KNAPSACK

Learning Goals

- Write correct pseudocode for a dynamic prog. alg [DP2]

:

Announcements

- CS Seminar: '26-'27 Course Preview!
  Friday 4/3 12:20-1:20, 75 Shannon 102

- Ethics assignment questions

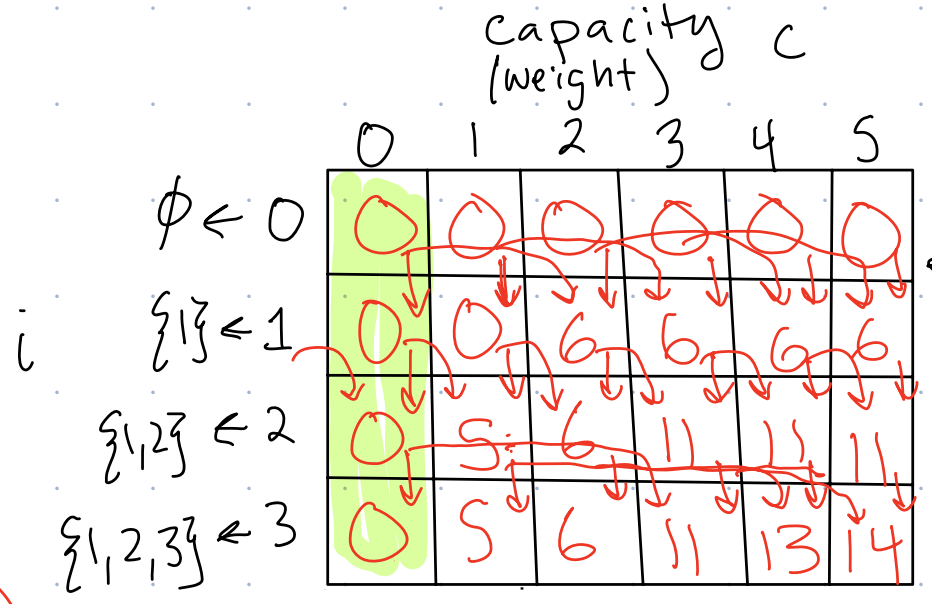
- DP2 on exam

Exit Ticket Warm Ups

Input:

	1	2	3
value	6	5	8
weight	2	1	3

$W = 5$



$$A[n, W] = \begin{cases} \max\{A[n-1, W-w_n] + v_n,\ A[n-1, W]\} \\ 0 \text{ if } n = 0 \\ 0 \text{ if } W = 0 \end{cases}$$

← OK to add to base case?

Knapsack Problem

Input: n items
- $v_i \in \mathbb{N}$ is value of i^{th} item
- $w_i \in \mathbb{N}$ is weight of i^{th} item

Capacity W (max weight) $\in \mathbb{N}$

Applications
- Shipping
- Exams (?)

Output: $S \subseteq [n]$ $\quad [n] = \{1, 2, 3, \dots, n\}$

such that
- $V(S) = \sum_{i \in S} v_i$ is maximized
- $W(S) = \sum_{i \in S} w_i \leq W$

Brute Force
MWIS

$O(n \cdot 2^n)$

ex: $W = 5$

	1	2	3	4
value	8	5	6	7
weight	2	1	3	4

What is optimal set?

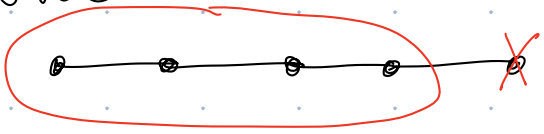
A) $\{1, 3\}$ B) ~~$\{1, 1\}$~~ C) $\{1, 2, 3\}$ D $\{2, 3\}$

