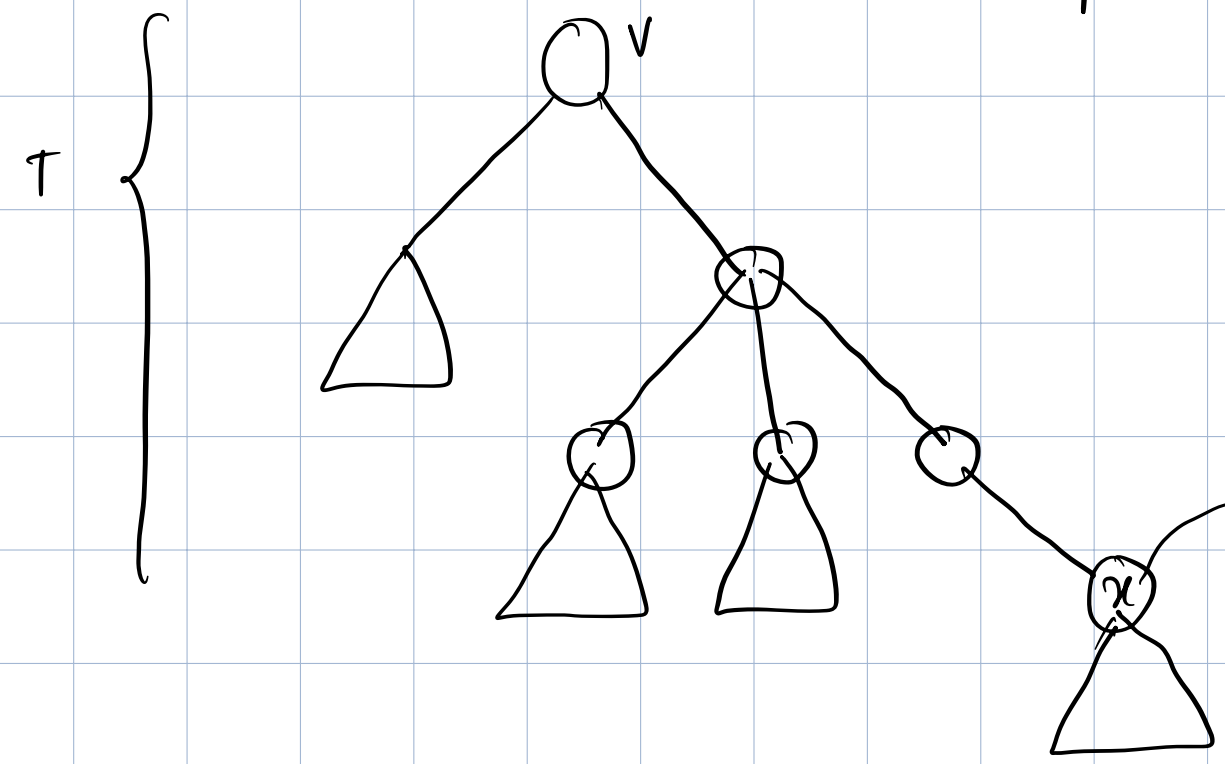
24 Feb 2025 - Algorithms - Week 09

Dynamic Programming : Game trees.

