CS & IT ENGINEERING

Algorithm

Dynamic Programming

Lecture No.- 01

By- Dr. Khaleel Khan sir

Recap of Previous Lecture

Topic Single Source Shortest Paths

Topic Problem Solving

Introduction to Dynamic Programming

Topics to be Covered

Topic — Elements of Dynamic Programming

Topic — DP vs Divide & Conquer

Multistage Graphs

Topic : Dynamic Programming: (DP)

Dynamic programming (DP) is an algorithm design method used for solving problems, whose solutions are viewed as a result of making a set/sequence of decisions;

- One way of making these decisions is to make them one at a time in a step-wise (sequential) step-by-step manner and never make an erroneous decision. This is true of all problems solvable by Greedy method.
- For many other problems it is not possible to make step-wise decisions based on local information available at every step, In such a manner that the sequence of decision made is optimal.

Topic : Dynamic Programming: (DP)

- 1) One way of solving problem in which it is not possible to make sequence of decisions in step-wise manner leading to optimal solutions is to enumerate all decision sequences (Brute force) & then pick up the best solution (optimal).
- But the drawback with brute force/ Enumeration is excessive Time/Space requirements.

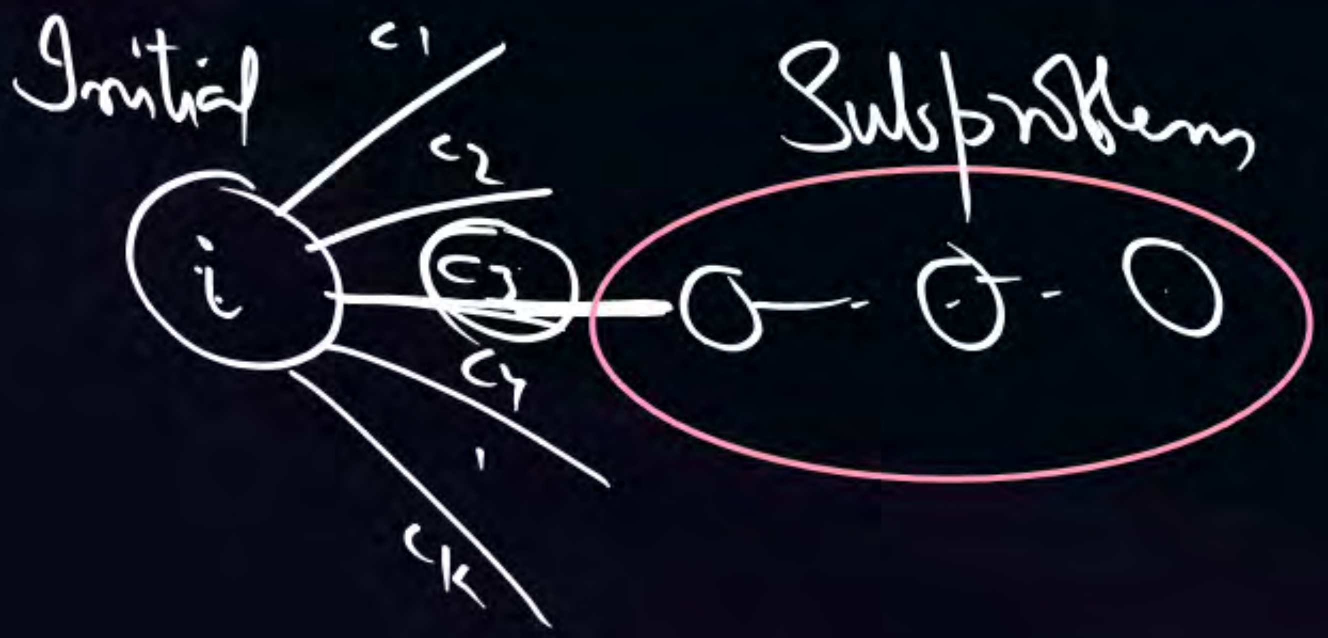
Cut Down | remove

- 2) **Dynamic Programming (DP)** based on Enumeration often tries to reduce the amount of enumeration by curtailing those decision sequence from where these is no possibility of getting optimal solution. (That's how it may bring down the time complexity)
- In **Dynamic programming** these set of optimal decisions are made by applying Principle of Optimality. Sahni | Optimal Substructure Property

(Global optimality)

- **Principle of Optimality**: states that whatever the initial state & decision are the remanning sequence of decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

- Essential difference between **Greedy method & Dynamic programming (DP)** is that Greedy method always generates only one decision sequence. Whereas in DP enumeration many decision sequences can be generated.



