PROGRAM –

```
#include <stdio.h>
#include <string.h>
#define MAX 100
#include <time.h>

char p[MAX];
char t[MAX];
int cmp[MAX] = {0};
int comparison_count = 0;
int store;
int lastocc_table[256];
int lastoccurrence(char a) {
    int m = strlen(p);
    for (int i = m - 1; i >= 0; i--) {
        if (p[i] == a) {
            return i;
        }  }
    return -1;
}
void build_lastocc_table() {
    int m = strlen(p);
    for (int i = 0; i < 256; i++) lastocc_table[i] = -1;
    for (int i = 0; i < m; i++) lastocc_table[(unsigned char)p[i]] = i;
}
void print_lastocc_table() {
    int m = strlen(p);
    printf("\nLast Occurrence :\n");
    printf("Char: ");
    for (int i = 0; i < m; i++) printf("%4c", p[i]);
    printf("\nIdx : ");
    for (int i = 0; i < m; i++) printf("%4d", lastocc_table[(unsigned char)p[i]]);
    printf("\n\n");}
void display_comparison(int i, int j) {
    printf("Comparing t[%d] = '%c' with p[%d] = '%c' : ", i, t[i], j, p[j]);
    if (t[i] == p[j]) {
        printf("Equal\n");
    } else {
        printf("Not Equal\n");  }}
long long current_time_us()
{
    clock_t now = clock();
    return (long long)((double)now * 1000000.0 / CLOCKS_PER_SEC);
}
int min(int a, int b) {
    return (a <= b) ? a : b;
}
void print_text() {
    int n = strlen(t);
    printf("Pattern: %s\n", p);
    printf("    ");
    for (int i = 0; i < n; i++) {
        printf("%4d", i);
    }
    printf("\n");

    printf("    ");
    for (int i = 0; i < n; i++) {
        printf("----");
    }
    printf("\n");
    printf("    ");
    for (int i = 0; i < n; i++) {
        printf("|%3c", t[i]);    }
    printf("|\n");
    printf("    ");
    for (int i = 0; i < n; i++) {
        printf("----");   }
    printf("\n");}
void print_pattern(int i, int j, int lastocc) {
    int m = strlen(p);
    int n = strlen(t);
    printf("\n");
    for (int k = 0; k < (i - j + 1); k++) {
        printf("    ");     }
    for (int idx = 0; idx < m; idx++) {
        printf("|%3c", p[idx]);
    }
    printf("| i = %d lastocc = %d  ", i, lastocc);
    display_comparison(i, j);
    for (int k = 0; k < (i - j + 1); k++) {
        printf("    ");
    }
    for (int idx = 0; idx < m; idx++) {
        printf("|%3d", cmp[idx]);
    }
    printf("| j = %d\n", j);
    printf("Last Occurrence Table: ");
    for (int idx = 0; idx < m; idx++) {
        printf("%c:%d ", p[idx], lastocc_table[(unsigned char)p[idx]]);
    }
    printf("\n");
    printf("\n\n"); // 1-2 blank lines for clarity
```

```
}
void print_comparison_table() {
    int m = strlen(p);
    printf("\nTotal Comparisons Table:\n");
    printf("Char: ");
    for (int i = 0; i < m; i++) printf("%4c", p[i]);
    printf("\nCmp : ");
    for (int i = 0; i < m; i++) printf("%4d", cmp[i]);
    printf("\n");
}
int BM() {
    print_text();
    int m = strlen(p);
    int n = strlen(t);
    int i = m - 1;
    int j = m - 1;
    int flag = 1;
    do {
        comparison_count++;
        if (p[j] == t[i]) {
            cmp[j]++;
            if (j == 0) {
                return i;
            } else {
                i--;
                j--;  }
        } else {
            cmp[j]++;
            int lastocc = lastoccurrence(t[i]);
            store = n - i - (m - j);
            print_pattern(i, j, lastocc);
            i = i + m - min(j, lastocc + 1);
            j = m - 1;
```

