Week 10 - Friday

COMP 2230

Last time

- Matrix representations of graphs
- Graph isomorphism

Questions?

Assignment 5

Logical warmup

- Jean and Gene have fallen in love on a dating app (without meeting in person)
- Gene wants to mail Jean an engagement ring
- These days, the mail has become increasingly subject to theft, and anything will be stolen unless it's sent in a locked box
- Jean and Gene each have plenty of padlocks, but neither has a key to any of the other one's locks
- How can Gene get the ring safely to Jean?

Back to Matrices for a Second

Finding powers of a matrix

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix}$$

- Compute A^0
- Compute A^1
- Compute A^2

Matrix powers for graphs

- We can find the number of walks of length k that connect two vertices in a graph by raising the adjacency matrix of the graph to the k^{th} power
- Raising a matrix to the 0^{th} power means you can only get from a vertex to itself (identity matrix)
- Raising a matrix to the first power means that the number of paths of length one from one vertex to another is exactly the number of edges between them

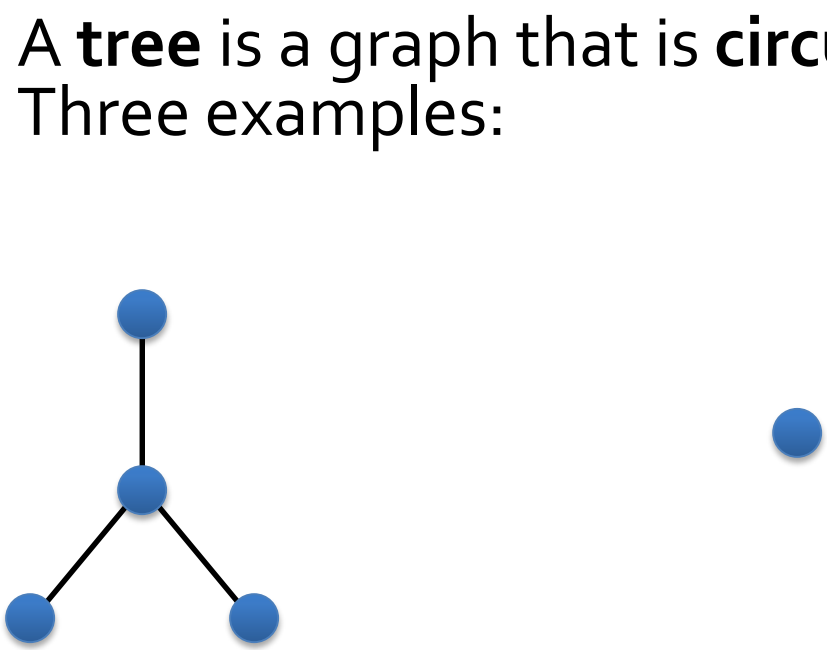
Trees, Rooted Trees, and Spanning Trees

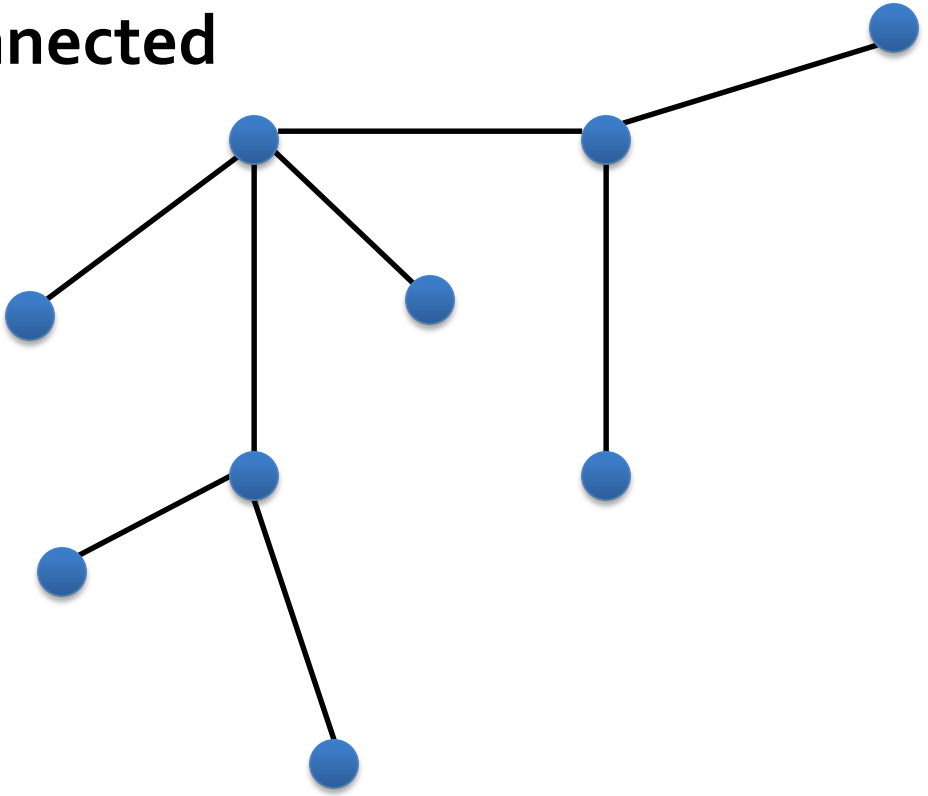
Three-Sentence Summary

Trees

Trees

- A **tree** is a graph that is **circuit-free** and **connected**
- Three examples:

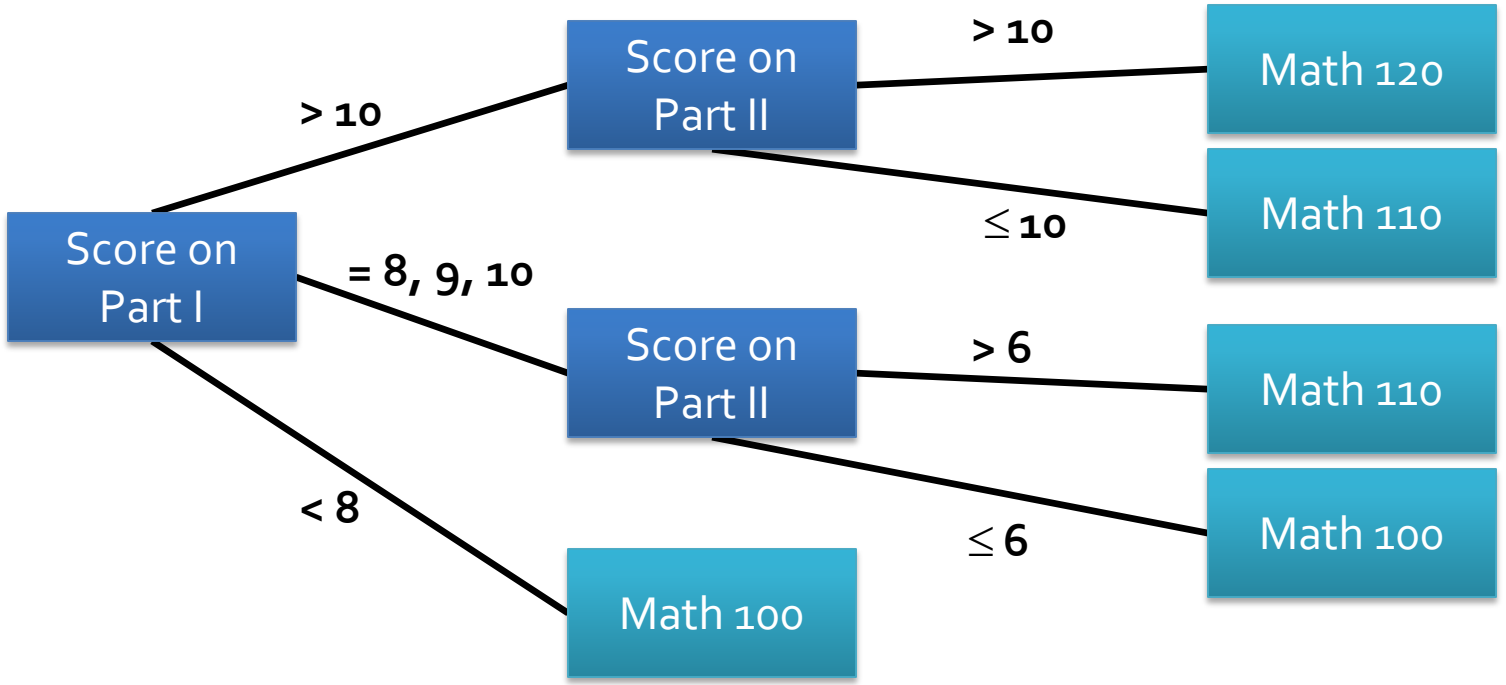




- A graph made up of disconnected trees is called a **forest**

Applications of trees

- Trees have almost unlimited applications
- You should all be familiar with the concept of a decision tree from programming



Parse trees

- A grammar for a formal language (such as we will discuss in a couple of weeks) is made up of rules that allow non-terminals to be turned into other non-terminals or terminals
- For example:
  1. <sentence> → <noun phrase><verb phrase>
  2. <noun phrase> → <article><noun> | <article><adjective><noun>
  3. <verb phrase> → <verb><noun phrase>
  4. <article> → a | an | the
  5. <adjective> → funky
  6. <noun> → DJ | beat
  7. <verb> → plays | spins
- Make a parse tree corresponding to the sentence, "The DJ plays a funky beat"

