- Sat 3/6 Midterm
- Wed 3/10 Lecture Jay Aslam
  Teams Only (@ WVF 020)
- cheating on HW
- HW 5 due Fri 3/5
- HW 6 dyn Prog

PB

Greedy

- Divide (split) SUBPB
  Decide
  Break
- Solve SubPb
- $\boxed{Sol}$ = combine (Subpb SOL)

Dynamic Programming.

- Look at all possible SUBPB
  dont know how to break it
- Solve all possible SUBPB
  (even ones we dont need)
- given SOL(subpb) decide the split ⇒ which SUBPB we need
- $\boxed{SOL}$ = comb (selected subpb SOL)

Brute Force

- Try all possibilities
- Keep track of obj
- Return best SOL (OPT SOL)

SOL

Act. Sel

all subsets of non-overlapping activities

$|\mathcal{P}(A)| = 2^n$

DP writing parts (required)

1. Chard OPT SOL = SPLIT Funct (sub-pb OPT SOL)
   - thinking exercise for you, rather than formal

2A. C[input] = value <u>recurrence</u> of OBJ.
   "Rec value of OPT SOL"

2B. Subproblem-dependency table/graph
   (drawing, usually a table ⇒ visual of 2A

3. Compute C[ ] table (usually all inputs/table
   $\boxed{\text{usually bottom-up}}$
   <u>sometimes</u> rec+cache top down)

4. Trace the solution/choices

5. Run Time

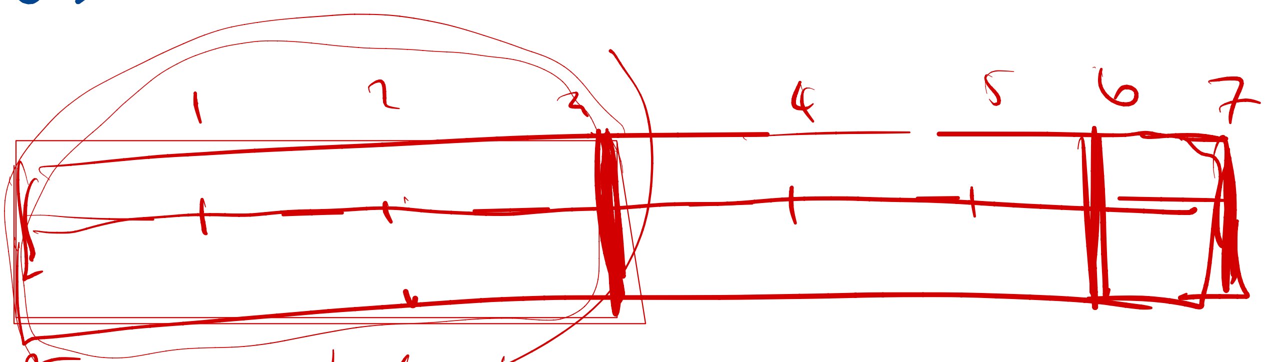
$\boxed{\text{DP 1}}$ Rod-cutting n length rod.

table of prices

length	1	2	.	—	—	n
price	p_1	p_2				p_n

p ∀ length

Task: cut the rod to max total value



OPT sol | Greedy choice ≠ OPTSOL

L	1	2	3	4	5	6
V	1	2	4.5	6.4	8	
Q	1	1	1.5	1.6		
Tot			9	6.4 + 2		

① OPT sol charact / split

ex. $l_1 = 3$, $l_2 = 3$, $l_3 = 1$

⇒ OPT sol comp. for each piece of rod

$n = 3 \Rightarrow$ OPT SOL $= (l = 3)$

$n = 4 \Rightarrow$ OPSOL $= (3, 1)$

(2A) $C[n, (p_1, p_2, \dots, p_n)]$ = max total value

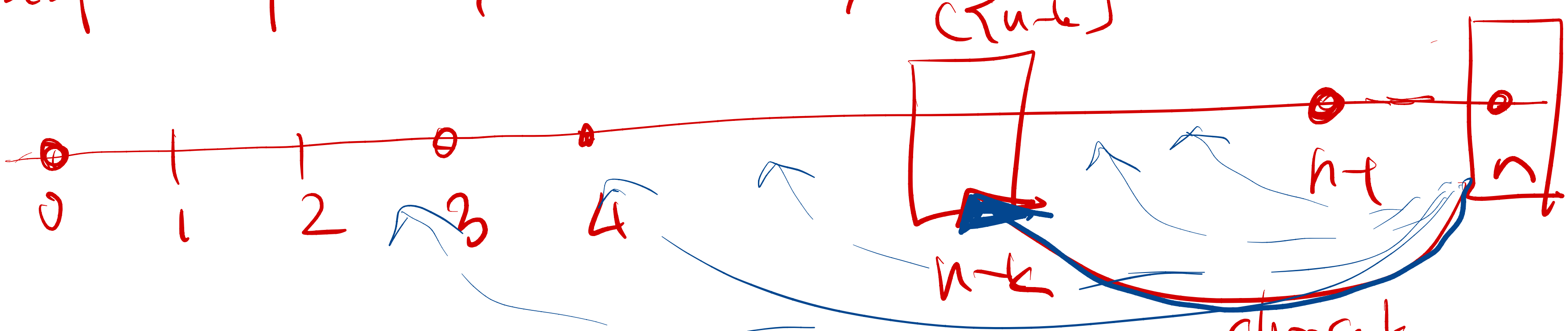
Global

k = first cut (unknown)

$$C[n] = \max_{\underset{(\text{search})}{k}} \left\{ \underbrace{p_k}_{\text{value at cut } k} + \underbrace{C[n-k]}_{\text{sub pb}} \right\}$$

- solve $C[n-k]$ all subproblems first

(2B) subpb dependency 1-dim / table



1-dim array/matrix/table

③ Fill / compute table C[] bottom up.

