Dynamic Programming (part 1) Lecture 6

PB

⇓ Divide & Conquer

OPTSOL = SOL1 | SOL2

SPB1   decision (split)   SPB2

⇒ NO D&C

- Brute Force (try all possib)
- approximation

⇓ GREEDY

- Divide/split/decide
  ⇒ 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)

DP 1 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

b		$n-b$		
ℓ_1	ℓ_1	ℓ_2	ℓ_3 ...	ℓ_k
v_1	v_1	v_2	v_3	v_k

lengths being cut.

$\Downarrow$ OPTSOL for that length b

$\Downarrow$ OPT for length $n-b$

so D&C

works by exchange arg

② A DP

objective recurrence input $= n$

search ~~select~~ first length (k)

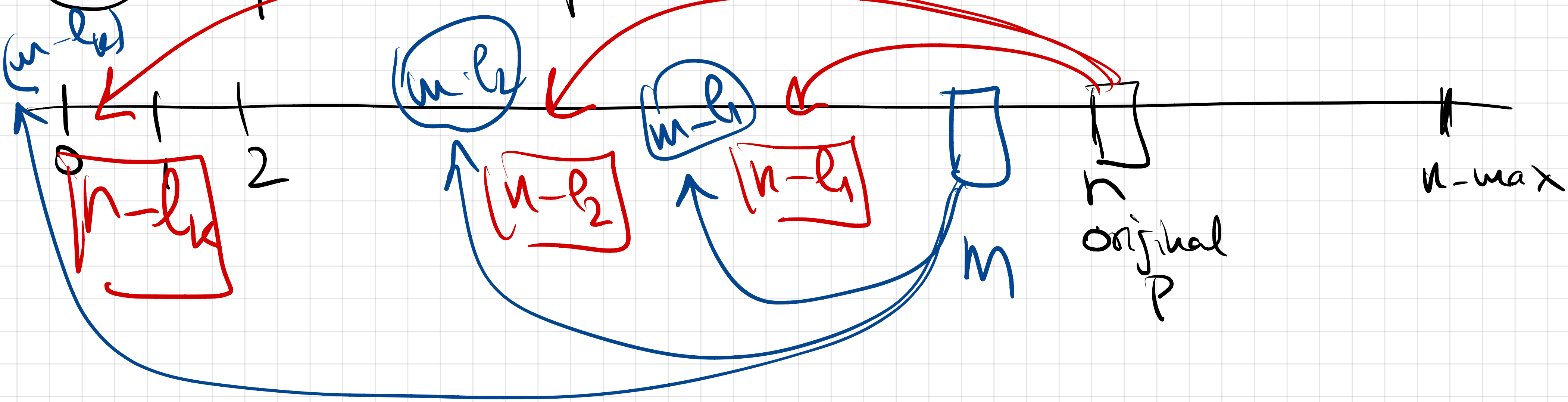
$$C[n] = \max_{1 \le k \le n} \left\{ v_k + C[n-\ell_k] \right\}$$

