AI_ ML Module1

Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015

1. What is AI?
2. Foundations of AI
3. History of AI
4. Problem-solving: Problem-solving agents
5. Example problems
6. Searching for Solutions
7. Uninformed Search Strategies:
    a. Breadth First search
    b. Depth First Search
    c. Time and Space Complexity of BFS and DFS

Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015

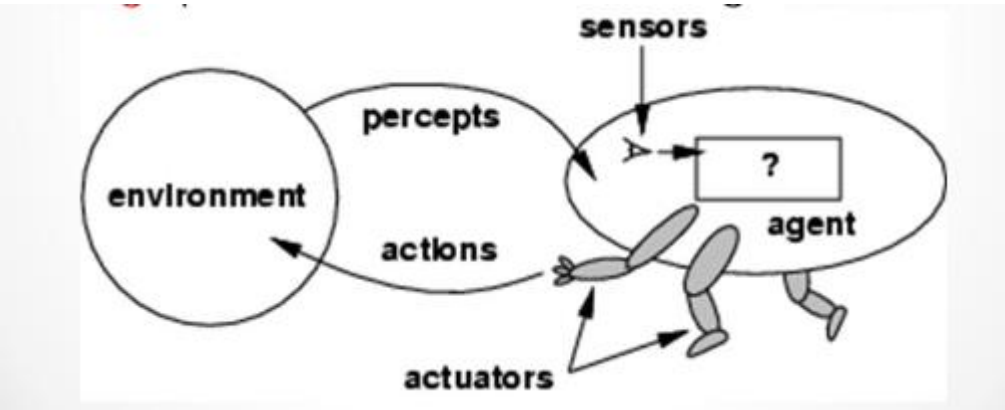
1.1 What is AI?

Artificial Intelligence (AI) is a field of computer science dedicated to develop systems capable of performing tasks that would typically require human intelligence. These tasks include **learning, reasoning, problem-solving, perception, understanding natural language**, and even interacting with the environment. AI aims to create machines or software that can mimic or simulate human cognitive functions.

According to Russell and Norvig, AI can be defined as follows:

" *AI (Artificial Intelligence) is the study of agents that perceive their environment, reason about it, and take actions to achieve goals. Each such agent implements a function that maps percept sequences to actions, and the study of these functions is the subject of AI.*"

Agents: In AI, an agent is any entity that can perceive its environment and take actions to achieve its goals. These agents can be physical robots, software programs, or any system capable of interacting with the world. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.



Source: The Intelligent Agent depicted on books

Percepts: Percepts are the inputs or information an agent receives from the environment. This could include data from sensors, cameras, microphones, or any other sources of information.

Actions: Actions are the responses or behaviours the agent can exhibit to achieve its goals or objectives. These can be physical movements, data processing, decision-making, or any other form of output.

Function: The central idea of AI is to design a function (or algorithm) that maps percept sequences to actions, allowing the agent to make intelligent decisions based on its observations.

Intelligent Agent: This term refers to the function that enables the agent to act in a way that is considered intelligent, i.e., making decisions that are adaptive and can lead to goal achievement.

Russell and Norvig's definition underscore the goal of AI, which is to create systems that can perceive their environment, reason about it, and take appropriate actions to achieve specific objectives. The field of AI encompasses a wide range of techniques and approaches, from symbolic reasoning to machine learning and deep learning, all aimed at building intelligent agents that can perform tasks typically associated with human intelligence

In **Figure,** there are **eight explanations** of **AI shown in two groups**. The top ones talk about thinking and reasoning, while the bottom ones focus on behavior. The left-side definitions judge based on how close to **human-like performance** they are, while the right-side definitions assess based on an ideal measure called **rationality**. A system is considered **rational** if it makes the best decisions based on the information it has.

Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015

Let us look at the four approaches in more detail.

1. Acting humanly: The Turing Test approach

Turing Test : The Turing test is a method proposed by Alan Turing to evaluate a machine's capability to exhibit intelligent behavior that is indistinguishable from that of a human. In this test, a human evaluator engages in a conversation with

both a human and a machine through text, without knowing which is which. If the evaluator cannot reliably differentiate between the machine's responses and the human's, the machine is said to have passed the Turing test, demonstrating human-like intelligence.

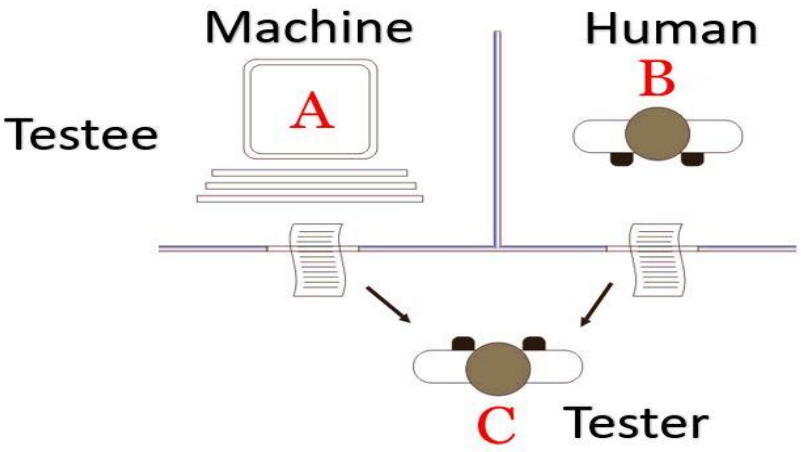


Image Source: https://www.how2shout.com/what-is/what-is-turing-test-and-it-used-for.html

In order to pass **Total Turing Test** the computer must have the following features:

1. **Proficiency in natural language processing** for effective communication in English.
2. **Capability for knowledge representation** to store acquired information.
3. **Automated reasoning to utilize stored data** for answering questions and deriving new conclusions.
4. **Machine learning to adapt to novel situations**, detect patterns, and extrapolate information.
5. **Computer vision** capabilities for object perception.
6. **Robotics proficiency** for manipulating objects and navigating its surroundings.

These **six** disciplines encompass the majority of artificial intelligence (AI), and Turing deserves recognition for designing a test that maintains its relevance six decades later.

2. Thinking humanly: The cognitive modeling approach

To ascertain whether a program exhibits human-like thinking, understanding human thought processes is essential. **This involves exploring the inner workings of the human mind through introspection, psychological experiments, and brain imaging**. Once a precise theory of the mind is

established, it can be translated into a computer program. If the program's input-output behaviour aligns with human behaviour, it suggests shared mechanisms.

3. Thinking rationally: The "laws of thought" approach

Aristotle, an early philosopher, sought to formalize "right thinking" through syllogisms, offering patterns for sound reasoning. These principles, foundational to logic, inspired logicians in the 19th century to develop precise notation for various objects and their relations. By 1965, programs capable of theoretically solving any problem in logical notation emerged, paving the way for the logicist tradition in artificial intelligence.

4. Acting rationally: The rational agent approach

Computer agents are expected to autonomously operate, perceive their environment, persist over time, adapt to change, and pursue goals. A rational agent seeks the best outcome by emphasizing correct inferences, aligning with the Turing Test skills. While correct inference is a facet of rationality, it's not exhaustive, as some situations lack provably correct actions. The rational-agent approach, incorporating knowledge representation, reasoning, and learning, offers generality and scientific advantages. Despite challenges acknowledged in the book, it adopts the hypothesis that perfect rationality is a useful starting point, simplifying complex environments for foundational discussions in the field.

Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015

1.2 The Foundations of Artificial Intelligence

This section offers a concise history of the disciplines influencing AI, focusing on key figures, events, and ideas.

Philosophy: In the philosophical exploration of AI, following questions arises:

- Can formal rules be used to draw valid conclusions?
- How does the mind arise from a physical brain?
- Where does knowledge come from?
- How does knowledge lead to action

Aristotle, formulated laws for the rational mind, developed syllogisms for mechanical reasoning. Historical figures like Ramon Lull, Thomas Hobbes, and Leonardo da Vinci contributed to the idea that mechanical artifacts could perform useful reasoning. Descartes introduced the mind-matter distinction, raising debates on free will and advocating rationalism and dualism. Materialism emerged as an alternative to dualism, positing that the brain's operation constitutes the mind. The empiricism movement, led by Bacon and Locke, emphasized sensory origins of understanding. Logical positivism, developed by the Vienna Circle, combined rationalism and empiricism, shaping the computational theory of mind presented by Carnap and Hempel in analysing knowledge acquisition from experience.

Mathematics: In the mathematical exploration of AI, following questions arises:

- What are the formal rules to draw valid conclusions?
- What can be computed?
- How do we reason with uncertain information?

The mathematical formalization of logic, computation, and probability played crucial roles. George Boole developed Boolean logic, extended later by Frege and Tarski. Kurt Godel's incompleteness theorem revealed the limits of logical deduction. Alan Turing addressed computability, introducing the Church-Turing thesis. The concept of tractability, differentiating polynomial and exponential complexity, emerged in the mid-1960s. NP-completeness theory, by Cook and Karp, identified intractable problems. Probability theory, pioneered by Cardano and advanced by Pascal, Bernoulli, Laplace, and Bayes, became essential in handling uncertain information.

