Group Leader: Michael Beaver

Group Testing Leader: Andrew Hamilton

Group Requirement and Documentation Leader: Drew Aaron

Course: CS 355

Semester: Fall 2012

Assignment Number: 12

Assignment Type: Group 4

Assignment Description: Provide a team work schedule for final project  
Provide a proposed project

Assignment Due Date: Tuesday, November 27, 2012 (beginning of class)

To Be Included in Portfolio: YES

Total Grade: Team Schedule (50), Proposed Project (100)

**Team Schedule**

|  |  |  |  |
| --- | --- | --- | --- |
| **Dates** | **Time** | **Members Available** | **Projected Task** |
| 11/26 | 4:00pm – 6:00pm | Drew, Andrew, Michael | Develop simulation design |
| 11/27 | 4:00pm – 6:00pm | Drew, Andrew, Michael | Design and begin developing back-end |
| 11/28 | 4:00pm – 8:00pm | Drew, Andrew, Michael | Continue developing back-end |
| 11/29 | 4:00pm – 6:00pm | Drew, Andrew, Michael | Finalize back-end |
| 11/30 | 2:00pm – 5:00pm† | Drew, Michael | Begin developing front-end |
| 12/1 | 1:00pm – 4:00pm | Drew, Andrew, Michael | Continue developing front-end |
| 12/2 | 1:00pm – 4:00pm | Drew, Andrew, Michael | Finalize front-end and combine with back-end |
| 12/3 | 4:00pm – 6:00pm | Drew, Andrew, Michael | Finish combining front-end and back-end; test |
| 12/4 | 11:30am – 12:30pm | Drew, Andrew | Final testing and documenting |
| **Total Hours** | 22 hours\* |  |  |

† Andrew may or may not be available pending a doctor’s appointment

\* Group members will work individually based on free time

Group Leader: Michael Beaver

Group Testing Leader: Andrew Hamilton

Group Requirement and Documentation Leader: Drew Aaron

Course: CS 355

Semester: Fall 2012

Assignment Number: 12

Assignment Type: Group 4

Assignment Description: Provide a team work schedule for final project  
Provide a proposed project

Assignment Due Date: Tuesday, November 27, 2012 (beginning of class)

To Be Included in Portfolio: YES

Total Grade: Team Schedule (50), Proposed Project (100)

**Proposed Weighted Shortest Path Problem Simulation**

Suppose a power station needs to send electricity over a distance in miles through a number of substations to a destination. To maximize the amount of electricity that arrives, the shortest path in miles may be wished to be traversed. A longer path with more distance in between each substation may lead to reduced electricity over time, which is not desirable. Hence, the shortest possible path from the power station to the destined substation is necessary to maintain the level of electricity at sufficient levels.

This problem may be modeled by using a graph consisting of nodes and edges. The nodes represent the stations along the path from the source to the destination. The source node, in this simulation, is the power station. Every other node, including the destination, is a substation. The edges are paths between each substation. The edges may not necessarily be roads; they may be the routes traced by power lines. The weight of each edge is the distance in miles from one node to the adjacent node. There should be no cycles because, ideally, the electricity should never make its way back to the power station (source node). Assuming electricity only travels in one direction from station to station, the graph is a weighted directed acyclic graph.

To determine the shortest path needed to reach the destination, any number of shortest path algorithms may be used. The most ideal solution seems to be Dijkstra’s Algorithm, with the addition of a minimum-priority queue. This algorithm allows for the distances between nodes to be calculated for each path. The algorithm considers the weight of each path when determining the distance between nodes. A minimum-priority queue allows for organizing the nodes to be traversed. This type of queue allows for efficient selection of the closest node. Therefore, Dijkstra’s Algorithm with a minimum-priority queue is possibly the best method for finding the weighted shortest path in the graph.

User input could be implemented to follow any reasonable format. One possible format is to have the user name the first node, attach the adjacent node with an arrow, and give the weight of the edge in parentheses. This visually represents the adjacency of the nodes, and it is easy to read. For example (S = Station):

|  |
| --- |
| > Input:  > S0 -> S1 (3) |
| > S1 -> S4 (2) |
| > S3 -> S1 (1) |
| > S0 -> S2 (5) |
| > S2 -> S5 (4) |
| > S5 -> S4 (3)  > S2 -> S3 (1) |
| > S3 -> S4 (2) |

This input produces the following graph:

S0

S1

S2

S3

S4

S5

3

2

1

5

4

3

1

2

Of course, the back-end system would represent this graph either as an adjacency list or adjacency matrix. For the sake of efficiency, an adjacency list would be used.

To present the graph to the user via the console, either a breadth-first traversal or a linear traversal over the adjacency list may be used. The format of the output would be very similar to the user’s original input. For example:

|  |
| --- |
| > Graph:  > S0 ---3---> S1 |
| > S0 ---5---> S2 |
| > S1 ---2 ---> S4 |
| > S2 ---1---> S3 |
| > S2 ---4---> S5 |
| > S3 ---1---> S1 |
| > S3 ---2---> S4 |
| > S5 ---3---> S4 |
| > Done. |

The output of the weighted shortest path from the source to the destination would follow a similar formatting. However, any superfluous edges would be excluded. For example, the weighted shortest path from S0 to S4:

|  |
| --- |
| > Weighted shortest path from S0 to S4: |
| > S0 ---3---> S1 |
| > S1 ---2---> S4 |
| > Total weight: 5 |
| > Total edges: 2 |
| > Done. |

While there are four paths from S0 to S4, the weighted shortest path is through the nodes S0, S1, and S4. Hence, in this example, the shortest distance the electricity would need to travel from the power station (S0) to the destined substation (S4) is 5 miles (weight units) over two power line segments (paths).