subpb

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

$|< \#$ of lengths in the table

for $k = 1 :$ all lengths possible

if $n - \ell_k < 0$ skip

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

best $= v_k + C[n-\ell_k]$

$\Theta(nk)$

best_k = k

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

④ Trace Solution + add in pseudocode

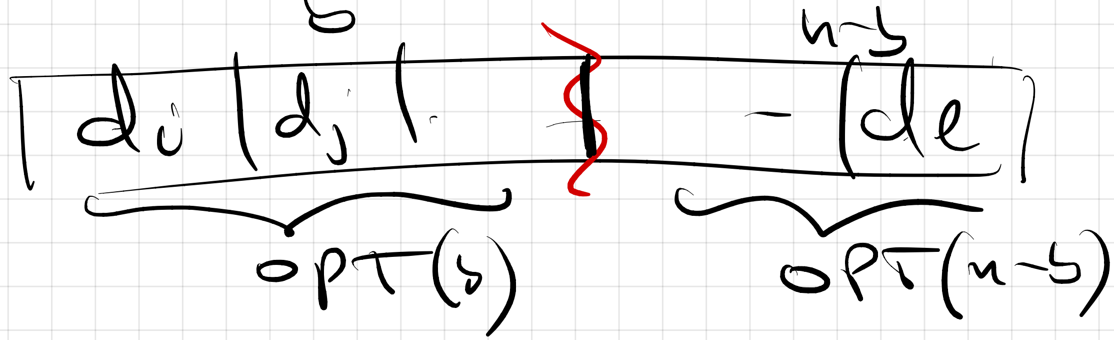
s[same input as c] = the choice made

Print Solution (n)
    output k = S(n) ⇒ ℓ_k, v_k
    PrintSolution ($n - \ell_k$)
    unless $n \leq 0$

(DP2) Given change to n cents with <u>min</u> #coins

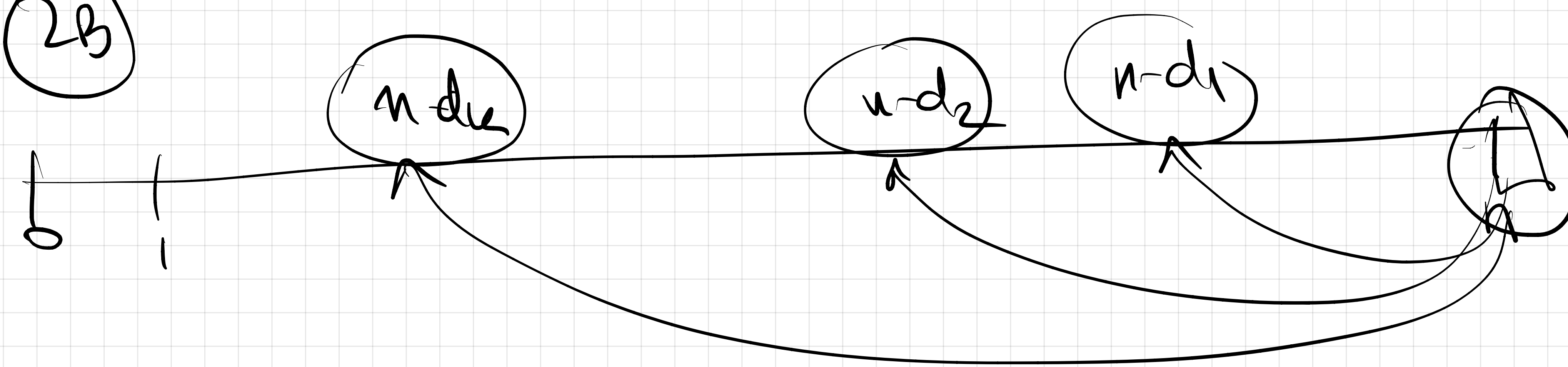
Coin denom $= \{ d_1 = 1, d_2, d_3, \dots, d_k \}$ ∞ amount coins.

(1) already done



(2) $C[n] = \min\limits_{k} \left\{ \text{search for first coin } k \quad \underbrace{1}_{\text{not added to OBJ}} + C[\underbrace{n-d_k}_{\text{subpb}}] \right\}$

(2B)



