Dijkstra’s Algorithm and the Shortest Path across Campus

Dijkstra’s algorithm, first conceived in 1956 by computer scientist Edsger Wybe Dijkstra, is fundamental to pathfinding applications that we use everyday such as Waze or Google Maps. We use these applications when we go to school or work and they prove to be useful by optimizing our travel time. Although there are many paths that can be taken to any given location, Dijkstra’s finds the most efficient path in seconds by representing roads and intersections in a computer program as a graph made up of nodes and edges.

A graph is a grouping of nodes that are connected by edges. They can be classified as connected or disconnected. On a connected graph every node has a path to every other node, however a disconnected graph contains gaps whereby there is no path from some nodes to other nodes. In addition one must know whether the graph is directed or undirected. Directed graphs tell you what direction you must travel between nodes. Visually this is represented by an arrow with one point or an arrow with two points. An arrow with one point can be thought of as a one way street because you can only travel one way in between nodes. Likewise an undirected graph is made up of two way streets and the direction one travels in between nodes does not matter. Finally one must know whether the graph contains weights. A graph that is weighted contains number values on every edge that represent a distance value. In the application of path finding weights can consider distance and speed to represent the time it takes to travel along the edge. Because weights are given a number value, on a graph the edges are not visually sized to scale. An unweighted graph means that it takes the same amount of time to travel down any given edge. In the research that I did in finding a shortest path, I was only concerned with undirected, weighted, and connected graphs.

Although Dijkstra’s algorithm works through a computer program, the process can be understood in a series of steps and the visual aid of a graph. These steps will help find the shortest path from one node to every other node on Graph B, which shows a connected, weighted, and directed graph. The first step to finding the shortest path is assigning a starting node and marking all other nodes as unchecked. The second step is to look at all adjacent nodes to any checked nodes, examine their distances from the starting node and check off the one with the least distance. Adjacent nodes are nodes that are connected by only one edge to a checked node. Step two will be repeated until all nodes have been checked off. Note that the path you take when checking off a node is the shortest possible path to that node from your starting node.

The starting node on Graph B is node 0, so it will be checked off. Step two considers nodes 1 and 2. Node 2 will be checked off because it has the shortest distance. One then considers the

<table>
<thead>
<tr>
<th>NODE #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISTANCE</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>13</td>
<td>5</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>NODES USED</td>
<td>0</td>
<td>0.1</td>
<td>0.2</td>
<td>0</td>
<td>0,1,4,6,3</td>
<td>0,1,4,5</td>
<td>0,1,4,6</td>
</tr>
</tbody>
</table>
adjacent nodes 1 and 5 and will check off node 1. Nodes 5, 4, and 3 are then considered and node 4 is checked off. Subsequently nodes 5, 6, and 3 are considered and node 5 is checked off. Node 6 and 3 are the only remaining nodes and node 6 is checked off because it has a shorter distance. Node 3 is then marked as checked. This process represents how Dijkstra’s algorithm works within a computer program and yields the shortest possible distance to each node as can be seen in graph C. Suppose that your end node was node 4. Once you check off node 4 you no longer have to continue the process because it is impossible for there to be a shorter distance to node 4 than the path that you took. By marking all nodes as unvisited and visiting one node at a time based on the fact that it is the closest adjacent node, this algorithm has proof of correctness. Doing this process with brute force helps to realize the advantages of the algorithm.

To further explore the benefits of Dijkstra’s Algorithm, I sought to find the shortest path across campus from Sather Gate to Soda Hall. The first thing that I needed to do was draw out an outline of reasonable paths that one might actually take between these two locations. This outline did not include nodes. The next step was finding a graph online that was coded and that I could use as a template for my graph of campus. It took a lot of time to find a graph that could be representative of this area of campus because I had to make sure that I could assign nodes to my outline that shared the same connections as the template graph. I came across a suitable graph on www.geeksforgeeks.org and then was able to put in nodes to my graph. By looking at graph D and graph E, you can see that every node shares the same adjacent nodes. This means that every connection on each respective graph is the exact same. Both graphs contain 14 edges and 9 nodes numbered 0-8. For simplicity I measured weights under the assumption that a person would be walking the same speed no matter what the terrain was like. This allowed me to assign weights based only on relative distances.

The relative symmetry of graph D and graph E gave me a huge shortcut in finding the shortest distance from Sather Gate to Soda Hall. After downloading the programming application IntelliJ, I was simply able to alter the
weights of the edges in the code so that they were representative of the weights on the edges of my campus graph. Running the code I found online gave me the output shown in Figure 1. The left column shows the node you are traveling to and the right column shows the distance to that node from node 0. The distance to node 4, the node that represents Soda Hall, is 152. The output does not give the nodes used to travel there, but the only path that yields 152 is by traveling from node 0 to nodes 7,6,5, and then 4.

Prior to doing this project, I had not been taking the most efficient route to class. This project allowed me to not only find the most efficient path across campus, but also to explore a topic that I had no exposure to. This exploration has expanded my mind and has encouraged me to take CS61A next semester to gain further exposure to the applications of computer science.