Artifical Intelligence

Introduction

→ One of the fascinating and universal field in computer science which has a greatest scope in the future.

→ It has a tendency to cause a machine to work as humans.

→ At present, AI is working with a variety of sub-fields, e.g., whatever gadgets we are using nowadays are based on AI ranging from general to specific - e.g. Self driving car, playing chess, in the mobile phone, playing music, videos, everything is related to AI

what is Artificial Intelligence?

"man made" + "thinking power"

"It is branch of computer science by which we can create an intelligent machine which can behave like a human, think like human & able to make decisions"

→ We do not need to pre-program the machine. e.g. if are talking about an gadget, computer, mobile phone etc, they are programmed to perform a specific task

Evergreen
Page No.
Date : / /20

⇒ In AI, we do not need to preprogram machine to do some work.
⇒ We have to create a machine with programmed algorithm which can work with own intelligence ie it can take its own decision with its own intelligence (man-made intelligence)

Reasons of Boost in AI
1) S/w or device can be made to solve real time problems.
2) Creation of virtual assistant [SIRI, CORTANA] to help/assist humans
3) Robots development
[Helps in dangerous environment condition] (e.g. nuclear affected, land mine, diffusion of bomb, disaster prone area]
4) New job opportunities

Applications of AI :-
(1) AI in Gaming : Chess, Poker, tic-toe

Why Artificial Intelligence?

→ With the help of AI, we can create such software and devices which can solve real-world problems very easily and with accuracy such as health issues, marketing, traffic issues.

→ With the help of AI, you can create your personal virtual assistant such as Google Assistant, Siri etc.

→ With the help of AI, you can build such robots which can work in an environment where human can be at risk (or survival of humans).

→ AI opens a path for new technologies, new devices and new opportunities.

Goals of Artificial Intelligence

(1) Replicate human intelligence
(2) Solve knowledge-intensive tasks (during [illegible])
(3) An intelligent connection of perception and action.
(4) Building a machine which can perform the task requires human intelligence such as:-
→ Proving a theorem
→ Playing chess
→ Plan some surgical operation
→ Driving a car in traffic (automatic car)
→ [illegible]

Linguistic intelligence → It refers to ability to use language effectively both in written and oral forms. It involves capacity to understand and use words effectively. People with good linguistic intelligence can learn new language easily...

(5) Creating some system which can exhibit intelligent behaviour, learn new things by itself, demonstrate, explain and can advise to its users.

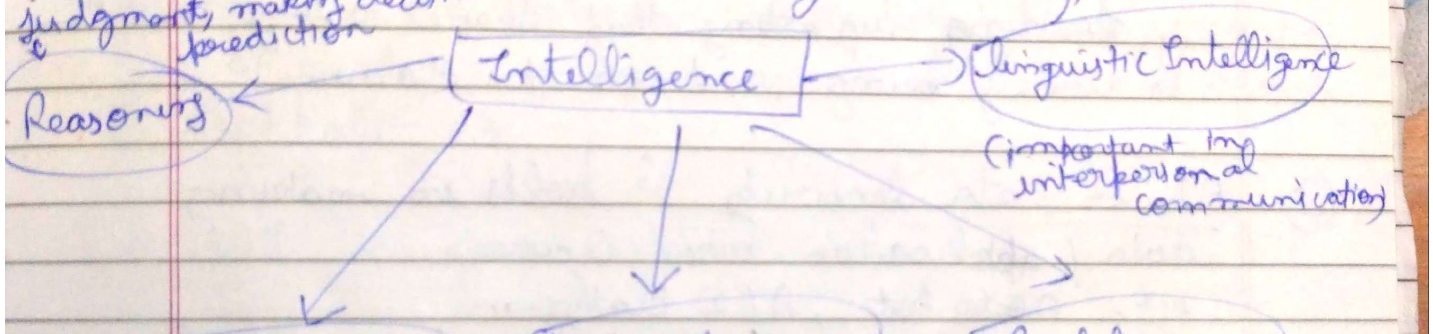
(6) Intelligent connection between Perception & action

what has been understood — what action has been taken

What is Intelligence composed of?

Artificial - Man made

intelligence - thinking power



Learning
↓
activity of gaining knowledge

Perception
↓
process of acquiring, gathering, interpreting, selecting, organizing sensor information

Problem Solving
↓
process of working through details of a problem to reach the solution (decision making)

Perception is a mechanism that puts data acquired by the sensor together in a meaningful manner.

Applications of AI

(1) AI in Gaming: e.g. chess, Poker, tic-tac-toe
Machine can think large number of possible moves

Robotic Surgery:- It is also known as robot-assisted surgery, is a type of minimally invasive surgery that uses robotic systems to perform surgical procedures. The robotic system is operated by a trained surgeon who controls the robot through console.

(2) AI in NLP: Natural language processing
↳ Machine can understand human language

(3) AI in Healthcare: Fast diagnosis of disease so timely
↳ Robotic surgery treatment can be provided

(4) AI in Finance: Adaptive intelligence
↳ automatic chatbots, algorithm trading (also which is deciding suggesting that share to buy, ML & AL is widely being used in this domain

(5) AI in Data Security: helps in making data / application more secure.
e.g. AEG bot, AI2 platform
They can detect bug in the software & prevent cyber attack efficiently in a more effective manner.

(6) Expert System:- It is an integration of software, machine and special information to provide reasoning and advice.
Ex: If we are not struck in some problem, we usually consult an expert that can give wise suggestion to deal with the problem. Likewise we can make expert system in the machine
(thoughts experience)

(7) Computer vision:- To understand the visual automatically by machine (e.g. image / video recognition)

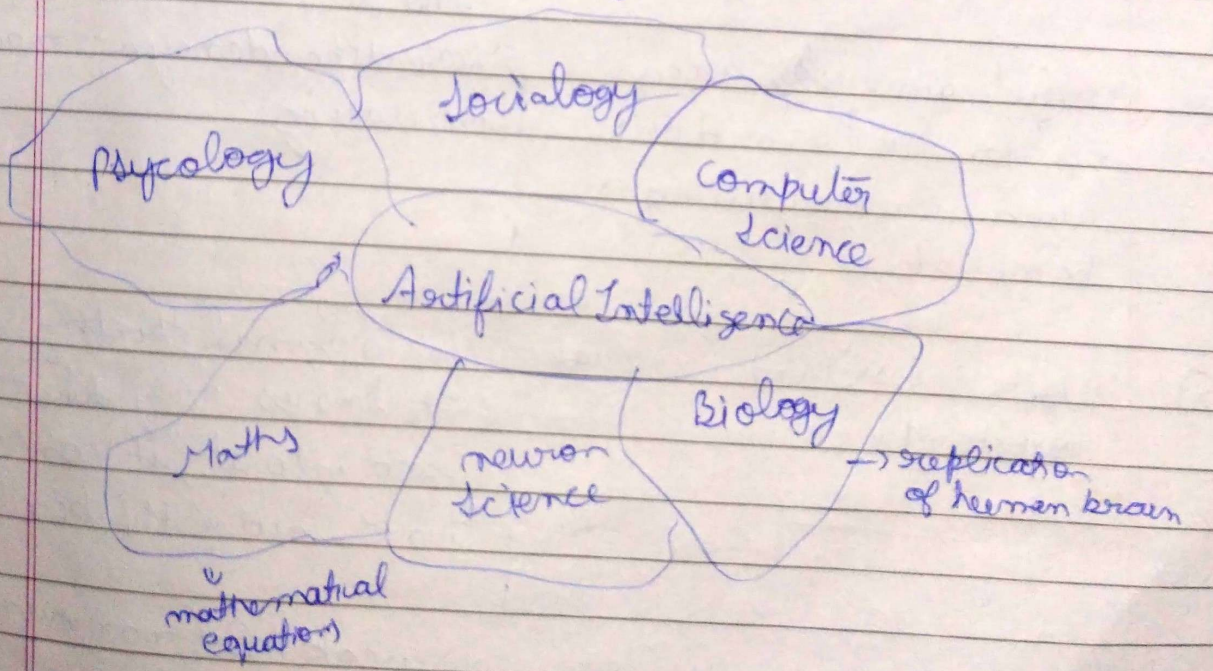
The robotic systems consists of surgeon's console, robotic arms and camera arm. The surgeon sits at the console and control the robot arms, which are equipped with surgical instruments. The camera arm provides 3D view of surgical site allowing surgeon to see area in greater detail

(8) Speech recognition: It helps in extracting the meaning of the sentence by human talk [Slang removal, noise removal etc.]

(9) Robotics: → Most Intelligent humanoid robots → Erica and Sophia (They can talk & behave like humans)

(10) AI in e-commerce: Automatic recomendation of products, handling service requests through chatbots etc.

AI is comprised of:-



→ Reasoning
→ Learning
→ Problem solving
→ Language understanding

Evergreen
Page No.
Date : / /20

Advantages of AI	Disadvantages of AI
① Accuracy ↑ & Error ↓	cost ↑ (as more S/w & H/w components will be used to increase the accuracy & reduce the error)
② High Speed / Fast decision making (eg AI system has dominated chess champion in chess game)	It can't think beyond the limits (eg System trained to play chess cannot play ludo)
③ Reliability is more	No feeling and emotions (eg replacing nurse with a robot)
④ useful in risky area (eg in biohazardous area, or diffusing a bomb etc)	More dependency on machine increasing
⑤ Digital Assistant (virtual assistant)	No original thinking (It knows only the method feed into it, it can't think out of the box)

Classification of AI

- weak AI (narrow AI)
- Strong AI (General AI)

i) Weak AI (also known as Narrow AI) (dedicated for one task)
- only trained for one specific task
- can fail in unpredictable ways if it goes beyond its limits
- It is able to perform dedicated task with intelligence
- It can't perform beyond its field or limitations
- It is not concerned with how the task is being performed but is affected by the efficiency of the task

volutionary algo. solves problem by employing processes
hat mimic the behaviors of living things. As such, it
es mechanisms that are typically associated with biological
volution, such as reproduction, mutation and recombination.
e weakest solutions are eliminated while stronger, more viable
phos are retained and re-evaluated in the next evolution.

Example
ll the goal being
to arrive at
ptimal actions to
achieve the desired
outcome.

Flying machine (predefined range of functions)
→ using logics
Apple SIRI (will be able to perform only the tasks for which it is trained)
→ Playing chess
Self driving cars, speech recognition, image recognition

perform like human
ii) Strong AI (also known as General AI)
could perform any intellectual task with efficiency like a human
It is the study and design of machines that simulate human mind to perform intelligent tasks. (which can think & perform tasks like humans, currently no such system exists)
The idea behind general AI is to make such a system which could be smarter and think like a human by its own.
i) borrowing ideas from physcology and neuroscience
ii) forgetting things, genetics, language understanding

Evolutionary AI

It is the study and design of machines that simulate simple creatures and attempt to evolve.
e.g. Ants, bees etc.

Super AI ← Intelligent than human (hypothetical concept)
↳ Seeing under thought process
Seeks to create machines that are better than humans
[machine communication machine]
machine could surpass human intelligence. It is an outcome of general AI.

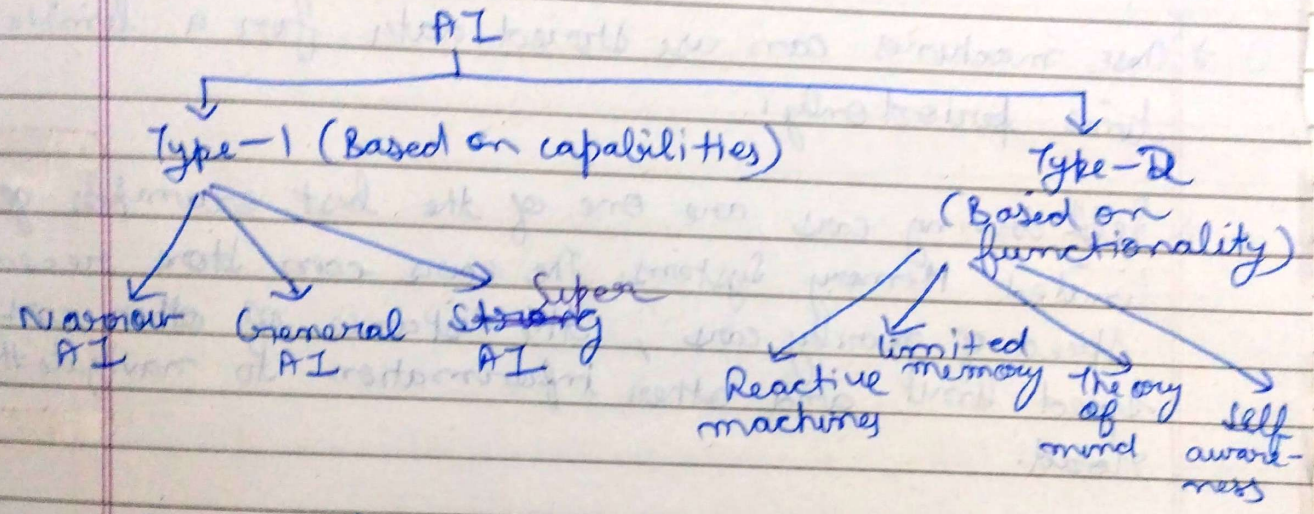
Page :
Date :

Super AI continued

Some key characteristics may include the ability to think, to reason, solve the puzzle, make judgements, plan, learn and communicate by its own.

Classification of AI

AI can be classified as



Type-1 has been discussed above.

Type-2

(1) Reactive machines

→ Purely reactive machines are the most basic types of AI

→ Such AI systems do not store memories or past experiences for future actions.

→ These machines only focus on current scenerio

and react on it as per possible best action.

→ IBM's Deep Blue system is an example of reactive machines

→ Google Alpha Go

(2) Limited Memory

→ Limited memory machines can store past experiences or some data for a short period of time.

→ These machines can use stored data for a limited time period only.

→ Self driving cars are one of the best examples of Limited Memory systems. The cars can store recent speed of nearby cars, the distance of other cars, speed limit and other information to navigate the road.

(3) Theory of mind

→ Theory of mind should understand human emotions, people beliefs and be able to interact socially like humans.

→ This type of AI machines are still not developed, but researchers are making lots of efforts and improvement for developing such AI machines.

(4) Self-Awareness

→ Self-Awareness AI is the future of AI. These

machines will be super intelligent, and will have their own conciousness, sentiments and self-awareness.

→ These machines will be smarter than human mind.

→ Self-Awareness AI does not exist in reality still and it is a hypothetical concept.

Page No.
Date : / /20

AI Agents [Responsible for any work output obtained from the system]

AI is defined as a study of rational agents

→ It is main component of AI
→ It is an intelligent ~~component of the~~ system that needs to be developed

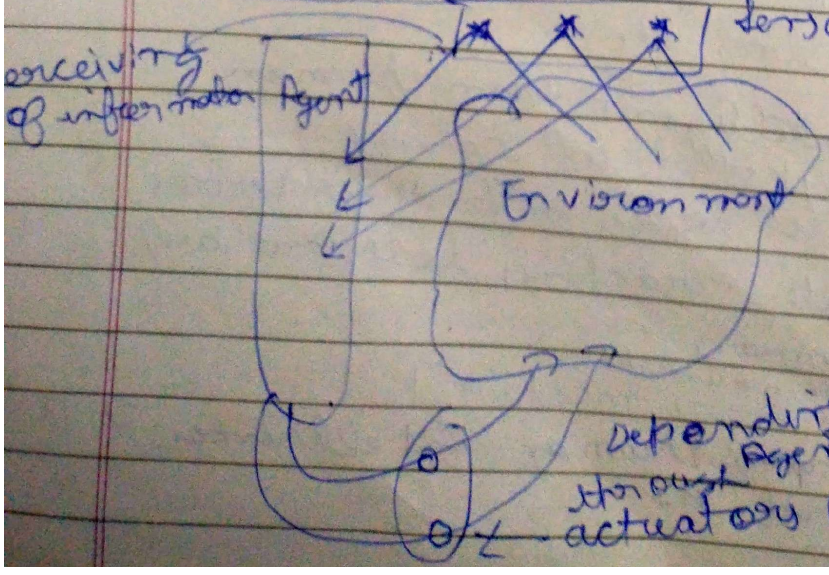
Agent → gathers information from the environment
→ perform action accordingly

Agents → Person
→ Firm
→ Machine / Software

rational agents are the one that can make decisions

AI system is composed of → Agent
→ environment

Agent is anything like :-
i) Perceiving its environment through [sensory]
ii) Acting upon that environment through [actuators].



Agent perceives info about the environment through the sensors

→ Depending upon the info received, agent acts upon the environment

eg: AC → If we set the temperature of AC at 25°C, now sensors in the AC keeps on sensing the environment and switch off as soon as the temperature reaches 25°C it switch of AC is switched off through actuators. Again it will keep on sensing the temperature and switch it on as it exceed through actuators as temperature exceeds 25°C

∴ Through sensors and actuators, agents act upon the environment.

Types of AI Agents

① Simple Reflex Agents → (works only on limited domain)

→ Works only on current situation / perception
→ and ignores the history of previous state

→ follows Condition - Action Rule
if condition true :- do this
If condition is true then take some action

limitations :-
i) very limited intelligence (can perform only the work that is stored in it)
ii) No knowledge about non perceptual parts
(if some issue arises for which it will have no knowledge regarding what to do)

Q.) of Simple reflex agent

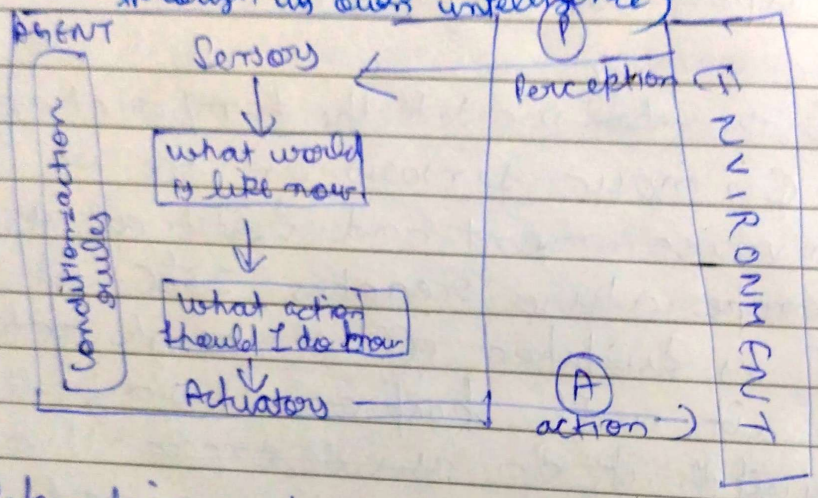
→ AI enabled camera that clicks photographs and performs auto focus or performs filters on applies its own based on current scenario. It does not depend upon the previous photograph taken.

Lack of State

(ii) can go into infinite loop

↳ (may try to complete the task that cannot be performed as it cannot analyze the current situation through its own intelligence)

vaccum cleaner is also an example of simple reflex agents



Depending upon the action, agent will instruct to perform some action on the environment through actuators.

② Model based reflex agents (Knowledge)

→ By analyzing the current situation, works by finding the rule whose condition matches current situation.

→ can work in partially observable environment and track situation.

→ Agent keeps track of internal state which is adjusted by each percept and that depends on percept history.

↳ Model → how things happen in the world

Agent state updation required information:

① how world is evolving

② how an agent's action affects the world.

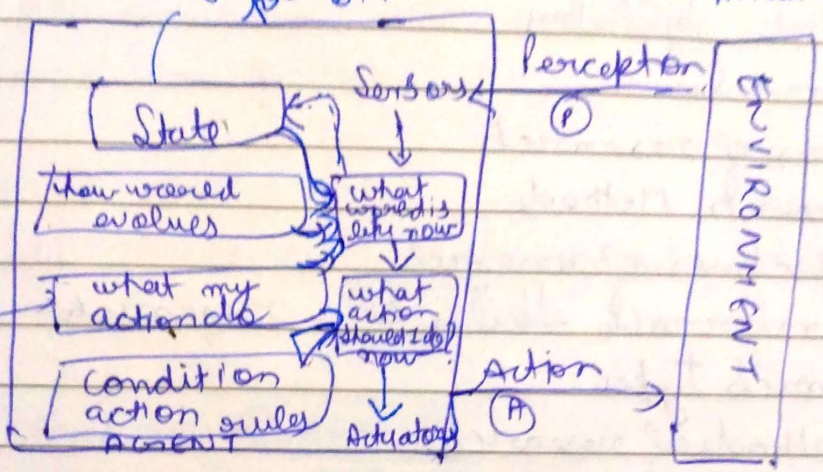
present state
updation required — how the world evolves
situation decision
① — ①
② — ②
③ — ③
what my action do

Everygreen
Page No.
Date

for storing previous experience

Sensor – perceives the environment
Actuator – performing the action

cleaner
e.g. vacuum
(environment – how floor)



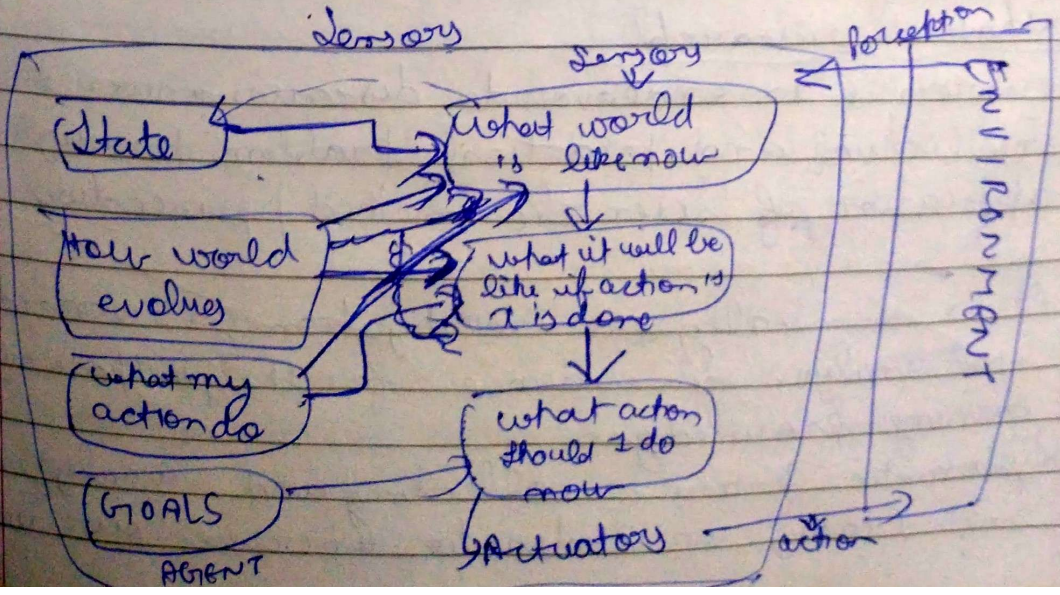
what is the result of action

⇒ It stores the previous experience also (in the form of state) which is used for examining the current state

