→ Graph (V, E) V = {}
E = {}

- Adjacency Matrix [ ] number MXM
  Dense (edges are large)
- Adjacency List
  Sparse (less number of edges)

In comparison to Fully Connected Graph.

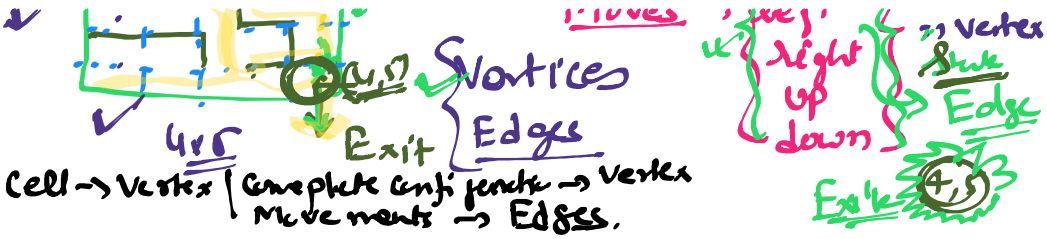
Problems: Maze Problem (Maze Game)

Path finding Problem

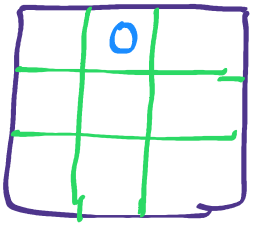
{Chakravyuh in Mahabharata}

'1' Obstacles

Grid cell



② Tic-Tac-Toe



Two-player

- Computer → X
- Human → O

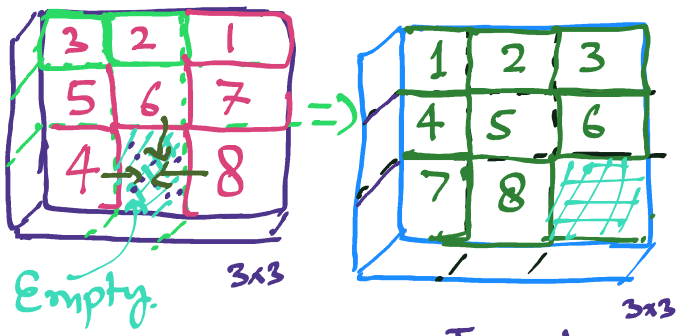
Win: Any row/column/diag you same symbol

3x3 5x5 7x7 12x12

③ 8-puzzle

Block Game

Only: You can slide the blocks (Not to take it out)



Given

Target

Moves: left, up, right, down

least # moves

4×4
5×5
6×6
...

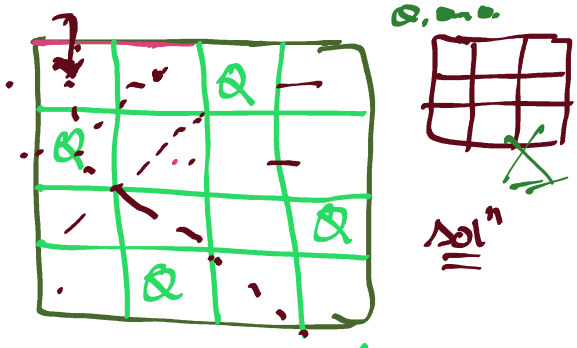
④ n×n Queen

Q_1, Q_2, Q_3, Q_4

Q

4 - Queens

Queens are not in attacking positions.



Sol"

No two Queens are in same row / column / diagonal

Every row, Every column must have a Queen

(5). Chess Board

multiplayer · (2 Player)

Computer → 6

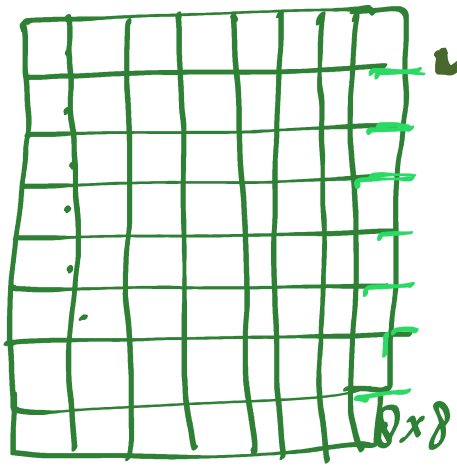
Human → 6

{King, Queen, Rook, Bishop, Pawn, Knight}

Win: Checkmate

Moves:

Reinforcement Learning



Past History

6. Traveling SalesMen Problem

NP-Complete

Hamiltonian Cycle

Salesmen.

