CSE 326: Data Structures
Topic 13: Cool Graphs n' Pretty Pictures

Luke McDowell

Summer Quarter 2003

RadixSort Answers

- Input: 95, 3, 927, 187, 604, 823, 805, 422, 159, 98, 123, 3, 987, 125

BinSort on lowest digit:

		422	3 123 823 3	604	125 805 95		987 187 927	98	159
0	1	2	3	4	5	6	7	8	9

BinSort on next-higher digit:

805 604 3 3		927 125 123 823 422			159			987 187	98 95
0	1	2	3	4	5	6	7	8	9

BinSort on highest digit:

98 95 3 3	187 159 125 123			422		604		823 805	987 927
0	1	2	3	4	5	6	7	8	9

Famous Dead Guy

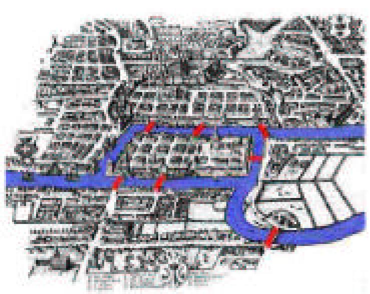
- Number theory
- Numerical Analysis
- Graph Theory
- See the History of Mathematics biography on Euler:
  - http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Euler.html

Leonhard Euler 1707-1783

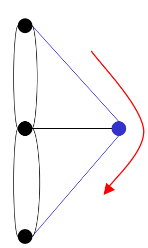
The Bridges of Königsberg

Can we walk around Konigsberg, crossing each bridge exactly once?

The (Graffitied) Bridges of Königsberg



The Bridges of Königsberg, Formally

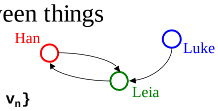


- Each bridge is an edge
- Each part of town is a vertex
- Is there a path that crosses each edge exactly once?

Graph… ADT?

Graphs are a formalism useful for representing relationships between things

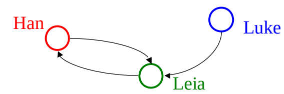
- A graph **G** is represented as **G = (V, E)**
  - **V** is a set of vertices: $\{v_1, v_2, \ldots, v_n\}$
  - **E** is a set of edges: $\{e_1, e_2, \ldots, e_m\}$ where each e_i connects two vertices (v_{i1}, v_{i2})



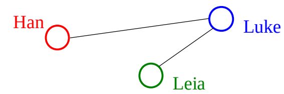

```
V = {Han, Leia, Luke}
E = {(Luke, Leia),
     (Han, Leia),
     (Leia, Han)}
```

Graph Definitions

In *directed* graphs, edges have a specific direction:



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



Vertices **u** and **v** are *adjacent* 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}

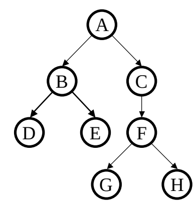
A *cycle* is a path that starts and ends at the same node:

p = {Seattle, Salt Lake City, Dallas, San Francisco, 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

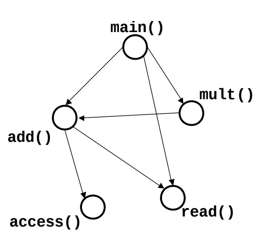
- Every tree is a graph with some restrictions:
  - The tree is *directed*
  - There are *no cycles* (directed or undirected)
  - There is a *directed path* from the root *to every node*



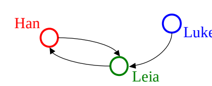
Directed Acyclic Graphs (DAGs)

DAGs are directed graphs with no cycles.

