18/09/24 **DAA**

Hashmap :- $(k_1, v_1), (k_2, v_2), \dots\dots (k_n, v_n)$.

Hashmap is useful to find (k) in $O(1)$ time.

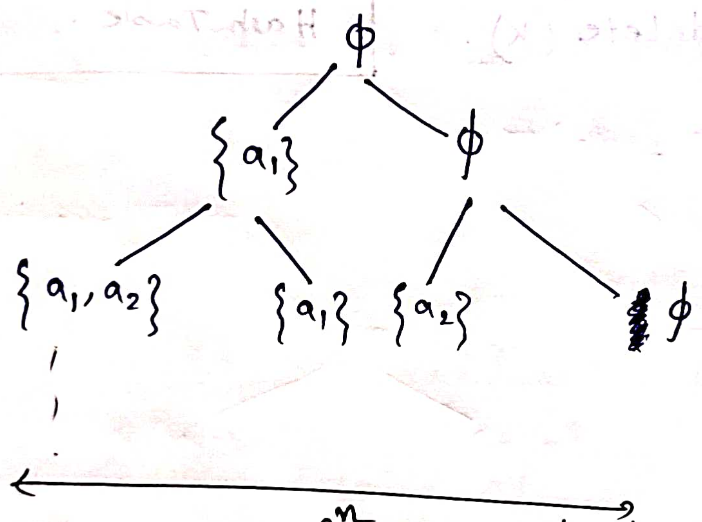
Dynamic Programming :→

(Recursion + Memoization).

(I) Generate all the subsets (subsequence).

a_1	a_2	- - - - -	a_n

- At every step we decide whether to include or exclude an item.



2^n many subset in leaf level.

(II) **Subset sum problem** :-

Input :-

a_1	a_2	- - - - -	a_n

T = Target sum

(Given a sequence).

Goal :- Does there exist a subset S, st. $\sum_{i \in S} a_i = T$

HW Recursively generate all subsets . .

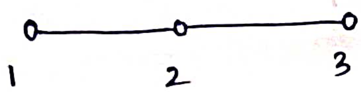
Fm

Brute force:- Generate all the subsets; and then check whether that is a correct solution.

(III) Independent Set:-

$G = (V, E)$ $S \subseteq V$

such that $u, v \in S$; $u, v \notin E$.



like $(1, 3) \in S$ as there is no edge.

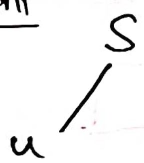
$W : V \rightarrow \mathbb{Z}$ (weighted independent set).

[NB- Weights are in the vertices].

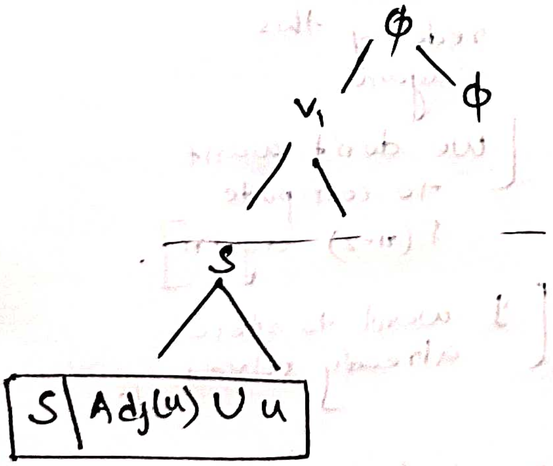
Goal:- Generate maximum independent set.

Brute force:- Generate all vertices set & check whether its independent set & then find max among them.

Efficient



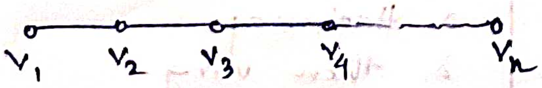
if u is adjacent to any of the vertices in S. $Adj(u)$ .. then try removing all vertices adjacent to S.



- Later using **Reduction**; we can tell that it can't be done using polynomial time.

Path Graph

In path graph using DP it can be done easily.



Fibonacci Sequence :-

Prob:- Given n, find the n^{th} number in the fibonacci seq.

Goal:-

$\boxed{\text{Step-1}}$ — Identify the subproblems.

I want to find F_i ; $i \leq n$.