Describing trees

- Any tree with more than one vertex has at least two vertices of degree 1
- If a vertex in a tree has degree 1 it is called a **terminal vertex** (or **leaf**)
- All vertices of degree greater than 1 in a tree are called **internal vertices** (or **branch vertices**)

A property of trees

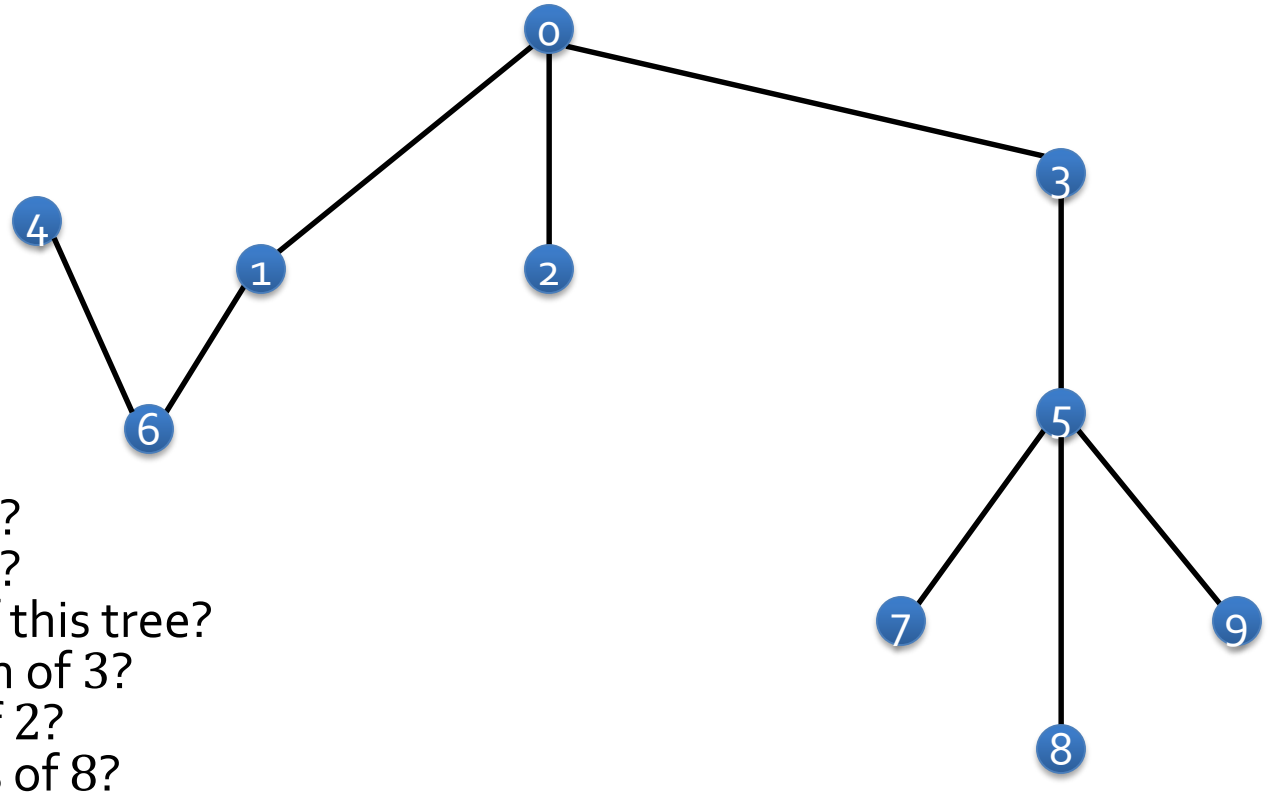
- For any positive integer n, a tree with n vertices must have $n - 1$ edges
- Prove this by mathematical induction

Rooted trees

- In a **rooted tree**, one vertex is distinguished and called the **root**
- The **level** of a vertex is the number of edges along the unique path between it and the root
- The **height** of a rooted tree is the maximum level of any vertex of the tree
- The **children** of any vertex v in a rooted tree are all those nodes that are adjacent to v and one level further away from the root than v
- Two distinct vertices that are children of the same parent are called **siblings**
- If w is a child of v, then v is the **parent** of w
- If v is on the unique path from w to the root, then v is an **ancestor** of w and w is a **descendant** of v

Rooted tree example

- Consider the following tree, rooted at 0



- What is the level of 5?
- What is the level of 0?
- What is the height of this tree?
- What are the children of 3?
- What is the parent of 2?
- What are the siblings of 8?
- What are the descendants of 3?

