COMP2521 23T3
Directed and Weighted Graphs

Kevin Luxa
cs2521@cse.unsw.edu.au

directed graphs
weighted graphs
In graphs representing real-world scenarios, edges are often directional and have a sense of cost.

Thus, we need to consider directed and weighted graphs.
We’ve mostly considered *undirected* graphs: an edge relates two vertices equivalently.

Some applications require us to consider directional edges: \( v \to w \neq w \to v \)

e.g., ‘follow’ on Twitter, one-way streets, etc.

In an **directed graph** or **digraph**: edges have direction; self-loops are allowed.

Each edge \((v, w)\) has a **source** \(v\) and a **destination** \(w\).
Directed Graphs

Example
<table>
<thead>
<tr>
<th>domain</th>
<th>vertex is...</th>
<th>edge is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWW</td>
<td>web page</td>
<td>hyperlink</td>
</tr>
<tr>
<td>chess</td>
<td>board state</td>
<td>legal move</td>
</tr>
<tr>
<td>scheduling</td>
<td>task</td>
<td>precedence</td>
</tr>
<tr>
<td>program</td>
<td>function</td>
<td>function call</td>
</tr>
<tr>
<td>journals</td>
<td>article</td>
<td>citation</td>
</tr>
</tbody>
</table>
in-degree or $d^{-1}(v)$: the number of directed edges leading into a vertex
out-degree or $d(v)$: the number of directed edges leading out of a vertex
Digraph Terminology (II)

directed path: a sequence of vertices $v_1, v_2, \ldots, v_n$ such that $v_i$ has an outgoing edge to $v_{i+1}$
directed cycle: a directed path where the first and last vertices are the same
reachability indicates existence of directed path: if a directed path $v, \ldots, w$ exists, $w$ is reachable from $v$

strongly connected indicates mutual reachability: if both paths $v, \ldots, w$ and $w, \ldots, v$ exist, $v$ and $w$ are strongly connected

strong connectivity every vertex reachable from every other vertex;
strongly-connected component maximal strongly-connected subgraph
Similar choices as for undirected graphs:

- adjacency matrix ... asymmetric, sparse; less space efficient
- adjacency lists ... fairly common solution
- edge lists ... order of edge components matters

Can we make our undirected graph implementations directed? Yes!
Directed Graphs

Implementation: Adjacency Matrix

**Directed Graphs**

![Directed Graphs Diagram]

- **Terminology**
- **Representation**
- **DAGs**
- **Weighted Graphs**

**Adjacency Matrix**

- **Unweighted, Undirected**
  
  \[
  \begin{bmatrix}
  0 & 1 & 1 & 1 & 0 \\
  1 & 0 & 1 & 0 & 0 \\
  1 & 1 & 0 & 1 & 1 \\
  1 & 0 & 1 & 0 & 1 \\
  0 & 0 & 1 & 1 & 0 \\
  \end{bmatrix}
  \]

- **Unweighted, Directed**
  
  \[
  \begin{bmatrix}
  0 & 1 & 1 & 1 & 0 \\
  0 & 0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 1 & 1 \\
  0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 1 & 0 \\
  \end{bmatrix}
  \]
### Digraph Complexity

<table>
<thead>
<tr>
<th></th>
<th>storage</th>
<th>edge add</th>
<th>has edge</th>
<th>outdegree</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjacency matrix</td>
<td>$O(V^2)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(V)$</td>
</tr>
<tr>
<td>adjacency list</td>
<td>$O(V + E)$</td>
<td>$O(d(v))$</td>
<td>$O(d(v))$</td>
<td>$O(d(v))$</td>
</tr>
<tr>
<td>array of edges</td>
<td>$O(E)$</td>
<td>$O(E)$</td>
<td>$O(E)$</td>
<td>$O(E)$</td>
</tr>
</tbody>
</table>

Overall, adjacency lists tend to be ideal: real digraphs tend to be sparse (large $V$, small average $d(v)$); algorithms often iterate over $v$’s edges.
- Is there a directed path from $s$ to $t$? (transitive closure)
- What is the shortest path from $s$ to $t$? (shortest path search)
- Are all vertices mutually reachable? (strong connectivity)

- How can I organise a set of tasks? (topological sort)
- How can I crawl the web? (graph traversal)
- Which web pages are important? (PageRank)
Is it a tree? Is it a graph?
No: it’s a DAG, a directed acyclic graph.

