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

Optimality Condition

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

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 $M[j] = -1$ return $M[j]$;
  else
    $M[j] = \max(\maxValue(j-1), w_j + \maxValue(p[j]));$
    return $M[j]$;

Iterative Algorithm

Express the MaxValue algorithm as an iterative algorithm

MaxValue {
  
}
Fill in the array with the Opt values

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

Dynamic Programming

- The most important algorithmic technique covered in CSE 421
- Key ideas
  - Express solution in terms of a polynomial number of sub problems
  - Order sub problems to avoid recomputation

Optimal linear interpolation

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

\[ \text{Notation} \]
- 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
- \( E_{ij} \) 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
  - \( \min_{i,j} (E_{1i} + E_{ij} + E_{jn}) \)
  - \( O(n^2) \) 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 multi-segment interpolation

Compute Opt\[k, j\] for 0 < k < j < n

for j := 1 to n
    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, Opt\[k-1, i\] + E_{i,j})
        Opt\[k, j\] = t

Determining the solution

- When Opt\[k,j\] is computed, record the value of i that minimized the sum
- Store this value in a 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

- Opt\[j\] = min(E_{1,j}, \min(\text{Opt}[i] + E_{i,j} + P))