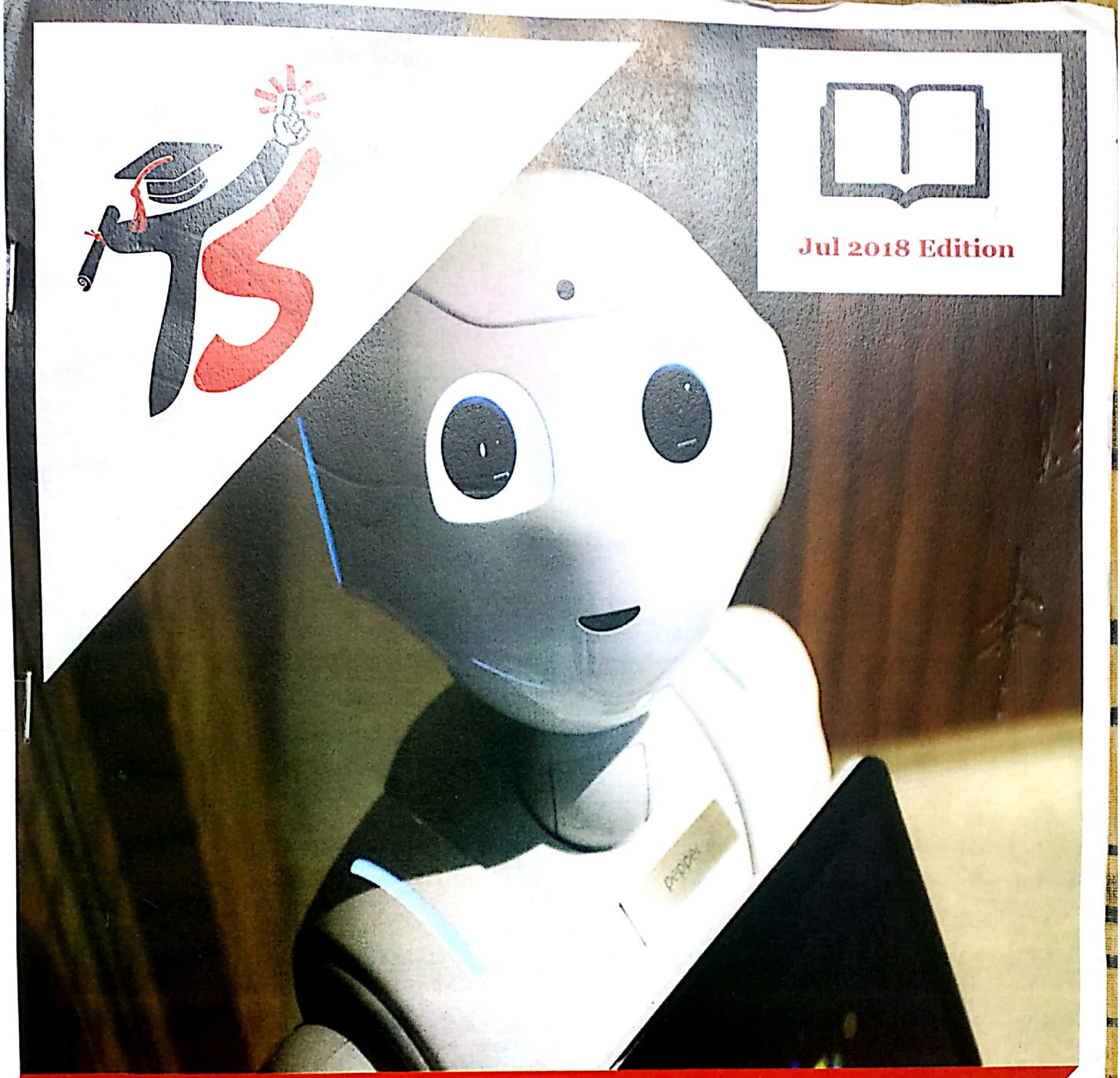


Jul 2018 Edition

Artificial Intelligence

(COMPUTER)

7 SEM

(As per Revised Syllabus w.e.f 2015-2016)

---- (Artificial Intelligence) ----

---- Marks Distribution ----

#	DEC-15	MAY-16	DEC-16	MAY-17	DEC-17	MAY-18
1.	-	-	-	05	-	-
2.	18	25	24	05	15	15
3.	38	60	52	50	60	45
4.	37	30	31	30	35	35
5.	22	15	18	35	15	30
6.	10	-	-	10	10	10
Rep.	-	**20**	**56**	**55**	**80**	**50**

--- Analysis by Topper's Solutions Team ---

LAST MINUTE PREPARATION:

Engineering is a notoriously demanding field of study. Being successful in engineering exams requires a **systematic and focused approach**. In order to do well, you will need to learn how to prepare for semester exam in engineering. Have you have already given up thinking, "**What the hell I can do at this moment? Tomorrow is exam!**" Think again! You are engineering student & last night studies are every engineering student's epitome.

"We engineers are known for our creativity"

Don't Worry entire **Topper's Solutions Team** is working out for betterment of students. Here are some techniques about **Artificial Intelligence (AI)** Subject.

➢ **How to score first 50 marks:**

Study **any one** of the below set and make sure you can easily attempt any questions from below chapters included in particular set.

SET - 1

#	Chapter Name	Weightage (Marks)
3	Problem solving	51
2	Intelligent Agents	18
5	Planning and Learning	23

OR

SET - 2

#	Chapter Name	Weightage (Marks)
3	Problem solving	51
4	Knowledge and Reasoning	33
5	Planning and Learning	23

➢ **How to score next 20 marks:**

Study the following 2 chapters.

#	Chapter Name	Weightage (Marks)
6	Applications	10
1	Introduction to Artificial Intelligence	5

Note: If you want to score good marks, study **ALL Chapters.**

Please Note: The Above Analysis is suggest by Topper's Solutions Team. Don't be completely dependent on it. It may change as per University of Mumbai Guidelines.

Contact No: 7507531198

Email ID: Support@ToppersSolutions.com

Website: www.ToppersSolutions.com

CHAPTER - 1: INTRODUCTION TO AI

Q1] Define AI. What are applications of AI? **[5M – May17]**

Ans:

AI:

1. AI Stands for **Artificial Intelligence.**
2. The term AI was coined by John McCarthy in 1956.
3. **John McCarthy** defines AI as "the branch of computer science **that** aims to create intelligent machines."
4. AI is usually the science of making computers to do things that require intelligence when done by humans.
5. It is a way of **making a computer, a computer-controlled robot, or a software think intelligently,** in the similar manner the intelligent humans think.
6. AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem.

GOALS OF AI:

- **To Create Expert Systems:** The systems which exhibit intelligent behavior, learn, explain, and advice its users is known as expert system.
- **To Implement Human Intelligence in Machines:** Creating systems that understand, think, learn, and behave like humans.

APPLICATIONS:

- **Gaming:** AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
- **Natural Language Processing:** It is possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems:** There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
- **Vision Systems:** These systems understand, interpret, and comprehend visual input on the computer.
- **Speech Recognition:** Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it.
- **Handwriting Recognition:** The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus
- **Intelligent Robots:** Robots are able to perform the tasks given by a human.

--- EXTRA QUESTIONS ---

Q1] Intelligent Agents

Ans:

1. Intelligence is broadly divided into:
   a. Natural Intelligence (Human).
   b. Artificial Intelligence (Machines/Robots).
2. Natural Intelligence is further divided into:
   a. Thought Process.
   b. Action.
3. Artificial Intelligence is further divided into:
   a. Think.
   b. Action.
4. **Intelligence System** is the system which thinks & acts like a **human** as well as the system which thinks & acts **rationally.**

DEFINITIONS OF INTELLIGENT AGENT:

- "The art of creating machines that performs functions that require intelligence when performed by people."
- "To study of how to make computers do things at which, at the moment people are better."
- "The automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning."
- "The branch of the computer science that is concerned with the automation of intelligent behavior."

Q2] Categorization of Intelligent Systems

Ans:

I) **Acting humanly:** The Turing Test approach.
- Natural language processing.
- Knowledge representation.
- Automated reasoning.
- Machine learning.

II) **Thinking humanly:** The cognitive modelling approach.

III) **Thinking rationally:** The laws of thought approach.
- Socrates is a man; all men are mortal; therefore Socrates is mortal.

IV) **Acting rationally:** The rational agent approach.
- Being able to see a tasty food helps one to move toward it.

CHAPTER - 2: INTELLIGENT AGENTS

Q1] What are the basic building blocks of Learning Agent? Explain each of them with a neat block diagram.

Q2] Draw and explain the basic building blocks of Learning Agent

Ans: [Q1 | 8M – Dec15, Dec16 & 10M – May16] & [Q2 | 10M – Dec17]

LEARNING AGENT:

1. Learning can be considered as the result of interaction between the agent and the world.
2. Turing proposed a new method to design the agent is to **build learning machines** and then to **teach them.**
3. A learning agent receives the percepts from its working environment.
4. It then analyses the feedback.
5. Later on it refines its actions with the help of the feedback received.
6. In Learning Agent, Learning refers to **improving initial knowledge** by operating in unknown environment.

LEARNING AGENT MODEL:

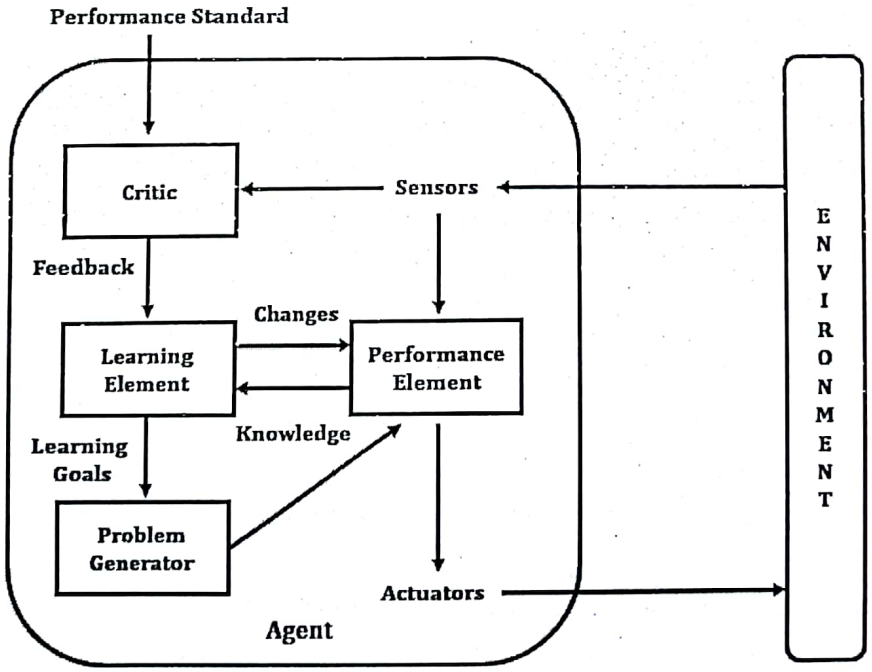


Figure 2.1: Learning Agent Model.

Figure 2.1 shows Model of Learning Agent. Learning Agent consists of four measure components.

I) **Performance Element:**

- Performance Element is an agent in itself.
- Performance Element takes the percept and decides an action.
- It has **condition-action rules.**
- This rules can be changed by Learning Element depending on the experience and feedback from Critic.

II) **Critic:**

- Critic provides **feedback** to the learning element.
- It determines how the performance element should be modified to do better in the future.
- Critic tells the learning element how well the agent is doing with respect to a fixed performance standard.
- It guides the agent from taking wrong steps in future by comparing its performance with the percepts.

III) **Learning Element:**

- It is consider as the **heart of Learning Agent.**
- Learning Element is responsible for making improvements.
- It learns and adapts to the changing needs of the environment.
- Learning Element can modify condition-action rules of the agent.
- It **guides the problem generator.**

iV) **Problem Generator:**

- Problem Generator is responsible for **suggesting actions** that will lead to new and informative experiences.
- It generates problems to explore the world.

EXAMPLE:

Automated Taxi Driver:

- When taxi goes out on the road, the critic element observes the world and passes information to the learning element.
- If taxi makes a quick left turn across three lanes of traffic, the critic observes the reaction of another driver.
- From this experience, the learning element is able to formulate a rule saying this was a bad action, and the performance element is modified by installing the new rule.
- Problem generator might identify certain areas of behavior in need of improvements and suggest experiments, such as trying out the brakes on different road surfaces under different conditions.

Q3] Define Rationality and Rational Agent. Give an example of rational action performed by any intelligent agent.

Ans: **[5M – Dec15]**

RATIONALITY:

1. Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.
2. Rationality implies the conformity of one's beliefs with one's reasons to believe, or of one's actions with one's reasons for action.
3. It is concerned with expected actions and results depending upon what the agent has perceived.
4. Performing actions with the **aim of obtaining useful information** is an important part of rationality.

RATIONAL AGENT:

1. A rational agent is an agent which has **clear preferences.**
2. It models uncertainty via expected values.
3. A rational agent can be anything that makes decisions, typically a person, firm, machine, or software.
4. A rational agent always performs right action, where the right action means the action that causes the agent to be most successful in the given percept sequence.
5. Rational agent is capable of **taking best possible action** in any situation.

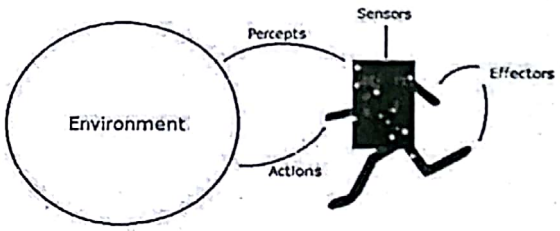


Figure 2.2: Agents Interact With Environment.

EXAMPLE OF RATIONAL ACTION PERFORMED BY ANY INTELLIGENT AGENT:

Automated Taxi Driver:

- **Performance Measure:** Safe, fast, legal, comfortable trip, maximize profits.
- **Environment:** Roads, other traffic, customers.
- **Actuators:** Steering wheel, accelerator, brake, signal, horn.
- **Sensors:** Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard.

Q4] Draw and describe the architecture of goal based agent. [6M – Dec16]

Ans:

GOAL BASED AGENTS:

1. Goal Based Agent is one of the type of **Agent Program.**
2. Goal Based Agent expand the capabilities of Model Based Agent by using **Goal information.**
3. Goal Based Agent maintains a Goal or Destination information.
4. Goal information describes situations that are desirable.
5. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches the goal state.
6. Searching and planning are used to achieve the agent's goal.
7. Goal Based Agent is based on **Reflex.**
8. It uses **Condition - Action Rule.**
9. It keeps the **track of world.**
10. Figure 2.3 shows the architecture of goal based agent.

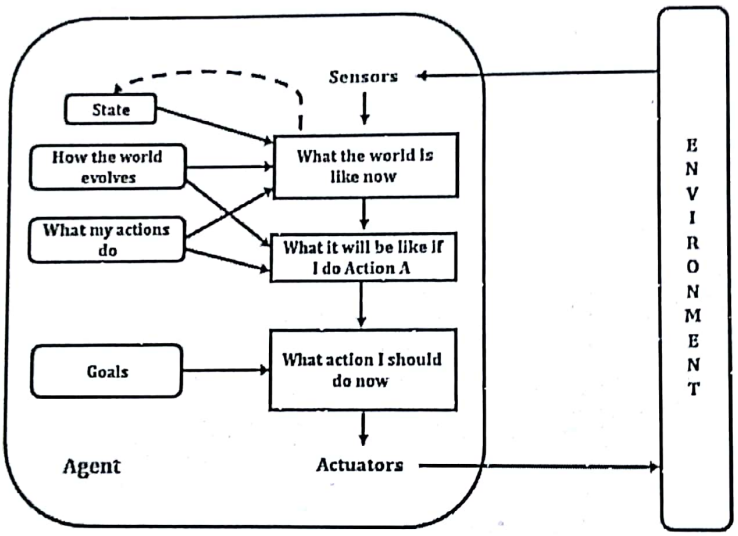


Figure 2.3: Goal Based Agent.

