Dynamic Programming (part 1) Lecture 6

PB

NO D&C

- Brute Force (try all possib)
- approximation

Divide & Conquer

OPTSOL = SOL1 SOL2

SPB1 decision (split) SPB2

GREEDY

- Divide/split/decide
  $\Rightarrow$ implies SUBPB
- Solve SUBPBs
- Sol = combine(subpb-solutions)

DP

- consider many/all possible splits
- solve many SUBPBs (some not going to be used)
- make decision/divide/split BASED ON SUBPB solutions
- consider SUBPB (decision) already solved?
- SOL = combine(subpb-solutions)

DP writing recipe (required)

1. Characterize OPTSOL = function (subpb-optsol) ⟹ D&C

2A. recurrence of the objective $C[\overset{pb}{\text{input}}] \overset{\text{formula}}{=} C[\text{subpb}]$

2B. visual table of PB—SUBPB dependencies.

3. bottom up computation : solve $\boxed{\text{all}}$ subpb in the table
   (Pseudocode)
   in what order?

4. Trace solution (if necessary)

DP1 Rod Cutting

n length wire (rod)

given price table

length (ft)	1	2	3	4	5	6
price/value	1	2	4.5	6.4	8	7
val/length av	1	1	1.5	1.6	1.6	7/6

task: cut wire for optimal total value.

Greedy? $n = 9$ — $6+3$ value 11.5

NO in general

$3+3+3 \Rightarrow 13.5$

$5+4$ $\boxed{\text{Greedy}}$ $\Rightarrow 8+6.4$

OPT SOL

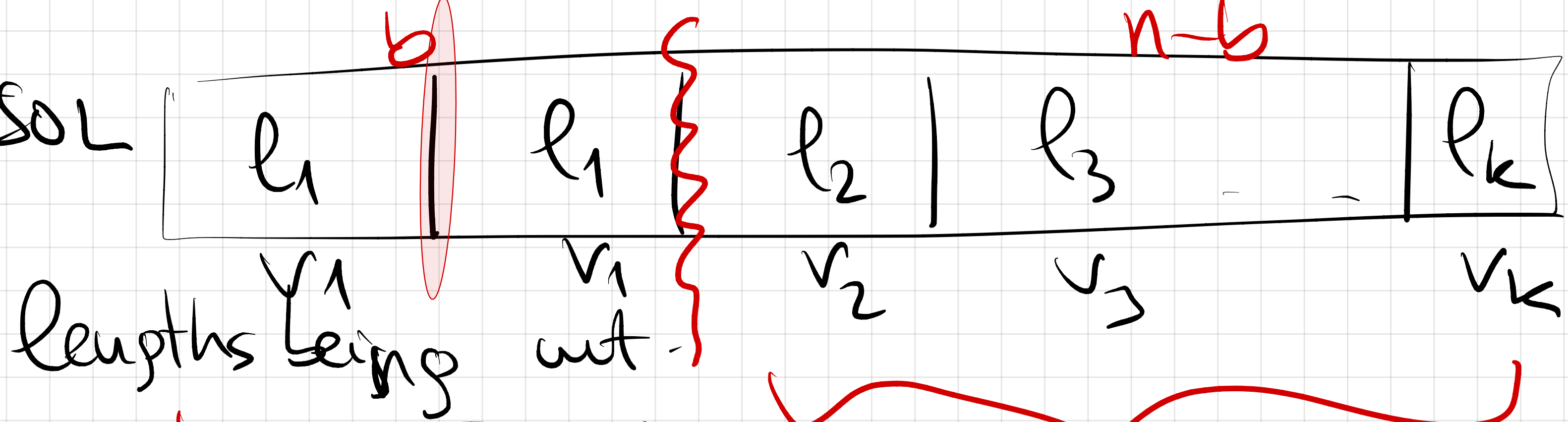
14.4

$n=11$ Greedy $5+5+1 \Rightarrow 17$

$4+4+3 \Rightarrow 17.3$

OPT SOL

1 OPTSOL



$\Downarrow$ OPTSOL for that length b

$\Downarrow$ OPT for length n−b

works by exchange arg

so D&C

2 A DP objective recurrence input=n

search ~~select~~ first length (k)

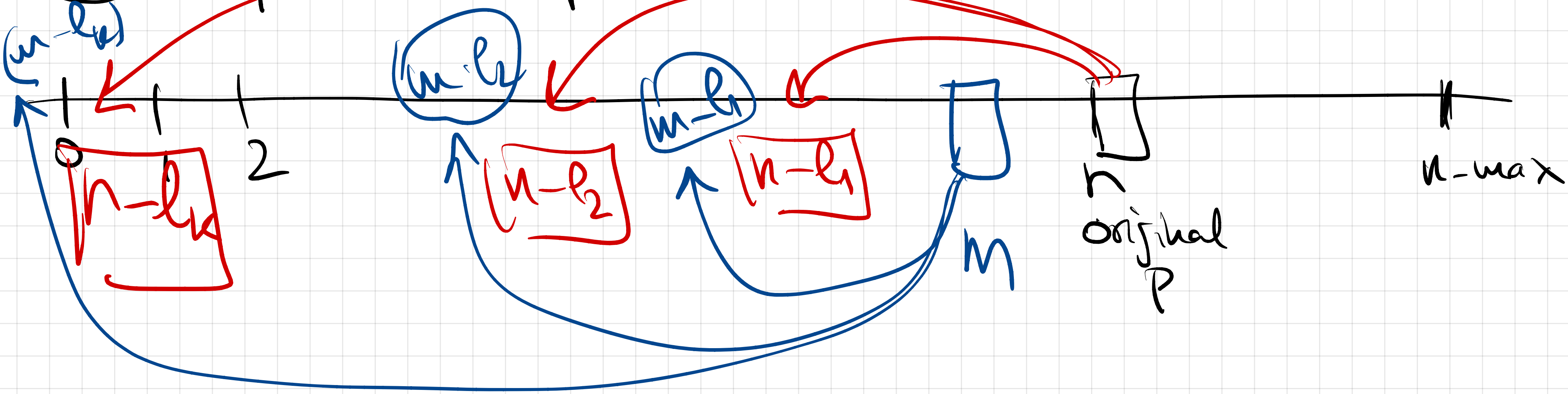
subpb

$$C[n] = \max_{1 \le k \le n} \{ v_k + C[n - \ell_k] \}$$

value added

best we can do for input

(2B) Subpb table dependencies : unidim



(3) bottom-up computation: solve all problems in table

order: left → to right

$C[0] = 0$

for $n = 1 : n\_max$ given

// search for k best = 0 bestk = -1

for $k = 1$: all lengths possible

if $n - l_k < 0$ skip

if $(v_k + C[n-l_k] > best)$

best $= v_k + C[n-l_k]$

$k <$ # of lengths in the table

$\Theta(nk)$

best_k = k

c[n] = best

s[n] = best_k // what cut k achieves obj c[n]

④ Trace Solution + add in pseudocode

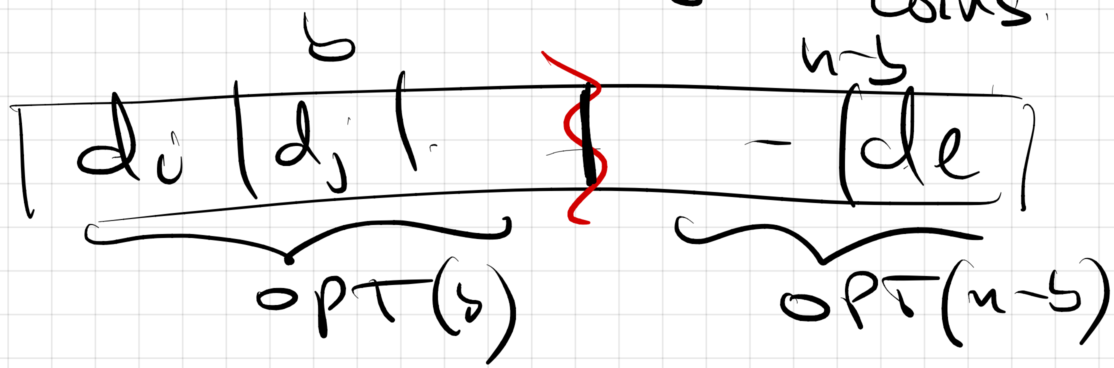
s[same input as c] = the choice made

```
PrintSolution (n)
    output k = s(n) => l_k, v_k
    PrintSolution (n - l_k)
    unless n ≤ 0
```

