Announcements

• Midterm today!
  – 60 minutes, start of class, closed book

• Reading for this week
  – 6.1, 6.2, 6.3, 6.4

• Makeup lecture
  – February 19, 6:30 pm.
    • Still waiting on confirmation on MS room.
Dynamic Programming

- Weighted Interval Scheduling
- Given a collection of intervals \( I_1, \ldots, I_n \) with weights \( w_1, \ldots, w_n \), choose a maximum weight set of non-overlapping intervals

\[ p[13] = 0, \quad p[23] = 0, \quad p[32] = 1 \]

\[ \text{Opt}_j \quad I_0, I_2, \ldots, I_j \]

\[ \text{Opt}_{j+1} - \text{function} \left( \text{Opt}_{j-1}, \text{Opt}_{j+2} \ldots \text{Opt}_n \right) \]
Optimality Condition

- $\text{Opt}[j]$ is the maximum weight independent set of intervals $I_1, I_2, \ldots, I_j$
- $\text{Opt}[j] = \max(\text{Opt}[j - 1], w_j + \text{Opt}[p[j]])$
  - Where $p[j]$ is the index of the last interval which finishes before $I_j$ starts

---

Include $	ext{Excl}$
Algorithm

MaxValue(j) =
    if j = 0 return 0
    else
        return max( MaxValue(j-1), w_j + MaxValue(p[j]) )

Worst case run time: $2^n$
A better algorithm

$M[j]$ initialized to -1 before the first recursive call for all $j$

MaxValue($j$) =

    if $j = 0$ return 0;
    else if $M[j] \neq -1$ return $M[j]$;
    else

        $M[j] = \max(\text{MaxValue}(j-1), w_j + \text{MaxValue}(p[j]))$;

    return $M[j]$;
Iterative version

MaxValue (j) {
    M[ 0 ] = 0;
    for (k = 1; k <= j; k++) {
        M[ k ] = max(M[ k-1 ], w_k + M[ P[ k ] ]);  
        return M[ j ];
    }
}
Fill in the array with the Opt values

\[ \text{Opt}[j] = \max \left( \text{Opt}[j-1], w_j + \text{Opt}[p[j]] \right) \]

\[
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
2 & 4 & 9 & 9 & 9 & 16 & 16
\end{array}
\]
Computing the solution

\[ \text{Opt}[j] = \max(\text{Opt}[j-1], w_j + \text{Opt}[p[j]]) \]

Record which case is used in Opt computation.
Dynamic Programming

- The most important algorithmic technique covered in CSEP 521
- Key ideas
  - Express solution in terms of a polynomial number of sub problems
  - Order sub problems to avoid recomputation
Optimal linear interpolation

\[ \text{Error} = \sum (y_i - ax_i - b)^2 \]
What is the optimal linear interpolation with three line segments
What is the optimal linear interpolation with two line segments?
What is the optimal linear interpolation with n line segments
Notation

- Points \( p_1, p_2, \ldots, p_n \) ordered by x-coordinate \( (p_i = (x_i, y_i)) \)
- \( E_{ij} \) is the least squares error for the optimal line interpolating \( p_i, \ldots, p_j \)
Optimal interpolation with two segments

- Give an equation for the optimal interpolation of $p_1, \ldots, p_n$ with two line segments

$$
\text{Opt}_2 = \min_k \left\{ E_{1,k} + E_{k,n} \right\}
$$

- $E_{i,j}$ is the least squares error for the optimal line interpolating $p_i, \ldots, p_j$
Optimal interpolation with k segments

- Optimal segmentation with three segments
  - $\min_{i,j}\{E_{1,i} + E_{i,j} + E_{j,n}\}$
  - $O(n^2)$ combinations considered

- Generalization to k segments leads to considering $O(n^{k-1})$ combinations
$\text{Opt}_k[j]$: Minimum error approximating $p_1\ldots p_j$ with $k$ segments

How do you express $\text{Opt}_k[j]$ in terms of $\text{Opt}_{k-1}[1],\ldots,\text{Opt}_{k-1}[j]$?
Optimal sub-solution property

Optimal solution with \( k \) segments extends an optimal solution of \( k-1 \) segments on a smaller problem.

