Branch and Bound

Branch and Bound Approach

- Branch and bound, or BnB, is an algorithm design paradigm that is normally used to solve optimization problems.
- BnB is used for solving the minimization problems. If we have given a maximization problem, then we can solve it by simply converting the problem into a minimization problem.
- In BnB, a state space tree is generated, in which each path shows cost for some partial solution. If there will more than one path from a node, then the path with the minimum cost is selected.
- A space state tree is a tree representing all the possible states (solution or nonsolution) of the problem from the root as an initial state to the leaf as a terminal state.

Branch and Bound Approach

- Branching is the process of generating subproblems.

- Bounding refers to ignoring the partial solutions that cannot be better than the current best solution / will never generate the required solution.

- BnB eliminates those part of search space tree which does not contain better solution (pruning).

- In this method, the cheapest path is normally extended.

Advantages of BnB

- The BnB algorithm does not explore all nodes in the tree. Thereby, the time complexity is significantly lesser than brute force algorithms.

- If the branching is done reasonably, the algorithm can find the optimal solution in a reasonable period.

BnB Algorithms

- FIFO BnB algorithm (Queue)
- LIFO BnB algorithm (Stack)
- Least cost (LC) BnB algorithm (Priority queue)

- In FIFO, the state space tree is generated in BFS manner. At each level, all the paths are first explored in BFS manner and then we move to the next level.
- In LIFO, the state space tree is generated in DFS manner. At each level, one path is first explored completely in DFS manner, then other paths are explored.
- In Least cost , the state space tree is generated by expanding the cheapest path at each level.

BnB Algorithms

Sum of subset

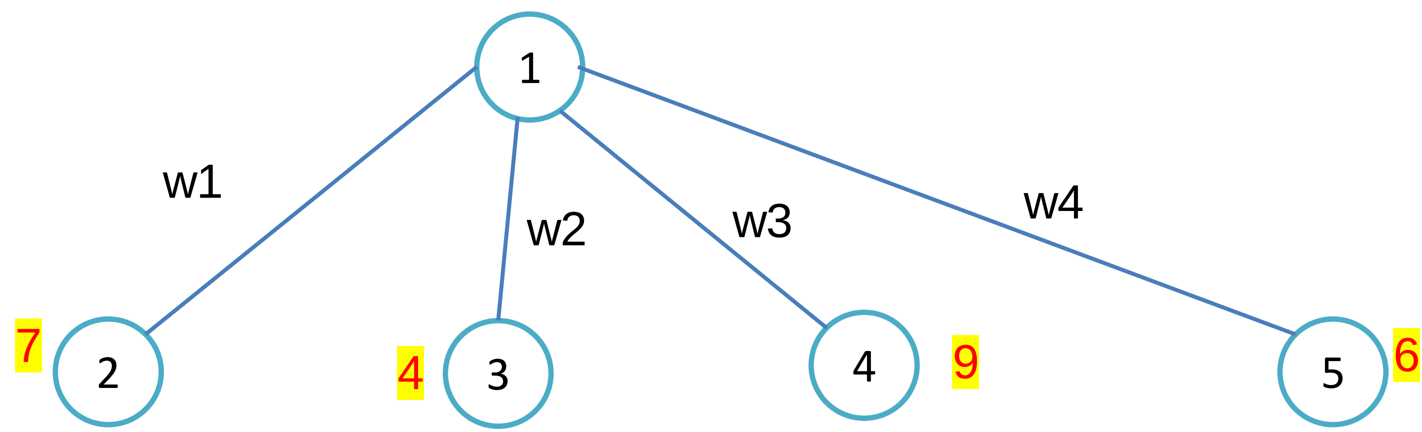
N=4, {w1, w2, w3, w4} ={7, 4, 9, 6}   sum =13

FIFO BnB

Define a queue and insert node 1 in the queue

1									

Initially subset does not contain any item. Thus, at the first level, we can take w1, w2, w3, and w4. After that, node 1 will be deleted from queue and node 2, 3, 4, and 5 will be inserted. Find the subset sum of each path.

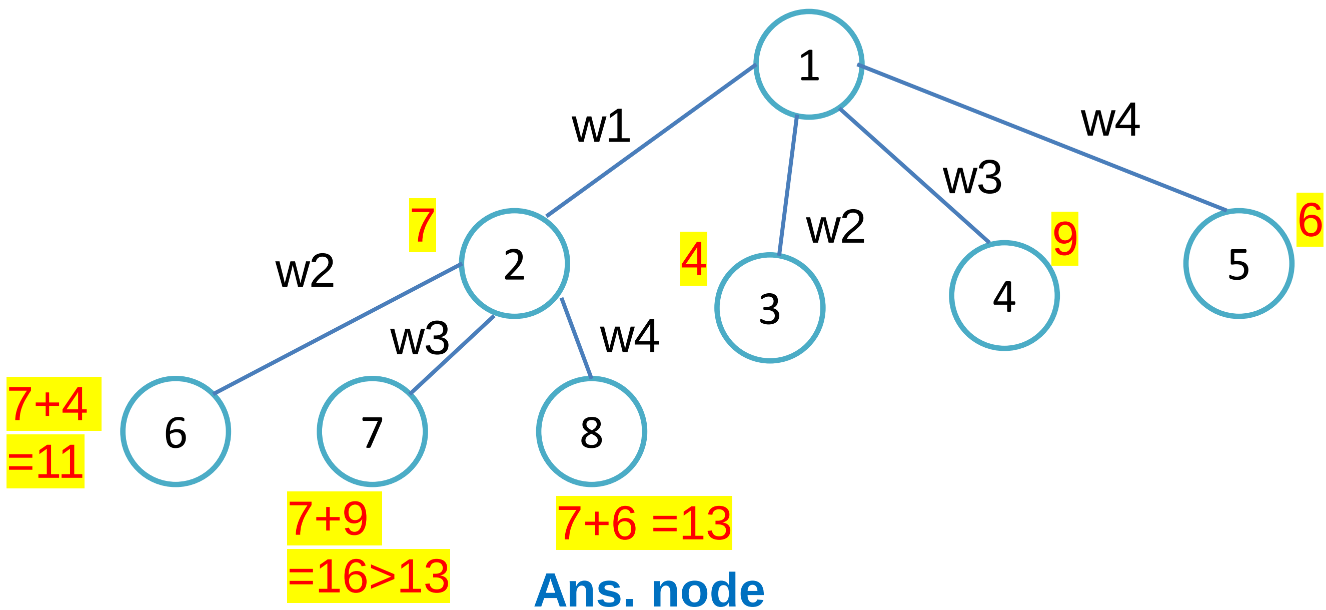


BnB Algorithms

N=4, {w1, w2, w3, w4} ={7, 4, 9, 6}   sum =13

2	3	4	5						

Explore node 2. From node 2, w1 is already taken, so it can take w2, or w3, or w4. Thus, node 6, 7, 8 will be inserted in queue.



Dead node
Need not to be explored

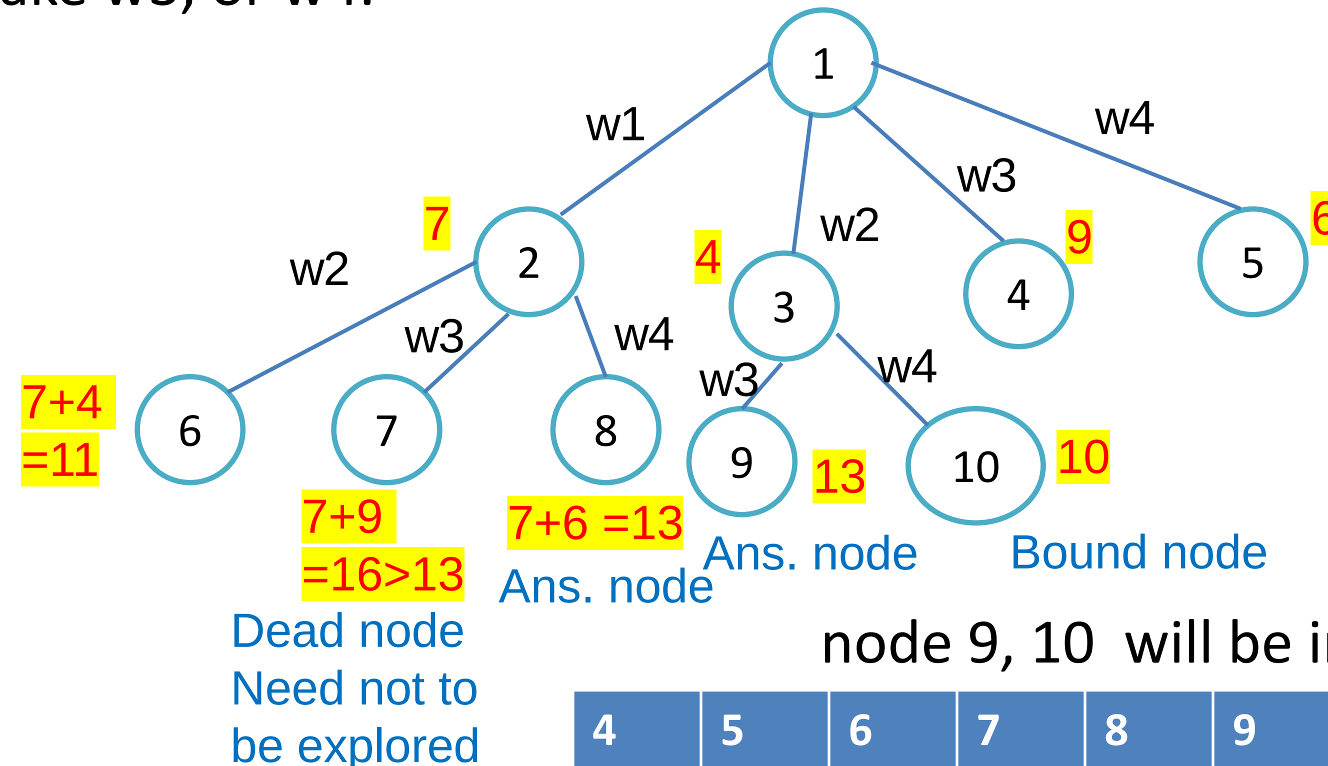
3	4	5	6	7	8				

BnB Algorithms

N=4, {w1, w2, w3, w4} ={7, 4, 9, 6}   sum =13

3	4	5	6	7	8				

Explore node 3. From node 3, w1, and w2 are already taken, so it can take w3, or w4.



node 9, 10 will be inserted in queue.

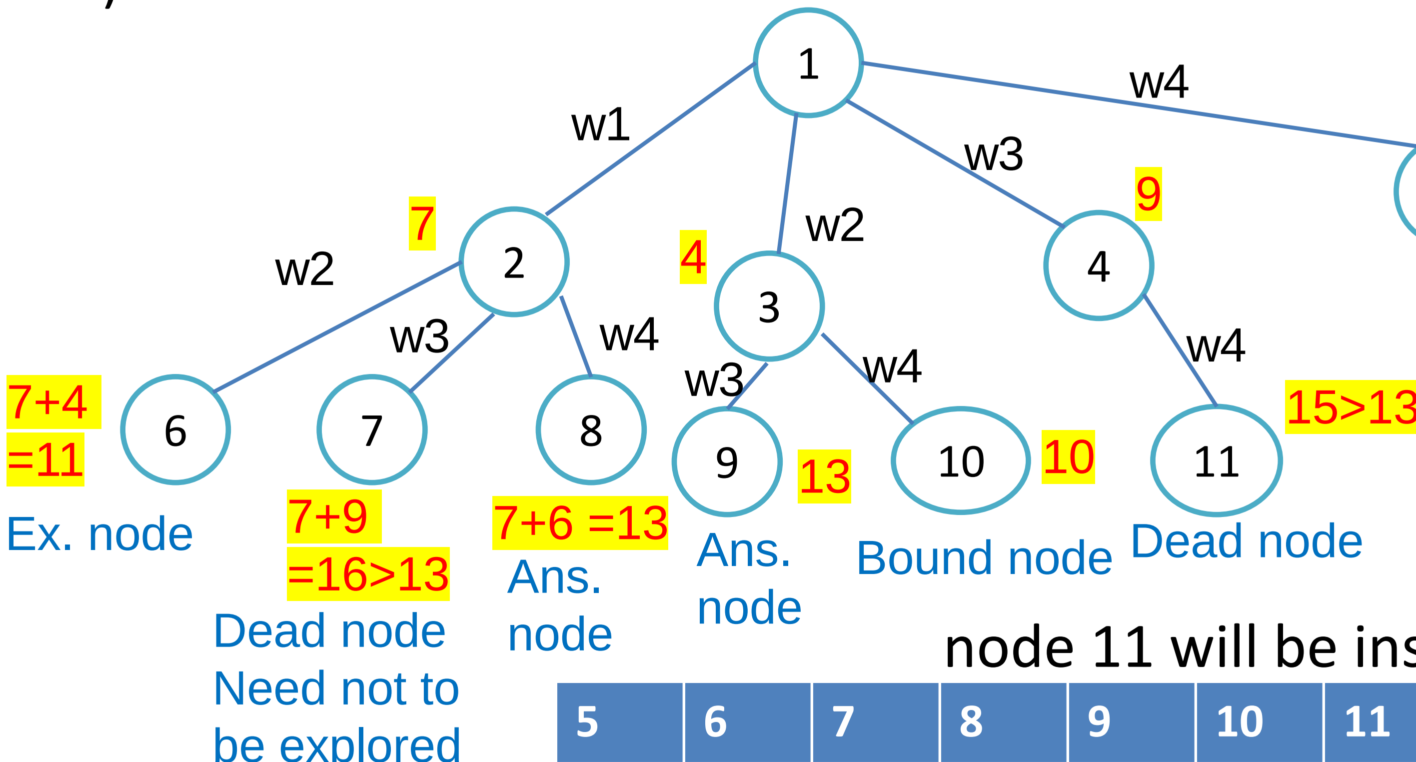
4	5	6	7	8	9	10			

BnB Algorithms

N=4, {w1, w2, w3, w4} ={7, 4, 9, 6}   sum =13

4	5	6	7	8	9	10			

Explore node 4. From node 4, it can take w4. (w1, w2 and w3 are already taken).



node 11 will be inserted in queue.

