Graphs

Weiss, from the beginning of chapter 14 through section 14.1 (pages 527–539)
Graph

• Applications
  – Airline routes
  – Computer networks
  – Course prerequisite

https://courses.cs.washington.edu/courses/cse373/12sp/lectures/05-11-graphs/19-graphs.pdf
Graph (cont.)

• A data structure that contains
  – Set of vertices $V$
  – Set of edges $E$
  – $(v, w)$ is an edge that connects $v$ and $w$

• Represent graph as $G = (V, E)$
  – $V = \{ V0, V1, V2, V3 \}$
  – $E = \{ (V0, V1), (V0, V2), (V1, V2), (V2, V3) \}$
  – $|V| = 4$, $|E| = 4$
Graph (cont.)

• Path is a sequence of connected vertices
  – Path from V0 to V3 is { V0, V2, V3 }
  – Path length is the number of edges (2)

• Undirected graphs: Edge (V0, V2) implies (V2, V0)

• Weighted graphs
  – An edge has a weight (cost)
  – E.g., distance of flight between two airports

• Directed graphs
  – Edges (V0, V2) and (V2, V0) have different weights.

• Cycle is a path from a vertex to itself
  – E.g., {V0, V2, V0}

• Best path from V0 to V3
  – The fewest number of edges
  – Sum of edge weights is minimum
Graph Representation

• Adjacency matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INF</td>
<td>4</td>
<td>10</td>
<td>INF</td>
</tr>
<tr>
<td>1</td>
<td>INF</td>
<td>INF</td>
<td>5</td>
<td>INF</td>
</tr>
<tr>
<td>2</td>
<td>INF</td>
<td>INF</td>
<td>INF</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>INF</td>
<td>INF</td>
<td>INF</td>
<td>INF</td>
</tr>
</tbody>
</table>

$O(|V|^2)$ time and space

• Adjacency list

0 1, 4  ➔ 2, 10 .
1 2, 5 .
2 3, 2 .
3

$O(|V| + |E|)$ time and space
Weiss Implementation

- Two components
  - Vertex object
  - Vertex map

<table>
<thead>
<tr>
<th>dist</th>
<th>prev</th>
<th>name</th>
<th>adj</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>V1</td>
<td>V2</td>
<td></td>
</tr>
</tbody>
</table>

- name: vertex name V2
- adj: adjacency list (V2, V3)
- dist: cost of the shortest path from the source vertex \{V0, V1, V2\}
- prev: vertex that is connected to this vertex on the shortest path V1

Start vertex

V0

V1 - V2

V3

V0 - V2

V0 - V1

V0 - V3

Input file

<table>
<thead>
<tr>
<th>V0</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

- V2
  - dist: 9
  - prev: V1

- V3
  - dist: 11

- V1
  - dist: 4
  - prev: V0

- V0
  - dist: 0
Breadth-First Graph Traversal

Weiss, section 14.2 (pages 539–545)
Shortest-Path Algorithms

• Shortest in terms of time, distance, cost, ...

• Applications
  – Map navigators
  – Flight reservation

• Problem definition
  – Input: weighted directed graph, where each edge $(V_i, V_j)$ has a cost $c_{i,j}$
  – Cost of path $V_1$ to $V_N$ is $\sum_{i=1}^{N-1} c_{i,i+1}$.
  – In the unweighted graph, the cost of all edges is 1.
  – Path length is N-1.
Unweighted Shortest Paths

• Find the shortest path (path with the fewest edges) from a start vertex to every vertex
• Use Breadth-First Search (BFS)
• For each vertex, determine
  – Whether it is visited
  – Its distance from the start vertex
  – Its previous vertex on the shortest path to this vertex
### Vertex visited dist prev

<table>
<thead>
<tr>
<th>Vertex</th>
<th>visited</th>
<th>dist</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>T</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V2</td>
<td>F</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V3</td>
<td>T</td>
<td>0</td>
<td>null</td>
</tr>
<tr>
<td>V4</td>
<td>F</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V5</td>
<td>F</td>
<td>INF</td>
<td>null</td>
</tr>
<tr>
<td>V6</td>
<td>F</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V7</td>
<td>F</td>
<td>INF</td>
<td>null</td>
</tr>
</tbody>
</table>

### Running time: $O(|E|)$

---

### Vertex visited dist prev

<table>
<thead>
<tr>
<th>Vertex</th>
<th>visited</th>
<th>dist</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>F</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V2</td>
<td>F</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V3</td>
<td>T</td>
<td>0</td>
<td>null</td>
</tr>
<tr>
<td>V4</td>
<td>F</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V5</td>
<td>F</td>
<td>INF</td>
<td>null</td>
</tr>
<tr>
<td>V6</td>
<td>F</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V7</td>
<td>F</td>
<td>INF</td>
<td>null</td>
</tr>
</tbody>
</table>

### V1 removed, V2 & V4 added

<table>
<thead>
<tr>
<th>Vertex</th>
<th>visited</th>
<th>dist</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>T</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V2</td>
<td>F</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V3</td>
<td>T</td>
<td>0</td>
<td>null</td>
</tr>
<tr>
<td>V4</td>
<td>F</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V5</td>
<td>F</td>
<td>INF</td>
<td>null</td>
</tr>
<tr>
<td>V6</td>
<td>F</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V7</td>
<td>F</td>
<td>INF</td>
<td>null</td>
</tr>
</tbody>
</table>

