PROGRAMME DESIGN COMMITTEE

Prof. (Retd.) S.K. Gupta , IIT, Delhi
Prof. Ela Kumar, IGDTUW, Delhi
Prof. T.V. Vijay Kumar JNU, New Delhi
Prof. Gayatri Dhingra, GVMITM, Sonipat
Mr. Milind Mahajan,. Impressico Business Solutions, New Delhi
Sh. Shashi Bhushan Sharma, Associate Professor, SOCIS, IGNOU
Sh. Akshay Kumar, Associate Professor, SOCIS, IGNOU
Dr. P. Venkata Suresh, Associate Professor, SOCIS, IGNOU
Dr. V.V. Subrahmanyam, Associate Professor, SOCIS, IGNOU
Sh. M.P. Mishra, Assistant Professor, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU

COURSE DESIGN COMMITTEE

Prof. T.V. Vijay Kumar, JNU, New Delhi
Prof. S.Balasundaram, JNU, New Delhi
Prof D.P. Vidyarthi, JNU, New Delhi
Prof. Anjana Gosain, USICT, GGSIPU, New Delhi
Dr. Ayesha Choudhary, JNU, New Delhi
Sh. Shashi Bhushan Sharma, Associate Professor, SOCIS, IGNOU
Sh. Akshay Kumar, Associate Professor, SOCIS, IGNOU
Dr. P. Venkata Suresh, Associate Professor, SOCIS, IGNOU
Dr. V.V. Subrahmanyam, Associate Professor, SOCIS, IGNOU
Sh. M.P. Mishra, Assistant Professor, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU

SOCIS FACULTY

Prof. P. Venkata Suresh, Director, SOCIS, IGNOU
Prof. V.V. Subrahmanyam, SOCIS, IGNOU
Dr. Akshay Kumar, Associate Professor, SOCIS, IGNOU
Dr. Naveen Kumar, Associate Professor, SOCIS, IGNOU (on EOL)
Dr. M.P. Mishra, Associate Professor, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU
Dr. Manish Kumar, Assistant Professor, SOCIS, IGNOU

PREPARATION TEAM

Dr.Sudhansh Sharma, (Writer- Unit 1)
Assistant Professor SOCIS, IGNOU
(Unit-1 : Partially Adapted from MCSE003
Artificial Intelligence & Knowledge Management)

Mr. Anant Kumar Jayswal, (Writer – Unit 2 & Unit 3)
Assistant Professor
Amity School of Engineering and Technology

Prof. Arvind (Writer Unit-4)
Department of Mathematics,Hansraj College
University of Delhi

Prof Ela Kumar (Content Editor)
Department of Computers & Engg. IGDTUW, Delhi

Prof.Parmod Kumar (Language Editor)
SOH, IGNOU, New Delhi

Course Coordinator: Dr.Sudhansh Sharma,

Print Production

Sh Sanjay Aggarwal,Assistant Registrar, MPDD

, 2022

ISBN-

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110068.
Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by MPDD, IGNOU.

UNIT 1 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Structure

1.1 INTRODUCTION

Today, artificial intelligence is used in a wide variety of applications, including engineering, technology, the military, opinion mining, sentiment analysis, and many more. It is also used in more advanced domains, such as language processing and applications for aerospace.

AI is everywhere in today's world, and people are gradually becoming accustomed to its presence. It is utilised in systems that recognise both voices and faces. In addition to this, it can provide you with shopping recommendations that are tailored to your own purchasing preferences. Finding spam and preventing fraudulent use of credit cards is made much easier when you have this skill. The most cutting-edge technology currently on the market are virtual assistants like Apple's Siri, Amazon's Alexa, Microsoft's Cortana, and Google's own Google Assistant. It's possible that you're already familiar with the technology involved in artificial intelligence (AI). Are you?

AI has become very popular all over the world today. It imitates human intelligence in machines by programming them to do the same things people do. As a technology, AI is going to have a bigger impact on how people live their daily lives. Everyone wants to connect to Artificial Intelligence as a technology these days. Before we can understand AI, we need to know and talk about some basic things. For example, what is the difference between knowledge and

intelligence? The key to starting this unit is the answer to this question.

The accumulation of information and abilities that a person has gained through their life experiences is known as knowledge. While intelligence refers to one's capacity to put one's knowledge into practise. To put it simply, knowledge is what we have learned over the years, and it expands as time passes. Because of this, it represents the culmination of everything that we have realised over the course of our lives. It is important to highlight that having information does not automatically make one intelligent; rather, intelligence is what makes one smart.

There is a well-known proverb that says "marks are not the measure of intelligence." This is due to the fact that intelligence is not a measurement of how much information one possesses. In fact, it is the measure of how much we comprehend and put into practise. People who are knowledgeable may gain a lot of information, but an intelligent person understands how to comprehend, analyze, and use the information. You could have a lot of knowledge but still be the least intelligent person in the room. Knowledge and intelligence are inextricably linked, and each contributes to the other's development. Knowledge enables one to learn the understandings that others have of things, whereas intelligence is the foundation for one's ability to grasp the things themselves.

Now that we have an understanding of the distinction between intelligence and knowledge, our next issue is: what exactly is artificial intelligence? Incorporating intelligence into a machine is related to the field of Artificial Intelligence, whereas both concepts, namely the representation of knowledge and its engineering, are the basis of traditional AI research. This topic will be discussed in section 1.3 of this unit, but in a nutshell, incorporating intelligence into a machine is related to the field of Artificial Intelligence. Knowledge engineering is a subfield of artificial intelligence (AI) that applies rules to data in order to simulate the way in which an expert would think about the information. It does this by analysing the structure of a job or a decision in order to figure out how one arrives at a conclusion.

The subsequent units of this course, you will learn about some of the concepts that are essential for knowledge representation, such as frames, scripts, and other related topics. In addition, this course will address the issues that are associated with the knowledge representation for uncertain situations, such as employing the method of fuzzy logic, rough sets, and the Dempster Shafer theory, among other relevant topics.

Some of the prerequisites to get started with this subject.

a) Strong understanding of Basic concepts of Mathematics viz. Algebra, Calculus, probability andStatistics.
b) Experience in programming using Python or Java.
c) A good understanding of algorithms.
d) Reasonably good data analytics skills.
e) Fundamental knowledge in discrete mathematics.
f) Finally, internal will to learn

1.2 OBJECTIVES

After going through this unit, you will be able to:

- Understand the difference between knowledge and intelligence
- Answer – what is AI?
- Identify various approaches of AI
- Compare Artificial Intelligence (AI), Machine Learning (ML) & Deep Learning (DL).
- Understand the concept of agents in AI

1.3 BASICS OF ARTIFICIAL INTELLIGENCE?

Knowledge and intelligence are two important concepts, and we were able to gain an understanding of the fundamental distinction between the two terms. Now that we have your attention, let's talk about what artificial intelligence actually is. The meaning of Artificial Intelligence will be covered in this section; however, before we get started, it is important to note that the field of Artificial Intelligence is related to the process of incorporating intelligence into machines; the specifics of this process, as well as the mechanism itself, will be covered in this unit as well as the subsequent units of this training.

The following is a list of eight definitions of artificial intelligence that have been provided by well-known authors of artificial intelligence textbooks.

1) According to Haugeland in 1985, "The Exciting New Effort to Make Computers Think... Machines with Minds, in the Full and Literal Sense,"

2) According to Bellman, "the automation of behaviours that we connect with human thinking, activities such as decision-making, problem-solving, and learning..." 1978

3) "The study of mental capabilities through the application of computer models," (also known as "The Study of Mental Capabilities"), Charniak and McDermott's 1985.

4) According to Winston (1992), "the study of the calculations that make it possible to perceive, reason, and act."

5) "The art of building machines that execute functions that demand intellect when performed by people," as defined by Kurzweil in the year 1990.

5) "The art of building machines that execute functions that demand intellect when performed by people," as defined by Kurzweil in the year 1990. To the Rich and the Knight, 1991

7) According to Schalkoff (1990), "a field of study that aims to explain and replicate intelligent behaviour in terms of computational processes."

8) According to Luger and Stubblefield (1993), "the discipline of computer science that is concerned with the automation of intelligent behaviour."

According to the concepts presented earlier, there are four distinct objectives that might be pursued in the field of artificial intelligence. These objectives are as follows:

- The creation of systems that think in the same way as people do.
- The creation of systems that are capable of logical thought.
- The creation of machines that can mimic human behaviour.
- The creation of systems that behave in a logical manner

In addition, we discovered through our conversation in the earlier section 1.1 of this Unit, that Artificial Intelligence (AI) is the intelligence that is incorporated into machines; in other words, AI is the ability of a machine to display human-like capabilities such as reasoning, learning, planning, and creativity. We learned this information because AI is the ability of a machine to display human-like capabilities. Taking into mind the Emerging AI technologies to sense, interpret, and act according to the circumstances, relevant exemplary solutions are summarised in Figure 1, which attempts to encapsulate the understanding of the question "What is Artificial Intelligence?"

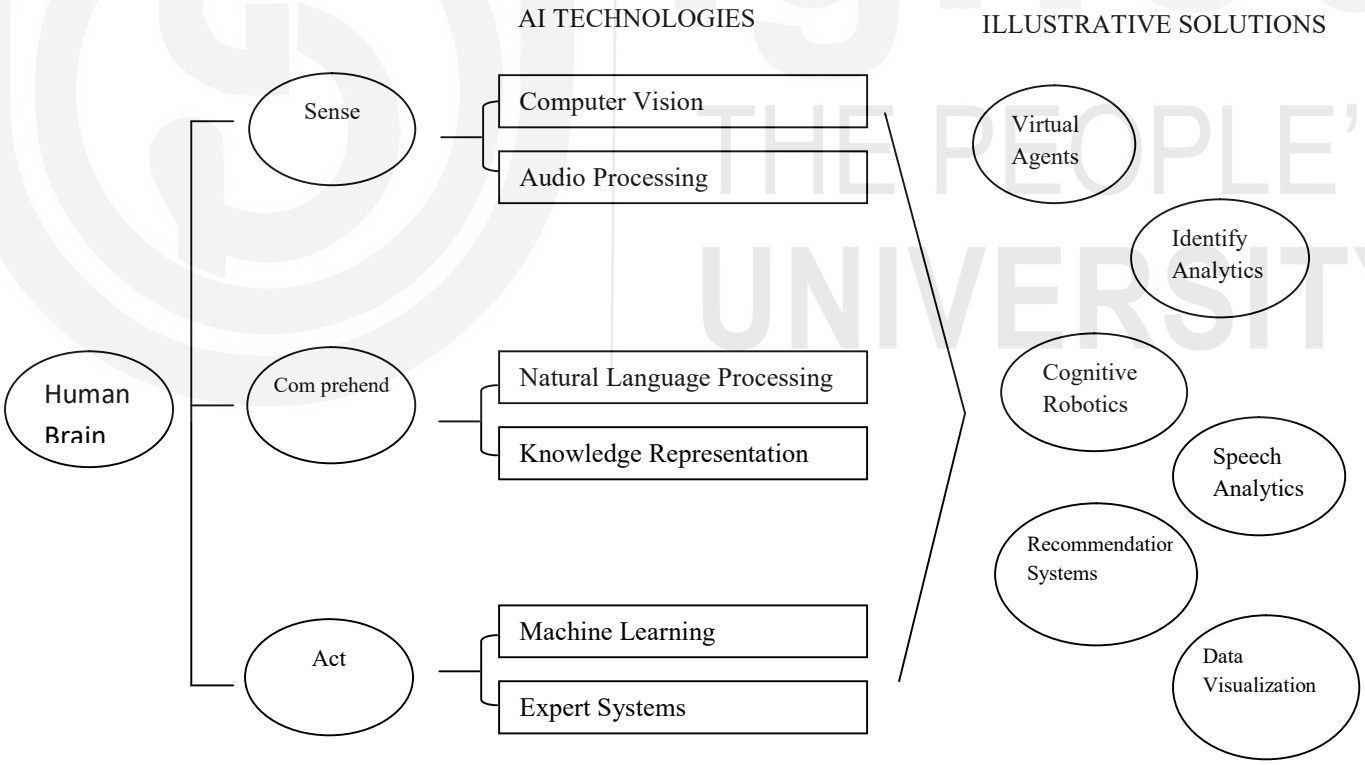


Figure-1: What is Artificial Intelligence? – Emerging AI Technologies

The applications of artificial intelligence (AI) that are shown in figure-1 do not all require the same kinds of AI. In a general sense, one may say that AI can be categorised into the following levels:

- software level and hardware level (i.e., Embodied AI). Where, the software level consists of things like search engines, virtual assistants, speech and facial recognition systems, picture analysis tools, and other things like that.
- Hardware level (Embedded) includes Robots, autonomous vehicles, drones, the Internet of Things, and other technologies fall under the category of embedded artificial intelligence (AI).

On the basis of the functionalities the AI can be classified based on Type 1 and Type 2

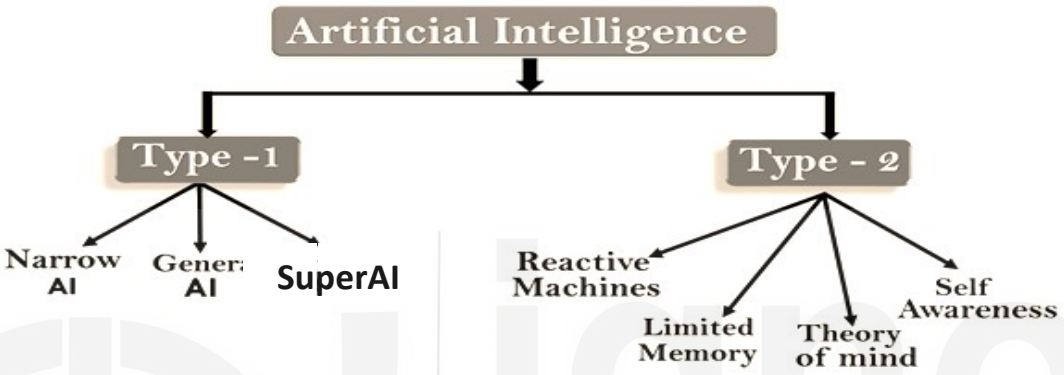


Figure 1(a) – Classification of Artificial Intelligence

Here's a brief introduction the first type of AI i.e., Type 1 AI. Following are the three stages of Type 1 - Artificial Intelligence:

a) Artificial Narrow Intelligence-(ANI)
b) Artificial General Intelligence-(AGI)
c) Artificial Super

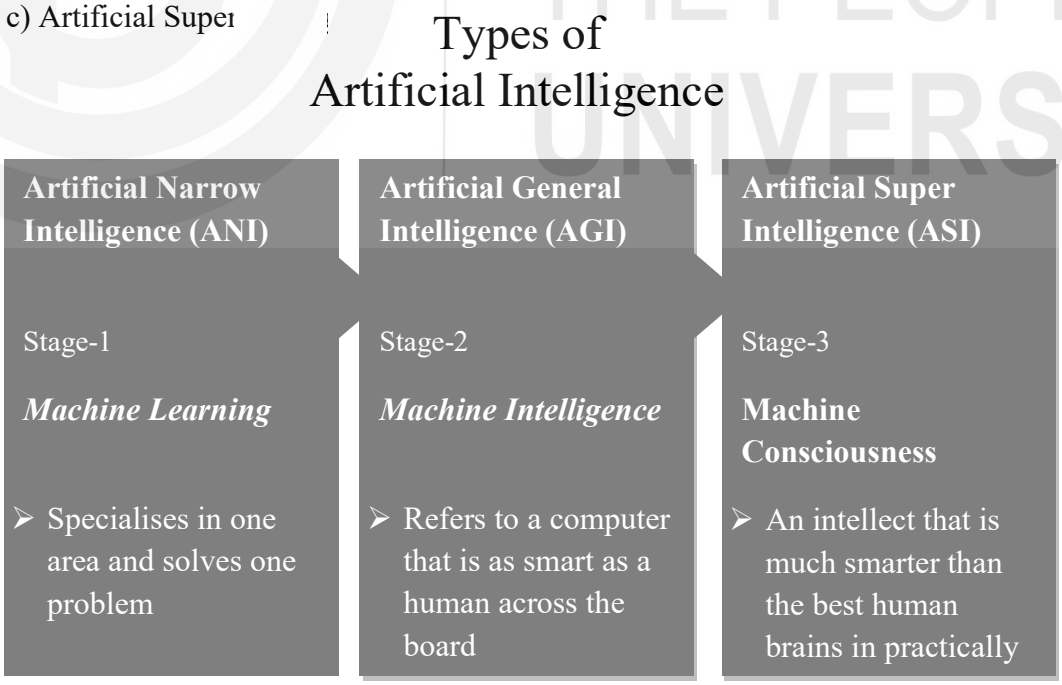


Figure 1(b) –Three Stages of Type-I Artificial Intelligence

The various categories of Artificial Intelligence are discussed as follows:

a) Artificial Narrow Intelligence (ANI), also called Weak AI or Narrow AI: Weak AI is a term for thinking that is "simulated." Such systems seem to act intelligently, but they don't have any awareness of what they are doing. For example, a chatbot might talk to you in a way that seems natural, but it doesn't know who it is or why it's talking to you. Artificial intelligence is a system that was built to do a certain job.

b) Artificial General Intelligence (AGI): Strong or General Artificial Intelligence, also called "actual" thinking. That is, acting like a smart human and thinking like one with a conscious, subjective mind. For instance, when two humans talk, they probably both know who they are, what they're doing, and why.

   Systems with strong or general artificial intelligence can do things that people can do. These systems tend to be harder to understand and more complicated. They are set up to handle situations where they might need to solve problems on their own without help from a person. Uses for these kinds of systems include self-driving cars and operating rooms in hospitals.

c) Artificial Super Intelligence (ASI) - Super intelligence: The term "super intelligence" usually refers to a level of general and strong AI that is smarter than humans, if that's even possible. The ASI is seen as the logical next step after the AGI because it can do more than humans can. This includes making decisions, making rational decisions, and even things like building emotional relationships. Their is a marginal difference between AGI and ASI.

☞ Check Your Progress 1

Q1 How Knowledge differs from intelligence? What do you understand by the term Artificial Intelligence (AI)? List the various technologies and their corresponding illustrative solutions.

……

……

……

Q2 Classify AI on the basis of the functionalities of AI

……

……

……

Q3 Compare ANI, AGI and ASI, in context of AI

……

……

1.4 BRIEF HISTORY - ARTIFICIAL INTELLIGENCE

AI's ideas come from early research into how people learn and think. Also very old is the idea that a computer could act like a person. Greek mythology is where the idea of machines that can think for themselves comes from.

• Aristotle, who lived from 384 BC to 322 BC, made a syllogistic logic system that was not formal. This is where the first formal system of deductive reasoning got its start.

At the start of the 17th century, Descartes said that animal bodies are just complex machines.

• Pascal made the first mechanical digital calculator in the year 1642.

In the 1800s, George Boole came up with a number system called "binary algebra" that showed (some) "laws of thought."

• Charles Babbage and Ada Byron worked on programmable mechanical calculators.

In the late 19th century and early 20th century, mathematicians and philosophers like Gottlob Frege, Bertram Russell, Alfred North Whitehead, and Kurt Godel built on Boole's first ideas about logic to make mathematical representations of logic problems.

When electronic computers came along, it was a big step forward in how we could study intelligence.

McCulloch and Pitts made a Boolean circuit model of the brain in 1943. They wrote about how neural networks can be used to do math in the paper "A Logical Calculus of Ideas Immanent in Nervous Activity."

• In 1950, Turing wrote a paper called "Computing Machines and Intelligence." This article gave a good overall picture of AI. To learn more about Alan Turing, go to http://www.turing.org.uk/turing.

Turing's paper talked about a lot of things, one of which was how to solve problems by using heuristics as guides to look through the space of possible solutions. He used the game of chess to explain how his ideas about how machines can think work. He even said that the machine could change its own instructions so that machines could learn from what they do.

The SNARC was built by Marvin Minsky and Dean Edmonds in 1951. It was the first randomly wired neural network learning machine (SNARC stands for Stochastic Neural Analog Reinforcement Computer). It was a computer with a network of 40 neurons and 3000 vacuum tubes.

Samuel made a number of programmes to help people play checkers between 1952 and 1956.

In 1956, Dartmouth was the site of a well-known meeting. At the conference, the people who came up with the idea of AI met for the first time. At this meeting, the name "Artificial Intelligence" was chosen.

• Newell and Simon's book The Logic Theorist came out. Many people think it was the first show to use artificial intelligence.

In 1959, Gelernter made a Geometry Engine. In 1961, James Slagle's Ph.D. dissertation at MIT was a programme called SAINT. It was written in LISP, and a first-year college student could use it to solve calculus problems.

Thomas Evan made a programme called "Analogy" in 1963 that could solve analogy problems like those on an IQ test. The first collection of articles about artificial intelligence was put together by Edward A. Feigenbaum and Julian Feldman. It was called "Computers and Thought." It was released in 1963.

In 1965, J. Allen Robinson came up with a way to prove things mechanically. He called it the Resolution Method. This made it possible for formal logic to work well as a language for representing programmes. In 1967, Feigenbaum, Lederberg, Buchanan, and Sutherland at Stanford showed how the Dendral programme could be used to understand the mass spectra of organic chemical compounds. This was the first programme that worked well and was based on scientific knowledge. The SRI robot Shakey showed in 1969 that it was possible to move, see, and solve problems all at the same time.

From 1969 to 1979, the first systems based on knowledge were set up in place.

• In 1974, MYCIN showed how powerful rule-based systems can be for representing and drawing conclusions about knowledge in medical diagnosis and treatment. For the Knowledge Representation Version 2 CSE IIT, Kharagpur, plans were made. There were also some frames that Minski had made. There are logic-based programming languages like Prolog and Planner. In the 1980s, Lisp Machines was made and sold. In 1985, neural networks were once again all the rage. In 1988, probabilistic and decision-theoretic methods were used again.

Early AI was based on general systems that didn't know much. AI researchers realised that for machines to be able to reason about complex tasks, they need to know a lot about a narrow field.

Dean Pomerleau made ALVINN at CMU in 1989. Autonomous Land Vehicle in a Neural Network is what ALVINN stands for. This is a system that learns to drive by watching someone else do it. It has a neural network that gets an image from a two-dimensional camera that is 30x32 units. The output layer tells the vehicle where it needs to go. The system drove a car from the East Coast to the West Coast of the United States, which is about 2,850 miles. A person only drove about 50 of these miles. The system took care of the rest.

In the 1990s, AI made a lot of progress, especially in machine learning, data mining, intelligent tutoring, case-based reasoning, multi-agent planning and scheduling, uncertain reasoning, understanding and translating natural language, vision, virtual reality, games, and other areas.

Rod Brooks' COG Project at MIT made a lot of progress toward making a humanoid robot with the help of a lot of people.

In the 1990s,

- 1997 was the year of the first official Robo-Cup soccer game. It was played on a tabletop with 40 teams of robots talking to each other.
- As more and more people use the web, web crawlers and other AI-based programmes that pull information from it are becoming more and more important.
- Deep Blue In 1997, Gary Kasparov, who was the world champion at the time, lost to IBM's Deep Blue chess programme.

In 2000,

- The Nomad robot goes to remote parts of Antarctica to look for meteorite samples.

- Space probes that are made of robots can work on their own to learn more about space. They keep an eye on what's going on around them, make decisions, and take action to get where they want to go. In April 2004, the first three-month missions of NASA's Mars rovers went well. The Spirit rover was looking at a group of hills on Mars that took two months to reach. It is finding strangely eroded rocks that might be new pieces of the puzzle that is the history of the area. Spirit's twin sister, Opportunity, was looking at the layers of rock in a crater.
- Internet agents: As the Internet grows quickly, more people want to use Internet agents to keep track of what users are doing, find the information they need, and figure out which information is the most useful. The reader can learn more about AI by reading about it in the news.

1.5 COMPONENTS OF INTELLIGENCE

According to the dominant school of thought in psychology, human intelligence should not be viewed as a singular talent or cognitive process but rather as a collection of distinct components. The majority of attention in the field of artificial intelligence research has been paid to the following aspects of intelligence: learning, reasoning, problem-solving, perception, and language comprehension.

Learning: There are numerous approaches to develop a learning system. Making mistakes is the simplest way to learn. A basic software that solves "mate in one" chess issues, for example, might test different moves until it finds one that answers the problem. The programme remembers which move worked so that the next time the computer is given the identical situation, it can provide an immediate response. The simple act of memorising things like answers to problems, words in a vocabulary list, and so on is known as "rote learning" or memorization.

We'll talk about another classification that doesn't depend on the way knowledge is represented or how it is represented. According to this system, there are five ways to learn:(ii)

(i) Rote Learning or memorising.
(ii) Learning by Instructions
(iii) Learning by analogy.
(iv) Learning by Induction
(v) Learning by deduction.

Rote learning Rote learning is the simplest method of learning since it involves the least amount of interpretation. The information is simply copied into a database in this method of learning. This is the technique for memorising multiplication tables.

On a computer, rote learning is relatively simple to implement. The difficulty of implementing what is known as generalisation is more difficult. Generalized learning allows the student to perform better in situations they haven't faced before. A programme that learns the past tenses of regular English verbs by rote will not be able to form the past tense of "jump" until it has been presented with "jumped" at least once, whereas a programme that can generalise from examples will be able to learn the "added" rule and thus form the past tense of "jump" even if it has never encountered this verb before. Modern techniques allow programmes to generalise complex rules based on data.

Learning by Instructions: The next method of learning is to be instructed. Because new knowledge must be contributed to an existing knowledge base in order to be useful, this type of learning necessitates greater inference. When a teacher instructs a student, this is the type of learning that occurs.

Learning by analogy: When you learn by analogy, you generate new ideas by connecting previously learned concepts. Textbooks frequently employ this method of instruction. For example, in the text, some problems are solved as examples, and students are subsequently given problems that are comparable to the examples. This type of learning also occurs when someone who can drive a light car attempts to drive a heavy vehicle.

Learning by Induction Learning through induction is the most common method of learning. This is a method of learning that employs inductive reasoning, which is a style of reasoning that involves drawing a conclusion from a large number of good instances. If we encounter a lot of cows, we might notice that they have four legs, are white, and have two horns in the same position on their head, for example. Even though inductive reasoning frequently leads to valid conclusions, the conclusions are not always unarguable. For example, with the above-mentioned concept of cow, we might come across a black cow, a three-legged cow who has lost one leg in an accident, or a single-horn cow.

Learning by deduction: Finally, we discuss deductive learning, which is founded on deductive inference, a non-debatable mode of thinking. By irrefutable method of reasoning, we mean that if the hypotheses (or given facts) are accurate, the conclusion arrived through deductive (i.e., any irrefutable) reasoning is always correct. This is the most common method of thinking in mathematics.

Inductive learning is a crucial component of an agent's learning architecture. An agent learns based on:

• What it is learning, such as concepts, problem-solving techniques, or game-playing techniques, etc.

• The representation, predicate calculus, frame, script, and other elements that were employed.

• The critic, who expresses their opinion of the agency in general.

Learning based on feedback is normally categorized as:

- Supervised learning
- Unsupervised learning
- Reinforcement Learning.

Supervised Learning: It has a function that learns from inputs and outputs that are shown as examples. Some examples of this kind of learning are figuring out useful things about the world from what you see, making a map from the current state's conditions to actions, and learning how the world changes over time.

Unsupervised Learning: There is no way to know what the inputs are and what the expected outputs are in this type of learning. So, the learning system has to figure out on its own which properties of objects it doesn't know about are important. For example, figuring out the shortest way to get from one city to another in a country you know nothing about.

Reinforcement (Rewards) for Learning: In some problems, the task or problem can only be seen, not said. Also, the job may be an ongoing one. The user tells the agent how happy or unhappy he or she is with the agent's work by sometimes giving the agent positive or negative rewards (i.e., reinforcements). The agent's job is to get as many rewards (or reinforcements) as possible. In a simple goal-attainment problem, the agent can be rewarded when it reaches the goal and punished when it doesn't.

You need an action plan to get the most out of this kind of task. But when it comes to tasks that never end, the future reward might be endless, making it hard to decide how to get the most out of it. One way to move forward in this kind of situation is to ignore future rewards after a certain point. That is, the agent may want rewards that will come soon more than rewards that will come a long time from now.

Delayed-reinforcement learning is the process of determining how to behave in situations when rewards are contingent on previous actions.

Reasoning: To reason is to draw conclusions that are appropriate for the situation. Both deductive and inductive reasoning can be used to make conclusions. An example of a deductive inference is, "Fred is either in the museum or in the cafe. He isn't in the cafe, so he must be in the museum." An example of inductive inference is, "In the past, accidents just like this one have been caused by instrument failure." The difference between the two is that in the deductive case, the truth of the premises guarantees the truth of the conclusion, while in the inductive case, the truth of the premises supports the conclusion that instrument failure caused the accident, but more research could show that the conclusion is actually false, even though the premises are true.

Programming computers to make inferences, especially deductive inferences, has had a lot of success. But you can't say that a programme can reason just because it can draw conclusions. To reason, you have to draw conclusions that make sense for the task or situation at hand. Giving computers the ability to tell what is important and what isn't is one of the hardest problems AI has to face.

Problem-solving: Problems usually go like this: given these data, find x. AI is used to solve a very wide range of problems. Some examples are finding the best way to win a board game, figuring out who someone is from a picture, and planning a series of steps that will allow a robot to do a certain task.

Methods for solving problems can be either specific or general. A special-purpose method is made to solve a specific problem and often takes advantage of very specific parts of the situation where the problem is happening. A general method can be used to solve many different kinds of problems. The difference between the current state and the goal state can be reduced step by step with means-end analysis, which is a technique used in AI. The programme chooses actions from a list of ways, which for a simple robot might include pick up, put down, move forward, move back, move left, and move right, until the current state is changed into the goal state.

Perception: Perception involves scanning the surroundings with numerous sense organs, genuine or artificial, and internal processes for analyzing the scene into objects, their features, and relationships. The fact that one and the same item can have various appearances depending on the angle from which it is viewed, whether or not parts of it are projecting shadows, and so on, complicates analysis.

Artificial perception has progressed to the point where a self-controlled car-like device can drive at modest speeds on the open road, and a mobile robot can search a suite of bustling offices for and remove empty drink cans. FREDDY, a stationary robot with a moving TV 'eye' and a pincer 'hand,' was one of the first systems to merge perception and action (constructed at Edinburgh University during the period 1966-1973 under the direction of Donald Michie). FREDDY could recognise a wide range of items and could be taught to create simple artefacts from a jumble of parts, such as a toy automobile.

Language-understanding: A language is a set of signs with predetermined meaning. For example, traffic signs establish a mini-language; it is a matter of convention that the hazard-ahead sign signifies trouble ahead. This language-specific meaning-by-convention is distinct from what is known as natural meaning, as evidenced by phrases like "Those clouds signify rain" and "The drop in pressure suggests the valve is malfunctioning."

The productivity of full-fledged human languages, such as English, separates them from other types of communication, such as bird sounds and traffic sign systems. A productive language is one that is rich enough to allow for the creation of an infinite number of different sentences.

1.6 APPROACHES TO ARTIFICIAL INTELLIGENCE

In the previous sections of this unit, we learned about various concepts of Artificial Intelligence but now the question is “how do we measure if Artificial Intelligence is making a machine to behave or act or perform like human being or not?”

Perhaps, in the future, we will reach a point where AI can behave like humans, but what guarantees do we have that this will continue? Is it possible to make a system that acts like a human to test the certainty of Artificial Intelligence? " The following approaches constitute the foundation for evaluating an AI entity's human-likeness:

- Turing Test
- Approach of The Cognitive Modelling
- Approach of The Law of Thought
- Approach of The Rational Agent

Let’s take a look at how these approaches perform:

In the past, researchers have worked hard to reach all four of these goals. But it is hard to find a good balance between approaches that focus on people and approaches that focus on logic. People are often "irrational" in the sense of being "emotionally unstable," so it's important to tell the difference between human and rational behaviour.

Researchers have found through their studies that a human-centered approach must be an empirical science with hypotheses and experiments to prove them. In a rationalist approach, math and engineering are used together. People in each group sometimes say bad things about the work done by the other groups, but the truth is that each way has led to important discoveries. Let's take a closer look at each one. Acting humanly: The approach of the Turing Test: Alan Turing, who is the most well-known name among the pioneers, thought about how to test an A.I. product to see if it was intelligent. Alan Turing came up with the idea for the Turing Test in 1950 (Turing).

Let's try to answer the question, "**What is the Turing Test in AI**?" Alan Turing, who is the most well-known name among the pioneers, thought about how to test an A.I. product to see if it was intelligent. Turing came up with a test, which is now known as the Turing Test, to see if something is smart. Below is a summary of the Turing test. See figure 2 for more details.

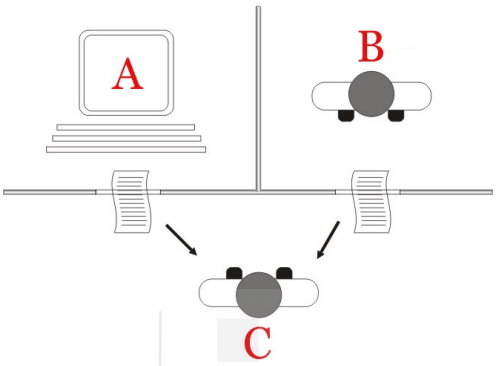


Figure 2: Turing Test

For the purpose of the test, There are three rooms that will be used for the test. In one of the rooms, there is a computer system that is said to be smart. In each of the other two rooms, there is one person sitting. One of the people, who we'll call C, is supposed to ask questions of the computer and the other person, who we'll call B, without knowing who each question is for and, of course, with the goal of figuring out who the computer is. On the other hand, the computer would reply in a way that would keep C from finding out who it is.

The only way for the three of them to talk to each other is through computer terminals. This means that the identity of the computer or person B can only be determined by how intelligent or not the responses are, not by any other human or machine traits. If C can't figure out who the computer is, then the computer must be smart. More accurately, the computer is smart if it can hide its identity from C.

Note that for a computer to be considered smart, it should be smart enough not to answer too quickly, at least not in less than a hundredth of a second, even if it can do something like find the sum of two numbers with more than 20 digits each.

Criticism to Turing Test: There have been a number of criticisms of the Turing test as a machine intelligence test. The Chinese Room Test, developed by John Searle, is one of the most well-known Criticism. The crux of the Chinese Room Test, which we'll discuss below, is that convincing a system, say A, that it possesses qualities of another system, say B, does not suggest that system A actually possesses those qualities. For example, a male human's ability to persuade people that he is a woman does not imply that he is capable of bearing children like a woman.

The scenario for the Chinese Room Test takes place in a single room with two windows. A Shakespeare scholar who knows English but not Chinese is sitting in the room with a kind of Shakespeare encyclopaedia. The encyclopaedia is printed so that for every pair of pages next to each other, one page is written in Chinese characters and the other page is an English translation of the Chinese page. Through one of the windows, Chinese characters with questions about Shakespeare's writing are sent to the person

inside. The person looks through the encyclopaedia and, when he or she finds the exact copy of the sequence of characters sent in, reads the English translation, thinks of the answer, and writes it down in English for his or her own understanding. The person then looks in the encyclopaedia for the corresponding sequence of Chinese characters and sends the sequence of Chinese characters through the other window. Now, Searle says that even though the scholar acts as though he or she knows Chinese, this is not the case. Just because a system can mimic a quality doesn't mean that it has that quality.

Thinking humanly: It is the cognitive modeling approach to thinking like a human, from this point of view, the Artificial Intelligence model is based on Human Cognition, which is the core of the human mind. This is done through three approaches, which are as follows:

- Introspection, which means to look at our own thoughts and use those thoughts to build a model.
- Psychological Experiments, which means running tests on people and looking at how they act.
- Brain imaging, which means to use an MRI to study how the brain works in different situations and then copy that through code.

Thinking rationally i.e. The laws of thought approach: This approach Relates to use the laws of thought to think logically: The Laws of Thought are a long list of logical statements that tell our minds how to work. This method, called "Thinking Rationally," is based on these laws. By putting in place algorithms for artificial intelligence, these laws can be written down and made to work. But solving a problem by following the law is very different from solving a problem in the real world. Here are the biggest problems with this approach.

Acting rationally i.e. The rational agent approach: In every situation, a rational agent approach tries to find the best possible outcome. This means that it tries to make the best decision it can given the circumstances. It means that the agent approach is much more flexible and open to change. The Laws of Thought approach, on the other hand, says that a thing must act in a way that makes sense. But there are some situations where there is no logically right thing to do and there are more than one way to solve the problem, each with different results and trade-offs. At that point, the rational agent method works well.

☞ Check Your Progress 2

Q4 Briefly discuss the various components of intelligence

……

……

Q5 how do we measure if Artificial Intelligence is making a machine to behave or act or perform like human being or not?"

……

……

Q6 What is Turing Test? What is the Criticism to the Turing Test?

……

……

1.7 COMPARISON - ARTIFICIAL INTELLIGENCE, MACHINE LEARNING & DEEP LEARNING

Artificial intelligence is a big field that includes a lot of different ways of doing things, from top-down (knowledge representation) to bottom-up (machine learning). In recent years, people have often talked about three related ideas: artificial intelligence (AI), machine learning (ML), and deep learning (DL) (DL). AI is the most general term, machine learning is a part of AI, and deep learning is a type of machine learning. Figure 5 shows how these three ideas are related to each other. Figure 2(b) shows that AI is a broad field with many different subdomains. However, AI's recent rise in popularity is largely due to how well machine learning, especially deep learning, works. So, this entry will talk about these two areas of AI: Machine Learning (ML) and Deep Learning (DL) Figure 2 (a)

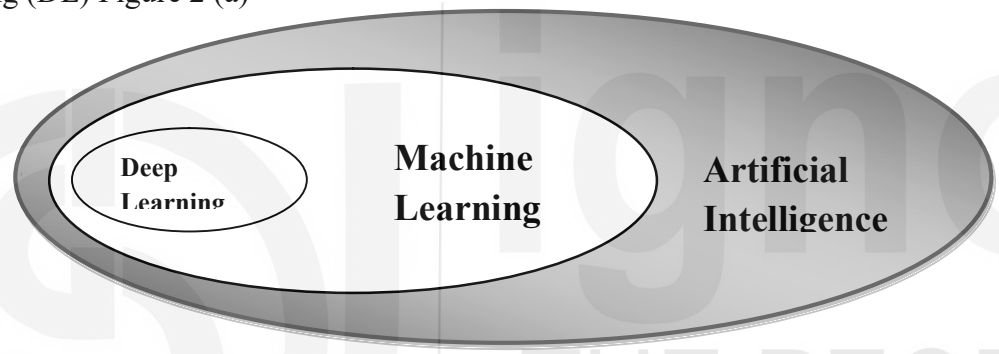


Fig 2(a):AI, ML, DL

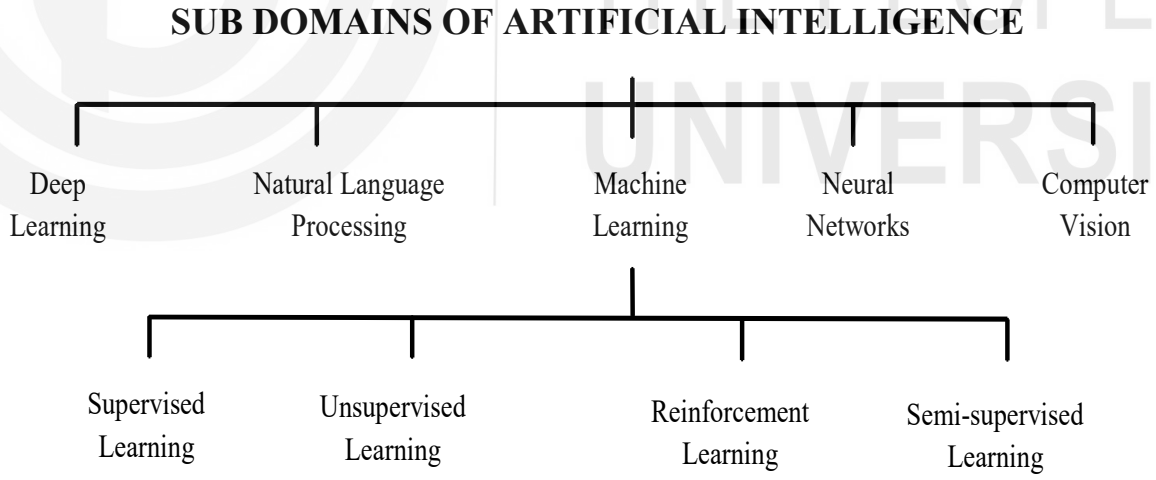


Figure 2(b) : Various Sub Domains of Artificial Intelligence

To make a system that is artificially intelligent, you have to carefully do Reverse-Engineering of human traits and machine abilities. Also, to understand how an AI system really works, you have to get a good grasp of the different parts of AI and how they can be used in different industries or industrial fields.

Introduction to Machine Learning (ML): Machine learning is a branch of artificial intelligence (AI). It explains one of the most important ideas in AI, which has to do with learning through experience and not through being taught. One of the most recent advances in AI, this way of learning is made possible by applying machine learning to very large data sets. Machine learning algorithms find patterns and learn how to make predictions and recommendations by using data and experiences instead of explicit programming instructions. The algorithms also change as they get new data and learn from their experiences. This makes them more effective over time.

Algorithms for machine learning are based on ways that people learn from their experiences. This means that they are programmed to learn from what they do and get better at what they do. They don't need to be told what to do to get the desired results. They are set up so that people can learn by looking at the data sets they can access and comparing what they see to examples of the final results. They also look for patterns in the output and try to figure out how to use the different parts to get the output they want.

ML shows a machine how to draw conclusions and make decisions based on what it has learned in the past. It looks for patterns and analyses past data to figure out what these patterns mean so that a possible conclusion can be reached without the need for human experience. Businesses save time and make better decisions by using automation to evaluate data and come to conclusions..

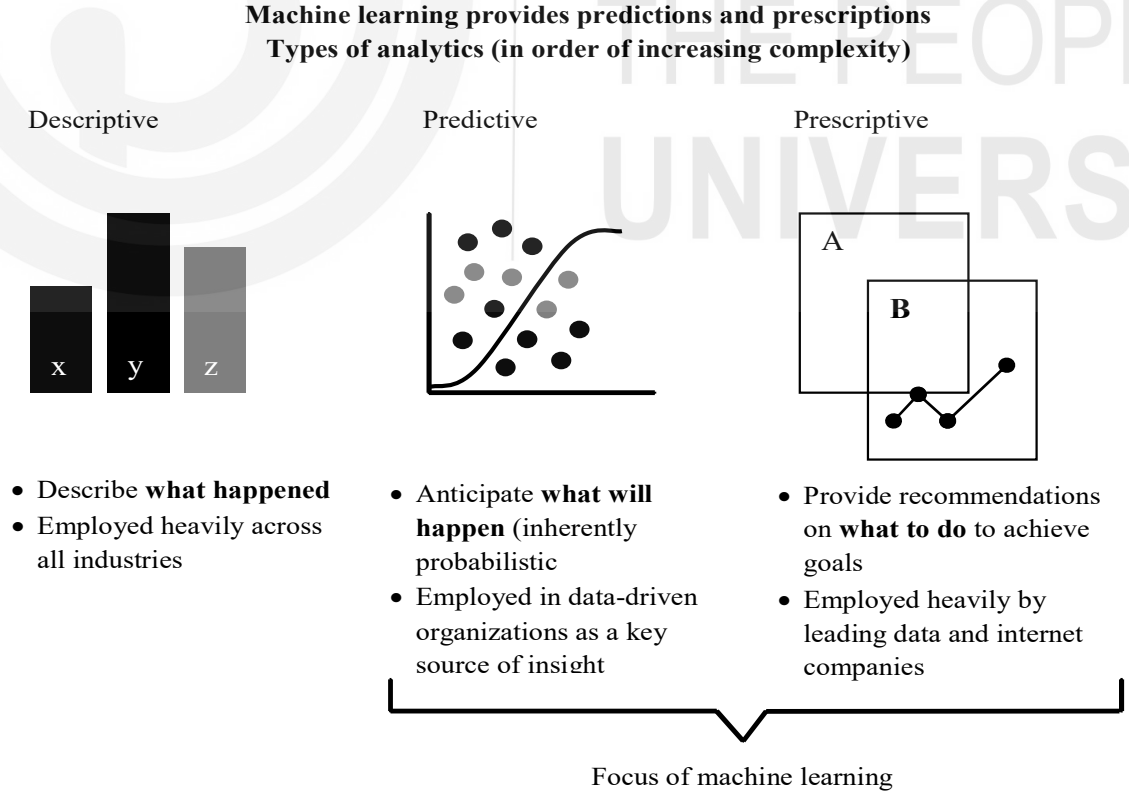


Figure 3 :Machine Learning : Descriptive, Predictive and Prescriptive Analytics

Introduction to Deep Learning (DL) Deep Learning is a subfield of machine learning that focuses on algorithms called "Artificial Neural Networks" (ANN) that are based on how the brain is built and how it works. Deep Learning is a type of machine learning that can handle a wider range of data sources, needs less pre-processing of data, and often gives more accurate results than traditional machine learning methods.

A neural network is made up of layers of software-based calculators called "neurons" that are linked together. This neural network can take in a huge amount of data and process it through many layers. At each layer, the network learns more complex features of the data. The network can then decide what to do with the data, find out if it was right, and use what it has learned to decide what to do with new data. Deep learning is a way of programming computers that uses the way neural networks work to teach computers to do things that humans do naturally. So, Deep Learning is a way to teach a computer model to run classification algorithms based on an image, text, or sound. Once a neural network knows what an object looks like, it can spot that object in a new picture.

Deep learning is becoming more popular because its models can get better results. It uses large sets of labeled data and neural network architectures to train the models..

☞ **Check Your Progress 3**

Q7 Compare Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL).

……

……

Q8 Compare Descriptive, Predictive and Prescriptive analytics performed under Machine Learning.

……

……

1.8 APPLICATION AREAS OF ARTIFICIAL INTELLIGENCE SYSTEMS

Artificial intelligence is the most important factor in the transformation of economies straight from the ground up, and it is contributing as an efficient alternative. It has a lot of potential to perform optimization in any industry, whether it smart cities or the health sector or agriculture or any other prospective sector of relevance, and below we have included a few of the systems in which AI is functioning as the major source of competitive advantage:

a) Healthcare: The application of AI in healthcare can help address issues of high barriers to access to healthcare facilities, particularly in rural areas that suffer from poor connectivity and a limited supply of healthcare professionals. This is especially true in areas where the supply of healthcare professionals is limited. The deployment of use cases like as AI-driven diagnostics, personalised treatment, early diagnosis of potential pandemics, and imaging diagnostics, amongst others, is one way to accomplish this goal.

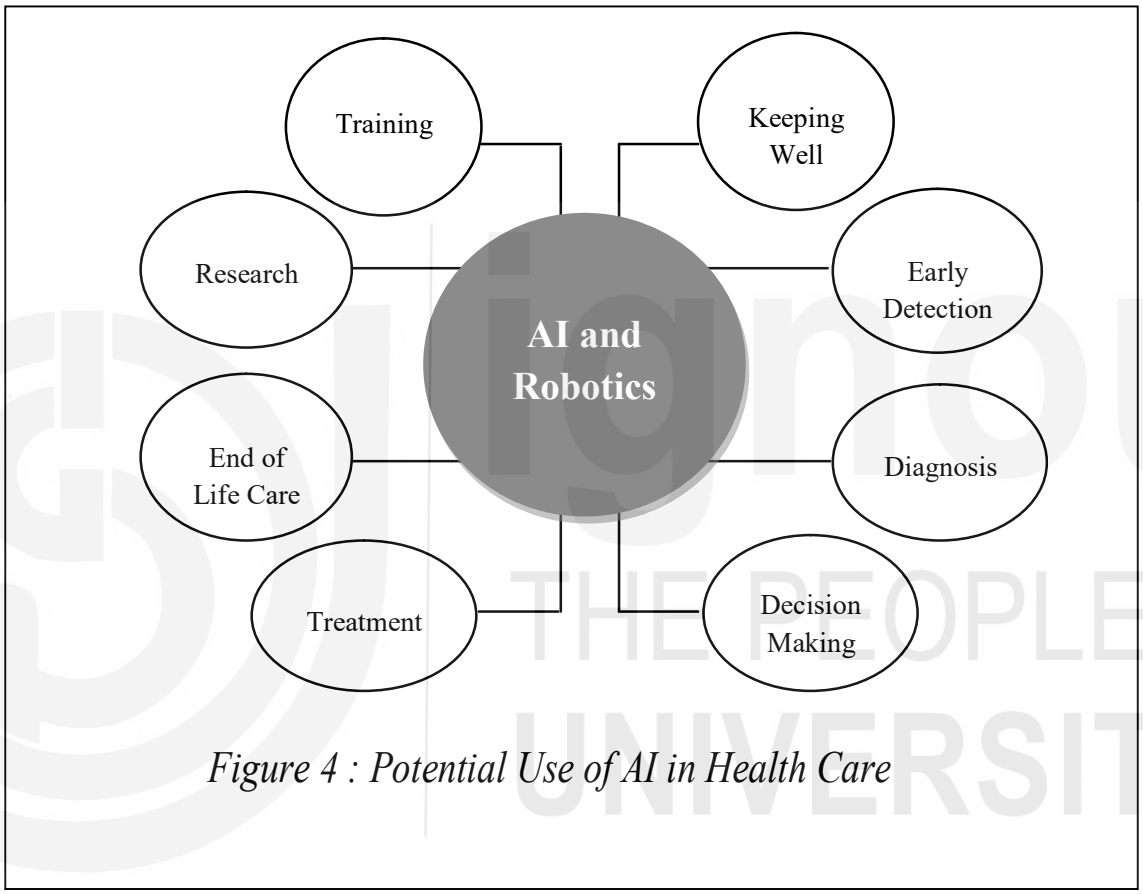


Figure 4 : Potential Use of AI in Health Care

b) Agriculture: AI has the potential to bring in a food revolution while simultaneously satisfying the ever-increasing need for food (global need to produce 50 percent more food and cater to an additional 2 billion people by 2050 as compared to today). It also has the ability to resolve issues such as inadequate demand prediction, a lack of secure irrigation, and the abuse or misuse of pesticides and fertilizers. These are only some of the problems that it could solve. The increase of crop output through real-time advising is one example of a use case. Other use cases include the advanced detection of pest infestations and the forecast of crop prices to advise sowing methods.

