Dynamic Programming

Divide & conquer — decision / split

PB ⟶ SUBPB → optsol (subpb)

OPTSOL = glue (optsol (subPB))

(1) is it possible for OPTSOL to be obtained via D&C?
Characterize OPTSOL

DP recipe (writing) → required

(2A) recurrence of objective C[pb input] = rec formula (C[sub pos])
obj value opt

(2B) visual table (PB → SPB) dependencies

(3) bottom up computation / pseudocode : solve all? subpb in the right order

(4) Trace solution / print S[pb] if necessary

DP1 Rod cutting

n = Rod length $n \in \mathbb{Z}^+$

price table

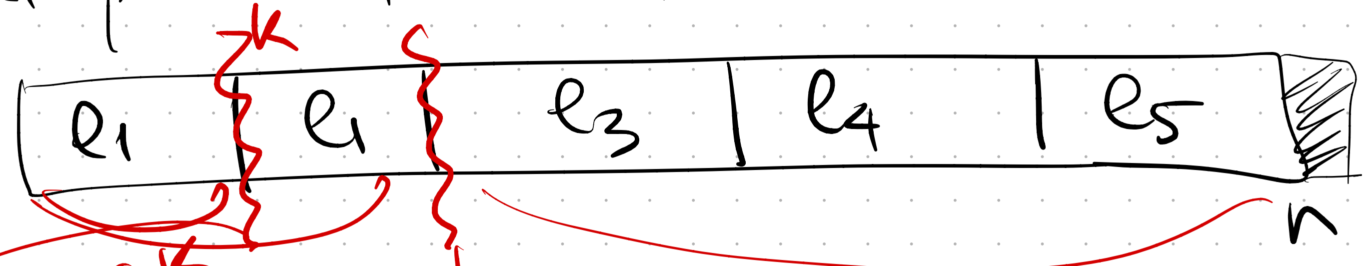
length	1	2	3	4	5	R6
price value	1	2	4.5	6.3	8	7
qual	1	1	1.5	1.59	1.6	7/6

$n = 11$ Greedy: 5+5+1

OPTS: 4+4+3

Task: cut n into pieces to max total value.

① OPTSOL given



OPT SOL ($n = e_1 + e_1$)

practical cut

any DOC cut

OPTSOL (n = rest) orig $n - e_1 - e_1$

②A $C[n] =$ Find the first cut at length k $\quad k \in \{1, 2, \ldots R\}$

pb input

max value for this input

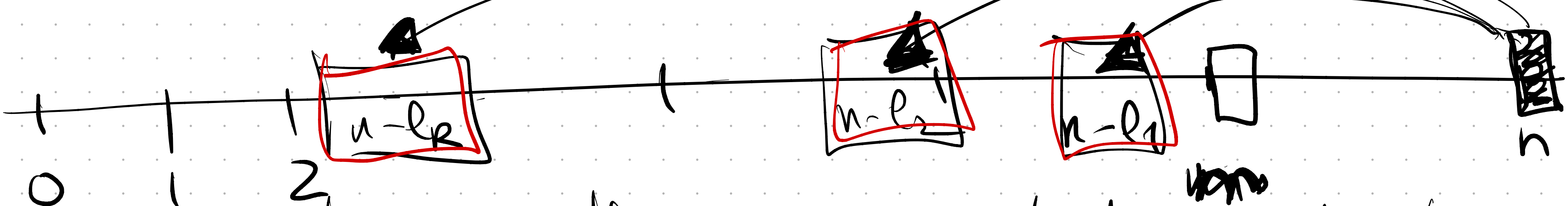
$$\max_{\text{all possible } k} \{ val_k + C[n-k] \}$$

$\Theta(k)$

val of first piece

best value of remain rod.

2B visual PB → SPB dependency



need to have those k SPB solved already by the time we solve PB(n)

bottom up computation

- non recursive
- solve all ? subproblems in what order ? see 2B
- store results in table
- every pb solved only once

③ bottom up computation order left → right

C[0] = 0 // C[ ] = array of n values

given n

for m = 1 : n //solve all subpb

//search for k   best = 0 ; best_k = −1

for k = 1 : all length in table   → table / price

if $(m - \ell_k) < 0$ skip

if $(V_k + C[m - \ell_k] > best)$ → subpb already solved

best = $V_k + C[m - \ell_k]$

best_k = k

C[m] = best

S[m] = best_k : first cut at end of length m

$\Theta(nk)$

output: C[n] = best value for rod length n

obj

(4)

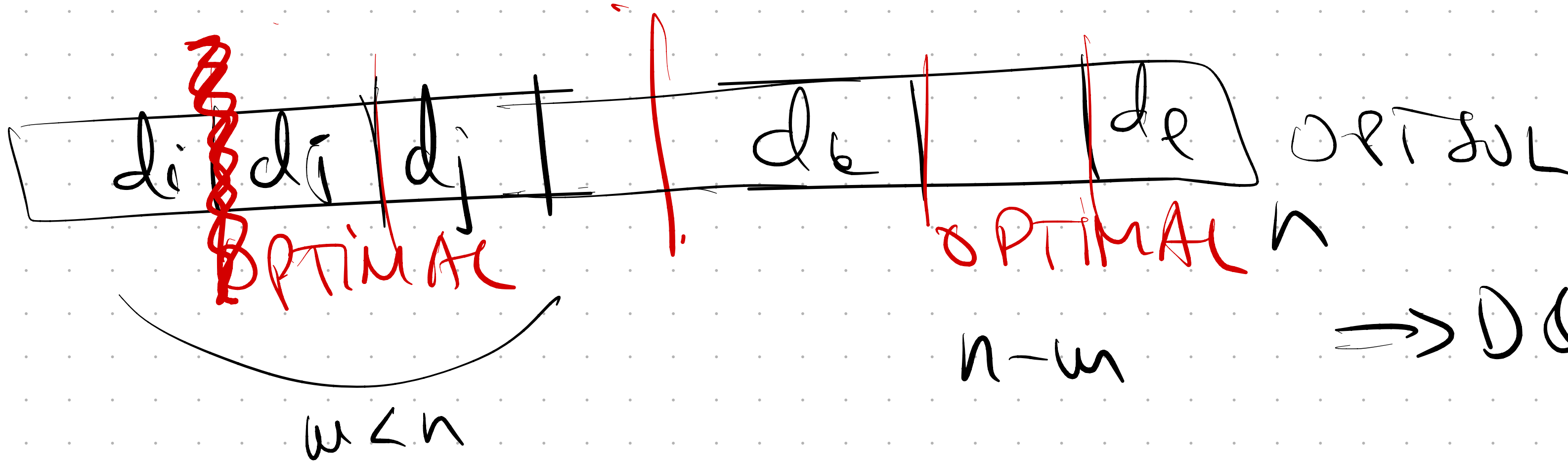
```
PrintSol(n)
    output S(n)
    PrintSol(n - S[n])
```

```
while n > 0
    output S(n)
    n = n - S(n)
```

DP2 Coin change $D = \{d_1, d_2, \ldots, d_k\}$, $d_1 = 1$

Task: min # of coins for exact change (n cents)

(1)



(2A) $C[n]$ = min # of coin sum up to n

search for first coin k

$$= \min_{\substack{\text{all possible } k \\ \text{that can}}} \left\{ 1 + C[n-d_k] \right\}$$

(2B)

③ bottom up comp (n_max)

C[0] = 0

$\Theta(nk)$

for n = 1 : n_max

best = ∞   best_k = -1

for k = 1 : largest_denom ≤ n

if (1 + C[n − d_k] < best) then

best = 1 + C[n − d_k]

best_k = k

C[n] = best

S[n] = best_k

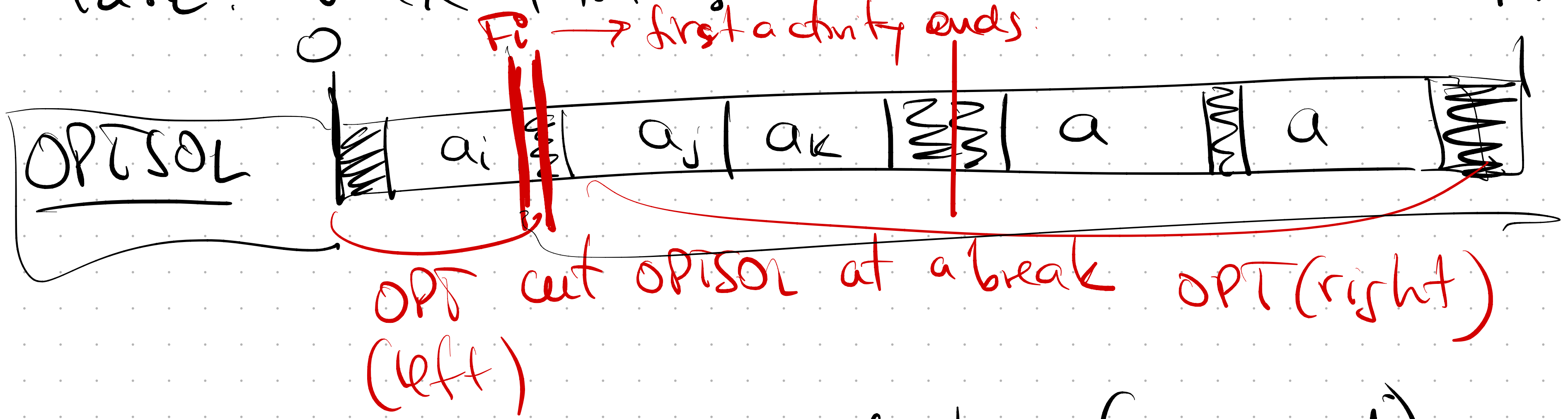
answer (orig) : C[n_max]

