CircuitTracer

CS 221 Programming Assignment

*“If you find a path with no obstacles, it probably doesn’t lead anywhere.”*

– Frank A. Clark

# Objectives

* Implement a brute-force search algorithm.
* Use command-line arguments.
* Manage Exceptions while parsing an input file.
* Analyze the impact of storage data structure choice on the search algorithm.
* Create a graphical user interface (optional).

# Background

You are an intern at a small, custom circuit-board design firm that is still firmly rooted in the paper-and-pencil world--all layouts are done by hand. Your boss finally buckled under the pressure of progress and bought an actual computer to enable some layout automation, but he has sworn that's all the money he's willing to spend on this newfangled technamology.

That's where you come in. Despite there being commercial software for this purpose, the boss thinks it will be cheaper to have you, the unpaid intern, write a program to find the shortest path for connecting a pair of components on a crowded circuit board.

Head engineer, Jim-Roy Huffnanny, has already come up with an [input file format](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/valid1.dat) to represent the layout of a circuit board. You'll need to be able to read in circuit layouts in this format and output your results in a similar [output format](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/valid1.out):

* First line of an input file gives two integer values for the number of rows and columns on the board.
* Each remaining line represents one row of the board. Various characters represent components, occupied spaces, and open spaces as follows:

'O' - open space (that's capital letter O, not zero)

'X' - an occupied position (a component or existing trace the new trace cannot cross)

'1' - first of two components that need to be connected

'2' - second of two components that need to be connected

'T' - a position in the new trace connecting the two components

A recent board that's giving Jim-Roy fits looks like this:

5 6

X O 1 O O O

X X X O O O

O O O O X O

O 2 O O X O

O X O O O O

The goal is to find a shortest path that connects the two components identified by digit characters. Diagonal traces are not allowed on account of issues with the manufacturing division--the division head, an old guy named Pak Mon, believes diagonal motion is unnatural.

Looking at the sample file, you quickly identified three equally satisfactory solutions for this board, each with a path length of 5:

X O 1 T O O

X X X T O O

O T T T X O

O 2 O O X O

O X O O O O

X O 1 T O O

X X X T O O

O O T T X O

O 2 T O X O

O X O O O O

X O 1 T O O

X X X T O O

O O O T X O

O 2 T T X O

O X O O O O

There are also many unsatisfactory solutions, such as Jim-Roy's latest effort:

X O 1 T T T

X X X O O T

O T T T X T

O 2 O T X T

O X O T T T

You recall a general-purpose, brute-force search algorithm from back in your CS 221 days that should work for this application, but you're not certain whether a stack or a queue will be the best underlying data storage. You decide to wrap both in a class called Storage and choose which one to use at runtime.

Here is the pseudo-code for the search algorithm:

initialize an empty *Storage* object called stateStore that stores objects of type *TraceState*

initialize an empty *List* called bestPaths that stores objects of type *TraceState*

add a new initial *TraceState* object (a path with one trace) to stateStore for each open position adjacent to the starting component

while (!stateStore.isEmpty)

retrieve the next *TraceState* object from stateStore

if that *TraceState* object is a solution (ends with a position adjacent to the ending component),

if bestPaths is empty or the *TraceState* object's path is equal in length to one of the *TraceState*s in *bestPaths*,

add it to *bestPaths*

else if that *TraceState* object's path is shorter than the paths in *bestPaths*,

clear *bestPaths* and add the current *TraceState* as the new shortest path

else generate all valid next *TraceState* objects from the current *TraceState* and add them to stateStore

Note:

* Initial TraceStates are constructed with a single starting point. Subsequent TraceStates build on an existing TraceState by adding one new position adjacent to the old path's last position.
* You can't extend the path into a position that isn't open or that is already part of the path.

This algorithm will search for all possible paths (search states) from the starting component to the ending component, but the choice of underlying data structure for Storage will affect the order in which paths are evaluated in the while loop and the number of states in stateStore during the search.

# Tasks

1. **Before** you starting coding your project:
   1. Use this simple layout to work *completely* through the search algorithm by hand twice -- once with stack storage, and once with queue storage  
        
      1 O O O  
      O O O 2
   2. Create a section in your README file called *Analysis* and thoroughly address the following questions:
      1. How does the choice of Storage configuration (stack vs queue) affect the sequence in which paths are explored in the search algorithm? (This requires more than a "stacks are LIFOs and queues are FIFOs" answer.)
      2. Is the total number of search states (possible paths) affected by the choice of stack or queue?
      3. Is using one of the storage structures likely to find a solution in fewer steps than the other? Always?
      4. Does using either of the storage structures guarantee that the first solution found will be a shortest path?
      5. How is memory use (the maximum number of states in Storage at one time) affected by the choice of underlying structure?
      6. What is the Big-Oh runtime order for the search algorithm? Does it reflect the maximum size of Storage? Does it reflect the number of board positions? Does it reflect the number of paths explored? Does it reflect the maximum path length? Is it something else? What is 'n'? What is the primary input factor that increases the difficulty of the task?
2. To code your project:
   1. Complete the CircuitBoard constructor.
      1. The constructor should parse the input file and populate the char[][] board class variable.
      2. Any formatting problems with the input file should cause an InvalidFileFormatException to be thrown from the constructor, with an appropriate message about the cause of the exception.
   2. Complete the CircuitTracer class.
      1. The class should:
         1. validate the command-line arguments
         2. construct a CircuitBoard from the input file
         3. search for all optimal trace paths using the user's storage choice
         4. report the results according to the user's display choice
      2. The program requires three command-line arguments:
         1. First argument:  
            -s -- use a stack for storage  
            -q -- use a queue for storage
         2. Second argument:  
            -c -- run program in console mode  
            -g -- run program in GUI mode (extra-credit--see below for details). If GUI mode is not supported, let the user know and exit the program.
         3. Third argument: name of the input file
      3. Use conditional checks and exception handling to appropriately deal with input errors from the command line and the input file. These [example input files](https://github.com/BoiseState/CS221-Public/tree/master/assignments/CircuitTracer/files/boards) include a variety of valid and invalid input files to use for testing.
         1. Give a clear [usage message](https://en.wikipedia.org/wiki/Usage_message) in response to invalid command line arguments and exit cleanly.
         2. Report InvalidFileFormatExceptions for any issues encountered while parsing the input file, then exit cleanly. Do not display full stack traces - only the InvalidFileFormatException's name and descriptive message as obtained from its toString() method.
         3. No unhandled exceptions should crash the program and no stack traces should be printed to the console while the program is running.

