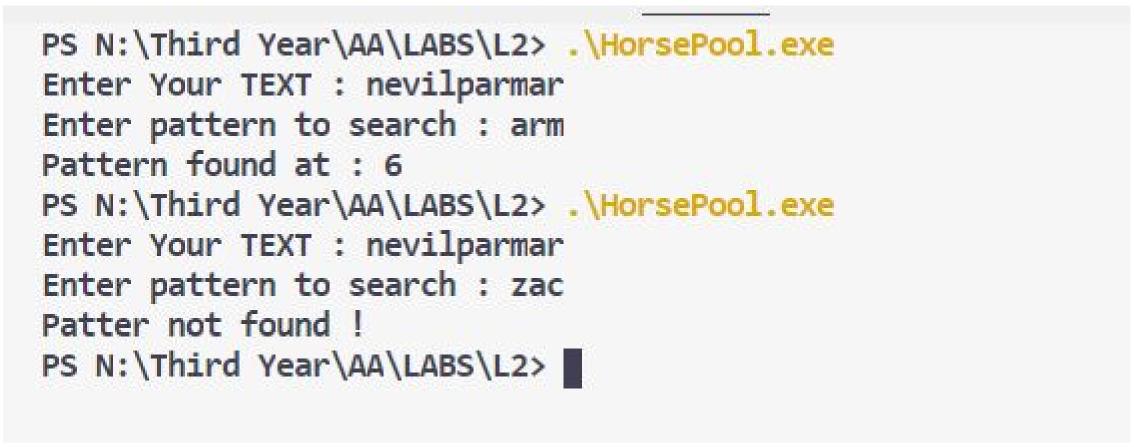
Task 2:

Implement horspool string matching algorithm.

Code :

```
/*
* @Author: nevil
* @Date:   2020-07-17 16:05:53
* @Last Modified by:   nevil
* @Last Modified time: 2020-07-26 15:01:56
*/
#include<bits/stdc++.h>
using namespace std;


int horspoolStringMatching(const string &P, const
string &T) {

    // P = input text
    // T = pattern to be searched in the text

    // shift table for algo
    int table[126];
    int k = 0, flag = 0;

    int m = T.length();
    int n = P.length();
    int i, j;


    for (i = 0; i < 126; i++)
        table[i] = m;
    for (j = 0; j < m - 1; j++)
        table[T[j]] = m - 1 - j;

    i = m - 1;

    while (i <= n - 1) {

        k = 0;
        while (k <= m - 1 && T[m - 1 - k] == P[i - k])
{
            k++;
        }

        if (k == m) {
            return i - m + 1;
            flag = 1;
            break;
        }

        else {
            i = i + table[P[i]];
        }
    }

    // returns -1 if the pattern is not found in the
given input string
    if (!flag)
        return -1;

}

int main()
{
    string text, pattern;
    cout << "Enter Your TEXT : ";
    cin >> text;
    cout << "Enter pattern to search : ";
    cin >> pattern;

    int index = horspoolStringMatching(text, pattern);
    if(index != -1)
        cout << "Pattern found at : " << index << endl
;
    else
        cout << "Patter not found !" << endl;
    return 0;
}
```

Output:



```
PS N:\Third Year\AA\LABS\L2> .\HorsePool.exe
Enter Your TEXT : nevilparmar
Enter pattern to search : arm
Pattern found at : 6
PS N:\Third Year\AA\LABS\L2> .\HorsePool.exe
Enter Your TEXT : nevilparmar
Enter pattern to search : zac
Patter not found !
PS N:\Third Year\AA\LABS\L2>
```

Conclusion :

The Boyer-Moore-Horspool algorithm execution time is linear in the size of the string being searched. It can have a lower execution time factor than many other search algorithms.

For one, it does not need to check all characters of the string. It skips over some of them with the help of the Bad Match table.

The algorithm gets faster as the substring being searched for becomes longer. This is because with each unsuccessful attempt to find a match between the substring and the string, the algorithm uses the Bad Match table to rule out positions where the substring cannot match.

Complexity :

The Naive approach for string matching compare the pattern with all the characters in given text hence its time complexity is O(m*n).

Horspool's algorithm for string matching requires some pre-processing. In the worst-case the performance of the Boyer-Moore-Horspool algorithm is O(mn), where m is the length of the substring and n is the length of the string.

The average time is O(n).