Knapsack problems: Greedy or not?

0-1 Knapsack — A thief robbing a store finds $n$ items worth $v_1, v_2, \ldots, v_n$ dollars and weight $w_1, w_2, \ldots, w_n$ pounds, where $v_i$ and $w_i$ are integers. The thief can carry at most $W$ pounds in the knapsack. Which items should the thief take if they want to maximize value.

Fractional knapsack problem — Same as above, but the thief happens to be at the bulk section of the store and can carry fractional portions of the items. For example, the thief could take 20% of item $i$ for a weight of $0.2w_i$ and a value of $0.2v_i$.

Algorithmic "techniques"

Iterative/incremental: solve problem of size $n$ by first solving problem of size $n-1$.

Divide-and-conquer: divide problem into independent subproblems. Solve each subproblem independently. Combine solutions to subproblem to create solution to the original problem.

Greedy: make locally optimal choice and repeat on remaining subproblem.
Dynamic programming

Method for solving problems where optimal solutions can be defined in terms of optimal solutions to subproblems

AND

the subproblems are overlapping

Fibonacci: a first attempt

\[
\text{FIBONACCI}(n)
\]

1. If \( n = 1 \) or \( n = 2 \), return 1
2. Else, return \( \text{FIBONACCI}(n - 1) + \text{FIBONACCI}(n - 2) \)

Running time

Each call creates two recursive calls

Each call reduces the size of the problem by 1 or 2

Creates a full binary of depth \( n \)

\( O(2^n) \)

Can we do better?

\[
\text{Fib}(n)
\]

\[
\begin{align*}
\text{Fib}(n-1) & \quad \text{Fib}(n-2) \\
\text{Fib}(n-3) & \quad \text{Fib}(n-4) & \quad \text{Fib}(n-5) & \quad \text{Fib}(n-6)
\end{align*}
\]
A lot of repeated work!

Identifying a dynamic programming problem

The solution can be defined with respect to solutions to subproblems

The subproblems created are overlapping, that is we see the same subproblems repeated

Overlapping sub-problems

Dynamic programming: steps

1a) optimal substructure: optimal solutions to the problem incorporate optimal solutions to related subproblems
   - convince yourself that there is optimal substructure

1b) recursive definition: use this to recursively define the value of an optimal solution

2) DP solution: describe the dynamic programming table:
   - size, initial values, order in which it’s filled in, location of solution

3) Analysis: analyze space requirements, running time
1a: optimal substructure

optimal solutions to a problem incorporate optimal solutions to subproblems

```
FIBONACCI(n)
1 if n = 1 or n = 2
2 else return FIBONACCI(n - 1) + FIBONACCI(n - 2)
```

Sometimes the problem setup/structure meets the optimal substructure criteria by definition.

1b: recursive definition

Define a function and clearly define the inputs to the function

The function definition should be recursive with respect to multiple subproblems

pretend like you have a working function, but it only works on smaller problems

Key: subproblems will be overlapping, i.e., inputs to subproblems will not be disjoint

```
F(n) = ?
```
1b: recursive definition

Fibonacci:

\[ F(n) = F(n-1) + F(n-2) \]

2: DP solution

The recursive solution will generally be top-down, i.e., working from larger problems to smaller

DP solution:
- work bottom-up, from the smallest versions of the problem to the largest
- store the answers to subproblems in a table (often an array or matrix)
- to build bigger problems, lookup solutions in the table to subproblems

What are the smallest possible values (subproblems)?

F(1) = 1, F(2) = 1

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

How should we fill in the table?

How should we fill in the table? \( F(1) \rightarrow F(n) \)
2: DP solution

```python
FIBONACCI-DP(n)
1 fib[1] ← 1
2 fib[2] ← 1
3 for i ← 3 to n
   4 fib[i] ← fib[i-1] + fib[i-2]
5 return fib[n]
```

Store the intermediary values in an array \( \text{fib} \)

3: Analysis

Space requirements? 
Running time?

```
FIBONACCI-DP(n)
1 fib[1] ← 1
2 fib[2] ← 1
3 for i ← 3 to n
   4 fib[i] ← fib[i-1] + fib[i-2]
5 return fib[n]
```

Counting binary search trees

How many unique binary search trees can be created using the numbers 1 through \( n \)?

```
        4
       /\  /\  /
      2 5  3 6
     /  /  /  /
    1  3  6  4
```

Space requirements: \( \Theta(n) \)
Running time: \( \Theta(n) \)
Step 1: What is the subproblem?

