COMP 261 2024 Tri 1
Graphs and Data Structures

Assign 1 due Friday noon

- 10 Tutorials
- 4 Help desks
- Comp261-help@ecs.vuw.ac.nz

Graph

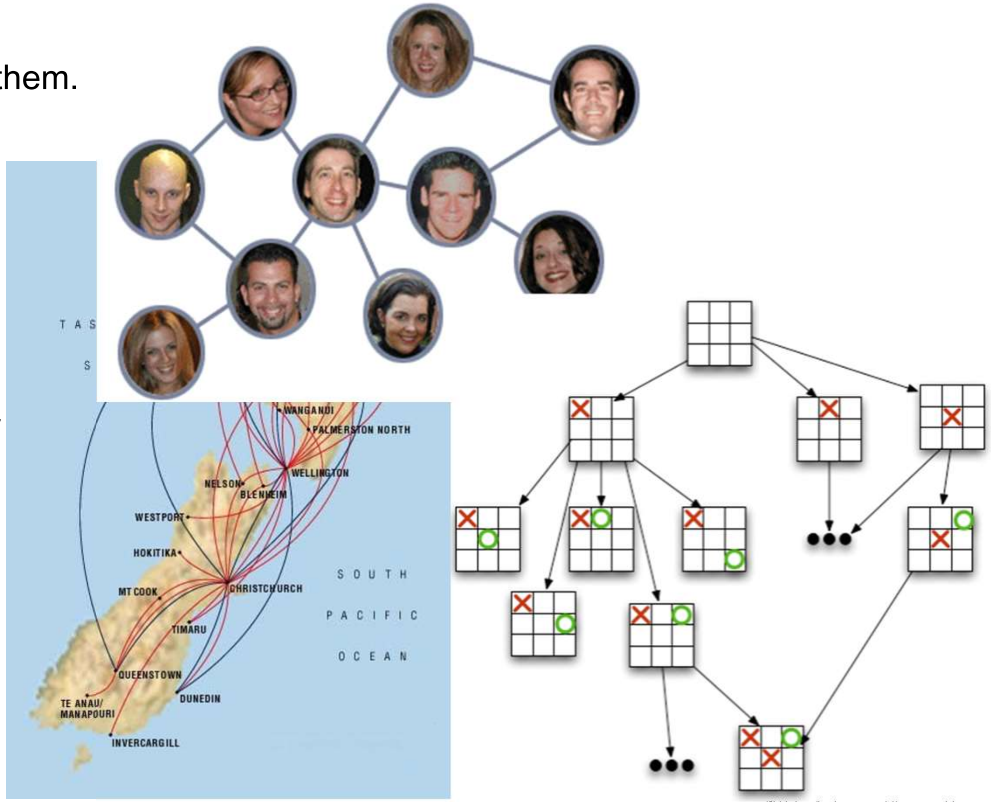
- Collection of nodes and edges:
  - entities and relationships between them.

- Many real-world applications
  - places with connections

    airports & flights,
    intersections & roads,
    stations & railway tracks
    network switches and cables ….
  - entities with relationships

    social networks,
    biological models
    web pages ….
  - states and actions

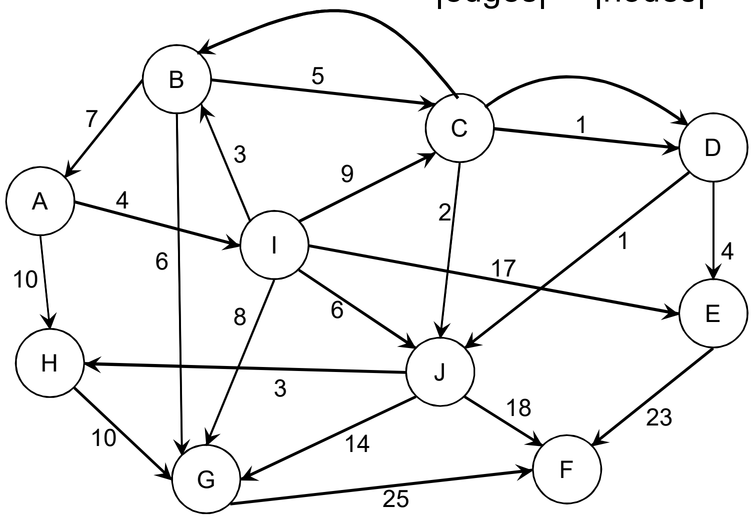
    games, plans, …..