Dynamic Programming Design Strategy

1. Think about solution as sequence of choices. Final choice?

   MWIS: final vertex in set or not ↙ S_i

2. (a) For each option, think about relevant subproblem + (b) create recurrence



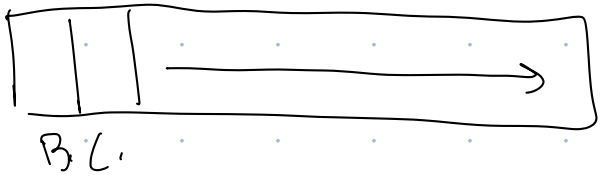
$$S_n = \begin{cases} S_{n-1} & \text{if } v_n \notin S_n \\ S_{n-2} \cup \{v_n\} & \text{if } v_n \in S_n \end{cases}$$

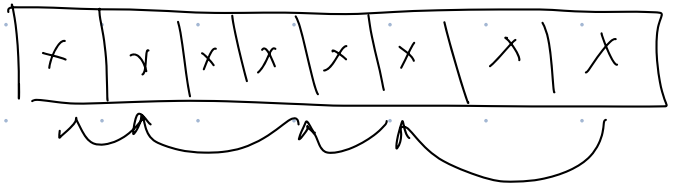
3. Convert into recurrence for objective function

$$A(n) = \max\{A(n-1),\ A(n-2) + w(v_n)\}, \text{ B.C.}$$

4. Pseudocode:
   - create A array with FOR loop from base case
   - work backwards through A to get solution

A





Knapsack Problem

Input: n items
- v_i is value of i^{th} item
- w_i is weight of i^{th} item

Capacity W (max weight)

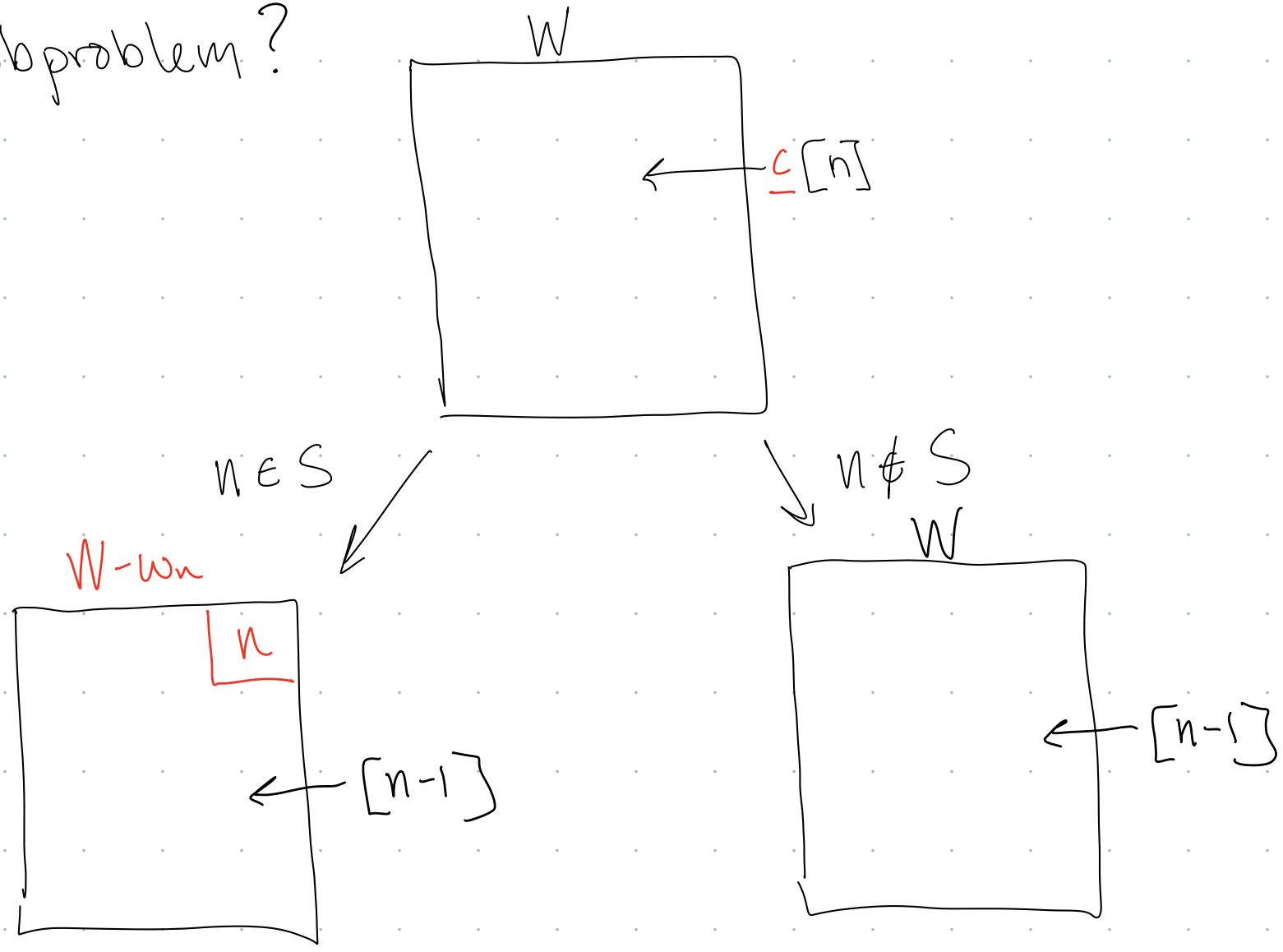
Output: $S \subseteq [n]$ ← notation $\{1, 2, 3, \ldots, n\}$

s.t. • $V(S) = \sum_{i \in S} v_i$ is maximized

• $W(S) = \sum_{i \in S} w_i \leq W$

1. Final option?
   - Include $n \in S$ or $n \notin S$

2a Subproblem?



In subproblems, Weight changes, items available change.
Need subproblems that depend on both

Define:

$[i] = \{1, 2, 3, \ldots, i\}$

$$S_{i,w} = \begin{cases} \text{a subset } S \subseteq [i] \text{ such that} \\ \quad \cdot\ W(S) \leq w \\ \quad \cdot\ V(S) \text{ is maximized} \end{cases}$$

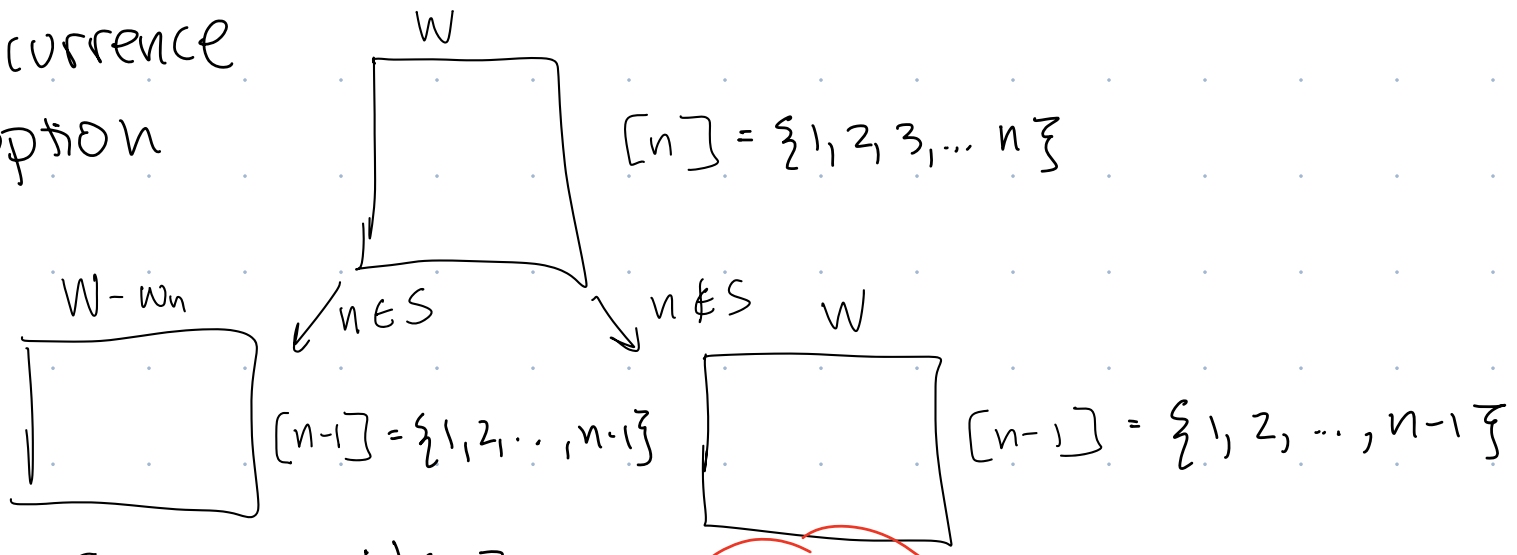
items	1	2	3	4
value	6	5	8	7
weight	2	1	3	4