③ Bottom up comp L → 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

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

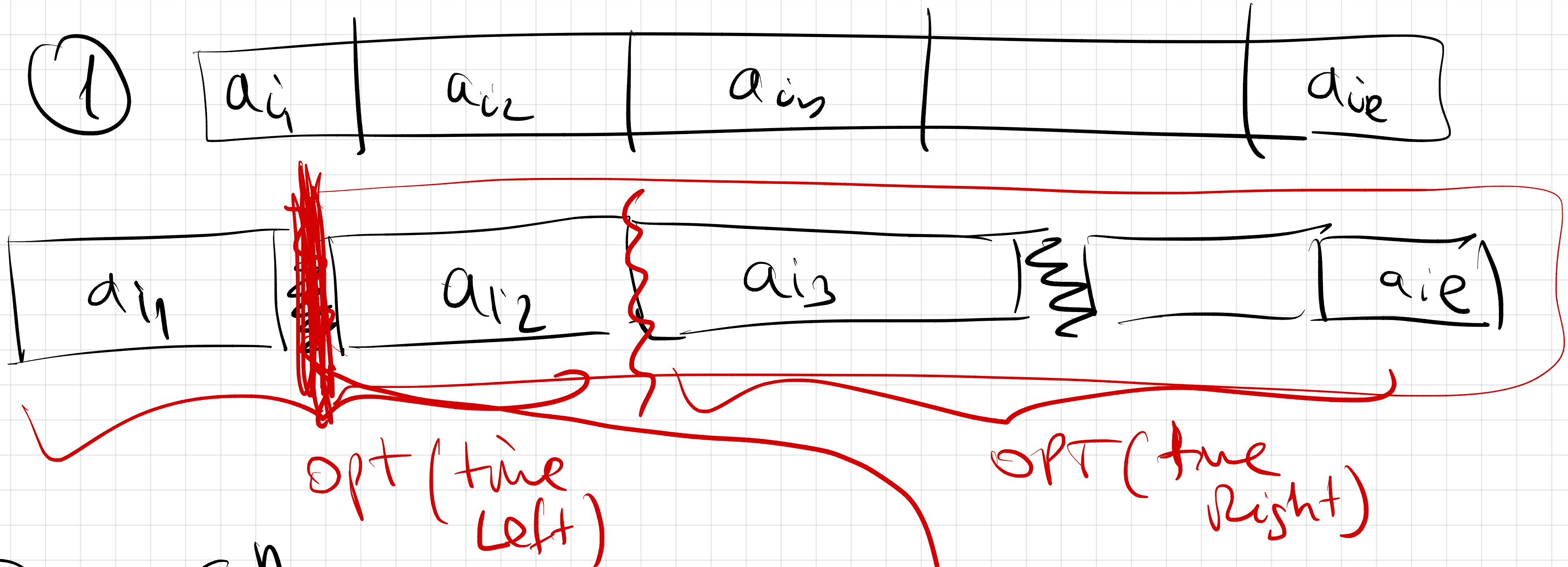
④ Trace

Exercise Activities Selection

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

OBJ: max total time

(1)

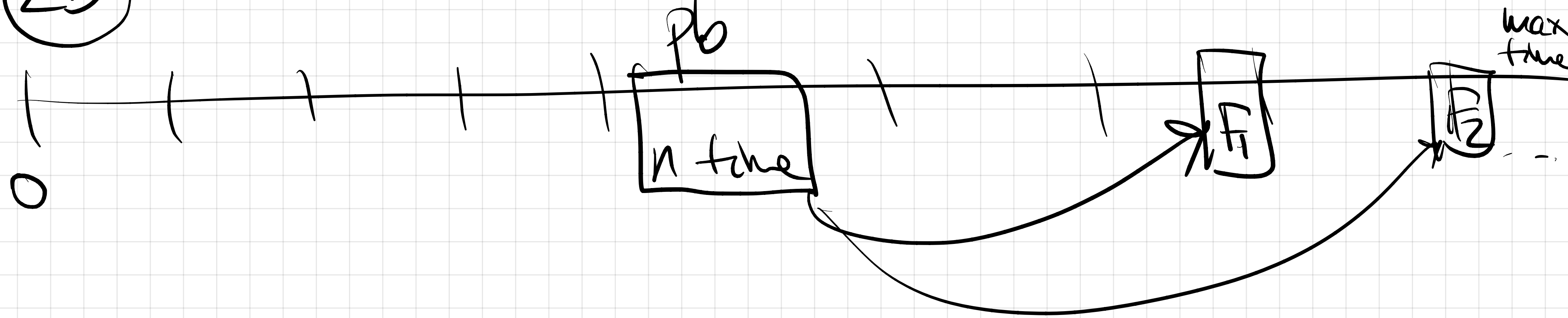


(2A) $C[\overset{n}{\text{start time}}] = \max_k \left\{ \underbrace{\Delta k}_{F_k - S_k} + C[F_k] \right\}$