DP2 Given change to n cents with min #coins

coin denom $= \{ d_1 = 1, d_2, d_3, \ldots, d_k \}$ ∞ amount coins

1 already done

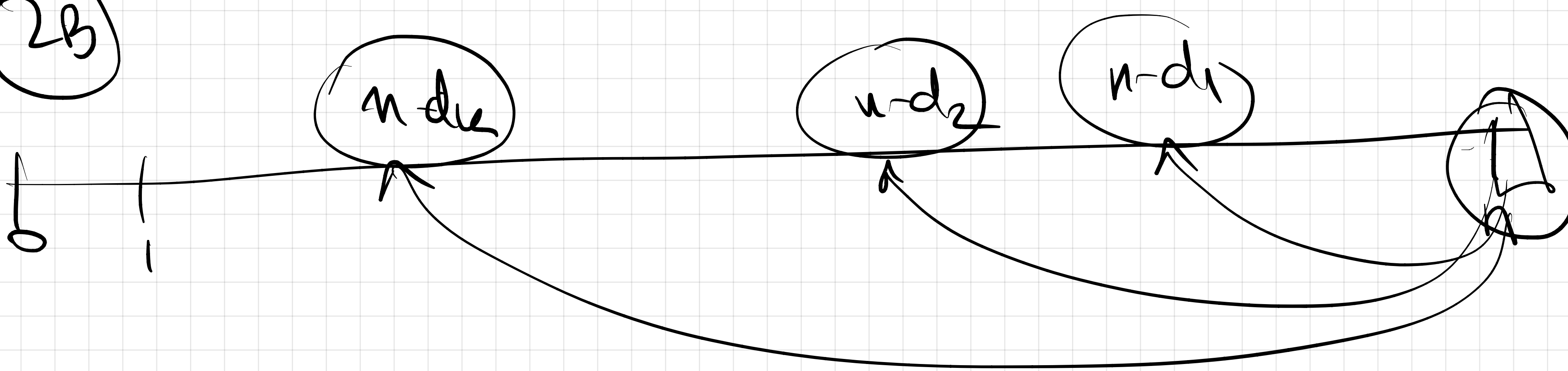


2 $C[n] = \min_k \left\{ \text{search for first coin } k \quad 1 + C[n - d_k] \right\}$

val added to OBJ

subpb

2B



(3) bottom up comp $L \to R$

$c[0] = 0$

For $n = 1 : n\_max$ — input counts

init: $best = \infty$ $best\_k = -1$

for $k = 1$: last denom $\leq n$

if $(1 + c[n - d_k] < best)$ then $\begin{cases} best = 1 + c[n - d_k] \\ best\_k = k \end{cases}$

$c[n] = best$ // the # of coins (min)

$s[n] = best\_k$ // the coin

$c[n\_max] = value$

without s

$c[n\_max]$

try ever coin k

check

$c[n\_max] = 1 + c[n\_max - d_k]$

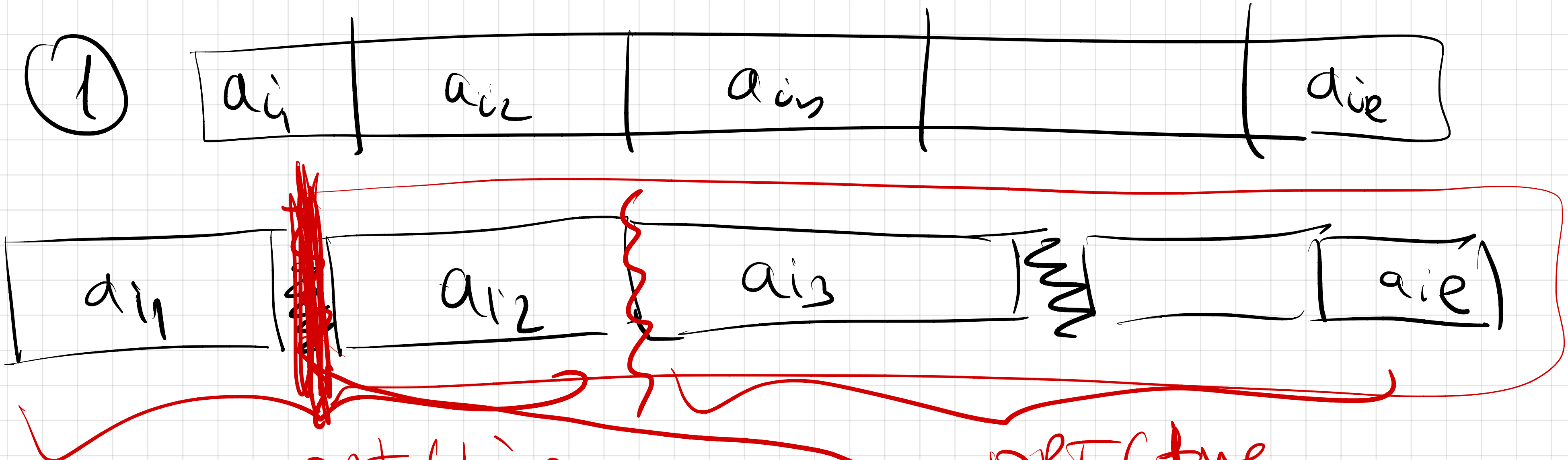
(4) Trace

Exercise Activities Selection

$S_1, S_2, \ldots S_n$
$F_1, F_2, \ldots F_n$ } times

OBJ: max total time

(1)



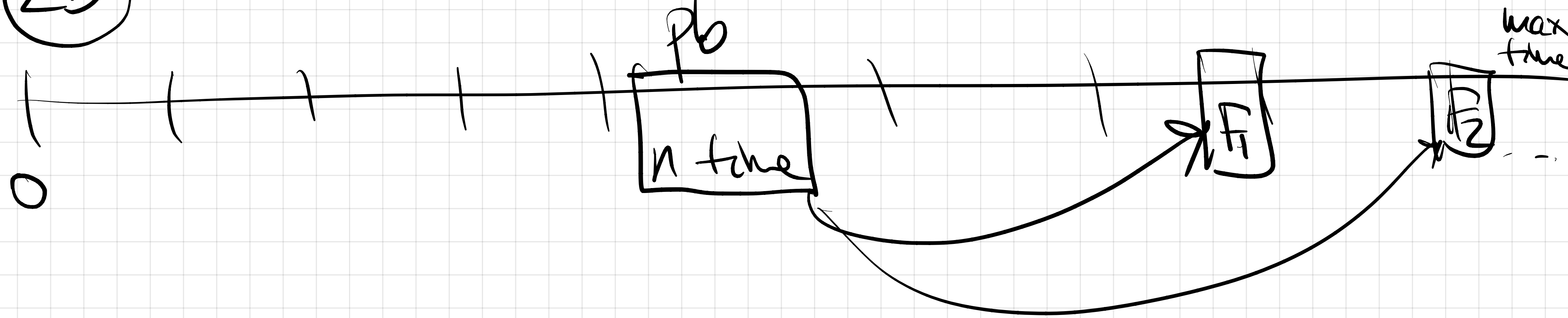
opt(time Left) OPT(time Right)

(2A) $C[\overset{n}{\text{start time}}] = \max_{k} \left\{ \underbrace{\Delta k}_{F_k - S_k} + C[F_k] \right\}$

add to obj

search first activity k ($S_k \geq n$)

