Dynamic Programming:
Knapsack Problem and Sequence Alignment
CMSC 451, Summer 2009

Knapsack Problem

Given \( n \) objects and a "knapsack."

- Item \( i \) weighs \( w_i > 0 \) kilograms and has value \( v_i > 0 \).
- Knapsack has capacity of \( W \) kilograms.

Goal: fill knapsack so as to maximize total value.

Ex: \( \{3, 4\} \) has value 40.

Greedy: repeatedly add item with maximum ratio \( \frac{v_i}{w_i} \).

Ex: \( \{5, 2, 1\} \) achieves only value = 35 \( \Rightarrow \) greedy not optimal.

Knapsack Problem: Example

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
<th>( W = 11 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Knapsack Algorithm

Input: \( n, W, w_1, \ldots, w_n, v_1, \ldots, v_n \)

for \( w = 0 \) to \( W \)

\( M[0, w] = 0 \)

for \( i = 1 \) to \( n \)

for \( w = 1 \) to \( W \)

if \( w_i > w \)

\( M[i, w] = M[i-1, w] \)

else

\( M[i, w] = \max \{M[i-1, w], v_i + M[i-1, w-w_i]\} \)

return \( M[n, W] \)

Knapsack Problem: Proof of correctness

Proof of correctness.

- The algorithm examines the entire solution space and only the solution space by construction.
  - All possibilities are considered since we consider all paths associated with choosing or not choosing to pack each object.
  - The chosen solution is feasible since any item which puts us over the weight limit is not included.
- The algorithm chooses the maximum value over the solution space by construction, since the recurrence returns the max of its two recursive calls.

Knapsack Problem: Running Time

Running time: \( \Theta(nW) \).

It takes \( O(1) \) time to fill in each entry and there are \( nW \) such entries that are each filled in exactly once. Again, our progress measure is the number of nonempty entries, and this begins at 1 and increases by 1 until there are \( nW \) nonempty entries.

- Not polynomial in input size!
- "Pseudo-polynomial."
- Decision version of Knapsack is NP-complete. [Chapter 8]
  (Can a value of at least \( v \) be achieved without exceeding \( W \)?)

Knapsack approximation algorithm.

There exists a poly-time algorithm that produces a feasible solution that has value within 0.01% of optimum. [Section 11.8]
6.6 Sequence Alignment

Applications
- Basis for Unix diff
- Speech recognition
- Computational biology

- Gap penalty $\alpha$
- Mismatch penalty $\beta$
- Cost = sum of gap and mismatch penalties.

Sequence Alignment: Problem Structure

Def. $\text{OPTE}(i, j) = \min$ cost of aligning strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$
- Case 1: OPT matches $x_i y_j$
  - pay possible mismatch for $x_i y_j$, $\min$ cost of aligning two strings $x_1 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_{j-1}$
- Case 2a: OPT leaves $x_i$ unmatched.
  - pay gap for $x_i$, $\min$ cost of aligning $x_1 x_2 \ldots x_i y_1 y_2 \ldots y_{j-1}$
- Case 2b: OPT leaves $x_i$ unmatched.
  - pay gap for $y_j$, $\min$ cost of aligning $x_1 x_2 \ldots x_i y_1 y_2 \ldots y_{j-1}$

$$\text{OPTE}(i, j) = \begin{cases} 
\delta & \text{if } i = 0 \\
\delta & \text{if } j = 0 \\
\alpha + \text{OPTE}(i-1, j-1) & \text{if } i > 0 \text{ and } j > 0 \\
\alpha + \text{OPTE}(i-1, j) & \text{if } i > 0 \text{ and } j = 0 \\
\beta + \text{OPTE}(i, j-1) & \text{otherwise} \\
\end{cases}$$
Sequence Alignment: Example

```c
Sequence-Alignment(m, n, x1, ..., x_m, y1, ..., y_n, δ, α) {
    for i = 0 to m
        M[0, i] = iδ
    for j = 0 to n
        M[j, 0] = jδ
    for i = 1 to m
        for j = 1 to n
            M[i, j] = min(α[x_i, y_j] + M[i-1, j-1],
                           δ + M[i-1, j],
                           δ + M[i, j-1])
    return M[m, n]
}
```

X = CCGT  m=4  Y = CGTA  n=4
δ=2  a_w0  a_w1  a_w2  a_w3  a_x0  a_x1  a_x2  a_x3
α0  a_y0  a_y1  a_y2  a_y3

Sequence Alignment: Analysis

```c
Analysis. \(O(mn)\) time and space.
English words or sentences: \(m, n, m \leq 10\).
Computational biology: \(m = n = 100,000\). 10 billions ops OK, but 10GByte array?