Unit - 5

In greedy, after taking a decision, we can't go back & change, but in backtracking, we can revoke the decision taken. To solve backtracking, we use a tree (BT).

In a statespace tree which we construct in backtrack we have promising (optimal solution), or lead to final solution) and non-promising (not possible, to get solution).

Promising: leads to final solution or optimal solution. Can be non-leaf nodes as well as leaf nodes

Non-promising: which don't lead to the final solution, generally leaf nodes.

Problems under backtracking

Sum of subsets:

$S = \{1, 2, 3, 4, 5\}$

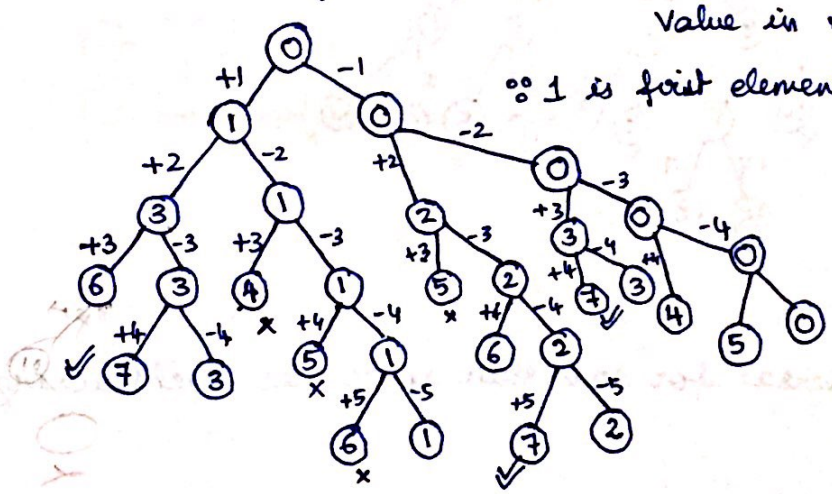
$d = 7$ (sum of subset = 7).

$S_1 = \{1, 2, 4\}$ $S_2 = \{3, 4\}$ $S_3 = \{1, 2, 4\}$.

Left: add current no., Right: don't add. (repd by subⁿ) –

Value in node = sum

∵ 1 is first element.



∴ 3 possible subsets.

The objective of the sum of subset problem is to find the subsets from a given set $S = \{S_1, S_2, S_3, S_4 \ldots S_n\}$ whose summation is equal to an integer D.

NOTE: The elements in set S should always be in increasing order.

Algorithm for sum of subsets problems:

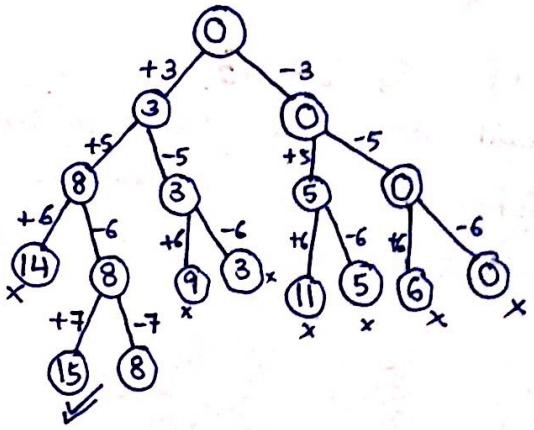
Algorithm Sum_of_Subsets(s, k, r) (s → initial sum, k → index of first element, r → remaining sum // initially 'd')

// Input: W[1...n] which holds the elements in the increasing order and d.

// Output: Subsets whose summation is d

```
x[k] = 1
if W[k] + s == d:
        write (x[1...n])
else if s + W[k] + W[k+1] <= d:
        Sum_of_Subsets(s + W[k], k+1, r - W[k])
else if s + r - W[k] >= d and s + W[k+1] <= d:
        x[k] ← 0
        Sum_of_Subsets(s, k+1, r)
```

