CSE 332: Graphs
Richard Anderson
Spring 2016

Announcements
• This week and next week – Graph Algorithms
• Reading, Monday and Wednesday, Weiss 9.1-9.3
• Guest lecture, Paul Beame

Graphs
• A formalism for representing relationships between objects

\[ G = (V, E) \]

- Set of vertices:
  \[ V = \{v_1, v_2, \ldots, v_n\} \]

- Set of edges:
  \[ E = \{e_1, e_2, \ldots, e_m\} \]
  where each \( e_i \) connects one vertex to another \( (v_j, v_k) \)

For directed edges, \( (v_j, v_k) \) and \( (v_k, v_j) \) are distinct. (More on this later…)

Examples of Graphs
For each, what are the vertices and edges?

• The web
• Facebook
• Highway map
• Airline routes
• Call graph of a program
• …

Directed Graphs
In directed graphs (a.k.a., digraphs), edges have a direction:

Thus, \( (u, v) \in E \) does not imply \( (v, u) \in E \).
I.e., \( v \) adjacent to \( u \) does not imply \( u \) adjacent to \( v \).

In-degree of a vertex: number of inbound edges.
Out-degree of a vertex: number of outbound edges.
**Undirected Graphs**

In **undirected** graphs, edges have no specific direction (edges are always two-way):

Thus, \((u, v) \in E\) does imply \((v, u) \in E\). Only one of these edges needs to be in the set; the other is implicit.

**Degree** of a vertex: number of edges containing that vertex. (Same as number of adjacent vertices.)

---

**Weighted Graphs**

Each edge has an associated weight or cost.

---

**Paths and Cycles**

- A **path** is a list of vertices \(\{w_1, w_2, \ldots, w_q\}\) such that \((w_i, w_{i+1}) \in E\) for all \(1 \leq i < q\)
- A **cycle** is a path that begins and ends at the same node

**Path Length and Cost**

- **Path length**: the number of edges in the path
- **Path cost**: the sum of the costs of each edge

---

**Simple Paths and Cycles**

A **simple path** repeats no vertices (except that the first can also be the last):

- \(P = \{\text{Seattle, Salt Lake City, San Francisco, Dallas}\}\)
- \(P = \{\text{Seattle, Salt Lake City, Dallas, San Francisco, Seattle}\}\)

A **cycle** is a path that starts and ends at the same node:

- \(P = \{\text{Seattle, Salt Lake City, Dallas, San Francisco, Seattle}\}\)
- \(P = \{\text{Seattle, Salt Lake City, Seattle, San Francisco, Seattle}\}\)

A **simple cycle** is a cycle that is also a simple path (in undirected graphs, no edge can be repeated).

---

**Paths/Cycles in Directed Graphs**

Consider this directed graph:

Is there a path from A to D?
Does the graph contain any cycles?
Undirected Graph Connectivity

Undirected graphs are *connected* if there is a path between any two vertices:

- Connected graph
- Disconnected graph

A *complete undirected* graph has an edge between every pair of vertices:

(Complete = *fully connected*)

Directed Graph Connectivity

Directed graphs are *strongly connected* if there is a path from any one vertex to any other.

Directed graphs are *weakly connected* if there is a path between any two vertices, ignoring direction.

A *complete directed* graph has a directed edge between every pair of vertices.

(Again, complete = *fully connected*)

Trees as Graphs

A tree is a graph that is:
- undirected
- acyclic
- connected

Hey, that doesn’t look like a tree!

Rooted Trees

We are more accustomed to:
- Rooted trees (a tree node that is "special")
- Directed edges from parents to children (parent closer to root).

A rooted tree (root indicated in red) drawn two ways

Rooted tree with directed edges from parents to children.

Characteristics of this one?

Directed Acyclic Graphs (DAGs)

DAGs are directed graphs with no (directed) cycles.

Aside: *If program call-graph is a DAG, then all procedure calls can be inlined*

|E| and |V|

How many edges |E| in a graph with |V| vertices?

