SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

19ITB201 – DESIGN AND ANALYSIS OF ALGORITHMS

II YEAR IV SEM

UNIT-V-COPING WITH THE LIMITATIONS OF ALGORITHM

TOPIC: Branch and Bound- Assignment problem

Prepared by
T.Shanmugapriya,AP/IT

Branch and Bound

Branch and Bound

- It is an **optimization technique** to get an optimal solution to the problem.
- It is used for **solving combinatorial** problems.
- It looks for the **best solution** for a given problem in the entire space of the solution.
- Technique that applies where the greedy method and dynamic programming fail.

State Space Tree

- It traverses the state space tree by **BFS(Breadth First Search)** manner.

- But **not all nodes get expanded.**

Rather, a carefully **selected criterion** tells

- which node to expand
- **when to expand a node**
- when an optimal solution has been found.

State Space Tree –Terminologies

- **Problem state**: **Each node in the tree** defines a problem state.
- **State space**: **All paths from the root to other nodes .**
- **Solution states**: Solution states are those problem states s for which the **path from the root to s defines a tuple in the solution space.**
- **State space Tree**: **The tree organization** of the **solution space** is referred to as the state space tree.
- **Live Node**: **A node which has been generated** and all of whose **children have not yet been generated** is called a live node.
- **E-Node**: The live node **whose children are currently being generated** is called E-node.
- **Dead Node**: A dead node is generated node which has **not to be expanded further** or **all of whose children have been generated**

Backtracking	Branch & Bound
Algorithm for finding all solutions to some **computational problems**.	Algorithm for discrete and combinatorial **optimization problems.**
Finds the solution to the overall issue by finding a solution to the first sub problem and them recursively solving other sub problems based on the solution of the first issue.	Solves a given problem by dividing it into at least two new restricted sub problems.
Backtracking **is more efficient** than the Branch and Bound algorithm.	**Less efficient**
It traverses the state space tree by **DFS**(Depth First Search) manner.	It may traverse the tree in any manner, **DFS or BFS**.
It realizes that it has made a **bad choice & undoes** the last choice by **backing up.**	It realizes that it already **has a better optimal solution** that the pre-solution leads to so it abandons that pre-solution.
It **searches** the state space **tree until it has found a solution.**	It **completely searches the state space** tree to get optimal solution.
It involves **feasibility function.**	It involves a **bounding function**.

Job Assignment Problem

- It is a problem of assigning n people to n jobs so that the total cost of the assignment is as small as possible .

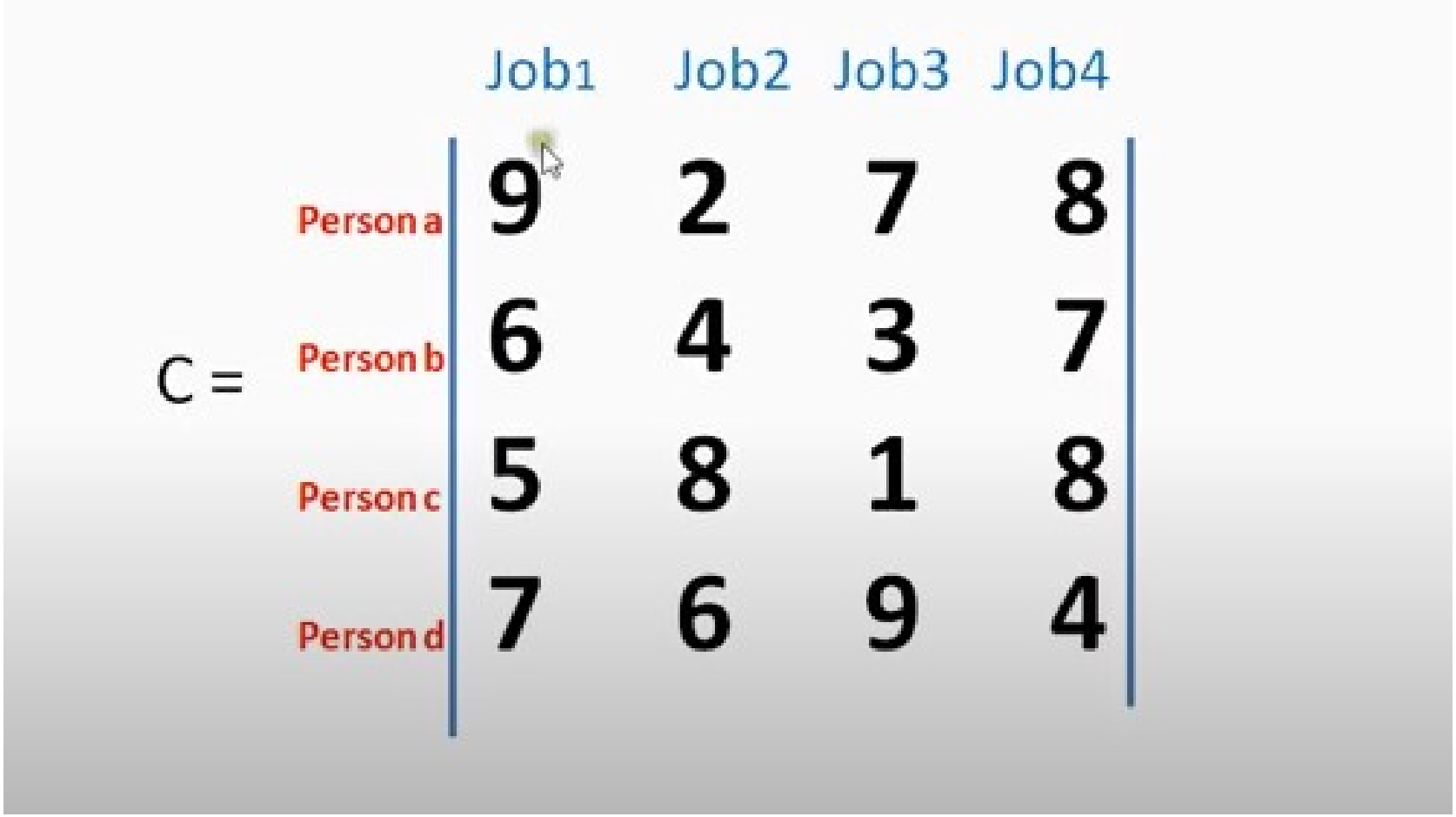
- Let there be n Person and n jobs.
- **Any person** can be **assigned to** perform **any job**.
- It is required to **perform all jobs .**
- Assigning **exactly one person to each job.**
- Assigning **exactly one job to each person .**
- **The total cost of the assignment** should be **minimum.**

Two approaches to calculate the cost function

- **For each person**, we choose job with minimum cost from list of unassigned jobs (take **minimum entry from each row).**

- **For each job**, we choose a person with lowest cost for that job from list of unassigned persons (take **minimum entry from each column).**

Solve the Job Assignment Problem for the below instance



Root Node-start

From each row select minimum cost of jobs.

For person **a** minimum job cost is **2**

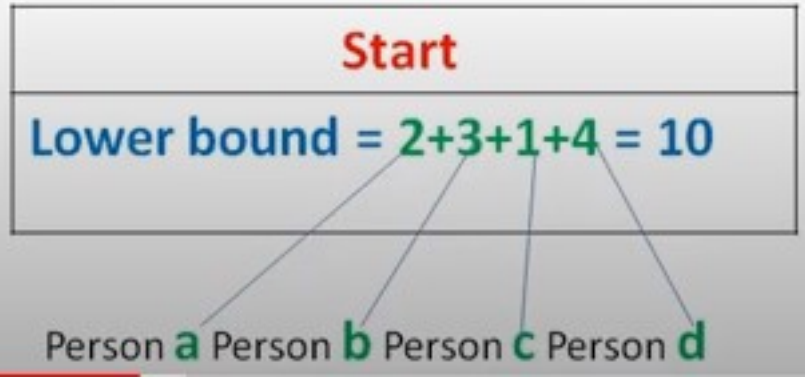
For person **b** minimum job cost is **3**

