Graph Algorithms

CptS 223 – Advanced Data Structures

Larry Holder
School of Electrical Engineering and Computer Science
Washington State University
Graphs

Protein-protein Interaction

Social Network

Power Grid

Internet

Web
Some Graph Statistics

- **Web**
  - 10B pages, 1T hyperlinks
  - Topology storage: 10TB
  - Google PageRank: Eigenvector on 10Bx10B adjacency matrix (sparse)

- **MySpace**
  - 100M users, 10B friendship links
  - Clique/community detection
  - 300K new users per day
Graph Problems

- Degree
- Diameter
- Centrality
- Shortest path
- Cycles/tours
- Minimum spanning tree
- Traversals/search
- Connectivity

- Clustering
- Partitioning
- Cliques
- Motifs
- Subgraph isomorphism
- Frequent subgraphs
- Pattern learning
- Dynamics
Definitions

- Graph $G = (V, E)$ consists of vertices $V$ and edges $E$
- $E = \{(u, v) \mid u, v \in V\}$
  - $v$ is adjacent to $u$
- E.g.,
  - $V = \{A, B, C, D, E, F, G\}$
  - $E = \{(A, B), (A, D), (B, C), (C, D), (C, G), (D, E), (D, F), (E, F)\}$
Definitions

- **Undirected graph**
  - Edges are unordered

- **Directed graph**
  - Edges are ordered

- **Weighted graph**
  - Edges have a weight $w(u,v)$ or cost $c(u,v)$
Definitions

- **Degree of a vertex**
  - Number of edges incident on a vertex

- **Indegree**
  - Number of directed edges to vertex

- **Outdegree**
  - Number of directed edges from vertex

![Diagram of a graph with vertices v1 to v7 and their degrees]

- $\text{degree}(v_4) = 6$
- $\text{indegree}(v_4) = 3$
- $\text{outdegree}(v_4) = 3$

- $\text{indegree}(v_1) = 0$
- $\text{outdegree}(v_1) = 3$

- $\text{indegree}(v_6) = 3$
- $\text{outdegree}(v_6) = 0$
Definitions

- **Path**
  - Sequence of vertices $v_1, v_2, \ldots, v_N$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i < N$
  - Path **length** is number of edges on path $(N-1)$
  - **Simple** path has unique intermediate vertices

- **Cycle**
  - Path where $v_1 = v_N$
  - Usually simple and directed
  - **Acyclic** graphs have no cycles
Definitions

- Undirected graph is connected if there is a path between every pair of vertices.
- Connected, directed graph is called strongly connected.
- Complete graph has an edge between every pair of vertices.
Representation

Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\(|V^2|\)

Adjacency List

\(|V| + |E|\)
Topological Sort

Order vertices in a directed, acyclic graph such that if \((u,v) \in E\), then \(u\) before \(v\) in the ordering.

Topological order:
\[v_1, v_2, v_5, v_4, v_3, v_7, v_6\]
Topological Sort

- **Solution #1**
  - While vertices left in graph \( \mathcal{O}(|V|) \)
    - Find vertex \( v \) with indegree = 0 \( \mathcal{O}(|V|) \)
    - Output \( v \)
    - Remove edges to/from \( v \)
  - \( T(V,E) = \mathcal{O}(|V|^2) \)

Topological order: 
\( v_1, v_2, v_5, v_4, v_3, v_7, v_6 \)
Topological Sort

- Solution #2
  - Don’t need to search all vertices for indegree = 0
  - Only vertices who lost an edge from the previous vertex’s removal
  - \( T(V,E) = O(|V| + |E|) \)
  - Note: \( |E| = O(|V|^2) \)

Topological order:

\[ v1, v2, v5, v4, v3, v7, v6 \]
void Graph::topsort()
{
    Queue<Vertex> q;
    int counter = 0;

    q.makeEmpty();
    for each Vertex v
        if( v.indegree == 0 )
            q.enqueue( v );

    while( !q.isEmpty() )
    {
        Vertex v = q.dequeue();
        v.topNum = ++counter; // Assign next number

        for each Vertex w adjacent to v
            if( --w.indegree == 0 )
                q.enqueue( w );

    }

    if( counter != NUM_VERTICES )
        throw CycleFoundException();
}
### Topological Sort

<table>
<thead>
<tr>
<th>Vertex</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$v_2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$v_3$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$v_4$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$v_5$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$v_6$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$v_7$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Enqueue**
- $v_1$
- $v_2$
- $v_5$
- $v_4$
- $v_3, v_7$
- $v_6$

**Dequeue**
- $v_1$
- $v_2$
- $v_5$
- $v_4$
- $v_3$
- $v_7$
- $v_6$
Graph Algorithms

- Topological Sort
- Shortest paths
- Network flow
- Minimum spanning tree
- Applications
- NP-Completeness