# Files

**Do not start programming** until you have first done the algorithm walkthrough by hand and documented your observations as described above.

In order to complete this project, you will need the following files:

* [Storage.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/Storage.java) - a completed class that can be implemented using either a stack or a queue.
  + Contains an enumeration defining the available underlying data structures that the user can choose from when a Storage object is instantiated.
* [CircuitBoard.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/CircuitBoard.java) - a partially-completed class that represents a circuit board.
  + Contents of a circuit board loaded from file or part of a search state.
  + Complete code to read the input file to construct a new board.
* [TraceState.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/TraceState.java) - a completed class that represents a search state with the trace path.
* [InvalidFileFormatException.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/InvalidFileFormatException.java), and [OccupiedPositionException.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/OccupiedPositionException.java) - exceptions used by other classes in this project
* [CircuitTracer.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/CircuitTracer.java) - a partially-completed driver class.  
  You will need to complete code to:
  + parse and validate command line options
  + configure Storage stateStore for the search
  + run search for the best paths
  + report results from the search
* Test files:
  + [CircuitTracerTester.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/CircuitTracerTester.java) - a test class for the CircuitTracer and CircuitBoard classes. ([sample test output](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/sample_test_output.txt) when all is working)  
    NOTE: Eclipse will only run CircuitTracer from the tester if you have configured your project to "Use project folder as root for source and data files" as you were instructed to do back in [Lesson 02 of CS 121](https://docs.google.com/document/d/1Dk9Bkoe3-K8X3J-s5g36TYnbJqkhryEay8XXJqFlUKc/edit#heading=h.11pi87omqg1a). If your project is divided between src/ and bin/ directories, it won't work.  
    ALSO NOTE: a "pass" is contingent on whether your output actually meets requirements. All output will be reviewed when your project is graded.  
    ALSO ALSO NOTE: This is not the only way you can test CircuitTracer! You can and should also run CircuitTracer directly to test and debug using specific command line args and specific input files.
  + [Additional input files](https://github.com/BoiseState/CS221-Public/tree/master/assignments/CircuitTracer/files/boards) for testing, including both valid and invalid boards.

Here is a [zip file](https://github.com/BoiseState/CS221-Public/raw/master/assignments/CircuitTracer/files/CircuitTracer.zip) with all of these files.

# Extra Credit

In GUI mode, the program should initially display the original board layout as a grid of labels and a selectable list of all shortest path solutions. If the user selects a path from the list, the positions on the board that formed the path should be somehow highlighted in the grid. For an example, refer to these [screenshots](https://github.com/BoiseState/CS221-Public/tree/master/assignments/CircuitTracer/files/gui_screenshots).

# Sample Run

To specify the stack implementation in your Storage class with console output and an input file called *grid.dat*, you should use the following command-line statement to run your program:

java CircuitTracer -s -c grid.dat

Your program should then print the solutions to the console in the [required output format](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/CircuitTracer/files/valid1.out).

To specify the queue implementation in your Storage class with GUI output and an input file called *grid2.dat*, you should use the following command-line statement to run your program:

java CircuitTracer -q -g grid2.dat

Your program should then open a GUI window to display solutions, or print an appropriate usage message to the console if a GUI is unavailable.

# README

Your plain text README should follow the formatting and content guidelines [given here](https://drive.google.com/file/d/1lkysQn5_6OruZ3de0hSKqixus4MiF0AN/view?usp=sharing), with special attention given to the following:

* Analysis: address all of the questions in the *Tasks* section.
* Design: the design and major concepts of this program.
* Testing: a detailed description of your testing procedure. You should describe how you tested your class and what bugs you found and fixed by doing so.
* Discussion: a reflective discussion of your experience writing this assignment. This could be in the form of a journal.

# Video Overview

A video overview of this project is available [here](https://youtu.be/U9TsMisnKL8).

# Grading

Points will be awarded according to the following breakdown:

| **Tasks** | **Points** |
| --- | --- |
| Documentation - README (with analysis), Javadocs, comments | 20 |
| Functionality - correct results, error handling, etc. | 60 |
| Quality - code formatting, naming conventions, encapsulation, etc. | 20 |
| Extra Credit - fully functioning GUI | up to 10 |

# Required Files

Please submit the following files:

* README
* CircuitTracer.java (updated)
* CircuitBoard.java (updated)
* *Any other files required to compile CircuitTracer*

**Do not** submit any of the sample input files or the given CircuitTracerTester test class.

# Submission

Submit all files from the same directory. Do not include any unnecessary files.

Use the submission command given on your section's class web page from the directory containing your files.