Problem Solving Strategies and Computational Approaches SCS1304

Handout 8 : Branch Bound

Prof Prasad Wimalaratne PhD(Salford),SMIEEE

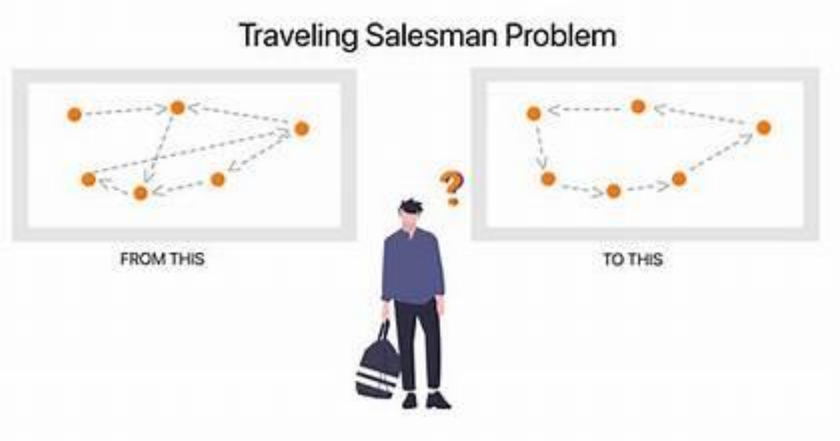
Branch and Bound

- Branch and bound is a systematic method for solving optimization problems, which are often about finding the best solution among many possible options.
  - Unlike backtracking, which focuses on feasibility (finding any solution that satisfies the problem's constraints), branch and bound aims to find the optimal solution by efficiently exploring the entire solution space.
- This technique involves dividing the problem into smaller parts (branching) and then evaluating these parts to decide whether to explore further or discard them (bounding).
  - The "bound" part uses logical tests or bounds to eliminate suboptimal paths in the search process, hence speeding up the solution finding.
- For instance, consider the problem of finding the shortest traveling path through several cities.
  - Here, the algorithm would explore routes (branching) and use bounds (like the minimum distance already traveled) to discard longer routes early in the process.

https://www.naukri.com/code360/library/difference-between-backtracking-and-branch-and-bound

Examples:

- The example problems that can be solved by using backtracking are:
  - 8 Queens problem
  - Knapsack problem using backtracking
  - Hamiltonian Path
- An example problem that can be solved by using branch and bound is:
  - Travelling Salesman Problem

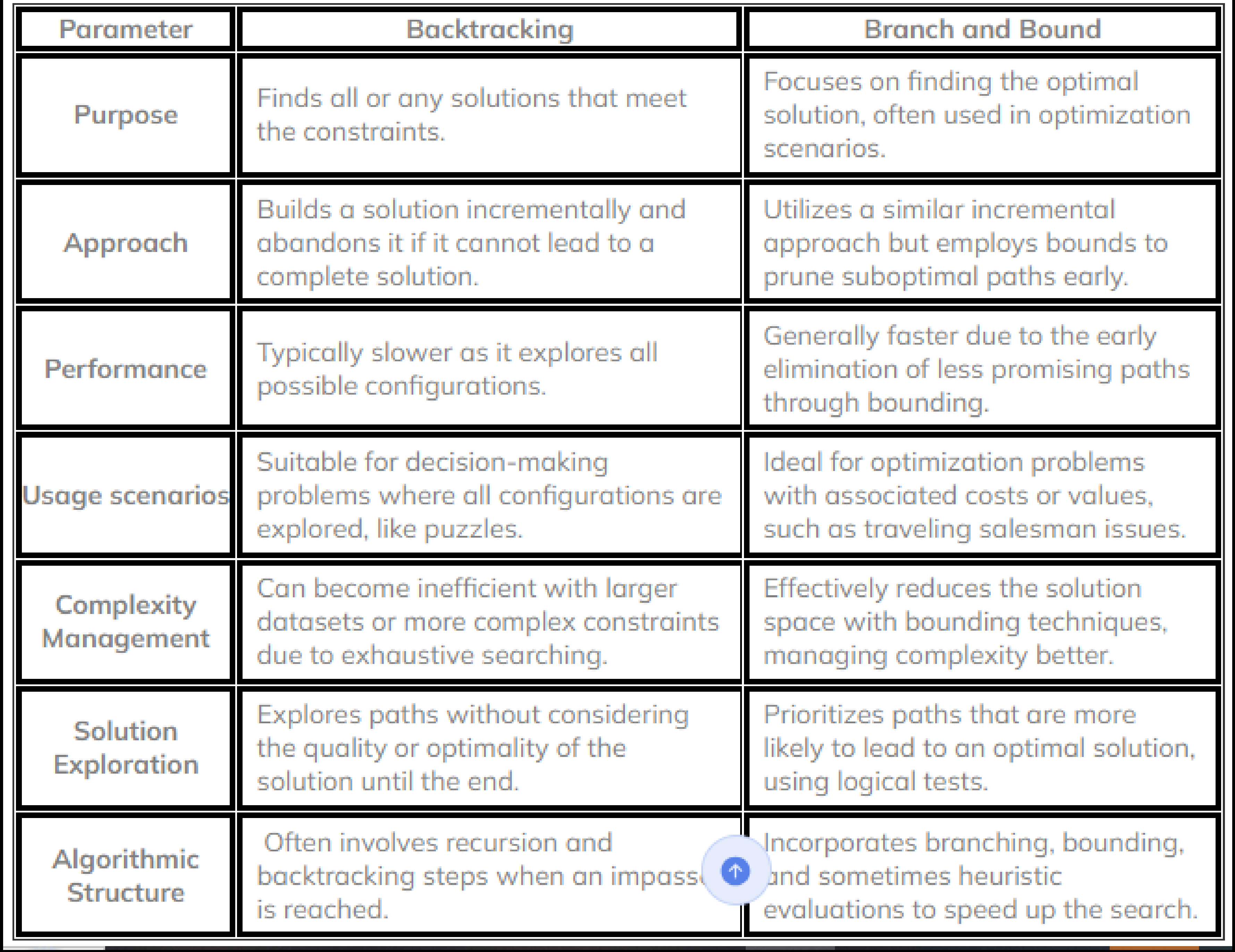


Branch and Bound

- Branch and Bound is similar to Backtracking(last lecture), but differs!
  - Backtracking uses DFS to generate state sapce tree
  - Backtracking can be used to find max or min , but inefficient as all need to be found, then can take max or min – hence avoid Backtracking optimization problems
  - Branch and Bound only generate branches which will lead to a fruitful solution and NOT generate all branches like Backtracking
- The concept branch and bound and backtracking follow the Brute force method and generate the state space tree.
- But both of them follows different approaches.

Backtracking vs Branch and Bound

- Difference Between Backtracking and Branch and Bound
- Unlike backtracking, which focuses on feasibility (finding any solution that satisfies the problem's constraints), branch and bound aims to find the optimal solution by efficiently exploring the entire solution space.



Parameter	Backtracking	Branch and Bound
Purpose	Finds all or any solutions that meet the constraints.	Focuses on finding the optimal solution, often used in optimization scenarios.
Approach	Builds a solution incrementally and abandons it if it cannot lead to a complete solution.	Utilizes a similar incremental approach but employs bounds to prune suboptimal paths early.
Performance	Typically slower as it explores all possible configurations.	Generally faster due to the early elimination of less promising paths through bounding.
Usage scenarios	Suitable for decision-making problems where all configurations are explored, like puzzles.	Ideal for optimization problems with associated costs or values, such as traveling salesman issues.
Complexity Management	Can become inefficient with larger datasets or more complex constraints due to exhaustive searching.	Effectively reduces the solution space with bounding techniques, managing complexity better.
Solution Exploration	Explores paths without considering the quality or optimality of the solution until the end.	Prioritizes paths that are more likely to lead to an optimal solution, using logical tests.
Algorithmic Structure	Often involves recursion and backtracking steps when an impasse is reached.	Incorporates branching, bounding, and sometimes heuristic evaluations to speed up the search.

https://www.naukri.com/code360/library/difference-between-backtracking-and-branch-and-bound

N Queen Problem

- https://www.javatpoint.com/branch-and-bound
- https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound

- Branch and bound algorithms are used to find the optimal solution for combinatory, discrete, and general mathematical optimization problems.
- A branch and bound algorithm provides an optimal solution to an NP-Hard problem by exploring the entire search space.
  - NP-Hard problems are very difficult problems for which no efficient (polynomial-time) algorithm is known. Solving them quickly for large inputs is practically impossible. Example: TSP (Traveling Salesman Problem) is NP-Hard
- Through the exploration of the entire search space, a branch and bound algorithm identifies possible candidates for solutions step-by-step.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound

- There are many optimization problems in computer science, many of which have a finite number of the feasible shortest path in a graph or minimum spanning tree that can be solved in polynomial time.
- Typically, these problems require a worst-case scenario of all possible permutations.
- The branch and bound algorithm create branches and bounds for the best solution.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: search techniques

- Different search techniques in branch and bound:
- The Branch algorithms incorporate different search techniques to traverse a state space tree. Different search techniques used in B&B are listed below:
    1. LC search
    2. BFS
    3. DFS

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound : search techniques

- 1. LC search (Least Cost Search):
  - It uses a heuristic cost function to compute the bound values at each node. Nodes are added to the list of live nodes as soon as they get generated.
  - The node with the least value of a cost function selected as a next E-node. (.e E-node = "Live node currently being expanded.“)
- 2.BFS(Breadth First Search):
  - It is also known as a FIFO search.
  - It maintains the list of live nodes in first-in-first-out order i.e, in a queue, The live nodes are searched in the FIFO order to make them next E-nodes.
- 3. DFS (Depth First Search):
  - It is also known as a LIFO search.
  - It maintains the list of live nodes in last-in-first-out order i.e. in a stack.
  - The live nodes are searched in the LIFO order to make them next E-nodes.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound

- When to apply Branch and Bound Algorithm?
  - It is appropriate to use a branch and bound approach if the given problem is a discrete optimization.
    - i.e choose the best solution (minimum cost, maximum profit, shortest path, etc.), from a finite or countable set of possibilities (not continuous).
  - Discrete optimization refers to problems in which the variables belong to the discrete set.
    - Examples of such problems include 0-1 Integer Programming and Network Flow problems.
  - When it comes to combinatory optimization problems, branch and bound work well.
  - An optimization problem is optimized by combinatory optimization by finding its maximum or minimum based on its objective function.
    - The combinatory optimization problems include Boolean Satisfiability and Integer Linear Programming.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: Basic Concepts

- Basic Concepts of Branch and Bound:
- ‣ Generation of a state space tree:
  - As in the case of backtracking, B&B generates a state space tree to efficiently search the solution space of a given problem instance.
  - In B&B, all children of an E-node (E-node = "Live node currently being expanded.“) in a state space tree are produced before any live node gets converted in an E-node
  - Thus, the E-node remains an E-node until i becomes a dead node.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound

- Evaluation of a candidate solution:
- Unlike backtracking, B&B needs additional factors evaluate a candidate solution:
    1. A way to assign a bound on the best values of the given criterion functions to each node in a state space tree: It is produced by the addition of further components to the partial solution given by that node.
    2. The best values of a given criterion function obtained so far: It describes the upper bound for the maximization problem and the lower bound for the minimization problem.
- A feasible solution is defined by the problem states that satisfy all the given constraints.
- An optimal solution is a feasible solution, which produces the best value of a given objective function.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: Bounding function

- Bounding function :
- It optimizes the search for a solution vector in the solution space of a given problem instance.
- It is a heuristic function that evaluates the lower and upper bounds on the possible solutions at each node.
- The bound values are used to search the partial solutions leading to an optimal solution.
- If a node does not produce a solution better than the best solution obtained thus far, then it is abandoned without further exploration.
- The algorithm then branches to another path to get a better solution.
- The desired solution to the problem is the value of the best solution produced so far.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound

- The 3 reasons to dismiss a search path at the current node :
  1. The bound value of the node is (a) lower than the upper bound in the case of the maximization problem and (b) higher than the lower bound in the case of the minimization problem.

     (i.e. the bound value of the node is not better than the value of the best solution obtained until that node).
  2. The node represents infeasible solutions, violation of the constraints of the problem.
  3.]The node represents a subset of a feasible solution containing a single point. In this case, if the latest solution is better than the best solution obtained so far, the best solution is modified to the value of a feasible solution at that node.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound

- Types of Branch and Bound Solutions:
- The solution of the B&B problem can be represented in two ways:
  1. Variable size solution: Using this solution, we can find the subset of the given set that gives the optimized solution to the given problem.
     - For example, if we have to select a combination of elements from {A, B, C, D} that optimizes the given problem, and it is found that A and B together give the best solution, then the solution will be {A, B}.
  2. Fixed-size solution: There are 0s and 1s in this solution, with the digit at the ith position indicating whether the ith element should be included, for the above example, the solution will be given by {1, 1, 0, 0}, here 1 represent that we have select the element which at ith position and 0 represent we do NOT select the element at ith position.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Example :Types of Branch and Bound Solutions

- Example :
- Jobs = {j1, j2, j3, j4} => Assume j1 and j2 together give the best solution
- The solutions can be written in two ways, as shown below:
- If we want to do jobs j1 and j2, the solution can be represented :
  - Variable Size Solution
    - The subsets of jobs are the first way to represent the solutions.
    - S1 = {j1, j2},
  - Fixed Size Solution
    - The second way to represent the solution is that the first and second jobs are completed, and the third and the fourth jobs are NOT completed .
    - S2 = {1, 0, 0, 1},

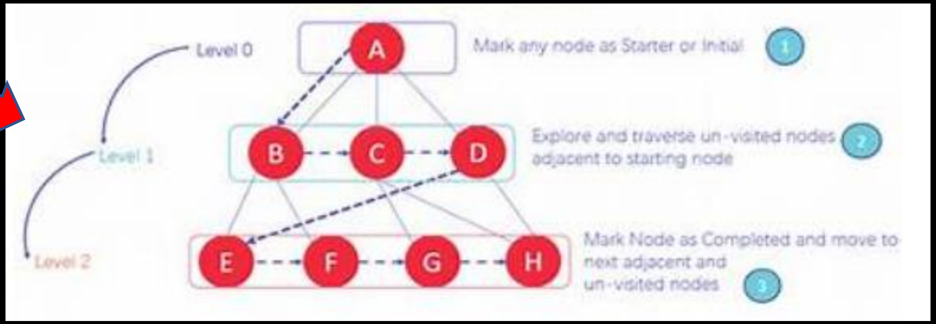
Branch Bound: Classification

- Classification of Branch and Bound Problems:
- The Branch and Bound method can be classified into three types based on the order in which the state space tree is searched.
    1. FIFO Branch and Bound
    2. LIFO Branch and Bound
    3. Least Cost-Branch and Bound
- To denote the solutions in these methods, we will use the variable solution method. i.e {A,B} etc not {1,1,0,0} as describe before

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: FIFO

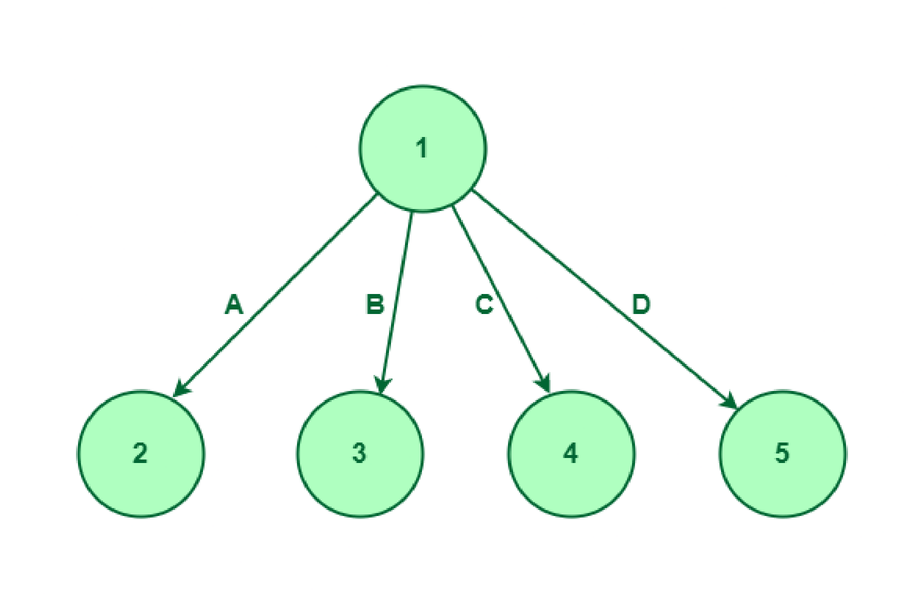
- 1. FIFO Branch and Bound
- First-In-First-Out is an approach to the branch and bound problem that uses the queue approach to create a state-space tree.
- In this case, the breadth-first search (BFS) is performed, that is, the elements at a certain level are all searched, and then the elements at the next level are searched, starting with the first child of the first node at the previous level.
- FIFO Brach and Bound uses a Queue to explore the next node in BFS fashion


Level 0
A
Mark any node as Starter or Initial
1
Level 1
B
C
D
Explore and traverse un-visited nodes adjacent to starting node
2
Level 2
E
F
G
H
Mark Node as Completed and move to next adjacent and un-visited nodes
3

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: FIFO

- For a given set {A, B, C, D}, the state space tree will be constructed as follows :



State Space tree for set {A, B, C, D}

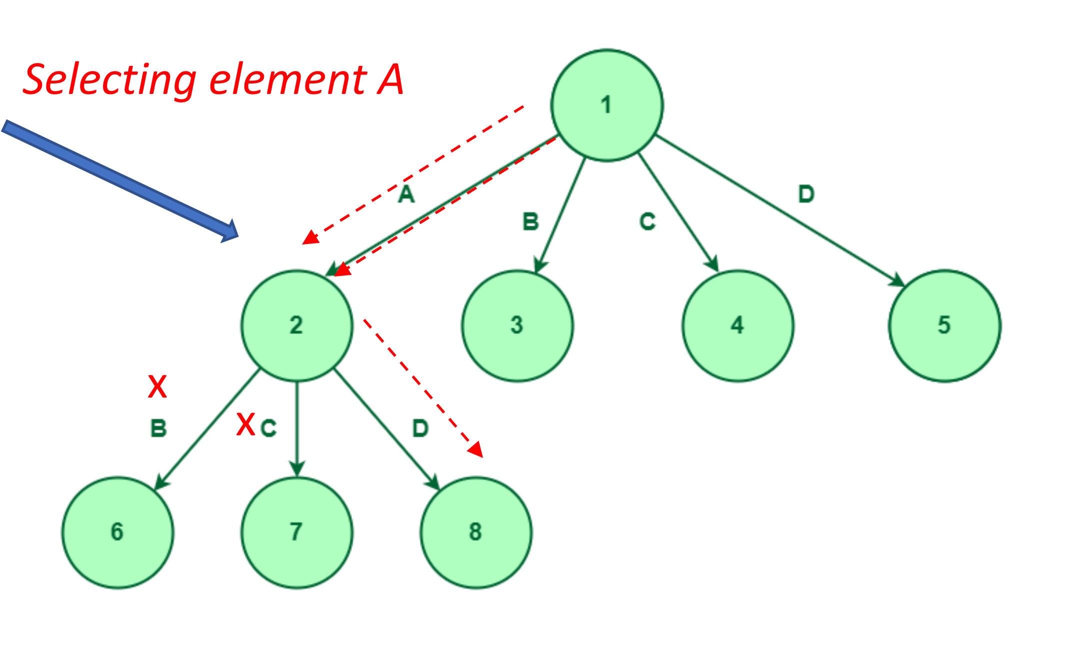
- The above diagram shows that we first consider element A, then element B, then element C and finally we will consider the last element which is D. We are performing BFS while exploring the nodes.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: FIFO

- once the first level is completed. We consider the first element, then we can consider either B, C, or D.
    - example
        - If we select the elements A and D only, then it says that we are selecting elements A and D and we are NOT considering elements B and C

FIFO start point

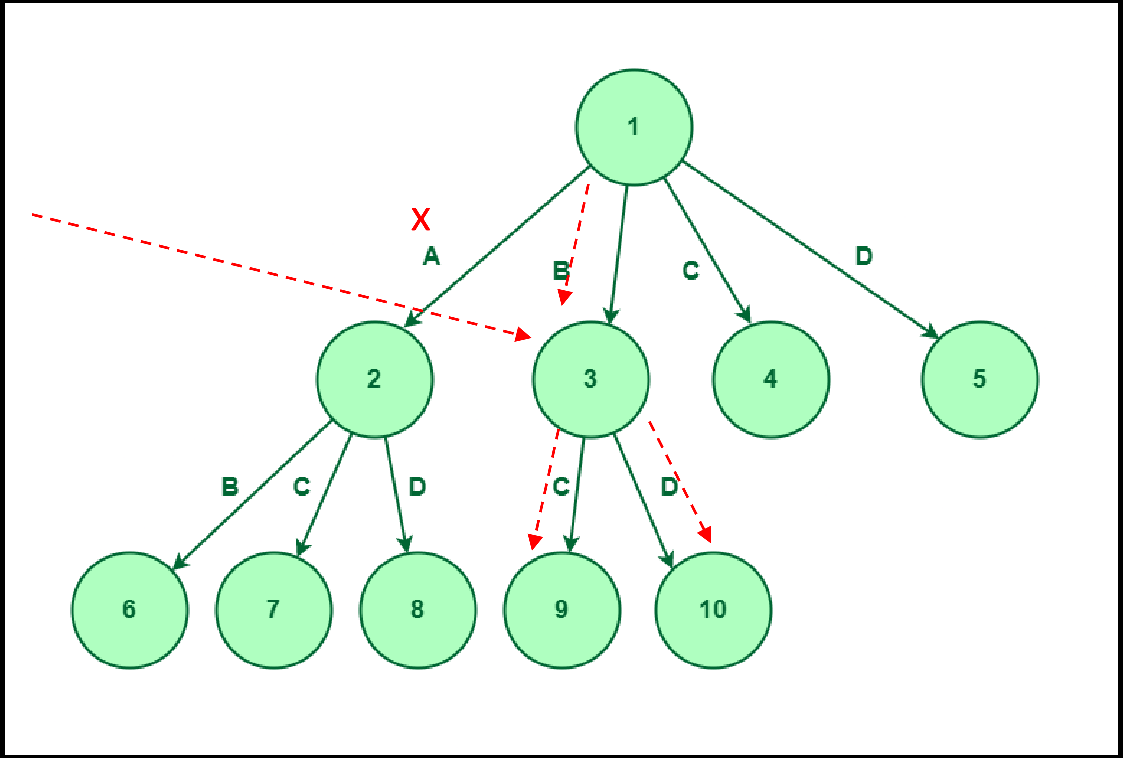


https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound : FIFO

- Now, we will expand node 3, as we have considered element B and NOT considered element A, so, we have two options to explore that is elements C and D. Let us create nodes 9 and 10 for elements C and D, respectively.

