Design and Analysis of Algorithms

PCC-CS 501

Feasible Solution vs. Optimal Solution

- DFS, BFS, hill climbing and best-first search can be used to solve some searching problem for searching a feasible solution.

- However, they cannot be used to solve the optimization problems for searching an (the) optimal solution.

The branch-and-bound strategy

- This strategy can be used to solve optimization problems without an exhaustive search in the average case.

Branch-and-bound strategy

- 2 mechanisms:
  - A mechanism to generate branches when searching the solution space
  - A mechanism to generate a bound so that many branches can be terminated

Branch-and-bound strategy

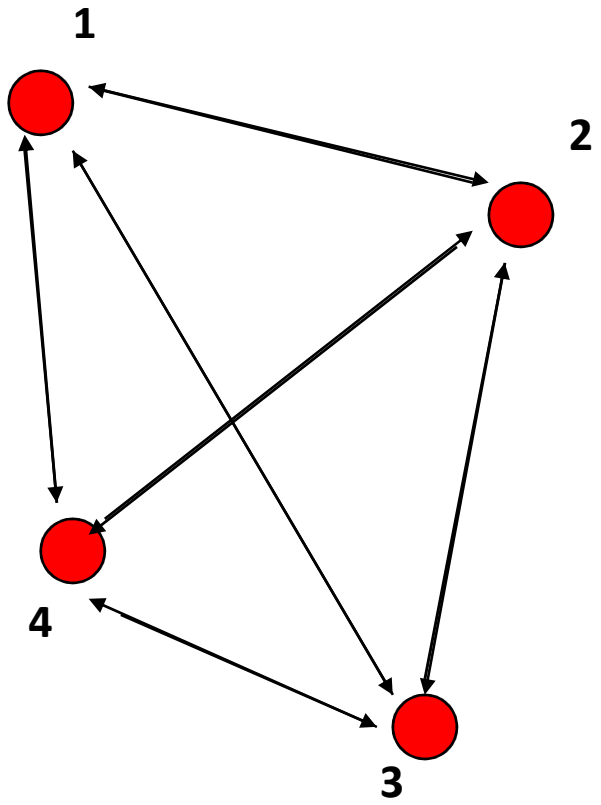
- It is efficient in the average case because many branches can be terminated very early.
- Although it is usually very efficient, a very large tree may be generated in the worst case.
- Many NP-hard problem can be solved by B&B efficiently in the average case; however, the worst case time complexity is still exponential.
- ***Branch and bound*** *is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. The Branch and Bound Algorithm technique solves these problems relatively quickly.*

Types

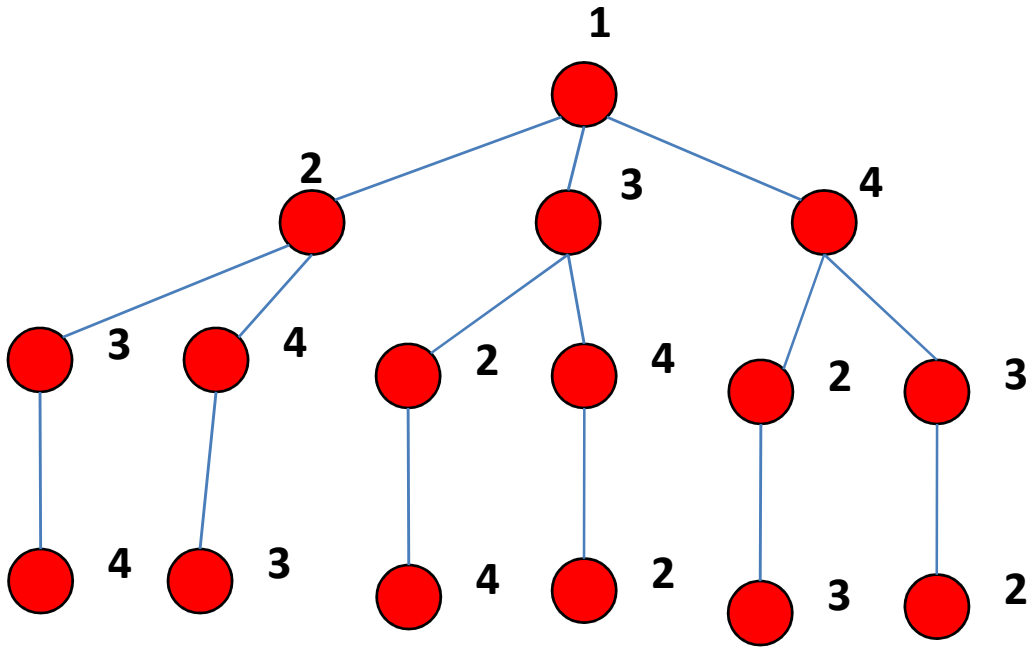
- FIFO Branch and Bound
- LIFO Branch and Bound
- Least Cost Branch and Bound

Travelling Salesman Problem using Branch and Bound

- The **travelling salesman problem** (also called the **travelling** salesperson **problem** or TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"



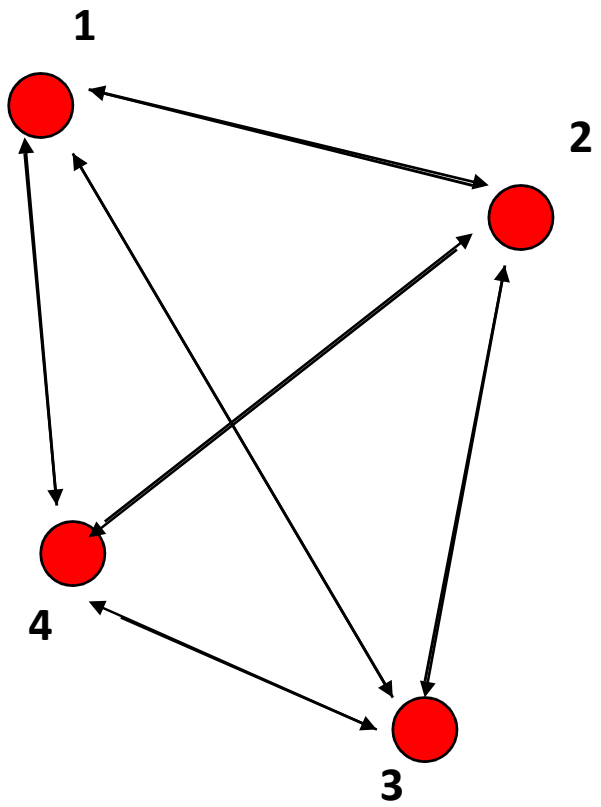
State Space Tree



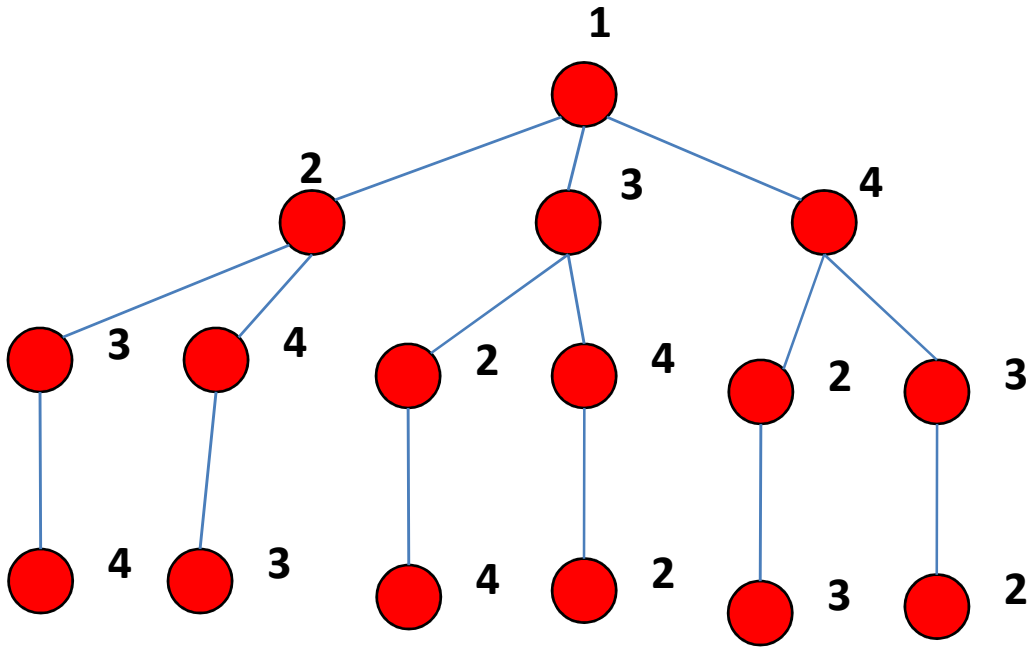
•Consider a salesman travelling from point 1 going to 4 and then back to 1. We need the cost of a tour for which the cost is minimum
We generate a state space tree, which explores all the different possibilities of the possible tour that might be conducted

•For branch and bound we will use level order

•Backtracking uses DFS, it is much time consuming as gives individual solutions for all path, and then if for any solution, a low value is found, then discards the others.
For optimization problems, we avoid backtracking, we use it for permutation problems.



