Feb 2025

Professional - III

Page No.
Date

Back Tracking :-

Terminologies :-

1) Criteria function : A fn used to evaluate partial soln in backtracking to determine if they meet the necessary condition to proceed further.

2) Pruning : the process of eliminating certain branches in the search space that cannot lead to a valid solution, improving efficiency.

3) Decision Tree : A tree structure representing all possible choices at each step of a backtracking algo, where each node corresponds to a decision.

4) State - space tree : A conceptual tree representation of the problem where nodes represent states and edges represent transition between states.

Backtracking problems :-

1) Sudoku
2) N-Queen
3) Sum of subset
4) 0/1 knapsack
5) Hamiltonian cycle problem
6) Graph coloring
7) Few games like maze

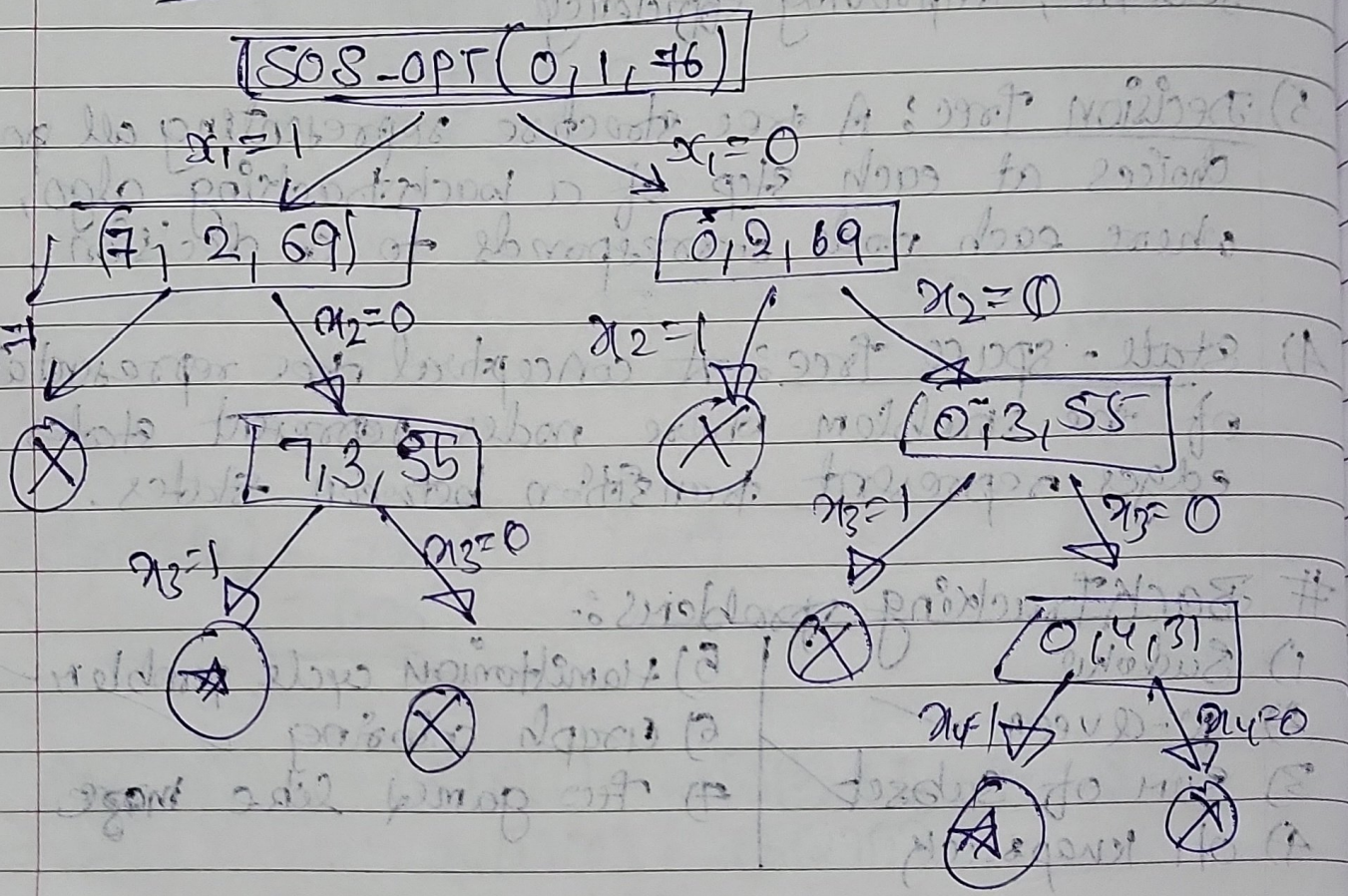
vld

(F) Sum of subset - BT

- numbers: {7, 17, 24, 31}
- targetSum: 31, X[] = {0,0,0,0}; DesSum = 76.

~~Brute force algo~~:

Decision tree:



Optimal answers

Algo: SOS_opt(s, k, r) → remaining sum
(currSum, idx) → O(2^n)

```
{
// Include
x[k] = 1.
if(s + nums[k] == targetSum)
{
    print(x);
    return;
}
else if(s + nums[k+1] + num[k] <= m)
{
    SOS_opt(s+num[k], k+1, r - num[k]).
}

// not include
if(s + r - num[k] >= targetSum &&
                    s + nums[k+1] <= m)
{
  x[k] = 0
  SOS_opt(s, k+1, r - num[k]);
}
}
```

(II) N-Queens :-

Algo : NQueens(k, N) ⟶ O(n^n)

```
for (i = 1 to N)
{
    if (place(k, i) == true)
    {
        x[k] = i
        if (k == n)
            print(x)
        else
            NQueens(k+1, n);
    }
}
```

place(k, j) → row, column ⟶ O(n)

```
for (j = 1 to k-1)
{
    if (x[j] == i or abs(j-k) == abs(x[j]-i))
        return false
}
return true;
```

→ Diagonal check in place function is

- we have (r_1, c_1) and (r_2, c_2) on same diagonal if $|r_1 - r_2| = |c_1 - c_2|$.

so, if Q_1 attacks Q_2 diagonally given
Q_1 position $(j, x[j])$
Q_2 position (k, i)
$\rightarrow |j - k| = |x[j] - i|$.

