Today’s Outline

- Admin
  - Project 3 in-progress check-in due tonight!

- Graph Algorithms
  - Representation
  - Applications
  - Topological sort

A Great Mathematician

- Number theory
- Numerical Analysis
- Graph Theory
- Physical sciences

- See the History of Mathematics biography on Euler:
  - http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Euler.html

Leonhard Euler 1707-1783

The Bridges of Königsberg

- Can we walk around Königsberg, crossing each bridge exactly once?

The (Graffitied) Bridges of Königsberg

- Each part of town is a vertex
- Each bridge is an edge
- Eulerian path: a path that visits each edge exactly once

Does there exist an Eulerian path in the graph?
Graph... ADT?

- Not quite an ADT... operations not clear
- A formalism for representing relationships between objects

```
Graph G = (V, E)
- Set of vertices: V = {v_1, v_2, ..., v_n}
- Set of edges: E = {e_1, e_2, ..., e_m}
where each e_i connects two vertices (v_{i1}, v_{i2})
```

Graph Definitions

In directed graphs, edges have a specific direction:

```
V = (Han, Leia, Luke)
E = (Luke, Leia), (Han, Leia)
```

In undirected graphs, they don’t (edges are two-way):

```
v is adjacent to u if (u, v) ∈ E
```

More Definitions: Simple Paths and Cycles

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

- p = (Seattle, Salt Lake City, San Francisco, Dallas)
- p = (Seattle, Salt Lake City, Dallas, San Francisco, Seattle)

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

- p = (Seattle, Salt Lake City, Dallas, San Francisco)
- p = (Seattle, Salt Lake City, Seattle, San Francisco, Seattle)

A simple cycle is a cycle that repeats no vertices except that the first vertex is also the last (in undirected graphs, no edge can be repeated)

Trees as Graphs

- Every tree is a graph!
- Not all graphs are trees!

A graph is a tree if

- There are no cycles (directed or undirected)
- There is a path from the root to every node

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

Graph Representations

0. List of vertices + list of edges
1. 2-D matrix of vertices (marking edges in the cells) “adjacency matrix”
2. List of vertices each with a list of adjacent vertices “adjacency list”

Things we might want to do:
- iterate over vertices
- iterate over edges
- iterate over vertices adj. to a vertex
- check whether an edge exists
Representation 1: Adjacency Matrix

A $|V| \times |V|$ array in which an element $(u, v)$ is true if and only if there is an edge from $u$ to $v$.

Representation 2: Adjacency List

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

Some Applications:
Moving Around Washington

What’s the shortest way to get from Seattle to Pullman?
Edge labels:

Some Applications:
Communication in Washington

What’s the cheapest inter-city public network?
Edge labels:

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?

Food for Thought

Before next class, think how you would efficiently solve each of the preceding problems!

More Applications:
Orderings and Dependencies

Okay, everybody, get up and stretch! Pretend as if you were about to leave!!

What actions did you take?

Ordering of Actions:
Dependency Graph

Let’s order the actions by what you did first:

Ordering on Graphs:
Total and Partial

A → B
意味着 A 必须在 B 之前

Application: Topological Sort

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

Is the output unique?
Topological Sort: Take One

1. Label each vertex with its in-degree (# of 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:*

---

Graph Traversals

- Breadth-first search (and depth-first search) work for arbitrary (directed or undirected) graphs - not just mazes!
  - Must mark visited vertices so you do not go into an infinite loop!
- Either can be used to determine connectivity:
  - Is there a path between two given vertices?
  - Is the graph (weakly) connected?
- Which one:
  - Uses a queue?
  - Uses a stack?
  - Always finds the shortest path (for unweighted graphs)?

---

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. \( v = Q\text{.dequeue}() \); 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 \( Q\text{.enqueue}(u) \)

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

*Runtime:*

---

Graph Connectivity

- Undirected graphs are *connected* if there is a path between any two vertices
- 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 graph has an edge between every pair of vertices

---

To Do

- Project 3: in-progress turnin tonight!
- Think of solutions to problems posed in today’s lecture
- Read Chapter 9, sections 9.1-9.3