4. Print solution (exercise)

DP3 Activity Selection $(S_1, F_1)(S_2, F_2) \ldots (S_n, F_n)$

Task: max total scheduled time   MA

①



② $C[n]$ = max scheduled time of time (n : end)

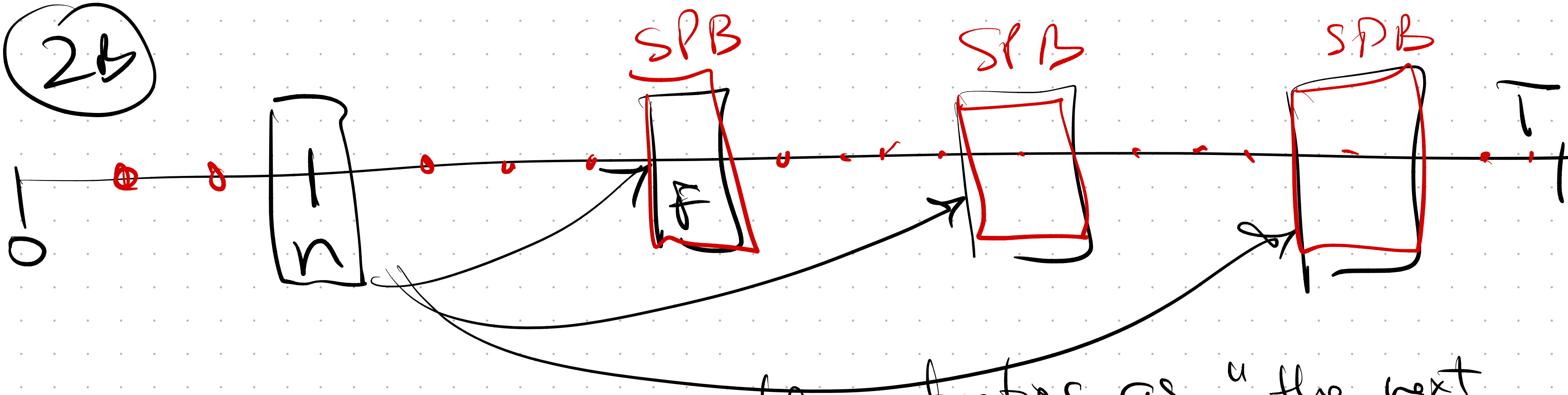
activity K = search for cut of first activity = $F_{(first)}$

$S_K \geq n$

$$= \max_{\substack{K \\ S_K \geq n}} (F_K - S_K) + C[F_K]$$

$(F_K - S_K)$: added sched time

$C[F_K]$ → best total scheduled time starting at F_K



possible activities as "the next the first to schedule next"

$S_k \geq n$

assume: range of all times (S, F) integers $0 \leq S, F \leq T$

(3) exercise

(4) exercise

$T = \max F$ time [T:T]

First PB to solve $C[T] = 0$

c[99:100] = 0 Second: T-1 ? second largest F

(3) Solve all subpb in C[ ] table

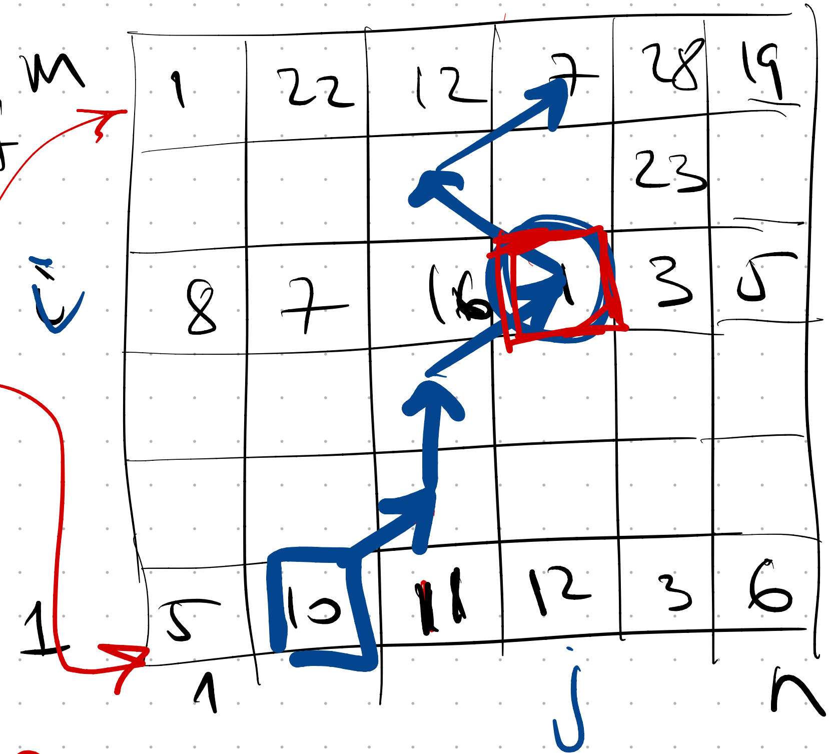
index at F times

$C[i]$ = max scheduled time possible $[F_i : end]$

$F_1 \leq F_2 \leq \dots F_n$

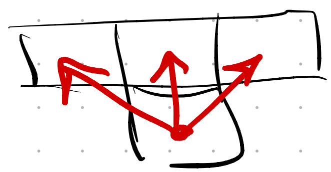
DP4 checkboard

given, $P[i,j] = P_{ij}$ = penalty of stepping on cell (i,j)



Task path from cell ∈ first row to any cell ∈ last row with min total penalty.

- continuos
- going up



possible "moves"

- cylinder column(n+1) = column 1, column(-1) = col n-1

① D&C? OPTSOL = path $(1, j_1) \to (2, j_2) \to \dots \to (m, j_m)$

valid, $j_{k+1} \in \{j_k - 1, j_k, j_k + 1\}$

cell in path (i,j) break there

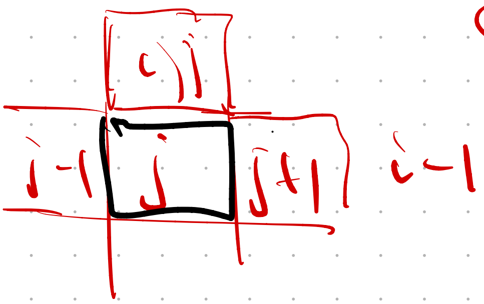
→ path(row 1 → c(i,j)) optimal; path(cell c(i,j) → row m) optimal reformulate PB : Task : find min-total-penalty path (row 1 → cell (n, j))

OPSOL : (1, j1) (2, j2) . . . (n−1, j_{n−1}) / (n−1, j_n) / (n−1, j_n+1) → given cell (n, j_n)

one of 3 moves

decision

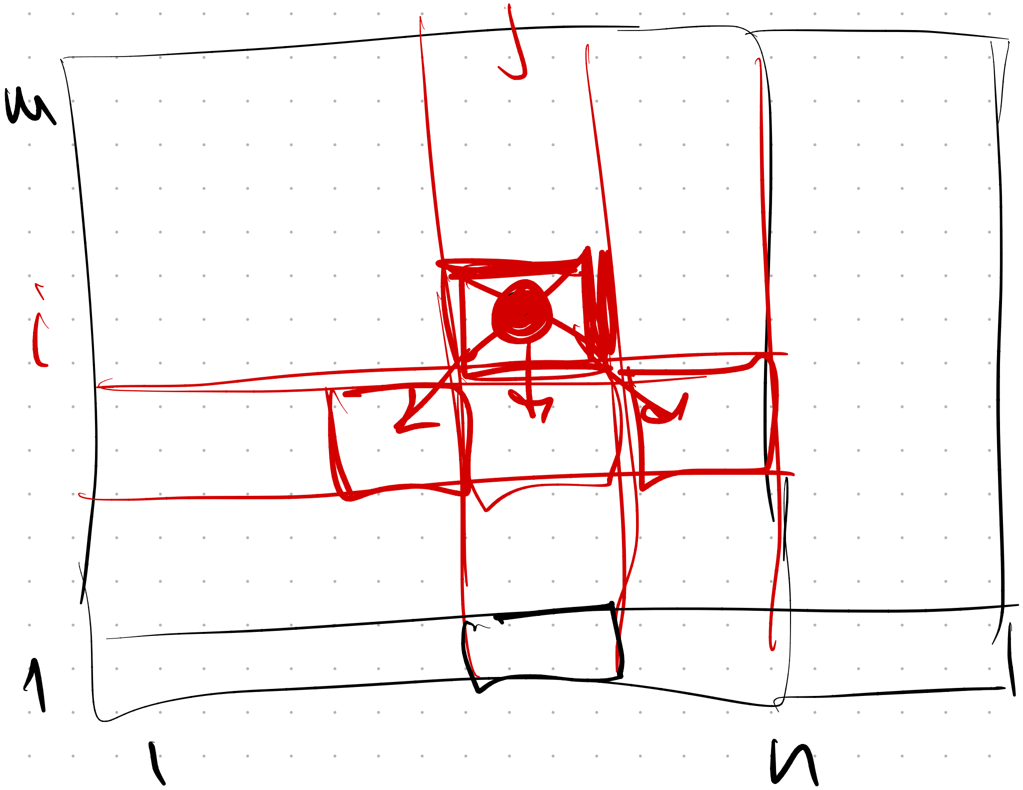
② $C[i,j]$ = min best path from row 1 to cell $[i,j]$