Next in the queue is 3
Considered element B and not considered element A

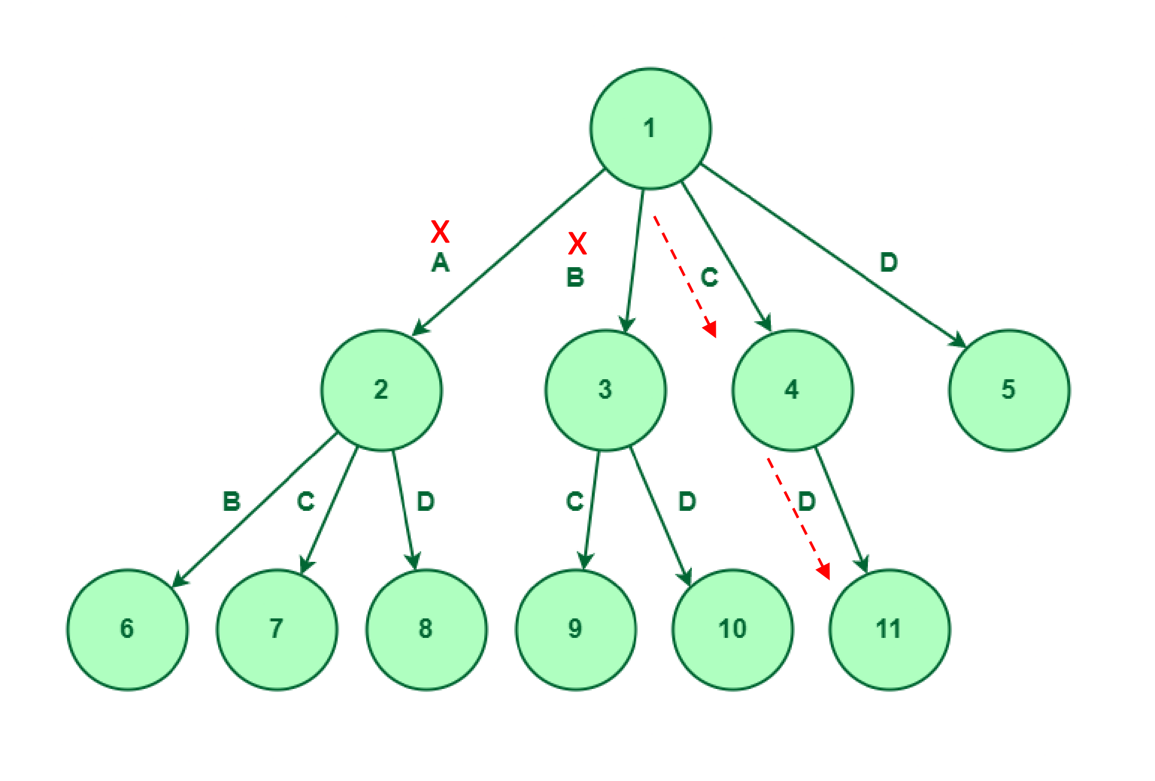


https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound : FIFO

- Now, we will expand node 4 as we have only considered elements C and NOT considered elements A and B, so, we have only one option to explore which is element D. Let us create node 11 for D.

Next in the queue is 4

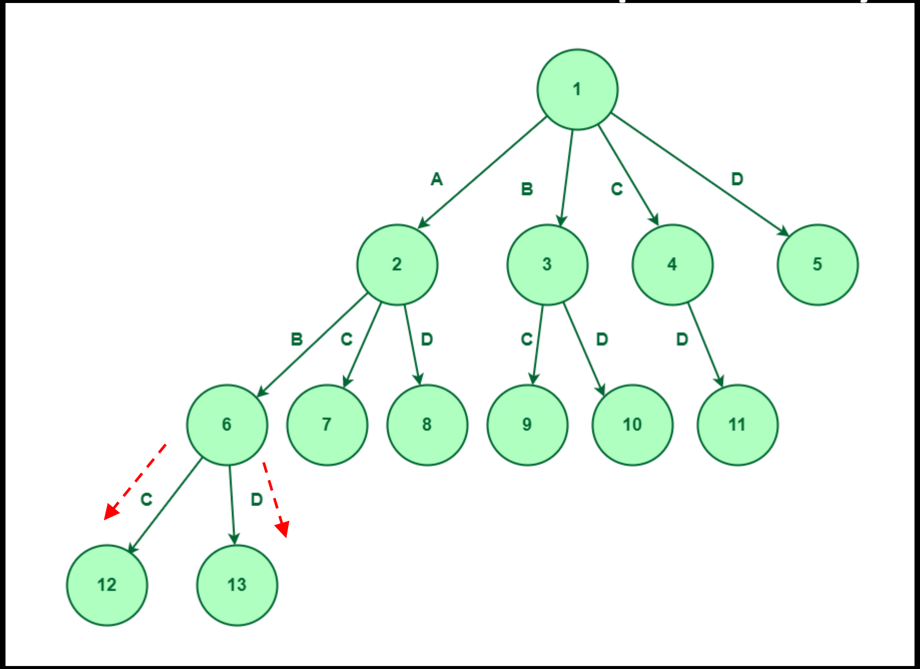


Considered elements C and not considered elements A and B

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound : FIFO

- Till node 5, we have only considered elements D, and NOT selected elements A, B, and C. So, We have no more elements to explore. Therefore on node 5, there won’t be any expansion.
- Now, we will expand node 6 as we have considered elements A and B, so, we have only two options to explore that is element C and D. Let us create node 12 and 13 for C and D respectively.



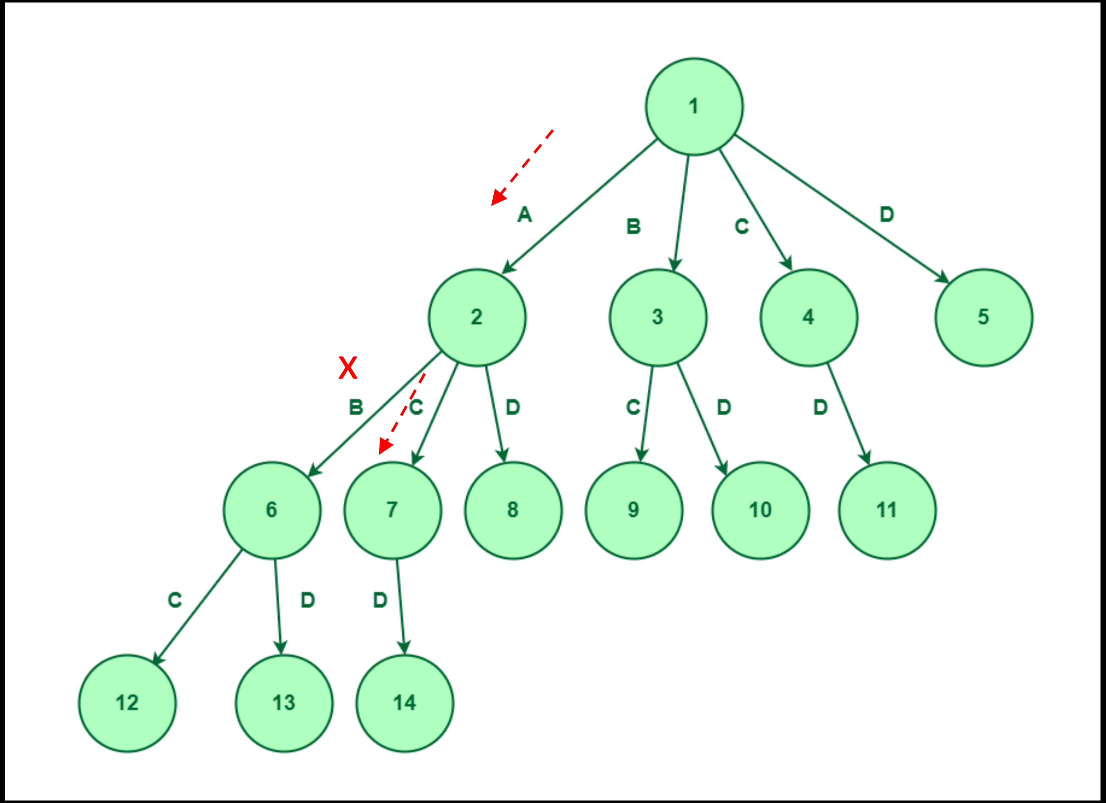
Next in the queue is 6

Expand node 6

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound : FIFO

- Now, we will expand node 7 as we have considered elements A and C and NOT consider element B, so, we have only one option to explore which is element D. Let’s create node 14 for D.



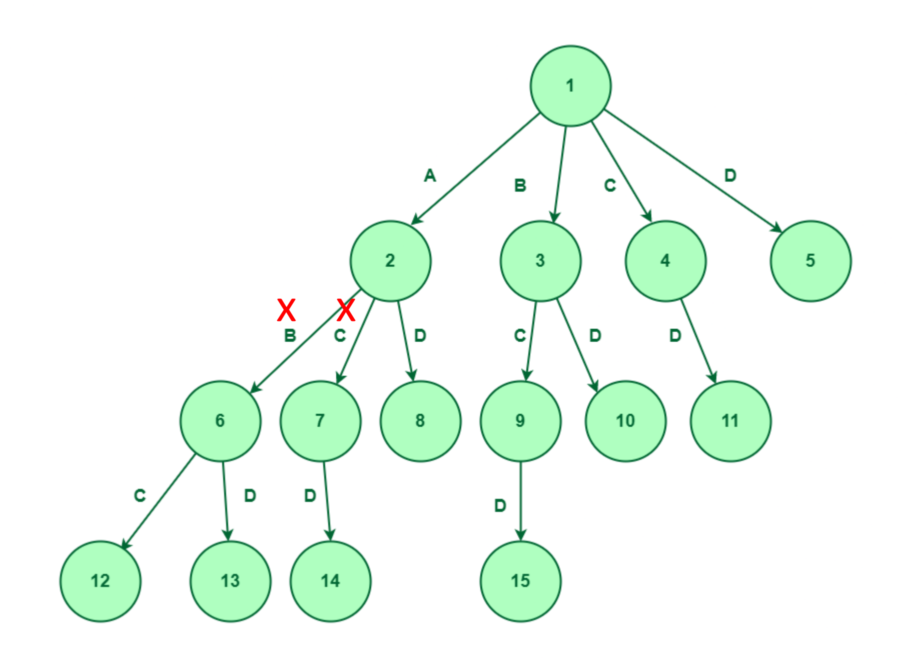
Next in the queue is 7

Expand node 7

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound : FIFO

- Till node 8, we have considered elements A and D, and NOT selected elements B and C, So, We have no more elements to explore, Therefore on node 8, there won't be any expansion.
- Now, we will expand node 9 as we have considered elements B and C and not considered element A, so, we have only one option to explore which is element D. Let's create node 15 for D.



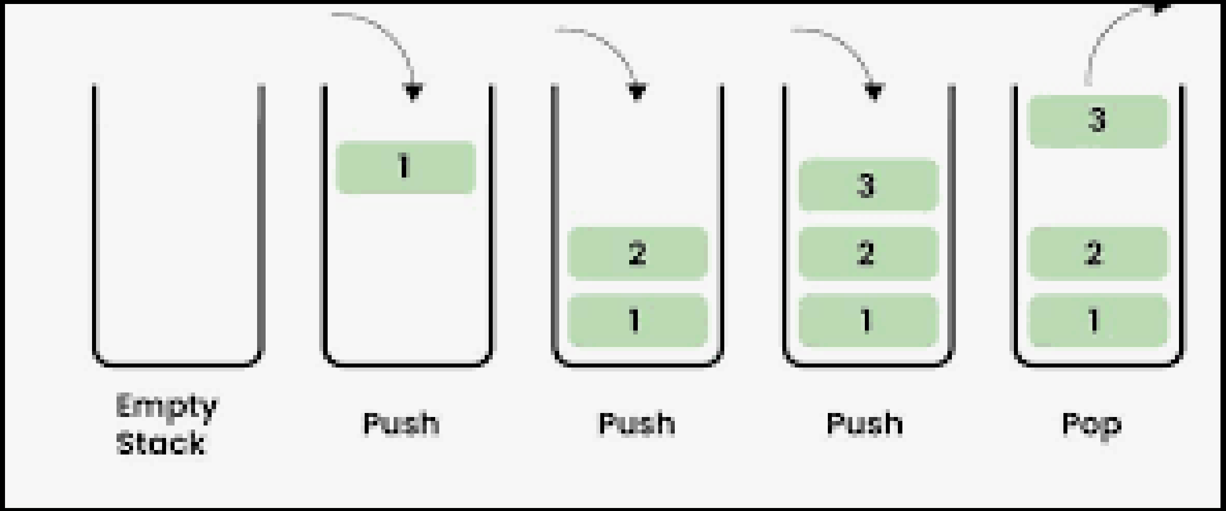
<=state space tree

Expand node 9

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: LIFO

- 2. LIFO Branch and Bound
- The Last-In-First-Out approach for this problem uses stack in creating the state space tree.
    - When nodes are added to a state space tree, they are added to a stack.
    - After all nodes of a level have been added, we pop the topmost element from the stack and explore it.
- LIFO Brach and Bound uses a Stack to explore next node in BFS fashion

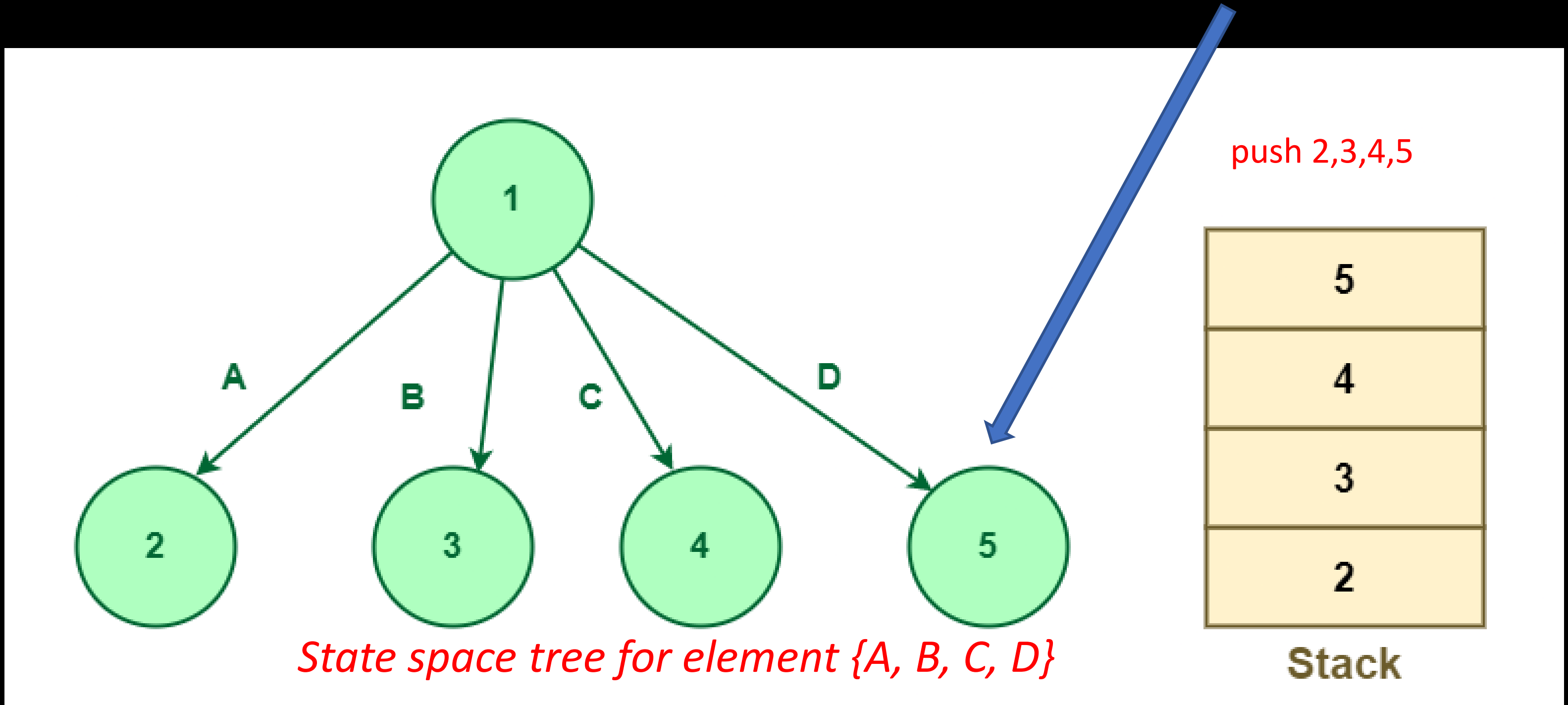


https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: LIFO

- For a given set {A, B, C, D}, the state space tree will be constructed as follows :

LIFO start point



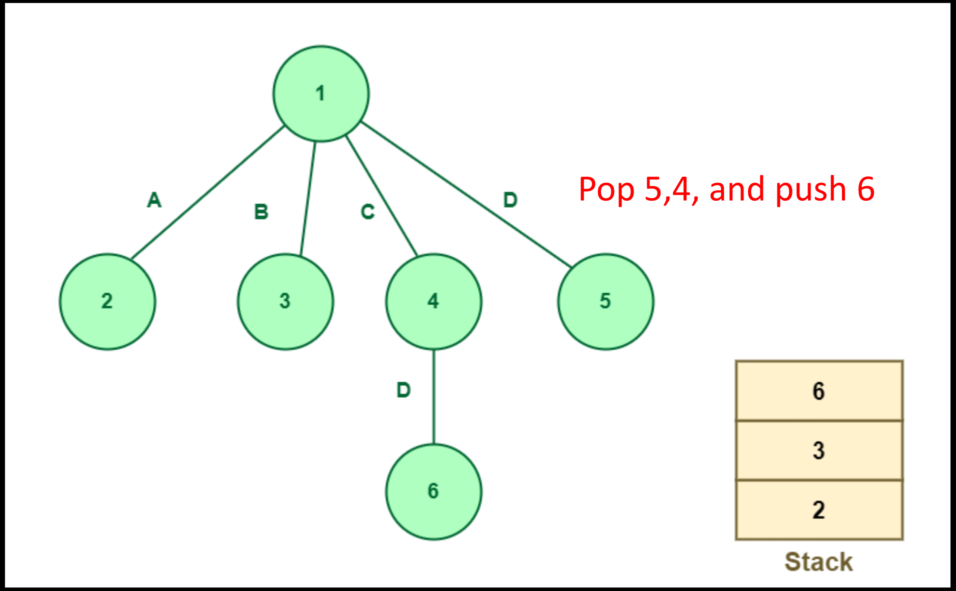
State space tree for element {A, B, C, D}

- first consider element A, then element B, then element C and finally we will consider the last element which is D

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: LIFO

- Now the expansion would be based on the node that appears on the top of the stack. Since node 5 appears on the top of the stack, so we will expand node 5. We will pop out node 5 from the stack. Since node 5 is in the last element, i.e., D so there is no further scope for expansion.
- The next node that appears on the top of the stack is node 4. Pop-out node 4 and expand. On expansion, element D will be considered and node 6 will be added to the stack shown below:



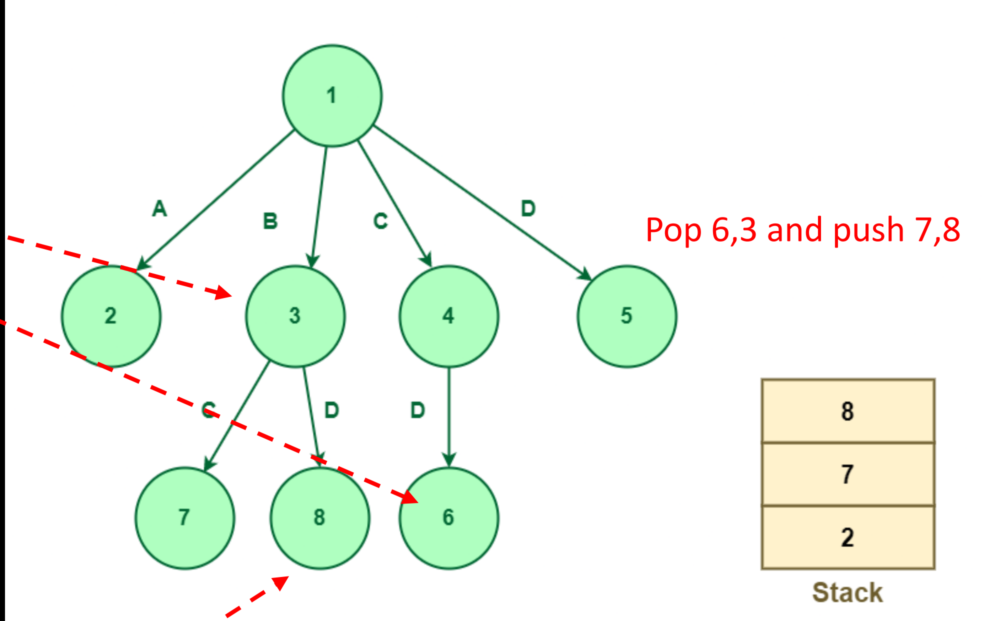
Pop 5,4, and push 6

Expand node 4

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: LIFO

- The next node is 6 which is to be expanded. Pop-out node 6 and expand. Since node 6 is in the last element, i.e., D so there is no further scope for expansion.
- The next node to be expanded is node 3. Since node 3 works on element B so node 3 will be expanded to two nodes, i.e., 7 and 8 working on elements C and D, respectively. Nodes 7 and 8 will be pushed into the stack.
- The next node that appears on the top of the stack is node 8. Pop-out node 8 and expand. Since node 8 works on element D so there is no further scope for the expansion.



