Artificial Intelligence

Dt:- 06/09/23

→ AI contains multiple techniques like searching, planning, learning

Deeplearning - Artificial Neural Network.

History of AI:

1943 - Evaluation of Artificial Neuron

1950 - Turing Machine

1956 - Dartmouth conference (organizer - Thon Mc Cathy)

1966 - ELIZA - first boatchat

1972 - first intelligent Robot - WABOT

1974 to 1980 - first winter of AI

1980 - Development of Expert System.

1987 to 1993 - Second winter of AI

1997 - IBM Deep ~~group~~ Blue. (chess playing org~~de~~

2002 - people started developing robots in homes

AI in home

→ Roomba - vaccum cleaner

2006 to 2009 - lots of work happened in Artificial Network.

Reboot the Artificial Neural Network

2009 - First ~~safe~~ self driving car came

* Google's first self driving car.

2011 - AI from IBM Watson (quiz show)

2011 to 2014 - Developing of smart phones assistants

Siri, Cortona, Alexa, googlenow

Apple — Microsoft — Amazon

2014 - ~~Generating~~ Generative Adversarial Network (GAN)

- Image generating
- stable Diffusion
- DallE

2016 - Google Alpha Go (Game Go)

defeat the champion of Go game.

2019 - Transformer - NLP

BOOK :-

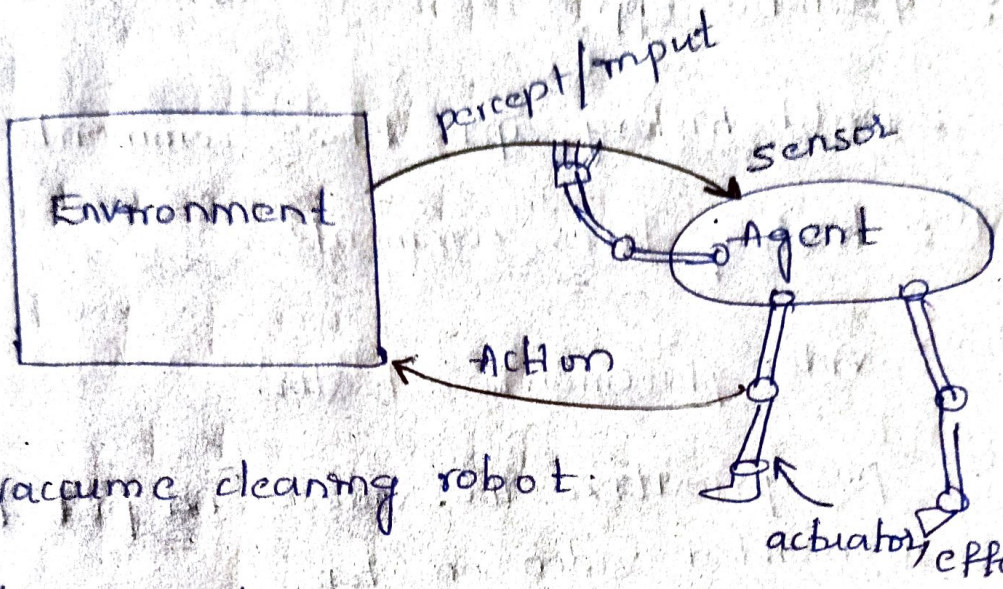
A.I - A Modern Approach by Russel & Norvig

History of AI

Dt:- 08/09/2023

Intelligent Agent:

Agent:



Eg:- vacuume cleanmg robot:

Eg:- Robotic Agent

- vaccume cleanmg robot Environment: rooms, dirt

Sensors - dirt, location sensor.

percept - clean or not, location A or B.

Actuators: wheel, succesion pipe (Actions) move left, right, suck the dirt.

Eg:- Car drivmg agent

Sensors - camera, gps, auditory sensor

Percept - visual and auditory input, location

Actuators - steermg, accelerabon, breat, start, stop,

actions - accelerate, apply breat etc--

Env - roads, traffic, other cars

* Software agent api = application programming interface

- Youtube video recomending by agent.

Env : Youtube app.

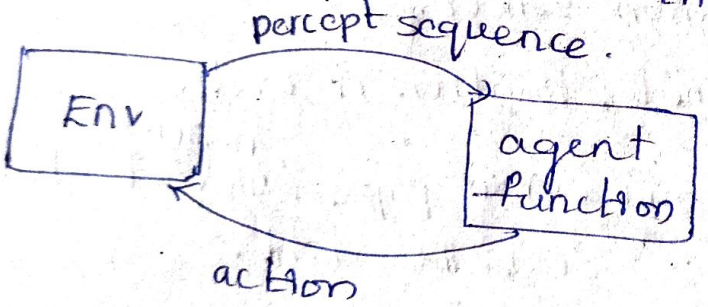
Sensors : watch history. api, likes, comments api

Percept : history, likes, comments.

Actuator : video placement actuator.

Actions: prioritize the videos and put the recomended video on top.

→ **Agent function:** mapping b/w percept sequence and the action.



Eg- vaccum cleaning agent.

percept sequences (Agent function)	action
[A, clean]	right
[A, dirty]	clean
[B, clean]	left
[B, dirty]	clean

* **Rational Agent** : based on outcome of your action.

- does the right thing

P - Performance measure

E - agent's prior knowledge of Environment

A - Actions

S - Agent's percept sequence.

Eg:- **case-1**: Performance Measure: +1 point for cleaning the room.

case-2: performance measure: +1 point for cleaning a room
-1 point for movement.

Ex:-

[A, clean] - do nothing

[A, clean], [A, clean] - do nothing

[A, clean], [A, clean], [A, dirty] - clean

[A, clean], [A, dirty], [A, clean] - right

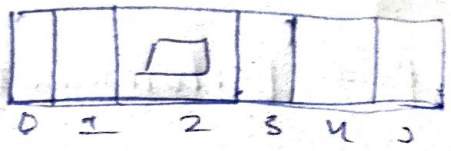
└ rational

* Properties of Environment:

① fully observable vs Partially observable

↓

you know about the information



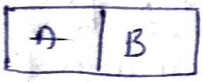
→ Global dirt sensor (fully observable)

→ Local dirt sensor (Partially observable)

② Discrete Vs Continuous:

↓ count the no. of possible states.