search for cell on row i−1

$$C[i,j] = \min_{k \in \{j-1,\, j,\, j+1\}} \{ C[i-1,k] \} + P[i,j]$$

(2B)



for PB(i,j) i need SPB (i-1,j) (i-1,j+1) (i-1,j-1)

m × n SPB

order: row by row up

each SPB Θ(1)

(3) (incomplete)

first row: C[1,j] = P[1,j]

for r = 2 : m

for col : 1 : n

c[r,col] = P[r,col] + min { c[r-1,col-1], c[r-1,col], c[r-1,col+1] }

(3B) postprocess lastrow

(4)

Simple DP-like pb that dont appear DP

(DP5) Fibonacci #

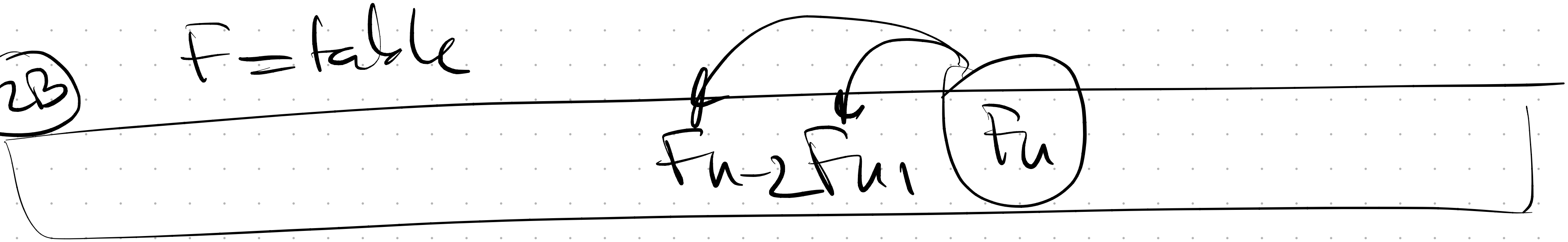
$$F_n = F_{n-1} + F_{n-2}$$

$F_0 = 0, \quad F_1 = 1$

Compute $F(n)$

F = table

(2B)




```
F(0)=0; F(1)=1
for k = 2:n
    F(k) = F(k-2) + F(k-1)
output F(n)
```

DP6 Compute $\binom{n}{k} = C_{n,k} = nCk$

= # of ways to pick k items out of a set on n

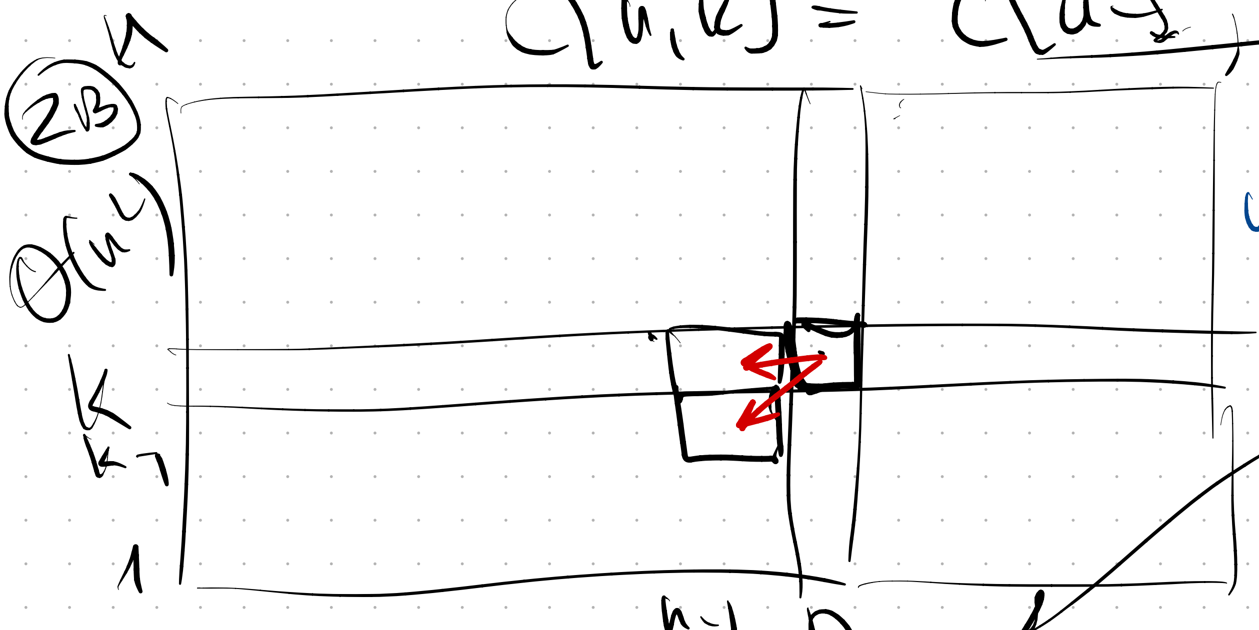
= # subsets of size k of set of size n

$$= \frac{n!}{k!\,(n-k)!} \qquad S = \{1, 2, \dots n\}$$

recurrence 2A $C[n,k] = \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

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

2B



proof $S = \{1, 2, \dots n-1, n\}$
want to choose k of them.
2 disjoint option groups
- include "n" $\binom{n-1}{k-1}$
- do not include "n" $\binom{n-1}{k}$

DP7 Discrete Knapsack

$v_1, v_2, \ldots v_n$ vals
$w_1\ w_2 \ldots w_n$ weigth } integers

Z = knapsack max weight + integer

Task: maximize value in knapsack without breakin Z limit

discret: every item take all of it, or nothing.

(1)

$w_R = Z - w_L$

optimal (w_L, items)   optimal (w_R, items)

