Project 2: HOW FAR AWAY?

Group 3
Andrew McDaniel, Becky Plummer, Sowmya Viswanath

1. Overview

In this project, the goal was to design a map scheme that allowed a user to identify the best route to travel between any pair of points on the map. A route that takes the shortest time is deemed to be the best route, and does not necessarily mean shortest in terms of distance. Thus given the distance and the speed of a link, the time taken, which is the ratio of distance to the speed, is used as the measure to compute the shortest paths. Again the speed of the link is dependent on the means of transportation. For instance, highways are faster than local roads; non-stop flights are faster than those with stopovers and so on.

Unlike a typical road atlas, the means of transportation between any two points is not constricted to one. Thus it is possible that the best route might consist of taking a train to an intermediate junction, and then walking the rest of the way to the destination. Thus the map that we come up with should show the actual route to take, and not just the time. It also was required to calculate the best route between any pair of points and hence an algorithm like Dijkstra's is not sufficient. The reading provided for the project gave us an idea of how complex the distance-time function can be to compute and how people have gone about approaching this problem.

Given a set of cities along with their unique (x,y) co-ordinates, and a set of links connecting the cities along with the distance and speed for each of them, we had to come out with a map that gave the user the shortest path between any two cities. While the initial part of the problem which consisted of applying a shortest path algorithm to the above dataset was straightforward enough, the difficulties arose in representing all this information in a map. The size of the map, which is the size of a standard letter-sized sheet of paper, placed a huge constraint on the problem.

Thus this project was a test of our abilities to convey maximum information in the restricted environment of a single sheet of paper, and in a manner that is easily understandable by the user. Additionally, we were briefed that the final maps would judged by people with expertise in different domains, so we had to come up with something that would be appreciated by all.

2. Datasets

Each group had to contribute 2 datasets for the class to work with. We came up with a map of the subset of the Manhattan subway network, and that of the popular spots in Central Park.

- **Manhattan Subway Map:** This had 95 points and over 100 links connecting them. We chose points on the red, green and blue lines. We introduced variations in speed by keeping track of local trains, express trains, the Shuttle connecting Grand Central to Times Square as well as short walks between subway stations. This map was also chosen to be one of the tournament maps.
- **Central Park Map:** This map had exactly 50 points and about 100 links. We chose some interesting attractions that people would want to see, such as main entrances, Belvedere Castle and the Bethesda Water fountain. There were links of varying sizes: bus routes, walking, and horse and buggy.

3. Ideas and Evolution

In the first discussion class, we noticed that the graphical approach was very popular with a majority of the groups wanting to represent the cities and the links connecting them as an image. This brought up a lot of issues concerning the natural geometry of the map, methods to distinguish between the best routes and the others, displaying hubs and so on. Some of the other ideas included distorting the natural geometry of the map to depict shorter paths, using coloring and lines of different thickness to indicate the best route, paths etc. Another interesting idea that came up was the use of text instead of graphics to display the map information.

The idea of using text and representing the map as a chart appealed to us mainly because of its simplicity and ease of use. The map when printed would consist of a matrix that would be indexed by the city names. The intersection of any two cities would give the user the next city on the path to the destination, and the corresponding next link. Thus for any given pair of cities, the chart would give you the next shortest path to take without any ambiguity. To find the path between any two cities, the user looks up the links for all the intermediate cities till he reaches the destination.

The text approach also did not have all the other problems that constrained the graphical approach.

- The natural geometry of the map is not important
- Distortion did not come into the picture as representing the shortest path in terms of distance is not an issue with the chart
- Cluttering of the map due to cities or links being close to each other is not a concern here
- There is no ambiguity in the best link to take, as the user is given only one option which is the next best link to the destination
- The chart is concise and accurate
- There is no requirement to show alternate paths
- It is also a novel approach that distinctly stands out from the rest

4. All Pairs Shortest Path Algorithm

When presented with the "How Far Away" problem we decided to focus on presenting the user with a map that they can use to find the exact path between two places, which takes the shortest amount of time. We decided that an all pairs shortest path algorithm would be the best solution to determining the shortest time path between all pairs of points. The Floyd-Warshall algorithm for computing the shortest path was a great place to start. Even though the algorithm does compute the shortest time it takes to go from every point to every other point in the graph, if does not keep track of the links the user would need to travel. We chose this algorithm, because it is guaranteed to be complete as long as the graph is connected and in our circumstances the graph is connected. This algorithm computes the shortest path between all pairs in $O(n^3)$ time. The other algorithm considered was Dijkstra's algorithm which computes the shortest paths between a single source and all other destinations. In implementing this algorithm we would have to run Dijkstra's algorithm on each of the sources. Dijkstra's algorithm is also complete if the graph is connected, but would take $O(n^4)$ time. Therefore we decided to implement a version of the All Pairs Shortest Path algorithm with some modifications.

```
Algorithm AllPairsShortestPaths(G):
    Input: A simple weighted digraph graph without negative-weight cycles.
    Output: A numbering $v_1, v_2, \ldots, v_n$ of the verities of G and the matrix D,
    such that D[i, j] is the distance from $v_i$ to $v_j$ in G.

    Let $v_1, v_2, \ldots, v_n$        be an arbitrary numbering of the vertices of G.

    for i ← 1 to n do
        for j ← 1 to n do
            if i = j then
                D[i, j] ← 0
            if ($v_i$ , $v_j$) is an edge in G then
                D[i, j] ← w(($v_i$ , $v_j$))
            else
                D[i, j] ← + ∞

    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                D[i, j] ← min{ D[i, j],  D[i, k] + D[k, j] }

    return matrix D
```

A dynamic programming algorithm to compute all-pairs shortest path distances in a digraph graph without negative edge cycles.[1]

[1] Goodrich, Michael T. and Roberto Tamassia, *Algorithm Desgin*, John Wiley & Sons, Inc. 2002, pp355.

