Dynamic Programming

- Consider the instance: $i, W', (w_1, w_2, \cdots, w_i)$;
- $opt[i, W']$: the optimum value of the instance

$$opt[i, W'] = \begin{cases} 0 & i = 0 \\ opt[i-1, W'] & i > 0, w_i > W' \\ \max \left\{ \begin{array}{c} opt[i-1, W'] \\ opt[i-1, W' - w_i] + w_i \end{array} \right\} & i > 0, w_i \leq W' \end{cases}$$

Dynamic Programming

1: **for** $W' \leftarrow 0$ to W **do**
2: $\quad opt[0, W'] \leftarrow 0$
3: **for** $i \leftarrow 1$ to n **do**
4: $\quad$ **for** $W' \leftarrow 0$ to W **do**
5: $\quad\quad opt[i, W'] \leftarrow opt[i-1, W']$
6: $\quad\quad$ **if** $w_i \leq W'$ and $opt[i-1, W'-w_i] + w_i \geq opt[i, W']$ **then**
7: $\quad\quad\quad opt[i, W'] \leftarrow opt[i-1, W'-w_i] + w_i$
8: **return** $opt[n, W]$

Recover the Optimum Set

1: **for** $W' \leftarrow 0$ to W **do**
2: $\quad opt[0, W'] \leftarrow 0$
3: **for** $i \leftarrow 1$ to n **do**
4: $\quad$ **for** $W' \leftarrow 0$ to W **do**
5: $\quad\quad opt[i, W'] \leftarrow opt[i-1, W']$
6: $\quad\quad b[i, W'] \leftarrow$ N
7: $\quad\quad$ **if** $w_i \leq W'$ and $opt[i-1, W'-w_i] + w_i \geq opt[i, W']$ **then**
8: $\quad\quad\quad opt[i, W'] \leftarrow opt[i-1, W'-w_i] + w_i$
9: $\quad\quad\quad b[i, W'] \leftarrow$ Y
10: **return** $opt[n, W]$

Recover the Optimum Set

1: $i \leftarrow n, W' \leftarrow W, S \leftarrow \emptyset$
2: **while** $i > 0$ **do**
3: **if** $b[i, W'] = \mathsf{Y}$ **then**
4: $W' \leftarrow W' - w_i$
5: $S \leftarrow S \cup \{i\}$
6: $i \leftarrow i - 1$
7: **return** S

Running Time of Algorithm

1: **for** $W' \leftarrow 0$ to W **do**
2: $\quad opt[0, W'] \leftarrow 0$
3: **for** $i \leftarrow 1$ to n **do**
4: $\quad$ **for** $W' \leftarrow 0$ to W **do**
5: $\quad\quad opt[i, W'] \leftarrow opt[i-1, W']$
6: $\quad\quad$ **if** $w_i \leq W'$ and $opt[i-1, W'-w_i] + w_i \geq opt[i, W']$ **then**
7: $\quad\quad\quad opt[i, W'] \leftarrow opt[i-1, W'-w_i] + w_i$
8: **return** $opt[n, W]$

Running Time of Algorithm

```
1: for W' ← 0 to W do
2:     opt[0, W'] ← 0
3: for i ← 1 to n do
4:     for W' ← 0 to W do
5:         opt[i, W'] ← opt[i − 1, W']
6:         if w_i ≤ W' and opt[i − 1, W' − w_i] + w_i ≥ opt[i, W'] then
7:             opt[i, W'] ← opt[i − 1, W' − w_i] + w_i
8: return opt[n, W]
```

- Running time is $O(nW)$

Running Time of Algorithm

```
1: for W' ← 0 to W do
2:     opt[0, W'] ← 0
3: for i ← 1 to n do
4:     for W' ← 0 to W do
5:         opt[i, W'] ← opt[i − 1, W']
6:         if w_i ≤ W' and opt[i − 1, W' − w_i] + w_i ≥ opt[i, W'] then
7:             opt[i, W'] ← opt[i − 1, W' − w_i] + w_i
8: return opt[n, W]
```

- Running time is $O(nW)$
- Running time is pseudo-polynomial because it depends on value of the input integers.

