1. **Introduction**

The field of **artificial intelligence**, or AI, it attempts not just to understand but also to *build* intelligent entities. Work in AI started soon after World War II, and the name itself was coined in 1956. Along with molecular biology, AI is regularly refereed as the “field I would most like to be in” by scientists in other disciplines.

AI currently includes variety of subfields, ranging from the general (learning and perception) to the specific, such as playing chess, proving mathematical theorems, writing poetry, driving a car on a crowded street, and diagnosing diseases. AI is relevant to any intellectual task; it is truly a universal field.

1. **What is AI?**

Some definitions of artificial intelligence, organized into four categories

|  |  |
| --- | --- |
| **Thinking Humanly**  “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) | **Thinking Rationally**  “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**  “The art of creating machines that per- form 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) | **Acting Rationally**  “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) |
|  | |

Figure 1: Some definitions of AI, organized into four cases

* The definitions on top are concerned with *thought processes* and *reasoning*, whereas the ones on the bottom address *behavior*
* The definitions on the left measure success in terms of fidelity to *human* performance, whereas the ones on the right measure against an *ideal* performance measure, called **rationality**

### Acting humanly: The Turing Test approach

### The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence.

### An interrogator will ask a set of written question to computer. If the interrogator cannot identify whether the written response is from a human or computer then the computer passes the test.

### The computer would need to possess the following capabilities to pass the test:

### Natural language processing to enable it to communicate successfully in English;

### Knowledge representation to store what it knows or hears;

### Automated reasoning to use the stored information to answer questions and to draw new conclusions;

### Machine learning to adapt to new circumstances and to detect and extrapolate patterns

* Turing’s test avoided direct physical interaction between the interrogator and the computer, because *physical* simulation of a person is unnecessary for intelligence
* Total turning test includes a video signal so that the interrogator can test the subject’s perceptual abilities, as well as the opportunity for the interrogator to pass physical objects “through the hatch.”
* To pass the total Turing Test, the computer will need

1. **Computer vision** to perceive objects, and
2. **Robotics** to manipulate objects and move about.

* AI researchers have not concentrated on Turing Test, believing that it is more important to study the underlying principles of intelligence than to duplicate an exemplar

1. **Thinking humanly: The cognitive modeling approach**

* For a program to think like human we must determine how humans think.
* There are three ways to find how a human think.
* introspection—trying to catch our own thoughts as they go by;
* psychological experiments—observing a person in action; and through
* brain imaging—observing the brain in action.
* Having a sufficiently accurate theory of the mind helps to express the theory as a computer program.
* If the program’s input–output behavior matches corresponding human behavior, that is evidence that some of the program’s mechanisms could also be operating in humans.
* The interdisciplinary field of **cognitive science** brings together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind.
* The cognitive science and AI have helped each other especially in computer vision, which uses neurophysiologic evidences to form computational models.

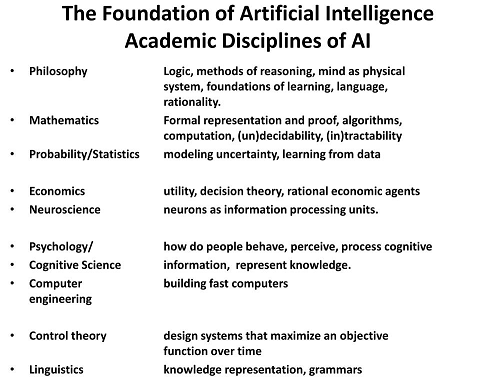
1. **Thinking rationally: The “laws of thought” approach**

* Aristotle’s **syllogisms** provided patterns for argument structures that always gave correct conclusions when given correct premises. For example, “Socrates is a man; all men are mortal; therefore, Socrates is mortal.” These laws of thought were supposed to govern the operation of the mind; their study initiated the field called **logic**.
* Logicians in the 19th century developed a precise notation for statements about all kinds of objects in the world and the relations among them
* By 1965, programs existed that could, in principle, solve *any* solvable problem described in logical notation.
* The so-called **logicist** tradition within artificial intelligence hopes to build on such programs to create intelligent systems.
* There are two problems with this approach
* First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain.
* Second, there is a big difference between solving a problem “in principle” and solving it in practice.

