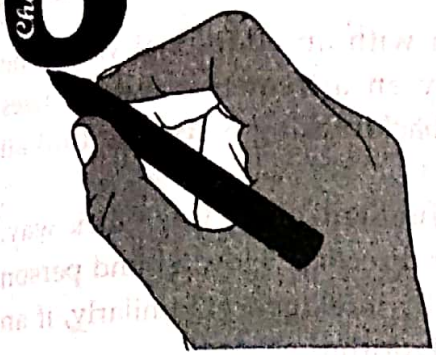
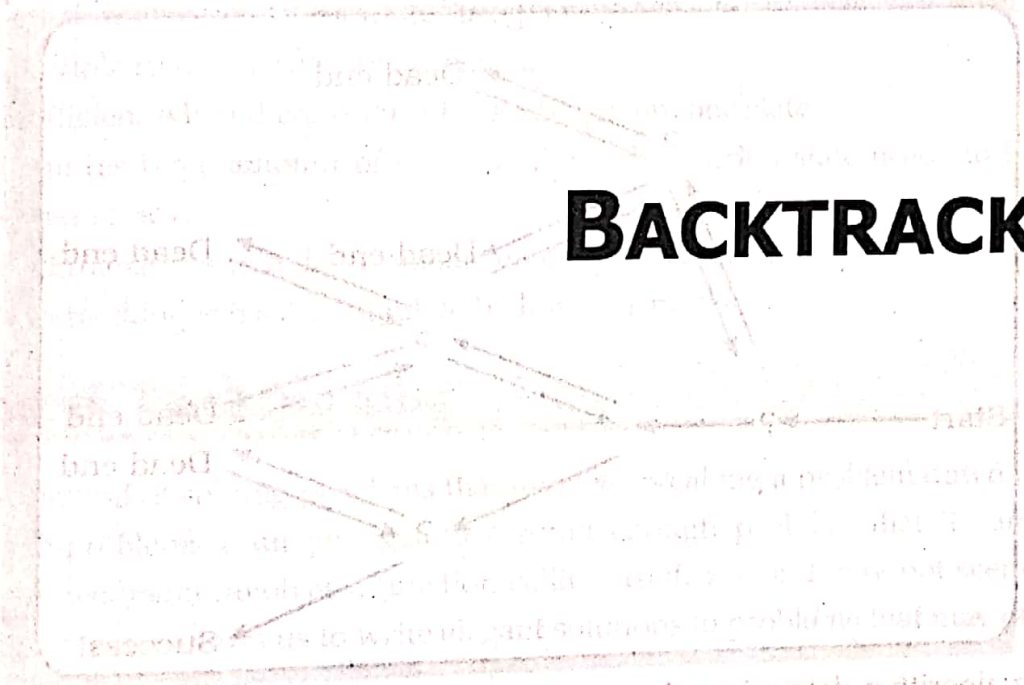
Chapter 6



