Written exercises

1. What is the big-Oh space complexity of an adjacency list? Justify your answer.

adjacency list is a linked data structure, O(V+E) worse case need to traverse the whole graph by all vertices and edges.

2. What is the big-Oh space complexity of an adjacency matrix? Justify your answer.

O(V^2) since it's a 2D array storing the connectivity of all nodes where V is the number of vertices

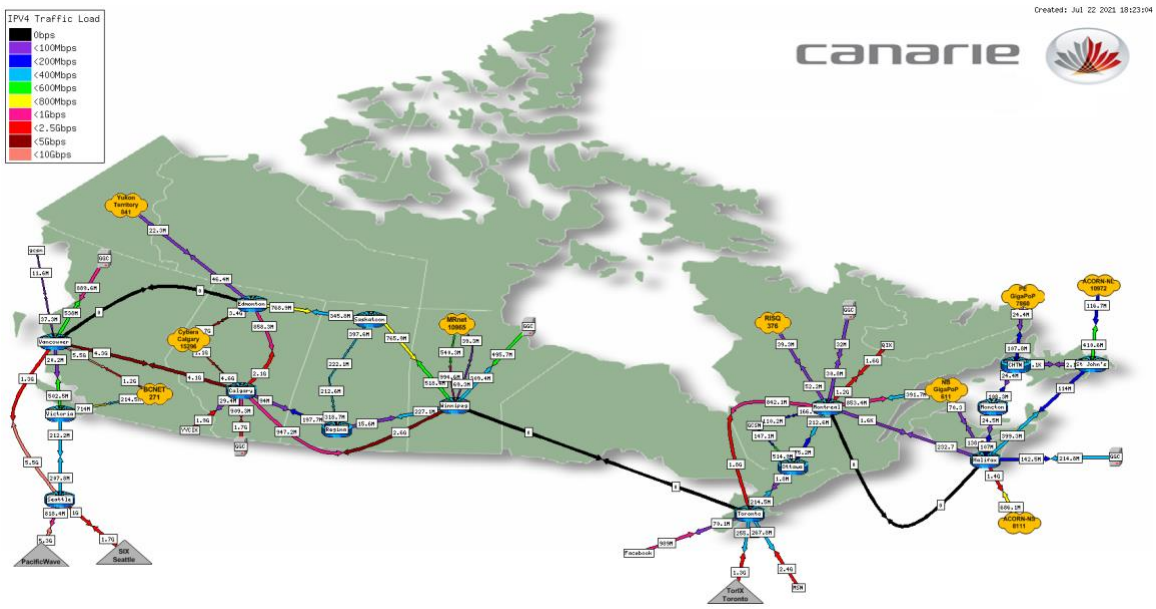
3. What is the big-Oh time complexity for searching an entire graph using *depth-first search* (DFS)? Does the representation of the graph make a difference? Justify your answer.

O(V+E), the representation doesn't make a difference since it needs to search an entire graph. But if just search a certain node for adjacency matrix it's O(1) time while for adjacency list it's O(n) time.

4. What is the big-Oh time complexity for searching an entire graph using *breadth-first search* (BFS)? Does the representation of the graph make a difference? Justify your answer.

O(V+E), the representation doesn't make a difference since it needs to search an entire graph. But if just search a certain node for adjacency matrix it's O(1) time while for adjacency list it's O(n) time.

Team Work use the given graph



1. Are there cycles in the network?

yes for example: Vancouver->Calgary->Edmonton->Vancouver since it's a traffic graph, it's reasonable that edges between different vertices are bi-directional and multiple circles exist.

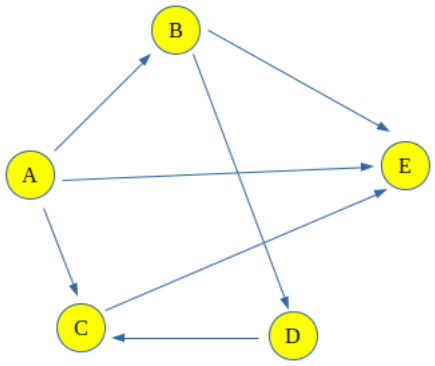
2. Given a start node and a destination node, how many "hops" along a path does it take for data to travel between nodes? (Notes: you can assume there is always a path, and you do not need to find the shortest path... yet!) :)

Vancouver(src)->Calgary->Regina->Wimmipeg->Toronto(dst) 4 hops in total

the number of hops equals the number of edges between source node and destination node.

3. Given a start node and a destination node, what are all of the possible paths between them?

the original graph is too complex for this question, I use another relatively simpler graph instead.



all paths from A(src) to E(dst) are:

A->E, A->B->E, A->C->E, A->B->D->C->E, can use backtracking to count all the paths.

4. Which of DFS or BFS are most appropriate for answering these questions, and why?

BFS is suitable to find shortest path between two nodes since it explores the graph level by level thus more efficiently find shorted path without need to try longer paths.

DFS is suitable to find cycles & different paths since it's easier to detect cycle when recursion stack holds duplicate vertice with current vertice;

for exploring all paths, it's more efficient for dfs to find path connecting src and dst then return the current path then continue searching, even though bfs can perform similar operation.