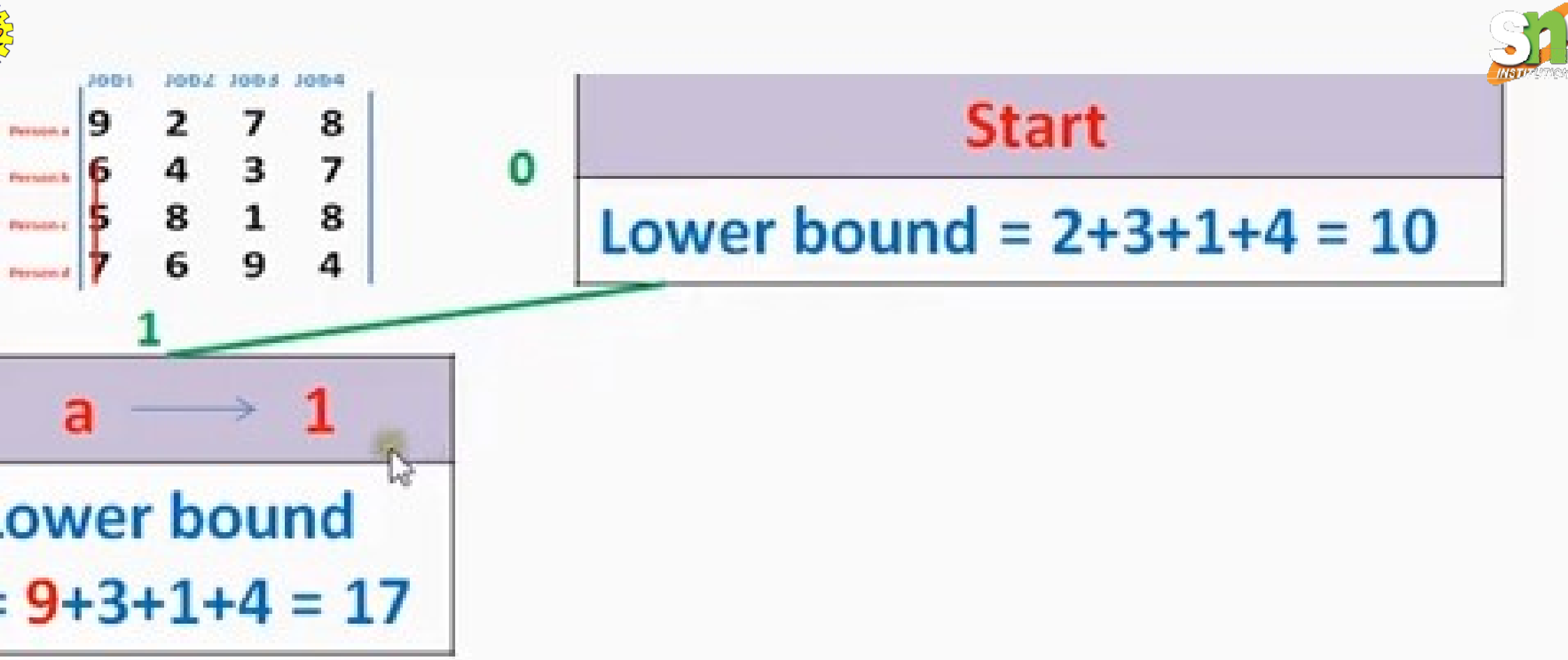
For person **c** minimum job cost is **1**

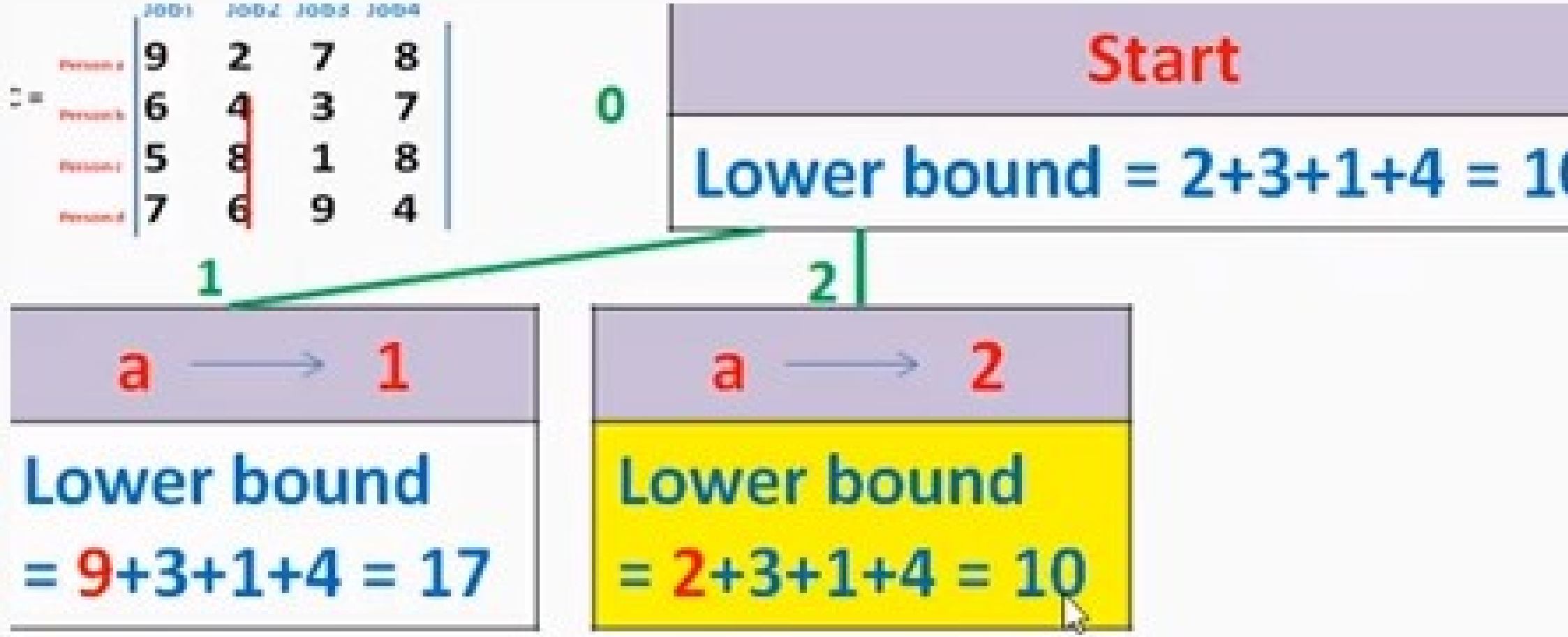
For person **d** minimum job cost is **4**