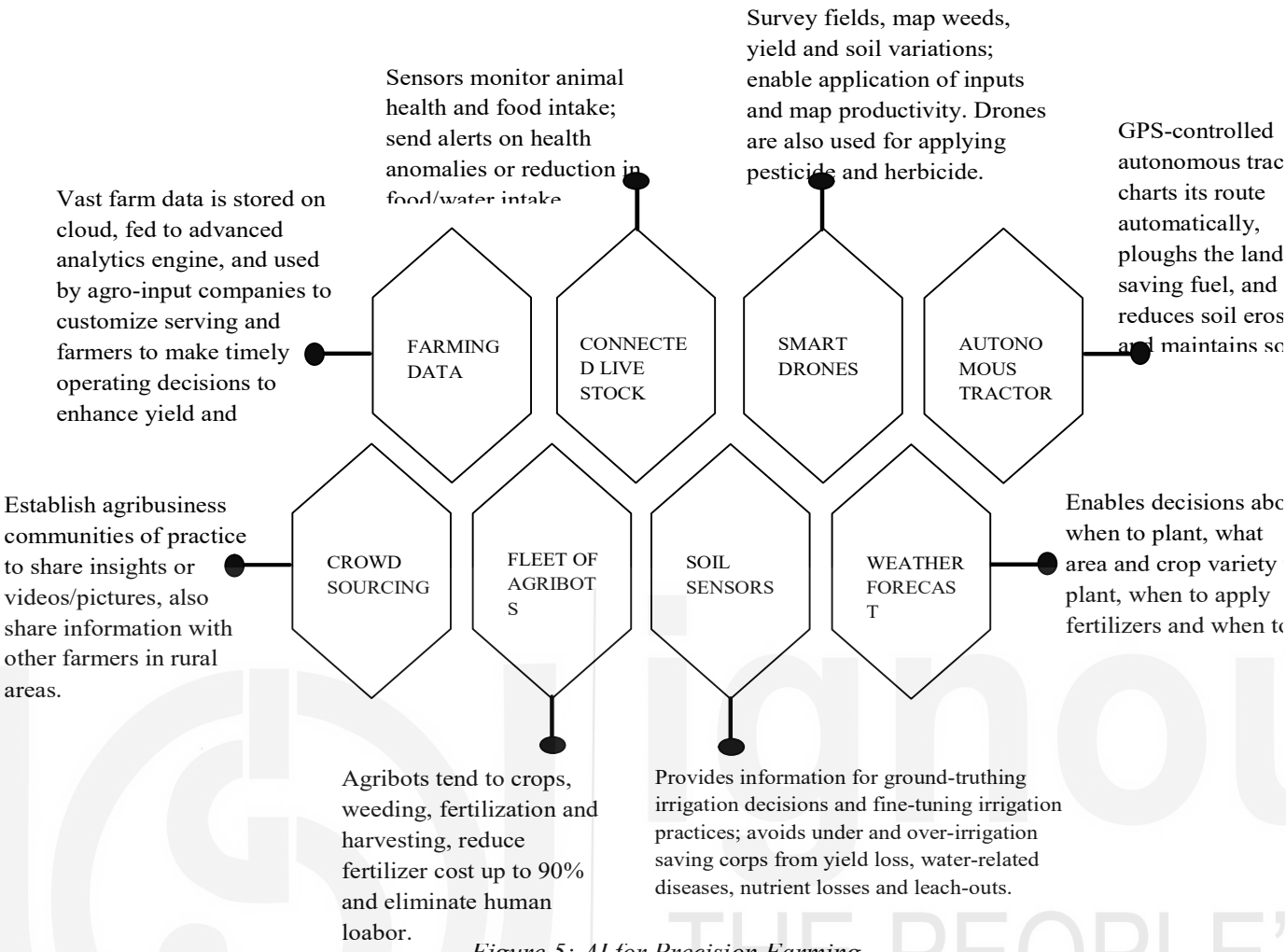


Figure 5: AI for Precision Farming -

All of the stages of the agricultural value chain indicated above in figure 5 have the potential for the application of artificial intelligence and other associated technologies to have an impact on the levels of production and efficiency at those stages.

c) Smart Mobility, including Transports and Logistics: Autonomous fleets for ride sharing, semi-autonomous features such as driver assistance, and predictive engine monitoring and maintenance are all possible use cases for smart mobility, which includes transportation and logistics. Other areas where AI can have a positive impact include self-driving trucks and delivery, as well as better traffic control.

d) Retail: The retail industry was one of the first to use AI solutions. For example, personalised suggestions, browsing based on user preferences, and image-based product search have all been used to improve the user experience. Other use cases include predicting what customers will want, keeping track of inventory better, and managing deliveries more efficiently.

e) Manufacturing: AI-based solutions are expected to help the manufacturing industry the most. This will make possible the "Factory of the Future" by allowing flexible and adaptable technical systems to automate processes and machinery that can respond to new or unexpected situations by making smart decisions. Impact areas include engineering (AI for R&D), supply chain management (predicting demand), production (AI can cut costs and increase efficiency), maintenance (predictive maintenance and

better use of assets), quality assurance (e.g., vision systems with machine learning algorithms to find flaws and differences in product features), and in-plant logistics and warehousing.

f) Energy: In the energy sector, possible use cases include modelling and forecasting the energy system to make it less unpredictable and make balancing and using power more efficient. In renewable energy systems, AI can help store energy through smart metres and intelligent grids. It can also make photovoltaic energy more reliable and less expensive. AI could also be used to predict maintenance of grid infrastructure, just like it is in manufacturing.

g) Smart Cities: Integrating AI into newly built smart cities and infrastructure could also help meet the needs of a population that is moving to cities quickly and improve the quality of life for those people. Some possible use cases include controlling traffic to reduce traffic jams and managing crowds better to improve security.

h) Education and Skilling: Quality and access problems in the education sector might be fixed by AI. Possible uses include adding to and improving the learning experience through personalized learning, automating and speeding up administrative tasks, and predicting when a student needs help to keep them from dropping out or to suggest vocational training.

i) Financial industry: The financial industry also uses AI. For example, it helps the fraud department of a bank find and flag suspicious banking and finance activities like unusual debit card use and large account deposits. AI is also used to make trading easier and more efficient. This is done by making it easier to figure out how many securities are being bought and sold and how much they cost.

Top Used Applications of Artificial Intelligence

- Tools and checkers for plagiarism
- Recognizing faces;
- Putting an AI autopilot on commercial planes
- Applications for sharing rides (E.g.: Uber, Lyft)
- E-mail spam filters; voice-to-text features; search suggestions
- Google's predictions based on AI (E.g.: Google Maps)
- Protecting against and stopping fraud.
- Smart personal assistants (E.g.: Siri, Alexa)

There are various ways to use artificial intelligence. The technology can be used in different industries and sectors, but the adoption of AI by different sectors has been affected by technical and regulatory challenges, but the biggest factor has been how it will affect business.

1.9 INTELLIGENT AGENTS

An agent may be thought of as an entity that acts, generally on behalf of someone else. More precisely, an agent is an entity that perceives its environment through sensors and acts on the environment through actuators. Some experts in the field require an agent to be additionally autonomous and goal directed also.

A percept may be thought of as an input to the agent through its censors, over a unit of time, sufficient enough to make some sense from the input.

Percept sequence is a sequence of percepts, generally long enough to allow the agent to initiate some action.

In order to further have an idea about what a computer agent is, let us consider one of the first definitions of agent, which was coined by John McCarthy and his friends at MIT.

A software agent is a system which, when given a goal to be achieved, could carry out the details of the appropriate (computer) operations and further, in case it gets stuck, it can ask for advice and can receive it from humans, may even evaluate the appropriateness of the advice and then act suitably.

Essentially, a computer agent is a computer software that additionally has the following attributes:

(i) it has autonomous control i.e., it operates under its own control
(ii) it is perceptive, i.e., it is capable of perceiving its own environment
(iii) it persists over a long period of time
(iv) it is adaptive to changes in the environment and
(v) it is capable of taking over others' goals.

As the concept of Intelligent Agents is of relatively new, different pioneers and other experts have been conceiving and using the term in different ways. There are two distinct but related approaches for defining an agent. The first approach treats an agent as an ascription i.e., the perception of a person (which includes expectations and points of view) whereas the other approach defines an agent on the basis of the description of the properties that the agent to be designed is expected to possess.

Let us first discuss the definition of agent according to first approach. Among the people who consider an agent as an ascription, a popular slogan is "Agent is that agent does". In everyday context, an agent is expected to act on behalf of someone to carry out a particular task, which has been delegated to it. But to perform its task successfully, the agent must have knowledge about the domain in which it is operating and also about the properties of its current user in question. In the course of normal life, we hire different agents for different jobs based on the required expertise for each job. Similarly, a non-human intelligent agent also is imbedded with required expertise of the domain as per requirements of the job under consideration. For example, a football-playing agent would be different from an email-managing agent, although both will have the common attribute of modeling their user.

According to the second approach, an agent is defined as an entity, which functions continuously and autonomously, in a particular environment, which may have other agents also. By continuity and autonomy of an agent, it is meant that the agent must be able to carry out its job in a flexible and intelligent fashion and further is expected to adapt to the changes in its environment without requiring constant human guidance or intervention. Ideally, an agent that functions continuously in an environment over a long period of time would also learn from its experience. In addition, we expect an agent, which lives in a multi-agent environment, to be able to communicate and cooperate with them, and perhaps move from place to place in doing so.

According to the second approach to defining agent, an agent is supposed to possess some or all of the following properties:

- Reactivity: The ability of sensing the environment and then acting accordingly.
- Autonomy: The ability of moving towards its goal, changing its moves or strategy, if required, without much human intervention.
- Communicating ability: The ability to communicate with other agents and humans.
- Ability to coexist by cooperating: The ability to work in a multi-agent environment to achieve a common goal.
- Ability to adapt to a new situation: Ability to learn, change and adapt to the situations in the world around it.
- Ability to draw inferences: The ability to infer or conclude facts, which may be useful, but are not available directly.
- Temporal continuity: The ability to work over long periods of time.
- Personality: Ability to impersonate or simulate someone, on whose behalf the agent is acting.
- Mobility: Ability to move from one environment to another.

Task environments or problem environments are the environments, which include all the elements involved in the problems for which agents are thought of as solutions. Task environments will vary with every new task or problem for which an agent is being designed. Specifying the task environment is a long process which involves looking at different measures or parameters. Next, we discuss a standard set of measures or parameters for specifying a task environment under the heading PEAS.

PEAS (Performance, Environment, Actuators, Sensors)

For designing an agent, the first requirement is to specify the task environment to the maximum extent possible. The task environment for an agent to solve one type of problems, may be described by the four major parameters namely, performance (which is actually the expected performance), environment (i.e., the world around the agent), actuators (which include entities through which the agent may perform actions) and sensors (which describes the different entities through which the agent will gather information about the environment).

The four parameters may be collectively called as PEAS. We explain these parameters further, through an example of an automated agent, which we will preferably call automated public road transport driver. This is a much more complex agent than the simple boundary following robot which we have already discussed.

Example (An Automated Public Road Transport Driver Agent)

We describe the task environment of the agent on the basis of PEAS.

Performance Measures: Some of the performance measures which can easily be perceived of an automated public road transport driver would be:

- Maximizing safety of passengers
- Maximizing comfort of passengers
- Ability to reach correct destination
- Ability to minimize the time to reach the destination Obeying traffic rules
- Causing minimum discomfort or disturbance to other agents
- Minimizing costs, etc.

Environment (or the world around the agent) We must remember that the environment or the world around the agent is extremely uncertain or open ended. There are unlimited combinations of possibilities of the environment situations, which such an agent could face. Let us enumerate some of the possibilities or circumstances which an agent might face:

- Variety of roads e.g., from 12-lane express-ways, freeways to dusty rural bumpy roads; different road rules including the ones requiring left-hand drive in some parts of the world and right-hand drive-in other parts.
- The degree of knowledge of various places through which and to which driving is to be done.
- Various kinds of passengers, including high cultured to almost ruffians etc.
- All kind of other traffic possibly including heavy vehicles, ultra-modern cars, three-wheelers and even bullock carts.

Actuators: These include the following:

- Handling steering wheel, brakes, gears and accelerator
- Understanding the display screen
- A device or devices for all communication required

Sensors: The agent acting as automated public road transport driver must have some way of sensing the world around it i.e., the traffic around it, the distance between the automobile and the automobiles ahead of it and its speed, the speeds of neighboring vehicles, the condition of the road, any turn ahead etc. It may use sensors like odometer, speedometer, sensors telling the different parameters of the engine, Global Positioning System (GPS) to understand its current location and the path ahead. Also, there should be some sort of sensors to calculate its distance from other vehicles etc.

We must remember that the agent example the automated public road transport driver, which we have considered above, is quite difficult to implement. However, there are many other agents, which operate in

comparatively simpler and less dynamic environments, e.g., a game playing robot, an assembly line robot control, and an image processing agent etc.

In respect of the design and development of intelligent agents, with the passage of time, the momentum seems to have shifted from hardware to software, the letter being thought of as a major source of intelligence. But, obviously, some sort of hardware is essentially needed as a home to the intelligent agent.

There are two parts of an agent or its structure:

- A (hardware) device with sensors and actuators in which that agent will reside, called the *architecture* of the agent.
- An agent program that will convert or map the percepts in to actions.

Also, the agent program and its architecture are related in the sense that for a different agent architecture a different type of agent program is required and vice-versa. For example, in case of a *boundary following robot*, if the robot does not have the capability of sensing adjacent cells to the right, then the agent program for the robot has to be changed.

Next, we discuss different categories of agents, which are differentiated from each other on the basis of their agent programs. Capability to write efficient agent programs is the key to the success for developing efficient rational agents. Although the table driven approach (in which an agent acts on the basis of the set of all possible percepts by storing these percepts in tables) to design agents is possible yet the approach of developing equivalent agent programs is found much more efficient.

Next, we discuss some of the general categories of agents based on their agents' programs, Agents can be grouped into five classes based on their degree of perceived intelligence and capability. All these agents can improve their performance and generate better action over the time. These are given below:

- ***SR (Simple Reflex) agents***
- ***Model Based reflex agents***
- ***Goal-based agents***
- ***Utility based agents***
- ***Stimulus-Response Agents***
- ***Learning agents***

SR (Simple Reflex) agents: These are the agents or machines that have no internal state (i.e., the don't remember anything) and simply react to the current percepts in their environments. An interesting set of agents can be built, the behaviour of the agents in which can be captured in the form of a simple set of functions of their sensory inputs. One of the earliest implemented agents of this category was called ***Machina Speculatrix .*** This was a device with wheels, motor, photo cells and vacuum tubes and was designed to move in the direction of light of less intensity and was designed to avoid the direction of the bright light. **A boundary following robot is also an SR agent**. For an automobile-driving agent also, some aspects of its behavior like applying brakes immediately on observing either the vehicle immediately ahead applying brakes or a human being coming just in front of the automobile suddenly,

show the simple reflex capability of the agent. Such a simple reflex action in the agent program of the agent can be implemented with the help of simple condition-action rules.

For example : ***IF*** *a human being comes in front of the automobile suddenly*
THEN *apply breaks immediately*.

Although implementation of SR agents is simple **yet on the negative side** this type of agents has very limited intelligence because ***they do not store or remember anything.*** As a consequence, they cannot make use of any previous experience. In summary, they do not learn. Also **they are capable of operating correctly only if the environment is fully observable.**

Model Based Reflex agents : Simple Reflex agents are not capable of handling task environments that are not fully observable. In order to handle such environments properly, in addition to reflex capabilities, the agent should, maintain some sort of internal state in the form of a function of the sequence of percepts recovered up to the time of action by the agent. Using the percept sequence, the internal state is determined in such a manner that it reflects some of the aspects of the unobservable environment. Further, in order to reflect properly the unobserved environment, the agent is expected to have a model of the task environment encoded in the agent's program, where the model has the knowledge about–

(i) the process by which the task environment evolves independent of the agent and
(ii) effects of the actions of the agent have on the environment.
Thus, in order to handle properly the partial observability of the environment, the agent should have a model of the task environment in addition to reflex capabilities. Such agents are called **Model-based Reflex Agents**

Goal Based Agents : In order to design appropriate agent for a particular type of task, we know the nature of the task environment plays an important role. Also, it is desirable that the complexity of the agent should be minimum and just sufficient to handle the task in a particular environment. In this regard, first we discussed the simplest type of agents, viz., Simple Reflex Agents. The action of this type of agent is decided by the current precept only. Next, we discussed the Model-Based Reflex Agents, for which an action is decided by taking into consideration not only the latest precept, but the whole precept history summarized in the form of internal state. Also, action for this type of agent is also decided by taking into consideration the knowledge of the task environment, represented by a model of the environment and encoded into the agent's program. However, in respect of a number of tasks, even this much knowledge may not be sufficient for appropriate action. For example, when we are going from city A to city B, in order to take appropriate action, it is not enough to know the summary of actions and path which has taken us to some city C between A and B. We also have to remember the goal of reaching to city B.

Goal based agents are **driven by the goal** they want to achieve, i.e., **their actions are based on the information regarding their goal, in addition to, of course, other information in the current state.** This goal information is also a part of the current state description and it describes everything that is desirable to achieve the goal. As mentioned earlier, an example of a goal-based agent is an agent that is required to find the path to reach a city. In such a case, if the agent is *an automobile driver agent*, and if the road is splitting ahead into two roads, then the agent has to decide which way to go to achieve its goal

of reaching its destination. Further, if there is a crossing ahead then the agent has to decide, whether to go straight, to go to the left or to go to the right. In order to achieve its goal, the agent needs some information regarding the goal which describes the desirable events and situations to reach the goal. The agent program would then use this goal information to decide the set of actions to take in order to reach its goal.

Another desirable capability which a good goal-based agent should have been that if an agent finds that a part of the sequence of the previous steps has taken the agent away from its goal then it should be able to retract and start its actions from a point which may take the agent toward the goal.

In order to take appropriate action, decision-making process in goal-based agents may be simple or quite complex depending on the problem. Also, **the decision-making required by the agents of this kind needs some sort of looking into the future.** For example, it may analyze the possible outcome of a particular action before it actually performs that action. In other words, we can say that ***the agent would perform some sort of reasoning of if-then-else type***, e.g., an automobile driver agent having one of its goals as not to hit any vehicle in front of it, when finds the vehicle immediately ahead of it slowing down may not apply brakes with full force and instead may apply brakes slowly so that the vehicles following it may not hit it.

As the goal-based agents may have to reason before they take an action, these agents might be slower than other types of agents but will be more flexible in taking actions as their decisions are based on the acquired knowledge which can be modified also. Hence, **as compared to SR agents** which may require rewriting of all the condition-action rules in case of change in the environment, the goal-based agents can adapt easily when there is any change in its goal.

Utility Based Agents : Goal based agent's success or failure is judged in terms of its capability for achieving or not achieving its goal. A goal-based agent, for a given pair of environment state and possible input, only knows whether the pair will lead to the goal state or not. Such an agent will not be able to decide in which direction to proceed when there are more than one conflicting goals. Also, in a goal-based agent, there is no concept of partial success or somewhat satisfactory success. Further, if there are more than one method of achieving a goal, then no mechanism is incorporated in a Goal-based agent of choosing or finding the method which is faster and more efficient one, out of the available ones, to reach its goal.

A more general way to judge the success or happiness of an agent may be, through assigning to each state a number as an approximate measure of its success in reaching the goal from the state. In case, the agent is embedded with such a capability of assigning such numbers to states, then it can choose, out of the reachable states in the next move, the state with the highest assigned number, out of the numbers assigned to various reachable states, indicating possibly the best chance of reaching the goal.

It will allow the goal to be achieved more efficiently. Such an agent will be more useful, i.e., will have more utility. A utility-based agent uses a **utility function**, which maps each of the world states of the agent to some degree of success. If it is possible to define the utility function accurately, then the agent will be able to reach the goal quite efficiently. Also, a utility-based agent is *able to make decisions in case of conflicting goals,* generally choosing the goal with higher success rating or value. Further, in

environments with multiple goals, the utility-based agent quite likely chooses the goal with least cost or higher utility goal out of multiple goals.

Stimulus-Response Agents : A stimulus response agent (or a reactive agent) take input from the world through sensors, and then take action based on those inputs through actuators. Between the stimulus and response, there is a processing unit that can be arbitrarily complex. An example of such an agent is one that controls a vehicle in a racing game: the agent "looks" at the road and nearby vehicles, and then decides how much to turn and break. Such Agents (Stimulus-Response Agents are the Reactive agents) represents a special category of agents, which do not possess internal, symbolic models of their environments; instead, they act/respond in a stimulus-response manner to the present state of the environment in which they are embedded. These agents are relatively simple and they interact with other agents in basic ways. Nevertheless, complex patterns of behavior emerged from the interactions when the ensemble of agents is viewed globally

Learning Agents : It is not possible to encode all the knowledge in advance, required by a rational agent for optimal performance during its lifetime. This is especially true of the real life, and not just theoretical, environments. These environments are **dynamic** in the sense that the environmental conditions change, not only due to the actions of the agents under considerations, but due to other environmental factors also. For example, all of a sudden, a pedestrian comes just in front of the moving vehicle, even when there is green signal for the vehicle. In a multi-agent environment, all the possible decisions and actions an agent is required to take, are generally unpredictable in view of the decisions taken and actions performed simultaneously by other agents. Hence, **the ability of an agent to succeed in an uncertain and unknown environment depends on its learning capability** i.e., its capability to change approximately its knowledge of the environment. For an agent with learning capability, some initial knowledge is coded in the agent program and after the agent starts operating, it learns from its actions the evolving environment, the actions of its competitors or adversaries etc. so as to improve its performance in ever-changing environment. If approximate learning component is incorporated in the agent, then the knowledge of the agent gradually increases after each action starting from its initial knowledge which was manually coded into it at the start.

Conceptually the learning agent consists of four components:

***(i) Learning Component*:** It is the component of the agent, which on the basis of the percepts and the feedback from the environment, gradually improves the performance of the agent.

***(ii) Performance Component*:** It is the component from which all actions originate on the basis of external percepts and the knowledge provided by the learning component.

The design of learning component and the design of performance element are very much related to each other because a learning component is of no use unless the performance component can be designed to convert the newly acquired knowledge into better useful actions.

(iii) Critic Component*:** This component finds out how well the agent is doing with respect to a certain fixed performance standard and it is also responsible for any future modifications in the performance component. ***The critic is necessary to judge the agent's success with respect to the chosen

performance standard, especially in a dynamic environment. **Forexample**, in order to check whether a certain job is accomplished, the critic will not depend on external percepts only but it will also compare the current state to the state, which indicates the completion of that task.

(iv) Problem Generator Component: This component is responsible for suggesting actions (some of which may not be optimal) in order to gain some fresh and innovative experiences. Thus, ***this component allows the agent to experiment a little*** by traversing sometimes uncharted territories by choosing some new and suboptimal actions. This may be useful, because the actions which may seem suboptimal in a short run, may turn out to be much better in the long run.

In the case of *an automobile driver agent*, this agent would be of little use if it does not have learning capability, as the environment in which it has to operate is totally dynamic and unpredictable in nature. **Once the automobile driver agent starts operating, it keeps on learning from its experiences, both positive and negative**. If faced with a totally new and previously unknown situation, e.g., encountering a vehicle coming from the opposite direction on a one-way road, the problem generator component of the driver agent might suggest some innovative action to tackle this new situation. Moreover, the learning becomes more difficult in the case of an automobile driver agent, because the environment is only partially observable.

Different Forms of Learning in Agents: The purpose of embedding learning capability in an agent is that it should not depend totally on the knowledge initially encoded in it and on the external percepts for its actions. The agent learns by evaluating its own decisions and/or making observations of new situations it encounters in the ever-changing environment.

There may be various criteria for developing learning taxonomies. The criteria may be based on –

- The type of knowledge learnt, e.g., concepts, problem-solving or game playing,
- The type of representation used, e.g., predicate calculus, rules or frames,
- The area of application, e.g., medical diagnosis, scheduling or prediction.

☞ Check Your Progress 4

Q9 What are Intelligent agents in AI? Briefly discuss the properties of Agents.

……

……

Q10 What are Task environments? Briefly discuss the standard set of measures or parameters for specifying a task environment under the heading PEAS.

……

……

1.10 SUMMARY

In this unit we learned about the difference between knowledge and intelligence and also pointed out the meaning of Artificial Intelligence (AI), along with the application of AI systems in various fields. The unit also covers the historical development of the field of AI systems. Along with the development of the AI as a discipline, the need of classification of AI systems was felt, and hence the unit discussed the classification of the AI systems in detail. Further, the unit discussed about the concepts of Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL). Finally, the unit discussed the components of Intelligence, which was extended for the understanding of the concepts of Intelligent Agents, with special emphasis on Stimulus - Response Agents

1.11 SOLUTIONS/ANSWERS

☞ **Check Your Progress 1**

Q1 How Knowledge differs from intelligence? What do you understand by the term Artificial Intelligence (AI) ? List the various technologies and their corresponding illustrative solutions.

Sol- Refer section 1.3

Q2 Classify AI on the basis of the functionalities of AI

Sol- Refer section 1.3

Q3 Compare ANI, AGI and ASI, in context of AI

Sol- Refer section 1.3

☞ **Check Your Progress 2**

Q4 Briefly discuss the various components of intelligence

Sol – Refer Section 1.5

Q5 how do we measure if Artificial Intelligence is making a machine to behave or act or perform like human being or not ?"

Sol – Refer Section 1.6

Q6 What is Turing Test? What is the Criticism to the Turing Test ?

Sol – Refer Section 1.6

☞ **Check Your Progress 3**

Q7 Compare Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL).

Sol – Refer Section 1.7

Q8 Compare Descriptive, Predictive and Prescriptive analytics performed under Machine Learning.

Sol – Refer Section 1.7

☞ **Check Your Progress 4**

Q9 What are Intelligent agents in AI? Briefly discuss the properties of Agents.

Sol – Refer Section 1.9

Q10 What are Task environments? Briefly discuss the standard set of measures or parameters for specifying a task environment under the heading PEAS.

Sol – Refer Section 1.9

1.12 FURTHER READINGS

1. Ela Kumar, “Artificial Intelligence”, IK International Publications
2. E. Rich and K. Knight, “Artificial intelligence”, Tata Mc Graw Hill Publications
3. N.J. Nilsson, “Principles of AI”, Narosa Publ. House Publications
4. John J. Craig, “Introduction to Robotics”, Addison Wesley publication
5. D.W. Patterson, “Introduction to AI and Expert Systems" Pearson publication

UNIT 2 PROBLEM SOLVING USING SEARCH

Structure **Page No**

2.0 INTRODUCTION

Many AI-based applications need to figure out how to solve problems. In the world, there are two types of problems. First, the problem which can be solved by using deterministic procedure and the success is guaranteed. But most real-world problems can be solved only by searching a solution. AI is concerned with these second types of problems solving.

To build a system to solve a problem, we need to

- ***Define the problem precisely***-find initial and final configuration for acceptable solution to the problem.
- ***Analyse the problem***-find few important features that may have impact on the appropriateness of various possible techniques for solving the problem

- ***Isolate and represent*** *task* knowledge necessary to solve the problem
- ***Choose the best problem-solving technique***(s) and apply it to the particular problem.

To provide a formal description of a problem, we need to do the following:

a. Define a **state space** that contains all the possible configurations of the relevant objects.

b. Specify one or more states that describe possible situations, from which the problem-solving process may start. These states are called ***initial states***.

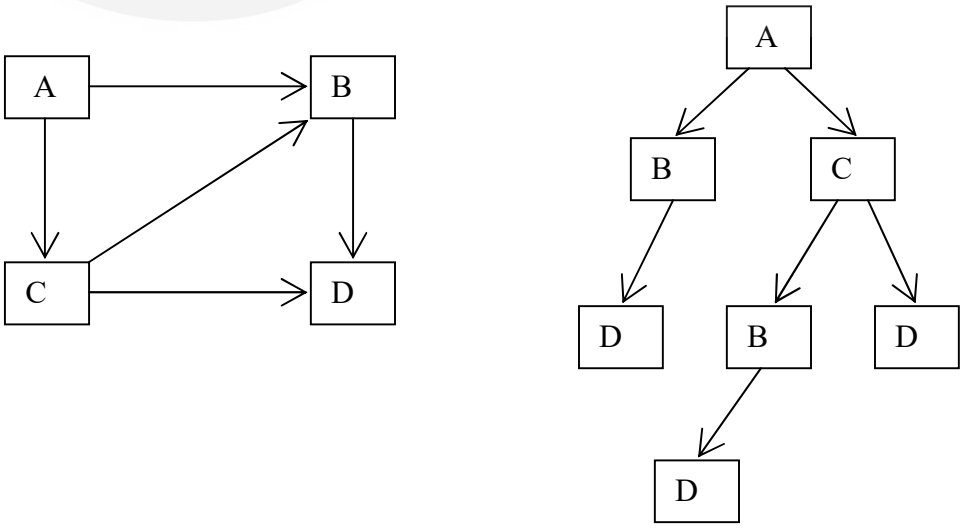
c. Specify one or more than one ***goal states***.

d. Defining a *set of rules* for the actions (operators) that can be taken.

The problem can then be solved by using the rules, in combination with an appropriate ***control strategy***, to move through the problem space until a path from an ***initial state*** to a ***goal state*** is found. This process is known as ***'search***'. Thus, search is fundamental to the problem-solving process. Search is a general mechanism that can be used when a more direct method is not known. Search provides the framework into which more direct methods for solving subparts of a problem can be embedded. **All Al problems are formulated as search problems**.

A **problem space** is represented by a directed graph, where *nodes* represent *search state* and *paths* represent the *operators* applied to change the state. To simplify search algorithms, it is often convenient to logically and programmatically represent a problem space as a ***tree***. A tree usually decreases the complexity of a search at a cost. Here, the cost is due to duplicating some nodes on the tree that were linked numerous times in the graph, e.g., node B and node D shown in example below.

Graph **Tree**



A **tree** is a graph in which any two vertices are connected by exactly one path. Alternatively, any connected graph with no cycles is a tree.

Before an AI problem can be solved it must be represented as a **state space**. Here **state** means representation of elements at a given moment. Among all possible states, there are two special states called ***initial state*** (the start point) and ***final state*** (the goal state). A ***successor function (a set of operators)***is used to change the state. It is used to move from one state to another. A state space is the set of all states reachable from the initial state. A ***state space*** essentially consists of a set of nodes representing each state of the problem, arcs between nodes representing the legal moves from one state to another, an initial state, and a goal state. Each state space takes the form of a tree or a graph. In AI, a wide range of problems can be formulated as search problem. The process of searching means a sequence of action that take you from an initial state to a goal state as sown in the following figure 1.

Fig 1: A sequence of action in a search space from initial to goal state.

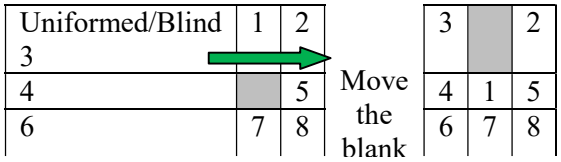
So, State space is the one of the methods to represent the problem in AI. A set of all possible states reachable from the initial state by taking some sequence of action (using some operator) for a given problem is known as the **state space** of the problem. A state space represents a problem in terms of ***states*** and ***operator***s that change states".

In this unit we examine the concept of a **state space** and the different **search process** that can be used to explore the search space in order to find a solution (Goal) state. In the worst case, search explores all possible paths between the *initial state* and the *goal state*.

For better understanding of these definitions describe above, consider the following 8-puzzle problem:

Eight-Puzzle problem Formulation from AI perspectives

initial state: some configuration of the 8-tiles on a 9-cell board.
operators (Action): it's easier if we focus on the blank. There are 4 operators that is, "Moving the blank" : UP, DOWN, LEFT and RIGHT.



Goal state: Tiles in a specific order

	1	2
3	4	5
6	7	8

Solution: Optimal sequence of operators (Actions)
Path costs:
cost of each action = 1
cost of a sequence of actions= the number of actions

A state space representation of 8-puzzle problem is shown in figure-2

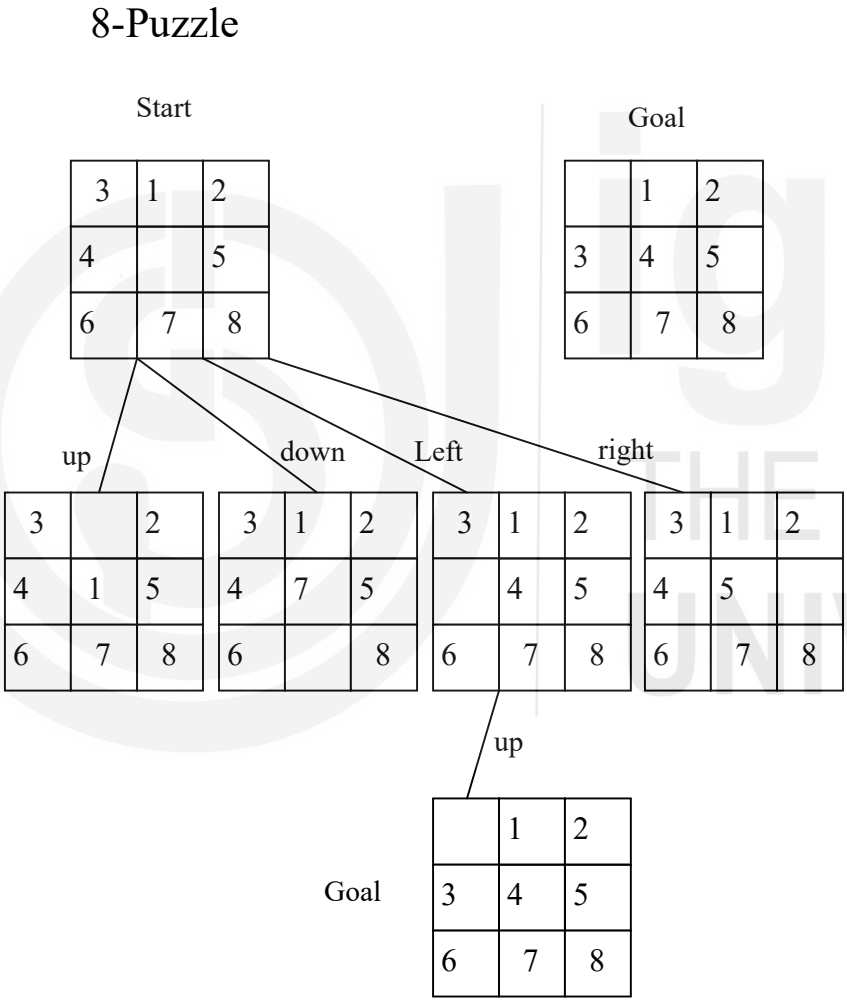


Fig2: A state space representation of 8-puzzle problem generated by "**Move Blank**" operator

2.1 OBJECTIVES

After studying this unit, you should be able to:

- Understand the state space search
- Formulate the problems in the form of state space
- Understand how implicit state space can be unfolded during search
- Explain State space search representation for Water-Jug, 8-puzzle and N-Queen's problem.
- Solve N Queen's Problem using Backtracking approach
- Understand adversarial search (two agent search)
- Differentiate between Minimax and Alpha-beta pruning search algorithm.

2.2 Introduction to State Space Search

It is necessary to represent an AI problem in the form of a state space before it can be solved. A state space is the set of all states reachable from the initial state. A state space forms a graph in which the nodes are states and the arcs between nodes are actions. In state space, a path is a sequence of states connected by a sequence of actions.

2.2.1 Structure of a state space:

The structures of state space are trees and graphs. A tree has one and only one path from any point to any other point. Graph consists of a set of nodes (vertices) and a set of edges (arcs). Arcs establish relationship (connections) between the nodes, i.e., a graph has several paths to a given node. **Operators** are directed arcs between nodes.
The method of solving problem through AI involves the process of defining the search space, deciding start and goal states and then finding the path from start state to goal state through search space.
Search process explores the state space. In the worst case, the search explores all possible paths between the initial state and the goal state.

2.2.2 Problem Solution:

In a state space, a ***solution is a path*** from the initial state to a goal state or sometime just a goal state. A numeric cost is assigned to each path. It also gives the cost of applying the operators to the states. A path cost function is used to measure the quality of solution and out of all possible solutions, an optimal solution has the lowest path cost. The importance of cost depends on the problem and the type of solution asked.

2.2.3 Problem formulation:

Many problems can be represented as ***state space***. The state space of a problem includes: an *initial state*, one or more *goal state*, set of *state transition operator* (or a ***set of production rules***), used to change the current state to another state. This is also known as ***actions.*** A ***control strategy*** is used that specifies the order in which the rules will be applied. For example, Depth-first search (DFS), Breath-first search (BFS) etc. It helps to find the goal state or a path to the goal state.

In general, a state space is represented by 4 tuples as follows:$\boldsymbol{S_s}$: $[\boldsymbol{S}, \boldsymbol{s_0}, \boldsymbol{O}, \boldsymbol{G}]$

Where **S**: Set of all possible states.
$\boldsymbol{s_0}$**:** start state (initial configuration) of the problem, $s_0 \in S$.
O: Set of production rules (or set of state transition operator) used to change the state from one state to another. It is the set of arcs (or links) between nodes.

The ***production rule*** is represented in the form of a pair. Each pair consists of a left side that determines the applicability of the rule and a right side that describes the action to be performed, if the rule is applied.

G: Set of Goal state, $G \in S$.

The sequence of actions (or operators) is called a solution path. It is a path from the initial state to a goal state. This sequence of actions leads to a number of states, starting from initial state to a goal state, as $\{s_0, s_1, s_2, \ldots \ldots, s_n \in G\}$.A sequence of state is called a path. The cost of a path is a positive number. In most of the cases the path cost is computed as the sum of the costs of each action.

The following figure 3 shows a search process in a given state space.

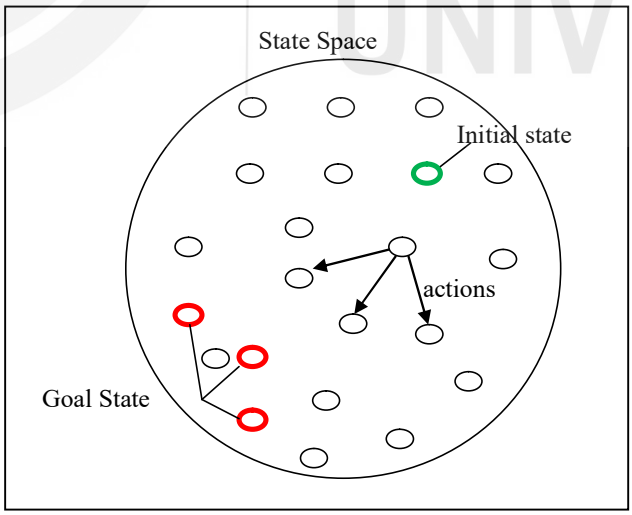


Fig 3 State space with initial and goal node

We need to identify a sequence of actions that will turn the initial state s_0 into the desired goal state G. State space is commonly defined as a directed graph or as a tree in which each node is a state and each arc represents the application of an operator transforming a state to a successor state.

2.2.4 SEARCHING FOR SOLUTIONS

After formulating our problem, we are ready to solve it, this can be done by searching through the state space for a solution, this search will be applied On a search tree or generally a graph that is generated using the initial state and the successor function.

Searching is applied to a search tree which is generated through state expansion, that is applying the successor function to the current state, note that here we mean by state a node in the search tree.

Generally, search is about selecting an option and putting the others aside for later in case the first option does not lead to a solution. The choice of which option to expand first is determined by the search strategy used.

Thus, the problem is solved by using the rules (operators), in combination with an appropriate control strategy, to move through the problem space until a path from **initial state to a goal state** is found. This process is known as **search**. A solution path is a path in state space from s_0(initial sate) to G (Goal state).

Example1: State space representation of Water-Jug problem (WJP):

Problem statement:

Given two jugs, a 4-gallon and a 3-gallon, both of which do not have measuring indicators on them. The jugs can be filled with water with the help of a pump that is available (as shown in figure 4)

The question is "how can you get exactly 2 gallons of water into 4-gallon jug".

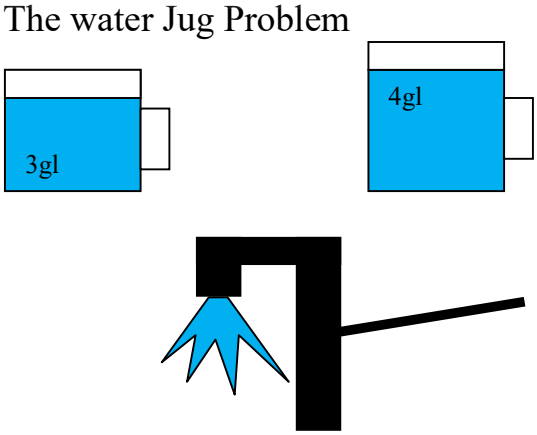


Fig 4 The Water Jug problem

The state space or production rule of this problem can be defined as a collection of ordered pairs of integers (x,y) where x=0,1,2,3,4and y= 0,1,2,or 3.
In the order pair (x,y), x is the amount of water in four-gallon jug and y is the amount of water in the three-gallon jug.
State space: all possible combination of (x,y)
The **start state** is (0,0) and
Goalstate is $(2, n), where\ n = 0,1,2,3$
The following table-1 shows the set of ***production rules*** (actions) that can be used to change one state to another.
Table-1 : Production rules for Water Jug problem

Production Rules		
Rule No	**Production**	**Meaning**
R1	$(x, y \mid x < 4) \rightarrow (4, y)$	Fill 4-gallon jug
R2	$(x, y \mid y < 3) \rightarrow (x, 3)$	Fill 3-gallon jug
R3	$(x, y \mid x > 0) \rightarrow (0, y)$	Empty 4-gallon jug
R4	$(x, y \mid y > 0) \rightarrow (x, 0)$	Empty 3-gallon jug
R5	$(x, y \mid x + y \geq 4\ and\ y > 0) \rightarrow (4, y - (4 - x))$	Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full
R6	$(x, y \mid x + y \geq 3\ and\ x > 0) \rightarrow (x - (3 - y), 3)$	Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full
R7	$(x, y \mid x + y \leq 4\ and\ y > 0) \rightarrow (x + y, 0)$	Pour all water from 3-gallon jug into 4-gallon jug.
R8	$(x, y \mid x + y \leq 3\ and\ x > 0) \rightarrow (0, x + y)$	Pour all water from 4-gallon jug into 3-gallon jug.
R9	$(x, y \mid x > 0) \rightarrow (x - d, y)$	Pure some water d out from 4-gallon jug
R10	$(x, y \mid y > 0) \rightarrow (x, y - d)$	Pure some water d out from 3-gallon jug

The following 2 solutions are found for the problem "how can you get exactly 2 gallons of water into 4-gallon jug", as shown in Table-2 and in Table-3.

Solution-1:
Table-2Getting exactly 2 gallons of water into 4-gallon jug (solution1)

4-gal jug	**3-gal jug**	**Rule applied**
0	0	Initial state
4	0	R1 {fill 4-gal jug}
1	3	R6 {Pure all water from 4-gal jug to 3-gal jug
1	0	R4 {empty 3-gal jug}
0	1	R8 {Pour water from 4 to 3-gal jug}
4	1	R1 {Fill 4-gal jug}
2	3	R6 {Pour water from 4-gal jug to 3-gal jug until it is full}
2	0	R4 {empty 3-gal jug}

Solution-2

Table-3Getting exactly 2 gallons of water into 4-gallon jug (solution2)

4-gal jug	3-gal jug	Rule applied
0	0	Initial state
0	3	R2 {fill 3-gal jug}
3	0	R7 {Pure all water from 3-gal jug to 4-gal jug
3	3	R2 {fill 3-gal jug}
4	2	R5 {Pour from 3 to 4-gal jug until it is full}
0	2	R3 {Empty 4-gal jug}
2	0	R7 {Pour all water from 3-gal jug to 4-gal jug}

A state space tree for WJP with all possible solution is shown in figure 7.

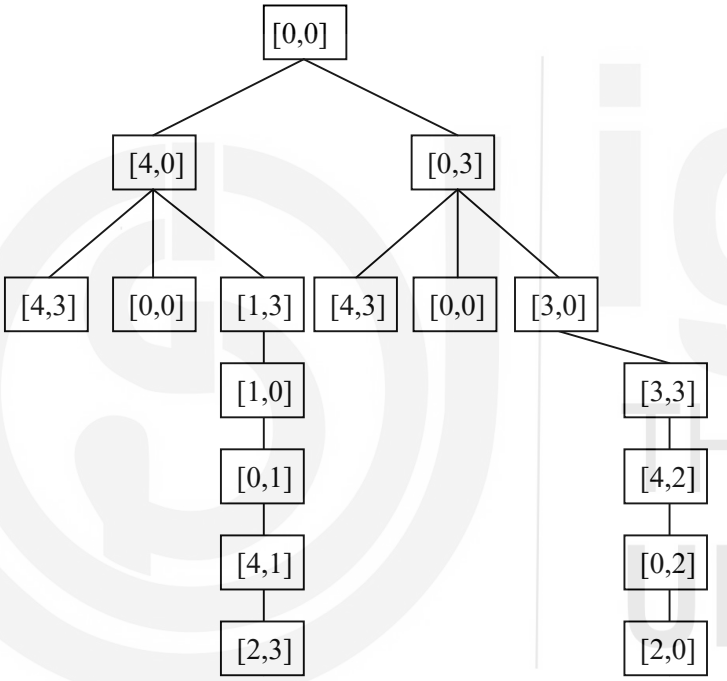


Fig 5 all possible Solution of WJP using state space tree

2.3 Formulation of 8 Puzzle problem

The eight-tile puzzle consists of a 3-by-3 (3×3) square frame board which holds eight (8) movable tiles numbered as 1 to 8. One square is empty, allowing the adjacent tiles to be shifted. The objective of the puzzle is to find a sequence of tile movements that leads from a starting configuration to a goal configuration

The Eight Puzzle Problem formulation:

Given a 3×3 grid with 8 sliding tiles and one "blank"
Initial state: some other configuration of the tiles, for example

3	1	2
4		5
6	7	8

Goal state:

	1	2
3	4	5
6	7	8

Operator: Slide tiles (Move Blank) to reach the goal (as shown below). There are 4 operators that is, "Moving the blank":
Move the blank UP,
Move the blank DOWN,
Move the blank LEFT and
Move the blank RIGHT.

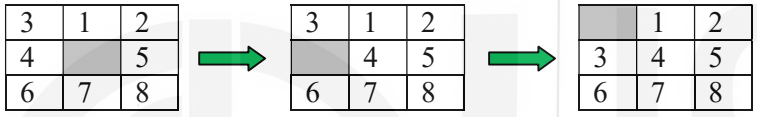


Fig6 Moving Blank LEFT and then UP

Path Cost: Sum of the cost of each path from initial state to goal state. Here cost of each action (blank move) = 1, so cost of a sequence of actions= the number of actions. A optimal solution is one which has a lowest cost path.

Performing State-Space Search: Basic idea:

If the initial state is a goal state, return it.
If not, apply the operators to generate all states that are one step from the initial state (its successors)

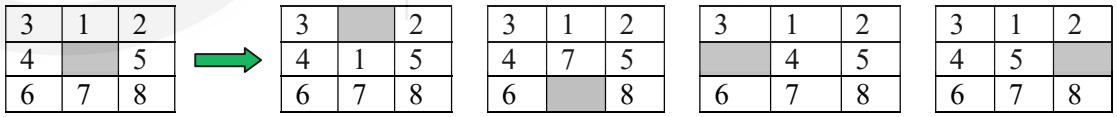


Fig 7 All possible successors for a given initial state

Consider the successor (and their successors…) until you find a goal state.
Different search strategies consider the state in different orders. They may use different data structures to store the states that have yet to be considered.

State-Space Search Tree for 8-Puzzle problem:

The predecessor reference connects the search nodes, creating a data structure known as a tree.

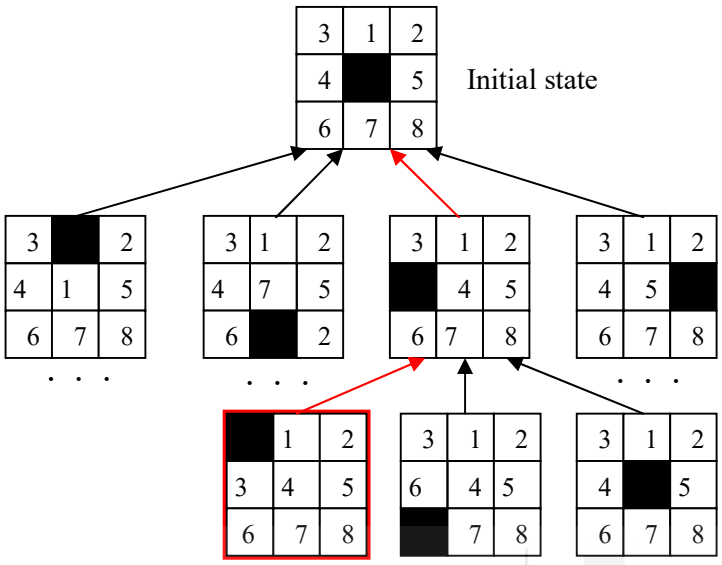


Fig 8 Tracing a tree bottom-up form Goal state to initial sate

When we reach a goal, we trace up the tree to get the solution i.e., the sequence of actions from the initial state to the goal.

Q.1 Find the minimum cost path for the 8-puzzle problem, where the start and goal state are given as follows:

Initial State: {(1,2,3),(4,8,-),(7,6,5)}

Successor State: {(1,2,3),(4,8,5),(7,6,-)}; Move 5 up or – to down

{(1,2,3),(4,8,5),(7,-,6)}; Move 6 right or – to left

{(1,2,3),(4,-,5),(7,8,6)}; Move 8 down or – to up

{(1,2,3),(4,5,-),(7,8,6)}; Move 5 to left or – to right

Goal State: {(1,2,3),(4,5,6),(7,8,-)}; Move 6 to up or – to down

PATH COST=5

2.4 N Queen's Problem-Formulation and Solution

The N-Queen problem is the problem of placing N Queen's (Q1,Q2,Q3,….Qn) on an $N \times N$ chessboard so that no two queens attack each other. The colour of the queens is meaningless in this puzzle, and any queen is assumed to be attack any other. So, a solution requires that no two queens share the same row, column, or diagonal.

