UNIT-5

Branch and Bound

- Branch and bound is an optimization technique to get an optimal solution to the problem.
- **Feasible solution** - is a point in the problem's search space that satisfies all the problem's constraints.
  Eg:- Hamiltonian circuit in TSP.
- **Optimal solution** - is a feasible solution with the best value of the objective function.
  Eg:- Shortest Hamiltonian circuit of TSP.
- Branch & Bound is used for solving combinational 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

- Traverses the state space tree by BFS manner.
- But not all nodes get expanded
- The selected criterion tells
  * Which node to expand
  * When to expand a node
  * When an optimal soln has been found.

State space tree - Terminologies

Problem state - Each node in the tree defines a problem state

State space - All paths from root to other nodes

Soln. space - Soln space 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 & 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

1) Alg. for finding all solutions to some computational problems
2) Finds the solution to the overall issue by finding a solution to the first sub problems & then recursively solving other sub problems based on the solution of the first issue.
3) More efficient
4) It traverses the state space tree by DFS manner.
5) It involves feasibility function.
6) It searches the state space tree until it has found a solution.

Branch & Bound

1) Alg. for discrete and Combinatorial optimization problems.
2) Solves a given problem by dividing it into atleast two new restricted sub problems.
3) Less efficient
4) It traverses by DFS or BFS manner.
5) It involves bounding function.
6) It completely searches the state space tree to get optimal solution.

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 payoffs per weight unit & the last one gives the worst payoff per weight unit

to compute the upper bound ub.

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

The principle behind 0/1 knapsack problem is either to take an item ~~in its~~ entirely or not to take it at all.

