Nanyang Technological University

CZ2001 Example Class 4 Report

Application of BFS to Flight Scheduling

Group Members:
Cheng Hui Thong
Laurensia Anjani
Takafumi Takemoto
Teo Zhe Wei
Ufuk Civan Atbas

1. Flight Schedule Graph

Figure 1 shows the graph that represents airline flights between cities in the world, which our group uses in our analysis of BFS algorithm.

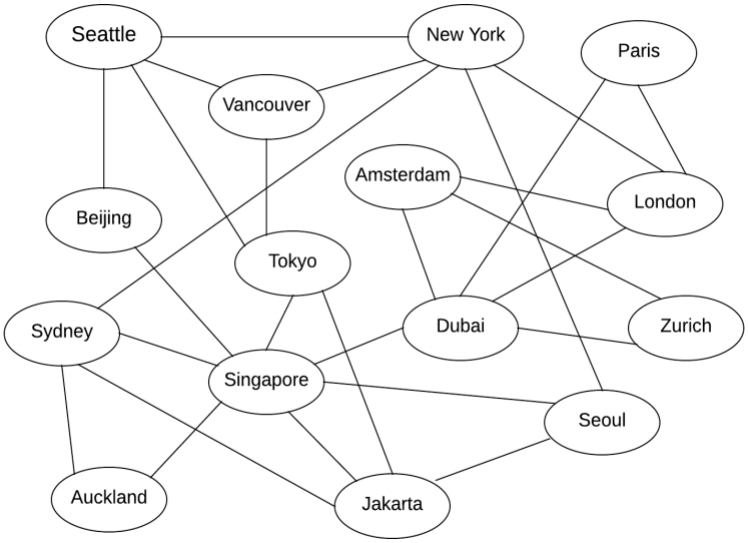
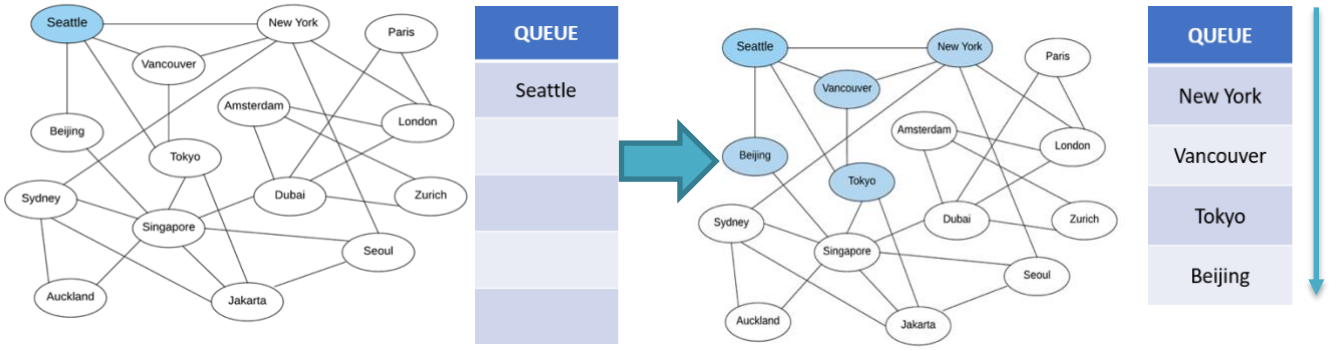


Figure 1. Undirected graph representing airline flights between cities

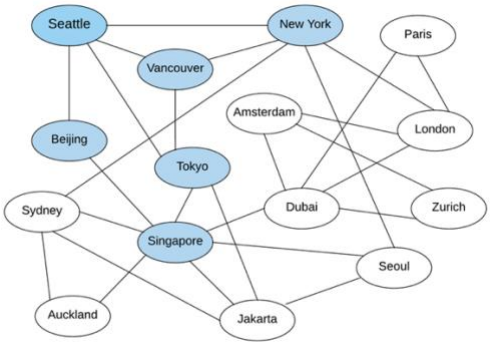
2. Breadth-first Search (BFS) Algorithm

BFS is a traversing algorithm that starts from the source node and traverses the adjacent nodes layerwise. It will first move horizontally and visit all the nodes of the current layer, before moving to the next layer. The algorithm implements queue, which can be illustrated using the following scenario of finding the shortest flight path from Seattle to Singapore.



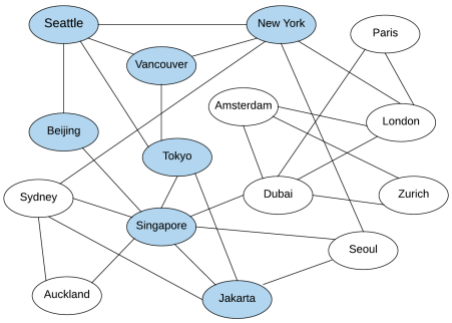
Given a graph, the source node, Seattle, is enqueued and marked as visited.