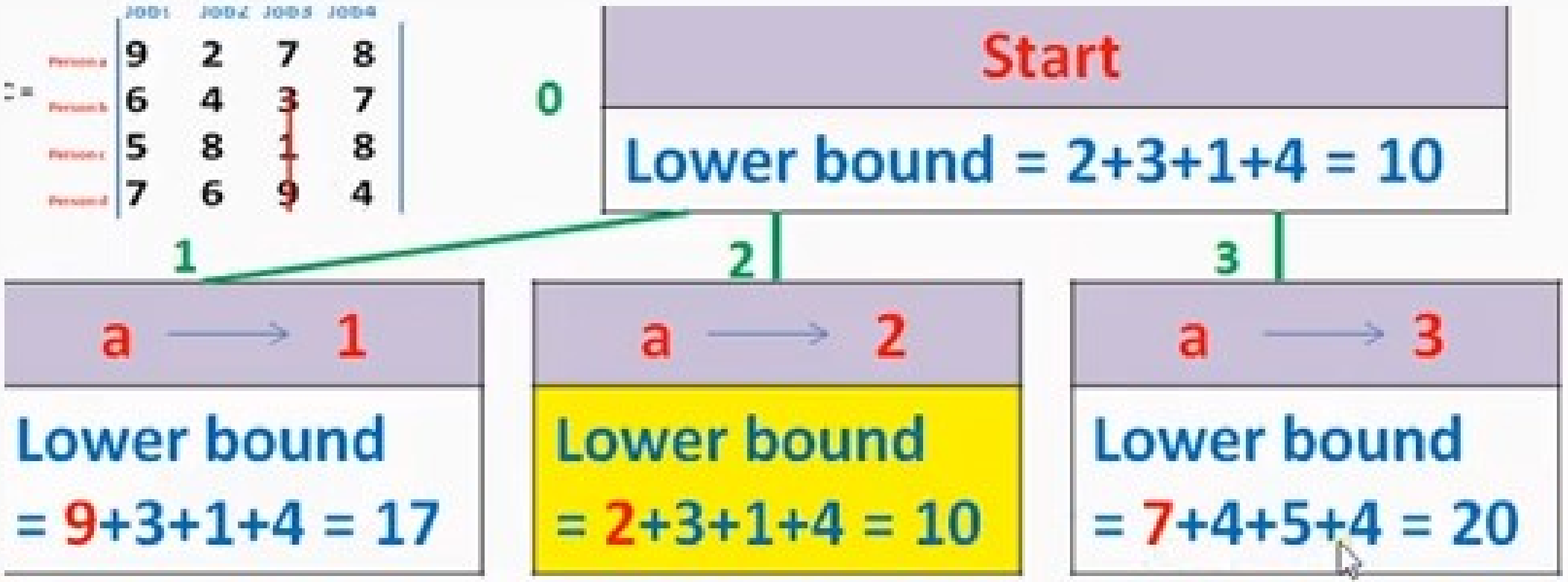
C =

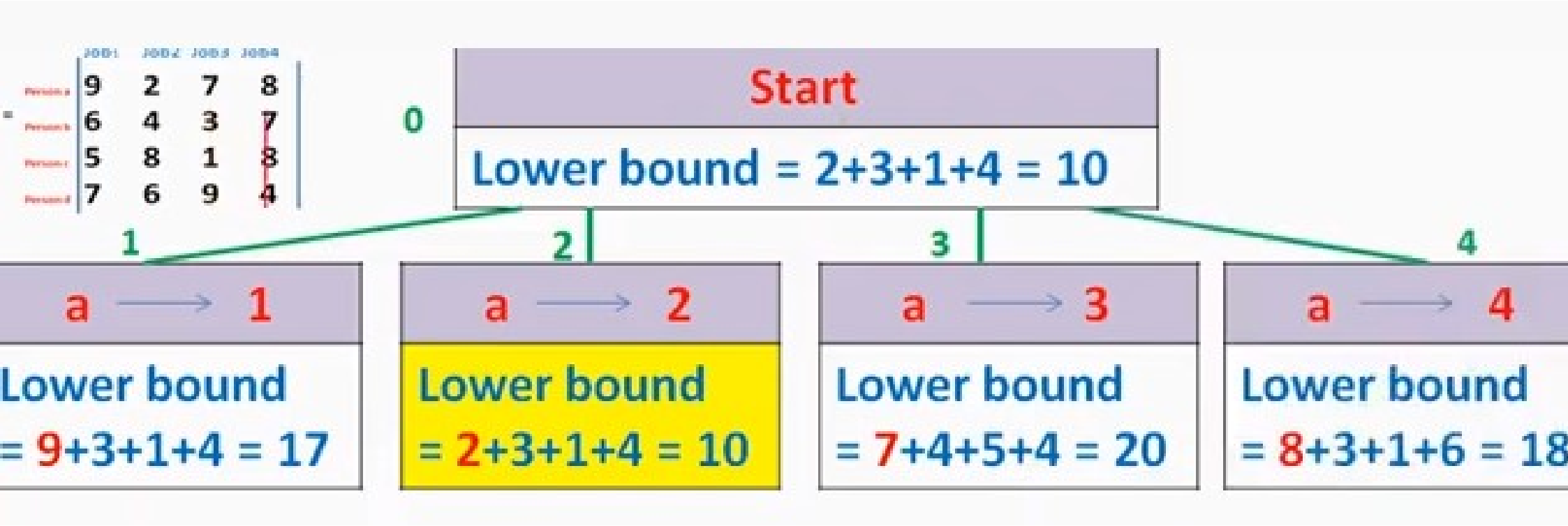
	Job1	Job2	Job3	Job4
Person a	9	2	7	8
Person b	6	4	3	7
Person c	5	8	1	8
Person d	7	6	9	4