EXAMPLE:

- At a road junction, the taxi can go left, turn right, or go straight.
- The correct decision depends on where the taxi is trying to get to (i.e. destination or goal).
- Such agent differs from the reflex agent because in reflex agent we are having static table of condition - action and in the model based it involves consideration of the future status – "What will happen if I do such – and – such?"
- Goal based agent appears less efficient but more flexible due to its dynamic nature.
- If it starts to rain, the agent can update its knowledge of how effectively its brakes will operate.

Q5] Draw and Describe the Architecture of Utility based agent. How is it different from Model based agent?

Ans:

[10M – May18]

UTILITY BASED AGENT:

1. Goal-based agents only distinguish between goal states and non-goal states.
2. It is possible to define a measure of how desirable a particular state is.
3. This measure can be obtained through the use of a utility function.
4. Utility based agent choose actions based on a **preference (utility)** for each state.
5. Such agents try to maximize the performance with high quality behavior.
6. Performance measure = Degree of happiness.
7. Utility based agent **keeps the track of world.**
8. It can also work in **Partial Observable Environment.**
9. **Example:** There can be multiple action sequences that will get the taxi to its destination but some are quicker, safer, more reliable, or cheaper than others.
10. Utility agent examines "how happy I will be in such a state?"
11. Figure 2.4 present utility based agent.

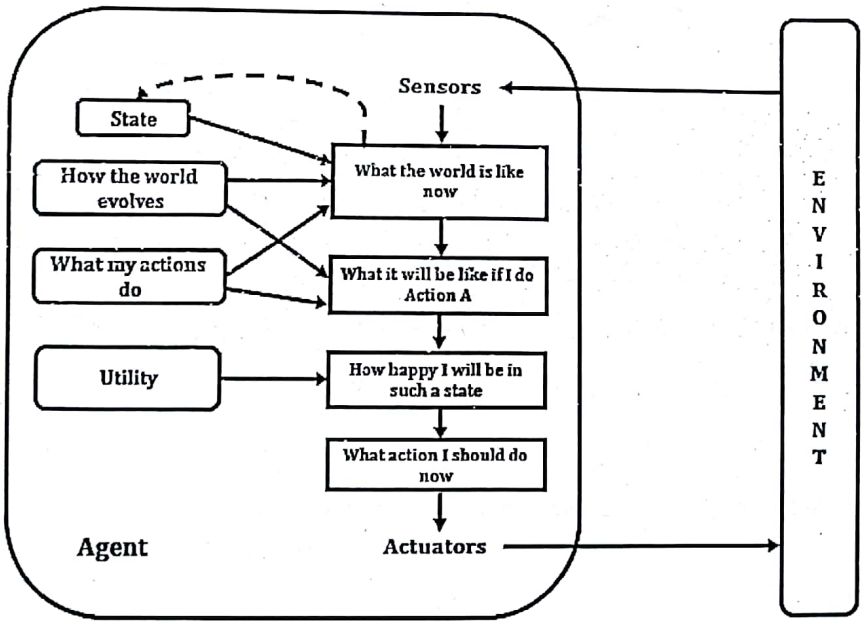


Figure 2.4: Utility Based Agent.

COMPARISON BETWEEN MODEL BASED AGENT WITH UTILITY BASED AGENT:

Refer Q12.

Q6] Explain Turing Test designed for satisfactory operational definition of intelligence. [5M – May16]

Ans:

TURING TEST:

1. Turing Test was proposed by **Alan Turing in 1950.**
2. It was designed to provide a satisfactory operational definition of intelligence.
3. It measures the performance of intelligent machine against that of human being.
4. The computer possess the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or not.
5. Therefore Turing Test can be considered as the art of creating machines that performs functions that require intelligence when performed by people.
6. Figure 2.5 shows the example of turing test.

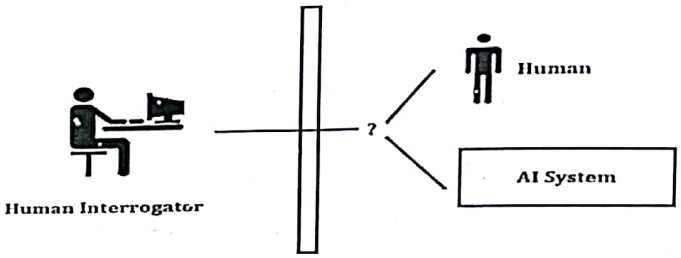


Figure 2.5: Turing Test Example.

TO PASS A TURING TEST, A COMPUTER MUST HAVE FOLLOWING CAPABILITIES:

- **Natural Language Processing (NLP):** Must be able to communicate successfully in English.
- **Knowledge Representation:** To store what it knows and hears..
- **Automated Reasoning:** Answer the Questions based on the stored information.
- **Machine Learning:** Must be able to adapt in new circumstances.

PROBLEM:

- Turing test is not reproducible, constructive, or amenable to mathematical analysis.

Q7] Define Intelligent Agent. What are the characteristics of Intelligent Agent?

Ans: **[5M – May18]**

INTELLIGENT AGENT:

1. An intelligent agent is a **component of artificial intelligence.**
2. An intelligent agent is an autonomous entity which observes through sensors and acts upon an environment using actuators and directs its activity towards achieving goals.
3. Intelligent agents may also learn or use knowledge to achieve their goals.
4. They may be very simple or very complex.
5. **Siri & Alexa** are the example of intelligent agent.

SOME DEFINITIONS OF INTELLIGENT AGENT:

- "The art of creating machines that performs functions that require intelligence when performed by people."
- "To study of how to make computers do things at which, at the moment people are better."
- "The automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning."
- "The branch of the computer science that is concerned with the automation of intelligent behavior."

CHARACTERISTICS OF INTELLIGENT AGENT:

Internal characteristics are as follows:

- **Learning/reasoning:** An agent has the ability to learn from previous experience and to successively adapt its own behavior to the environment.
- **Reactivity:** An agent must be capable of reacting appropriately to information from its environment.
- **Autonomy:** An agent must have both control over its actions and internal states.
- **Goal-oriented:** An agent has well-defined goals and gradually influence its environment.

External characteristics are as follows:

- **Communication:** An agent often requires an interaction with its environment to fulfill its tasks, such as human, other agents, and arbitrary information sources.
- **Cooperation:** Cooperation of several agents permits faster and better solutions for complex tasks that exceed the capabilities of a single agent.
- **Mobility:** An agent may navigate within electronic communication networks.
- **Character:** Like human, an agent may demonstrate an external behavior with many human characters as possible.

Q8] Give PEAS description for a Robot Soccer Player. Characterize its environment.

Ans: [5M – May16]

PEAS FOR ROBOT SOCCER PLAYER:

Performance Measure (P): To Play, Make Goal & Win the Game.

Environment (E): Soccer, Team Members, Opponents, Referee, Audience and Soccer Field.

Actuators (A): Navigator, Legs of Robot, View Detector for Robot.

Sensors (S): Camera, Communicators and Orientation & Touch Sensors.

ENVIRONMENT:

Partially Observable, Stochastic, Sequential, Dynamic, Continuous and Multi Agent.

Q9] What are PEAS descriptors? Give PEAS descriptors for a robot meant for cleaning the house.

Ans: [5M – Dec16]

PEAS:

Task Environment / Agents Environment can be specified with **PEAS.**

- Performance Measure (P): It specifies the performance expected by the agent.
- Environment (E): It specifies the surrounding condition where the agent has to perform a task.
- Actuators (A): It specifies the tool available for the agent to complete the task.
- Sensors (S): It specifies the tool required to sense the work environment.

PEAS FOR ROBOT MEANT FOR CLEANING THE HOUSE:

- Performance Measure (P): Maximize energy consumption, Maximize Dirt Pick up, Percentage of precision of cleaning.
- Environment (E): House, Dirt distribution unknown, assume actions are deterministic and environment is static.
- Actuators (A): Jointed Arm and hand, View Detector for Robot, Left, Right, Suck and NoOp.
- Sensors (S): Camera, Orientation & Touch Sensors, Sensors to identify the dirt and Potentiometric Sensor.

ENVIRONMENT:

Partially Observable, Stochastic, Sequential, Dynamic, Continuous and Multi Agent.

Q10] Give PEAS description for an Autonomous Mars Rover. Characterize its environment

Ans: **[5M – Dec17]**

PEAS:

Task Environment / Agents Environment can be specified with **PEAS.**

- **Performance Measure (P):** It specifies the performance expected by the agent.
- **Environment (E):** It specifies the surrounding condition where the agent has to perform a task.
- **Actuators (A):** It specifies the tool available for the agent to complete the task.
- **Sensors (S):** It specifies the tool required to sense the work environment.

PEAS FOR AUTONOMOUS MARS ROVER:

- **Performance Measure (P):** Terrain explored and reported, samples gathered and analyzed.
- **Environment (E):** Launch vehicle, lander and Mars.
- **Actuators (A):** Wheels/Legs, sample collection device, analysis devices and radio transmitter.
- **Sensors (S):** Camera, Touch Sensors, Accelerometers, Orientation Sensors, Wheels/Joint Encoders and Radio Receiver.

ENVIRONMENT:

Partially Observable, Stochastic, Sequential, Dynamic, Continuous and Single Agent.

Q11] Compare and Contrast problem solving agent and planning agent.

Ans: **[5M – Dec15, May16 & Dec16]**

Problem Solving Agent	Planning Agent
Problem Solving Agent is one kind of **Goal Based Agent.**	Planning Agent is the **combination of Problem Solving Agents & Knowledge-based Agents.**
Problem Solving Agent decides what to do by finding sequences of actions that lead to desirable states.	Planning Agent constructs the plans that achieve its goals, and then executes them.
Problem Solving Agent **does not use domain-independent heuristic function.**	Planning Agent **use domain-independent heuristic function.**

Problem Solving Agent demonstrate one specific solution.	Planning Agent can be viewed as the producer or generator of the solution.
Problem Solving Agent is concerned with **plan execution.**	Planning Agent is concerned with **plan generation.**
Limitations of the Problem Solving Approach motivates the design of planning systems.	Limitations of the Planning Approach does not motivates the design of Problem Solving systems.
Example: Vacuum World.	**Example:** STRIPS

Q12] Compare Model based Agent with Utility based Agent.

Ans: **[5M – May17]**

Model Based Agent	Utility Based Agent
Model Based Agent choose actions based on model of the world.	Utility Based Agent choose actions based on preference (utility) for each state.
Model Based Agent does not examines "how happy I will be in such a state?"	Utility Based Agent examines "how happy I will be in such a state?"
It does not use utility function.	It uses utility function.
It is Reflex Agent.	It is not a Reflex Agent.
It uses condition action rule.	It does not condition action rule.
Diagram: Refer Figure 2.6	**Diagram:** Refer Figure 2.7

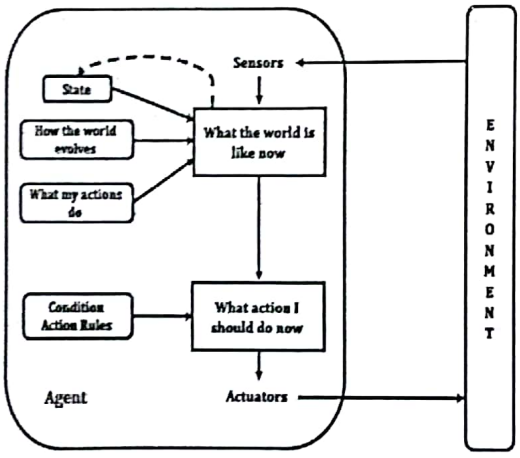


Figure 2.6: Model Based Reflex Agent.

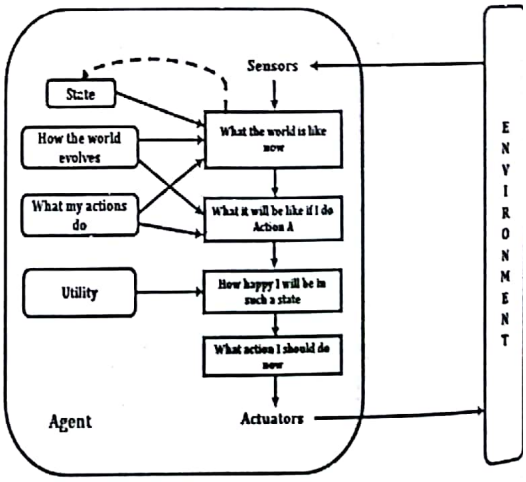


Figure 2.7: Utility Based Agent.

--- EXTRA QUESTIONS ---

Q1] Types of Agents?

Ans:

I) TABLE DRIVEN AGENT:

- It is the simplest type of Agent.
- It uses Look-up table for percept & action mapping.
- It is not suitable for real life problems.
- **Example:** Chess playing Agents which has total combination of 35^{100}
- It is difficult & tedious to prepare Look-up table.
- It requires large memory space to store table.
- An agent has no Autonomy or Intelligence.

II) SIMPLE REFLEX AGENTS:

- This is improved agent compared to Table Driven Agent.
- It uses Condition-Action rule instead of Percept-Action.
- Such agents select action on the basis of the current percept, ignoring the rest of the percept history.
- Having admirable property of being simple, but they turn out to be of very limited intelligent.
- Suitable only when environment is fully observable.
- **Example: If** car in front is braking **then** initiate- braking.
- Figure 2.8 shows the schematic of the simple reflex agent.

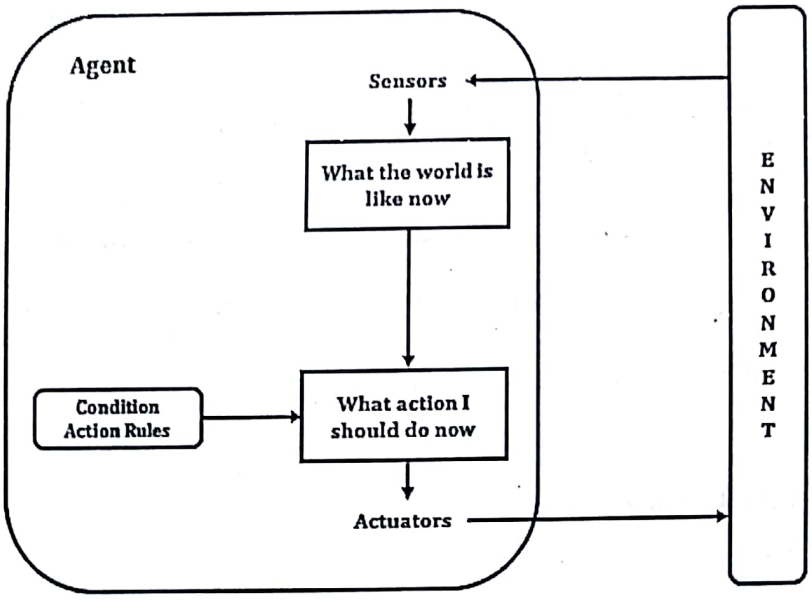


Figure 2.8: Simplex Reflex Agent.

III) MODEL-BASED REFLEX AGENTS:

- Model Based Agent keeps track of world.
- It is based on reflex.
- It also uses Condition-Action Rule.
- It can work in Partial Observable Environment.
- It maintains some sort of internal state that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.
- Updating the internal state information as time goes by requires two types of knowledge to be encoded in the agent program.
  - How the world evolves independently of the agent.
    For example: Overtaking car will be closer behind than it was a moment ago.
  - How the agent's own actions affect the world.
    For example: When the agent turns the steering wheel clockwise, the car turns to the right.
- Such knowledge if implemented in scientific theory or Boolean circuits is called model and the agent that uses such a model is called a model-based agent.

