# Algorithms and Data Structures

**COMP261**

**Tutorial Week 1**

Alex Potanin, Yi Mei

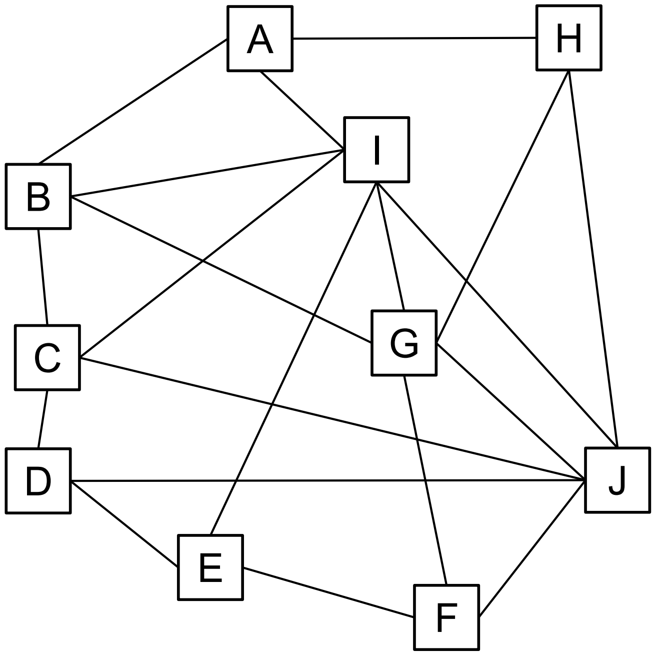
*yi.mei@ecs.vuw.ac.nz*

# Outline

- Data structures of graph
  - Adjacency matrix
  - Adjacency list

# Adjacency Matrix

- Use a 2D matrix to represent the graph
  - Number of rows and columns = number of nodes
  - $M_{ij} = \boxed{w_{ij}}$ is the weight (e.g. length) of the directed edge from *i* to *j*
  - $M_{ij} = \infty$, or leave blank if there is no edge from *i* to *j*.
- Cannot deal with multi-graph.




| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| A | | 5 | | | | | | 5 | 2 | |
| B | 5 | | 3 | | | | 7 | | 6 | |
| C | | 3 | | 1 | | | | | 7 | 9 |
| D | | | 1 | | 3 | | | | | 9 |
| E | | | | 3 | | 4 | | | 9 | |
| F | | | | | 4 | | 5 | | | 4 |
| G | | 7 | | | | 5 | | 6 | 3 | 4 |
| H | 5 | | | | | | 6 | | | 7 |
| I | 2 | 6 | 7 | | 9 | | 3 | | | 6 |
| J | | | 9 | 9 | | 4 | 4 | 7 | 6 | |

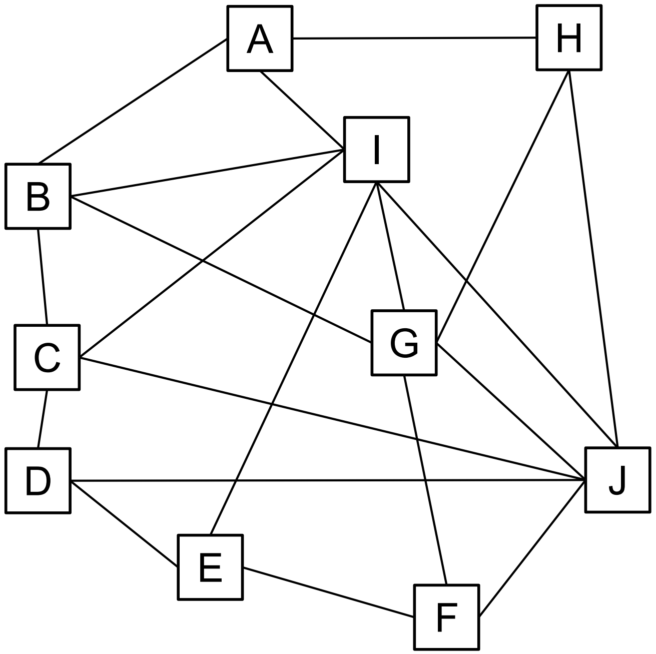
# Adjacency Matrix

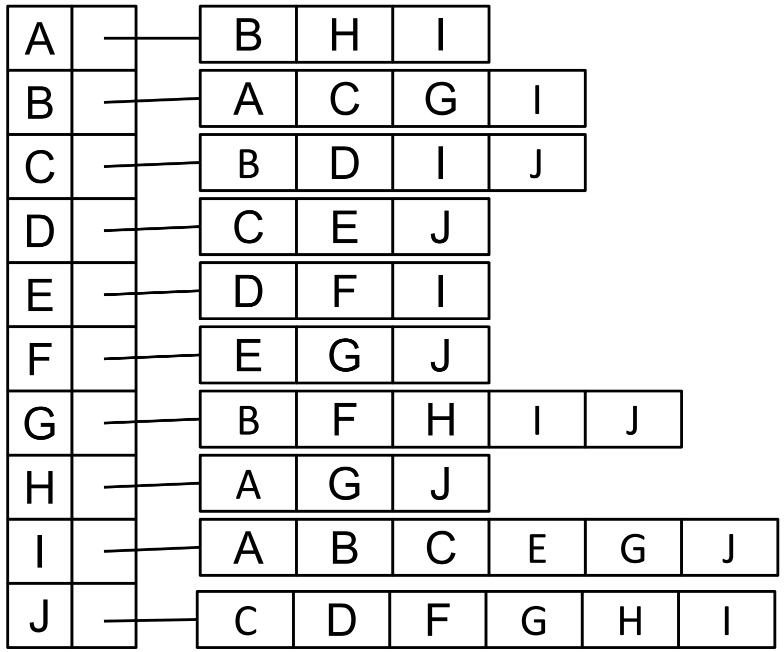
- Use a 2D matrix to represent the graph
  - Number of rows and columns = number of nodes
  - $M_{ij}$ is a list of edge objects
- An edge object is unique for each edge

```
Class Edge {
    Node fromNode;
    Node toNode;
}

Edge e1 = new Edge(A, B);
Edge e2 = new Edge(A, B);

System.out.println(e1.equals(e2));
```

# Adjacency List

- For each node, store
  - A list of outgoing edge objects
  - A list of incoming edge objects
- Need to store a list of edge objects to store edge information, e.g. edge length



# Complexity Comparison

- Assume simple graph: at most one edge between each pair of nodes, with $N$ nodes and $M$ directed edges
- Max Degree of the graph: $\Delta_{in} = \Delta_{out} = \Delta$
  - **Adjacency matrix**: each entry stores an edge object
  - **Adjacency list**: each node has two lists, one for outgoing edge objects, and the other for incoming edge objects

| | **Adjacency Matrix** | **Adjacency List** |
|---|---|---|
| Find all nodes | $O(N)$ | $O(N)$ |
| Find all edges | $O(N^2)$ | $O(M)$ |
| Find all outgoing edges of a node | $O(N)$ | $O(\Delta)$ |
| Find all incoming edges of a node | $O(N)$ | $O(\Delta)$ |
| Find all outgoing node neighbours of a node | $O(N)$ | $O(\Delta)$ |
| Find all incoming node neighbours of a node | $O(N)$ | $O(\Delta)$ |
| Check if there is an edge from u to v | $O(1)$ | $O(\Delta)$ |

- Adjacency list has better complexity overall

# Example in Assignment 1

**Roads (roadID-roadInfo.tab)**

| roadid | type | name | city | 1way | sp | rc | !car | !ped | !bic |
|---|---|---|---|---|---|---|---|---|---|
| 16060 | 6 | cowley st | waterview | 0 | 2 | 0 | 0 | 0 | 0 |
| 16473 | 6 | walmer rd | point chevalier | 0 | 2 | 0 | 0 | 0 | 0 |
| 16501 | 4 | carrington rd | point chevalier | 0 | 2 | 2 | 0 | 0 | 0 |

