Algorithmics I - Assessed Exercise

Status and Implementation Reports

Sam George
2261522g

November 12, 2018

Status report

Both my Dijkstras and Backtracking algorithm compile and run with the correct traversal and output. The algorithms run in similar times for small files but as expected the backtracking algorithm runs considerably slower than the dijkstras for large graphs. The output distances and path taken are correct for both algorithms and run in a reasonable time.

Implementation report

Dijkstra's Algorithm

To create a working dijkstras algorithm I first initialise the source vertex and its connected distances. Then I find the minimum distance to a vertex which hasn't been visited and visit that vertex. From that I update all vertices' distances as long as they havent been added to the list of visited nodes. If the new distance is shorter than the current distance then I update the vertex's distance and update it's predecessor. My Dijkstra's algorithm implementation uses a list of vertices. I chose a list because I need to index it at random points which is more efficient than traversing over like you would do with a linked list. The list is a fixed size (num_vertices) after it has been populated which means we don't need to consider the efficiency of adding and removing elements. I also use a linked list to store the current set of nodes visited (vList). I use a linked list because I will never be removing elements, I will only add elements to the end and I can use Java's .size() function to check if every node has been visited. Because we want to find the minimum distance of nodes we haven't visited it would be inefficient to search the list of all distances and check if it is in the vList. To counter this I use an ArrayList of integers for the indexes of vertices which haven't been visited and then perform a simple O(n) find minimum search over that. The reason I chose an ArrayList is because it won't be a fixed size and I may need to remove elements half way through the list. Which is less efficient for Linked Lists. Another hurdle was storing the route taken. Either way to print the path taken you must traverse over the method chosen to store them. So my main concern was memory efficiency, to save as much memory instead of making a list of the vertices taken in the path I just updates the vertex class to hold a predecessor. This means to traverse it I loop from the destination back the way until I reach the source.

Backtrack Search

I implemented the backtrack with an ArrayList for current path which parameter n points to the latest element in. I then loop over the last elements adjacent vertices recursively calling and adding until we get to the destination. Then it will backtrack to the last node and traverse other routes to the destination. I improved the run time on it significantly by filtering out vertex steps

based on the current distance. So for example if the current path distance is longer than the best path distance we can skip this current vertex in the traversal. I also did the same filtering but on a vertex level so if the current path distance is greater than the best path distance to a vertex then we can skip this traversal. Also to prevent the backtrack from entering an infinite loop I had to make a visited attribute for the vertices. I use an ArrayList of Vertices for the current path because it was easier to index remove and add vertices to. For remembering the best path I did the same as my Dijkstra implementation. This is because to print out the path is already slow and whether you store it in an array or by vertex predecessor you have to traverse to print them. So the I chose the vertex predecessor method to save memory. The other variables are just integers used for comparisons so they decrease run time rather than increase it.

Empirical results

Data6_1.txt

Dijkstras:
The shortest path from vertex 2 to vertex 5 is 11
Shortest Path: 5 0 1 2
Elapsed time: 103 milliseconds

Backtrack:
The shortest path between 2 and 5 is 11
Shortest path: 5 0 1 2
Elapsed time: 109 milliseconds

Data6_2.txt

Dijkstras:
There is no path between vertex 2 and 2
Elapsed time: 76 milliseconds

Backtrack:
The is no path between vertices 2 and 5
Elapsed time: 82 milliseconds

Data20.txt

Dijkstras:
The shortest path from vertex 3 to vertex 4 is 1199
Shortest Path: 4 0 3
Elapsed time: 122 milliseconds

Backtrack:
The shortest path between 3 and 4 is 1199
Shortest path: 4 0 3
Elapsed time: 114 milliseconds

Data40.txt

Dijkstras:
The shortest path from vertex 3 to vertex 4 is 1157
Shortest Path: 4 36 3
Elapsed time: 133 milliseconds

Backtrack:
The shortest path between 3 and 4 is 1157
Shortest path: 4 36 3

Elapsed time: 131 milliseconds
Data60.txt

Dijkstras:
The shortest path from vertex 3 to vertex 4 is 1152
Shortest Path: 4 49 3
Elapsed time: 151 milliseconds

Backtrack:
The shortest path between 3 and 4 is 1152
Shortest path: 4 49 3
Elapsed time: 134 milliseconds
Data80.txt

Dijkstras:
The shortest path from vertex 4 to vertex 3 is 1152
Shortest Path: 3 49 4
Elapsed time: 166 milliseconds

Backtrack:
The shortest path between 4 and 3 is 1152
Shortest path: 3 49 4
Elapsed time: 172 milliseconds
Data1000.txt

Dijkstras:
The shortest path from vertex 24 to vertex 152 is 17
Shortest Path: 152 837 371 963 957 810 922 24
Elapsed time: 768 milliseconds

Backtrack:
The shortest path between 24 and 152 is 17
Shortest path: 152 837 371 963 957 810 922 24
Elapsed time: 8758 milliseconds