Lecture 18:
Graph Representations

Not in book
What is a Graph?

• Representation of data and relationships
• Points connected by lines
• The points are *vertices (or nodes)*, the lines *edges*

A graph is defined by two sets $G = \{V, E\}$

Where $V$ is a set of vertices, e.g. $V = \{a, b, c, d, e\}$

$E$ is a set of edges given by 2-tuples, e.g. $E = \{(a, c), (a, d), (b, d), (b, e), (c, e), (d, c)\}$
Symbolic Representation

- A drawing of a graph is a useful visual aid, but a graph is not a drawing.
- Identical graphs can be drawn differently.
- Complicated graphs can be difficult to draw.
- We’ll focus on how graphs are represented in a computer, and how to write graph algorithms.
Special Types Graphs

• Directed Graphs (digraph)
  – Graph in which each edge has a direction
  – Vertices with only outgoing edges are called “sources” and vertices with only incoming edges are called “sinks”
  – Digraph paths that repeat vertices are called “cycles”

• Directed Acyclic Graphs (DAGs)
  – A digraph containing no cycles
More Graph Variants

• Weighted graphs
  – Edges or vertices can have associated “weights”
  – Weights can represent anything
    • Distances, costs, capacities, etc.

• Trees
  – A tree is a general (undirected) graph containing no cycles.
  – Note: Generally, you can consider an edge in an undirected graph as “bidirectional”
Graph Details

• More Terminology
  – Vertex “degree” – the number of edges that connect to a specified vertex
  – Directed graphs have two types of degrees
    • The number of incoming edges - “indegree”
    • The number of outgoing edges - “outdegree”.

• Typical Graph Problems
  – Path-related problems
  – Connectedness problems
  – Spanning tree problems
Path Finding

- Path between “a” and “h”.

Path length is 20.

Comp 590/Comp 790-90   Fall 2009
Path Finding

• A second path from “a” to “h”

Path length is 28

• What is the shortest path?
• Is there a path through all edges?
• What is the minimal cost path through all vertices?
Example Of No Path

No path between a and h.
Connected Graph

• Undirected graph.
• There is a path between every pair of vertices.
There is no path from a to h (e, f, or g).
The subgraphs $SG_1 = \{a, b, c, d\}$, and $SG_2 = \{e, f, g, h\}$ are “connected components”.

![Diagram of connected components](image-url)
Connected Component

• A maximal subgraph that is connected.
  ▪ Cannot add an additional vertices and its edges from original graph and still retain connectedness.

• A connected graph has exactly \(1\) component.
• Removal of an edge that is on a cycle does not affect connectedness.
• Connected graph that has no cycles.
• \( n \) vertex connected graph with \( n-1 \) edges.
Spanning Tree

- Subgraph that includes all vertices of the original graph.
- Subgraph is a tree.
  - If original graph has $n$ vertices, the spanning tree has $n$ vertices and $n-1$ edges.
Minimum Cost Spanning Tree

- Reduce graph to tree of smallest cost

- Tree cost is sum of edge weights/costs.
Minimum Cost Spanning Tree

- Approach – Remove the most expensive edge on each cycle

Spanning tree cost = 29.
Graph Representations

- How to store graphs in a computer?
  - Adjacency Matrix
  - Adjacency Lists
    - Linked Adjacency Lists
    - Array Adjacency Lists
Adjacency Matrix

- Binary \((1/0)\ n \times n\) matrix, where \(n = \# \) of vertices
- \(A(i,j) = 1\) iff \((i,j)\) is an edge

```
1 0 1 0 1 0
2 1 0 0 0 1
3 0 0 0 0 1
4 1 0 0 0 1
5 0 1 1 1 0
```
• Diagonal entries are zero.

• Adjacency matrix of an undirected graph is symmetric.

\[ A(i,j) = A(j,i) \text{ for all } i \text{ and } j. \]
• Diagonal entries are zero. (If self-edges are disallowed)

• Adjacency matrix of a digraph need not be symmetric.
Adjacency Matrix

- $n^2$ bits of space
- For an undirected graph, may store only lower or upper triangle (exclude diagonal).
  - $(n-1)n/2$ bits
- $O(n)$ time to find vertex degree and/or the set of all vertices adjacent to a given vertex.
- Constant time to test if an edge exists
Adjacency Lists

• Adjacency list for vertex \( i \) is a linear list of vertices adjacent to each vertex \( i \).
• A dictionary of lists. \( \text{aList} = \{\} \)

\[
\begin{align*}
\text{aList}[1] &= [2,4] \\
\text{aList}[2] &= [1,5] \\
\text{aList}[4] &= [5,1] \\
\text{aList}[5] &= [2,4,3]
\end{align*}
\]
Linked Adjacency Lists

- Or, Alternatively, a list of lists.

![Graph diagram]

\[aList = \{[[1, 3], [0, 4], [4], [0, 4], [1, 2, 3]]\}\]

- # dictionary entries = \(n\)
- # of adjacencies = \(2e\) (undirected graph)
- # of adjacencies = \(e\) (digraph)
Weighted Graphs

• Cost adjacency matrix.
  Stores the edge weight in the adjacency matrix
  - \( C(i,j) = \text{cost of edge (i,j)} \)

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Weighted Graphs

- Adjacency lists =>
  each list element is a pair
  (adjacent vertex, edge weight)

- \( G = \{ 'a': [('b',2), ('d',5)],
  'b':[('c',4),('d',1)],
  'c':[('e',1)],
  'd':[('c',2),('e',4)],
  'e':[]\} \)