item	Weight	Value
1	7	$ 42
2	3	$12
3	4	$40
4	5	$25

capacity W = 10

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

$W = 10$

item	kleight	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

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

Knapsack empty

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

$ub = 0 + (10-0)(v_1/w_1)$

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

$ub = 100$

kith 1 | kithout 1

Node : 1
w = 4 v = 40
ub = 76

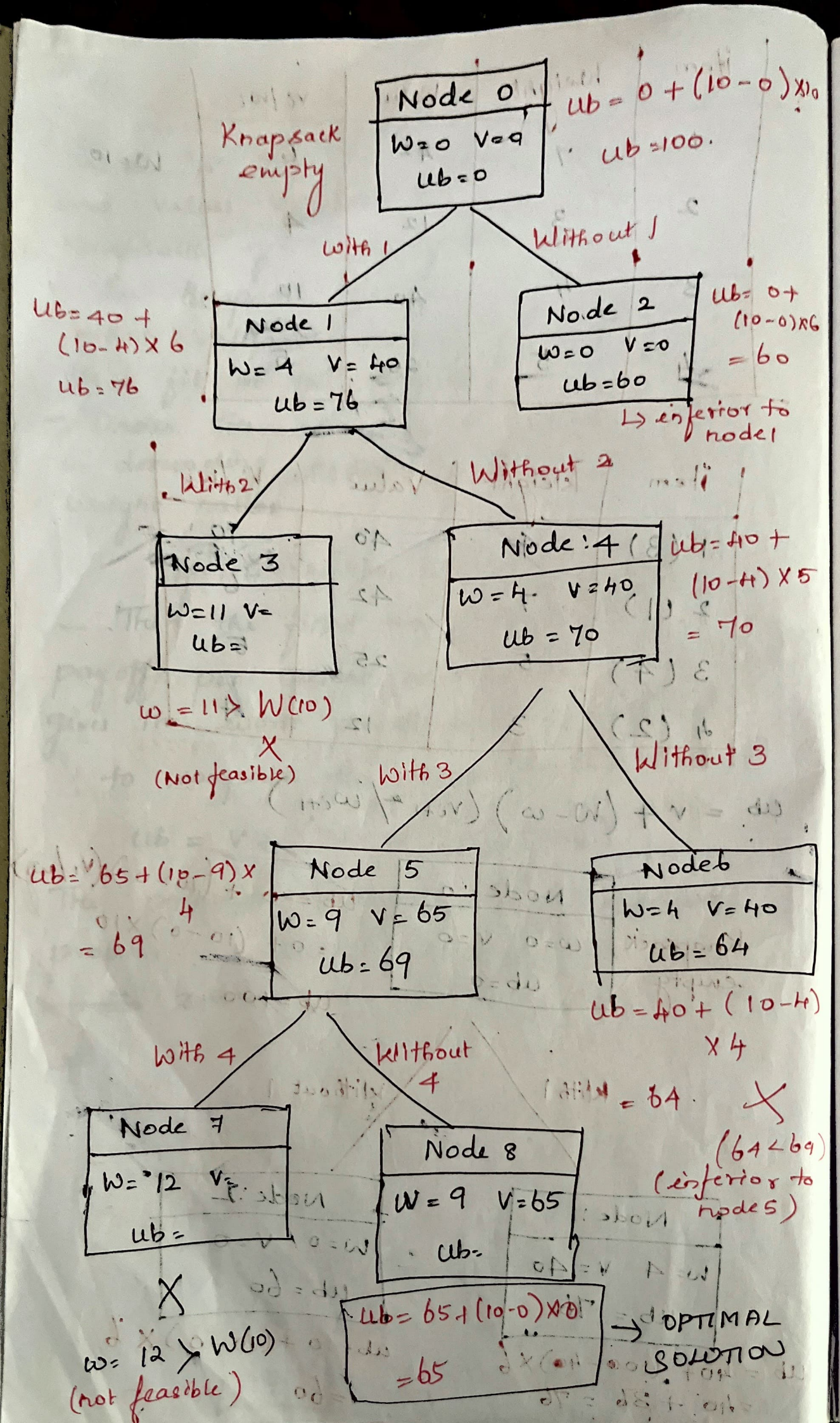
$ub = 40 + (100 - 40) \times 6$

$= 40 + 36 = 76$

Node : 2
w = 0 v = 0
ub = 60

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

$= 60$


Node 0
W=0 V=0
ub=0
Knapsack empty
ub = 0 + (10-0) x 10
ub = 100.
With 1
Without 1
Node 1
W=4 V=40
ub=76
ub = 40 + (10-4) x 6
ub = 76
Node 2
W=0 V=0
ub=60
ub = 0 + (10-0) x 6
= 60
↳ inferior to node1
With 2
Without 2
Node 3
W=11 V=
ub=
w = 11 > W(10)
X
(Not feasible)
Node 4
W=4 V=40
ub = 70
ub = 40 + (10-4) x 5
= 70
