\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Project: Circuit Tracer/P3

\* Class: CS221-02

\* Date: Dec 10, 2019

\* Daniel Rao

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ANALYSIS:

1. How does the choice of Storage configuration (stack vs queue) affect the order in which paths are explored in the search algorithm? (This requires more than a "stacks are LIFOs and queues are FIFOs" answer.)

Answer: The order choice of storage configuration does not affect the paths that are explored in the search algorithm. The Traces State objects aren’t

1. Is the total number of search states (possible paths) affected by the choice of stack or queue?

Answer: Total number of search states aren’t affected either.

1. What is the Big-Oh runtime for the search algorithm?

Answer: The Big(O) for the search algorithm is n.

So, O(n) = n2

1. How is memory use (the maximum number of states in Storage at one time) affected by the choice of underlying structure?

Answer: For a very large data set the use of linkedList will be better, however for a small data set the use of arrayList will be better. The cost of the node is constant whereas the cost of the linkedList is linear.

1. Does using one of the storage structures usually find a solution faster than the other? Always?

Answer: No, because we have to search through the StateStore containing all the TraceState containing the to be able to determine the best Paths.

1. Does using either of the storage structures guarantee that the first solution found will be a shortest path?

Answer: Neither of the storage structures guarantee that the first solution found will be the shortest.

OVERVIEW:

The program reads in files and depending on if its valid constructs a circuit board. If the files are not valid the program tends to throw exceptions with a very specific message. Circuit tracer class helps us take arguments from user and based on the provided args the user may be able to print the best

paths to the console. The user must provide either stack or queue

implementation of data structure (State Store) to store all the state traces.

The program also offers the user to print out the toString() version of the best paths to the console. The program however does not have any functionalities for GUI.

INCLUDED FILES:

CircuitTracer.java

CircuitBoard.java

README

COMPILING AND RUNNING:

From the directory containing all source files, compile the

driver class (and all dependencies) with the command:

$ javac CircuitTracer.java

Run the compiled class file with the command:

$ java CircuitTracer -q -c valid1.dat

OR

$ java CircuitTracer -s -c valid1.dat

OR

$ java CircuitTracer -q -g valid1.dat

OR

$ java CircuitTracer -s -g valid1.dat

Console output will give the results after the program finishes.

PROGRAM DESIGN AND IMPORTANT CONCEPTS:

The program reads in files and depending on if its valid constructs a circuit board. If the files are not valid the program tends to throw exceptions with a very specific message. Circuit tracer class helps us take arguments from user and based on the provided args the user may be able to print the best paths to the console. The user must provide either stack or queue implementation of data structure (State Store) to store all the state traces.

The program also offers the user to print out the toString() version of the best paths to the console. The program however does not have any functionalities for GUI.

These are the classes that the program interferes with:

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

1. 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.
2. CircuitBoard.java - a partially completed class that represents a circuit board.
   * Contents of circuit board can loaded from file or are part of a search state.
   * Complete code to read the input file to construct a new board.
3. TraceState.java - a completed class that represents a search state with the trace path.
4. InvalidFileFormatException.java, and OccupiedPositionException.java - exceptions used by other classes in this project to inform the user.
5. CircuitTracer.java - a partially completed driver class.  
   You will need to complete code to:
   * parse command line options.
   * configure Storage stateStore for the search.
   * run search for the best paths.
   * report results from the search.

I think that we as a class were, kind of forced to design the program in a specific way, which I don’t necessarily consider bad, but it’s just that. I might be unaware about its pros and cons. The sudo code for the search algorithm, was already kind of provided to us when we got a pretty good head start to work on it.

TESTING:

The program correctly prints the output on the console.

DISCUSSION:

I think it took me some time to understand the functionality of the program itself. Frist reading the file itself was pretty hard, since I forgot about some of the functionalities of scanner class itself. One of the hardest path was to think about all the cases in which I would be throwing those exceptions. Invalid6.dat and invalid10.java as where InvalidFileFormatException() was supposed to be thrown. After getting it fixed, I was successfully able to fix it through the use of other scanner methods. Later as the program progressed, I really had to understand how storage class and the traceState class was interacting in the program and their functionalities. The search algorithm was provided to us which made life much easier and I was able to build the program successfully.

EXTRA CREDIT:

No extra credit was attempted.

----------------------------------------------------------------------------

image will be downloaded