Find orientation of a pattern in a matrix

Given a matrix of characters and a pattern, find the orientation of pattern in the matrix. In other words, find if pattern appears in matrix in horizontal or vertical direction. Achieve this in minimum time possible.

```
Input:
mat[N][N] = { {'a', 'b', 'c', 'd', 'e'},
              {'f', 'g', 'h', 'i', 'j'},
              {'k', 'l', 'm', 'n', 'o'},
              {'p', 'q', 'r', 's', 't'},
              {'u', 'v', 'w', 'x', 'y'}};
pattern = "pqrs";

Output: Horizontal
```

We strongly recommend you to minimize your browser and try this yourself first.

A simple solution is for each row and column, use Naive pattern searching algorithm to find the orientation of pattern in the matrix. The time complexity of Naive pattern searching algorithm for every row is O(NM) where N is size of the matrix and M is length of the pattern. So, the time complexity of this solution will be **O(N*(NM))** as each of N rows and N columns takes O(NM) time.

Can we do better?

The idea is to use KMP pattern matching algorithm for each row and column. The KMP matching algorithm improves the worst case to O(N + M). The total cost of a KMP search is linear in the number of characters of string and pattern. For a N x N matrix and pattern of length M, complexity of this solution will be **O(N*(N+M))** as each of N rows and N columns will take O(N + M) time.

```
// C program for finding orientation of the pattern
// using KMP pattern searching algorithm
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define N 5

// Used in KMP Search for preprocessing the pattern
void computeLPSArray(char *pat, int M, int *lps)
{
    // length of the previous longest prefix suffix
    int len = 0;
    int i = 1;

    lps[0] = 0; // lps[0] is always 0

    // the loop calculates lps[i] for i = 1 to M-1
    while (i < M)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i++] = len;
        }
        else // (pat[i] != pat[len])
        {
            if (len != 0)
            {
                // This is tricky. Consider the example
                // AAACAAAA and i = 7.
                len = lps[len-1];

                // Also, note that we do not increment i here
            }
            else // if (len == 0)
            {
                lps[i++] = 0;
            }
            }
        }
    }
}

int KMPSearch(char *pat, char *txt)
{
    int M = strlen(pat);

    // create lps[] that will hold the longest prefix suffix
    // values for pattern
    int *lps = (int *)malloc(sizeof(int)*M);
    int j = 0; // index for pat[]

    // Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pat, M, lps);

    int i = 0; // index for txt[]
    while (i < N)
    {
        if (pat[j] == txt[i])
        {
            j++;
            i++;
        }
        if (j == M)
        {
            // return 1 is pattern is found
            return 1;
        }
        // mismatch after j matches
        else if (i < N && pat[j] != txt[i])
        {
            // Do not match lps[0..lps[j-1]] characters,
            // they will match anyway
            if (j != 0)
                j = lps[j-1];
            else
                i = i+1;
        }
    }
    free(lps); // to avoid memory leak
    // return 0 is pattern is not found
    return 0;
}

// Function to find orientation of pattern in the matrix
// It uses KMP pattern searching algorithm
void findOrientation(char mat[][N], char *pat)
{
    // allocate memory for string contaning cols
    char *col = (char*) malloc(N);

    for (int i = 0; i < N; i++)
    {
        // search in row i
        if (KMPSearch(pat, mat[i]))
        {
            printf("Horizontal\n");
            return;
        }

        // Construct an array to store i'th column
        for (int j = 0; j < N; j++)
            col[j] = *(mat[j] + i);

        // Search in column i
        if (KMPSearch(pat, col))
            printf("Vertical\n");
    }

    // to avoid memory leak
    free(col);
```

```
}

// Driver program to test above function
int main()
{
    char mat[N][N] =
    {
        {'a', 'b', 'c', 'd', 'e'},
        {'f', 'g', 'h', 'i', 'j'},
        {'k', 'l', 'm', 'n', 'o'},
        {'p', 'q', 'r', 's', 't'},
        {'u', 'v', 'w', 'x', 'y'}

    };
    char pat[] = "pqrs";

    findOrientation(mat, pat);

    return 0;
}
```

Output :

```
Horizontal
```