Experiment - 9

Student Name: Manjot Singh
UID: 23BCS12549
Branch: BE-CSE
Section/Group: KRG_2B
Semester: 5th
Date of Performance: 30/10/25
Subject Name: DAA
Subject Code: 23CSH-303

1. Problem Description/Aim:

To implement and execute the **Knuth-Morris-Pratt (KMP) algorithm** in C++ to efficiently find all occurrences of a given pattern string (`P`) within a larger text string (`S`).

2. Objective:

- Understand the core principle of the KMP algorithm: **avoiding redundant comparisons** by utilizing prefix information.
- Implement the **Longest Proper Prefix which is also a Suffix (LPS) array** computation function.
- Implement the main KMP search logic to achieve linear time complexity, $O(M+N)$, where N is the length of the text and M is the length of the pattern.
- Clearly display the text, the pattern, and all starting indices where the pattern is found.

3. CODE

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

void computeLPSArray(const string& pat, int M, vector<int>& lps) {
    int len = 0;
    lps[0] = 0;
    int i = 1;

    while (i < M) {
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
```

```
        } else {
            if (len != 0) {
                len = lps[len - 1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
}

void KMPSearch(const string& pat, const string& txt) {
    int M = pat.length();
    int N = txt.length();
    vector<int> lps(M);

    computeLPSArray(pat, M, lps);

    int txtIdx = 0;
    int patIdx = 0;
    vector<int> found_indices;

    while ((N - txtIdx) >= (M - patIdx)) {
        if (pat[patIdx] == txt[txtIdx]) {
            txtIdx++;
            patIdx++;
        }

        if (patIdx == M) {
            found_indices.push_back(txtIdx - patIdx);
            patIdx = lps[patIdx - 1];
        } else if (txtIdx < N && pat[patIdx] != txt[txtIdx]) {
            if (patIdx != 0) {
                patIdx = lps[patIdx - 1];
            } else {
                txtIdx++;
            }
        }
    }
```

```
    cout << "String S: " << txt << endl;
    cout << "Pattern P: " << pat << endl;

    if (found_indices.empty()) {
        cout << "Pattern not found in the string." << endl;
    } else {
        cout << "Pattern found at the following indices:" << endl;
        for (int index : found_indices) {
            cout << index << endl;
        }
    }
}

int main() {
    string txt = "ABABDABACDABABCABAB";
    string pat = "ABABCABAB";

    KMPSearch(pat, txt);

    return 0;
}
```

Output:

Output 1:

```
PS C:\cu ppt and notes\sem 5\DAA Experiments\Experiment 9> cd
($?) { .\Exp9 }
String S: ABABDABACDABABCABAB
Pattern P: ABABCABAB
Pattern found at the following indices:
10
PS C:\cu ppt and notes\sem 5\DAA Experiments\Experiment 9>
```

Learning Outcomes:

- Successfully implemented a non-trivial, efficient string matching algorithm in C++.
- Gained a clear understanding of the **LPS array** construction and its purpose in optimizing search complexity from $O(NM)$ (naive) to **$O(N+M)$ (KMP)**.
- Developed modular C++ functions for the two core components of the KMP algorithm: computeLPSArray and KMPSearch.
- Enhanced procedural programming skills, including the use of std::vector for dynamic array management and string manipulation.
- Verified the algorithm's correctness by successfully finding the pattern's sole occurrence at the expected starting index.