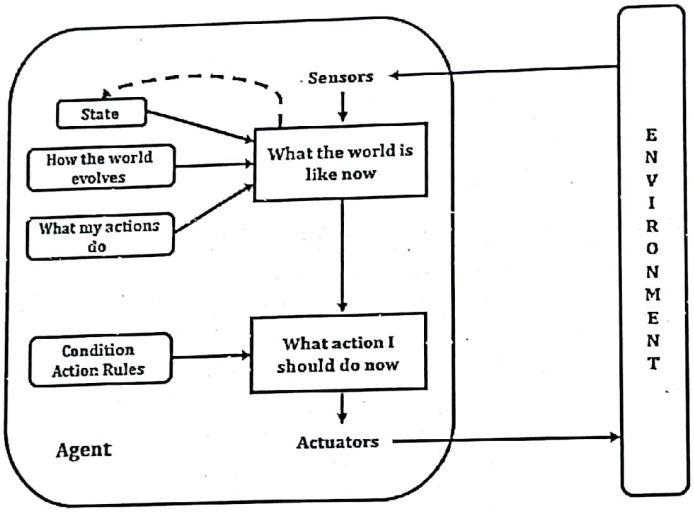


Figure 2.9: Model Based Reflex Agent.

IV) GOAL BASED AGENTS:

- **Refer Q4.**

V) UTILITY BASED AGENTS:

- **Refer Q5.**

Q2] Dimensions/ Properties of Environment?

Ans:

Used to categorize the environment and should be considered while designing particular agent.

I) **Fully Observable vs. Partially Observable:**

- **Fully Observable:**
  - Entire information can be accessed using sensors.
  - Easy to work for agent.
  - No need to keep track of world.
- **Partially Observable:**
  - Entire information is not available.
  - Difficult to work for Agent.
  - Agent needs to keep track of world.
- **Example:** Automated taxi cannot see what other drivers are thinking.

II) **Deterministic vs. Stochastic:**

- **Deterministic:** Next state can be uniquely defined if current state and action or input is known.
- **Stochastic:** Next state can't be uniquely defined. Hence difficult to write.
- Taxi driving is clearly stochastic.

III) **Episodic vs. Sequential:**

- In an episodic environment, the agent's experience is divided into atomic episodes. Each episode consists of the perceiving and then performing a single action.
- In episodic environment, the choice of action in each episode depends only on the episode itself.
- In sequential environment, on the other hand, the current decision could affect all future decisions.
- **Example:** Detecting defective parts on an assembly line is episodic task while chess and taxi driving are the sequential tasks.

IV) **Static vs. Dynamic:**

- **Static:**
  - If the environment cannot change while an agent is deliberating, then we say the environment is static.
  - Static environment is easy to deal with.
- **Dynamic:**
  - If the environment can change while an agent is deliberating, then we say the environment is dynamic.

- Difficult to work for agent.
- Action may become redundant.

➢ **Example:** Taxi driving is clearly dynamic. Crossword puzzles are static.

V) **Discrete vs. Continuous:**

➢ Such distinction is applied to the environment according to the way in which time is handled.

➢ **Example:** Chess game is discrete-time and taxi driving is continuous.

VI) **Single agent vs. Multi agent:**

➢ Depends upon no. of agents present in the environment.

➢ An agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is two-agent environment.

CHAPTER - 3: PROBLEM SOLVING

Q1] Explain Hill-Climbing algorithm with an Example.

Ans: **[5M – May16 & Dec17]**

HILL CLIMBING ALGORITHM:

1. Hill Climbing Algorithm is the **local search algorithm.**
2. It is used for continuous state space problem or when number of nodes in the tree are very large.
3. It uses **numerical optimization technique** to find the goal.
4. The node which optimizes performance measure is selected.
5. Search starts with initial random guess and proceeds till the maxima are reached.
6. Hill Climbing Algorithm needs very small amount of memory as no tree search is performed.

EXAMPLE:

Consider the Example of Hill Climbing Algorithm shown in Figure 3.1.

Search Path = [S, B, E, G_1]

S → B → E → G_1

Cost = 1 + 2 + 3

= 6

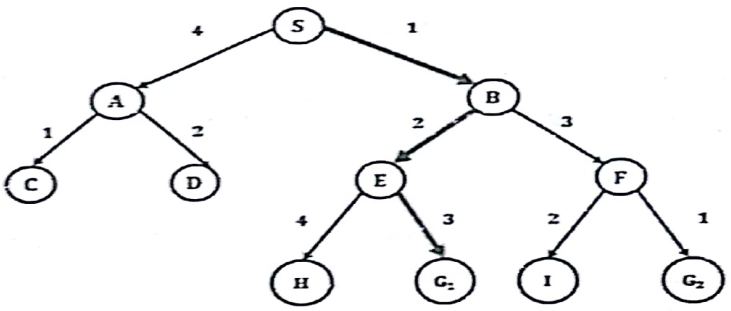


Figure 3.1: Hill Climbing Example.

Q2] What are the problems/frustration that occur in hill climbing technique? Illustrate with an example.

Ans: **[6M – Dec15 & 5M – Dec16 & May17]**

PROBLEMS IN HILL CLIMBING TECHNIQUE:

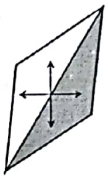
I) Local Maxima:

- Local Maxima is a state that is better than all of its neighbours.
- But it is not better than some other states far away.

- Hill Climbing Algorithm tends to find **only local maxima.**
- This problem can be solved using **Backtracking.**

II) **Ridges:**

- Ridge is a curve in the search space that can **lead to maxima.**
- The orientation of the high region, compared to the set of available moves, makes it impossible to climb up.
- However, two moves executed serially may increase the height.
- Move to the several direction at once can help in dealing with the ridges.



III) **Plateau:**

- Plateau is a flat area of the search space in which all neighbouring states have the same value.
- The heuristic of Plateau region has **same value.**
- In plateau it is not possible to determine the best direction by using local comparison.
- Solution to plateau is to take a big jump to any direction to get to a new search space.

EXAMPLE:

Depending on the initial state, Hill-Climbing gets stucks in Local Maxima.

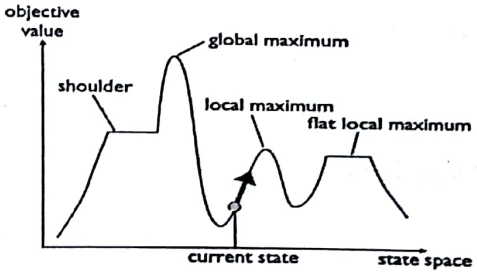


Figure 3.2: Local Maxima.

Consider the example of 8 Puzzle using Hill Climbing. As shown in figure 3.3 it get stuck in Local Maxima. Hence we need to backtrack.

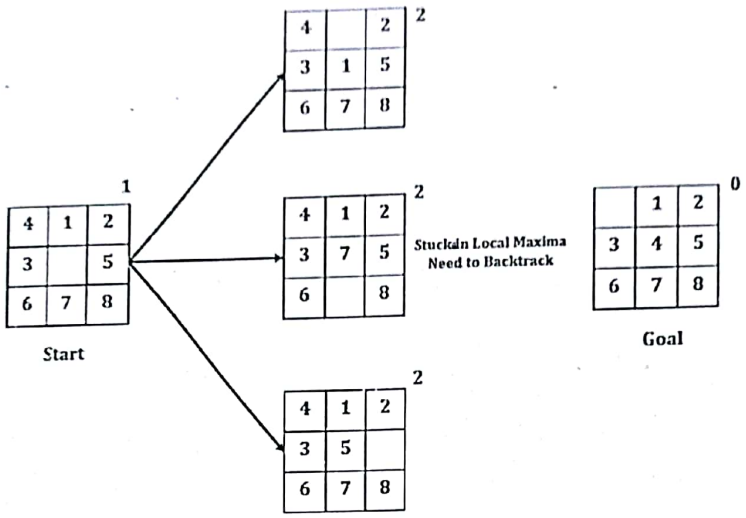


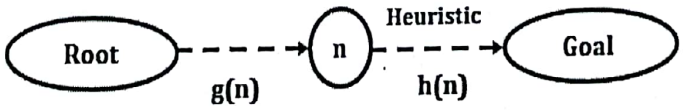
Figure 3.3: Example of Local Maxima.

Q3] Define heuristic function. Give an example heuristics function for Blocks World Problem. **[5M – Dec15]**

Ans:

HEURISTIC FUNCTION:

1. Heuristic Function is the function that gives an estimation on the Cost of getting from node to the goal state.
2. Heuristic Function is used in **Informed Search Technique.**
3. It is used in a decision process to try to make the best choice of a list of possibilities.
4. Best move is the one with the least cost.
5. Heuristic function helps in **implementing goal oriented search.**
6. It is usually used to increase the efficiency of the search process.
7. Figure 3.4 represents Heuristic Function.

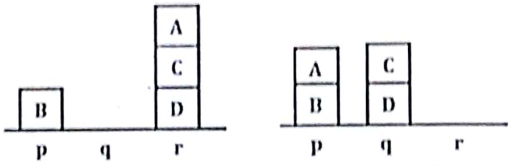


Note: h(n) – Cost from Node n to Goal

Figure 3.4: Heuristic Function.

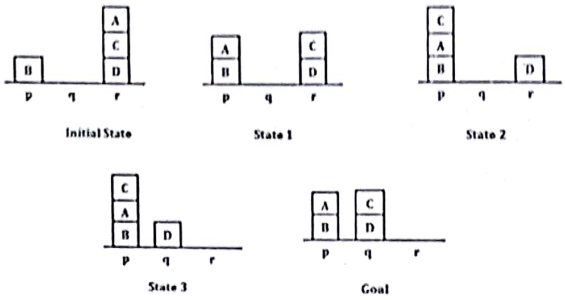
EXAMPLE OF HEURISTICS FUNCTION FOR BLOCKS WORLD PROBLEM:

Consider the example of Block World Shown Below:



Initial State Goal

As shown below the heuristic value of H(s) = 4. Since we take 4 move to reach from initial state to Goal State.



Q4] Find the heuristics value for a particular state of the Block World Problem.

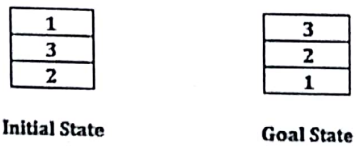
Ans: [5M – Dec15]

BLOCK WORLD PROBLEM:

1. Block World is the Classic Toy-World Problem of AI.
2. It has been used to develop AI System for Vision, Learning, Language Understanding and Planning.
3. It consist of set of blocks placed on the table top.
4. The task is usually to stack the blocks in some predefined order.

EXAMPLE:

Consider the example of Block World shown below:



Initial State Goal State

Figure 3.5: Block World heuristic Example.

- To compute the heuristic value of a state, we give point to each block in the state.
- The point to a block can be +1 or -1.

- It is +1 if the block has a correct block immediately below it compared to the goal state.
- It is -1 if the block has an incorrect block immediately below it compared to the goal.
- Finally the points of all blocks in the state are added to give the heuristic value of the particular state.

For instance let us compute the heuristic value for the initial state as shown in figure 3.5.

- **For Block 2:** In the initial state nothing is below block 2, but in the goal state 1 is below block 2 hence point for block 2 is -1.
- **For Block 3:** In the initial state block 2 is below block 3 and in the goal state also block 2 is below block 3 and hence point for block 3 is +1.
- **For Block 1:** In the initial state block 3 is below block 1 but in the goal state nothing is below block 1 and hence point for block 1 is -1

Now final heuristic value for the initial state can be computed as addition of all the points

Hence, **H (initial state) = (-1) + (+1) + (-1) = -1** and so on heuristic value for each state can be computed.

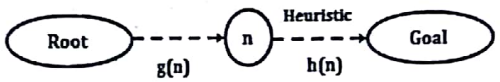
For goal state since all the blocks are in place each block will have a point of +1 and hence its heuristic value is +3.

Q5] Define heuristic function. Give an example heuristics function for 8-puzzle problem. Find the heuristics value for a particular state of the Blocks World Problem.

Ans: **[5M - Dec16 & May17]**

HEURISTIC FUNCTION:

1. Heuristic Function is the function that gives an estimation on the Cost of getting from node to the goal state.
2. Heuristic Function is used in **Informed Search Technique.**
3. It is used in a decision process to try to make the best choice of a list of possibilities.
4. Best move is the one with the least cost.
5. Heuristic function helps in **implementing goal oriented search.**
6. It is usually used to increase the efficiency of the search process.
7. Figure 3.6 represents Heuristic Function.



Note: h(n) – Cost from Node n to Goal

Figure 3.6: Heuristic Function.

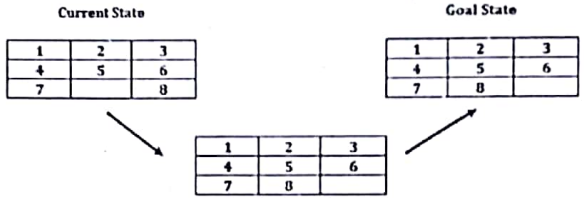
HEURISTICS FUNCTION FOR 8-PUZZLE PROBLEM:

For 8-Puzzle Problem, two heuristics are commonly used. i.e. No. of misplaced tiles and Manhattan Distance.

I) **No. of Misplaced tiles:**

- In this case, heuristic function is determined using number of misplaced titles (not including the blank space).
- **For Example:**
- Consider the example given below. It consists of initial state and goal state.

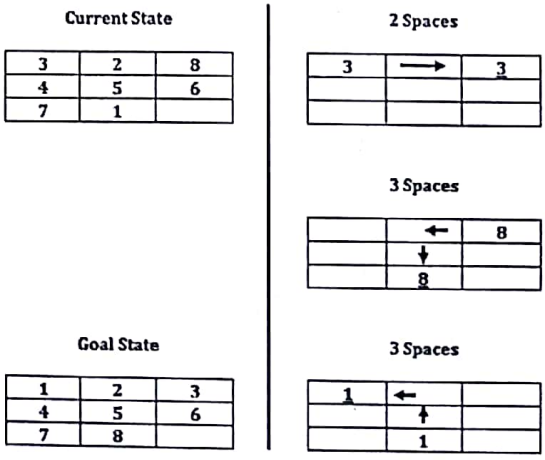
- As shown in example, only "8" is misplaced, so the heuristic function evaluates to 1.



∴ **Heuristic h (n) = 1**

II) **Manhattan Distance:**

- Manhattan Distance is the sum of the distances of the tiles from their goal positions.
- **For Example:**
- Consider the example given below. It consists of initial state and goal state.
- In this case, only the "3", "8" and "1" titles are misplaced by 2, 3 and 3 spaces respectively. So the heuristic function evaluates to 8.
- Therefore, Heuristic h (n) = 2 + 3 + 3 = 8



HEURISTICS VALUE FOR A PARTICULAR STATE OF THE BLOCKS WORLD PROBLEM:

Refer Q4.

Q6] Design a planning agent for a Blocks World problem. Assume suitable initial state and final state for the problem.

[10M – May16 & Dec17]

Ans:

PLANNING AGENT:

1. Planning Agent has Ideal Planner along with knowledge base inside its memory.
2. Agent tells the knowledge base about the percept.
3. But instead of asking the action to knowledge base, it ask the Goal.
4. Once Goal is provided by the knowledge base, it forwards it to Ideal Planner.
5. Ideal Planner prepares a Plan to achieve the Goal.
6. The steps of this plan are then implemented as the action.
7. Figure 3.7 represents the planning agent.

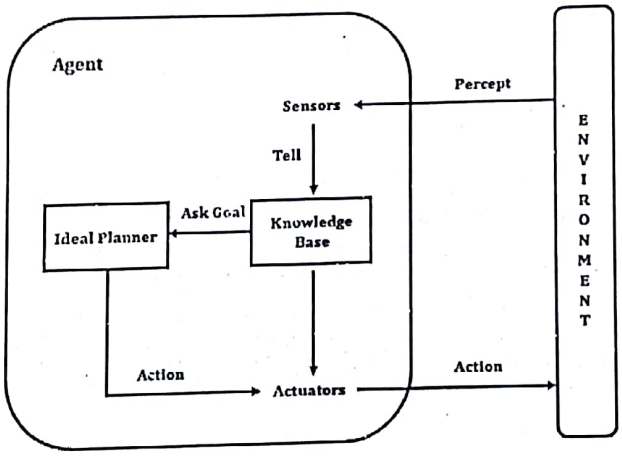


Figure 3.7: Planning Agent.

PLANNING AGENT FOR A BLOCKS WORLD PROBLEM:

Consider the Initial State and Goal State as shown below in figure 3.8 for Block World Problem.

