CSN-261:

Data Structures Laboratory

Assignment-2

Name-Kavya Barnwal

Class- B.Tech

CSESub Batch- O2

Enrollment No- 18114039

Email Id-kbarnwal@cs.iitr.ac.in

Problem Statement 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 ′).

For example, if n = 5, a = 3, and b = 2, the string Hello, world! would be encrypted like this:

in: H e l l o , w o r l d ! \0 \0

i: 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4

j: 2 0 3 1 4 2 0 3 1 4 2 0 3 1 4

out: l H l e o w , o r ! l \0 d \o

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.

Algorithms and Data Structures used in the implementation

Data Structures Used-

1-d char array to store text from file

Algorithm Used -

---transpose.c is used to encrypt the file provided its key(n,a,b) and path from where file is to read.

---inverseTranspose.c is used to decrypt the file provided its key(n,a,b) path from where file is to read.

---compareFile.c to compare two files(here used to compare inputfile and decryptedOutput).

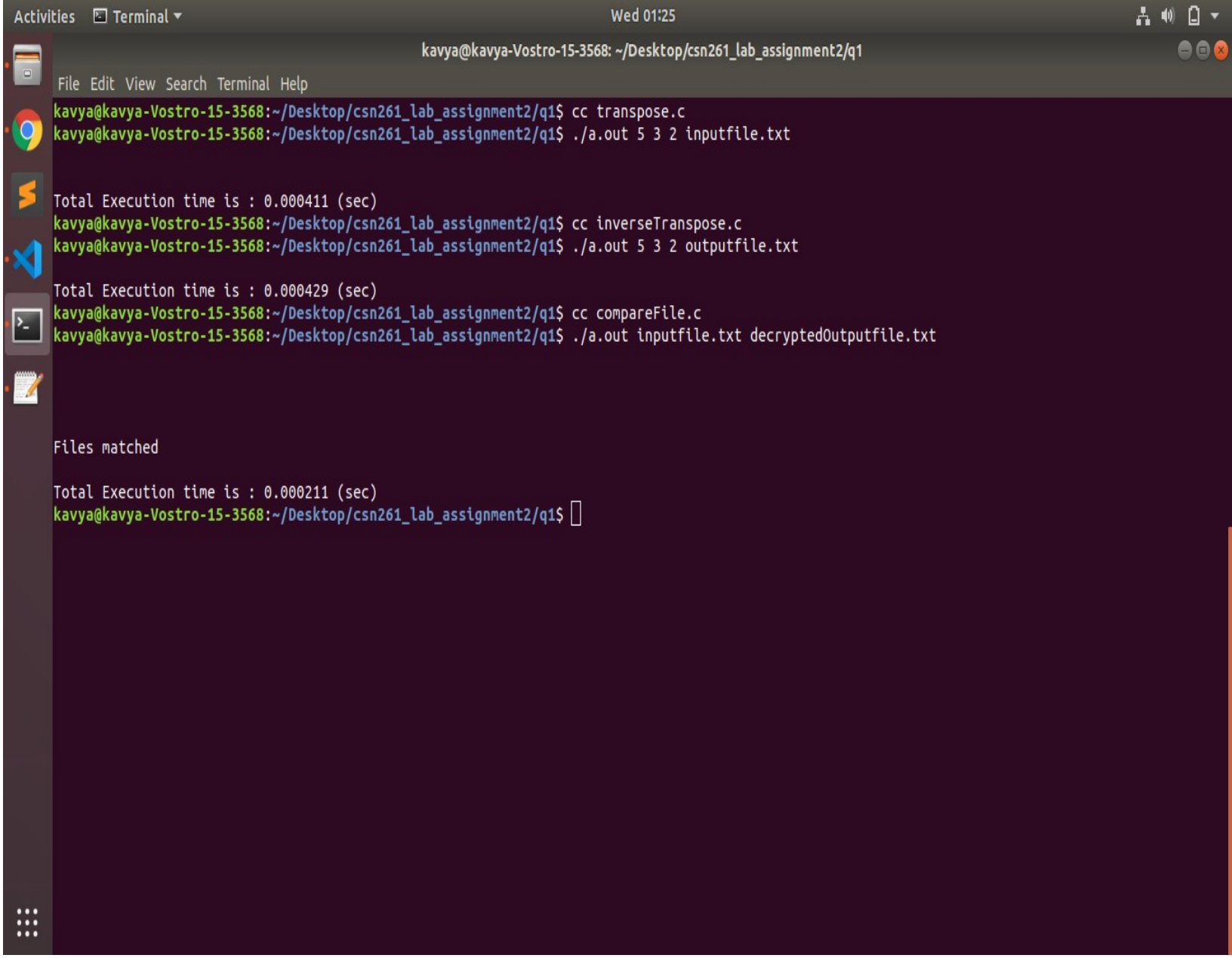
---readFile() method used to read File from a given path and return it as a string

---generateFile() method is used to generate a new File to a given path by passing a string.

---modularInverese() method is used to get inverse-modulo of a w.r.t b using number thoery for computing keys for the decryption.

---inverseTranspose() method will be used to generate decrypted file with name "decryptedOutputfile.txt" by decrypting the file whose path is provided.

---transpose() method will be used to read File from a given path and return it as a string .


