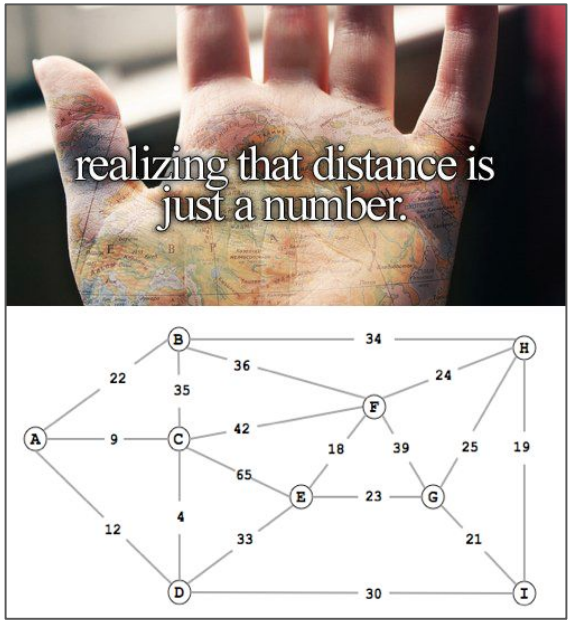
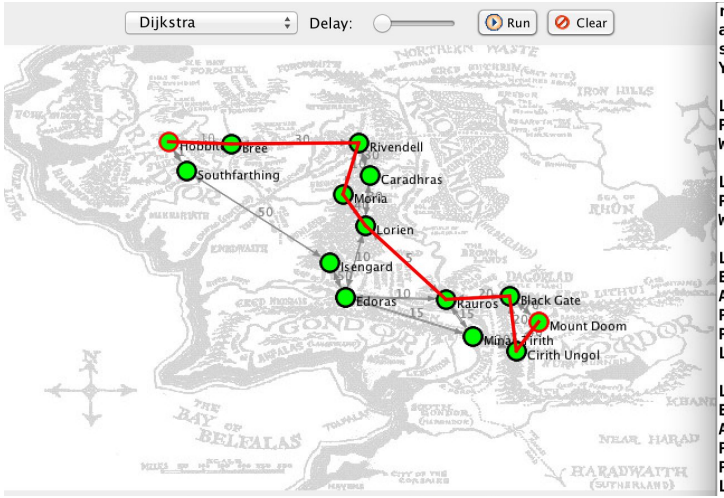
YEAH - Trailblazer

Anton Apostolatos



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)
```




```
representing roadmaps. It demonstrates several graph
algorithms for finding paths, such as breadth-first
search (BFS), Dijkstra's Algorithm, and A* search,
You can use alternate route to find a different path.

Loading world from map-usa.txt ...
Preparing world model ...
World model completed.

Loading world from map-middleearth.txt ...
Preparing world model ...
World model completed.

Looking for a path from Hobbiton to Mount Doom.
Executing breadth-first search algorithm ...
Algorithm complete.
Path length: 7
Path cost: 186
Locations visited: 14

Looking for a path from Hobbiton to Mount Doom.
Executing Dijkstra's algorithm ...
Algorithm complete.
Path length: 9
Path cost: 175
Locations visited: 14
```

RoadGraph