Economics: In the Economics of AI, various attempts have been done to address the following questions:

- How should we make decisions so as to maximize payoff?
- How should we do this when others may not go along?
- How should we do this when the payoff may be far in the future

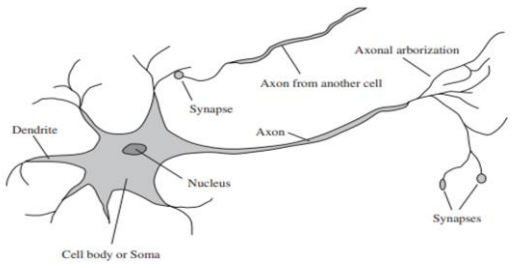
The science of economics began in 1776 with Adam Smith's "An Inquiry into the Nature and Causes of the Wealth of Nations," treating economics as a science focused on individual agents maximizing their economic well-being. Léon Walras formalized the mathematical treatment of utility, later improved by Frank Ramsey, John von Neumann, and Oskar Morgenstern in "The

Theory of Games and Economic Behavior" (1944). Decision theory, combining probability and utility theory, provides a framework for decisions under uncertainty. Game theory, developed by Von Neumann and Morgenstern, includes the surprising result that rational agents may adopt randomized policies. Operations research, initiated in World War II, addressed sequential decision problems, formalized by Richard Bellman. Herbert Simon's work on satisficing, making "good enough" decisions, earned him a Nobel Prize in economics in 1978. Recent interest in decision-theoretic techniques for agent systems has emerged since the 1990s.

Neuroscience (How do brains process information?)

Neuroscience, the study of the nervous system, particularly the brain, aims to understand the mechanisms enabling thought. While the brain's role in cognition has been recognized for millennia, it wasn't until the 18th century that the brain was widely acknowledged as the seat of consciousness. Paul Broca's study of aphasia in 1861 highlighted localized brain areas for specific functions, and Camillo Golgi's staining technique (1873) allowed the observation of individual neurons. Santiago Ramon y Cajal applied mathematical models to the nervous system, while Nicolas Rashevsky pioneered mathematical modeling in neuroscience. Hans Berger's electroencephalograph (EEG) in 1929 and recent developments in functional magnetic resonance imaging (fMRI) provide detailed brain activity images. Despite advances, understanding cognitive processes remains a challenge. Brains and computers differ in properties, with futurists speculating on a singularity when computers surpass human intelligence.

Structure of Neuron and comparison of Human brain with Computers:



	Brain	Computer
number of processors	≈ 10 billion *neurons* (massively parallel)	1 *CPU* (intrinsically serial)
processor complexity	simple inaccurate	complex accurate
processor speed	slow (millisec)	fast (nanosec)
inter-processor communications	fast (μsec)	slow (millisec)
learning mode	learn from experience	manual programming
failure robustness	many neurons die without drastic effect	single fault often leads to system failure
memory organization	content addressable (CAM)	location addressable (LAM)

	Supercomputer	Personal Computer	Human Brain
Computational units	10^4 CPUs, 10^{12} transistors	4 CPUs, 10^9 transistors	10^{11} neurons
Storage units	10^{14} bits RAM 10^{15} bits disk	10^{11} bits RAM 10^{13} bits disk	10^{11} neurons 10^{14} synapses
Cycle time	10^{-9} sec	10^{-9} sec	10^{-3} sec
Operations/sec	10^{15}	10^{10}	10^{17}
Memory updates/sec	10^{14}	10^{10}	10^{14}

Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015

Psychology: How do humans and animals think and act?

Scientific psychology traces its origins to Hermann von Helmholtz and Wilhelm Wundt. Helmholtz applied the scientific method to human vision, and Wundt established the first experimental psychology laboratory in 1879. Wundt emphasized introspective experiments, but behaviorism, led by John Watson, emerged, rejecting mental processes. Cognitive psychology, rooted in the information-processing view of the brain, finds early roots in William James's works. Frederic Bartlett and Kenneth Craik, at Cambridge's Applied Psychology Unit, advocated for cognitive modeling. Craik outlined a knowledge-based agent's three key steps: stimulus translation, internal representation manipulation, and retranslation into action. Donald Broadbent continued Craik's work, modeling psychological phenomena as information processing. The field of cognitive science emerged in the U.S., notably at MIT's 1956 workshop where Miller, Chomsky, Newell, and Simon presented influential papers, demonstrating how computer models could address memory, language, and logical thinking in psychology.

Computer Engineering (How can we build an efficient computer?)

For artificial intelligence (AI) to succeed, two key elements are essential: **intelligence and an artifact**, with the **computer** being the chosen artifact. The modern digital electronic computer, crucial for AI, was independently invented in three countries during World War II. Early machines like **Heath Robinson** and **Colossus** paved the way, and the **Z-3**, developed by Konrad Zuse in 1941, marked the first programmable computer. The **ABC**, built by John Atanasoff and Clifford Berry in the early 1940s, was the first electronic computer, but the **ENIAC**, developed by **John Mauchly** and John Eckert, emerged as a highly influential precursor to modern computers.

Each generation of computer hardware since then has seen increased speed, capacity, and reduced costs. While early calculating devices existed, programmable machines like Joseph Marie Jacquard's loom in 1805 and Charles Babbage's designs in the mid-19th century, especially the **Analytical Engine**, were significant precursors to modern computers. Ada Lovelace, Babbage's colleague, is regarded as the world's **first programmer**. AI also acknowledges the debt to computer science's software side, contributing to operating systems, programming languages, and tools, with reciprocal innovation between AI and mainstream computer science.

Control Theory and Cybernetics (How can artifacts operate under their own control?)

Ktesibios of Alexandria, around 250 B.C., constructed the first self-controlling machine—an innovative **water clock with a regulator maintaining** a constant flow rate. This invention marked a shift in the capabilities of artifacts, enabling them to modify behavior in response to environmental changes. Control theory, rooted in stable feedback systems, saw significant developments in the 19th century. James Watt's steam engine governor and Cornelis Drebbel's thermostat exemplify self-regulating feedback control systems.

The central figure in the establishment of control theory, especially cybernetics, was Norbert Wiener (1894–1964). Wiener, along with Arturo Rosenblueth and Julian Bigelow, challenged

behaviorist orthodoxy, viewing purposive behavior as arising from regulatory mechanisms minimizing "error" between current and goal states. Wiener's book "Cybernetics" (1948) became influential in shaping the public perception of artificially intelligent machines. W. Ross Ashby in Britain also pioneered similar ideas, emphasizing the creation of intelligence through homeostatic devices.

Modern control theory, particularly stochastic optimal control, aims to design systems maximizing an objective function over time, aligning with AI's goal of creating optimally behaving systems. Despite shared objectives, AI and control theory emerged as distinct fields due to the different mathematical techniques and problem sets each addressed. While control theory relied on calculus and matrix algebra for systems with continuous variables, AI, leveraging logical inference and computation, ventured into addressing problems like language, vision, and planning beyond the scope of control theory.

Linguistics (How does language relate to thought?):

In 1957, B. F. Skinner published "Verbal Behavior," a comprehensive account of the behaviorist approach to language learning. However, the book faced significant criticism in a review by linguist Noam Chomsky, who had recently published his own book, "Syntactic Structures." Chomsky argued that behaviorism failed to explain the creativity inherent in language acquisition, particularly how children understand and generate novel sentences. Chomsky's theory, rooted in syntactic models dating back to the ancient linguist Panini, offered a formal framework programmable in principle.

The intersection of modern linguistics and AI occurred around the same time, giving rise to computational linguistics or natural language processing. The complexity of language understanding became evident, extending beyond sentence structure to encompass subject matter and context, a realization that gained prominence in the 1960s. Early work in knowledge representation, aimed at rendering knowledge in a computationally usable form, was closely tied to language and influenced by linguistic research, itself connected to philosophical analyses of language over decades.

Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015

Foundations of AI (Summarized contributions)

Philosophy

Name	Year	Suggestion
Aristotle	384–322 B.C.	Formulated laws for the rational mind and developed syllogisms for mechanical reasoning.
Ramon Lull	Died 1315	Envisioned useful reasoning by mechanical artifacts.
Thomas Hobbes	1588–1679	Likened reasoning to numerical computation.
Leonardo da Vinci	1452–1519	Designed a functional mechanical calculator around 1500.
Wilhelm Schickard	1592–1635	Constructed the first known calculating machine in 1623.
Blaise Pascal	1623–1662	Built the famous Pascaline calculator in 1642.
Gottfried Wilhelm Leibniz	1646–1716	Speculated on machines thinking and acting independently.
Thomas Hobbes (Leviathan)	1588–1679	Suggested the concept of an "artificial animal."
René Descartes	1596–1650	Discussed the mind-matter distinction and advocated rationalism and dualism.
Vienna Circle (Rudolf Carnap)	1891–1970	Developed logical positivism, combining rationalism and empiricism.
David Hume	1711–1776	Proposed the principle of induction in the 18th century.
Francis Bacon	1561–1626	Initiated the empiricism movement.
John Locke	1632–1704	Emphasized the sensory origins of understanding.
Ludwig Wittgenstein	1889–1951	Contributed to the work of the Vienna Circle.
Bertrand Russell	1872–1970	Contributed to the work of the Vienna Circle.
Carl Hempel	1905–1997	Introduced the confirmation theory to analyze knowledge acquisition.
Antoine Arnauld	1612–1694	Correctly described a quantitative formula for deciding what action to take in cases like this.
John Stuart Mill	1806-1873	John Stuart Mill's (1806–1873) book Utilitarianism (Mill, 1863) promoted the idea of rational decision criteria in all spheres of human activity

Mathematics

George Boole	1815–1864	Developed propositional (Boolean) logic.
Gottlob Frege	1848–1925	Extended Boole's logic to include objects and relations, creating first-order logic.
Alfred Tarski	1902–1983	Introduced a theory of reference, linking logic to the real world.
Kurt Godel	1906–1978	Formulated incompleteness theorems, highlighting limits of logical deduction.
Alan Turing	1912–1954	Investigated computability and introduced the Church-Turing thesis.
Steven Cook	1971	Pioneered NP-completeness theory, identifying intractable problems.
Richard Karp	1972	Contributed to NP-completeness theory, identifying intractable problems.
Gerolamo Cardano	1501–1576	Framed the idea of probability in the context of gambling events.
Blaise Pascal	1623–1662	Applied probability to predict outcomes in gambling games.
James Bernoulli	1654–1705	Advanced probability theory.
Pierre Laplace	1749–1827	Contributed to probability theory and introduced statistical methods.
Thomas Bayes	1702–1761	Proposed Bayes' rule for updating probabilities in the light of new evidence.

