Ahmedabad Institute of Technology

1.

CE /IT Department

Hand Book

ARTIFICIAL INTELLIGENCE

(2180703)

Year: 2020-2021

Prepared By: Prof Dhaval Khatri, CE & IT Dept., AIT

Index

Ahmedabad Institute of Technology

Artificial Intelligence

Prof. Dhaval Khatri

What is Artificial Intelligence ?

- According to the father of Artificial Intelligence, John McCarthy, it is *"The science and engineering of making intelligent machines, especially intelligent computer programs".*

- Artificial Intelligence is a way of **making a computer, a computer-controlled robot, or a software think intelligently**, in the similar manner the intelligent humans think.

- AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

What is Artificial Intelligence ?

- Artificial intelligence is the study of how to make computers do things which, at moment people do better.
- Artificial Intelligence is composed of two words **Artificial** and **Intelligence**, where Artificial defines *"man-made,"* and intelligence defines *"thinking power"*, hence AI means *"a man-made thinking power."*

So, we can define AI as:

- "It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions"
- Artificial Intelligence exists when a machine can have human based skills such as learning, reasoning, and solving problems

- From the perspective of intelligence, artificial intelligence is making machines "intelligent" -- acting as we would expect people to act.
  - The inability to distinguish computer responses from human responses is called the Turing test.
  - Intelligence requires knowledge.
- From a business perspective AI is a set of very powerful tools, and methodologies for using those tools to solve business problems.
- From a programming perspective, AI includes the study of symbolic programming, problem solving, and search.
  - Typically AI programs focus on symbols rather than numeric processing.
  - Problem solving i.e. to achieve a specific goal.
  - Search - rarely access a solution directly. Search may include a variety of techniques.
- It is the science and engineering of making intelligent machines, especially intelligent computer programs.

Why Artificial Intelligence?

- With the help of AI, you can create such software or devices which can solve real-world problems very easily and with accuracy such as health issues, marketing, traffic issues, etc.
- With the help of AI, you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- With the help of AI, you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

AI Problems (Task Domains of AI)

Mundane tasks	Formal tasks	Expert tasks
Perception – Computer Vision – Speech, Voice	Games – Go – Chess (Deep Blue) – Ckeckers	Engineering – Design – Fault Finding – Manufacturing – Monitoring
Natural Language Processing – Understanding – Language Generation – Language Translation	Mathematics – Geometry – Logic – Integration and Differentiation	Scientific Analysis
Common Sense Reasoning	Theorem Proving	Financial Analysis
Planning		Medical Diagnosis
Robot Control		

What is an AI technique?

- Artificial intelligence problems span a very broad spectrum. They appear to have very little in common except that they are hard.

- AI Research of earlier decades results into the fact that intelligence requires knowledge.

- Knowledge possess following properties:
  - It is voluminous.
  - It is not well-organized or well-formatted.
  - It is constantly changing.
  - It differs from data. And it is organized in a way that corresponds to its usage.

What is an AI technique?

- AI technique is a method that exploits knowledge that should be represented in such a way that:
  - Knowledge captures generalization. Situations that share common properties are grouped together. Without this property, inordinate amount of memory and modifications will be required.
  - It can be understood by people who must provide it. Although bulk of data can be acquired automatically, in many AI domains most of the knowledge must ultimately be provided by people in terms they understand.
  - It can easily be modified to correct errors and to reflect changes in the world.
  - It can be used in many situations even though it may not be totally accurate or complete.
  - It can be used to reduce its own volume by narrowing range of possibilities.

What is an AI technique?

➢There are three important AI techniques:

- Search –
  - Provides a way of solving problems for which no direct approach is available.
  - It also provides a framework into which any direct techniques that are available can be embedded.
- Use of knowledge –
  - Provides a way of solving complex problems by exploiting the structure of the objects that are involved.
- Abstraction –
  - Provides a way of separating important features and variations from many unimportant ones that would otherwise overwhelm any process.

Classification of AI

- Weak AI
- Strong AI
- Evolutionary AI

Advantages of Artificial Intelligence

Following are some main advantages of Artificial Intelligence:

- **High Accuracy with less errors:**
- **High-Speed:**
- **High reliability:**
- **Useful for risky areas:**
- **Digital Assistant:**
- **Useful as a public utility:**

Disadvantages of Artificial Intelligence

Following are the disadvantages of AI:

- **High Cost:**
- **Can't think out of the box:**
- **No feelings and emotions:**
- **Increase dependency on machines:**
- **No Original Creativity:**

Applications of AI

AI has been dominant in various fields such as –

- **Gaming**
- **Natural Language Processing**
- **Expert Systems**
- **Vision Systems**
- **Speech Recognition**
- **Handwriting Recognition**
- **Intelligent Robots**

2

Problem Solving Through AI

2.1 INTRODUCTION

In the last chapter, we have tried to explain that Artificial Intelligence is the science and technology applied to make machines intelligent. The ultimate aim of researchers is to develop universal intelligent system to match the intelligence capabilities of human beings. In this regard, lot of progress has been made and considerable amount of success has been achieved, although, universal intelligent system is still a dream. Scientists have developed techniques to use AI in a limited domain and have successfully developed many AI systems, which work in a problem specific domain and show expertise not only matching those of human experts, but also exceeding those in many ways and in many applications. It should be kept in mind that for the time being, the most important application of AI is to develop intelligent systems to solve real world problems, which otherwise take considerable amount of time and human efforts, and hence, become uneconomical as well as inefficient at times. Hence, problem solving becomes major area of study, which involves methods and various techniques used in problem solving using AI.

In this chapter, we will discuss the methods of solving real world problems using Artificial Intelligence (AI) techniques. In real world, there are different types of problems. We will try to explain various characteristics of problems and their solution methodologies.

The method of solving problem through AI involves the process of defining the search space, deciding start and goal states and then finding the path from start state to goal state through search space. The movement from start state to goal state is guided by set of rules specifically designed for that particular problem (sometimes called production rules). The production rules are nothing but valid moves described by the problems.

Let us first discuss some terms related with AI problem solution methodology:

Problem — It is the question which is to be solved. For solving a problem it needs to be precisely defined. The definition means, defining the start state, goal state, other valid states and transitions.

techniques are devised for this phase. These KR methodologies have their respective advantages and disadvantages, and specific techniques are suitable for specific applications. Various KR techniques are described in detail in following chapters.

2.1.4 Finding the Solution

After representation of the problem and related knowledge in the suitable format, the appropriate methodology is chosen which uses the knowledge and transforms the start state to goal state. The techniques of finding the solution are called search techniques. Various search techniques are developed for this purpose. They are dealt-with in detail in one of the following sections and also in next chapter.

2.2 REPRESENTATION OF AI PROBLEMS

In this section, we would learn the technical aspects of problem representation required from computational perspective. From this orientation, the representation of AI problems can be covered in following four parts:

1. *A lexical part*: that determines which symbols are allowed in the representations of the problem. Like the normal meaning of the lexicon, this part abstracts all fundamental features of the problem.
2. *A structural part*: that describes constraints on how the symbols can be arranged. This corresponds to finding out possibilities required for joining these symbols and generating higher structural unit.
3. *A procedural part*: that specifies access procedures that enable to create descriptions, to modify them, and to answer questions using them.
4. *A semantic part*: that establishes a way of associating meaning with the descriptions.

Let us understand these steps by an example presented below:

We are given the problem of "playing chess". To define the problem, we should specify the starting position of chess board, the rules that define the legal moves and the board positions that represent a win for one side or the others. Here, the:

1. Lexical part contains the board position, which may be an 8 x 8 array where each position contains a symbol indicating an appropriate chess piece in the official chess opening position. The goal is any board position in which opponent does not have a legal move and his king in under attack.
2. The structural part describes the legal moves. The legal moves provide the

way of getting from an initial state to a goal state. They are described as set of rules consisting of a left hand side that serves as a pattern to be matched against the current board position and a right hand side that describes changes to be made to board position to reflect the move.

3. The procedural part will be methods for applying appropriate rule for winning the game. It will include representing the "set of winning moves" of standard chess playing. Out of all legal moves, only those moves, which bring the board position to a winning position of respective player arecaptured and stored in procedural part.

   However, in chess the total '*legal moves'* are of the order of 10^{120}. Such a large number of moves are difficult to be written, so only '*useful moves'* are written. It should be noted that there is a difference between legal moves and useful moves. The legal moves are all those moves which are permissible according to game rules and useful moves will be thoselegal moves which bring the game in a winning position. Hence the 'useful moves' will be a subset of 'legal moves'. The state space is total validstates possible in a problem and finding a problem solution is "starting atan initial state, using a set of rules to move from one state to another and attempting to end up in one of a set of final states."
4. The semantic part is not required in "chess game" because, there is no hidden meaning associated with any piece and all the move meanings are explicit. However in natural language processing type of applications, where there is "meaning" associated with words and "complete message" associated with sentence, the semantic part captures and stores, the meaning of words and message conveyed by sentence.

Thus, in AI, to design a program for solution, the first step is the creation of a formal and manipulable description of problem itself. This includes performing following activities:

1. Define a state space that contains all the possible configurations of therelevant objects.
2. Specify one or more states within that space, which describe possible situations from which problem solving process may start. These are called initial states.
3. Specify one or more states that would be acceptable as solutions to the problem. These are called goal states.
4. Specify a set of rules that describes the action (operators) available.

2.3 PRODUCTION SYSTEM

In the above section, we have discussed basic aspects of AI problem solution. An

AI system developed for solution of any problem is called production system. Once the problem is defined, analyzed and represented in a suitable formalism, the production system is used for application of rules and obtaining the solution. A production system consists of following components:

1. *A set of production rules*, which are of the form P → Q. Each rule consists of a left hand side constituent that represents the current problem state and a right hand side that represents a result or generated output state. A rule is applicable if its left hand side matches with the current problem state. Thus, the left side of the rule determines the applicability of the rule and the right side that describes the output, if rule is applied.
2. *Knowledge intensive*: The knowledge base of production system stores extensive and pure knowledge. This part contains no control or programming information. The problem of semantics is resolved by representing the knowledge in proper structure.

3 *Opacity*: Along with the advantages, there are certain disadvantages also associated with production systems. Opacity is the problem generated by combination of production rules. Though, the individual production rules may be models of clarity, the combined operation and effects of control program may be opaque. The opacity is generated because of less prioritization of rules.

4. *Inefficiency*: Sometimes, several of the rules become active during execution. A well devised control strategy reduces this problem. As the rules of production system are large in number and they are hardly written in hierarchical manner, it requires exhaustive search through all the production rules for each cycle of control program. It makes the functioning inefficient.
5. *Absence of learning*: The simple rule based production system does notstore the results of computations for later use. Hence, it does not exhibitany type of learning capabilities.
6. *Conflict Resolution*: The rules in ideal production system should not have any type of conflict. The new rule whenever added in the database should ensure that it does not have any conflict with the existing rules. Besides this, there may be more than one number of rules that can be applied(typically called *fired*) in one situation. While choosing the rule also, the conflict should not happen. If conflict is found, it should be resolved in following ways:
   (i) Assign priority to the rules and fire the rule with the highest priority. This method is used in many expert systems. Its virtue lies in its simplicity, and by ordering the rules in the approximate order of their firing frequency. This can be made using a relatively efficient strategy.
   (ii) Use a longest matching strategy. This means that fire a rule having largest

number of matching constraints. Its advantage is that the discrimination power of a strict condition is greater than of a more general condition. A rule with more constraints provides more knowledge.

(iii) Choose most recently used rule for firing. Its advantage is that it represents a depth first search, which follows the path of greatest activity.

2.3.2 Characteristics of Production System

Now let us describe some of the main characteristics of production system from the aspect of their storage in computer system. The production systems can be of various types, but the most popular production system is monotonic system. A monotonic production system is a production system in which the application of one rule never prevents the later application of another rule. The characteristics of production systems are presented below:

Board position: = {1,2,3,4,5,6,7,8,9}

An element contains the value 0, if the corresponding square is blank; 1, if it is filled with "O" and 2, if it is filled with "X".

Hence starting state is {0,0,0,0,0,0,0,0,0}

The goal state or winning combination will be board position having "O" or "X" separately in the combination of ({1,2,3}, {4,5,6}, {7,8,9},{1,4,7},{2,5,8}, {3,6,9}, {1,5,9}, { 3,5,7}) element values. Hence two goal states can be {2,0,1,1,2,0,0,0,2} and {2,2,2,0,1,0,1,0,0}. These values correspond to the goal states shown in the figure.

The start and goal state are shown in Fig. 2.2.

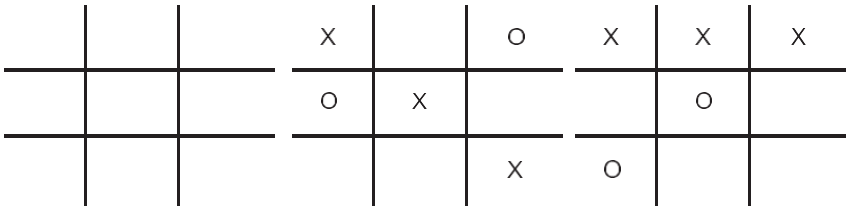


Fig 2.2: Start and goal states of TIC-TAC-TOE

Any board position satisfying this condition would be declared as win for corresponding player. The valid transitions of this problem are simply putting '1' or '2' in any of the element position containing 0.

In practice, all the valid moves are defined and stored. While selecting a move

it is taken from this store. In this game, valid transition table will be a vector (having 3^9 entries), having 9 elements in each.

2.5.2 Water-Jug Problem

This problem is defined as:

> *"We are given two water jugs having no measuring marks on these. The capacities of jugs are 3 liter and 4 liter. It is required to fill the bigger jug with exactly 2 liter of water. The water can be filled in a jug from a tap".*

In this problem, the start state is that both jugs are empty and the final state is that 4-liter jug has exactly 2 liters of water. The production rules involve filling a jug with some amount of water, filing the water from one jug to other or emptying the jug. The search will be finding the sequence of production rules which transform the initial state to final state.

Part of search tree of water jug problem is shown in Fig. 2.3.
sequence to transform the start state to goal state. One solution is applying the rules in the sequence 2, 9, 2, 7, 5, 9. The solution is presented in the following Table 2.1.

Table 2.1: Production rules applied in Water-Jug problem

Rule applied	Water in 4-liter jug	Water in 3-liter jug
Start state	0	0
2	0	3
9	3	0
2	3	3
7	4	2
5	0	2
9	2	0

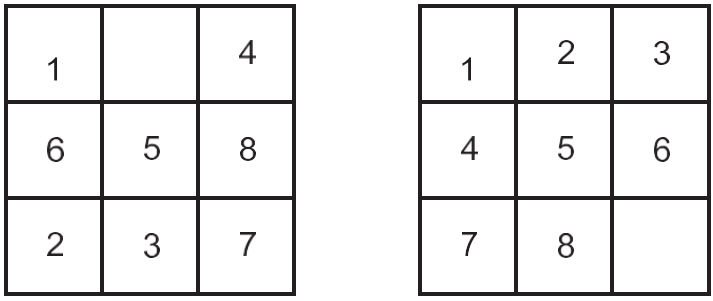
2.5.3 8-Puzzle Problem

The 8-puzzle problem belongs to the category of "sliding-block puzzle" types of problems. It is described as follows:

> *"It has set of a 3x3 board having 9 block spaces out of which, 8 blocks are having tiles bearing number from 1 to 8. One space isleft blank.*

The tile adjacent to blank space can move into it. Wehave to arrange the tiles in a sequence."

The start state is any situation of tiles, and goal state is tiles arranged in a specific sequence. Solution of this problem is reporting of "movement of tiles" in order to reach the goal state. The transition function or legal move is any one tile movement by one space in any direction (i.e. towards left or right or up or down) if that space is blank. It is shown in following Fig. 2.4:



(a) Start state (b) Goal state

Fig. 2.4: Start and Goal states of 8 puzzle problem

Here the data structure to represent the states can be 9-element vector indicating the tiles in each board position. Hence, a starting state corresponding to above configuration will be {1, blank, 4, 6, 5, 8, 2, 3, 7} (there can be various different start positions). The goal state is {1, 2, 3, 4, 5, 6, 7, 8, blank}. Here, the possible movement outcomes after applying a move can be many. They are represented as tree. This tree is called state space tree. The depth of the tree will depend upon the number of steps in the solution. The part of state space tree of 8-puzzle is shown in Fig 2.5.

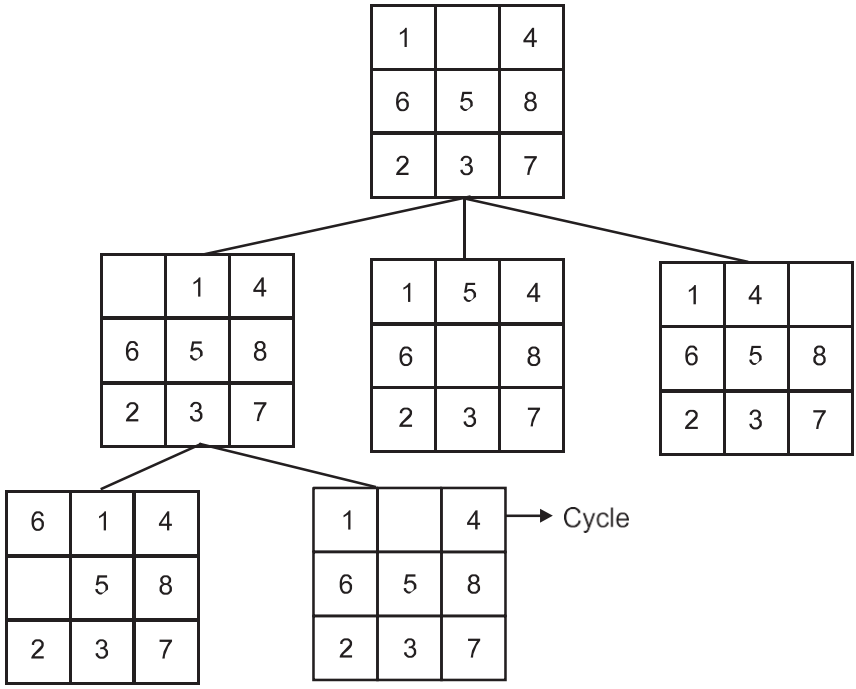


Fig 2.5: Partial search tree of 8-puzzle problem

2.5.4 8-Queens Problem

This problem is presented as follows:

> *"We have 8 queens and a 8 x 8 chessboard having alternate black and white squares. The queens are placed on the chessboard. Any queen can attack any another queen placed on same row, or column, or diagonal. We have to find the proper placement of queens on the chessboard in such a way that no queen attacks other queen".*

The 8-queen problem is shown in following diagram:

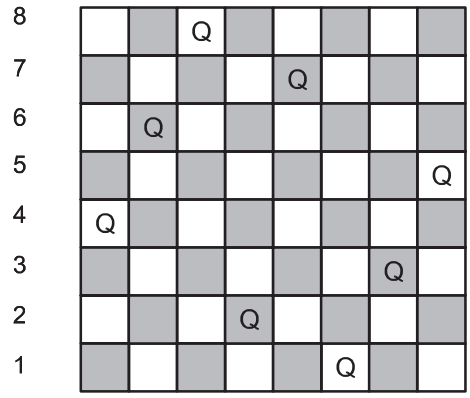


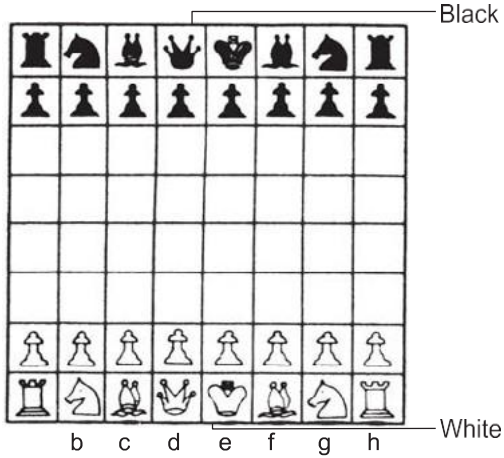
Fig 2.6[a] A possible board configuration of 8 queens problem

2.5.5 Chess Problem

It is a normal chess game. In a chess game problem, the start state is the initial configuration of chessboard. The final or goal state is any board configuration, which is a winning position for any player (clearly, there may be multiple final positions and each board configuration can be thought of as representing a state of the game). Whenever any player moves any piece, it leads to different state of game.

It is estimated that the chess game has more than 10^{120} possible states. The game playing would mean finding (or searching) a sequence of valid moves which bring the board from start state to any of the possible final states.

The start state of chess game is shown in Fig 2.7.



2.5.6 Missionaries and Cannibals Problem

The problem is stated as follows:

> *"Three missionaries and three cannibals are present at one side of a river and need to cross the river. There is only one boat available. At any point of time, the number of cannibals should not outnumber the number of missionaries at that bank. It is also known that onlytwo persons can occupy the boat available at a time."*

The objective of the solution is to find the sequence of their transfer from one bank of river to other using the boat sailing through the river satisfying these constraints.

We can form various production rules as presented in water-jug problem. Let Missionary is denoted by 'M' and Cannibal, by 'C'. These rules are described below:

Rule 1 :	(0, M)	:	One missionary sailing the boat from bank-1 to bank-2
Rule 2 :	(M, 0)	:	One missionary sailing the boat from bank-2 to bank-1
Rule 3 :	(M, M)	:	Two missionaries sailing the boat from bank-1 to bank-2
Rule 4 :	(M, M)	:	Two missionaries sailing the boat from bank-2 to bank-1
Rule 5 :	(M, C)	:	One missionary and one Cannibal sailing the boat from bank-1 to bank-2
Rule 6 :	(C, M)	:	One missionary and one Cannibal sailing the boat from bank-2 to bank-1
Rule 7 :	(C, C)	:	Two Cannibals sailing the boat from bank-1 to bank-2
Rule 8 :	(C, C)	:	Two Cannibals sailing the boat from bank-2 to bank-1
Rule 9 :	(0, C)	:	One Cannibal sailing the boat from bank-1 to bank-2
Rule 10 :	(C, 0)	:	One Cannibal sailing the boat from bank-2 to bank-1

All or some of these production rules will have to be used in a particular sequence to find the solution of the problem. The rules applied and their sequence is presented in the following Table 2.2.

Table 2.2: Rules applied and their sequence in Missionaries and Cannibals problem

After application of rule	persons in the river bank-1	persons in the river bank-2	boat position
Start state	M, M, M, C, C, C	0	bank-1
5	M, M, C, C	M, C	bank-2
2	M, M, C, C, M	C	bank-1
7	M, M, M	C, C, C	bank-2

10	M, M, M, C	C, C	bank-1
3	M, C	C, C, M, M	bank-2
6	M, C, C, M	C, M	bank-1
3	C, C	C, M, M, M	bank-2
10	C, C, C	M, M, M	bank-1
7	C	M, M, M, C, C	bank-2
10	C, C	M, M, M, C	bank-1
7	0	M, M, M, C, C, C	bank-2

2.5.7 Tower of Hanoi Problem

This is a historic problem. It can be described as follows:

> *"Near the city of 'Hanoi', there is a monastery. There are three tall posts in the courtyard of the monastery. One of these posts is having sixty-four disks, all having a hole in the centre and are of different diameters, placed one over the other in such a way that always a smaller disk is placed over the bigger disk. The monks of the monastery are busy in the task of shifting the disks from one post to some other, in such a way that at no point of time, a bigger disk is placed above smaller disk. Only one disk can be removed at a time. Moreover, at every point of time during the process, all other disks than the one removed, should be on one of the posts. The third post can be used as a temporary resting place for the disks. We have to help the monks in finding the easiest and quickest way to do so".*

Can we really help the monks? Before indulging ourselves in finding the solution of this problem, let us realize that even the quickest method to solve this problem might take longer time than the time left for the whole world to finish! Are you still interested in attempting the problem?

64-disk configuration is shown in Fig 2.8.

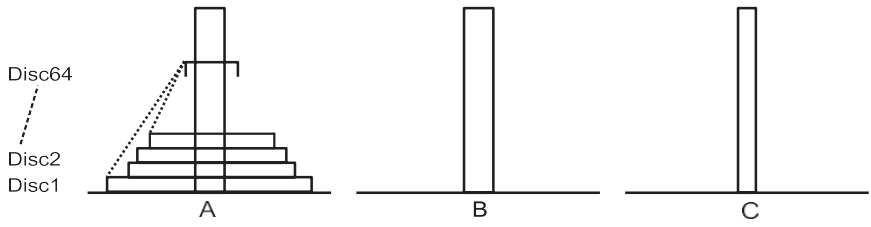


Fig 2.8: Tower of Hanoi problem

2.5.8 Traveling Salesperson Problem

This problem falls in the category of path finding problems. The problem is defined as follows:

> *"Given 'n' cities connected by roads, and distances between each pair of cities. A sales person is required to travel each of the cities exactly once. We are required to find the route of salesperson sothat by covering minimum distance, he can travel all the cities and come back to the city from where the journey was started".*

Diagrammatically, it is shown in Fig 2.9.

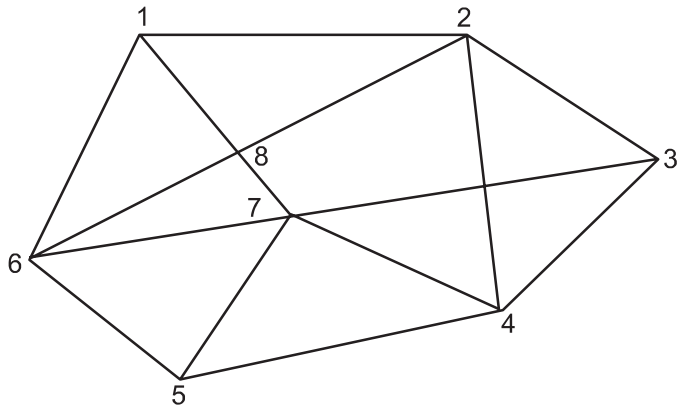


Fig 2.9: Cities and paths connecting these

The basic travelling salesperson problem comprises of computing the shortest route through a given set of cities.

Following Table 2.3 shows number of cities and the possible routes mentioned against them.

Table 2.3: Possible routes of travelling salesperson problem

Number of cities	Possible Routes
1	1
2	1 -2-1
3	1 -2 -3 1
	1 -3 -2 1

4	1- 2- 3- 4-1 1- 2- 4- 3- 1 1- 3- 2- 4- 1 1- 3- 4- 2- 1 1- 4- 2- 3-1 1- 4- 3- 2- 1

We can notice from here that the number of routes between cities is proportional to the factorial of the (number of cities – 1), i.e., for three cities, the number of routes will be equal to !2 (2x1), and for four cities, !3 (3x2x1).

While there are !9 = 362880 routes for 10 cities, there are !29 = 8.8 E30 possible routes for 30 cities. The travelling salesperson problem is a classic example of ***combinatorial explosion***, because the number of routes increases so rapidly that there are no practical solutions for realistic number of cities. If it takes 1 hour of a mainframe CPU time to solve for 30 cities, it will take 30 hoursfor 31 cities and 330 hours for 32 cities. Practically, the actual application of such type of problem is for routing a data packet on computer networks. It requires managing thousands of cities. Hence for this much large amount of data the required problem solution time will be unaffordable.

The solution of this problem is done using neural network. The neural network can solve the 10 city case just as fast as the 30 city case whereas a conventional computer takes much longer time for these.

2.5.9 Magic Square

The problem is represented as follows:

> *"We are given a square of same number of rows and columns, where consecutive numbers are filled in blank spaces of each row and column such that numbers in each row, column and two diagonals, add up to the same number. The numbers have to beconsecutive and one number can be used only once".*

Therefore, the 3 ¥ 3 square and 4 ¥ 4 square would require any set of 9 and 16 consecutive numbers respectively. The problem is finding out the placement of numbers. The magic squares of order 3 ¥ 3, where numbers from 1 to 9 are filled, is shown in Fig 2.10. It should be noted that there could be multiple solutions of magic square problem.

6	7	2
1	5	9
8	3	4

Fig. 2.10: A 3 ¥ 3 magic square configuration

Let us describe the manual process of solving 3 ¥ 3 magic square by filling numbers from 1 to 9. The steps required to attempt the problem are presented as follows:

Step 1: The sum of the numbers in each row and column can be predetermined as:

$$\text{Sum of numbers in each row, column and diagonal} = \frac{\text{Sum of numbers used}}{\text{Number of rows or columns}}$$

In our case, this figure is equal to 45/3, i.e. 15. (Sum of numbers from 1 to 9 divided by 3)

Step 2: Suppose the numbers to be filled in each blank space are x_1 to x_9. The magic square then will look like:

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

Now, since we know the sum of numbers in each row, column and diagonals is 15, we can form following equations:

$x_1 + x_2 + x_3 = 15$ ..... (1)
$x_4 + x_5 + x_6 = 15$ ..... (2)
$x_7 + x_8 + x_9 = 15$ ..... (3)
$x_1 + x_4 + x_7 = 15$ ..... (4)
$x_2 + x_5 + x_8 = 15$ ..... (5)

$x_3 + x_6 + x_9 = 15$ …….. (6)
$x_1 + x_5 + x_9 = 15$ …….. (7)
$x_3 + x_5 + x_7 = 15$ …….. (8)

Step 3: We will have to solve the above equations for finding the values of numbers. You may notice that number x_5 comes in most of the equations hence; we should try to find the value of x_5 first.

Add the equations (5), (7) and (8), to get:

$x_2 + x_5 + x_8 + x_1 + x_5 + x_9 + x_3 + x_5 + x_7 = 45$

Putting values from equations (1) and (3), we get,

$3\ x_5 = 15 \Rightarrow x_5 = 5$.

Step 4: Put the value of x_5 in the equations where it appears, we get,

$x_4 + x_6 = 10$ ……. (9)
$x_2 + x_8 = 10$ ……. (10)
$x_1 + x_9 = 10$ ……. (11)
$x_3 + x_7 = 10$ ……. (12)

Step 5: Assume $x_1 = 1$, hence $x_9 = 9$ [from equation (11)]

After getting three values, our magic square will look like:

1	x_2	x_3
x_4	5	x_6
x_7	x_8	9

Now, try to find the location of number 2. Any number out of x_2, x_3, x_4, and x_7 cannot be 2, because in that case remaining number will be 12 to make the sum as 15, which is not permissible. Hence, either x_6 or x_8 will be 2. But, it is also not possible, because in that case, x_3 or x_7 will be 4 and x_2 or x_4 will become 10, which is again not possible. Therefore, x_1 cannot be 1. Revise this step with another assumption.

Step 6: Assume $x_2 = 1$, hence $x_8 = 9$ [from equation (10)].

After getting three values, our magic square will look like drawn ahead.

x_1	1	x_3
x_4	5	x_6
x_7	9	x_9

Now, any number out of x_1 and x_3 cannot be 2, because of the reason mentioned in the above step, hence one number out of x_4, x_6, x_7 and x_9 will be equal to 2. Assume $x_4 = 2$, hence $x_6 = 8$ [from equation (9)]. Notice that 3 can not come in the same row, column or diagonal in which either 1 or 2 comes, because then, the remaining number will be either 11 or 10, which is not possible. Hence, for $x_4 = 2$, x_1, x_3, and x_7 cannot be 3, resulting in $x_9 = 3$. However, this is also not possible, because number 3 cannot come with 9 since the remaining number in that case will also be 3, which cannot be. Therefore, x_2 or x_4 cannot be 2. Try another assumption.

Step 7: Assume x_7 or $x_9 = 2$, resulting in x_3 or $x_1 = 8$ [from equations (12) and (11)].

After getting one more number, our magic square will look like:

x_1	1	x_3
x_4	5	x_6
2	9	x_9

x_1	1	x_3
x_4	5	x_6
x_7	9	2

From the above squares, we can calculate the remaining numbers as, $x_9 = 4$, $x_1 = 6$, $x_3 = 8$, $x_6 = 3$ and $x_4 = 7$ for the first square; and $x_7 = 4$, $x_3 = 6$, $x_1 = 8$, $x_4 = 3$ and $x_6 = 7$ for the second square; and the solutions for the magic square will be:

6	1	8
7	5	3
2	9	4

8	1	6
3	5	7
4	9	2

As mentioned earlier, there can be several solutions to the magic square depending upon which number you assume as 1 to start with in the step 5.

Similarly, we can solve magic square of the order 4 ¥ 4. A typical solution is given as follows in the Fig 2.11 :

7	13	12	2
10	4	5	15
1	11	13	8
16	6	3	9

Figure 2.11: Magic square of the order 4 x 4

2.5.10 Language Understanding Problems

This type of problems include the understanding of natural languages, conversion of one language to another language, language comprehension, design of intelligent natural language interfaces etc. It also includes answering of a query using a database. More regarding such type of problems and other various issues involved in natural language understanding are deal with in another chapter of this book.

2.5.11 Monkey and Banana Problem

This problem is described as follows:

> *"A monkey and a bunch of banana are present in a room. The bananas are hanging from the ceiling. The monkey cannot reach the bananas directly. However, in the room there is one chair and a stick. The monkey can reach the banana standing on the chair. We have to find the sequence of events by which monkey can reach the bananas".*

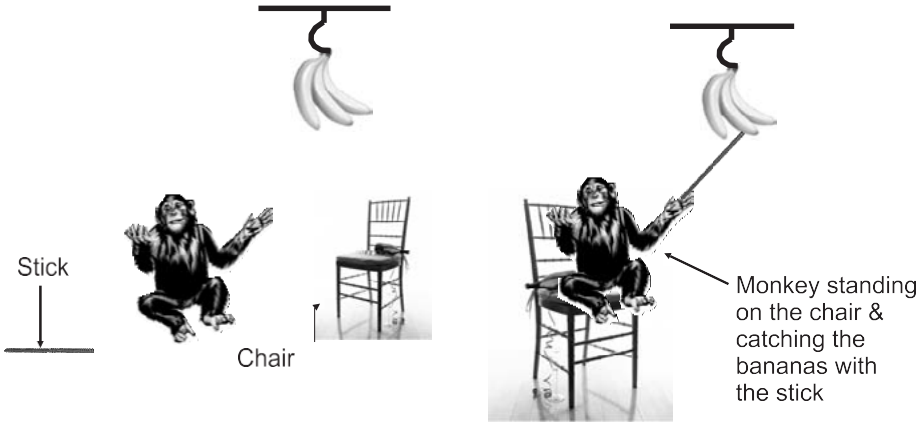


Fig 2.12: Monkey and banana problem

Solution of this problem means finding the sequence of actions for the monkey to reach the banana. It is much simpler than the problems of water and jug or Missionaries and Cannibals discussed above, hence, we leave it for the readers to formulate the set of production rules and to find the appropriate sequence of the actions required for the solution. The problem is pictorially represented in Fig 2.12.

2.5.12 Cryptarithmatic Puzzle

It is a puzzle involving decoding of digit represented by a character. It is in the form of some arithmetic equation where digits are distinctly represented by some characters. The problem requires finding of the digit represented by each character. One such problem is shown in Fig. 2.13.

```
  R I N G
+ D O O R
---------
B E L L
```

Fig. 2.13: A cryptarithmatic problem

These types of problems require constraint satisfaction. Constraints are that all the laws of arithmetic must hold good and any letter should represent same digit wherever it comes in the puzzle. Also, no two different letters can be represented by same digit. For example, in the puzzle shown above, the laws of summation of two given numbers must hold good and digit R, which comes at two different places must be represented by same digit. Similarly, letter O and L must be represented by same digit, however, R, O and L, together with other letters, must be represented by different digits. The process involved in solving these puzzles is described in detail in the next chapter of this book.

2.5.13 Block World Problems

The problem can be represented as follows:

> *"There are some cubic blocks, out of which, each is placed either on the table or on other block forming a particular configuration. We have to move the blocks to arrange them to form some other given configuration by applying minimum number of moves. The requirement of moving a block is that only the top block from a group can be shifted in a move. It can be placed either on the table or on top of any other block."*

The live application of this type of problem can be seen in a container depot where lot of containers are placed one over the other in several groups. A crane or a robot is used to arrange these containers in different groups for loading in trains and ships. The start and goal states of a typical block world problem are shown in Fig 2.14.

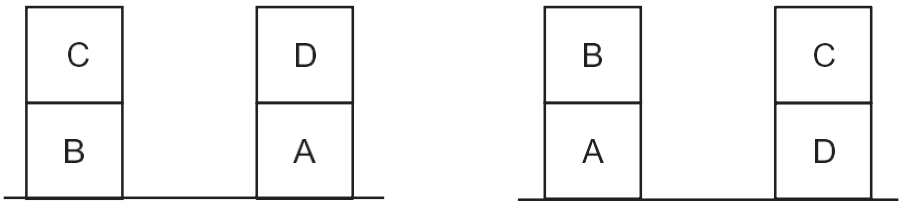


Fig. 2.14: The start and goal configuration of block world problem

Detailed account of this problem is given in the section of *Planning* of chapter -11 of this book.

2.6 NATURE OF AI PROBLEMS

In this section, we will highlight the nature of real world AI problems. As mentioned earlier, problems can be of several types and solution of a particular problem depends upon the nature of the problem. We may categorize problems based on their nature in the manner discussed hereunder:

2.6.1 Path Finding Problems

In this type of problems, the solution involves reporting of path (or sequence of steps used to obtain the solution). The traveling salesperson problem discussed above is an example of this kind of problem.

2.6.2 Decomposable Problems

In this type of problems, the problem can be broken into small subproblems and the solution of the main problem is obtained by joining the solutions of small subproblems, e.g. in mathematical equation, if the expression involves the solution of various components then these components can be solved independently, and the complete answer can be obtained by joining the solutions of subproblems. However, all the real world problems are not decomposable problems.

Consider following arithmetic equation:

$$\int (x^3 + 3x^2 + \sin^2 x + \cos^2 x)\, dx = 0$$

If we want to solve this equation, we can find the solution of different integrals separately and then join the results to get complete solution. Hence, above problem is broken into four problems, i.e. (i) $\int x^3\, dx$, (ii) $\int 3x^2\, dx$, (iii) $\int \sin^2 x\, dx$, (iv) $\int \cos^2 x\, dx$.

2.6.3 Recoverable Problems

There are certain types of problems, where the application of operator can be reverted if required, and on the initial state, a new operator can be applied. Such types of problems are called recoverable problems. 8-puzzle problem is an example of this kind because if you apply a move and find that it is wrong or not worth, you may revert to the original position and apply another move. The recoverability of a problem plays an important role in determining the control

structure necessary for problem solution. There is another type of problems like "theorem proving" where the steps can be ignored. These are called ignorable problems. These can be solved using simple control structures, which are easy to implement. But there are certain types of problems like medical diagnosis problems where if an operator is applied, it cannot be reverted to original state. In such types of problems, the selection of operator is very critical because if a wrong operator is applied, it may be possible that a solution is never obtained even if it exists. Such types of problems are called '**irrecoverable**' problems. Chess playing is another irrecoverable problem where once you apply a move, you are not allowed to revert. The recoverable problems require little complex control structure, which checks the outcomes of every move and in case themove is found to be wrong, it is reverted and initial state is restored back. The control structure of irrecoverable problems is difficult because it will involve a system that expands a great deal of effort in exploring each decision, since the decisions once chosen must be final.

2.6.4 Predictable Problems

These are the types of problems where all the outcomes of a particular move can be judged with definiteness. For example, consider the 8-puzzle problem where the outcomes after applying a certain move can be easily predicted because all the numbers and their positions are open before you. Hence, while deciding about a move, the suitability of these outcomes can be judged and planning the entire sequence of moves is possible. Whereas in card games like bridge or game where all the cards are not open, it cannot be decided in advance that what will be the effect of a particular move because your next move depends entirely upon the move of other players involved, hence planning a sequence of moves is not possible. Similar situation will be there in a game involving two players. One player can not judge the move played by other player. In such situations, multiple moves are considered for suitability and a probabilistic approach is applied to decide about the best move.

2.6.5 Problems Affecting the Quality of Solution

There are certain types of AI problems where the process of finding solution stops by just finding one solution and there is no need to ensure the validity of this solution by finding the other solutions, e.g., the database query applications. In query applications, whenever the query is answered, the other possible answers are not checked. However, in route finding problems (like traveling salesperson problem) once a route from source to destination is obtained, still other possibilities need to be checked to judge whether it is the shortest path or not (optimal solution).

Thus, we can say that there are certain types of problems where any one solution is enough and in other types of problems where to accept a solution all the possible solutions need to be checked.

2.6.6 State Finding Problem

In this type of AI problems, the answer is reporting of a state instead of path. The examples of such type of problems are natural language understanding applications. In such type of applications the interpretation of a particular sentence is required and it is not important how the solution is obtained. As far as the interpretation of the input sentence is correct, the steps used in obtaining the solution can be ignored. However, there are another kinds of problems where the solution demands the reporting of state as well as path. These are the problems requiring justification, e.g., medical diagnosis problems using expert system. Here, besides the name of the medicine, the patient requires explanation about the process or the path applied for finding that particular medicine. Thus, it involves the 'belief' of user as well. To satisfy him, the process of obtaining the solution (i.e. the path) and the solution (i.e. the state) both need to be stored. Hence, the types of problems, which involve human belief, fall in the category of problems where state and path both should be reported.

2.6.7 Problems Requiring Interaction

This is the kind of problems, which require asking some questions from the user or interacting with the user. There are many AI programs already built which interact with the user very frequently to provide additional input to the program and to provide additional reassurance to the user. In the situations where computer is generating some advice for a particular problem, it is always advisable to take the views of user at regular interval. Doing this will help providing the output in the form suitable to the user.

2.6.8 Knowledge Intensive Problems

These are kinds of AI problems, which require large amount of knowledge for their solution. Consider the tic-tac-toe game. Here, to play the game, the required amount of knowledge is only about legal moves and other player's moves. This is little amount of knowledge. Whereas in chess game, the number of legal moves are more, and to decide a particular move, more future moves of opponent and those of oneself in "*look ahead manner*" (technically called *ply*) are visualized. Hence, the amount of knowledge required to play a chess game is more as compared to tic-tac-toe game. Further, if we consider the medical expert system, the amount of knowledge required is enormous. Such types of AI applications are called

knowledge intensive applications. Similarly, in the application of a data query, the knowledge required is only about the database and query language whereas if it is a natural language understanding program, then it will require vast amount of knowledge comprising of syntactic, semantic and pragmatic knowledge. Hence, the amount of knowledge and role of knowledge varies in different AI problems.

From the above discussion, it is evident that the nature of various AI problems is widely different and in order to choose the most appropriate method for solving a particular problem, the nature of problem needs to be analyzed. The problem characteristics can be summarized as follows:

1. If the problem is decomposable into independent smaller or easier sub problems.
2. Is backtracking possible or not.
3. Is the problem's universe predictable.
4. Is a good solution to the problem obvious without comparison to all other possible solutions.
5. Is the desired solution a state or a path.
6. Is a large amount of knowledge absolutely required to solve the problem or is knowledge important only to constrain the search.

2.7 SEARCH TECHNIQUES

As stated earlier, the process of the solution of AI problem searches the path from start state to goal state. This is very important aspect of problem solving because search techniques not only help in finding most viable path to reach the goal state, but also make the entire process efficient and economical. In this section, we discuss basic search techniques adopted for finding the solution of AI problems. Broadly, the search is of two types:

(i) *Blind (or unguided or uninformed) search*: The uninformed or blind search is the search methodology having no additional information about states beyond that provided in the problem definitions. In this search total search space is looked for solution.

(ii) *Heuristic (or guided or informed) search*: These are the search techniques where additional information about the problem is provided in order toguide the search in a specific direction.

In this chapter, we will discuss the unguided search techniques and the guided search techniques will be dealt with in the next chapter. Following search techniques comes under the category of unguided search techniques:

(i) Breadth-First search (BFS)
(ii) Depth-First search (DFS)
(iii) Depth-Limited search (DLS)
(iv) Bidirectional search

2.7.1 Breadth-First Search

In this type of search, the state space is represented in form of a tree. In addition, the solution is obtained by traversing through the tree. The nodes of tree represent start state, various intermediate states and the goal state. While searching for the solution, the root node is expanded first, then all the successors of the root node are expanded, and in next step all successors of every node are expanded. The process continues till goal state is achieved, e.g., in the tree shown in following Fig. 2.15, the nodes will be explored in order A, B, C, D, E, F, G, H, I, J, K, L:

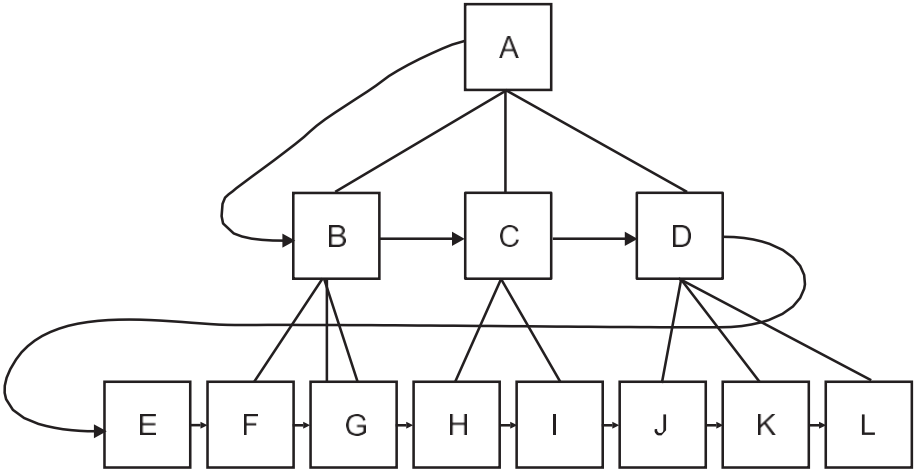


Fig. 2.15: Breadth-First Search Tree

In breadth-first search, the space complexity is more critical as compared to time complexity. Analysis shows that the time and memory requirement of the problem of depth 8 and 10 are 31 hours, 1 terabyte and 129 days, 101 terabyte respectively. Fortunately, there are other strategies taking lesser time in performing

the search. In general, the problems involving search of exponential complexity (like chess game) cannot be solved by uninformed methods for the simple reason, the size of the data being too big. The data structure used for breadth first search is First In First Out (FIFO).

Algorithm for breadth first search is described as follows:

(i) Create a variable called Node-List and set it to initial state.
(ii) Until a goal state is found or Node-List is empty do:
  (a) Remove the first element from node-list and call it E. If node-list wasempty, quit.
  (b) For each way that each rule can match the state described in E do:
    (i) Apply the rule to generate a new state.
    (ii) If new state is a goal state, quit and return this state.
    (iii) Otherwise, add new state to the end of node-list.

2.7.1.1 Advantages of Breadth First Search (BFS)

The breadth first search is not caught in a blind alley. This means that, it will not follow a single unfruitful path for very long time or forever, before the path actuallyterminates in a state that has no successors. In the situations where solution exists, the breadth first search is guaranteed to find it. Besides this, in the situations where there are multiple solutions, the BFS finds the minimal solution. The minimalsolution is one that requires the minimum number of steps. This is because of thefact that in breadth first search, the longer paths are never explored until all shorter ones have already been examined. Thus, if the goal state is found during the search of shorter paths, longer paths would not be required to be searched, saving time and efforts.

Traveling sales person problem discussed above can be solved using Breadth- First Search technique. It will simply explore all the paths possible in the tree and will ultimately come out with the shortest path desired. However, this strategy works well only if the number of cities is less. If we have large number of cities in the list, it fails miserably because number of paths and hence the time taken to perform the search become too big to be controlled by this method efficiently.

2.7.2 Depth First Search

There can be another type of search strategy than the one described above. In thistype of approach, instead of probing the width, we can explore one branch of a tree until the solution is found or we decide to terminate the search because either a dead end is met, some previous state is encountered or the process becomes longer than the set time limit. If any of the above situations is encountered and the process is terminated, a backtracking occurs. A fresh search will be done on some other branch of the tree, and the process will be repeated until goal state is reached.This type of technique is known as Depth-First Search technique.

It generates the left most successor of root node and expands it, until dead end is reached. The search proceeds immediately to the deepest level of search tree, or until the goal is encountered. The sequence of explored nodes in the previous tree will be A, B, E, F, C, G, H, K, L, D, I, J. The search is shown in Fig 2.16 below:

The DFS has lesser space complexity, because at a time it needs to store onlysingle path from root to leaf node. The data structure used for DFS is Last-In- First-Out (LIFO).

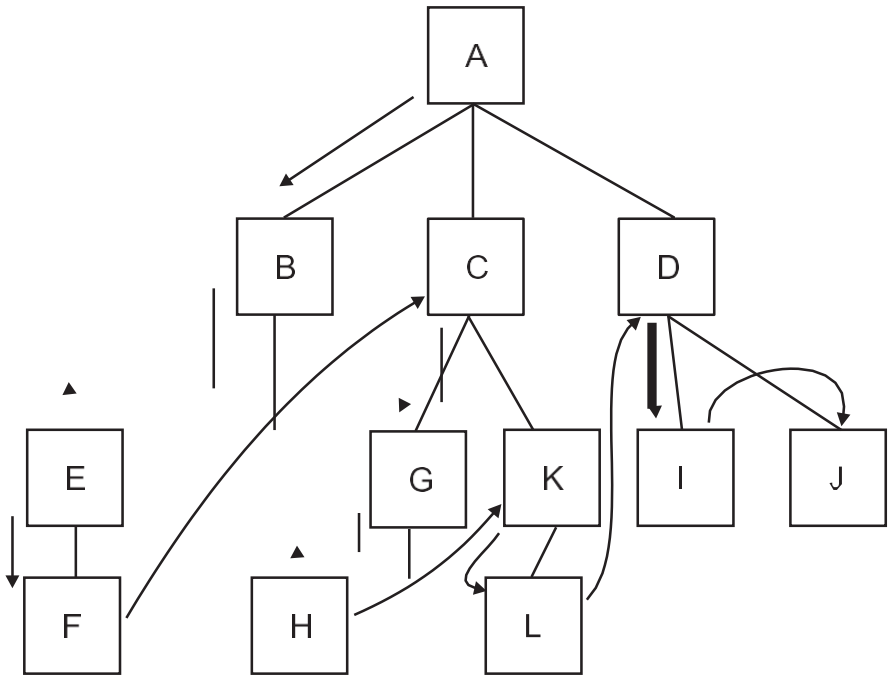


Fig 2.16: Depth First search tree

Algorithm for Depth First Search is described as follows:

1. If initial state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is reported:
   (a) Generate a successor E of the initial state. If there are no more successorssignal failure.
   (b) Call Depth-First Search with E as initial state.
   (c) If success is obtained, return success, otherwise continue in this loop.

The procedure to perform DFS for tree of node 'n' is given below:Procedure: unbounded DFS for tree of node n, [DFS1(n)].
Start
If n is a goal node thenStart

Solution : = n;
Exit;
End;
For each successor ni of n do
If ni is not an ancestor of n then
P1(ni);

2.7.4 Bidirectional Search

As the name indicates, it is search in two directions. In the search methods discussedabove, the search proceeds only from start to goal, unlike in bidirectional search, where two simultaneous searches are done. One search proceeds in forward direction (i.e. starting from start node towards goal node) and another search proceeds

in the backward direction (i.e. starting from goal node towards start node). Wherever two searches meet, the process stops. The bidirectional search is advantageous then unidirectional search, because for a tree of depth 'd' and branching factor 'b', the unidirectional search requires b^d number of comparisonsbut bidirectional search requires $b^{d/2}$ comparisons in each direction. As$b^{d/2} + b^{d/2} < b^d$, the bidirectional search has lesser time complexity. However, it is applicable to only on those situations where search tree is completely defined, i.e. all actions in state space are reversible. A bidirectional search is shown in Fig 2.17.

Here A, B, C, E, F are cities to be traveled and the distance between each pair of cities is mentioned in the graph. In the solution of this problem, the location ofcities can be represented in graph by nodes indicating the cities. The starting state

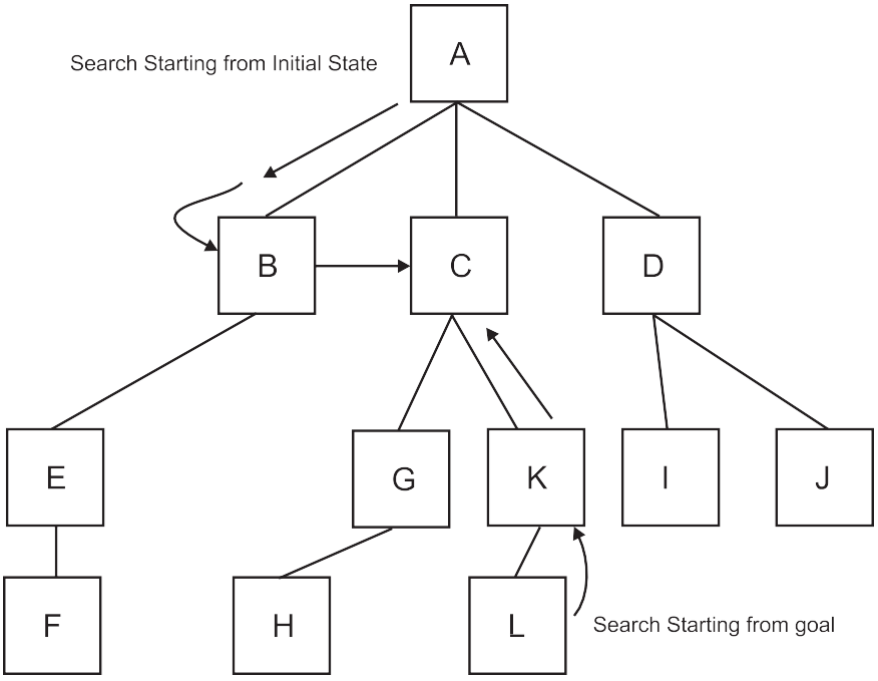


Fig. 2.17: Example of Bidirectional search

What is the difference between informed and uninformed searches?

An uninformed search is a searching technique that has no additional information about the distance from the current state to the goal.

Informed Search is another technique that has additional information about the estimate distance from the current state to the goal.

Basis of comparison	Informed search	Uninformed search
Basic knowledge	Uses knowledge to find the steps to the solution.	No use of knowledge
Efficiency	Highly efficient as consumes less time and cost.	Efficiency is mediatory
Cost	Low	Comparatively high
Performance	Finds the solution more quickly.	Speed is slower than the informed search.
Algorithms	Heuristic depth-first and breadth-first search, and A* search	Depth-first search, breadth-first search, and lowest cost first search

Difference between BFS and DFS

Breadth First Search

BFS stands for **Breadth First Search** is a vertex based technique for finding a shortest path in graph. It uses a Queue data structure which follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS.

Ex-

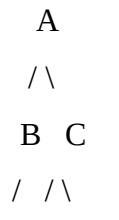
A

Output is:
A, B, C, D, E, F

Depth First Search

DFS stands for **Depth First Search** is a edge based technique. It uses the Stack data structure, performs two stages, first visited vertices are pushed into stack and second if there is no vertices then visited vertices are popped.

Ex-



Output is:
A, B, D, C, E, F

BFS vs DFS

S.NO	BFS	DFS
1.	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
2.	BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
3.	BFS can be used to find single source shortest path in an unweighted graph, because in BFS, we reach a vertex with minimum number of edges from a source vertex.	In DFS, we might traverse through more edges to reach a destination vertex from a source.

3.	BFS is more suitable for searching vertices which are closer to the given source.	DFS is more suitable when there are solutions away from source.
4.	BFS considers all neighbors first and therefore not suitable for decision making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.
5.	The Time complexity of BFS is O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.	The Time complexity of DFS is also O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.

Uninformed Search Algorithms

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

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

1. Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise 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:

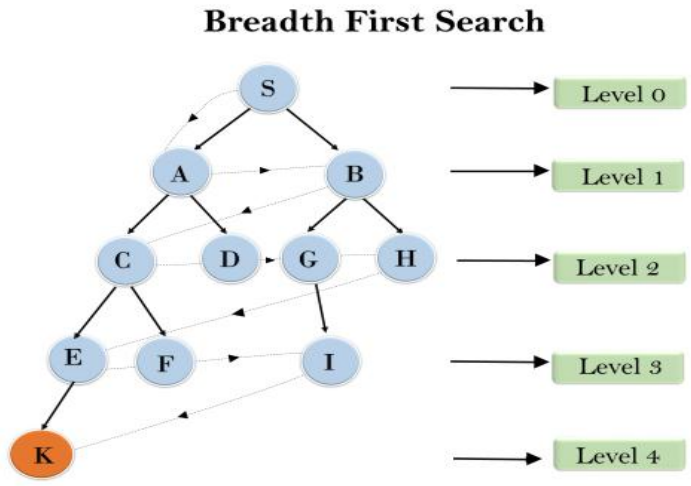
- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

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:**

1. S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K



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.

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

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

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.

2. Depth-first Search

- Depth-first search isa 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.

Note: <u>Backtracking is an algorithm technique for finding all possible solutions using recursion.</u>

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:

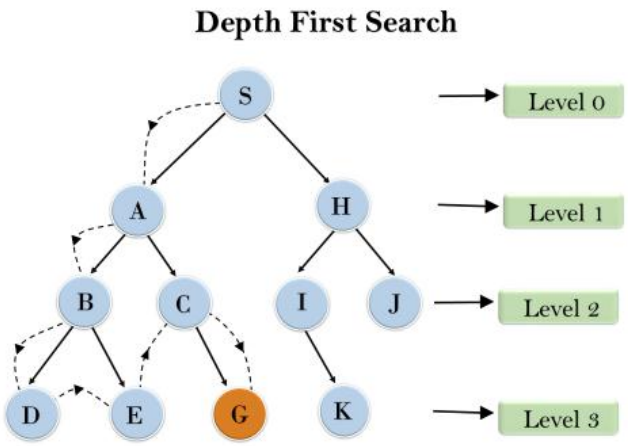
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Example:

In the below search tree, we have shown the flow of depth-first search, 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:

$T(n)= 1+ n^2+ n^3 +.....+ n^m=O(n^m)$

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.

3. Depth-Limited Search Algorithm:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.

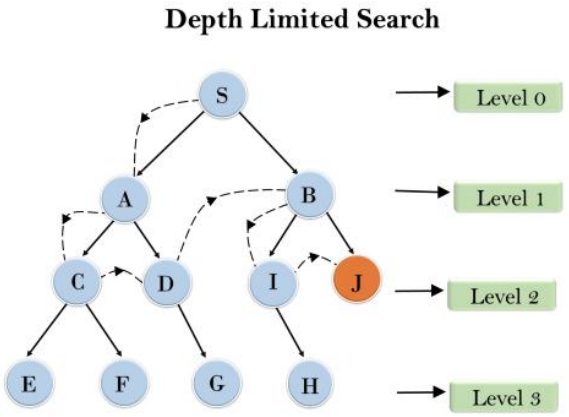
Advantages:

Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Example:



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is **$O(b^{\ell})$**.

Space Complexity: Space complexity of DLS algorithm is **$O(b \times \ell)$**.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell > d$.

4. Uniform-cost Search Algorithm:

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 form 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.

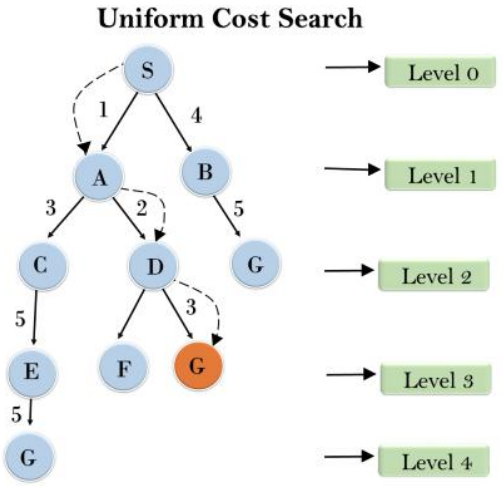
Advantages:

- 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 = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.

Hence, the worst-case time complexity of Uniform-cost search is$\mathbf{O(b^{1 + \lceil C^*/\varepsilon \rceil})}$/.

Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $\mathbf{O(b^{1 + \lceil C^*/\varepsilon \rceil})}$.

Optimal:

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

5. Iterative deepeningdepth-first Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Advantages:

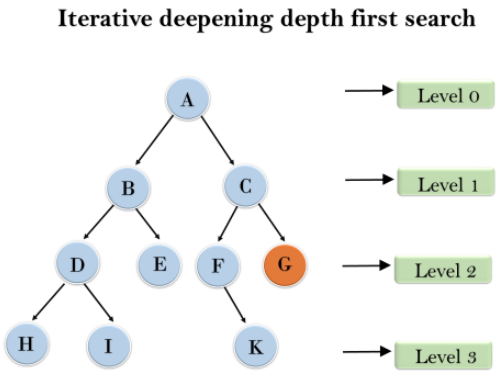
- Itcombines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A, B, D, E, C, F, G
4'th Iteration------>A, B, D, H, I, E, C, F, K, G
In the fourth iteration, the algorithm will find the goal node.

Completeness:

This algorithm is complete is ifthe branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **$O(b^d)$**.

Space Complexity:

The space complexity of IDDFS will be **O(bd)**.

Optimal:

IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

6. Bidirectional Search Algorithm:

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small sub graphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

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

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

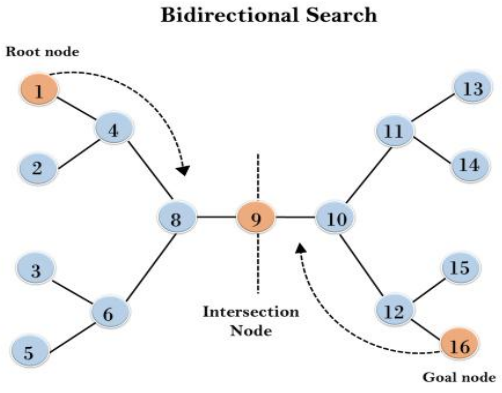
Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- **In bidirectional search, one should know the goal state in advance.**

Example:

In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.



Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is **$O(b^d)$**.

Space Complexity: Space complexity of bidirectional search is **$O(b^d)$**.

Optimal: Bidirectional search is Optimal.

Problem Solving In AI : Introduction

- Problem Solving in games such as "Sudoku" can be an example. It can be done by building an artificially intelligent system to solve that particular problem. To do this, one needs to define the problem statements first and then generating the solution by keeping the conditions in mind.
- Some of the most popularly used problem solving with the help of artificial intelligence are:
    1. **Chess.**
    2. **Travelling Salesman Problem.**
    3. **Tower of Hanoi Problem.**
    4. **Water-Jug Problem.**
    5. **N-Queen Problem.**

Problem Searching

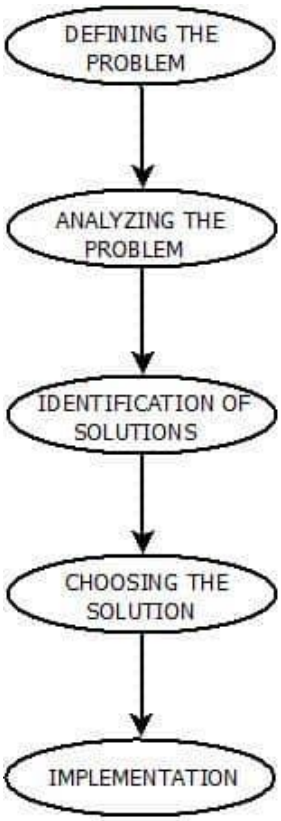
- **In general, searching refers to as finding information one needs.**
- **Searching is the most commonly used technique of problem solving in artificial intelligence.**
- **The searching algorithm helps us to search for solution of particular problem.**

Problem

- **Problems are the issues which comes across any system. A solution is needed to solve that particular problem.**

Steps : Solve Problem Using Artificial Intelligence

- **The process of solving a problem consists of five steps. These are:**



Problem Solving in Artificial Intelligence

1. **Defining The Problem:**
   The definition of the problem must be included precisely. It should contain the possible initial as well as final situations which should result in acceptable solution.
2. **Analyzing The Problem:**
   Analyzing the problem and its requirement must be done as few features can have immense impact on the resulting solution.
3. **Identification Of Solutions:**
   This phase generates reasonable amount of solutions to the given problem in a particular range.
4. **Choosing a Solution:**
   From all the identified solutions, the best solution is chosen basis on the results produced by respective solutions.
5. **Implementation:**
   After choosing the best solution, its implementation is done.

Searching Strategies : Introduction

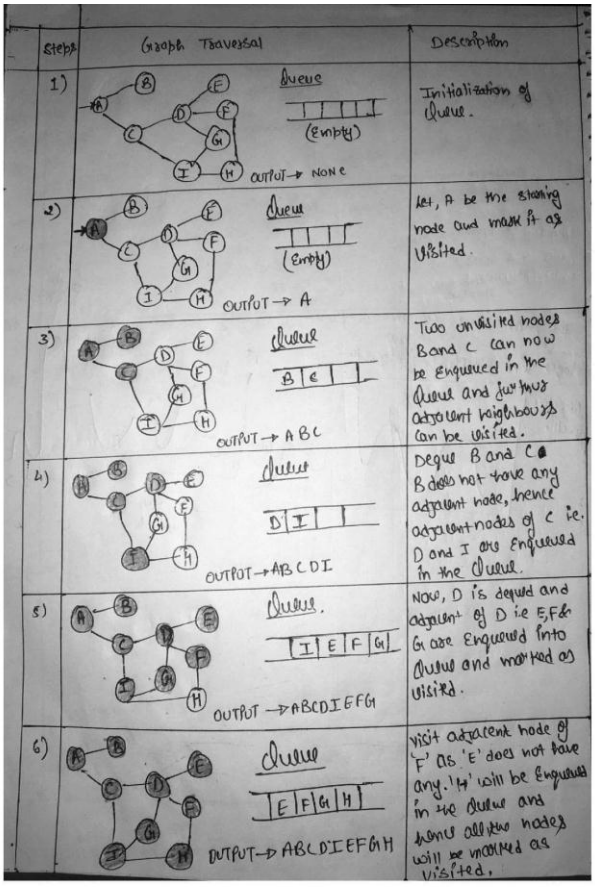
- Searching is a process to find the solution for a given set of problems. This in artificial intelligence can be done by using either uninformed searching strategies of either informed searching strategies.

Un-Informed Search Strategy

- **Un-Informed search strategy further includes two techniques. These are:**
  1. **Breadth First Search.**
  2. **Depth First Search.**

Breadth First Search

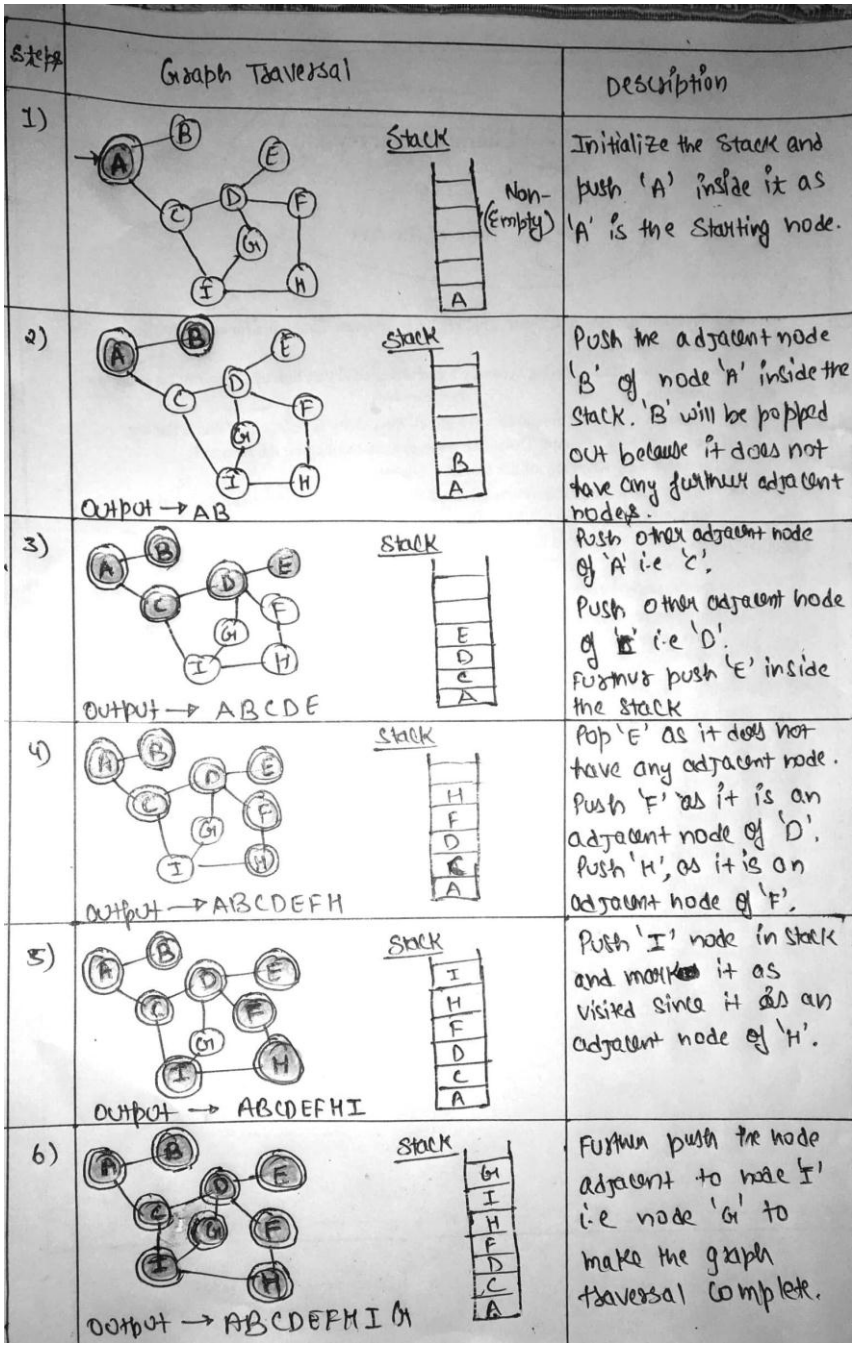
- **In Breadth First Search(BFS), the root node of the graph is expanded first, then all the successor nodes are expanded and then their successor and so on i.e. the nodes are expanded level wise starting at root level.**
- **For Example :**



Searching Strategies: Breadth First Search

Depth First Search

- In Depth First Search(DFS), the deepest node is expanded in the current unexplored node of the search tree. The search goes into depth until there is no more successor nodes.
- For Example :



Searching Strategies : Depth First Search

Crypt Arithmetic Problem

⇒ Type of Constraint Satisfaction Problem (CSP)

* Constraint
  - ↳ No Two letter have Same value.
  - ↳ Sum of digit must be as shown in problem.
  - ↳ There should be only one Carry forward.
* Digit that can be assigned to a word/alphabet (0 - 9)
  - (0 - 9) → Range.

Prof. Dhaval Khatri

Ex

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

letter	Digit
T	2
O	1
G	8
U	0

Left most digit = 1 ≠ 0

[0 - 9]

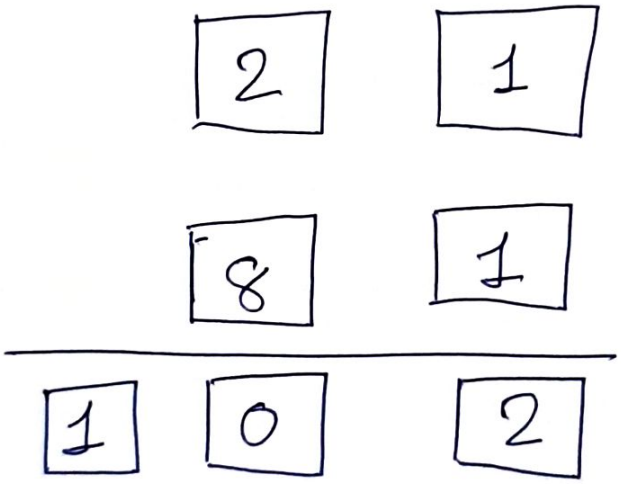
9 + 8 = 17

carry

2 + 6 = 6 + 10

2 + 9 = 11

2 + 8 = 10



Prof. Dhaval Khatri

Ex

```
  S E N D
+ m O R E
---------
m O N E Y
```

⇒ Starting left most

C_4	C_3	C_2	C_1	
	S: 9	E: 5	N: 6	D: 7
+	m: 1	O: 0	R: 8	E: 5
m: 1	O: 0	N: 6	E: 5	Y: 2

⇒ S+m, if m = 1, S+m ⩾ 10

9+1 ⩾ 10

S = 9

⇒ O = S+m = 9+1 = 10

O = 0

⇒ E+0 = N if O = 0, E+O = N

↳ E+0 = N ↳ zero } if C_2 = 0 ✗

C_2 = 1

let E = 5

E+0+C_2 = N

N = 6

$\Rightarrow N + R = E$

↳ $6 + R = 5$ $\quad C_1 = 1$ ⟶ $6 + 8 + 1 = 15$

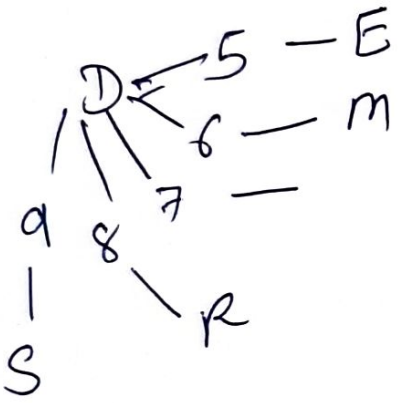
let $R = 9$ → $S = 9$ ⟶ R.

$6 + 9 = 15$ Not Possible.

$\Rightarrow D + E = Y$

$D + 5 = Y \quad = 7 + 5 = Y$

$D \geq 5 \quad = Y = 12.$



Prof. Dhaval Khedr.

⇒ Starting with left most

$\boxed{D = 1}$, $C_5 = 1$ (carry).

↳ $S + S = R$

↳ always Even

Two same numbers added always given even number.

Possible value of R. [0 to 9]

let $S = 2$,

$S + S = R$

$2 + 2 = R$

$\boxed{\begin{array}{l} R = 4 \\ S = 2 \end{array}}$

Prof. Dhaval Khatri

$\Rightarrow C + R = A$

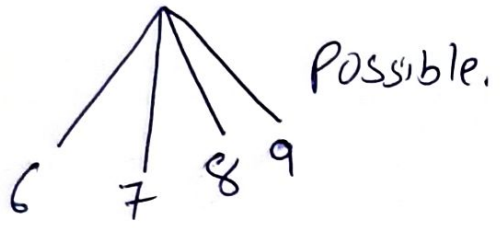
$\hookrightarrow C + R = A + 10$

$C_4 + C + R = A + 10$

$0 + C + 4 = A + 10$

$C = A + 6$

$C \geq 6$



$C = 9$ Suppose :

$R = 4$

$S = 2$

— It will be conflict

$C + R = A$

$9 + 4 = A$

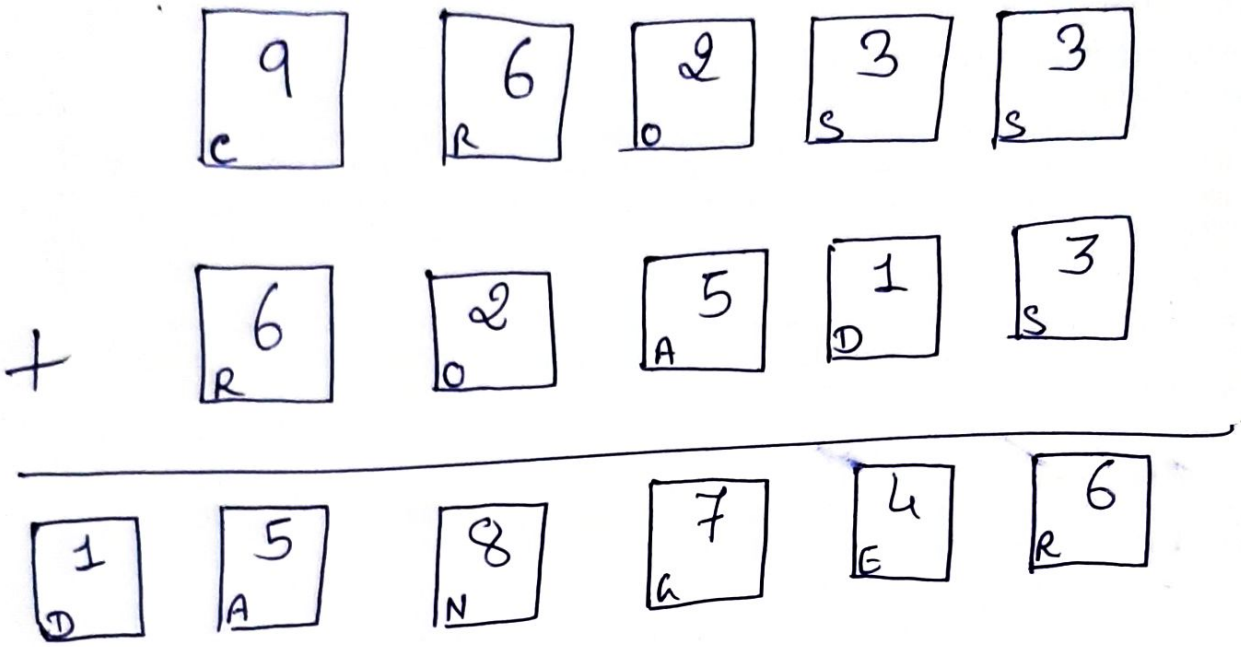
$A = 13$

↳ We have already 3 for E

Element

Ex

```
  C R O S S
  R O A D S
-----------
D A N G E R
```



D = 1
C = 9
R = 6
A = 5
O = 2
N = 8
G = 7
S = 3
E = 4

Prof. Dhaval Ichaporia

$\Rightarrow S = 3$

$S + S = R$

$3 + 3 = R$

$\boxed{R = 6}$

$\boxed{S = 3}$

$\Rightarrow R + O = N$

$6 + O = N$ — max value = 9

We have carry

$(6, 1)$ = Not possible (conflict with ①)

$(6, 2)$ = Possible.

$(6, 3)$ =

$\boxed{O = 2}$

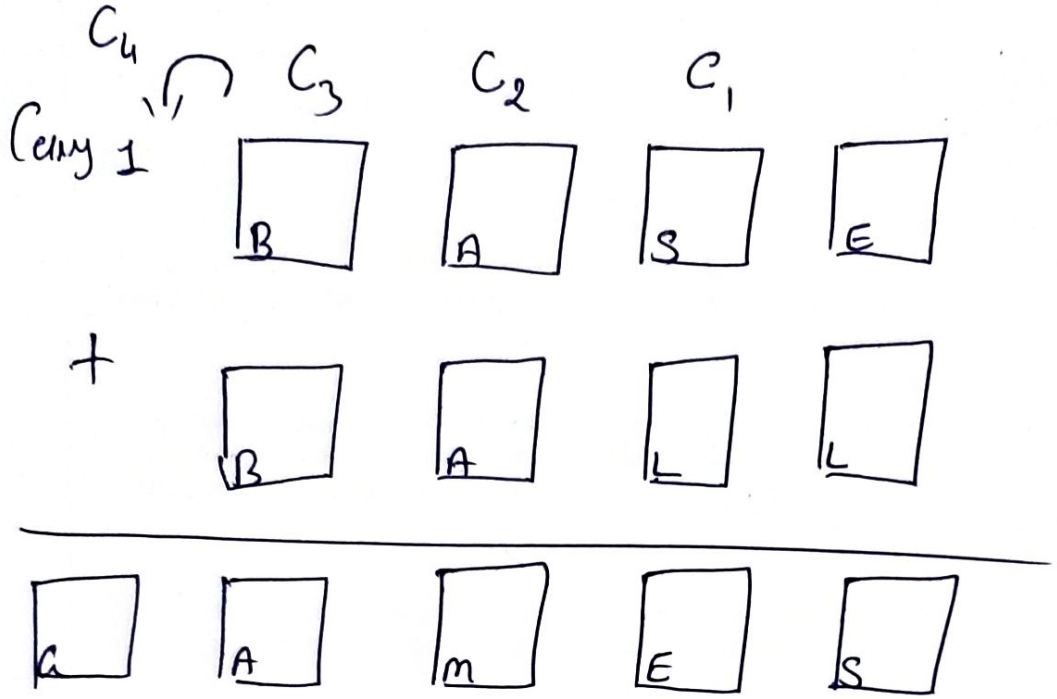
$6 + 2 = N$

$\boxed{N = 8}$

Solved Example

☆ Ex

```
  B A S E
+ B A L L
---------
G A M E S.
```



$\Rightarrow G = 1$

Prof. Dhaval Khetri

→ Start 2 (unb.

Side 1 → 2 (2 (amible)

Side 2 → Side 1 (1 (am)

Side 1 → Side 2 (2 (am)

Side 2 → Side 1 (1 (am)

Side 1 → Side 2 (2 m)

↳ 2 m, 2 c

Side 2 (1, m, 1c) → Side 1.

Side 1 (2m) → Side 2 (3m, 1c)

Side 2 (1 c) → Side 2 (3m)

Side 2 (2 c) → Side 2 (3m, 2c)

Side 2

⇒ Informed Search

⇒ The Searching Techniques.

↳ In the previous chapter we have seen the concept of uninformed search or blind search.

↳ Now we will see more efficient Search strategy, the informed Search Startegy.

↳ that uses problem - specific knowledge beyond the defination of the problem itself.

- The general method followed in informed Search is best first Search.
- BFS is similar to graph Search or tree Search algorithm wherein node expension is done based on certain criteria.

Prof. Dharmi Khatri

* Informed Search

↳ Information about Goal state is present.

↳ Better than Uninformed Search.

↳ Two way using

① Heuristic Search

② Heuristic function

↳ It finds optimal soln to reach Goal state:

- min Path cost.

using HEURISTIC FUNCTION

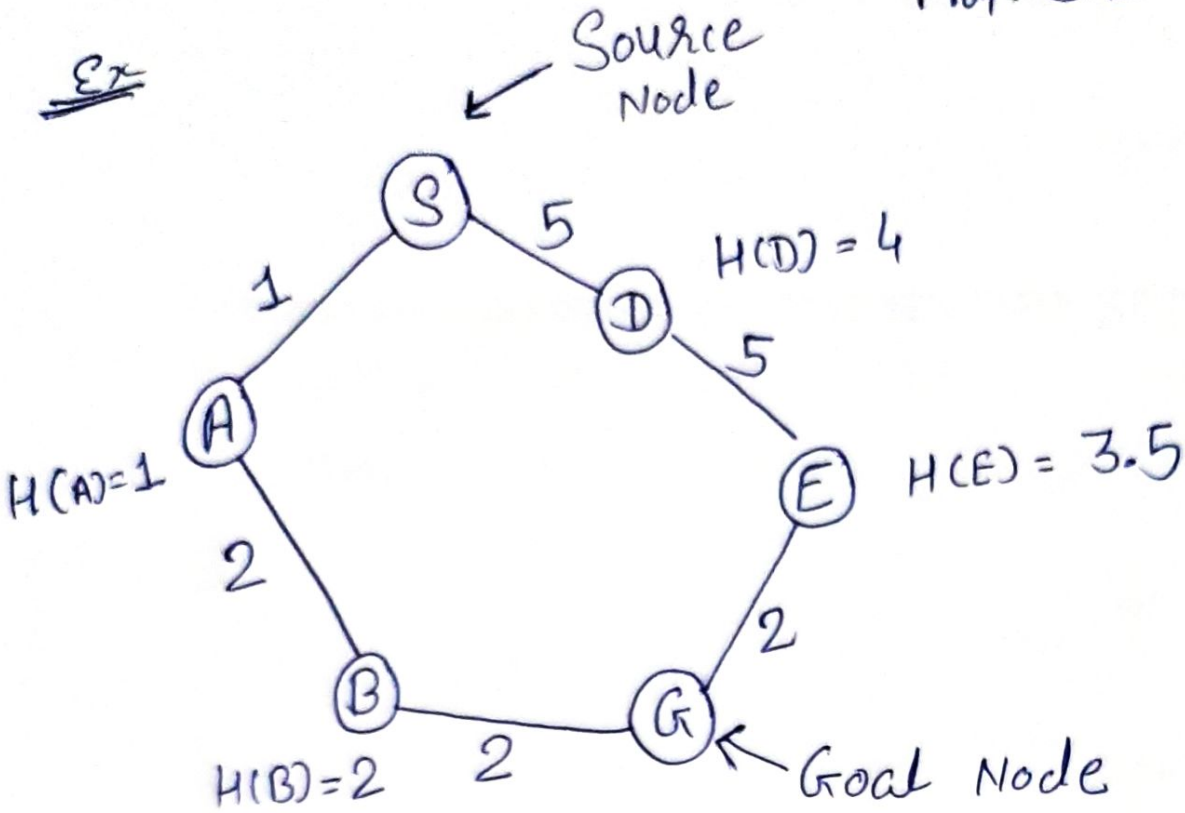
↳ It is a Search which tries to reduce amount of Search that must be done by making intelligent choices for the nodes that are selected for expansion.

Eg: 1) Best first Search Algo

2) A* Search Algo

Prof. Dhaval Khati

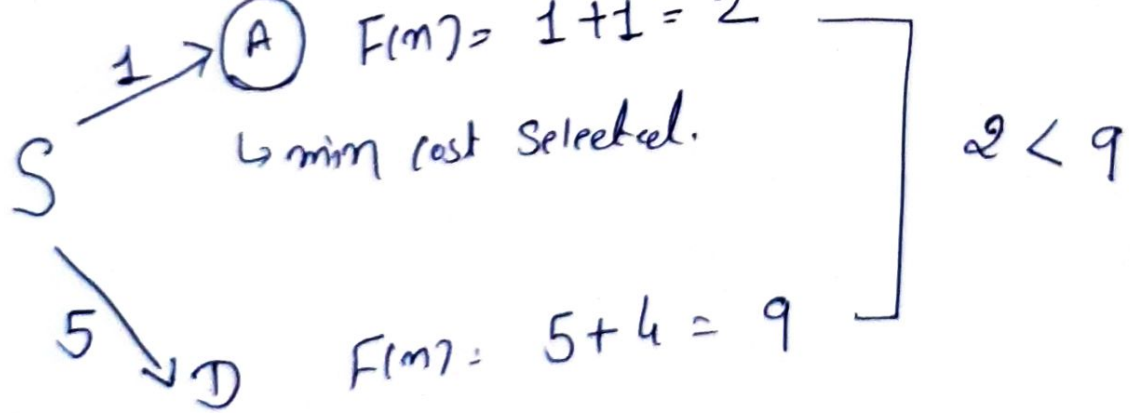
Ex



$$f(n) = G(n) + H(n)$$ → for every Node

- $f(n)$: Overall cost
- $G(n)$: Path cost
- $H(n)$: Heuristic value.

$\Rightarrow A = 1 + 1 = 2$



$S \rightarrow A \rightarrow B \quad F(n) = 3 + 2 = 5$

$S \xrightarrow{1} A \xrightarrow{2} B \xrightarrow{2} G = 5$

☆ Heuristic Search: [Informed Search]

→ Tries to optimize a problem using heuristic function. → Tries to solve problem in minimum steps/cost.

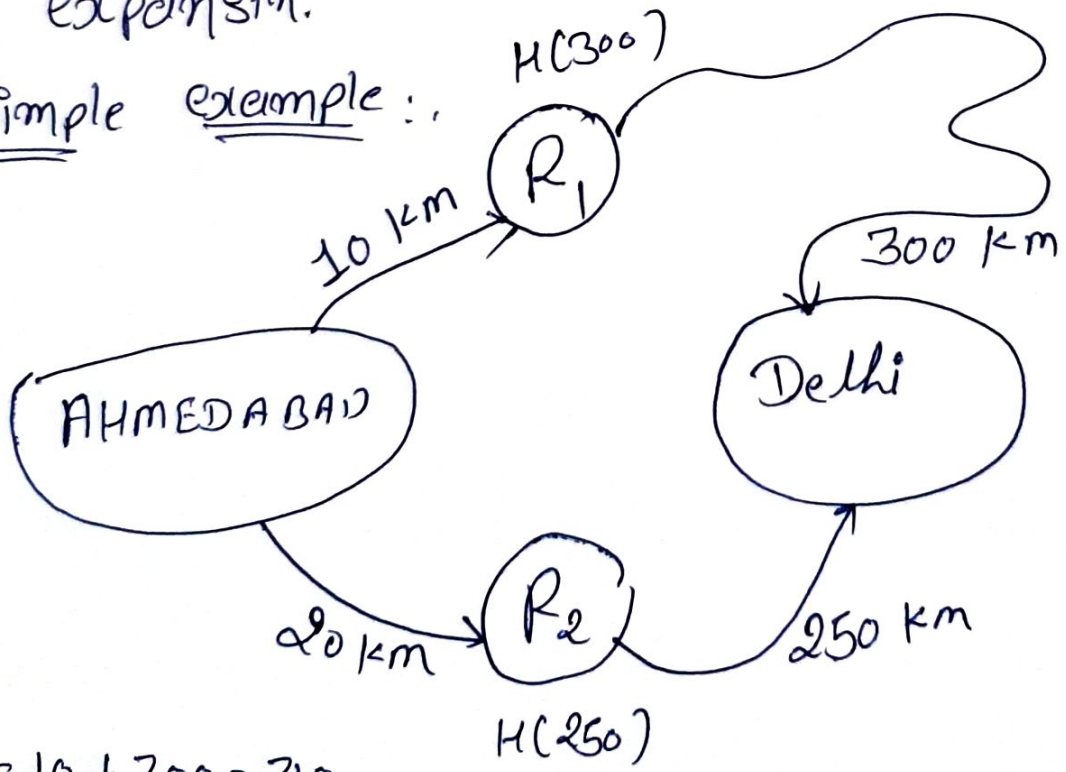
Heuristic Function:

It is a function H(n) that gives an estimation on the cost of getting from node 'n' to the goal state.

↳ means estimated value

Helps in selecting optimal node for expansion.

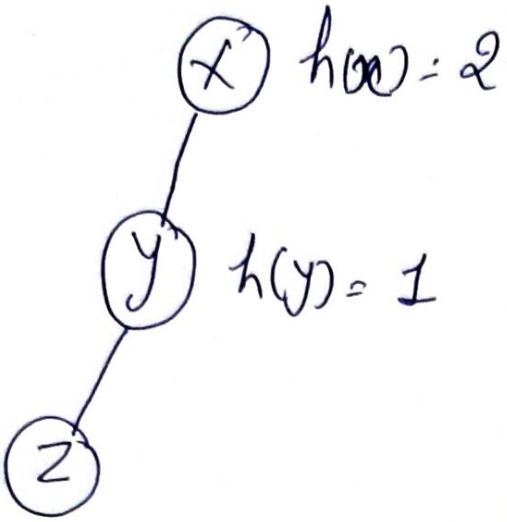
- Simple example:



$R_1 = 10 + 300 = 310$

$R_2 = 20 + 250 = 270$ ← Selecting R_2 (min cost)

Ex Node



↳ Goal node.

Types of Heuristic

① Admissible

② Non - Admissible.

① Admissible :- (underestimates)

In this Heuristic function never overestimates the cost of reaching the goal.

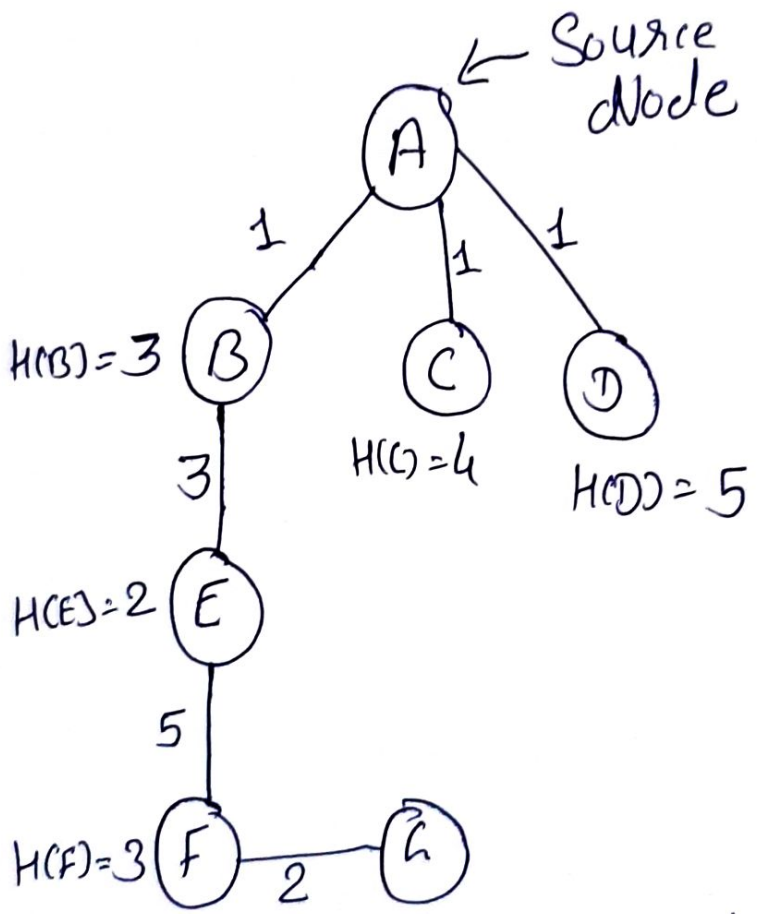
↳ h(n) is always less than or equal to actual cost of lowest-cost path from node 'n' to goal.

$H(n) \leq H'(n)$ } Goal

(2) Non-Admissible:

In this Heuristic function, overestimates the cost of reaching the goal.

$$H(n) > H'(n)$$



H(B) = 3, H(C) = 4, H(D) = 5

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

B = 1+3 = 4 ← minimum

C = 1+4 = 5

D = 1+5 = 6

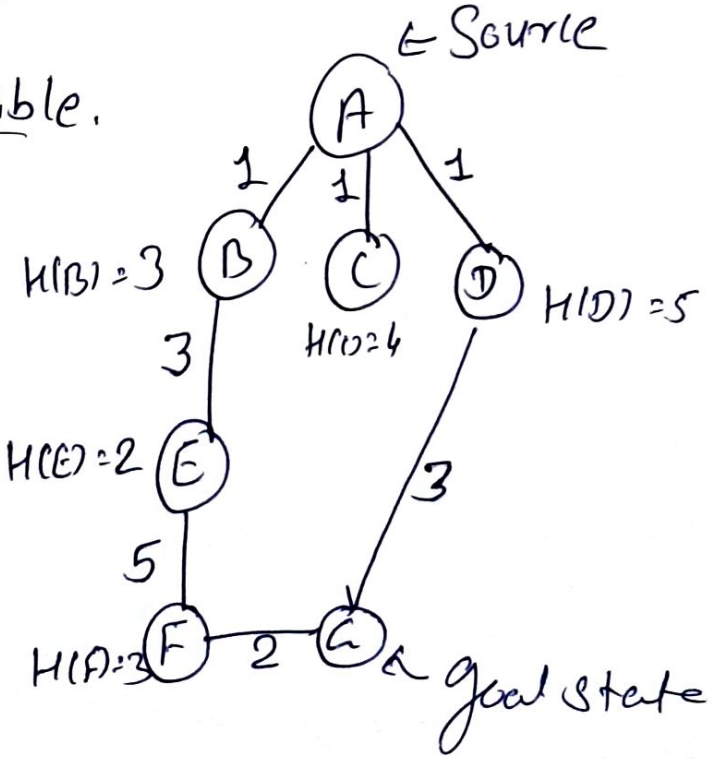
→ Actual Cost = 11

↳ 1+3+5+2

B Cost = 3

3 < 11

→ Non-Admissible.



H(D) = 5

A → D → G

1 3

4

5 > 4

✱ Generate and Test Search in AI:

In this Search Algo which uses depth first Search Technique.

In this Technique all the Solution are generated and tested for best Solution.

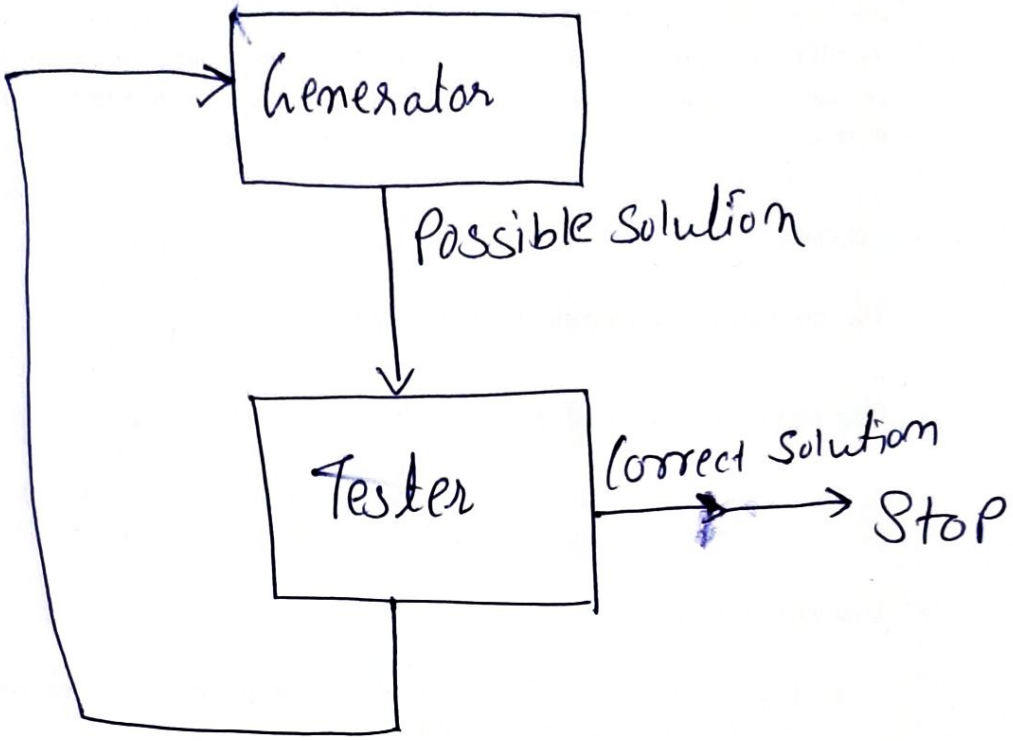
A heuristic is needed so that the Search is improved

Algorithem

1) Generate a possible Solution which can either be a point in the problem space or a path from the initial state.
2) Test to see if this possible Solution is a real (actual) Solution by comparing the state reached with the set of goal states.

Prof. Dhaval Khatri

3) If it is Real Solution then Return the Solution otherwise Repeat from Stat 1.



- Generator and Test is acceptable for Simple Problems whereas it is inefficient for Problems with large Spaces.

★ BEST FIRST SEARCH (GREEDY Search).

↳ It is uses evaluation Algorithm (fun^{n}) to decide which adjacent node is most promising and then explore.

↳ Category of Heuristic or Informed Search.

↳ Priority Queue is used to store cost of nodes.

↳ It's working both BFS and DFS.

↳ In the best first search algo, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function.

$$f(n) = g(n)$$

Where $h(n)$ = estimated cost from node n to the goal.

⇒ Algorithm

Priority Queue 'PQ' containing initial states.

Loop.

if PQ = Empty Return FAIL

ELSE

NODE ← Remove_First (PQ)

if

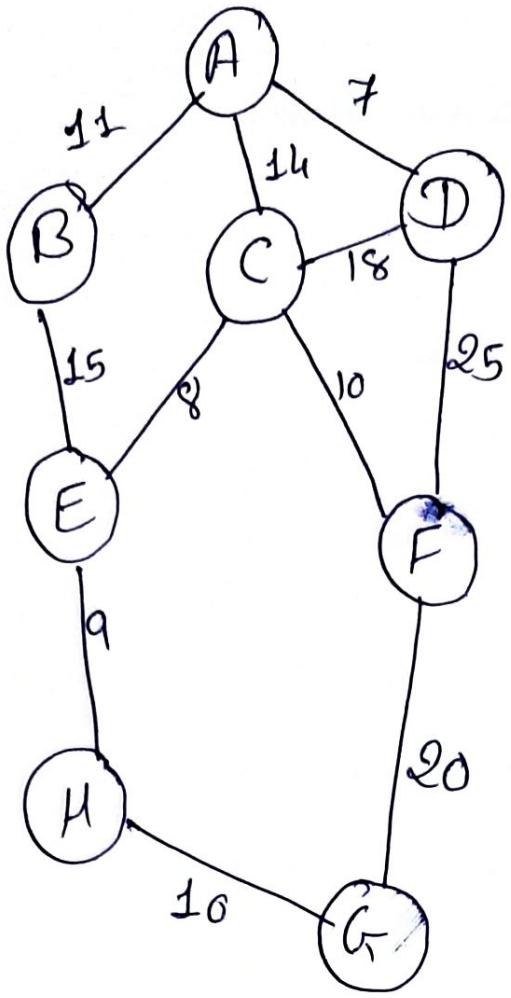
NODE = GOAL

Return Path from Initial NODE

ELSE

Generate all Successors of NODE and insert newly generated NODE into PQ according to cost value.

Ex



⇒ Given Straight line distance.

A → G = 40

B → G = 32

C → G = 25

D → G = 35

E → G = 19

F → G = 17

G → G = 0

H → G = 10

Prof. Dhaval Khatri

Prof. Dhaval Khatri

⇒ We are using two list which are OPEN and CLOSED list.

⇒ following are the iteration for traversing the above example.

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

↓ Al-De put it down. (Successors)

↳ (Successors)

⇒ Path

A → C → F → G

└ this is best first Search.

(Distance = 44)

⇒ Time Complexity:

The Worst case time Complexity of Greedy best first Search $O(b^m)$

⇒ Space Complexity:

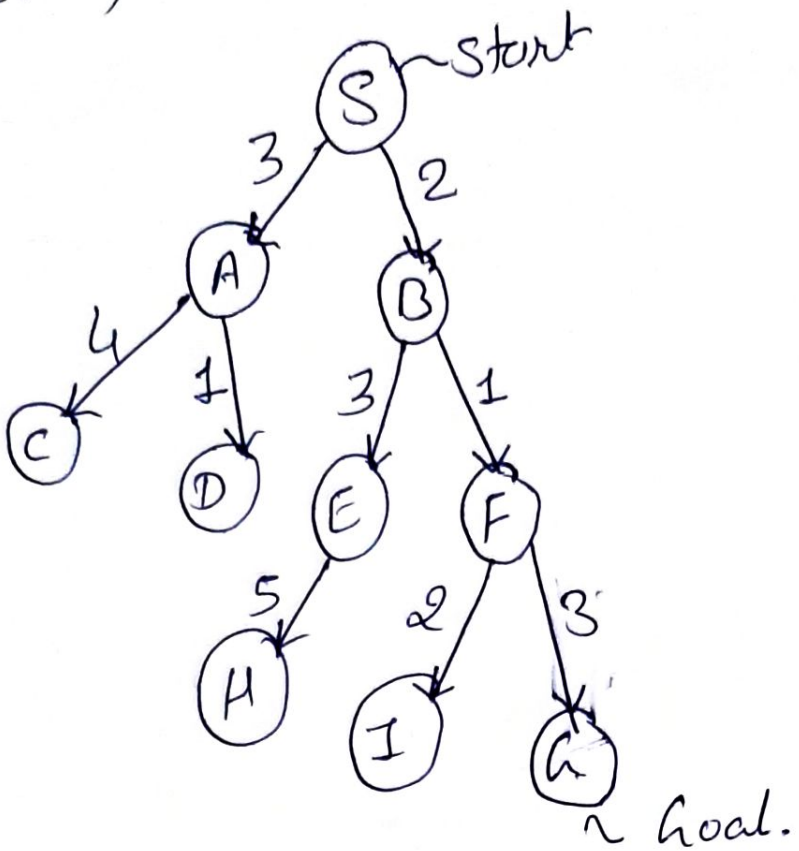
The Worst case time Complexity of Greedy best first Search $O(b^m)$, where m is the maximum depth of the search space.

⇒ Complete:

Greedy best first Search is also in complete, even if the given state space is infinite.

⇒ optimal:- Greedy best first Search Algo is not optimal.

Ex Best first Search.



⇒

NODE (n)	H(n)
A	12
B	04
C	07
D	03
E	08
F	02
H	04
I	09
S	13
G	00

	Open	Closed
→	[S]	[ ]
→	[B, A]	[S]
→	[F, E, A]	[S, B]
→	[G, I, E, A]	[S, B, F]
→	[I, E, A]	[S, B, F, 4]

Path: S → B → F → G

Ex

✱ A* Search Algorithm

A* Search is the most commonly known form of best-first Search.

It uses heuristic function h(n), and cost to reach the node 'n' from the start state g(n).

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

- $f(n)$: estimated cost.
- $g(n)$: cost to reach node
- $h(n)$: heuristic value (child node)

finds shortest path through Search space.

fast and optimal result.

✱ NOTE: At each point in the search space, only those node is expanded which have the lowest value of $f(n)$, and the Algo terminates when the goal node is found

Prof. Dhaval Khatri

* Algorithm:

i) Enter starting node in open list

ii) if
open list is empty Return FAIL

iii) Select node from open list which has Smallest value (g+h).

↳ if node = Goal, Return success.

iv) Expand node 'n' and generate all Successors.

↳ Compute (g+h) for each Successor node.

v) if node 'n' is already in open/closed, attach to back pointer.

vi) go to (ii)

Advantage:

1) A* Search algo is the best algo than Other Search Algo

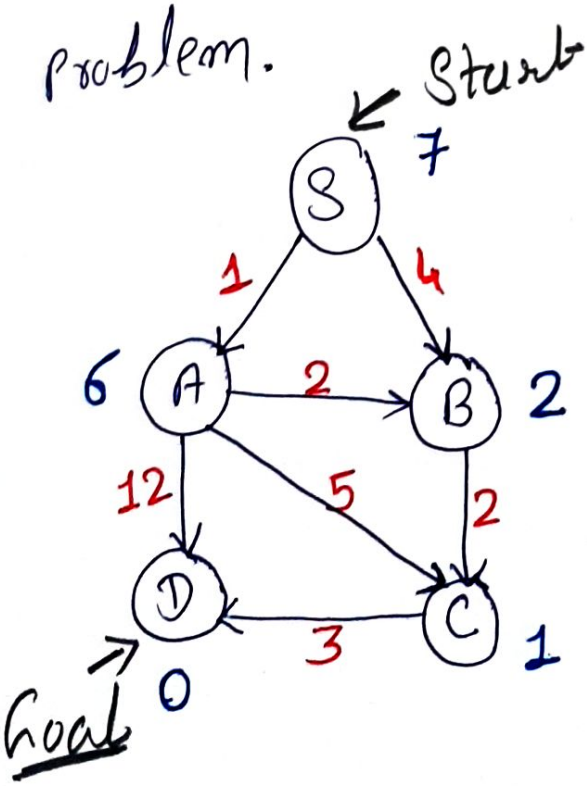
2) A* Search algo is optimal and Complete.

3) This algo can solve very complex Problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristic and approximation.
- A* Search Algo has some Complexity issues.
- The main drawback A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various lerge scale problem.

Ex



⇒ Starting node : S

$S \rightarrow A$

$S \rightarrow B$

$S \rightarrow A : f(n) = g(n) + h(n)$

$= 1 + 6$

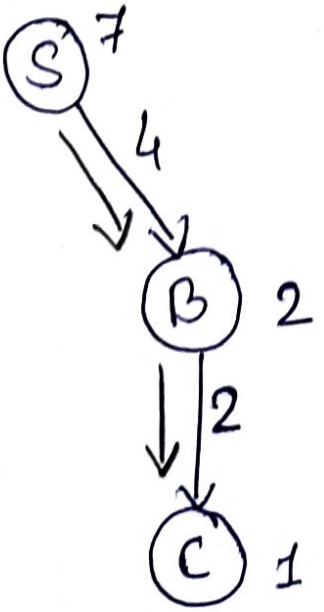
$S \rightarrow A = 7$ ← 3rd

$S \rightarrow B = f(n) = g(n) + h(n)$

$= 4 + 2$

$S \rightarrow B = 6$ ← 1st

min value expand



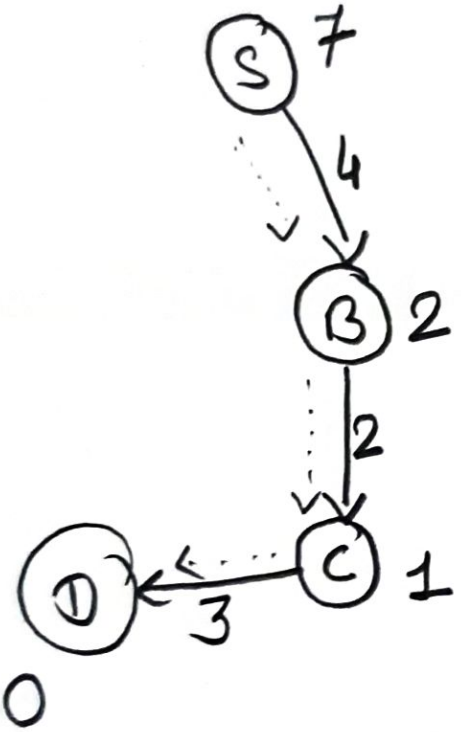
$\Rightarrow S \rightarrow B \rightarrow C$

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

$= 4 + 2 + 1$

$S \rightarrow B \rightarrow C = 7$ → 2nd

→ further expand.



$S \rightarrow B \rightarrow C \rightarrow D$:

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

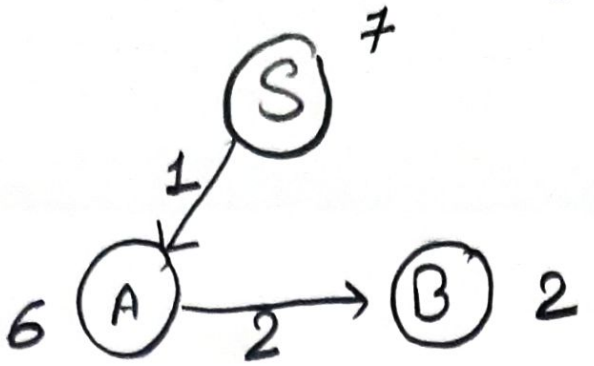
$= 4 + 2 + 3 + 0$

$= 9$

$S \rightarrow B \rightarrow C \rightarrow D = 9$ → This is one Path Initial State & Goal State

Ans=

→ There are two possible way expand.



⇒ S → A → B

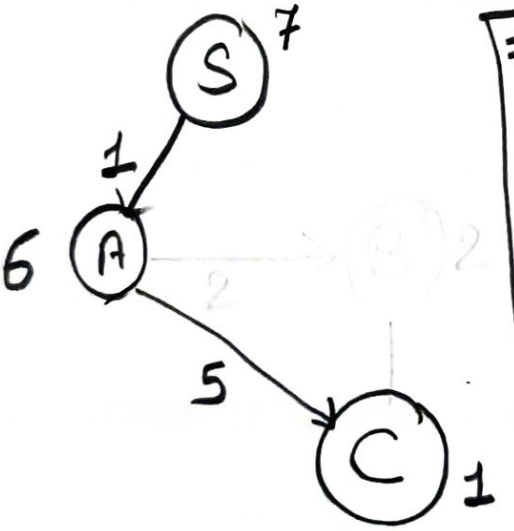
$f(n) = g(n) + h(n)$

$= 1 + 2 + 2$

$= 5$

S → A → B = 5	← 4th

further expand.



⇒ S → A → C

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

$= 1 + 5 + 1$

$= 7$

S → A → C = 7	← 6th

Further expanded

S 7, 1, A 6, 12, D 0

⇒ S → A → D

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

$= 1 + 12$

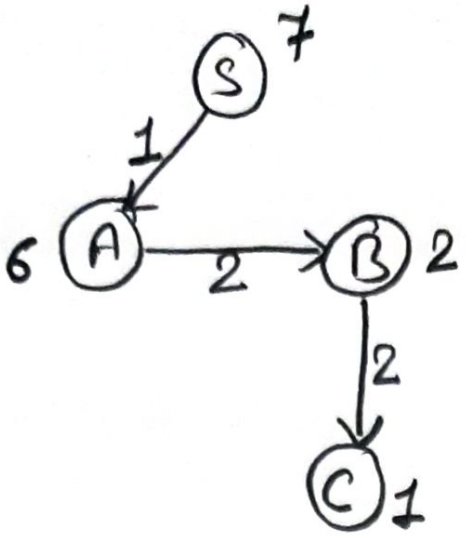
$= 13$

S → A → D = 13

One path.

Prof. Dhaval Khatri

$\Rightarrow S \rightarrow A \rightarrow B \rightarrow C$



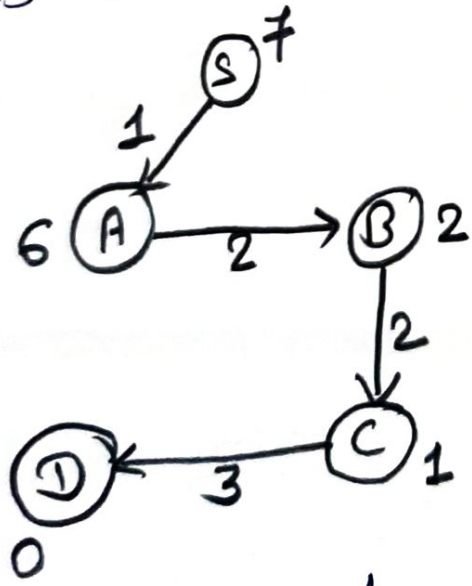
$f(n) = g(n) + h(n)$

$= 1+2+2+1$

$= 6$

$S \rightarrow A \rightarrow B \rightarrow C = 6$ ← 5th further expand.

$\Rightarrow S \rightarrow A \rightarrow B \rightarrow C \rightarrow D$



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

$S \rightarrow A \rightarrow B \rightarrow C \rightarrow D$

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

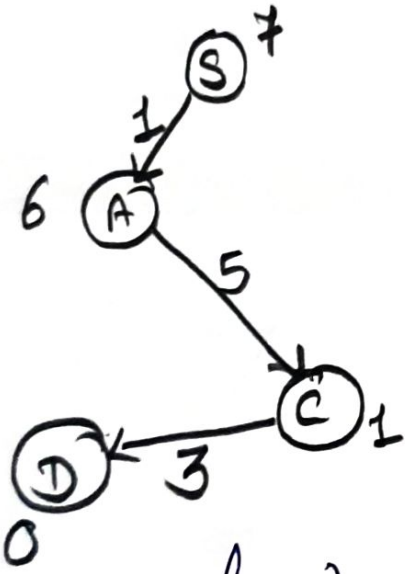
$= 1 + 2 + 2 + 3 + 0$

$= 8$

P_3= $S \rightarrow A \rightarrow B \rightarrow C \rightarrow D = 8$ ← Path

↑ This is Smallest path

$\Rightarrow S \rightarrow A \rightarrow C \rightarrow D.$



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

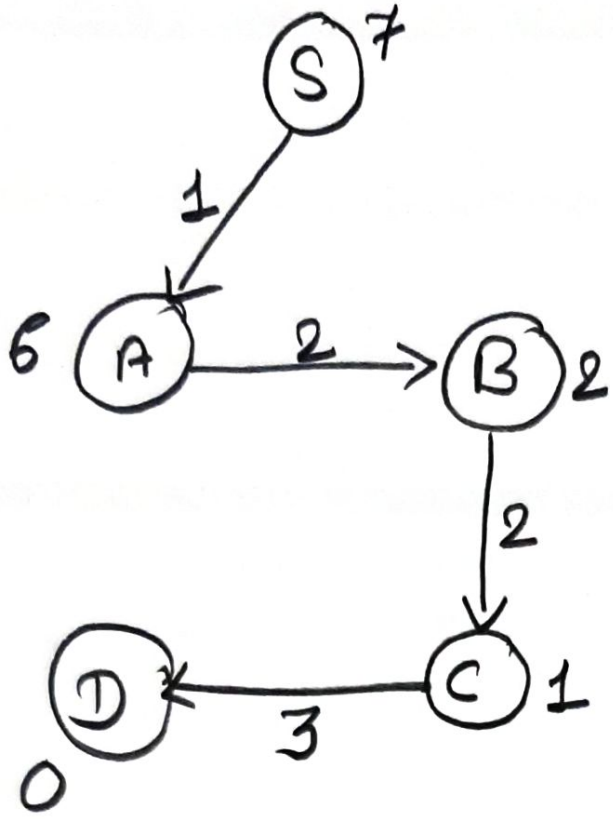
$= 1 + 5 + 3 + 0$

$= 9$

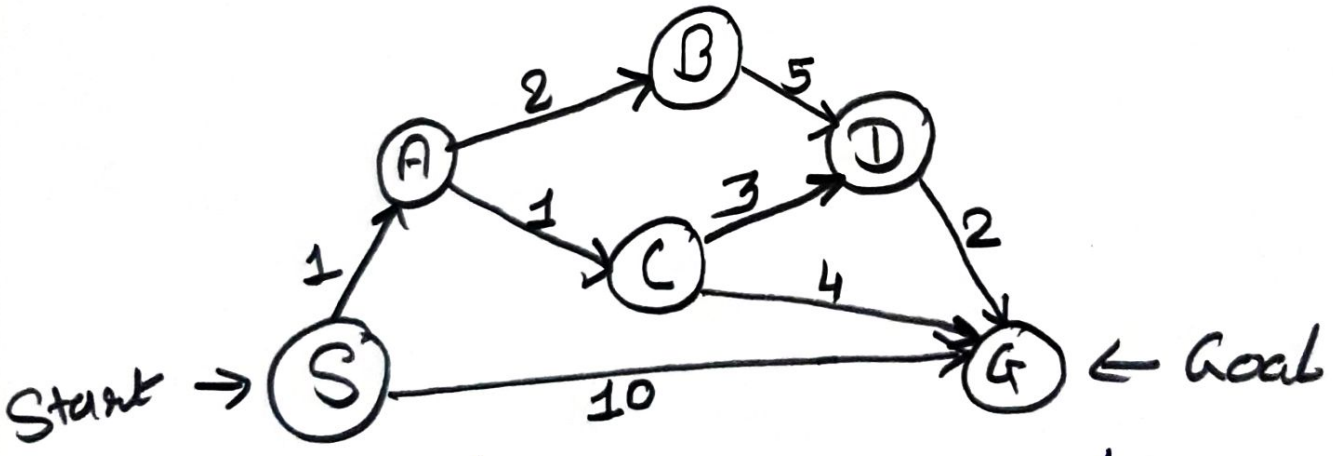
P_4= $S \rightarrow A \rightarrow C \rightarrow D = 9$ ← Path

Final Path

Prof. Dhaval Khatri



Ex Example of A* Algorithm:



State	h(n)
S	5
A	3
B	4
C	2

State	h(n)
D	6
G	0

Sol^n :

$S \to A$

$S \to G$

$\Rightarrow S \to A$

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

$= 1 + 3$

$S \to A = 4$ ← 1st explore.

$\Rightarrow S \to G$

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

$= 10 + 0$

$S \to G = 10$ ← path

⇒ There are two possible way.

$S \to A \to B = 1 + 2 + 4 = 7$ ← 3rd expand

$S \to A \to C = 1 + 1 + 2 = 4$ ← 2nd expand

⇒ $S \to A \to C$ Two way expand.

$S \to A \to C \to D = 1 + 1 + 3 + 6 = 11$ ← 4th expand

$S \to A \to C \to G = 1 + 1 + 4 + 0 = 6$ ← path.

$\Rightarrow S \to A \to B \to D = 1+2+5+6 = 14$

↑ 5th around

$\Rightarrow$

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

$1+1+3+2+0$

$= 7$

$S \to A \to C \to D \to G = 7$ ← Path

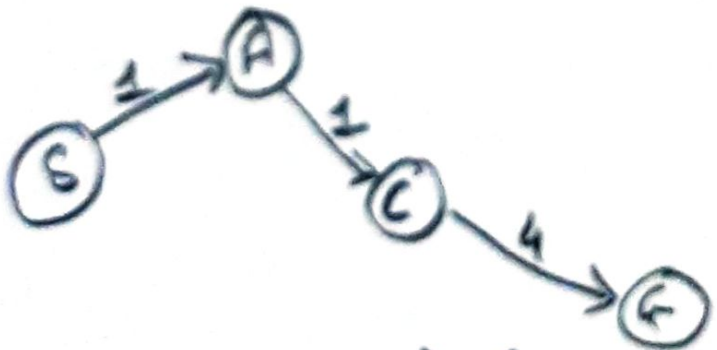
$\Rightarrow S \to A \to B \to D \to G$

$= 1+2+5+2+0$

$= 10$

$S \to A \to B \to D \to G = 10$ ← Path

final Path



Prof. Dhaval Khatri

$S \to A \to C \to G = 6$ (optimal cost)

complexity can be reduced by devising good heuristic function.

3.1.6 A* Search

The A* algorithm

1) A* is most popular form of best first search.
2) A* evaluates node based on two functions namely
   1) g(n) - The cost to reach the node 'n'.
   2) h(n) - The cost to reach the goal node from node 'n'.

These two function's costs are combined into one, to evaluate a node. New function f(n) is deviced as,

$$f(n) = g(n) + h(n) \text{ that is,}$$

f(n) = Estimated cost of the cheapest solution through 'n'.

Working of A*

1) The algorithm maintains two sets.
   a) OPEN list : The OPEN list keeps track of those nodes that need to be examined.
   b) CLOSED list : The CLOSED list keeps track of nodes that have already been examined.
2) Initially, the OPEN list contains just the initial node, and the CLOSED list is empty. Each node n maintains the following : g(n), h(n), f(n) as described above.
3) Each node also maintains a pointer to its parent, so that later the best solution, if found, can be retrieved. A* has a main loop that repeatedly gets the node, call it 'n', with the lowest f(n) value from the OPEN list. If 'n' is the goal node, then we are done, and the solution is given by backtracking from 'n'. Otherwise, 'n' is removed from the OPEN list and added to the CLOSED list. Next all the possible successor nodes of 'n' are generated.
4) For each successor node 'n', if it is already in the CLOSED list and the copy there has an equal or lower 'f' estimate, and then we can safely discard the newly generated 'n' and move on. Similarly, if 'n' is already in the OPEN list and the

copy there has an equal or lower f estimate, we can discard the newly generated n and move on.

5) If no better version of 'n' exists on either the CLOSED or OPEN lists, we remove the inferior copies from the two lists and set 'n' as the parent of 'n'. We also have to calculate the cost estimates for n as follows :

Set g(n) which is g(n) plus the cost of getting from n to n;

Set h(n) is the heuristic estimate of getting from n to the goal node ;

Set f(n) is g(n) + h(n)

6) Lastly, add 'n' to the OPEN list and return to the beginning of the main loop.

Performance measurement for A*

1) Completeness : A* is complete and guarantees solution.

2) Optimality : A* is **optimal if h(n)** is admissible heuristic. **Admissible heuristic** means h(n) never over estimates the cost to reach the goal. Tree-search algorithm gives optimal solution if h(n) is admissible.

If we put extra requirement on h(n) which is consistency also called as monotonicity then we are sure expect the optimal solution. A heuristic function h(n) is said to be consistent, if, for every node 'n' and every successor 'ns' of 'n' generated by action 'a', the estimated cost of reaching the goal from 'n' is not greater than the step cost of getting to 'ns'.

$h(n) \leq \text{cost}(n, a, ns) + h(ns)$

A* using graph-search is optimal if h(n) is consistent.

Note : The sequence of nodes expanded by A* using graph-search is in, increasing order of f(n). Therefore the first goal node selected for expansion must be an optimal solution because all latter nodes will be either having same value of f(n) (same expense) or greater value of f(n) (more expensive) than currently expanded node.

A* never expands nodes with $f(n) > C^*$ (where C^* is the cost of optimal solution).

A* algorithm also do pruning of certain nodes.

Pruning means ignoring a node completely without any examination of that node, and thereby ignoring all the possibilities arising from these node.

3) Time and space complexity : If number of nodes reaching to goal node grows exponentially then time taken by A* eventually increases.

The main problem area of A* is memory, because A* keeps all generated nodes in memory.

A* usually runs out of space long before it runs out of time.

A* optimality proof

- Assume A* finds the (sub optimal) goal G2 and the optimal goal is G.
- Since h is admissible

$$h'(G2) = h'(G) = 0$$

- Since G2 is not optimal

$$f(G2) > f(G)$$

- At some point during the search, some node 'n' on the optimal path to G is not expanded.

We know,

$$f(n) \leq f(G)$$

3.1.8 AO* Search and Problem Reduction Using AO*

When a problem can be divided in a set of sub problems, where each sub problem can be solved seperately and a combination of these will be a solution. AND-OR graphs or AND-OR trees are used for representing the solution.

AND-OR graphs

1) The decomposition of the problem or problem reduction generates AND arcs.
2) One AND arc may point to any number of successor nodes.
3) All these must be solved so that the arc will give rise to many arcs, indicating several possible solutions. Hence the graph is known as AND-OR instead of AND.

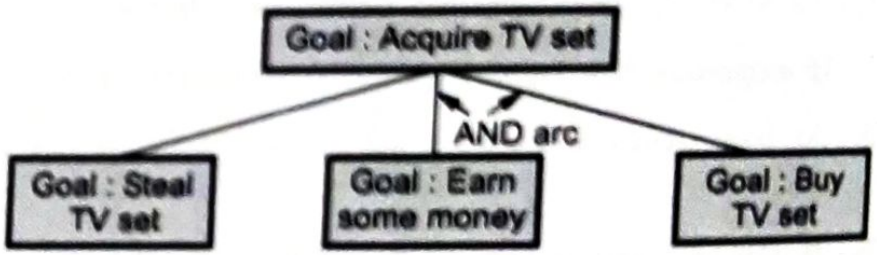


Fig. 3.1.2 [AND-OR graph example]

4) AO* is a best-first algorithm for solving problems represented as a cyclic AND/OR graphs problems.
5) An algorithm to find a solution in an AND-OR graph must handle AND area appropriately.

The comparative study of A* and AO*

1) Unlike A* algorithm which used two lists OPEN and CLOSED, the AO* algorithm uses a single structure G.
2) G represents the part of the search graph generated so far.
3) Each node in G points down to its immediate successors and upto its immediate predecessors, and also has with it the value of 'h' cost of a path from itself to a set of solution nodes.

4) The cost of getting from the start nodes to the current node 'g' is not stored as in the A* algorithm. This is because it is not possible to compute a single such value since there may be many paths to the same state.

5) AO* algorithm serves as the estimate of goodness of a node.

6) A* algorithm cannot search AND-OR graphs efficiently.

7) AO* will always find minimum cost solution.

- The algorithm for performing a heuristic search of an AND-OR graph is given below.

AO* algorithm

1) Initialize the graph to start node.

2) Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved.

3) Pick any of these nodes and expand it and if it has no successors call this value as, FUTILITY, otherwise calculate only f' for each of the successors.

4) If 'f' is 0 then mark the node as SOLVED.

5) Change the value of 'f' for the newly created node to reflect its successors by back propagation.

6) Wherever possible use the most promising routes and if a node is marked as SOLVED then mark the parent node as SOLVED.

7) If starting node is SOLVED or value greater than FUTILITY, stop, else repeat from

☆ AO* Algorithm

Prof. Dhaval Chauhan

Describe how Problem Reduction is done with AO* Algo.

→ AND-OR Graphs:-

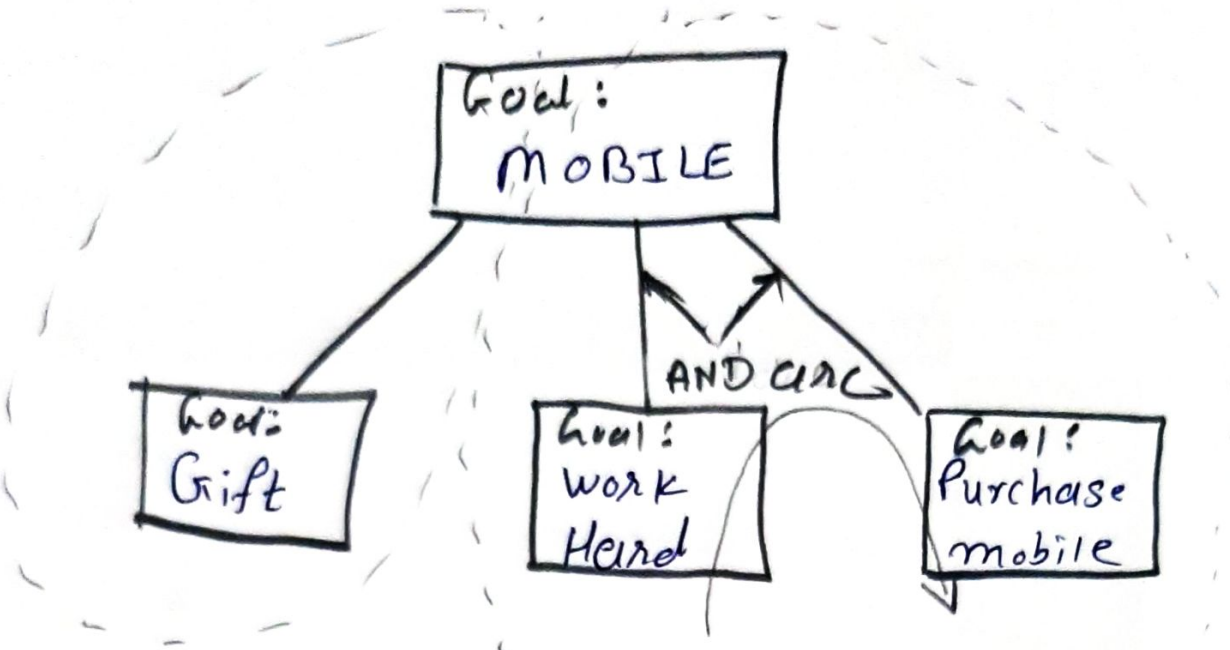
It is useful for Representing the Problem Solution that can be solved by decomposing them into Smaller set of problems, all of which must be solved.

When a Problem can be divided in a set of Sub problem, where each Sub Problem can be solved Seperately and a Combination of these will be a Solution.

AND ARCS → may Point to 'n' no of Successor nodes.

↳ Algo like Best fit Search can be used that has the ability to handle AND ARCS

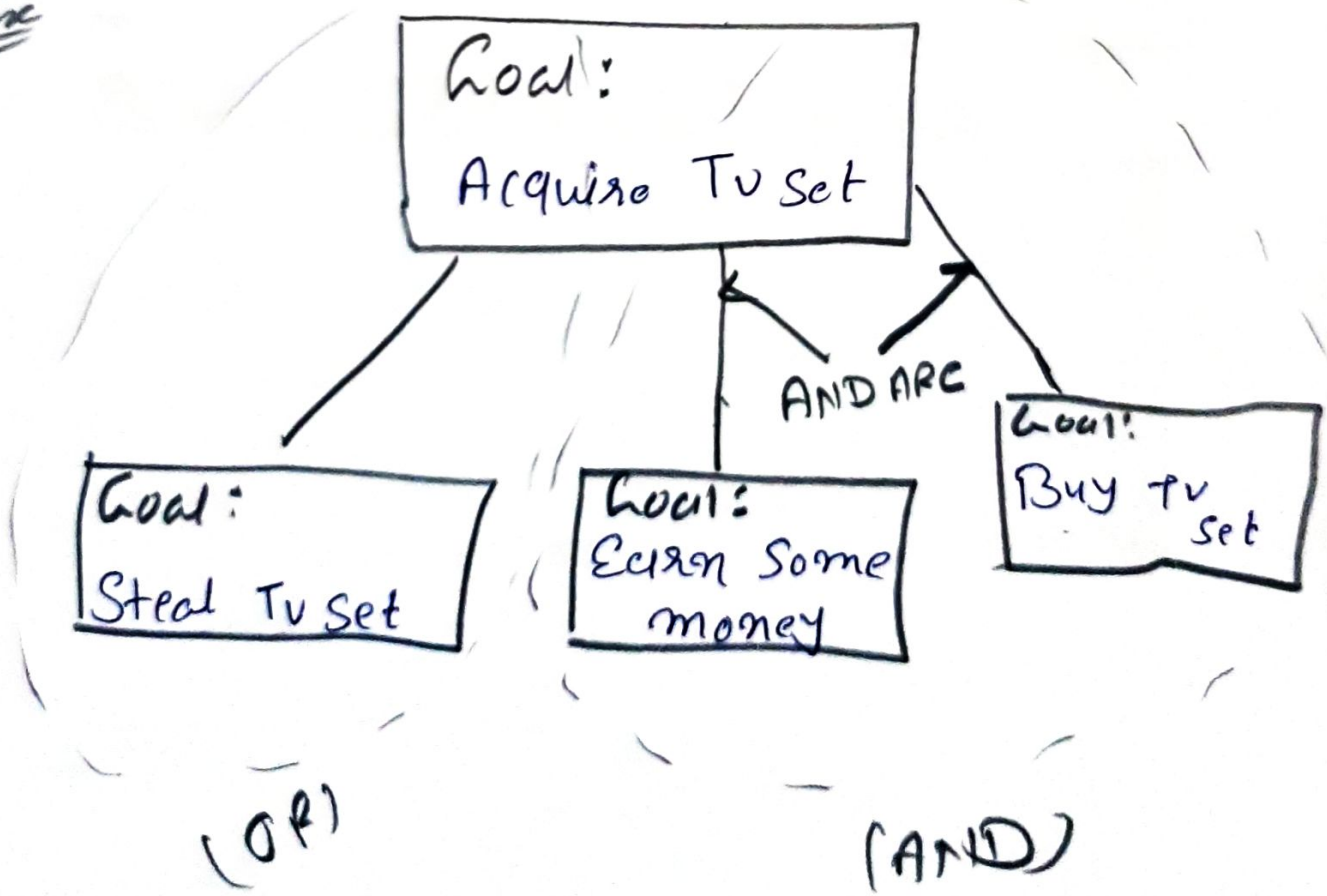
Exe



(OR)

↳(AND)

Exe



(OR)

(AND)

⇒ futility :- (Threshold).

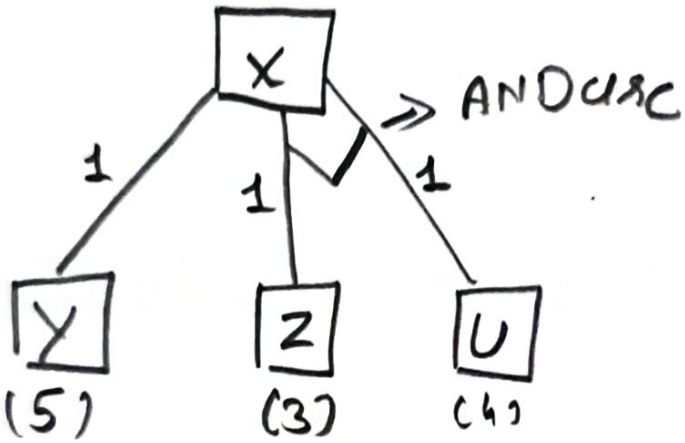
It is should be chosen to correspond to a threshold value.

if Est. Cost > fulility.

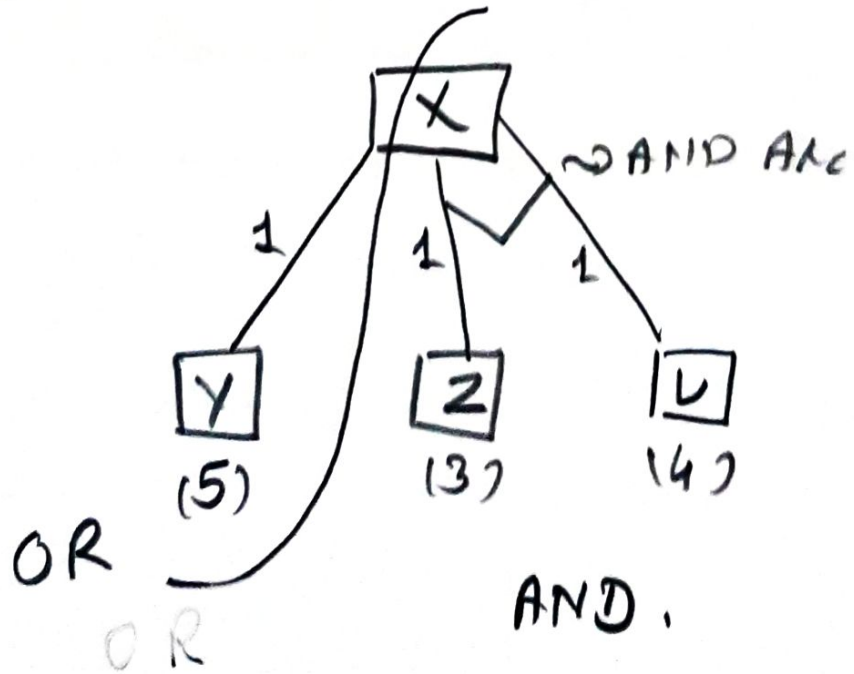
then stop Search.

+ A* does not explore all the solution paths once it got a solution.

Eg: A or Algo



⇒ Solution



OR

$x \rightarrow y = 1+5 = 6$ ✓

We choose this one bcoz Small value

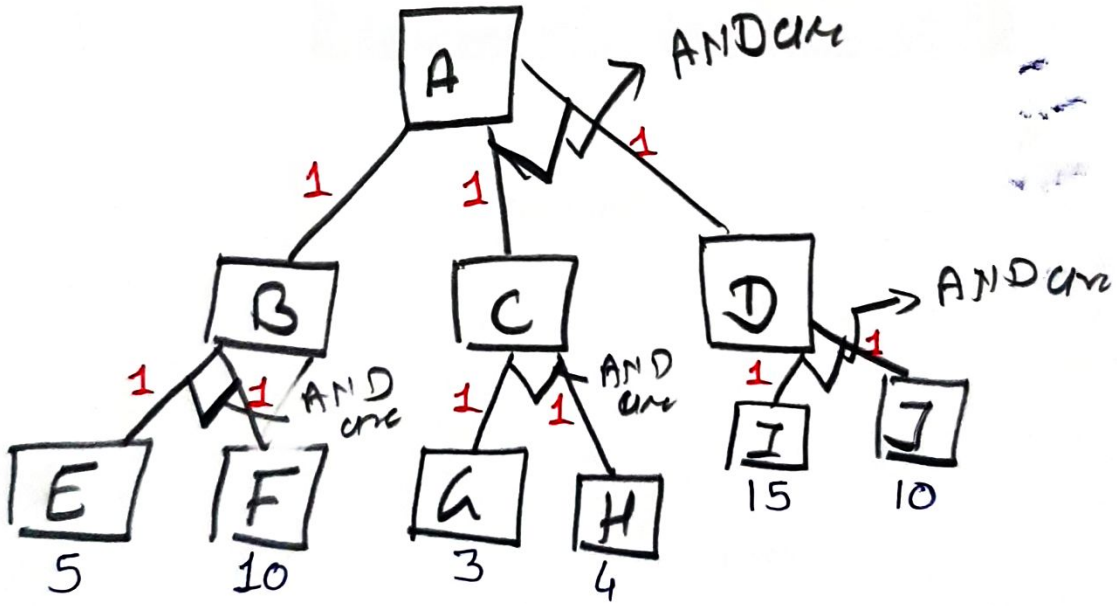
AND

$x \rightarrow z \rightarrow x \rightarrow v$

$= 1+3+1+4$

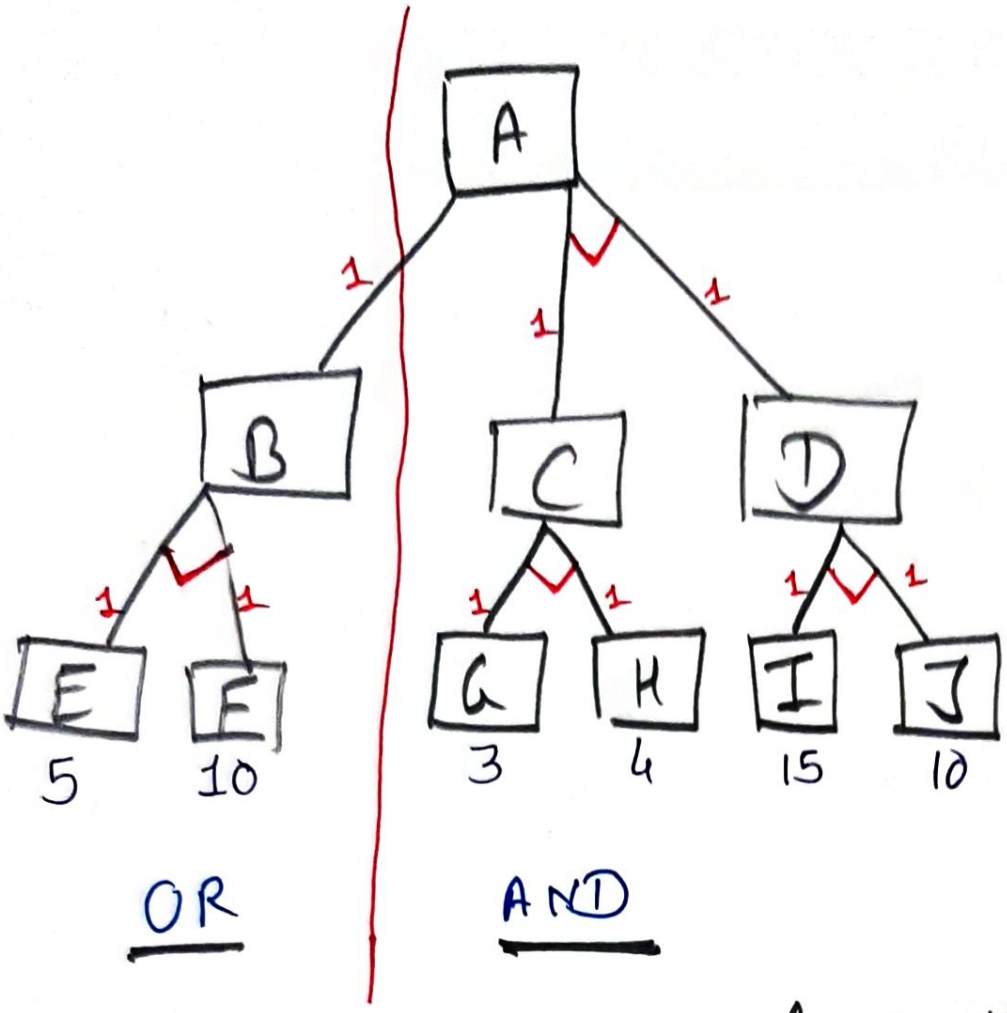
$= 9$

Ex Ans

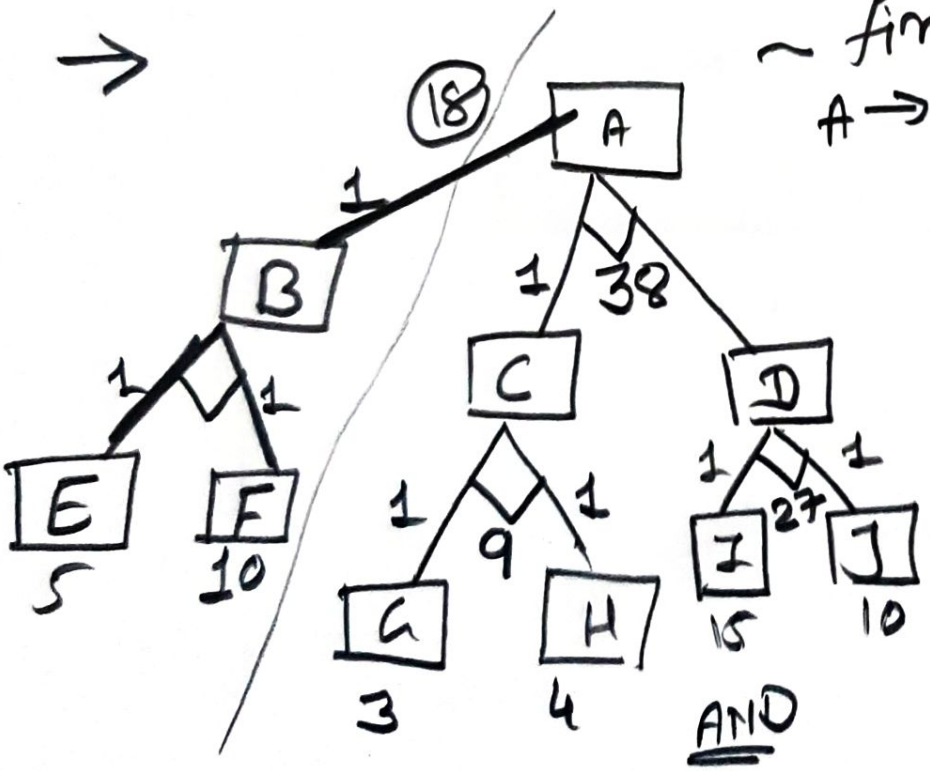


⇒ Soln

Prof. Dhaval Khatri



→



~ final O/P :-

A → B → E → F

A = 1 + 9 + 1 + 27

= 38

B = 1 + 5 + 1 + 10

= 17

A = 1 + 17 = 18

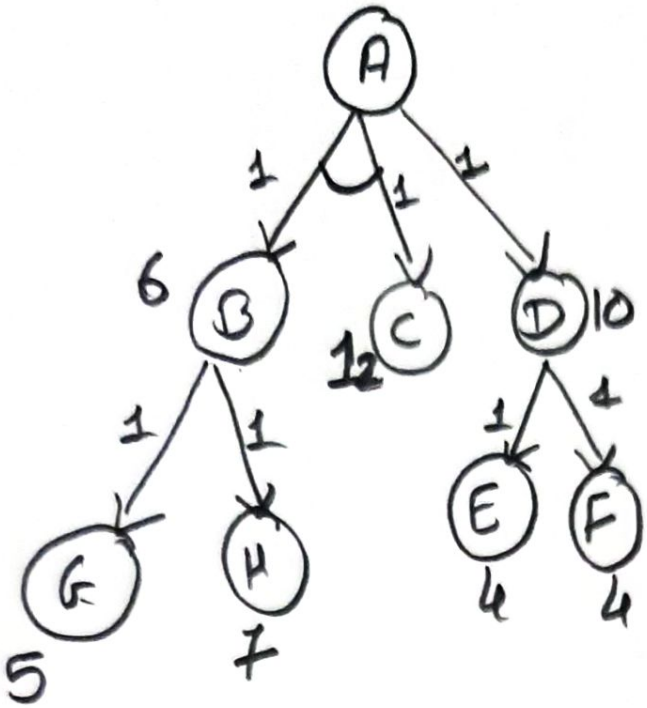
C = 1 + 3 + 1 + 4

C = 9

D = 1 + 15 + 1 + 10

= 27

Ex: Solved Ao+ Algo Example.



Prof. Dhaval Kheni

☆ Hill - climbing

Write Short note on "Hill - climbing"

Search.

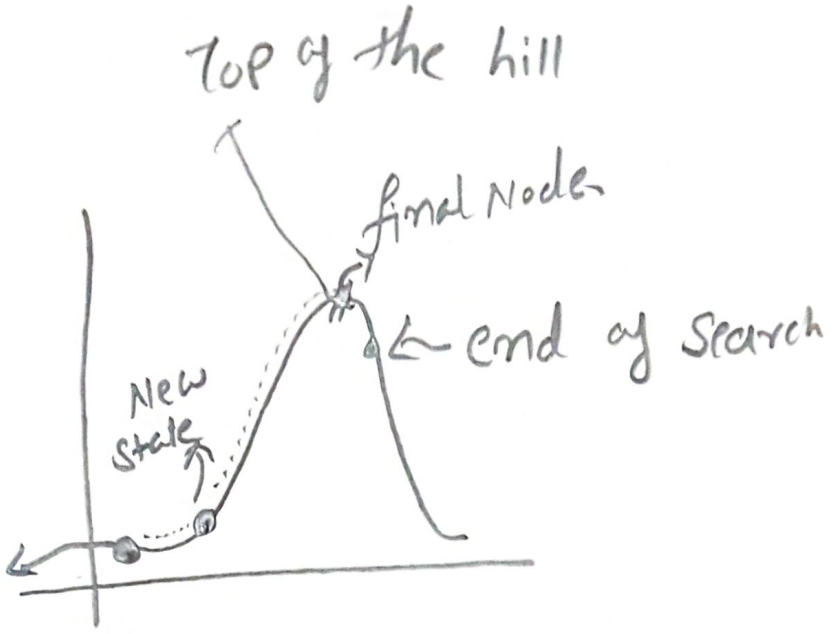
⇒ It is a variant of generate and test method in which feedback from test procedure is used to help generator decide which dirn to move in search space.

↳ It is always move in a single dirn.

↳ It is like DFS.

↳ Hill. climbing
  1. local Search
  2. Greedy Approach
  3. No Backtracking

Prof Dhaval khatri



if new state is better

then Current State

↓↓

New State = Current State

⇒ Top of the hill

↓↓

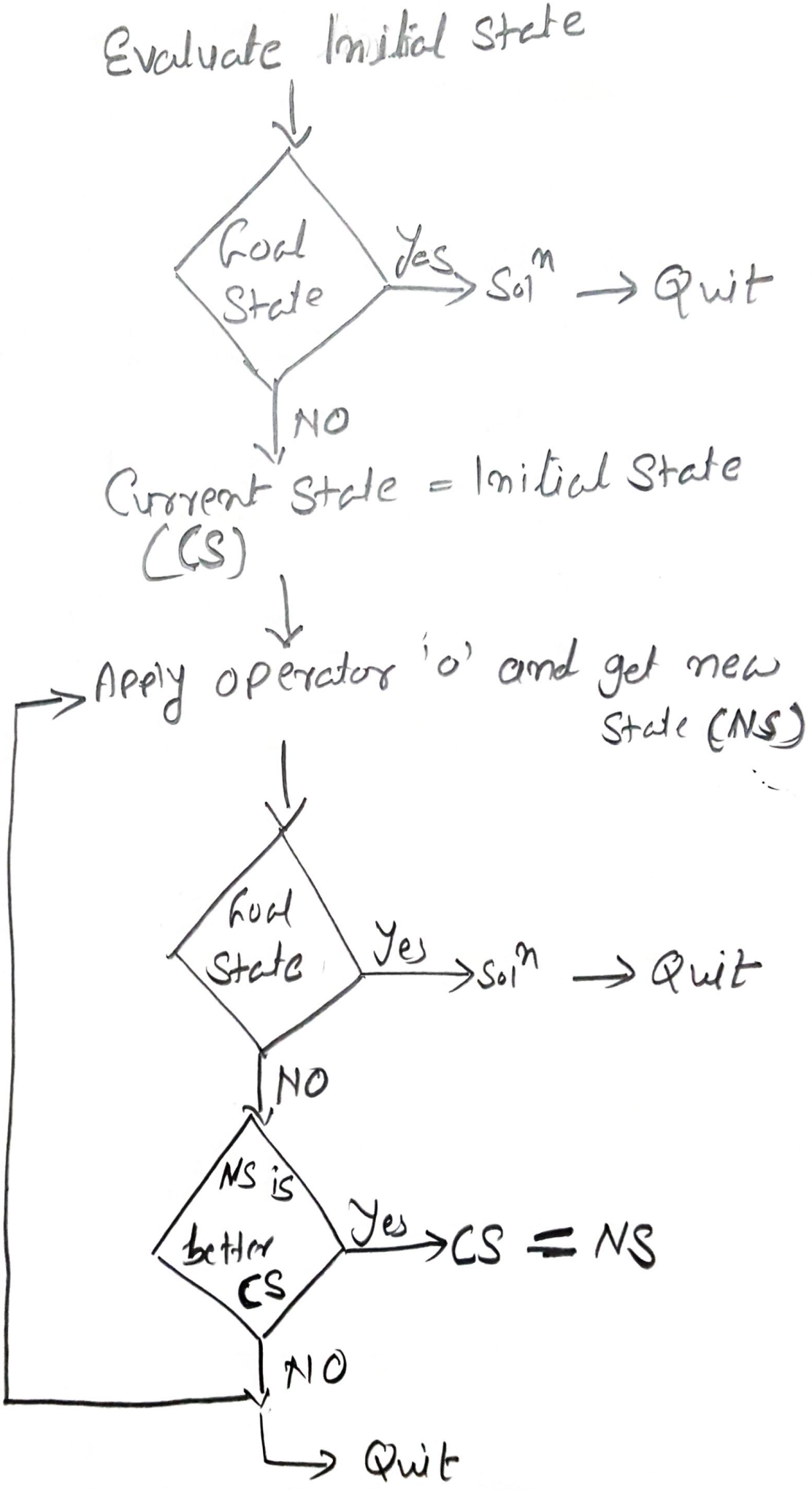
Search down side

It is not better

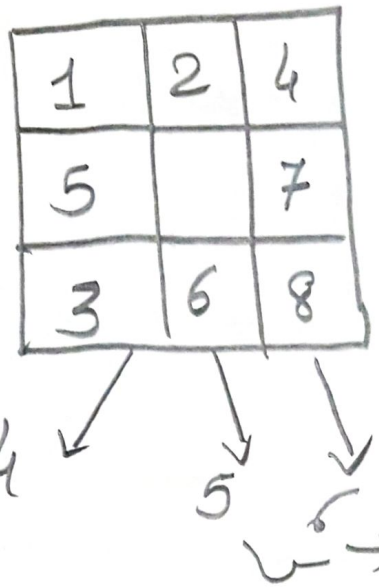
than end of Search

→ Hill Climbing Search one way

✯ Flow chart


Evaluate Initial State
Goal State
Yes
Soln → Quit
NO
Current State = Initial State (CS)
Apply operator 'o' and get new State (NS)
Goal State
Yes
Soln → Quit
NO
NS is better CS
Yes
CS = NS
NO
Quit

Eg Hill climbing



(Starting State)

→ discard this value.

→ We will choose this.

⇒ limitations:

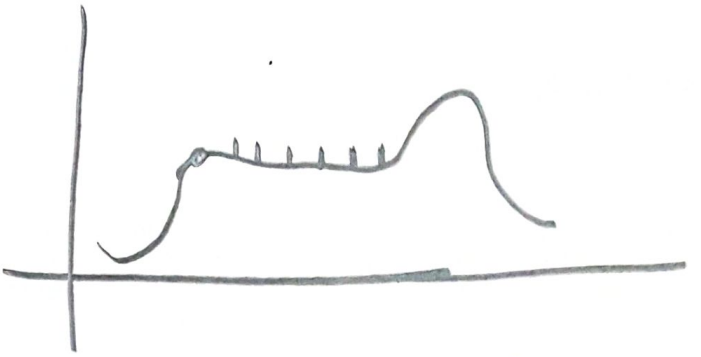
① local maxima: (Local Search)

→ algo stops there.

→ only one direction work

They know only local data not above near hill.

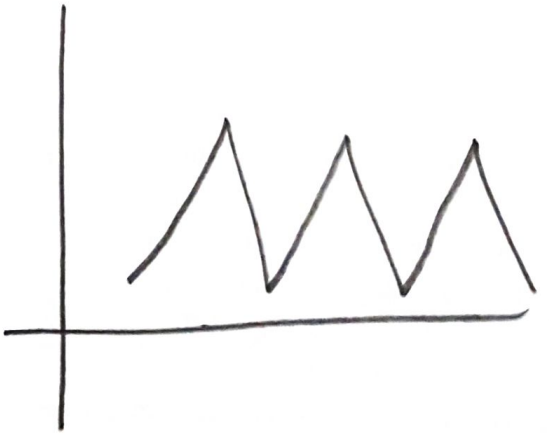
② Plateau:



→ One point to go further plain Surface as it is previous data.

→ algo get same value / soln for some time.

③ Ridge:



→ All successor gets same value.

Prof. Dhaval Khatri

Artificial Intellegence

↑ non-natural

↑ ability to understand, think and learn.

→ It is study of how to make Computers do the things which at present human can do better.

Other

Artificial / Intellegence

↑ "man made" + ↑ thinking power.

★ A.I. Techniques:

① Search :- Solving problem

② use of knowledge. :-

③ Abstraction :- Seperating important features and variations from the enemy unimportant one. It is the hide the details of something.

A.I. Problem characterstics.

① Is the problem decomposable into a set of (nearly) independent smaller or easier subproblem?

② Can solution step be ignored or at least undone if they prove unwise?

- Ignorable (Theram proving)
- Recoverable (8 Puzzle)
- Irrecoverable (chess)

③ Is the problem's universe predictable?

- Certain outcome (8-Puzzle)
- uncertain outcome (bridge, controlling a robot arm,

④ Is a good solution to the problem obvious without comparison to all other possible solutions.

- Absolute:- any path problem.

Consider the problem of answering qu^n based on a database of simple facts, such as following.

Prof. Dhaval Khatri

1. Marcus was a man.
2. Marcus was Pompeiian.
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeiian died when the Volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 1991 A.D

question "Is marcus alive"

Soln 1:.

1. marcus was a man (given)
4. All men are mortal (given)
8. marcus is mortal (new) (from 1 & 4)
3. marcus was born in 40 A.D (given)
7. It is now 1991 A.D (given)
9. marcus age is 1951 years (new from 3,7)
6. No mortal lives longer than 150 years (given)
10. marcus is dead (new, from 8, 6, 9)

Prof. Dhaval Khatri

Soln : 2

7. It is now 1991 A.D (given)
5. All Pomperian died in 79 A.D. (given)
11. All Pomperian are dead now (from 7, 5)
2. Marcus was a Pomperian (given)
12. Marcus is dead (11, 2)

- Relative: (TSP) best path

(5) Is the desired solution a state of the world or a path to a state?

- Natural lang understanding

(6) Is a large amount of knowledge absolutely required to solve the problem, or knowledge important only to constrain the search. (ex chess). (Election News)

(7) Can a computer that is simply given the problem return the solution, or will the solution of the problem require interation bet^n the computer and a person?

Knowledge Representation Issues.

⇒ Representation and mappings.

- Solve the complex problem encountered in A.I, one needs both a large amount of knowledge and some mechanisms for manipulating the knowledge to create Sol^n to new problems.

✱ Two diff^n kind of entities:

1: Facts:- truths in some relevant world. These are the things we want to represent.

2: Representation of facts in some chosen formalism. These are the things we will actually be able to manipulate.

✱ One way to think of structuring these entities is as two levels.

1: The knowledge level:- at which fact. (including each agent's behaviors and current goals) are described.

2. The Symbol level: at which representation of object at the knowledge level are defined in term of symbol that can be manipulated by pgm.

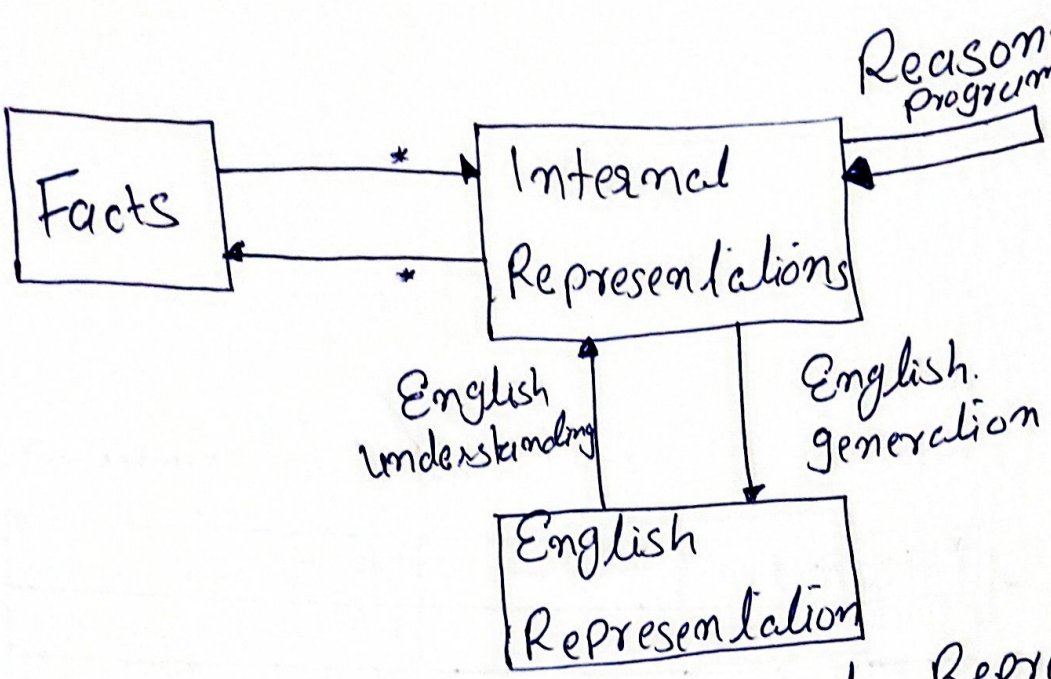


Fig. Mapping bet^n Facts and Representation

→ The forward Representation mapping maps from facts to Representation

→ The backward Representation mapping goes the other way, from Representation to facts.

* Knowledge Representation.

① logical Representation :

- Propositional logic
- First order logic
- other logics.

② Production Rule:-

③ Semantic Networks.

④ Frame Representation

⇒ Semantic Networks:-

These represent knowledge in the form of graphical networks.

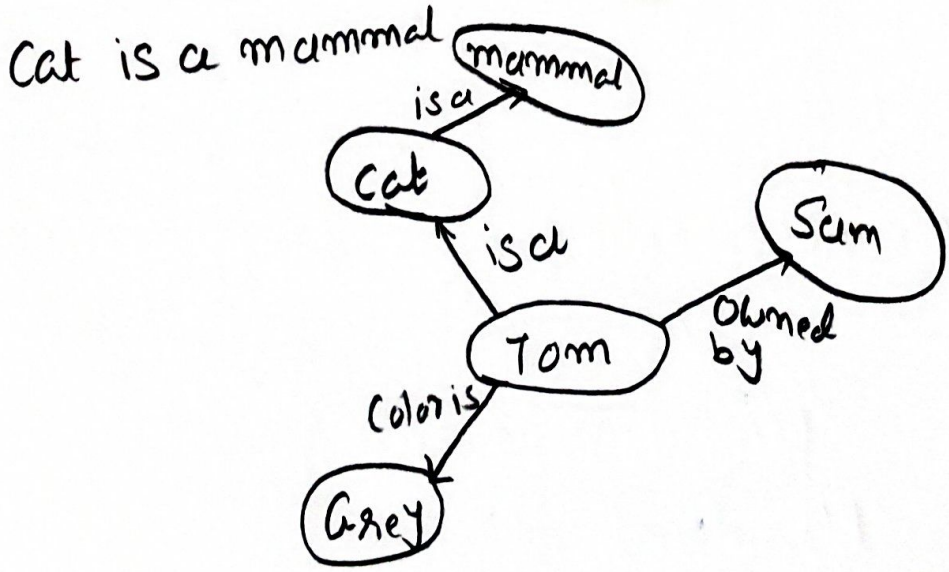
→ IS-A Relation → Kind of-relation

E.g: Tom is a cat.

Tom is grey in color

Tom is mammal

Tom is owned by Sam.



Ex ① Jerry is a cat

② Jerry is a mammal

③ Jerry is owned by Khemak

④ Jerry is brown colored

(5) All mammal are animal

④ Frame Representation:

Frame are Record like structures that consist of a collection of slots and attributes and the corresponding slot values.

⇒ Slots have name and values call facts.

says → keyword.

Ex: Name of person.

Dhaval is an engineer as a profession, and his age is 28, he lives in city London, and the country is England.

(Dhaval)
(Profession (value Engineer))
(Age (value 28))
(city (value London))
(~~state~~ country (value England))

Store diffⁿ data.

Prof. Dhaval Khatri

① Logical Representation:

↳ means drawing a conclusion based on various conditions.

↳ lays down some important communication rules.

↳ Each sentence can be translated into logic using "Syntax" and "Semantics"

★ Syntax:-

- Syntaxes are the rules which decide how we can construct legal sentences in the logic.
- It determines which symbol we can use in knowledge representation
- How to write those symbols.

★ Semantics:-

- Semantics are the rules by which we can interpret the sentence in the logic
- Semantic also involves assigning a meaning to each sentence.

① Propositional logic (PL)

↳ Simplest form of logic where all the statement are made by propositions.

↳ A Propositional is a declarative statement which is either true or false.

↳ It is a technique of knowledge representation in logical and mathmatical form.

Ex The Sun rises from west . ✗

② 5 is a prime numbers. ✓

★ Syntax of Propositional logic:

a. Atomic propositions:

↳ the simple proposition.

↳ It consists of a single proposition symbol.

↳ true or false.

Ex a. $2+2=4$, . ✓

b. "The Sun is cold". ✗

Prof. Dhaval Khatri

(b). Compound Proposition:-

↳ Constructed by combining Simpler or atomalic Propositions,

↳ using Parenthesis and logical connectives.

Ex

1. "It is raining today, and street is wet".

2. "Dhaval is a Engi, and his office is in Ahmedabad".

⇒ Logical Connectives:-

↳ used to connect two simpler Propositions or Representing a sentence logically.

↳ There are mainly five connective.

1. Negation :- A Sentemie as ¬P is called negation of P.

A literal can be either Positive literal or negative literal.

② Conjunction:- A Sentence which has $\wedge$ connective such as, $P \wedge Q$ is called a conjunction.

Ex: Dhaval is intelligent and hardworking.

P = Dhaval is intelligent

Q : Dhaval is hardworking.

$\Rightarrow P \wedge Q$.

③ Disjunction:- A Sentence which has $\vee$ connective, such as, $P \vee Q$ is called disjunction, where P and Q are the propositions.

Ex: "Dhaval is a doctor or Engineer"

$\rightarrow P \vee Q$

④ Implication:- A Sentence such as $P \rightarrow Q$, is called an implication. Implication are also known as if-then rules.

like: If it is raining, then the street is wet.

$P \rightarrow Q$

P :- It is raining Q : Street is wet

⑤ Biconditional:- A Sentence such.

$P \Leftrightarrow Q$ is a Biconditional sentence.

Ex If I am breathing, then I am alive.

P = I am breathing

Q : I am alive

$\Rightarrow P \Leftrightarrow Q$.

Symbol	word
$\wedge$	AND
$\vee$	OR
$\rightarrow$	Implies.
$\Leftrightarrow$	if and only if
$\neg$ or $\sim$	NOT

Ex Propositional logic:

1. It is hot
2. It is humid
3. It is raining
4. If it is humid, then it is hot.
5. If it is hot and humid then it is not raining.

Soln:

A = It is hot

B = It is humid

C = It is raining.

4 = $B \rightarrow C$

5. $(A \wedge B) \rightarrow \neg C$

Ex Propositional logic.

① Today is Not Friday.

$\neg P$

Truth table	
T	F
F	T

② You should study or watch TV at a time.

P Q

V - means plus.

P V Q

Truth table:		
T	T	T
T	F	T
F	T	T
F	F	F

Prof. Dhaval Khatri

③ Please like my video and
Subscribe my channel.
p
q.

$\wedge$ = multiplication

$\Rightarrow p \wedge q$

Truth table		
T	T	T
T	F	F
F	T	F
F	F	F

④ if there is rain then the
roads are wet.
p
q.

$\Rightarrow p \rightarrow q$

Truth table		
T	T	T
T	F	F
F	T	T
F	F	T

(5) I will go to mall if I have to do shopping.

⇒ $P \Leftrightarrow Q$

Truth table		
T	T	T
T	F	F
F	T	F
F	F	T

Ex You can access the internet from campus only if you are CE student or you are not freshmen.

$$P \rightarrow (Q \vee \neg R).$$

⇒ limitation:

- We cannot represent relation, like ALL, Some or none
  - Ex Some apple are sweet.
- We cannot describe statement in terms of their properties or logical relationships.

② First-order logic:

↳ It is an extension to Propositional logic.

↳ also known as Predicate logic or First order predicate logic

↳ First-order logic (like natural lang) does not only assume that the world contains facts like proposition logic but also assume the following things:

- Objects:- A, B, People, numbers, colors, wars, ...
- Relation: It can be unary Relation such as: Red, round, is adjacent. or n-ary relation such as - the sister of, brother of, has color, comes between
- Function:- Father of, best friend, end of.

Atomic Sentence:

→ most basic sentence of first order logic

→ These sentence are formed from a predicate Symbol followed by a parenthesis with a seq^{n}g term.

⇒ ~~Complex Sentences~~:

Atomic Sentence as predicate

(term1, term2 ... termn)

Ex A and B are brothers.

⇒ Brothers(A, B)

Chinky is a cat.

⇒ cat(chinky).

Complex Sentence:

↳ Two part

① Subject:- It is the main part of the statement

② Predicate:- It can be defined as a relation, which binds two atoms together in a statement.

→ Quantifier in first-order logic

↳ the symbol that permit to determine or identify the range and scope of the variable in the logical expression.

There are two types.

① Universal Quantifier
(for all, everyone, everything)

② Existential quantifier
(for some, at least one).

① Universal Quantifier :-

It is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

→ Symbol ∀, which resembles an inverted A.

Note: We use implication "→"

Ex: If x is a variable, then ∀x is read as.

- for all x
- for each x
- for every x

Ex All boys like Cricket

$\forall x : boy(x) \rightarrow like(x, Cricket)$

Ex All men drink coffee

② Existential Quantifier:-

↳ which express that the statement within its scope is true for at least one instance of something.

→ Symbol $\exists$, which resembles as inverted E,

Note: We always use AND or $\wedge$ Symbol (Conjunctions)

If x is a variable, then existential quer will be $\exists x$ or $\exists(x)$.

- There exists a 'x'
- For some 'x'
- For at least 'x'

Ex: Some Boys like cricket.

$\exists x : boys(x) \wedge like(x, cricket)$

Ex: Some boys are intelligent

$\exists x : boys(x) \wedge intelligent(x)$

Solve example:

① All birds fly.

② Every man respects his parent

③ Some boys play Cricket

④ Not all Student like both mathematics and Science.

⑤ only one student failed in mathematics.

★ Resolution:
→ to do or not to do

Steps:
negate the statement to be proved
convert given facts into FOL
Convert FOL into CNF
Draw Resolution Graph.

$\vee$ = OR
$\wedge$ = and

Step 3: Rules to convert FOL into CNF

⇒ Eliminate '$\rightarrow$' & '$\leftrightarrow$'

$a \rightarrow b \quad \therefore \neg a \vee b$

$a \leftrightarrow b \quad \therefore a \rightarrow b \wedge b \rightarrow a$

⇒ move '$\neg$' inward

- $\neg(\forall x\, p) = \exists_x \neg p$
- $\neg(\exists x\, p) = \forall x \neg p$
- $\neg(a \vee b) = \neg a \wedge \neg b$
- $\neg(a \wedge b) = \neg a \vee \neg b$
- $\neg\neg a = a$

⇒ Rename variable (not use same variable)

⇒ Replace Existential quantifier by skolem constant.

Ex $\exists x\, Rich(x) = Rich(G_1)$ → any capital Alphabet

⇒ Drop universal quantifier

Ex $\forall x\, (Smile(x)) = Smile(x)$.

Ex

1. All People who are graduating are happy.
2. All happy People Smile
3. Some one is Graduating

Step: 1:- Convert to FoL

2: Convert FoL to CNF

3: Prove that "Is Someone Smiling?" using Resolution

4. Draw Resolution tree

Sol^n: Step 1:

All People who are graduating are happy:

- People:- x

$\forall x: (Graduating(x) \rightarrow happy(x))$

All happy People Smile.

$\forall x: (happ(x) \rightarrow Smile(x))$

Someone is Graduating.

$\exists x: Graduating(x)$

We need to prove that is

Someone Smiling?

$\neg \exists x : Smile(x)$

Step 2: Convert FOL to CNF (Eliminate Implications) $\alpha \rightarrow \beta = \neg \forall \vee \beta$

$\rightarrow \forall x : (Graduating(x) \rightarrow Happy(x))$

$\forall x : [\neg Graduating(x) \vee Happy(x)]$

$\forall x : (Happy(x) \rightarrow Smile(x))$

$\forall x : [\neg Happy(x) \vee Smile(x)]$

$\exists x : Graduating(x)$

$\exists x : Graduating(x)$

$\exists x : Smile(x)$

$\neg \exists x : Smile(x)$

Step 3: Standarize variables

$\forall x : [\neg Graduating(x) \vee Happy(x)]$

$\forall y : [\neg Happy(y) \vee Smile(y)]$

$\exists z : Graduating(z)$

$\neg \exists w : Smile(w)$

Step 4: Move Negation Inwards:

$\forall x [\neg \text{Graduating}(x) \vee \text{Happy}(x)]$

$\forall x [\neg \text{Happ}(y) \vee \text{Smile}(y)]$

$\exists z \ \text{Graduating}(z)$

$\forall w \neg \text{Smile}(w)$

Step 5: Skolemization: (Remove $\exists x$)

$\forall x [\neg \text{Graduating}(x) \vee \text{Happy}(x)]$

$\forall x [\neg \text{Happy}(y) \vee \text{Smile}(y)]$

$\text{Graduating}(A)$

$\forall w \neg \text{smile}(w)$

Step 6: Drop Universal Quantifier

$\neg \text{Graduating}(x) \vee \text{Happy}(x)$

$\neg \text{Happy}(y) \vee \text{Smile}(y)$

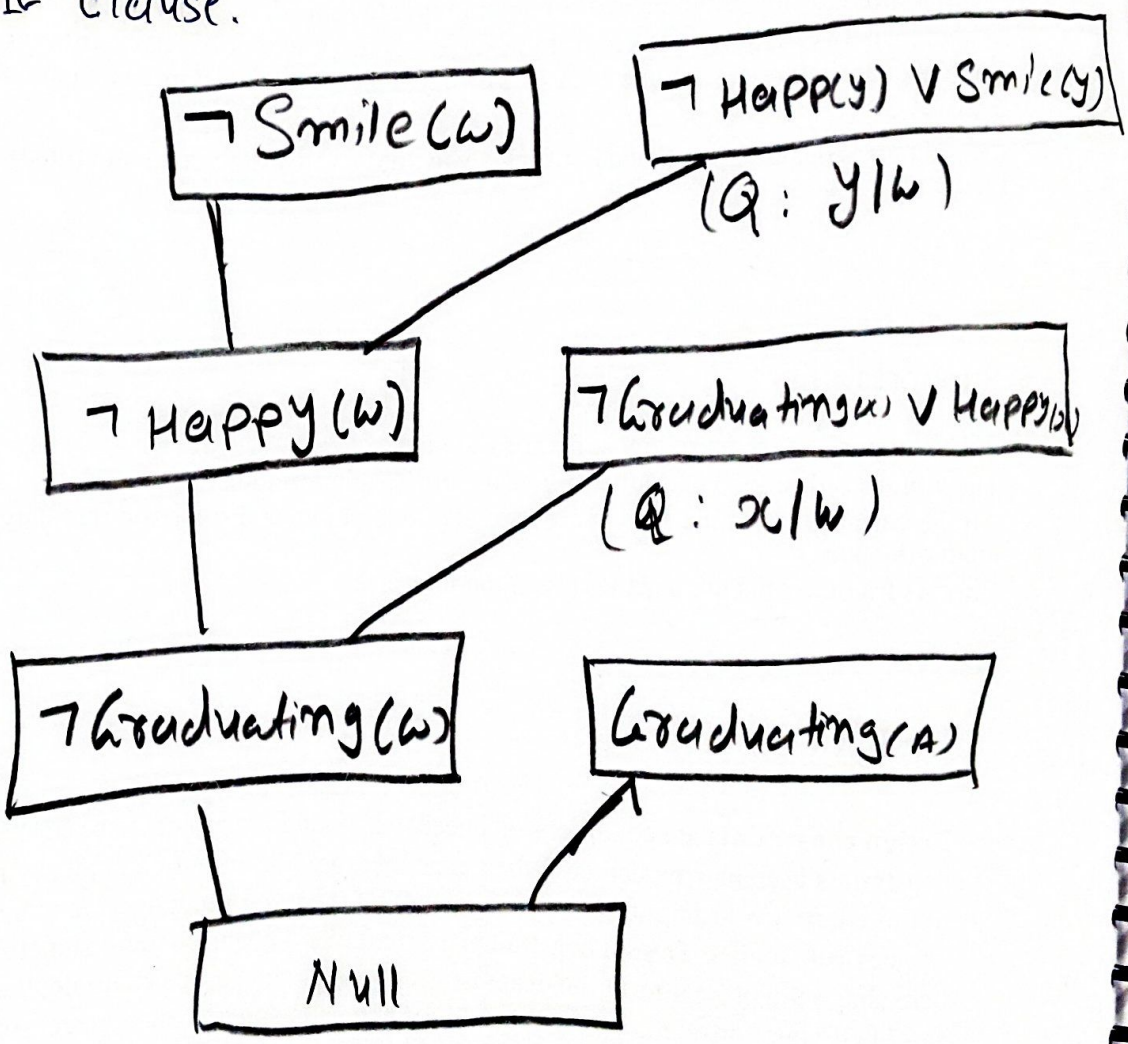
$\text{Graduating}(A)$

$\neg \text{Smile}(w)$.

now the Sentence are in CNF.

Resolution Tree:

- If Fact 'F' is to be proved that it Start with '¬F'.
- It contradicts all the other rules in Knowledge Base
- The Process Stop when it returns. NULL clause.



Hence Someone is Smiling.

✱ Resolution:

↳ to do or to not do

Eg

a) Ravi likes all kind of food

b) Apple and chicken are food

c) Anything anyone eats and is not killed is food.

D) Dhaval eats penuts and still alive

e) Rita eats that Dhaval eats.

Prove that: Ravi like penuts.

Soln

Step 1: negate the statement to be proved

$\neg$ likes (Ravi, Penuts)

Step 2: Convert into FOL

$\forall x$: food (x) $\rightarrow$ likes (Ravi, x)

i) food (apple) ii) food (chicken)

$\forall x \forall y$: eats (x, y) $\wedge$ $\neg$ killed (x) $\rightarrow$ food (y)

eats (Dhaval, Penuts) $\wedge$ alive (Dhaval)

$\forall x$: $\neg$ killed (x) $\rightarrow$ alive (x)

$\forall x$: alive (x) $\rightarrow$ $\neg$ killed (x)

Step 3: Convert FOL into CNF

$\forall x : food(x) \rightarrow likes(Ravi, x)$

$\neg food(x) \vee likes(Ravi, x)$

$\forall x \forall y : eats(x, y) \wedge \neg killed(x) \rightarrow food(y).$

$\neg [eats(x, y) \wedge \neg killed(x)] \vee food(y)$

$\neg eats(x, y) \vee killed(x) \vee food(y)$

$eats(Dhaval, Peanuts) \wedge alive(Dhaval)$

$\forall x : \neg killed(x) \rightarrow alive(x)$

$\neg\neg killed(x) \vee alive(x)$

$killed(x) \vee alive(x).$

$\forall x : alive(x) \rightarrow \neg killed(x).$

$\neg alive(x) \vee \neg killed(x)$

Resolution Graph