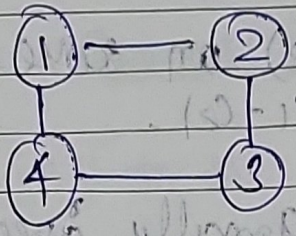
⇒ Back-Tracking explores soln in depth first manner.

→ The state space search tree is created and the function which decides this is known as Bounding function.

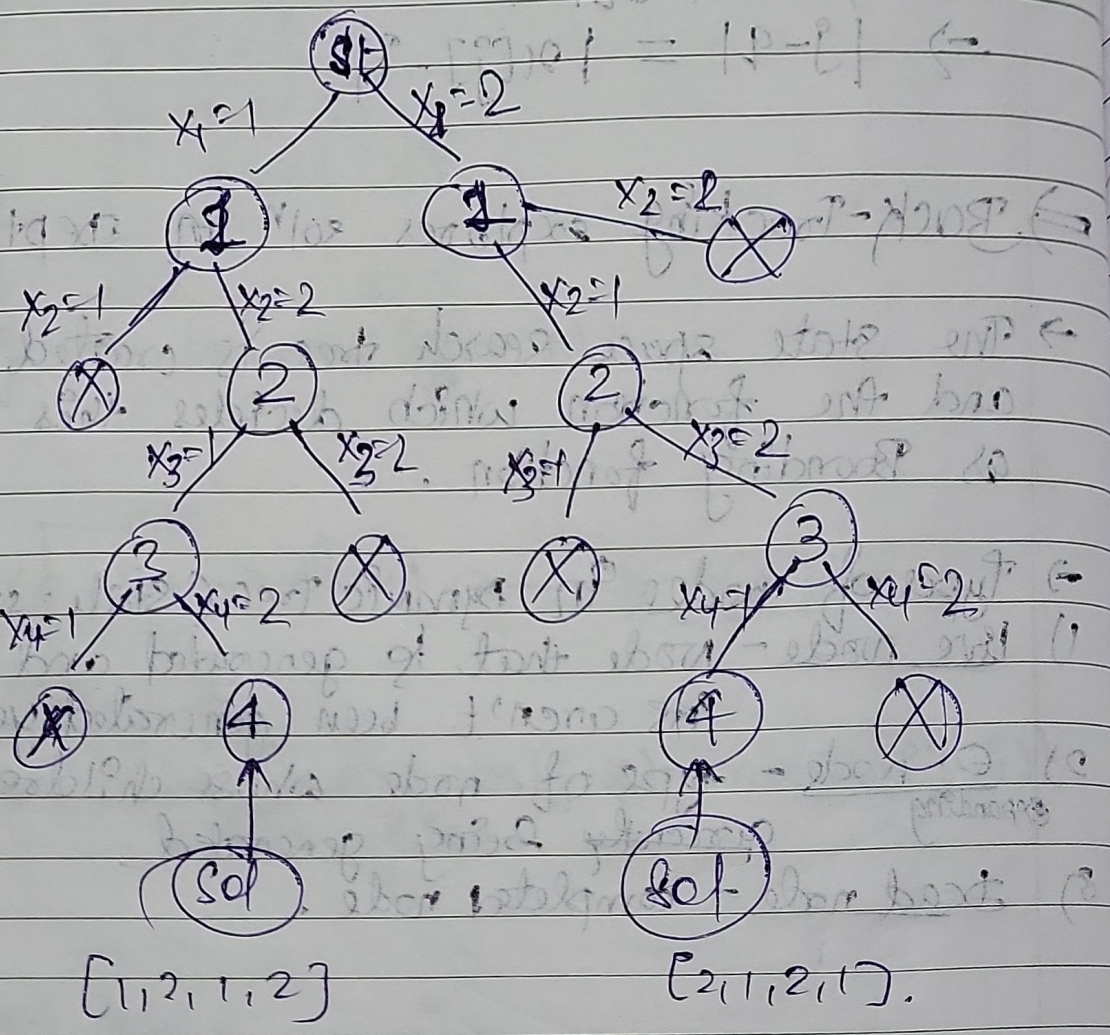
→ Types of nodes in dynamic tree (state-space search)
1) Live node - node that is generated and children of its aren't been generated yet.
2) E-node - type of node whose children are currently being generated.
expanding
3) Dead node - completed node.

(III) Graph colouring problem

Eg.



numofcolors (m) = 2 ; colors = [1, 2]



→ Max colors needed = (max. degree + 1)

→ $O(m^n)$

```
m_colouring(k)          // initially k=1
{                       // possible colours m.
 do
 {nextValue(k);
  if(x[k] == 0)  // if no colour left to
    return       //assign to k
  if(k == n)
     print(sol)
    // return → set comment to check all sol^n.
  else
     m_colouring(k+1);
 } while(true);
}
```

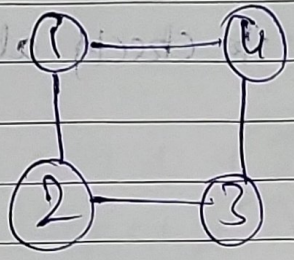
→ nextValue(k) → $O(n \times m)$.

```
{
 do
 {
  x[k] = (x[k]+1) mod (m+1);
   if(x[k] == 0) return;

   for(j=1 to n)    //n = no. of nodes.
   {   if((G[k,j] ≠ 0) && (x[k] == x[j]))
          break;
   }
   if(j == n+1)
        return
 } while(true);
}
```