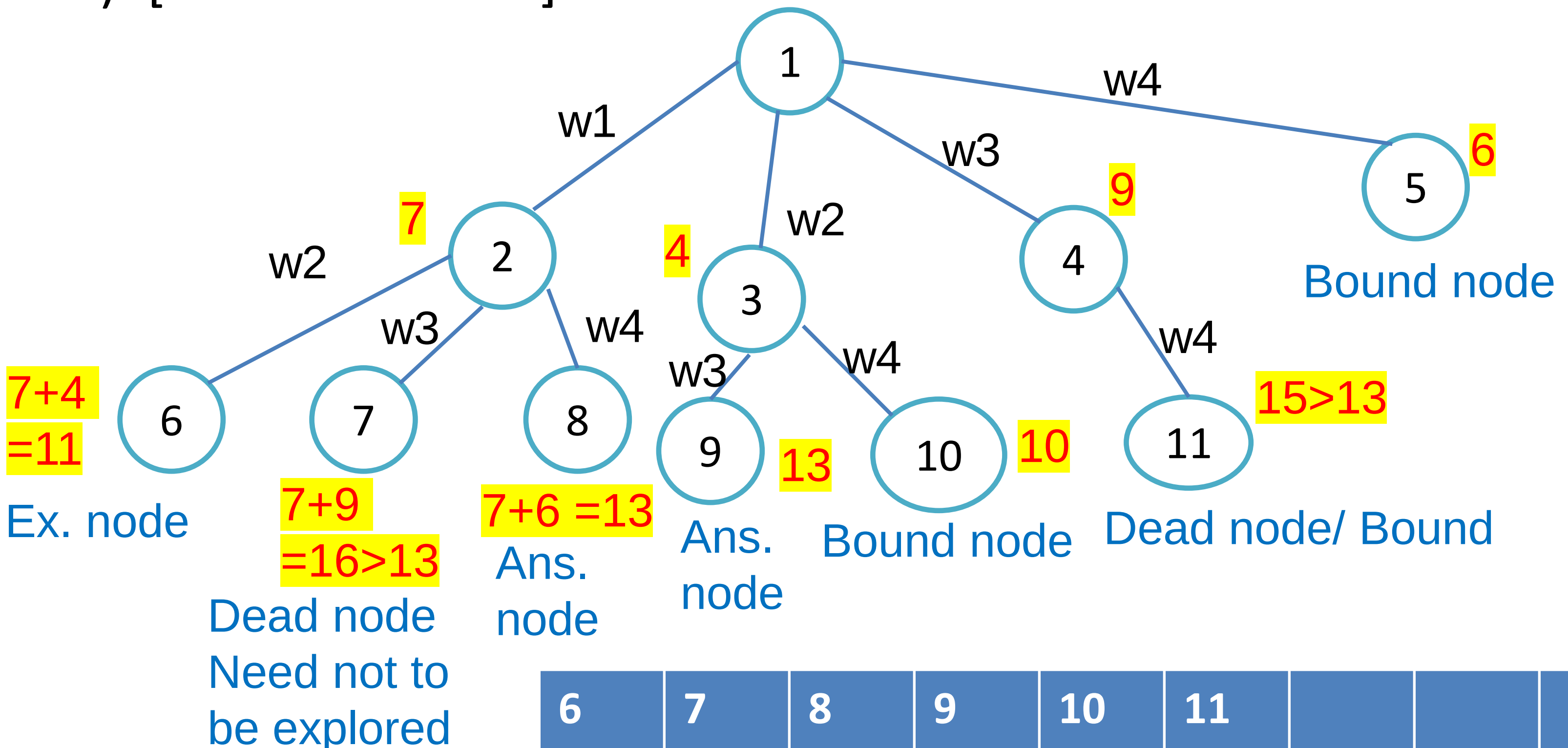
5	6	7	8	9	10	11			

BnB Algorithms

N=4, {w1, w2, w3, w4} ={7, 4, 9, 6}   sum =13

4	5	6	7	8	9	10	11		

Explore node 5. Nothing can be taken. (w1, w2, w3, and w4 are already taken). [bound function]



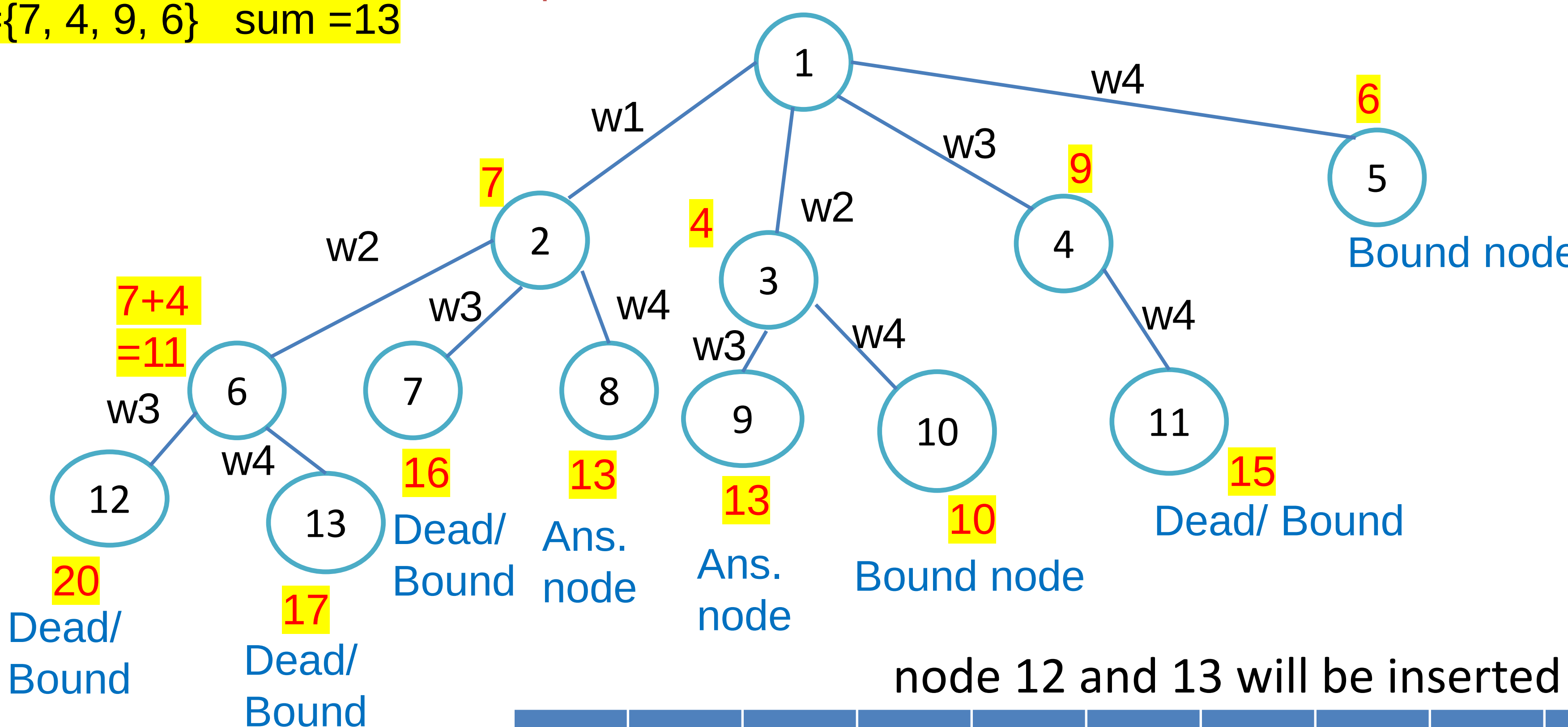
6	7	8	9	10	11				

BnB Algorithms

Node 6 will be explored since it is Ex. node. it can take w3 and w4. (w1, and w2 are already taken).

6	7	8	9	10	11	12	13		

N=4, {w1, w2, w3, w4} ={7, 4, 9, 6} sum =13



node 12 and 13 will be inserted

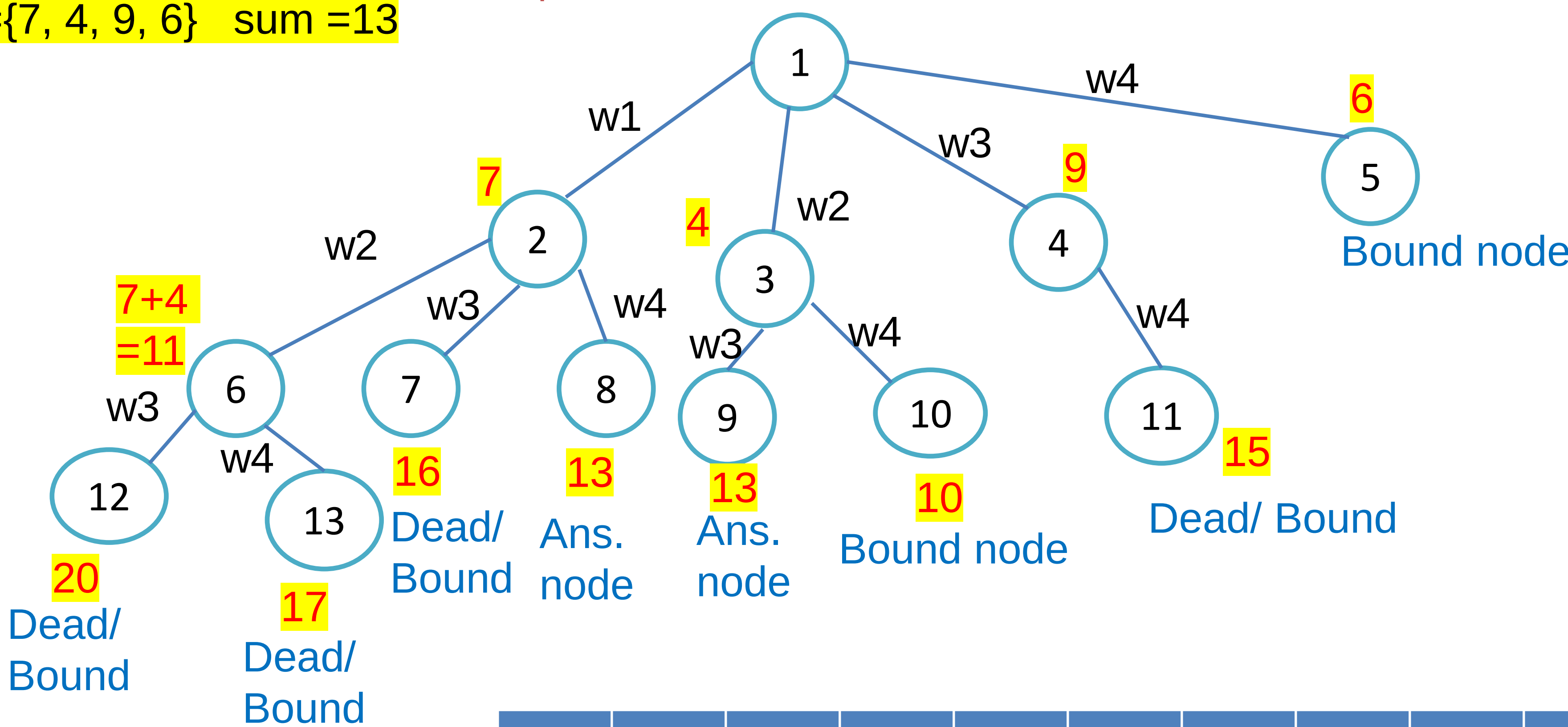
7	8	9	10	11	12	13			

BnB Algorithms

Explore node 7. Dead end, need not to be explored.

7	8	9	10	11	12	13			

N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13



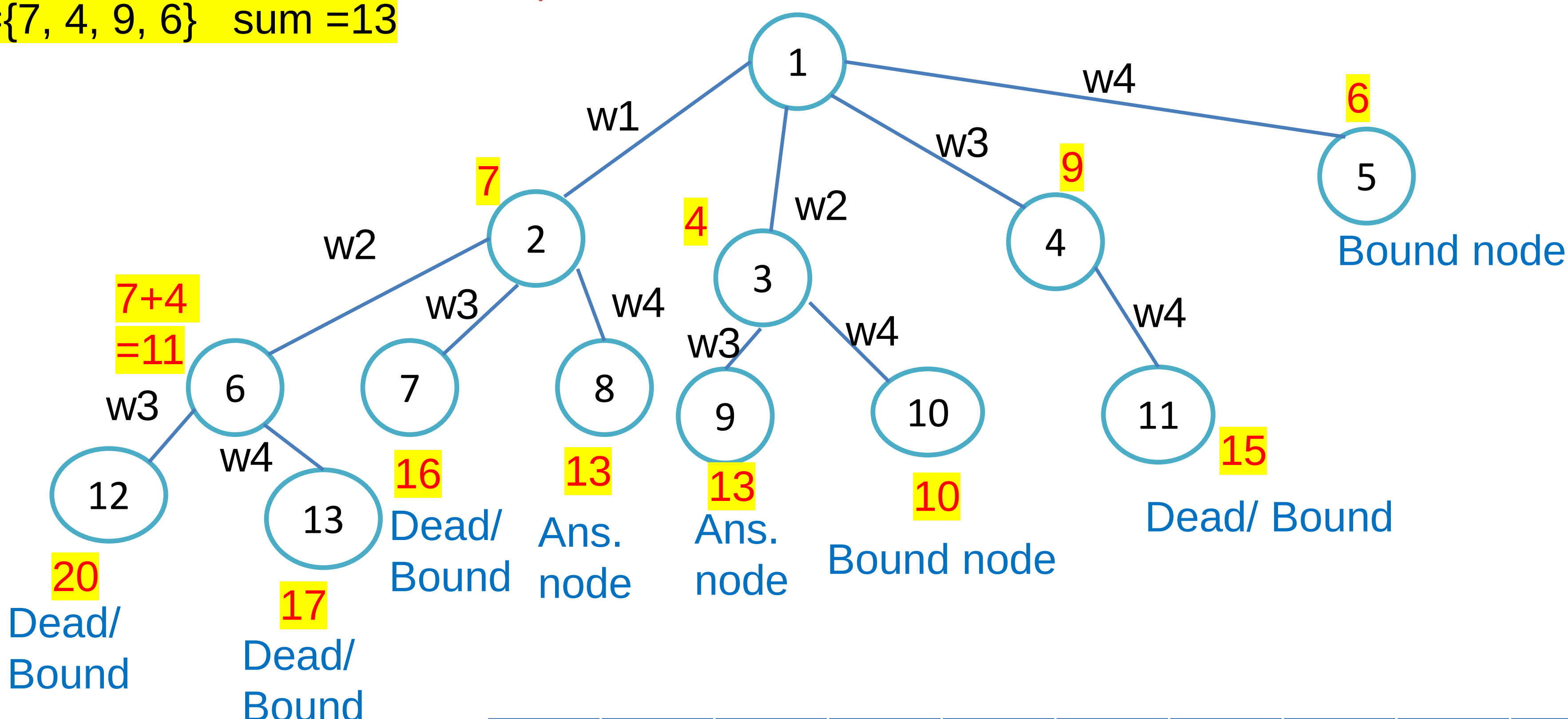
8	9	10	11	12	13	14			

BnB Algorithms

Node 8 and 9 cannot be explored since they are answers node.

N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13

8	9	10	11	12	13				

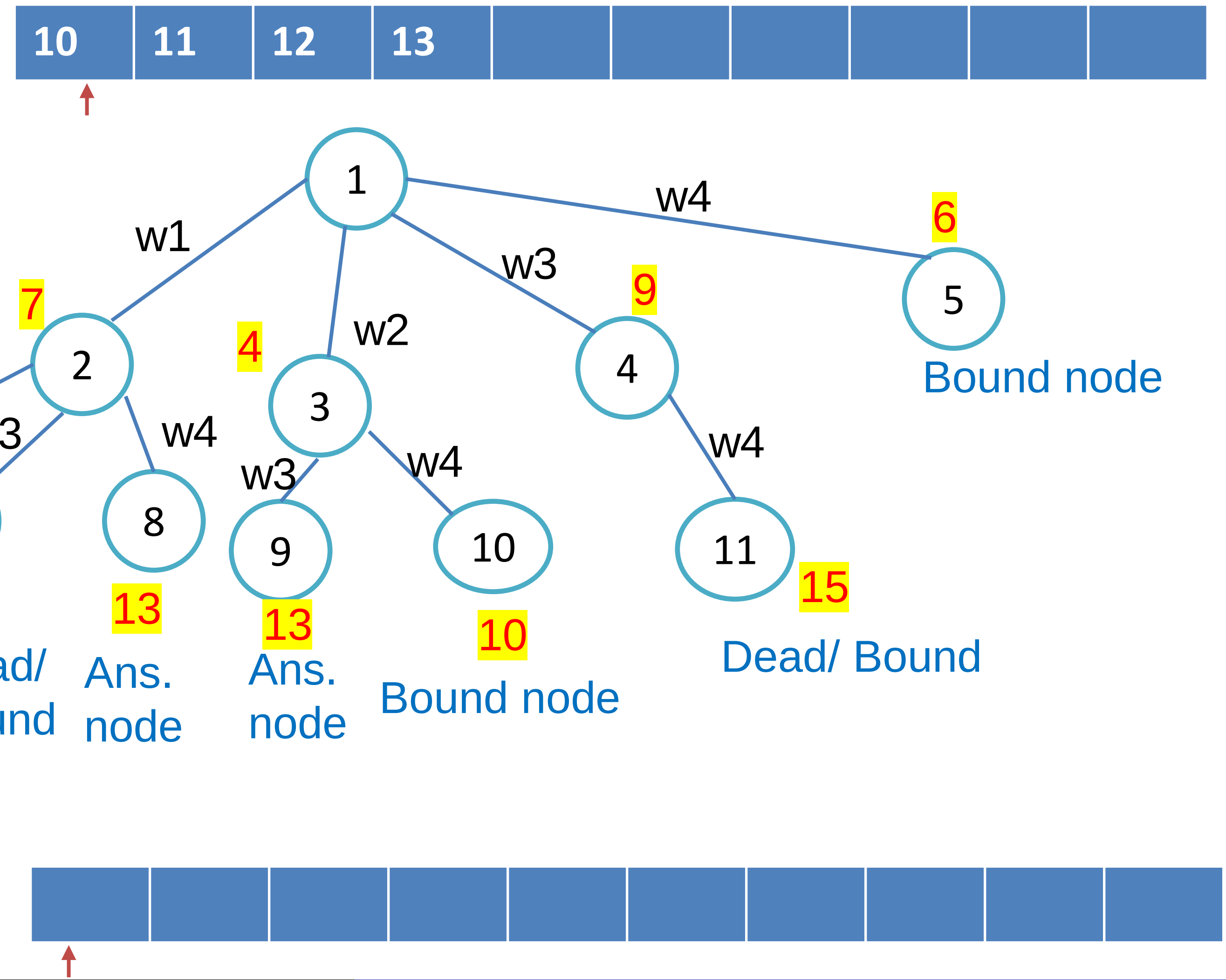


10	11	12	13						

BnB Algorithms

Node 10, 11, 12, and 13 cannot be explored since they are bound node.

N=4, {w1, w2, w3, w4} ={7, 4, 9, 6} sum =13

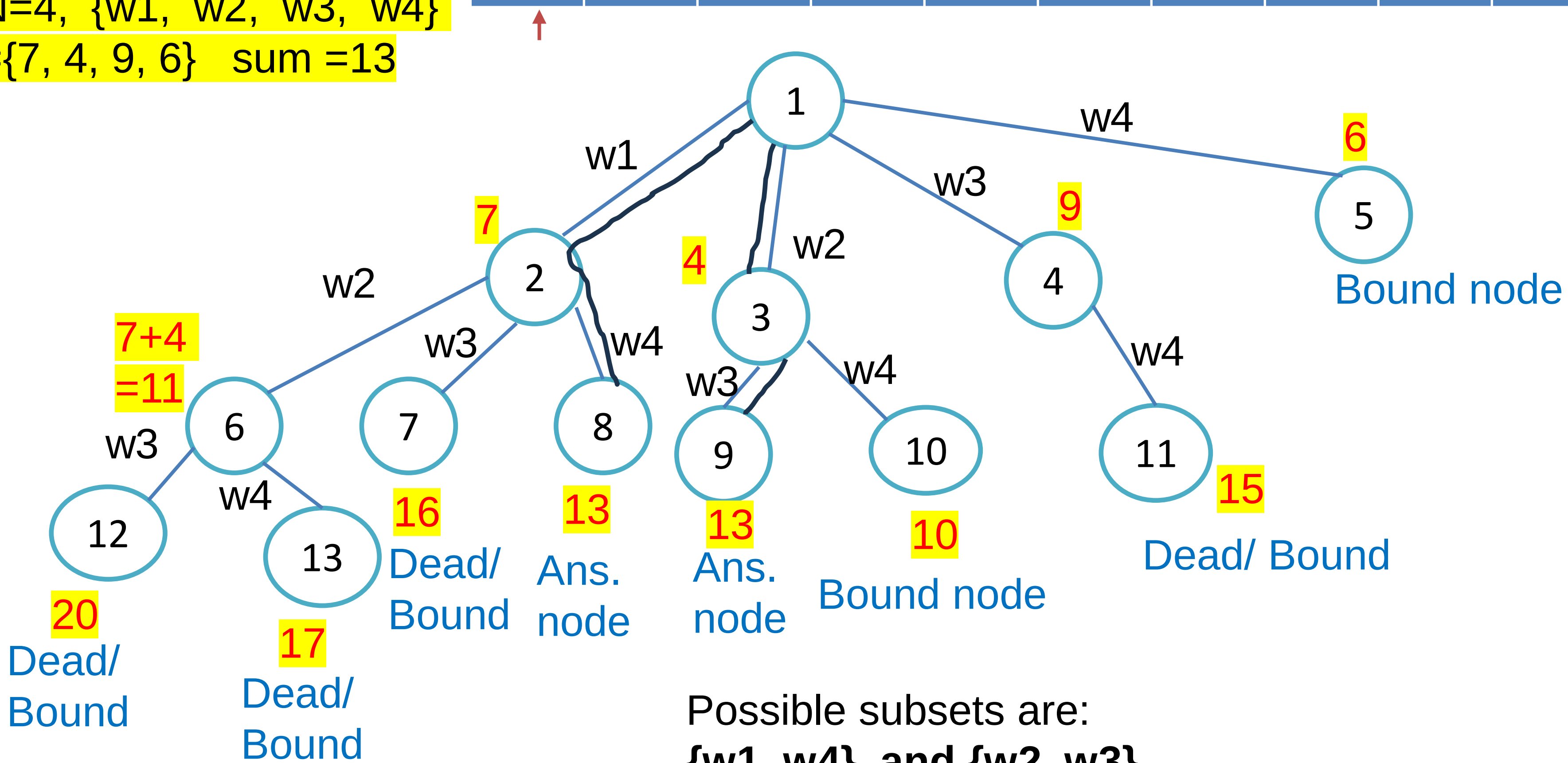


BnB Algorithms

Node 10, 11, 12, and 13 cannot be explored since they are bound node.

10	11	12	13						

N=4, {w1, w2, w3, w4} ={7, 4, 9, 6} sum =13



Possible subsets are:
{w1, w4}, and {w2, w3}

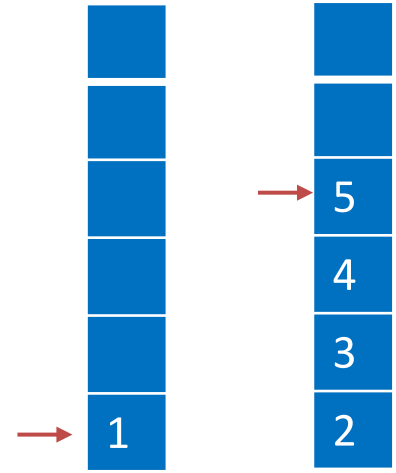
BnB Algorithms

Sum of subset

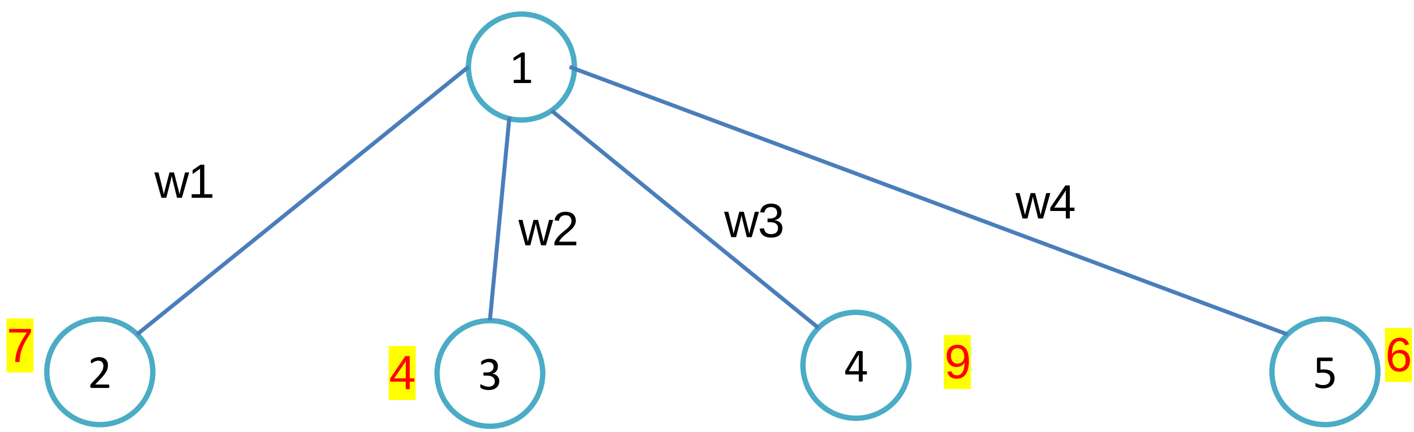
N=4, {w1, w2, w3, w4} ={7, 4, 9, 6}   sum =13

LIFO BnB

Define a stack and push node 1



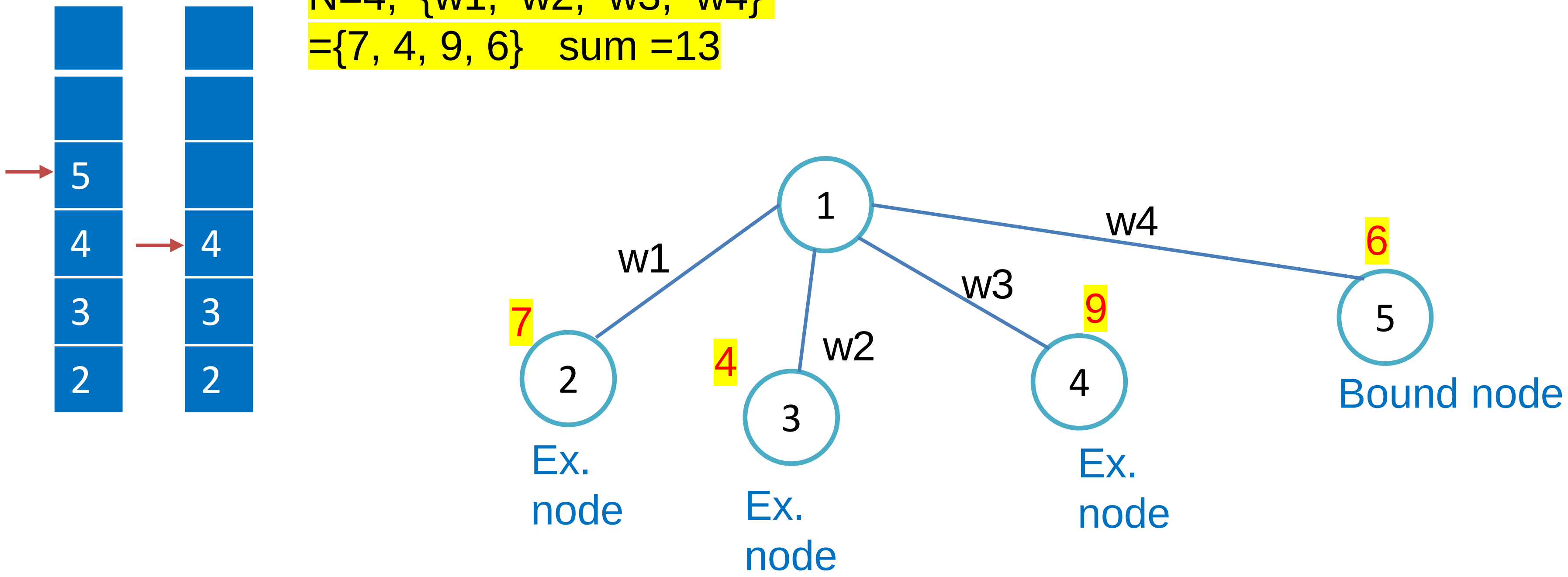
At the first level, we can take w1, w2, w3, and w4. After that, node 1 will be popped from stack and node 2, 3, 4, and 5 will be pushed in same order. Find the **subset sum** of each path.



BnB Algorithms

Explore node 5 because it is at the top of the stack. We cannot explore 5, since it is bound node, just pop 5 from stack.

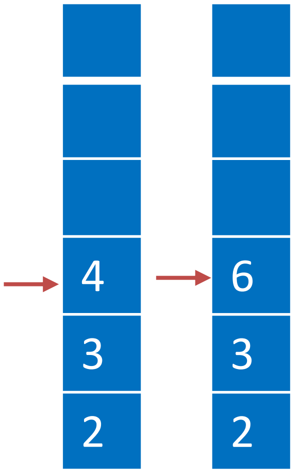
N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13

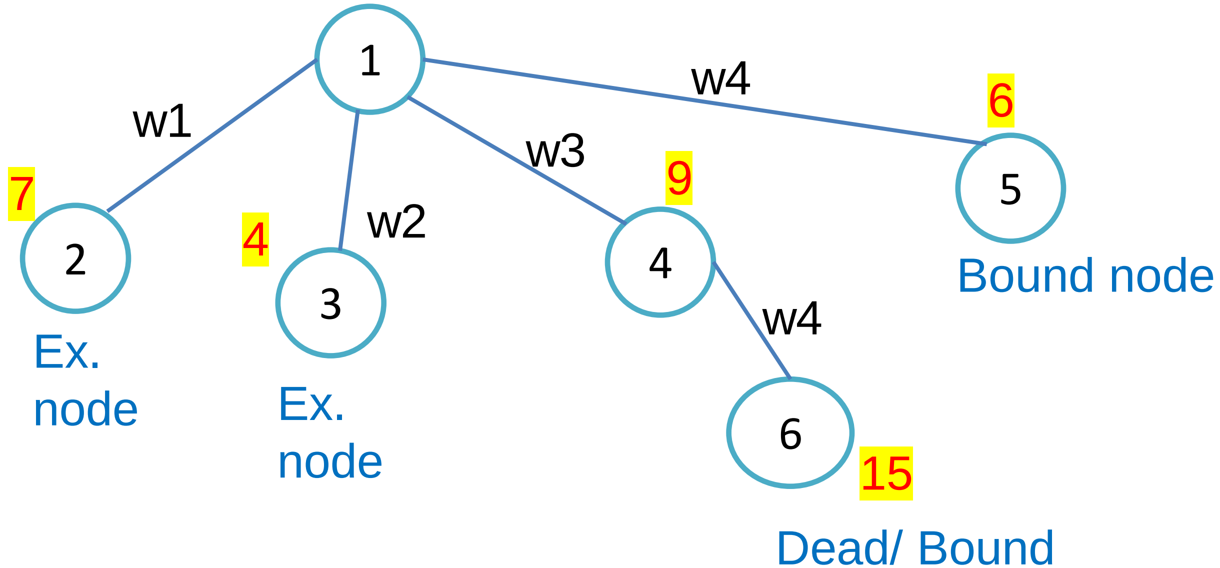


BnB Algorithms

Explore node 4, w4 can be taken. So, pop 4 from stack and push node 6.

N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13

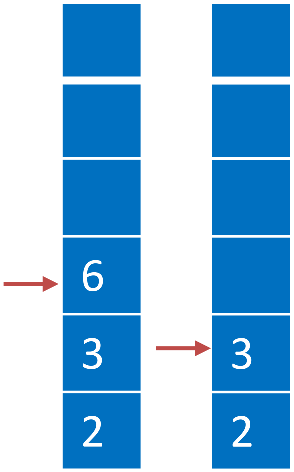


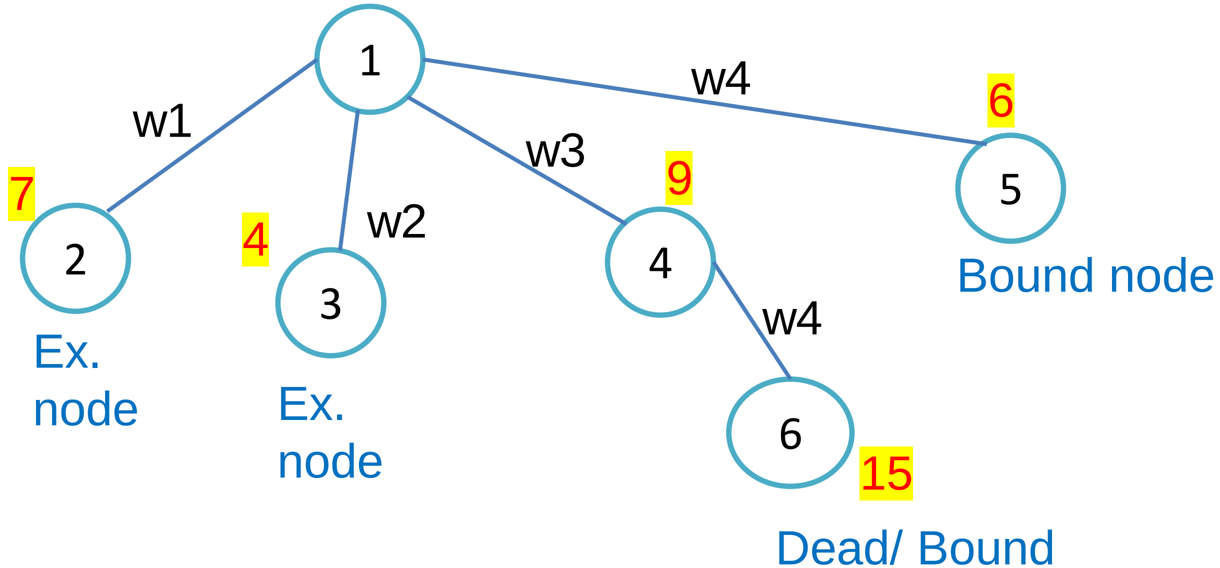


BnB Algorithms

Node 6 cannot be explored since it is a dead/ bound node. So, pop 6 from stack.

N=4, {w1, w2, w3, w4} ={7, 4, 9, 6} sum =13

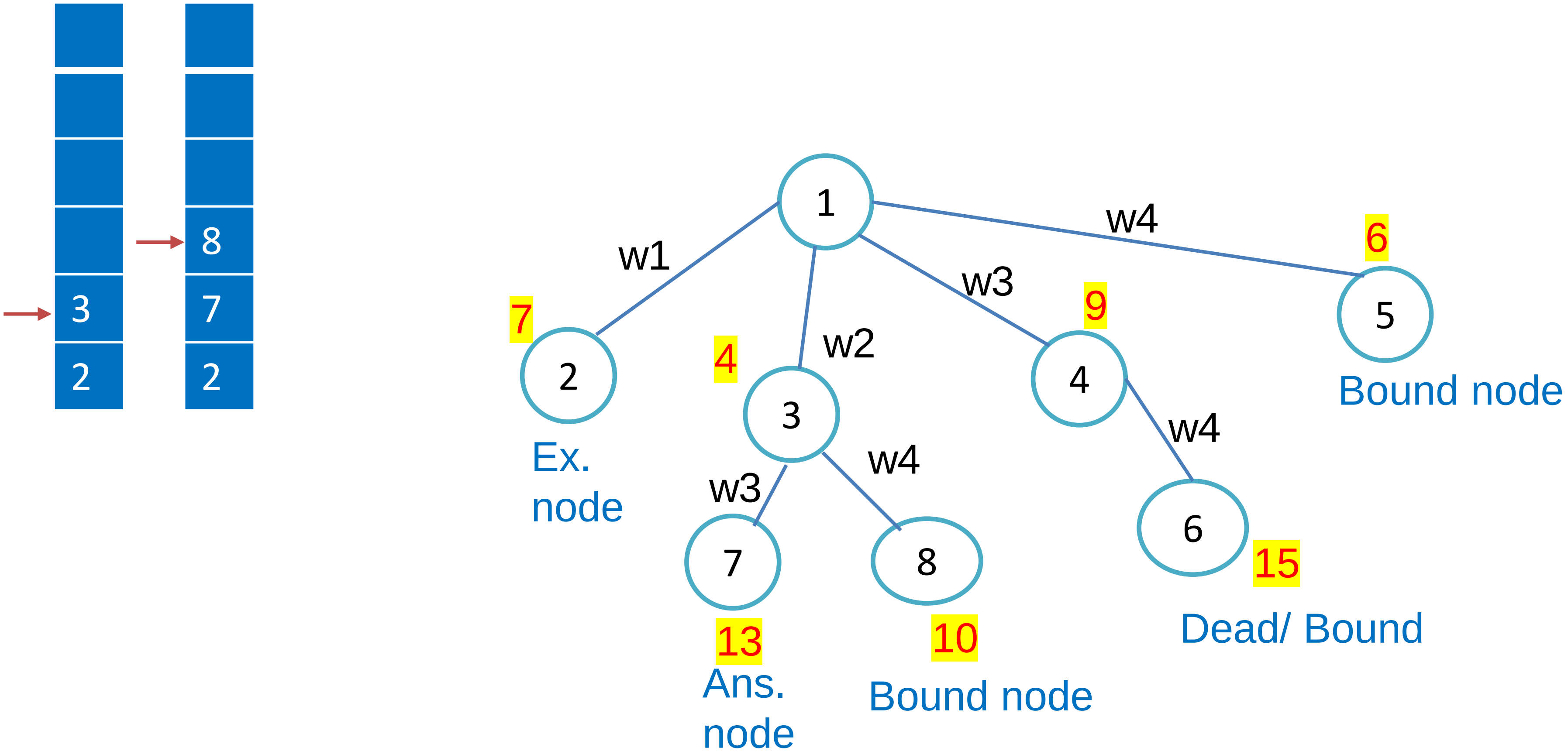




BnB Algorithms

Explore Node 3. w3 or w4 can be taken. So, pop 3 and push node 7 and 8 into stack.

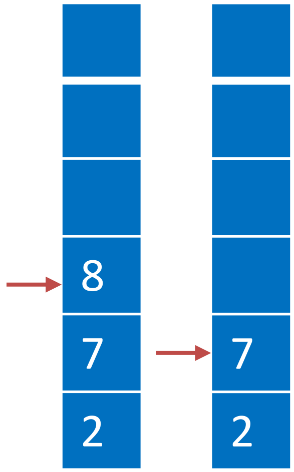
N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13

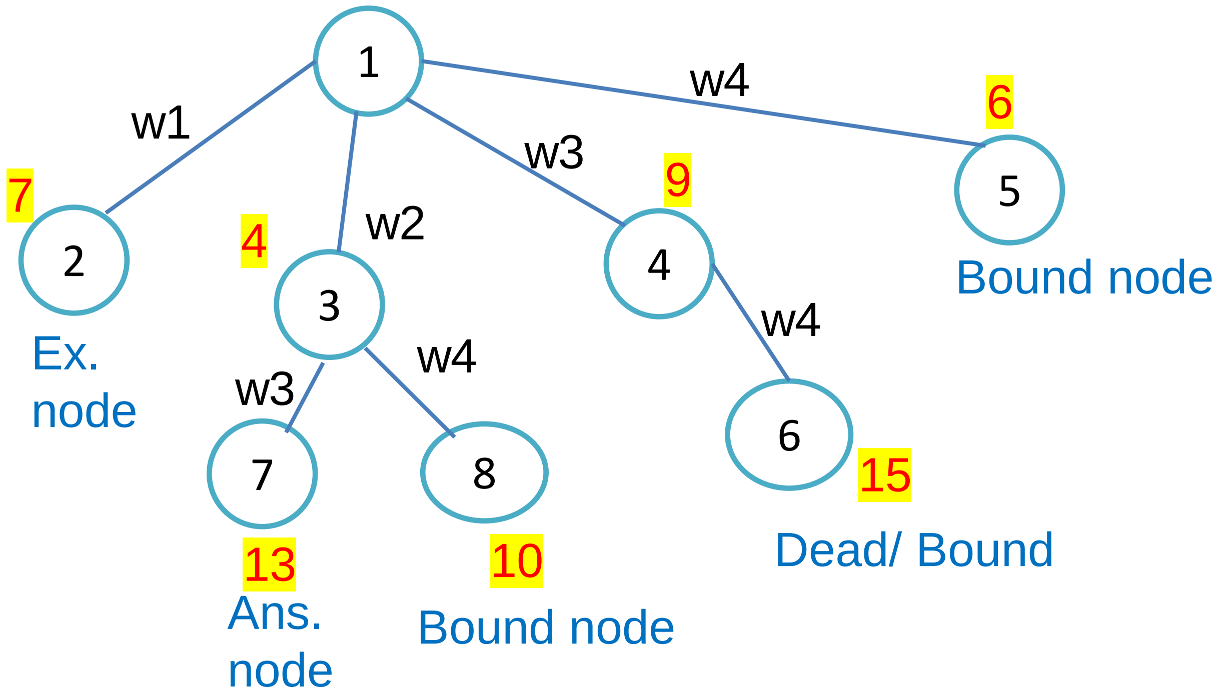


BnB Algorithms

Node 8 cannot be explored. So, just pop 8 from stack.

N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13

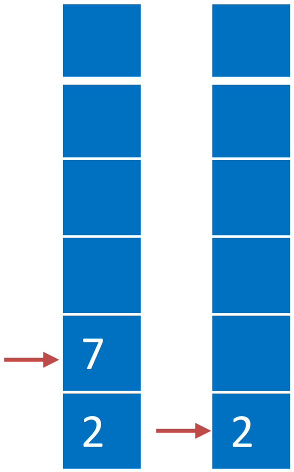


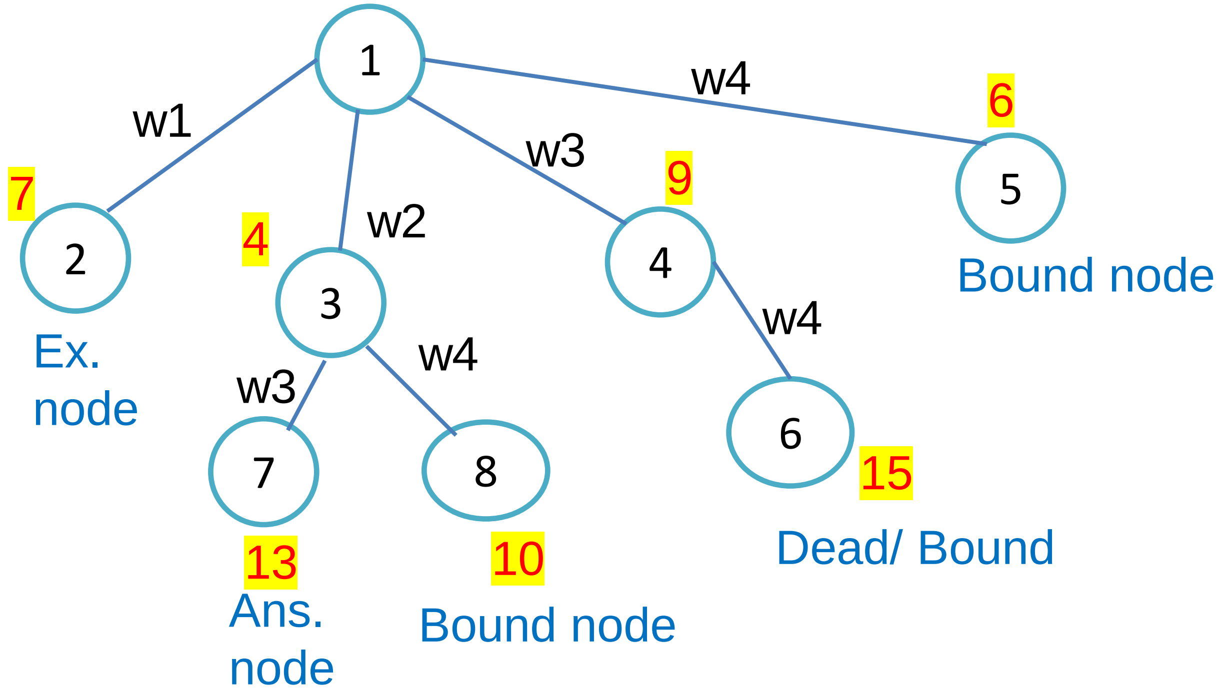


BnB Algorithms

Node 7 cannot be explored since it is answer node. So, just pop 7 from stack.

N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13

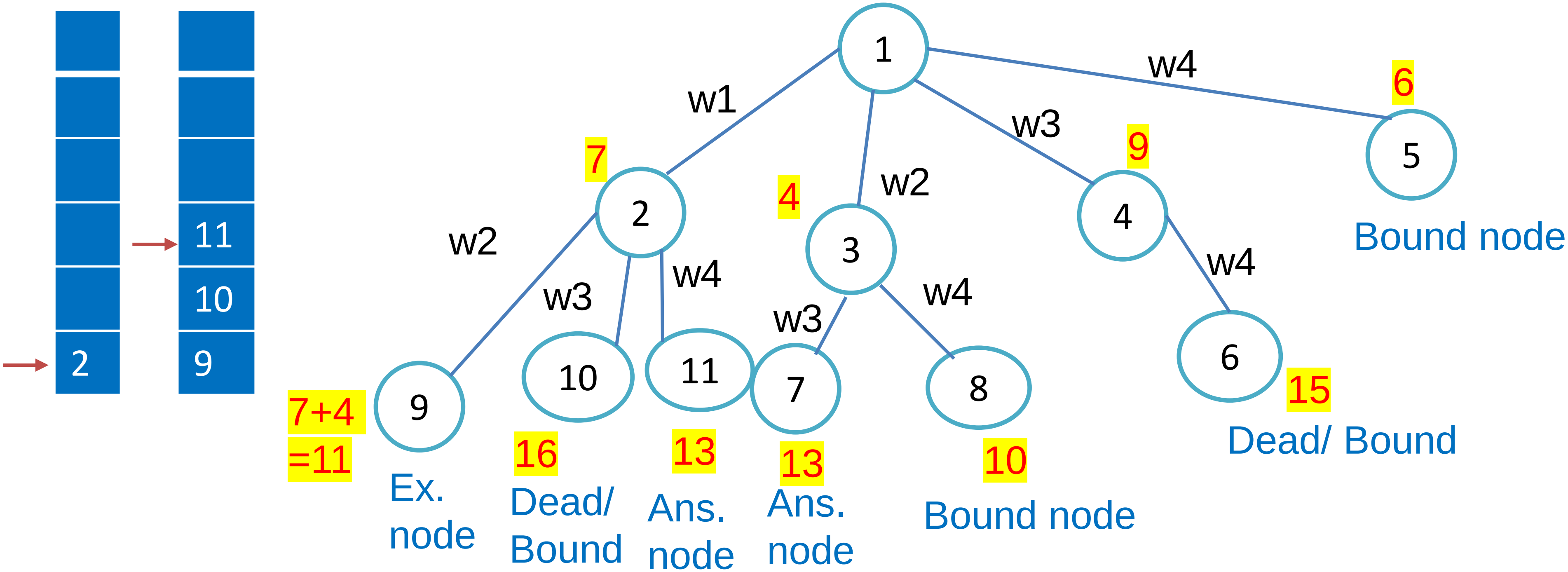




BnB Algorithms

Explore Node 2, w2, or w3, or w4 can be taken. So, pop 2 from stack and push node 9, 10, and 11 into stack.