1. **Acting rationally: The rational agent approach**

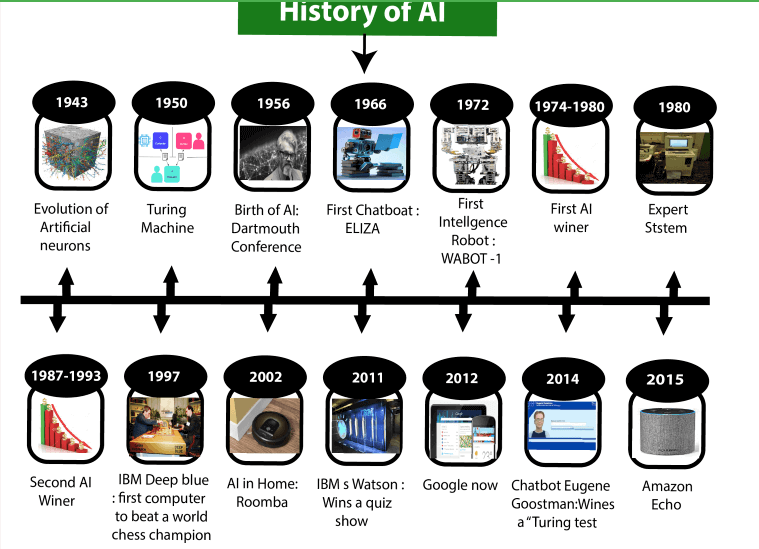
* An **agent** is just something that acts.
* Computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals
* A **rational agent** is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.
* Making inference is part of being rational agent.
* Making only Inference is not complete rationality. There are also ways of acting rationally that cannot be said to involve inference. For example taking hand from a hot stove is a reflex action that is usually more successful than a slower action taken after careful thought.
* Knowledge representation and reasoning enable agents to reach good decisions.
* The rational-agent approach has two advantages over the other approaches.
* First, it is more general than the “laws of thought” approach because correct inference is just one of several possible mechanisms for achieving rationality.
* Second, it is more agreeable to scientific development than are approaches based on human behavior or human thought.
* The standard of rationality is mathematically well defined and completely general, and can be “unpacked” to generate agent designs that provably achieve it.

1. **Foundations of Artificial Intelligence**

****

1. **History of Artificial Intelligence**

Artificial Intelligence is not a new word and not a new technology for researchers. This technology is much older than you would imagine. Even there are the myths of Mechanical men in Ancient Greek and Egyptian Myths. Following are some milestones in the history of AI which defines the journey from the AI generation to till date development.



1. **Maturation of Artificial Intelligence (1943-1952)**

* **Year 1943:** The first work which is now recognized as AI was done by Warren McCulloch and Walter pits in 1943. They proposed a model of **artificial neurons**.
* **Year 1949:** Donald Hebb demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called **Hebbian learning**.
* **Year 1950:** The Alan Turing who was an English mathematician and pioneered Machine learning in 1950. Alan Turing publishes **"Computing Machinery and Intelligence"** in which he proposed a test. The test can check the machine's ability to exhibit intelligent behavior equivalent to human intelligence, called a **Turing test**.

1. **The birth of artificial intelligence (1952-1956)**

* **Year 1955:** An Allen Newell and Herbert A. Simon created the "first artificial intelligence program"Which was named as **"Logic Theorist"**. This program had proved 38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.
* **Year 1956:** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.

At that time high-level computer languages such as FORTRAN, LISP, or COBOL were invented. And the enthusiasm for AI was very high at that time.

1. **The golden years-Early enthusiasm, great expectations (1956–1974)**

* **Year 1966:** The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first chatbot in 1966, which was named as ELIZA.
* **Year 1972:** The first intelligent humanoid robot was built in Japan which was named as WABOT-1.

1. **The first AI winter(1974–1980)**

* The duration between years 1974 to 1980 was the first AI winter duration. AI winter refers to the time period where computer scientist dealt with a severe shortage of funding from government for AI researches.
* During AI winters, an interest of publicity on artificial intelligence was decreased.

1. **A Boom of AI (1980–1987)**

* **Year 1980:** After AI winter duration, AI came back with "Expert System". Expert systems were programmed that emulate the decision-making ability of a human expert.
* In the Year 1980, the first national conference of the American Association of Artificial Intelligence **was held at Stanford University**.

1. **The second AI winter (1987–1993)**

* The duration between the years 1987 to 1993 was the second AI Winter duration.
* Again Investors and government stopped in funding for AI research as due to high cost but not efficient result. The expert system such as XCON was very cost effective.

1. **Emergence of intelligent Agents(1993-2011)**

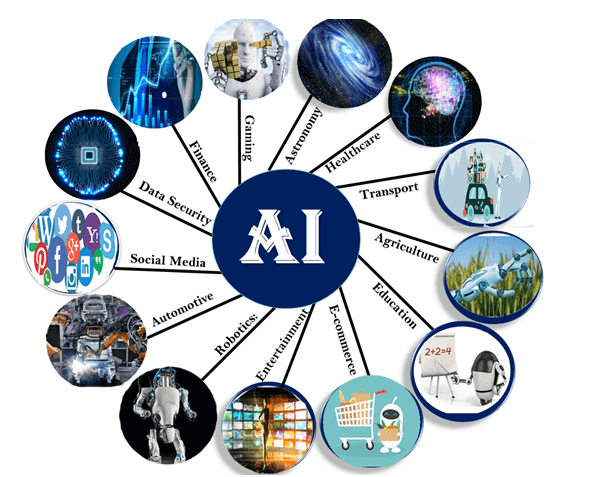
* **Year 1997:** In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
* **Year 2002:** for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.
* **Year 2006:** AI came in the Business world till the year 2006. Companies like Facebook, Twitter, and Netflix also started using AI.

