# Dynamic Programming

Recap:

- Greedy Alg
  - Greedy choice $\Rightarrow$ show safety
  - Reduce to sub-problem and solve iteratively
  - Focus: correctness.

- Divide-and-Conquer
  - Break into many independant sub-problems
  - Solve each sub-prob separately
  - Combine sols for sub-probs to form a sol for the original one.
  - Focus: efficiency

- Dynamic Programing

  - Break into many overlapping sub-problems
    - Unlike DaC, the sub-probs are usually decremental: only slightly smaller than the original problem.
  - Combine sols for sub-probs to form a sol for the original one
  - Use extra space to store sols of sub-probs for reuse.

## ▷ Recap: DP alg for Fibonacci Number

- $F_0 = F_1 = 1$
- $F_N = F_{N-1} + F_{N-2}$ , $\forall N \geq 2$

```
Fib(N):
    F ← array of length N+1
    F[0], F[1] ← 1.
    for i = 2 to N:
        F[i] = F[i-1] + F[i-2]
    return F[N]
```

▷ sub-problems: $F_{N-1}$, $F_{N-2}$

- overlapping: $F_{N-1}$ contains $F_{N-2}$
- decremental: $F_{N-1}$ & $F_{N-2}$ are very "close" to $F_N$

▷ Extra Space

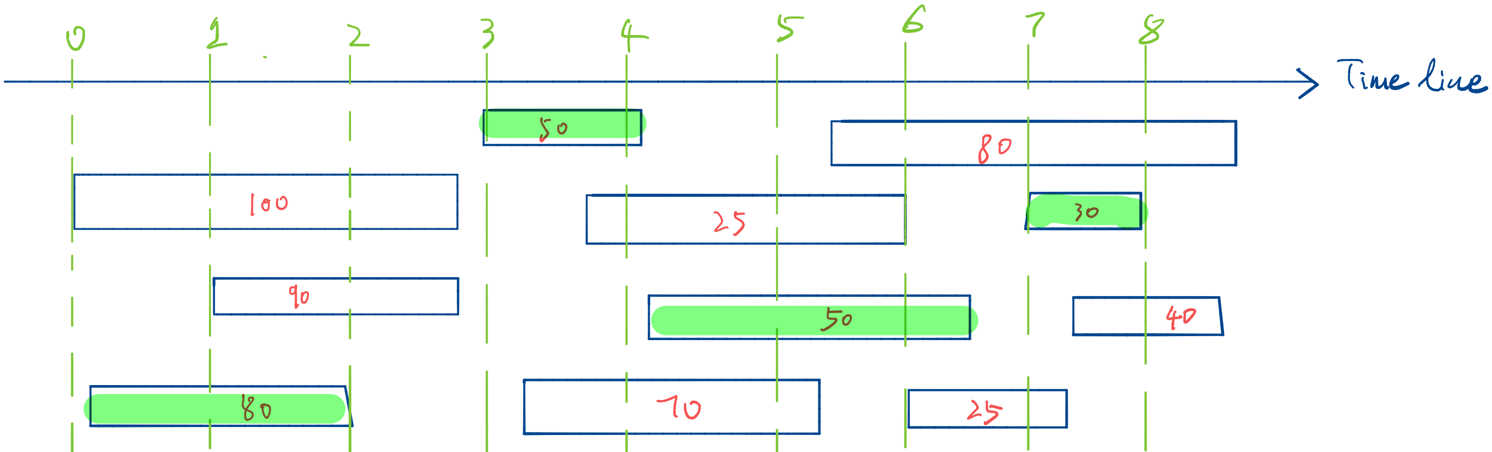
Store $F[i]$ for future use.