Economics

Name	Year	Contribution
Adam Smith	1723–1790	Published "An Inquiry into the Nature and Causes of the Wealth of Nations."
Léon Walras	1834–1910	Formalized the mathematical treatment of utility.
Frank Ramsey	1931	Contributed to the formalization of utility theory.
John von Neumann	1903–1957	Developed game theory in "The Theory of Games and Economic Behavior" (1944).
Oskar Morgenstern	1902–1977	Collaborated with von Neumann in developing game theory.
Richard Bellman	1920–1984	Formalized Markov decision processes in operations research.
Herbert Simon	1916–2001	Pioneered satisficing and received the Nobel Prize in economics in 1978.

Neuroscience

Name	Year	Contribution
Aristotle	335 B.C.	Acknowledged human brains as proportionally larger than other animals.
Paul Broca	1824–1880	Demonstrated localized brain areas for specific functions, particularly speech.
Camillo Golgi	1843–1926	Developed staining technique allowing observation of individual neurons.
Santiago Ramon y Cajal	1852–1934	Conducted pioneering studies on the brain's neuronal structures.
Nicolas Rashevsky	1936, 1938	Applied mathematical models to the study of the nervous system.
Hans Berger	1929	Invented the electroencephalograph (EEG) for measuring intact brain activity.
John Searle	1992	Coined the phrase "brains cause minds," emphasizing the connection between brains and consciousness.

Psychology

Name	Year	Contribution
Hermann von Helmholtz	1821–1894	Applied the scientific method to the study of human vision, contributing to physiological optics.
Wilhelm Wundt	1832–1920	Established the first laboratory of experimental psychology in 1879, emphasizing introspective experiments.
John Watson	1878–1958	Led the behaviorism movement, rejecting theories involving mental processes and advocating for the study of objective measures.
William James	1842–1910	Contributed to cognitive psychology, viewing the brain as an information-processing device.
Frederic Bartlett	1886–1969	Directed Cambridge's Applied Psychology Unit, fostering cognitive modeling.
Kenneth Craik	1943	Developed a knowledge-based agent model, emphasizing stimulus translation, internal representation manipulation, and retranslation into action.
Donald Broadbent	1926–1993	Continued Craik's work, modeling psychological phenomena as information processing.
George Miller	1956	Presented influential work on the psychology of memory at the 1956 MIT workshop.
Noam Chomsky	1956	Presented influential work on language models at the 1956 MIT workshop.
Allen Newell and Herbert Simon	1956	Presented influential work on logical thinking models at the 1956 MIT workshop.

Computer Engineering

Name	Year	Contribution
Alan Turing	1940	Led the team that built the electromechanical Heath Robinson computer for deciphering German messages.
Konrad Zuse	1941	Invented the Z-3, the first operational programmable computer in Germany.
John Atanasoff and Clifford Berry	1940–1942	Assembled the ABC, the first electronic computer at Iowa State University.
John Mauchly and John Eckert	1940s	Developed the ENIAC, a highly influential precursor to modern computers.
Joseph Marie Jacquard	1805	Invented a programmable loom that used punched cards for storing weaving instructions.
Charles Babbage	1792–1871	Designed the Difference Engine and Analytical Engine, significant precursors to modern computers.
Ada Lovelace	1815–1852	Collaborated with Charles Babbage, wrote programs for the Analytical Engine, and is considered the world's first programmer.

Control Theory and Cybernetics

Name	Year	Contribution
Ktesibios of Alexandria	c. 250 B.C.	Built the first self-controlling machine—a water clock with a regulator that maintained a constant flow rate.
James Watt	1736–1819	Created the steam engine governor, a self-regulating feedback control system.

Cornelis Drebbel	1572–1633	Invented the thermostat, another example of a self-regulating feedback control system.
Norbert Wiener	1894–1964	Pioneered control theory and cybernetics, exploring the connection between biological and mechanical control systems.
W. Ross Ashby	1903–1972	Elaborated on the idea that intelligence could be created by using homeostatic devices containing appropriate feedback loops.

Linguistics

Name	Year	Contribution
B. F. Skinner	1957	Published "Verbal Behavior," a comprehensive account of behaviorist language learning, sparking subsequent debate.
Noam Chomsky	1957	Criticized behaviorist theory in a review of Skinner's book, highlighting the inadequacy in explaining language creativity. Published "Syntactic Structures" with a formal, programmable theory of language.
Panini	c. 350 B.C.	Ancient linguist whose syntactic models influenced Chomsky's theory and contributed to the formal understanding of language.

Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015

1.3 History of Artificial Intelligence

Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015

The history of artificial intelligence (AI) is a fascinating journey spanning several decades. Here's a brief timeline highlighting key milestones in the development of AI, merged with a structured overview:

1.3.1 The Gestation of Artificial Intelligence (1943–1955)

1943: Warren McCulloch and Walter Pitts develop the first mathematical model of a neural network.
1949: Hebbian learning, formulated by Donald Hebb, becomes a lasting influence on neural network development.
1950: Alan Turing introduces the Turing Test and key AI principles.
1951: UNIVAC I, the first commercially produced computer, is used for statistical analysis, laying the groundwork for data processing.
1951: McCarthy earns his PhD and later plays a pivotal role in establishing AI at Dartmouth College.

1.3.2 The Birth of Artificial Intelligence (1956)

1956: John McCarthy organizes a landmark AI workshop at Dartmouth College, marking the official birth of artificial intelligence.
1956: The term "artificial intelligence" is coined at the Dartmouth Conference.

1.3.3 Early Enthusiasm, Great Expectations (1952–1969)

Late 1950s: IBM produces early AI programs challenging predefined task limitations.
1958: McCarthy defines the Lisp language and introduces time-sharing.
1963: Marvin Minsky establishes Stanford's AI lab, emphasizing practical functionality.
1965: Joseph Weizenbaum creates ELIZA, an early natural language processing program.
1966–1973: Setbacks occur as early successes fail to scale, leading to reduced support for AI research.
1969: The Stanford Research Institute develops Shakey, the first mobile robot with reasoning abilities.

1.3.4 A Dose of Reality (1966–1973)

1969: Despite setbacks, the discovery of back-propagation learning algorithms for neural networks leads to a resurgence of interest.
1970s: Initial enthusiasm for AI fades, leading to the first "AI winter" as progress stalls

1.3.5 Knowledge-Based Systems: The Key to Power? (1969–1979)

1969: DENDRAL exemplifies a shift towards domain-specific knowledge.
Late 1970s: The Heuristic Programming Project explores expert systems, emphasizing domain-specific knowledge.
1979: Marvin Minsky and Seymour Papert publish "Perceptrons," a book critical of certain AI approaches.

1.3.6 AI Becomes an Industry (1980–Present)

Early 1980s: The first successful commercial expert system, R1, is implemented at Digital Equipment Corporation.
1981: Japan's "Fifth Generation" project and the U.S.'s Microelectronics and Computer Technology Corporation respond to AI's growing influence.
1980s: Rapid growth followed by the "AI Winter," a period of decline in the AI industry.
1985: Expert systems, software that emulates decision-making of a human expert, gain popularity.

1.3.6 The Return of Neural Networks (1986–Present)

Mid-1980s: Rediscovery of the back-propagation learning algorithm leads to the emergence of connectionist models.
Late 1980s: The AI industry experiences a decline known as the AI Winter.

1.3.7 AI Adopts the Scientific Method (1987–Present)

Late 1980s: AI shifts towards a more scientific and application-focused approach, experiencing a revival in the late 1990s.
1990-2005: Neural Networks Resurgence and Practical Applications
1997: IBM's Deep Blue defeats chess champion Garry Kasparov.
1999: Rodney Brooks introduces the concept of "embodied intelligence" with Cog, a humanoid robot.

1.3.9 The Emergence of Intelligent Agents (1995–Present)

Late 1980s: The SOAR architecture addresses the "whole agent" problem.
Late 1990s–2000s: AI technologies underlie Internet tools, contributing to search engines, recommender systems, and website aggregators.

1.3.10 The Availability of Very Large Data Sets (2001–Present)

Late 1990s: A revival in AI with a shift towards a more scientific approach.
2000s: Emphasis on the importance of large datasets in AI research, leading to significant advancements.

2001: The DARPA Grand Challenge initiates research in autonomous vehicles.
2005: Stanford's Stanley wins the DARPA Grand Challenge, showcasing advances in self-driving technology.
2006: Geoffrey Hinton and colleagues publish a paper on deep learning, reigniting interest in neural networks.
2011: IBM's Watson wins Jeopardy!, demonstrating the power of natural language processing.
2012: AlexNet, a deep convolutional neural network, achieves a breakthrough in image recognition at the ImageNet competition.
2012-Present: AI in the Mainstream and Ethical Concerns
2014: Facebook's AI lab introduces DeepFace for facial recognition, reaching human-level accuracy.
2016: AlphaGo, an AI developed by DeepMind, defeats world champion Go player Lee Sedol.
2018: OpenAI releases GPT-2, a large-scale language model.
2020s: AI applications become integral in various industries, raising concerns about ethics, bias, and job displacement.

