Mentor hours this week

Thursday: 6-8pm (Aidan)
Friday: 1-3pm (Emily)
Saturday: 9:30-11:30am (Millie)
Sunday: 7-9pm (Carl), 8-10pm (Alan)

LC meetings

Thursday:
- 8-9pm (Emily—Edmunds upstairs, Carl—Edmunds upstairs)

Friday:
- 9-10am (Millie—Edmunds downstairs)
- 2-3pm (Jiahao—Edmunds downstairs, Aidan)
- 3-4pm (Jiahao—Edmunds downstairs)
- 4-5pm (Millie)
Longest increasing subsequence
Given a sequence of numbers $X = x_1, x_2, \ldots, x_n$ find the longest increasing subsequence $(i_1, i_2, \ldots, i_m)$, that is a subsequence where numbers in the sequence increase.

5 2 8 6 3 6 9 7

1a: optimal substructure
Prove: optimal solutions to the problem incorporate optimal solutions to related subproblems

5 2 8 6 3 6 9 7

What would a solution to a subproblem look like?
1a: optimal substructure

Prove: optimal solutions to the problem incorporate optimal solutions to related subproblems

Proof by contradiction:
Assume: \( \{i_1, i_2, i_3, \ldots, i_m\} \) is a solution to \( x_1 \ldots x_n \) but \( \{i_2, i_3, \ldots, i_m\} \) is not a solution to \( x_1 \ldots x_n \).

Then some solution to \( x_{i_1} \ldots x_n \) exists, \( \{i'_2, i'_3, \ldots, i'_k\} \) where \( k > m \).

We could create a solution \( \{i_1, i'_2, i'_3, \ldots, i'_k\} \) to the original problem that is a better solution ... contradiction.

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

include 5

5 + LIS(8 6 3 6 9 7)
What is this function exactly?

longest increasing sequence of the numbers

This would allow for the option of sequences starting with 3 which are NOT valid!

Do we need to consider anything else for subsequences starting at 5?
1b: recursive solution

\[ 5 \ 2 \ 8 \ 6 \ 3 \ 6 \ 9 \ 7 \]

\[ \text{LIS}(2 \ 8 \ 6 \ 3 \ 6 \ 9 \ 7) \]

Anything else?

Technically, this is fine, but now we have LIS and LIS' to worry about.

Can we rewrite LIS in terms of LIS'?

---

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

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'):

\[ 5 \ 2 \ 8 \ 6 \ 3 \ 6 \ 9 \ 7 \]
2: DP solution (bottom-up)

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

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

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

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

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

$LIS' = \begin{bmatrix}
4 & 2 & 2 & 3 & 2 & 1 & 1 \\
5 & 2 & 8 & 6 & 3 & 6 & 9 & 7
\end{bmatrix}$

$LIS(i) = \max_i \{LIS'(i)\}$

What does the data structure for storing answers look like?
2: DP solution (bottom-up)

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

1-D array: only one thing changes for recursive calls

What are the “smallest” possible subproblems?
LIS'(n) and that is well-defined for this problem
To calculate LIS'(i), 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?

```plaintext
max[LIS'(1), ..., LIS'(n)]
```
2: DP solution (bottom-up)

LIS(X)
1. \texttt{u = \textit{length}(X)}
2. \texttt{create array \textit{lis} with n entries}
3. for \texttt{i = n to 1}
   4. \texttt{\textit{max} = 1}
   5. for \texttt{j = i + 1 to n}
      6. if \texttt{\textit{X}[j] > \textit{X}[i]} \texttt{or \textit{lis}[j] > \textit{max}}
         7. \texttt{\textit{max} = 1 + \textit{lis}[j]}
   8. \texttt{\textit{lis}[i] = \textit{max}}
9. \texttt{\max = 0}
10. for \texttt{i = 1 to n}
11. if \texttt{\textit{lis}[i] > \max}
12. \texttt{\max = \textit{lis}[i]}
13. \texttt{return \max}

\textbf{start from the end (bottom)}

3: Analysis

Space requirements?
Running time?
3: Analysis

Space requirements: $\Theta(n)$

Running time: $\Theta(n^2)$

Another solution

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

Can we use LCS to solve this problem?

