Dynamic Programming:
Knapsack Problem and Sequence Alignment

CMSC 451, Summer 2009

Knapsack Problem

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.

<table>
<thead>
<tr>
<th>#</th>
<th>value</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

\( W = 11 \)

Greedy: repeatedly add item with maximum ratio \( v_i / w_i \).
Ex: \( \{ 5, 2, 1 \} \) achieves only value \( 35 \) \( \Rightarrow \) greedy not optimal.
Knapsack Problem: Example

**Input:** n, W, w₁,…,wₙ, v₁,…,vₙ

for w = 0 to W
  M[0, w] = 0

for i = 1 to n
  for w = 1 to W
    if (wᵢ > w)
      M[i, w] = M[i-1, w]
    else
      M[i, w] = max {M[i-1, w], vᵢ + M[i-1, w-wᵢ]}

return M[n, W]

---

### Knapsack Algorithm

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

\[ W = 11 \]

\[ \text{OPT: \{4, 3\}} \]

value = 22 + 18 = 40

\[ W = 11 \]
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

String Similarity

How similar are two strings?

- occurrence
- occurrence

6 mismatches, 1 gap

1 mismatch, 1 gap

0 mismatches, 3 gaps
Edit Distance

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

- Gap penalty $\delta$; mismatch penalty $\alpha_{pq}$.
- Cost = sum of gap and mismatch penalties.

\[ \alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA} \quad 2\delta + \alpha_{CA} \]

Sequence Alignment

Goal: Given two strings $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$ find alignment of minimum cost.

Def. An alignment $M$ is a set of ordered pairs $x_i$-$y_j$ such that each item occurs in at most one pair and no crossings.

Def. The pair $x_i$-$y_j$ and $x_{i'}$-$y_{j'}$ cross if $i < i'$, but $j > j'$.

\[
\text{cost}(M) = \sum_{(x_i, y_j) \in M} \alpha_{x_i y_j} + \sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta
\]

Ex: CTACCG vs. TACATG.
Sol: $M = x_2$-$y_1$, $x_3$-$y_2$, $x_4$-$y_3$, $x_5$-$y_4$, $x_6$-$y_6$. 
\[ C \quad T \quad A \quad C \quad C \quad - \quad G \]
\[ - \quad T \quad A \quad C \quad A \quad T \quad G \]

\[ y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \quad y_6 \]
Sequence Alignment: Problem Structure

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

$$OPT(i, j) = \begin{cases} 
  j\delta & \text{if } i = 0 \\
  \min \left\{ \alpha_{x_i y_j} + OPT(i-1, j-1), \right. \\
  \left. \delta + OPT(i-1, j), \delta + OPT(i, j-1) \right\} & \text{otherwise} \\
  i\delta & \text{if } j = 0 
\end{cases}$$

Sequence Alignment: Algorithm

```plaintext
Sequence-Alignment(m, n, x_1 x_2 \ldots x_m, y_1 y_2 \ldots y_n, \delta, \alpha) {
    for i = 0 to m
        M[0, i] = i\delta
    for j = 0 to n
        M[j, 0] = j\delta
    for i = 1 to m
        for j = 1 to n
            M[i, j] = \min(\alpha[x_i, y_j] + M[i-1, j-1], \delta + M[i-1, j], \delta + M[i, j-1])
    return M[m, n]
}
```
Sequence Alignment: Example

```c
Sequence-Alignment(m, n, x_1, x_2, ..., x_m, y_1, y_2, ..., 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]
}
```

\[
\begin{align*}
X &= \text{CCGT} & m &= 4 \\
Y &= \text{CGTA} & n &= 4 \\
\delta &= 2 & \alpha_{AA} &= 0 & \alpha_{AC} &= 4 & \alpha_{AG} &= 4 & \alpha_{AT} &= 1 & \alpha_{CC} &= 0 & \alpha_{CG} &= 1 & \alpha_{CT} &= 2 \\
\alpha_{GG} &= 0 & \alpha_{GT} &= 2 & \alpha_{TT} &= 0
\end{align*}
\]

Sequence Alignment: Analysis

```c
Sequence-Alignment(m, n, x_1, x_2, ..., x_m, y_1, y_2, ..., 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]
}
```

**Analysis.** \(\Theta(mn)\) time and space.

**English words or sentences:** \(m, n \leq 10\).

**Computational biology:** \(m = n = 100,000\). 10 billions ops OK, but 10GB array?