# Weighted Independent Set on A Path.

Input o—o—o ..... o—o

$v_1$ $v_2$ .... $v_{n-1}$ $v_n$.

with weights $w_1$ $w_2$ ..... $w_n$.

Output an independent set $S$ with maximum weight.

a subset of vertices s.t. no two are connected by an edge

o—o—o—o

$v_1$ $v_2$ $v_3$ $v_4$

1 4 5 4

=> if greedy

first 5, then 1 => wrong

=> if divide & conquer

first 4, then 5. but connected

=> wrong

Input o—o—o ..... o—o

$v_1$ $v_2$ .... $v_{n-1}$ $v_n$.

with weights $w_1$ $w_2$ ..... $w_n$.

case 1: $v_n \notin S^*$ $S^*$ = opt for $G_{n-1}$ : o-o··· o $v_{n-1}$

case 2: $v_n \in S^*$ $S^* = \{v_n\} \cup$ opt for $G_{n-2}$

Subproblems:

for $i \in [0, n]$, define

$C[i]$ = total weight of opt for $G_i$

$$C[n] = \max\{C[n-1],\ C[n-2] + w_n\}$$

Recurrences:

$$C[i] = \max\{C[i-1],\ C[i-2] + w_i\} \text{ for any } i \in [2, n]$$

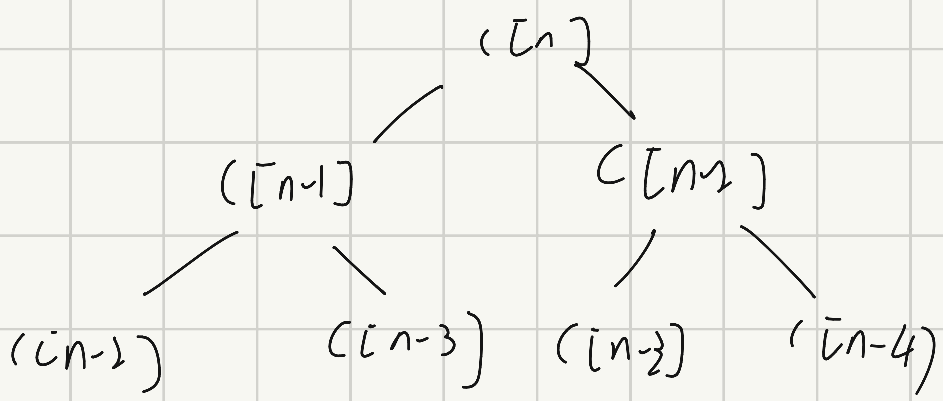
Base case $\begin{cases} C[1] = w_1 \\ C[0] = 0 \end{cases}$

Computing $C[i]$

1. recursion.

```
recur(i)
    if i == 0 or i == 1
        return base case.
    else if i >= 2
        return max{ recur(i-1), recur(i-2) + w_i }
```

$$T(n) = T(n-1) + T(n-2) + O(1) \Rightarrow T(n) = O(c^n)$$



2. recursion with memorization

global $c[0, \cdots n]$.

$c[0] = 0$ $c[1] = W_1$ $c[i] = -1$ for $i > 1$.

recur (i)

if $c[i] \geq 0$.

return $c[i]$

else

$c[i] = \max\{ \text{recur}(i-1), \text{recur}(i-2) + w_i \}$

return $c[i]$

$O(n)$

3. Iteration

$c[0] = 0$

$c[1] = w_1$

$O(n)$

for $i = 2$ to $n$

$c[i] = \max\{ c[i-1], c[i-2] + w_i \}$

Reconstructing opt solution.

$c[0]$, $c[1]$, $\cdots$ $c[n]$

$S^*$ ?

if $c[n] == c[n-1]$

$v_n \notin S^*$

else // $c[n] == c[n-2] + W_n$

$v_n \in S^*$

Iteration version :

```
S* = φ
i = n
while i >= 2
    if c[i] == c[i-1]
        i = i-1
    else // c[i] == c[i-2] + wi
        S* = S* U {vi}
        i = i-2
if i == 1
    S* = S* U {vi}
return S*
```

```
recon c(n)
    if n == 0 or 1
        base case
    if n >= 2
        if c[n] == c[n-1]
            return recon(n-1)
        else // case 2
            return {vn} U recon(n-2)
```

Dynamic Programming

1. define subproblem
2. finding recurrence
3. computing the optimal value for (sub) problems
4. reconstructing the optimal solution

Knapsack Problem

Input: $n$ items with weight $w_1, \ldots, w_n$
and values $v_1, \ldots, v_n$

capacity : $C$.

output = a subset of items with maximum $\sum_{i \in S} v_i$
s.t. $\sum_{i \in S} w_i \le C$.

case 1: $n \notin S^*$ $\quad S^* :=$ opt for first $n-1$ items with total weight $\le C$,

case 2: $n \in S^*$ $\quad S^* := \{n\} +$ opt first $n-1$ items with total weight $\le C - W_n$

subproblems:

for $i \in [0, n]$ for $c \in [0, C]$
define $V[i][c]$ be the maximum total value of a subset of first $i$ items with total weight at most $c$

$V[n][c] = \max\{ V[n-1][c], V_n + V[n-1][c-w_n] \}$

Recurrence

for any $i \in [1,n]$ for any $c \in [0,C]$

$$\begin{cases} V[i][c] = \max \{ V[i-1][c], \ v_i + V[i-1][c-w_i] \} \\ V[0][c] = 0 \quad \text{for } c \in [0, C] \\ V[i][c] = -\infty \quad \text{for } c < 0 \end{cases}$$

Computing $V[\ ][\ ]$

```
V[0][c] = 0      for c ∈ [0,C]
for i=1 to n
    for c = 0 to C
        if Σ_{k=1}^{i} w_k > c
            V[i][c] = V[i-1][c]
        else
            V[i][c] = max{V[i-1][c], V[i-1][c-w_i] + v_i}
return V[n][C]
```

time: $O(nC)$

space: $nC$

Reconstructing opt sol

```
c = C
S = ∅
for i = n to 1
    if c >= w_i and V[i][c] == V[i-1][c-w_i] + v_i
        S = S ∪ {i}
        c = c - w_i
return S
```

time $O(n)$

space: $O(nC)$

Remark

time: $O(nC)$ → $\log_2 C$ bits to represent $C$. (exp)

→ pseudo - polynomial time

space: if only cares step 1 to 3

$O(n+C)$ actually

if requires opt solutions

$O(n+C)$ ← can reduce to

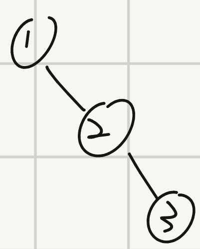
## Optimal BST

Input: $n$ keys $1, 2, \ldots, n$ with

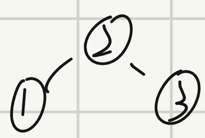
freq $p_1$ $p_1$ $p_2$, $\ldots$ $p_n$.

output: a BST with minimum average search time

1. 0.8
2. 0.1
3. 0.1

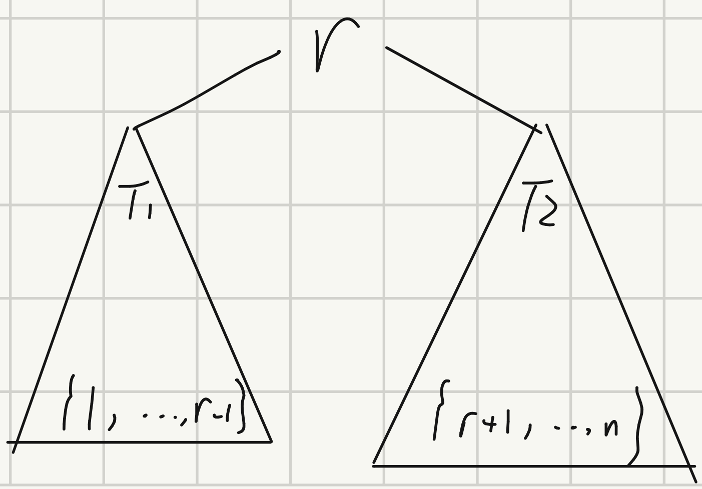



: $0.8 + 0.2 + 0.3 = 1.3$




: $0.8 \times 2 + 0.1 \times 1 + 0.1 \times 2 = 1.9$ ×

$T^*$



search time of $K$ in $T^* = 1 +$ search time of $K$ in $T_1$
(if $K$ in $T_1$)

$\sum_{K=1}^{n} P_K \cdot$ search time of $K$ in $T^* = \sum_{K=1}^{r-1} P_K$ (search time of $K$ in $T_1$ +1)
$+ P_r$
$+ \sum_{K=r+1}^{n} P_K$(search time of $K$ in $T_2$ +1)

$= \sum_{K=1}^{r-1} P_K$ (search time of $K$ in $T_1$)
$+ \sum_{K=1}^{n} P_K$
$+ \sum_{K=r+1}^{n} P_K$ (search time of $K$ in $T_2$)

average search time of $T^* = \sum_{K=1}^{n} P_K +$ average search time in $T_1$
+ average search time in $T_2$

$c[1][n]$ : max $\{ \sum_{K=1}^{n} P_K$ $c[1][r-1]$ $c[r+1][n] \}$
⇑
subproblems
for $i \in [1, n+1]$, $j \in [0, n]$
define $c[i][j]$ be the average search time of the optimal BST for key $[i, \dots, j]$ with freq $P_i, \dots P_j$.

$$c[1][n] = \min_{1 \le r \le n} \{ c[1][r-1] + c[r+1, n] + \sum_{K=1}^{n} P_K \}$$

$n$ case in total

Recurrence

$$\begin{cases} c[i][j] = \min_{i \le r \le j} \{ c[i][r-1] + c[r+1][j] + \sum_{K=1}^{j} P_K \} \\ c[i][j] = 0 \quad \text{if } i > j \end{cases}$$

① Optimize : ⊙.

computing $c[\ ][\ ]$

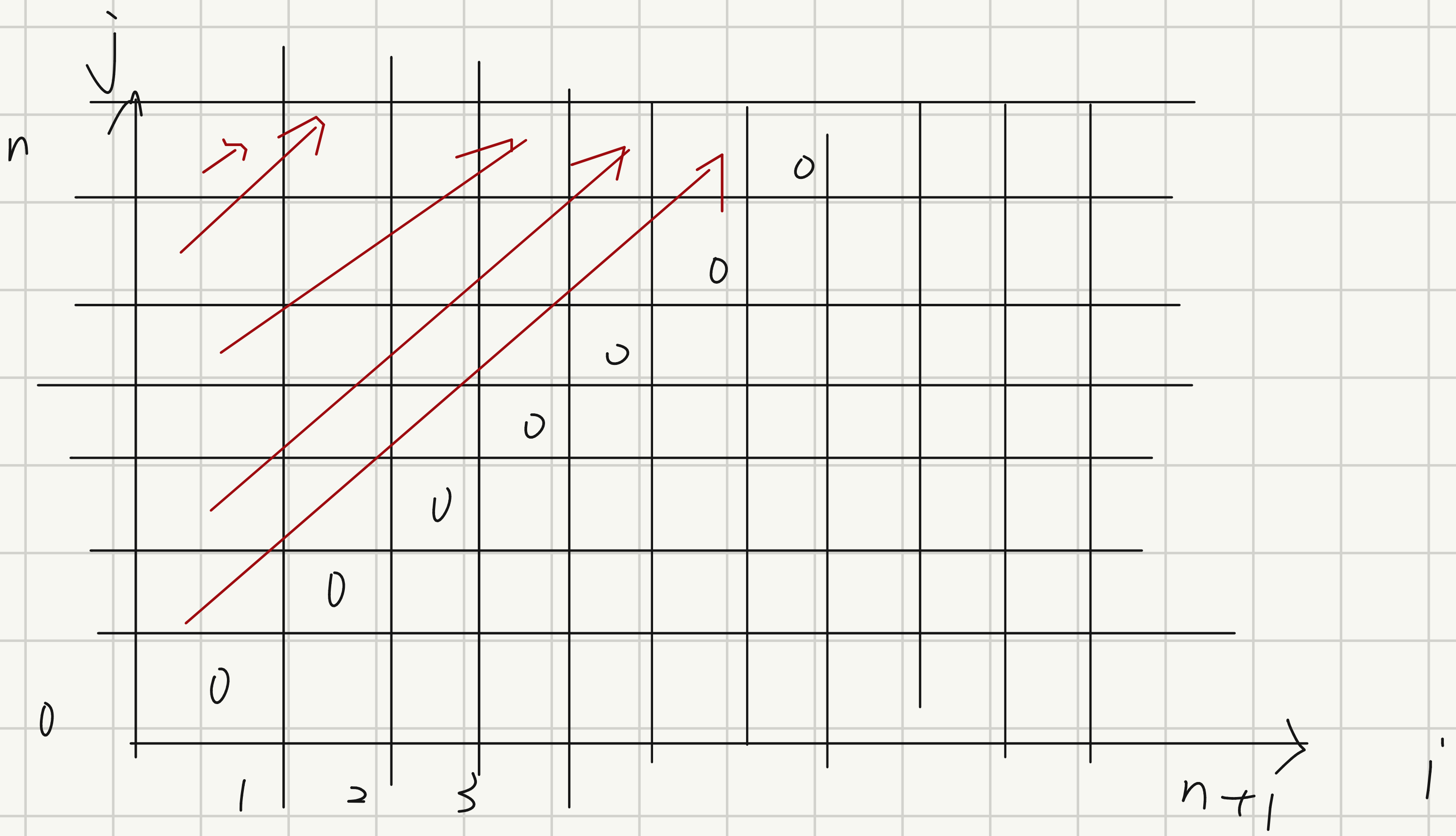
$c[i][i-1] = 0$ for $i=1$ to $n$. $\qquad O(n^3)$

for $t=0$ to $n$.

for $i=1$ to $n-l$.

$\underbrace{c[i][i+l]}_{r[i][i+l]} = \sum_{k=i}^{i+l} p_k + \min_{i \le r \le j} \{c[i][r-1] + c[r+1][i+l]\}$

return $c[1][n]$




Reconstruction:

recur $(i, j)$

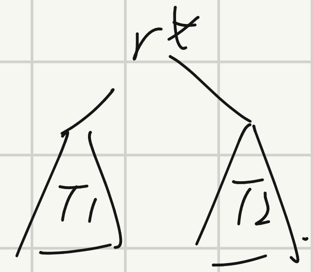
1. if $i = j$, trivial.

2. $r^* = \arg\min_{i \le r \le j} \{ c[i][r-1] + c[r+1][j] + \sum_{k=i}^{j} p_k \}$

3 $T_1 =$ recur $(i, r^*-1)$

4. $T_2 = \text{recur}(r^* + 1, j)$.

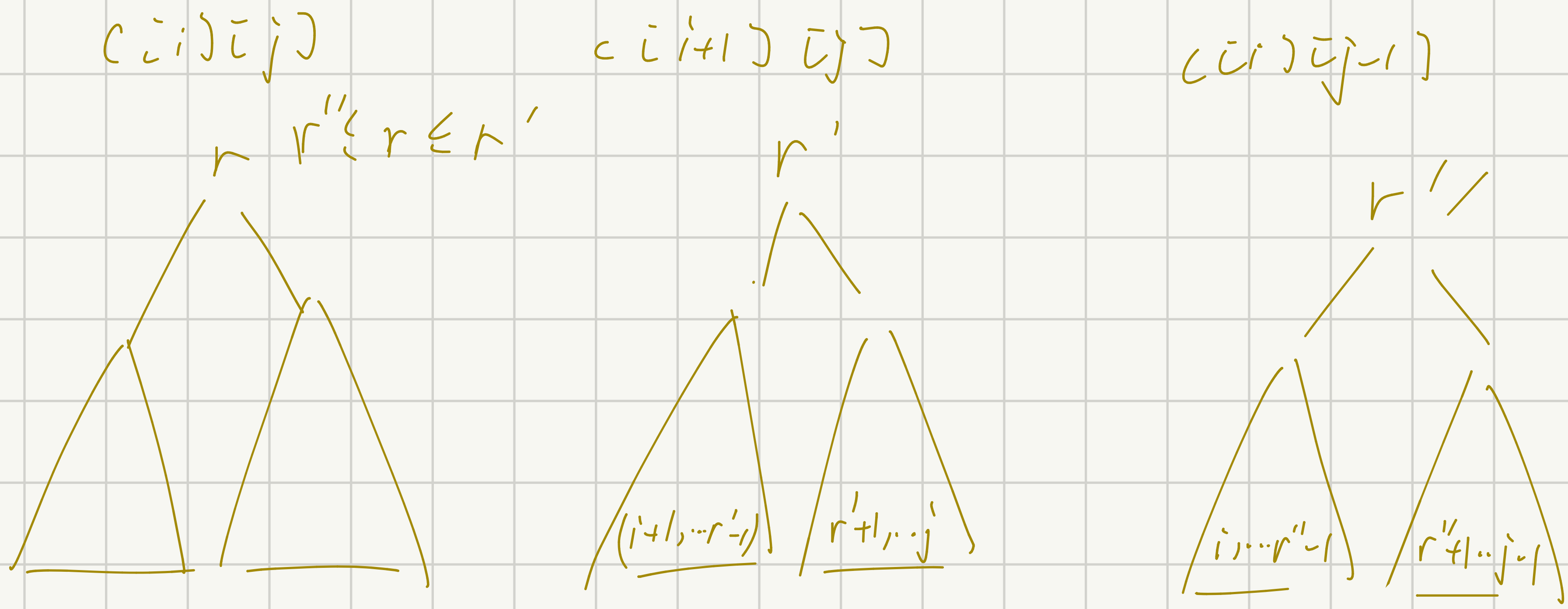
5. return




\# recursive calls : $\Theta(n)$ (each time 1 root, n roots in total)

time for each: $O(n) \to O(1)$

total: $O(n^2) \to O(n)$




now just search from $r''$ to $r'$

in computing

$$M_{a\times b}M_{b\times c} = a\left(\overbrace{\;-\;}^{b}\right) \quad b\left(\,|\,\right)^{c}$$

$$= \underset{\text{each entry}}{b} \times \underset{\text{num}}{(a\times c)}$$

$$= a\times b\times c$$

$$M_{4\times3}\,M_{3\times2}\,M_{2\times1}$$

$$(M_{4\times3}M_{3\times2})\,M_{2\times1} = (4\times3\times2) + (4\times2\times1) = 32.$$

$$(M_{4\times3})\,(M_{3\times2}\,M_{2\times1}) \quad (3\times2\times1) + (4\times3\times1) = 18$$