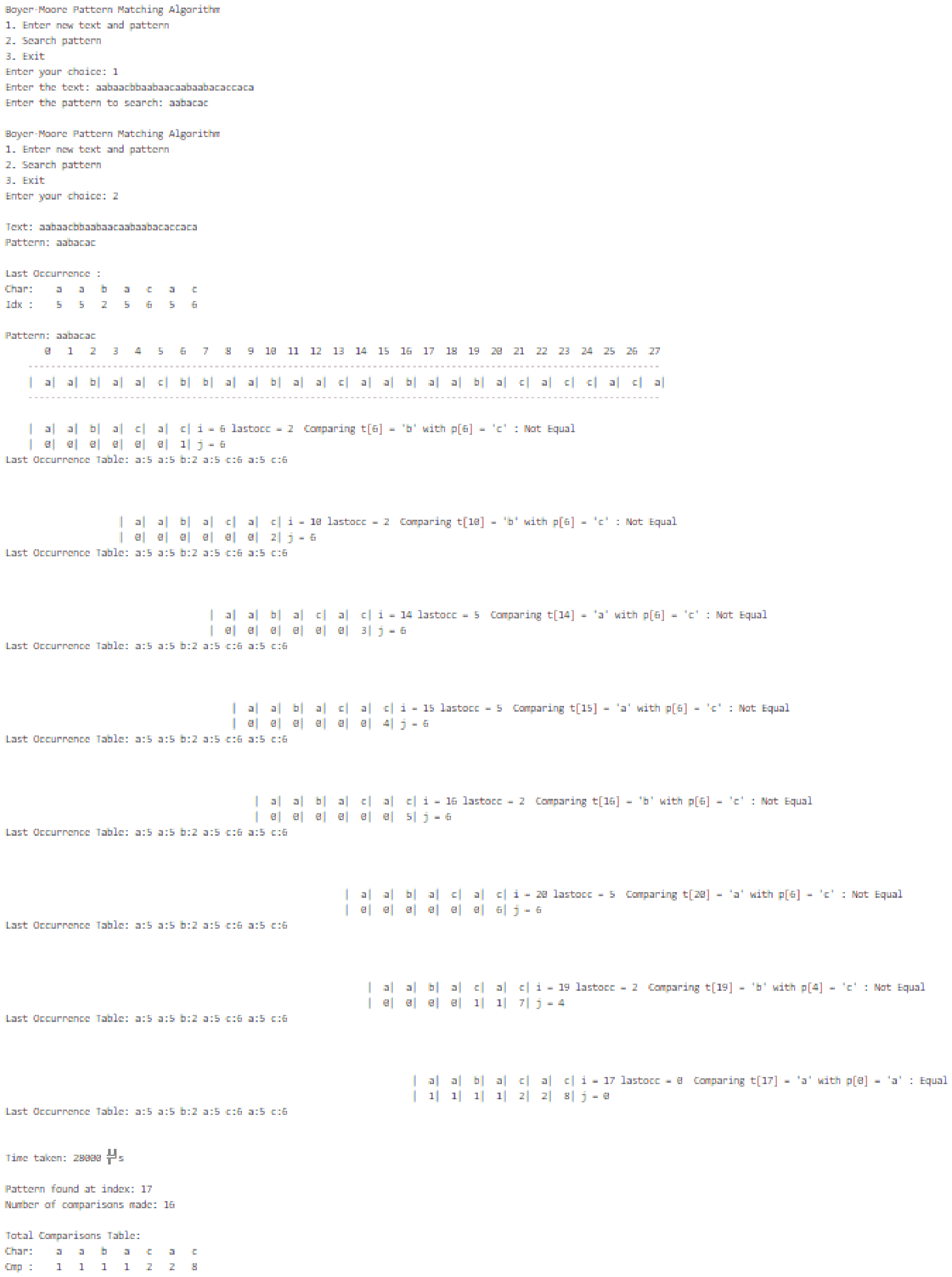
```
        }} while (i <= n - 1);
    return -1;
}
int main() {
    int choice;
    long long start_time, end_time;
    do {
        printf("\nBoyer-Moore Pattern Matching Algorithm\n");
        printf("1. Enter new text and pattern\n");
        printf("2. Search pattern\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        if (scanf("%d", &choice) != 1) {
            int ch;
            while ((ch = getchar()) != '\n' && ch != EOF);
            choice = -1;     }
        getchar();
        switch(choice) {
            case 1:
                printf("Enter the text: ");
                fgets(t, MAX, stdin);
                t[strcspn(t, "\n")] = 0;

                printf("Enter the pattern to search: ");
                fgets(p, MAX, stdin);
                p[strcspn(p, "\n")] = 0;
                break;
            case 2:
                if (strlen(t) == 0 || strlen(p) == 0) {
                    printf("Please enter text and pattern first!\n");
                    break;
                }
```

```
                comparison_count = 0;
                memset(cmp, 0, sizeof(cmp));
                build_lastocc_table();
                printf("\nText: %s\n", t);
                printf("Pattern: %s\n", p);
                print_lastocc_table();
                start_time = current_time_us();
                int i = BM();
                print_pattern(i, 0, 0);
                end_time = current_time_us();
                printf("Time taken: %lld µs\n", end_time - start_time);
                if (i != -1) {
                    printf("\nPattern found at index: %d\n", i);
                } else {
                    printf("\nPattern not found in the text\n");
                }
                printf("Number of comparisons made: %d\n", comparison_count);
                print_comparison_table();
                break;
            case 3:
                printf("Exiting program...\n");
                break;

            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 3);
    return 0;
}
```

