CS & IT ENGINEERING

Algorithms

Dynamic Programming (DP)

Lecture No.- 02

By- Aditya Jain sir

Recap of Previous Lecture

Topic

Topic

Dijkstra SSSP

Intro to DP

Topics to be Covered

- Topic
- Topic
- Topic

DP

Applications

About Aditya Jain sir

1. Appeared for GATE during BTech and secured AIR 60 in GATE in very first attempt - City topper
2. Represented college as the first Google DSC Ambassador.
3. The only student from the batch to secure an internship at Amazon. (9+ CGPA)
4. Had offer from IIT Bombay and IISc Bangalore to join the Masters program
5. Joined IIT Bombay for my 2 year Masters program, specialization in Data Science
6. Published multiple research papers in well known conferences along with the team
7. Received the prestigious excellence in Research award from IIT Bombay for my Masters thesis
8. Completed my Masters with an overall GPA of 9.36/10
9. Joined Dream11 as a Data Scientist
10. Have mentored 12,000+ students & working professions in field of Data Science and Analytics
11. Have been mentoring & teaching GATE aspirants to secure a great rank in limited time
12. Have got around 27.5K followers on Linkedin where I share my insights and guide students and professionals.

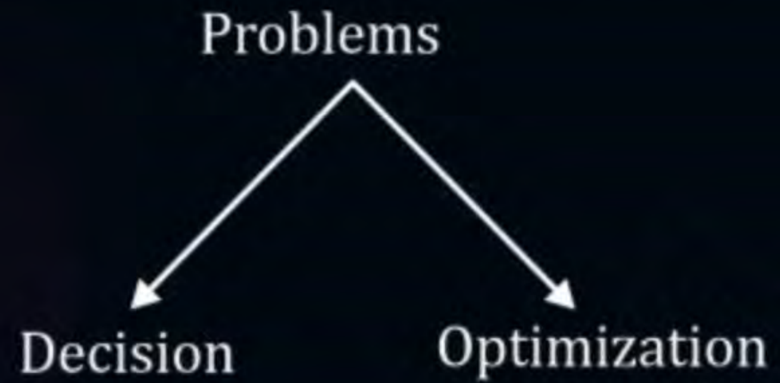
Telegram Link for Aditya Jain sir: https://t.me/AdityaSir_PW

Topic : Dynamic Programming

Introduction to Dynamic Programming:

~~Sorting~~/ Tabulating the values or Results of the sub- Problems.

Storing



Topic : Dynamic Programming

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

Eg. 0/1 Knapsack

Main diff. between greedy & DP.

Sometimes greedy appr Does not give optimal solution of some problems.

Topic : Dynamic Programming

Consider the weights and values of times listed below: Note that there is only one unit of each item.

M = 11 (Capacity)

Item number	Weight (in kgs)	Profit (in Rupees)
1	10	60
2	7	28
3	4	20
4	2	24

Binary Knapsack

Topic : Dynamic Programming

	W	P
01	10	60
02	7	28
03	4	20
04	2	24

$\rightarrow 60/10 = 6 \rightarrow 2 \times x_1 = 0$

$\rightarrow 27/7 = 4 \rightarrow 4 \times x_2 = 0$

$\rightarrow 40/4 = 5 \rightarrow 3 \times x_3 = 1$

$\rightarrow 24/2 = 12 \rightarrow 4 \times x_1 = 1$

M = 11

11- 2 =9

10 > 9 → Skip

9 – 4 = 5

Maximum Profit By Queued approach to 0/1 Knapsack:

$= \sum_{i=1}^{4} P_i \times X_i$

$= 60 * 0 + 28 * 0 + 20 * 1 + 24 * 1$

$= 20 + 24$

$= 44$

Topic : Dynamic Programming

But optimal profit

when taken only object O_1

= 60

Hence, greedy approach fails to give optimal solution in 0/1 knapsack (may or not give)

Fractional KS ⟶ Greedy

0/1 KS ⟶ DP

Topic : Dynamic Programming

Eg.1. Coin change Problem (DP vs Greedy)

Problem:- Given a set of coin values, construct a sum of money using as few coins as possible.

You can use each coin value any number of times.

Coin Values: $\{C_1, C_2 \ldots C_K\}$

Target Money (Sum) : N

Topic : Dynamic Programming

Eg.1. Coins = {1,2,5}, Sum = N =12

Greedy Method: 5 + 5 + 2 = 12

Number of coins required = 3

in this case, Greedy gives optimal answer.

Topic : Dynamic Programming

Eg.1. Coins = {1, 3, 4 } , N = 6 (sum)

Greedy approach: 4 + 1+ 1 = 6 (Number of coins = 3)

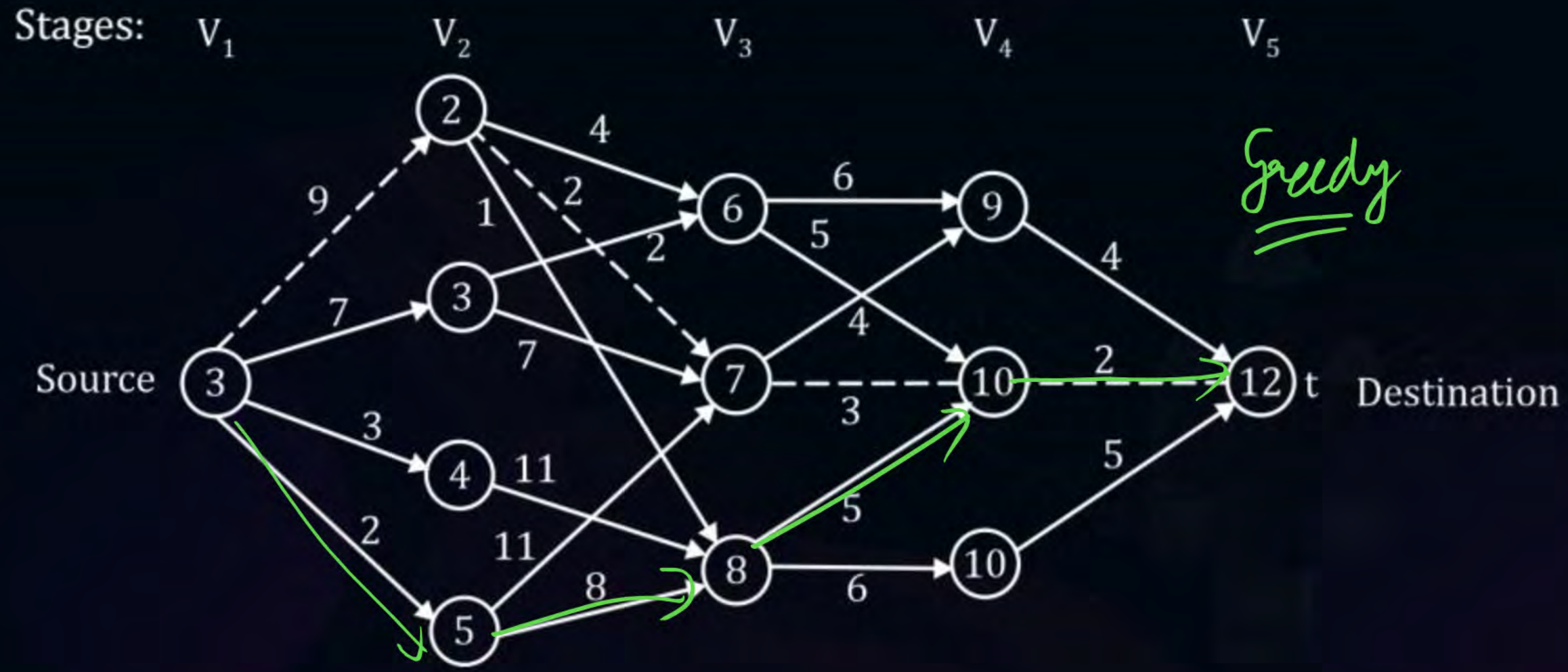
in general, optimal sol. 3 + 3 = 6 (number of coins = 2)

Topic : Dynamic Programming

In above example , greedy approach failed to give optimal solution.