What is $S_{2,4}$?

$S_{2,2} = \{1\}$

A) $\{1, 1\}$ B) $\{1, 2\}$ C) $\{2, 3\}$ D) $\{4\}$

2b Create recurrence for each option (DP1)



$$S_{n,W} = \begin{cases} S_{n-1, W-w_n} \cup \{n\} & \text{if } n \in S_{n,W} \\ S_{n-1,W} & \text{if } n \notin S_{n,W} \end{cases}$$

Base case later

3. Create Recurrence for objective function

$$A(n,W) = \begin{cases} \max\{A(n-1, W-w_n) + v_n, A(n-1,W)\} \\ \text{Base cases} \end{cases}$$

value of $S_{n,W}$

4. Fill out this array using recurrence for these items:

Input:

	1	2	3
value	6	5	8
weight	2	1	3

$W = 5$

capacity c (weight)

i	0	1	2	3	4	5
$\phi \leftarrow 0$	0	0	0	0	0	0
$\{1\} \leftarrow 1$	0	0	6	6	6	6
$\{1,2\} \leftarrow 2$	0	5	6	11	11	11
$\{1,2,3\} \leftarrow 3$	0	5	6	11	13	14

this option only if $W - w_n \geq 0$

$$A[n, W] = \begin{cases} \max\{A[n-1, W-w_n] + v_n,\ A[n-1, W]\} \\ 0 \quad \text{if} \quad n = 0 \end{cases}$$

0) Names, Spring break

a) Determine Base Case(s)

b) Something else needed to properly use recurrence...

2. Write pseudocode to create A, working backwards

Input: length-n arrays v and w, integer W

Output: Set $S \subseteq [n]$

Initialize an array A of size $(n+1) \times (W+1)$, ~~fill with 0~~

For $c \leftarrow 0$ to W:

| $A[0, c] \leftarrow 0$ // Base case

For $i \leftarrow 1$ to n:

| For $c \leftarrow 0$ to W:

| | $A[i, c] \leftarrow A[i-1, c]$

| | If $c \geq w[i]$ and $A[i-1, c-w[i]] + v[i] \geq A[i-1, c]$:

| | | $A[i, c] \leftarrow A[i-1, c-w[i]] + v[i]$

$S \leftarrow \emptyset$

$i \leftarrow n$

$c \leftarrow W$

While $i \geq 1$

| If $A[i, c] = A[i-1, c]$ then $i--$

| Else:

| | $S \leftarrow S \cup \{i\}$

| | $c \leftarrow c - w[i]$

| | $i--$

// Working backwards thru 0 base cases (not needed)

Tips for working backwards through A:

- Use if-else sequences, not if-if
  - → In edge case where two terms are equal:

$$\max\{A[n-1, W-w_n] + v_n,\ A[n-1, W]\}$$

your code can fail :

- Pick which option to check first wisely to avoid index-out-of-bounds errors
  - → If check if $A[i, c] = A[i-1, c]$ first, no I-O-O-B error.
  - → If check if $A[i, c] = A[i-1, c-w[n]] + v[n]$ first can get I-O-O-B error if $c-w[n]$ goes out-of-bounds. Would need extra if statement to avoid.

