# CSN 261

# DATA STRUCTURES LAB

# ASSIGNMENT 2

DIVYANSHU SETIA

18114020

B. TECH, CSE

# 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) mod 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').

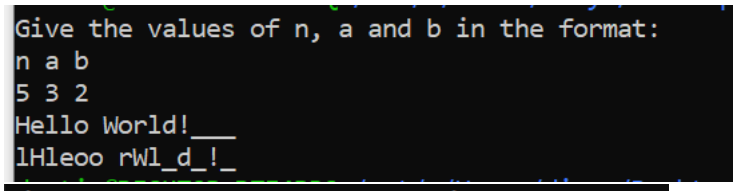
## Task to Perform:

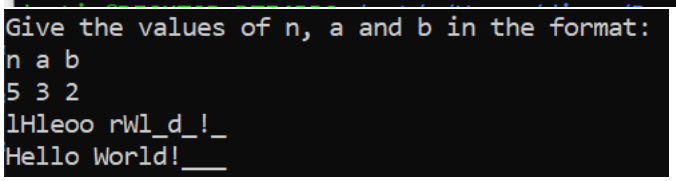
Write a program transpose.c that takes n, a, b, inputfile.txt in argv[1], argv[2], argv[3], and argv[4], respectively, applies the above encryption; and writes the result to outputfile.txt. Further, write a program inverseTranspose.c that decrypt the outputfile.txt and result in a new file named decryptedOutputfile.txt. 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.

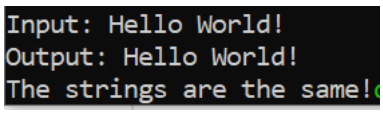
# INPUT:

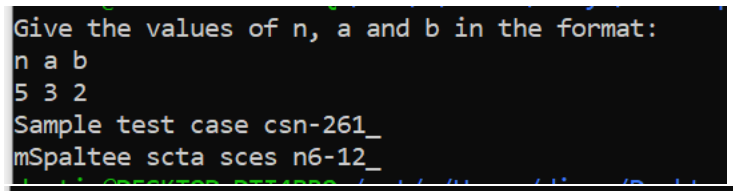
```
FILE* inputFile = fopen("inputfile.txt", "r");
fgets(fileData, 1000, inputFile);
strtok(fileData, "\n\n");

if((a == 0)||(gcd(abs(a),n) != 1)){
        printf("ERROR: Encryption is not possible.\nChoose appropriate a and b\n");
        return 0;
}

opData = encrypt(n, a, b, fileData);

printf("%s\n", fileData);
printf("%s\n", opData);

fprintf(opFile,"%s",opData);
```

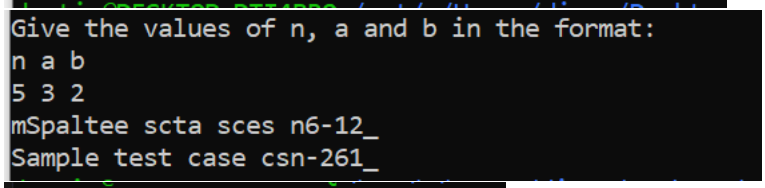
# OUTPUT:

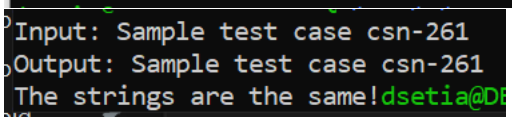



# ALGORITHMS AND DATA STRUCTURES:

- The txt file is read using fgets()
- To find a’ and b’ we use the properties (a*a’)%n = 1 and (a’*b + b’)%n = 0
- strcmp() is used to compare the strings

## TIME:

a)
```
real    0m1.839s
user    0m0.000s
sys     0m0.016s
```

b)
```
real    0m2.023s
user    0m0.000s
sys     0m0.016s
```

c)
```
real    0m0.017s
user    0m0.000s
sys     0m0.016s
```

---

# PROBLEM 2:

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 x n array (as shown in Fig. 1 using 6 x 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 6x6 region array should be mapped at the bottom-left corner of a 8x8 binary array as shown in Fig. 2(b))