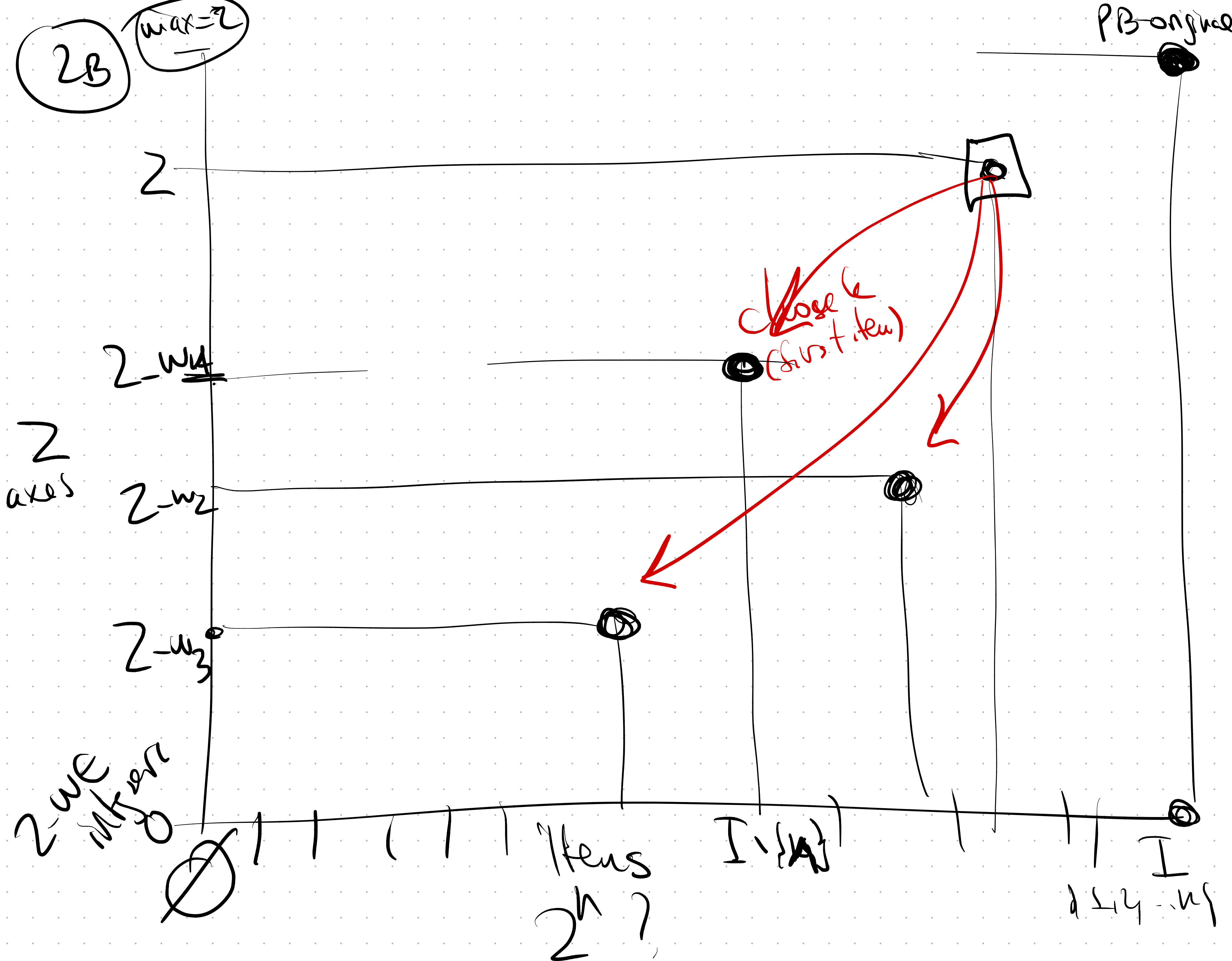
decision item k

knapsack SPB ($Z - w_k$, Items \ {k})

$I = \{1, 2, \ldots n\}$ all items

(2A) $C[Z, I] = \max_k \{ v_k + C[Z - w_k, I \setminus \{k\}] \}$

search for first item k $w_k \le Z$


2B
max=2
PB original
2
2_w1
Z
axes
2_w2
2_w3
0
Items
I

Indexing trick: global order of items (any order)

1, 2, ... n

X axis

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

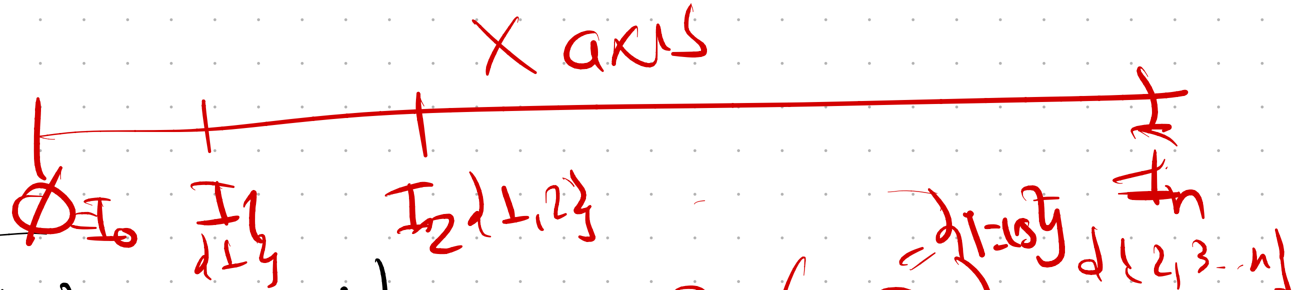
$I_{n-1} = I[1:n-1] = \{1, 2, \ldots n-1\}$

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

$\vdots$

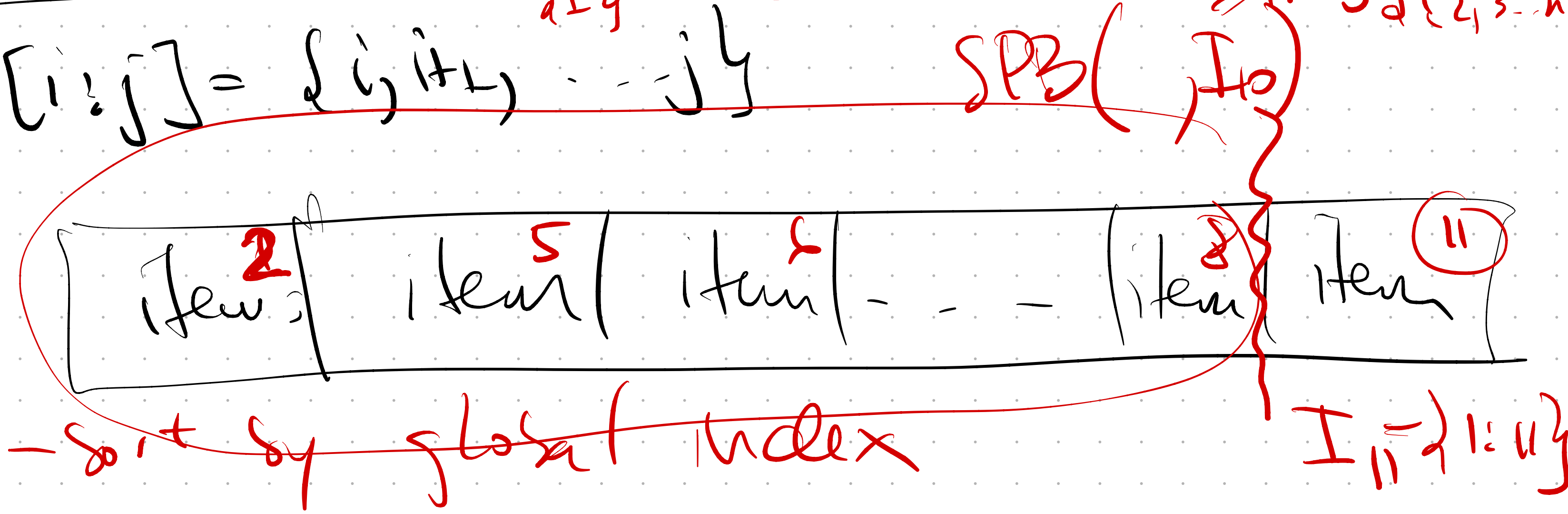
$I_1 = \{1\}$

$I_0 = \emptyset$



chunk $I[i:j] = \{i, i+1, \ldots j\}$

① OPTSOL STRUCT



2A $C[Z, I_n] = \max \begin{cases} \text{2 possib.:} \\ \text{- pick item } n \to v_n + C[Z - w_n, I_{n-1}] \\ \text{- don't pick item } n \to 0 + C[Z, I_{n-1}] \end{cases}$

$\{1:n\}$

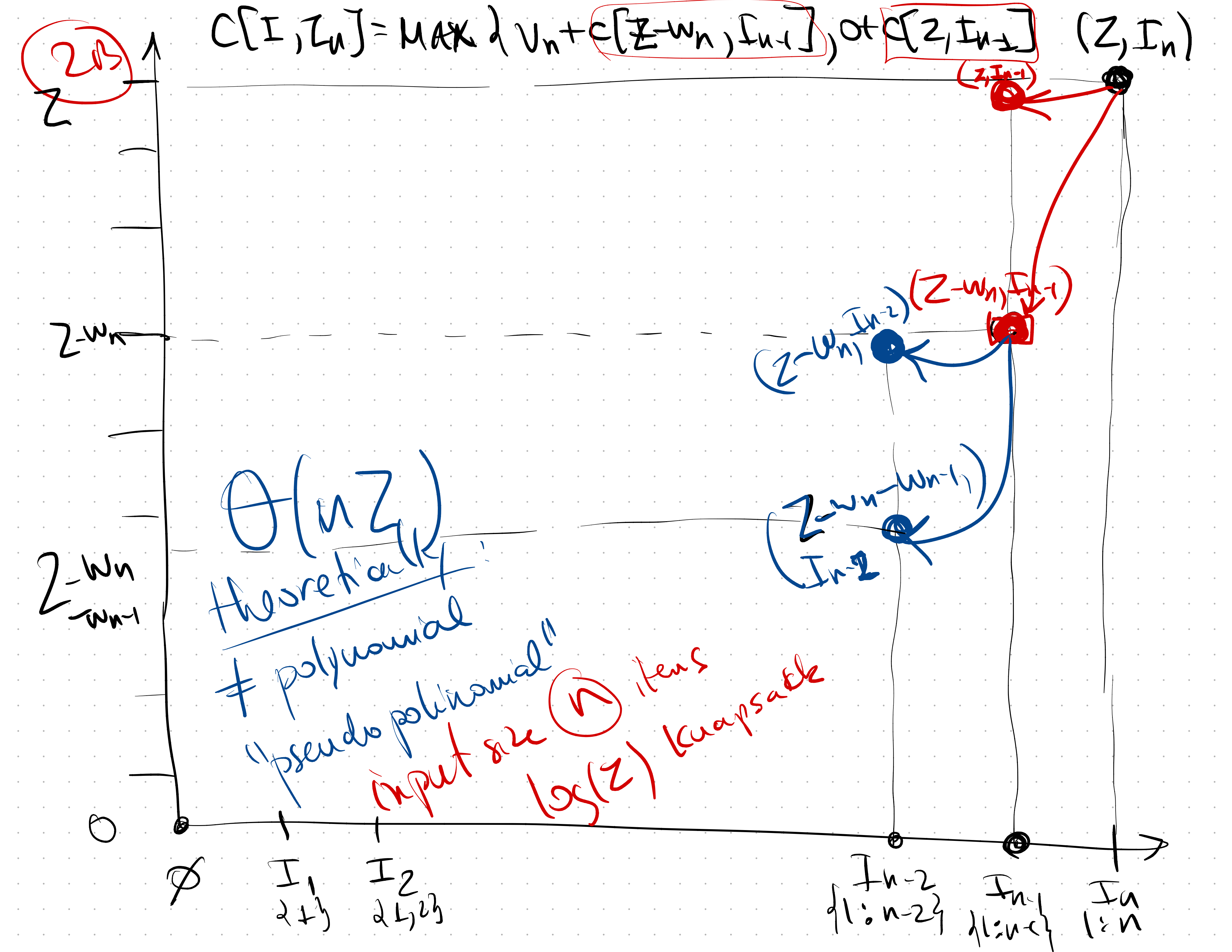
max value obtainable with items $\{1, 2, \ldots n\}$