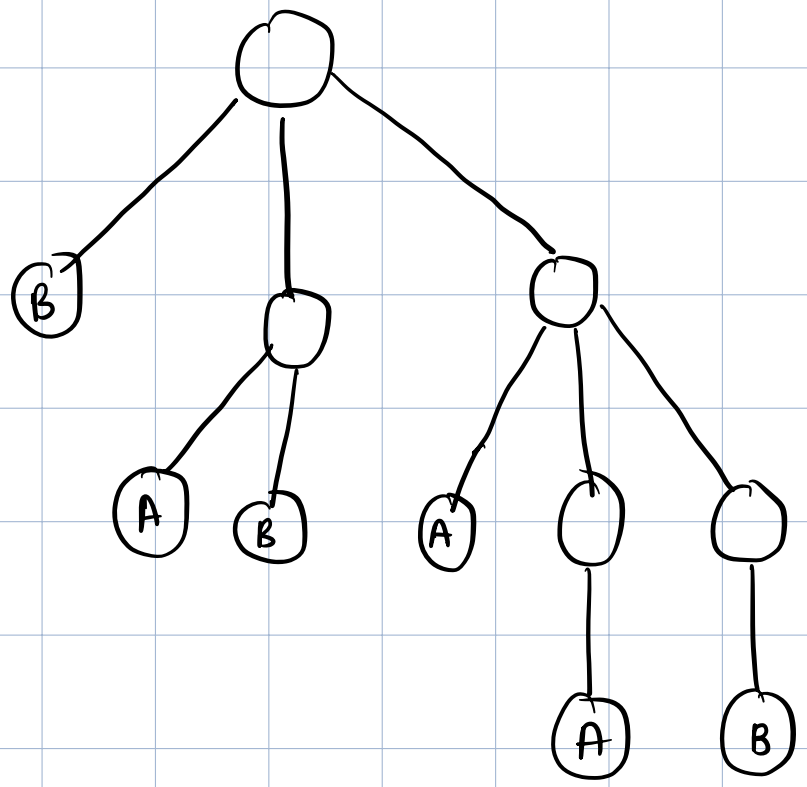
Compute $f(T, v)$



$f(T_x, x)$

$= A$ if A wins the game starting at node x

$= B$ o/w



If x is a leaf,

$$f(x) = l(x)$$

In practice, game trees are too large.

→ Probe the trees to a particular depth

→ Zero sum game (A scoring -4 ≡ B scoring 4)

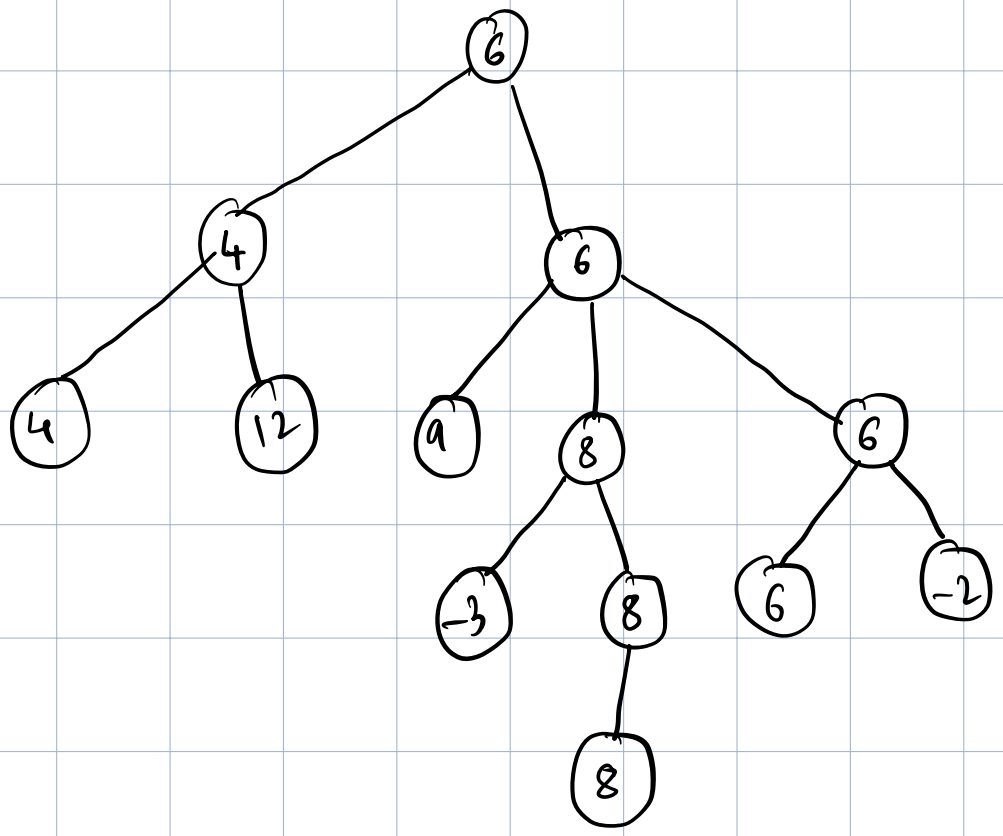
Goal for player A

Goal for player B

→ Max - min problem

$f(x)$ = Max score that A can obtain starting at node x.