If program call-graph is a DAG, then all procedure calls can be in-lined



Graph Representations



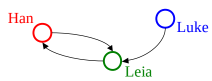
0. List of vertices + list of edges
1. 2-D matrix of vertices (marking edges in the cells)
   "adjacency matrix"
2. List of vertices each with a list of adjacent vertices
   "adjacency list"

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| x |V|` array in which an element `(u, v)` is true if and only if there is an edge from **u** to **v**



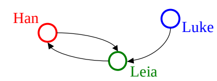
	Han	Luke	Leia
Han			
Luke			
Leia			

space requirements:

runtime:

Representation 2: Adjacency List

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

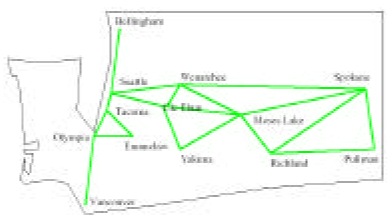


Han	
Luke	
Leia	

space requirements:

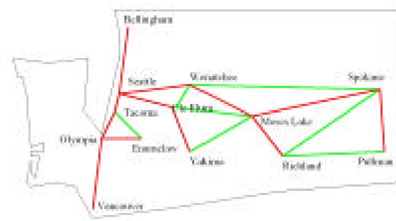
runtime:

Some Applications: Moving Around Washington



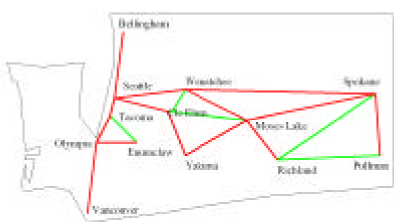
What's the fastest way from Seattle to Spokane?

Some Applications: Communication in Washington



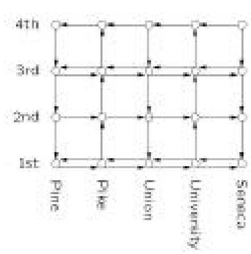
What's the cheapest inter-city network?

Some Applications: Reliability of Communication



If we lose Wenatchee, can Seattle still talk to Spokane?

Some Applications: Bus Routes in Downtown Seattle

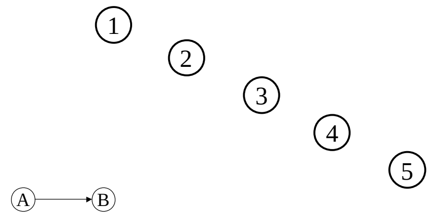


If we're at 3rd and Pine, can we get to 1st and Union?

Some Applications:
Orderings and Determining Dependencies

Okay, everybody, get up and stretch!

Total Ordering on Graphs

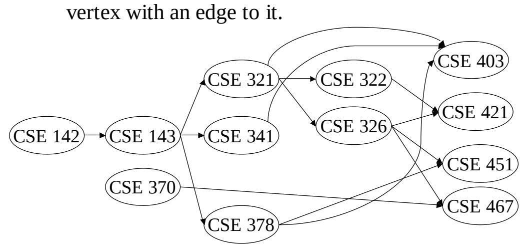


means A must go before B

Partial Order: Taking a Break in Class

Some Applications: Topological Sort

Given a 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 output unique?

Topo-Sort (Take One)

Label each vertex's *in-degree* (# of inbound edges)
While there are vertices remaining
  Pick a vertex with in-degree of zero and output it
  Reduce the in-degree of all vertices adjacent to it
  Remove it from the list of vertices

Runtime:

Topo-Sort (Take Two)

Label each vertex's in-degree
Initialize a queue to contain all in-degree zero vertices
While there are vertices remaining in the queue
  Get a vertex *v* from queue (has in-degree of zero)
  Output *v*
  Reduce the in-degree of all vertices adjacent to *v*
  Put any of these with new in-degree zero on the queue

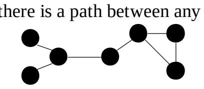
Runtime:

Graph Traversals

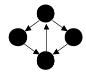
- Breadth-first search (and depth-first search) work for arbitrary (directed or undirected) graphs - not just mazes!
  - Must mark visited vertices so you do not go into an infinite loop!
- Either can be used to determine connectivity:
  - Is there a path between two given vertices?
  - Is the graph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

"Weakly connected": A detour into connectivity

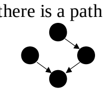
Undirected graphs are *connected* if there is a path between any two vertices



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