Activities
Terminal
Wed 01:25
kavya@kavya-Vostro-15-3568: ~/Desktop/csn261_lab_assignment2/q1
File Edit View Search Terminal Help
kavya@kavya-Vostro-15-3568:~/Desktop/csn261_lab_assignment2/q1$ cc transpose.c
kavya@kavya-Vostro-15-3568:~/Desktop/csn261_lab_assignment2/q1$ ./a.out 5 3 2 inputfile.txt
Total Execution time is : 0.000411 (sec)
kavya@kavya-Vostro-15-3568:~/Desktop/csn261_lab_assignment2/q1$ cc inverseTranspose.c
kavya@kavya-Vostro-15-3568:~/Desktop/csn261_lab_assignment2/q1$ ./a.out 5 3 2 outputfile.txt
Total Execution time is : 0.000429 (sec)
kavya@kavya-Vostro-15-3568:~/Desktop/csn261_lab_assignment2/q1$ cc compareFile.c
kavya@kavya-Vostro-15-3568:~/Desktop/csn261_lab_assignment2/q1$ ./a.out inputfile.txt decryptedOutputfile.txt
Files matched
Total Execution time is : 0.000211 (sec)
kavya@kavya-Vostro-15-3568:~/Desktop/csn261_lab_assignment2/q1$

Problem2: Medial axis transformation (MAT)

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. 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 Fig 1: Image array representation of a region 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.

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 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-d array with indices.

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).

Algorithms and Data Structures used in the implementation

Data Structures Used-

2-d Arrays to store given Image-Array and Maximal

square Array.

Node for the implementation of required quad-tree.

Algorithm Used -

---check() method will be used to check whether the quadrant with size rounds,of image-array which starts from beginRow and beginCol.

---nearest2Exponent method will be used to get the just next coming exponent of 2.

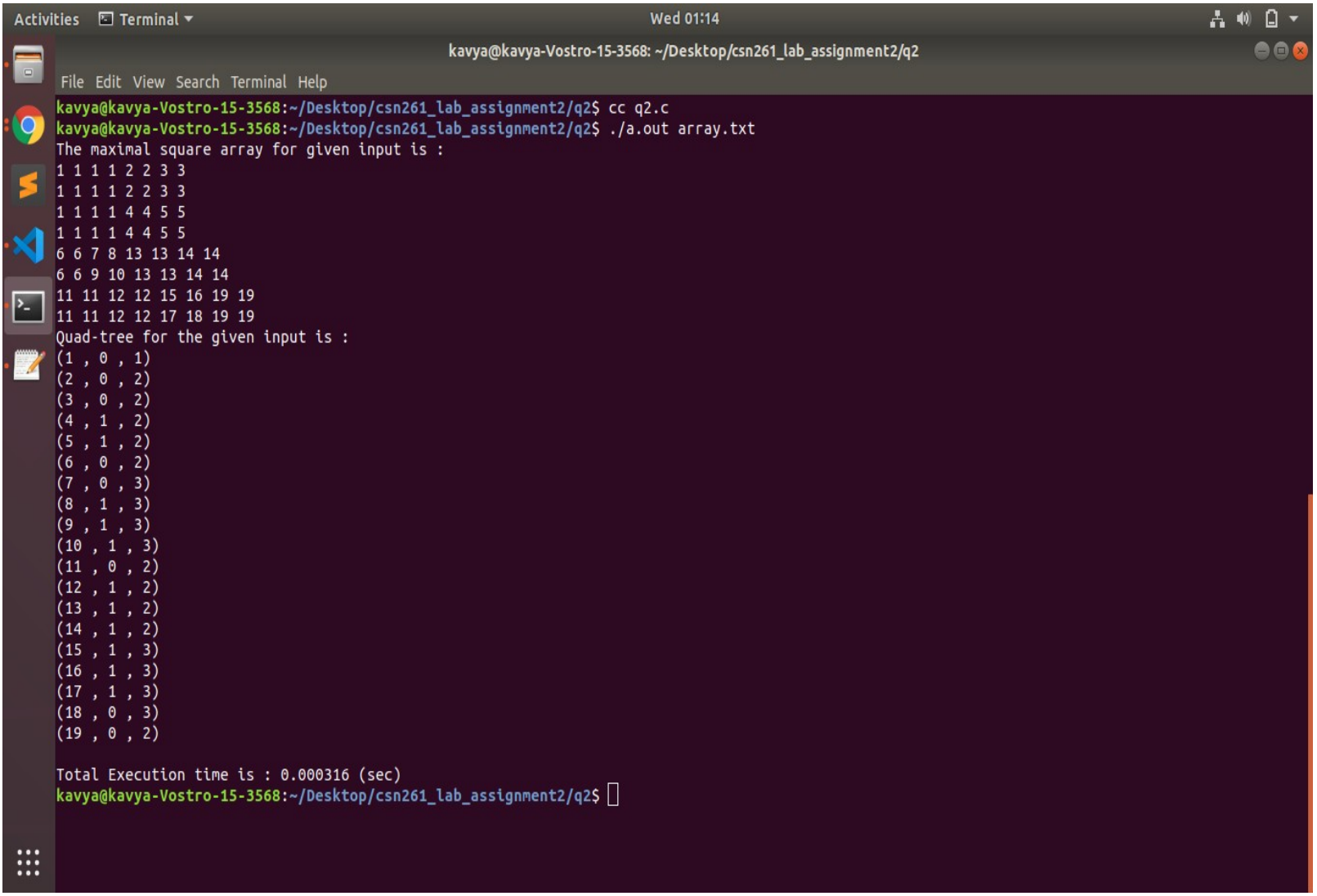
---partition() method will be used to partition the Image-Array into its further quadrants.

---addNode() method will be used to add the node to the quad-tree.

---fillMaximalSquareArray() method will be used to fill Maximal Square Array.

---printQuadTree() method will be used to print the quad-tree.

---printMaximalSquareArray() method will be used to print the the maximal square array.



Total Execution Time is around 0.000316 seconds