Assume we have a working function (call it $T$) that can give us the answer to smaller subproblems.

How can we use the answer from this to answer our question?

How many options for the root are there?

1 2 3 $\ldots$ $n$

Subproblems

How many trees have $i$ as the root?

Subproblems

1, 2, ..., $i-1$ $i+1$, $i+2$, ..., $n$

$T(i-1)$

subproblem of size $i-1$
Subproblems

Subproblems

1, 2, ..., i-1

T(i-1)

subproblem of size i-1

1+1, i+2, ..., n

T(n-i)

subproblem of size n-i

Number of trees for i+1, i+2, ..., n is the same as the number of trees from 1, 2, ..., n+i

1a: optimal substructure

optimal solutions to a problem incorporate optimal solutions to related subproblems

T(i-1)  T(n-i)

By definition of binary trees: binary trees are recursive structures
Given solutions for \(T(i-1)\) and \(T(n-i)\) how many trees are there with \(i\) as the root?

Trees with \(i\) as root = \(T(i-1) \times T(n-i)\)

Trees with \(i\) as root = \(T(i-1) \times T(n-i)\)

Trees total:

\[T(n) = \sum_{i=1}^{n} T(i-1) \times T(n-i)\]
A recursive implementation

\[ T(n) = \sum_{i=1}^{n} T(i-1) \cdot T(n-i) \]

BST-Count(n):
1) if \( n = 0 \) return 1
2) else sum = 0
3) for \( i = 1 \) to \( n \)
4) \( \text{sum} \leftarrow \text{sum} + \text{BST-Count}(i-1) \times \text{BST-Count}(n-i) \)
5) return \( \text{sum} \)

Like with Fibonacci, we're repeating a lot of work

2: DP solution (from the bottom-up)

\[ T(n) = \sum_{i=1}^{n} T(i-1) \cdot T(n-i) \]

What are the smallest possible subproblems?

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

How should we fill in the table?

2: DP solution (from the bottom-up)

\[ T(n) = \sum_{i=1}^{n} T(i-1) \cdot T(n-i) \]

What are the smallest possible subproblems?

\( T(0) = 1, T(1) = 1 \)

Why do we need \( T(0) \) and why is it 1?

Need to think carefully about base cases/edge cases
What are the smallest possible subproblems? 
\( T(0) = 1, T(1) = 1 \)

To calculate \( T(n) \), what are all the subproblems we need to calculate? This is the “table”. \( T(0) \ldots T(n-1) \)

How should we fill in the table? \( T(0) \rightarrow T(n) \)

\[
T(n) = \sum_{i=1}^{n} T(i-1) \cdot T(n-i)
\]

---

**Fill in the first 4 values**

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>...</td>
</tr>
</tbody>
</table>
BST-COUNT-DP(n)
1 \text{ } c[0] = 1
2 \text{ } c[1] = 1
3 \text{ for } k \leftarrow 2 \text{ to } n
4 \text{ \quad } c[k] \leftarrow 0
5 \text{ \quad for } i \leftarrow 1 \text{ to } k
6 \text{ \quad \quad } c[k] \leftarrow c[k] + c[i - 1] \times c[k - i]
7 \text{ return } c[n]

\text{c[0]\times c[1] + c[1]\times c[0]}

\begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 & \quad 5 & \ldots & \quad n \\
1 & \quad 1 & \quad c[0] & \quad c[1] & \quad c[2] & \quad c[3] & \quad c[4] & \quad c[5] & \quad \ldots & \quad c[n]
\end{align*}

BST-COUNT-DP(n)
1 \text{ } c[0] = 1
2 \text{ } c[1] = 1
3 \text{ for } k \leftarrow 2 \text{ to } n
4 \text{ \quad } c[k] \leftarrow 0
5 \text{ \quad for } i \leftarrow 1 \text{ to } k
6 \text{ \quad \quad } c[k] \leftarrow c[k] + c[i - 1] \times c[k - i]
7 \text{ return } c[n]
3: Analysis

Space requirements?

