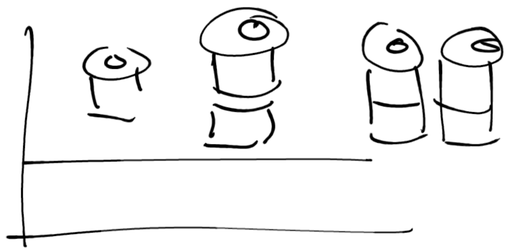
cs 473 Algorithms: Lecture 4 (2022-01-27)

logistics:
- pset 0 — revised
  - due F17
- pset 1 — out F17
- lectures in-person next week

last time: dynamic programming
- weighted interval scheduling
  - brute force: $O(n \cdot 2^n)$
  - recursive: $O(2^n)$
  - memoized recursive: $O(n)$
  - iterative: $O(n)$

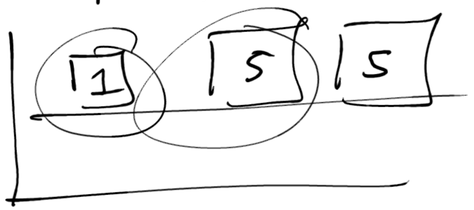
today: dynamic programming

Q: buying tailor paper during pandemic?



Q: can we fill [greedily]?
- natural algorithmic paradigm
- when [local] optimal ⇒ [global] optimum

eg: capacity 10 knapsack



sorted in increasing order

⇒ pickup $1+5=6<10$

$\underbrace{5+5}_{OPT}$

⇒ greedy fails

eg: capacity 10

sorted decreasing order

picks $6<10=\underbrace{5+5}_{OPT}$

⇒ greedy fails

Q: [also] optimize / [quickly] at tailor pgm?

def: the knapsack problem is given

- weights $w_1, \dots, w_n \in \mathbb{N}$
- weight limit $W \in \mathbb{N}$
- values $v_1, \dots, v_n \in \mathbb{N}$

compute $$\max_{S \subseteq [n]} \sum_{i \in S} v_i$$
$$\text{s.t.} \sum_{i \in S} w_i \leq W$$

$\uparrow$ arg

the subset sum problem is when $v_i = w_i$

rmk: - items of negative value can be ignored if all weights are nonneg

- items of negative weight - non physical
  - can be handled similarly

Q: any algos?

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

Q: efficient algo?

dynamic programming?

Q: efficient also?

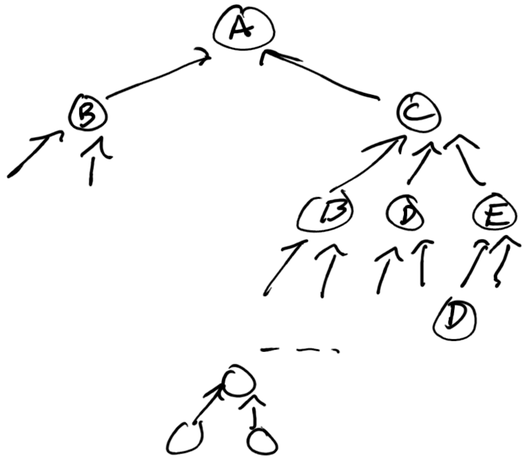
dynamic programming?

idea (dynamic programming)

compress recursion tree into recursion directed acyclic graph DAG

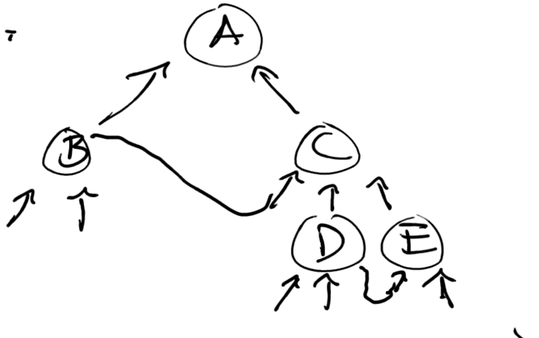
- identify small set of subproblems
- identify relations between subproblems

eg: tree:



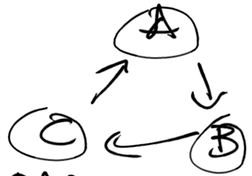
exp - many nodes

DAG:



poly many in DAG

no cycles:

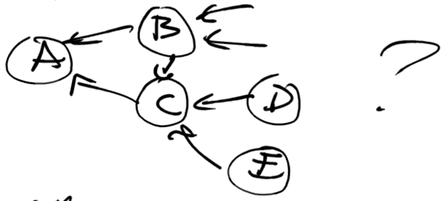


idea: solve recursion DAG

(a) recursion - top-down

- implicitly exploring DAG

eg:

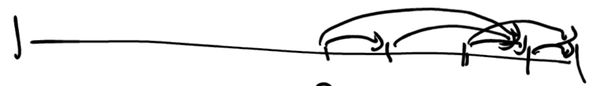


?

(b) iterative: - bottom up

- explicitly construct DAG, ↓ topological sort

→ time



D E B C A

subproblems: - design iterative DP

- clearly indicate subproblems

def: knapsack. weights $w_1, \dots, w_n, W$

values $v_1, \dots, v_n$

compute $\max_{\sum_{i \in S} w_i \le W} \sum_{i \in S} v_i$?

Q: dynamic programming

subproblems?

relations?

idea: mimic weighted interval scheduling

def: $OPT(k) = \max_{S \subseteq [k]} \sum_{i \in S} v_i$ s.t. $\sum_{i \in S} w_i \le W$

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

$\{$ feasible sols for $OPT(k-1)\}$

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

$\supseteq \{$ feasible sols for $OPT(k)\}$

cor: $OPT(k) \ge OPT(k-1)$

cor: if exists optimum sol $S \subseteq [k]$ s.t. $k \notin S$

then $OPT(k) = OPT(k-1)$

Q: if $k \in S$?

then k has weight w_k

$\Rightarrow$ leaves weight $W - w_k$ for rest of items

then a subproblem we've considered

idea: expand subproblems

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

prop: $\{ S \subseteq [k] : \sum_{i \in S} w_i \le t \}$ ← feasible soln to $OPT(k,t)$

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

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

$\{ \{k\} \cup T : T \subseteq [k-1], \sum_{i \in T} w_i \le t - w_k \}$ ← feasible soln to $OPT(k-1, t-w_k)$