# BACKTRACKING



## CHAPTER OUTLINE

**After studying this chapter, the reader will be able to understand the**

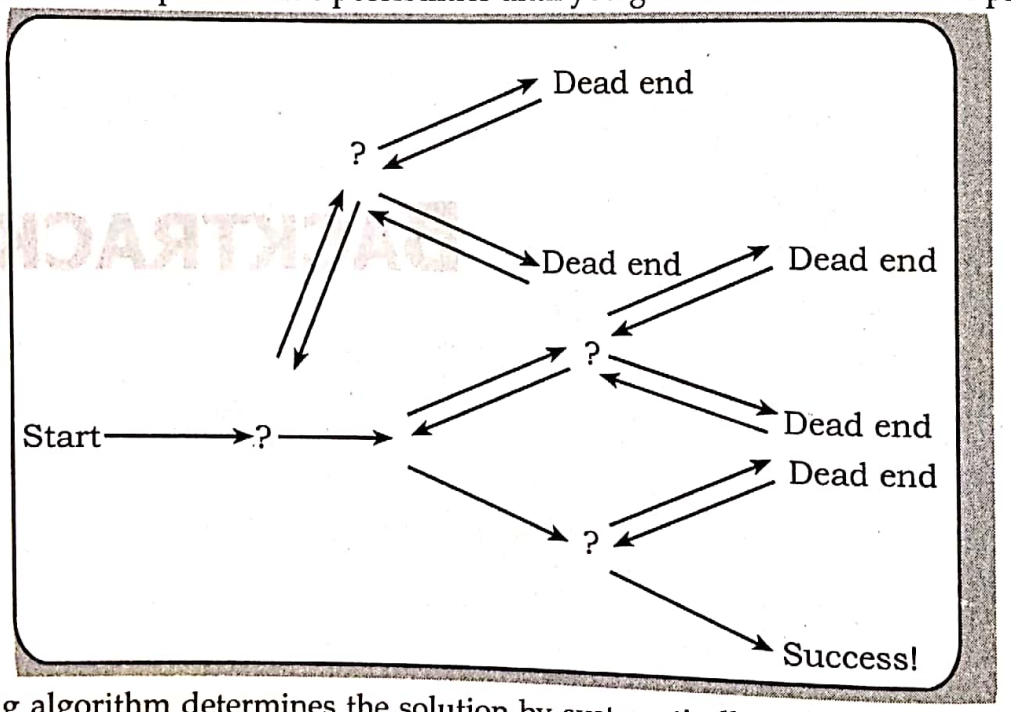
- Concept of Backtracking, Recursion vs Backtracking
- Backtracking Algorithms: Subset-sum Problem, Zero-one Knapsack Problem, N-queen Problem and their Analysis.

## Concept of Backtracking

The Backtracking is an algorithmic-method to solve a problem with an additional way. The Backtracking is an algorithmic-technique to solve a problem by an incremental way. It uses recursive approach to solve the problems. We can say that the backtracking is used to find all possible combination to solve an optimization problem.

Have you ever seen poor blind people walking in roads? If they find any obstacles in their way, they would just move backward. Then they will proceed in other direction. How a blind person could move backward when he finds obstacles? Simple answer by intelligence! Similarly, if an algorithm backtracks with intelligence, it is called backtracking algorithm.

**Backtracking** is general algorithm for finding solution to some computational problem. We have set of several choices. If one choice from set of choices proves incorrect, computation backtracks or restarts at the point of choice and tries another choice. In backtracking you use recursion in order to explore all the possibilities until you get the best result for the problem.




Backtracking algorithm determines the solution by systematically searching the solution space for the given problem. Backtracking is a depth-first search with any bounding function. All solution using backtracking is needed to satisfy a complex set of constraints. The constraints may be explicit or implicit. **Explicit Constraint** is ruled, which restrict each vector element to be chosen from the given set. **Implicit Constraint** is ruled, which determine which each of the tuples in the solution space, actually satisfy the criterion function. Backtracking is undoubtedly quite simple - we "explore" each node, as follows:

To "explore" node N:

1. If N is a goal node, return "success"
2. If N is a leaf node, return "failure"
3. For each child C of N,
   Explore C
   If C was successful, return "success"
4. Return "failure"

### Advantages of Backtracking

- It is a step-by-step representation of a solution to a given problem, which is very easy to understand
- It has got a definite procedure.
- It is easy to first develop an algorithm, & then convert it into a flowchart & then into a computer program.
- It is independent of programming language.
- It is easy to debug as every step is got its own logical sequence.
- State changes are stored in stack, meaning we do not need to concern ourselves about them.
- Code size is usually small.

### Disadvantages Backtracking Algorithm

- It is time consuming computer program
- Multiple function calls are expensive.
- Inefficient when there is lots of branching from one state.
- Requires large amount of space as the each function state needs to be stored on system stack.
- Backtracking is non-deterministic unless you tracked it.
- Backtracking is hard to simulate by human simulate.

