Knapsack

You are robbing a store.
Store has \( n \) items, each with a
weight \( W_i \) pounds
& value \( V_i \) dollars

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

\[
f(K, W) := \text{maximum value using the first } K \text{ items & } \leq W \text{ total weight.}
\]

\[
f(0, W) = 0 \text{ if } W > 0
\]

\[
f(K, w) = \max \left\{ \begin{array}{l}
f(K-1, w) \quad \text{don't take item } K \\
\min \left\{ f(K-1, w-W_i) + V_i \quad \text{if } w \geq W_i \right\} \quad \text{do take item } K
\end{array} \right. 
\]

\( O(nC) \) space & time
Knapsack Variants

Above is "0-1" Knapsack.

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

\[
f(k, w) = \max \left( \begin{array}{c}
f(k-1, w) \quad \text{< don't take item } k \\
f(k, w-w_i) + v_i \quad \text{< if } w \geq w_i \\
\text{< do take item } k \\
\text{< can still reuse it}
\end{array} \right)
\]

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

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

\[O(C) \text{ space, } O(nC) \text{ time.}\]

- high multiplicity:

  each item usable \( \leq k_i \) times

  Easy: \( O(C \cdot \max k_i) \)

  Straightforward: \( O(C \cdot \sum \frac{1}{k_i}) \)

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

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

  Each column only depends on previous column

  \( \Rightarrow \) only need to track two columns

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

  [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:**

- Solution path
- Back pointer to location of path on column \( n/2 \)

Instead of full \( nC \) back pointers, only store pointer to where the path was at column \( n/2 \).

This can be kept in sliding window \( \Rightarrow O(C) \) space.
This finds one point on optimal path in \( O(nC) \) time, \( O(C) \) space.

Call it \( w_{\frac{n}{2}} \).

Then

\[
\text{Path}(\varepsilon_1, \ldots, n_2, C) = \text{Path}(\varepsilon_1, \ldots, \frac{n_2}{2}, w_{\frac{n}{2}}) + \left( \frac{n_2}{2}, w_{\frac{n}{2}} \right) + \text{Path}(\frac{n_2}{2}+1, \ldots, n_2, C - w_{\frac{n}{2}})
\]

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

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

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

---

**Note**  
Knapasck is **not** polynomial time

because \( C \) can be very large.

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

Only fast if \( W_i \) small

(or: \( v_i \) small & \( W_i \) large, by

\( f(k, v) = \min \) weight for given value

instead of \( F(k, v) = \max \) value for given weight

)