rmk: disjoint decomposition

cor: if $w_k > t$

$OPT(k,t) = OPT(k-1,t)$

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

prop: knapsack solvable in $O(n \cdot W)$ time

pf: also $O(1)$ array $M[0,\dots,n][0,\dots,W]$

$O(W)$ (2) for $0 \le t \le W$

$O(1)$ (a) $M[0][t] = 0$

$O(n)$ (3) for $1 \le k \le n$

$O(W)$ (a) for $0 \le t \le W$

$O(1)$ (i) if $w_k > t$, $M[k][t] = M[k-1][t]$

$O(1)$ (ii) else $M[k][t] = \max \{ M[k-1][t], M[k-1][t-w_k] + v_k \}$

(4) return $M[n][W]$

correctness: clear

complexity: clear

□

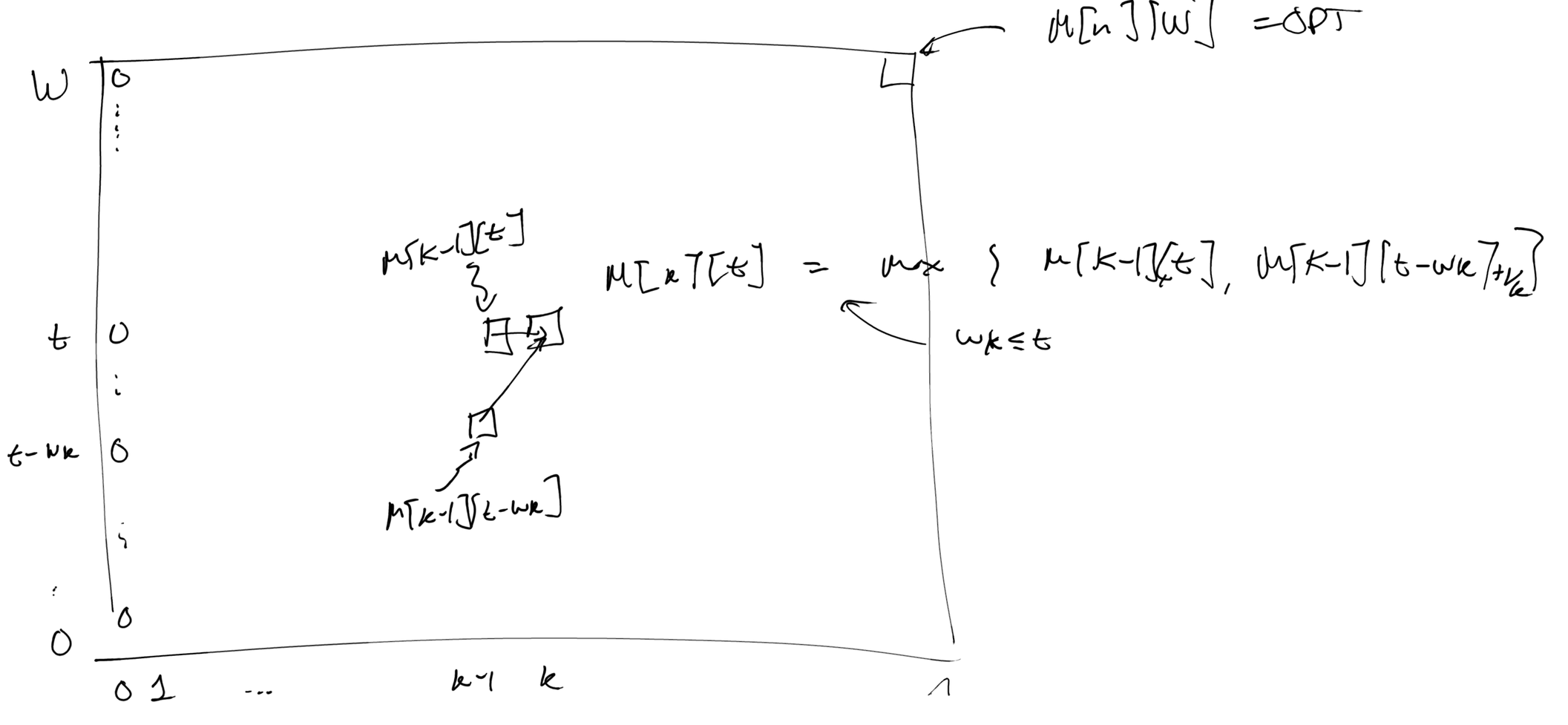
Q: find optimal (solution)?

prop: given filled array $M[\cdot][\cdot]$ from above optimal (solution) can be computed in $O(n)$ time.

Ca: $w_k \le t$

(exists) OPT soln S for $M[k][t]$ w/ item k iff $M[k-1][t-w_k] + v_k \ge M[k-1][t]$

eg:



Q: is this actually efficient?

yes, if $W \in n^{O(1)}$

this is true in context of course of $O(\lg n)$-bit integers

no, if say $W = 2^n$

but arithmetic still efficient here

def: an algorithm on n integers $a_1, \ldots, a_n$ is pseudo polynomial if it runs in

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

rmk: polytime would be $\text{poly}\left(\sum O(\lg |a_i|)\right)$

bit complexity of input

or: knapsack has a pseudopoly time algo

thm: knapsack is NP-complete when W is allowed to be exponential

today: dynamic programming

- abstract - trees vs DAGs
- memoization vs iterative
- knapsack - parameterize subproblems by new variables

- next lecture: DP

logistics: - pset 0 - revised - due F/17
- pset 1 - due F/17
- lecture interval next week

- pseudopoly time also