The N-queen problem must follow the following rules:

1. There is at most one queen in each column.
2. There is at most one queen in each row.
3. There is at most one queen in each diagonal.

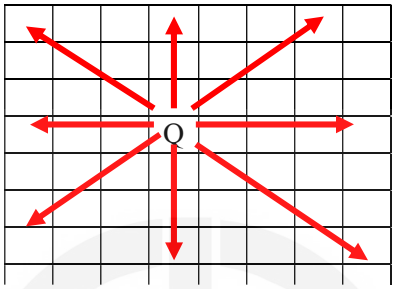


Fig-9 No two queens placed on same row, column or diagonal

The N Queen's problem was originally proposed in 1848 by the chess player Max Bazzel, and over the years, many mathematicians, including Gauss have worked on this puzzle. In 1874, S. Gunther proposed a method of finding solutions by using determinants, and J.W.L. Glaisher refined this approach.

The solutions that differ only by summary operations (rotations and reflections) of the board are counted as one. For 4 queen's problems, there are 16_{C_4} possible arrangements on a 4×4 chessboard and there are only 2 possible solutions for 4 Queen's problem. Note that,there are only 1 unique solution, out of 2 possible solutions as second solution is just a mirror image of the first solution

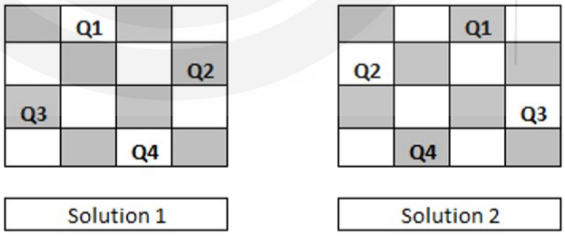


Fig 10 Two possible solutions of 4-Queen's problem

Similarly, the one possible solution for 8-queen's problem is shown in figure 11.

The 8-queen problem is computationally very expensive since the total number of possible arrangements of queen on a 8×8 chessboard is $64_{C_8} = 64!/(56! \text{ x } 8!) \approx 4.4 \times 10^9$. Note that, 8-Queens problem has 92 **distinct** solutions and 12 **unique** solutions, as shown in table-5

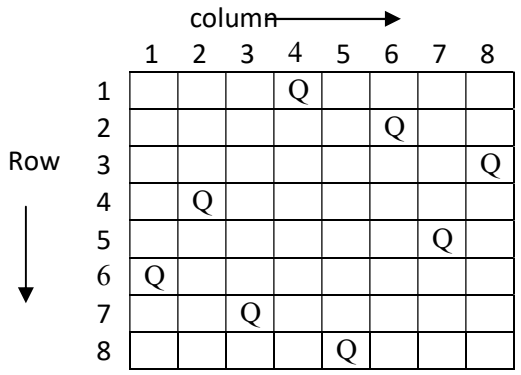


Fig 11 One possible solution of 8 Queen's problem

8-tuple = (4, 6, 8, 2, 7, 1, 3, 5)

The following table-4 summarizes the both distinct and unique solution for the problem of 1-Queen to 26 Queens problem. In general, there is no known formula to find the exact number of solutions for N queen's problem.

Table-4 Solution of N Queen's problem for N=1 to N=26, both Unique and Distinct

No. of Queen's	Unique Solution	Total distinct solutions
1	1	1
2	0	0
3	0	0
4	1	2
5	2	10
6	1	4
7	6	40
8	12	92
9	46	352
10	92	724
11	341	2,680
12	1787	14,200
13	9233	73,712
14	45,752	365,596
15	285,053	2,279,184
16	1,846,955	14,772,184
17	11,977939	95,815,104
18	83,263,591	666,090,624
19	621,012,754	4,968,057,848
20	4,878,666,808	39,029,188,884
21	39,333,324,973	314,666,222,712
22	336,376,244,042	2,691,008,701,644

23	3,029,242,658,210	24,233,937,684,440
24	28,439,272,956,934	227,514,171,973,736
25	275,986,683,743,434	2,207,893,435,808,352
26	2,789,712,466,510,289	22,317,699,616,364,044

2.4.1 Formulation of 4-Queen's problem:

States: any arrangement of 0 to 4 queens on the board

Initial state: 0 queens on the board

Successor function: Add queen in any square

Goal test: 4 queens on the board, none attacked

For the initial state, there are 16 successors. At the next level, each of the states has 15 successors, and so on down the line. This search tree can be restricted by considering only those successors where No queens are attacking each other. To do that, we have to check the new queen with all the other queens on the board. In this way, the answer is found at a depth 4.For the sake of simplicity, you can consider a problem of 4-Queen's and see how 4-queen's problem is solved using the concept of "Backtracking".

2.4.2 State space tree for 4 Queen's Problem

We place queen row-by-row (i.e.,Q_1in row 1, Q_2 in row 2 and so on).
Backtracking gives "all possible solution". If you want optimal solution, then go for Dynamic programming.
Let's see Backtracking method, there are ${}^{16}_{4}C$ ways to place a queen on a 4x4 chess board as shown in the following state space tree (figure 12). In a tree, the value (i, j) means in thei^{th} row, j^{th} queen is placed.

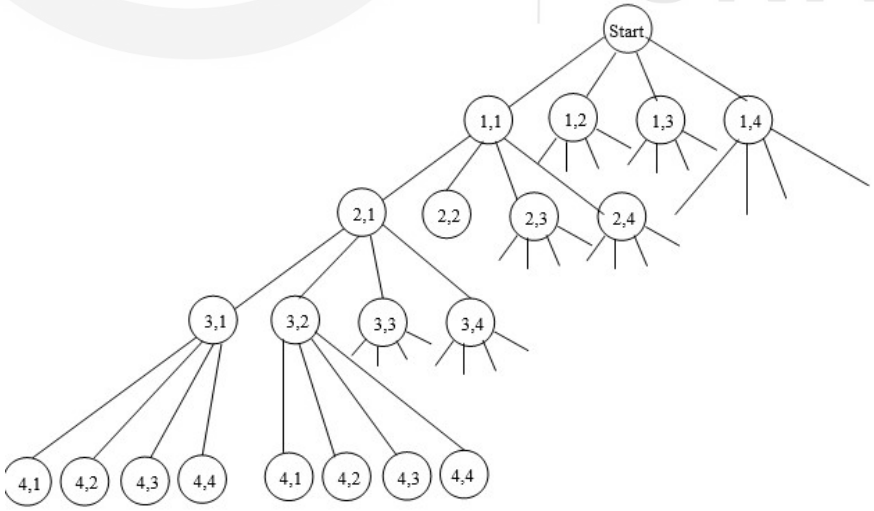


Fig 12 State-space tree showing all possible ways to place a queen on a 4x4 chess board

So, to reduce the size (not anywhere on chess board, since there are ${}^{16}_{4}C$ Possibilities), we place queen row-by-row, and no Queen in same column.This tree is called a permutation tree (**here we avoid same row or same columns but allowing diagonals**)

Total nodes=$1 + 4 + 4 \times 3 + 4 \times 3 \times 2 + 4 \times 3 \times 2 \times 1 = 65$

The edges are labeled by possible values of x_i. Edges from level 1 to level 2 nodes specify the values for x_1. Edges from level $\boldsymbol{i}$ to level $\boldsymbol{i+1}$ are labeled with the values of x_i.
The solution space is defined by all paths from root node to leaf node. There are 4! = 24 leaf nodes are in the tree Nodes are numbered as depth first Search. The state space tree for 4-Queen's problem (**avoid same row or same columns but allowing diagonals**)is shown in figure 13.

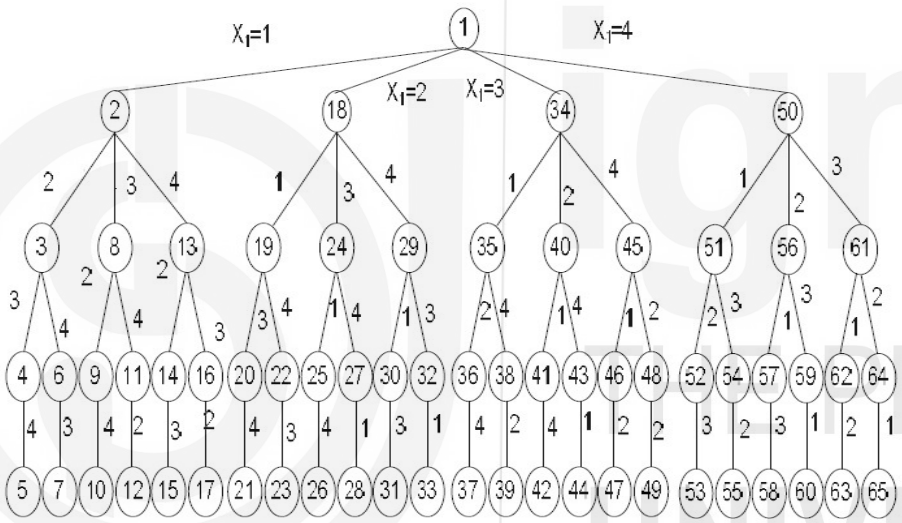


Fig13 State space tree for 4 queen's problems (allowing same diagonal but not same row and same column)

The two solutions are found in the tree, as
$(x_1, x_2, x_3, x_4) = (2,4,1,3)$ and
$(x_1, x_2, x_3, x_4) = (3,1,4,2)$, which is shown in figure-14

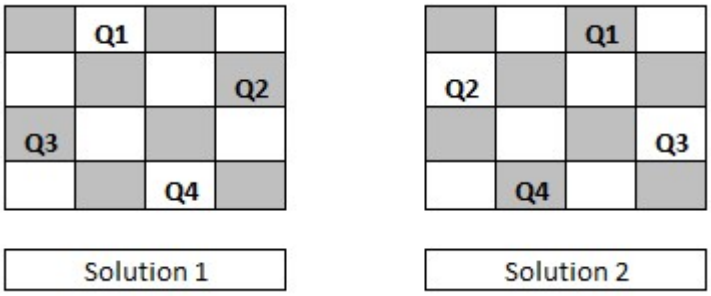


Fig 14 Two possible solutions of 4-Queen's problem

Note that the second solution is just a mirror image of the first solution.

We can further reduce the search space, as shown in figure 6 by avoiding diagonal also. Now you can avoid the **same row, avoid same columns and avoid same diagonals**, while placing any queen. In this case, the state space tree is look like as shown in figure 15.

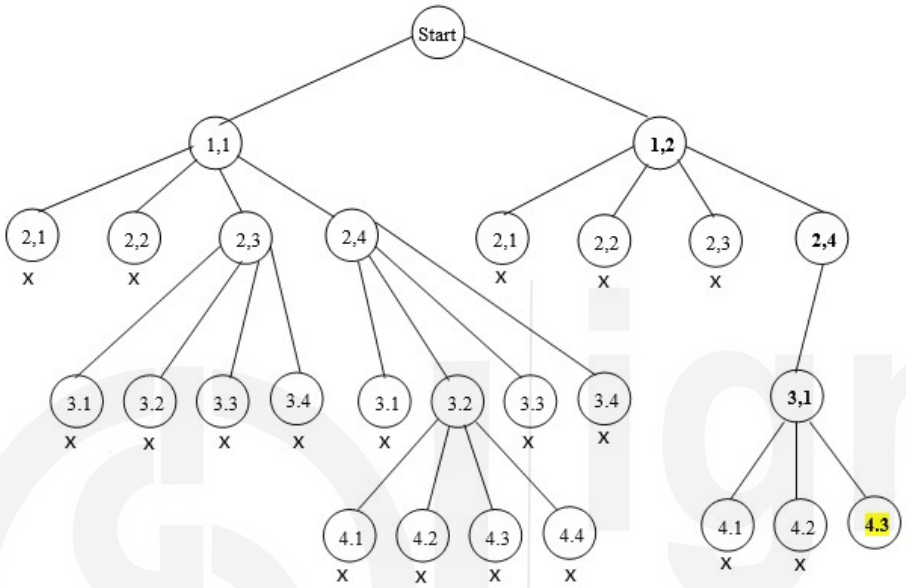


Fig 15 State space tree for 4 queen's problem (avoiding same row, columns, and diagonal)

Note that Queens are placed row-by-row, that is,Q_1in row 1, Q_2 in row 2 and so on. In a tree, node (1,1) is a promising node (no queen attack) as Q_1 is placed in 1st row and 1st column. Node (2,1) is non promising node, because we cannot place Q_2in the same column (as Q_1 is already placed in column 1). Note that nonpromising node is marked as ×. So, we try (2,2) again nonpromising (due to same column), next try (2,3), it's a promising node, so proceed and try to place 3rd queen on 3rd row. But in 3rd row, all positions (3,1),(3,2),(3,3) and (3,4) are non promising and we cannot place the Q_3 in any of this position. So, we backtrack to (1,1) and try for (2,4) and so on.Backtracking approach gives "all possible solution". Figure 7 shows one possible solution for 4-Queen's problem as {(1,2),(2,4),(3,1),(4,3)}. This can also be written as $(x_1, x_2, x_3, x_4) = (2,4,1,3)$. There are 2 possible solution of 4-Queen's problem. Another solution is $(x_1, x_2, x_3, x_4) = (3,1,4,2)$, which is a mirror image of 1st solution.

2.4.3 Backtracking Approach to solve N queen's Problem:

Consider the chess board squares indices of the 2-Dimentional array [1…n,1…n]. we observe that every element on the same diagonal that rows from the upper left to the right has same $(row - column)$ value. It is called ***left diaonals***. Similarly, every elemnet on the same diagonal that goes from the upper rigth to the lower left has same $(row + column)$ value. This is called

Right Diagonals. For example consider a 5×5 chessboard as [1…5,1…5] (as shown in figure 16).

Case1:(Left diagonal):- suppose queen's are palced in same diadinal in locations: (1,2),(2,3),(3,4),(4,5) or (1,4),(2,5) or any other same left diagonal value. Observe that every element on the same diagonal has the same $(row - column)$value. Similarly,

Case2:(right diagonal):- suppose queen's are palced in same diadinal in locations: (1,3),(2,2),(3,1) or (1,4),(2,3),(3,2),(4,1) or any other same right diagonal value. Observe that every element on the same diagonal has the same $(row + column)$value.

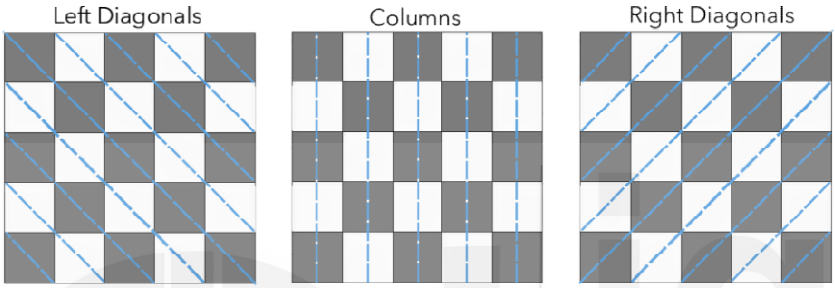


Fig 16 Left diagonal and right diagonal for 5×5 chessboard.

Suppose two queen's are placed at position (i, j) and (k, l) then they are on the same diagonal if and only if:

$(i - j) = (k - l)$ *or* $(j - l) = (i - k)$ ----------(1) [left diagonal]

$(i + j) = (k + l)$ *or* $(j - l) = (k - i)$ ---------- (2) [right diagonal]

From equation (1) and (2), we can combine and write a one condition to check diagonal as: $\boldsymbol{abs(j - l) = abs(i - k).}$

Algorithm NQueen(k,n)

// This procedure prints all possible placement of n queen's on $n \times n$
//chessboard so that they are non-attacking.
{
1. for i=1 to n do
2. {
3. if place(k,i) then
4. {
5. x[k]=i;
6. if (k==n) then print(x[1….n])
7. else
8. NQueen(k+1,n);
9. }

```
10.  }
11.  }

Algorithm place(k,i)
  // This algorithm return true, if a queen can be placed in kth row ith
  //column. Else it return false. X[] is a global array. Abs® returns
  //absolute value of r.

1. {
2.    for j=1 to k-1 do
3.     {
4.if(x[j]=i)      // in the same column
5.Or (abs(x[j]-i)==abs(j-k))// in the
                          //same diagonal
6.          Return false;
7.       }
8.      Return true;
9.     }
```

☞ **Check Your Progress 1**

Q.1 What are the various factors need to be taken into consideration when developing a statespace representation?

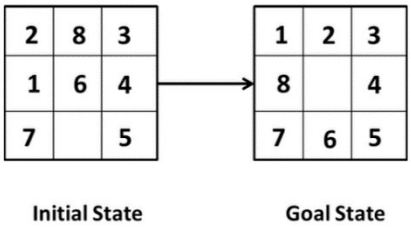
Q.2 Consider the following Missionaries and cannibal problem:

Three missionaries and three cannibals are side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side, without ever leaving a group of missionaries outnumbered by cannibals.

a) Formulate the missionaries and cannibal problem.
b) Solve the problem formulated in part (a)
c) Draw the state- space search graph for solving this problem.

Q.3 Draw a state space tree representation to solve Tower of Hanoi problem. (Hint: You can take number of disk n=2 or 3).

Q.4: Draw the state space tree for the following 8-puzzle problem, where the start and goal state are given below. Also Find the minimum cost path for this 8-puzzle problem. Each blank move is having cost=1.



Q.5 Discuss a Backtracking algorithm to solve a N-Queen's problem. Draw a state space tree to solve a 4-Queen's problem.

2.5 Adversarial Search-Two agent search

In computer science, a search algorithm is an algorithm for finding an item with specified properties among a collection of items. The items may be stored individually as records in a database; or may be elements of a search space defined by a mathematical formula or procedure.
Adversarial search is a **game-playing** technique where the agents are surrounded by a competitive environment. A conflicting goal is given to the agents (multiagent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search. Here, game-playing means discussing those games where **human intelligence** and **logic factor** is used, excluding other factors such as **luck factor**. **Tic-tac-toe, chess, checkers**, etc., are such type of games where no luck factor works, only mind works.
Mathematically, this search is based on the concept of **'Game Theory.'** ***According to game theory, a game is played between two players. To complete the game, one has to win the game and the other loses automatically.'***

Techniques required to get the best optimal solution
There is always a need to choose those algorithms which provide the best optimal solution in a limited time. So, we use the following techniques which could fulfil our requirements:

- **Pruning:** A technique which allows ignoring the unwanted portions of a search tree which make no difference in its final result.
- **Heuristic Evaluation Function:** It allows to approximate the cost value at each level of the search tree, before reaching the goal node.

2.5.1 Elements of Game Playing search

To play a game, we use a game tree to know all the possible choices and to pick the best one out. There are following elements of a game-playing:

- **S_0:** It is the **initial state** from where a game begins.
- **PLAYER (s):** It defines which player is having the current turn to make a move in the state.
- **ACTIONS (s):** It defines the set of legal moves that a player can make to change the state.
- **RESULT (s, a):** It is a transition model which defines the result of a move.
- **TERMINAL-TEST (s)**: It defines that the game has ended (or over) and returns true. States where the game has ended are called **terminal states.**
- **UTILITY (s,p):** It defines the final value with which the game has ended. This function is also known as **Objective function** or **Payoff function**. This utility function gives a numeric value for the outcome of a game i.e.

For example, in **chess, tic-tac-toe,** we have two or three possible outcomes. Either win or lose, or draw the match, which we can represent by the values **+1,-1 or 0.** In other word we can say that **(-1):** if the PLAYER loses, **(+1),** if the PLAYER wins and **(0):** If there is a draw between the PLAYERS.

Let's understand the working of the elements with the help of a game tree designed for **tic-tac-toe**. Here, the node represents the game state and edges represent the moves taken by the players. The root of the tree is the initial state. Next level is all of MAX's moves, then next level is all of MIN's moves and so on. Note that root has 9 blank square (MAX), level 1 has 8 blank squares (MIN), level 2 has 7 blank square (MAX) and so on.

Objective:
Player1: Maximize outcome and **Player2**: Minimize outcome
Terminal (goal) state:
utility: -1, 0, +1 (that is win for X is +1 and win for O is -1 and 0 for draw. The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX. High values are assumed to be good for MAX and bad for MIN (which is how the players get their

names). It is MAX's job to use the search tree to determine the best move. Note that if MAX win then Utility value is +1, if MIN wins then utility value is -1 and if DRAW then utility value is 0.

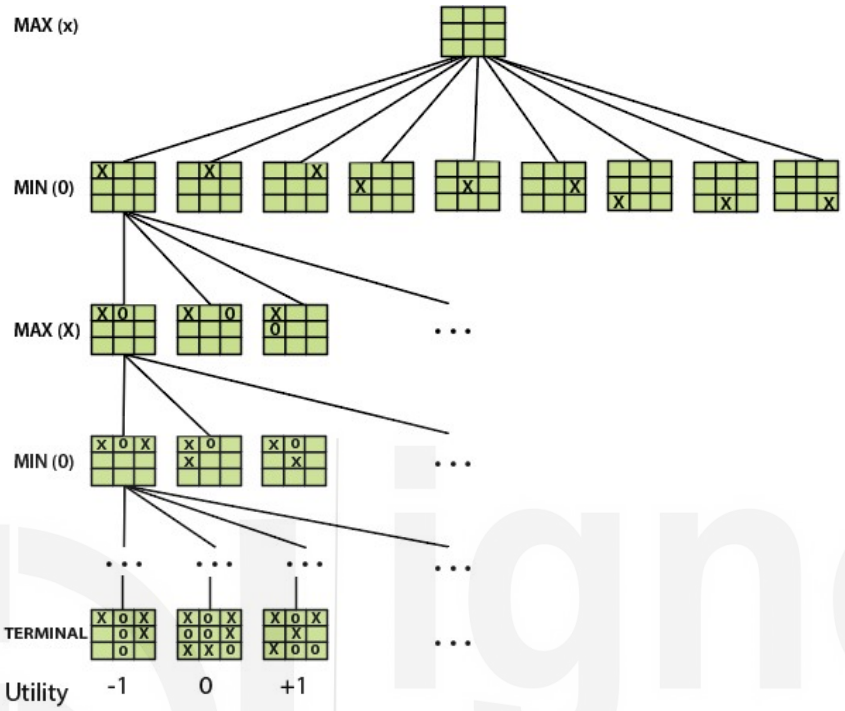


Fig 17**A game-tree for tic-tac-toe**

In a tic-tac-toe game playing, as shown in figure 17, we have the following elements:

- **INITIAL STATE (S_0):** The top node in the game-tree represents the initial state in the tree and shows all the possible choice to pick out one.
- **PLAYER (s):** There are two players, **MAX and MIN**. **MAX** begins the game by picking one best move and place **X** in the empty square box.
- **ACTIONS (s):** Both the players can make moves in the empty boxes chance by chance.
- **RESULT (s, a):** The moves made by **MIN** and **MAX** will decide the outcome of the game.
- **TERMINAL-TEST(s):** When all the empty boxes will be filled, it will be the terminating state of the game.
- **UTILITY:** At the end, we will get to know who wins: **MAX** or **MIN,** and accordingly, the price will be given to them**.** If MAX win then Utility value is +1, if MIN wins then utility value is -1 and if DRAW then utility value is 0.

2.5.2 Issues in Adversarial search

In a **normal search**, we follow a sequence of actions to reach the goal or to finish the game optimally. But in an **adversarial search**, the result depends on the players which will decide the result of the game. It is also obvious that the solution for the goal state will be an optimal solution because the player will try to win the game with the shortest path and under limited time. **Minimaxsearch Algorithm**is an example of adversarial search. **Alpha-beta Pruning** in this is used to reduce search space.

2.6 Min-Max search strategy

In artificial intelligence, minimax is a **decision-making** strategy under **game theory,** which is used to minimize the losing chances in a game and to maximize the winning chances. This strategy is also known as '**Min-Max,' 'MM,' or 'Saddle point.'** Basically, it is a two-player game strategy where *if one wins, the other lose the game*. This strategy simulates those games that we play in our day-to-day life. Like, if two persons are playing chess, the result will be in favour of one player and will againstthe other one. The person who will make his bes*t try, efforts as well as cleverness, will surely win.*
We can easily understand this strategy via **game tree**-where nodes are the states of the game, and edges are moves that were made *by the players in the game*. Players will be two namely:

- **MIN:** Decrease the chances of **MAX** to win the game.
- **MAX:** Increases his chances of winning the game.

They both play the game alternatively, i.e., turn by turn and following the above strategy, i.e., if one wins, the other will definitely lose it. Both players look at one another as competitors and will try to defeat one-another, giving their best.
In minimax strategy, the result of the game or the utility value is generated by a **heuristic function** by propagating from the initial node to the root node. It follows the **backtracking technique** and backtracks to find the best choice. MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX's utility value.

2.6.1 MINIMAX Algorithm:

MINIMAX algorithm is a backtracking algorithm where it backtracks to pick the best move out of several choices. MINIMAX strategy follows the **DFS (Depth-first search)** concept. Here, we have two players **MIN and MAX,** and the game is played alternatively between them, i.e., when **MAX** made a move, then the next turn is of **MIN.** It means the move made by MAX is fixed and, he cannot change it. The same concept is followed in DFS strategy, i.e., we follow the same path and cannot change in the middle. That's why in MINIMAX algorithm, instead of BFS, we follow DFS. The following steps are used in MINMAX algorithm:

- Generate the whole game tree, all the way down to the terminal states.

- Apply the utility function to each terminal state to get its value.
- Use the utility of the terminal states to determine the utility of the nodes one level higher up in the search tree.
- Continue backing up the values from the leaf nodes toward the root, one layer at a time.
- Eventually, the backed-up values reached the top of the tree; at that point, MAX chooses the move that leads to the highest value.

This is called a minimax decision, because it maximizes the utility under the assumption that the opponent will play perfectly to minimize it. To better understand the concept, consider the following game tree or search tree as shown in figure 18.

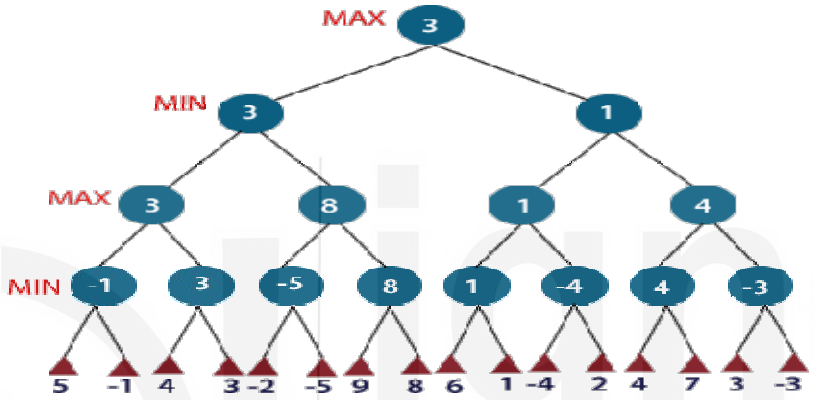


Fig 18 Two player game tree

In the above figure 2, the two players **MAX** and **MIN** are there. **MAX** starts the game by choosing one path and propagating all the nodes of that path. Now, **MAX** will backtrack to the initial node and choose the best path where his utility value will be the maximum. After this, its **MIN** chance. **MIN** will also propagate through a path and again will backtrack, but **MIN** will choose the path which could minimize **MAX** winning chances or the utility value.
So, if the level is minimizing, the node will accept the minimum value from the successor nodes. If the level is maximizing, the node will accept the maximum value from the successor.

In other word we can say that - Minimax is a decision rule algorithm, which is represented as a game-tree. It has applications in decision theory, game theory, statistics and philosophy. Minimax is applied in two player games. The one is the MIN and the other is the MAX player. By agreement the root of the game-tree represents the MAX player. It is assumed that each player aims to do the best move for himself and therefore the worst move for his opponent in order to win the game. The question may arise "How to deal with the contingency problem?" The answer is:

- Assuming that the opponent is rational and always optimizes its behaviour (opposite to us) we consider the best response. opponent's
- Then the minimax algorithm determines the best move

2.6.2 Working of Minimax Algorithm:

Minimax is applicable for decision making for two agent systems participating in competitive environment. These two players P1 and P2, also known as MIN and MAX player, maximizes and minimizes utility value of heuristics function. Algorithm uses recursion to search through game tree and compute minimax decision for current state. We traverse the complete game tree in a depth-first search (DFS) manner to explore the node. MAX player always select the maximum value and MIN always select the minimum value from its successor's node. The initial value of MAX and MIN is set to as $MAX = -\infty$ and $MIN = +\infty$. This is a worst value assigned initially and as the algorithm progress these values are changes and finally, we get the optimal value**.**

Example1: Let's take an example of two-player game tree search (shown in figure 19a) to understand the working of Minimax algorithm.

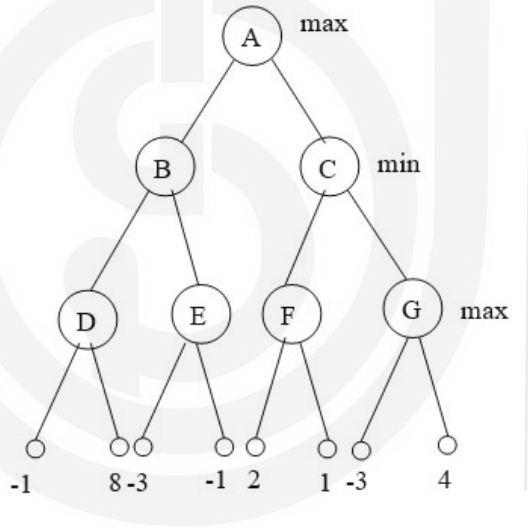


Fig 19a Two player game tree

The initial value of MAX and MIN is set to as $MAX = -\infty$ and $MIN = +\infty$. The tree is traversed in a DFS manner. So, we start from node A, then move to node B and then D.

Now at node D[$\boldsymbol{MAX = -\infty}$**].** Now, at D, it first checks the left child (which is a terminal node) with value-1. This node returns a value of $MAX = (-\infty, \mathbf{-1}) = \mathbf{-1}$. So,modified value at node D is $[MAX = -1]$. Next, we proceedfor right child of Node D (which has terminal value 8) and compare this value (8) with previous value at node D. that is $MAX = max(\mathbf{-1}, \mathbf{8}) = \mathbf{8}$. So final value at node D is 8.

Similarly,

the value at node E (which is a Max node) is $MAX = max(-\infty, \mathbf{-3}) = \mathbf{-3}$, then $\boldsymbol{max(-3, -1) = -1.}$

So, at node B, which is at MIN level, select the minimum value from its successor node D and E as $MIN = min(8, -1) = -1$

Similarly, the value at node F (which is also Max node) is $MAX = max(-\infty, \mathbf{2}) = \mathbf{2}, \boldsymbol{then}\ \boldsymbol{max}(\mathbf{2}, \mathbf{1}) = \mathbf{2},$ and
The value at node G (which is also MAX node) is
$MAX = max(-\infty, -\mathbf{3}) = -\mathbf{3},$ and then$max(-3,4) = 4$**.**
Thus, at node C, which is also at MIN level, select the minimum value from its successor node F and G as $MIN = min(2,4) = 2.$

Now, the value at node B and C is -1 and 2 respectively.

Thus, finally, the value at node A, which is at MAX level, is

$MAX = max(-1,2) = 2.$

The final game tree with max or min value at each node and optimal path, with shaded line A→C→F→2, is shown in the following figure 19(b).

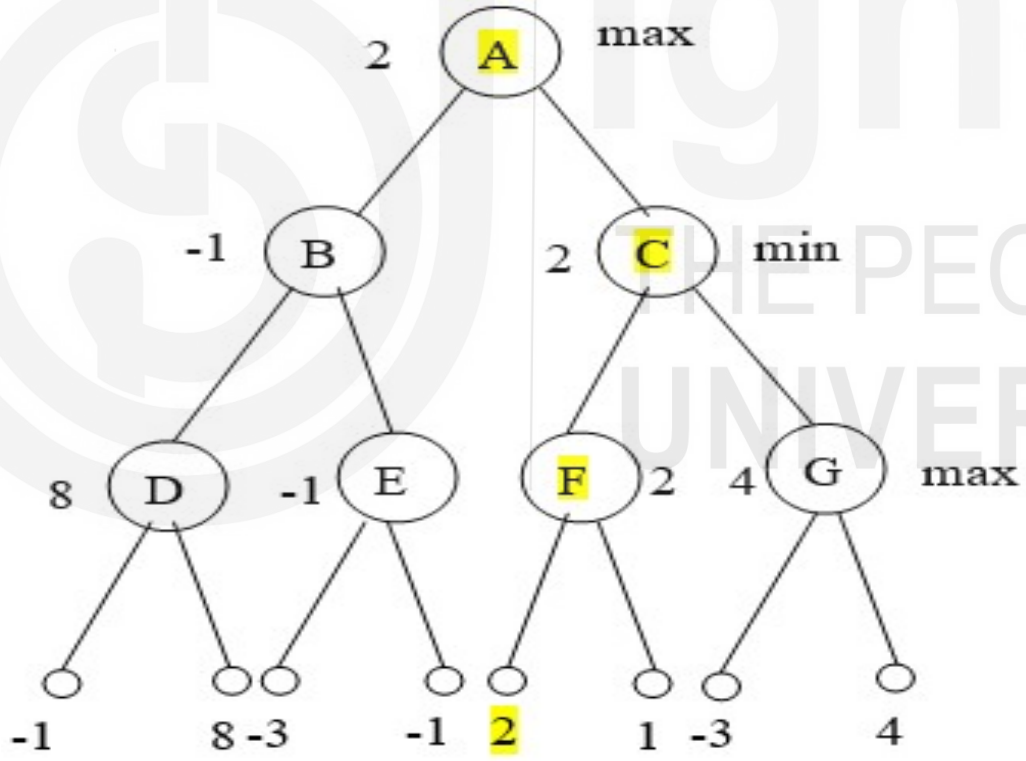


Fig 19(b) Game tree with final value at each node with optimal path

Example2 Consider the following two-player game tree search. The working of Minimax algorithmis illustrated from fig (a)-fig(k)

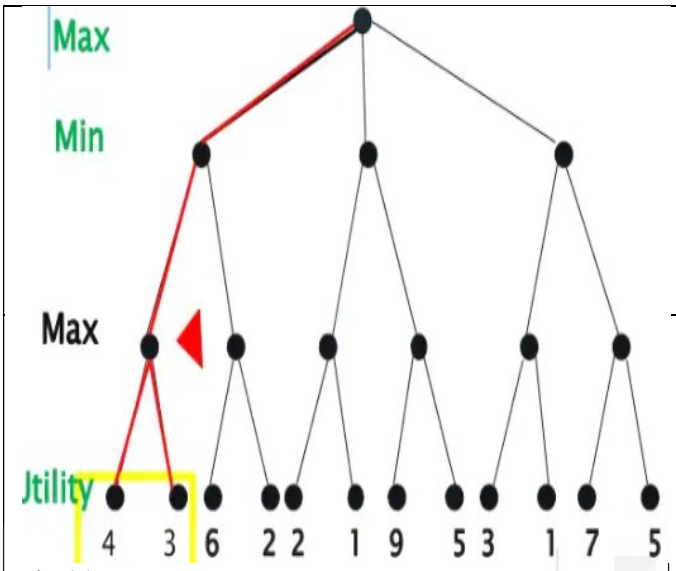


Fig (a)

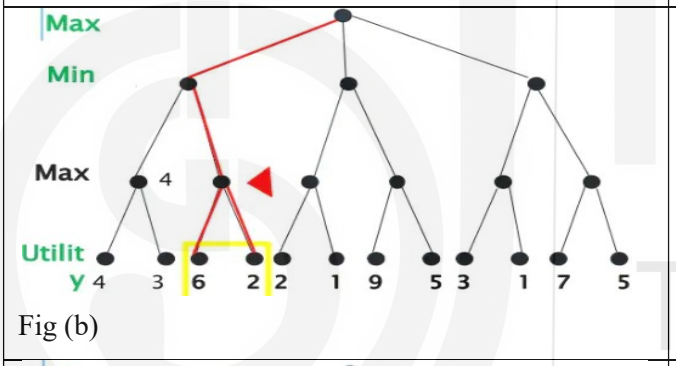


Fig (b)

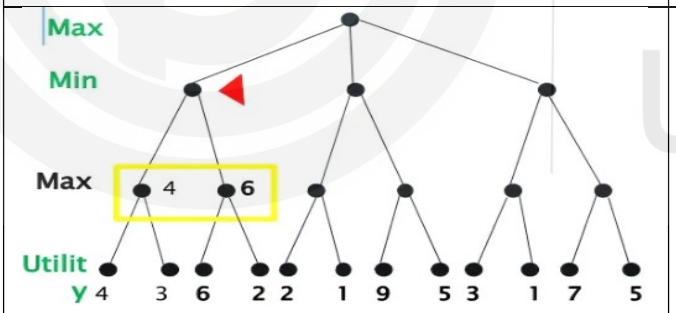


Fig (c)

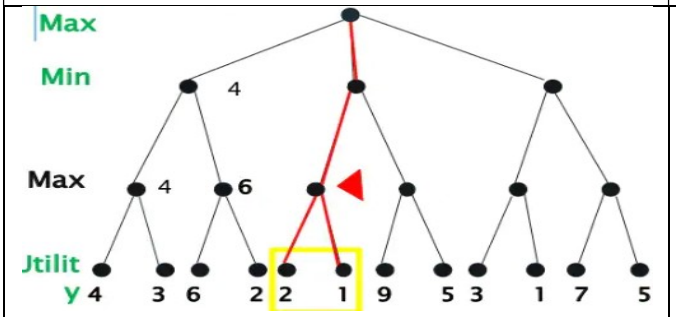


Fig (d)

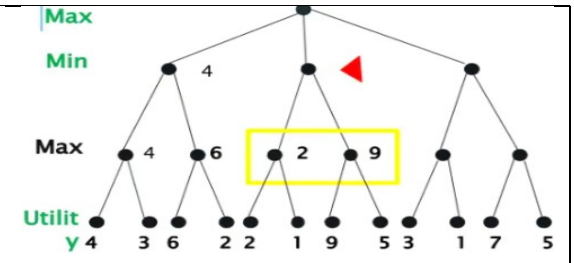


Fig (f)

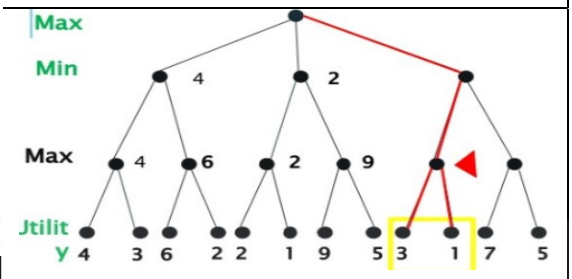


Fig (g)

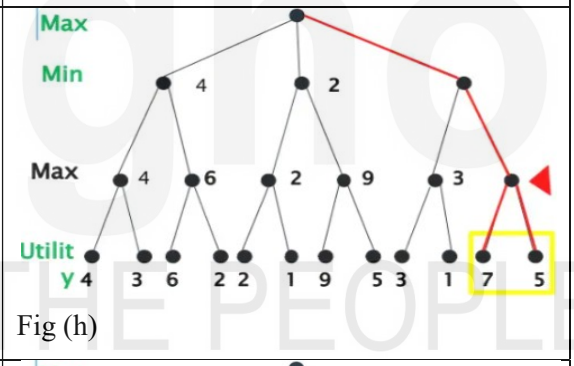


Fig (h)

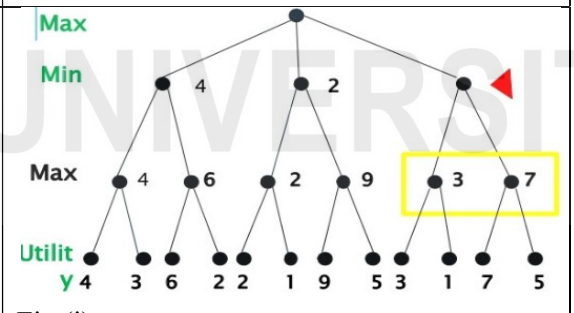


Fig (i)

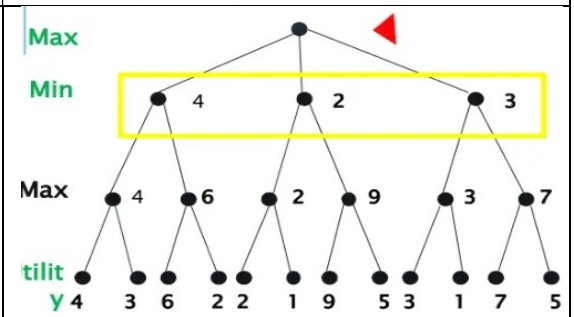


Fig (j)

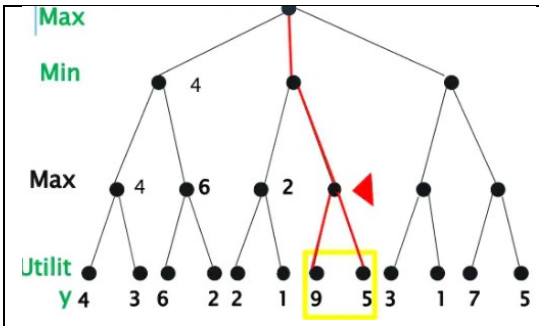


Fig (e)

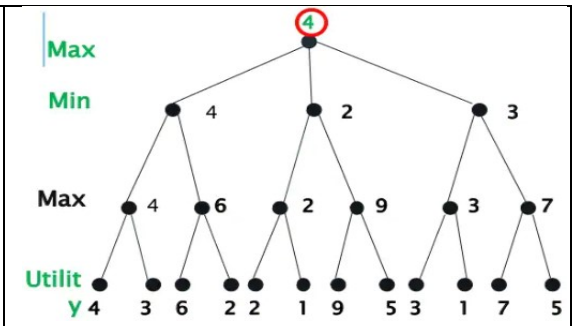


Fig (k)

2.6.3 Properties of Minimax Algorithm:

1. **Complete:** Minimax algorithm is complete, if the tree is finite.
2. Optimal: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution is said to be an optimal solution. Minimax is Optimal.
3. **Time complexity:** $O(b^m)$, where b: Branching Factor and m is the maximum depth of the game tree.
4. **Space complexity:** $O(bm)$

For example, in chess playing game b = 35, m ≈100 for "reasonable" games. In this case exact solution completely infeasible

2.6.4 Advantages and disadvantages of Minimax search

Advantages:

- Returns an optimal action, assuming perfect opponent play.
- Minimax is the simplest possible (reasonable) game search algorithm.

Disadvantages:

- It's completely infeasible in practice.
- When the search tree is too large, we need to limit the search depth and apply an evaluation function to the cut-off states.

2.7 Alpha-beta Pruning

The drawback of Minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths. This increases its time complexity. If b is the branching factor and

d is the depth of the tree, then time complexity of MINIMAX algorithm is $O(b^d)$ that is exponential. But as we know, the performance measure is the first consideration for any optimal algorithm. Alpha-beta pruning is a method to reduce (prone) search space. Using Alpha-Beta pruning, the Minimax algorithm is modified. Therefore, alpha-beta pruning reduces this drawback of minimax strategy by less exploring the nodes of the search tree.

The method used in alpha-beta pruning is that its **cut-off the search** by exploring a smaller number of nodes. It makes the same moves as a minimax algorithm does, but it prunes the unwanted branches using the pruning technique (discussed in adversarial search). Alpha-beta pruning works on two threshold values, i.e., α **(alpha)** and β **(beta).**

- $\boldsymbol{\alpha}$ **:** It is the best highest value; a **MAX** player can have. The initial value of $\boldsymbol{\alpha}$ is set to negative infinity value, that is

  $\boldsymbol{\alpha} = -\infty$. As the algorithm progress its value may change and finally get the best (highest) value**.**
- $\boldsymbol{\beta}$ **:** It is the best lowest value; a **MIN** player can have. The initial value of $\boldsymbol{\beta}$ is set to positive infinity value, that is

  $\boldsymbol{\beta} = +\infty$. As the algorithm progress its value may change and finally get the best (lowest) value**.**

So, each MAX node has $\boldsymbol{\alpha}$-value, which never decreases, and each MIN node has $\boldsymbol{\beta}$-value, which never increases. The main condition which required for alpha-beta pruning is $\boldsymbol{\alpha \geq \beta}$**,** that is if$\boldsymbol{\alpha \geq \beta}$**,** then prune (cut) the branches otherwise proceed.

Note: Alpha-beta pruning technique can be applied to trees of any depth, and it is possible to prune the entire sub-trees easily.

2.7.1 Working of Alpha-beta Pruning

As we know there are two-parameter is defined for Alpha-beta pruning, namely alpha ($\boldsymbol{\alpha}$)and beta($\boldsymbol{\beta}$)**.** The initial value of alpha and beta is set to as $\boldsymbol{\alpha = -\infty}$ **and** $\boldsymbol{\beta = +\infty}$.As the algorithm progresses its values are changes accordingly. Note that in Alpha-beta pruning (cut), at any node in a tree, if $\boldsymbol{\alpha \geq \beta}$**,** then prune (cut) the next branch else search is continued. Note the following point for alpha-beta pruning:

- The MAX player will only update the value of$\boldsymbol{\alpha}$(on MAX level).
- The MIN player will only update the value of $\boldsymbol{\beta}$(on MIN level).
- We will only pass the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$value from top to bottom (that is from any parent to child node, but never from child to parent node).
- While backtracking the tree, the node values will be passed to upper node instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**.**

- Before going to next branch of the node in a tree, we check the value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$. If the value of$\boldsymbol{\alpha \geq \beta}$, then prune (cut) the next (unnecessary) branches (i.e., no need to search the remaining branches where the condition $\boldsymbol{\alpha \geq \beta}$is satisfied) else search continued.

Consider the below example of a game tree where **P** and **Q** are two players. The game will be played alternatively, i.e., chance by chance. Let, **P** be the player who will try to win the game by maximizing its winning chances. **Q** is the player who will try to minimize **P'**s winning chances. Here, $\boldsymbol{\alpha}$ will represent the maximum value of the nodes, which will be the value for **P** as well. $\boldsymbol{\beta}$ will represent the minimum value of the nodes, which will be the value of **Q**.

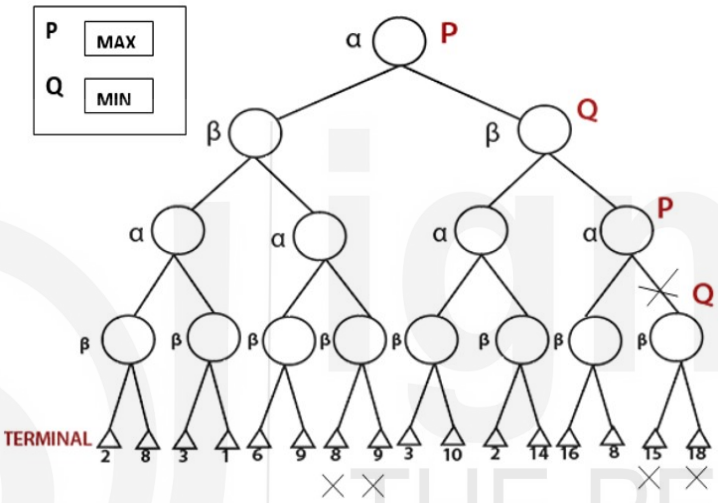


Fig 20 Alpha-beta pruning

- Any one player will start the game. Following the DFS order, the player will choose one path and will reach to its depth, i.e., where he will find the **TERMINAL** value.
- If the game is started by player P, he will choose the maximum value in order to increase its winning chances with maximum utility value.
- If the game is started by player Q, he will choose the minimum value in order to decrease the winning chances of A with the best possible minimum utility value.
- Both will play the game alternatively.
- The game will be started from the last level of the game tree, and the value will be chosen accordingly.
- Like in the figure 5, the game is started by player Q. He will pick the leftmost value of the TERMINAL and fix it for beta ($\boldsymbol{\beta}$). Now, the next TERMINAL value will be compared with the $\boldsymbol{\beta}$-value. If the value will be smaller than or equal to the $\boldsymbol{\beta}$-value, replace it with the current $\boldsymbol{\beta}$-value otherwise no need to replace the value.

- After completing one part, move the achieved $\boldsymbol{\beta}$-value to its upper node and fix it for the other threshold value, i.e.,$\boldsymbol{\alpha}$.
- Now, its P turn, he will pick the best maximum value. P will move to explore the next part only after comparing the values with the current $\boldsymbol{\alpha}$-value. If the value is equal or greater than the current $\boldsymbol{\alpha}$-value, then only it will be replaced otherwise we will prune the values.
- The steps will be repeated unless the result is not obtained.
- So, number of pruned nodes in the above example are **four** and MAX wins the game with the maximum **UTILITY** value, i.e.,**3.**

The rule which will be followed is: **"Explore nodes, if necessary, otherwise prune the unnecessary nodes."**
Note: It is obvious that the result will have the same **UTILITY** value that we may get from the MINIMAX strategy.

Alpha beta cut-off (or pruning):

1. for each node store limit $[\alpha, \beta]$.

2. Update $[\alpha, \beta]$,

where α is the lower bound at max node; it can't decrease.

β is the upper bound at min node; it can't increase.

3. If α value of a max node is greater than β value of its parent ($\alpha \geq \beta$), thesubtree of that max node need not be evaluated (i.e., pruned).

4. If β value of a min node is lesser than α value of its parent ($\beta \leq \alpha$), the subtree of that min node need not be evaluated (i.e., pruned).

Example1: Let's take an example of two-player search tree (Figure 21) to understand the working of alpha-beta pruning.

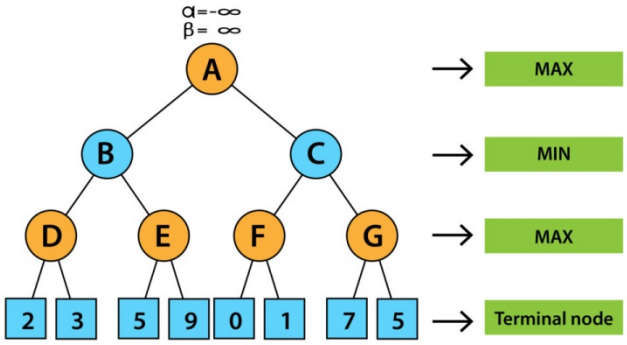


Fig 21 Two player search tree

We initially start the search by setting the initial value of $\boldsymbol{\alpha} = -\infty$and$\boldsymbol{\beta} = +\infty$to root **node A**.

