## Module-5

### Backtracking:

- Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally.
- Principal idea is to construct solutions of one component at a time & evaluate such partially constructed candidate as follows:
  - constraints are verified for partially constructed solution, if not violated it is continued.
  - if violated, backtracks to replace the last component of the partially constructed solution with its next option.

### Implementation- State space tree.

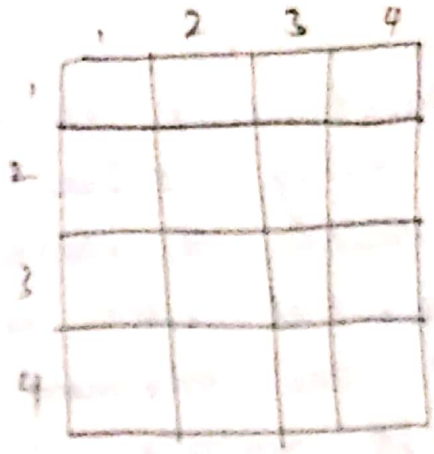
- A state space tree consists of nodes representing state, & arc representing the legal moves from one state to another.

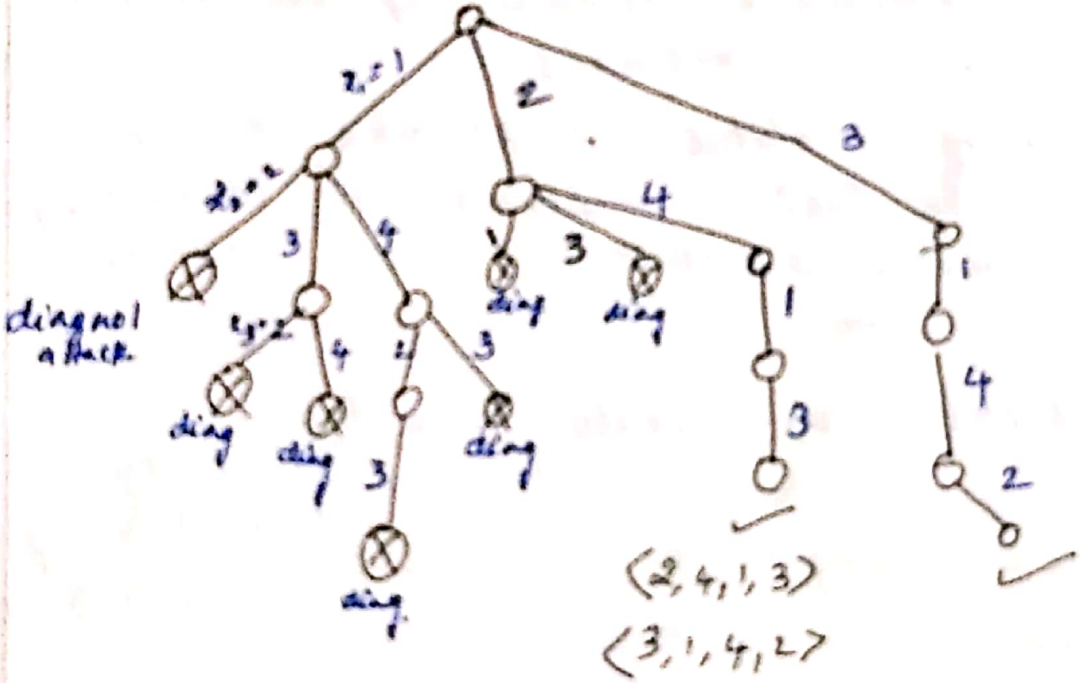
- Backtracking is mainly applied to combinatorial problems.

## n - queens problem:

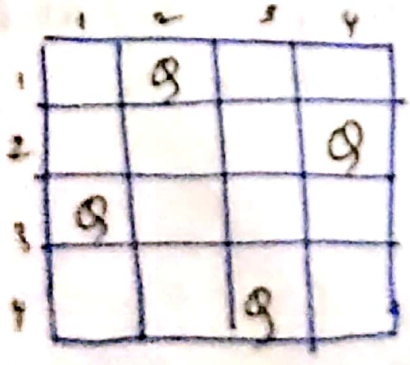
$Q_1, Q_2, Q_3, Q_4$

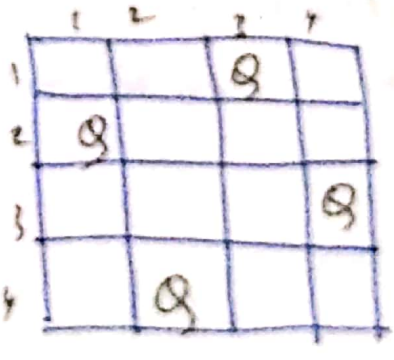
→ The problem is to place n queens on an n by n chessboard so that no two queens attack each other by being in the same row or in the same column or on the same diagnol.




2 solutions :





## Sum of Subsets:

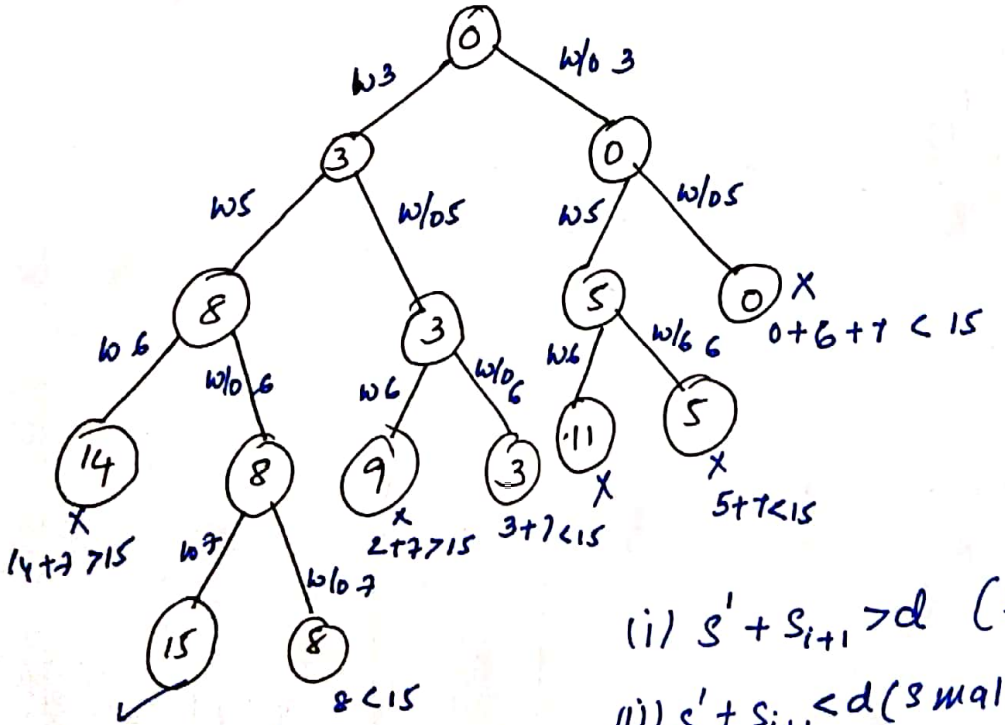
- "Given a set of $S = \{s_1, \ldots, s_n\}$ of n positive integers, find a subset whose sum is equal to a given positive integer d"

Ex: Let $s = \{1, 2, 5, 6, 8\}$ and $d = 9$, then the subsets summing to d are:

$\{1, 2, 6\}$ and $\{1, 8\}$

problem: Construct the state space tree & solve the following instance of subset_sum problem: $S = \{3, 5, 6, 7\}$ and $d = 15$.

Sol ⇒ The root of the tree represents starting points. Its left & right children represent the inclusion & exclusion of elements respectively:




(i) $s' + s_{i+1} > d$ (sum is too large)

(ii) $s' + s_{i+1} < d$ (small)

* Apply backtracking to solve the following instance of subset-sum problem

(i) $S = \{1, 3, 4, 5\}$ and $d = 11$

(ii) $S = \{11, 13, 24, 7\}$ and $d = 31$

and draw the state-space tree.

Find all possible solutions:

* Let $w = \{5, 7, 10, 12, 15, 18, 20\}$ & $m = 35$.

Find all subsets of w that sum to m

Draw the state-space tree.

## Graph Coloring:

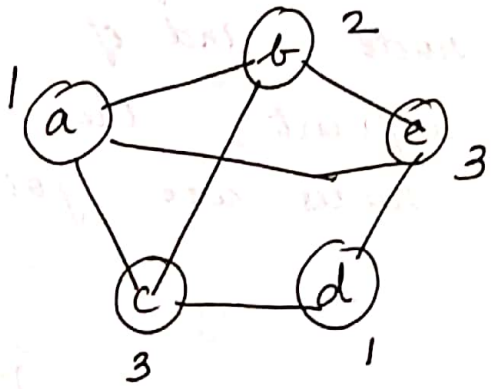
Let $G = (V, E)$ be a graph & "m" a positive integer - no of colors.

problem → Find whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color and to use only 'm' colors. This is called m-color decision problem.

### - m color - optimization problem -

smallest integer "m" that can be to solve m-color decision problem. This integer 'm' is called chromatic number of graph.

- Consider the following graph which can be colored with 3 colors - 1, 2 & 3




Note: If d is degree of a graph, then it can be colored with d+1 colors.

## 4 color problem for planar graphs:

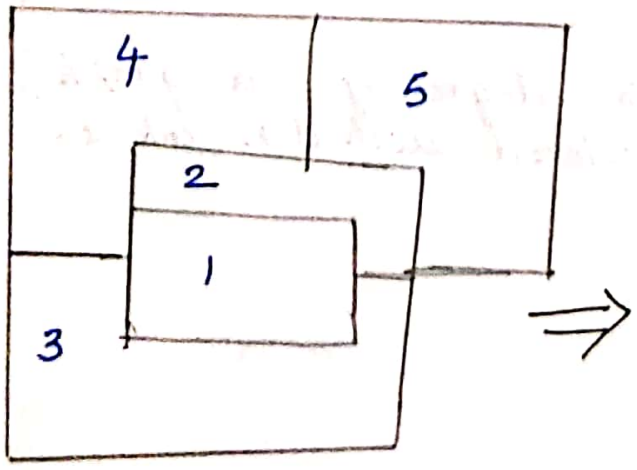
- A graph is said to be planar iff it can be drawn in a plane in such a way that no two edges cross each other.

- The famous special case of m-Colorability decision problem is 4-color problem for planar graph:

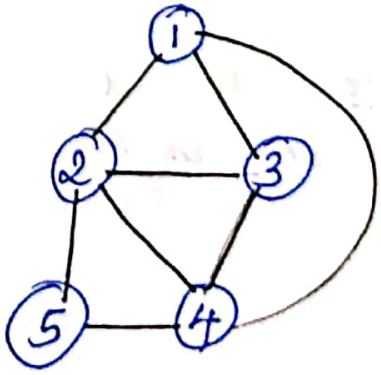
"Given any map, can the regions be colored in such a way that no two adjacent regions have the same color, yet only four colors are needed."

- A map can easily be transformed into a graph. Each region of map becomes a node and if two regions are adjacent, then the corresponding nodes are joined by an edge:

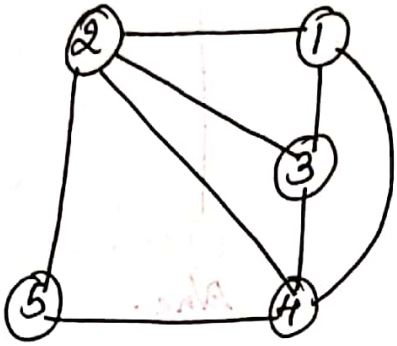
Ex: map




Graph



* Solve the following instance of graph coloring problem. Draw the state space tree, let m=4




Solutions:

Nodes

1

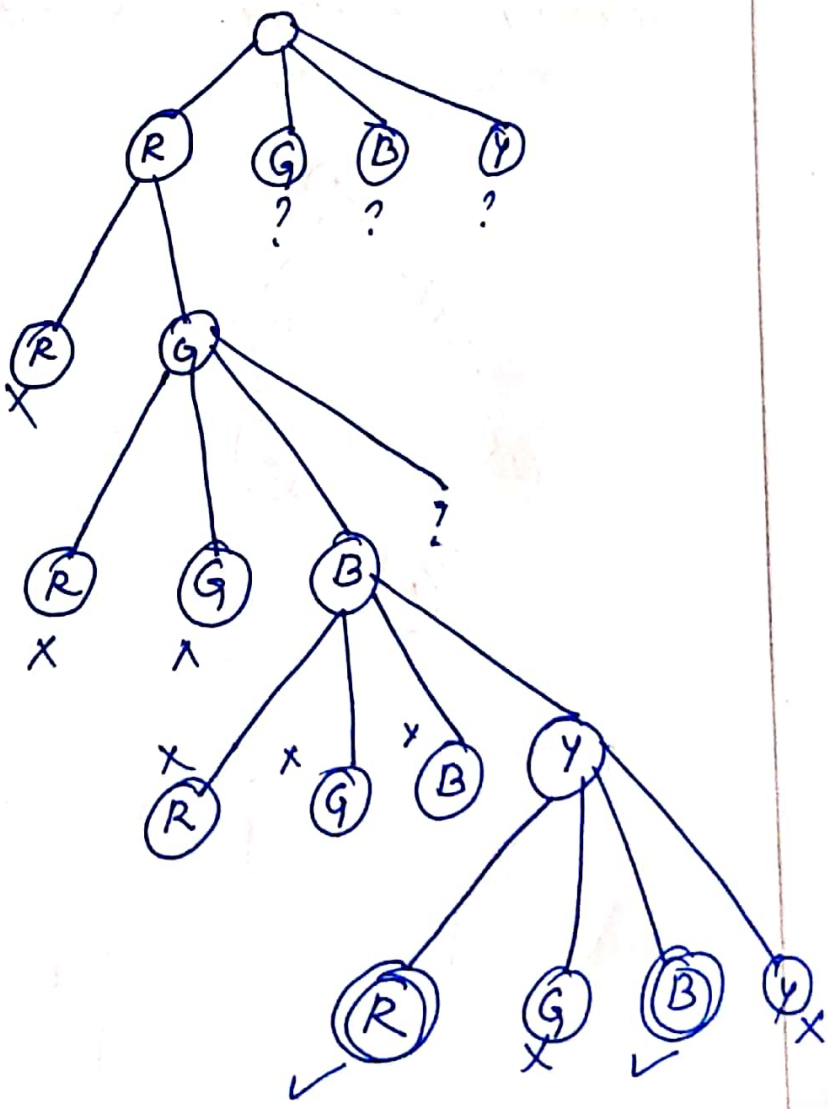
2

3

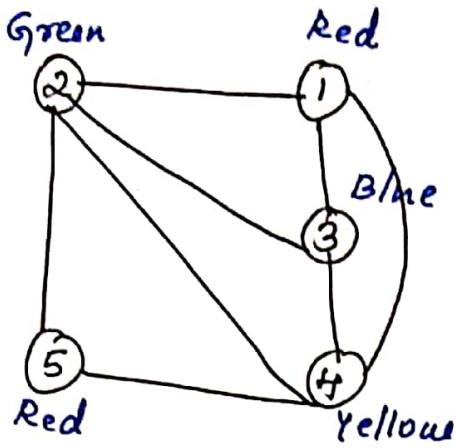
4

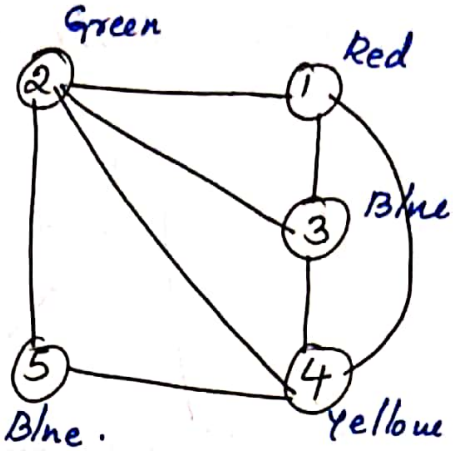
5

Tree

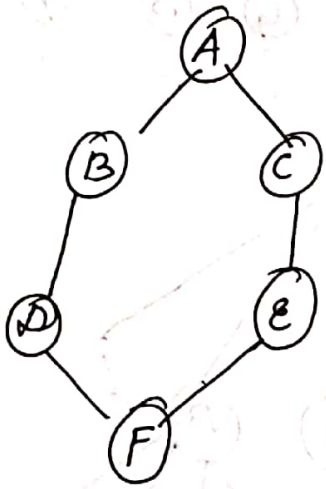


Solutions are:




* Solve the following instance of graph coloring problem & obtain the state space tree:



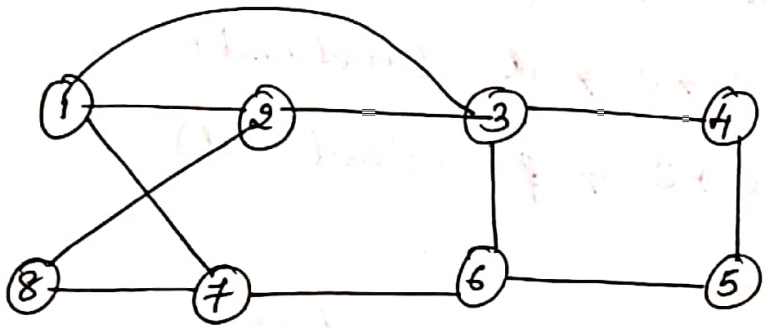
## Hamilton cycles:

- Let $G=(V,E)$ be a connected graph with 'n' vertices. A Hamilton cycle is a round tripath along 'n' edges of G that visits every vertex once and returns to its starting position.

- Hamilton cycle begins at some vertex $v_1 \in G$ and the vertices are visited with edges $(v_i, v_{i+1})$ where $v_i$ are distinct.

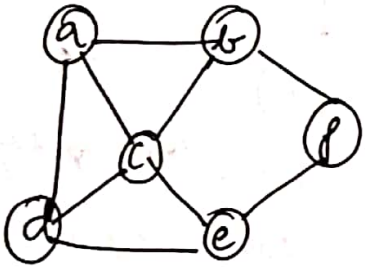
- Ex:

(1)

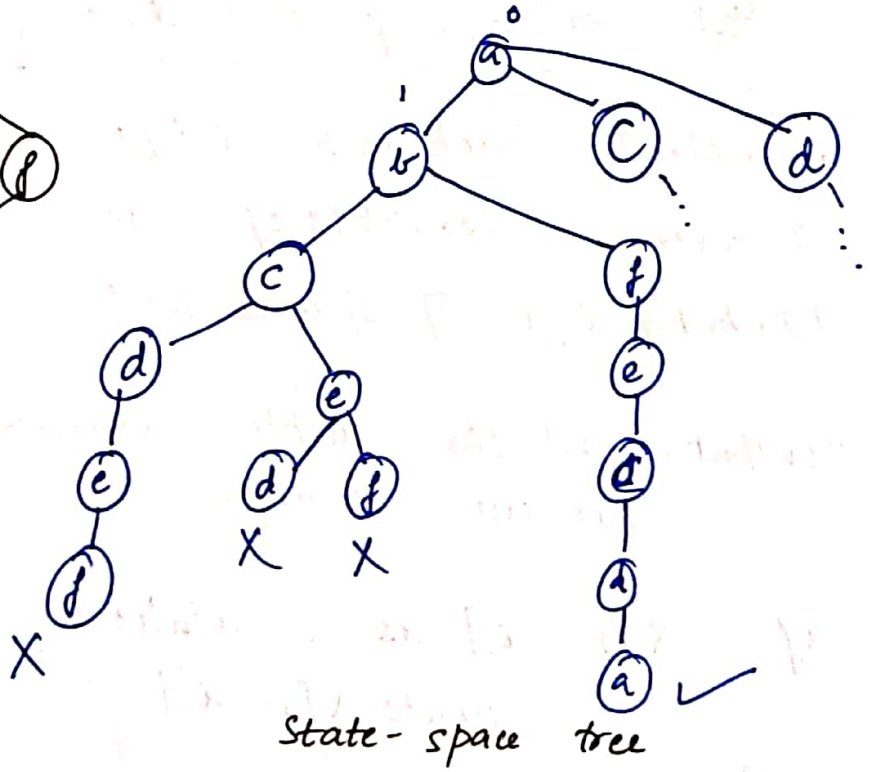



Here 1,2,8,7,6,5,4,3,1 is the Hamilton cycle.

(2)




Graph




State-space tree

Steps:

- Alphabet 'a' vertex is selected as a root-vertex.
- adjacent to a → b, c, d
- First select b (DFS)

→ From b, the algorithm proceeds to c & f

→ from b → c → d → e → f × dead end.

→ so backtrack from

b → c → e → d (dead end)

b → e → e → f (dead end)

→ backtrack

b → f → e → c → d → a

soln → [a, b, f, e, c, d, a]

---

Algorithm: Backtrack (X[1 .... i])

// Given a template of generic back tracking algo

// Input: X[1 ... i] specifies first i promising components of sol

// Output: All the tuples represent the problem solution.

if X[1 ... i] is a solution

write X[1 ... i]

else

discard

for each element $x \in S_{i+1}$ consistent with $x[1 \cdots i]$ & constraints do

$x[i+1] \leftarrow x$

Backtrack $[x[1 \cdots i+1]]$

---

## Hamiltonian cycle

```
Algorithm HAMILTONIAN(k)
{
    do
    {
        next_vertex(k);
        if (x[k] == 0)
            return n;
        if (k == n)
            print [x[1:. n]);
        else
            Hamiltonian (k+1);
    }
    while (true);
}

Algorithm Next vertex (k)
{
    do
    {
        x[k] = (x[k] + 1) mod (n+1);
        if (x[k] == 0) return;
```

```
if (G(x[k-1], x[k]) ≠ 0)
{
    for j ← 1 to k-1 do
        if (x[j] == x[k]) break;
    if (j == k)
        if (k<n as (k==n) &&
            G(x[n], x[1]) ≠ 0
        return;
    }
} while (true);
}
```

## Branch and Bound:

- It is an algorithmic approach to solve discrete & combinatorial optimization problem.
- In this method, a space tree with all possible solutions is generated. Then partitioning or branching is done at each node of the tree.
- Bound value is computed at each node.
- Node's bound value is compared with best solution so far.
- If the bound value of some node is not better than the best solution, then that corresponding node is not expanded further, such node is called non-promising node.
- Reasons for Terminating search paths in state space tree of branch & bound are:
  1. value of node's bound is not better than best solution.
  2. The node represents no feasible solutions because the constraints of problem are already violated.
  3. Subset of feasible has no further choices can be made.

| | Backtracking | Branch & Bound. |
|---|---|---|
| 1. | Typically decision problems can be solved using backtracking. | Typically optimization problems can be solved using Branch and Bound. |
| 2. | It can consider bad choices also | It proceeds on better solutions. so there cant be a bad solution. |
| 3. | state space tree is searched untill the solution is obtained | state space tree needs to be searched completly |
| 4. | Applications:<br>• N-queens<br>• sum of subset<br>• Graphcoloring etc | • TSP<br>• 0/1 knapsack etc |
| 5. | Solution is traced using Depth first search | Breadth first search. |
| 6. | In this method all possible solutions are tried. If sol$^n$ is not satisfying the constraint then backtrack & try another sol$^n$ | It uses bounding values i/e either upper bound / lower bound. |

## Assignment problem:

Given 'n' people and 'n' jobs, the problem is to assign 'n' jobs to 'n' people such that cost of assignment is as small as possible.

- First lower bound (LB) is computed.
  The nodes with LB is compared, & expanded only if satisfying

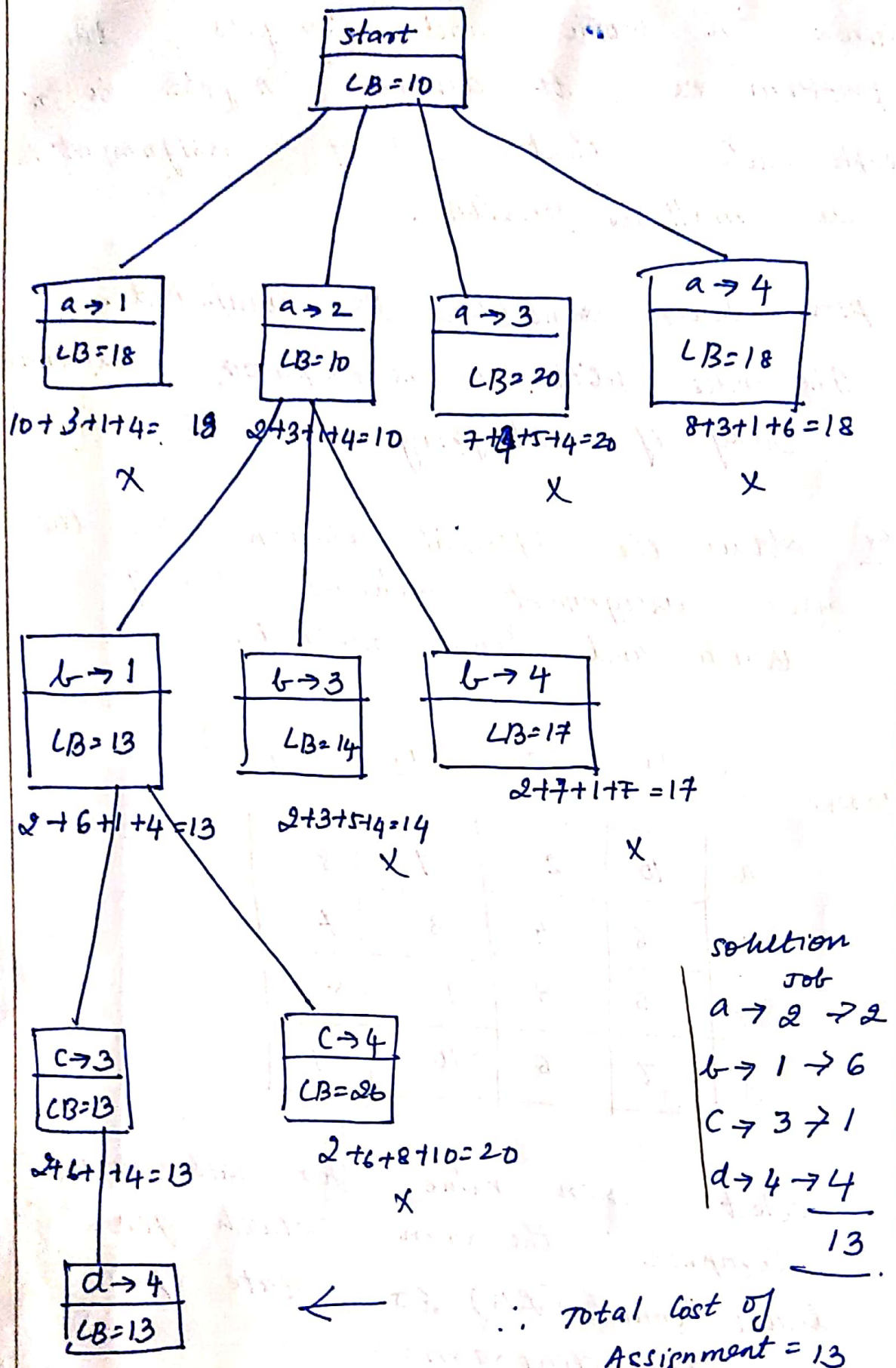
**Ex:** Obtain the optimal solution for the given assignment problem using branch and bound method:

| person | $J_1$ | $J_2$ | $J_3$ | $J_4$ |
|---|---|---|---|---|
| a | 10 | 2 | 7 | 8 |
| b | 6 | 4 | 3 | 7 |
| c | 5 | 8 | 1 | 8 |
| d | 7 | 6 | 10 | 4 |

