CompSci 161
Winter 2023 Lecture 12:
Dynamic Programming III:
Subset Sum

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) : "is there a subset of S[1...n] that adds to T?

Use S[n] or don't

↳ Sub(n-1)

Recursive solution, attempt two

`Sub(n,T)` : "does a subset of $S[1\ldots n]$ add to T?
// Tautology: if yes, $S[n]$ is used or it is not
// if_no = Sub$(n-1, T)$
// if_yes = Sub(n-1, T- S[n]) // unless S[n] > T

base cases? T=0 → yes.
T>0 and n=0 : no.

// Now the code:

if T=0 return true
elif n=0 return false

else return Sub(n-1,T) or (T ≥ S[n] and Sub(n-1, T-S[n]))

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
    ($j - S[i] \geq 0$ and SubsetSum($i - 1, j - S[i]$))

---

for $i = 0 \dots n$ Sub[i, 0] = true
for $j = 1 \dots T$ Sub[0, j] = false

for $i = 1 \dots n$
  for $j = 1 \dots T$
    fill in Sub[i,j] as per above $\} O(1)$

$\} O(nT)$

Subset Sum: Visualization

Example 1: $S = \{2, 3, 4\}$, $T = 6$.

		0	1	2	3	4	5	6
$j=0$	$\{\}$	T	F	F	F	F	F	F
$j=1$	$\{2\}$	T	F	T	F	F	F	F
	$\{2,3\}$	T						
	$\{2,3,4\}$	T						

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]
```

- What is the running time of Subset Sum?

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]
```

- 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?

Subset Sum: Find the Subset

SubsetSum(S[$1 \ldots n$], T) // iterative

for $i = 0 \ldots n$ **do**
  SUB[$i, 0$] $=$ true
for $j = 1 \ldots T$ **do**
  SUB[$0, j$] $=$ false
for $i = 1 \ldots n$ **do**
  for $j = 1 \ldots T$ **do**
    Fill in SUB[i, j] in $O(1)$
if SUB[n, T] is true **then**

$i \leftarrow n$, $j \leftarrow T$ // Sub[i,j] true

while $i > 0$:
  if ! Sub[i-1, j]
    Output S[i]
    j = j - S[i]
  i--