The exercises below should be implemented using openMp. programs must be named the way it is mentioned in the exercise. All programs must be able to read command line parameters. All inputs should be taken from files. Output of the programs must also be written to files. Console can be used for printing timing or debugging information.

For all these problems, you must ensure that the sequential version and the parallel version produce the same output.

1. randMatrices

Create matrices of size *r* rows and *c* columns, where both are read as parameters. Take another integer *k* as input. Fill up a *r* x *c* matrix such that the entries in the matrix are integers which are uniformly randomly chosen from 1 to *k*.

It is necessary that the entries are random, there should be no discernible pattern based on the position, value and so on. There are several tests to check if a matrix is sufficiently random. Perform the simplest: take a histogram of the values in each row, each column and the whole matrix. The histograms must be almost same for each of 1 to *k*.

(The histogram plots are not necessary. Just check if the histograms are roughly equal.)

r, c can each be upto 10,000 and the matrix should not be necessarily square. $2 <= k <= \min(r,c)/10$.

Input: r, c, k, <output filename>

Output: <filename> must contain the matrix

2. shuffle

Given a r x c integer matrix, a shuffle is defined as a random permutation of rows or columns. You must take a sequence of commands R i j (or C i j) which stands for shuffle of row (column) i with row (or column) j. The number of shuffles is limited to 10,000.

Input: r, c, <file containing input matrix>, <file with R or C commands, one command per line>, <output file name>

Output: <output file name> must contain the shuffled matrix

3. **threshold**

You will be given a r x c matrix of integers called M which can be treated as an image. Given M, you should construct a binary image B such that Bij = 1 if no more than *p* percentage of pixels in M are greater than Mij.

Input: r, c, <file name containing M>, p, <file name to store B>

Output: <B stored in the file>

The values in M are in the range of 0 to 300 both inclusive.

4. **gameOfLife**

Conway's game of life is a cellular automaton where the game depends on the initial configuration and nothing else. The initial configuration is two dimensional grid of cells each of which is either *dead* or *alive.* The game proceeds in steps where every cell interacts with the vertical, horizontal or diagonal neighbors and decides on its status in the next step. At each step, the following rules are used:

1. Any live cell with fewer than two live neighbours dies, as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The initial configuration is called the *seed* and starting from this all the cells take steps in tandem.

Input: <seed file>, r, c, #steps, <output file>

Output: <file containing the board after #steps starting from the seed>

Seed should have not more than 10% of cells that are alive.

Deliverables:

For all the programs above, run with r = c = 1,000 and r = c = 10,000.

For problem 2 use 1000 shuffles.

Use p = 10% for problem 3.

Be careful about the number of cells that are alive in the seed in Problem 4.

Tabulate the following:

Problem name, Lines of Code, sequential run time, parallel runtime, rough estimate of time it took to develop, peak memory used in the program during the parallel phase.

Write a brief report explaining your parallelization strategy.