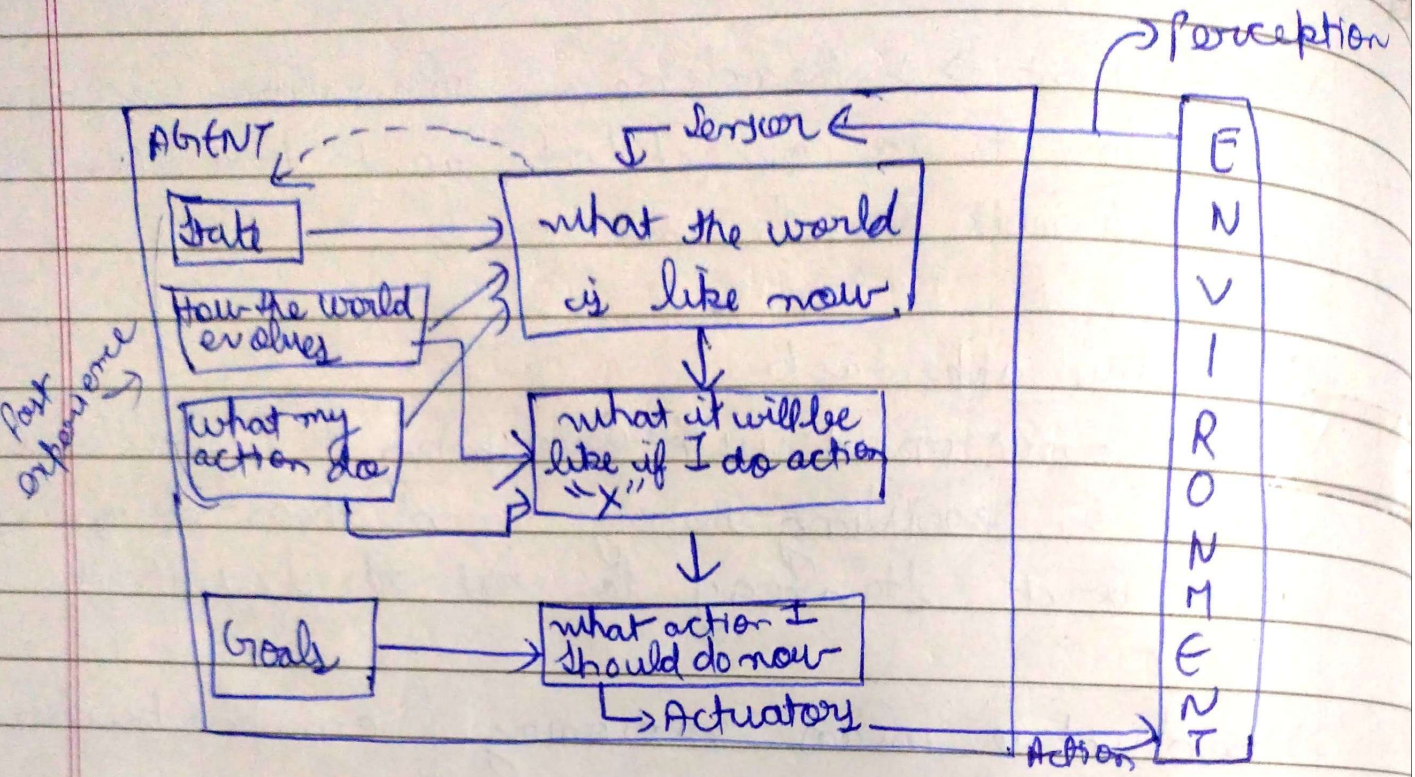
(3) Goal based Agents

→ focuses only on reaching the goal set

⇒ Agent takes decision based on how far it is currently from the goal state.
→ Every action is taken to minimize the distance to goal state
→ more flexible agent (know the goal)



Goal based agents (continue)

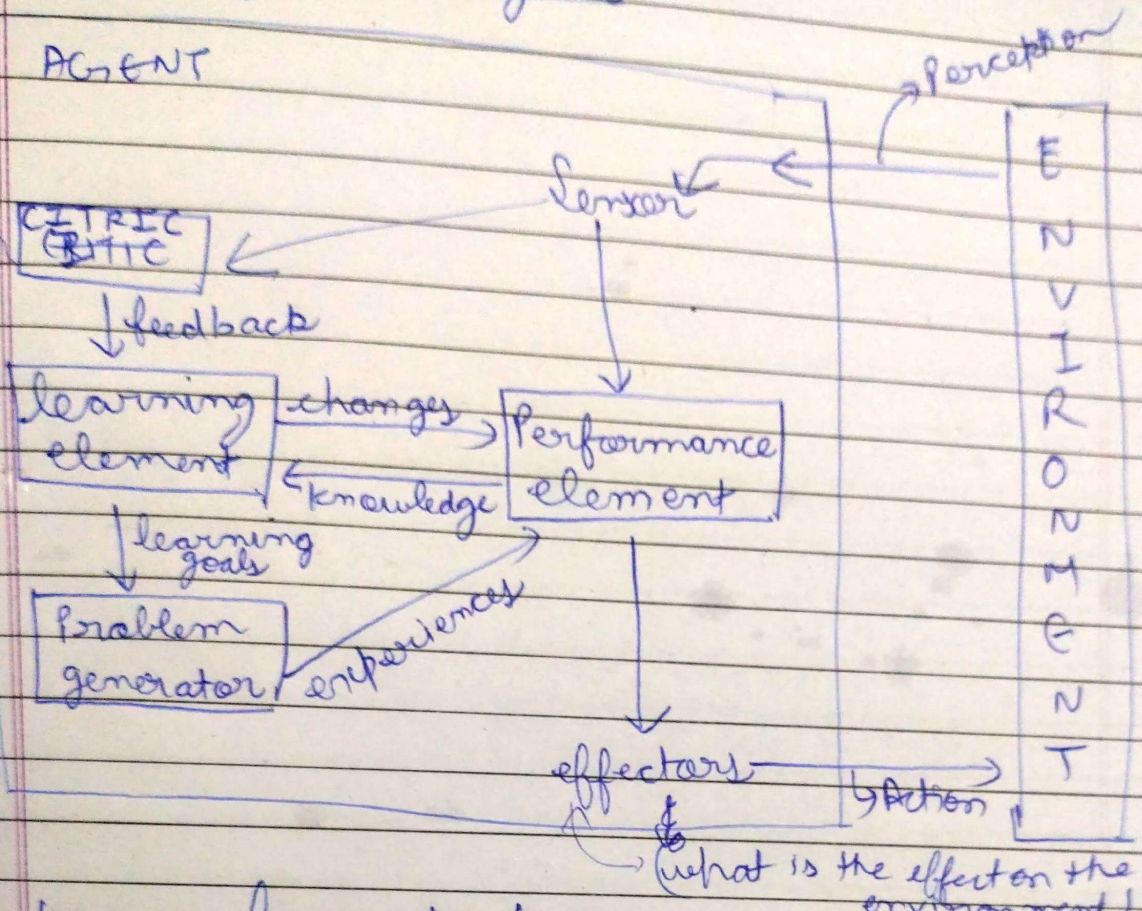


Ex. Rubics cube

Environment – Rubics cube
Sensors are perceiving Rubics cube (current condition)
Then agent will see that how much it becomes nearer to the goal (i.e rubics i.e how much is the rubic's cube solved in order if it takes action X

State – Stores the previous actions taken by the agent.
∴ Agent also checks if that specific state is previously encountered by consulting the state. If similar / same scenario is encountered before also, it will take the actions

(5) Learning based agents



→ Learn from its past experiences
→ Start to act with basic knowledge and then able to act by adapting learning. (learning mechanism keeps on enhancing with past experiences)

Components:-

learning element :- Improve system by learning from environment. (It suggests the learning goals to the problem generator)

Critic → checks agent working and gives feedback. (check the performance & gives feedback accordingly to the learning element)

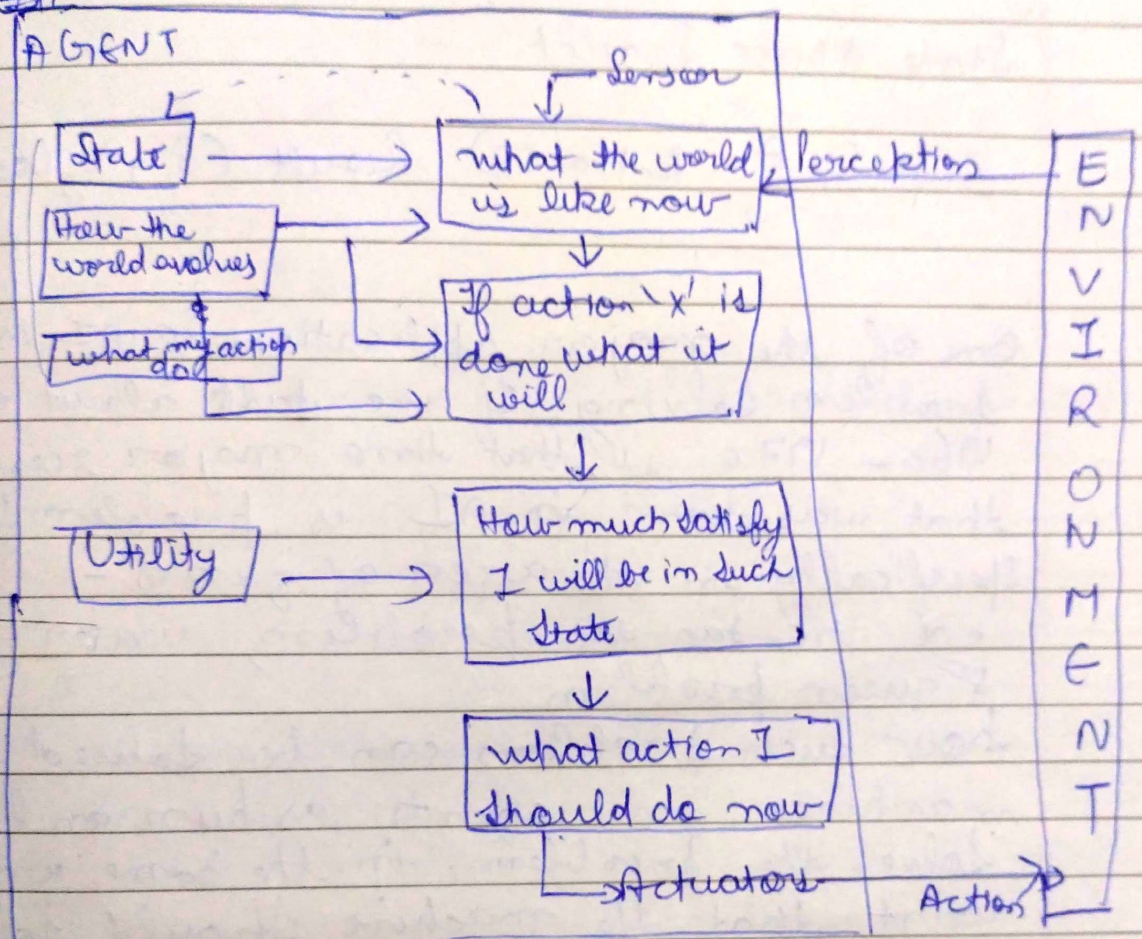
Problem generator :- Suggest the actions to be taken and gain information.

Performance element :- Select the action to perform. (Takes the decision)

Robot Sofia → Talks to people + reply to them and stores the experience / knowledge gained & use it for future

accordingly. Then actions are performed through actuators. Besides, it also keeps on comparing the with the goal state in order to check, to what extent current state is near to the goal state.

(4) Utility based agents



(with Utility, it checks how much it is fulfilled)

→ Agents are more focussed towards the preference of each state.

→ Act based not only on goals but also the best way to achieve goals

→ May be there are many alternatives for one state but agent choose best action to perform that task.

Let's say a task has many subtasks too. Then utility based agent focusses on each subtask too. They try to find the best way to achieve the solution so that task can be performed efficiently.

ex: google maps

AI

Problem Solving

State Space Search

S: { S, A, Action(s), Result(S, a), Cost(S_A) }

One of the major application of AI is problem solving. If we talk about era of 1960 - 1970, at that time major research that was done in AI is problem solving specifically in the area of games:-
e.g. tic tac toe problem, water jug problem, 8 queen problem
how such problems can be solved through machines and agents, as human being solves the problem, in the same way we want that the machine should solve the problem. That is called problem solving area.

→ In problem solving area, state space search is very important.

What is state space search over here?
↳ No. of states in which a problem can go i.e. All the different states in which a problem can go starting from initial state to goal state.

e.g. Let say, we need research on a topic, then obviously we will not reach at all the places in one time, we start with a start

State, we have a goal state, now in between start and goal state, there may be many intermediate states. we go level by level in order to achieve the goal state.
So all set of states, we use them in the form of representation.

In AI, we majorly talk that how to represent the problem
→ precisely

→ analyze

Because representing the problem is a major task because of you are giving this problem to your machine or to your agents then you cannot tell it verbly, that this is the problem, If you start playing any new game, you don't know anything about that game, you want someone to tell you that how to play that game, all the rules, what are the set of states in it, All those points are necessary for us to solve the problem precisely.

If we precisely represent the problem, we can easily analyze it also
ex how to play the game, how different steps can be taken

Evergreen
Page No.
Date : / /20

→ If we want the problem to represent in all set of states, then we use State Space Search

$S: \{S, A, Action(S), Result(S,a), Cost(S,a)\}$

where S represents the total no. of states major states in S could be start state, goal, intermediate

S= start, intermediate, goal

A → set of actions

eg: 8 puzzle problem

very first requirement is → to tell the machine or agent, the problem very clearly

2	3	4
5		1
8	7	6

↓
start state

3*3 board with 9 spaces in which we have 8 tiles which are numbered & one empty space

→ Represent the problem precisely
→ board of 3*3
→ we must also know goal state

S

2	3	4
5		1
8	7	6

3*3

Start State

G

1	2	3
8		4
7	6	5

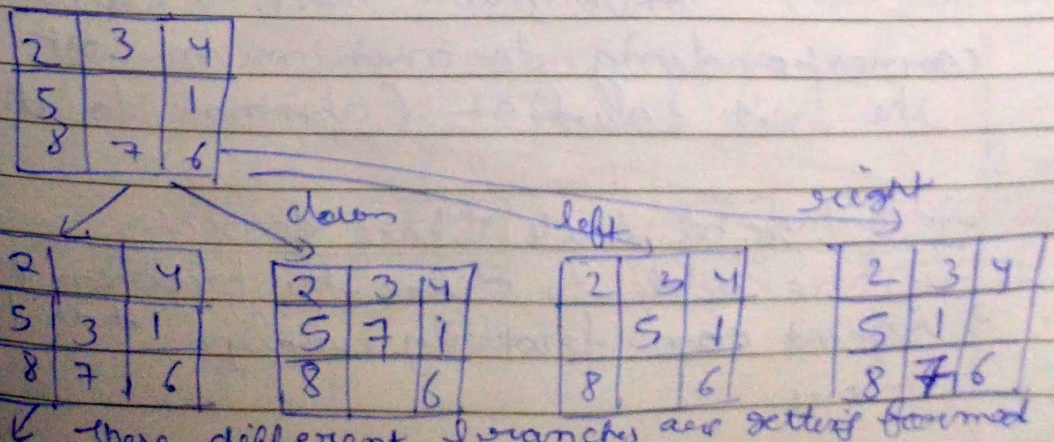
Goal State

Now when we start exploring start state in order to reach the goal state, many intermediate states will start forming. Each intermediate state will be compared with the goal state to check, if it is a goal state or not, if goal state is found then searching will stop at that point.

Action → set of all possible actions considering the empty space

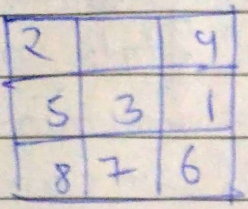
up
↑
left ← empty state can be moved → right
↓
down

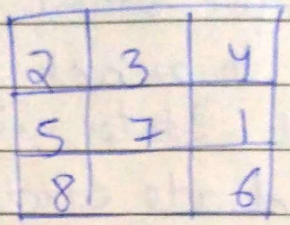
→ there can be four possibilities (up, down, left, right)
→ we should know about legal and illegal states
→ All such conditions must be predefined
Ex empty space can be replaced by left, right, upon down
→ Now one action will be chosen out of these actions



Evergreen
Page No.
Date : / /20

upward →





tile can be moved in left, right & down position, up is invalid

legal - left, right, up
illegal - down

legal → right, up, down
illegal → left

→ So this is set of all possible moves
→ which ever set of action you choose, it will be the Action(),

Result → the above all are resultant states

Cost → It is any constant value (e.g. in case of graphs, moving from one edge to another and in case of trees, moving from one node to another)

→ objective → Searching should be done in minimum cost → Solution corresponding to minimum cost will be the best solution (optimal solution)

Advantage of State Space Search

→ we are able to define the problem precisely
→ Agent can properly analyse that how to move

Evergreen
Page No.
Date : / /20

→ In AI whenever when we talk about any searching problem we have started exploring the states which will lead to more no. of states in order to reach the goal state

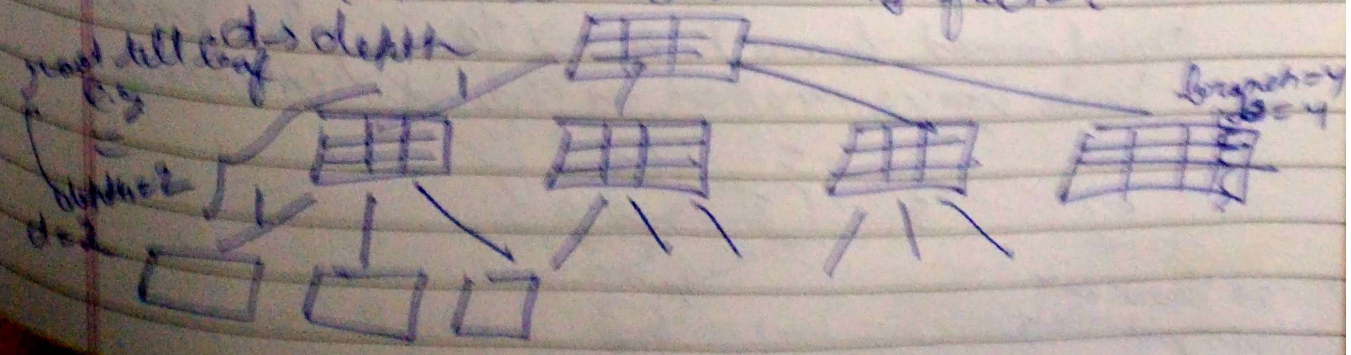
Searching can be done in 2 ways :-

- uninformed (blind search)
- informed (heuristic search)

uninformed → don't know any thing about the domain of the problem, only start and goal is known, states are being explored and matched with the goal state. Stop at the point when state is matched with the goal state.

informed search - concept of heuristic is used in order to solve the problem quickly we go toward the local benefit so that problem can be solved quickly.

uniformed search takes exponential time (non polynomial)
→ $O(b^d)$
where b is the branching factor

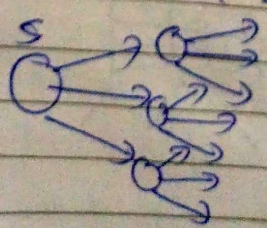


In the problem discussed, uninformed search can go upto $3^{20} \rightarrow 3.5$ billion states can be covered
that is why informed search is more preferable than uninformed search

Uninformed Searching vs Informed Searching

Uninformed Searching	Informed Searching
1) Search without information	1) Search with information
2) No knowledge	2) use knowledge to find steps to solution
3) Time consuming	3) Quick solution
4) More complexity (Time, Space)	4) less complexity (Time, Space)
5) DFS, BFS etc. DFS → Depth first search, BFS → Breadth first search	5) A*, AO*, ~~Heuristic DFS~~, best first search, Hill climbing

Uninformed searching → It is also known as brute force method or blind searching



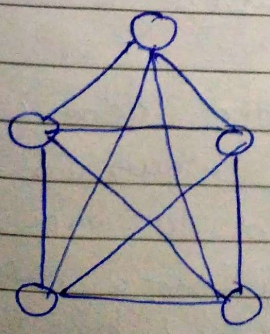
e.g.

S → Start state
G → goal state

ie we will start from start state and explore all the possible states in order to find the goal state. If goal state is not found, then exploration will continue on the next ~~state too~~ step too, in order to look for goal state.
In this way, after searching all the state spaces, finally we will be able to reach the goal state (we are checking again & again if the intermediate state is the goal state

→ only knowledge of start and goal state
→ no knowledge about the domain or problem
→ no knowledge about how to reach ~~the~~ the goal
→ no guide is there that can guide about the path to be followed

Informed searching → we are using information. This information is known as heuristics
∴ Eg travelling salesman problem (TSP)



There are 5 cities connected in this way

Problem statement

→ We have to start at one city, cover all the other cities such that distance covered is minimum then have to reach again at the same city.

In case of 5 cities, if we apply uninformed search ie brute force method (ie cover all the possibilities)
then acc to TSP no. of possibilities (search space) = (n-1)!
In this case, it would be (5-1)! = 4! = 24 states
i.e 24 possibilities needs to be explored
After covering all the 24 possibilities, we will get a final optimal solution. It is guaranteed to provide an optimal solution.

limitation
→ exponential time is required

If no. of cities = 100
then 99! search space, that is a very large no. which will be very time consuming.

Informed Search

Here we use heuristic method
→ Heuristic means assumption

when do we use heuristic search?
→ when search spaces grows in exponential power then we need to use heuristic search
→ time complexity is very high which we generally call as NP (Non polynomial) problem which can't be solved in polynomial time

→ To solve non polynomial time problem, cost is very high (as time & space complexity is very high)

→ If we want to solve non polynomial problem in polynomial time then we can use heuristic

→ We need to take knowledge of the domain that will help to reduce the searching time

→ time required for the search will be reduced but we may not get an optimal solution

Let's take 24 puzzle problem
In case of uninformed search, we may ~~to have~~ have to search 10^{24} states, which is practically not possible.

(b^d ← depth
↳ branch factor)

→ Informed search may or may not give an optimal solution.
→ uninformed search is guaranteed to give an optimal solution.

Search Algo.

Breadth First Search

Uninformed
- Breadth First Search
- Uniform cost Search
- Depth First Search
- Depth Limited Search
- Iterative deepening depth first search (IDFS)
- Bidirectional Search

Informed
- Best first search
- A* search
- AO* algorithm
- Hill climbing

Uninformed Search

(I) Breadth First Search

1. Create a variable called Node-list and set it to the initial state.
2. Until a goal state is found, or Node-list is empty:

a) Remove the first element from Node-list and call it E. If Node-list was empty, then quit

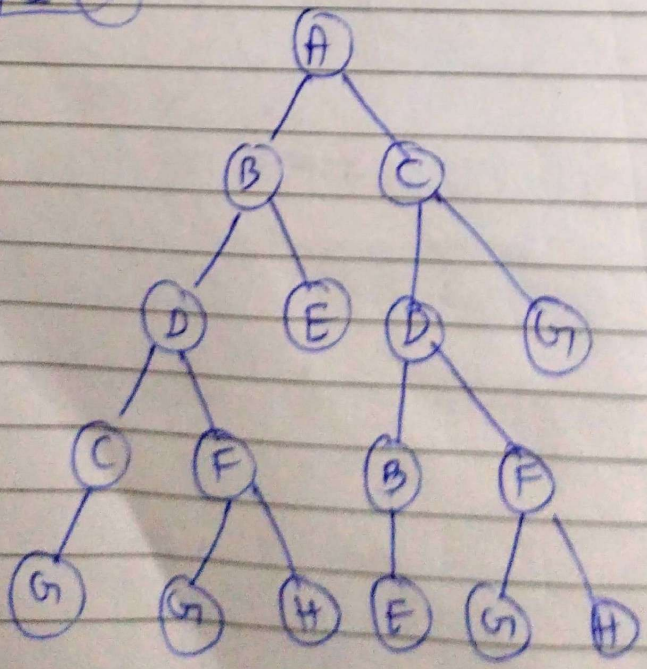
b) For element E do the following:-

i) Apply the rule to generate a new state;

ii) If the new state is a goal state, quit and return this state.

iii) Otherwise, add the new state to the end of Node-list.

