MATHEMATICAL PROGRAMMING

CO 2 : Dynamic Programming

Dr.Vuda Sreenivasa Rao

DYNAMIC PROGRAMMING

- Dynamic programming is a technique that **breaks the problems into sub-problems**, and **saves the result for future purposes** so that we **do not need to compute the result again**.
- The **main use of dynamic programming** is to **solve optimization problems.**
- Here, **optimization problems** mean that when we are trying to **find out the minimum or the maximum solution of a problem.**

CONT..

- The dynamic programming guarantees to **find the optimal solution of a problem if the solution exists.**
- The **definition of dynamic programming says** that it is a technique for **solving a complex problem by first breaking into a collection of simpler subproblems**, **solving each subproblem just once**, and then **storing their solutions to avoid repetitive computations.**

CONT..

- Dynamic programming is **powerful design technique for optimization problems.**

- **Principle of optimality : "In an optimal sequence of decisions or choices, each sub sequence must also be optimal".**

CONT..

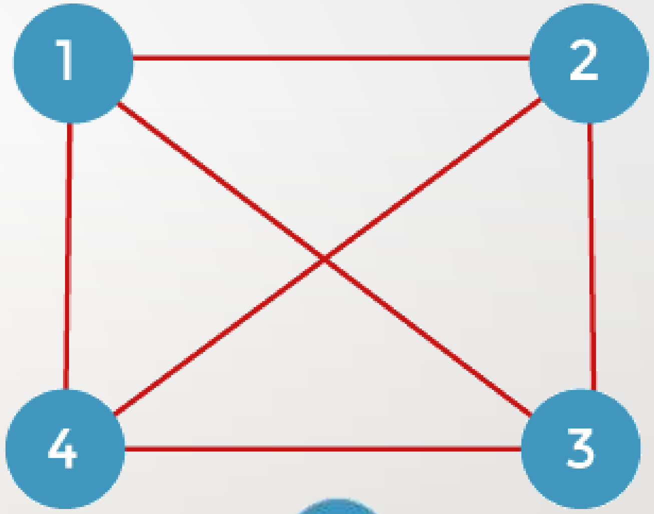
- The principle of optimality is the **heart of dynamic programming.**
- It states that to **find the optimal solution of the original problem**, a **solution of each sub problem also must be optimal**.
- **It is not possible to derive optimal solution using dynamic programming if the problem does not possess** the principle of optimality.

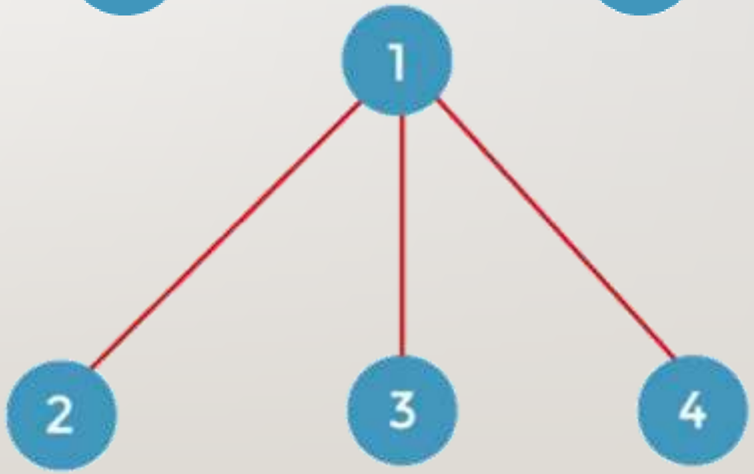
HOW DOES THE DYNAMIC PROGRAMMING APPROACH WORK?

- The following are the steps that the dynamic programming follows:
  - **It breaks down the complex problem into simpler subproblems.**
  - **It finds the optimal solution to these sub-problems.**
  - **It stores the results of subproblems (memorization). The process of storing the results of subproblems is known as memorization.**
  - **It reuses them so that same sub-problem is calculated more than once.**
  - **Finally, calculate the result of the complex problem.**

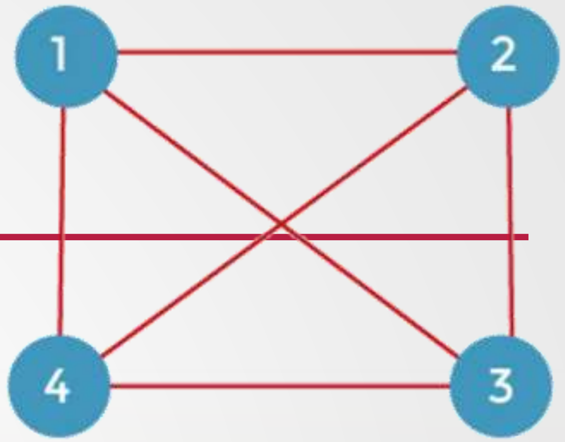
EXAMPLE

- Suppose we start travelling from **vertex 1 and return back to vertex 1.**
- There are various ways to travel through all the vertices and returns to vertex 1.
- From the starting vertex 1, we can go to either vertices 2, 3, or 4, as shown in the below diagram.

