

# IIT Roorkee

## CSN 261:Data Structures

### Laboratory

Assingment 2

Name:Ritesh Singh

Enrollment Number:18114067

Email: [rsingh1@cs.iitr.ac.in](mailto:rsingh1@cs.iitr.ac.in)

## Problem 1-

In this Problem, you have to implement a simple transposition cipher, where this cipher encrypts and decrypts a sequence of characters by dividing the sequence into blocks of size  $n$ , where  $n$  is specified by the encryption key. If the input text has a length that is not a multiple of  $n$ , the last block is padded with null characters (`'\0'`). In addition to  $n$ , the key also specifies two parameters  $a$  and  $b$ . For each block, the  $i$ -th output character, starting from 0 as usual, is set to the  $j$ -th input character, where  $j = (ai + b) \bmod n$ . For appropriate choices of  $a$  and  $b$ , this will reorder the characters in the block in a way that can be reversed by choosing a corresponding decryption key  $(n, a', b')$ .

### Task to Perform

Write a program `transpose.c` that takes  $n$ ,  $a$ ,  $b$ , `inputfile.txt` in `argv[1]`, `argv[2]`, `argv[3]`, and `argv[4]`, respectively, applies the above encryption; and writes the result to `outputfile.txt`.

Further, write a program `inverseTranspose.c` that decrypt the `outputfile.txt` and result in a new file named `decryptedOutputfile.txt`. Finally, write a program `compareFiles.c` to find the equivalence between the `inputfile.txt` and `decryptedOutputfile.txt` files.

You may assume that  $n$ ,  $a$ , and  $b$  are all small enough to fit into variables of type `int`. Your program should exit with a nonzero exit code if  $n$  is not at least 1 or if it is not given exactly

four arguments, but you do not need to do anything to test for badly-formatted arguments. You should not make any other assumptions about the values of  $n$ ,  $a$ , or  $b$ ; for example, either of  $a$  or  $b$  could be zero or negative.

## Program Files:

- 1.transpose.c
- 2.inverseTranspose.c
- 3.compareFiles.c

## Algorithm Used:

In transpose.c ,I have used FILE function of C to read and write data.Also I have used  $j=(a*i-b)\%n$  to encrypt the file.

In the inverseTranspose.c,I have used algos similar as above.In this the file get decrypted for a key provided by user.

In compareFiles.c I have used strcmp to compare the two files

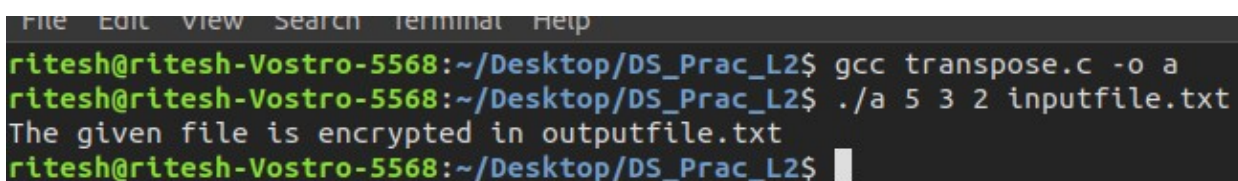
## Data-Structures Used:

- 1.Character Arrays

## Snapshots:

### Starting Programs:

transpose.c-



```
File Edit View Search Terminal Help
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ gcc transpose.c -o a
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ ./a 5 3 2 inputfile.txt
The given file is encrypted in outputfile.txt
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$
```

inverseTranspose.c

```

ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ gcc inverseTranspose.c -o b
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ ./b 5 2 1 outputfile.txt
The given file is decrypted in decryptedOutputfile.txt
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ █

```

## compareFiles.c

```

The given file is decrypted in decryptedOutputfile.txt
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ gcc compareFiles.c -o c
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ ./c
The input and output files are EQUAL
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ █

```

## Input ,Output and Decrypted Files:

```

Hello, world!
CSN-261

```

```

|Hleow,o r!l$d$
NC-S2$6$1$!l$d$

```

```

Hello, world!
CSN-261

```

## Time:

```

ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ time ./a 5 3 2 inputfile.txt
The given file is encrypted in outputfile.txt

real    0m0.003s
user    0m0.003s
sys     0m0.000s
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ time ./b 5 2 1 outputfile.txt
The given file is decrypted in decryptedOutputfile.txt

real    0m0.003s
user    0m0.001s
sys     0m0.003s
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ time ./c
The input and output files are EQUAL

real    0m0.003s
user    0m0.001s
sys     0m0.003s
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ █

```

## Problem 2-

A region can be represented either by its interior or by its boundary. Here we represent the region by its interior using one of the most common methods called image array. In this case we have a collection of pixels. Since the number of elements in the array can be quite large, the main objective is to reduce its size by aggregating equal-valued pixels.

