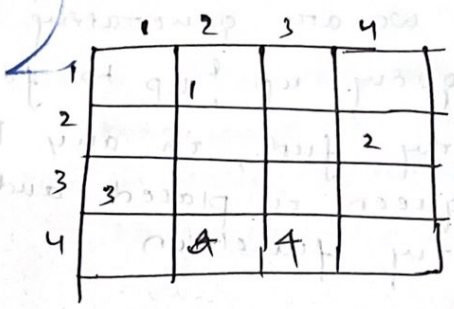
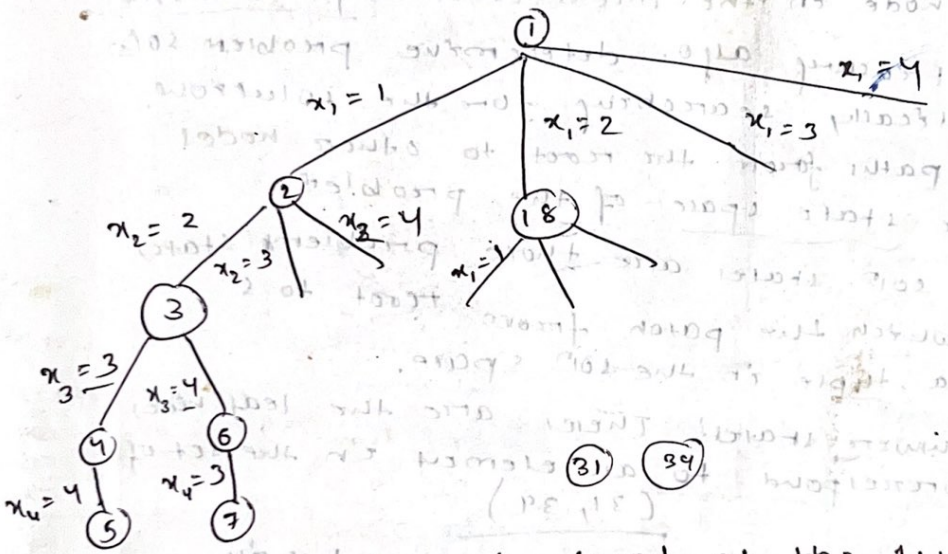
Depth first Node generation



$Q_1 \rightarrow 2$
$Q_2 \rightarrow 4$
$Q_3 - 1$
$Q_4 - 3$

$\{2,4,1,3\}$, $\{3,1,4,2\}$.

(2, 4, 1, 3)
1-18-29-30-31
1-34-35-38-39-
$\{3,1,4,2\}$



$x_1 = 1$ (1st queen is placed at the 1st column.

$x_2 = 2$ (2nd " " " " " 2nd column)

$(x_2 = 3, x_2 = 4)$ 3rd & 4th queen at 3rd & 4th colm.

when $x_2 = 2$ mean 2nd queen is placed, then we have options of 3rd and 4th queen.

Tree tree Organization of the 4-queen soln space.

soln space is all possible spaces without any constraints

If we apply constrain no-two queen can attack then our solution is $\{2,4,1,3\}$ or $\{3,4,4,2\}$.

Depth first node generation with bounding function is called backtracking.

We are generating depth first node then going up, up to find the solution. Bounding fun, in any level of tree only one queen is placed and checking for bounding function

Terminology

* Each node in the tree is called a problem state
* Backtracking also determine problem solution by systematically searching for the solution.
* All paths from the root to other nodes define the state space of the problem.
* The soln states are those problem states (s) for which the path from root to s define a tuple in the soln space.
* Answer state: These are the leaf nodes which correspond to an element in the set of solution. (31, 39)

Live node - node generated but its children not generated.

E-node - Live node whose children are currently being explored

Dead node - generated node not to be expanded further.

Algo

$n = 4$ $\quad n \times n = 4 \times 4$

$k = 1$, $i = 1$ } 1st row 1st column. (Initially we can place bcoz there is not attacking.

If queen's can be placed it returns (Algo place (k, i)) true otherwise it returns false.

$k = 1$

$i = 1$

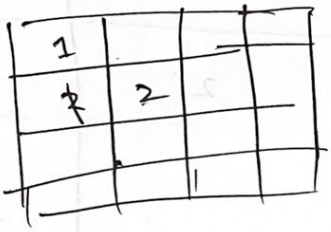
global array so Initially.

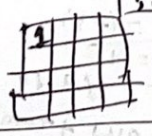
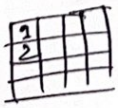
$x[1] = \cancel{0}\ 1$

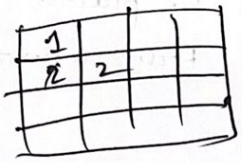
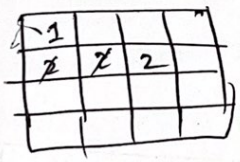
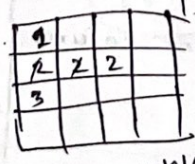
$x[2] = \cancel{0}\ 3$

$x[3] = 0$

$x[4] = 0$.



$k =$	i	$j = 1$ to $k-1$	True/False	soln x
1	1 	1 to 0 (not go inside loop at least once)	True	$x[1] = 1$ (Place 1st queen at 1, 1) $k+1 = 2$
2	1 Place (2,1) place (2nd row 1st col) (same col so attack)  (k i) (2, 1)	$j = 1$ to $2-1$ $= 1$ $x[j] = i$ $x[1] = 1$ (1, 1) // same column.	False means so if place(k,i) then { will not be so executed } $i = i+1$	$i = i+1$ k can't be true bcoz False
2	2 (k i) (2, 2)	$j = 1$ to $2-1$ $= 1$ $x[1] = i$ (1 = 2)		

k	i	j	T/F	SOP
cont. 2	2 (k i: 2,2) 	j = 1 to 2-1 = 1 (x[j] = i) (1 ≠ 2) abs(x[j]-i) = abs(j-k) ⇒ abs(1-2) = abs(1-2) (There R same diagonal)	False that why i = i+1 & k will not be increment	i = i+1 = 2+1 = 3.
2	3 (k i: 2,3) 	j = 1 to 2-1 = 1 x[j] = i (1 = 3) False. (abs(x[j]-i) = abs(j-k)) abs(x[1]-3) = abs(1-2)) abs(1-3) = abs(1-2) (-2) = (-1) (2 = 1) false. comes out from loop & j = 2	True	x[k] = i x[2] = 3 then x[2] = 0 now if i is 3. so k = k+1 (3) x[1] = 0,1 x[2] = 0,3 x[3] = 0 x[4] = 0
k = 2 i = 1, 2, 3				
3	1 (k i: 3,1)  3rd queen at (3,1) not possible same colm.			
when i = 2 (not possible same diagonal)				

