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

www.ics.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 $X[1 \ldots n]$ and $Y[1 \ldots m]$.

```
//base cases:
  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: Iterative Solution : "Memoize"

```
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) × (m+1)

for j = 1...m LCS[0][j] = 0 } O(m) LCS[0][0] = 0

for i = 1...n LCS[i][0] = 0 } O(n)

$O(nm)$

for i = 1..n

  for j = 1...m

    // LCS[i][j] as per above } 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

Sub(n,T): does a subset of S[1...n] add to T?

if S[n]>T, must do "ifno"

// use S[n] in 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] ≥ 0 and SubsetSum(i − 1, j − S[i]))
```

for $i = 0 \ldots n$
  $Sub[i,0] = True$

for $j = 1 \ldots T$
  $Sub[0, j] = false$

$Sub[0 \ldots n, 0 \ldots T]$

for $i = 1 \ldots n$
  for $j = 1 \ldots T$
    $Sub[i,j] = Sub[i-1,j]$ or $(j - S[i] \geq 0$ and $Sub[i-1, j-S[i]])$

$O(nT)$

Subset Sum: Running Time

```
SubsetSum(S[1...n], T) // iterative
  for i = 0...n do
     SUB[i, 0] = true
  for j = 1...T do
     SUB[0, j] = false
  for i = 1...n do
     for j = 1...T do
        Fill in SUB[i, j] in O(1)
  return SUB[n, T]
```

$O(n \quad T)$

elts in S

target #

- 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.