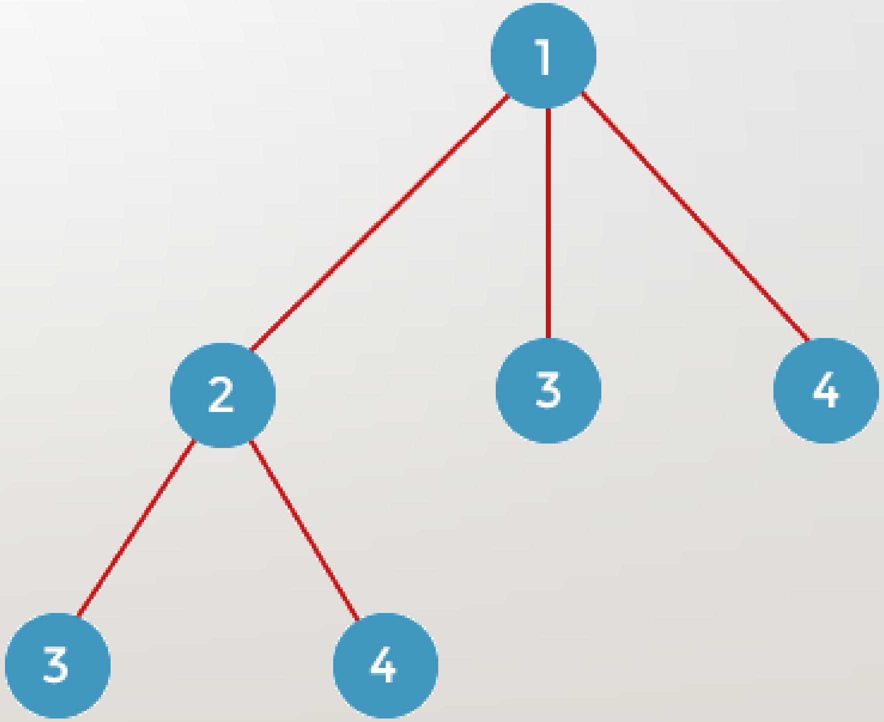




CONT..

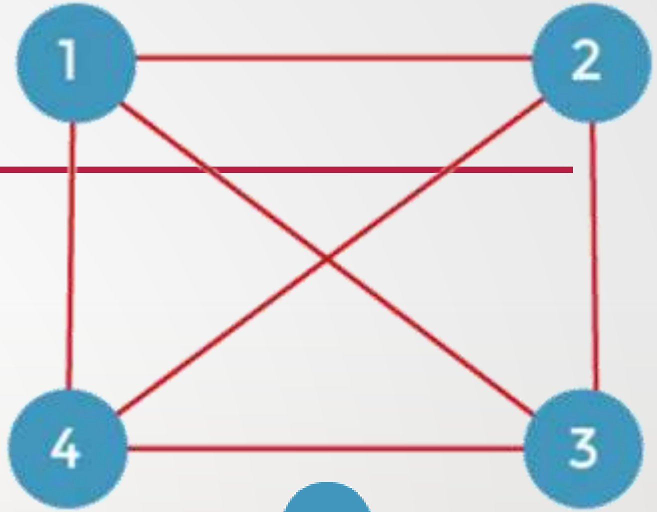


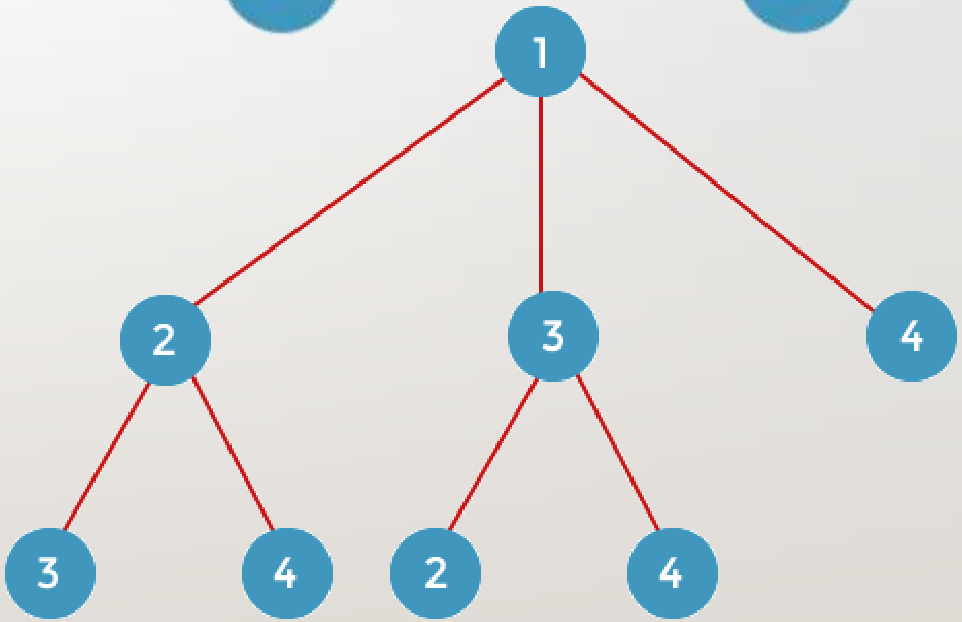
- From vertex 2, we can go either to vertex 3 or 4. If we consider vertex 3, we move to the remaining vertex, i.e., 4. If we consider the vertex 4 shown in the below diagram:



CONT..