After Seattle node is dequeued, the child nodes of Seattle node will then be enqueued and marked as visited.



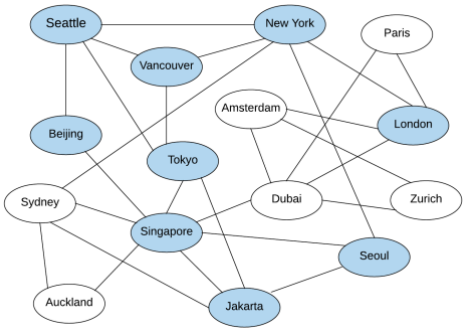
QUEUE
Singapore
New York
Vancouver
Tokyo

When Beijing is dequeued, Singapore node is enqueued and marked.



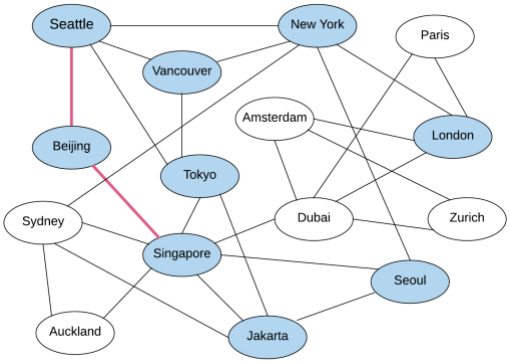
QUEUE
Jakarta
Singapore
New York
Vancouver

Tokyo is dequeued and Jakarta is enqueued and marked.



QUEUE
London
Seoul
Jakarta
Singapore

Vancouver is dequeued, but all the child nodes have been marked. So New York is dequeued. Seoul and London are enqueued and marked.



Singapore is dequeued and it matches the destination node, so printShortestPath is called.

The shortest flight path from Seattle to Singapore is then found: Seattle – Beijing – Singapore.

3. Performance Statistics

Our analysis of running times will be based on the average CPU time to find the shortest path, when there are minimum, maximum, and intermediate number of non-stop flights. For each of the three categories, we also change the size of the graph, **n**, to be 100, 1000 and 2000.

3.1 CPU time statistics (n = 100, 1000, 2000 | Minimum number of non-stop flights)

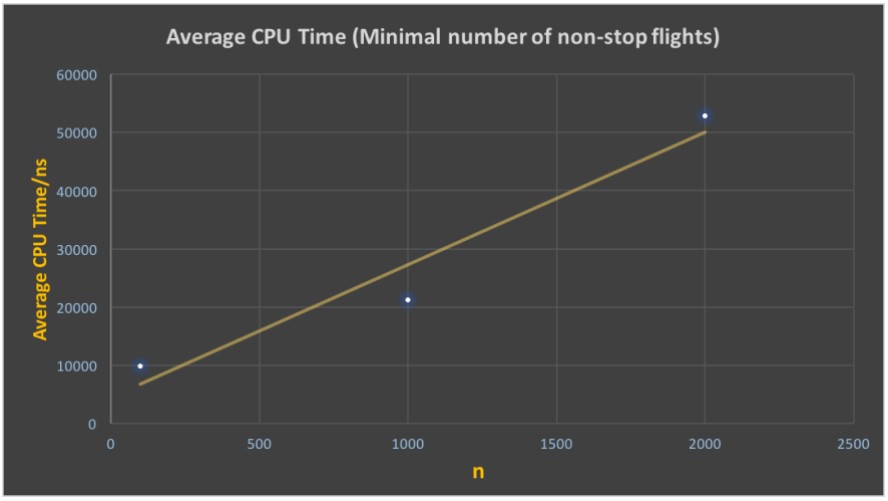


Figure 2. The linear best-fit line of the average CPU time to find the shortest flight route for minimal number of non-stop flights.