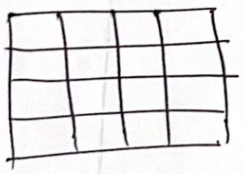
3 ~~k = 3~~ i = 1, 2, 3, 4 it will backtrace & find the place at k = 2 i = 4	i = 3 (not possible) same column. i = 4 (not possible) same diagonal.	false	
k = 2 k = 2 i = 1, 2, 3, 4	i = 4	True.	x[2] = ~~0~~, ~~3~~, 4
k = 3 k = 3 i = 1, 2, 3, 4 (before) now i = ~~1~~,	i = 1 (same col^n) i = 2 (Yes place)	False. True.	
k = 4 i = 1, 2, 3 4	i = 1 (4, 1) - no i = 2 (4, 2) - no i = 3 (4, 3) - no i = 4 (4, 4) = no, so backtrace to k = 3 and i value. 2.		
k = 3 i = 1, 2, 3	i = 3 (3, 3) - no i = 4 (3, 4) - no so again backtrace. Now k = 1, i = 2		
k = 1 i = 1, 2	i = 2 (1, 2) - Yes	True.	x[1] = ~~0~~, 1, 2

again
$k = 2$
$i = 1, 2, 3, 4$ (previous)
now $i = 1, 2, 3, (4)$ ← place here at $(2, 4)$ Q_2

then
$k = 3$
$i = 1, 2, 3, 4$ (previously)
now $i = 1$, ✓ (Place 3rd queen here at $(3, 1)$ Q_3

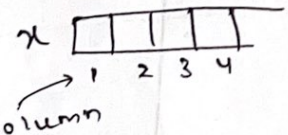
then
$k = 4$.
$i = 1, 2, 3, 4$ (previous)
now $i = 1, 2, 3$ ✓ (place 4th queen at $(4, 3)$ Q_4.

N Queens



many solution –
optimal soln — Dynamic programming

In how many ways you can place them in 4×4 matrix.



$^{16}C_4$

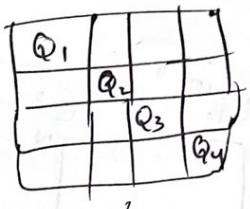
To reduce the size we ll we some bounding function i.e.

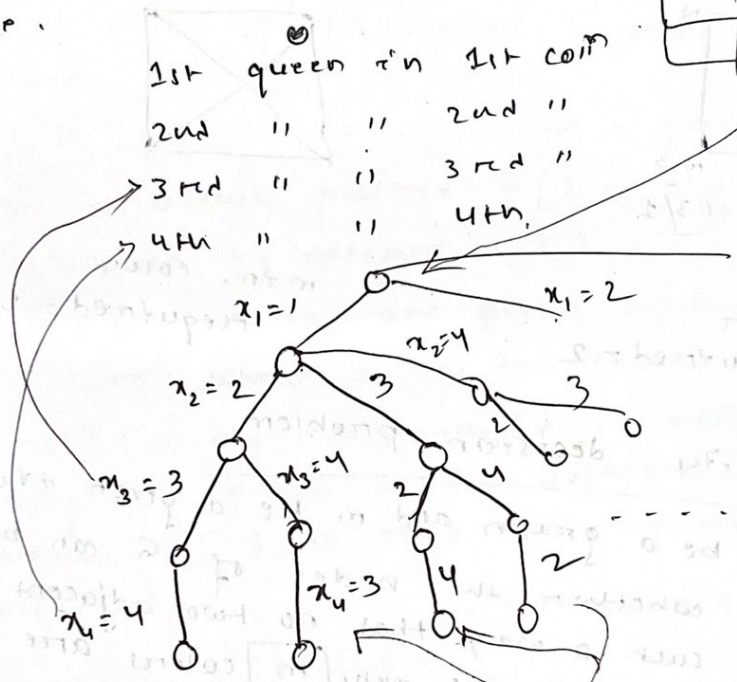
1st queen in 1st row
2nd " " 2nd "
3rd " " 3rd "
4th " " 4th "

So we have to decide the column & what that column name is $x[1]$, $x[2]$ ... where to Try to avoid keeping two queens in same column.

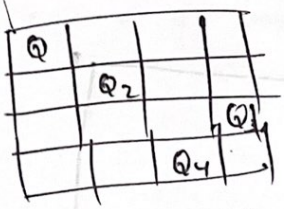
state space tree (diagonal)

without attack find the state space.

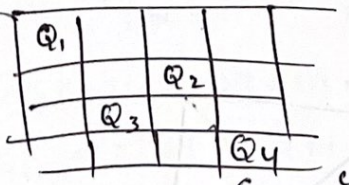




move back from $x_4 = 4$. bcoz we can't change the path of $x_4 = 4$. So move back to $x_3 = 3$



Can we again keep Q_4 anywhere no, Q_3 anywhere no. There is no place. So move back to Q_2 and find is there any way.



Total how many nodes generating (same cond^n row & same coln)

$$1 + 4 + 4 \times 3 + 4 \times 3 \times 2 + 4 \times 3 \times 2 \times 1 =$$

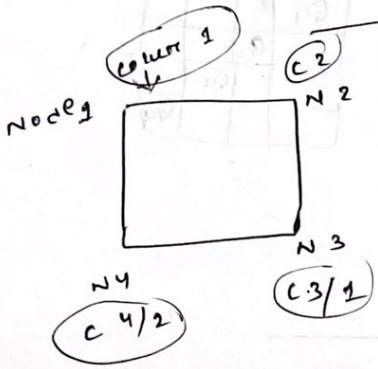
$$1 + \sum_{i=0}^{3} \left[\prod_{j=0}^{i} (4 - j) \right]$$

Product of

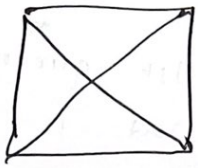
$4 + 12 + 24 + 24 = 65$ nodes

for 4.