Runtime

```
For c ← 0 to W:
    A[0, c] = 0 // Base case
For i ← 1 to n:
    For c ← 0 to W:
        A[i, c] ← A[i-1, c]
        If c ≥ w[i] and A[i-1, c - w[i]] + v[i] > A[i-1, c]:
            A[i, c] ← A[i-1, c - w[i]] + v[i]
S ← ∅
i ← n
c ← W
While (i > 0 and c > 0):
    if A[i, c] = A[i-1, c]: i--
    else:
        S ← S ∪ {i}
        c ← c - w[i]
        i--
```

$O(nW)$

$O(n)$

Input: n items
- $v_i \in \mathbb{N}$ is value of i^{th} item
- $w_i \in \mathbb{N}$ is weight of i^{th} item

Capacity W (max weight) $\in \mathbb{N}$

Runtime

$\Theta(nW)$

$O(n^2)$ $\quad$ $O(W \log W)$

Input size (# of bits) in terms of n, W, V (max item value)

W' (max item weight)

A: $O(\log n + \log W + \log V)$

B $O(n \log V + \log W)$

C $O(n \log V + W)$

D $O(nV + W)$

$O(n \log V + n \log W' + \log W)$

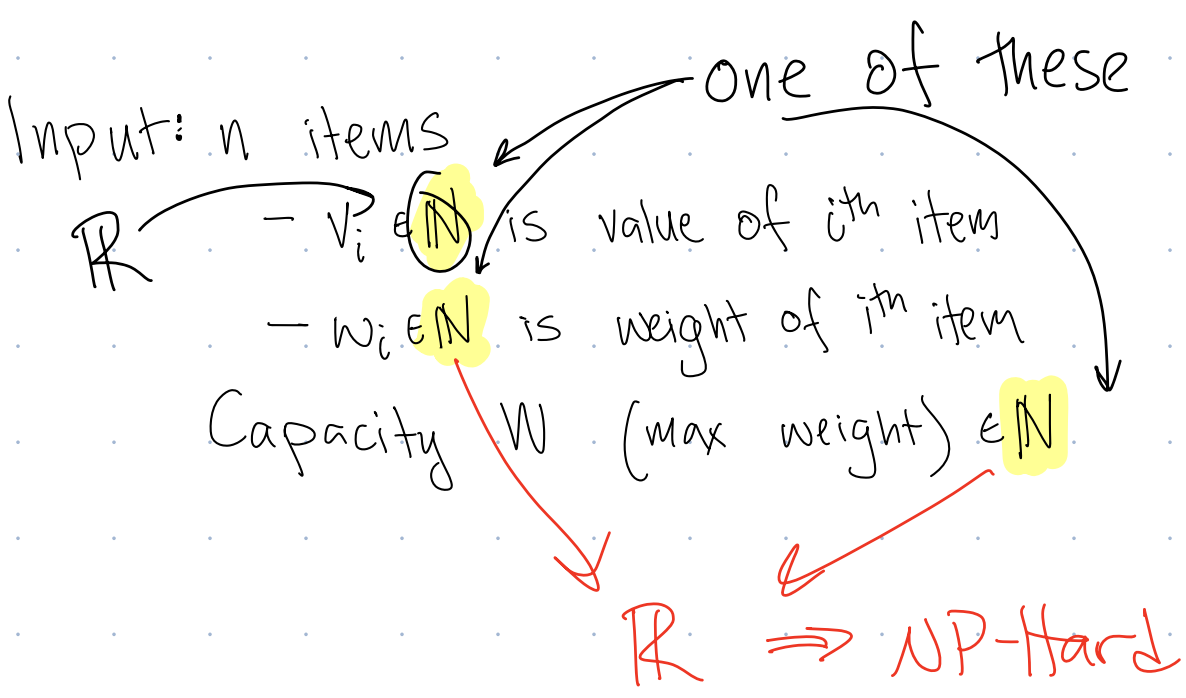
- $n \log V$: item value
- $n \log W'$: item weight
- $\log W$: total capacity

$n \approx W \rightarrow O(n) \rightarrow P$

$2^n \approx W$

$n \approx \log W \rightarrow O(\log W)$

$\rightarrow$ NP-hard



$$A[n\ W] = \begin{cases} \max\{A[n-1, W-w_n] + v_n,\ A[n-1, W]\} \\ \end{cases}$$