Note the following important point to apply the Alpha-beta pruning:

- We will only pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top to bottom (that is from any parent to child node), but never from child to parent node.
- While backtracking the tree (from bottom to top node), the node values will be passed to upper node instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**.**
- Before exploring the next branch in a tree, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$**.** If **YES,** then prune (cut) the next (unnecessary) branches (i.e., no need to search the remaining branches where the condition $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$is satisfied) else search continued.
- The MAX player will only update the value of$\boldsymbol{\alpha}$(on MAX level) and the MIN player will only update the value of $\boldsymbol{\beta}$(on MIN level).

Step1: We traverse the tree in a depth-first search (DFS) manner and assign (pass) this value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$down to subsequent nodes**B** and then to node**D**as $[\boldsymbol{\alpha} = -\infty; \boldsymbol{\beta} = +\infty]$**.**
Now at node D $[\boldsymbol{\alpha} = -\infty, \boldsymbol{\beta} = +\infty]$**,** Since node D is at MAX level, so only $\boldsymbol{\alpha}$value will be changed. Now, at D, it first checks the left child (which is a terminal node) with value 2. This node returns a value of 2. Now, the value of $\boldsymbol{\alpha}$ **at** node D is calculated as$\alpha = max(-\infty, 2) = 2$. So modified value at node D is $[\alpha = 2, \beta = +\infty]$. To decide whether it's worth looking at its right node or not, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer in NO since $2 \ngeq +\infty$. So, proceed and search is continued for right child of Node D.
The value of right child (terminal node with value=3) of D returns a value 3. Now at D, the value of $\alpha$is compared with terminal node value 3, that is, $\alpha = max(2,3) = 3$.Now the value of

Node(D)=3, and the final values of α and $\beta$is updated at node D as$[\alpha = 3, \beta = +\infty]$ as shown in figure 21(a).

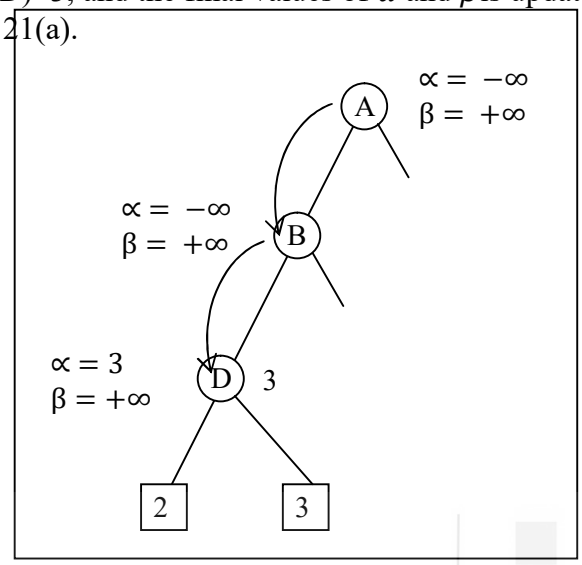


Fig 21(a)

Step 2. We backtrack from node D to B. Note that, while backtracking the node in a tree, the node values of D(=3) will be passed to upper nodeB instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**.**Now the value of node(B)=node(D)=3. Since B is at MIN level, so only$\boldsymbol{\beta}$value will be changed. Now at node **B** $\boldsymbol{[\alpha = -\infty, \beta = 3]}$ (note that β is change from $+\infty$ to 3). Here we again check $\alpha \geq \beta$. It is False, so search is continued on right side of B, as shown in figure (b).

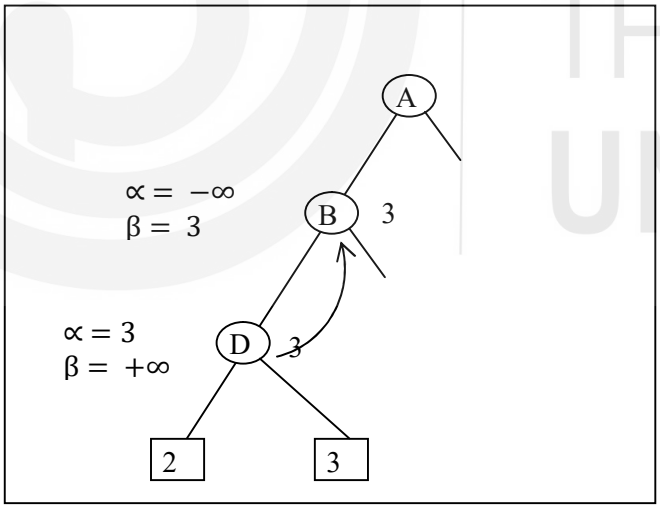


Fig21(b)

Step 3.B now calls E, we pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node B to bottom node E as $[\boldsymbol{\alpha = -\infty, \beta = 3}]$**. Since,** node E is at MAX level, so only $\boldsymbol{\alpha}$value will be change.Now, at E, it first checks the left child (which is a terminal node) with value5. This node returns a value of 5.

Now, the value of $\boldsymbol{\alpha}$ **at** node E is calculated as$\alpha = max(-\infty, 5) = 5$, so value of Node(E)=5 and modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$ at node Eis $[\alpha = 5, \beta = 3]$. To decide whether it's worth looking at its right node or not, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer is**YES**, since $5 \geq 3$. So,we prune (cut) the right branch of E, as shown in figure 21(c).

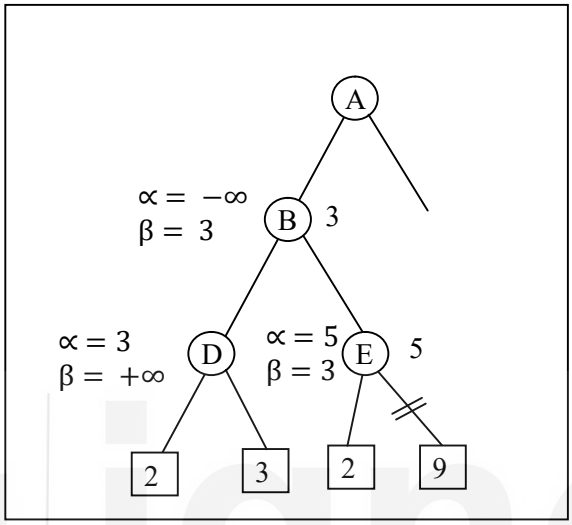


Fig21(c)

Step4. We backtrack from node E to B. Note that, while backtracking, the node values of E(=5) will be passed to upper node B, instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**. E return a value 5 to B**.Since B is at MIN level, so only$\boldsymbol{\beta}$value will be changed. Previously, at node **B** $[\boldsymbol{\alpha = -\infty, \beta = 3}]$**, but now**$\boldsymbol{\beta} = min(3,5) = 3$, so, there is no change in $\boldsymbol{\beta}$ value and value of node(B) is still 3. Thus finally, modified value at node **B is**$[\boldsymbol{\alpha = -\infty, \beta = 3}]$**.**
We backtrack from node B to A. Again, note that, while backtracking the tree, the value of node(B)=3 will be passed to upper node A, instead of values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$**.Now value of Node(A)=3.**

Since A is at MAX level, so only$\boldsymbol{\alpha}$value will be changed. Previously, at node **A** $[\alpha = -\infty, \beta = +\infty]$ and after comparing value of node(B)=3 with old value of α at node A, that is $\alpha = max(-\infty, 3) = 3$. Thus finally, at node A $[\alpha = 3, \beta = +\infty]$ and value of Node(A)=3. we check $\alpha \geq \beta$, it is False, so proceed on right side. Now, we completed the left sub tree of A and proceed towards right subtree, as shown in figure 21(d).

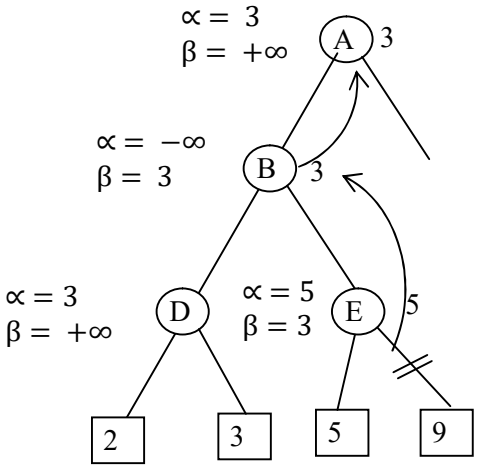


Fig21(d)

Step 5.

Now at node C, we pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node A to bottom node C as $[\boldsymbol{\alpha = 3, \beta} = +\infty]$**. Check,**$\alpha \geq \beta$. The answer in NO. So, search is continued. Now pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node C to bottom node F as $[\boldsymbol{\alpha = 3, \beta} = +\infty]$**.** Since F is at MAX level, so only$\boldsymbol{\alpha}$value will be changed.

Now, at F, it first checks the left child (which is a terminal node) with value0. This node returns a value of 0. Now, the value of $\boldsymbol{\alpha}$ **at** node F is calculated as$\alpha = max(3,0) = 3$. So modified value at node Fis $[\alpha = 3, \beta = +\infty]$. To decide whether it's worth looking at its right node or not, we check $\boldsymbol{\alpha \geq \beta}$. The answer in NO since $3 \ngeq +\infty$. So, proceed and search is continued for right child of Node F.

The value of right child (terminal node with value=1) of F returns a value 1, so finally, value of node(F)=1. Now at F, the value of α is compared with terminal node value 1, that is, $\alpha = max(3,1) = 3$, and the final values of α and β is updated at node F as$[\alpha = 3, \beta = +\infty]$ as shown in figure 21(e).

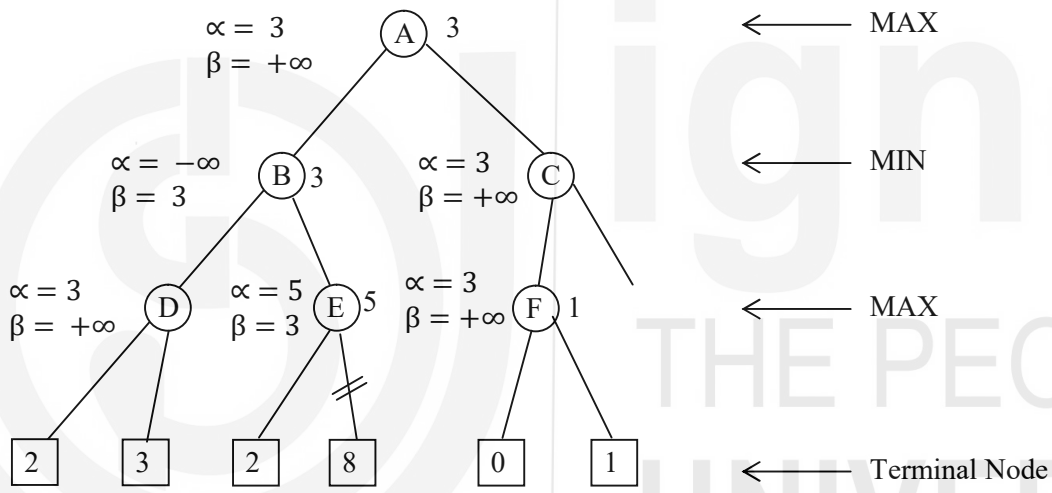


Fig21(e)

6. We backtrack from node F to C. Note that, while backtracking the tree, the node values of F(=3) will be passed to upper node C**.** Now the value of **node(C)=node(F)=1**.

Since C is at MIN level, so only βvalue will be changed. Previously, at node **C** $[\alpha = 3, \beta = +\infty]$. Now, old value of $\boldsymbol{\beta} = +\infty$ is compared with value of node(F)=node(C)=1. That is, $\boldsymbol{\beta} = min(+\infty, 1) = 1$. Thus finally, at node **B** $[\boldsymbol{\alpha = 3, \beta = 1}]$**. Now we check,**$\alpha \geq \beta$. It is TRUE, so we prune (cut) the right branch of node C. That is node G will be pruned and algorithm stop searching on right subtree of node C.

Thus finally, **we** backtrack from node C to A and node C return the value 1 to node A. Since A is a MAX node, so only$\boldsymbol{\alpha}$value will be changed.

Previously, at node **A** $[\alpha = 3, \beta = +\infty]$ and after comparing value of node(C)=1 with old value of α at node A, that is $\alpha = max(3,1) = 3$. Thus finally, at node A $[\alpha = 3, \beta = +\infty]$ and value of Node(A)=3. Now, we completed the right sub tree of A also.

Following is the final game tree, showing the nodes which are computed and nodes which are pruned (cut) during search process of Alpha-beta pruning. Here the optimal value for the maximizer is 3 and there are 3 terminal nodes are pruned (9, 7 and 5). The optimal search path is A→B→D→3.

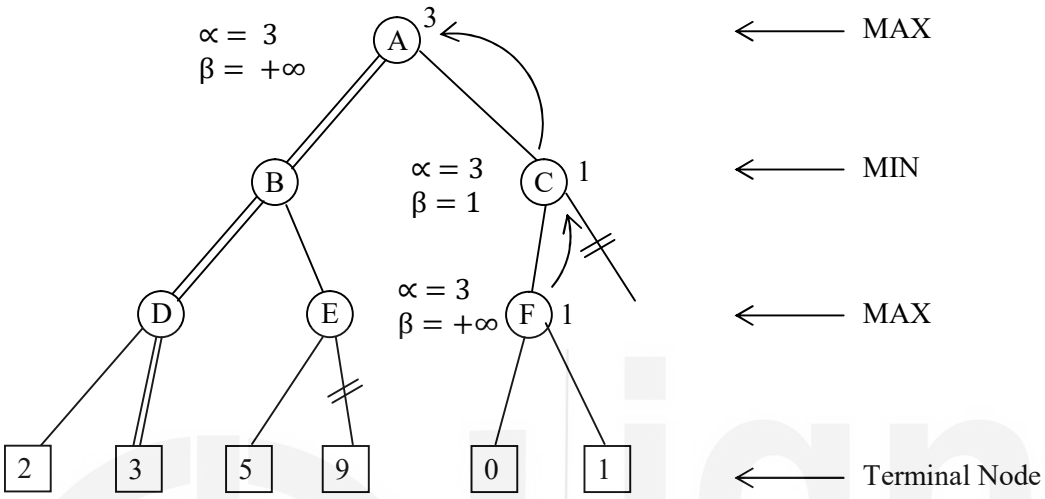


Fig 21(f)

Example2: Consider the following game tree (figure 22) in which root is maximizing node and children are visited from left to right. Find which nodes are pruned by the Alpha-beta pruning.

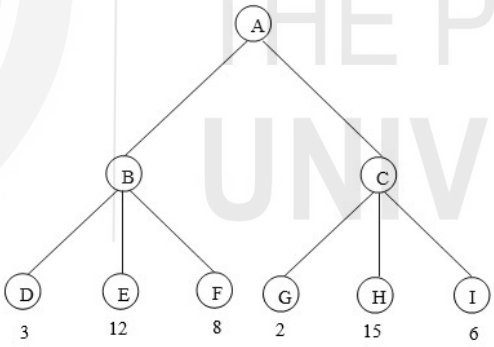


Fig 22 Two Player search tree

Solution:

Step1: We start the search by setting the initial value for node A as$[\boldsymbol{\alpha = -\infty, \beta = +\infty}]$. We traverse the node in depth-first search (DFS) manner so assign the same value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$tonode **B**$[\boldsymbol{\alpha = -\infty, \beta = +\infty}]$**,** Since node B is at MIN level, so only $\boldsymbol{\beta}$value will be changed at node B. Now, at D, it looks at it left child (terminal node), which returns a value 3 to D. So, we compare the old value of β at node B with this terminal node value 3, that is $\beta = \boldsymbol{min}(+\infty, \mathbf{3}) = \mathbf{3}$.So modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$tonode **B** is$[\boldsymbol{\alpha = -\infty}, \beta = 3]$.

To decide whether it worth looking at right subtree of B, we check

$\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer in NO, since $-\infty \not\geq 3$. So, proceed and search is continued for right child of Node B, that is E.

The terminal value of E=12. Now, the value of right child terminal E(=12) is compared with previous old value of $\beta = 3$, that is, $\beta = min(3,12) = 3$.So no change in β value. So, at present, current modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$atnode **B** is same$[\boldsymbol{\alpha} = -\infty, \beta = 3]$. Again check,$\alpha \geq \beta$. The answer in NO. So, proceed and search is continued for right child of Node B, that is F. The terminal value of F=8. The value of right child terminal at F(=8) is compared with previous old value of $\beta = 3$, that is, $\beta = min(8,3) = 3$. So no change in β value. So, finally value of **Node(B)=3** and modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$atnode **B**$[\boldsymbol{\alpha} = -\infty, \beta = 3]$ as shown in figure 22(a)

Step2: We backtrack from node B to A. Note that, while backtracking the tree, the node values of B(=3) will be passed to upper node A, instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**. Now the value of node(A)=node(B)=3.** Since A is at MAX level, so only$\boldsymbol{\alpha}$value will be changed. Previously, at node **A** $[\alpha = -\infty, \beta = +\infty]$ and after comparing value of node(B)=3 with old value of α at node A, that is $\alpha = max(-\infty, 3) = 3$. Thus finally, at node **A** $[\boldsymbol{\alpha = 3, \beta = +\infty}]$ and value of **Node(A)=3**.

To decide whether it's worth looking at its right node of A or not, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer in NO since $3 \not\geq +\infty$. So, proceed and search is continued for right child of Node A. Now, we completed the left sub tree of A and proceed towards right subtree.

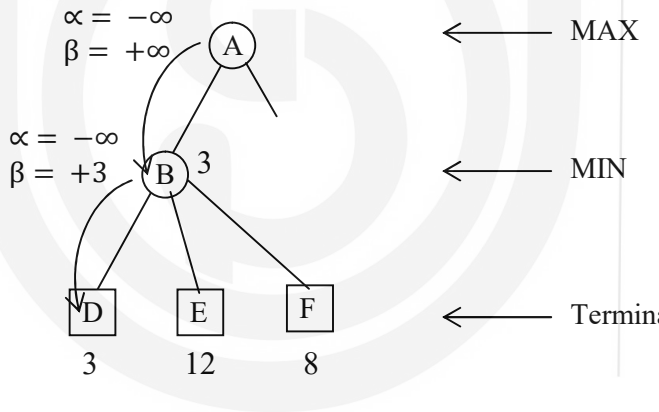


Fig22(a)

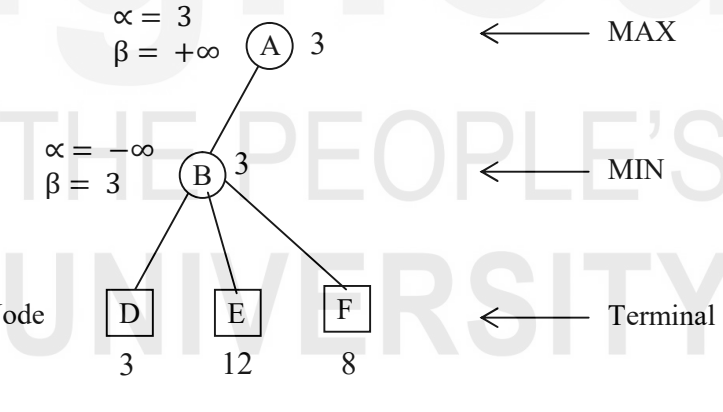


Fig 22(b)

Step 3: Now at node C, we pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node A to bottom node C as $[\boldsymbol{\alpha = 3, \beta = +\infty}]$**. Check,**$\alpha \geq \beta$. The answer in NO. So, continue the search on right side. Since C is at MIN level, so only$\boldsymbol{\beta}$value will be changed.

Now, we first check the left child (terminal) of node C, that is G=2. So, we compare the old value of $\boldsymbol{\beta}$ **at node C with this terminal node value 2, that is** $\boldsymbol{\beta = min(+\infty, 2) = 2}$**. So value of Node(C)=2** and modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$atnode**C** is$[\boldsymbol{\alpha = 3}, \beta = 2]$. Now, before proceed next, we again check$\alpha \geq \beta$. The answer is YES. So,we prune (cut) the right branch of node C. That is **node H and I** will be pruned (cut) and algorithm stop searching on right subtree of node C, as shown in figure 22(c).

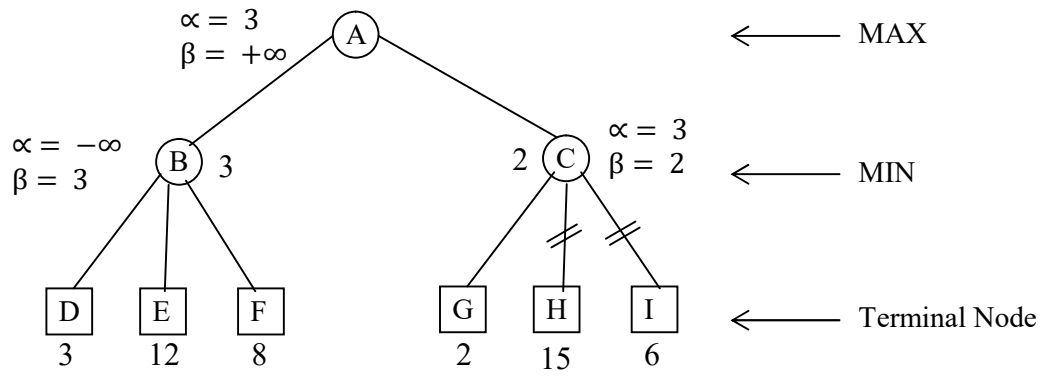


Fig22(c)

Step4: Finally, we backtrack from node C to A. Note that, while backtracking the node in a tree, the node values of C(=2) will be passed to upper node A, instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**.The previous node(A)=3 value is compared with this new node(C)=2 value. The best value at node(A)=**$\alpha = max(3,2) = 3$.

The previous α and β value at node A$[\alpha = 3, \beta = +\infty]$. Since A is at **MAX** level so only α value is change. So, we compare old $\alpha = 3$ value with value at node(C)=2. That is $\alpha = max(3,2) = 3$. Thus, there is no change in α value as well.

Thus finally, $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value at node A is$[\alpha = 3, \beta = +\infty]$ and value of Node(A)=3. So, optimal value for the maximizer is 3 and there are 2 terminal nodes are pruned (H and I). The optimal search path is A→B→D (as shown in figure 22(d)).

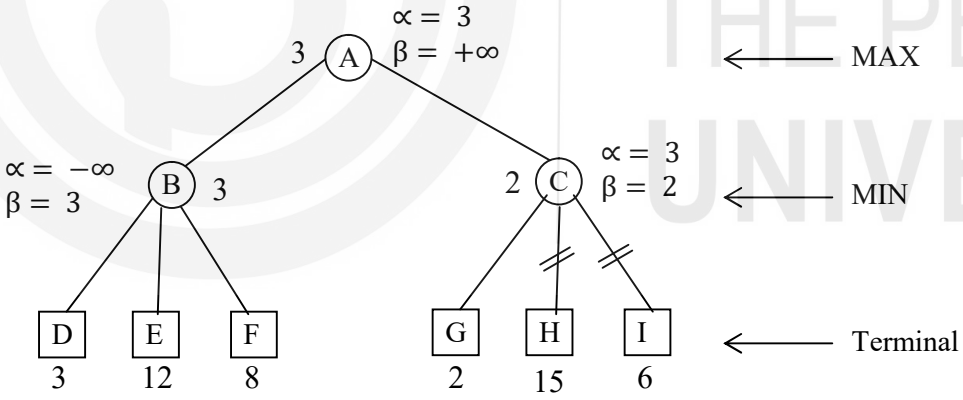


Fig 22(d)

2.7.2 Move ordering of Alpha-beta pruning

The effectiveness of Alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning. We have two types of move ordering:

Worst case ordering: In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree and works exactly as MiniMax algorithm. In this case, it consumes more time because of alpha-

beta factors, such a move of pruning is called a worst ordering. The time complexity for such an order is $O(b^m)$where b: Branching Factor and m is the depth of the tree.

Best (ideal) case ordering: The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best move occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. The time complexity for best case order is $O(b^{m/2})$ (since we search only left sub tree, not a right subtree).

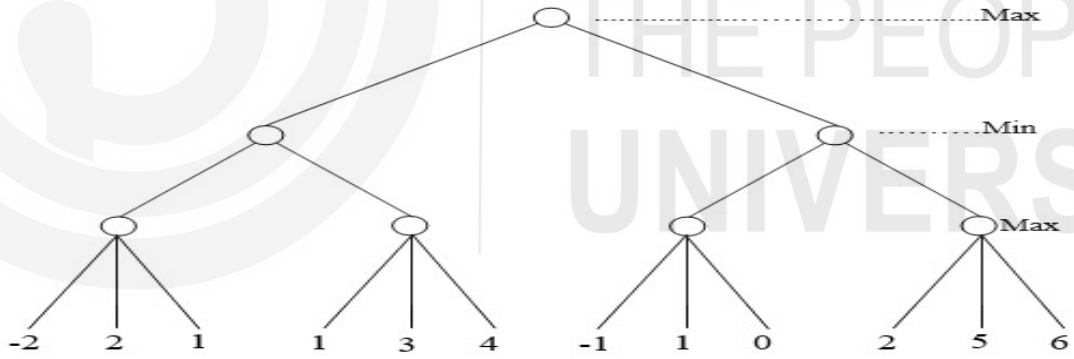
Note that pruning does not affect the final result. Good move ordering improves the effectiveness of pruning. With ideal case ordering, time complexity is $O(b^{\frac{m}{2}})$

In Alpha-beta pruning:
- α value can never decrease and β value can never increase. Search can be discontinued at anode if:
-It is a Max node and $\alpha \geq \beta$ it is beta cutoff
-It is a Min node and $\beta \leq \alpha$ it is a alpha cutoff.

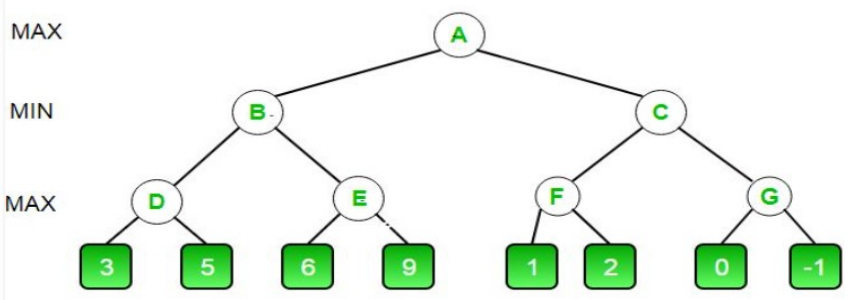
☞ **Check Your Progress 2**

Q.1: Compare the MINIMAX and Alpha-Beta Pruning algorithm with respect to Time complexity.

Q.2 Consider the following Minimax game tree search in which root is maximizing node and children are visited from left to right. Find the value of the root node of the game tree?



Q.3 Apply Alpha-Beta pruning algorithm on the following graph and find which node(s) are pruned?



Q.4: Consider the following Minimax game tree search (figure1) in which root is maximizing node and children are visited from left to right.

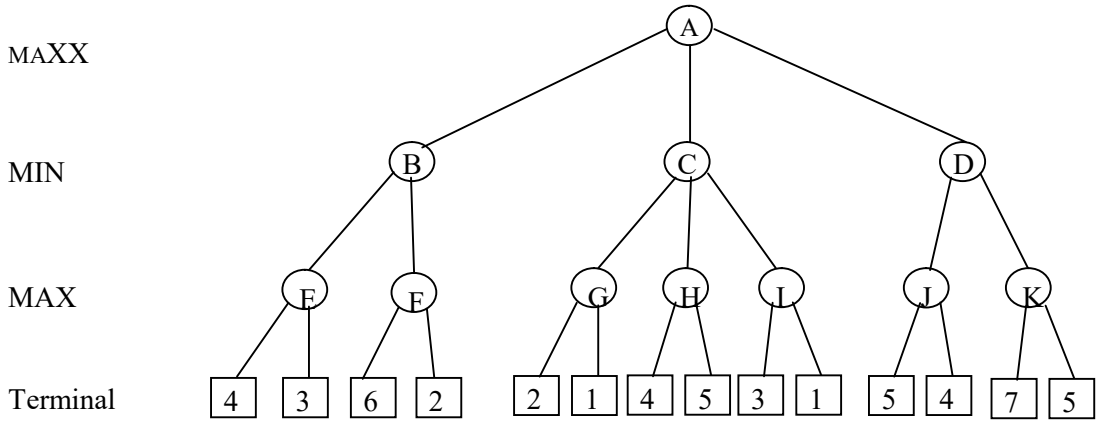
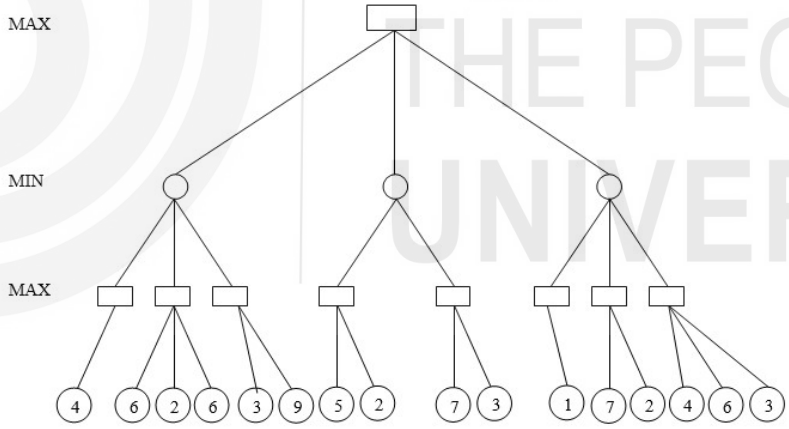


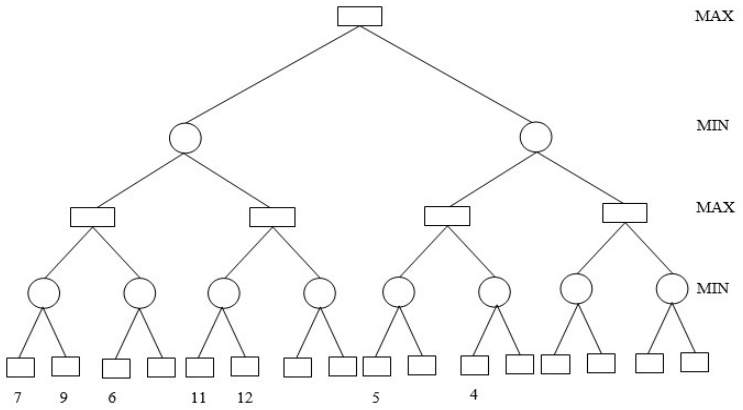
Figure1(a)

(a) Find the value of the root node of the game tree?
(b) Find all the nodes pruned in the tree?
(c) Find the optimal path for the maximizer in a tree?

Q.5: Consider the following Minimax game tree search in which root is maximizing node and children are visited from left to right. Find what will be the value propagated at the root?



Q.6: Consider the following Minimax game tree search in which root is maximizing node and children are visited from left to right. Find the value of the root node of the game tree?



Multiple choice Question

Q.7: Consider the following Minimax game tree search in which root is maximizing node and children are visited from left to right. Find the value of the root node of the game tree?

A. 14 B. 17 C. 111 D. 112

2.8 Summary

- Before an AI problem can be solved it must be represented as a **state space**. Among all possible states, there are two special states called ***initial state*** (the start point) and ***final state*** (the goal state).
- A ***successor function (a set of operators)***is used to change the state. It is used to move from one state to another.
- A state space is set of all possible states of a problem.
- A ***state space*** essentially consists of a set of nodes representing each state of the problem, arcs between nodes representing the legal moves from one state to another, an initial state, and a goal state. Each state space takes the form of a tree or a graph.
- The process of searching means a sequence of action that take you from an initial state to a goal state.

- search is fundamental to the problem-solving process. Search means the problem is solved by using the rules, in combination with an appropriate***control strategy***, to move through the problem space until a path from an ***initial state*** to a ***goal state*** is found.
- A **problem space** is represented by a directed graph, where *nodes* represent *search state*and *paths*represent the *operators* applied to change the state.
- In general, a state space is represented by 4 tuples as follows:$\boldsymbol{S_s}$: $[\boldsymbol{S}, \boldsymbol{s_0}, \boldsymbol{O}, \boldsymbol{G}]$**,** Where **S**: Set of all possible states (possibly infinite)**,** $\boldsymbol{s_0}$**:** start state (initial configuration) of the problem, $s_0 \in S$. **O**: Set of production rules (or set of state transition operator) used to change the state from one state to another. It is the set of arcs (or links) between nodes.
- Adversarial search is a game-playing technique where the agents are surrounded by a competitive environment. A conflicting goal is given to the agents (multiagent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search.
- In a **normal search**, we follow a sequence of actions to reach the goal or to finish the game optimally. But in an **adversarial search**, the result depends on the players which will decide the result of the game. It is also obvious that the solution for the goal state will be an optimal solution because the player will try to win the game with the shortest path and under limited time.
- There are 2 types of adversarial search: **Minimax Algorithm** and **Alpha-beta Pruning.**
- Minimax is a two-player (namely MAX and MIN) game strategy where *if one wins, the other lose the game*. This strategy simulates those games that we play in our day-to-day life. Like, if two persons are playing chess, the result will be in favour of one player and will go against the other one. **MIN:** Decrease the chances of **MAX** to win the game and **MAX:** Increases his chances of winning the game. They both play the game alternatively, i.e., turn by turn and following the above strategy, i.e., if one wins, the other will definitely lose it. Both players look at one another as competitors and will try to defeat one-another, giving their best.
- In minimax strategy, the result of the game or the utility value is generated by a **heuristic function** by propagating from the initial node to the root node. It follows the **backtracking technique** and backtracks to find the best choice. MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX's utility value.
- The drawback of minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths. This increases its time complexity.
- If b is the branching factor and d is the depth of the tree, then time complexity of MINIMAX algorithm is $O(b^d)$ that is exponential.
- Alpha-beta pruning is an advance version of MINIMAX algorithm. Therefore, alpha-beta pruning reduces the drawback of minimax strategy by less exploring the nodes of the search tree.

- The alpha-beta pruning method **cut-off the search** by exploring a smaller number of nodes. It makes the same moves as a minimax algorithm does, but it prunes the unwanted branches using the pruning technique.
- Alpha-beta pruning works on two threshold values, i.e., α **(alpha)** and β **(beta).$\boldsymbol{\alpha}$:** It is the best highest value; a **MAX** player can have. The initial value of $\boldsymbol{\alpha}$ is set to negative infinity value, that is $\boldsymbol{\alpha} = -\infty$. As the algorithm progress its value may change and finally get the best (highest) value**. $\boldsymbol{\beta}$:** It is the best lowest value; a **MIN** player can have. The initial value of $\boldsymbol{\beta}$ is set to positive infinity value, that is $\boldsymbol{\beta} = +\infty$. As the algorithm progress its value may change and finally get the best (lowest) value**.**
- So, each MAX node has $\boldsymbol{\alpha}$ value, which never decreases, and each MIN node has $\boldsymbol{\beta}$ value, which never increases. The main condition which required for alpha-beta pruning is $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$**,** that is if$\boldsymbol{\alpha} \geq \boldsymbol{\beta}$**,** then prune (cut) the branches otherwise search is continued.
- As we know there are two-parameter is defined for Alpha-beta pruning, namely alpha ($\boldsymbol{\alpha}$)and beta($\boldsymbol{\beta}$)**.** The initial value of alpha and beta is set to as $\boldsymbol{\alpha} = -\infty$ **and** $\boldsymbol{\beta} = +\infty$.As the algorithm progresses its values are changes accordingly. Note that in Alpha-beta pruning (cut), at any node in a tree, if $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$**,** then prune (cut) the next branch else search is continued.
- The effectiveness of Alpha-beta pruning is highly dependent on the order in which each node is examined.
- **Worst case ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree and works exactly as MiniMax algorithm. In this case, the time complexity is $O(b^m)$where b: Branching Factor and m is the depth of the tree.
- **Best (ideal) case ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best move occur at the left side of the tree. The time complexity for best case order is $O(b^{m/2})$ (since we search only left sub tree, not a right subtree).

2.9 Solutions/Answers

Check your progress 1:

Answer1: A number of factors need to be taken into consideration when developing a statespace representation. Factors that must be addressed are:

- What is the goal to be achieved?
- What are the legal moves or actions?
- What knowledge needs to be represented in the state description?

- Type of problem - There are basically three types of problems. Some problems only need a representation, e.g., crossword puzzles. Other problems require a yes or no response indicating whether a solution can be found or not. Finally, the last type problem are those that require a solution path as an output e.g., mathematical theorems Towers of Hanoi. In these cases we know the goal state and we need to know how to attain this state

- Best solution vs. Good enough solution - For some problems a good enough solutionis sufficient. For example: theorem proving eight squares. However, some problems require a best or optimal solution, e.g., the traveling salesman problem.

Answer 2

(a) Formulation of Missionaries and Cannibal problem:

State: (#M,#C,0/1)

Where #M represents Number of missionaries in the left side bank (i.e., left side of the river)

#C : represents the number of cannibals in the left side bank (i.e., left side of the river)

0/1 : indicate the boat position of the boat. 0 indicates the boat is on the left side of the river and 1 indicate the boat is on the right side.

Start state:(3,3,0)

Goal State: (0,0,1)

Operator: Sate will be changed by moving missionaries and (or) cannibals from one side to another using boat. So, it can be represented as number of persons on the either side of the river. Note that the boat can carries maximum 2 persons.

Boat carries: (1,0) or (0,1) or (1,1) or (2,0) or (0,2).Here in (i,j), i represents number of missionaries and j means number of cannibals.

(b) Solution of Missionaries and Cannibal problem:

Start state:(3,3,0)

Goal State: (0,0,1)

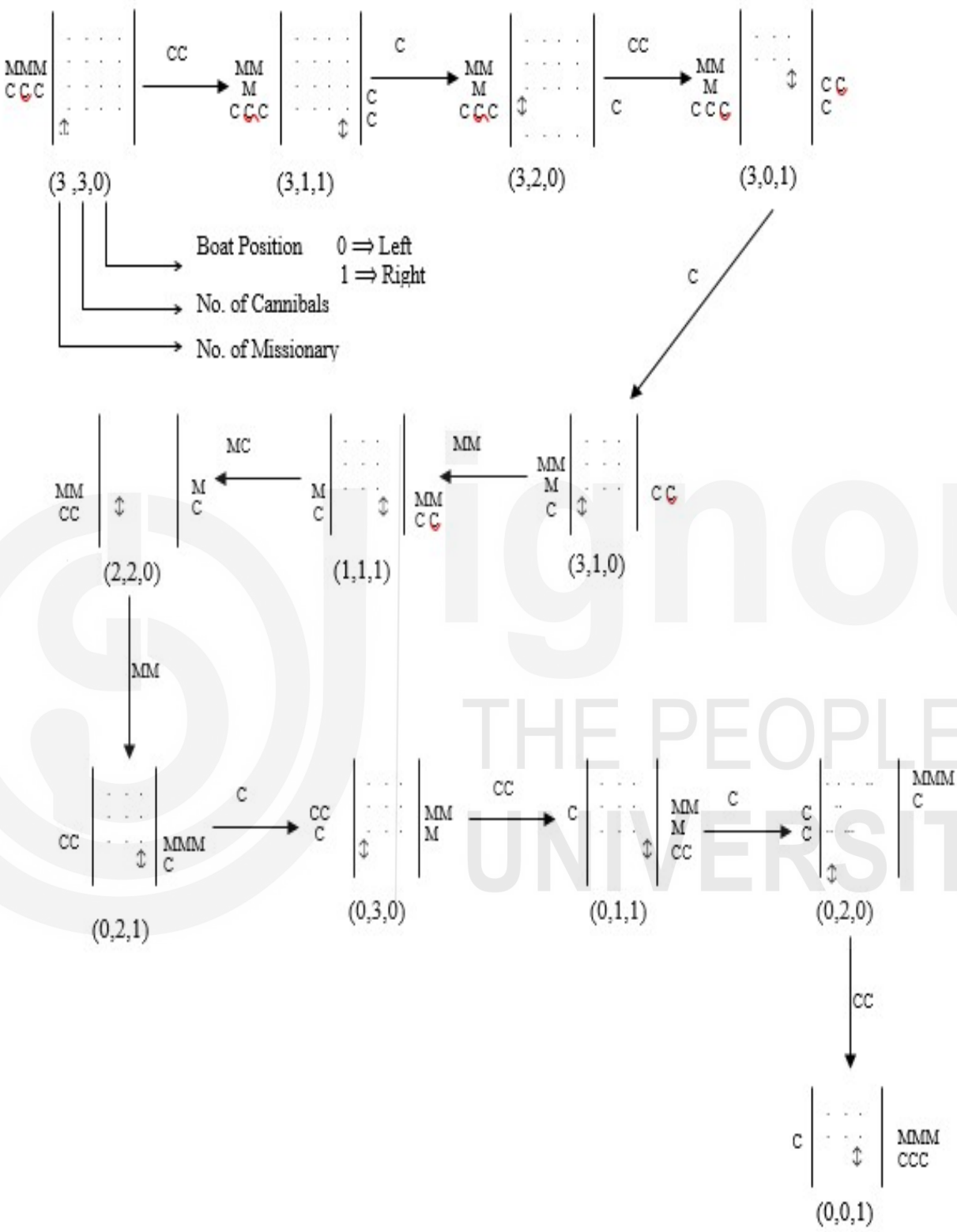


Figure: Solution of Missionaries and Cannibal problem

A state space tree for this problem is shown below:

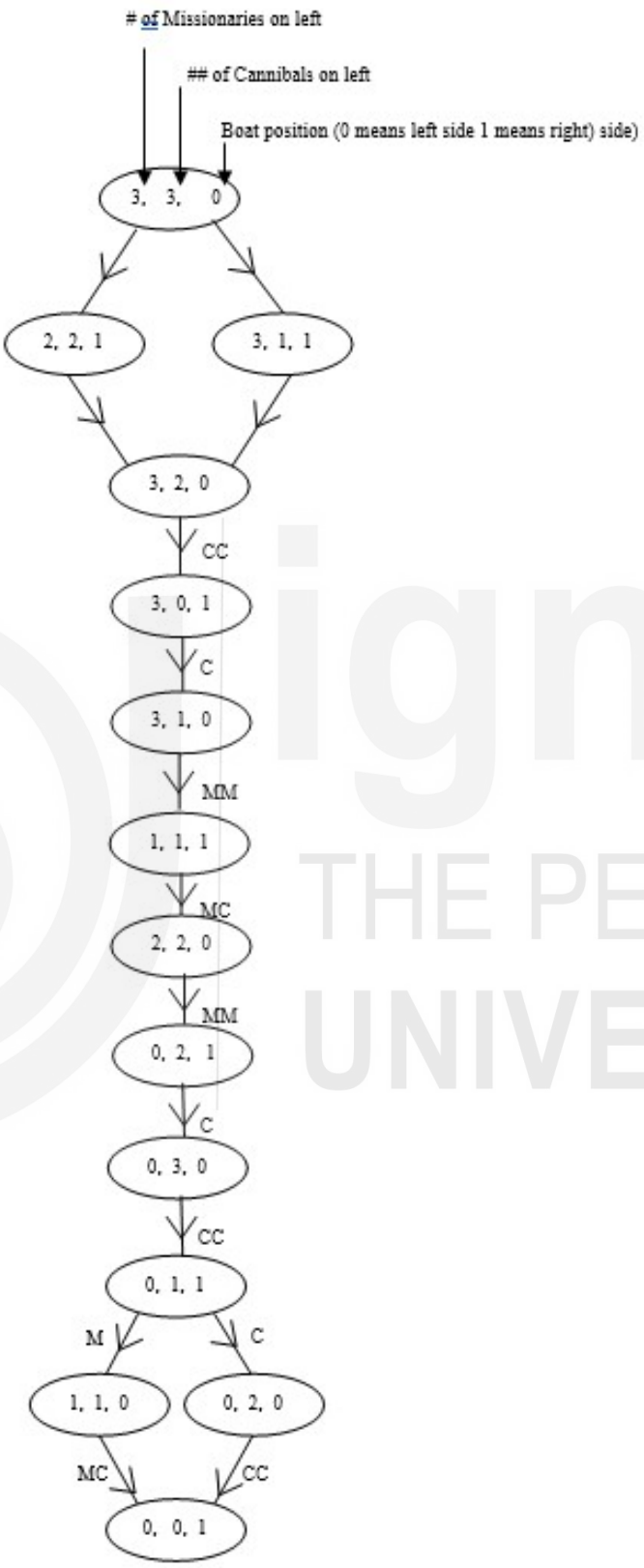


Figure: A state space tree showing all possible solution of missionaries and cannibal problem.

Answer 3: Towers Hanoi A possible state space representation of the Towers Hanoi problem using a graph is indicated in **Figure 1.**

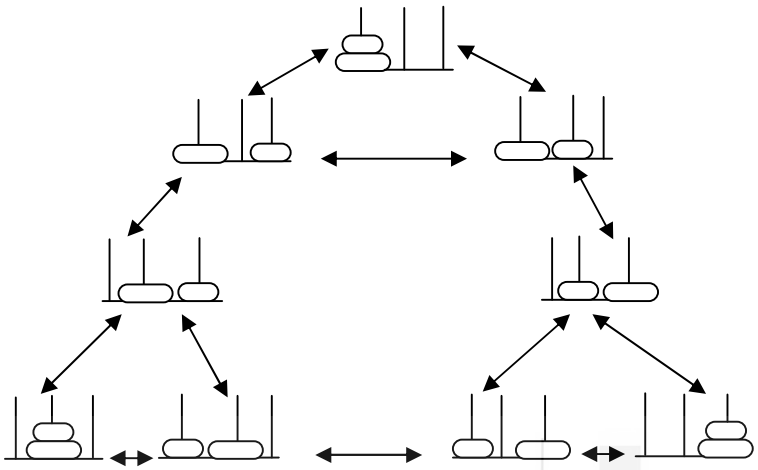
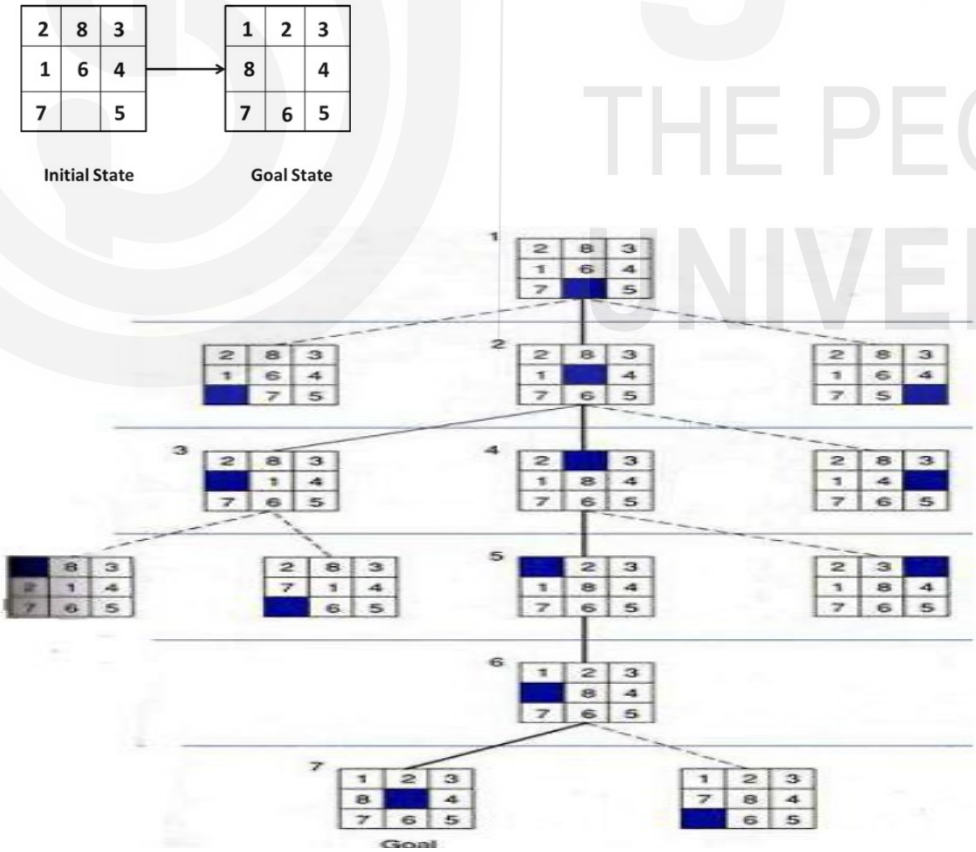


Figure 1: Towers of Hanoi slate space representation for n=2

The legal moves in this state space involve moving one ring from one pole to another, moving one ring at a time, and ensuring that a larger ring is not placed on a smaller ring.

Answer 4:



Minimum Cost path for solution= 6

Answer 5: Backtracking Algorithm: The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false.

1) Start in the leftmost column

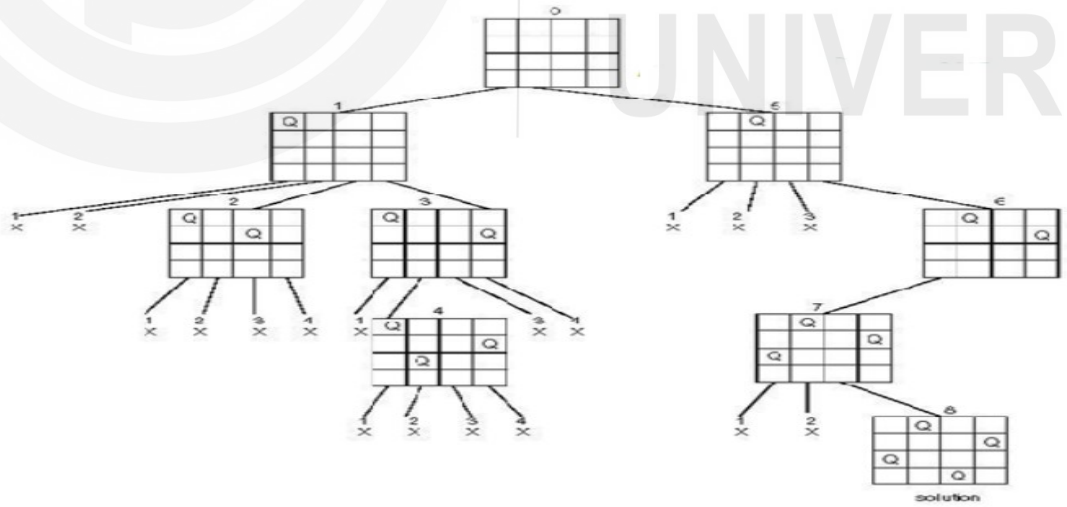
2) If all queens are placed
return true

3) Try all rows in the current column.
Do following for every tried row.

