DYNAMIC PROGRAMMING: EVEN MORE FUN!

David Kauchak
CS 140 – Spring 2023

Admin
Assignment 5

Overall how is the class going

How difficult is the class
What's going well?

- The short clips at the start working with partners!
- I finally get to learn DP!
- The versatility of the PS because I feel like I'm practicing multiple different concepts
- The topic is genuinely interesting and I love thinking of algorithms, they remind me of puzzles.

What could be improved?

- Sometimes the pace of the lectures feel a bit fast
- No group sessions
- Late days
- The content feels too theoretical
- Less proofs, less inductions pls
- Possible Saturday mentor sessions

- It also feels like a level of background is expected from students, even though that background has not been built through previous Pomona CS classes so it feels very unfair to those of us who weren't exposed to CS beyond or before Pomona.
### Rod splitting example

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>16</td>
</tr>
</tbody>
</table>

\[
R(n) = \max_{i=1}^{n} (p_i + R(n-i))
\]

```
R  0 1 2 3 4 5 6 7 8 9 10 11 12
0   1   2   6
```

**Choice:** 1

---

1: 1 + R[5] = 6

```
R  0 1 2 3 4 5 6 7 8 9 10 11 12
0   1   2   6   7
```

**Choice:** 1 1 3
<table>
<thead>
<tr>
<th>Rod splitting example</th>
<th>Rod splitting example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>length</strong>: 1 3 5 6 8</td>
<td><strong>length</strong>: 1 3 5 6 8</td>
</tr>
<tr>
<td><strong>price</strong>: 1 6 9 13 16</td>
<td><strong>price</strong>: 1 6 9 13 16</td>
</tr>
</tbody>
</table>

<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>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</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>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Choice:</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**5:** $9 + R[0] = 9$

**3:** $6 + R[2] = 8$

**1:** $1 + R[4] = 8$

**Choice:** 1 1 3 3 5

---

**6:** $13 + R[1] = 13$

**5:** $9 + R[2] = 12$

**3:** $6 + R[4] = 10$

**1:** $1 + R[6] = 10$

Choice: 1 1 3 3 5 6

---

**6:** $13 + R[1] = 14$

**5:** $9 + R[2] = 15$

**3:** $6 + R[4] = 15$

**1:** $1 + R[7] = 15$

Choice: 1 1 3 3 5 6 8

---

**8:** $16 + R[0] = 16$

**6:** $13 + R[2] = 15$

**5:** $9 + R[4] = 15$

**3:** $6 + R[5] = 15$

**1:** $1 + R[7] = 15$

Choice: 1 1 3 3 5 6 8
Rod splitting example

| length: | 1 | 3 | 5 | 6 | 8 |
| price:  | 1 | 6 | 9 | 13 | 16 |

What cuts do we make?

0 1 2 6 7 9 13 14 16 19 20 22 26
R 0 1 2 3 4 5 6 7 8 9 10 11 12
Choice: 1 1 3 3 5 6 6 8 6 6 8 6

What cuts do we make?

0 1 2 6 7 9 13 14 16 19 20 22 26
R 0 1 2 3 4 5 6 7 8 9 10 11 12
Choice: 1 1 3 3 5 6 6 8 6 6 8 6

What cuts do we make?

0 1 2 6 7 9 13 14 16 19 20 22 26
R 0 1 2 3 4 5 6 7 8 9 10 11 12
Choice: 1 1 3 3 5 6 6 8 6 6 8 6

What cuts do we make?

0 1 2 6 7 9 13 14 16 19 20 22 26
R 0 1 2 3 4 5 6 7 8 9 10 11 12
Choice: 1 1 3 3 5 6 6 8 6 6 8 6

What cuts do we make?

0 1 2 6 7 9 13 14 16 19 20 22 26
R 0 1 2 3 4 5 6 7 8 9 10 11 12
Choice: 1 1 3 3 5 6 6 8 6 6 8 6

What cuts do we make?

0 1 2 6 7 9 13 14 16 19 20 22 26
R 0 1 2 3 4 5 6 7 8 9 10 11 12
Choice: 1 1 3 3 5 6 6 8 6 6 8 6

What cuts do we make?

0 1 2 6 7 9 13 14 16 19 20 22 26
R 0 1 2 3 4 5 6 7 8 9 10 11 12
Choice: 1 1 3 3 5 6 6 8 6 6 8 6
Rod splitting example

<table>
<thead>
<tr>
<th>length</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3 5 6 8</td>
<td>1 6 9 13 16</td>
</tr>
</tbody>
</table>

What cuts do we make?

