CS2110 Lecture 31 Apr. 5, 2021

- HW 7 due Wednesday
- No required DS tomorrow. Attend if you want help finishing HW7.
- I will post a DS8 problem later this week. It will help with HW8 that will also be posted later this week.

Last time

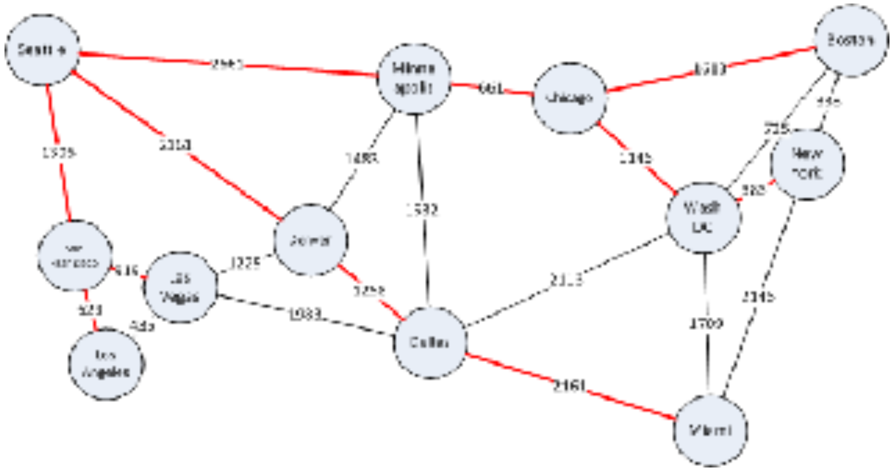
- Finished sorting - efficient O(n log n) sorts
  - Merge sort and quicksort
- demo of plotting/graphing sorting results using matplotlib/pylab

This week

- Greedy algorithms
- Begin optimization and graph algorithms

Optimization and graph problems

- Many computing tasks these days involve solving **optimization problems** – finding the smallest, biggest, best, cheapest of something
- In general, optimization problems are expressed in terms of two components
  - An *objective function* that is to be minimized/maximized (e.g. airfare, travel distance, travel time)
  - *A set of constraints* that must be met (e.g. route must include these intermediate cities, departure must be after 8am, arrival must be before noon)
- Many optimization problems can be addressed as problems on **graphs** (the computer science kind, consisting of nodes/vertices and edges/ connections, not the 2D x-y *plots* you have been using to visualize running time behavior.)



- HW 8 will focus on optimization and graphs. Before we get to graphs, a quick look at other optimization problems …

A simple optimization problem

Suppose you need to give someone n cents in change (given US coins – penny, nickel, dime, quarter). How do you do it with the minimum number of coins?

- "greedily" give as many large value coins as possible first, then next largest size, etc. That is, first as many quarters as possible, then as many dimes, …
- E.g. for 56¢: 2 quarters, 1 nickel, 1 penny