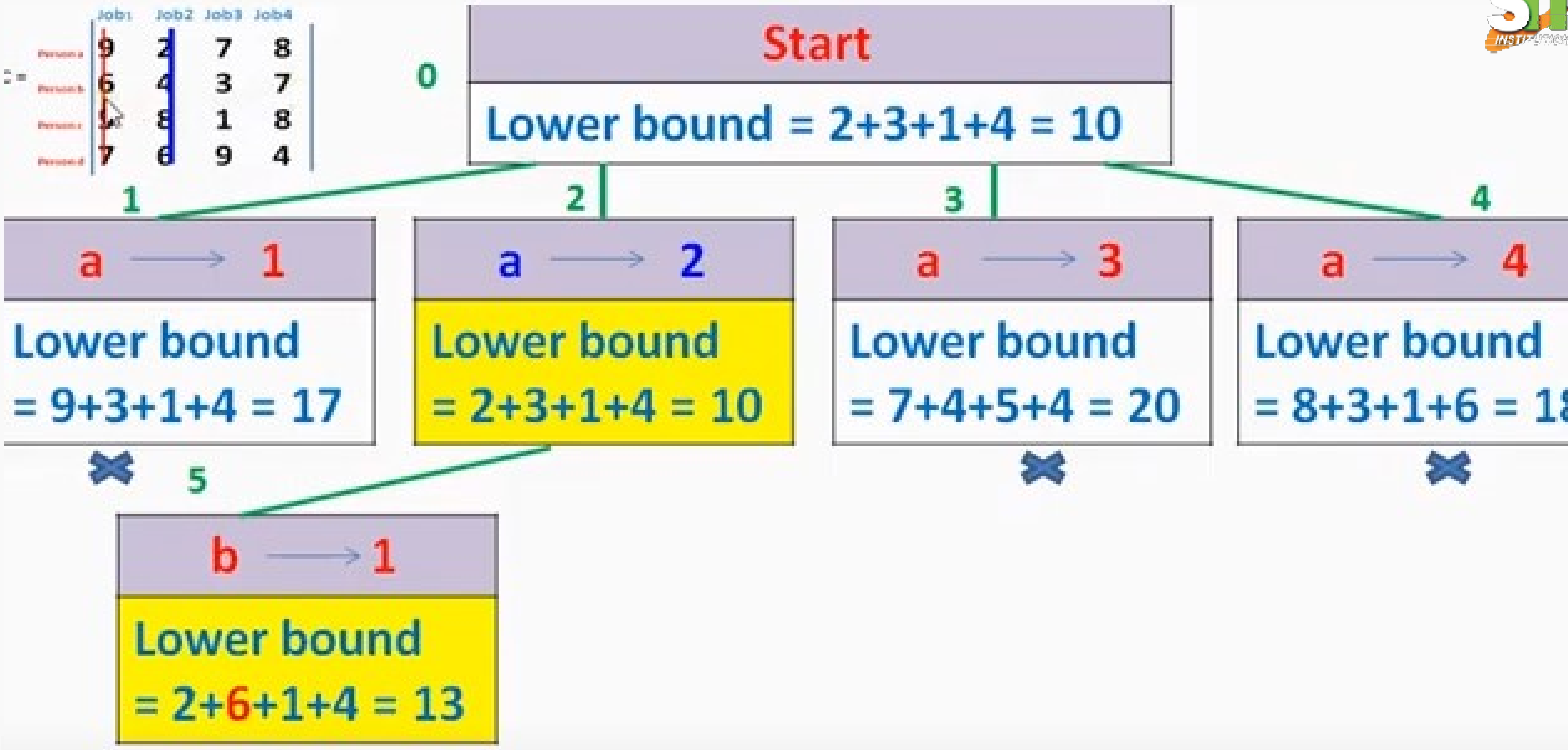


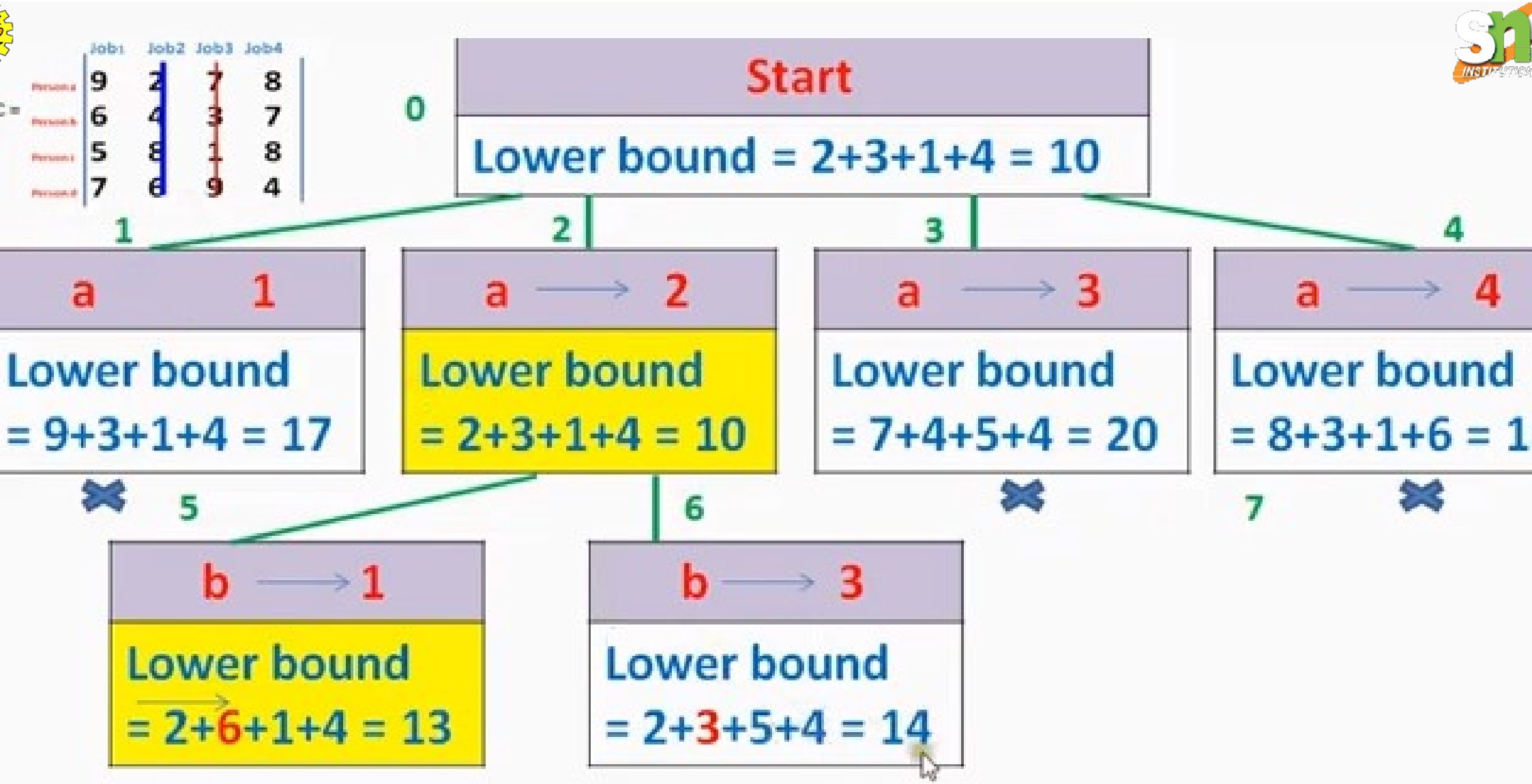

JOB1 JOB2 JOB3 JOB4
9 2 7 8
6 4 3 7
5 8 1 8
7 6 9 4
0
Start
Lower bound = 2+3+1+4 = 10
1
a 1
ower bound
9+3+1+4 = 17

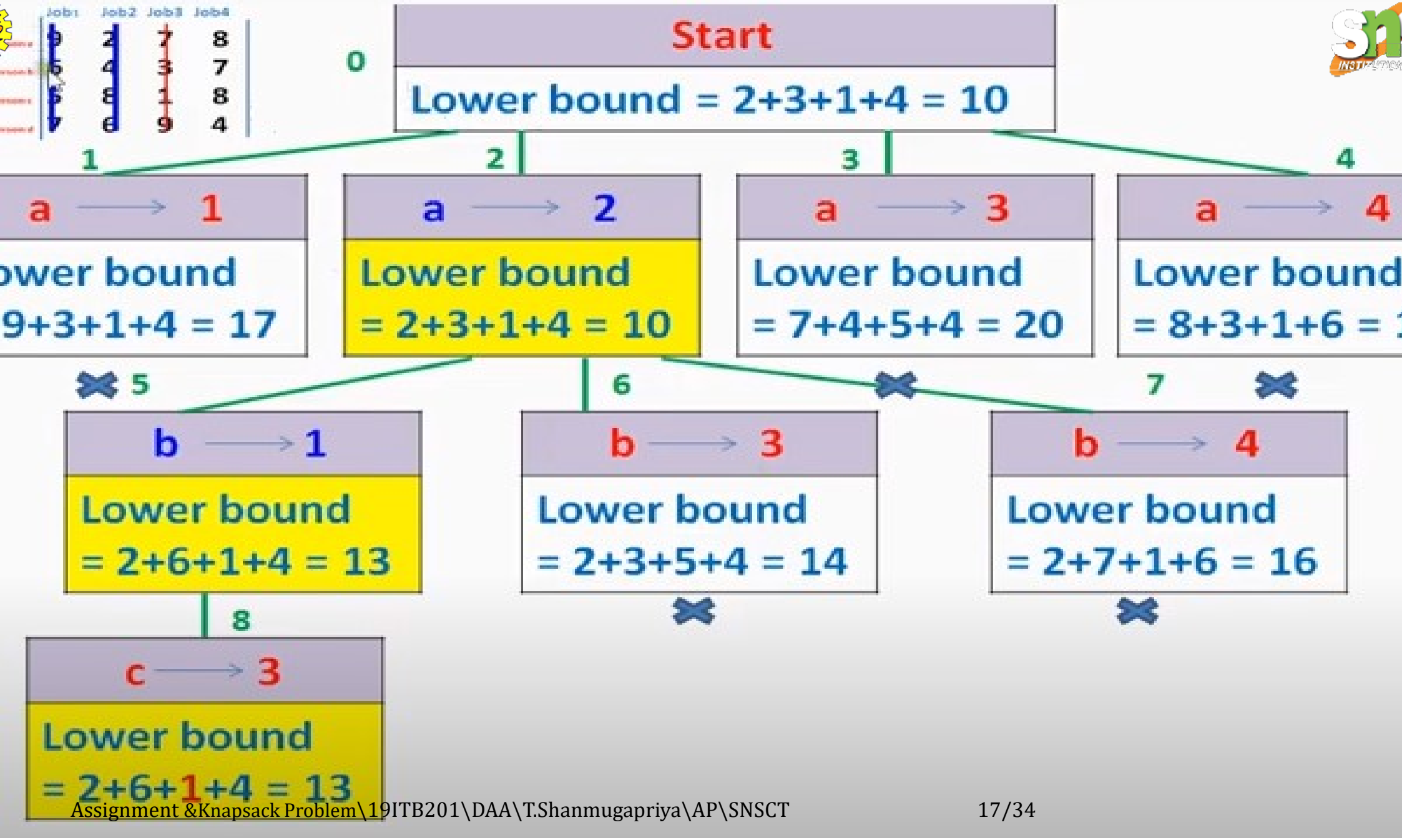

Job1 Job2 Job3 Job4
Person a 9 2 7 8
Person b 6 4 3 7
Person c 5 8 1 8
Person d 7 6 9 4
0
Start
Lower bound = 2+3+1+4 = 1
1
2
a → 1
Lower bound
= 9+3+1+4 = 17
a → 2
Lower bound
= 2+3+1+4 = 10