Graph Coloring



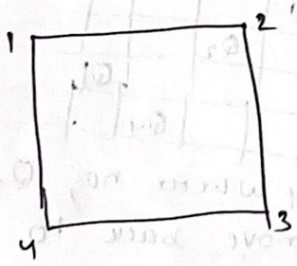
min. color required = 2



min. color required = 4

m-colorability decision problem.

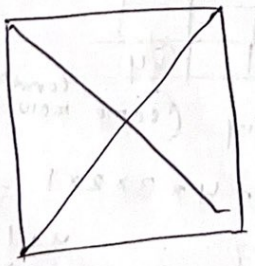
Let G be a graph and m be a given +ve integer. Determining whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used is called M-colorability decision problem.



	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

G(1,3) = 0 G(4,3) = 1

Boolean adjacency matrix.



	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

mcoloring( )

① To determine all the different ways in which a given graph can be colored using at most m colors.

② Suppose we represent a graph by its Boolean adjacency matrix G[1:n, 1:n]; the colors are represented by integers 1, 2, 3 . . . .

③ The solution are given by the n-tuple (x1, ... xn) where xi is the color of node i.

④ Function mcoloring( ) is being by first affixing the graph to its adjacency matrix, setting the array x[ ] to zero and involving statement mcoloring(1);

Node 1 is given with color 1.

x[1] = 1
x[2] = 0
x[3] = 0
x[4] = 0

Node 4 is initialized to color 0. (No color)

k = 1
repeat
    nextvalue(k)
until k = n+1

nextvalue(k)
repeat
    (do while in your C prog)
    x[k] := (x[k] + 1) mod (m+1)
    // x[k] := 0+1 mod (2+1)
    x[1] = 1 % 3 = 1
    if (x[k] = 0)
        x[1] = 0 false
    for j = 1 to n (4)
until (x[k] = j)
    2 colors for 1st edge

x[1] = 1

for (j = 1 to 4 do.

	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

{ j = 1

(1) if ((G[k, j] ≠ 0) and (x[k] = x[j]))

G(1, 1) ≠ 0

If (0 ≠ 0) and somewhere // adjacent vertices

false

false.

then If take next j value i.e. 2.

j = 2

if (G(1, 2] ≠ 0) and x[k] = x[j]

1 ≠ 0 x[1] = x[2]

True ← + → (1 = 0)

false.

so false.

Now j = 3 if (G[1, 3] ≠ 0) and x[k] = x[j]

0 ≠ 0

false.

Now j = 4 if (G[1, 4] ≠ 0) and x[k] = x[j]

(1 (Non zero) ≠ 0) x[1] = x[4]

1 = 0

True + false.

false.

Come out of loop for j = 5

if (j = n + 1) then return; // new color found.

5 = 4 + 1

true. That why it goes to next value (k) and the x[i] is 1.

if (x[k] = 0) then return.

b

false

if (k = n) then //

false.

else. m colours (k + 1) i.e. k = 2

Now we are ready to colour node 2 and successfully coloured node 1.

$k = 1$

$x[1] = 1 \% 3 = (1)$

$j = 1, 2, 3, 4, 5$ — out of for loop ($j = n+1$)

$x[1] = \emptyset\ 1$

$x[2] = \emptyset\ \cancel{1}\ 2$

$k = 2$

$j = 1, 2, 3, 4, 5$

$x[3] = \emptyset, \cancel{1}\ 1$

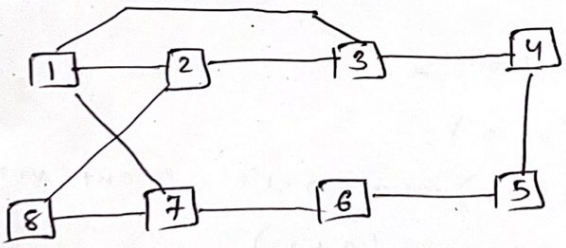
$k = 3$

$j = \cancel{1}\ \cancel{2}\ \cancel{3}\ \cancel{4}\ 5$

$k = 4$

$j = \ldots$

Hamiltonian cycles:-



1 - 2 - 8 - 7 - 6 - 5 - 4 - 3 - 1

Note: each node traversed once, starting node, end node same.

Travelling salesperson problem (shortest path)

1. NP

Hamiltonian cycle is a round-trip path along n-edges of G that visits every vertex once and returns to its starting position.

Defn

$G(3,7) = 0$

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	1	0
2	1	0	1	0	0	0	0	1
3	1	1	0	1	0	0	0	0
4	0	0	1	0	1	0	0	0
5	0	0	0	1	0	1	0	0
6	0	0	0	0	1	0	1	0
7	1	0	0	0	0	1	0	1
8	0	1	0	0	0	0	1	0

Initial

$x[1] = 1$ $\quad$ $x[5] = 0$

$x[2] = 0$ $\quad$ $x[6] = 0$

$x[3] = 0$ $\quad$ $x[7] = 0$

$x[4] = 0$ $\quad$ $x[8] = 0$

---

final

$x[1] = 1$ $\quad$ $x[5] = \not{0}\ \not{5}\ 6$

$x[2] = \not{0}\ \not{x}\ 2$ $\quad$ $x[6] = \not{0}\ \not{6}\ 5$

$x[3] = \not{0}\ \not{x}\ \not{2}\ \not{3}\ 8$ $\quad$ $x[7] = \not{0}\ \not{4}\ 4$

$x[4] = \not{0}\ \not{x}\ \not{2}\ \not{3}\ \not{4}\ 7$ $\quad$ $x[8] = \not{0}\ \not{5}\ 3$

Algorithm Hamiltonian(k) $\quad$ Also next value(k)

next value(k)

if

⋮

k should from start from 2.

next value(k = 2)

$x[k] = (x[k] + 1) \bmod (n+1)$ //next vertex

$x[2] = (x[2] + 1) \bmod (8+1)$

$= 1 \bmod 9 = (1)$

$x[2] = 1.$

if $(x[2] = 0)$ then return.

