Knapsack

You are robbing a store. The store has $n$ items, each with a weight $w_i$ pounds and a value $v_i$ dollars.

You can carry up to $C$ pounds. What is the maximum total value you can take?

$$f(k, w) := \text{maximum value using the first } k \text{ items & } \leq w \text{ total weight,}$$

$$f(0, w) = 0 \text{ if } w \geq 0$$

$$f(k, w) = \max \left( f(k-1, w) \quad \text{don't take item } k, \right.$$  
$$\left. f(k-1, w-w_i) + v_i \quad \text{if } w \geq w_i, \quad \text{do take item } k \right)$$

$O(nC)$ space & time
Knapsack Variants

Above is "0-1" Knapsack.

- Infinite multiplicity: can take many copies of each item.

\[
f(k, w) = \max \left( f(k-1, w) \left\{ \begin{array}{l} \text{\text{do not take item } k} \\ f(k, w-w_i) + v_i \left\{ \begin{array}{l} \text{\text{do take item } k} \\ \text{\text{can still reuse it}} \end{array} \right. \end{array} \right. \right)
\]

\[
f(w) = \text{opt. for weight } w \text{ w/all } n \text{ items:}
\]

\[
f(w) = \max (0, \max_{i \in [n]} f(w-w_i)+v_i)
\]

- High multiplicity:
  each item usable \( \leq n_i \) times

Easy: \( O(nC \cdot \leq n_i) \)

Straightforward: \( O(nC \cdot \leq \log n_i) \)

Tricky: \( O(nC) \)
- Sliding Window:

For regular knapsack, get \(O(C)\) space, \(O(C)\) time.

Each column only depends on previous column

\[ f(k \mod 2, w) \]

[Implementation trick: one column, scan down]

\( \Rightarrow O(nC) \) time, \( O(C) \) space

Issue: gives solution value but not solution
[would want the back pointers, which take \( nC \) space...]

Trick:

Instead of full \( nC \) back pointers, only store pointer to where the path was at column \( \frac{n}{2} \).
This can be kept in sliding window \( \Rightarrow O(C) \) space.
\[ \text{This finds one point on optimal path in } O(nC) \text{ time, } O(C) \text{ space. Call it } w_{\frac{n}{2}}. \]

Then

\[ \text{Path } (x_1, \ldots, n, C) = \text{Path } (x_1, \ldots, n, C - w_{\frac{n}{2}}) \]
\[ + \text{Path } (x_{\frac{n}{2} + 1}, \ldots, n, C - w_{\frac{n}{2}}) \]

\[ T(n, C) = nC + T(\frac{n}{2}, w_{\frac{n}{2}}) + T(\frac{n}{2}, C - w_{\frac{n}{2}}) \]

Total area remaining is \( \frac{nC}{2} \)

\( \Rightarrow O(nC) \) time.

\[ \text{Note: Knapsack is not polynomial time} \]

because \( C \) can be very large.

(Think: 64-bit integers \( \equiv n \cdot 2^{64} \))

Only fast if \( W_i \) small

Or: \( W_i \) small & \( V_i \) large, by

\[ f(k, v) = \min \text{ weight for given value} \]

instead of \( f(k, v) = \max \text{ value for given weight} \)