Lab -

DIP : Design and Analysis of Algorithm

AIM : Write a program to solve the following Algorithm

1. NAÏVE METHOD.(ITERATIVE)
2. NAÏVE METHOD.(RECURSIVE)
3. STRING EDIT DISTANCE(DISPLAY ONLY DISTANCE COUNT)
4. STRING EDIT DISTANCE(DISPLAY DISTANCE COUNT & REQUIRED OPERATION AT PARTICULAR LOCATION/INDEX)

Name : Ambalia Harshit

Department : Computer Engineering (M.Tech sem II)

Roll no : MT001

Date : Jan 2024

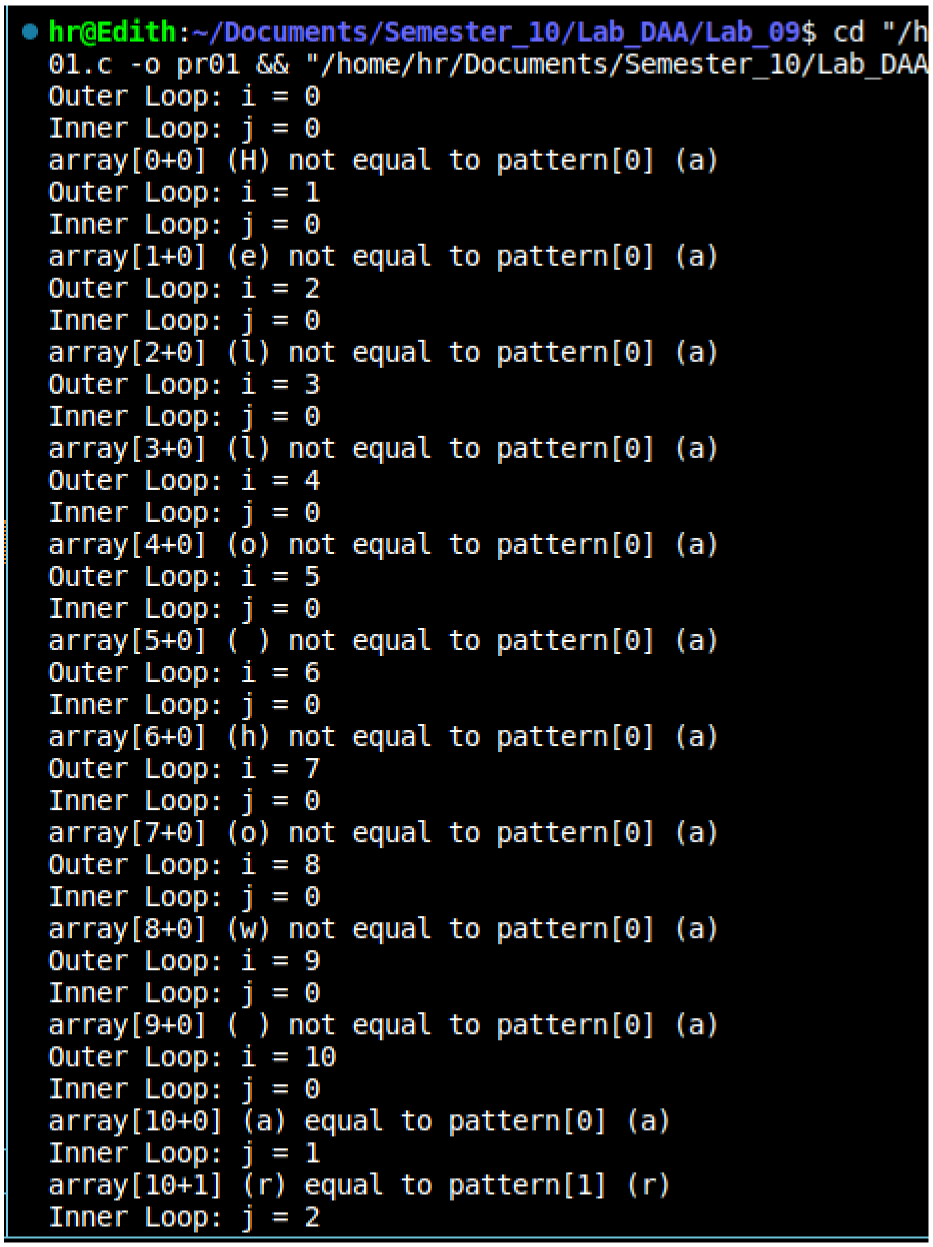
Program 01 : NAÏVE METHOD.(ITERATIVE)