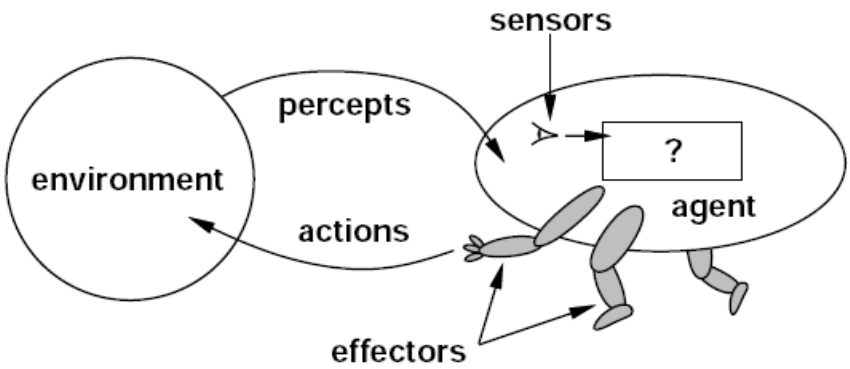
Present: AI applications are deeply embedded in various industries, marking a new spring for the field.
The history of AI is marked by cycles of optimism, followed by periods of stagnation, but recent years have seen unprecedented progress and integration of AI technologies into everyday life. Ongoing research and ethical considerations will continue to shape the future of artificial intelligence.

Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015

1.4 Problem Solving Agents

Agent: In the context of AI, **an agent is a system or program** that perceives its environment through sensors, makes decisions or takes actions to achieve specific goals, and is capable of autonomy. Agents can be physical entities like robots or virtual entities like software programs.

- (Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Novig)



Sources: (Pattie Maes, MIT Media Lab) , (Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Novig),

Types of Agents:

Agent Type	Description
Simple Reflex Agents	Select actions based on the current percept, without considering the history of past percepts.
Model-Based Reflex Agents	Maintain an internal model of the world, considering the history of percepts for decision-making.
Goal-Based Agents	Designed to achieve specific objectives, using internal goals to determine actions.
Utility-Based Agents	Evaluate actions based on a utility function, quantifying the desirability of different outcomes.
Learning Agents	Improve performance over time through learning from experience.
Logical Agents	Use logical reasoning for decision-making in environments with explicit knowledge representation.
Reactive Agents	Select actions based on the current situation, suitable for real-time, dynamic environments.
Deliberative Agents	Consider multiple possible actions, consequences, and plan ahead to achieve goals thoughtfully.
Mobile Agents	Have the ability to move through the environment, commonly used in robotics and autonomous systems.
Collaborative Agents	Work together, communicate, and coordinate actions to achieve common goals.
Rational Agents	Make decisions that maximize expected utility, aiming for the best outcome given knowledge and goals.

Problem Solving Agent is a type of goal based intelligent agent in artificial intelligence that is designed to analyse a situation, identify problems or goals, and then take actions to achieve those goals. These agents are designed to address and solve complex problems or tasks in its environment.

In the city of Arad, Romania, an agent on a touring holiday has a performance measure with various goals, such as improving suntan, language skills, exploring sights, and avoiding hangovers. The decision problem is complex, involving tradeoffs and guidebook analysis. However, if the agent has a nonrefundable ticket to fly out of Bucharest the next day, it is logical for the agent to prioritize the goal of reaching Bucharest. This simplifies the decision problem, as actions not leading to Bucharest can be disregarded. Goal formulation, the first step in problem-solving, helps organize behaviour by narrowing down objectives based on the current situation and the agent's performance measure.

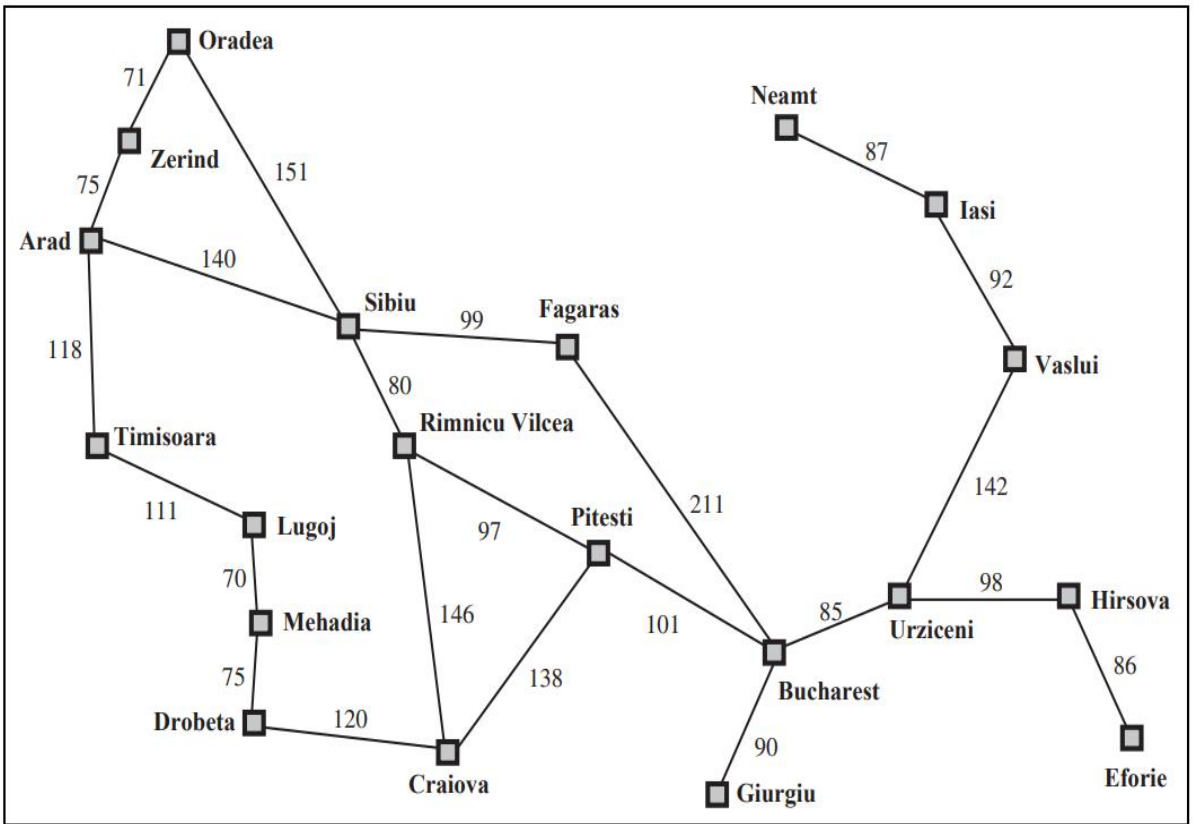


Figure 3.2 A simplified road map of part of Romania.

1.4.1 Steps followed by Problem Solving Agents:

Problem-solving agents follow a series of steps to analyse a situation, formulate goals, and take actions to achieve those goals. The typical steps followed by problem-solving agents are:

1. **Perception**: Gather information about the current state of the environment through sensors or perceptual mechanisms.

2. **Goal Formulation**: Define the objectives or goals that the agent is trying to achieve. This involves specifying what the agent is aiming for in the given situation.

3. **Problem Formulation**: Convert the vague goals into a specific, actionable problem. This step defines the current state, the desired state, and the possible actions that can be taken to move from the current state to the desired state.

4. **Search:** Explore possible sequences of actions to find a solution to the formulated problem. This involves considering different paths and evaluating their feasibility and desirability.

5. **Action Selection**: Choose the best sequence of actions based on the results of the search. The selected actions should lead the agent from the current state to the desired state.

6. **Execution**: Implement the chosen actions in the real world. This involves interacting with the environment and carrying out the planned sequence of actions using actuators.

7. **Learning (Optional):** In some cases, problem-solving agents may incorporate learning mechanisms to improve their performance over time. Learning can be based on feedback from the environment or from the consequences of past actions.

8. **Feedback and Iteration**: If the goals are not fully achieved or if the environment changes, the agent may need to iterate through the problem-solving process. This involves revisiting the perception, goal formulation, and problem formulation steps to adapt to new information.

Hence, we employ a straightforward "**formulate, search, execute**" framework for the agent, illustrated in Figure 3.1. Following the formulation of a goal and a corresponding problem, the agent initiates a search procedure to find a solution. Subsequently, the agent follows the solution's guidance for its actions, executing the recommended next steps—usually starting with the initial action of the sequence—and removing each completed step. Once the solution has been implemented, the agent proceeds to formulate a new goal.

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Figure 3.1 A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

1.4.2 Well Defined Problems and Solutions:

A problem can be defined formally by five components:

1. **State Representation**: Encompasses the initial state from which the agent begins its problem-solving journey, represented, for example, as "*In(Arad)*."
2. **Actions and Applicability**: Describes the set of possible actions available to the agent in a given state, denoted as ACTIONS(s). For instance, in the state *In(Arad)*, applicable actions include {*Go(Sibiu), Go(Timisoara), Go(Zerind)}*.
3. **Transition Model**: Specifies the consequences of actions through the transition model, represented by the function RESULT(s,a), which yields the state resulting from performing action a in state s. For example, RESULT(*In(Arad),Go(Zerind))=In(Zerind)*.

4. **Goal Specification and Test**: Defines the goal state or states and includes a test to determine whether a given state satisfies the goal conditions. In the example, the goal is represented as the singleton set *{In(Bucharest)}*.
5. **Cost Functions**: Encompasses both the path cost function, assigning a numeric cost to each path, and the step cost, denoted as $c(s,a,s')$, which represents the cost of taking action **a** in state **s** to reach state s'. The cost functions play a crucial role in evaluating and optimizing the performance of the agent's solution.

These **five components** collectively provide a comprehensive definition of a problem and serve as inputs to problem-solving algorithms. The solution to the problem is an action sequence that leads from the initial state to a state satisfying the specified goal conditions, with the quality of the solution evaluated based on the assigned costs.

1.4.3 Formulating Problems:

The previous section introduced a problem formulation for reaching Bucharest, involving the initial state, actions, transition model, goal test, and path cost. However, this formulation is a theoretical model, lacking real-world intricacies. The chosen state description, such as "In(Arad)," simplifies the complex reality of a cross-country trip, excluding factors like travel companions, radio programs, and weather. This simplification, known as abstraction, is essential.