## Deep learning, big data and artificial general intelligence (2011-present)

* **Year 2011:** In the year 2011, IBM's Watson won jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.
* **Year 2012:** Google has launched an Android app feature "Google now", which was able to provide information to the user as a prediction.
* **Year 2014:** In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."
* **Year 2018:** The "Project Debater" from IBM debated on complex topics with two master debaters and also performed extremely well.
* Google has demonstrated an AI program "Duplex" which was a virtual assistant and which had taken hairdresser appointment on call, and lady on other side didn't notice that she was talking with the machine.
* Now AI has developed to a remarkable level. The concept of Deep learning, big data, and data science are now trending like a boom. Nowadays companies like Google, Facebook, IBM, and Amazon are working with AI and creating amazing devices. The future of Artificial Intelligence is inspiring and will come with high intelligence.
  1. **Applications of AI**

Artificial Intelligence has various applications in today's society. It is becoming essential for today's time because it can solve complex problems with an efficient way in multiple industries, such as Healthcare, entertainment, finance, education, etc. AI is making our daily life more comfortable and fast.

Following are some sectors which have the application of Artificial Intelligence:

****

### AI in Astronomy

* Artificial Intelligence can be very useful to solve complex universe problems. AI technology can be helpful for understanding the universe such as how it works, origin, etc.

2. AI in Healthcare

* In the last, five to ten years, AI becoming more advantageous for the healthcare industry and going to have a significant impact on this industry.
* Healthcare Industries are applying AI to make a better and faster diagnosis than humans. AI can help doctors with diagnoses and can inform when patients are worsening so that medical help can reach to the patient before hospitalization.

### 3. AI in Gaming

* AI can be used for gaming purpose. The AI machines can play strategic games like chess, where the machine needs to think of a large number of possible places.

### 4. AI in Finance

* AI and finance industries are the best matches for each other. The finance industry is implementing automation, chatbot, adaptive intelligence, algorithm trading, and machine learning into financial processes.

### 5. AI in Data Security

* The security of data is crucial for every company and cyber-attacks are growing very rapidly in the digital world. AI can be used to make your data more safe and secure. Some examples such as AEG bot, AI2 Platform,are used to determine software bug and cyber-attacks in a better way.

### 6. AI in Social Media

* Social Media sites such as Facebook, Twitter, and Snapchat contain billions of user profiles, which need to be stored and managed in a very efficient way. AI can organize and manage massive amounts of data. AI can analyze lots of data to identify the latest trends, hashtag, and requirement of different users.

### 7. AI in Travel & Transport

* AI is becoming highly demanding for travel industries. AI is capable of doing various travel related works such as from making travel arrangement to suggesting the hotels, flights, and best routes to the customers. Travel industries are using AI-powered chatbots which can make human-like interaction with customers for better and fast response.

### 8. AI in Automotive Industry

* Some Automotive industries are using AI to provide virtual assistant to their user for better performance. Such as Tesla has introduced TeslaBot, an intelligent virtual assistant.
* Various Industries are currently working for developing self-driven cars which can make your journey more safe and secure.

### 9. AI in Robotics:

* Artificial Intelligence has a remarkable role in Robotics. Usually, general robots are programmed such that they can perform some repetitive task, but with the help of AI, we can create intelligent robots which can perform tasks with their own experiences without pre-programmed.
* Humanoid Robots are best examples for AI in robotics, recently the intelligent Humanoid robot named as Erica and Sophia has been developed which can talk and behave like humans.

### 10. AI in Entertainment

* We are currently using some AI based applications in our daily life with some entertainment services such as Netflix or Amazon. With the help of ML/AI algorithms, these services show the recommendations for programs or shows.

### 11. AI in Agriculture

* Agriculture is an area which requires various resources, labor, money, and time for best result. Now a day's agriculture is becoming digital, and AI is emerging in this field. Agriculture is applying AI as agriculture robotics, solid and crop monitoring, predictive analysis. AI in agriculture can be very helpful for farmers.

### 12. AI in E-commerce

* AI is providing a competitive edge to the e-commerce industry, and it is becoming more demanding in the e-commerce business. AI is helping shoppers to discover associated products with recommended size, color, or even brand.

### 13. AI in education:

* AI can automate grading so that the tutor can have more time to teach. AI chatbot can communicate with students as a teaching assistant.
* AI in the future can be work as a personal virtual tutor for students, which will be accessible easily at any time and any place.
  1. **Solving Problems by Searching**
* It is the process which describes how an agent can find a sequence of actions that achieves its goals when no single action will do.

1. **Problem Solving Agents**

* **Problem-solving agent is a** goal-based agent
* Problem-solving agents use atomic representations, that is, states of the world are considered as wholes, with no internal structure visible to the problem solving algorithms.
* **Goal formulation**, based on the current situation and the agent’s performance measure, is the first step in problem solving.
* **Problem formulation** is the process of deciding what actions and states to consider, given a goal
* An agent with several immediate options of unknown value can decide what to do by first examining future actions that eventually lead to states of known value.
* “Examining future actions” indicates it should be more specific about properties of environment.
* The process of looking for a sequence of actions that reaches the goal is called **search**.
* A search algorithm takes a problem as input and returns a **solution** in the form of an action sequence.
* Once a solution is found, the actions it recommends can be carried out. This is called the **execution** phase. Thus, we have a simple “formulate, search, execute” design for the agent, as shown in Figure

|  |
| --- |
| **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: A simple problem-solving agent.