Topic : Dynamic Programming: (DP)

- 3) Another important feature of D.P is that optimal solutions of the sub problemss are retained (cached/ stored in a table) to avoid recomputing their values. (Invariably this feature also leads to saving of time) Memoization / Tabulation

1) D.P implementation
   1) Memoization (Top-Down Approach) Recursive
   2) Tabulation (Bottom – Up) Iterative

Topic : The Elements/Properties of D.P

(i) Splitting of original problems into Subproblems: Be able to split the original problem into subproblems in a recursive manner (so that the Subproblems can be further divided into sub-subproblems). This process of splitting should continue till the Subproblems becomes small.

(ii) Subproblems Optimality (Optimal Substructure): An optimal solution to the problem must result from optimal solutions to the subproblems with combine operation.

(iii) Overlapping Subproblems : Many subproblems themselves contain common sub-subproblems. Therefore, it is desirable to solve the small problem & store/ cache their results, so that they can be used in other subproblems.

Topic : Dynamic Programming: (DP)

Ex- Fibonacci Number:- n^{th}:

Fib series : 0,1,1,2,3,5,8,13,21,34..........

Fib(n) = Fib (n–1) + Fib (n–2) , n > 1

Fib(n) =1, n = 1

Fib (n) = 0, n = 0

Computing n^{th} Fibonacci No:

$$F(3) = \overset{1}{F(2)} + \overset{1}{F(1)} = 2$$

$$F(2) = F(1) + F(0)$$
$$= 1 + 0$$
$$= 1$$

Topic : Dynamic Programming: (DP)

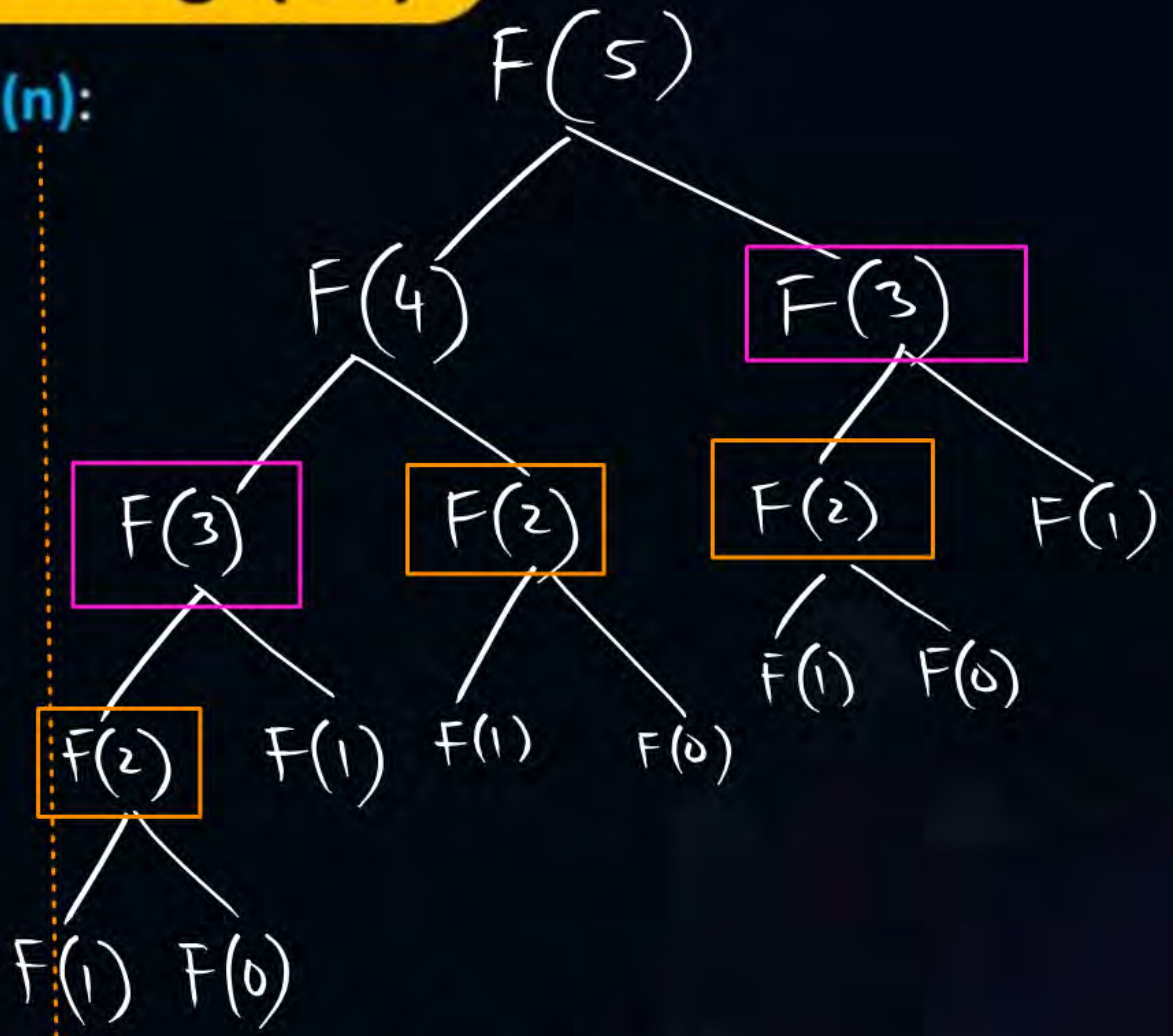
Normal Recursive Implementation of Fib (n):

MemFib

```
Algo Fib (int n )
{
    If (n ≤ 1)  return (n);
    else
    {
        return ( Fib (n–1)+ Fib (n–2) );
    }
}
```

Subproblems

$$T(n) = c\,,\quad n \le 1$$
$$= T(n-1) + T(n-2) + a\,,\quad n > 1$$



Tree of Calls

$$T(n) = T(n-1) + T(n-2) + a \quad — \quad (1)$$

$$\boxed{T(n-2) < T(n-1)}$$

$$\therefore \; T(n) < T(n-1) + T(n-1) + a$$

$$\boxed{T(n) < 2 \cdot T(n-1) + a \quad — \quad (2)}$$

$$T(n) < 2^n$$

$$\boxed{\therefore T(n) = O(2^n)}$$

1) Top-down – Memoized implementation of Fib(n)

$m[0 \ldots n]$

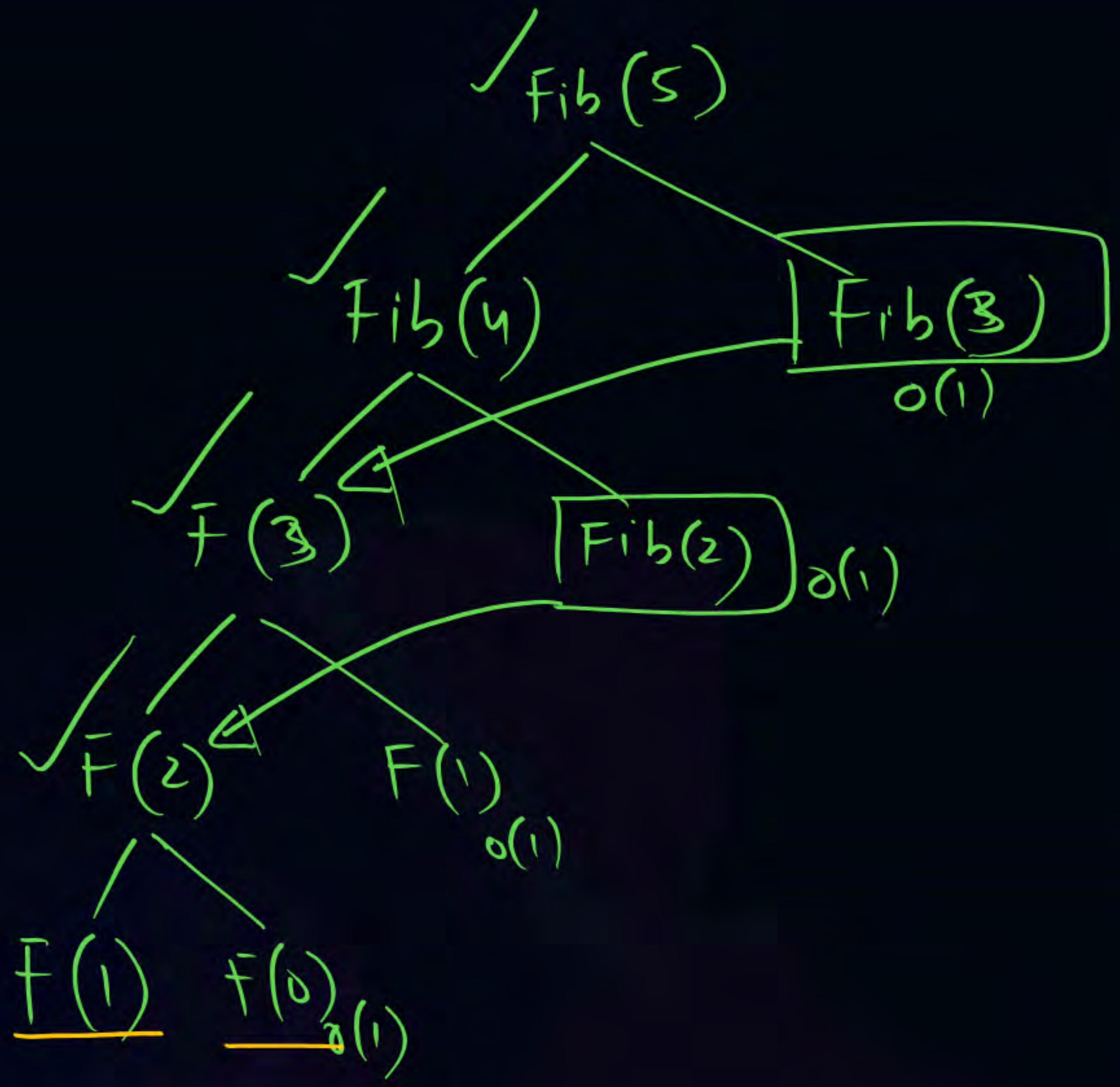
```
Algo memiFb(n)
{
        if (m [n] is undefined)
        {
                if (n < =1) result = n;
                else
                result = memFib (n–1) + memFib(n–2);
            m[n] = result ; //memorizing (caching)
        }
        return (m[n]) ;
}
```

Time : $O(n)$

Recursion Stack : $O(n)$

	0	1	2	3	4	5
m	0	1	1	2	3	5

memFib(5) res = 3 + 2 = 5
- → memFib(4) res = 3
  - → memFib(3) res = 2 + 1
    - → memFib(2) res = 1
      - → memFib(1) = 1
      - → memFib(0) = 0
    - → memFib(1) = 1
  - → memFib(2) = 1
- → memFib(3) = 2



Fib(5)
Fib(4)
Fib(3)
O(1)
F(3)
Fib(2)
O(1)
F(2)
F(1)
O(1)
F(1)
F(0)
O(1)

Topic : Dynamic Programming: (DP)

2) Bottom-up approach of D.P for Fib (n) ✓

M[0 .. n]

```
Algo memFib (n)
{
    M [0] = 0
    M [1] = 1
    for i = 2 to n
    {
        M [i] = M [i –1] + M [i – 2];
    }
    return (M[n]);
}
```

m[2] + m[1]

3 1 + 1

Result

	0	1	2	3	4	5
M	0	1	1	2	3	5

Time : O(n)

Topic : Dynamic Programming: (DP)

When to use Tabulation & when Memoization:

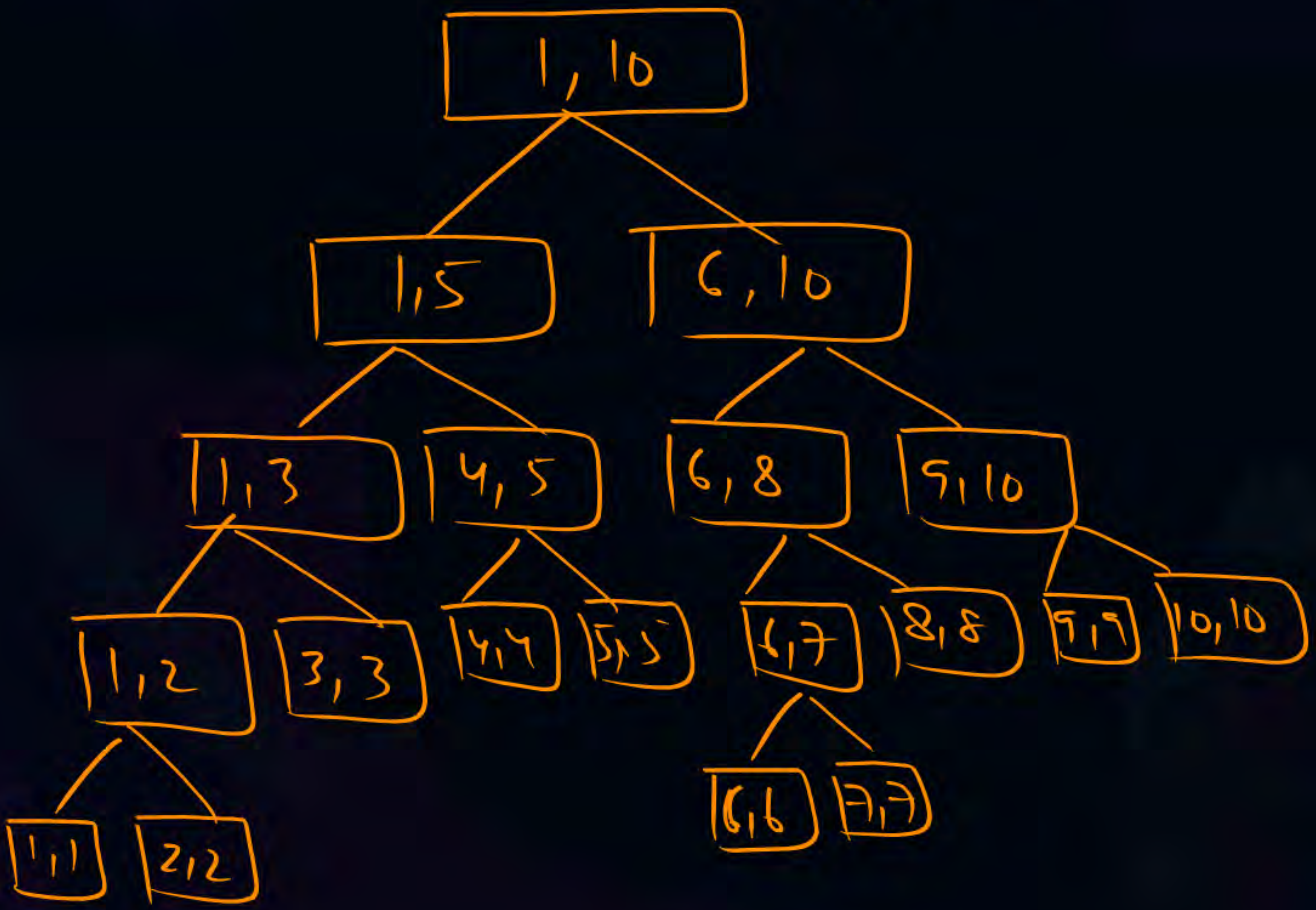
- If the original problem requires all subproblems to be solved, Tabulation usually outperforms Memoization.
- Tabulation has no overhead of Recursion & can use preallotted array.
- If only some of the subproblems needs to be solved in the original problem, then Memoization technique is preferable because the subproblems are solved lazily.

Topic : Dynamic Programming: (DP)

Dynamic Programming vs Greedy Method vs Divide & Conquer:

- In all methods the problem is divided into subproblem;
- Greedy Method: Building up of the solution to the problem is done in a step-wise manner (incrementally) by applying local options only (local optimality).
- Divide & conquer: Breaking up a problem into separate problems (independent), then solve each subproblem separately (i.e. independently) & combine the solution of subproblems to get the solution of original problem.
- Dynamic Programming: Breaking up of a problem into a series of overlapping subproblems & building up solution of larger & larger subproblems.

Merge sort

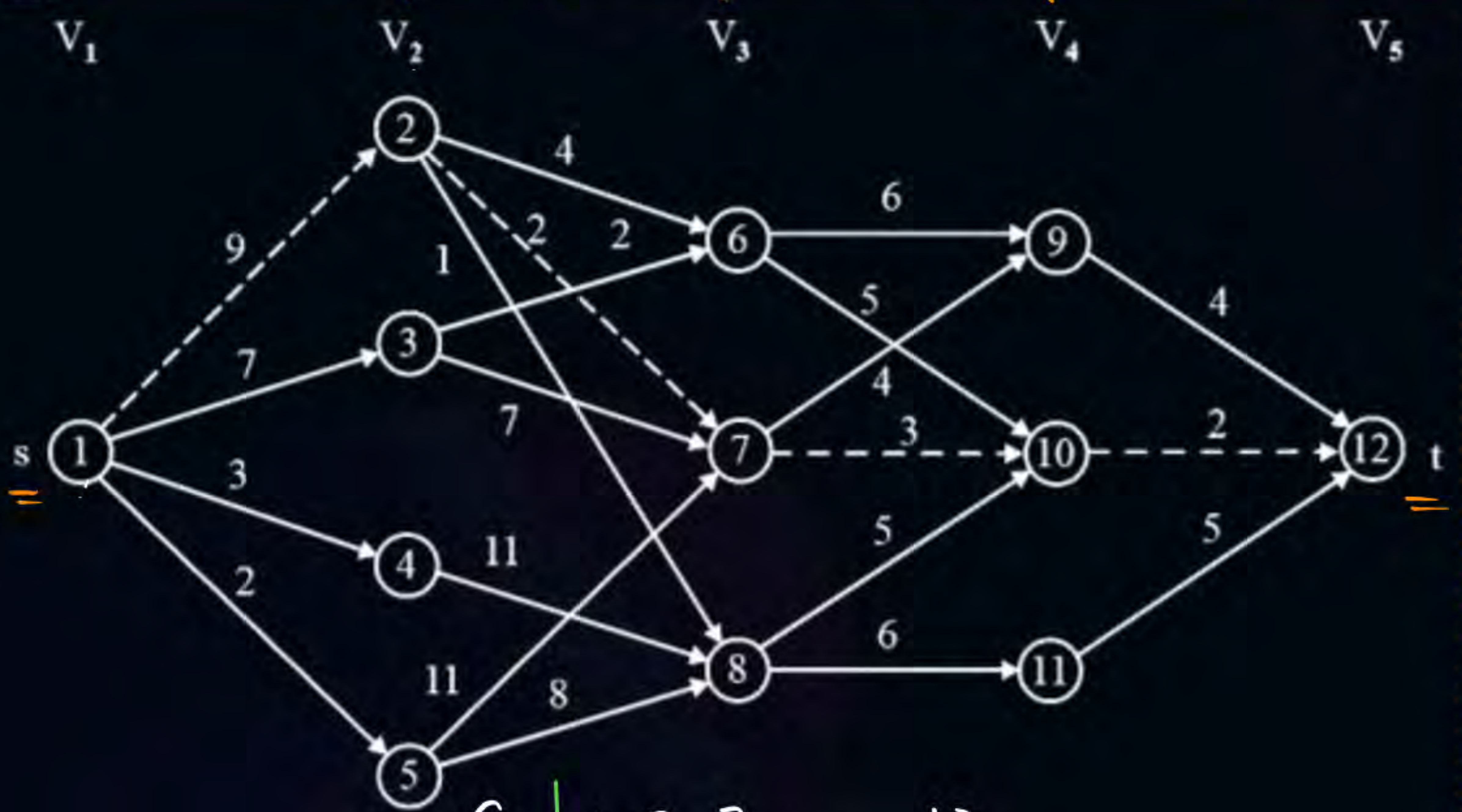


Topic : Dynamic Programming: (DP)

- Unlike Divide and Conquer, D.P typically involves solving all subproblems, rather than a small portion of subproblem.
- D.P tends to solve each subproblem only once since the results of the subproblems are stored, which are used later again when required. This is going to reduce the computation drastically . (In most case, the complexity)

Ex- Fib $\begin{cases} \text{Brute force : } O(2^n) \\ \text{D.P Implementation : } O(n) \end{cases}$

Topic : Dynamic Programming: (DP)



C	1	2	3	. .	12
1	-	9	7		
2					
3					
⋮					
12					

$c(i,j)$ = edge Cost

1) Multi-Stage Graph

$G = (V, E)$ $|V| = n$ $|E| = e$

l-Stage

$V_i \rightarrow V_{i+1}$

Let COST(i, j) repr. Cost of the Path from vertex 'j' present in Stage 'i', to reach dest. vertex 't';

COST$(1,1)$

$$\text{COST}(1,1) = \min_{\substack{K \in V_2 \\ \langle 1,K\rangle \in E}} \left\{ c(1,K) + \text{COST}(2,K) \right\}$$

original path

$$\text{COST}(1,1) = c(1,K) + \text{COST}(2,K)$$

$\langle 1 — K - - - - - - - - - - t \rangle$

$K \in V_2$

$\langle 1,K\rangle \in E$

Sub path

$\text{COST}(2,K)$

generalize this formula : $\text{COST}(i,j)$

$\text{COST}(i,j)$

i: Stage #

j: Vertex in Stage 'i'

THANK - YOU