• **DISCLAIMER:** These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

• Richard gone this week for personal emergency

• Main topics:
  – Knapsack (book section 6.4).
  – All pairs shortest paths (book section 6.6).

1 **Knapsack**

**Problem:** Given set of items \( \{1, ..., n\} \), each with weight and value \((w_i, v_i)\) such that \(w_i\) is an integer. For a given \(W\) choose a set of items with total weight \(\leq W\) such that the total value is maximized.

**Can choose same item multiple times**

1. Let \(K[w]\) be the optimal solution of knapsack with \(W = w\).
2. Base case: \(K[0] = 0\).
3. States: \(K[w]\) with \(w \leq W\)
4. Transition: \(K[w] = \max_i \{K[w - w_i] + v_i : w - w_i \geq 0\}\)

<table>
<thead>
<tr>
<th>Knapsack with repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. (K[0] = 0)</td>
</tr>
<tr>
<td>2. For each (w = 1) to (W)</td>
</tr>
<tr>
<td>(a) (K[w] = \max_i {K[w - w_i] + v_i : w - w_i \geq 0}).</td>
</tr>
<tr>
<td>3. Return (K[W]).</td>
</tr>
</tbody>
</table>

1. Each transition takes \(O(n)\) time
2. The for loop iterates \(O(W)\) times
3. The full runtime is then $O(nW)$ time.

**Can only choose each item once**

1. Let $K[w, \{1, ..., i\}]$ be the optimal solution of knapsack with $W = w$, and the set of items $\{1, ..., i\}$

2. Base cases: $K[w, \{1\}] = v_1$ if $w - w_1 \geq 0$, and 0 otherwise

3. States: $K[w, \{1, ..., i\}]$ with $w \leq W$ and $i \leq n$

4. Transition: $K[w, \{1, ..., i\}] = \max\{K[w - w_i, \{1, ..., i - 1\}] + v_i, K[w, \{1, ..., i - 1\}]\}$

<table>
<thead>
<tr>
<th>Knapsack without repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $K[w, {1}] = v_1$ if $w - w_1 \geq 0$, 0 otherwise</td>
</tr>
<tr>
<td>2. For $i = 1$ to $n$</td>
</tr>
<tr>
<td>(a) For each $w = 1$ to $W$</td>
</tr>
<tr>
<td>i. $K[w, {1, ..., i}] = \max{K[w - w_i, {1, ..., i - 1}] + v_i, K[w, {1, ..., i - 1}]}$</td>
</tr>
<tr>
<td>3. Return $K[W, {1, ..., n}]$.</td>
</tr>
</tbody>
</table>

1. Each transition takes $O(1)$ time

2. The first for loop iterates $O(n)$ times

3. The second for loop iterates $O(W)$ times

4. The full runtime is then $O(nW)$ time.

2 All Pairs Shortest Path

**Problem:** Given a graph with edge weights, find the shortest path between all pairs of vertices.

**Apply Bellman-Ford**

The Bellman-Ford algorithm takes a fixed vertex $s$ and returns the shortest path between $s$ and any other vertex $v$ in the graph, taking $O(nm)$ time. Therefore, we could run Bellman-Ford for each vertex as $s$, obtaining the shortest path between all vertices in $O(n^2m)$ time.

**Dynamic Algorithm**

This running time can be improved to $O(n^3)$ time with the Floyd-Warshall algorithm by using dynamic programming.

The algorithm is based on the fact that each vertex appears on the shortest $ij$ path at most once.
1. State: \( d[i, j, k] \): shortest \( i \rightarrow j \) path only containing intermediate vertices \( 1 \ldots k \).

2. Base case: \( d[i, j, 0] = l(i, j) \).

3. Transition:

\[
d[i, j, k] = \min\{d[i, j, k - 1], d[i, k, k - 1] + d[k, j, k - 1]\}.
\]

4. Ordering: order by \( k \), any order of \( i, j \).

The motivation for the transition is that the path either

- does not use the vertex \( k \), in which case the maximum id on it is \( k - 1 \);

- or it uses \( k \), in which case the portions between \( i \) and \( k \), as well as \( k \) and \( j \), don’t use \( k \), and hence use only ids between 1 and \( k - 1 \).

This has \( O(n^3) \) states, but only \( O(1) \) time for transitions, giving a simple \( O(n^3) \) time algorithm.