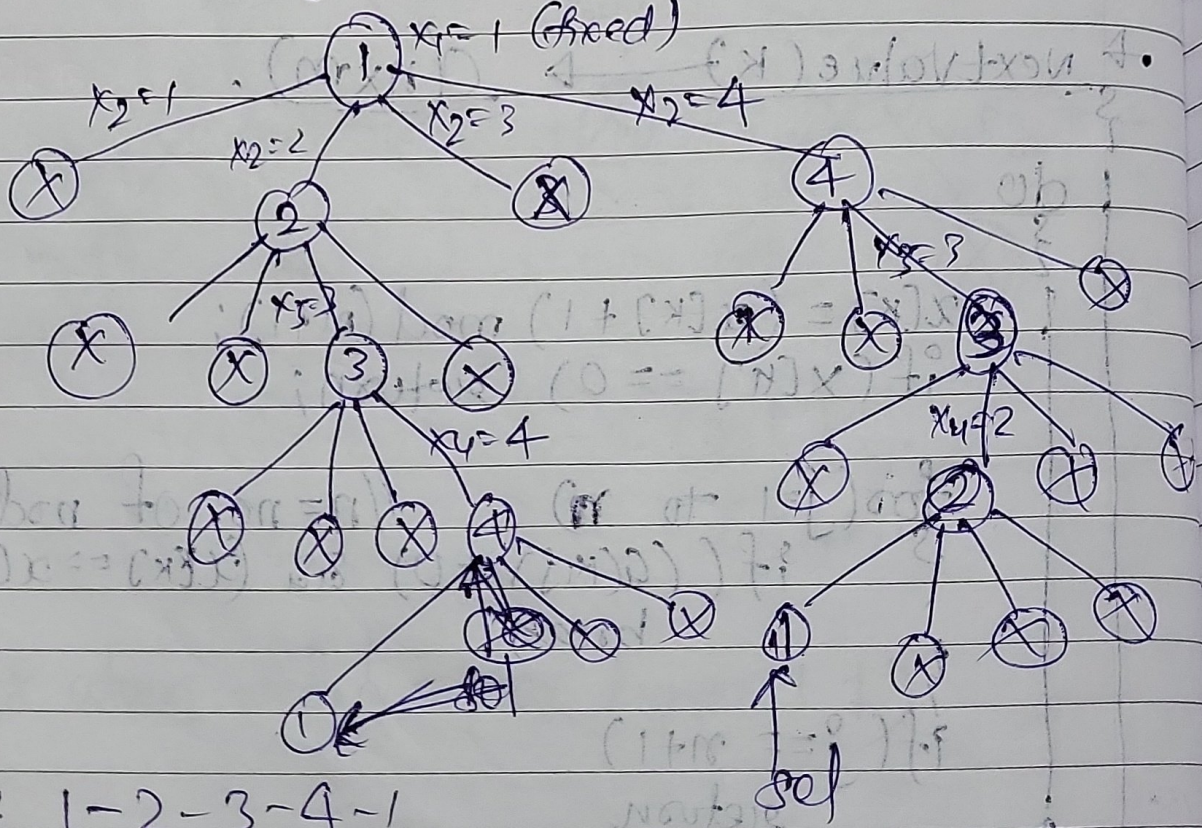
(IV) Hamiltonian cycle :-

- find Hamiltonian cycle in graph and print cycle.
- Hamiltonian cycle : cycle that covers every vertex exactly one time except starting vertex.



→ HC : 1-2-3-4-1

- Assume : starting address/vertex is always 1.



HC : 1-2-3-4-1
HC : 1-4-3-2-1

Algorithm: HC(k)

```
{
  do
  {
    NextValue(k);
    if (x[k] == 0)
        return;
    if (k == n && G[x[n]][1] == 1)
        print(x);
    else if (k < n)
        HC(k+1);
  } while(true);
}
```

```
main()
{
  x[1] = 1
  HC(2);
}
```

NextValue(k)

```
{
  do
  {
    x[k] = (x[k]+1) % (n+1);
    if (x[k] == 0) return;
    else
    {
      if(G[x[k-1]][x[k]] == 1)
      {
        for(j=1 to k-1)
        {
          if (x[k] == x[j])
              break;
        }
        if (j == k) return
      }
    }
  } while(true);
}
```

04th Mar 2025

Branch & Bound :-

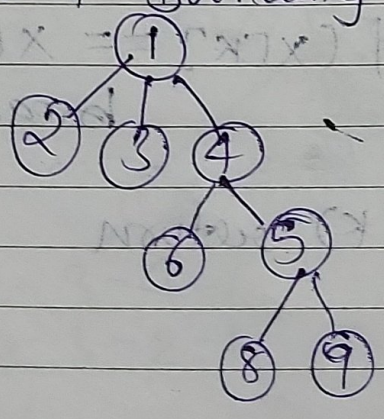
→ state-space tree
→ Type of nodes
- live nodes
- E-node
- Dead nodes

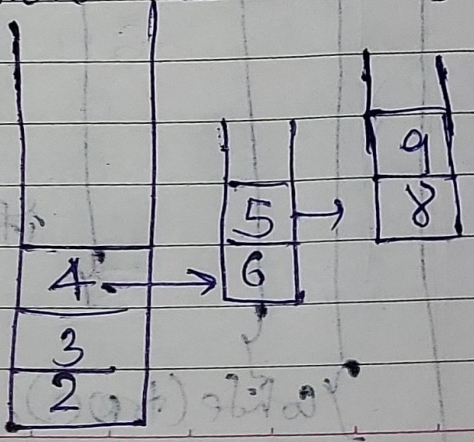
- Techniques to generate problem states (nodes)

1) DFS
- E-node will generate child and then that child becomes E-node, after child becomes dead the start node again becomes E-node.
- Bounding functions are used to kill the live nodes without generating all their children.
- DFS generation with bounding function is called Backtracking.