↓ Eg: car driving



④ - clean, clean
clean, dirty
dirty, clean
dirty, dirty

Eg: vaccume cleaning in two rooms

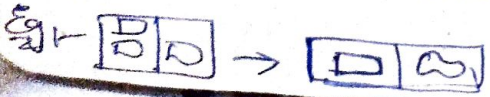
③ Episodic Vs Sequential:

↓ Eg: car assembly line

↓ Eg: car driving

④ Deterministic Vs stochastic:

↓ next state fully depend on current state

Eg: 

↓ next state is partially depend on current state

Eg: betting on the next game

⑤ Single agent Vs Multi Agent

↓ ↓

Eg:- Sudoku Eg:- chess game

⑥ static Vs Dynamic :

↓ ↓

Env not changing environment keep changing.

Eg:- Cross word puzzle Eg:- car driving

⑦ Known Vs Unknown :

↓ ↓

Agent knows the may not known

environment will change

for an action.

Dt: 11/09/2023

* Structure of an agent :

→ Agent Program

- simple reflex agent
- Model based agent
- Goal based agent
- Utility based agent
- Learning agent.

Eg:-

① Thermostat agent to control room temperature

② Simple vaccume cleaning agent

③ Autonomous delivery : drone agent

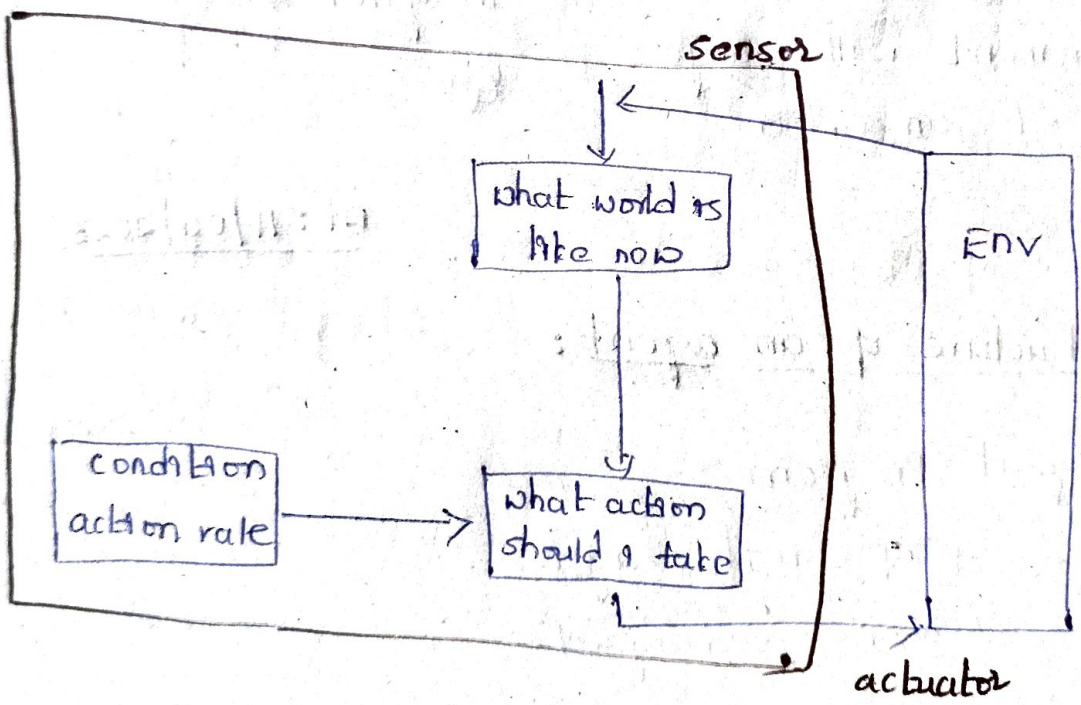
④ Autonomous delivery drone that aims to minimize battery usage.

⑤ Teaching agent.

- Simple reflex agent:

Ex: Thermostat agent to control room temperature

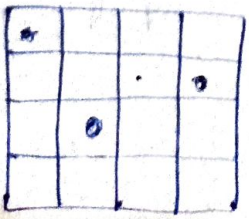
- select action base on current percept and ignore previous percept history.



- Model based agent.

It requires representation of the world = Model

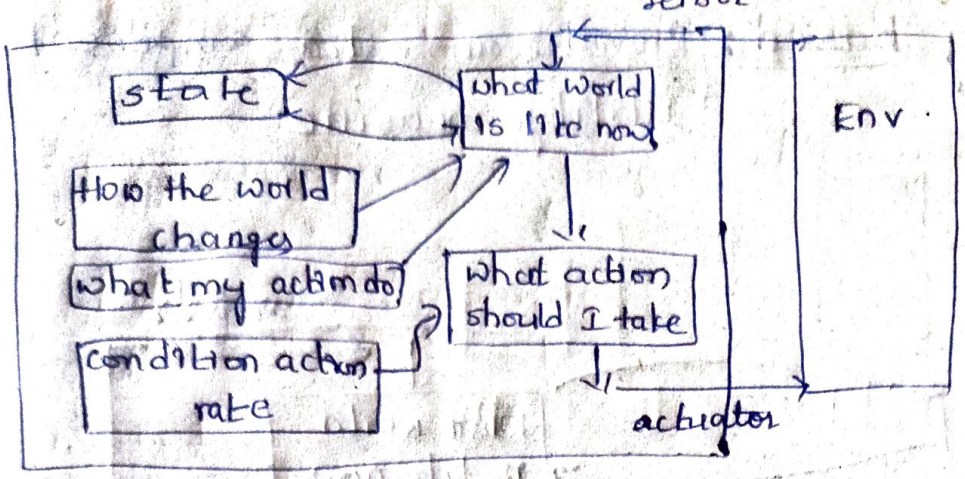
Eg: 4-Queens



Model

→ 2D array

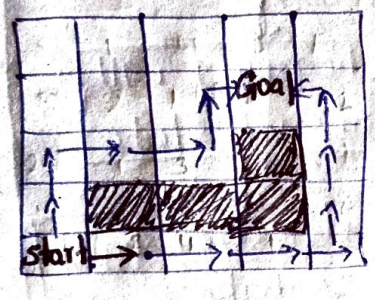
→ Graph



Goal based agent:

Ex: Autonomous delivery drone agent.