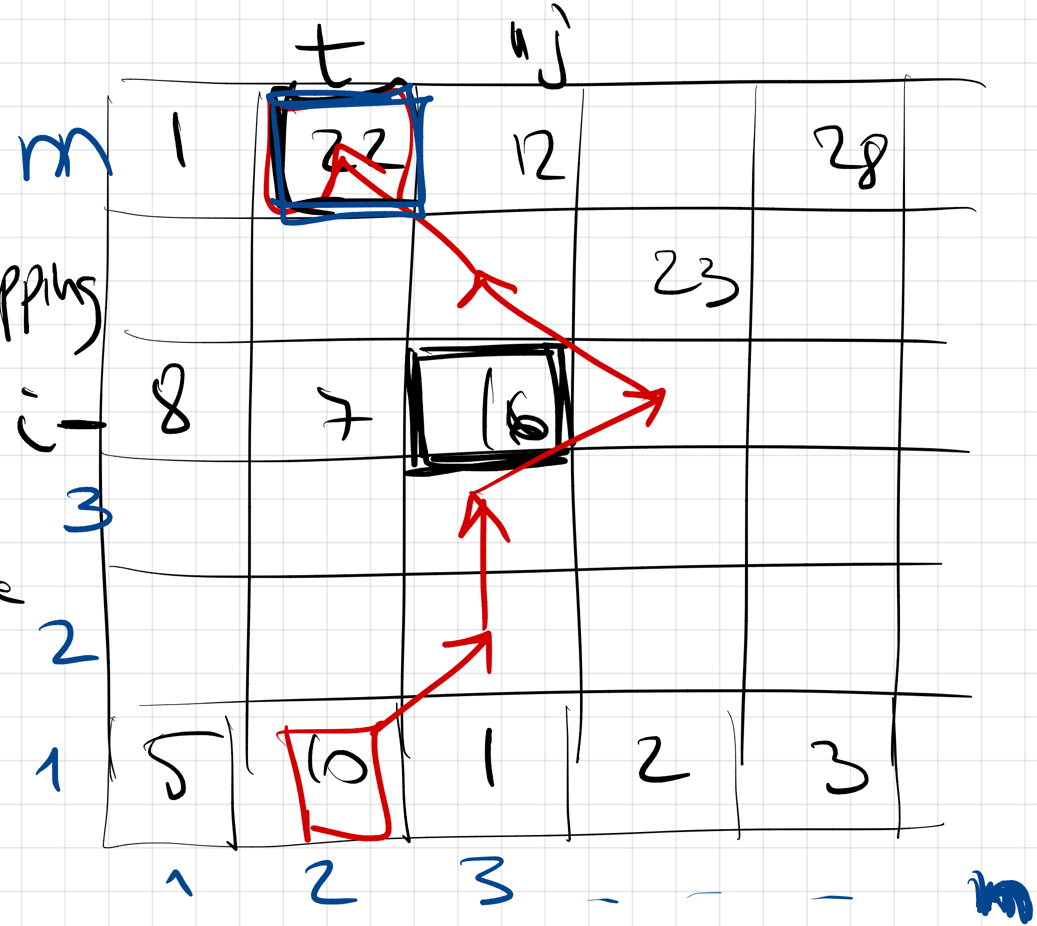
2B



DP4 CheckBoard

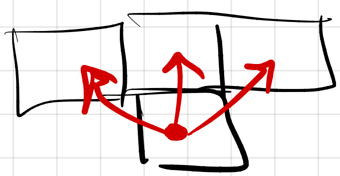
given,

$P[i,j]$ = penalty for stepping on cell $[i,j]$



Task walk from anywhere row=1 to anywhere on row = m

3 valid moves



m rows, n columns

minimize total penalty

• assume table = cylinder column $n+1$ = column 1

column -1 = column n

For step 1, reformulate pb: min path from first row to cell (m,t) on last row

If path $\underbrace{\text{row } 1 \to \text{cell}}_{\text{optimal}}\ \boxed{i,j}\ \underbrace{\to \text{row } m}_{\text{optimal}}$ is optimal

optimal row 1 to cell i,j

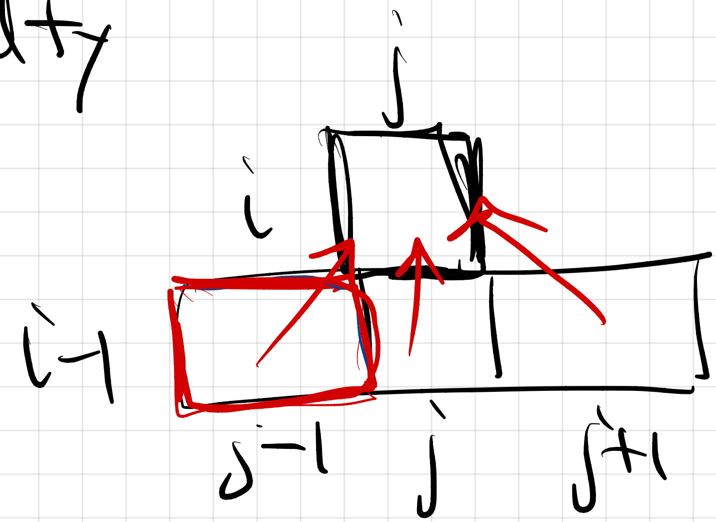
optimal anywhere row m down to cell (i,j)

(2A) $C[\text{row } 1 \text{ to cell } i,j]$ = total penalty

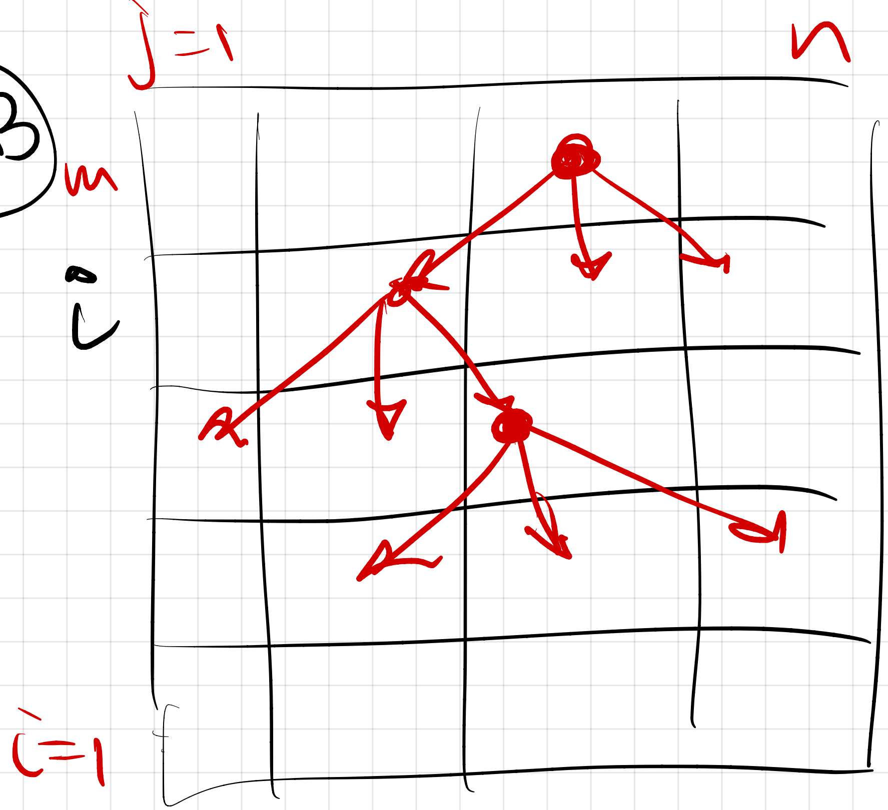
search last move from $(i-1,j-1)$

$$P(i,j) + \min \begin{cases} C[i-1,j-1] \\ C[i-1,j] \\ C[i-1,j+1] \end{cases}$$

2D table



2B



$C[i,j]$ fill all table

$1 \le i \le m$; $1 \le j \le n$

constraint: previous row must be already computed.

each row: left → right

3

first row: $c[1,j] = P(1,j]$ $\forall j$

for $i = 2:m$ //row order matters

 for $j = 1:n$

  $k = \underset{t}{\operatorname{argmin}} \{ c[i-1, j-1], c[i-1, j], c[i-1, j+1] \}$

  $c[i,j] = \underbrace{P[i,j]}_{\text{penalty}} + c[i-1, k]$

  $s[i,j] = k$

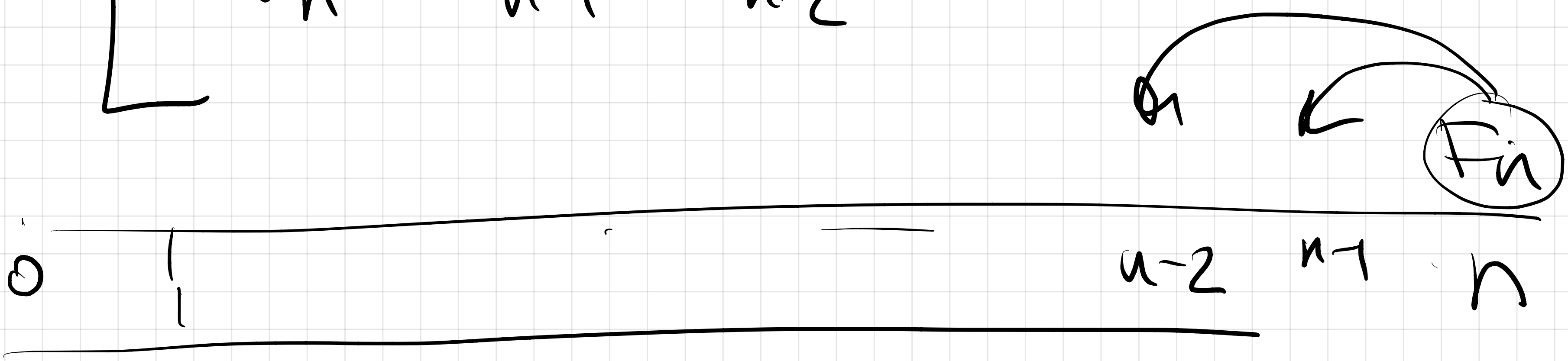
Simple DPs look like not DP

$F_0 = 0$ $F_1 = 1$

for $n = 2 : \max n$

- $F_n = F_{n-1} + F_{n-2}$

$F_n = F[n]$



$\frac{n!}{k!(n-k)!} = \binom{n}{k}$ = n choose k

choose subset of size k out of n elements

$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ sum rule

$\{1, 2, \dots, n\}$

- choose "n"
- not choose "n"