Example (1)



(In this case, there are multiple goal nodes)

Evergreen
Page No.
Date : / /20

Step 1: Initially Node-list contains only one node corresponding to the source state A

Node-list : A

Step 2: A is removed from Node-list. The node is expanded, but its [Now remove A from the Node-list (since its not the goal state and add its successors)] children B and C are generated. They are placed at the back of Node-list.

Node-list : B C

Step 3: Node B is removed from Node-list and is expanded. Its children D and E are generated and put at the back of Node-list.

Node-list : C D E

Step 4: Node C is removed from Node-list and is expanded. Its children D and G are added to the back of Node-list.

Node-list : D E D G

Step 5: Node D is removed from Node-list. Its children C & F are generated and added to the back of Node-list.

Node-list : E D G C F

Step 6: Node E is removed from the Node-list. It has no children

Node-list : D G C F

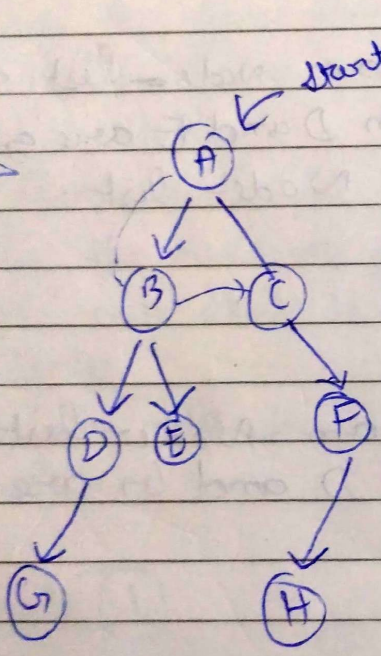
removed from the Node-list its children

Step 7: D is ~~expanded~~, B and F are added to the Node-list

Node-list: G1 C F B E

Step 8: G1 is selected for expansion. It is found to be goal node.
Hence algorithm returns the path A-C-G1 by following the parent pointers of the node corresponding to G1.

Example 2



Start node - A

Goal

Node-list: {A}

Node-list: {B, C}

can also be called as queue

Node-list: {C, D, E}

Node-list: {D, E, F}

Node-list: {E, F, G1}

{A}
↳ {B, C}
↳ {C, D, E}
↳ {D, E, F, G}
↳ {E, F, G, H}
↳ {F, G, H, I}
↳ {G, H, I, J}
↳ {H, I, J}
↳ {I, J, K}
↳ {J, K}
↳ {K}

route
A → B → C → D → E → F → G
K ← J ← I ← H

Evergreen
Page No.
Date : / /20

Node-list : { F, G }

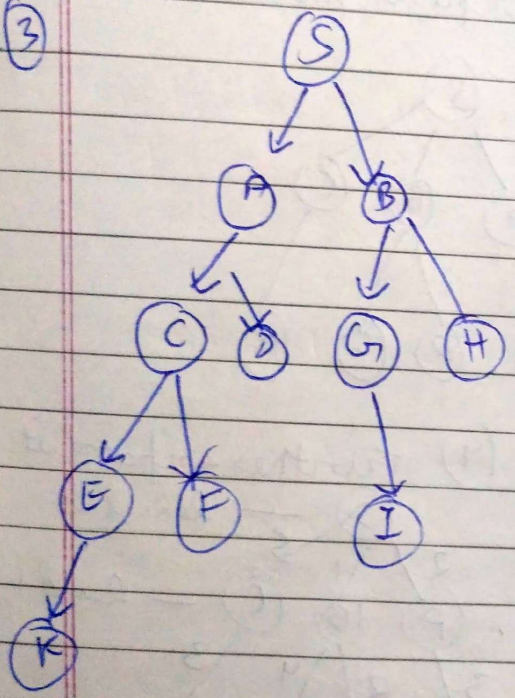
Node-list : { G, H }

Node-list : { H }

Path

A → B → C → D → E → F → G → H

Ex (3)



{S}

↳ {A, B}

↳ {B, C, D}

↳ {C, D, G, H}

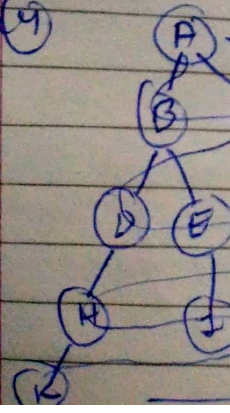
↳ {D, G, H, E, F}

↳ {H, E, F, I}

↳ {F, I, K}

↳

S → A → B → C → D → G → H → E → F → I → K



level 0

→ level 1

level 2

level 3

level 4

Source node - A

Goal node - E

{A}

↳ {B, C}

↳ {D, E, F, G}

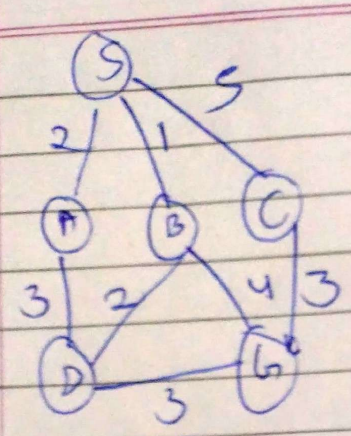
↳ {E, I, H}

↳ {I, K, H}

↳ {K, H}

Evergreen
Page No.
Date : / /20

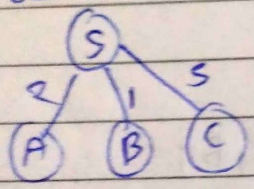
Eg (4)



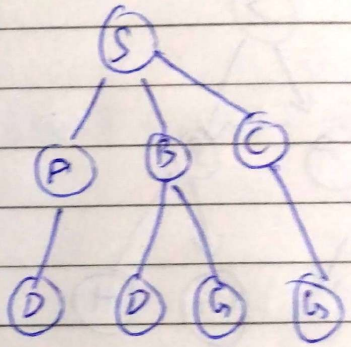
Find route from S to G using BFS

Step (1) S

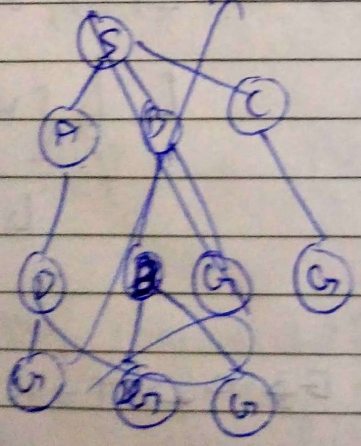
Step (2) Expand



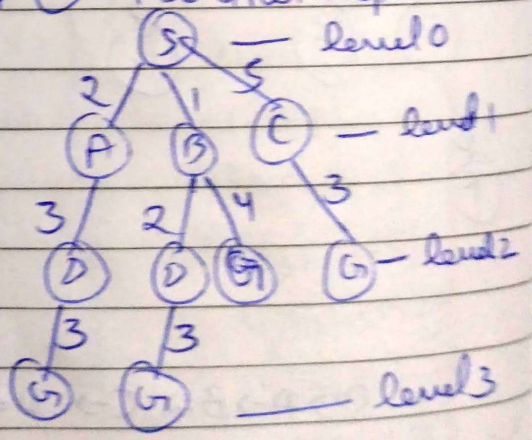
Step (3) Expand further



Step (4) Further expand it



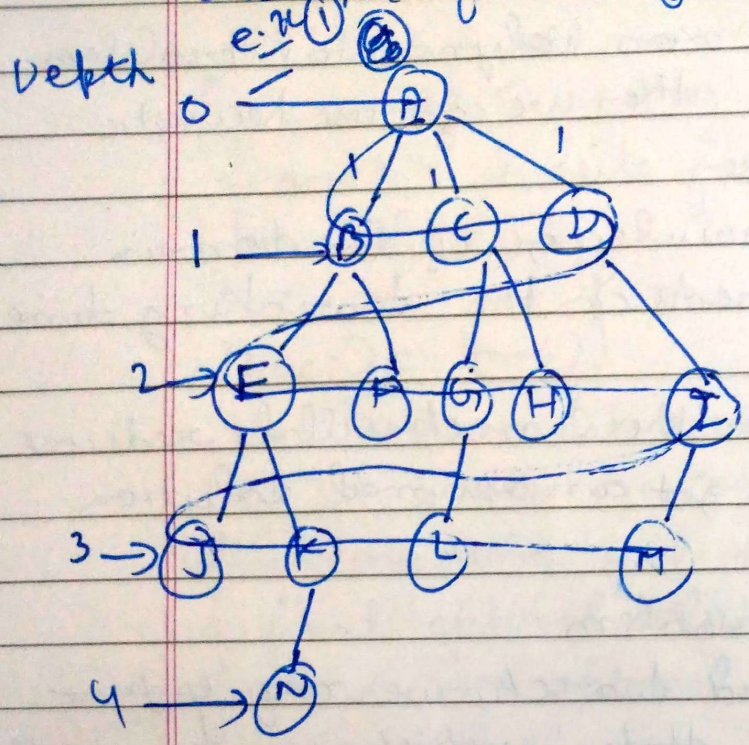
Step (4) Further expand it



Path → SBG or SCG

~~Example of depth first~~

Example of Uninformed Search ① Breadth first Search



Assuming cost = 1

→ usually when cost of all the edges are same, if cost is different then heuristic is used

Uninformed search technique (brute force)

→ It follows FIFO principle. Data structure used- Queue

→ It explores shallowest node first (it goes level by level, covering all the nodes at a particular level then moving onto the next level.

→ optimal for unweighted graph and is particularly suitable when all the edges have the same cost.

→ It is complete

→ It gives optimal solution

→ ~~It is optimal~~ Time complexity

$O(V+E)$

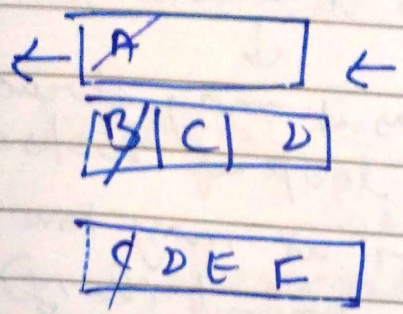
In AI it is represented as $O(b^d)$ (b = branch factor i.e. max no. of children a node can have, d = depth)

→ BFS is also known as

b=3

considering ex (1)

Traversing this tree with BFS using Queue DS



E F G H

F G H I

G H I J K

H I J K

I J K L

J K L

K L M

L M

M N

N

Let us assume goal state to be G, it is to be searched

then complexity = $O(b^d)$

Path from root to Goal A(b)

b=3, d=2 (in this case) (transition value)

assuming branch factor = 3

then complexity = $b^d = 3^2 = 9$

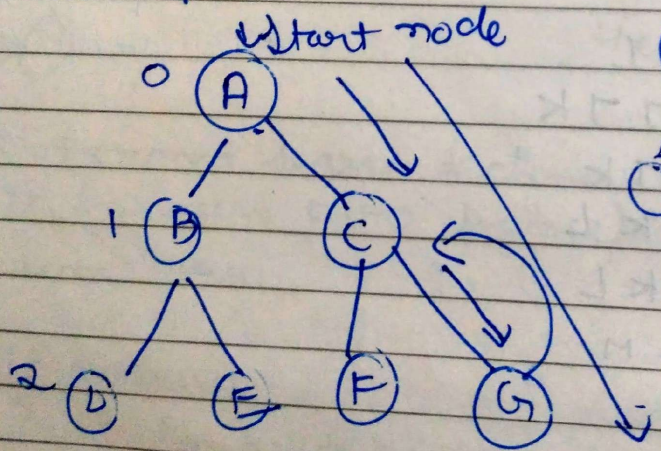
max no. search required = 9

Blind Search

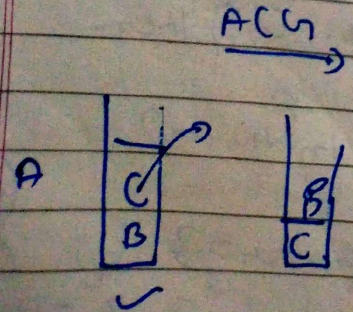
Evergreen
Page No.
Date : / /20

② Depth first search → works in a brute force manner with present knowledge (does not work on heuristic)
→ uninformed search technique
→ Stack [LIFO]
- Deepest Node
- Incomplete (may not give the solution)
  ↓
  as it may struck into a loop if loops are present, or it may have to go to infinite (long) depth if search space is infinite
- Non optimal
- Time complexity
  O(V+E)
  $O(b^d)$

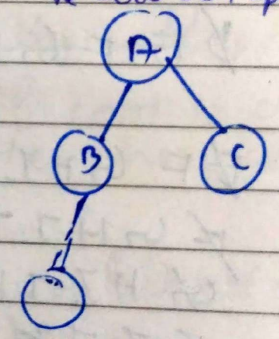
b = branching factor
d = depth



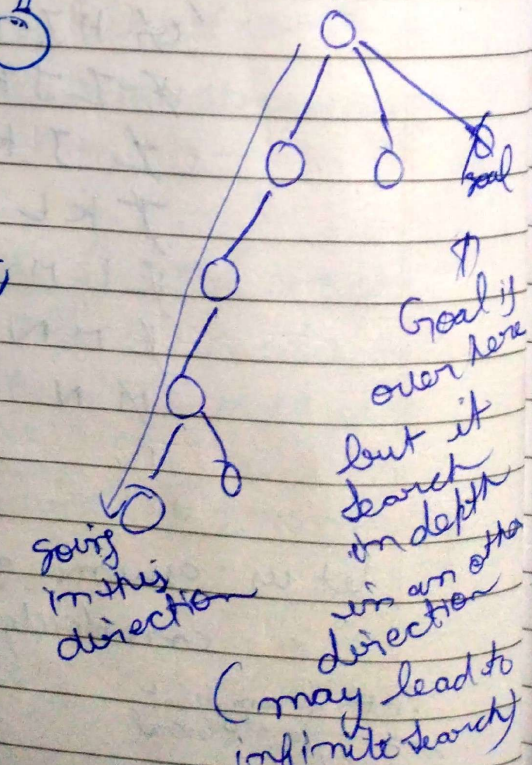
→ A C G F B E D

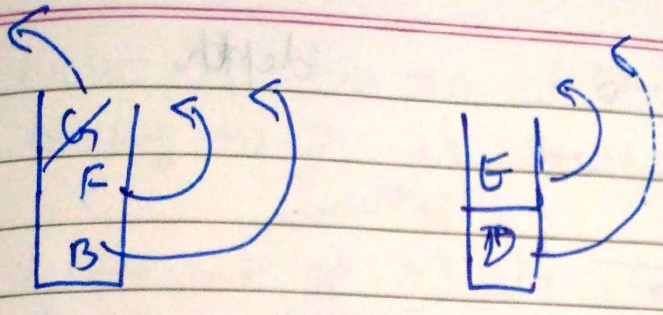


order → Root left Right



e.g.





Let G be the goal state to be searched

Complexity = $O(b^d)$

let us assume a binary tree with branching factor = 2

no of children possible

Complexity = $2^2 = 4$ (max 4 search operations are required)

Deepest Node → It will go deep in one direction in order to find the goal state. If it does not find the goal state, it ~~backtracks~~ backtracks in ~~order to~~ another direction to search for the goal state

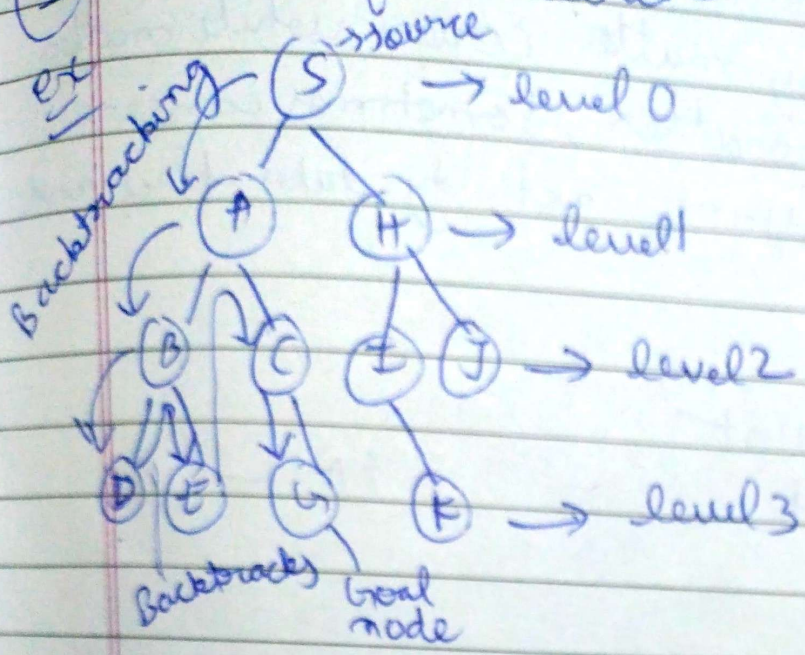
Depth First Iterative Deepening (DFID)

To know about DFID, we should be clear about the concepts of Depth first search (DFS) and Depth limited search (DLS)

Considering the example

(II) Depth first search

Ex:



After reaching at (G), it will terminate as goal node is found.

Time complexity $T(n) = 1 + n^2 + n^3 + \dots n^m$

$\simeq O(n^m)$

m = max depth of any node

Space complexity $S(n) = O(bm)$

b - level

⇒ is non-optimal as it may go to infinite loop can may generate large no. of steps

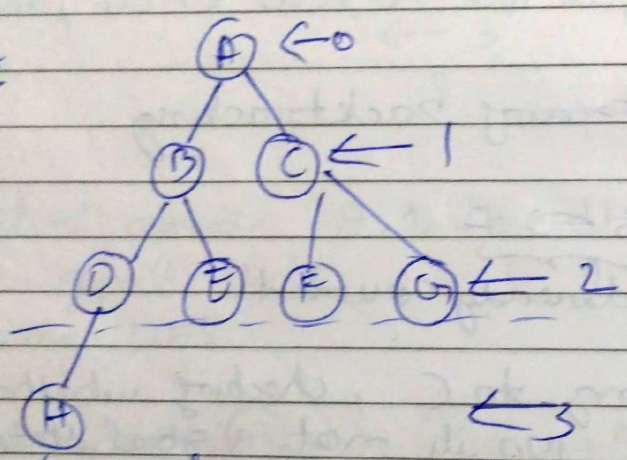
→ Similar to DFS with a predefined / predetermined limit. It can solve the drawback of the infinite path in DFS. In this algo, the node at the depth limit will treat as it has no successors & nodes further.

(3) Depth limited Search

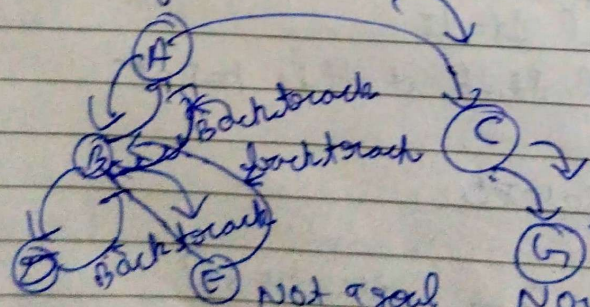
Used to remove the problem associated with depth limited search

→ DLS works like DFS but it imposes a limit on depth to be searched

Ex



let's assuming limit = 2 (L=2)
ie we can search upto second limit we can't go beyond this limit.
It doesn't matter how many states are there after this limit



DLS can be terminated with two conditions of failure:-
Standard failure value:- It indicates that problem does not have any solution.

Cutoff failure value: It defines no solution for the problem within the given depth limit.

Advantages: - It is memory efficient

Disadvantages: - ⇒ incompleteness (can move only upto specified limit even if the goal is not found, cant go further)
⇒ It may not be optimal if the problem has more than one solution.