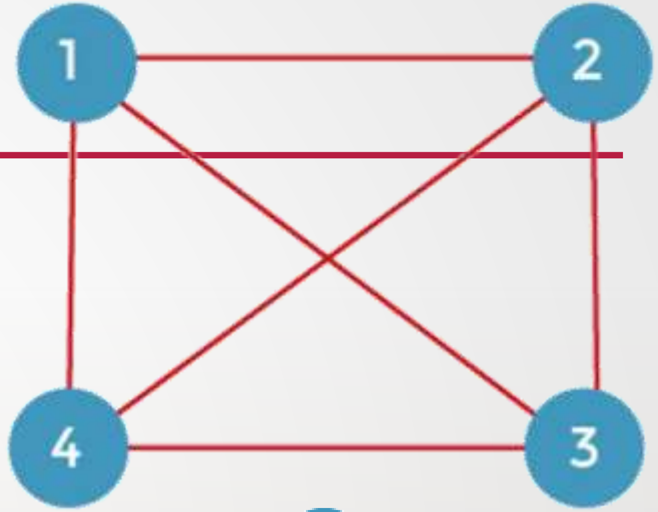
- From vertex 3, we can go to the remaining vertices, i.e., 2 or 4. If we consider the vertex 2, then we move to remaining vertex 4, and if we consider the vertex 4 then we move to the remaining vertex, i.e., 3 shown in the below diagram:

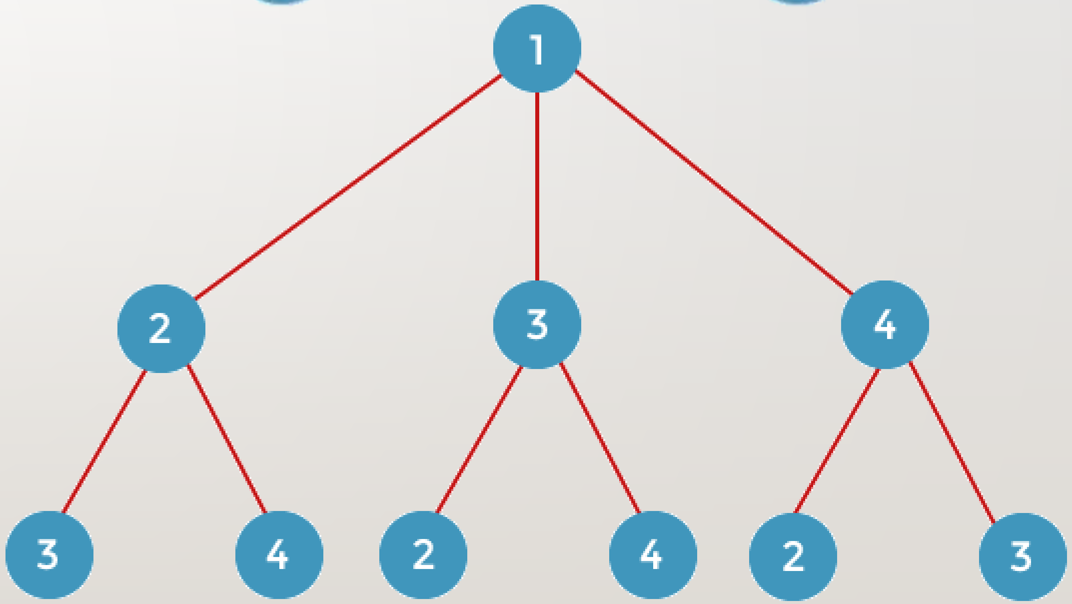




CONT..

- From vertex 4, we can go to the remaining vertices, i.e., 2 or 3. If we consider vertex 2, then we move to the remaining vertex, i.e., 3, and if we consider the vertex 3, then we move to the remaining vertex, i.e., 2 shown in the below diagram:





LIMITATIONS

- The method is applicable to only those problems which possess the property of principle of optimality.

- Dynamic programming is more complex and time-consuming.

APPLICATIONS OF DYNAMIC PROGRAMMING

- Dynamic programming is used to solve optimization problems.
- It is used to solve many real life problems such as,
  - **Make a change problem**
  - **Knapsack problem**
  - **Optimal binary search tree**
  - **Travelling salesman problem**
  - **All pair shortest path problem**
  - **Assembly line scheduling**
  - **Multi stage graph problem**