OUTPUT –



```
Boyer-Moore Pattern Matching Algorithm
1. Enter new text and pattern
2. Search pattern
3. Exit
Enter your choice: 1
Enter the text: aabaacbbaabaacaabaabacaccaca
Enter the pattern to search: aabacac

Boyer-Moore Pattern Matching Algorithm
1. Enter new text and pattern
2. Search pattern
3. Exit
Enter your choice: 2

Text: aabaacbbaabaacaabaabacaccaca
Pattern: aabacac

Last Occurrence :
Char:    a   a   b   a   c   a   c
Idx :    5   5   2   5   6   5   6

Pattern: aabacac
       0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
    ----------------------------------------------------------------------------------------------------------------
    |  a|  a|  b|  a|  a|  c|  b|  b|  a|  a|  b|  a|  a|  c|  a|  a|  b|  a|  a|  b|  a|  c|  a|  c|  c|  a|  c|  a|
    ----------------------------------------------------------------------------------------------------------------


    |  a|  a|  b|  a|  c|  a|  c| i = 6 lastocc = 2  Comparing t[6] = 'b' with p[6] = 'c' : Not Equal
    |  0|  0|  0|  0|  0|  0|  1| j = 6
Last Occurrence Table: a:5 a:5 b:2 a:5 c:6 a:5 c:6



                    |  a|  a|  b|  a|  c|  a|  c| i = 10 lastocc = 2  Comparing t[10] = 'b' with p[6] = 'c' : Not Equal
                    |  0|  0|  0|  0|  0|  0|  2| j = 6
Last Occurrence Table: a:5 a:5 b:2 a:5 c:6 a:5 c:6



                                    |  a|  a|  b|  a|  c|  a|  c| i = 14 lastocc = 5  Comparing t[14] = 'a' with p[6] = 'c' : Not Equal
                                    |  0|  0|  0|  0|  0|  0|  3| j = 6
Last Occurrence Table: a:5 a:5 b:2 a:5 c:6 a:5 c:6



                                        |  a|  a|  b|  a|  c|  a|  c| i = 15 lastocc = 5  Comparing t[15] = 'a' with p[6] = 'c' : Not Equal
                                        |  0|  0|  0|  0|  0|  0|  4| j = 6
Last Occurrence Table: a:5 a:5 b:2 a:5 c:6 a:5 c:6



                                            |  a|  a|  b|  a|  c|  a|  c| i = 16 lastocc = 2  Comparing t[16] = 'b' with p[6] = 'c' : Not Equal
                                            |  0|  0|  0|  0|  0|  0|  5| j = 6
Last Occurrence Table: a:5 a:5 b:2 a:5 c:6 a:5 c:6



                                                            |  a|  a|  b|  a|  c|  a|  c| i = 20 lastocc = 5  Comparing t[20] = 'a' with p[6] = 'c' : Not Equal
                                                            |  0|  0|  0|  0|  0|  0|  6| j = 6
Last Occurrence Table: a:5 a:5 b:2 a:5 c:6 a:5 c:6



                                                                |  a|  a|  b|  a|  c|  a|  c| i = 19 lastocc = 2  Comparing t[19] = 'b' with p[4] = 'c' : Not Equal
                                                                |  0|  0|  0|  0|  1|  1|  7| j = 4
Last Occurrence Table: a:5 a:5 b:2 a:5 c:6 a:5 c:6



                                                                        |  a|  a|  b|  a|  c|  a|  c| i = 17 lastocc = 0  Comparing t[17] = 'a' with p[0] = 'a' : Equal
                                                                        |  1|  1|  1|  1|  2|  2|  8| j = 0
Last Occurrence Table: a:5 a:5 b:2 a:5 c:6 a:5 c:6


Time taken: 28000 µs

Pattern found at index: 17
Number of comparisons made: 16

Total Comparisons Table:
Char:    a   a   b   a   c   a   c
Cmp :    1   1   1   1   2   2   8
```

Conclusion: BM algorithm was implemented successfully in C .

