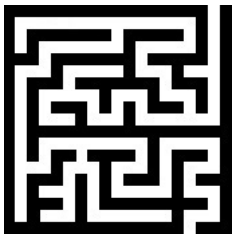
# CMPE 300 - Analysis of Algorithms
# Fall 2015
# Programming Project

Due: 10.01.2015 23:59

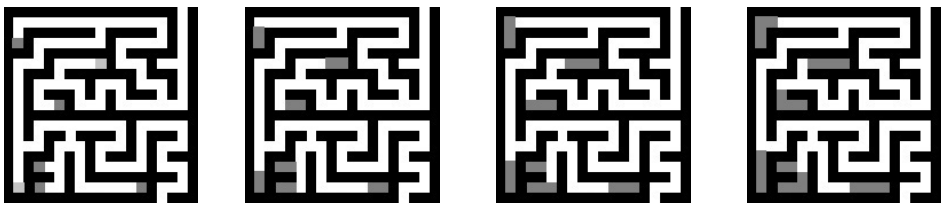
## Introduction

In this project, you are going to experience parallel programming with C/C++ using MPI library. You are going to implement a parallel algorithm for solving mazes.

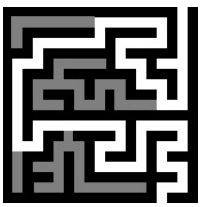
## Maze Solving



A maze is a two dimensional grid consisting of intersecting paths. For this project, we will consider square mazes which have unique solutions with two entrance points. Each cell in the grid is either a black cell representing a wall or a white cell. Suppose there were no walls. Except for the cells in the border, the number of possible moves from each cell would be equal to 4; namely, up, down, right and left. We call a cell dead-end if it is surrounded by 3 walls since once you reach that cell, you can not move to another cell. The algorithm we are going to implement is based on the observation that the dead-ends can not be a part of the solution. Therefore, our approach for solving the maze will be to fill the dead-ends iteratively until no more dead-ends are present. Since we assume that there is a unique solution, we will be left with the solution for the maze once the algorithm terminates. The algorithm is illustrated below.



First 4 steps of the algorithm



Several steps later

At each step, white cells (paths) which are surrounded by 3 walls become walls and the updated cells form the new grid for the next iteration. The grid can be interpreted as a two dimensional array where

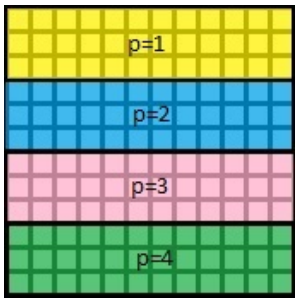
the black cells are represented by 0's and the white cells are represented by 1's. The pseudocode for the algorithm is given below.

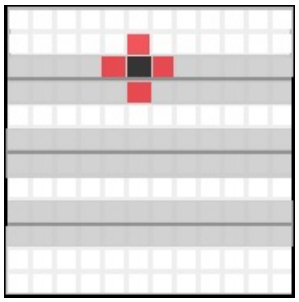
```
Function: Maze(A[n][n])
Input:  A (Maze representation)
Output: B (Solution of the maze)
    while deadEndsPresent do
        B ← A
        for i ← 0 to n − 1 do
            for j ← 0 to n − 1 do
                if A[i][j] = 1 and wallNeighbors(A, i, j) = 3 then
                    B[i][j] ← 0
                end if
            end for
        end for
        A ← B
    end while
end Maze
```

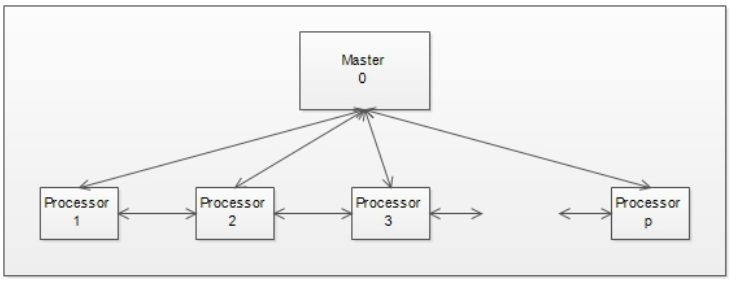
## Parallel Maze Solving

In this project, our aim is to implement the sequential maze solving algorithm using MPI. Two different parallel approaches will be discussed. We will assume that the maze is a square of size $n \times n$. There will be 1 master and p slave processors and we will assume that $n$ is divisible by the number of slave processors $p$.