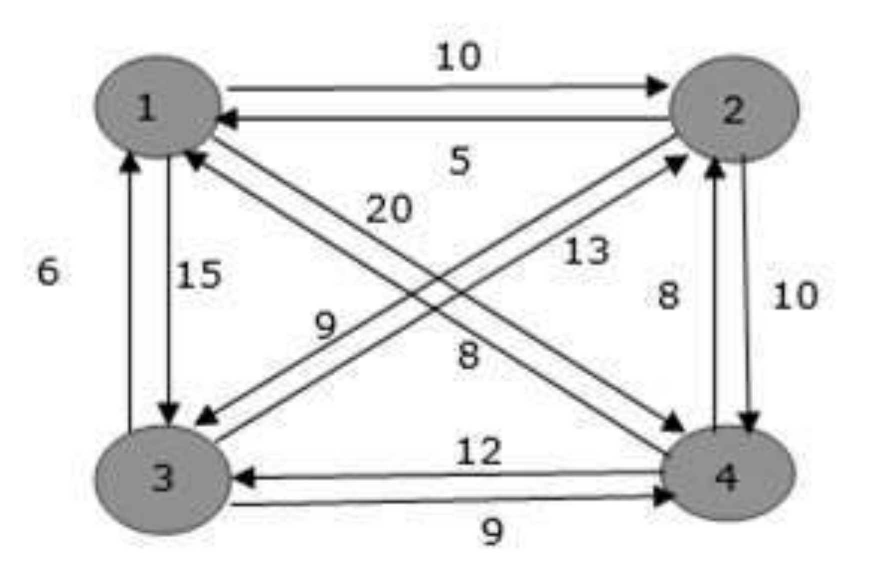
TRAVELLING SALESMAN PROBLEM USING DYNAMIC PROGRAMMING

- Travelling Salesman Problem-
  - You are given-
    - **A set of some cities.**
    - **Distance between every pair of cities.**
- Travelling Salesman Problem states-
  - **A salesman has to visit every city exactly once.**
  - **He has to come back to the city from where he starts his journey.**
  - **What is the shortest possible route that the salesman must follow to complete his tour?**

COMPLEXITY ANALYSIS OF TRAVELING SALESMAN PROBLEM

- **Dynamic programming creates $n.\ 2^n$ subproblems for n cities.**
- Each sub-problem can be solved in linear time.
- Thus the time complexity of TSP using dynamic programming would be **$O(n^2 2^n)$.**
- It is much less than n! but still, it is an exponent.
- Space complexity is also exponential.

