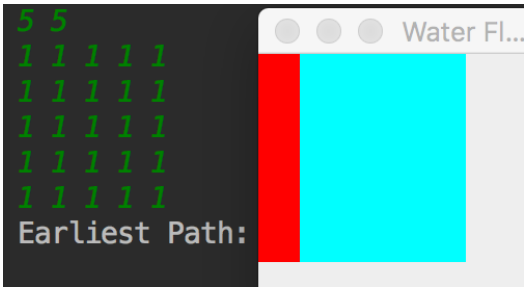
Analytical Questions

1. Minimum number of cells would be n cells when there's a straight line from top of the grid to the bottom of the grid without shifting to any columns left or right. Such a path would look like this:



In this example, Earliest flow path is comprised of five cells as it starts from the 1st row to the last one without moving left or right.

Maximum number of cells will be calculated based on rows and columns. To create maximum number of cells, we would like to have the water flow left to right and vice versa while going down only when it's on the first or last column.

Formula for this case is: $\left(\left\lfloor\frac{n-1}{2}\right\rfloor * m\right)+(n-\left\lfloor\frac{n-1}{2}\right\rfloor)$ where n = rows and m = columns
For example, if n & m = 5, there would be 13 maximum number of cells which is shown at the image above. $\left(\left\lfloor\frac{n-1}{2}\right\rfloor * m\right)$ = 10 and $(n-\left\lfloor\frac{n-1}{2}\right\rfloor)$ = 3. Total equals 13.

2. There can be a maximum number of 3 cell-disjoint flow paths between the two cells. If the starting cell and the final cell are in the same column, there are three possible paths. 1st path is to move to the left cell of the starting cell and then move down until row = n-1. Then move right to the final cell. 2nd path is to move down from starting cell until you reach the final cell. 3rd path is to move to the right by one and move down until row = n-1. Then move left to the final cell.

3. I have used four queues to determine the flow. "Current" queue processes the cells that were stored in queue 1. When "Current" queue is empty after a flow, it gets the nodes from queue 1. queue 1 gets the nodes from queue 2 and so on. It's better to use a linked list queue instead of array based data structures since we don't know how many elements are going to be stored in each queue. Adding and deleting nodes from linked list is also better

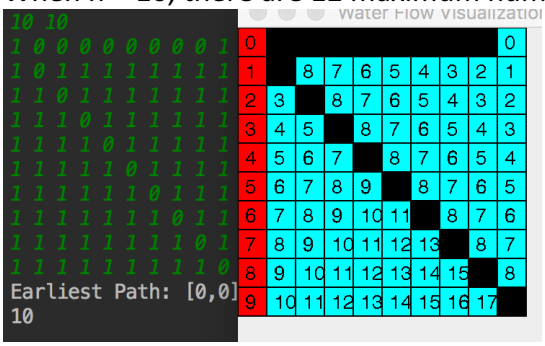
with handling memory space. Array based data structures could have unused memory space while determineFlow method is running.

4. In worst case scenario the program must go through all of the cells in delayTimeGrid and enque & dequeue them. When they are dequeued, at most you must check neighboring 3 cells as well. Just like formula mentioned in 1, the highest term will be nm and thus it would take O(nm).

5. Maximum number of cells my data structure could have is n when n <= 6. If n > 6, then maximum number of cells is n + $\left\lceil \frac{n-6}{2} \right\rceil$. Both cases have maximum number of cells when t = n – 2. This can be achieved when the grid has only two cells of 1 in the first row and the table is divided by half diagonally as two different flows will flow from two different ends at the same time.

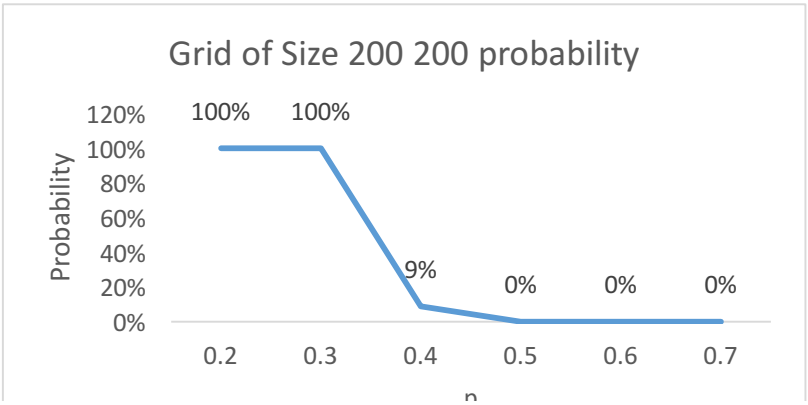
When n = 6, there are 6 maximum number of cells when t = 4.

When n = 10, there are 12 maximum number of cells when t = 8.



Experimental Analysis and Questions

1. When p is 0.2 & 0.3, all of the grids had earliest flow path. Probability of having a flow out of row 199 decreases dramatically as p increases from 0.3 to 0.4. Probability is 0 when p equals 0.5, 0.6, and 0.7.



2. Grids of size 50 by 200 had 100% probability when p was 0.4 or below. When p was 0.5 or 0.6, probability of success was at 0%. However, grids of size 200 by 50 had 100% probability only when p was 0.3 or below. Test cases with 0.4 or above had 0% success rate. Grids of size 50 by 200 seems to have more success as it has more columns which means there are more options left and right. It also has fewer rows and thus a shorter pass from top to bottom.