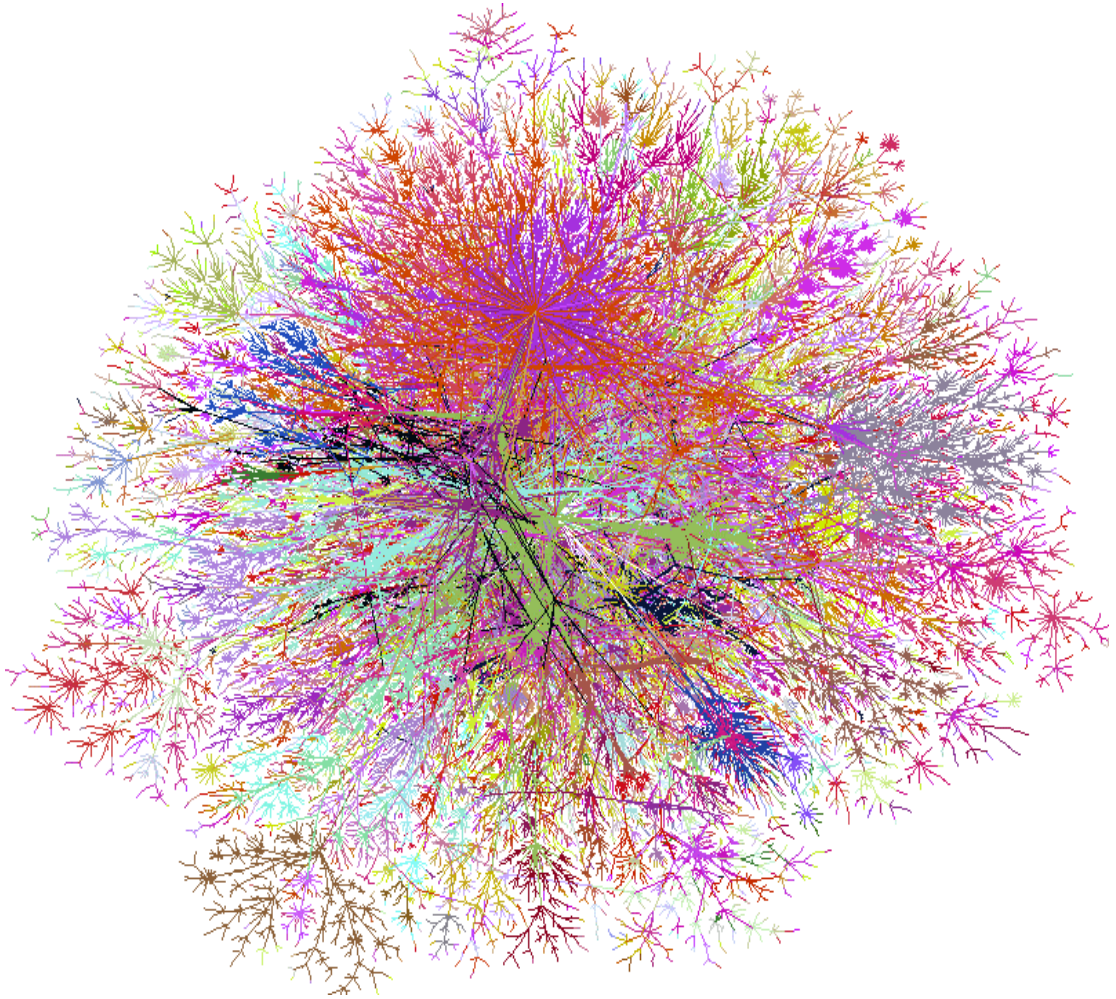
Graphs



www.cheswick.com/ches/**map**/index.html

Topic Coverage

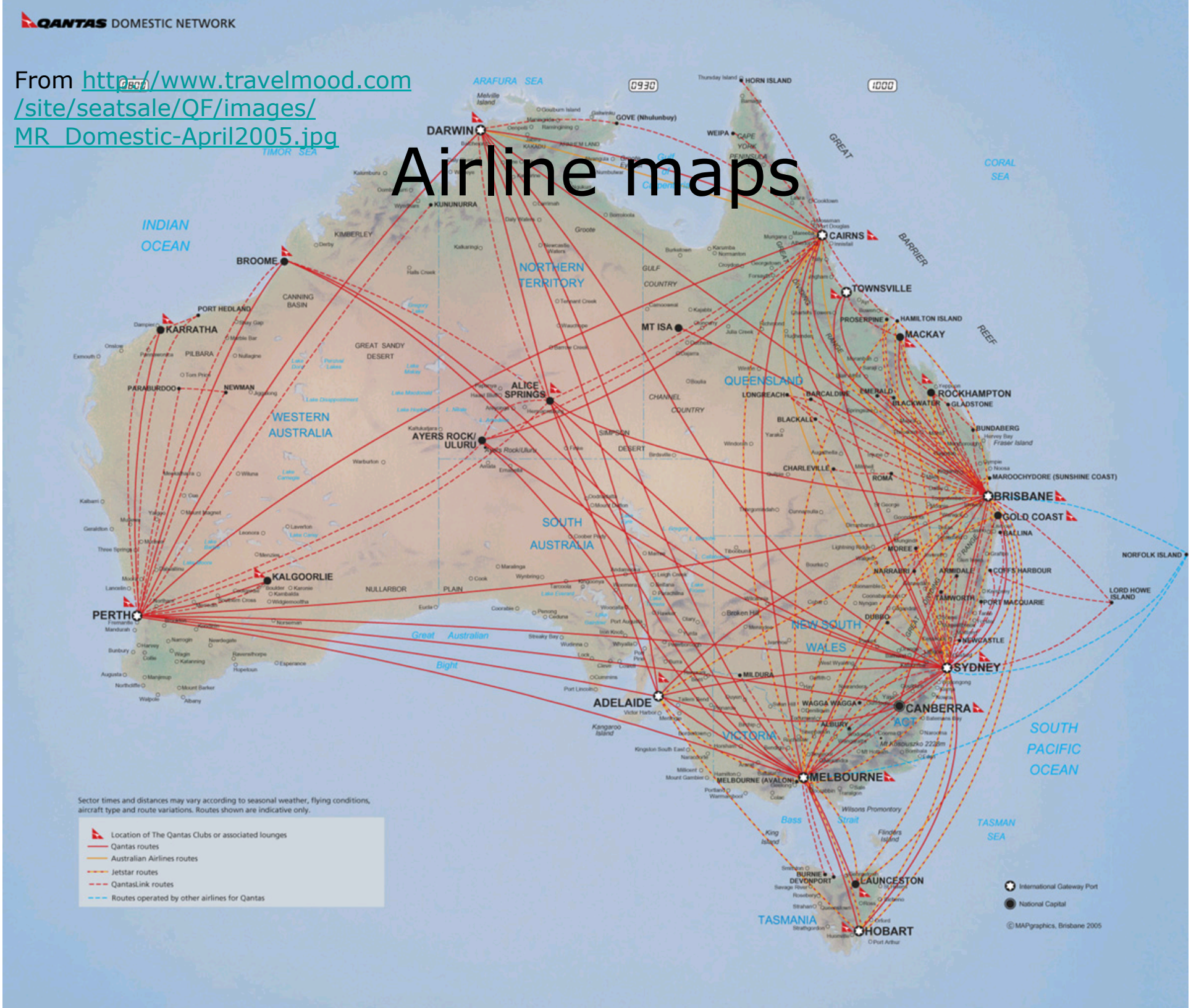
- Graph terminology
  - Lots of it!
- Ways of representing graphs and costs & benefits
  - Adjacency matrix
  - Adjacency list
- Solving problems with graphs
  - Topological sort
  - Traveling salesperson

Graph examples

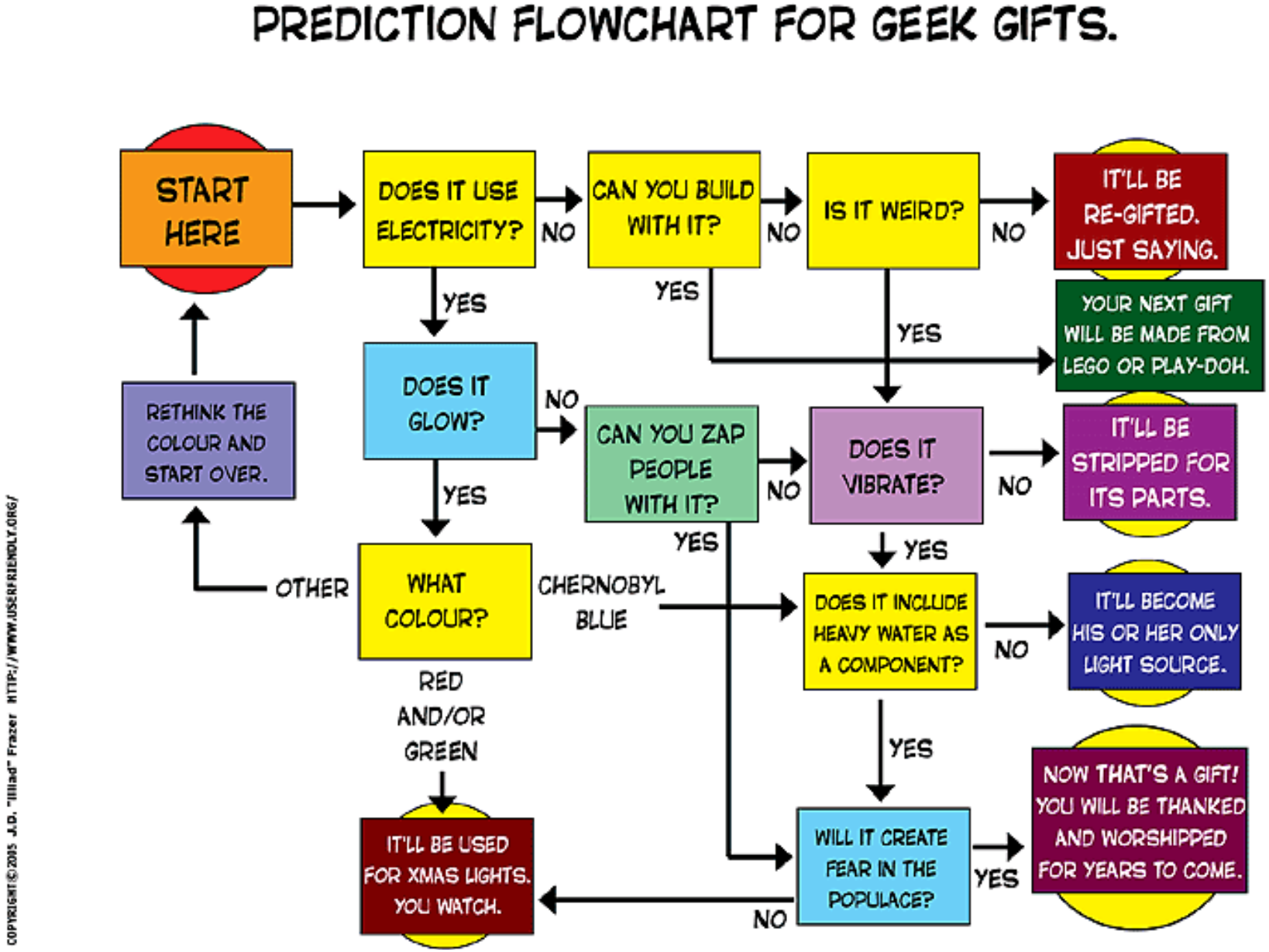
- Google Maps, of course
  - Which actually uses MapQuest data
  - But we'll call it Google Maps data
- Others?