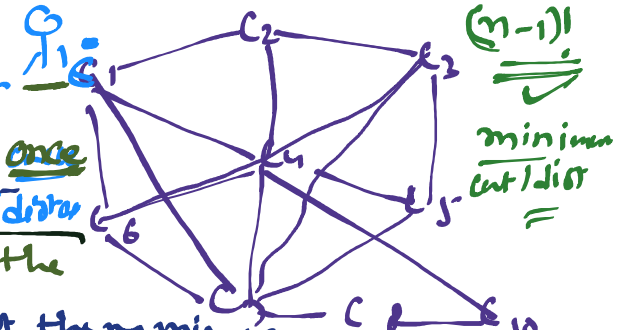
Network of Cities

SM:

Visit all the cities once

in min time / cost / distance

and come back to the start city



start

$(n-1)!$

minimum cost / dist

Let 10 cit: Net the minimum one

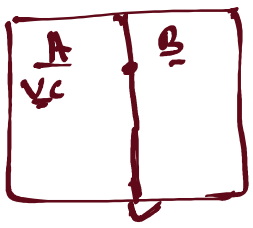
$1 \to 3 \to 2 \to 4 \to 5 \to 6 \to 9 \to 7 \to 8 \to 10 \to 1$

$1 \to 9 \to 3 \to 2 \to 6 \to 7 \to 8 \to 5 \to 4 \to 10 \to 1$

2.43×10^{18} sec 21 city 100 cities $\simeq 10^{157}$ sec

7. Robot Navigation:

Vacuum Cleaner Robot

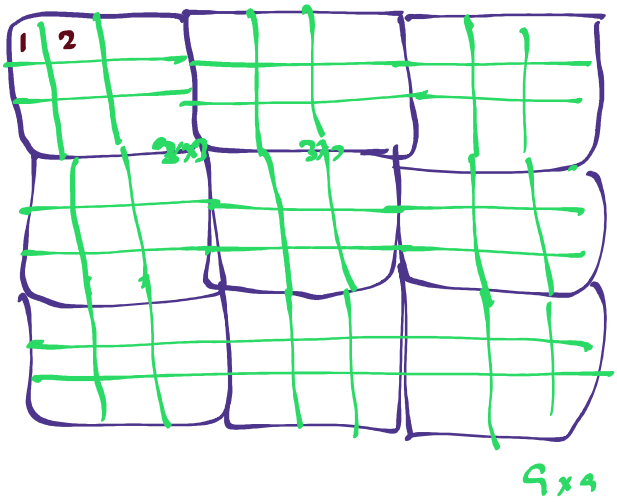


⑧ Sudoku Problem