Input: $M_1,\ M_2.\ M_3\ \cdots\cdots\ M_n.$

$r_0\cdot r_1\quad r_1\cdot r_2\quad r_2\cdot r_3 \qquad r_{n-1}\cdot r_n.$

Output: best order of performing multiplication

$b_i$ = # ways to multiply $i$ matrices

$b_1 = 1$

$b_2 = 1$

$b_3 = 2$

$b_4 = b_3 b_1 \times 2 + b_2 b_2 = 5$

$$b_i = b_1 b_{i-1} + b_2 b_{i-2} + \vdots + b_{i-1} b_1$$

$M_1\ M_2\ M_3\ M_4$

$(M_1 M_2 M_3)(M_4)$ $\quad b_3$

$(M_1 M_2)(M_3 M_4)$ $\quad b_2\cdot b_2$

$(M_1)(M_2 M_3 M_4)$ $\quad b_3$

decide on where to divide (into two pieces)

$(M_1, \ldots, M_K)(M_{K+1} \quad M_n)$

$M_{r_0 \times r_K}$ $\quad$ $M_{r_K \cdot r_n}$

Step 3:

$O(r_0 \times r_K \times r_n)$

Step 1: best of $K$

2: best of $n-K$

$\therefore$ $O(r_0 \times r_K \times r_n)$ + minimum time multiply first $K$ martices

+ minimum time multiply last $n-K$ martices

subproblem:

for $i, j \in [1, n]$

$C[i][j]$ = min cost for perform $M_i \ldots M_{i+1}, \ldots M_j$.

$$C[1][n] = \min_{1 \le K \le n-1} \{ r_0 r_K r_n + C[1][K] + C[K+1][n] \}$$

$$\begin{cases} C[i][j] = \min_{i \le K \le j-1} \{ r_{i-1} r_K \cdot r_j + C[i][K] + C[K][j] \} \\ C[i][i] = 0. \end{cases}$$