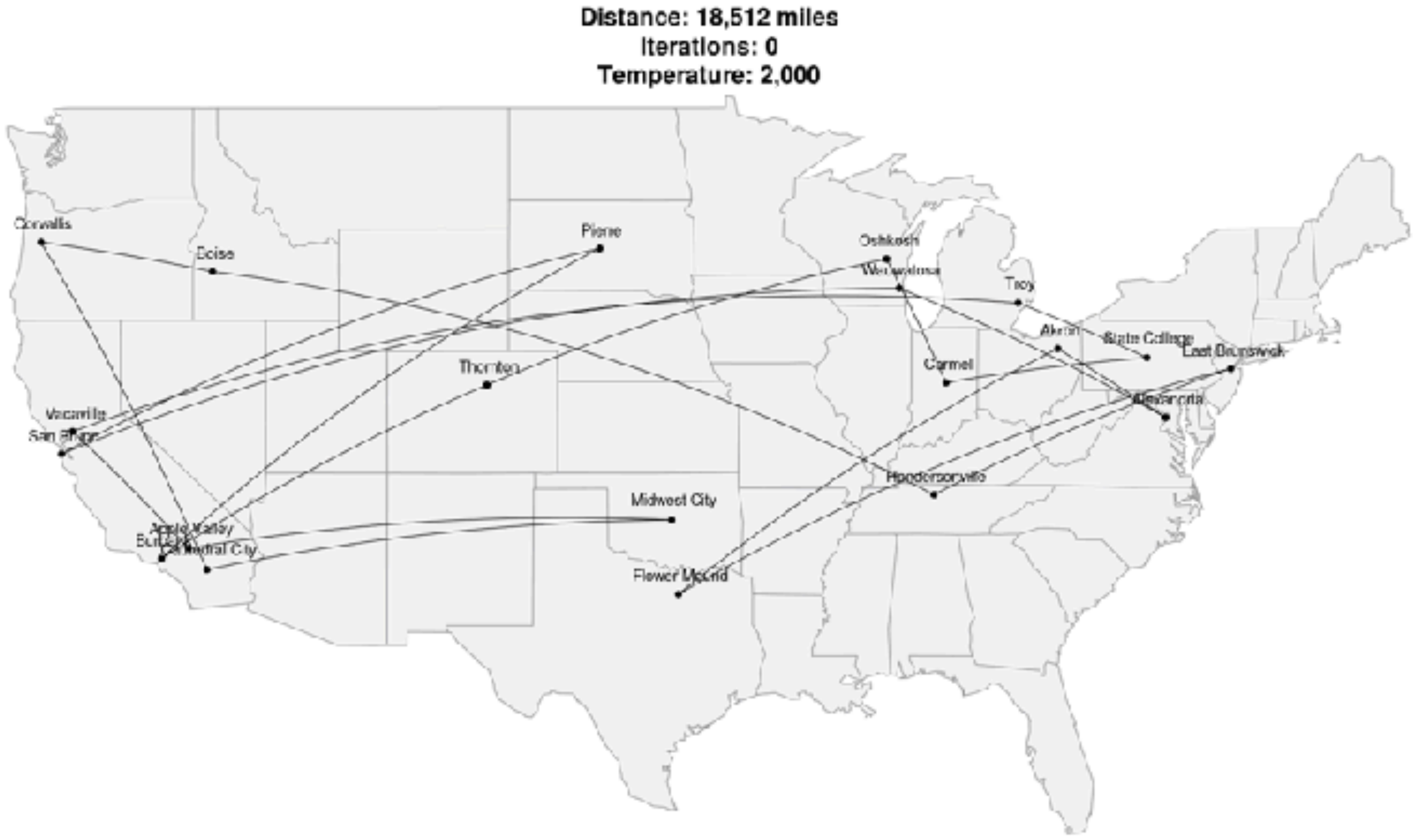
What if we replace nickel (5c) with 3 cent and 4 cent coins? Does same greedy approach work?

- for 56¢ greedy approach yields 25, 25, 4, 1, 1 but 25, 25, 3, 3 is fewer coins!

