Graphs

- Nonlinear collection consisting of nodes and links
- Node is referred to as **vertex**, link (connecting 2 vertices) is called **edge**
- Even less "order" than a tree because there's no root, etc.

- **Undirected graph**: each edge connects 2 vertices with no direction
- **Directed graph**: each edge connects a source vertex to a target vertex

- **Loop:** edge that connects a vertex to itself
- **Path:** sequence of vertices such that each adjacent pair of vertices are connected by an edge (**length** of path is # edges)
- **Cycle:** path that begins and ends with same vertex

- **Simple graph:** no loops, no multiple edges

- **Degree of vertex:** in undirected graph it is # edges it is part of; in directed graph it is # edges coming into it (**in-degree**), or # edges going out of it (**out-degree**)

Implementations

Adjacency Matrix

bool A[n][n] where n = # vertices

A[i][j] = true if there is an edge from vertex i to vertex j; otherwise, false

Adjacency List (a.k.a. Edge Lists)

NODE* A[n] where n = # vertices

A[i] is linked list of all vertices with which vertex i has an edge

Which implementation is better to use depends on how often you'll be:

- Adding or removing edges
- Checking whether a particular edge is present
- Processing each edge for a particular source vertex

Traversals

Depth-First Search (DFS)

```
initialize visited[n] to false;

for each vertex v do
    if (visted[ v ] == false)
        DFS(v);


void DFS(vertex v) {
  visited[ v ] = true;
  for each neighbor w of v do
     if (visited[ w ] == false)
       DFS(w);
}
```

Breadth-First Search (BFS)

```
initialize visited[n] to false;

for each vertex v do
    if (visted[ v ] == false)
        BFS(v);


void BFS(vertex v) {

   visited[ v ] = true;
   enqueue vertex v into rear of queue;
   while (queue is not empty) {
      dequeue vertex u from front of queue;
      for each neighbor w of u do
         if (visited[ w ] == false) {
            visited[ w ] = true;
            enqueue w into rear of queue;
        }
      }
   }
}
```