From http://www.travelmood.com/site/seatsale/QF/images/MR_Domestic-April2005.jpg

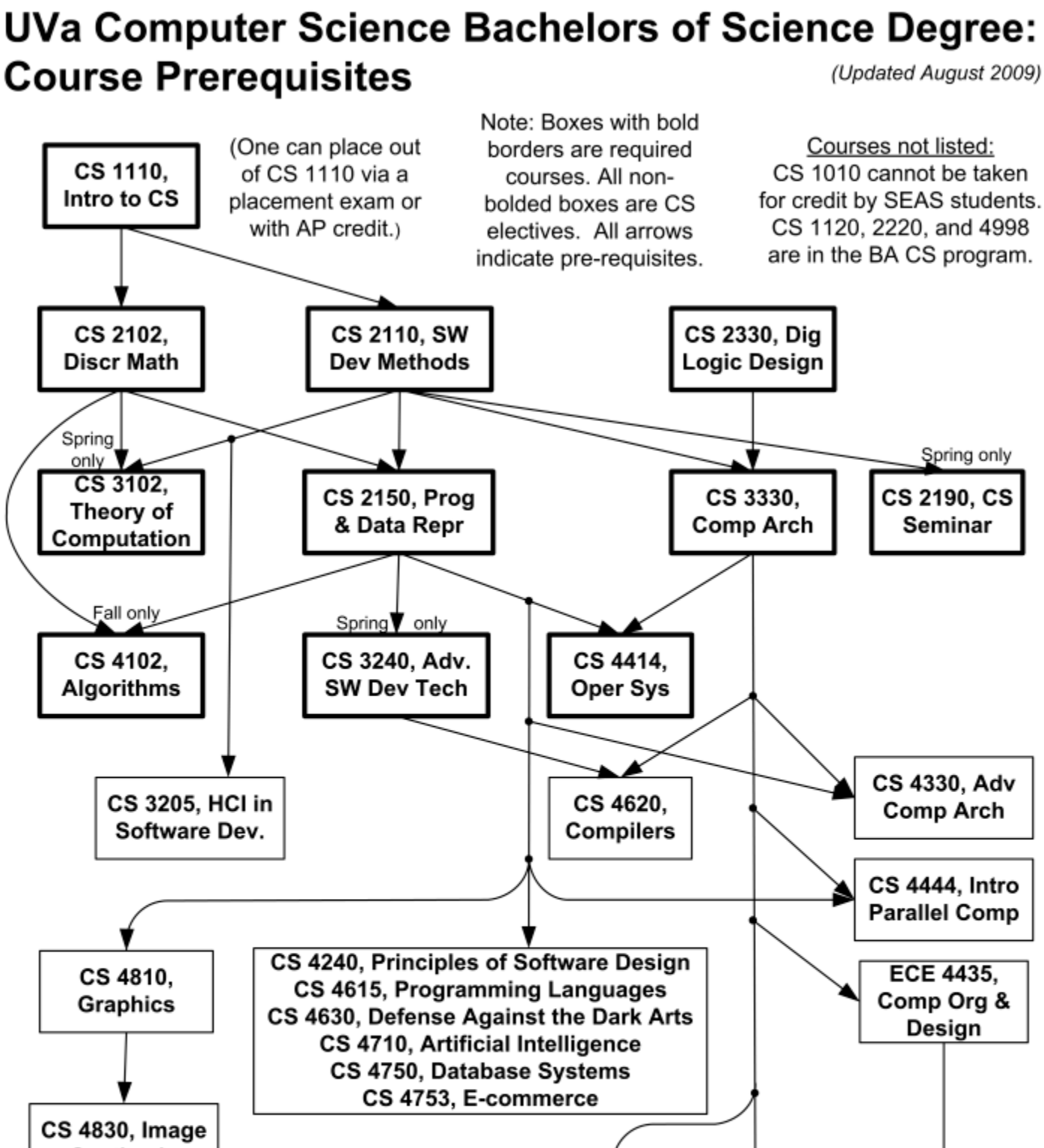
Airline maps



Flowcharts


PREDICTION FLOWCHART FOR GEEK GIFTS.
START HERE
DOES IT USE ELECTRICITY?
NO
CAN YOU BUILD WITH IT?
NO
IS IT WEIRD?
NO
IT'LL BE RE-GIFTED. JUST SAYING.
YES
YES
YOUR NEXT GIFT WILL BE MADE FROM LEGO OR PLAY-DOH.
YES
DOES IT GLOW?
NO
CAN YOU ZAP PEOPLE WITH IT?
NO
DOES IT VIBRATE?
NO
IT'LL BE STRIPPED FOR ITS PARTS.
RETHINK THE COLOUR AND START OVER.
YES
YES
YES
OTHER
WHAT COLOUR?
CHERNOBYL BLUE
DOES IT INCLUDE HEAVY WATER AS A COMPONENT?
NO
IT'LL BECOME HIS OR HER ONLY LIGHT SOURCE.
RED AND/OR GREEN
YES
IT'LL BE USED FOR XMAS LIGHTS. YOU WATCH.
WILL IT CREATE FEAR IN THE POPULACE?
YES
NOW THAT'S A GIFT! YOU WILL BE THANKED AND WORSHIPPED FOR YEARS TO COME.
NO
COPYRIGHT©2005 J.D. "Illiad" Frazer HTTP://WWW.USERFRIENDLY.ORG/

Course pre-req graphs



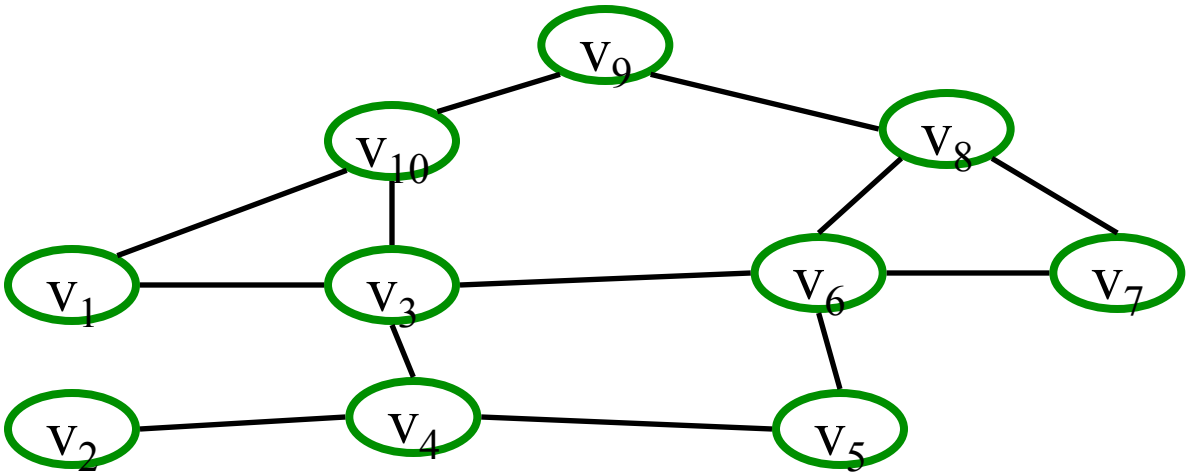
Graphs

G = (V, E)

V are the **vertices**; **E** are the **edges**.

Edges are of the form **(v, w)**, where v, w ∈V.

- ordered pair: directed graph or digraph
- unordered pair: undirected graph



How big are these graphs?

- These are educated guesses, by the way

- Airport codes
  - There are probably 3,000 world-wide airports
  - Assume you can fly to 25 airports from each
  - That's 3,000*25 = 75,000 edges

- Google maps
  - There are probably 30 million vertices in the US
  - Assume each one connects to three others
  - That's 3 * 30 million ≈ 100 million edges

Terminology

- A **weight** or **cost** can be associated with each edge.
- w is **adjacent** to v iff $(v, w) \in E$.
- **path**: sequence of vertices $w_1, w_2, w_3, \ldots, w_N$ such that $(w_i, w_{i+1}) \in E$ for $1 \leq i < N$.
- **length** of a path: number of edges in the path.
- **simple path**: all vertices are distinct.

How to weight a graph…

- For Google maps?
- For airline routes?

More terminology

cycle:

directed graph: path of length ≥ 1 such that $w_1 = w_N$.

undirected graph: same, except all edges are distinct.

connected: there is a path from every vertex to every other vertex.

loop: $(v, v) \in E$.

complete graph: there is an edge between every pair of vertices.

End of lecture on Mon, Nov 16

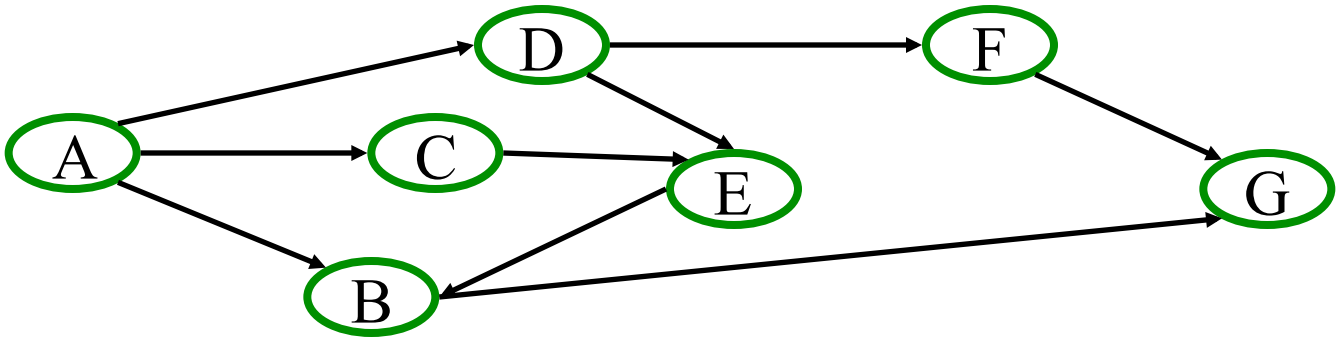
- Also went over the slides 32-47 (end) of 12-huffman

Digraph terminology

directed acyclic graph: no cycles. "DAG"

strongly connected: there is a path from every vertex to every other vertex.

weakly connected: the underlying undirected graph is connected.

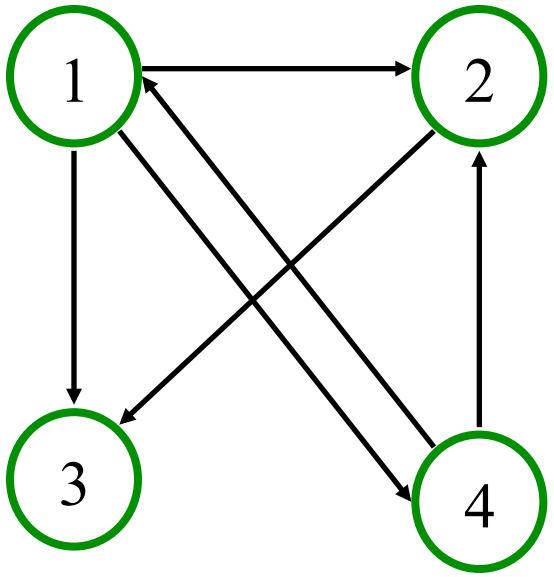


Representation

- adjacency **matrix**:

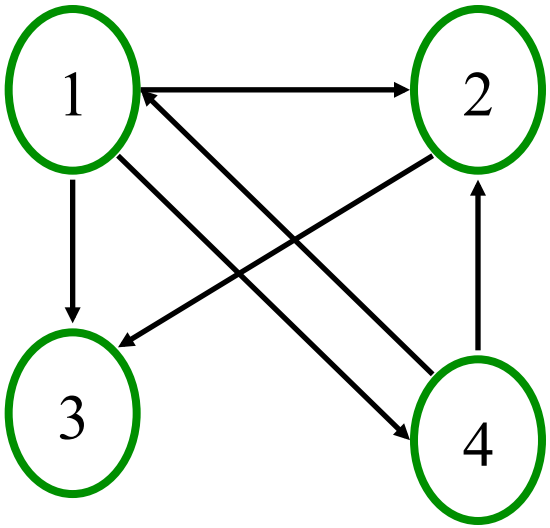
$$A[u][v] = \begin{cases} \text{weight} & , \text{if } (u, v) \in E \\ 0 & , \text{if } (u, v) \notin E \end{cases}$$

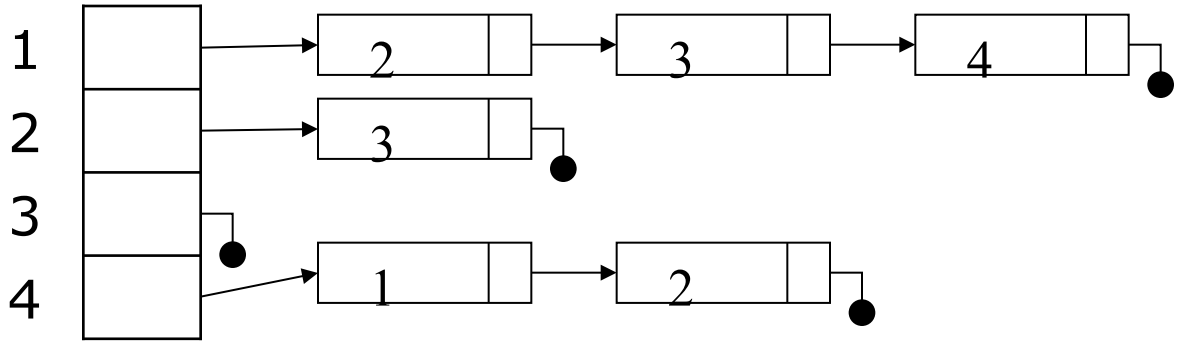
	1	2	3	4
1				
2				
3				
4				



Representation

- adjacency **list:**





Representation in the real world

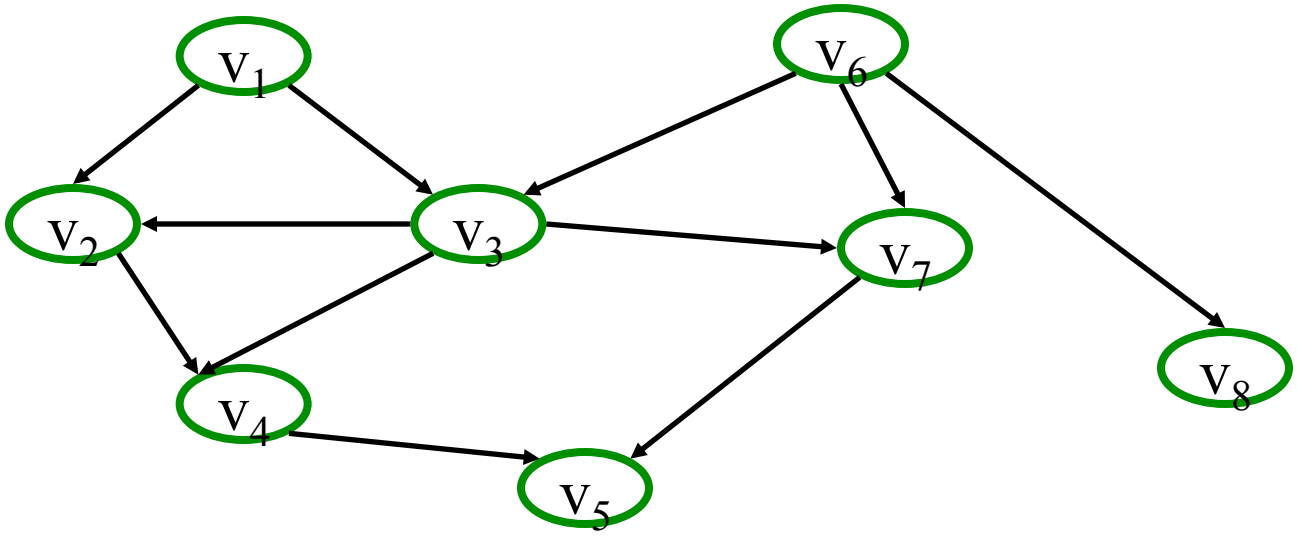
- Two types of representation
  - Adjacency matrix
  - Adjacency graph

- How does Google maps probably store it?
- How do airline routes probably store it?

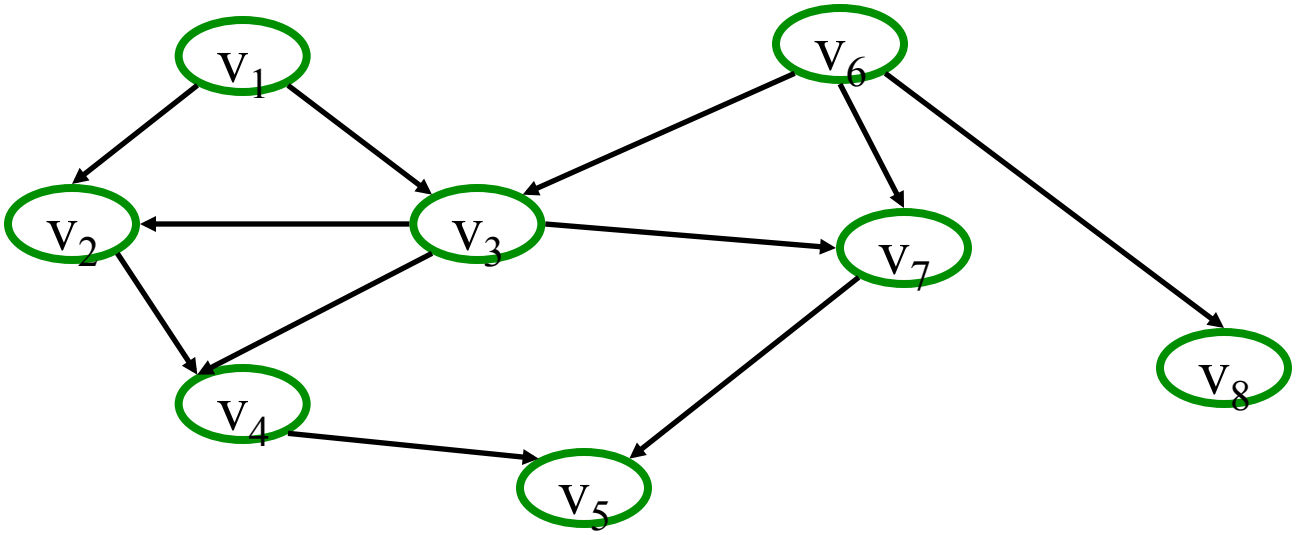
Topological Sort

Topological Sort

- Given a directed acyclic graph, construct an **ordering** of the vertices such that if there is a path from v_i to v_j, then v_j appears after v_i in the ordering.
  - The result is a linear list of vertices
- **indegree** of v: # of edges (u, v).

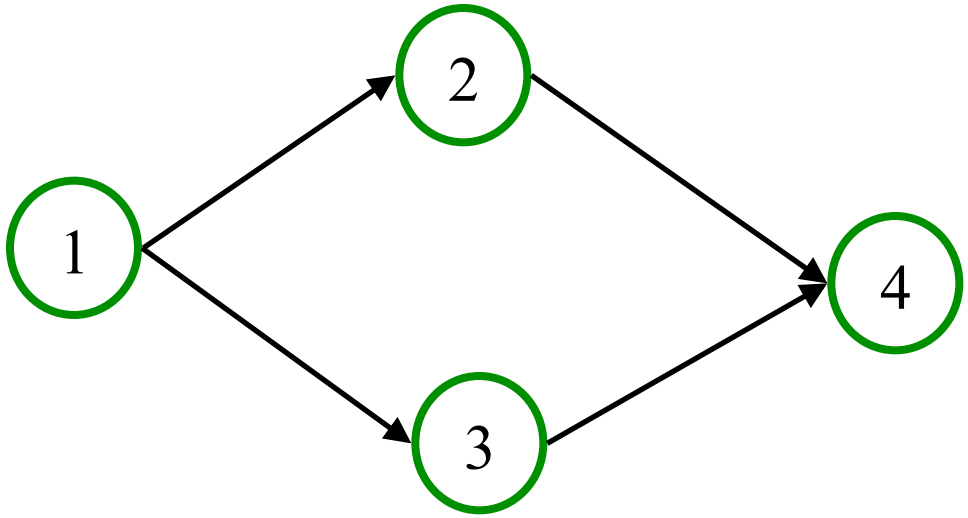


Topological Sort

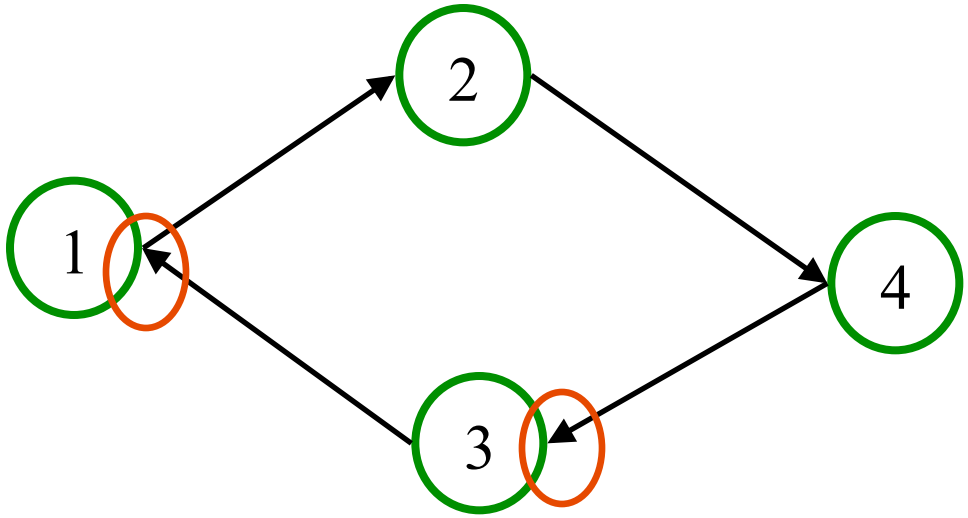


- A valid topological sort is:
  - V1, V6, V8, V3, V2, V7, V4, V5

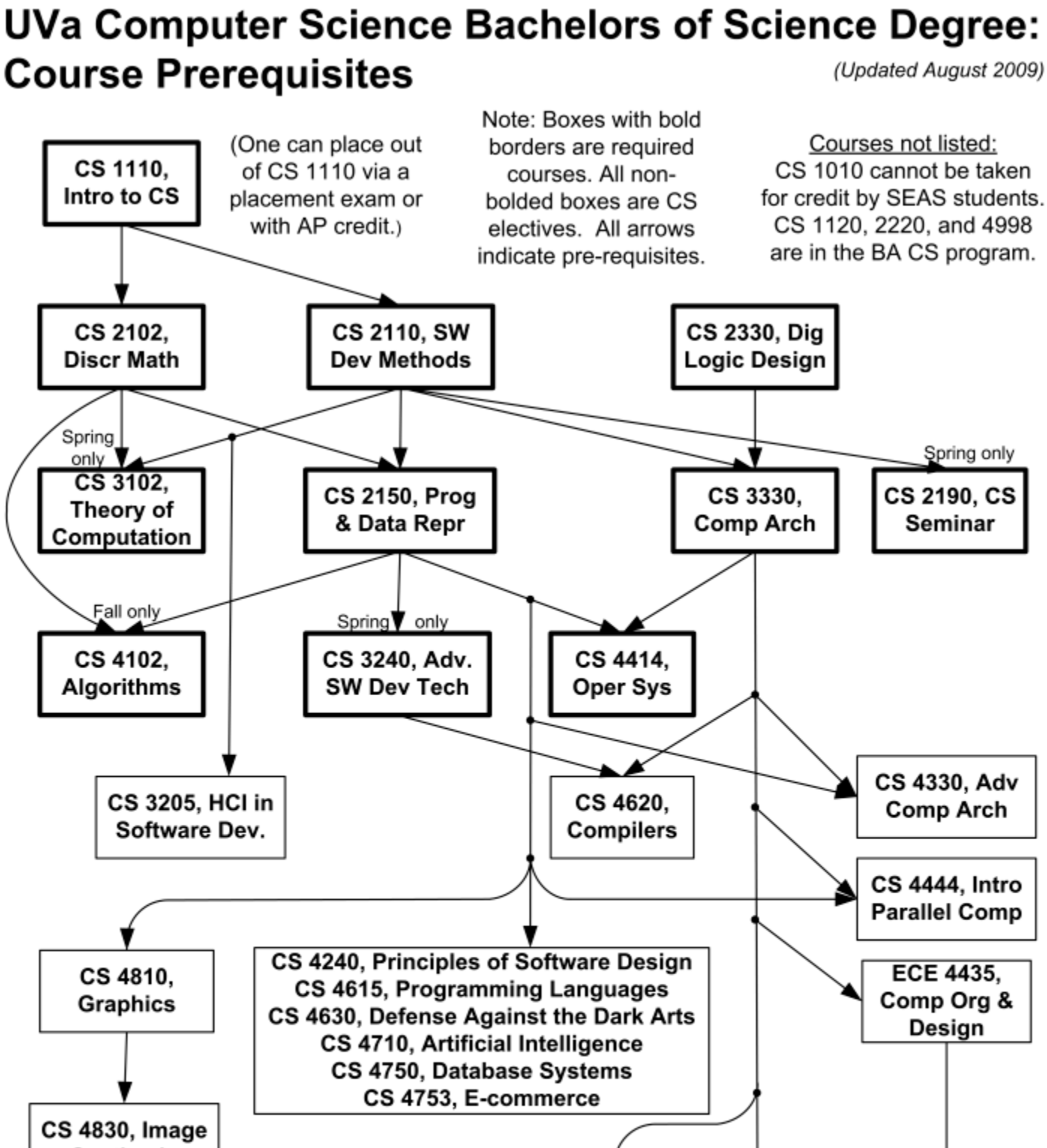
What is the topological sort?



What is the topological sort?



This is already topo-logically sorted!



Topological sort

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

- What's the big-Oh running time?
- Observation: The only new (eligible) vertices with indegree 0 are the ones adjacent to the vertex just processed.

Topological sort

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

  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);
  }
  if (counter != NUM_VERTICES)
    throw CycleFound();
}
```

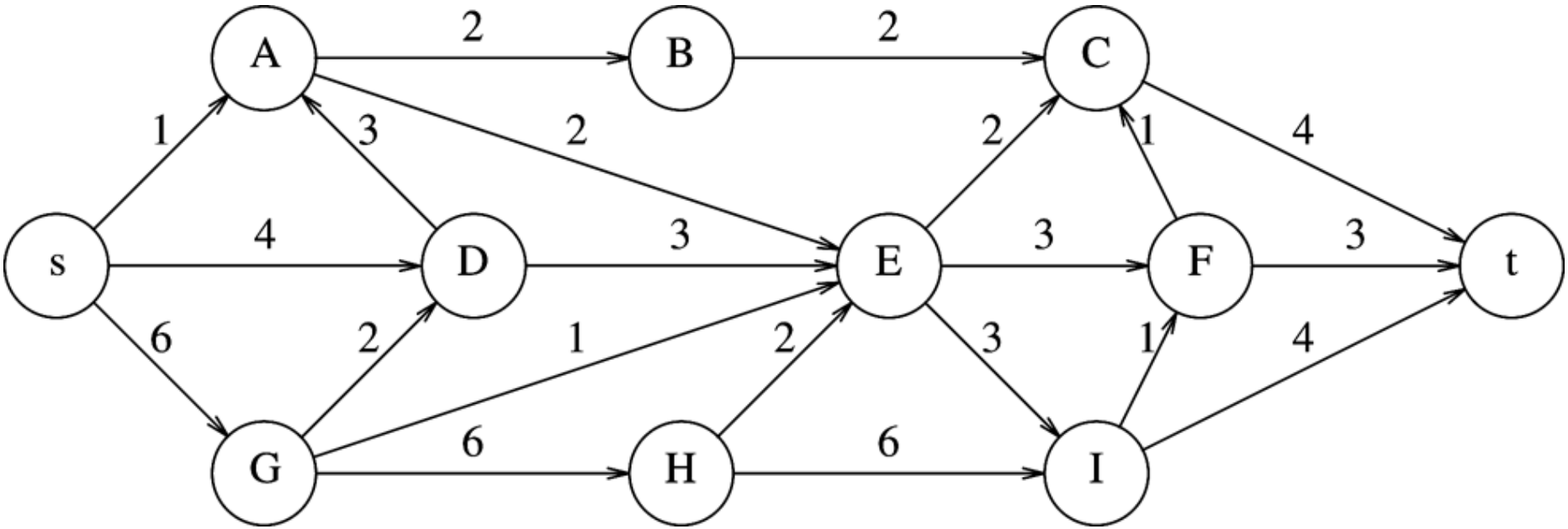
intialize the queue

get a vertex with indegree 0

insert new eligible vertices

Problem 9.1 from Textbook

- Topological sort



Shortest Path Algorithms

Unweighted and Weighted Graphs

Why do we care about shortest paths?

- The obvious answers:
  - Map routing (car navigation systems, Google Maps, flights)
  - 6 degrees of separation

- But what else?
  - Internet routing
  - Puzzle answers (Rubik's cube)

End of lecture on Wed, Nov 18

3 types of algorithms

- Single pair
- Single source
- All pairs

- And a variant that we'll see later:
  - Travelling salesperson

Shortest Path Algorithms

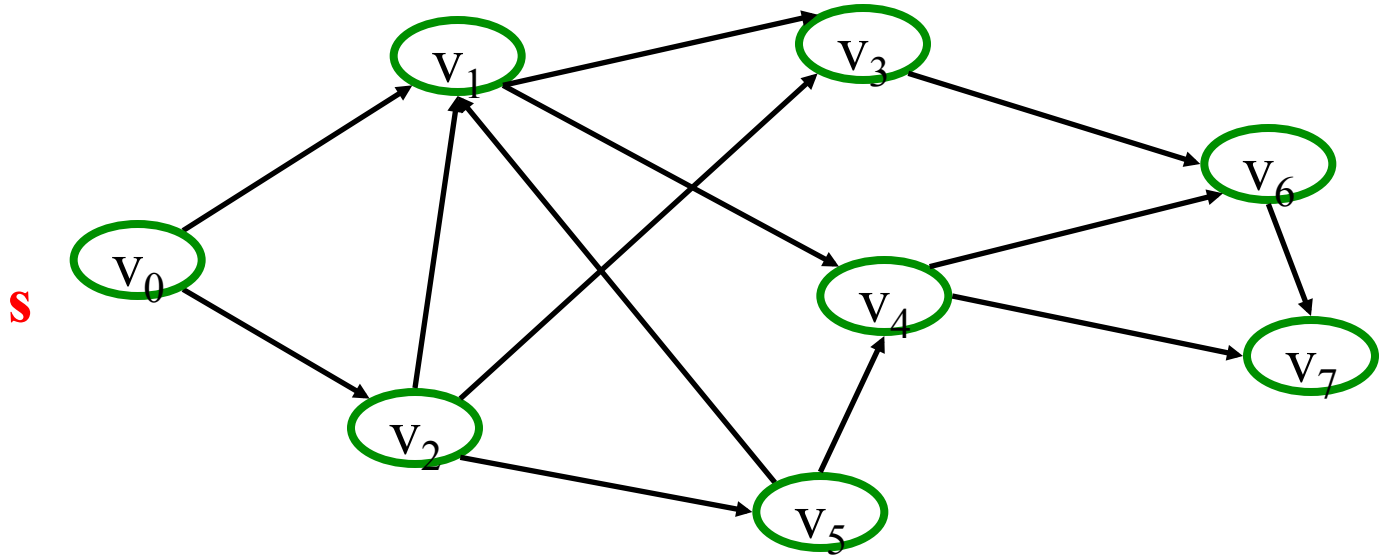
- This version is called the "single-source" shortest path
- Given a graph **G** = (V, E) and a single distinguished vertex **s**, find the shortest weighted path from **s** to every other vertex in **G**.

weighted path length of $v_1, v_2, \ldots, v_N$:

$$\sum_{i=1}^{N-1} c_{i,i+1}$$, where $\mathbf{c_{i,j}}$ is the cost of edge $(\mathbf{v_i}, \mathbf{v_j})$

Unweighted Shortest Path

- Special case of the weighted problem: all weights are 1.
- Solution: breadth-first search. Similar to level-order traversal for trees.



```
void Graph::unweighted (Vertex s){
  Queue q(NUM_VERTICES);
  Vertex v, w;
  q.enqueue(s);
  s.dist = 0;

  while (!q.isEmpty()){
    v = q.dequeue();
    for each w adjacent to v
      if (w.dist == INFINITY){
        w.dist = v.dist + 1;
        w.path = v;
        q.enqueue(w);
      }
    }
  }
```

each edge examined at most once – if adjacency lists are used

each vertex enqueued at most once

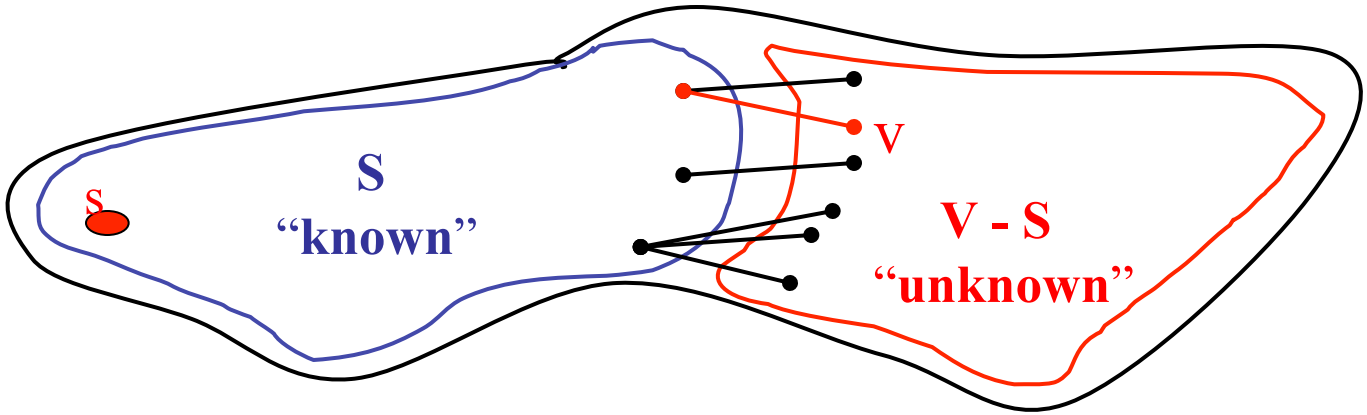
total running time: O(?)

Weighted Shortest Path

- no negative weight edges.
- **Dijkstra's algorithm**: uses similar ideas as the unweighted case.

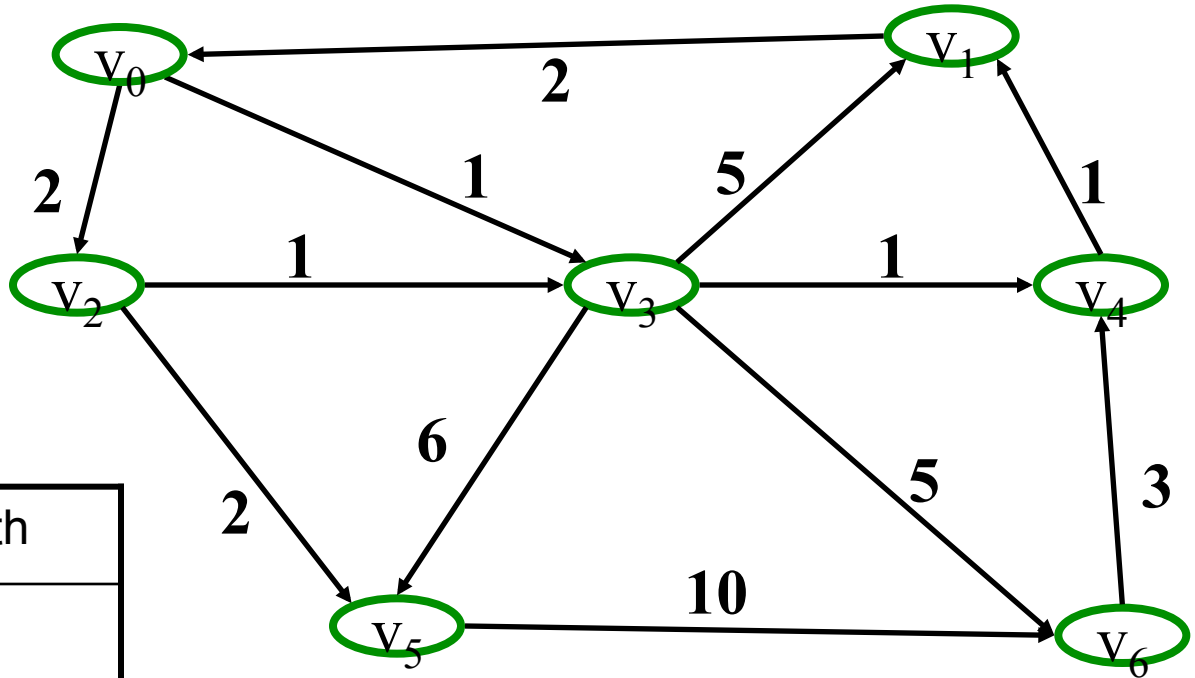
Greedy algorithms:

do what seems to be best at every decision point.



Dijkstra's algorithm

- Initialize each vertex's distance as infinity
- Start at a given vertex *s*
  - Update *s*'s distance to be 0
- Repeat
  - Pick the next unknown vertex with the shortest distance to be the next *v*
    - If no more vertices are unknown, terminate loop
  - Mark *v* as known
  - For each edge from *v* to adjacent unknown vertices *w*
    - If the total distance to *w* is less than the current distance to *w*
      - Update w's distance and the path to *w*



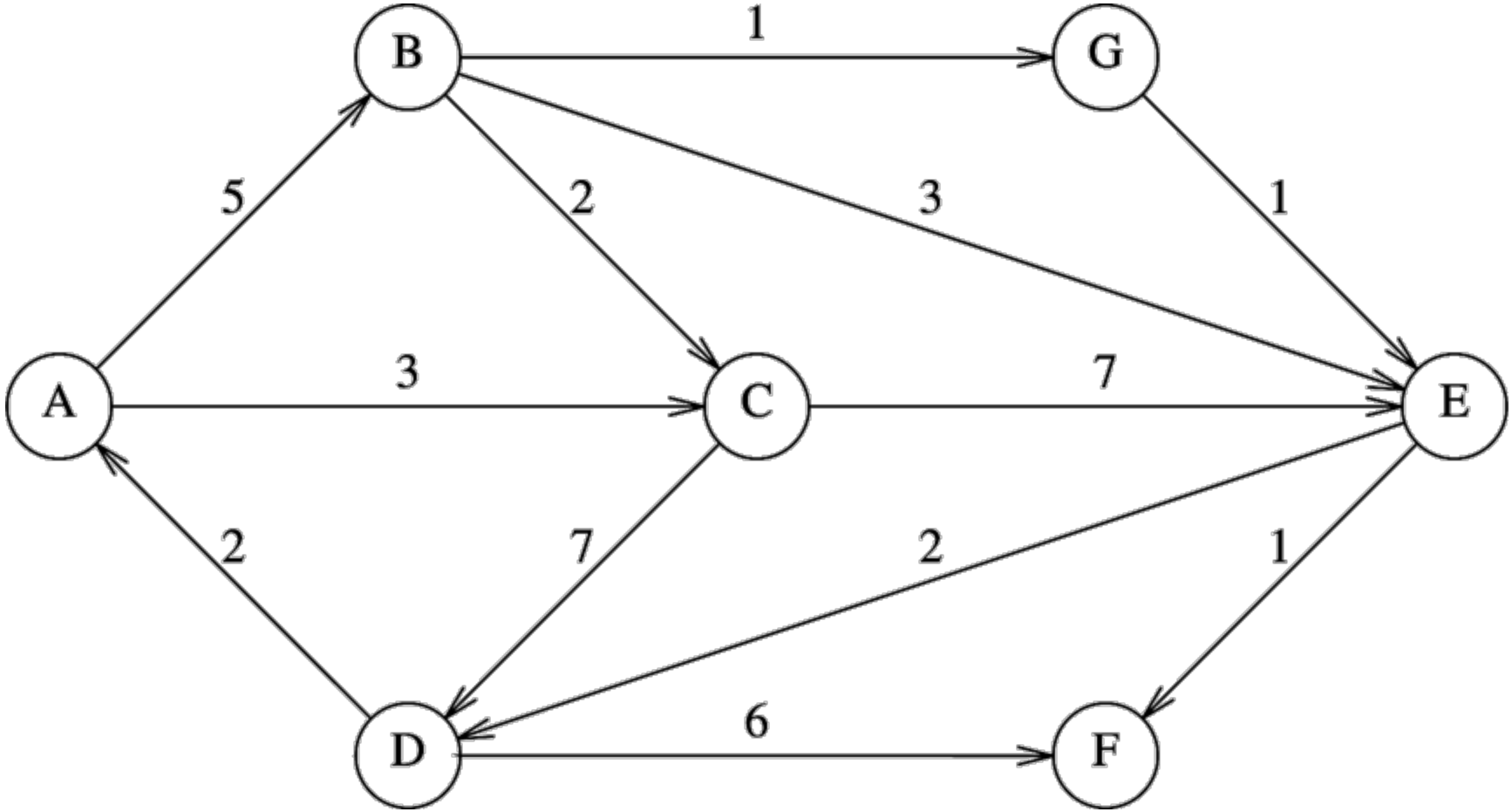
V	Known	Dist	path
v0			
v1			
v2			
v3			
v4			
v5			
v6			

```
void Graph::dijkstra(Vertex s){
  Vertex v,w;
  s.dist = 0;

  while (there exist unknown vertices, find the
         unknown v with the smallest distance)
    v.known = true;

    for each w adjacent to v
      if (!w.known)
        if (v.dist + Cost_VW < w.dist){
          w.dist = v.dist + Cost_VW;
          w.path = v;
        }
  }
}
```

Unweighted & Weighted Single Source Shortest Paths (Weiss 9.5)



Analysis

- How long does it take to find the smallest unknown distance?
  - simple scan using an array: $O(v)$
- Total running time:
  - Using a simple scan: $O(v^2+e) = O(v^2)$
- Optimizations?
  - Use adjacency graphs and heaps
  - Assuming that the graph is connected (i.e. $e > v-1$), then the running time decreases to $O(e + v \log v)$
  - We can simplify this to $O(e \log v)$
    - although we won't see how to do that here

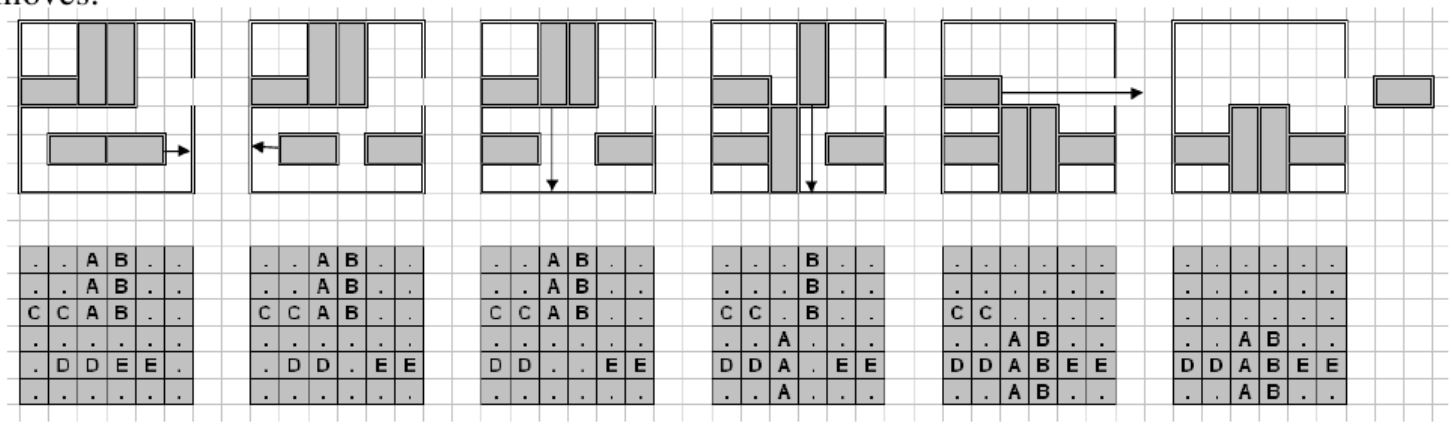
Negative Cost Edges?

- Perhaps the graph weights are the amount of fuel expended
  - Positive means fuel was used
  - And passing by a fuel station is a refueling, which is a negative cost edge
- Dijkstra's algorithm does not work for negative cost edges
  - Others do, but are much less efficient
- What about negative cost cycles?

Shortest Path Example Problem (from the ICPC Mid-Atlantic Regionals, 2009)

Problem B: Block Game

Bud bought this new board game. He is hooked. He has been playing it over and over again such that he thinks can finish the game with the minimum number of moves, but he is uncertain. He wants you to help him check whether the moves he has listed are indeed the minimum number of moves.



You are given a 6x6 board, and a set of 2x1 or 3x1 (vertical) or 1x2 or 1x3 (horizontal) pieces. You can slide the horizontal pieces horizontally only, and the vertical pieces vertically only. You are only allowed to slide a piece if there is no other piece, nor a wall, obstructing its path.

There will be one special 1x2 horizontal piece. There will also be a gap in the wall, on the right side, on the same row as the special piece, that *only* the special piece can fit through. The goal of the game is to get that one special horizontal piece out of the gap on the right side.

A "move" in this game is when you take a piece and slide it any number of squares (i.e. if you slide a piece horizontally one square, that is one move, and sliding it 2 squares at once is also considered one move).

Input

Google Maps

More on shortest path

- We studied finding the shortest path from a single vertex to *every* vertex
- But what about just 1 destination?

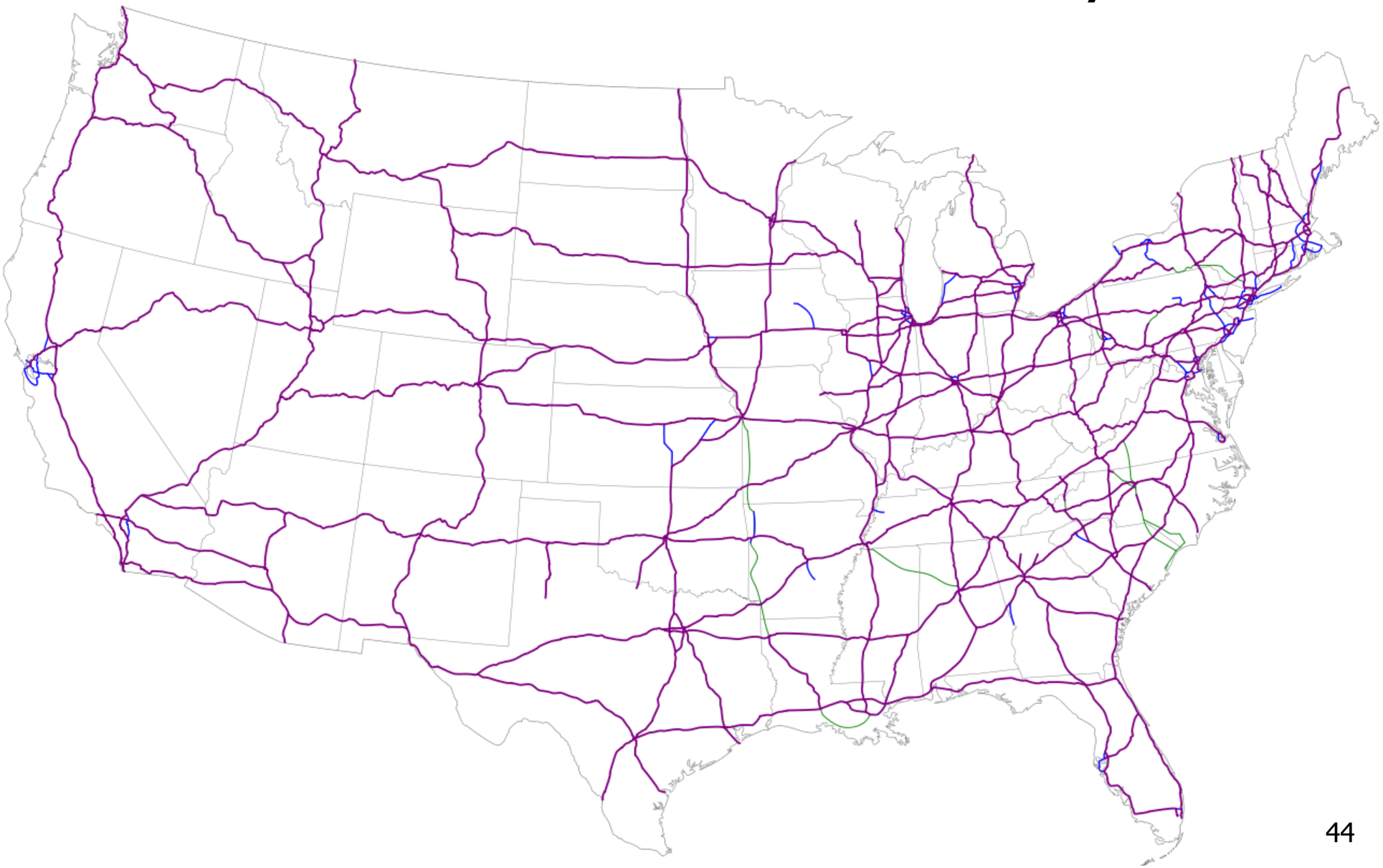
- Do the same algorithm, but stop when the destination enters the set S

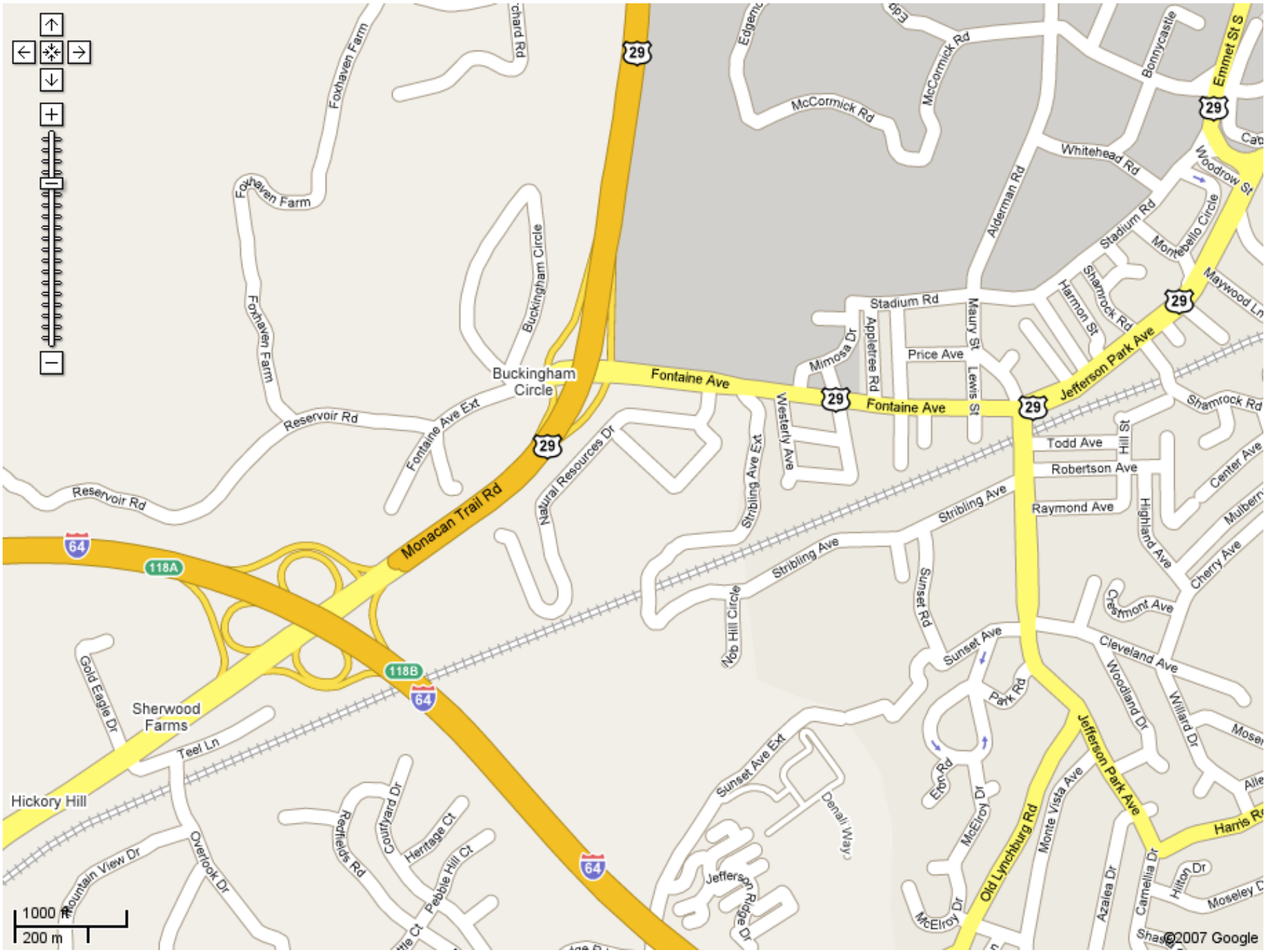
- Thus, the running time is the same!

How would you drive to Seattle?

- What constitutes a “highway”?

The Eisenhower Interstate System




Foxhaven Farm
Foxhaven Farm
Foxhaven Farm
Buckingham Circle
Buckingham Circle
Reservoir Rd
Reservoir Rd
Fontaine Ave Ext
Fontaine Ave
Fontaine Ave
Monacan Trail Rd
Natural Resources Dr
Stribling Ave Ext
Westerly Ave
Stribling Ave
Stribling Ave
Nob Hill Circle
Sunset Rd
Sunset Ave
Sunset Ave Ext
Denali Way
Jefferson Ridge Dr
McCormick Rd
McCormick Rd
Edgemont Rd
Alderman Rd
Whitehead Rd
Bonnycastle
Emmet St S
Woodrow St
Stadium Rd
Montebello Circle
Stadium Rd
Mimosa Dr
Appletree Rd
Price Ave
Maury St
Lewis St
Harmon St
Shamrock Rd
Shamrock Rd
Maywood Ln
Jefferson Park Ave
Jefferson Park Ave
Todd Ave
Hill St
Robertson Ave
Raymond Ave
Highland Ave
Center Ave
Cherry Ave
Chesmont Ave
Cleveland Ave
Woodland Dr
Willard Dr
Park Rd
Erbo Rd
McElroy Dr
McElroy Dr
Old Lynchburg Rd
Monte Vista Ave
Azalea Dr
Camellia Dr
Hilton Dr
Moseley Dr
Harris Rd
Gold Eagle Dr
Teel Ln
Sherwood Farms
Hickory Hill
Mountain View Dr
Overlook Dr
Redfields Rd
Courtyard Dr
Heritage Ct
Pebble Hill Ct
29
64
118A
118B
1000 ft
200 m
©2007 Google

Google Maps' algorithm

- (This is an educated guess, btw)

1. Assume you are starting on a "side road"
2. Transition to a "main road"
3. Transition to a "highway"
4. Get as close as you can to your destination via the "highway" system
5. Transition to a "main road", and get as close as you can to your destination
6. Transition to a "side road", and go to your destination

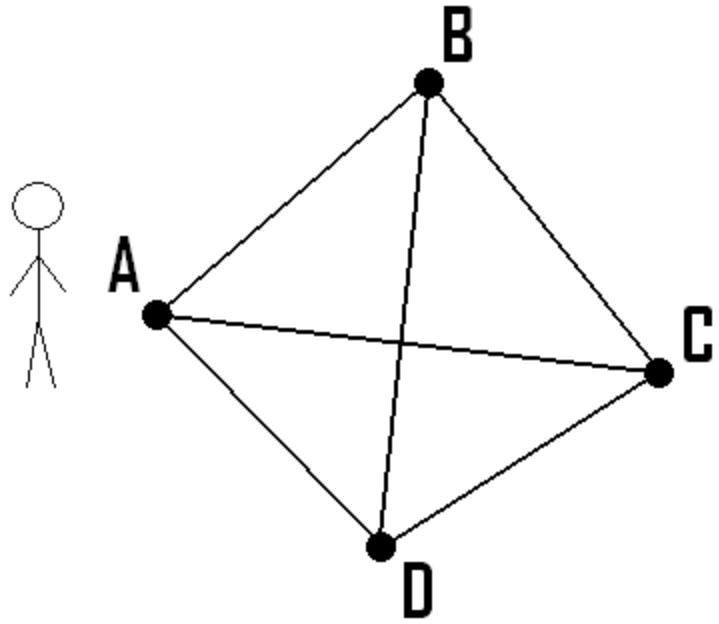
End of lecture on Fri, Nov 20

Travelling Salesman Problem

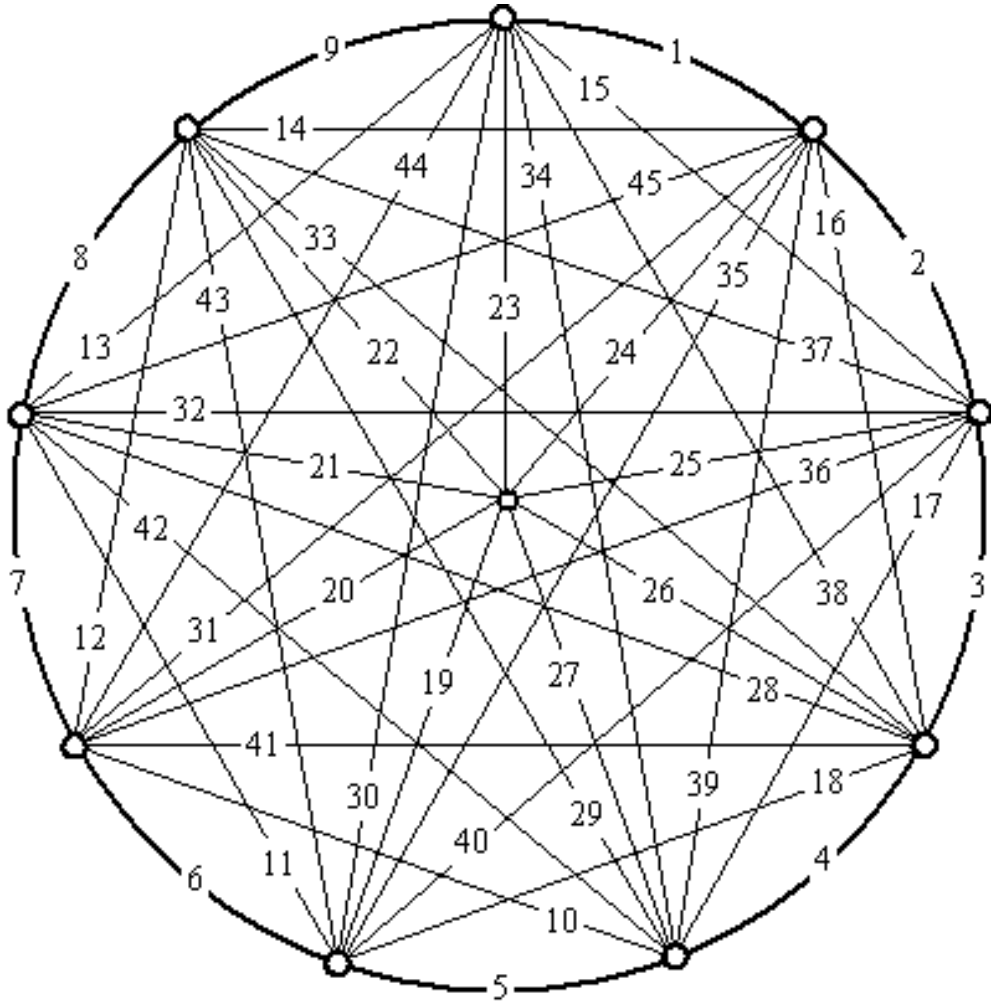
Travelling Salesman Problem (TSP)

- Given a number of cities and the costs of traveling from any city to any other city, what is the least-cost round-trip route that visits each city exactly once and then returns to the starting city?
- Really important problem for:
  - UPS, Federal Express, USPS
  - Any transport delivery system
  - Cost = distance because more fuel is used

Easy



Hard



- From http://www.geocities.com/~harveyh/Image_object/graph-10.gif

Really Hard

- From http://www.earthday.net/UER/report/images/earthday1_MAPONLY_CITY.gif

Analysis

- Hamiltonian path: a path in a connected graph that visits each vertex exactly once
  - Hamiltonian cycle: a Hamiltonian path that ends where it started
- The traveling salesman problem is thus to find the least weight Hamiltonian path (cycle) in a connected, weighted graph
- The size of the solution space is ½(n-1)!
  - Which means it's an O(n!) algorithm
  - That's exponential
  - For 10 cities: 181,440
  - For 20 cities: $6 * 10^{16}$

More Analysis

- This problem is ***NP-complete***
  - Meaning there is no known efficient solution
  - Just to try every possible path
- But there are ways to get a somewhat efficient solution (*Heuristic*)
  - It just might not be the most efficient
- What's the (usually) least expensive way to get between two US cities?
  - And is that significantly slower than the "best" algorithm?

The Record

- http://en.wikipedia.org/wiki/Traveling_salesman
- In April 2006, a computer cluster computed a path of 85,900 cities visited in 136 CPU years
  - About 3-6 months of "wall time"
- 85,900! = $9.61 * 10^{386,526}$
- Assume you can compute 1 million paths each second
  - That would take $3.04 * 10^{386,516}$ years!
  - They used acceleration techniques, obviously…

Lab 11

Lab 11

- Pre-lab: implement a topological sort
- In-lab: implement a brute-force traveling salesman problem
  - Using locations in Tolkien's Middle Earth
- Post-lab: do a report containing analysis and a study of acceleration techniques

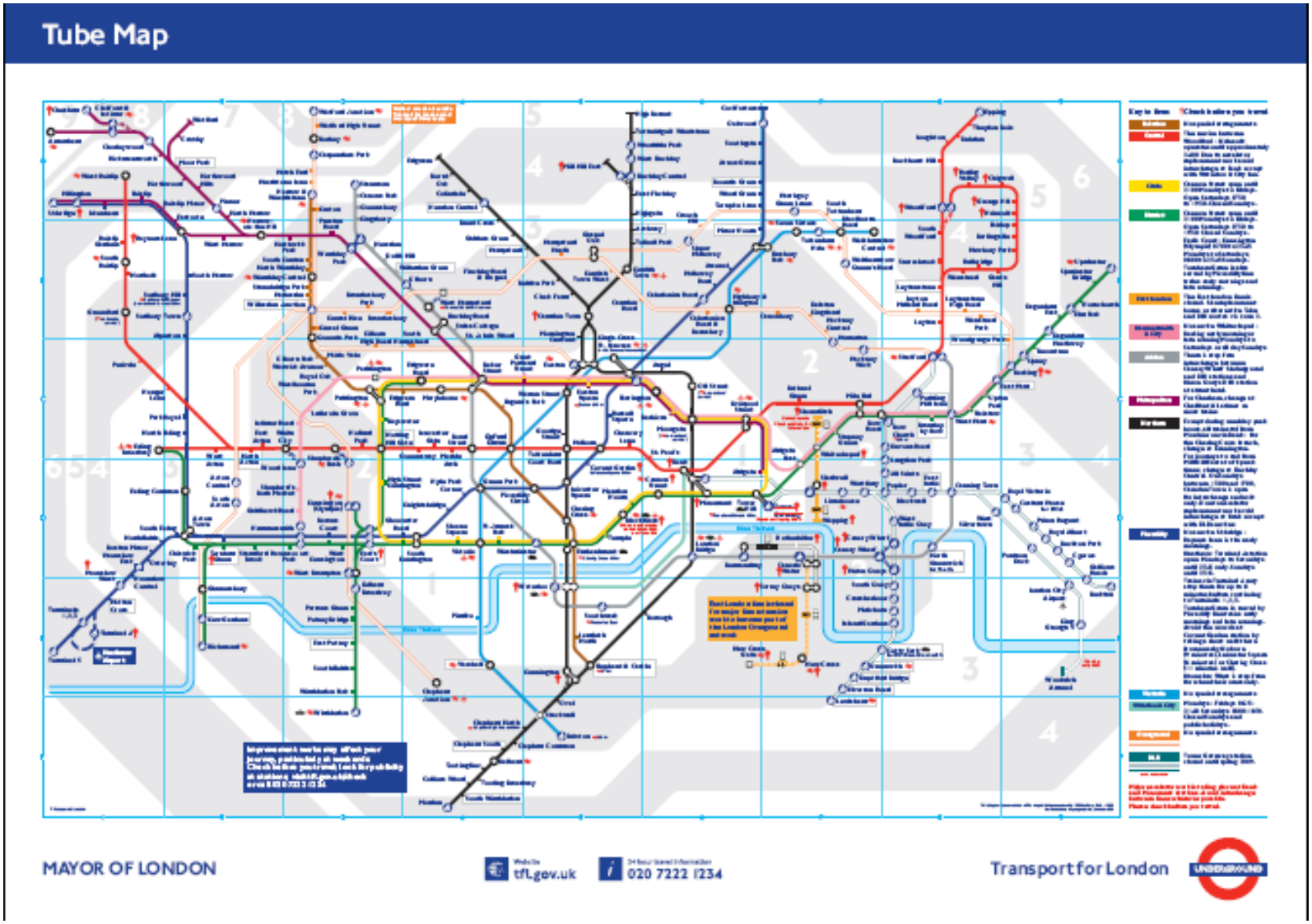
End of lecture on Mon, Nov 23

Minimum Spanning Trees (MST)

Spanning Tree

- Suppose you are going to build a transport system:
  - Set of cities
  - Roads, rail lines, air corridors connecting cities
- Trains, buses or aircraft between cities
- Which links do you actually use?
  - Cannot use a complete graph
  - Passengers can change at connection points
- Want to minimize number of links used
- Any solution is a ***tree***

Spanning Tree

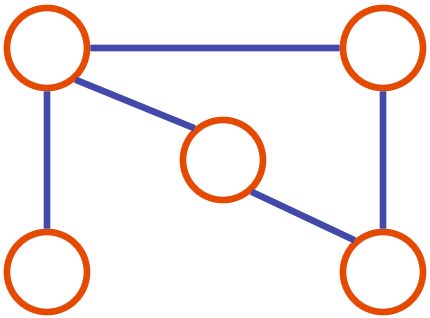


Spanning Tree

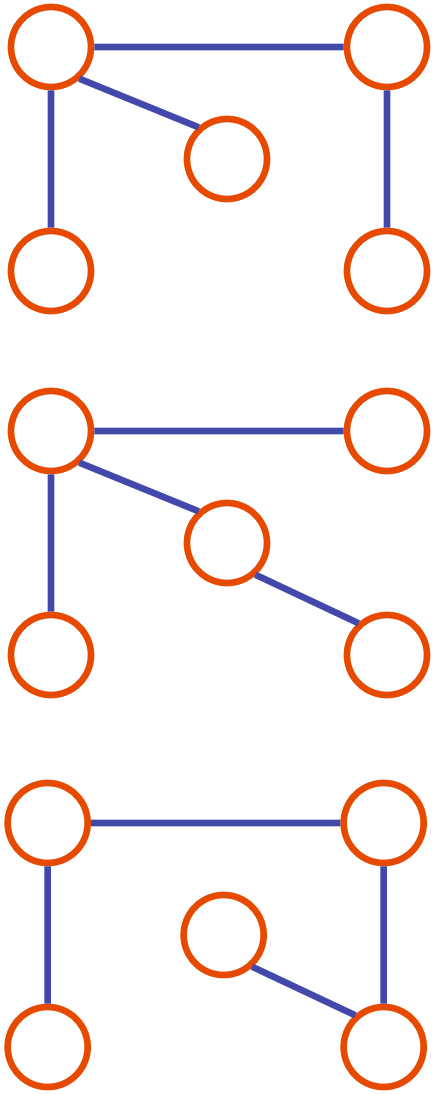
- A ***spanning tree*** of a graph G is a subgraph of G that contains every vertex of G and is a tree
- Any connected graph has a spanning tree
- Any two spanning trees of a graph have the same number of nodes
- Construct a spanning tree:
  - Start with the graph
  - Remove an edge from each cycle
  - What remains has the same set of vertices but is a tree

Spanning Tree

Graph



Spanning Trees



Minimal Spanning Tree

- Spanning trees are simple
- But suppose edges have weights!
  - "Cost" associated with the edge
  - Miles for a transport link, for example
- Spanning trees each have a different total weight
- Minimal-weigh Spanning Tree:

Spanning tree with the minimal total weight

Minimum Spanning Trees

- Given an undirected graph **G**=(**V**,**E**), find a graph **G'=(V,E')** such that
  - E' is a subset of E
  - |E'| = |V| - 1
  - G' is connected
  - $\sum_{(u,v)\in E'} c_{uv}$ is minimal

  G' is a **minimal spanning tree**.

Applications: wiring a house, cable TV lines, power grids, Internet connections

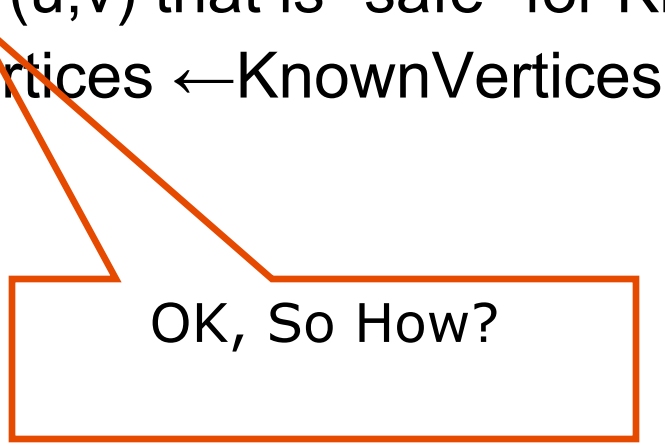
Generic Minimum Spanning Tree Algorithm

KnownVertices ← {}

while KnownVertices does not form a spanning tree **loop**

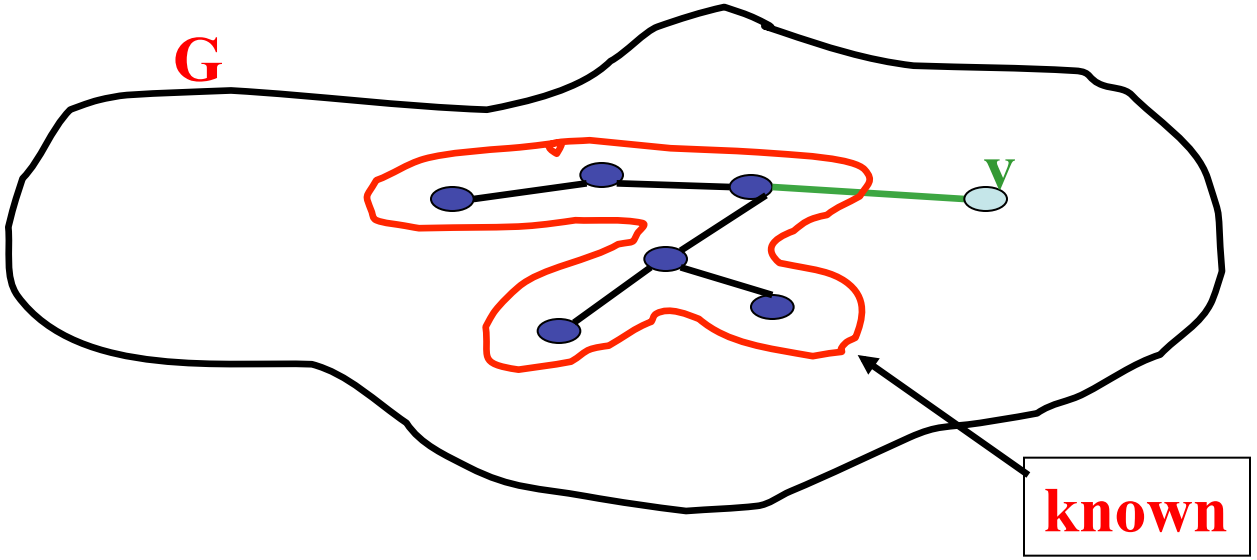
find edge (u,v) that is “safe” for KnownVertices
KnownVertices ←KnownVertices U {(u,v)}

end loop



Prim's algorithm

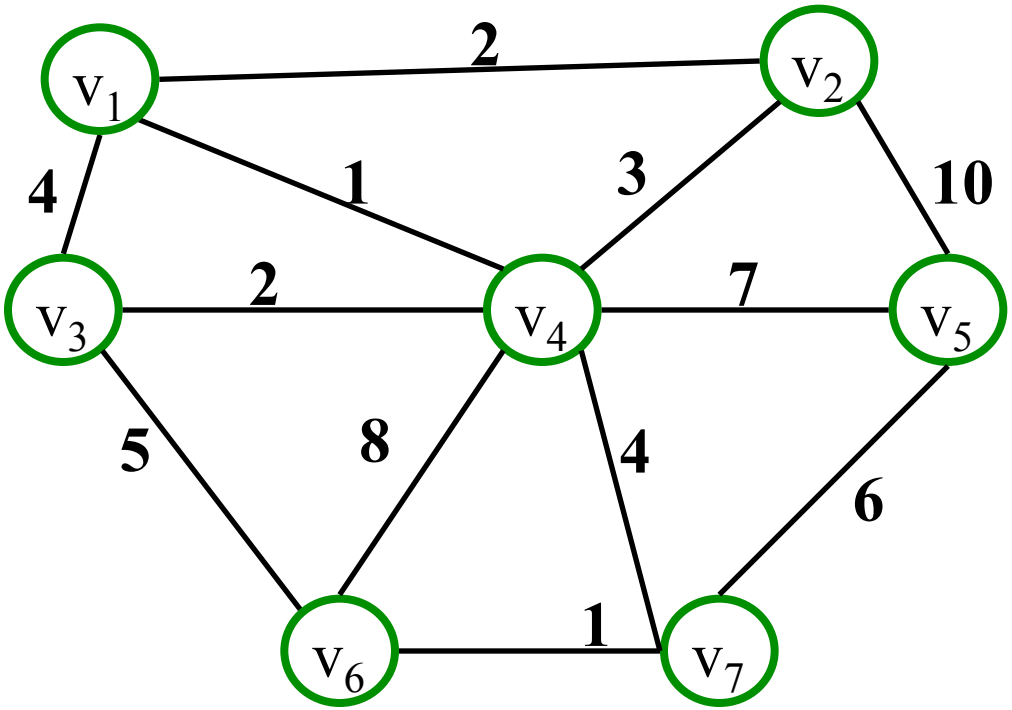
Idea: Grow a tree by adding an edge from the "known" vertices to the "unknown" vertices. Pick the edge with the smallest weight.



Prim's Algorithm for MST

- Pick one node as the root,
- Incrementally add edges that connect a "new" vertex to the tree.
- Pick the edge (u,v) where:
  - u is in the tree, v is not AND
  - where the edge weight is the smallest of all edges (where u is in the tree and v is not).

MST



v1

{v1, v4}

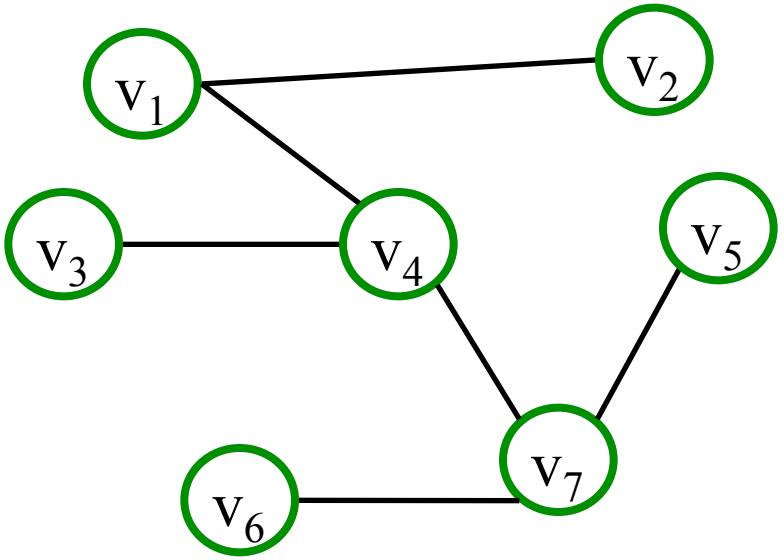
{v1, v2}

{v4, v3}

{v4, v7}

{v7, v6}

{v7, v5}



Analysis

Running time: Same as Dijkstra's: O(*e* log *v*)

Correctness:

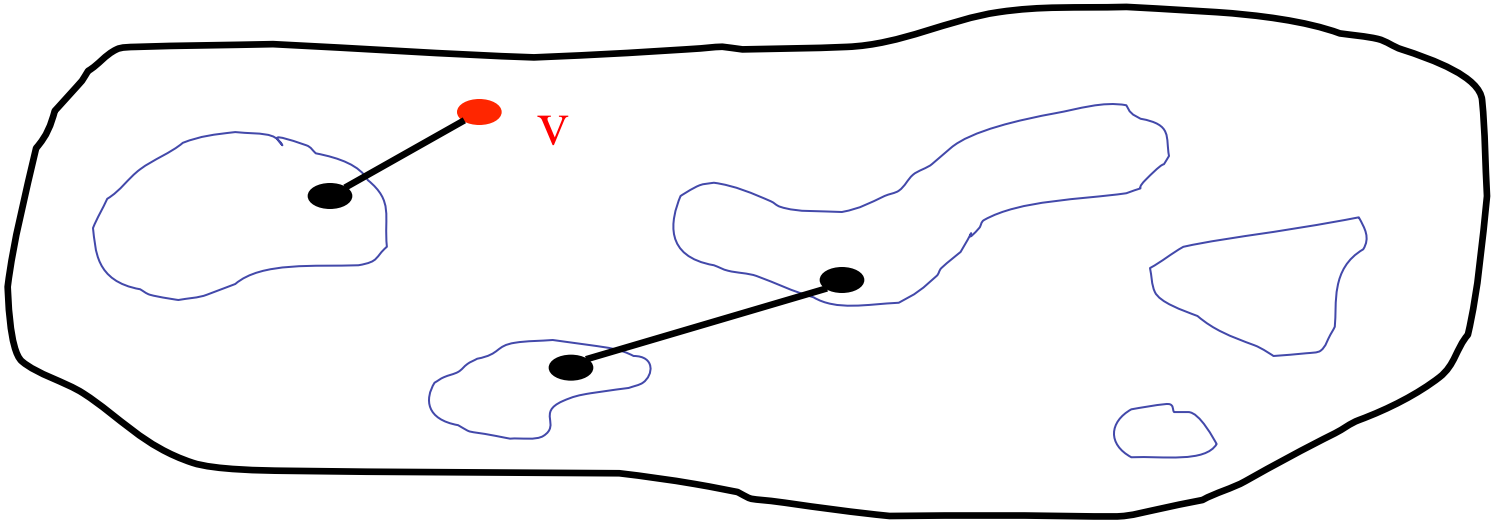
Suppose we have a partially built tree that we know is contained in some minimum spanning tree T. Let (u,v)∈E, where u is "known" and v is "unknown" and has minimal cost.

Then there is a MST T' that contains the partially built tree and (u,v) that has as low a cost as T.

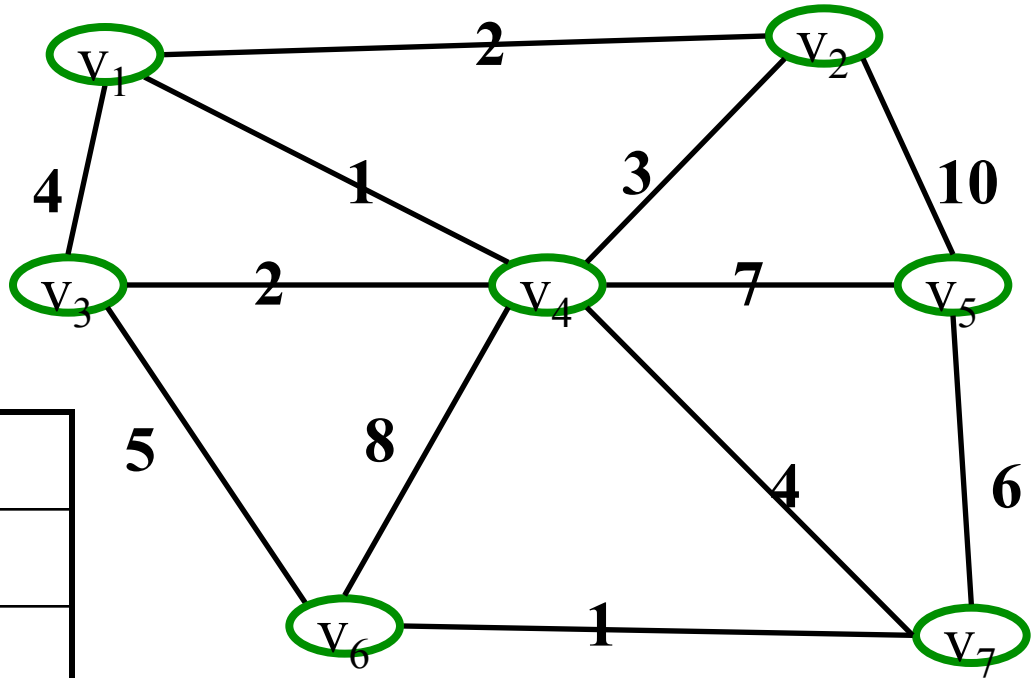
Kruskal's MST Algorithm

- Idea: Grow a forest out of edges that do not create a cycle. Pick an edge with the smallest weight.

G=(V,E)



MST

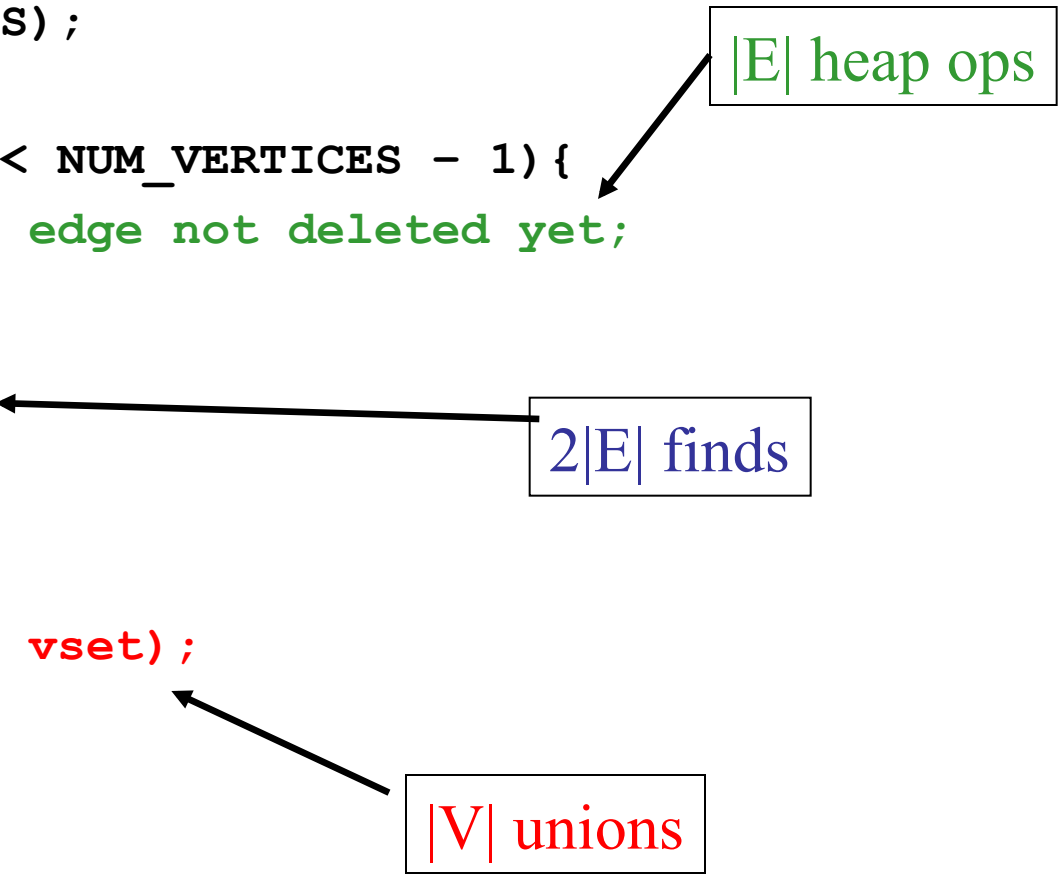


V	Kwn	Distance	path
v1			
v2			
v3			
v4			
v5			
v6			
v7			

Kruskal code

```
void Graph::kruskal(){
  int edgesAccepted = 0;
  DisjSet s(NUM_VERTICES);

  while (edgesAccepted < NUM_VERTICES - 1){
    e = smallest weight edge not deleted yet;
    // edge e = (u, v)
    uset = s.find(u);
    vset = s.find(v);
    if (uset != vset){
      edgesAccepted++;
      s.unionSets(uset, vset);
    }
  }
}
```



End of lecture on Wed, Nov 30

- Also went over the first 7 slides of 14-memory