3.1 CPU time statistics (n = 100, 1000, 2000 | Maximum number of non-stop flights)

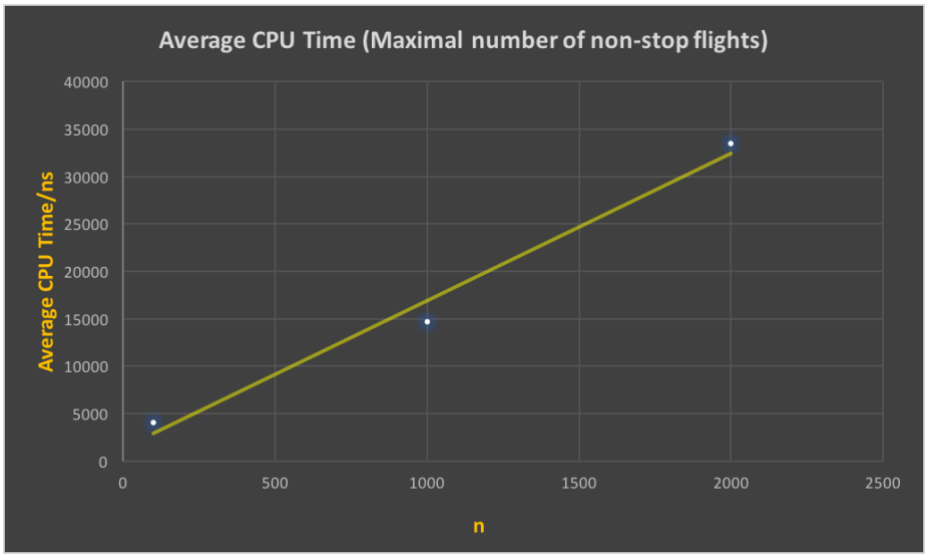


Figure 3. The linear best-fit line of the average CPU time to find the shortest flight route for maximum number of non-stop flights.

3.3 CPU time statistics (n = 100, 1000, 2000 | Intermediate number of non-stop flights)

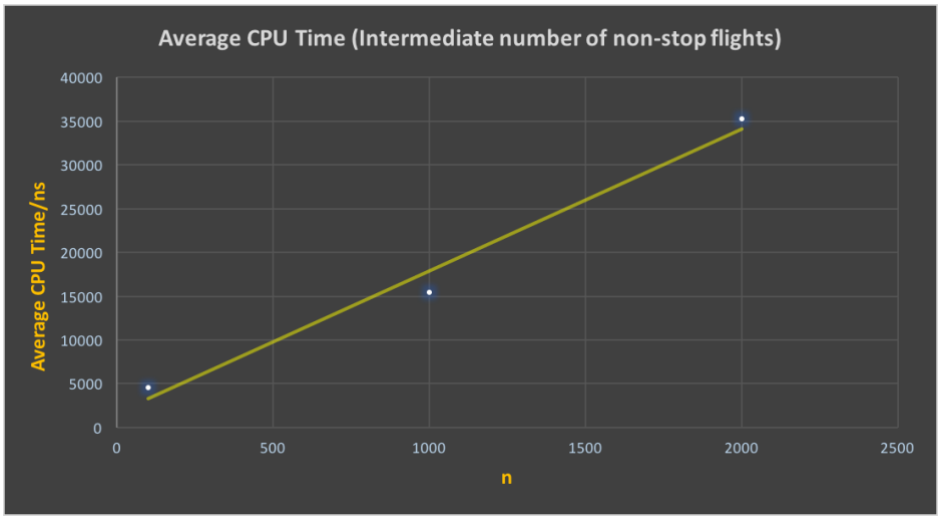


Figure 4. The linear best-fit line of the average CPU time to find the shortest flight route for intermediate number of non-stop flights.

3.4 Analysis on Running Times

Through observation on Figures 2, 3 and 4, we can see that the running time of BFS algorithm increases linearly with the increase in number of cities, which in this case is represented by the increase in graph size. It is because larger graph size would mean greater number of nodes to traverse, which results in longer running time.

Table 1. Average CPU time statistics for different graph sizes and number of non-stop flights

n	Average CPU Time (Min non-stop)	Average CPU Time (Max non-stop)	Average CPU Time (Intermediate non-stop)
100	9885	4018	4540
1000	11415	10690	10863
2000	31601	18753	19868

Referring to Table 1, we can also conclude that the average CPU time to find the shortest flight route is longest for graph that has the least number of non-stop flights, while it is fastest for graph that has the maximum number of non-stop flights. The time complexity of BFS using adjacency list, is O(|V| + |E|).

4. Applicability of Depth-first Search (DFS) in Shortest Path Finder

Depth-first search (DFS) algorithm cannot be used in place of BFS in finding the minimum number of stops between two cities.

DFS is a recursive algorithm that visits the ‘deeper’ nodes until it reaches a leave node, before going back to traverse more nodes through backtracking. Whereas, BFS visits adjacent nodes level by level. The difference in algorithms between the two results in one being applicable to find the shortest path, and one is not.

In this application problem, one city might have multiple routes to another city, which some might not be the shortest path. Using DFS algorithm, the search terminates once it finds the node with the desired destination, which may not result in the shortest possible route. Thus, DFS cannot be used to replace BFS in this case.

5. Conclusion

In conclusion, the BFS algorithm can be implemented in real-life application, such as in finding the shortest flight path between two cities. We have also observed that:

- The larger the size of the graphs, the longer it takes to find the shortest path.
- The higher the number of non-stop flights available, the shorter the time taken to find the shortest path.

Moreover, while DFS algorithm can be used in finding a path between the source and destination cities, the algorithm might not give the shortest path possible. Thus, for finding the shortest path, only BFS can be used.