In addition to abstracting the state, actions must also be abstracted. Driving, for instance, involves numerous effects beyond changing location, such as time consumption, fuel usage, and pollution generation. The formulation only considers location changes, omitting actions like turning on the radio or slowing down for law enforcement.

Determining the appropriate level of abstraction requires precision. Abstract states and actions correspond to large sets of detailed world states and sequences. A valid abstraction allows expanding abstract solutions into detailed ones. The abstraction is useful if executing abstract actions is easier than the original problem, ensuring they can be carried out without extensive search or planning by an average agent. Constructing effective abstractions involves removing unnecessary details while maintaining validity and ensuring ease of execution. Without this ability, intelligent agents would struggle to navigate the complexities of the real world.

1.5 Example Problems: Toy problems and Real-World Problems

The problem-solving methodology has found application in diverse task environments, encompassing a broad range of scenarios. We categorize these scenarios into two types**: toy problems and real-world problems**.

A toy problem is designed to showcase or test various problem-solving techniques, featuring a precise and concise description. This allows different researchers to use it for comparing algorithm performances.

On the other hand, **a real-world problem** is one that holds significance for people, lacking a single universally agreed-upon description. However, we can provide a general sense of their formulations.

1.5.1 Toy Problems

1.5.1.1 Vacuum World Problem:

The problem can be formalized as follows:

1. **States:** The state is defined by the agent's location and the presence of dirt in specific locations. The agent can be in one of two locations, each potentially containing dirt. Consequently, there are 8 possible world states (2×2^2). For a larger environment with n locations, there would be $n \cdot 2^n$ states.
2. **Initial state**: Any state can serve as the initial state.
3. **Actions**: In this uncomplicated environment, each state presents three actions: Left, Right, and Suck. More extensive environments might also include Up and Down.
4. **Transition model**: Actions produce expected effects, except for instances where moving Left in the leftmost square, moving Right in the rightmost square, and Sucking in a clean square result in no effect. The comprehensive state space is depicted in Figure 3.3.
5. **Goal test**: This assesses whether all squares are clean.
6. **Path cost**: Each step incurs a cost of 1, making the path cost equivalent to the number of steps taken in the path.

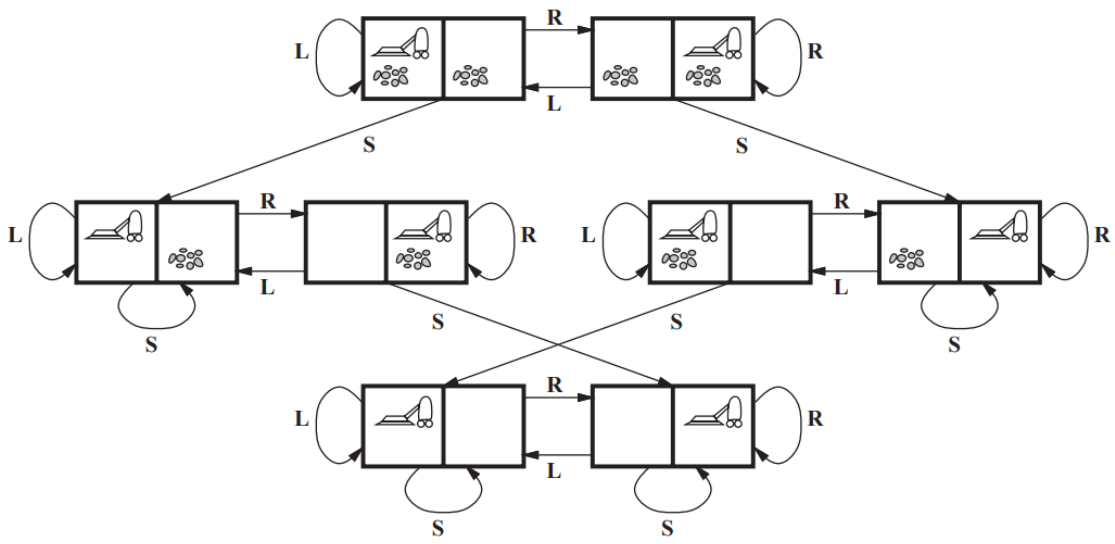


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

1.5.1.2 Eight Puzzle:

The 8-puzzle, illustrated in Figure 3.4, features a 3×3 board with eight numbered tiles and an empty space. Tiles adjacent to the empty space can slide into it, and the goal is to achieve a specified configuration, as depicted on the right side of the figure. The standard formulation is outlined as follows:

1. **States:** A state description indicates the position of each of the eight tiles and the empty space within the nine squares.
2. **Initial state:** Any state can be designated as the initial state.
3. **Actions:** In its simplest form, actions are defined as movements of the empty space—Left, Right, Up, or Down. Different subsets of these actions are possible based on the current location of the empty space.
4. **Transition model**: Given a state and an action, the model returns the resulting state. For instance, applying Left to the starting state in Figure 3.4 would switch the positions of the 5 and the empty space.
5. **Goal test:** This checks if the state aligns with the specified goal configuration shown in Figure 3.4. Other goal configurations are also conceivable.
6. **Path cost:** Each step incurs a cost of 1, making the path cost equivalent to the number of steps taken in the path.

The abstraction in this formulation is evident in the treatment of actions, which are abstracted to their fundamental movements—Left, Right, Up, or Down—depending on the position of the empty space.

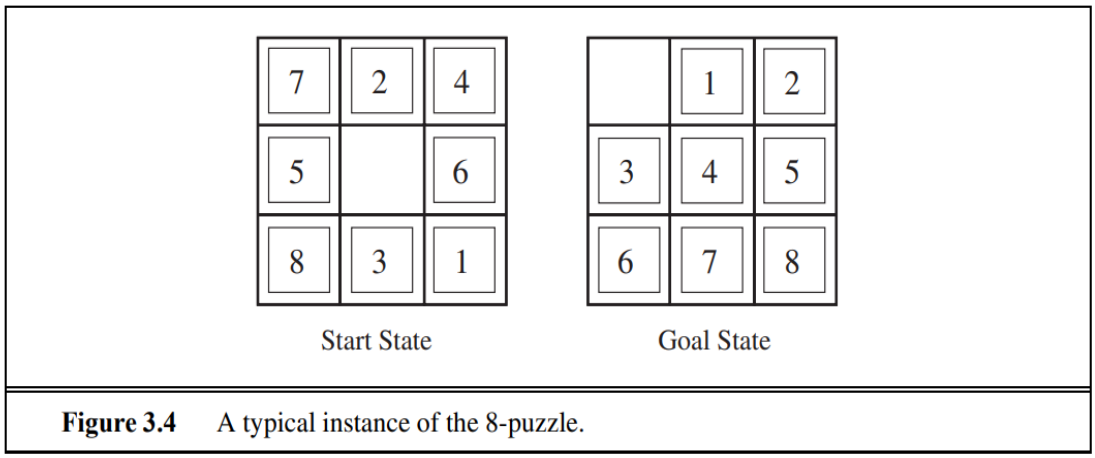
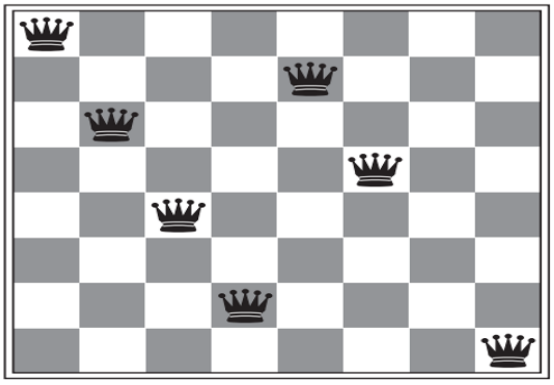


Figure 3.4 A typical instance of the 8-puzzle.

1.5.1.3 Eight Queens Problem:

The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.) Figure 3.5 shows an attempted solution that fails: the queen in the rightmost column is attacked by the queen at the top left.



1. **States:** Any arrangement of 0 to 8 queens on the board is a state.
2. **Initial state**: No queens on the board.
3. **Actions:** Add a queen to any empty square.
4. **Transition model**: Returns the board with a queen added to the specified square.
5. **Goal test:** 8 queens are on the board, none attacked.

1.5.1.4 Math's Sequences:

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5 .$$

The problem definition is as follows:

1. **States:** Positive numbers.
2. **Initial state**: 4.
3. **Actions:** Apply factorial, square root, or floor operation (factorial for integers only).
4. **Transition model**: As given by the mathematical definitions of the operations.
5. **Goal test**: State is the desired positive integer

1.5.2 Real Time Problems

1. Route Finding Problem
2. Touring Problem
3. Traveling Salesperson Problem
4. VLSI Layout
5. Robot Navigation

Consider the **airline travel problems that must be solved by a travel-planning Web site**:

1. **States:** Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and their status as domestic or international, the state must record extra information about these "historical" aspects.
2. **Initial state**: This is specified by the user's query.
3. **Actions**: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
4. **Transition model**: The state resulting from taking a flight will have the flight's destination as the current location and the flight's arrival time as the current time.
5. **Goal test**: Are we at the final destination specified by the user?

6. **Path cost**: This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

1.6 Searching for Solutions:

- The **SEARCH TREE** possible action sequences starting at the initial state form a search tree with the initial state NODE at the root; the **branches** are **actions** and the **nodes** correspond to states in the state space of the problem.
- **Expanding** the current state applying each legal action to the current state, thereby **generating** a new set of state.

