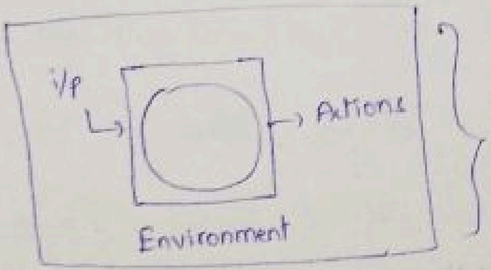
Artificial Intelligence

Date: 04-09-2023

→ In 1950's computers were discovered and started programming from that time onwards.

Intelligence:

→ Problem solving comparisons
→ Logical thinking (Rational)
→ creativity
→ Learning
→ Memory
→ Decision making
→ Communication
→ Emotional Intelligence
→ Self-awareness.



This is known as Intelligence Agent.

Based on the Environment situations and According to i/p's we Perform some tasks / Actions which is required based on condition, is known as Intelligent Agent

Intelligent Agents:

→ Two dimension
  ↳ Think
  ↳ Act

→ Two manners
  ↳ Human performance
  ↳ Rationality

machines that think like a human	machines that thinks rationally
Machines that acts like a human	machines that acts Rationally

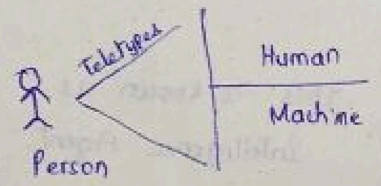
→ The above four Para digion to develop intelligence

Machines that Act like a Human

→ Tuning test

The machine / Instrument used to check the intelligence of developed machine.

→ If the developed machine works / Instruct like a human then the machine is good to develop.



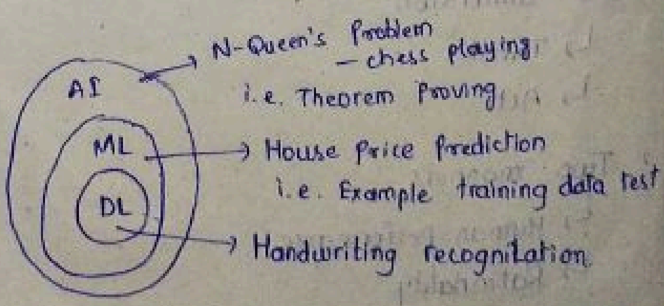
→ A person teletypes the intelligence of Human & machine

Trending technologies Nowadays

⇒ AI - Artificial Intelligence

→ ML - Machine learning

⇒ DL - Deap learning



History of AI:

1943 - Evolution of Artificial Neuron (Similar to the neuron in our body
→ neurons connected with network node (Connecting point) to process i/p & forward o/p

1950 - Turing machine (Basic model for modern computer.
↳ (Algorithms, computation machine)

1956 - Dartmouth Conference (It played a key role in developing AI)
↳ Key Person (Jhon Mc Carthy)

1966 - ELIZA - first chat Bot

1972 - First Intelligent Robot - WABOT (first human robot)

(1974 - 1980) - Winter of A.I (No fund & Not interested to develop AI)

1980 - Expert System (designed to solve complex problems)

(1987 - 1993) - Second winter of AI

1997 - IBM Deep Blue. (chess playing bot, which defeated Grand Master)

2002 - AI in home - Roomba.

(2006 - 2009) - Reboot of Neural networks

2009 - Google's first self driving car.

2011 - IBM Watson (Able to defeat a 1 person (Top player) in quiz competition)

(2011 - 2014) - Siri, Google now, Cortona.

(2014) - Generative Adeverserial Network (GAN)
↳ (Image generating AI).

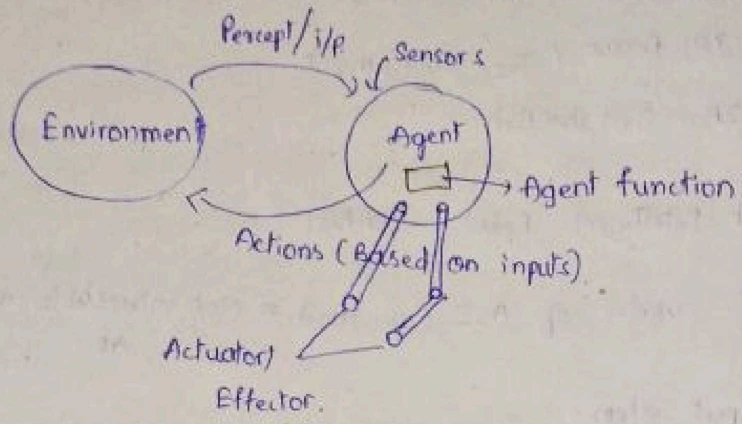
(2016) - GO Game. → Google built (AlphaGo' bot (A bot developed to defeat top player of Go Game.

2019 - Transformer for NLP (chat GPT)

Terminologies in AI

Intelligent Agent:

↳ Agent: Intelligent Agent is a program and the place where it works is known as Environment.



Agent:

↳ Sensors – Percept (through which we take input) – input.

↳ Actuation

↳ take action

Environment: Where Agent works.

Ex.

— Human Agent

— Human driving a car.

Environment –

↳ road, other car, traffic lights, traffic police, pedestrian

Agent –

Sensors – Eyes, Ears, Skin

Percept – Visual & auditory inputs.

