Today:
- Return, discuss Exam #1
- Dirac's and Ore's Theorems
- Closure of a Graph
- Traveling Salesperson Problem (TSP)
- Greedy Algorithms for TSP

Reading:
- [CH] 3.3-3.4

Exercises:
- [CH] p. 109: 3.3.11
- [CH] p. 119: Apply all the TSP methods discussed in today's slides to Problem 3.4.1.

Proving Dirac's and Ore's Theorems

Dirac's Theorem follows from Ore's Theorem

Prove Ore's Theorem by contradiction:
- Assume \( n \geq 3 \) and \( \deg(v) + \deg(w) \geq n \) for all non-adjacent vertices \( v, w \), but \( G \) is not hamiltonian.
- Add as many edges to \( G \) as possible, keeping it non-hamiltonian. Resulting graph (still call it \( G \)) is called maximal non-hamiltonian.

Why must each of the following statements be true?
- 1. There must be a pair of non-adjacent vertices \( v \) and \( w \).
- 2. If \( \deg(v) = r \), then \( \deg(w) \geq n-r \).
- 3. Add the edge \( v-w \) to \( G \); the resulting graph must now have a hamiltonian cycle.
- 4. There must be hamiltonian path \( P \) from \( v \) to \( w \).

Finishing proof of Ore's Theorem

On the Hamiltonian path \( P \) from \( v \) to \( w \):
- \( \deg(v) = r \), so connect \( v \) to all its \( r \) neighbors, \( v_1, ..., v_r \).
- For each \( v_i \) call its predecessor on \( P \) \( u_i \).
- \( \deg(w) = n - r \), so \( w \) must be adjacent to some \( u_i \)—why?
- If we remove the edge \( u_i-v_i \) and add the edge \( u_i-w \), we get a Hamiltonian cycle, contradicting our assumption!
Corollary of Ore’s Theorem;
Closure $c(G)$

- **Corollary:** Let $G$ be a simple graph with $n$ vertices and let $v$ and $w$ be non-adjacent vertices in $G$ such that $\deg(v) + \deg(w) \geq n$. Then $G$ is Hamiltonian if and only if the supergraph $G + \{v-w\}$ is Hamiltonian.
- **Proof:** Follows from proof of Ore’s Theorem.
- The closure $c(G)$ of a simple graph $G$ is the graph obtained by repeatedly joining non-adjacent vertices $u$ and $v$ that have $\deg(u) + \deg(v) \geq n$.
- **Theorem:** $G$ is Hamiltonian iff the closure $c(G)$ is Hamiltonian.
- **Examples:**

**Traveling Salesperson Problem**

- **Traveling Salesperson Problem (TSP for short):**
  - **Non-technical version:** A salesperson wants to design a route, beginning and ending at the home office, that will visit each client office exactly once, with minimum total distance traveled.
  - **Graph theory version:** Given the complete graph $K_n$ with weights on the edges, find the shortest cycle that includes all the vertices.
- Suppose we have an algorithm to solve TSP. How could we use it to find Hamiltonian cycles in a simple graph $G$?

**Graph Theory Problems with Varying Time Complexity**

- **Simple, low-degree polynomial algorithms:**
  - Most problems we’ve seen so far: Connectedness, minimal spanning tree, shortest path, Euler Circuit
- **More complex algorithms (in terms of design and/or degree), but still polynomial**
  - Planarity, maximum matching, $k$-connectivity
- **NP-hard:** No known polynomial algorithm; proven to be harder than most other problems
  - Hamiltonian Cycle, TSP, longest path, chromatic number, largest independent set, largest clique
- **Unknown complexity:** No known polynomial algorithm, but not known to be NP-hard
  - Graph isomorphism
Hamiltonian Cycles and bounds for the Traveling Salesperson Problem

- **TSP Input:** An edge-weighted complete graph \( G \)
- **TSP Question:** What is a Hamiltonian cycle with minimum total weight?
- The length of any Hamiltonian cycle is an upper bound.

**Theorem.** Let \( u \) be any vertex. Let \( w_1 \) and \( w_2 \) be the weights of the two smallest edges incident with \( u \), and let \( W \) be the weight of a minimum-weight spanning tree of \( G \). Then the sum, \( w_1 + w_2 + W \), is a lower bound on the length of a TSP tour.

Proof?

---

Nearest Neighbor and Cheapest Link Algorithms

- These are **greedy algorithms** – like Kruskal and Prim, but unlike them, not guaranteed to produce optimal results.
- **Nearest Neighbor:** Start at an arbitrary vertex \( v \), go to nearest unvisited neighbor, continue until all vertices are visited.
- **Repeated Nearest Neighbor:** Perform Nearest Neighbor starting at every vertex, then use the cheapest route.
- **Cheapest Link:** Repeatedly add the cheapest edge that doesn’t make any degree > 2, and that doesn’t prematurely complete the cycle

---

Some MC302 State Capitals

<table>
<thead>
<tr>
<th>State</th>
<th>DC</th>
<th>AL</th>
<th>FL</th>
<th>GA</th>
<th>NC</th>
<th>SC</th>
<th>TX</th>
<th>WA</th>
<th>WA-C</th>
<th>OR</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albany, NY</td>
<td>0</td>
<td>96</td>
<td>626</td>
<td>1429</td>
<td>293</td>
<td>113</td>
<td>186</td>
<td>293</td>
<td>113</td>
<td>186</td>
<td>293</td>
</tr>
<tr>
<td>Boston, MA</td>
<td>170</td>
<td>0</td>
<td>1162</td>
<td>700</td>
<td>700</td>
<td>700</td>
<td>700</td>
<td>700</td>
<td>700</td>
<td>700</td>
<td>700</td>
</tr>
<tr>
<td>Raleigh, NC</td>
<td>626</td>
<td>846</td>
<td>125</td>
<td>0</td>
<td>676</td>
<td>676</td>
<td>676</td>
<td>676</td>
<td>676</td>
<td>676</td>
<td>676</td>
</tr>
<tr>
<td>New Orleans, LA</td>
<td>1429</td>
<td>772</td>
<td>772</td>
<td>0</td>
<td>1139</td>
<td>1139</td>
<td>1139</td>
<td>1139</td>
<td>1139</td>
<td>1139</td>
<td>1139</td>
</tr>
<tr>
<td>Hartford, CT</td>
<td>113</td>
<td>1048</td>
<td>1048</td>
<td>1048</td>
<td>0</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
<td>113</td>
</tr>
<tr>
<td>Trenton, NJ</td>
<td>196</td>
<td>1048</td>
<td>1048</td>
<td>1048</td>
<td>1048</td>
<td>0</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>183</td>
</tr>
</tbody>
</table>
**Closest Insertion Algorithm**

- **General idea:** Keep replacing a cycle of length $k$ by one of length $k + 1$, by inserting a new vertex that is closest to one already in tour.
- **FACT:** If distances satisfy **Triangle Inequality**, then this is at worst twice the optimal total distance.
  1. $c_1$ = arbitrary vertex $v_1$.
  2. Repeat steps 3 and 4 until all vertices have been added:
  3. Find vertex $v$ that is closest to the current cycle, $c_j = v_{j_1} ... v_{j_l}$. ($c_1$ and $c_j$ are walks.)
  4. Create cycle $c_{i+1}$ by inserting $v$ in position that makes the new cycle as short as possible.