### V6 removed

<table>
<thead>
<tr>
<th>Vertex</th>
<th>visited</th>
<th>dist</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>T</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V2</td>
<td>T</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V3</td>
<td>T</td>
<td>0</td>
<td>null</td>
</tr>
<tr>
<td>V4</td>
<td>F</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V5</td>
<td>F</td>
<td>3</td>
<td>V2</td>
</tr>
<tr>
<td>V6</td>
<td>T</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V7</td>
<td>F</td>
<td>INF</td>
<td>null</td>
</tr>
</tbody>
</table>

### V2 removed, V5 added

<table>
<thead>
<tr>
<th>Vertex</th>
<th>visited</th>
<th>dist</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>T</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V2</td>
<td>T</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V3</td>
<td>T</td>
<td>0</td>
<td>null</td>
</tr>
<tr>
<td>V4</td>
<td>F</td>
<td>2</td>
<td>V1</td>
</tr>
<tr>
<td>V5</td>
<td>F</td>
<td>3</td>
<td>V2</td>
</tr>
<tr>
<td>V6</td>
<td>T</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>V7</td>
<td>F</td>
<td>INF</td>
<td>null</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Weiss, section 14.3 (pages 545–552)
Weighted Shortest Paths

- Dijkstra’s algorithm
- Use priority queue to store unvisited vertices
  - Remove a vertex v
  - For each vertex w that is adjacent to v
    - Find out whether v offers a shorter distance for w. If so, update w’s distance and add it to the priority queue.
  - When a vertex is removed, its distance is already at minimum.
There are, as many as $|E|$ insertions and deletions. Running time is $O(|E| \log |E|)$. 

```java
class Path {
    Vertex dest;
    double dist;
}

while( priority queue not empty) {
    // remove a Path object
    // from the priority queue
    // check whether the vertex
    // of this Path object is
    // already processed.
    ...
}
```

There are, as many as $|E|$ insertions and deletions. Running time is $O(|E| \log |E|)$. 

Bellman-Ford Algorithm

Weiss, section 14.4 (pages 552–554)
Negative-Weighted Shortest Paths

• Edge weights may be negative.
• Dijkstra’s algorithm may give incorrect shortest distance for a visited vertex
• The cost of path from $S \rightarrow u \rightarrow v$ may be shorter than $S \rightarrow v$.
• Bellman-Ford algorithm
  – Need to add $v$ to queue and re-visit it

$\text{v.dist} < \text{u.dist}$

$\text{c}_{uv} < 0$
Negative-Weighted Shortest Paths

- Negative-cost cycle
  - The cost of the shortest path is undefined.

- Detect the negative-cost cycle with *scratch* variable
  - Increment *scratch* every time that a vertex is added and removed from queue
    - scratch/2 is the number of times that the vertex is removed
    - For a graph without cycles, scratch/2 ≤ |V| - 1.
Topological Sorting

Weiss, section 14.5 (pages 555–562)
Weighted Shortest Paths for Acyclic Graph

- Acyclic graph has no cycle.
  - No cycle with negative cost
- Topological sorting
  - Order vertices of a directed acyclic graph such that for \((u, v) \in E\) \(u\) appears before \(v\) in the ordering

Topological order: \(v_1, v_2, v_5, v_4, v_3, v_7, v_6\)
Example: order of courses taken at colleges
Weighted Shortest Paths for Acyclic Graph (cont.)

- Algorithm for topological ordering
  - Find a vertex $v$ with no incoming edges
  - Print the vertex $v$
  - Remove the edges from the vertex $v$
  - For a vertex $w$ adjacent to $v$, decrement the number of incoming edges for vertices adjacent to vertex $v$

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Indegree of vertices before removing from queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$v_2$</td>
<td>1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$v_3$</td>
<td>2 1 1 1 0 0 0</td>
</tr>
<tr>
<td>$v_4$</td>
<td>3 2 1 0 0 0 0</td>
</tr>
<tr>
<td>$v_5$</td>
<td>1 1 0 0 0 0 0</td>
</tr>
<tr>
<td>$v_6$</td>
<td>3 3 3 3 2 1 0</td>
</tr>
<tr>
<td>$v_7$</td>
<td>2 2 2 1 0 0 0</td>
</tr>
</tbody>
</table>

add $v_1$ $v_2$ $v_5$ $v_4$ $v_3$, $v_7$ $v_6$
remove $v_1$ $v_2$ $v_5$ $v_4$ $v_3$ $v_7$ $v_6$
Weighted Shortest Paths for Acyclic Graph (cont.)

- Use topological ordering to find the shortest path
Weighted Shortest Paths for Acyclic Graph (cont.)

- Use topological ordering to find the shortest path
Weighted Shortest Paths for Acyclic Graph (cont.)

• Use topological ordering to find the shortest path
Weighted Shortest Paths for Acyclic Graph (cont.)

• Use topological ordering to find the shortest path
Weighted Shortest Paths for Acyclic Graph (cont.)

• Use topological ordering to find the shortest path
Weighted Shortest Paths for Acyclic Graph (cont.)

- Use topological ordering to find the shortest path
Weighted Shortest Paths for Acyclic Graph (cont.)

• Use topological ordering to find the shortest path