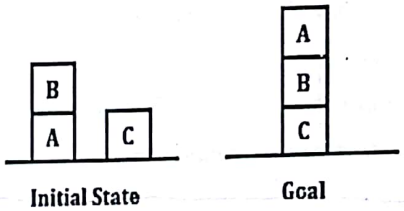


Figure 3.8: Example of Block World Problem.

Planning Agent Component for Block World Problem:

Initial State	Goal	A Plan
Clear (A)	On (B, C)	Pickup (B)
Clear (B)	On (A, B)	Stack (B, C)
Clear (C)		Pickup (A)
onTable (A)		Stack (A, B)
onTable (B)		
onTable (C)		

Q7] Write a short note on genetic algorithm.

Ans: **[8M – Dec15 & 10M – May17]**

GENETIC ALGORITHM:

1. Genetic Algorithms are a part of evolutionary computing, which is a rapidly growing area of Artificial Intelligence.
2. Genetic Algorithms are inspired by **Darwin's theory about evolution.**
3. It is an **intelligent random search technique.**
4. It is used to **solve optimization problem.**
5. Genetic Algorithm use encode solutions as fixed length "bit strings".
6. **Example:** 101110, 111111, 000101
7. Genetic Algorithm works by testing any string and getting a score indicating how good that solution is.
8. The set of all possible solutions [0...1000] is called as search space or state space.

GENETIC ALGORITHM PSEUDO CODE:

```
Generate an initial population of individuals
Evaluate the fitness of all individuals
While termination condition not met do
    Select fitter individuals for reproduction.
    Recombine between individuals
    Mutate individuals
    Evaluate the fitness of the modified individuals
    Generate a new population
End While
```

GENETIC ALGORITHM FLOWCHART:

Figure 3.9 shows the flowchart for genetic algorithm.

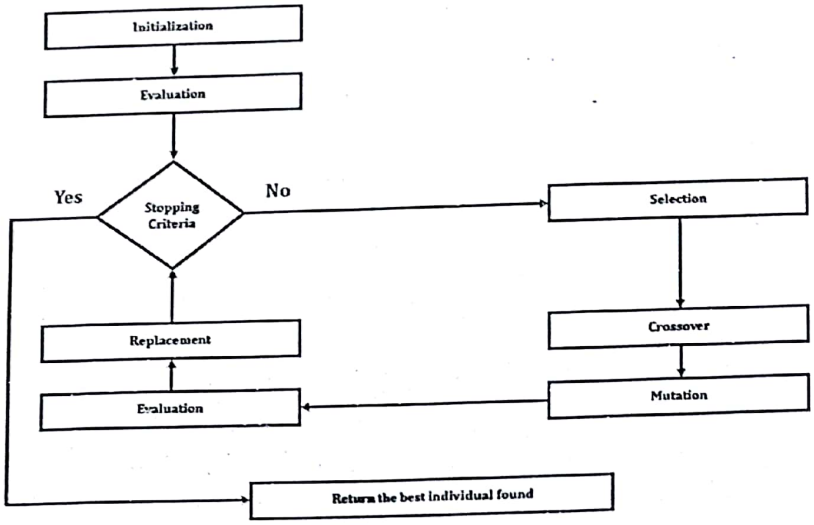


Figure 3.9: Genetic Algorithm Flowchart.

COMPONENTS OF GENETIC ALGORITHM:

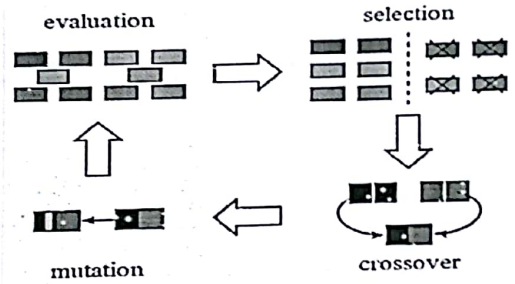


Figure 3.10: Genetic Algorithm Components.

1. For example, in Tournament Selection, the algorithm selects an individual with the highest fitness value from a random subset of the population.
2. Genetic Algorithm's evolution operates in three stages.
3. Selection, where it chooses a relatively fit subset of individuals for breeding.
4. Crossover, where it recombines pairs of breeders to create a new population.
5. Mutation, where it potentially modifies portions of new chromosomes to help maintain the overall genetic diversity.
6. Arrows in the figure 3.10 indicate transitions into the next genetic operation within one generation.

ADVANTAGES:

- It is easy to understand.
- Chance of getting optimal solution is higher.

LIMITATIONS OF GENETIC ALGORITHM:

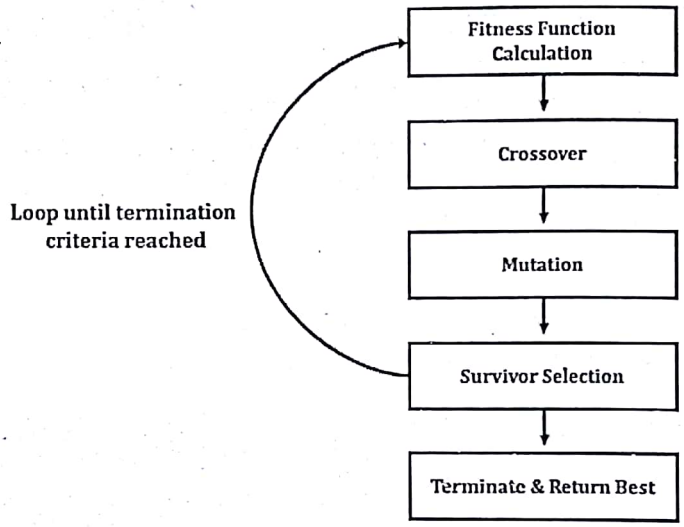
- Cross over rate should be 80-95%.
- Fitness function must be accurate.

Q8] Explain how genetic algorithm can be used to solve a problem by tasking a suitable example

Ans: [10M – May16]

MAXONE Problem:

Genetic Algorithm steps to solve problem:



1. Produce an initial population of individuals.
2. Evaluate the fitness of all individuals.
3. While termination condition not met do.
    a. Select fitter individuals for reproduction.
    b. Recombine between individuals.
    c. Mutate individuals.
    d. Evaluate the fitness of the modified individuals.
    e. Generate a new population.
4. End while.

Encoding:

- An individual is encoded (naturally) as a string of 'L' binary digits
- Let's say L = 10.
- Then, 1 = 0000000001 (10 bits)

Produce an initial population of individuals:

- We start with a population of n random strings. Suppose that $l = 10$ and $n = 6$.
- We toss a fair coin 60 times and get the following initial population:

$S_1 = 1111010101$
$S_2 = 0111000101$
$S_3 = 1110110101$
$S_4 = 0100010011$
$S_5 = 1110111101$
$S_6 = 0100110000$

Evaluate the fitness of all individuals:

Now we evaluate the fitness based on 1's.

$S_1 = 1111010101$	$f(S_1) = 7$
$S_2 = 0111000101$	$f(S_2) = 5$
$S_3 = 1110110101$	$f(S_3) = 7$
$S_4 = 0100010011$	$f(S_4) = 4$
$S_5 = 1110111101$	$f(S_5) = 8$
$S_6 = 0100110000$	$f(S_6) = 3$

Therefore $f(S) = 34$

Selection:

- Next we apply fitness proportionate selection with the roulette wheel method.
- We repeat the extraction as many times as the number of individuals.
- We need to have the same parent population size (6 in our case)
- Suppose that, after performing selection, we get the following population:

$S_1` = 1111010101$	(S1)
$S_2` = 1110110101$	(S3)
$S_3` = 1110111101$	(S5)
$S_4` = 0111000101$	(S2)
$S_5` = 0100010011$	(S4)
$S_6` = 1110111101$	(S5)

Crossover:

- For each couple we decide according to crossover probability (for instance 0.6) whether to actually perform crossover or not.
- Suppose that we decide to actually perform crossover only for couples (s1`, s2`) and (s5`, s6`).

- For each couple, we randomly extract a crossover point, for instance 2 for the first and 5 for the second.

Before Crossover:

S_1` = 1111010101 $S_5$` = 0100010011

S_2` = 1110110101 $S_6$` = 1110111101

After Crossover:

S_1`` = 1110110101 $S_5$`` = 0100011101

S_2`` = 1111010101 $S_6$`` = 1110110011

Mutation:

- The final step is to apply random mutation.
- For each bit that we are to copy to the new population we allow a small probability of error (for instance 0.1)
- Causes movement in the search space (local or global)
- Restores lost information to the population

Before applying mutation:

S_1`` = 1110110101

S_2`` = 1111010101

S_3`` = 1110111101

S_4`` = 0111000101

S_5`` = 0100011101

S_6`` = 1110110011

After applying mutation:

S_1``` = 1110100101

S_2``` = 1111110100

S_3``` = 1110101111

S_4``` = 0111000101

S_5``` = 0100011101

S_6``` = 1110110001

Evaluate the fitness of the modified individuals:

After Applying Mutation:

S_1``` = 1110100101 $f(S_1```) = 6$

S_2``` = 1111110100 $f(S_2```) = 7$

S_3``` = 1110101111 $f(S_3```) = 8$

S_4``` = 0111000101 $f(S_4```) = 5$

S_5``` = 0100011101 $f(S_5```) = 5$

S_6``` = 1110110001 $f(S_6```) = 6$

Therefore $f(S) = 37$

In one generation, the total population fitness changed from 34 to 37, thus improved by ~9%

At this point, we go through the same process all over again, until a stopping criterion is met.

Q9] Define the terms chromosome, fitness function, crossover and mutation as used in Genetic algorithms. Explain how genetic algorithms work?

[10M – Dec17 & May18]

Ans:

CHROMOSOME:

1. In genetic algorithm, a chromosome is a set of parameters which define a proposed solution to the problem that the genetic algorithm is trying to solve.
2. The set of all solutions is known as the **population.**

FITNESS FUNCTION:

1. A fitness function is a particular type of **objective function.**
2. The fitness function is a function which takes a candidate solution to the problem as input and produces as output how "fit" our how "good" the solution is with respect to the problem in consideration.

CROSSOVER:

1. Crossover is also known as **recombination.**
2. Crossover is a genetic operator used to combine the genetic information of two parents to generate new offspring.

FITNESS FUNCTION:

1. Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next.
2. It is analogous to biological mutation.

GENETIC ALGORITHM:

Refer Q8.

Q10] Explain A* Algorithm with example

[10M – May18]

Ans:

A*:

1. A* Algorithm is form of **Best First Search.**
2. It is an **Informed Search Technique.**

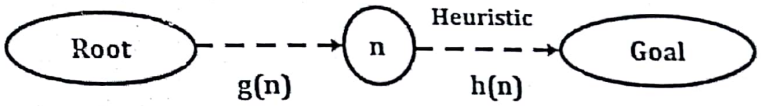
3. Additional knowledge in terms of heuristic function is made available.
4. A* Algorithm uses **Priority Queue.**
5. New members are added in the Queue in the ascending order at Evaluation Function f (n).
6. A* uses heuristic function h (n) as well as g (n) for Evaluation.

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

   g (n) = cost so far to reach n

   h (n) = estimated cost from n to goal

   f (n) = estimated total cost of path through n to goal
7. Heuristics Function h (n) gives estimated cost of reaching Goal from node n. while g (n) gives cost of reaching node n from root.



Note: h(n) – Cost from Node n to Goal

ALGORITHM:

1. Create a single member Queue comprising of Root Node.
2. If first member of Queue is Goal, then go to step 5.
3. It first member of Queue is not Goal then remove it from the Queue.

   Consider its Successor if any, and add them to the Queue in ascending order of Evaluation Function f (n).
4. If the Queue is not empty, then go to step 2.

   If the Queue is empty, then go to step 6.
5. Print "SUCCESS" & Stop.
6. Print "FAILURE" & Stop.

ADVANTAGES:

- A* algorithm is complete.
- It is optimal due to admissible heuristic.

LIMITATIONS:

- Large memory space needed as it generates nodes which are not used.
- It never opens nodes for which f (n) > C*, where C* is optimal cost.

Q11] Prove that A* is admissible if it uses a monotone heuristic.

Q12] Prove the admissibility of A*

[Q11 | 5M – May16] & [Q12 | 6M - Dec16]

Ans:

A*:

1. A* is an **informed search technique.**
2. An admissible heuristic is one that never overestimates the cost to reach the goal.
3. A* uses heuristics function h (n) as well as g (n) for evaluation.

$$F(n) = G(n) + H(n)$$

4. A heuristic is monotone, if the heuristic value is non-decreasing along any path from start to goal.
5. All monotone heuristics are **admissible.**
6. Heuristic Value of A* technique at the Goal node is $H(n^*) = 0$
7. So one step away from the goal,

$$H(n) \leq H(n') + Cost(n, n') \leq Cost(n, n^*)$$

8. By induction, $H(n) \leq Cost(n, n*)$ from any node.
9. Thus it is admissible since it always underestimates the path cost to the goal state.
10. Therefore if a node is expanded using a monotone heuristic, A* has found the optimal route to that node.
11. A* is admissible if it uses an optimistic heuristic estimate,
12. Therefore all A* Algorithms are admissible.

Q13] Draw game tree for a Tic-Tac-Toe Problem.

[4M – Dec15]

Ans:

TIC-TAC-TOE PROBLEM:

1. Tic-Tac-Toe is also known as **noughts and crosses or X's and O's.**
2. It is a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3×3 grid.
3. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.
4. Because of the simplicity of Tic-Tac-Toe, it is often used as a pedagogical tool for teaching the concepts of good sportsmanship.
5. Artificial Intelligence use Tic-Tac-Toe Problem to deal with the searching of game trees.

---

6. Figure 3.11 shows Game Tree for Tic-Tac-Toe Problem.

GAME TREE FOR TIC-TAC-TOE PROBLEM:

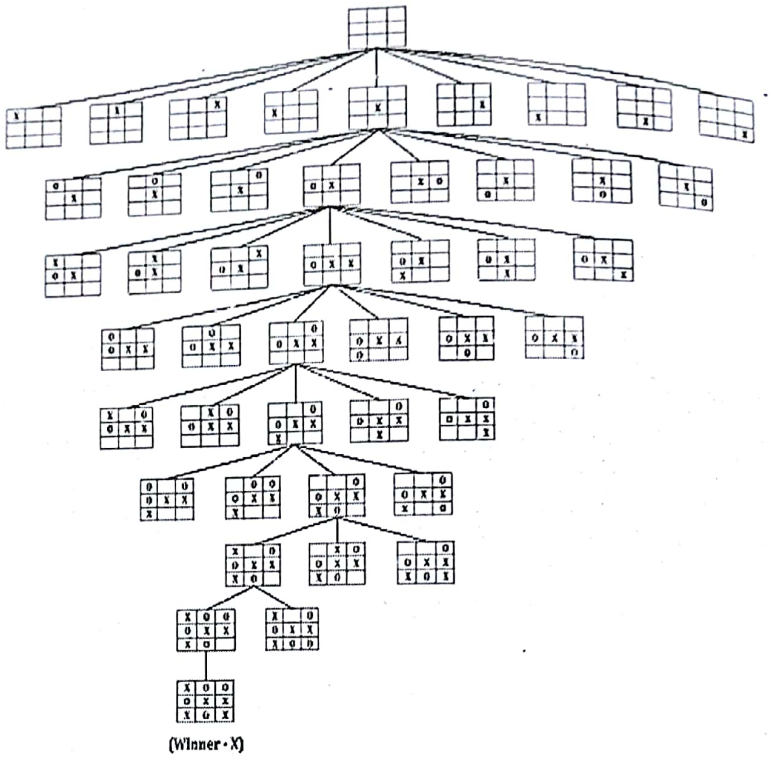


Figure 3.11: Game Tree for Tic-Tac-Toe Problem.

Q14] Explain with example various uninformed search techniques.