$\Longrightarrow$ Pascal Δ

1 1

1 2 1

1 3 3 1

1 4 6 4 1

$C[n,k] = \binom{n}{k}$

for $n = 1 : \text{max-}n$

for $k = 1 : n$ $\qquad \Theta(n^2)$

$C[n,k] = C[n-1, k-1] + C[n-1, k]$

DP5 Discrete Knapsack items (value, weigths)

whole or nothing

$v_1 \; v_2 \; \cdots \; v_n$

$w_1 \; w_2 \; \cdots \; w_n$

Z = Knapsack total weight

OBJ: max total value

$\sum$ weights $\leq Z$

critical

- diff vs coins, rod-cuts:

  table changes because each item can be used 0/1 times.

  itemset (or similar) must be part of changing input

1 Knapsack

optimal $C[b; \text{all items}]$

optimal $C[n-b; \text{all items not on left}]$

Lecture 7

allitems = set

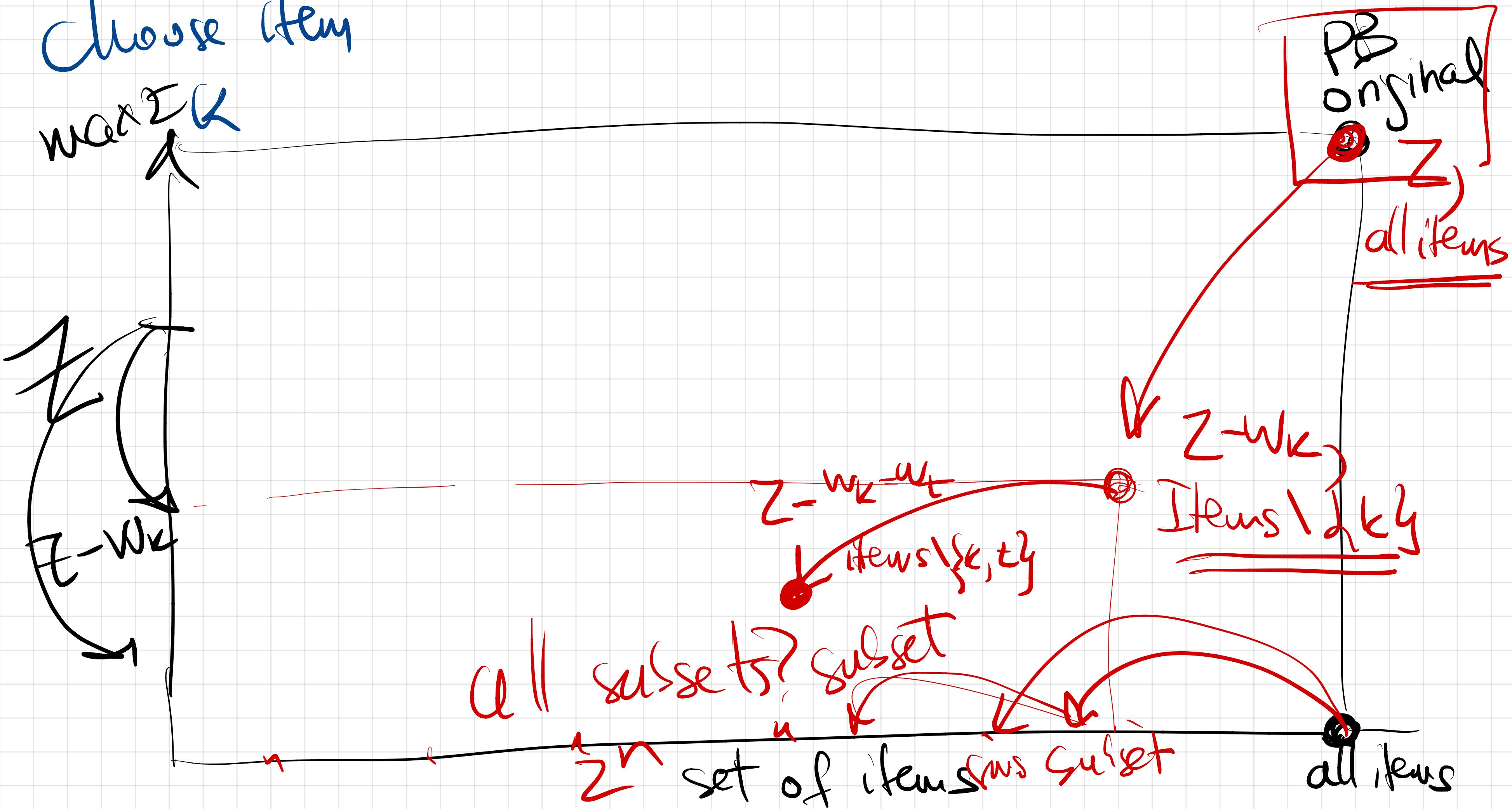
② tentative ? wishful thinking

A

$$C[Z, \text{all items}] = V_K + C[\underline{Z - W_K}, \underline{\text{all items} \setminus \{k\}}]$$

choose item

B



Indexing Trick: index elements global order
1, 2, 3, . . . n   not changeable

not necess. $\boxed{\text{input order}}$, weight-sort, value-sort

Item set $I_n = \{1, 2, \ldots n\}$ $I[1:n]$

Partial sets $I_{n-1} = \{1, 2, \ldots n-1\}$ $I[1:n-1]$

$I_{n-2} = \{1, 2, \ldots n-2\}$ $I[1:n-2]$

$I_4 = \{1, 2, 3, 4\}$ $I[1:4]$

$\Theta(n)$ subsets (prefix)

step 1?

knapsack   items available   item chosen   prefix set

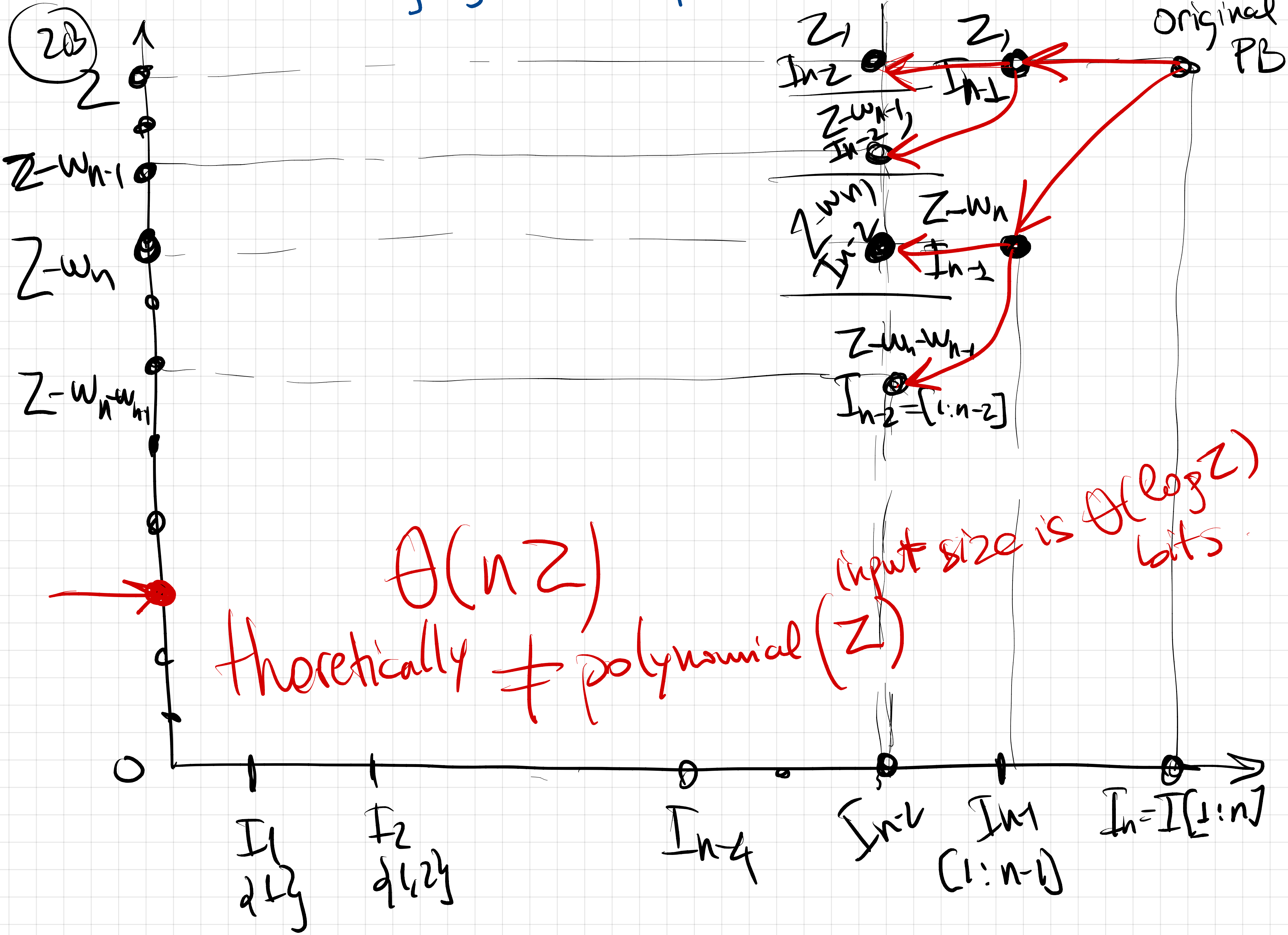
step 2A $C[Z, I_n] = \max\left(v_{\cancel{x}\,n} + C[Z - w_{\cancel{x}\,n}, I_{n-1}] \right.$, $\left. C[Z, I_{n-1}] \right)$

search (max)   ~~or~~   don't use item n

$\Theta(1)$

$S[Z, I_n] = 1$ if n is used, 0 if not

wishful thinking: use prefix subsets (indexing trick)