Binary trees

- A **binary tree** is a rooted tree in which every parent has at most two children
- Each child is designated either the **left child** or the **right child**
- In a **full binary tree**, each parent has exactly two children
- Given a parent v in a binary tree, the **left subtree** of v is the binary tree whose root is the left child of v
- Ditto for **right subtree**

Binary tree applications

- As we all know from data structures, a binary tree can be used to make a data structure that is efficient for insertions, deletions, and searches
- But it doesn't stop there!
- We can represent binary arithmetic with a binary tree
- Make a binary tree for the expression $((a - b) \cdot c) + (d/e)$
  - The root of each subtree is an operator
  - Each subtree is either a single operand or another expression

Full Binary Tree Theorem

- If k is a positive integer and T is a full binary tree with k internal vertices, then T has a total $2k + 1$ vertices and has $k + 1$ terminal vertices
- Prove it!
- **Hint:** Induction isn't needed. We just need to relate the number of non-terminal nodes to the number of terminal nodes
- **Second hint:** The number of vertices that have a parent is twice the number of parents (because it's a full binary tree)

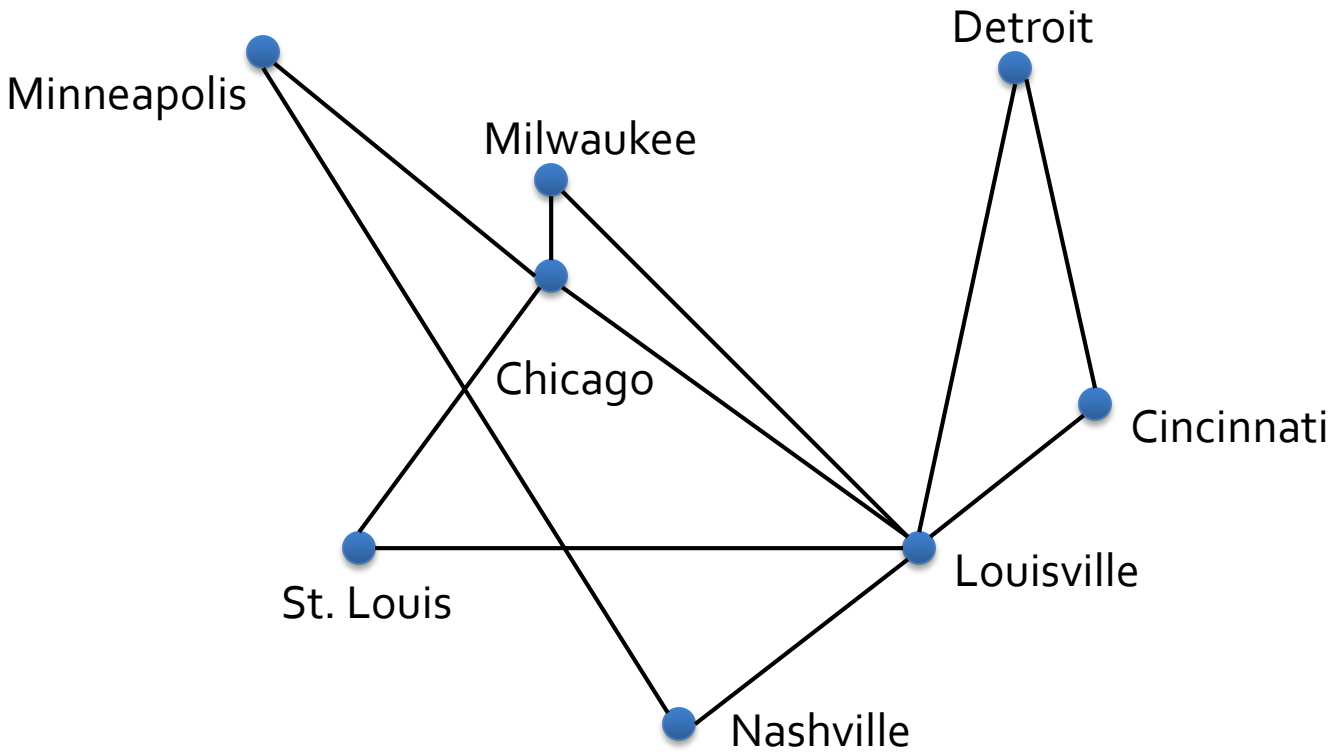
Leaves in a tree

- If T is a binary tree with t terminal vertices and height h, then $t \leq 2^h$
- Prove it using strong induction on the height of the tree
- Hint: Consider cases where the root of the tree has 1 child and 2 children separately