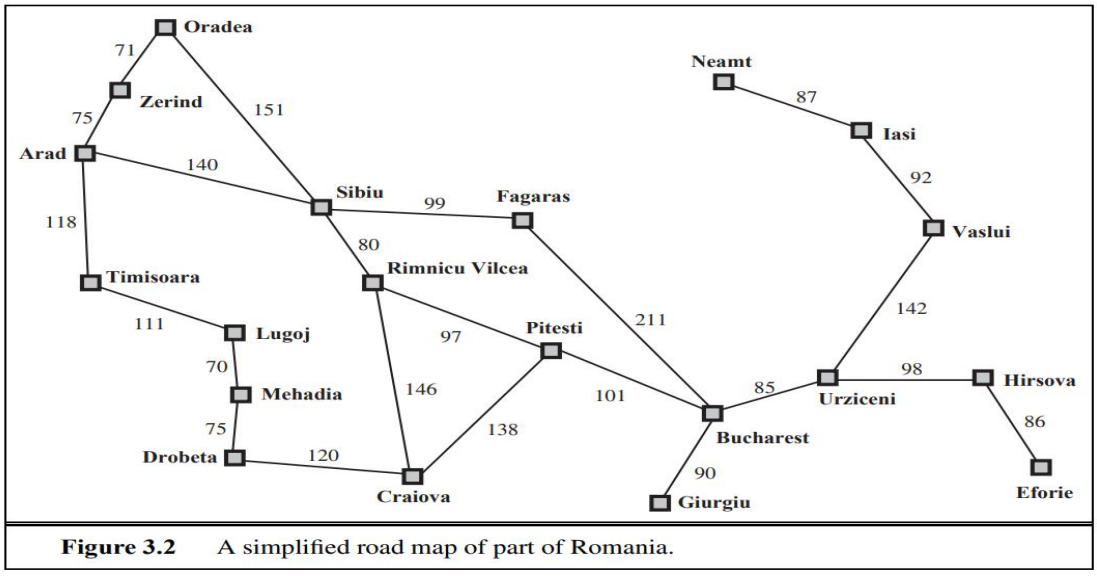
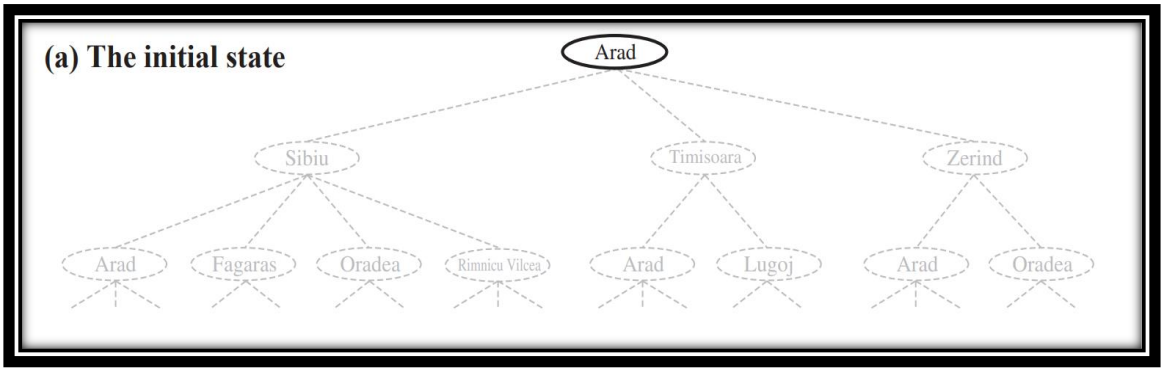
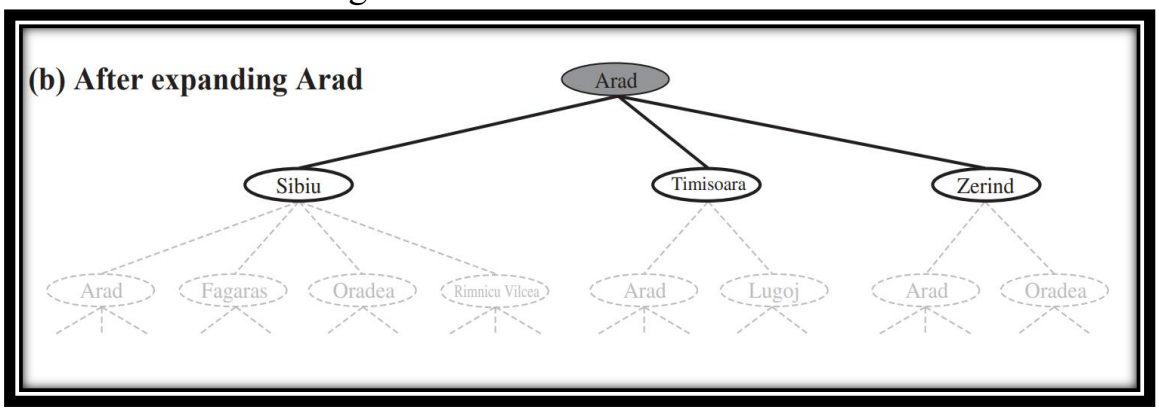


Figure 3.2 A simplified road map of part of Romania.

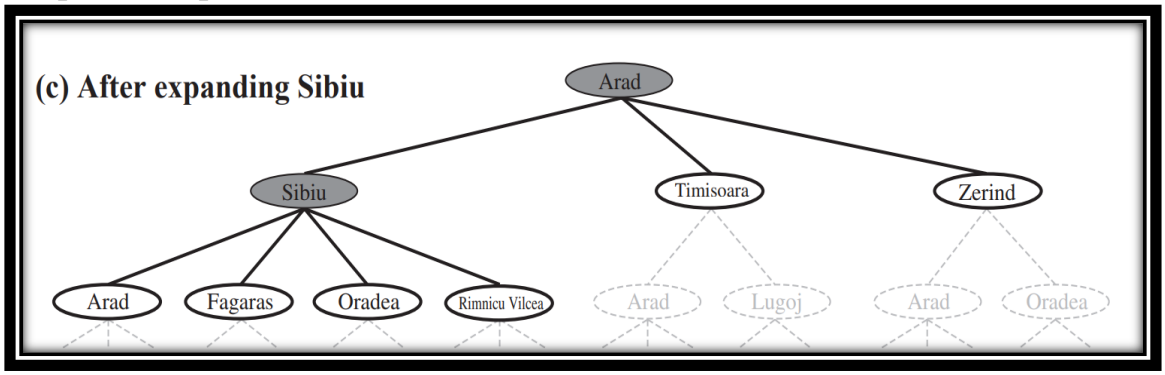
Partial search trees for finding a route from Arad to Bucharest is as shown in the following figures. Initially the searching for route starts from the route node "Arad".



The set of all leaf nodes available for expansion at any given point is called the frontier. Arad node has frontier { Sibiu, Timisoara, Zerind}.Expansion of nodes will be done from left to right.



In the following figure the left most node Sibiu will be further expanded to explore the path to reach the Goal i.e Bucharest.



But after expanding Sibiu , the following frontier will be obtained : {Arad, Fagaras, Oradea, Rimincu Vilcea} . Since Arad is already visited, searching continues from Fagaras.

General Pseudocode for Tree Search is as illustrated below:

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal state then return the corresponding solution
      expand the chosen node, adding the resulting nodes to the frontier
```

General Pseudocode for Graph Search is as illustrated below:

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal state then return the corresponding solution
      add the node to the explored set
      expand the chosen node, adding the resulting nodes to the frontier
         only if not in the frontier or explored set
```

Following figure illustrates the snapshots of sequences of search trees generated by a graph search on the Romania problem (i.e. To find the route from Arad to Bucharest).

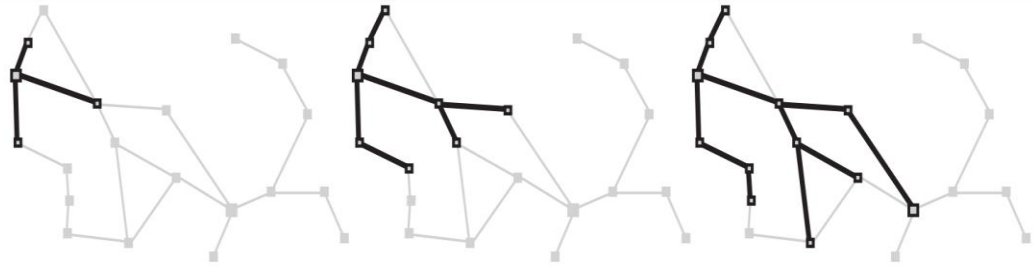


Figure 3.8 A sequence of search trees generated by a graph search on the Romania problem of Figure 3.2. At each stage, we have extended each path by one step. Notice that at the third stage, the northernmost city (Oradea) has become a dead end: both of its successors are already explored via other paths.

Following figure illustrates the separation property of Graph search illustrated on a rectangular grid problem:

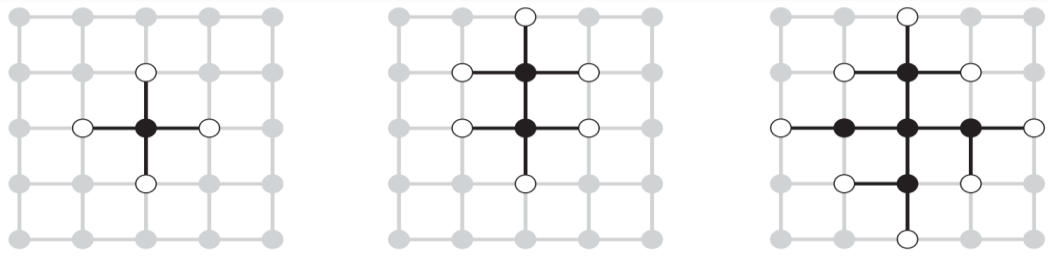


Figure 3.9 The separation property of GRAPH-SEARCH, illustrated on a rectangular-grid problem. The frontier (white nodes) always separates the explored region of the state space (black nodes) from the unexplored region (gray nodes). In (a), just the root has been expanded. In (b), one leaf node has been expanded. In (c), the remaining successors of the root have been expanded in clockwise order.

