

# 

# 

# CZ2001 Algorithm

Example Class 4 Report

|  |  |
| --- | --- |
| Members - SE1 (GROUP 2) | |
| U1620986A | Lee Yi Zhuo |
| U1721978B | Ngo Jun Hao Jason |
| U1722308D | Thomas Stephen Felix |
| U1722841L | Tan Yong Heng Kenrick |
| N1804770J | Yung-Hsueh Lee |

Data

The data set of cities used for this lab is taken from a repository in GitHub as shown in Reference.

Implementation of Graph

In our implementation, we will be using adjacency list to represent the graph (undirected, non-weighted). In this class, it will hold all the function to create and modify the graph that will be used by Graph generation class.

Methods:

1. Connect: Implementing the undirected edge between 2 cities
2. Density: 2\* number of non-stop flights (undirected graph) / the size of the adjacency list
   1. num of non-stop flights = density \* max possible number of non-stop flights
      1. max possible number of non-stop flights = size\*(size-1)/2

(3) printGraph - Although the the Graph is implemented as an adjacency list, we use print the graph out as an adjacency matrix as it allows us to easily see which cities are connected.

Implementation of Graph Generation

In this implementation, we will be using size, density and number of non-stop flight to generate random graph by randomly generating linked list to each city.

Methods:

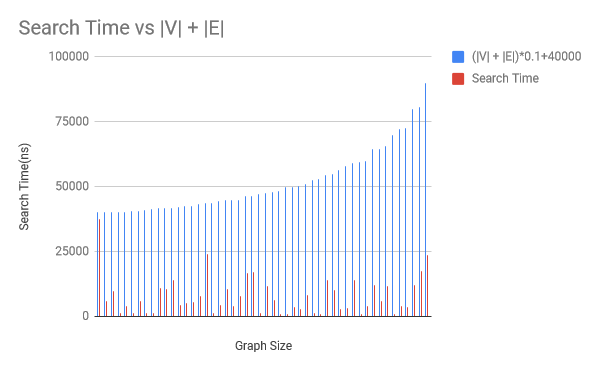
1. buildNonStopFlights (building edges between cities by using adjacency matrix)
   1. edges are built on top of pre-existing graph
      1. only cities that are not connected should be selected
      2. selection of cities is done using adjacency matrix
         1. search time of O(1)

Implementation of Searching Algorithms

* Our searching algorithms are implemented in the Searcher class
  + the Searcher class make use of a linked list, L
    - L is used as a queue in BFS
    - L is used as a stack in DFS
* Both BFS and DFS have the same code structure; they:
  + visit the source
  + place the source in L
  + while L is not empty (ie graph is not fully traversed)
    - remove the the first City in L
    - visit the first City’s neighbouring Cities if they are not already visited
    - add the neighbouring Cities to L
* Difference in BFS and DFS
  + upon visiting a neighbouring City that is the destination
    - BFS: return path to destination as the shortest route
    - DFS: compare new path found to previously known shortest route;
      * if the newly-found path is shorter than current shortest route, or there is no shortest route currently, the new path found is stored as the shortest route
  + for adding of neighbouring Cities to L
    - BFS: add neighbouring Cities to back of L (i.e. L is used as a queue)
    - DFS:
      * if neighbouring Cities are already in L, remove them from L first
      * add neighbouring Cities to front of L (i.e. L is used as a stack)
  + at the end of the function
    - BFS: returns null since no path can be found
    - DFS: returns the shortest route, or null if no path can be found

Analysis of running times

**based on number of cities (size of graph/ |V|) and number of non-stop flights (|E|)**



Based on the graph, search time = O(0.1 (|V| +|E|) + 40000) = O(|V|+|E|).

Conclusion

DFS algorithm cannot be used in place of BFS as it does not guarantee that if node 1 is visited before another, node 2, starting from a source vertex, then node 1 is closer to the source than node 2. This is also reflected in the backtracking nature of DFS in which it visits the “deeper” nodes further away from the source nodes first; going as far as it possibly can before returning to the unvisited adjacent nodes.

On the other hand, BFS visits nodes in increasing order of their distance from the source; visiting all nodes at the same “level” before proceeding to the next. Therefore it would be more practical to search wide instead of deep and this can be seen empirically based on a better overall average search time by using BFS.

Nonetheless, DFS can also be used in place of BFS, but that would require the program to find all possible ways to reach the destination node from source node and updating the shortest path with each new discovery; however, this comes with high complexity and it is simply not recommended in this instance.

Reference

<https://github.com/datasets/world-cities>