This algorithm first initializes the matrix so that points which are the same have 0 distance between them, points with a direct link between them are labeled with the distance of that direct link, and points that have no direct link between them are labeled with an infinite distance. Then the algorithm picks two points, i and j, and for each third point, k, calculates if there is a shorter path from point i to point j through point k.

First, we had to make some minor changes in order to adapt the algorithm for an undirected graph. Namely, we would not be using a complete matrix in the calculation but only a corner of the matrix as the shortest path between two destinations (in our problem) is the same path traversing in both directions. We chose to make this mutation, because we don't need to waste time computing the shortest path between i and j and then j and i. It is a little bit less than half, because we ignore the case of traveling from a point to itself.

	0	1	2	3	4	5	6
0	-	🟧	🟧	🟧	🟧	🟧	🟧
1		-	🟧	🟧	🟧	🟧	🟧
2			-	🟧	🟧	🟧	🟧
3				-	🟧	🟧	🟧
4					-	🟧	🟧
5						-	🟧
6							-

The orange region represents the part of the matrix we make use of in our calculations.

In making this first change we were able to reduce the runtime. The algorithm then became three for loops:

```
for(int k = 0; k < n; k++)
        for(int i = 0; i < (n-1); i++)
                for(int j = (i+1); j < n; j++){
                        // do calculations
                }
```

In this section of code, i is the source, j the destination, and k the intersection point. The index k must still range over the entire set of points, because we want to check every middle point in the graph. Then i must range over the entire set of points other than the last one. Index j must only range from i + 1 (we don't care to calculate the case when i is the same as j) to n. The reason that i can ignore the nth point in the data set is that when i is n, j would be n+1, which doesn't exist so why waste the cycles to check it.

Secondly, we had to change the algorithm to accommodate the case when k is greater than i and k is greater than i. If we left it as the algorithm sample above we would be trying to access parts of the matrix in the lower left corner and that would cause errors. Therefore we check and make alterations based on our findings:

```
p1 = ( i < k )? matrix[i][k] : matrix[k][i];      // path distance matrix[i, k]
p2 = ( j < k )? matrix[j][k] : matrix[k][j];      // path distance matrix[k, j]
p3 = matrix[i][j];                                // path distance matrix[i ,j]
```

Here it is obvious that we check to make sure that we are not accessing memory in which we are not planning to calculate paths. Note: this is only possible, because the shortest paths are the same in both directions between the two points.

Lastly, we had to consider the largest problem with the All Pairs Shortest Path algorithm. The algorithm does in fact calculate the shortest distance between two nodes, but it does not keep any kind of record of the links traversed in the path creation. We attacked this problem piece by piece. First, we created a class MapShortestPath to contain the source and destination nodes, the path time, and a vector of the links in the path (in the order that they were inserted):

```
public MapCity node1,node2;          // end points
public double pathTime = 99999999;// total time of the path
public Vector path = new Vector();   // links in the path
```

We initialize the path time to a large number that would probably be larger than the speed of a link. Next, we had to figure out how we were going to get the shortest path into the path vector. We traversed the algorithm on a simple integer data set so that we could deeply understand the workings of the algorithm and how it skipped around to create the shortest path. We discovered that when a new shorter path is discovered the existing path is discarded and replaced by the new path. This was a pleasant surprise having never traced the algorithm we thought that perhaps the paths were added, but after testing it is obvious that this is what happens. It is especially evident from:

$$D[i, j] \leftarrow \min\{ D[i, j],\ D[i, k] + D[k, j] \}$$

We were not completely convinced by this initially because this is dealing only with path time and not path construction. Therefore, we wrote a simple replace algorithm to fuse the two new paths and replace the old.

```
public void newPath(Vector p1, Vector p2){

        // reinitialize the path and time
        path.removeAllElements();
        pathTime = 0;

        // add all the parts of path 1
        for(int i = 0; i < p1.size(); i++){
                link = (MapLink)p1.get(i);
```

```
                        path.addElement(link);
                        pathTime += link.time;
                }

                // add all the parts of path 2
                for(int i = 0; i < p2.size(); i++){
                        link = (MapLink)p2.get(i);
                        path.addElement(link);
                        pathTime += link.time;
                }
        }
```

This function simply takes the two paths from i to k and k to j and fuses them to create the path from i to j from the function call within the shortest path algorithm:

```
        if(p3.pathTime > p1.pathTime + p2.pathTime)
                matrix[i][j].newPath(p1.path,p2.path);
```

This algorithm is also guaranteed to be complete as long as the graph is connected. However, the slight drawback with our implementation is that the links are inserted in the order in which the paths before the current fusing were created. Therefore, the paths are not sorted moving from i to j going through each intermediate point in succession, however all the necessary links are present in the path. We decided to deal with the sorting of the path when we create the Path Matrix for display.

We feel that this mutation of the all pairs shortest path algorithm was a good choice, because we were focused on giving the user an exact shortest path between any pair of points in the graph, it is guaranteed to be complete on a connected graph, and it is faster than running Dijkstra's algorithm on each point in the set. We believe this to be a very interesting part of the How Far Away problem, because we were able to do some algorithm exploring and mutation.

5. Computation of the First-Step Matrix

5.1 Reasoning

The challenge was to condense as much information as possible into a small space, yet the user would still be able to navigate through the shortest path between any two points on the graph. The Matrix our program computes exploits the fact the fact that if a path passes through n points (nodes 1 → 2 → 3→ … → N) then the path of n-1 points alter moving through the first graph in an edge will be congruent to the original path. Therefore, we need only to provide the first and last steps for each path (since paths are symmetrical, the first step in moving from A → B is also the last step in moving from B → A.)

5.2 Approach

All the functionality for creating and representing the First-Step matrix is contained in the class PathMatrix. This class has a single constructor with two parameters: An index of cities and a collection of paths. The index of cities defines how the cities are index in the matrix. This way, the code from other parts of the player can easily decide in which manner the matrix should be index without having to change the underlying code. For example, the code could order the cities on the final output alphabetically, or calculate the "hubs" and list those first.

The second argument is a matrix of MapShortestPath objects. Each element in this array contains a list of the paths between any two points on the graph. This matrix is only half filled in the upper triangular. This is because the paths are symmetrical, so the code to computer shortest paths does not bother with redundant work. After saving some descriptive information, the constructor then calls the computerMatrix( ) function call, which populates the final matrix. The contents of each cell is a Hop class, which is a simple wrapper class around a point city and link objects along with a boolean value to denote a blank hop (because you cannot to/from the same city, the diagonal of the resulting matrix is filled with 'blank' cells.). The program examines every path in the tree and fills in two of the cells on the table for the matching end points of the path. The final result is that it examines $(n^2-n)/2$ paths to fill n^2-n cells.

After computeMatrix( ) finishes, the PathMatrix object is now initialized as a functional wrapper around the path matrix. To the rest of the code, they underlying data structures aren't terribly important, and by using the access methods can treat it much like a simple array. By drawing on these benefits of object orient programming, this allow changes and ideas to be tried out on either the path calculation and representation sides or the final display side without much fear of breaking the other parts of the program.

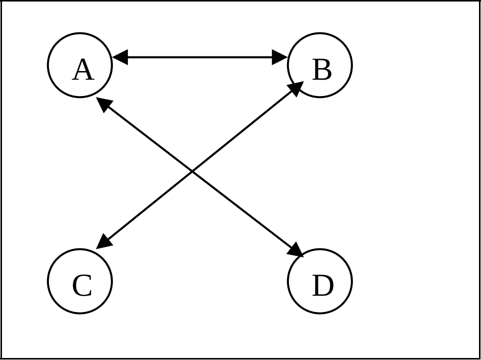
5.3 Formatting the Result

By far the biggest challenge is formatting the text information so that as many complete sets of data can be presented in the final printout as possible. We considered a number of different ways to do this, but eventually decided a large table with all the information was the best way to go.

To accomplish this, we wrote a method for PathMatrix called "printTable." printTable has a single argument, a PrintStream object. PrintStream was chosed for a very specific reason, because the standard out stream in java, System.out , is a PrintStream object. In addition, it is very easy to write to a file through a PrintStream, and therefore allowed us to easily redirect our text output a number of different ways. This let us play around with the text formatting without having to spend a lot of time worry about the graphics output.

The strategy for using the graphics was to setup a table where a user could easily look up the next link to take based on their current position (rows) and their intended destination (columns). First, it was decided to limit all labels fields to exactly 5 characters. If a label for a city or link is longer than 5 letters then it is truncated. If it is shorter, then spaces are added to pad out the column. Vertical pipe bars (|) were used to separate fields, with double bars (||) used to denote the border between the row index and matrix contents. Equals signs (=) were used to separate the column index from the matrix. For the "blank" cells along the diagonal, 5 dashes (-----) were used. Each row of the matrix actually had two lines of text, one for the City, and one for the Link name. The resulting table is displayed in a fixed width font, it appears as an evenly spaced table, like so:

```
     ||DestA|DestB|DestC|DestD
DestA||-----|DestB|DestB|DestD
     ||-----|Link1|Link1|Link2
DestB||DestA|-----|DestC|DestA
     ||Link1|-----|Link3|Link1
DestC||DestB|DestB|-----|DestB
     ||Link3|Link3|-----|Link3
DestD||DestA|DestA|DestA|-----
     ||Link2|Link2|Link2|-----
```

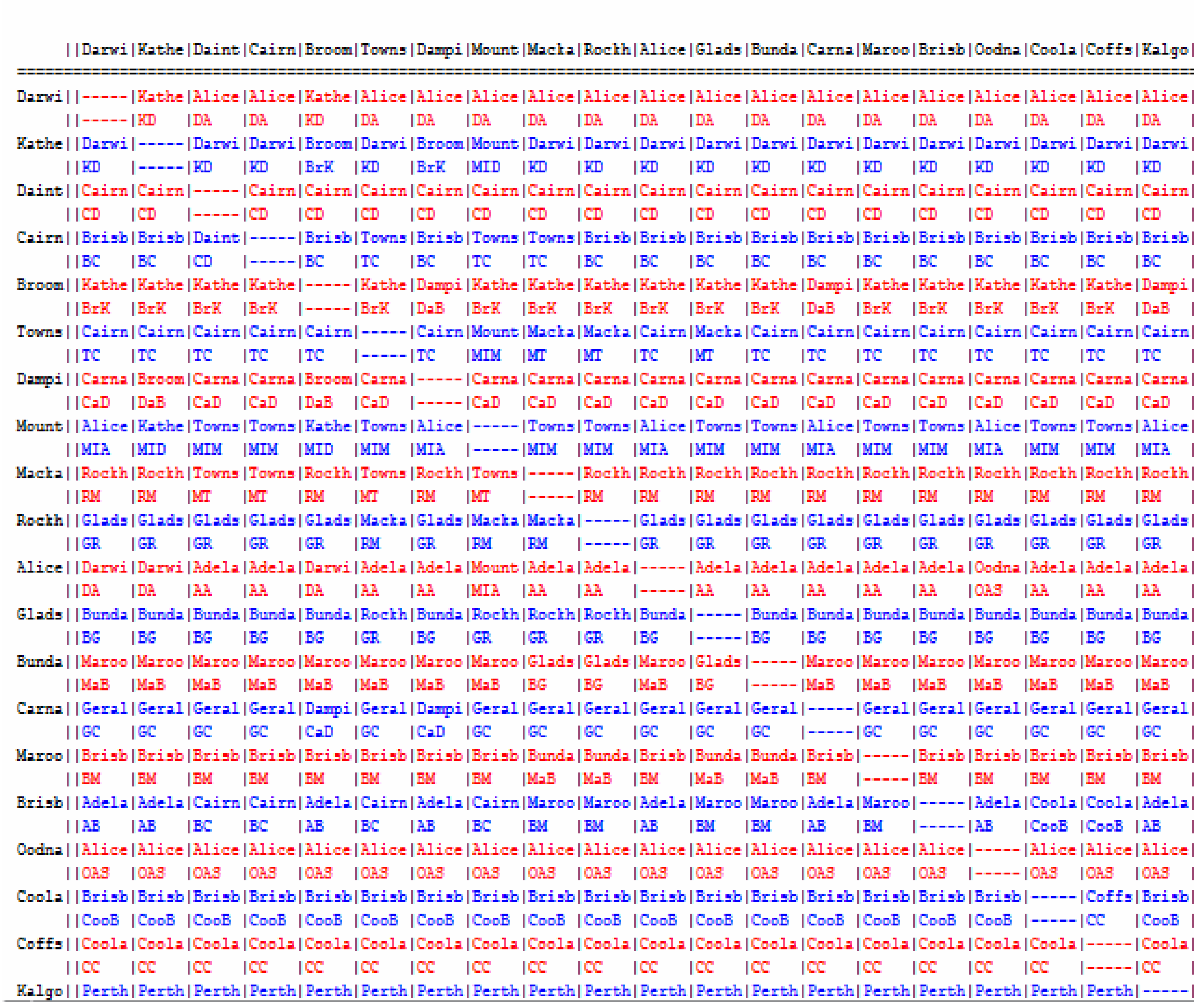


Resulting Table Source Graph

In order to print a copy of the table to the simulator screen, we created a simple extension of the PrintStream class with the println( ) method over ridden. Then it was a single matter of passing the string into the simulator image via this method.

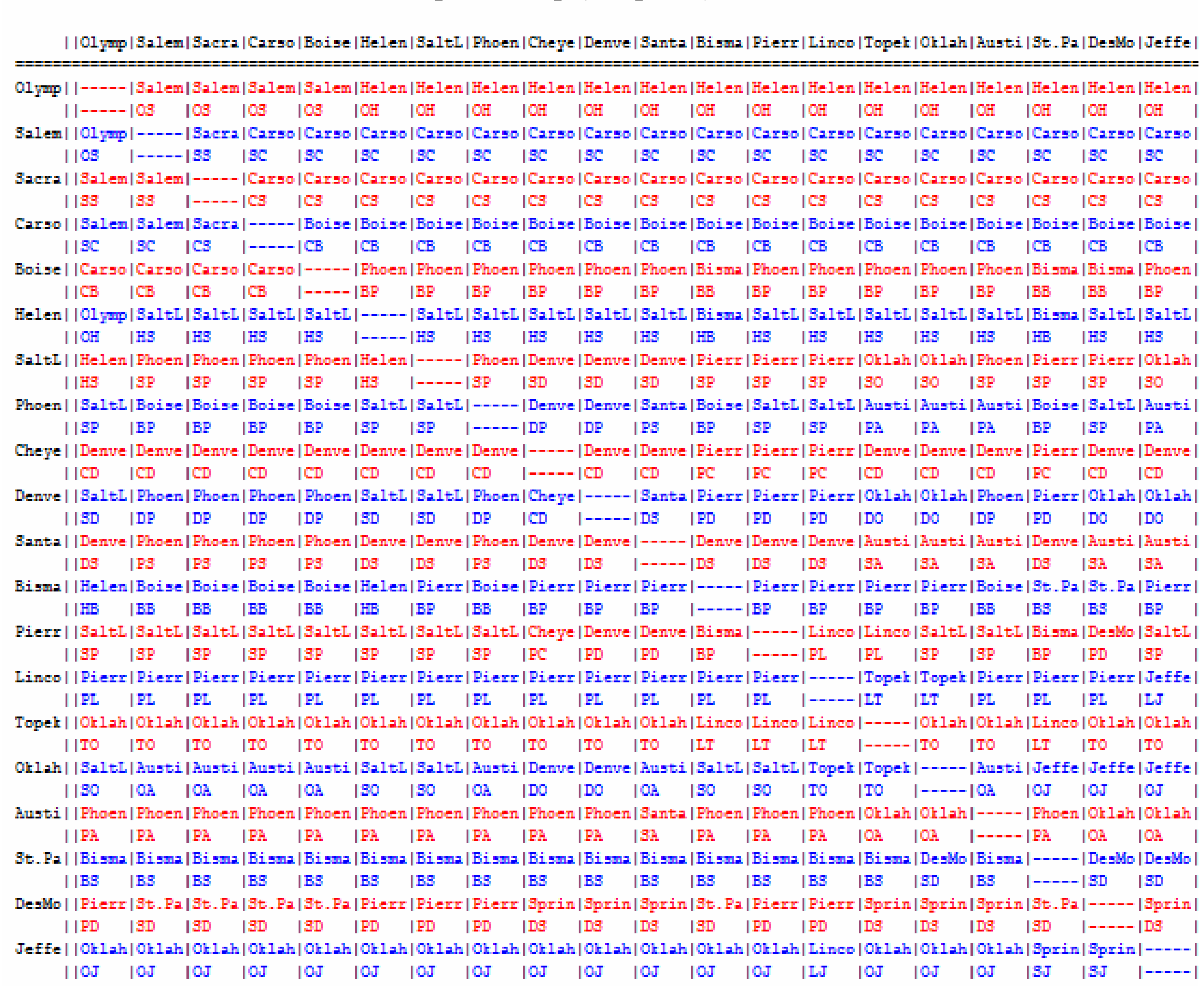
After some experimentation, it became apparent that we wanted to add a good deal of formatting and colors to our tables to make them somewhat easier to read. Unfortunately, the java tool kit is not too robust, and converting this from text to an image and then printing it seemed like an inefficient method of going about things. So we copied the code from printTable to a new method called printTableHTML, and then adapted the code to provide html coding tags around the table for coloring, size, and spacing. After this, it was a matter of loading the resulting html file into a word processor so that the file could be printed properly on a single page. (No changes to formatting were made at this point, aside from lowering font size to it on a single page.) We chose the color scheme of alternating red and blue rows, with black indexes and division bars to enable the user to quickly locate the information they are interested in despite of the small font size constraints of the chart.