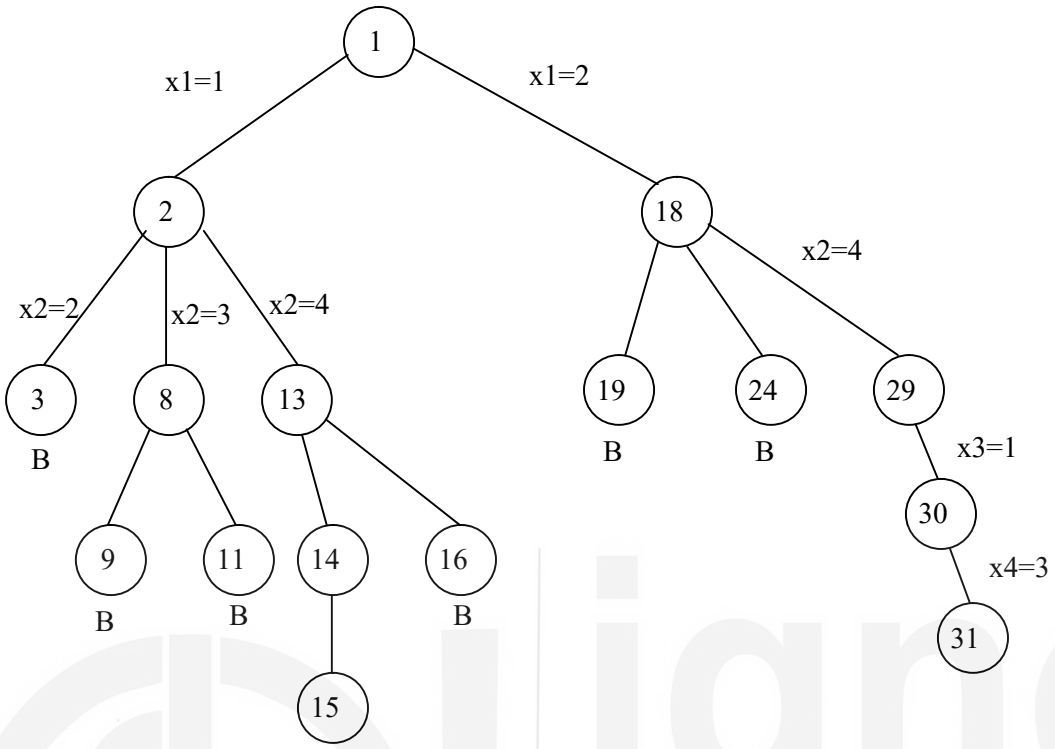
a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
b) If placing the queen in [row, column] leads toa solution then return true.
c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
4) If all rows have been tried and nothing worked, return false to trigger backtracking.

State space tree for 4 Queen's problem

A state-space tee (SST) can be constructed to show the solution to this problem.The following SST (figure 1) shows one possible solution$\{x1, x2, x3, x4\} = \{2,4,3,1\}$for the 4 Queen's Problem.



Or we can also denote the State space tree as follows:



B denotes the Dead Node (nonpromising node). The figure 1 shows the Implicit tree for 4 queen problem for solution <2,4,1,3>. The Root represents an initial state. The Nodes reflect the specific choices made for the components of a solution. Explore The state space tree using **depth-first search.** "Prune" non-promising nodesdfs stops exploring subtree rooted at nodes leading to no solutions and then backtracks to its parent node

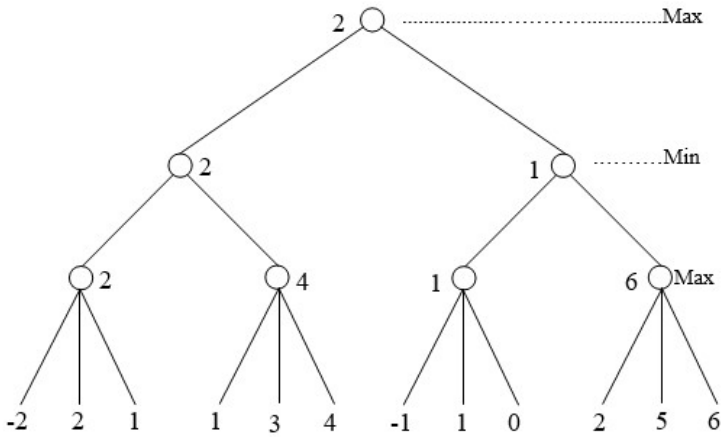
Check your progress 2

Answer1:

Solution: Alpha-beta pruning is an advance version of MINIMAX algorithm. The drawback of minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths. This increases its time complexity. If b is the branching factor and d is the depth of the tree, then time complexity of MINIMAX algorithm is $O(b^d)$ that is exponential.

Alpha-Beta pruning is a way of finding the optimal Minimax solution while avoiding searching subtrees of moves which won't be selected. The effectiveness of Alpha-beta pruning is highly dependent on the order in which each node is examined. **The** ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best move occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. The time complexity for best case order is $O(b^{m/2})$ (since we search only left sub tree, not a right subtree).

Answer 2: The final game tree with max and min value at each node is shown in the following figure.



Answer3:

Solution: Solve the question as sown in Example1

The initial call starts from A. We initially start the search by setting the initial value of $\boldsymbol{\alpha} = -\infty$and$\boldsymbol{\beta} = +\infty$for root node A. These values are passed down to subsequent nodes in the tree. At A the maximizer must choose max of B and C, so A calls B first. At B it the minimizer must choose min of D and E and hence calls D first. At D, it looks at its left child which is a leaf node. This node returns a value of 3. Now the value of alpha at D is max($-\infty$, 3) which is 3. To decide whether it's worth looking at its right node or not, it checks the condition $\alpha \geq \beta$. This is false since $\boldsymbol{\beta} = +\infty$ and ∞= 3. So, it continues the search.

D now looks at its right child which returns a value of 5. At D, alpha = max(3, 5) which is 5. Now the value of node D is 5. Value at node D=5, move up to node B(=5). Now at node B, β value will be modified as

$\beta = min(+\infty, 5) = 5.$

B now calls E, we pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node B to bottom node E as $[\boldsymbol{\alpha} = -\infty, \boldsymbol{\beta} = \mathbf{5}]$. **Since,** node E is at MAX level, so only $\boldsymbol{\alpha}$value will be change. Now, at E, it first checks the left child (which is a terminal node) with value 6. This node returns a value of 6.

Now, the value of $\boldsymbol{\alpha}$ **at** node E is calculated as $\alpha = max(-\infty, 6) = 6$, so value of Node(E)=6 and modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$ at node E is $[\alpha = 6, \beta = 5]$. To decide whether it's worth looking at its right node or not, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer is **YES**, since $6 \geq 5$. So, we prune (cut) the right branch of E, as shown in figure (a).

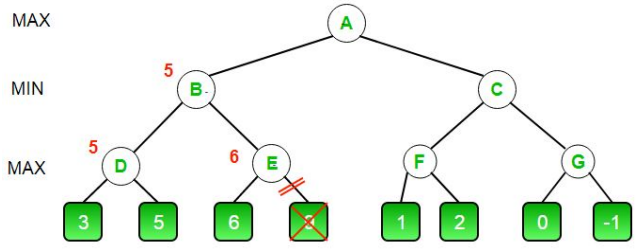


Figure (a) game tree after applying alpha-beta pruning on left side of node A

Similarly, we solve for right sub tree for Node A [refer the example 1 and solve for right sub tree part]. The final tree with node value at every node is shown in the figure(b).

Thus finally, $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value at node A is$[\alpha = 5, \beta = +\infty]$ and best value at Node(A)=max (5,2)=5. So, optimal value for the maximizer is 5 and there are 3 terminal nodes are pruned (9, 0 and -1). The optimal search path is A→B→D→5 , as shown in figure (b).

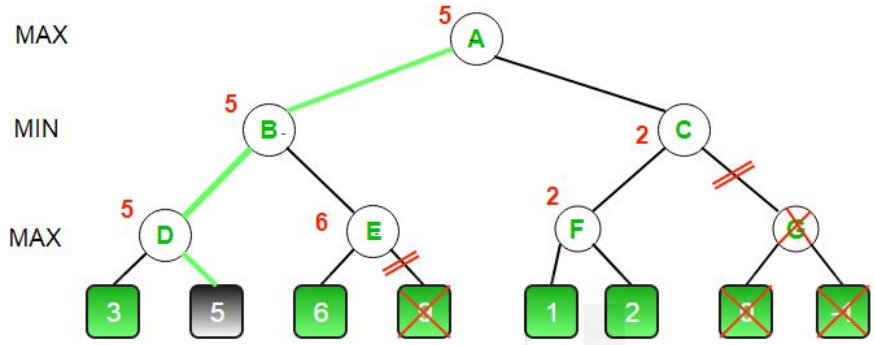
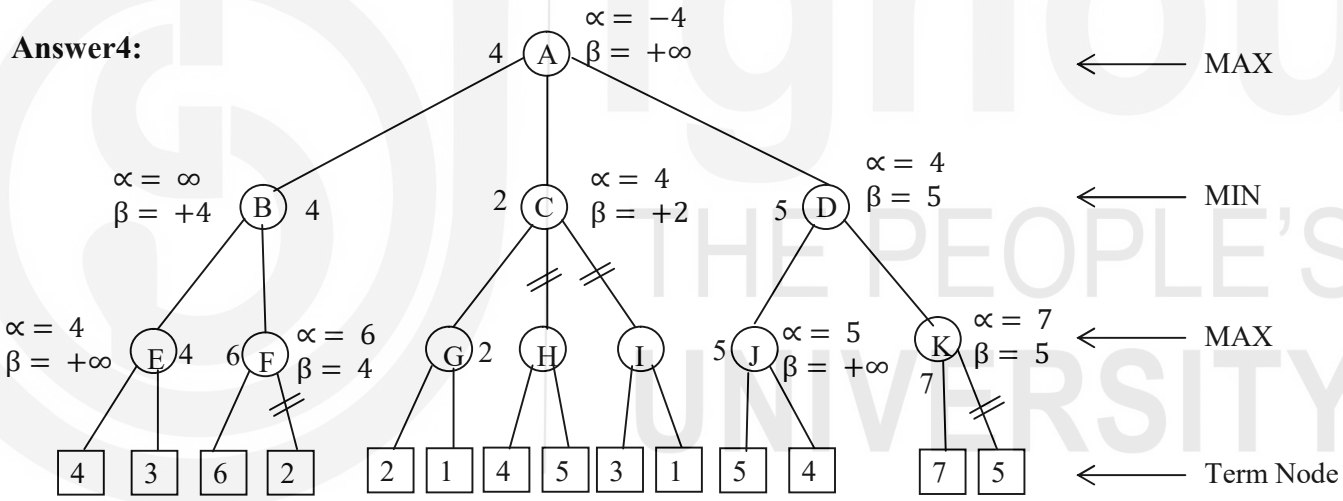


Figure (b) Final tree with node value on every node with prune branches

Answer4:



Answer5: 5

Answer6: 7

Answer 7: Option (B) 17

2.11 FURTHER READINGS

1. Ela Kumar, " Artificial Intelligence", IK International Publications
2. E. Rich and K. Knight, "Artificial intelligence", Tata Mc Graw Hill Publications
3. N.J. Nilsson, "Principles of AI", Narosa Publ. House Publications
4. John J. Craig, "Introduction to Robotics", Addison Wesley publication
5. D.W. Patterson, "Introduction to AI and Expert Systems" Pearson publication

UNIT 3 UNINFORMED & INFORMED SEARCH

Structure **Page No**

3.0 INTRODUCTION

Before an AI problem can be solved it must be represented as a **state space**. In AI, a wide range of problems can be formulated as search problem. The process of searching means a sequence of action that take you from an initial state to a goal state as sown in the following figure 1.

Fig 1: A sequence of action in a search space from initial to goal state.

In the unit 2 we have already examined the concept of a **state space** and adversarial (game playing) search strategy. In many applications there might be multiple agents or persons searching for solutions in the same solution space. In adversarial search, we need a path to take action towards the winning direction and for finding a path we need different type of search algorithms.

Search algorithms are one of the most important areas of Artificial Intelligence. This unit will explain all about the search algorithms in AI which explore the search space to find a solution.

In Artificial Intelligence, Search techniques are **universal problem-solving methods**. Rational agents or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem-solving agents are the goal-based agents and use atomic representation.

One **disadvantage** of state space representation is that it is not possible to visualize all states for a given problem. Also, the resources of the computer system are limited to handle huge state space representation. But many problems in AI take the form of state-space search.

Many problems in Al take the form of ***state-space search***.

- The ***states*** might be legal board configurations in a game, towns and cities in some sort of route map, collections of mathematical propositions, etc.
- The state-space is the configuration of the possible states and how they connect to each other e.g., the legal moves between states.
- When we don't have an algorithm which tells us definitively how to negotiate the state-space we need to search the state-space to find an optimal path from a start slate to a goal state,

We can only decide what to do (or where to go), by considering the possible moves from the current state, and trying to look ahead as far as possible. Chess, for example is a very difficult state space search problem.

Searching is the process looking for the solution of a problem through a set of possibilities (state

space).In general, the searching process starts from the initial state (root node) and proceeds by performing the following steps:

- Check whether the current state is the goal state or not?
- Expand the current state to generate the new sets of states.
- Choose one of the new states generated for search depending upon search strategy (for example BFS, DFS etc.).
- Repeat step 1 to 3 until the goal state is reached or there are no more state tube expanded.

Evaluation (properties) of search strategies : A search strategy is characterized by the sequence in which nodes are expanded. Any search algorithms are commonly evaluated according to the following four criteria. The following four essential properties of search algorithms is used to compare the efficiency of any search algorithms.

Completeness: A search algorithm is said to be complete if it guarantees to return a solution, if exist.

Optimality/Admissibility: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

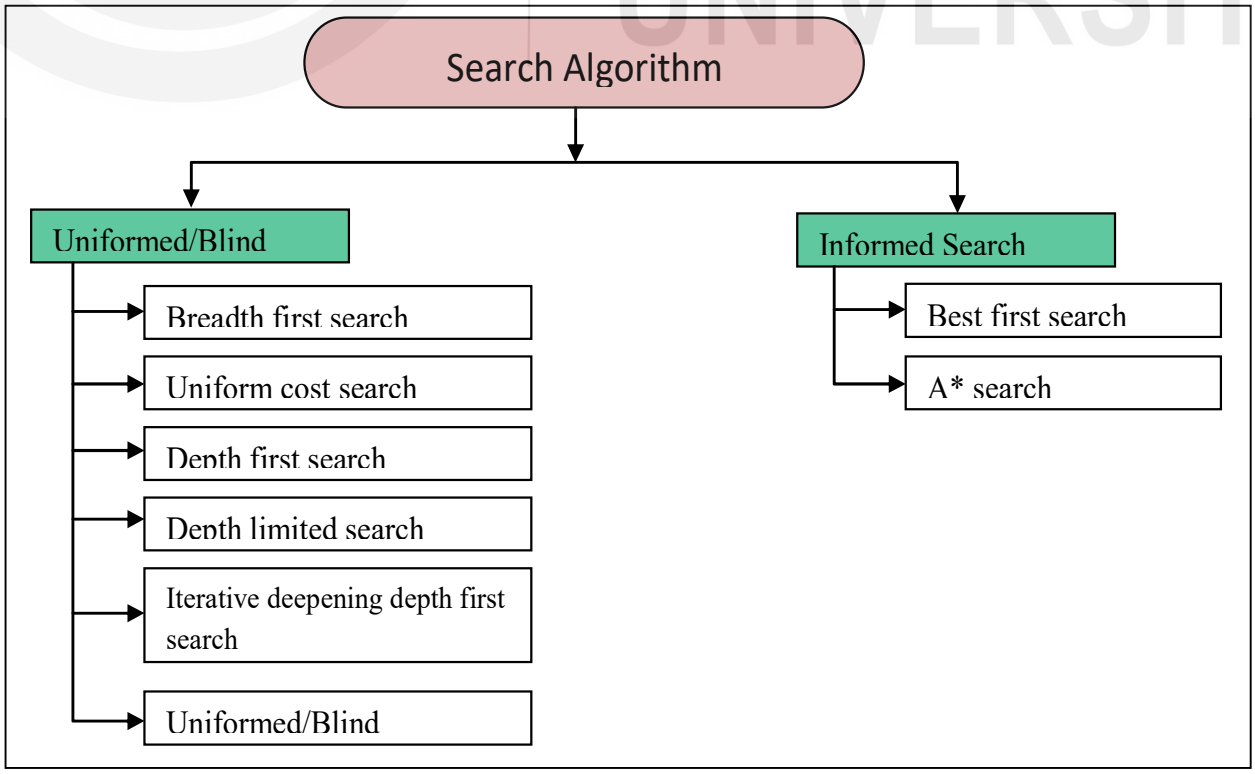
Time Complexity: Time complexity is a measure of time for an algorithm to complete its task. Usually measured in terms of the number of nodes expended during the search.

Space Complexity: It is the maximum storage space required at any point during the search. Usually measured in terms of the maximum number of nodes in memory at a time.

Time and space complexity are measured in terms of:

- *b* - max branching factor of the search tree
- *d* - depth of the least-cost solution
- *m* - max depth of the search tree (may be infinity)

In all search algorithms, the order in which nodes are expended distinguishes them from one another. There are two broad classes of search methods:



Uninformed search is also called Brute force search or Blind search or Exhaustive search. It is called blind search because of the way in which search tree is searched without using any information about the search space. It is called Brute force because it assumes no additional knowledge other than how to traverse the search tree and how to identify the leaf nodes and goal nodes. This search ultimately examines every node in the tree until it finds a goal.

Informed search is also called as Heuristic (or guided) search. These are the search techniques where additional information about the problem is provided in order to guide the search in a specific direction. A heuristic is a method that might not always find the best solution but is guaranteed to find a good solution in reasonable time. By sacrificing completeness, it increases efficiency.

The following table summarizes the differences between uninformed and informed search:

Uninformed Search	**Informed Search**
No information about the path, cost, from the current state to the goal state. It doesn't use domain specific knowledge for searching process.	The path cost from current state to goal state is calculated, to select the minimum cost path as the next state. It uses domain specific knowledge for the searching process.
It finds solution slow as compared to informed	It finds solution more quickly.
Less efficient	More efficient
Cost is high.	Cost is low
No suggestion is given regarding the solution init. Problem to be solved with the given information only.	It provides the direction regarding the solution. Additional information can be added as assumption to solve the problem.
Examples are: ▪ Depth First Search, ▪ Breadth First Search, ▪ Depth limited search, ▪ Iterative Deepening DFS, ▪ Bi-directional search	Examples are: ▪ Best first search ▪ Greedy search ▪ A* search

3.1 OBJECTIVES

After studying this unit, you should be able to:

- Differentiate the Uninformed and informed search algorithm
- Formulate the search problem in the form of state space
- Explain the differences between various uninformed search approaches such as BFS, DFS, IDDFS, Bi-directional search.
- Evaluate the various Uninformed search algorithm with respect to Time, space and Optimality/Admissibility criteria.
- Explain Informed search such as Best-First search and A* algorithm.

- Differentiate between advantages and disadvantages of heuristic search: A* and AO* algorithm
- Differentiate between memory bound search: Iterative Deepening A* and Recursive Best-First Search.

3.2 Formulating search in state space

A state space is a graph, (V, E) where V is a set of nodes and E is a set of arcs, where each arc is directed from one node to another node.

- **V: a node is a data structure that contains** state description, plus, optionally other information related to the parent of the node, operation to generate the node from that parent, and other bookkeeping data.
- **E: Each arc** corresponds to an applicable action/operation. The source and destination nodes are called as parent (**immediate predecessor**) and child **(immediate successor**) nodes with respect to each other. Ancestors(also called predecessors) and descendants (also called successors) node. Each arc has a fixed, non-negative cost associated with it, corresponding to the cost of the action.

Each node has a set of successor nodes. Corresponding to all operators (actions) that can apply at source node's state. Expanding a node is generating successor nodes and adding them (and associated arcs) to the state-space graph. One or more nodes may be designated as start nodes.

A **goal** test predicate is applied to a node to determine if its associated state is a goal state. A **solution** is a sequence of operations that is associated with a path in a state space from a start node to a goal node. The**cost of a solution** is the sum of the arc costs on the solution path.

State-space search is the process of searching through a state space for a solution by making explicit a sufficient portion of an implicit state-space graph to include a goal node.

Hence, initially V={S}, where S is the start node; when S is expanded, its successors are generated, and those nodes are added to V and the associated arcs are added to E. This process continues until a goal node is generated (included in V) and identified (by goal test).

To implement any ***Uninformed search*** algorithm, we always initialize and maintain a list called **OPEN** and put start node of G in OPEN. If after some time, we find **OPEN** is empty and we are not getting "goal node", then terminate with failure. We select a node $\boldsymbol{n}$ from OPEN and if $\boldsymbol{n \in Goal\ node}$, then terminate with success, ***else*** we generate the successor of n (using operator O) and insert them in OPEN. In this way we repeat the process till search is successful or unsuccessful.

Search strategies differ mainly on how to select an OPEN node for expansion at each step of search.

A general search algorithm
1. Initialize: Set = $\{s\}$, where s is a start state.
2.Fail: If $OPEN = \{\ \}$, terminate with failure**.**
3. Select: Select a state, n, from OPEN
4.Terminate: If $n \in Goal\ node$, terminate with success
5. Expand: Generate the successor of n using operator O and insert them in OPEN.
6. LOOP: Goto Step 2

But the problem with the above search algorithm, it is not mentioned that when a node is already visited, then do not revisit that node again. That is "***how we can maintain a part of the state space that is already visited***". So, we have an extension of the same algorithm, where we can save the explicit state space. To save the explicit space, we maintained another list called **CLOSED.**

Thus, to implement any Uninformed search algorithm efficiently, two list **OPEN** and **CLOSED** are used.

Now we can select a node from OPEN and save it in CLOSED. The CLOSED list keeps record of nodes that are Opened. The major difference of this algorithm with the previous algorithm is that when we generate successor node from CLOSED, we check whether it is already in $(OPEN \cup CLOSED)$. If it is already in $(OPEN \cup CLOSED)$, we will not insert in OPEN again, otherwise insert. The following modified algorithm is used to save the explicit space using the list CLOSED.

Modified search algorithm to saving the explicit space

1. Initialize: Set $OPEN = \{s\}, CLOSED = \{\ \}$.
2.Fail: If $OPEN = \{\ \}$, terminate with failure**.**
3. Select: Select a state, n, from OPEN and save n in CLOSED
4.Terminate: If $n \in Goal\ node$, terminate with success
5. Expand: Generate the successor of n using operator O
For each successor, m, insert m in OPEN,only if $m \notin (OPEN \cup CLOSED)$
6. LOOP: Goto Step 2.

Here the OPEN and CLOSED list are used as follows:

OPEN: Nodes are yet to be visited.

CLOSED: Keeps track of all the nodes visited already

Note that initially OPEN list initializes with start state of G (e.g.,$\boldsymbol{OPEN = \{s\}}$**)** and CLOSED list as empty (e.g.,$\boldsymbol{CLOSED = \{\}}$**).**

Insertion or removal of any node in OPEN depends on specific search strategy.

3.2.1 Evaluation of Search Algorithms:

In any search algorithm, we select a node and generate its successor. Search strategies differ mainly on how to select an **OPEN** node for expansion at each step of search. Also, Insertion or deletion of any node from **OPEN** list depends on specific search strategy. Any search algorithms are commonly evaluated according to the following 4 criteria:It is the measure to evaluate the performance of the search algorithms:

- **Completeness:** Guarantees finding a solution whenever one exists.
- **Time Complexity:** How long (worst or average case) does it take to find a solution? Usually measured in terms of the number of nodes expanded.
- **Space Complexity:** How much space is used by the algorithm? Usually measured in terms of the maximum size that the "OPEN" list becomes during the search. The Time and Space complexity are measured in terms of: The branching factor or maximum number of successors of any node and **d**: the depth of shallowest goal node (depth of the least cost solution) and **m:** The maximum depth (length) of any path in the state space (may be infinite).
- **Optimality/Admissibility:** If a solution is found, is it guaranteed to be an optimal one? For example, is it the one with minimum cost?

Search process constructs a search tree, where **root** is the initial state S, and **leaf nodes** are nodesnot yet been expanded (i.e., they are in OPEN List) or having no successors (i.e., they're dead ends"). Search tree may be infinite because of loops even if statespace is small Search strategies mainly differ on select OPEN**.** Each node represents a partial solution path and cost of the partial solution path) from the start node to the given node. In general, from this node there are many possible paths (and therefore solutions that have this partial path as a prefix.

All search algorithms are distinguished by the order in which nodes are expended. There are two broad classes of search methods: Uninformed search and Heuristic Search. Let us first discuss the Uninformed search.

3.3 UNINFORMED SEARCH

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which searchtree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node. Sometimes we may not get much relevant information to solve a problem.

For Example, suppose we lost our car key, and we are not able to recall where we left, we have to search for the key with some information such as in which places, we used to place it. It may be our pant pocket or may be the table drawer. If it is not there, then we must search the whole house to get it. The best solution would be to search in the places from the table to the wardrobe. Here we need to search blindly with less clue. This type of search is called uninformed search or blind search.

Based on the order in which nodes are expended, we have the following types of uninformed search algorithms:

- Breadth-first search
- Depth-first search
- Uniform cost search
- Iterative deepening depth-first search
- Bidirectional Search

3.3.1 Breadth-first search (BFS):

It is the simplest form of blind search. In this technique the root node is expanded first, then all its successors are expanded and then their successors and so on. **In general, in BFS, all nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.** It means that all immediate children of nodes are explored before any of the children's children are considered. The search tree generated by BFS is shown below in Fig 2.

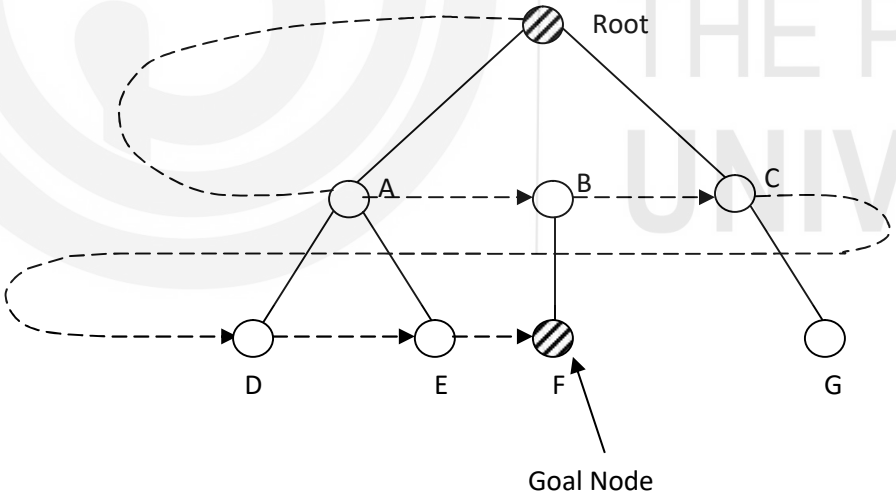


Fig 2 Search tree for BFS

Note that BFS is a brute-search, so it generates all the nodes tor identifying the goal and note that we are using the convention that the alternatives are tried in the left-to-right order.

A BFS algorithm uses a data structure-**queue** that works on FIFO principle. This queue will hold all generated but still unexplored nodes. **Please remember that the order in which nodes are placed on the queue or removal and exploration determines the type of search.**

We can implement it by using two lists called OPEN and CLOSED. The OPEN list contains those states that are to be expanded and CLOSED list keeps track of state already expanded. Here OPEN list is used as a **queue.**

BFS is effective when the search tree has a low branching factor.

Breath-First Search (BFS) Algorithm
1. Initialize: Set = $\{s\}$, where s is a start state.
2.Fail: If $OPEN = \{\ \}$, terminate with failure**.**
3. Select: Remove a left most state (say $\boldsymbol{a}$) from OPEN.
4.Terminate: If $a \in Goal\ node$, terminate with success, else
5. Expend: Generate the successor of node $\boldsymbol{a}$, discard the successors of $\boldsymbol{a}$ if it's already in OPEN, Insert only remaining successors on right end of OPEN [i.e., QUEUE]
6. LOOP: Goto Step 2

Let us take an example to see how this algorithm works.

Example1: Consider the following graph in fig-1 and its corresponding state space tree representation in fig-2. Note that A is a start state and G is a Goal state.

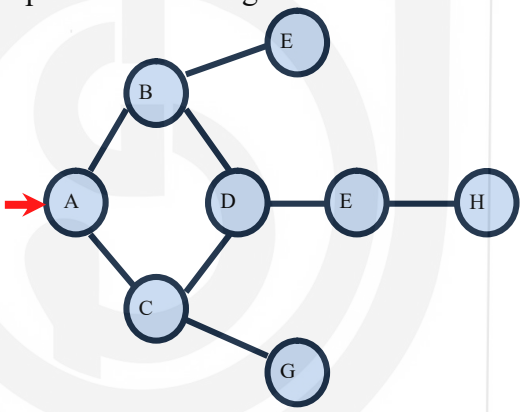


Fig-1 State space graph

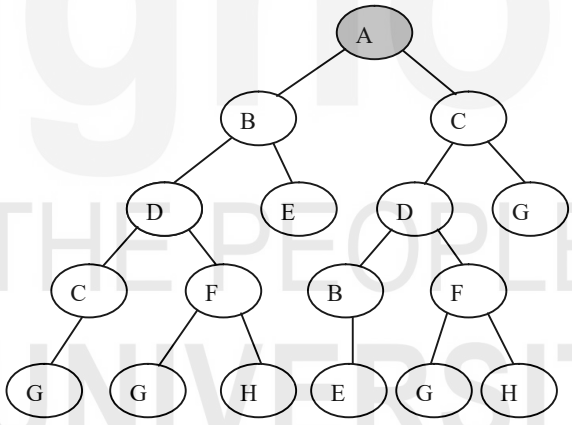
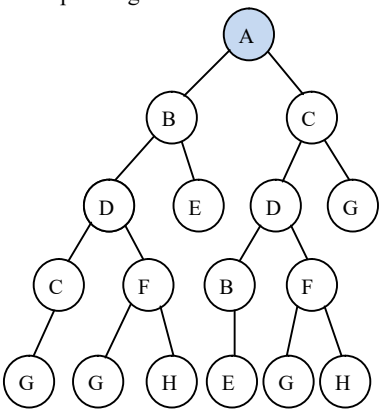


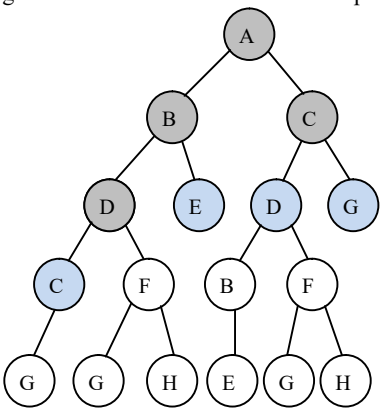
Fig-2 State space tree

Step1: Initially open contains only one node corresponding to the source state A.



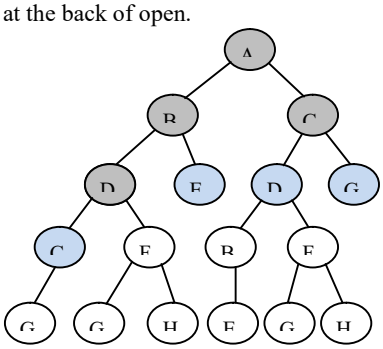
Open= A CLOSED={ }

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



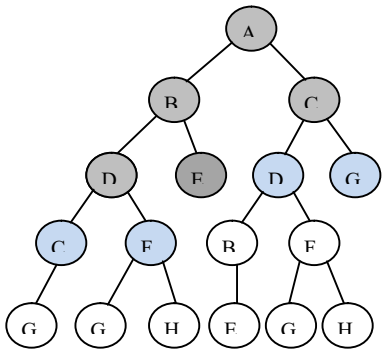
Open= E,D,G,C,F CLOSED=A,B,C,D

Step2: A is removed from open. The node is expended, and its children B and C are generated. They are placed at the back of open.



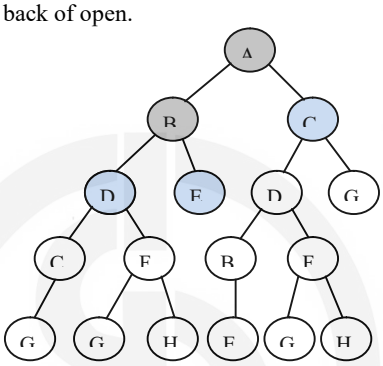
Open= B,C CLOSED={A }

Step 6: Node E is removed from open. It has no children



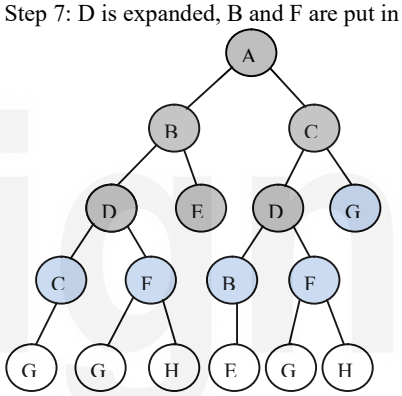
Open= D,G,C,F CLOSED=A,B,C,D,E

Step: 3: Node B is removed from open and is expended. Its children D, E are generated and put at the back of open.



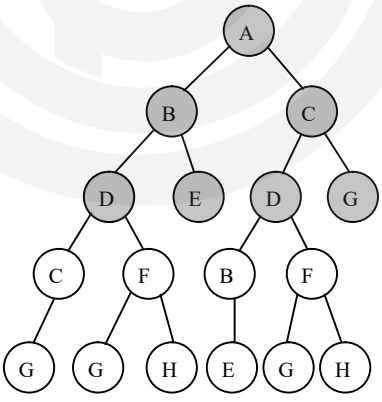
Open= C,D,E
CLOSED={A,B}

Step 7: D is expanded, B and F are put in OPEN.



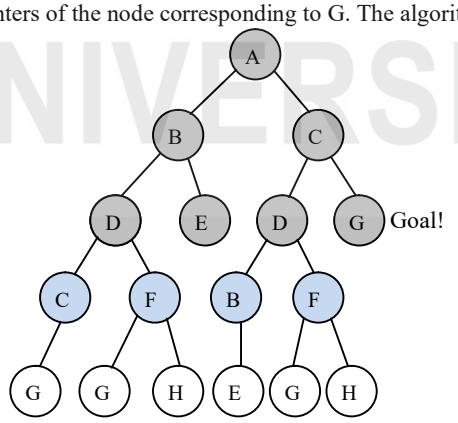
Open= G,C,F,B,F
CLOSED=A,B,C,D,E,D

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



Open=D,E,D,G
CLOSED={A,B,C}

Step 8: G is selected for expansion. It is found to be a goal node. So, the algorithm returns the path ACG by following the parent pointers of the node corresponding to G. The algorithm terminates.



Open=C,F,B,F
CLOSED={A,B,C,D,E.G}

3.3.2 Time and Space complexity of BFS:

Consider a complete search tree of depth d where each non-leaf node has b children (i.e., branching factor), has a total of

$1 + b + b^2 + b^3 + \cdots + b^d = \frac{1.(b^{d+1}-1)}{b-1}$nodes.

Time complexity is the number of nodes generated, so time complexity of BFS algorithm is $O(b^d)$

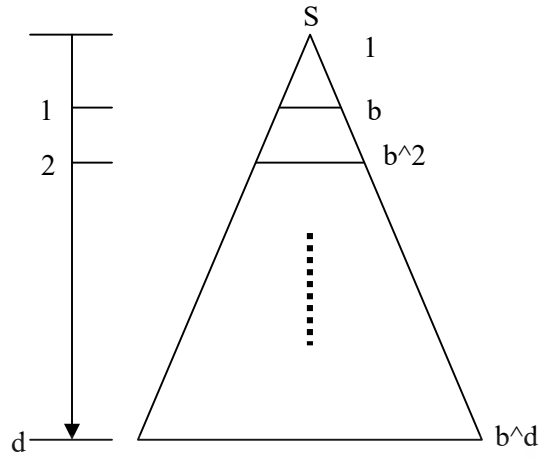


Fig 3 Search tree with b branching factor

For Example, consider a complete search tree of depth 12, where every node at depths $0,1,\ldots,11$ has 10 children (branching factor b=10) and every node at depth 12 has 0 children, then there are

$$1 + 10 + 10^2 + 10^3 + \cdots + 10^{12} = \frac{1.(10^{13} - 1)}{10 - 1}$$

$= \frac{10^{13}-1}{9} = O(10^{12})$ nodes in the complete search tree.

• BFS is suitable for problems with shallow solutions

Space complexity:

BFS has to remember each and every node it has generated. Space complexity (maximum length of OPEN list):

So, space complexity is given by:

$1+b+b^2+b^3+\ldots b^d = O(b^d)$.

Performance of BFS:

- **Time** Required for BFS for tree of b branching factor and d depth is $O(b^d)$.
- S**pace** (memory) requirement for a tree with b branching factor and d depth is also $O(b^d)$
- BFS algorithm is **Complete** (if b is finite).
- BFS algorithm is **Optimal** (if cost = 1 per step)

Space is the bigger problem in BFS as compared to DFS.

3.3.3 Advantages and disadvantages of BFS:

Advantages: BFS has some advantages and are given below

1. BFS will never get trapped exploring bund alley.

2. It is guaranteed to find a solution if one exists.

Disadvantages: BFS has certain disadvantages also. They are given below-

1. Time complexity and Space complexity are both $O(b^d)$ i.e., exponential type. This is very hurdle.

2. All nodes are to be generated in BFS. So, even unwanted nodes are to be remembered (stored in queue) which is of no practical use of the search.

3.3.4 Depth First Search

A Depth-First Search (DFS) explores a path all the way to a leaf before backtracking and exploring another path. That is expand deepest unexpanded node (expand most recently generated deepest node first).

The search tree generated by the DFS is show in figure below:

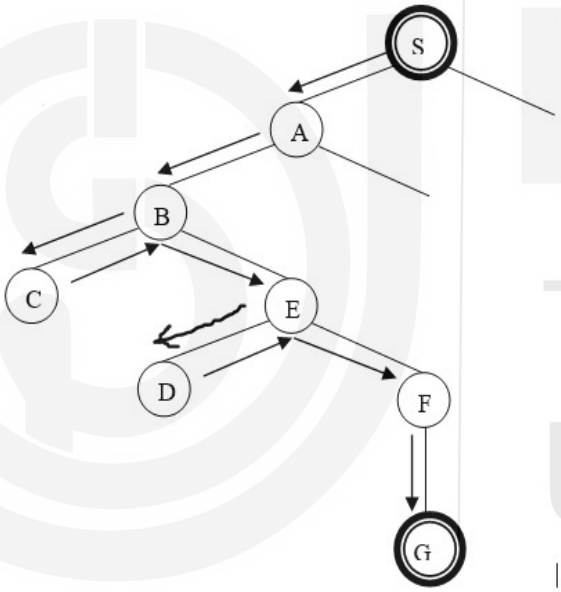


Fig 4 Depth first search (DFS) tree

In depth-first search we go as far down as possible into the search tree/graph before backing up and trying alternatives. It works by always generating a descendent of the most recently expanded node until some depth cut off is reached and then backtracks to next most recently expanded node and generates one of its descendants. So only path of nodes from the initial node to the current node is stored, in order to execute the algorithm. For example, consider the following tree and see how the nodes are expended using DFS algorithm.

Example1:

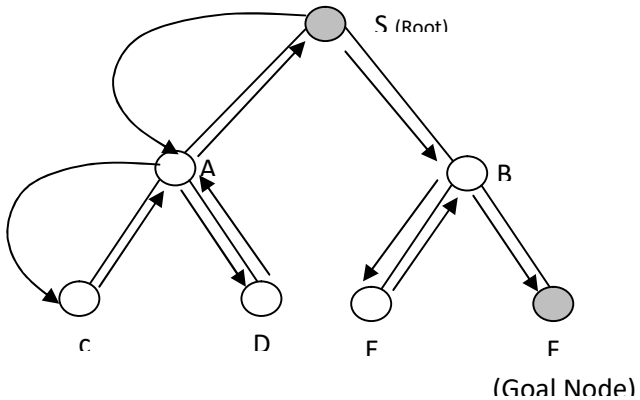


Fig. 5 Search tree for DFS

After searching root node S, then A and C, the search backtracks and tries another path from A. Nodes are explored in the order S, A, C, D, B, E, F.

Here again we use the list OPEN as a STACK to implement DFS. If we found that the first element of OPEN is the Goal state, then the search terminates successfully.

Depth-First Search (DFS) Algorithm

1. Initialize: Set = $\{s\}$, where s is a start state.

2.Fail: If $OPEN = \{\}$, terminate with failure**.**

3. Select: Remove a left most state (say $\boldsymbol{a}$) from OPEN.

4.Terminate: If $a \in Goal\ node$, terminate with success, else

5. Expend: Generate the successor of node $\boldsymbol{a}$, discard the successors of $\boldsymbol{a}$ if it's already in OPEN, Insert remaining successors on **left** end of OPEN [i.e., STACK]

6. LOOP: Goto Step 2

Note: The only difference between BFS and DFS is in **Expend** (step 5). In BFS, we always insert the generated successors at the right end of OPEN list, whereas in DFS at the left end of OPEN list.

Properties of DFS Algorithm:

Supposeb(branching factor), that is Maximum number of successors of any node and M: maximum depth of a leaf node, then

Number of nodes generated (in worst case): $1 + b + b^2 + \cdots + b^m = O(b^m)$

3.3.5 Performance of DFS:

- **Time** Required for DFS for tree of b branching factor and m depth (of shallowest goal node) is $O(b^m)$.
- **Space** (memory) requirement for a tree with b branching factor and m depth (of shallowest goal node) is also $O(bm)$
- BFS algorithm is **Complete** (if b is finite).
- BFS algorithm isnot**Optimal.**

Space is the advantage of DFS as compared to BFS.

3.3.6 Advantages and disadvantages of DFS:

Advantages:

- If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.
- The advantage of depth-first Search is that memory requirement is only linear with respect to the search graph. This is in contrast with breadth-first search which requires more space.

Example1: Consider the following graph in fig-1 and its corresponding state space tree representation in fig-2. Note that A is a start state and G is a Goal state.

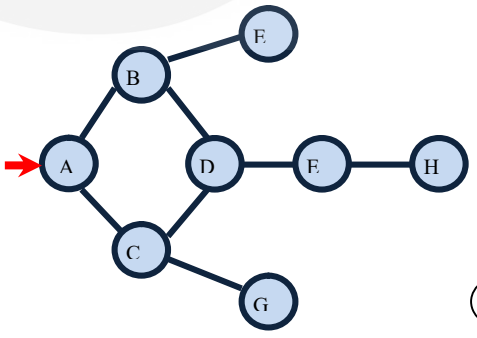


Fig-1 State space graph

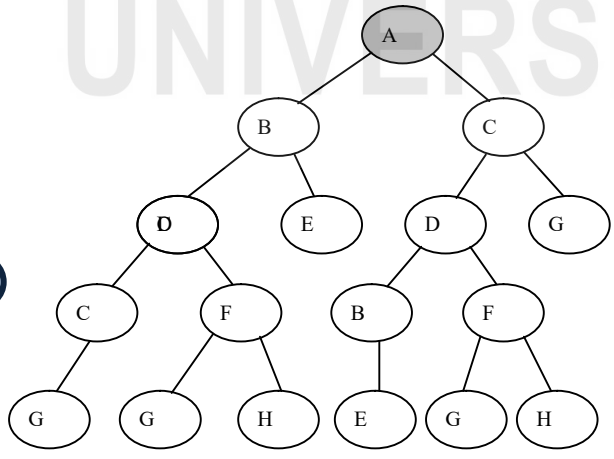
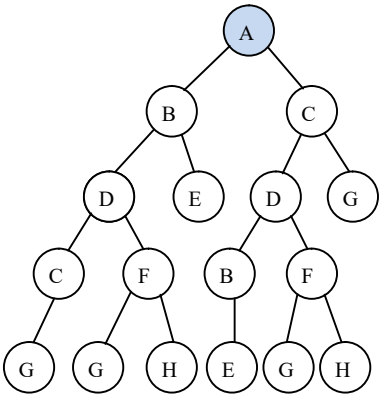


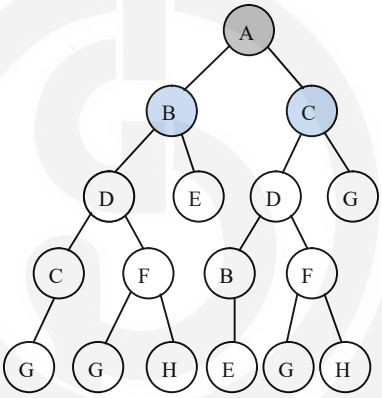
Fig-2 State space tree

Step 1: Initially open contains only one node corresponding to the source state A.



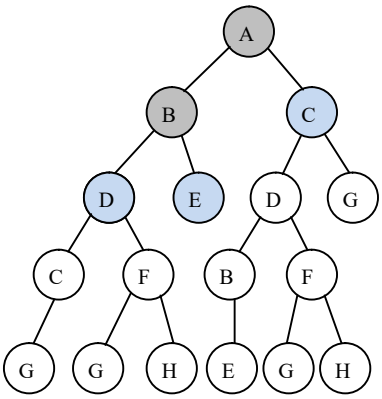
Open= A CLOSED={ }

Step 2: A is removed from open. The node A is expanded, and its children B and C are generated. They are placed at the from of open.



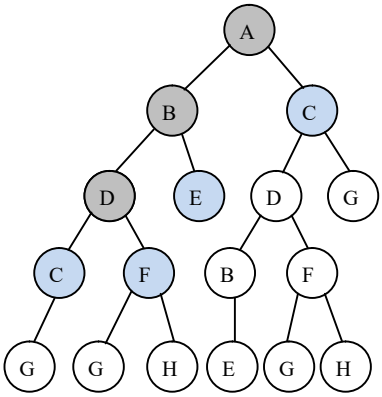
Open= B,C CLOSED={A}

Step 3: Node B is removed from open and is expended. Its children D, E are generated and put at the front open.



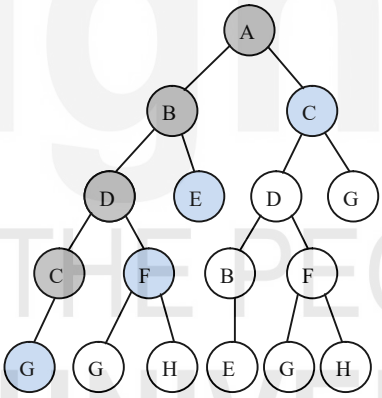
Open= D,E,C CLOSED={A,B}

Step 4: Node D is removed from open and is expanded. Its children C and F are added to the front of open.



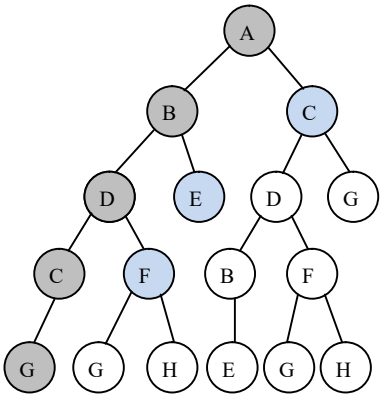
Open= C,F,E,C CLOSED=A,B,D

Step 5: Node C is removed from open. Its children G and F is added to the front of open.



Open= G,F,E,C CLOSED=A,B,D,C

Step 6: Node G is expanded and found to be a goal node. The solution path A-B-D-C-G is returned and the algorithm terminates.



Open= F,E,C CLOSED=A,B,D,C,G

3.3.7Comparison of BFS and DFS

BFS goes level wise , but requires more space as compared to DFS. The space required by DFS is O(d) where d is depth of tree, but space required by BFS is $O(b^d)$.

DFS: The problem with this appraoch is, if there is a node close to root, but not in first few subtrees explored bt DFS, then DFS reaches that node very late. Also, DFS may not find shortest path to a node (in terms of number of edges.)

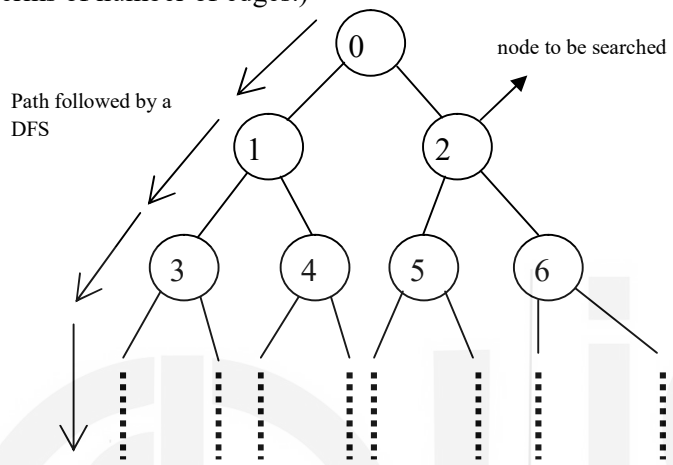


Fig 6: Path sequence in DFS

DFS: The problem with this approach is, if there is a node close to root, but not in first few subtrees explored by DFS, then DFS reaches that node very late. Also, DFS may not find shortest path to a node (in terms of number of edges).

Suppose, we want to find node- '2' of the given infinite undirected graph/tree. A DFS starting from node-0 will dive left, towards node 1 and so on.

Whereas, the node 2 is just adjacent to node 1.

Hense, a DFS wastes a lot of time in coming back to node 2.

An **Iteratie Deepenign Depth First Search** overcomes this andquickly find the required node.

3.4 Iterative Deepening Depth First Search (IDDFS)

Iterative Deepening Depth First Search (IDDFS) neither suffers the drawbacks of BFS nor DFS on trees. It takes the advantages of both the strategy.
It begins by performing DFS to a depth of zero, then depth of one, depth of two, and so on until a solution is found or some maximum depth is reached.
It is like BFS in that it explores a complete layer of new nodes at each iteration before going to next layer.
It is likes DFS for a single iteration.

