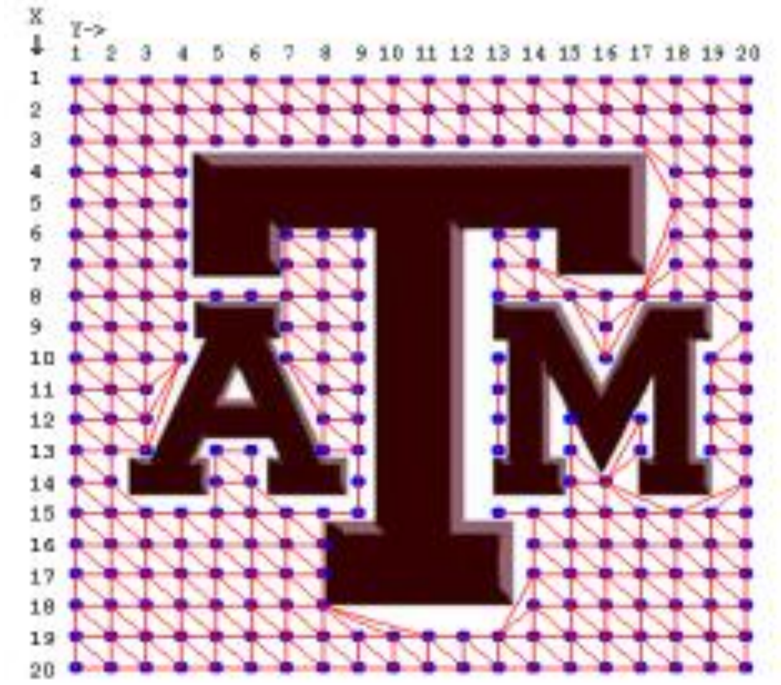
9/8/2016 10:04 AM

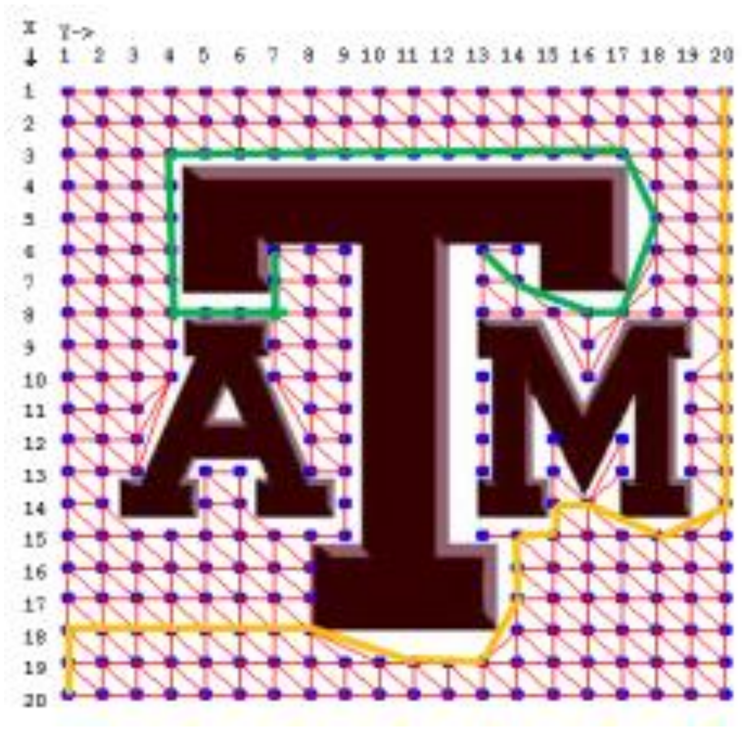
CSCE 420 – Fall 2016
Project 1
Due: Thursday, 9/29/16, class time (11:10am)

The objective of this project is to implement **GraphSearch** combined with **GreedySearch** for solving robot navigation problems (path planning) on a 2D map with obstacles. Consider the following graph…



The robot is restricted to the discrete positions (blue nodes), and may only move along the (red) edges shown. Each position on the map can be specified by a pair of coordinate, from (1,1) upper-left to (20,20) lower-right.

Given the coordinates of an arbitrary start position and goal, the objective is to find a path the robot can travel from the start to the destination. For example, the green path in the map shown below is the solution for (6,13) to (6,7), and the yellow path is the solution for (1,20) to (20,1).



The graph structure will be read in from an input file, i.e. a text file (downloaded from the course website, http://faculty.cs.tamu.edu/ioerger/cs420-fall16/ATM_graph2.txt) that lists all the vertices and edges. All lines begin with a "v" or "e" followed by a list of integers that are space-separated. "v" lines give the id and coordinates of vertices**: v ID X Y**. ID is the vertex identifier (a number), and X and Y are coordinates of the vertex (in the coordinate system shown above). "e" lines give a list of neighbors for each node: **e ID n1 n2...**, where ID is the vertex identifier and n1... are identifiers of vertices connected to ID by edges. So in the example below, vertex 216 is connected to vertices 215 and 204 (inside the "M", based on the coordinates for these nodes).

```
v 0 1 1
v 1 2 1
v 2 3 1
...
v 204 14 16
v 214 8 17
v 215 12 17
...
e 216 215 204
e 217 218 205 231 204 232
e 214 201 230 202 229 203 228 227
```

Implementation

In addition to the input file for the graph, your program should take 4 integer arguments: the coordinates of the start location and goal location. It should implement the Graph Search algorithm shown in the book (Fig 3.7). To accomplish greedy search, you should use a **priority queue** as the data structure for the frontier. Use **straight-line distance** (SLD) from each node to the goal as the **heuristic** to keep the nodes in the queue sorted. Since there are multiple paths from any one node to another, you will have to to keep track of which nodes have been **visited** (and the shortest path so far to each visited node). When you reach the goal node, you should print out:

- Search order (list of all nodes visited in chronological order)
- Solution path (nodes from root to goal node; obtain traceback by following parent pointers)
- Summary statistics: total nodes searched (iterations), length of solution path, max frontier size

The program must be implemented in C++ or Java. Your program should be able to run on other input graphs.

What to Turn In:

- Use the Turnin program on CSNet
- A document (e.g. README) that describes any details about how to compile and run your program.
- In addition to your source code, include a document showing some program outputs for the two example searches shown above and 3 more test cases of your choice (i.e. different start/goal combinations)

Sample Transcript/Output:

```
> greedy_search ATM.graph 1 1 4 4
iter 0: checking (1,1), frontier size=3
iter 1: checking (2,2), frontier size=4
iter 3: checking (3,3), frontier size=6
iter 4: checking (4,4), goal found!

solution path:
 vertex 0 (1,1)
 vertex 21 (2,2)
 vertex 42 (3,3)
 vertex 62 (4,4)

total iterations = 4
max frontier size= 6
vertices visited = 4
path length      = 3
```

Note: searches will not always start at (1,1) and will not always go straight to the goal.