Wrapping Up DP

- A few words on traveling salesperson problem.
  - The problem.
  - Brute force algorithm.
  - Dynamic programming algorithm.
Greedy Algorithms

• If we can view our algorithm as making a series of choices, greedy algorithms:
  − Always make the choice that currently seems best.
  − Never go back to undo a previous choice.

• Change example...

• Greedy algorithms tend to be very efficient.

• It's not always possible to find a greedy algorithm.

• Greedy algorithms often make good approximate algorithms.
Minimum Spanning Trees

- Today we will consider undirected graphs.
- A tree is a graph with no cycles.
- In a connected graph, there is a path from every node to every other node.
- A spanning tree of a graph $G$, is a connected subgraph of $G$ that contains all of the vertices in $G$ and is a tree.
- A minimum spanning tree is a spanning tree with minimum weight.
- We will look at two minimum spanning tree algorithms.
Prim's Algorithm

- Let's go over an example...
Prim's Pseudocode

- Call the set of vertices already considered $Y$.
- We will maintain two arrays,
  - $\text{nearest}[i] =$ index of the vertex in $Y$ closest to $v_i$.
  - $\text{distance}[i] =$ weight on edge between $v_i$ and the vertex indexed by $\text{nearest}[i]$. (or -1 if $i$ is in $Y$).
Void prim(int n, number W[][], set_of_edges& F) {
    index vnear;
    number min;
    edge e;
    index nearest[2..n];  number distance[2..n];
    F = EMPTYSET;
    for (i = 2; i<=n i++) {
        nearest[i] = 1;
        distance[i] = W[i][1];
    }
    CONTINUED ON NEXT SLIDE...
repeat (n - 1 times) {
    min = ∞;
    for (i = 2; i <= n; i++) {
        if (0 <= distance[i] < min) {
            min = distance[i];
            vnear = i;
        }
    }
    e = (vnear, nearest[vnear]);
    add e to F;
    distance[vnear] = -1;
    for (i = 2; i <= n; i++) {
        if (W[i][vnear] < distance[i]) {
            distance[i] = W[i][vnear];
            nearest[i] = vnear;
        }
    }
}

Find the next vertex to add to Y.
Add the edge from our new vertex to Y to our MST.
Update distance and nearest to reflect the new Y.
Prim's Analysis

- $\Theta(n^2)$ every case. Why?
- Do you see any room for improvement?
- Is it guaranteed to find a minimum spanning tree?
Prim's Correctness Proof

• A subset of edges $F$ is **promising** if edges can be added to it to create a minimum spanning tree.

• Proof outline:
  - The empty set is promising.
  - Show (by induction) that with every edge Prim's adds, the set of edges remains promising.

• First a lemma (lemma 4.1 in the book):
  - Given a graph $G = (V, E)$ let $F$ be a promising subset of $E$. Let $Y$ be the vertices connected by the edges in $F$. If $e$ is an edge of minimum weight that connects a vertex in $Y$ to a vertex in $V - Y$, then $F \cup \{e\}$ is promising.
Proof of Lemma 4.1

• $F$ is promising, which means there must be some subset of edges $F'$ such that $F \subseteq F'$ and $(V, F')$ is an MST.

• Two cases:
  
  FIRST: If $\{e\} \in F'$ then $F \cup \{e\} \subseteq F'$ and $F \cup \{e\}$ is promising.

  SECOND: If $\{e\} \notin F'$ then $F' \cup \{e\}$ must contain a cycle.
  
  • Let's draw a picture...
  
  • There must be a vertex $\{e'\} \in F'$ that also connects $Y$ with $V-Y$.
  
  • The weight of $e$ and $e'$ must be the same.
  
  • $F' \cup \{e\} - \{e'\}$ is an MST.
  
  • $F \cup \{e\} \subseteq F' \cup \{e\} - \{e'\}$, so $F \cup \{e\}$ is promising.
Prim Proof Continued
Kruskal's Algorithm

• Let's do an example...
Handling Disjoint Sets

- We need a data structure for maintaining disjoint sets.
  - initial(n) – create n disjoint subsets, each of which contains exactly one of the indices between 1 and n.
  - p = find(i) – retrieve a pointer to the set containing i.
  - merge(p,q) – merge the two sets pointed to by p and q.
  - equal(p,q) – check to see if two pointers refer to the same set.

- If we initially create n disjoint sets, then call find, merge and equal c * t times the cost is: \( \Theta(t \lg t) \).

- The implementation and analysis of this is interesting in its own right, but we won't talk about it today.
Kruskal's Pseudocode

```c
int kruskal(int n, int m, set_of_edges E, set_of_edges& F) {
    set_pointer p, q;
    edge e;
    sort the m edges in E by weight;
    initial(n);
    while (# of edges in F < n-1) {
        e = next lowest weight edge;
        i, j = e.indices();
        p = find(i);
        q = find(j);
        if (!equal(p, q)) {
            merge(p, q);
            add e to F;
        }
    }
}
```
Kruskal's Analysis

- We want the complexity in terms of number of edges $m$, and number of vertices $n$.
- Three potentially time consuming phases:
  - Sorting.
  - Initializing the disjoint sets.
  - The while loop.
Kruskal's Analysis

- Three potentially time consuming phases:
  - Sorting.
    - \( \Theta(m \lg m) \)
  - Initializing the disjoint sets.
    - \( \Theta(n) \)
  - The while loop.
    - \( \Theta(m \lg m) \)
- Overall: \( \Theta(m \lg m) \), or \( \Theta(n^2 \lg n) \)
- Is it guaranteed to find the minimum spanning tree?
Dijkstra's Algorithm

• If time...