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

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  ➔  ABACCEED  ➔  DABACCED
```

Insert "C"

Insert "D"
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

**Substitution:**

ABACED $\rightarrow$ ABADE $\rightarrow$ ABADES

Sub ‘D’ for ‘C’  Sub ‘S’ for ‘D’
Edit distance examples

Edit(Kitten, Mitten) = 1

Operations:
Sub ‘M’ for ‘K’ Mitten

Edit distance examples

Edit(Happy, Hilly) = 3

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

Edit distance examples

Edit(Banana, Car) = 5

Operations:
Delete ‘B’ anana
Delete ‘a’ nana
Delete ‘n’ naa
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’ Appple
Sub ‘m’ for ‘p’ Apple
Edit distance

Why might this be useful?

Is edit distance symmetric?

that is, is Edit(s_1, s_2) = Edit(s_2, s_1)?

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

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?

Insert

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

How can we use insert to transform \( X \) into \( Y \)?

57

Insert

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

insert the last character of \( Y \) to the end of \( X \)

59

Insert

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

How does this make the problem smaller?

60
How can we use delete to transform $X$ into $Y$?

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

How can we use substitution to transform $X$ into $Y$?

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

\[ X = \textcolor{red}{A B C B D A} ? \]

\( \text{Edit} \)

\[ Y = \textcolor{red}{B D C A B} ? \]

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

Anything else?

\[ X = \textcolor{red}{A B C B D A} ? \]

\[ Y = \textcolor{red}{B D C A B} ? \]

Equal

\[ X = \textcolor{red}{A B C B D A} ? \]

\[ Y = \textcolor{red}{B D C A B} ? \]

What if the last characters are equal?

\[ Edit(X, Y) = Edit(X_{1..n-1}, Y_{1..m-1}) \]
1b: recursive solution - combining results

Insert: \( \text{Edit}(X, Y) = 1 + \text{Edit}(X_{1:n}, Y_{1:m-1}) \)

Delete: \( \text{Edit}(X, Y) = 1 + \text{Edit}(X_{1:n-1}, Y_{1:m}) \)

\( X_n \neq Y_m \) Substitute: \( \text{Edit}(X, Y) = 1 + \text{Edit}(X_{1:n-1}, Y_{1:m-1}) \)

\( X_n = Y_m \) Equal: \( \text{Edit}(X, Y) = \text{Edit}(X_{1:n-1}, Y_{1:m-1}) \)

How do we decide between these?

2: DP solution (bottom-up)

\[
\text{Edit}(X, Y) = \min \begin{cases} 
1 + \text{Edit}(X_{i+1}, Y_{j+1}) & \text{insertion} \\
1 + \text{Edit}(X_{i}, Y_{j+1}) & \text{deletion} \\
\text{Diff}(x_i, y_j) + \text{Edit}(X_{i+1}, Y_{j+1}) & \text{equal/substitution}
\end{cases}
\]

What does the data structure for storing answers look like?

\[
\text{Edit}(X_{1:i}, Y_{1:j})
\]

\( d[i, j] \): edit distance between \( X_{1:i} \) and \( Y_{1:j} \)
2: DP solution (bottom-up)

Edit(X, Y) = min
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [ Edit(X, Y) ]</td>
<td>insertion 1 + Edit(X[-1], Y)</td>
</tr>
</tbody>
</table>

What are the “smallest” possible subproblems?

To calculate \( d(n, m) \), 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?

Space requirements?

Running time?

3: analysis

Edit(X, Y) = min
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [ Edit(X, Y) ]</td>
<td>insertion 1 + Edit(X[-1], Y)</td>
</tr>
</tbody>
</table>

Edit(X, Y)
1 \[ m \rightarrow \text{length}(X) \]
2 \[ n \rightarrow \text{length}(Y) \]
3 for \( i \rightarrow 0 \) to \( m \)
4 \[ d[i, 0] \rightarrow i \]
5 for \( j \rightarrow 0 \) to \( n \)
6 \[ d[0, j] \rightarrow j \]
7 for \( i \rightarrow 1 \) to \( m \)
8 for \( j \rightarrow 1 \) to \( n \)
9 \[ d[i, j] = \text{min}(1 + d[i], 1 + d[j], 1 + d[i-1, j], \text{diff}(x, y) + d[i-1, j-1]) \]
10 return \( d[m, n] \)
3: analysis

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

Edit distance variants

\[
\text{Edit}(X,Y) = \min\left\{ \begin{array}{l}
1 + \text{Edit}(X_{-1}, Y_{-1}) \quad \text{insertion} \\
1 + \text{Edit}(X_{-1}, Y_{-1}) \quad \text{deletion} \\
\text{Diff}(x_{i}, y_{j}) + \text{Edit}(X_{-1}, Y_{-1}) \quad \text{equal/substitution}
\end{array} \right.
\]

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

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

```python
def Edit(X, Y):
    m = len(X)
    n = len(Y)
    for i in range(m):
        for j in range(n):
            d[i][j] = \text{\text{distance}}[x_{i}, y_{j}]
            for k in range(1, \text{\text{minimum}}(i, j)):
                d[i][j] = \text{\text{min}}(d[i][j], d[i-k][j-k] + 1)
                d[i][j] = \text{\text{min}}(d[i][j], d[i-k][j] + 1)
                d[i][j] = \text{\text{min}}(d[i][j], d[i][j-k] + 1)
    return d[m][n]
```

Skiers and Skis

<table>
<thead>
<tr>
<th>Skis:</th>
<th>1 5 5 7 9 12 12 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skiers:</td>
<td>6 7 7 10 12</td>
</tr>
</tbody>
</table>

What is the optimal matching?