## Recursion vs. Backtracking

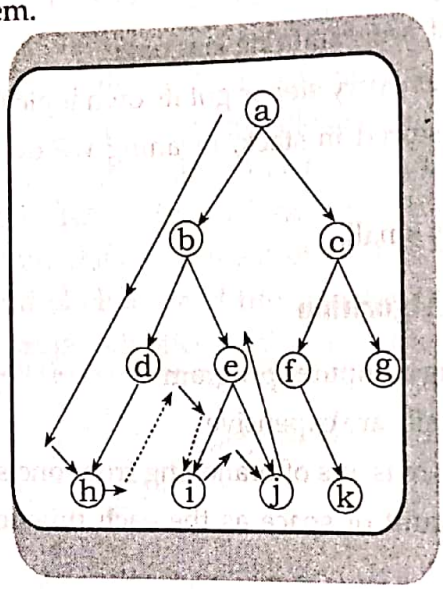
Recursion is a method of solving problems that involves breaking a problem down into smaller and smaller sub-problems until you get to a small enough problem that it can be solved trivially. Usually recursion involves a function calling itself. While it may not seem like much on the surface, recursion allows us to write elegant solutions to problems that may otherwise be very difficult to program.

In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

```
int fact(int n)
{
  if (n < = 1) // base case
    return 1;
  else
    return n*fact(n-1);
}
```

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

**Backtracking** is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time. **Backtracking** is general algorithm for finding solution to some computational problem. We have set of several choices. If one choice from set of choices proves incorrect, computation backtracks or restarts at the point of choice and tries another choice. In backtracking we use recursion in order to explore all the possibilities until we get the best result for the problem.




condition jaba satisfy hudenwni
taba backtrack garihlxa
ani satisfy bhainjel samma janxa agadi

## Backtracking Algorithms: Subset-sum Problem

In this problem, there is a given set with some integer elements. And another some value is also provided, we have to find a subset of the given set whose sum is the same as the given sum value.

Here backtracking approach is used for trying to select a valid subset when an item is not valid, we will backtrack to get the previous subset and add another element to get the solution.

The **Subset-Sum** Problem is to find a subset's' of the given set S = (S1, S2, S3...Sn) where the elements of the set S are n positive integers in such a manner that s' ∈ S and sum of the elements of subset 's' is equal to some positive integer 'X.'

The Subset-Sum Problem can be solved by using the backtracking approach. In this implicit tree is a binary tree. The root of the tree is selected in such a way that represents that no decision is yet taken on any input. We assume that the elements of the given set are arranged in increasing order:

$$S1 \leq S2 \leq S3... \leq Sn$$

The left child of the root node indicated that we have to include 'S1' from the set 'S' and the right child of the root indicates that we have to execute 'S1'. Each node stores the total of the partial solution elements. If at any stage the sum equals to 'X' then the search is successful and terminates.

The dead end in the tree appears only when either of the two inequalities exists:

- The sum of s' is too large i.e.

  $s' + Si + 1 > X$

- The sum of s' is too small i.e.

