Network Software Modelling
Assignment 1

Dijkstra's Algorithm

Student Name (Student Number)	Eoin Carroll (16202781)
Module Code	MIS40550
Module Title	Network Software Modelling
Assessment Title	Assignment 1
Module Co-ordinator	Dr James McDermott
Revision	2
Date Submitted	12/03/2017

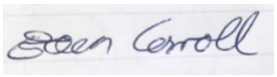
Plagiarism

The unacknowledged inclusion of another person's writings or ideas or works, in any formally presented work (including essays, examinations, projects, laboratory reports or presentations). The penalties associated with plagiarism designed to impose sanctions that reflect the seriousness of University's commitment to academic integrity. Ensure that you have read the University's Briefing for Students on Academic Integrity and Plagiarism and the UCD Plagiarism Statement, Plagiarism Policy and Procedures, (http://www.ucd.ie/registrar/).

Declaration of Authorship

I declare that all material in this assessment is my own work except where there is clear acknowledgement and appropriate reference to the work of others.

Signature

Signed:  Date: 12/03/2017

Revision History

Revision Number	Date Released	Reason/Notes
0	05/03/2017	Initial Release
1	08/03/2017	Code improvements More Testing Added Improved report Note: Code revision matches report revision number
2	12/03/2017	Theoretical run time answer clarification Q1 Numerous further wording improvements No code changes Note: Code revision matches report revision number

Contents

Question 1

Dijkstra's algorithm takes a network, a start node and a goal node as inputs. We create a set S of all nodes in the network and a set P with only the initial node as an element. We calculate the cost of moving to each connected node from the initial node. We add only the node associated with the lowest cost to set P and remove that node from S. We repeat this using all nodes in P as starting points and recording the cost to get to each node. The algorithm stops as soon as the goal node has been added to P and the path is returned to the user.

The time complexity of Dijkstra's algorithm with a list data structure is O(en) as we iterate through the list to find a next smallest cost move where e represents the number of edges and n is number of vertices/nodes. For a connected graph, we would see roughly $O(n^2)$. This can be reduced to O(e*log(n)) with a heap data structure where we can easily access the next priority move.

Question 2

The bidirectional Dijkstra algorithm can be summarised as running two original Dijkstra's algorithms simultaneously (We have described the original Dijkstra's algorithm above). The first instance is started at the initial node as normal and the second instance is started from the goal node. The second instance is instructed to work backwards through the network, treating any directed edges in reverse. The stopping condition of the bidirectional Dijkstra algorithm differs by finishing when any node has been added to the set P of both algorithms. At this point, we have found a shortest path from the initial starting node to the goal node.

Question 3

In the following diagram, we can see Dijkstra's algorithm finding the shortest path for both methods. The orange path is shortest and we can see the point where both instances meet in the bidirectional implementation. S is the start node and G is the goal node.

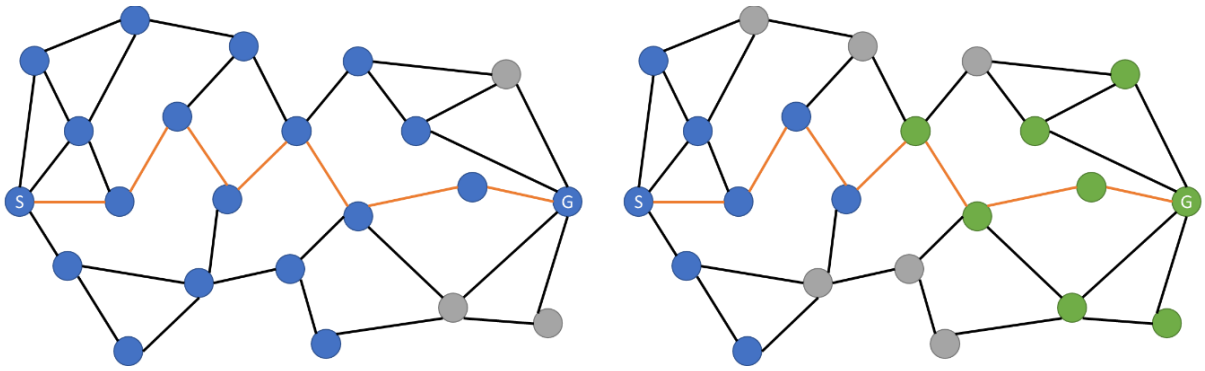


Figure 1: Original Dijkstra's algorithm (Left) & bidirectional Dijkstra algorithm (Right)

Question 4

The time complexity of a bidirectional Dijkstra's algorithm with heap data structure is still in the range O(e*log(n)). This is the same worst case run time as the original Dijkstra's algorithm however in practice we are likely to see efficiency gains due to the bidirectional algorithm's ability to omit exploration of many nodes. The two algorithm instances are likely to create a shortest path in less iterations than one instance by itself. An example of omitted nodes is shown in grey in Figure 1.

Question 5

Both Dijkstra's algorithm and the bidirectional variant have been implemented using a priority queue data structures for efficiency. A python file has been created, tested and included in this submission. Please read the comments in the appendix and comments in the file to understand the usage.

