1. Amaz(e)ing meeting

I read in *simple_maze_20x30.txt* and constructed a int[][] *inputMaze* from it. Then I constructed Graph *G* from that 2x2 list. The vertices got value according to which cell they represented (e.g. *inputMaze* [0][4] was represented with vertex with value 4 and *inputMaze* [1][4] with 34). For each cell/tile with value 1 I checked its neighbor cells and added edges between them if they also had 1.

To solve with Breadth-First Search I used *BreadthFirstPaths.java* and searched for the middle vertex (with value 215). I inserted an if test to isolate the paths starting at vertex 2 and 599 (which are the enterances of the maze).

I also solved the maze based on *maze.java*. I did some modifications in the original file:

- Changed the solve method from DFS to BFS
- Added a path tracked to keep track of the depth of the search
- generated the maze based on connected edges in the Graph *G*
- Created a backtracking method to backtrack the coordinates of the shortest paths and then draw it

2. Preorder vs Postorder DFS

To print out the pre-order I added a print statement in the beginning of the *dfs* method. This printa the nodes when they're "arrived" at, before handling their edges.

To print the post-order I added the print statement in the end of the *dfs* method after visiting all the edges. This prints the nodes from last to first.

For the reverse post-order I did as above but instead of printing I added the nodes to a stack. Then when the DFS was done, I printed the stack in the main method.

The last one; reverse postorder of a DAG is a topological order (pf. s 582).