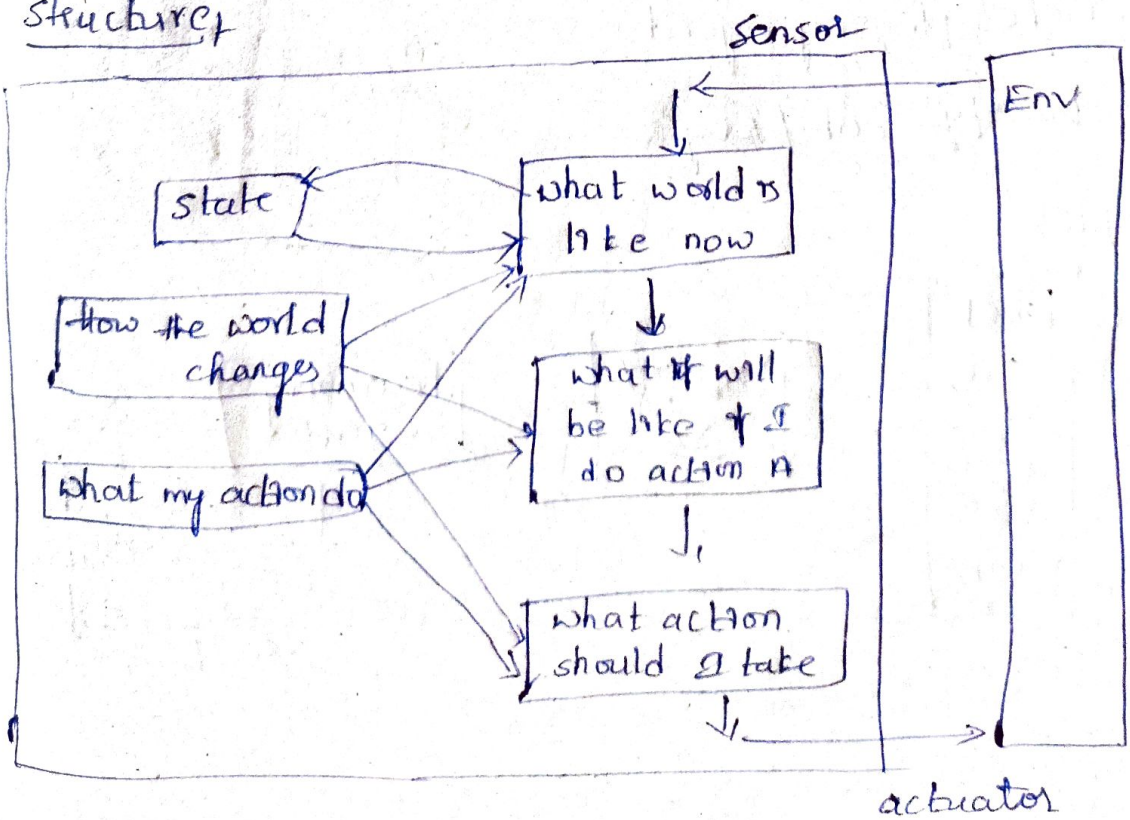
Eg:



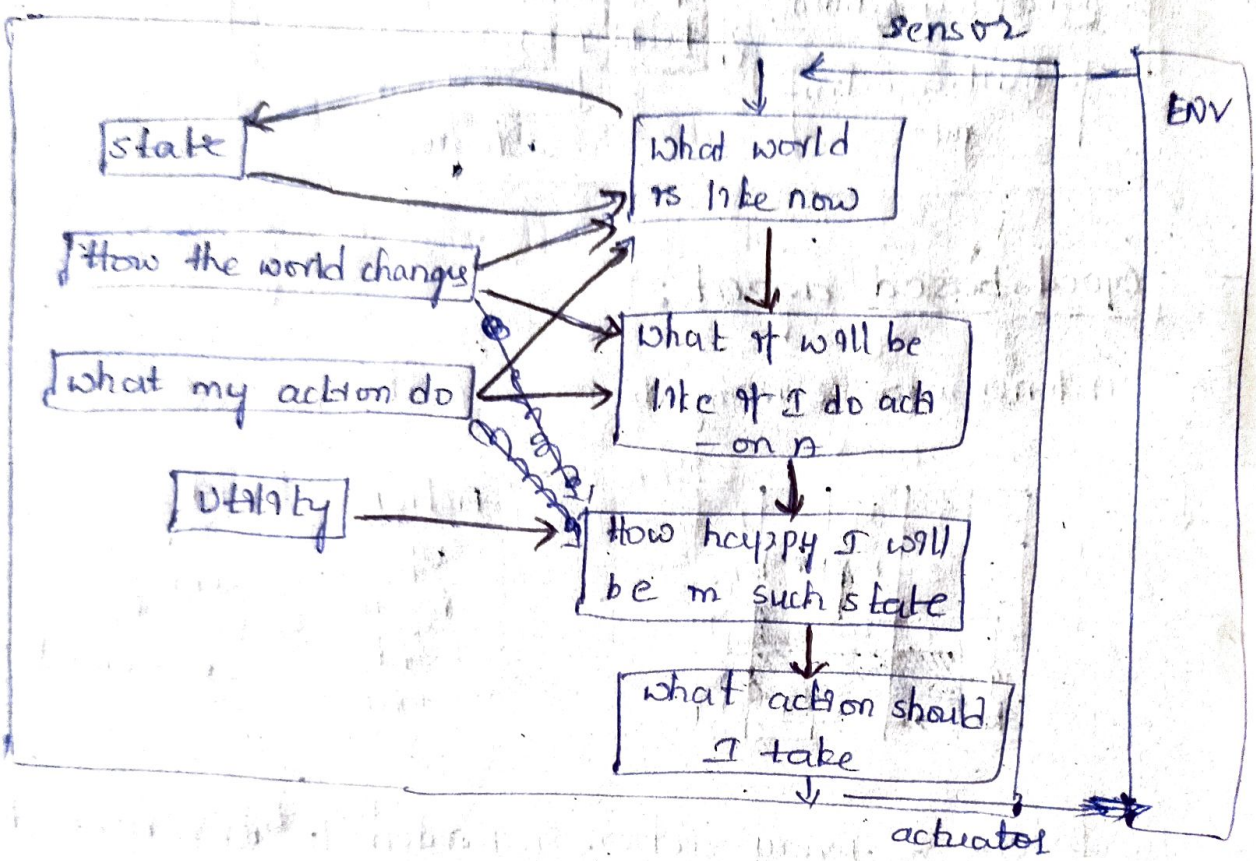
Action
up
down
Left
Right.