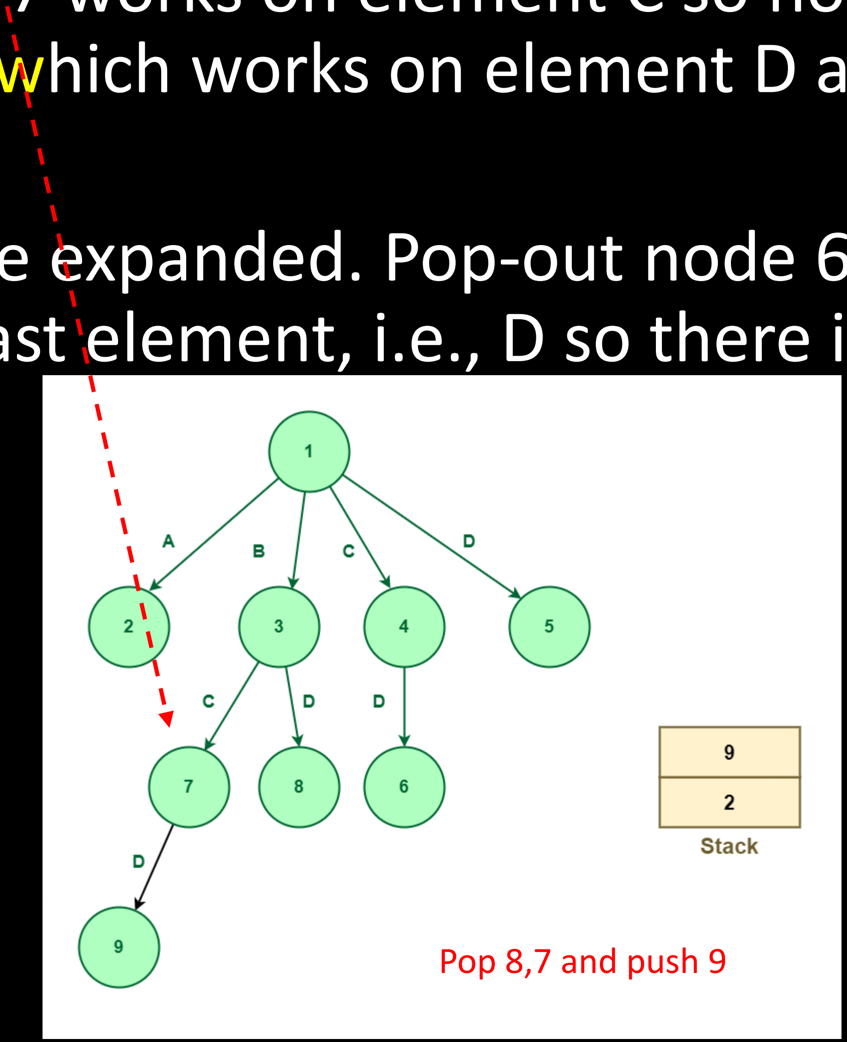
Pop 6,3 and push 7,8

Expand node 3

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: LIFO

- The next node that appears on the top of the stack is node 7. Pop-out node 7 and expand. Since node 7 works on element C so node 7 will be further expanded to node 9 which works on element D and node 9 will be pushed into the stack.
- The next node is 6 which is to be expanded. Pop-out node 6 and expand. Since node 6 is in the last element, i.e., D so there is no further scope for expansion.



Pop 8,7 and push 9

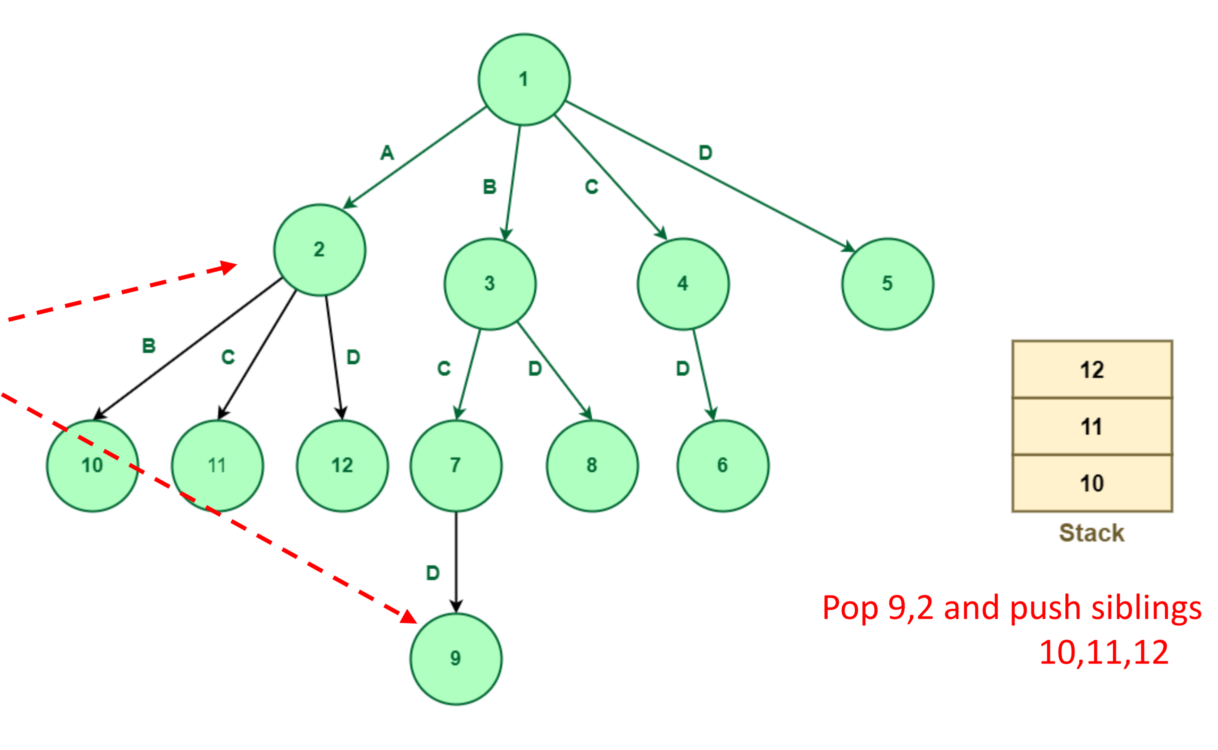
Pop 8,7 and push 9

Expand node 7

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: LIFO

- The next node that appears on the top of the stack is node 9. Since node 9 works on element D, there is no further scope for expansion.
- The next node that appears on the top of the stack is node 2. Since node 2 works on the element A so it means that node 2 can be further expanded. It can be expanded up to three nodes named 10, 11, 12 working on elements B, C, and D respectively. There new nodes will be pushed into the stack shown as below:



12

11

10

Stack

Pop 9,2 and push siblings 10,11,12

Expand node 2

In the above method, we explored all the nodes using the stack that follows the LIFO principle.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: Least Cost-Branch and Bound

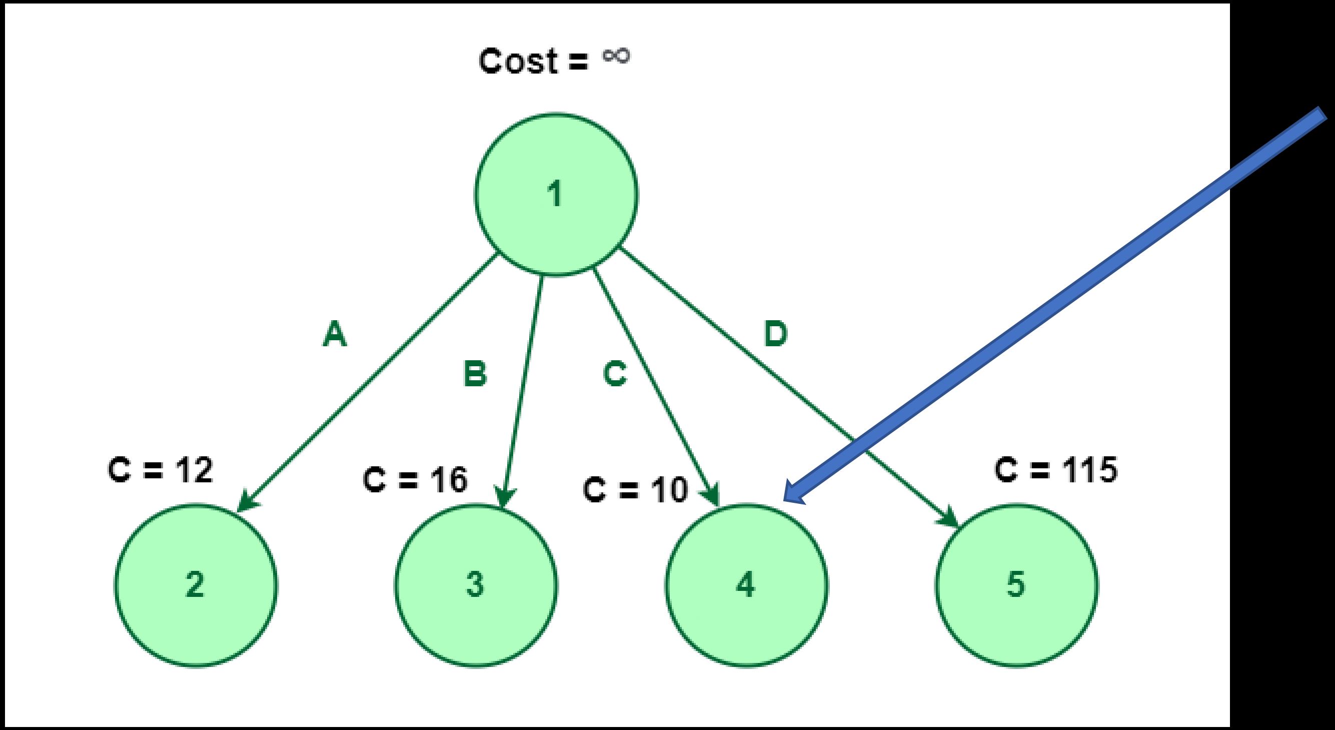
- To explore the state space tree, this method uses the cost function.
- The previous two methods also calculate the cost function at each node, but the cost is not used for further exploration(i.e LIFO and FIFO).
- In this technique, nodes are explored based on their costs. The cost of a node can be defined using the problem, and with the help of the given problem, we can define the cost function.
- Once the cost function is defined, we can define the cost of the node. Now, consider a node whose cost has been determined.
- If this value is greater than U0, this node or its children will not be able to give a solution.
- As a result, we can kill this node and not explore its further branches.
- As a result, this method prevents us from exploring cases that are not worth it, which makes it more efficient for us.
- Generally Faster than LIFO and FIFO approaches

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: Least Cost-Branch and Bound (LCBB)

- Let us first consider node 1 having cost infinity, shown below:
- In the following diagram, node 1 is expanded into four nodes named 2, 3, 4, and 5.

- Usually in LCBB implementations, before computing the actual bound (reduced cost, lower bound, etc.), cost is temporarily initialized to ∞.
- Then, if a valid cost is found, it is updated. If not, it remains ∞
- Nodes with ∞ are effectively excluded from consideration.



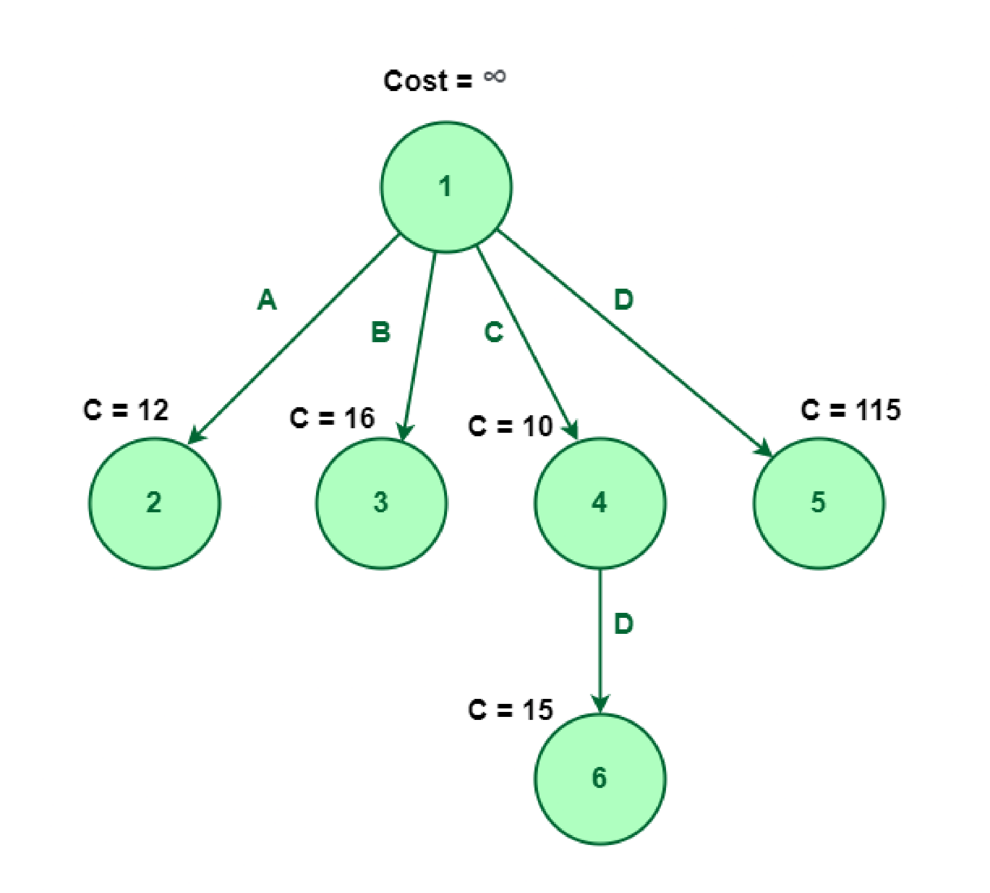
LC start point

Node 1 is expanded into four nodes named 2, 3, 4, and 5

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: Least Cost-Branch and Bound

- Assume that cost of the nodes 2, 3, 4, and 5 are 12, 16, 10, and 115 respectively.
  In this method, we will explore the node which is having the least cost.
- In the above figure, we can observe that the node with a minimum cost is node 4. So, we will explore node 4 having a cost of 10.
- During exploring node 4 which is element C, we can notice that there is only one possible element that remains unexplored which is D (i.e, we already decided not to select elements A, and B).
- So, it will get expanded to one single element D, let's say this node number is 6.



Exploring node 4 which is element C

Now, Node 6 has no element left to explore. So, there is no further scope for expansion. Hence the element {C, D} is the optimal way to choose for the least cost.

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

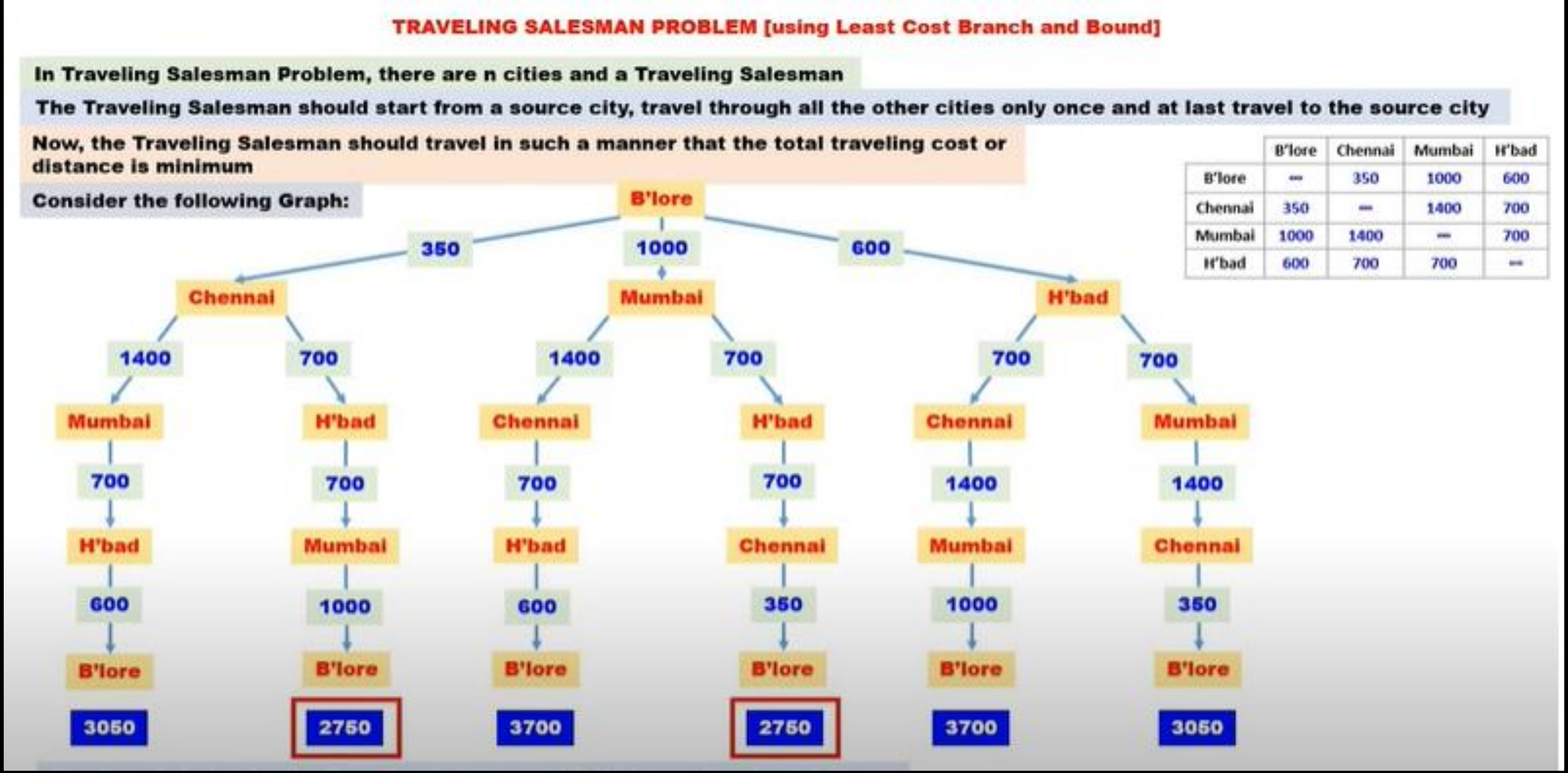
Branch Bound Example TSP

- **Traveling Salesman Problem - Branch and Bound**
- https://www.youtube.com/watch?v=1FEP_sNb62k

- https://www.baeldung.com/cs/branch-and-bound

https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

In Traveling Salesman Problem, there are n cities and a Traveling Salesman

The Traveling Salesman should start from a source city, travel through all the other cities only once and at last travel to the source city

Now, the Traveling Salesman should travel in such a manner that the total traveling cost or distance is minimum

Consider the following Graph:

	B'lore	Chennai	Mumbai	H'bad
B'lore	--	350	1000	600
Chennai	350	--	1400	700
Mumbai	1000	1400	--	700
H'bad	600	700	700	--

B'lore

350 → Chennai → 1400 → Mumbai → 700 → H'bad → 600 → B'lore 3050

350 → Chennai → 700 → H'bad → 700 → Mumbai → 1000 → B'lore 2750

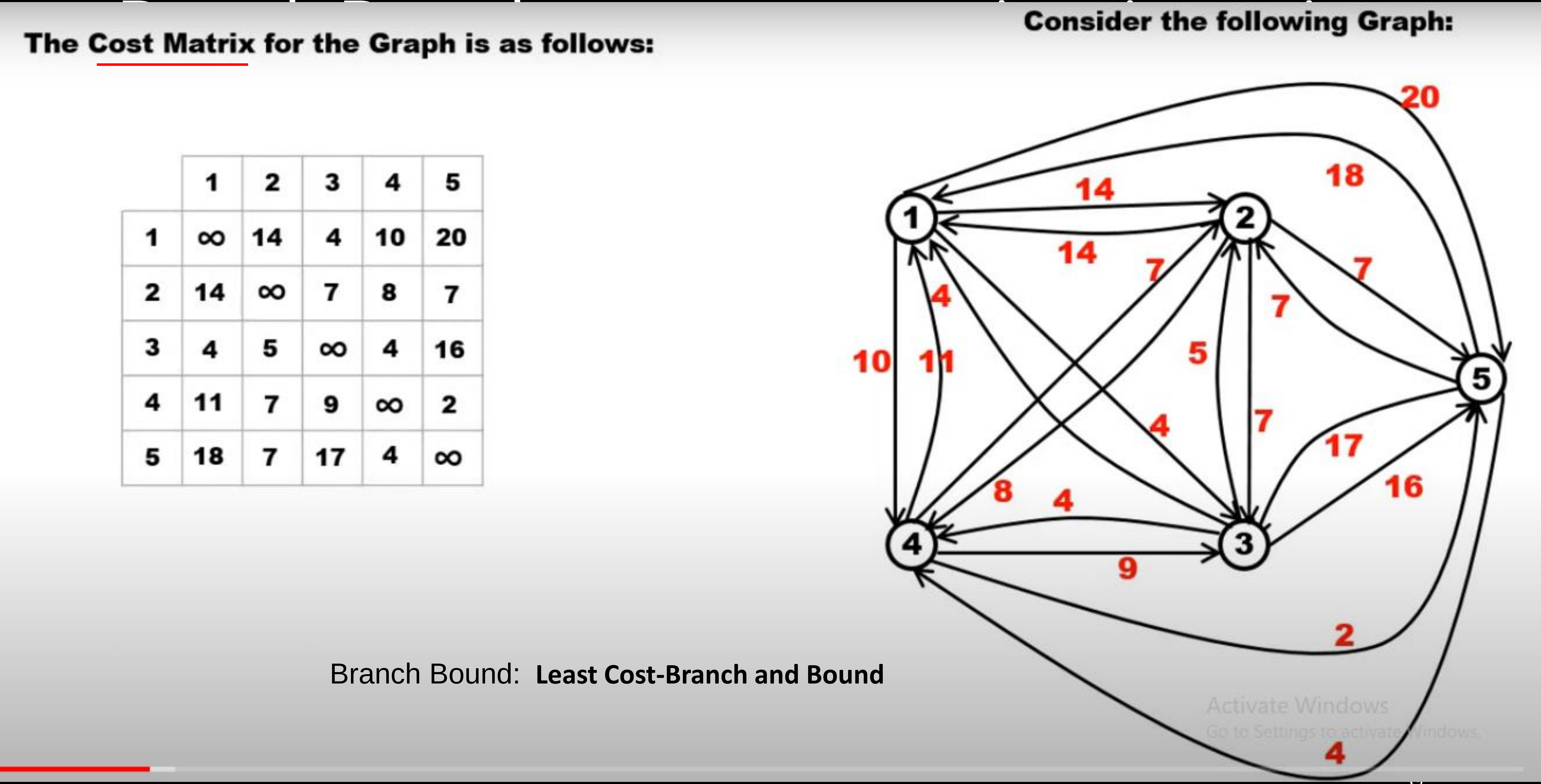
1000 → Mumbai → 1400 → Chennai → 700 → H'bad → 600 → B'lore 3700

1000 → Mumbai → 700 → H'bad → 700 → Chennai → 350 → B'lore 2750

600 → H'bad → 700 → Chennai → 1400 → Mumbai → 1000 → B'lore 3700

600 → H'bad → 700 → Mumbai → 1400 → Chennai → 350 → B'lore 3050

https://www.youtube.com/watch?v=HjSbaKF8Gi0



The Cost Matrix for the Graph is as follows:

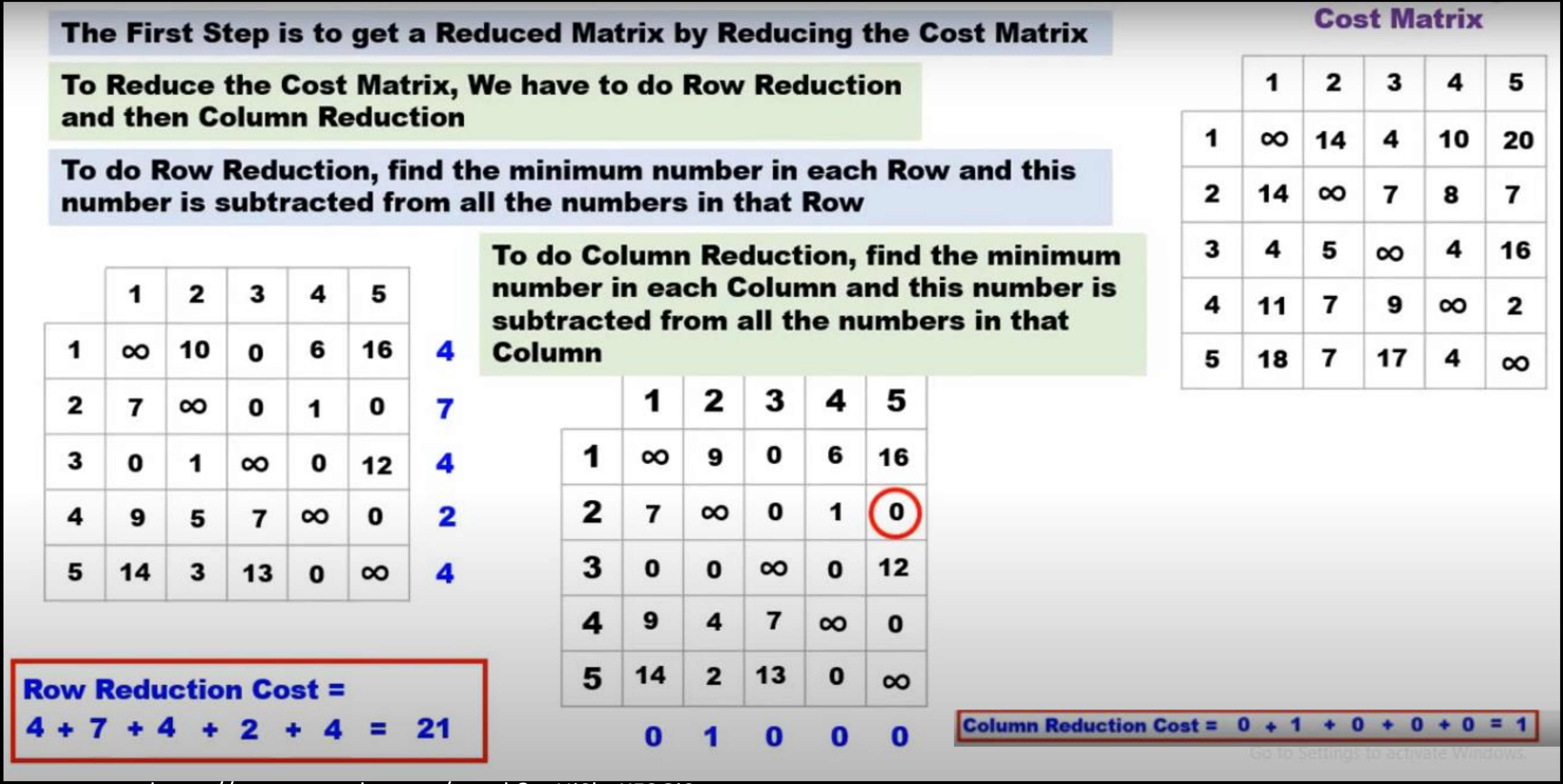
Consider the following Graph:

	1	2	3	4	5
1	∞	14	4	10	20
2	14	∞	7	8	7
3	4	5	∞	4	16
4	11	7	9	∞	2
5	18	7	17	4	∞

Branch Bound: **Least Cost-Branch and Bound**

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



The First Step is to get a Reduced Matrix by Reducing the Cost Matrix

To Reduce the Cost Matrix, We have to do Row Reduction and then Column Reduction

To do Row Reduction, find the minimum number in each Row and this number is subtracted from all the numbers in that Row

To do Column Reduction, find the minimum number in each Column and this number is subtracted from all the numbers in that Column

Cost Matrix

	1	2	3	4	5
1	∞	14	4	10	20
2	14	∞	7	8	7
3	4	5	∞	4	16
4	11	7	9	∞	2
5	18	7	17	4	∞

	1	2	3	4	5	
1	∞	10	0	6	16	4
2	7	∞	0	1	0	7
3	0	1	∞	0	12	4
4	9	5	7	∞	0	2
5	14	3	13	0	∞	4

Row Reduction Cost =
4 + 7 + 4 + 2 + 4 = 21

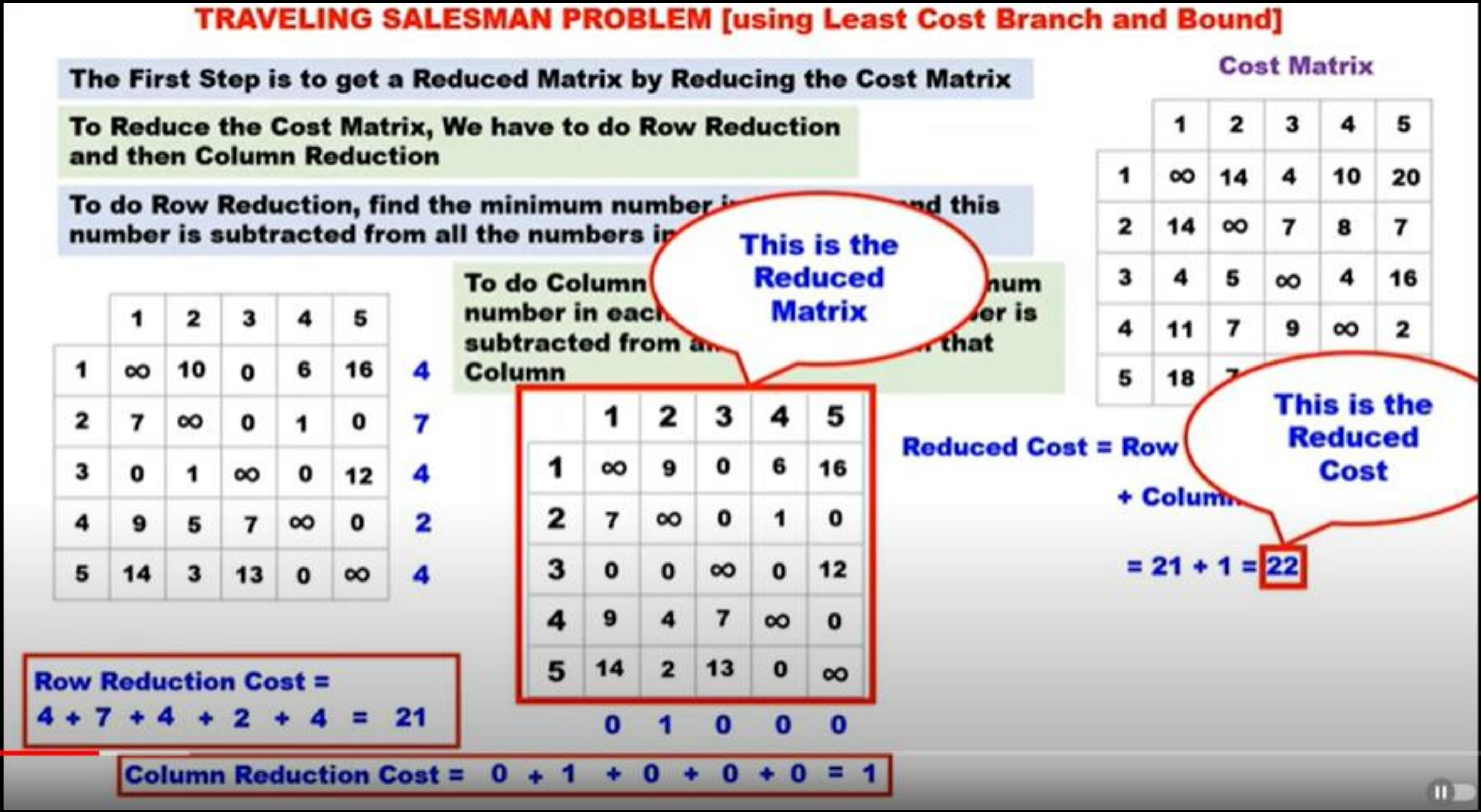
	1	2	3	4	5
1	∞	9	0	6	16
2	7	∞	0	1	0
3	0	0	∞	0	12
4	9	4	7	∞	0
5	14	2	13	0	∞
	0	1	0	0	0

Column Reduction Cost = 0 + 1 + 0 + 0 + 0 = 1

Go to Settings to activate Windows.

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

The First Step is to get a Reduced Matrix by Reducing the Cost Matrix

To Reduce the Cost Matrix, We have to do Row Reduction and then Column Reduction

To do Row Reduction, find the minimum number i nd this number is subtracted from all the numbers i

To do Column num number in eac er is subtracted from a that Column

Cost Matrix

	1	2	3	4	5
1	∞	14	4	10	20
2	14	∞	7	8	7
3	4	5	∞	4	16
4	11	7	9	∞	2
5	18	7	[illegible]	[illegible]	[illegible]

	1	2	3	4	5	
1	∞	10	0	6	16	4
2	7	∞	0	1	0	7
3	0	1	∞	0	12	4
4	9	5	7	∞	0	2
5	14	3	13	0	∞	4

Row Reduction Cost =
4 + 7 + 4 + 2 + 4 = 21

This is the Reduced Matrix

	1	2	3	4	5
1	∞	9	0	6	16
2	7	∞	0	1	0
3	0	0	∞	0	12
4	9	4	7	∞	0
5	14	2	13	0	∞
	0	1	0	0	0

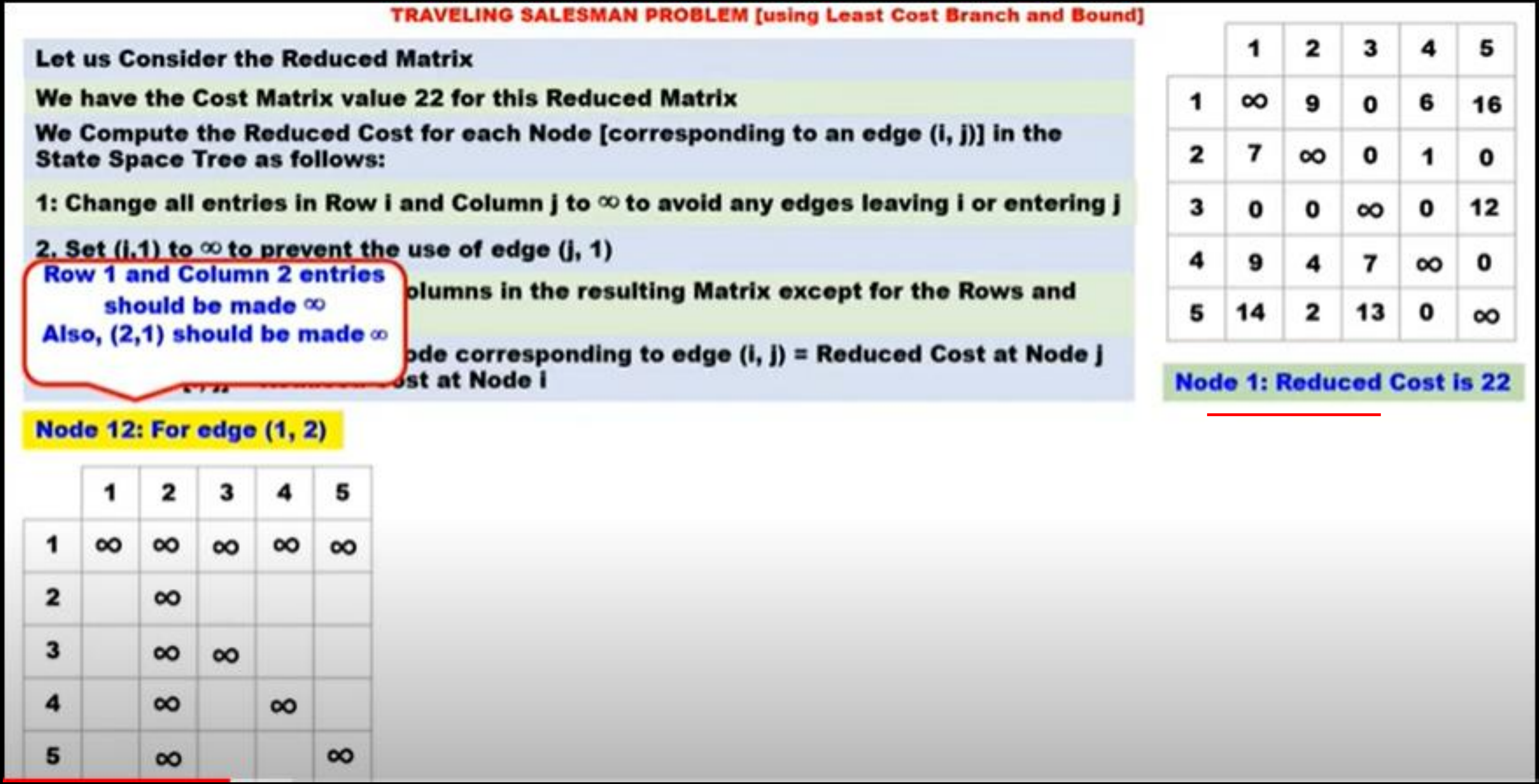
Column Reduction Cost = 0 + 1 + 0 + 0 + 0 = 1

Reduced Cost = Row
+ Colum
= 21 + 1 = 22

This is the Reduced Cost

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Let us Consider the Reduced Matrix

We have the Cost Matrix value 22 for this Reduced Matrix

We Compute the Reduced Cost for each Node [corresponding to an edge (i, j)] in the State Space Tree as follows:

1: Change all entries in Row i and Column j to ∞ to avoid any edges leaving i or entering j

2. Set (j,1) to ∞ to prevent the use of edge (j, 1)

olumns in the resulting Matrix except for the Rows and

ode corresponding to edge (i, j) = Reduced Cost at Node j

st at Node i

Row 1 and Column 2 entries should be made ∞
Also, (2,1) should be made ∞

	1	2	3	4	5
1	∞	9	0	6	16
2	7	∞	0	1	0
3	0	0	∞	0	12
4	9	4	7	∞	0
5	14	2	13	0	∞

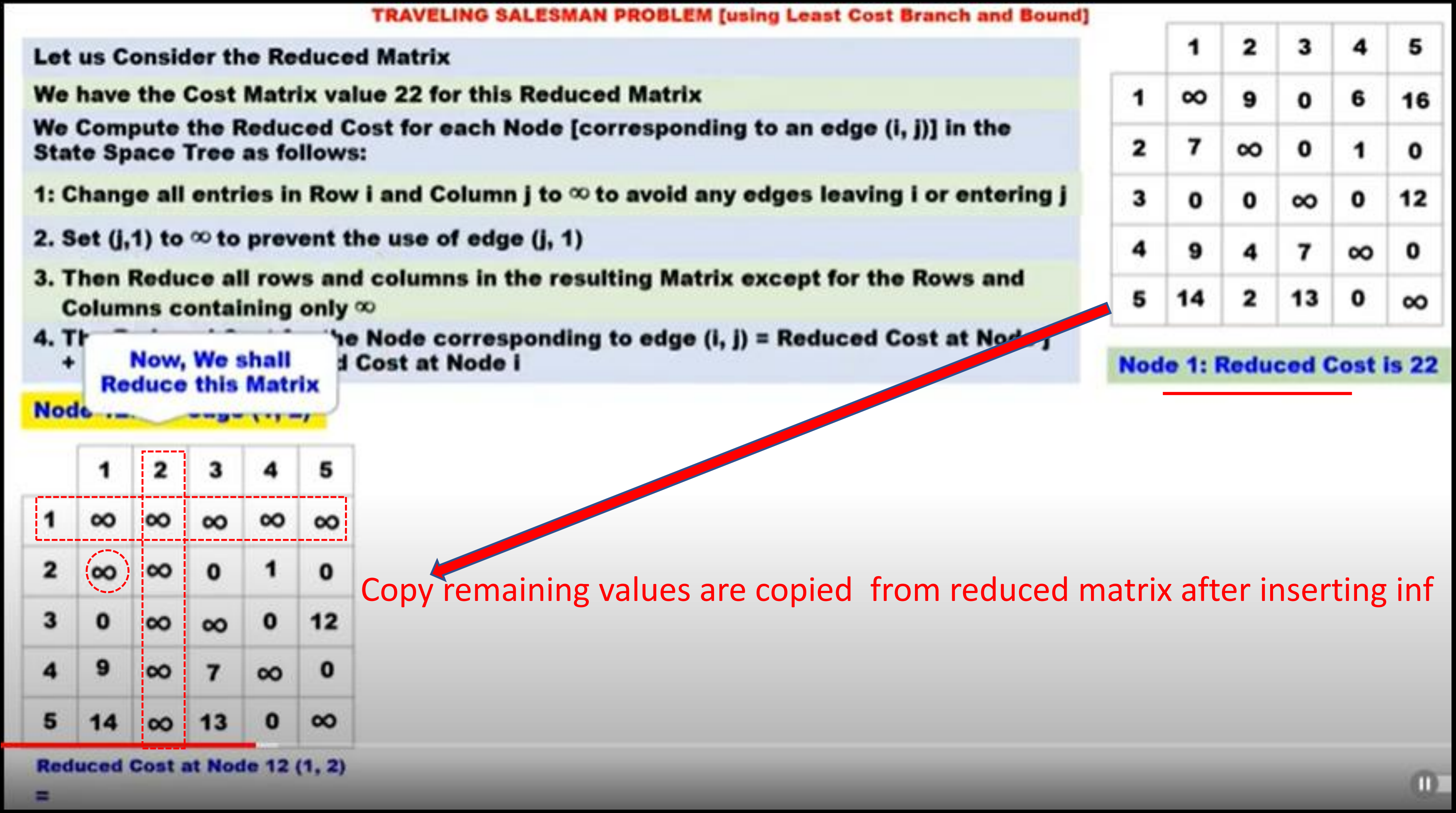
Node 1: Reduced Cost is 22

Node 12: For edge (1, 2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2		∞			
3		∞	∞		
4		∞		∞	
5		∞			∞

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Let us Consider the Reduced Matrix

We have the Cost Matrix value 22 for this Reduced Matrix

We Compute the Reduced Cost for each Node [corresponding to an edge (i, j)] in the State Space Tree as follows:

1: Change all entries in Row i and Column j to ∞ to avoid any edges leaving i or entering j

2. Set (j,1) to ∞ to prevent the use of edge (j, 1)

3. Then Reduce all rows and columns in the resulting Matrix except for the Rows and Columns containing only ∞

4. Th he Node corresponding to edge (i, j) = Reduced Cost at Node j + d Cost at Node i

Now, We shall Reduce this Matrix

Node

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	0	1	0
3	0	∞	∞	0	12
4	9	∞	7	∞	0
5	14	∞	13	0	∞

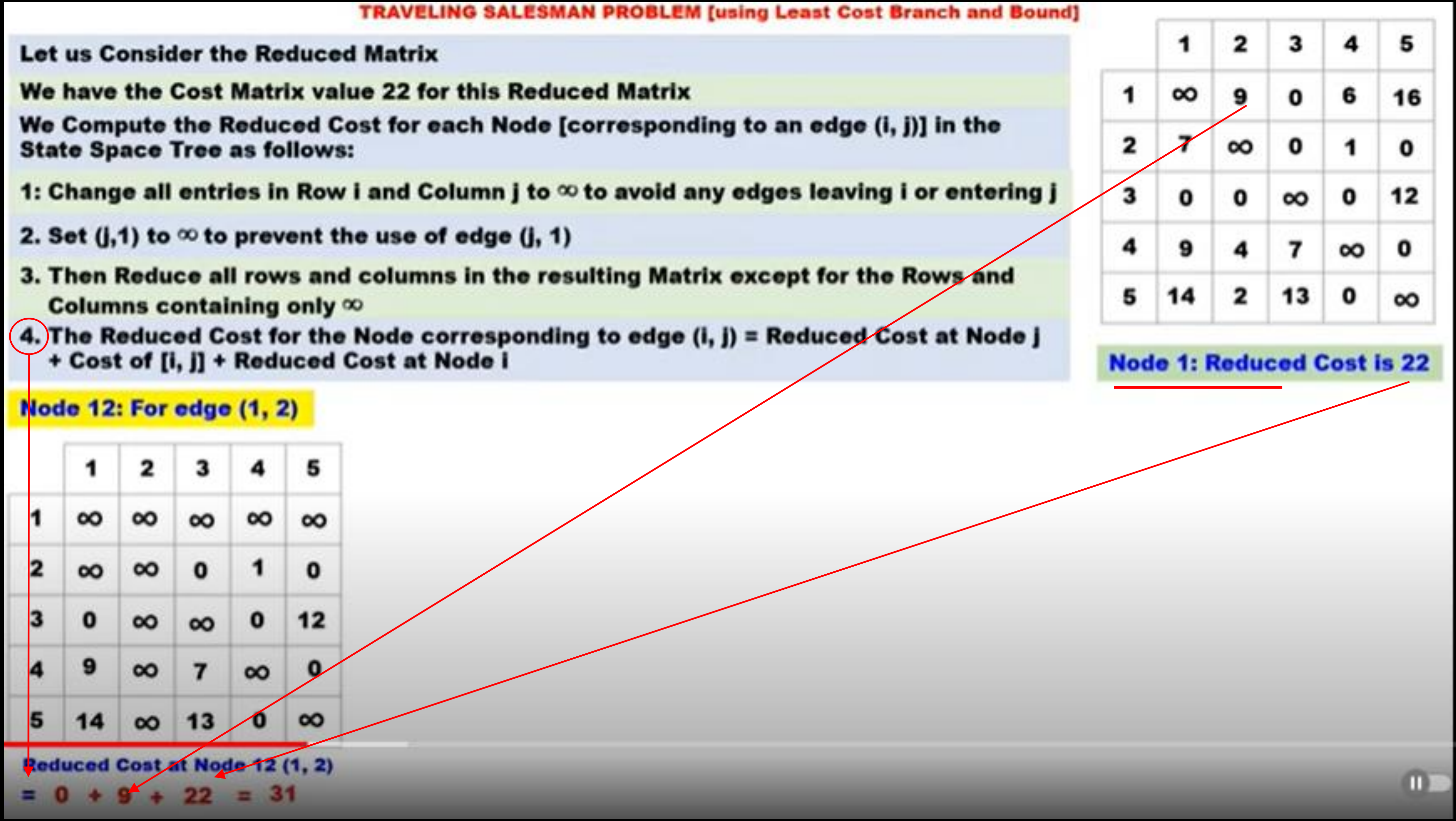
Reduced Cost at Node 12 (1, 2)
=

	1	2	3	4	5
1	∞	9	0	6	16
2	7	∞	0	1	0
3	0	0	∞	0	12
4	9	4	7	∞	0
5	14	2	13	0	∞

Node 1: Reduced Cost is 22

Copy remaining values are copied from reduced matrix after inserting inf

https://www.youtube.com/watch?v=HjSbaKF8Gi0



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Let us Consider the Reduced Matrix

We have the Cost Matrix value 22 for this Reduced Matrix

We Compute the Reduced Cost for each Node [corresponding to an edge (i, j)] in the State Space Tree as follows:

1: Change all entries in Row i and Column j to ∞ to avoid any edges leaving i or entering j

2. Set (j,1) to ∞ to prevent the use of edge (j, 1)

3. Then Reduce all rows and columns in the resulting Matrix except for the Rows and Columns containing only ∞

4. The Reduced Cost for the Node corresponding to edge (i, j) = Reduced Cost at Node j + Cost of [i, j] + Reduced Cost at Node i

	1	2	3	4	5
1	∞	9	0	6	16
2	7	∞	0	1	0
3	0	0	∞	0	12
4	9	4	7	∞	0
5	14	2	13	0	∞

Node 1: Reduced Cost is 22

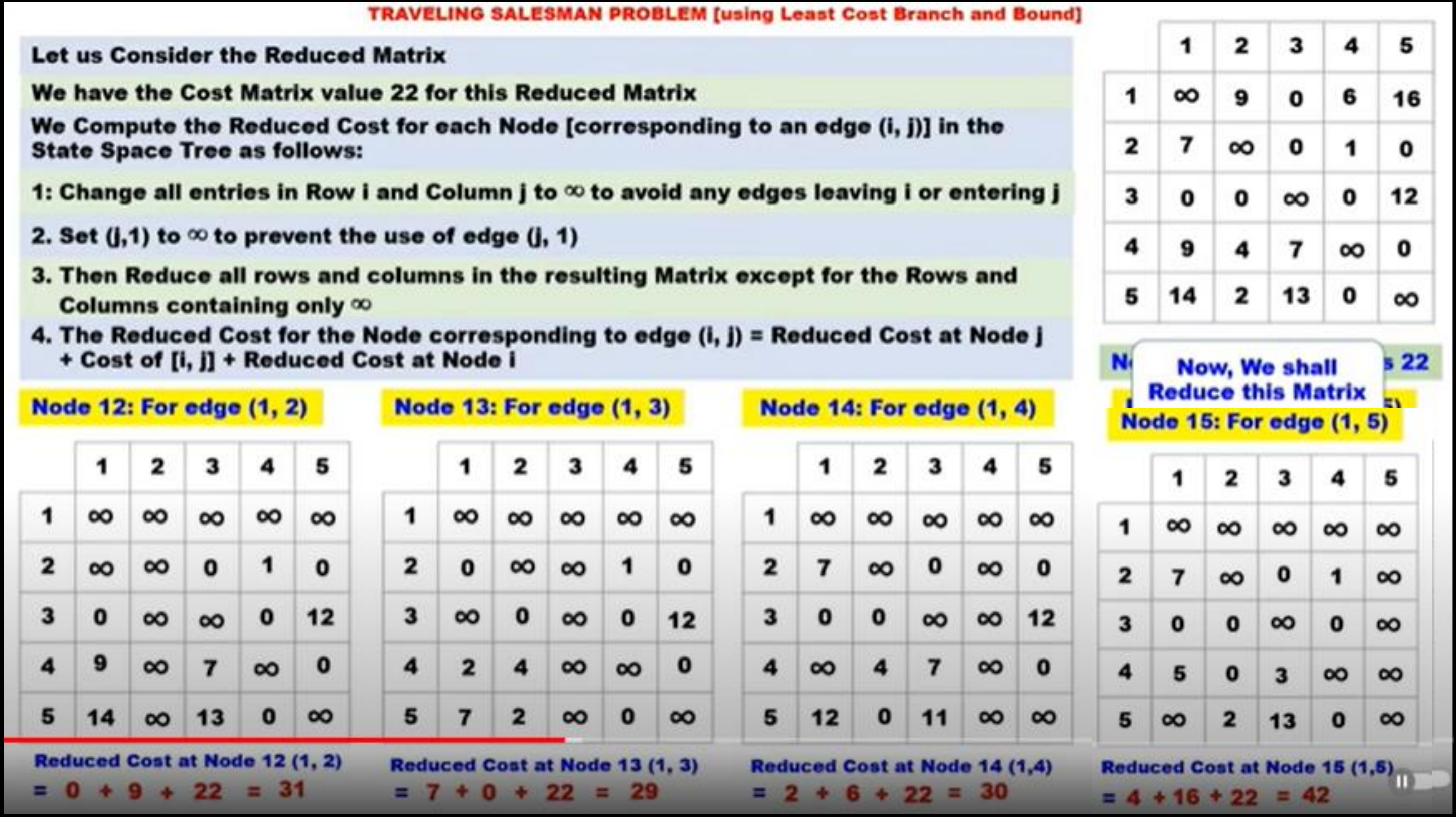
Node 12: For edge (1, 2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	0	1	0
3	0	∞	∞	0	12
4	9	∞	7	∞	0
5	14	∞	13	0	∞

Reduced Cost at Node 12 (1, 2)

= 0 + 9 + 22 = 31

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Let us Consider the Reduced Matrix

We have the Cost Matrix value 22 for this Reduced Matrix

We Compute the Reduced Cost for each Node [corresponding to an edge (i, j)] in the State Space Tree as follows:

1: Change all entries in Row i and Column j to ∞ to avoid any edges leaving i or entering j

2. Set (j,1) to ∞ to prevent the use of edge (j, 1)

3. Then Reduce all rows and columns in the resulting Matrix except for the Rows and Columns containing only ∞

4. The Reduced Cost for the Node corresponding to edge (i, j) = Reduced Cost at Node j + Cost of [i, j] + Reduced Cost at Node i

	1	2	3	4	5
1	∞	9	0	6	16
2	7	∞	0	1	0
3	0	0	∞	0	12
4	9	4	7	∞	0
5	14	2	13	0	∞

N s 22

Now, We shall Reduce this Matrix

Node 12: For edge (1, 2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	0	1	0
3	0	∞	∞	0	12
4	9	∞	7	∞	0
5	14	∞	13	0	∞

Reduced Cost at Node 12 (1, 2)
= 0 + 9 + 22 = 31

Node 13: For edge (1, 3)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	0
3	∞	0	∞	0	12
4	2	4	∞	∞	0
5	7	2	∞	0	∞

Reduced Cost at Node 13 (1, 3)
= 7 + 0 + 22 = 29

Node 14: For edge (1, 4)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	∞	0
3	0	0	∞	∞	12
4	∞	4	7	∞	0
5	12	0	11	∞	∞

Reduced Cost at Node 14 (1,4)
= 2 + 6 + 22 = 30

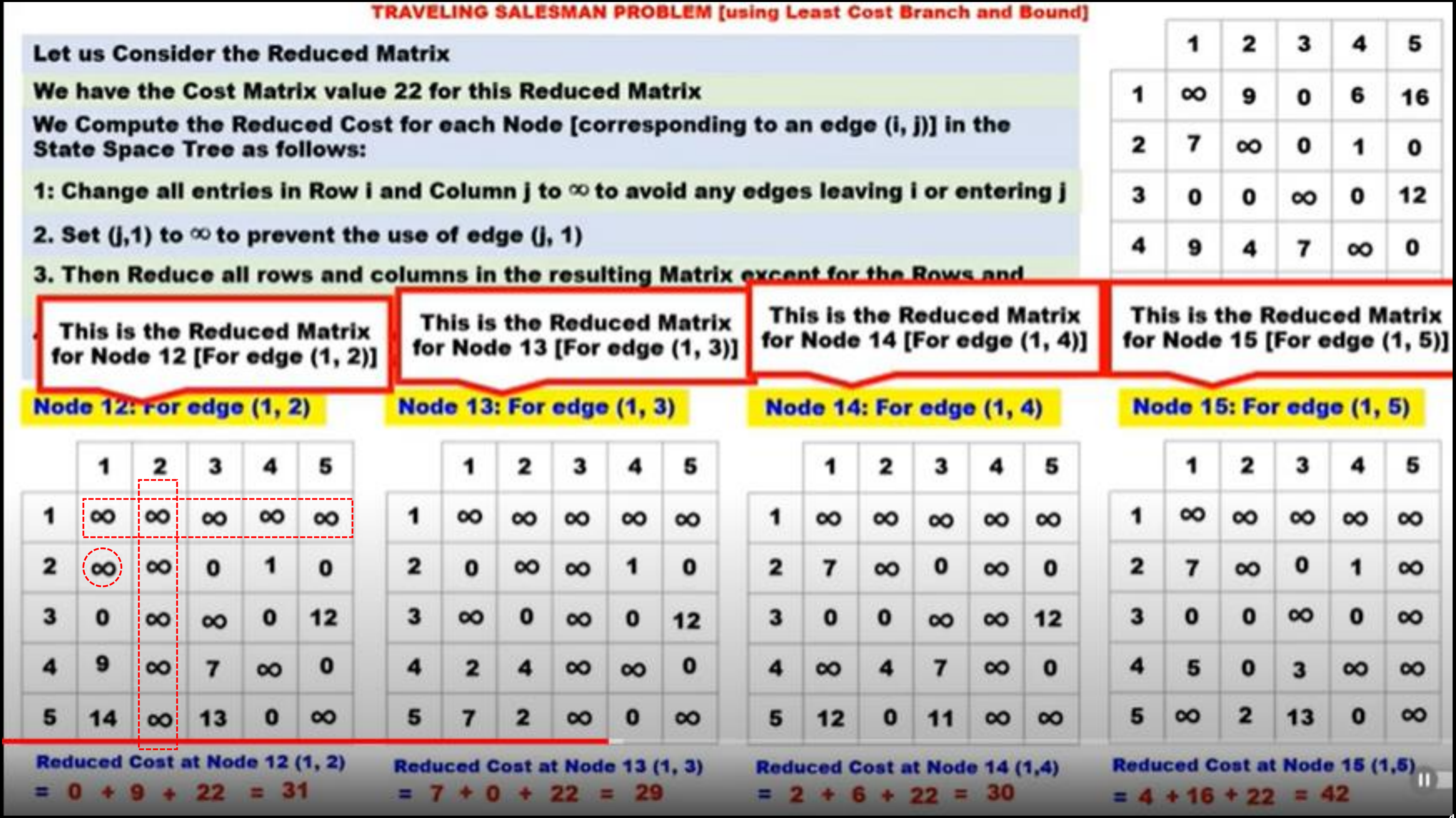
Node 15: For edge (1, 5)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	1	∞
3	0	0	∞	0	∞
4	5	0	3	∞	∞
5	∞	2	13	0	∞

Reduced Cost at Node 15 (1,5)
= 4 + 16 + 22 = 42

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Let us Consider the Reduced Matrix

We have the Cost Matrix value 22 for this Reduced Matrix

We Compute the Reduced Cost for each Node [corresponding to an edge (i, j)] in the State Space Tree as follows:

1: Change all entries in Row i and Column j to ∞ to avoid any edges leaving i or entering j

2. Set (j,1) to ∞ to prevent the use of edge (j, 1)

3. Then Reduce all rows and columns in the resulting Matrix except for the Rows and

	1	2	3	4	5
1	∞	9	0	6	16
2	7	∞	0	1	0
3	0	0	∞	0	12
4	9	4	7	∞	0

This is the Reduced Matrix for Node 12 [For edge (1, 2)]

This is the Reduced Matrix for Node 13 [For edge (1, 3)]

This is the Reduced Matrix for Node 14 [For edge (1, 4)]

This is the Reduced Matrix for Node 15 [For edge (1, 5)]

Node 12: For edge (1, 2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	0	1	0
3	0	∞	∞	0	12
4	9	∞	7	∞	0
5	14	∞	13	0	∞

Reduced Cost at Node 12 (1, 2)
= 0 + 9 + 22 = 31

Node 13: For edge (1, 3)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	0
3	∞	0	∞	0	12
4	2	4	∞	∞	0
5	7	2	∞	0	∞

Reduced Cost at Node 13 (1, 3)
= 7 + 0 + 22 = 29

Node 14: For edge (1, 4)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	∞	0
3	0	0	∞	∞	12
4	∞	4	7	∞	0
5	12	0	11	∞	∞

Reduced Cost at Node 14 (1,4)
= 2 + 6 + 22 = 30

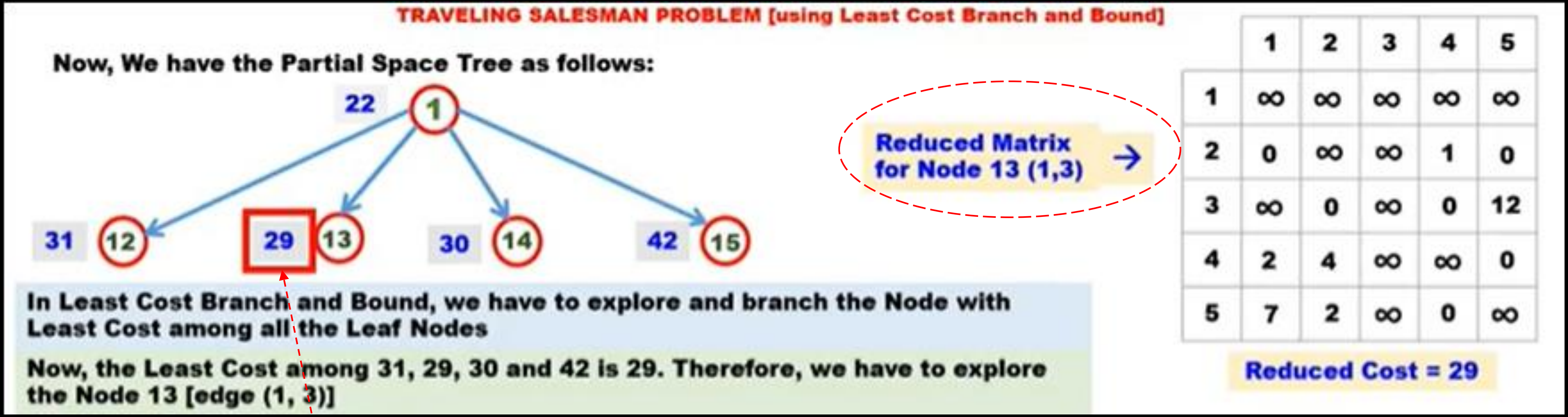
Node 15: For edge (1, 5)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	1	∞
3	0	0	∞	0	∞
4	5	0	3	∞	∞
5	∞	2	13	0	∞

Reduced Cost at Node 15 (1,5)
= 4 + 16 + 22 = 42

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Now, We have the Partial Space Tree as follows:

Reduced Matrix for Node 13 (1,3) →

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	0
3	∞	0	∞	0	12
4	2	4	∞	∞	0
5	7	2	∞	0	∞

Reduced Cost = 29

In Least Cost Branch and Bound, we have to explore and branch the Node with Least Cost among all the Leaf Nodes

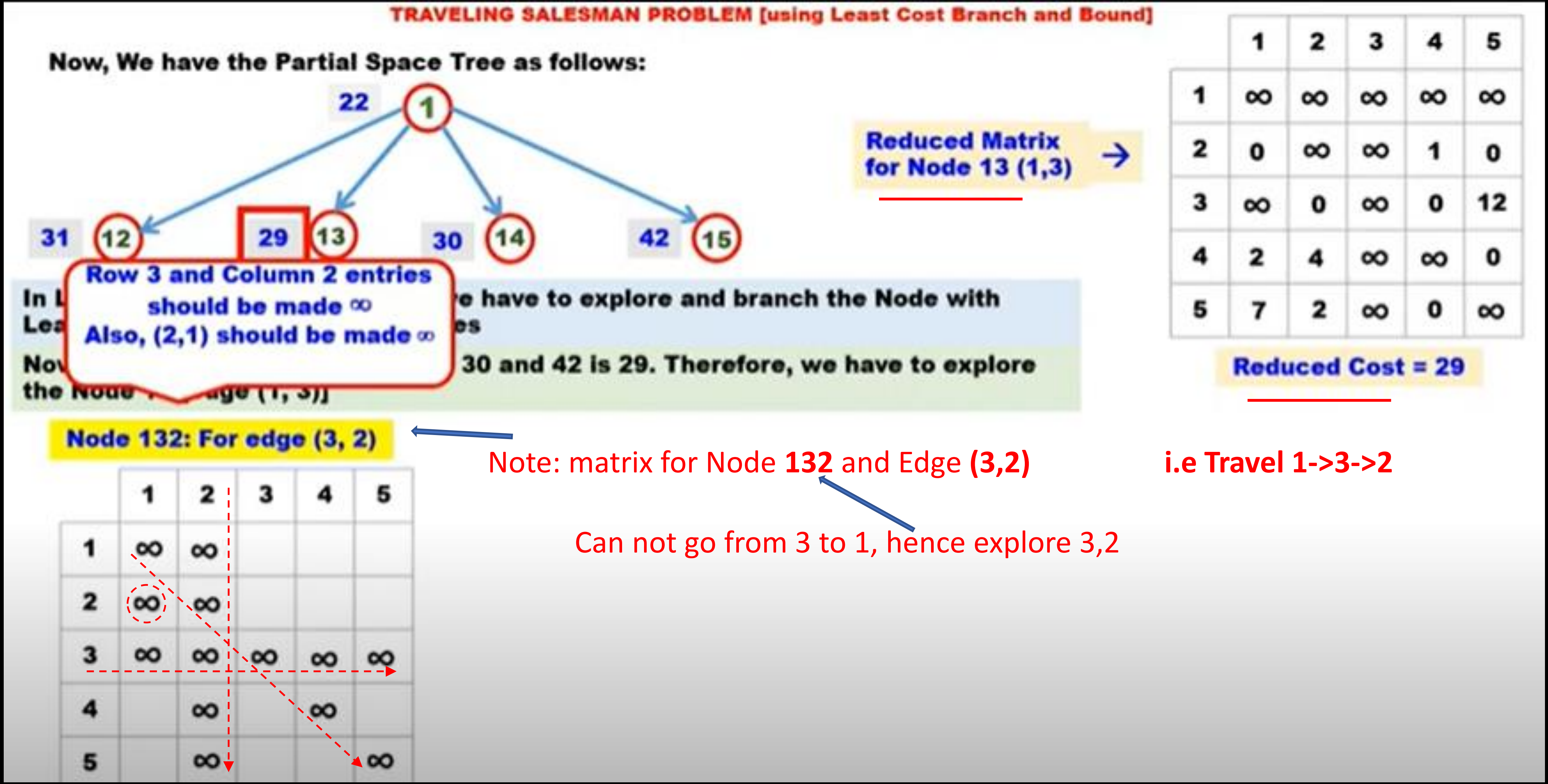
Now, the Least Cost among 31, 29, 30 and 42 is 29. Therefore, we have to explore the Node 13 [edge (1, 3)]

Note the node naming convention. 12: 1->2; 13: 1->3 etc

Least Cost node 13 having a cost of 29

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Now, We have the Partial Space Tree as follows:

Row 3 and Column 2 entries should be made ∞
Also, (2,1) should be made ∞

In L... e have to explore and branch the Node with
Lea... es

No... 30 and 42 is 29. Therefore, we have to explore
the Node ... ge (1, 3)]

Reduced Matrix for Node 13 (1,3) →

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	0
3	∞	0	∞	0	12
4	2	4	∞	∞	0
5	7	2	∞	0	∞

Reduced Cost = 29

Node 132: For edge (3, 2)

	1	2	3	4	5
1	∞	∞			
2	∞	∞			
3	∞	∞	∞	∞	∞
4		∞		∞	
5		∞			∞

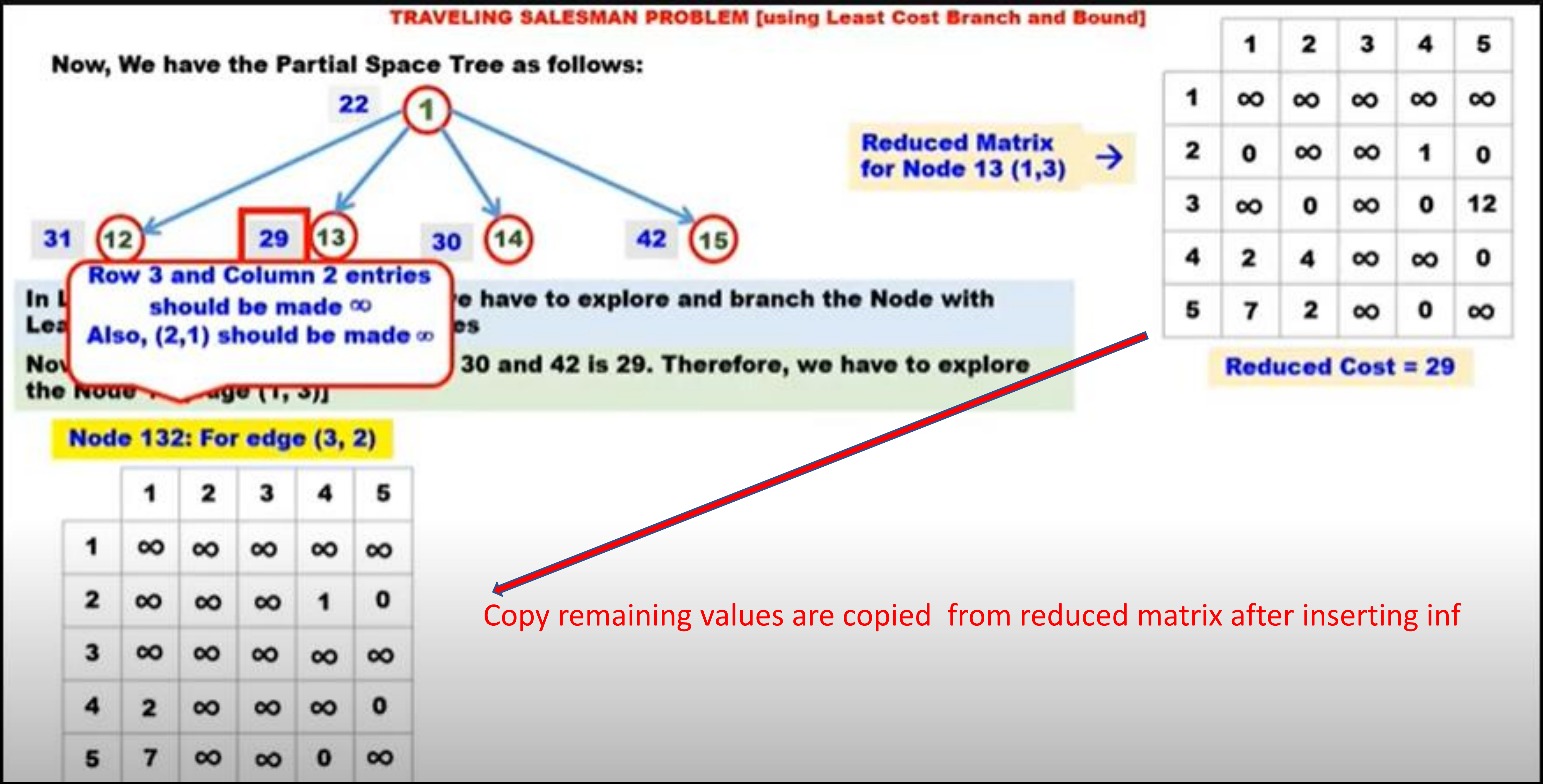
Note: matrix for Node **132** and Edge **(3,2)**

i.e Travel 1->3->2

Can not go from 3 to 1, hence explore 3,2

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Now, We have the Partial Space Tree as follows:

Row 3 and Column 2 entries should be made ∞
Also, (2,1) should be made ∞

In L… e have to explore and branch the Node with Lea… es

Nov… 30 and 42 is 29. Therefore, we have to explore the Node …age (1, 3)]

Reduced Matrix for Node 13 (1,3) →

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	0
3	∞	0	∞	0	12
4	2	4	∞	∞	0
5	7	2	∞	0	∞

Reduced Cost = 29

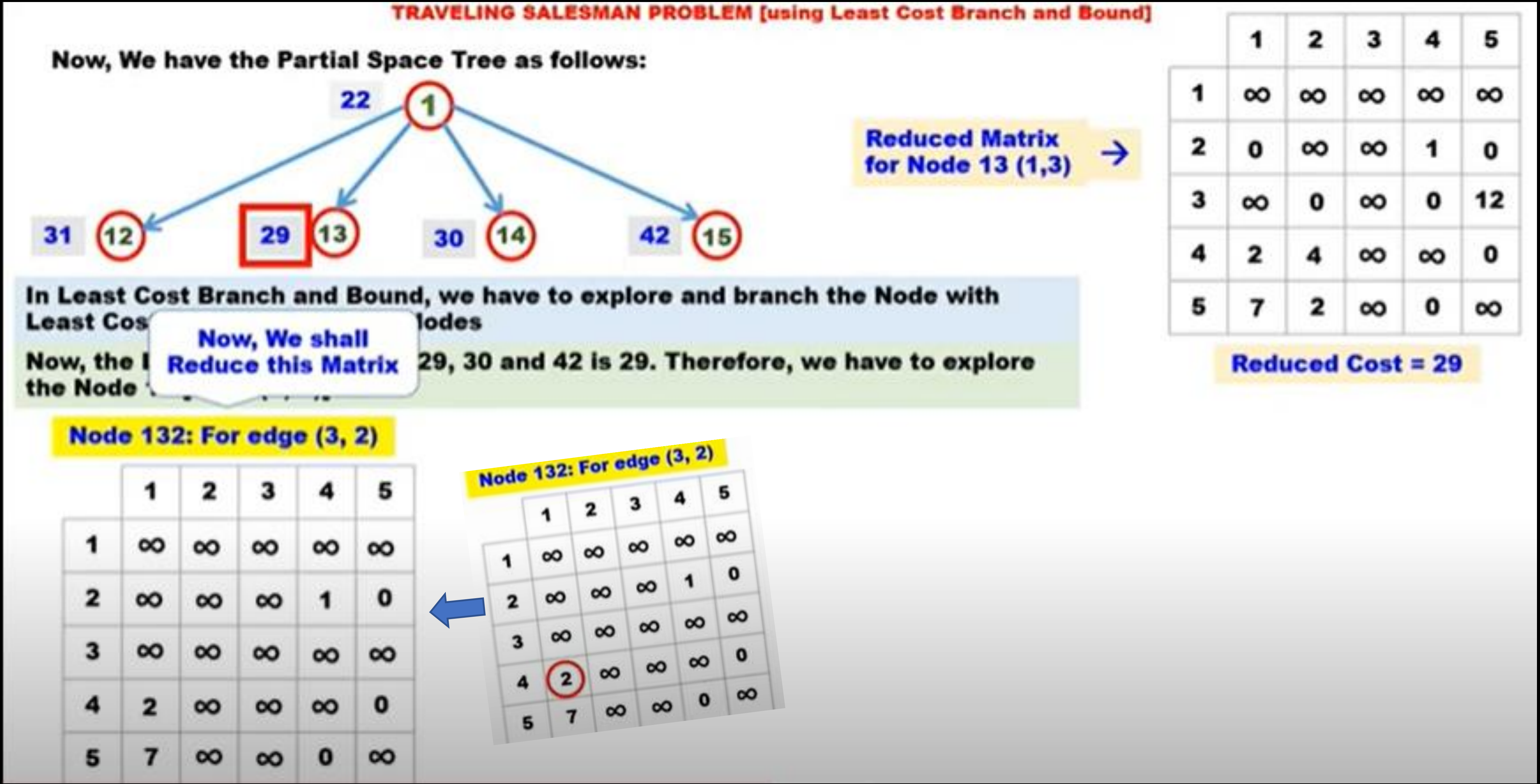
Node 132: For edge (3, 2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	1	0
3	∞	∞	∞	∞	∞
4	2	∞	∞	∞	0
5	7	∞	∞	0	∞

Copy remaining values are copied from reduced matrix after inserting inf

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Now, We have the Partial Space Tree as follows:

Reduced Matrix for Node 13 (1,3) →

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	0
3	∞	0	∞	0	12
4	2	4	∞	∞	0
5	7	2	∞	0	∞

Reduced Cost = 29

In Least Cost Branch and Bound, we have to explore and branch the Node with Least Cos lodes

Now, We shall Reduce this Matrix

Now, the l 29, 30 and 42 is 29. Therefore, we have to explore the Node

Node 132: For edge (3, 2)

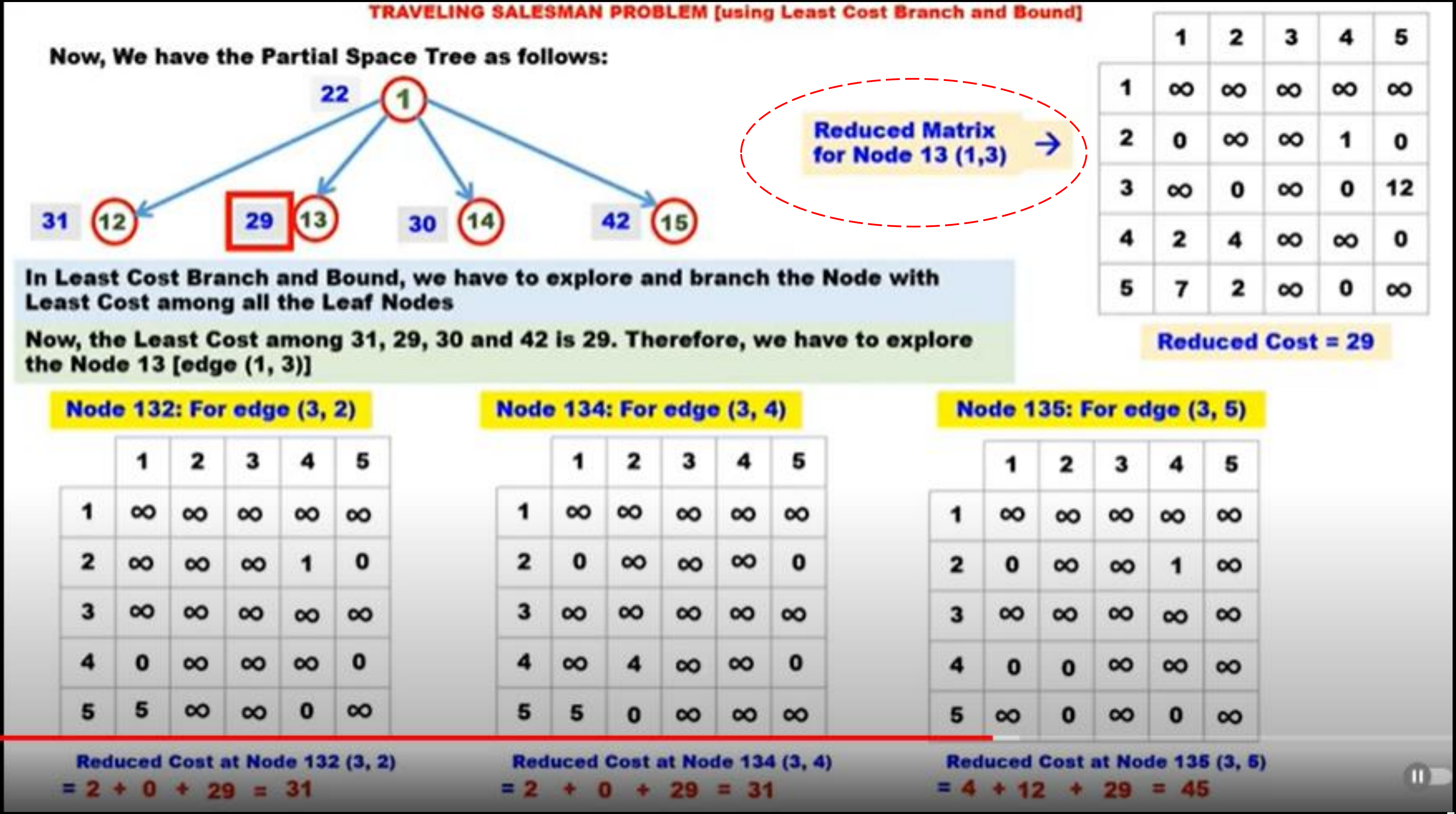
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	1	0
3	∞	∞	∞	∞	∞
4	2	∞	∞	∞	0
5	7	∞	∞	0	∞

Node 132: For edge (3, 2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	1	0
3	∞	∞	∞	∞	∞
4	2	∞	∞	∞	0
5	7	∞	∞	0	∞

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Now, We have the Partial Space Tree as follows:

Reduced Matrix for Node 13 (1,3) →

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	0
3	∞	0	∞	0	12
4	2	4	∞	∞	0
5	7	2	∞	0	∞

Reduced Cost = 29

In Least Cost Branch and Bound, we have to explore and branch the Node with Least Cost among all the Leaf Nodes

Now, the Least Cost among 31, 29, 30 and 42 is 29. Therefore, we have to explore the Node 13 [edge (1, 3)]

Node 132: For edge (3, 2)

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

Reduced Cost at Node 132 (3, 2)
= 2 + 0 + 29 = 31

Node 134: For edge (3, 4)

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

Reduced Cost at Node 134 (3, 4)
= 2 + 0 + 29 = 31

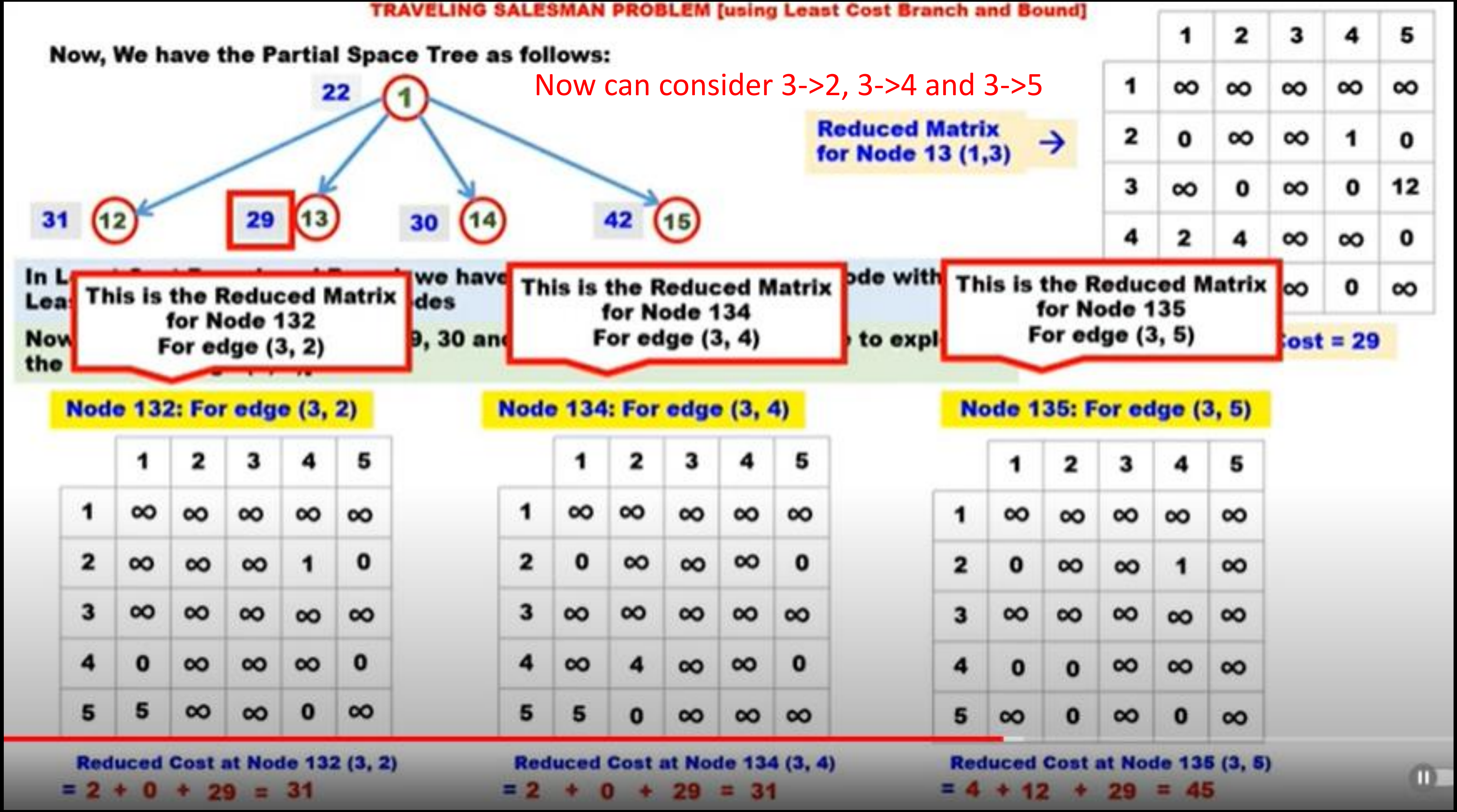
Node 135: For edge (3, 5)

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

Reduced Cost at Node 135 (3, 5)
= 4 + 12 + 29 = 45

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Now, We have the Partial Space Tree as follows:

Now can consider 3->2, 3->4 and 3->5

Reduced Matrix for Node 13 (1,3) →

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	0
3	∞	0	∞	0	12
4	2	4	∞	∞	0
			∞	0	∞

In L … we have … ode with

Lea … des

Now … 9, 30 an … to expl … ost = 29

the

This is the Reduced Matrix for Node 132 For edge (3, 2)

This is the Reduced Matrix for Node 134 For edge (3, 4)

This is the Reduced Matrix for Node 135 For edge (3, 5)

Node 132: For edge (3, 2)

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

Reduced Cost at Node 132 (3, 2)
= 2 + 0 + 29 = 31

Node 134: For edge (3, 4)

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

Reduced Cost at Node 134 (3, 4)
= 2 + 0 + 29 = 31

Node 135: For edge (3, 5)

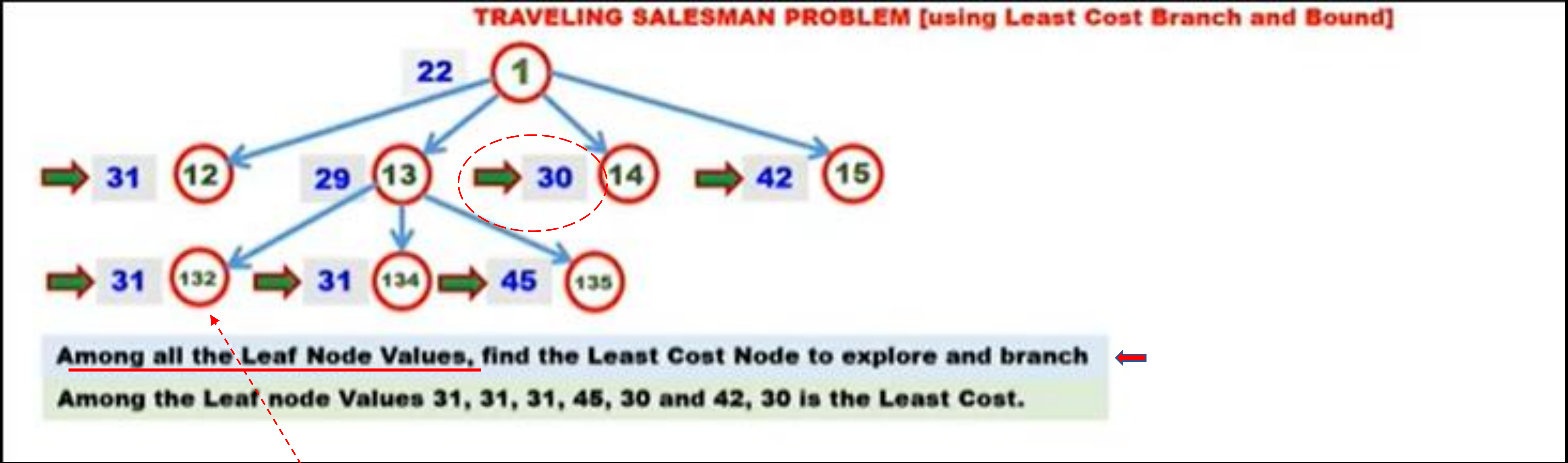
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	∞
3	∞	∞	∞	∞	∞
4	0	0	∞	∞	∞
5	∞	0	∞	0	∞

Reduced Cost at Node 135 (3, 5)
= 4 + 12 + 29 = 45

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound

- State space tree for LC BnB



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

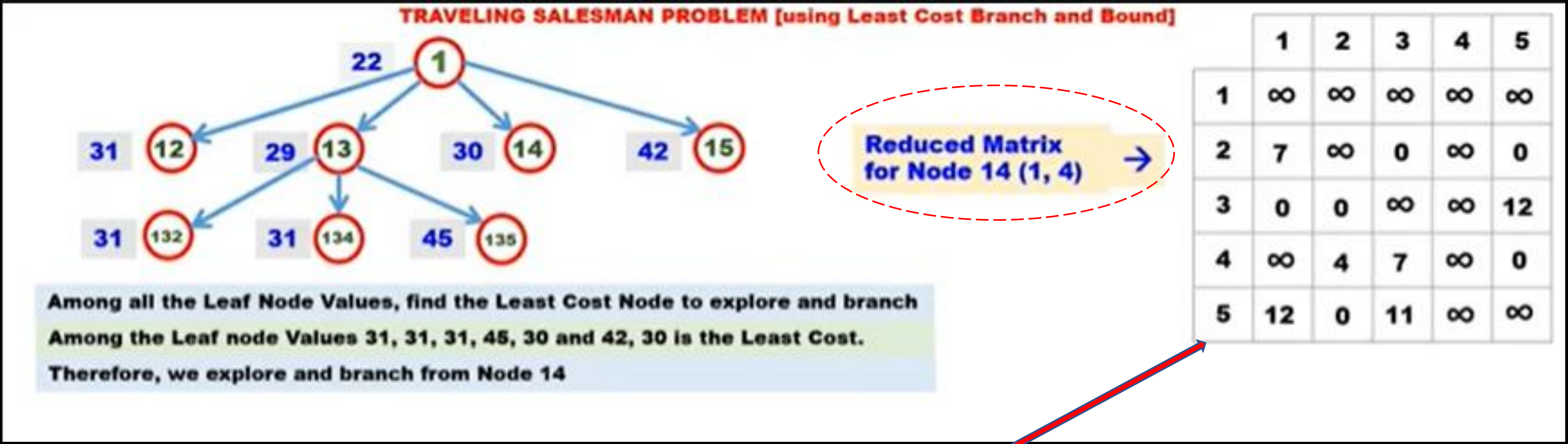
Among all the Leaf Node Values, find the Least Cost Node to explore and branch

Among the Leaf node Values 31, 31, 31, 45, 30 and 42, 30 is the Least Cost.