Ans: [10M – May17]

*** Note: We have explained each uninformed search technique in detail. Cut short it as per your understanding for 5 or 10 Marks whichever is asked in exam. ***

UNINFORMED SEARCH TECHNIQUES:

1. Uniformed Search Algorithm generates the search tree without using any domain specific knowledge.
2. It is also known as **Blind Search Technique.**
3. Uniformed search techniques operates in **brute-force way.**
4. It includes:
   a. Breadth-First Search.
   b. Depth-First Search.
   c. Depth-Limited Search.
   d. Iterative Deepening Depth-First Search.

I) **Breadth-First Search (BFS):**

- BFS is uninformed search technique.
- It is used for solving problem by Tree search.
- Only tree is specified without any additional knowledge of favorable nodes.
- Its uses two Queues for its implementation.
  - Open Queue.
  - Close Queue or Visited Queue.
- Successors are added in Open Queue from Back End [FIFO Queue].

Algorithm:

1. Create a single member Queue comprising of Root Node.
2. If first member of Queue is Goal, then go to step 5.
3. It first member of Queue is not Goal then remove it from the Queue and add to the Visited or Closed Queue.
   Consider its Successor if any, and add them to the Queue from Back/Rear End [FIFO].
4. If the Queue is not empty, then go to step 2.
   If the Queue is empty, then go to step 6.
5. Print "SUCCESS" & Stop.
6. Print "FAILURE" & Stop.

Performance of BFS:

- **Completeness:** Yes.
- **Optimality:** It gives shallowest goal. Optimal, if path cost is not decreasing with depth.
- **Space Complexity:** $O(b^{d+1})$
- **Time Complexity:** $O(b^{d+1})$ (d: Depth of the tree & b: Branching factor of tree)

Example:

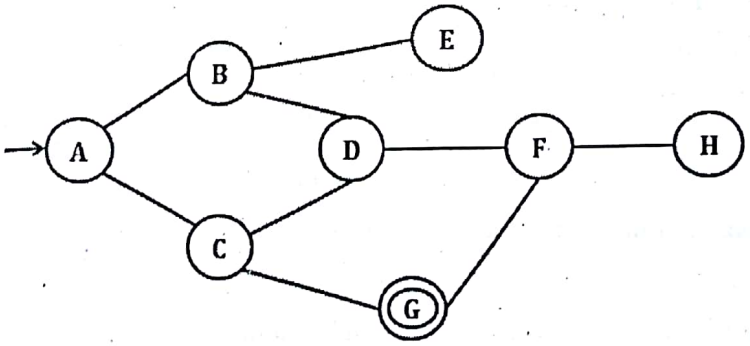


Figure 3.12: BFS Example.

Consider the Example as shown in figure 3.12. Here 'A' is the Initial Node & 'G' is the Goal.

Open Queue	Close Queue
[A]	[]
[B C]	[A]
[C D E]	[A B]
[D E G]	[A B C]
[E G F]	[A B C D]
[G F]	[A B C D E]
[G F]	[A B C D E G]
"Success"	

II) Depth-First Search (DFS):

- DFS is an **uninformed search technique.**
- Depth First Search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph.
- It starts from root node of the search tree and going deeper and deeper until a goal node is found, or until it hits a node that has no children.
- Then the search backtracks, returning to the most recent node it hasn't finished exploring.
- Its uses two Queues for its implementation.
  - Open Queue.
  - Close Queue or Visited Queue.
- Successors are added in Open Queue from Front End [LIFO Queue].

Algorithm:

1. Create a single member Queue comprising of Root Node.
2. If first member of Queue is Goal, then go to step 5.
3. It first member of Queue is not Goal then remove it from the Queue and add to the Visited or Closed Queue.
   Consider its Successor if any, and add them to the Queue from Front End [LIFO].
4. If the Queue is not empty, then go to step 2.
   If the Queue is empty, then go to step 6.
5. Print "SUCCESS" & Stop.
6. Print "FAILURE" & Stop.

Performance of DFS:

- **Completeness:** No. (Due to dead end)
- **Optimality:** No.
- **Space Complexity:** O (b * m) Where m = maximum depth.
- **Time Complexity:** O (b^m)

Example: Consider the Example as shown in Question 18.

III) **Depth-Limited Search (DLS):**

- Depth Limited Search is used to avoid Dead-End problem of DFS.
- It puts Limit on maximum depth to which search is allowed.
- Beyond that limit, search is not performed.
- It then explores other branch.
- Depth of each state is recorded as it is generated.
- When picking the next state to expand, only those with depth less or equal than the current depth are expanded.
- Once all the nodes of a given depth are explored, the current depth is incremented.

Algoritlim:

1. Create a single member Queue comprising of Root Node.
2. If first member of Queue is Goal, then go to step 5.
3. It first member of Queue is not Goal then remove it from the Queue and add to the Visited or Closed Queue.
   Consider its Successor if any, and add them to the Queue from Front End [LIFO].
4. If the Queue is not empty, then go to step 2.
   If the Queue is empty, then go to step 6.
5. Print "SUCCESS" & Stop.
6. Print "FAILURE" & Stop.

Performance of DLS:

- **Completeness:** No. (Due to dead end)
- **Optimality:** Yes if l >= d
- **Space Complexity:** O (b * l) Where l= Depth limit.
- **Time Complexity:** O (b^l)

Example:

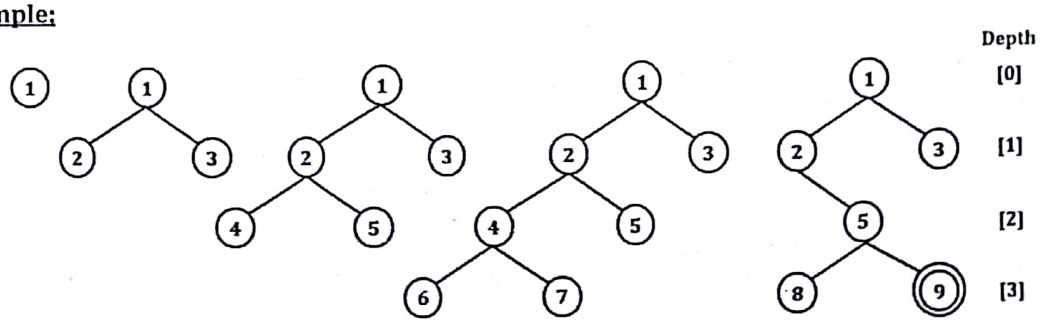


Figure 3.13: DLS Example.

Consider the Example as shown in figure 3.13. Here '1' is the Initial Node & '9' is the Goal.
Since the Depth Limit is 3, it explores the node till 3rd depth.

IV) **Iterative Deepening Depth-First Search:**

- It is variation of DFS.
- It combines the advantages of BFS like completeness & optimality.
- The depth limit is iteratively increased, starting from 0 till Goal is reached.
- Hence, it becomes complete and optimal like BFS.
- Also its memory space requirement is less like DFS.
- Successors are added in Open Queue from Front End [LIFO Queue].

Algorithm:

1. Create a single member Queue comprising of Root Node.
2. If first member of Queue is Goal, then go to step 5.
3. It first member of Queue is not Goal then remove it from the Queue and add to the Visited or Closed Queue.
   Consider its Successor if any, and add them to the Queue from Front End [LIFO].
4. If the Queue is not empty, then go to step 2.
   If the Queue is empty, then go to step 6.
5. Print "SUCCESS" & Stop.
6. Print "FAILURE" & Stop.

Performance:

- **Completeness:** Yes.
- **Optimality:** Yes (If path cost is not decreasing with depth).
- **Space Complexity:** O (b * d) Where b = branching factor & d = depth of the solution.
- **Time Complexity:** O (b^d)

Example:

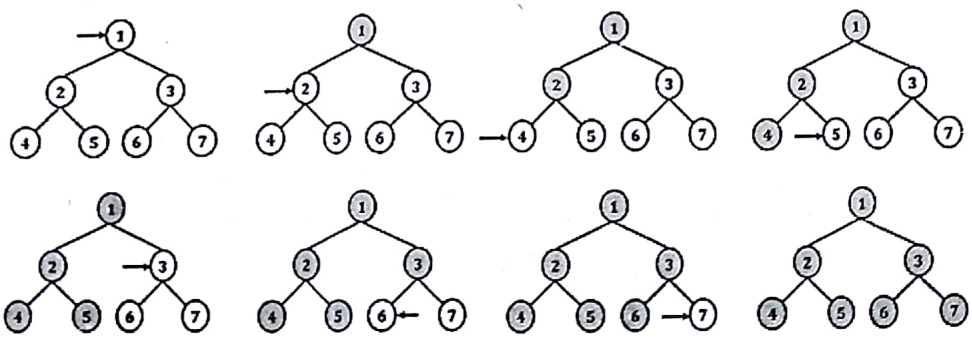


Figure 3.14: Iterative Deepening Depth First Search Example.

Consider the Example as shown in figure 3.14. Since the Depth Limit is 2, it explores the node till 2nd depth.

Q15] Compare following informed searching algorithms based on performance measure with justification: Complete, Optimal, Time Complexity and Space Complexity.

(i) Greedy Best First.
(ii) A*
(iii) Recursive Best-First (RBFS)

Q16] Compare Greedy Best first search and A* search algorithms based on performance measure with justification: Complete, Optimal, Time and Space complexity

Ans: **[Q15 | 10M – May16] & [Q16 | 10M – Dec17]**

GREEDY BEST FIRST:

Greedy best-first search expands the nodes that appears to be closest to the goal.

Performance Measure	Greedy Best First	Justification
Complete	No	Can get Stuck in loops.
Optimal	No	Usually Path Selected by Algorithm is Longest as compared to optimal Solution.
Time Complexity	Best: O(d) and Worst: $O(b^m)$	But a good heuristic can give dramatic improvement. (m = Maximum Depth)
Space Complexity	$O(b^m)$	Keeps all nodes in memory.

A*:

A* Algorithm avoid expanding paths that are already expansive.

Performance Measure	A*	Justification
Complete	Yes	Unless there are infinitely many nodes with $F \leq F(G)$
Optimal	Yes	If the heuristic is admissible and No Algorithm with the same heuristic is guaranteed to expand fewer nodes.
Time Complexity	$O(b^d)$	Exponential.
Space Complexity	$O(b^d)$	Keeps all nodes in memory.

RECURSIVE BEST-FIRST (RBFS):

RBFS keeps the track of Evaluation Value of the best alternative path available.

Performance Measure	RBFS	Justification
Complete	Yes	It is Better Efficient than IDA*
Optimal	Yes	If the heuristic is admissible
Time Complexity	O(bd)	Exponential.
Space Complexity	O(bd)	Keeps all nodes in memory.

Q17] Given a full 4-gallon jug and an empty 3- gallon jug, the goal is to fill the 4-gallon jug with exactly 2-gallons of water. Give state space representation.

Ans: [10M – Dec16]

1. The state space for this problem can be represented by set of ordered pairs of integers (X, Y)
2. Where,
   a. X represents the quantity of water in the 4 - gallon jug.
   b. Y represents the quantity of water in 3 - gallon jug.
3. **State Space = (X, Y)**
4. **Such that:**

   $X = \{0, 1, 2, 3, 4\}$ & $Y = \{0, 1, 2, 3\}$
5. As given the start state is: **Start State: (4, 0)**
6. The goal state is 2 gallons of water in the 4 gallon jug: **Goal State: (2, 0)**
7. Now we have to generate production rules for the water jug problem.

PRODUCTION RULES:

Rule	State	Process
1	(X, Y | X < 4)	(4, Y) {Fill 4 gallon jug}
2	(X, Y | Y < 3)	(X, 3) {Fill 3 gallon jug}
3	(X, Y | X > 0)	(0, Y) {Empty 4 gallon jug}
4	(X, Y | Y > 0)	(X, 0) {Empty 3 gallon jug}
5	(X, Y | 0 < X + Y ≥ 4 and Y > 0)	(4, Y - (4 - X)) { Pour water from 3 gallon jug to fill 4 gallon jug }
6	(X, Y | 0 < X + Y ≥ 3 and X > 0)	(X - (3 - Y), 3) { Pour water from 4 gallon jug to fill 3 gallon jug }
7	(X, Y | 0 < X + Y ≤ 4 and Y ≥ 0)	(X + Y, 0) { Pour all of water from 3 gallon jug into 4 gallon jug
8	(X, Y | 0 < X + Y ≤ 3 and X ≥ 0)	(0, X + Y) { Pour all of water from 4 gallon jug into 3 gallon jug
9	(0, 2)	(2, 0) {Pour 2 gallon water from 3 gallon jug into 4 gallon j

INITIALIZATION:

Start State: (4, 0)
Apply Rule 3: (X, Y | X > 0) → (0, 0)

{Empty 4 - gallon jug}
Now the state is (0, 0)

Iteration 1:

Start State: (0, 0)
Apply Rule 2: (X, Y | Y < 3) → (X, 3)

{Fill 3 - gallon jug}
Now the state is (X, 3)

Iteration 2:

Current State: (X, 3)
Apply Rule 7: (X, Y | 0 < X + Y ≤ 4 and Y ≥ 0) → (X+Y, 0)

{Pour all water from 3-gallon jug into 4-gallon jug}
Now the state is (3, 0)

Iteration 3:

Current State: (3, 0)
Apply Rule 2: (X, Y | Y < 3) → (3, 3)

{Fill 3-gallon jug}
Now the state is (3, 3)

Iteration 4:

Current State: (3, 3)
Apply Rule 5: (X, Y | 0 < X + Y ≥ 4 and Y > 0) → (4, Y - (4 - X))

{Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full}
Now the state is (4, 2)

Iteration 5:

Current State: (4, 2)
Apply Rule 3: (X, Y | X > 0) → (0, Y)

{Empty 4-gallon jug}
Now state is (0, 2)

Iteration 6:

Current State: (0, 2)
Apply Rule 9: (0, 2) → (2, 0)

{Pour 2 gallon water from 3 gallon jug into 4 gallon jug}
Now the state is (2, 0)

Goal Achieved.

STATE SPACE TREE:

Figure 3.15 represents the state space tree for the problem.

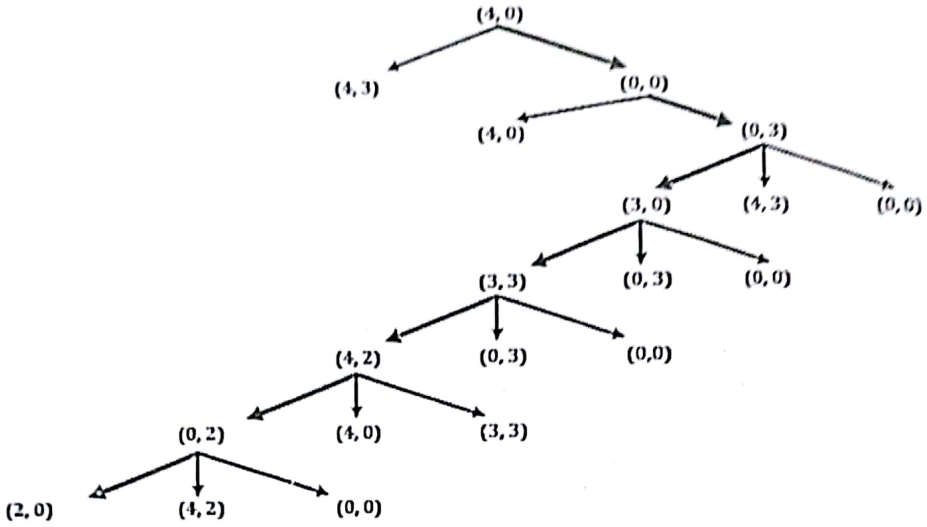


Figure 3.15: State Space Tree.

Q18] Give the initial state, goal test, successor function, and cost function for the travelling salesperson problem (TSP), there is a map involving N cities some of which are connected by roads. The aim is to find the shortest tour that starts from a city, visits all the cities exactly once and comes back to the starting city.

Ans: **[6M – Dec16]**

TRAVELLING SALESPERSON PROBLEM (TSP):

1. The Travelling Salesman Problem (TSP) is a classic algorithmic problem in the field of computer science.
2. It is focused on **optimization.**
3. In this context better solution often means a solution that is cheaper.
4. TSP is a mathematical problem.
5. It is most easily expressed as a graph describing the locations of a set of nodes.