  $s' + \Sigma_j^n = i + 1 S_{j<x}$

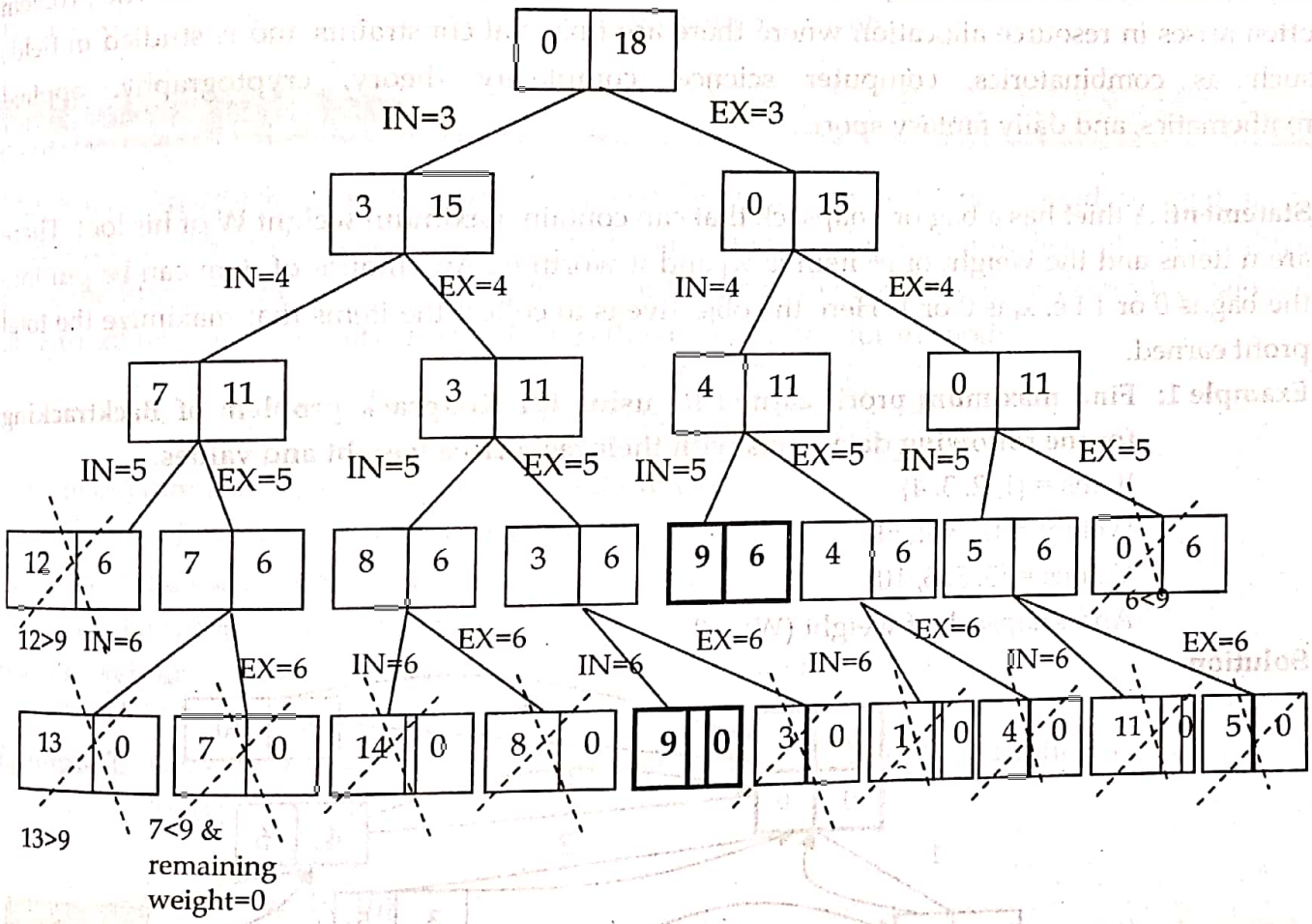
**Example 1: Given a set S= {3, 4, 5, 6} and X=9. Obtain the subset sum using backtracking approach.**

**Solution:**

Initially S = (3, 4, 5, 6) and X=9

S′ = {Φ}

The implicit binary tree for the subset sum problem is shown as fig:




**Algorithm**

1. Start
2. if index == array.length then
   Return false
3. if array[index] == sum then
   Return true
4. Iterate given array from index to array.length
   If array[i] > sum then
   Don't do anything take next element from array
   If array[i] == sum then
   Return true
   Recursively call with index +1 and sum – array[i]
   If last recursive call was success then
   Return true
5. return false
6. Stop

index=0

uta ko tw fail khayo so hami side ko wala ma janxam ni tw tarw tesma tyo wala array[i] add gareko xaewn so tyo calll gareko