2) B&B
- E-node remains E-node until it is dead.
- this method is called Branch-and-Bound method.

→ Branch and Bound:
1) BFS + Bounding function (FIFO B&B)
2) D search + Bounding function (LIFO B&B)
↳ Eg.





- In B&B, always all childs of a E-node is created and then one of the child is further explored.

3) Least Cost (LC) B&B

- the selection of which child to be explored after the child of ~~first~~ E-node is generated, it is selected after some calculation that which child could give me solution with least cost :

→ The rank/cost to any node can be given from calculating:

1) no. of nodes needed to be generated before reaching answer in subtree
2) no. of levels needed to be created to reach the answer node in the subtree.

→ Above 2 cost calculating method are useless, so cost/rank to every node is given by estimation.

- ∴ cost function of any node:

$c(x) = g(x) + h(x)$

$g(x)$ = cost to reach answer from x.

$h(x)$ = cost to reach x from start.

- Now, new E-node is selected where $c(x)$ is minimum and this technique is called LC search.

- BFS, D-search are special cases of LC search.

→ LC search + Bounding function is LC B&B search.

- If no answer exists from node x then $c(x) = \infty$
- If x is the answer node, $c(x) = h(x)$, $\hat{c}(x) = 0$
- for internal nodes from which answer node exists there $c(x) = h(x) + \hat{c}(x)$.

(I) 8-puzzle - LC - BnB :-

qy

1	2	3
	4	6
7	5	8

⟹

1	2	3
4	5	6
7	8	

up, down, right, left ⊗

	2	3
1	4	6
7	5	8

$c(x) = 1 + 4$

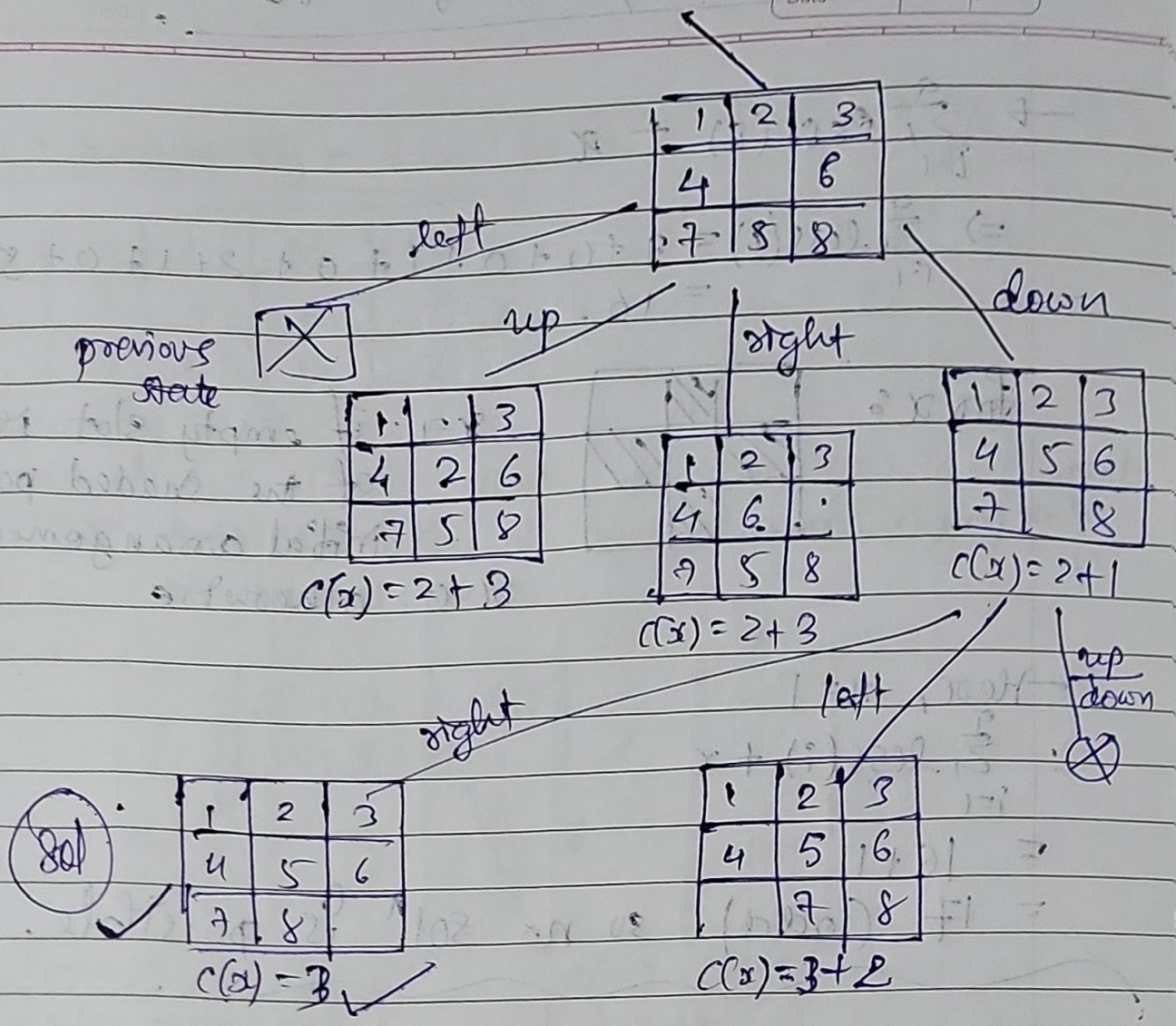
1	2	3
7	4	6
	5	8

$c(x) = 2 + 4$

1	2	3
4		6
7	5	8

$c(x) = 1 + 3$ ✓

here, we defined $c(x) = h(x) + \hat{c}(x)$ where
$h(x)$ = levels above current state
$\hat{c}(x)$ = no. of times misplaced or Σ manhatten of all the



→ when $c(x) = 0$ we get the soln
→ All other live nodes have $c(x) > 3$ so no need to explore them, just make them dead nodes.
→ Soln ∴ [Right, down, right].