## Output:

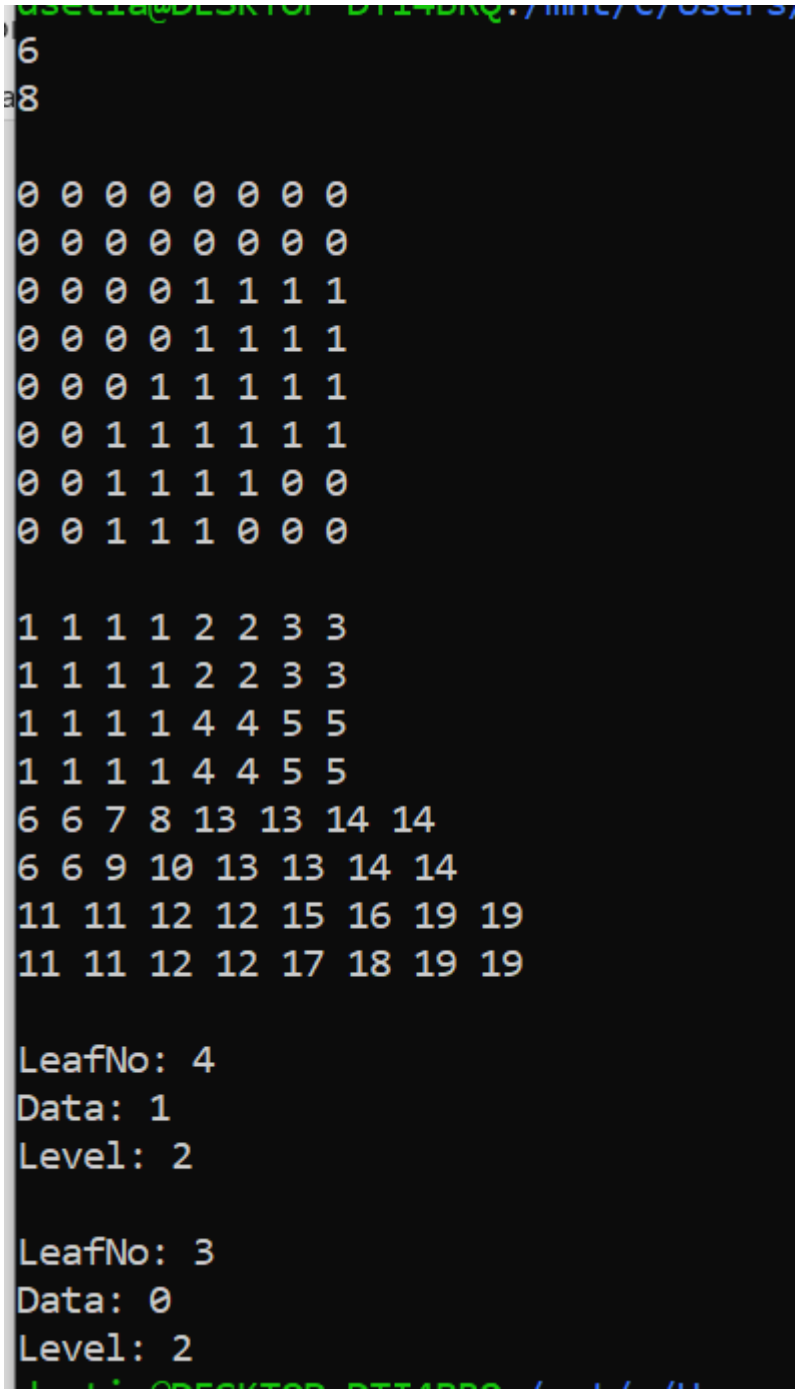
1. Print the Maximal square array where it should be filled in the following order: top-right, top-left, bottom-right and bottom-left quadrant, this should be done recursively for all the sub-quadrants. All the cells within a maximal square block should be filled with its corresponding block number. For example, with respect to Fig. 2(c) maximal array should be represented as 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).

INPUT:

```
printf("%d\n", cols);
int size = calculateSize(cols);
printf("%d\n", size);

printf("\n");

dataArray = (struct dataCell**) calloc(size, sizeof(struct dataCell*));
for(int i = 0; i < size; i++){
        dataArray[i] = (struct dataCell*) calloc(size, sizeof(struct dataCell));
}

insertArray(input, size, cols);
for(int i = 0; i < size; i++){
        for(int j = 0; j <size; j++){
                printf("%d ", dataArray[i][j].data);
        }
        printf("\n");
}

printf("\n");

makeTree(0, size, 0, size, 0);
```

```
for(int i = 0; i < size; i++){
        for(int j = 0; j <size; j++){
                printf("%d ", dataArray[i][j].leafNo);
        }
        printf("\n");
}

printf("\n");

printDetails(3,4);

printf("\n");

leafDetails(3, size);
```

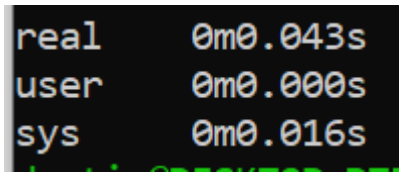
OUTPUT:


6
8
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 1 1 1 1 1
0 0 1 1 1 1 1 1
0 0 1 1 1 1 0 0
0 0 1 1 1 0 0 0
1 1 1 1 2 2 3 3
1 1 1 1 2 2 3 3
1 1 1 1 4 4 5 5
1 1 1 1 4 4 5 5
6 6 7 8 13 13 14 14
6 6 9 10 13 13 14 14
11 11 12 12 15 16 19 19
11 11 12 12 17 18 19 19
LeafNo: 4
Data: 1
Level: 2
LeafNo: 3
Data: 0
Level: 2

# ALGORITHMS AND DATA STRUCTURES:

- A struct is used to store all the required data of a node
- A recursive function is used to go through the 2-D array and form the quadtee

# TIME:




---