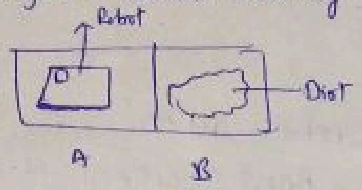
Actuator – steering wheel, break, acceldator, horn

Actions:

→ turn, accelerator, stop.

Ex-2:

Robotic Agent – Vaccum cleaning robot.



Environment

→ Two squares, dirt, no dirt

Agent –

Sensors – Dirt sensors,

Location sensors.

Percept – Location A or B.

clean or not clean.

Actuation – wheel, succession pipe

Action

↳ move left, move right, suck the dirt.

Ex-3 Sofware agent – Spam filtering agent

Environment – Internet, Email client, flow of incoming email.

Agent –

Sensors – Content, meta data (details about data), user feedback sensor

Percept – Email content (using content), metadata info, user feedback data (using above data sensors).

Actuator – Email labeling actuator, Email sorting actuator, Notification actuator.

Action: classify, filter, notify.

Ex 4: Software Agent – Youtube Video recommendor agent

Percept – watch history, video tags, categories

Action – display the recommended video at the top.

Agent
- → Robotic Agent
- → Software Agent.

Agent function: For which input, what action.

Based on Yesterday's example, Agent function is

Percept Sequence	Action
[A, clean]	right
[A, Dirt]	Suck
[B, clean]	left
[B, dirt]	Suck

Rational Agent: does the right thing

As per the above example, it is not a rational agent. When both rooms are clean, it will not turn off and Keep moving from one room to other. it is a power consumption

To make Agent a Rational Agent, we use the following Technique.

PEAS task environment: (Qualities/characteristics of rational Agent)

P – Performance measure

E → Agent's prior environment knowledge

A – Actions that can be performed

S → Percept Sequence to date.

Example

Cases: Performan

over a life ti

Case 2: +1 Poin

-1 Poi

Using Percept

↳ After comple

Task Environm

↳ the prob

Properties: (affe

① Fully obser

↳ Exn

② Discrete Vs

Exn → discre

→ continuo

③ Episodic Vs

Exn – Episodic

– Sequentic

Example

Case1: Performance measure: 1 point for each clean Square over a life time of 1000 time.

Case 2: +1 point for each clean Square } over time of 1000 time.
-1 point for each move

Using Percept Sequence

↳ After completeing the cleaning of Room A, we goto Room B.

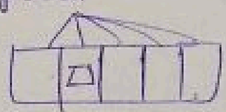
Task Environments

↳ the problem for which rational agent is the Solution.

Properties: (affects the design of the agent)

1. Fully observable vs partially observable.

   ↳ Exn Global dirt sensor vs local dirt sensor.



2. Discrete Vs Continuous.

   Exn → discrete - Vaccume cleaning robot in two rooms.
   - chess playing. (where no. of states are countable)

   → continuous - car driving

3. Episodic Vs Sequential.

   Exn - Episodic - car assembling agent.
   - Sequential - car driving agent.

4) Deterministic Vs Stochastic

→ Next state completely depends on ~~for~~ Current state

→ Stochastic → the current state completely doesn't decide the next state. (Ex:- probability of winning a cricket match)

5) Single Agent Vs multiple Agent

Exn.- cross-word (Single Agent)

- chess (multiple Agents).

6) Known Vs Unknown

Known Agent: Agent, which completely know the state of environment after an action.

Unknown Agent: It is opposite to known agent.

Structure of Agents:

→ Depends on the Agent Program, that we are using.

agent = Architecture + agent Program. (mapping from I/p to action)

↓

what kind of Sensors

— 4 types

- Simple reflex agent.
- Model based agent
- Goal based agent
- utility based agent.

→ classical agents

→ One additional agent

- learning agent.

Difference bet

① thermostat c

2) robot vacc

3) Autonomous

4) Autonomous

battery, deliver

Difference between the following agents:

1. thermostat agent to control temperature
2. robot vaccume cleaner
3. Autonomous delivery drones.
4. Autonomous delivery drones that aims to efficiently utilize battery, delivery time and maximize safety.

Structure of an Agent continuation:

Simplex reflex Agent:

↳ Simple Program.. based on inputs, it gives output.

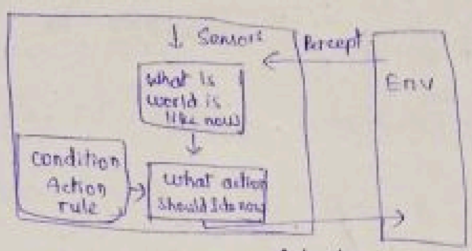
Ex: thermostat agent for controlling temperature of a room.

↳ It will not remember Previous actions & Previous inputs.

defn: Selects an action based on Current Percept.

↳ Ignores previous history.