Eg2

8	1	2
	4	3
7	6	5

→ no soln possible.

→ Sol is possible only if $\sum_{i=1}^{9} less(i) + x = even$.
where less(i) = no. of tiles [i=1] j such that
$j < i$ & position(j) > position(i).
here, less(8) = 7, less(1) = 0, less(2) = 0, less(4) = 1, less(empty) = 0, less(3) = 0, less(7) = 2, less(6) = 1, less(5) = 0, less(9) = 5

Check for 15 puzzle from Pb

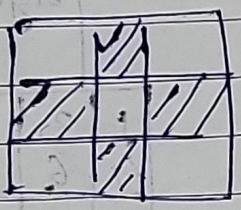
Page No.
Date

→ $\sum_{i=1}^{9} less(i) + x$

$\Rightarrow \sum_{i=1}^{9} less(i) = 9+0+0+1+0+2+1+0+5$

$= 16.$

For x:



x=1 : if empty slot is at one of the shaded part in the initial arrangement

x=0 : otherwise

→ Here, x=1

$\therefore \sum_{i=1}^{9} less(i) + x$

$= 16+1$

$= 17$ (odd) so no sol^{n} is possible.

(II) 0/1 - knapsack - LC - BnB

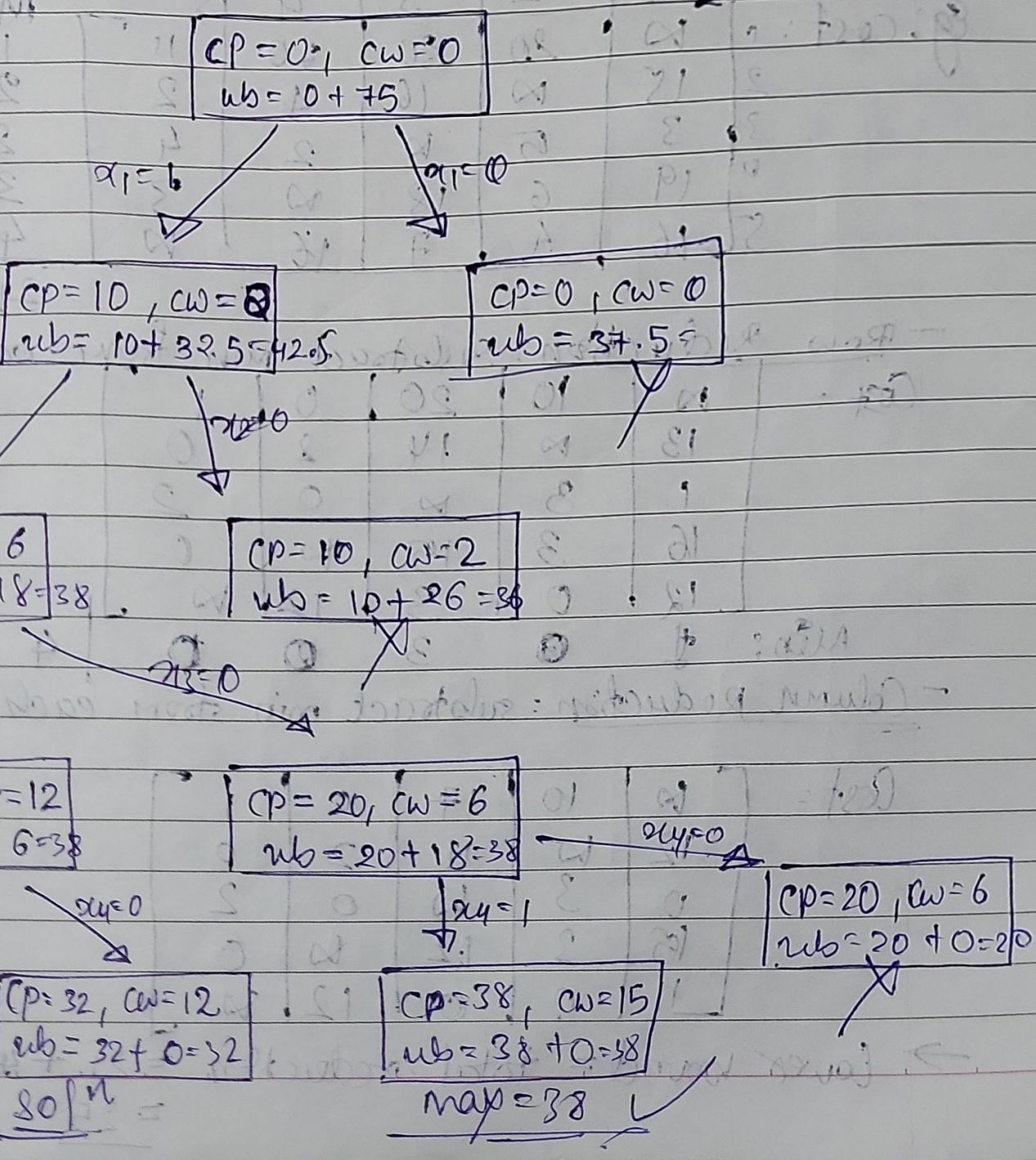
Eg!