Tree-like: each vertex has ‘children’.
Graph-like: a child vertex may have multiple parents.
The most common application of a DAG is topological sorting: ordering vertices such that, for any vertices $u$ and $v$, if $u$ has a directed edge to $v$, then $v$ comes after $u$ in the ordering.
NOT EXAMINABLE (and not taught until ’4128)

The most common application of a DAG is topological sorting:
ordering vertices such that, for any vertices $u$ and $v$,
if $u$ has a directed edge to $v$, then $v$ comes after $u$ in the ordering.

Computable with a DFS, tracking post-order sequence:
vertices only added after their children have been visited
$\Rightarrow$ a valid topological ordering
The most common application of a DAG is topological sorting: ordering vertices such that, for any vertices \( u \) and \( v \), if \( u \) has a directed edge to \( v \), then \( v \) comes after \( u \) in the ordering.

Computable with a DFS, tracking post-order sequence: vertices only added after their children have been visited \( \Rightarrow \) a valid topological ordering

dependency problems: *make*(1), spreadsheets
version-control systems: Git, Fossil, etc.
Mostly the same algorithms as for undirected graphs:
DFS and BFS should all Just Work

e.g., Web crawling: visit every page on the web.
BFS with implicit graph;
on visit, scans page for content, keywords, links
... assumption: www is fully connected.
Weighted Graphs
Some applications require us to consider a cost or weight assigned to a relation between two nodes.

In a **weighted graph**, each edge \((s, t, w)\) has a weight \(w\).

Weights can be used in both directed and undirected graphs.
Example: Major airline routes in Australia
Adjacency matrix:
- store *weight* in each cell, not just true/false.
- need some “no edge exists” value: zero might be a valid weight.

Adjacency list
- add weight to each list node

Edge list:
- add weight to each edge

Works for directed and undirected graphs!
**Weighted Graphs**

Implementation: Adjacency Matrix

- **Unweighted, undirected:**
  \[
  \begin{bmatrix}
  0 & 1 & 1 & 1 & 0 \\
  1 & 0 & 1 & 0 & 0 \\
  1 & 1 & 0 & 1 & 1 \\
  1 & 0 & 1 & 0 & 1 \\
  0 & 0 & 1 & 1 & 0
  \end{bmatrix}
  \]

- **Weighted, undirected:**
  \[
  \begin{bmatrix}
  - & 0.2 & 0.4 & 0.5 & - \\
  0.2 & - & 0.5 & - & - \\
  0.4 & 0.5 & - & 0.1 & 0.1 \\
  0.5 & - & 0.1 & - & 0.9 \\
  - & - & 0.1 & 0.9 & -
  \end{bmatrix}
  \]
Weighted directed graph:

Weighted Digraph

Adjacency Matrix

0 1 2 3 4
--- --- --- --- ---
0 * 0.2 0.4 * *
1 * 0.3 0.6 * *
2 * 0.5 * 0.1 *
3 0.5 * * * 0.9
4 * * 0.1 * *
Weighted directed graph:

Weighted Digraph

Adjacency Lists
Weighted directed graph:

**Weighted Digraph**

**Edge List**

- 0 → 1: 0.2
- 1 → 3: 0.6
- 1 → 4: 0.5
- 2 → 3: 0.9
- 2 → 4: 0.1
- 3 → 0: 0.4
- 3 → 1: 0.1
- 4 → 2: 0.3
- 4 → 0: 0.1

Implementation: Array of Edges
https://forms.office.com/r/aPF09YHZ3X