1.6.1 Infrastructure for search algorithms

Search algorithms require a data structure to keep track of the search tree that is being constructed.

For each **node n** of the tree, we have a structure that contains four components:

- **n.STATE**: the state in the state space to which the node corresponds;
- **n.PARENT**: the node in the search tree that generated this node;
- **n.ACTION**: the action that was applied to the parent to generate the node;
- **n.PATH-COST**: the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers.

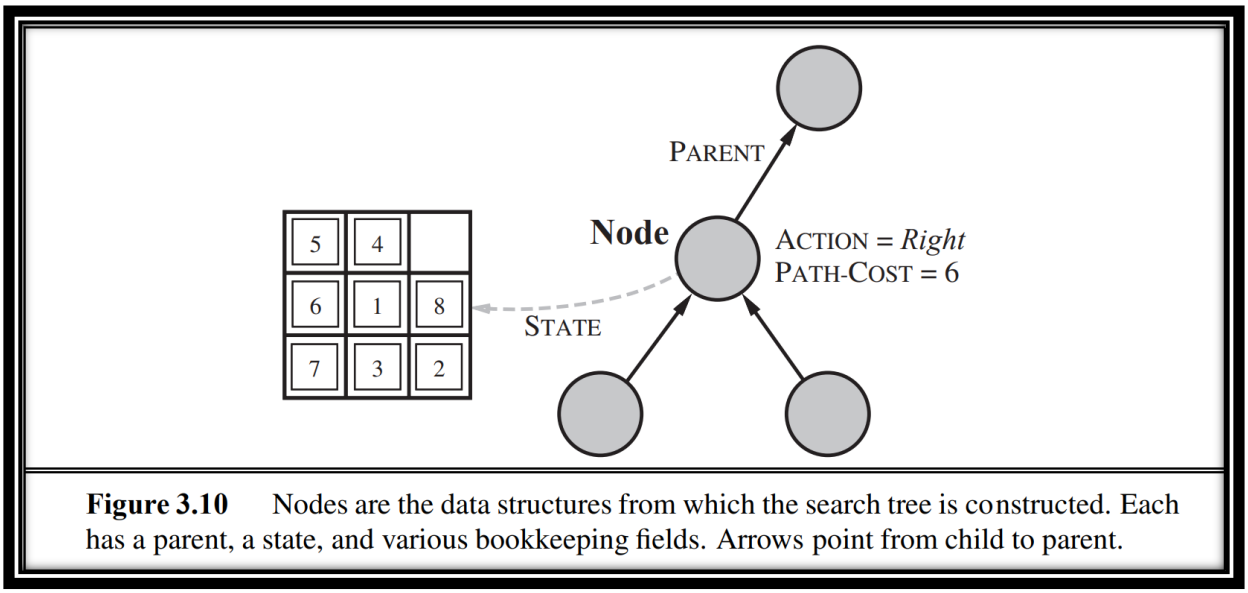


Figure 3.10 Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

Following figure illustrates the generic pseudocode for any child node in search tree:

```
function CHILD-NODE(problem, parent, action) returns a node
  return a node with
      STATE = problem.RESULT(parent.STATE, action),
      PARENT = parent, ACTION = action,
      PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

1.6.2 Measuring problem-solving performance:

We can evaluate an algorithm's performance in four ways:

1. **COMPLETENESS:** Is the algorithm guaranteed to find a solution when there is one?
2. **OPTIMALITY:** Does the strategy find the optimal solution?
3. **TIME COMPLEXITY:** How long does it take to find a solution?
4. **SPACE COMPLEXITY:** How much memory is needed to perform the search?

1.7 Uninformed and Informed Search Strategies:

In the context of search algorithms in artificial intelligence and computer science, the terms "informed" and "uninformed" refer to different strategies for exploring a search space. These strategies are commonly used in algorithms designed to find solutions to problems, typically within a graph or a state space.

Uninformed Search: Also known as blind search, uninformed search algorithms do not have any additional information about the problem other than the connectivity of the states or nodes in the search space.

The algorithms explore the search space without considering the specific characteristics of the problem or the goal location. Uninformed search methods are generally simpler and may explore a large portion of the search space, which can be inefficient for certain types of problems.

Examples:
1.Breadth-first search
2.Uniform-cost search
3.Depth-first search
4.Depth-limited search
5.Iterative deepening depth-first search
6.Bidirectional search

Informed Search: Informed search algorithms, also called heuristic search algorithms, use additional knowledge about the problem to guide the search more efficiently towards the goal.

These algorithms make use of heuristics, which are rules or estimates that provide information about the desirability of different paths. The heuristics help in selecting paths that are more likely to lead to a solution.

A classic example of an informed search algorithm is **A* (A-star),** which incorporates a heuristic function to evaluate the desirability of different paths and combines it with information about the cost incurred so far.

Examples:

1.Greedy best-first search
2.A* search: Minimizing the total estimated solution cost
3.Memory-bounded heuristic search
4.AO* Search
5.Problem Reduction
6.Hill Climbing

Key differences

Characteristic	Uninformed Search	Informed Search
Information Utilization	No additional knowledge	Additional knowledge (heuristics)
Decision Making	Based solely on structure of the search space	Uses heuristics to intelligently guide the search
Goal State	May require exhaustive exploration to find the goal state	More efficient, guided towards potential solution paths
Search Algorithms	Examples: Breadth-First Search Depth-First Search	Examples: A*, Greedy Best-First Search, etc.
Completeness	Completeness depends on the specific algorithm	Completeness depends on the specific algorithm and heuristic
Optimality	May not guarantee the most optimal solution	A* is optimal under certain conditions
Efficiency	May be less efficient for certain problems due to exhaustive exploration	Generally more efficient due to heuristic guidance
Examples	BFS, DFS, Uniform Cost Search, etc.	A*, Greedy Best-First Search,etc.

Enqueue and Dequeue Operations

In the context of any queue-based algorithm, enqueue and dequeue are operations related to adding elements to the queue and removing elements from the front of the queue, respectively.

Enqueue Operation: Adding an element to the end of the queue.

Example: If the queue is [A, B, C] and you enqueue the node D, the queue becomes [A, B, C, D].

Dequeue Operation: Removing an element from the front of the queue.

Example: If the queue is [A, B, C, D] and you dequeue, the result is A, and the queue becomes [B, C, D].

These operations are fundamental for maintaining the order of exploration in BFS. A queue is a First-In-First-Out (FIFO) data structure, meaning that the first element added to the queue will be the first one to be removed. The queue ensures that nodes are processed in a level-wise manner, which is essential for BFS to systematically explore the tree or graph.

1.7.1 Breadth-First Search (BFS)

Breadth-First Search (BFS), a strategy where we start from a root node, expand it to generate its children, and then put those children in a queue (i.e, FIFO) to expand then later. This means all nodes at some depth level d of the tree get expanded before any node at depth level d+1 gets expanded. The goal test is applied when nodes are immediately detected (i.e., before adding it to the queue) because there's no benefit to continue checking nodes. BFS is complete and optimal, but it also suffers from horrible space and time complexity.

Algorithm: Breadth-First Search (BFS): Breadth-First Search is an uninformed search algorithm that explores all the nodes at the current depth before moving on to nodes at the next depth level. It starts at the root node and explores each neighbour before moving to the next level.

Description:

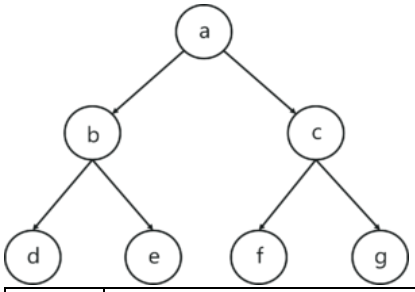
1. Initialize a queue with the initial state (usually the root node).
2. While the queue is not empty:
    a. Dequeue a node from the front of the queue.
    b. If the node contains the goal state, return the solution.
    c. Otherwise, enqueue all the neighbouring nodes that have not been visited.
3. If the queue becomes empty and the goal state is not found, then there is no solution.

Pseudocode for BFS:

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

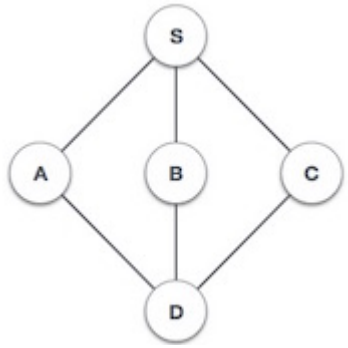
Figure 3.11 Breadth-first search on a graph.

Example 1: BFS on binary tree



Step	Details	Visited Nodes	QUEUE
1	Initialization: Start with the root node A.	V = {}	a
2	Node **a,** Visited, Dequeue node **a** and enqueue neighbour nodes **b** and **c** to queue	V = {a}	b c b
3	Node **b** visited, deque node **b** and enqueue neighbour nodes **d** and **e** to queue.	V = {a,b}	c d c e d c
4	Node **c** visited, deque node **c** and enqueue neighbour nodes **f** and **g** to queue	V = {a,b,c}	e d f e d g f e d
5	Node **d** visited, deque node **d**	V = {a,b,c,d}	g f e
6	Node **e** visited, deque node **e**	V = {a,b,c,d,e}	g f
7	Node **f** visited, deque node **f**	V = {a,b,c,d,e,f}	g
8	Node **g** visited, deque node **g**	V = {a,b,c,d,e,f,g}	

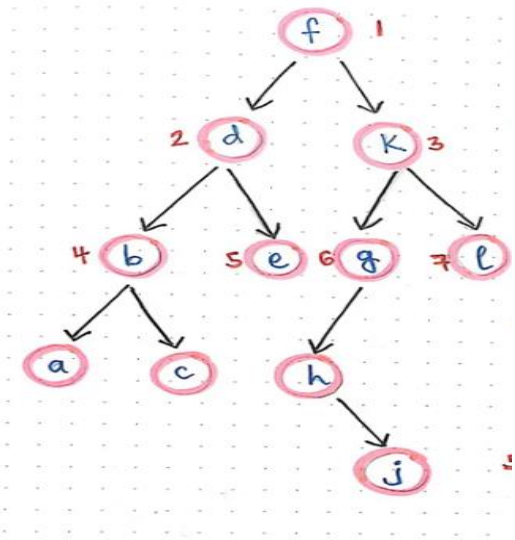
Example 2: BFS on Simple Graph

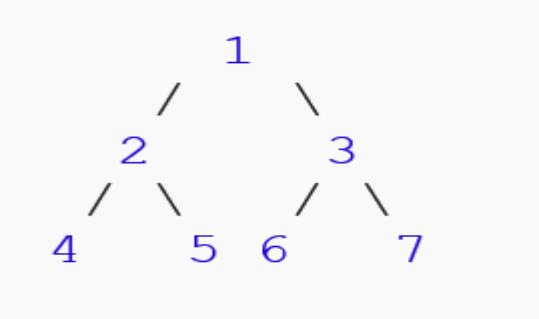


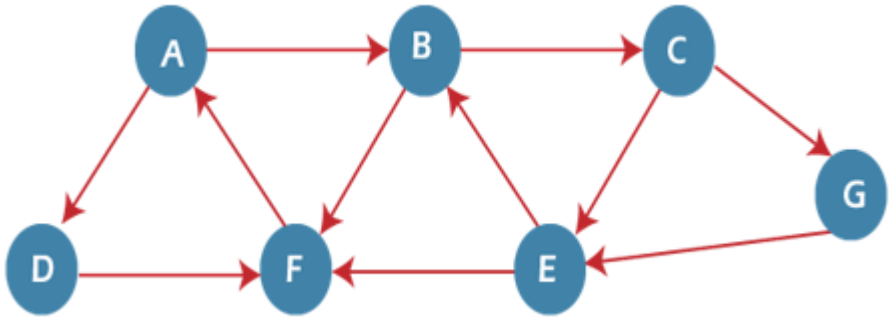
Step	Description	Visited Nodes	Queue
1	Initialize the queue.	V= {}	S
2	Node **S** Visited, Dequeue node **S** and enqueue neighbour nodes **A, B** and **C** to queue.	V= {S}	A B A C B A
3	Node **A** Visited, Dequeue node **A** and enqueue neighbour node **D**	V= {S,A}	D C B
4	Node **B** Visited, Dequeue node **B**	V= {S,A,B}	D C
5	Node **C** Visited, Dequeue node **C**	V= {S,A,B,C}	D
6	Node **D** Visited, Dequeue node **D**	V= {S,A,B,C,D}	

Breadth First Search: S A B C D

Exercise: Apply BFS on Following







1.7.2 Depth First Search

Depth-First Search (DFS) is a tree traversal algorithm that explores as far as possible along each branch before backtracking. In the context of a binary tree, DFS can be implemented using **recursion or a Stack (LIFO QUEUE**). Let's go through the DFS algorithm on a binary tree with an example.

DFS Algorithm on Binary Tree:

Recursive Approach:

- Start at the root node.
- Visit the current node.
    - Recursively apply DFS to the left subtree.
    - Recursively apply DFS to the right subtree.

Stack-Based Approach:

Depth-First Search (DFS) can also be implemented using a Last-In, First-Out (LIFO) queue/Stack. The basic idea is to push nodes onto the queue and explore as deeply as possible before backtracking. In the context of a binary tree, the LIFO queue simulates the stack used in the recursive and stack-based approaches. Here's how the DFS algorithm works on a binary tree using a LIFO queue with an example:

DFS Algorithm with LIFO Queue:

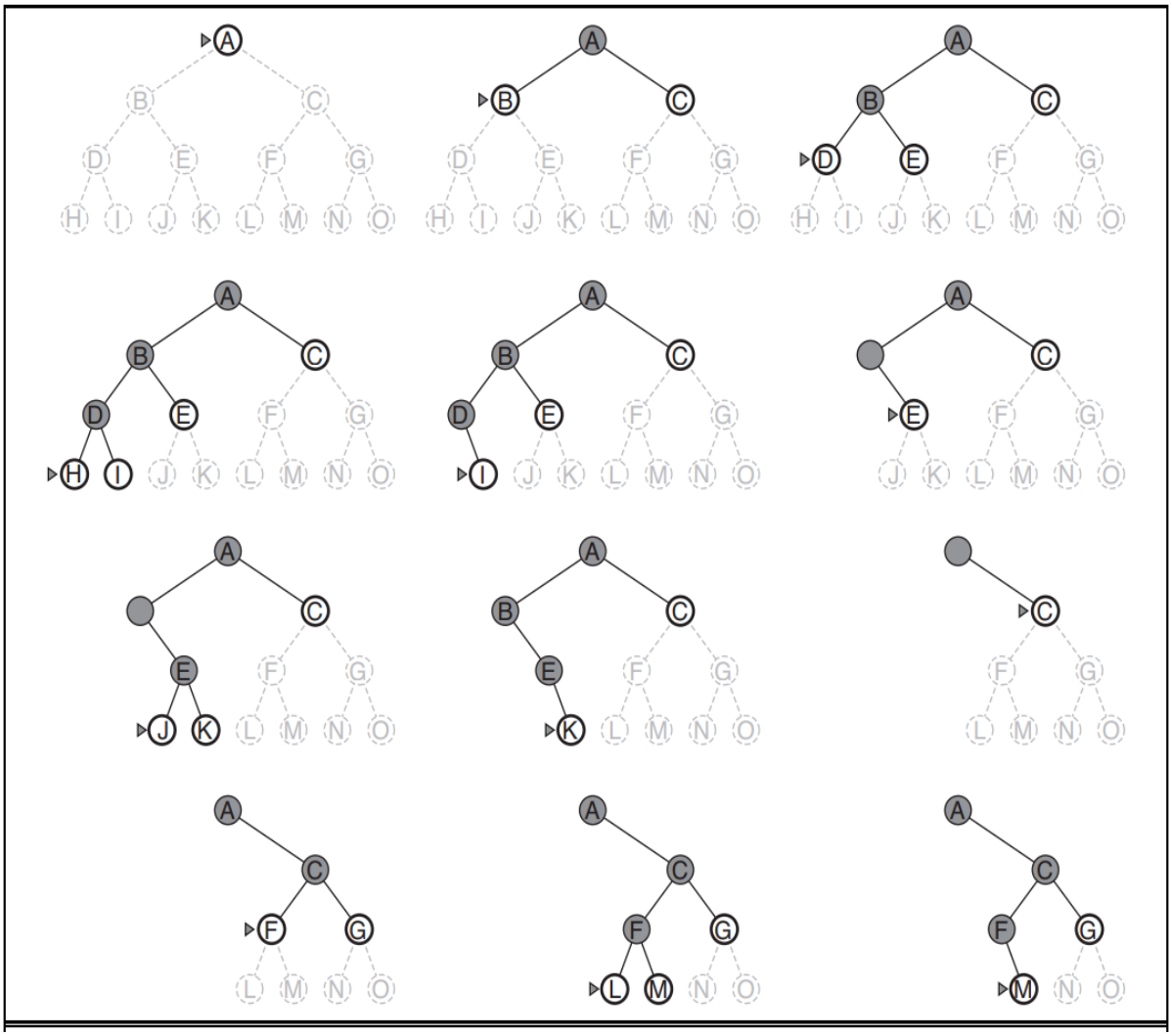
Initialization:
Push the root node onto the LIFO queue.

Traversal:

While the LIFO queue is not empty:

- Pop a node from the LIFO queue.
- Visit the popped node.
- Push the **right child** onto the LIFO queue (if exists).
- Push the **left child** onto the LIFO queue (if exists).

Figure below illustrates the Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.



DFS Resultant Path for the Goal **M** in the above figure is **A->C->F->M**

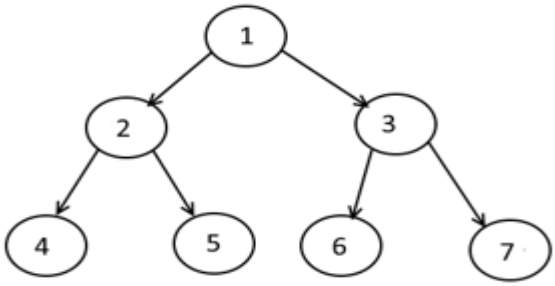
Example: Consider the following binary tree:

```
      A
    /   \
   B     C
  / \   / \
 D   E F   G
```

Step	Description	Visited Nodes	LIFO QUEUE
1	Push 'A' onto the LIFO queue	{}	[A]
2	Pop 'A', visit it, and push its children 'B' and 'C' onto the LIFO queue	{A}	[C, B]
3	Pop 'B', visit it, and push its children 'E' and 'D' onto the LIFO queue.	{A, B}	[C, E, D]
4	Pop 'D', visit it (no children to push)	{A,B,D}	[C, E]
5	Pop 'E', visit it (no children to push)	{A,B,D,E}	[C]
6	Pop 'C', visit it, and push its children 'F' and 'G'	{A,B,D,E,C}	[G,F]
7	Pop 'F', visit it (no children to push)	{A,B,D,E,C,F}	[G]
8	Pop 'G', visit it (no children to push)	{A,B,D,E,C,F,G}	[]

Resulting DFS Traversal Order: **A,B,D,E,C,F,G**

Exercise: Apply the DFS for the following (Assume the Goal is 6)



1.7.3 Time and Space Complexity of BFS and DFS

Strategy	Graph		Binary Tree	
	Time Complexity	Space Complexity	Time Complexity	Space Complexity
BFS	O(V + E)	O(V)	$O(b^d)$	$O(b^d)$
DFS	O(V + E)	O(V)	$O(b^d)$	O(bd)