↳ doesn't require internal representation (Ex: No. of rooms)



Structure of Simple reflex Agent

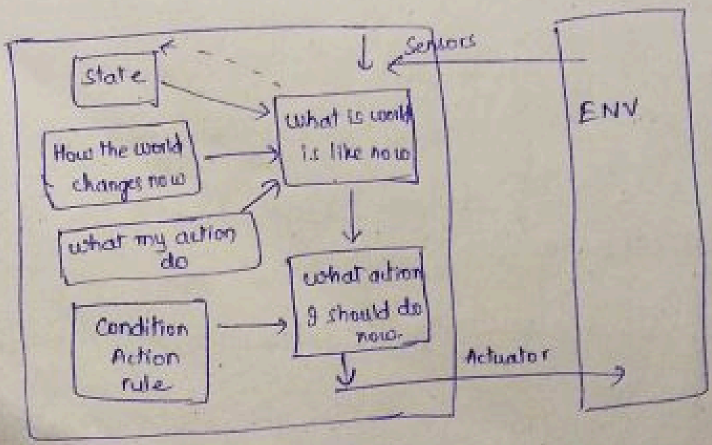
Model based Agent: → It requires previous history (Ex: N-Queens).

Ex: Simple robotic Vaccume cleaner.

↳ It requires internal representation of world (i.e. no. of rooms to clean).

Internal representation also known as

"Model of the world"



Structure of Model based Agent

Goal Based Agen

Ex: Autonomous d

↳ Every action

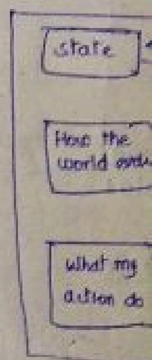
[Ex: Travelling

↳ min

→ It requires

model

Actions



Utility based agent

Ex: An autonomous

Safety.

Ex: time table d

time table d

→ utility bas

Goal Based Agent: → More specific regarding performing a particular task.

Ex: Autonomous delivery drones

↳ Every action is intended to minimize the distance from the goal.

(Ex: Travelling salesman problem)

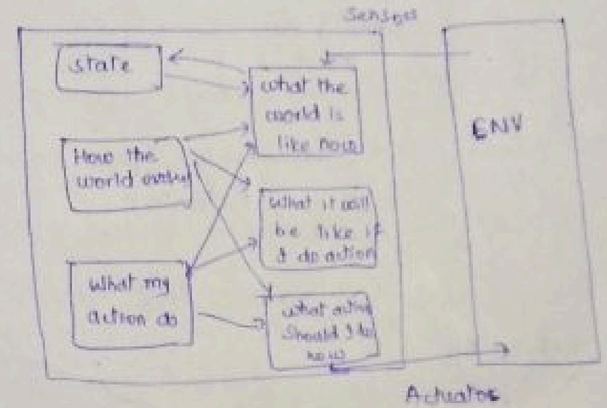
↳ minimum cost path to reach from source to destination.

→ It requires model & goal.

model – blocks

– obstacles

Actions – L, R, U, D



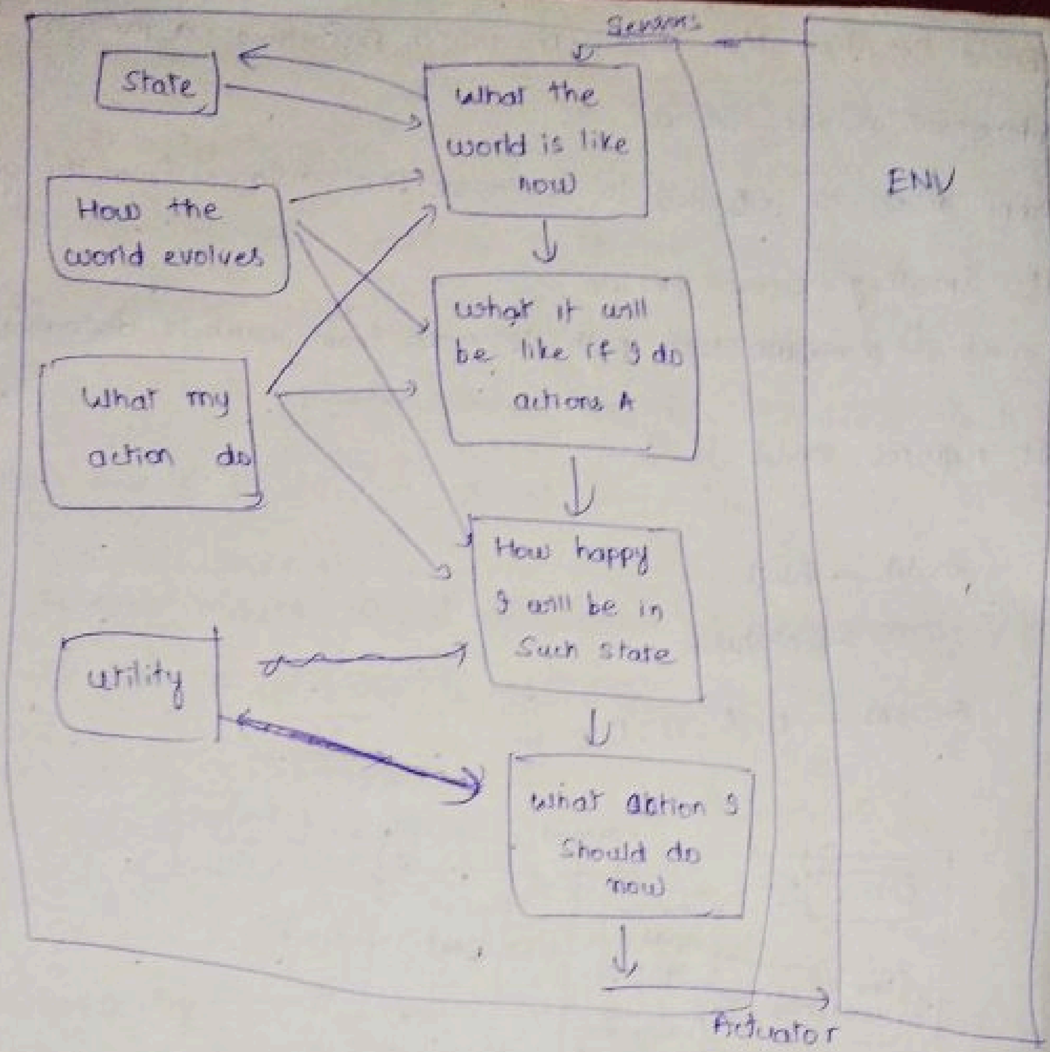
Utility based agent: → reaching the goal happily