Running Time of Algorithm

```
1: for W' ← 0 to W do
2:     opt[0, W'] ← 0
3: for i ← 1 to n do
4:     for W' ← 0 to W do
5:         opt[i, W'] ← opt[i − 1, W']
6:         if w_i ≤ W' and opt[i − 1, W' − w_i] + w_i ≥ opt[i, W'] then
7:             opt[i, W'] ← opt[i − 1, W' − w_i] + w_i
8: return opt[n, W]
```

- Running time is $O(nW)$
- Running time is pseudo-polynomial because it depends on value of the input integers.
- Game's running time: https://courses.csail.mit.edu/6.5440/fall23/

Avoiding Unncessary Computation and Memory Using Memoized Algorithm and Hash Map

compute-opt(i, W')

```
1: if opt[i, W'] ≠ ⊥ then return opt[i, W']
2: if i = 0 then r ← 0
3: else
4:     r ← compute-opt(i − 1, W')
5:     if w_i ≤ W' then
6:         r' ← compute-opt(i − 1, W' − w_i) + w_i
7:         if r' > r then r ← r'
8: opt[i, W'] ← r
9: return r
```

- Use hash map for opt

Outline

Knapsack Problem

Input: an integer bound $W > 0$

a set of n items, each with an integer weight $w_i > 0$

a value $v_i > 0$ for each item i

Output: a subset S of items that

$$\text{maximizes } \sum_{i \in S} v_i \qquad \text{s.t.} \sum_{i \in S} w_i \leq W.$$

Knapsack Problem

Input: an integer bound $W > 0$

a set of n items, each with an integer weight $w_i > 0$

a value $v_i > 0$ for each item i

Output: a subset S of items that

$$\text{maximizes } \sum_{i \in S} v_i \qquad \text{s.t.} \sum_{i \in S} w_i \leq W.$$

- Motivation: you have budget W, and want to buy a subset of items of maximum total value

DP for Knapsack Problem

- $opt[i, W']$: the optimum value when budget is W' and items are $\{1, 2, 3, \cdots, i\}$.
- If $i = 0$, $opt[i, W'] = 0$ for every $W' = 0, 1, 2, \cdots, W$.

$$opt[i, W'] = \begin{cases} & i = 0 \\ & i > 0, w_i > W' \\ & i > 0, w_i \leq W' \end{cases}$$

DP for Knapsack Problem

- $opt[i, W']$: the optimum value when budget is W' and items are $\{1, 2, 3, \cdots, i\}$.
- If $i = 0$, $opt[i, W'] = 0$ for every $W' = 0, 1, 2, \cdots, W$.

$$opt[i, W'] = \begin{cases} 0 & i = 0 \\ & i > 0, w_i > W' \\ & i > 0, w_i \leq W' \end{cases}$$

DP for Knapsack Problem

- $opt[i, W']$: the optimum value when budget is W' and items are $\{1, 2, 3, \cdots, i\}$.
- If $i = 0$, $opt[i, W'] = 0$ for every $W' = 0, 1, 2, \cdots, W$.

$$opt[i, W'] = \begin{cases} 0 & i = 0 \\ opt[i-1, W'] & i > 0, w_i > W' \\ & \\ & i > 0, w_i \leq W' \end{cases}$$

DP for Knapsack Problem

- $opt[i, W']$: the optimum value when budget is W' and items are $\{1, 2, 3, \cdots, i\}$.
- If $i = 0$, $opt[i, W'] = 0$ for every $W' = 0, 1, 2, \cdots, W$.

$$opt[i, W'] = \begin{cases} 0 & i = 0 \\ opt[i-1, W'] & i > 0, w_i > W' \\ \max \left\{ \begin{array}{c} opt[i-1, W'] \\ opt[i-1, W' - w_i] + v_i \end{array} \right\} & i > 0, w_i \leq W' \end{cases}$$

Exercise: Items with 3 Parameters

Input: integer bounds $W > 0, Z > 0$,

a set of n items, each with an integer weight $w_i > 0$

a size $z_i > 0$ for each item i

a value $v_i > 0$ for each item i

Output: a subset S of items that

$$\text{maximizes } \sum_{i \in S} v_i \qquad \text{s.t.}$$

$$\sum_{i \in S} w_i \leq W \text{ and } \sum_{i \in S} z_i \leq Z$$