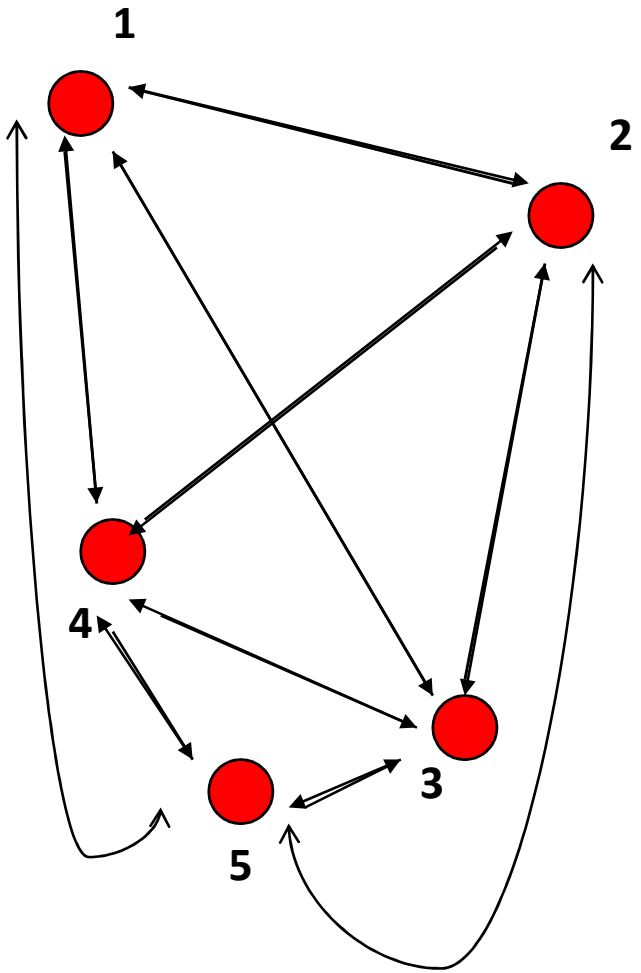
State Space Tree



•For branch and bound we will use level order. We will generate the cost of every node while generating the levels.

•For any node, if the cost of a node is greater than some upper bound then we will kill that node. No further children will be explored for that node. We will not generate all the nodes.

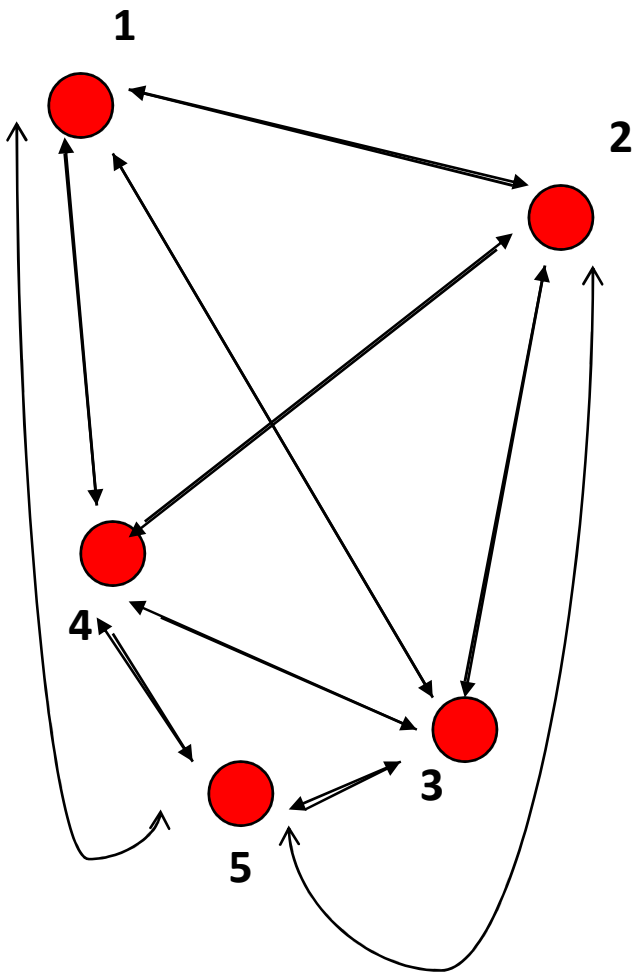
•We will only generate nodes which are fruitful, i.e. we will try the path, i.e. the ones which takes us closer to the optimized solution path.



Cost Adjacency Matrix

	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

Traveling Salesperson Problem: Find the least cost tour starting at 1, traveling through the other nodes exactly once and returning to 1.