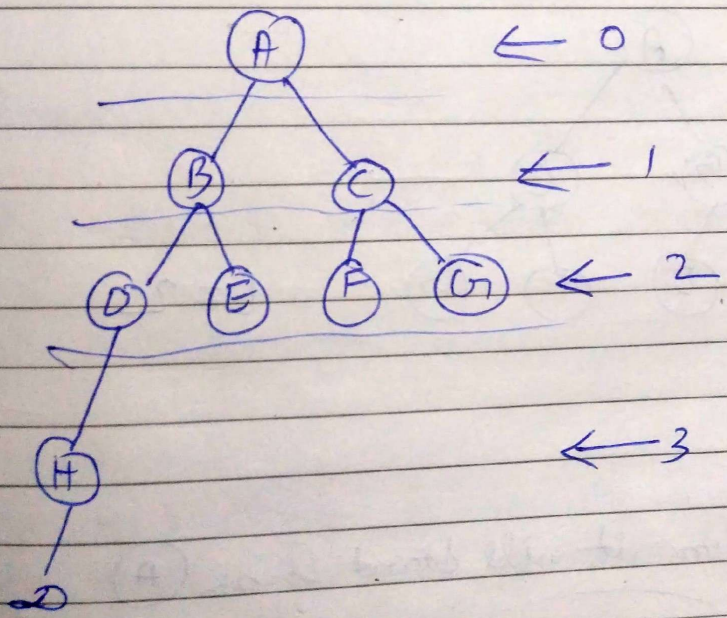
Problem

If Goal State = H & depth limit = 2
In this case we will never be able to reach the goal state

Now to overcome this drawback of depth limited search (DLS)

(4) (b) ⇒ Depth First Iterative Deepening (DFID) is used

let us understand through an example

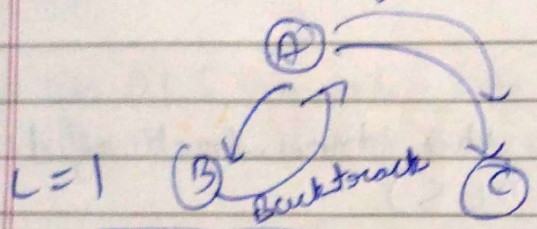


let us assume goal state = H & start state = A
limit will always start with 0 ; L=0

Now it check if A is the goal state, No it is not
so depth limit, L will be increased by 1
∴ L=1

→ Now it will again start searching from (A)

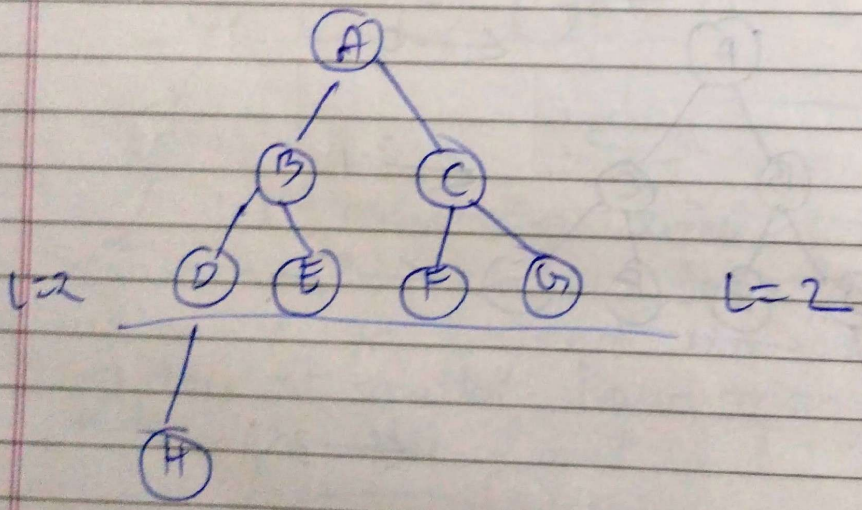
A is not the goal state, it goes to (B)



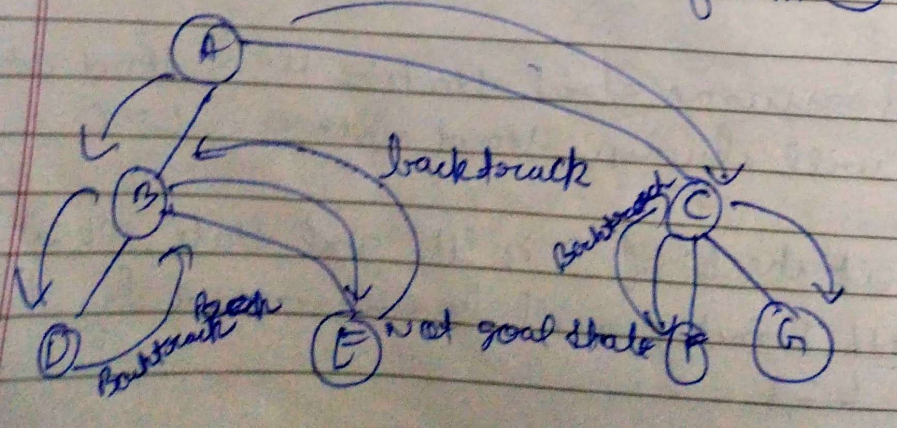
B is also not the goal state

Then Go to C, its not the goal state

Now again it will increase the depth limit by 1 ∴ L = 1+1 = 2

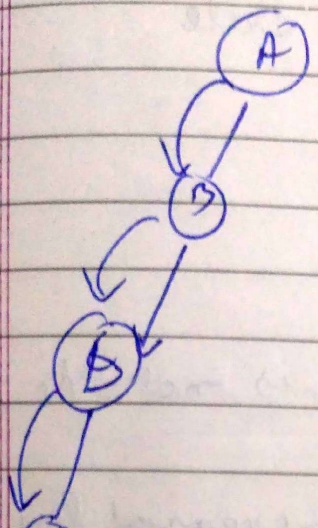


Now again it will start from (A)



Evergreen
Page No.
Date : / /20

Again Goal State not found
So L is increased by 1 again
L = 2 + 1 = 3



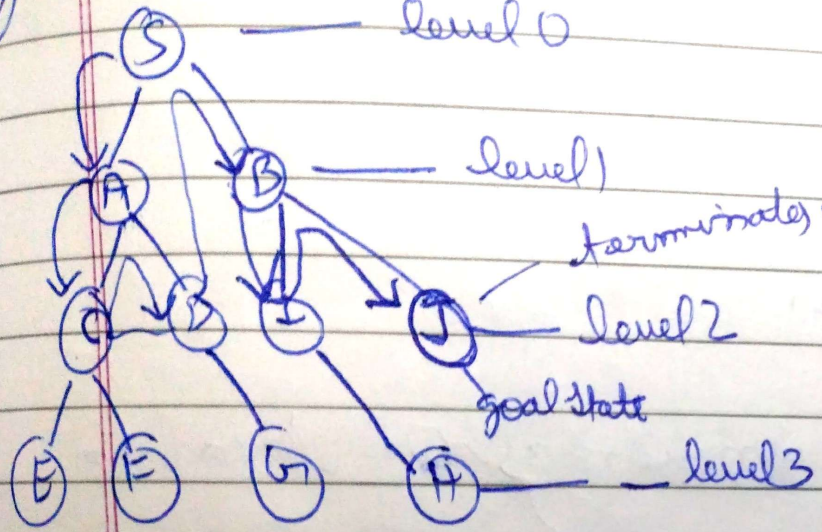
L=3 H ← Goal State found

Advantages

1) It inherits advantages of both depth first DFS & DLS
Fast searching

2) Disadvantages
1) Keeps repeating process of previous phase
It has to start again from the starting point in every iteration.

(3) DLS Example



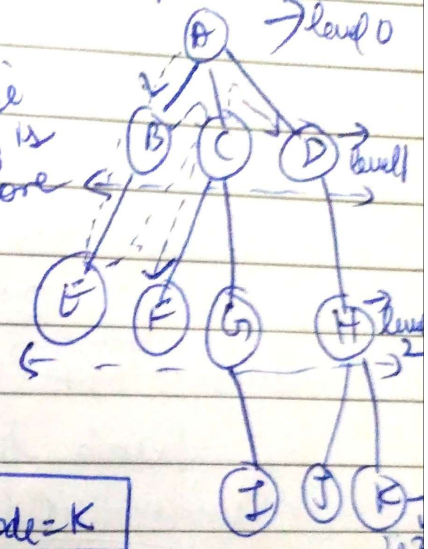
limit - level 2

~~Even~~ if (I) would have been at level 3, then goal could not be found.

DFID searches at a particular level till the goal node is found. If the goal node is found, it stops otherwise its level is increased by 1 & search again begins from the source.

(4) DFLD example

~~Depth wise~~ level wise searching is being done



Goal node = K

Depth/ level	Iterative deepening search
0	A
1	ABCD (level increased by 1)
2	~~ABCDE~~ ABECFGDH (upto level 2)
3	ABECFGIDHJ(K) (upto level 3) Goal node

State Space Search

(4) Uniform cost search algorithm:-

→ It is used for traversing weighted tree or graph.
→ It comes into play when different cost is available for each edge.
→ The goal of UCS is to find a path to the goal node which has the lowest cumulative cost.
→ It expands the nodes according to their path cost from the root.
→ It can be used to solve any graph/tree where the optimal cost is in demand.
→ A UCS algo. is implemented by the priority queue.
→ It gives maximum priority to the lowest cumulative cost.
→ It is equivalent to BFS if the path cost of all the edges is same.

Advantages

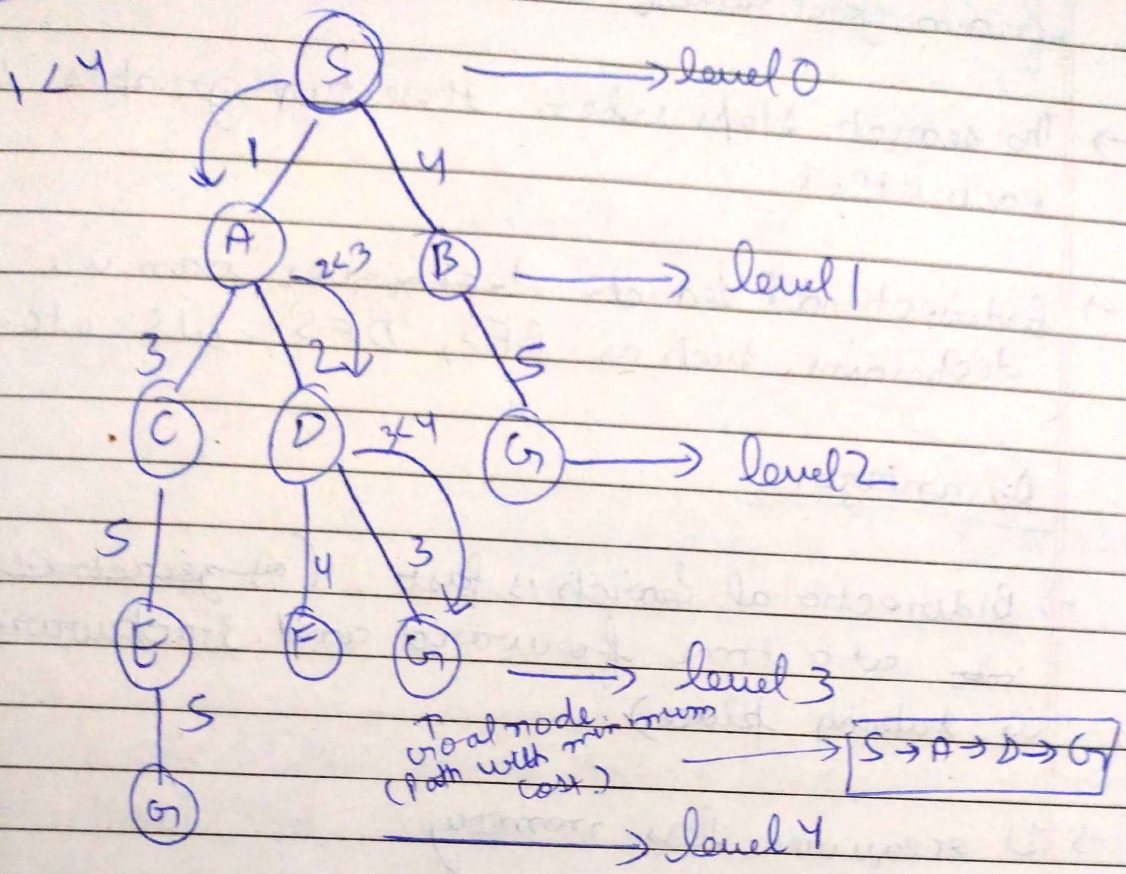
→ UCS is optimal because at every state the path with the least cost is chosen.

Disadvantages

→ It does not care about the no. of steps involved in searching and only concerned about the path cost. Due to this, this algo. may be stuck in an infinite loop.

Page :
Date :

Example



G → Goal node

(There are many G nodes, but which G to search, it depends upon the cost of path involved)

(6) Bidirectional Search

→ It runs two simultaneous searches, one from initial state called as forward search and other from goal node called backward search to find goal node.

→ It replaces one single search graph with two small subgraphs in which one starts the search from initial vertex and other starts

from goal vertex.

→ The search stops when these two graphs intersect each other.

→ Bidirectional search ~~techniques~~ can use search techniques such as BFS, DFS, DLS etc.

Advantages

→ Bidirectional search is fast (~~as graph is divided into~~ at a time forward and backward search is taking place)

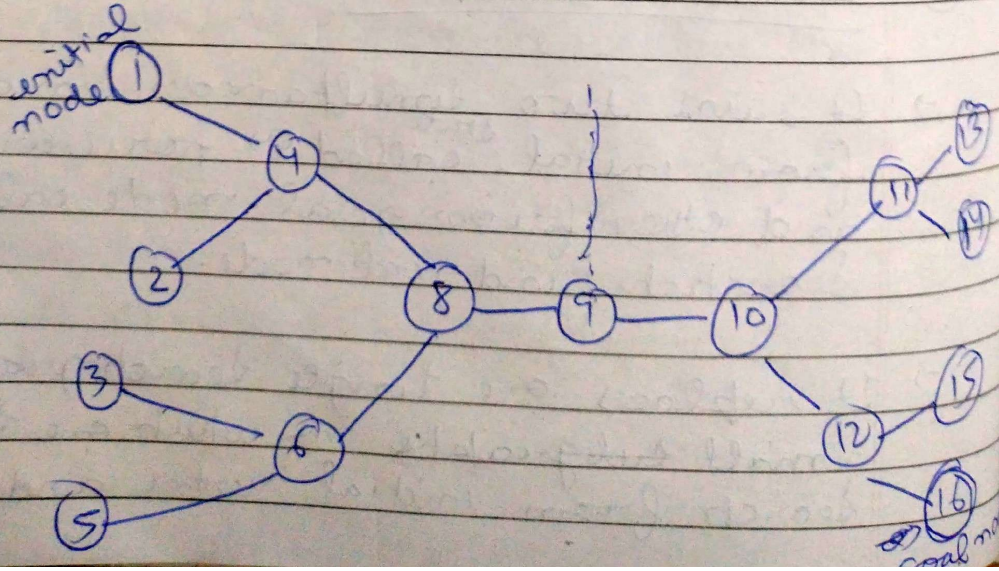
→ It requires less memory.

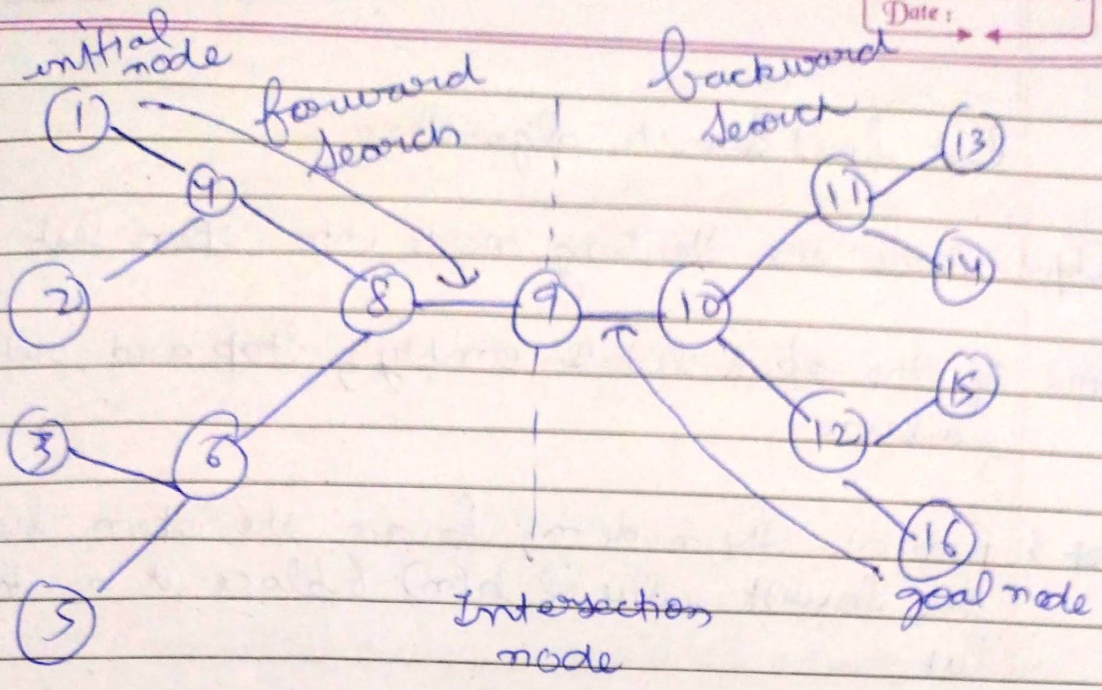
Disadvantages

→ Implementation of the bidirectional search tree is difficult.

→ In bidirectional search, one should know the goal state in advance.

Example





→ It is complete (If uses BFS in both searches)
→ It is optimal

Informed search algorithm

① Best - first search (Greedy search) $h(n)$ — no. of nodes

→ It always selects the path which appears best at that moment.
→ It is a combination of depth first search (DFS) and Breadth first search (BFS)
→ It uses the heuristic function $h(n) <= h^*(n)$

$h(n)$ = heuristic cost
$h^*(n)$ = estimated cost

→ The greedy best first algorithm is implemented by priority queue.

Date :

Best first Search algorithm :-

Step 1: Place the starting node into open list

Step 2: If the open list is empty, stop and return failure.

Step 3: Remove the node n, from the open list which has lowest value of h(n) & place it in the closed list.

Step 4: Expand the node n and generate the successors of node n.

Step 5: Check each successor of node n and find whether node is a goal node or not. If any successor is a goal node, then return success & terminate the search, else proceed to Step 6

Step 6: For each successor node, algorithm checks for evaluation function F(n), & then check if node has been in either Open or Closed list. If the node has not been in both list, then add it to the OPEN list.

Step 7: Return to Step 2

Advantages

→ Best first search can switch between BFS & DFS by gaining advantages of both the algorithm.

→ This algorithm is more efficient than BFS and DFS algorithm.

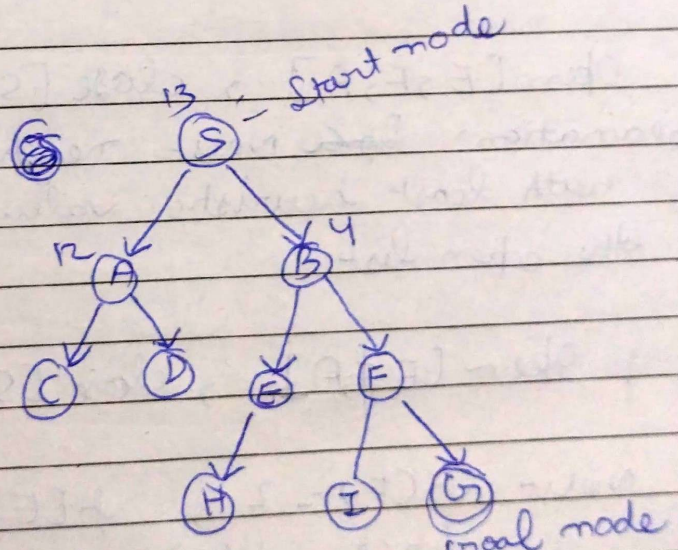
Disadvantages

→ It can behave as an unguided depth-first search in the worst case scenario.

→ It can get stuck in a loop as DFS.

→ This algorithm is not optimal.

Example



Heuristics given

node	h(n)
S	13
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
G	0

(Note: Final nodes should be placed in closed list)

(The node with less h value will be placed in closed list)

open ; close

S → Initialization
open [A, B] ; close [S]
check with A
(I) open [A]

S → Initialization
open [A, B] ; close [S]

(1) open [A, B] ; close [S, B]

h(A) = 12 , h(B) = 4

Explanation: Since h(B) < h(A) i.e 4 < 12
Remove B from the open list and place it in the closed list.

(2) Open [E, F, A] ; close [S, B]

Explanation: Now neighbours of B (the node with least heuristic value) will be added to the open list.

Open [E, F, A] ; close [S, B, F]

Now h(F) = 2 , h[E] = 8
Since h(F) < h[E] ie 2 < 8
Remove F from the open list and add it to the closed list.

(3) open [E, A, I, G] ; close [S, B, F]

Explanation: Now add successors of F ie I and G

Page:
Date:

Open [E, A, I, G] ; Close [S, B, F, G]

Now ~~Since~~ h[I] = 9 , h(G) = 0
Since h(G) < h(I) i.e 0 < 9
∴ ~~add~~ Remove G from the open list and add it to the closed list

∴ Path will be
S → B → F → G

→ Time complexity ⇒ T(n) = O(b^m)
S(C) = O(b^m)
where m is max depth of search space

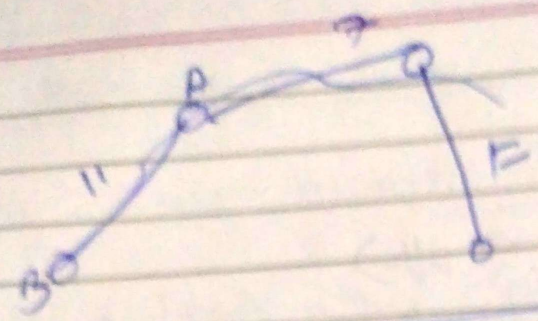
→ Best first search is incomplete & not optimal

② A* Search Algorithm

→ A* search algorithm finds the shortest path through the search space using the heuristic function.

→ It uses h(n) and cost to reach the node n from the start state g(n)
F(n) = g(n) + h(n)

→ This algo. expands less search time and provides optimal results faster.



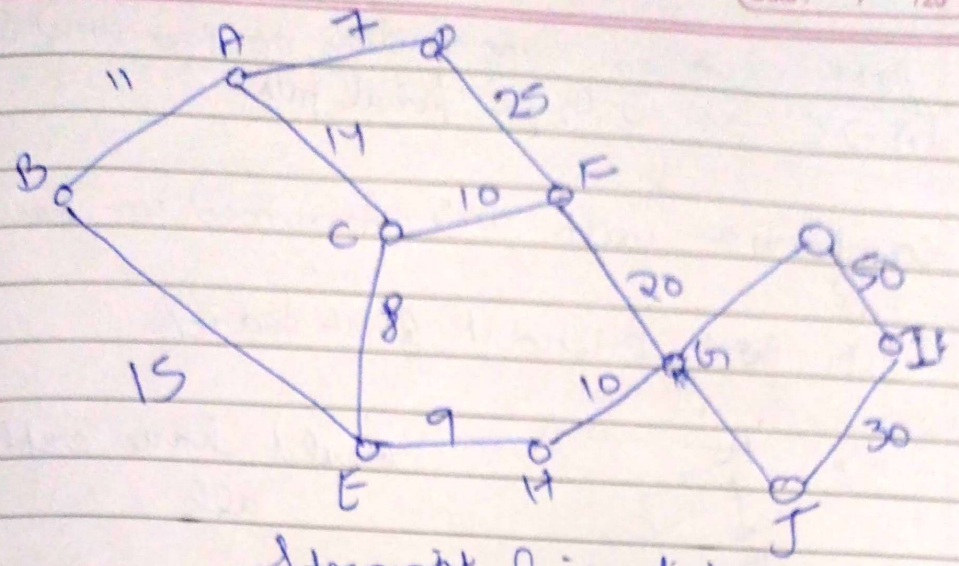
Informed Search Algorithms

(1) Best First Search (Informed, Heuristic)

Algorithm

→ Let OPEN be a priority queue containing initial state

Loop
if OPEN is empty return failure
NODE ← Remove-First (OPEN)
if Node is a goal
then return the path from initial to Node
else generate all successors of Node and
put the newly generated node into OPEN
according to their f values
End loop

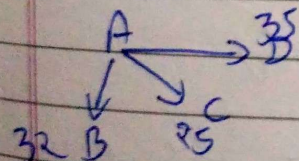


Straight line distance.

A → G = 40
B → G = 32
C → G = 25
D → G = 35
E → G = 19
F → G = 17
H → G = 10
G → G = 0

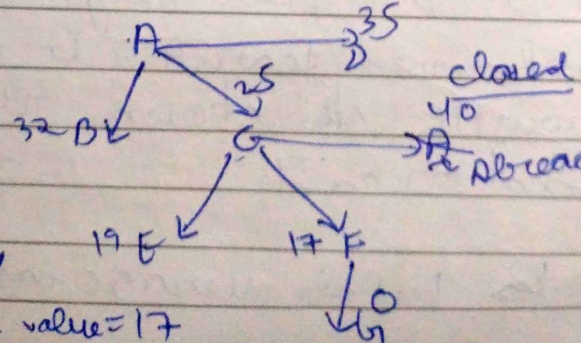
(Already given in question)

Priority → the one with least heuristic value is the best and explore that node



Successors of A → B, C, D
Put B, C & D in queue based on heuristic value

A, F, B, D
We need to explore C



Now successors of C
→ E, F, A already explored
Pick F with heuristic value = 17

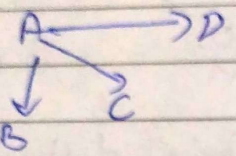
Evergreen
Page No.
Date : / /20

Path chosen according to Best First Search

⇒ A → C → F → G ← final path

Comparison with uninformed or heuristic

like Breadth first search



would have explored B, C, D all

complexity $O(b^d)$

→ Best first search - uses greedy method based on heuristic value

→ Time complexity is reduced

→ It always gives the good solution

→ It may or may not give an optimal solution

→ Much better than uninformed method with polynomial time complexity

→ In worst case, complexity can be equal to $O(b^d)$, it may behave as uninformed search. It depends upon the heuristic we chosen, whether heuristic chosen is good or bad

→ In best or average case, its complexity is

(2) A* Search Algorithm

→ A* search algorithm finds the shortest path through the search space using the heuristic function.