1-9

Valid Complete

Winning Strategy



(9) Rubik's Cube

Computers

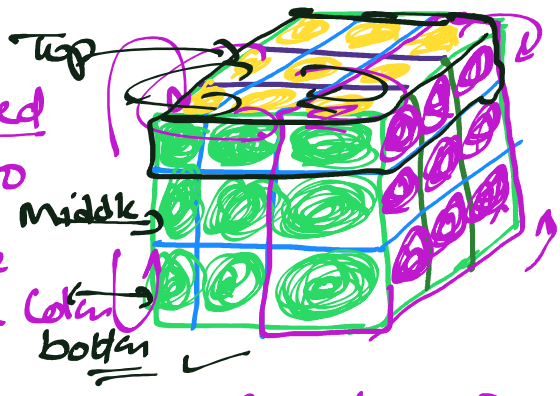
Solved

jumble it and try to

Solve. every face

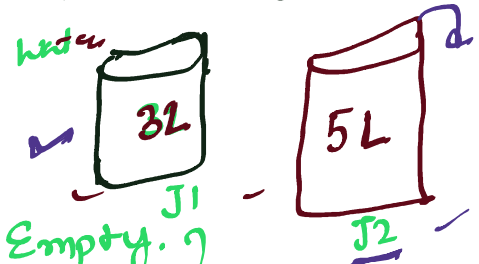
must have same color

bits



Moves: Rotation {Clockwise, Anticlockwise
top to bottom, bottom to top}

(10) Water - Jug Problem Puzzle



Empty. }
Full the jars }
J1 & J2
J2 → J1

1L litre

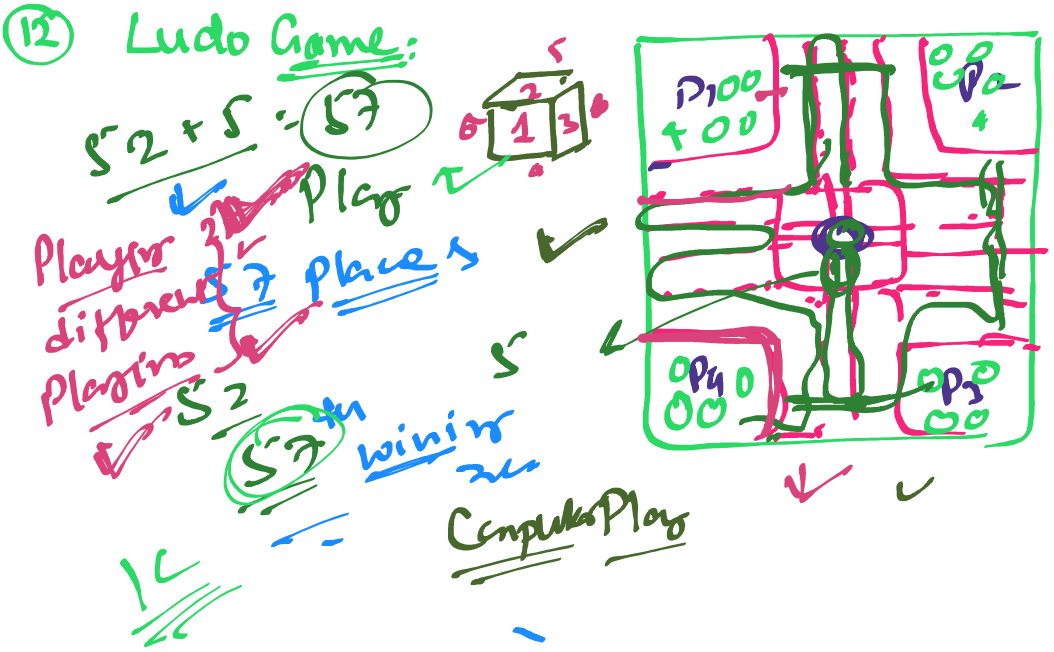
How will you measure

1L - Jar2

2L
4L
6L
7L
8L
:
:

(11) Man-goat-Cabbage-lion River Crossing Problem

(12) Ludo Game:



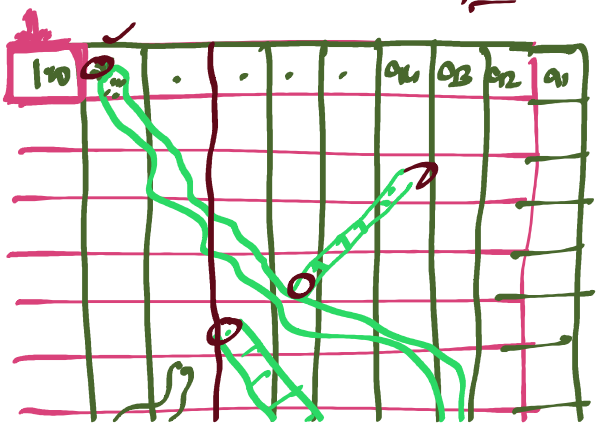
(13) Fair Play

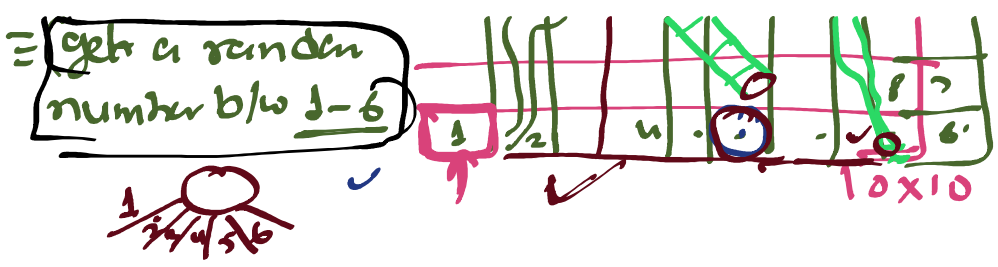
Snake - ladder

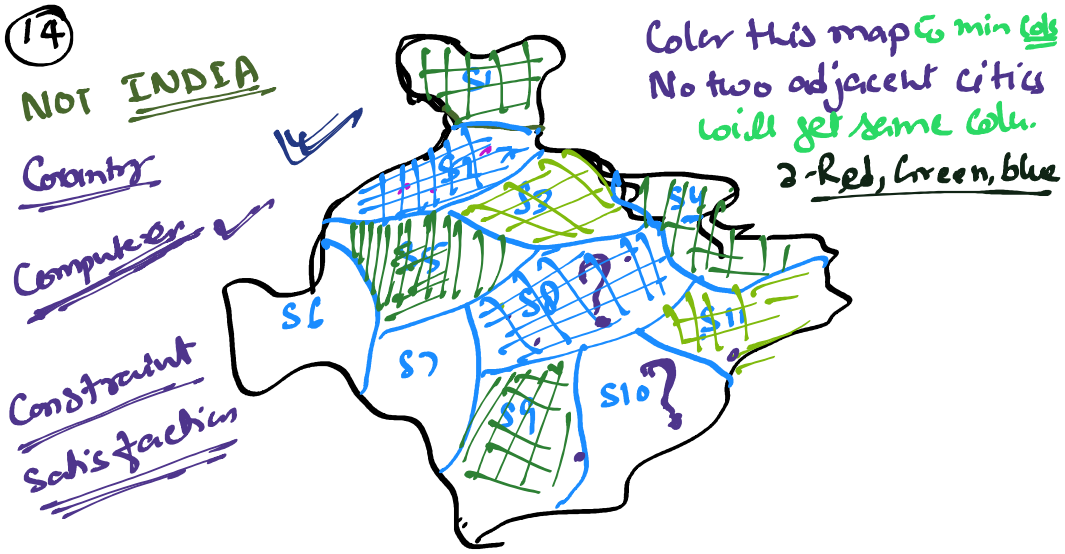
Cubicine

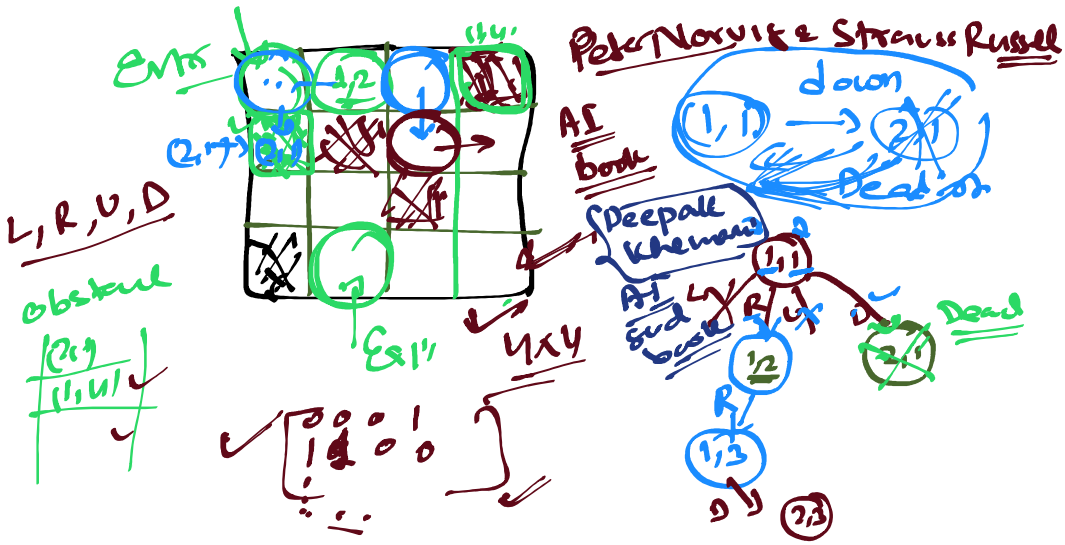
Game Play rule

rolling the dice




get a random number b/w 1-6
1
2
3
4
5
6
10x10


14
NOT INDIA
Country
Computer
Constraint satisfaction
Color this map to min cols
No two adjacent cities will get same colu.
3-Red, Green, blue
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11