NO (If we did find Hamiltonian cycle this cond^n true)

if $(g[x[k-1], x[k]] < > 0)$ then — non zero

if $g(x[2-1], x[2])$

$(g(1, 1) = 0 <> 0)$ (from matrix)

Is it non zero. No it zero that's why condition is false. It will come out from if (did) clause ending at 24 line.

25th line until (False) mean again it will repeat from line 13.

line 15 $x[k] = (x[k] + 1) \bmod (n+1)$

$1 = (1+1) \bmod (8+1)$

$= 2 \bmod 9 = (2)$ $\quad$ $x[2]$ now 2

line 17 if (g [x [k-1], x [k] < > 0) then
2-1 2

g (1 , 2) = 1 Now it is True. so goes outside the loop

line 19 // is there an edge.

for j = 1 to k-1 do. (1 to 1)
2-1 = 1

line 20. if (x [j] = x [k])

x [1] = x [2]

(1 = 2) False. thats why take next j value.

line 22 if (j = k) then j ≠ 2
2 2 True. when j = 2 it comes out from loop.

line 23 if ((k < n) or (k = n)

and g [x [n], x [1] < > 0)) then

return;

if (t_1 or (t_2 and t_3)) like this

T or something is always True.

F or some cases we have to check.

(k < n)

2 < 3 = True so return. x [2] = 0 ✗ 2

Now come to line no. 6

if (x [k] = 0) then return;

False.

if (k = n) then write (x [1 : n]);
2 3 → No thats why else part i.e.

Hamiltonian (k+1)

2+1

= 3

k = 3

1 % 9 = 1 only.

x [3] = 1.

line 17. x [k-1] x [k] x [3] = 0 ✗ 1
3-1 3

g (2 , 1)

for j = 1 to k-1 do. 1 to 3-1 = 1 to 2

if it is 3 it will come out from loop.

∴ 2 % 9 = 2 x [3] = 0 ✗ 2 ✗ 3

Branch & Bound

This is similare to backtracking in the sense that it also use the state space tree for solving the problem. solution is represented like a state space tree. But it is usefull for solving optimization problem. and particularly minimization problem.

job sequencing:

$$jobs = \{J_1, J_2, J_3, J_4\}$$

$$P = \{10, 5, 8, 3\}$$

$$d = \{1, 2, 1, 2\}$$

lets say

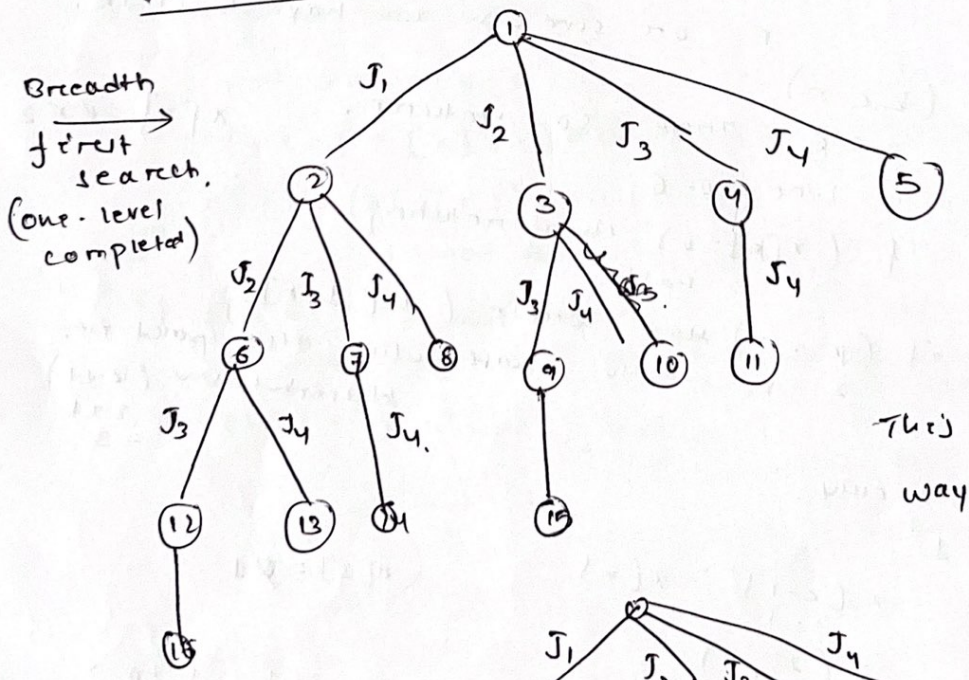
$$s = \{J_1, J_4\}$$

$$s = \{1, 0, 0, 1\}$$

fixed size

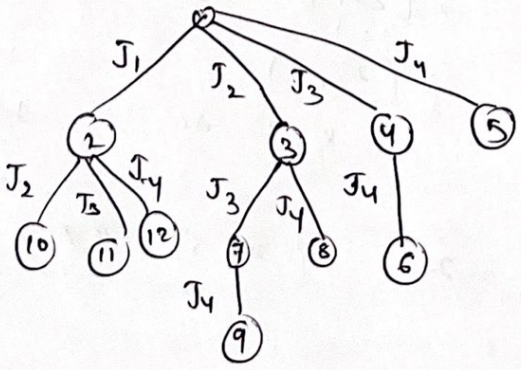
variable size (subset method)

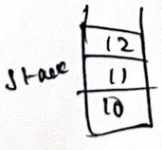
1st method

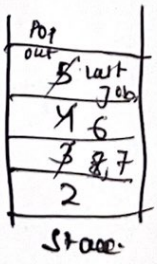


This is one way of generate model.

2nd method







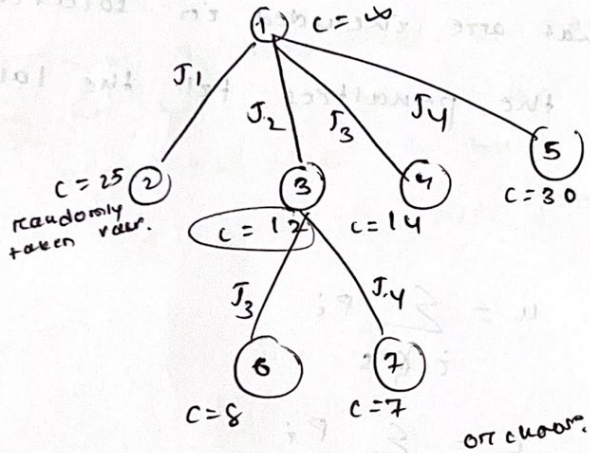
previously (1st method) we are using Queue and 2nd method we are using stack for node exploration.