Running time?

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

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

For a sequence $X = x_1, x_2, ..., x_n$, a subsequence is a subset of the sequence defined by a set of increasing indices $(i_1, i_2, ..., i_k)$ where $1 \leq i_1 < i_2 < ... < i_k \leq n$

$X = A B A C D A B A B$

ABA?

Subsequences

For a sequence $X = x_1, x_2, ..., x_n$, a subsequence is a subset of the sequence defined by a set of increasing indices $(i_1, i_2, ..., i_k)$ where $1 \leq i_1 < i_2 < ... < i_k \leq n$

$X = A B A C D A B A B$

ABA

Subsequences

For a sequence $X = x_1, x_2, ..., x_n$, a subsequence is a subset of the sequence defined by a set of increasing indices $(i_1, i_2, ..., i_k)$ where $1 \leq i_1 < i_2 < ... < i_k \leq n$

$X = A B A C D A B A B$

ACA?

Subsequences

For a sequence $X = x_1, x_2, ..., x_n$, a subsequence is a subset of the sequence defined by a set of increasing indices $(i_1, i_2, ..., i_k)$ where $1 \leq i_1 < i_2 < ... < i_k \leq n$

$X = A B A C D A B A B$

ACA
Subsequences

For a sequence $X = x_1, x_2, \ldots, x_n$, a subsequence is a subset of the sequence defined by a set of increasing indices $(i_1, i_2, \ldots, i_k)$ where $1 \leq i_1 < i_2 < \ldots < i_k \leq n$

$X = A \ B \ A \ C \ D \ A \ B \ A \ B$

DCA?

$X = A \ B \ A \ C \ D \ A \ B \ A \ B$

AADAA?

$X = A \ B \ A \ C \ D \ A \ B \ A \ B$

DCA

$X = A \ B \ A \ C \ D \ A \ B \ A \ B$

AADAA

$X = A \ B \ A \ C \ D \ A \ B \ A \ B$

AADAA

$X = A \ B \ A \ C \ D \ A \ B \ A \ B$

AADAA
Longest common subsequence (LCS)

Given two sequences X and Y, a common subsequence is a subsequence that occurs in both X and Y. Given two sequences X = x₁, x₂, …, xₙ and Y = y₁, y₂, …, yₙ, what is the longest common subsequence?

LCS problem

Given two sequences X and Y, a common subsequence is a subsequence that occurs in both X and Y. Given two sequences X = x₁, x₂, …, xₙ and Y = y₁, y₂, …, yₙ, what is the longest common subsequence?

X = A B C B D A B
Y = B D C A B A

1a: optimal substructure

optimal solutions to a problem incorporate optimal solutions to subproblems

Often a proof by contradiction:

Show: optimal solutions incorporate optimal solutions to subproblems

Assume the optimal solution does not contain optimal solutions to subproblems

Show this leads to a contradiction (often that we could create a better solution using the solution to the subproblem)
1a: optimal substructure

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

Proof by contradiction:
Assume: \( s_1, s_2, \ldots, s_m \) is the LCS\((X,Y)\), but \( s_2, \ldots, s_m \) is not the optimal solution to LCS\(\text{substring_after}(s_1, X), \text{substring_after}(s_1, Y)\).

If that were the case, then we could make a longer subsequence by:
\( s_1 \text{ LCS}\(\text{substring_after}(s_1, X), \text{substring_after}(s_1, Y)\)\)

contradiction

1b: recursive solution

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

Assume you have a solver for smaller problems

1b: recursive solution

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

Is the last character part of the LCS?

1b: recursive solution

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

Two cases: either the characters are the same or they're different
1b: recursive solution

\[ X = \text{A B C B D A A} \]
\[ Y = \text{B D C A B A} \]

\text{LCS}

If they're the same

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

If they're different

\[ LCS(X, Y) = LCS(X_{1 \ldots n-1}, Y) \]

1b: recursive solution

\[ X = \text{A B C B D A B} \]
\[ Y = \text{B D C A B A} \]

If they're different

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

If they're different

\[ LCS(X, Y) = LCS(X_{1 \ldots n-1}, Y) \]
### 1b: Recursive solution

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

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

\[
LCS(X, Y) = \begin{cases} 
1 + LCS(X_{i+1}, Y_{j+1}) & \text{if } x_i = y_j \\
\max(LCS(X_{i+1}, Y), LCS(X, Y_{j+1})) & \text{otherwise}
\end{cases}
\]