## Zero-one Knapsack Problem

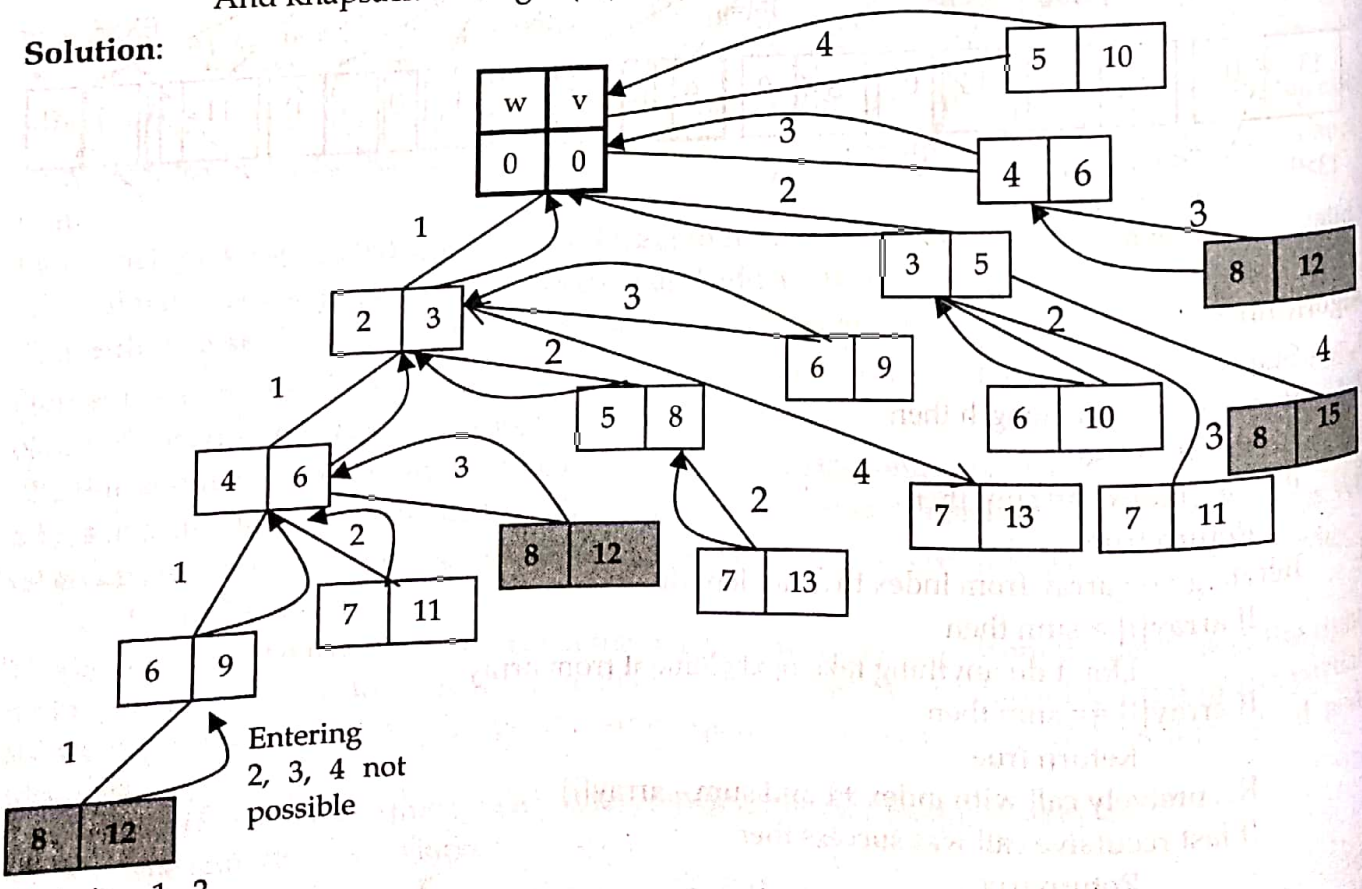
Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. The problem often arises in resource allocation where there are financial constraints and is studied in fields such as combinatorics, computer science, complexity theory, cryptography, applied mathematics, and daily fantasy sports.

**Statement:** A thief has a bag or knapsack that can contain maximum weight W of his loot. There are n items and the weight of $i^{th}$ item is $w_i$ and it worth $v_i$. An amount of item can be put into the bag is 0 or 1 i.e. $x_i$ is 0 or 1. Here the objective is to collect the items that maximize the total profit earned.

**Example 1: Find maximum profit earned by using 0/1 Knapsack problem of Backtracking for the following data items with their respective weight and values.**

Items = {1, 2, 3, 4}
Weight = {2, 3, 4, 5}
Values = {3, 5, 6, 10}
And knapsack of weight (W) = 8

**Solution:**



Also we can calculate the maximum profit of above problem efficiently by using branch and bound method.

The branch and bound technique like Backtracking explores the implicit graph and deals with the optimal solution to a given problem. In this technique at each stage we calculate the bound for a particular node and check whether this bound will be able to give the solution or not. If we find that at any node the solution so obtained is appropriate but the remaining solution is not leading to a best case then we leave this node without exploring.

## Knapsack problem

The above knapsack problem can be solved by using branch and bound method more appropriately. Here we need to construct a binary tree in which each node contains the three parts, the first part indicates the total weight of the item, the second part indicates the value of the current item and the third part indicates the upper bound for the node.

| Wt. | Value |
|---|---|
| Upper bound (ub) | |

The upper bound ub of the node can be computed as,

ub = v1+v2+.....+ fraction part of (vi)

Where, v is the value of the current node,

W is the total weight of the knapsack

w is the weight of the current node and ub is the upper bound of the node.

**Example 2:** Consider three items along with their respective weights and values as,

I = {I1, I2, I3, I4}

w = {3, 5, 9, 5}

v = {45, 30, 45, 10}

the knapsack has the maximum capacity W = 16, we have to pack this knapsack using the branch and bound technique so as to give the maximum possible value while considering all constraints.

**Solution:** At first we need to calculate vi/wi as below

| Items | Wi | vi | vi/wi |
|---|---|---|---|
| I1 | 3 | 45 | 15 |
| I2 | 5 | 30 | 6 |
| I3 | 9 | 45 | 5 |
| I4 | 5 | 10 | 2 |

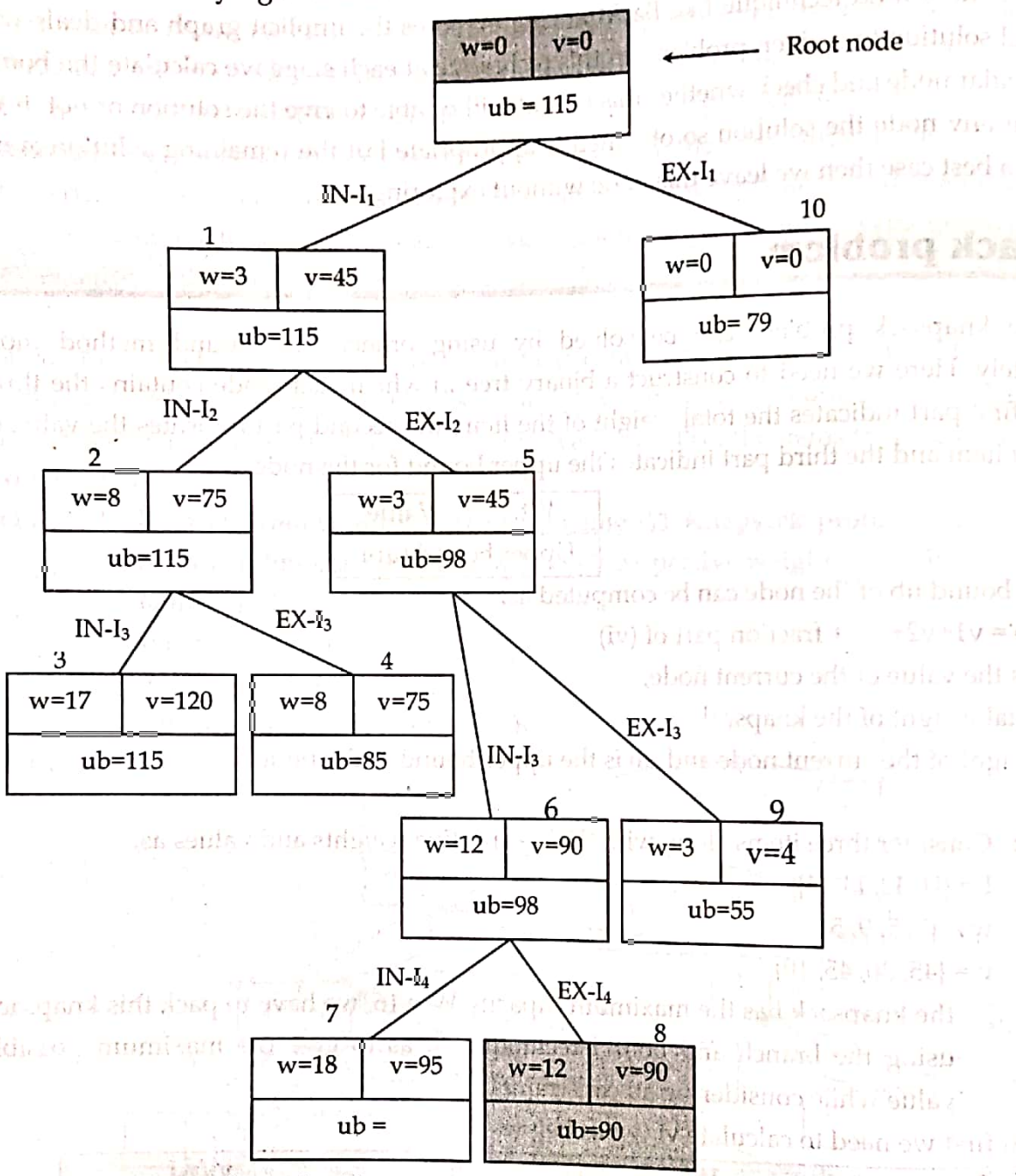
Since data items are already in sorted order thus we do not need to arrange the data items in descending order of vi/wi,

Now we start with root node, the upper bound for the root node can be computed as,

ub = value of I1 + value of I2+ fraction part of I3 = 45+30+45/9*8

= 45+30+40=115

Next we include item I1, which is indicated by the left branch and exclude item I1 which is indicated by right branch.




Thus maximum profit = 90 with sequence of item={$I_1$, $I_3$}

## N-queen Problem

This problem is to find an arrangement of N queens on a chess board, such that no queen can attack any other queens on the board. The chess queens can attack in any direction as horizontal, vertical, horizontal and diagonal way. A binary matrix is used to display the positions of N Queens, where no queens can attack other queens.

So initially we are having n × n un-attacked cells where we need to place n queens. Let's place the first queen at a cell (i, j), so now the number of un-attacked cells is reduced, and number of

queens to be placed is n−1. Place the next queen at some un-attacked cell. This again reduces the number of un-attacked cells and number of queens to be placed becomes n−2. Continue doing this, as long as following conditions hold.

- The number of un-attacked cells is not 0.
- The number of queens to be placed is not 0.

If the number of queens to be placed becomes 0, then it's over, we found a solution. But if the number of un-attacked cells become 0, then we need to backtrack, i.e. remove the last placed queen from its current cell, and place it at some other cell. We do this recursively.