add to obj

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

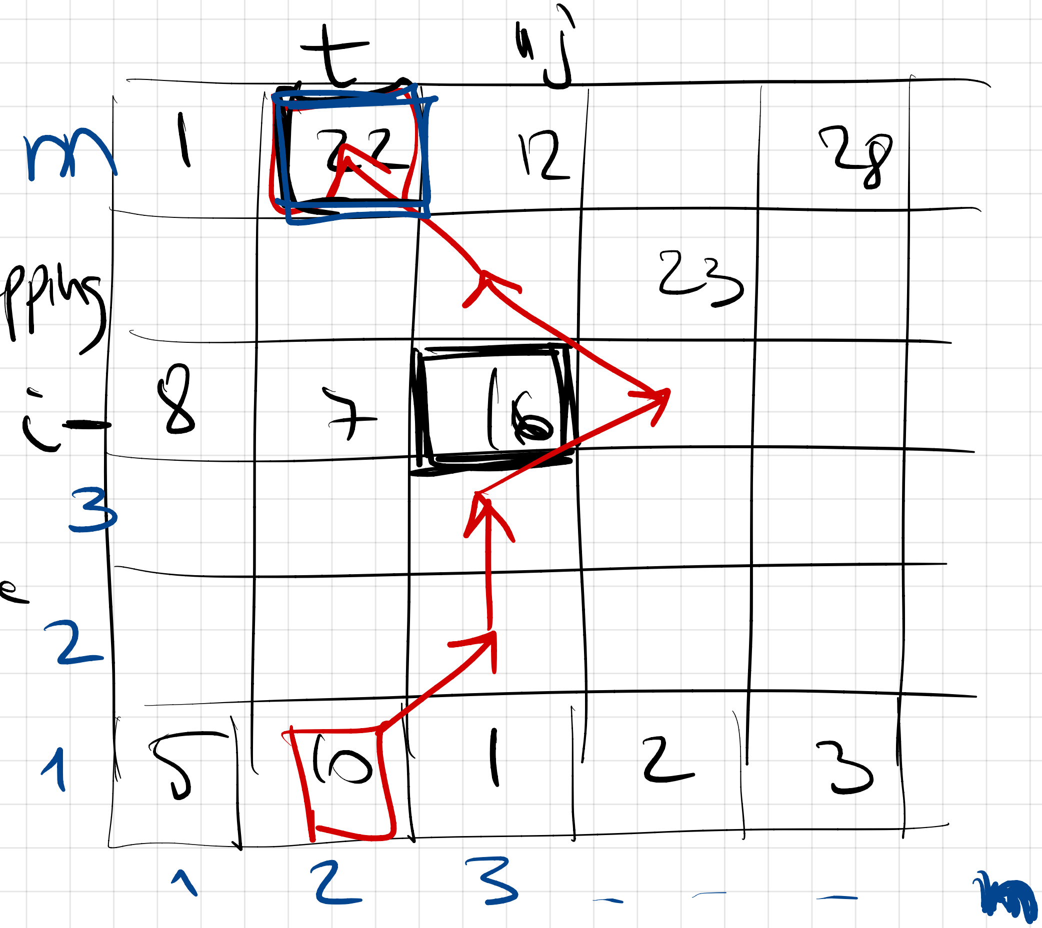
2B



DP4 CheckBoard

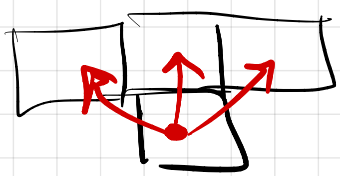
given.