<table>
<thead>
<tr>
<th>R</th>
<th>0 1 2 3 4 5 6 7 8 9 10 11 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choice</td>
<td>1 1 3 3 5 6 6 8 6 8 6 8 6</td>
</tr>
</tbody>
</table>

Longest increasing subsequence

Given a sequence of numbers $X = x_1, x_2, \ldots, x_n$ find the longest increasing subsequence $[i_1, i_2, \ldots, i_m]$, i.e., a subsequence where numbers in the sequence increase.

$5 2 8 6 3 6 9 7$
1b: recursive solution

5 2 8 6 3 6 9 7

Is 5 part of the LIS?

Two options:
Either 5 is in the LIS or it’s not

1b: recursive solution

5 2 8 6 3 6 9 7

include 5

5 + LIS(8 6 3 6 9 7)

What is this function exactly?

longest increasing sequence of the numbers

longest increasing sequence of the numbers starting with 8
What is this function exactly?

longest increasing sequence of the numbers starting with 8

Do we need to consider anything else for subsequences starting at 5?

Anything else?

Technically, this is fine, but now we have $\text{LIS}$ and $\text{LIS}'$ to worry about.

Can we rewrite $\text{LIS}$ in terms of $\text{LIS}'$?
1b: recursive solution

\[ LIS(X) = \max_i \{LIS'(i)\} \]

Longest increasing sequence for \( X \)

is the longest increasing sequence

starting at any element

And what is \( LIS' \) defined as (recursively)?

\[ LIS'(i) = \begin{cases} 1 + \max_{j : i < j \text{ and } x_j > x_i} LIS'(j) \\ \end{cases} \]

Longest increasing sequence starting at \( i \)

2: DP solution (bottom-up)

\[ LIS'(i) = 1 + \max_{j : i < j \text{ and } x_j > x_i} LIS'(j) \]

\[ LIS': \]

5 2 8 6 3 6 9 7

\[ LIS': \]

5 2 8 6 3 6 9 7

\[ LIS': \]

1
2: DP solution (bottom-up)

\[ \text{LIS}'(i) = 1 + \max_{j : i < j \text{ and } x_i > x_j} \text{LIS}'(j) \]

\[ \text{LIS}' = \begin{cases} 1 \\ 5 \ 2 \ 8 \ 6 \ 3 \ 6 \ 9 \ 7 \end{cases} \]
2: DP solution (bottom-up)

\[ LIS'(l) = 1 + \max_{j : x_j < x_l \text{ and } x_j > x_l} LIS'(j) \]

\[ LIS' = [3, 2, 1, 1, 5, 2, 8, 6, 3, 6, 9, 7] \]

2: DP solution (bottom-up)

\[ LIS'(l) = 1 + \max_{j : x_j < x_l \text{ and } x_j > x_l} LIS'(j) \]

\[ LIS' = [2, 3, 2, 1, 1, 5, 2, 8, 6, 3, 6, 9, 7] \]

2: DP solution (bottom-up)

\[ LIS'(l) = 1 + \max_{j : x_j < x_l \text{ and } x_j > x_l} LIS'(j) \]

\[ LIS' = [2, 2, 3, 2, 1, 1, 5, 2, 8, 6, 3, 6, 9, 7] \]

2: DP solution (bottom-up)

\[ LIS'(l) = 1 + \max_{j : x_j < x_l \text{ and } x_j > x_l} LIS'(j) \]

\[ LIS' = [4, 2, 2, 3, 2, 1, 1, 5, 2, 8, 6, 3, 6, 9, 7] \]
What does the table for storing answers look like?

1-D array: only one thing changes for recursive calls
2: DP solution (bottom-up)

\[ LIS'(i) = 1 + \max_{j<i \text{ and } x_j > x_i} LIS'(j) \]

What are the “smallest” possible subproblems?

To calculate LIS'(n), what are all the subproblems we need to calculate? This is the “table”.

How should we fill in the table?