Ex: An autonomous drone that aims to maximize energy efficiency & safety.

Ex: time table design agent – goal based

time table design agent with teacher's requirement

→ utility based agent.



Learning Agent:

Learns from ~~past~~ experience.

Exn classify cat/dog picture.

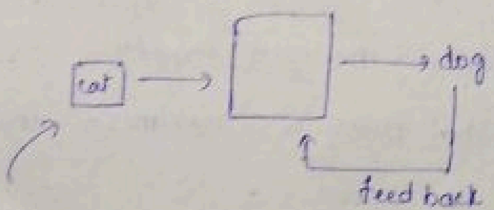
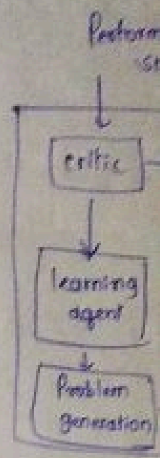


Image	label
	cat
	dog



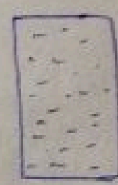
Problem Solving u

- Goal based agents

  - Find out a

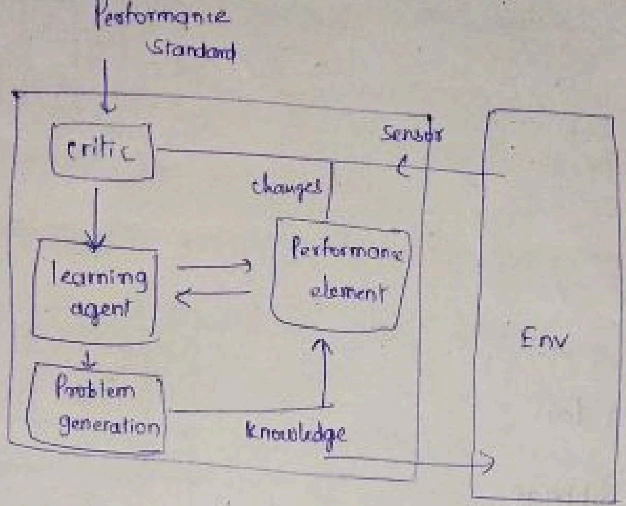
  State to goal

Exn Three jug Pr



8

Goal: measure

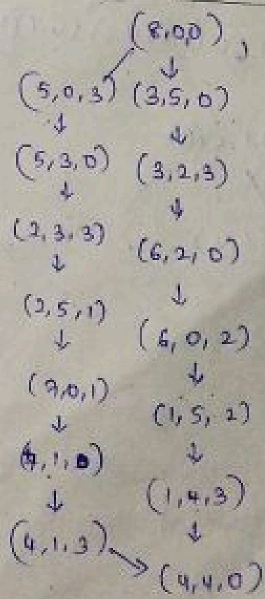


Structure of learning agent

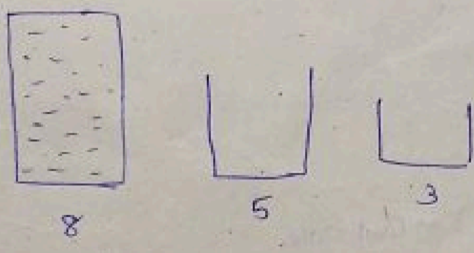
Problem Solving using Search methods:

- Goal based agents can be implemented using search methods.
  - Find out a sequence of actions that will take from initial state to goal state.

State space graph



Exn Three jug Problem.



Goal: measure of 4 litre of water.

Easy Problem / environment / world

- Discrete
- Static
- Single agent
- known
- Deterministic
- Actions do not fail

Ex-2: Man, Lion, Goat, cabbage

Restrictions:

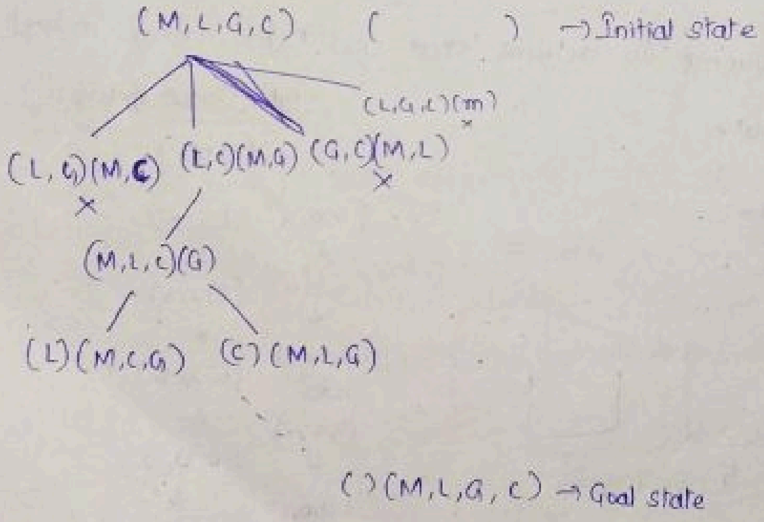
Cross river using a boat & a man can drive the boat

↳ He can take only one item at a time.