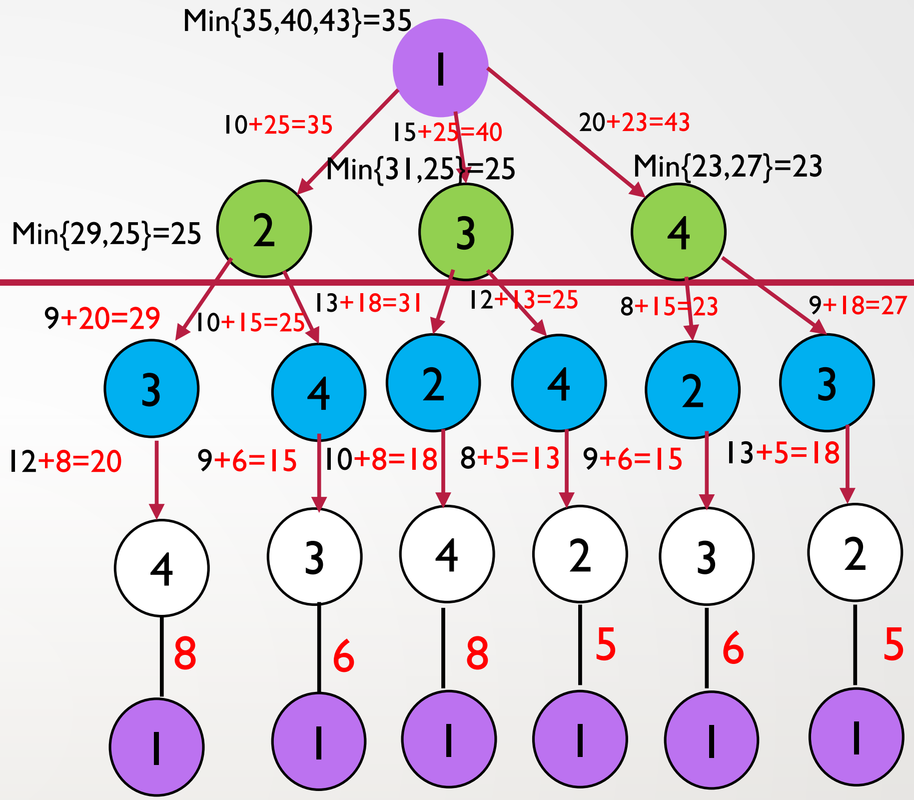
EXAMPLE

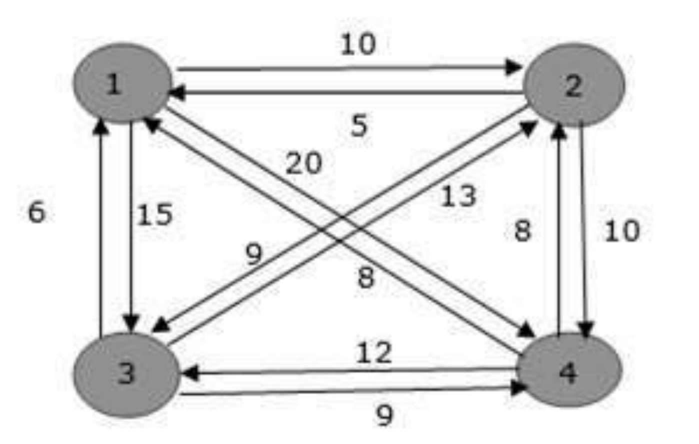


- Table

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

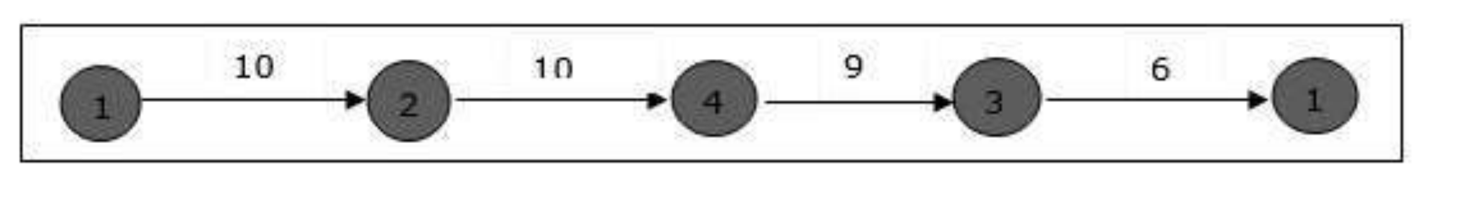
CONT…





	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

The minimum cost path is 35.



ALGORITHM FOR TRAVELING SALESMAN PROBLEM

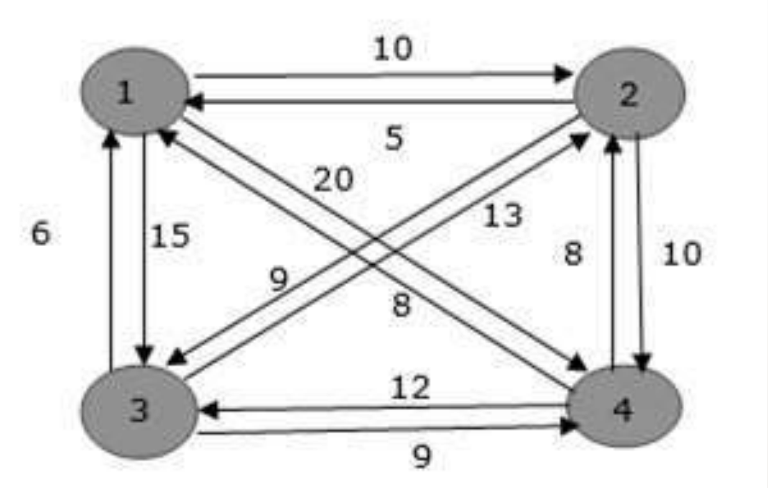
- **Step 1:**
  - Let d[i, j] indicates the distance between cities i and j.
  - Function C[x, V – { x }]is the cost of the path starting from city x.
  - V is the set of cities/vertices in given graph.
  - The aim of TSP is to minimize the cost function.
- **Step 2:**
  - Assume that graph contains n vertices $V_1, V_2, \ldots, V_n$.
  - TSP finds a path covering all vertices exactly once, and the same time it tries to minimize the overall traveling distance.

CONT..

- **Step 3:**
  - Mathematical formula to find minimum distance is stated below:
    - **$C(i,V) = \min \{ d[i,j] + C(j, V - \{ j \}) \}$,** $j \in V$ and $i \notin V$.
- TSP problem possesses the principle of optimality, i.e. for $d[V_1,V_n]$ to be minimum, any intermediate path (V_i,V_j) must be minimum.

PROBLEM

- **Solve the traveling salesman problem with the associated cost adjacency matrix using dynamic programming.**