Different Kinds of Graphs

- Directed or Undirected:
  Are the edges symmetric or one-way?
- Single or multi-graph:
  Can there be two edges between a pair of nodes?
- Do the edges have information attached?
  weights, lengths, labels,….
- Bipartite graphs
  Two kinds of nodes
  Edges between types
  (eg: supervisors and projects)
- Is the graph known, or is it constructed as you traverse it ("Implicit" graph)

- Sparse Graphs
  most pairs of nodes not connected
  $|edges| \ll |nodes|^2$
- Dense Graphs
  nodes connected to most other nodes
  $|edges| \approx |nodes|^2$



Graph Data Structure

What data structure(s) should be use to represent a graph?

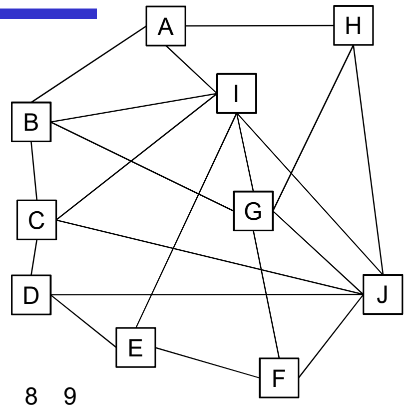
- A good data structure should support the important operations efficiently
  - e.g.
    - Find all the nodes of the graph
    - Find all the edges of the graph
    - Find all outgoing edges of a node
    - Find all incoming edges of a node
    - Find all the outgoing node neighbours of a node
    - Find all the incoming node neighbours of a node
    - Find out whether two nodes are directly connected or not
    - Find the edge between two nodes (if connected)
  - …
- Two traditional data structures
  - Adjacency matrix,
  - adjacency list

Object-based data structures

Collection of Node objects with lists of neighbours

Collection of Edge objects with pairs of Nodes

Adjacency Matrix



- Use integers 0..n-1 to represent nodes
- Use an array to represent info about nodes

```
private Node[] nodes;
```

0	1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I	J

- Use a 2D matrix to represent the graph

```
private Edge[] edges;
```

  - Number of rows and columns = number of nodes
  - $M_{ij} = 1$ if there is an edge from node i to node j
  - $M_{ij} = 0$ (blank) otherwise

	0	1	2	3	4	5	6	7	8	9
0		5						5	2	
1	5		3				7		6	
2		3		1					7	9
3			1		3					9
4				3		4			9	
5					4		5			4
6		7				5		6	3	4
7	5						6			7
8	2	6	7		9		3			6
9			9	9		4	4	7	6	

Edge object

- What about edges with labels (lengths/weights/capacities/etc)?

- Cannot deal with multi-graphs.

Time Complexity of Adjacency Matrix

- Assume simple graph: at most one edge between each pair of nodes, with N nodes and E edges, typically $N < E < N^2$
- 2D adjacency matrix, requires $O(N^2)$ memory space
- Time cost:
  - Find all nodes
  - Find all edges
  - Find all outgoing edges of a node
  - Find all incoming edges of a node
  - Find all outgoing node neighbours
  - Find all incoming node neighbours
  - Check if there is an edge between two nodes

Undirected or Directed?

To:

From:	0	1	2	3	4	5	6	7	8	9
0		5						5	2	
1	5		3				7		6	
2		3		1					7	9
3			1		3					9
4				3		4			9	
5					4		5			4
6		7				5		6	3	4
7	5						6			7
8	2	6	7		9		3			6
9			9	9		4	4	7	6	

Adjacency List

- Use integers 0..n-1 to represent nodes,
  and array to represent info about nodes:

  ```
  private Node[] nodes;
  ```

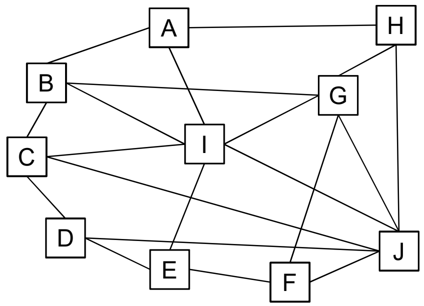
- Use an array of arrays/lists to represent the graph

  ```
  private int[][] neighbours;    or
  private List<Integer>[] neighbours;
  ```

- What about edge information?