$C[0] = 0$

for $i = 1 : n$

$C[i] = \max_{1 \le k \le n} \{ p_k + C[n-k] \}$ $\Theta(n)$ // the value

$S[i] = \arg\max_k (p_k + C[n-k])$ // the k

// I now have C[n]

④ Trace Solution

→ track it from C[] itself ⇒ procedure

→ or explicit S[input] = choice/decision ⇒ nothing

PrintSolution (n)

if $n \le 0$ exit

print S[n] = k,

PrintSolution (n−k)

...

DP 2 Coin change

$d_1, d_2, \ldots, d_n$ denominations

∞ coins

Task: min # of coins

n = # cents to make

1 Charact OPT SOL

k = # coins

$$\underbrace{\overbrace{d_1 \mid d_3 \mid d_1 \mid d_2}^{4 \text{ coins}}}_{\substack{b \text{ cents} \\ \text{optimal}}} \;\Big\|\; \underbrace{\overbrace{d_4 \mid d_5 \mid d_2}^{3 \text{ coins}}}_{\substack{n-b \text{ cents} \\ \text{optimal}}}$$

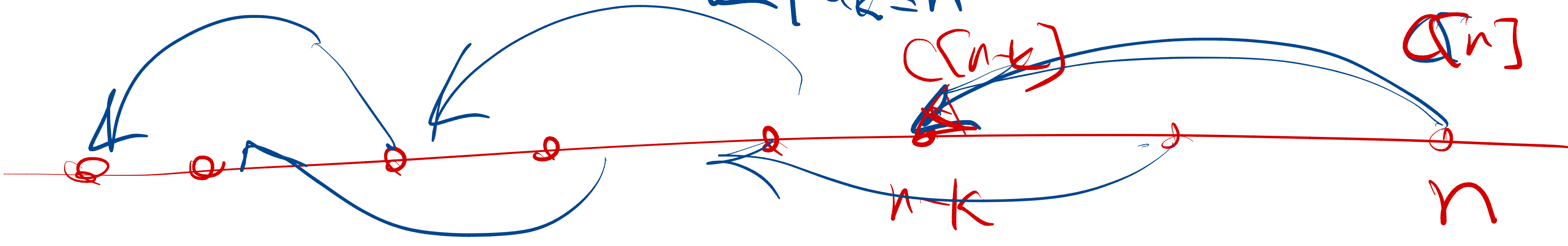
2

$C[\overset{\text{input}}{n}]$ = # of coins

$$C[n] = \min_{k \mid d_k \le n} \left\{ 1 + C[n-d_k] \right\}$$

search for first coin d_k

2B



③ Fill the C[ ] table left → Right

$C[0]=0$

for $i=1:n$

$$C[i] = \min_{1 \le k \le \boxed{d_k \gg n}} \{1 + C[n-d_k]\}$$

$$S[i] = \arg\min_{k} \{ \cdots \quad — \}$$ ⟹ k or d_k

④ Trace Solution

search for k

$C[100] = 11$ need to find k / d_k

so $\boxed{C[100-d_k] = 10}$

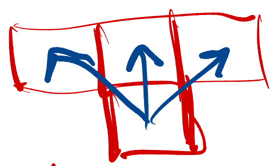
OPT sol subpbs: 1, 2, 3, 4, 5, 6

DP3 Chock board best path
min

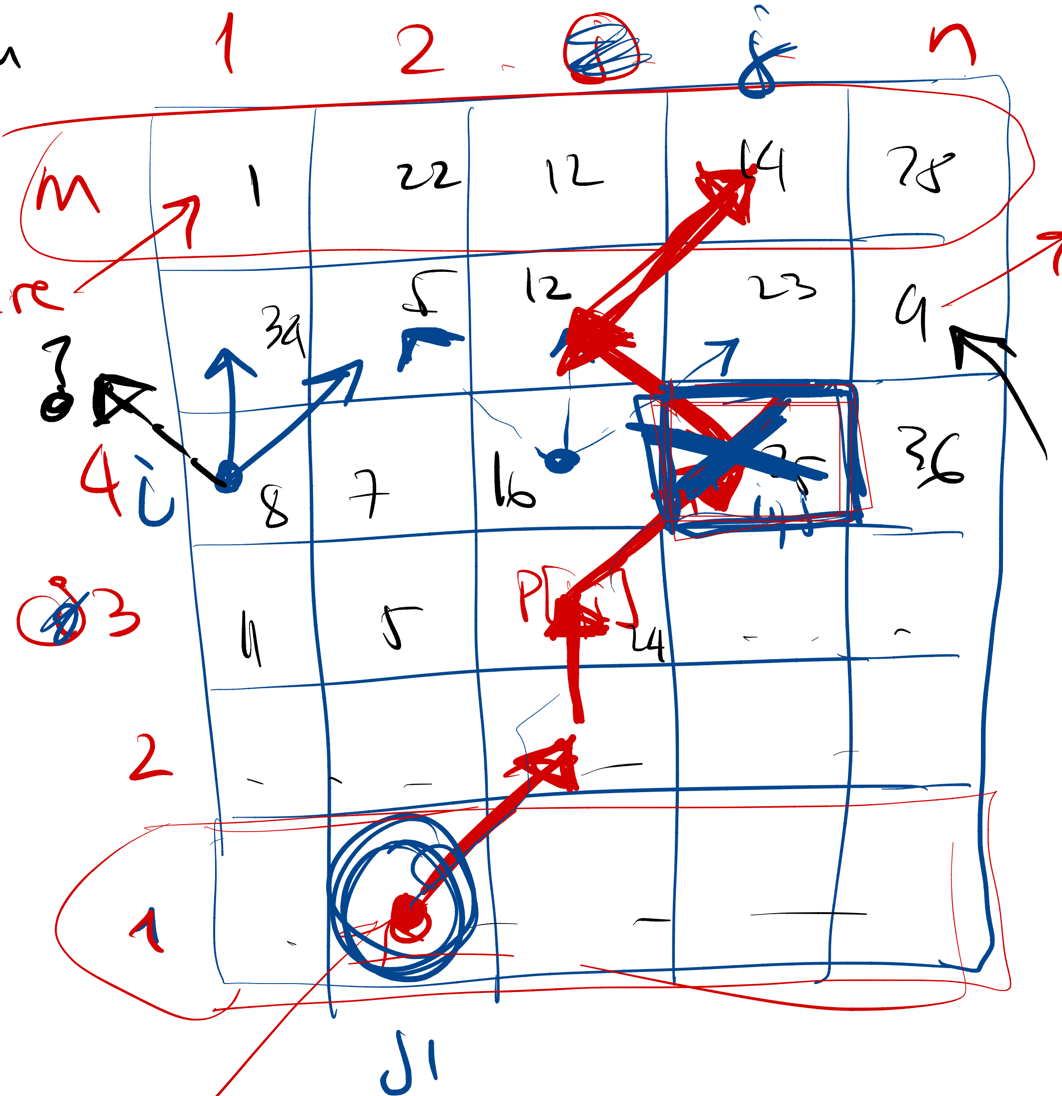
P[i,j] = penalty of stepping here
row, column

- start anywhere on row = 1
- finish anywhr on row = m

3 moves are up one row



Task ~~min~~ path total penalty.



1. Character OPTSOL $\Rightarrow$ new task
optimal path from
cell $(1, j_1)$ to cell (i, j)

new task

any path from anywhere on first row → all (i,j)

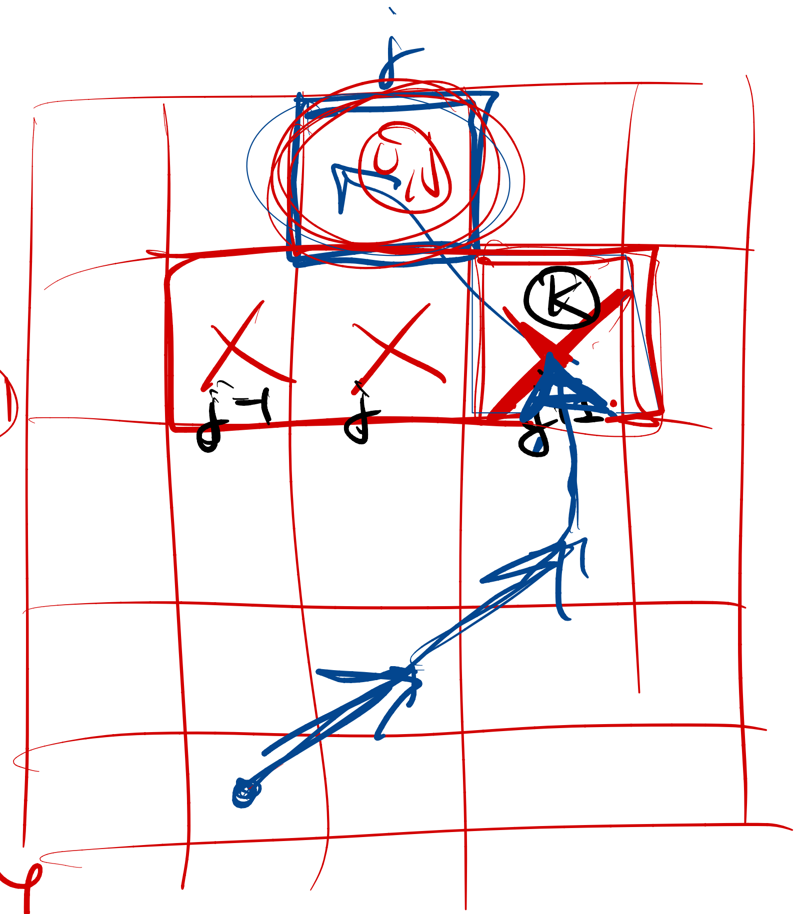
② $C[i,j]$ = penalty of best path to $\boxed{i,j}$

search for the last move ↗↑↖

$\underbrace{j, j-1, j+1}_{k}$ row $i-1$, colum $j-1$, j, $j+1$

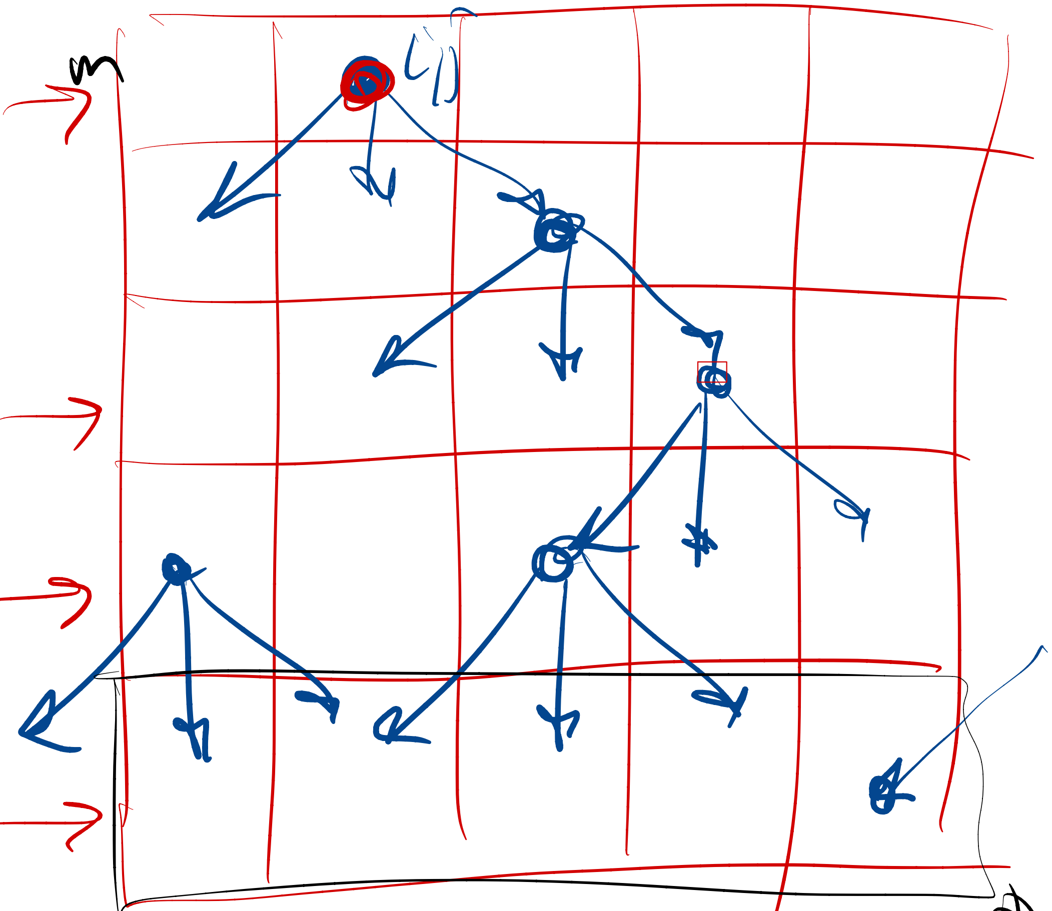
$= P[i,j] + C[i-1,k]$

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



(2b) SUBPB dependency table

at row k, must have all prev rows 1→k-1



(3) bottomup

C[first row] = P[first row]

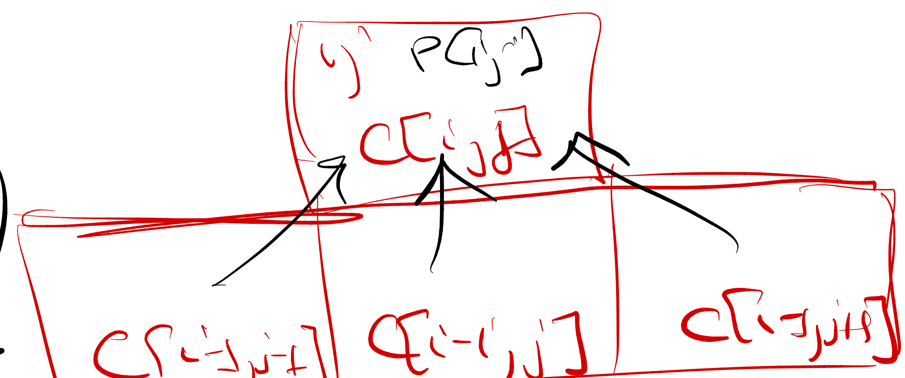
for r=2:m
    for c=1:n
        //solve C[r,c]

$$C[r,c] = p[r,c] + \min\{ C[r-1,c-1],\ C[r-1,c],\ C[r-1,c+1] \}$$

S[ ] = ? store the j under it

(4) solution (i,j)

Prin P[i,j], (i,j)



j_{new} =? $C[i-1, j_{new}] + P[i,j] == C[i,j]$

orig pb: - find cell $i = m$, j with min $C[i,j]$
last row
- use that cell in solved pb.

Kinda DP

$\binom{n}{k}$ = # of subsets of size k out of n

= # ways to pick k items out of n

$C[n,k]$ = # of ways . . . $= \binom{n}{k}$

1. indexed

1	2	3	. . n−1 objects	n

choose k out of n — 2 options

→ choose item n — $C[n-1,k-1] = \binom{n-1}{k-1}$

→ do not choose item n — $C[n-1,k]$

2. $$\boxed{C[n,k] = C[n-1,k-1] + C[n-1,k]}$$

n

1   1

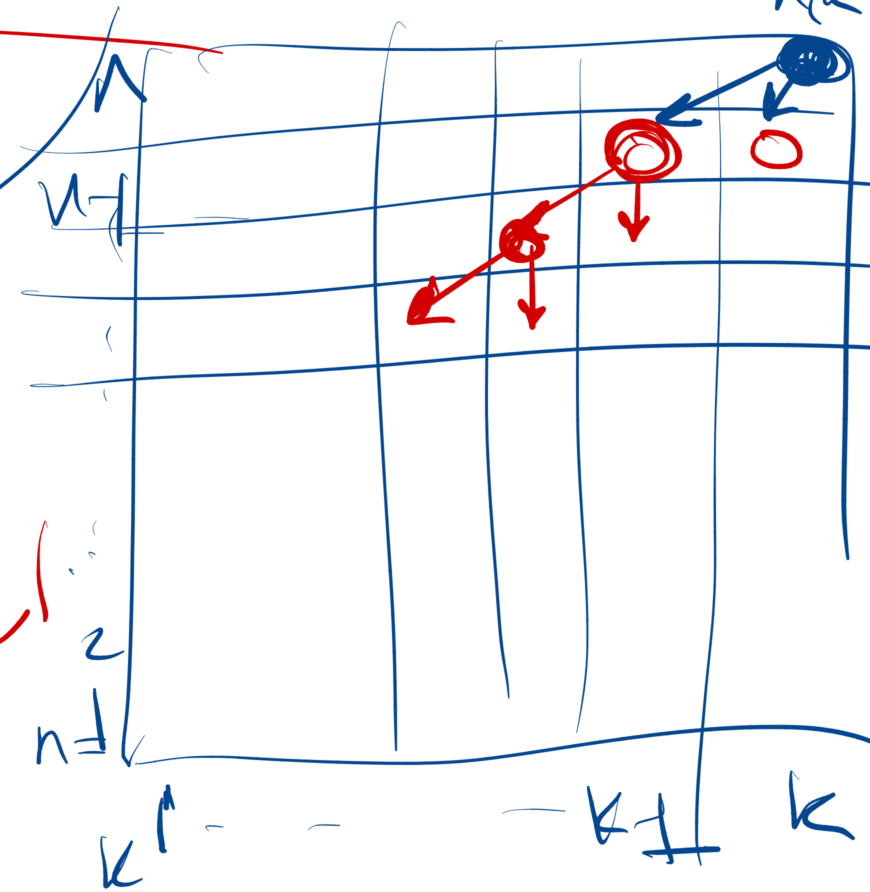
2   1 2 1

3   1 3 3 1

4   1 4 6 4 1

5   1 5 10 10 5 1

6   1 6 15 20 15 6 1



n

Discrete Knapsack

	1	2	3	...	n
values	v_1	v_2	v_3	— —	v_n
weights	w_1	w_2	w_3	— —	w_n

Knapsack max weigh W

Task: select the max-value subset of items subject to total weight $\leq W$

$$C[W, \underset{\{1,2,\dots n\}}{\text{item\_set}}]$$

choose k

$$\max_{\underset{\text{item}}{k}} \left\{ v_k + C[W - w_k, \text{item\_set} \setminus \{k\}] \right\}$$

not good

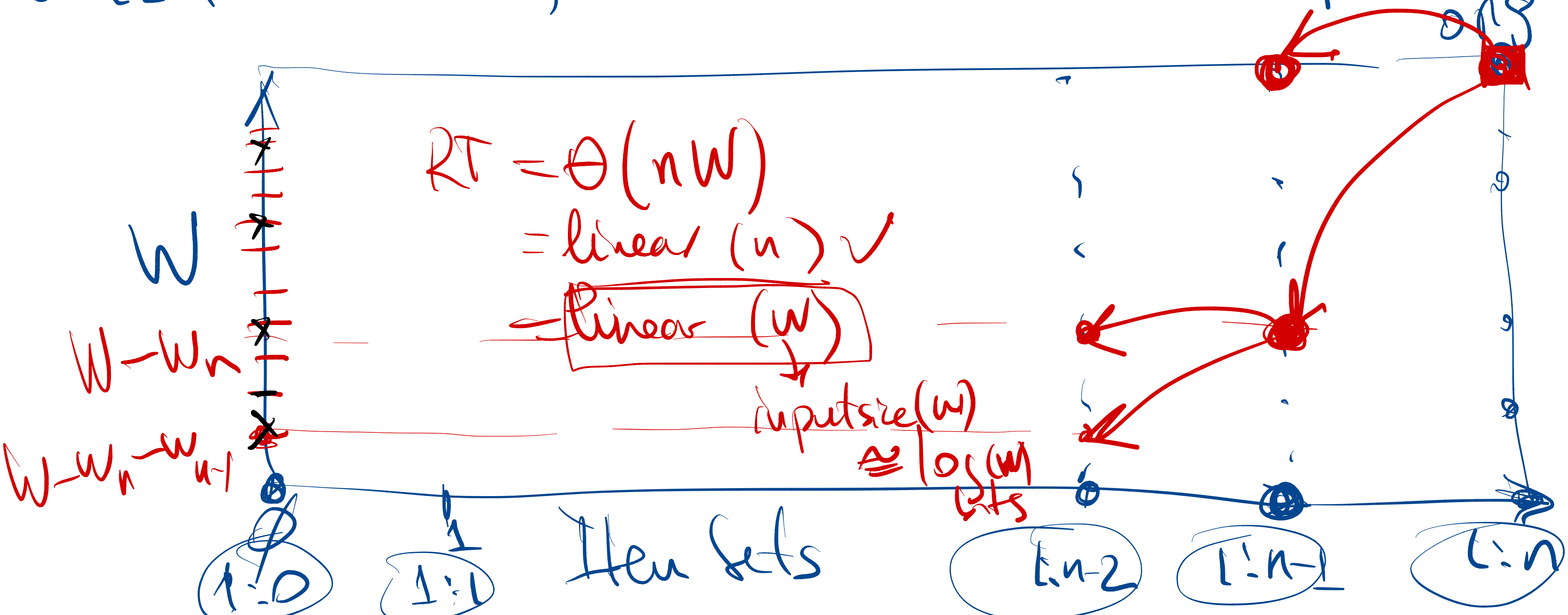
2B Subproblem Table

$= \exp(2^n)$ bad!

$$C[W, \text{Itemset}_{1:n}] = \max \begin{cases} \text{choose item } n: & V_n + \boxed{C[W - w_n, \text{Itemset}_{1:n-1}]} \\ \text{don't choose item } n: & \boxed{0 + C[W, \text{Itemset}_{1:n-1}]} \end{cases}$$

- is this "last" choice $\Longrightarrow$ . . $\Longrightarrow$ OPT SOL

- is this solving the "Item-set" axis exp issue? ans?



items

OPT SOL = { 2, 3, 5, 11 }

$\boxed{\text{items only } 1:20}$

RT = linear(W)

W ≠ linear input ; W = exp(input size)

$\log_2 W$ bits

$X_t = (x_1, x_2, \ldots x_t)$

$Z_t = (z_1, z_2, \ldots z_t)$

Longest Common Subsequence

$X = (x_1 x_2, \dots x_m)$

$Y = (y_1, y_2 \dots y_n)$

Find Z = longest common subsequence

ex $subseq(X) = (x_1, x_3, x_7, x_9)$

① Z = OPT SOL CHAR $\quad Z = (z_1 z_2 \dots z_k)$

1) • if $x_m = y_n \Rightarrow z_k = x_m = y_n$; $Z_{k-1} = LCS(X_{m-1}, Y_{n-1})$ $\quad 1:k-1$

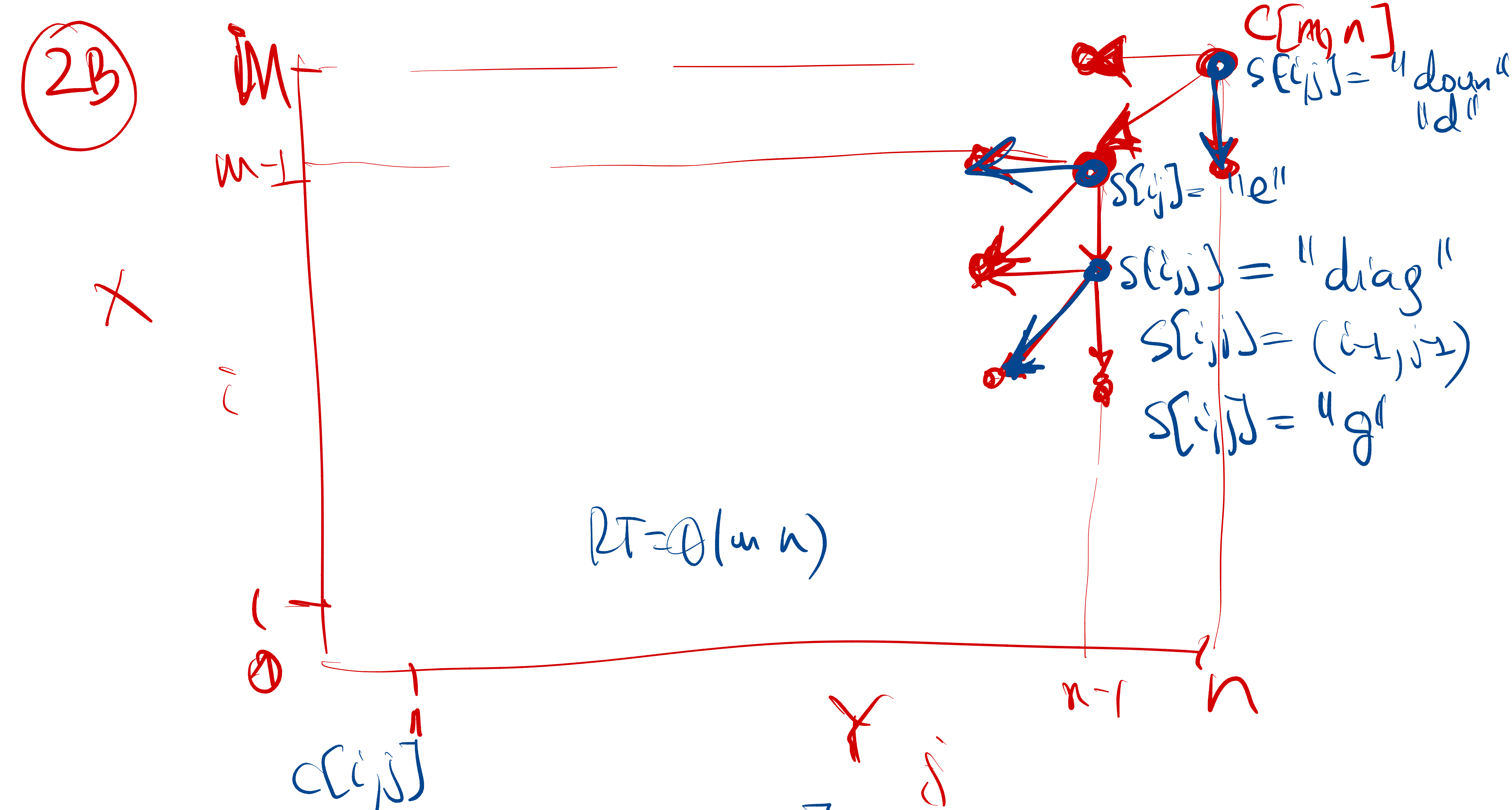
2) • $\cancel{x_m \neq y_n}$ and $z_k \neq x_m \Rightarrow Z = Z_k = LCS(X_{m-1}, Y = Y_n)$

3) • $z_k \neq y_n \Rightarrow Z = Z_k = LCS(X_m, Y_{n-1})$

②

$c[i,j] = \text{length}[LCS(X_i, Y_j)]$ $\quad 1:i \quad 1:j$

$$c[i,j] = \begin{cases} 0 & \text{if} \dots \\ 1 + c[i-1, j-1] & \text{if } x_m = y_n \\ \max(c[i, j-1], c[i-1, j]) & x_m \neq y_n \end{cases}$$



$S[\text{same input as } c]_{i,j}$ = choice used in OPT SOL

Matrix Chain Multiplication.

$$A_1 \times A_2 \times A_3 \ldots A_k \times A_{k+1} \times A_{k+2} \ldots \times A_n$$

$(p_0 \times p_1)$ $(p_{k-1} \times p_k)$ $(p_k \times p_{k+1})$ $(p_{n-1} \times p_n)$

best order?

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

30×5 $\underbrace{5 \times 10 \quad 10 \times 20}$ 30×5 5×10 10×20

$RT = 5 \cdot 10 \cdot 20 = 1000$

$RT = 30 \cdot 5 \cdot 10$

1500

30×5 5×20

$RT = 30 \cdot 5 \cdot 20 = 3000$

Total $= 4000$

(30×10) (10×20)

$RT = 30 \cdot 10 \cdot 20$

6000

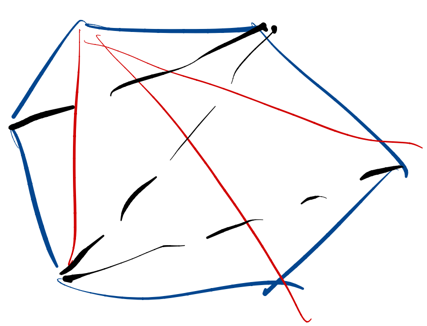
Total 7500

all possibilities?

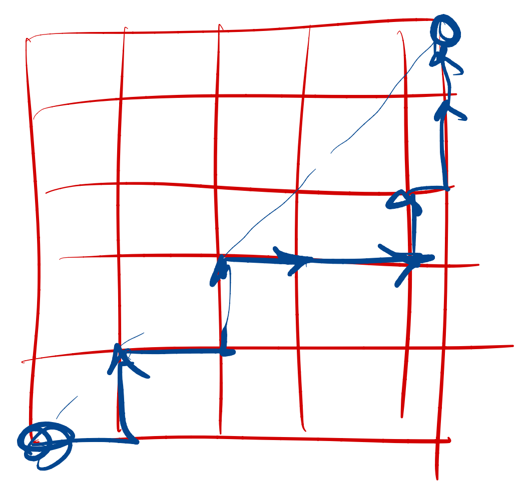
$(A_1 A_2)(A_3 A_4)$

$A_1((A_2 \cdot A_3) A_4)$

$((A_1 \cdot A_2) \cdot A_3) \cdot A_4$



$\geq 2^n$



① matrices from $i..j$

$$\left(A_i\,(A_{i+1})\,(\ldots)\,A_k\right)\left(A_{k+1}\,(\ldots)\,(\ldots)\,A_j\right)$$

$p_{i-1} \times p_i$ $p_{k-1} \times p_k$ $p_k \times p_{k+1}$ $p_{j-1} \times p_j$

OPT SOL(i, j)

$p_{i-1} \times p_k$ last multiplication $p_k \times p_j$

claim: L side, R side have to be optimal parenthesization $\Longleftrightarrow$ opt. mult. sequence

② $c[i,j]$ = min # of multipl. for matrix seq $A_i \cdot A_{i+1} \cdots A_j$

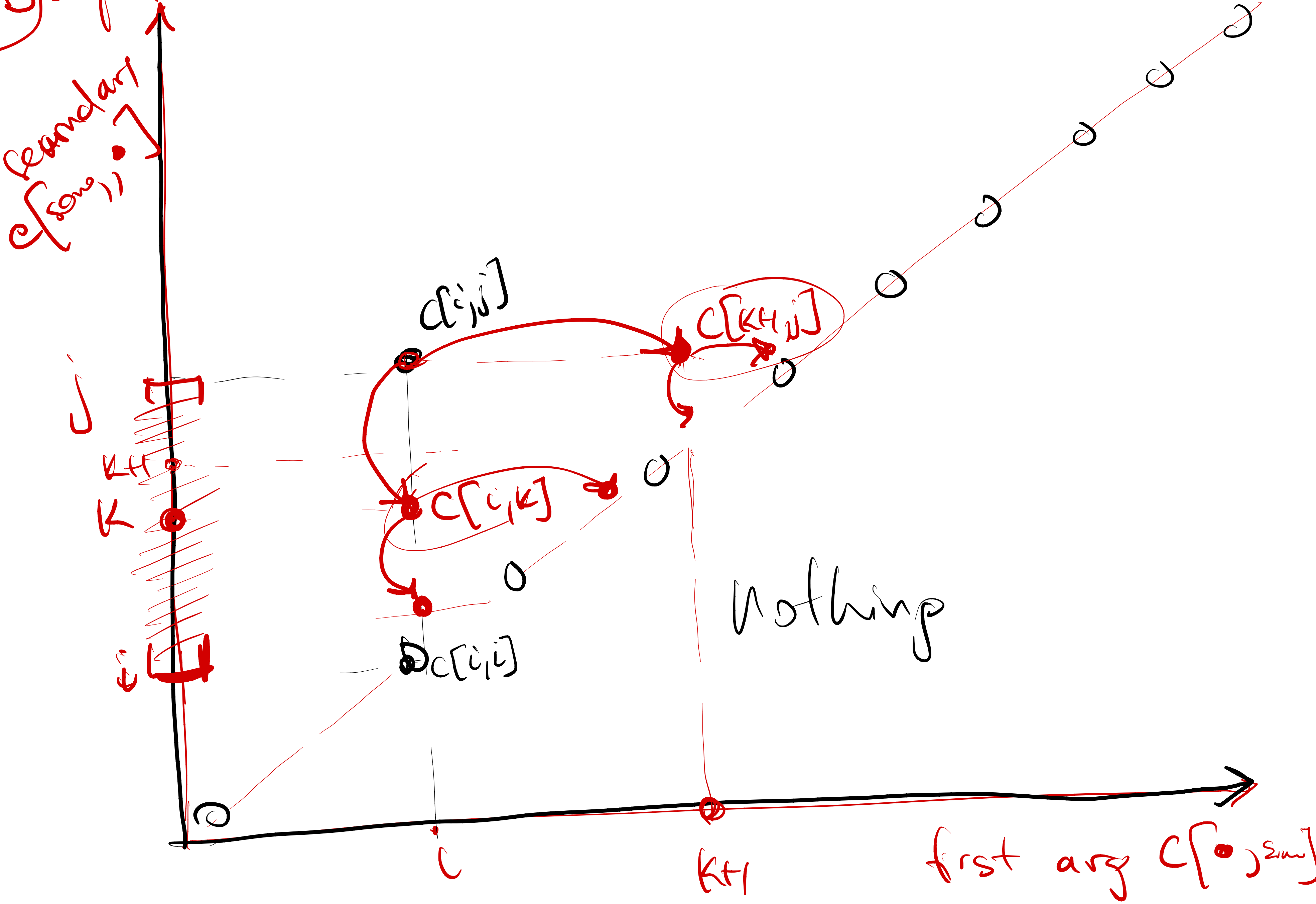
- if it breaks at k (last mult is $A_k \times A_{k+1}$)

$$c[i,j] = \underbrace{c[i,k]}_{\substack{\text{left side} \\ i:k}} + \underbrace{c[k+1,j]}_{\substack{\text{right side} \\ k+1:j}} + \underbrace{p_{i-1} \cdot p_k \cdot p_j}_{\text{last mult.}}$$

$i \leq k < j$

$c[i,i] = 0$

2B dep. table


secondary
c[some, j]
c[i,j]
C[k+1,j]
j
k+1
k
C[i,k]
i
c[i,i]
nothing
i
k+1
first arg

Memoization : top-down computation + cache
+ recursion

1) c still a table

2) cache c[ ] computed

3) never compute or recurse on stored c[ ]

3) only recurse on { not-computed c[ ]
required

```
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.

HW questions

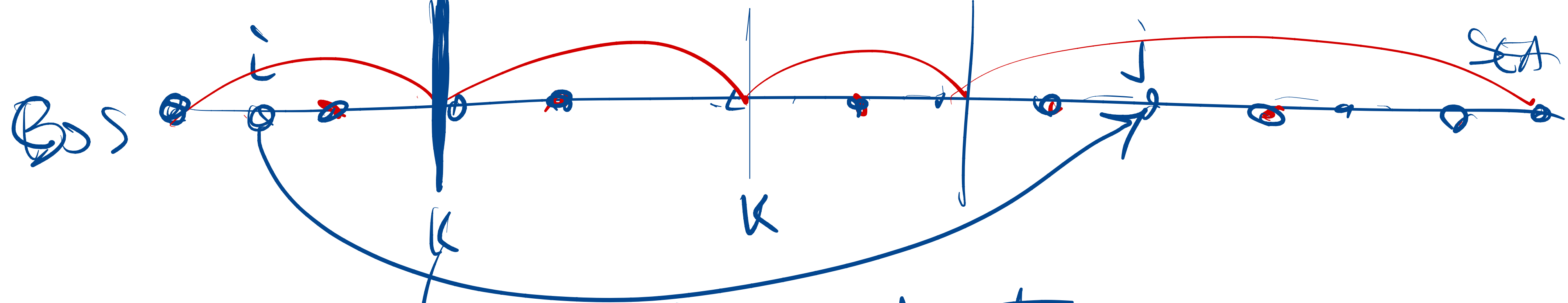
$X_{1:i} \rightarrow Y_{1:j}$

op

cond

$$C[i,j] = \min \begin{cases} C[i-1,j-1] + \text{cost}(\text{copy}) & x_i = y_j \\ C[i-1,j-1] + \text{cost}(\text{replace}) & x_i \neq y_j \end{cases}$$

cost



first change ranks

$P[i,j]$ = direct

$C[i,j]$ = min cost (with hops) from $i \to j$

$C[i]$ = min cost from BOS to i ???

Virgil OH Thu 4:30pm ?

not OH Monday