→ It uses $h(n)$ and cost to reach the node n from the start state $g(n)$

$$f(n) = g(n) + h(n)$$

→ This algo. expands less search tree and provides optimal results faster.

g(n) =

Page :
Date :

→ It is similar to UCS except that it uses $g(n)+h(n)$ instead of $g(n)$

→ A* uses search heuristic as well as the cost to reach the node. Hence we combine both costs as

$$F(n) = g(n) + h(n)$$

(fitness number)

$g(n)$ = cost from source node to a particular node

$F(n)$ = Estimated cost of the cheapest solution

$g(n)$ = cost to reach node n from start state

$h(n)$ = cost to reach from node n to goal node

Algorithm of A* search

Step (1) Place the starting node in OPEN list.

Step (2) Check if the OPEN list is empty or not, if the list is empty, return failure and stop.

Step (3) Select the node from OPEN list which has the smallest value of evaluation function $(g+h)$. If node n is a goal node, then return success & stop. Otherwise,

Step (4) Expand node n and generate all of its successors and put n in the closed list.

- For each successor 'n', check whether n is already in the OPEN and CLOSED list.
- If not then compute evaluation function for 'n' and place it in OPEN list.

Step 5: Else if node 'n' is already in open and CLOSED then it should be attached to the back pointer which reflects the lowest g(n') value.

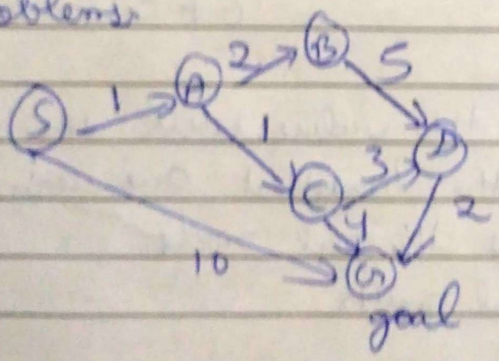
Step 6: Return to Step 2

Advantages

(1) It is best algorithm than other search algorithms
(2) It is optimal & complete.
(3) It can solve very complex problems

Disadvantages

- It does not always produce shortest path.
- It is not practical for various large-scale problems.



Given:

State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Page :
Date :

① $S \to A \Rightarrow F(n) = g(n) + h(n) \Rightarrow g(n) + h(A)$

$= 1 + 3 = 4$

$S \to G \Rightarrow F(n) = g(n) + h(n) \Rightarrow g(n) + h(G)$

$= 10 + 0 = 10$ (hold) ✗

② $S \to A \to B \Rightarrow F(n) = g(n) + h(n)$

$= (1+2) + g(B)$

$\Rightarrow 3 + 4 = 7$ (hold) ✗

$S \to A \to C \Rightarrow F(n) = g(n) + h(n)$

$= (1+1) + h(C)$

$\Rightarrow 2 + 2 = 4$ ✓

③ $S \to A \to C \to D \Rightarrow F(n) = g(n) + h(n)$

$= 1 + 1 + 3 + g(D)$

$= 5 + 6 = 11$ hold ✗

$S \to A \to C \to G \Rightarrow F(n) = 1 + 1 + 4 + h(G)$

$= 6 + 0 = 6$ ✓

Now compare this value with the values on hold. Since this is the smallest one among other and also contains the goal node.

∴ It is the final path

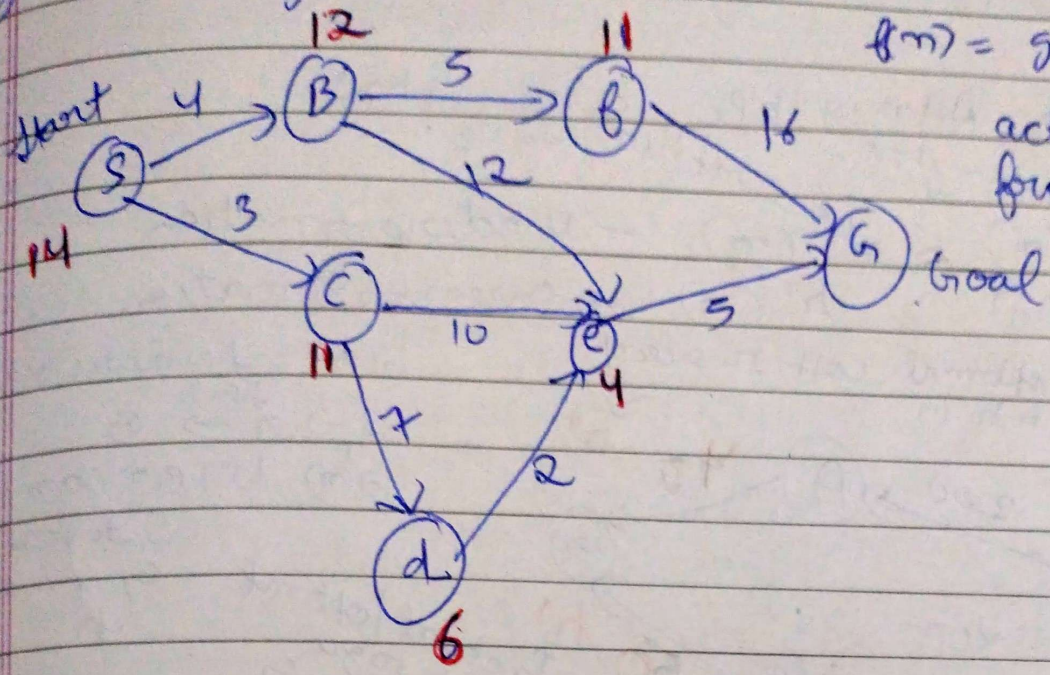
$S \to A \to C \to G$

Cost = 6

③ → AO* Algorithm is a heuristic search algorithm

→ uses the concept of AND-OR graphs to decompose any complex problem given into smaller set of problems which are further solved.

(2) A* Algorithm



$f(n) = g(N) + h(N)$

↓ actual cost from start node to n node

Now we have to reach from S to G

$f(S) = 0 + 14$ $(g(n) + h(n) =$

$= 14$

$S \to B$

$= 4 + 12$

$= 16$

$S \to C$

$= 3 + 11$

$= 14$

(Choose the one with minimum cost)

$SB \to f = 4+5+11 = 20$

$SB \to e = 4+12+4 = 20$

$SBf \to G = 4+5+16+0$

$= 25$

$SC \to e = 3+10+4 = 17$

$SC \to d = 3+7+6 = 16$ ✓

$SCd \to e \Rightarrow 3+7+2+4$

$= 16$ ✓

$SCde \to G \Rightarrow 3+7+2+5+0$

$= 17$

choose the minimum. Goal state

Time complexity = $O(b^d)$ (dekho)
↳ branch factor
(no. of children of a particular node)

Space complexity = $O(b^d)$

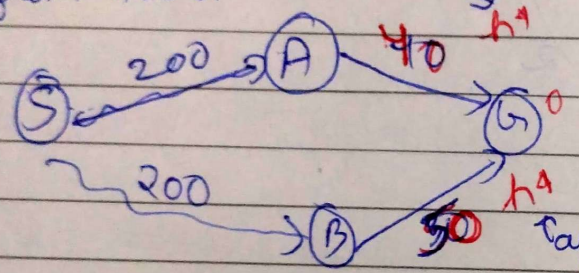
⇒ A* is Admissible algorithm

estimated value / actual value

$h(n) \leq h^*(n)$ — underestimation

$h(n) \geq h^*(n)$ — overestimation

actual/optimal cost to reach from A to G



$h(n)$ = heuristic value

$S \xrightarrow{g(n)} n \xrightarrow{h(n)} G$

$g(n)$ ↳ i.e. n can take S to goal state G in cost h

actual/optimal cost to go from B to G

S → Start State G → Goal State

A & B → intermediate node

$h(A) = 20$

$h(B) = 200$

$g(A) = 200$

$g(B) = 200$

Case I overestimation

estimated value: $h(A) = 80$, $h(B) = 70$ > actual value h^*

$f(n) = g(n) + h(n)$

(in this case $h^*(A) = 40$, $h^*(B) = 50$)

$f(A) = g(A) + h(A)$

$f(A) = 200 + 80 = 280$; $f(B) = g(B) + h(B)$

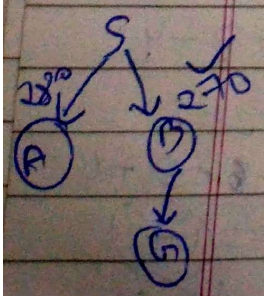
$f(B) = 200 + 70 = 270$ (Minimum)

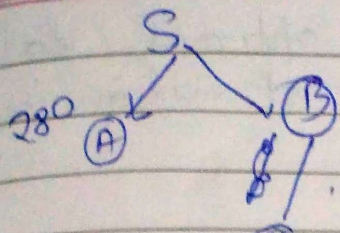
$f(G) = g(G) + h(G)$

$= 250 + 0$ ∵ $h(G) = 0$ (heuristic is assumed to be 0 to reach goal)

$= 250$

$g(G) = 200 + 50 = 250$ (via B)



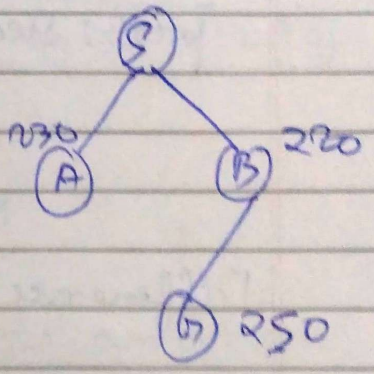


→ this is the minimum cost, so it will stop over here.

$f(G) = g(G) + h(G)$
$= 200 + 50 = 250$

case 2 underestimation

$h(A) = 30$
$h(B) = 20$
$f(A) = g(A) + h(A)$
$f(A) = 200 + 30 = 230$
$f(B) = g(B) + h(B)$ $f(B) = 200 + 20 = 220$ ✓



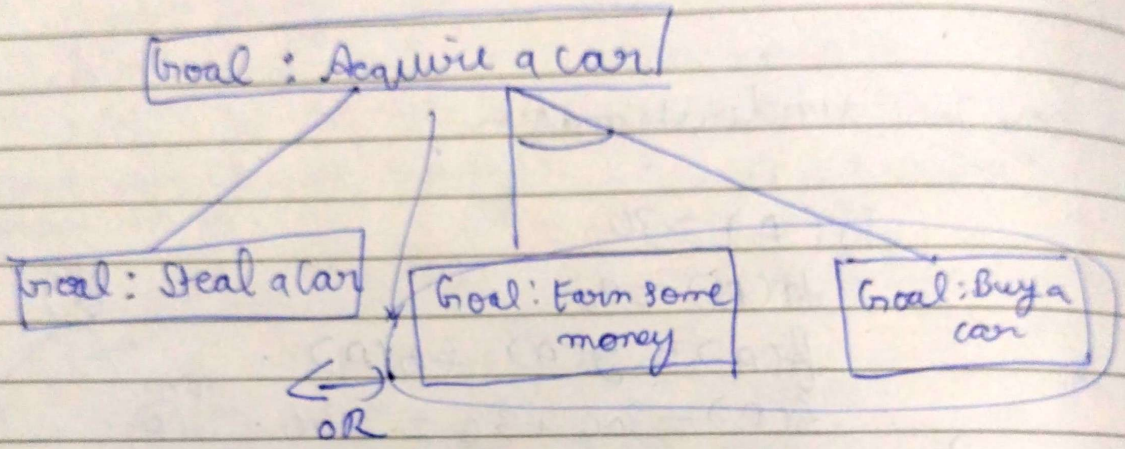
via B
$f(G) = g(G) + h(G)$
$= 250 + 0 = 250$

Now it will not stop as state (A) has (230) value lesser than 250

Now $f(G)$ will be computed again
via A
$f(G) = g(G) + h(G)$
$= (200 + 40) + 0$ $f(G) = 200 + 40$
$f(G) = 240$ This is the optimal value to reach the goal state

Underestimation leads to optimal solution always but overestimation may not

③ AO* (And / OR) (And-OR graphs) → Problem decomposition (Breakdown into small pieces)



Difference between A* & AO*

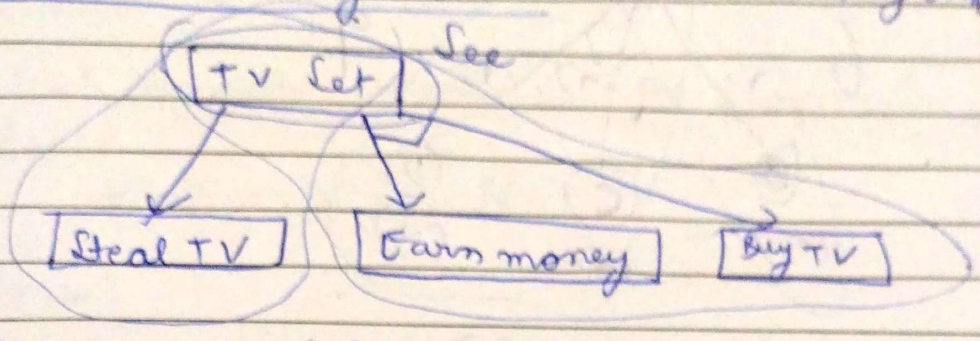
Similarity
→ Both work on strategy of Best First Search
→ Both are informed search techniques and work on the basis of heuristic value

Difference
→ A* always give the optimal solution
→ AO* is not guaranteed to give an optimal solution (* means admissible)
→ AO* does not explore all the solution paths once it got a solution
→ A* always gives an optimal solution

Page :
Date :

AO* Search Algorithm (AND - OR) graph

Example



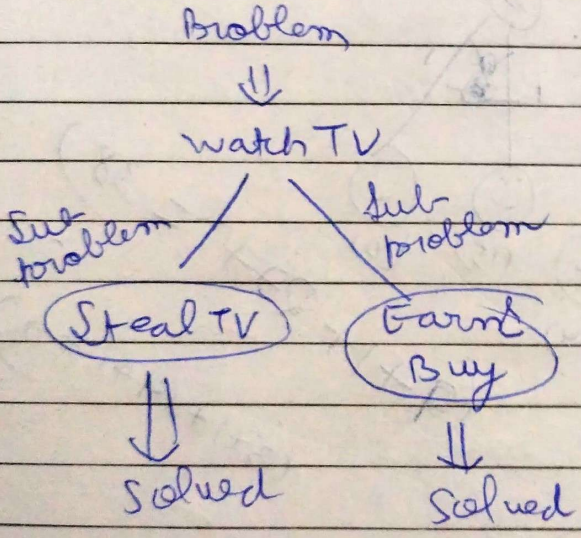
Earn money + Buy TV ⇒ watch TV

+ AND is represented by +

Arnd OR

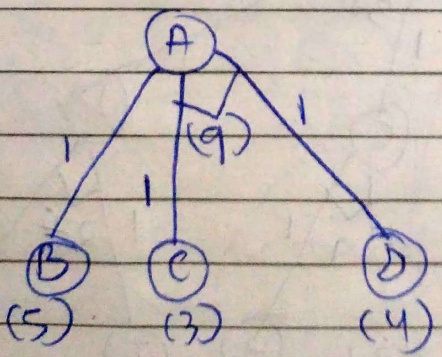
OR

Steal the TV ⇒ watch TV

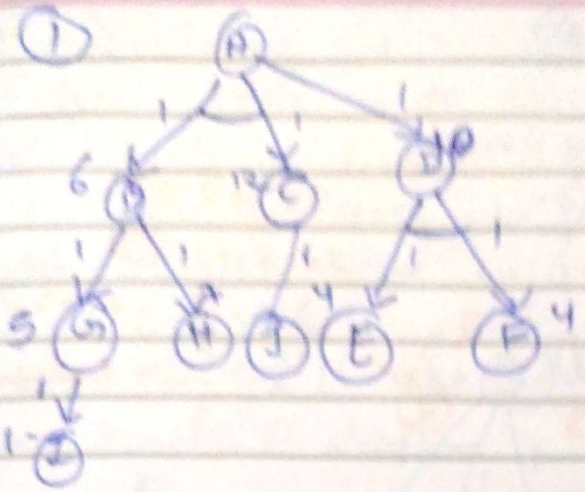


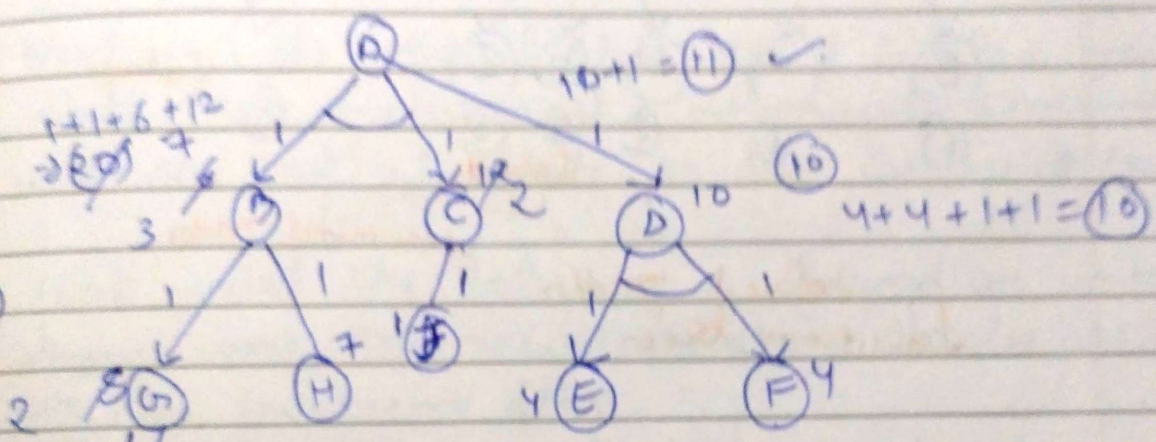
Each sub problem is solved separately.

Example



ex (1)



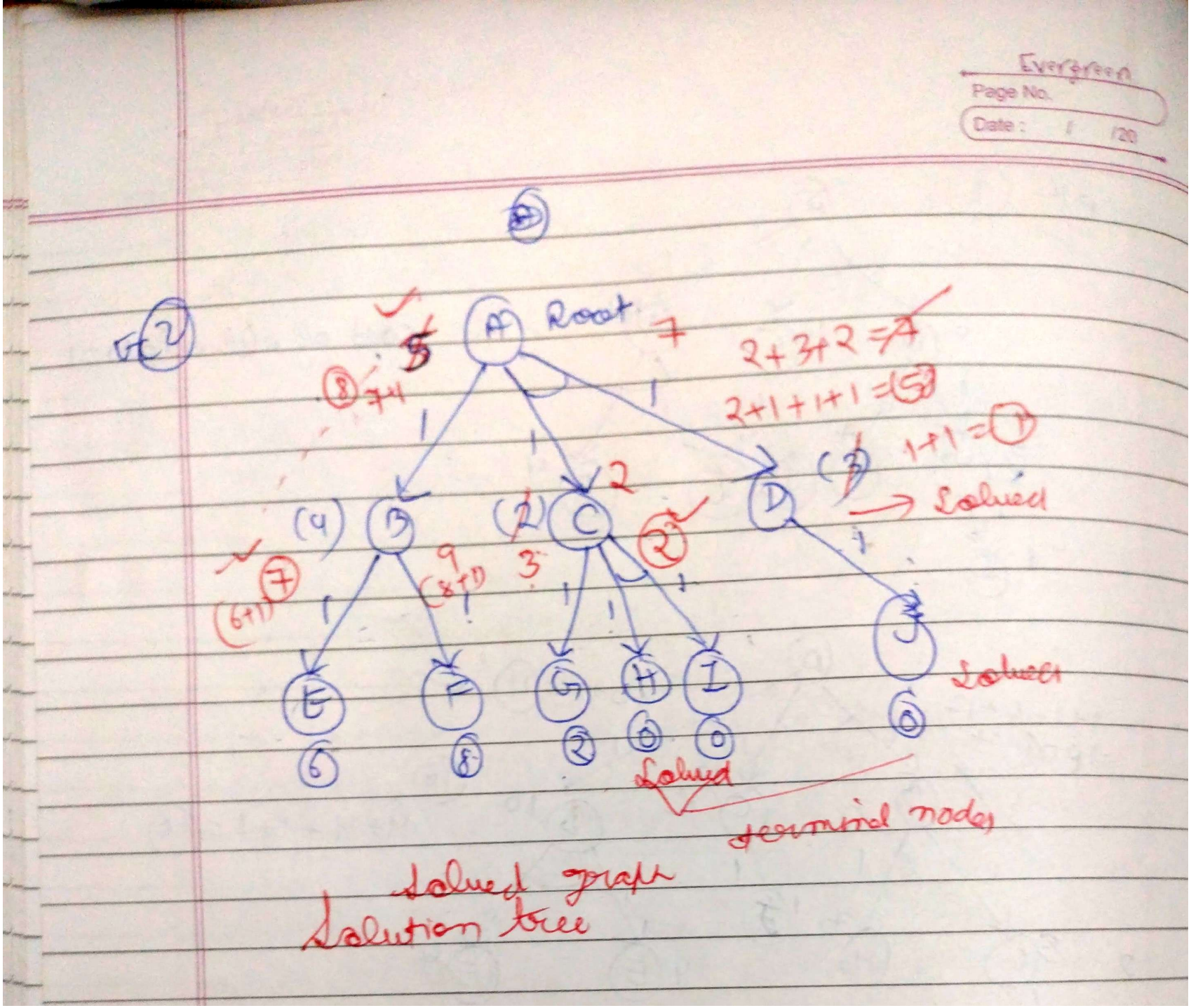


Now it will not explore another path.

