Longest common subsequence (LCS):

Input: strings \(X = x_1, \ldots, x_n\) & \(Y = y_1, \ldots, y_m\)

Goal: find length of longest string which is a subsequence in both \(X\) & \(Y\).

Example: \(X = \overline{BCDBCDA}\)
\(Y = \overline{ABECBA}\)

answer = 4 for BCBA

Application: used in Unix diff for comparing files

First step, define subproblem.

Look at prefixes.
Natural to have a 2-dimensional table since 2 input strings

For \(i\) & \(j\) where \(0 \leq i \leq n\) & \(0 \leq j \leq m\) let:

\[ L(i,j) = \text{length of LCS of } x_1, \ldots, x_i \text{ with } y_1, \ldots, y_j \]
Base cases: 
\[ L(i, 0) = 0 \]
\[ L(0, j) = 0 \]

For recurrence, two cases: \( X_i = Y_j \) & \( X_i \neq Y_j \)

Suppose \( X_i \neq Y_j \): 
\[ X = \square \rightarrow x_i \]
\[ Y = \square \rightarrow y_j \]
Can't match \( X_i \) with \( Y_j \) so one or both is not in the solution, so can drop \( X_i \) or \( Y_j \).
Try both & take best.
If \( X_i \) is dropped then \( L(i, j) = L(i-1, j) \)
If \( Y_j \) is dropped then \( L(i, j) = L(i, j-1) \)
Therefore if \( X_i \neq Y_j \) then:
\[ L(i, j) = \max\{L(i-1, j), L(i, j-1)\} \]

Suppose \( X_i = Y_j \):
Either the LCS ends with \( X_i = Y_j \) then 
\[ L(i, j) = 1 + L(i-1, j-1) \]

or it doesn't use \( X_i = Y_j \) then \( X_i \) matched with some other character or \( Y_j \) matched elsewhere, so 
\[ L(i, j) = L(i, j-1) \text{ or } L(i-1, j) \]

Therefore if \( X_i = Y_j \) then
\[ L(i, j) = \max\{1 + L(i-1, j-1), L(i, j-1), L(i-1, j)\} \]

[In fact can argue that: \( L(i, j) = 1 + L(i-1, j-1) \) since we might as well match \( X_i = Y_j \).]
L = i

fill the table row by row (or column by column)

to get $L(i, j)$ look at $\leq 3$ previous neighbors:
$L(i-1, j), L(i, j-1), L(i-1, j-1)$

LCS$(X, Y)$:

for $i = 0 \rightarrow n$, $L(i, 0) = 0$
for $j = 0 \rightarrow m$, $L(0, j) = 0$

for $i = 1 \rightarrow n$,

for $j = 1 \rightarrow m$

if $X_i = Y_j$

then $L(i, j) = max\{1 + L(i-1, j-1), L(i, j-1), L(i-1, j)\}$

else $L(i, j) = max\{L(i, j-1), L(i-1, j)\}$

Return $(L(n, m))$

Running time: $O(nm)$
Earlier example:

\[ X = \text{BCDBCDAB} \]
\[ Y = \text{ABECBA} \]

\[ L = \begin{array}{cccccc}
\emptyset & A & B & E & C & B & A \\
\text{B} & 0 & 0 & 1 & 1 & 1 & 1 \\
\text{C} & 0 & 0 & 1 & 1 & 2 & 2 \\
\text{D} & 0 & 0 & 0 & 0 & 2 & 2 \\
\text{E} & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{A} & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]
Closely related:

**Edit distance:**

**Input:** strings $X = x_1, \ldots, x_n$ & $Y = y_1, \ldots, y_m$

**Goal:** Min # of edits (Insertion, Deletion, Substitution) to go between $X$ & $Y$

**Example:**

$X = \text{AAGCTGCCTAA}$
$Y = \text{AACCGCAATA}$

$\text{AA(} G \text{C(T)} \text{GC(C-TA(A))}$
$\text{AAC(} C \text{G(CA)} \text{ATA(A))}$

**Edit distance = 5**

Variant used in BLAST—widely used in Biology to align DNA sequences.