↳ If man is not present Goat eat cabbage & Lion eat goat

↳ Lion won't eat man.

State Space diagram of Ex-2

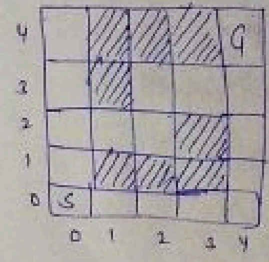


Ex-3: 8-puzzle problem

6		2
1	8	4
7	3	5

Initial state

Ex-4: Maze



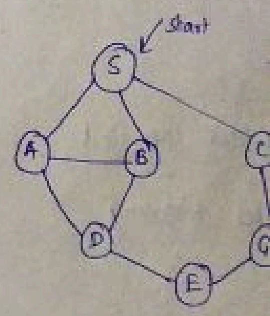
Generate & Test

(1) MoveGen – Generating

state in 1 step

(2) Goal Test – Returns tru

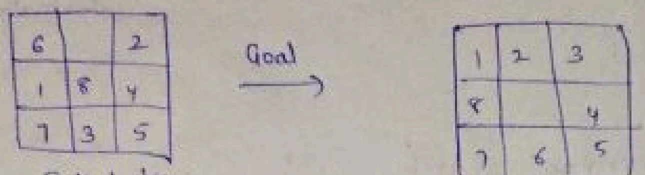
return false



State Space -

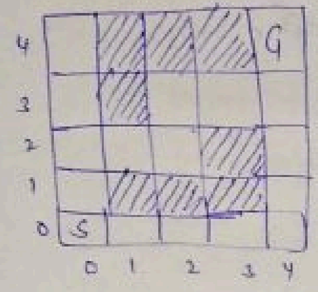
st

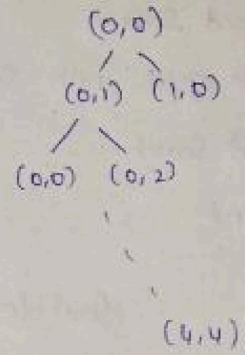
Ex-3: 8-puzzle problem



Initial state Goal state.

Ex-4: Maze

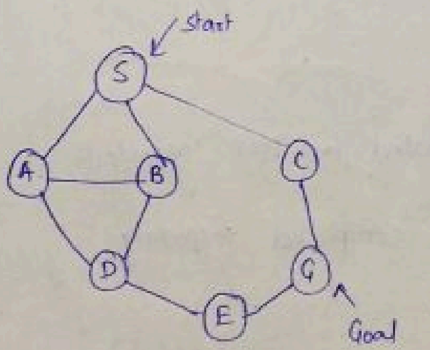




Generate & test

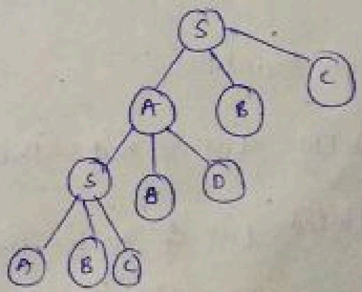
① MoveGen – Generating the next states that is reachable from current state in 1 step

② Goal Test – Returns true if the current node is goal node otherwise return false



State Space – Set of all possible states

State Space Search tree



↳ It provides all possible ways to reach the goal.

Blind Search / Uninformed Search → Searching for goal without any information

- DFS
- BFS
- Iterative Deepening Search

→ Heuristic / Informed Search

↳ Searching the goal with some information

- Best first Search
- A^* Search

x	MoveGen(x)	Goal test(x)
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

Simple Search:

- OPEN: List of nodes that are generated but not inspected.
- closed: List of nodes that we have completed inspecting.

open	close
[S]	[ ]
[A, B, C]	[S]
[B, C, D]	[A, S]
[C, D]	[B, A, S]
[G, D]	[C, B, A, S]

↓
Goal is stop

Simple Search Algorithm:

OPEN ← {S}

close ← { }

while open is not e

do pick a node n

CLOSE = CLOSE U

OPEN = OPEN - n

if Goaltest (n) =

return tru

else

children No

OPEN = OPEN U

return Failure.

→ From OPEN, we will alw

Returning Path:

Maintain (node, Parent)

open

[(S, nil)]

[(A, S), (B, S), (C, S

[(D, A), (B, S), (C, S)]

[(E, D), (B, S), (C, S)]

[(G, E), (B, S), (C, S)]

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

Simple Search Algorithm:

OPEN ← {s}

close ← { }

while open is not empty

do pick a node n from OPEN

CLOSE = CLOSE ∪ {n}

OPEN = OPEN - n

if Goaltest (n) = True

return true

else

children Nodes = MoveGen(n)

OPEN = OPEN ∪ (children Nodes - CLOSE)

return failure.