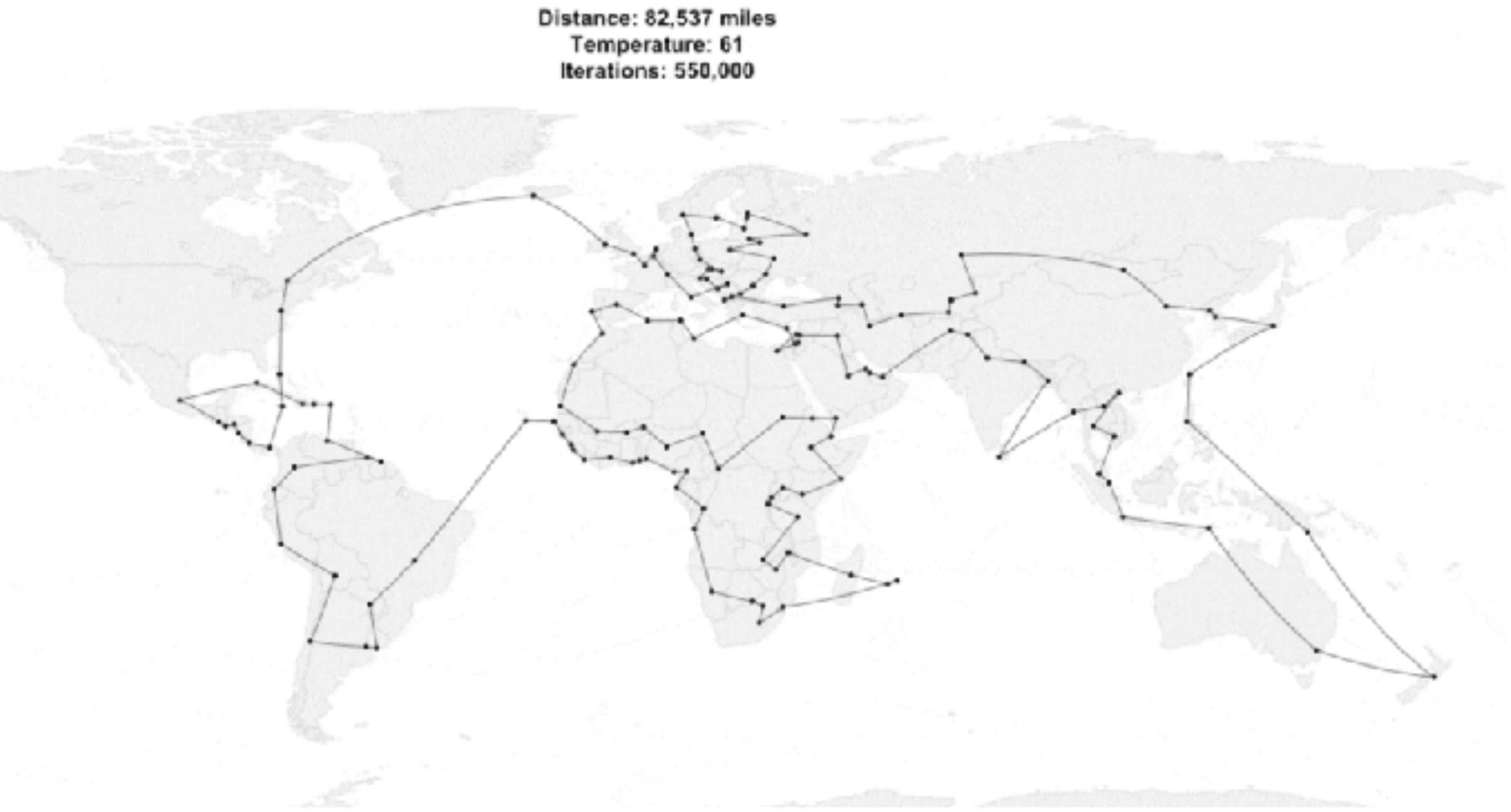
For US coins, the algorithm works. It is an example of a broad class of "greedy algorithms" – if you take Algorithms (CS3330), you will likely study more about greedy algorithms.

Greedy algorithms

- Generally, a greedy algorithm is one that proceeds as follows:
  - At each step, choose the “locally”/”apparently”/”immediately” best option (e.g. the one that seems like to make the most progress toward a solution)
- The idea (hope!) behind greedy algorithms is that by making many locally optimal choices we end up with overall optimal solution.
- But, as we saw, *doesn’t always succeed!* Sometimes need to work harder.
- Greedy algorithm for Travelling Salesperson problem? No, does not always yield optimal solution
- Greedy algorithm for shortest driving route between two cities? (E.g. driving directions in Google maps). Yes, Dijkstra’s algorithm.
- Greedy algorithms are very important and useful. But you need to think carefully about whether greedy approach indeed gives you an optimal solution (or, if not, a good enough one)
- for more, see http://en.wikipedia.org/wiki/Greedy_algorithm



Source: http://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/



Source: http://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/

See also nice short video at:http://www.youtube.com/watch?v=SC5CX8drAtU&list=PLxH6ufuE9gKtM5-bbFMTp_1-avAN-iuiq&index=3

Egyptian fractions

3/4 -> sum of (different) fractions all with 1 as numerator

E.g. 3 / 4 -> 1 / 2 + 1 / 4

Greedy algorithm?

Try in sequence ½, 1/3, ¼, 1/5, …

Implementing this will be Q1 of HW8. Note: DO NOT use any division – it does not work well!)

Lots of info about this problem at: http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fractions/egyptian.html

If current fraction is n/d and current candidate to subtract is 1/c:

- How do we know if 1/c is less than n/d?
- How do we calculate (n/d) - (1/c)?

both without using division?

Another optimization problem

	Value	Weight
clock	175	10
painting	90	9
radio	20	4
vase	50	2
book	10	1
computer	200	20

Burglar with a knapsack in a home full of valuable items

- Objective function: fill knapsack with maximum value
- Constraint: knapsack can only hold 20 pounds

Algorithm to solve this?