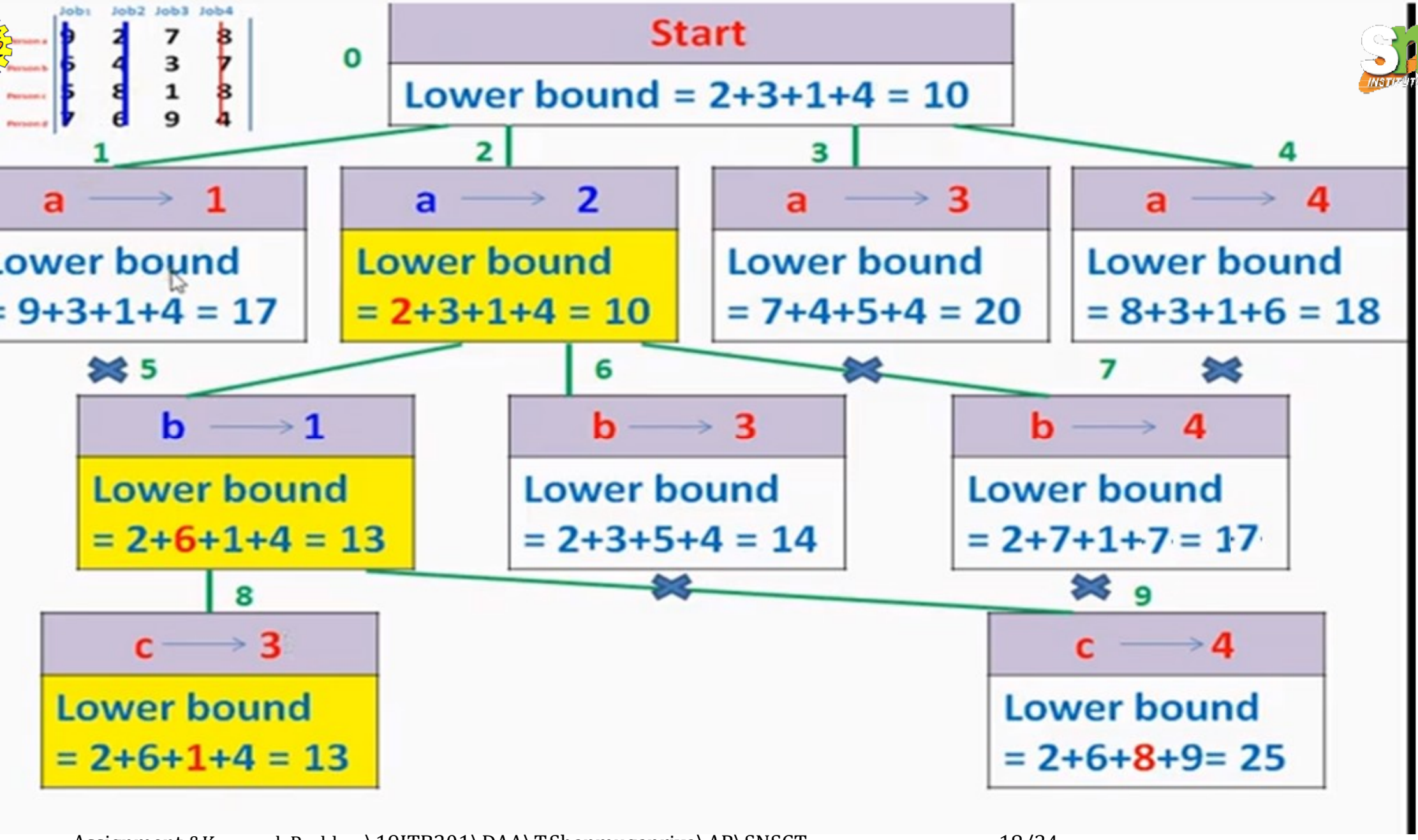

JOB1 JOB2 JOB3 JOB4
9 2 7 8
6 4 3 7
5 8 1 8
7 6 9 4
0
Start
Lower bound = 2+3+1+4 = 10
1
2
3
a → 1
Lower bound
= 9+3+1+4 = 17
a → 2
Lower bound
= 2+3+1+4 = 10
a → 3
Lower bound
= 7+4+5+4 = 20

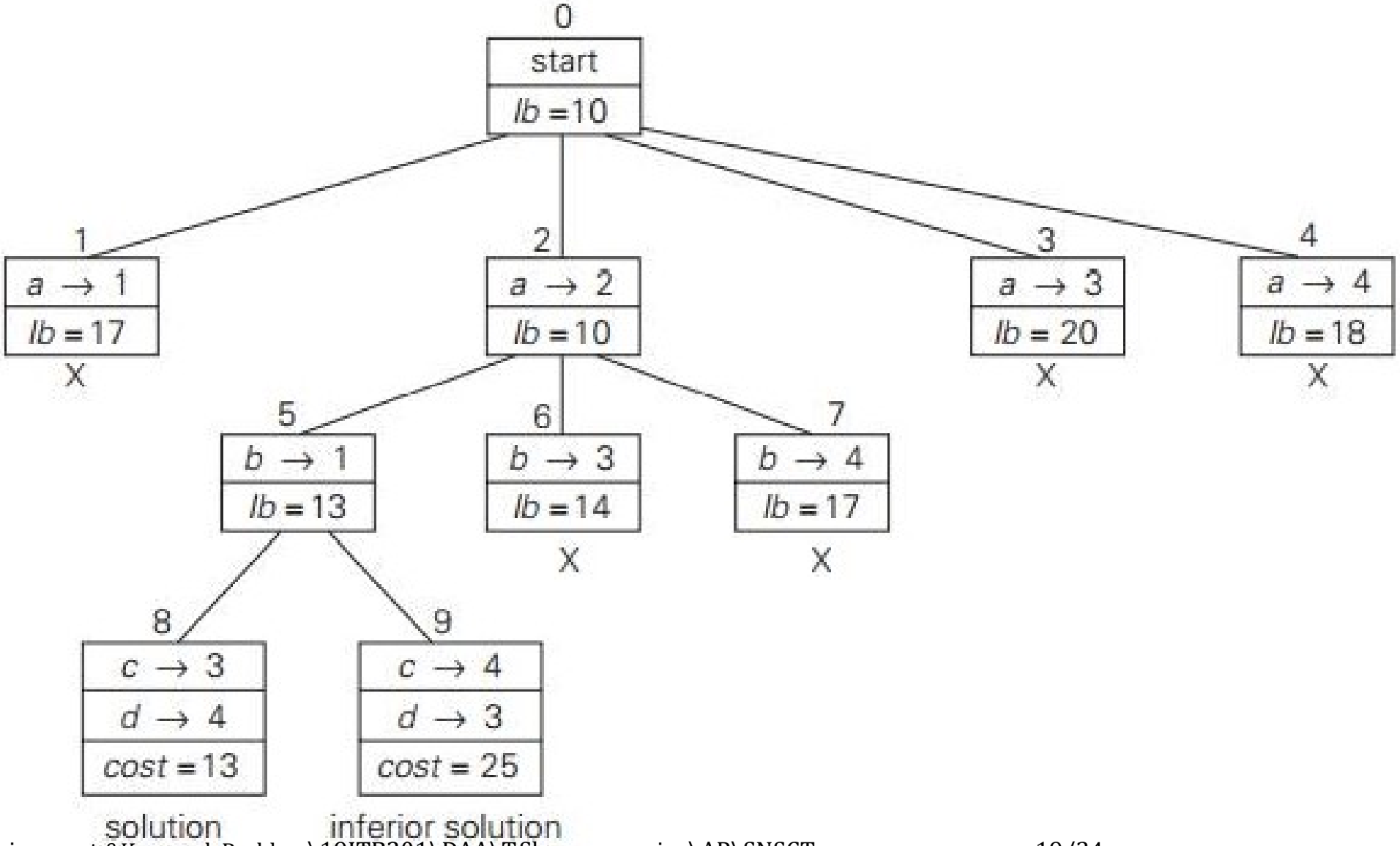

Job1 Job2 Job3 Job4
Person a 9 2 7 8
Person b 6 4 3 7
Person c 5 8 1 8
Person d 7 6 9 4
0
Start
Lower bound = 2+3+1+4 = 10
1
2
3
4
a → 1
Lower bound
= 9+3+1+4 = 17
a → 2
Lower bound
= 2+3+1+4 = 10
a → 3
Lower bound
= 7+4+5+4 = 20
a → 4
Lower bound
= 8+3+1+6 = 18


Job1 Job2 Job3 Job4
9 2 7 8
6 4 3 7
5 8 1 8
7 6 9 4
0
Start
Lower bound = 2+3+1+4 = 10
1
2
3
4
a → 1
Lower bound
= 9+3+1+4 = 17
a → 2
Lower bound
= 2+3+1+4 = 10
a → 3
Lower bound
= 7+4+5+4 = 20
a → 4
Lower bound
= 8+3+1+6 = 18
5
b → 1
Lower bound
= 2+6+1+4 = 13


Job1 Job2 Job3 Job4
9 2 7 8
6 4 3 7
5 8 1 8
7 6 9 4
0
Start
Lower bound = 2+3+1+4 = 10
1
2
3
4
a 1
Lower bound
= 9+3+1+4 = 17
a → 2
Lower bound
= 2+3+1+4 = 10
a → 3
Lower bound
= 7+4+5+4 = 20
a → 4
Lower bound
= 8+3+1+6 = 1
5
6
7
b → 1
Lower bound
= 2+6+1+4 = 13
b → 3
Lower bound
= 2+3+5+4 = 14


Job1 Job2 Job3 Job4
2 7 8
6 4 3 7
5 8 1 8
7 6 9 4
0
Start
Lower bound = 2+3+1+4 = 10
1
2
3
4
a → 1
ower bound
9+3+1+4 = 17
a → 2
Lower bound
= 2+3+1+4 = 10
a → 3
Lower bound
= 7+4+5+4 = 20
a → 4
Lower bound
= 8+3+1+6 =
5
6
7
b → 1
Lower bound
= 2+6+1+4 = 13
b → 3
Lower bound
= 2+3+5+4 = 14
b → 4
Lower bound
= 2+7+1+6 = 16
8
c → 3
Lower bound
= 2+6+1+4 = 13