BFS with Queue → FIFO Branch & Bound.

Stack → LIFO - LIFO BB.

List cost Branch & Bound → LC - BB

LC - BB



I.M.P.

In FIFO - BB we take the Node 2 from i.e. c = 25 on account. for processing.

In LIFO - BB we we the last node i.e. node 5 with c = 30 for processing.

In LC - BB we consider c = 12 (least cost) i.e. node 3 for exploration.

Job - sequencing with deadline

Basically it is a maximization problem, where profits are given but here we are considering this as a minimization problem with penalty value. (BB can solve minimization problem)

So penalty means we have to pay. we will loose, we will not gain anything.

The objective is to select the job and finish them in such a manner the penalty is minimum. so we are minimize the penalty and thereby maximizing the profit.

jobs.	1	2	3	4
penalty.	5	10	6	3
deadline	1	3	2	1
time.	1	2	1	1

U = Upper bound = sum of all penalties except those that are included in solution

$\hat{c}$ = sum of all the penalties till the last job considered.

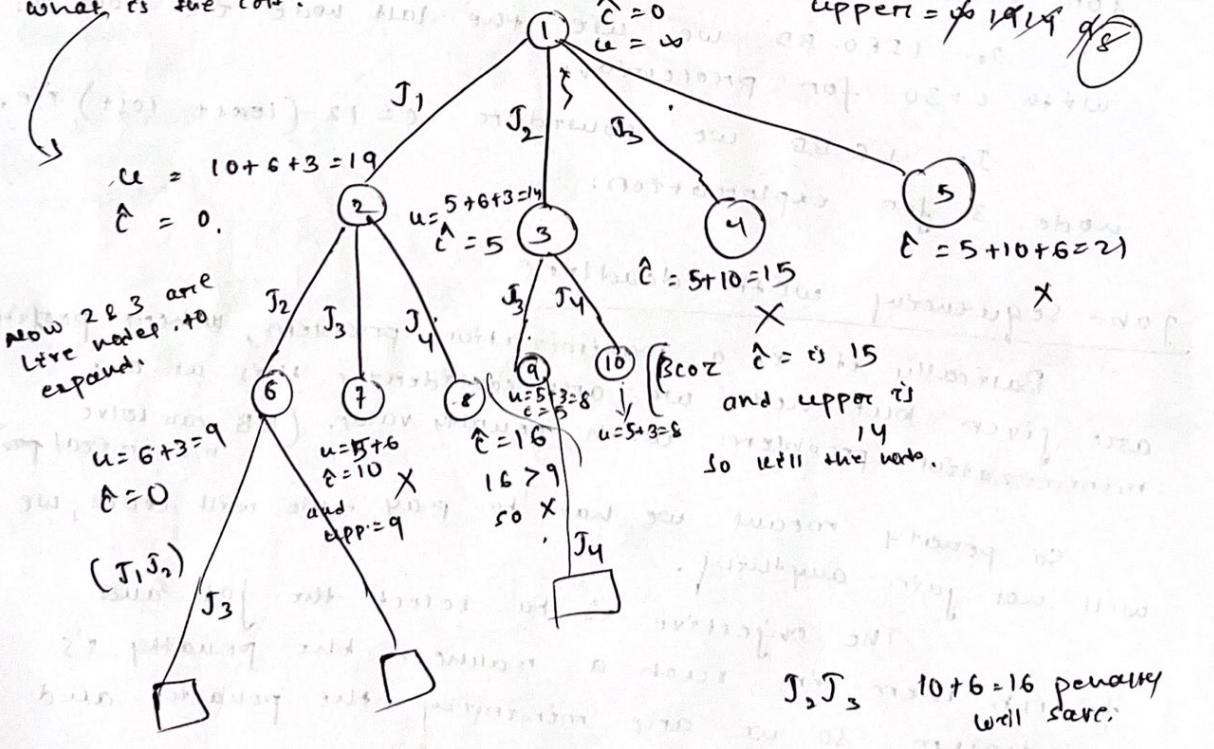
s =

$$u = \sum_{i \notin s} p_i$$

$$c = \sum_{i \in s_k} p_i$$

If J_1 included what is the cost.

new upper value smaller than prev. then considered.



doesn't

$J_1 + J_2 + J_3$ is possible or not.

$J_2 + J_3 + J_4$ is it possible or not

$J_2 J_3$ 10+6 = 16 penalty will save.

penalty we will be paying of 8

job 1 deadline 1, job 2 deadline 3, job 3 deadline 2.

J_1	J_2	J_3
1	3	1
1	2	1

process time →

Total time.

1 + 2 = 3

Though J_2 required 2 hours, 1st hours J_1 remain for J_2. J_3 can't be done based on their time. bcoz it needs 2 hours.

So job ☐ (this job can't be done.

