CMSC 433 Project 3: A-MAZE-ing Race

Due April 10th (Friday), 11:59:59pm

Goal: In this project you will create an efficient solver for two-dimensional mazes. Informally, each maze is a grid of positions which may have walls between them in each compass direction (north, south, east, west). For a given maze, your program should be able to either return a solution or find that there is no solution faster than a single-threaded solver. The implementations details for how to solve the maze are left up to you, and you may use any default Java library classes you need.

Contest: Unlike previous projects, this project will include a contest component in which your program will be run against programs submitted by other students in the class on a Contest Server and the results will be posted to a shared leaderboard. This board will detail every submission's running time on various undisclosed test mazes, and *extra credit will be awarded based on the performance of your program.* Submission to the Submit Server will automatically queue your program to run on the Contest Server's next cycle.

Further details regarding the Contest Server will be posted to **Piazza**, so be sure to check there.

Givens: The following will hold true of all the mazes we will use to test:

- Mazes will contain at most one solution
- Mazes will not contain cycles and/or loops
- Mazes will be no larger than 20000x20000

Additionally, when we run tests on the Contest Server we will overwrite all classes provided in the skeleton (with the exception of `Main.java` and of course `StudentMTMazeSolver.java`) with the default distributed versions. Therefore these classes should not be modified. Also, since we guarantee them not to change, you may want to examine and take advantage of some of them in your implementation.

Rules: Because the Contest Server will restore the skeleton classes to their defaults before running, you should keep your changes contained in `StudentMTMazeSolver.java`. However, you may introduce new classes as needed (such as to extend provided classes) and they will not be overwritten. The following are guidelines about the classes you may modify:

- `StudentMTMazeSolver.java`:
  This class is where we will check for your implemented solution. You may change which class it extends.
  - `StudentMTMazeSolver(maze)`
    The constructor takes a `Maze` object which is the target to solve and stores it for later reference.
  - `solve()`
    This is where your main computation should be. A `List<Direction>` is expected as a return if a solution exists, which consists of every `Direction` taken at every position leading from the start to (but not including) the endpoint. If no solution exists, this method should return `null`.
- `Main.java`:
  This class was provided to facilitate testing your code and can be modified as desired. It will not be run by the Submit Server, and any changes made to it will be overwritten on the Contest Server.
  - `read(filename)`
    This method reads in `Maze` object a from a file and stores it to a variable.
  - `solve()`
    Function which executes the provided solver(s) and prints out their results.

- `initDisplay()`
  Displays a graphical representation of a maze for debugging purposes. Your maze solver can also be configured to draw its path on the display to make tracing easier.
- `main()`
  Runs through a test routine. Opens a `Maze` from a file, runs the given solvers, and displays the results alongside a picture of the `Maze` if requested.

Additionally, you **SHOULD NOT** store the maze files in the project directory when submitted since they increase the file size. Failure to follow these directions may jeopardize your program's runs.

Additional classes: For this project you are only required to modify `StudentMTMazeSolver.java`. However, it may be beneficial to use some of the pre-provided classes. A brief description of some of the classes follows below. Refer to method comments in each class for further notes on what they do.

- `Maze.java`:
  Represents a two-dimensional maze as an `AtomicIntegerArray` and provides methods returning information about the maze. The method `checkSolution()` can be used to verify results. For this project all mazes are read and deserialized from files.
- `Position.java`:
  Represents a point in the `Maze`.
- `Direction.java`:
  Represents the four compass directions, used to refer to neighboring `Positions`.
- `Move.java`:
  Represents a move by storing a `Position` along with the `Direction` of the move. Also has a reference to the previous `Move` for traversal-building.
- `Choice.java`:
  Represents movement possibilities for a `Position` in a `Maze` as part of a traversal. In addition to `Position`, it stores the `Direction` of the previous `Position` as well as valid `Directions` to proceed in. A wall at a given `Position` is represented by a lack of `Choice` in a given `Direction`.
- `MazeSolver.java`:
  Superclass of all maze solvers. Whatever your implementation is, it MUST subclass this class.
- `SkippingMazeSolver.java`:
  `MazeSolver` that for each move, follows a `Direction` and progresses through the `Maze` until a `Choice` is reached, allowing for a sparser representation of a traversal. This partial `Direction` list can be converted to a full `Direction` list upon returning. A method to color `Positions` may be used in debugging.

In addition, we have included several fully-implemented single-threaded solvers, `STMazeSolverBFS`, `STMazeSolverDFS`, and `STMazeSolverRec`, for you to compare times against and/or use as a basis for your solution.

Testing: Among the tests we will use for this project are races where we will compare your solver's time against a single-threaded solver's time. For full credit, your program should be able to consistently win these races. We will, of course, also test your output for correctness. For offline testing, we have provided several maze files for you to run, as well as some single-threaded solvers you can use as benchmarks. Because of the provided files and the presence of the Contest Server, we will **NOT** have public tests on the Submit Server. Discussion for this project is **encouraged**.

Submission: Submit a .zip file containing your project files to the Submit Server.