Had it explored other path, it could provide better solution

let's explore For example

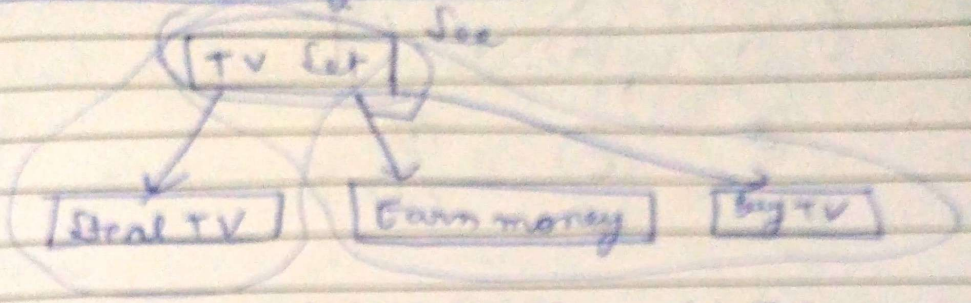
In A*, we need to calculate new estimation value at every step and estimate the move towards goal


A
Root
B
C
D
E
F
G
H
I
J
Solved
Solved
terminal nodes
solved graph
Solution tree

AO* algorithm

AO* Search Algorithm (AND - OR) graph

Transcribe



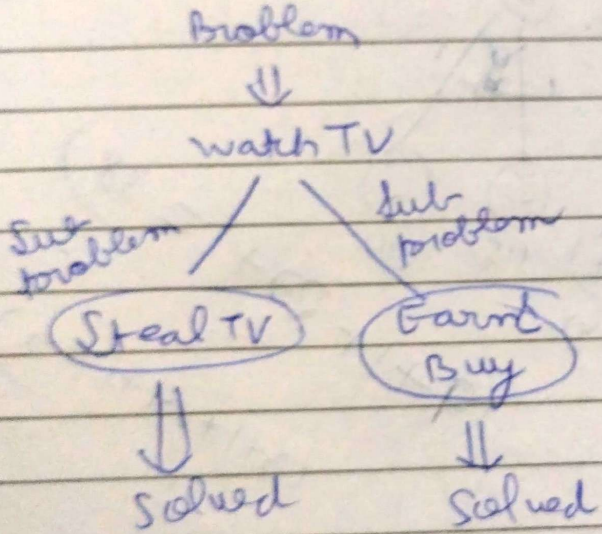
Earn money + Buy TV ⇒ watch TV

(+ AND is represented by +)

AND OR

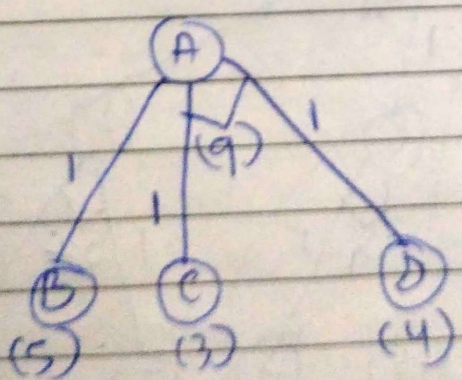
OR

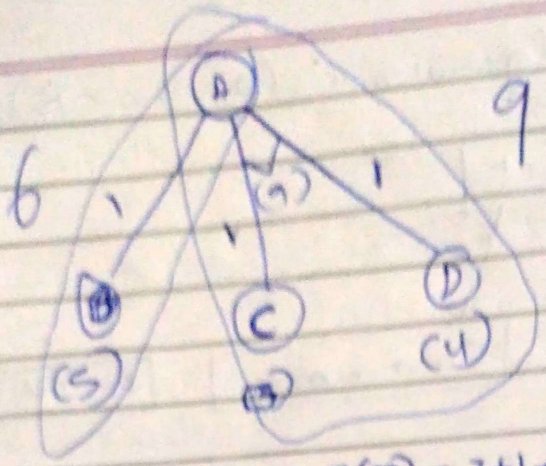
Steal the TV ⇒ watch TV



Each sub problem is solved separately

Example

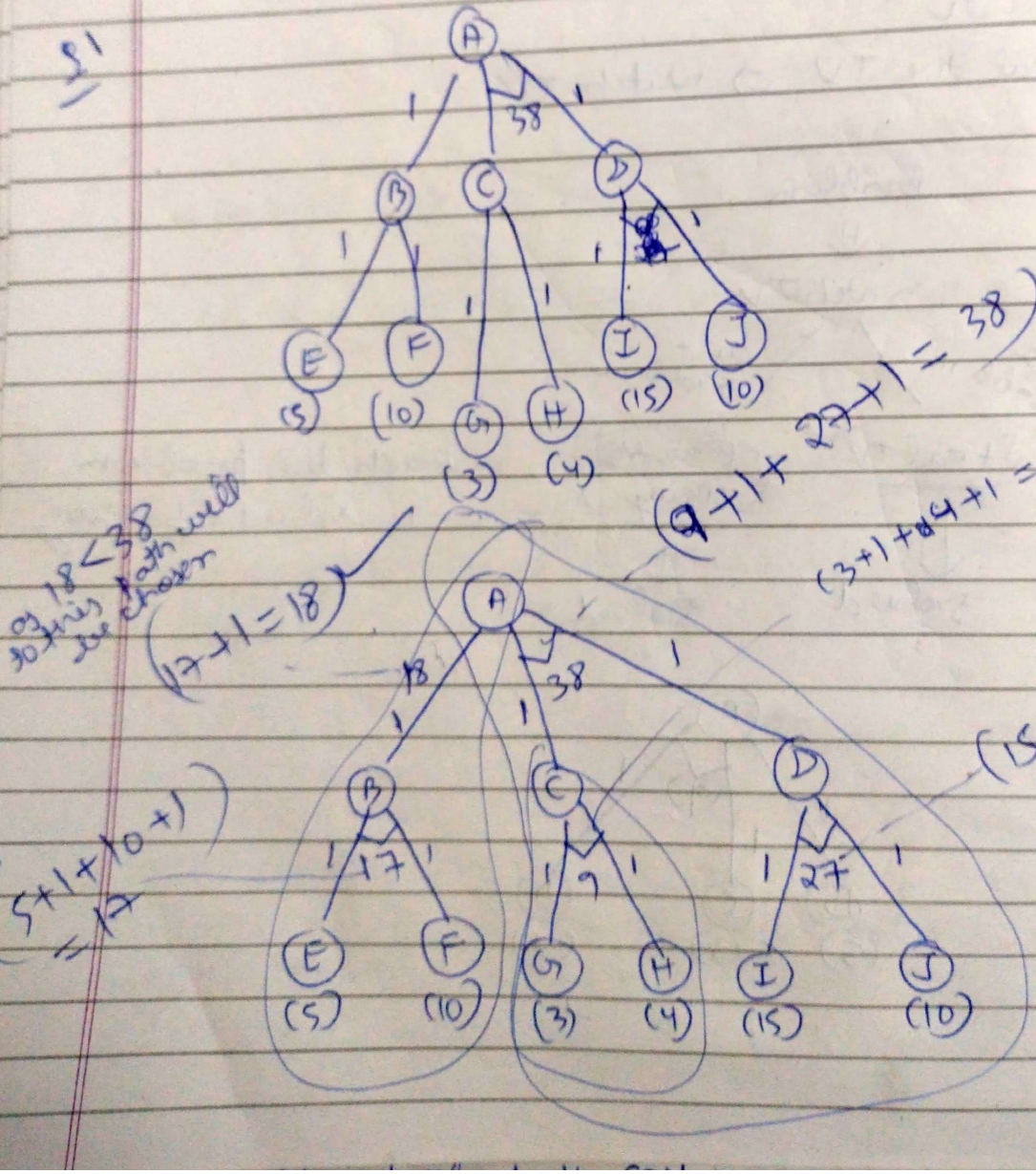




F(m) = 5+1
= 6

F(m) = 3+1+4+1
= 9

Q1



working of AO* algorithm

→ The AO* algorithm works on the formula given below:

$$f(n) = g(n) + h(n)$$

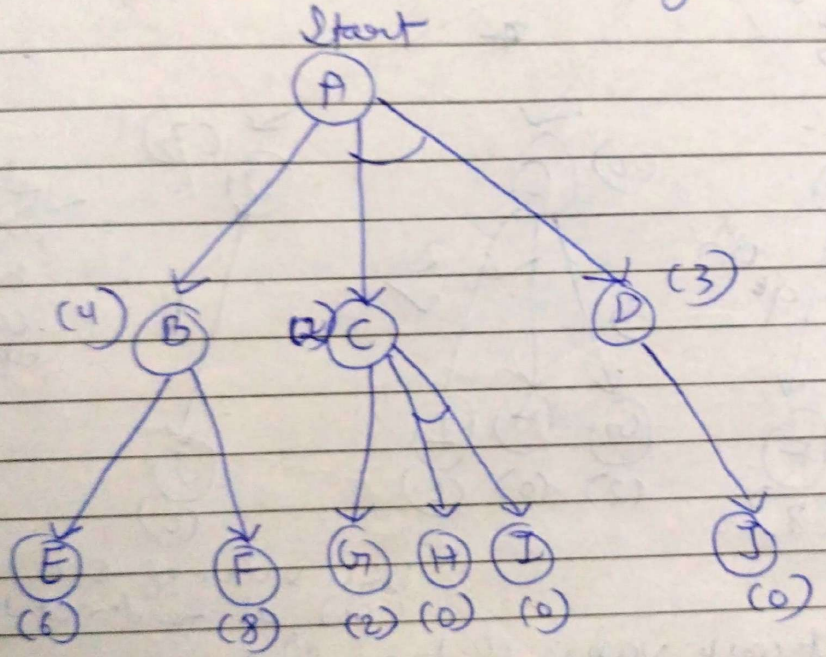
where

$g(n)$ - The actual cost of traversal from initial state to the current state.

$h(n)$ - The estimated cost of traversal from the current state to the goal state.

$f(n)$ - The actual cost of traversal from initial state to the goal state.

Q2 →



→ Here, in the above example, all the numbers in brackets are the heuristic values i.e $h(n)$

→ Each edge is considered to have a value of 1 by default.

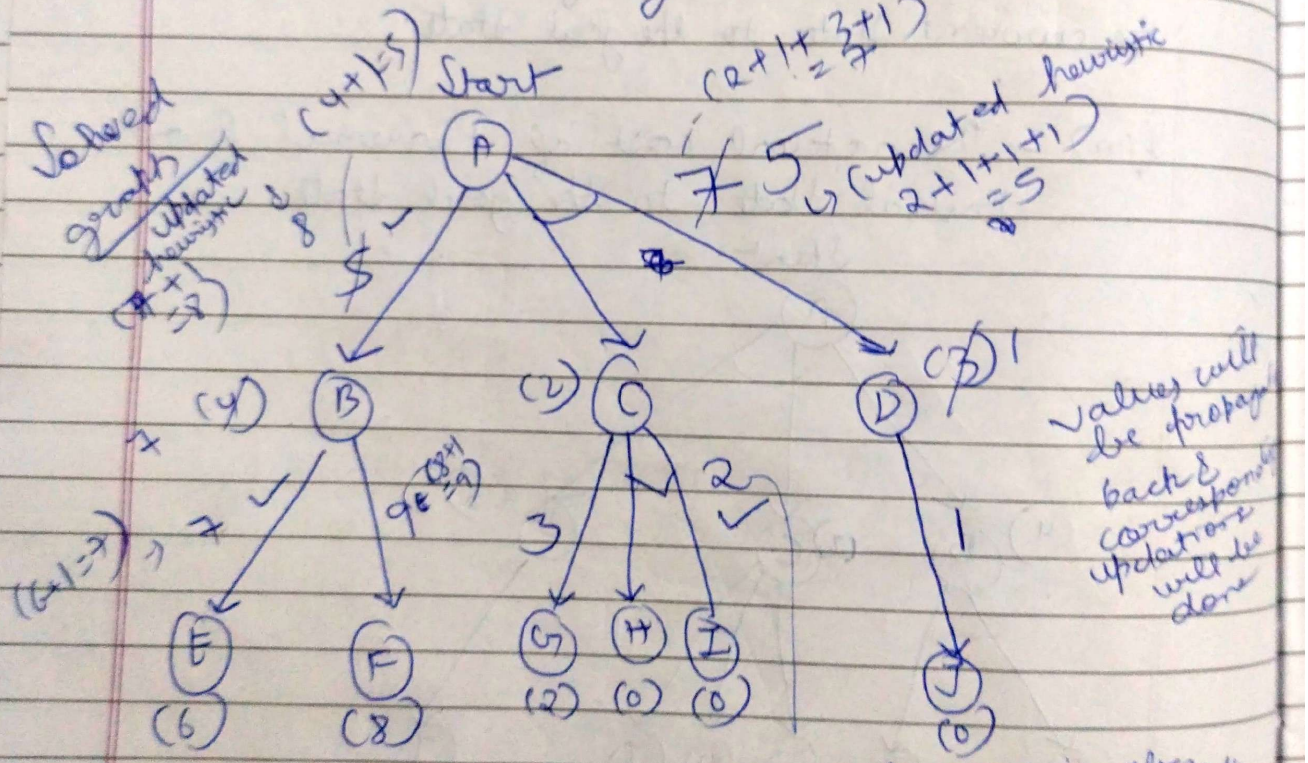
Step 1 : Starting from node A, we first calculate the best path.

$f(A-B) = g(B) + h(B) = 1 + 4 = 5$, where 1 is the default cost value of travelling from A to B and 4 is the estimated cost from B to Goal State.

$f(A-(C-D)) = g(C) + h(C) + g(D) + h(D)$

$= 1 + 2 + 1 + 3 = 7$

Here, we are calculating the path cost as both C and D because they have the AND-Arc.



The default cost value of travelling from A-C is 1 and from A-D is 1 but the heuristic value given from C & D are 2 and 3 respectively hence making the cost as 7.

Page :
Date :

Step 2 Now cost of path A-B is lesser (i.e 5) than cost of path A-(-) (i.e 7) hence this path will be chosen

from the B node,

$f(B-E) = 1+6 = 7$
$f(B-F) = 1+8 = 9$

→ Hence B-E path has lesser cost. Now the heuristics have to be updated since there is a difference between actual and heuristic value of B.

→ The minimum ~~path~~ cost path is chosen and is updated as the heuristic, in our case the value is 7.

→ And because of change in heuristic of B there is also change in heuristic of A, which is to be calculated again

$f(A-B) = g(B) + \text{updated } (h(B)) = 1+7 = 8$

Step 3: Now the current node becomes C node & the cost of the path is calculated.

$f(C-G) = 1+2 = 3$
$f((-H-I) = 1+0+1+0 = 2$

$f((-H-I)$ is chosen as minimum cost path

→ Heuristic of path H and I are 0 and hence they are solved

→ But path A-D also needs to be calculated, since it has an AND arc.

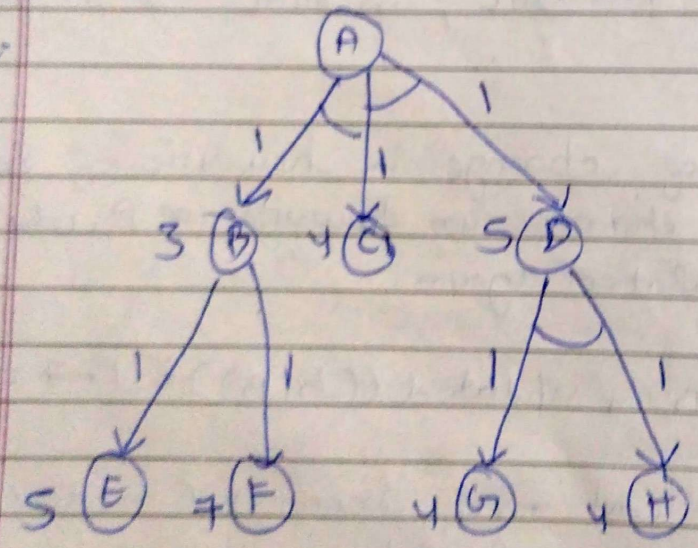
→ f(D-J) = 1+0 = 1, hence heuristic of D needs to be updated to 1

→ finally, f(A-(C-D)) needs to be updated.

f(A-(C-D)) = g(C) + h(C) + g(D) + updated (Ch(D))

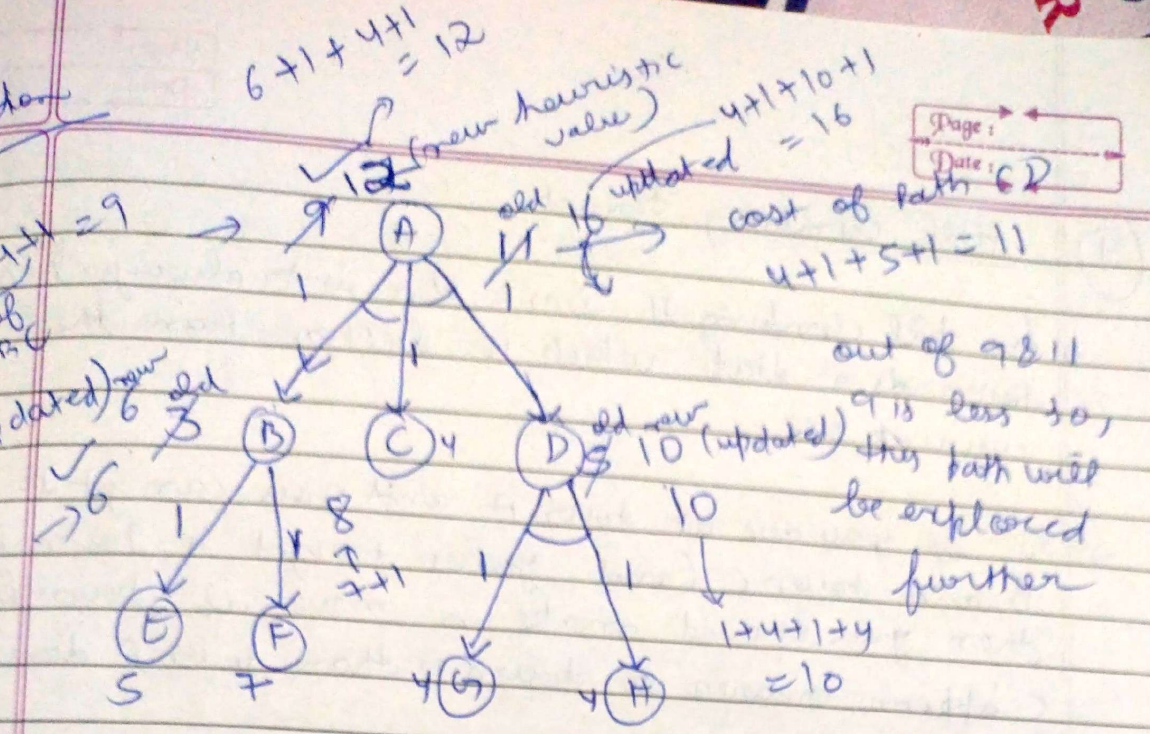
= 1+2+1+1 = 5

93.



In order to solve A, solve B and C or solve C & D

Solution



cost of path 6 2

$4+1+5+1 = 11$

out of 9 & 11 9 is less so, this path will be explored further

Now inorder to solve B either path BE or BF can be considered as they are not connected through AND

Now after updation of heuristic value as 12 of path A-B-C, heuristic value of path ACD seems to be less (ie 11) so it will be explored further

After solving this graph

Cost of path A-B-C = 12 ✓

Cost of path A-C-D = 16

optimal value will be lesser one ie with path A-B-C = 12 (so it will be chosen)

A

B C

E

(optimal path)

Page :
Date :

(4) Hill climbing Search

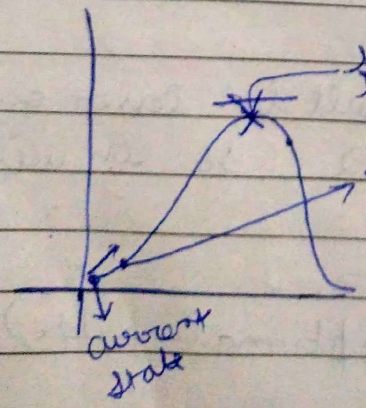
→ In hill climbing the basic idea is to always head towards a state which is better than the current one.

→ So, if you are at town A and you can get to town B and town C (and your target is town D) then you should make a move if town B or C appear nearer to town D than town A does.

two possibilities i.e
- A → B → D
- or
- A → C → D

} choose the path which is appears better than the current state

Hill climbing Search (key points)
→ Local search algorithm → always moves in a single direction
→ Greedy approach
→ No backtracking
i.e if a node is selected during the searching process, the same node will expanded further. We can't do backtracking and can not select any other node in between.



(as the next state will not be better than the current state)

If we are climbing over a hill, upward state is considered better than the current state.

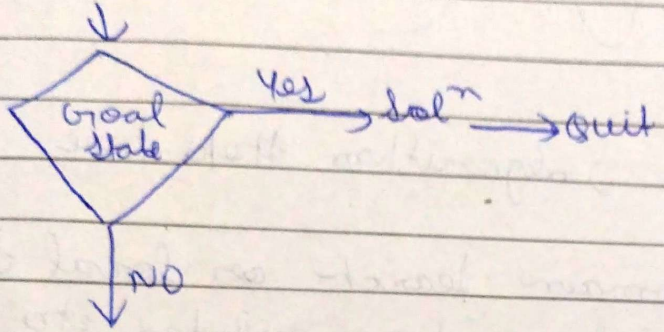
If new state is better than current state,

then

new state = current state

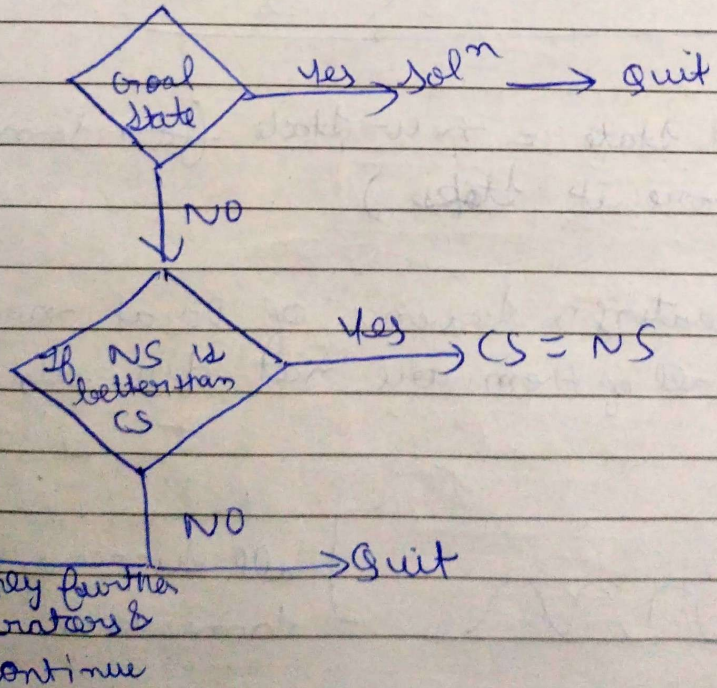
If new state is not better than the current state, then stop

Evaluate initial state



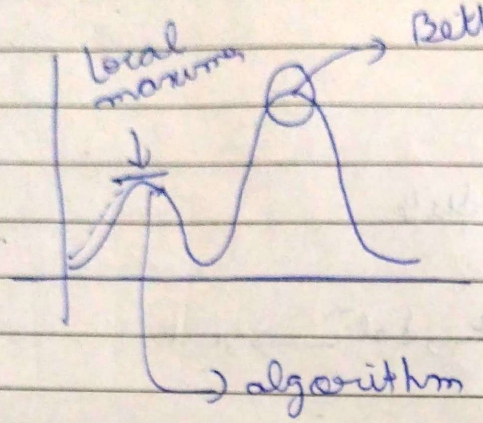
current state (CS) = Initial state

Apply operator 'O' and get new state (NS)



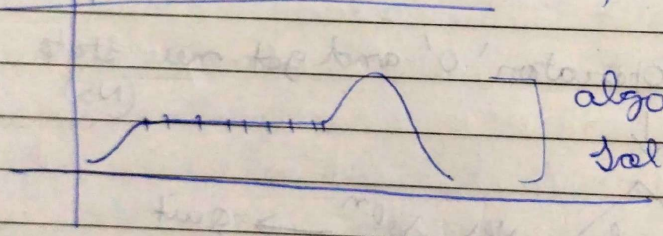
Limitations of hill climbing

(1) Local maxima :- → State which is better than all of its neighbours but not better than some other state which is not directly connected to that state



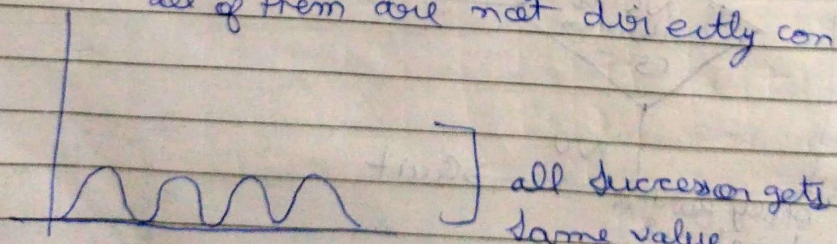
also known as Local domain search or local search as it has information related to local domain only.