J_2	J_3
3	2
2	1

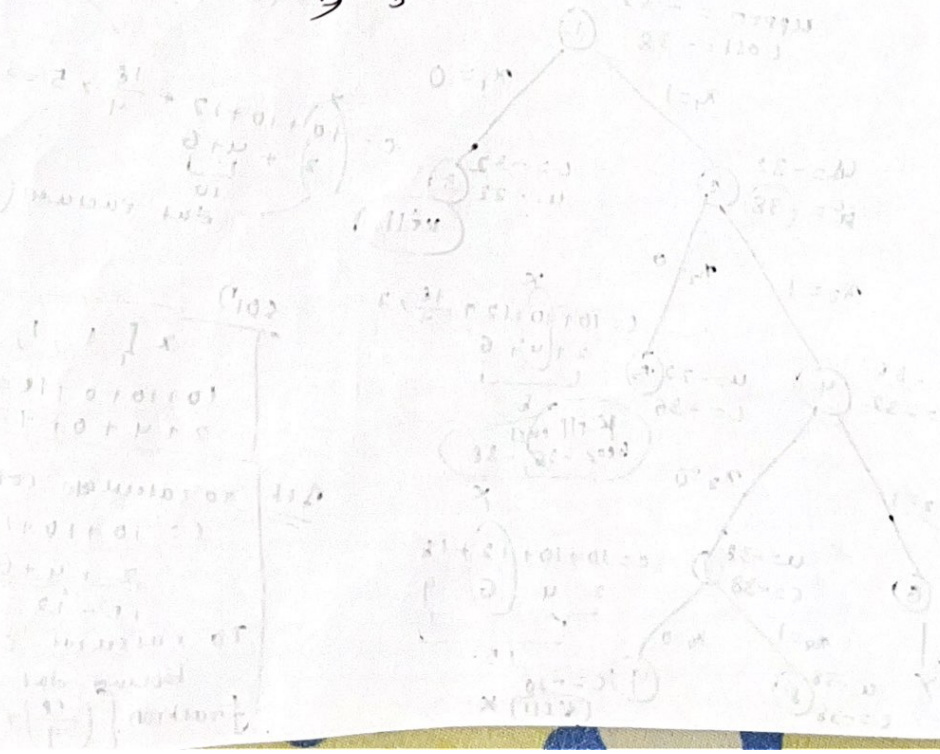
π_1

So after this sequence is feasible. So the feasible π's

$C = 5$

$a = 8$

x	J_1	J_4	= 8
	5	3	



0/1 knapsack problem

	1	2	3	4
Profit	10	10	12	18
weight	2	4	6	9

$m = 15$
$n = 4$

The objective is to fill the bag with those objects such that the total weight should be less than or equal to capacity i.e. 15.

- either take full or don't take.

Here again it is a maximization problem so we have to convert the profits to -ve value, for branch & bound soln.

We will solve it by **LC-BB** and

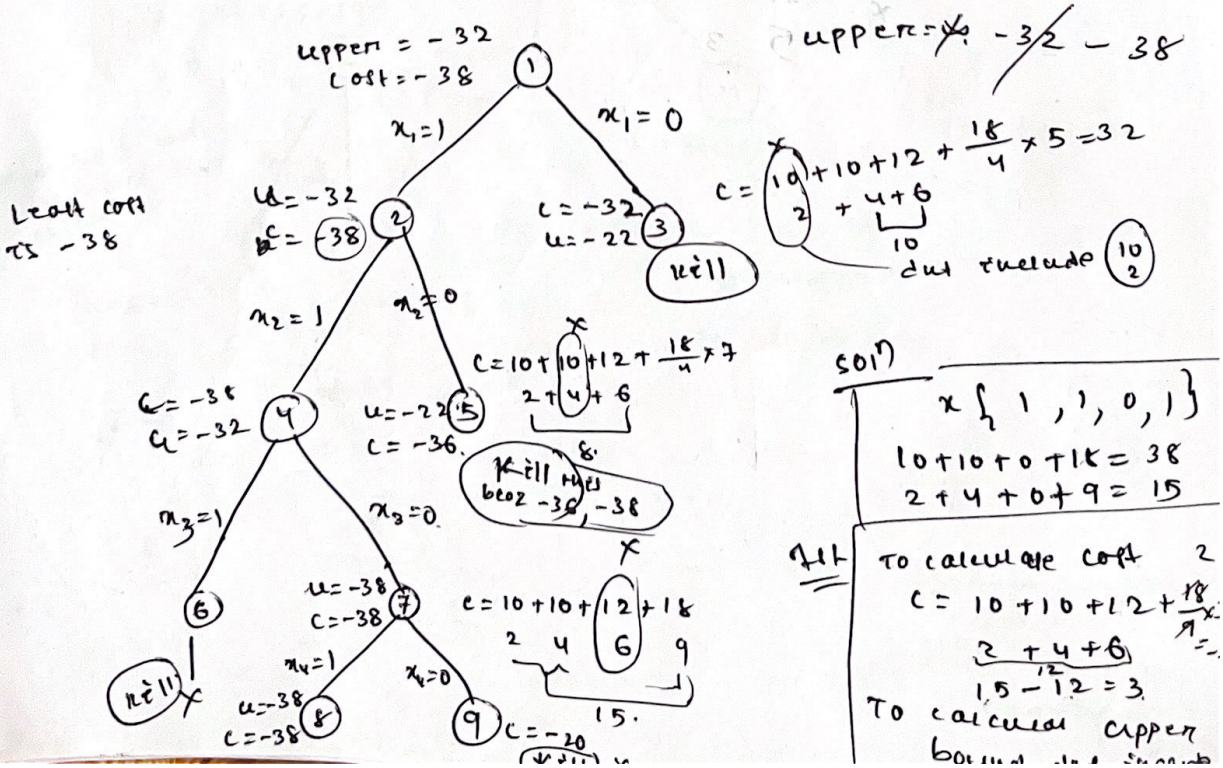
upper bound: $u = \sum_{i=1}^{n} p_i x_i \quad \leq m$

$$cost = \sum_{i=1}^{n} p_i x_i \quad \text{(with fraction)}$$

Do I want the solution in subset form or fixed tuple size solution.

$S = \{x_1, x_3\}$

$S = \{1, 0, 1, 0\}$



upper = -32 - 38

$c = \left(10 + 10 + 12 + \frac{18}{4} \times 5\right) = 32$

$c = \left(\begin{matrix}10\\2\end{matrix}\right) + 10 + 12 + \frac{18}{4} \times 5 = 32$

$2 + 4 + 6$ (10)

don't include $\left(\begin{matrix}10\\2\end{matrix}\right)$

$c = 10 + \cancel{10} + 12 + \frac{18}{4} \times 7$

$2 + 4 + 6$

soln

$x\{1, 1, 0, 1\}$

$10 + 10 + 0 + 18 = 38$

$2 + 4 + 0 + 9 = 15$

To calculate cost

$c = 10 + 10 + 12 + \frac{18}{\cancel{6}} \times$

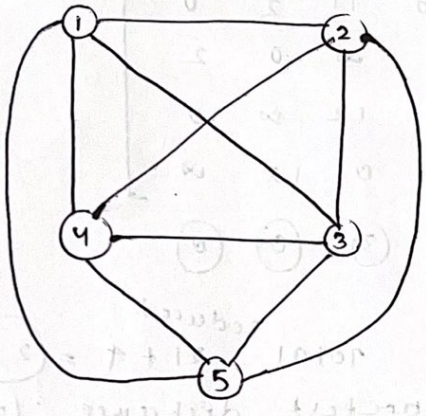
$2 + 4 + 6$ (12)