The Secret Map (Snapshot)



```
     ||Darwi|Kathe|Daint|Cairn|Broom|Towns|Dampi|Mount|Macka|Rockh|Alice|Glads|Bunda|Carna|Maroo|Brisb|Oodna|Coola|Coffs|Kalgo|
===============================================================================================================================
Darwi||-----|Kathe|Alice|Alice|Kathe|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|
     ||-----|KD   |DA   |DA   |KD   |DA   |DA   |DA   |DA   |DA   |DA   |DA   |DA   |DA   |DA   |DA   |DA   |DA   |DA   |DA   |
Kathe||Darwi|-----|Darwi|Darwi|Broom|Darwi|Broom|Mount|Darwi|Darwi|Darwi|Darwi|Darwi|Darwi|Darwi|Darwi|Darwi|Darwi|Darwi|Darwi|
     ||KD   |-----|KD   |KD   |BrK  |KD   |BrK  |MID  |KD   |KD   |KD   |KD   |KD   |KD   |KD   |KD   |KD   |KD   |KD   |KD   |
Daint||Cairn|Cairn|-----|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|
     ||CD   |CD   |-----|CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |
Cairn||Brisb|Brisb|Daint|-----|Brisb|Towns|Brisb|Towns|Towns|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|
     ||BC   |BC   |CD   |-----|BC   |TC   |BC   |TC   |TC   |BC   |BC   |BC   |BC   |BC   |BC   |BC   |BC   |BC   |BC   |BC   |
Broom||Kathe|Kathe|Kathe|Kathe|-----|Kathe|Dampi|Kathe|Kathe|Kathe|Kathe|Kathe|Kathe|Dampi|Kathe|Kathe|Kathe|Kathe|Kathe|Dampi|
     ||BrK  |BrK  |BrK  |BrK  |-----|BrK  |DaB  |BrK  |BrK  |BrK  |BrK  |BrK  |BrK  |DaB  |BrK  |BrK  |BrK  |BrK  |BrK  |DaB  |
Towns||Cairn|Cairn|Cairn|Cairn|Cairn|-----|Cairn|Mount|Macka|Macka|Cairn|Macka|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|Cairn|
     ||TC   |TC   |TC   |TC   |TC   |-----|TC   |MIM  |MT   |MT   |TC   |MT   |TC   |TC   |TC   |TC   |TC   |TC   |TC   |TC   |
Dampi||Carna|Broom|Carna|Carna|Broom|Carna|-----|Carna|Carna|Carna|Carna|Carna|Carna|Carna|Carna|Carna|Carna|Carna|Carna|Carna|
     ||CaD  |DaB  |CaD  |CaD  |DaB  |CaD  |-----|CaD  |CaD  |CaD  |CaD  |CaD  |CaD  |CaD  |CaD  |CaD  |CaD  |CaD  |CaD  |CaD  |
Mount||Alice|Kathe|Towns|Towns|Kathe|Towns|Alice|-----|Towns|Towns|Alice|Towns|Towns|Alice|Towns|Towns|Alice|Towns|Towns|Alice|
     ||MIA  |MID  |MIM  |MIM  |MID  |MIM  |MIA  |-----|MIM  |MIM  |MIA  |MIM  |MIM  |MIA  |MIM  |MIM  |MIA  |MIM  |MIM  |MIA  |
Macka||Rockh|Rockh|Towns|Towns|Rockh|Towns|Rockh|Towns|-----|Rockh|Rockh|Rockh|Rockh|Rockh|Rockh|Rockh|Rockh|Rockh|Rockh|Rockh|
     ||RM   |RM   |MT   |MT   |RM   |MT   |RM   |MT   |-----|RM   |RM   |RM   |RM   |RM   |RM   |RM   |RM   |RM   |RM   |RM   |
Rockh||Glads|Glads|Glads|Glads|Glads|Macka|Glads|Macka|Macka|-----|Glads|Glads|Glads|Glads|Glads|Glads|Glads|Glads|Glads|Glads|
     ||GR   |GR   |GR   |GR   |GR   |RM   |GR   |RM   |RM   |-----|GR   |GR   |GR   |GR   |GR   |GR   |GR   |GR   |GR   |GR   |
Alice||Darwi|Darwi|Adela|Adela|Darwi|Adela|Adela|Mount|Adela|Adela|-----|Adela|Adela|Adela|Adela|Adela|Oodna|Adela|Adela|Adela|
     ||DA   |DA   |AA   |AA   |DA   |AA   |AA   |MIA  |AA   |AA   |-----|AA   |AA   |AA   |AA   |AA   |OAS  |AA   |AA   |AA   |
Glads||Bunda|Bunda|Bunda|Bunda|Bunda|Rockh|Bunda|Rockh|Rockh|Rockh|Bunda|-----|Bunda|Bunda|Bunda|Bunda|Bunda|Bunda|Bunda|Bunda|
     ||BG   |BG   |BG   |BG   |BG   |GR   |BG   |GR   |GR   |GR   |BG   |-----|BG   |BG   |BG   |BG   |BG   |BG   |BG   |BG   |
Bunda||Maroo|Maroo|Maroo|Maroo|Maroo|Maroo|Maroo|Maroo|Glads|Glads|Maroo|Glads|-----|Maroo|Maroo|Maroo|Maroo|Maroo|Maroo|Maroo|
     ||MaB  |MaB  |MaB  |MaB  |MaB  |MaB  |MaB  |MaB  |BG   |BG   |MaB  |BG   |-----|MaB  |MaB  |MaB  |MaB  |MaB  |MaB  |MaB  |
Carna||Geral|Geral|Geral|Geral|Dampi|Geral|Dampi|Geral|Geral|Geral|Geral|Geral|Geral|-----|Geral|Geral|Geral|Geral|Geral|Geral|
     ||GC   |GC   |GC   |GC   |CaD  |GC   |CaD  |GC   |GC   |GC   |GC   |GC   |GC   |-----|GC   |GC   |GC   |GC   |GC   |GC   |
Maroo||Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Bunda|Bunda|Brisb|Bunda|Bunda|Brisb|-----|Brisb|Brisb|Brisb|Brisb|Brisb|
     ||BM   |BM   |BM   |BM   |BM   |BM   |BM   |BM   |MaB  |MaB  |BM   |MaB  |MaB  |BM   |-----|BM   |BM   |BM   |BM   |BM   |
Brisb||Adela|Adela|Cairn|Cairn|Adela|Cairn|Adela|Cairn|Maroo|Maroo|Adela|Maroo|Maroo|Adela|Maroo|-----|Adela|Coola|Coola|Adela|
     ||AB   |AB   |BC   |BC   |AB   |BC   |AB   |BC   |BM   |BM   |AB   |BM   |BM   |AB   |BM   |-----|AB   |CooB |CooB |AB   |
Oodna||Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|Alice|-----|Alice|Alice|Alice|
     ||OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |OAS  |-----|OAS  |OAS  |OAS  |
Coola||Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|Brisb|-----|Coffs|Brisb|
     ||CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |CooB |-----|CC   |CooB |
Coffs||Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|Coola|-----|Coola|
     ||CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |CC   |-----|CC   |
Kalgo||Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|Perth|-----|
```