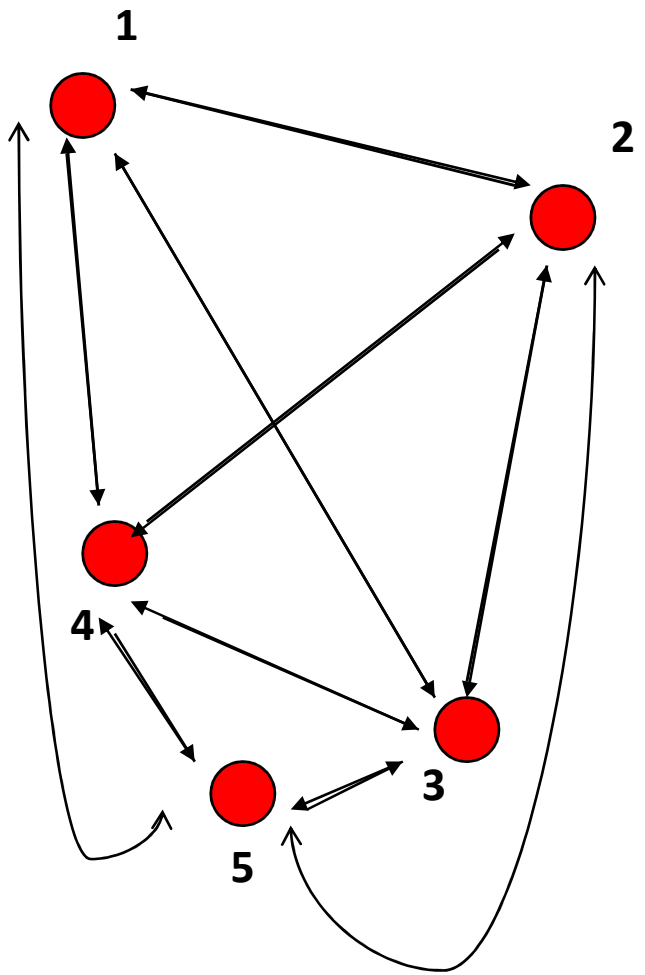


Cost Adjacency Matrix

Step 1 to reduce: Search each row for the smallest value

	1	2	3	4	5	
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4

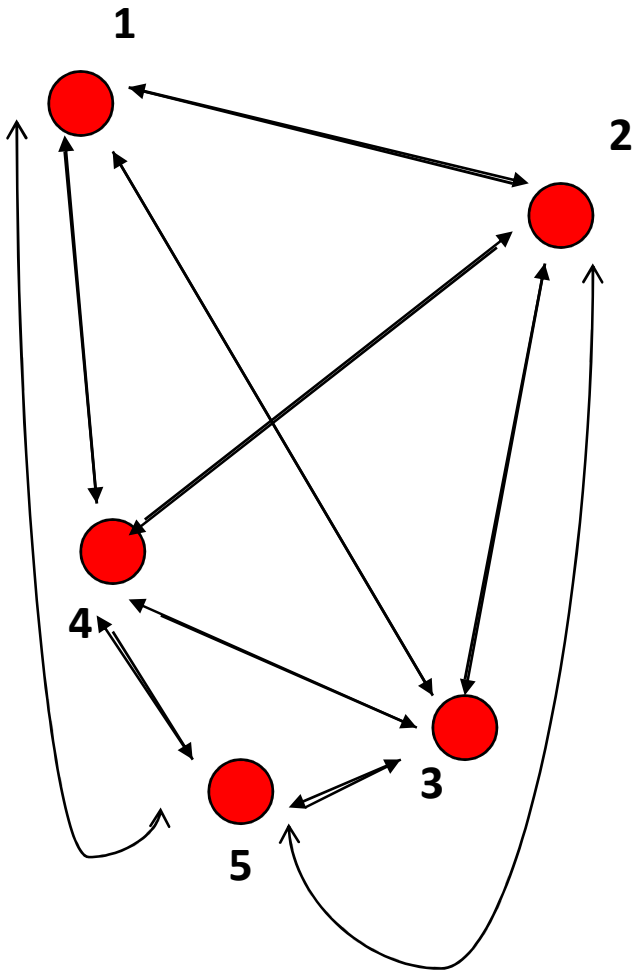
Traveling Salesperson Problem: Find the least cost tour starting at 1, traveling through the other nodes exactly once and returning to 1.



Cost Adjacency Matrix

Subtract each rows with the smallest value

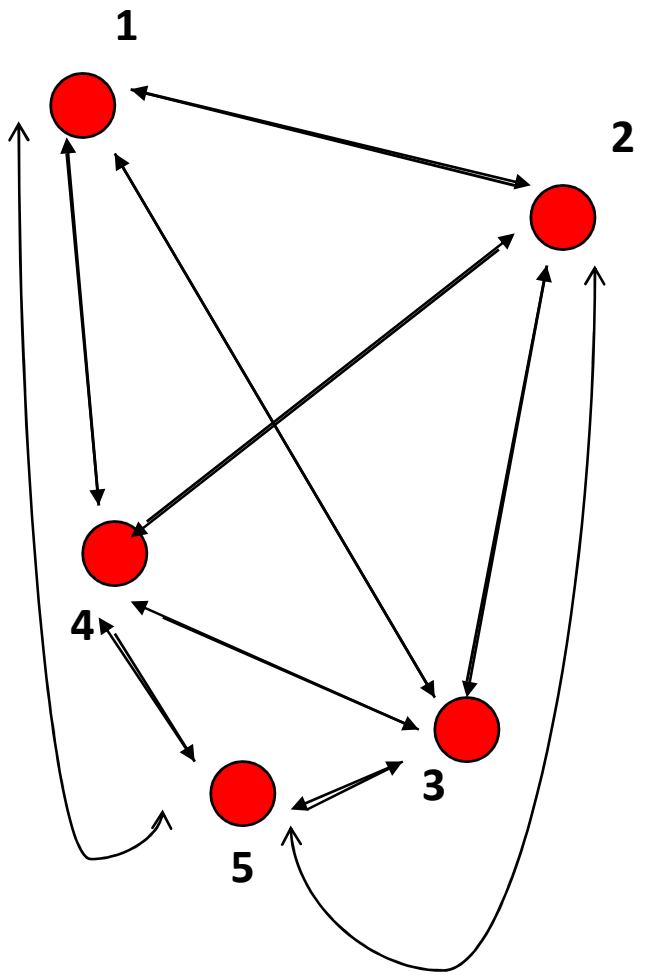
	1	2	3	4	5	
1	∞	10	20	0	1	10
2	13	∞	14	2	0	2
3	1	3	∞	0	2	2
4	16	3	15	∞	0	3
5	12	0	3	12	∞	4
						21



Cost Adjacency Matrix

Step 2 to reduce: Search each column for the smallest value

	1	2	3	4	5	
1	∞	10	20	0	1	10
2	13	∞	14	2	0	2
3	1	3	∞	0	2	2
4	16	3	15	∞	0	3
5	12	0	3	12	∞	4
	1	0	3	0	0	21

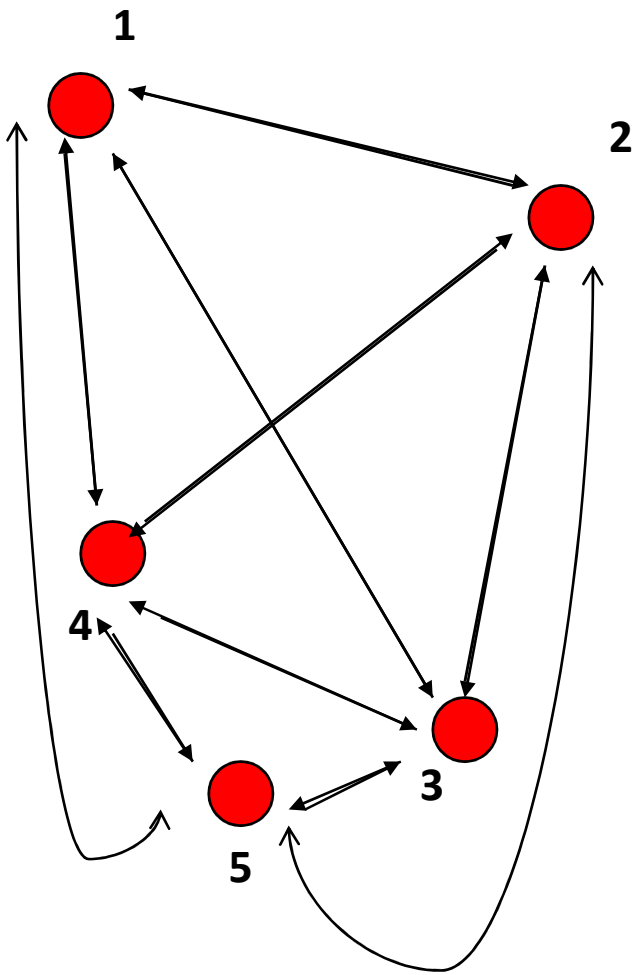


Cost Adjacency Matrix

Subtract each column with the smallest value

	1	2	3	4	5	
1	∞	10	17	0	1	10
2	12	∞	11	2	0	2
3	0	3	∞	0	2	2
4	15	3	12	∞	0	3
5	11	0	0	12	∞	4
	1	0	3	0	0	

21
+
4
=25



Cost Adjacency Matrix

Subtract each column with the smallest value

	1	2	3	4	5	
1	∞	10	17	0	1	10
2	12	∞	11	2	0	2
3	0	3	∞	0	2	2
4	15	3	12	∞	0	3
5	11	0	0	12	∞	4
	1	0	3	0	0	

21
+
4
=25

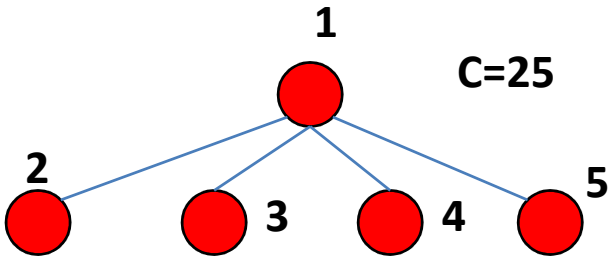
Reduced cost: 25

i.e. The minimum cost of the tour is at least, it also may be more than 25

Reduced matrix

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

State Space Tree

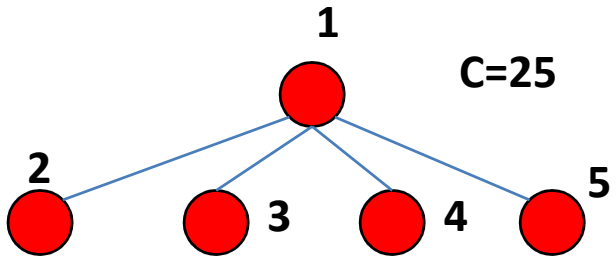


Now we will find the cost with the help of the state space tree
For every node, we will be finding a cost
Cost of 1st matrix= 25 i.e. the reduced cost
We will be maintaining a upper bound value which will be initially ∞
This will update once we have reached the leaf node (the value of upper will not be checked for every node)

Reduced matrix

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

State Space Tree

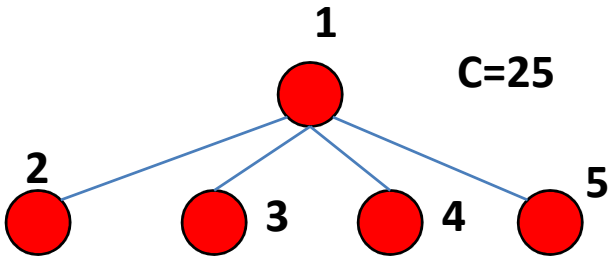


We will be finding the cost of 2nd node 2, i.e. node 1 to node 2 **(1,2)**
Make the 1st row and 2nd column as ∞

Reduced matrix

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

State Space Tree

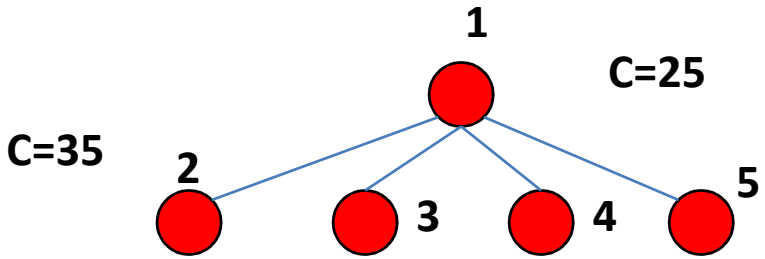


We will be finding the cost of 2nd node 2, i.e. node 1 to node 2 (1,2)
Make the 1st row and 2nd column as ∞
Once we come from 1 to 2, we should not go back to 1, so (2,1) is also ∞

Reduced matrix

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

State Space Tree



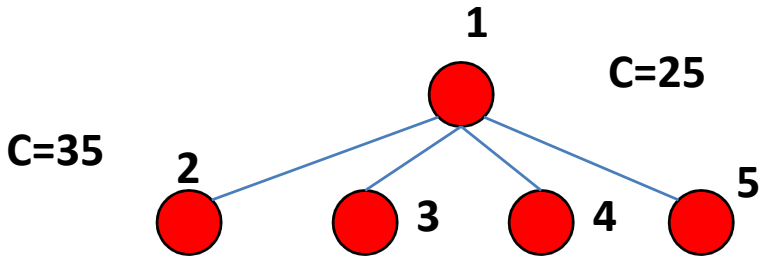
We will be finding the cost of 2nd node 2, i.e. node 1 to node 2 (1,2)
Make the 1st row and 2nd column as ∞
Once we come from 1 to 2, we should not go back to 1, so (2,1) is also ∞
Now check if the matrix is reduced or not (i.e. the minimum values for all rows and colums are 0 or not)

So cost is C(1,2) + previous reduced cost + any more reduction done
= 10 + 25 + 0 = 35 (cost of 2nd node is 35)

Reduced matrix

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

State Space Tree



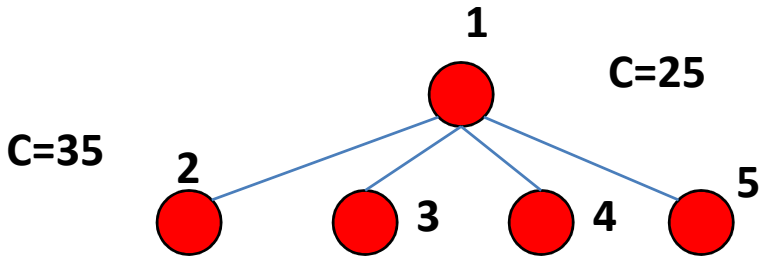
We will be finding the cost of 2nd node 2, i.e. node 1 to node 2 (1,2)
Make the 1st row and 2nd column as ∞
Once we come from 1 to 2, we should not go back to 1, so (2,1) is also ∞
Now check if the matrix is reduced or not

So cost is C(1,2) + previous reduced cost + any more reduction done
= 10 + 25 + 0 = 35

Reduced matrix

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

State Space Tree

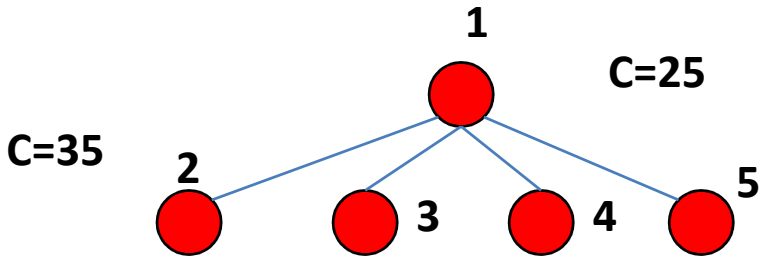


We will be finding the cost of 3rd node 3, i.e. node 1 to node 3 **(1,3)**
Make the 1st row and 3rd column as ∞

Reduced matrix

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	∞	2	0
3	∞	3	∞	0	2
4	15	3	∞	∞	0
5	11	0	∞	12	∞

State Space Tree

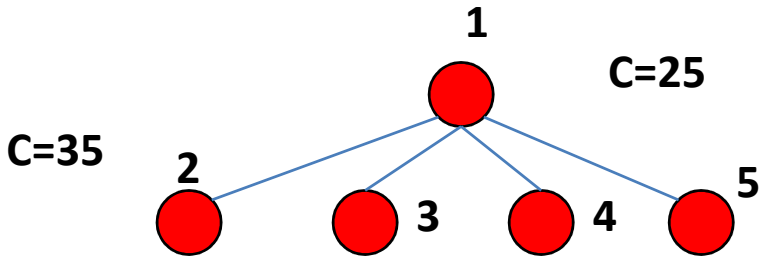


We will be finding the cost of 3rd node 3, i.e. node 1 to node 3 (1,3)
Make the 1st row and 3rd column as ∞
Also, we should not have any path going back from 3 to 1, so (3,1) is also ∞

Reduced matrix

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	
2	12	∞	∞	2	0	0
3	∞	3	∞	0	2	0
4	15	3	∞	∞	0	0
5	11	0	∞	12	∞	0
	11	0		0	0	

State Space Tree

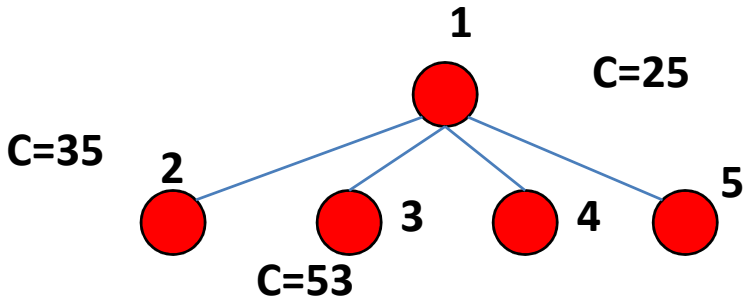


We will be finding the cost of node 3, i.e. node 1 to node 3 (1,3)
Make the 1st row and 3rd column as ∞
Also, we should not have any path going back from 3 to 1, so (3,1) is also ∞
Check for zeros for all rows and columns, i.e. for reduction
If not reduced then reduce with the minimum value found

Reduced matrix

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

State Space Tree



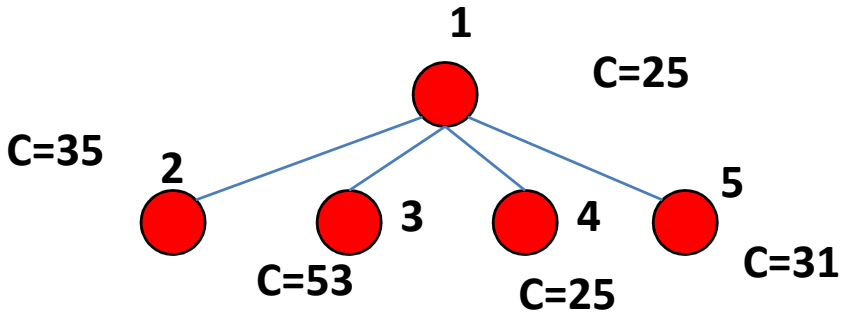
We will be finding the cost of 3rd node 3, i.e. node 1 to node 3 (1,3)
Make the 1st row and 3rd column as ∞
Also, we should not have any path going back from 3 to 1, so (3,1) is also ∞
Check for zeros for all rows and columns, i.e. for reduction

So cost= C(1,3) + previous reduced cost + any more reduction done = 17+25+11=53

Reduced matrix

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

State Space Tree

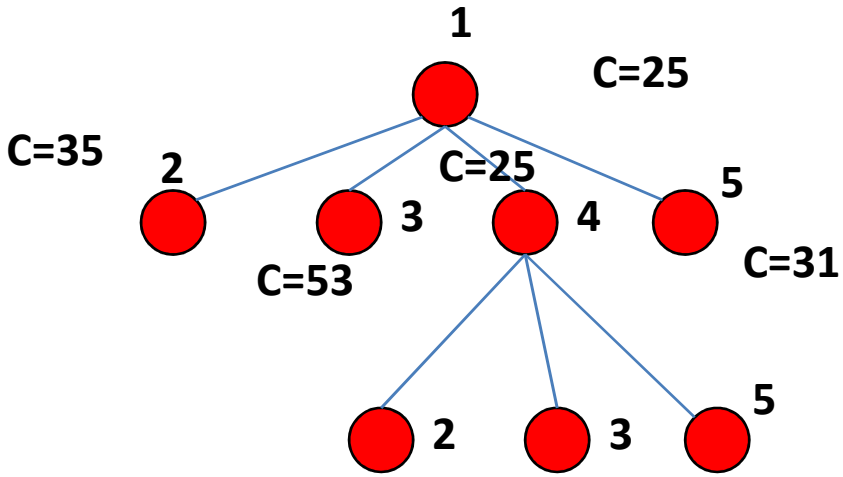


Similarly we repeat for 4th and 5th vertex and get Cost of 4 as 25 and cost of 5 as 31
Moving to the next level vertices, we should see what we should explore next.
Now, if we explore the node of children sequentially, i.e. 1st of 2, then of 3, then of 4 and lastly of 5, then it is known as FIFO branch and bound. The reverse order is LIFO branch and bound.
Else if we explore the children of least cost node 1st and later the others as required then it is Least cost branch and bound (LC branch and bound)

Reduced matrix of node 4, i.e. (1,4)

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

State Space Tree

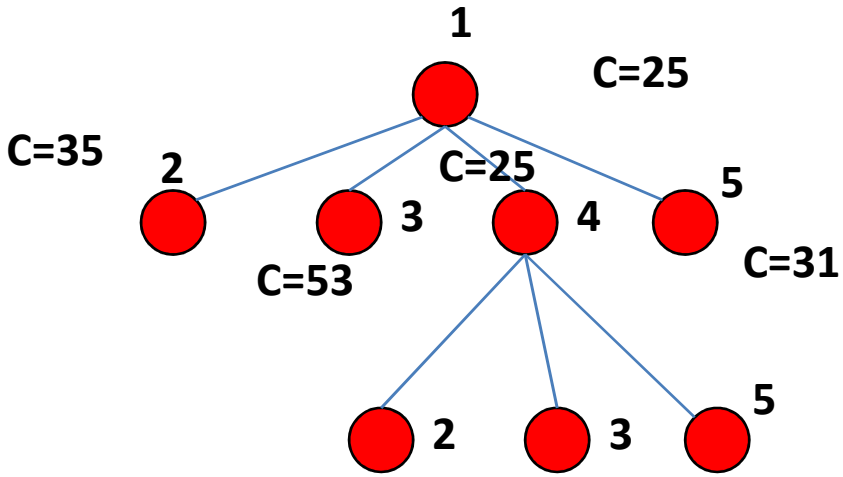


If we explore the children of nodes 4, we have vertex 2, 3 and 5, we consider the matrix given above

Reduced matrix of node 4

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

State Space Tree



If we explore the children of nodes 4, we have vertex 2, 3 and 5, we consider the matrix given above

Now for the vertex (4,2), we have 4^{th} rows as ∞ and 2^{nd} columns as ∞

Consider the path 1-4 and 4-2, so it should not go back to 1, therefore (2,1) is ∞

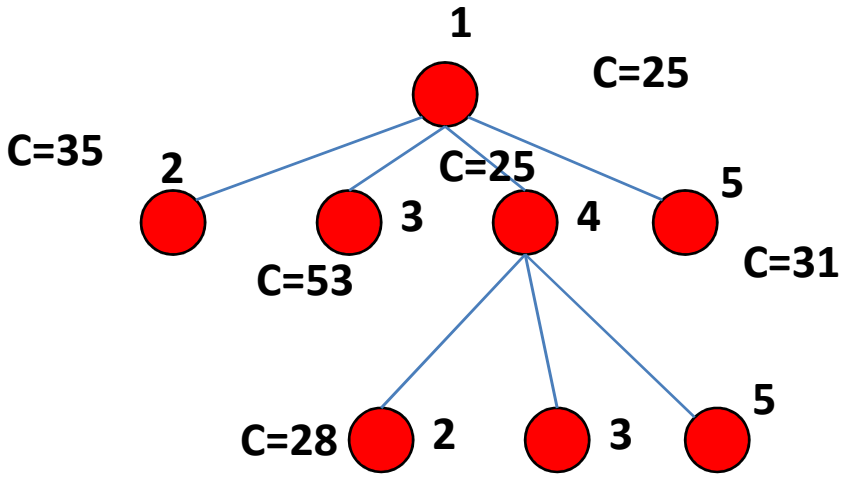
All rows and columns are reduced (i.e. has 0)

Now Cost is C(4,2)+ C(4)+ cost of reduction = 3+25+0=28

Reduced matrix of node 4

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

State Space Tree



If we explore the children of nodes 4, we have vertex 2, 3 and 5, we consider the matrix given above

Now for the vertex (4,2), we have 4^{th} rows as ∞ and 2^{nd} columns as ∞

Consider the path 1-4 and 4-2, so it should not go back to 1, therefore (2,1) is ∞

All rows and columns are reduced (i.e. has 0)

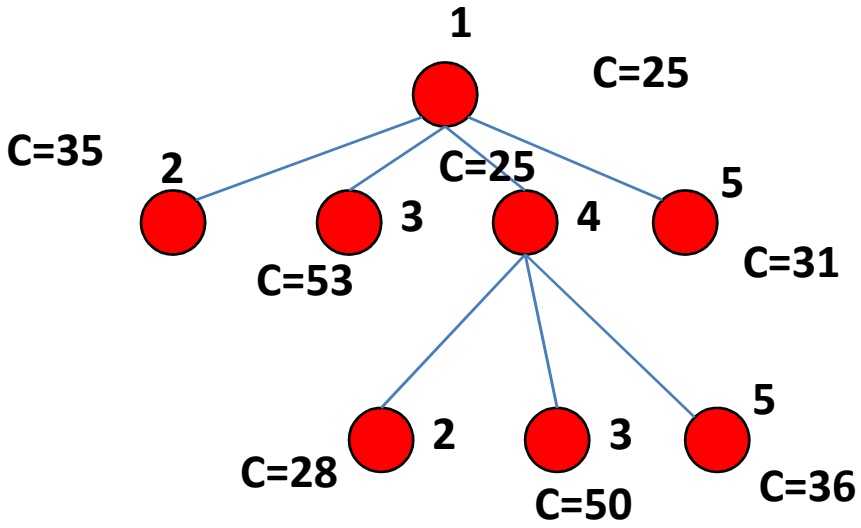
Now Cost is C(4,2)+ C(4)+ cost of reduction

= 3+25+0=28

Reduced matrix of node 4

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

State Space Tree

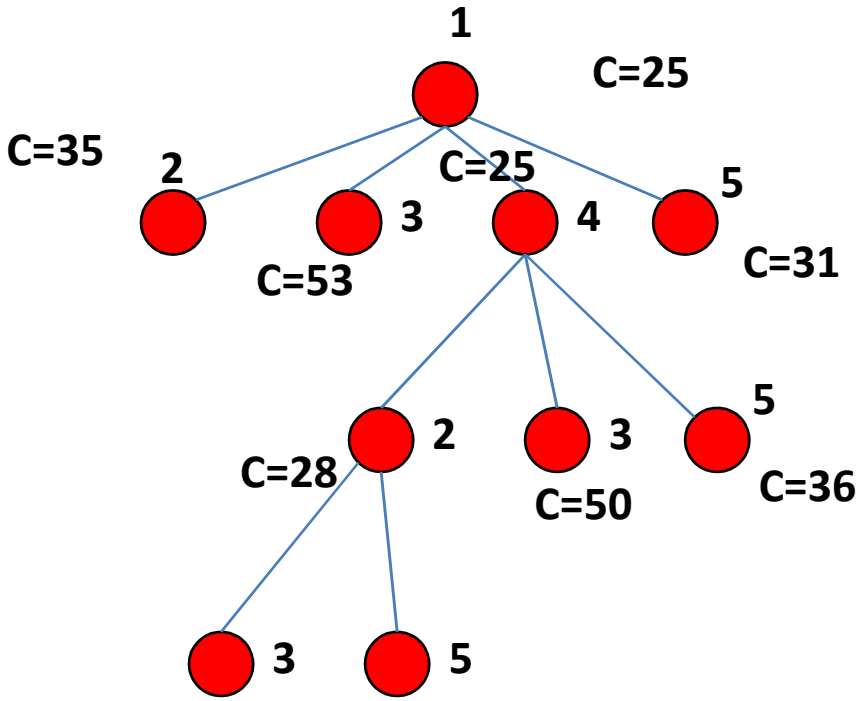


If we explore the children of nodes 4, we have vertex 2, 3 and 5, we consider the matrix given above
Similarly for (4,3) and (4,5) is found to be 50 and 36 respectively
Now amongst all cost we again get 28 as least, so we should explore that path, i.e. (4,2)

Reduced matrix of node 4

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

State Space Tree



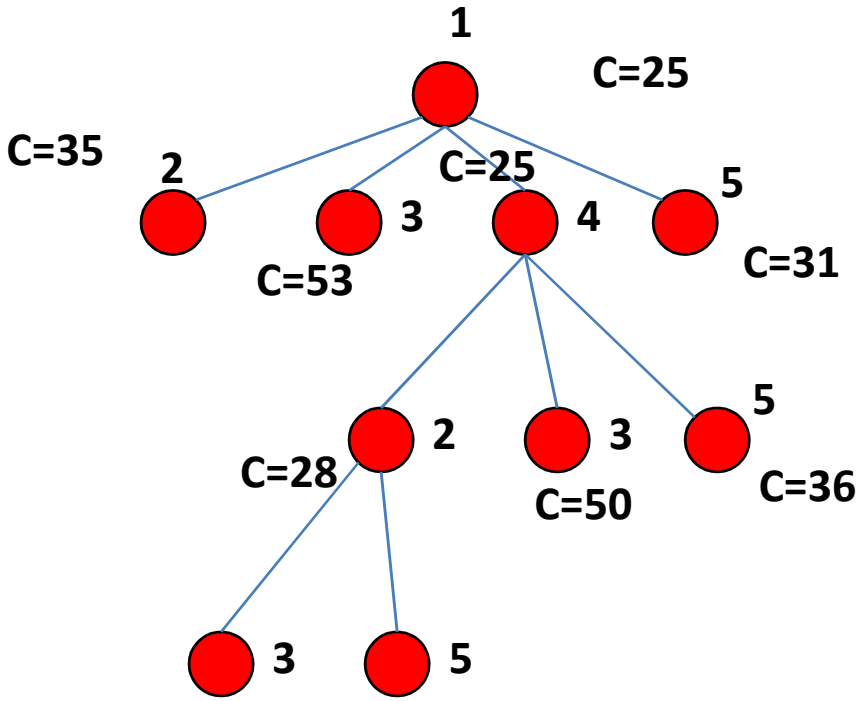
If we explore the children of nodes 4, we have vertex 2, 3 and 5, we consider the matrix given above
Similarly for (4,3) and (4,5) is found to be 50 and 36 respectively
Now amongst all cost we again get 28 as least, so we should explore that path, i.e. (4,2)

So nodes remaining for the chosen path are 3 and 5.
Lets explore them

Reduced matrix of (4,2)

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

State Space Tree

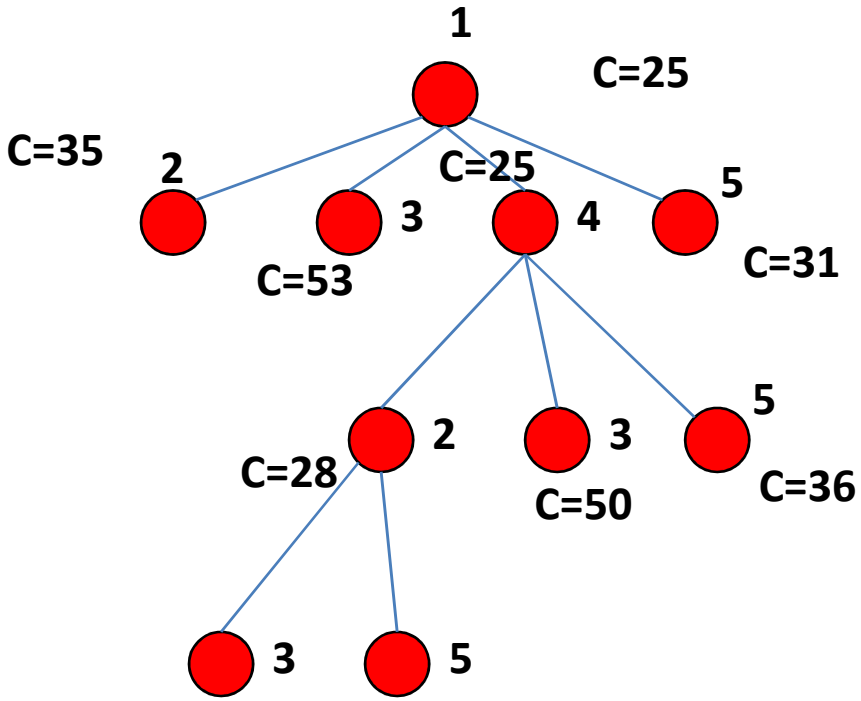


For the path (2,3), we have taken the reduced matrix as obtained priorly

Reduced matrix of (4,2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	∞	∞	∞

State Space Tree



For the path (2,3), we have taken the reduced matrix as obtained priorly.

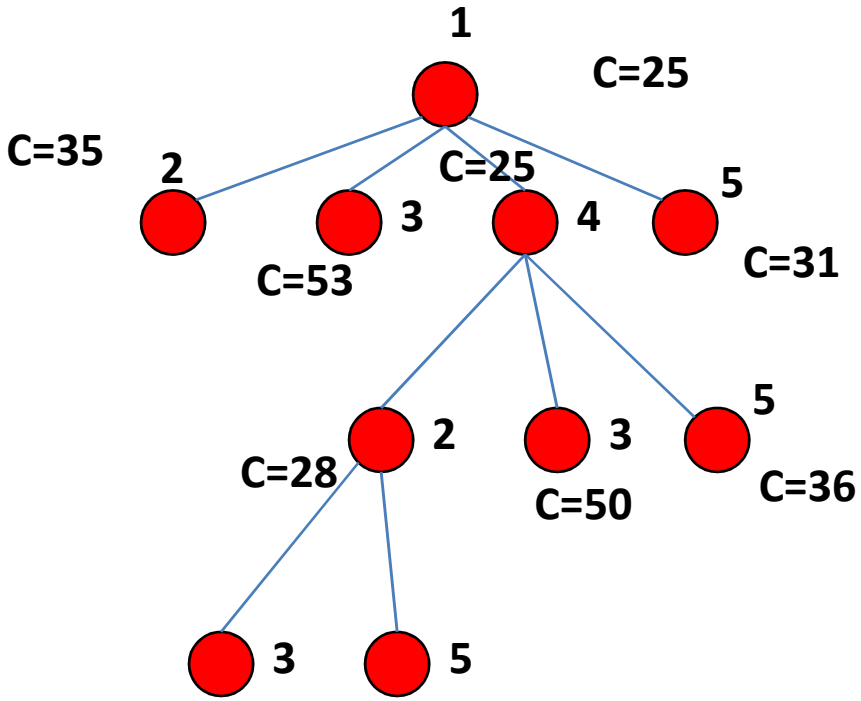
2^{nd} row and 3^{rd} column is made ∞

Again (3,1) is made ∞, as we cant go back from 3 to 1 via this path.