```
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;
};
```

RoadNode

```
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
};
```

RoadEdge

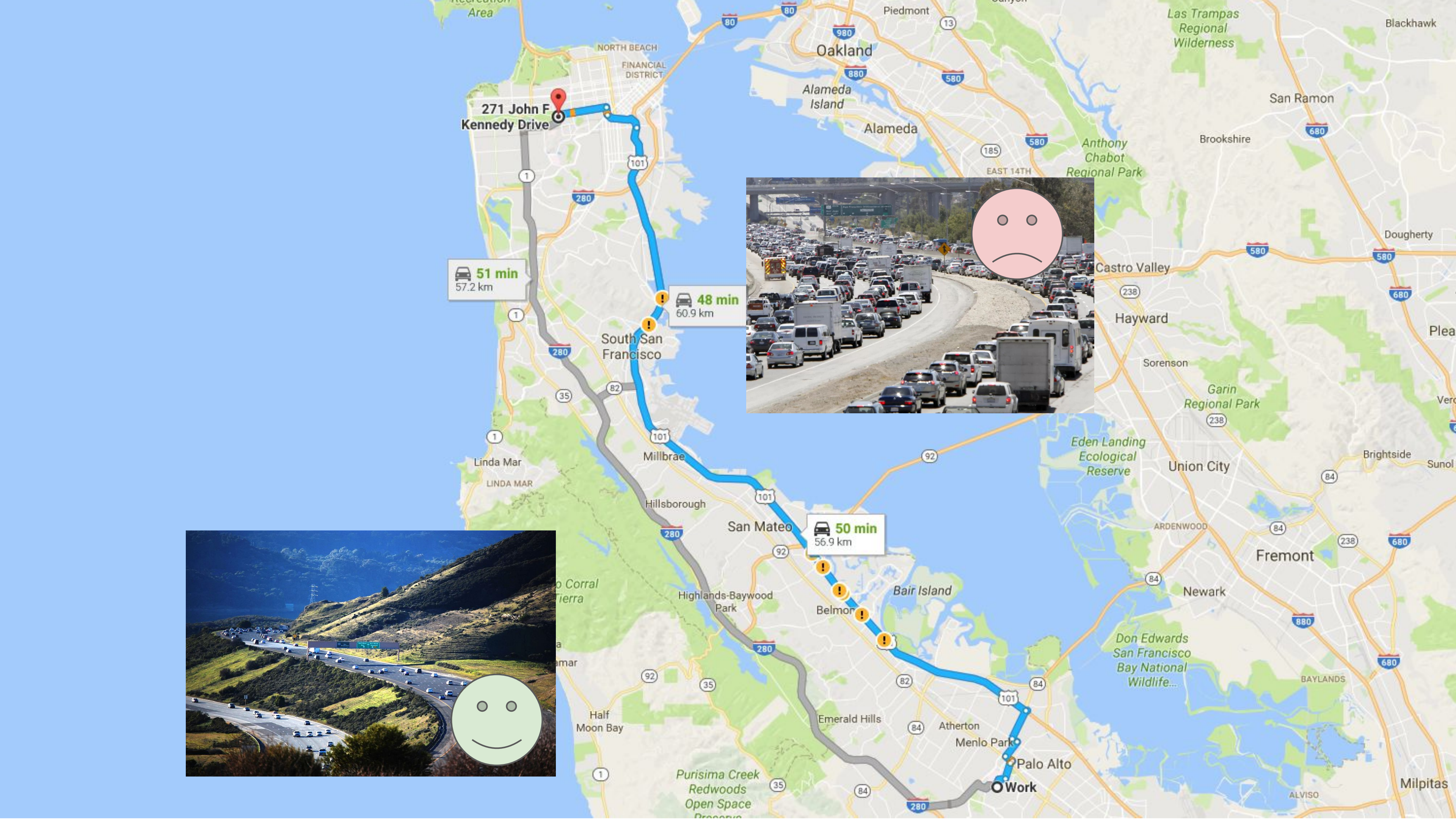
```
class RoadEdge {
    RoadNode* from() const;              // Which node this edge starts from

    RoadNode* to() const;                // Where node this edge ends at

    double cost() const;                 // The cost associated with this edge

    string toString() const;             // For debugging
};
```

Path

```
using Path = Vector<RoadNode*>;
```

```
RoadNode* current;
Vector<RoadNode*> vec;
vec.add(current);
```

```
RoadNode* current;
Path vec;
vec.add(current);
```



271 John F Kennedy Drive
51 min
57.2 km
48 min
60.9 km
50 min
56.9 km
Work
NORTH BEACH
FINANCIAL DISTRICT
Oakland
Alameda Island
Alameda
Piedmont
Las Trampas Regional Wilderness
Blackhawk
San Ramon
Brookshire
Anthony Chabot Regional Park
EAST 14TH
Dougherty
Castro Valley
Hayward
Sorenson
Garin Regional Park
Eden Landing Ecological Reserve
Union City
Brightside
Sunol
ARDENWOOD
Fremont
Newark
Don Edwards San Francisco Bay National Wildlife...
BAYLANDS
Milpitas
ALVISO
South San Francisco
Millbrae
Hillsborough
San Mateo
Bair Island
Highlands-Baywood Park
Belmont
Emerald Hills
Atherton
Menlo Park
Palo Alto
Linda Mar
LINDA MAR
Half Moon Bay
Purisima Creek Redwoods Open Space

Alternate Path

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

$$\texttt{diff} = \frac{\texttt{\# of nodes in alt. path not in main path}}{\texttt{\# of nodes in alt. path}}$$

Strategy:

1. Find optimal path **`start`** → **`end`** node
2. For each edge in optimal path, find shortest path **`start`** → **`end`** that doesn't use that edge
3. Return best path found in **(2)** that is at least 20% different than best path

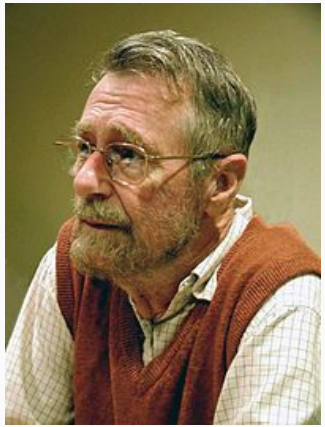
A revolutionary new algorithm...

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(∞)**



Demo!

Extension: What if I don’t want it to be O(∞)?

```
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?