# ▷ Exmp I: Weighted Interval Scheduling

**Input:** $n$ jobs, job $i$ with start time $s_i$ & finish time $f_i$

each job has a weight $v_i > 0$

**Output:** A maximum-weight subset of mutually-compatible jobs



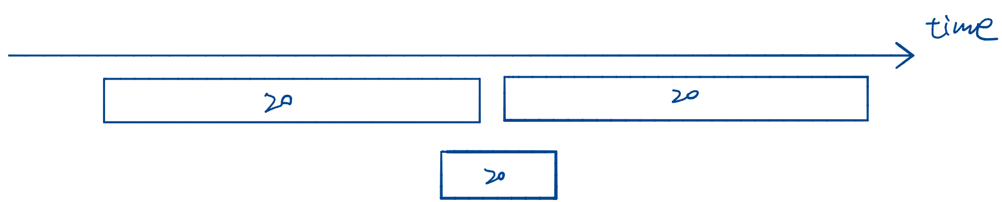

Total wt = 210

▷ Hard to design greedy algorithm

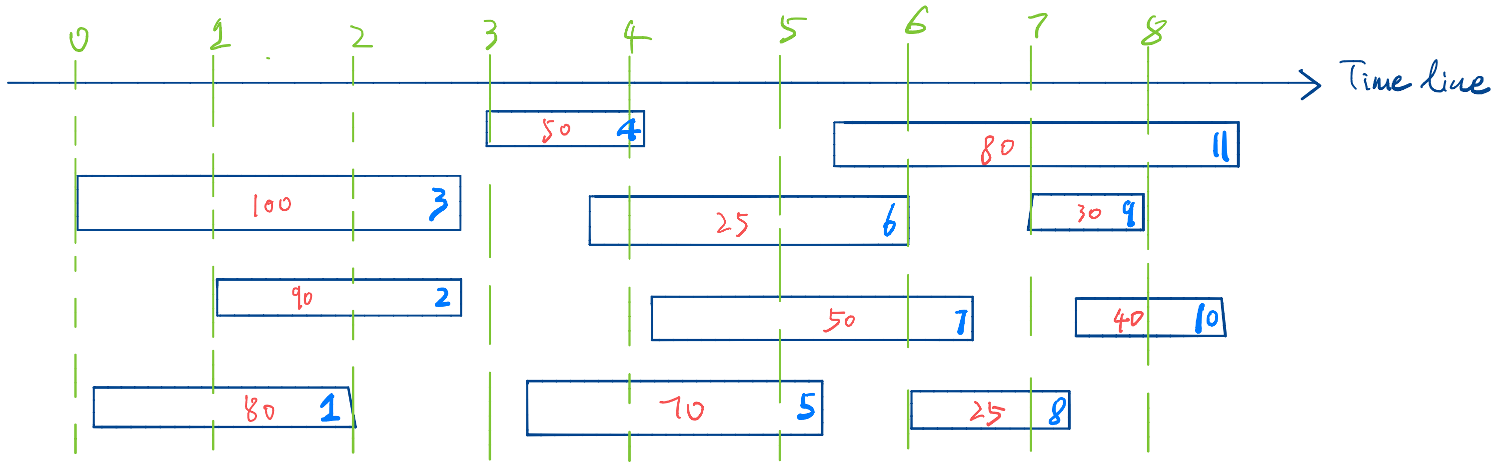
Q: which job is safe to choose?

- Earliest finishing time? No. it ignores weight
- largest weight? No. it ignores times.
- largest $\frac{\text{weight}}{\text{length}}$? No. if all wt are equal, then this is equiv to choosing the shortest job.

Eg:



▷ Dynamic programming




- Numbering jobs by their finishing time

- Let $OPT_i$ := optimal value for the sub-prob that contains only jobs $1, 2, \cdots, i$

▷ Recurrence formula for $OPT_n$

- There're only two possible cases for the optimal sol:
  - Either it does not select the n-th job