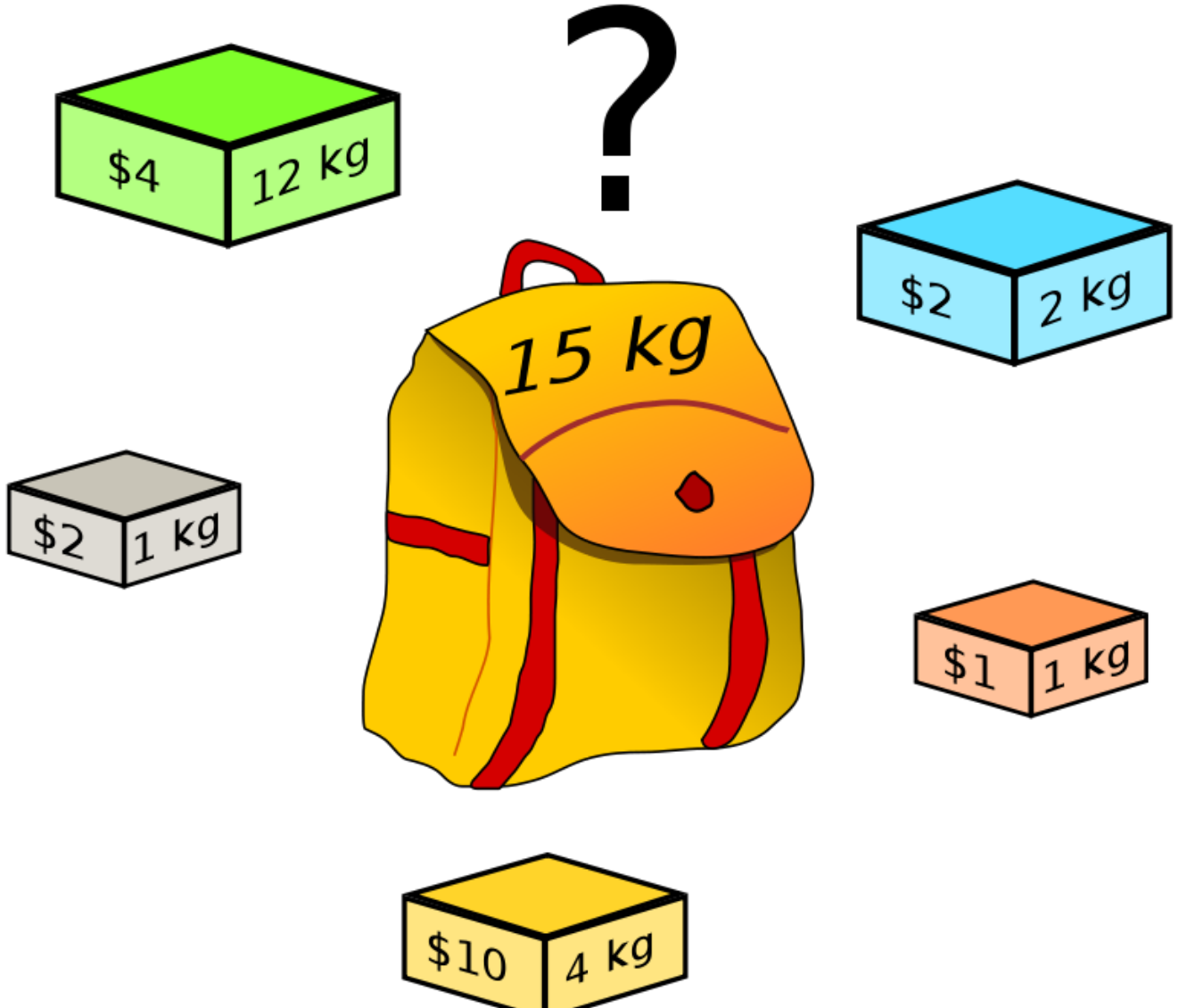

Job1 Job2 Job3 Job4
7 8
3 7
1 8
9 4
0
Start
Lower bound = 2+3+1+4 = 10
1
2
3
4
a → 1
ower bound
= 9+3+1+4 = 17
a → 2
Lower bound
= 2+3+1+4 = 10
a → 3
Lower bound
= 7+4+5+4 = 20
a → 4
Lower bound
= 8+3+1+6 = 18
5
6
7
b → 1
Lower bound
= 2+6+1+4 = 13
b → 3
Lower bound
= 2+3+5+4 = 14
b → 4
Lower bound
= 2+7+1+7 = 17
8
9
c → 3
Lower bound
= 2+6+1+4 = 13
c → 4
Lower bound
= 2+6+8+9= 25


0
start
lb = 10
1
a → 1
lb = 17
X
2
a → 2
lb = 10
3
a → 3
lb = 20
X
4
a → 4
lb = 18
X
5
b → 1
lb = 13
6
b → 3
lb = 14
X
7
b → 4
lb = 17
X
8
c → 3
d → 4
cost = 13
solution
9
c → 4
d → 3
cost = 25
inferior solution

- Given *n* tasks and *n* agents.
  - Each agent has a cost to complete each task
  - Assign each agent one task; each task one agent;
    - A 1:1 mapping
  - *Minimize* cost

	1	2	3	4
A	11	12	18	40
B	14	15	13	22
C	12	17	19	23
D	17	14	20	28

Knapsack Problem



Knapsack Problem

- Given n items of known weights w_i and values v_i, $i = 1, 2, \ldots, n$, and a knapsack of capacity W.

- The knapsack problem is to find the most valuable subset of the items that fit in the knapsack.

- Order the items of a given instance in descending order by their value-to-weight ratios

$$v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n.$$

- Then the first item gives the best payoff per weight unit and the last one gives the worst payoff per weight unit

Knapsack Problem

to compute the upper bound *ub*

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

The 0/1 knapsack problem

either to take an item in its entirety or not to take it at all.

Knapsack Problem

item	weight	value
1	7	$42
2	3	$12
3	4	$40
4	5	$25

$W = 10$

Knapsack Problem

item	weight	value	v_i / w_i
1	7	\$42	6
2	3	\$12	4
3	4	\$40	10
4	5	\$25	5

$W = 10$

em	weight	value	v_i / w_i
3	4	\$40	10
1	7	\$42	6
4	5	\$25	5
2	3	\$12	4

item	weight	value	v_i / w_i
1 (3)	4	\$40	10
2 (1)	7	\$42	6
3 (4)	5	\$25	5
4 (2)	3	\$12	4

$ub = v + (W - w)(v$

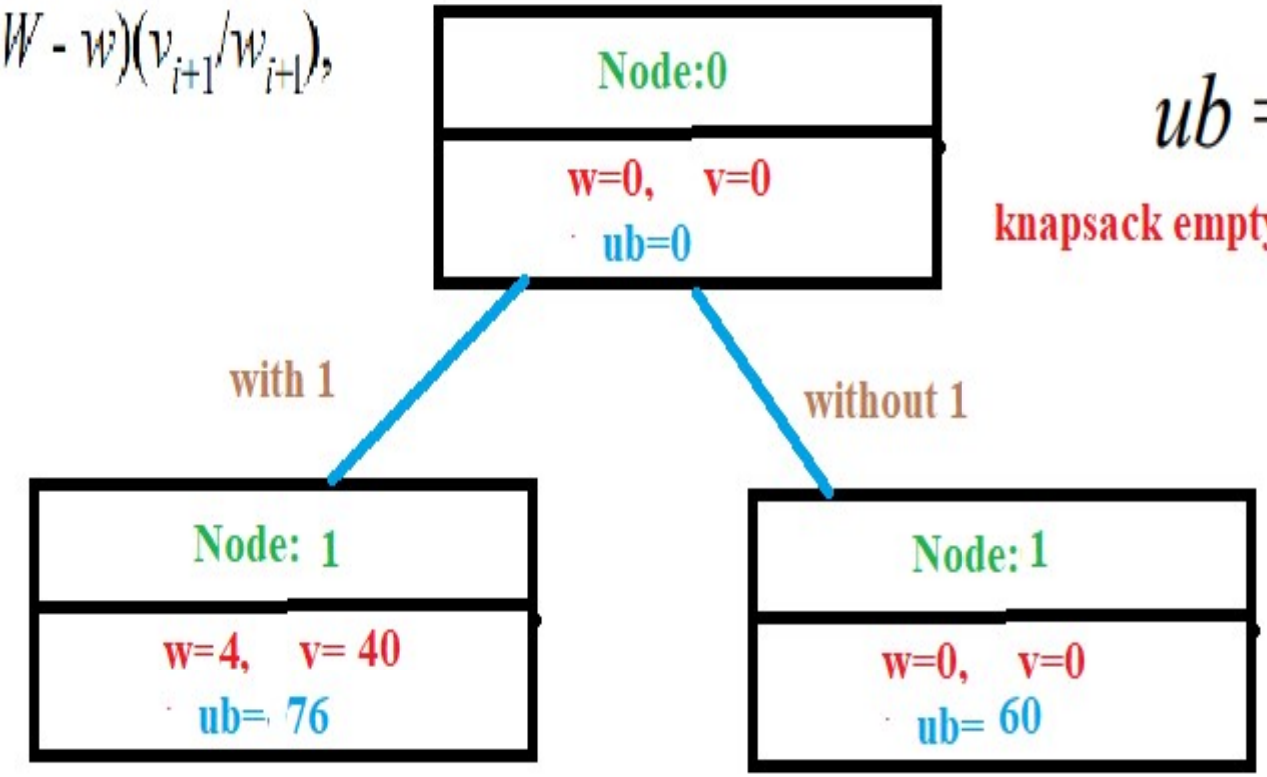
Node:0
w=0, v=0 ub=0

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

knapsack empty

item	weight	value	v_i / w_i
1 (3)	4	$40	10
2 (1)	7	$42	6
3 (4)	5	$25	5
4 (2)	3	$12	4

