HARRISBURG UNIVERSITY OF SCIENCE AND TECHNOLOGY

CISC 340

Introduction to Artificial Intelligence

Lab Booklet #03

Lab Title: Heurisitic Searching ¶

Instructors

Brian Grey M.S.

Chad Van Chu M.S.

Objectives

Searching problems are central to gaining an understanding of Artificial Intelligence. We will be continuing our work from the last class by implementing two classic algorithms central to the concept of modern AI: Dijkstra's algorithm and the heuristic-based extension, A*. We will construct an algorithm so that, if fed information in the proper format, our algorithm can find the shortest path between two specified points and, potentially, display the state of the search along the way.

After completing this lab, you will be able to:

- **Create and implement a novel design for representing a graph**
- **Understand the differences in theory and application between Dijksta's search algorithm and A***
- **Understand the principle of an admissible optimistic heuristic**
- **Design and parameterize robust and necessary output**

Resources

While the lab will explain everything that you need to know, you may want to consult the following sources for additional knowledge:

- https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)
- https://en.wikipedia.org/wiki/A*_search_algorithm (https://en.wikipedia.org/wiki/A*_search_algorithm)

Deliverables

For this lab, you will need to submit:

- Jupyter notebook files (.ipynb), named and coded as instructed.
    - CISC 340 Lab 03 FA19.ipynb

Instructions & Questions

Dijkstra and A* both work on directed and undirected connected graphs where edge weights are non-negative. Given this we are going to need to construct a graph of this type and store it within our code. We can then pass this graph to the function which runs the search. The call to the function is to take the form:

```
searchXxxxx(Graph g, Node start, Node end, bit robust_output)
```

Where `Xxxxx` is the name of the algorithm, `Graph g` is the graph being searched, `Node start` is the starting point, `Node end` is the ending point, and `bit robust_output` represents whether the program will display more output while it is running. When storing your graph, you will need to store each node within the graph and the connections between the nodes. For this lab, we are going to assume that edge weights are determined by Euclidean distance. Therefore, you will also need to store the coordinate of each node.

For your implementation, you will run a search using Dijkstra and A* on the graph given to determine the shortest path for `Node start` to `Node end`. The program will output the shortest path between the two nodes, the length of the path, and the number of nodes visited for each algorithm.

Part 1: Design

First, you need to determine certain design aspects regarding your solution.

1. Explain, in detail, how you will store your graph and nodes within your code. Be sure to explain how the data will be structured internally.

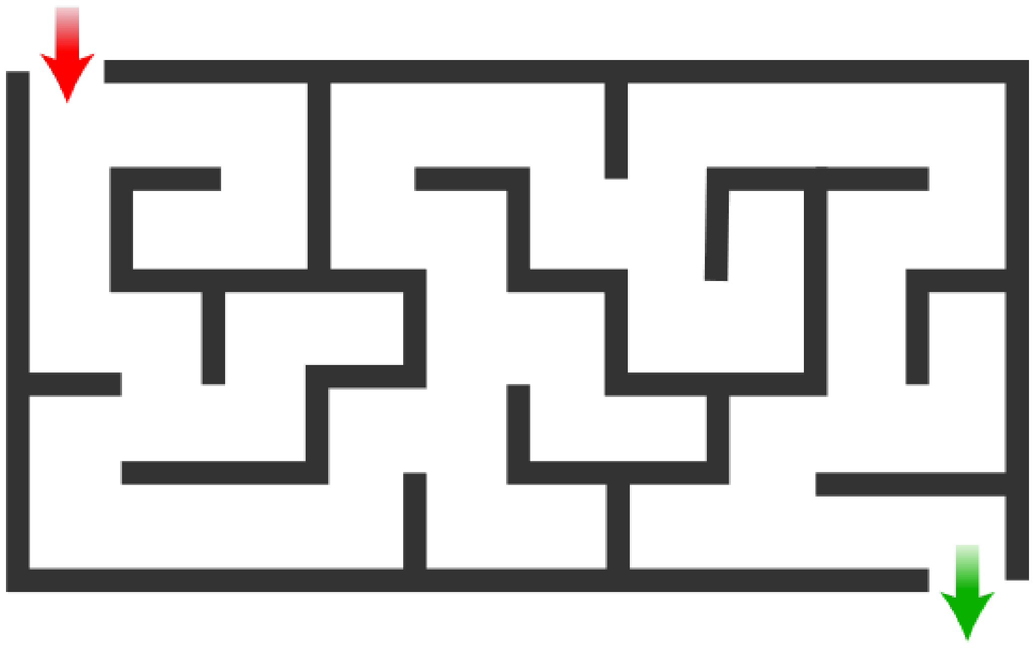
2. Explain, in detail, what additional information you will display when the *robust_output* flag is set to true. Information may be displayed before the search algorithm starts, during each iteration of searching, or after the search algorithm completes. Be sure to include when your additional data will be displayed.

Part 2: Concept

3. Translate the following maze into your graph storage methodology. You may document this using conceptual representations or code-based representations. Be sure to include "dead end" nodes in your graph.

In [1]:
```
from IPython.display import Image
Image(filename="./images/Maze.png", format="png")
```

Out[1]:



Part 3: Implementation

4. Implement your solution for the `searchDijsktra` function. You may write any helper functions that you wish. Be sure to test this code with the graph you created in Question 4, above, as well as with any other graph examples that you wish to use.

```
In [ ]:
```

5. Implement your solution for the `searchAStar` function. You may write any helper functions that you wish. Be sure to test this code with the graph you created in Question 4, above, as well as with any other graph examples that you wish to use.

```
In [ ]:
```

6. How does the performance of your `searchDijsktra` function compare with the performance of the `searchAStar` function? Explain, in detail, with whatever relevant data you have from your code.