Structure: Every action intended to minimize the distance from the goal.

Structure:



- **Utility based:** Provide an extra component of utility measure.



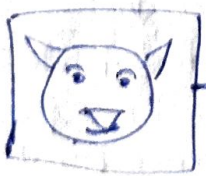
- **Learning Agent:**

- cricket playing agent

eg:- classify cat / dog

Examples

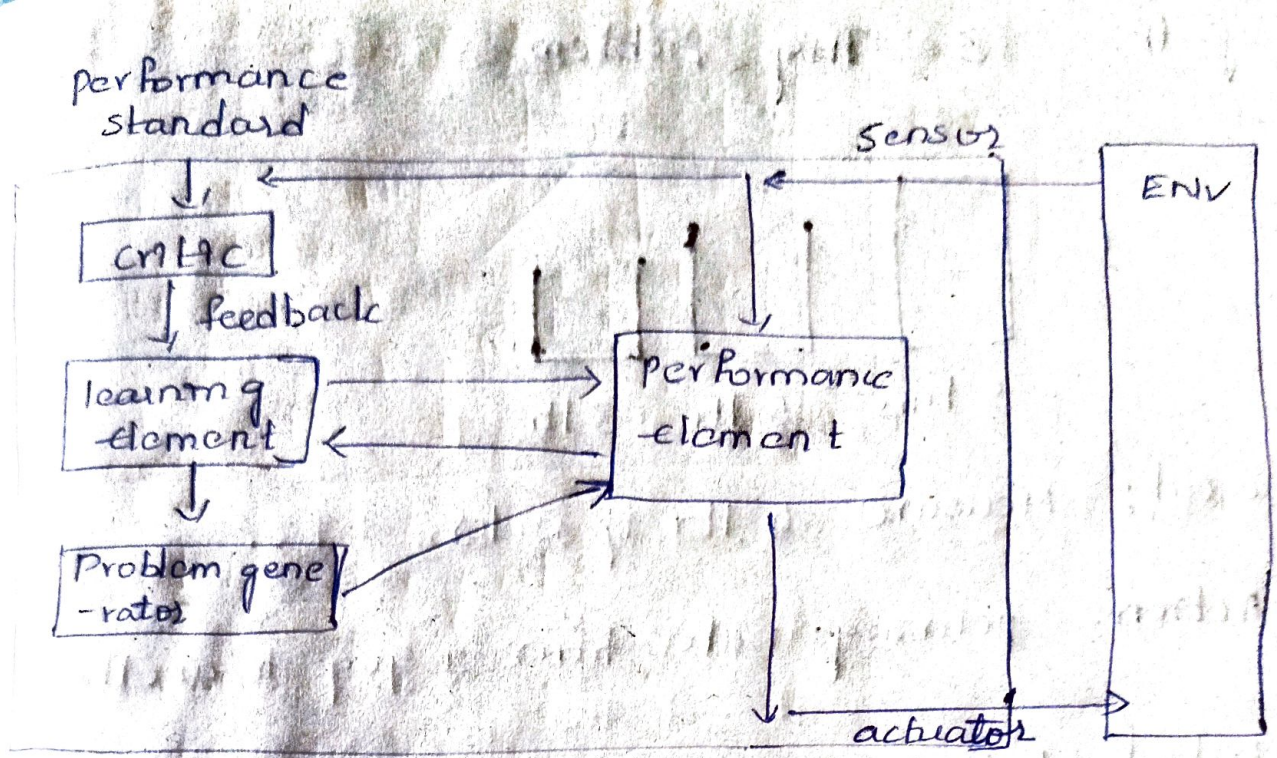
Image	label
	cat
	dog
	cat
	dog



learning agent → Dog

Critic

feedback



Dt: 13/09/2023

* Problem Solving by Searching:

- Goal based agent can be implemented using searching.
- Task is to find sequence of actions take from initial state to goal state.
- Easy problem/ Environment / world
  - Discrete
  - static
  - single agent
  - known
  - Deterministic
  - Action will never fail.