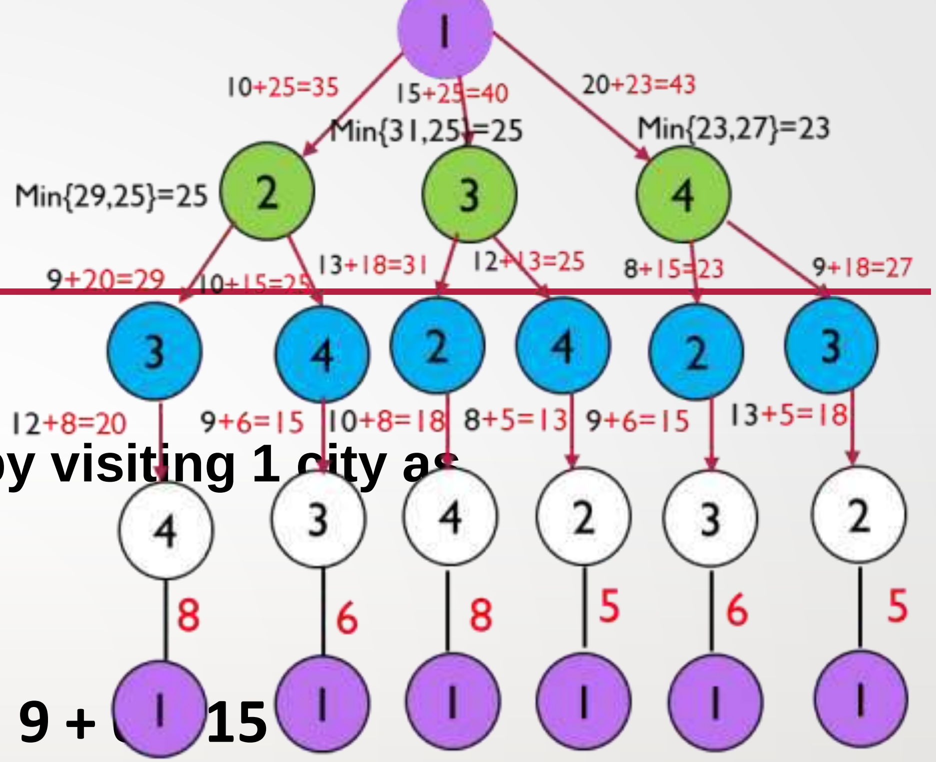
- Table

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

SOLUTION

- Let us start our tour from city 1.
- **Step 1:**
- **Initially, we will find the distance between city 1 and city {2, 3, 4} without visiting any intermediate city.**
  - Cost(x, y, z) represents the distance from x to z and y as an intermediate city.
  - **Cost(2, Φ, 1) = d[2, 1] = 5**
  - **Cost(3, Φ, 1) = d[3, 1] = 6**
  - **Cost(4, Φ , 1) = d[4, 1] = 8**

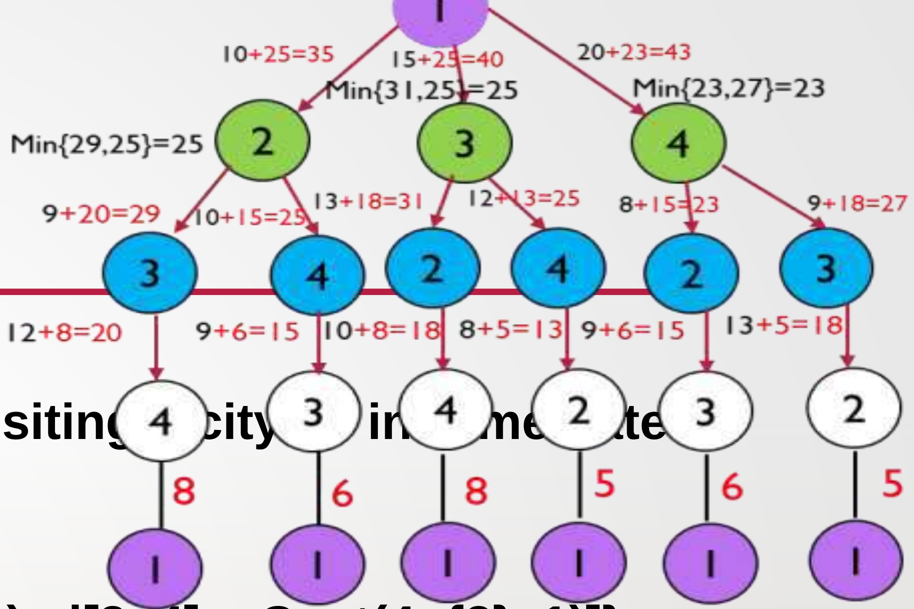
CONT….



- **Step 2:**
- **In this step, we will find the minimum distance by visiting 1 city as intermediate city.**
  - C(i,V) = min { d[i, j] + C(j,V – { j }) },
  - Cost{2, {3}, 1} = d[2, 3] + Cost(3, Φ, 1) = 9 + 6 = 15
  - Cost{2, {4}, 1} = d[2, 4] + Cost(4, Φ, 1) = 10 + 8 = 18
  - Cost{3, {2}, 1} = d[3, 2] + Cost(2, Φ, 1) = 13 + 5 = 18
  - Cost{3, {4}, 1} = d[3, 4] + Cost(4, Φ, 1) = 12 + 8 = 20
  - Cost{4, {3}, 1} = d[4, 3] + Cost(3, Φ, 1) = 9 + 6 = 15
  - Cost{4, {2}, 1} = d[4, 2] + Cost(2, Φ, 1) = 8 + 5 = 13

CONT….

- **Step 3:**
- **In this step, we will find the minimum distance by visiting city in te city.**
  - **C(i,V) = min { d[i, j] + C(j,V – { j }) },**
  - Cost(2, {3, 4}, 1) = min { d[2, 3] + Cost(3, {4}, 1), d[2, 4] + Cost(4, {3}, 1)]}
    = min { [9 + 20], [10 + 15] }
    = min {29, 25} = 25.
  - Cost(3, {2, 4}, 1) = min { d[3, 2] + Cost(2, {4}, 1), d[3, 4] + Cost(4, {2}, 1)]}
    = min { [13 + 18], [12 + 13] }
    = min {31, 25} = 25.
  - Cost(4, {2, 3}, 1) = min{ d[4, 2] + Cost(2, {3}, 1), d[4, 3] + Cost(3, {2}, 1)]}
    = min { [8 + 15], [9 + 18] }
    = min {23, 27} = 23.



CONT….