Reduced matrix of (4,2)

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	
2	∞	∞	∞	∞	∞	
3	∞	∞	∞	∞	2	2
4	∞	∞	∞	∞	∞	
5	11	∞	∞	∞	∞	11

State Space Tree



For the path (2,3), we have taken the reduced matrix as obtained priorly.

2nd row and 3rd column is made ∞

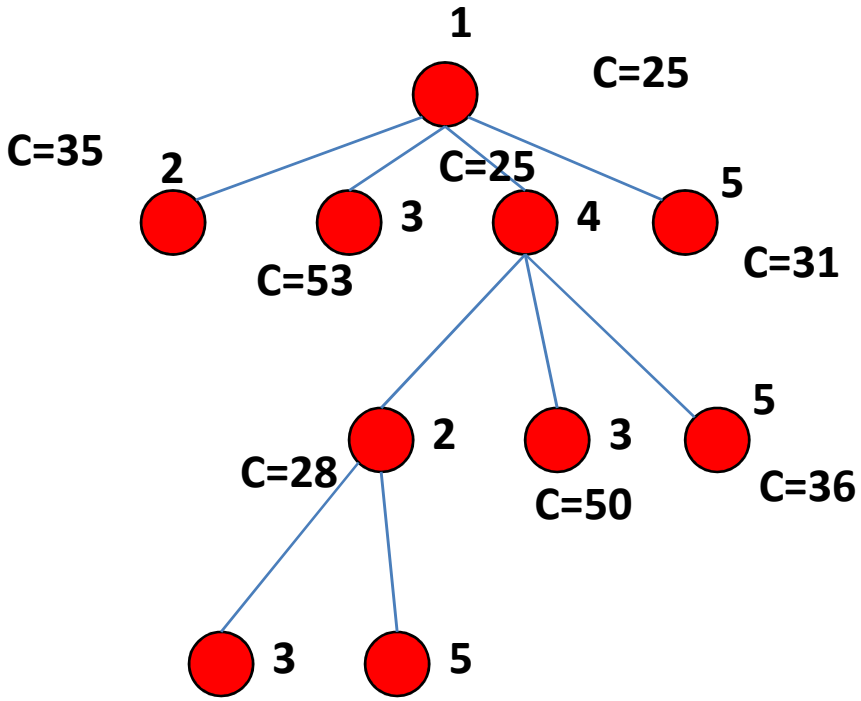
Again (3,1) is made ∞, as we cant go back from 3 to 1 via this path.

Start reducing.

Reduced matrix of (4,2)

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	
2	∞	∞	∞	∞	∞	
3	∞	∞	∞	∞	0	2
4	∞	∞	∞	∞	∞	
5	0	∞	∞	∞	∞	11
						13

State Space Tree

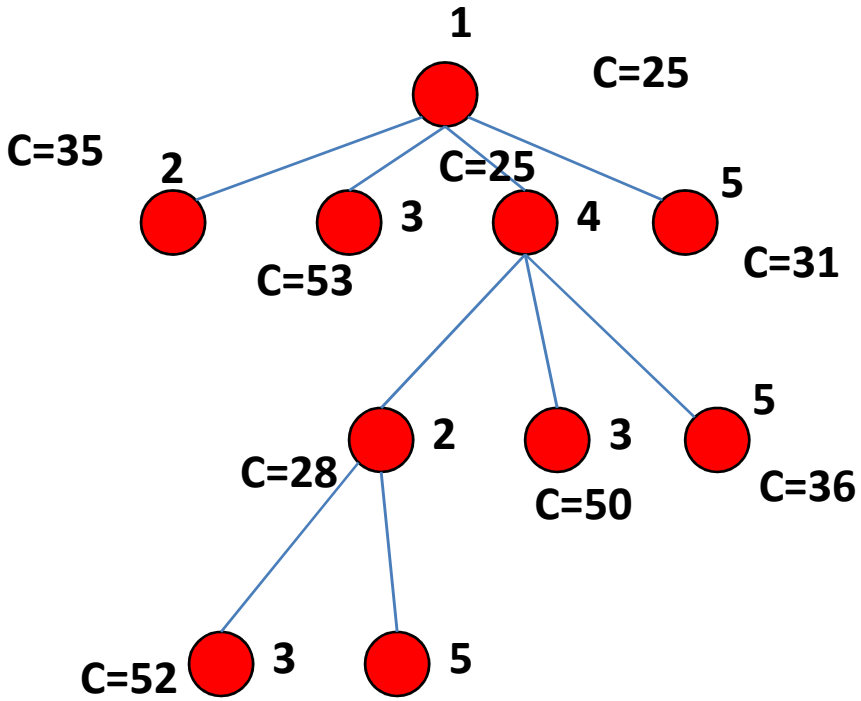


For the path (2,3), we have taken the reduced matrix as obtained priorly.
2nd row and 3rd column is made ∞
Again (3,1) is made ∞, as we cant go back from 3 to 1 via this path.
Start reducing.
Now cost is evaluated as C(2,3)+ previous reduced cost of respective node + any more reduction done
=11+28+13= 52

Reduced matrix of (4,2)

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	
2	∞	∞	∞	∞	∞	
3	∞	∞	∞	∞	0	2
4	∞	∞	∞	∞	∞	
5	0	∞	∞	∞	∞	11
						13

State Space Tree

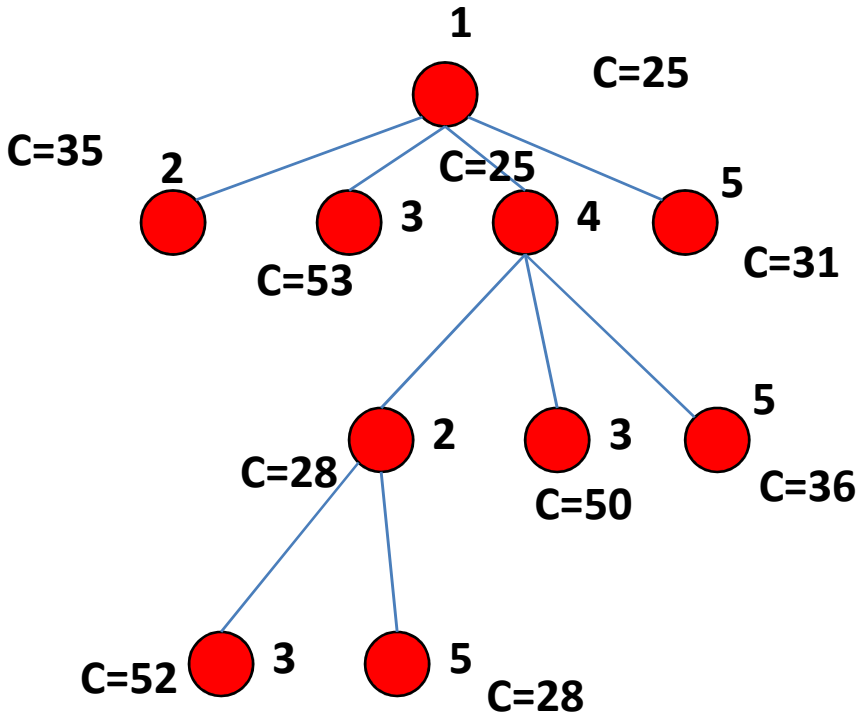


For the path (2,3), we have taken the reduced matrix as obtained priorly.
2^{nd} row and 3^{rd} column is made ∞
Again (3,1) is made ∞, as we cant go back from 3 to 1 via this path.
Start reducing.
Now cost is evaluated as C(2,3)+ previous reduced cost + any more reduction done
=11+28+13= 52

Reduced matrix of (4,2)