Hence, greedy approach doesn't guarantee (may or may not give) optimal solution to coins change problem.

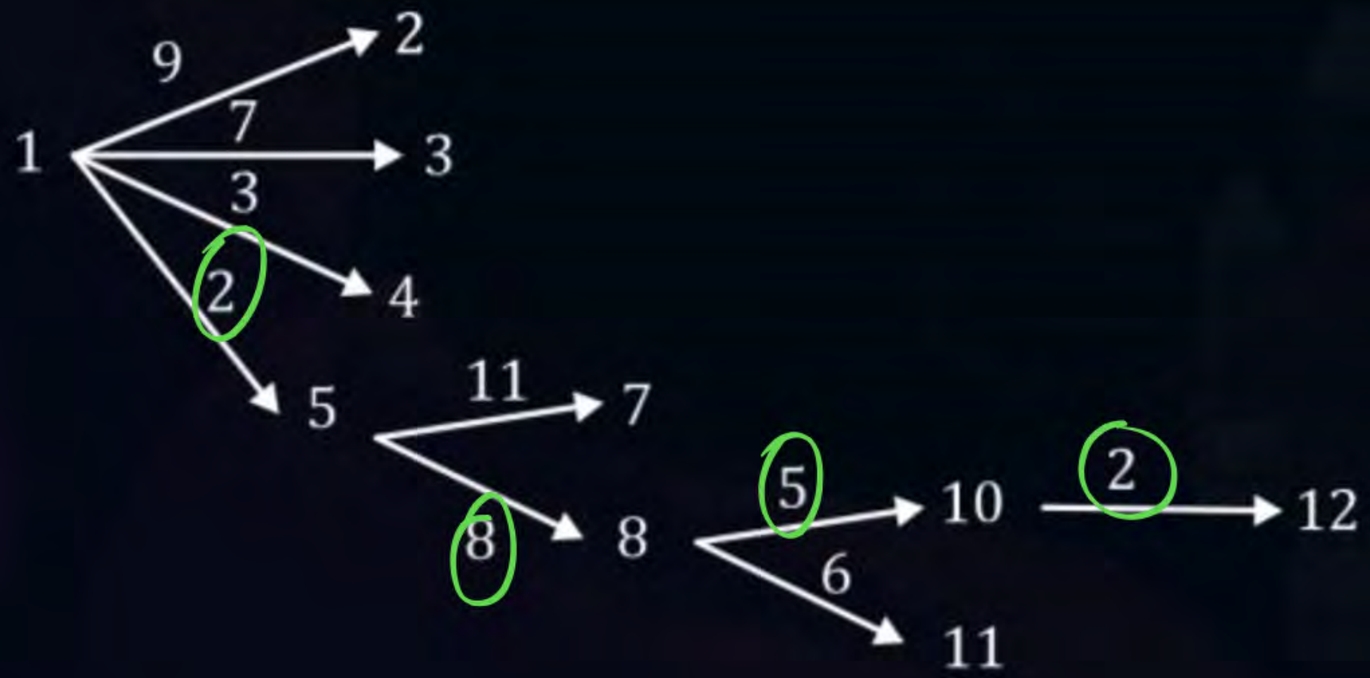
Topic : Dynamic Programming



Topic : Dynamic Programming

Greedy approach :- $1 \rightarrow 5 \rightarrow 8 \rightarrow 10 \rightarrow 12$

Cost = 2 + 8 + 5 + 2 = 17 {not optimal}



Topic : Dynamic Programming

But is 17 optimal /min path cost?

$< 1 \rightarrow 2 \rightarrow 7 \rightarrow 10 \rightarrow 12 >$

$9 + 2 + 3 + 2 = 16$

$< 1 \rightarrow 3 \rightarrow 6 \rightarrow 10 \rightarrow 12 >$

$7 + 2 + 5 + 2 = 16$

Actual minimum cost paths from $1 \rightarrow 12$

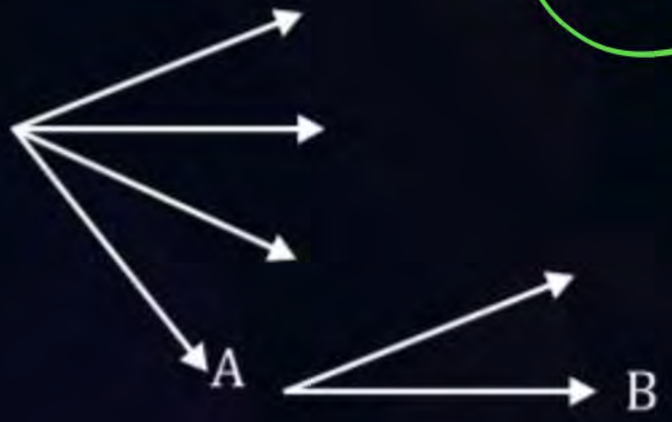
Hence, greedy approach of FGP failed to give optimal cost here.

Topic : Dynamic Programming

Local vs Global info

greedy

~~Global~~ DP



Topic : Dynamic Programming: (DP)