⟶ From OPEN. we will always start from beginning.

Returning Path:

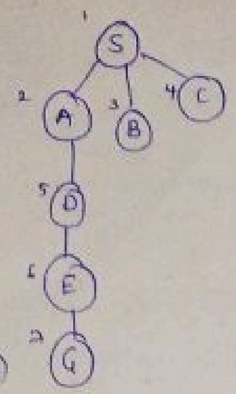
Maintain (node, Parent)

open	close
[(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)]
[(B,S), (C,S)]	[(G,E), (E,D), (D,A), (A,S), (S, nil)]

↑ Goal node

G-E-D-A-S-nil.

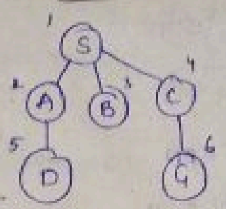
DFS (LIFO)



LIFO (stack)

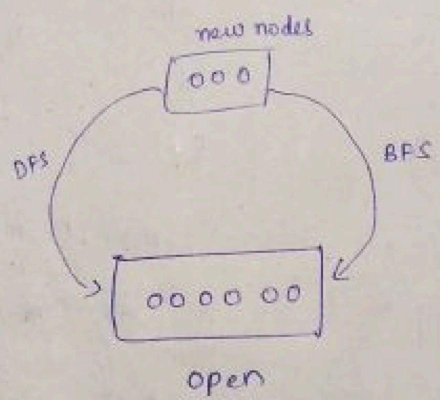
Open	close
{S}	{ }
A B C	S
D B C	A S
E B C	D A S
G B C	E D A S
	G E D A S
	↓
	Goal

BFS (FIFO)



FIFO (queue)

open	close
S	{ }
A B C	S
B C D	A S
C D	B A S
D G	C B A S
G	D C B A S
	G D C B A S



DFS Algo:-

OPEN ← (start, null)

CLOSED ← [ ]

while OPEN is not e

nodepair ← hea

(N, -) ← node

if {Goal Test(N)

return

else CLOSED

child

newN

new

OPEN ← tail OPEN : newNodes $\xleftarrow{BFS}$ OP

return emptylist

Comparison:

- Time complexity
- Space complexity
- Quality of Solⁿ
- Completeness
  - ↳ If there

Behaviour of BFS &

→ Branching factor: no. of

Branching factor

→ BFS require more Space

↓

require exponential Space [l

DFS Algo:-

```
OPEN ← (start, null) : [ ]
CLOSED ← [ ]
while OPEN is not empty
        nodepair ← head, OPEN
        (N, -) ← nodePair
        if {Goal Test(N) = True
                return Reconstruct Path ( nodePair, CLOSED)
        else CLOSED ← nodePair : CLOSED
                children ← MoveGen (N)
                newNodes ← RemoveSeen (children, OPEN, CLOSED)
                newPairs ← MakePairs (newNodes , N)
OPEN ← tail OPEN : newPairs ←BFS   OPEN ← newPairs : tail OPEN //→all nodes are appended
                                                                  at the beginning.
return emptylist
```

Comparison:

- Time complexity
- Space complexity
- Quality of sol^n (Shortest Path)
- Completeness
  - ↳ If there is goal, a complete algo will always find the goal.

Behaviour of BFS & DFS:-

→ Branching factor : no. of children for each node.
Branching factor = 3.

→ BFS require more Space than DFS.

BFS ↓ require exponential Space [i.e. 3^d]

DFS ↓ require Linear Space (i.e 2d]

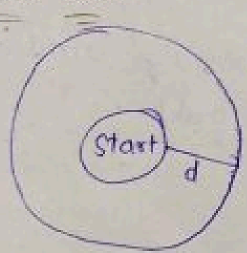
	DFS	BFS
Space complexity:	Linear Space (Size of Open) w.r.t. depth	Exponential Space w.r.to depth. ↳ $(\text{branching factor})^d$
Time Complexity:	Avg: $O(b^d)$	Avg: $O(b^d)$.
Shortest Path:	Not guaranteed	Always find the Shortest Path.
Completeness:	May not find the goal if Search Space is infinite.	Always finds the goal & terminates.

→ If Search Space is infinite, incase of DFS, if goal not found in the DFS, it keeps going & never come back. (May not terminate)

To avoid going into infinite loop, we can keep a limit for the depth for the search.

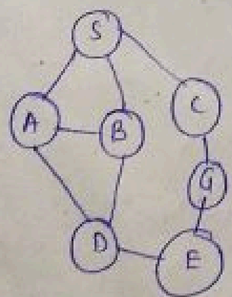
↳ This algorithm is called Depth bound DFS.

Depth bound DFS:



d → depth bound.