With 3
Without 3
Node 5
W=9 V=65
ub=69
ub = 65 + (10-9) x 4
= 69
Node6
W=4 V=40
ub=64
ub = 40 + (10-4) x 4
= 64
X
(64 < 69)
(inferior to nodes)
With 4
Without 4
Node 7
W=12 V=
ub=
X
w = 12 > W(10)
(not feasible)
Node 8
W=9 V=65
ub=
ub = 65 + (10-0) x 0
= 65
→ OPTIMAL SOLUTION

Solution

V = 65

W = 9

items included = {1,3} i.e, {3,4}

Exercise problem to Solve

items	Weight	Value
1	3	8
2	2	3
3	4	9
4	1	6

W = 5

items	Weight	Value
1	5	10
2	3	20
3	8	25
4	4	8

W = 13

Travelling sales man Problem

The travelling Salesman Problem (TSP) is to find the shortest tour through a given set of n cities that visits each city exactly once before returning to the city where it started.

The problem can be modelled by a weighted graph, with the graph's vertices representing the cities and the edge weights specifying the distance.

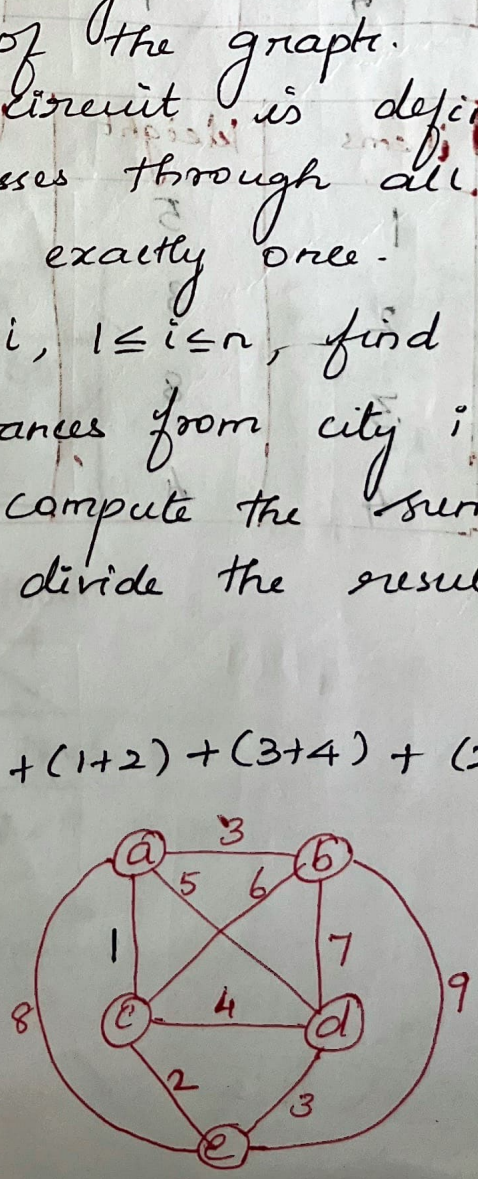
Then the TSP problem can be stated as the problem of finding the shortest Hamiltonian circuit of the graph.