* In figure 3, 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.
* After formulating a goal and a problem to solve, the agent calls a search procedure to solve it. It then uses the solution to guide its actions, doing whatever the solution recommends as the next thing to do—typically, the first action of the sequence—and then removing that step from the sequence. Once the solution has been executed, the agent will formulate a new goal.
* Agent ignores its percepts when choosing an action because it knows in advance what they will be. Thus the system is open loop.

1. **Well-defined problems and solutions**

* A **problem** can be defined by five components

1. The **initial state** that the agent starts in. Example: *In(Karnataka)*
2. A description of the possible **actions** available to the agent.

For a state **s,**

**ACTIONS(s)** 🡪 returns the set of actions that can be executed in s. These actions are **applicable** in s.

**Example:** *In(Karnataka)*

{*Go*(*Mandya*), *Go*(*Mysore*),*Go*(*Bangalore*)}

1. **Transition model** gives a description of the possible **actions** available to the agent. It is represented by a function **RESULT(s, a).**

**RESULT(s, a) –** gives the next(successor) state after executing the action **a,** in state **s.**

**Example: RESULT(***In(Karnataka), Go*(*Mandya*)) = *In*(*Mandya*)

The initial state, actions, and transition model define the **state space** of theproblem.

State space gives the set of all states reachable from the initial state by any sequence of actions. The state space forms a directed network or **graph** in which the nodes are states and the links between nodes are actions.

A **path** in the state space is a sequence of states connected by a sequence of actions.

1. The **goal test**, which determines whether a given state is a goal state.
2. A **path cost** function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure.

The **step cost** of taking action a in state s to reach state is denoted by

**c(s, a, ).**

A **solution** to a problem is an action sequence that leads from the initial state to a goal state. **Solution quality** is measured by the path cost function, and an optimal solution has the lowest path cost among all solutions.

1. **Formulating Problems**

Formulation of the problem is a *model*—an abstract mathematical description in terms of the initial state, actions, transition model, goal test, and path cost. The process of removing detail from a representation is called **abstraction.** The choice of a good abstraction involves removing as much detail as possible while retaining validity and ensuring that the abstract actions are easy to carry out.

1. **Example Problems**

There are two set where problem solving approaches are applied viz- toy problem and real world problem. Toy problem are cases which are used by researchers to compare the performance of algorithms. A real world problem is one whose solutions people actually care.

* **Toy problems Examples:**

1. **Vacuum world:**

* **States:** An environment with locations has states. In this example the number of locations is two so the possible world states are states.
* **Initial State:** Any state can be considered as the initial state.
* **Actions:** in this environment each state has just three actions: *Left*, *Right*, and *Suck*.
* **Transition model:** The actions have their expected effects, except that moving *Left* in the leftmost square, moving *Right* in the rightmost square, and *Suck*ing in a clean square have no effect. The complete state space is shown in Figure 4



Figure 4: The state space for the vacuum world. Links denote actions: L=Left, R=Right, S=Suck

* **Goal test**: This checks whether all the squares are clean.
* **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.
* The abstraction included here is that, compared with the real world, this toy problem has discrete locations, discrete dirt, reliable cleaning, and it never gets any dirtier.

1. **The 8-puzzle:**

Figure 5: A typical instance of the 8-puzzle

* **States:** Specifies the location of each of the eight tiles and the blank in one of the nine squares.
* **Initial state:** Any state can be designated as the initial state.
* **Actions:** The simplest formulation defines the actions as movements of the blank space *Left*, *Right*, *Up*, or *Down*. Different subsets of these are possible depending on where the blank is.
* **Transition model**: Given a state and action, this returns the resulting state.
* **Goal test**: This checks whether the state matches the goal configuration shown in Figure 5.
* **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.
* The abstraction included here is that the intermediate locations when block slides is ignored. Shaking the board, exacting the pieces with knife and putting them back when pieces get stuck are ignored.

1. **8-queens problem**

**The goal** 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.)

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

In this formulation, we havepossible sequences to investigate. A better formulation would prohibit placing a queen in any square that is already attacked:

* **States**: All possible arrangements of n queens (0 ≤ n ≤ 8), one per column in the leftmost n columns, with no queen attacking another.
* **Actions**: Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.

This formulation reduces the 8-queens state space from 1.8×1014 to just 2,057, and solutions are easy to find.



Figure 6: Almost solution to 8 queens problem

1. **To Reach desired positive number:**

Knuth showed that starting with the number 4, a sequence of factorial, square root, and floor operations will reach any desired positive integer. For example, we can reach 5 from 4 as follows:

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

1. Routing Problem:The airline travel problems that must be solved by a travel-planning Web site:

* **States**: Each state 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.
  + **Initial state**: This is specified by the user’s query.
  + **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.
  + **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.
  + **Goal test**: Are we at the final destination specified by the user?
  + **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. **Touring problems** are closely related to route-finding problems, but with an important difference. Each state must include not just the current location but also the set of cities the agent has visited. The goal test would check whether the agent has visited all the cities.