- Code :

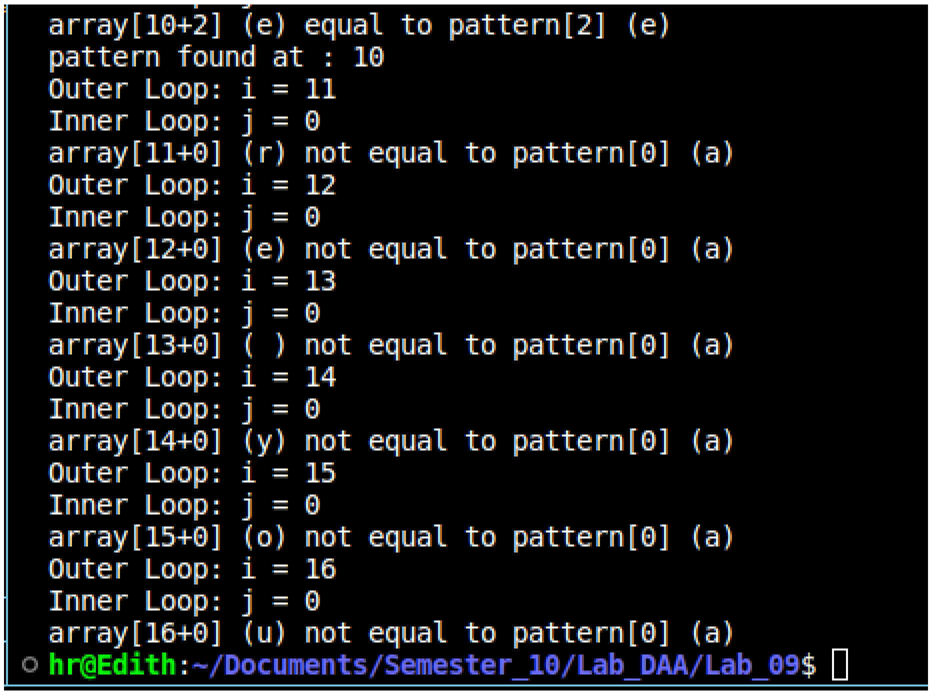
```c
// NAÏVE METHOD.(ITERATIVE)

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

void string_match(char array[], char pattern[]) {
   int array_len = strlen(array);
   int pattern_len = strlen(pattern);
   if (pattern_len>array_len) {
       printf("String not found");
       exit(0);
   }
   for( int i=0;i<array_len;i++ ) {
       printf("Outer Loop: i = %d\n", i);
       int j;
       for( j=0;j<pattern_len;j++ ) {
           printf("Inner Loop: j = %d\n", j);
           if( array[i+j]==pattern[j]) {
               printf("array[%d+%d] (%c) equal to pattern[%d] (%c)\n", i,
j, array[i + j], j, pattern[j]);
               continue;
           }
           else {
               printf("array[%d+%d] (%c) not equal to pattern[%d] (%c)\n",
i, j, array[i + j], j, pattern[j]);
               break;
           }
       }
       if(j==pattern_len) {
           printf("pattern found at : %d\n", i);
       }
   }
}
```

```c
int main() {
   char array[] = "Hello how are you";
   char pattern[] = "are";
   string_match(array, pattern);
}
```

- Output Screen-shots / Tracing :



```
● hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_09$ cd "/h
  01.c -o pr01 && "/home/hr/Documents/Semester_10/Lab_DAA
  Outer Loop: i = 0
  Inner Loop: j = 0
  array[0+0] (H) not equal to pattern[0] (a)
  Outer Loop: i = 1
  Inner Loop: j = 0
  array[1+0] (e) not equal to pattern[0] (a)
  Outer Loop: i = 2
  Inner Loop: j = 0
  array[2+0] (l) not equal to pattern[0] (a)
  Outer Loop: i = 3
  Inner Loop: j = 0
  array[3+0] (l) not equal to pattern[0] (a)
  Outer Loop: i = 4
  Inner Loop: j = 0
  array[4+0] (o) not equal to pattern[0] (a)
  Outer Loop: i = 5
  Inner Loop: j = 0
  array[5+0] ( ) not equal to pattern[0] (a)
  Outer Loop: i = 6
  Inner Loop: j = 0
  array[6+0] (h) not equal to pattern[0] (a)
  Outer Loop: i = 7
  Inner Loop: j = 0
  array[7+0] (o) not equal to pattern[0] (a)
  Outer Loop: i = 8
  Inner Loop: j = 0
  array[8+0] (w) not equal to pattern[0] (a)
  Outer Loop: i = 9
  Inner Loop: j = 0
  array[9+0] ( ) not equal to pattern[0] (a)
  Outer Loop: i = 10
  Inner Loop: j = 0
  array[10+0] (a) equal to pattern[0] (a)
  Inner Loop: j = 1
  array[10+1] (r) equal to pattern[1] (r)
  Inner Loop: j = 2
```