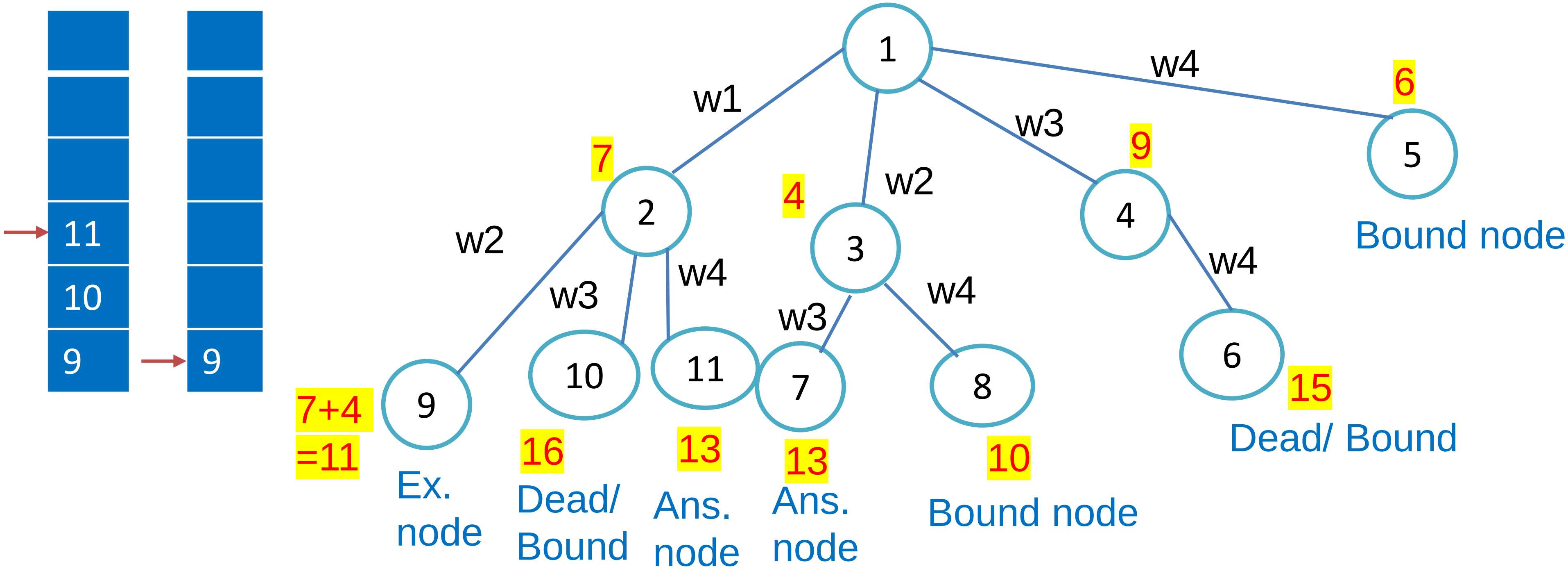
N=4, {w1, w2, w3, w4} ={7, 4, 9, 6} sum =13



BnB Algorithms

Node 11 and 10 cannot be explored. So, pop 11 followed by 10 from stack.

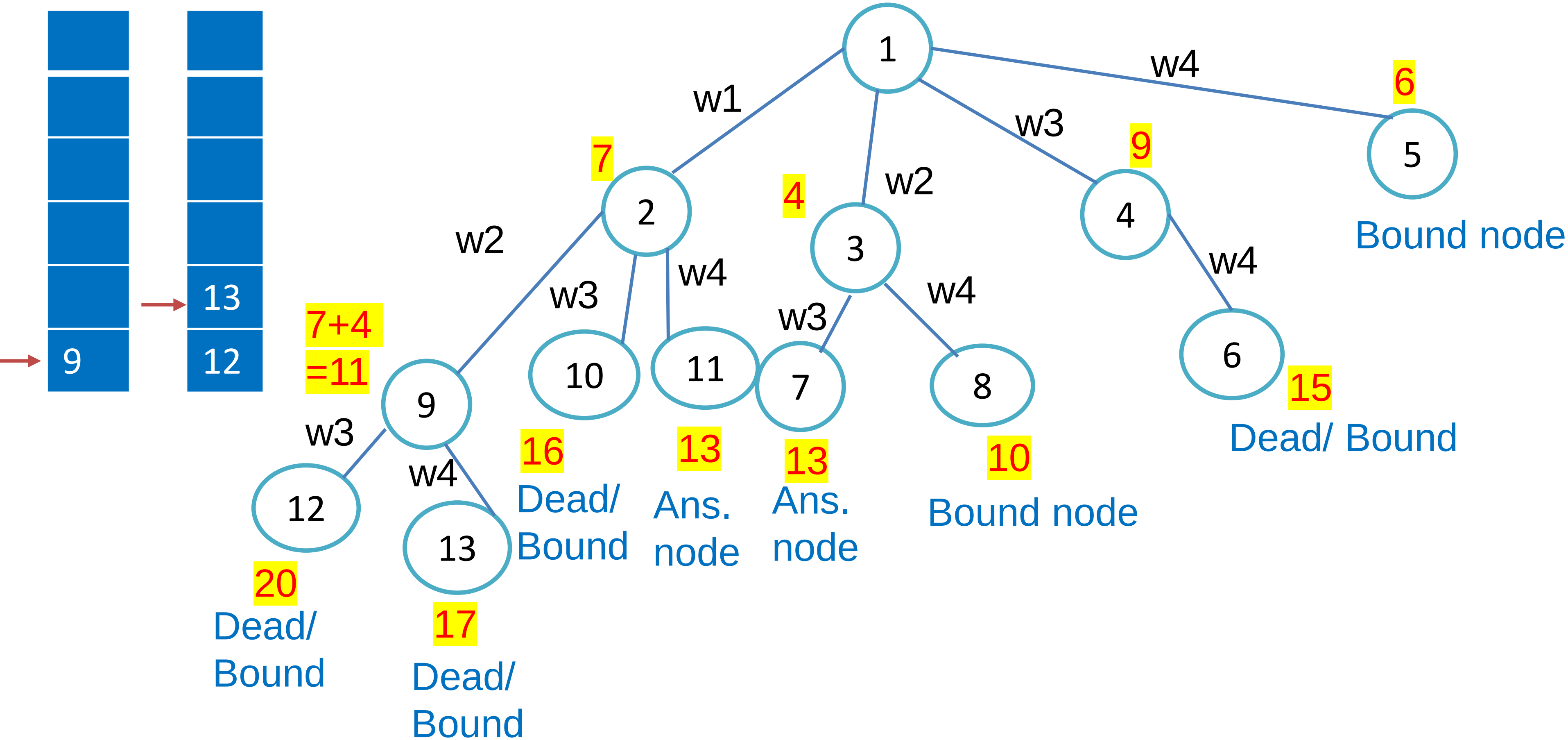
N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13



BnB Algorithms

Explore Node 9, w3, or w4 can be taken. So, pop 9 and push node 12 and 13.

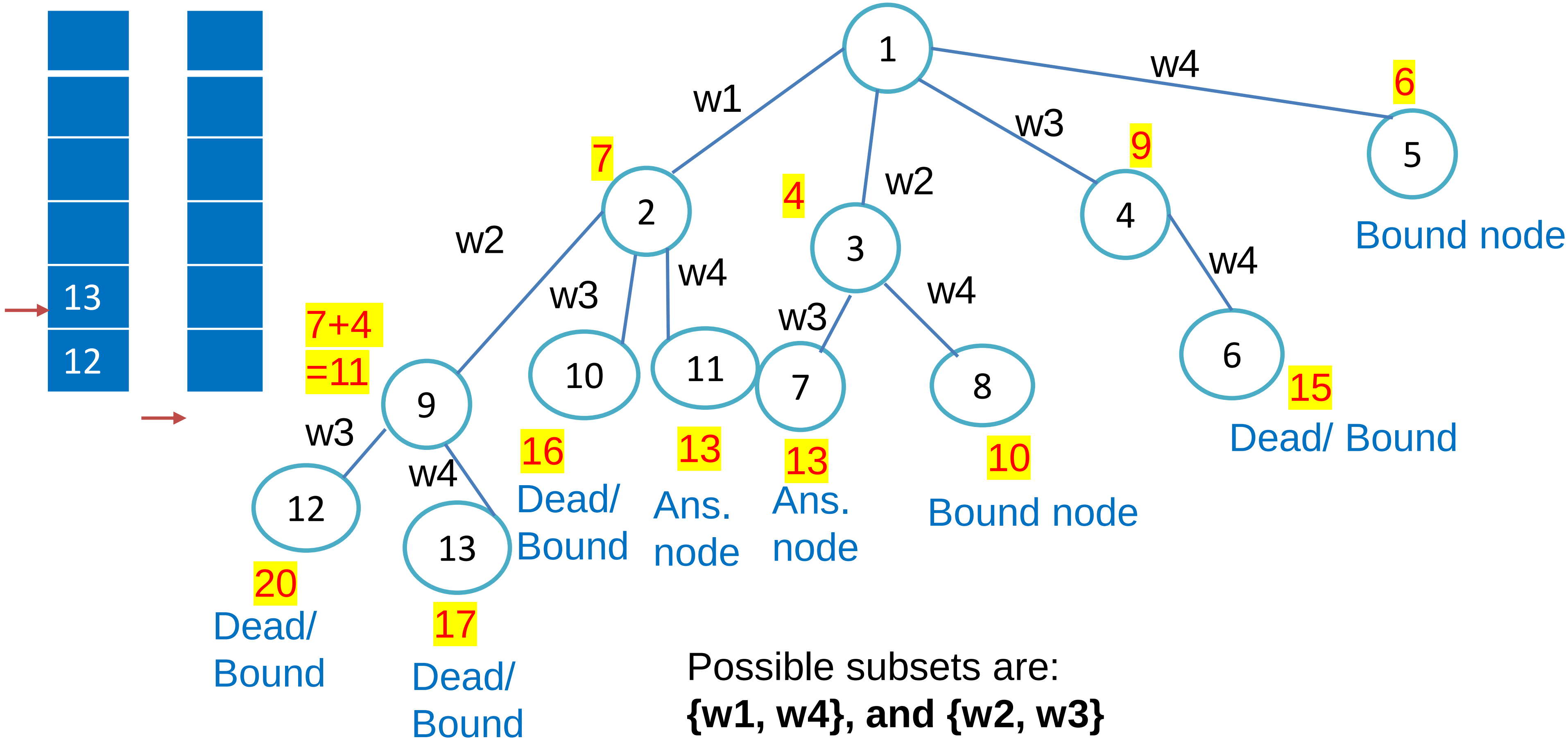
N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13



BnB Algorithms

Node 13 and 12 cannot be explored. So, pop 13 followed by 12 from stack.

