DYNAMIC PROGRAMMING: EVEN MORE FUN!

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

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
Is 5 part of the LIS?

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

include 5

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

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?

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

Can we rewrite LIS in terms of 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)?

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

Longest increasing sequence starting at $i$.

2: DP solution (bottom-up)

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

$LIS': 
\begin{align*}
5 & \quad 2 & \quad 8 & \quad 6 & \quad 3 & \quad 6 & \quad 9 & \quad 7 \\
\end{align*}$
2: DP solution (bottom-up)

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

\[ \text{LIS'}: \]
\[ 1 \]
\[ 5 \ 2 \ 8 \ 6 \ 3 \ 6 \ 9 \ 7 \]

20

2: DP solution (bottom-up)

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

\[ \text{LIS'}: \]
\[ 1 \ 1 \]
\[ 5 \ 2 \ 8 \ 6 \ 3 \ 6 \ 9 \ 7 \]

21

2: DP solution (bottom-up)

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

\[ \text{LIS'}: \]
\[ 1 \ 1 \]
\[ 5 \ 2 \ 8 \ 6 \ 3 \ 6 \ 9 \ 7 \]

22

2: DP solution (bottom-up)

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

\[ \text{LIS'}: \]
\[ 2 \ 1 \ 1 \]
\[ 5 \ 2 \ 8 \ 6 \ 3 \ 6 \ 9 \ 7 \]

23
2: DP solution (bottom-up)

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

LIS':

\[
\begin{array}{ccccccc}
3 & 2 & 1 & 1 \\
5 & 2 & 8 & 6 & 3 & 6 & 9 & 7
\end{array}
\]

24

2: DP solution (bottom-up)

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

LIS':

\[
\begin{array}{ccccccc}
3 & 2 & 1 & 1 \\
5 & 2 & 8 & 6 & 3 & 6 & 9 & 7
\end{array}
\]

25

2: DP solution (bottom-up)

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

LIS':

\[
\begin{array}{ccccccc}
2 & 2 & 3 & 2 & 1 & 1 \\
5 & 2 & 8 & 6 & 3 & 6 & 9 & 7
\end{array}
\]

26

2: DP solution (bottom-up)

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

LIS':

\[
\begin{array}{ccccccc}
4 & 2 & 2 & 3 & 2 & 1 & 1 \\
5 & 2 & 8 & 6 & 3 & 6 & 9 & 7
\end{array}
\]

27
2: DP solution (bottom-up)

<table>
<thead>
<tr>
<th>$LIS'(i)$</th>
<th>1 + $\max_{j&lt;i \text{ and } x_j &gt; x_i} LIS'(j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$LIS'$</td>
<td>3 4 2 2 3 2 1 1</td>
</tr>
<tr>
<td></td>
<td>5 2 8 6 3 6 9 7</td>
</tr>
</tbody>
</table>

What does the data structure for storing answers look like?

1-D array: only one thing changes for recursive calls
What are the “smallest” possible subproblems?

To calculate $\text{LIS}'(i)$, 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?

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

\[
\ell_i = \begin{cases} 
1 & \text{if } n = 1 \\
\max \left( \ell_i, \ell_j + 1 \right) & \text{otherwise}

2: DP solution (bottom-up)

LIS(X)
1 u ← LENGTH(X)
2 create array lis with n entries
3 for i ← 1 to n
4 max ← 1
5 for j ← i + 1 to n
6 if X[j] > X[i]
7 if 1 + lis[j] > max
8 max ← 1 + lis[j]
9 lis[i] ← max
10 max ← 0
11 for i ← 1 to n
12 if lis[i] > max
13 max ← lis[i]
14 return max

\[ UIS'(i) = 1 + \max_{j<i} \text{lis}(j) \]

LIS(X)
1 u ← LENGTH(X)
2 create array lis with n entries
3 for i ← 1 to n
4 max ← 1
5 for j ← i + 1 to n
6 if X[j] > X[i]
7 if 1 + lis[j] > max
8 max ← 1 + lis[j]
9 lis[i] ← max
10 max ← 0
11 for i ← 1 to n
12 if lis[i] > max
13 max ← lis[i]
14 return max

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

3: Analysis

Space requirements: \( \Theta(n) \)
Running time: \( \Theta(n^2) \)

LIS(X)
1 u ← LENGTH(X)
2 create array lis with n entries
3 for i ← 1 to n
4 max ← 1
5 for j ← i + 1 to n
6 if X[j] > X[i]
7 if 1 + lis[j] > max
8 max ← 1 + lis[j]
9 lis[i] ← max
10 max ← 0
11 for i ← 1 to n
12 if lis[i] > max
13 max ← lis[i]
14 return max

\[ LIS(X) = \max\{LIS'(i)\} \]
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 $\rightarrow$ ABACED $\rightarrow$ 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:**

\[ ABACED \rightarrow ABADE \rightarrow ABADES \]
Sub 'D' for 'C'  Sub 'S' for 'D'

Edit distance examples

\[ \text{Edit}(\text{Kitten, Mitten}) = 1 \]

Operations:

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

\[ \text{Edit}(\text{Happy, Hilly}) = 3 \]

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

Edit distance examples

\[ \text{Edit}(\text{Banana, Car}) = 5 \]

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

Edit distance examples

\[ \text{Edit}(\text{Simple, Apple}) = 3 \]

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

Why might this be useful?
Is edit distance symmetric?

that is, is \( \text{Edit}(s_1, s_2) = \text{Edit}(s_2, s_1) \)?

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

Why?
- sub ‘i’ for ‘j’ → sub ‘j’ for ‘i’
- delete ‘i’ → insert ‘i’
- insert ‘i’ → 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

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?

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

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

How can we use delete to transform X into Y?

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

Substitution

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

How can we use substitution to transform X into Y?

\[ Edit(X, Y) = 1 + Edit(X_{1 \ldots n-1}, Y_{1 \ldots m-1}) \]
What if the last characters are equal?

Equal

- $X = A B C B D A$?
- $Y = B D C A B$?

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

1b: recursive solution - combining results

- **Insert**: $Edit(X, Y) = 1 + Edit(X_{1...n}, Y_{1...m})$
- **Delete**: $Edit(X, Y) = 1 + Edit(X_{1...n-1}, Y_{1...m})$
- **Substitute**: $Edit(X, Y) = 1 + Edit(X_{1...n-1}, Y_{1...m-1})$
- **Equal**: $Edit(X, Y) = Edit(X_{1...n-1}, Y_{1...m-1})$

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

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

1: if they’re different
0: if they’re the same

2: DP solution (bottom-up)

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

What does the data structure for storing answers look like?

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?
### 2: DP solution (bottom-up)

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

What are the “smallest” possible subproblems?

- **Edit**($X, Y$) = len($X$) and **Edit**($Y, X$) = len($Y$)

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

$i < n$ and $j < m$

How should we fill in the table?

$i = 1 \rightarrow j = 1$

Where will the answer be?

$d_{nm}$

---

### 3: analysis

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

#### 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

---

https://leetcode.com/problems/house-robber/

198. House Robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

After an integer amount of time, an array representing the amount of money in each house, must the robber return the maximum amount of money you can rob without alerting the police.

Example 1:

```plaintext
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```plaintext
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.
```

---

https://leetcode.com/problems/interleaving-string/

76