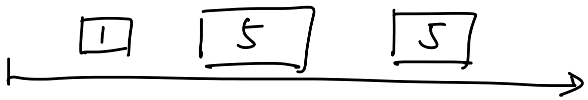
Buying Toilet Paper

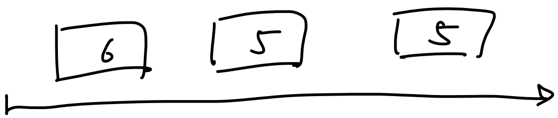
Greedy: local optimal choices leads to global optimum.

Suppose capacity 10.



sort increasing

from left to right, grab whatever can fit. $\Longrightarrow$ Greedy gives 6
Global gives 10.



sort decreasing

$\Longrightarrow$ Greedy gives 6
Global gives 10

def: knapsack problem.

- weight $w_1, \dots, w_n \in \mathbb{N}$ — constraint $W \in \mathbb{N}$
- value $v_1, \dots, v_n \in \mathbb{N}$

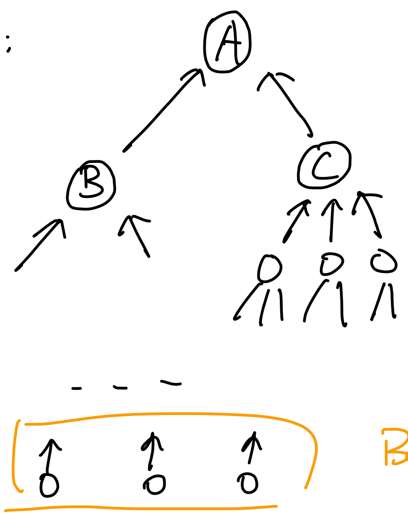
Goal: Compute

$$\max_{\substack{S \subseteq [n] \\ \sum_{i \in S} w_i \le W}} \sum_{i \in S} v_i$$

prop knapsnap solvable in $O(n \cdot 2^n)$

Using DP in Knapsnap: recurse and memoize → compress $\boxed{\text{rec tree}}$ into $\boxed{\text{recursion DAG}}$ ☆ interesting

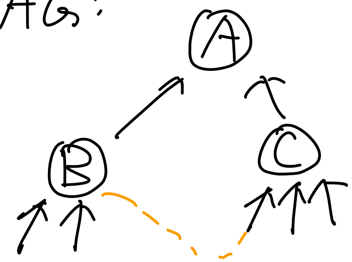
Tree:



Base cases

→ exponential size

DAG:



if B was used as subproblem of C previously.

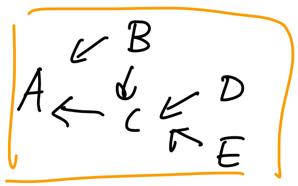
No cycles: Otherwise, cannot be solved.

Directed: problem and subproblem

} → DAG

Idea: solve recursive DAG by:

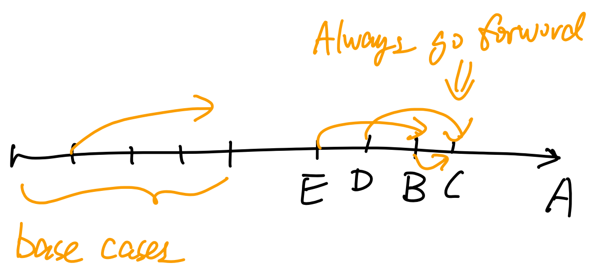
- memoization: top down
  - implicitly creates DAG



- iterative: bottom up

preferred

  - explicitly creates DAG w/ topological sort.