EXAMPLE:

Consider the following graph of cities as shown in figure 3.16.

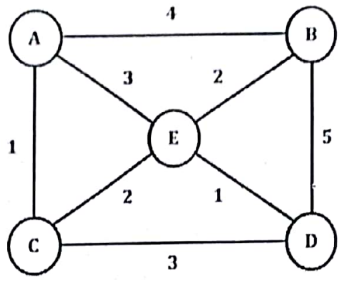


Figure 3.16: Graph of Cities.

FORMULATION:

States: Cities.

Initial state: A

Successor function: Travel from one city to another connected by a road.

Goal test: The trip visits each city only once that starts and ends at A.

Path cost: Traveling time.

Q19] Give the initial state, goal test, successor function, and cost function for the following problem "You have to colour a planar map using only 4 colours, in such a way that no two adjacent regions have the same colour"

[5M – Dec17]

Ans:

FOUR COLOUR THEOREM:

The four color theorem, or the four color map theorem, states that, given any separation of a plane into contiguous regions, producing a figure called a map, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color.

FORMULATION:

Initial State: Planar map with no regions colored.

Goal Test: All regions of the map are colored and no two adjacent regions have the same color.

Successor function: Choose an uncolored region and color it with a color that is different from all adjacent regions.

Cost function: Could be 1 for each color used or number of assignments.

Q20] Consider the given instance of 8-puzzle.

1	2	3
4	5	6
7	8	

Goal State

1	2	3
4	6	
7	5	8

Initial State

Compare and contrast uninformed search strategies with respect to solving 8-puzzle problem.

Ans: [10M – Dec16]

8-PUZZLE PROBLEM:

1. In the 8-puzzle problem we have a 3 × 3 square board and 8 numbered tiles.
2. The board has one blank position.
3. Blocks can be slid to adjacent blank positions.
4. We can alternatively and equivalently look upon this as the movement of the blank position up, down, left or right.
5. The objective of this puzzle is to move the tiles starting from an initial position and arrive at a given goal configuration.

COMPARISON OF UNIFORMED SEARCH ALGORITHM:

Criterion	Breadth First	Uniform Cost	Depth First	Depth Limited	Iterative Deepening
Complete	Yes [a]	Yes [a, b]	No	No	Yes [a]
Time	$O(b^d)$	$O(b^{1+[C^*/E]})$	$O(b^m)$	$O(b^L)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+[C^*/E]})$	O (bm)	O (bL)	O (bd)
Optimal	Yes [c]	Yes	No	No	Yes [c]

Where,

b → Branching Factor.

d → Depth of the shallowest solution.

m → Maximum Depth.

L → Depth Limit.

Superscript caveats are as follows:

a → Complete if 'b' is finite.

b → Complete if step costs ≥ ε

c → Optimal if step costs are all identical.

Q21] Local Search Algorithms

Ans: **[10M – May18]**

LOCAL SEARCH ALGORITHMS:

1. In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution.
2. In such cases, we can use **local search algorithms.**
3. Local search algorithms can start from a prospective solution and then move to a neighboring solution.
4. It can return a valid solution even if it is interrupted at any time before they end.
5. Local search algorithm includes:

I) **Hill Climbing Search:**

➢ Refer Q1.

II) **Local Beam Search:**

➢ Local Beam Search is a **heuristic search algorithm.**

➢ It explores a graph by expanding the most promising node in a limited set.

➢ Local Beam search is an optimization of best-first search that reduces its **memory requirements.**

➢ In this algorithm, it holds k number of states at any given time.

➢ At the start, these states are generated randomly.

➢ The successors of these k states are computed with the help of objective function.

➢ If any of these successors is the maximum value of the objective function, then the algorithm stops.

➢ Otherwise the (initial k states and k number of successors of the states = 2k) states are placed in a pool.

➢ The pool is then sorted numerically.

➢ The highest k states are selected as new initial states.

➢ This process continues until a maximum value is reached.

III) **Simulated Annealing:**

➢ Simulated Annealing (SA) is an effective and general form of **optimization.**

➢ It is useful in finding global optima in the presence of large numbers of local optima

➢ Annealing is the process of heating and cooling a metal to change its internal structure for modifying its physical properties.

➢ When the metal cools, its new structure is seized, and the metal retains its newly obtained properties.

- In simulated annealing process. the temperature is kept variable.
- We initially set the temperature high and then allow it to 'cool' slowly as the algorithm proceeds.
- When the temperature is high, the algorithm is allowed to accept worse solutions with high frequency.

IV) Travelling Salesman Problem:

- Refer Q18.

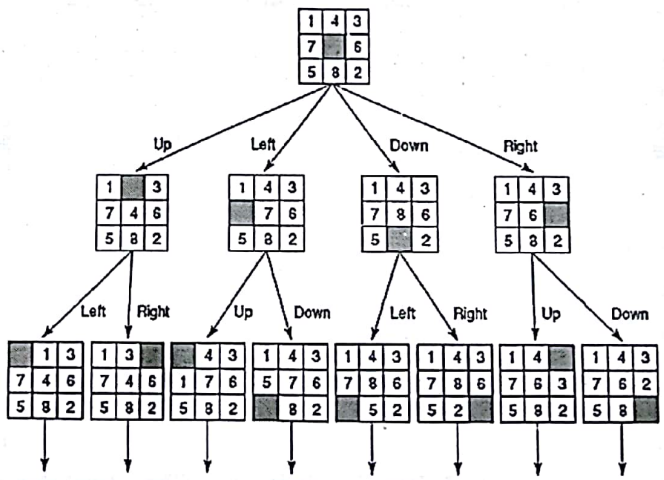
Q22] Give state space representation for 8 puzzle Problem. What are possible Heuristic functions for it?

Ans: **[5M – May18]**

8-PUZZLE PROBLEM:

1. In the 8-puzzle problem we have a 3 × 3 square board and 8 numbered tiles.
2. The board has one blank position.
3. Blocks can be slid to adjacent blank positions.
4. We can alternatively and equivalently look upon this as the movement of the blank position up, down, left or right.
5. The objective of this puzzle is to move the tiles starting from an initial position and arrive at a given goal configuration.

STATE SPACE REPRESENTATION FOR 8 PUZZLE PROBLEM:



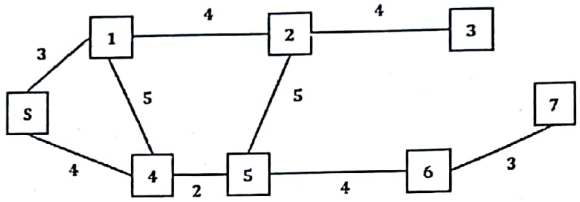
8-PUZZLE PROBLEM HEURISTIC FUNCTION:

Refer Q5.

Q23] Consider the graph given below in Figure below.

Assume that the initial state is S and the goal state is 7. Find a path from the initial state to the goal state using A* Search. Also report the solution cost. The straight line distance heuristics estimates for the nodes are as follows:

H (1) = 14, H (2) = 10, H (3) = 8, H (4) = 12, H (5) = 10, H (6) = 10, H (S) = 15

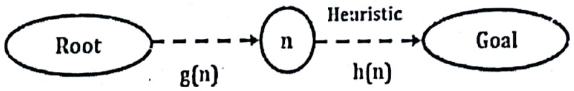


[10M – Dec15]

Ans:

A* ALGORITHM:

1. A* Algorithm is an informed Search Technique.
2. It uses additional information beyond problem formulation or tree.
3. A* Algorithm uses priority Queue.
4. It uses heuristics function H (n) as well as G (n) for evaluation.



Note: h(n) – Cost from Node n to Goal

$$F(n) = G(n) + H(n)$$

EXAMPLE:

Consider the figure 3.17 shown below. Here 'S' is the initial state & '7' is the goal state.

The heuristic/cost is given along branches & links.

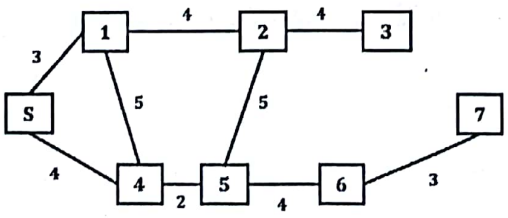
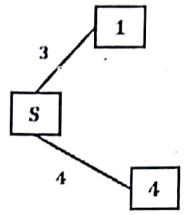
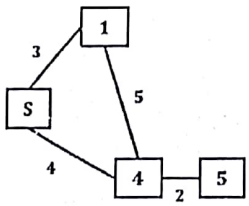
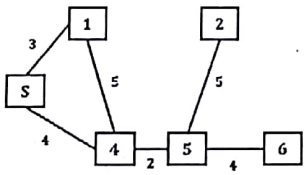
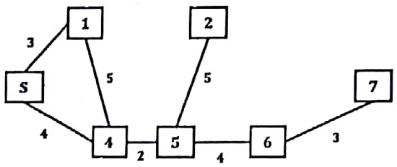


Figure 3.17: Example of A*

STEPS:

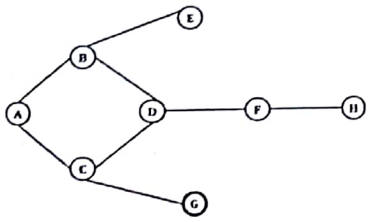
	Tree	Cost	Open Queue	Close Queue
1		F (1) = G (1) + H (1) = 3 + 14 = 17 F (4) = G (4) + H (4) = 4 + 12 = 16	[S] [1 4]	[S] [S 4]
2		F (1) = G (1) + H (1) = 5 + 14 = 19 F (5) = G (5) + H (5) = 2 + 10 = 12	[S] [S 4] [S 4 1 5]	[S] [S 4] [S 4 5]
3		F (2) = G (2) + H (2) = 5 + 10 = 15 F (6) = G (6) + H (6) = 4 + 10 = 14	[S] [S 4] [S 4 5] [S 4 5 2 6]	[S] [S 4] [S 4 5] [S 4 5 6]
4		F (7) = G (7) + H (7) = 3 + 0 = 3	[S] [S 4] [S 4 5] [S 4 5 6] [S 4 5 6 7]	[S] [S 4] [S 4 5] [S 4 5 6] [S 4 5 6 7]

"Success" Hence we have found the Optimal Solution for the problem.

Optimal Cost = 4 + 2 + 4 + 3 = 13

Optimal Path = S → 4 → 5 → 6 → 7

Q24] Consider the graph given in Figure below. Assume that the initial state is A and the goal state is G. Find a path from the initial state to the goal state using DFS. Also report the solution cost.



[10M –May16]

Ans:

DFS:

1. DFS Stands for **Depth First Search.**
2. DFS is an **uninformed search technique.**
3. Depth First Search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph.
4. It starts from root node of the search tree and going deeper and deeper until a goal node is found, or until it hits a node that has no children.
5. Then the search backtracks, returning to the most recent node it hasn't finished exploring.

EXAMPLE:

Consider the figure 3.18 shown below. Here 'A' is the initial state & 'G' is the goal state.

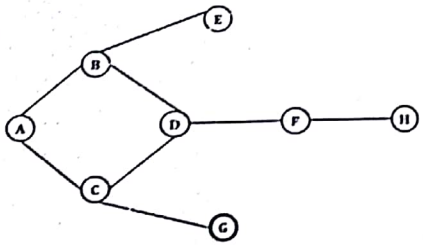
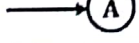
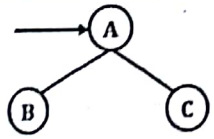
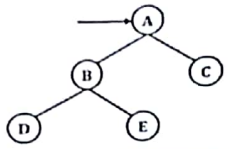
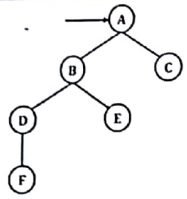
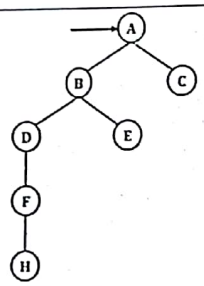
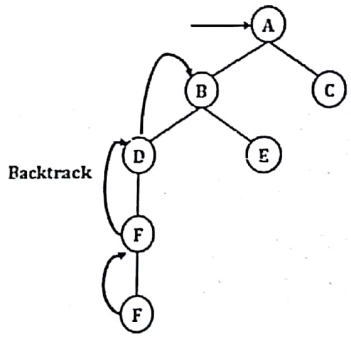
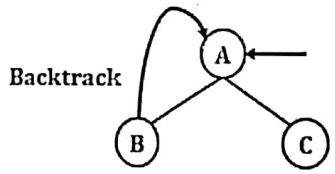
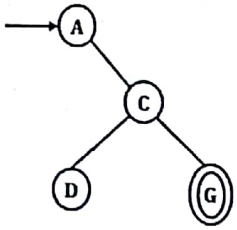


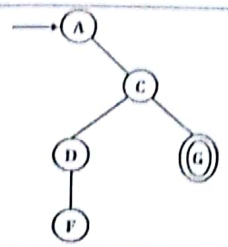
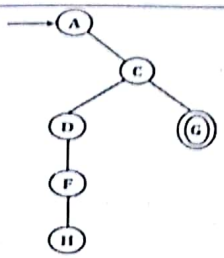
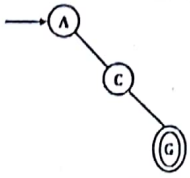
Figure 3.18: DFS Example.

Assuming that the alphabetically smaller node is expanded first to break ties.

STEPS:

	Tree Representation	Open Queue	Close Queue
1		[A]	[ ]
2		[B C]	[A]

3		[D E C]	[A B]
4		[F E C]	[A B D]
5		[H E C]	[A B D F]
	Since Node 'H' Does not have any Child, hence Backtrack.		
6		[E C]	[A B]
7		[C]	[A]
8		[D G]	[A C]

9		[F G]	[A C D]
10		[H G]	[A C D F]
Since Node 'H' Does not have any Child, hence Backtrack.			
11		[G]	[A C]
Success.... Goal Reached			

DFS Path: A → C → G

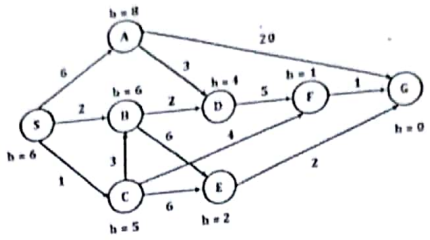
Suppose the Cost Between the nodes be as shown in table below

State	Next State	Cost
A	B	4
A	C	1
B	D	3
B	E	8
C	D	2
C	G	6
D	F	3
F	H	2

Therefore Solution Cost of Path

A → C → G = 1 + 6 = 7

Q25] Consider the search problem below with start state S and goal state G. The transition costs are next to the edges and the heuristic values are next to the states. What is the final cost using A * search.

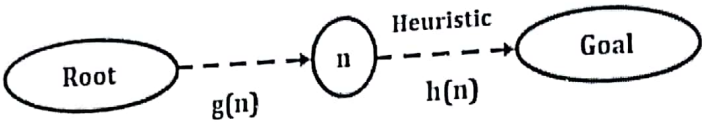


[10M – May17]

Ans:

A* ALGORITHM:

1. A* Algorithm is an **Informed Search Technique.**
2. It uses additional information beyond problem formulation or tree.
3. A* Algorithm uses priority Queue.
4. It uses heuristics function H (n) as well as G (n) for evaluation.



Note: h(n) – Cost from Node n to Goal

$$F(n) = G(n) + H(n)$$

EXAMPLE:

Consider the figure 3.19 shown below. Here 'S' is the initial state & 'G' is the goal state.

The heuristic/cost is given along branches & links.