It is preferred when there is a large search space, and the depth of a solution is notknown. But it performs the wasted computation before reaching the goal depth. Since IDDFS expends all nodes at a given depth before expending any nides at greater depth, it is guaranteed to find a shortest-length (path) solution from initial state to goal state.
At any given time, it is performing a DFS and never searches deeper than depth 'd'. Hence, it uses same space as DFS.
Disadvantage of IDDFS is that it performs wasted computation prior to reaching the goal depth.

Algorithm (IDDFS)

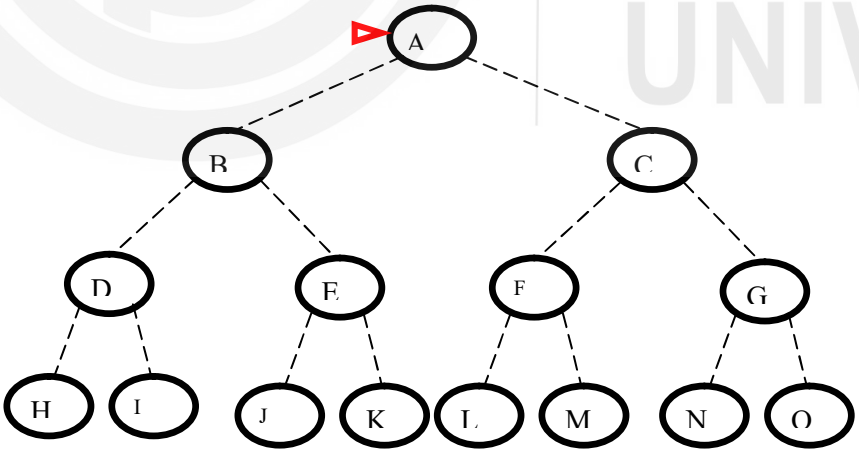
$Initialized\ d = 1/*$ depth of search tree $*/$, found $=$ false
$While\ (Found\ =\ False)$
$DO\{$
$perform\ a\ depth\ first\ search\ from\ start\ to\ depth\ d.$
$if\ goal\ state\ is\ obtained$
$then\ Found\ =\ true$
$else$

$discard\ the\ nodes\ generated\ in\ the\ search\ after\ depth\ d$
$(i.e.\ d+1\ onwards\ till\ last\ of\ tree)$
$\}/*\ end\ while\ */$

$Report\ the\ solution, if\ obtained$
$Stop$

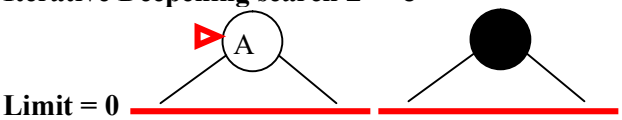
Let us consider the following example to understand the IDDFS:

- Here initial state is A and goal state is M :

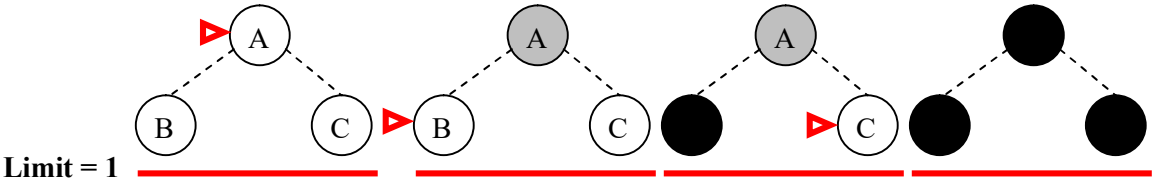


The Iterative deepening search proceeds as follows:

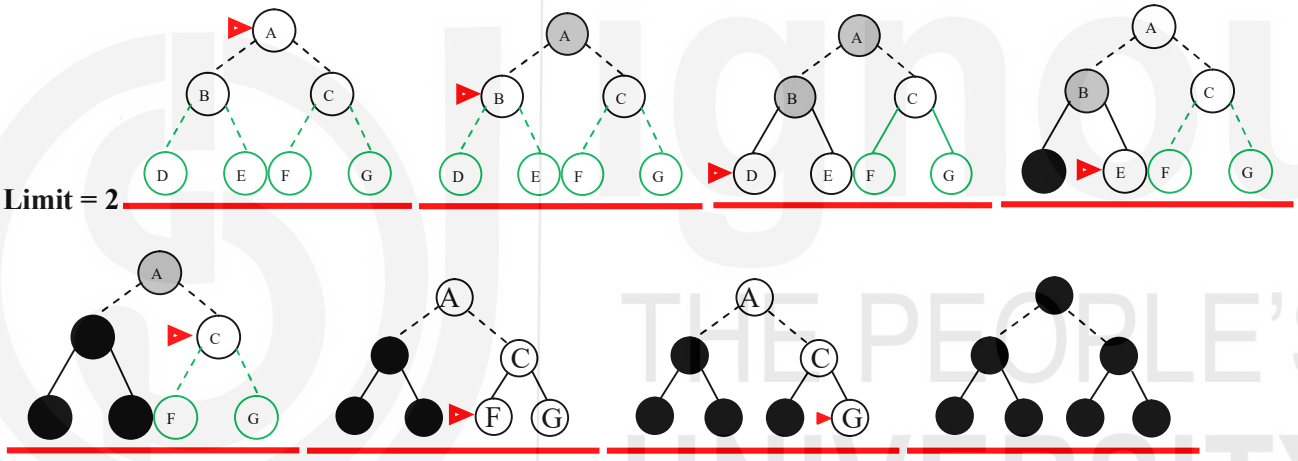
Iterative Deepening search $L = 0$



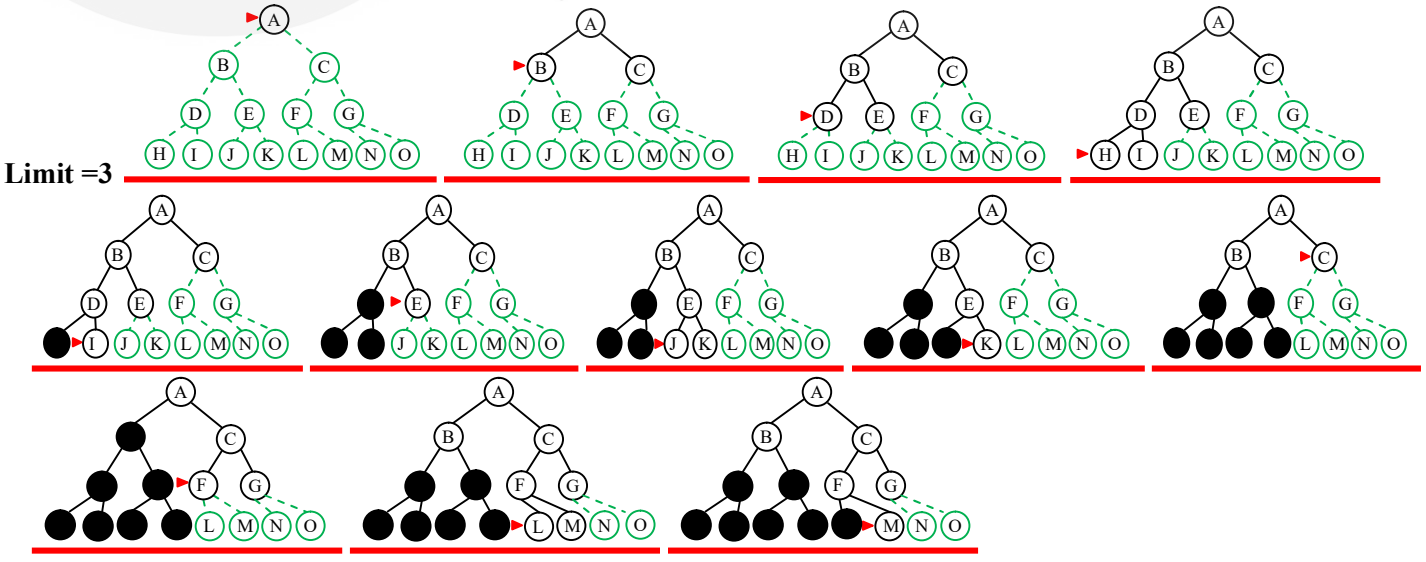
Iterative Deepening search $L = 1$



Iterative Deepening search $L = 2$



Iterative Deepening search $L = 3$



3.4.1 Time and space complexities of IDDFS

The time and space complexities of IDDFSalgorithm is $O(b^d)$ and O(d) respectively.

It can be shown that depth first iterative deepening is **asymptotically optimal,** among brute force tree searches, in terms of time, space and length of the solution. In fact, it is linear in its space complexity like DFS, and is asymptotically optimal to BFS in terms of the number of nodes expanded.

Please note that in general iterative deepening is preferred uninformed search method when there is large search space, and the depth of the solution is unknown. Also note that iterative deepening search is analogous to BFS in that is explores a complete layer going to the next layer.

3.4.2 Advantages and Disadvantages of IDDFS:

Advantages:

1. It combines the benefits of BFS and DFS search algorithms in terms of fast search and memory efficiency.
2. It is guaranteed to find a shortest path solution.
3. It is a preferred uniformed search method when the search space is large and the depth of the solution is not known.

Disadvantages

1. The main drawback of IDDFS is that it repeats all the work from the previous phase. That is, it performs wasted computations before reaching the goal depth.
2. The time complexity is $O(b^d)$ i.e., exponential type only.

The following summarizes when to use which algorithm:

DFS	BFS	IDDFS
Many solutions exist Know (or have a good estimate of) the depth of solution.	Some solutions are known to be shallow	Space is limited and the shortest solution path is required

3.5 Bidirectional Search

This search technique expands nodes from the start and goal state simultaneously. Check at each stage if the nodes of one have been generated by the other. If so, the path concatenation is the solution.

- This search is used when a problem has a single goal state that is given explicitly and all the node generation operators have inverses,

- So, it is used to find shortest path from an initial node to goal node instead of goal itself along with path.
- It works by searching forward from the initial node and backward from the goal node simultaneously, by hoping that two searches meet in the middle.
- Check at each stage if the nodes of one have been generated by the other, i.e., they meet in the middle.
- If so, the path concatenation is the solution.

Thus, the BS Algorithm is applicable when generating predecessors is easy in both forward and backward directions and there exist only 1 or fewer goal states. The following figure illustrate how the Bidirectional search is executed.

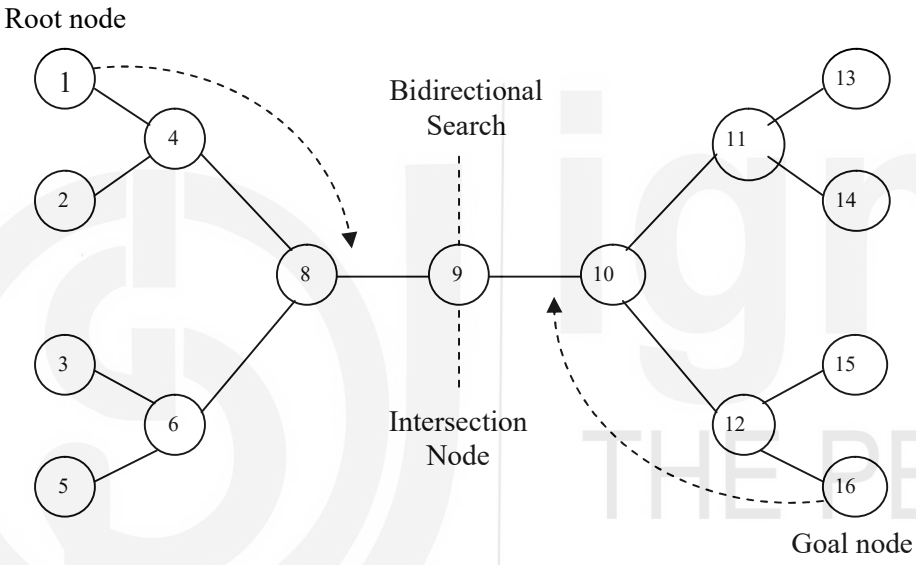


Fig 7 Bidirectional search

We have node 1 as the start/root node and node 16 as the goal node. The algorithm divides the search tree into two sub-trees. So, from start node 1, we do a forward search and at the same time, we do a backward search from goal node 16. The forward search traverse's nodes 1, 4, 8, and 9 whereas the backward search traverses through nodes 16, 12, 10, and 9. We see that both forward and backward search meets at node 9 called the intersection node. So, the total path traced by forwarding search and the path traced by backward search is the optimal solution. This is how the BS Algorithm is implemented.

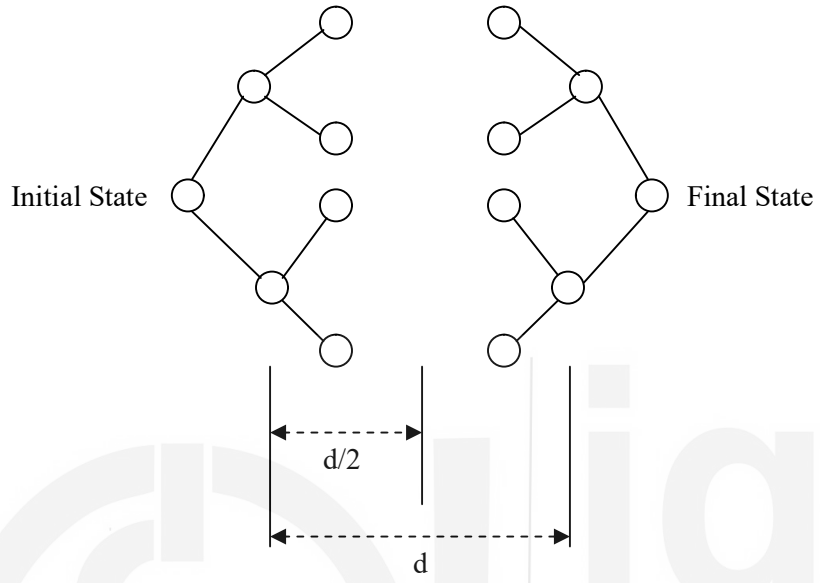
Advantages:

- Since BS uses various techniques like DFS, BFS, Depth limited search (DLS) etc, it is efficient and requires less memory.

Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- In bidirectional search, one should know the goal state in advance.
- Practically inefficient due to additional overhead to perform insertion operation at each point of search.

Time complexity:
The total number of nodes expended in Bidirectional search is= $2b^{d/2} = \boldsymbol{O\left(b^{d/2}\right)}$, where b is a branching factor and d is the depth of the shallowest goal node.



Finally, the Bidirectional search is:

- Complete? Yes
- Time Complexity: $O(b^{d/2})$
- Space complexity: $O(b^{d/2})$
- Optimal: Yes (if step cost is uniform in both forward and backward directions)

3.6 Comparison of Uninformed search strategies

The following table-1compare the efficiency of uninformed search algorithms. These are the measure to evaluate the performance of the search algorithms:

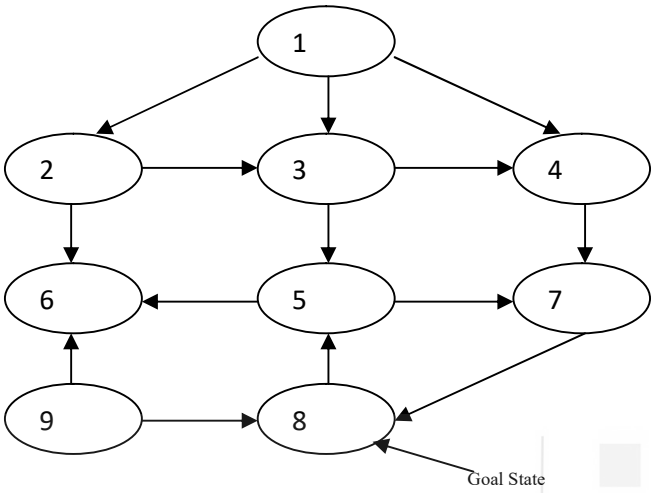
Table -1 Performance of uninformed search algorithm

	BFS	**DFS**	**IDDFS**	**Bidirectional Search (if applicable)**
Time	b^d	b^d	b^d	$b^{d/2}$
Space	b^d	bm	bd	$b^{d/2}$
Optimum?	Yes	No	Yes	Yes
Complete?	Yes	No	Yes	Yes

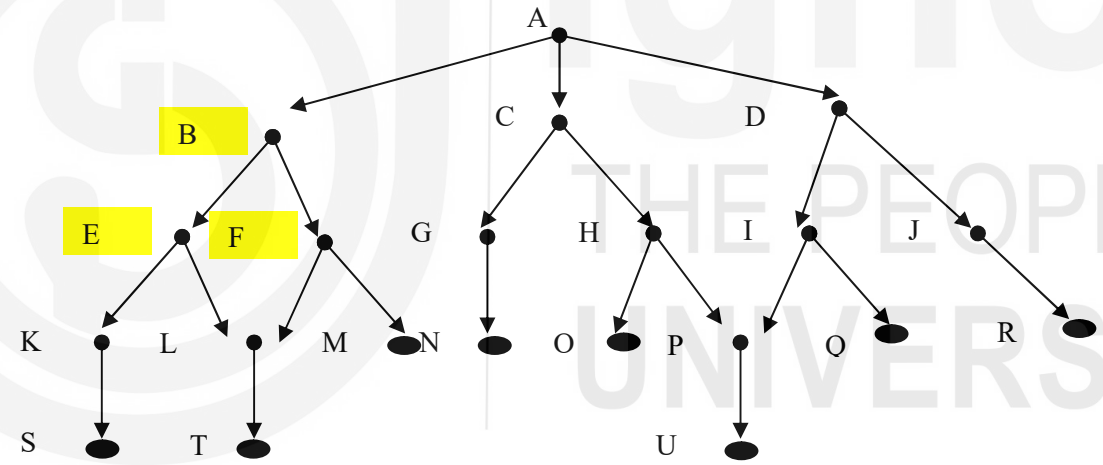
Were
b=branching factor
d=depth of shallowest goal state
m=Maximum depth of the search space

☞ **Check Your Progress 1**

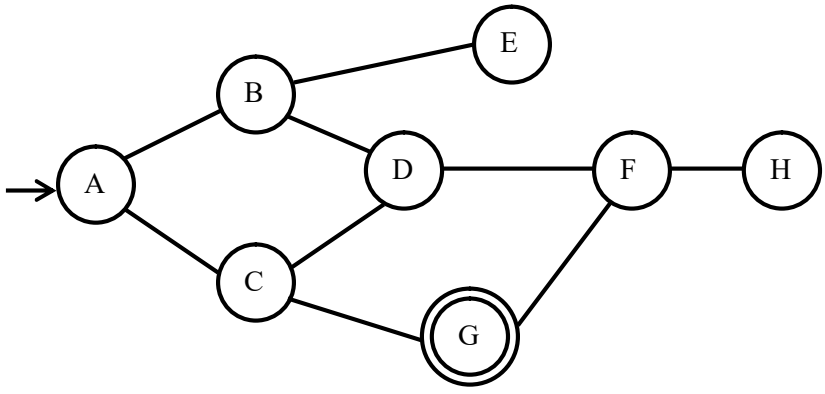
Q.1 Apply BFS and DFS algorithm on the following graph, clearly show the contents of OPEN and CLOSE list.



Q.2 Apply BFS algorithm on the following tree (M is goal node)



Q.3 Apply BFS algorithm on the following graph.



Let A be the state and G be the final or goal state to be searched.

Q.4 Compare the Uninformed search algorithm with respect to Time, space, Optimal and Complete.

3.7 INFORMED (HEURISTIC) SEARCH

Uninformed (blind) search is inefficient in most cases because they do not have any domain specific knowledge about goal state. Heuristic Search Uses domain-dependent (heuristic) information beyond the definition of the problem itself in order to search the space more efficiently.

The following are some ways of using heuristic information:

- Deciding which node to expand next, instead of doing the expansion in a strictly breadth-first or depth-first order;
- In the course of expanding a node, deciding which successor or successors to generate, instead of blindly generating all possible successors at one time:
- Deciding that certain nodes should be discarded, or pruned, from the search space.
-

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently thanan uninformed search strategy. Informed search is also called a **Heuristic search**.
Heuristics is a guess work, or additional information about the problem. It may miss the solution, if wrong heuristics is supplied. However, in almost all problems with correct heuristic information, it provides good solution in reasonable time
Informed search can solve much complex problem which could not be solved in another way.
We have the following informed search algorithm:

1. Best-First Search
2. A* algorithm
3. Iterative Deepening A*

3.7.1Strategies for providing heuristics information:

The informed search algorithm is more useful for large search space. All the informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

"Heuristics are criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal"

Heuristic Function:

Heuristic information is provided in form of function called **heuristic function.**

- ❖ Heuristic is a function which is used in Informed Search, and it finds the most promising path.
- ❖ It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.
- ❖ Heuristic function estimates how close a state is to the goal. It is represented byh(n), and it calculates the cost of an optimal path between the pair of states.
- ❖ The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.
- ❖ This technique always uses to find solution quickly.

- Informed Search Define a heuristic function. h(n). that estimates the "goodness" of a node n.
- The heuristic function is an estimate, based on domain-specific information that is computable from the current state description of how close we are to a goal.
- Specifically, h(n) = estimated cost (or distance) of minimal cost path from state 'n' to a goal state.

A heuristic function at a node n is an estimate of the optimum cost from the current node toa goal. Denoted by *h(n)*

h(n) = estimated cost of the cheapest path from node n to a goal node

For example, suppose you want to find a shortest path from Kolkata to Guwahati, then heuristic for Guwahati may be straight-line distance between Kolkata and Guwahati, that is

$h(Kolkata) = \boldsymbol{euclideanDistance(Kolkata, Guwahati)}$

3.7.2 Formulation of informed (heuristic) search problem as State Space

Informed search problems can be represented as state space. The state space of a problem includes: an Initial state, one or more goal state, set of state transition operator O (a set of rules), used to change the current state to another state and a heuristic function h.

In general, a state space is represented by 5 tuples as follows: S: $[S, s_0, O, G, h]$

Where S: (Implicitly specified) Set of all possible states (possibly infinite).

s_0: start state of the problem, $s_0 \in S$.

O: Set of state transition operator, each having same cost. This is used to change the state from one state to another. It is the set of arcs (or links) between nodes

G: Set of Goal state, $G \subseteq S$.

$h()$: A heuristic function, estimating the distance toa goal node.

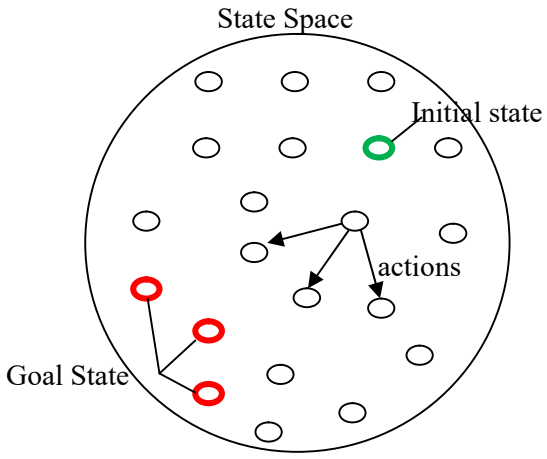


Fig 8 State space with initial and goal node

We need to find a sequence of actions which transform the agent from the initial sate s_0 to Goal state G. State space is commonly defined as a directed graph or as a tree in which each node is a state and each arc represents the application of an operator transforming a state to a successor state.
Thus, the problem is solved by using the rules (operators), in combination with an appropriate control strategy, to move through the problem space until a path from **initial state to a goal state** is found. This process is known as **search**. A solution path is a path in state space from s_0(initial sate) to G (Goal state).

We have already seen an OPEN list is used to implement an uninformed (Blind) search (section 3.2). But the problem with using only one list OPEN is that, it is not possible to keep track of the node which is already visited. That is "***how we can maintain a part of the state space that is already visited***". To save the explicit space, we maintained another list called **CLOSED.** Now we can select a node from OPEN and save it in **CLOSED**. Now, when we generate successor node from CLOSED, we check whether it is already in $(OPEN \cup CLOSED)$. If it is already in $(OPEN \cup CLOSED)$, we will not insert in OPEN again, otherwise insert.

3.7.3 Best-First Search

Best first search uses an evaluation function f(n) that gives an indication of which node to expand next for each node. Every node in a search space has an evaluation function (heuristic function) associated with it. A heuristic function value h(n) on each node indicates how the node is from the goal node. Note that Evaluation function=heuristic cost function (in case of minimization problem) OR objective function(in case of maximization).Decision of which node to be expanded depends onvalue of evaluation function. Evaluation value= cost/distance of current node from goal node and for goal node evaluation function value=0

Based on the evaluation function, f(n), Best-first search can be categorized into the following categories:

1) Greedy Best first search
2) A* search

The following 2 list (**OPEN** and **CLOSED**) are maintained to implement these two algorithms.

1. **OPEN** – all those nodes that have been generated & have has heuristic function applied to them but have not yet been examined.
2. **CLOSED**- contains all nodes that have already been examined.

3.7.4Greedy Best-First search:

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

f(n)= g(n).

Were, h(n)= estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the **priority queue (**or to store the heuristic function value).

Best first search algorithm:
1. Initialize: Set $OPEN = \{s\}$, CLOSED={ };
$f(s) = h(s)$
2.Fail: If $OPEN = \{\}$, terminate with failure**.**
3. Select: Select the minimum cast state $\boldsymbol{a}$ from OPEN, save n in CLOSED**.**
4.Terminate: If $a \in Goal\ node$, terminate with success and return f(n), else
5. Expend: For each successor, m of n,
If $m \notin [OPEN \cup CLOSED]$
$Set\ f(m) = h(m)\ and\ \ Insert\ m\ in\ OPEN$
……
6. LOOP: Goto Step 2

OR Pseudocode (for Best-First Search algorithm)

Step 1: Place the starting node into the OPEN list.
Step 2: If the OPEN list is empty, Stop and return failure.
Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
Step 4: Expand the node n, and generate the successors of node n.
Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both lists, then add it to the OPEN list.
Step 7: Return to Step 2.

Advantages of Best-First search:

- Best first search can switch between BFS and DFS, thus gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

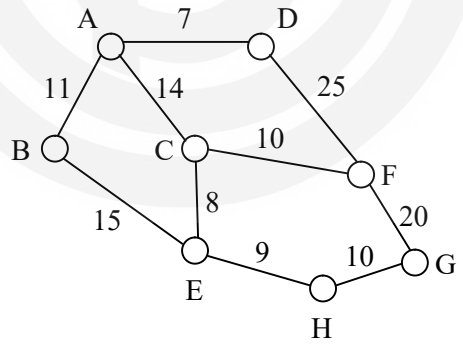
Disadvantages of Best-First search:

- Chances of getting stuck in a loop are higher.
- It can behave as an unguided depth-first search in the worst-case scenario.

Consider the following example for better understanding of greedy Best-First search algorithm.

Example1: Consider the following **example** (graph) with heuristic function value h(n)[Fig 2] which illustrate the greedy Best-first search. Note that in the following example, heuristic function is defined as

$$\boldsymbol{h_{SLD}} = straight\ line\ distance\ from\ n\ to\ goal$$



Let heuristic function value h(n) for each node n to goal node G is defined as

Straight line distance
A→G=h(A)=40
B→G=h(B)=32
C→G=h(C)=25
D→G=h(D)=35
E→G=h(E)=19
F→G=h(F)=17

H→G=h(H)=10
G→G=h(G)=0

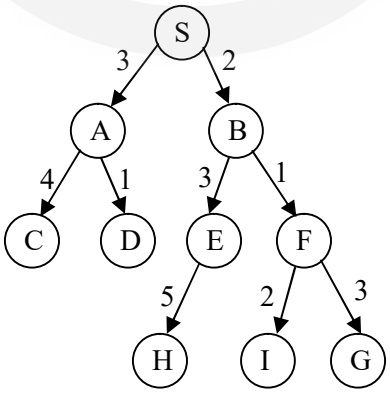
h(n)= straight line distance from node n to G

Note that h(G)=0

The nodes added/deleted from OPEN and CLOSED list using Best-First Search algorithm are shown below.

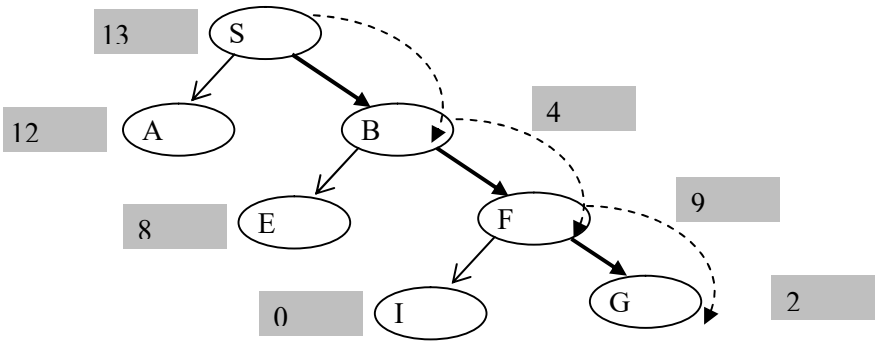
OPEN	**CLOSED**
[A]	[ ]
[C,B,D]	[A]
B,D	A,C
F,E,B,D	A,C
G.E.B.D	A,C,F
E,B,D	A,C,F,G

Explanation are as follows:
Step1:initially OPEN list start with start state 'A' and CLOSED list with empty.
Step2: Children of A={C[25], B[32] D[35]}, so
OPEN={C[25], B[32], D[35]} therefore Best=C, so expend C node next.
Step3: Children of C={E[19],F[17]}, so
OPEN={F[17],E[19], B[32], D[35]} therefore Best=F, so expend node F next.
Step4: Children of F={G[0]}, therefore OPEN= {G[0], E[19], B[32], D[35]} Best=G, this is a goal node so Stop.
Finally, we got the shortest path: A→C→F→G and cost is 44.
Example2: Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



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

Here, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]
Iteration2: Open [E, F, A], Closed [S, B]
: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]
: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be **S----> B----->F----> G**

Evaluation of Best-First Search algorithm:

Time Complexity:

The worst-case time complexity of Greedy best first search is $O(b^m)$.

Space Complexity:

The worst-case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

Example3: Apply Greedy Best-First Search algorithm on the following graph (L is a goal node).

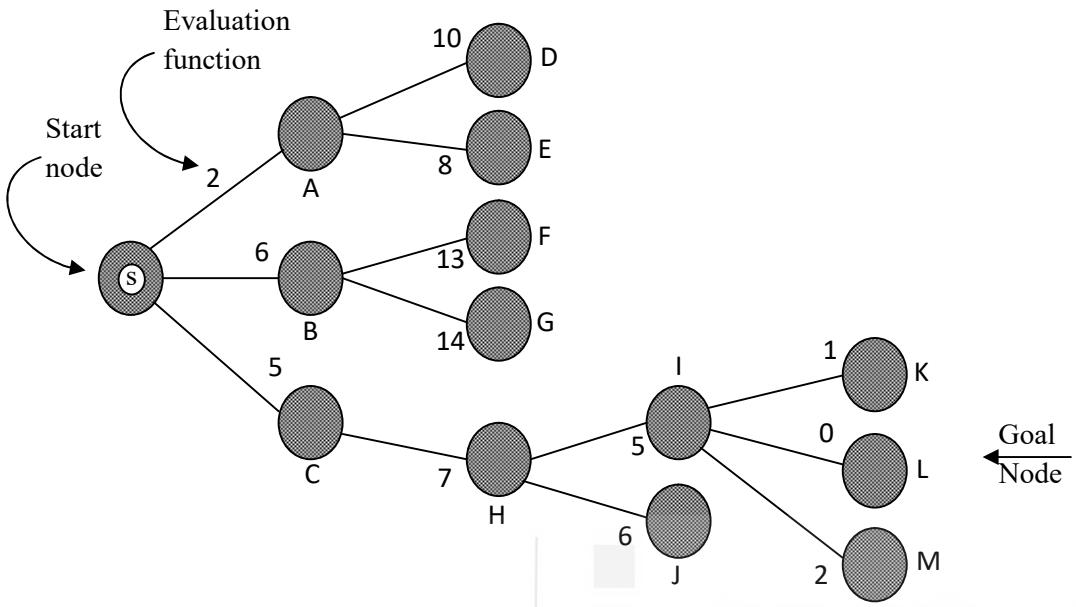


Fig. 2.17: Greedy best first search graph.

Working: We start with a start-nodes, S. Now, S has three children i.e., A, B and C with their Heuristic function values 2, 6 and 5 respectively. These weights show approximately, how far theyare from goal node. So, we write, children of S are -(A:2), (B: 6), (C:5)
Out of these, the node with minimum value is (A : 2). So, we select A and its children are explored (or generated).
Its children are(D: 10) and (E:8)
The search process now has four nodes to search for, namely-
(B : 6), (C: 5), (8.10) and (E: 8)
Out of these, node-C has the minimal value of 5. So, we select it and expand. So, we get(H: 7) as its child.
Now, the nodes to search are as follows-
(B:6), (D: 10,(E: 8) and (H : 7) and so on.

Working of the algorithm can be represented in tabular firm as follows

Step	**Node being expanded**	**Children (on expansion)**	**Available nodes (to search)**	**Node Chosen**
1.	S	(A:2), (B:6),(C:5)	(A:2), (B:6),(C:5)	(A:2)
2.	A	(D:10), (E:8)	(B:6),(C:5), (D:10), (E:8)	(C:5)
3.	C	(H:7)	(B:6), (D:10), (E:8),(H:7)	(B:6)
4.	B	(F:13), (G:14)	(D:10), (E:8), (H:7) ,(F:13), (G:14)	(H:7)
5.	H	(I:5), (J:6)	(D:10), (E:8),(F:13), (G:14), (I:5), (J:6)	(I:5)
6.	I	(K:1), (L:0), (M:2)	(D:10), (E:8), (H:7) ,(F:13), (G:14), (J:6),(K:1),(L:0),(M:2)	Goal node is found. So, search stops now

3.8 A* Algorithm

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of uniform cost search (UCS) and greedy best-first search, by which it solves the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster.
In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence, we can combine both costs as following, and this sum is called as a **fitness number (Evaluation Function)**.

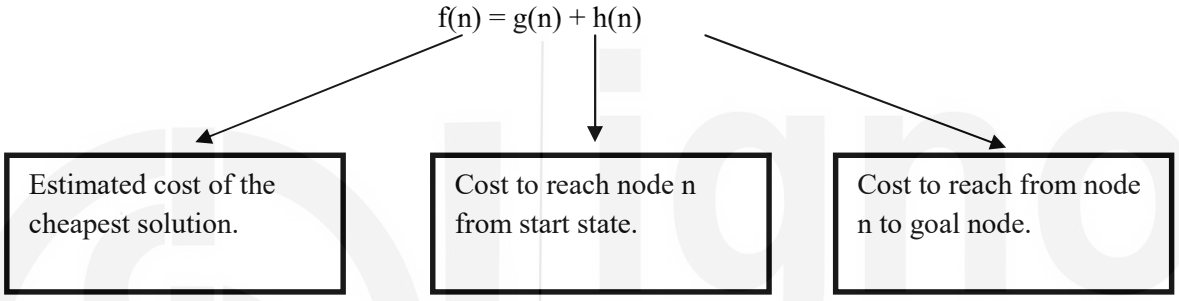


Fig 9 Evaluation function f(n) in A* Algorithm

A* algorithm **evaluation function** $\boldsymbol{f(n)}$is defined as $\boldsymbol{f(n) = g(n) + h(n)}$
Where $\boldsymbol{g(n)}$ =sum of edge costs from start state to n
And $\boldsymbol{h(n)}$ = estimate of lowest cost path from node n→goal node.

If h(n) is **admissible** then search will find optimal solution. Admissible means underestimates cost of any solution which can reached from node. In other words, a heuristic is called admissible if it always ***under-estimates***, that is, we always have $h(n) \leq h^*(n)$, where h*(n) denotes the minimum distance to a goal state from state n.
A* search begins at root node and then search continues by visiting the next node which has the least evaluation value f(n).
It evaluates nodes by using the following evaluation function
f(n) = h(n) +g(n) = estimated cost of the cheapest solution through n.
Where,
g(n): the actual shortest distance traveled from initial node to current node, it helps to avoid expanding paths that are already expansive
h(n): the estimated (or "heuristic") distance from current node to goal, it estimates which node is closest to the goal node.
Nodes are visited in this manner until a goal is reached.

Suppose s is a start state then calculation of evaluation function f(n) for any node n is shown in following figure 10.

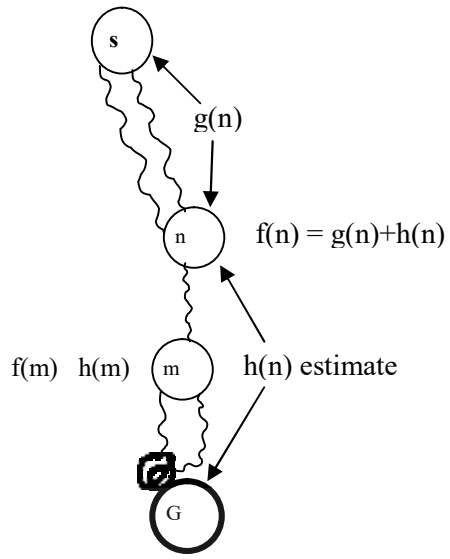


Fig 10 Calculation of evaluation function f(n).

Algorithm A*

1. **Initialize:** Set OPEN = (s), CLOSED = {}, g(s) = 0, f(s) = h(s)
2. **Fail:** If OPEN = { }, Terminate & fail
3. **Select** : Select the minimum cost state, n, from OPEN, Save n in CLOSED.
4. **Terminate**: If n ∈ G, terminate with success, and return f(n)
5. **Expand:** For each successor, m, of n

   If m ∉ [OPEN ∪ CLOSED]
   Set g(m) = g(n) + C(n,m)
   Set f(m) = g(m) +h(m)
   Insert m in OPEN

   If m ∈ [OPEN ∪ CLOSED]
   Set g(m) = min {min {g(m), g(n) + C(n,m)}
   Set f(m) = g(m) + h(m)

   If f(m) has decreased and m ∈ CLOSED,
   move m to OPEN

In step5, we generate a successor of n (say m) and for each successor m, if it does not belong to OPEN or CLOSED that is $m \notin [OPEN \cup CLOSED]$, then we insert it in OPEN with the cost g(n)+C(n,m) i.e., cost up to n and additional cost from n→m.
If $m \in [OPEN \cup CLOSED]$ then we set g(m) with original cost and new cost $[g(n) + C(n,m)]$. If we arrive at some state with another path which has less cost from original one, then we replace the existing cost with this minimum cost.
If we find f(m) is decreased (if larger then ignore) and $m \in CLOSED$ then move m from CLOSED to OPEN.

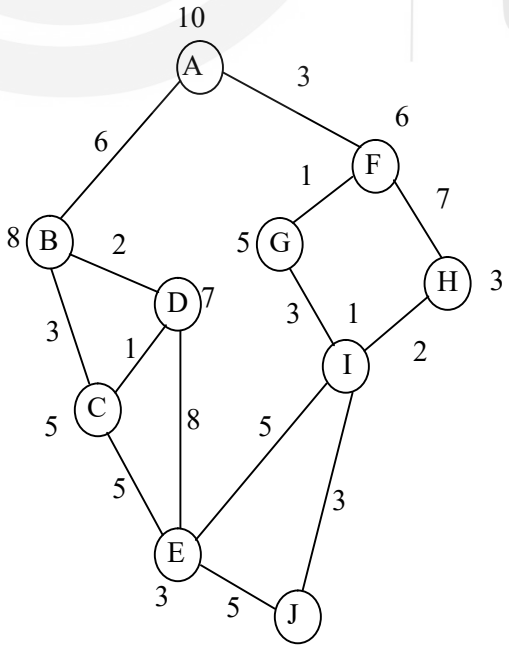
Note that, the implementation of A* Algorithm involves maintaining two lists- **OPEN** and **CLOSED**. The list **OPEN** contains those nodes that have been evaluated by the heuristic function but have not expanded into successors yet and the list **CLOSED** contains those nodes that have already been visited.

See the following steps for working of A* algorithm:

Step-1: Define a list OPEN. Initially, OPEN consists of a single node, the start node S.
Step-2: If the list is empty, return failure and exit.
Step-3: Remove node n with the smallest value of f(n) from OPEN and move it to list CLOSED.
If node n is a goal state, return success and exit.
Step-4: Expand node n.
Step-5: If any successor to n is the goal node, return success and the solution by tracing the path from goal node to S.
Otherwise, go to Setp-6.
Step-6: For each successor node,
Apply the evaluation function f to the node.
If the node has not been in either list, add it to OPEN.
Step-7: Go back to Step-2.

3.8.1 Working of A* algorithm

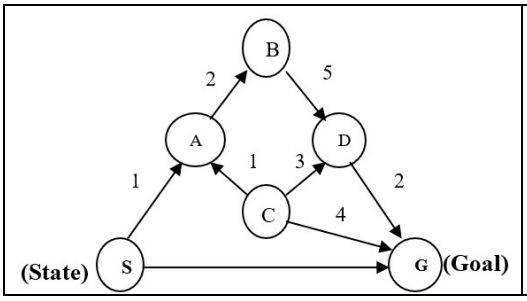
Example1: Let's us consider the following graph to understand the working of A* algorithm. The numbers written on edges represent the distance between the nodes. The numbers written on nodes represent the heuristic value. Find the most cost-effective path to reach from start state **A** to **final** state **J** using A* Algorithm.



Step-1:
We start with node A. Node B and Node F can be reached from node A. A* Algorithm calculates f(B) and f(F). Estimated Cost f(n) = g(n) +h(n) for Node B and Node F is:
f(B) = 6+8=14
f(F) = 3+6=9

Since f(F) < f(F), so it decides to go to node F.
➔ Closed list (F)
Path- A →F
Step-2: Node G and Node H can be reached from node F.
A* Algorithm calculates f(G) and f(H).
f(G) = (3+1)=5=9
f(H) = (3+7) +5=13
Since f(G) < f(H), so it decides to go to node G.
➔ Closed list (G)
Path- A→F→G
Step-3: Node I can be reached from node G.
A* Algorithm calculates f(I).
f(I)=(3+1+3)+1=8; It decides to go to node I.
➔ Closed list (I).
Path- A→F→G→I
Step-4: Node E, Node H and Node J can be reached from node I.
A* Algorithm calculates f(E), f(H), f(J).
f(E) = (3+1+3+5) + 3 =15
f(H) = (3+1+3+2) + 3 = 12
f(J) = (3+1+3+3) +0 = 10
Since f(J) is least, so it decides to go to node J.
➔ Closed list (J)
Shortest Path - A→F→G→I→J
Path Cost is 3+1+3+3=10

Example 2: Consider the following graph and apply A* algorithm and find the most cost-effective path to reach from start state **S** to **final** state**G**. The heuristic function value of each node n is defined in the table given.



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

Solution:

S→A=1+3=4

S→G=10+0=10

S→A→B=1+2+4=7

S→A→C=1+1+2=4

S→A→C→D=1+1+3+6=11

S→A→C→G=1+1+4=6

S→A→B→D=1+2+5+6=14

S→A→C→D→G=1+1+3+2=7

S→A→B→D→G=1+2+5+2=10

3.8.2 Advantages and disadvantages of A* algorithm

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

3.8.3 Admissibility Properties of A* algorithm

A heuristic is called admissible if it always ***under-estimates***, that is, we always have $h(n) \leq h^*(n)$, where $h^*(n)$ denotes the minimum distance to a goal state from state n.For finite state spaces, A* always terminates.

In other words, **if the heuristic function h always underestimates then true cost h* (that is heuristic function cost h(n) is smaller than true cost h*(n)), then A* is guaranteed to find an optimal solution.**

If there is a path from s to a goal state, A* terminates (even when the state space is infinite)**.** Algorithm A* is admissible, that is, if there is a path from s to a goal state, A* terminates by finding an optimal path **.** If we are given two or more admissible heuristics, we can take there max to get a stronger admissible heuristic.

Admissibility Condition :

By admissible algorithm. we mean that the algorithm is sure to find a most optimal solution if one exists. Please note that this is possible only when the evaluation function value never overestimates the distance of the node to the goal. Also note that if the evaluation function value which is a heuristic one is exactly the same of the distance of the node to the goal, then this algorithm will immediately give the solution.

For example, the A* algorithm discussed above is admissible. There are three conditions to be satisfied for A to be admissible.
They are as follows-

1. Each node in the graph has finite number of successors (or O).
2. All arcs in the graph have costs greater than some positive amount, (say C).
3. For each node in the graph, n, h(n) ≤h'(n).

This implies that the heuristic guess of the cost of getting from node n to the goal is never an overestimate. This is known as a **heuristic condition.** Only if these three conditions are satisfied, A* is guaranteed to find an optimal (least) cost path. **Please note that A* algorithm is admissible for any node n if on such path, h'(n) is always less than or equal to h(n).** This is possible only when the evaluation function value never overestimates the distance of the node to the goal. Although the admissibility condition requires h'(n) to be a lower bound on h(n), it is expected that the more closely h'(n) approaches h(n), the better is the performance of the algorithm.

If h(n) = h'(n) -an optimal solution path would be found without over expanding a node of the path. We assume that one optimal solution exists. If h'(n)=0 then A* reduces to blind uniform cost algorithm or breadth-first algorithm.

Please note that the admissible heuristics are by nature optimistic because they think that the cost of solving the problem is less than it actually is because g(n) is the exact cost for each n. Also note that f(n) should never overestimate the true cost of a solution through n.
For example, consider a network of roads and cities with roads connecting these cities. Our problems to find a path between two cities such that the mileage/fuel cost is minimal. Then an admissible heuristic would be to use distance to estimate the costs from a given city to the goal city. Naturally, the air distance will be either equal to the real distance or will underestimate it i.e., h(n) ≤ h'(n).

3.8.4 Properties of heuristic Algorithm:

1. **Admissibility condition**: Algorithm A is admissible if it guarantees to return an optimal solution when one exists. A heuristic function h is called admissive if many general estimates., we always have $h(n) \leq h^*(n)$
2. **Completeness condition**: Algorithm A is complete if it always terminates with a solution when one exists.
3. **Dominance property**: If A_1 and A_2 are two Admissible versions of Algorithm A such that A is more informed than A_2 $h_1(n) > h_2(n)$.
4. **Optimal Property**: Algorithm A is optimal over a class of Algorithms If a dominates all members of the class.

3.8.5 Results on A* Algorithm

1. **A* is admissible: Algorithm A* is admissible** , that is, if there is a path from S to goal state, A* terminates by finding an optimal solution.
2. **A* is complete**: If there is a path from S to goal state, **A terminates** (Even when the state space is ∞).

3. **Dominance property**: If A_1& A_2→ two Admissible versions of A* S.t.
   A1 is more informed than A_2, then A_2 expends at least as many states as does A_1. (So A_1 dominates A_2 Here, b/s its better heuristics than A_2)

   If we are given two or more admissible heuristics, we can take their max to get a stronger admissible heuristic.

3.9 Problem Reduction Search

Problem reduction search is broadly defined as a planning how best to solve a problem that can be recursively decomposed into subproblems in multiple ways. There are many ways to decompose a problem, we have to find the best decomposition, which gives the quality of searching or cost is minimum.

We already know about the divide and conquer strategy, a solution to a problem can be obtained by decomposing it into smaller sub-problems. Each of this sub-problem can then be solved to get its sub solution. These sub solutions can then be recombined to get a solution as a whole. That is called is **Problem Reduction**. This method generates arc which is called as AND arcs. One AND arc may point to any number of successor nodes, all of which must be solved for an arc to point to a solution.

When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, AND-OR graphs or AND - OR trees are used for representing the solution. The decomposition of the problem or problem reduction generates AND arcs. Consider the following example to understand the AND-OR graph (figure-11).

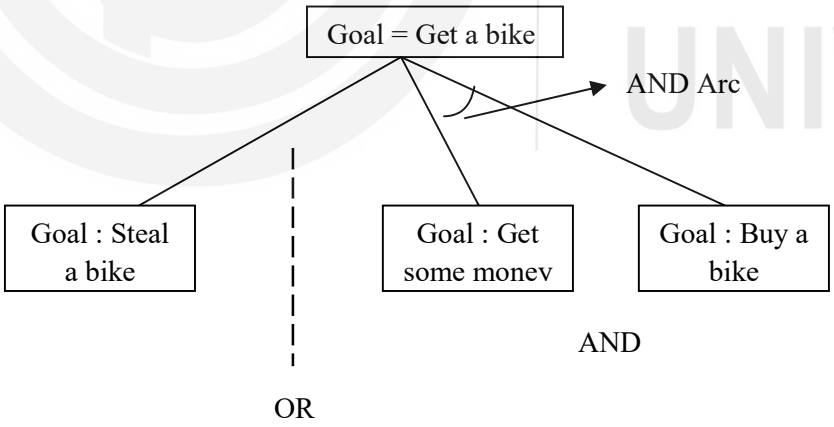


Fig 11**AND-OR graph**

The figure-11 shows an AND-OR graph. In an AND-OR graph, OR node represents a choice between possible decompositions, and an AND node represents given decomposition. For example, to Get a bike, we have two options, either:

1.(Steal a bike)

OR

2. Get some money **AND** Buy a Bike.

In this graph we are given two choices, first Steal a bike or get some money **AND** Buy a Bike. When we have more than one choice and we have to pick one, we apply OR condition to choose one.(That's what we did here).
Basically, the ARC here denotes AND condition.
Here we have replicated the arc between the Get some money and buy a bike because by getting some money possibility of buying a bike is more than stealing.
AO* search algorithm is based on AND-OR graph, so it is called AO* search algorithm. AO* Algorithm basically based on problem decomposition (Breakdown problem into small pieces).
The main difference between the A*(A star) and AO*(AO star) algorithms is that A* algorithm represents an OR graph algorithm that is used to find a single solution (either this or that). Butan **AO* algorithm** represents an AND-OR graph algorithm that is used to find more than one solution by ANDing more than one branch.
A* algorithm guarantees to give **an optimal solution** while AO* doesn't since AO* doesn't explore all other solutions once it got a solution.

3.9.1 Problem definition in AND-OR graph:

Given **[G, s, T]**
Where **G:** Implicitly specified AND/OR graph
s: Strat node of the AND/OR graph
T: Set of terminal nodes (called SOLVED)
h(n): Heuristic function estimating the cost of solving the sub problem at n.

Example1:
Let us see one example with the presence of heuristic value at every node (see fig 2) . The estimated heuristic value is given at each node. The heuristic value h(n) at any node indicates "from this node at least h(n) value (or cost) is required to find solution". Here we assume the edge cost value (i.e., g(n) value) for each edge is 1. Remember in OR node we always mark that successor node which indicates best path for solution.