A general approach is to treat the region as a quadtree, where the region is represented as a union of maximal non-overlapping square blocks whose sides are in power of 2. The quadtree can be generated by successive subdivision of the image array into four equal sized quadrants. If the sub-array does not consist entirely of 1s or entirely of 0s, it is then further subdivided into quadrants and sub-quadrants, etc.

Example:

The above figures represent a region and its corresponding quadtree representation,

(a) Defined as the Sample region having all the bit value to 1.

(b) Defined as the binary array of size  $8 \times 8$  which having bit value other than sample region within it is 0.

(c) Represent the conversion of binary array into the blocks where each block is formed as a union of maximal square having the same bit value. This kind of array is called a maximal square array.

(d) Quadtree representation, where the root node corresponds to the entire array. Each son of a node represents a quadrant of the region represented by that node. The leaf nodes of the tree correspond to those blocks for which no further subdivision is necessary. A leaf node is said to be black or white, depending on whether its corresponding block is entirely inside or entirely outside of the represented region. All non-leaf nodes are said to be gray.

Task to perform:

Write a C program, MAT.c to represent any region (in image array representation), into its quadtree form.

Input:

Sample region is represented as  $n \times n$  array (as shown in Fig. 1 using  $6 \times 6$  matrix).

The format of the input file should be as follows:

the pixel values in the input file are separated by a single space and rows are separated

by a newline character (refer to the sample L2\_P2\_inputsample.txt file shared in

Piazza).

(Note: The  $6 \times 6$  region array should be mapped at the bottom-left corner of a  $8 \times 8$  binary array as shown in Fig. 2(b))

Output:

1. Print the Maximal square array where it should be filled in the following order:

top-right, top-left, bottom-right and bottom-left quadrant, this should be done

recursively for all the sub-quadrants. All the cells within a maximal square block

should be filled with its corresponding block number.

2. Print the quadtree in the following manner, labels of leaf nodes, corresponding bit value and their level information (assuming the level of the root node to be 0), while traversing the quadtree in postorder. For example, in Fig. 2(d) the leaf node 3 having bit value 0 at level 2 and should be printed as (3,0,2).



## Algorithm Used:

1. I have used recursion to build the Quad tree. The break point of the recursion is that when all the elements of the array will be equal then the recursion will stop.
2. For maximal array, I have divided the array into four arrays recursively until I get the  $2 \times 2$  array. Now if all elements are same then I have stored the same value of bit in the array else I have stored different value of bits.
3. I have used a power method to calculate the power indices of 2

## Data Structure Used:

1. Quad-Tree Node which contains one 2d array and 4 pointers
2. POS Node which contains the data of height of tree and the value contained in it.
3. Arrays

## Snapshots:

### Starting the Program:

```
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ gcc Mat.c -o abc
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ ./abc
Input the size of array in input file
```

Output:

```
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ gcc Mat.c -o abc
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$ ./abc
Input the size of array in input file
6

01 01 01 01 02 02 03 03
01 01 01 01 02 02 03 03
01 01 01 01 04 04 05 05
01 01 01 01 04 04 05 05
06 06 07 09 13 13 14 14
06 06 08 10 13 13 14 14
11 11 12 12 15 17 19 19
11 11 12 12 16 18 19 19

(1,0,1)
(2,0,2)
(3,0,2)
(4,1,2)
(5,1,2)
(6,0,2)
(7,0,3)
(8,1,3)
(9,1,3)
(10,1,3)
(11,0,2)
(12,1,2)
(13,1,2)
(14,1,2)
(15,1,3)
(16,1,3)
(17,1,3)
(18,0,3)
(19,0,2)
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$
```

Time:

```
real    0m1.426s
user    0m0.003s
sys     0m0.000s
ritesh@ritesh-Vostro-5568:~/Desktop/DS_Prac_L2$
```