$\Theta(nZ)$

theoretically $\neq$ polynomial(Z)

input size is $\Theta(\log Z)$ bits

Y axis $Z, Z-w_n, Z-w_n-w_{n-1}, Z-w_{n-1}$

Z - subset of weights

• requirement: Z, w integers (discrete range)

example $I = [1:15]$

step 1: OPTSOL

i_1 2	i_2 5	i_3 6	. . .	i_{m-1} 7	i_m 9

assume OPTSOL its presented in global order

$\text{OPTSOL}_{Z,\ \text{all items}} \Rightarrow \text{OPTSOL}_{Z,\ I[1:i_m]} \Rightarrow i_m$ item $+$ OPTSOL $(Z - w_m, I_{i_m - 1})$

$+$ → shifted

$1, 2, \ldots, i_m - 1$

$1:8$

Matrix Chain Multiplication

ex $A \times (B \times C) = (A \times B) \times C$

$n=3$ $\quad 30\times 5 \quad 5\times 10 \quad 10\times 20 \qquad\qquad 30\times 5 \quad 5\times 10 \quad 10\times 20$

cell

$5\times 10\times 20$ $\Downarrow$ 5×20

$30\times 5\times 20$

$30\times 5\times 10$ $\Downarrow$ 30×10

$30\times 10\times 20$

total $3000 + 1000$

last operation

total $1500 + 6000$

STEP 1

optimal

optimal

$n=n$

$$A_1 \cdot A_2 \cdot A_3 \; \cdots \; A_k \cdot A_{k+1} \; \cdots \; A_n$$

$p_0\times p_1 \quad p_1\times p_2 \quad p_2\times p_3 \quad p_{k-1}\times p_k \quad p_k\times p_{k+1} \quad p_{n-1}\times p_n$

$p_0 \times p_k \qquad p_k \times p_n$

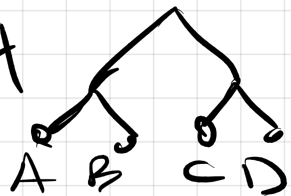
What is the best multip order? $\equiv$ putting parenthesis

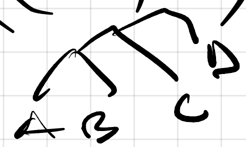
last multip (op) : $p_0 \times p_k \times p_n$

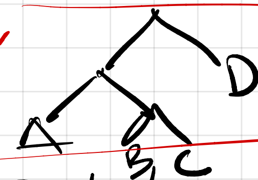
Brute force: Try all possible parenthesis

$(AB)(CD)$ | $((AB)C)D$ | $(A(BC))D$ | $A((BC)D)$ | $A(B(CD))$

$n=4$







#ways to parenthesis $=$ Catalan$(n) \simeq 4^n$ exact $\binom{2n}{n} - \binom{2n}{n-1}$

last

$(A((BC)D))(EF)$

step 2

$A_1 \cdot A_2 \ldots \cdot A_i \cdot A_{i+1} \ldots A_k \mid A_{k+1} \ldots A_j \, A_{j+1} \ldots A_n$

$C[i,j]$

$p_{i-1} \times p_k \quad p_k \times p_j$

$C[i,j]$ = best cost for multiplying $A_i \cdot A_{i+1} \ldots \cdot A_j$

$i \le j$

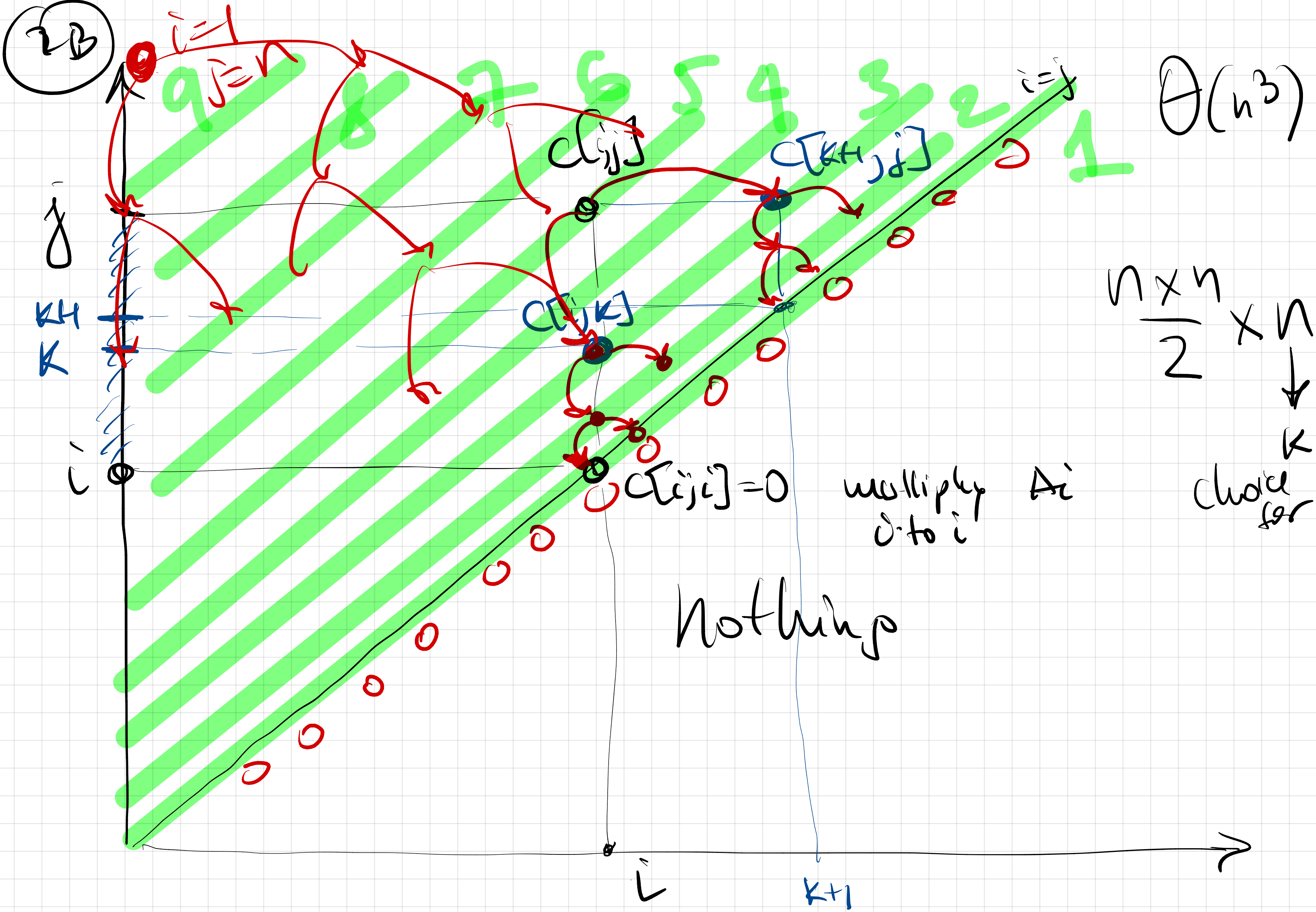
cost last op

look for last op (k)

$p_{i-1} \cdot p_k \cdot p_j + C[i,k] + C[k+1,j]$

$\boxed{i < k < j}$

$S[i,j] = k$


2B
i=1
j=n
9 8 7 6 5 4 3 2 1
i=j
c[i,j]
c[k+1,j]
c[i,k]
c[i,i]=0
multiply Ai 0 to i
j
k+1
k
i
nothing
i
k+1
Θ(n^3)
n×n/2 ×n
k
choice for

MEMOIZED-MATRIX-CHAIN(p)

1 $n = p.length - 1$
2 let $m[1..n, 1..n]$ be a new table
3 **for** $i = 1$ **to** n
4   **for** $j = i$ **to** n
5         $m[i, j] = \infty$
6 **return** LOOKUP-CHAIN$(m, p, 1, n)$

LOOKUP-CHAIN(m, p, i, j) → recursive

1 **if** $m[i, j] < \infty$
2   **return** $m[i, j]$ → m = C
3 **if** $i == j$
4     $m[i, j] = 0$
5 **else for** $k = i$ **to** $j - 1$
6         $q =$ LOOKUP-CHAIN(m, p, i, k)
            $+$ LOOKUP-CHAIN$(m, p, k+1, j) + p_{i-1} p_k p_j$
7   **if** $q < m[i, j]$
8             $m[i, j] = q$ → choice s[i,j]=k
9 **return** $m[i, j]$

The MEMOIZED-MATRIX-CHAIN procedure, like MATRIX-CHAIN-ORDER, maintains a table $m[1..n, 1..n]$ of computed values of $m[i, j]$, the minimum number of scalar multiplications needed to compute the matrix $A_{i..j}$. Each table entry initially contains the value ∞ to indicate that the entry has yet to be filled in. Upon calling LOOKUP-CHAIN(m, p, i, j), if line 1 finds that $m[i, j] < \infty$, then the procedure simply returns the previously computed cost $m[i, j]$ in line 2. Otherwise, the cost is computed as in RECURSIVE-MATRIX-CHAIN, stored in $m[i, j]$, and returned. Thus, LOOKUP-CHAIN(m, p, i, j) always returns the value of $m[i, j]$, but it computes it only upon the first call of LOOKUP-CHAIN with these specific values of i and j.