P	10	10	12	18
W	2	4	6	9
P/W	5	2.5	2	2

- capacity = 15 = m

→ Here,

c(x) = total profit at current node + (m - currWeight) * next P/W.



(III) TSP

- Let $G = \langle V, E \rangle$ be a directed graph.
- Let C_{ij} = cost of edge (i, j)
  $C_{ij} = \infty$ if $(i, j) \notin E$

- Here, we want min. cost Hamiltonian cycle.
- Brute force approach has $(n-1)!$ options.

Eg: Cost:

	1	2	3	4	5	Min
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4
						21

- Row Reduction: subtract min from each row

Cost:

∞	10	20	0	1	
13	∞	14	2	0	
1	3	∞	0	2	
16	3	15	∞	0	
12	0	3	12	∞	
Min:	1	0	3	0	0	4

- Column Reduction: subtract min from each column

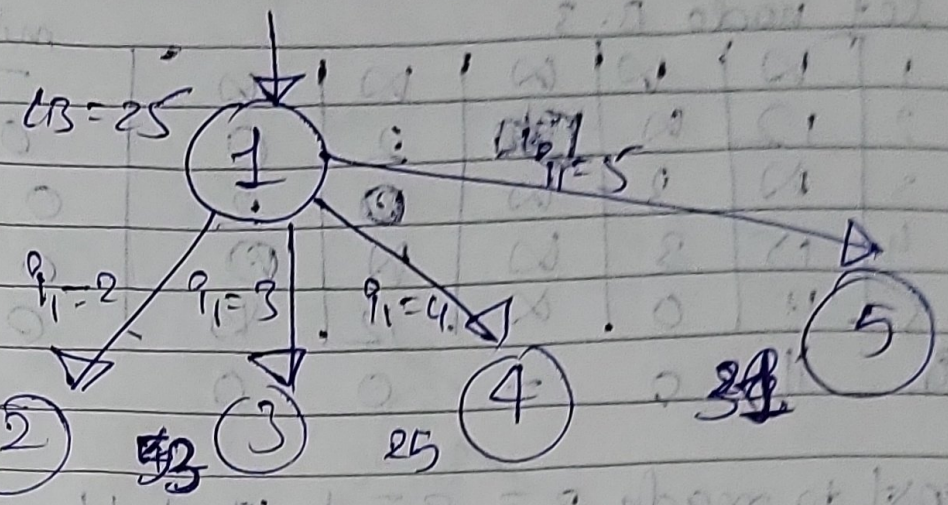
Cost:

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	0	12	∞

→ Lower bound = total reduction = 21 + 4
= 25

Ans = ∞

→ Tree :-



- cost at node 2 : lower bound at parent + cost of node B from node A in reduced cost mat. + cost of reduction R.

→ Now, on reaching node B,
- there could not be any outgoing edge from node B
- there could not be any incoming edge from node B
- no edge from node B to node A.

→ let node B = 2

∴ cost :

					Min
∞	∞	∞	∞	∞	
∞	∞	11	2	0	0
0	∞	∞	0	2	0
15	∞	12	∞	0	0
11	∞	0	12	∞	0
Min	0	0	0	0	

- lower bound at node 2 = 0 + 0
= 0

- cost to reach 2 = 25 + 10 + 0
= 35

→ let node B: 3

						min
1	∞	∞	∞	∞	∞	—
2	12	∞	[illegible]	2	0	0
3	∞	3	∞	[illegible]	2	0
4	15	3	∞	∞	[illegible]	0
5	11	0	∞	12	∞	0
min	11	0	[illegible]	0	0	

cost to reach 3 = 25 + 17 + 11
= 42 + 11
= 53

→ B: 4

						min
	∞	∞	∞	∞	∞	—
	12	∞	11	∞	0	0
	0	3	∞	∞	2	0
	∞	3	12	∞	0	0
	11	0	0	∞	∞	0
min	0	0	0	[illegible]	0	