Sol^n:

Select min value for each row & Compute the sum which gives lower bound (LB) for state space tree diagram:

$$2 + 3 + 1 + 4 = 10$$




solution

| Job | |
|---|---|
| a → 2 | → 2 |
| b → 1 | → 6 |
| c → 3 | → 1 |
| d → 4 | → 4 |
| | 13 |

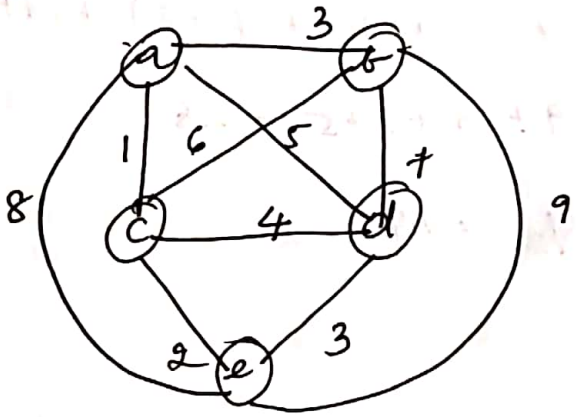
∴ Total cost of Assignment = 13

## Travelling Salesperson - Branch & Bound:

Given 'n' cities and costs incurred in travelling from any city to other, the task is to find the shortest tour possible from a given source city.

**problem:**

Solve the following instance of TSP using branch & bound method:




Compute $\hat{f}B$

- for each city i, $1 \le i \le n$, find sum 's' of the distances from city i to the two nearest cities.

Compute sum 's' of these 'n' numbers, divide the result by 2 ie $\hat{f}B = (s/2)$.

$$\therefore \hat{f}B = [\overset{a}{(1+3)} + \overset{b}{(3+6)} + \overset{c}{(1+2)} + \overset{d}{(2+4)} + \overset{e}{(2+3)}] / 2 = 28/2$$

$$= 14$$

Assumptions:

(i) Consider only tours that starts at 'a'

(ii) Generate only tours in which 'b' is visited before 'c'.

- Now consider the edges (a,b) (a,c) (a,d) (a,e) at level 1

$$(a,b) = \overset{a}{(3+1)} + \overset{b}{(3+6)} + \overset{c}{(1+2)} + \overset{d}{(3+4)} + \overset{e}{(2+3)}$$

$$= 4 + 9 + 3 + 7 + 5 = 28$$

$$= 28/2 = 14$$

$(a,b) = 14$.

$(a,c)$ → Not considered as b is not before c

$$(a,d) = \overset{a}{(1+5)} + \overset{b}{(3+6)} + \overset{c}{(1+2)} + \overset{d}{(3+5)} + \overset{e}{(2+3)}$$

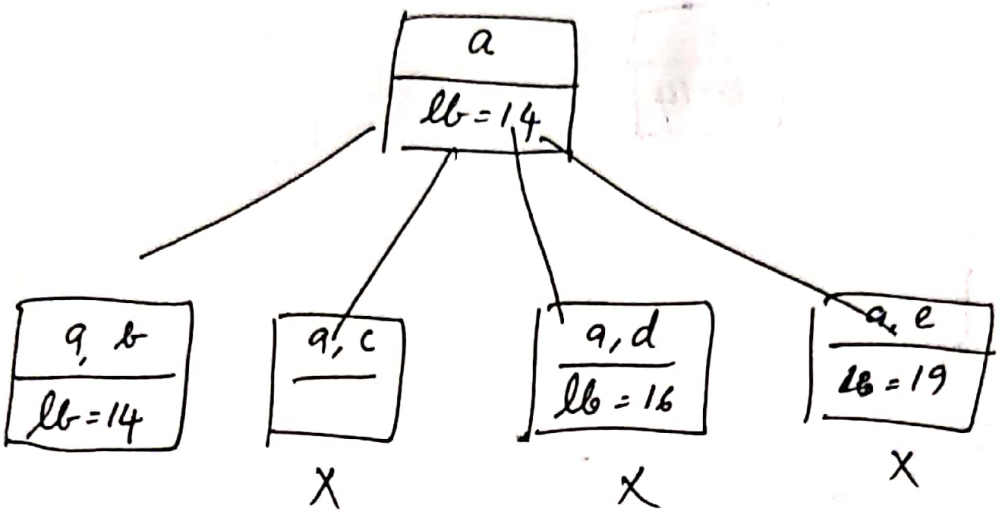
$$= 6 + 9 + 3 + 8 + 5 = 31$$

$$= \frac{31}{2} = \underline{16}$$

$$(a,e) = \overset{a}{(1+8)} + \overset{b}{(3+6)} + \overset{c}{(1+2)} + \overset{d}{(4+3)} + \overset{e}{(2+8)}$$

$$= 9 + 9 + 3 + 7 + 10 = 38$$

$$= 38/2 = \underline{19}.$$




→ Now consider (a, b, c) (a, b, d) (a, b, e)

level-2

$$(b,c) = (1+3) + (6+3) + (6+1) + (3+4) + (2+3)$$

$$= 4 + 9 + 7 + 7 + 5$$

$$= 32/2 = 16$$

$(b,c) = 16$

$$(b,d) = (1+3) + (7+3) + (1+2) + (7+3) + (2+3)$$

$$= 4 + 10 + 3 + 10 + 5$$

$$= 32/2 = 16$$

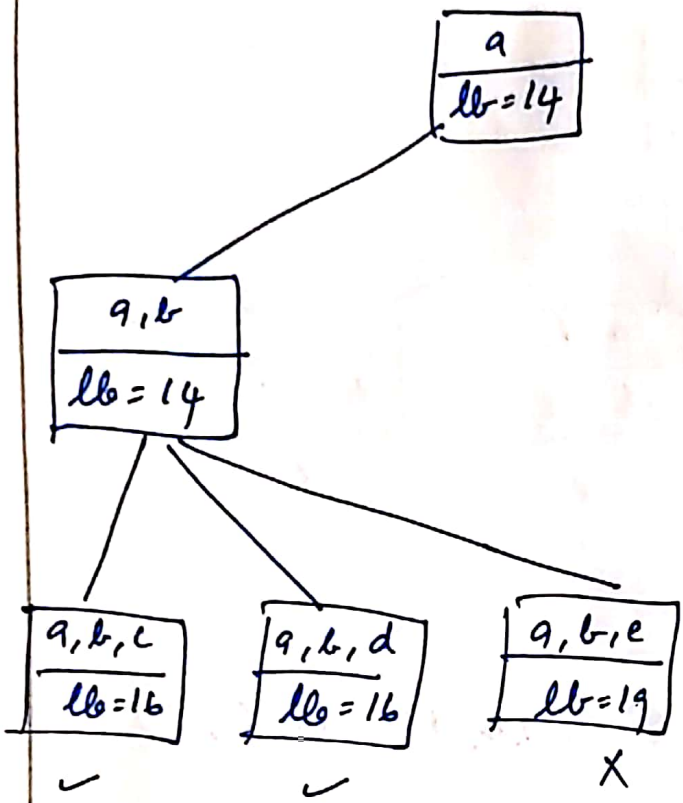
$(b,d) = 16$

$$(b,e) = (1+3) + (9+3) + (1+2) + (3+4) + (9+2)$$

$$= 4 + 12 + 3 + 7 + 11 = 37$$

$$= 37/2 \approx 19$$

$(b,e) = 19$




Now consider

(i) a, b, c, d, (e, a) (ii) a, b, c, e (d, a)

and

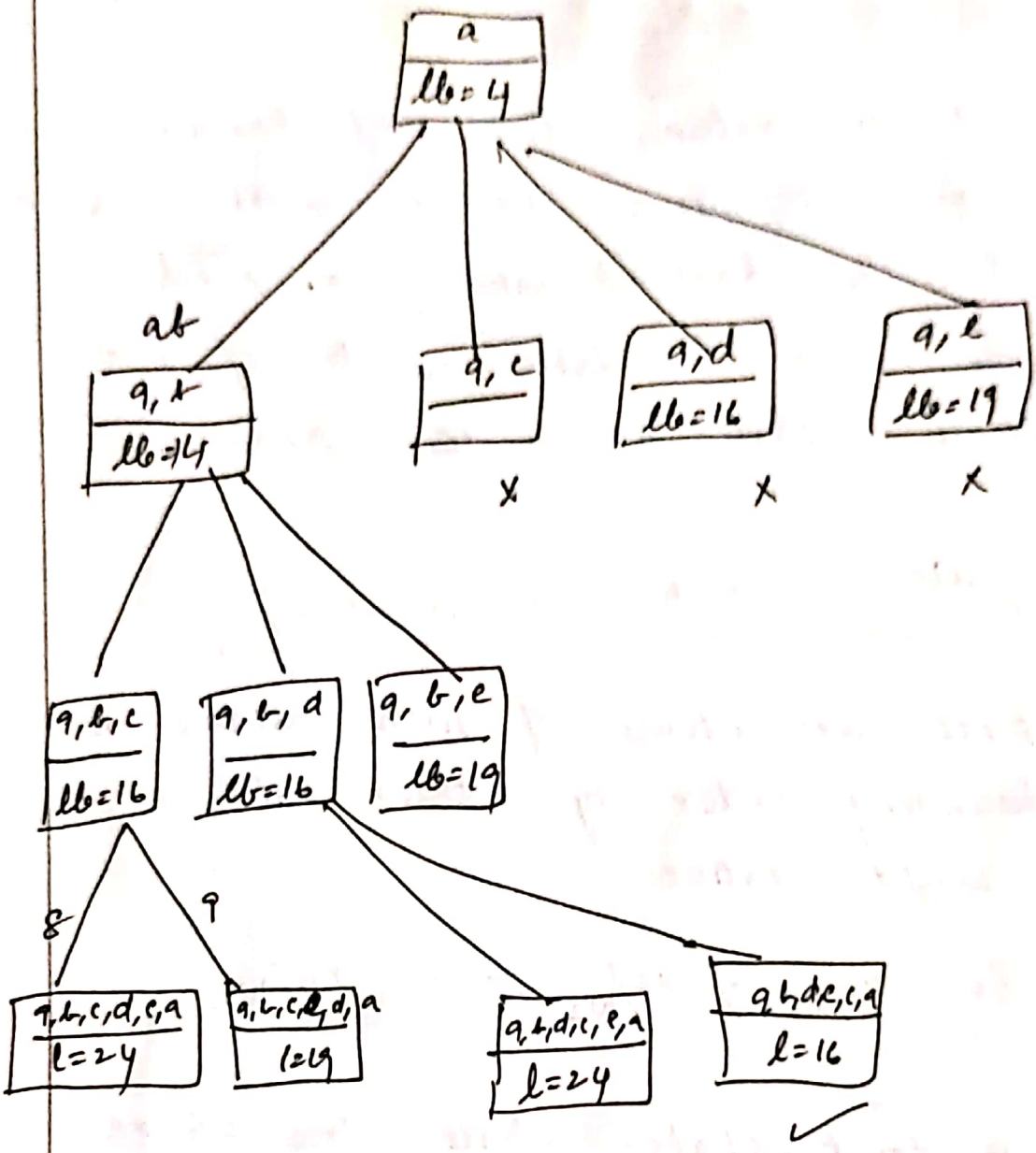
(i) a, b, d, e, (e, a) (ii) a, b, d, e, (c, a)

for a, b, c, d, e, a $l = 3+6+4+3+8 = 24$

a, b, c, c, d, a $l = 3+6+2+3+5 = 19$

a, b, d, c, e, a $l = 3+7+4+2+8 = 24$

a, b, d, e, c, a $l = 3+7+3+2+1 = 16$ ✓




optimal Tour = $a \to b \to d \to e \to c \to a$

optimal cost = 16

## 0/1 Knapsack problem:

Given $n$ instances / items of known weights $w_i$ and values $v_i$ for $i = 1, 2, \cdots n$ and the knapsack capacity $W$. find the most valuable subset of the items that fit in the knapsack.

$$ub = v + (W - w)(v_{i+1} / w_{i+1})$$

- First sort items of given instance in descending order by their value to weight ratio's:

  i.e $v_1/w_1 \geq v_2/w_2 \geq \cdots v_n/w_n$

- Construct a state-space tree where each node on the $i^{th}$ level of this tree $0 \leq i \leq n$ represents all the subsets of $n$ items.

Obtain the optimal solution for the following instance of knapsack problem using branch & bound technique. Draw the state space tree:

| item | weight | value | value/weight |
|---|---|---|---|
| 1 | 4 | 40 | 10 |
| 2 | 7 | 42 | 6 |
| 3 | 5 | 25 | 5 |
| 4 | 3 | 12 | 4 |

$W = 10$

$$ub = v + (W - w)\ (v_{i+1}/w_{i+1})$$

- Initial state

$$ub = 0 + (10 - 0)\ 10 = 100$$

Consider $i = 1$

with item = 1

$V = 40,\ w = 4$

$$ub = 40 + [10 - 4]\left[\frac{42}{7}\right]$$

$$= 40 + 6 * 6$$

$ub = 76$ ✓ Greater, so include item1

w/o item = 1

$V = 0,\ w = 0$

$$ub = 0 + 10\left(42/7\right)$$

$$= 0 + 60$$

$$= 60$$

Consider $i=2$, with item 1

with item 2

$V = 42 + 40 = 82$

$w = 7+4 = 11$

$W = 10$

Not feasible

$w > W$

w/o item 2

$v = 40$, $w = 4$

$ub = 40 + (10-4)\left(\frac{v_3}{w_3}\right)$

$= 40 + 6 * 5$

$= 40 + 30 = 70$

$ub = 70$

Consider $i=3$, w/o item 2

with item = 3

$V = 25 + 40 = 65$

$w = 5+4 = 9$

$W = 10$

$ub = 65 + 1 * (v_4/w_4)$

$= 65 + 4 = 69$

w/o item 3

$v = 40$

$w = 4$

$ub = 40 + (10-4)(v_4/w_4)$

$= 40 + 6 * 4$

$= 40 + 24 = 64$

consider $i = 4$

(i) with item 4

$V = 12 + 65 = 77$

$W = 9 + 3 = 12$

$W = 10$

$w > W$

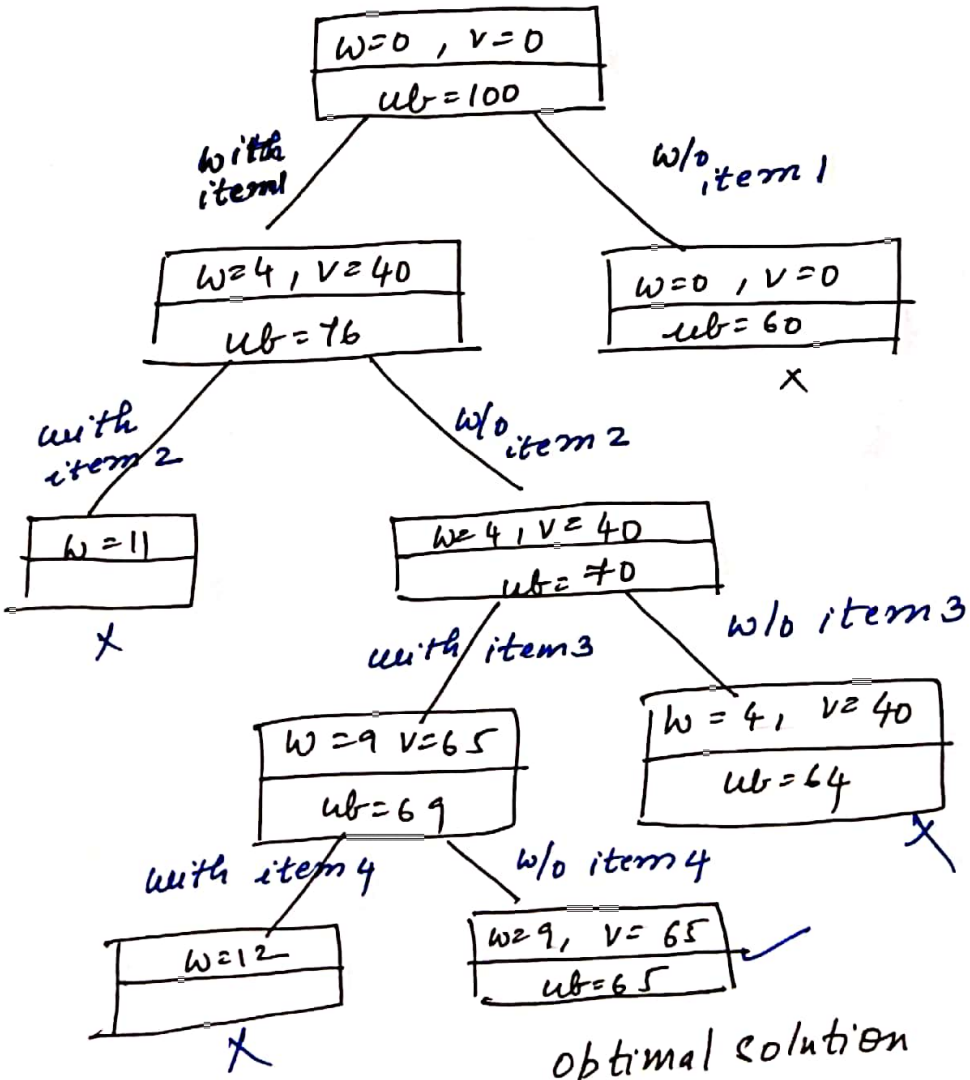
$\therefore$ Not feasible

(ii) w/o item 4

$V = 65$ , $W = 4 + 5 = 9$

$ub = 65$

$\therefore$ state space tree is




$\therefore$ Solution vector = {1, 3}, optimal solution = 65

* with the help of state space tree, solve the following instance of knapsack problem by branch & bound algorithm

| Item | weight | value | |
|---|---|---|---|
| 1 | 10 | 100 | |
| 2 | 7 | 63 | |
| 3 | 8 | 56 | |
| 4 | 4 | 12 | $W = 16$. |

## LC Branch and Bound solution:

**Live Node:** Is a node that has been generated but whose children have not yet been generated.

**E-Node:** Is a live node whose children are currently being explored

**Dead-Node:** Is a generated node, that is not to be expanded/explored any further

solve the following instance of knapsack problem using LC Branch & Bound solution method. Draw the state space tree

| Item | profit | Weight |
|---|---|---|
| i | Pi | Wi |
| 1 | 10 | 2 |
| 2 | 10 | 4 |
| 3 | 12 | 6 |
| 4 | 18 | 9 |

Max capacity = 15

steps to be followed for LCBB sol^n are:

1. Draw state space tree
2. compute $\hat{c}(\cdot)$ and $u(\cdot)$ for each node

$\hat{c}(x)$ → Approximation cost used for computing the least cost $c(x)$

$u(x)$ → denotes upper Bound.

3. If $\hat{c}(x) >$ upper Bound, Kill Node x

4. otherwise minimum cost $\hat{c}(x)$ becomes E-Node. Generate Children for E-Node.

5. Repeat Step 3 & 4 untill all the nodes get covered.

6. Minimum cost $\hat{c}(x)$ becomes the answer node. Trace the path in backward direction from x to root for solution subset.

Sol/n → First calculate least cost & upper bound for root node. Root node ie node 1

$$u(1) = -\sum p_i = -(10+10+12) = -32$$

Item 4 is not select, as it excede capacity.

$$\hat{c}(1) = u(1) - \left[\frac{m - \text{current total wt}}{\text{Actual weight of remaining object}}\right] * \left[\text{Actual profit of remaining object}\right]$$

$$= -32 - \left[\frac{15 - (2+4+6)}{9}\right] * 18$$

$$= -32 - \frac{3}{9} \times 18 = \underline{\underline{-38}}$$

with item 1

$u(2) = -\sum p_i - (10+10+12)$

$U(2) = -32$

$\hat{c}(2) = -38$

w/o item 1

$U(3) = -\sum p_i$

$U(3) = -(10+12) = -22$

$\hat{c}(3) = -22 - \left(\frac{15-10}{7}\right) * 18$

$= -22 - \frac{5}{7} * 18$

$\hat{c}(3) = -32$

$u(2) \geq u(3)$

∴ Branching take place at node 2 i.e with item 1 as it has least upper bound.