  Lists could store edge objects containing
  - nodes at each end
  - length/capacity/labels on edges

  ```
  private List<Edge>[] edges;
  ```



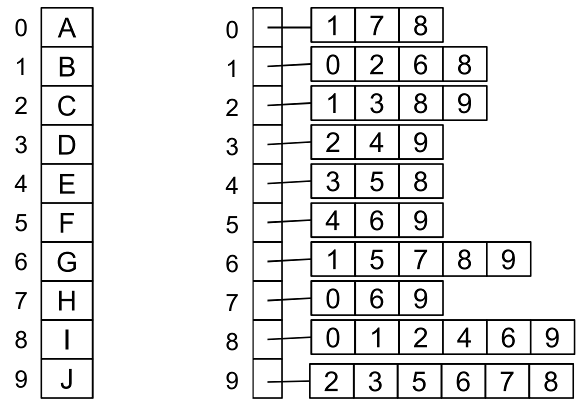
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
9	J

0	1	7	8			
1	0	2	6	8		
2	1	3	8	9		
3	2	4	9			
4	3	5	8			
5	4	6	9			
6	1	5	7	8	9	
7	0	6	9			
8	0	1	2	4	6	9
9	2	3	5	6	7	8

Time Complexity of Adjacency List

- Assume simple graph: at most one edge between each pair of nodes, with N nodes and E directed edges, assume $N < E < 2N^2$
- node.adjList() is a list of outgoing **node neighbours** of node *i*

  - Find all nodes
  - Find all edges
  - Find all edges of a node
  - Find all node neighbours
  - Check if there is an edge between two nodes



Adjacency List, Directed Graph

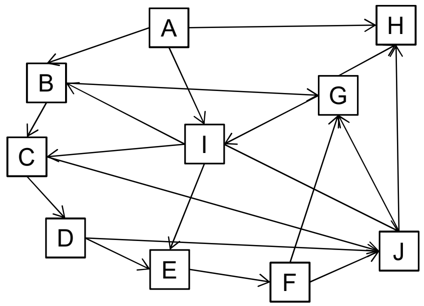
Same data structure

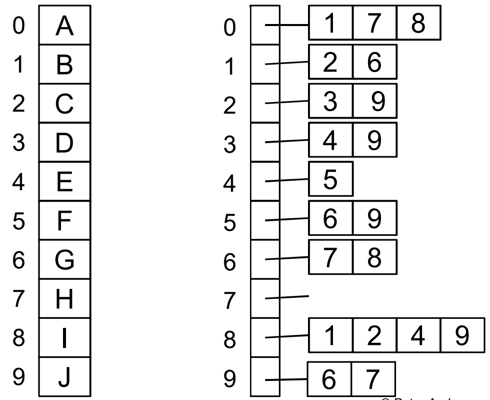
- Use integers 0..n-1 to represent nodes, and array to represent info about nodes:

  ```
  private Node[] nodes;
  ```

- Use an array of arrays/lists to represent the graph

  ```
  private int[][] outNeighbours;    or
  private List<Integer>[] outNeighbours;
  private List<Edge>[] outEdges;
  ```





Time Complexity of Adjacency List, Directed

- Assume simple graph: at most one edge between each pair of nodes, with N nodes and E directed edges, assume $N < E < 2N^2$
- If graph has a maximum in-degree and/or out-degree: Δ_{in}, Δ_{out}, $\Delta = \max(\Delta_{in}, \Delta_{out})$
  - (maximum number of neighbours)
  - Find all nodes
  - Find all edges
  - Find all outgoing edges of a node
  - Find all incoming edges of a node
  - Find all outgoing node neighbours
  - Find all incoming node neighbours
  - Check if there is an edge between two nodes

0	A	0	1, 7, 8
1	B	1	0, 2, 6, 8
2	C	2	1, 3, 8, 9
3	D	3	2, 4, 9
4	E	4	3, 5, 8
5	F	5	4, 6, 9
6	G	6	1, 5, 7, 8, 9
7	H	7	0, 6, 9
8	I	8	0, 1, 2, 4, 6, 9
9	J	9	2, 3, 5, 6, 7, 8

i^{th} list has the outgoing **neighbours** of node i