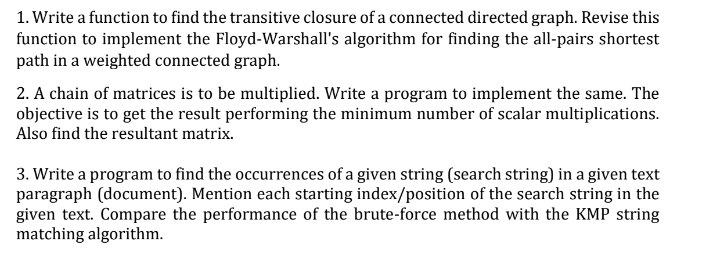
GRADED ASSIGNMENT -4

BY:D Veera Harsha Vardhan Reddy,210258



CODE:

import java.util.\*;

class KMP\_String\_Matching {

    void KMPSearch(String pat, String txt)

    {

        int M = pat.length();

        int N = txt.length();

        int lps[] = new int[M];

        int j = 0;

        computeLPSArray(pat, M, lps);

        int i = 0;

        while ((N - i) >= (M - j)) {

            if (pat.charAt(j) == txt.charAt(i)) {

                j++;

                i++;

            }

            if (j == M) {

                System.out.println("Found pattern "

                        + "at index " + (i - j));

                j = lps[j - 1];

            }

            else if (i < N && pat.charAt(j) != txt.charAt(i)) {

                if (j != 0)

                    j = lps[j - 1];

                else

                    i = i + 1;

            }

        }

    }

    void computeLPSArray(String pat, int M, int lps[])

    {

        int len = 0;

        int i = 1;

        lps[0] = 0;

        while (i < M) {

            if (pat.charAt(i) == pat.charAt(len)) {

                len++;

                lps[i] = len;

                i++;

            }

            else

            {

                if (len != 0) {

                    len = lps[len - 1];

                }

                else

                {

                    lps[i] = len;

                    i++;

                }

            }

        }

    }

    public static void main(String args[])

    {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the main string: ");

        String txt = sc.nextLine();

        System.out.println("Enter the string that you have to match: ");

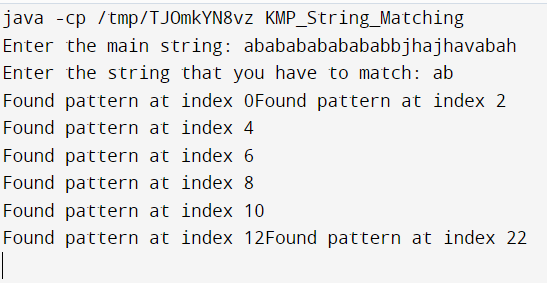
        String pat = sc.next();

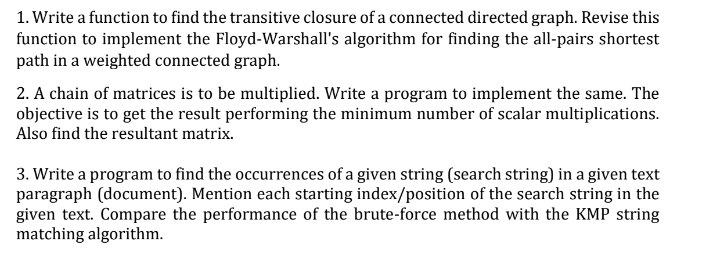
        new KMP\_String\_Matching().KMPSearch(pat, txt);

    }

}

OUTPUT:





CODE:

    #include <bits/stdc++.h>

    #include <iostream>

    using namespace std;

    #define V 4

    #define INF 99999

    void printSolution(int dist[][V]);

    void floydWarshall(int graph[][V])

    {

        int dist[V][V], i, j, k;

        for (i = 0; i < V; i++)

            for (j = 0; j < V; j++)

                dist[i][j] = graph[i][j];

        cout << "All Shortest distance matrix:"<< endl;

        cout<<endl;

        for (k = 0; k < V; k++) {

            for (i = 0; i < V; i++) {

                for (j = 0; j < V; j++) {

                    if (dist[i][j] > (dist[i][k] + dist[k][j])

                        && (dist[k][j] != INF

                            && dist[i][k] != INF))

                        dist[i][j] = dist[i][k] + dist[k][j];

                }

            }

            printSolution(dist);

        }

    }

    void printSolution(int dist[][V])

    {

        for (int i = 0; i < V; i++) {

            for (int j = 0; j < V; j++) {

                if (dist[i][j] == INF)

                    cout << "INF"

                        << "     ";

                else

                    cout << dist[i][j] << "  ";

            }

            cout << endl;

        }

        cout<< endl;

    }

    int main()

    {

        int graph[V][V] = { { 1, 6,INF,7 },

                            { 9, 0,2,INF },

                            { 5, INF, 3,1 },

                            { 2, INF, INF, 1 }

                            };

        floydWarshall(graph);

        return 0;

    }

OUTPUT:

