Module-2

Back Tracking & Branch and Bound

BCSC0012

Designing Techniques of Algorithms

Design and Analysis of Algorithms

Dr. Gaurav Kumar
Asst. Prof, CEA, GLAU, Mathura

Designing Techniques of Algorithms

01

Divide and Conquer Strategy

02

Greedy Technique

03

Dynamic Programming

04

Back Tracking & Branch and Bound

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

04

Back Tracking, Branch and Bound

Back Tracking

01

Backtracking is nothing but the modified process of the **Brute force approach**

02

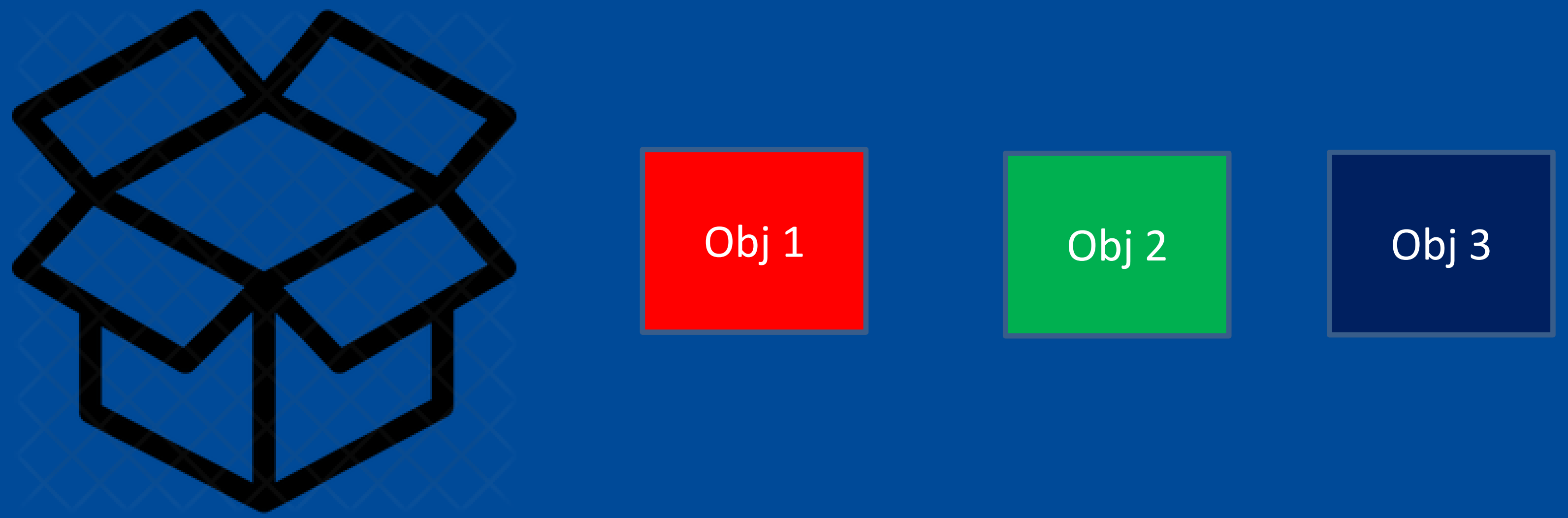
It is a technique where multiple solutions to a problem are available, and it searches for the solution among all the available options.

Brute Force Approach - It is an intuitive, direct, and straightforward technique of problem-solving in which all the possible ways or all the possible solutions to a given problem are enumerated. Example: Daily Life Activities (Home to College/Market)

Understanding Brute Force with Example

Example

Consider the box, and we have three objects of three different colors. One object is of red color, the second object is of green color, and the third object is of blue color.



- Now, we have to keep these objects inside the box.
- We have multiple options of placing the order of object in the box.
- As per the Brute force approach, we have to consider all the options and identify the best option among all the possible options.

Understanding Brute Force with Example

Obj 1	Obj 2	Obj 3

Suppose we first fill red object, after then green object and then blue object. This is the first solution to this problem. The possible solution is red, green, and blue.

Solution 1	Obj 1	Obj 2	Obj 3
Solution 2	Obj 1	Obj 3	Obj 2
Solution 3	Obj 2	Obj 1	Obj 3
Solution 4	Obj 2	Obj 3	Obj 1
Solution 5	Obj 3	Obj 1	Obj 2
Solution 6	Obj 3	Obj 2	Obj 1

These are the all possible solutions, and we have to identify the best possible solution out of all the possible solutions. This approach is known as a **brute force approach**.

Backtracking is similar to the brute force approach, but it is a modified process of the brute force approach.

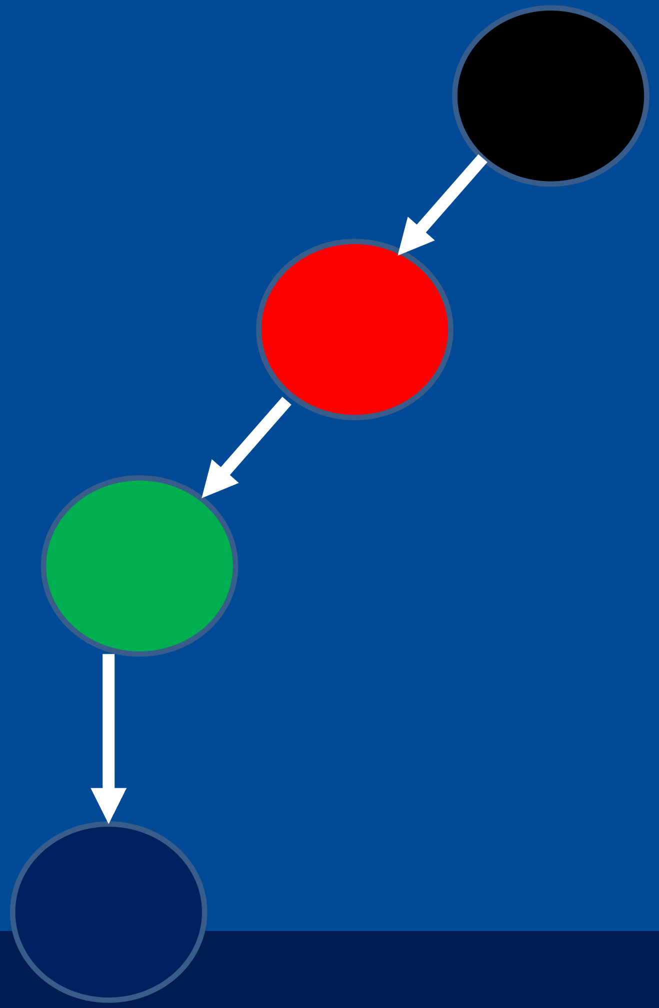
Understanding Back Tracking

- In backtracking, solutions are represented in the form of a tree and that tree is known as a **State Space Tree**.

- Backtracking follows the **DFS strategy** to form a State Space Tree.

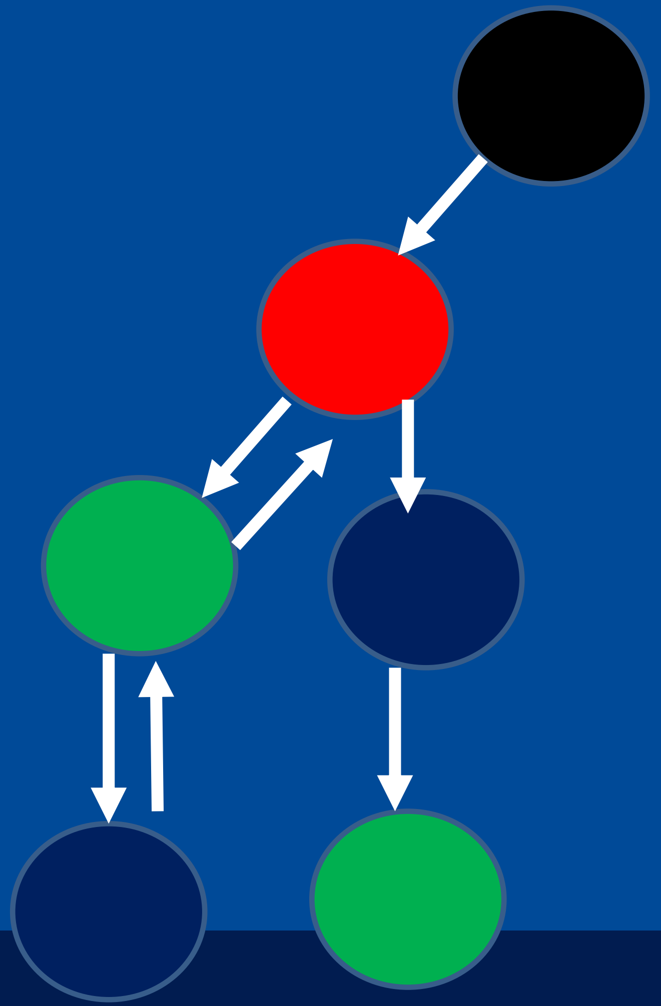
Understanding Back Tracking

Lets take the same previous example. Consider the first solution, i.e., **red, green, blue,** and it can be represented in the form of a tree as shown as below.



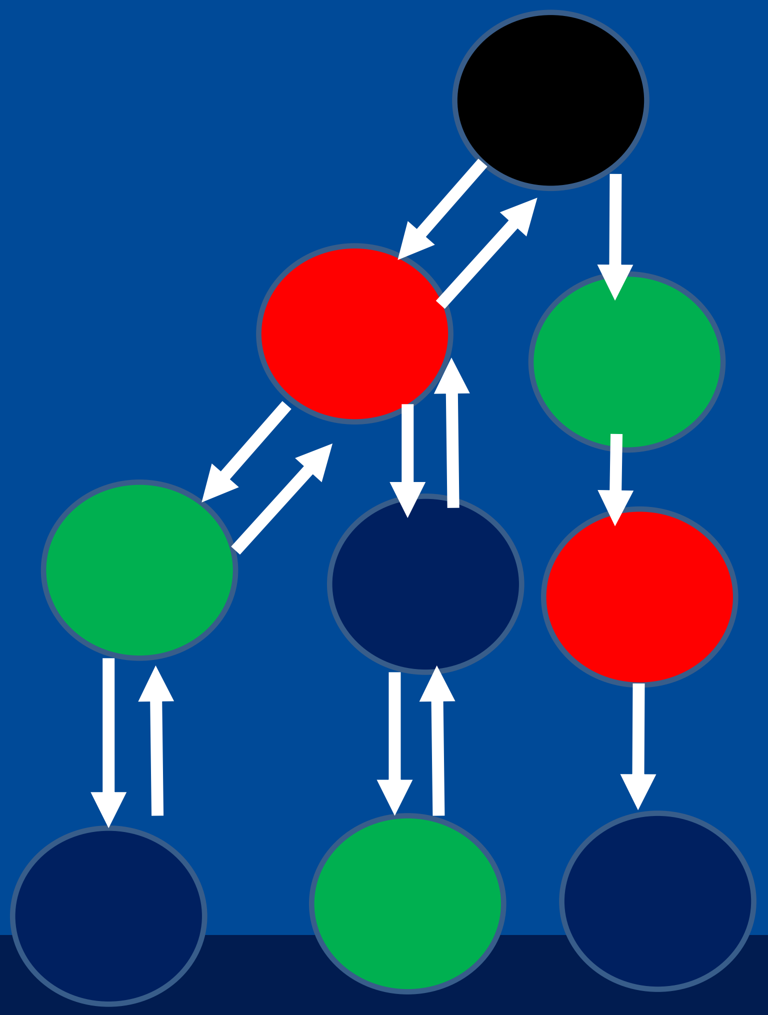
Understanding Back Tracking

Consider the second solution, i.e., **red, blue, green**. Since there is no more object after blue so we will backtrack and move to the Red. After Red, instead of taking green, we first take blue and then green, shown as below



Understanding Back Tracking

The next solution is **green, red and blue.** Since we cannot explore the green object, so we move back and reach the blue object.



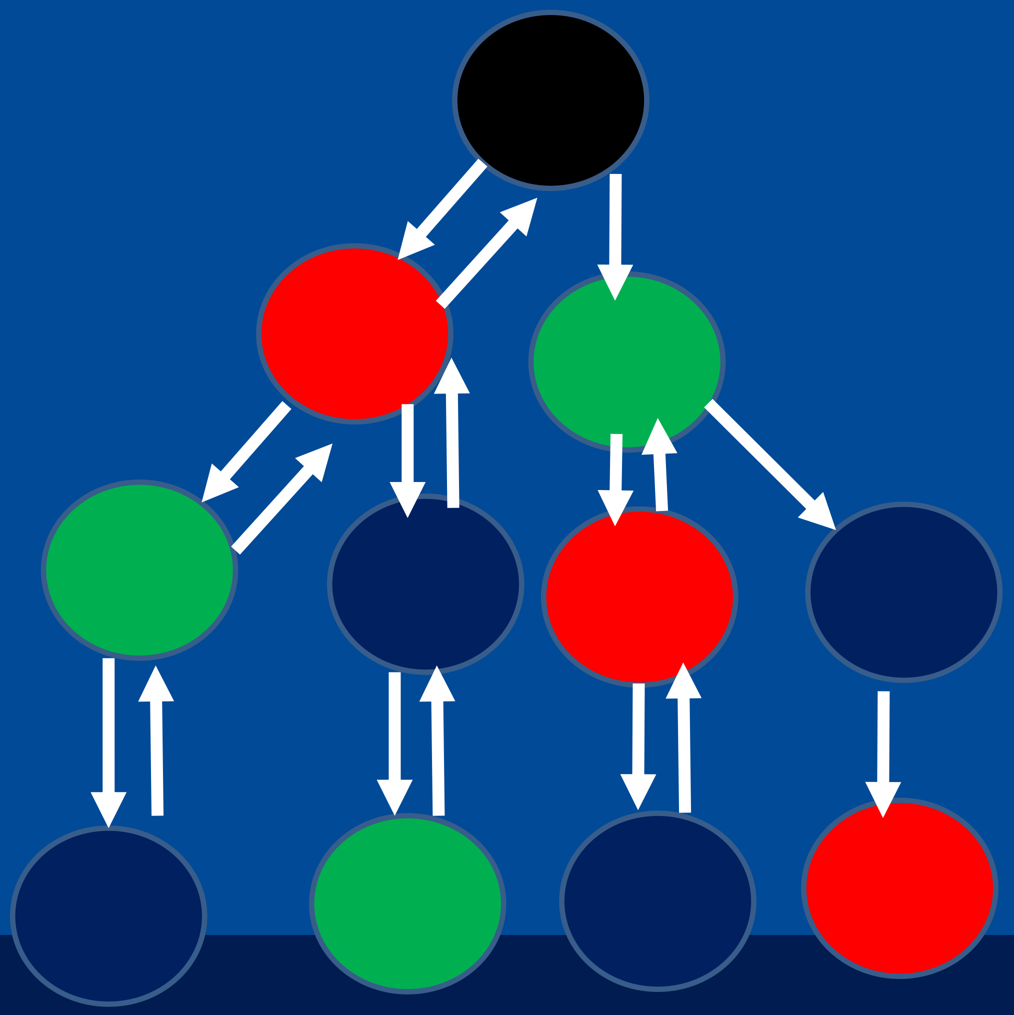
We cannot explore the blue object, so we again move back and reach to the red object. Instead of using the red object, we will use the **green object**.

After the green object, we use the red object and then we use the blue object.

We cannot explore the blue object further. Now the sequence green, red and blue is formed as shown as below.

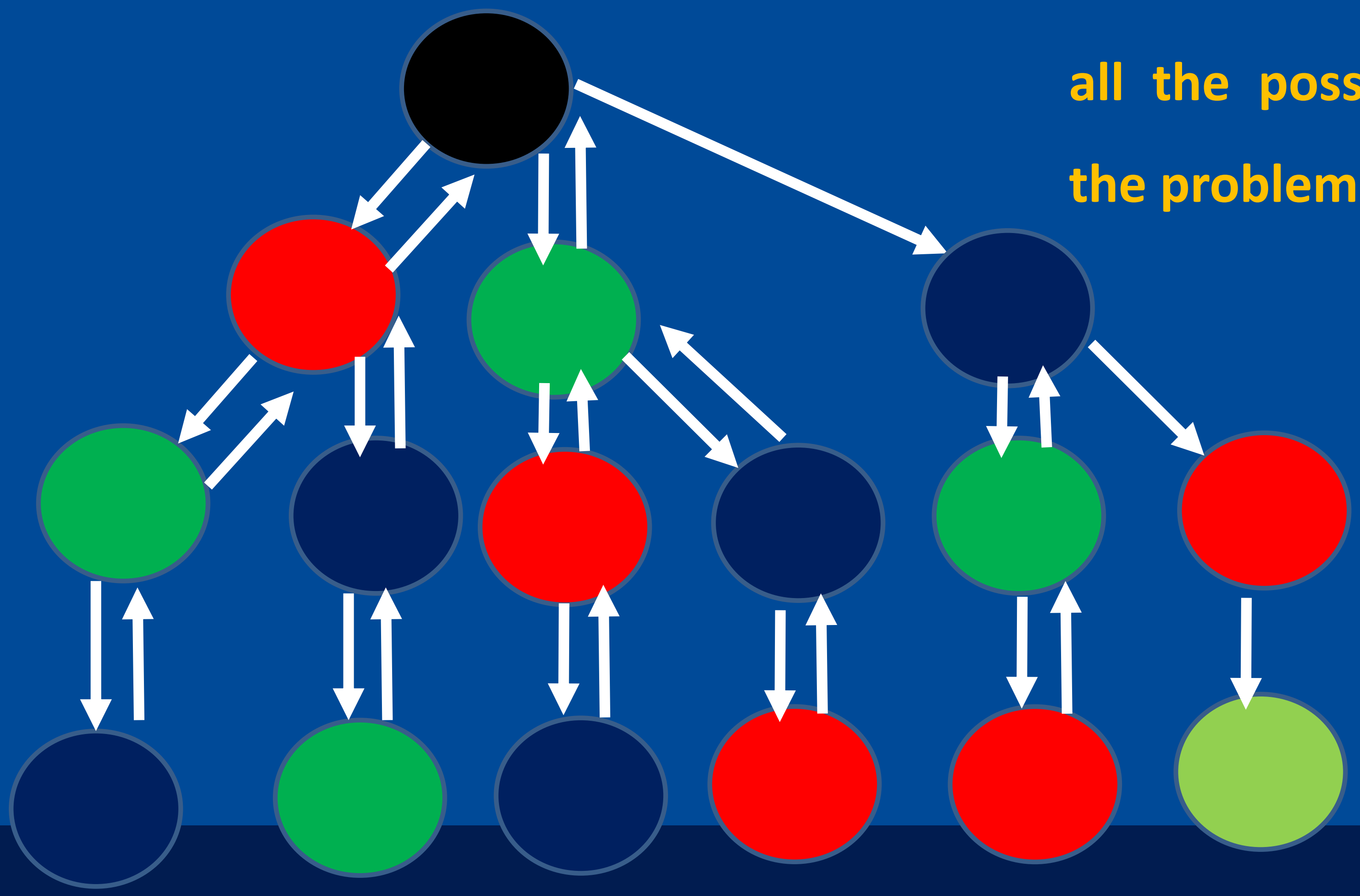
Understanding Back Tracking

The next solution is **green, blue and red**. Since we cannot explore the blue object, we move back and continue.



Understanding Back Tracking

The next solution is **blue, red and green.** Since we cannot explore red object so we backtrack and reach to the green object.

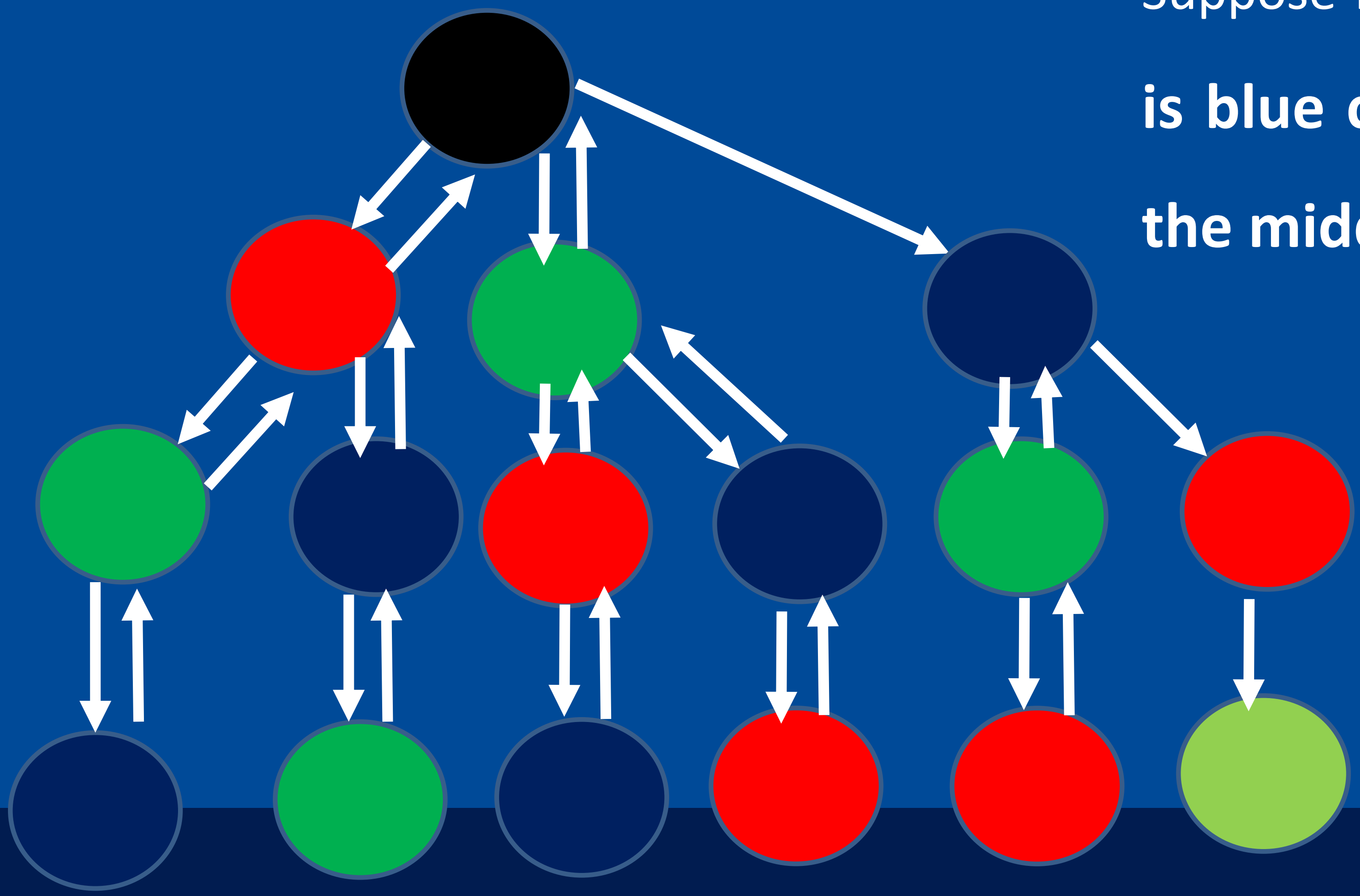


This is the state space tree that shows all the possible solutions related to the problem.

Solution 1	Obj 1	Obj 2	Obj 3
Solution 2	Obj 1	Obj 3	Obj 2
Solution 3	Obj 2	Obj 1	Obj 3
Solution 4	Obj 2	Obj 3	Obj 1
Solution 5	Obj 3	Obj 1	Obj 2
Solution 6	Obj 3	Obj 2	Obj 1

Understanding Back Tracking

Using backtracking, we can solve the problem that has some constraints and we can find the solution based on these constraints.

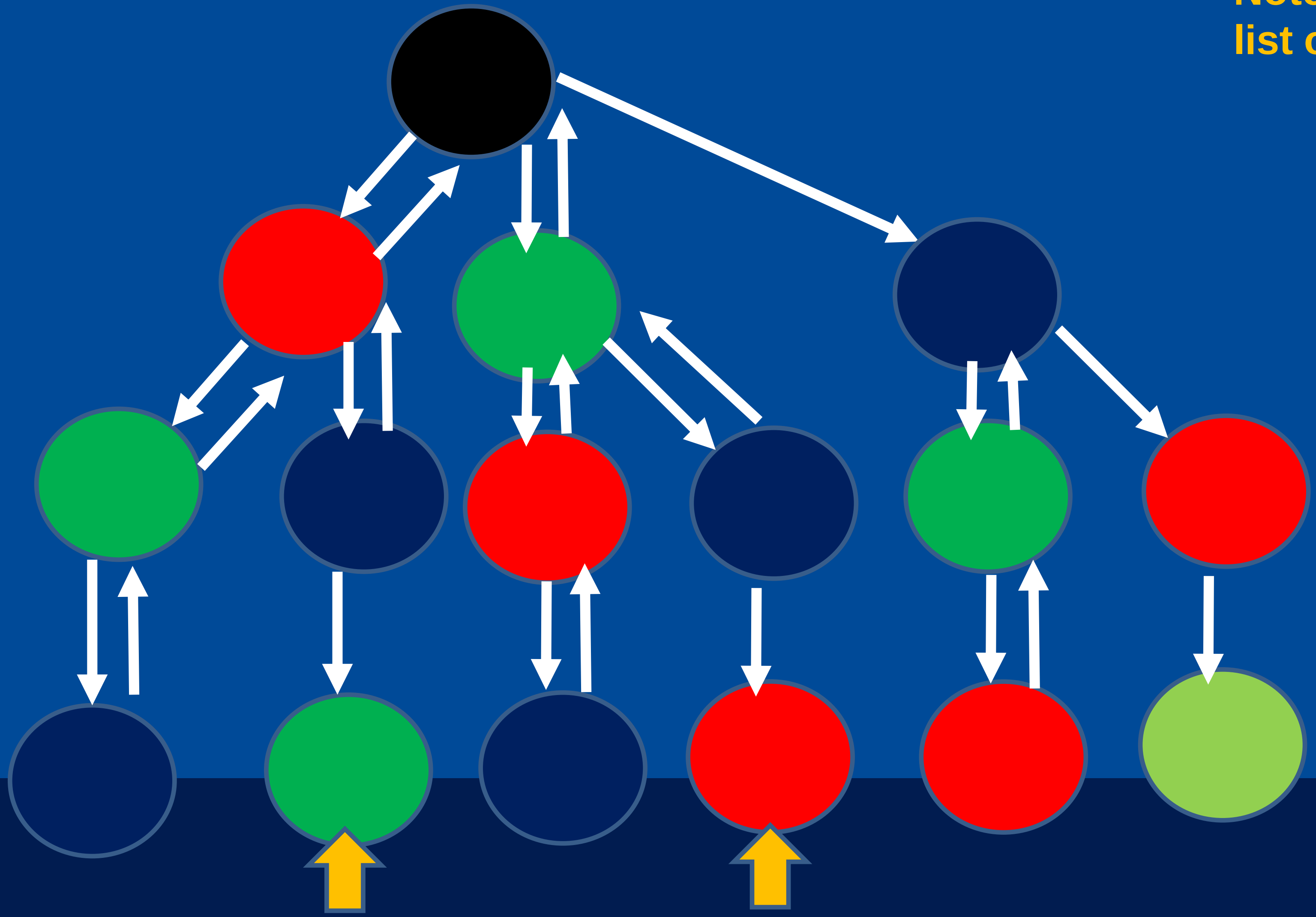


Suppose in this example; the **constraint is blue color object must not be in the middle (feasible function).**

Understanding Back Tracking

Suppose in this example; the **constraint is blue color object must not be in the middle** (bounding function).

Note: bounding function consists of list of conditions/constraint



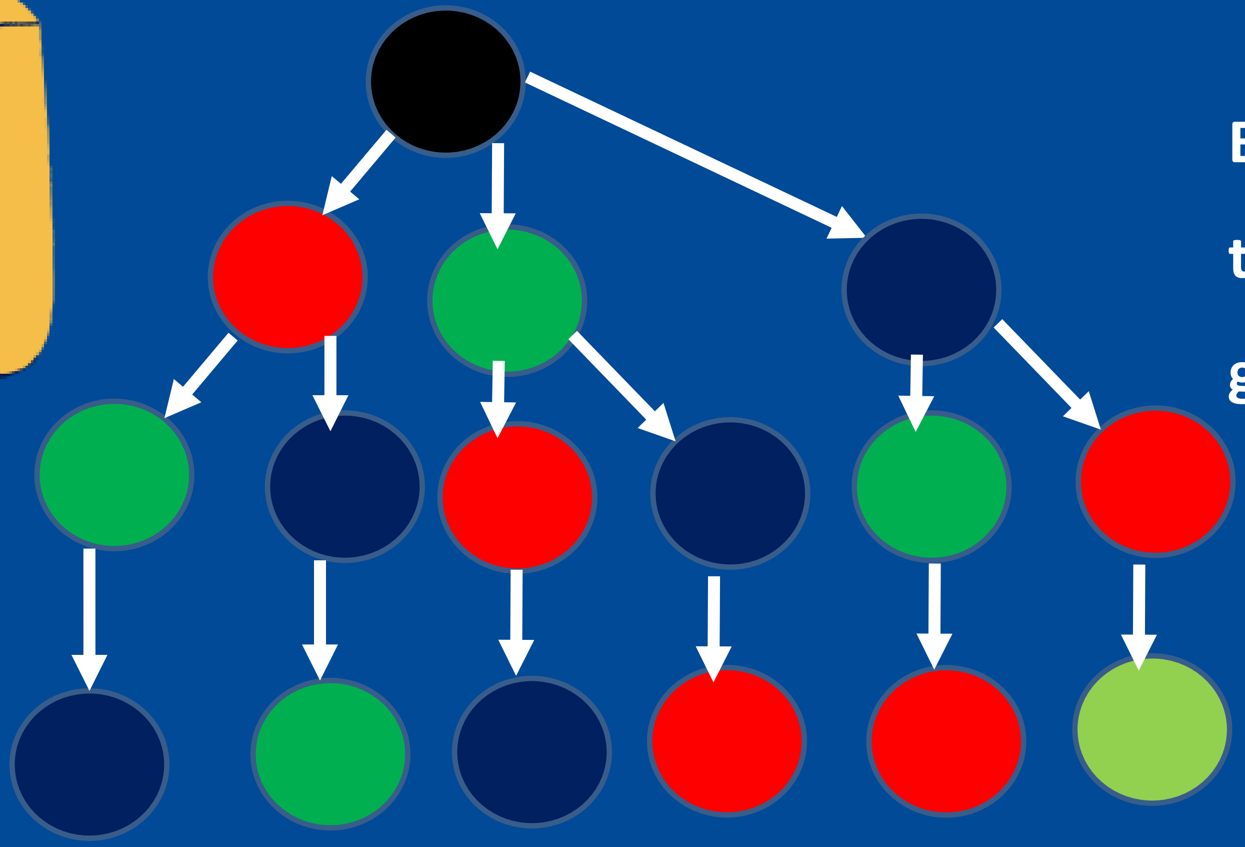
This solution will be not selected

This solution will be not selected

The above is the state space tree that does not have blue object in the middle of the solution.

Understanding Branch and Bound

It is similar to 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. The way to generate the tree is different.



Backtracking follows the DFS, whereas the branch n bound follows the BFS to generate the tree.

As it follows the BFS, so first all the nodes of the same level are added then we move to the next level.

Difference between Back Tracking and Branch & Bound

Parameter	Backtracking	Branch and Bound
Approach	Backtracking is used to find all possible solutions available to a problem. When it realizes that it has made a bad choice, it undoes the last choice by backing it up. It searches the state space tree until it has found a solution for the problem.	Branch-and-Bound is used to solve optimization problems. When it realizes that it already has a better optimal solution that the pre-solution leads to, it abandons that pre-solution. It completely searches the state space tree to get optimal solution.
Traversal	Backtracking traverses the state space tree by **DFS(Depth First Search)** manner.	Branch-and-Bound traverse the tree in any manner, **DFS or BFS.**
Function	Backtracking involves **bounding function to handle constrains**	Branch-and-Bound involves a **feasibility function** to handle constraints.
Problems	Backtracking is used for solving Decision Problem.	Branch-and-Bound is used for solving Optimization Problem.
Searching	In backtracking, the state space tree is searched until the solution is obtained.	In Branch-and-Bound as the optimum solution may be present any where in the state space tree, so the tree need to be searched completely.
Efficiency	Backtracking is more efficient.	Branch-and-Bound is less efficient.
Applications	Useful in solving N-Queen Problem, Sum of subset.	Useful in solving Travelling Salesman Problem.

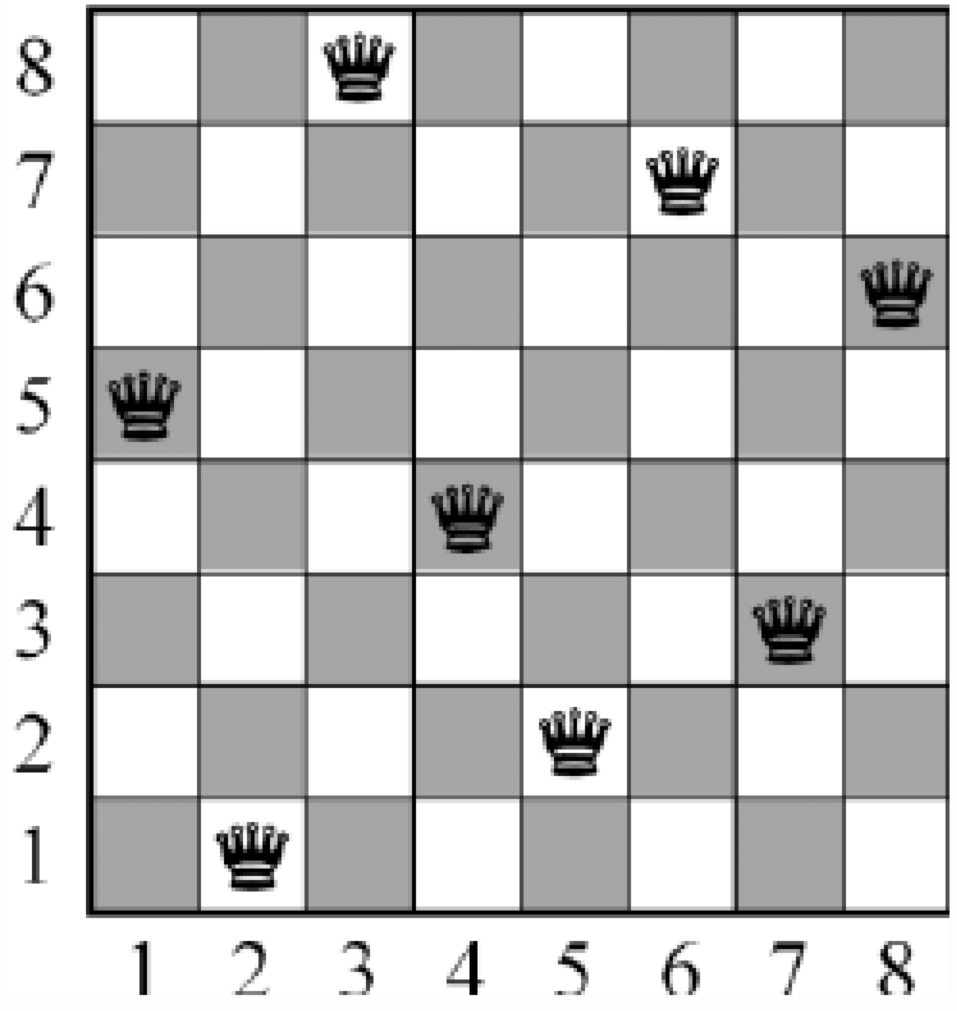
Application of Back Tracking and Branch & Bound

The problems that can be solved by using backtracking are:

- N Queens problem
- Sum of Subsets Problem

The problem that can be solved by using branch and bound is:

- Travelling Salesman Problem

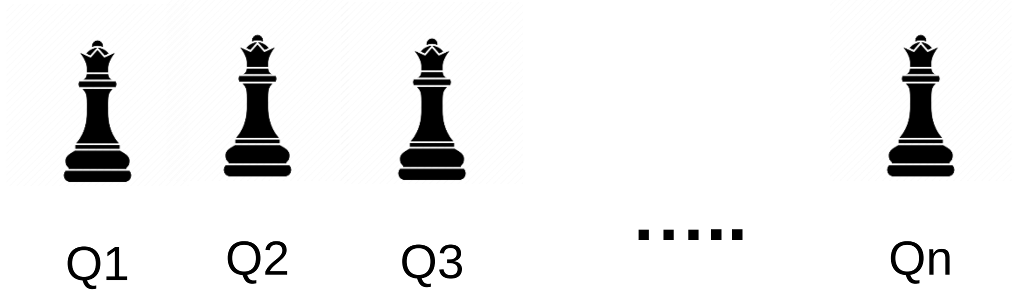


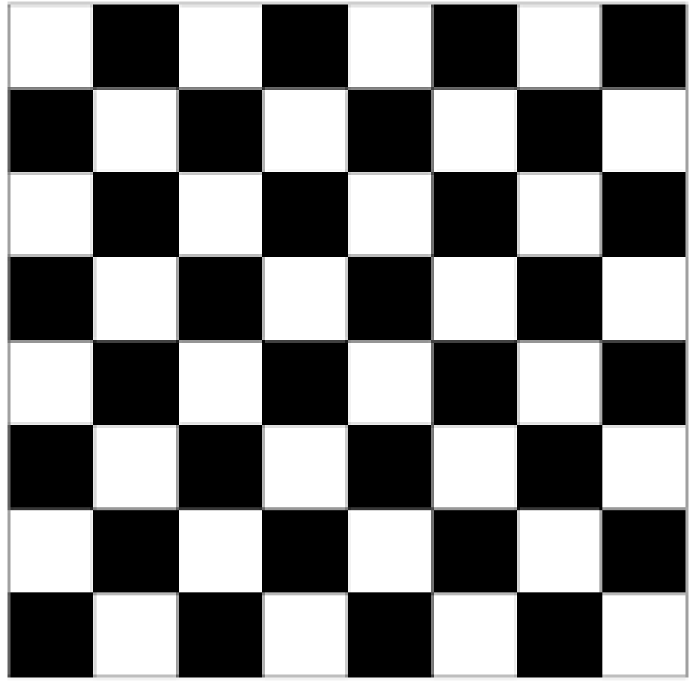
01. N Queens problem

Using Backtracking

N Queens problem

- The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. i.e., **no two queens are on the same row, column or diagonal**.

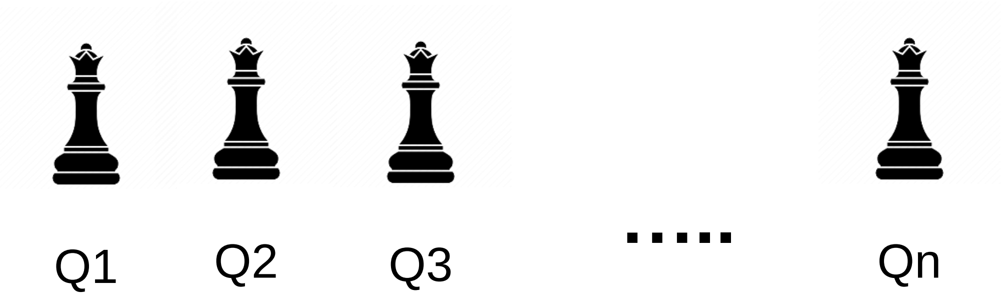


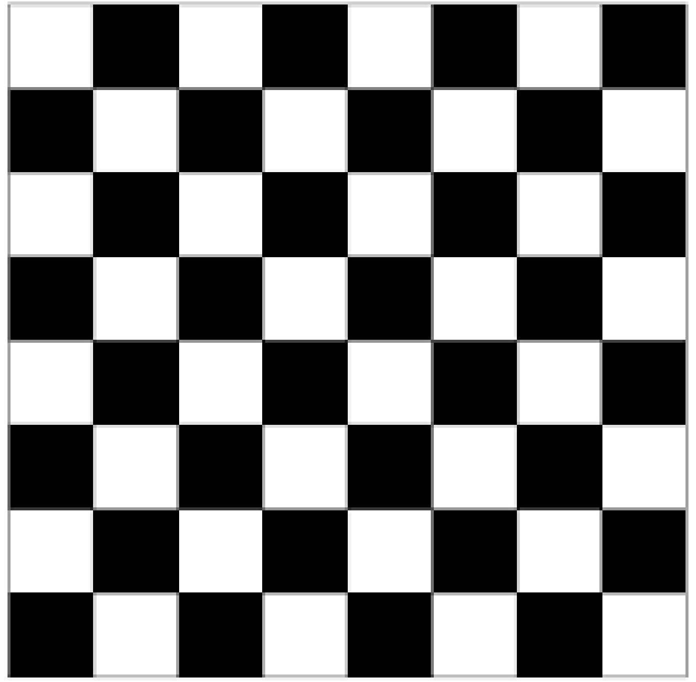


N Queens problem

Algorithm for N Queens Problem

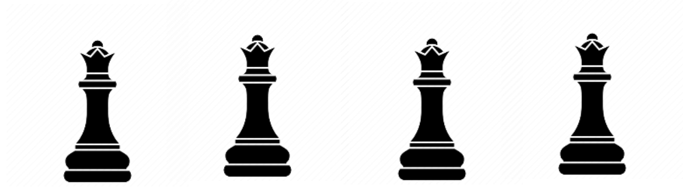
- Starts solving the problem by putting a queen on the first row of the board and tries to put the second queen on the second row in a way it wouldn’t conflict the first one.
  (If it satisfy the bounding functions (attacking condition), then kill the node and backtrack it)
- It continues putting the queens on the board row by row until it puts the last one on the nth row.
- If it couldn’t fill any tile on a row it would backtrack and change the position of the previous row’s queen.

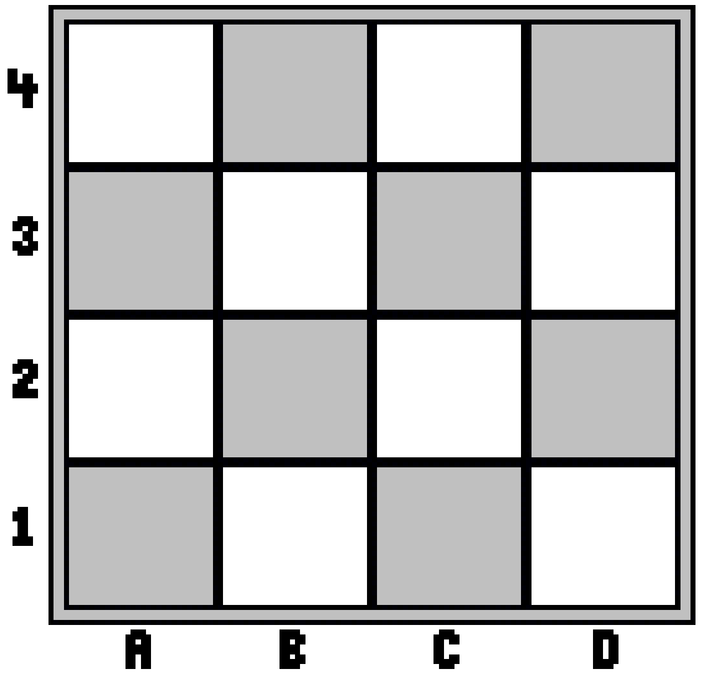




Example- 4 Queens problem

- The 4 Queen is the problem of placing 4 chess queens on an 4×4 chessboard so that no two queens attack each other. i.e., **no two queens are on the same row, column or diagonal**.





The expected output is a binary matrix which has 1s indicates for the blocks where queens are placed, 0 indicates not placed.

```
{ 0, 0, 0, 0}
{ 0, 0, 0, 0}
{ 0, 0, 0, 0}
{ 0, 0, 0, 0}
```

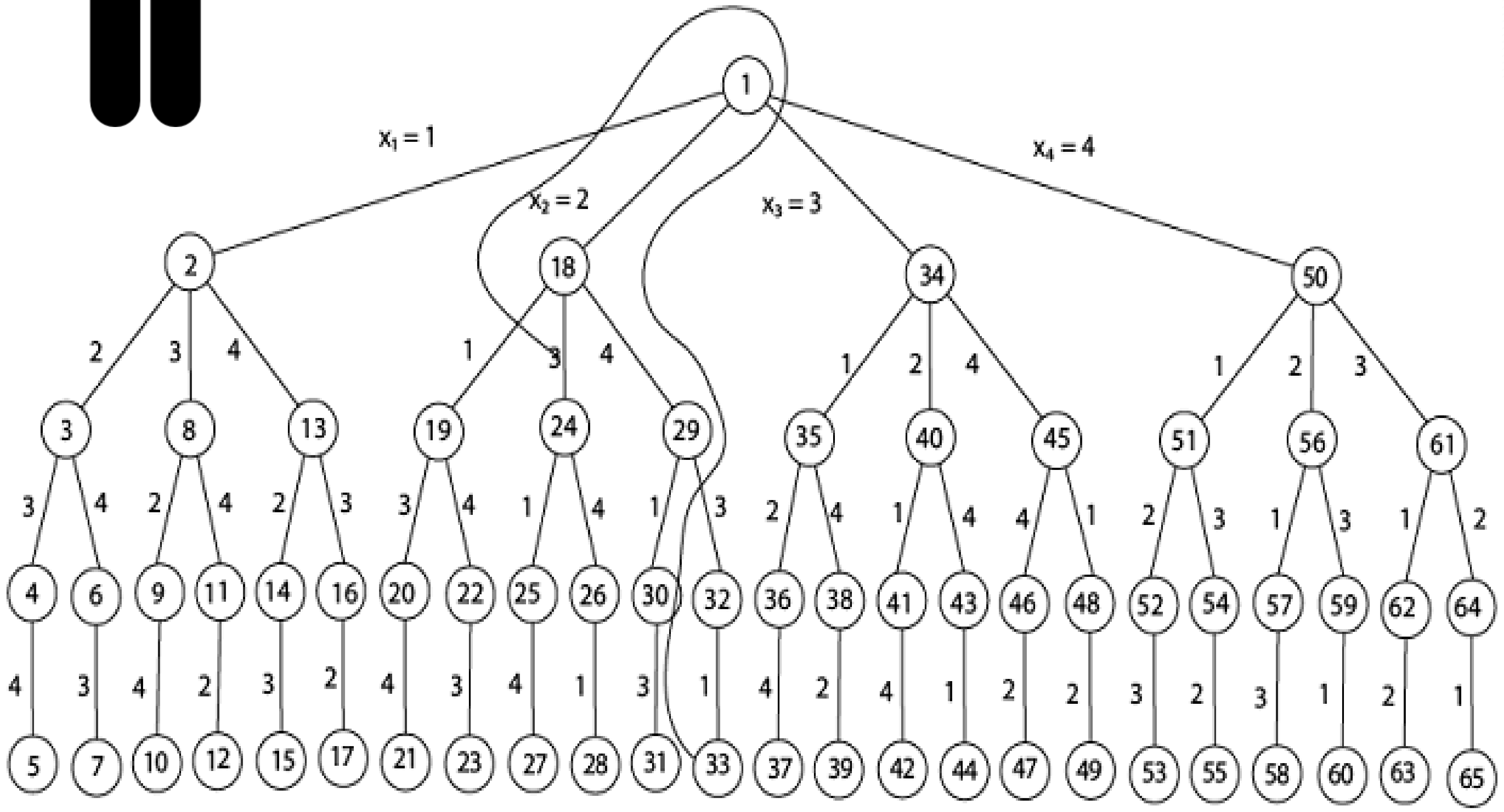
Note: Discussion on Blackboard

X=	0	0	0	0

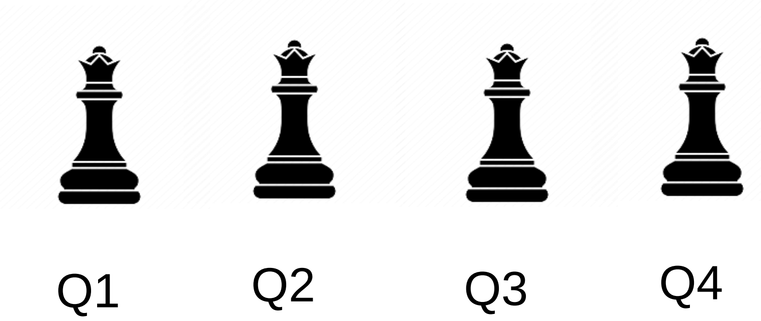
Example- 4 Queens problem

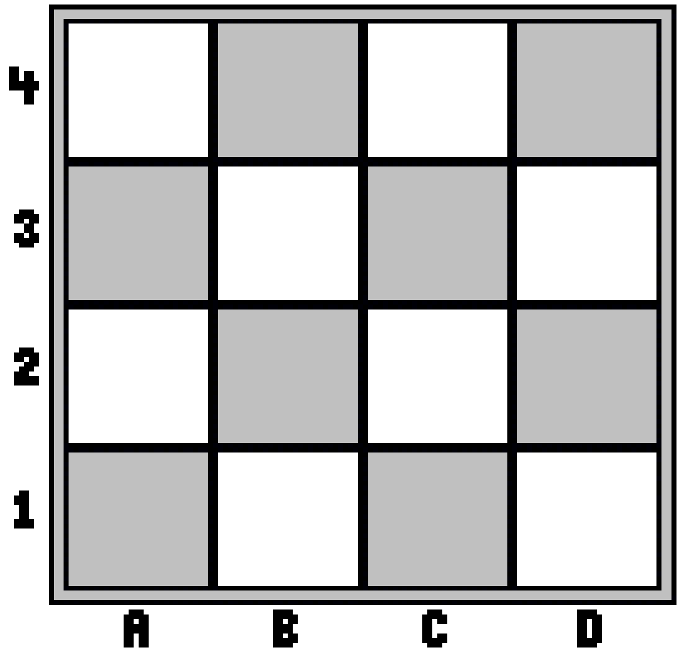
One way of thinking, use Naive Algorithm (brute force approach)

Generate all possible configurations of queens on board and print a configuration that satisfies the given constraints or list of conditions (bounding functions)



Possible State Space Tree (Without diagonal attack)



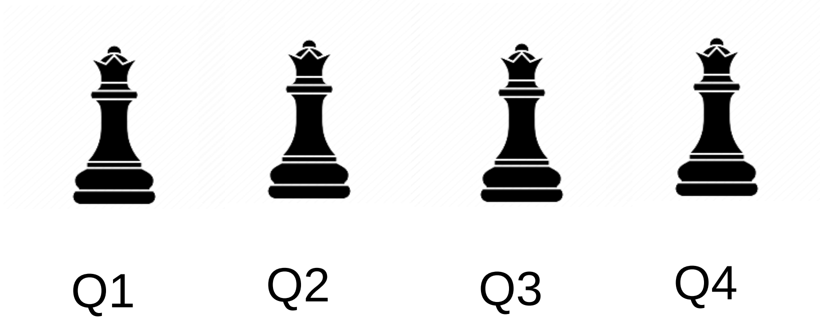


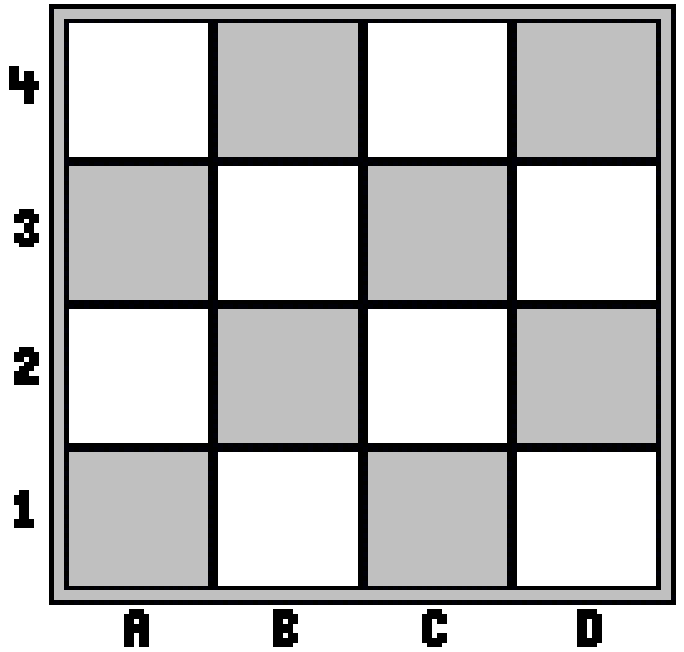
Note: Discussion on Blackboard

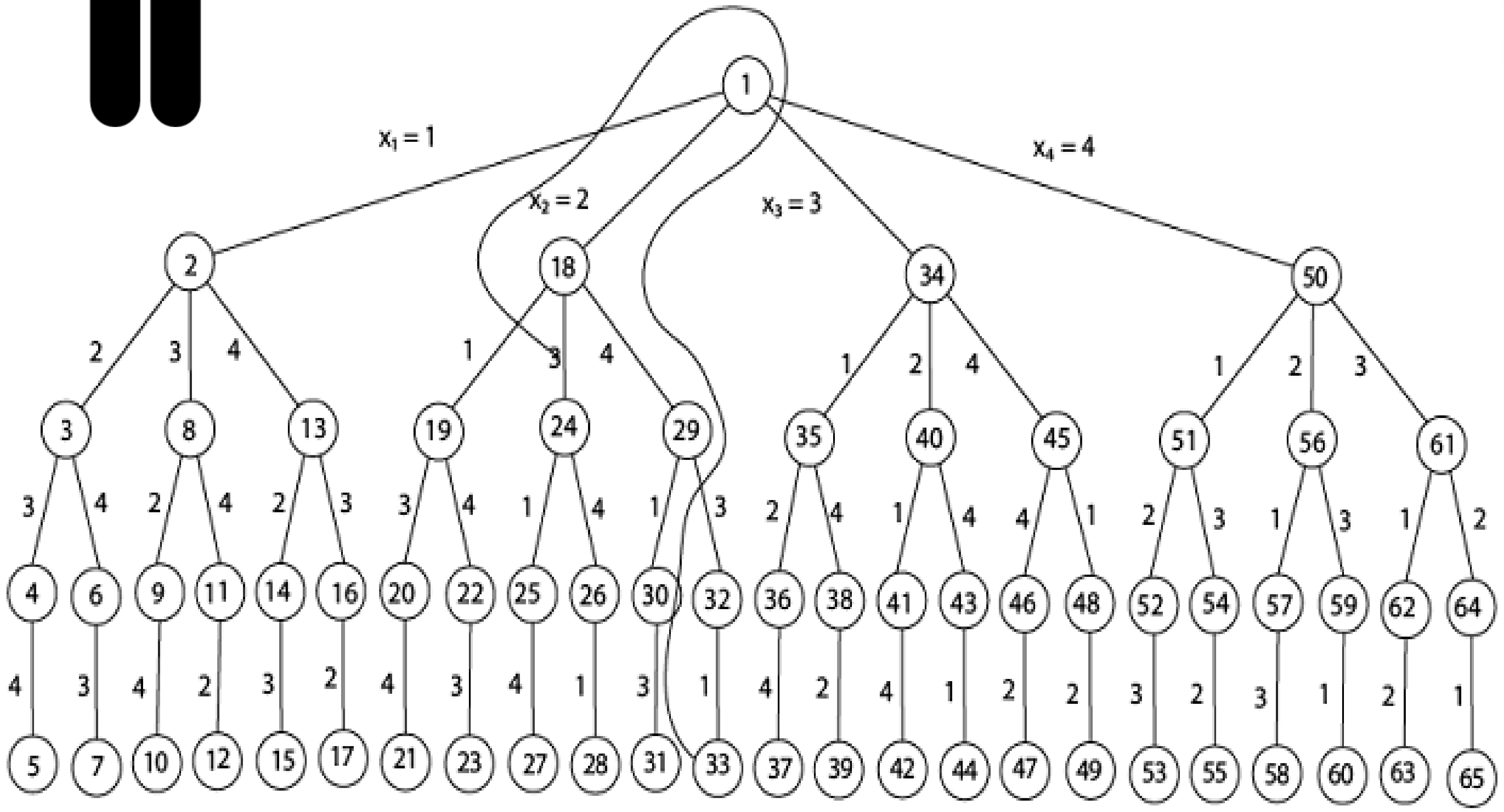
Example- 4 Queens problem

One way of thinking, use Naive Algorithm (brute force approach)

Generate all possible configurations of queens on board and print a configuration that satisfies the given constraints or list of conditions (bounding functions)







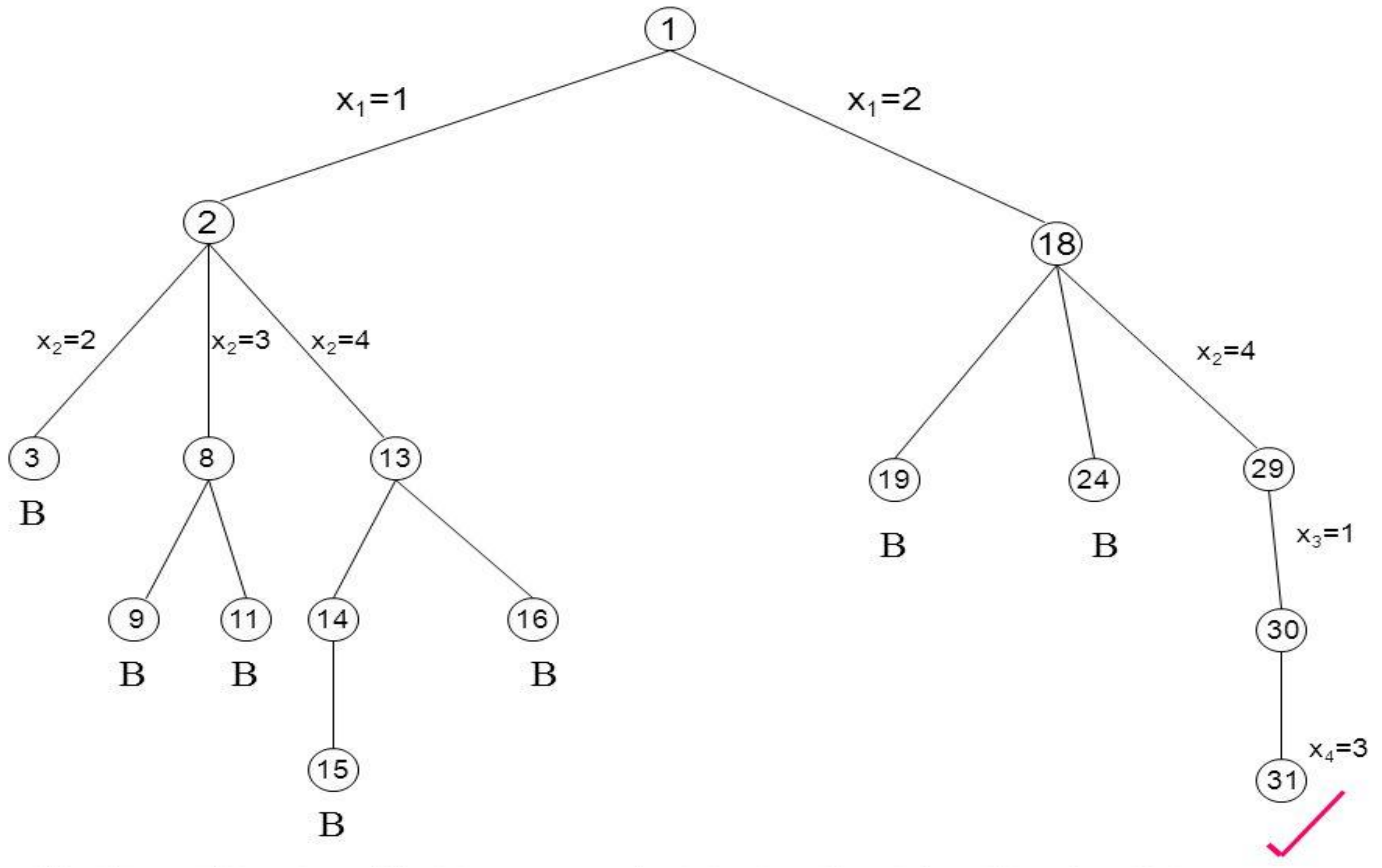
Possible State Space Tree (Without diagonal attack)

Total 65 Nodes are created

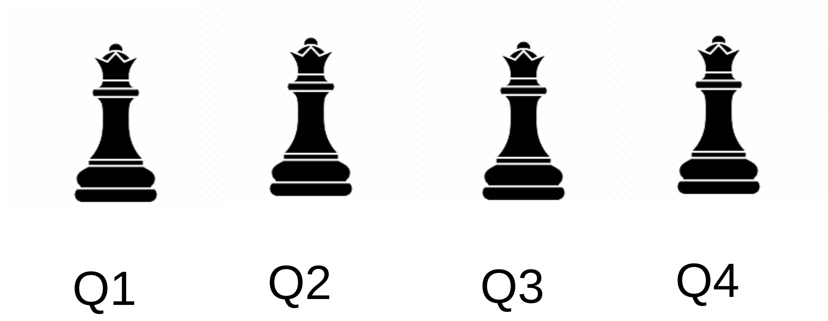
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

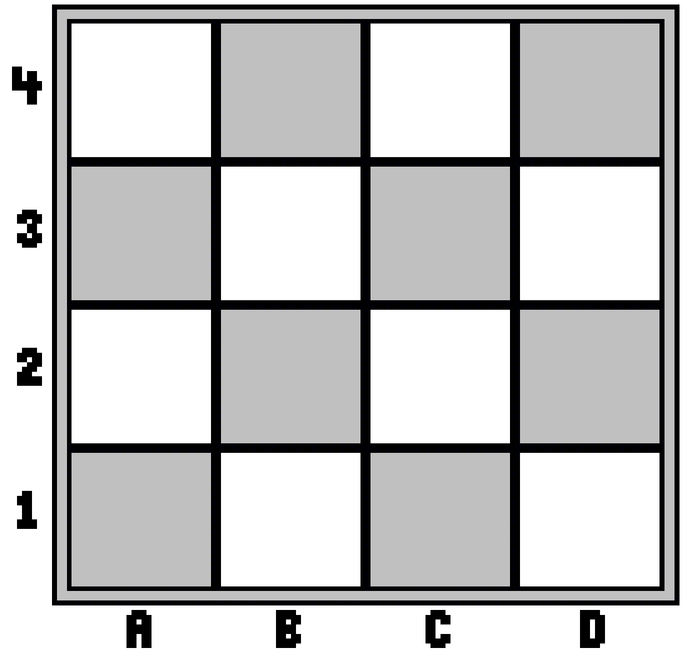
Solution of 4 Queens problem

Apply backtracking (bounding functions) and print a configuration that satisfies the given constraints or list of conditions

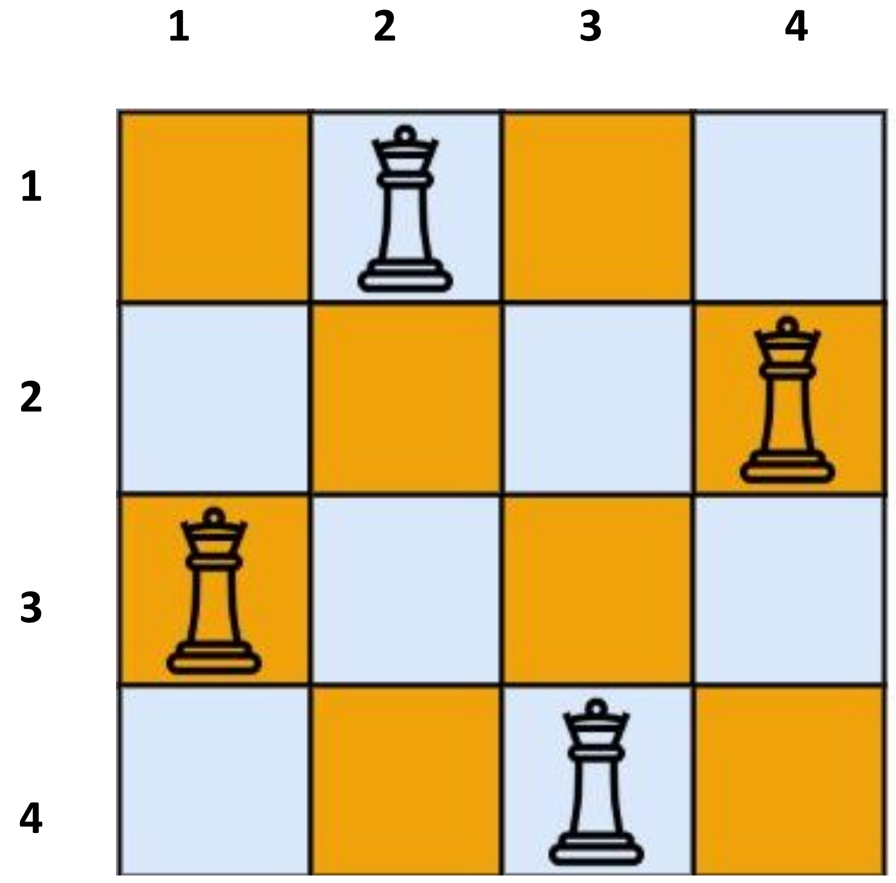


One Possible Final State Space Tree (With attack)





Solutions of 4 Queens problem



One Solution of 4 Queens Problem

The expected output is a binary matrix which has 1s for the blocks where queens are placed. The output matrix for above 4 queen solution.

$$X= \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & \{\ 0, & 1, & 0, & 0\} \\ 2 & \{\ 0, & 0, & 0, & 1\} \\ 3 & \{\ 1, & 0, & 0, & 0\} \\ 4 & \{\ 0, & 0, & 1, & 0\} \end{array}$$

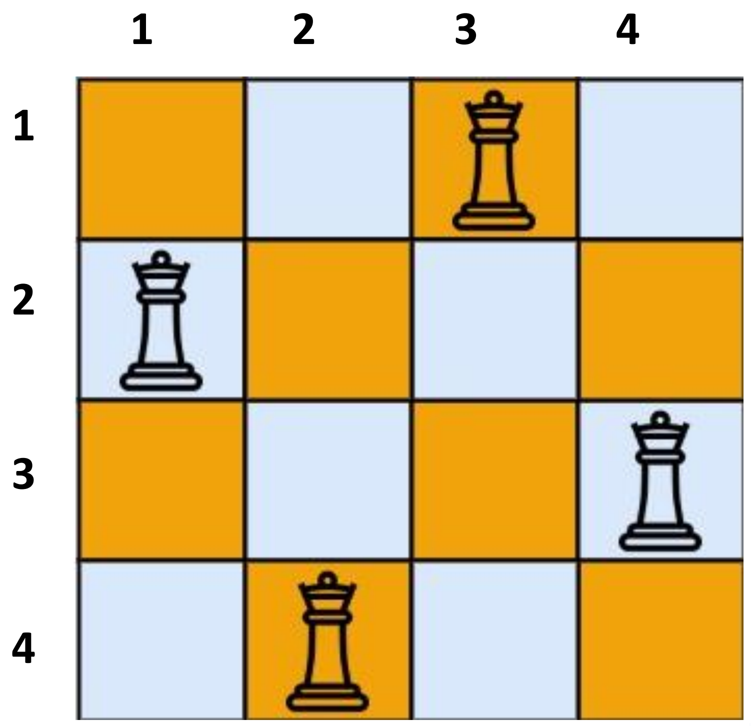
X_1=

2	4	1	3

Second Solution of 4 Queens Problem

X_2=

3	1	4	2



N Queens problem

Pseudo Code for N Queens Problem

```
Nqueens(k,n){
for i=1 to n{
   if(place(k,i)){
     x[k]=i
     if (k==n){
       for j=1 to n
         print x[i]}
     else
       Nqueens(k+1,n) }}}
place(k,i){
for j=1 to k
   if(x[j]==i || abs(x[j]-i ==abs(j-k))
       return false
   return true }
```

OR

1. Start in the leftmost column
2. If all queens are placed return true & Print Result
3. Try all rows in the current column

   Do following for every tried row.

   a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

   b) If placing the queen in [row, column] leads to a solution then return true.

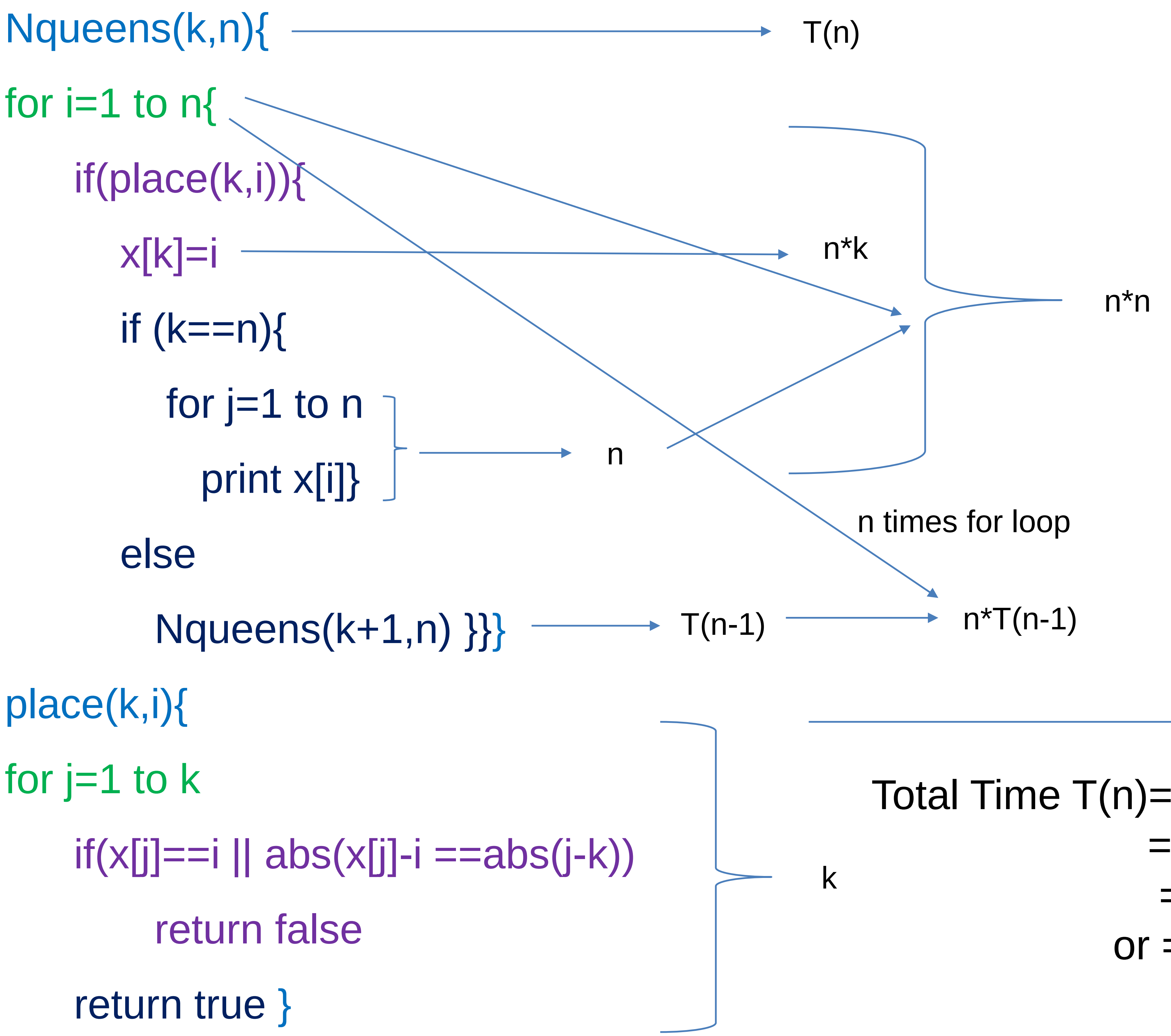
   c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.

4. If all rows have been tried and nothing worked, return false to trigger backtracking.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Time Complexity of N Queens problem

))



```
Nqueens(k,n){
for i=1 to n{
    if(place(k,i)){
        x[k]=i
        if (k==n){
            for j=1 to n
                print x[i]}
        else
            Nqueens(k+1,n) }}}
place(k,i){
for j=1 to k
    if(x[j]==i || abs(x[j]-i ==abs(j-k))
            return false
    return true }
```

Total Time T(n)= n*n +n*k + n*T(n-1)

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

$= O(n^n)$

or = O(n!)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

02. Sum of Subset Problems

Using Backtracking

Sum of Subset Problems

Suppose we are given *n* distinct positive numbers (usually called weights) and we desire to find all combinations of these numbers whose sum is *M*. This is called the ***sum of subsets* problem**.

The problem is given an A set of integers a1, a2,…., an upto n integers.

The question arises that is there a non-empty subset such that the sum of the subset is given as M integer?.

For example, the set is given as [5, 2, 1, 3, 9], and the sum of the subset is 9; the answer is **YES** as the sum of the subset [5, 3, 1] is equal to 9.

Example of Sum of Subset Problems

We have a set of 5 integers given: **N = 4, -2, 2, 3, 1**

We want to find out the subset whose sum is equal to 5.

There are many solutions to this problem.

```
Possible Subsets = { 0, 0, 0, 0, 1}
                 = { 0, 0, 0, 1, 0}
                 = { 0, 0, 0, 1, 1}
               .
               .
               .
total possible subsets=2⁶
```

The naïve approach, i.e., brute-force search generates all the possible subsets of the original array, i.e., there are 2^n possible states.

Solve the problem using Back Tracking Strategy

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Algorithm of Sum of Subset Problems

Step 1- Start with an empty set (draw State Space Tree)

Step 2 -Add the next element from the list to the set (add node in the tree in the DFS manner)

Step 3- If the subset is having sum M, then stop with that subset as solution.

Step 4- If the subset is not feasible or if we have reached the end of the set, then backtrack through the subset until we find the most suitable value.

Step 5- If the subset is feasible (sum of subset < M) then go to step 2.

Step 6- If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop without solution.

Example of Sum of Subset Problems

Problem Statement: Given a set of positive integers weight, and a value sum, determine that the sum of the subset of a given set is equal to the given sum.

w [1:5] = {2, 3, 5, 7, 10}, **Sum(w) = 14, n=5**

x =

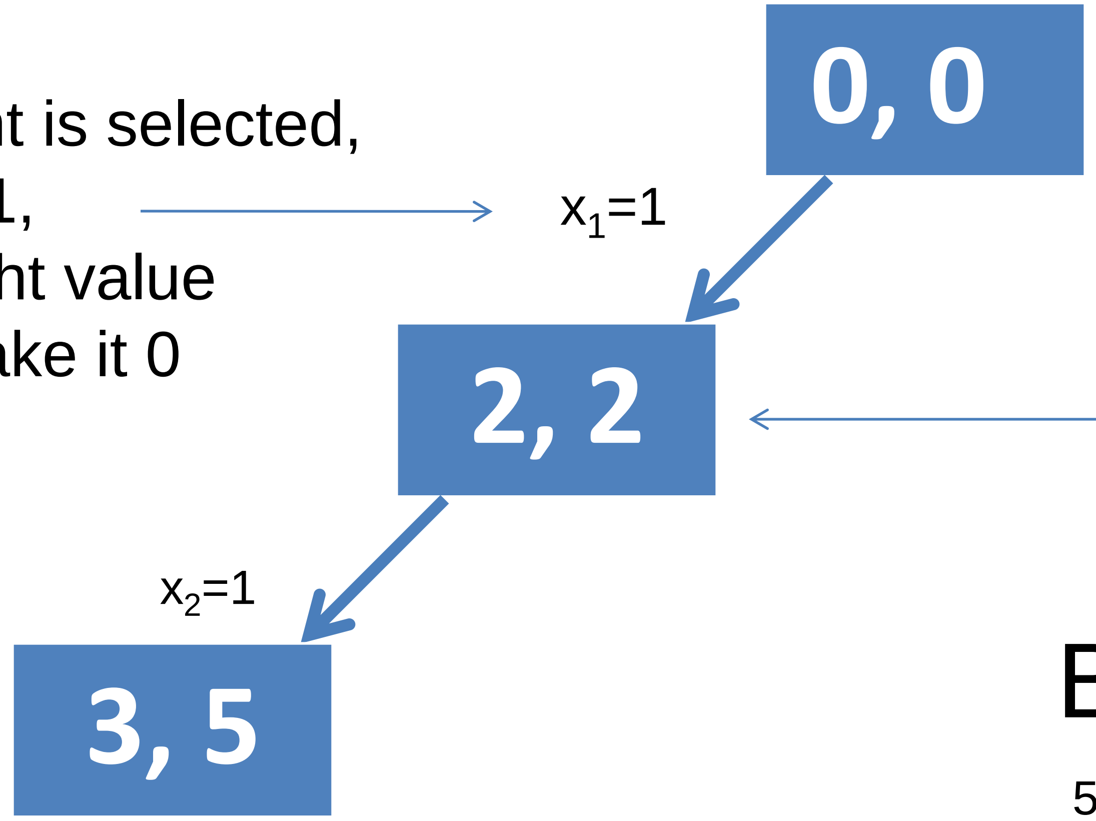
Fill 1 if weight is selected otherwise fill 0 if weight is not selected

Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10}, Sum(w) = 14, n=5

Step-1

First, we create a State Space Tree in DFS order, check the bounding function condition, if condition true, keep visiting it otherwise kill the node.



No weight selected, initial sum=0

If given weight is selected,
then make it 1,
sum the weight value
Otherwise make it 0

First weight w_1 is selected, Total sum=2

x =

1	1			

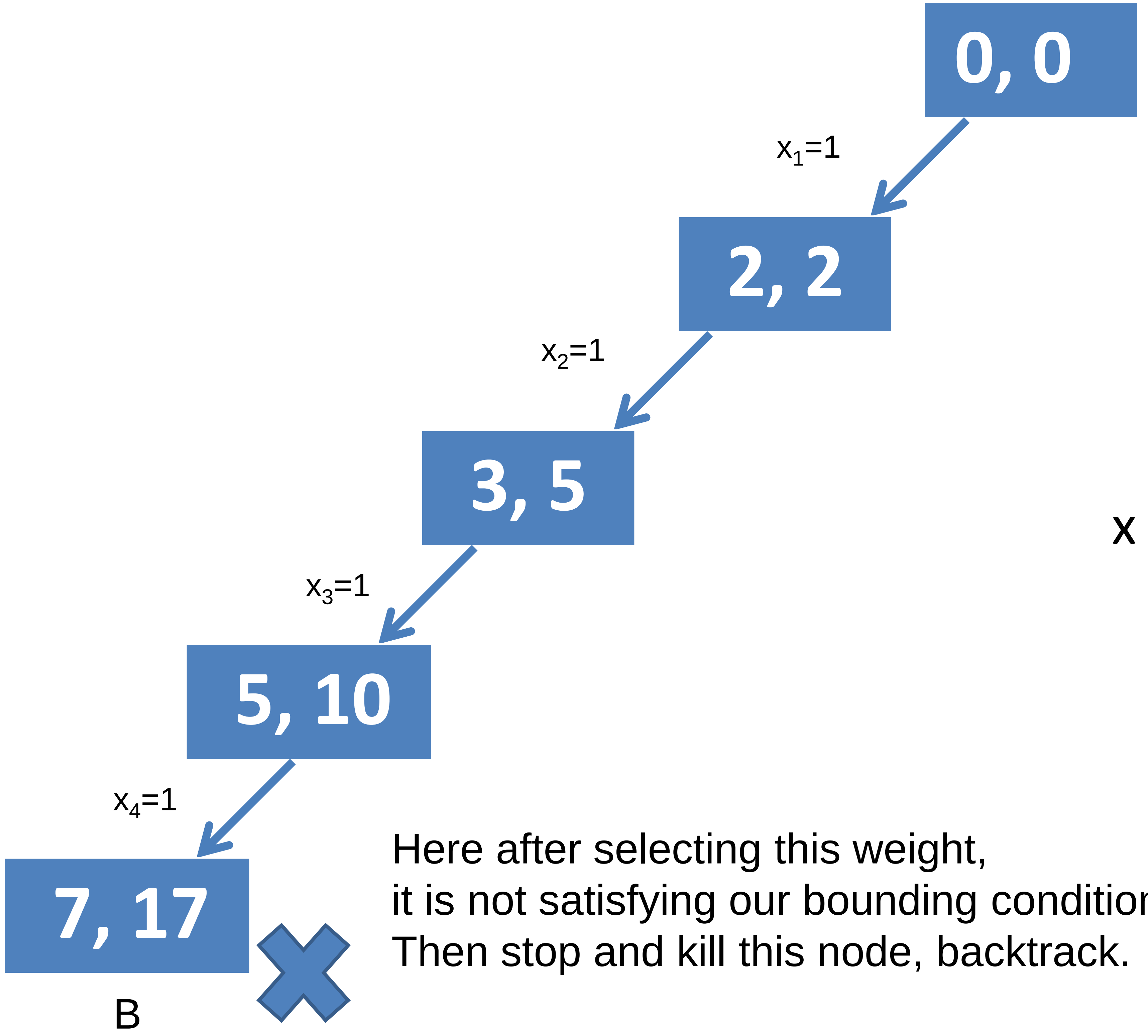
Bounding Condition

$$\sum_{i=1}^{5} w_i x_i + w_i <= \text{Sum (w)}$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10}, Sum(w) = 14, n=5



0, 0

$x_1=1$

2, 2

$x_2=1$

3, 5

$x_3=1$

5, 10

$x_4=1$

7, 17

B

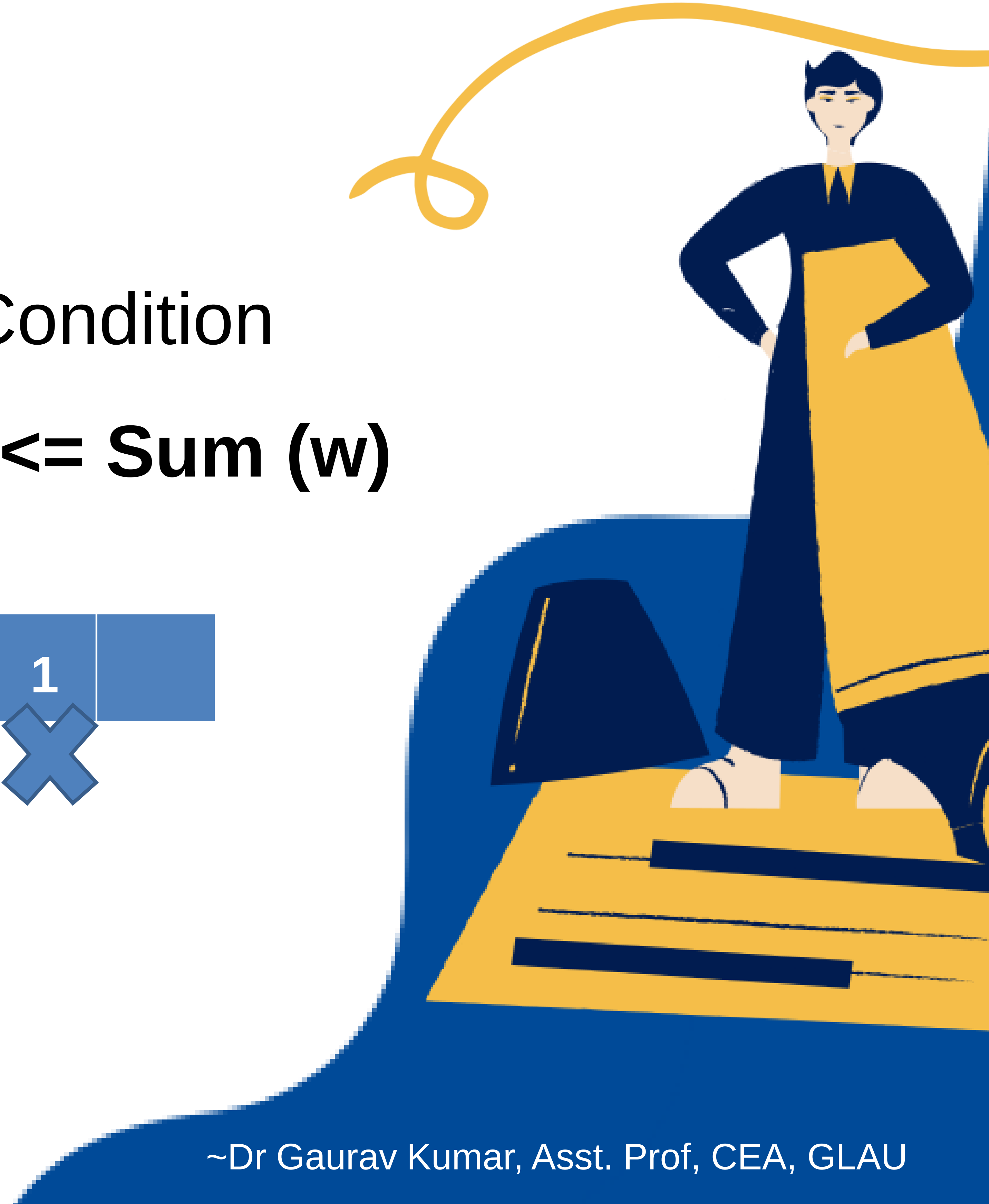
Here after selecting this weight,
it is not satisfying our bounding condition,
Then stop and kill this node, backtrack.

Bounding Condition

$$\sum_{i=1}^{5} w_i x_i + w_i <= \text{Sum (w)}$$

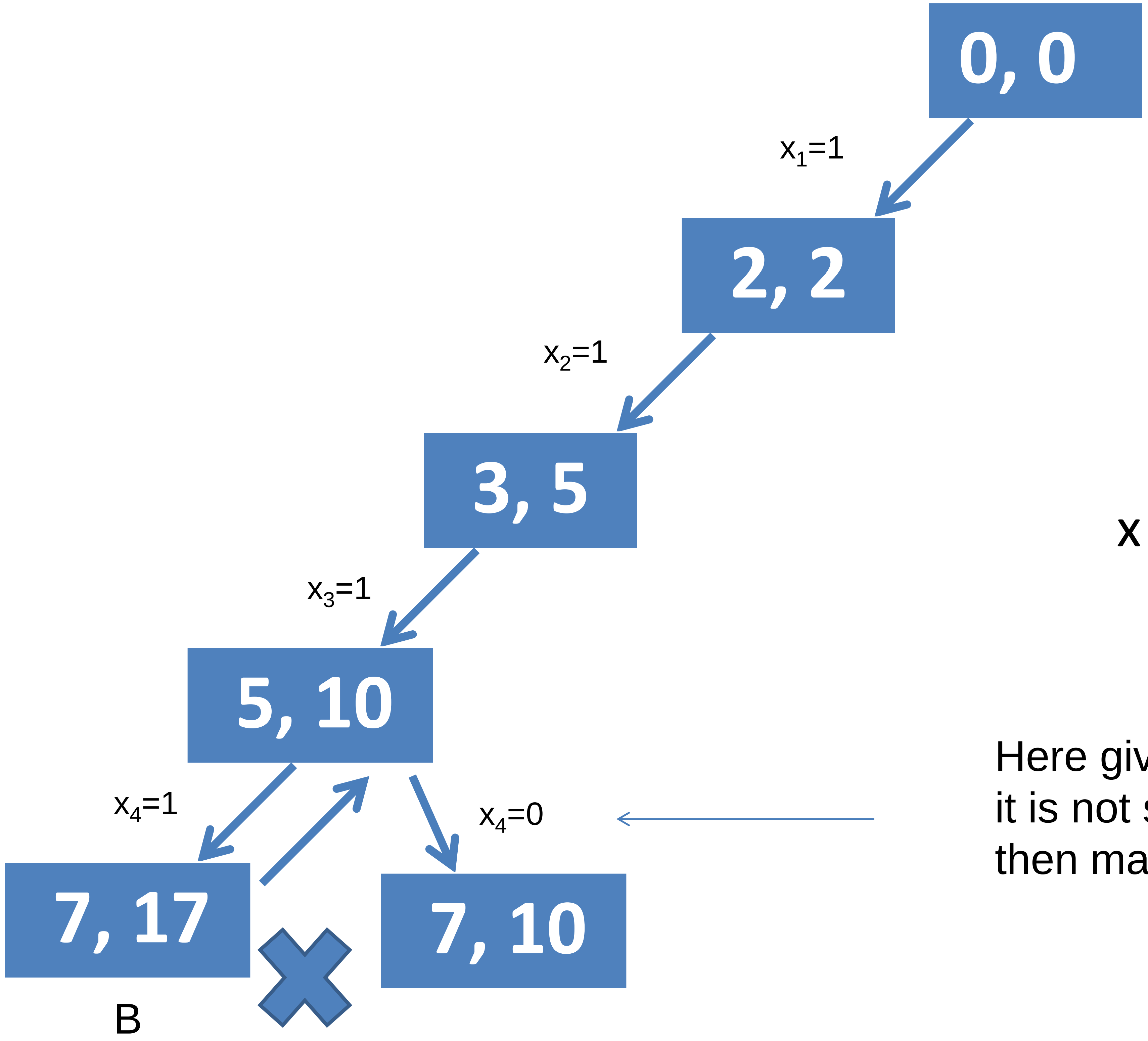
x =

1	1	1	1 ✗	



Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10}, Sum(w) = 14, n=5



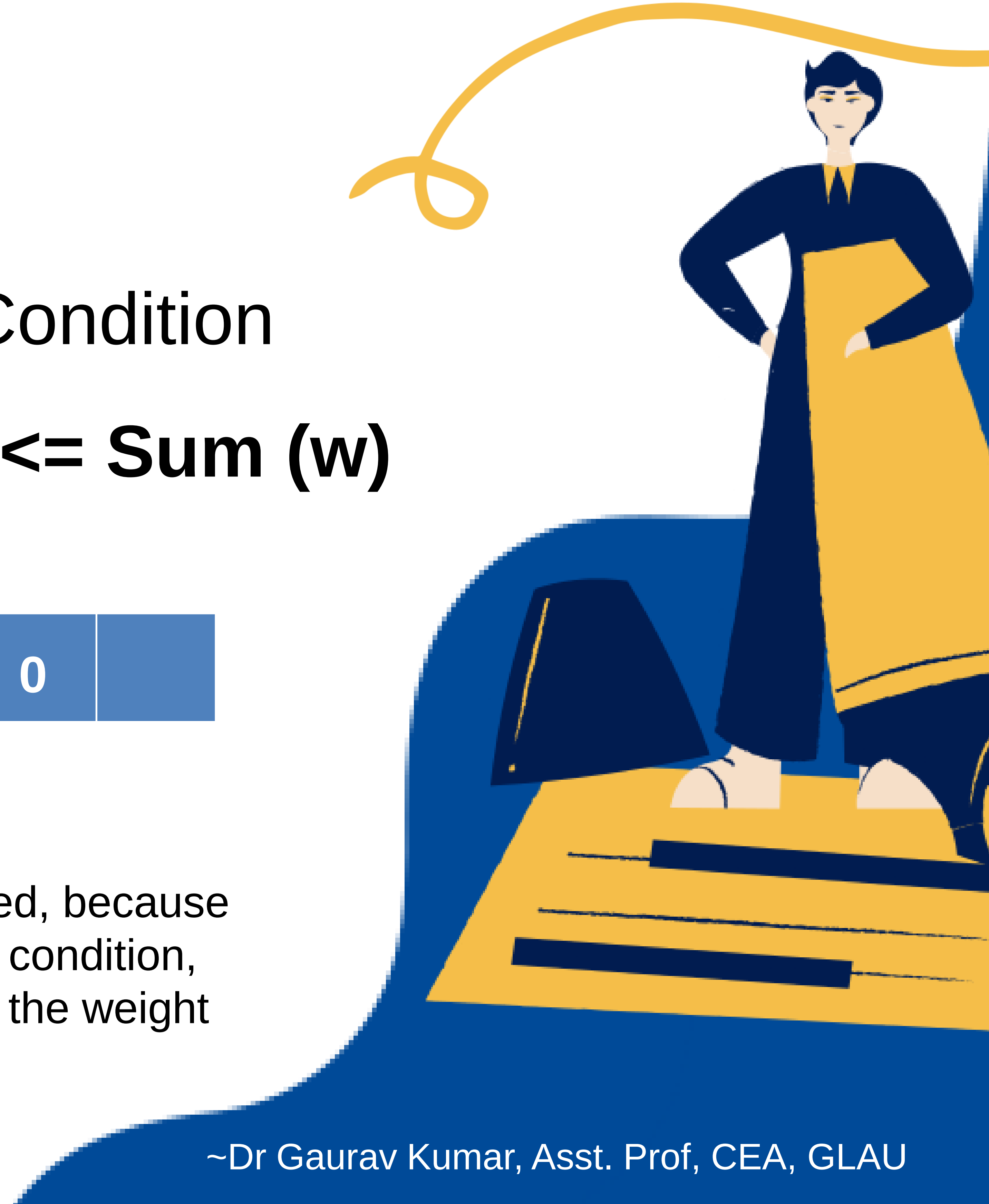
Bounding Condition

$$\sum_{i=1}^{5} \mathbf{w_i x_i} + \mathbf{w_i} <= \textbf{Sum (w)}$$

x =

1	1	1	0	

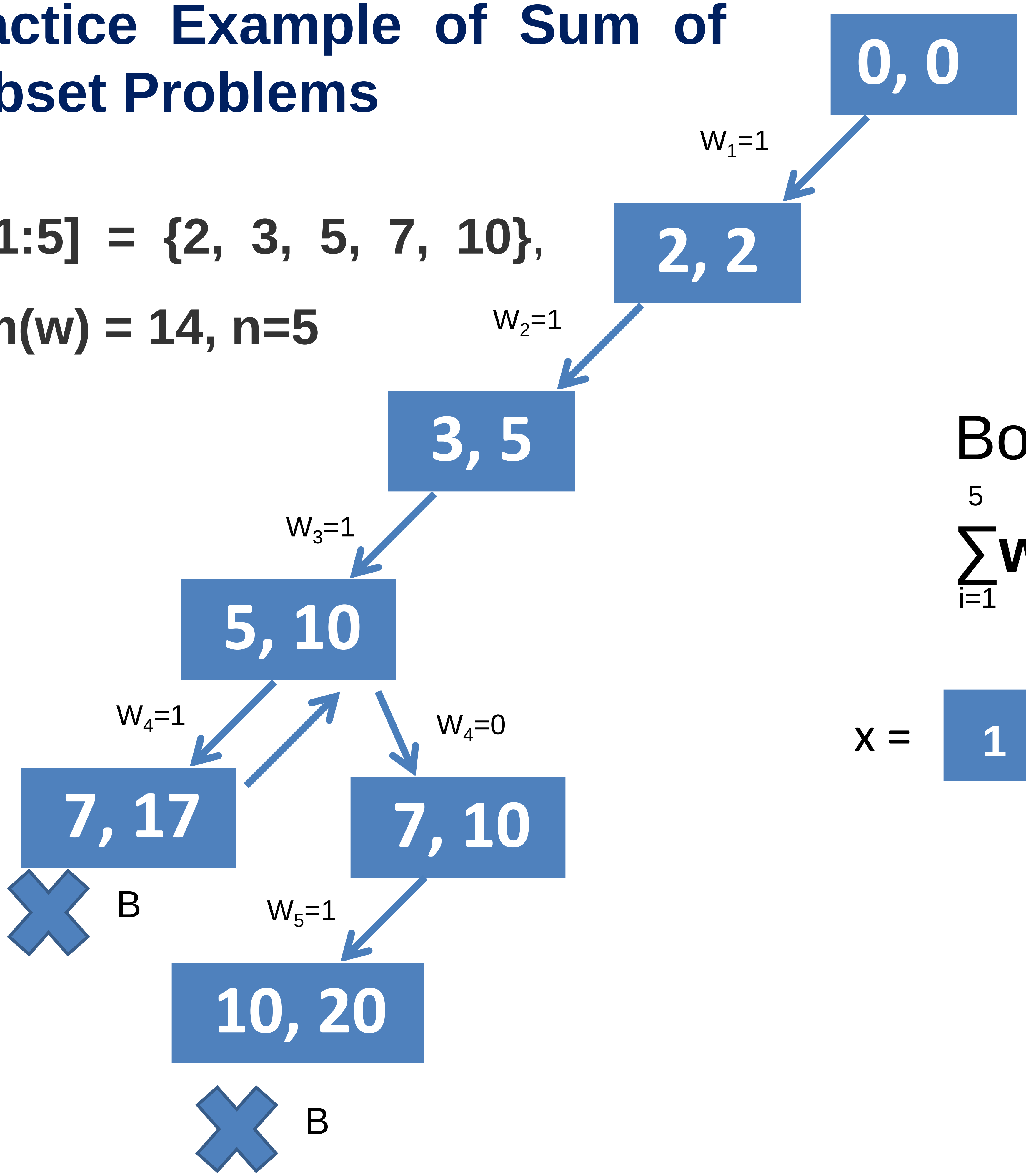
Here given weight is not selected, because it is not satisfying our bounding condition, then make it 0, and do not sum the weight



Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10},

Sum(w) = 14, n=5

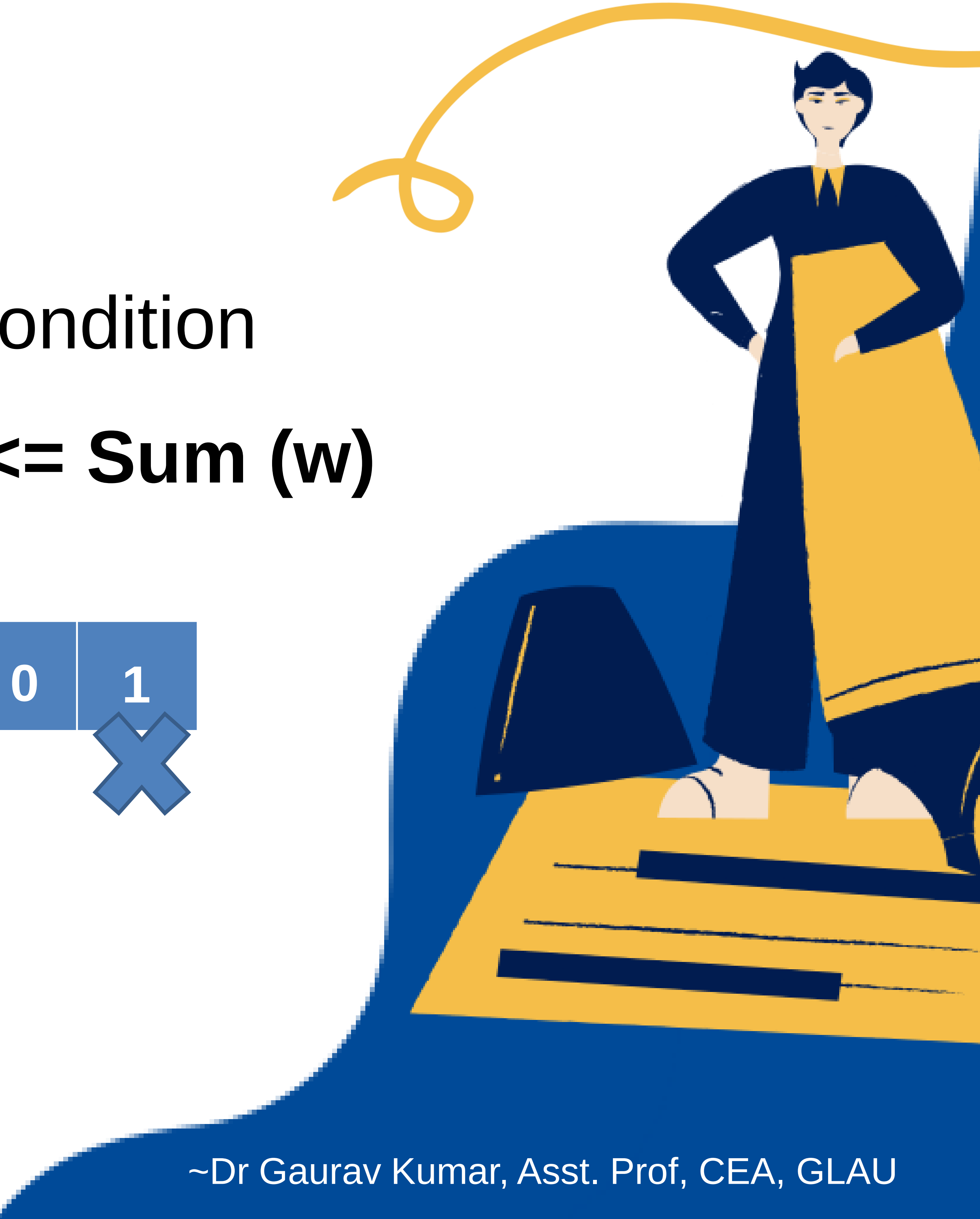


Bounding Condition

$$\sum_{i=1}^{5} w_i x_i + w_i <= \text{Sum (w)}$$

x =

1	1	1	0	~~1~~

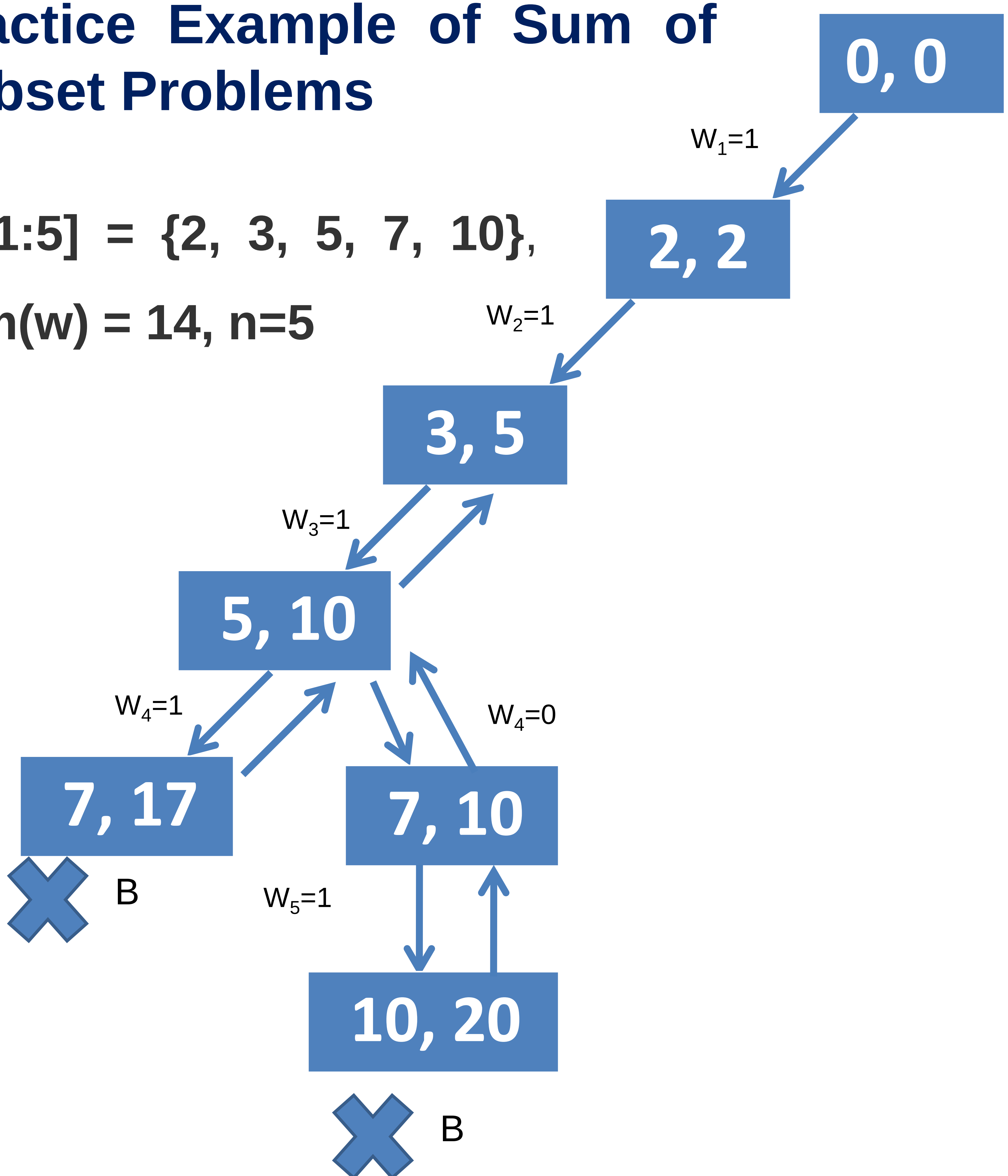


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10},

Sum(w) = 14, n=5



Bounding Condition

$$\sum_{i=1}^{5} w_i x_i + w_i <= \text{Sum } (w)$$

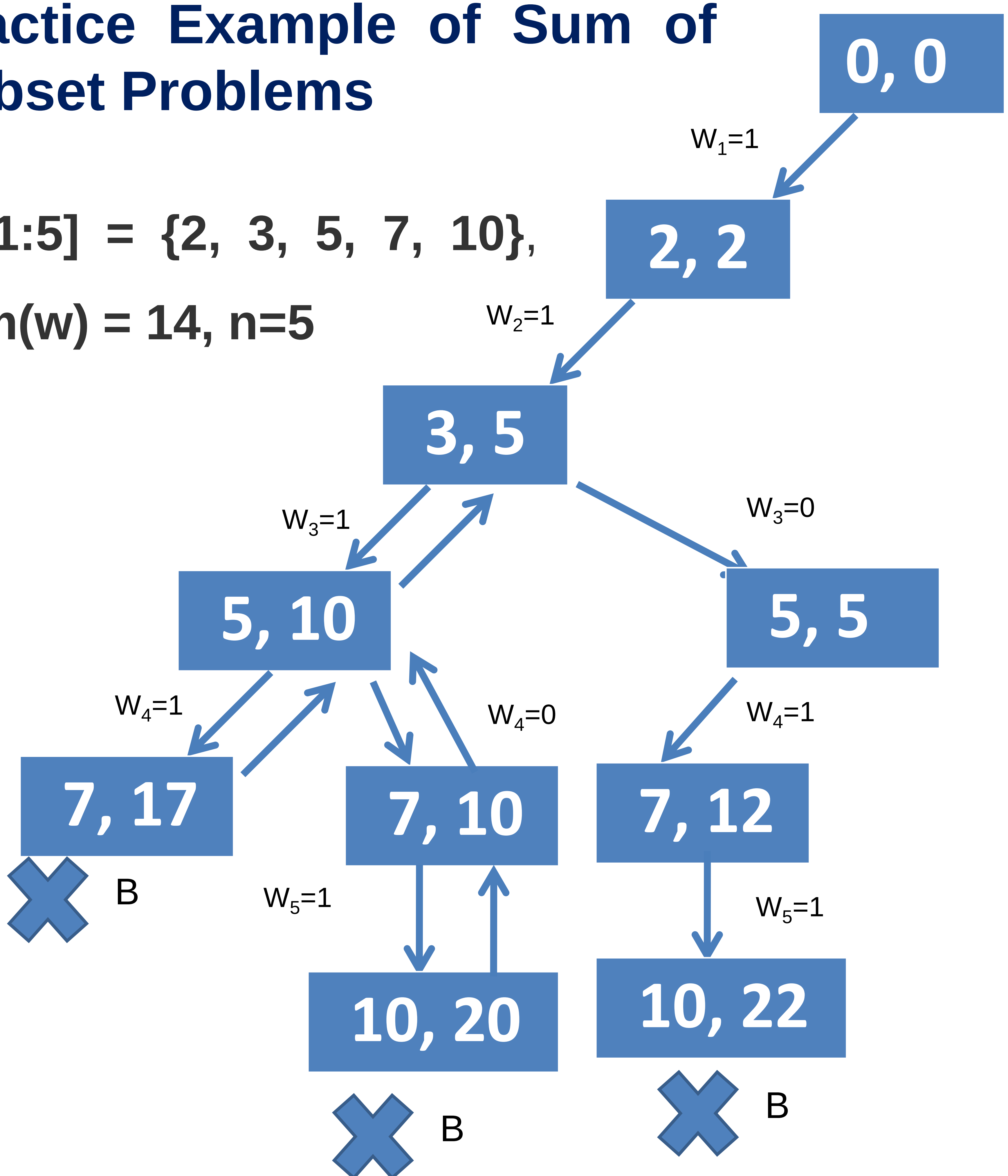
x =	1	1	1	0	1

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10},

Sum(w) = 14, n=5



0, 0

W_1=1

2, 2

W_2=1

3, 5

W_3=1

W_3=0

5, 10

5, 5

W_4=1

W_4=0

W_4=1

7, 17

B

7, 10

7, 12

W_5=1

W_5=1

10, 20

B

10, 22

B

Bounding Condition

$$\sum_{i=1}^{5} w_i x_i + w_i <= \text{Sum (w)}$$

x =

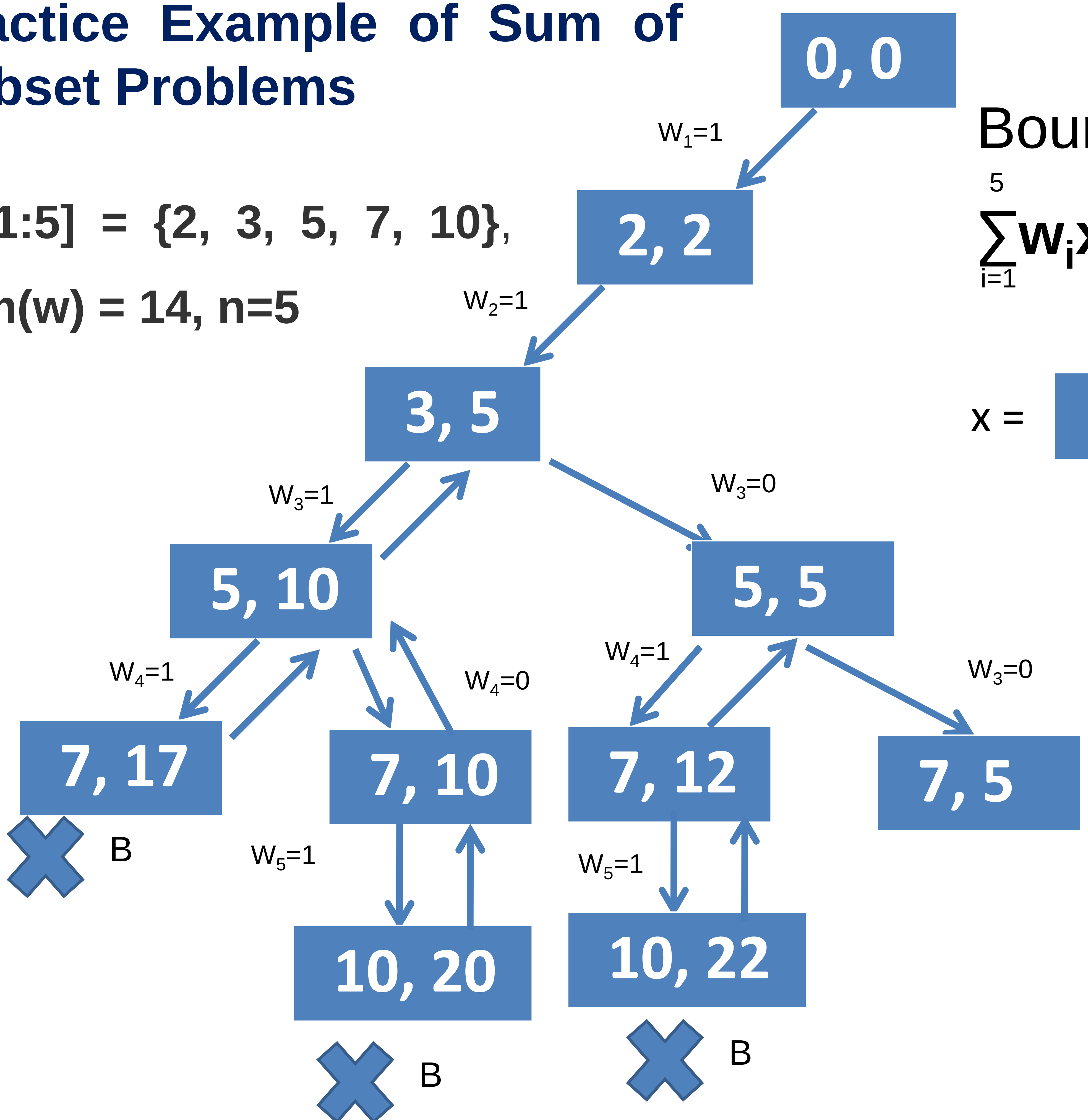
1	1	0	1	1 ✗

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10},

Sum(w) = 14, n=5


0, 0

$W_1=1$

2, 2

$W_2=1$

3, 5

$W_3=1$

$W_3=0$

5, 10

5, 5

$W_4=1$

$W_4=0$

$W_4=1$

$W_3=0$

7, 17

7, 10

7, 12

7, 5

B

$W_5=1$

$W_5=1$

10, 20

10, 22

B

B

Bounding Condition

$$\sum_{i=1}^{5} w_i x_i + w_i <= \text{Sum (w)}$$

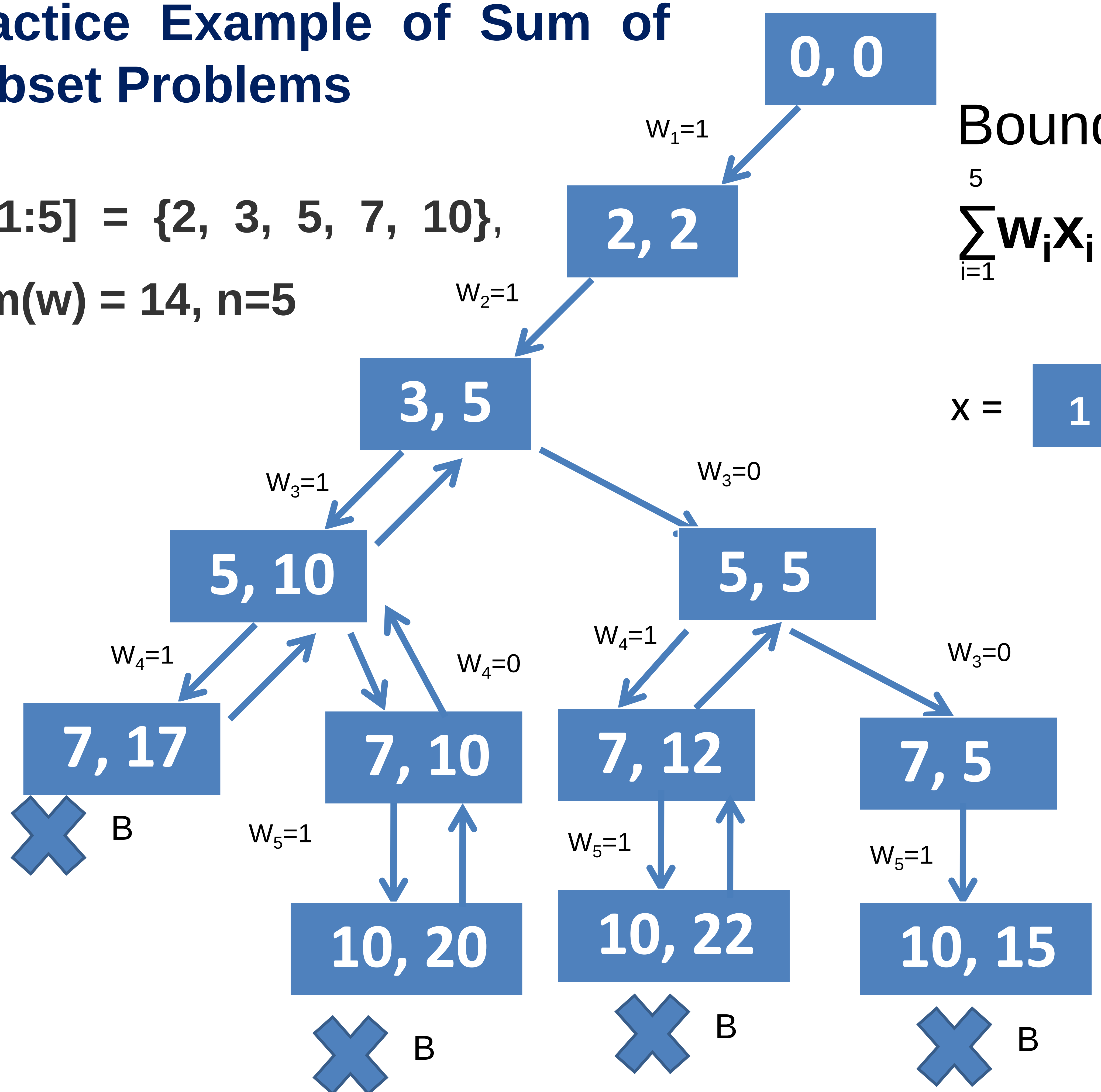
x =

1	1	0	1	1
				✗

Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10},

Sum(w) = 14, n=5



Bounding Condition

$$\sum_{i=1}^{5} w_i x_i + w_i <= \text{Sum}(w)$$

x =

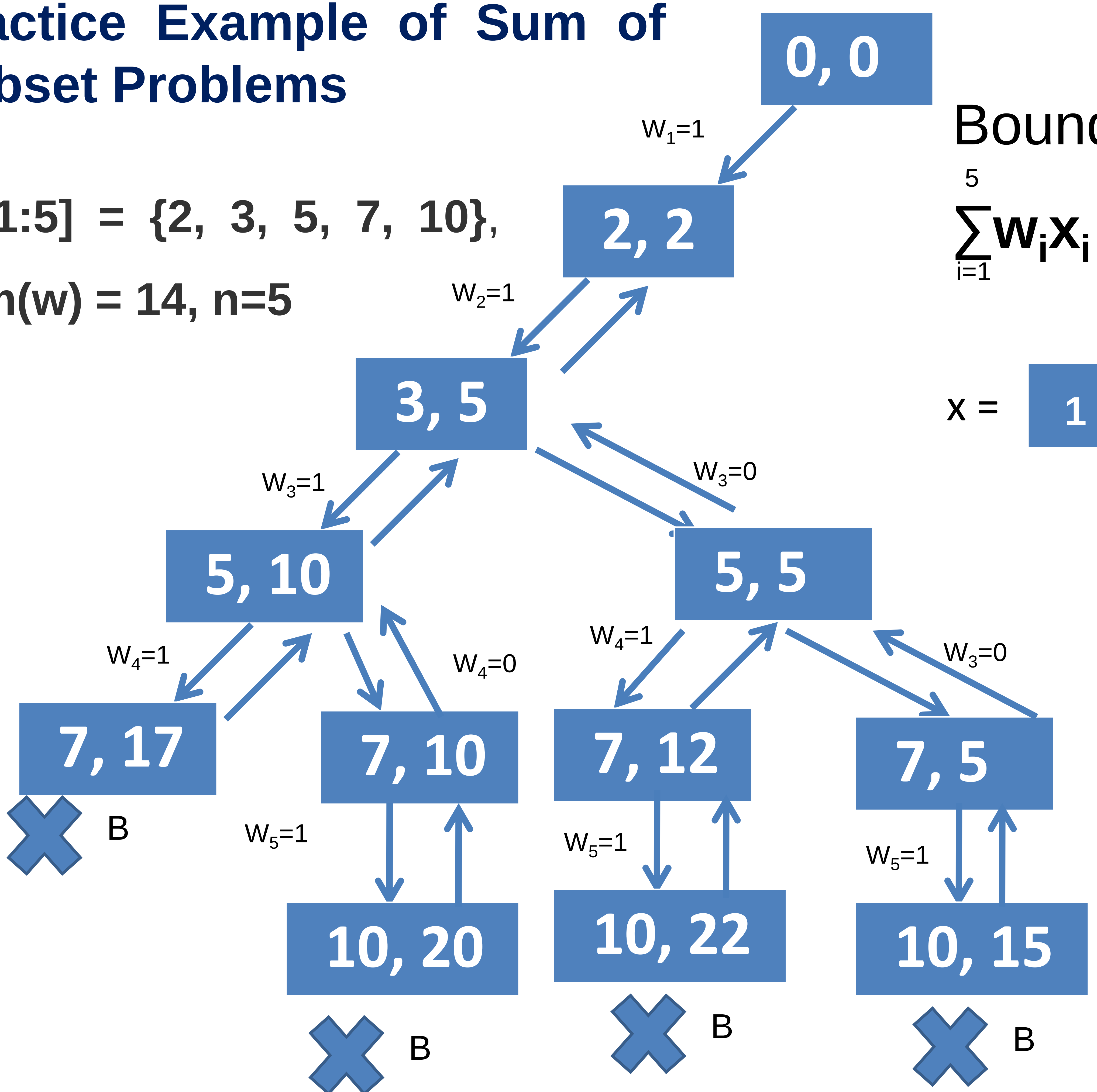
1	1	0	0	1

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10},

Sum(w) = 14, n=5



Bounding Condition

$$\sum_{i=1}^{5} w_i x_i + w_i <= \text{Sum (w)}$$

x =	1	1	0	0	1

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Sum of Subset Problems

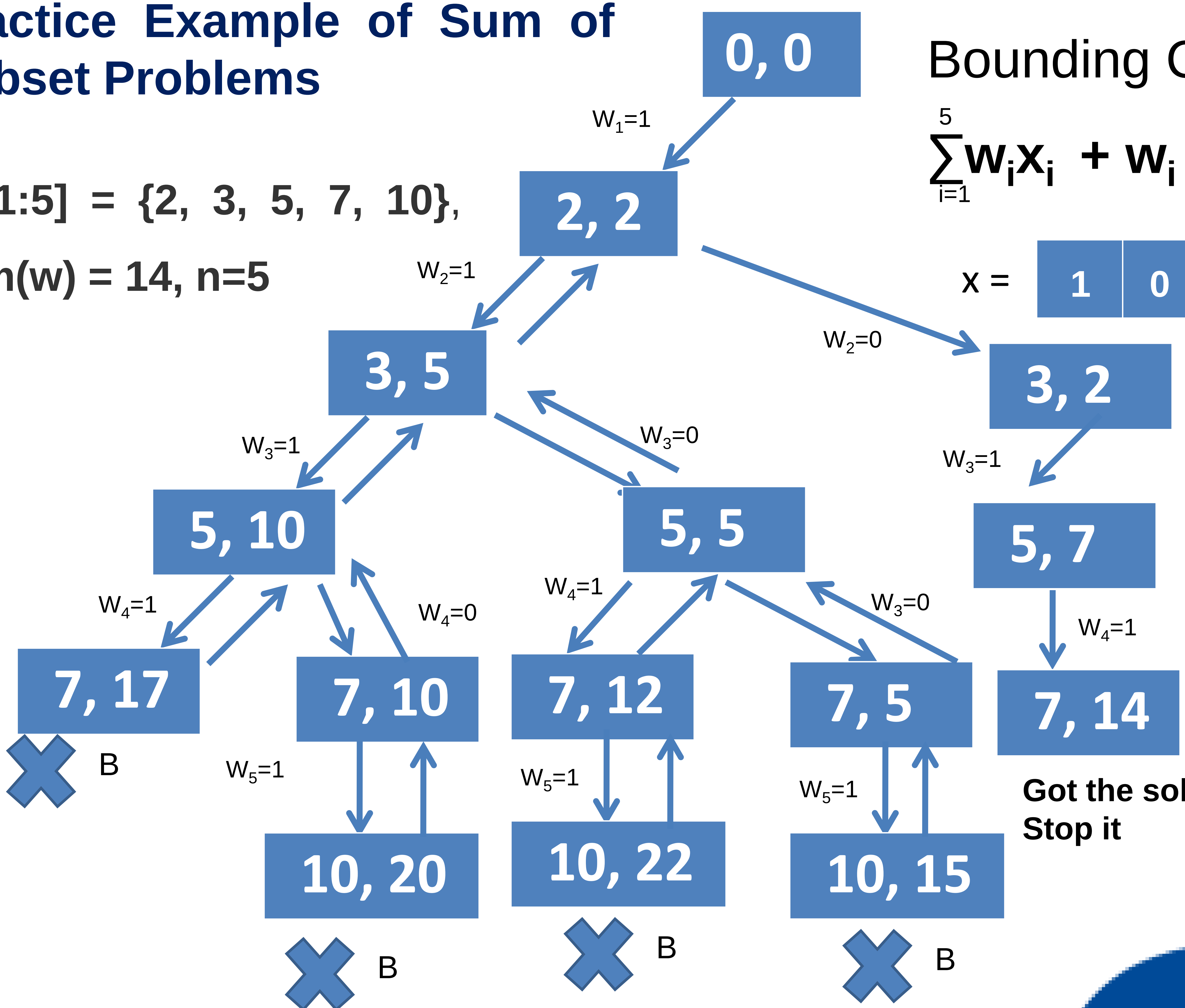
w [1:5] = {2, 3, 5, 7, 10},

Sum(w) = 14, n=5

Bounding Condition

$$\sum_{i=1}^{5} w_i x_i + w_i <= \text{Sum (w)}$$

x =	1	0	1	1	

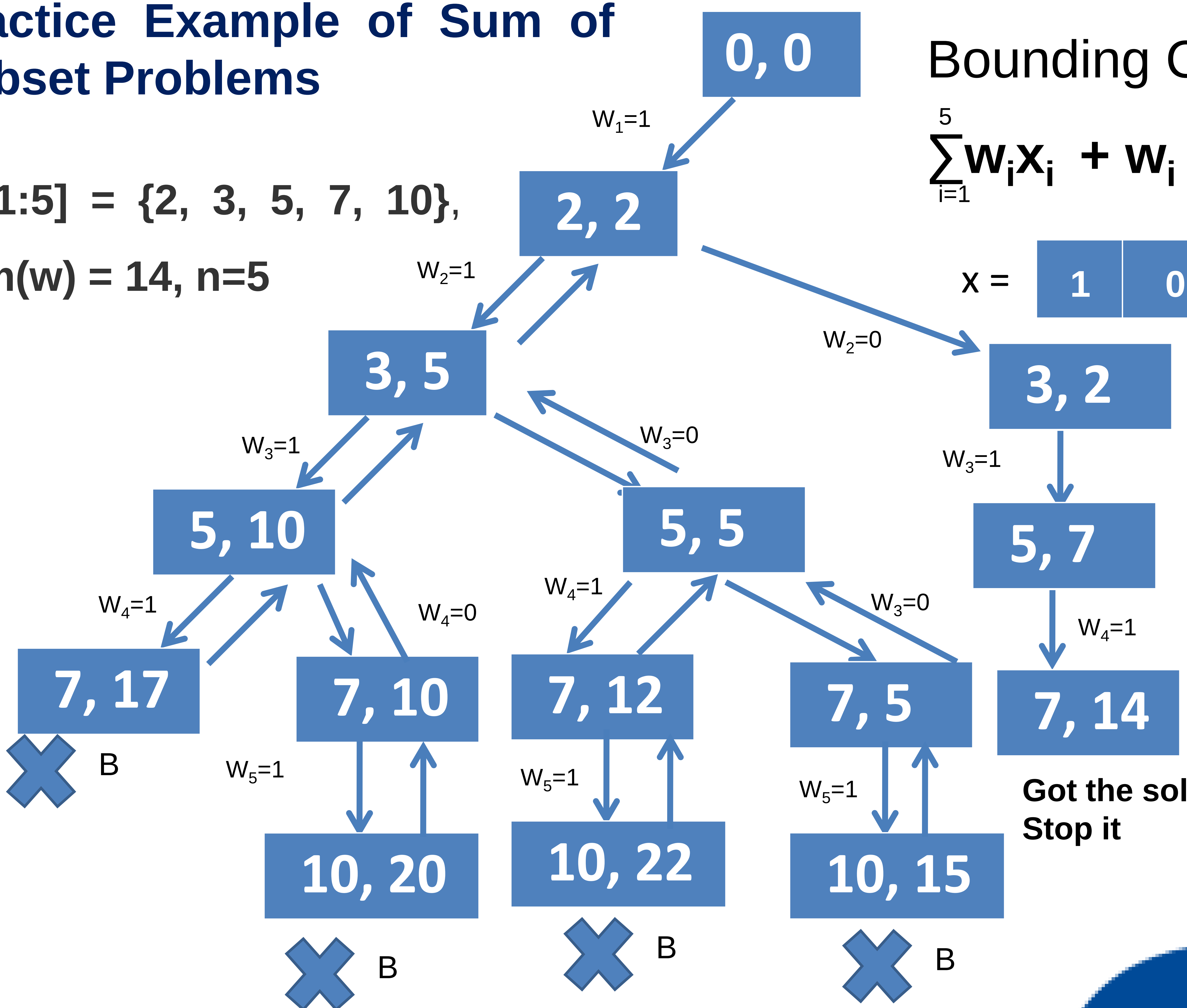


Got the solution
Stop it

Practice Example of Sum of Subset Problems

w [1:5] = {2, 3, 5, 7, 10},

Sum(w) = 14, n=5



Got the solution
Stop it

Bounding Condition

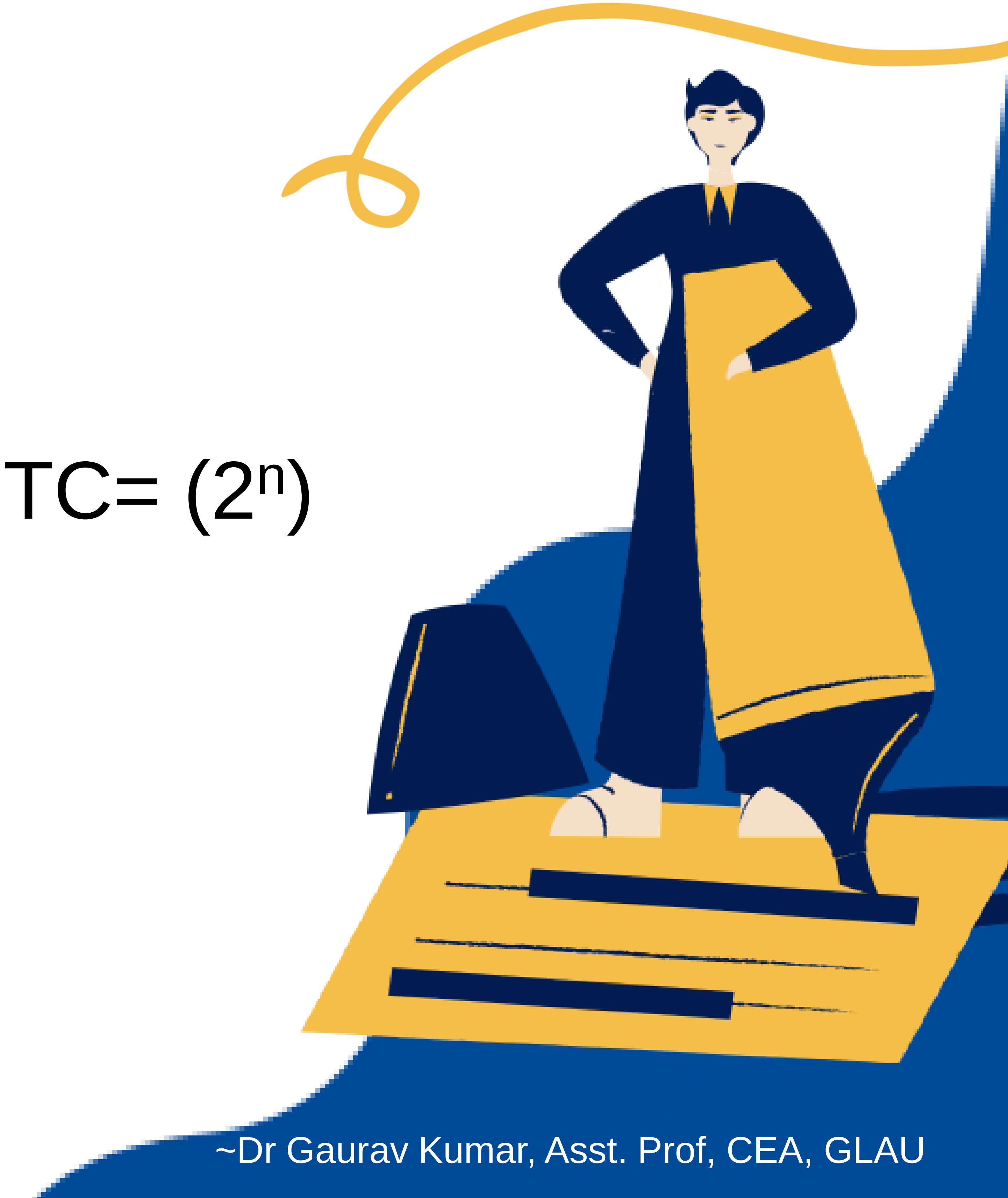
$$\sum_{i=1}^{5} w_i x_i \; + w_i <= \text{Sum (w)}$$

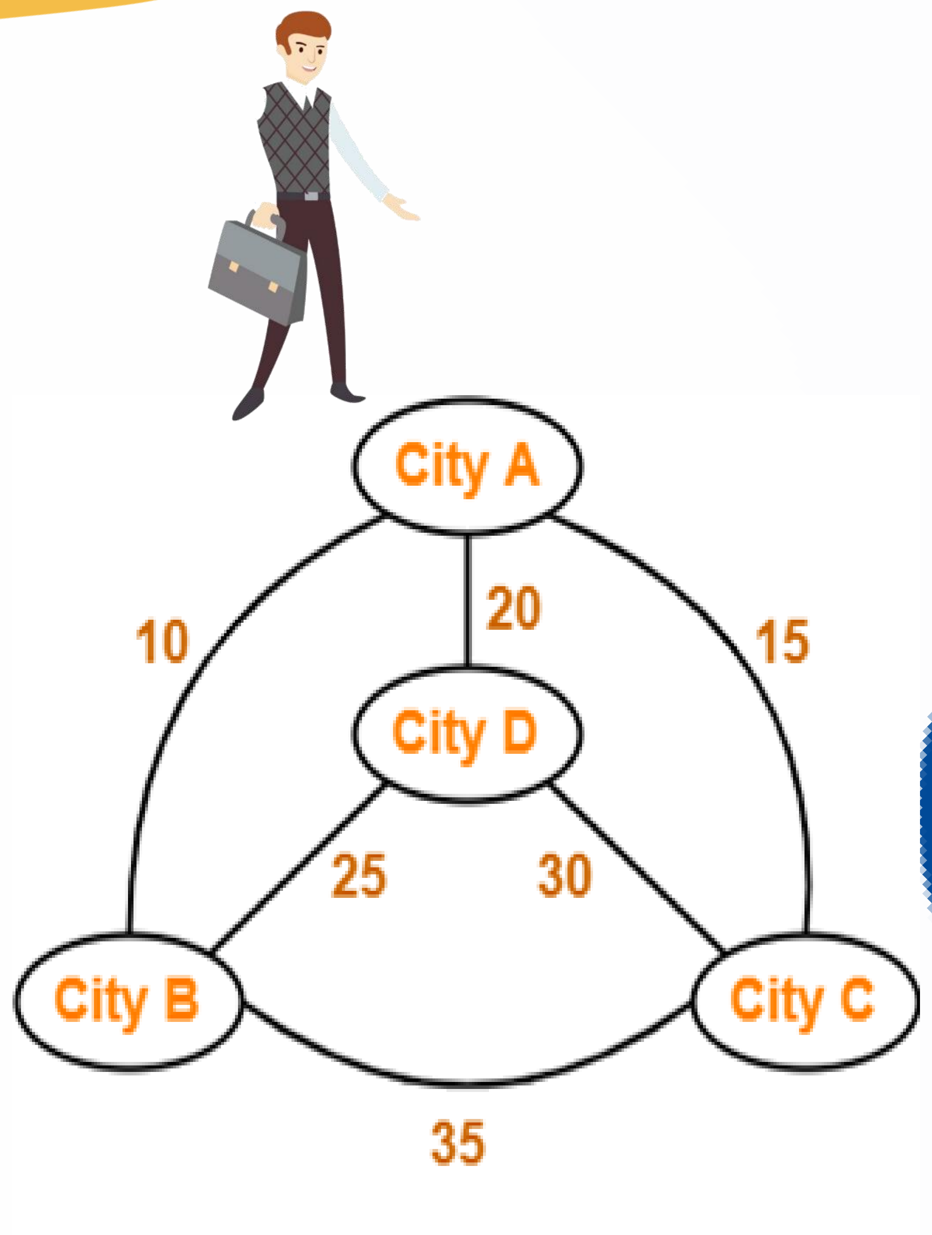
x =				
1	0	1	1	0

Time Complexity of Sum of Subset Problems

```
void subset_sum(int list[], int sum, int starting_index, int target_sum)
{
if( target_sum == sum )
{
subset_count++;
if(starting_index < list.length)
subset_sum(list, sum - list[starting_index-1],starting_index, target_sum);
}
else
{
for( int i = starting_index; i < list.length; i++ )
{
subset_sum(list, sum + list[i], i + 1, target_sum);
}
}
}
```

TC= (2^n)





03. Traveling Salesman Problem

Using Branch and Bound

Traveling Salesman Problem

Branch-and-Bound

- A **BFS-like state space search** will be called FIFO search as the list of live nodes is a first-in-first-out list (or queue).

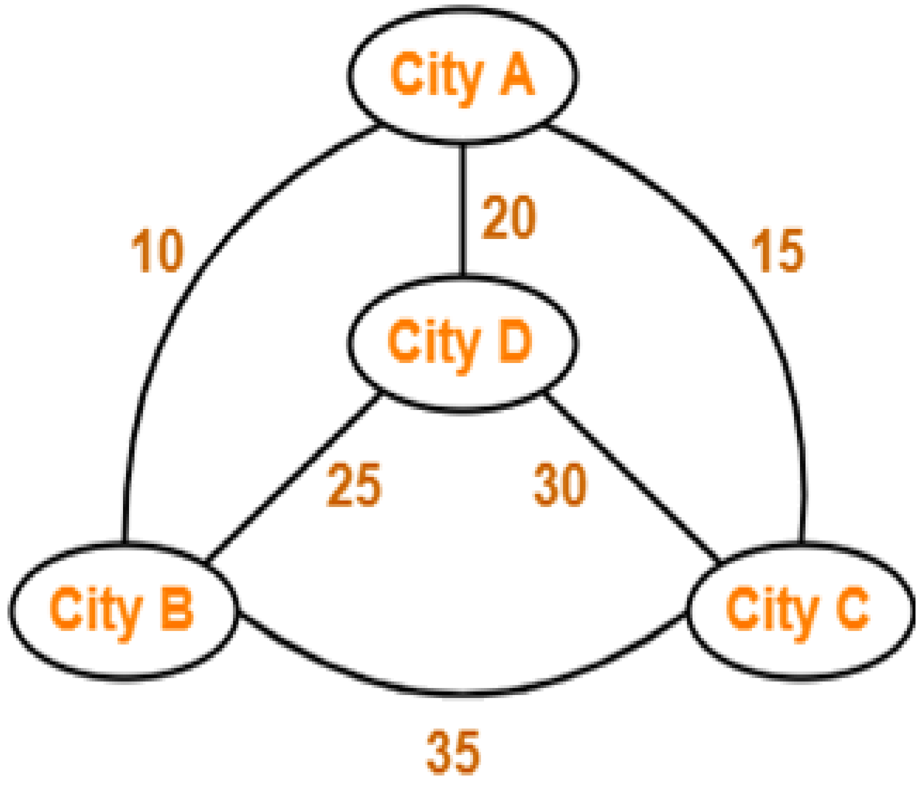
or

- A **DFS-search like state space search** will be called LIFO search as the list of live nodes is a last-in-first-out list (or stack).

Traveling Salesman Problem

Problem Statement

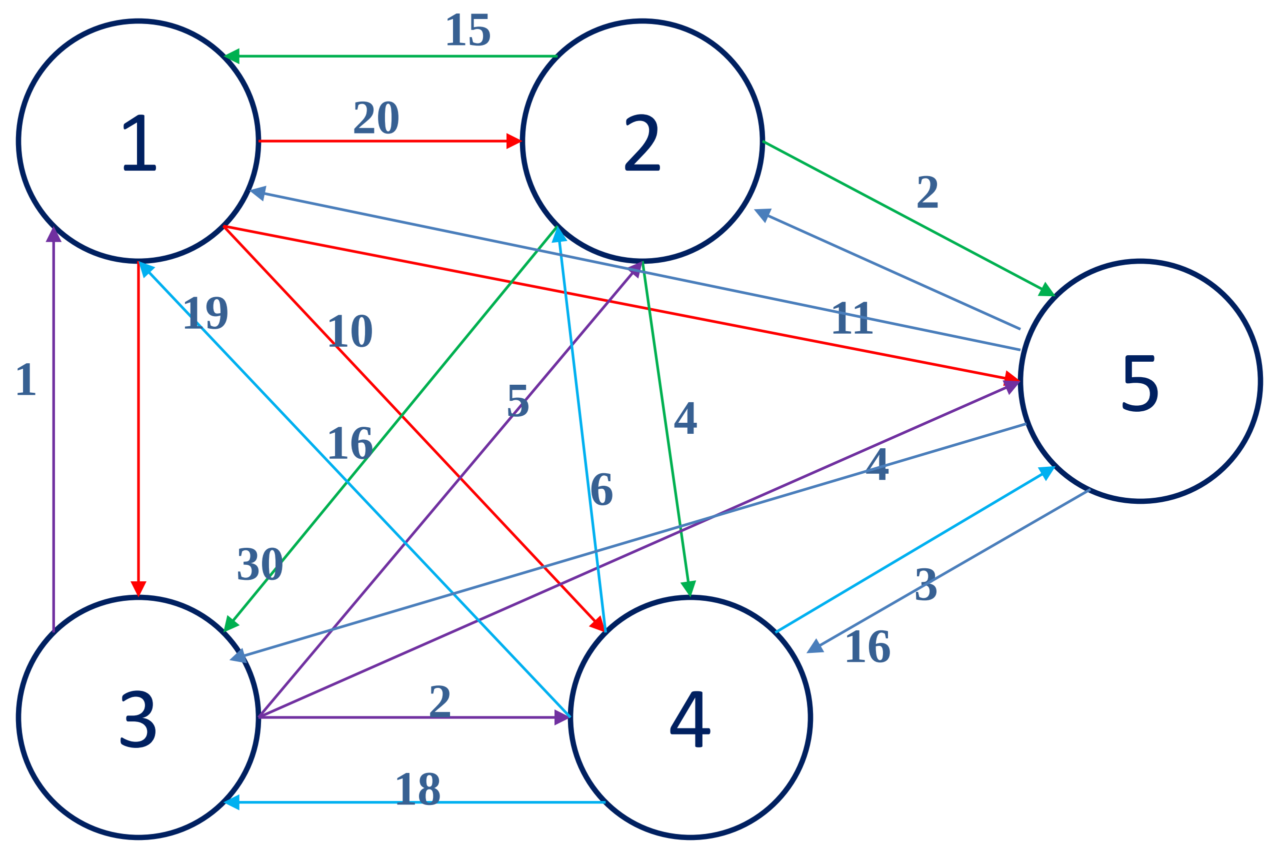
- Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.



Example of Traveling Salesman Problem

Step-01:

Write the initial cost matrix and reduce it



$$
\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & \infty & 20 & 30 & 10 & 11 \\
2 & 15 & \infty & 16 & 4 & 2 \\
3 & 3 & 5 & \infty & 2 & 4 \\
4 & 19 & 6 & 18 & \infty & 3 \\
5 & 16 & 4 & 7 & 16 & \infty
\end{array}
$$

Rules

- To reduce a matrix, perform the row reduction and column reduction of the matrix separately.
- A row or a column is said to be reduced if it contains at least one entry ‘0’ in it.

Traveling Salesman Problem

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & 20 & 30 & 10 & 11 \\ 2 & 15 & \infty & 16 & 4 & 2 \\ 3 & 3 & 5 & \infty & 2 & 4 \\ 4 & 19 & 6 & 18 & \infty & 3 \\ 5 & 16 & 4 & 7 & 16 & \infty \end{array} \xrightarrow{\text{Row-wise Min.}} \begin{array}{c|ccccc|c} & 1 & 2 & 3 & 4 & 5 & \\ \hline 1 & \infty & 20 & 30 & 10 & 11 & 10 \\ 2 & 15 & \infty & 16 & 4 & 2 & 2 \\ 3 & 3 & 5 & \infty & 2 & 4 & 2 \\ 4 & 19 & 6 & 18 & \infty & 3 & 3 \\ 5 & 16 & 4 & 7 & 16 & \infty & 4 \end{array}$$

Row-wise Total Cost = 21

Row Reduction

Consider the rows of above matrix one by one.

If the row already contains an entry ‘0’, then-

- There is no need to reduce that row.

If the row does not contains an entry ‘0’, then-

- Reduce that particular row.
- Select the least value element from that row.
- Subtract that element from each element of that row.
- This will create an entry ‘0’ in that row, thus reducing that row.

Traveling Salesman Problem

Column Reduction

Consider the columns of above row-reduced matrix one by one.

If the column already contains an entry ‘0’, then-

- There is no need to reduce that column.

If the column does not contains an entry ‘0’, then-

- Reduce that particular column.
- Select the least value element from that column.
- Subtract that element from each element of that column.
- This will create an entry ‘0’ in that column, thus reducing that column

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

= 4 (Colum-wise total cost)

Col.-wise Min.

Traveling Salesman Problem

	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	∞

Row-wise Min.

	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

Row-wise Total Cost = 21

Subtract

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

=4 (Colum-wise total cost)

Col.-wise Min.

Subtract

Reduced Cost Matrix

Reduced Cost= Row-wise cost + Column wise cost

C = 21+4=25

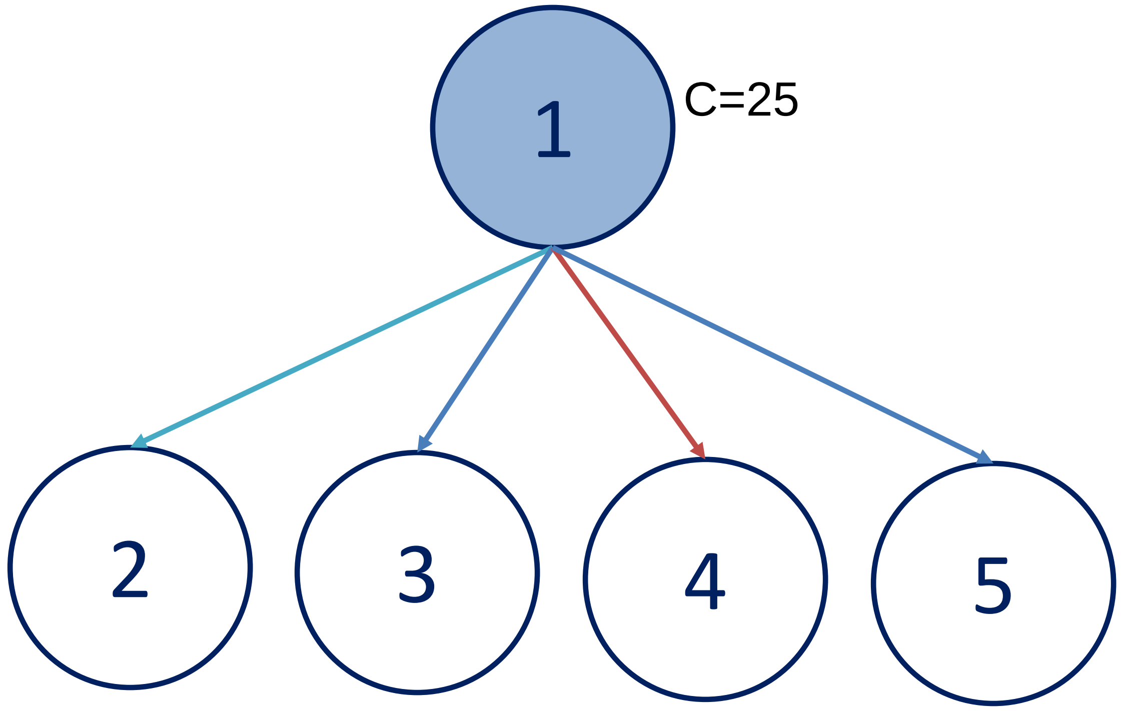
Reduced Cost Matrix at Node 1 (RCM-1)

	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	∞

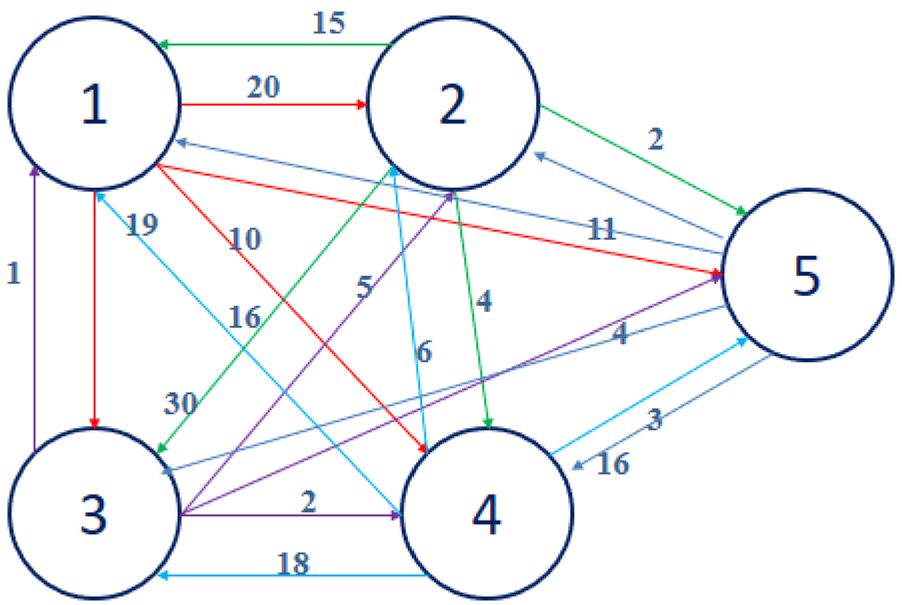
Traveling Salesman Problem

Step-02

- We consider all other vertices one by one.
- We select the best vertex where we can land upon to minimize the tour cost.



C=?



Choosing To Go To Vertex-2: Node-2 (Path 1 → 2)

- From the reduced matrix of step-01, Cost[1,2] = ?
- Set row-1 and column-2 to ∞
- Set c[2,1] = ∞

Now, resulting cost matrix is-

	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	∞

RCM-1

1 to 2 →

Now, We reduce this matrix.

Then, we find out the cost of node-02.

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

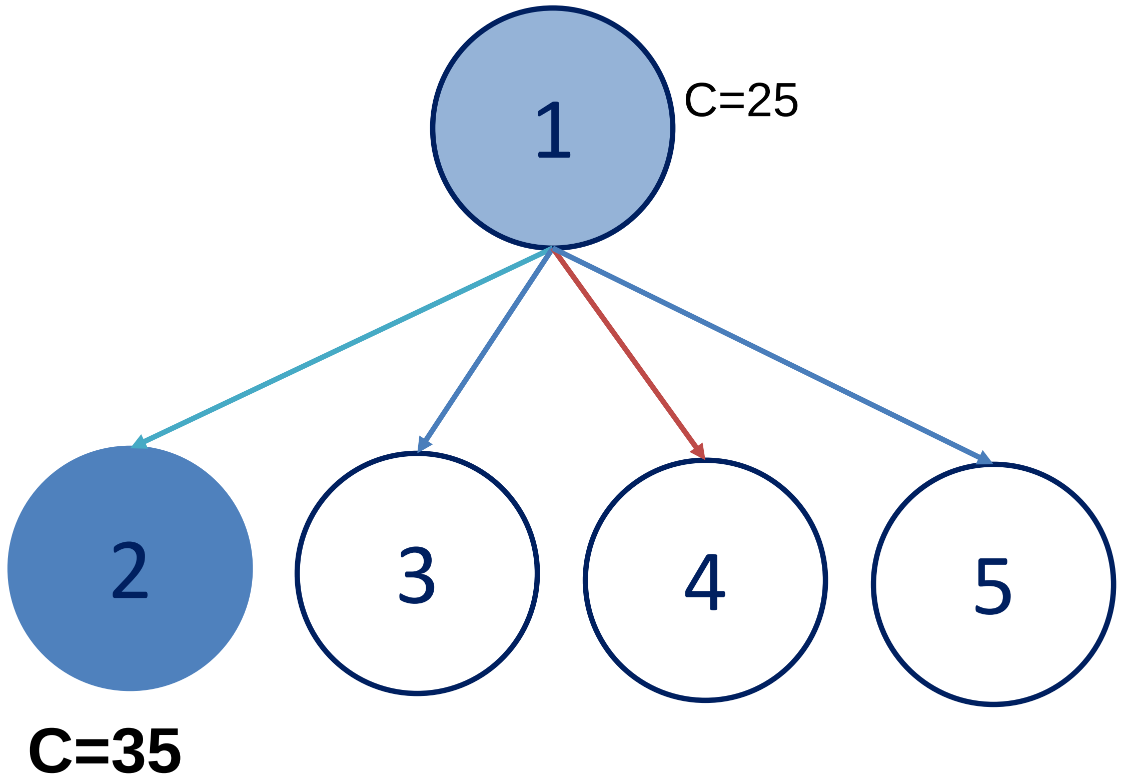
Row Reduction

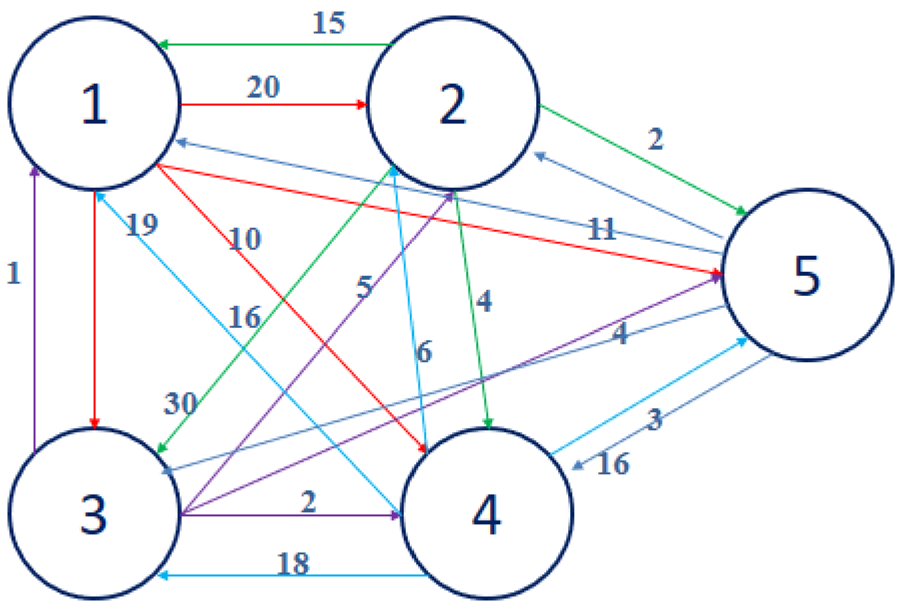
=0

Traveling Salesman Problem

Step-02

- We consider all other vertices one by one.
- We select the best vertex where we can land upon to minimize the tour cost.





Choosing To Go To Vertex-2: Node-2 (Path 1 → 2)

- From the reduced matrix of step-01, Cost[1,2] = ?
- Set row-A and column-B to ∞
- Set c[2,1] = ∞

Now, resulting cost matrix is-

	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	∞

1 to 2

Now, We reduce this matrix.

Then, we find out the cost of node-02.

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

Row Reduction

=0

RCM-1

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

Column Reduction

=0

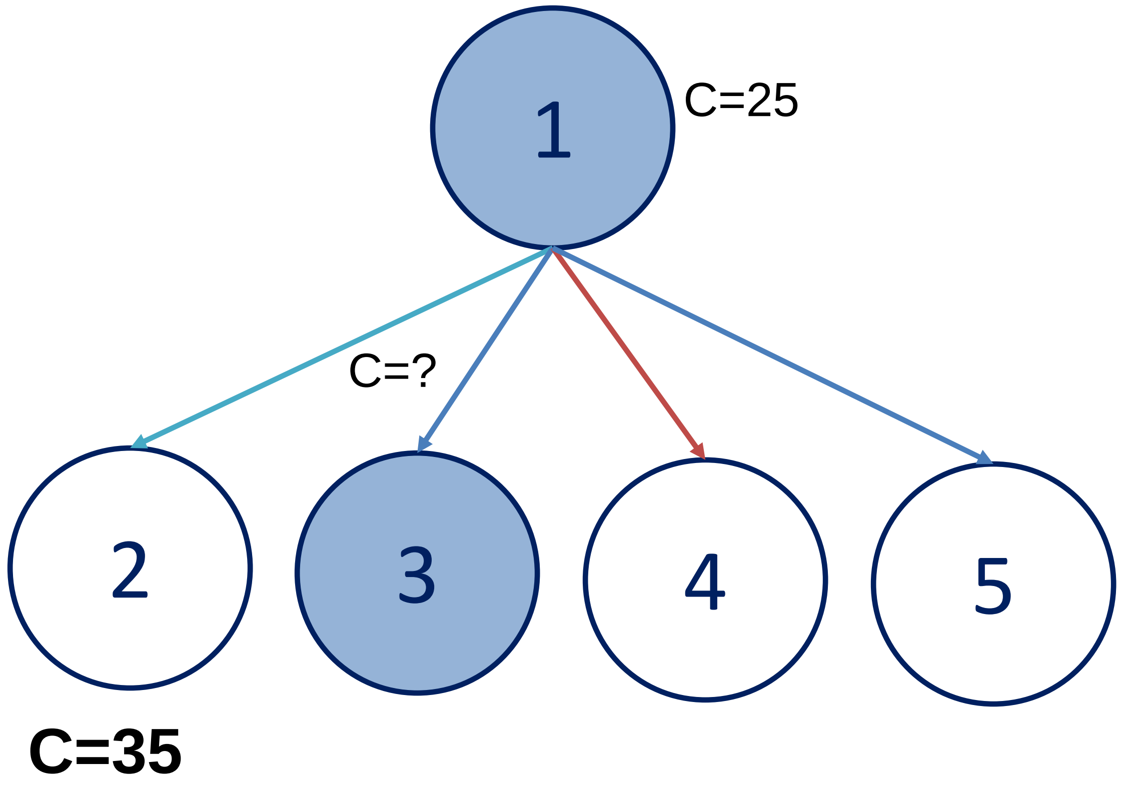
Cost(1,2)= c(1,2)+C+C’
= 10+25+0
= 35

Traveling Salesman Problem

Choosing To Go To Vertex-3: Node-3 (Path 1 → 3)

- From the reduced matrix of step-01, Cost[1,3] = ?
- Set row-1 and column-3 to ∞
- Set c[3,1] = ∞

Now, resulting cost matrix is-



$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & 10 & 17 & 0 & 1 \\ 2 & 12 & \infty & 11 & 2 & 0 \\ 3 & 0 & 3 & \infty & 0 & 2 \\ 4 & 15 & 3 & 12 & \infty & 0 \\ 5 & 11 & 0 & 0 & 12 & \infty \end{array}$$

1 to 3

$$\begin{array}{c|ccccc|c} & 1 & 2 & 3 & 4 & 5 & \\ \hline 1 & \infty & \infty & \infty & \infty & \infty & 0 \\ 2 & 12 & \infty & \infty & 2 & 0 & 0 \\ 3 & \infty & 3 & \infty & 0 & 2 & 0 \\ 4 & 15 & 3 & \infty & \infty & 0 & 0 \\ 5 & 11 & 0 & \infty & 12 & \infty & 0 \end{array}$$

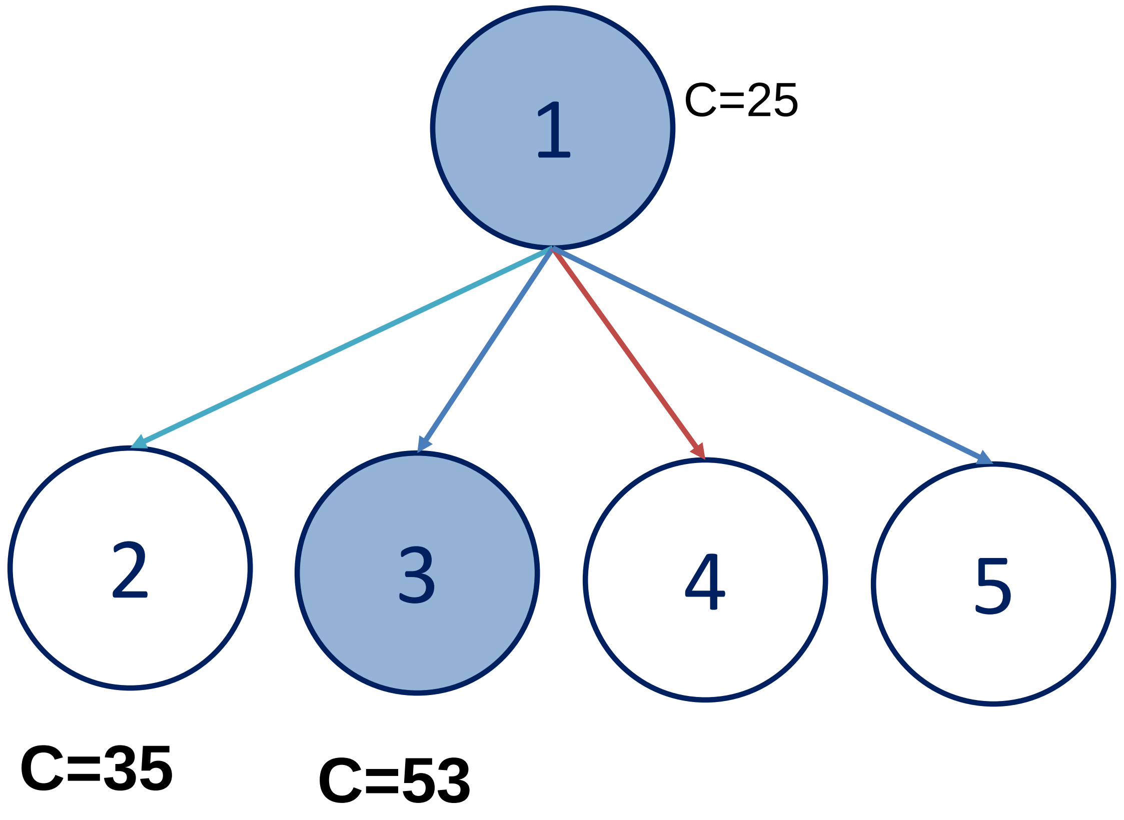
=0

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 12 & \infty & \infty & 2 & 0 \\ 3 & \infty & 3 & \infty & 0 & 2 \\ 4 & 15 & 3 & \infty & \infty & 0 \\ 5 & 11 & 0 & \infty & 12 & \infty \\ \hline & 11 & 0 & 0 & 0 & 0 \end{array}$$

=11

Cost(1,3)= c(1,3)+C+C'
= 17+25+11
= 53

Traveling Salesman Problem



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

1 to 3

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

=0

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

=11

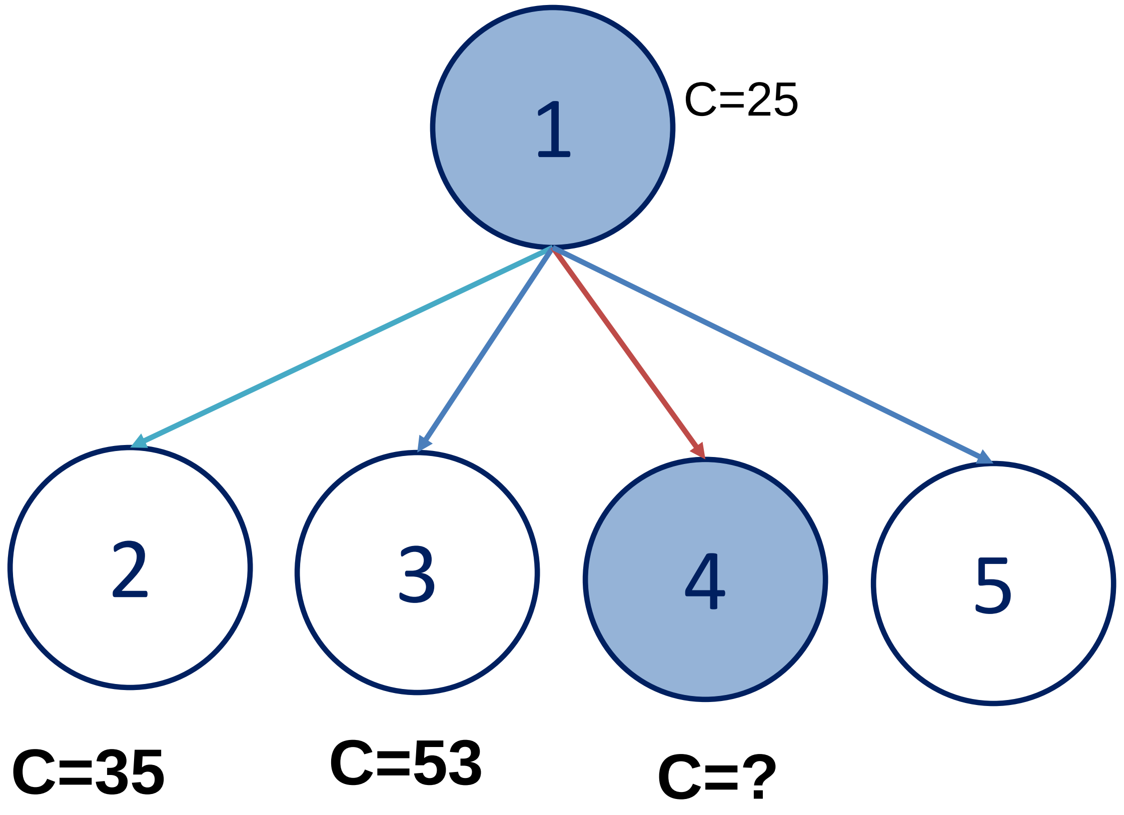
Cost(1,3)= c(1,3)+C+C’
= 17+25+11
= 53

Traveling Salesman Problem

Choosing To Go To Vertex-4: Node-4 (Path 1 → 4)

- From the reduced matrix of step-01, Cost[1,4] = ?
- Set row-1 and column-4 to ∞
- Set c[4,1] = ∞

Now, resulting cost matrix is-



	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	∞

1 to 4

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

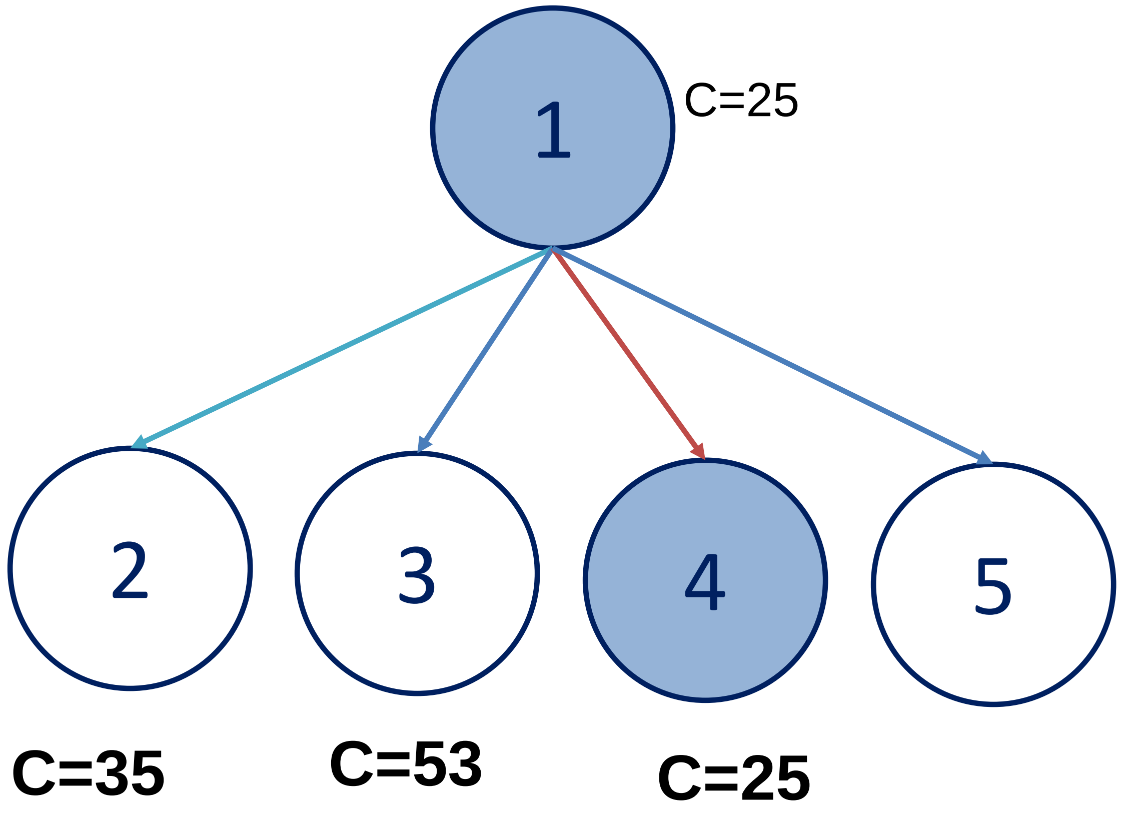
=0

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

=0

Cost(1,4)= c(1,4)+C+C’
= 0+25+0
= 25

Traveling Salesman Problem



	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	∞

1 to 4

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

=0

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

=0

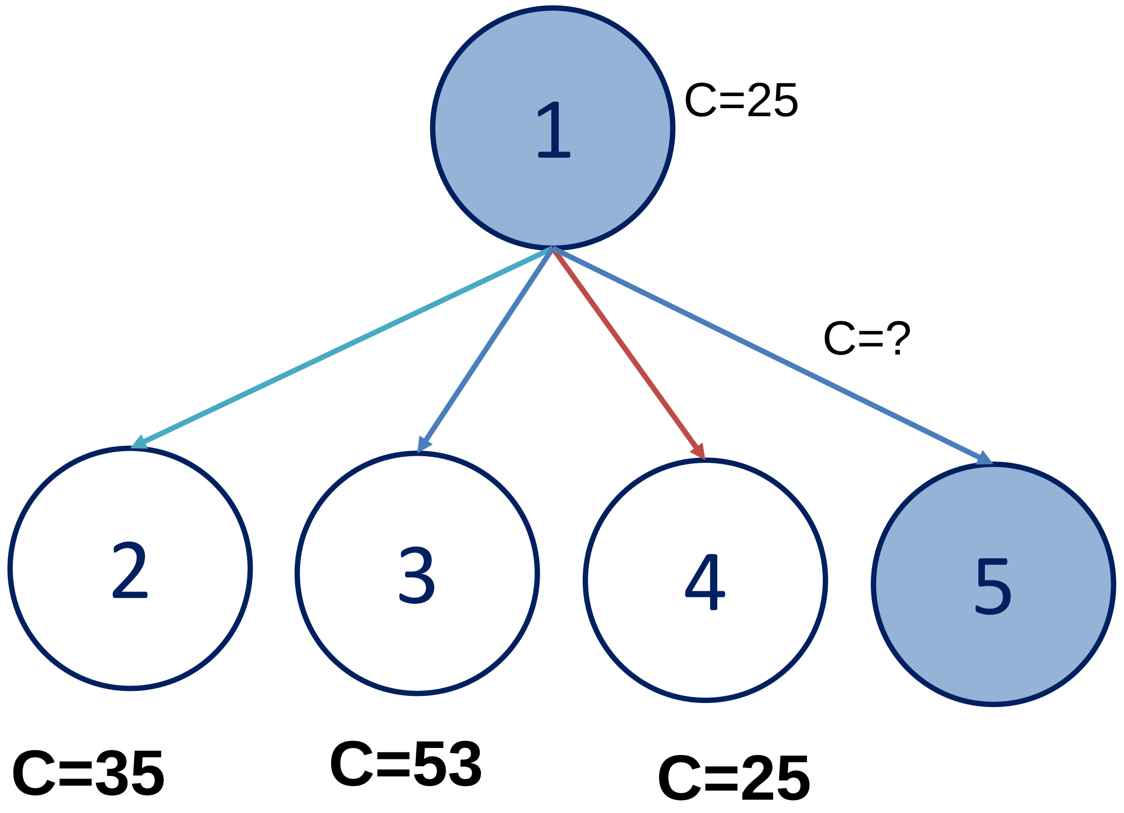
Cost(1,4)= c(1,4)+C+C’
= 0+25+0
= 25

Traveling Salesman Problem

Choosing To Go To Vertex-5: Node-5 (Path 1 → 5)

- From the reduced matrix of step-01, Cost[1,] = ?
- Set row-1 and column-5 to ∞
- Set c[5,1] = ∞

Now, resulting cost matrix is-



	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	∞

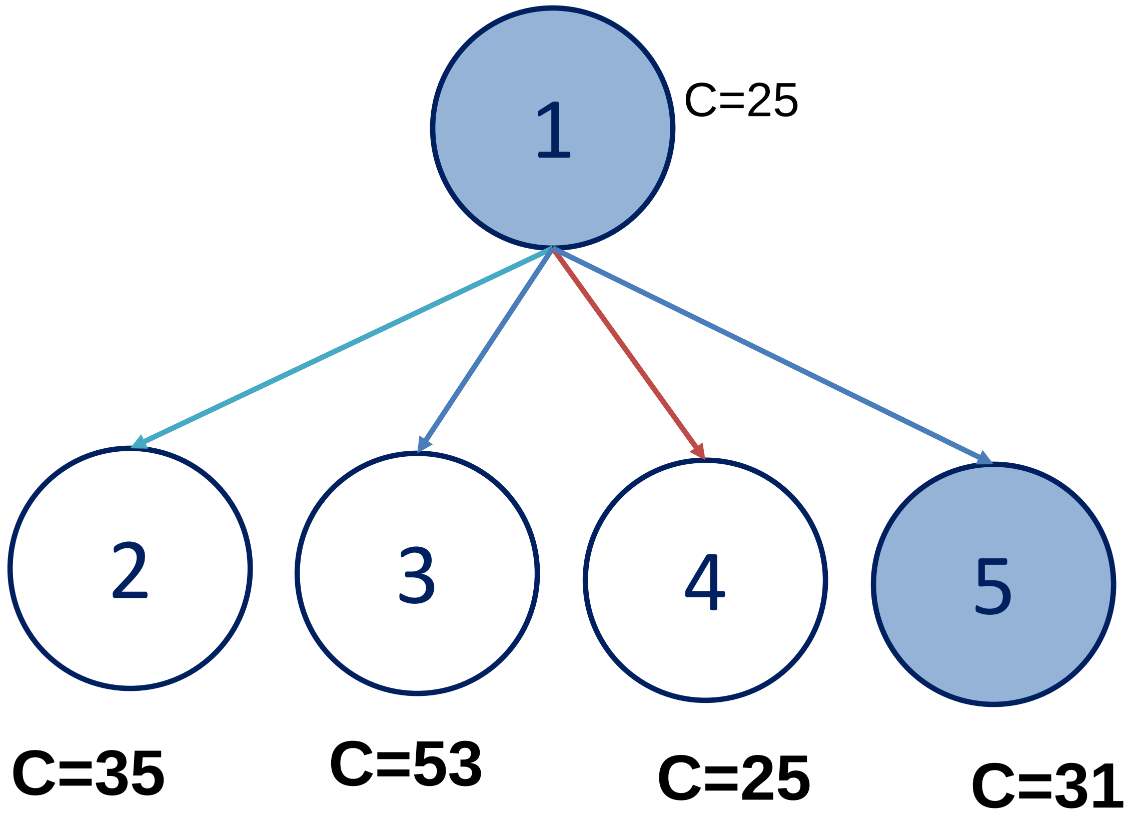
1 to 5 →

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

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	
2	10	∞	9	0	∞	
3	0	3	∞	0	∞	
4	12	0	9	∞	∞	
5	∞	0	0	12	∞	
	0	0	0	0	0	=0

Cost(1,5)= c(1,5)+C+C'
= 1+25+5
= 31

Traveling Salesman Problem



	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	∞

1 to 5

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

=5

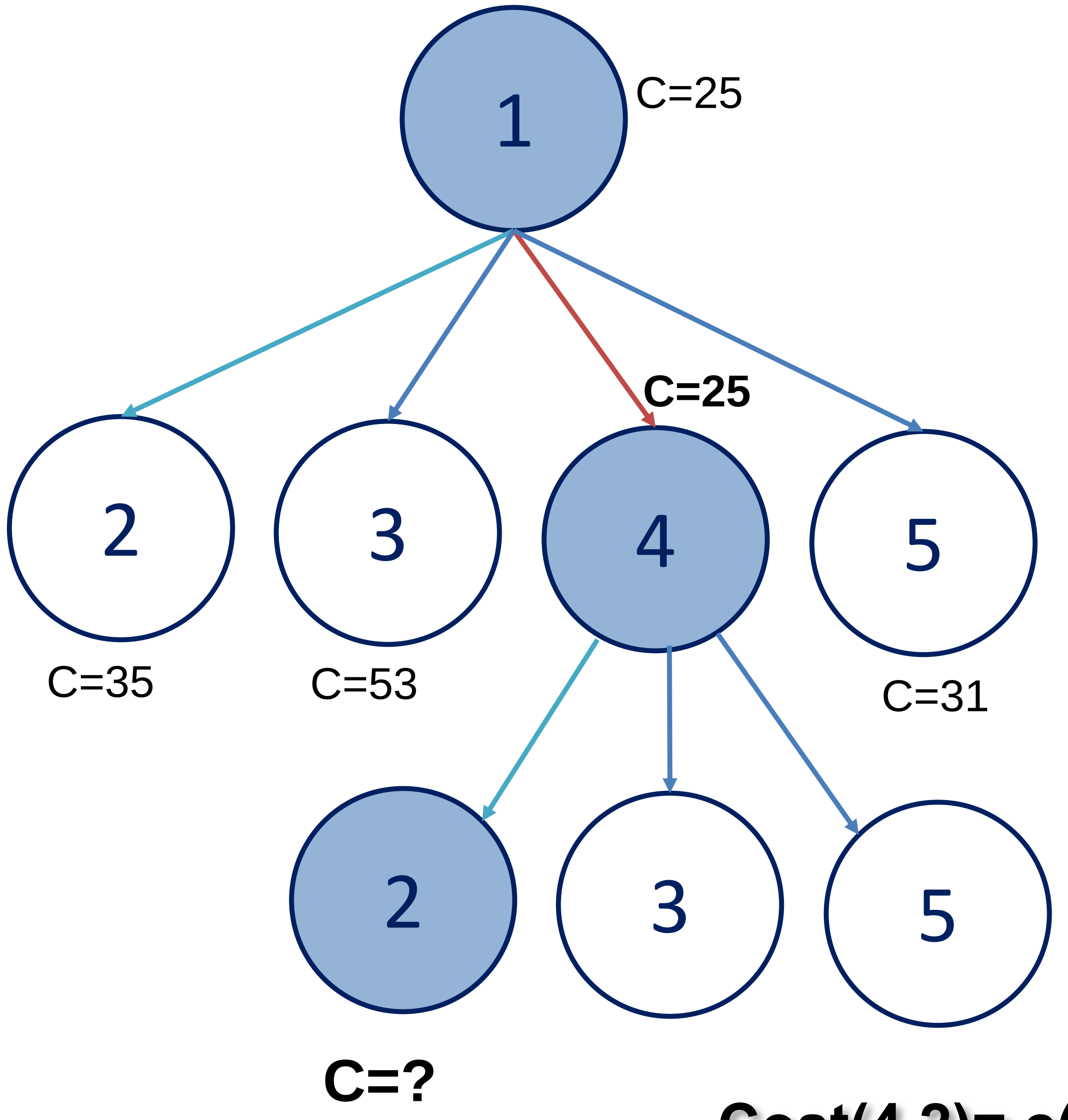
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	10	∞	9	0	∞
3	0	3	∞	0	∞
4	12	0	9	∞	∞
5	∞	0	0	12	∞
	0	0	0	0	0

=0

Select the minimum cost path between two pairs and continue till all paths are explored (repeat the same steps of row and column reductions for the next possible pairs)

Cost(1,5)= c(1,5)+C+C’
= 1+25+5
= 31

Traveling Salesman Problem



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

1,4-2

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

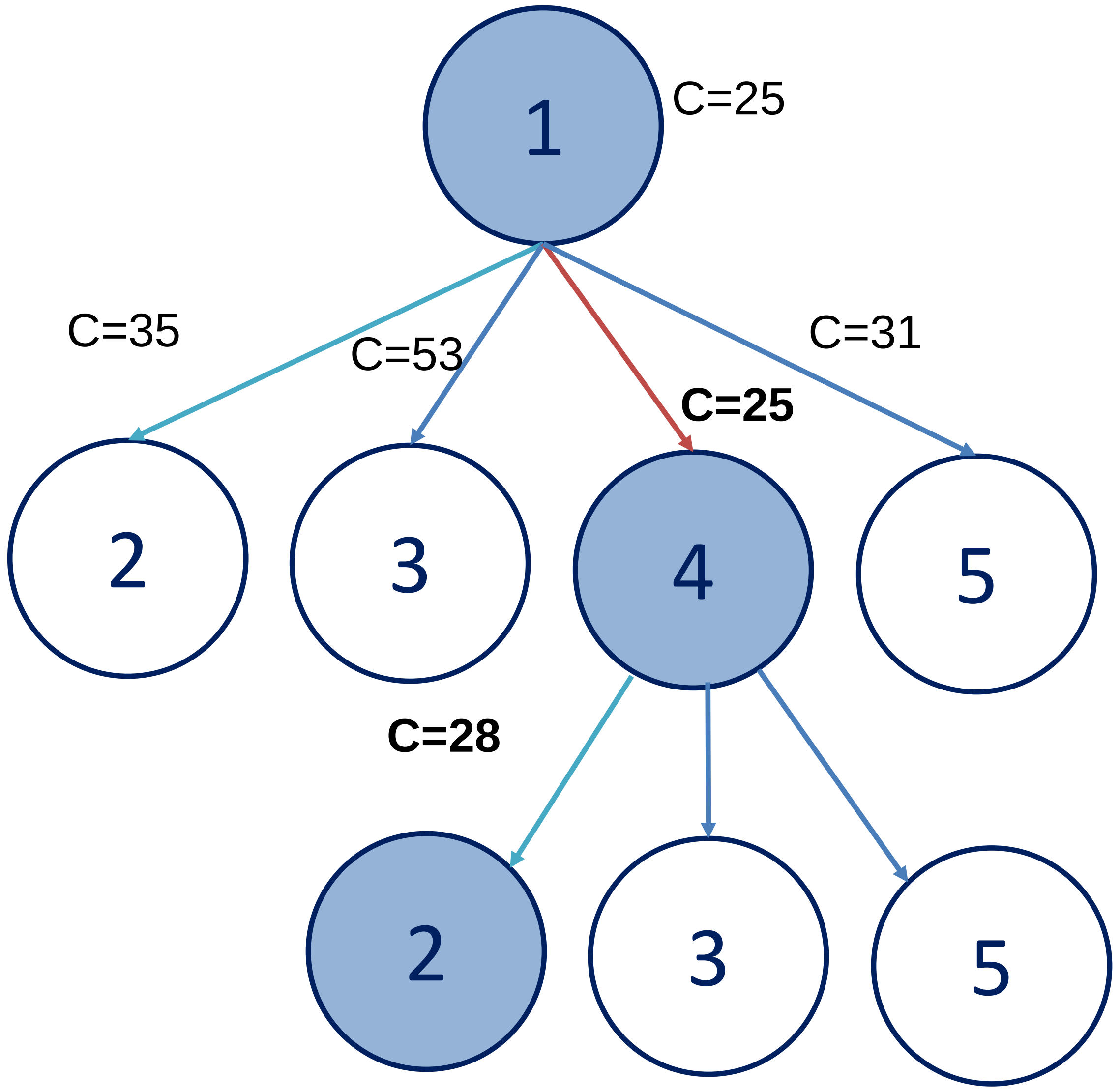
=0

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

=0

Cost(4,2)= c(4,2)+C+C'
= 3+25+0
= 28

Traveling Salesman Problem



Cost(4,2)= c(4,2)+C+C'
= 3+25+0
= 28

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

1,4-2

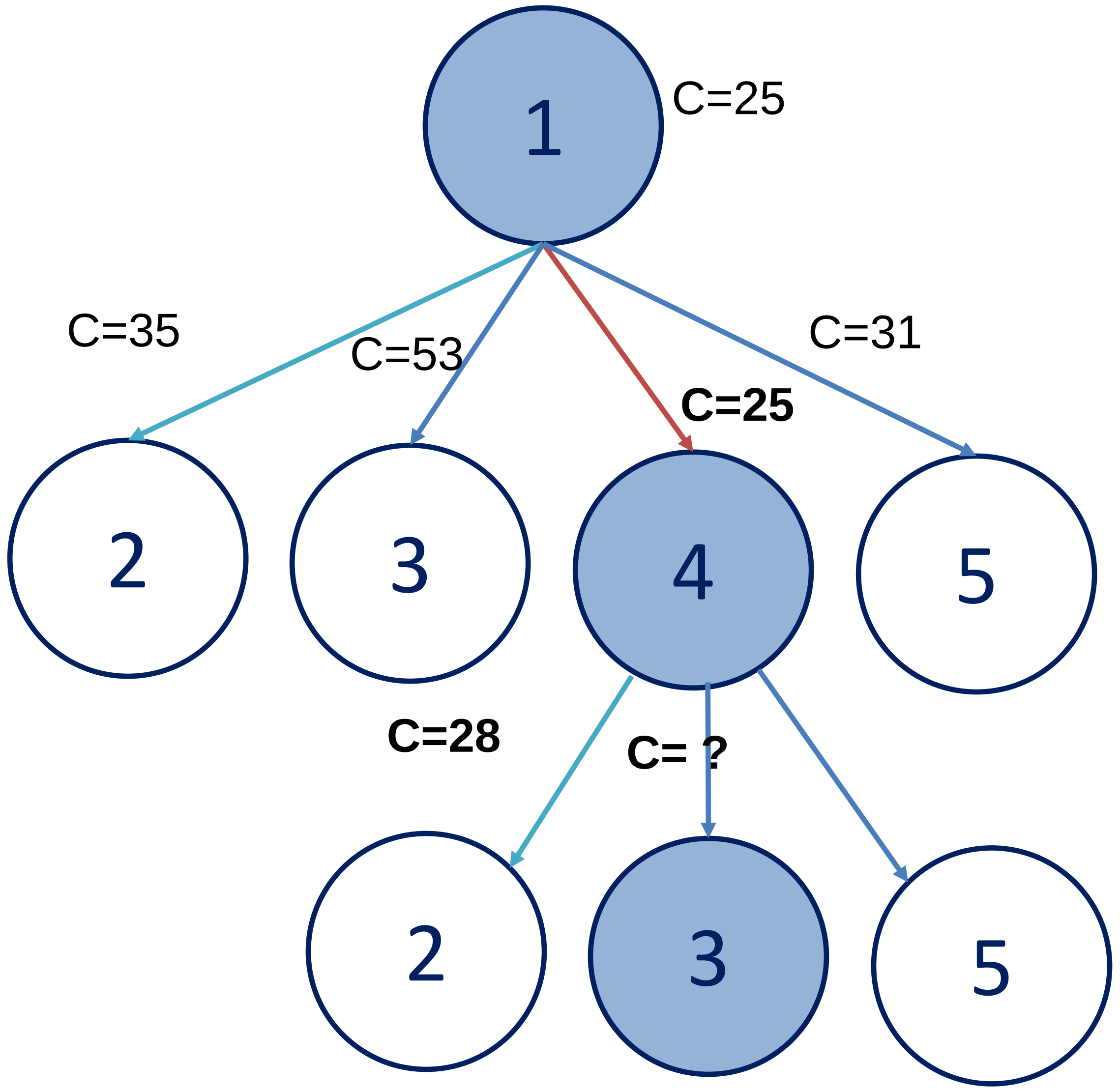
	1	2	3	4	5	
1	ꝏ	ꝏ	ꝏ	ꝏ	ꝏ	0
2	ꝏ	ꝏ	11	ꝏ	0	0
3	0	ꝏ	ꝏ	ꝏ	2	0
4	ꝏ	ꝏ	ꝏ	ꝏ	ꝏ	0
5	11	ꝏ	0	ꝏ	ꝏ	0

=0

	1	2	3	4	5
1	ꝏ	ꝏ	ꝏ	ꝏ	ꝏ
2	ꝏ	ꝏ	11	ꝏ	0
3	0	ꝏ	ꝏ	ꝏ	2
4	ꝏ	ꝏ	ꝏ	ꝏ	ꝏ
5	11	ꝏ	0	ꝏ	ꝏ
	0	0	0	0	0

=0

Traveling Salesman Problem



Cost(4,3)= c(4,3)+C+C’
= 12+25+13
= 50

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

1,4-3

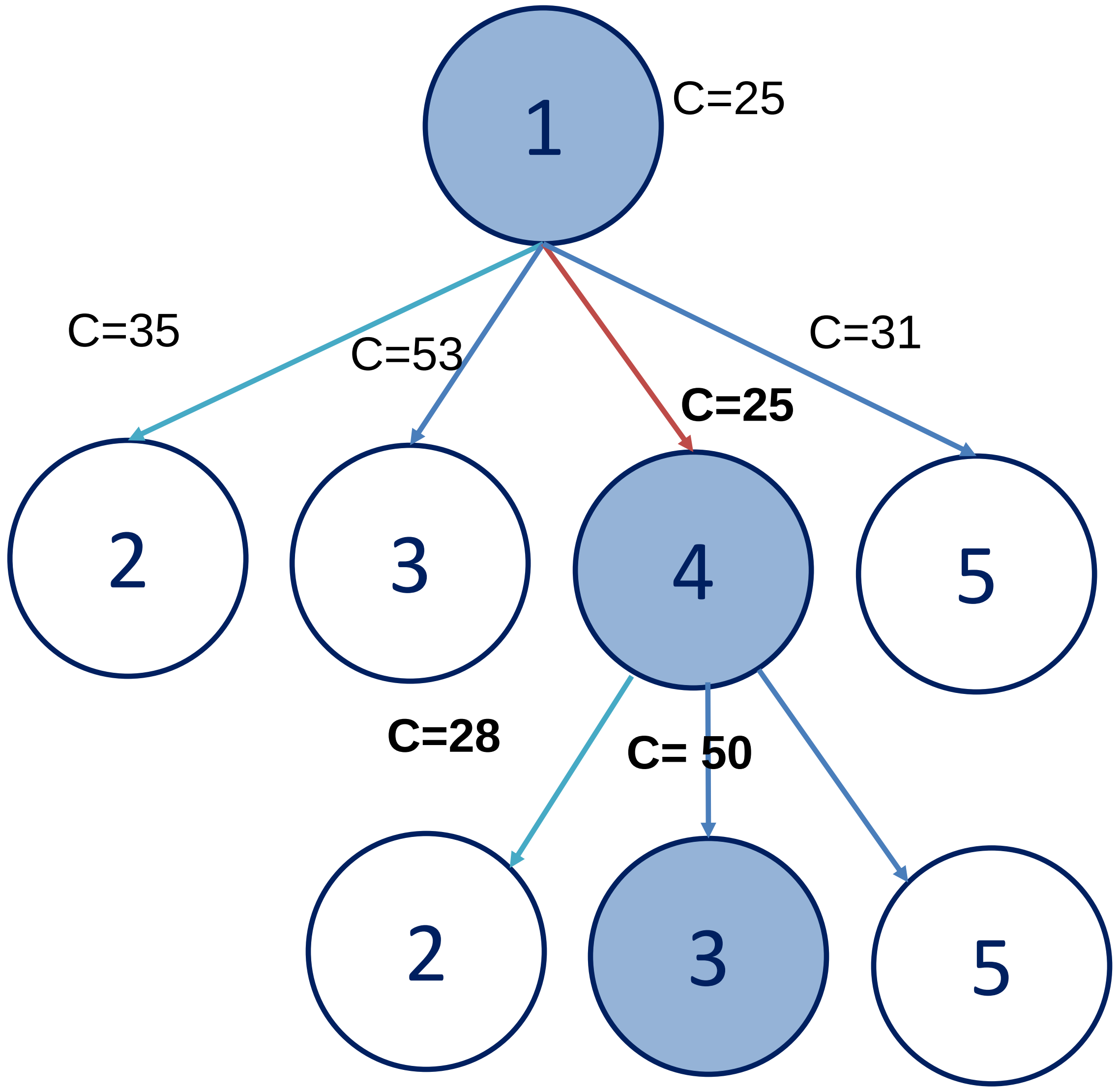
	1	2	3	4	5	
1	ꝏ	ꝏ	ꝏ	ꝏ	ꝏ	0
2	12	ꝏ	ꝏ	ꝏ	0	0
3	ꝏ	3	ꝏ	ꝏ	2	2
4	ꝏ	ꝏ	ꝏ	ꝏ	ꝏ	0
5	11	0	ꝏ	ꝏ	ꝏ	0

=2

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

=0

Traveling Salesman Problem



Cost(4,3)= c(4,3)+C+C’
= 12+25+13
= 50

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 12 & \infty & 11 & \infty & 0 \\ 3 & 0 & 3 & \infty & \infty & 2 \\ 4 & \infty & 3 & 12 & \infty & 0 \\ 5 & 11 & 0 & 0 & \infty & \infty \end{array}$$