N=4, {w1, w2, w3, w4}
={7, 4, 9, 6} sum =13

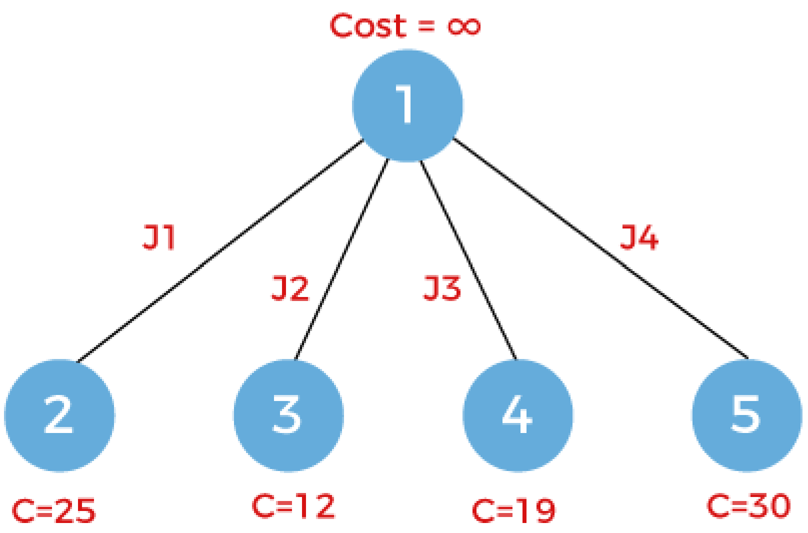


Possible subsets are:
{w1, w4}, and {w2, w3}

BnB Algorithms

Least cost BnB algorithm (Priority queue)

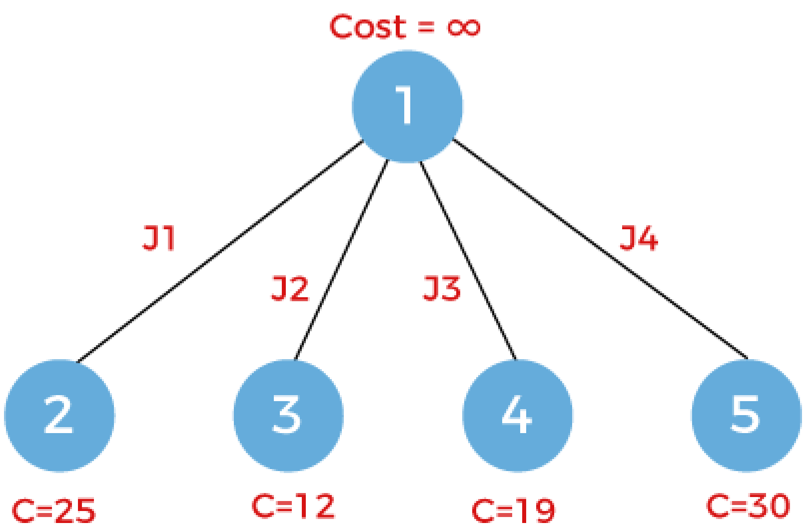
- In Least cost , the state space tree is generated by expanding the cheapest path. It is normally used to solve minimization optimization problems.
- For the following, the cost for the node 3 is the cheapest, so next time this node will be expanded.

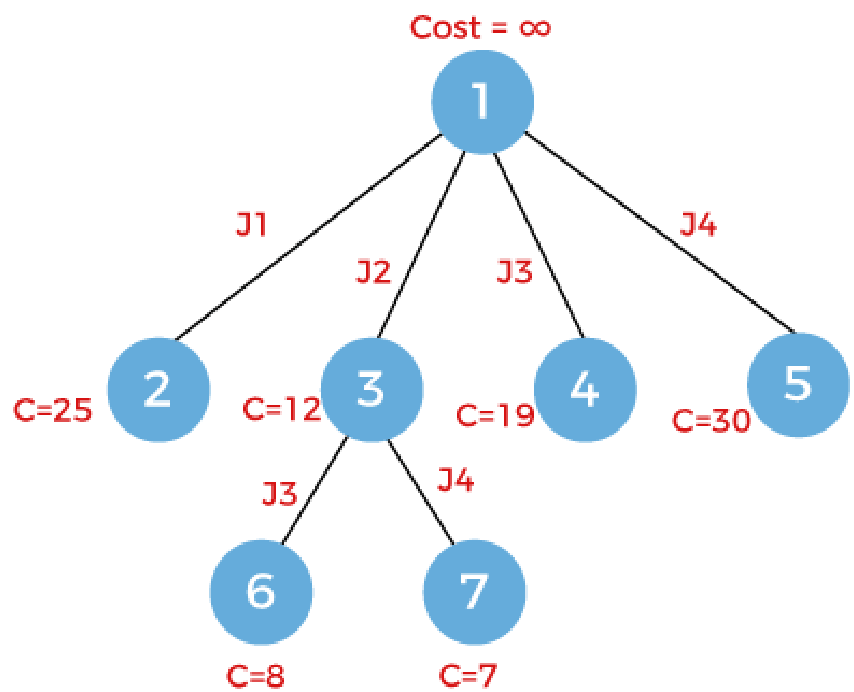


BnB Algorithms

Least cost BnB algorithm (Priority queue)

- For the following, the cost for the node 3 is the cheapest, so next time this node will be explored.





0/1 Knapsack Problem

We are a given a set of *n* objects, each have a value *vi* and a weight *wi*. The objective of the *0/1* Knapsack problem is to find a subset of objects such that the total value is maximized, and the sum of weights of the objects does not exceed a given threshold *W*.

The goal of knapsack problem is to maximize total value/profit, while BnB is generally used to solve minimization problem. So, first, the problem is converted to minimization and then it is solved.

Define upper bound (u) and cost (c) for each node in the state space tree using the following equation.

$\boxed{u = \sum vixi < W \text{ (for } i = 1,2, \ldots n)}$ and $\boxed{c = \sum vixi \text{ (for } i = 1,2, \ldots n) \text{ with fraction}}$

The initial value of upper bound is set to ∞, which is modified in each iteration using vixi (where vi denotes the value/profit of that objects, and xi is a binary value, which indicates whether the object is to be included or not).

0/1 Knapsack Problem

Solve the following instance of knapsack using LC-BnB.
n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15.

First find the initial values of u and c
The upper bound (u) = sum of all the profits for which total weight is less than or equal to knapsack capacity.

$$u = \sum_{i}^{n} ViXi \quad \text{and } \sum w <= W$$

$$c = \sum_{i}^{n} ViXi \quad \text{(with fraction)}$$

Fraction is only used for generation of the state-space tree.

First set initial value of u= ∞, and then compute values of u and c for next iterations.

For first node
u= 10+10+12 =32,
remaining capacity of knapsack w= 15-(2+4+6) = 3, which is less than the weight of fourth item. So, item 4 cannot be kept in the knapsack. **u= -32,** [for minimization problem]
Compute c (with fraction)
c= 10+10+12+18*3/9 =38, **c= -38**

Let us generate state-space tree.

0/1 Knapsack Problem

Please note that a node will be killed or not explored further if the cost c of that node is greater than the upper bound (u).

Any node will only be explored if the cost c of that node is smaller or equal to the upper bound (u).

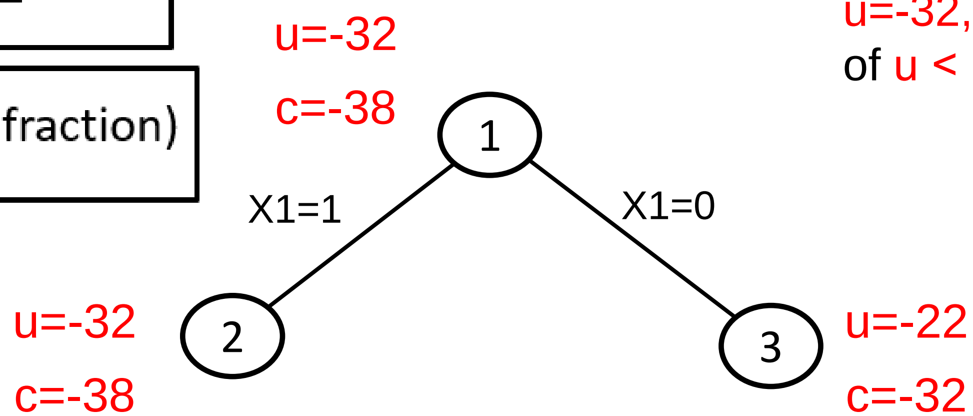
0/1 Knapsack Problem

Solve the following instance of knapsack using LC-BnB.
n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15.

$u = \sum_{i}^{n} ViXi$ and $\sum w <= W$

$c = \sum_{i}^{n} ViXi$ (with fraction)

initial value of u= ∞
For first node
u=-32, and c=-38, New value of u < ∞. So, change it.



When, we include first object, u and c will not change, since it is already present in u and c.
if first object is not included, then
u= -(10+12) = -22, and 15-(4+6)<15 => 5<15
c= -(10+12+18*5/9) => -32

u= -22, c= -32, here, cost c=-32 which is smaller than the upper bound u=-22, so keep this node will not be killed.

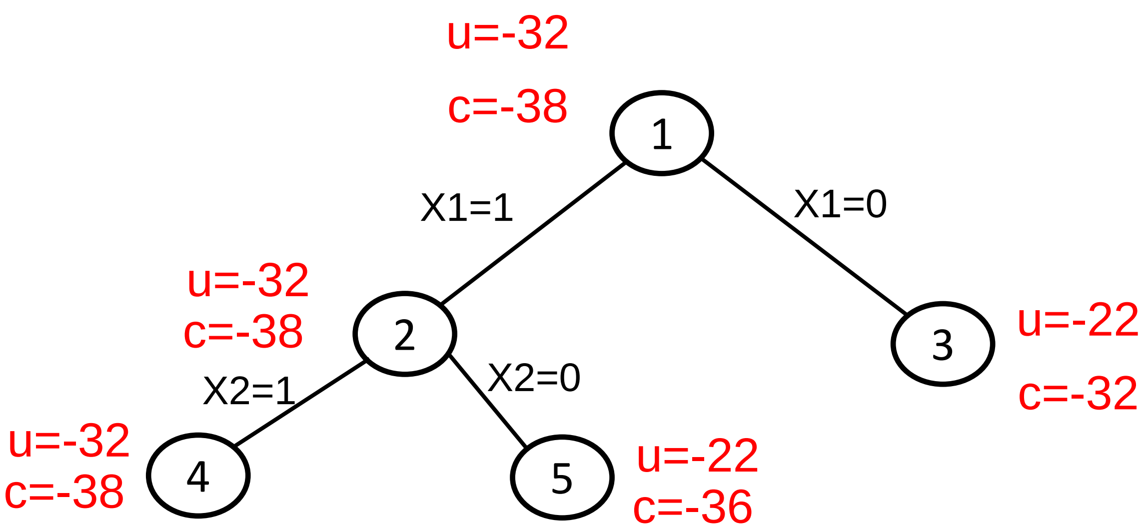
Now, which branch/node of the tree will be explored?
For **LC-BNB**, the node having least **cost c** will be explored, i.e., **node 2**.

0/1 Knapsack Problem

Solve the following instance of knapsack using LC-BnB.
n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15.

$$u = \sum_{i}^{n} ViXi \quad \text{and } \sum w <= W$$

$$c = \sum_{i}^{n} ViXi \quad \text{(with fraction)}$$



u=-32, and c=-38

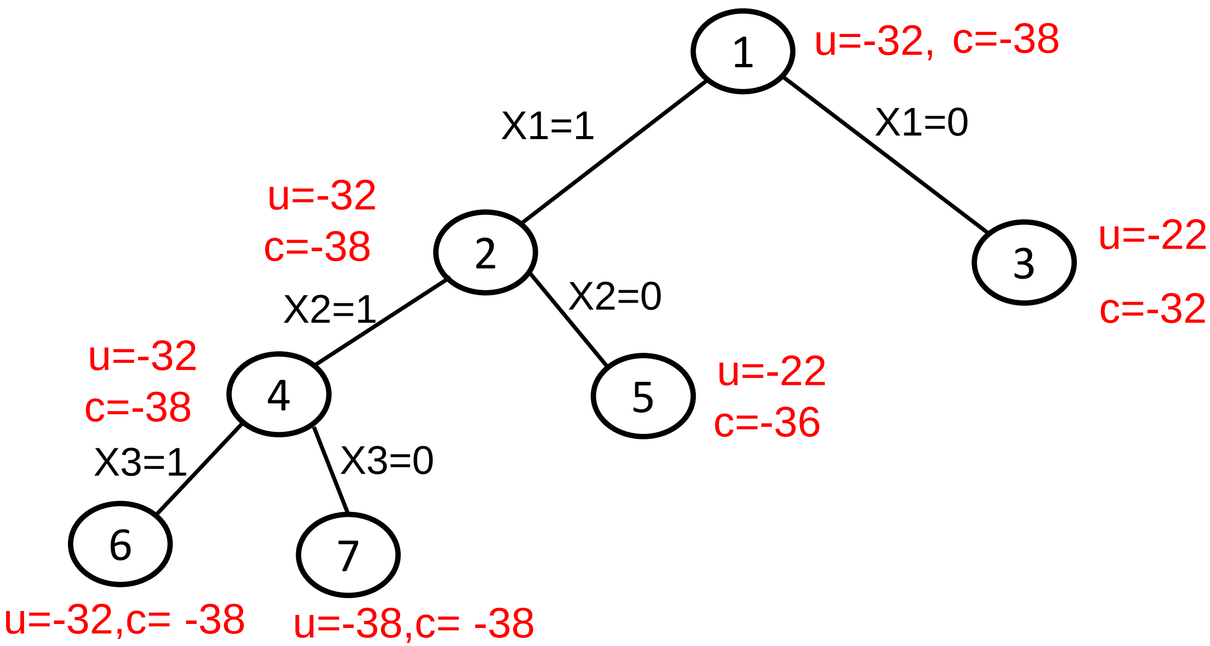
For node 2 we have two choices, either we include 2nd object or not. If object 2 is included, the values u and c will not change, since 2nd object is already present in u and c.
if 2nd object is not included, then
u= -(10+12) = -22, and 15-(2+6)<15 => 7<15
c= -(10+12+18*7/9) => -36

u= -22, c= -36, and c<u i.e., -36<-32, so this node will not be killed.

