SOFE 3720
Introduction to Artificial Intelligence

Winter 2019

Dr. Sukhwant Kaur Sagar

Assignment 1

Submission Deadline: Tuesday, Feb. 12, 2019, 11:59 PM

Mohamed Ibrahim 100626201
Sachin Teckchandani 100620287
Kajan Ravindran 100608620

Cost Calculation:

Each node in the XML file was queried and matched with an associated elevation parsed from the elevation file. The cost calculation uses two components to score each node.

Component 1 f(x):

Using the longitude and latitude combined with the elevation data, we calculate the straight line distance from the starting node to the goal node. This is an implementation of the Pythagorean theorem, where the straight line distance to the goal node is the hypotenuse of the right triangle created by using the current and goal nodes.

Component 2 g(x):

Similar to the method used in component 1, the second component of our calculation takes into account the straight line distance, calculated the same way using longitude and latitude and elevation.

Thus the combination of f(x)+g(x) results in our scoring for the heuristic function h(x).

This is implemented in our calculate_distance(start_node, end_node) method, which completes the calculation for any two nodes passed into it (seen in the code snippet below).

```
    def calculate_distance(start_node, end_node):
        coords_1 = (start_node.lat, start_node.lon)
        coords_2 = (end_node.lat, end_node.lon)
        height_diff = abs(start_node.elevation - end_node.elevation)
        straight_distance = geopy.distance.vincenty(coords_1, coords_2).km
        return sqrt((height_diff)**2+(straight_distance)**2)
```

Using this method to calculate the distance between two points, our astar(graph, start, end) method takes in the additional parameter of the complete graph, to use the cost function to build the most efficient path. It makes internal calls to the calculate_distance function, however, it uses current and adjacent nodes to build the path incrementally.

```
 def astar(graph, start, end):
   visited = set()
   discovered = set()
   discovered.add(start)
   trace = {}
   g = {}
   for node in graph.nodes():
       g[node] = float("inf")
   g[start] = 0

   f = {}
   for node in graph.nodes():
       f[node] = float("inf")

   f[start] = Graph.calculate_distance(start, end)

   while len(discovered) != 0:
       lowest = None
       for node in discovered:
           if lowest is None:
               lowest = node
           else:
               if f[node] < f[lowest]:
                   lowest = node
       curr = lowest
       if curr == end:
           return getTrace(trace, curr)

       discovered.remove(curr)
       visited.add(curr)
       for way in graph.adjacencyList[curr]:
           for adjacentNode in way.connected_nodes:
               if adjacentNode not in visited:
                   tmp_g = g[curr] + Graph.calculate_distance(curr, adjacentNode)
                   if adjacentNode not in discovered:
                       discovered.add(adjacentNode)
                   elif tmp_g >= g[adjacentNode]:
                       continue
                   trace[adjacentNode] = curr
                   g[adjacentNode] = tmp_g
                   f[adjacentNode] = g[adjacentNode] +
Graph.calculate_distance(adjacentNode, end)
```

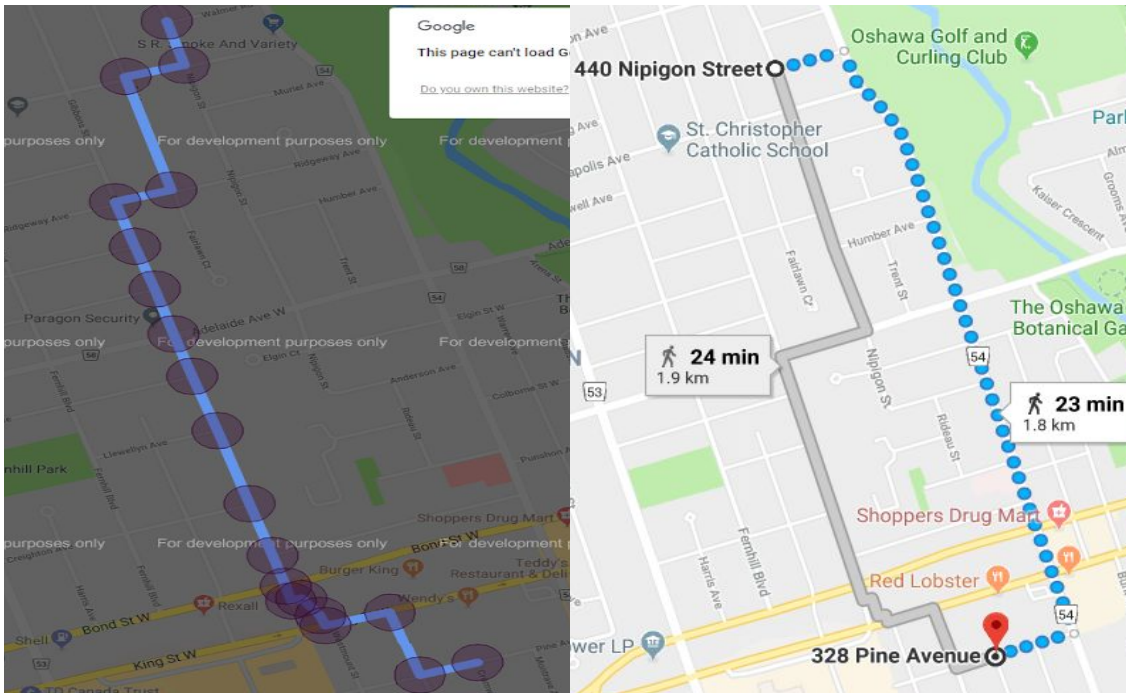
Correctness:

To determine the correctness of our generated path, the longitude, and latitude of the starting and ending points were fed into Google maps and compared to the path generated by our path nodes transposed onto the Google maps API. Although the paths were not exactly identical, our generated paths did not greatly vary from the paths generated from Google. The variation could be attributed to other data metrics Google accounts for such as traffic data, allowing walking paths from only streets and most commonly traveled routes deduced from mobile data.

```
C:\Users\100626201\Desktop>cd C:\Users\100626201\Downloads\files\walk\submission

C:\Users\100626201\Downloads\files\walk\submission>py main.py
[(43.9061816, -78.8811364), (43.905311, -78.8807938), (43.905089, -78.8819348), (43.9028576, -78.8810427), (43.9026258,
 -78.8821343), (43.9017475, -78.8817536), (43.9009098, -78.8813609), (43.9000279, -78.880998), (43.8991949, -78.8806305
), (43.8981231, -78.8801399), (43.896689, -78.8795184), (43.8956655, -78.8790663), (43.8950645, -78.8788147), (43.89477
64, -78.8787127), (43.8948247, -78.8785091), (43.8944889, -78.8781363), (43.8942897, -78.8780455), (43.8945481, -78.876
7831), (43.8933141, -78.8762017), (43.8935657, -78.8750795)]
[Node(392165038 lat: 43.9061816, lon: -78.8811364, elevation: 119.0), Node(392166748 lat: 43.905311, lon: -78.8807938,
elevation: 119.0), Node(392174889 lat: 43.905089, lon: -78.8819348, elevation: 119.0), Node(392166154 lat: 43.9028576,
lon: -78.8810427, elevation: 119.0), Node(392178728 lat: 43.9026258, lon: -78.8821343, elevation: 119.0), Node(39217205
6 lat: 43.9017475, lon: -78.8817536, elevation: 119.0), Node(392178745 lat: 43.9009098, lon: -78.8813609, elevation: 11
9.0), Node(392174328 lat: 43.9000279, lon: -78.880998, elevation: 119.0), Node(392178528 lat: 43.8991949, lon: -78.8806
305, elevation: 120.5), Node(392169026 lat: 43.8981231, lon: -78.8801399, elevation: 120.5), Node(392168370 lat: 43.896
689, lon: -78.8795184, elevation: 120.5), Node(392177425 lat: 43.8956655, lon: -78.8790663, elevation: 120.5), Node(392
166837 lat: 43.8950645, lon: -78.8788147, elevation: 120.5), Node(4588707080 lat: 43.8947764, lon: -78.8787127, elevati
on: 120.5), Node(4588707077 lat: 43.8948247, lon: -78.8785091, elevation: 120.5), Node(4588707075 lat: 43.8944889, lon:
 -78.8781363, elevation: 120.5), Node(4588707078 lat: 43.8942897, lon: -78.8780455, elevation: 120.5), Node(278543405 l
at: 43.8945481, lon: -78.8767831, elevation: 120.5), Node(278830183 lat: 43.8933141, lon: -78.8762017, elevation: 120.5
), Node(278543790 lat: 43.8935657, lon: -78.8750795, elevation: 120.5)]
```

The images shown below are a comparative test, to determine the correctness of the path from (Latitude: 43.9061816, Longitude: -78.8811364, Elevation: 119.0m) to (Latitude: 43.8935657, Longitude: -78.8750795, Elevation: 120.5m). The image on the left is the path generated from our proposed A-Star algorithm, and the image on the right is the path generated by Google maps. The route proposed by our algorithm is very similar to Google's proposed alternate route.



Similarly, in the second test between points (Latitude: 43.9081308, Longitude: -78.8597288) and (Latitude: 43.891414, Longitude: -78.862693) our algorithm deviated slightly from the proposed Google path. This can be again attributed to different possibilities. For example, our algorithm takes a path through two separate parks which are possible to do so in real life. This is likely a result of node points being available and fitting the guides of the heuristic function. The Google path however possible adheres to privacy laws and other criteria in its conservative path.