Figure 15.7 illustrates how MEMOIZED-MATRIX-CHAIN saves time compared with RECURSIVE-MATRIX-CHAIN. Shaded subtrees represent values that it looks up rather than recomputes.

Like the bottom-up dynamic-programming algorithm MATRIX-CHAIN-ORDER, the procedure MEMOIZED-MATRIX-CHAIN runs in $O(n^3)$ time. Line 5 of MEMOIZED-MATRIX-CHAIN executes $\Theta(n^2)$ times. We can categorize the calls of LOOKUP-CHAIN into two types:

1. calls in which $m[i, j] = \infty$, so that lines 3–9 execute, and
2. calls in which $m[i, j] < \infty$, so that LOOKUP-CHAIN simply returns in line 2.

Dependency (order subPb comp)

⟺ recurse stack

if m[ij] already computed not make recursive call

Memoization

- C = still a table

C[] cache computed values

- never compute again

- never recurse on computed value

only recurse on non-computed C[] cell THAT ARE NEEDED

Memoization speeds up when it does not solve all subPB in dependency table

PB	Top Down Memoization Must solve all subPB anyway?
	No = not nec. all probs
Rod Cut	
Coin Change	
CheckBoard	
0/1 Knapsack	No
Matrix Chain	Yes
LCS	
Optimal BST	

YES

Exercise: fill this table

LCS = longest common subsequence

last occ(z_k)

$X = x_1 \ x_2 \ \ldots \ x_n$

last occ(z_k)

$X_i = X[1:i]$ prefixes based

$Y = y_1 \ y_2 \ \ldots \ y_m$

$Y_j = Y[1:j]$ given index

common subsequence $Z = z_1 \ z_2 \ \ldots \ z_k$

max k

Z subseq of X

Z subseq of Y

1) OPTSOL structure $= Z \Rightarrow Z_{k-1} = Z[1:k-1]$ opt solution to some $X[1:?]$, $Y[1:?]$

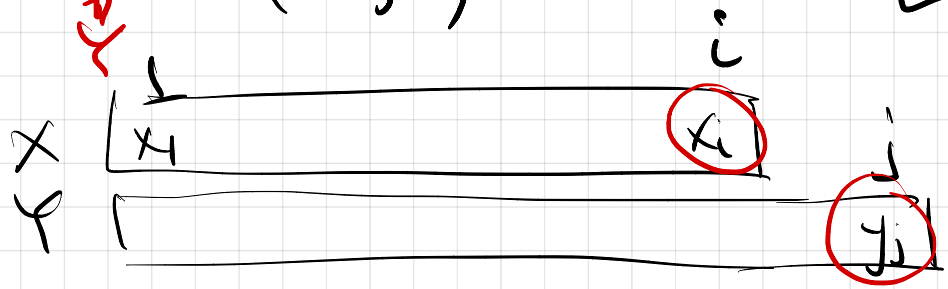
prefixes up to z_k value

assume

z_k occurs in X and Y

2A $C[i,j] =$ max (length) of common subseq / LCS $\{X[1:i],\ Y[1:j]\}$

$i \to X$, $j \to Y$



$=$

Z_k is last found $\iff x_i = y_j$: $C[i-1, j-1] + 1$

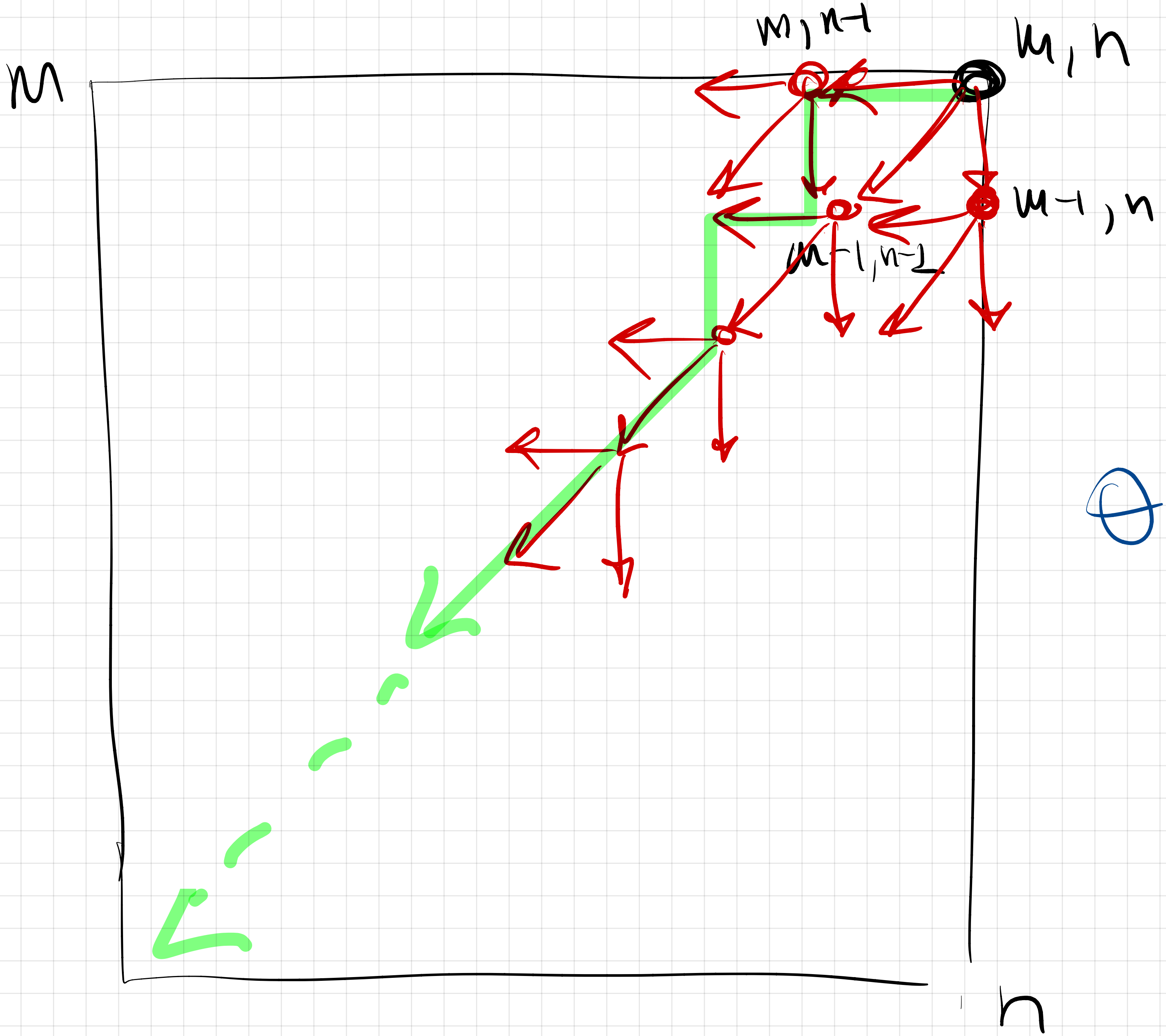
$x_i \neq y_j$ → cut x_i / cut y_j: $\max \begin{cases} C[i-1, j] \\ C[i, j-1] \end{cases}$

$\theta(1)$

2B

$S[i,j] = \{$ which move is max-obj

" $\leftarrow$ " " $\swarrow$ " " $\downarrow$ "