DATE –

KMP ALGORITHM

Aim: C program to implement KMP algorithm.

Problem Statement:

The String Pattern Matching problem is to find the starting index of the first occurrence of a pattern string P in a given text string T. For each possible starting position in T, we need to determine if the substring of T matches P.

Input:

- A string T (the text) of length n
- A string P (the pattern) of length m

Output:

- The starting index of the first substring of T matching P, or a message indicating P is not a substring of T

ALGORITHMS –

Algorithm KMPFailureFunction(P):

Input: String P (pattern) with m characters
Output: The failure function f for P, which maps *j* to the length of the longest prefix of *P* that is a suffix of P[1..j]

```
i ← 1

j ← 0

f(0) ← 0

while i < m do

  if P[j] = P[i] then

    {we have matched j + 1 characters}

    f(i) ← j + 1

    i ← i + 1

    j ← j + 1

  else if j > 0 then

    {j indexes just after a prefix of P that must match}
```

III] **Worst Case:**

Time Complexity: O(n)

➞ Even in the worst case, due to the failure function f, the algorithm never backtracks on i and progresses through the text linearly.

Space Complexity

I] **Best Case:**

Space Complexity: O(m)

➞ Only space used is for the failure function f[0...m-1].

II] **Average Case:**

Space Complexity: O(m)

➞ Space remains the same, as only the pattern length affects auxiliary memory.

III] **Worst Case:**

Space Complexity: O(m)

➞ No recursion or stack usage; space is dominated by the f[] array for the pattern.