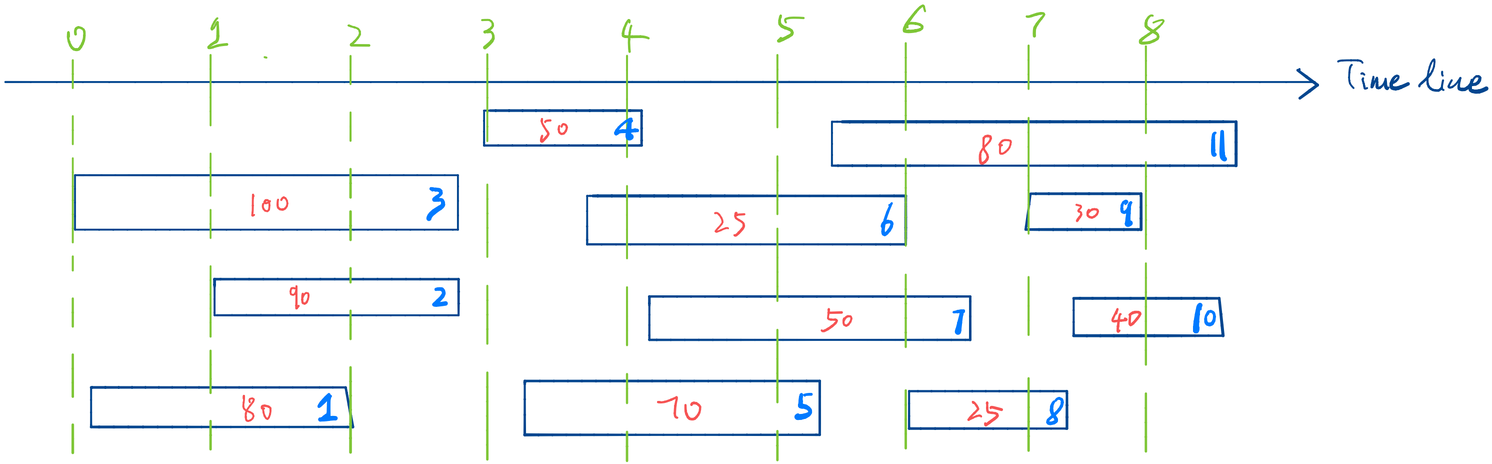
    $\Rightarrow \quad OPT_n = OPT_{n-1}$

  - Or it selects the n-th job

    $\Rightarrow \quad OPT_n = v_n + OPT_{p_n}$

    where $p_n :=$ the largest $j$ such that $f_j \leq s_n$

$$\Rightarrow \quad OPT_n = \max\{OPT_{n-1},\ OPT_{p_n} + v_n\}$$




$OPT_0 = 0, \quad OPT_1 = 80, \quad OPT_2 = 90, \quad OPT_3 = 100$

$OPT_4 = \max\{OPT_3, 50 + OPT_3\} = 150, \quad OPT_5 = \max\{150, 70 + OPT_3\} = 170$

$OPT_6 = \max\{OPT_5, 25 + OPT_3\} = 170, \quad OPT_7 = \max\{OPT_6, 50 + OPT_4\} = 200$

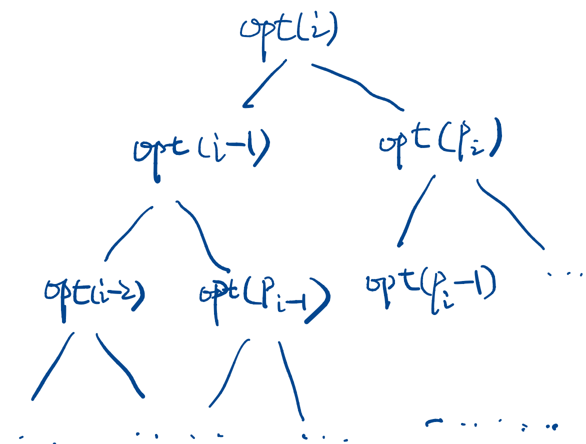
$OPT_8 = \max\{OPT_7, 25 + OPT_6\} = 200, \quad OPT_9 = 230$

$OPT_{10} = 240, \quad OPT_{11} = \max\{OPT_{10}, 80 + OPT_5\} = 250$

## ▷ Implementation

- Top-down: apply recursion directly

```
def opt(i):
    if i = 0:
        return 0
    else:
        return max{opt(i-1), v_i + opt(p_i)}.
```




- Running time $\exp(O(n))$

- Bottom-up: solve smaller sub-probs and store the values.

1. sort jobs by non-decreasing order of finishing times
2. compute $p_1, p_2, \cdots, p_n$
3. $opt[0] \leftarrow 0$
4. for $i \leftarrow 1$ to $n$
5. $\quad opt[i] \leftarrow \max\{opt[i-1], v_i + opt[p_i]\}$

---

- Running time:

①: $O(n \log n)$

②: $n \times O(\log n) = O(n \log n)$

④-⑤: $O(n)$

$T(n) = O(n \log n)$

▷ Recover the optimal schedule.

sort jobs by non-decreasing order of finishing times

compute $p_1, p_2, \cdots, p_n$

$opt[0] \leftarrow 0$

for $i \leftarrow 1$ to $n$

    if $opt[i-1] \geq v_i + opt[p_i]$

        $opt[i] \leftarrow opt[i-1]$

        $b[i] \leftarrow$ N

    else

        $opt[i] \leftarrow v_i + opt[p_i]$

        $b[i] \leftarrow$ Y

$b[i]$: whether to include job $i$ in $OPT_i$

$i \leftarrow n, S \leftarrow \emptyset$

while $i \neq 0$

    if $b[i] =$ N

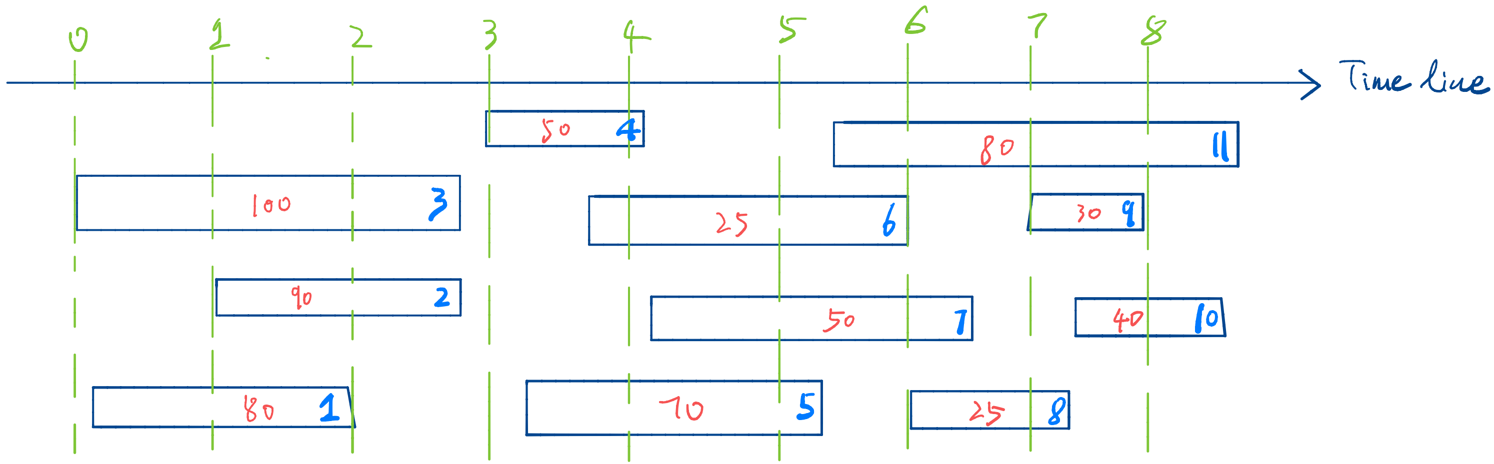
        $i \leftarrow i - 1$

    else

        $S \leftarrow S \cup \{i\}$

        $i \leftarrow p_i$

return $S$




| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| opt[i] | 0 | 80 | 90 | 100 | 150 | 170 | 170 | 200 | 200 | 230 | 240 | 250 |
| b[i] | Y | Y | Y | Y | Y | Y | N | Y | N | Y | Y | Y |

$\Rightarrow S = \{11, 5, 3\}$

# Problem II: Longest Increasing Subseq (LIS)

**Input:** array of $n$ numbers $A = [a_1, \cdots, a_n]$

**Output:** the longest increasing subseq of $A$

Exmp: $A = [2, 1, 4, 3, 5, 7, 6, 8, 0, 10]$

LIS: $[2, 4, 5, 7, 8, 10]$ or

$[1, 3, 5, 6, 8, 10]$ or

......

▷ DP alg for LIS:


- Let $OPT[i]$ := length of the LIS ending at $A[i]$

- Recurrence: $$OPT[i] = 1 + \max_{\substack{j < i \\ A[j] < A[i]}} OPT[j]$$

- Alg:

```
OPT[i] ← 1, ∀i
for i ← 1 to n
    for j ← 1 to i-1
        if A[j] < A[i]
            OPT[i] ← max{OPT[i], 1+OPT[j]}
return OPT
```

length of the LIS

$= \max_i OPT[i]$

▷ Recover the actual LIS

- Let $\pi[i]$ := predecessor of $A[i]$ in the LIS ending at $A[i]$

E.g. $A = [\,2, 1, 4, 3, 5, 7, 6, 11, 0, 18\,]$

index: 1 2 3 4 5 6 7 8 9 10

LIS : $[\,2, 4, 5, 7, 11, 18\,]$

$\pi[10]$ = index of 8 = 8

- Alg: $g \leftarrow \arg\max_i \mathrm{OPT}[i]$

  $\mathrm{LIS} \leftarrow \{g\}$

  while $\pi[g] \neq g$:

  $\quad g \leftarrow \pi[g]$

  $\quad \mathrm{LIS} \leftarrow \mathrm{LIS} \cup \{g\}$

# Problem III: Longest Common Subseq (LCS)

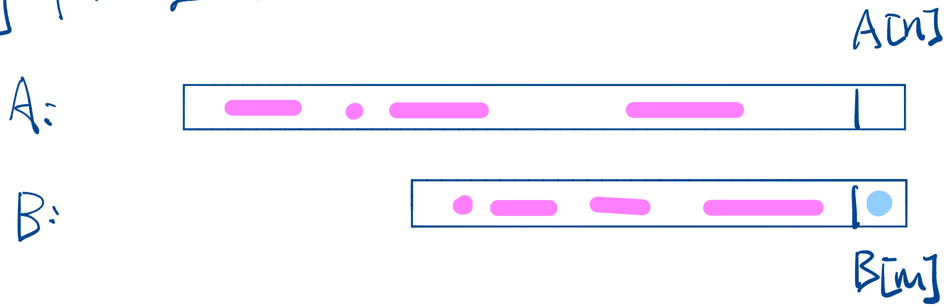
Input: two arrays $A[1 \cdots n]$ and $B[1 \cdots m]$

Output: the longest common subseq of $A$ and $B$

Exmp: $A=$ "bacdca", $B=$ "adbcda"

$\Rightarrow LCS(A, B) =$ "adca"

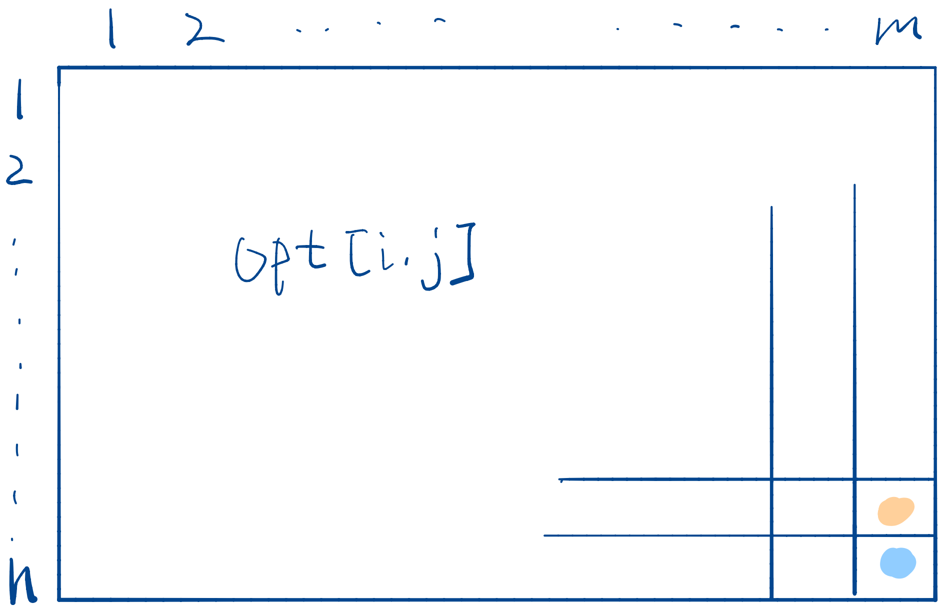
## ▷ DP Alg for LCS.




- $A[n] = B[m]$ : LCS contain $A[n]$ ($B[m]$)
- $A[n] \neq B[m]$ : LCS is one of LCS($A[1 \cdots n-1]$, $B$) and LCS($A$, $B[1 \cdots m-1]$)
- recurrence : Let $opt[i,j]$ denote the length of LCS($A[1 \cdots i]$, $B[1 \cdots j]$)

$$opt[i,j] = \begin{cases} opt[i-1,j-1]+1, & \text{if } A[i]=B[j] \\ \max\{opt[i-1,j],\ opt[i,j-1]\}, & \text{if } A[i]\neq B[j] \end{cases}$$

# ▷ Recover the LCS




Case ①:

$opt[i,j] = 1 + opt[i-1, j-1]$

Case ②

$opt[i,j] = opt[i-1, j]$

Case ③

$opt[i,j] = opt[i, j-1]$

Observation: $A[i]$ ($B[j]$) is in the LCS iff case ① happens for $j \leftarrow 0$ to $m$ do

  $opt[0, j] \leftarrow 0$

for $i \leftarrow 1$ to $n$

  $opt[i, 0] \leftarrow 0$

  for $j \leftarrow 1$ to $m$

    if $A[i] = B[j]$ then

      $opt[i, j] \leftarrow opt[i-1, j-1] + 1, \pi[i, j] \leftarrow$ "$\nwarrow$"

    elseif $opt[i, j-1] \geq opt[i-1, j]$ then

      $opt[i, j] \leftarrow opt[i, j-1], \pi[i, j] \leftarrow$ "$\leftarrow$"

    else

      $opt[i, j] \leftarrow opt[i-1, j], \pi[i, j] \leftarrow$ "$\uparrow$"

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $A$ | b | a | c | d | c | a |
| $B$ | a | d | b | c | d | a |

$\pi$

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | $0 \perp$ | $0 \perp$ | $0 \perp$ | $0 \perp$ | $0 \perp$ | $0 \perp$ | $0 \perp$ |
| 1 | $0 \perp$ | $0 \leftarrow$ | $0 \leftarrow$ | $1 \nwarrow$ | $1 \leftarrow$ | $1 \leftarrow$ | $1 \leftarrow$ |
| 2 | $0 \perp$ | $1 \nwarrow$ | $1 \leftarrow$ | $1 \leftarrow$ | $1 \leftarrow$ | $1 \leftarrow$ | $2 \nwarrow$ |
| 3 | $0 \perp$ | $1 \uparrow$ | $1 \leftarrow$ | $1 \leftarrow$ | $2 \nwarrow$ | $2 \leftarrow$ | $2 \leftarrow$ |
| 4 | $0 \perp$ | $1 \uparrow$ | $2 \nwarrow$ | $2 \leftarrow$ | $2 \leftarrow$ | $3 \nwarrow$ | $3 \leftarrow$ |
| 5 | $0 \perp$ | $1 \uparrow$ | $2 \uparrow$ | $2 \leftarrow$ | $3 \nwarrow$ | $3 \leftarrow$ | $3 \leftarrow$ |
| 6 | $0 \perp$ | $1 \nwarrow$ | $2 \uparrow$ | $2 \leftarrow$ | $3 \uparrow$ | $3 \leftarrow$ | $4 \nwarrow$ |

$\pi[n, m]$

# Prob IV : Subset Sum

Input: $n$ items with weight $w_1, \cdots, w_n$, bound $W$

Output: find $S \subseteq [n]$ s.t. $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} w_i$ is maximized.

Exmp: $W = 35$, $n = 5$, $w = (14, 9, 17, 10, 13)$

OPT = 33 obtained via $S = \{1, 2, 4\}$

1) Greedy alg fails

Candidate greedy alg:

① sort all $w_i$'s in non-decreasing order

② Add items in the sorted order as long as total wt $\le W$

- Counter example: $W = 100$, $n = 3$, $w = (1, 50, 50)$

- What about non-increasing order?