$\boxed{\text{Step-2}}$ — Relate the subproblems.

$$\boxed{F_i = F_{i-1} + F_{i-2}}$$

$\boxed{\text{Step-3}}$ — $\boxed{\text{Base-Case}}$ $\Rightarrow$ $F_0 = 0$, $F_1 = 1$

$\boxed{\text{Step 4}}$ $\Rightarrow$ Solve for F_n.

$$T(n) = T(n-1) + T(n-2) + 1$$

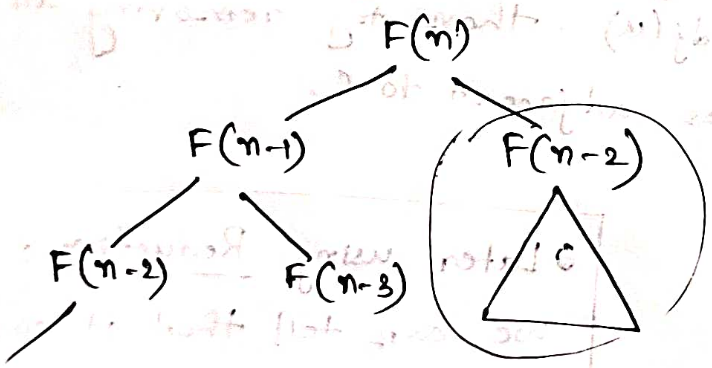
$$\geq \phi^n$$

$$\boxed{1.5 \leq \phi \leq 2}$$

$\Downarrow$

Golden ratio.

Recursive Tree



→ I am redoing this again.

[We don't want to compute F(n-2) again].

[I want to store already solved ...].

Memoization

Running time

= # subproblems × time you need to solve each subproblem.

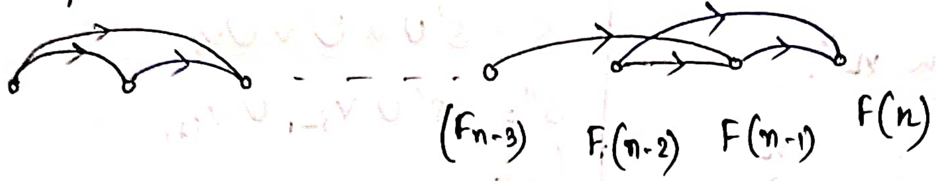
= $n \times$ constant time.

$\approx O(n)$.

We can store in a Hashmap... & then using $O(1)$ we can get...

Recursion + Memoization = DP

Subproblem Dependency graph :→

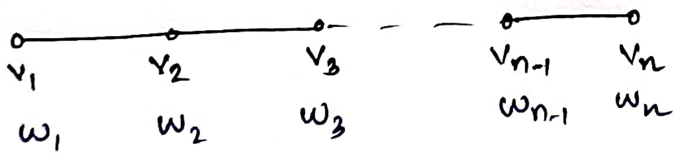


If we want to avoid the computation then we want to just write loop to maintain the table.

```
int F[n];
F[0] = 0
F[1] = 1
for i = 2 to n :
    F[i] = F[i-1] + F[i-2].
```

P2 Weighted Independent Set on a path graph :→

"# of subproblems" need to be polynomially bounded otherwise T(n) won't also be polynomially bounded.



[Each vertex has some associated weights].

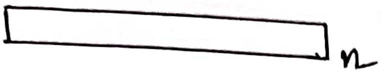
Step-1 → Identify the sub-problem.

Subproblem → WIS $v_1, \ldots, v_i$ $\boxed{i \leq n}$

[Taking the prefixes].

Step-2 → $v_1, v_2, \ldots, v_i, v_{i+1}$

P3 Rod cutting problem



"Rod of length n".

0	1	2	3	...	n
0	P_1	P_2	P_3	...	P_n

Say $n=7$

1	2	3	4	5	6	7
1	10	13	18	20	31	32

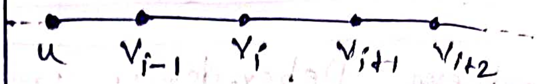
Brute

Generate all partitions of $n=7$.

& check for maximum profit.

(i) $\max\{P_i + R(n-i)\}$

Greedy



$S = S' \cup u \cup v \cup v_{i-} \ldots$

$S'' = S' \cup v_{i-1} \cup v_{i+1} \ldots$

↳ These can be larger, my claim.