(for now, let’s just worry about counting the length of the LCS)

---

### 2: DP solution

\[
LCS(X, Y) = \begin{cases} 
1 + LCS(X_{i+1}, Y_{j+1}) & \text{if } x_i = y_j \\
\max(LCS(X_{i+1}, Y), LCS(X, Y_{j+1})) & \text{otherwise}
\end{cases}
\]

What types of subproblem solutions do we need to store?

\[ LCS(X_{1...j}, Y_{1...k}) \]

---

### 2: DP solution

\[
LCS[i, j] = \begin{cases} 
1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\
\max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise}
\end{cases}
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j )</td>
<td>( x )</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

For Fibonacci and tree counting, we had to initialize some entries in the array. Any here?
\[ LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases} \]

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

To fill in an entry, we may need to look:
- up one
- left one
- diagonal up and left

Just need to make sure these exist.
\[
LCS[i, j] = \begin{cases} 
1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\
\max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise}
\end{cases}
\]

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>\text{y}_i</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
LCS[i, j] = \begin{cases} 
1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\
\max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise}
\end{cases}
\]

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>\text{y}_i</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
LCS[i, j] = \begin{cases} 
1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\
\max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise}
\end{cases}
\]

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>\text{y}_i</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
LCS[i, j] = \begin{cases} 
1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\
\max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise}
\end{cases}
\]

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>\text{y}_i</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[
LCS[i, j] = \begin{cases} 
1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\
\max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise}
\end{cases}
\]

### Table 1

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LCS(ABC, BDC)**

### Table 2

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

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

**Running time:** \( \Theta(nm) \)
The algorithm

LCS-LENGTH(X, Y)
1 m ← length[X]
2 n ← length[Y]
3 c[0, 0] ← 0
4 for i ← 1 to m
5    c[i, 0] ← 0
6 for j ← 1 to n
7    c[0, j] ← 0
8 for i ← 1 to m
9    for j ← 1 to n
10       if x(i) = y(j)
11          c[i, j] ← 1 + c[i−1, j−1]
12 else if c[i−1, j] > c[i, j−1]
13          c[i, j] ← c[i−1, j]
14 else
15          c[i, j] ← c[i, j−1]
16 return c[m, n]
The algorithm

LCS-LENGTH(X, Y)
1 m ← length[X]
2 n ← length[Y]
3 c(0, 0) ← 0
4 for i ← 1 to m
5 c(i, 0) ← 0
6 for j ← 1 to n
7 c(0, j) ← 0
8 for i ← 1 to m
9 for j ← 1 to n
10 if xi = yj
11 c(i, j) ← 1 + c(i-1, j-1)
12 else if (i-1, j) > (i, j-1)
13 c(i, j) ← c(i-1, j)
14 else
15 c(i, j) ← c(i, j-1)
16 return c(m, n)

Keeping track of the solution

Our LCS algorithm only calculated the length of the LCS between X and Y. What if we wanted to know the actual sequence?

LCS[i, j] = \[
\begin{cases}
1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\
\max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise}
\end{cases}
\]

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>

| i  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|-----------------------------|
| x_i | B | D | C | A | B | C | A |
| y_j | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 A | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 B | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| 3 C | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| 4 B | 0 | 1 | 1 | 2 | 2 | ? | |
| 5 D | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 A | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 B | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

LCS(ABCB, BDCAB)
\[ LCS[i, j] = \begin{cases} 1 + LCS[i - 1, j - 1] & \text{if } x_i = y_j \\ \max(LCS[i - 1, j], LCS[i, j - 1]) & \text{otherwise} \end{cases} \]

How do we generate the solution from this?
\[ \text{LCS}[i, j] = \begin{cases} 
1 + \text{LCS}[i-1, j-1] & \text{if } x_i = y_j \\
\max(\text{LCS}[i-1, j], \text{LCS}[i, j-1]) & \text{otherwise}
\end{cases} \]

<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>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td>0</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td><strong>j</strong></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>y_j</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

We can follow the arrows to generate the solution BCBA.