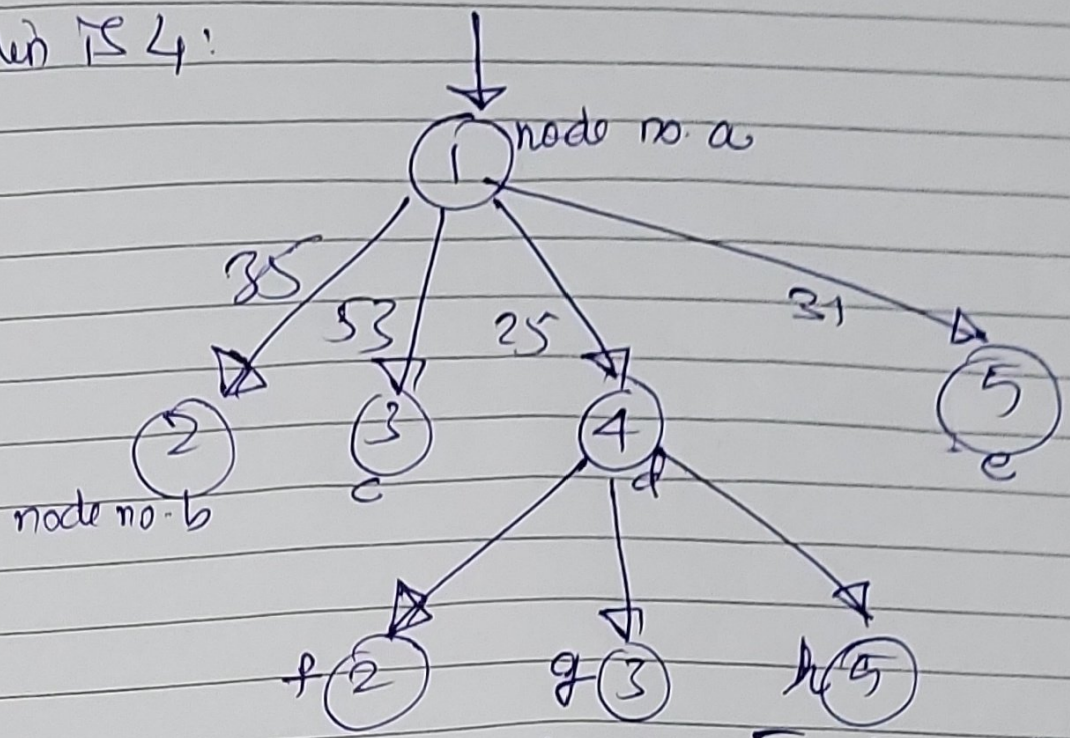
→ cost to reach 4 = 25 + 0 + 0 = 25

→ B: 5

						min
	∞	∞	∞	∞	∞	—
	10	∞	11	2	∞	2
	0	3	∞	0	∞	0
	15	3	12	∞	∞	3
	∞	0	0	12	∞	0
min	0	0	0	0	—	

→ cost to reach 5 = 25 + 1 + 5
= 31

⇒ Min is 4:



Reduced matrix at 4:

∞	∞	∞	∞	∞
12	∞	11	∞	0
0	3	∞	∞	2
∞	3	12	∞	0
11	0	0	∞	∞

→ to reach f: 2

cost:

∞	∞	∞	∞	∞
	∞		∞	
	∞	∞	∞	
∞	∞		∞	
	∞		∞	∞

→ check sir's pdf for TSP.

(IV) Job Assignment using LC B&B :-

Given

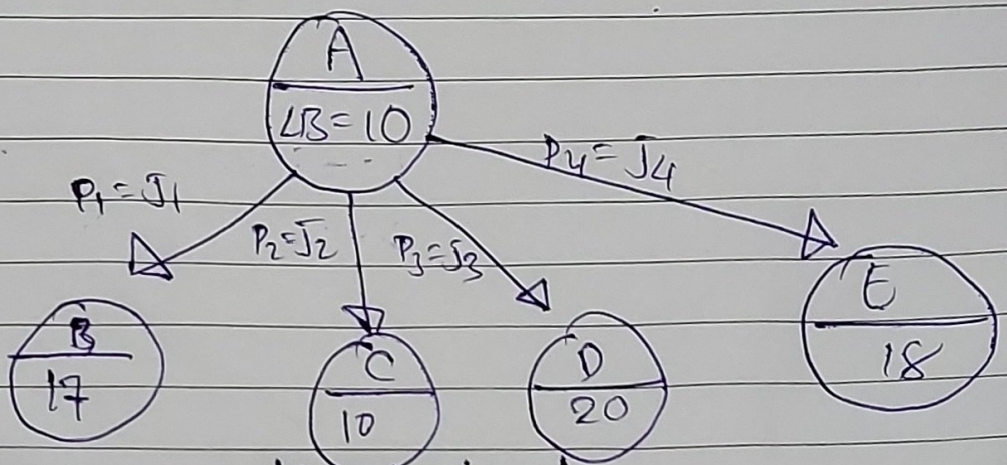
mat :

person \ Job →	J1	J2	J3	J4	min :
P1	9	2	7	8	2
P2	6	4	3	7	3
P3	5	8	1	8	1
P4	7	6	9	4	4

10. → lower bound.

→ C(x) = Job cost till + min. from left person jobs.

Eg:



for B:

LB = P1.J1 +

	J1	J2	J3	J4	min
P1	~~9~~	~~2~~	~~7~~	~~8~~	
P2	~~6~~	4	3	7	3
P3	~~5~~	~~8~~	1	8	1
P4	~~7~~	6	9	4	4
					8