Note the naming convention 132 : 1->3->2

https://www.youtube.com/watch?v=HjSbaKF8Gi0

Branch Bound: Least Cost-Branch and Bound



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

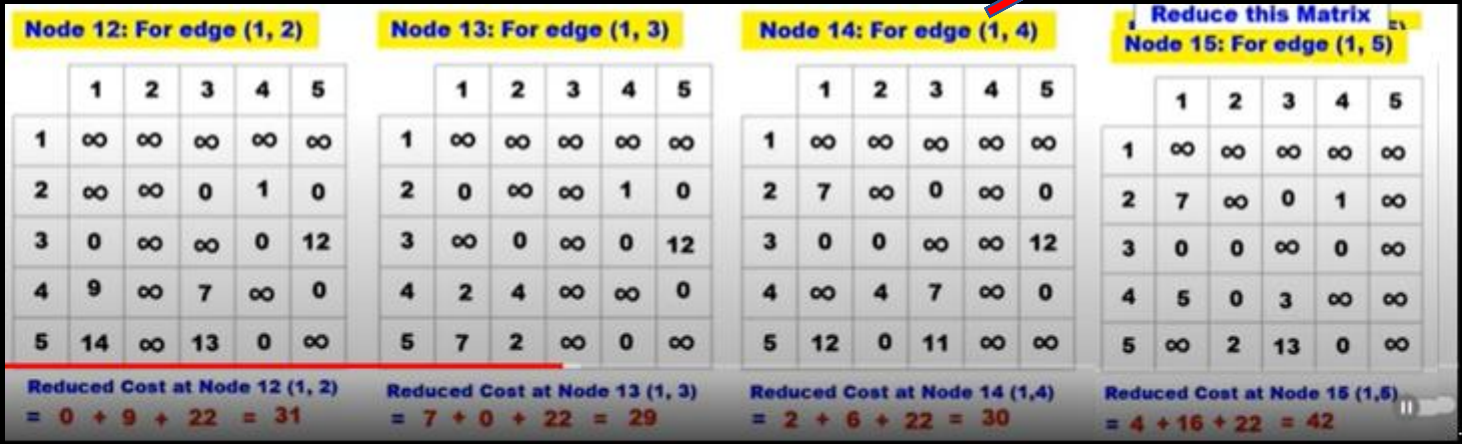
Among all the Leaf Node Values, find the Least Cost Node to explore and branch

Among the Leaf node Values 31, 31, 31, 45, 30 and 42, 30 is the Least Cost.

Therefore, we explore and branch from Node 14

Reduced Matrix for Node 14 (1, 4) →

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	∞	0
3	0	0	∞	∞	12
4	∞	4	7	∞	0
5	12	0	11	∞	∞



Node 12: For edge (1, 2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	0	1	0
3	0	∞	∞	0	12
4	9	∞	7	∞	0
5	14	∞	13	0	∞

Reduced Cost at Node 12 (1, 2)
= 0 + 9 + 22 = 31

Node 13: For edge (1, 3)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	1	0
3	∞	0	∞	0	12
4	2	4	∞	∞	0
5	7	2	∞	0	∞

Reduced Cost at Node 13 (1, 3)
= 7 + 0 + 22 = 29

Node 14: For edge (1, 4)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	∞	0
3	0	0	∞	∞	12
4	∞	4	7	∞	0
5	12	0	11	∞	∞

Reduced Cost at Node 14 (1,4)
= 2 + 6 + 22 = 30

Reduce this Matrix

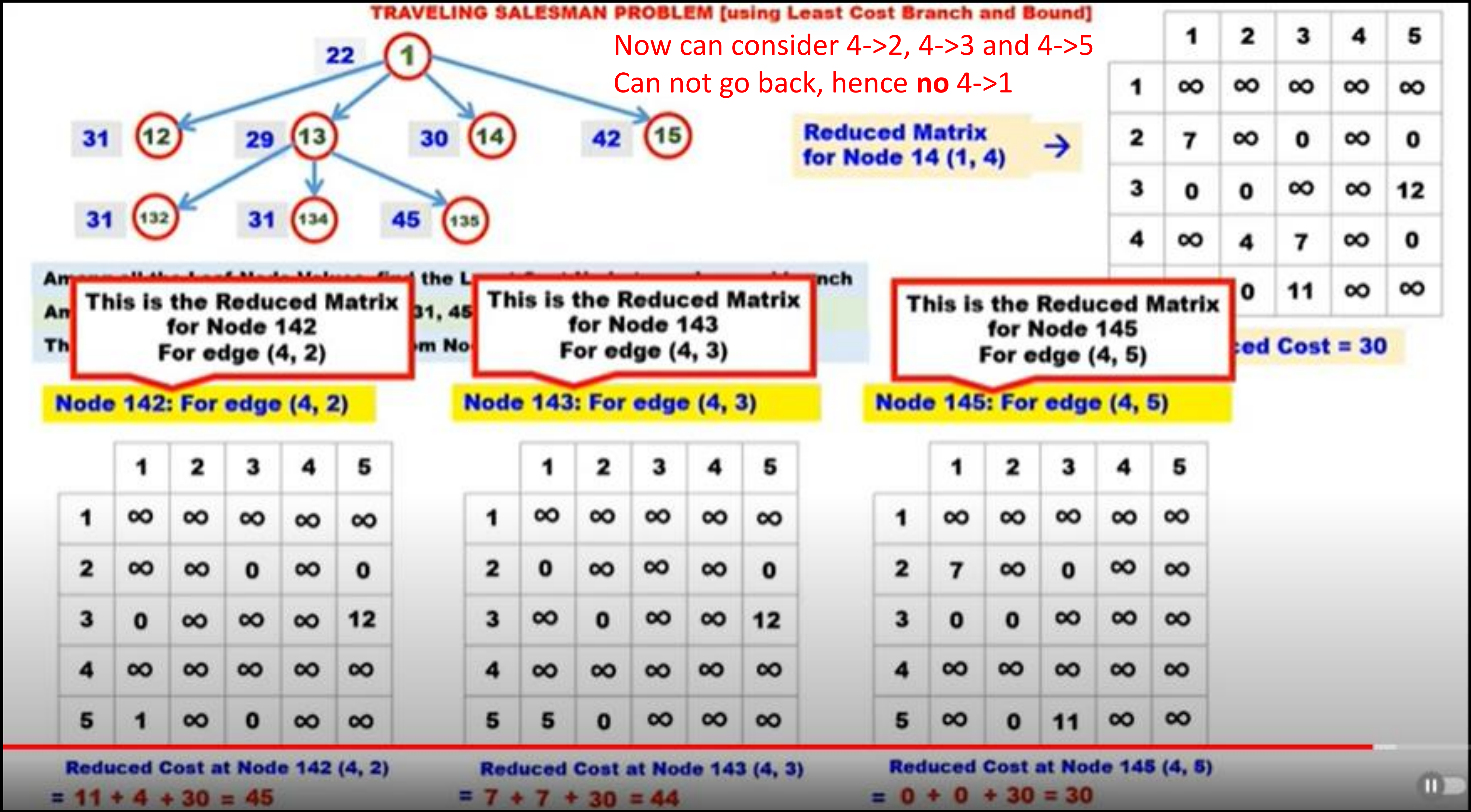
Node 15: For edge (1, 5)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	1	∞
3	0	0	∞	0	∞
4	5	0	3	∞	∞
5	∞	2	13	0	∞

Reduced Cost at Node 15 (1,5)
= 4 + 16 + 22 = 42

Take Reduced matrix for Node 14 from previous calculations

https://www.youtube.com/watch?v=HjSbaKF8Gi0



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Now can consider 4->2, 4->3 and 4->5
Can not go back, hence **no** 4->1

Reduced Matrix for Node 14 (1, 4) →

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	∞	0
3	0	0	∞	∞	12
4	∞	4	7	∞	0
5	[illegible]	0	11	∞	∞

…ed Cost = 30

Am… the L… nch

An… 31, 45

Th… m No

This is the Reduced Matrix
for Node 142
For edge (4, 2)

This is the Reduced Matrix
for Node 143
For edge (4, 3)

This is the Reduced Matrix
for Node 145
For edge (4, 5)

Node 142: For edge (4, 2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	0	∞	0
3	0	∞	∞	∞	12
4	∞	∞	∞	∞	∞
5	1	∞	0	∞	∞

Reduced Cost at Node 142 (4, 2)
= 11 + 4 + 30 = 45

Node 143: For edge (4, 3)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	0	∞	∞	∞	0
3	∞	0	∞	∞	12
4	∞	∞	∞	∞	∞
5	5	0	∞	∞	∞

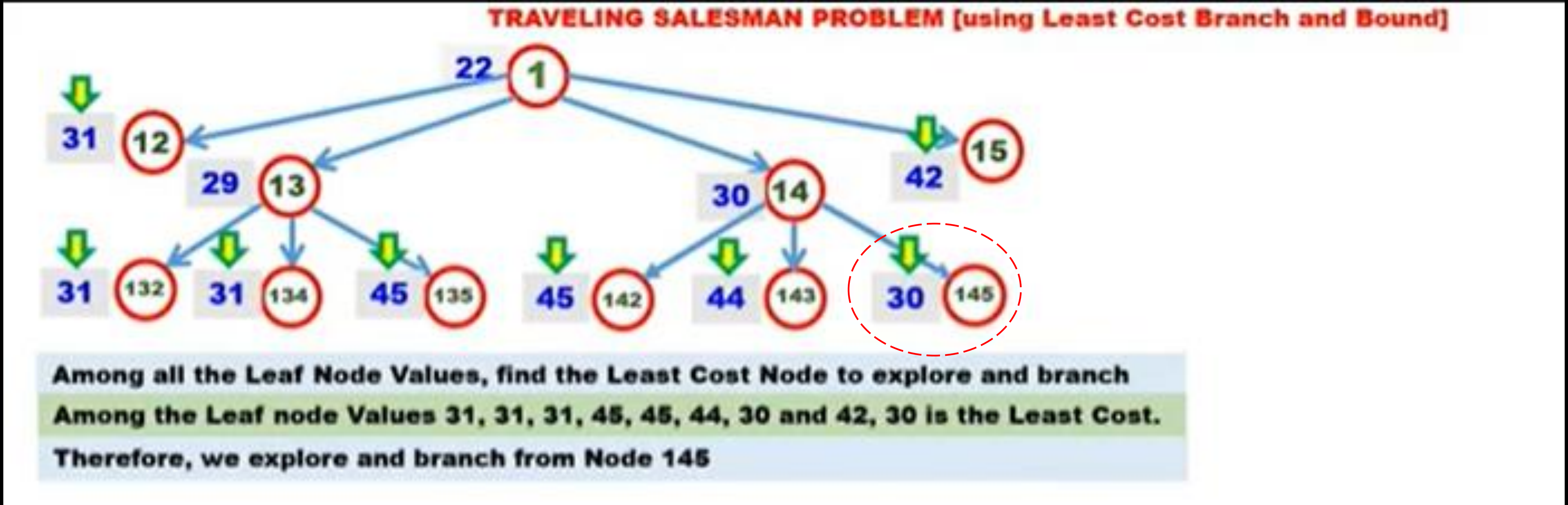
Reduced Cost at Node 143 (4, 3)
= 7 + 7 + 30 = 44

Node 145: For edge (4, 5)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	∞	∞
3	0	0	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	0	11	∞	∞

Reduced Cost at Node 145 (4, 5)
= 0 + 0 + 30 = 30

https://www.youtube.com/watch?v=HjSbaKF8Gi0



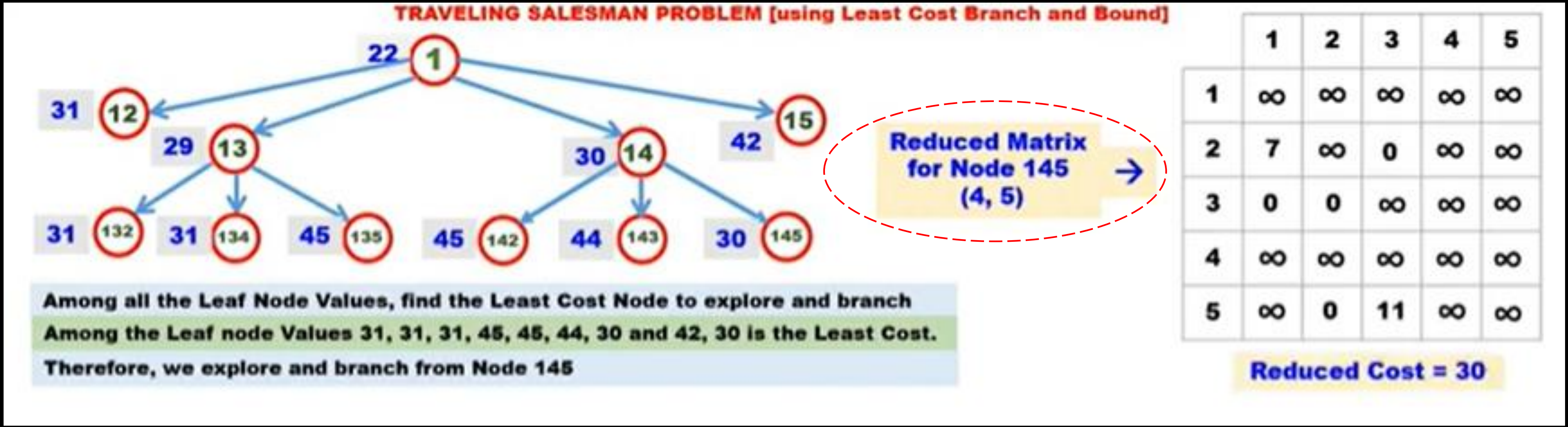
TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Among all the Leaf Node Values, find the Least Cost Node to explore and branch

Among the Leaf node Values 31, 31, 31, 45, 45, 44, 30 and 42, 30 is the Least Cost.

Therefore, we explore and branch from Node 145

https://www.youtube.com/watch?v=HjSbaKF8Gi0



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Reduced Matrix for Node 145 (4, 5) →

Among all the Leaf Node Values, find the Least Cost Node to explore and branch

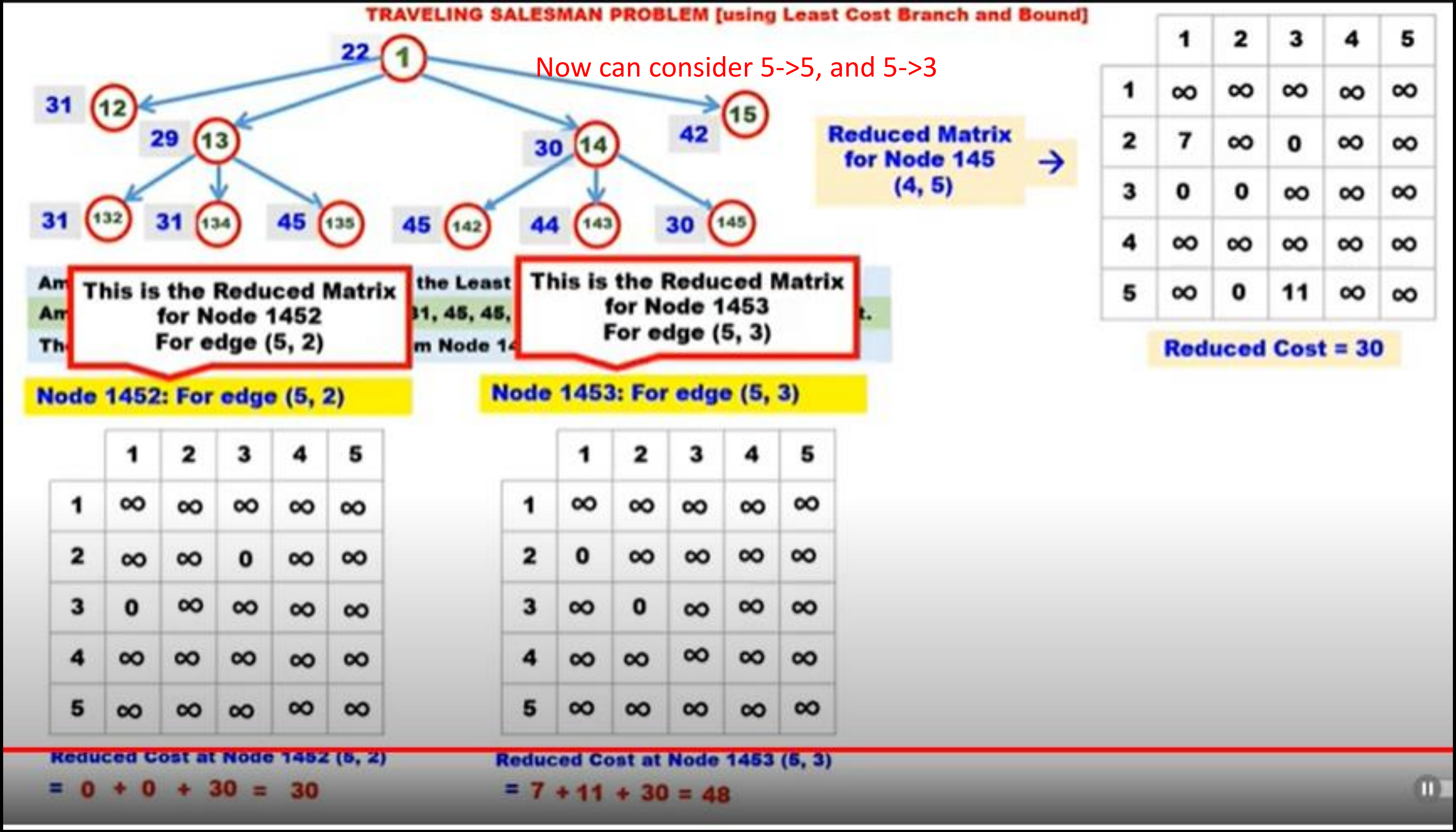
Among the Leaf node Values 31, 31, 31, 45, 45, 44, 30 and 42, 30 is the Least Cost.

Therefore, we explore and branch from Node 145

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	∞	∞
3	0	0	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	0	11	∞	∞

Reduced Cost = 30

https://www.youtube.com/watch?v=HjSbaKF8Gi0



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Now can consider 5->5, and 5->3

Reduced Matrix for Node 145 (4, 5) →

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	7	∞	0	∞	∞
3	0	0	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	0	11	∞	∞

Reduced Cost = 30

Am ... the Least

Am ... 31, 45, 45, ... t.

Th ... m Node 14

This is the Reduced Matrix for Node 1452 For edge (5, 2)

This is the Reduced Matrix for Node 1453 For edge (5, 3)

Node 1452: For edge (5, 2)

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

Reduced Cost at Node 1452 (5, 2)

= 0 + 0 + 30 = 30

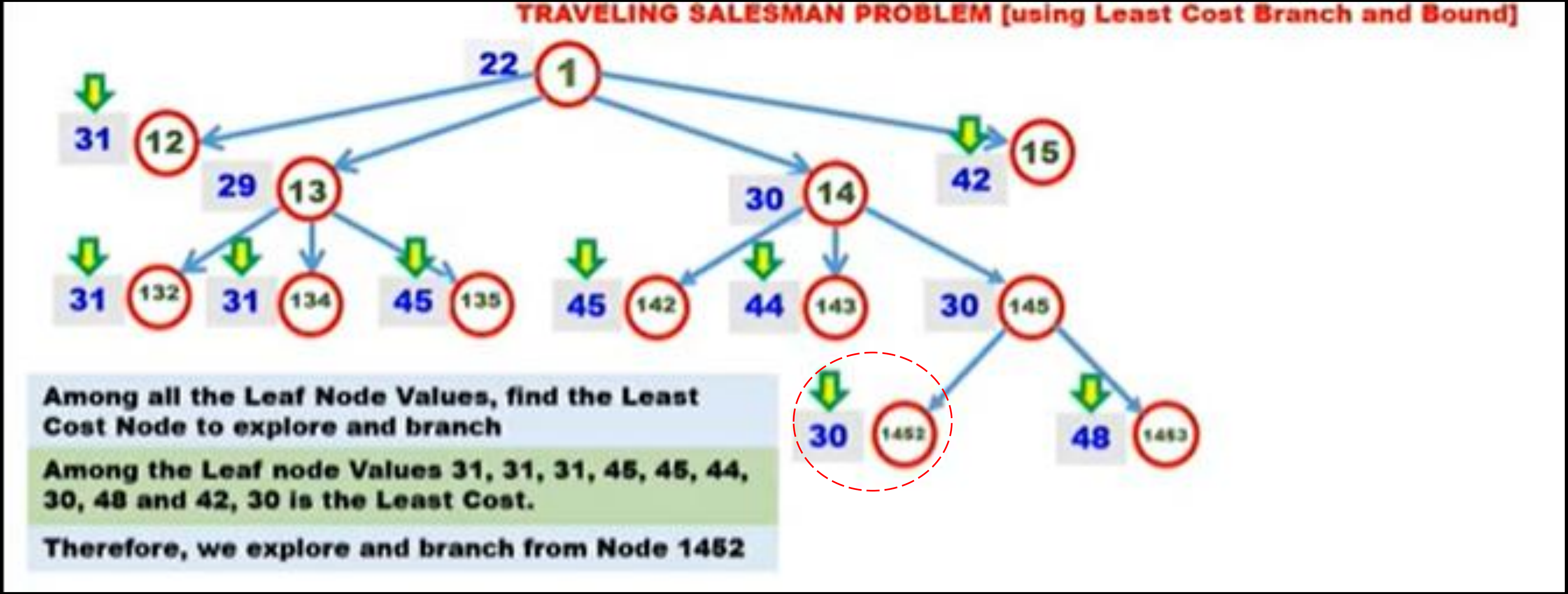
Node 1453: For edge (5, 3)

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

Reduced Cost at Node 1453 (5, 3)

= 7 + 11 + 30 = 48

https://www.youtube.com/watch?v=HjSbaKF8Gi0



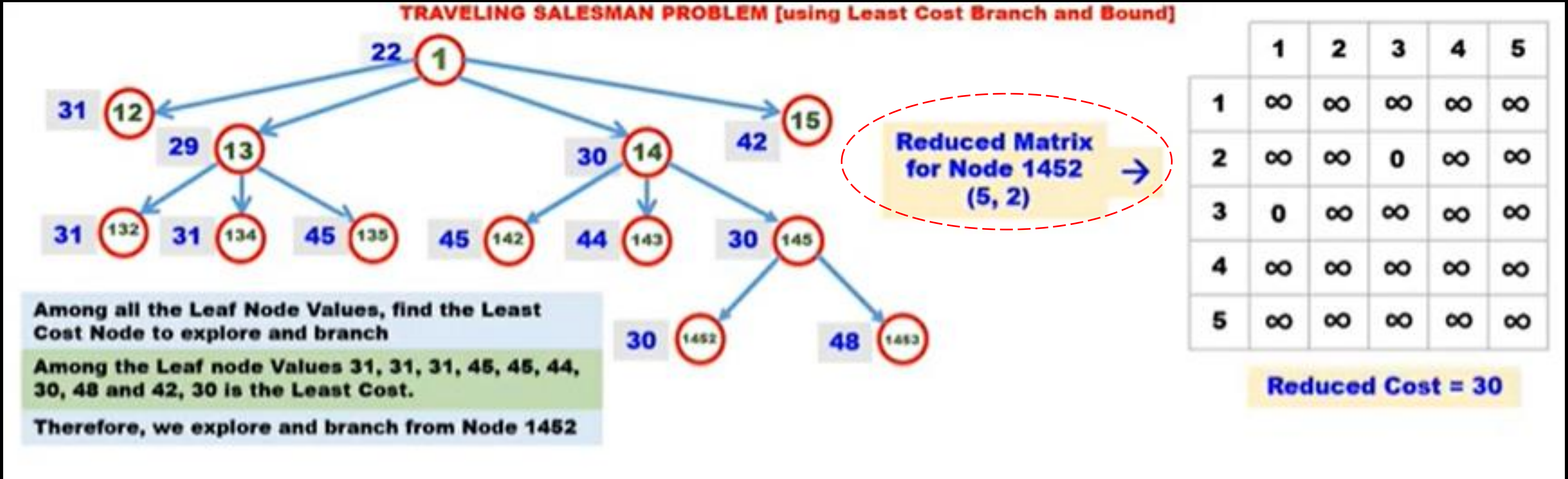
TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Among all the Leaf Node Values, find the Least Cost Node to explore and branch

Among the Leaf node Values 31, 31, 31, 45, 45, 44, 30, 48 and 42, 30 is the Least Cost.