Burglar filling knapsack

	Value	Weight	Val/wt
clock	175	10	17.5
painting	90	9	10
radio	20	4	5
vase	50	2	25
book	10	1	10
computer	200	20	10

Constraint:
Knapsack
can hold up
to 20lbs

- Greedy approach says to pick "best" at each step. What rule could we use for best here?
  - Highest value -> computer only -> $200 total
  - Lowest weight -> book, vase, radio painting, -> $170 total
  - Highest value/weight -> vase, clock, book, radio -> $255 total

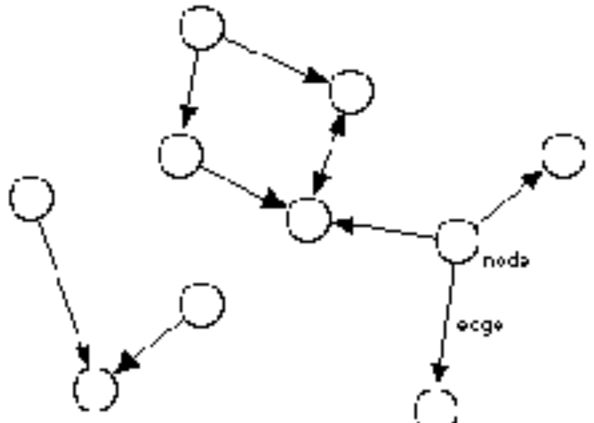
None of these criteria produce the optimal solution for this particular situation (best total is $275 via clock, painting, book).

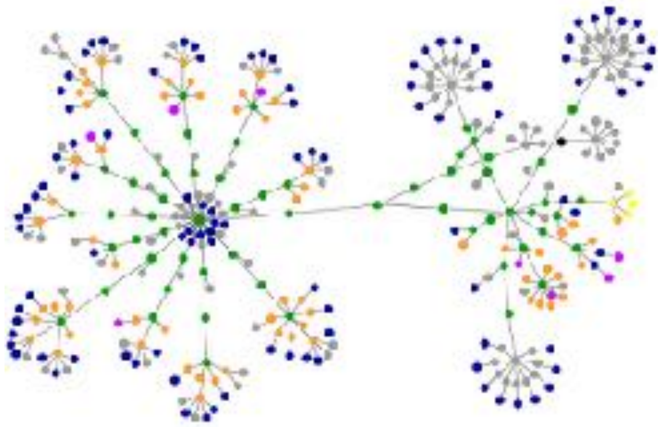
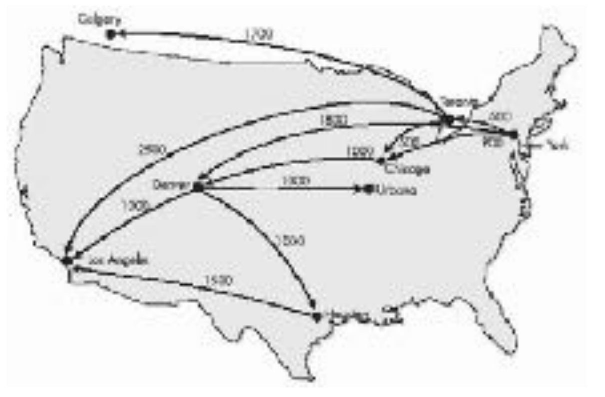
We could easily write an algorithm that always find the best by trying every subset of items. However, this solution is very inefficient for many knapsack-like problems. If have n items, how many subsets? 2^n, so exhaustive search potentially very slow

At present, no known efficient algorithm for knapsack problems

Graphs and optimization problems based on graphs

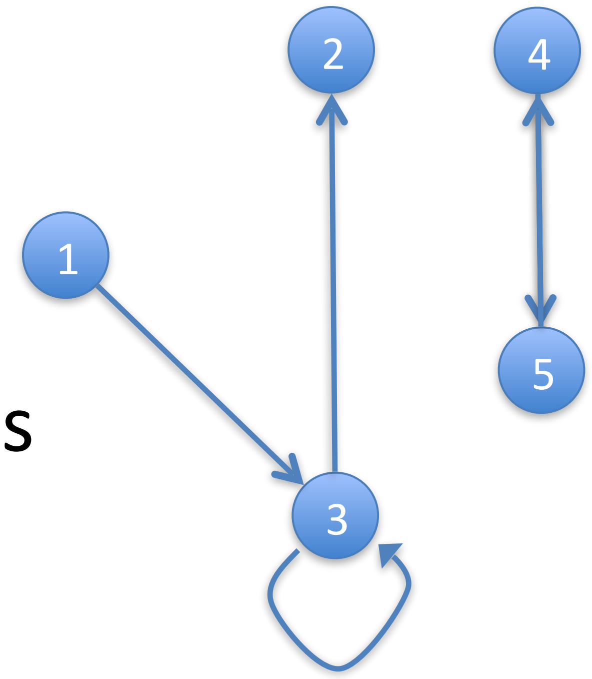
- *Many* important real-world problems can be modeled as optimization problems on **graphs**
- A graph is:
  - A set of nodes (vertices)
  - A set of edges (arcs) representing connections between pairs of nodes
- There are several types of graphs:
  - **Directed**. Edges are "one way" from source to destination)
  - **Undirected**. Edges have no particular direction – can travel either way, "see" each node from other, etc.
  - **Weighted**. Edges have associated numbers called weights that can be used to represent cost, time, flow capacity, etc.
- See Ch. 2 of follow-up book to our text – Think Complexity - http://greenteapress.com/complexity/html/thinkcomplexity003.html

