realizing that distance is just a number.
Trailblazer To-Do

Path `breadthFirstSearch` (RoadGraph graph, RoadNode* start, RoadNode* end)
Path `dijkstrasAlgorithm` (RoadGraph graph, RoadNode* start, RoadNode* end)
Path `aStar` (RoadGraph graph, RoadNode* start, RoadNode* end)
Path `alternateRoute` (RoadGraph graph, RoadNode* start, RoadNode* end)

[Map of Middle-earth with labeled nodes and paths]
class RoadGraph {
    /* Returns the set of all the nodes adjacent to the given node. */
    Set<RoadNode*> neighborsOf(RoadNode* v) const;

    /* Given a start and end node, returns the edge that links them, or *
    * nullptr if there is no such edge. */
    RoadEdge* getEdge(RoadNode* start, RoadNode* end) const;

    /* Returns the highest speed permitted on any road in the network. */
    double getMaxRoadSpeed() const;

    /* Returns the "straight-line" distance between the two nodes; that is,
    * the distance between them if you just drew a line connecting them. */
    double getCrowFlyDistance(RoadNode* start, RoadNode* end) const;
};
class RoadNode {
    string nodeName() const; // Name of the node, for testing and debugging

    Set<RoadEdge*> outgoingEdges() const; // Outgoing edges from this node

    void setColor(Color color); // Should be one of Color::GRAY, Color::YELLOW, or Color::GREEN
                                // Node: there is no function to read colors

    string toString() const;     // For debugging
};
class RoadEdge {
    RoadNode* from() const;  // Which node this edge starts from
    RoadNode* to() const;   // Where node this edge ends at
    double cost() const;    // The cost associated with this edge
    string toString() const; // For debugging
};
using Path = Vector<RoadNode*>;
Alternate Path

**Goal:** Find best path that is at least 20% different than best path

\[
\text{diff} = \frac{\# \text{ of nodes in alt. path not in main path}}{\# \text{ of nodes in alt. path}}
\]

**Strategy:**

1. Find optimal path \textbf{start} \to \textbf{end} node
2. For each edge in optimal path, find shortest path \textbf{start} \to \textbf{end} that doesn’t use that edge
3. Return best path found in (2) that is at least 20% different than best path
A revolutionary new algorithm...

AntonSearch!
anton-search():

create an empty path

make a current node equal to the start node
color the start node green
add the start node to the path

while (the current node is not the end node) {
    randomly sample a new current node that is a neighbor of current
    color current green
    add current to the path
}

return the constructed path

Worst case? $O(\infty)$
Demo!
Extension: What if I don’t want it to be $O(\infty)$?
anton-super-search():

create an empty path

make a current node equal to the start node
color the start node green
add the start node to the path

while (the current node is not the end node) {
    if (current node has been seen more than once) {
        return an empty path
    }
    randomly sample a new current node that is a neighbor of current
    color current green
    add current to the path
}

return the constructed path

Note: this algorithm is a terrible graph search algorithm (it will rarely give you even a correct answer!). It is only meant as an exercise in writing pseudo-code.
Demo!
General questions?