2^{30} nodes in the graph

BFS, DFS

NOT POSSIBLE in Maven

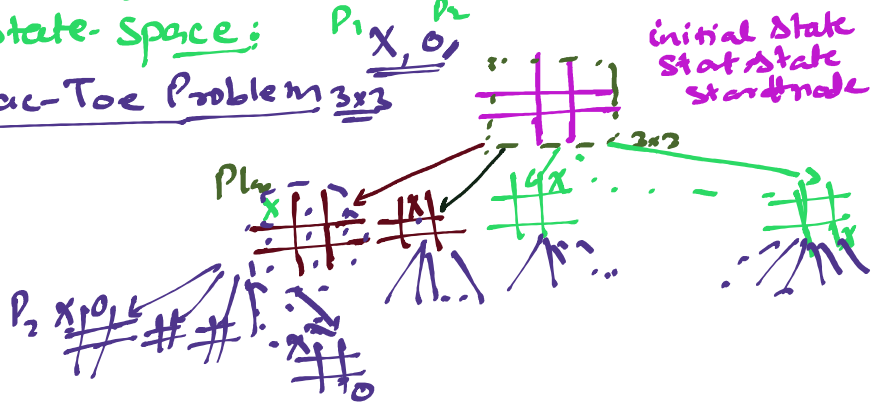
memory

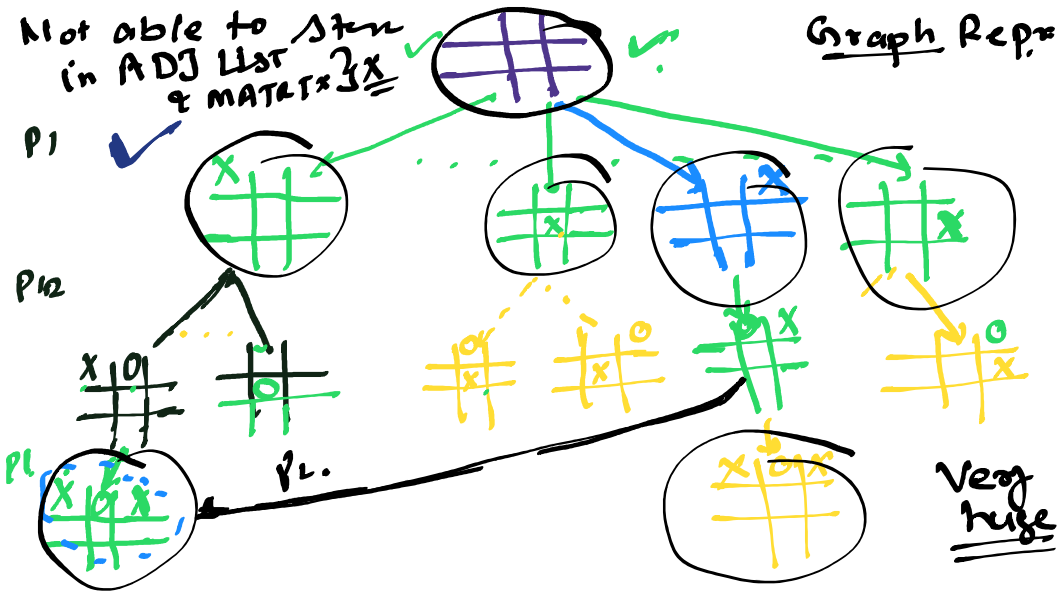
OPTIMAL SO: Explore all the sol" and get the best one.

Every Node might be soln : Heuristic approach
→ Genetic Algo.
Nature Inspired PSO, Simulated Anne

→ Number of nodes in the Graph represents the state-space;

Tic-Tac-Toe Problem 3×3





-> Problems : Start / Initial Configuration / Node
Vertexis known.

State / Search Space

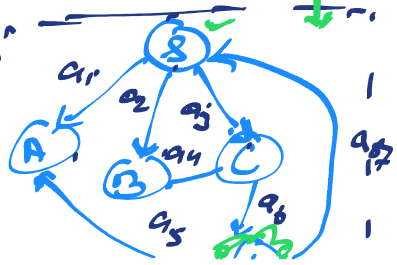
Actions : Edges / links

Set of all possible State changes defines the STATE

G = {S, A, B, C, D}

edge : {$a_1, a_2, \ldots$

(S, A), (S → B)



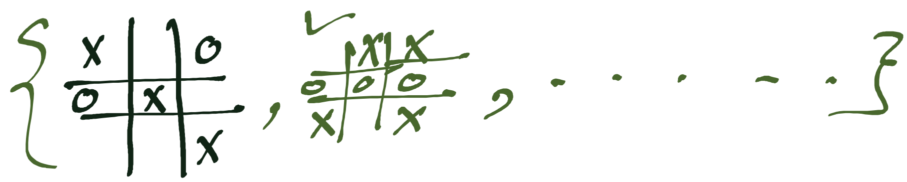
space (search space), . . . ,

define the sol^n as goal state.

Wining Condition / Draw

Configuration which fulfills the wining Condition we call it goal state.

TicTacToe . Wining States   Goal STATES

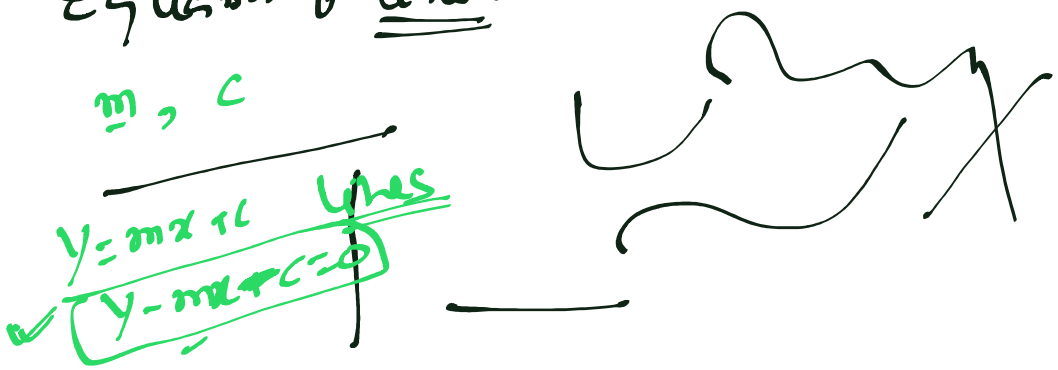


P1 → (9) P2 → (8) P1 → (7) P2 → (6) B →

How Soln look like : We must know.

Equation of line :

m, c



→ function to identify a goal state.