- Counter example: $W = 100$, $n = 3$, $w = (51, 50, 50)$

▷ DP alg for subset sum

$OPT[n, W]$ := opt value for the sub-problem consists of $(w_1, \cdots, w_n)$ with bound $W'$

- $n$th item is selected

$$OPT[n, W] = OPT[n-1, W-w_n] + w_n$$

- $n$th item is not selected.

$$OPT[n, W] = OPT[n-1, W]$$

$$\Rightarrow \quad OPT[n, W] = \begin{cases} \max \left\{ \begin{array}{l} OPT[n-1, W-w_n] + w_n, \\ OPT[n-1, W] \end{array} \right\}, & \text{if } w_n \le W \\ 0, & \text{if } n = 0 \\ OPT[n-1, W], & \text{if } w_n > W \end{cases}$$

for $W' \leftarrow 0$ to $W$

  $opt[0, W'] \leftarrow 0$

for $i \leftarrow 1$ to $n$

  for $W' \leftarrow 0$ to $W$

    $opt[i, W'] \leftarrow opt[i-1, W']$

    if $w_i \leq W'$ and $opt[i-1, W'-w_i] + w_i \geq opt[i, W']$ then

      $opt[i, W'] \leftarrow opt[i-1, W'-w_i] + w_i$

return $opt[n, W]$

$$T(n) = O(nW)$$

- $W$ is part of the input: $O(\log W)$-bits
- Pseudo-polynomial running time.

Ex: recover the actual subset of items selected.

# Prob V: Knapsack

Input: $n$ items with $\begin{cases} \text{weight } w_1, \cdots, w_n, \\ \text{Value } v_1, \cdots, v_n \end{cases}$

weight bound $W$

Output: find $S \subseteq [n]$ s.t. $\sum_{i \in S} w_i \le W$ and

$\sum_{i \in S} v_i$ is maximized.

▷ DP alg for knapsack

$OPT[n, W]$ := opt value for the sub-problem consists of $(w_1, \cdots, w_n)$ with bound $W$

$$OPT[n,W] = \begin{cases} \max \left\{ \begin{array}{l} OPT[n-1, W-w_n] + v_n, \\ OPT[n-1, W] \end{array} \right\}, & \text{if } w_n \leq W \\ 0, & \text{if } n=0 \\ OPT[n-1, W], & \text{if } w_n > W \end{cases}$$

## Prob VI: Shortest path with Negative edge weights

**Input:** $G=(V,E)$, edge cost $w: E \to \mathbb{R}$. (can be negative)

vertex $s \in V$.

**Output:** The shortest path from $s$ to $v$, for every $v \in V$

- For simplicity, Assume all vertices are reachable from $s$.
- Fact:
  - When $\exists$ neg cycle, the shortest path from $s$ to some $v$ can have $-\infty$ path length.
  - Dijkstra's Alg is unable to tell if there $\exists$ neg-cycle and fail to find the $-\infty$ path.

▷ A DP Alg

- For now assume there is no neg-cycles

  ⇒ Any shortest path have $\le n-1$ edges

- Subproblem?

  $opt[i, v] :=$ shortest $(s,v)$-path that uses $\le i$ edges.


  Goal: $opt[n-1, v]$ for every $v$.

- $$opt[i, v] = \min \Big\{ \underbrace{opt[i-1, v]}_{\text{the path uses } < i \text{ edges.}},\ \min_{u:(u,v)\in E} opt[i-1, u] + w(u,v) \Big\}$$

▷ Detecting Neg-Cycle.