*In(Karnataka) Visited*({*Mandya*, *Mysore*,*Bangalore*})

1. The **traveling salesperson problem** (TSP) is a touring problem in which each city must be visited exactly once. The aim is to find the *shortest* tour.
2. A **VLSI layout** problem requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield. The layout problem comes after the logical design phase and is usually split into two parts: **cell layout** and **channel routing**. In cell layout, the primitive components of the circuit are grouped into cells, each of which performs some recognized function. Each cell has a fixed footprint (size and shape) and requires a certain number of connections to each of the other cells. The aim is to place the cells on the chip so that they do not overlap and so that there is room for the connecting wires to be placed between the cells. Channel routing finds a specific route for each wire through the gaps between the cells.
3. **Robot navigation** is a generalization of the route-finding problem. For a circular robot moving on a flat surface, the space is essentially two-dimensional. When the robot has arms and legs or wheels that must also be controlled, the search space becomes many-dimensional. Advanced techniques are required just to make the search space finite.
4. **Automatic assembly sequencing** of complex objects by a robot was first demonstrated by FREDDY (Michie, 1972). In assembly problems, the aim is to find an order in which to assemble the parts of some object. If the wrong order is chosen, there will be no way to add some part later in the sequence without undoing some of the work already done. The generation of legal actions is the expensive part of assembly sequencing.
5. Another important assembly problem is **protein design**, in which the goal is to find a sequence of amino acids that will fold into a three-dimensional protein with the right properties to cure some disease.
6. **Searching for Solutions**

A solution is an action sequence, so search algorithms work by considering various possible action sequences. The state space of the problem is represented as a search tree. The branches are actions, nodes represent the state, and the initial state is represented as root node.

Figure shows the first few steps in growing the search tree for finding a route from Bangalore to Udupi. The root node of the tree corresponds to the initial state, *In(Bangalore)*.

The first step is to test whether this is a goal state. if the goal state is not reached then need to consider taking various actions. We do this by **expanding** the current state; that is applying each legal action to the current state, thereby **generating** a new set of states. In this case, we add three branches from the **parent node** *In(Bangalore)*.leading to three new **child nodes**: *In(Mysuru), In(Mandya),* and *In(Tumkur).* This is the essence of search—following up one option now and putting the others aside for later, in case the first choice does not lead to a solution. L**eaf node** is a node with no children in the tree. The set of all leaf nodes available for expansion at any given point is called the **frontier(Open list).** The process of expanding nodes on the frontier continues until either a solution is found or there are no more states to expand. The general TREE-SEARCH algorithm is shown informally in Figure. Search algorithms all share this basic structure; they vary primarily according to how they choose which state to expand next—the so-called **search strategy.** Loopy paths are a special case of the more general concept of redundant paths, which exist whenever there is more than one way to get from one state to another. The way to avoid exploring redundant paths is to remember where one has been. To do this, we augment the TREE-SEARCH algorithm with a data structure called the **explored set** (also known as the **closed list**), which remembers every expanded node. Newly generated nodes that match previously generated nodes—ones in the explored set or the frontier—can be discarded instead of being added to the frontier.