Spanning Trees

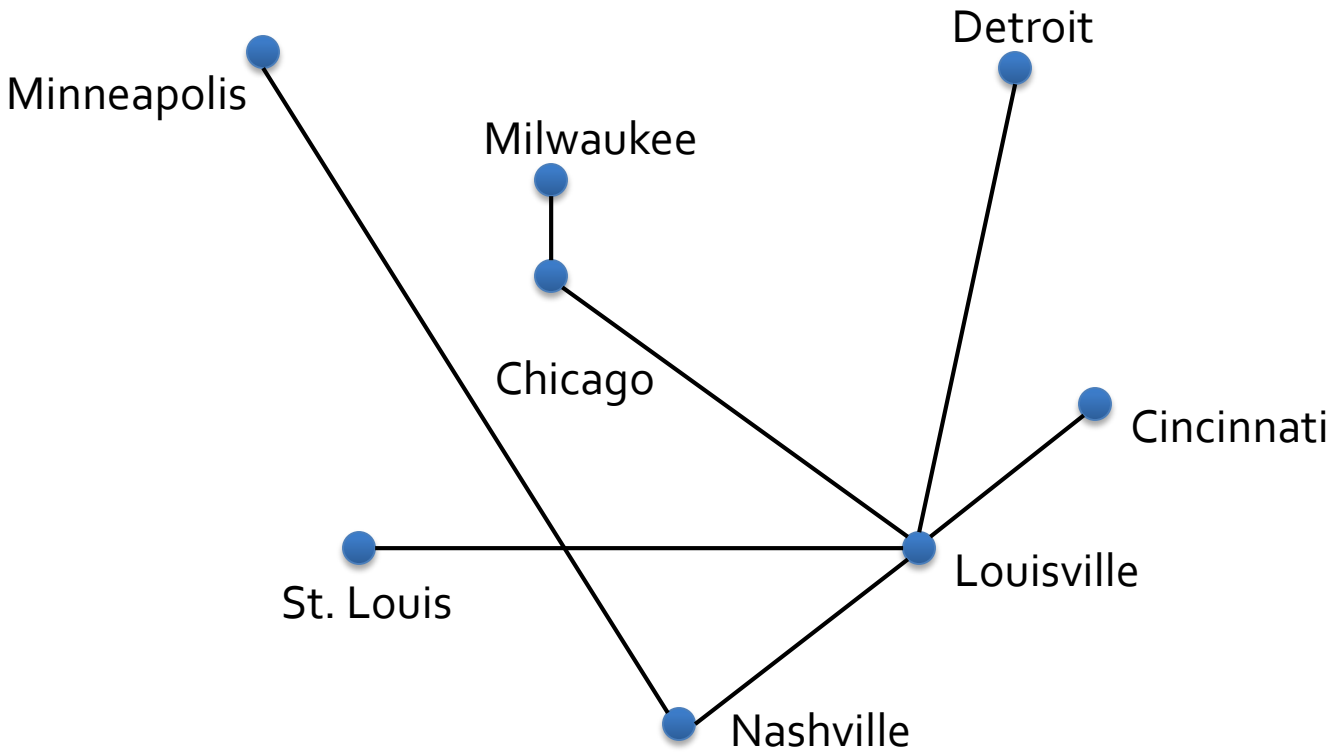
Turning graphs into trees

- Consider the following graph that shows all the routes an airline has between cities



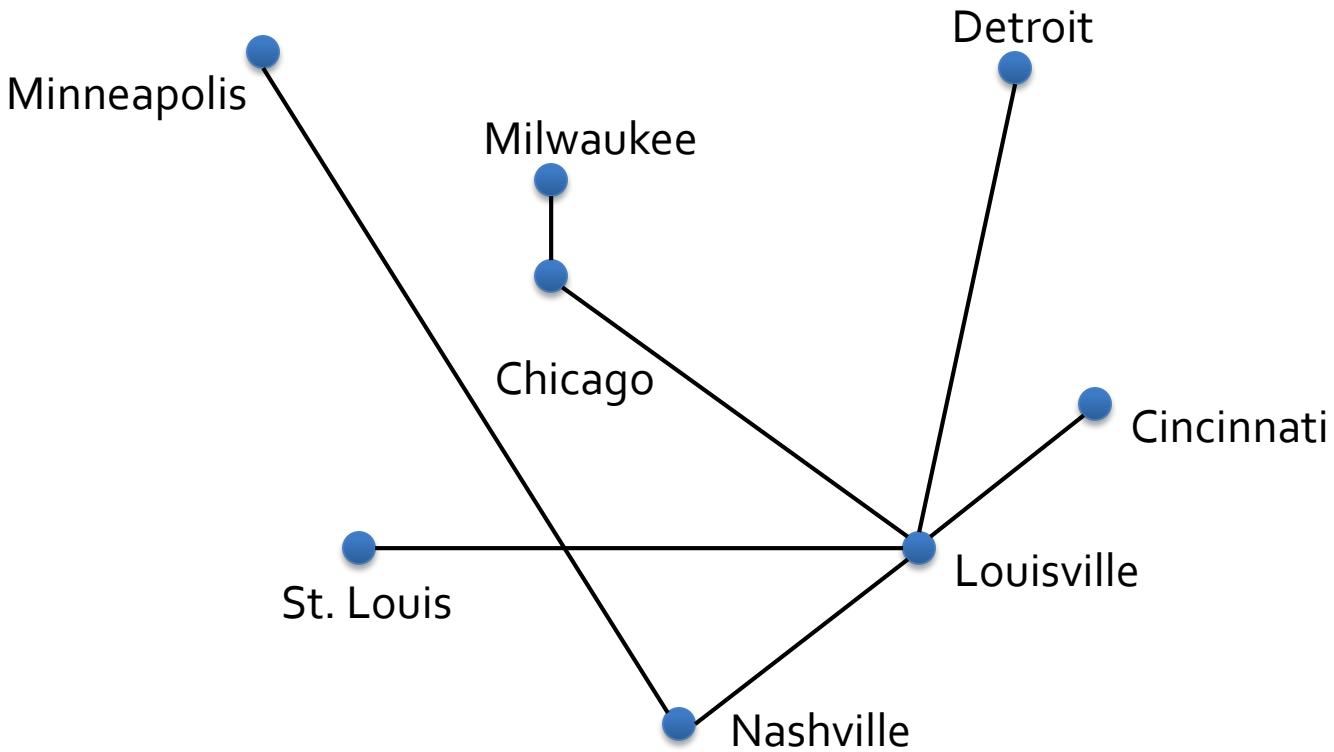
Turning graphs into trees

- What if we want to remove routes (to save money)?
- How can we keep all cities connected?



The best tree?

- Does this tree have the smallest number of routes?
- Why?



Spanning trees

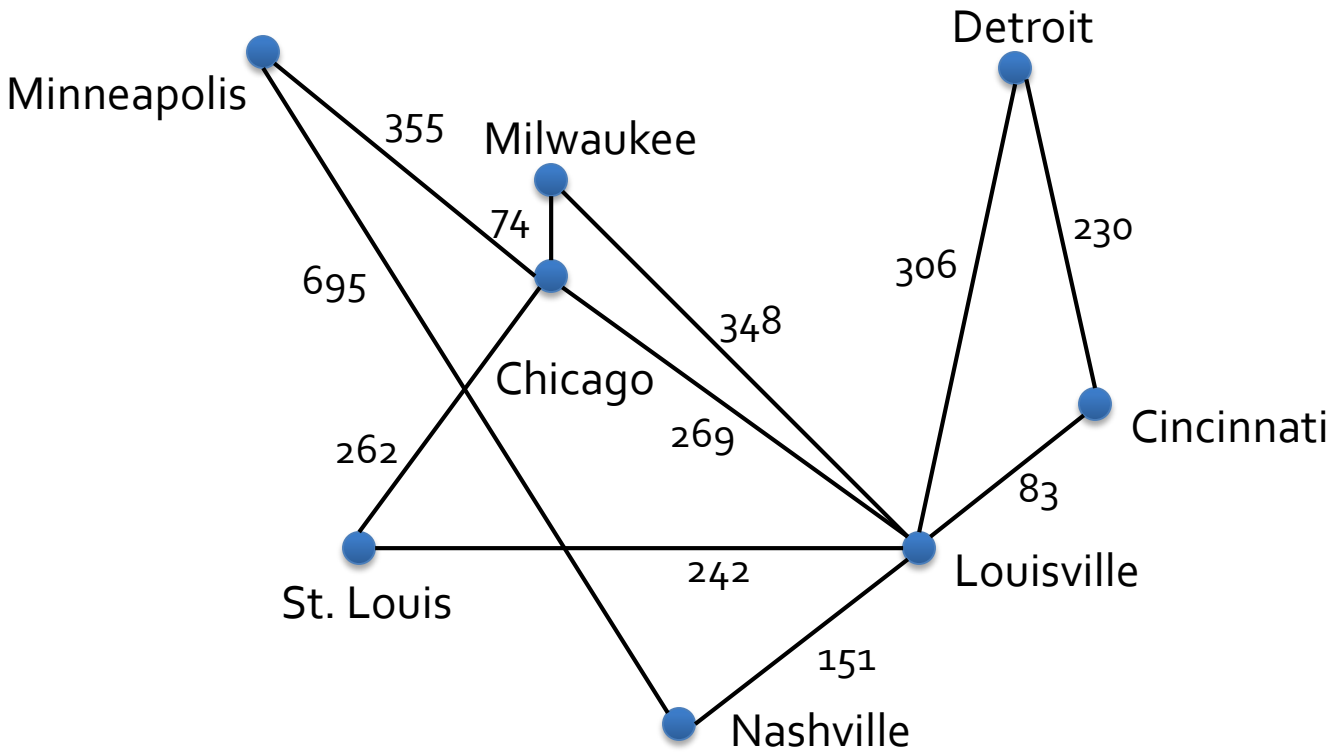
- A **spanning tree** for a graph G is a subgraph of G that contains every vertex of G and is a tree
- Some properties:
  - Every connected graph has a spanning tree
    - Why?
  - Any two spanning trees for a graph have the same number of edges
    - Why?

Weighted graphs

- In computer science, we often talk about **weighted graphs** when tackling practical applications
- A weighted graph is a graph for which each edge has a real number **weight**
- The sum of the weights of all the edges is the **total weight** of the graph
- Notation: If e is an edge in graph G, then $w(e)$ is the weight of e and $w(G)$ is the total weight of G
- A **minimum spanning tree** (**MST**) is a spanning tree of lowest possible total weight

Weighted graphs example

- Here is the graph from before, with labeled weights

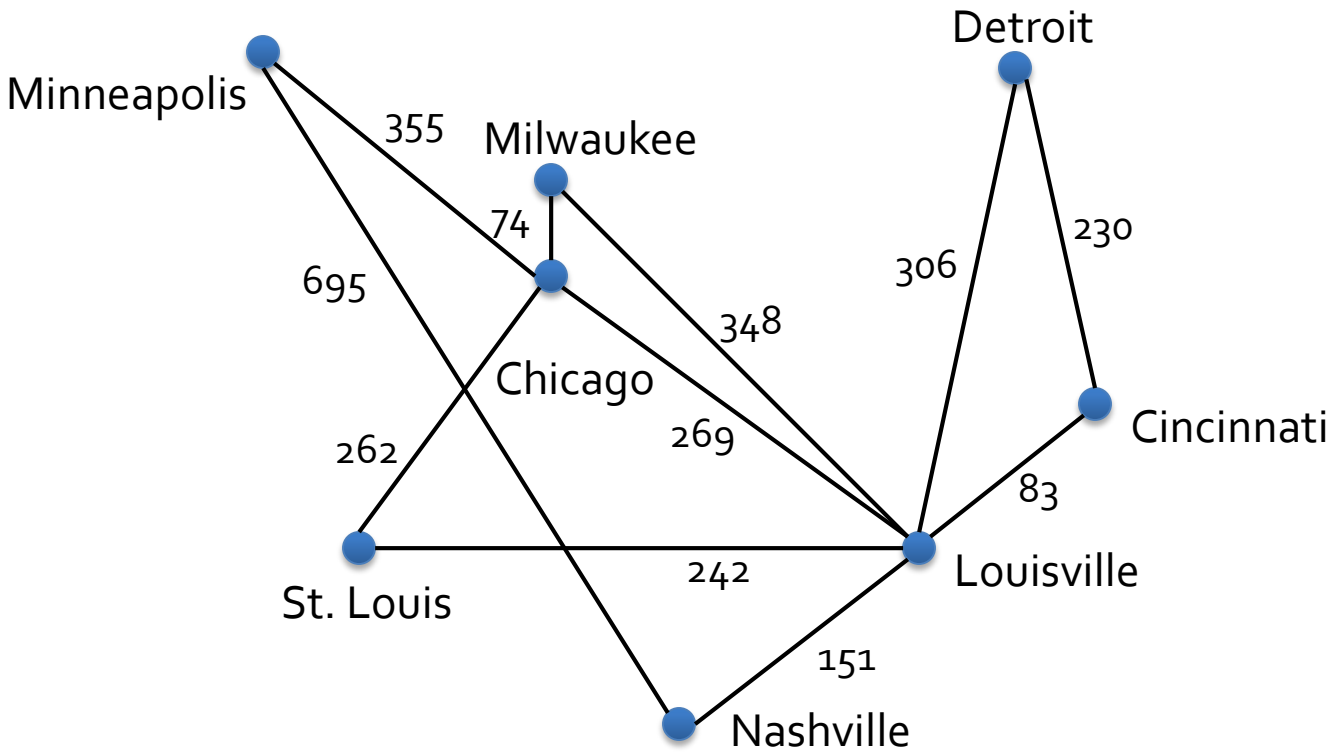


Finding a minimum spanning tree

- Kruskal's algorithm gives an easy to follow technique for finding an MST on a weighted, connected graph
- Informally, go through the edges, adding the smallest one, unless it forms a circuit
- **Algorithm:**
  - Input: Graph G with n vertices
  - Create a subgraph T with all the vertices of G (but no edges)
  - Let E be the set of all edges in G
  - Set $m = 0$
  - While $m < n - 1$
    - Find an edge e in E of least weight
    - Delete e from E
    - If adding e to T doesn't make a circuit
      - Add e to T
      - Set $m = m + 1$
  - Output: T