Recurrence for $f(x)$



A

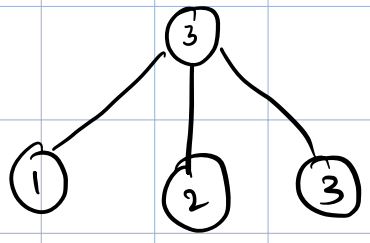
B

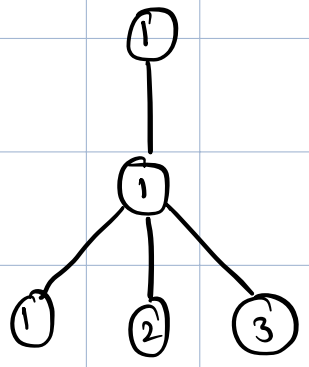
A

B

A

height



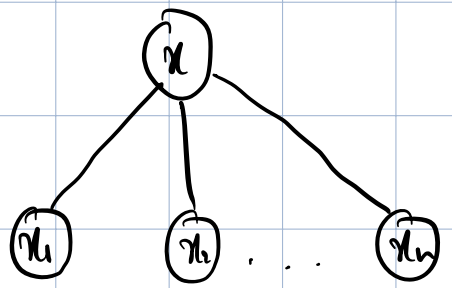


A

B

L

$$f(x) = \begin{cases} \max\{f(x_1), \ldots f(x_n)\} & \text{if A's turn at } x \; \{ h(x)\%2 == 0 \\ \min\{f(x_1), \ldots, f(x_2)\} & \text{if B's turn at } x \; \{ h(x)\%2 \neq 0 \end{cases}$$



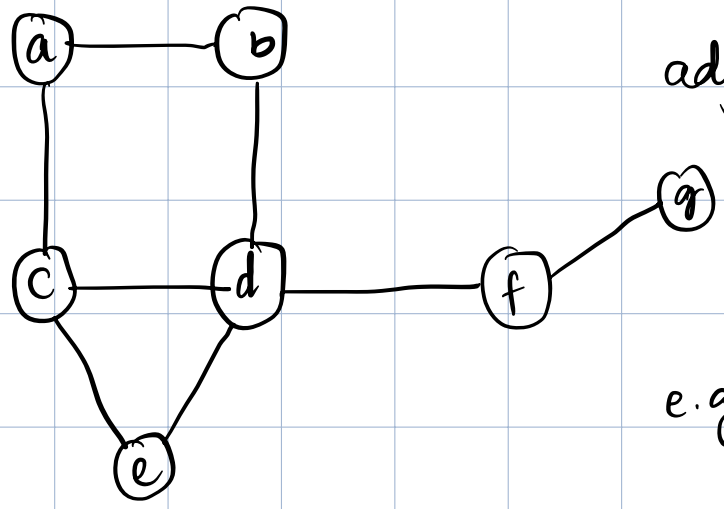
A's turn :

$$f(x) = \max_{y \text{ child of } x} f(y)$$

B's turn :

$$f(x) = \min_{y \text{ child of } x} f(y)$$

Independent Set Problem



→ No vertex in the set should be adjacent (naturally occur in scheduling problems) how?

e.g : $\{a, d, g\}$

$\{a, d\}$

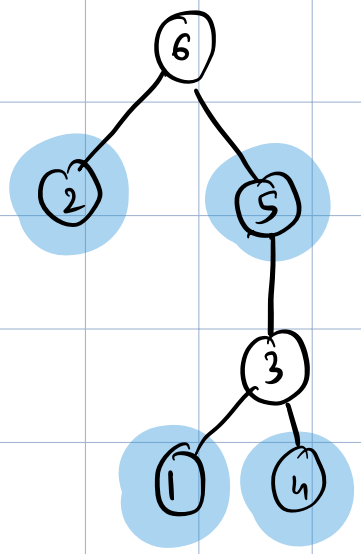
For a given graph G, find independent set of maximum possible size.

NP - hard : No efficient algorithm known (not even for approximation)

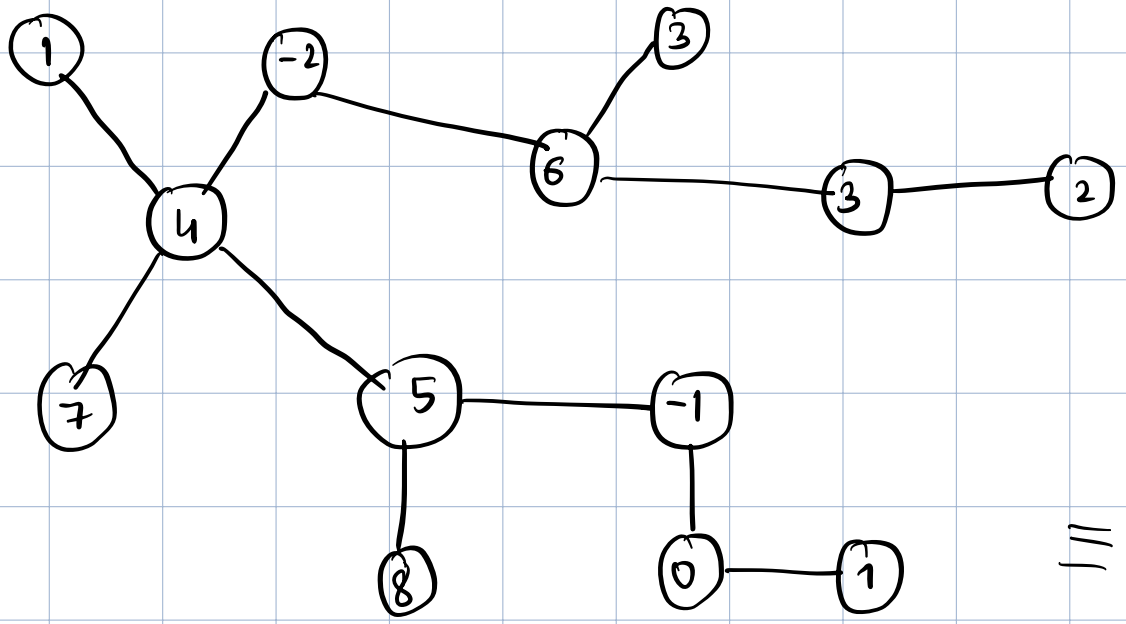
Some algorithms : subexponential / subpolynomial $O(2^{\sqrt{n}})$

If the input graph is "simple" such as a tree, there are efficient algorithms

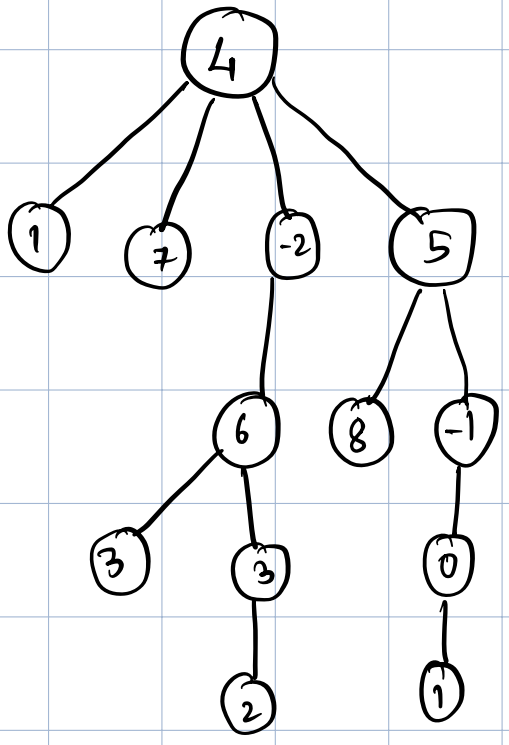
Maximum Weight Independent Set



→ We can find maximum weight independent set in $O(n)$ time.

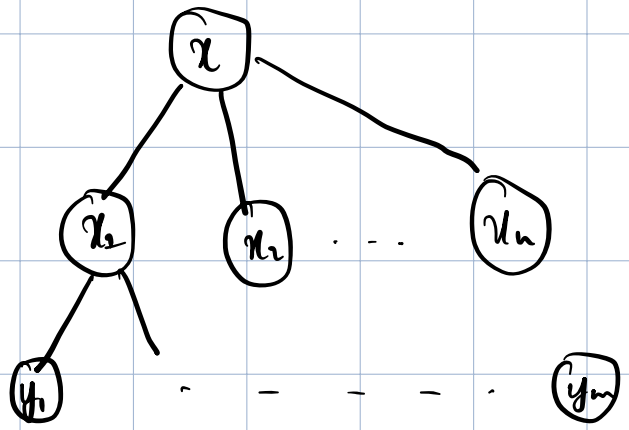


≡

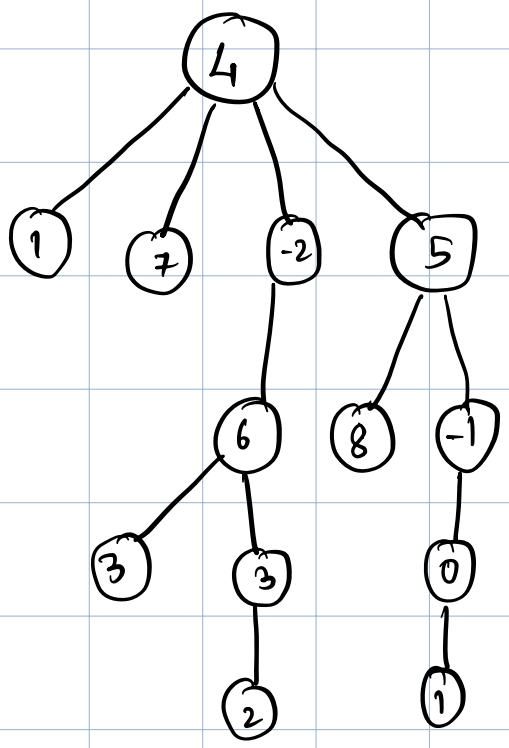


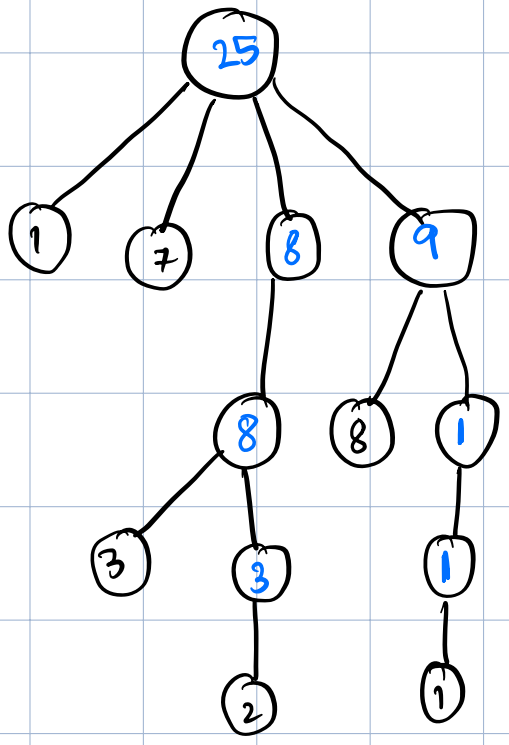
Subproblem:

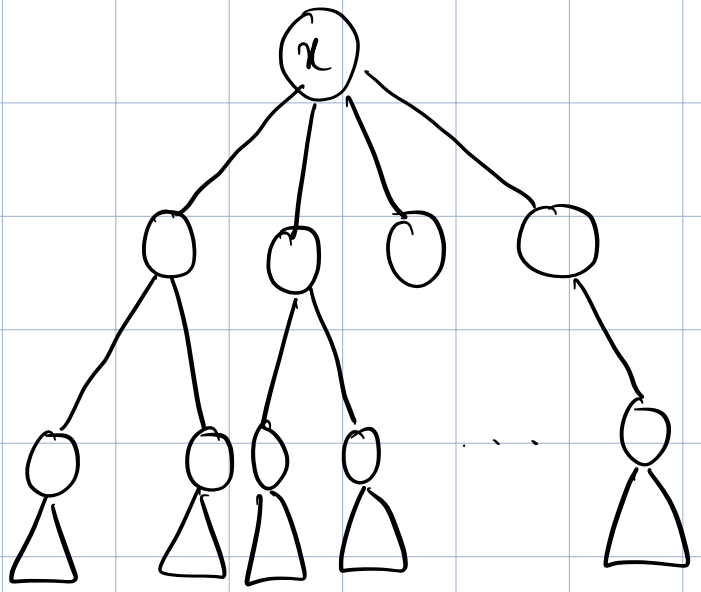
$f(x)$ = max independent weight at node x



$$f(x) = \max \{ x + f(y_1) + f(y_2) + \cdots + f(y_m),$$
$$f(x_2) + f(x_1) + \cdots + f(x_n) \}$$


4
1
7
-2
5
6
8
-1
3
3
0
2
1


25
1
7
8
9
8
8
1
3
3
1
2
1



→ For leaf with negative weight, return empty set

$$f(x) = \left[\max \left(\sum_{\substack{z \text{ grand} \\ \text{child}}} f(z) + w(x), \sum_{\substack{y \text{ child} \\ \text{of } x}} f(y) \right) \right]$$

Every node is checked 3 times

- as child
- as grandchild
- itself

∴ Time complexity = $O(n)$

27 Feb 2025

Greedy Algorithms : Activity Selection

Greedy algorithm: successively make greedy choices (locally optimal choices)

Activity selection

i	1	2	3	4	5	6	7	8	9	10	11
s(i)	1	3	0	5	3	5	6	8	8	2	12
f(i)	4	5	6	7	8	9	10	11	12	13	14

} → increasing order

Input : A set $S = \{a_1, \ldots, a_n\}$ of activities $(s(i), f(i))$

Output :

→ Using DP → see slides

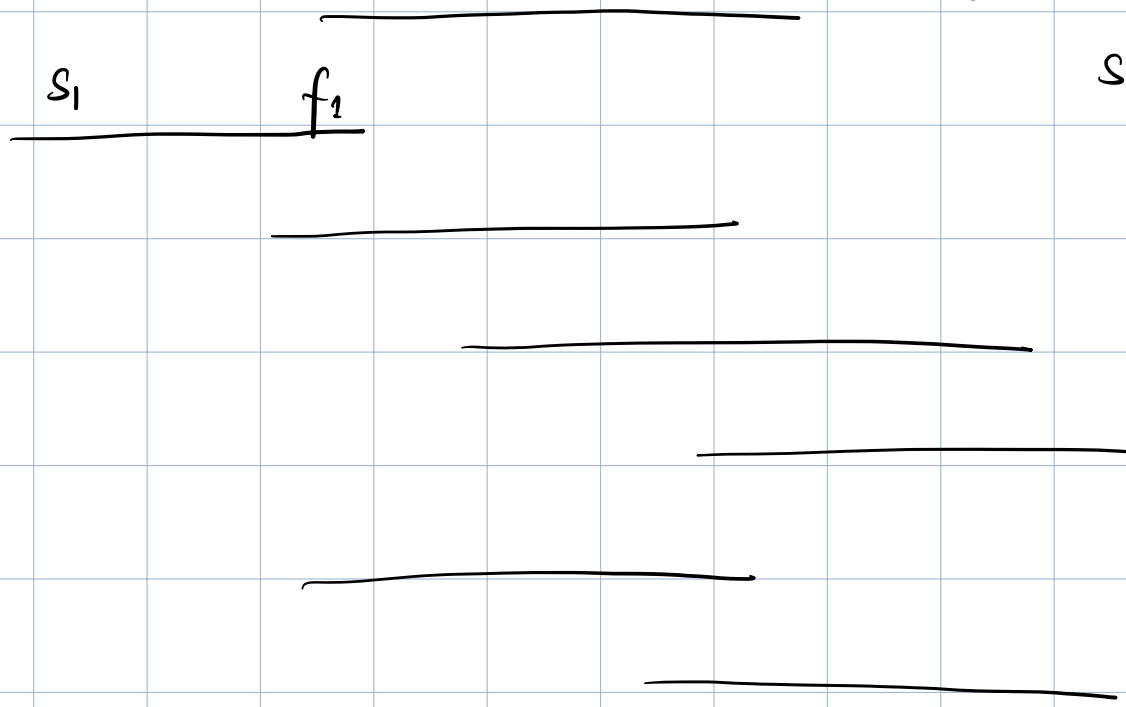
→ Using greedy approach :

1. Sort activities wrt finish times } $O(n \log n)$
2. $S = \{a_1\}$
3. $k = 1$
4. For $i = 2$ to n
   if $s_i > f_k$ add a_i to S
   $k = i$

   } $O(n)$
5. Return S

Correctness : Most common idea → exchange argument

Let S be an optimal solution



$a_1 \notin S$ → optimal

S : [a_1]

s_{i_1} f_{i_1} s_{i_2} f_{i_2} . . . .

Let i_1 be the activity with least finish time in S.

$S \setminus \{i_1\} \cup \{a_1\}$ is a mutually compatible set

Size has not changed.

$\therefore$ there is some optimal solution that contains a_1

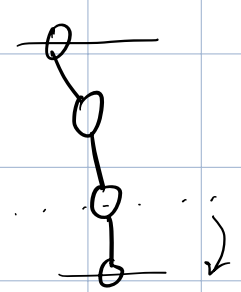
$\Rightarrow a_1$ was safe to pick.

Exchange argument : Proof sketch (see slides)

Problem 1: Maximum Independent Set on Trees

Given a tree T as input, design a greedy algorithm to find the maximum independent set

problem 2: Knapsack, 0/1 Knapsack, fractional Knapsack

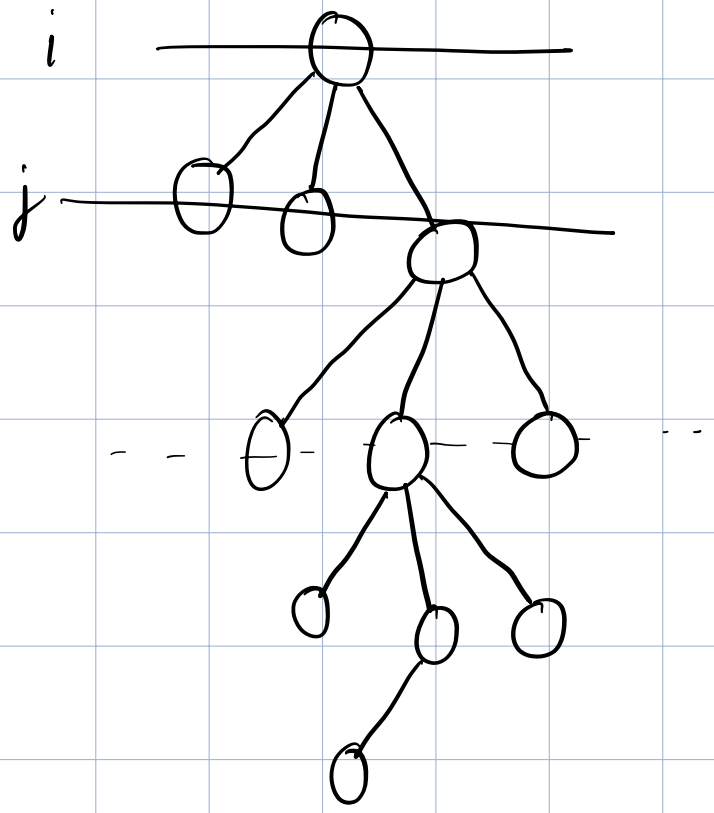


$i = 1 \quad j = 2$

For each level i, j

if $w(i) > w(j)$

$j++$



??

Hint: Find a safe choice

1) Sort vertices in decreasing order of weights.

2) Find two largest vertices that are adjacent : V_1 , V_2

3) $S_1 = V_1$; $S_2 = V_2$

4) For all vertices $(\neq V_1, V_2)$

if vertex not adjacent to V_1

$S_1 \mathrel{+}= w(\text{vertex})$

if vertex not adjacent to V_2

$S_2 \mathrel{+}= w(\text{vertex})$

5) Return $\max\{S_1, S_2\}$

1) Add all leaves

1) Sort items in decreasing order of price / kg.

2) i = 1 ; total weight = 0

3) While total weight < W

    add sorted item i to bag ;

    total weight += weight of sorted item i

    i++;

4) total weight = total weight − weight of sorted item [i − 1]

5) If total weight < W

    add (W − total weight) kg of item (i − 1)

    to W.

Correctness: Suppose the optimal solution did not contain sorted item 1. Replace the item(s) with highest price / kg with sorted item 1.

item(s) → such that sum of weights = weight of sorted item 1

In that case, sorted item 1 will have contribute a higher price, which contradicts that the solution is optimal.