Bool goalState(Si) / bool goalTest(Si)

bool goalState(Si)
{ if Si is the goal state
return True
else return False.
}

Every row is filled with 8 ✓

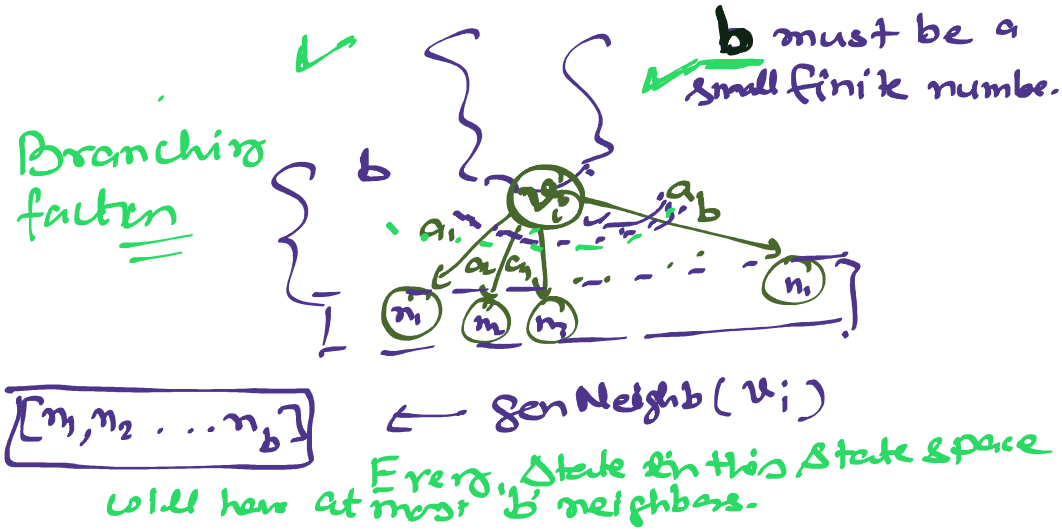
→ We can not store the Graph / State space in Memory, we define a function which generate neighbours.

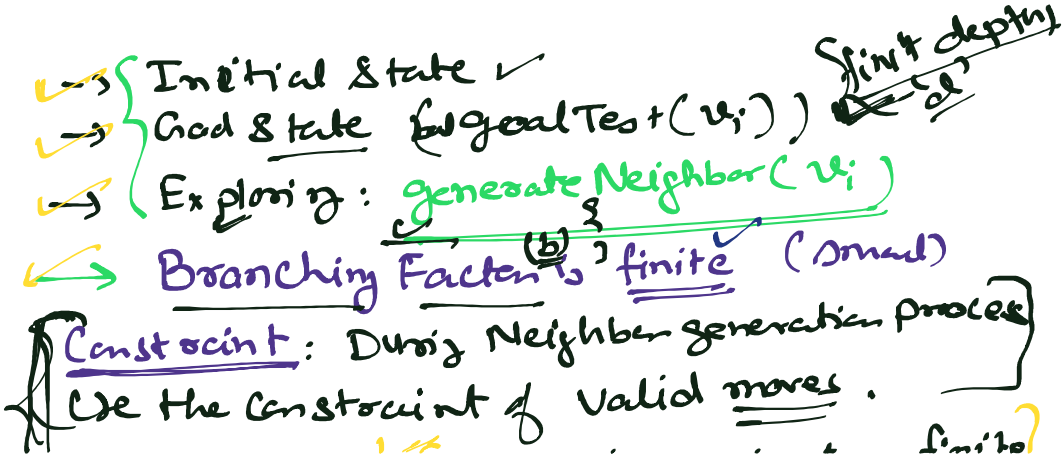
→ We are at any state / node, generate the neighbours by performing actions / moves.

{New nodes, Repeated nodes}

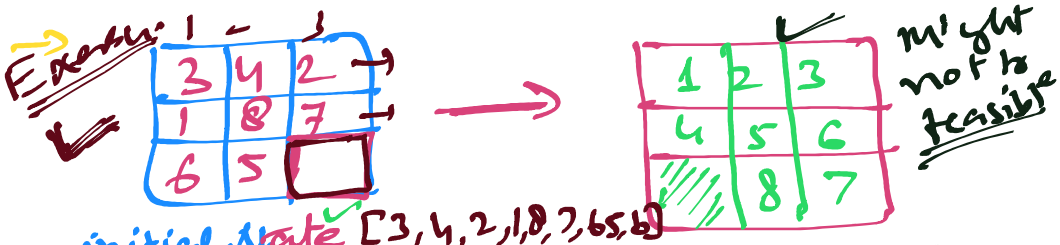
$[n_1, n_2, \ldots n_k]$ ← generateNeighbour(current state S_i)

List becomes empty if game is Draw


b must be a small finite numbe.
Branching factor
b
v_i
a_1
a_b
a_2
a_3
n_1
n_2
n_3
n_b
$[n_1, n_2 \ldots n_b]$
← genNeighb(v_i)
Every State in this State space will have at most 'b' neighbors.


Initial State
Goal State (isGoalTest(v_i))
Exploring: generateNeighbor(v_i)
Branching Factor is finite (small)
Constraint: During Neighbor generation process Use the constraint of valid moves.
limited
first depth 'd'

Assumption: Goal Node exist at a finite depth 'd' (Infinite depth?)



initial State [3,4,2,1,8,7,6,5,b]

Represent it as graph problem.