Where will the answer be?

\[ LIS'(i) = 1 + \max_{j<i \text{ and } x_j > x_i} LIS'(j) \]
2: DP solution (bottom-up)

```
\text{LIS}(X)
1 \quad n \gets \text{len}(X)
2 \quad \text{create array } L\text{is with } n \text{ entries}
3 \quad \text{for } i \gets 1 \text{ to } n
4 \quad \text{\hspace{1cm} } \text{max} \gets 1
5 \quad \text{\hspace{1cm} for } j \gets i - 1 \text{ \textbf{to 0}}
6 \quad \text{\hspace{1.5cm} if } X[j] > X[i]
7 \quad \text{\hspace{1.5cm} \text{\hspace{1cm} max} \gets \text{\bf 1} + \text{\bf max}}
8 \quad \text{\hspace{1.5cm} } L\text{is}[i] \gets j
9 \quad \text{\hspace{1cm} } \text{\bf return } \text{\bf max}
10 \text{\bf max} \gets 0
11 \text{\bf for } c \gets 1 \text{ \textbf{to } } n
12 \quad \text{\bf if } L\text{is} > \text{\bf max}
13 \quad \text{\bf max} \gets L\text{is}
14 \text{\bf return } \text{\bf max}
```

3: Analysis

**Space requirements:** $\Theta(n)$

**Running time:** $\Theta(n^2)$
Can we use LCS to solve this problem?

5 2 8 6 3 6 9 7
2 3 5 6 6 7 8 9

Another solution

Edit distance (aka Levenshtein distance)

Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string $s_1$ into string $s_2$.

Insertion:

ABACED → ABACED → DABACED

Insert "C" Insert "D"

Deletion:

ABACED
Edit distance (aka Levenshtein distance)

Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string $s_1$ into string $s_2$.

**Deletion:**

$ABACED \rightarrow BACED$

Delete 'A'.

**Substitution:**

$ABACED \rightarrow ABACED \rightarrow ABADES$

Sub 'D' for 'C' then 'S' for 'D'.

**Example:**

$Edit(Kitten, Mitten) = 1$

Operations:

Sub 'M' for 'K' then 'Mitten'.
Edit distance examples

Edit(Happy, Hilly) = 3

Operations:
Sub ‘a’ for ‘i’ Happy
Sub ‘i’ for ‘p’ Hippy
Sub ‘i’ for ‘p’ Hilly

Edit distance examples

Edit(Banana, Car) = 5

Operations:
Delete ‘B’ anana
Delete ‘a’ nana
Delete ‘n’ nna
Sub ‘C’ for ‘n’ Caa
Sub ‘a’ for ‘r’ Car

Edit distance examples

Edit(Simple, Apple) = 3

Operations:
Delete ‘S’ imple
Sub ‘A’ for ‘i’ Ample
Sub ‘m’ for ‘p’ Apple

Why might this be useful?
Is edit distance symmetric?

Is edit distance symmetric? 
that is, is Edit(s₁, s₂) = Edit(s₂, s₁)?

\[ \text{Edit}(	ext{Simple, Apple}) =? \text{Edit}(	ext{Apple, Simple}) \]

Why?
- sub 'i' for 'j' \( \rightarrow \) sub 'j' for 'i'
- delete 'i' \( \rightarrow \) insert 'i'
- insert 'i' \( \rightarrow \) delete 'i'

Calculating edit distance

\[ X = A B C B D A B \]
\[ Y = B D C A B A \]

Ideas? How can we break this into subproblems?

Calculating edit distance

\[ X = A B C B D A \]
\[ Y = B D C A B \]

After all of the operations, X needs to equal Y
Start with the last two characters

Calculating edit distance

\[ X = A B C B D A \]
\[ Y = B D C A B \]

Operations: Insert Delete Substitute
Assume they’re different
How can we make them the same?
How can we use insert to transform X into Y?

Insert the last character of Y to the end of X

How does this make the problem smaller?

Edit(Y, X) = 1 + Edit(X_{i\to}, Y_{(j-1)\to})
Delete

\[ X = A B C B D A \? \]
\[ Y = B D C A B \? \]

How can we use delete to transform X into Y?

\[ \text{Edit}(X, Y) = 1 + \text{Edit}(X_{1..n-1}, Y_{1..m}) \]

Substitution

\[ X = A B C B D A \? \]
\[ Y = B D C A B \? \]

How can we use substitution to transform X into Y?

\[ \text{Edit}(X, Y) = 1 + \text{Edit}(X_{1..n-1}, Y_{1..m}) \]
Anything else?

Equal

\[ \begin{align*}
X &= A \ B \ C \ B \ D \ A \ ? \\
Y &= B \ D \ C \ A \ B \ ?
\end{align*} \]

What if the last characters are equal?