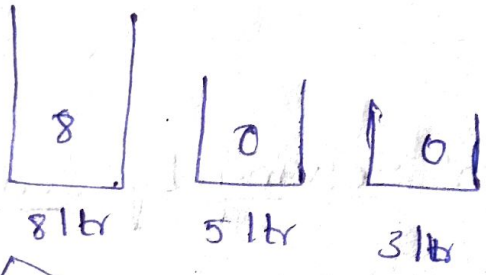
Eg-01: 3- Jug Problem

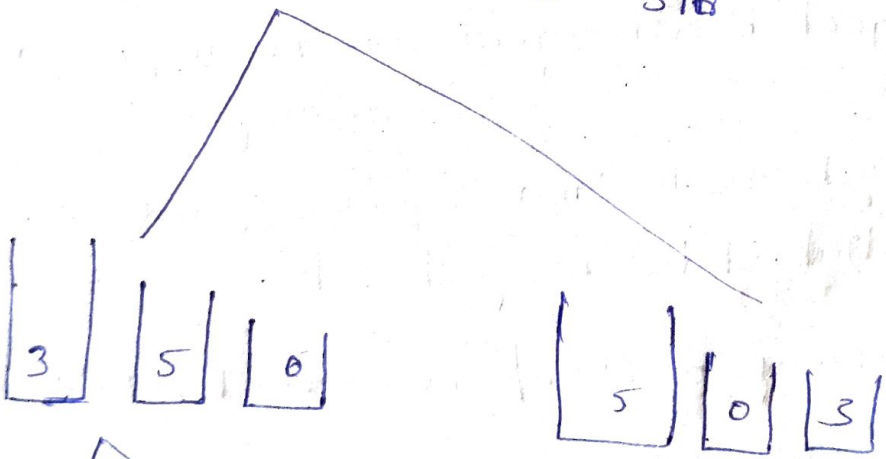
8 ltrs 5 ltr 3 ltr

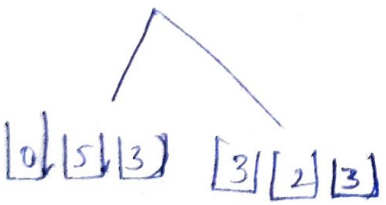
Goal: Measure 4 ltr of water.

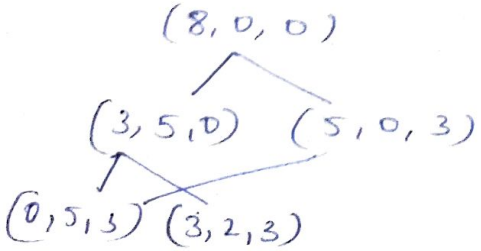
Action: pouring water from 1 jug to another

Initial state:









state space graph.

(8, 0, 0) = Initial

|

(3, 5, 0)

|

(3, 2, 3)

|

(6, 2, 0)

|

(6, 0, 2)

|

(1, 5, 2)

|

(1, 4, 3) = goal.

another jug

Eg-② : 8 Puzzle Problem

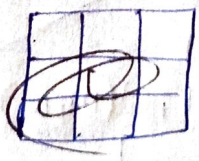
6		2
1	8	4
7	3	5

Initial state

goal = state

1	2	3
8		4
7	6	5

Actions : Move Left, Right, Up, Down.



6		2
1	8	4
7	3	5

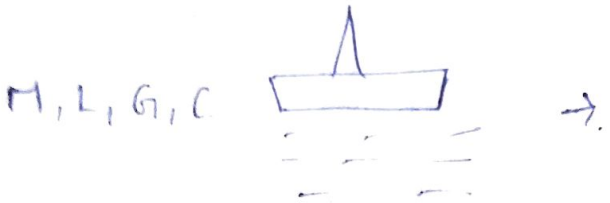
L | D | R

	6	2
1	8	4
7	3	5

6	8	2
1		4
7	3	5

6	2	
1	8	4
7	3	5

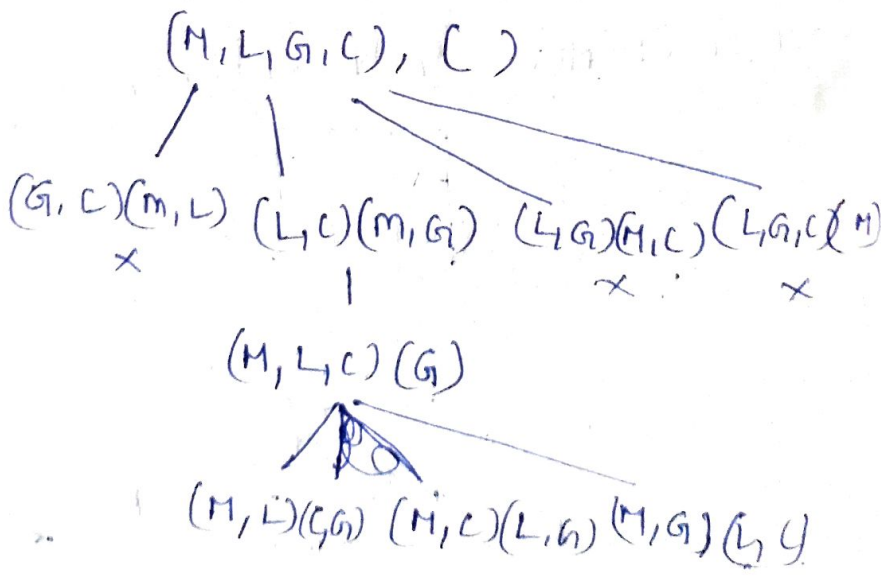
Eg-③: Man, Lion, Goat, Cabbage.



- only man can drive the boat
- At time only one item he can take.
- If man is not present
  L will eat the G
  G " " " C

~~M~~LG~~C~~ ⟶ ~~M~~G
~~M~~LG ⟶ ~~M~~C~~G~~
~~M~~LG ⟶ ~~M~~LC
~~M~~G ⟶ MGLC

Initial state:



→ Travelling Sales person

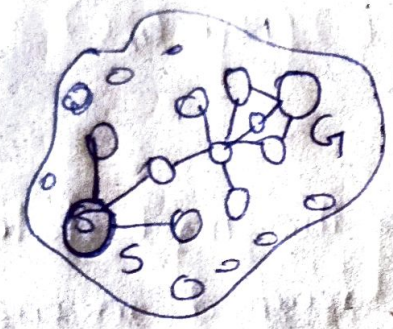
→ N-queens

→ Towers of Hanoi

→ Maze.

→ **State space:**

→ state space is implicit.



- generate
- select next state from available candidate states
- Goal test.

⇒ **Domain Independent algorithm:**

Two function

MoreGen(n)

- take a state input and return set of states that are reachable in step from input state.