Optimal use of \( k-1 \) segments for \( P_i \ldots P_j \).
Optimal multi-segment interpolation

Compute $\text{Opt}[k, j]$ for $0 < k < j < n$

for $j := 1$ to $n$
\[ \text{Opt}[1, j] = E_{1,j} \]
for $k := 2$ to $n-1$
for $j := 2$ to $n$
\[ t := E_{1,j} \]
for $i := 1$ to $j - 1$
\[ t = \min(t, \text{Opt}[k-1, i] + E_{i,j}) \]
\[ \text{Opt}[k, j] = t \]
Determining the solution

- When $\text{Opt}[k,j]$ is computed, record the value of $i$ that minimized the sum
- Store this value in an auxiliary array
- Use to reconstruct solution
Variable number of segments

- Segments not specified in advance
- Penalty function associated with segments
- Cost = Interpolation error + C x #Segments
Penalty cost measure

\[ \text{Opt} [ j ] = \min (E_{1,j}, \min_i (\text{Opt} [ i ] + E_{i,j} + P)) \]
Subset Sum Problem

• Let $w_1, \ldots, w_n = \{6, 8, 9, 11, 13, 16, 18, 24\}$
• Find a subset that has as large a sum as possible, without exceeding 50
Adding a variable for Weight

- $\text{Opt}[j, K]$ the largest subset of $\{w_1, \ldots, w_j\}$ that sums to at most $K$

- $\{2, 4, 7, 10\}$
  - $\text{Opt}[2, 7] = 6$
  - $\text{Opt}[3, 7] = 7$
  - $\text{Opt}[3, 12] = 11$
  - $\text{Opt}[4, 12] = 12$
Subset Sum Recurrence

- $\text{Opt}[j, K]$ the largest subset of $\{w_1, \ldots, w_j\}$ that sums to at most $K$

$$\text{Opt}[j, K] = \max \left[ \text{Opt}[j-1, K], w_j + \text{Opt}[j-1, K-w_j] \right]$$
Subset Sum Grid

\[ \text{Opt}[j, K] = \max(\text{Opt}[j - 1, K], \text{Opt}[j - 1, K - w_j] + w_j) \]

\[
\begin{array}{cccccccccccccccccccc}
4 & 0 & 2 & 2 & 4 & 4 & 6 & 7 & 7 & 7 & 9 & 10 & 11 & 12 & 13 & 14 & 14 & 16 & 17 \\
3 & 0 & 2 & 2 & 4 & 4 & 6 & 7 & 7 & 9 & 9 & 11 & 11 & 13 & 13 & 13 & 13 & 13 \\
2 & 0 & 2 & 2 & 4 & 4 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
1 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\{2, 4, 7, 10\}
Knapsack Problem

- Items have weights and values
- The problem is to maximize total value subject to a bound on weight
- Items \( \{l_1, l_2, \ldots, l_n\} \)
  - Weights \( \{w_1, w_2, \ldots, w_n\} \)
  - Values \( \{v_1, v_2, \ldots, v_n\} \)
  - Bound \( K \)
- Find set \( S \) of indices to:
  - Maximize \( \sum_{i \in S} v_i \) such that \( \sum_{i \in S} w_i \leq K \)
Knapsack Recurrence

Subset Sum Recurrence:

\[ \text{Opt}[j, K] = \max(\text{Opt}[j-1, K], \text{Opt}[j-1, K-w_j] + w_j) \]

Knapsack Recurrence:

\[ \text{Opt}[i, K] = \max(\text{Opt}[i-1, K], \text{Opt}[i-1, K-w_i] + v_i) \]
\[ K = \sum wi \]

**Knapsack Grid**

\[ \text{Opt}[j, K] = \max(\text{Opt}[j - 1, K], \text{Opt}[j - 1, K - w_j] + v_j) \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Weights \{2, 4, 7, 10\} \ Values: \{3, 5, 9, 16\}