US Capitals Map (Snapshot)



```
     ||Olymp|Salem|Sacra|Carso|Boise|Helen|SaltL|Phoen|Cheye|Denve|Santa|Bisma|Pierr|Linco|Topek|Oklah|Austi|St.Pa|DesMo|Jeffe|
===============================================================================================================================
Olymp||-----|Salem|Salem|Salem|Salem|Helen|Helen|Helen|Helen|Helen|Helen|Helen|Helen|Helen|Helen|Helen|Helen|Helen|Helen|Helen|
     ||-----|OS   |OS   |OS   |OS   |OH   |OH   |OH   |OH   |OH   |OH   |OH   |OH   |OH   |OH   |OH   |OH   |OH   |OH   |OH   |
Salem||Olymp|-----|Sacra|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|
     ||OS   |-----|SS   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |SC   |
Sacra||Salem|Salem|-----|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|Carso|
     ||SS   |SS   |-----|CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |CS   |
Carso||Salem|Salem|Sacra|-----|Boise|Boise|Boise|Boise|Boise|Boise|Boise|Boise|Boise|Boise|Boise|Boise|Boise|Boise|Boise|Boise|
     ||SC   |SC   |CS   |-----|CB   |CB   |CB   |CB   |CB   |CB   |CB   |CB   |CB   |CB   |CB   |CB   |CB   |CB   |CB   |CB   |
Boise||Carso|Carso|Carso|Carso|-----|Phoen|Phoen|Phoen|Phoen|Phoen|Phoen|Bisma|Phoen|Phoen|Phoen|Phoen|Phoen|Bisma|Bisma|Phoen|
     ||CB   |CB   |CB   |CB   |-----|BP   |BP   |BP   |BP   |BP   |BP   |BB   |BP   |BP   |BP   |BP   |BP   |BB   |BB   |BP   |
Helen||Olymp|SaltL|SaltL|SaltL|SaltL|-----|SaltL|SaltL|SaltL|SaltL|SaltL|Bisma|SaltL|SaltL|SaltL|SaltL|SaltL|Bisma|SaltL|SaltL|
     ||OH   |HS   |HS   |HS   |HS   |-----|HS   |HS   |HS   |HS   |HS   |HB   |HS   |HS   |HS   |HS   |HS   |HB   |HS   |HS   |
SaltL||Helen|Phoen|Phoen|Phoen|Phoen|Helen|-----|Phoen|Denve|Denve|Denve|Pierr|Pierr|Pierr|Oklah|Oklah|Phoen|Pierr|Pierr|Oklah|
     ||HS   |SP   |SP   |SP   |SP   |HS   |-----|SP   |SD   |SD   |SD   |SP   |SP   |SP   |SO   |SO   |SP   |SP   |SP   |SO   |
Phoen||SaltL|Boise|Boise|Boise|Boise|SaltL|SaltL|-----|Denve|Denve|Santa|Boise|SaltL|SaltL|Austi|Austi|Austi|Boise|SaltL|Austi|
     ||SP   |BP   |BP   |BP   |BP   |SP   |SP   |-----|DP   |DP   |PS   |BP   |SP   |SP   |PA   |PA   |PA   |BP   |SP   |PA   |
Cheye||Denve|Denve|Denve|Denve|Denve|Denve|Denve|Denve|-----|Denve|Denve|Pierr|Pierr|Pierr|Denve|Denve|Denve|Pierr|Denve|Denve|
     ||CD   |CD   |CD   |CD   |CD   |CD   |CD   |CD   |-----|CD   |CD   |PC   |PC   |PC   |CD   |CD   |CD   |PC   |CD   |CD   |
Denve||SaltL|Phoen|Phoen|Phoen|Phoen|SaltL|SaltL|Phoen|Cheye|-----|Santa|Pierr|Pierr|Pierr|Oklah|Oklah|Phoen|Pierr|Oklah|Oklah|
     ||SD   |DP   |DP   |DP   |DP   |SD   |SD   |DP   |CD   |-----|DS   |PD   |PD   |PD   |DO   |DO   |DP   |PD   |DO   |DO   |
Santa||Denve|Phoen|Phoen|Phoen|Phoen|Denve|Denve|Phoen|Denve|Denve|-----|Denve|Denve|Denve|Austi|Austi|Austi|Denve|Austi|Austi|
     ||DS   |PS   |PS   |PS   |PS   |DS   |DS   |PS   |DS   |DS   |-----|DS   |DS   |DS   |SA   |SA   |SA   |DS   |SA   |SA   |
Bisma||Helen|Boise|Boise|Boise|Boise|Helen|Pierr|Boise|Pierr|Pierr|Pierr|-----|Pierr|Pierr|Pierr|Pierr|Boise|St.Pa|St.Pa|Pierr|
     ||HB   |BB   |BB   |BB   |BB   |HB   |BP   |BB   |BP   |BP   |BP   |-----|BP   |BP   |BP   |BP   |BB   |BS   |BS   |BP   |
Pierr||SaltL|SaltL|SaltL|SaltL|SaltL|SaltL|SaltL|SaltL|Cheye|Denve|Denve|Bisma|-----|Linco|Linco|SaltL|SaltL|Bisma|DesMo|SaltL|
     ||SP   |SP   |SP   |SP   |SP   |SP   |SP   |SP   |PC   |PD   |PD   |BP   |-----|PL   |PL   |SP   |SP   |BP   |PD   |SP   |
Linco||Pierr|Pierr|Pierr|Pierr|Pierr|Pierr|Pierr|Pierr|Pierr|Pierr|Pierr|Pierr|Pierr|-----|Topek|Topek|Pierr|Pierr|Pierr|Jeffe|
     ||PL   |PL   |PL   |PL   |PL   |PL   |PL   |PL   |PL   |PL   |PL   |PL   |PL   |-----|LT   |LT   |PL   |PL   |PL   |LJ   |
Topek||Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Linco|Linco|Linco|-----|Oklah|Oklah|Linco|Oklah|Oklah|
     ||TO   |TO   |TO   |TO   |TO   |TO   |TO   |TO   |TO   |TO   |TO   |LT   |LT   |LT   |-----|TO   |TO   |LT   |TO   |TO   |
Oklah||SaltL|Austi|Austi|Austi|Austi|SaltL|SaltL|Austi|Denve|Denve|Austi|SaltL|SaltL|Topek|Topek|-----|Austi|Jeffe|Jeffe|Jeffe|
     ||SO   |OA   |OA   |OA   |OA   |SO   |SO   |OA   |DO   |DO   |OA   |SO   |SO   |TO   |TO   |-----|OA   |OJ   |OJ   |OJ   |
Austi||Phoen|Phoen|Phoen|Phoen|Phoen|Phoen|Phoen|Phoen|Phoen|Phoen|Santa|Phoen|Phoen|Phoen|Oklah|Oklah|-----|Phoen|Oklah|Oklah|
     ||PA   |PA   |PA   |PA   |PA   |PA   |PA   |PA   |PA   |PA   |SA   |PA   |PA   |PA   |OA   |OA   |-----|PA   |OA   |OA   |
St.Pa||Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|Bisma|DesMo|Bisma|-----|DesMo|DesMo|
     ||BS   |BS   |BS   |BS   |BS   |BS   |BS   |BS   |BS   |BS   |BS   |BS   |BS   |BS   |BS   |SD   |BS   |-----|SD   |SD   |
DesMo||Pierr|St.Pa|St.Pa|St.Pa|St.Pa|Pierr|Pierr|Pierr|Sprin|Sprin|Sprin|St.Pa|Pierr|Pierr|Sprin|Sprin|Sprin|St.Pa|-----|Sprin|
     ||PD   |SD   |SD   |SD   |SD   |PD   |PD   |PD   |DS   |DS   |DS   |SD   |PD   |PD   |DS   |DS   |DS   |SD   |-----|DS   |
Jeffe||Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Oklah|Linco|Oklah|Oklah|Oklah|Sprin|Sprin|-----|
     ||OJ   |OJ   |OJ   |OJ   |OJ   |OJ   |OJ   |OJ   |OJ   |OJ   |OJ   |OJ   |OJ   |LJ   |OJ   |OJ   |OJ   |SJ   |SJ   |-----|
```

6. Aspirations

In creating the textual approach to solving the How Far Away problem we encountered many problems with displaying a large amount of information in a very limited area. Some of our implemented approaches were shrinking the font size to an almost illegible size, truncating the number of points to 50 and only displaying the first five letters in the name of the point and the link. However, we did start to implement a trimming strategy for making the textual approach more robust, but had very little time to implement and test it.

The basic idea behind the trimming strategy was, after inserting all the points with their links, to iterate through the points and remove all those with degree 1 and their outgoing links. We decided that our goal number would be 40 points, because this number could be easily shown on a letter size paper in a 6 or 8 pt font, thus making it more legible.

```
  public MapCity[] trimGraph(int limit, MapCity[] gcity){

        for (int i = 0; i < gcity.length; i++){

                if (gcity[i].nlinks <= limit){    // is the degree of the city the desired number?

                        int cnt = gcity[i].nlinks;
                        for(int j = 0; j < cnt; j++){
                                if(gcity[i].links[j].node1 == gcity[i])
                                        removeLink(gcity[i].links[j].node2, gcity[i].links[j]);
                                else
                                        removeLink(gcity[i].links[j].node1, gcity[i].links[j]);
                        }
                        gcity = removeCity(gcity, i);
                        i--;
                }
        }
```

We figured that we would loop and trim the 1 degree cities until the number of cities became 40 or less or we were not able to trim any more cities on the last iteration, thus allowing the algorithm to halt. However, if we had removed all the 1 degree cities and we had not yet reached 40, we decided to then take out the city with the smallest degree and then run the leaf clipping algorithm again under the same circumstances.

This idea seemed pretty robust for reducing the data set, much the same as some other groups found with identifying the hubs in the map and emphasizing them. After doing this trimming we would have a graph with the most important cities and then calculate the shortest paths between them. However, this idea also depends on the data set given by the user. For example if a really important city was inputted with a low degree along with many other points, there is a chance that the city would be removed. This draw back is a chance we would have been willing to ignore given more time to implement the project.

7. Acknowledgements

Andy came up with the wonderful idea of creating a textual representation of the data. We would like to thank all those who supported our idea as well as those who criticized. All your comments have been very helpful. Also, we would like to acknowledge group 1 for attempting a textual representation.

A large amount of our ideas came out of the first class discussion, but each idea was suggested by a team mate. Considering that we have a textual representation, and most class periods discussed the problems with graphical representation, we did not have many opportunities to seize ideas from class time. We did however brain storm in our group meeting times.

8. Tournament Analysis

The tournament results were in accordance with our expectations. For all pairs of cities whose source and destination were printed on the map, our chart not only found the path, but it always gave the user the shortest i.e. the best path. This is illustrated clearly in the results presented by Judge Kamra:

Map used	**Source**	**Destination**	**Ans1**	**Ans2**
Secret Map	Carnarvon	Orbost	Yes	Yes
US Capitals	Albany	Phoenix	Yes	Yes

Where Ans1 indicated if a path was found or not, and Ans2 indicates whether the path found was indeed the shortest path. In the other two maps, the cities were not printed on the chart due to the constraint on the size of the paper.

We observed similar results in the maps judged by Prof.Grinspun as well.

Map used	**Source**	**Destination**	**Result**
Secret Map	Coffsharbour	Wangaratta	Direct Link
US Capitals	Montpeller	Lincoln	Via Springfield

The comments we got included “Least visually appealing, but the most accurate”.

This trend continued with the other two judges as well, who were able to locate the shortest path accurately for the Australian map as well as that of the US capitals. This is completely in sync with our approach, as we truncated the number of cities that we printed out to be 50, due to lack of space. Other issues that came up were the small font size (again, space constraint), limiting the city and link names to 5 letters and list not being in alphabetical order (which we had taken care of initially, but looks like we overlooked the sorting in the last minute hustle!).

To put in a nutshell, our map performed the task accurately as long as the number of cities was limited to 50. In this scenario, the chart will ALWAYS provide you with the best (shortest) path between any pair of cities. It also eliminates ambiguity by providing the user with only the best path, and no other choices that might end up confusing him/her. On the other hand, the disadvantages were that not all the cities could be printed out, the font size was small, and it is not visually appealing from an aesthetic point of view. It was also very heartening to see that the text based approach had many supporters amongst the various groups who felt that it was the only map that ends up giving the accurate information you are seeking.

8. Conclusion

Text and graphical display of maps both have positives and negatives. We stayed with a textual display and focused on the algorithm for computing the shortest path between two nodes as well as sorting the paths for display and placing them in a path matrix that would be easier to read. We are pleased with the way our project turned out, but with more time we had ideas for improved clarity of our map. Our map does display the exact shortest path between two points, but does not work as efficiently with a large data set on a small piece of paper. With more time we would have liked to create an algorithm to trim the data set. We started to implement one, but ran out of time improving other aspects. Having to truncate the data set hurt us in the tournament, but we feel that our idea was very clear and precise otherwise.