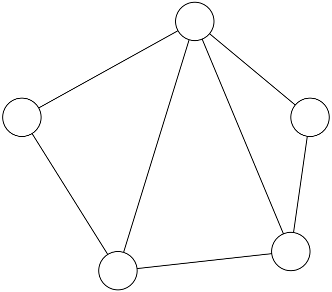
Graphs, Part II

February 4, 2011

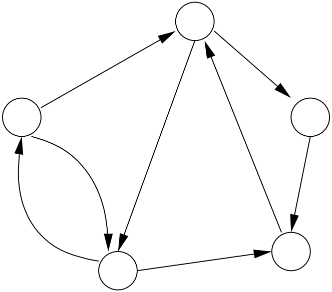
What Is a Graph?

- Set of nodes (or vertices)
- Set of edges between pairs of nodes

Undirected

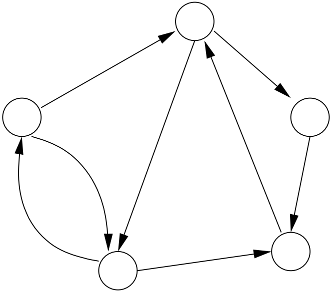


Directed

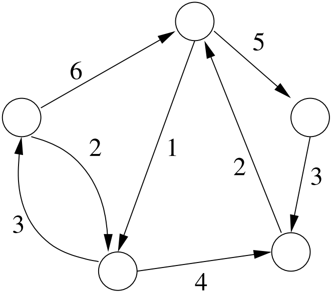


Types of Graphs

Unweighted



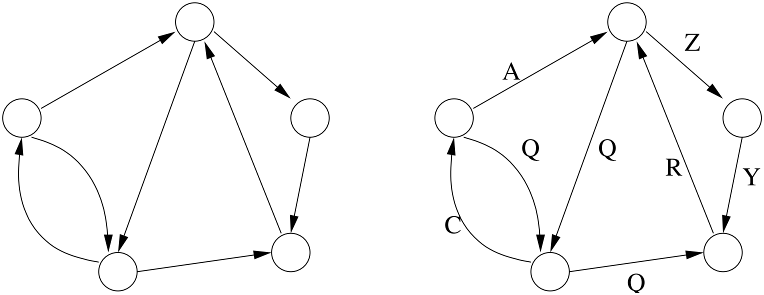
Weighted



Types of Graphs

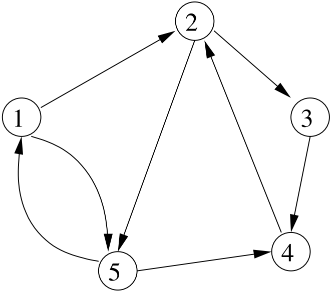
Unlabelled

Labelled



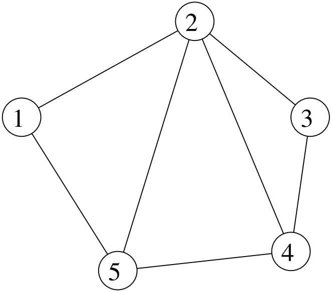
Adjacency List Representation (Directed Graph)

- Assume nodes are numbered 1 to n.
- Use an array of lists, $list[1..n]$.
- For each node u, $list[u]$ contains nodes v for which there is an edge $u \rightarrow v$.



$$\begin{aligned} list[1] &= \{2, 5\} \\ list[2] &= \{3, 5\} \\ list[3] &= \{4\} \\ list[4] &= \{2\} \\ list[5] &= \{1, 4\} \end{aligned}$$

Adjacency List Representation (Undirected Graph)



$$
\begin{aligned}
list[1] &= \{2, 5\} \\
list[2] &= \{1, 3, 4, 5\} \\
list[3] &= \{2, 4\} \\
list[4] &= \{2, 3, 5\} \\
list[5] &= \{1, 2, 4\}
\end{aligned}
$$

Adjacency Matrix vs Adjacency Lists

- Adjacency matrix is simpler.
- Adjacency matrix is good for dense graphs (i.e., more than half of the edges present). Note: 1000 vertices $\Rightarrow$ 1 MB of memory.
- Adjacency lists are good for sparse graphs (i.e., fewer than half of the edges present).

Adjacency Lists in Java

- Instead of using an array of linked lists, use Java's built-in data structures that can access elements faster.
- For unlabelled graph, use array of TreeSet.
- For labelled or weighted graph, use array of TreeMap.

Example: Unlabelled Directed Graph

Assume nodes are numbered 1 to n.

- Create the data structure:

```
Set<Integer>[] list = new TreeSet[n+1];
for (int i=1; i<=n; i++)
    list[i] = new TreeSet<Integer>();
```

- Add an edge u $\rightarrow$ v:

```
list[u].add(v);
```

- Check if there is an edge u $\rightarrow$ v:

```
boolean isEdge = list[u].contains(v);
```

- Iterate across all nodes v for which there is an edge u $\rightarrow$ v:

```
for (int v : list[u]) {...}
```

Example: Labelled Directed Graph

Use TreeMap instead of TreeSet.
Key of entry is the destination, value of entry is the label.

- Create the data structure:

```
Map<Integer,String>[] list = new TreeMap[n+1];
for (int i=1; i<=n; i++)
    list[i] = new TreeMap<Integer,String>();
```

- Add a labelled edge u $\rightarrow$ v:

```
list[u].put(v,label);
```

- Check if there is an edge u $\rightarrow$ v:

```
boolean isEdge = list[u].containsKey(v);
```

- Get label associated with edge u $\rightarrow$ v:

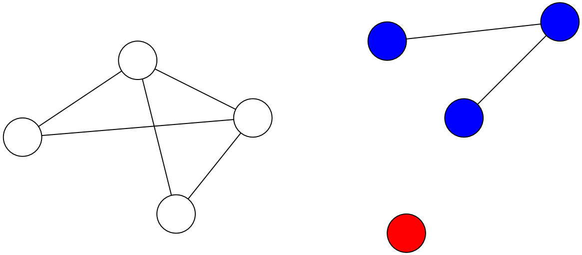
```
String label = list[u].get(v);
```

- Iterate across all nodes v for which there is an edge u $\rightarrow$ v:

```
for (int v : list[u].keySet()) {...}
```

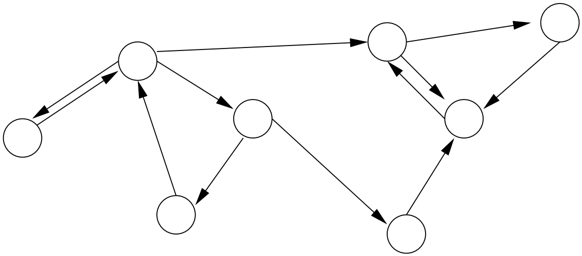
More Graph Terminology: Connectivity

- A *subgraph* of a graph G is a graph whose vertices and edges are all in G.
- An undirected graph is *connected* if there is a path from each node to each other node.
- A *connected component* of an undirected graph G is a maximal connected subgraph of G.



More Graph Terminology: Connectivity

- A directed graph is *strongly connected* if there is a path from each node to each other node.
- A *strongly connected component* of a directed graph G is a maximal connected subgraph of G.



More Graph Terminology: Connectivity

- A directed graph is *strongly connected* if there is a path from each node to each other node.
- A *strongly connected component* of a directed graph G is a maximal connected subgraph of G.