PROGRAM –

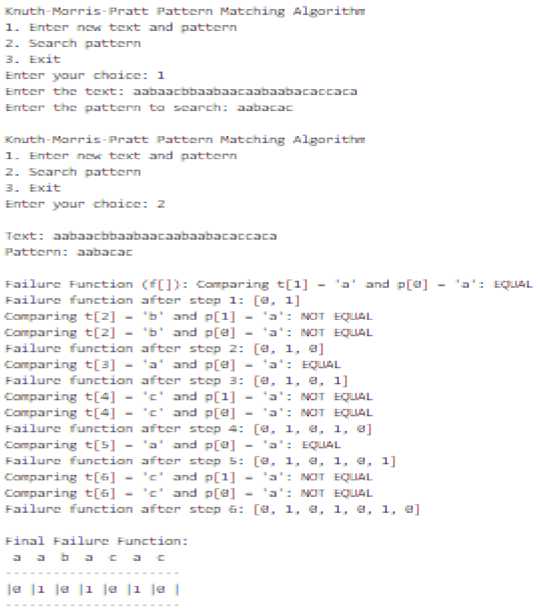
```
#include <stdio.h>
#include <string.h>
#define MAX 100
#include <time.h>
char p[MAX];
char t[MAX];
int cmp[MAX] = {0};
int comparison_count = 0;
int f[MAX];
int store;
void compare_reason(char a, char b, int i, int j) {
    printf("Comparing t[%d] = '%c' and p[%d] = '%c': ", i, a, j, b);
    if (a == b)
        printf("EQUAL\n");
    else
        printf("NOT EQUAL\n");
}
void display_failure_step(int m, int upto) {
    printf("Failure function after step %d: [", upto);
    for (int k = 0; k <= upto; k++) {
        printf("%d", f[k]);
        if (k < upto) printf(", ");
    }
    printf("]\n");
}
long long current_time_us(){
    return (long long)(clock() * 1000000LL / CLOCKS_PER_SEC);
}

void failureFunction(int m){
    f[0] = 0;
    int i = 1, j = 0;
    printf("\nFailure Function (f[]): ");
    while (i < m) {
        // Show comparison reason
        compare_reason(p[i], p[j], i, j);
        if (p[i] == p[j]) {
            f[i] = j + 1;
            display_failure_step(m, i);
            i++;
            j++;    }
        else if (j > 0) {
            j = f[j - 1];    }
        else {
            f[i] = 0;
            display_failure_step(m, i);
            i++;
        }
    }
    printf("\nFinal Failure Function:\n");
    for (int k = 0; k < m; k++)
        printf("%2c ", p[k]);
    printf("\n");
    for (int k = 0; k < m; k++)
        printf("---");
    printf("-\n|");
    for (int k = 0; k < m; k++)
        printf("%d |", f[k]);
    printf("\n");
    for (int k = 0; k < m; k++)
        printf("---");
    printf("-\n");
    printf("\n");
}
void print_pattern(int i, int j){
    int m = strlen(p);
    int n = strlen(t);
    printf("\n");
    for (int k = 0; k < (i - j + 1); k++)    {
        printf("    ");
    }
    for (int idx = 0; idx < m; idx++)    {
        printf("|%3c", p[idx]);
    }
    printf("| i = %d\n", i);
    for (int k = 0; k < (i - j + 1); k++)    {
        printf("    ");
    }
    for (int idx = 0; idx < m; idx++)    {
        printf("|%3d", cmp[idx]);
    }
    printf("| j = %d\n", j);
}


void print_text(){
    int n = strlen(t);
    printf("Pattern: %s\n", p);
    printf("    ");
    for (int i = 0; i < n; i++)    {
        printf("%4d", i);
    }
    printf("\n");
    printf("    ");
    for (int i = 0; i < n; i++)    {
        printf("----");
    }
    printf("\n");
    printf("    ");
    for (int i = 0; i < n; i++)    {
        printf("|%3c", t[i]);
    }
    printf("|\n");
    printf("    ");
    for (int i = 0; i < n; i++)    {
        printf("----");
    }
    printf("\n");
}

int KMP(){
    int m = strlen(p);
    int n = strlen(t);
    failureFunction(m);
    print_text( );
    int i = 0, j = 0;
    while (i < n){
        comparison_count++;
        compare_reason(t[i], p[j], i, j);
        if (t[i] == p[j])       {
            cmp[j]++;
            if (j == m - 1)
            {
                return i - m + 1; }
            i++;
            j++;
        }
        else if (j > 0)   {
            store = i;
            cmp[j]++;
            print_pattern( i, j);
            j = f[j - 1];
        }
        else       {
            cmp[j]++;
            print_pattern( i, j);
            i++;
        }
    }
    return -1;
}
int main(){
    int choice;
    long long start_time, end_time;
    do    {
        printf("\nKnuth-Morris-Pratt Pattern Matching Algorithm\n");
        printf("1. Enter new text and pattern\n");
        printf("2. Search pattern\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();
        switch (choice)       {
        case 1:
            printf("Enter the text: ");
            fgets(t, MAX, stdin);
            t[strcspn(t, "\n")] = 0;
            printf("Enter the pattern to search: ");
            fgets(p, MAX, stdin);
            p[strcspn(p, "\n")] = 0;
            break;
        case 2:
            if (strlen(t) == 0 || strlen(p) == 0)
            {
                printf("Please enter text and pattern first!\n");
                break;
            }
            comparison_count = 0;
            memset(cmp, 0, sizeof(cmp));
            printf("\nText: %s\n", t);
            printf("Pattern: %s\n", p);
            start_time = current_time_us();
            int i = KMP();
            print_pattern( i, 0);
            end_time = current_time_us();
            printf("Time taken: %lld µs\n", end_time - start_time);
            if (i != -1)          {
                printf("\nPattern found at index: %d\n", i);
            }
            else          {
                printf("\nPattern not found in the text\n");
            }
            printf("Number of comparisons made: %d\n", comparison_count);
            break;
        case 3:
            printf("Exiting program...\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 3);
    return 0;
}
```