with item 2

$U(4) = -\sum p_i$

$= -(10+16+12)$

$U(4) = -38$

$\hat{c}(4) = -38$

w/o item 2

$U(5) = -\sum p_i$

$= -(10+12) = -22$

$\hat{c}(5) = -22 - \left[\frac{15-8}{7}\right] * 18$

$= -22 - \left(\frac{7}{7}\right) * 18$

$= -22 - 14 = -36$

$U(4) > U(5)$

∴ Branching take place with item 2

node 6

with item 3

$U(6) = -\sum p_i$

$U(6) = -32$

$\hat{c}(6) = -32 - \left(\frac{15-12}{9}\right) * 18$

$= -32 - 3(2)$

$\hat{c}(6) = -38$

node 7

w/o item 3

$U(7) = -\sum p_i$

$U(7) = -38$

$\hat{c}(7) = -38 - \left(\frac{15-15}{6}\right) * 12$

$= -38 - 0$

$\hat{c}(7) = -38$

$U(6) < U(7)$

So branching take place at $U(7)$, w/o item3

with item 4

$U(8) = -\sum p_i = -(10+10+18)$

$= -38$

$\hat{c}(8) = -38 - \left[\frac{15-15}{6}\right] * 12$

$\hat{c}(8) = -38$

w/o item 4

$U(9) = -\sum p_i$

$U(9) = -(30+10) = -20$

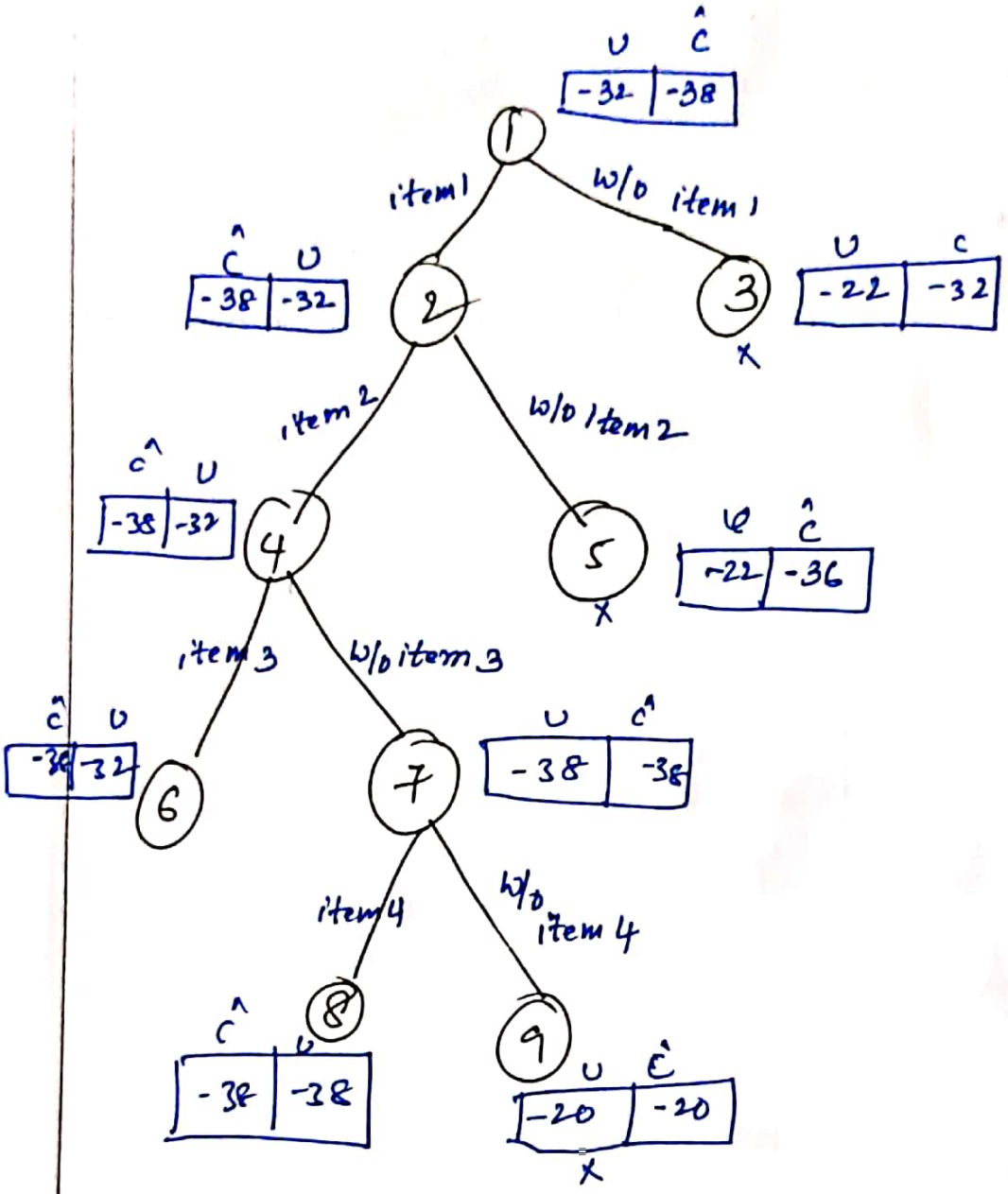
$\hat{c}(9) = -20 - \left(\frac{15-6}{9}\right) 0$

$= -20$

$U(8) > U(9)$

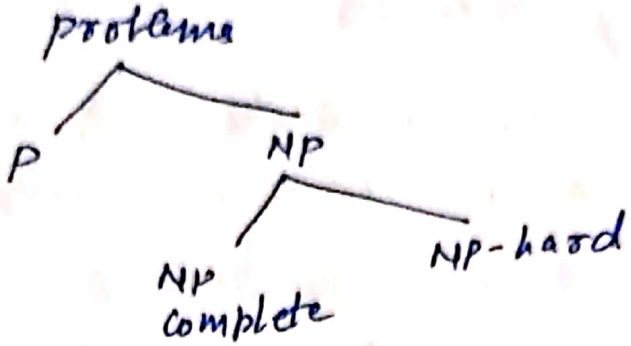
Solution vector = {1, 1, 0, 1}

Max profit = 38



# P, NP, NP complete & NP hard problems:

- The problems can be classified as 2 groups:




**P class**: set of all decision problems solvable by deterministic algorithms in polynomial time:

eg: searching, sorting.

**NP Class**: set of all decision problems solvable by non deterministic algorithms in polynomial time.

eg: TSP, Graph coloring, knapsack problem etc

$P \subseteq NP$.

**NP- Complete**:

A decision problem D is said to be NP-complete if

(i) It belongs to class NP

(ii) every problem in NP is polynomially reducible to D.

## NP-hard:

A problem $L$ is NP hard iff satisfiability reduces to $L$. A problem is NP complete iff $L$ is NP hard & $L \in NP$.

## Polynomially Reducible:

A decision problem $D_1$ is polynomially reducible to a decision problem $D_2$ if there exists a function $t$ that transforms instances of $D_1$ to instances of $D_2$ such that:

(i) $t$ maps all yes instances of $D_1$ to yes instances of $D_2$ & all no instances of $D_1$ to no instances of $D_2$

(ii) $t$ is computable by a polynomial time algo.

If problem $D_1$ is polynomially reducible to some problem $D_2$ that can be solved in polynomial time, then problem $D_1$ can also be solved in polynomial time.