Algorithmics I - Assessed Exercise

Status and Implementation Reports

Jawad Shahid Mahmoud
2253483m

November 11, 2018

Status report

Both my programs give the same results for each of the input files.

I also checked the outputs from my files by inputting the data files into a web graphing application which provided functionality of finding shortest path between two vertices using the dijkstra's algorithm. Graphing Application : http://graphonline.ru/en/

Couldn't insert data1000.txt into the web app, but as both my programs give the same output, I think this verifies the output from this file as well.

Errors:

Compilation: No errors during compilation.
Runtime: No errors during runtime.

Implementation report

The graph is implemented by taking inputs from the scanner. The scanner uses the first line of the input file to set the number of vertices present and the last line to extract the first and last vertices for finding shortest path between.

AdjListNode: Edited to contain the weight of the edge between two vertices alongside the index of the vertex.

Dijkstra's Algorithm

Data Structures:

- HashSet is used for storing the visited and non-visited vertices. The HashSet is significantly faster for all its operations than other implementations of a Set. It provides the basic functionality required of pointing to a vertex directly and removing it rather than having to use indices and vertices can just as easily be added to the HashSet. This gives us $O(1)$ time for all necessary operations in the implementation. The ordering is not important for our implementation and so the HashSet is ideal.
- HashMap is used for storing the distances and predecessors of vertices visited during the execution of the algorithm. HashMap gives us the capability of storing key-value pairs which is essential for storing distances and predecessors. HashMap provides $O(1)$ time on average for most of the simple operations such as adding and removing, and searching for values takes $O(n)$ time. We don't require the items to be particularly sorted hence HashMap is better than TreeMap.
- LinkedList is used to store flow path (shortest path extracted from the predecessors HashMap). This is done to allow getting the order in which vertices were visited by checking one predecessor after another. The LinkedList can then be reversed in $O(n)$ time. Using a Stack instead would not result in any significant improvement as we would require another loop to iterate over the items in the stack and output them.

Processes:

- Starts off with putting the start vertex into distances with value of 0. And then gives all other vertices of the graph a distance of max possible Integer value. Then, initializes the predecessors map, giving each vertex the start vertex as its predecessor (except of start vertex). Then adds all vertices to the non-visited set.
- Until all vertices have been visited, the program does the following:
  - Get the vertex which has the least distance:
    Iterate through the distances and return the vertex with the lowest distance value
  - Update visited and nonvisited bertices.
  - Relaxation:
    Updates the distances for all the unvisited vertices that are adjacent to the current vertex. If a new minimum distance is found for a vertex, its predecessor is updated to the vertex the new distance is calculated with.
- Checks if a path has been found, returns the path and distance from start to end vertex if found, otherwise prints that path does not exist between the two vectors.

Backtrack Search

New Classes:

- BTPath:
  - Containes two fields, a LinkedList for storing vertices in the order they are added and an integer storing the distance of path. The vertices in the LinkedList are a path from the start vertex to the end vertex, and everytime a vertex is added it also adds to the distance of the BTPath object and vice versa for removing vertices.

Data Structures:

- LinkedList is used for storing vertices in the BTPath object because it adds vertices to the end of the list, most of the operations are done in $O(1)$ time. The list allows us to access the last vertex added easily and allows iteration over its contents easily. The LinkedList can also be easily cloned, which is essential for cloning the currentpath to the bestpath if a new minimum distance is found.

Processes:

- Starts off by creating two BTPath objects, one initialised with the starting vertex and distance 0, whereas the other is initialised with max distance and the path linkedlist is null.
- Until all vertices have been visited and their adjacent vertices checked, the program does the following:
  - Each vertex that is adjacent to the last vertex added to the currentpath is checked whether it is visited, if unvisited, the vertex is added to path along with the weight of the edge
  - If after this the distance of the currentpath is less than bestpath and the vertex added is equal to the ending vertex, the bestpath is updated with the values of the currentpath. Otherwise, the backtrack search is called again, which will now process the recently added vertex.
  - Before the loop repeats, the recent vertex is removed from the current path.
- Checks if a path has been found, returns the path and distance from start to end vertex if found, otherwise prints that path does not exist between the two vectors.

Empirical results

All outputs generated on Linux (Elementary OS)

Dijkstra's Algorithm

- **data6_1.txt**
  Shortest distance from vertex 2 to 5 is 11
  2 1 0 5
  Elapsed time: 22 milliseconds
- **data6_2.txt**
  Path does not exist.

  Elapsed time: 23 milliseconds
- **data20.txt**
  Shortest distance from vertex 3 to 4 is 1199
  3 0 4
  Elapsed time: 30 milliseconds
- **data40.txt**
  Shortest distance from vertex 3 to 4 is 1157
  3 36 4
  Elapsed time: 33 milliseconds
- **data60.txt**
  Shortest distance from vertex 3 to 4 is 1152
  3 49 4
  Elapsed time: 41 milliseconds
- **data80.txt**
  Shortest distance from vertex 4 to 3 is 1152
  4 49 3
  Elapsed time: 50 milliseconds
- **data1000.txt**
  Shortest distance from vertex 24 to 152 is 17
  24 582 964 837 152
  Elapsed time: 588 milliseconds

Backtrack Search

- **data6_1.txt**
  Shortest distance from vertex 2 to 5 is 11
  2 1 0 5
  Elapsed time: 20 milliseconds
- **data6_2.txt**
  Path does not exist.

  Elapsed time: 20 milliseconds
- **data20.txt**
  Shortest distance from vertex 3 to 4 is 1199
  3 0 4
  Elapsed time: 25 milliseconds
- **data40.txt**
  Shortest distance from vertex 3 to 4 is 1157
  3 36 4
  Elapsed time: 50 milliseconds
- **data60.txt**
  Shortest distance from vertex 3 to 4 is 1152
  3 49 4
  Elapsed time: 243 milliseconds
- **data80.txt**
  Shortest distance from vertex 4 to 3 is 1152
  4 49 3
  Elapsed time: 7099 milliseconds
- **data1000.txt**
  Shortest distance from vertex 24 to 152 is 17
  24 582 964 837 152
  Elapsed time: 98538 milliseconds