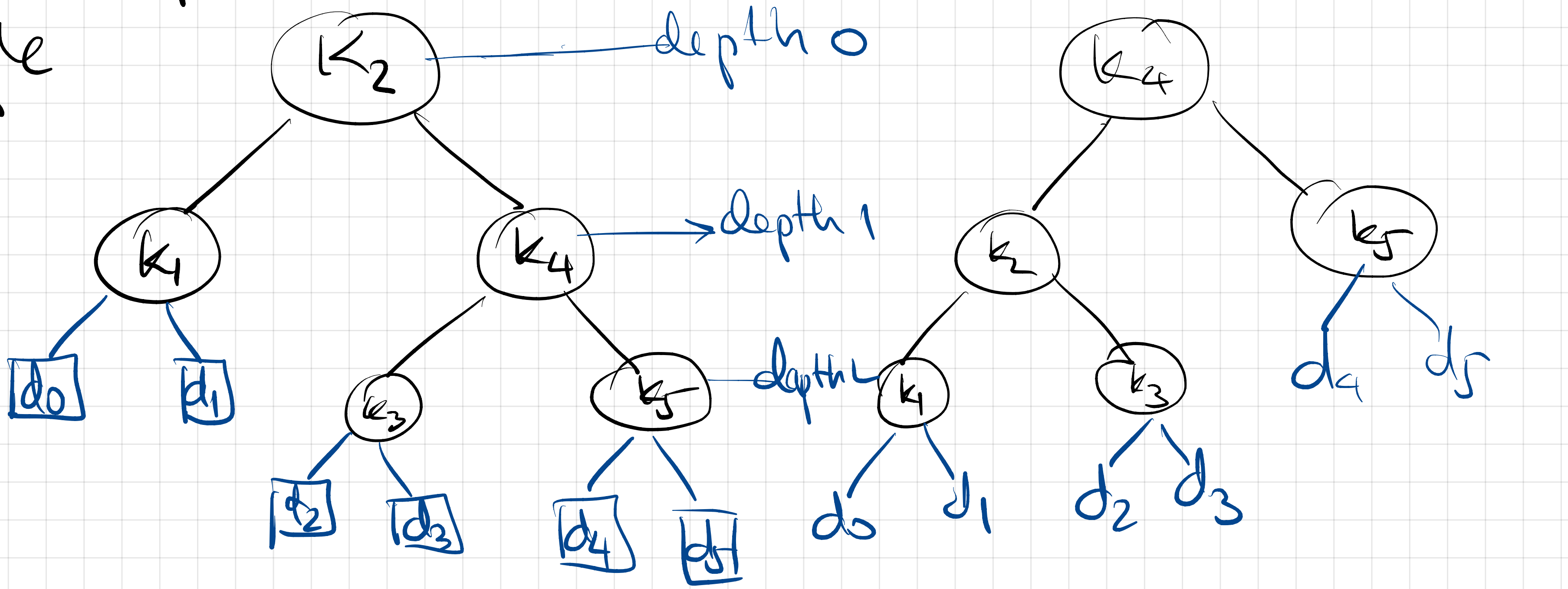


$\Theta(mn)$

LECTURE 8 Optimal BST

ordered values $k_1 \le k_2 \le \dots \le$

example BST



Search probability: $Pr(k_i) = p_i$ not uniform

$Pr(d_i) = q_i$

d_i = searches for values $k_i < val < k_{i+1}$

$\sum p_i + \sum q_i = 1$

depth = level

OPTIMALITY: min expected search cost

$$\sum_{i=1}^{n} [depth(k_i)+1]\, p_i + \sum_{i=0}^{n} [depth(d_i)+1] \cdot q_i$$

Step 1 OPTSOL given

$k_1 < \dots <$ k_r $< \dots k_n$

Left subprb Right subpb.

PbL 0.6 sum to $1 - p_r$ not 1 0.4 PbR

$\sum p_{1:r-1}$ all probs for
$+ \sum q_{0:r-1}$ Left side

$k[1:r-1]$
$1 \le k \le r-1$
$d[0:r-1]$

$k[r+1:n]$
$d[r:n]$

$p_1, \dots p_{r-1}$
$q_0 \dots q_{r-1}$
example 0.6 = PbL

PbR 0.4 = 1 - PbL
$p_{r+1} \dots p_n$
$q_r \dots q_n$

Search: $0.6 \cdot 1^{\text{root}}$ +
best way to search $(k[1:r-1])$

Search: $0.4 \cdot 1^{\text{root}}$
+ best way to search $k[r+1:n]$

② $C[i,j]$ = best cost tree for keys $k_i, k_{i+1}, k_r, \ldots, k_j$

$E[\boxed{\text{search time}}]$ //

find r (search) $= ??\ C[i, r-1] + ??\ C[r+1, j]$

Left subtree

prob PbL   prob PbR

Search = MIN

time $(k_r = \text{root})$

search in Left side   PbL 1. step   search Right   Pbr 1step

$\boxed{1}\, p_r + C[i, r-1] + W[i, r-1] + C[r+1, j] + W[r+1, j]$

Left   Right

looking for root k_r

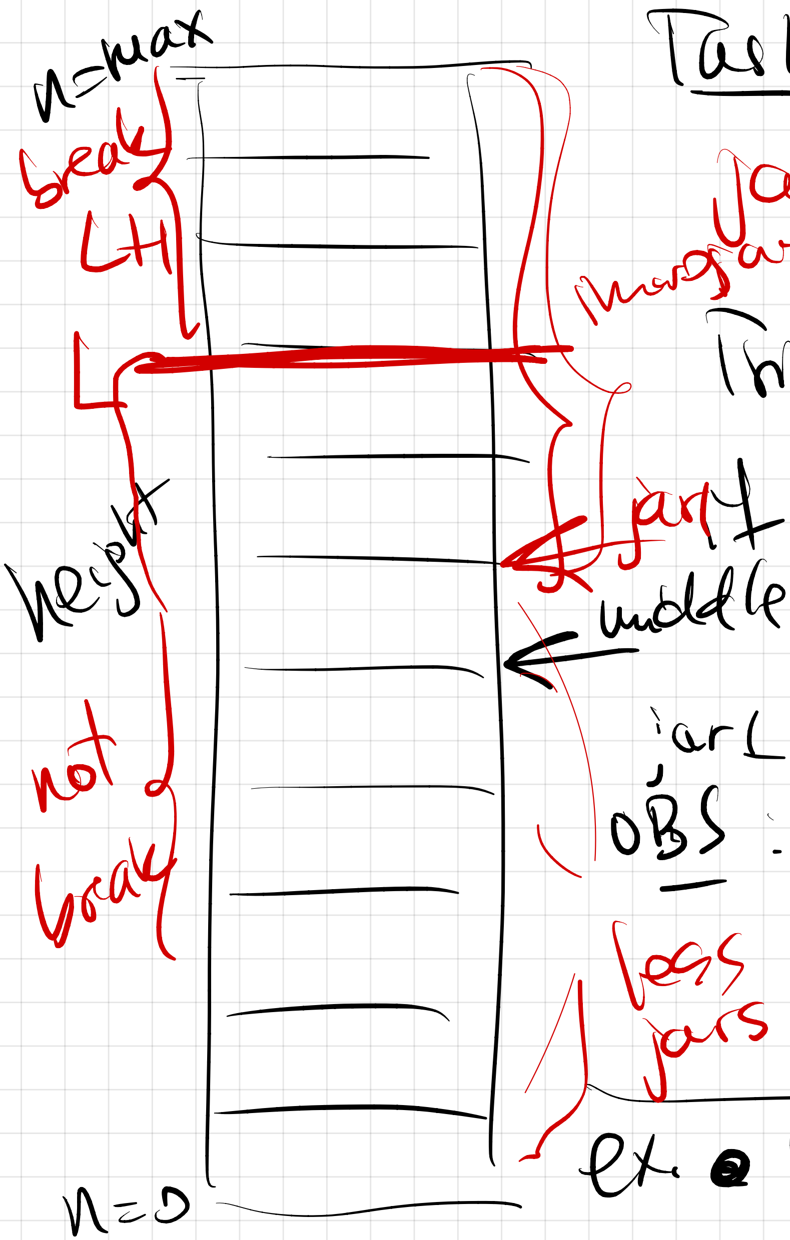
$$W[i,j] = \sum_{t=i} p_t + \sum_{t=i-1} q_t$$

all probas in $[i:j]$

sum of probas left side

Jars on Ladder n steps k jars

<u>Task</u>: find highest level L where jars don't break (they break at $L+1$)