$15 - 12 = 3$

To calculate upper

Travelling Salesperson Branch & Bound

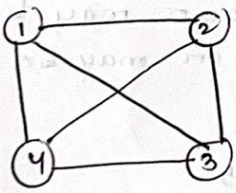
ex-1



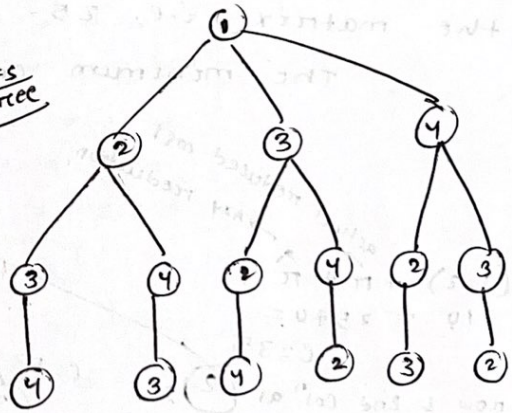
	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

cost adjacency matrix

ex-2



BFS tree



	1	2	3	4	5	
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

	1	2	3	4	5	
1	∞	10	20	0	1	10
2	13	∞	14	2	0	2
3	1	3	∞	0	2	2
4	16	3	5	∞	0	3
5	12	0	3	12	∞	4

21 (reduced row)

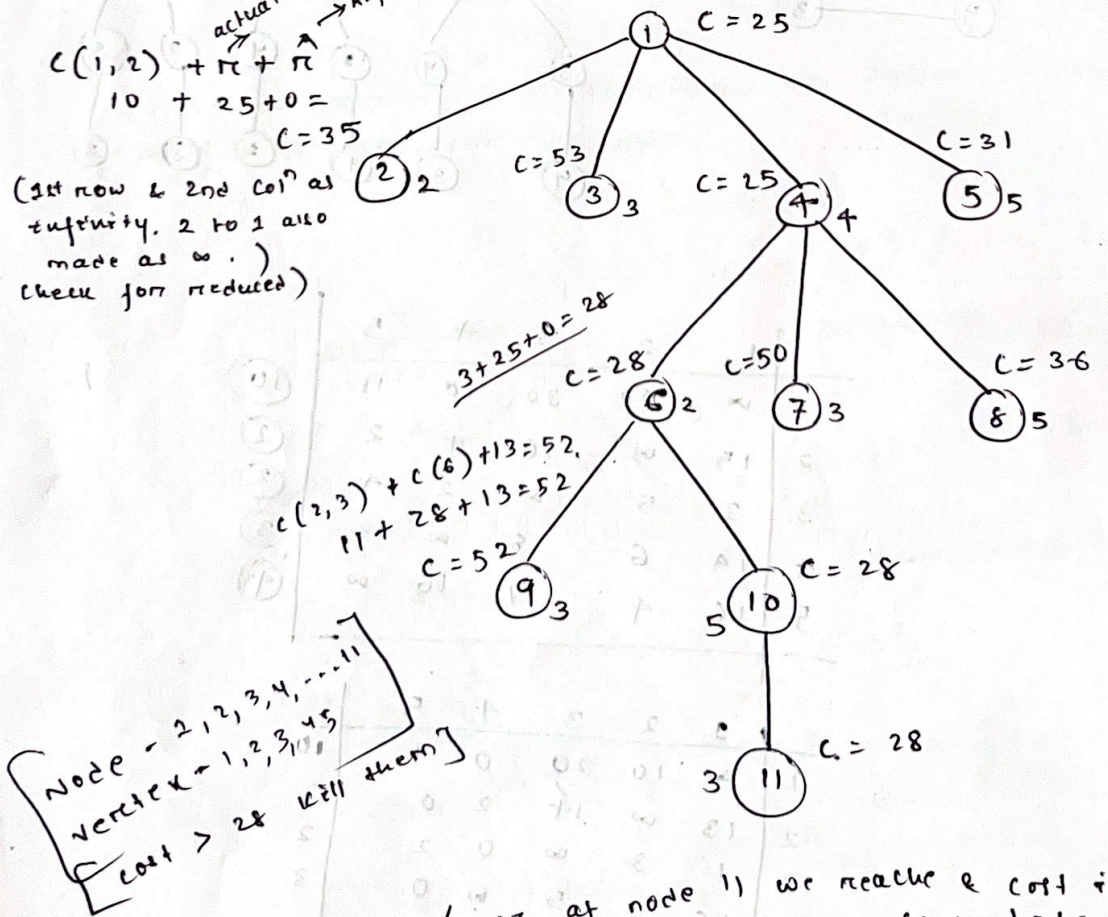
Reduced cost = 25

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞
	(1)	(0)	(3)	(0)	(0)

Total reduced $21 + 4 = (25)$

We have found the shortest distance from the matrix i.e. 25.

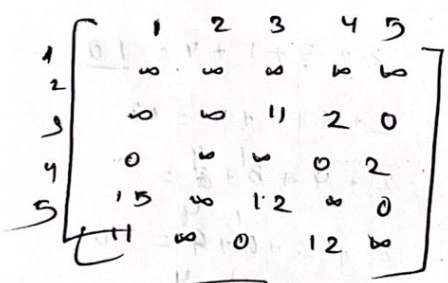
The minimum cost of tour may be 25. or may be more than 25.