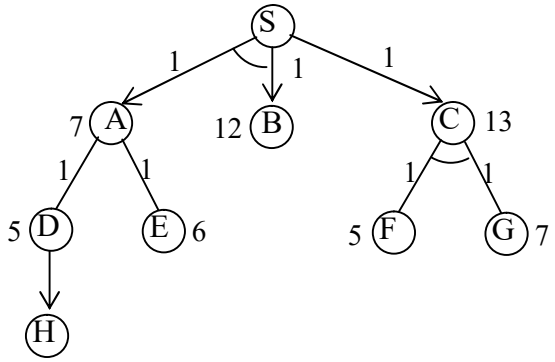
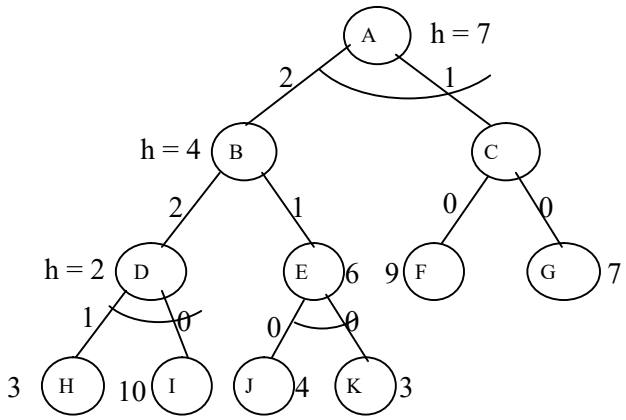


Fig 12: AND-OR graph with heuristic value at each node

Note that, the graph given in fig 2, there are two paths for solution from start state S: either S-A-B or S-C. To calculate the cost of the path we use the formula f(n)=g(n)+h(n) [note that here g(n) value is 1 for every edge].
Path1: f(S-A-B)=1+1+7+12=21
Path2: f(S-C)=1+13=14

Since $min(21,14) = 14$; so, we select successor node C, as its cost is minimum, so it indicates best path for solution.
Note that C is a AND node; so, we consider both the successor node of C. The cost of node C is f(C-F-G)=1+1+5+7=14; so, the revised cost of node C is 14 and now the revised cost of node S is f(S-C)=1+14=15 (revised).
Note that once the cost (that is f value) of any node is revised, we propagate this change backward through the graph to decide the current best path.
Now let us explore another path and check whether we are getting lessor cost as compared to this cost or not.
F(A-D)=1+5=6 and f(A-E)=1+6=7; since A is an OR node so best successor node is D since min(6,7)=6. So revised cost of Node A will be 6 instead of 7, that is f(A)=6 (revised). Now next selected node is D and D is having only one node H so f(D-H)=1+2=3, so the revised cost of node D is 3, so now the revised cost of node A, that is f(A-D-H)=4. This path is better than f(A-E)=7. So, the final revised cost of node A is 4. Now the final revised cost of f(S-A-B)=1+1+4+12=18 (revised).
Thus, the final revised cost for
Path1: f(S-A-B)=18 and
Path2: f(S-C)=15
So optimal cost is 15.

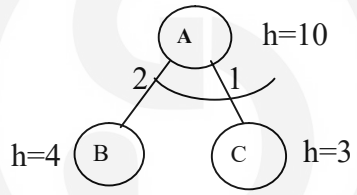
Example2:
Consider the following AND-OR Graph with estimated heuristic cost at every node. Note that A,D,E are AND node and B, C are OR node. Edge cost (i.e., g(n) value) is also given.
Apply AO* algorithm and find the optimal cost path using AO* algorithm.



A,D,E → AND Node
B,C → OR Node

Fig- 1: AND-OR graph with heuristic function value.

Heuristic (Estimated) cost at every Node is given. For example, heuristic cost at node A is h=7, which means at least 7-unit cost required to find a solution.
Since A is AND Node, so we have to solve Both of its successor Node B and C.
Cost of Node A i.e., f(A-B-C) = (2+4) + (1+3) = 10
We perform cost revision in Bottom-up fashion.



i.e., heuristic value of Node A will be 10.

Fig (a)Cost revision

Now, Let's see R.H.S first, Node C is an OR Node.
So, cost f(C-F)= 0+9=9
& f(C-G) =0+7=7
So Best successor of C in G, so we perform cost revision in Bottom-up fashion as follows:

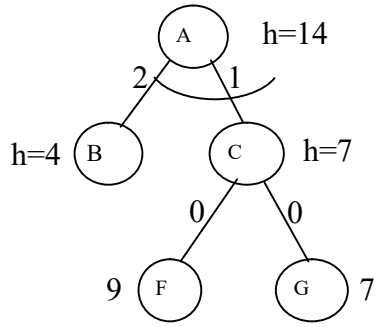


Fig (b): Cost revision in Bottom-up.

Now revision cost of Node A is 14; So, till now, we can say that the best path to solve the problem is: G-C-A. Now expend left node of root, i.e., Node B.

Since B is an OR Node; So, we have to See, which successor is best; i.e., D or E.
The revised cost of Node D is = (3+1) +(10+0) =14
And the revised cost of Node E = (40+0)+(3+0) = 7
So, the Best promising successor Node is E.
Now, perform cost Revision in Bottom-up fashion.

Note that all the leaf Node in the Marked tree is solved is solved. So, the best way to solve the problem is to following the marked tree and solving those marked problem. This best cost to solve the problem is **18.** Note that for **AND Node**: if both successor (unit problem) is solved, then we declared SOLVED and for **OR Node**: if any one best successor is SOLVED, then we declared SOLVED.

3.9.2 AO* algorithm

Our real-life situations cannot be exactly decomposed into either AND tree or OR tree but is always combination of both. So, we need an AO* algorithm where O stands for 'ordered'. Instead of two lists OPEN and CLOSED of A* algorithm, we use a single structure GRAPH in AO* algorithm. If represents a part of the search graph that has been explicitly generated so far. Please note that each node in the graph will point both down to its immediate successors and to its immediate predecessors. Also note that each node will have some h'(n) value associated with it. But unlikeA* search, g(n) is not stored. It is not possible to compute a single value of g(n) due to many paths to the same state. It is not required also as we are doing top-down traversing along best-knownpath.
This guarantees that only those nodes that are on the best path are considered for expansion
Hence, h'(n) will only serve as the estimate of goodness of a node.
Next, we develop an AO* algorithm.

Algorithm AO*

1. Initialize: Set G* ={s}, f(s) =h(s)
   If s ∈ T, label s as SOLVED
2. Terminate: If s is SOLVED, then Terminate
3. Select: Select a non-terminal leaf node n from the marked
   sub-free
4. Expand: Make explicit the successors of n for each new successors, m:
   Set f(m) =h(m)
   If m is terminal, label m SOLVED
5. Cost Revision: Call C**ost-Revise (n)**
6. Loop: Go to Step 2.

Cost Revision in AO*: Cost-Revise(n)

1. Create Z = {n}
2. If Z = {} return
3. Select a node m from Z such that m has no descendants in Z
4. If m is an AND node with successors $r1, r2, \ldots rk$:

Set $f(m) = \sum [f(r_i) + c(m, r_i)]$
Mark the edge to each successor of m
If each successor is labelled SOLVED,
then label m as SOLVED.

5. If m is an OR node with successor $r1, r2, \ldots rk$:

Set $f(m) = min\{f(r_i) + c(m, r_i)\}$
Mark the edge to the best successor of m
If the marked successor is labelled SOLVED, label m as SOLVED

6. If the cost or label of m has changes, then insert those parents of m into Z for which ma is a marked successor

3.9.3 Advantage of AO* algorithm

Note that **AO* will always find a minimum cost solution if one exists if** $h'(n) < h(n)$ and that all arc costs are positive. The efficiency of this algorithm will depend on how closely $h'(n)$ approximates $h(n)$. Also note that **AO* is guaranteed to terminate** even on graphs that have cycles**.**

Note: When the graph has only OR node then AO* algorithm works just like A* algorithm.

3.10 Memory Bound Heuristic Search

The following are the commonly used memory bound heuristics search:

1. Iterative deepening A* (IDA*)
2. Recursive Best-First search (RBFS)
3. Memory bound A* (MBA*)

3.10.1: Iterative Deepening A* (IDA*)

IDA* is a variant of the A* search algorithm which uses iterative deepening to keep the memory usage lower than in A*.It is an informed search based on the idea of the uniformed iterative deepening search. Iterative deepening A* or IDA* is similar to iterative-deepening depth-first, but with the following modifications:

The depth bound modified to be an f-limit
1. Start with limit = h(start)
2. Prune any node if f(node) > f-limit
3. Next f-limit = minimum cost of any node pruned

Iterative Deepening is a kind of uniformed search strategy. It combines the benefits of depth-first and breadth- first search.

Advantage of IDA* is: It is optimal and complete like breadth first search and modest memory requirement like depth-first search.

IDA* algorithm

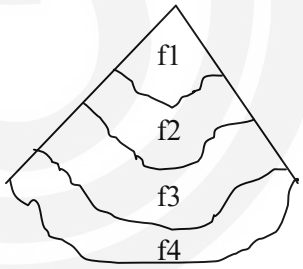
1. Set C= f(s)
2. Perform DFBB with cut-off C
   Expand a state, n, only if its f-value is less than or equal to C
   If a goal is selected for expansion, then return C and terminate
3. Update C to the minimum f-value which exceeded C among states which were examined and Go TO Step 2.

3.10.2 Working of IDA*

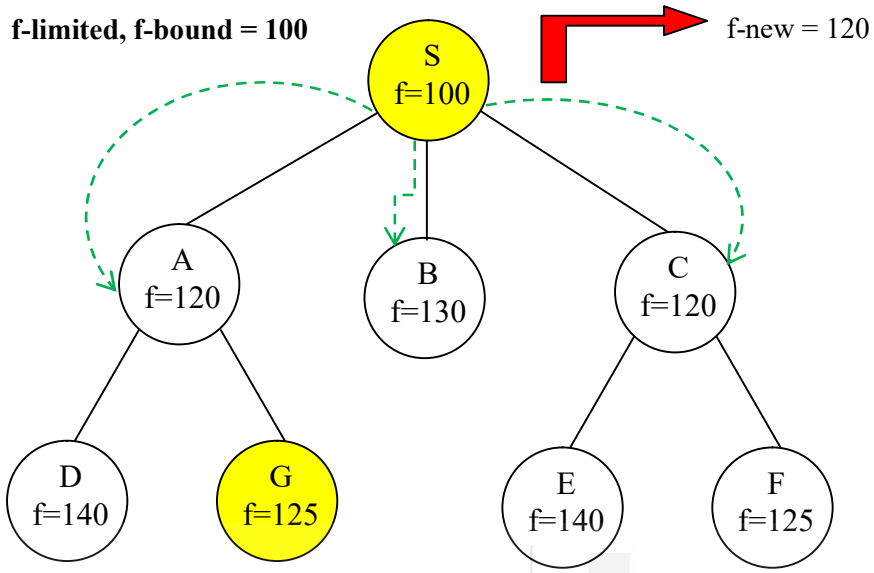
- Perform depth-first search LIMITED to some f-bound.
- If goal found then ok.
- Else: increase the f-bound and restart.

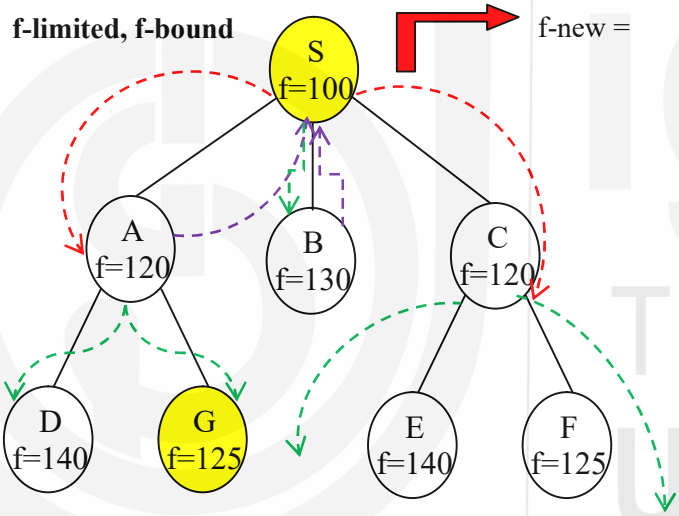
How to establish the f-bound?
Initially: f(S)

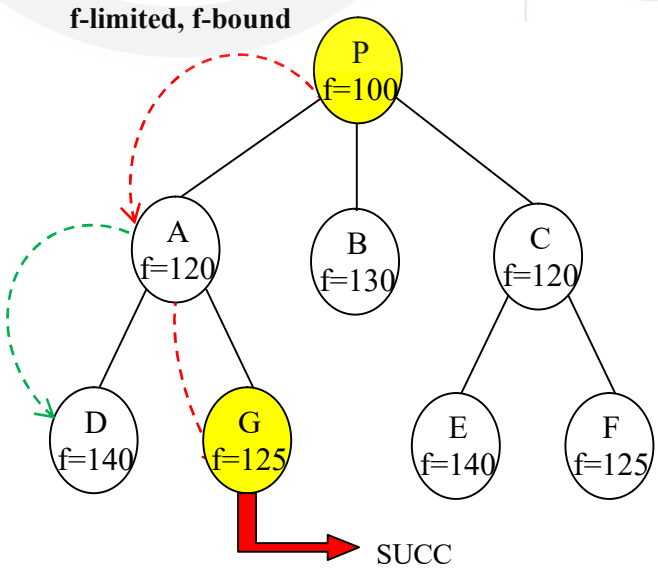
- Generate all successors
- Record the minimal f(succ) > f(S)
- Continue with minimal f(succ) instead of f(S)



Consider the following example to understand the IDA*


f-limited, f-bound = 100
S
f=100
f-new = 120
A
f=120
B
f=130
C
f=120
D
f=140
G
f=125
E
f=140
F
f=125


f-limited, f-bound
S
f=100
f-new =
A
f=120
B
f=130
C
f=120
D
f=140
G
f=125
E
f=140
F
f=125


f-limited, f-bound
P
f=100
A
f=120
B
f=130
C
f=120
D
f=140
G
f=125
E
f=140
F
f=125
SUCC

3.10.3 Analysis of IDA*

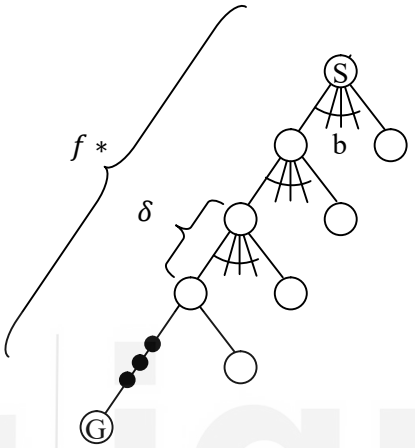
IDA* is complete, optimal, and optimally efficient (assuming a consistent, admissible heuristic), and requires only a polynomial amount of storage in the worst case:

F* = optimal path cost to a goal

B = branching factor

δ= minimum operator step cost

$\frac{bf*}{\delta}$nodes of storage required



Note that IDA* is complete & optimal Space usage is linear in the depth of solution. Each iteration is depth first search, and thus it does not require a priority queue.

3.10.4 Comparison of A* and IDA* algorithm:

- Iterative Deepening Search (IDS) is nothing but BFS plus DFS for tree search.
- IDA* algorithm is "complete and Optimal" algorithm.
- BFA and A* is good for optimality, but not memory.
- DFS: good for memory O(bd), but not optimality
- In the worst case, only one new state is expanded in each iteration
- (IDA*). If A* expands N states, then IDA* can expand:

  $1+2+3+\ldots+N=O(N^2)$

3.11 Recursive Best first search (RBFS)

The idea of recursive best first search is to simulate **A*** search with **O(bd)** memory, where b is the branching factor and d is the solution depth.

It is a memory bound, simple recursive algorithm that works like a standard best first search but only takes up linear space. There are some things that make it different from recursive DFS. It keeps track of f, the value of the best alternative path that can be found from any ancestor of the current node, instead of continuing indefinitely down the current path.

RBFS mimic the operation of standard Best-First search algorithm. IBFS keep track of the f-value of the best alternative path available from any ancestor of the current node. If the current node exceeds the limit, the recursion unwinds back to the alternative path. As the recursion unwinds, RBFS replaces the f-value of each node along the path with the best f-value of its

children. In this way, RBFS remembers the f-value of the best leaf in the forgotten subtree and can therefore decide whether it's worth re-expanding the subtree at some later time.
RBFS is somewhat most efficient than IDA*, but still suffers from excessive node regeneration.
A* and RBFS are optimal algorithms if heuristic function h(n) is admissible.