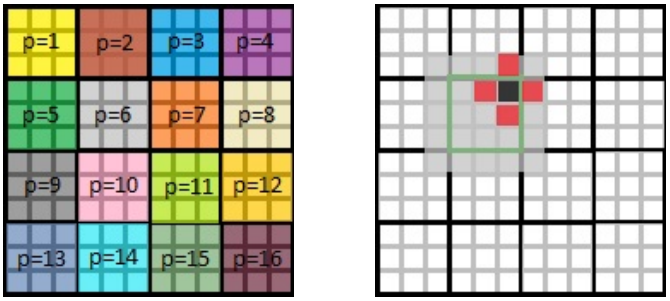
1. In the first approach, each processor is responsible from a group of $n/p$ adjacent rows. Each processor works on $(n/p \times n)$ cells and these cells are stored locally by the processor. When a cell is inspected, the processor should inspect all of its neighbors. When the cell is not on the boundary of two adjacent processors, information about the neighbor cells are present to the processor. When a cell on the boundary is inspected, the processor needs to obtain information from the adjacent processor. Consider the black cell in the below figure. Processor 1 should communicate with Processor 2 in order to learn the status of the south neighbor of the black cell. Therefore, each processor should communicate with the adjacent processor at every iteration. Information about those neighbor cells of the boundary can be stored locally and updated at every iteration.

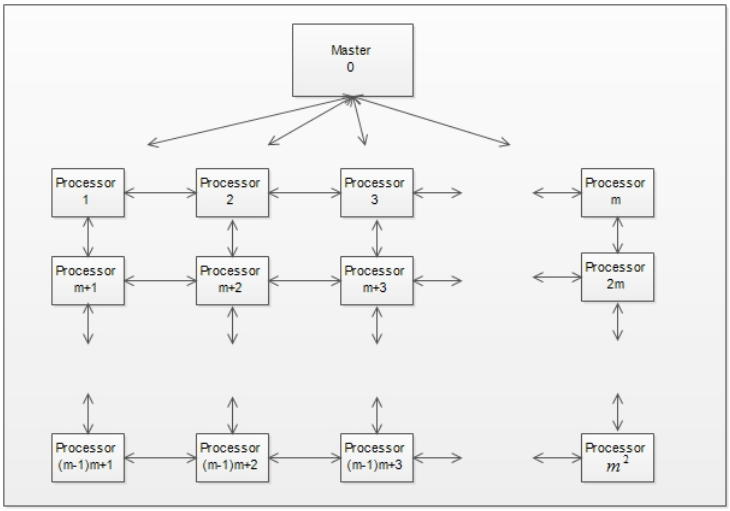







2. In the second approach, the grid is divided into $(n/p \times n/p)$ blocks and each process is responsible from a block. Now the updates are more trickier since each process has more than 1 adjacent process.




# Implementation

1. For the MPI environment and tutorials please visit the web page. You can find many other tutorials in the web.
2. You are expected to implement the first approach. There will be bonus points for those who also implement the second approach.
3. Start by distributing the input among the processesors and let each processor work on its cells without any communication. Once you accomplish, add the communication.
4. The program will read the input from a text file and print the result in another text file. Text file will contain the information about the maze where 0's represent the walls and 1's the open cells. The first line of the input file will be the size of the maze followed in the next line by the maze where 0's and 1's will be separated by spaces. You may assume that the input maze will have only one solution, two entrance points, and will contain no loops. An example input and output file are as follows:

Input file

```
8
0 0 0 0 0 1 0 0
0 1 0 0 0 1 0 0
0 1 1 1 0 1 0 0
0 1 0 1 0 1 1 0
0 1 0 1 1 1 0 0
0 1 0 0 0 0 0 0
0 1 1 1 1 1 0 0
0 1 0 0 0 0 0 0
```

Output file

```
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
0 1 1 1 0 1 0 0
0 1 0 1 0 1 0 0
0 1 0 1 1 1 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
```

5. The whole input shouldn't be read/stored by each process. Input should be read and distributed to the other processors by the master processor.

6. Any functioning of the program regarding the whole program such as printing the output should be done by the master processor.

7. When two processors are exchanging information about the boundary cells, be careful to avoid deadlock. Before moving on the next iteration, make sure that all processors have finished their jobs.

8. The names of the input and output files will be given on the command line, as well as the number of processes. An example execution for a Windows user that runs on 4 processors and uses input.txt and output.txt files would be:
mpiexec -$n$ 4 project.exe input.txt output.txt

9. Several input and output files to test your code will be provided in the course website.

## Submission

1. The deadline for submitting the project is January 10, 23:59. The deadline is strict. We will have two demo sessions the following two days.
2. This is an individual project. Your code should be original.
3. Your code should be sufficiently commented.
4. You should write a header comment in the source code file and list the following information.
   /*
   Student Name: Ali Veli
   Student Number: 2008123456
   Compile Status: Compiling/Not Compiling
   Program Status: Working/Not Working
   Notes: Anything you want to say about your code that will be helpful in the grading process.
   */
5. You must prepare a document about the project, which is an important part of the project. You should talk about the efficiency of the parallel approach when compared to the sequential one. Follow the guidelines given in the "Programming Project Documentation" link on `http://www.cmpe.boun.edu.tr/~gungort/informationstudents.htm`.
6. Submit your project and document as a compressed archive (zip, rar, ...) to the following link: `https://www.dropbox.com/request/vIYu0cBa58GO2XXtC3Qw`. Your archive file should be named in the following format: "name_surname.ext". You don't need to submit any hard copies.
7. If you have any further questions, send an e-mail to *ozlem.salehi@boun.edu.tr*.