Recitation 13

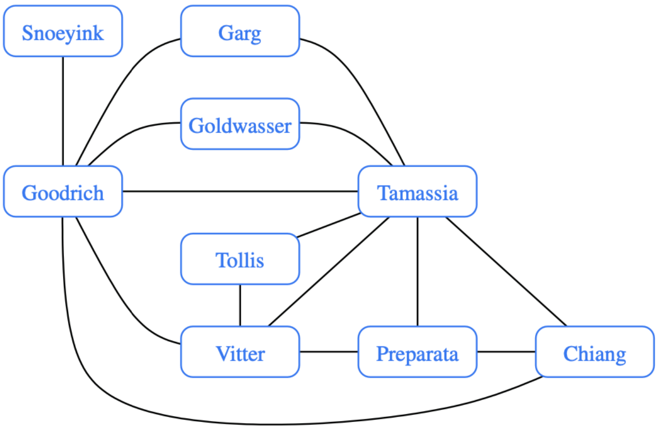

Figure 14.1: Graph of coauthorship among some authors.

R-14.4 Draw an adjacency list representation of the undirected graph shown in Figure 14.1.

R-14.7 Give pseudocode for performing the operation insertEdge(u, v, x) in $O(1)$ time using the adjacency matrix representation.

R-14.11 Would you use the adjacency matrix structure or the adjacency list structure in each of the following cases? Justify your choice.

a. The graph has 10,000 vertices and 20,000 edges, and it is important to use as little space as possible.

b. The graph has 10,000 vertices and 20,000,000 edges, and it is important to use as little space as possible.

c. You need to answer the query getEdge(u, v) as fast as possible, no matter how much space you use.

C-14.38 Suppose we wish to represent an n-vertex graph G using the edge list structure, assuming that we identify the vertices with the integers in the set $\{0, 1, \ldots, n-1\}$. Describe how to implement the collection E to support $O(\log n)$-time performance for the getEdge(u, v) method. How are you implementing the method in this case?

Simplified from github

```
public class EdgeListGraph<V, E>{
	public class Vertex<V> {
		V element;
	}
	public class Edge<E, V>{
		E element;
		Vertex<V>[] endpoints;
	}
	DoublyLinkedList<Vertex<V>> vertices;
	DoublyLinkedList<Edge<E, V>> edges;
	int n_vertices;
	int n_edges;
	List<Edge<E, V>> outgoingEdges(Vertex<V> v);
	List<Edge<E, V>> incomingEdges(Vertex<V> v);
	Edge<E, V> getEdge(Vertex<V> from, Vertex<V> to);
	Vertex<V>[] endVertices(Edge<E, V> e);
	Vertex<V> opposite(Vertex<V> v, Edge<E, V> e);
}
```

Implement the following methods for the class

```
Vertex<V> findVertexWithValue(V v) {
	for (Vertex<V> i: vertices) {
		if (i.element.equals(v)) return i;
	}
	Return null;
}

Edge<E, V> getEdgeWithValues(V a, V b) {
	for (Vertex<V> i: vertices) { O(n)
		If (i.element.equals(a)) {
			For (Edge e: i.incomingEdges()) { O(m)
				Vertex j = opposite(i, e);
				if (j.element.equals(b)) return e;
			}
		} else if (i.element.equals(b))
	}
	Return null;
}
```

O(nm)

O(m)

Method	Edge List	Adj. List	Adj. Map	Adj. Matrix
numVertices( )	$O(1)$	$O(1)$	$O(1)$	$O(1)$
numEdges( )	$O(1)$	$O(1)$	$O(1)$	$O(1)$
vertices( )	$O(n)$	$O(n)$	$O(n)$	$O(n)$
edges( )	$O(m)$	$O(m)$	$O(m)$	$O(m)$
getEdge(u, v)	$O(m)$	$O(\min(d_u, d_v))$	$O(1)$ exp.	$O(1)$
outDegree(v) inDegree(v)	$O(m)$	$O(1)$	$O(1)$	$O(n)$
outgoingEdges(v) incomingEdges(v)	$O(m)$	$O(d_v)$	$O(d_v)$	$O(n)$
insertVertex(x)	$O(1)$	$O(1)$	$O(1)$	$O(n^2)$
removeVertex(v)	$O(m)$	$O(d_v)$	$O(d_v)$	$O(n^2)$
insertEdge(u, v, x)	$O(1)$	$O(1)$	$O(1)$ exp.	$O(1)$
removeEdge(e)	$O(1)$	$O(1)$	$O(1)$ exp.	$O(1)$