Dynamic Programming
0/1 Knapsack

(Class 22)

0/1 Knapsack Problem: Dynamic Programming Approach

- For each $i \leq n$ and each $w \leq W$, solve the knapsack problem for the first i objects when the capacity is w.
- Why will this work?
- Because solutions to larger subproblems can be built up easily from solutions to smaller ones.

- We construct a matrix $V[0 \dots n, 0 \dots W]$.
- For $1 \leq i \leq n$, and $0 \leq j \leq W$, $V[i, j]$ will store the maximum value of any set of objects $\{1, 2, \dots, i\}$ that can fit into a knapsack of weight j.
- $V[n, W]$ will contain the maximum value of all n objects that can fit into the entire knapsack of weight W.

- To compute entries of V we will imply an inductive approach.
- As a basis, $V[0, j] = 0$ for $0 \leq j \leq W$ since if we have no items then we have no value.
- We consider two cases:
  - **Leave object $\boldsymbol{i}$:** If we choose to not take object i, then the optimal value will come about by considering how to fill a knapsack of size j with the remaining objects $\{1, 2, \dots, i-1\}$. This is just $V[i-1, j]$.

  - **Take object $\boldsymbol{i}$:** If we take object i, then we gain a value of v_i. But we use up w_i of our capacity.
    With the remaining $j - w_i$ capacity in the knapsack, we can fill it in the best possible way with objects $\{1, 2, \dots, i-1\}$.
    This is $v_i + V[i-1, j-w_i]$. This is only possible if $w_i \leq j$.

- This leads to the following recursive formulation:

$$V[i,j] = -\infty \qquad \text{if } j < 0$$

$$V[0,j] = 0 \qquad \text{if } j \geq 0$$

$$V[i,j] = \begin{cases} V[i-1,j] & \text{if } w_i > j \\ \max\{V[i-1,j],\ \ v_i + V[i-1,j-w_i]\} & \text{if } w_i \leq j \end{cases}$$

- A naive evaluation of the running time of this recursive definition is exponential.
- So, as usual, we avoid re-computation by making a table.

- Example: The maximum weight the knapsack can hold is W is 11.
- There are five items to choose from.
- Their weights and values are presented in the following table:

Weight limit (j):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1$ $v_1 = 1$												
$w_2 = 2$ $v_2 = 6$												
$w_3 = 5$ $v_3 = 18$												
$w_4 = 6$ $v_4 = 22$												
$w_5 = 7$ $v_5 = 28$												

- The $[i, j]$ entry here will be $V[i, j]$, the best value obtainable using the first i rows of items if the maximum capacity were j.
- We begin by initializing the first row.

Weight limit:	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1$ $v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2$ $v_2 = 6$	0											
$w_3 = 5$ $v_3 = 18$	0											
$w_4 = 6$ $v_4 = 22$	0											
$w_5 = 7$ $v_5 = 28$	0											

- Recall that we take $V[i, j]$ to be 0 if either i or j is 0.
- We then proceed to fill in top-down, left-to-right always using:

$$V[\mathrm{i}, \mathrm{j}] = \max\{V[\mathrm{i}-1, \mathrm{j}], v_{\mathrm{i}} + V[\mathrm{i}-1, \mathrm{j}-w_{\mathrm{i}}]\}$$

Weight limit:	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1$ $v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2$ $v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5$ $v_3 = 18$	0											
$w_4 = 6$ $v_4 = 22$	0											
$w_5 = 7$ $v_5 = 28$	0											

Weight limit:	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1$ $v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2$ $v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5$ $v_3 = 18$	0	1	6	7	7	18	19	*24*	25	25	25	25
$w_4 = 6$ $v_4 = 22$	0											
$w_5 = 7$ $v_5 = 28$	0											

- As an illustration, the value of $V[3,7]$ was computed as follows:

$$
\begin{aligned}
V[3,7] &= \max\{V[3-1,7],\ \ v_3 + V[3-1, 7-w_3]\} \\
&= \max\{V[2,7],\ \ 18 + V[2, 7-5]\} \\
&= \max\{7,\ \ 18+6\} \\
&= 24
\end{aligned}
$$

Weight limit:	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1$ $v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2$ $v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5$ $v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6$ $v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7$ $v_5 = 28$	0											

- Finally, we have

Weight limit:	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1$ $v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2$ $v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5$ $v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6$ $v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7$ $v_5 = 28$	0	1	6	7	7	18	22	28	29	34	35	*40*

- The maximum value of items in the knapsack is 40, the bottom-right entry.

- The dynamic programming approach can now be coded as the following algorithm:

```
KNAPSACK(n,W)
for w ← 0 to W
   do V[0,w] ← 0
for i ← 0 to n
   do V[i, 0] ← 0
   for w ← 0 to W
      if (w_i < w & v_i + V[i − 1,w − w_i] > V[i − 1,w])
         then V[i,w] ← v_i + V[i − 1,w − w_i]
         else V[i,w] ← V[i − 1,w]
```

- The time complexity of this algorithm is clearly $O(n \cdot W)$.
- It must be cautioned that as n and W get large, both time and space complexity become significant.

Constructing the Optimal Solution

- The algorithm for computing $V[i, j]$ does not keep record of which subset of items gives the optimal solution.
- To compute the actual subset, we can add an auxiliary boolean matrix $keep[i, j]$ which is 1 if we decide to take the i^{th} item and 0 otherwise.

- We will use all the values $keep[i, j]$ to determine the optimal subset T of items to put in the knapsack as follows:
  - If $keep[n, W]$ is 1, then $n \in T$. We can now repeat this argument for $keep[n-1, W-w_n]$.
  - If $keep[n, W]$ is 0, the $n \notin T$ and we repeat the argument for $keep[n\ -\ 1, W]$.

- We will add this to the knapsack algorithm:

```
KNAPSACK(n,W)
for w ← 0 to W
    do V[0,w] ← 0
for i ← 0 to n
    do V[i, 0] ← 0
    for w ← 0 to W
        if (wi < w & vi + V[i − 1,w − wi] > V[i − 1,w])
            then V[i,w] ← vi + V[i − 1,w − wi]
            else V[i,w] ← V[i − 1,w]
  // output the selected items
k ← W
for i ← n downto 1
    if keep[i, k] = 1
        then output i
        k ← k − wi
```

- Here is the keep matrix for the example problem.

Weight limit:	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1$ $v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2$ $v_2 = 6$	0	0	1	1	1	1	1	1	1	1	1	1
$w_3 = 5$ $v_3 = 18$	0	0	0	0	0	$\boxed{1}$	1	1	1	1	1	1
$w_4 = 6$ $v_4 = 22$	0	0	0	0	0	0	1	0	1	1	1	$\boxed{1}$
$w_5 = 7$ $v_5 = 28$	0	0	0	0	0	0	0	1	1	1	1	0

- When the item selection algorithm is applied, the selected items are 4 and 3.
- This is indicated by the boxed entries in the table above.