Indian Institute of Technology Roorkee

Department of Computer Science and Engineering

CSN-261: Data Structures Laboratory (Autumn 2019-2020)

Lab Assignment-2 (L2) **Date: July 24, 2019** **Duration: 2 Weeks**

General Instructions:

1. Every Lab Assignment will be performed by the students individually. No group formation is required and the evaluations will be done every week for the students individually.
2. Use of Linux OS is mandatory for this Lab to complete all the Lab Assignments.
3. The total execution time or CPU time is to be reported for all the programs in each Lab Assignment.
4. The student should use Deoxygen tool (http://www.doxygen.nl) for getting automatic documentation of the codes written by him/her.
5. For version control of the written codes, the students are being instructed to learn and use any open-source CSV tool (https://www.nongnu.org/cvs) or GitHub (https://github.com).

Submission and Evaluation Instructions:

1. **Submit your** zipped folder (**<filename>.zip** or **<filename>.tar.gz**) through your account in Moodle through the submission link for this Lab Assignment in Moodle course site: https://moodle.iitr.ac.in/course/view.php?id=46
2. **Hard deadline for Final submission in Moodle: Aug 07, 2019 (1:00 pm Indian Time).** For any submission after Final Deadline, 20% marks will be deducted (irrespective of it is delayed by a few seconds or a few days). The key to success is starting early. You can always take a break, if you finish early.
3. The submitted zipped folder (**<filename>.zip** or **<filename>.tar.gz**) must contain the following:
   (a) The source code files in a folder
   (b) A report file (**<filename>.DOC** or **<filename>.PDF**) should contain the details like:
      i. Title page with details of the student
      ii. Problem statements
      iii. Algorithms and data structures used in the implementation
      iv. Snapshots of running the codes for each problem
4. The submission by each student will be checked with others' submission to identify any copy case (using such detection software). If we detect that the code submitted by a student is a copy (partially or fully) of other's code, then the total marks obtained by one student will be divided by the total number of students sharing the same code.

Instructions for L2:

1. Objective of this Lab Assignment L2 is to make the students familiar with different data structures while coding the programs in the C language to solve some real-life problems.
2. The students are expected to have a basic knowledge of data structures and the C programming language.
3. It is mandatory to learn and use gdb (GNU Debugger) for debugging the programs in Linux platform after installing gdb (https://www.gnu.org/software/gdb). There is also an interesting online GDB tool to learn: https://www.onlinegdb.com
4. The student will have to demonstrate and explain the coding done for this Lab Assignment L1 in the next laboratory class to be held on **Aug 07, 2019** for evaluation.

Problem1.
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').

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.

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	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.

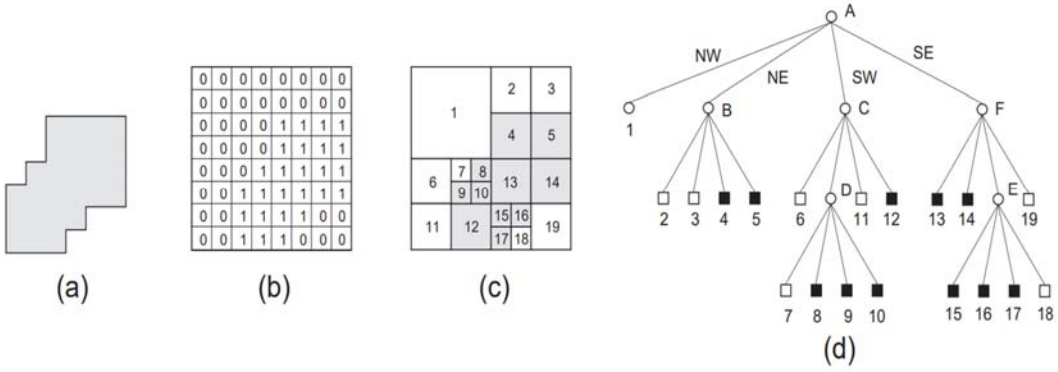

Fig 2: Flow of quadtree representation from the sample region

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*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 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

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

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).