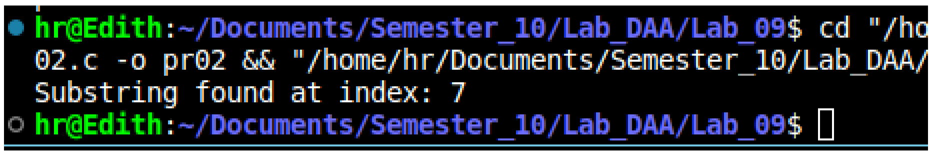
```
array[10+2] (e) equal to pattern[2] (e)
pattern found at : 10
Outer Loop: i = 11
Inner Loop: j = 0
array[11+0] (r) not equal to pattern[0] (a)
Outer Loop: i = 12
Inner Loop: j = 0
array[12+0] (e) not equal to pattern[0] (a)
Outer Loop: i = 13
Inner Loop: j = 0
array[13+0] ( ) not equal to pattern[0] (a)
Outer Loop: i = 14
Inner Loop: j = 0
array[14+0] (y) not equal to pattern[0] (a)
Outer Loop: i = 15
Inner Loop: j = 0
array[15+0] (o) not equal to pattern[0] (a)
Outer Loop: i = 16
Inner Loop: j = 0
array[16+0] (u) not equal to pattern[0] (a)
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_09$
```

Program 02 : NAÏVE METHOD.(RECURSIVE)

- 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 :



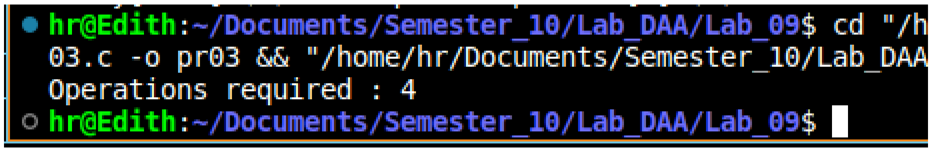
```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_09$ cd "/ho
02.c -o pr02 && "/home/hr/Documents/Semester_10/Lab_DAA/
Substring found at index: 7
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_09$ 
```

Program 03 : STRING EDIT DISTANCE(DISPLAY ONLY DISTANCE COUNT)

- Code :

```
// STRING EDIT DISTANCE(DISPLAY ONLY DISTANCE COUNT)

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

int my_minimum_of_three_integer(int x, int y, int z) {
   if (x <= y && x <= z) return x;
   if (y <= x && y <= z) return y;
   return z;
}

int my_minimum_edit_string_distance(char *init_string, char *final_string,
int init_length, int final_length, int **memo) {

   // Base cases
   if (init_length == 0)
       return final_length;
   if (final_length == 0)
       return init_length;

   // If already computed, return memoized value
   if (memo[init_length][final_length] != -1)
       return memo[init_length][final_length];

   // If last characters are same, ignore last characters and recur for
remaining strings
   if (init_string[init_length - 1] == final_string[final_length - 1])
       return memo[init_length][final_length] =
my_minimum_edit_string_distance(init_string, final_string, init_length-1,
final_length-1, memo);

   // If last characters are not same, consider all three operations
   return memo[init_length][final_length] = 1 +
my_minimum_of_three_integer(
       my_minimum_edit_string_distance(init_string, final_string,
init_length, final_length-1, memo), // Insert
       my_minimum_edit_string_distance(init_string, final_string,
init_length-1, final_length, memo), // Remove
       my_minimum_edit_string_distance(init_string, final_string,
init_length-1, final_length-1, memo) // Replace
   );
}

int main() {
   char init_string[] = "harshit";
   char final_string[] = "itharsh";
   int init_length = strlen(init_string);
   int final_length = strlen(final_string);

   int **memo = (int **)malloc((init_length + 1) * sizeof(int *));
   for (int i = 0; i <= init_length; i++) {
       memo[i] = (int *)malloc((final_length + 1) * sizeof(int));
       memset(memo[i], -1, (final_length + 1) * sizeof(int)); //
Initialize memoization table with -1
   }

   printf("Operations required : %d\n",
my_minimum_edit_string_distance(init_string, final_string, init_length,
final_length, memo));
   return 0;
}
```