Trial jar at level l

$l \leq L \Rightarrow$ doesn't break $\Rightarrow$ can reuse it

$l \geq L+1 \Rightarrow$ jar breaks cannot reuse it

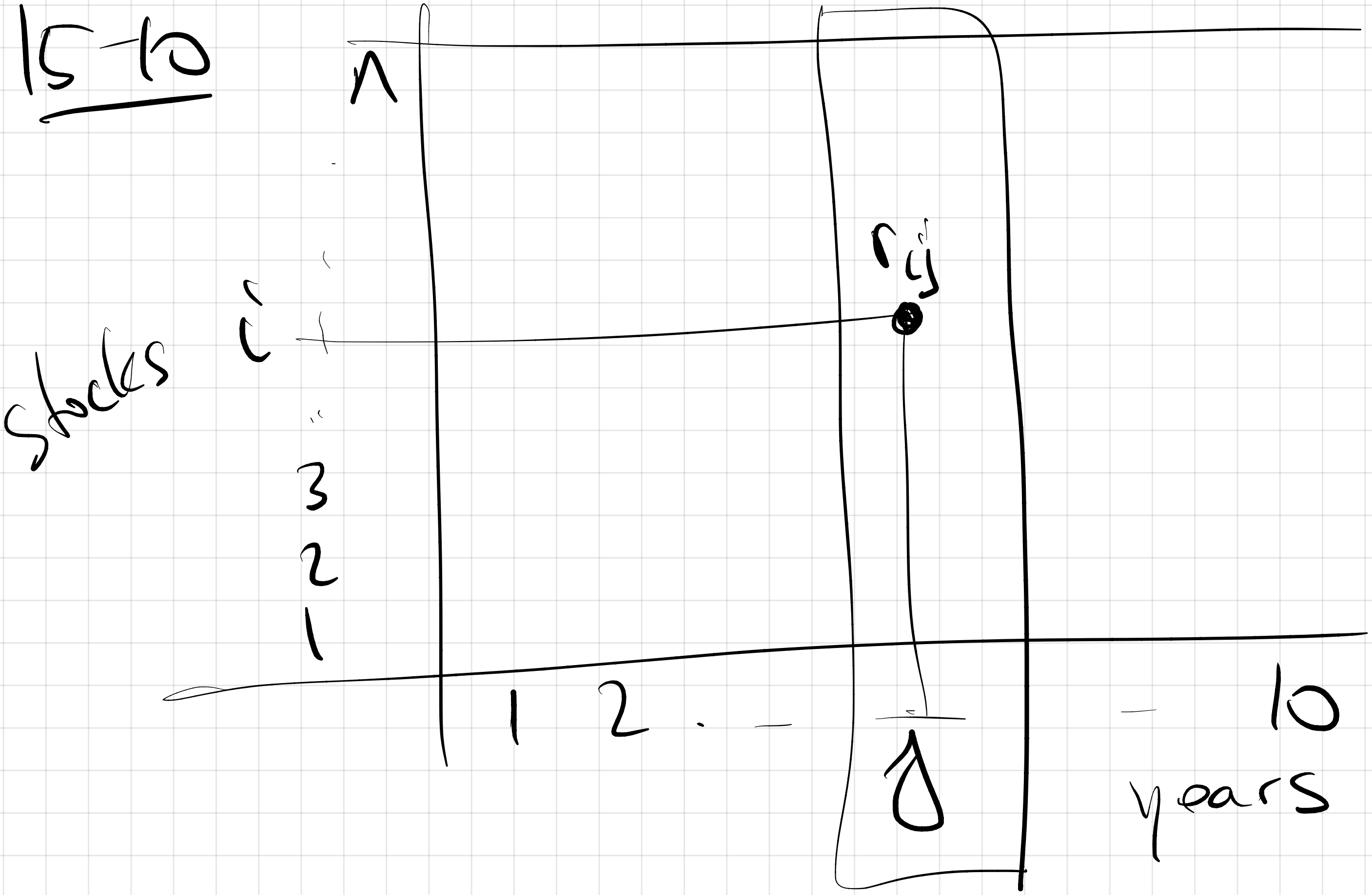
<u>OBS</u>: minimum # trials in worst case

You must 100% find L

ex. • $k=1 \Rightarrow$ bottom up on stair $\Theta(n)$

• $k > \log n \Rightarrow$ binary search

15-10



r_{ij} = return % of stock i year j

put $\$N \rightarrow \$N + \$N\, r_{ij}$

$\$N \rightarrow (1+r_{ij})\$N$

a) pick best stock that year

b) OPT SOL

find split based on fee?

c) pseudocode + RT

d) max allocation per stock $15000

OPTSOL charact?

prev

NOT WORKING

still DP?

Speculation: - DP structure works

- C[] table of subph has to be bounded?

Painters (fence problem) fence = boards sequence

b_i = length of board i

k painters same speed paint done / unit

painter 1: b_1 - - - b_{p_2} → time 1 = $\sum$ length(boards) = $\sum_{i=1}^{p_2} b_i$ (first painter)

2: b_{p_2+1} - - - b_{p_3}

$k-1$: $b_{p_{k-1}+1}$ - - - b_{p_k}

k: b_{p_k+1} - - - b_n

Task: partition boards (find p_2, p_3 - - - p_k) such that the longest job is minimum

(scenario: painters work in parallel ⇒ → time to finish = longest job/partition)

$C[j, k] =$ best (max time in parallel) to paint boards $1:j$ with painters $1, 2, \dots k$

$1 \le j \le n$

j: board index

k: #painters

$= \text{MIN search for } \left[\text{MAX} \left\{ C[1:p_k, k-1] \; ; \; \sum_{t=p_{k+1}}^{n} b_t \right. \right]$

$k-1 \le p_k \le n-1$??

p_k: last board painted by painter $k-1$

$\Theta(n)$

Total R.T. $\Theta(nk \times n)$

search in $\Theta(\log n)$ time $\longrightarrow \Theta(n \, k \log n)$

exercise

$\Delta_{\text{candidates } p_k} ? = C[p_k, k-1] - \left(\sum_{t=p_{k+1}}^{n} b_t \right)$ last job

$$\sum_{t=p_k+1}^{n} b_t = \underset{\sum_{t=1}^{n} b_t}{\overset{\text{cumul}}{B_n}} - \underset{\sum_{t=1}^{p_k} b_t}{\overset{\text{cumul}}{B_{p_k}}}$$

NEXT: Selected previous midterm questions

Set of values $A = \{a_1, a_2, \ldots \ldots \ldots a_n\}$

Create $B \cap C = \emptyset$ $B \cup C = A$

partition

$$\sum_{a_i \in B} a_i \underset{\text{as close as possible}}{\simeq} \sum_{a_j \in C} a_j$$

Subset sum

want B of elements $\sum_{a_i \in B} a_i \leq$ as close as possible $\cancel{Z}$ = given $\cancel{Z} = \frac{\sum a_i}{2}$

Knapsack a_i = values = weights

*** Same problem $A = \{a_1 \ldots a_n\}$

Want 3-set partition $B \cup C \cup D = A$

any $B \cap C$ etc $= \emptyset$

minimize $\left| \begin{matrix}\text{max-size} \\ \text{part}\end{matrix} - \begin{matrix}\text{min size} \\ \text{part}\end{matrix} \right|$

*** Same pb $A = B \cup C$ $B \cap C = \emptyset$

want balance $\sum_B a_i \simeq \sum_C a_j$

$|B| = |C| = \frac{n}{2}$ n=even

hint $C[\_,\_,\_]$ 3 args

$n \in \mathbb{Z}$ given   denominations $\{d_1, d_2 \ldots d_k\}$

want to list all possible ways of partition

n as sum(denom).

$n = 30$ denom $= \{1, 5, 10, 25\}$

$25 + 5$

$25 + 1 + 1 + 1 + 1 + 1$

$10 + 10 + 10$

$10 + 10 + 5 + 5$

$10 + 10 + 5 + 1 + 1 + 1 + 1 + 1$

⋮

version   denom $= \{1 : k\}$

version: print #possib, not list them

* denom = $\mathbb{N}$ actual used $d_i \leq n$

Partition Problem: decompose n into a sum (additive) of smaller integers

---

m nuts $n_1, n_2 \ldots \ldots n_m$

m bolts $b_1 \; b_2 \ldots \ldots b_m$

pairwise match

Match m nuts with m bolts pb

match them

cannot sort, only try n vs b

$n > b$

$n = b$

$n < b$

(cannot compare 2 nuts)

• use n_1 as pivot for bolts

use b_1 to partition nuts

smaller s b_1 bigger bolts

nut b_1

n_2

$n_2 < s_1$ | $n_2 > s_1$

partition left splits by n_2

$A = [a_1 \ldots a_n]$ — Maintain I, S sorted indices

S = perumtation of indices $[1 \ldots n]$

$A[S] = A[s_1]\ A[s_2] \ldots A[s_n]$ sorted

I = inverse perm $\quad S[I] = [1, 2, \ldots, n]$

---

a value gets changed $\quad a_i$ = new value

fix S, I $\boxed{\text{efficiently}}$ $\quad \Theta(\Delta \text{ranks for } a_i)$

$r_{old} - r_{new}$

want example

$T(n) >$ any polynomial $\quad T(n) = \omega(n^c) \ \forall c$

$T(n) <$ any exponentia $\quad T(n) = o(a^n) \ \forall a$

Tournament: $A = [a_1 \ldots \ldots a_n]$

majority elem occurs $\geq \frac{n}{2} + 1$ times.

Find majority elem if it exists

- a_i are not sortable, but comparable $a_i == a_j$?