Ex:



→ with depth (d) = 2

OPEN

[(S, nil, 0)

[(A, S, 1), (B, S, 1),

[(D, A, 2), (B, S, 1)

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

[(C, S, 1)]

[(G, C, 2)]

[]

DBDFS Algorithm:

depthBound ← user

count ← 0

OPEN ← (S, n

CLOSED ← []

while OPEN

nodeP

(N, _,

if Go

else

OPEN	CLOSE
[(S,nil,0)]	{ }
[(A,S,1), (B,S,1), (C,S,1)]	[(S,nil,0)]
[(D,A,2), (B,S,1), (C,S,1)]	[(A,S,1), (S,nil,0)]
[(B,S,1), (C,S,1)]	[(D,A,3), (A,S,1), (S,nil,0)]
[(C,S,1)]	[(B,S,1), (D,A,3), (A,S,1), (S,nil,0)]
[(G,C,2)]	[(C,S,1), (B,S,1), (D,A,3), (A,S,1), (S,nil,0)]
[ ]	[(G,C,2), (C,S,1), (B,S,1), (D,A,3), (A,S,1), (S,nil,0)]

DBDFS Algorithm:

```
depthBound ← user input
    count ← 0
    OPEN ← (S,nil,0) [ ]
    CLOSED ← [ ]
    while OPEN is not empty
        nodePair ← head: OPEN
        (N, _ ,depth) ← node Pair
        if GoalTest (N) = True
            return ReconstructPath (nodePair, OPEN)}
        else CLOSED ← nodePair : CLOSED
            If depth < depthBound
                children ← MoveGen (N)
                newNodes ← RemoveSeen (children, OPEN, CLOSED)
```

newPairs ← MakePairs (newNodes, N, depth + 1)

OPEN ← newPairs : tail OPEN

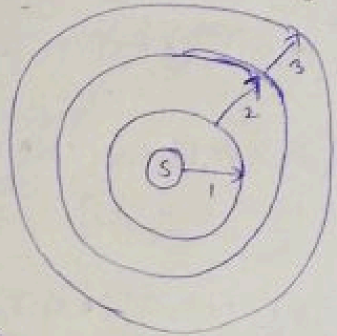
Count ← Count + length newPairs

else

OPEN ← tail OPEN

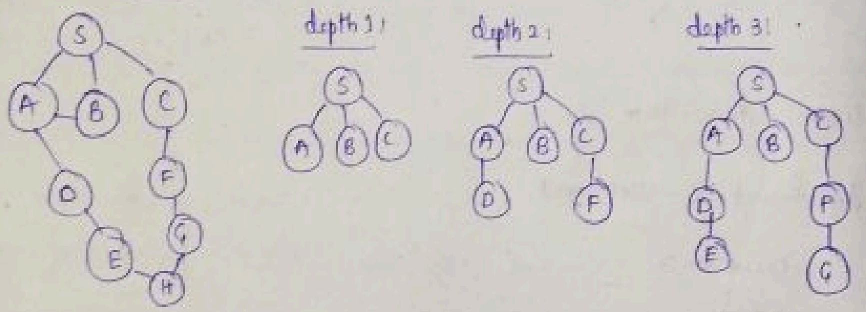
return (Count, emptyList)

Depth First Iterative Deepening: → Applying dfs & checking each layer.



→ If Goal not found in depth 1, it increases the depth to 2, repeats the same process untill goal is reached. It returns the depth.

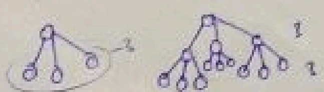
eg:



Time Complexity:

b = branching factor

d = depth at which goal is present.



$b \times d + b^2 \times (d-1) + b^3 (d-2) + \ldots$

$b^{d-2} \times (3) + b^{d-1} \times 2 + b^d \times 1$

largest value

$= O(b^d)$

Space: DFS = Linear

Shortest Path = Find

Completeness: Terminate

DFID()

Count = -1.

Path = [ ]

depthBound = 0

repeat

Previous Co

Count, Pat

depthBound

until (path

return path

DFID

Ex: MoveGen

S → DCBA

A → SBJE

B → SFA

C → SDHG

D → SIC

E → AJK

F → BKJ

G → CL

H → CIM

I → DH

J → AFE

K → EF

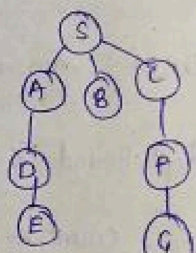
L → GHM

M → L H

king each layer.

d in depth 1, it
depth to 2, repeat
rocess untill goal is
returns the depth.

depth 3!



$b^3 (d-2) + - - -$

$(3) + b^{d-1} \times 2 + b^d \times 1$

Space: DFS = Linear

Shortest Path = Find shortest path.

Completeness: Terminate if Goal exists

DFID(S)

Count = -1.

Path = [ ]

depthBound = 0

repeat

Previous Count ← count.

(count, Path) ← DBDFS (S, depthBound)

depthBound ← depthBound + 1

until (path is not empty) or (previous count = count)

return path

you completed all node & Goal node not found

DFID

Ex:

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

S → Start node

G → Goal node

a. Show the index in which DFID applied Goal test to nodes?

b. which path it finds?

c. Show the order in which BFS will test the nodes?

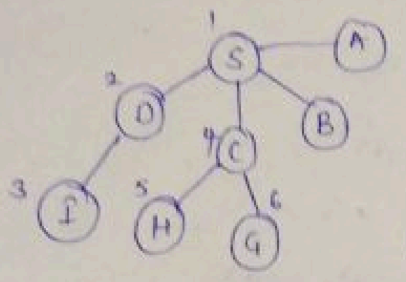
d. Total how many nodes BFS will open!