- Output Screen-shots / Tracing :



```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_09$ cd "/h
03.c -o pr03 && "/home/hr/Documents/Semester_10/Lab_DAA
Operations required : 4
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_09$
```

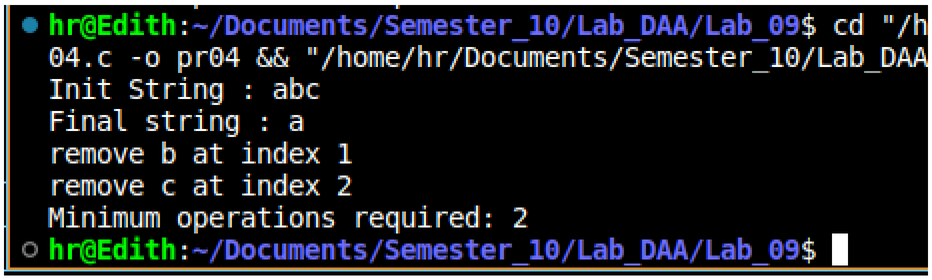
Program 04 : STRING EDIT DISTANCE(DISPLAY DISTANCE COUNT & REQUIRED OPERATION AT PARTICULAR LOCATION/INDEX)

- Code :

```
// STRING EDIT DISTANCE(DISPLAY DISTANCE COUNT & REQUIRED OPERATION AT
PARTICULAR LOCATION/INDEX)

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

// Function to find minimum of three numbers
int min(int x, int y, int z) {
   if (x <= y && x <= z) return x;
   if (y <= x && y <= z) return y;
   return z;
}

// Function to compute the edit distance between two strings
int editDistance(char *initStr, char *finalStr, int initLen, int finalLen)
{
   // Base cases
   if (initLen == 0) return finalLen;
   if (finalLen == 0) return initLen;

   // If last characters are same, ignore last characters and recur for
remaining strings
   if (initStr[initLen - 1] == finalStr[finalLen - 1])
       return editDistance(initStr, finalStr, initLen - 1, finalLen - 1);

   // If last characters are not same, consider all three operations on
last character of initial string,
   // recursively compute minimum cost for all three operations and take
minimum of three values
   int insert_cost = editDistance(initStr, finalStr, initLen, finalLen -
1);      // Insert
   int remove_cost = editDistance(initStr, finalStr, initLen - 1,
finalLen);      // Remove
```

```
    int replace_cost = editDistance(initStr, finalStr, initLen - 1,
finalLen - 1);  // Replace
    if (insert_cost <= remove_cost && insert_cost <= replace_cost){
        printf("insert %c at %d\n", finalStr[finalLen-1], finalLen-1);
    }
    else if (remove_cost <= insert_cost && remove_cost <= replace_cost){
        printf("remove %c at index %d\n", initStr[initLen-1], initLen-1);
    }
    else {
        printf("Replace %c with %c\n", initStr[initLen-1],
finalStr[initLen-1]);
    }
    return 1+min(insert_cost, remove_cost, replace_cost);
}

int main() {
    char initStr[] = "abc";
    char finalStr[] = "a";
    int initLen = strlen(initStr);
    int finalLen = strlen(finalStr);

    printf("Init String : %s\n", initStr);
    printf("Final string : %s\n", finalStr);
    printf("Minimum operations required: %d\n", editDistance(initStr,
finalStr, initLen, finalLen));
    return 0;
}
```

- Output Screen-shots / Tracing :



```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_09$ cd "/h
04.c -o pr04 && "/home/hr/Documents/Semester_10/Lab_DAA
Init String : abc
Final string : a
remove b at index 1
remove c at index 2
Minimum operations required: 2
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_09$
```