= 9 + 8

= 17

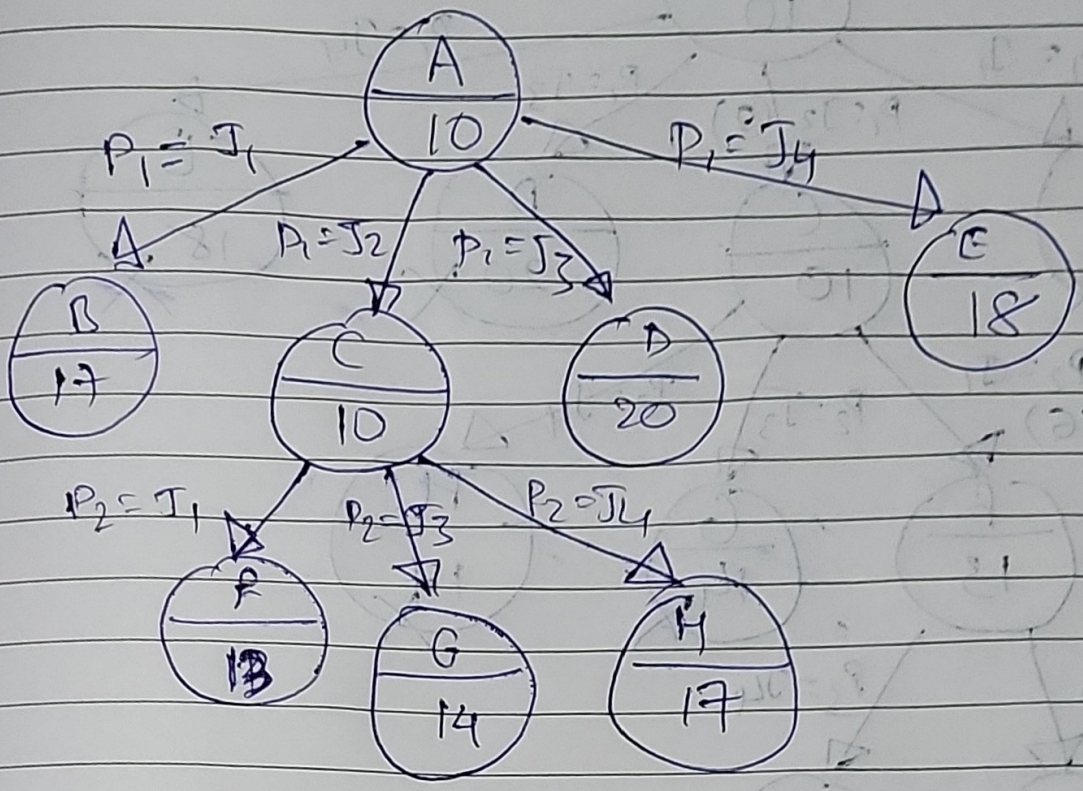
for C: LB = P1.J2 +

~~9~~	~~2~~	7	8	
6	4	3	7	3
5	~~8~~	1	8	1
7	6	9	4	4

= 2 + 8

= 10

→ min. is C with LB = 10.



for F: P_2J_1 +

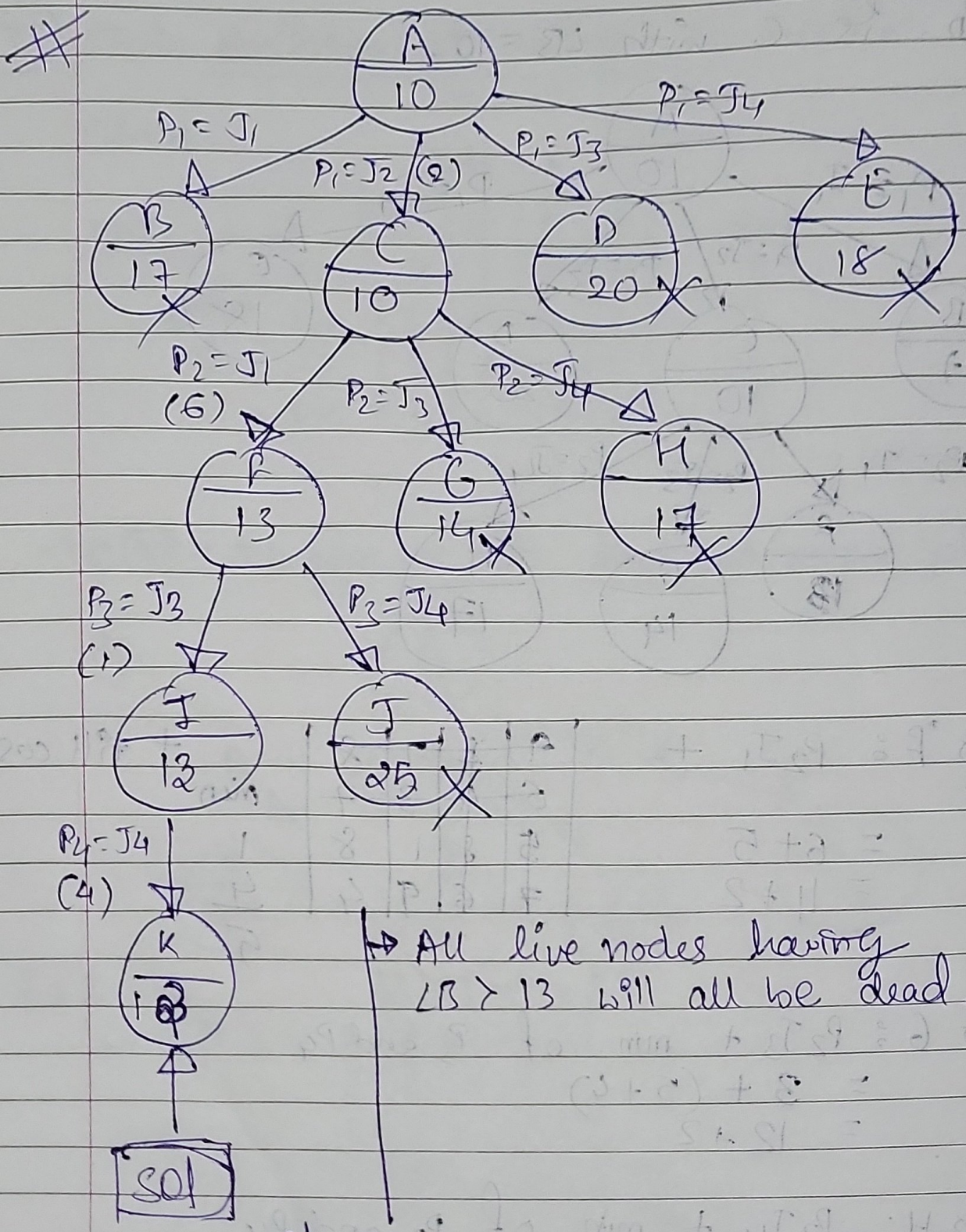
9	2	7	8	
6	4	3	7	min
5	8	1	8	1
7	6	9	4	4
				5

+ till cost

= 6 + 5
= 11 + 2

for G: P_2J_3 + min. of P_3 and P_4
= 3 + (5 + 4)
= 12 + 2

for H: P_2J_4 + min. of P_3 and P_4
= 7 + (1 + 7)
= 15 + 2



→ All live nodes having $\angle B > 13$ will all be dead.

← Min. sol $\langle J_2, J_1, J_3, J_4 \rangle$
and min. cost = 13.