- **Step 4:**
- **In this step, we will find the minimum distance by visiting 3 city as intermediate city.**
  - **C(i,V) = min { d[i, j] + C(j,V – { j }) },**
  - Cost(1, {2, 3, 4}, 1) = min { d[1, 2] + Cost(2, {3, 4}, 1), d[1, 3] + Cost(3, {2, 4}, 1), d[1, 4] + Cost(4, {2, 3}, 1)}

    = min { 10 + 25, 15 + 25, 20 + 23}

    = min{35, 40, 43} = 35.
- **Thus, minimum length tour would be of 35.**
- **Trace the path:**
  - **Let us find the path that gives the distance of 35.**
  - **Cost(1, {2, 3, 4}, 1) is minimum due to d[1, 2], so move from 1 to 2. Path = {1, 2}.**
  - **Cost(2, {3,4}, 1) is minimum due to d[2,4], so move from 2 to 4. Path = {1, 2, 4}.**
  - **Cost(4, {3}, 1) is minimum due to d[4, 3], so move from 4 to 3. Path = {1, 2, 4, 3}.**
  - **All cities are visited so come back to 1. Hence the optimum tour would be 1 – 2 – 4– 3 – 1.**

PROBLEM2

- **Solve the traveling salesman problem with the associated cost adjacency matrix using dynamic prog**

	1	2	3	4	5
1	-	24	11	10	9
2	8	-	2	5	11
3	26	12	-	8	7
4	11	23	24	-	6
5	5	4	8	11	-

	1	2	3	4	5
1	-	24	11	10	9
2	8	-	2	5	11
3	26	12	-	8	7
4	11	23	24	-	6
5	5	4	8	11	-

SOLUTION

- **Let us start our tour from city 1.**
- **Step 1:**
  - **Initially, we will find the distance between city 1 and city {2, 3, 4, 5} without visiting any intermediate city.**
  - **Cost(x, y, z) represents the distance from x to z and y as an intermediate city.**
  - **Cost(2, Φ, 1) = d[2, 1] = 24**
  - **Cost(3, Φ, 1) = d[3, 1] = 11**
  - **Cost(4, Φ , 1) = d[4, 1] = 10**
  - **Cost(5, Φ , 1) = d[5, 1] = 9**

SOLUTION

	1	2	3	4	5
1	-	24	11	10	9
2	8	-	2	5	11
3	26	12	-	8	7
4	11	23	24	-	6
5	5	4	8	11	-

- **Step 2: In this step, we will find the minimum distance by visiting 1 city as intermediate city.**
- Cost{2, {3}, 1} = d[2, 3] + Cost(3, f, 1) = 2 + 11 = 13
- Cost{2, {4}, 1} = d[2, 4] + Cost(4, f, 1) = 5 + 10 = 15
- Cost{2, {5}, 1} = d[2, 5] + Cost(5, f, 1) = 11 + 9 = 20
- Cost{3, {2}, 1} = d[3, 2] + Cost(2, f, 1) = 12 + 24 = 36
- Cost{3, {4}, 1} = d[3, 4] + Cost(4, f, 1) = 8 + 10 = 18
- Cost{3, {5}, 1} = d[3, 5] + Cost(5, f, 1) = 7 + 9 = 16
- Cost{4, {2}, 1} = d[4, 2] + Cost(2, f, 1) = 23 + 24 = 47
- Cost{4, {3}, 1} = d[4, 3] + Cost(3, f, 1) = 24 + 11 = 35
- Cost{4, {5}, 1} = d[4, 5] + Cost(5, f, 1) = 6 + 9 = 15
- Cost{5, {2}, 1} = d[5, 2] + Cost(2, f, 1) = 4 + 24 = 28
- Cost{5, {3}, 1} = d[5, 3] + Cost(3, f, 1) = 8 + 11 = 19
- Cost{5, {4}, 1} = d[5, 4] + Cost(4, f, 1) = 11 + 10 = 21

SOLUTION

	1	2	3	4	5
1	-	24	11	10	9
2	8	-	2	5	11
3	26	12	-	8	7
4	11	23	24	-	6
5	5	4	8	11	-

- **Step 3: In this step, we will find the minimum distance by visiting 2 cities as intermediate city.**
  - **Cost(2, {3, 4}, 1) = min { d[2, 3] + Cost(3, {4}, 1), d[2, 4] + Cost(4, {3}, 1)]}**

    = min { [2 + 18], [5 + 35] }

    = min{20, 40} = 20
  - **Cost(2, {4, 5}, 1) = min { d[2, 4] + Cost(4, {5}, 1), d[2, 5] + Cost(5, {4}, 1)]}**

    = min { [5 + 15], [11 + 21] }

    = min{20, 32} = 20
  - **Cost(2, {3, 5}, 1) = min { d[2, 3] + Cost(3, {4}, 1), d[2, 4] + Cost(4, {3}, 1)]}**

    = min { [2 + 18], [5 + 35] }

    = min{20, 40} = 20

-	24	11	10	9
8	-	2	5	11
26	12	-	8	7
11	23	24	-	6
5	4	8	11	-

SOLUTION

- Cost(3, {2, 4}, 1) = min { d[3, 2] + Cost(2, {4}, 1), d[3, 4] + Cost(4, {2}, 1)]}

  = min { [12 + 15], [8 + 47] }

  = min{27, 55} = 27
- Cost(3, {4, 5}, 1) = min { d[3, 4] + Cost(4, {5}, 1), d[3, 5] + Cost(5, {4}, 1)]}

  = min { [8 + 15], [7 + 21] }

  = min{23, 28} = 23
- Cost(3, {2, 5}, 1) = min { d[3, 2] + Cost(2, {5}, 1), d[3, 5] + Cost(5, {2}, 1)]}