Question 6

Run-time behaviour of both algorithms was tested on randomly generated graphs of varying sizes to demonstrate scaling behaviour. The algorithm was repeated for 5 different random graphs for each n. An edge connected roughly 10% of nodes and the start and goal nodes were never directly connected. A simplified table and a detailed chart presenting results have been included. A full output from the program is presented in the appendix.

n = Nodes in Graph	dijkstra_priority (seconds)	bi_dijkstra_priority (seconds)
n = 500	1.15786	0.79324
n = 1000	3.65100	1.66569
n = 5000	225.14007	64.13306
n = 10000	514.52582	210.11141

Table 1: Run-time Behaviour Table

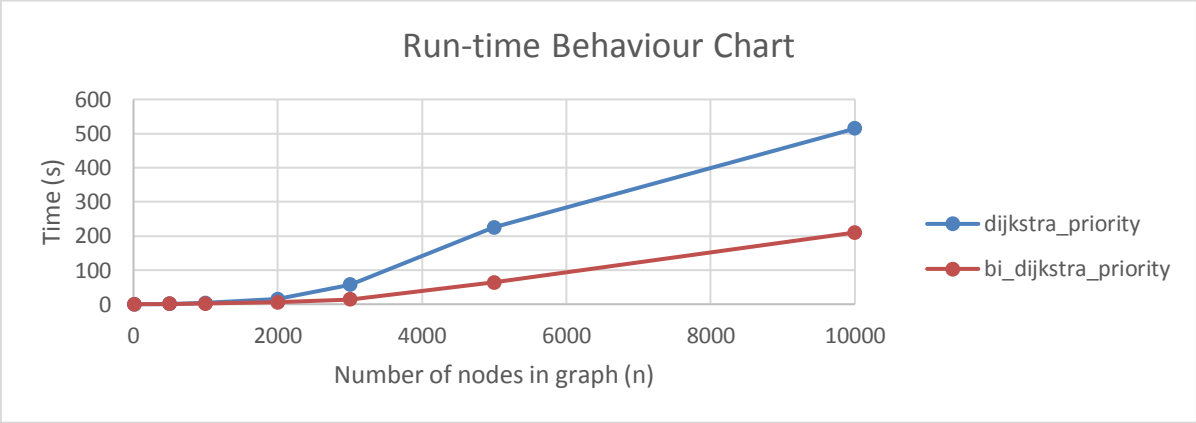


Figure 2: Run-time Behaviour Chart

Question 7

The run time of Dijkstra's algorithms as implemented should be O(e*log(n)) however this value is the worst-case performance and will rarely be seen in practice. Our run time output does indeed resemble a logarithmic result for both algorithms as we had anticipated. For small graphs, both algorithms have resulted in similar run times. As the number of nodes and edges to be explored increased, we began to see a clear difference in efficiency. The bi-directional Dijkstra algorithm results were available in less than half of the time required for Dijkstra which is in a significant time and resource saving. An original Dijkstra algorithm with lists was tested and the results were somewhat surprising. The list algorithm ran faster than both heap algorithms however this was only tested up to 2000 nodes. This indicates that improvements could have been made to both heap algorithms to increase efficiency.

Conclusion

With this basic implementation of both the Dijkstra and the Bi-Directional Dijkstra algorithm, we have explored the run time differences between two effective shortest path methods.

References/Resources

1. MIS40550 Network Software Modelling Course notes, labs and sample code.
2. Erik Demaine, and Srinivas Devadas. 6.006 Introduction to Algorithms. Fall 2011. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

Appendix

1. Please see python code "MIS40550_Ass1_Eoin_Carroll_16202781.py" uploaded to Blackboard.

2. Program Output Sample:

   Dijkstra results averaged over 5 repetitions for each n with density = 0.1

   With n = 500
   dijkstra_priority time: 1.1578661918640136
   bi_dijkstra_priority time: 0.7932453632354737

   With n = 1000
   dijkstra_priority time: 3.6510088443756104
   bi_dijkstra_priority time: 1.6656953334808349

   With n = 2000
   dijkstra_priority time: 14.608035564422607
   bi_dijkstra_priority time: 5.927138948440552

   With n = 3000
   dijkstra_priority time: 57.29047679901123
   bi_dijkstra_priority time: 13.530573892593384

   With n = 5000
   dijkstra_priority time: 225.1400773525238
   bi_dijkstra_priority time: 64.13306818008422

   With n = 10000
   dijkstra_priority time: 514.5258292198181
   bi_dijkstra_priority time: 210.1114176273346

3. This program differs to the programs in labs whereby the shortest path is not returned by default. Shortest path and cost can be returned by commenting out the last section in each algorithm and inserting a print().

4. Code is currently set to run for random graphs with n nodes where n = (500,1000,2000,3000,5000,10000) and 10% density. It will repeat this 10 times and report the average. Expected run time is in the range of 3 hours. The user can change these input values to run smaller tests. Alternatively, import the python file and use dijkstra_priority(G, r, t) and bi_dijkstra_priority(G, r, t) to find the shortest paths in your favourite graphs.

5. The python file includes:
   - Main() Program to run dijkstra for n random networks
   - dijkstra_list(): Basic dijkstra algorithm using a list data structure (Reference only - not used)
   - dijkstra_priority(): Basic dijkstra algorithm using a priority queue data structure
   - bi_dijkstra_priority(): Bbidirectional Dijkstra algorithm using a priority queue data structure
   - class PriorityQueue: Priority heap data structure with peek feature