CSE 332: Graphs
Richard Anderson, Steve Seitz
Winter 2014

Announcements (2/12/14)
• Exams
• Return at end of class
• Mean 62.5, Median 63, sd 7.2
• HW 5 available
• Project 2B due Thursday night
• Reading for this lecture: Chapter 9.

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 \( \{w_1, w_2, ..., w_q\} \) 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):

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

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

A 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

### \(|E| \text{ and } |V|\)

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 2: Adjacency List

A list (array) of length $|V|$ in which each entry stores a list (linked list) of all adjacent vertices

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

Space requirements?

Best for what kinds of graphs?

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?

Representing Undirected Graphs

What do these reps look like for an undirected graph?

Some Applications: Moving Around Washington

What’s the shortest route to from Seattle to Pullman?

Some Applications: Moving Around Washington

What’s the quickest way to get from Seattle to Pullman?
Some Applications: Reliability of Communication

If Wenatchee’s phone exchange goes down, can Seattle still talk to Pullman?

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

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.

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

Topological Sort: Take One

What kind of input graph is allowed? Is the output unique?

void Graph::topsort()

142
143
321
341
378
370
322
326
c 303
403
421
451
457
467
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:

```cpp
void Graph::topsort()
{
    Queue q(NUM_VERTICES);
    int counter = 0;
    Vertex v, w;
    labelEachVertexWithItsIn-degree();
    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);
    }
}
```