CMSC 132:
Object-Oriented Programming II

Graph Implementation

Department of Computer Science

University of Maryland, College Park

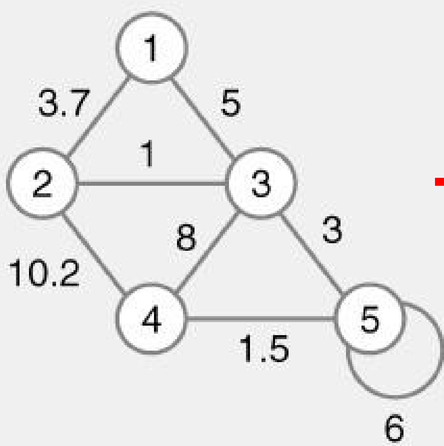
Graph Implementation

- How do we represent edges?
  - Adjacency matrix
    - 2D array of neighbors
  - Adjacency list
    - List of neighbors
  - Adjacency set / map
    - Set / map of neighbors

- Important for very large graphs
  - Affects efficiency / storage

Adjacency Matrix

- **Representation**
  - **2D array**
  - **Position j, k $\Rightarrow$ edge between nodes n_j, n_k**
- **Example**



1	∘				
2	3.7	∘			
3	5.0	1.0	∘		
4	∘	10.2	8.0	∘	
5	∘	∘	3.0	1.5	6.0
	1	2	3	4	5

Adjacency Matrix

- **Representation (cont.)**
  - **Single array for entire graph**
  - **Undirected graph**
    - **Only upper / lower triangle matrix needed**
    - **Since n_j, n_k implies n_k, n_j**
  - **Unweighted graph**
    - **Matrix elements $\Rightarrow$ boolean**
  - **Weighted graph**
    - **Matrix elements $\Rightarrow$ weight**

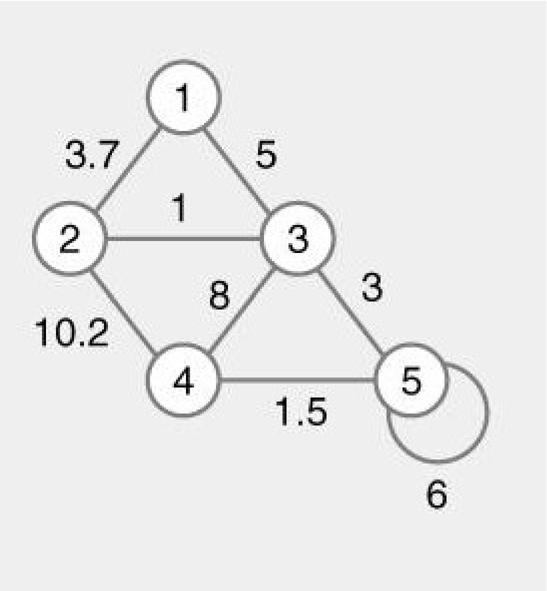
Adjacency List/Set

- Representation
  - For each node, store
    - List/Set of neighbors / successors
      - Linked list
      - Array list
  - For weighted graph
    - Also store weight for each edge
    - Using a Map is a good choice
  - For undirected graph with edge (a $\leftrightarrow$ b)
    - Nodes a & b need to store each other as neighbor
  - For directed graph with edge (a $\rightarrow$ b)
    - Node a needs to store node b as neighbor

Adjacency List

- **Example**
  - **Unweighted graph**

node 1: {2, 3}
node 2: {1, 3, 4}
node 3: {1, 2, 4, 5}
node 4: {2, 3, 5}
node 5: {3, 4, 5}



  - **Weighted graph**

node 1: {2=3.7, 3=5}
node 2: {1=3.7, 3=1, 4=10.2}
node 3: {1=5, 2=1, 4=8, 5=3}
node 4: {2=10.2, 3=8, 5=1.5}
node 5: {3=3, 4=1.5, 5=6}

Adjacency Set / Map

- Representation
  - For each node, store
    - Set or map of neighbors / successors
  - For unweighted graph
    - Use set of neighbors
  - For weighted graph
    - Use map of neighbors, w/ value = weight of edge
  - For undirected graph with edge (a ↔ b)
    - Nodes a & b need to store each other as neighbor
  - For directed graph with edge (a → b)
    - Node a needs to store node b as neighbor

Graph Space Requirements

- **Adjacency matrix**
  - **½ N^2 entries (for graph with N nodes, E edges)**
  - **Many empty entries for large, sparse graphs**
- **Adjacency list**
  - **2×E entries**
- **Adjacency set / map**
  - **2×E entries**
  - **Space overhead per entry**
    - **Higher than for adjacency list**

Graph Time Requirements

- **Adjacency matrix**
  - Can find individual edge (a,b) quickly
  - Examine entry in array Edge[a,b]
    - Constant time operation
- **Adjacency list / set / map**
  - Can find all edges for node (a) quickly
  - Iterate through collection of edges for a
    - On average E / N edges per node

Graph Time Requirements

- **Average Complexity of operations**
  - **For graph with N nodes, E edges**

Operation	Adj Matrix	Adj List	Adj Set/Map
Find edge	O(1)	O(E/N)	O(1)
Insert edge	O(1)	O(E/N)	O(1)
Delete edge	O(1)	O(E/N)	O(1)
Enumerate edges for node	O(N)	O(E/N)	O(E/N)

Choosing Graph Implementations

- **Graph density**
  - **Ratio edges to nodes (dense vs. sparse)**
- **Graph algorithm**
  - **Neighbor based**

```
For each node X in graph
    For each neighbor Y of X  // adj list faster if sparse
        doWork( )
```

  - **Connection based**

```
For each node X in …
    For each node Y in …
        if (X,Y) is an edge          // adj matrix faster if dense
            doWork( )
```