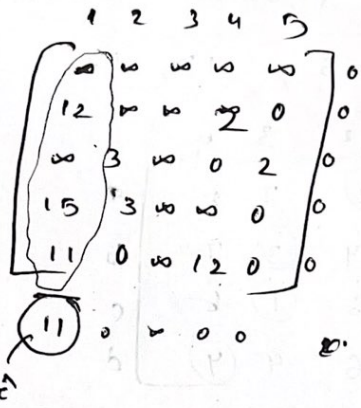
upper = 28 (bcoz at node 11 we reache & cost is 28 so update it)

then kill the nodes bcoz $28 < 35, 53 \ldots 31, 52$

Once we reach leaf node we will update upper

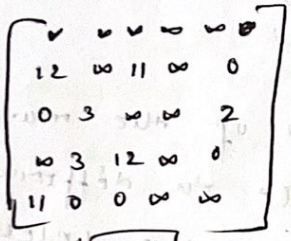


1st

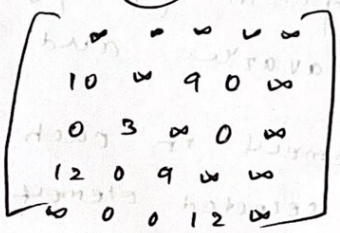


2nd

$$c(3) = c(1,3) + \pi + \hat{\pi}$$
$$= 17 + 25 + 11$$
$$= 53.$$



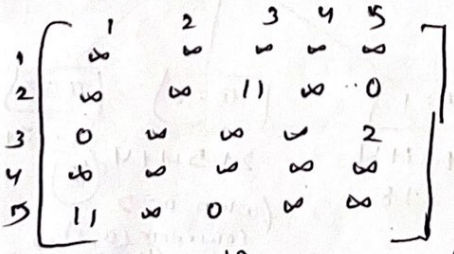
3rd



4th

If I am follow explore children of 2, 3, 4, 5 then it is FIFO. But least cost is 25. So it is LC-BB.

Consider the 4th matrix for expanding the 6th, 7th, and 8th node.



5th

1, to 4, to 6(2nd)

No reduction

$c(4,2) + c(4) + \hat{r}$ from 4th matrix

$3 + 25 + 0 = 28$

Again after explore of 6th, 7th, and 8th node consider the minimum live node. i.e

$(2, 3, 5, 6, 7, 8) = (35, 53, 31, 28, 50, 36)$

1 - 4 - 2 - 5 - 3 - 1

Assignment Problem Branch & Bound.

13, 14, 17.

	job 1	job 2	3	4	
	9	(2)	7	8	a
	6	4	(3)	7	b
	5	8	(1)	8	c
	7	6	9	(4)	d

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

$2+6+1+4 = 13$

$2+4+8+8 = x$

$2+3+8+4 = 10$

$2+7+8+4$ 14

eg. A taxi firm (Ola/Uber) has 3 taxis (capacity) available and there

Select one element in each row of the matrix so that all selected elements are in different column and the total sum of the selected elements is the smallest possible.

objective - atleast a person will have a job the cost should be lowest, or optimized assignment. Firstly find the lower bound i.e. the least cost.

	1	2	3	4
a	9	(2)	7	8
b	6	4	(3)	7
c	5	8	(1)	8
d	7	6	(2)	4

4 is min when a=1 — 1 min — 4 min

(a,1) + min. of 2, 3, & 4th column

LB = min. value of each row.
= 2 + 3 + 1 + 2 = (8)

a=1 | a=2 | a=3 | a=4

9 + 4 + 1 + 4 = 18 (a,1)

2 + 5 + 1 + 4 = (12) (when a=2, consider (a,2) = 2)

7 + 5 + 4 + 4 = 20

8 + 5 + 4 + 4 = (17)

Here job 2 is assigned to a

cut the b. or b row

b=1 | b=3 | b=4

(b,1) 6 + 2 + 1 + 4 = (13) — job 1 to person b

(b,3) 3 + 5 + 2 + 4 = (14) — min. of 1st row

(b,4) 7 + 5 + 2 + 1 = (15)

optimal soln.

2 + 6 + 1 + 4.

(13) is upper bound

a=2, b=1, c=3, d=4 | a=2, b=1, c=4, d=3

15-puzzle Branch & Bound Problem

1	3	4	5
2		5	12
7	6	11	14
8	9	10	13

Empty spot (ES)

An arrangement.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

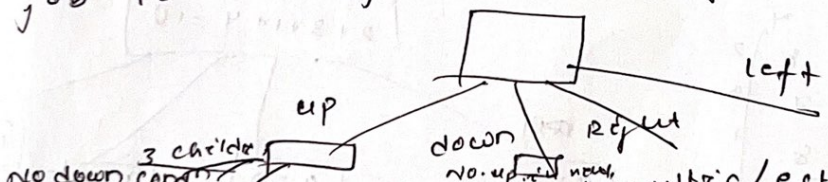
Goal arrangement.

ES/tiled can move left, Right, up and down.

From the starting arrangement we have to move tiles in such a manner to reach the goal state in a shortest path.

Basically it's a permutation problem. In the starting state we can have $16! = 20.9 \times 10^{12}$ permutation arrangement.

The state space is huge. Its a difficult job to reach from start to goal state.



we should have some heuristic/estimate with the help of which, we can reach goal state early.

By this heuristic/estimate we should not find all possible ways, we should branch & bound some of them. If I find any movement is promising then I should apply boundry then branching for that node.

$$\hat{c}(x) = f(x) + \hat{g}(x)$$

$\hat{c}(x)$ = estimated ~~cost~~ min cost to reach to the goal node.

$f(x)$ = Is the length of the path from the root to node x.

$\hat{g}(x)$ = Is an estimate of length of a shortest path from x to a goal node in the subtree with root x.

One possible choice of $\hat{g}(x)$ is

$\hat{g}(x)$ = Number of non blank tiles not in their goal position.

$\hat{c}(x) = g(x) + \hat{p}(x)$

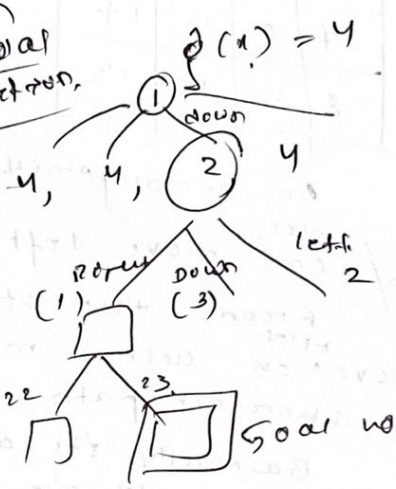
root to node x

$\hat{p}(x)$ from x (current node) to goal pos^n.

→ No. of non blank tiles not in the goal position

1	2		4
5	6	3	8
9	10	7	11
13	14	15	12

4 positions not in goal position.



Assume Puzzle

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