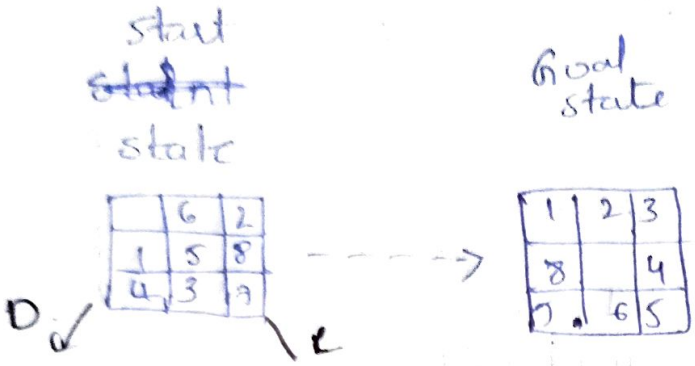
Goal test(n)

- return true if input state is goal state otherwise return future.

- One set to maintain state of the node
  - open set.

dt: 22/09/23

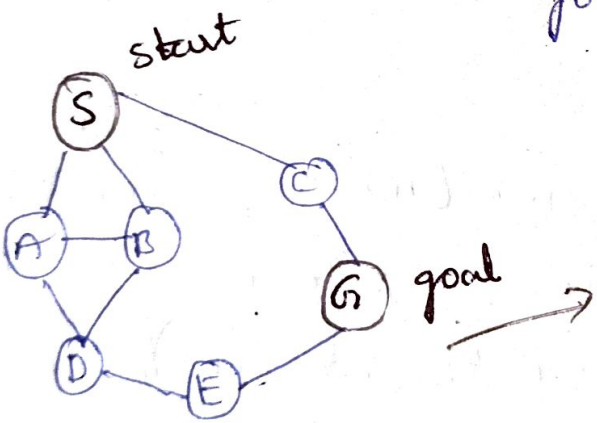
→ State Space Search:



(M, L, G, C) () ---> ( ) (M, L, G, C)

→ **MoveGen(N):** Returns the next states that can be reached in one step.

→ **Goal ~~Gen~~ Test(N):** Return True if N is a goal node.

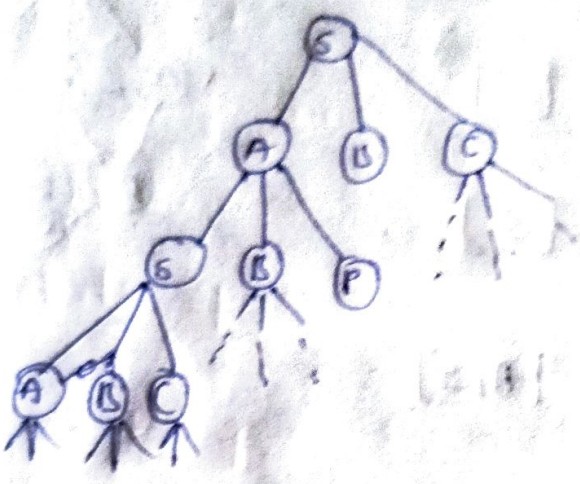


state space.

Nodes	MoveGen(N)	Goal Test(N)
S →	(A, B, C)	F
A →	(S, B, D)	F
B →	(S, A, D)	F
C →	(S, G)	F
D →	(A, B, E)	F
E →	(D, G)	F
G	(C, E)	T

Redundancy Not

State Space Search Tree:



→ Algorithms for searching: 2 types

1) Blind Search / Uninformed Search.
   - DFS
   - BFS
   - DFID

2) Heuristic search / Informed search
   - Best first search
   - A* search.

① Blind search:

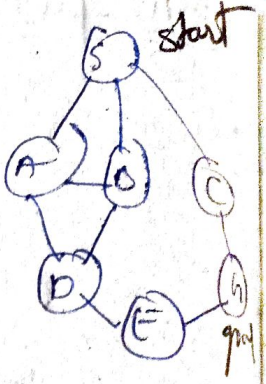
① Simple search:

- We use Two lists to maintain the status of the nodes.

(a) OPEN - List of nodes that generated but not inspected

(b) CLOSED - nodes already inspected / tested

open	closed
[S]	[ ]
[A, B, C]	[S]
[D, B, C]	[A, S]
[E, B, C]	[D, A, S]
[G, B, C]	[E, D, A, S]



Goal test(a) = T.

return

Algorithm:

OPEN ← [S]

CLOSED ← [ ]

while (OPEN is not empty)

do pick a node N from OPEN

OPEN ← OPEN - N

CLOSED ← CLOSED ~~U~~ U N

if (Goal Test(N) = True)

then return N

else

children = MoveGen(N)

OPEN ← OPEN ∪ (children − CLOSED)

return FAILURE

(NODE, parent) – To find the path from start → goal

Simple search with path information:

OPEN	CLOSED
[(S, NIL)]	[ ]
[(A,S), (B,S), (C,S)]	[(S, NIL)]
[(D,A), (B,S), (C,S)]	[(A,S), (S,NIL)]
[(E,D), (B,S), (C,S)]	[(D,A), (A,S), (S, NIL)]
[(G,E), (B,S), (C,S)]	[(E,D), (D,A), (A,S), (S,NIL)]
	[(G,E), (E,D), (D,A), (A,S), (S,NIL)]

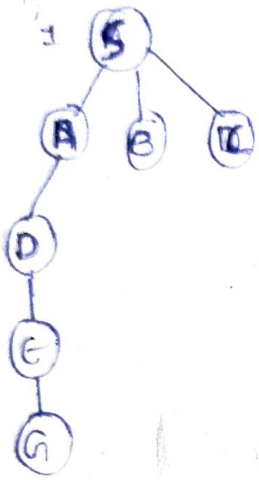
G-E-D-A-S-NIL

construct

Simple search with Path Information:

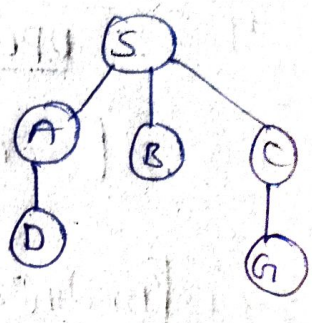
DFS & BFS :-

DFS :-



OPEN	CLOSED
[S]	[ ]
[ABC]	[S]
[DBC]	[AS]
[EBC]	[DAS]
[GBC]	[EDAS]
EMPTY ←	[GEDAS] ↑ goal

BFS :-



OPEN	CLOSED
[S]	[ ]
[ABC]	[S]
[BCD]	[AS]
[CD]	[BAS]
[DG]	[CBAS]
[G]	[DCBAS]
[ ] ↓ EMPTY	[GDCBAS] ↑ goal

→ 0 | 0 0 0 0

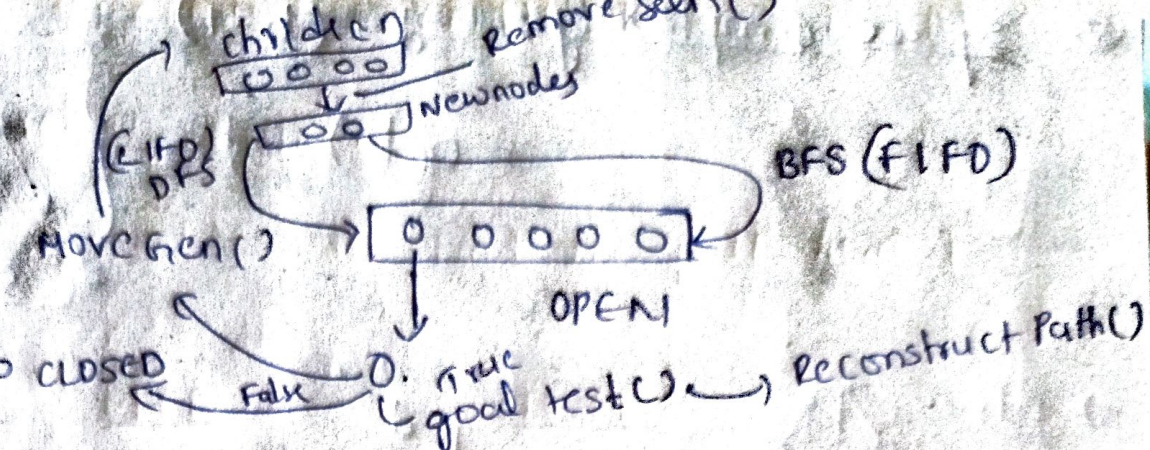
OPEN

We pick from begin and we add to closed.

CLOSED.

→ head [2, 3, 4, 5]
= 2. (first element)

→ tail [2, 3, 4, 5]
= 3, 4, 5 (except first)



APPEND ⇒ [2,3] : [4,5] = [2,3,4,5]

← assignment

= comparison.

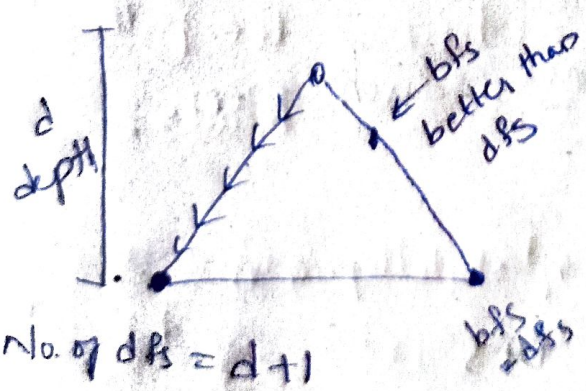
Dt:- 25/09/23

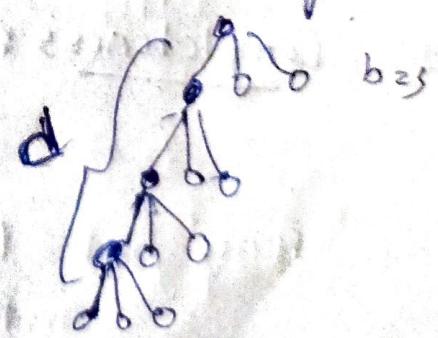
→ **DFS & BFS Comparision:**

	DFS	BFS
1)	Time $= O(b^d)$	$O(b^d)$
2)	Space - Linear	Exponential
3)	Quality of solution (shortest path)	
4)	Completeness (Remarks of goal exists)	

→ Time Complexity: No. of nodes ~~in~~ inspected (closed)

b = branching factor



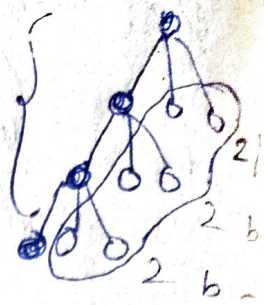


No. of dfs = d+1

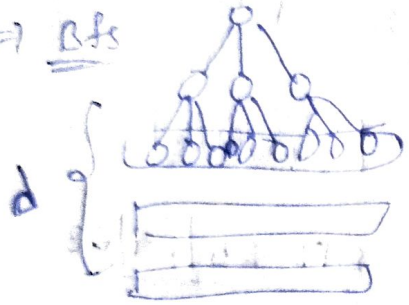
No. of bfs = $b^{d-1}+1$

② Space Complexity: $d * (b-1)$

$= O(d)$ dfs



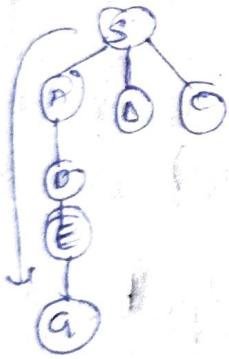
⇒ Bfs



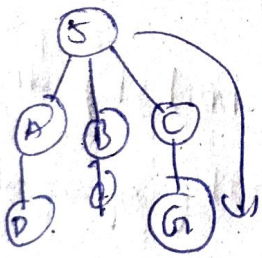
$\Rightarrow b^d$

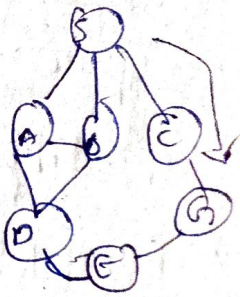
③ shortest path:- DFS — Not guranted | BFS — Always return the shortest path.

DFS



BFS





④ Completeness: Infinite search space.