State, actions/move {left, right, up, down}

STATE-SPACE Compute

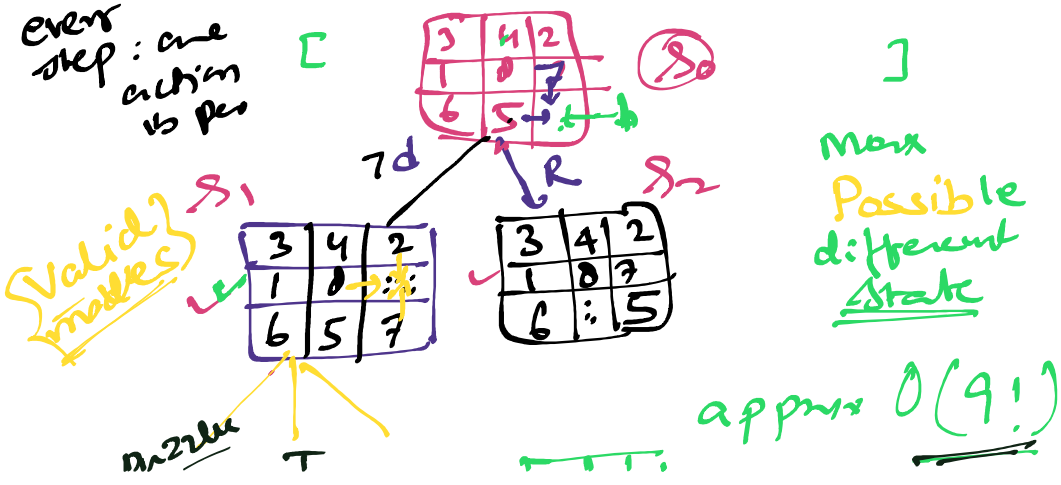
Sol? [1, 2, 3, 4, 5, 6, b, 8, 7] goal State

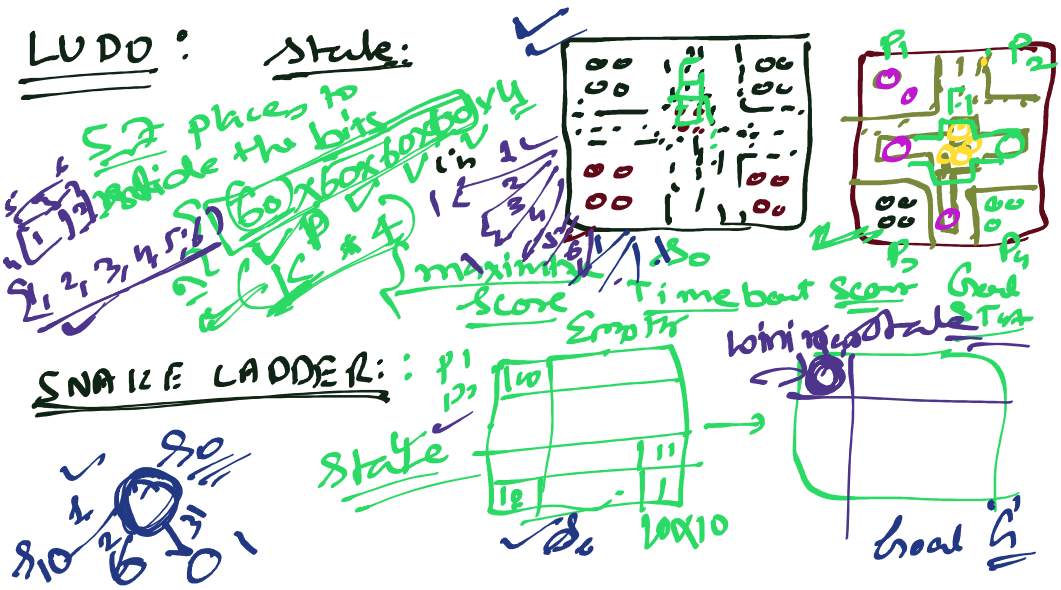
Config. State

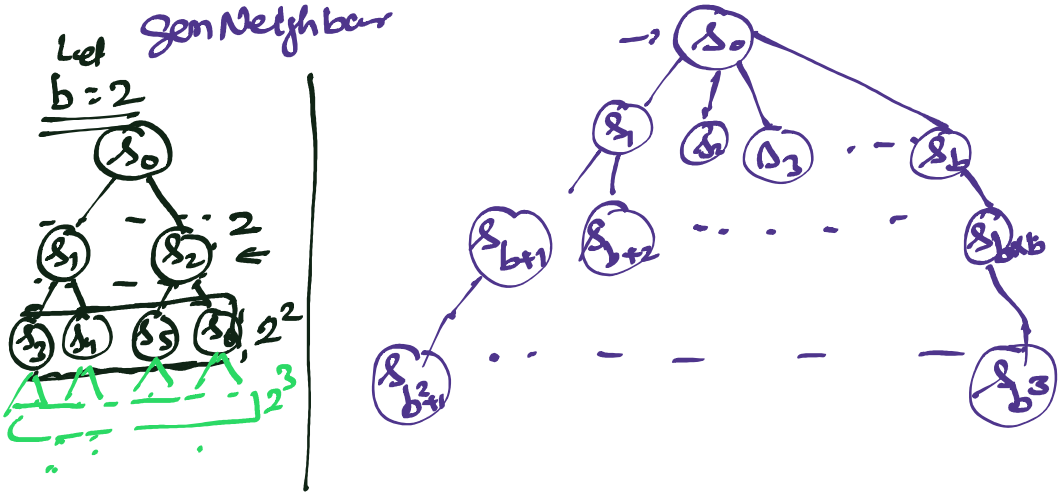
← genNeighbors(v_i)