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



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

3 valid moves



m rows, n colums

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

rows to cell i,j

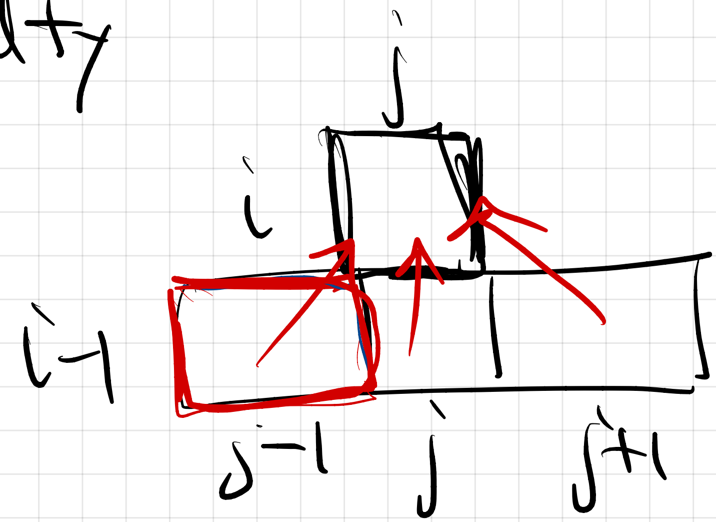
anywhere row m down to cell (i,j)

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

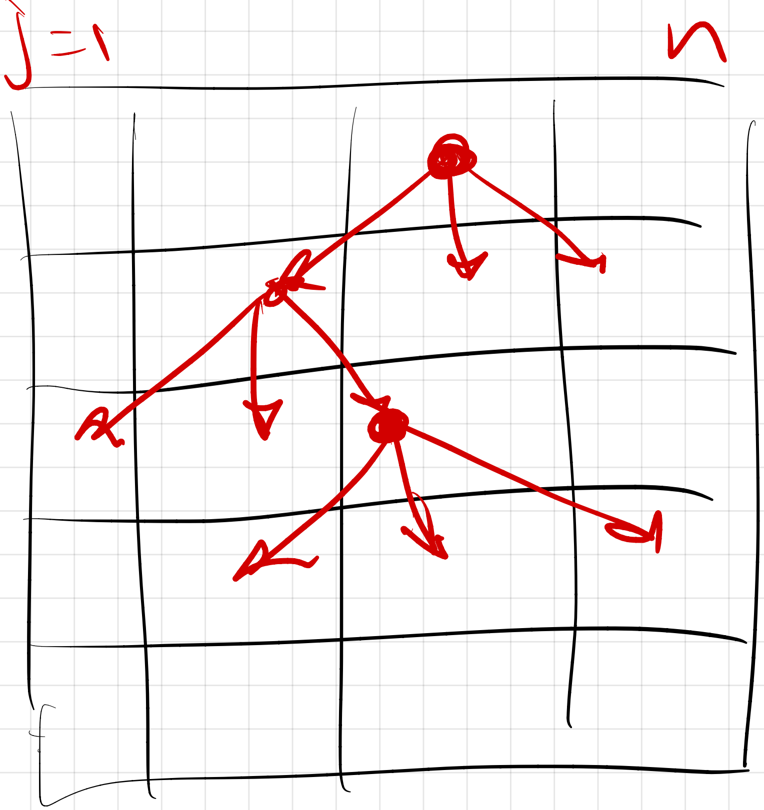
search last move from $(i-1, j-1)$

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

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 $j = 1:n$
  $k = \text{argmin}_k \{ c[i-1, j-1], c[i-1, j], c[i-1, j+1] \}$
  $c[i,j] = P[i,j] + c[i-1, k]$
  penalty
  $s[i,j] = k$