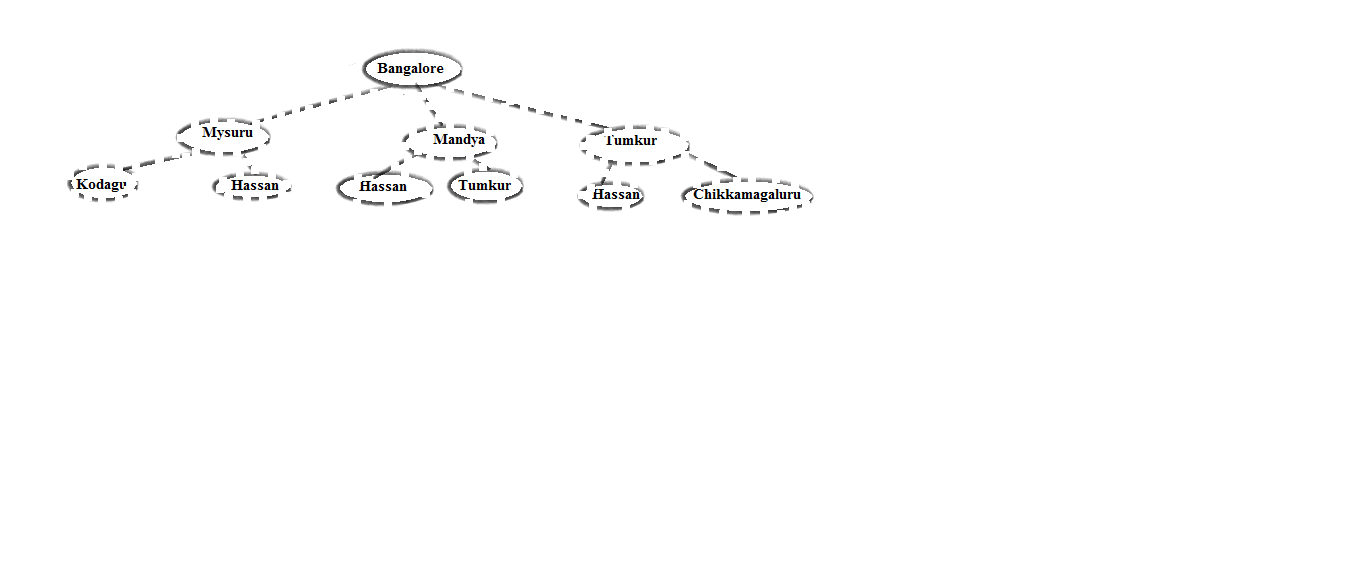


Figure 7: Initial state

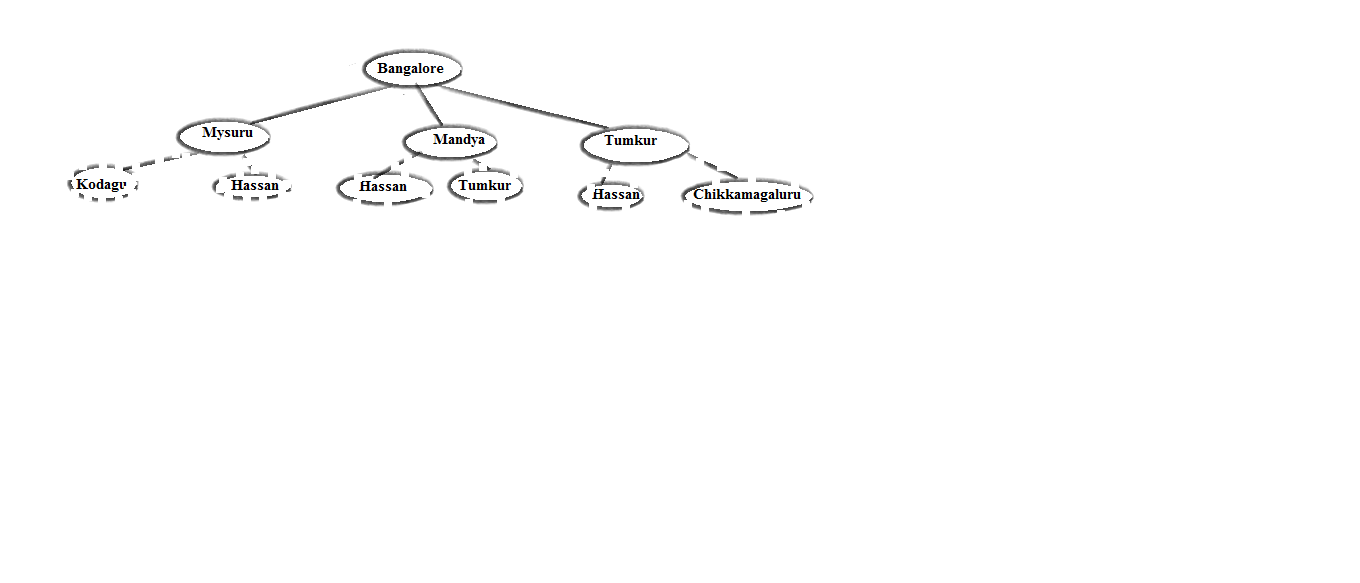


Figure 8: After Expanding Bangalore

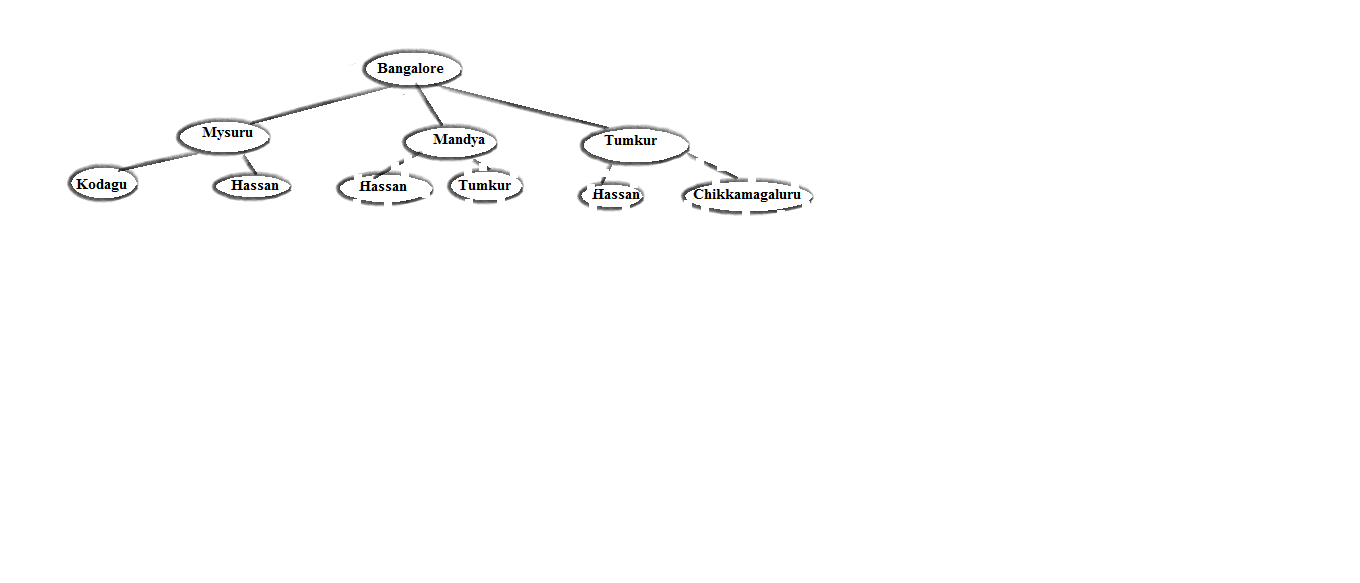


Figure 9:After expanding Mysuru

|  |
| --- |
| **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  **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*** |
| Figure: An informal description of the general tree-search and graph-search algorithms. |

