• Our first non-linear data structure!
• A graph $G$ consists of two sets $G = \{V, E\}$
  • A set of $V$ vertices, or nodes
  • A set of $E$ edges, relationships between nodes
• A subgraph $G'$ consists of a subset of the vertices and edges of $G$
• Adjacent vertices are two vertices joined by an edge
A graph that has an edge between each pair of distinct vertices

How many edges does a complete graph with n nodes have?

A connected graph
- A graph that has a path between each pair of distinct vertices

A disconnected graph
- A graph that has at least one pair of vertices without a path between them
• **Directed** graph
  - Each edge is a directed edge, or an arc, or link
  - May have two arcs between a given pair of vertices, one in each direction
  - Vertex y is adjacent to vertex x if and only if there is a directed edge from x to y

• **Directed Acyclic Graph (DAG)**
  - Directed graph with no cycles

How few arcs can you remove to make the graph a DAG?

• **Weighted** graph
  - A graph whose edges have weights
  - Weight is the “cost” or “magnitude” of the relationship represented by the edge

```java
public interface Graph<T>   // partial
{
  public boolean isEmpty();  // returns true iff a graph is empty
  public int n();           // returns the number of vertices in a graph
  public int m();           // returns the number of edges in a graph
  public void addVertex(T v);  // Insert a vertex in a graph
  public void removeVertex(T v);  // Deletes a vertex from a graph along with any edges between the vertex and other vertices
  public void addEdge(T v1, T v2);  // Insert an edge between two given vertices in a graph
  public void removeEdge(T v1, T v2);  // Deletes the edge between two given vertices in a graph
  public T findVertex(String key);  // Retrieves and returns the vertex that contains a given search key
  public boolean isEdge(T v1, T v2);  // returns true iff an edge exists between two given vertices
  public LinkedList<T> getNeighbors(T v);  // FOR UNDIRECTED GRAPH
  // Retrieves and returns a list of the vertices adjacent to vertex v
}```
• **Adjacency matrix for graph with**
  - vertices: $n$ numbered 0, 1, ..., $n - 1$
  - arcs: boolean $n \times n$ array where $arcs[i][j] =$
    - 1 (true) iff there is an arc from vertex $i$ to vertex $j$
    - 0 (false) iff there is no arc from vertex $i$ to vertex $j$

![Adjacency matrix diagram]

What property does the matrix of an undirected graph have?

- An adjacency list for graph with
  - $n$ vertices numbered 0, 1, ..., $n - 1$
  - arcs: array of $n$ linked lists
    - The $i^{th}$ linked list has a list entry for vertex $j$
      iff the graph contains an arc from vertex $i$ to vertex $j$

![Adjacency list diagram]

- Which representation supports better these two frequent operations on graphs?
  - isEdge($v$, $w$)
    Determine whether there is an edge from vertex $v$ to vertex $w$
  - getNeighbors($v$)
    Return list of all vertices linked to from a given vertex $v
Adjacency matrix for a weighted graph with \( n \) vertices numbered 0, 1, ..., \( n - 1 \)
- An \( n \times n \) array matrix \( \text{EdgeW} \) such that \( \text{EdgeW}[i][j] = \)
  - The weight of the arc from vertex \( i \) to vertex \( j \) iff there is an edge from \( i \) to \( j \)
  - \( \infty \) iff there is no edge from vertex \( i \) to vertex \( j \)

Multiple Edges

- **Multigraph**
  - Not a graph
  - Allows multiple edges between vertices
  - Multiple edges indicate multiple relations between vertices
  - We will not deal with multigraphs

Tree: A Special Graph

- A **tree** is a connected graph in which there is exactly one simple path connecting any two nodes

How many edges does a tree with \( n \) nodes have?
• yEd: A great and simple graph visualization
• Download it from http://www.yworks.com/en/products_yed_about.html
• You can create any graph by clicking (for vertices) and clicking-and-dragging (for edges)
• Lots of graph formats supported. Use .tgf
• TGF format: a text file listing lines of:
  * vertexID vertexName (for vertices)
  * #
  * vertexID pairs (for arcs)
• Once you upload a file, choose Layout > Circular to see it laid out nicely.

• A graph-traversal algorithm
  – Visits all the vertices that it can reach starting at some vertex
  – Visits all vertices of the graph iff the graph is connected (effectively computing Connected Components)
  – Must not loop forever, if a graph contains a cycle
  – Must never visit a vertex more than once
• Connected component (for undirected graphs) =
  – The subset of vertices visited during a traversal that begins at a given vertex
• Strongly connected component (for directed graphs) =
  – The subset of vertices visited during a traversal that begins at any of its members
High Planes Airline Company (HPAir) Problem

- For each customer request, indicate whether a sequence of HPAir flights exists from the origin city to the destination city.

The flight map for HPAir is a directed graph

- Arc between vertices (a,b) means
  - There is a flight from city a to city b
- Directed path means
  - There is a sequence of flight connections

DFS(originCity): Searching the Flight Map

```
stk = new Stack<E>();
stk.push(originCity);
while (a sequence of flights from originCity to destinCity has not been found) {
    if (you cannot go anywhere from the city on top of stack)
        stk.pop(); // backtrack
    else
        select a neighbor, anotherCity, from the city on top of stack;
        stk.push(anotherCity);
}
```

... and remember where you’ve been

```
stk = new Stack<E>(); Clear Marks;
stk.push(originCity);
Mark(originCity) as visited;
while (a sequence of flights from originCity to destinCity has not been found) {
    if (you cannot find an unvisited city from the city on top of stack)
        stk.pop(); // backtrack
    else
        select an unvisited neighbor, anotherCity, from the city on top of stack;
        stk.push(anotherCity);
        Mark(anotherCity) as visited;
}
Depth-First-Search Example: From P->Z

Would DFS(oC) work for undirected graphs?

List visited (marked)
P R X W S T Y Z

Stack stk

Mazes as Graphs

Testing for Connectivity using DFS(oC)

Connected: An undirected graph for which there is a path from any node to any other node

Is this graph connected?

Connected component: A connected sub-graph

Can we use DFS to find all connected components?