Q. Construct a state space tree for the sum of subsets problem with inputs as $S = \{3, 5, 6, 7\}$, $d = 15$

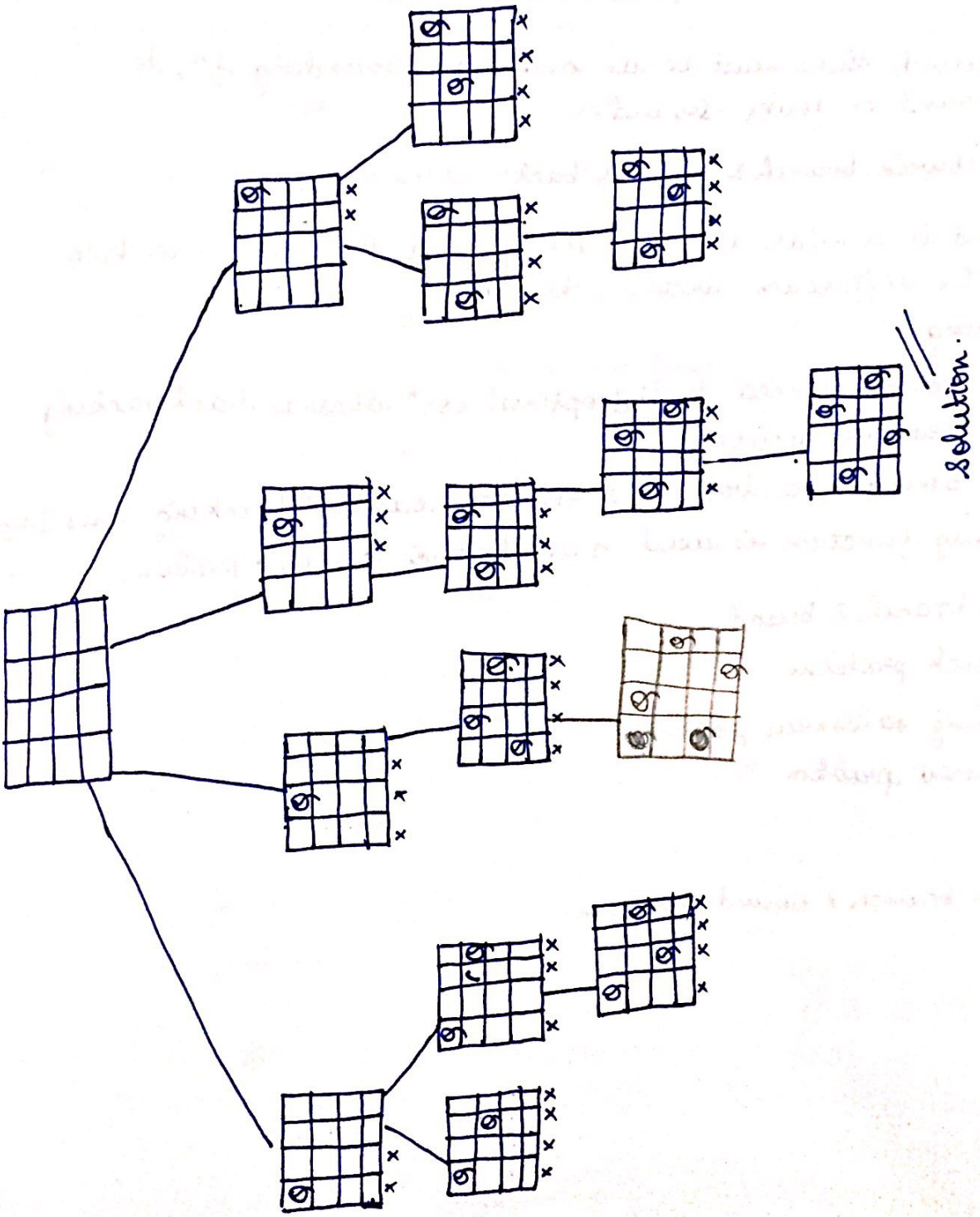


Use DFS traversal for state space tree backtracking.

N-Queen's problem.

Objective of problem is to place n-queens on a chessboard of dimension n x n such that no two queens attack each other. Queens attack each other if they are in same row, same column, or same diagonal to each other.

Statespace tree for 4 queens.



Branch & Bound

Backtracking cannot be used to get optimal solution, only gives feasible soln.

This is fixed / improved by using branch & bound, to get optimal solution.

In backtracking, DFS used to get all feasible soln. Uses a stack.

In branch & bound, both DFS & BFS traversal used to get optimal soln. ∴ queue may be used.

For branch & bound, there must be an associated bounding fn, to give upper bound or lower bound.

Common Diff. between branch & bound & backtracking.

Branch & bound is similar to backtracking w.r.t. the state space tree construction. The difference between two are:

- ~~Backtracking~~
- Branch & bound is used to find optimal soln whereas backtracking gives all feasible solutions.
- Branch & bound uses both DFS & BFS, whereas backtracking uses DFS
- A bounding function is used in all branch & bound problems

Problems under branch & bound

- Knapsack problem
- Travelling salesman problem.
- Assignment problem.

SEE 1 qn from branch & bound for 8m.

The objective of knapsack is to maximize the profit & it is an upper-bound function.

Here, we should always order the items in the descending order of their profits to weight ratio. i.e, $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \frac{v_3}{w_3} \ldots \frac{v_n}{w_n}$.

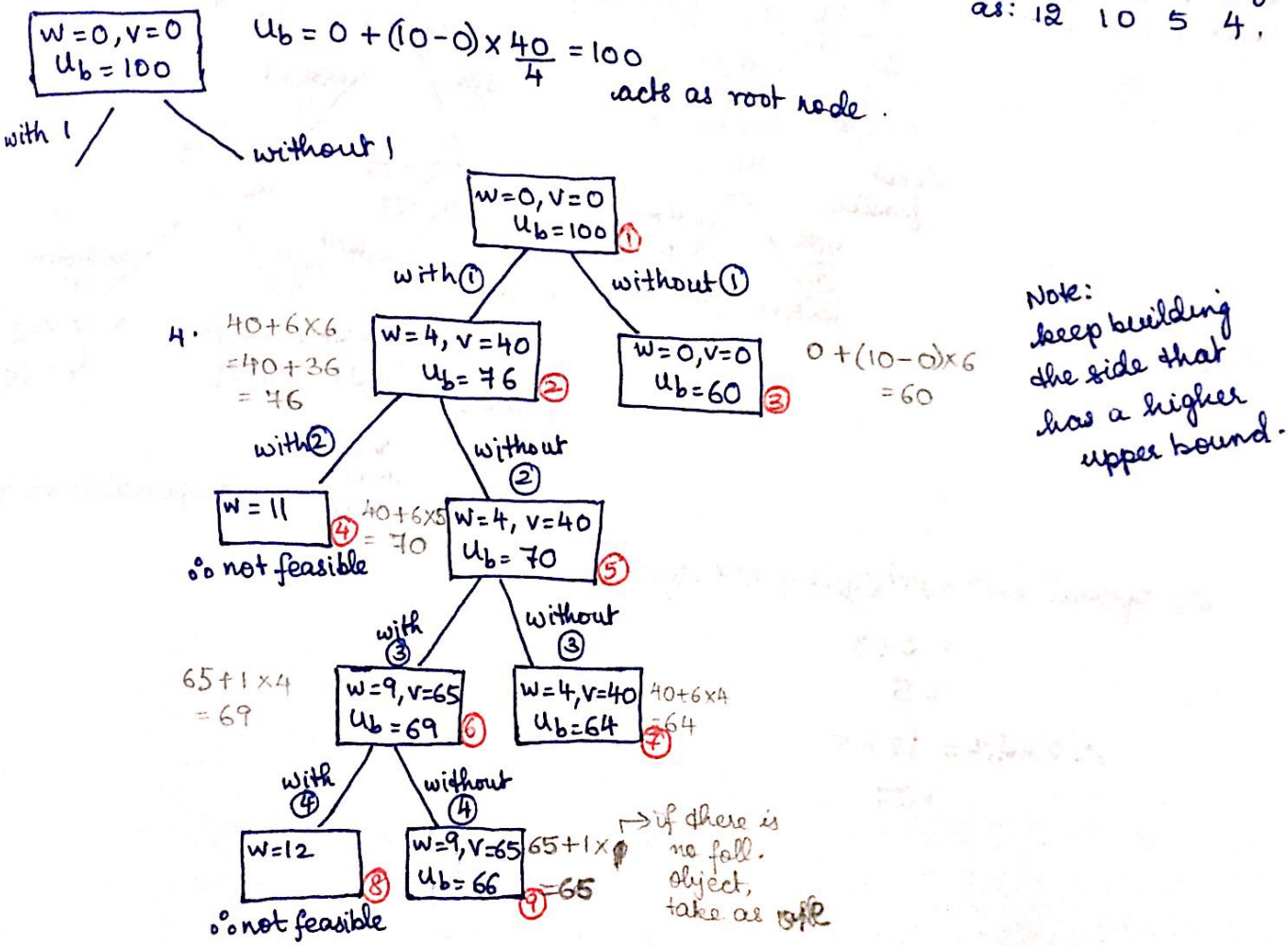
The objective to order the profits with weight is:
if the first item in the order is placed, we get the maximum profit & placing the last object, we get the least profit.

Let M be the knapsack capacity; $V = \{v_1, v_2, v_3 \ldots v_n\}$ & $W = \{w_1, w_2, w_3 \ldots w_n\}$ denotes the corresponding profits & weights of 'n' objects. The upperbound of the knapsack problem is calculated using $u_b = v_i + (M - w_i)\frac{v_{i+1}}{w_{i+1}}$

↳ profit : wt. ratio of next object.

Solve the following instance of knapsack using branch & bound.

M = 10
n = 4
w = 4 7 5 3
v = 40 42 25 12
$\frac{v}{w}$ = 10 6 5 4 } already in descending. If obj 10 12 5 4, arrange as: 12 10 5 4.



Node 7 is inferior to node 9, ∴ do not continue. (level 3 vs. level 4)
Node 3 is inferior to node 9, ∴ do not continue. (level 1 vs level 4)

'inferior', u_b of 9 > u_b of 7
u_b of 9 > u_b of 3

∴ Optimal soln: 1st object + 3rd object = 1+3

∴ Profit = 65

Solve the given instance of knapsack problem.

$M = 5$

$n = 4$

$w = \{ 2 \quad 1 \quad 3 \quad 2 \}$

$v = \{ 8 \quad 16 \quad 15 \quad 12 \}$

$\frac{v}{w} = 4 \quad 6 \quad 5 \quad 6$

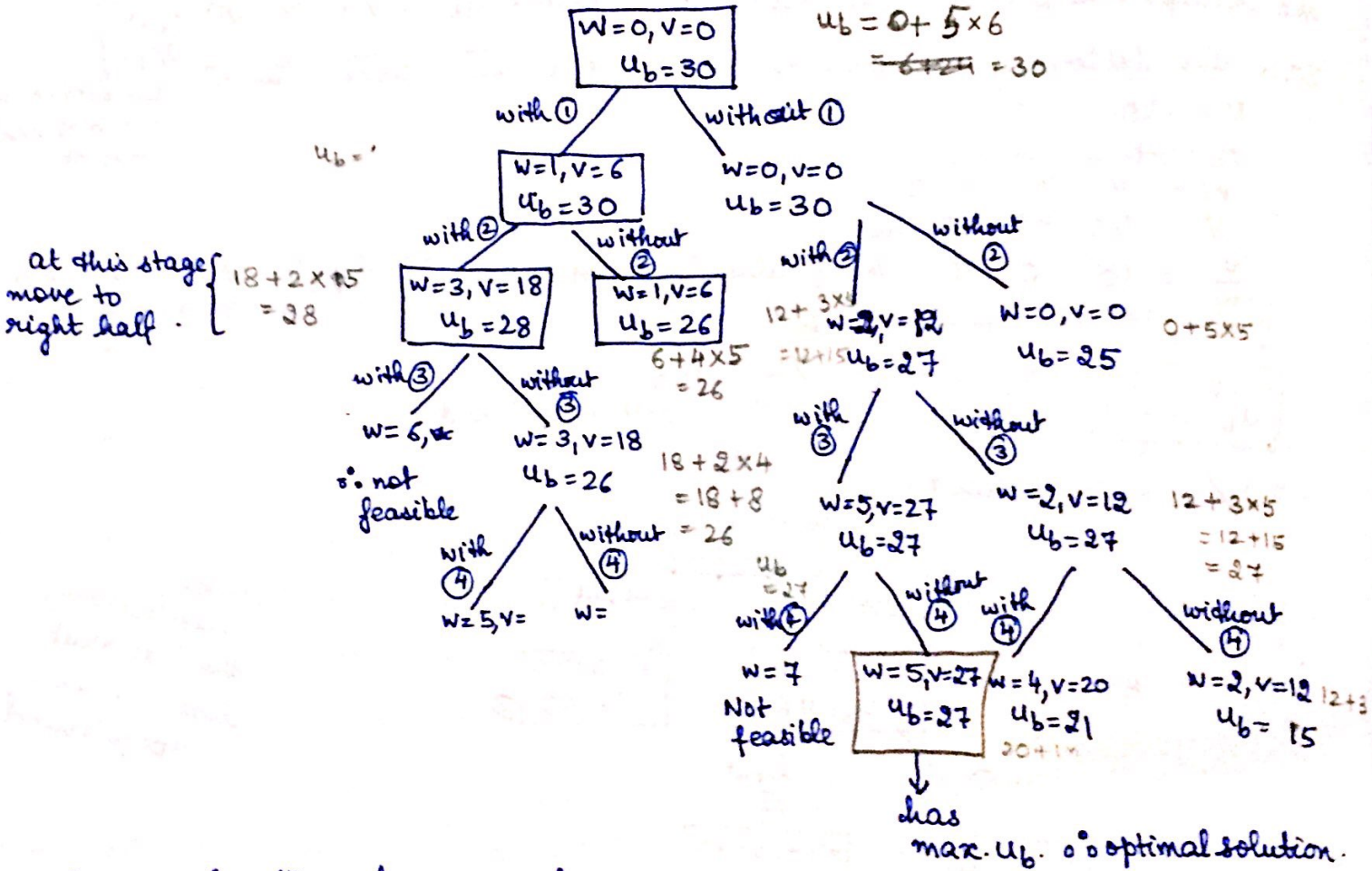
$w=0, v=0$ ~~$u_b =$~~

$\therefore$ order: $\frac{v}{w} = 6 \quad 6 \quad 5 \quad 4$

$w = 1 \quad 2 \quad 3 \quad 2$

$v = 6 \quad 12 \quad 15 \quad 8$

$u_b = v + (M - w) \cdot \frac{v_{i+1}}{w_{i+1}}$



∴ Optimal soln = 2nd object + 3rd object

= 2+3

= 5

∴ Profit = 12+15

= 27.

Assignment problem

The objective of the assignment problem is to assign 'n' jobs to 'n' processes (or people) such that the total cost of the assignment is as small as possible.

processors ↓

	J_1	J_2	J_3	J_4
P_1	9	2	7	8
P_2	6	4	3	7
P_3	5	8	1	8
P_4	7	6	9	4

} jobs.

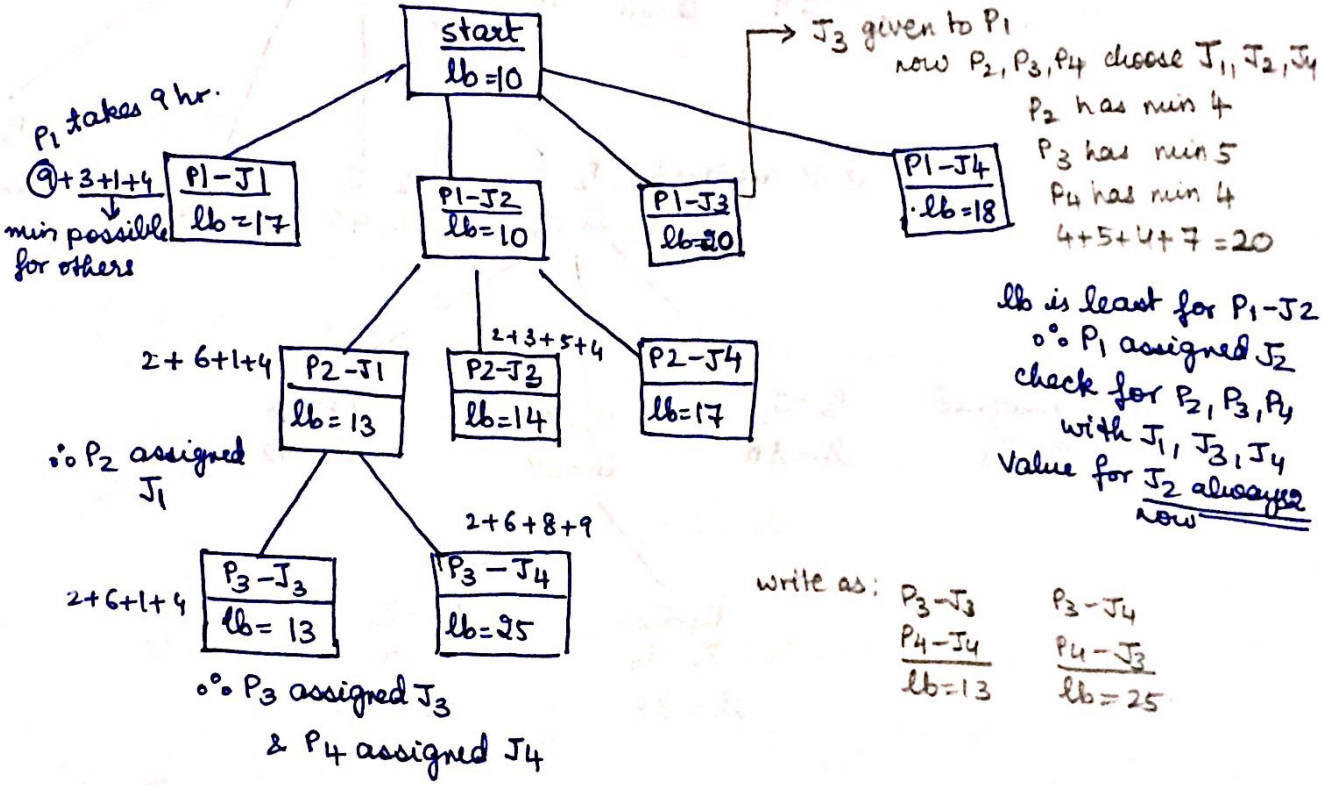
Time must be min.
Values given in hours

→ given input

Assignment problem is a lower-bound problem.

The lower bound for assignment problem is calculated as the summation of least value in each row.

For given e.g. lb = 2 + 3 + 1 + 4 = 10.



Process ↔ Job : 1:1 correspondence.

∴ Optimal solution:

$P_1 - J2$

$P_2 - J1$

$P_3 - J3$

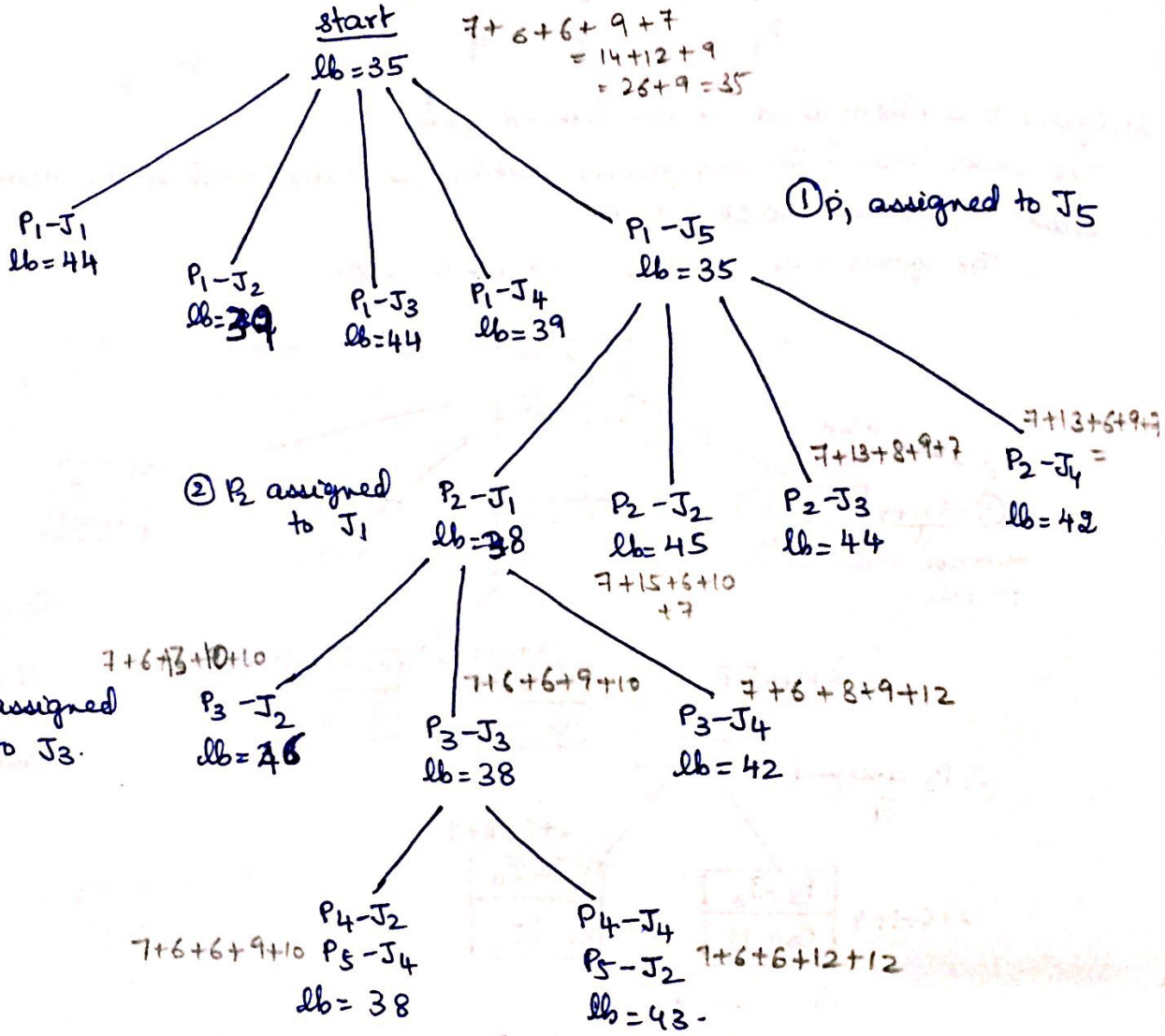
$P_4 - J4$

[Can find answer by obsⁿ in exam. Build tree & ensure they match].

Solve the given instance of assignment problem using branch & bound

	J_1	J_2	J_3	J_4	J_5
P_1	9	11	14	11	7 ①
P_2	6 ②	15	13	13	10
P_3	12	13	6 ③	8	8
P_4	11	9	10	12	9
P_5	7	12	14	10	14

$J_5 - P_1$
$J_1 - P_2$
$J_3 - P_3$
$J_2 - P_4$
$J_4 - P_5$



∴ optimal solution.

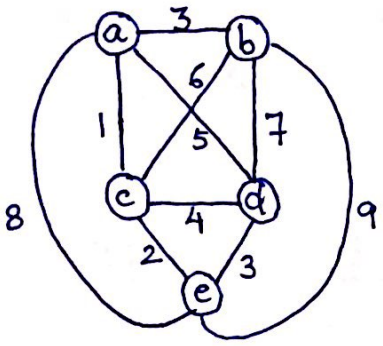
$P_1 - J_5$
$P_2 - J_1$
$P_3 - J_3$
$P_4 - J_2$
$P_5 - J_4$

Travelling Salesman Problem

The objective of the travelling salesman problem is to find a route such that a salesperson visits all the cities and returns back from the city where he started the tour with the minimum cost.

TSP problem is a minimization problem, and the lower bound is calculated as $lb = \lceil s/2 \rceil$, where 's' is the summation of the sum of two closest cities from each city.

Eg.



Assumption 1: 'a' is the starting city.

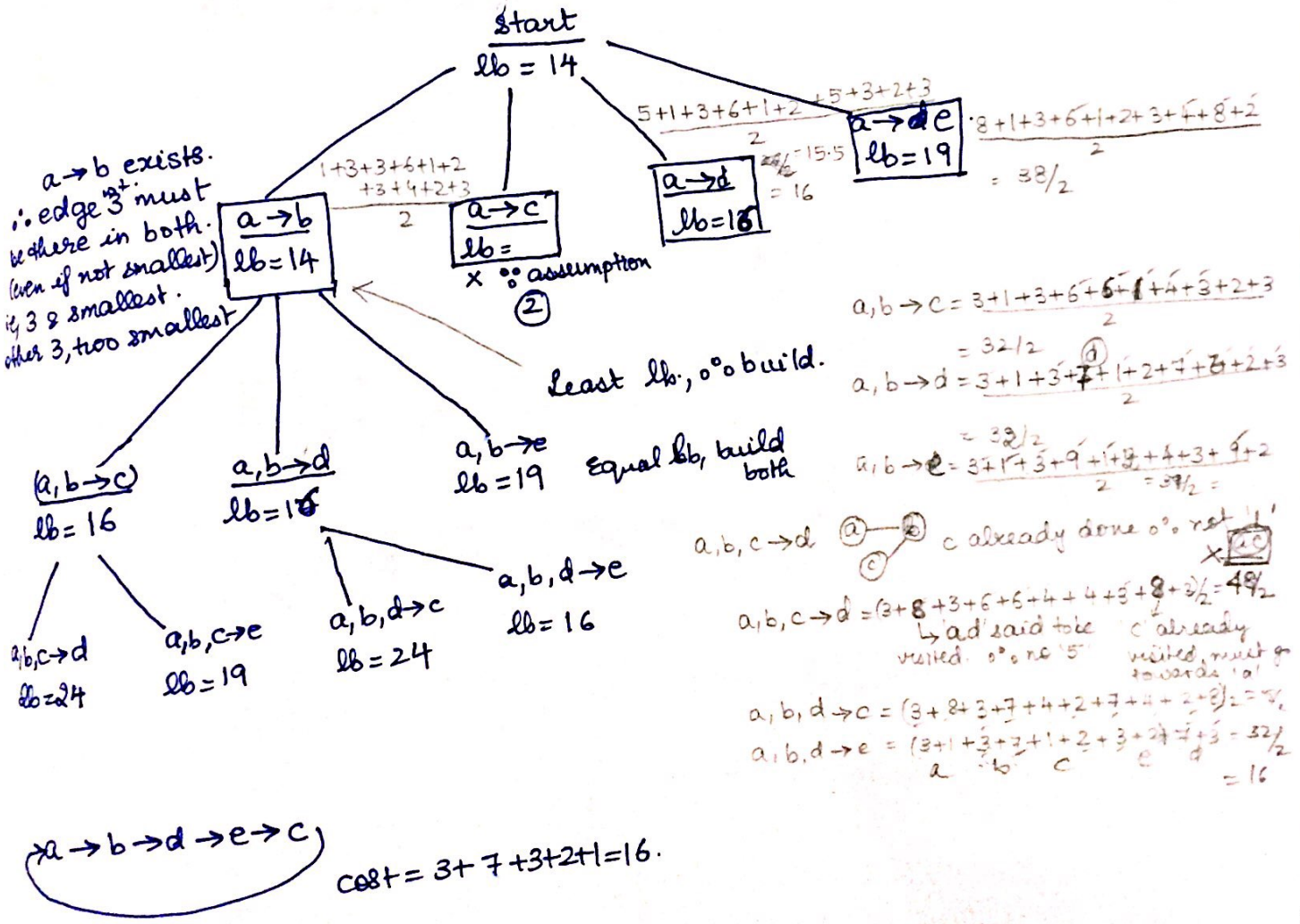
* Assumption 2: city 'b' is visited before city 'c'. ↑ optional. only used to set some cond° on soln

$lb = \lceil s/2 \rceil$

At start: for a, for b, for c, for d, for e

$= \frac{1+3+3+6+1+2+3+4+2+3}{2}$

$= 14$



$a,b \to c = \frac{3+1+3+6+6+1+4+3+2+3}{2} = 32/2$

$a,b \to d = \frac{3+1+3+7+1+2+7+3+2+3}{2} = 32/2$

$a,b \to e = \frac{3+1+3+9+1+2+4+3+9+2}{2} = 37/2$

$a,b,c \to d$ — c already done °° not 1' ×

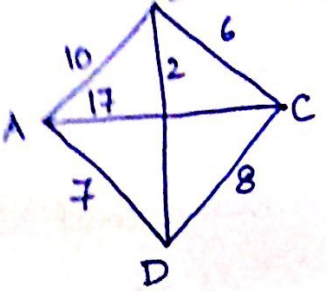
$a,b,c \to d = \frac{3+8+3+6+6+4+4+3+8+2}{2} = 49/2$

↳ 'd' said to be visited °° no '5' — c already visited, must go towards 'a'

$a,b,d \to c = \frac{3+8+3+7+4+2+7+4+2+3}{2} = 47/2$

$a,b,d \to e = \frac{3+1+3+7+1+2+3+2+7+3}{2} = 32/2 = 16$

(a → b → d → e → c) cost = 3+7+3+2+1 = 16.



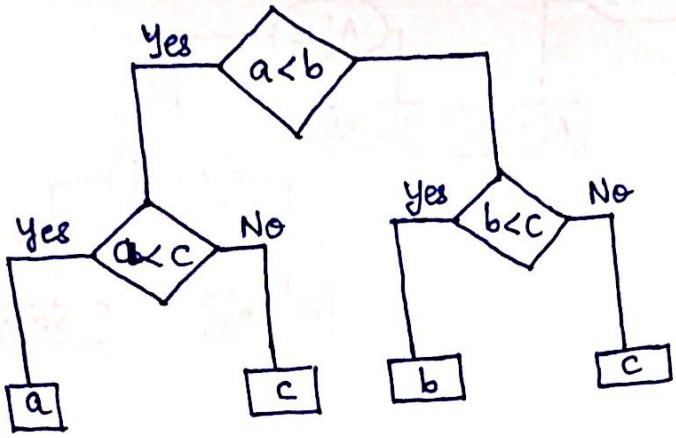
assume 'A' as start

Decision Tree

Decision trees are the devices used to check the performance of the searching & sorting algorithms. (Leaf nodes are decisions)

⇒ Decision tree to find the minimum of 3 elements.

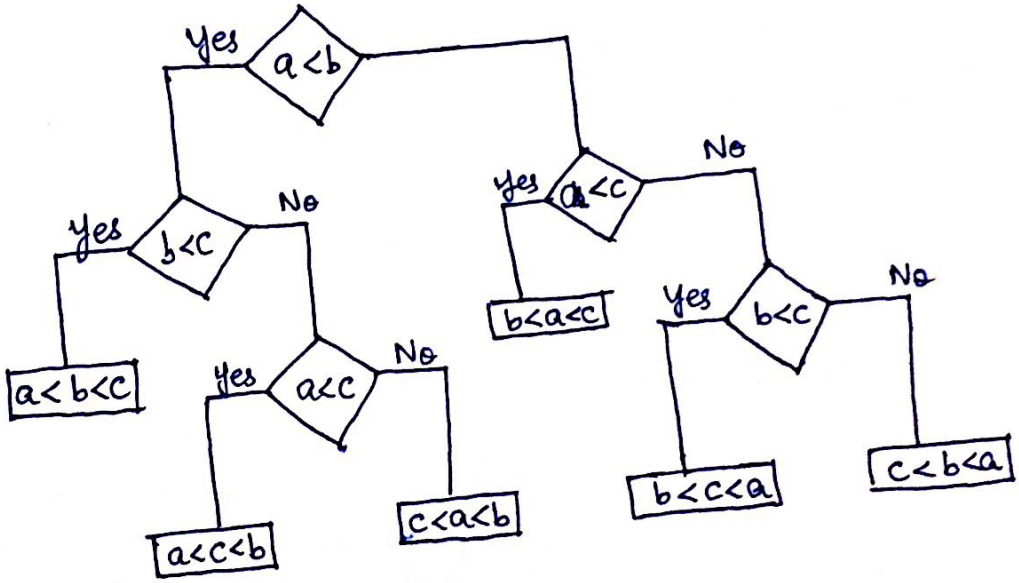
Let a, b, c be elements



If c is min, 2 DFS traversals performed.

⇒ Write a decision tree to perform 3-element insertion sort.

Let elements be a, b, c.



Construct a decision tree to perform binary search on 4 elements.
Elements stored in array A[], i.e, A[0], A[1], A[2], A[3].