CSCE 420
Programing Assignment #2
due: Tues, Sept 30 (by start of class)

Objective

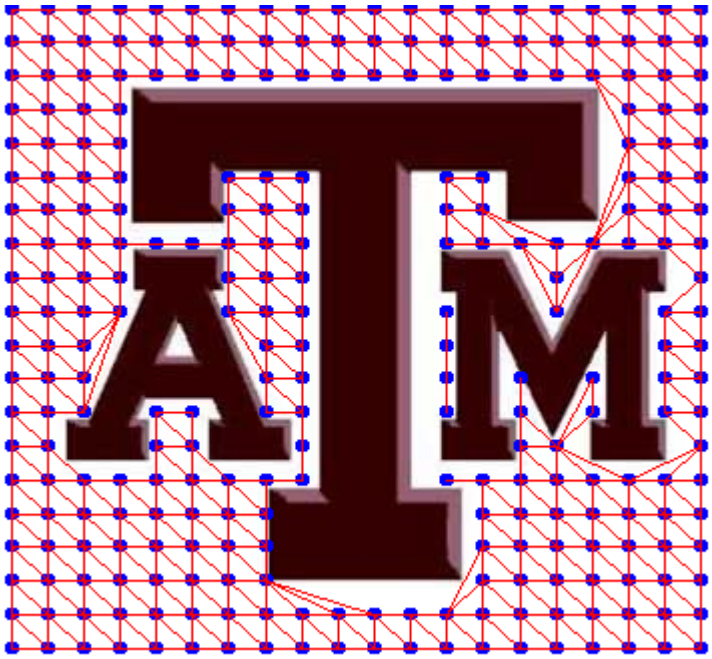
The goal of this assignment is to implement, test, and compare three different search algorithms - DFS, BFS, and GREEDY - on a simple navigation (path-finding) problem.

The focus of this assignment is on implementing a **queue-based** version of the basic search algorithm. That is, the search involves maintaining a frontier, in which nodes are popped, goal-tested, and then the successor nodes are pushed in, and the process iterates until a goal node is found. Using this framework, DFS, BFS, and GREEDY are almost identical. With BFS, you use a *queue* for the frontier. With DFS, you use a *stack*. With GREEDY, you use a *priority queue*, with the nodes sorted based on straightline distance to goal as the heuristic score.

The code must be written in C++. You will have to use *queue, stack,* and *priority_queue* from the Standard Template Library (STL). You must use g++ as a compiler (e.g. on Unix command line) rather than Visual Studio, which will facilitate the grader in compiling and testing your code.

This assignment is a stepping stone toward the next assignment, where you will be using the code written here as infrastructure to implement A* search. This simplicity of this problem (with a finite search space) will help you debug your algorithms.

The task is the same Navigation Task as from Assignment #1. For example, in the ATM graph below, we could try to go from upper-left corner to lower right, (1,1) to (20,20). Or we could try to go from inside the T on one side to the other, (7,6) to (13,6).



In Assignment 1, you implemented the queue-based version of BFS. It should be trivially easy to change it to DFS. All you have to do is change the data structure for the frontier from a *queue* to a *stack*. GREEDY search can be implemented by switching the frontier to a *priority_queue*, where nodes a popped in order of least distance to goal (i.e. using SLD as a heuristic). Note that one way to to this is to pass the Goal coordinates into the constructor for the Node class, which can calculate the heuristics score (using the Euclidean distance formula) for each node at its initialization. You will probably need to define a comparison operator for Nodes that tells the priority_queue how to keep them sorted.

Evaluation

Then you will want to compare the performance of the three algorithms on example search problems. Run each of the algorithms on the search problems above and **answer the following questions**: Which algorithm is fastest (finds goal in fewest iterations)? Which is most memory efficient (smallest max frontier size)? Which visits the fewest vertices? Which generates the shortest path length? (Is any of them optimal?)

What to Turn in

- You will submit your code for testing using the web-based CSCE *turnin* facility, which is described here: https://wiki.cse.tamu.edu/index.php/Turning_in_Assignments_on_CSNet
- You should include a Word document that shows **example program traces** (transcripts) for all 3 algorithms.
- You should also include a **table of performance metric**s for the 3 algorithms, for comparing things such as number of iterations (goal tests), maximum queue size, mean solution path length, etc., either for ≥5 selected problems, or averaged over ≥5 randomly chosen start/goal combinations. **Use the data to answer the questions listed above**.