→ May not terminate if state space is infinite — DFS

→ Always terminate if goal exists — BFS

Depth first Iterative Deepening (DFID)

1) Time Complexity:

$b * d + b^2 * d-1 + \ldots + b^{d-1} * 2 + b^d * 1 = O(b^d)$

2) Space Complexity: Linear

3) shortest path: Yes

4) Completeness: Yes

Algorithm: depth information for each node need to be maintained.

(node, parent, depth)

→ Depth Bound DFS (DBDFS)

→ DFS [depthBound]

→ DFID → For depth = 1 to n

DBDFS (depth)

DFDFS: depth Bound = 2

OPEN	CLOSED
[(S, nil, 0)]	[ ]
[(A, S, 1), (B, S, 1), (C, S, 1)]	[(S, nil, 0)]
[(D, A, 2); (B, S, 1), (E, S, 1)]	[(A, S, 1), (S, nil, 0)]

OPEN

[(B, S, 1), (C, S, 1)]

[(C, S, 1)]

[(G, C, 2)]

goal ← stop

CLOSE

[(D, A, 2), (A, S, 1), (S, nil, 0)]

[(B, S, 1), (D, A, 2), (A, S, 1), (S, nil, 0)]

[(C, S, 1), (B, S, 1), (D, A, 2), (A, S, 1), (S, nil, 0)]

path :- nil - S - C - G

Dt:- 27/09/23

* **DFID Example:**

MoveGen

S → DCBA
A → SBJE
B → SFA
C → SDHG
D → SIC
E → AJK
F → BKJ
G → CL
H → CIML
I → DH
J → AFE
K → EF
L → GHM
M → LH

a. Show the order in which DFID applies Goal Test to nodes.

b. What is the path it finds?

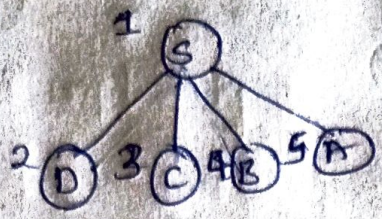
c. Show the order in which BFS applies Goal Test to nodes.

D. How many nodes BFS will open.

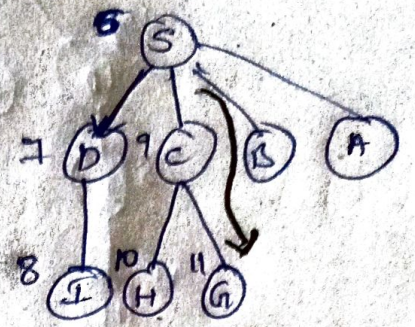
e. D.FS

f. No. of nodes opened in DFS

depthBound = 1



depthBound = 2



a. Order

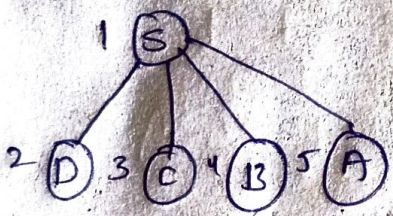
S D C B A

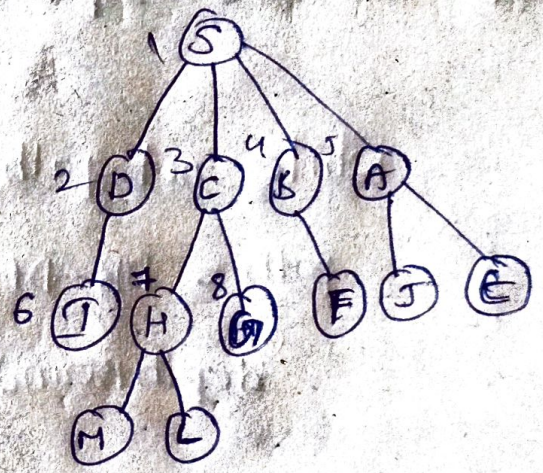
S D I C H G

b. Path

S → C → G

c.





order:

S D C B A I H G

path:

S → C → G.

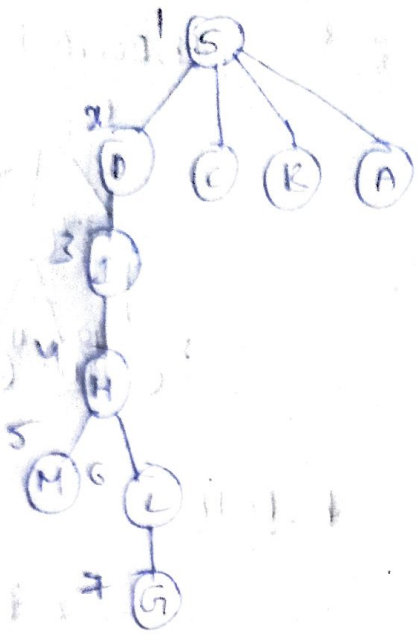
d. No. of nodes opened = 13

DFS

order → S - D - I - H - M - L - G

Path ⇒ S - D - I - H - L - G

No. of nodes opened = 10



Algo: DB-DFS (depth Bound)

```
while not path:
    path = DB-DFS (dept Bound)
    depthBound += 1
```

Algo: DFID(G)

```
count ← -1
path ← empty list
depthBound ← 1
repeat
    previous count ← count
    (count, path) ← DB-DFS
    depth Baend ← depth Bound+1
until (path is not empty) or (previousCount = count)
return path.
```

```
DB.DFS(s, depthBound)
    count ← 0
    OPEN ← [(s, null, 0)] : []        → appending
    CLOSED ← []
    while OPEN is not empty
        nodePair ← head OPEN
        (N, _, depth) ← nodePair
            ↳ placeholder - used when variable is not needed to be stored
        if GoalTest(N) = True
            return ReconstructPath(nodepair, CLOSED)
        else
            CLOSED ← nodepair : CLOSED
            if depth < depthBound
                children ← MoveGen(N)
                newNodes ← RemoveSeen(children, OPEN, CLOSED)
                newpairs ← MakeNodepair(newNodes, N, depth)
                OPEN ← newPairs : tail OPEN
                count ← count + length of newpairs
            else
                OPEN ← tail OPEN
    return (count, EmptyList)
```