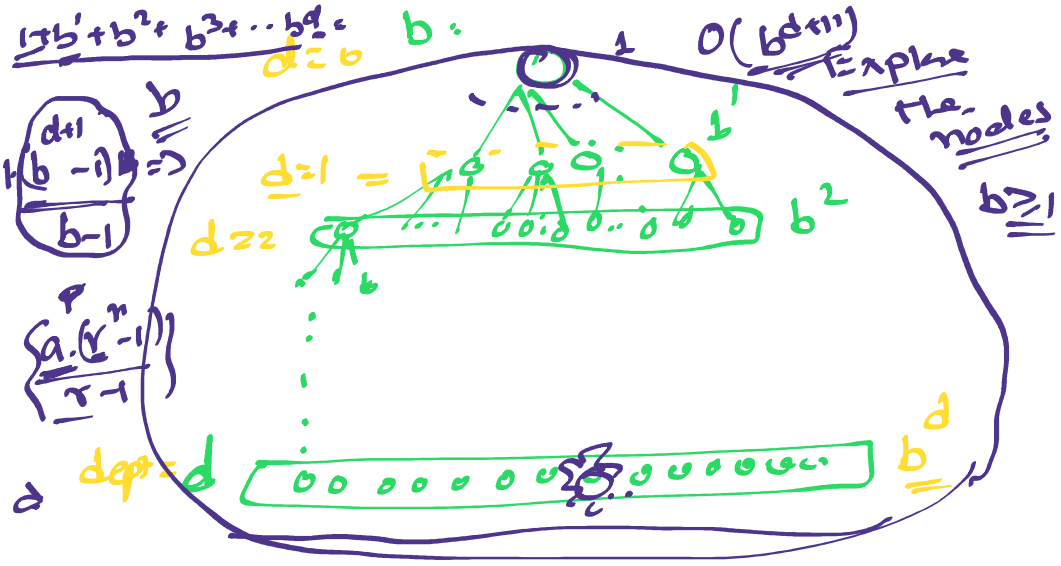
{

}

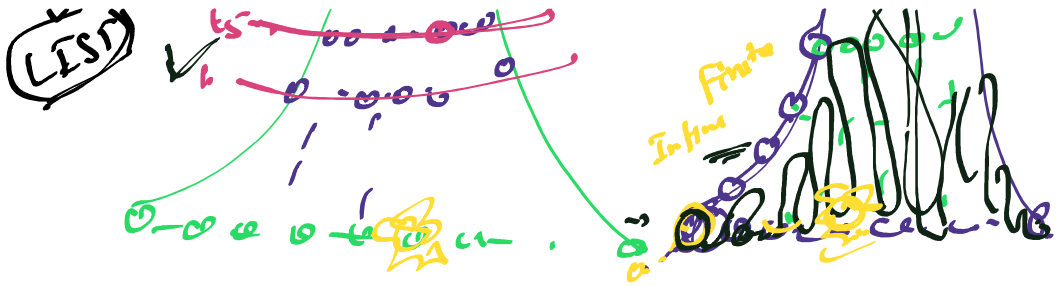



LUDO : Style:
SNAKE LADDER:
State
maximum score
Timeout
Goal


Let
b = 2
Gen Neighbour
s_0
s_1
s_2
s_3
s_4
s_5
s_6
2
2^2
2^3
s_{b+1}
s_{b+2}
s_{b^2+1}
s_{b^3}


1+b+b²+b³+..bᵈ
d=0
b
1
O(bᵈ⁺¹)
Explore the nodes
b≥1
(d+1)(b-1)/(b-1)
d=1
b¹
d=2
b²
a(rⁿ-1)/(r-1)
depth=d
bᵈ
d


Search:
Breadth First Search (BFS)
Depth First Search (DFS)
Queue
BFS
goal()
Stack
DFS
goal



→ BFS ✓ & DFS ✓

Search Zoning { Problem
State-space
5 points

Tictac-Toe : { Code & Algo
Steps will be shared by Group }

[Graph : Infinite depth But [Goal exist at finite depth d']