  = min { [12 + 20], [7 + 28] }

  = min{32, 35} = 32

-	24	11	10	9
8	-	2	5	11
26	12	-	8	7
11	23	24	-	6
5	4	8	11	-

SOLUTION

- Cost(4, {2, 3}, 1) = min{ d[4, 2] + Cost(2, {3}, 1), d[4, 3] + Cost(3, {2}, 1)]}

  = min { [23 + 13], [24 + 36] }

  = min{36, 60} = 36

- Cost(4, {3, 5}, 1) = min{ d[4, 3] + Cost(3, {5}, 1), d[4, 5] + Cost(5, {3}, 1)]}

  = min { [24 + 16], [6 + 19] }

  = min{40, 25} = 25

- Cost(4, {2, 5}, 1) = min{ d[4, 2] + Cost(2, {5}, 1), d[4, 5] + Cost(5, {2}, 1)]}

  = min { [23 + 20], [6 + 28] }

  = min{43, 34} = 34

-	24	11	10	9
8	-	2	5	11
26	12	-	8	7
11	23	24	-	6
5	4	8	11	-

SOLUTION

- Cost(5, {2, 3}, 1) = min{ d[5, 2] + Cost(2, {3}, 1), d[5, 3] + Cost(3, {2}, 1)]}

  = min { [4 + 13], [8 + 36] }

  = min{17, 44} = 17

- Cost(5, {3, 4}, 1) = min{ d[5, 3] + Cost(3, {4}, 1), d[5, 4] + Cost(4, {3}, 1)]}

  = min { [8 + 18], [11 + 35] }

  = min{26, 46} = 26

- Cost(5, {2, 4}, 1) = min{ d[5, 2] + Cost(2, {4}, 1), d[5, 4] + Cost(4, {2}, 1)]}

  = min { [4 + 15], [11 + 47] }

  = min{19, 58} = 19

-	24	11	10	9
8	-	2	5	11
26	12	-	8	7
11	23	24	-	6
5	4	8	11	-

SOLUTION

- **Step 4 :**
- **In this step, we will find the minimum distance by visiting 3 cities as intermediate city.**
  - Cost(2, {3, 4, 5}, 1) = min { d[2, 3] + Cost(3, {4, 5}, 1), d[2, 4] + Cost(4, {3, 5}, 1), d[2, 5] + Cost(5, {3, 4}, 1)}

    = min { 2 + 23, 5 + 25, 11 + 36}

    = min{25, 30, 47} = 25
  - Cost(3, {2, 4, 5}, 1) = min { d[3, 2] + Cost(2, {4, 5}, 1), d[3, 4] + Cost(4, {2, 5}, 1), d[3, 5] + Cost(5, {2, 4}, 1)}

    = min { 12 + 20, 8 + 34, 7 + 19}

    = min{32, 42, 26} = 26

-	24	11	10	9
8	-	2	5	11
26	12	-	8	7
11	23	24	-	6
5	4	8	11	-

SOLUTION

- Cost(4, {2, 3, 5}, 1) = min { d[4, 2] + Cost(2, {3, 5}, 1), d[4, 3] + Cost(3, {2, 5}, 1), d[4, 5] + Cost(5, {2, 3}, 1)}

  = min {23 + 30, 24 + 32, 6 + 17}

  = min{53, 56, 23} = 23

- Cost(5, {2, 3, 4}, 1) = min { d[5, 2] + Cost(2, {3, 4}, 1), d[5, 3] + Cost(3, {2, 4}, 1), d[5, 4] + Cost(4, {2, 3}, 1)}

  = min {4 + 30, 8 + 27, 11 + 36}

  = min{34, 35, 47} = 34

-	24	11	10	9
8	-	2	5	11
26	12	-	8	7
11	23	24	-	6
5	4	8	11	-

SOLUTION

- **Step 5 : In this step, we will find the minimum distance by visiting 4 cities as an intermediate city.**
- **Cost(1, {2, 3, 4, 5}, 1) = min { d[1, 2] + Cost(2, {3, 4, 5}, 1), d[1, 3] + Cost(3, {2, 4, 5}, 1), d[1, 4] + Cost(4, {2, 3, 5}, 1) , d[1, 5] + Cost(5, {2, 3, 4}, 1)}**

  = min { 24 + 25, 11 + 26, 10 + 23, 9 + 34 }

  = min{49, 37, 33, 43} = 33
- **Thus, minimum length tour would be of 33.**

-	24	11	10	9
8	-	2	5	11
26	12	-	8	7
11	23	24	-	6
5	4	8	11	-

SOLUTION

- **Trace the path:**
  - **Let us find the path that gives the distance of 33.**
  - **Cost(1, {2, 3, 4, 5}, 1) is minimum due to d[1, 4], so move from 1 to 4. Path = {1, 4}.**
  - **Cost(4, {2, 3, 5}, 1) is minimum due to d[4, 5], so move from 4 to 5. Path = {1, 4, 5}.**
  - **Cost(5, {2, 3}, 1) is minimum due to d[5, 2], so move from 5 to 2. Path = {1, 4, 5, 2}.**
  - **Cost(2, {3}, 1) is minimum due to d[2, 3], so move from 2 to 3. Path = {1, 4, 5, 2, 3}.**
  - **All cities are visited so come back to 1. Hence the optimum tour would be 1 – 4 – 5 – 2 – 3 – 1.**