OUTPUT



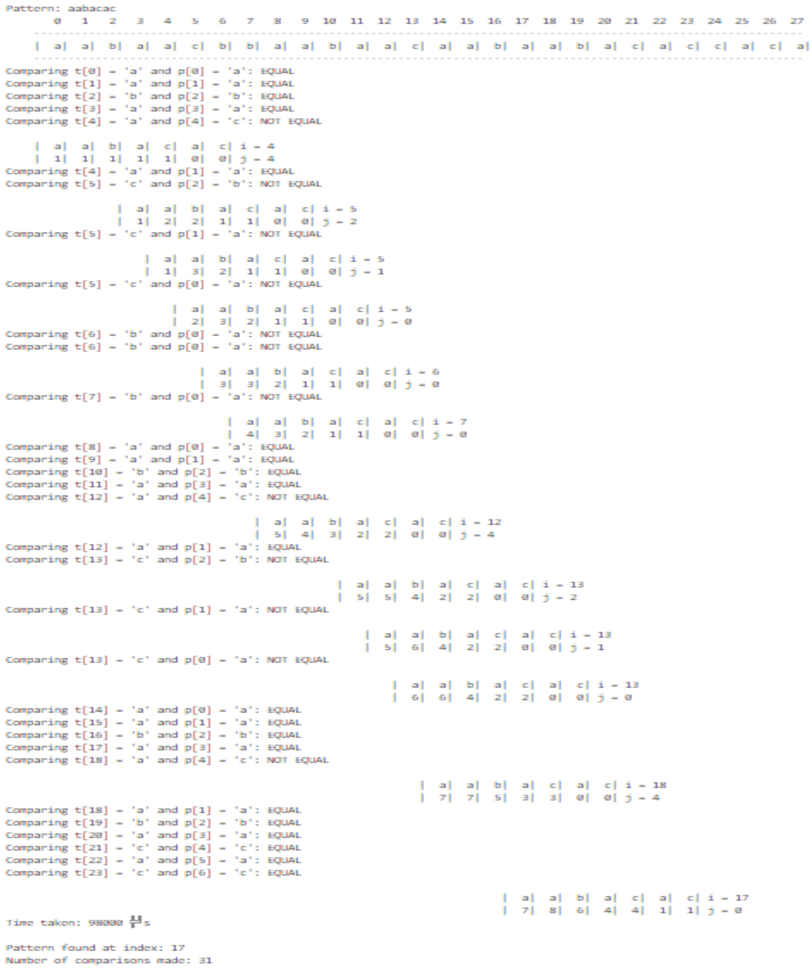
```
Knuth-Morris-Pratt Pattern Matching Algorithm
1. Enter new text and pattern
2. Search pattern
3. Exit
Enter your choice: 1
Enter the text: aabaacbbaabaacaabaabacaccaca
Enter the pattern to search: aabacac

Knuth-Morris-Pratt Pattern Matching Algorithm
1. Enter new text and pattern
2. Search pattern
3. Exit
Enter your choice: 2

Text: aabaacbbaabaacaabaabacaccaca
Pattern: aabacac

Failure Function (f[]): Comparing t[1] = 'a' and p[0] = 'a': EQUAL
Failure function after step 1: [0, 1]
Comparing t[2] = 'b' and p[1] = 'a': NOT EQUAL
Comparing t[2] = 'b' and p[0] = 'a': NOT EQUAL
Failure function after step 2: [0, 1, 0]
Comparing t[3] = 'a' and p[0] = 'a': EQUAL
Failure function after step 3: [0, 1, 0, 1]
Comparing t[4] = 'c' and p[1] = 'a': NOT EQUAL
Comparing t[4] = 'c' and p[0] = 'a': NOT EQUAL
Failure function after step 4: [0, 1, 0, 1, 0]
Comparing t[5] = 'a' and p[0] = 'a': EQUAL
Failure function after step 5: [0, 1, 0, 1, 0, 1]
Comparing t[6] = 'c' and p[1] = 'a': NOT EQUAL
Comparing t[6] = 'c' and p[0] = 'a': NOT EQUAL
Failure function after step 6: [0, 1, 0, 1, 0, 1, 0]

Final Failure Function:
 a  a  b  a  c  a  c
----------------------
|0 |1 |0 |1 |0 |1 |0 |
----------------------
```



```
Pattern: aabacac
       0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
    ----------------------------------------------------------------------------------------------------------------
    |  a|  a|  b|  a|  a|  c|  b|  b|  a|  a|  b|  a|  a|  c|  a|  a|  b|  a|  a|  b|  a|  c|  a|  c|  c|  a|  c|  a|
    ----------------------------------------------------------------------------------------------------------------
Comparing t[0] = 'a' and p[0] = 'a': EQUAL
Comparing t[1] = 'a' and p[1] = 'a': EQUAL
Comparing t[2] = 'b' and p[2] = 'b': EQUAL
Comparing t[3] = 'a' and p[3] = 'a': EQUAL
Comparing t[4] = 'a' and p[4] = 'c': NOT EQUAL

    |  a|  a|  b|  a|  c|  a|  c| i = 4
    |  1|  1|  1|  1|  1|  0|  0| j = 4
Comparing t[4] = 'a' and p[1] = 'a': EQUAL
Comparing t[5] = 'c' and p[2] = 'b': NOT EQUAL

                |  a|  a|  b|  a|  c|  a|  c| i = 5
                |  1|  2|  2|  1|  1|  0|  0| j = 2
Comparing t[5] = 'c' and p[1] = 'a': NOT EQUAL

                    |  a|  a|  b|  a|  c|  a|  c| i = 5
                    |  1|  3|  2|  1|  1|  0|  0| j = 1
Comparing t[5] = 'c' and p[0] = 'a': NOT EQUAL

                        |  a|  a|  b|  a|  c|  a|  c| i = 5
                        |  2|  3|  2|  1|  1|  0|  0| j = 0
Comparing t[6] = 'b' and p[0] = 'a': NOT EQUAL
Comparing t[6] = 'b' and p[0] = 'a': NOT EQUAL

                            |  a|  a|  b|  a|  c|  a|  c| i = 6
                            |  3|  3|  2|  1|  1|  0|  0| j = 0
Comparing t[7] = 'b' and p[0] = 'a': NOT EQUAL

                                |  a|  a|  b|  a|  c|  a|  c| i = 7
                                |  4|  3|  2|  1|  1|  0|  0| j = 0
Comparing t[8] = 'a' and p[0] = 'a': EQUAL
Comparing t[9] = 'a' and p[1] = 'a': EQUAL
Comparing t[10] = 'b' and p[2] = 'b': EQUAL
Comparing t[11] = 'a' and p[3] = 'a': EQUAL
Comparing t[12] = 'a' and p[4] = 'c': NOT EQUAL

                                    |  a|  a|  b|  a|  c|  a|  c| i = 12
                                    |  5|  4|  3|  2|  2|  0|  0| j = 4
Comparing t[12] = 'a' and p[1] = 'a': EQUAL
Comparing t[13] = 'c' and p[2] = 'b': NOT EQUAL

                                                |  a|  a|  b|  a|  c|  a|  c| i = 13
                                                |  5|  5|  4|  2|  2|  0|  0| j = 2
Comparing t[13] = 'c' and p[1] = 'a': NOT EQUAL

                                                    |  a|  a|  b|  a|  c|  a|  c| i = 13
                                                    |  5|  6|  4|  2|  2|  0|  0| j = 1
Comparing t[13] = 'c' and p[0] = 'a': NOT EQUAL

                                                        |  a|  a|  b|  a|  c|  a|  c| i = 13
                                                        |  6|  6|  4|  2|  2|  0|  0| j = 0
Comparing t[14] = 'a' and p[0] = 'a': EQUAL
Comparing t[15] = 'a' and p[1] = 'a': EQUAL
Comparing t[16] = 'b' and p[2] = 'b': EQUAL
Comparing t[17] = 'a' and p[3] = 'a': EQUAL
Comparing t[18] = 'a' and p[4] = 'c': NOT EQUAL

                                                            |  a|  a|  b|  a|  c|  a|  c| i = 18
                                                            |  7|  7|  5|  3|  3|  0|  0| j = 4
Comparing t[18] = 'a' and p[1] = 'a': EQUAL
Comparing t[19] = 'b' and p[2] = 'b': EQUAL
Comparing t[20] = 'a' and p[3] = 'a': EQUAL
Comparing t[21] = 'c' and p[4] = 'c': EQUAL
Comparing t[22] = 'a' and p[5] = 'a': EQUAL
Comparing t[23] = 'c' and p[6] = 'c': EQUAL

                                                                        |  a|  a|  b|  a|  c|  a|  c| i = 17
                                                                        |  7|  8|  6|  4|  4|  1|  1| j = 0
Time taken: 98000 µs

Pattern found at index: 17
Number of comparisons made: 31
```

Conclusion: KMP algorithm was implemented successfully in C .