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, ∞
  for each s in successors do /* update f with value from previous search, if any */
      s.f ← max(s.g + s.h, node.f))
  loop do
      best ← the lowest f-value node in successors
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result
```

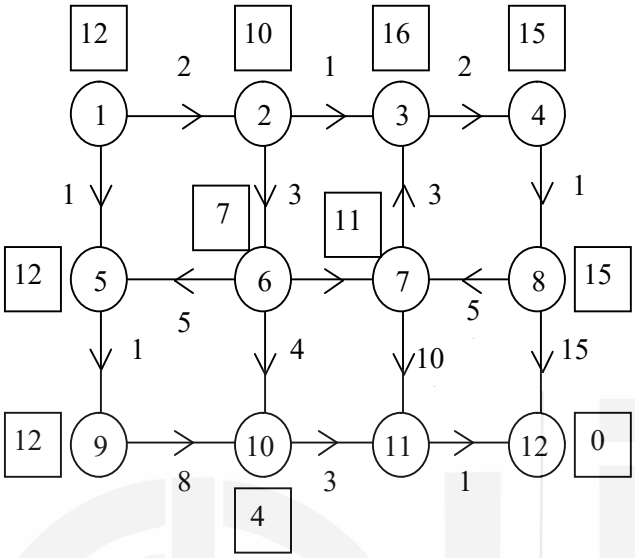
Fig12 Algorithm for recursive Best first Search

3.11.1 Advantages and Disadvantages of RBFS:

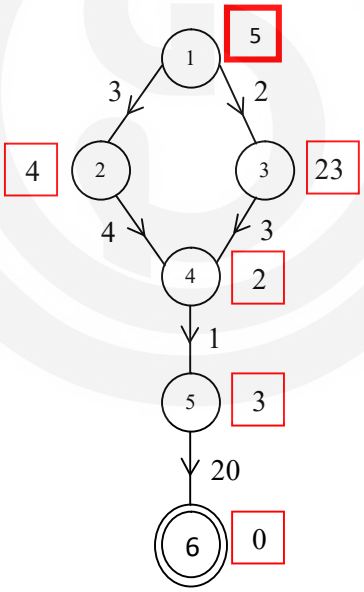
- More efficient than IDA* and still optimal.
- Best-first Search based on next best f-contour; fewer regeneration of nodes.
- Exploit results of search at a specific f-contour by saving next f-contour associated with a node who successors have been explored.
- Like IDA* still suffers from excessive node regeneration.
- IDA* and RBFS not good for graphs.
- Can't check for repeated states other than those on current path.
- Both are hard to characterize in terms of expected time complexity.

☞ Check Your Progress 2

Q.1 Apply A* on the following graph:-

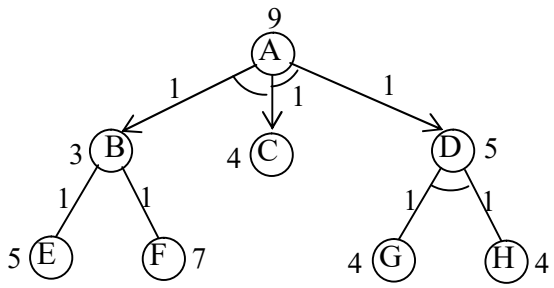


Q.2 Apply A* algorithm on the following graph

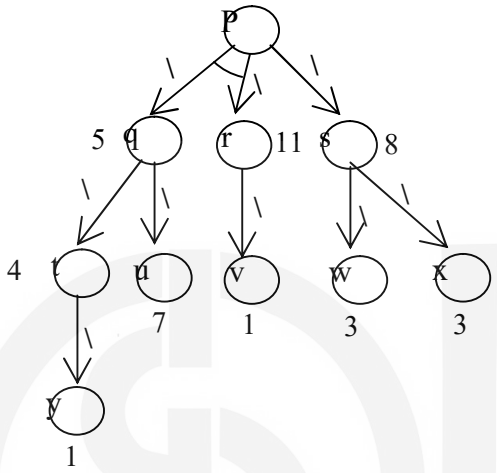


Q.3 Differentiate between the A* and AO* algorithm.

Q.4: Apply AO* algorithm on the following graph. Heuristic value is also given at every node and assume the edge cost value of each node is 1.



Q.5 Apply AO* algorithm on the following graph. Heuristic value is also given at every node and assume the edge cost value of each node is 1.



Example 6: Given the 3 matrices A1, A2, A3 with their dimensions $(3 \times 4), (4 \times 10), (10 \times 1)$. Consider the problem of solving this **chain matrix multiplication**. Apply the concept of AND-OR graph and find a minimum cost solution tree.

(Multiple choice Questions)

Q.6 A* algorithm always to find an optimal solution if

A. h' is always 0
B. g is always 1
C. h' never overestimates h
D. h' never underestimate h

Q.7A* algorithm uses $f^* = g + h^*$ to estimate the cost of getting from the initial state to the goal state, where g is a measure of cost getting from initial state to the current node and the function h^* is an estimate of the cost of getting from the current node to the goal state. To find a path involving the fewest number of steps, we should test,

A. g=1
B. g=0
C. $h^* = 0$
D. $h^* = 1$

3.12 Summary

- As the name 'Uninformed Search' means the machine blindly follows the algorithm regardless of whether right or wrong, efficient or in-efficient.
- These algorithms are brute force operations, and they don't have extra information about the search space; the only information they have is on how to traverse or visit the nodes in the tree. Thus, uninformed search algorithms are also called **blind search** algorithms.
- The search algorithm produces the search tree without using any domain knowledge, which is a brute force in nature. They don't have any background information on how to approach the goal or whatsoever. But these are the basics of search algorithms in AI.
- **The different types of uninformed search algorithms are as follows:**
    - Depth First Search
    - Breadth-First Search
    - Depth Limited Search
    - Uniform Cost Search
    - Iterative Deepening Depth First Search
    - Bidirectional Search (if applicable)
- The following terms are frequently used in any search algorithms:

- **State:** It provides all the information about the environment.
- **Goal State**: The desired resulting condition in a given problem and the kind of search algorithm we are looking for.
- **Goal Test:** The test to determine whether a particular state is a goal state.
- **Path/Step Cost:** These are integers that represent the cost to move from one node to another node.

- To evaluate and compare the efficiency of any search algorithm, the following 4 properties are used:

- **Completeness:** A search algorithm is said to be complete if it guarantees to return a solution, if exist.
- **Optimality/Admissibility:** If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.
- **Space Complexity:** A function describing the amount of space(memory) an algorithm takes in terms of input to the algorithm. That is how much space is used by the algorithm? Usually measured in terms of the maximum number of nodes in memory at a time.
- **Time Complexity:** A function describing the amount of time the algorithm takes in terms of input to the algorithm. That is, how long (worst or average case) does it take to find a solution?
- Time and space complexity are measured in terms of:'b' – maximum branching factor (Max number of successor (child) of any node) in a tree, 'd' – the depth of the shallowest goal node, and 'm' – maximum depth of the search tree (maybe infinity).
- The following table summarizes the 4 properties(**Completeness, Optimality/Admissibility,SpaceComplexity, Time Complexity)** of the search algorithm.

	BFS	DFS	IDDFS	Bidirectional Search (if Applicable)
Time	b^d	b^d	b^d	$b^{d/2}$
Space	b^d	bm	bd	$b^{d/2}$
Optimum?	Yes	No	Yes	Yes
Complete?	Yes	No	Yes	Yes

- The advantage of DFS is it requires very little memory as compared to BFS, as it only needs to store a stack of the nodes on the path from the root node to the current node. The disadvantages of DFS are as follows: There is the possibility that many states keep reoccurring, and there is no guarantee of finding the solution. The DFS algorithm goes for deep down searching and sometimes it may go to the infinite loop.
- On the other hand, BFS gives a guarantee to get an optimal solution, if any solution exists (Completeness) and if there is more than one solution for a given problem, then BFS will provide the minimal solution which requires the least number of steps (Optimal), but one major drawback of BFS is that it requires lots of memory space since each level of the tree must be saved into memory to expand the next level.
- Iterative Deepening DFS (IDDFS) combines the benefits of both BFS and DFS search algorithms in terms of fast search and memory efficiency. It is better than DFS and needs less space than BFS. But the main drawback of IDDFS is that it repeats all the work from the previous phase.
- The advantage of Bidirectional search (BS) is that it uses various techniques like DFS, BFS, DLS, etc, so it is efficient and requires less memory. The Implementation of the bidirectional search tree is difficult and in bidirectional search, one should know the goal state in advance.
- In Informed search the domain dependent (heuristic) information is used in order to search the space more efficiently.
- Informed searched include the following search:
- **Best-first search:** Order agenda based on some measure of how 'good' each state is.
- **Uniform-cost search**: Cost of getting to current state from initial state = g(n)
- **Greedy search:** Estimated cost of reaching goal from current state – Heuristic evaluation functions, h(n)
- **A* search:** f(n) = g(n) + h(n)
- Admissibility: h(n)never overestimates the actual cost of getting to the goal state.
- Informed Ness: A search strategy which searches less of the statesperson order to find a goal state is more informed.
- A* algorithm avoid expanding paths that are already expensive.
- In A*, Evaluation function f(n) = g(n) + h(n), where g(n) = cost so far to reach n, h(n) = estimated cost to goal from n, f(n) = estimated total cost of path through n to goal.
- A∗ search uses an admissible heuristic function, i.e., $h(n) \leq h*(n)$ where $h*(n)$ is the true cost of cheapest solution from n.
- A∗ search has a very good property: A∗ search is optimal! So, if there is any solution, A∗ search is guaranteed to find a least cost solution. Remember, this needs an admissible heuristic.

- The commonly used memory bound heuristics search are Iterative deepening A* (IDA*), Recursive Best-First search (RBFS) and Memory bound A* (MBA*).
- Iterative Deepening Search (IDS) is nothing but BFS plus DFS for tree search. IDA* algorithm is "complete and Optimal" algorithm.
- The idea of recursive best first search is to simulate **A*** search with **O(bd)** memory, where b is the branching factor and d is the solution depth.

3.13 Solutions/Answers

Check your progress 1:

Answer 1:

Table 1: Open and closed list for BFS

OPEN LIST	CLOSED LIST
1	1
2,3,4	2
3,4,6	3
4,6,5	4
6,5,7	5
5,7	6
7,	7
8-Goal State	-

Table 2: Open and closed list for DFS

OPEN LIST	CLOSED LIST
1	1
4,3,2	2
4,3,6	6
4,3	3
4,5	5
4,7	7
4,8-Goal State	-

Answer 2

1. **open = [A]; closed = [ ]** B is not the goal.
2. **open = [B,C,D]; closed = [A]** Put his children onto the queue.
3. **open = [C,D,E,F}; closed = [B,A]** Put him in closed. He is done.
4. **open = [D,E,F,G,H]; closed =[C,B,A]** Items between red bard are siblings.
5. **open = [E,F,G,H,I,J]; closed = [D,C,B,A]**

6. **open = [F,G,H,I,J,K,L]; closed = [E,D,C,B,A]**
7. **open = [G,H,I,J,K,L,M,N]** (as L is already on open);**closed = [G,F,E,D,C,B,A]**
8. **open = [H,I,J,K,L,M,N]; closed = [G,F,E,D,C,B,A]**
9. and so on until either goal is reached or open is empty.

Answer 3: Refer BFS section for solution.

Answer 4

	BFS	**DFS**	**IDDFS**	**Bidirectional Search (if Applicable)**
Time	b^d	b^d	b^d	$b^{d/2}$
Space	b^d	bm	bd	$b^{d/2}$
Optimum?	Yes	No	Yes	Yes
Complete?	Yes	No	Yes	Yes

Check your progress 2

Answer 1:
The OPEN and CLOSED list are shown below. Node with their f(n) values are inserted in OPEN list and that node will be expended next whose f(n) value is minimum.

CLOSED

$1^{(12)}$	$2^{(12)}$	$6^{(12)}$	$5^{(13)}$	$10^{(13)}$	$11^{(13)}$	$12^{(13)}$

OPEN

$1^{(12)}$			
$2^{(12)}$	$5^{(13)}$		
$5^{(13)}$	$3^{(14)}$	$6^{(12)}$	
$5^{(13)}$	$3^{(14)}$	$7^{(17)}$	$10^{(13)}$
$3^{(19)}$	$7^{(17)}$	$10^{(13)}$	
$3^{(19)}$	$7^{(17)}$	$10^{(13)}$	$9^{(14)}$
$3^{(19)}$	$7^{(17)}$	$9^{(14)}$	$11^{(13)}$
$3^{(19)}$	$7^{(17)}$	$9^{(14)}$	$12^{(13)}$

Note that only 6 nodes are expended to reach to a goal node. Optimal cost to reach from start state (1) to goal node (12) is **13**.

Note: If all the edge cost is positive then **Uniform cost search** (UCS) algorithm is same as Dijkstra's algorithm. Dijkstra algorithm fails if graph is having a negative weight cycle. A* algorithm allows negative weight also. It means A* algorithm work for negative (-ve) edge cost also. If some edge cost is

negative, then at any point of successive iteration, we cannot say till that node we have optimum cost (because of the negative cost). So, in this case (-ve edge cost), nodes come back from CLOSED to OPEN. Let us see one example (Example 2) having negative edge cost and you can also see how nodes come back from CLOSED to OPEN.

Answer 2:
The OPEN and CLOSED list are shown below. Node with their f(n) values are inserted in OPEN list and that node will be expended next whose f(n) value is minimum.

CLOSED

$1^{(15)}$ $2^{(7)}$ ~~$4^{(9)}$~~ ~~$5^{(11)}$~~ $3^{(25)}$ $4^{(7)}$ $5^{(9)}$

OPEN

$1^{(15)}$	
$2^{(7)}$	$3^{(25)}$
$3^{(25)}$	$4^{(9)}$
$3^{(25)}$	$5^{(11)}$
$3^{(25)}$	$6^{(28)}$
$4^{(7)}$	$6^{(28)}$
$5^{(9)}$	$6^{(28)}$
$6^{(26)}$	$6^{(28)}$

This is goal Node, but we can't Pick this because better cost/path may exist.
(Note: In open $6^{(28)}$ is Not min. cost ∴ we do not pick them).

Cost of 4 is decreased from 9 →7 so bring 4 from close →open
∴$4^{(9)}$

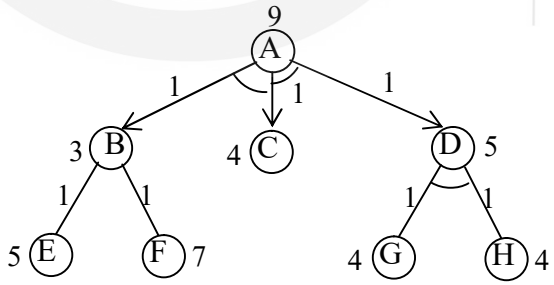
Same Logic

Goal (optimal cost is 26)

Optimal cost to reach from start state (1) to goal node (6) is **26**.

Answer 3: An A* algorithm represents an OR graph algorithm that is used to find a single solution (either this or that), but an AO* algorithm represents an AND-OR graph algorithm that is used to find more than one solution by ANDing more than one branch.

Answer 4 :

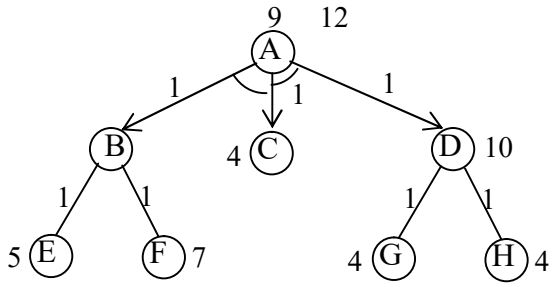


Path – 1: f(A-B-C) = 1+1+3+4=9
f(B-E) = 1+5=6 f(B-F) = 1+7=8
f(A-B-C) = 1+1+6+4=12

Path-2 : f(A-C-D) = 1+1+4+5=11
f(D-G-H) = 1+1+4+4=10

f(A-C-D)= 1+1+4+10=16

AND-OR graph with revised cost is shown in figure.



So, the optimal cost f(A-B-C)= 12

Note that AO* algorithm does not explore all the solution path once it finds a solution.

Answer 5: Similar to Q.4

Answer 6:

Given a chain of matrices: $A1, A2, \ldots\ldots, An$, where each matrix A_i has a dimension $p_{i-1} \times p_i$. Problem is to determine the order of multiplication or the way of parenthesizing the product of matrices, so that the minimum number of required operations (i.e., multiplications) can be minimized.

There are only 2 ways to parenthesizing the given 3 matrices, A1, A2, A3:

$A1 \times (A2 \times A3)$

or

$(A1 \times A2) \times A3$

As we know, if matrix A is of dimension $(p \times q)$ and matrix B is of dimension $(q \times r)$, then the cost of multiplying A to B that is $(A \times B)$ is $(p \times q \times r)$ and the final dimension of the resultant matrix $(A \times B)$ is $(p \times r)$.

Let us see how AND-OR graph is used to get the solution of this problem.

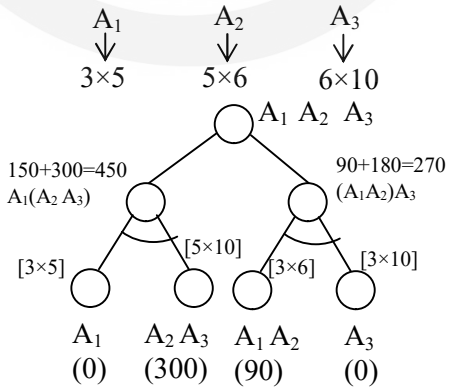


Figure-1 AND/OR graph for multiplying $A1 \times A2 \times A3$

In this AND-OR graph, parent (root) node indicates the given problem for multiplying A1A2A3. Next level of the tree (2 successors node) indicates the 2 choices (or ways) of multiplying (or

parenthesizing) the A1A2A3; first way is $A1 \times (A2 \times A3)$ and another way is $(A1 \times A2) \times A3$. Since out of these two choices, anyone will be the solution so there is a OR node for this. In an OR node, we always mark the current best successor node. Next level we have an AND node. For any AND node we must add the cost of both the successor node.
Cost of multiplying $(A2 \times A3) = 5 \times 6 \times 10 = 300$ and dimension of $A2 \times A3$ is (5×10). Since the dimension of A1 is (3×5) and the dimension of $A2 \times A3$ is (5×10), so the cost of multiplying $A1 \times (A2 \times A3) = 3 \times 5 \times 10 = 150$. Thus the total cost will be 300+150=450. Similarly,
The cost of multiplying $(A1 \times A2) = 3 \times 5 \times 6 = 90$ and dimension of $A1 \times A2$ will be (3×6). Since the dimension of $A1 \times A2$ is (3×6) and the dimension of A3 is (6×10), so the cost of multiplying $(A1 \times A2) \times A3 = 3 \times 6 \times 10 = 180$. Thus the total cost will be 90+180=270. So, the best way to multiplying $A1 \times A2 \times A3$ is $(A1 \times A2) \times A3$ and the minimum cost of multiplying $A1 \times A2 \times A3$
is 270.

Multiple Choice Questions

Answer 6: Option C

Answer 7: Option A

3.14 FURTHER READINGS

1. Ela Kumar, " Artificial Intelligence", IK International Publications
2. E. Rich and K. Knight, "Artificial intelligence", Tata Mc Graw Hill Publications
3. N.J. Nilsson, "Principles of AI", Narosa Publ. House Publications
4. John J. Craig, "Introduction to Robotics", Addison Wesley publication
5. D.W. Patterson, "Introduction to AI and Expert Systems" Pearson publication

s

UNIT 4 PREDICATE ANDPROPOSITIONAL LOGIC

4.1 INTRODUCTION

Logic is the study and analysis of the nature of the valid argument, the reasoning tool by which valid inferences can be drawn from a given set of facts and premises. It is the basis on which all the sciences are built, and this mathematical theory of logic is called symbolic logic. The English mathematician George Boole (1815-1864) seriously studied and developed this theory, called symbolic logic.

The reason why the subject-matter of the study is called Symbolic Logic is that symbols are used to denote facts about objects of the domain and relationships between these objects. Then the symbolic representations and not the original facts and relationships are manipulated in order to make conclusions or to solve problems.

The basic building blocks of arguments in symbolic logic are declarative sentences, called propositions or statements. In MCS – 212 i.e., Discrete Mathematics you learned about predicates and propositions and ways of combining them to form more complex propositions. Also, you learned about the propositions that contain the quantifiers 'for All' and 'there exists'. In symbolic logic, the goal is to determine which propositions are true and which are false. Truth table a tool to find out all possible outcome of a proposition's truth value was also discussed in MCS-212.

Logical rules have the power to give accuracy to mathematical statements. These rules come to rescue when there is a need to differentiate between valid and invalid mathematical arguments. Symbolic logic may be thought of as a formal language for representing facts about objects and relationships between objects of a problem domain along with a precise inferencing mechanism for reasoning and deduction.

Using symbolic logic, we can formalize our arguments and logical reasoning in a manner that can easily show if the reasoning is valid, or is a fallacy. How we symbolize the reasoning is what is presented in this unit.

4.2 OBJECTIVES

After going through this unit, you should be able to:

1. Understand the meaning of propositional logic.
2. Differentiate between atomic and compound propositions.
3. Know different types of connectives, their associated semantics and corresponding truth tables.
4. Define propositional rules of inference and replacement.
5. Differentiate between valid and satisfiable arguments.

4.3 INTRODUCTION TO PROPOSITIONAL LOGIC

Apart from the application of logic in mathematics, it also helps in various other tasks related to computer science. It is widely used to design the electronic circuitry, programming of android applications, applying artificial intelligence to different tasks, etc. In simple terms, a proposition is a statement which is either true or false.

Consider the following statements:

1. Earth revolves around the sun.
2. Water freezes at 100° Celsius.
3. An hour has 3600 seconds.
4. 2 is the only even prime number.
5. Mercury is the closest planet to the Sun in the solar system.
6. The USA lies on the continent of North America.
7. $1 + 2 = 4$.
8. 15 is a prime number.
9. Moon rises in the morning and sets in the evening.
10. Delhi is the capital of India.

For all the above statements, one can easily conclude whether the particular statement is true or false so these are propositions. First statement is a universal truth. Second statement is false as the water freezes at 0° Celsius. Third statement is again a universal truth. Fourth statement is true. Fifth statement is also true as it is again a universal truth. On similar lines, the sixth statement is also true. Seventh and eighth statements are false again as they deny the basic mathematical rules. The Ninth statement is a negation of the universal truth so it is a false statement. The Tenth Statement is also true.

Now consider the following statements:

1. What is your name?
2. a + 5 = b.
3. Who is the prime minister of India?
4. p is less than 5.
5. Pay full attention while you are in the class.
6. Let's play football in the evening.
7. Don't behave like a child, you are grown up now!
8. How much do you earn?
9. ∠X is an acute angle greater than 27°.

For all the above statements, we can't say anything about their truthfulness so they are not propositions. First and third statements are neither true nor false as they are interrogative in nature. Also, we can't say anything about the truthfulness of the second statement until and unless we have the values of a and b. Similar reasoning applies to the fourth statement as well. We can't say anything about fifth, sixth and seventh statements again as they are informative statements. Eighth statement is again interrogative in nature. Again, we can't say anything about the truthfulness of the ninth statement until and unless we have the value of ∠X.

Propositional logic has the following facts:

1. Propositional statements can be either true or false, they can't be both simultaneously.
2. Propositional logic is also referred to as binary logic as it works only on the two values 1 (True) and 0 (False).

3. Symbols or symbolic variables such as x, y, z, P, Q, R, etc. are used for representing the logic and propositions.
4. Any proposition or statement which is always valid (true) is known as a tautology.
5. Any proposition or statement which is always invalid (false) is known as a contradiction.
6. A table listing all the possible truth values of a proposition is known as a truth table.
7. Objects, relations (or functions) and logical connectives are the basic building blocks of propositional logic.
8. Logical connectives are also referred to as logical operators.
9. Statements which are interrogative, informative or opinions such as "Where is Chandni Chowk located?", "Mumbai is a good city to live in", "Result will be declared on 31st March" are not propositions.

4.4 SYNTAX OF PROPOSITIONAL LOGIC

The syntax of propositional logic allows two types of sentences to represent knowledge. The two types are as follows:

4.4.1 Atomic Propositions

These are simplest propositions containing a single proposition symbol and are either true or false. Some of the examples of atomic propositions are as follows:

1. "Venus is the closest planet to the Sun in the solar system" is an atomic preposition since it is a false fact.
2. "7 – 3 = 4" is an atomic preposition as it is a true fact.

4.4.2 Compound Propositions

They are formed by a collection of atomic propositions joined with logical connectives or logical operators. Some of the examples of compound propositions are as follows:

1. The Sun is very bright today and its very hot outside.
2. Diana studies in class 8th and her school is in Karol Bagh.

☞ **Check Your Progress 1**

Which of the following statements are propositions? Write yes or no.

1. How are you?
2. Sachin Tendulkar is one of the best cricketers in India.
3. The honorable Ram Nath Kovind is the 10th and current president of India.
4. Lord Ram of the kingdom of Ayodhya is an example of a people's king.
5. In which year did prophet Muhammad received verbal revelations from the Allah in the cave Mount Hira presently located in the Saudi Arabia?
6. Akbar was the founder of the Mughal dynasty in India.
7. In the year 2019, renowned actor Shri Amitabh Bachchan was awarded with the Padma Vibhushan which is the second highest civilian honour of the republic of India.
8. One should avoid eating fast food in order to maintain good health.
9. What is your age?
10. The first case of COVID-19 was detected in China.
11. Name the author of the book series “Shiva Trilogy”.
12. Former prime minister of India, Shri Atal Bihari Vajpayee was a member of which political party?
13. Wing Commander Rakesh Sharma is the only Indian citizen to travel in space till date.
14. Where do you live?

4.5 LOGICAL CONNECTIVES

Logical connectives are the operators used to join two or more atomic propositions (operands). The joining should be done in a way that the logic and truth value of the obtained compound proposition is dependent on the input atomic propositions and the connective used.

4.5.1 Conjunction

A proposition "A ∧ B"with connective ∧ is known as *conjunction* of A and B. It is a proposition (or operation) which is true only when both the constituent propositions are true. Even if one of the input propositions is false then the output is also false. It is also referred to as AND-ing the propositions. Example:

Ram is a playful boy and he loves to play football. It can be written as:

A = Ram is a playful boy.

B = Ram loves to play football.

A ∧ B = Ram is a playful boy and he loves to play football.

4.5.2 Disjunction

A proposition "A ∨ B"with connective ∨ is known as *disjunction* of A and B. It is a proposition (or operation) which is true when at least one of the constituent propositions are true. The output is false only when both the input propositions are false. It is also referred to as OR-ing the propositions. Example:

I will go to her house or she will come to my house. It can be written as:

A = I will go to her house.

B = She will come to my house.

A ∨ B = I will go to her house or she will come to my house.

4.5.3 Negation

The proposition ¬ A (or ~A) with ¬ (or ~) connective is known as *negation* of A. The purpose of negation is to negate the logic of given proposition. If A is true, its negation will be false, and if A is false, its negation will be true. Example:

University is closed. It can be written as:

A = University is closed.

¬ A = University is not closed.

4.5.4 Implication

The proposition A → B with → connective is known as *A implies B*. It is also called *if-then* proposition. Here, the second proposition is a logical consequence of the first proposition. For example, "If Mary scores good in examinations, I will buy a mobile phone for her". In this case, it means that if Mary scores good, she will definitely get the mobile phone but it doesn't mean that if she performs bad, she won't get the mobile phone. In set notation, we can also say that A ⊆ B i.e., if something exists in the set A, then it necessarily exists in the set B. Another example:

If you score above 90%, you will get a mobile phone.

A = You score above 90%.

B = You will get a mobile phone.

A → B = If you score above 90%, you will get a mobile phone.

4.5.5 Bi-conditional

A proposition A ↔ B with connective ↔ is known as a *biconditional* or *if-and-only-if* proposition. It is true when both the atomic propositions are true or both are false. A classic example of biconditional is "A triangle is equivalent if and only if all its angles are60° each". This statement means that if a triangle is an equivalent triangle, then all of its angles are 60° each. There is one more associated meaning with this statement which means that if all the interior angles of a triangle are of 60° each then it's an equivalent triangle. Example:

You will succeed in life if and only if you work hard.

A = You will succeed in life.

B = You work hard.

A ↔ B = You will succeed in life if and only if you work hard.

☞ Check Your Progress 2

Which of the following propositions are atomic and which are compound?

1. The first battle of Panipat was fought in 1556.
2. Jack either plays cricket or football.

3. Posthumously, at the age of 22, Neerja Bhanot became the youngest recipient of the Ashok Chakra award which is India's highest peacetime gallantry decoration.

4. Chandigarh is the capital of the Indian states Haryana and Punjab.

5. Earth takes 365 days, 5 hours, 59 minutes and 16 seconds to complete one revolution around the Sun.

6. Dermatology is the branch of medical science which deals with the skin.

7. Indian sportspersons won 7 medals at the 2020 Summer Olympics and 19 medals at the 2020 Summer Paralympics both held at the Japanese city of Tokyo.

8. Harappan civilization is considered to be the oldest human civilization and it lies in the parts of present-day India, Pakistan and Afghanistan.

9. IGNOU is a central university and offers courses through the distance learning mode.

10. Uttarakhand was carved out of the Indian state of Uttar Pradesh in the year 2000.

4.6 SEMANTICS

You had already learned various of the concepts to be covered in this unit, in MCS-212 i.e., Discrete Mathematics, here is a quick revision to those concepts and we will extend our discussion to the advanced concepts, which are useful for our field of work i.e., Artificial Intelligence. In MCS – 212 i.e., Discrete Mathematics you learned that **Propositions are the** declarative sentence or statements which is either true or false, but not both, such sentences can either be **universally true** or **universally false**.

On the other hand, consider the declarative sentence 'Women are more intelligent than men'. Some people may think it is true while others may disagree. So, it is neither universally true nor universally false. Such a sentence is not acceptable as a statement or proposition in mathematical logic.

Note that a **proposition should be either uniformly true or uniformly false**.

In propositional logic, as mentioned earlier also, symbols are used to denote propositions. For instance, we may denote the propositions discussed above as follows:

P : The sun rises in the west,
Q : Sugar is sweet,
R : Ram has a Ph.D. degree.

The symbols, such as P, Q, and R, that are used to denote propositions, are called **atomic formulas, or atoms.,** in this case, the truth-value of P is False, the truth-value of Q is True and the truth-value of R, though not known yet, is exactly one of 'True' or 'False', depending on whether Ram is actually a Ph. D or not.

At this stage, it may be noted that once symbols are used in place of given statements in, say, English, then the propositional system, and, in general, a symbolic system is aware **only** of symbolic representations, and the associated truth values. The system operates only on these representations. And, except for possible final translation, is **not aware** of the original statements, generally given in some natural language, say, English.

When you're talking to someone, do you use very simple sentences only? Don't you use more complicated ones which are joined by words like 'and', 'or', etc? In the same way, most statements in mathematical logic are combinations of simpler statements joined by words and phrases like 'and'. 'or', 'if … then'. 'If and only if', etc. We can build, from atoms, *more complex propositions*, sometimes called***compound propositions***, by using logical **connectives,**

The Logical Connectives are used to frame compound propositions, and they are as follows:

a) Disjunction The **disjunction** of two propositions p and q is the compound statement **por q,** denoted by $\mathbf{p \vee q}$.

The **exclusive disjunction** of two propositions p and q is the statement **'Either of the two (i.e. p or q) can be true, but both can't be true'.** We denote this by $\mathbf{p \oplus q}$ **.**

b) Conjunction We call the compound statement **'p and q'** the **conjunction** of the statements p and q. We denote this by $\mathbf{p \wedge q}$.

c) Negation The **negation** of a proposition p is **'not p'**, denoted by ~p.

d) Implication (Conditional Connectives) Given any two propositions p and q, we denote the statement **'If p, then q' by** $\mathbf{p \rightarrow q}$**.** We also read this as 'p **implies** q'. or 'p is sufficient for q', or 'p **only if** q'. We also call p the **hypothesis** and q the conclusion. Further, a statement of the form $p \rightarrow q$ is called a **conditional statement** or a **conditional proposition**.

Let p and q be two propositions. The compound statement $(p \rightarrow q) \wedge (q \rightarrow p)$ is the **bi-conditional** of p and q. We denote it by $\mathbf{p \leftrightarrow q}$**,** and read it as 'p **if and only** q'

Note : The two connectives $\rightarrow$ and $\leftrightarrow$ are called **conditional connectives**

The rule of precedence: The order of preference in which the connectives are applied in a formula of propositions that has no brackets is

i) $\sim$
ii) $\wedge$

iii) $\vee$ and $\oplus$

iv) $\rightarrow$ and $\leftrightarrow$

Note that the 'inclusive or' and 'exclusive or' are both third in the order of preference. However, if both these appear in a statement, we first apply the left most one. So, for

instance, in $p \vee q \oplus \sim p$, we first apply $\vee$ and then $\oplus$. The same applies to the 'implication' and the 'biconditional', which are both fourth in the order of preference.

Let's see the working of the various concepts learned above, with the help of truth tables. In the following truth table, we write every TRUE value as Tand every FALSE value as F.

4.6.1 Negation Truth Table

α	$\sim \alpha$
_F(0)_	_T(1)_
_T(1)_	_F(0)_

4.6.2 Conjunction/Disjunction/Implication /Biconditional Truth Table

α_1	α_2	Conjunction $\alpha\_1 \wedge \alpha\_2$	Disjunction $\alpha\_1 \vee \alpha\_2$	Implication $\alpha\_1 \rightarrow \alpha\_2$	Biconditional $\alpha\_1 \leftrightarrow \alpha\_2$
_F(0)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_
_F(0)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_
_T(1)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_F(0)_
_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_

4.6.3 Conjunction and Disjunction with three variables

α_1	α_2	α_3	$(\alpha\_1 \wedge \alpha\_2)$	$(\alpha\_2 \wedge \alpha\_3)$	$(\alpha\_1 \vee \alpha\_2)$	$(\alpha\_2 \vee \alpha\_3)$	$(\alpha\_1 \wedge \alpha\_2) \wedge \alpha\_3$ or $\alpha\_1 \wedge (\alpha\_2 \wedge \alpha\_3)$	$(\alpha\_1 \vee \alpha\_2) \vee \alpha\_3$ or $\alpha\_1 \vee (\alpha\_2 \vee \alpha\_3)$
_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_
_T(1)_	_T(1)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_T(1)_	_F(0)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_T(1)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_
_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_T(1)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_F(0)_	_T(1)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_

Using these logical connectives, we can transform any sentence in to its equivalent mathematical representation in Symbolic Logic and that representation is referred as Well

Form Formula (WFF), you had already learned a lot about WFF in MCS-212, lets briefly discuss it here also, as it has wide applications in Artificial Intelligence also.

A Well-formed formula, or *wff* or *formula* in short, in the propositional logic is defined recursively as follows:

1. An atom is a wff.
2. If A is a wff, then (~A) is a wff.
3. If A and B are wffs, then each of $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, and $(A \leftrightarrow B)$ is a wff.
4. Any wff is obtained only by applying the above rules.

From the above recursive definition of a wff it is not difficult to see that expression:

$((P \rightarrow (Q \wedge (\sim R))) \leftrightarrow S)$ is a wff; because , to begin with, each of P, Q , $(\sim R)$ and S, by definitions is a wff. Then, by recursive application, the expression: $(Q \wedge (\sim R))$ is a wff. Again, by another recursive application, the expression: $(P \rightarrow (Q \wedge (\sim R)))$ is a wff. And, finally the expression given initially is a wff.

Further, it is easy to see that according to the recursive definition of a wff, each of the expressions: $(P \rightarrow (Q \wedge))$ and $(P (Q \wedge R))$ is **not** a wff.

Some pairs of parentheses may be dropped, for simplification. For example,

$A \vee B$ and $A \rightarrow B$ respectively may be used instead of the given wffs $(A \vee B)$ and $(A \rightarrow B)$, respectively. We can omit the use of parentheses by assigning *priorities in increasing order* to the connectives as follows:

$$\leftrightarrow, \quad \rightarrow, \quad \vee, \quad \vee, \quad \sim.$$

Thus, '$\leftrightarrow$' has least priority and '$\sim$' has highest priority. Further, if in an expression, there are no parentheses and two connectives between three atomic formulas are used, then the operator with higher priority will be applied first and the other operator will be applied later.

For example: Let us be given the wff $P \rightarrow Q \wedge \sim R$ without parenthesis. Then among the operators appearing in wff, the operator '~' has highest priority. Therefore, ~ R is replaced by (~R). The equivalent expression becomes $P \rightarrow Q \wedge (\sim R)$. Next, out of the two operators viz '$\rightarrow$' and '$\wedge$', the operators '$\wedge$' has higher priority. Therefore, by applying parentheses appropriately, the new expression becomes $P \rightarrow (Q \wedge (\sim R))$. Finally, only one operator is left. Hence the *fullyparenthesized expression* becomes $(P \rightarrow (Q \wedge (\sim R)))$

Following are the rules of finding the truth value or meaning of a wff, when truth values of the atoms appearing in the wff are known or given.

1. The wff ~ A is *True* when A is *False*, and ~ A is *False* when A is *true*. The wff ~ A is called the ***negation*** of A.

2. The wff (A $\wedge$ B) is True if A and B are both True; otherwise, the wff A $\wedge$ B is False. The wff (A $\wedge$ B) is called the **conjunction** of A and B.
3. The wff (A $\vee$ B) is true if at least one of A and B is True; otherwise, (A $\vee$ B) is False. (A $\vee$ B) is called the **disjunction** of A and B.
4. The wff (A $\rightarrow$ B) is False if A is True and B is False; otherwise, (A $\rightarrow$ B) is True. The wff (A $\rightarrow$ B) is read as "If A, then B," or "**A implies B**." The symbol '$\rightarrow$' is called **implication**.
5. The wff (A $\leftrightarrow$ B) is True whenever A and B have the same truth values; otherwise (A $\leftrightarrow$ B) is False. The wff (A $\leftrightarrow$ B) is read as "**A if and only if B**."

☞ **Check Your Progress 3**

Q1. Draw the truth table for the following:

a) $\alpha\_2 \leftrightarrow (\sim \alpha\_1 \rightarrow (\alpha\_1 \vee \alpha\_2))$

b) $(\sim\alpha\_1 \leftrightarrow (\alpha\_2 \leftrightarrow \alpha\_3)) \vee (\alpha\_3 \wedge \alpha\_2)$

c) $((\alpha\_1 \wedge \alpha\_2) \rightarrow \alpha\_3) \vee \sim\alpha\_4$

d) $((\alpha\_1 \rightarrow \sim \alpha\_2) \leftrightarrow \alpha\_3) \rightarrow \sim (\alpha\_1 \vee \alpha\_1)$

Q2. Verify the De Morgan's Laws using Truth Tables

Q3. Write WFF for the following statements:

a) Every Person has Mother

b) There is a woman and she is mother of Siya

4.7 PROPOSITIONAL RULES OF INFERENCE

We need intelligent computers based on the concept of artificial intelligence which are able to infer new "knowledge" or logic from the existing logic using the theory of inference. Inference rules help us to infer new propositions and conclusions based on existing propositions and logic. They act as templates to generate new arguments from the premises or predicates. We deduce new statements from the statements whose truthfulness is already known. These rules come to the rescue when we need to prove something logically. In general, inference rules preserve the truth. Depending on the problem, some or all of these rules may be applied to infer new propositions. The procedure of determining whether a proposition is a conclusion of the given propositions is known as inferring a proposition. The inference rules are described below.

4.7.1 Modus Ponens (MP)

It states that if the propositions A → B and A are true, then B is also true. Modus Ponens is also referred to as the implication elimination because it eliminates the implication A → B and results in only the proposition B. It also affirms the truthfulness of antecedent. It is written as:

$$\alpha\_1 \rightarrow \alpha\_2, \alpha\_1 \Rightarrow \alpha\_2$$

4.7.2 Modus Tollens (MT)

It states that if A → B and ¬ B are true then ¬ A is also true. Modus Tollens is also referred to as denying the consequent as it denies the truthfulness of the consequent. The rule is expressed as:

$$\alpha\_1 \rightarrow \alpha\_2, \sim\alpha\_2 \Rightarrow \sim\alpha\_1$$

4.7.3 Disjunctive Syllogism (DS)

Disjunctive Syllogism affirms the truthfulness of the other proposition if one of the propositions in a disjunction is false.

Rule 1: $\alpha\_1 \vee \alpha\_2, \sim\alpha\_1 \Rightarrow \alpha\_2$

Rule 2: $\alpha\_1 \vee \alpha\_2, \sim\alpha\_2 \Rightarrow \alpha\_1$

4.7.4 Addition

The rule states that if a proposition is true, then its disjunction with any other proposition is also true.

Rule 1: $\alpha\_1 \Rightarrow \alpha\_1 \vee \alpha\_2$

Rule 2: $\alpha\_2 \Rightarrow \alpha\_1 \vee \alpha\_2$

4.7.5 Simplification

Simplification means that if we have a conjunction, then both the constituent propositions are also true.

Rule 1: $\alpha\_1 \wedge \alpha\_2 \Rightarrow \alpha\_1$

Rule 2: $\alpha\_1 \wedge \alpha\_2 \Rightarrow \alpha\_2$

4.7.6 Conjunction

Conjunction states if two propositions are true, then their conjunction is also true. It is written as:

$$\alpha\_1, \alpha\_2 \Rightarrow \alpha\_1 \wedge \alpha\_2$$

4.7.7 Hypothetical Syllogism (HS)

The rule says that the conclusion $\alpha\_1 \rightarrow \alpha\_3$ is true, whenever conditional statements $\alpha\_1 \rightarrow \alpha\_2$ and $\alpha_2 \rightarrow \alpha_3$hold the truth values. This rules also shows the transitive nature of implication operator.

$$\alpha\_1 \rightarrow \alpha\_2, \alpha\_2 \rightarrow \alpha\_3 \Rightarrow \alpha\_1 \rightarrow \alpha\_3$$

4.7.8 Absorption

The rule states that if the literal α_1 conditionally implies another literal α_2 i.e.,$\alpha\_1 \rightarrow \alpha\_2$ is true, then $\alpha\_1 \rightarrow (\alpha\_1 \wedge \alpha\_2)$ also holds.

$$\alpha\_1 \rightarrow \alpha\_2 \Rightarrow \alpha\_1 \rightarrow (\alpha\_1 \wedge \alpha\_2)$$

4.7.9 Constructive Dilemma (CD)

According to the rule, if proposition $(\alpha\_1 \vee \alpha\_3)$ and proposition $((\alpha_1 \rightarrow \alpha_2) \wedge (\alpha_3 \rightarrow \alpha_4))$have true values, then the well-formed formula $(\alpha\_2 \vee \alpha\_4)$ also holdsa true value.

$$(\alpha\_1 \vee \alpha\_3), (\alpha\_1 \rightarrow \alpha\_2) \wedge (\alpha\_3 \rightarrow \alpha\_4) \Rightarrow \alpha\_2 \vee \alpha\_4$$

4.8 PROPOSITIONAL RULES OF REPLACEMENT

We learned the concepts of Predicate and Propositional logic in MCS-212 (Discrete Mathematics), just to brief the understanding, it is to remind you here that "***a proposition is a specialized statement whereas Predicate is a generalized statement***". To be more specific the propositions use the logical connectives only and the predicates uses logical connectives and quantifiers (universal and existential), both.

Note :$\exists$ is the symbol used for the Existential quantifier and $\forall$ is used for the Universal quantifier.

In predicate logic, a replacement rule is used to replace an argument or a set of arguments with an equivalent argument. By equivalent arguments, we mean that the logical interpretation of the arguments is the same. These rules are used to manipulate the propositions. Also, the axioms and the propositional rules of inference are used as an aid to generate the replacement rules. Given below is the table summarizing the different replacement rules over the propositions α_1, α_2 and α_3.

Replacement Rule	Proposition	Equivalent
Tautology (Conjunction of a statement with itself always implies the statement)	α_1 ∧α_1	α_1
Double Negation (DN) (Also Called negation of negation)	(~(~α_1))	α_1
Commutativity (Valid for Conjunction and Disjunction)	α_1 ∧α_2	α_2 ∧α_1
	α_1 ∨α_2	α_1 ∨α_2
Associativity (Valid for Conjunction and Disjunction)	(α_1 ∧α_2) ∧α_3	α_1 ∧ (α_2 ∧α_3)
	(α_1 ∨α_2) ∨α_3	α_1 ∨ (α_2 ∨α_3)
DeMorgan's Laws	~ (α_1 ∧α_2)	(~α_1) ∨ (~α_2)
	~ (α_1 ∨α_2)	(~α_1) ∧ (~α_2)
Transposition (Defined over implication)	α_1 → α_2	(~α_2) → (~α_1)
Exportation	α_1 → (α_2 → α_3)	(α_1 ∧α_2) → α_3
Distribution (AND over OR, and OR over AND)	α_1 ∧ (α_2 ∨α_3)	(α_1 ∧α_2) ∨ (α_1 ∧α_3)
	α_1 ∨ (α_2 ∧α_3)	(α_1 ∨α_2) ∧ (α_1 ∨α_3)
Material Implication (MI) (Defined over implication)	α_1 → α_2	~ α_1 ∨α_2
Material Equivalence (ME)(Defined over biconditional)	α_1↔α_2	(α_1 → α_2) ∧ (α_2 → α_1)

4.9 VALIDITY AND SATISFIABILITY

An argument is called *valid* in propositional logic if the argument is a tautology for all possible combinations of the given premises. An argument is called *satisfiable* if the argument is true for at least one combination of the premises.

Consider the example given below. Kindly note, every TRUE value is written as _T(1)_, and every FALSE value is written as _F(0)_.

Example 1: Consider the expression α_1 ∨ (α_2 ∨α_3) ∨ (~α_2 ∧ ~ α_3), whose truth table is given below.

α_1	α_2	α_3	~α_2	~α_3	α_1 ∨α_2	~α_2 ∧ ~α_3	α_1 ∨ (α_2 ∨α_3) ∨ (~α_2 ∧ ~α_3)
_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_
_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_T(1)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_
_T(1)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_
_T(1)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_
_T(1)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_T(1)_	_T(1)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_

It is noteworthy from the above truth table that the argument α_1 ∨ (α_2 ∨α_3) ∨ (~α_2 ∧ ~α_3) is a valid argument as it has true values for all possible combinations of the premises α_1, α_2 and α_3.

Example 2: Consider the expression α_1 ∧ ((α_2 ∧α_3) ∨ (α_1 ∧α_3)) for the

α_1	α_2	α_3	α_1 ∧α_3	α_2 ∧α_3	(α_2 ∧α_3) ∨ (α_1 ∧α_3)	α_1 ∧ ((α_2 ∧α_3) ∨ (α_1 ∧α_3))
_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_
_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_F(0)_
_F(0)_	_T(1)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_
_F(0)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_F(0)_
_T(1)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_
_T(1)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_
_T(1)_	_T(1)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_
_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_

propositions α_1,α_2, and α_3whose truth table is given below.

In this example, as the argument α_1 ∧ ((α_2 ∧α_3) ∨ (α_1 ∧α_3)) is true for a few combinations of the premises α_1, α_2, and α_3, hence it is a satisfiable argument.

4.10 INTRODUCTION TO PREDICATE LOGIC

Now it's time to understand the difference between the Proposition and the Predicate(also known as propositional function). In short, a proposition is a specialized statement whereas Predicate is a generalized statement. To be more specific the propositions use the logical connectives only and the predicates uses logical connectives and quantifiers (universal and existential), both.

Note :∃ is the symbol used for the Existential quantifier and ∀ is used for the Universal quantifier.

Let's understand the difference through some more detail, as given below.

A propositional function, or a **predicate,** a variable x in a sentence p(x) involving x becomes a proposition when we give x a definite value from the set of values it can take. We usually denote such functions by p(x), q(x), etc. The set of values x can take is called the universe of discourse.

So, if p(x) is 'x > 5', then p(x) is not a proposition. But when we give x particular values, say x = 6 or x = 0, then we get propositions. Here, p(6) is a true proposition and p(0) is a false proposition.

Similarly, if q(x) is 'x has gone to Patna.', then replacing x by 'Taj Mahal' gives us a false proposition.

Note that a predicate is usually not a proposition. But, of course, every proposition is a prepositional function in the same way that every real number is a real-valued function, namely, the constant function.

Now, can all sentences be written in symbolic from by using only the logical connectives? What about sentences like 'x is prime and x + 1 is prime for some x.'? How would you symbolize the phrase 'for some x', which we can rephrase as 'there exists an x'? You must have come across this term often while studying mathematics. **We use the symbol '∃' to denote this quantifier, 'there exists'.** The way we use it is, for instance, to rewrite 'There is at least one child in the class.' as'(∃ x in U)p(x)',

where p(x) is the sentence 'x is in the class.' and U is the set of all children.

Now suppose we take the negative of the proposition we have just stated. Wouldn't it be 'There is no child in the class.'? We could symbolize this as 'for all x in U, q(x)' where x ranges over all children and q(x) denotes the sentence 'x is not in the class.', i.e., q(x) ≡ ~ p(x).

We have a **mathematical symbol for the quantifier 'for all', which is '∀'.**So, the proposition above can be written as

'$(\forall x \in U)q(x)$', or '$q(x), \forall x \in U$'.

An example of the use of the existential quantifier is the true statement.

$(\exists x \in \mathbf{R})\ (x + 1 > 0)$, which is read as 'There exists an x in **R** for which $x + 1 > 0$.'.

Another example is the false statement

$(\exists x \in \mathbf{N})\ (x - \frac{1}{2} = 0)$, which is read as 'There exists an x in **N** for which $x - \frac{1}{2} = 0$.'.

An example of the use of the universal quantifier is $(\forall x \notin \mathbf{N})\ (x^2 > x)$, which is read as 'for every x not in **N**, $x^2 > x$.'. Of course, this is a false statement, because there is at least one $x \notin \mathbf{N}$, $x \in \mathbf{R}$, for which it is false.

As you have already read in the example of a child in the class,

$(\forall x \in U)p(x)$ is logically equivalent to $\sim (\exists x \in U)\ (\sim p(x))$. Therefore,

$\sim(\forall x \in U)p(x) \equiv \sim\sim (\exists x \in U)\ (\sim p(x)) \equiv (\exists x \in U)\ (\sim p(x))$.

This is one of the rules for negation that relate $\forall$ and $\exists$. The two rules are

$\sim (\forall \mathbf{x} \in \mathbf{U})\mathbf{p(x)} \equiv (\exists x \in \mathbf{U})\ \mathbf{(\sim p(x))}$, and

$\sim (\exists \mathbf{x} \in \mathbf{U})\mathbf{p(x)} \equiv (\forall x \in \mathbf{U})\ \mathbf{(\sim p(x))}$

Where U is the set of values that x can take.

To Sum up "***a proposition is a specialized statement whereas Predicate is a generalized statement***". To be more specific the propositions use the logical connectives only and the predicates uses logical connectives and quantifiers (universal and existential), both.

Note : $\exists$ is the symbol used for the Existential quantifier and $\forall$ is used for the Universal quantifier.

By interpretation of symbolic logic, we mean assigning the meaning to the symbols of any formal language. By default, the statements do not have any meaning associated with them but when we assign the symbols to them, their meaning is automatically associated with them i.e., TRUE (T) or FALSE (F).

Example 1: Consider the following statements

a) Weather is not hot today and it is windy than yesterday.
b) Kids will go to park only if it is not hot.
c) If kids do not go to park, then they can play near pool.
d) If kids play near pool, then we can offer them a juice.

Let the statements be represented using literals as following:

H: Weather is not hot today
W: It is windy than yesterday
K: Kids will go to park
P: They can play near pool.
J: We can offer them juice.

The given statements can be interpreted as:

a) H ∧W
b) K -> ~H
c) ~K -> P
d) P -> J

These interpretations may be verified with the help of truth tables.

4.11 INFERENCING IN PREDICATE LOGIC

In general, we are given a set of arguments in predicate logic. Now, using the rules of inference, we can deduce other arguments (or predicates) based on the given arguments (or predicates). This process is known as entailment as we entail new arguments (or predicates). The inference rules you learned in MCS-212 and also in section 4.7 above, of this unit are also applicable here for the process of entailment or making inferences. Now, with the help of the following example we will learn how the rules of inference, discussed above, can be used to solve the problems.

Example : There is a village that consists of two types of people – those who always tell the truth, and those who always lie. Suppose that you visit the village and two villagers A and B come up to you. Further, suppose

A says, "B always tells the truth," and

B says, "A and I are of opposite types".

What types are A and B ?

Solution: Let us start by assuming A is a truth-teller.

∴ What A says is true.

∴ B is a truth-teller.

∴ What B says is true.

∴ A and B are of opposite types.

This is a contradiction, because our premises say that A and B are both truth-tellers.

∴ The assumption we started with is false.

∴ A always tells lies.

∴ What A has told you is lie.

∴ B always tells lies.

∴ A and B are of the same type, i.e., both of them always lie.

Let us now consider the problem of showing that a statement is false. I.e.,**Counter examples :** A common situation in which we look for counterexamples is to disprove statements of the form $p \rightarrow q$ needs to be an example where $p \wedge \sim q$. Therefore, a counterexample to $p \rightarrow q$ needs to be an example where $p \wedge \sim q$ is true, i.e., p is true and $\sim q$ is true, i.e., the hypothesis p holds but the conclusion q does not hold.

For instance, to disprove the statement 'If n is an odd integer, then n is prime.', we need to look for an odd integer which is not a prime number. 15 is one such integer. So, n = 15 is a counterexample to the given statement.

Notice that a **counter example to a statement p proves that** p is false, i.e., $\sim$ **p is true**.

Example: Following statements are to be Symbolized and thereafter construct a proof for the following valid argument:

(i) If the BOOK_X is literally true, then the Earth was made in six days.
(ii) If the Earth was made in six days, then carbon dating is useless and Scientists/Researchers are liars.
(iii) Scientists/Researchers are not liars.
(iv) The BOOK_X is literally true, Hence
(v) God does not exist.

Solution: Let us symbolize as follows:

B : BOOK_X is literally true
E : The Earth was created in six days
C : Carbon_dating techniques are useless
S : Scientists/Researchers are frauds
G : God exists

Therefore, the statements in the given arguments are symbolically represented as :

(i) $B \rightarrow E$

(ii) $E \rightarrow C \wedge S$

(iii) $\sim$ S

(iv) B

(v) ~ G (*to show*)

Using ModusPonens on (i) and (iv), we get expression (vi) E

Using ModusPonens on (ii) & (vi) we get expression (vii) $C \wedge S$

Using Simplification on (vii), we get expression (viii) S

Using Addition on (viii), we get expression (ix) $S \vee \sim G$

Using DisjunctiveSyllogism(D.S.) on (iii) & (ix) we get expression (x) ~ G

The last statement is what is to be proved.

Remarks: (iii) and (viii) are contradicts with each other in the above deduction. In general, if we come across two statements (like S and ~ S) that contradict each other during the process of deduction, we can deduce any statement, even if the statement can never be True in any way. So, we can assume that any statement is true if both S and ~ S have already happened in the process of derivation.

4.12 PROOF SYSTEMS

A sequence of statements that follows logically from the previous set of statements or observations are the mathematical proofs in propositional logic. The last statements in the proof becomes the theorem. This proof system symbolizes the science of valid inference.

There are more than two main styles of proof system for propositional logic, but the two main styles are:

a) *Axiomatic Proof Systems and*
b) *Natural deduction Systems*

Axiomatic Proof Systems: This is a system where conclusion is derived from either the given hypothesis or using assumed premise, which are considered as truth. Such hypothesis or assumed premise is an *axiom*.

Example 1: Show that (α_1 ∨α_2) is a logical consequence of (α_2 ∧α_1) using proof systems.

Proof:

Step 1: (α_2 ∧α_1) (Premise)

Step 2: α_1 (Simplification, Step 1)

Step 3: α_2 (Simplification, Step 1)

Step 4: α_1 ∨α_2 (Addition on either Step 2 or Step 3)

We observe here that each step is either a truth or a logical consequence of previously established truths. In general, there can be more than one proof for establishing a given conclusion.

Thus, the proof system in propositional logic is quite similar to the proofs in mathematics, which follows a step-wise derivation of consequent from the given hypothesis. However, in propositional logic, we use well-formed formulas obtained from literals and connectors, rather than using English statements. We use the rules of inference (Section 4.7) and replacement rules (Section 4.8) to prove our conclusion.

Example 2: Show that α_1 -> ~α_4 can be derived from the given premises:

(α_1-> (α_2 ∨α_3)), α_2 -> ~α_1, α_4 -> ~α_3

Proof:

Step 1: (α_1-> (α_2 ∨α_3)) (Premise)

Step 2: α_1(Assumed Premise)

Step 3: α_2 ∨α_3 (Modus Ponens, Step 1, 2)

Step 4: α_2 -> ~α_1 (Premise)

Step 5: ~α_2 (Modus Tollens, Step 2, step 4)

Step 6: α_3 (Disjunctive Syllogism, Step 1, 2, 4)

Step 7: α_4-> ~α_3 (Premise)

Step 8: ~α_4 (Modus Tollens, Step 1, 2, 4, 7)

Step 9: α_1 -> ~α_4 (Step 2, 4)

Hence proved.

The discussion over Natural deduction Systems is given in section 4.13 below

4.13 NATURAL DEDUCTIONS

So far, we have discussed methods, of solving problems requiring reasoning of propositional logic, that were based on

i) Truth-table construction
ii) Use of inference rules,

and follow, directly or indirectly, **natural deduction approach.**

In order to determine whether or not a conclusion C in an argument is valid or invalid based on a given set of facts or axioms A_1, A_2,…, A_n, the only thing we currently know is that either a truth table should be constructed for the formula P: $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow C$,, or this formula should be converted to CNF or DNF by substituting equivalent formulas and simplifying it. There are other possible options available as well. The trouble with these methods, on the other hand, is that as the number n of axioms increases, the formula becomes more complicated (imagine n being equal to 50), and the number of variables involved, say k, also normally increases. When there are k different variables to consider in an argument, the size of the truth table grows to 2^k. For big values of k, the number of rows, denoted by 2^k, approaches an almost unmanageable level. As a result, it is necessary for us to look for alternative approaches that, rather than processing the entire argument as a single formula, process each of the individual formulas A_1, A_2, and C of the argument as well as their derivatives by applying some principles that ensure the validity of the results.

In an earlier section, we introduced eight different inference rules that can be used in propositional logic to help derive logical inferences. The methods of drawing valid conclusions that have been discussed up until this point are examples of an approach to drawing valid conclusions that is called the **natural deduction approach** of making inferences. This is an approach to drawing valid conclusions in which the reasoning system starts the reasoning process from the axioms, uses inferencing rules, and, if the conclusion can be validly drawn, then it ultimately reaches the conclusion that was intended. On the other hand, there is a method of obtaining legitimate conclusions that is known as the **Refutation approach**. This method, which will be covered in the following part, will be addressed.

The normal forms (CNF and DNF) also play a vital role in both Natural deductions and Resolution approach. To understand the normal forms, we need to start with the basic concepts of clauses, literals etc.

Some Definitions: A **clause** is a disjunction of literals. For example, $(E \vee \sim F \vee \sim G)$ is a clause. But $(E \vee \sim F \wedge \sim G)$ is not a clause. A **literal** is either an atom, say A, or its negation, say ~A.

Definition: A formula E is said to be in a **Conjunctive Normal Form (CNF)** if and only if E has the form E : $E_1 \wedge \ldots \wedge E_n$, $n \geq 1$, where each of E_1,…., E_n is a **disjunction** of literals.

Definition: A formula E is said to be in **Disjunctive Normal Form (DNF)** if and only if E has the form E: $E_1 \vee E_2 \vee \ldots E_n$, where each E_i is a **conjunction** of literals.

Examples: Let A, B and C be atoms. Then F:(~A∧B)∨(A∧~B∧~C) is a formula in a disjunctive normal form.

Example: Again G: (~A∨B)∧(A∨~B∨~C) is a formula in Conjunctive Normal Form, because it is a conjunction of the two disjunctions of literals viz of (~ A ∨ B) and (A ∨ ~ B ∨ ~ C)

Example: Each of the following is neither in CNF nor in DNF

(i) (~A∨B)∨(A∧~B∨C)
(ii) (A→B)∧ (~B∧~A)
Using table of equivalent formulas given above, any valid Propositional Logic formula can be transformed into CNF as well as DNF.

The steps for conversion to DNF are as follows

Step 1: Use the equivalences to remove the logical operators '↔' and '→':

(i) E↔G=(E→g)∧(G→E)

(ii) E→G=~E∨G

Step 2 Remove ~'s, if occur consecutively more than once, using

(iii) ~(~E)=E

(iv) Use De Morgan's laws to take '~' nearest to atoms

(v) ~(E∨G)=~E∧~G

(vi) ~(E∧G)=~E∨~G

Step 3 Use the distributive laws repeatedly

(vii) E∨(G∧H)=(E∨G)∧(E∨H)

(viii) E∧(G∨H)=(E∧G)∨(E∧H)

Example : Obtain a disjunctive normal form for the formula ~ (A → (~ B ∧ C)).
Consider A→(~B∧C)=~A∨(~B∧C) *(Using (E→F)=(~E∨F))*

Hence, ~(A→(~B∧C))=~(~A∨(~B∧C))

=~(~A)∧(~(~B∧C)) *(Using~(E∨F)=~E∧~F)*

= A ∧ (B ∨ (~ C)) *(Using~(~E)=E and ~(E∧F)=~E∨~F*

= (A ∧ B) ∨ (A ∧ (~ C)) *(Using E∧(F∨G)=(E ∧ F) ∨(E ∧ G))*

However, if we are to obtain CNF of $(\sim A \rightarrow (\sim B \wedge C))$, in the last but one step, we obtain

$\sim(A \rightarrow (\sim B \wedge C)) = A \wedge (B \vee \sim C)$, which is in CNF, because, each of A and

$(B \vee \sim C)$ is a disjunct.

Example: Obtain conjunctive Normal Form (CNF) for the formula: $D \rightarrow (A \rightarrow (B \wedge C))$

Consider

$D \rightarrow (A \rightarrow (B \wedge C))$ (using $E \rightarrow F = \sim E \vee F$ for the inner implication)

$= D \rightarrow (\sim A \vee (B \wedge C))$ (using $E \rightarrow F = \sim E \vee F$ for the outer implication)

$= \sim D \vee (\sim A \vee (B \wedge C))$

$= (\sim D \vee \sim A) \vee (B \wedge C)$ (using Associative law for disjunction)

$= ((\sim D \vee \sim A \vee B) \wedge (\sim D \vee \sim A \vee C)$

The last line denotes the conjunctive Normal Form of $D \rightarrow (A \rightarrow (B \wedge C))$

(Using distributivity of $\vee$ over $\wedge$)

Note: If we stop at the last but one stop, then we obtain $(\sim D \vee \sim A) \vee (B \wedge C) = \sim D \vee \sim A \vee (B \wedge C)$ is a Disjunctive Normal Form for the given formula : $D \rightarrow (A \rightarrow (B \wedge C))$

Sum up of the technique of natural deduction - Apart from constructing truth table to show that the conclusion follows from the given set of premises, another method exists known as *natural deduction.* In this case, we assume that the conclusion is not valid. We consider negated conclusion as a premise along with other given premises. We apply certain implications, equivalences and replacement rules to derive a contradiction to our assumption. Once, we obtain a contradiction, this proves that the given argument is true.

Example 1: Show that Z is a valid conclusion from the premises X,X->Y and Y->Z.

Proof: Step 1: ~Z (Negated conclusion as Premise)

Step 2: X (Premise)

Step 3:X-> Y (Premise)

Step 4: Y (using Step 2,3andModus Ponens)

Step 5: Y -> Z (Premise)

Step 6: Z (using Step 4, 5 and Modus Ponens)

We obtain the conclusion Z from the given premises, which is a contradiction to our assumption ~Z. Thus, the given conclusion is valid.

Example 2: Show that ~P is concluded from RV S, S-> ~Q, P -> Q and R ->~Q.

Proof: Step 1: ~(~P) (Negated conclusion as Premise)

Step 2: P (Step 1, Double negation)

Step 3: P -> Q (Premise)

Step 4: Q (Step 2, 3, Modus Ponens)

Step 5: S -> ~Q (Premise)

Step 6: ~S (Step 4, 5,Modus Tollens)

Step 7: R V S (Premise)

Step 8: R (Step 1, 3, 5, 7, Disjunctive Syllogism)

Step 9: R -> ~Q (Premise)

Step 10: ~Q (Step 1, 3, 5, 7, 9and Modus Ponens)

We obtain ~Q in Step 10and Q in step 4, which is a contradiction. Thus, our assumption is not valid. Hence, the conclusion follows from the set of premises.

4.14 PROPOSITIONAL RESOLUTION

For the most part, there are two distinct strategies that can be implemented in order to demonstrate the correctness of a theorem or derive a valid conclusion from a given collection of axioms:
i) natural deduction
ii) the method of refutation

In the method known as **natural deduction**, one begins with a given set of axioms, applies various rules of inference, and ultimately arrives at a conclusion. This method is strikingly similar to the intuitive reasoning that is characteristic of humans.

When using a **refutation approach**, one begins with the denial of the conclusion that is to be drawn and then proceeds to deduce a contradiction or the word "false." We are able to deduce a contradiction as a result of having presupposed that the conclusion is incorrect; hence, the premise that the conclusion is incorrect is itself incorrect. Therefore, the argument concerning the technique of resolution leads to the correctness of the conclusion. In this part of the article, we will talk about a different method known as the **Resolution Method**, which was

proposed by Robinson in 1965 and is based on the **refutation approach**. The Robinson technique, which has served as the foundation for numerous computerised theorem provers, highlights the significance of the method in question.

Propositional resolution is a sound, complete and powerful rule of inference used in Propositional Logic. It is used prove the unsatisfiability of the given set of statements. This is done using a strategy called Resolution Refutation that uses Resolution rule as described below.

Resolution Rule: The rule states given two statements as $\{\alpha_1, \alpha_2, \alpha_3 \ldots \alpha, \alpha_m\}$ and $\{\gamma_1, \gamma_2, \gamma_3 \ldots . \gamma_n\}$,then the conclusion is $\{\alpha_1, \alpha_2, \alpha_3 \ldots \alpha_m, \gamma_1, \gamma_2, \gamma_3, \ldots . \gamma_n\}$.

For example, the statements {A, B} and {C, ~B} leads to the conclusion {A, C}.

***Resolution Refutation*:**

a) Convert all the given statements to Conjunctive Normal Form (CNF).It is also described as AND of ORs'.For eg., $(A \vee B) \wedge (\sim A \vee B) \wedge (\sim B \vee A)$ is a CNF.
b) Obtain the negation of the given conclusion
c) Apply the resolution rule until either the contradiction is obtained or the resolution rule cannot be applied anymore.

4.14.1 Clausal Form

Any atomic sentence or its negation is called as a *literal*. A literal or the disjunction of at least two literals is called as the *clausal form* or *clause expression*. Next, a *clause* is defined as the set of literals in the clause form or clause expression. For example, consider two atomic statements represented using the literals X and Y. Their clausal expressions are X, ~X and (X V Y) and the clauses for these expressions are {X}, {~X} and {X, Y}. It is noteworthy that the empty set {} is always a clause as it represents an empty disjunction and hence, is unsatisfiable. Kindly note, conjunctive normal form (CNF) of an expression represents the corresponding clausal form.

Now, we shall first understand certain rules for converting the statements to the clause form as given below.

i) Operator: $(\beta_1 V \beta_2 V \beta_3 V \ldots . V \beta_m) => \{\beta_1, \beta_2, \beta_3, \ldots . \beta_m\}$
$(\beta_1 \wedge \beta_2 \wedge \beta_3 \wedge \ldots . \wedge \beta_m) => \{\beta_1\}, \{\beta_2\}, \{\beta_3\}, \ldots, \{\beta_m\}$
ii) Negation: same as Double Negation and De Morgan's Law in section 4.8
iii) Distribution: as in sub-section 4.8
iv) Implications: as Material implication and Material Equivalence (section 4.8)

Example 1: Convert A∧ (B->C) to clausal expression.

Step 1: A ∧ (~B V C) (using rule Material Implication to eliminate ->)

Step 2: {A}, {~B V C} (using rule Operator to eliminate ∧)

Example 2: Derive the Clausal form or Conjunctive normal form of X <-> Y.

Step 1: Replace bi-condition using Material equivalence rule:

$(X \rightarrow Y) \wedge (Y \rightarrow X)$

Step 2: Use Material Implication replacement rule to replace the implication:

$(\neg X \vee Y) \wedge (\neg Y \vee X)$

Example 3: Derive the CNF of $\sim Z \wedge \sim((\sim X) \rightarrow (\sim Y))$

Step 1: Replace implication using Material Implication(MI):

$\sim(\sim Z \wedge (\sim(\sim X) \vee \sim Y))$

Step 2: Use double negation (DN):

$\sim(\sim Z \wedge (X \vee \sim Y))$

Step 3: Apply DeMorgan's Law:

$\sim\sim Z \vee \sim(X \vee \sim Y)$

Step 4: Apply Double Negation:

$Z \vee \sim(X \vee \sim Y)$

Step 5: Apply DeMorgan's Law again:

$Z \vee (\sim X \wedge \sim\sim Y)$

Step 6: Apply Double Negation on ~~Y:

$Z \vee (\sim X \wedge Y)$

Step 7: Lastly, apply Distributive law to obtain CNF:

$(Z \vee \sim X) \wedge (Z \vee Y)$, which is the AND of OR's form.

$(Z \vee \sim X)$, $(Z \vee Y)$ are the clausal forms for the given expression.

4.14.2 Determining Unsatisfiability

If the obtained set of clauses is not satisfiable, then one can derive an empty clause using resolution principle as described above. In other words, to determine whether a set of propositions or premises {P} logically entails a conclusion C, write P ∨ {¬C} in clausal form and try to derive the empty clause as explained in examples below.

Example 1: Given a set of propositions : X -> Y, Y -> Z. Prove X -> Z.

Proof: To prove the conclusion, we add the negation of conclusion i.e ~(X -> Z) to the set of premises, and derive an empty clause.

Step 1: X -> Y (Premise)
Step 2: ~X ∨ Y (Premise, Material Implication)
Step 3: Y -> Z (Premise)
Step 4: ~Y ∨ Z (Premise, Material Implication)
Step 5: ~(X -> Z) (Premise, Negated Conclusion)
Step 6: ~(~X ∨ Z) (Premise, Material Implication)
Step 7: X ∧ ~Z (Premise, DeMorgans)
Step 8: X (Clausal form, Operator)
Step 9: ~Z (Clausal form, Operator)
Step 10: Y (Resolution rule on Premises in Step 2 and Step 8)
Step 11: Z (Resolution rule on Step 10 and Step 4)
Step 12: {} (Conjunction on Step 11 and Step 9)

Thus, the given set of premises entail the conclusion.

Example 2:Use propositional resolution to derive the goal from the given knowledge base.

a) Either it is a head, or Lisa wins.
b) If Lisa wins, then Mary will go.
c) If it is a head, then the game is over.
d) The game is not over.

Conclusion: Mary will go.

Proof: First consider propositions to represent each of the statement in knowledge base.

Let H: It is a head

L: Lisa wins

M: Mary will go

G: Game is over.

Re-writing the given knowledge base using the propositions defined.

a) H ∨ L
b) L -> M
c) H -> G
d) ~G

Conclusion: M

Step 1: H∨L (Premise)
Step 2: L -> M (Premise)
Step 3: ~L ∨ M (Step 4, Material Implication)
Step 4: H -> G (Premise)
Step 5: ~H ∨ G (Step 4, Material Implication)
Step 6: ~G (Premise)
Step 7: ~M (Negated conclusion as Premise)
Step 8: H ∨ M (Resolution principle on Step 1 and 3)
Step 9: M ∨ G (Resolution principle on Step 8 and 5)
Step 10: M (Resolution principle on Step 9 and 6)
Step 11: {} (Sep 10 and 7)

After applying Proof by Refutation i.e., contradicting the conclusion, the problem is terminated with an empty clause ({}). Hence, the conclusion is derived.

Example 3: Show that ~S1 follows from S1 -> S2 and ~(S1 ∧ S2).

Proof:

Step 1: S1 -> S2 (Premise)
Step 2: ~S1 ∨ S2 (Material Implication, Step 1)
Step 3: ~(S1 ∧ S2) (Premise)
Step 4: ~S1 ∨~S2 (De Morgan's, Step 3)
Step 5: ~S1 (Resolution, Step 2, 4)

The resolution mechanism in PL is not used until after the given statements or wffs have been converted into **clausal forms**. To obtain the clasual form of a wff, one must first convert the wff into the Conjuctive Normal Form (CNF). We are already familiar with the fact that a phrase is a formula (and only a formula) of the form: $A_1 \vee A_2 \vee \ldots\ldots\ldots \vee A_n$,where A_i might be either any atomic formula or its negation.

The method of resolution is actually generalization of Modus Ponens, whose expression is

$\frac{P, P \rightarrow Q}{Q}$ which can be written in the equivalent form as $\frac{P, \sim P \vee Q}{Q}$ (i.e. by using the relation $P \rightarrow Q \Rightarrow \sim P \vee Q$).

If we are provided that both P and $\sim P \vee Q$ are true, then we may safely assume that Q is also true. This is a straightforward application of a general resolution principle that will be covered in more detail in this unit.

The construction of a truth table can be used to demonstrate the validity of a resolution process (generally). In order to talk about the resolution process, we will first talk about some of the applications of that method.

Example: Let $C_1: Q \vee R$ and $C_2: \sim Q \vee S$ be two given clauses, so that, one of the literals i.e., Q occurs in one of the clauses (in this case C_1)and its negation $(\sim Q)$ occurs in the other clause C_2. Then application of resolution method in this case tells us to take disjunction of the remaining parts of the given clause C_1 and C_2, i.e., to take $C_3: R \vee S$ as **deduction** from C_1 and C_2. Then C_3 is called a **resolvent** of C_1 and C_2.

The two literals Q and $(\sim Q)$ which occur in two different clauses are called **complementary literals**.

In order to illustrate resolution method, we consider another example.

Example: Let us be given the clauses $C_1: \sim S \vee \sim Q \vee R$ and $C_2: \sim P \vee Q$.

In this case, complementary pair of literals viz. Q and $\sim Q$ occur in the two clause C_1 and C_2.

Hence, the resolution method states: $Conclude C_3: \sim S \vee R \vee (\sim P)$

Example: Let us be given the clauses $C_1: \sim Q \vee R$ and $C_2: \sim Q \vee S$

Then, in this case, the clauses do not have any complementary pair of literals and hence,

resolution method cannot be applied.

Example: Consider a set of three clauses

$C_1: R$

$C_2: \sim R \vee S$

$C_3: \sim S$

Then, from C_1 and C_2 we conclude, through resolution:

C_4:S

From C_3 and C_4, we conclude,

C_5:FALSE

However, a resolvent FALSE can be deduced only from an***unstatisfiable set of clauses*.** Hence, the set of clauses C_1, C_2 and C_3 is an unsatisfiable set of clauses.

Example: Consider the set of clauses

C_1:$R \vee S$
C_2:$\sim R \vee S$
C_3:$R \vee \sim S$
C_4:$\sim R \vee \sim S$
Then, from clauses C_1 and C_2 we get the resolvent
$C_5 : S \vee S = S$

From C_3 and C_4 we get the resolvent

C_6:$\sim$S

From C_5 and C_6 we get the resolvent

C_7:FALSE

Thus, again the set of clauses C_1, C_2, C_3 and C_4 is unsatisfiable.

Note: We could have obtained the resolvent FALSE from only two clauses, viz., C_2 and C_3. Thus, out of the given four clauses, even set of only two clauses viz, C_2 and C_3 is unsatisfiable. Also, a superset of any unsatisfiable set is unsatisfiable.

Example: Show that the set of clauses:

C_1:$R \vee S$
C_2:$\sim S \vee W$
C_3:$\sim R \vee S$
C_4:$\sim W$ is unsatisfiable.
From clauses C_1 and C_3 we get the resolvent
C_7:S
From the clauses C_7 and C_2 we get the resolvent
C_8:W
From the clauses C_8 and C_4 we get
C_9:FALSE
Hence, the given set of clauses is unsatisfiable.
Solution of the Problem Using the Resolution Method As was discussed before, the resolution process can also be understood as a refutation approach. The following is an example of a proving technique that can be used to solve problems:

After the symbolic representation of the issue at hand, an additional premise in the form of the negation of the wff, which stands for conclusion, should be added. You can infer either false or a contradiction from this improved set of premises and axioms. If we are able to get

to the conclusion that the statement is not true, then the conclusion that was required to be formed is correct, and the issue has been resolved. If, despite our best efforts, we are unable to arrive at the conclusion that the hypothesis is false, then we are unable to determine whether or not the conclusion is correct. As a result, the predicament cannot be solved using the axioms that have been provided and the conclusion that has been drawn.

Let's go on to the next step and apply Resolution Method to the issues we discussed earlier.

Example: If the interest rate goes up, the stock prices might go down. Also, let's say that most people are unhappy when the price of stocks goes down. Let's say that the rate of interest goes up. Show that most people are unhappy and that we can draw that conclusion.

To show the above conclusion, let us denote the statements as follows:

- A : Interest rate goes up,
- S : Stock prices go down
- U : Most people are unhappy

The problem has the following four statements

1) If the interest rate goes up, stock prices go down.
2) If stock prices go down, most people are unhappy.
3) The interest rate goes up.
4) Most people are unhappy. (to conclude)

These statements are first symbolized as wffs of PL as follows:

(1′) $A \rightarrow S$

(2′) $S \rightarrow U$

(3′) A

(4′) U*(to conclude)*

Converting to clausal form, we get

(i) $\sim A \vee S$

(ii) $\sim S \vee U$

(iii) A

(iv) U *(to be concluded)*

As per resolution method, assume (iv) as false, i.e., *assume ~U as initially given statement, i.e., an axiom.*

Thus, the set of axioms in clasual form is:

(i)$\sim A \vee S$

(ii)$\sim S \vee U$

(iii)A

(iv)$\sim U$

Then from (i) and (iii), through resolution, we get the clause

(v) S.

From (ii) and (iv), through resolution, we get the clause

(vi)~S

From (vi) and (v), through resolution we get,

(viii)FALSE

Hence, the conclusion, i.e., (iv)*U:Most people are unhappy* **is valid.**

From the above solution using the resolution method, we might have noticed that clausal conversion is a major step that takes a lot of time after translation to wffs. Most of the time, once the clause form is known, proof is easy to see, at least by a person.

☞ **Check Your Progress 4**

Ques 1. Prove that given set of premises are unsatisfiable:

a) {X, Y}, {~X, Z}, {~X,~Z}, {X, ~Y}

Ques 2. Consider the given knowledge base to prove the conclusion.

KB1. If Mary goes to school, then Mary eats lunch.

KB 2. If it is Friday, then Mary goes to school or eats lunch.

Conclusion: If it is Friday, then Mary eats lunch.

4.15 ANSWERS/SOLUTIONS

Check Your Progress 1

Ques No	Ans	Ques No	Ans
1	No	8	No
2	No	9	No
3	Yes	10	Yes
4	Yes	11	No
5	No	12	No
6	Yes	13	Yes
7	Yes	14	No

Check Your Progress 2

Ques No	Ans	Ques No	Ans
1	Atomic	6	Atomic
2	Compound	7	Compound
3	Compound	8	Compound
4	Compound	9	Compound
5	Atomic	10	Atomic

Check Your Progress 3

1. $(\sim\alpha\_1 \rightarrow (\alpha\_1 \vee \alpha\_2)) \leftrightarrow \alpha\_2$

α_1	**α_2**	**α_1 ∨α_2**	**~ α_1 → (α_1 ∨α_2)**	**(~α_1 → (α_1 ∨α_2)) ↔α_2**
_F(0)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_
_F(0)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_
_T(1)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_
_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_

2. $(\sim\alpha\_1 \leftrightarrow (\alpha\_2 \leftrightarrow \alpha\_3)) \vee (\alpha\_3 \wedge \alpha\_2)$

α_1	**α_2**	**α_3**	**α_2 ↔α_3**	**(~α_1 ↔ (α_2 ↔ S3)**	**α_3 ∧α_2**	**(~α_1 ↔ (α_2 ↔α_3)) ∨ (α_3 ∧α_2)**
_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_T(1)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_F(0)_	_F(0)_
_F(0)_	_F(0)_	_T(1)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_
_T(1)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_T(1)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_
_T(1)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_
_T(1)_	_T(1)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_

3. $((\alpha\_1 \wedge \alpha\_2) \rightarrow \alpha\_3) \vee \sim\alpha\_4$

α_1	**α_2**	**α_1∧α_2**	**α_3**	**((α_1 ∧α_2) → α_3**	**α_4**	**((α_1 ∧α_2) → α_3) ∨ ~α_4**
_F(0)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_T(1)_
_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_
_F(0)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_T(1)_
_F(0)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_F(0)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_
_T(1)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_	_T(1)_
_T(1)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_T(1)_
_T(1)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_T(1)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_
_T(1)_	_T(1)_	_T(1)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_
_T(1)_	_T(1)_	_T(1)_	_F(0)_	_F(0)_	_T(1)_	_F(0)_
_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_
_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_

4. $((\alpha\_1 \rightarrow \sim \alpha\_2) \leftrightarrow \alpha\_3) \rightarrow \sim (\alpha\_1 \text{ V } \alpha\_1)$

α_1	**A_2**	**α_1 → ~α_2**	**α_3**	**~α_1**	**((α_1 → ~α_2) ↔ α_3)**	**((α_1 → ~α_2) ↔α_3) → ~ (α_1 V α_1)**
_F(0)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_T(1)_
_F(0)_	_F(0)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_F(0)_
_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_F(0)_	_F(0)_
_F(0)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_	_T(1)_
_T(1)_	_F(0)_	_T(1)_	_F(0)_	_F(0)_	_F(0)_	_T(1)_
_T(1)_	_F(0)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_F(0)_
_T(1)_	_T(1)_	_T(1)_	_F(0)_	_F(0)_	_F(0)_	_F(0)_
_T(1)_	_T(1)_	_T(1)_	_T(1)_	_F(0)_	_T(1)_	_T(1)_

Check Your Progress 4

1. For the given set of premises, derivse an empty clause {} using proposition resolution to show that the given set of premises are unsatisfiable.

Step 1: XV Y (Premise)
Step 2: ~X V Z (Premise)
Step 3: ~XV ~Z (Premise)
Step 4: X V ~Y (Premise)
Step 5: X (Resolution on Step 1 and 4)
Step 6: ~X (Resolution on Step 2 and 3)
Step 7:{} (Using Step 5 and 6)

Thus, the given set of premises are unsatisfiable.

2. First introduce notation set for the given knowledge base as:
S: Mary goes to school
L: Mary eats lunch
F: It is Friday
Corresponding knowledge base is:
KB1: S -> L
KB2:F -> (S V L)
Conclusion: F -> L

Proof:
Step 1: ~S V L (Premise, Material Implication)
Step 2: ~F V (S V L) (Premise, Material Implication)
Step 3: F (Negated conclusion)
Step 4: ~L (Negated conclusion)
Step 5: (S V L) (Resolution on Step 2 and 3)
Step 6: L (Resolution on Step 1 and 5)
Step 7:{}(Resolution on Step 4 and 6)

4.16 FURTHER READINGS

1. C. L. Liu & D. P. Mohapatra, Elements of Discrete Mathematics: A Computer Oriented Approach, Fourth Edition, 2017, Tata McGraw Hill Education.
2. David J. Hunter, Essentials of Discrete Mathematics, Third Edition, 2016, Jones and Bartlett Publishers.
3. James L. Hein, Discrete Structures, Logic and Computability, Fourth Edition, 2015, Jones and Bartlett Publishers.
4. Kenneth H. Rosen, Discrete Mathematics and Its Applications, Eighth Edition, 2021, Tata McGraw Hill Education.
5. Thomas Koshy, Discrete Mathematics with Applications, 2012, Elsevier Academic Press.