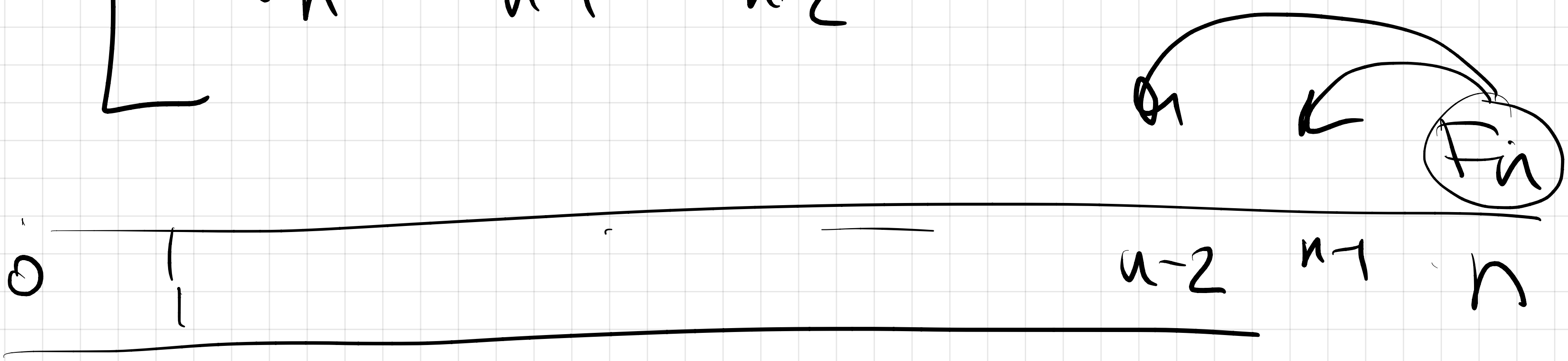
Simple DPs look like not DP

$F_0 = 0$ $F_1 = 1$

for $n = 2 : \max n$

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

$F_n = a[n]$



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

choose subset of size k out of n elements

Sum rule

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

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

- 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 $n = 1$: 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, weights)

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

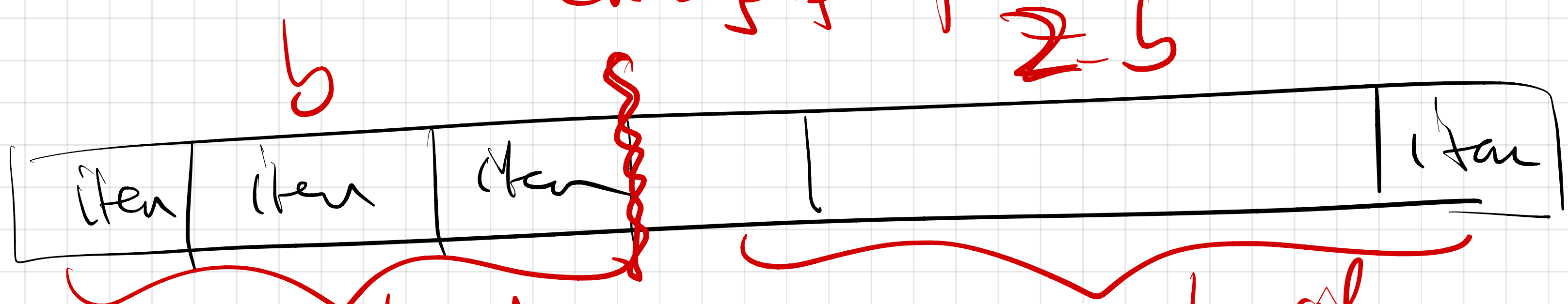
$\sum$ weights $\le Z$

- 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 $C[b;$ all items$]$

optimal $C[Z-b;$ all items not on left$]$

allitems = set

② tentative ? wishful thinking

A

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

(B)