Now. Node 4 will be explored since, it has smallest value of c.

0/1 Knapsack Problem

Solve the following instance of knapsack using LC-BnB.
n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15.



$u = \sum_{i}^{n} V_i X_i$ and $\sum w \leq W$

$c = \sum_{i}^{n} V_i X_i$ (with fraction)

u=-32, and c=-38

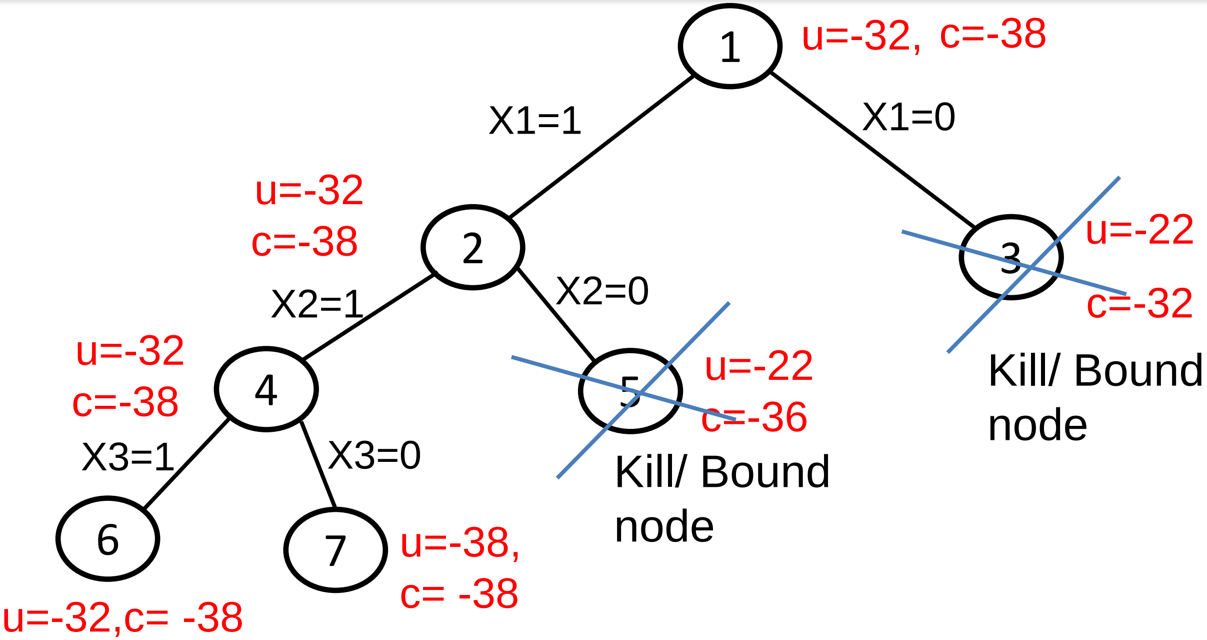
⇩

u=-38, and c=-38

For node 4, If object 3 is included, the values u and c will not change but if 3rd object is not included, then
u= -(10+10+18) = -38, and 15-(2+4+9)<=15
c= -(10+10+18) => -38

u= -38, c= -38, and c<=u i.e., -38<=-38, so node 7 will not killed, but node 3 and 5 is killed since their costs c are greater than the u. The new value of u is less than the existing one, so, u will be updated (u=-38).

0/1 Knapsack Problem



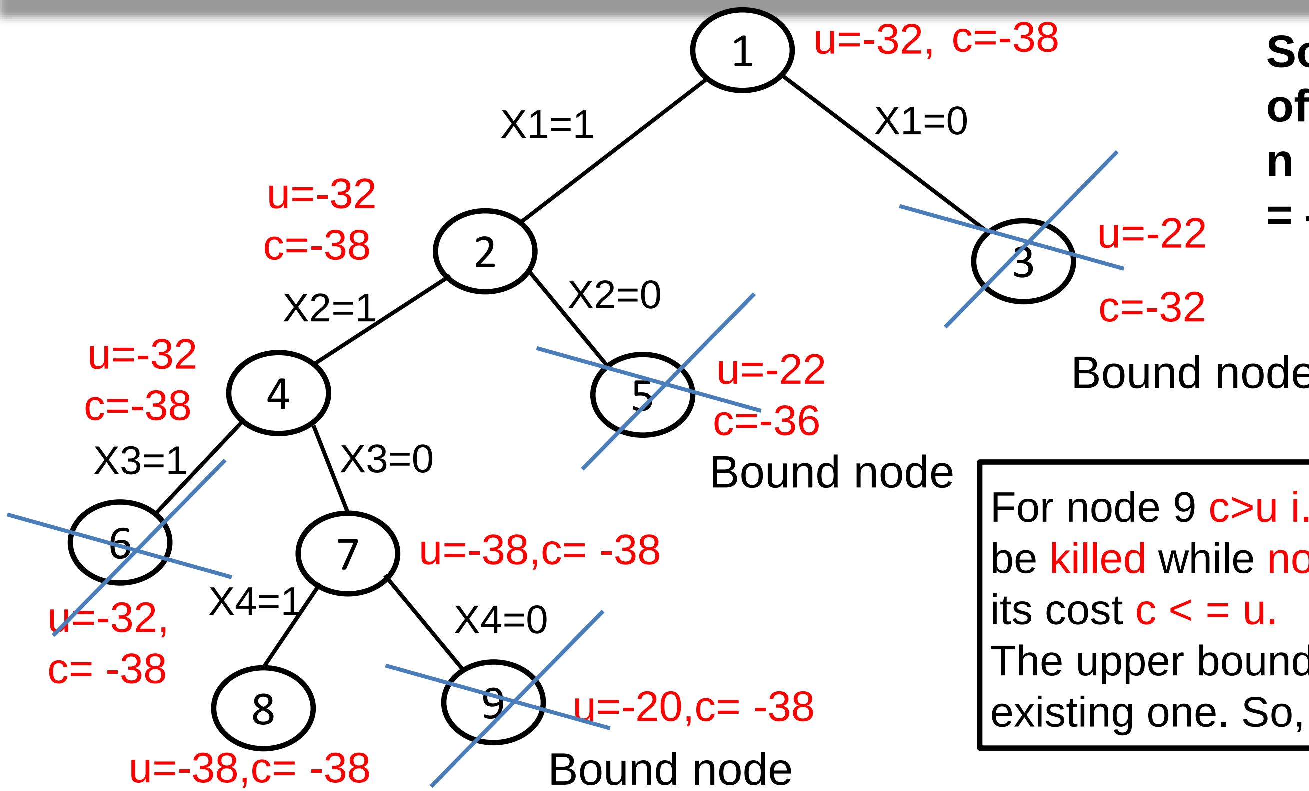
Solve the following instance of knapsack using LC-BnB.
n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15.

u=-38, and c=-38

Now, Node 6 and 7 both can be expanded, as their costs are same but node 6 cannot be expanded because object 4 cannot be included in the knapsack due to less remaining capacity.

0/1 Knapsack Problem

Solve the following instance of knapsack using LC-BnB. n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15.



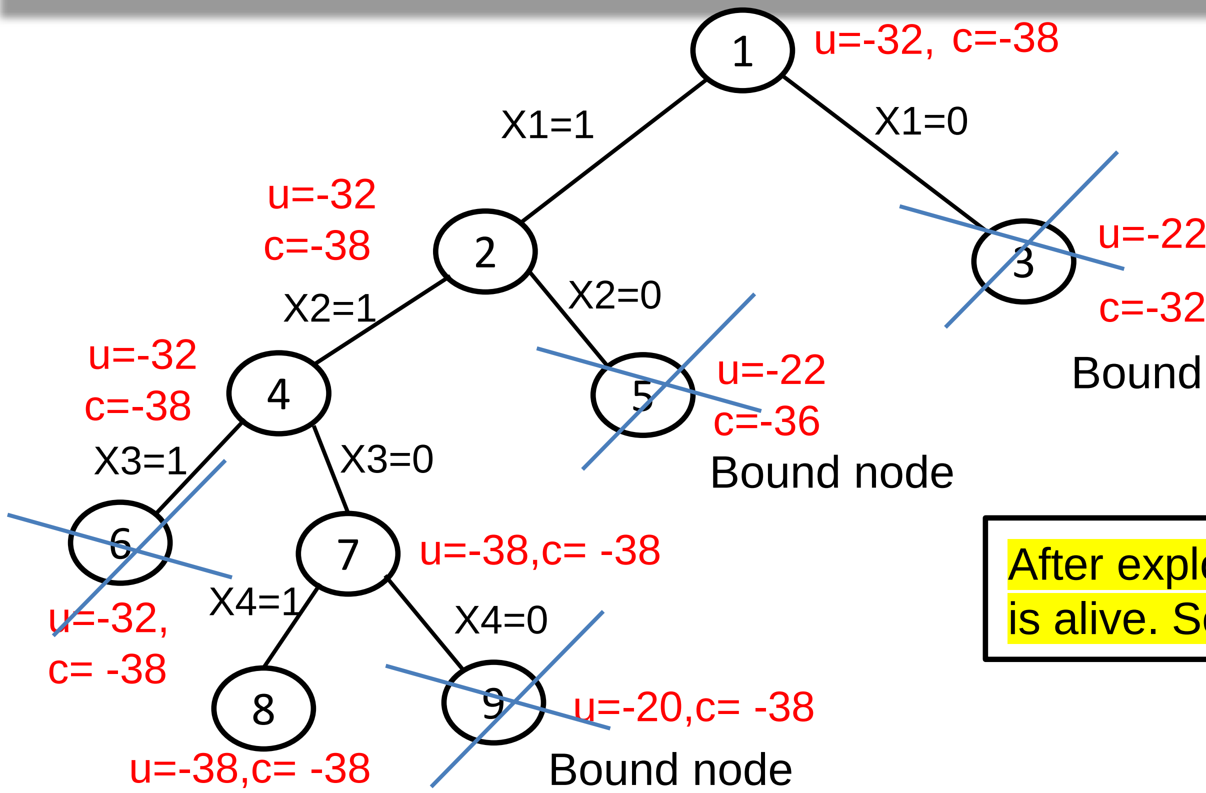
u=-38, and c=-38

For node 9 c>u i.e., -20<=-38, so this node will be killed while node 8 will not be killed because its cost c < = u.
The upper bound u of node 8 are equal to the existing one. So, it will not be updated

For node 7, If object 3 is not included and objects 1, 2, 4 are included, then
u= -(10+10+18) = -38, and 15-(2+4+9)<=15
c= -(10+10+18) => -38
If object 3 and 4 are not included and object 1 and 2 are included, then
u= -(10+10) = -20, and c= -38

After exploring all possibilities, only node 8 is alive. So, this will be the answered node.

0/1 Knapsack Problem



Solve the following instance of knapsack using LC-BnB.
n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15.

u=-38, and c=-38

After exploring all possibilities, only node 8 is alive. So, this will be the answered node.

Solution:
In the answer set 1st, 2nd and 4th object will be included. So, the total profit

P= 10+10+18 = 38

Item subset = {1, 1, 0, 1}

0/1 Knapsack Problem

Time Complexity

- The worst-case time complexity = $O(2^n)$, in cases where the entire tree has to be explored.
- However, in its best case, only one path through the tree will have to explored, and hence its best-case time complexity = $O(n)$.