DP vs G reedy

1. 0/1 Knapsack
2. Coin change problem
3. Multi-stage graph

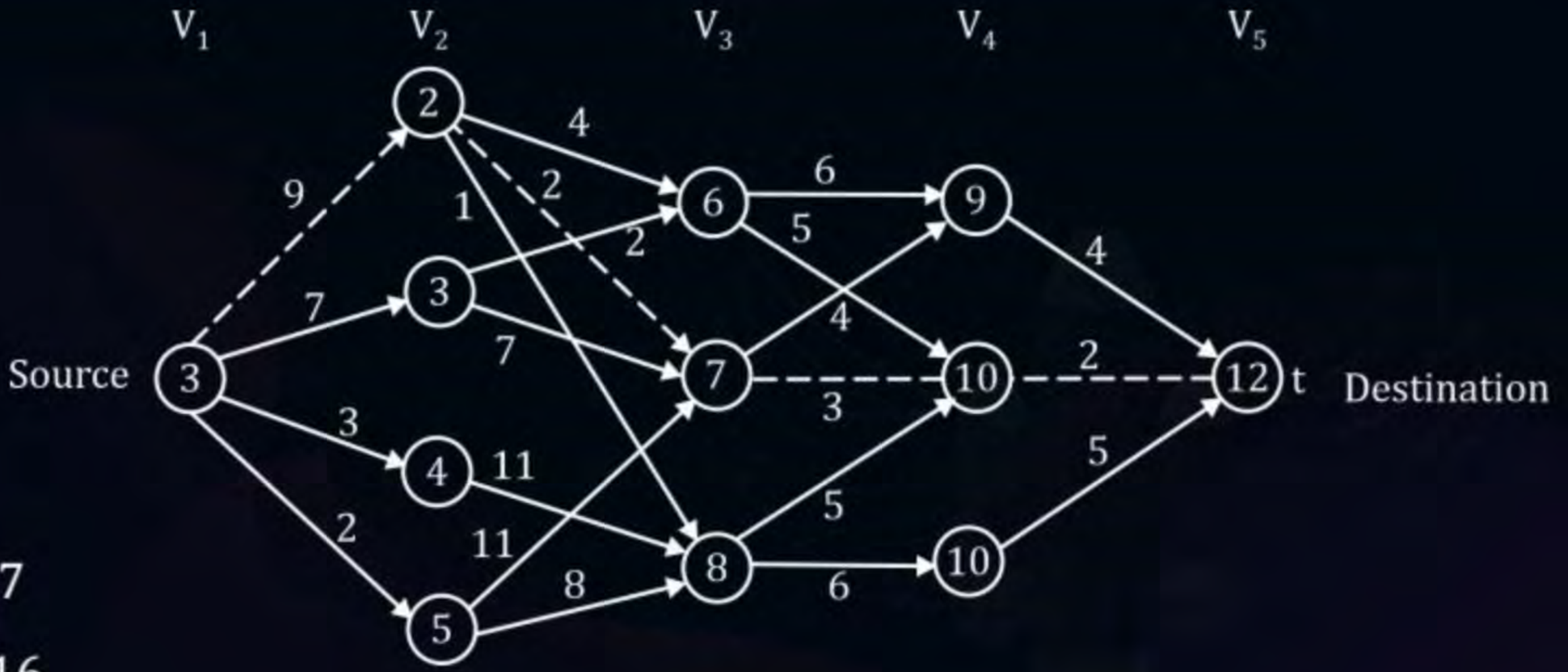
Greedy → Local optimisation

DP→ Global optimisation

DP gives optimal solution but greedy does not give optimal solution.

Topic : Dynamic Programming: (DP)

Multi-stage –graph



Greedy → 17

Optimal → 16

Topic : Dynamic Programming: (DP)

P → Path 1 → C1
Path 2 → C2
Path 3 → C3
Path 4 → C4
Path 5 → C5
Path n → Cn

→ Min→ Optimal solution

Enumeration **or** Brute force

High TC

DP 0→ Brut force

Topic : Dynamic Programming: (DP)

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

D.P implementation

- **Memorization (Top-Down Approach)** Recursive
- **Tabulation (Bottom-Up)** Iterative.

Topic : Dynamic Programming: (DP)

(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)

Fibonacci Number:

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

Topic : Dynamic Programming: (DP)

Normal Recursive Implementation of Fib (n):

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

Topic : Dynamic Programming: (DP)

Time complexity Recurrence

$T(n) = C, n = 1$

$T(n) = T(n-1) + T(n-2) + a , n > 1$

We know,

$T(n-2) < T(n-1)$

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

$T(n) = 2T(n-1) + a , n > 1$

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

Topic : Dynamic Programming: (DP)

$$T(n) = 2[2T(n-2) + a] + a$$

$$= 2^2 T(n-2) + 2a + a \ldots\ldots (2)$$

$$T(n-2) = 2T(n-3) + a$$

$$T(n) = 2^2 (2T(n-3) + a) + 2a + a$$

$$= 2^3 T(n-3) + (2^2 a + 2^1 a + 2^0 a)$$

General Term

$$T(n) = 2^k T(n-k) + (2^k - 1) * a$$

For Base Condition

$$n - k = 1 \rightarrow k = (n-1)$$

Topic : Dynamic Programming: (DP)

$$T(n) = 2^{(n-1)} * T(1) + (2^{(n-1)} - 1) * a$$

$$= c * \frac{2^n}{2} + \left(\frac{2^n}{2} - 1\right) * a$$

$$T(n) \rightarrow O(2^n)$$

EXPO

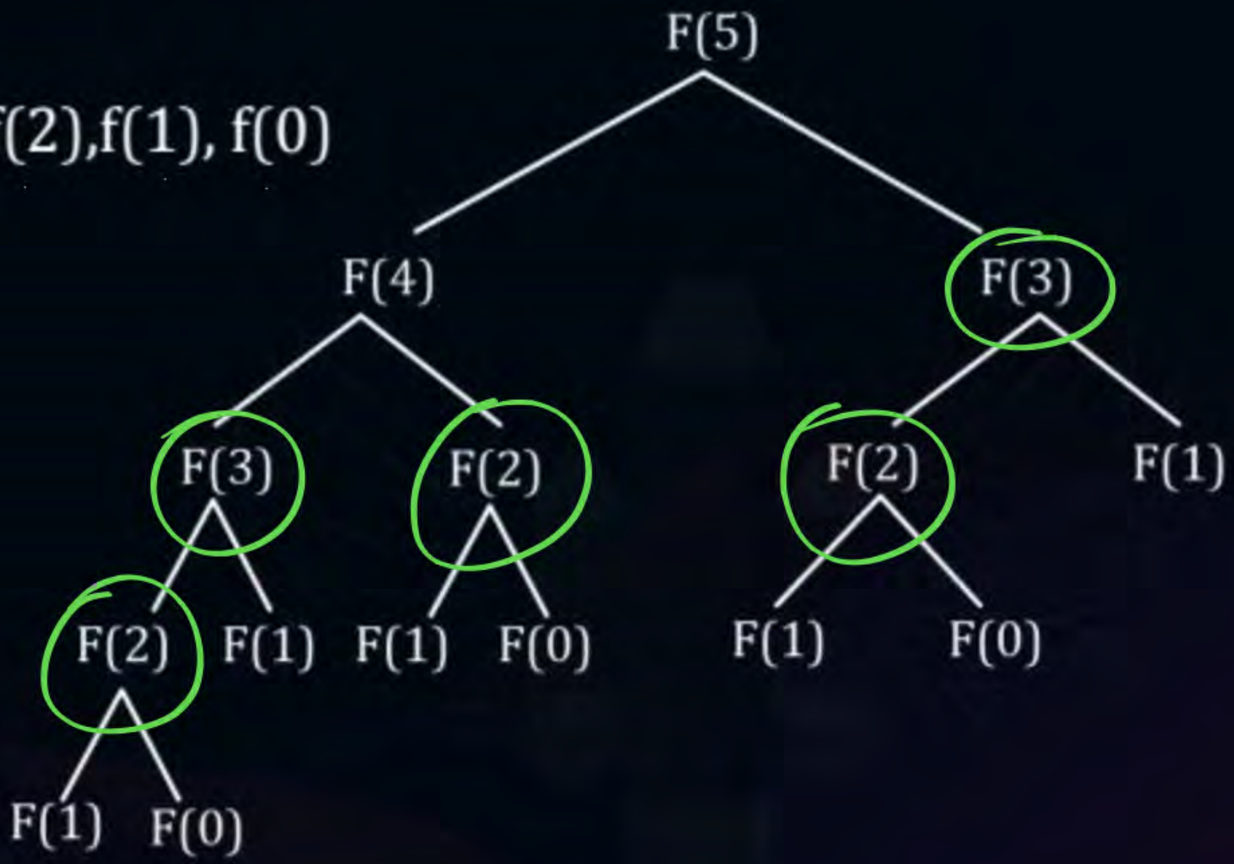
Topic : Dynamic Programming: (DP)

Normal Recursive Approach: Tree of function calls

Original problem → F(5)

unique Sub-problems → F(4), f(3), f(2),f(1), f(0)

(Q) Find fibo (5):



Topic : Dynamic Programming: (DP)

Top-down- Memorized implantation of Fib (n)

```
Algo memFb(n)
{
    m[0……..n] = ∞; //Initialization
  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]);
}
```

Topic : Dynamic Programming: (DP)

Top-Down code walkthrough:

#Q. Fib (5) = ?