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

State Space Tree

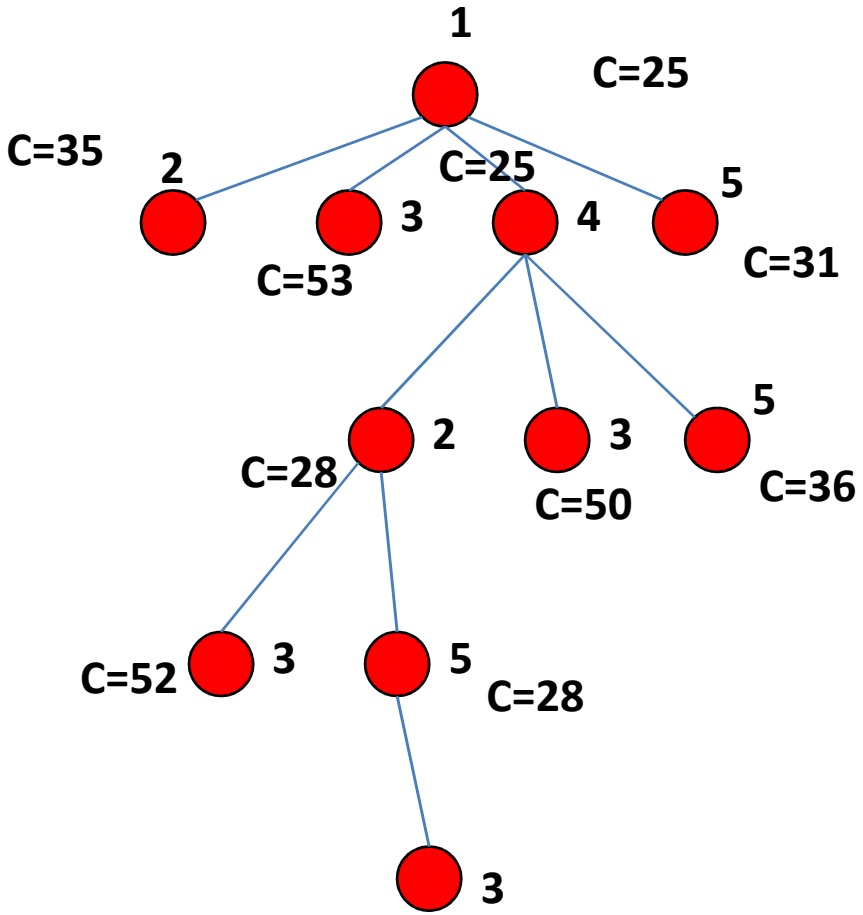


For the path (2,5), we have taken the reduced matrix as obtained priorly.
Similarly, on calculating as before we get the cost as 28
Therefore for this path we get again the least as 28.
Hence select this path and explore its children.

Reduced matrix of (2,5)

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

State Space Tree

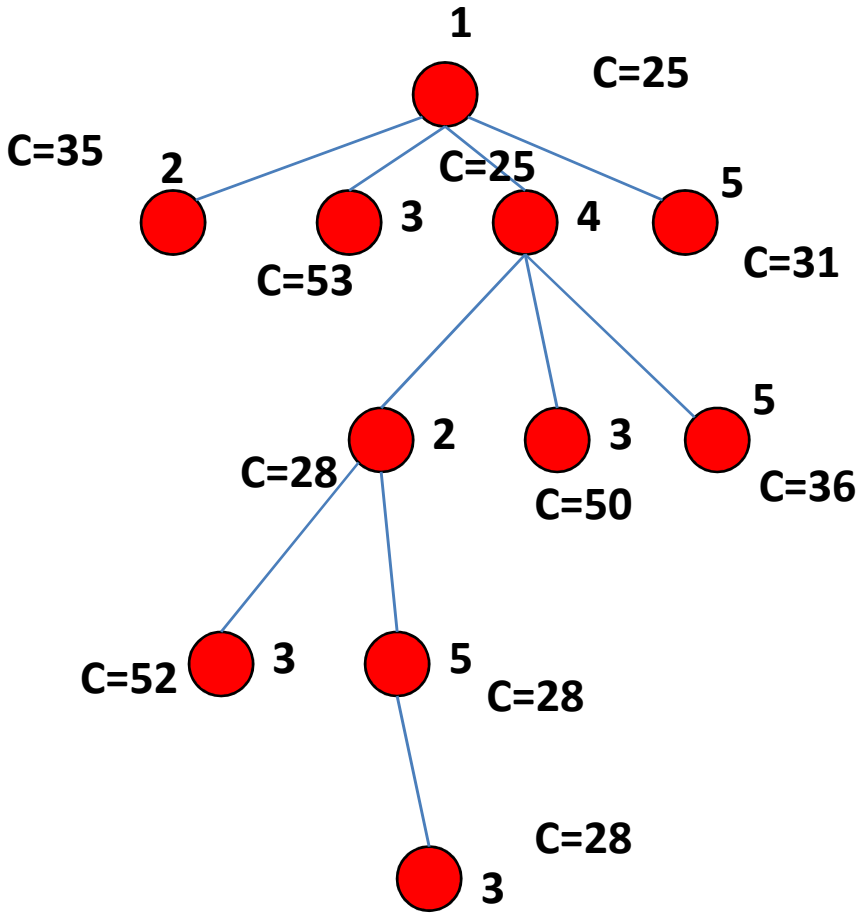


For the path (2,5), we have taken the reduced matrix as obtained priorly.
Similarly, on calculating as before we get the cost as 28
Therefore for this path we get again the least as 28.
Hence select this path and explore its children. Only 3 is left to be explored for this path.

Reduced matrix of (2,5)

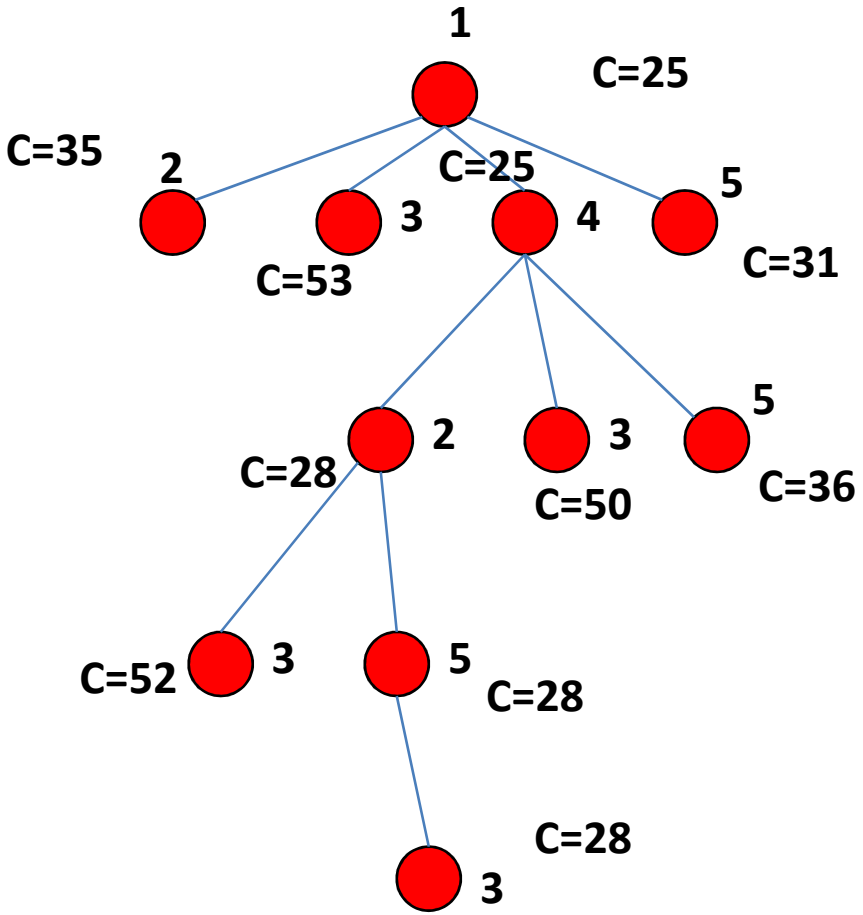
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	0	∞	∞

State Space Tree



Here nothing is left, so we will have the least cost as previously i.e. 28

We reached the end, now we have to move back to 1.
Since we have reached the end, i.e. leaf node, update upper to 28 here.
Every node whose cost is greater than upper, i.e. 28 gets killed here, i.e. we don't need to explore them.



The cost of the tour is 28 and the path is 1-4-2-5-3-1
This is the shortest tour of all the nodes.