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



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

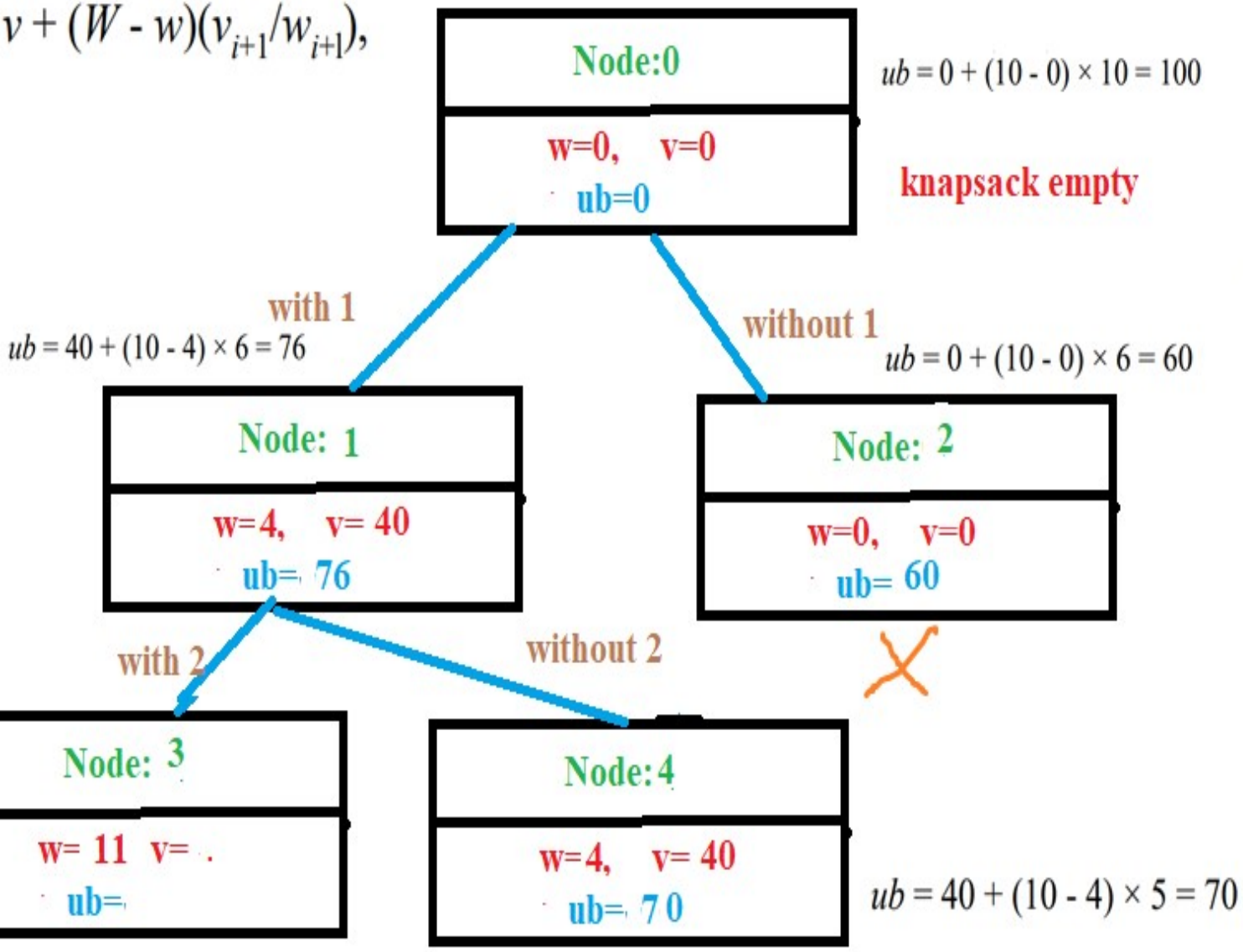
knapsack empty

$ub = 0 + (10 - 0) \times 6 = 6$

$= 40 + (10 - 4) \times 6 = 76$

item	weight	value	v_i /
1 (3)	4	$40	10
2 (1)	7	$42	6
3 (4)	5	$25	5
4 (2)	3	$12	4

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

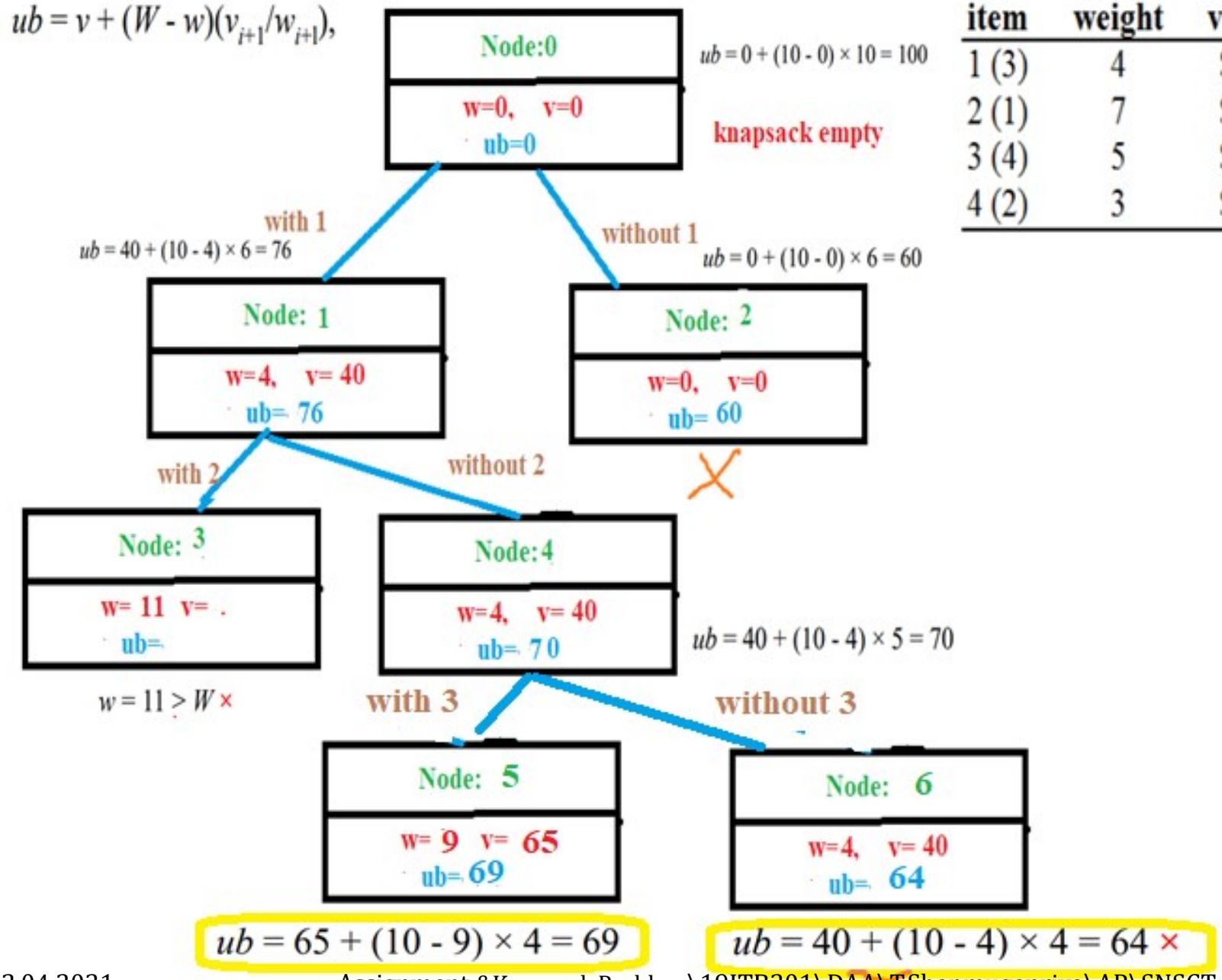


item	weight	value
1 (3)	4	$40
2 (1)	7	$42
3 (4)	5	$25
4 (2)	3	$12

$w = 11 > W$ ×

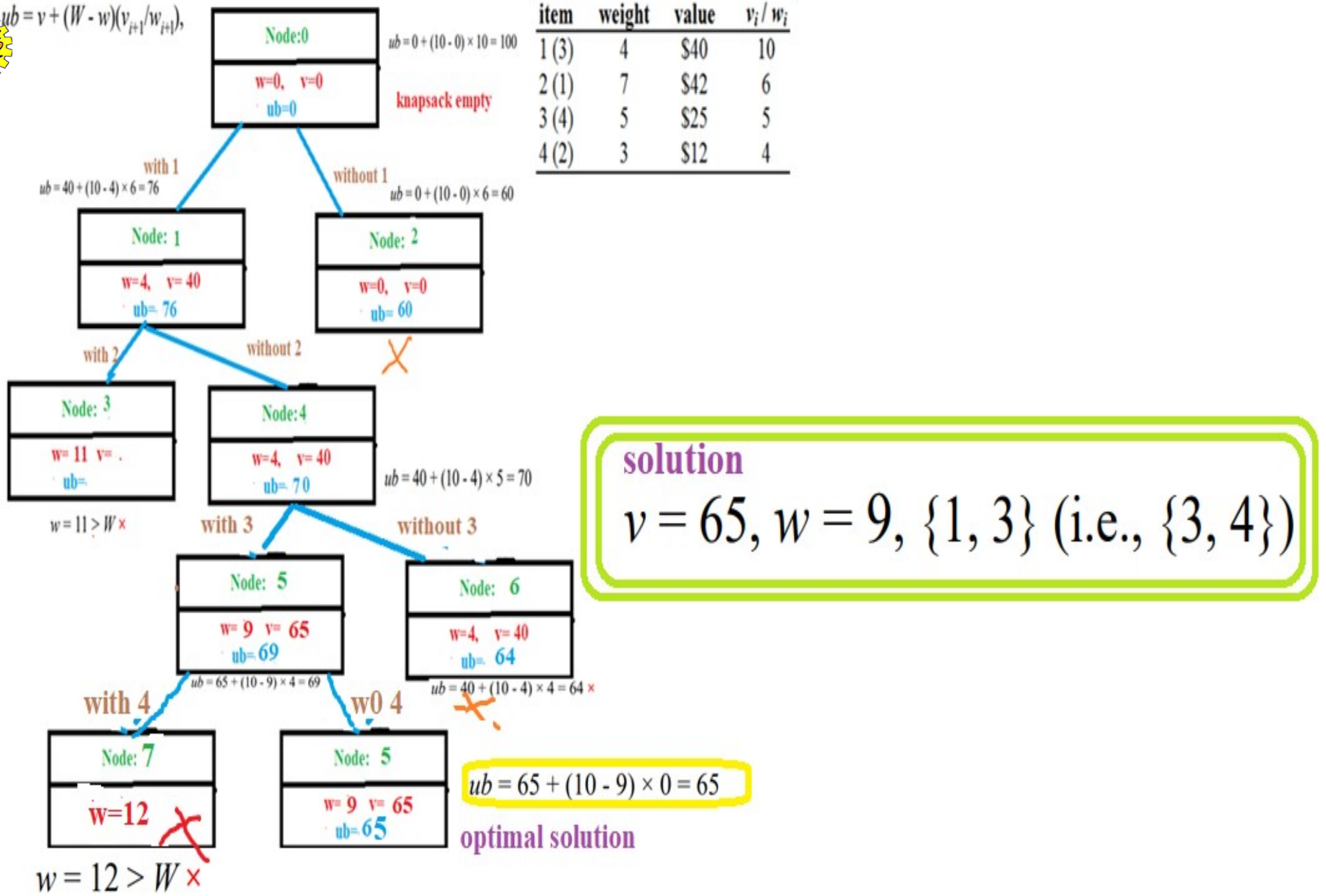
$ub = v + (W - w)(v_{i+1}/w_{i+1})$,

item	weight	value	v_i / w_i
