Algorithmics 3 Assessed Exercise

Status and Implementation Reports

Mamas Nicolaou
1000007

November 12, 2012

Status report

The Dijkstra's and Backtrack best path algorithms work as expected. I have tested these algorithms using the sample data files provided from the Algorithmics 3 Page on Moodle.

For data6.txt - data80.txt the results for both algorithms are the same. They both return the same best path and the same shortest distance. However, in the case of data1000.txt the 2 algorithms return different paths. Below are the results from the 2 algorithms when input was data1000.txt:

Dijkstra's Algorithm:

```
Shortest distance from vertex 24 to vertex 152 is: 17
Shortest path: 24 922 810 957 963 371 837 152
Elapsed time: 2266 milliseconds
```

Backtrack Algorithm:

```
Shortest distance from vertex 24 to vertex 152 is 17
Shortest path: 24 582 964 837 152
Elapsed time: 37770 milliseconds
```

This is expected behaviour in the case where 2 paths to the destination exist with the same distance. To verify that this is not a defect I have created a test file that has 4 vertices and has 2 possible paths from vertex 0 to vertex 3.
Adjacency Matrix of test file:

```
4
0 2 0 6
0 0 2 0
0 0 0 2
0 0 0 0
0 3
```

The first path to destination was directly from vertex 0 to vertex 3 and had a distance of 6. The second path to destination was going through vertices 1, then through 2 and then finally arriving at vertex 3. The total distance for this route was again 6.

When I executed Dijkstra's algorithm against this file the result was:
Shortest distance from vertex 0 to vertex 3 is: 6
Shortest path: 0 3
Elapsed time: 74 milliseconds

Now, from Backtrack the output was different but not wrong:
Shortest distance from vertex 0 to vertex 3 is 6
Shortest path: 0 1 2 3
Elapsed time: 113 milliseconds

This proves that the 2 algorithms might point the user to a different path when the distance is the same and that this is not a programming defect.

Implementation report

(a) I used the solution of the Laboratory exercise to create the graph that will be used for the Dijkstra's algorithm. I made a few changes on that code so that it also creates the adjacency tables for each of the vertices. An adjacent table will hold information such as to which vertex a node pointing to (node.getNumber()) and the distance that this vertex node has from the parent vertex.

A Dijkstra's algorithm requires a table of distances from the starting point to each of the vertices. So creating an array of size (number of vertices) was the obvious first step. I, then, created two linked lists which would hold the visited and unvisited vertices during execution. Then for each each unvisited vertex, I would add the vertex to the visited vertices list and then inspect its adjacency list. For adjacent vertex the algorithm checks the distance that it has recorded in the distances table and if the distance of the current path is less than the already discovered distance it would set the distance of the vertex to be the current distance from start point to here and set the predecessor of the current vertex to be the vertex of which the adjacency table is being inspected.

Repeating this procedure for all vertices results in having a distances table that has the best possible distance to each of the vertices from starting point. Apparently the distance recorded for the destination vertex is the distance of the best path to it. If the distance is still set to -1 it means that no path was found.

The algorithm will produce appropriate output in each case stating the best path that was found or informing the user that there was no available path to destination.

(b) For creating the Graph I copied the code from Dijkstra's main class. I created a new class called Path that would hold information for the current path and the best path found. This includes the distance of each path and a linked list containing the vertices which form the route to destination.

I followed the given pseudo code to create the code for the Backtrack search. The algorithm calls a recursive function called tryNextCandidate which takes the path position that the candidate will need to fill in, the current path, best path and the destination vertex. The current path initianally only contains the starting vertex and has its distance set to zero. The best path on the other hand has its distance set to Integer.MAX_VALUE. Every time a candidate path reaches the destination vertex, its distance is compared to the distance of best path found until now and if this is shorter the best path is updated to be the current path. This procedure is repeated until no other vertices can be reached. Each time a vertex is added to the current path, it is added on the tail of the current path linked list so then we only call currentPath.remove() to remove it from the tail and simply deduct the distance to the removed vertex from the total distance of the currentPath.

The algorithm then checks to see if the vertices in best path are more than 1 (the starting vertex) and if so reports the best path to the user and the distance to the destination vertex. If the number of vertices in best path are less than 2 it reports that no possible path could be found.

Empirical results

From the running times results it is obvious that even if the backtrack search algorithm is fast enough to compete with Dijkstra's algorithm at small inputs their difference changes dramatically when the number of inputs is increased. At 6 vertices the Backtrack algorithm finished even faster than Dijkstra's, however at 1000 vertices their difference in running time was more than 25000 milliseconds.

Results for Data6.txt
Dijkstra:
Shortest distance from vertex 2 to vertex 5 is: 11
Shortest path: 2 1 0 5
Elapsed time: 63 milliseconds
Backtrack:
Shortest distance from vertex 2 to vertex 5 is 11
Shortest path: 2 1 0 5
Elapsed time: 62 milliseconds

Results for Data20.txt

Dijkstra:
Shortest distance from vertex 3 to vertex 4 is: 1199
Shortest path: 3 0 4
Elapsed time: 140 milliseconds
Backtrack:
Shortest distance from vertex 3 to vertex 4 is 1199
Shortest path: 3 0 4
Elapsed time: 141 milliseconds

Results for Data40.txt
Dijkstra:
Shortest distance from vertex 3 to vertex 4 is: 1157
Shortest path: 3 36 4
Elapsed time: 171 milliseconds
Backtrack:
Shortest distance from vertex 3 to vertex 4 is 1157
Shortest path: 3 36 4
Elapsed time: 358 milliseconds

Results for Data60.txt
Dijkstra:
Shortest distance from vertex 3 to vertex 4 is: 1152
Shortest path: 3 49 4
Elapsed time: 374 milliseconds
Backtrack:
Shortest distance from vertex 3 to vertex 4 is 1152
Shortest path: 3 49 4
Elapsed time: 889 milliseconds

Results for Data80.txt
Dijkstra:
Shortest distance from vertex 4 to vertex 3 is: 1152
Shortest path: 4 49 3
Elapsed time: 452 milliseconds
Backtrack:
Shortest distance from vertex 4 to vertex 3 is 1152
Shortest path: 4 49 3
Elapsed time: 12652 milliseconds

Results for Data80.txt
Dijkstra:
Shortest distance from vertex 24 to vertex 152 is: 17
Shortest path: 24 922 810 957 963 371 837 152
Elapsed time: 2184 milliseconds
Backtrack:
Shortest distance from vertex 24 to vertex 152 is 17

```
Shortest path: 24 582 964 837 152
Elapsed time: 38349 milliseconds
```