Explanation:

depth Bound = 1

Order followed in depth Bound = 1 to find goal is SDCBA

depth Bound = 2

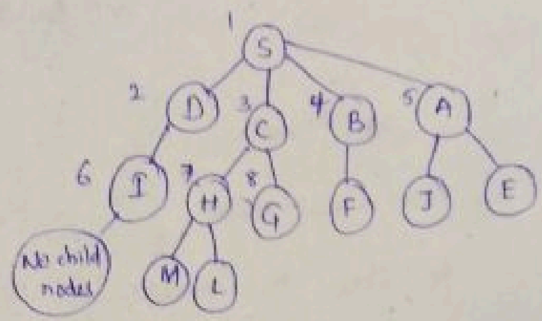


(a) Order followed to check goal node is

Order: SDICHG

(b) Path followed to find Goal node is S-C-G.

(c)



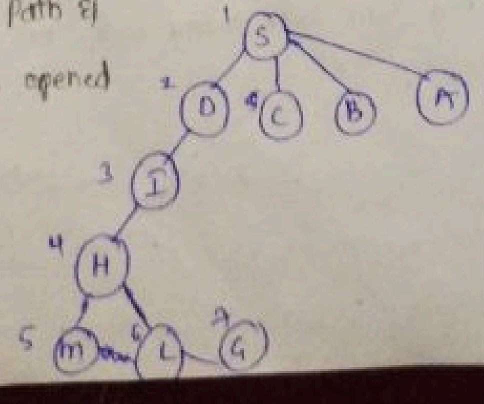
Order: SDCBAIHG

(d) No. of opened Nodes By BFS : 13

Path: S-C-G

(e) DFS Path &

No. of nodes opened by DFS

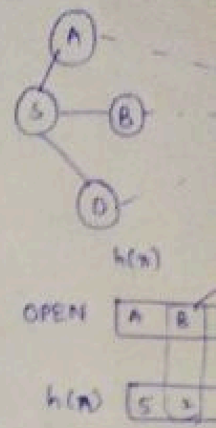


Order: SDIHMLG

no. of nodes: 10

Path: S-D-I-H-M-L-G

Heuristic Search / Informe



Sort on h(

Pick up the

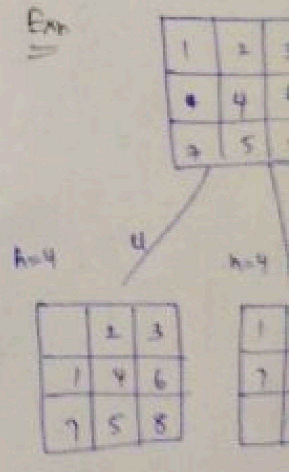
Heuristic function:

→ It estimate the

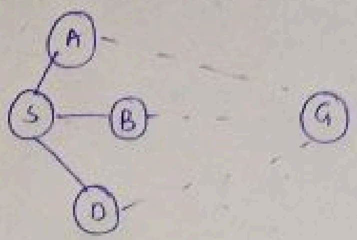
→ Used to decide

→ represented by

Exm



Heuristic Search / Informed Search:



	h(n)		
OPEN	A	B	D
h(n)	5	2	3

→ choose B, because it has the least heuristic value.

Sort on h(n)

Pick up the best node

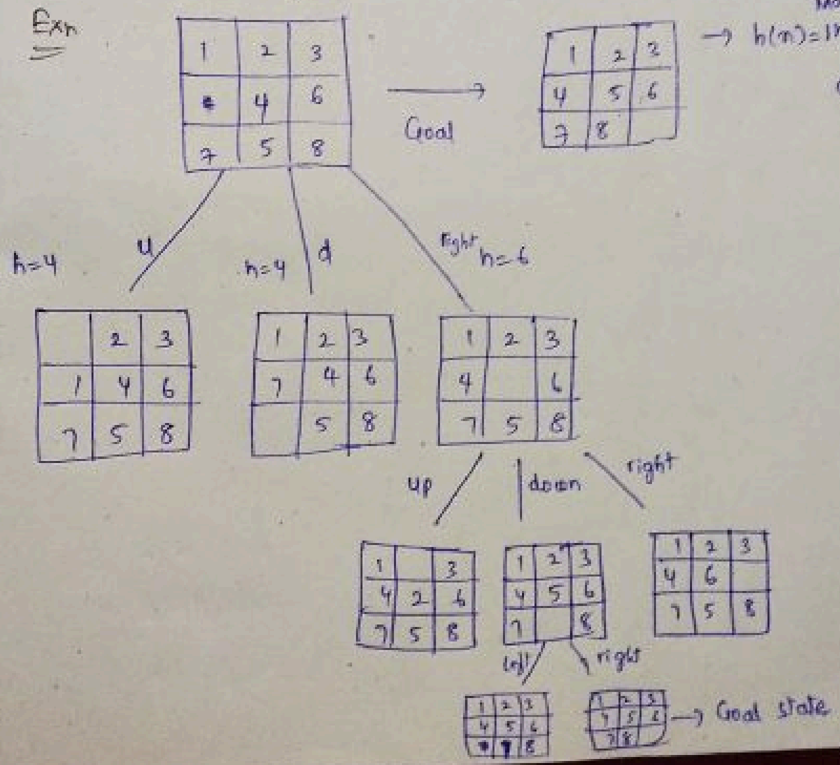
Heuristic function:

→ It estimate the distance to the goal. ~~h(n)=~~

→ Used to decide which node to pick next.

→ represented by h(n)

Exn



→ h(n) = Max no. of tiles same as goal state