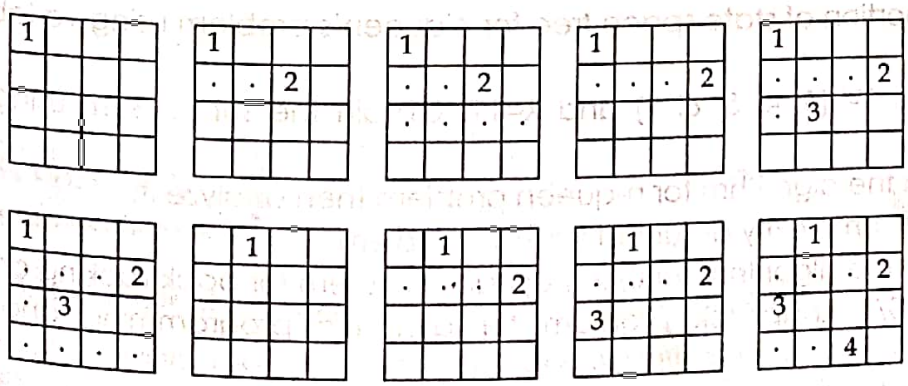
**Algorithm**

1. Start
2. Place the queens column wise, start from the left most column
3. If all queens are placed.
   a. Return true and print the solution matrix.
4. Else
   a. Try all the rows in the current column.
   b. Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
   c. If placing the queen in above step leads to the solution return true.
   d. If placing the queen in above step does not lead to the solution, BACKTRACK, mark the current cell in solution matrix as 0 and return false.
5. If all the rows are tried and nothing worked, return false and print NO SOLUTION.
6. Stop

**Analysis**

Solution of N Queen problem using backtracking checks for all possible arrangements of N Queens on the chessboard. And then checks for the validity of the solution. Now number of possible arrangements of N Queens on N x N chessboard is N!. So average and worst case complexity of the solution is O(N!). The best case occurs if we find our solution before exploiting all possible arrangements. This depends on our implementation. And if we need all the possible solutions, the best, average and worst case complexity remains O(N!)

**Example:** **Here's how it works for N=4.**



Which is one of the solutions of 4-queen problem of sequence is (2, 4, 1, 3)

Also by mirroring of this solution we get another possible solution as below,

| | | | |
|---|---|---|---|
| | | 1 | |
| 2 | . | . | . |
| . | | | 3 |
| . | 4 | . | |

The sequence is (3, 1, 4, 2)

## DISCUSSION EXERCISE

1. Define backtracking. Explain it with suitable example.
2. Differentiate between backtracking and recursion with suitable example.
3. State and explain 0/1 knapsack problem for backtracking with suitable example.
4. What are the application areas of backtracking? Explain
5. What are the advantages and **disadvantages** of backtracking?
6. **Explain** Subset-sum Problem with suitable example.
7. Write an algorithm for N – queen's problem. Give time and space complexity for 8 – queen's problem.
8. Give the statement of sum –of subsets problem. Find all sum of subsets for n=4, (w1, w2, w3, w4) = (11, 13, 24, 7) and M=31. Draw the portion of the state space tree using fixed – tuple sized approach.
9. Distinguish between backtracking and branch – and bound techniques.
10. Discuss the 4 – queen's problem. Draw the portion of the state space tree for n= 4 queens using backtracking algorithm.
11. Describe the backtracking 0/1 Knapsack Problem. Find an optimal solution for the backtracking 0/1 knapsack instance for n=3, m=6, profits are (p1, p2, p3 ) = (1,2,5), weights are (w1,w2,w3) =(2, 3, 4).
12. What is a Backtracking and give the 4 – Queens's solution.
13. Write control abstraction for backtracking.
14. What is a backtracking? Give the explicit and implicit constraints in 8 queen's problem.
15. Draw the portion of state space tree for 4 queen's problem using variable –tuple sized approach.
16. Given a set S= {6, 4, 5, 6, 9} and X=11. Obtain the subset sum using backtracking approach.
17. Write down the algorithm for n-queen problem then analyze it.
18. What is the complexity of sum of subsets problem?
19. Write down the algorithm for 0/1 knapsack problem for backtracking and analyze it.
20. Compare 0/1 knapsack problem for dynamic programming and backtracking approach.

❑❑❑