Lab -

DIP : Design and Analysis of Algorithm

AIM : WRITE PROGRAMS FOR FOR STRING MATCHING ALGORITHMS FOR FOLLOWING METHODOLOGIES..

1. ROBIN KARP METHOD.

2. KMP METHOD.

3. HORSPOOL METHOD.

4. FINITE AUTOMATA METHOD.

Name : Ambalia Harshit

Department : Computer Engineering (M.Tech sem II)

Roll no : MT001

Date : Jan 2024

Program 01 : ROBIN KARP METHOD.

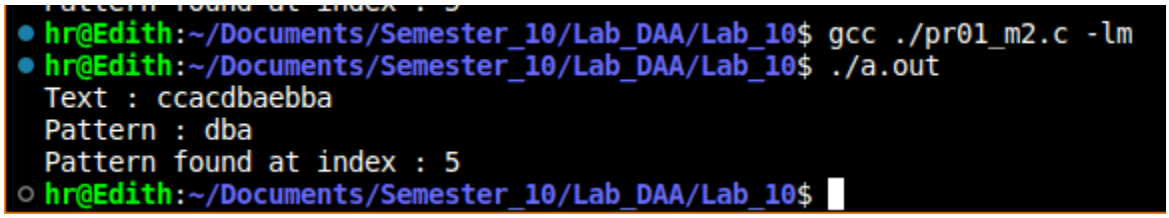
- Code :

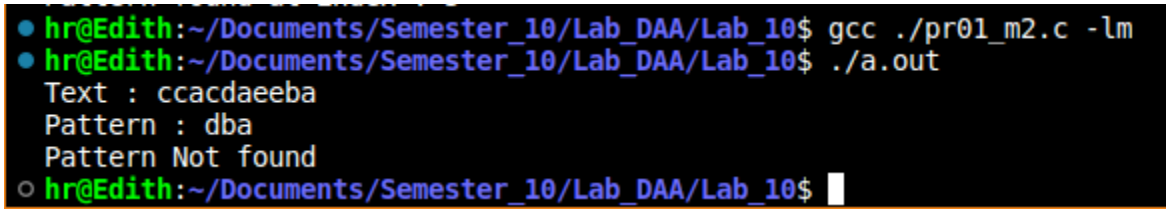
```
// ROBIN CARP METHOD for STRING MATCHING

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int find_code(char *mapping, char c) {
   for( int i=0;i<5;i++ )
       if(mapping[i]==c)
           return i+1;
}

int calculate_pattern_hash(char *pattern, char *mapping, int
mapping_length) {
   int pattern_len = strlen(pattern);
   int pattern_hash_code = 0;
   for( int i=0;i<pattern_len;i++ ){
       int temp = find_code(mapping, pattern[i]);
       pattern_hash_code += temp*pow(mapping_length, pattern_len-i-1);
   }
   return pattern_hash_code;
}

int my_robin_karp(char *str, char *pattern, char *mapping, int
mapping_length) {
   int str_len = strlen(str);
   int pattern_len = strlen(pattern);
   int pattern_hash_code = calculate_pattern_hash(pattern, mapping,
mapping_length);

   int temp_hash_code = 0;
   for( int i=0;i<str_len-pattern_len+1;i++ ) {
       for( int j=0;j<pattern_len;j++ ) {
           // printf(" %c-%d %d ", str[i+j], pattern_len-j-1, i);
           if( i==0 ) {
```

```
                int temp = find_code(mapping, str[i+j]);
                temp_hash_code += temp*pow(mapping_length,
pattern_len-j-1);
            }
            else {
                if( j==0 ) {
                    // printf(" i:%d j:%d ", i, j);
                    int temp_prev = find_code(mapping, str[i-1]);
                    temp_hash_code -= (temp_prev*pow(mapping_length,
pattern_len-1));
                    temp_hash_code = temp_hash_code*mapping_length;
                    int temp_next = find_code(mapping,
str[i+pattern_len-1]);
                    temp_hash_code += temp_next;
                }
            }
        }
        // printf(" Temp hash code : %d\n", temp_hash_code);
        if( temp_hash_code==pattern_hash_code ) {
            int isMatch = 1;
            for( int k=0;k<pattern_len;k++ ) {
                if(str[i+k]!=pattern[k]) {
                    isMatch = 0;
                    // printf("Hit but miss at index: %d", i+1);
                    break;
                }
            }
            if(isMatch==1){
                return i+1;
            }
        }
    }
    return -1;
}

int main() {
    int str_len = 11;
    char str[] = "ccacdbaebba";
    int pattern_len = 3;
    char pattern[] = "dba";
```

```
   char mapping[] = {'a', 'b', 'c', 'd', 'e'}; // mapping is done with the
index ie a-0, b-1, c-2, etc..
   int mapping_length = 5;



   int index = my_robin_karp(str, pattern, mapping, mapping_length);
   if( index!=-1 )
       printf("Pattern found at index : %d\n", index);
   else
       printf("Pattern Not found\n");
}
```

- Output Screen-shots / Tracing :



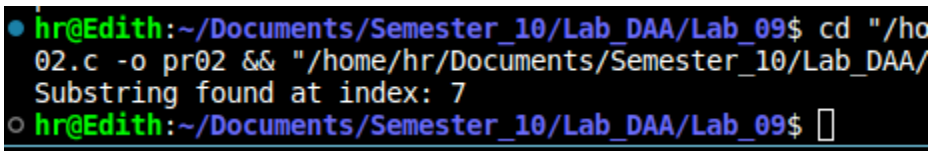


Program 02 : KMP METHOD.

- Code :

```c
// NAÏVE METHOD.(RECURSIVE)

#include <stdio.h>
#include <string.h>

int stringMatch(char *mainStr, char *substr, int index) {
   int mainLen = strlen(mainStr);
   int subLen = strlen(substr);

   if( index+subLen > mainLen )
       return -1;
   if( strncmp(mainStr+index, substr, subLen)==0 )
       return index;
   else
       return stringMatch(mainStr, substr, index + 1);
}

int main() {
   char mainStr[] = "Hello how are you";
   char substr[] = "how";

   int index = stringMatch(mainStr, substr, 0);
   if( index!=-1 )
       printf("Substring found at index: %d\n", index+1);
   else
       printf("Substring not found.\n");

   return 0;
}
```

- Output Screen-shots / Tracing :



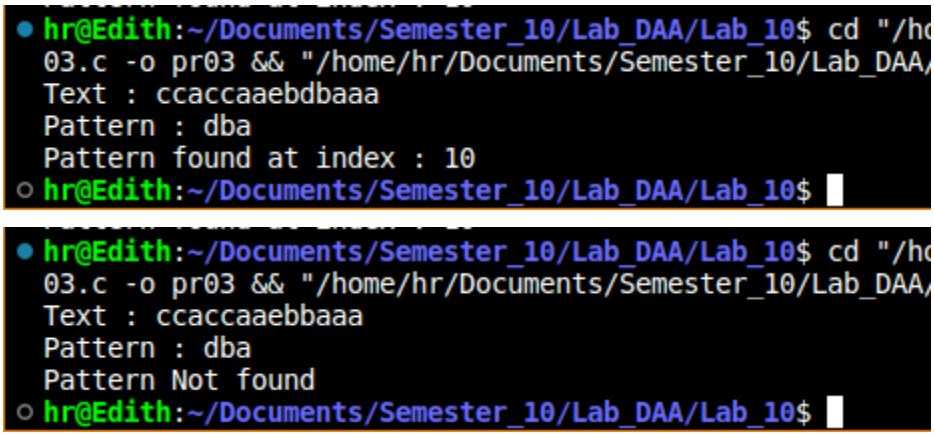
Program 03 : HORSPOOL METHOD.

- Code :

```
// HORSEPOOL METHOD for STRING MATCHING

#include <stdio.h>
#include <string.h>

void generate_shift_table(char *pattern, int *temp, int temp_len) {
   int m = strlen(pattern);
   for( int i=0;i<temp_len;i++ )
       temp[i] = m;
   for( int j=0;j<m-1;j++ )
       temp[pattern[j]] = m-j-1;
}

int my_horse_pool_algo(char *str, char *pattern, int *t) {
   int str_len = strlen(str);
   int pattern_len = strlen(pattern);
   int i = pattern_len-1;
   while( i<str_len ) {
       int temp = 0;
       while( (temp<pattern_len) && (pattern[pattern_len-temp-1]==str[i -
temp]) )
           temp++;
       if( temp==pattern_len )
           return (i-pattern_len+1);
       else
           i = i+t[str[i]];
   }
   return -1;
}

void main() {
   char str[100] = "ccaccaaebdbaaa", pattern[100] = "dba";
   int temp_len = 500;
   int temp[temp_len];

   generate_shift_table(pattern, temp, temp_len);
```

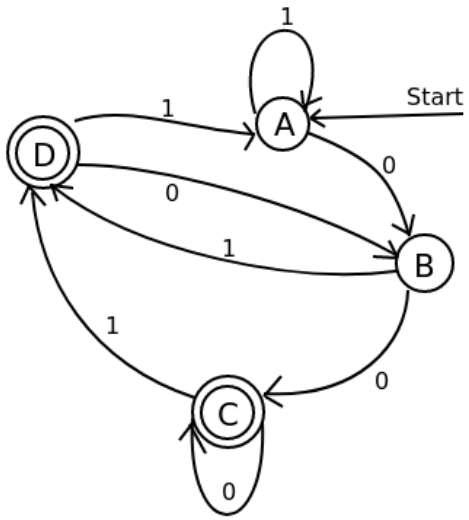
```
    // for( int i=0;i<500;i++ ) {
    //     printf("%d %d\n", i, temp[i]);
    // }

    int index = my_horse_pool_algo(str, pattern, temp);
    if( index >= 0)
        printf("Pattern found at index : %d\n", index + 1);
    else
        printf("Pattern Not found\n");
}
```

- Output Screen-shots / Tracing :



Program 04 : FINITE AUTOMATA METHOD.

- Program 01 :
    - This is not an actual finite automata method for string matching but this checks for the validation of the string for given automata.
    - This code is implemented for given finite automata

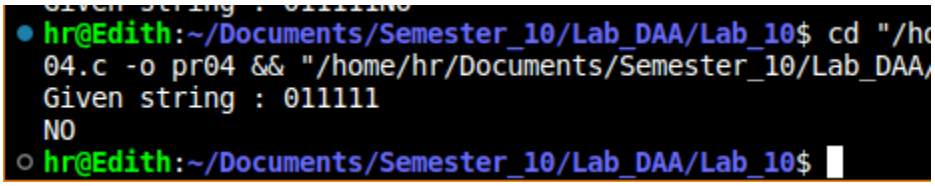


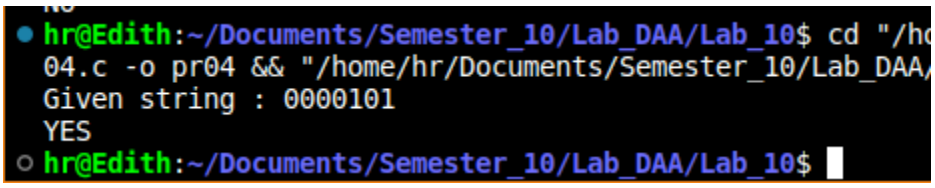
- Code :

```
// FINITE AUTOMATA METHOD for STRING MATCHING

#include <stdio.h>
#include <string.h>

int main() {
   char mapping[] = {'A', 'B', 'C'};
   char start = 'A';
   char end = 'C';
   int nodes = 3;
   char arr[][2] = {
       {'B', 'A'},
       {'C', 'A'},
       {'C', 'C'}
   };
```

```
    char str[] = "011111";

    int current = 0;
    int i;
    for (i = 0; str[i] != '\0'; i++) {
        int index = str[i] - '0';
        current = arr[current][index] - 'A';
    }

    printf("Given string : %s\n", str);
    if (mapping[end-'A'] == mapping[current]) {
        printf("YES\n");
    } else {
        printf("NO\n");
    }

    return 0;
}
```

- Output Screen-shots / Tracing :





- Program 02 :
    - This is an actual program for pattern matching.
    - Code :

```
#include<stdio.h>
#include<string.h>

int find_next_state(char *pattern, int cur_state, int character) {
   int pattern_len = strlen(pattern);

   if( cur_state<pattern_len && character==pattern[cur_state] )
       return cur_state+1;

   for( int nextState=cur_state, i=0;nextState>0;nextState-- ) {
       if( pattern[nextState-1]==character ) {
           for( i=0;i<nextState-1;i++ ) {
               if( pattern[i]!=pattern[cur_state-nextState+i+1])
                   break;
           }
           if( i==nextState-1 )
               return nextState;
       }
   }
   return 0;
}

int my_finite_automata_algo(char *pattern, char *text, int char_len) {
   int pattern_length = strlen(pattern);
   int text_length = strlen(text);

   int tr_table[pattern_length+1][char_len];
   for( int cur_state=0;cur_state<=pattern_length;cur_state++ )
       for( int character=0;character<char_len;character++ )
           tr_table[cur_state][character] = find_next_state(pattern,
cur_state, character);
```

```
    int cur_state = 0;
    for( int i=0;i<text_length;i++ ) {
        cur_state = tr_table[cur_state][text[i]];
        if( cur_state==pattern_length ) {
            return i-pattern_length+1;
        }
    }
    return -1;
}

int main() {
    char *text = "ccaccadbaaebbaaa";
    int char_len = 256;
    char *pattern = "dba";

    int index = my_finite_automata_algo(pattern, text, char_len);

    printf("Text : %s\n", text);
    printf("Pattern : %s\n", pattern);
    if( index!=-1 )
        printf("Pattern found at index : %d\n", index+1);
    else
        printf("Pattern Not found\n");
    return 0;
}
```

- Output Screen-shots / Tracing :