1,4-3

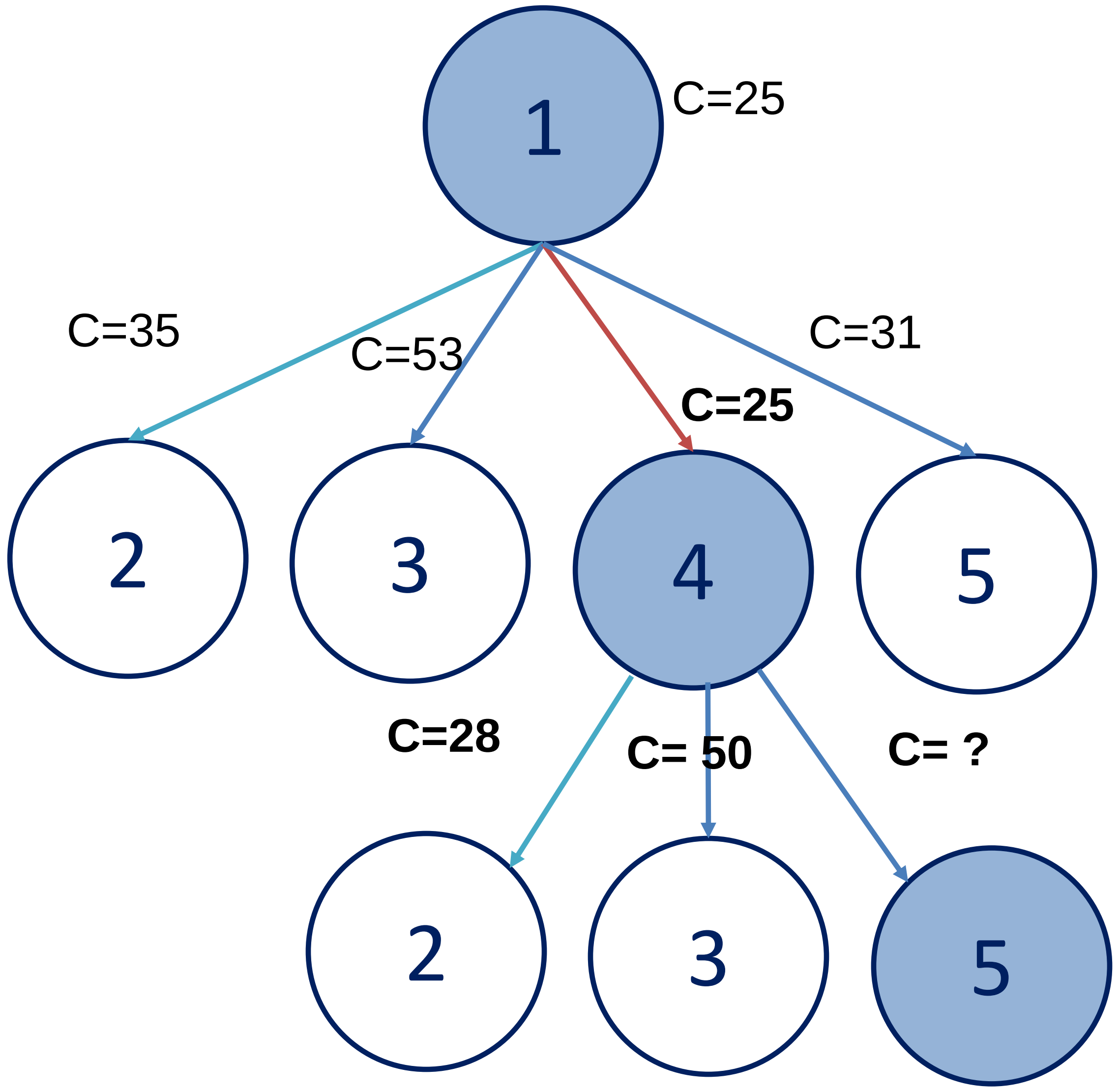
$$\begin{array}{c|ccccc|c} & 1 & 2 & 3 & 4 & 5 & \\ \hline 1 & \infty & \infty & \infty & \infty & \infty & 0 \\ 2 & 12 & \infty & \infty & \infty & 0 & 0 \\ 3 & \infty & 3 & \infty & \infty & 2 & 2 \\ 4 & \infty & \infty & \infty & \infty & \infty & 0 \\ 5 & 11 & 0 & \infty & \infty & \infty & 0 \end{array}$$

=2

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 1 & \infty & \infty & \infty & 0 \\ 3 & \infty & 1 & \infty & \infty & 0 \\ 4 & \infty & \infty & \infty & \infty & \infty \\ 5 & 0 & 0 & \infty & \infty & \infty \\ \hline & 11 & 0 & 0 & 0 & 0 \end{array}$$

=0

Traveling Salesman Problem



Cost(4,5)= c(4,5)+C+C’
= 0+25+11
= 36

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

1,4-5

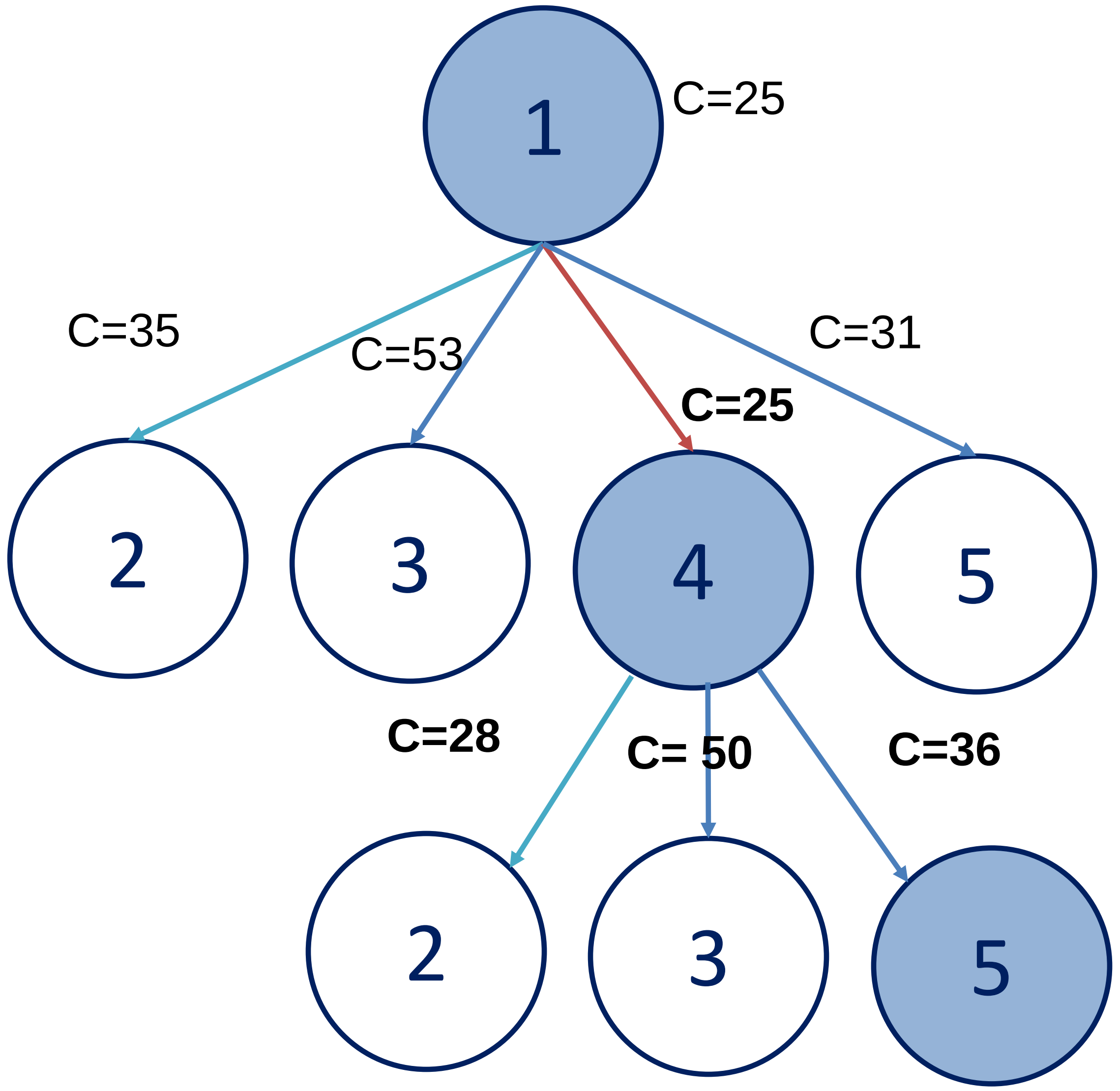
	1	2	3	4	5	
1	ꝏ	ꝏ	ꝏ	ꝏ	ꝏ	0
2	12	ꝏ	11	ꝏ	ꝏ	11
3	0	3	ꝏ	ꝏ	ꝏ	0
4	ꝏ	ꝏ	ꝏ	ꝏ	ꝏ	0
5	ꝏ	0	0	ꝏ	ꝏ	0

=11

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

=0

Traveling Salesman Problem



Cost(4,5)= c(4,5)+C+C'
= 0+25+11
= 36

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

1,4-5

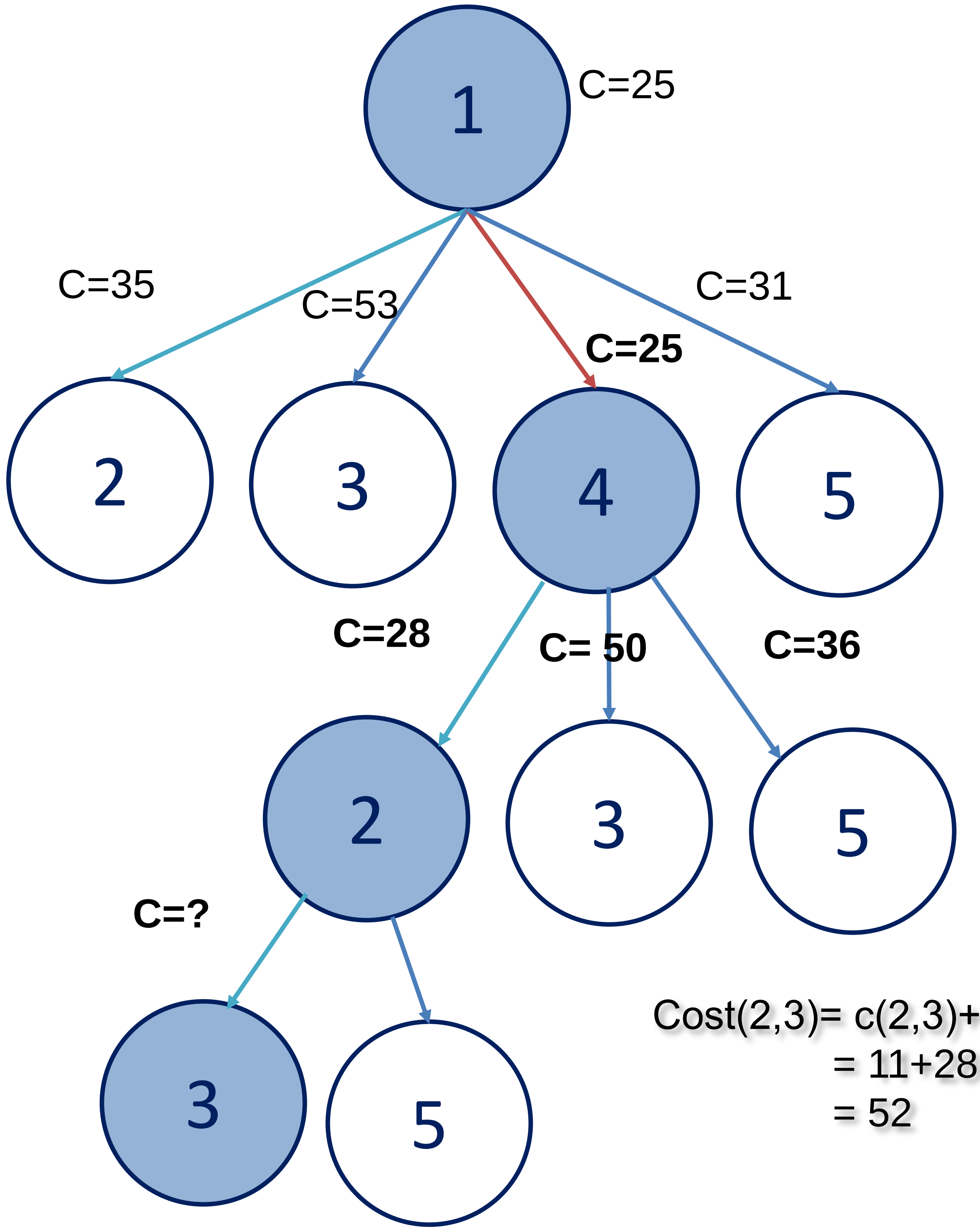
	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	12	∞	11	∞	∞	11
3	0	3	∞	∞	∞	0
4	∞	∞	∞	∞	∞	0
5	∞	0	0	∞	∞	0

=11

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

=0

Traveling Salesman Problem



Cost(2,3)= c(2,3)+C+C'
= 11+28+13
= 52

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

1-4-2-3

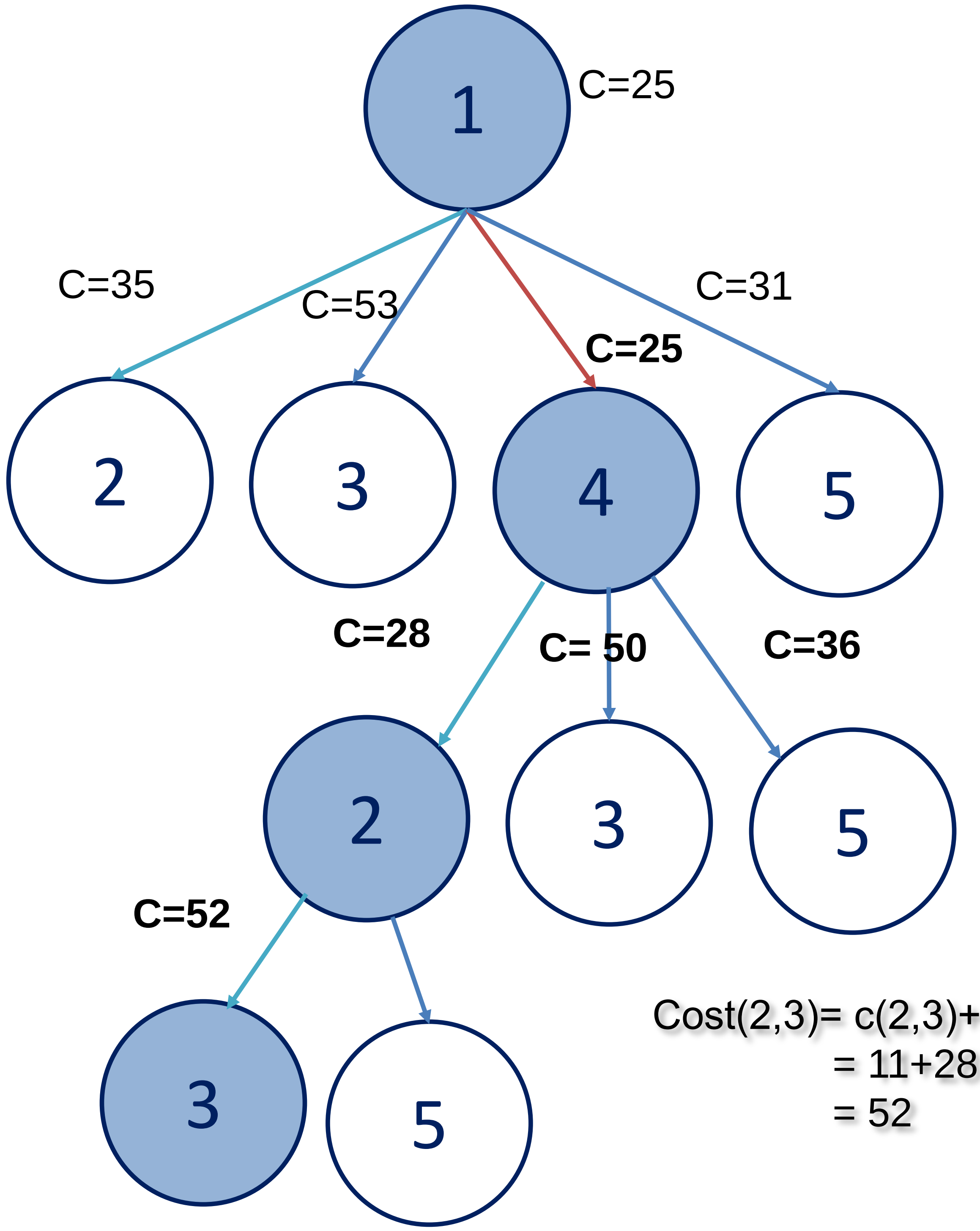
	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	∞	∞	∞	∞	∞	0
3	∞	∞	∞	∞	2	2
4	∞	∞	∞	∞	∞	0
5	11	∞	∞	∞	∞	11

=13

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

=0

Traveling Salesman Problem



Cost(2,3)= c(2,3)+C+C’
= 11+28+13
= 52

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

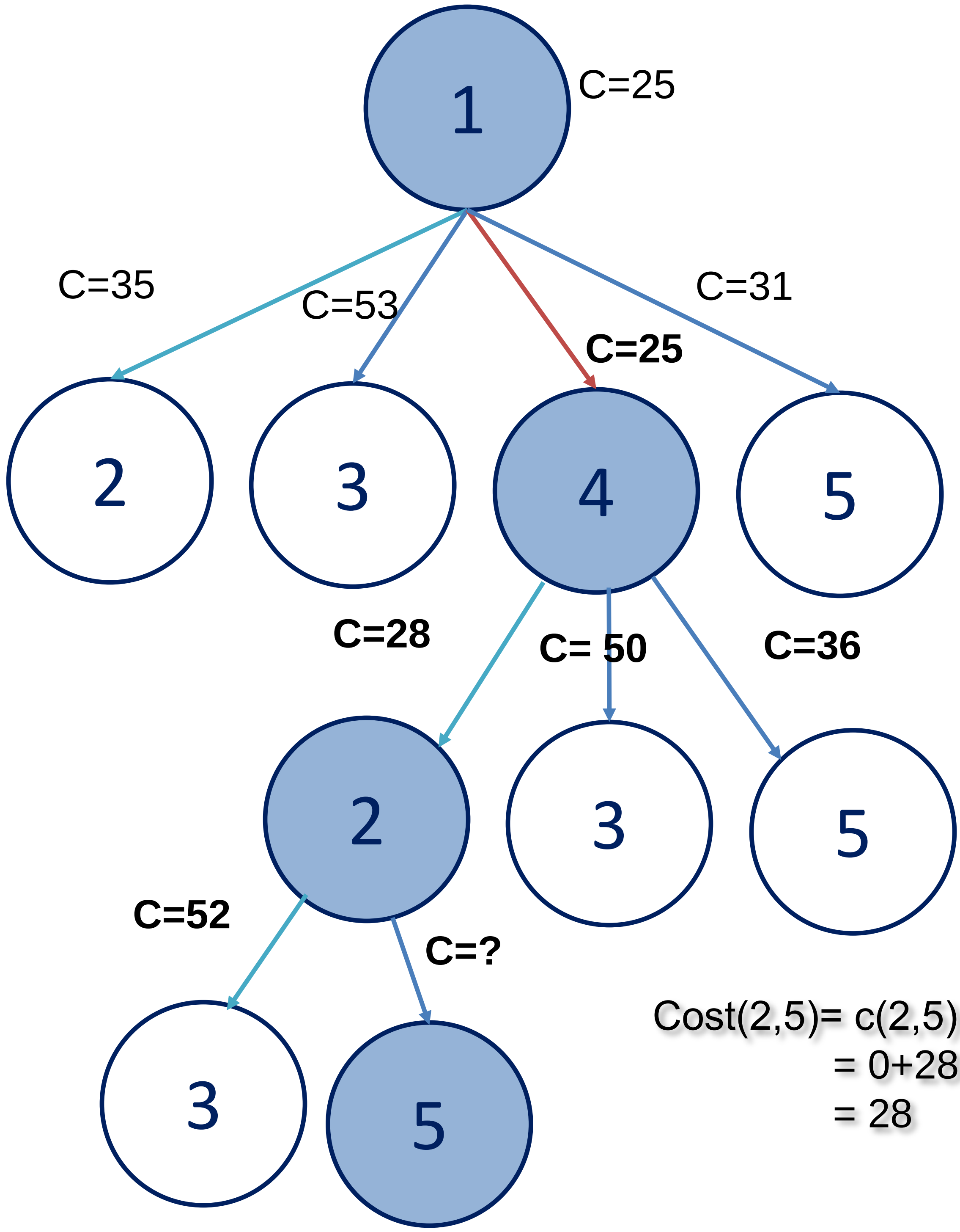
1-4-2-3

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

=13

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

Traveling Salesman Problem



Cost(2,5)= c(2,5)+C+C’
= 0+28+0
= 28

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

1-4-2-5

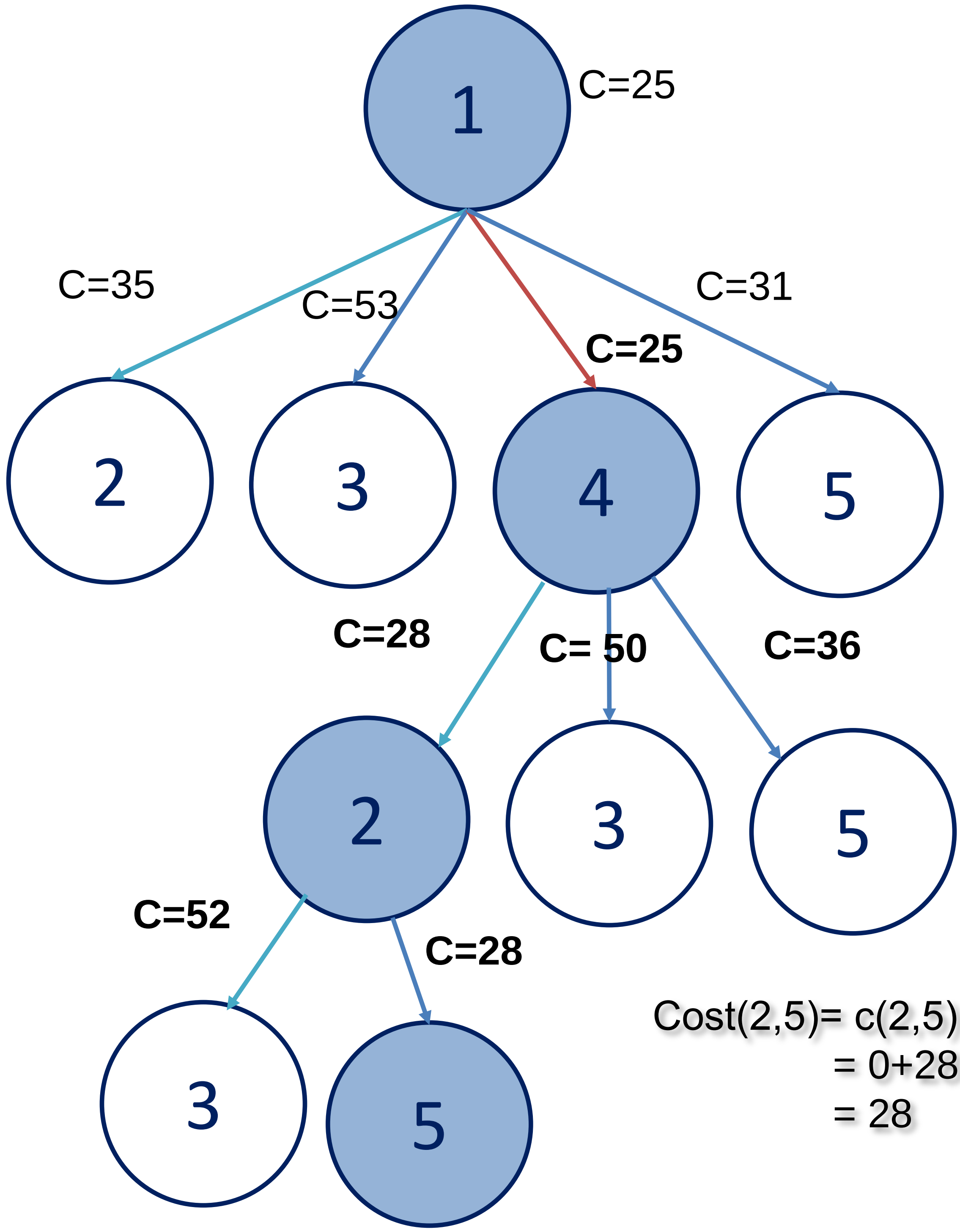
	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	∞	∞	∞	∞	∞	0
3	0	∞	∞	∞	∞	0
4	∞	∞	∞	∞	∞	0
5	∞	∞	0	∞	∞	0

=0

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

=0

Traveling Salesman Problem



Cost(2,5)= c(2,5)+C+C’
= 0+28+0
= 28

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

1-4-2-5

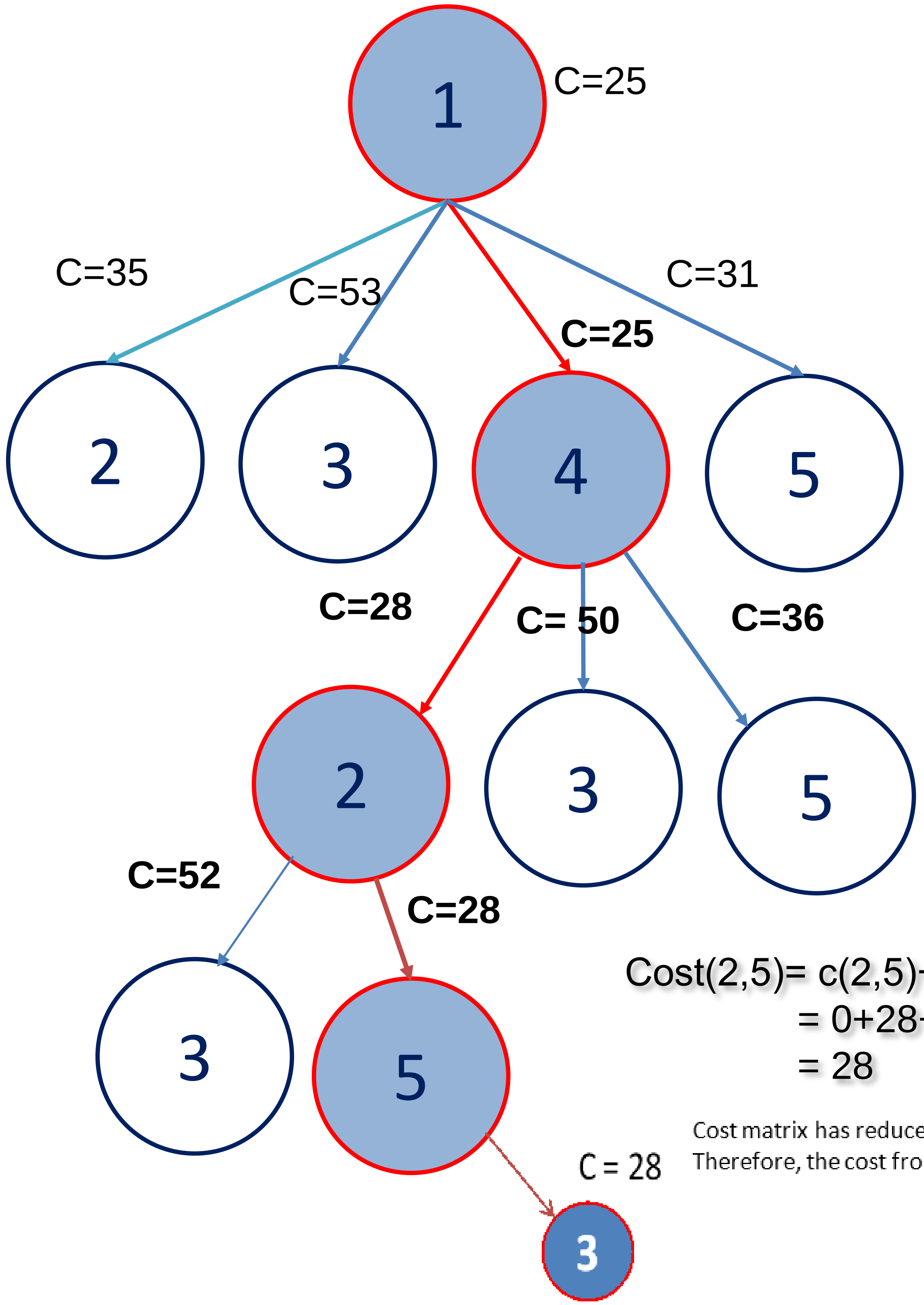
	1	2	3	4	5	
1	∞	∞	∞	∞	∞	0
2	∞	∞	∞	∞	∞	0
3	0	∞	∞	∞	∞	0
4	∞	∞	∞	∞	∞	0
5	∞	∞	0	∞	∞	0

=0

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

=0

Traveling Salesman Problem



Cost(2,5)= c(2,5)+C+C'
= 0+28+0
= 28

Cost matrix has reduced completely.
Therefore, the cost from Node 5 to Node 3 will remain same C =28

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

1-4-2-5 →

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

=0

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

=0

Total cost of traveling
sales man problem is 28

Time Complexity of Traveling Salesman Problem

- Consider city 1 as the starting and ending point. Since the route is cyclic, we can consider any point as a starting point.

- Generate all (n-1)! permutations of cities.

- Calculate the cost of every permutation and keep track of the minimum cost permutation.

- Return the permutation with minimum cost.

Total Time= O(n!) using Brute Force Approach

Total Time = O(n^2*2^n) using Branch and Bound Strategy or DP

GLA UNIVERSITY MATHURA

Accredited with A Grade by NAAC

Happy Learning!

If you have any doubts, or queries , can be discussed in the C-11, Room 310, AB-1.

or share it on WhatsApp 8586968801

if there is any suggestion or feedback about slides, please write it on gaurav.kumar@gla.ac.in

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU