CompSci 260P
Spring 2023 Lecture 2:
Dynamic Programming 1:
LCS and Subset Sum

www.lcs.uci.edu/~mikes/compsci260p
Longest Common Subsequence

Input: Two sequences (strings etc)

Output: Longest common subsequence.

Examples of common subsequences:

exercise
determine

morning
triangle

toward

thousand
LCS: Recursive Solution

Let $LCS(n,m)$ be the length of the longest common subsequence of $X[1...n]$ and $Y[1...m]$.

```cpp
// base case:
if 0 == n or 0 == m:
    return 0

// general. Tautology
if X[n] == Y[m] return 1 + LCS(n-1, m-1)
else return std::max(LCS(n-1, m), LCS(n, m-1))
```
LCS(n,m): // recursive for reference

if 0 == n or 0 == m then
    return 0
else if X[n] = Y[m] then
    return 1 + LCS(n - 1, m - 1)
else
    return max( LCS(n - 1, m), LCS(n, m - 1) )

declare LCS[][] of (n+1) x (m+1)
for j := 1...m  LCS[0][j] = 0 \(O(m)\)
for i := 1...n  LCS[i][0] = 0 \(O(n)\)
for i := 1...n
    for j := 1...m
        LCS[i][j] := LCS[i][j-1], LCS[i-1][j], LCS[i-1][j-1] \(O(1)\) per case
The Subset Sum Problem

**Problem Statement**: Given a set $S$ of $n$ positive integers, as well as a positive integer $T$, determine if there is a subset of $S$ that sums to exactly $T$.

**Example 1**: $S = \{2, 3, 4\}$, $T = 6$, answer is “yes”

**Example 2**: $S = \{2, 3, 5\}$, $T = 6$, answer is “no”
Subset Sum: recursive solution

As with any dynamic programming problem

- Try a recursive approach first
- Find a tautology, then list decisions

\[ \text{Sub}(n, T) : \text{does a subset of } S[1\ldots n] \text{ add to } T? \]

If \( S[n] > T \), // use \( S[n] \) in \( \text{Sum?} \) No or Yes

ifno = Sub(n-1, T)

ifyes = Sub(n-1, T - S[n])

return ifno or ifyes
Subset Sum: iterative solution

SubsetSum\(i, j\) // recursive for reference

if \(0 = j\) then
    return true
else if \(0 = i\) then
    return false
else
    return SubsetSum\((i - 1, j)\) OR
    \((j - S[i] \geq 0 \text{ and } \text{SubsetSum}(i - 1, j - S[i]))\)

Sub\([0...n, 0...T]\)

\[
\text{for } i = 1 \ldots n \\
\text{for } j = 1 \ldots T
\]

\[
\text{Sub}[i, j] = \begin{cases} 
0(nT) & \\
\text{Sub}[i-1, j] \text{ or } (j - S[i] \geq 0 \text{ and } \text{Sub}[i-1, j - S[i]])
\end{cases}
\]
Subset Sum: Running Time

\[
\text{SubsetSum}(S[1 \ldots n], T) // \text{iterative}
\]

\[
\text{for } i = 0 \ldots n \text{ do}
\]

\[
\text{SUB}[i, 0] = \text{true}
\]

\[
\text{for } j = 1 \ldots T \text{ do}
\]

\[
\text{SUB}[0, j] = \text{false}
\]

\[
\text{for } i = 1 \ldots n \text{ do}
\]

\[
\text{for } j = 1 \ldots T \text{ do}
\]

\[
\text{Fill in SUB}[i, j] \text{ in } O(1)
\]

\[
\text{return } \text{SUB}[n, T]
\]

▶ Suppose we double the size of \( S \), but leave \( T \) alone. Will your algorithm scale well?

▶ Suppose we double the size of \( T \), but leave \( S \) alone. Will your algorithm scale well?
Solving with Dynamic Programming

- Describe *in English* the function
  - Not *how* it works (yet)
  - Yes *what it solves.*
  - Skipping this step = 0 on problem

- Give that function a meaningful variable name.
  - Not “OPT” or “DP” or “table”
  - Not a single letter either.

- Give recursive formulation.

- Describe the iterative running time.