---

example : $I = \{1:10\}$

OPTSOL : $\{2, 4, 6, 7, 9\}$

- $\max\{v_{10} + C[Z - w_{10}, I_9],\ 0 + C[Z, I_9]\}$
- $\max\{v_9 + C[Z - w_9, I_8],\ 0 + C[Z, I_8]\}$
- $\max\{v_8 + C[Z - w_9 - w_8, I_7],\ 0 + C[Z - w_9, I_7]\}$
- $\max\{v_7 + C[Z - w_9 - w_7, I_6],\ 0 + C[Z - w_9, I_6]\}$


C[I, I_n] = MAX { V_n + C[Z-W_n, I_n-1], or C[Z, I_n-1]
(Z, I_n)
(Z, I_n-1)
2^B
Z
Z-W_n
(Z-W_n, I_n-1)
(Z-W_n, I_n-2)
Z-W_n-W_n-1, I_n-2
Z-W_n-W_n-1
Θ(nZ)
theoretically
≠ polynomial
"pseudo polynomial"
input size
n items
log(Z) knapsack
0
Ø
I_1
{1}
I_2
{1,2}
I_n-2
{1:n-2}
I_n-1
{1:n-1}
I_n
1:n

DP8 Matrix Chain Multiplication

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

$n=3$

30×5 5×10 10×20

$5\times10\times20$ → 1000

multip.

5×20

$30\times5\times20$ = 3000

Total mult: 3000 + 1000

30×5 5×10 10×20

$30\times5\times10$ → 1500

30×10

$30\times10\times20$ = 6000

Total mult: 1500 + 6000

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

$[(A_1 \times (A_2 \times (A_3) \ldots)) \times A_k][(A_{k+1}(\ldots)) \ldots (A_n)]$

$p_0\times p_k$ $p_k\times p_{k+1}$ $p_k\times p_n$

① OPTSOL ch: parenth ( )( ) — ⊙ last mult $(A_1 \ldots A_k)(A_{k+1} \ldots A_n)$

#mult total = best_Left + best_Right + $p_0 \times p_k \times p_n$

(2A) $C[i,j]$ = search for last/main break $k \mid k+1$ $(A_i \ldots A_k)(A_{k+1} \ldots A_j)$

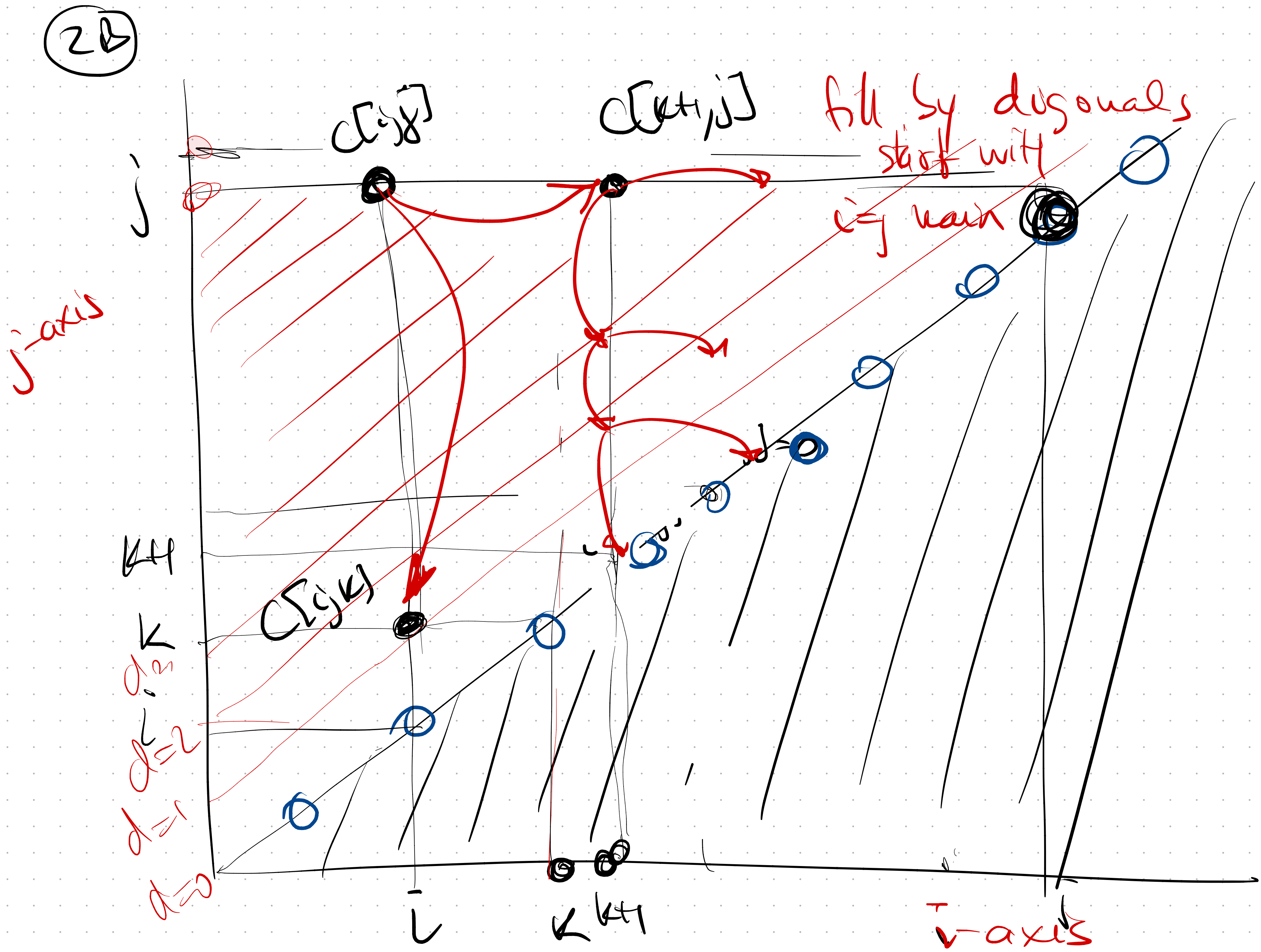
↙ min # multipli for $A_i \times A_{i+1} \times \ldots \times A_j$

$j \geq i$

$$C[i,j] = \min_{i \le k < j} \begin{cases} p_{i-1} \times p_k \times p_j & \rightarrow \text{last multip} \\ + \\ C[i,k] & \rightarrow \text{best left} \\ + \\ C[k+1,j] & \rightarrow \text{best right} \end{cases}$$

$S[i,j] = k$

$i=j$? $C[3,3] \Rightarrow A_3$ 1 matrix
$= 0$
no operation



2D
c[i,j]
c[k+1,j]
fill by diagonals
start with
i=j main
j-axis
j
k+1
k
c[i,k]
i
d=2
d=1
d=0
j=0
i
k
k+1
i-axis

```
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] = ∞
6  return LOOKUP-CHAIN(m, p, 1, n)
```

```
LOOKUP-CHAIN(m, p, i, j)
1  if m[i, j] < ∞
2      return m[i, j]
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
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.

DP 9? LCS = longest common subsequence

prefixes

$X_m = X = [x_1 \, x_2 \ldots x_m]$ $\quad X_i = [x_1 \ldots x_i]$

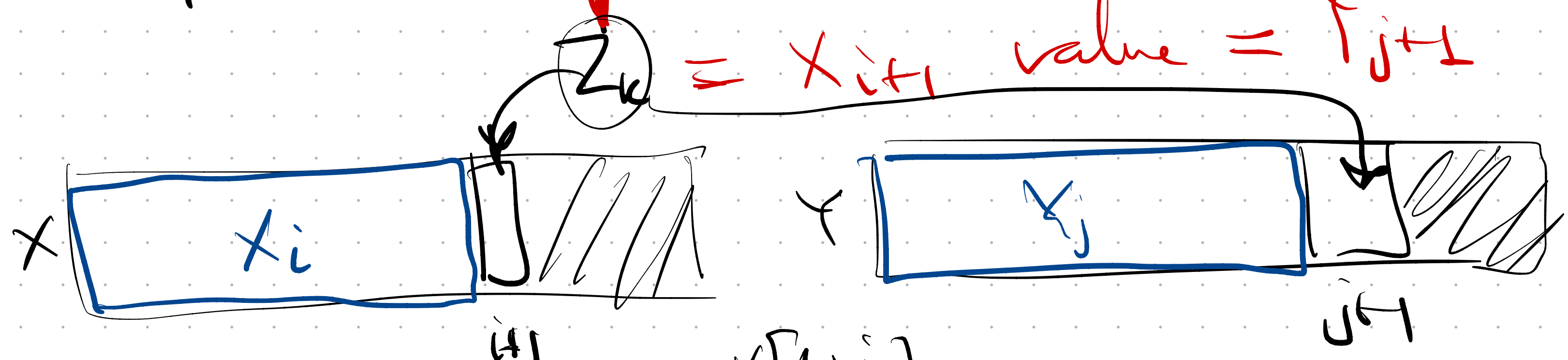
$Y_n = Y = [y_1 \, y_2 \ldots y_n]$ $\quad Y_j = [y_1 \ldots y_j]$

want longest common subseq $Z = [z_1 \, z_2 \ldots z_k]$ → subseq of X, → subseq of Y

① OPTSOL structure

||

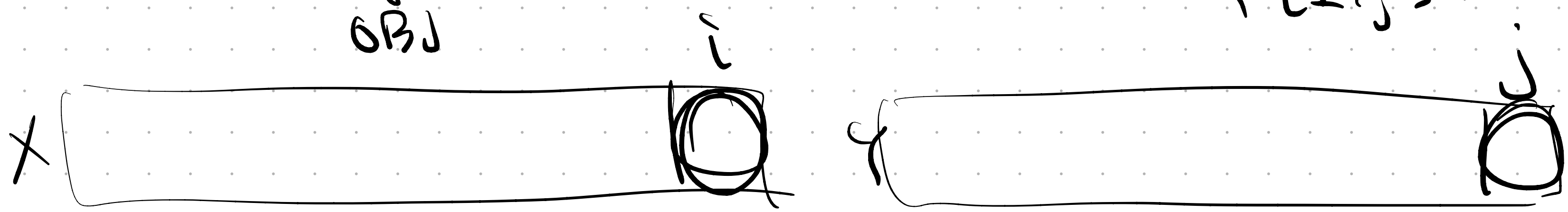
$Z = z_1 \, z_2 \ldots z_k$



X[1:i]

$Z_{k-1} = [z_1 \ldots z_{k-1}] = OPTSOL(X_i, Y_j)$

(2A) $C[i,j]$ = best "longest" OBJ LCS (X_i $X[1:i]$, Y_j $Y[1:j]$)



• if $X[i] == Y[j]$ ⇒ that's Z_k (last) + 1

↙ SPB: $C[i-1, j-1]$ + 1

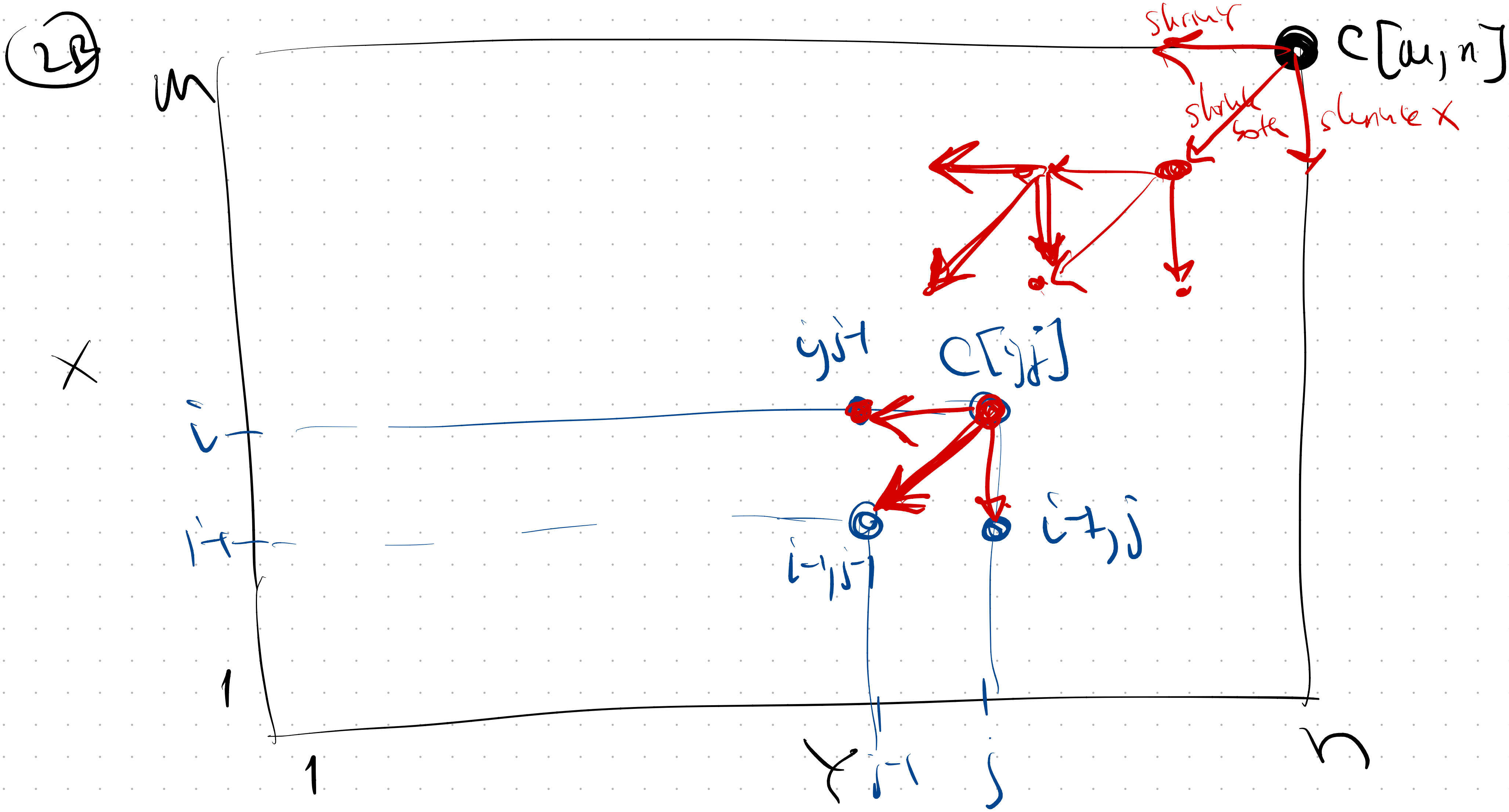
if $X[i] \neq Y[j]$ 2 possibilities

MAX { • no $X[i]$: $C[i-1, j]$ + 0 ↓

• no $Y[j]$: $C[i, j-1]$ ←

$S[i,j]$ = one of "↙", "←", "↓"

(2B)



$m \times n \times \Theta(1) = \Theta(mn)$

3, 4 – exercise

DP10 Optimal BST – weighted by probabilities

n ordered values (keys)

d_0 | k_1 | d_1 $\le$ | k_2 | d_2 $\le$ … $k_r \le$ … k_i $\le$ k_{i+1} … d_{n-1} … k_n | d_n

d_i: k_i < range < k_{i+1}

$n+1$ d-ranges

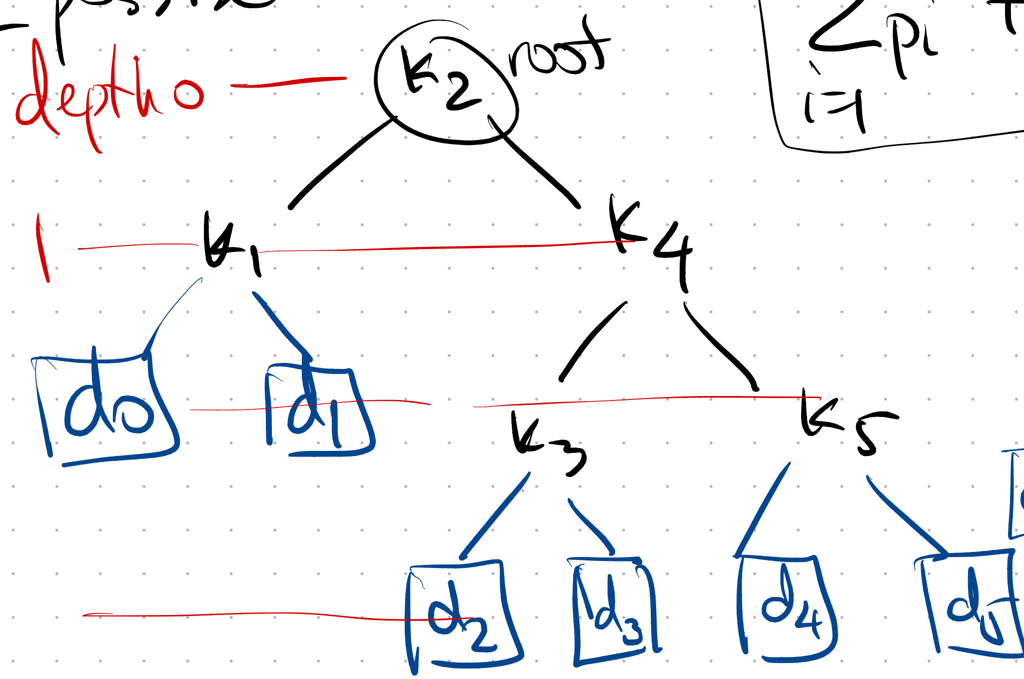
search(val): • val == k_i (found!) search prob p_i

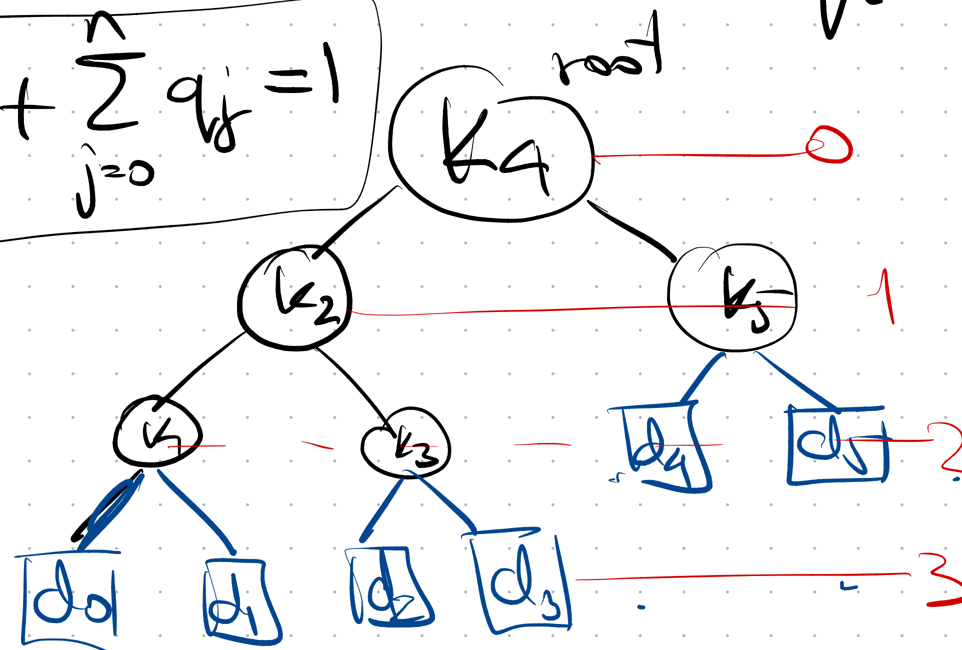
search(val): • val ≠ k (not found) search probab q_i

k_i < val < k_{i+1}

$$\sum_{i=1}^{n} p_i + \sum_{j=0}^{n} q_j = 1$$

Ex 2 possible BST n=5





$OBJ = OBJ(BST)$ = expected search cost

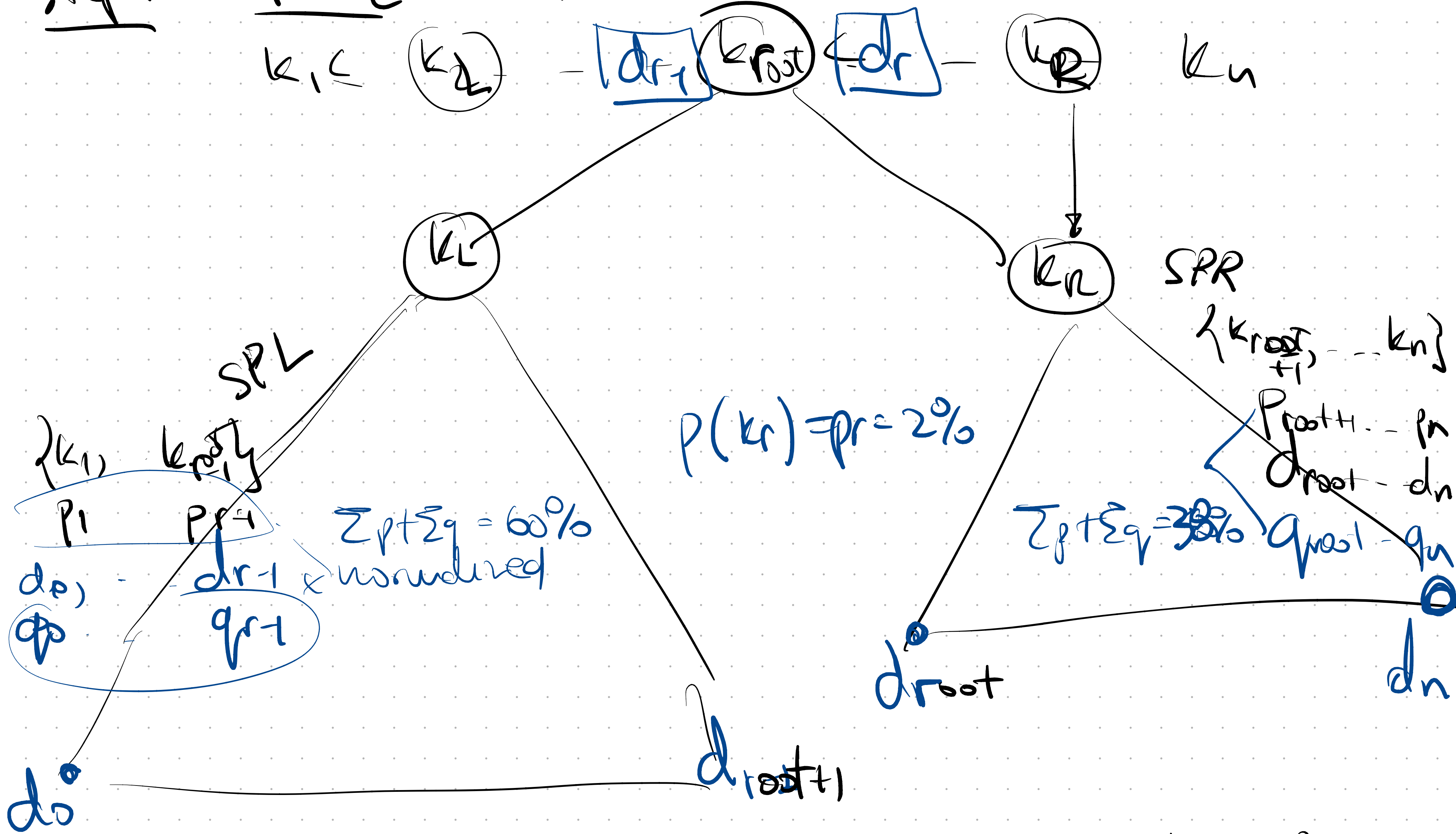
$$= \text{"} \sum_{\text{events}} \text{Cost(event)} \cdot \text{prob(event)} \text{"}$$

$\text{events} = \{ k_1, k_2, \ldots, k_n, d_0, d_1, \ldots, d_n \}$

$\text{probs} = \{ p_1, p_2, \ldots, p_n, q_0, q_1, \ldots, q_n \}$

costs = { depth of output
d-k_1 d-k_2 . d-k_n , d-d_0 . . depth- d_n }

step 1 optsol$_L$ = a BST(k, d)



claim: SPL optsol = L subtree ; SPR optsol = R subtree

(2A) $C[i, j]$ = best BST for $k_i \le k_{i+1} \dots k_{root} \le k_j$

Search for k_{root} between i, j

r = root index $i \le r \le j$

expected search cost

$\min_r = C[i, r-1] \cdot \text{prob?} + C[r+1, j] \cdot \text{prob}$

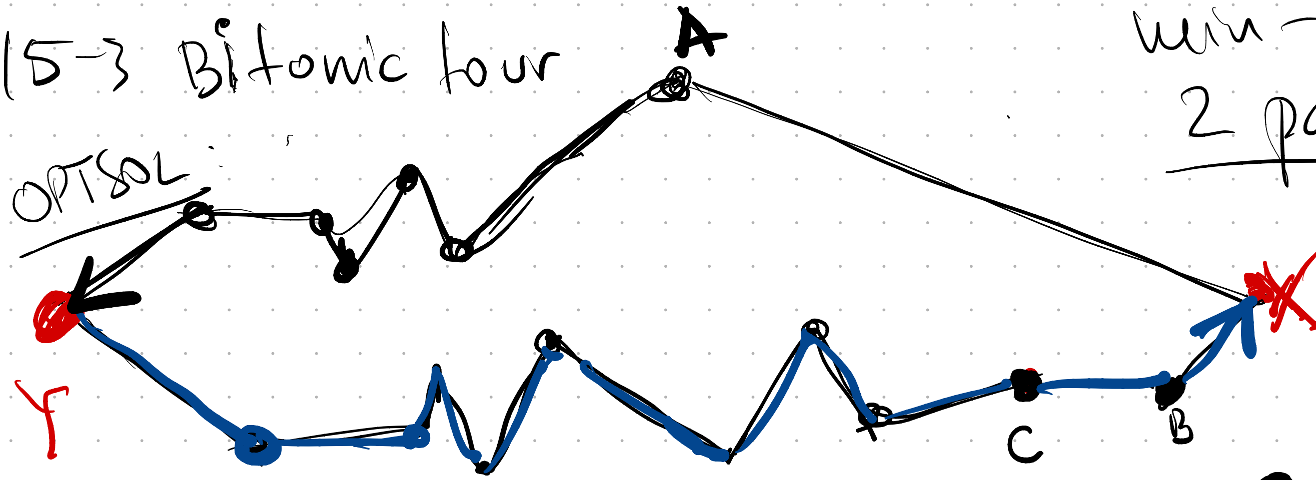
60% ... 20%

|| prob(root) · 1 + (depth)

prob(left SPR) · E[cost L] +

prob(right SPR) · E[cost R]

15-3 Bitonic tour



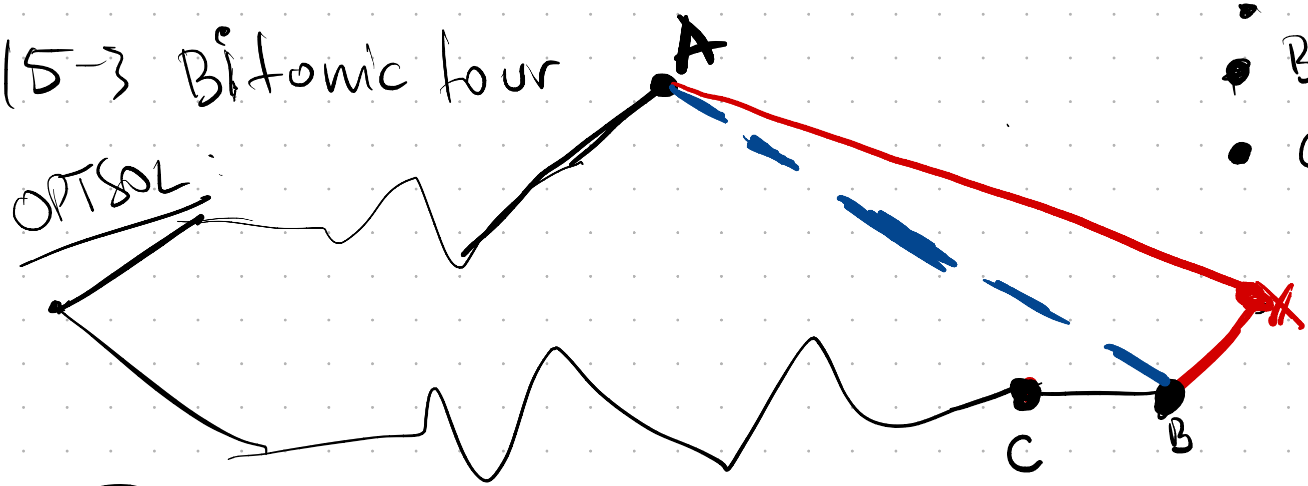
min-distance-total tour

2 paths : X, Y extreme L, R

- X → Y upper path
- Y → X lower path
- each point one of the paths.

- each path strictly directional L→R (lower) or R→L upper.

15-3 Bitonic tour



- A, B connect to X
- B closest (right most) to X
- C connects to B

SPB = PB \ {x} right most B

Idea! OPTSOL \ {x} is optimal for PB \ {x} = SPB ? Maybe Yes.
assume (contradict hypothesis) there is a better solution S for SPB = PB \ {x}

Then solution is:
- eliminate X, solve SPB = PB \ {x} with B right-most
- then connect X either to AB or to BC, whichever is better

$$C[PB_X] = C[SPB_B] + best\{+XA+XB-AB,\ +XB+XC-BC\}$$

idea 2

A × B > x closest to X → 2 possibilities in OPTSOL (A > B > X closest)

- A, B diff paths

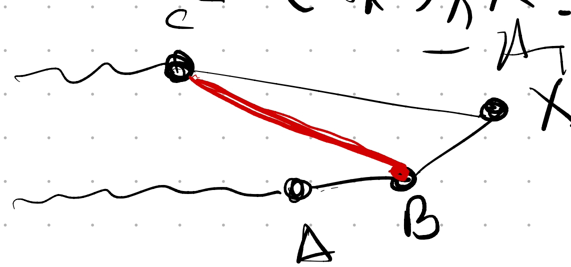
SPB = PB \ {x}

AB ∈ sol(SPB)

OPTSOL \ {AX, BX} = sol(SPB) \ {AB}
or exchange argument

Solution =
= sol(SPB) + XA + XB − AB
connect X

- A, B same path

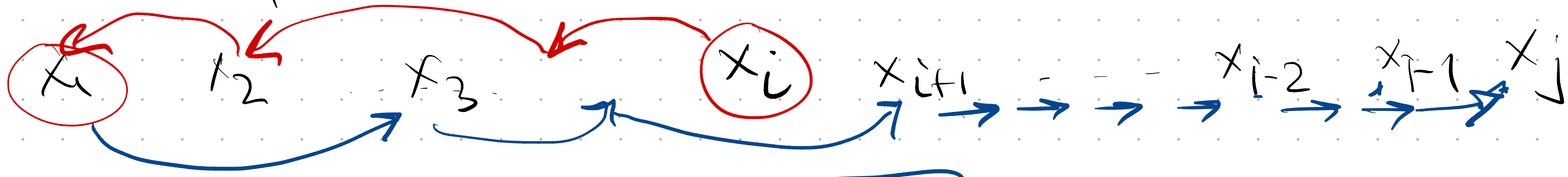


SPB = PB \ {B} where X is still right-most but A is closest

SOLUTION = sol(SPB) + BA + BX − XA
connect B

idea3

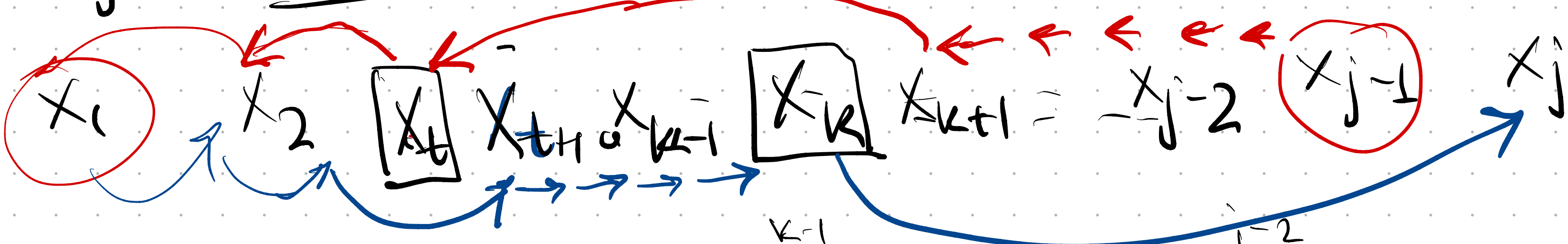
Sort x-positions 1–j. Pick x_i.



want bitonic tour $x_i \rightsquigarrow x_1$ (Left) $\rightsquigarrow x_j$ (Right)

$c[i,j]$ = best bitonic path objective from x_i to x_j passing through all points $x_1, x_2, \dots x_i, \dots x_j$ sorted.

- if i $j-1$ easy: $c[i,j] = c[i,j-1] + dist(x_{j-1}, x_j)$
- if $i = j-1$ <u>not easy</u>: search for x_k that right-jumps $x_k \curvearrowright x_j$



$$c[j-1,j] = \text{best}_{t,k} \left(\sum_{\ell=t}^{k-1} \|x_\ell x_{\ell+1}\| + \sum_{\ell=k+1}^{j-2} \|x_\ell x_{\ell+1}\| + c[t,k] + \|x_t x_{k+1}\| + \|x_k x_j\| \right.$$

$$C[j-1, j] = \underset{k}{\text{best}} \left(\sum_{\ell=k+1}^{j-2} \|x_\ell \, x_{\ell+1}\| + \|x_k \, x_j\| + \underset{\text{reversed}}{C[k, k+1]} \right)$$

Midterm PB4

$a_i \quad a_{i+1} \quad a \quad - - - \quad a_{j-1} \quad a_j$

$S_{ij} = \sum_{k=i}^{j} a_k$

$$C[i,j] = \max \begin{cases} \text{pick } a_i & S_{ij} - C[i+1, j] \\ \text{pick } a_j & S_{ij} - C[i, j-1] \end{cases}$$