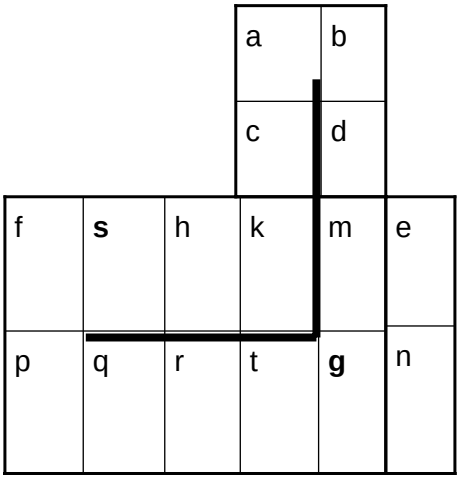
Due Date: January 27, 2016 at 11:59pm

Submission Instructions: Assignments are to be submitted through LEARN, in the Dropbox labelled *Assignment 1 Submissions* in the Assignment 1 folder. Late assignments will be accepted up until January 29 at 11:59pm. Please read the course policy on assignments submitted after the official due date. *No assignment will be accepted, for any reason, after 11:59pm on January 29.*

Lead TA: Sajin Sasy (ssasy@uwaterloo.ca). Office hours on Thursdays from 2:00pm-4:00pm in DC 3332.

Announcements: The following exercises are to be done individually. For the programming questions you may use the language of your choice.

Question 1 (12 pts)



In the above maze the successors of a cell include any cell directly to the east, south, west or north of the current cell. The exception is that no transition may pass through the central barrier. For example *successors*(m)=$\{d, g, e\}$.

The problem is to find a path from s to g. We are going to examine the order in which cells are expanded by various search algorithms.

a. (3 pts) Assume you run *depth-first search* until you expand the goal node. Assume that you always expand East first, followed by South, then West, then North. Assume your version of depth-first search avoids loops: it never expands a state of the current path. In what order do you expand the states?

b. Next you decide to use a Manhattan Distance Metric heuristic function. That is,

h(state)=shortest number of steps from state to g if there were no barriers

So $h(k) = 2$, $h(s) = 4$ and $h(g) = 0$.

***i.* (3 pts)** Is h admissible? Why or why not?

***ii.* (3 pts)** Assume you now use greedy best-first search using heuristic h, and further assume that you are using a version that never re-explores the same state twice. Give all the states in the order expanded, until the algorithm finds the goal.

***iii.* (3 pts)** Assume that you use A^* with heuristic h. Give all states in the order they are expanded. (Note that depending on the method that you use to break ties, more than one correct answer is possible.)

Question 2 (6 points)

Prove each of the following statements or give a counterexample.

a. Breadth-first search is a special case of uniform-cost search.

b. Depth-first search is a special case of best-first search.

c. Uniform-cost search is a special case of A^* search.

Question 3 (44 points)

In the next question you are asked to implement a search algorithm for solving the Traveling Salesperson Problem (TSP). In a TSP a salesperson must visit n cities, visiting every city exactly once and returning to the city they started from. The goal is to find the tour with the lowest cost. Assume that the cost of traveling from one city to the next is the Euclidean distance between the two cities. That is, given two cities with coordinates (x_1, y_1) and (x_2, y_2), the cost of traveling between the two is

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

The data can be found in the file *problem.zip* which is located in the Assignment 1 folder. The data format of the problem instances is as follows;

< number of cities >		
< city id >	< X >	< Y >
< city id >	< X >	< Y >
⋮	⋮	⋮
< city id >	< X >	< Y >

The < city id > is a capital letter. The identifiers < X > and < Y > represent the x and y coordinates (integers) of the city. Each city is on its own line and the number of cities is on the first line by itself.
Example:

3		
A	17	55
B	24	38
C	91	57

Write a program that uses the A* algorithm to solve the Traveling Salesperson Problem (TSP). Let the cost to move between cities be Euclidean distance. In the code, try to separate the domain independent part (search algorithm) from the domain specific part (TSP related functions).

a. Give a representation of the problem suitable for A* search. That is, describe states, their representation, the initial state, the goal state, the operators, and their cost.

Use the following conventions;

- Always start from node A. (You might as well since every tour has to go through A; thus A never needs to be backtracked.)
- Generate the successors of any given node in alphabetical order.
- Count nodes as they are generated as successors.

Generate and use a *good admissible* h function, and describe it in detail in the write-up. Think carefully about how to construct the h function since some choices of h functions may be too bad to allow you to execute the search in practice. If your program spends more than 15 minutes on a problem instance of 10 cities, something is very wrong: terminate the run and rethink your design.

b. Describe your heuristic function.

c. Run A* search on the provided random test instances. Note that there are 10 instances for each number of cities. Determine how many nodes, on average, A* generates for each number of cities, and plot them. Extrapolate from these results roughly how many nodes A* search would generate for a 36 city problem. To do this, you may want to use a logarithmic scale

on the y-axis. How long would such a search take? (If you want, you can try to run your implementation on the 36-city problem, but we do not require that your implementation successfully solves this particular instance).

Submit the following

- A well documented copy of your code.
- Your problem representation.
- The description of your heuristic function.
- The plot of the average number of nodes generated for each number of cities.
- A discussion of how you expect A* search to perform on a 36-city problem instance.