CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Graph Implementation

Department of Computer Science
University of Maryland, College Park
Graph Implementation

- How do we represent nodes/edges?
  - Adjacency matrix
    - 2D array of neighbors
  - Adjacency list/set/map
    - List/set/map of neighbors
- Important for very large graphs
  - Affects efficiency / storage
Adjacency Matrix

- Single **two-dimensional array** for entire graph
- Directed Graph
  - **Unweighted graph**
    - Matrix elements ⇒ boolean
  - **Weighted graph**
    - Matrix elements ⇒ values
    - Let’s see an example of each
- Undirected Graph
  - Let’s see an example for unweighted graph
  - Let’s see an example for weighted graph
- For Undirected graph
  - Only upper / lower triangle matrix needed
  - Since \( n_j \), \( n_k \) implies \( n_k \), \( n_j \)
Adjacency List/Set/Map

- For each node, **store neighbor information in a list, set, or map**
- The main structure can be a list, set, or map
- Directed Graph
  - **Unweighted graph**
    - List or set of neighbors
  - **Weighted graph**
    - Each entry keeps track of neighbor and weight
    - Easy to implement with maps
      - Maps of Maps (using HashMaps for efficiency)
  - Let’s see an example of each
- Undirected Graph
  - Let’s see an example for unweighted graph
  - Let’s see an example for weighted graph
Additional Examples

• Examples
  • Unweighted graph
    
    | Node  | Connections |
    |-------|-------------|
    | 1     | {2, 3}      |
    | 2     | {1, 3, 4}   |
    | 3     | {1, 2, 4, 5}|
    | 4     | {2, 3, 5}   |
    | 5     | {3, 4, 5}   |

  • Weighted graph
    
    | Node  | Connections |
    |-------|-------------|
    | 1     | {2=3.7, 3=5}|
    | 2     | {1=3.7, 3=1, 4=10.2}|
    | 3     | {1=5, 2=1, 4=8, 5=3}|
    | 4     | {2=10.2, 3=8, 5=1.5}|
    | 5     | {3=3, 4=1.5, 5=6}  |
Graph Properties

- **Graph density**
  - Ratio edges to nodes (dense vs. sparse)
  - For adjacency matrix many empty entries for large, sparse graph

- **Adjacency matrix**
  - Can find individual edge \((a,b)\) quickly
  - Examine entry in array \(\text{Edge}[a,b]\)
    - 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
### Complexity

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Adj Matrix</th>
<th>Adj List</th>
<th>Adj Set/Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find edge</td>
<td>$O(1)$</td>
<td>$O(E/N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Insert edge</td>
<td>$O(1)$</td>
<td>$O(E/N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Delete edge</td>
<td>$O(1)$</td>
<td>$O(E/N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Enumerate edges for node</td>
<td>$O(N)$</td>
<td>$O(E/N)$</td>
<td>$O(E/N)$</td>
</tr>
</tbody>
</table>
Choosing Graph Implementations

- Factors to consider
  - Graph density
  - Graph algorithm
    - Neighbor based
      For each node X in graph
      For each neighbor Y of X  // adj list faster if sparse
      \( \text{doWork}(\ ) \)
    - Connection based
      For each node X in …
      For each node Y in …
      if \((X,Y)\) is an edge  // adj matrix faster if dense
      \( \text{doWork}(\ ) \)