COMP261 Lecture 4

Graphs 3 of 3

Victoria UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui
CAPITAL CITY UNIVERSITY

Traditional Adjacency Matrix

Nodes referred to by an integer, 0..n
Array of node labels/info
Edges represented by entries in a 2D array
- Numbers for weights, NaN or sentinel number for no edge
- Strings for labels, null for no edge

		0	1	2	3	4	5	6	7	8	9
0	A		7						10	4	
1	B	7		5				6		3	
2	C		5		1					9	2
3	D			1		4					1
4	E				4		23			17	
5	F					23		25			18
6	G		6				25		10	8	14
7	H	10						10			3
8	I	4	3	9		17		8			6
9	J			2	1		18	14	3	6	

What Questions are fast?
- All nodes?
- Neighbours of node?
- Neighbours *to* node?
- Edge exists from j to k ?

Directed or Undirected?

Traditional Adjacency Matrix

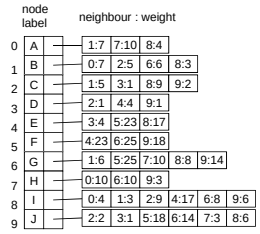
Nodes referred to by an integer, 0..n
Array of node labels/info
Edges represented by entries in a 2D array
- Numbers for weights, NaN or sentinel number for no edge
- Strings for labels, null for no edge

		0	1	2	3	4	5	6	7	8
0	A									
1	B	7								
2	C		5							
3	D			1						
4	E				4					
5	F					23				
6	G		6				25			
7	H	10						10		
8	I	4	3	9		17		8		
9	J			2	1		18	14	3	6

If Undirected

Traditional Adjacency List

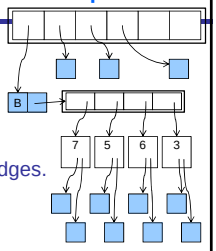
Nodes referred to by an integer 0..n
Array of node labels & array of lists
Lists of neighbour and edge weight/label



What Questions are fast?
- All nodes?
- Neighbours of node?
- Neighbours *to* node?
- Edge exists from j to k ?

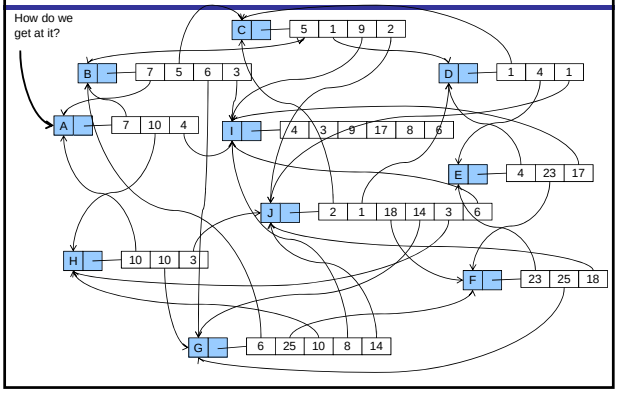
Directed or Undirected?

"Modern" Data Structure for Graphs

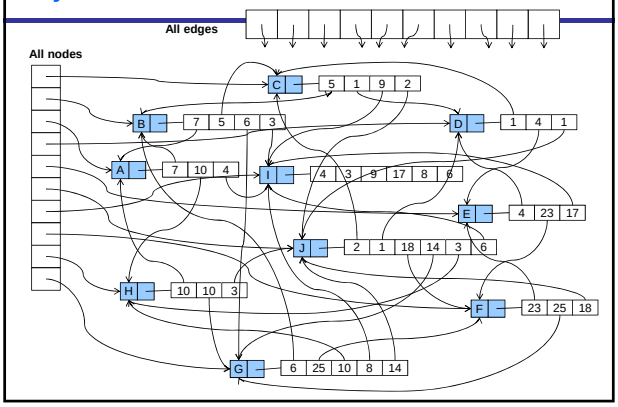


- We have a set of nodes
- Each node has edges out of it
  ⇒ Node should contain a collection of edges.
- (Directed) Each edge has a node at the other end
  ⇒ edge should contain a destination node.
  ⇒ edge might contain the source node as well

Object-based data structure.



Object-based data structure.



Java Type Declarations

- The graph:
  Set<Node> nodes = new HashSet<Node>();
- Nodes:
  String label;
  Set<Edge> edgesOut = new HashSet<Edge>();
  [Set<Edge> edgesIn = new HashSet<Edge>();]
- Edges:
  double length;
  Node start;
  Node end;

Data Structure for the Road Map

- Road objects (homogeneous chunks of road)
  - ID#, name, one way, speed limit, type, …..

  354, High St, Warkworth, false, 0, 6, …..
- Intersection objects
  - ID#, location, list of road segments out, [list of road segments in?]

  8743, 3.2, 4.8
- Road Segment objects
  - Road they are part of, intersection from, intersection to, geometry = list of locations for drawing

    3.2,4.8 3.3,5.0…
  - Why ID#'s?
  - road segment → intersection ? via ID or direct?
  - road segment → road ? via ID or direct?

Data structures for the Road Map

- The graph

```
Map<Integer,Node> nodes = new HashMap<Integer,Node>();
```

- The node (intersection)

```
List<Segment> outNeighbours = new ArrayList<Segment>(2);
List<Segment> inNeighbours = new ArrayList<Segment>(2);
```

- The Segment (the edge)

```
Node startNode;
Node endNode;
…
```

The Roads

```
Map<Integer,Road> roads = new HashMap<Integer,Road>()
```

Road objects

```
List<Segment> segments = new ArrayList<Segment>();
```

Segment objects

```
Node StartNode
Node  EndNode
Road road
```

Note about terminology: road consists of road segments which are between two nodes each and each road segment in turn is not just a straight line but a list of coordinates that can be used to draw a segment…

Loading data

- Read the Road data and the Intersection data
  - Remember to convert latitude/longitude to x/y coordinates (in kms).
- Put the objects into structures that gives fast access by ID.
- Read the RoadSegment data,
  - Convert the latitude/longitude data into x/y coordinates.
  - Look up the road and intersections by ID
  - Construct the road segment object(s) containing the road and intersections.
  - Add the road segment to the intersection objects
  - Add the road segment to the road objects
  - If road is not one way, you will need to create another segment object
  - Add it to the intersections