Announcements
Huffman codes

Goal: Transmit a message using as few bits as possible.

Use frequency counts (so know the message ahead of time).

Huffman codes: pre-fix free

Why? So we can scan & decode—no ambiguity.
Can visualize as a binary tree

M: 111
I: 10
S: 0
P: 110

When reading string, just follow tree and output letter when you reach a leaf.

Ex: 010111110
    SIMP
Given \( n \) letters plus frequency counts for each letter, find the code minimizing total length:

\[
\sum_{i=1}^{n} f[i] \cdot \text{depth}(i) = \text{cost}(T)
\]
Pseudo code:

Keep characters in min heap, w/priority = to frequency.
L, R, & P keep track of left/right and parent indices.

\textbf{BuildHuffman}(f[1..n]):
\begin{verbatim}
  for i ← 1 to n
    L[i] ← 0; R[i] ← 0
    INSERT(i, f[i])
  for i ← n to 2n - 1
    x ← \textbf{ExtractMin}()
    y ← \textbf{ExtractMin}()
    f[i] ← f[x] + f[y]
    L[i] ← x; R[i] ← y
    P[x] ← i; P[y] ← i
    INSERT(i, f[i])
  P[2n - 1] ← 0
\end{verbatim}
Proof of Correctness:

**Lemma:** Let \( x \) and \( y \) be the 2 least frequent characters. Then there is an optimal code where \( x \) and \( y \) are siblings and have maximum depth in the tree.

Did proof in class last time.
Thm: Huffman codes are optimal.

pf: Induction on # of letters.

Base case: \( n = 1 \) (or 2) \( \checkmark \)

Inductive hypothesis: Given \( n \) characters, Huffman's alg. is optimal.

Inductive step: \( n \) characters with frequency counts \( f[1..n] \). Without loss of generality, assume \( f[1] \times f[2] \) are least frequent.

By our previous lemma, some optimal tree has \( | < 2 \) as siblings (at deepest level).

Let \( T' \) be the Huffman tree for \( f[3\ldots n+1] \).

\[ \rightarrow \text{By IH, cost}(T') \text{ is smallest possible for any such tree.} \]

Create \( T \) by removing leaf \( n+1 \) and replacing it with internal node with 2 children.

Why is \( T \) optimal?
Claim: This is optimal for $f[i...n]$. 

$$\text{cost}(T) = \sum_{i=1}^{n} f[i] \cdot \text{depth}(i)$$

$$= \sum_{i=3}^{n+1} f[i] \cdot \text{depth}(i) - f[n+1] \cdot \text{depth}(n+1) + f[2] \cdot \text{depth}(2)$$

$$= \text{cost}(T') + (f[1] + f[2]) \cdot \text{depth}(1) - f[n+1] \cdot \text{depth}(n+1)$$

(know $f[1] + f[2] = f[n+1]$, and depth(1) = depth(n+1) + 1)

$$\Rightarrow = \text{cost}(T') + f[1] + f[2].$$
So \[ \text{cost}(T) = \text{cost}(T') + f[i3] + f[2]. \]

Suppose \( T \) was not optimal.

In optimal tree, remove 1 & 2 & get a Huffman tree for 3...n+1.

If do that, get a tree for 3...n+1 that is better than \( T' \).

Contradiction.
Shortest paths in a graph. (4.4)

Suppose we have \( G = (V, E) \) and each edge \( e \in E \) has a length \( l_e \).

Here, we will assume \( G \) is directed: \( u \rightarrow v \).

If given undirected graph, how could we adapt to directed model?

\[ O(2|E|) = O(|E|) \]
Goal: Given two vertices, find shortest path between them.

Why? Mapquest!

Idea? BFS - rightides

$\text{distance} = 1$
We'll actually do something harder:

Given a source vertex $s$, compute shortest
path from $s$ to every other vertex.

The reason — if we don't explore every thing,
we don't know if we've missed a shorter path.
Greedy idea:

Start with a set \( S \).
(Initially \( S = \emptyset \))

At each step, grow out from \( S \), taking next shortest path from \( S \) to a new vertex \( u \) adding that to \( S \).