What if the graph is directed?

What if it is undirected and connected?

Can the following bounds be simplified?
- Arbitrary graph: $O(|E| + |V|)$
- Arbitrary graph: $O(|E| + |V|^2)$
- Undirected, connected: $O(|E| \log|V| + |V| \log|V|)$

Some (semi-standard) terminology:
- A graph is *sparse* if it has $O(|V|)$ edges (upper bound).
- A graph is *dense* if it has $\Theta(|V|^2)$ edges.
What’s the data structure?

• Common query: which edges are adjacent to a vertex

Representation 1: Adjacency Matrix

A \(|V| \times |V|\) matrix \(M\) in which an element \(M[u, v]\) is true if and only if there is an edge from \(u\) to \(v\)

Runtimes:
Iterate over vertices?
Iterate over edges?
Iterate edges adj. to vertex?
Existence of edge?

Space requirements?
Best for what kinds of graphs?

Adjacency matrix:

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

Adjacency list:

A
B
C
D

Representing Undirected Graphs

What do these reps look like for an undirected graph?

Adjacency matrix:

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

Adjacency list:

A
B
C
D

Application: Topological Sort

Given a graph, \(G = (V, E)\), output all the vertices in \(V\) sorted so that no vertex is output before any other vertex with an edge to it.

Some Applications:

Bus Routes in Downtown Seattle

If we’re at 3rd and Pine, how can we get to 1st and University using Metro?
How about 4th and Seneca?

Application: Topological Sort

CSE 322
CSE 323
CSE 413
CSE 423
CSE 453
CSE 457
CSE 467
CSE 306
CSE 413
CSE 342
CSE 343
CSE 379
CSE 413
CSE 467

What kind of input graph is allowed?
Is the output unique?
Topological Sort: Take One

1. Label each vertex with its in-degree (# inbound edges)
2. While there are vertices remaining:
   a. Choose a vertex \( v \) of in-degree zero; output \( v \)
   b. Reduce the in-degree of all vertices adjacent to \( v \)
   c. Remove \( v \) from the list of vertices

Runtime:

```
void Graph::topsort(){
    Vertex v, w;
    labelEachVertexWithItsInDegree();
    for (int counter=0; counter < NUM_VERTICES; counter++){
        v = findNewVertexOfDegreeZero();
        v.topologicalNum = counter;
        for each w adjacent to v
            w.indegree--;
    }
}
```

Topological Sort: Take Two

1. Label each vertex with its in-degree
2. Initialize a queue \( Q \) to contain all in-degree zero vertices
3. While \( Q \) not empty
   a. \( v = Q.dequeue(); \) output \( v \)
   b. Reduce the in-degree of all vertices adjacent to \( v \)
   c. If new in-degree of any such vertex \( u \) is zero
      \( Q.enqueue(u) \)

Note: could use a stack, list, set, box, ... instead of a queue

Runtime:

```
void Graph::topsort(){
    Queue q(NUM_VERTICES);
    int counter = 0;
    Vertex v, w;
    labelEachVertexWithItsInDegree();
    q.makeEmpty();
    for each vertex v
        if (v.indegree == 0)
            q.enqueue(v);
    while (!q.isEmpty()){  
        v = q.dequeue();
        v.topologicalNum = ++counter;
        for each w adjacent to v
            if (--w.indegree == 0)
                q.enqueue(w);
    }
}
```

Find a topological order for the following graph
If a graph has a cycle, there is no topological sort

Consider the first vertex on the cycle in the topological sort. It must have an incoming edge.

Lemma: If a graph is acyclic, it has a vertex with in degree 0

Proof:
Pick a vertex \( v_1 \), if it has in-degree 0 then done.
If not, let \( (v_2, v_1) \) be an edge, if \( v_2 \) has in-degree 0 then done.
If not, let \( (v_3, v_2) \) be an edge . . .
If this process continues for more than \( n \) steps, we have a repeated vertex, so we have a cycle.