Graphs
John Von Neumann
1903-1957
AVL Rotation Issues

• The wayward node problem
• Deletions
  • may require up to $\lg(n)$ rotations
Graphs

- Consist of nodes and edges
  - edges may be
    - weighted or unweighted
    - Directed or undirected
- No distinguished starting location
- Loops allowed

A graph with 4 nodes and unweighted, directed edges
Representing Graphs

• Edge List
  • store edges in an array list

• Adjacency Matrix
  • store nodes in a 2D array

• Adjacency List
  • store edges from node in a list with node
Nodes and Edges

- Represent using internal classes
- Some graph representations will not need both

```java
private class Node<H> {
    public H payload;
    public Node(H payl) {
        this.payload = payl;
    }
}

private class Edge<J> {
    Node<J> from;
    Node<J> too;
    double wei;
    public Edge(Node<J> fr, Node<J> t2) {
        this(fr, t2, 1.0);
    }
    public Edge(Node<J> fr, Node<J> t2, double w) {
        this.from = fr;
        this.too = t2;
        this.wei = w;
    }
}
```
Edge Lists

- Just store every edge in graph

- often inconvenient

  - for instance, “where can I go from A”

```java
class EdgeList<G> {
    ArrayList<Edge<G>> egLi;
    HashMap<G, Node<G>> nodeHash;

    private Node<G> getNode(G g) {
        Node<G> ret = nodeHash.get(g);
        if (ret == null) {
            ret = new Node<>(g);
            nodeHash.put(g, ret);
        }
        return ret;
    }

    public void addEdge(G fr, G t2) {
        egLi.add(new Edge<G>(getNode(fr), getNode(t2)));
    }
}
```
public static void main(String[] args) {
    EdgeList<String> edl = new EdgeList<>("A");
edl.addEdge("A", "B");
edl.addEdge("A", "D");
edl.addEdge("B", "D");
edl.addEdge("C", "D");
edl.addEdge("C", "B");
edl.addEdge("C", "A");
edl.addEdge("D", "B");
System.out.println(edl);
}
Adjacency Matrix

- Edge lists are awkward.
- Instead, store edges implicitly in 2D array
  - rows are from
  - columns are to
- So, “where can I get to from A” is just read across row.
- Unweighted edges,
  - store T/F in matrix
- Weighted edges
  - store weights.
  - The “no edge problem”
public class AdjMatrix<M> {

    // The adjacency matrix
    double[][][] aMatrix;
    ArrayList<M> nodes;

    public AdjMatrix(ArrayList<M> nds) {
        nodes = new ArrayList<>();
        for (M nm : nds) {
            nodes.add(nm);
        }
        aMatrix = new double[nodes.size()][nodes.size()];
        for (int i=0; i<nodes.size(); i++) {
            for (int j = 0; j < nodes.size(); j++) {
                aMatrix[i][j] = -99;
            }
        }
    }

    public void addEdge(M fr, M t2, double w) {
        int fri = nodes.indexOf(fr);
        int t2i = nodes.indexOf(t2);
        if (fri < 0 || t2i < 0)
            return;
        aMatrix[fri][t2i] = w;
    }
}
Adjacency Lists

- Each node holds list of edges leaving the node
- Add an ArrayList of edges to the node definition
- Edge need only store destination
- How do you store bi-directional links?

```java
private class Node<H> {
    // Node content
    public H payload;
    // hold the list
    public ArrayList<Edge<G>> edges;

    public Node(H payl) {
        this.payload = payl;
        this.edges = new ArrayList<Edge<G>>();
    }

    public void addEdge(Node<G> n, double w) {
        edges.add(new Edge<G>(n, w));
    }
}
```
Adjacency Lists

- Only store in outer class is the nodes
- Get to from X is just the list in X

```java
public class AdjList<G> {
    private HashMap<G, Node<G>> nodeHash;

    public void addNode(G g) {
        nodeHash.put(g, new Node<G>(g));
    }

    public void addEdge(G fr, G t2, double w) {
        Node<G> frn = nodeHash.get(fr);
        Node<G> t2n = nodeHash.get(t2);
        if (frn == null || t2n == null)
            return;
        frn.addEdge(t2n, w);
    }
}
```