(2) Plateau / Flat maximum → A flat area of the search space in which all the neighboring states have same values

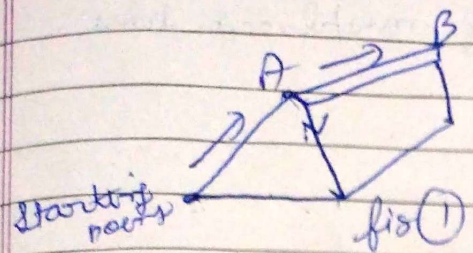


(current state = new state for some time, therefore it stops)

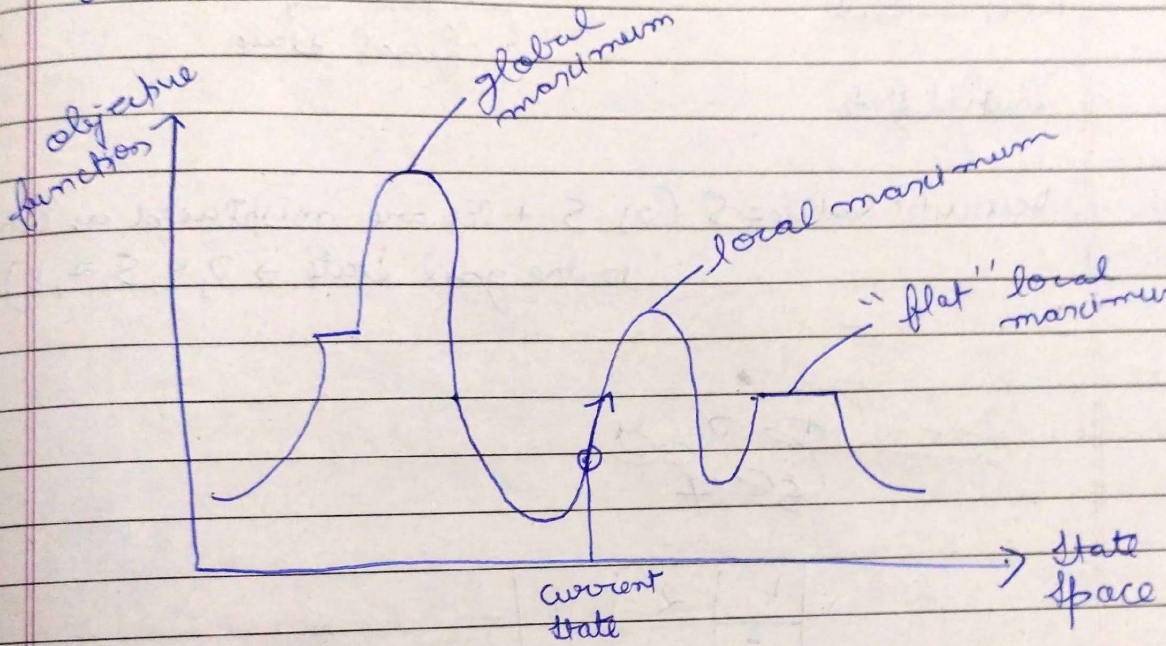
(3) Ridge (creating a sequence of local maxima but all of them are not directly connected



instead of path cost



Ex: In this fig ①, it is moving in upword direction towards A but after reaching at A, it has to stop. Although point B is above A, it can not go further because it can not change its direction.



Example Solve 8 puzzle problem using Hill climbing search

1	2	4
5		7
3	6	8

Initial state

1	4	7
2	5	8
3	6	

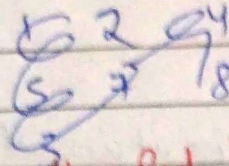
final state

Algorithm of hill climbing algorithm

[Any heuristic value which is not overestimate than its actual value is said to be admissible]

Page :
Date :

h1 = the number of misplaced tiles



Text with Red is representing the misplaced tiles

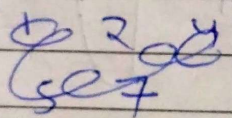
1	2	4
5		7
3	6	8

Initial State

1	4	7
2	5	8
3	6	

Final State

Heuristic value = 5 (as 5 tiles are misplaced as compared to the goal state → 2, 4, 5, 7, 8)



1	2	4
5		7
3	6	8

up

down

left

Right

Red color represents heuristic (no of misplaced tiles)

1		4
5	2	7
3	6	8

H = 5

1	2	4
5	6	7
3		8

H = 6

1	2	4
	5	7
3	6	8

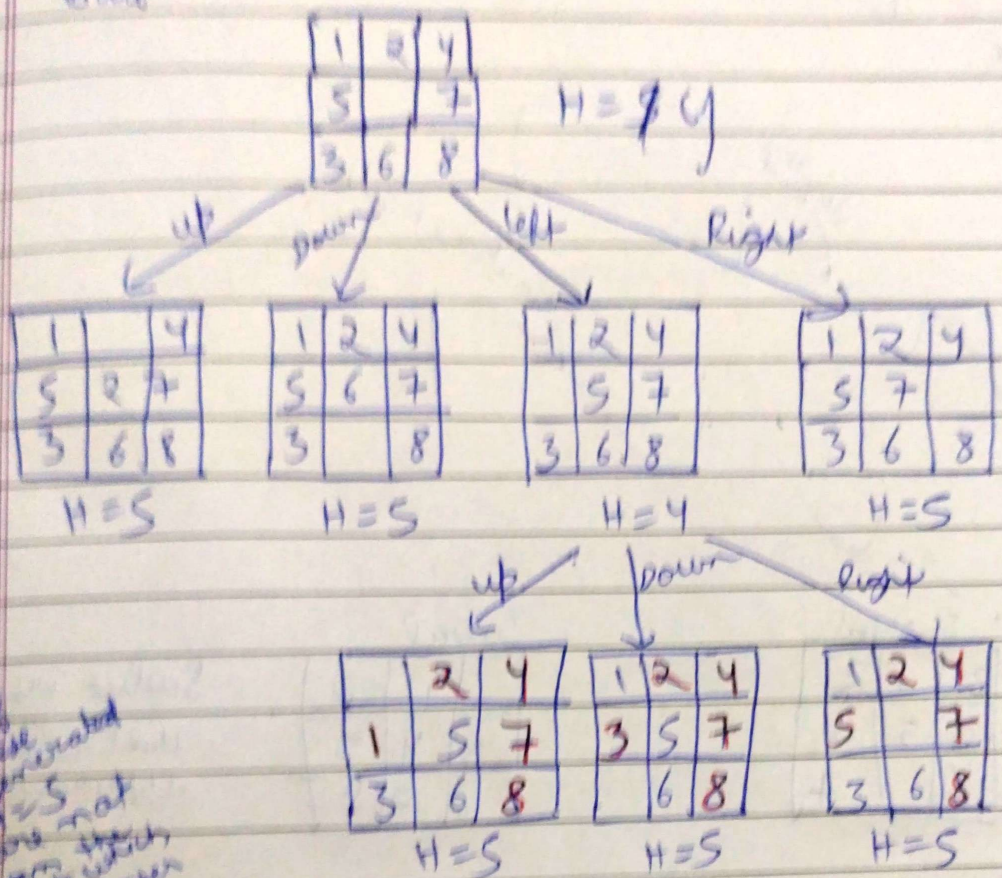
H = 4

1	2	4
5	7	
3	6	8

H = 5

Page :
Date :

Now state with H = 4 is better than the current state so it is more closer to the goal state therefore, it will be updated as the current state.

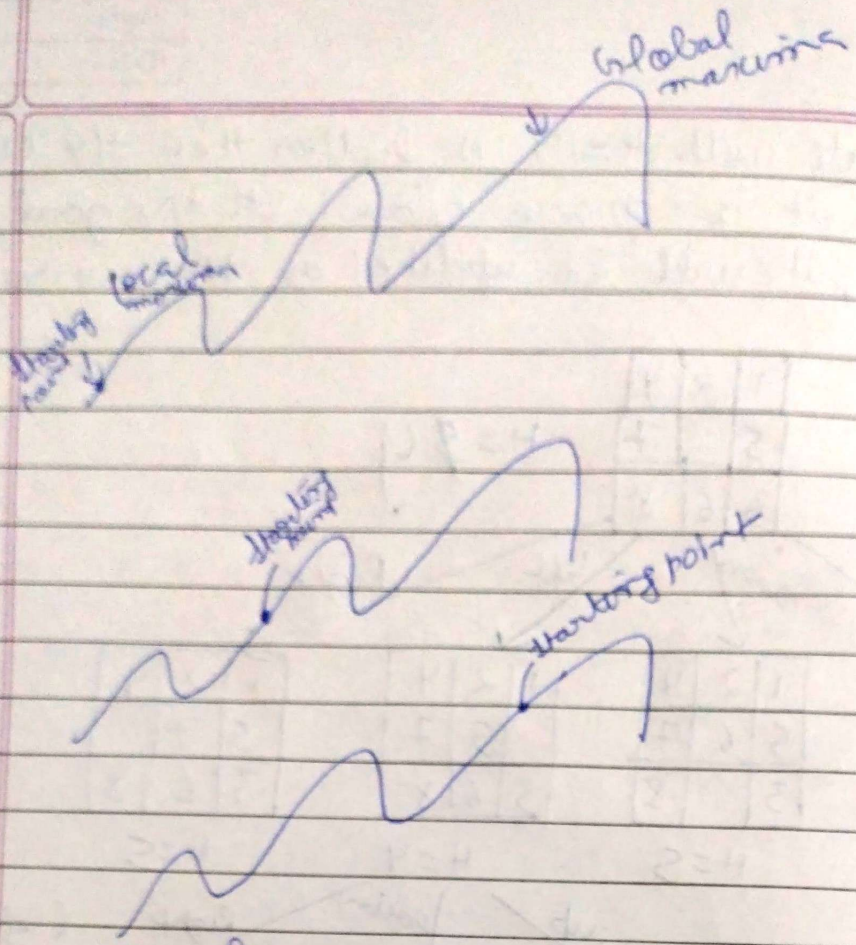


(All these generated now already generated have H=5 so they are not better than the current state which is 4 (one closer to the goal state) so the goal state will not be updated)

∴ This is a kind of local maxima i.e in every direction the solution generated is inferior than the current solution

Solution of hill climbing weaknesses

- Random initialization of starting state
- Jump



ex2

Initial

3	1	2
4	5	8
6		7

Goal

	1	2
3	4	5
6	7	8

Solve using Hill climbing approach

heuristic $h(x)$ = no. of misplaced tiles, we have to minimize the value of $h(x)$

3	1	2
4	5	8
6		7

$h(1) = 5$

up

left

Right

3	1	2
4		8
6	5	7

$h(2) = 5$

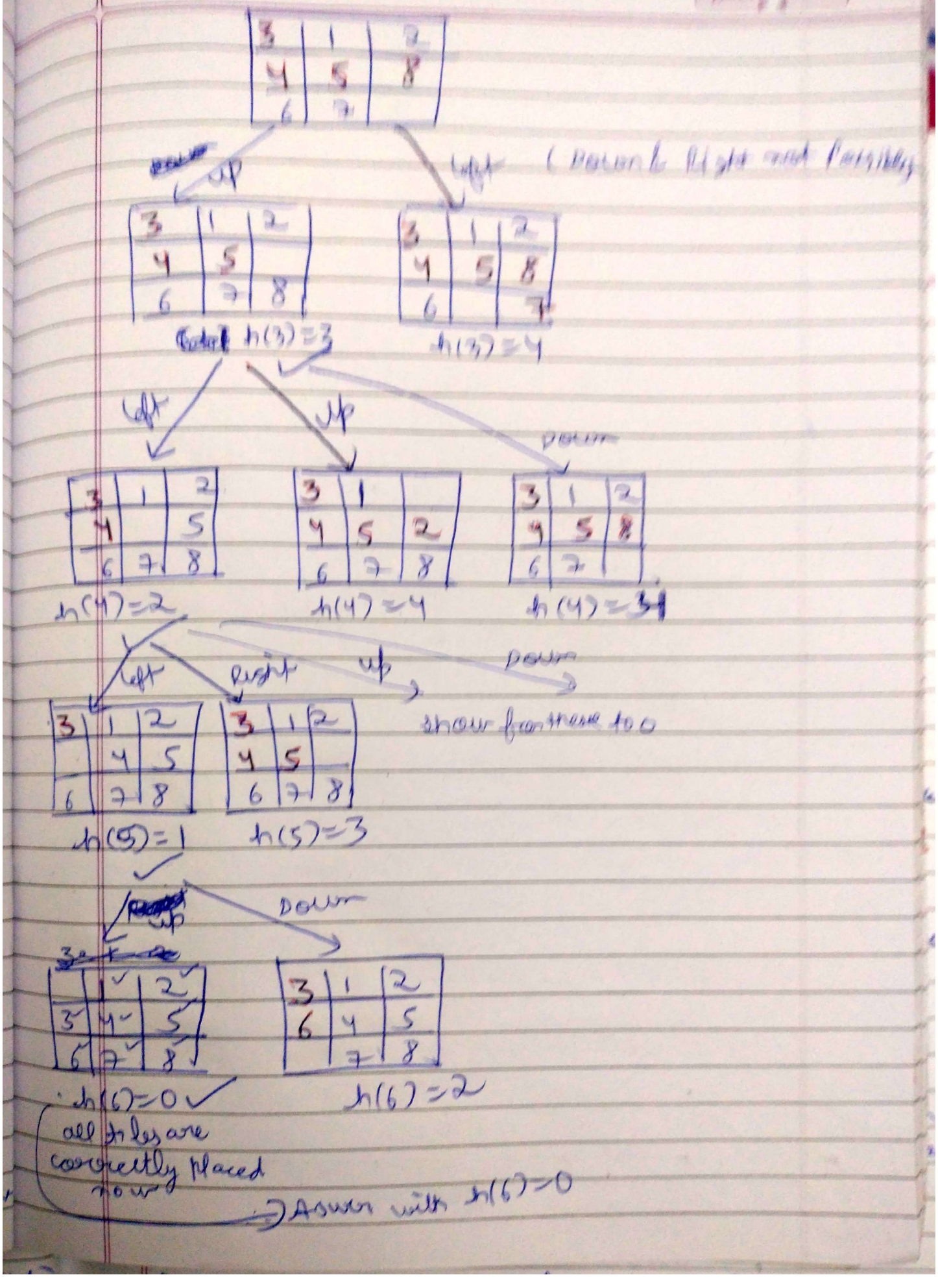
3	1	2
4	5	8
	6	7

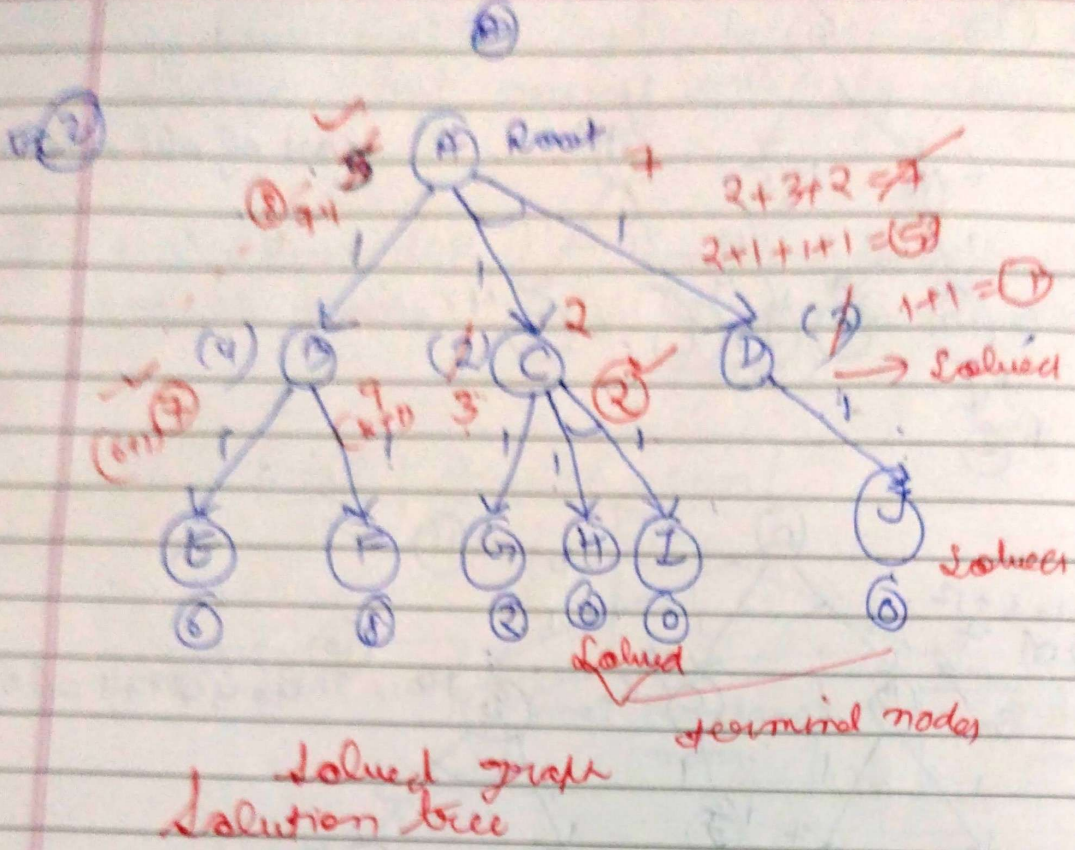
$h(2) = 6$

3	1	2
4	5	8
6	7	

$h(2) = 4$

better than current (more closer to the goal)


up
left
(Down & Right not possible)
h(3)=3
h(3)=4
left
up
down
h(4)=2
h(4)=4
h(4)=3
left
right
up
down
show from these to 0
h(5)=1
h(5)=3
up
down
h(6)=0
h(6)=2
all tiles are correctly placed now
Answer with h(6)=0



Hill climbing Algorithm

Local Search

→ Evaluate and modifying one or more current states rather than systematically exploring paths from an initial state

→ Good for those which requires only solution instead of path cost
→ requires less memory (save only current state)
→ move only to neighbors of that node (current node)

→ Hill climbing is a type of local search
→ The basic idea is to always head towards a state which is better than the current one
→ So, if you are at town A and you can get to town B and town C (and your target is town D) then you should make a move if town B or C appear nearer to town D than town A does.

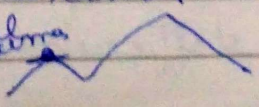
1. Evaluate the initial state. If it is also goal state then return it, otherwise continue with the initial state as the current state.

2. Loop until the solution is found or until there are no new operators to be applied in the current state
   a) Select an operator that has not yet been applied to current state and apply it to produce new state.

   b) Evaluate the new state
      i) If it is a goal state, then return it and quit
      ii) If it is not a goal state but it is better than the current state, then make it as a current state
      iii) If it is not better than the current state, then continue in the loop.

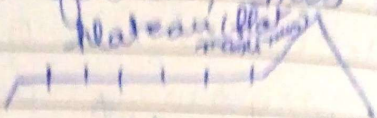
Drawbacks of hill climbing algorithm

1) Local maxima - A local maximum as opposed to global maximum

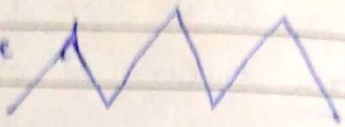


Some answer from all the possible moves

(i) Plateaus - An area of search space where evaluation function is flat, thus requiring random walk.



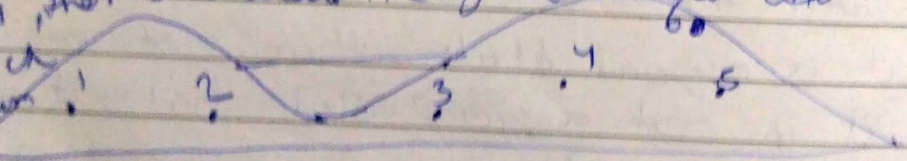
(ii) Ridge - where there are steep slopes, and the search direction is not towards top but towards the side



→ In each of the previous cases, (local maxima, plateaus & ridge), the algorithm reaches a point at which no progress is being made.

→ A solution is to do a random-restart hill climbing - where random initial states are generated, running each until it halts or makes no progress. The best result is chosen (the one which gives maximum value

(Rather than starting at one initial state, start with multiple initial states, then calculate the goal state. Choose then the one which gives maximum value of goal state) best value)



→ hill-climbing uses local information: It decides what to do next by looking only at the "immediate" consequences of its choices

→ will terminate when at local optimum - and may not be able to reach global optimum.

→ The order of application of operators can make big difference.

Evergreen
Page No.
Date : / /20

→ Global information might be encoded in heuristic functions → considers future consequences

→ Example of Simple hill climbing.

Example

Start

A
D
C
B

Goal

D
C
B
A

Blocks world

Solve this problem using hill climbing algorithm

Sol case 1: using local heuristic (Defining a heuristic function $h(x)$)

$h(x)$

→ +1 for each block that is resting on the thing it is supposed to be resting on. (if correctly positioned)

→ -1 for each block that is resting on a wrong thing. (if incorrectly placed)

Start

total = 0

A	-1
D	+1
C	+1
B	-1

Goal

total = 4

D	+1
C	+1
B	+1
A	+1

i.e we should start with 0 and at the end, should be able to reach at 4.