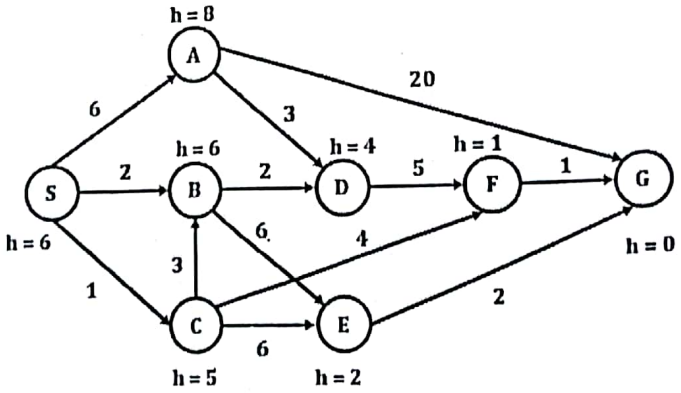
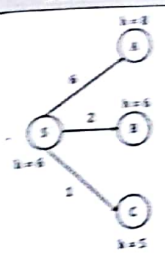
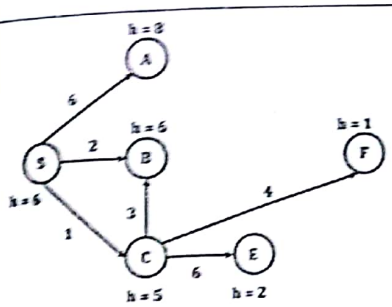
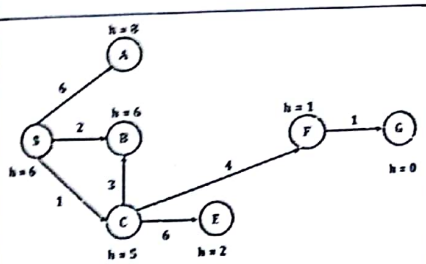


Figure 3.19: Example of A*

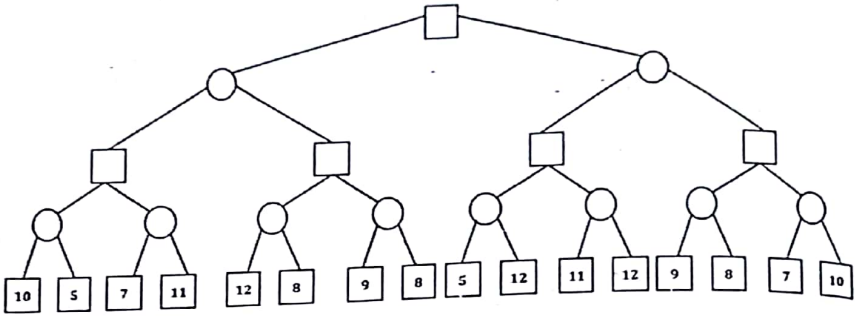
STEPS:

	Tree	Cost	Open Queue	Close Queue
1		F (A) = G (A) + H (A) = 6 + 8 = 14 F (B) = G (B) + H (B) = 2 + 6 = 8 F (C) = G (C) + H (C) = 1 + 5 = 6	[S] [S A B C]	[S] [S C]
2		F (B) = G (B) + H (B) = 3 + 6 = 9 F (F) = G (F) + H (F) = 4 + 1 = 5 F (E) = G (E) + H (E) = 6 + 2 = 8	[S] [S C] [S C B F E]	[S] [S C] [S C F]
3		F (G) = G (G) + H (G) = 1 + 0 = 1	[S] [S C] [S C F] [S C F G]	[S] [S C] [S C F] [S C F G]
	"Success" Hence we have found the Optimal Solution for the problem.			

Optimal Path = S → C → F → G

Optimal Cost = 1 + 4 + 1 = 6

Q26] Apply alpha-Beta pruning on example given in Figure below considering first node as max.



[10M – May16]

Ans:

ALPHA – BETA PRUNING:

1. Alpha – Beta Pruning is a search algorithm.
2. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.).
3. It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move.
4. Such moves need not be evaluated further.
5. Initially, $\alpha = -\infty$ and $\beta = \infty$
6. This Algorithm Prunes the Branches When Value of $\alpha \geq \beta$

EXAMPLE:

- Let Assign the alphabet for the above given example.
- Square indicates Max Node & Circle indicates Min Node.
- 'α' Value can be changed using Max Node only, similarly 'β' can be changed using Min Node only.

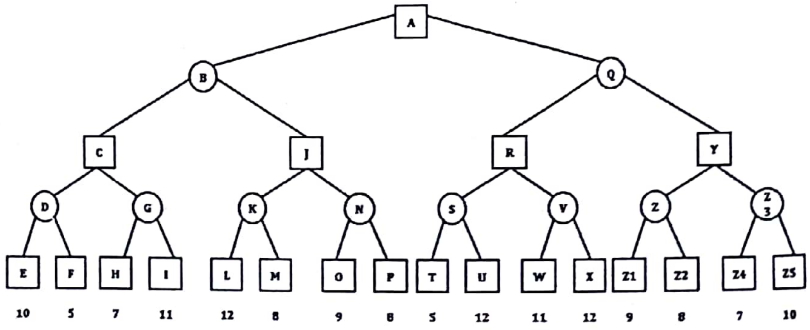


Figure 3.20: Alpha – Beta Pruning Example.

Node	Type	α	β	Score	Remarks
A	Max	-∞	∞		Initially, α = - ∞ and β = ∞
B	Min	-∞	∞		Depth First Traversal, Assigning α = - ∞ and β = ∞
C	Max	-∞	∞		Depth First Traversal, Assigning α = - ∞ and β = ∞
D	Min	-∞	∞		Depth First Traversal, Assigning α = - ∞ and β = ∞
E	Max			10	No More Sub node, Hence Assigning Value = 10 and Backtrack
D	Min	-∞	10		Since D is Min Node, Assigning β = 10
F	Max			5	No More Sub node, Hence Assigning Value = 5 and Backtrack
D	Min			5	D is the Parent Node of E & F, Min Value is 5. Hence Assigning Value = 5 and Backtrack
C	Max	5	∞		Since C is Max Node, Assigning α = 5
G	Min	5	∞		Depth First Traversal, Assigning α = 5
H	Max			7	No More Sub node, Hence Assigning Value = 7 and Backtrack
G	Min	5	7		Since G is Min Node, Assigning β = 7
I	Max			11	No More Sub node, Hence Assigning Value = 11 and Backtrack
G	Min			7	G is the Parent Node of H & I, Min Value is 7. Hence Assigning Value = 7 and Backtrack
C	Max			7	C is the Parent Node of D & G, Max Value is 7. Hence Assigning Value = 7 and Backtrack
B	Min	-∞	7		Since B is Min Node, Assigning β = 7
J	Max	-∞	7		Depth First Traversal, Assigning α = - ∞ and β = 7
K	Min	-∞	7		Depth First Traversal, Assigning α = - ∞ and β = 7
L	Max			12	No More Sub node, Hence Assigning Value = 12 and Backtrack
K	Min	-∞	7		Since 7 is less than 12, assigning β = 7
M	Max			8	No More Sub node, Hence Assigning Value = 8 and Backtrack
K	Min			8	K is the Parent Node of L & M, Max Value is 8. Hence Assigning Value = 8 and Backtrack
J	Max	8	7		Since J is Max Node, Assigning α = 8
J	Max			8	Since **α ≥ β.** Therefore Purge the Branch. And assigning value = 8
B	Min			7	B is the Parent Node of C & J, Min Value is 7. Hence Assigning Value = 7 and Backtrack
A	Max	7	∞		Depth First Traversal, Assigning α = 7 and β = ∞
Q	Min	7	∞		Depth First Traversal, Assigning α = 7 and β = ∞
R	Max	7	∞		Depth First Traversal, Assigning α = 7 and β = ∞
S	Min	7	∞		Depth First Traversal, Assigning α = 7 and β = ∞
T	Max			5	No More Sub node, Hence Assigning Value = 5 and Backtrack
S	Min	7	5		Since S is Min Node, Assigning β = 5
S	Min			5	Since **α ≥ β.** Therefore Purge the Branch. And assigning value = 5
R	Max	7	∞		Since 7 greater than 5, Assigning α = 7
V	Min	7	∞		Depth First Traversal, Assigning α = 7 and β = ∞
W	Max			11	No More Sub node, Hence Assigning Value = 11 and Backtrack

V	Min	7	11		Since G is Min Node, Assigning β = 11
X	Max			12	No More Sub node, Hence Assigning Value = 12 and Backtrack
V	Min			11	Since 11 is smaller than 12, Assigning β = 11 and backtrack
R	Max			11	R is the Parent Node of S & V, Max Value is 11. Hence Assigning Value = 11 and Backtrack
Q	Min	7	11		Since Q is Min Node, Assigning β = 11
Y	Max	7	11		Depth First Traversal, Assigning α = 7 and β = 11
Z	Min	7	11		Depth First Traversal, Assigning α = 7 and β = 11
Z1	Max			9	No More Sub node, Hence Assigning Value = 9 and Backtrack
Z	Min	7	9		Since Z is Min Node, Assigning β = 9
Z2	Max			8	No More Sub node, Hence Assigning Value = 8 and Backtrack
Z	Min			8	Z is the Parent Node of Z1 & Z2, Min Value is 8. Hence Assigning Value = 8 and Backtrack
Y	Max	8	11		Since Y is Max Node, Assigning α = 8
Z3	Min	8	11		Depth First Traversal, Assigning α = 8 and β = 11
Z4	Max			7	No More Sub node, Hence Assigning Value = 7 and Backtrack
Z3	Min	8	7		Since Z3 is Min Node, Assigning β = 7
Z3	Min			7	Since α ≥ β. Therefore Purge the Branch. And assigning value = 7 and backtrack
Y	Max			8	Y is the Parent Node of Z & Z3, Max Value is 8. Hence Assigning Value = 8 and Backtrack
Q	Min			8	Q is the Parent Node of R & Y, Min Value is 8. Hence Assigning Value = 8 and Backtrack
A	Max			8	A is the Parent Node of B & Q, Max Value is 8. Hence Assigning Value = 8 and Backtrack

Figure 3.21 shows Final Tree.

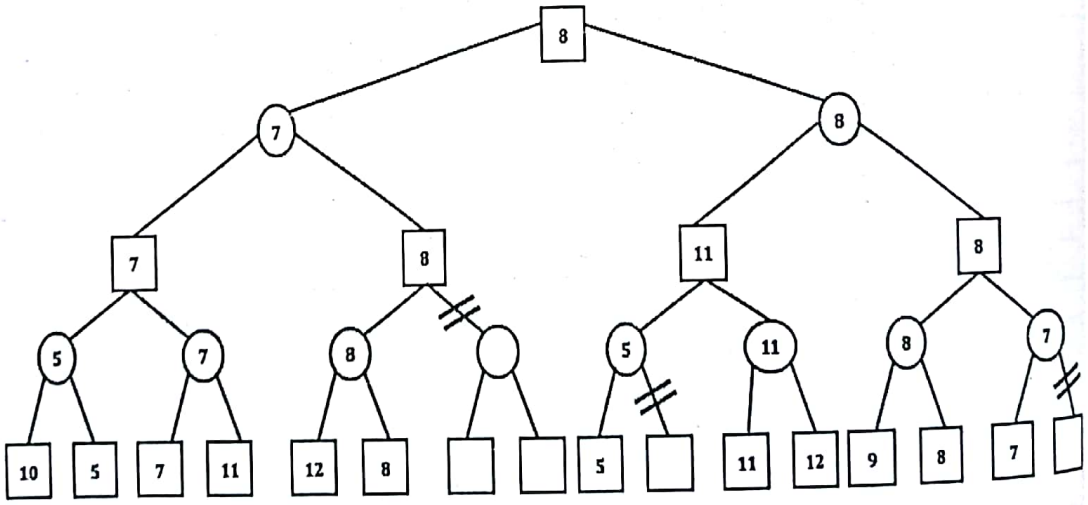
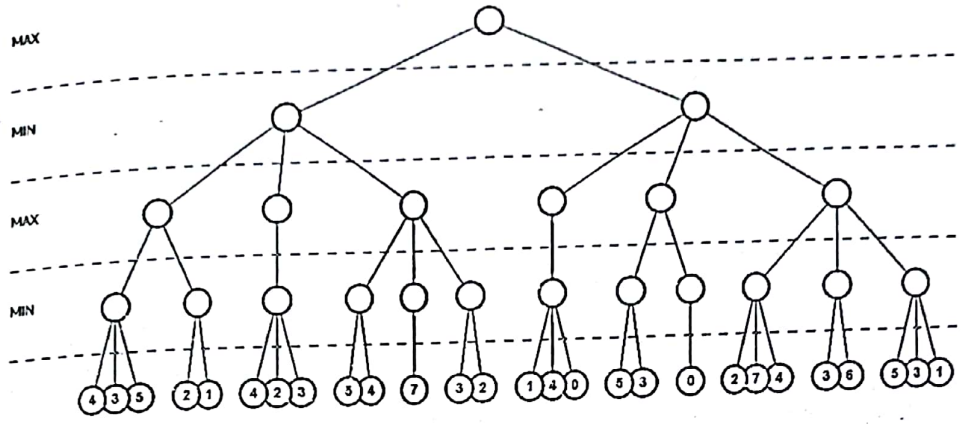


Figure 3.21: Final Tree.

Q27] Apply alpha-beta pruning on the following example considering first node as MAX.



[10M – Dec16 & May17]

Ans:

ALPHA – BETA PRUNING:

Refer Q26.

EXAMPLE:

- Let Assign the alphabet for the above given example.
- 'α' Value can be changed using Max Node only, similarly 'β' can be changed using Min Node only.
- The solution to the above problem is shown in figure 3.22.

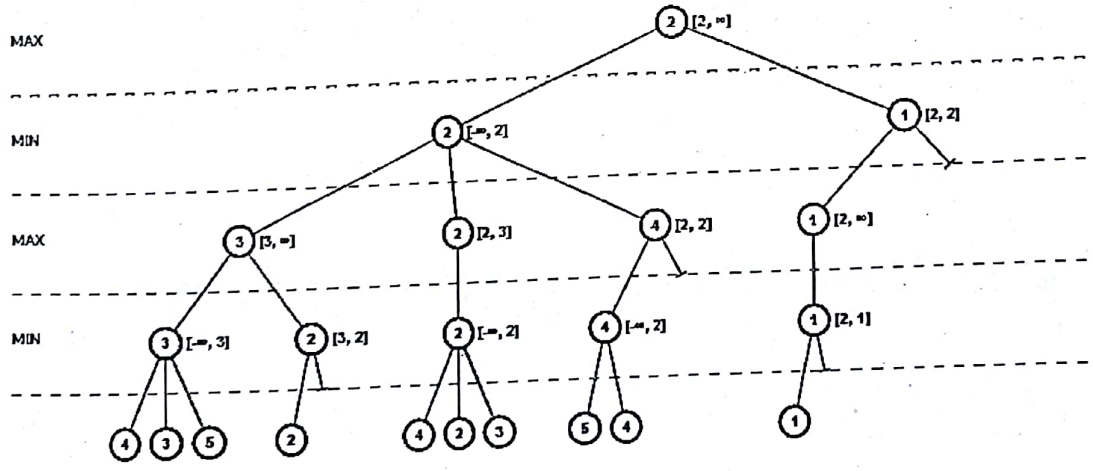
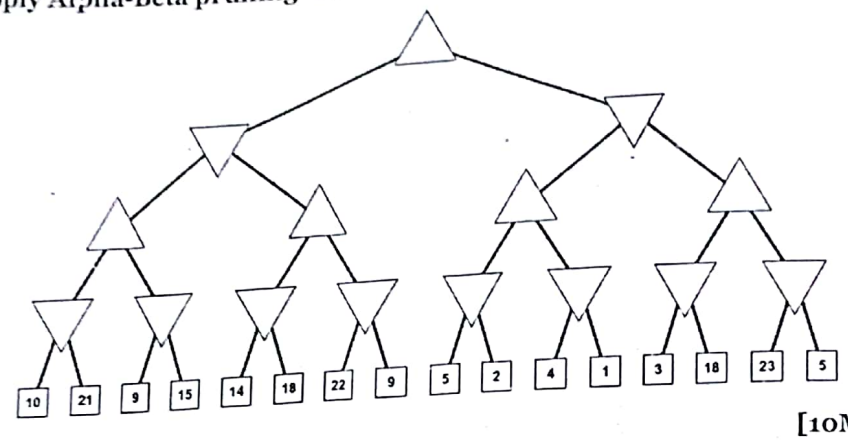


Figure 3.22: Alpha-Beta Pruning.