1b: recursive solution - combining results

Equal

\[ \begin{align*}
X &= A \ B \ C \ B \ D \ A \ ? \\
Y &= B \ D \ C \ A \ B \ ?
\end{align*} \]

How do we decide between these?
1b: recursive solution - combining results

\[
\begin{align*}
\text{Edit}(X, Y) &= \min \left\{ 
\begin{array}{ll}
1 + \text{Edit}(X_{1..m}, Y_{1..n}) & \text{insertion} \\
1 + \text{Edit}(X_{1..m}, Y_{1..n-1}) & \text{deletion} \\
D(Y_{1..n}, \text{Edit}(X_{1..m}, Y_{1..n-1})) & \text{equal/substitution}
\end{array} \right. \\
& \quad \text{if they're different} \\
& \quad 0 \quad \text{if they're the same}
\end{align*}
\]

2: DP solution (bottom-up)

What does the table for storing answers look like?

What are the "smallest" possible subproblems?

To calculate \(d(i, m)\), what are all the subproblems we need to calculate? This is the "table".

How should we fill in the table?

Where will the answer be?
2: DP solution (bottom-up)

\[
\begin{align*}
\text{Edit}(X,Y) &= \min \\
&= \begin{cases} 
1 + \text{Edit}(X_{1..n}, Y_{1..m}) & \text{insertion} \\
1 + \text{Edit}(X_{1..n}, Y_{1..m}) & \text{deletion} \\
\text{Diff}(X_{1..n}, Y_{1..m}) + \text{Edit}(X_{1..n}, Y_{1..m}) & \text{equal/insertion} \\
\end{cases}
\end{align*}
\]

What are the “smallest” possible subproblems?
\text{Edit}(\text{X}, \text{""}) = |\text{X}| \quad \text{and} \quad \text{Edit}(\text{""}, \text{Y}) = |\text{Y}|

To calculate \(d_{i,j}\), what are all the subproblems we need to calculate? This is the “table”.

\[
i < n \quad \text{and} \quad j < m
\]

How should we fill in the table?

\[
i = 1 \ldots , \quad j = 1 \ldots 
\]

Where will the answer be?

\[
d_{n,m}
\]

3: analysis

\[
\begin{align*}
\text{Edit}(X,Y) &= \min \\
&= \begin{cases} 
1 + \text{Edit}(X_{1..n}, Y_{1..m}) & \text{insertion} \\
1 + \text{Edit}(X_{1..n}, Y_{1..m}) & \text{deletion} \\
\text{Edit}(X_{1..n}, Y_{1..m}) & \text{equal/insertion} \\
\end{cases}
\end{align*}
\]

Space requirements: \(\Theta(nm)\)

Running time: \(\Theta(nm)\)

2: DP solution (bottom-up)

\[
\begin{align*}
\text{Edit}(X,Y) &= \min \\
&= \begin{cases} 
1 + \text{Edit}(X_{1..n}, Y_{1..m}) & \text{insertion} \\
1 + \text{Edit}(X_{1..n}, Y_{1..m}) & \text{deletion} \\
\text{Diff}(X_{1..n}, Y_{1..m}) + \text{Edit}(X_{1..n}, Y_{1..m}) & \text{equal/insertion} \\
\end{cases}
\end{align*}
\]

Space requirements: \(\Theta(nm)\)

Running time: \(\Theta(nm)\)
Edit distance variants

- Only include insertions and deletions
  - What does this do to substitutions?
- Include swaps, i.e., swapping two adjacent characters counts as one edit
- Weight insertion, deletion and substitution differently
- Weight specific character insertion, deletion and substitutions differently
- Length normalize the edit distance

DP in practice

For each aligned paragraph pair (i.e., a simple paragraph and one or more normal paragraphs), we then used a dynamic programming approach to find that best global sentence alignment following the framework laid out by Ethayarajh (2003). Specifically, given a normal sentence to align to, we normalize the edit distance $d(n, m)$ using the following recurrence:

$$d(i, j) =
\begin{cases}
    a(i, j - 1) + \text{sub} & \\
    a(i - 1, j) + \text{add} & \\
    a(i - 1, j - 1) + \text{match} & \\
    a(i - 1, j) + \text{del} & \\
    a(i, j - 1) + \text{ins} &
\end{cases}
$$