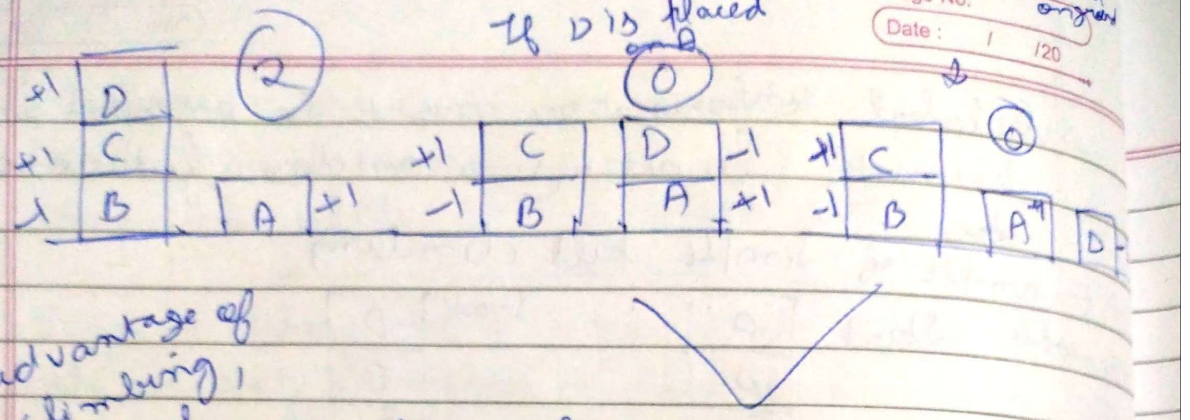
0

A
D
C
B

2 (1+1-1+1)
Total

+1	D
+1	C
-1	B

A	+1

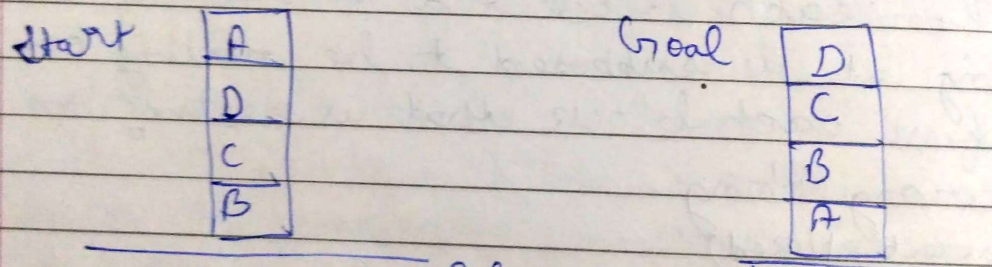


Disadvantage of hill climbing: access of local information is always the plateau

Now from both of these states, we are getting value = 0, so it is a kind of plateau

Mainly happens in case of local information

case 2 Using Global heuristic

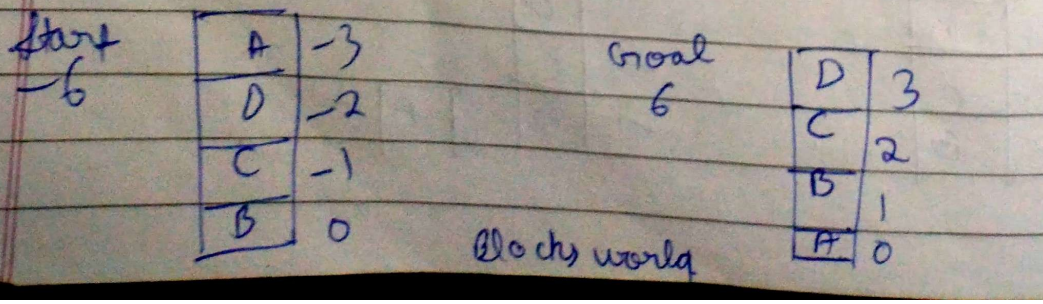


Blocks world

Global heuristic / (we do not just consider the block below the current block rather we are considering the support structure)

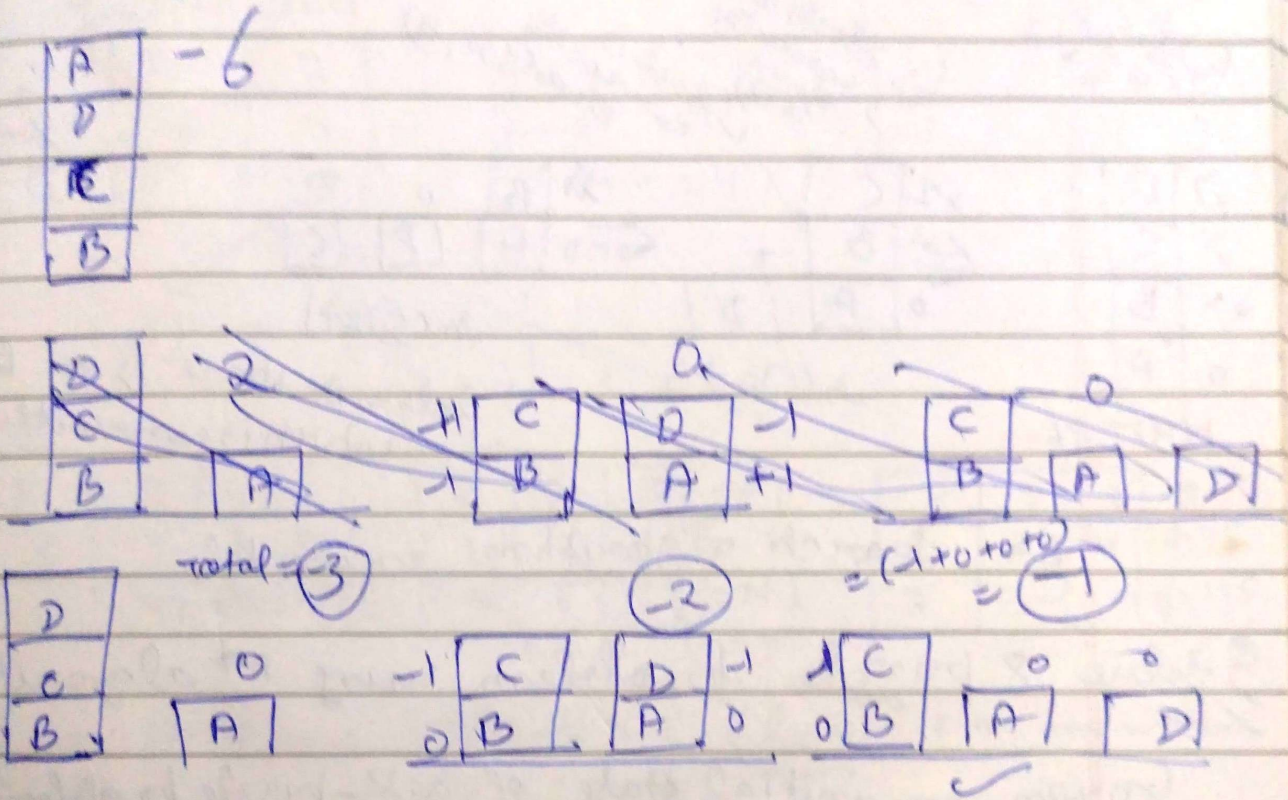
→ For each block that has the correct support structure : +1 to every block in the support structure

→ For each block that has a wrong support structure : −1 to every block in the support structure

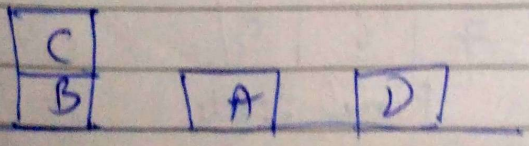


Start we are considering global info over here

Below B, nothing is placed so it is 0
Below C, B is incorrectly placed so it is -1
Below D, C & B are incorrectly placed,
∴ it is -2
Below A & D, C, B are incorrectly placed
so -3



Now comparing the three states, choosing the maximum value ie -1, so picking the third state.

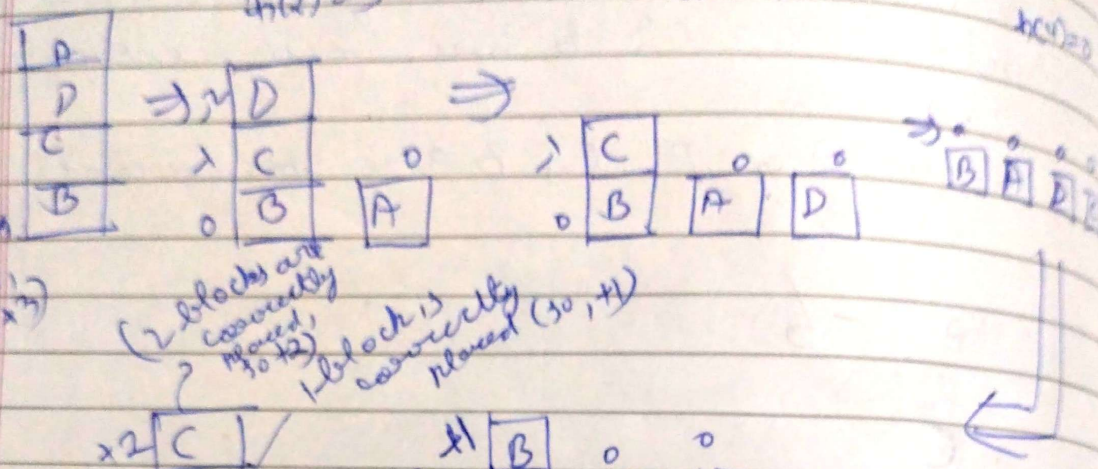


(as compared with the local search, same value was achieved from the 2nd & 3rd alternative/ possibility)

h(1) = -6

h(2) = +3

h(3) = -1



h(7) = +6

h(6) = +3

h(5) = +1

↳ Below A there is one blocks which is correctly placed

A* Search algorithm example

Q. Solve 8 puzzle problem using A* algorithm

Given an initial state of a 8-puzzle problem and final state to be ~~searched~~ reached

2	8	3
1	6	4
7		5

Initial State

1	2	3
8		4
7	6	5

Final State

Find the almost cost-effective path to reach the final state from initial state using A* algorithm

$$f(n) = g(n) + h(n)$$

Consider $g(n)$ = depth of node
$h(n)$ = no. of misplaced tiles

At root level depth = 0 $\therefore g(n) = 0$
$h(n)$ = no. of misplaced tiles = 4
(with respect to final state)

Initial state

2	8	3
1	6	4
7		5

$g = 0$
$h = 4$
$f = 0 + 4 = 4$

2	8	3
1	6	4
	7	5

$g = 1$
$h = 5$
$f = 1 + 5 = 6$

2	8	3
1		4
7	6	5

$g = 1$
$h = 3$
$f = 1 + 3 = 4$

2	8	3
1	6	4
7	5	

$g = 1$
$h = 5$
$f = 1 + 5 = 6$

If we compare all three ($f = 4$) is the minimum (with least misplaced tiles)

2	8	3
	1	4
7	6	5

$g = 2$
$h = 3$
$f = 2 + 3 = 5$

2		3
1	8	4
7	6	5

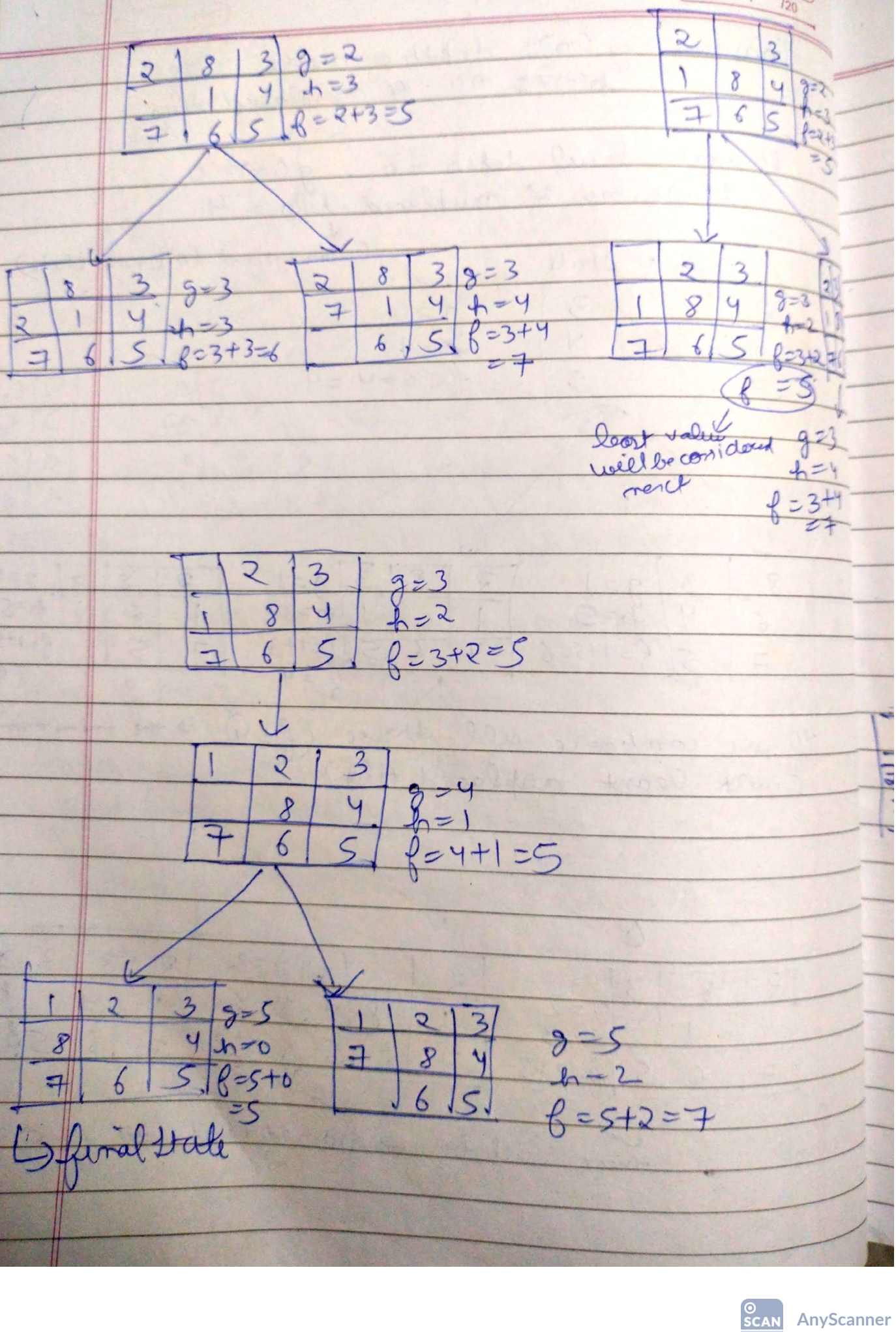
$g = 2$
$h = 3$
$f = 2 + 3 = 5$

2	8	3
1	4	
7	6	5

$g = 2$
$h = 4$
$f = 2 + 4 = 6$

These will be considered next with $f = 5$

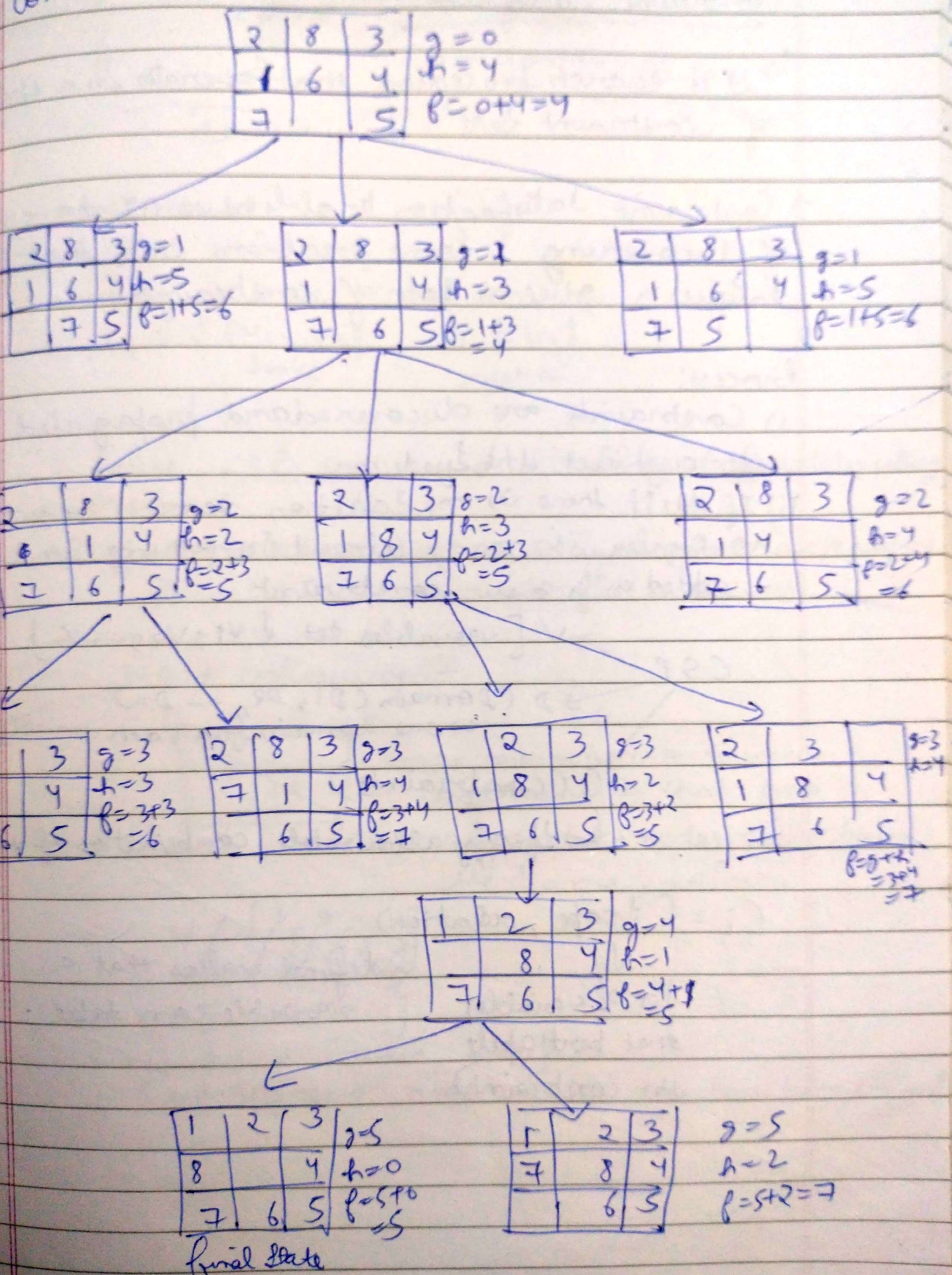
Now f = 5


2 8 3 g=2
1 4 h=3
7 6 5 f=2+3=5
g=3 h=3 f=3+3=6
g=3 h=4 f=3+4=7
g=3 h=2 f=3+2=5
f=5
least value will be considered next
g=3 h=4 f=3+4=7
g=4 h=1 f=4+1=5
g=5 h=0 f=5+0=5
final state
g=5 h=2 f=5+2=7

8-Puzzle using A*

Evergreen
Page No.
Date: / /20

Complete tree



Constraint Satisfaction

↳ It is search procedure that operates in a space of constraint sets

→ Constraint satisfaction problems in AI have goal of discovering some problem state that satisfies a given set of constraints.

Process:

i) Constraints are discovered and propagated throughout the system.

ii) If still there is no solution, search begins
→ Guess is made about something and added as a new constraint

CSP → V [variables set ↓ $V_1, V_2, \ldots V_n$]

→ D (Domain ($D_1, D_2, \ldots D_n$)
↳ one domain for each variable)

→ C (constraints)
↳ Specify allowable combination of values

C_i = (Scope, relation)

↳ defines values that a variable can take

(Set of variables that participate in constraint)

V_1 and V_2 (variables)

A B (A – domain of V_1, B – domain of V_2)

domain

(b) constraint

(1) values of V_1 & V_2 cant be same

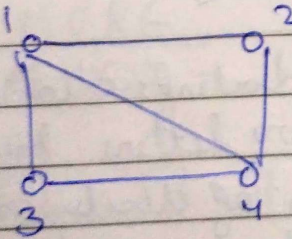
$C_1 = \{(V_1, V_2\}$, $V_1 \neq V_2\}$

scope , relation

To solve CSP, we used intelligent backtracking method

Intelligent Backtracking → only backtrack where conflict occurs

Ex Node colouring :-



we are given a graph. Constraint is that two adjacent nodes cant have the same colour

V = {1, 2, 3, 4}

D = {R, G, B}

V = {1, 2, 3, 4}

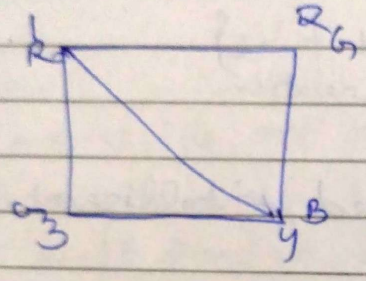
D = {Red, Green, Blue}

C = {adjacent nodes should not have same colour}

	1	2	3	4
Initial	R,G,B	R,G,B	R,G,B	R,G,B
1=R	R	G,B	G,B	G,B
2=G	R	G	G,B	B
3=B	R	G	B	Error
3=G	R	G	G	B

cannot be filled as no options left

∴ To solve this error we need to backtrack moving directly to point that caused problem



Advantages

① It is time efficient

Crypt Arithmetic Problem

↳ Types of constraint satisfaction problem (CSP)

↳ constraints → No two letters have same value

→ Sum of digits must be as shown in problem

→ There should be only one carry forward

→ Digits that can be assigned to a word/alphabet (0–9) range

→ Numbers must not begin with zero ie 0123 (zero is wrong), ~~(123)~~ 123 (correct)

Q 1

```
  T O
+ G O
-------
O U T
```

and left most digit can't be 0

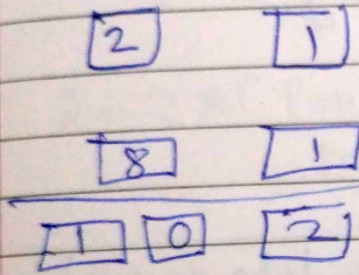
left most digit = ① (represents carry) at most

0–9 [9+8=17 ↓ max carry = 1)

Evergreen
Page No.
Date: / /20

Assign each letter a digit in the range [0-9] and the sum should be equal to OUT (given)

unique letter	Digit
T	2
O	1
G	8
U	0



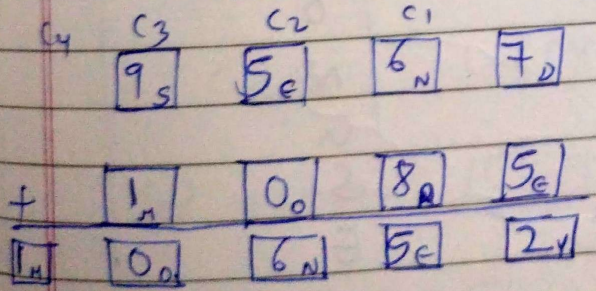
$2+G=U$, $2+G \geq 10$

$2+9=11$ ✗ can't be take as it will only 0 & U both to be 1 that is not possible

$2+8=10$ ✓

Q2 ✓

$$\begin{array}{r} S\ E\ N\ D \\ M\ O\ R\ E \\ \hline M\ O\ N\ E\ Y \end{array}$$



$S+M$, if $M=1$, $S+M \geq 10$

$S=9$

$E+O=N$ → zero(0) } if $C_2=0$ ✗ then $E=N$ which is wrong as every letter should be given a unique no.

$\therefore C_2=1$

$N+R=E$

$O+R=S$,

let $R=9$

$O+9=1S$ → E

↓ carry

$\rightarrow (\geq 1)\ O+8+1=1S$

$\therefore R=8$

but $S=9$ (already)

$\therefore$ R can't be 9

let $E=5$

$E+O+C_2=N$

$5+C_2=N$

$5+1=N \Rightarrow N=6$

$D + E = Y$

$D + S = Y$

we need carry that is possible only if $D \geq S$

$D - S - 6$

$-6 - N$

$-7 - E$

$-8 - R$

$-9 - S$

$7 + \overline{9} = Y$

$Y = 16$

carry

$Y = 10$

carry

$Y = 2$

Q3

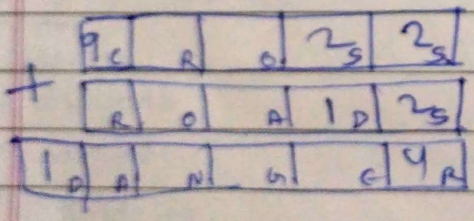
$C_5\ C_4\ C_3\ C_2\ C_1$

$C\ R\ O\ S\ S$

$+\ R\ O\ A\ D\ S$

$D\ A\ N\ G\ E\ R$

character	code
D	1
C	
R	
A	
O	
N	
G	
S	
E	



$S + S = R$

Rule 2: Sum of two same numbers is even

$\therefore$ $S + S = R$, So R is even (By using Rule 2)

$C_5 \neq 0$

$C_5 = 1$

Rule 1: Result should not should start with 0

$\therefore$ $C_5 = 1$