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 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

Given the components for a parent node, it is easy to see how to compute the necessary components for a child node. The function CHILD-NODE takes a parent node and an action and returns the resulting child node:

|  |
| --- |
| **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) |

The frontier needs to be stored in such a way that the search algorithm can easily choose the next node to expand according to its preferred strategy. The appropriate data structure for this is a **queue**. The operations on a queue are as follows:

* EMPTY?(queue) returns true only if there are no more elements in the queue.
* POP(queue) removes the first element of the queue and returns it.
* INSERT(element, queue) inserts an element and returns the resulting queue.

Queues are characterized by the *order* in which they store the inserted nodes. Three common variants are the first-in, first-out or **FIFO queue, LIFO queue (**also known as **a stack)** and the **priority queue.**

1. **Measuring problem-solving performance**

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

* **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
* **Optimality**: Does the strategy find the optimal solution?
* **Time complexity**: How long does it take to find a solution?
* **Space complexity**: How much memory is needed to perform the search?

In AI, the graph is often represented *implicitly* by the initial state, actions, and transition model and is frequently infinite. For these reasons, complexity is expressed in terms of three quantities:

* b, the **branching** **factor** or maximum number of successors of any node;
* d, the **depth** of the shallowest goal node (i.e., the number of steps along the path from the root);
* m, the maximum length of any path in the state space.

Time is often measured in terms of the number of nodes generated during the search, and space in terms of the maximum number of nodes stored in memory.

To find the effectiveness of a search algorithm, we can consider just the **search cost**— which typically depends on the time complexity but can also include a term for memory usage—or we can use the **total cost**, which combines the search cost and the path cost of the solution found.

1. **Uniformed Search Strategies**

Uniformed or blind search means that the strategies have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state.

1. **Breadth-first search**

* Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadth wise in a tree or graph, so it is called breadth-first search.
* BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
* The breadth-first search algorithm is an example of a general-graph search algorithm.
* Breadth-first search implemented using FIFO queue data structure.

**Advantages:**

* BFS will provide a solution if any solution exists.
* If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

**Disadvantages:**

* It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
* BFS needs lots of time if the solution is far away from the root node.

**Example:**

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

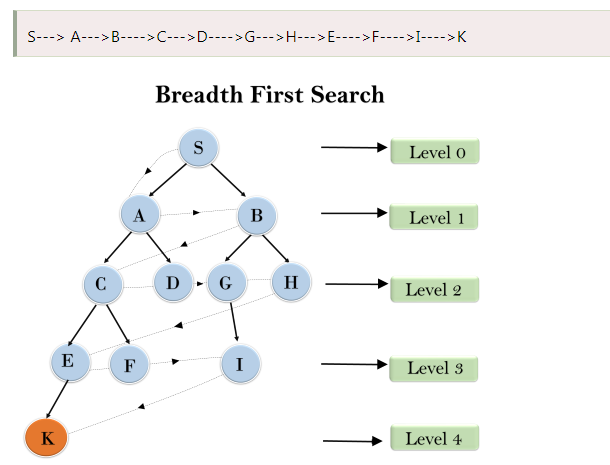


Figure :Breadth-first search on a simple binary tree.

* Pseudocode is given in the table

|  |
| --- |
| **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) |
| Breadth-first search on a graph. |

* **Time Complexity:** Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

**T (b) = 1+b2+b3+.......+ bd= O (bd)**

* **Space Complexity:** Space complexity of BFS algorithm is given by the Memory size of frontier which is O(bd).
* **Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.
* **Optimality:** BFS is optimal if path cost is a non-decreasing function of the depth of the node.

1. **Uniform-cost search**

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

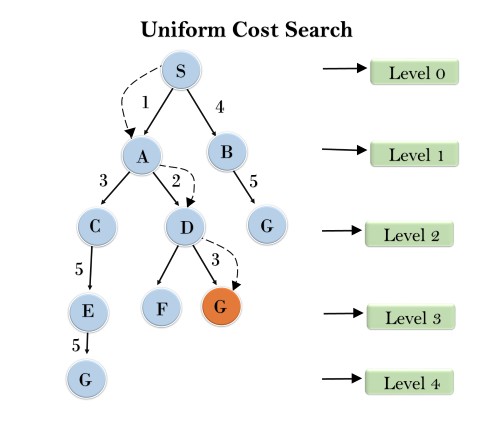
**Advantages:**

* Uniform cost search is optimal because at every state the path with the least cost is chosen.

**Disadvantages:**

* It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

**Example**



* **Completeness:**

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

* **Time Complexity:**

Let C\* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = Here we have taken +1, as we start from state 0 and end to

Hence, the worst-case time complexity of Uniform-cost search is is

* **Space Complexity:**

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is

* **Optimal:**

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

* The pseudocode is shown below

|  |
| --- |
| **function** UNIFORM-COST-SEARCH(problem) **returns** a solution, or failure  node ←a node with STATE = problem.INITIAL-STATE, PATH-COST = 0  frontier ←a priority queue ordered by PATH-COST, with node as the only element  explored ←an empty set  **loop do**  **if** EMPTY?( frontier) **then return** failure  node←POP( frontier ) /\* chooses the lowest-cost node in frontier \*/  **if** problem.GOAL-TEST(node.STATE) **then return** SOLUTION(node)  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**  frontier ←INSERT(child , frontier )  **else if** child .STATE is in frontier with higher PATH-COST **then**  replace that frontier node with child |
| Uniform-cost search on a graph |

* This algorithm has two other significant differences from breadth-first search.
* The goal test is applied to a node when it is *selected for expansion*
* A test is added in case a better path is found to a node currently on the frontier

An example of working is shown below

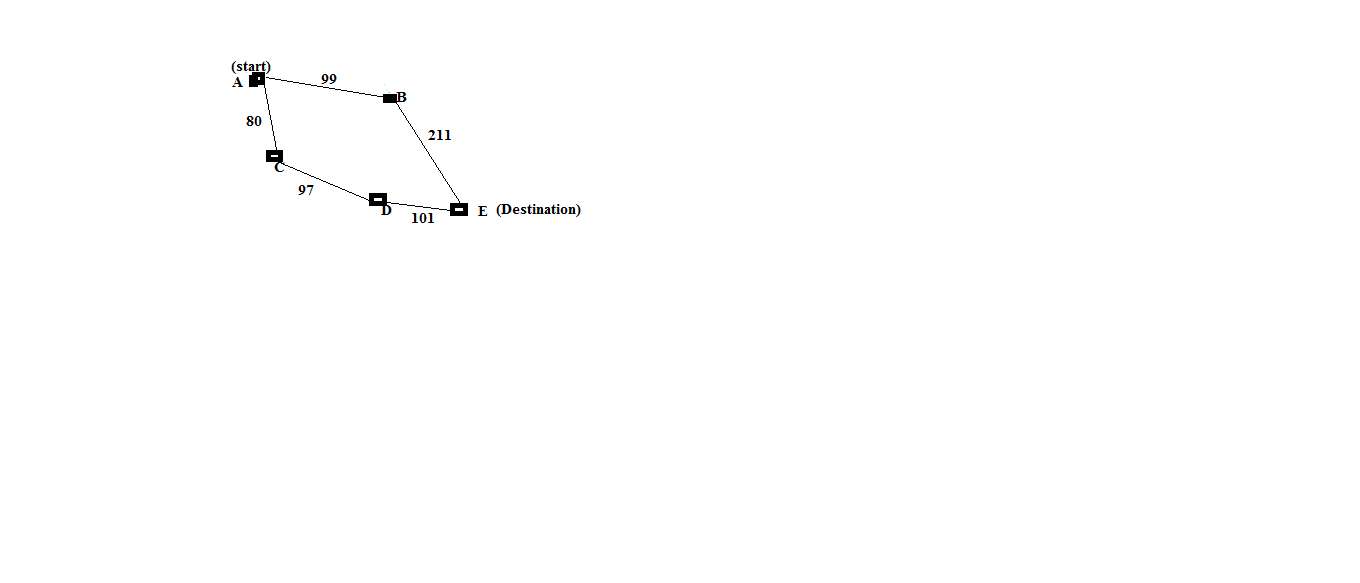


Figure 12: uniform cost search example

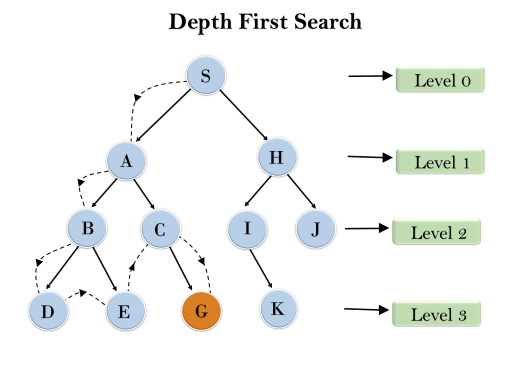
* The successors of node A are B and C with cost values 99 and 80 respectively.
* The least cost node is C with cost 80. Node C is expanded with D as 80+97=177.
* Now the least cost node is B with cost 99. It is expanded with E as 99+211=310.
* Even though goal is reached, the uniform cost search keeps its search, it will choose D and expand as 177+101=278.
* Now the algorithm checks to see if this new path is better than the old one; it is, so the old one is discarded.
* Uniform-cost search is optimal in general. Uniform-cost search expands nodes in order of their optimal path cost.

1. **Depth-first search**

* Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
* It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
* DFS uses a stack data structure for its implementation.
* The process of the DFS algorithm is similar to the BFS algorithm.
* Backtracking is an algorithm technique for finding all possible solutions using recursion.
* **Advantage:**
* DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
* It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).
* **Disadvantage:**
* There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
* DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

### Example:

* In the below search tree, the flow of depth-first search is shown, and it will follow the order as:
* Root node--->Left node ----> right node.
* It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



* **Completeness:** DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.
* **Time Complexity:** Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

**T(n)= 1+ n2+ n3 +.........+ nm=O(nm)**

**Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)**

* **Space Complexity:** DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.
* **Optimal:** DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node