$0 \rightarrow 5$

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

?

memFib(5): res: 3 + 2 = 5→ m [5]= 5

memFib(4):res: 2 + 1 = 3→ m [4]= 3

memFib(3):res: 1 + 1 = 2→ m [3]= 2

memFib(2): res: 1 + 0 = 1→ m [2]= 1

memFib(1):m [1]= 1

memFib(0): m [0]= 0

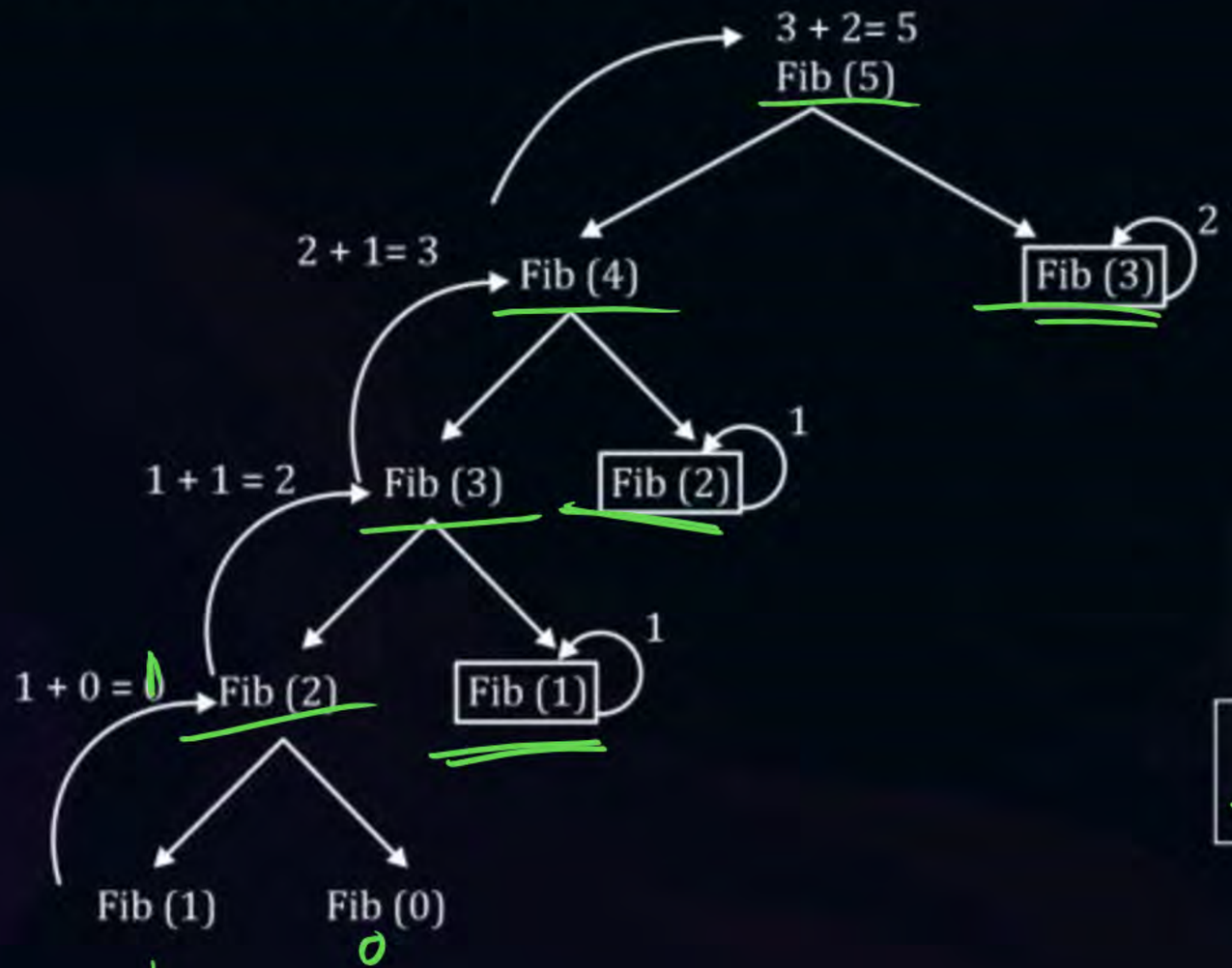
memFib(0): 1

memFib(2): m[2]

memFib(3): m [3]

Topic : Dynamic Programming: (DP)

Function call Tree for above approach.



$TC \rightarrow O(n)$

$O(2^n) \rightarrow O(n)$

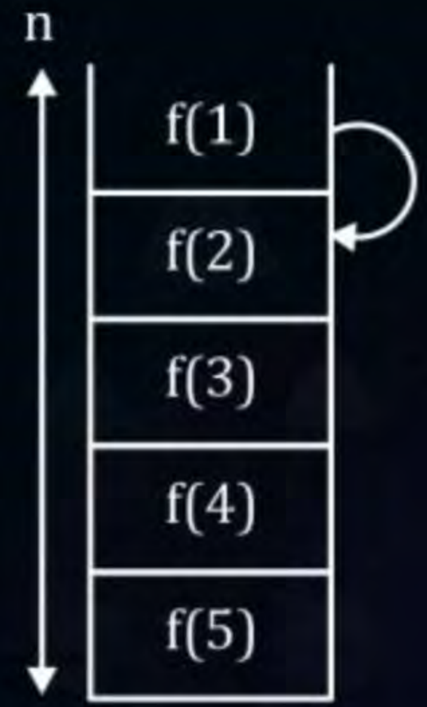
Topic : Dynamic Programming: (DP)

Space Complexity

- Recursion stack → O(n) {Stack size}
- Auxiliary array → O(n)

m[n]

Overall space Complexity → O(n + n) = O(n)

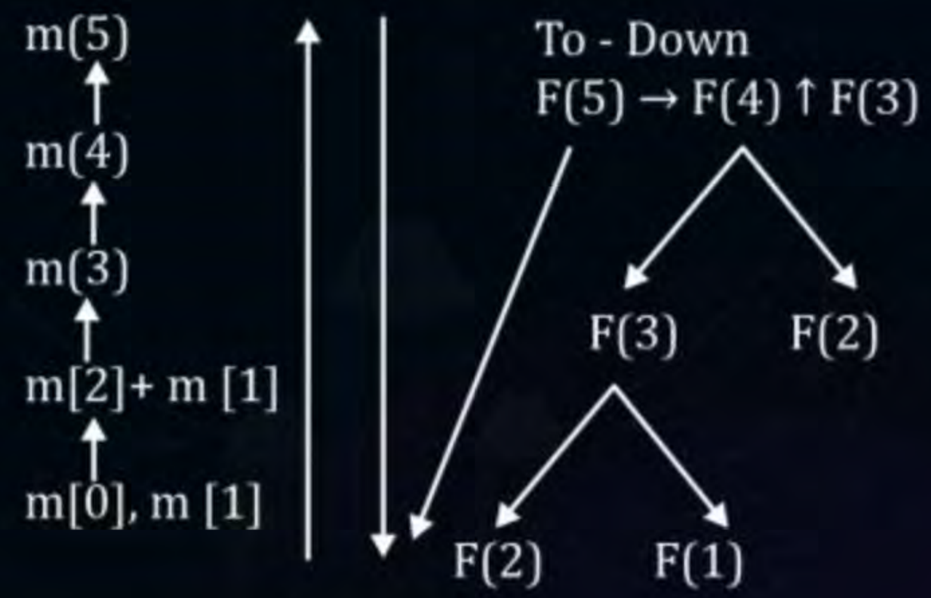


Topic : Dynamic Programming: (DP)

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

Appr 2: Tabulation (Iterative)

```
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]);
}
```



Topic : Dynamic Programming: (DP)

Eg.1. Fib(5)

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

$i = 2, m[2] = m[1] + m[0] = 1 + 0 = 1$

$i = 3, m[3] = m[2] + m[1] = 1 + 1 = 2$

$i = 4, m[4] = m[3] + m[2] = 2 + 1 = 3$

$O(2^n) \rightarrow O(n)$

$TC \rightarrow O(n)$

$SC \rightarrow O(n)$

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 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.

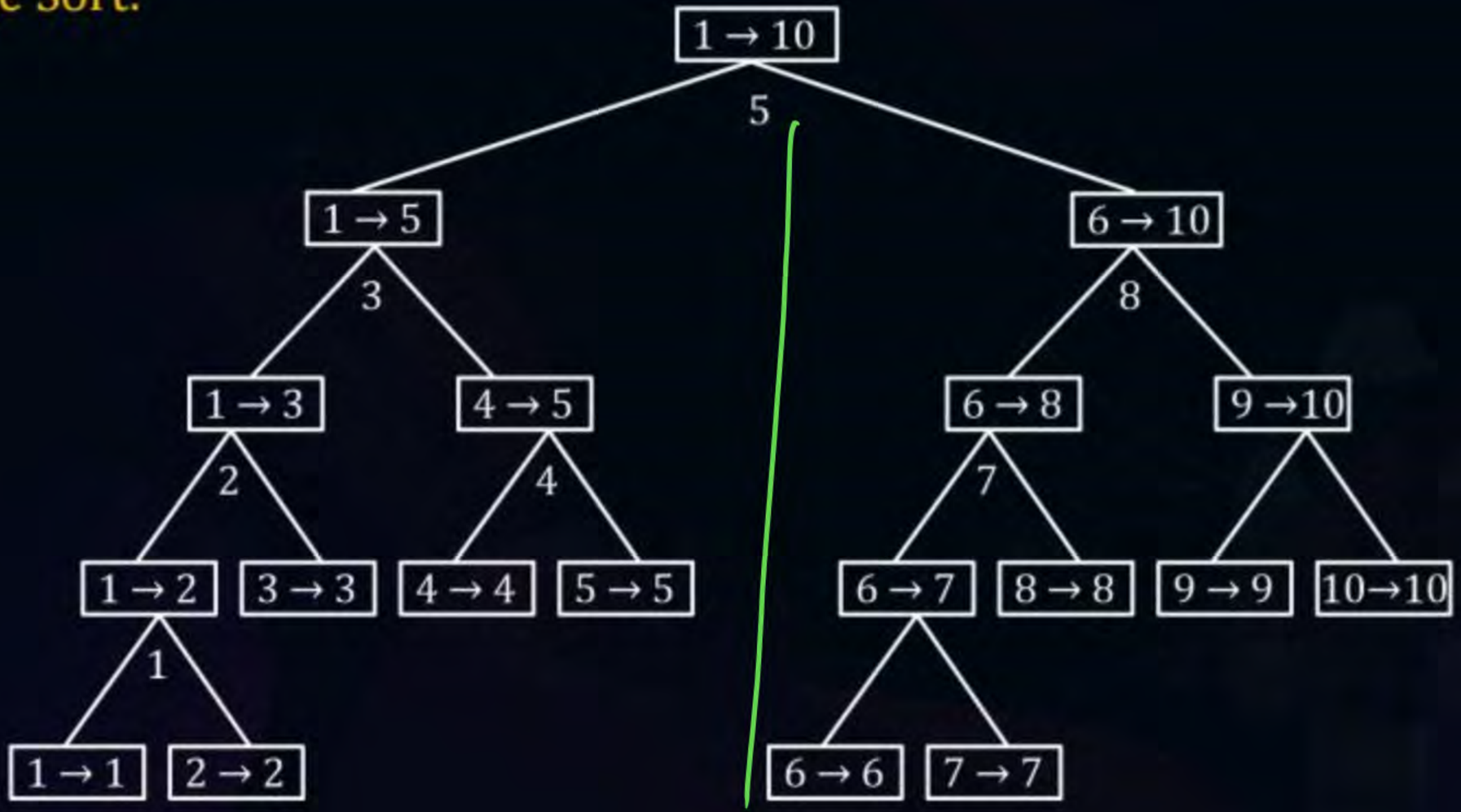
Topic : Dynamic Programming: (DP)

Imp. Diff. DandC and DP:

1. DnC: independent sub-problems solve separately
2. DP: Overlapping Sub- problems

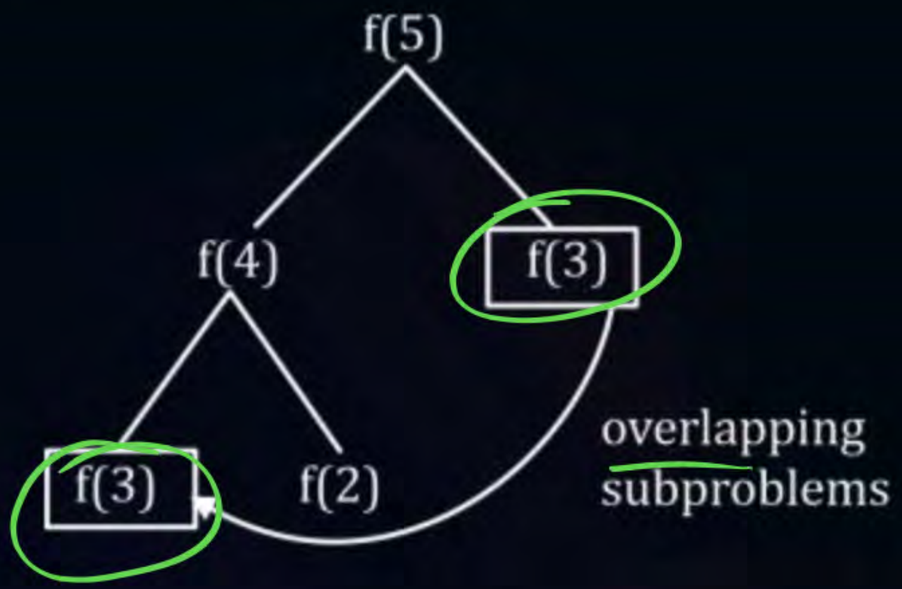
Topic : Dynamic Programming: (DP)

Merge Sort:



Topic : Dynamic Programming: (DP)

DP → Fibonacci



Application of DP:

Topic : Dynamic Programming: (DP)

Single source Shortest path (SSSP)

1. Dijkstra → Greedy
2. Bellman ford → DP

Distinction between both SSSP algorithms and which to use when?

Topic : Dynamic Programming: (DP)

1. Dijkstra's SSSP (greedy algo) always gives optimal solution to the SSSP problems, provided all the edges in the given graph are of +ve weight.

2. If the graph has any –ve weight edge, then Dijkastra SSSP algo may or May not give optimal solution. (Path costs)

Topic : Dynamic Programming: (DP)

3. If the graph has 0 or more –ve weight edge But NOT a –ve weight cycle, then the bellman ford SSSP algo (DP based) always gives optimal solution to SSSP problem.

4. If the graph has any –ve weight cycle that is reachable from source, then No algorithm works.

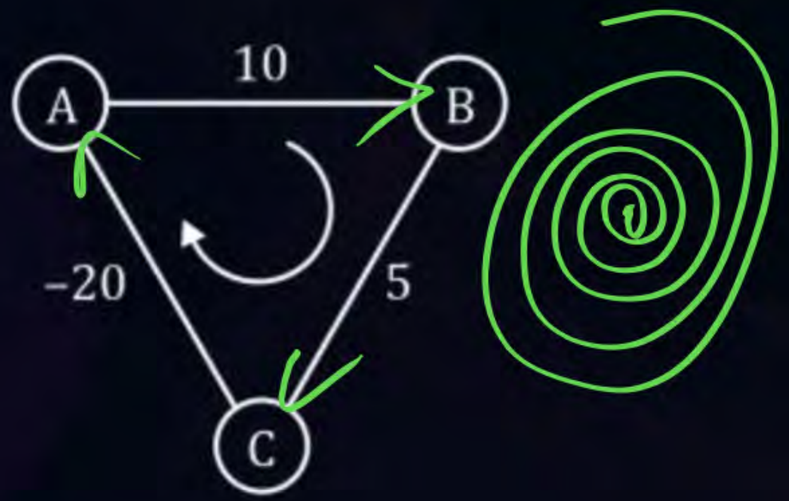
Topic : Dynamic Programming: (DP)

-ve weight cycle?

Net weight of cycle = 10 + 5 – 20

= -5 (-ve weight cycle)

* Min with path is not defined.



A → B?

10?

A → B: 10

A → B → C → A → B

- 5 + 10 = 5

10 → 5 → 0 → – 5 → –10

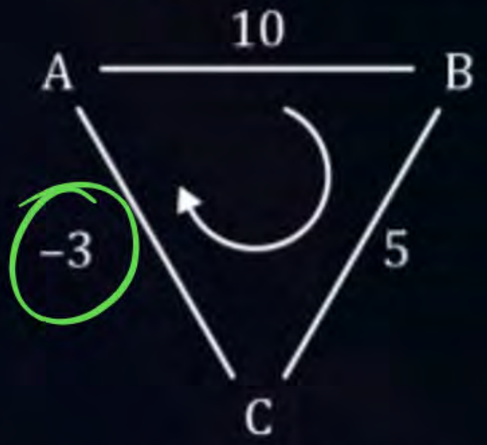
No algo

Topic : Dynamic Programming: (DP)

Eg.1. Graph with -ve weight edge but No ~~required with~~ cycle.

Net weight = 10 + 5 – 3

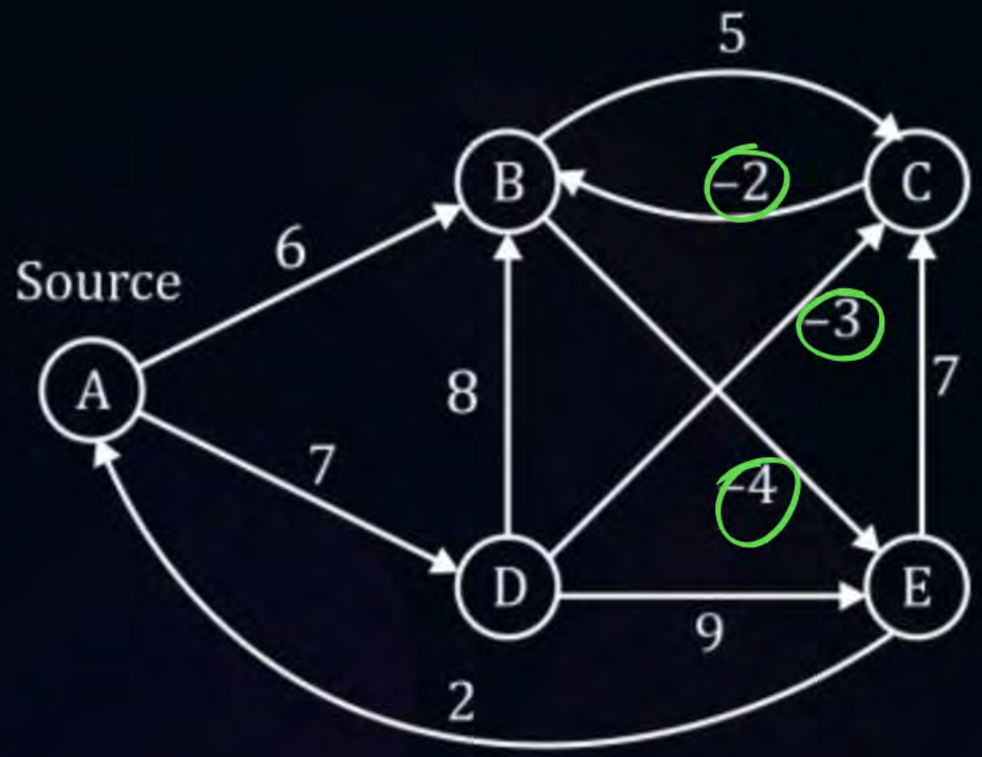
= 12 (+ve weight Cycle)



Dijkstra SSSP→ Not guarantee (-ve weight edge)

Topic : Dynamic Programming: (DP)

Imp Given: G



Apply Dijkstra SSSP Algo

	A	B	C	D	E
{A}	0	6	∞	7	∞
{A,B}	0	6	11	7	2
{A,B,E}	0	6	9	7	2
{A,B,E,D}	0	6	4	7	2
{A,B,E,D,C}	0	6	4	7	2

Topic : Dynamic Programming: (DP)

$A \rightarrow B$ min cost ?

$A \rightarrow D \rightarrow C \rightarrow B : \Rightarrow 7 + (-3) + (-2)$

$= 7 - 5 = 2$ {Actual Minimum}

Dijikstra $A \rightarrow B \rightarrow 6 \times$

$A \rightarrow E$: ?

$A \rightarrow D \rightarrow C \rightarrow B \rightarrow E \Rightarrow 7 + (-3) + (-2) + (-4) = 7 - 9 = -2$ {Actual min}

Dijikstra $A \rightarrow E \rightarrow 2 \times$

Topic : Dynamic Programming: (DP)

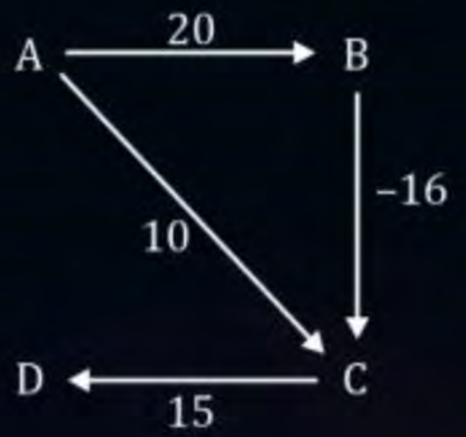
IMP Note:

1. In previous e.g. Dijkastra SSSP failed to give optimal path cost to few vertices, because in Dijkstra, once the vertex is selected is not further considered to be relaxed.

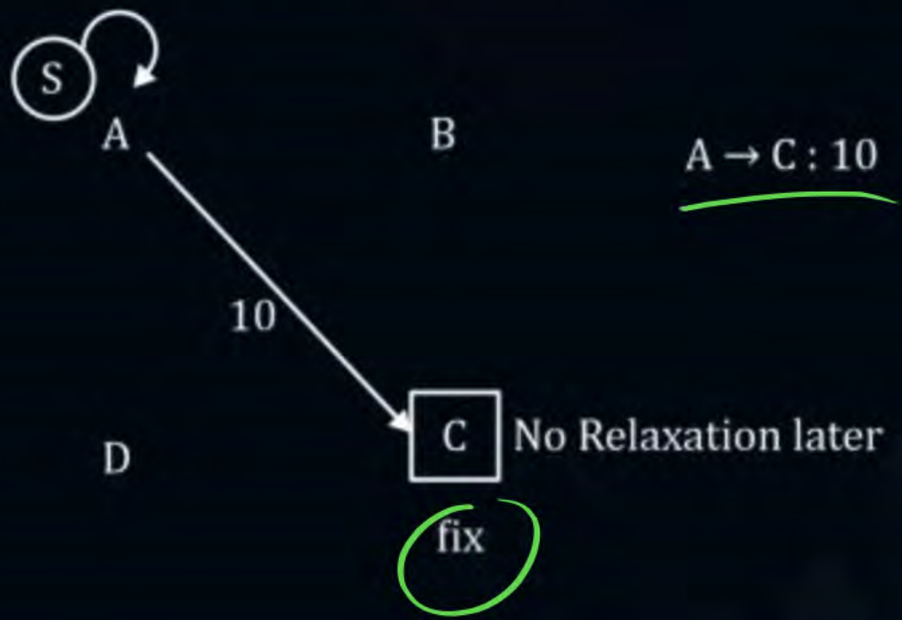
2. In Bellman ford, the relaxation is carried out w.r.t. to the edges in multiple iterations (n-1) iterations.

Topic : Dynamic Programming: (DP)

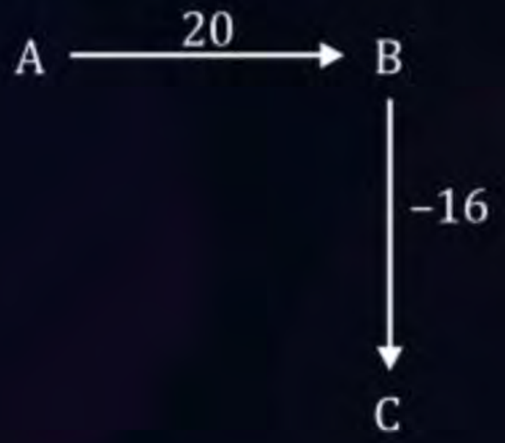
Idea:-



Dijkstra SSSP



$A \rightarrow C : 10$



$\Rightarrow A \rightarrow C : \quad 20 - 16$

$= 4$

Topic : Dynamic Programming: (DP)

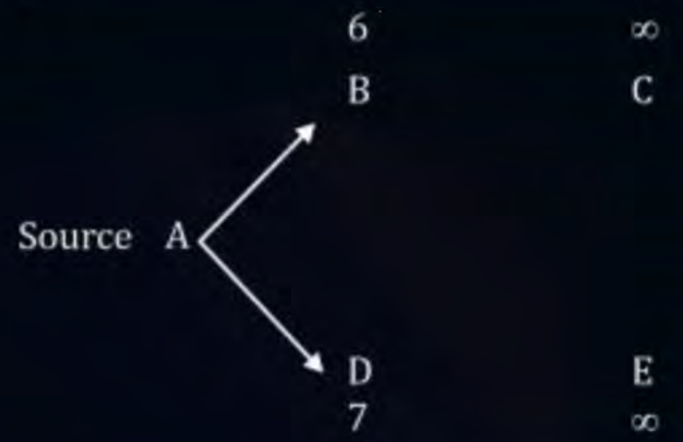
n = 5

Iteration: n −1 = 5-1= 4

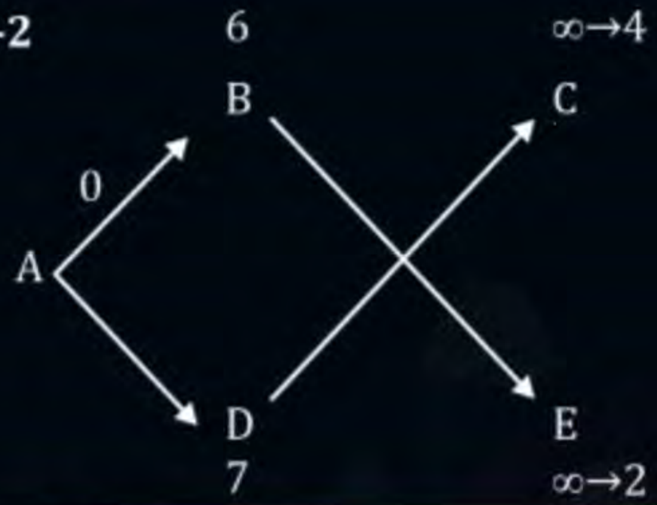
Topic : Dynamic Programming: (DP)

Bellman ford Walk through

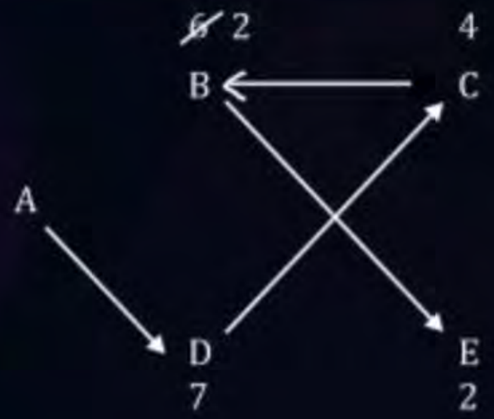
Iter.-1



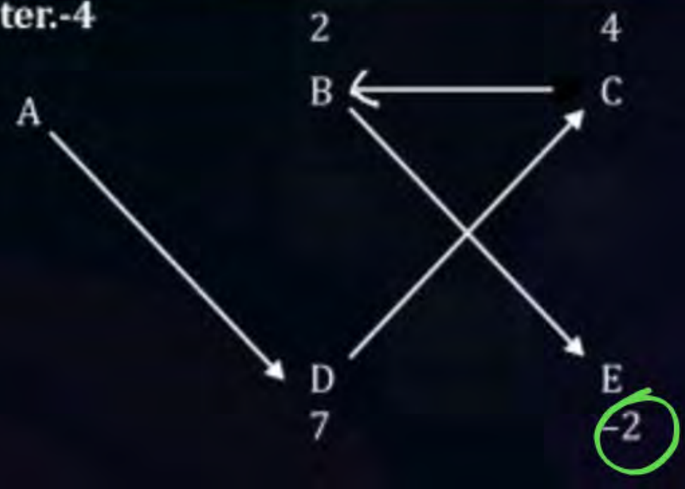
Iter.-2



Iter.-3



Iter.-4



Topic : Dynamic Programming: (DP)

Dj → A → B : 6

A → C : 4

A → D : 7

A → E : 2

Bellman ford → A → B : 2

A → C : 4

A → D : 7

A → E : –2

Imp. Observation:

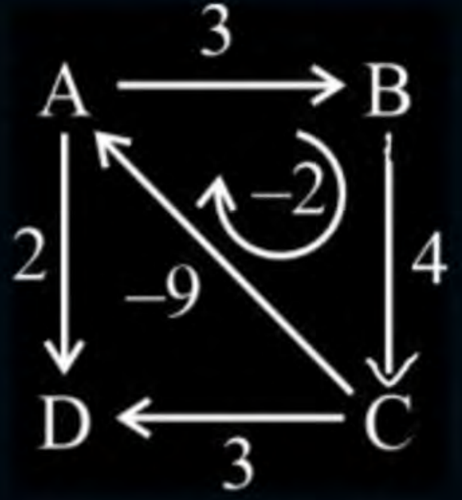
Dijkastra Greedy SSSP algorithm failed to give us the shortest path to few vertices. But bellman ford SSSP given shortest paths (optimal solution) to all the vertices. (Even through –ve weight edge are present but no-ve weight cycle.)

Topic : Dynamic Programming

Bellman Ford:

n = 4

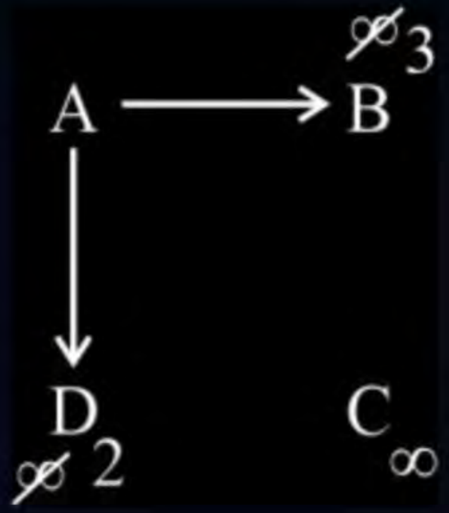
n – 1 = 3



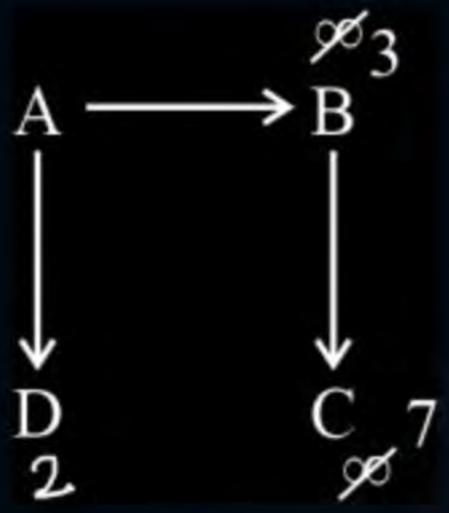
Apply BF

Topic : Dynamic Programming

Iteration 1:

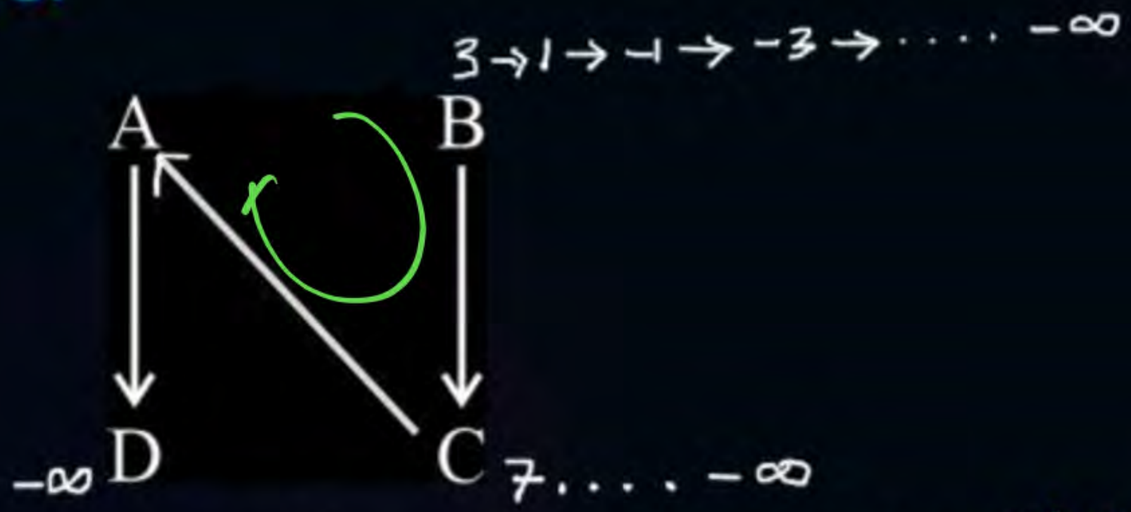


Iteration 2:



Topic : Dynamic Programming

Iteration 3:



In above example, as there is a negative edge weight cycle reachable from source, so even Bellman Ford fails

Topic : Dynamic Programming

Algorithm Bellman-Fort (G, w, s)

```
1.      Initialize-Single-Source(G,s)
2.      for i ← 1 to |V[G]| – 1
3.              do for each edge (u, v) ∈ E[G]
4.                      do Relax(u, v, w)
5.      for each edge (u, v) ∈ E[G]
6.              do if d[v] > d[u] + w(u, v)
7.                      then return FALSE
8.      return TRUE
```

n - 1

-ve cycle

Topic : Dynamic Programming

Initialize – Single- Source (G, s)

1. For each vertex $u \in V[G]$
2. do $d[u] \leftarrow \infty$
3. $\pi[v] \leftarrow NIL$
4. $d[s] \leftarrow 0$

Relax (u, v, w)

1. if $d[v] > d[u] + w(u, v)$
2. then $d[v] = d[u] + w(v, u)$
3. $\pi[v] \leftarrow u$

Topic : Dynamic Programming

Overall Time Complexity:

- Bellman Ford SSSP DP approach

  TC → O(n + n*e + e) = O(n * e)

O(V*E)

Topic : Dynamic Programming

[NAT]

#Q. The time complexity of Bellman Ford algo for a complete graph having n vertices is?

$TC : O(n^3)$

$e = O(n^2)$

Topic : Dynamic Programming

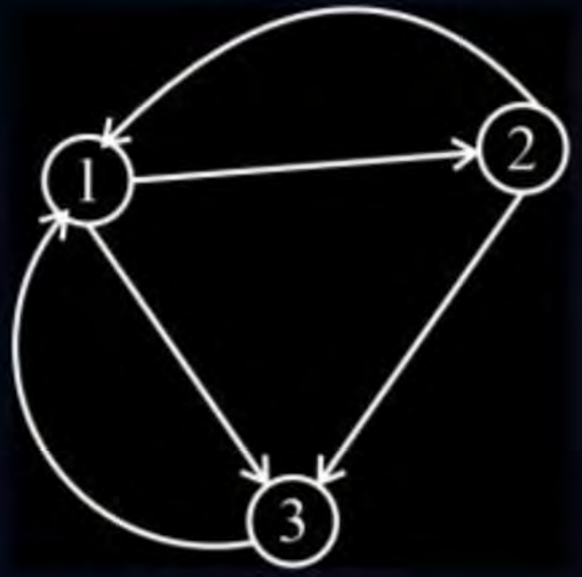
Floyd Warshall Algorithm:

- For APSP (All Pairs Shortest Paths)

Topic : Dynamic Programming

Example:

Given, Cost adj matrix



C	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

Topic : Dynamic Programming

DP based idea:

- Let $A^k(i, j)$ represent the cost of the path from a vertex 'i' (source) to some vertex 'j' (destination) with k being the highest intermediate vertex along the path from $i \rightarrow j$.

Topic : Dynamic Programming

$$A^k(i,j) = \min\{A^{k-1}(i,k) + A^{k-1}(k,j), A^{k-1}(i,j)\}$$

$$1 \le k \le n$$

$$A^0(i,j) = C(i,j) \quad , k=0$$

Topic : Dynamic Programming

```
Algo FloydWarshall (G, C, n, e)
{
        A[1...n, 1...n]
        for (i=1; i<=n; i++)
          for (j=1; j<=n; j++)
                    A[i, j] = C[i, j]
        for (k=1; k<=n; k++)
          for (i=1; i<=n; i++)
                  for (j=1; j<=n; j++)
                            A[i, j] = min(A[i,j], A[i, k] + A[k, j])
}
```

→ init

a = a + 2

Topic : Dynamic Programming

Time and Space Complexity Analysis:

1) Time Complexity = $O(n^2 + n^3) = O(n^3)$

2) Space Complexity = $O(n^2)$

THANK - YOU