In Biology we also have a $5 \times 5$ scoring matrix

$$S = \begin{bmatrix}
- & A & G & C & T \\
A & S & S & S & S \\
G & S & S & S & S \\
C & S & S & S & S \\
T & S & S & S & S \\
\end{bmatrix}$$

when $X_i \neq Y_j$: the cost is $S(X_i, Y_j)$
Knapsack:

Total capacity $B$

and $n$ objects with:

integer weights $w_1, \ldots, w_n$

& integer values $v_1, \ldots, v_n$

Goal: find subset $S$ of objects that

(a) fits in the backpack

(so total weight is $\leq B$)

& (b) maximizes the total value.

In other words, find subset $S$ of objects where

(a) $\sum_{i \in S} w_i \leq B$

& (b) maximizes $\sum_{i \in S} v_i$

Application: scheduling jobs with limited resources

Two versions:

1) without repetition: one copy of each object

2) with repetition: unlimited supply of each object.
Today: without repetition - one copy of each.

What about a greedy approach?

Example:

Object: 1 2 3 4

Values: 15 10 8 1

Weights: 15 12 10 5

B = 22

Sort objects by $r_i = \frac{v_i}{w_i} = \text{value per unit of weight}$

$r_1 > r_2 > r_3 > r_4$

Greedy: add object 1 then add object 4

Objects 1, 4

Total value = 16

Optimal: 2 & 3

Total value = 18
Dynamic Programming approach:

First, define the subproblem

Initial try:

\[ K(j) = \text{max value achievable using a subset of objects } 1, \ldots, j \]

Try to write a recurrence for \( K(j) \) in terms of \( K(1), \ldots, K(j-1) \)

Take \( K(j-1) \), how do we know if we can add object \( j \) to it?

Need to know how much weight is available.

For each weight \( b \) where \( 0 \leq b \leq B \) want to know the max value achievable.
For \( b \leq j \) where \( 0 \leq b \leq B \) and \( 0 \leq j \leq n \)

let \( K(b, j) = \text{max value achievable using a subset of objects } 1 \ldots j \) & total capacity \( b \)

Goal: compute \( K(B, n) \).

For recurrence for \( K(b, j) \):

either:
- use object \( j \):
  - then gain value \( v_j \) & available capacity is \( b - w_j \) so optimal is \( v_j + K(b - w_j, j - 1) \)

- don't use object \( j \):
  - then it's same as optimal for objects \( 1 \ldots j - 1 \) with capacity \( b \). So it's
  \[ K(b, j - 1) \]

Therefore:
- if \( w_j \leq b \) then \( K(b, j) = \max \{ v_j + K(b - w_j, j - 1) \} \)
- else \( K(b, j) = K(b, j - 1) \)

Base cases:
- \( K(b, 0) = 0 \)
- \( K(0, j) = 0 \)
Recurrence for $k(i,j)$ uses $k(i,j-1)$

So fill $k$ column by column.

\[
\begin{array}{c}
k = b \\
\vdots \\
\end{array}
\]

Knapsack No Repeat $(B, w_1, \ldots, w_n, v_1, \ldots, v_n)$

for $j = 0 \rightarrow n$, $k(0,j) = 0$

for $b = 0 \rightarrow B$, $k(b,0) = 0$

for $j = 1 \rightarrow n$

for $b = 1 \rightarrow B$

\[
\begin{cases}
\text{if } w_j > b \\
& \text{then } k(b,j) = k(b,j-1) \\
& \text{else } k(b,j) = \max \{ k(b_j-1), v_j + k(b-w_j, j-1) \}
\end{cases}
\]

Return $(k(B,n))$

Running time: $O(nB)$
Is the running time polynomial in the input size?

No, should be Poly(n, log B) which is unlikely since Knapsack is NP-complete.