A Hamiltonian circuit is defined as a cycle that passes through all the vertices of the graph exactly once.

For each city i, $1 \le i \le n$, find the sum s_i of the distances from city i to the two nearest cities; compute the sum s of these n numbers; divide the result by 2 ($lb = \lceil s/2 \rceil$).

Ex: $lb = \lceil ((1+3)+(3+6)+(1+2)+(3+4)+(2+3))/2 \rceil$

$lb = 14$



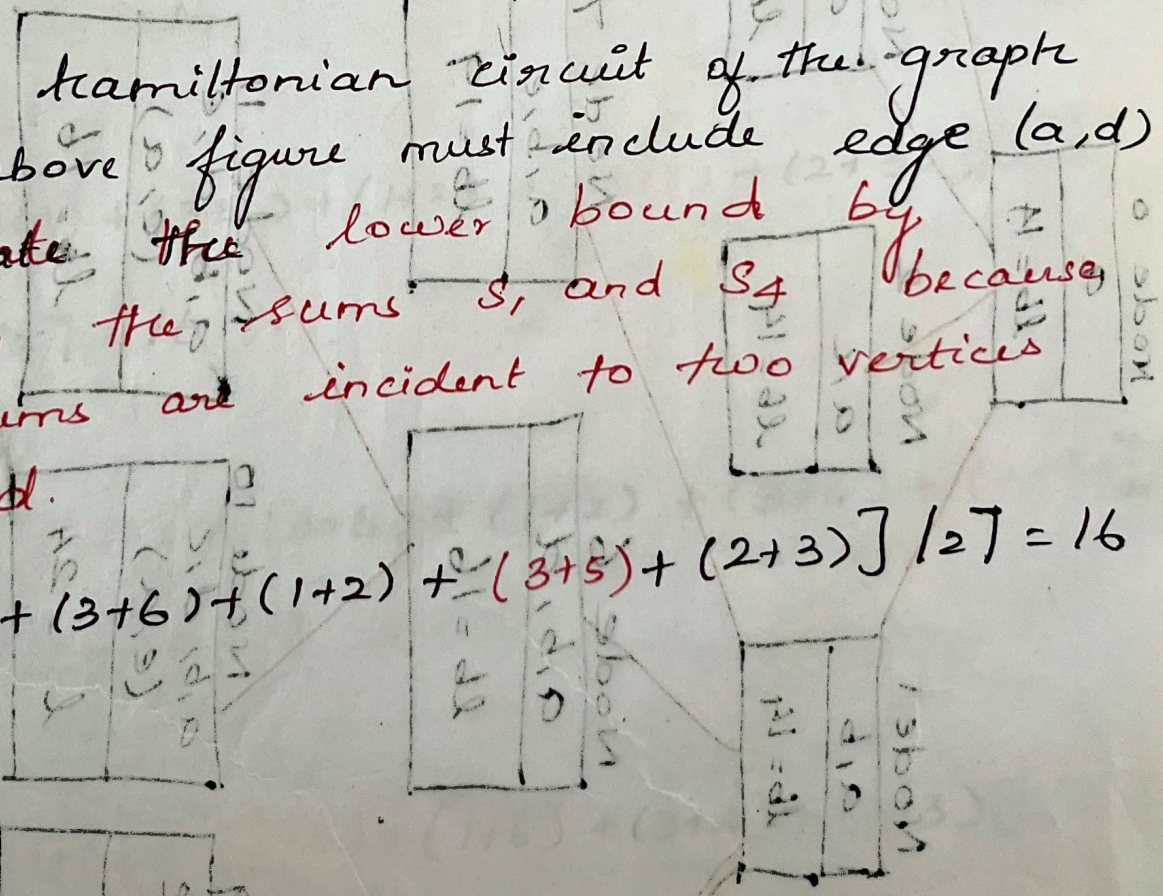
UNIT-5

Limitations of algorithms

- It is time consuming

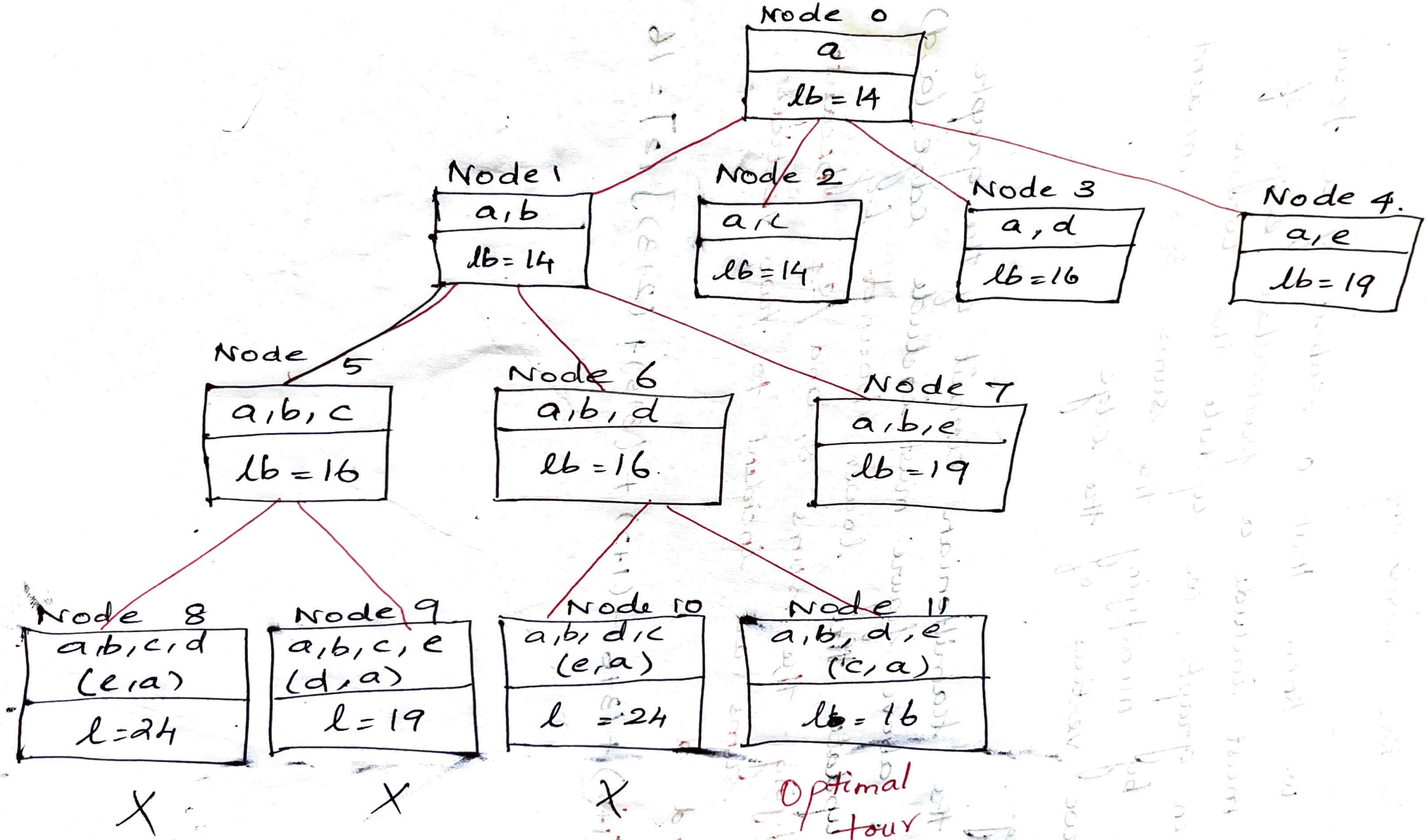
If we know that a subset of a tour that must include a particular edge of a given graph, we update the lower bound by modifying the sums s_i's incident to two vertices of the edge

for eg:



If the hamiltonian circuit of the graph in the above figure must include edge (a,d) we update the lower bound by modifying the sums s_1 and s_4 because these sums are incident to two vertices a and d.

$$\lceil [(1+5) + (3+6) + (1+2) + (3+5) + (2+3)] / 2 \rceil = 16$$


Node 0
a
lb = 14
Node 1
a, b
lb = 14
Node 2
a, c
lb = 14
Node 3
a, d
lb = 16
Node 4.
a, e
lb = 19
Node 5
a, b, c
lb = 16
Node 6
a, b, d
lb = 16
Node 7
a, b, e
lb = 19
Node 8
a, b, c, d
(e, a)
l = 24
Node 9
a, b, c, e
(d, a)
l = 19
Node 10
a, b, d, c
(e, a)
l = 24
Node 11
a, b, d, e
(c, a)
lb = 16
X
X
X
Optimal
tour

Node 0

$lb = \lceil ((1+3)+(3+6)+(1+2)+(3+4)+(2+3)]/2 \rceil$

$= \lceil 28/2 \rceil = 14$

Node 1

$lb = \lceil [(1+3)+(3+6)+(1+2)+(3+4)+(2+3)]/2 \rceil$

$= \lceil 28/2 \rceil = 14$

Node 2

$lb = \lceil [(1+3)+(3+6)+(1+2)+(3+4)+(2+3)]/2 \rceil$

$= \lceil 28/2 \rceil = 14$

Node 3

$lb = \lceil [(1+5)+(3+6)+(1+2)+(3+5)+(2+3)]/2 \rceil$

$= \lceil 31/2 \rceil = 16$ X

Node 4

$lb = \lceil [(1+3)+(3+6)+(1+2)+(3+4)+(2+3)]/2 \rceil$

$= \lceil 38/2 \rceil = 19$ X

Node 5 abc

$lb = \lceil [(1+3)+(3+6)+(1+6)+(3+4)+(2+3)]/2 \rceil$

$= \lceil 32/2 \rceil = 16$

Node 6 abd

$lb = \lceil [(1+3)+(3+7)+(1+2)+(3+7)+(2+3)]/2 \rceil$

$= \lceil 32/2 \rceil = 16$

Node 7 abe

$lb = \lceil [(1+3) + (3+9) + (1+2) + (3+4) + (2+9)] / 2 \rceil$

$= \lceil 37/2 \rceil = 19$

// Leaf nodes.

Node 8 : $l = 3+6+4+3+8 = 24$

Node 9 : $l = 3+6+2+3+5 = 19$

Node 10 : $l = 3+7+4+2+8 = 24$

Node 11 : $l = 3+7+3+2+1 = 16$ ✓

Solution

tour = (a, b, d, e, c, a)

tour length = 16

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 & n jobs
- Any person can be assigned to perform any job.
- It is required to perform all jobs.
- Assigning exactly one job to each person & vice versa
- 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).

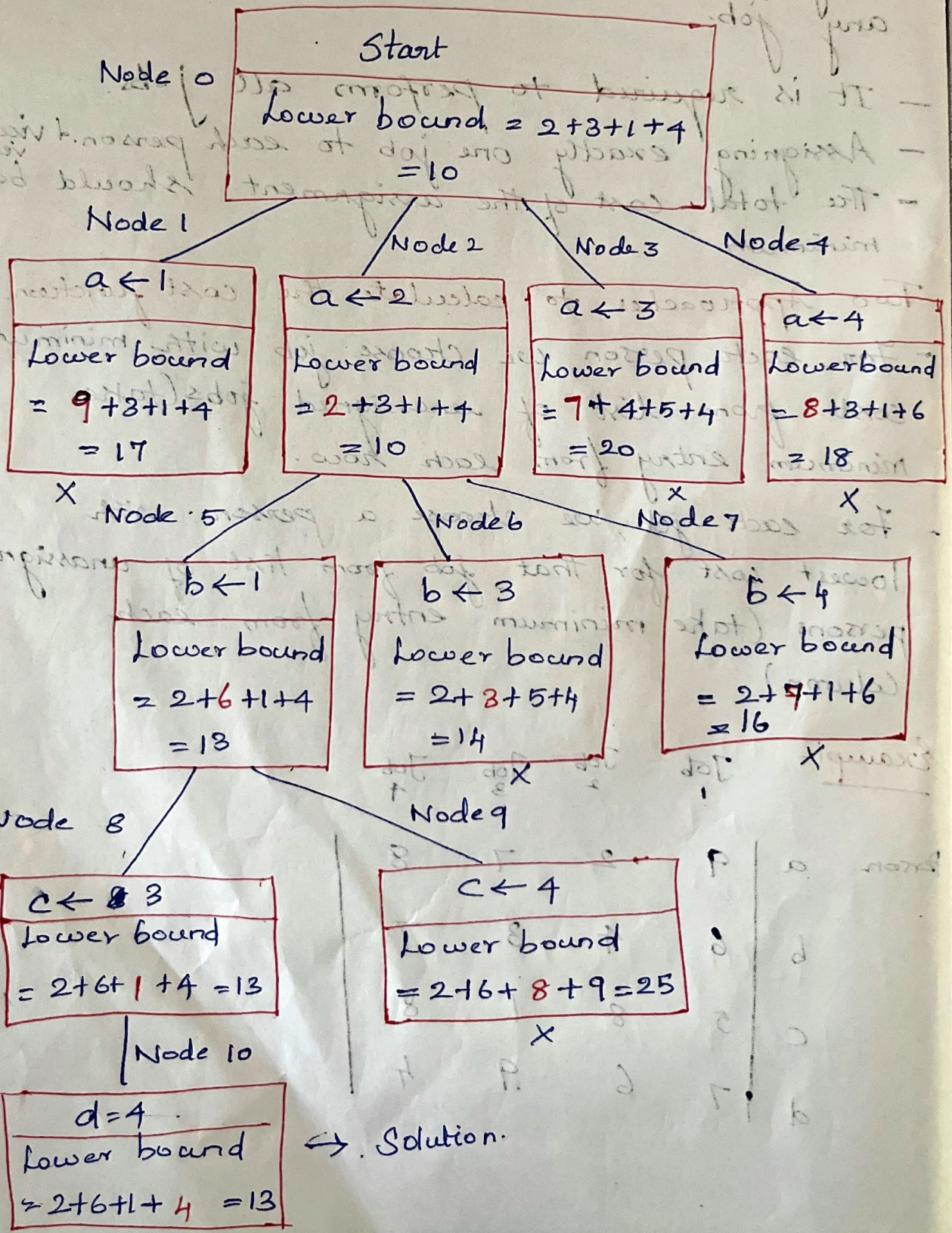
Example

Person	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4

Lower bound

↳ From each row choose the minimum cost of jobs.

a ← 2
b ← 3
c ← 1
d ← 4



Exercise Problems

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

Coping With Limitations

- Some problems are difficult to solve algorithmically
- Two algorithm design techniques for solving such problems
  - → Backtracking
  - → Branch-and-bound
- Both are based on the construction of state-space tree.
- Both techniques terminate a node as soon as it can be guaranteed that no solution to the problem
- Few approximation algorithms for solving
  - → Assignment Problem
  - → TSP
  - → Knapsack problem.
- Algorithms for solving Nonlinear Equation
  - → Bisection Method
  - → False Position Method
  - → Newton's Method.

Exact solution strategies.

- Exhaustive search (brute force)
  - → useful only for small n.
- Dynamic Programming
  - → applicable to some problems (eg: Knapsack)

Coping with Limitations of algorithmic power:

- Backtracking
  - → eliminates some unnecessary cases from consideration
  - → yields solution in reasonable time for many instances but worst case is still exponential.
- Branch and Bound
  - → further refines the backtracking idea for optimization problem.
- Backtracking
  - → n-Queens Problem
  - → Hamiltonian Circuit Problem
  - → subset sum problem.
- Branch and Bound
  - → Assignment Problem
  - → Knapsack problem
  - → TSP
- Approximation algorithms for NP-Hard problems.
  - ↳ for TSP & Knapsack problem.

Solving Difficult Combinatorial problems (NP-Hard)

↳ Use an approximation alg. that can find an approximate (sub-optimal) solution in polynomial time.

↳ Use a strategy that guarantee solving the pblm exactly but doesn't guarantee to find a soln in polynomial time.