Therefore, we explore and branch from Node 1452

https://www.youtube.com/watch?v=HjSbaKF8Gi0



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Among all the Leaf Node Values, find the Least Cost Node to explore and branch

Among the Leaf node Values 31, 31, 31, 45, 45, 44, 30, 48 and 42, 30 is the Least Cost.

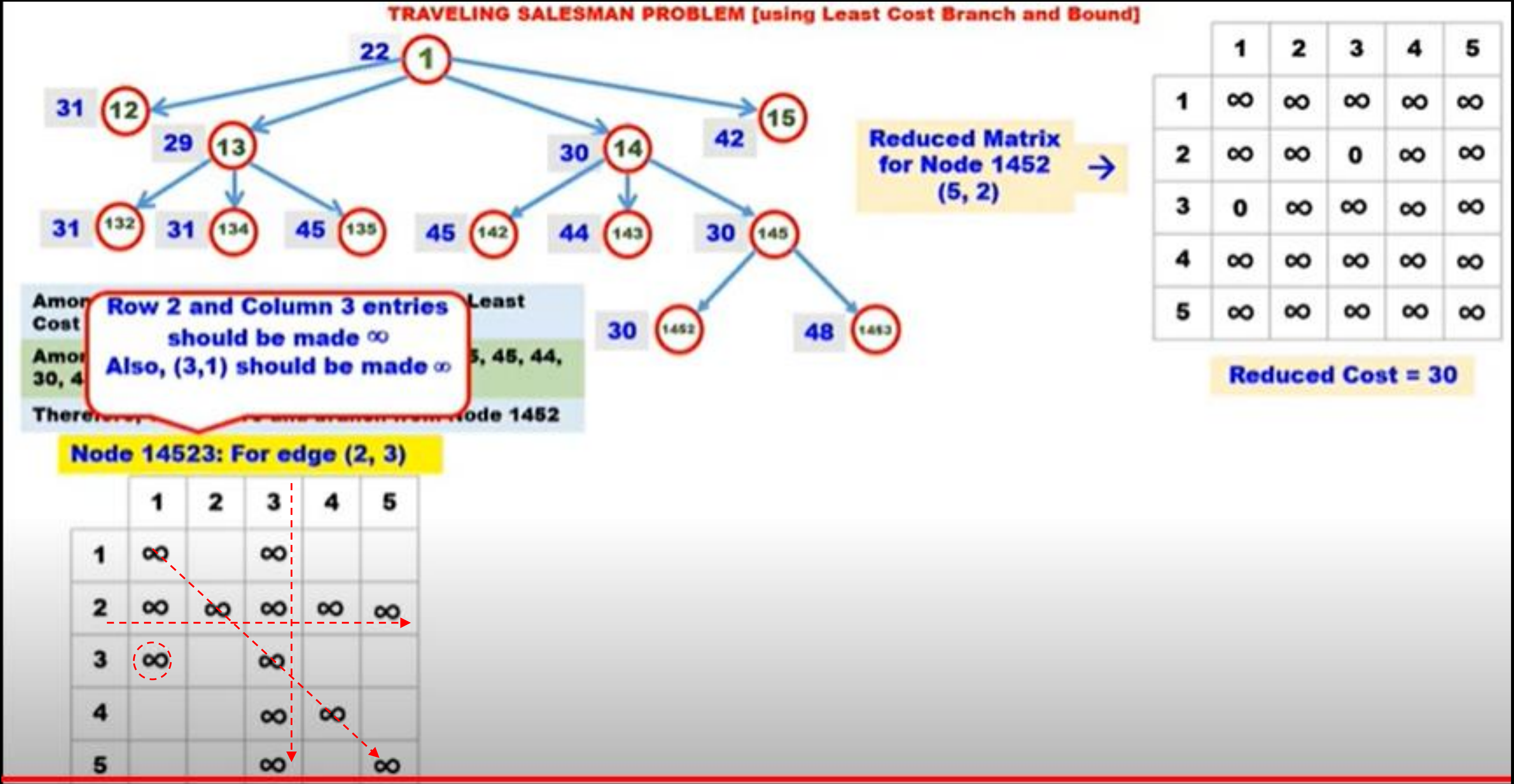
Therefore, we explore and branch from Node 1452

Reduced Matrix for Node 1452 (5, 2) →

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

Reduced Cost = 30

https://www.youtube.com/watch?v=HjSbaKF8Gi0



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Reduced Matrix for Node 1452 (5, 2) →

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

Reduced Cost = 30

Amon… Least Cost

Amon… 30, 4… 5, 45, 44,

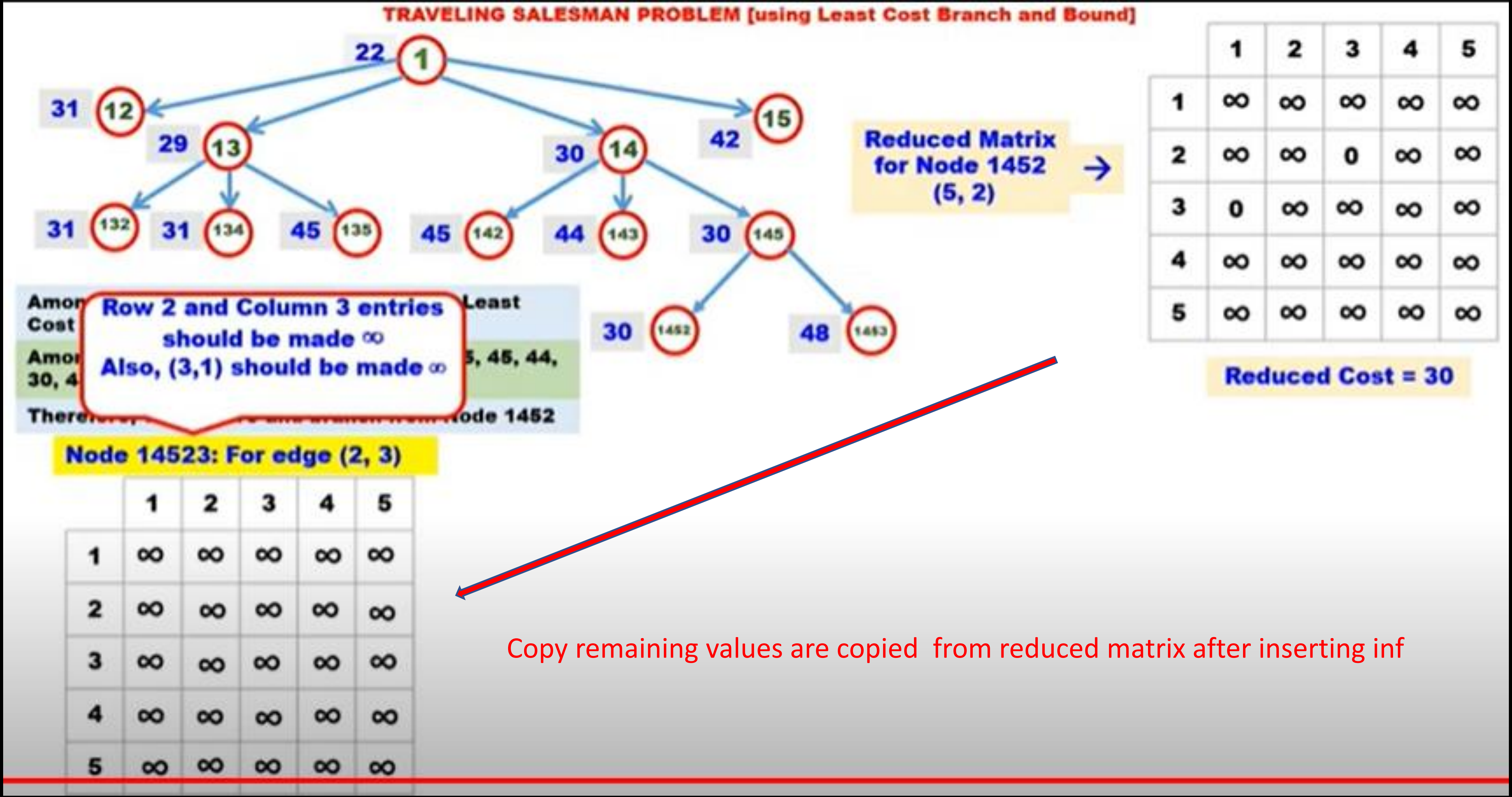
There… …ode 1452

Row 2 and Column 3 entries should be made ∞
Also, (3,1) should be made ∞

Node 14523: For edge (2, 3)

	1	2	3	4	5
1	∞		∞		
2	∞	∞	∞	∞	∞
3	∞		∞		
4			∞	∞	
5			∞		∞

https://www.youtube.com/watch?v=HjSbaKF8Gi0



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Amon… Least Cost

Amon… 5, 45, 44, 30, 4…

There… …ode 1452

Row 2 and Column 3 entries should be made ∞
Also, (3,1) should be made ∞

Reduced Matrix for Node 1452 (5, 2) →

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

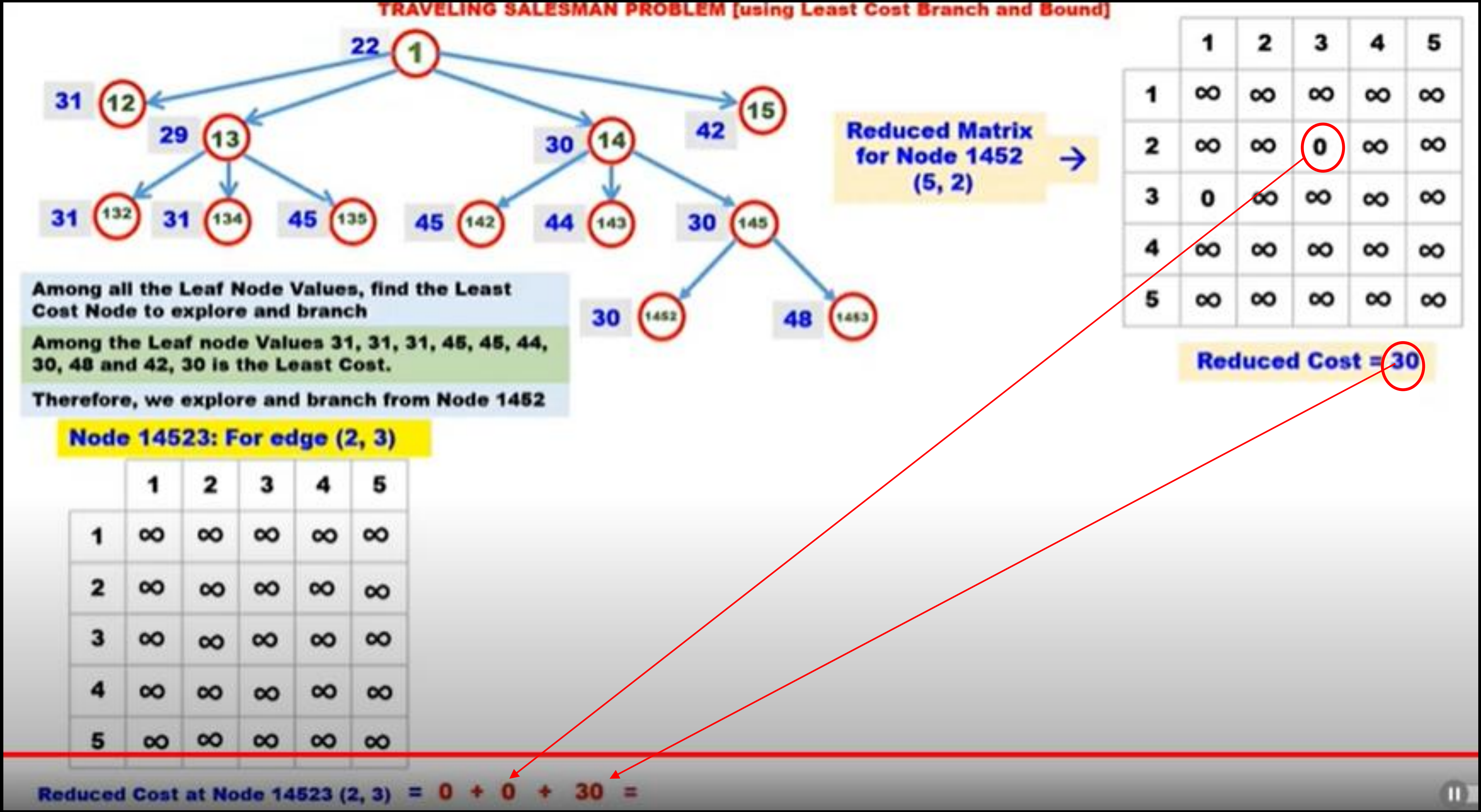
Reduced Cost = 30

Node 14523: For edge (2, 3)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞

Copy remaining values are copied from reduced matrix after inserting inf

https://www.youtube.com/watch?v=HjSbaKF8Gi0



TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

Reduced Matrix for Node 1452 (5, 2) →

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

Reduced Cost = 30

Among all the Leaf Node Values, find the Least Cost Node to explore and branch

Among the Leaf node Values 31, 31, 31, 45, 45, 44, 30, 48 and 42, 30 is the Least Cost.

Therefore, we explore and branch from Node 1452

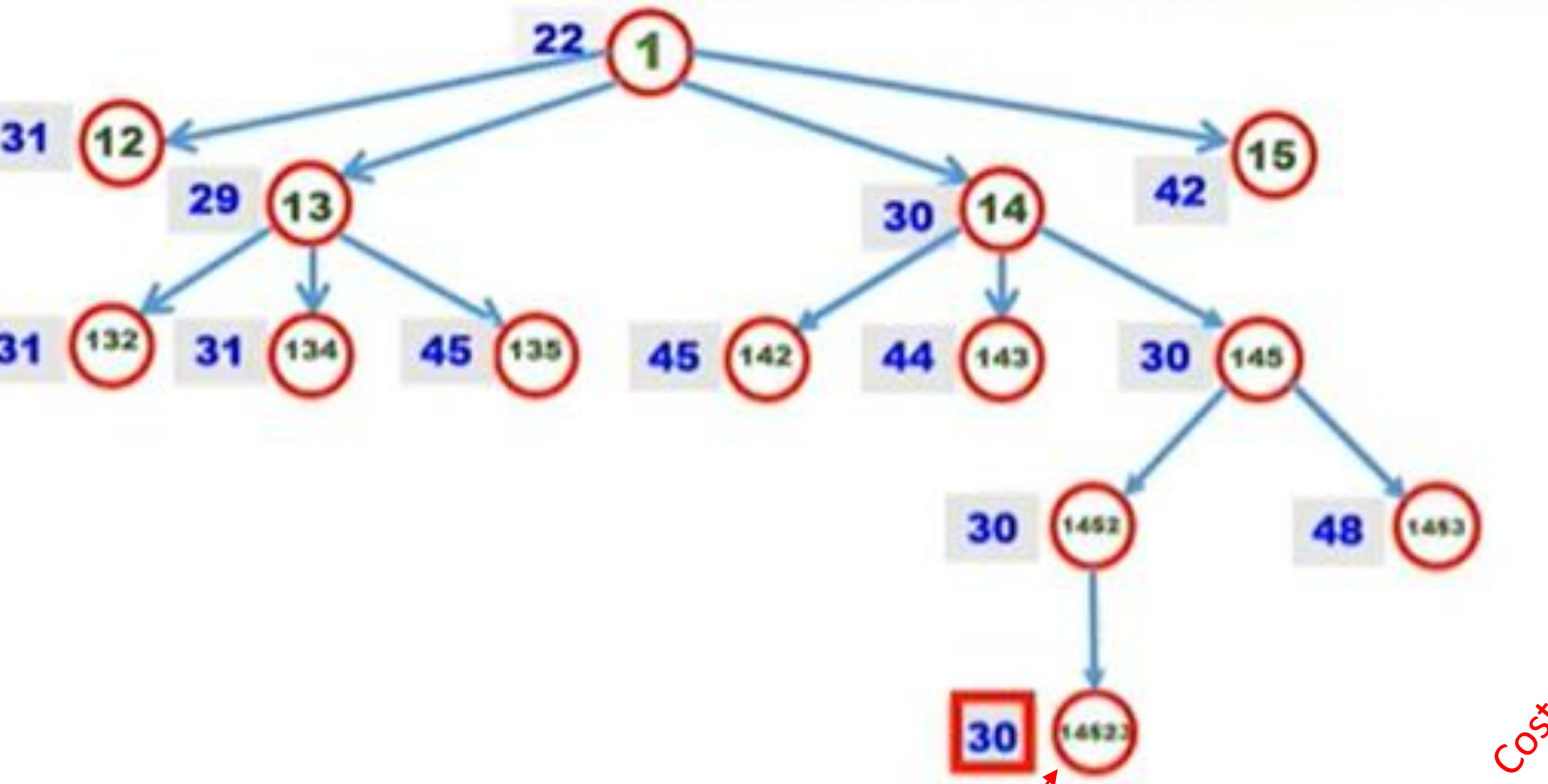
Node 14523: For edge (2, 3)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞

Reduced Cost at Node 14523 (2, 3) = 0 + 0 + 30 =

https://www.youtube.com/watch?v=HjSbaKF8Gi0

TRAVELING SALESMAN PROBLEM [using Least Cost Branch and Bound]

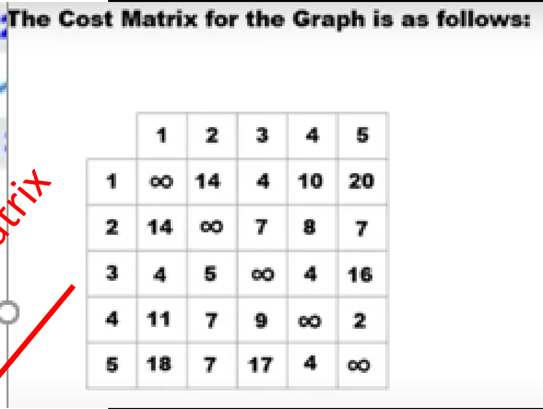


Among the Leaf node Values 31, 31, 31, 45, 45, 44, 30, 48 and 42, 30 is the Least Cost.

We have got the solution as 14523 and again back to 1. The Solution is as follows:

1 → 4 → 5 → 2 → 3 → 1

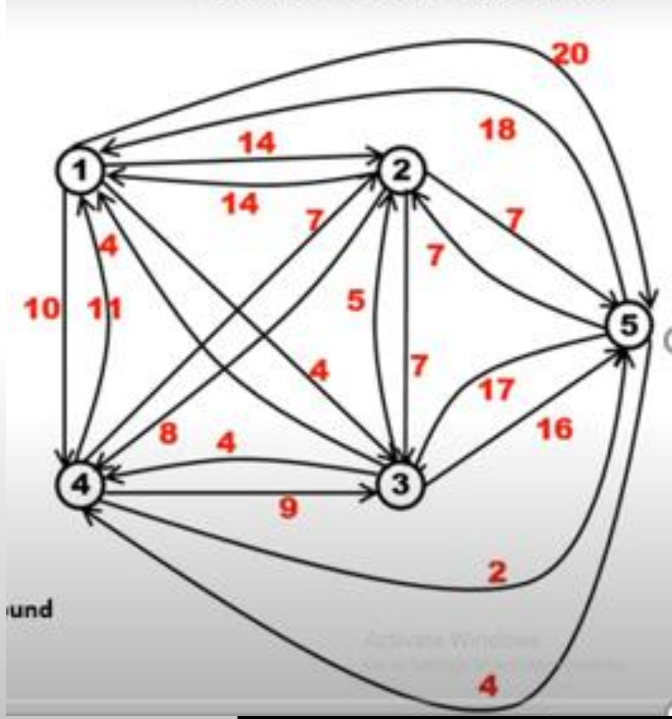
FINAL COST = 10 + 2 + 7 + 7 + 4 = 30



The Cost Matrix for the Graph is as follows:

	1	2	3	4	5
1	∞	14	4	10	20
2	14	∞	7	8	7
3	4	5	∞	4	16
4	11	7	9	∞	2
5	18	7	17	4	∞

Consider the following Graph:



und

Cost taken from original adjacency matrix

https://www.youtube.com/watch?v=HjSbaKF8Gi0