**Nodes (nodeID-lat-lon.tab)**

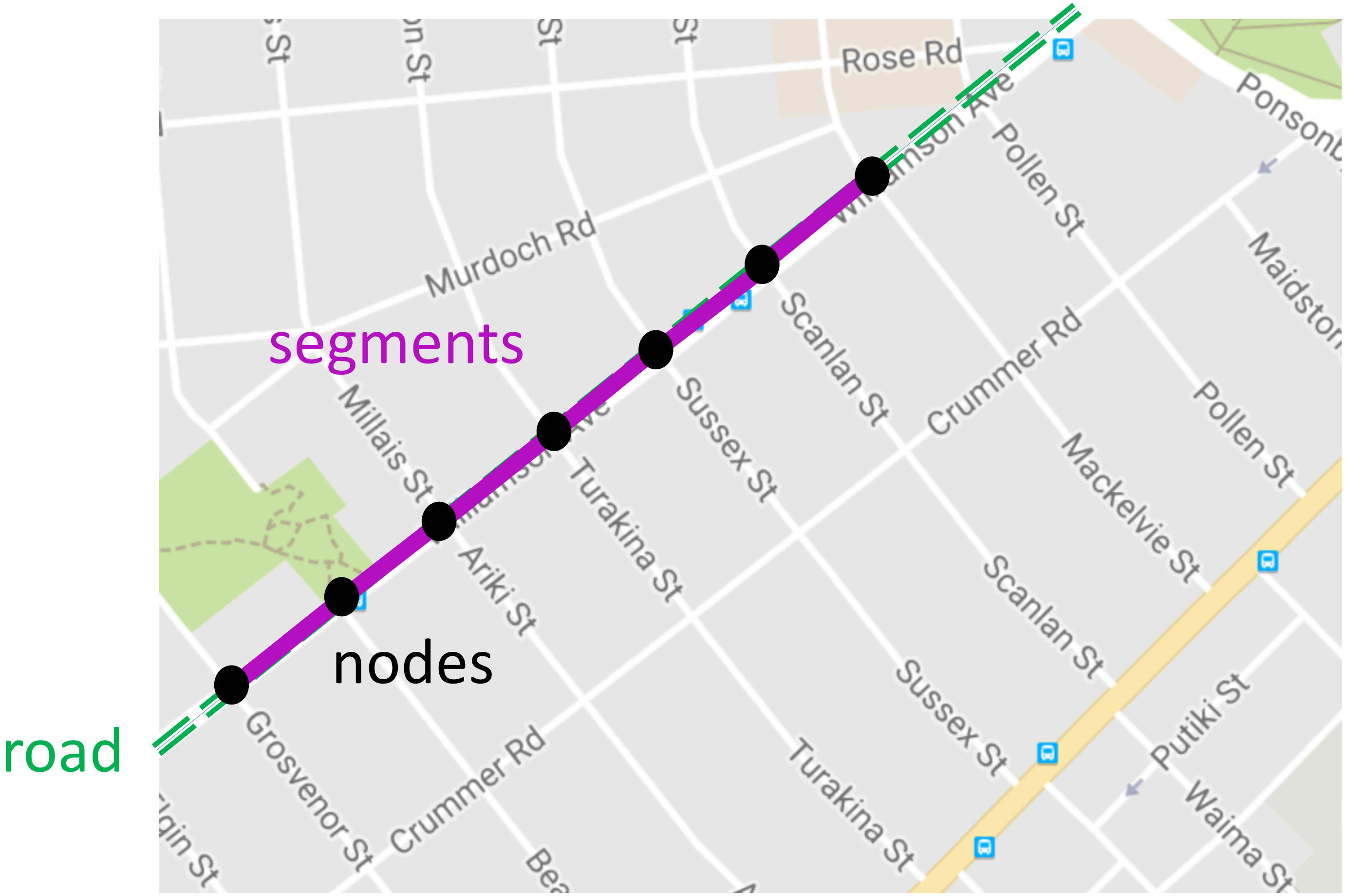
| | | |
|---|---|---|
| 10526 | -36.871900 | 174.693080 |
| 10518 | -36.871780 | 174.693510 |
| 10845 | -36.872000 | 174.699370 |

**Road segments (roadSeg-roadID-length-nodeID-nodeID-coords.tab)**

| roadID | length | nodeID1 | nodeID2 | coords |
|---|---|---|---|---|
| 16060 | 0.223 | 12420 | 12556 | -36.88853 174.72218 -36.88954 174.72361 -36.88992 174.72398 |
| 16501 | 0.243 | 13612 | 13689 | -36.88977 174.73364 -36.88765 174.73431 |
| 100 | 0.020 | 16931 | 16956 | -36.85512 174.76492 -36.85529 174.76501 |

# Example in Assignment 1

- **Roads** vs **road segments**
  - A **road** can have many intersections (e.g. Williamson Ave)
  - A **road segment** has no intersection in the middle
- **node = intersection, edge = segment**



# Example in Assignment 1

- Use adjacency list
  - Each node has two lists, one for outgoing edges, the other for incoming edges

- Graph
  - A set of nodes (intersections)
  - A set of edges (segments)

- Node class: outgoing edges, incoming edges, id, location, …
- Edge class: fromNode, toNode, id, length, …

# Example in Assignment 1

- Graph
  - Map<Integer, Node> nodes;  // key is node id
  - Collection<Segment> segments;
- Node
  - int id;
  - List<Segment> outSegs;
  - List<Segment> inSegs;
  - …
- Segment
  - int id;
  - Road road;
  - Node fromNode;
  - Node toNode;
  - …