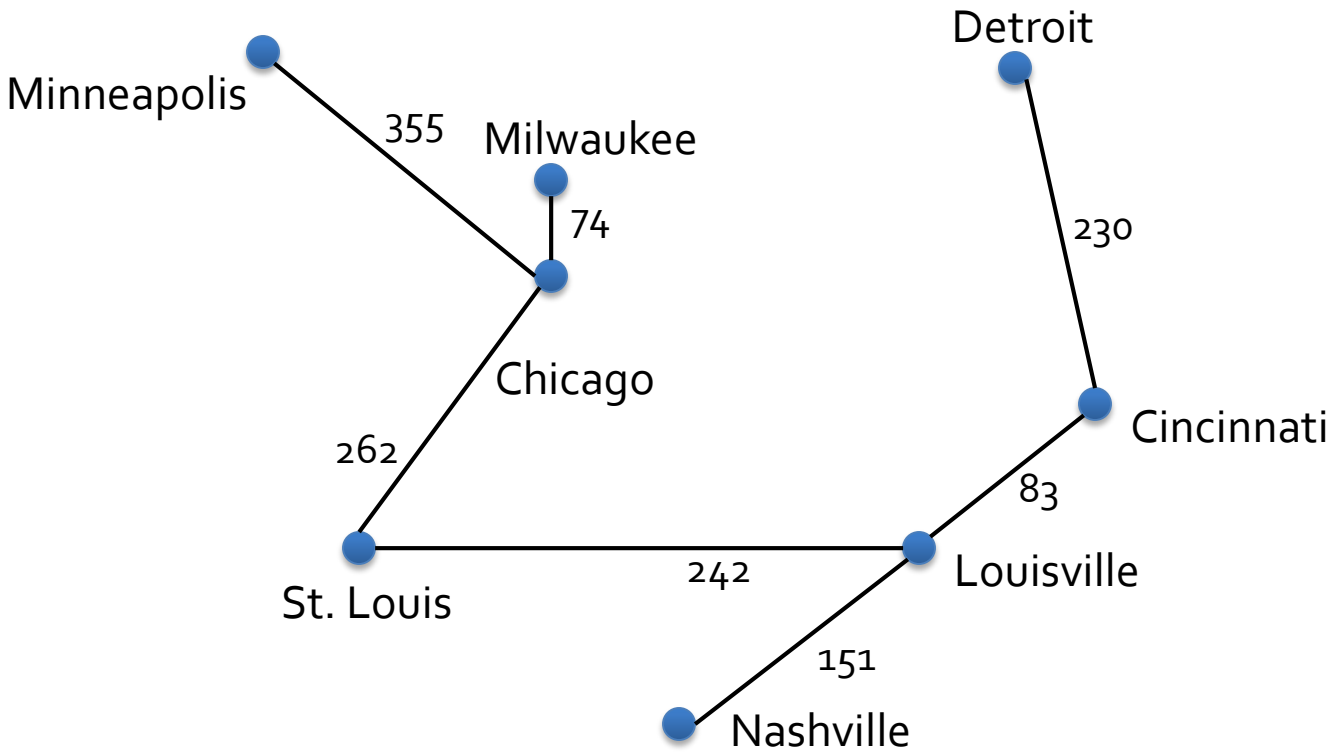
Minimum spanning tree example

- Run Kruskal's algorithm on the city graph:



Minimum spanning tree output

- Output:

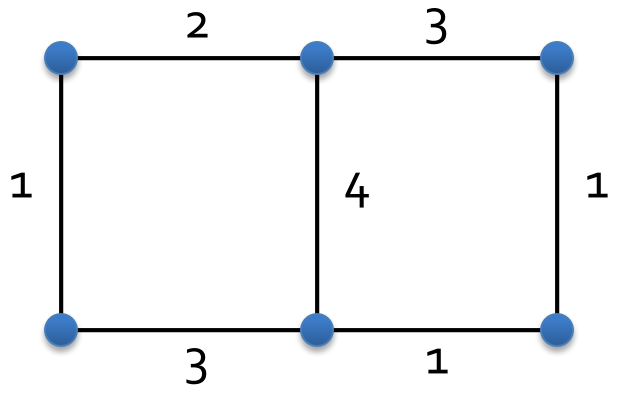


Prim's algorithm

- Prim's algorithm gives another way to find an MST
- Informally, start at a vertex and add the next closest node not already in the MST
- **Algorithm:**
  - Input: Graph G with n vertices
  - Let subgraph T contain a single vertex v from G
  - Let V be the set of all vertices in G except for v
  - For i from 1 to $n - 1$
    - Find an edge e in G such that:
      - e connects T to one of the vertices in V
      - e has the lowest weight of all such edges
    - Let w be the endpoint of e in V
    - Add e and w to T
    - Delete w from V
  - Output: T

Prim fights Kruskal

- Apply Kruskal's algorithm to the graph below
- Now, apply Prim's algorithm to the graph below
- Is there any other MST we could make?



Ticket Out the Door

Upcoming

Next time…

- Graphing functions
- Big-O, Big-Omega, and Big-Theta notation

Reminders

- **Work on Assignment 5**
- Read 11.1 and 11.2
  - Prepare a three-sentence summary
  - Extra credit if you get called on