Jake Traut

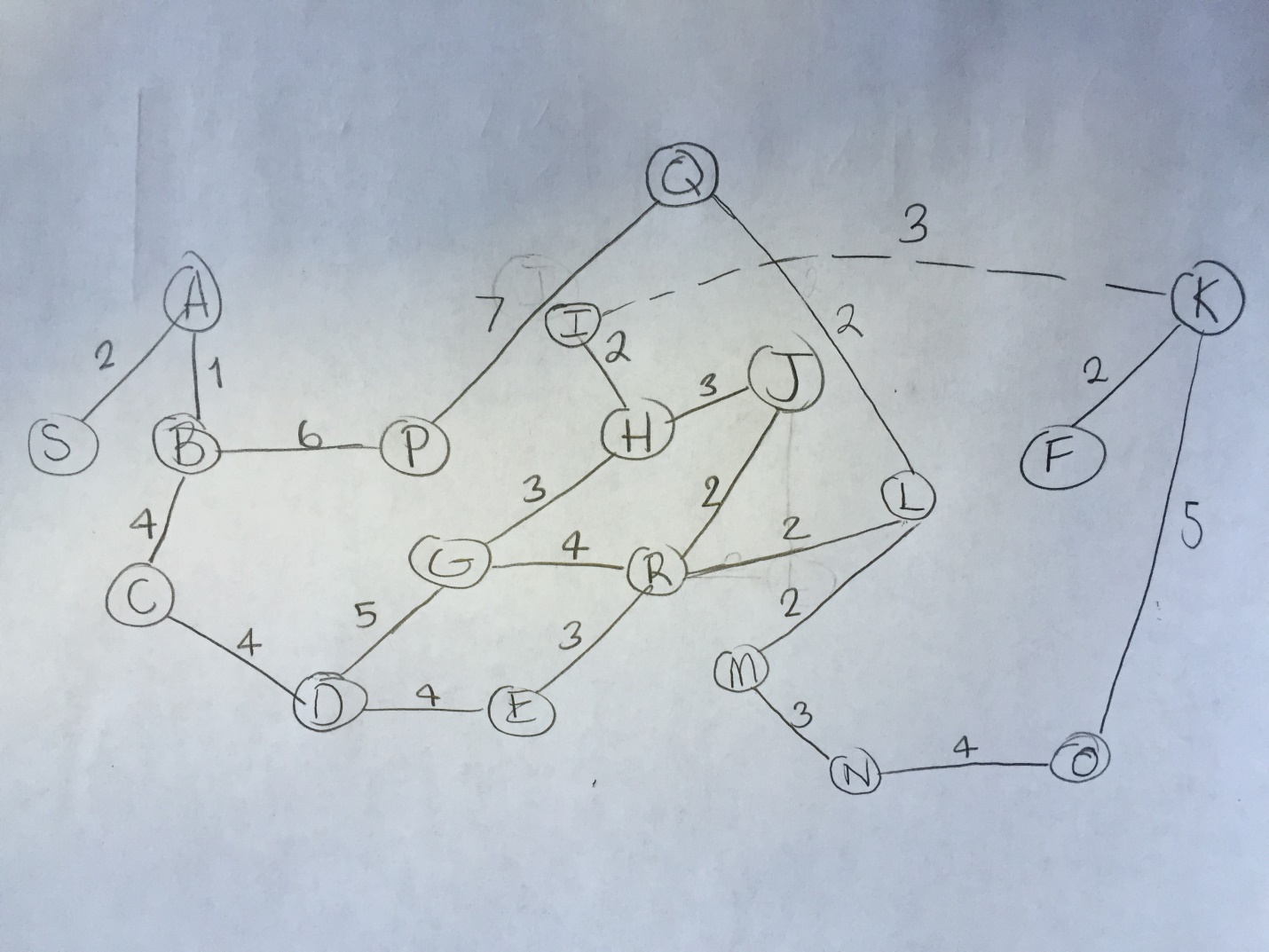
CSCI 3202 Assignment 3

**Purpose:**

This assignment got us more familiar with all the inner working and functionality of the search algorithms we have been discussing, specifically A\* and Dijkstra’s algorithm. Studying how the algorithm works and the order of evaluation can give an overview concept but actually programming the algorithms helps solidify our understanding. If you can program it then it’s safe to assume you know it.

**Data used:**

We were provided the .txt file that included all the nodes and edge weights for the graph we’re working with, as well as the heuristic values per node. Here is what the graph could possibly look like:



**Procedure:**

To start, I had to make alterations to our graph class from assignment 1 such that it included edge weights. I also changed the findVertex function of the graph so that it would return a list of the adjacent nodes instead of just returning true if adjacent nodes exist, this allowed me to pull the adjacent nodes of any given vertex from outside the graph class (i.e. in my search algorithms).

Next, I wrote a small function that generates the graph based on the data in the .txt file, and stored all the heuristic values as well. With this I was able to implement A\* algorithm, which I used the pseudocode as a guideline from the lecture notes. This took much more time Dijkstra’s, but that was to no surprise as I’m new to A\* and feel I have already internalized Dijkstra’s from previous classes. I also followed the pseudocode of the Dijkstra’s algorithm from the lecture notes but without encountering nearly as many errors.

Note code runs fully functional without need of specifying the .txt file on command line.

**Results:**

A\* solution: S A B C D G H I K F with distance of 26

Dijkstra’s solution: S A B C D G H I K F with distance of 26

A\* evaluated 18 nodes, in order: S A B C P D E Q G L H R M I J N K F

And Dijkstra’s evaluated all 19 nodes, in order: S A B C P D Q G E R L H J M I N K O F

Thus A\* is slightly more efficient than Dijkstra’s while both are still optimal in solution. Although, A\* was only optimal because the heuristic values were good. If we were given poor heuristic values then A\* becomes very ineffective.