- **Fact**: If $\exists$ neg-cycle, then $\exists v \in V$ s.t. $\lim_{i \to \infty} opt[i, v] = -\infty$

- $opt[n, v] = \min \{ opt[n-1, v], \min_{u : (u,v) \in E} opt[n-1, u] + w(u, v) \}$

**Claim**: If $\forall v$ there's $opt[n, v] = opt[n-1, v]$, then there's no neg-cycles in the graph.

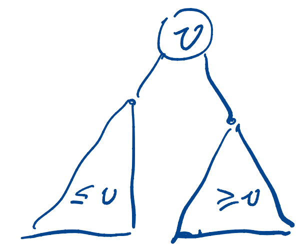
$$opt[n+1, v] = \min \{ \underbrace{opt[n, v]}_{= opt[n-1, v]}, \min_{u : (u,v) \in E} \underbrace{opt[n, u]}_{opt[n-1, u]} + w(u, v) \}$$

$$= opt[n, v] = opt[n-1, v].$$

$$\Rightarrow \forall N > n-1, \quad opt[N, v] = opt[n-1, v]. \ \forall v$$

## Prob VII: Optimum Binary Search Tree (BST)

Def (BST): A binary tree storing numerical values s.t. $\forall$ node with value $v$, all nodes in its left subtree have value $\leq v$, while all nodes in its right subtree have value $\geq v$.

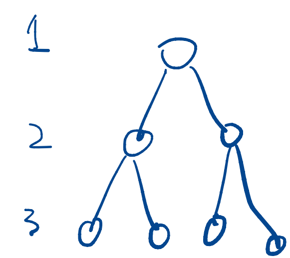



Input: $n$ elements $e_1 < e_2 < \cdots < e_n$

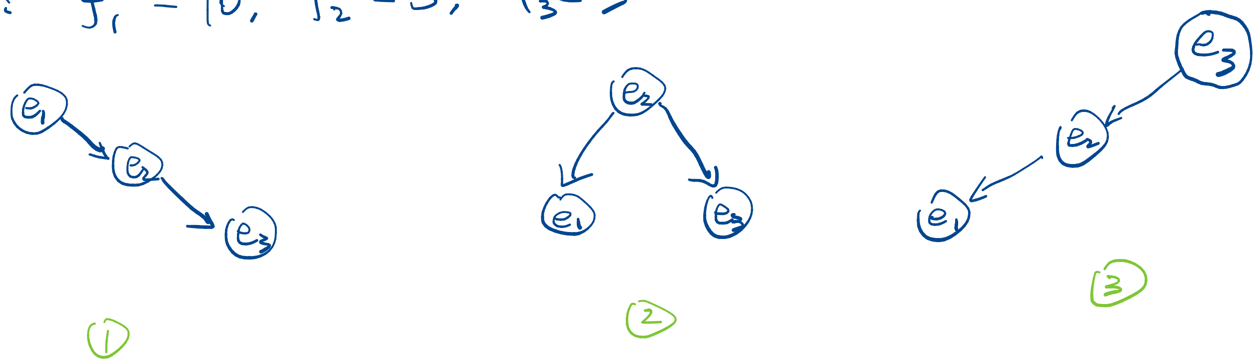
frequency $f_i \in \mathbb{R}_{\geq 0}$ for each $e_i$

Output: A BST for $\{e_1, \ldots, e_n\}$ minimizing

$$\text{cost} = \sum_{i=1}^{n} f_i \times \text{depth}(e_i)$$



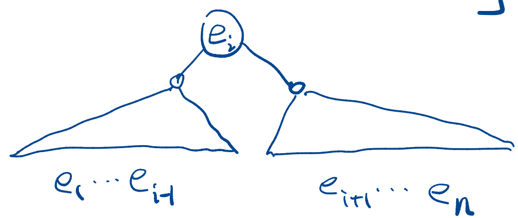
Exmp: $f_1 = 10, \; f_2 = 5, \; f_3 = 3$




- cost 1 $= 10 \times 1 + 5 \times 2 + 3 \times 3 = 29$
- cost 2 $= 10 \times 2 + 5 \times 1 + 3 \times 2 = 31$
- cost 3 $= 10 \times 3 + 5 \times 2 + 3 \times 1 = 43$

▷ DP alg for optimal BST

- Subproblem?
  - Consider the best BST with $e_i$ being the root.




Let $opt[i,j] :=$ optimum BST cost for element $e_i, \cdots, e_j$

$$opt[i,j] = \min_{i \le k \le j} \left\{ opt[i, k-1] + opt[k+1, j] + \sum_{k=i}^{j} f_k \right\}$$

- Goal: $opt[1, n]$.