Back to knapsnap:

Given $w_1, \cdots w_n, W \in \mathbb{N}$

$v_1, \cdots, v_n \in \mathbb{N}$

idea: WIS

Think about - subproblems

- relationship of

def: $OPT(k) = \max_{\substack{S \in \boxed{[k]} \\ \sum_{i \in S} w_i \le W}} \sum_{i \in S} v_i$

prop $\{S : S \subseteq [k-1], \sum_{i \in S} w_i \le W\} \subseteq \{S : S \subseteq [k], \sum_{i \in S} w_i \le W\}$

$\implies OPT(k) = OPT(k-1)$

cor If exists OPT s.t. $S \subseteq [k]$ for $OPT(k)$, $k \notin S$, $\implies OPT(k) = OPT(k-1)$

Problem: If we use w_k, then subproblem has weight contraint $W - w_k$. diff!

def:

weight constraint

$$OPT(k,t) = \max_{\substack{S \subseteq [k] \\ \sum_{i\in S} w_i \le t}} \sum_{i \in S} v_i$$

prop

$$\{ S \subseteq [k] : \sum_{i\in S} w_i = t \} = \boxed{\{ S \subseteq [k-1] : \sum_{i \in S} w_i = t \}}$$

OPT(k-1, t)

$$\cup \boxed{\{ S = \{k\} \cup T : T \subseteq [k-1], \sum_{i \in S} w_i \le t \}}$$

OPT(k-1, t-1)

$$\Downarrow$$

$$\sum_{i \in T} w_i \le t - w_k$$

derive a RR

- if $w_k > t$, $OPT(k,t) = OPT(k-1,t)$ too heavy!

- $$OPT(k,t) = \max \begin{cases} OPT(k-1,t) \\ OPT(k-1, t-w_k) + v_k \end{cases}$$

Then we add memoization :

prop: knapsnap in $O(n \cdot W)$

Algo:

- array $M[0,..,n][0,...,W]$

- for $0 \leq t \leq W$
  - $M[0][t] = 0$

- for $1 \leq k \leq n$
  for $0 \leq t \leq W$
  if $w_k > t$, $M[k][t] = \boxed{M[k-1][t]}$ solved

  else
  $$M[k][t] = \max \begin{cases} M[k-1][t] \\ M[k-1][t-w_k] + v_k \end{cases}$$

- return $M[n][W]$

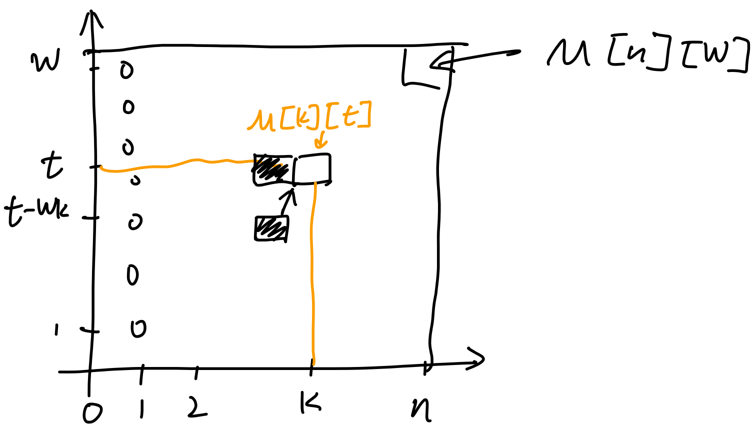
Correctness: clear as this implements RR

complexity: Two loops, unit operations, $O(n \cdot W)$

Find OPT w/ solution? Yes:

prop: Given filled M, can read of solution from M in $O(n)$

Pf sketch:

$M =$ 

Idea: $OPT(k,t)$ depends on these two possible solutions.

Do comparison, and recurse down.

$$OPT(k,t) \text{ solution } = \begin{cases} OPT(k-1,t) & \text{if } M[k-1][t] > M[k-1][t-w_k] \\ OPT(k-1, t-w_k) \cup \boxed{\{k\}} & \text{o.w.} \end{cases}$$

Q: is this efficient?

A. yes if $W \le n^{O(1)}$ $\Rightarrow$ normal

B. no if $W = 2^n$ $\Rightarrow$ arithmetic is still cheap def: algo on integers $a_1, \ldots, a_n$

$$\mathrm{poly}\left(\sum |a_i|\right)$$

$$\mathrm{poly}\left(\sum |\lg a_i|\right)$$