Q28] Apply Alpha-Beta pruning on following example considering first node as MAX



[10M – May18]

Ans:

ALPHA – BETA PRUNING:

Refer Q26.

EXAMPLE:

- Let Assign the alphabet for the above given example.
- 'α' Value can be changed using Max Node only, similarly 'β' can be changed using Min Node only.
- The solution to the above problem is shown in figure 3.23.

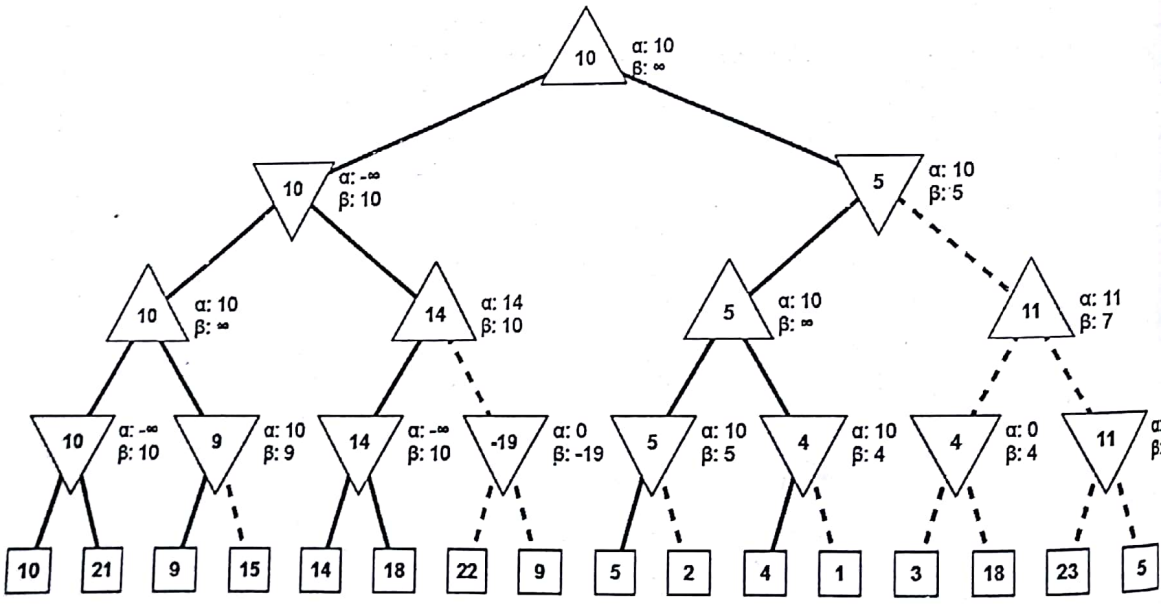
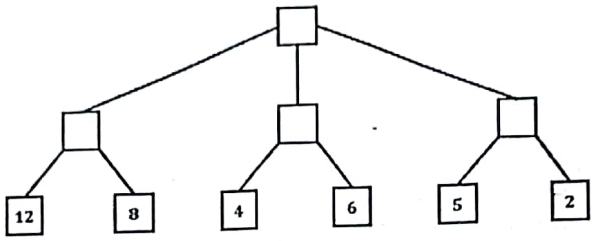


Figure 3.23: Alpha-Beta Pruning.

Q29] Write a pseudo code for alpha-beta algorithm. Apply alpha-beta pruning on example given in Figure 1 considering first node as max



[10M – Dec17]

Ans:

ALPHA – BETA PRUNING:

Refer Q26.

PSEUDO CODE:

```
Function minimax (Node, depth, isMaximizingPlayer, Alpha, Beta):
{
        If node is a leaf node:
                Return value of the node

        If isMaximizingPlayer:
                BestVal = - INFINITY
                For each child node:
                        Value = minimax (Node, depth+1, false, Alpha, Beta)
                        BestVal = max (BestVal, Value)
                        Alpha = max (Alpha, BestVal)
                        If Beta ≤ Alpha:
                        Break
                Return BestVal
        Else:
                BestVal = + INFINITY
                For each child node:
                        Value = minimax (Node, depth+1, true, Alpha, Beta)
                        BestVal = min (BestVal, Value)
                        Beta = min (Beta, BestVal)
                        If Beta ≤ Alpha:
                        Break
                Return BestVal
}
```

EXAMPLE:

- Let Assign the alphabet for the above given example.
- 'α' Value can be changed using Max Node only, similarly 'β' can be changed using Min Node only.
- The solution to the above problem is shown in figure 3.24.

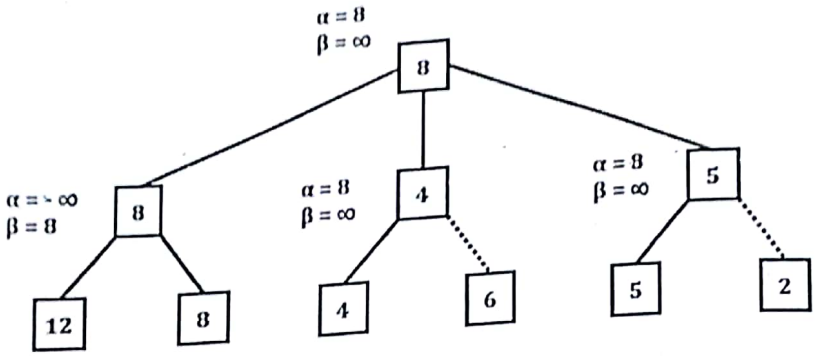
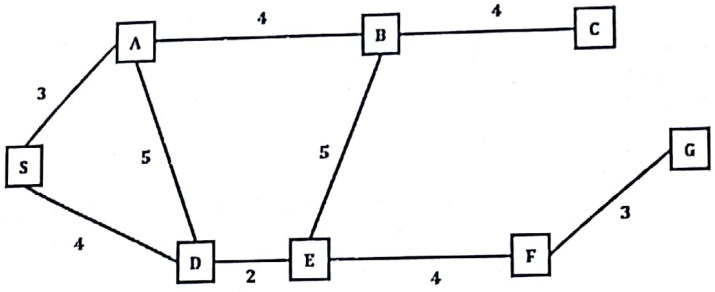


Figure 3.24: Alpha-Beta Pruning.

Q30] Consider the graph given in Figure below. Assume that the initial state is A and the goal state is G. Show how Greedy Best first Search would create a search tree to find a path from the initial state to the goal state:

At each step of the search algorithm, show which node is being expanded, and the content of fringe. Also report the eventual solution found by the algorithm, and the solution cost. Assuming the straight-line distance as the heuristics function:

h(S)=10.5, h(A)=10, h(B)=6, h(C)=4, h(D)=8, h(E)=6.5, h(F)=3 and h(G)=0



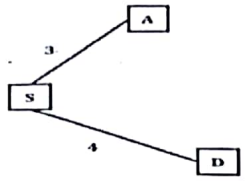
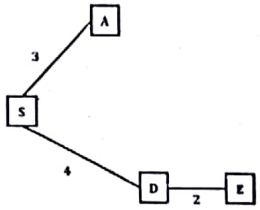
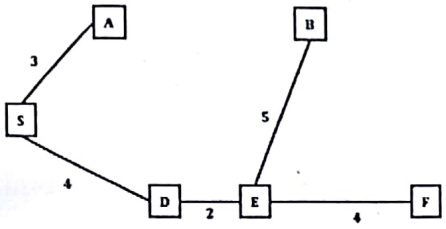
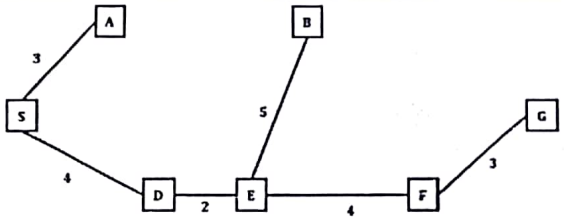
[10M – Dec17]

Ans:

GREEDY BEST FIRST SEARCH:

1. Greedy Best First Search is **Informed (Heuristic) Search Technique.**
2. It is a basic search that uses path-cost to determine the next node to expand.
3. It expands the node that is estimated to be closest to goal.
4. It expands nodes based on f(n) = h(n).
5. It is implemented using **priority queue.**

EXAMPLE:

	Tree	Cost	Open Queue	Close Queue
1		F (A) = H (A) = 10 F (D) = H (D) = 8	[S] [A D]	[S] [S D]
2		F (E) = H (E) = 6.5	[S] [A D] [S D E]	[S] [S D] [S D E]
3		F (B) = H (B) = 6 F (B) = H (F) = 3	[S] [A D] [S D E] [S D E B F]	[S] [S D] [S D E] [S D E B]
4		F (G) = H (G) = 0	[S] [A D] [S D E] [S D E B F] [S D E B G]	[S] [S D] [S D E] [S D E B] [S D E B G]
	"Success" Hence we have found the Solution for the problem.			

Cost = 8 + 6.5 + 3 + 0 = 17.5

Optimal Path = S → D → E → F → G

--- EXTRA QUESTIONS ---

Q1] Uniform Cost Search?

Ans:

UCS:

1. UCS is same as BFS.
2. It is uninformed search technique.
3. It is used to find worst case space & time complexity.
4. Uniform cost search is a search algorithm used to traverse, and find the shortest path in weighted trees and graphs.
5. Uniform Cost Search or UCS begins at a root node and will continually expand nodes, taking the node with the smallest total cost from the root until it reaches the goal state.
6. Uniform cost search doesn't care about how many steps a path has, only the total cost of the path.
7. UCS with all path costs equal to one is identical to breadth first search.
8. Let, $C^* \rightarrow$ Optimal Cost & $E \rightarrow$ Smallest step cost.
9. Maximum no. of steps = C^* / E

PERFORMANCE OF BFS:

- **Completeness:** Yes.
- **Optimality:** Yes.
- **Space Complexity:** $O(b^{1 + [C^* / E]})$
- **Time Complexity:** $O(b^{1 + [C^* / E]})$

EXAMPLE:

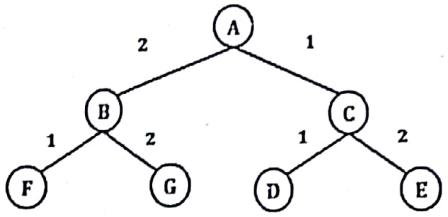


Figure 3.25: UCS Example.

Consider the Example as shown in figure 3.25. Here 'A' is the Initial Node & 'G' is the Goal.

Open Queue	Close Queue
[A] 0	[ ]
[B C] 2 1	[A]

[B D E] 2 2 3	[A C]
[D E F G] 2 3 3 4	[A C B]
[E F G] 3 3 4	[A C B D]
[F G] 3 4	[A C B D E]
[G] 4	[A C B D E F]
"Success"	

Q2] Best First Search?

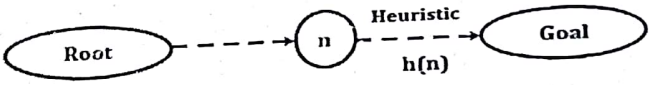
Ans:

BEST FIRST SEARCH:

1. Best First Search is an informed search technique.
2. Additional knowledge in terms of heuristic function is made available.
3. It uses Priority Queue.
4. New members are added in the Queue in the ascending order at Evaluation Function f (n).
5. Best First Search uses heuristic function h (n) itself as Evaluation Function.

$$f(n) = h(n)$$

6. Heuristic Function h (n) gives estimated cost for reaching Goal from Node "n"



Note: h(n) – Cost from Node n to Goal

ALGORITHM:

1. Create a single member Queue comprising of Root Node.
2. If first member of Queue is Goal, then go to step 5.
3. It first member of Queue is not Goal then remove it from the Queue.
   Consider its Successor if any, and add them to the Queue in ascending order of Evaluation Function f (n).
4. If the Queue is not empty, then go to step 2.
   If the Queue is empty, then go to step 6.
5. Print "SUCCESS" & Stop.
6. Print "FAILURE" & Stop.

ADVANTAGES & LIMITATIONS:

- Best First Search is complete but not optimum.
- It considers only h (n) to explore nodes.

PERFORMANCE OF BEST FIRST SEARCH:

- **Completeness:** No (Can get stuck in loops).
- **Optimality:** No.
- **Space Complexity:** O (b^m)
- **Time Complexity:** O (b^m)

EXAMPLE:

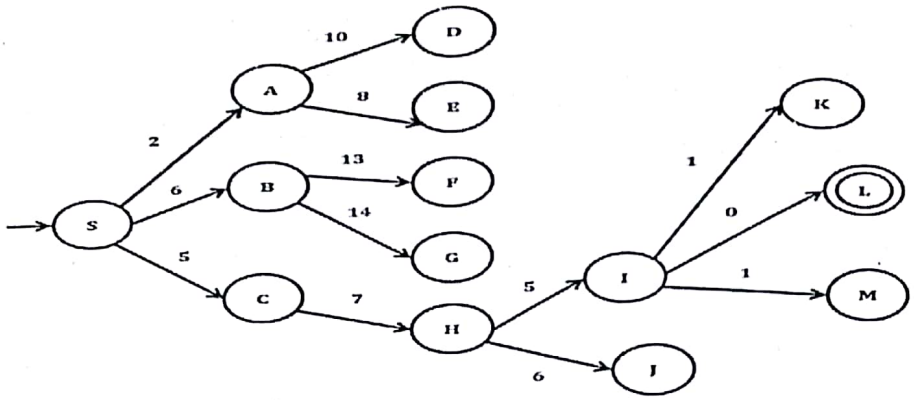


Figure 3.26: Best First Search Example.

Consider the Example as shown in figure 3.26. Here 'S' is the Initial Node & 'L' is the Goal.

Open Queue	Close Queue
[S] 0	[ ]
[A C B] 2 5 6	[S]
[C B E D] 5 6 8 10	[S A]
[B H E D] 6 7 8 10	[S A C]
[H E D F G] 7 8 10 13 14	[S A C B]
[I J H E D F G] 5 6 7 8 10 13 14	[S A C B H]
[L K M I J H E D F G] 0 1 1 5 6 7 8 10 13 14	[S A C B H I]
[K M I J H E D F G] 1 1 5 6 7 8 10 13 14	[S A C B H I L]
"Success"	

Q3] IDA* Algorithm?

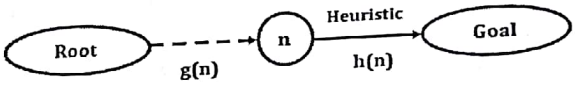
Ans:

IDA*:

1. IDA* stands for Iterative Deeping A* Algorithm.
2. It is used to remove memory space problem of A* Algorithm.
3. A* generates the nodes without using them.
4. It never opens nodes for which f (n) > C*.
5. Hence, they need to be flushed out from memory. But C* is not known Beforehand.
6. IDA* iteratively increases limit of C* starting from zero till Goal is reached.
7. It is Informed Search & uses Priority Queue.
8. New members are added in the Queue in the ascending order at Evaluation Function f(n).
9. IDA* uses heuristic function h (n) as well as g (n) for Evaluation.

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

10. Heuristics Function h (n) gives estimated cost of reaching Goal from node n. while g(n) gives cost of reaching node n from root.



Note: h(n) – Cost from Node n to Goal

ALGORITHM:

1. Create a single member Queue comprising of Root Node.
2. If first member of Queue is Goal, then go to step 6.
3. It first member of Queue is not Goal then remove it from the Queue.
   Consider its Successor if any, and add them to the Queue in ascending order of Evaluation Function f (n).
4. If the Queue is not empty, then go to step 2.
   If the Queue is empty, then go to step 7.
5. Nodes for which f (n) > C is flushed.
6. Print "SUCCESS" & Stop.
7. Print "FAILURE" & Stop.

ADVANTAGES:

- IDA* algorithm is complete.
- It is optimal due to admissible heuristic.
- It is memory efficient.

LIMITATIONS:

- Due to its iterative nature it takes more time to search.

EXAMPLE:

Same as A* Algorithm. Show the Flushing for f (n) > C* at end.