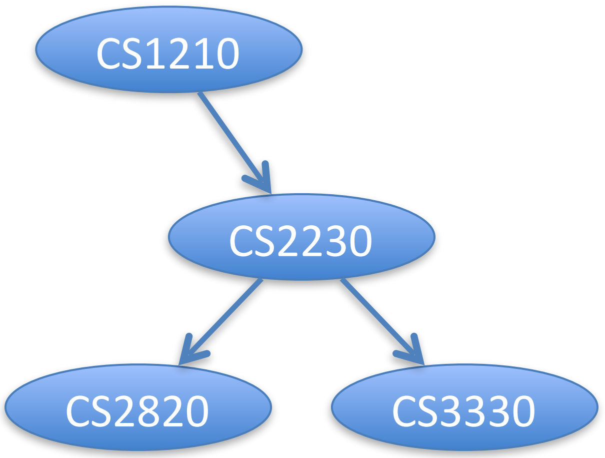


Directed graph

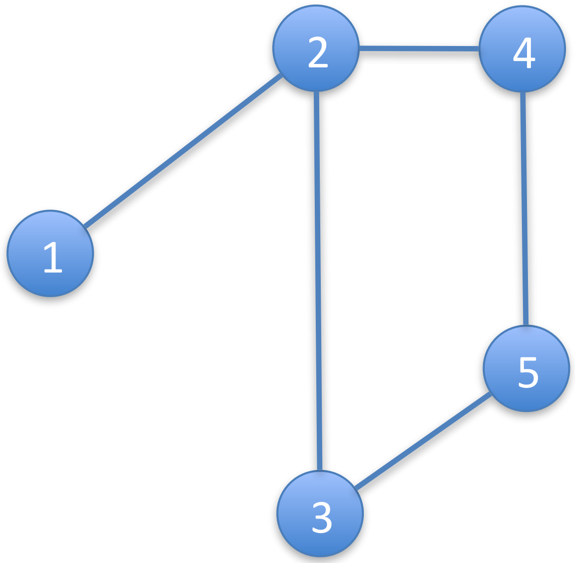
- edges are "one way" from source to destination
- Can have two (one each way) between a pair of nodes
- Node can have edge to self
- Example relationships:
  - course prerequisite
  - hyperlink between web pages
  - street between intersections
  - Twitter follower
  - Infection spread from-to