1 (3)	4	$40	10
2 (1)	7	$42	6
3 (4)	5	$25	5
4 (2)	3	$12	4



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

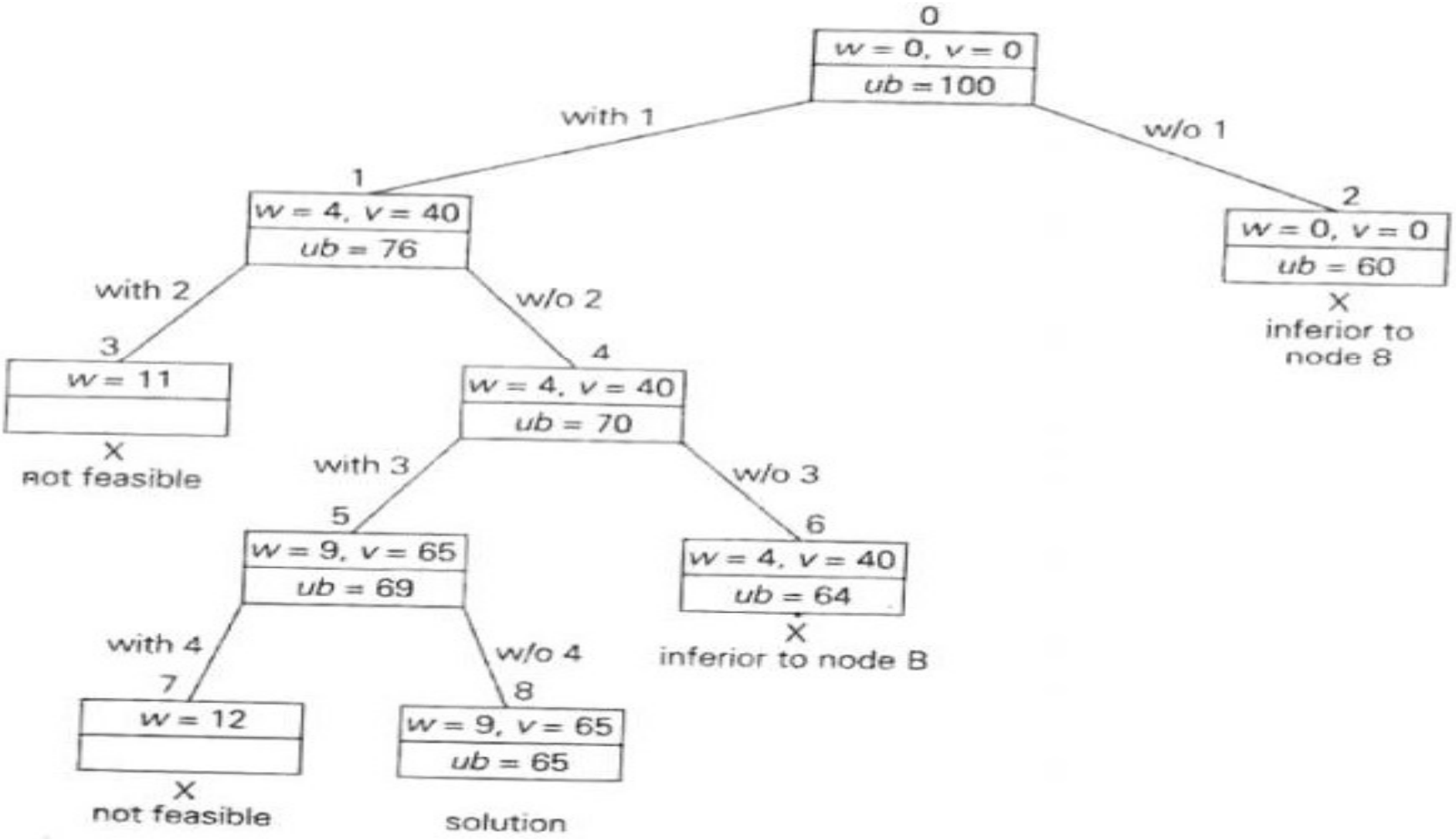
item	weight	value	v_i / w_i
1 (3)	4	$40	10
2 (1)	7	$42	6
3 (4)	5	$25	5
4 (2)	3	$12	4



solution

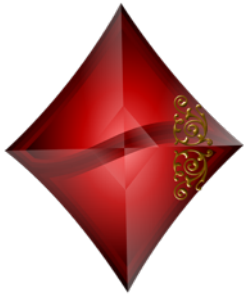
$v = 65, w = 9, \{1, 3\}$ (i.e., $\{3, 4\}$)

Knapsack Problem



Wt. = 5
Value = 10

Wt. = 3
Value = 20



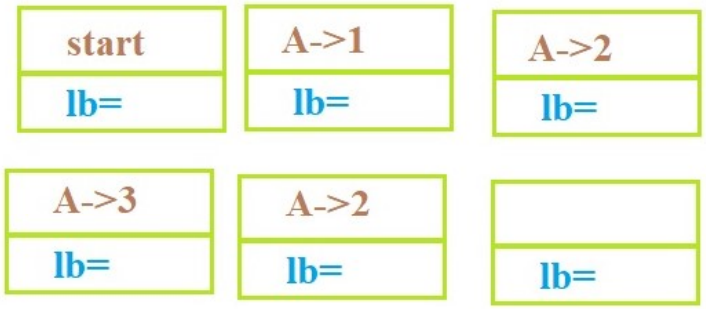
Wt. = 8
Value = 25

Wt. = 4
Value = 8

Maximum wt. = 13

i	1	2	3	4
wi	3	2	4	1
vi	8	3	9	6

W=5



	1	2	3	4
A	11	12	18	40
B	14	15	13	22
C	12	17	19	23
D	17	14	20	28


Thank You