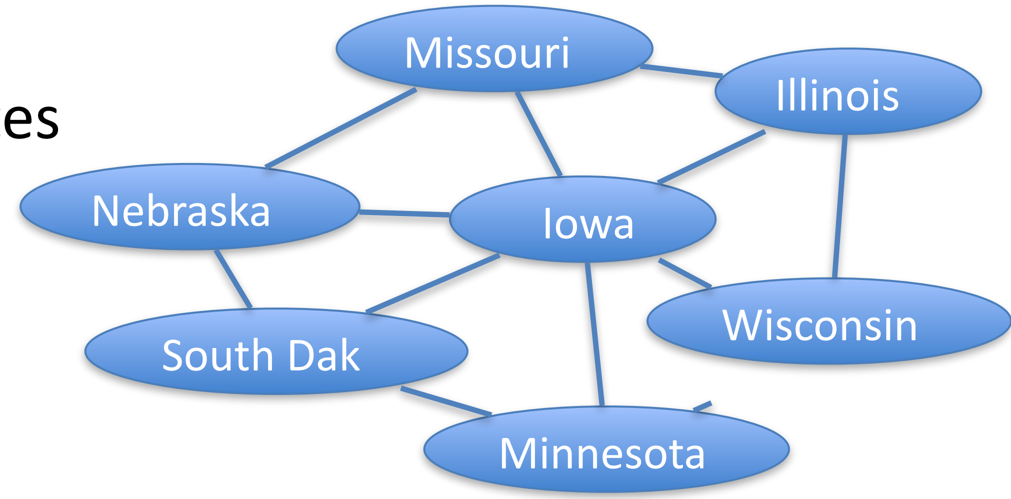




Undirected graph

- Edges have no direction. Can "travel" either direction
- Can have only one edge between a pair of nodes*
- Node cannot have edge to self
- Example relationships:
  - Facebook friend
  - Bordering countries/states

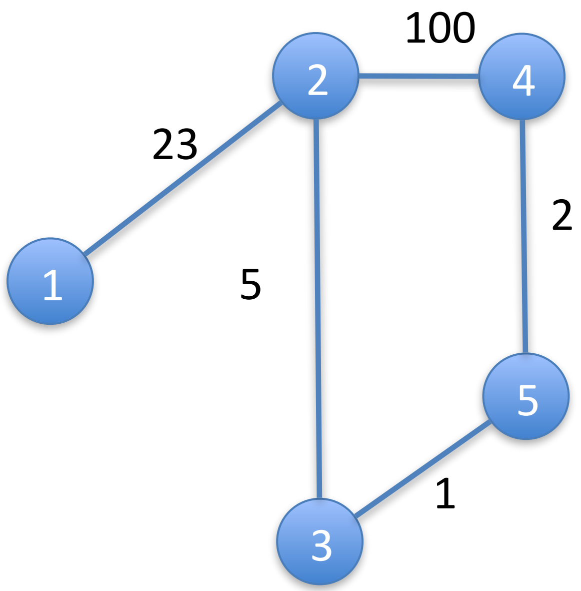


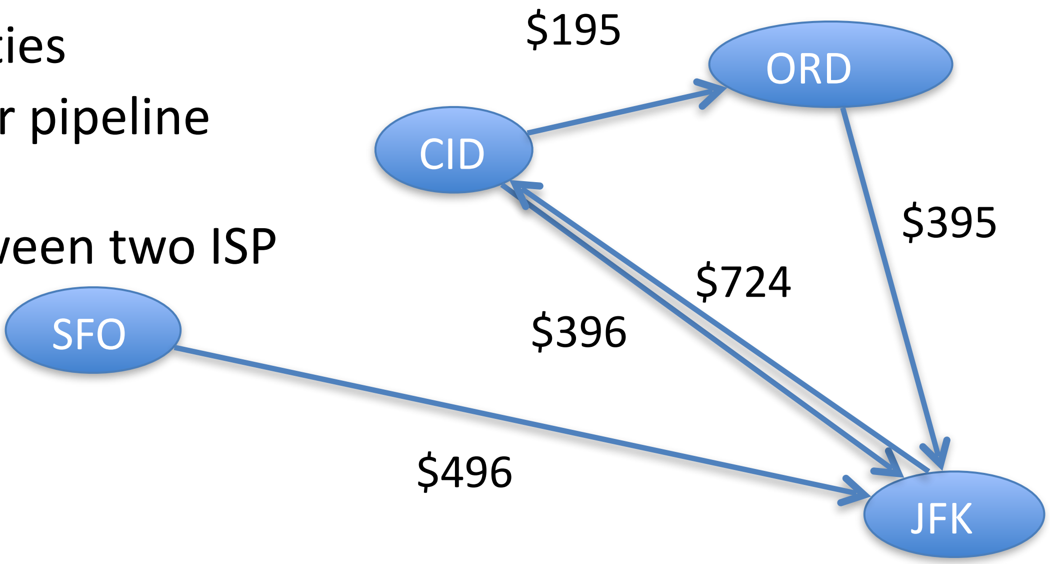


*another kind of graph – multigraph – relaxes this rule

Weighted graphs

- Variant of both directed and undirected graphs in which each edge has an associate number called a **weight** or cost
- Edge weight provides additional information about the relationship between the nodes.
- Example relationships:
  - Airfare between two cities
  - Distance between two cities
  - Flow capacity of oil/water pipeline between two points
  - Network bandwidth between two ISP nodes





The rest of the week

- Graph representations
- Graph traversals: BFS, DFS, and the HW8 word ladder problem