State space search

Problem Spaces and Search

Building a system to solve a problem requires the following 4 steps:

- Define the problem precisely including detailed specifications and what constitutes an acceptable solution;
- Analyse the problem thoroughly for some features may have a dominant effect on the chosen method of solution;
- Isolate and represent the background knowledge needed in the solution of the problem;
- Choose the best problem solving techniques in the solution.

Defining the Problem as state Search

Problems dealt with in artificial intelligence generally use a common term called 'state'. A state represents a status of the solution at a given step of the problem solving procedure. The solution of a problem, thus, is a collection of the problem states. The problem solving procedure applies an operator to a state to get the next state. Then it applies another operator to the resulting state to derive a new state. The process of applying an operator to a state and its subsequent transition to the next state, thus, is continued until the goal (desired) state is derived. Such a method of solving a problem is generally referred to as state space approach.

For example, in order to solve the problem play a game, which is restricted to two person table or board games, we require the rules of the game and the targets for winning as well as a means of representing positions in the game. The opening position can be defined as the initial state and a winning position as a goal state, there can be more than one. Legal moves allow for transfer from initial state to other states leading to the goal state. However the rules are far too copious in most games especially chess where they exceed the number of particles in the universe 10. Thus the rules cannot in general be supplied accurately and computer programs cannot easily handle them. The storage also presents another problem but searching can be achieved by hashing. The number of rules that are used must be minimised and the set can be produced by expressing each rule in as general a form as possible. The representation of games in this way leads to a state space representation. This representation allows for the formal definition of a problem which necessitates the movement from a set of initial positions to one of a set of target positions. It means that the solution involves using known techniques and a systematic search. This is quite a common method in AI.

Formal description of a problem

- Define a state space that contains all possible configurations of the relevant objects, without enumerating all the states in it. A *state space* represents a problem in terms of *states* and *operators* that change states
- Define some of these states as possible initial states;
- Specify one or more as acceptable solutions, these are goal states;
- Specify a set of rules as the possible actions allowed. This involves thinking about the generality of the rules, the assumptions made in the informal presentation and how much work can be anticipated by inclusion in the rules.

ALGORITHM OF PROBLEM SOLVING

Any one algorithm for a particular problem is not applicable over all types of problems in a variety ofsituations. So there should be a general problem solving algorithm, which may work for different strategies of different problems. Algorithm:

Step 1: Analyze the problem to get the starting state and goal state.

Step 2: Find out the data about the starting state goal state

Step 3: Find out the production rules from initial database for proceeding the problem to goal state.

Step 4: Select some rules from the set of rules that can be applied to data.

Step 5: Apply those rules to the initial state and proceed to get the next state.

Step 6: Determine some new generated states after applying the rules. Accordingly make them as current state.

Step 7: Finally, achieve some information about the goal state from the recently used current state and get thegoal state.

Step 8: Exit.

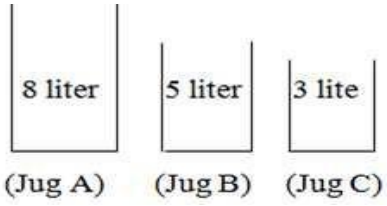
VARIOUS TYPES OF PROBLEMS AND THEIR SOLUTIONS

Water Jug Problem

Definition: Some jugs are given which should have non-calibrated properties. At least any one of the jugs should have filled with water. Then the process through which we can divide the whole water into different jugs according to the question can be called as water jug problem.

Procedure: Suppose that you are given 3 jugs A,B,C with capacities 8,5 and 3 liters respectively but are not calibrated(i.e. no measuring mark will be there). Jug A is filled with 8 liters of water. By a series of pouring backand forth among the 3 jugs, divide the 8 liters into 2 equal parts i.e. 4 liters in jug A and 4 liters in jug B.How?

In this problem, the start state is that the jug A will contain 8 liters water whereas jug B and jug C will be empty. The production rules involve filling a jug with some amount of water, taking from the jug A. The search will be finding the sequence of production rules which transform the initial state to final state. Thestate space for this problem can be described by set of ordered pairs of three variables (A, B, C) wherevariable A represents the 8 liter jug, variable B represents the 5 liter and variable C represents the 3 liters jug respectively.

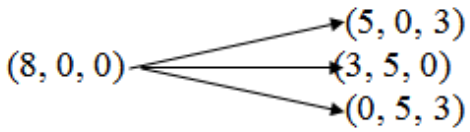


The production rules are formulated as follows:

Step 1: In this step, the initial state will be (8, 0, 0) as the jug B and jug C will be empty. So the water of jug Acan be poured like: (5, 0, 3) means 3 liters to jug C and 5 liters will remain in jug A.

(3, 5, 0) means 5 liters to jug B and 3 liters will be in jug A.

(0, 5, 3) means 5 liters to jug B and 3 liters to jug C and jug C and jug A will be empty.



Step 2: In this step, start with the first current state of step-1 i.e. (5, 0, 3). This state can only be implemented bypouring the 3 liters water of jug C into jug B. so the state will be (5, 3, 0). Next, come to the second

current state of step-1 i.e. (3, 5, 0). This state can be implemented by only pouring the 5 liters water of jugB into jug C. So the remaining water in jug B will be 2 liters. So the state will be (3, 2, 3). Finally come tothe third current state of step-1 i.e. (0, 5, 3). But from this state no more state can be implemented because after implementing we may get (5, 0, 3) or (3, 5, 0) or (8, 0, 0) which are repeated state. Hence thesestates are not considerably again for going towards goal.

So the state will be like:

(5, 0, 3) → (5, 3, 0)

(3, 5, 0) → (3, 2, 3)

(0, 5, 3) →X

Step 3: In this step, start with the first current state of step-2 i.e. (5, 3, 0) and proceed likewise the above steps.

(5, 3, 0) → (2, 3, 3)

(3, 2, 3)→(6, 2, 0)

Step 4: In this step, start with the first current state of step-3 i.e. (2, 3, 3) and proceed.

(2, 3, 3) → (2, 5, 1)

(6, 2, 0) → (7, 0, 1)

Step 5:

(2, 5, 1) → (7, 0, 1)

(6, 0, 2) → (1, 5, 2)

Step6:

(7, 0, 1) → (7, 1, 0)

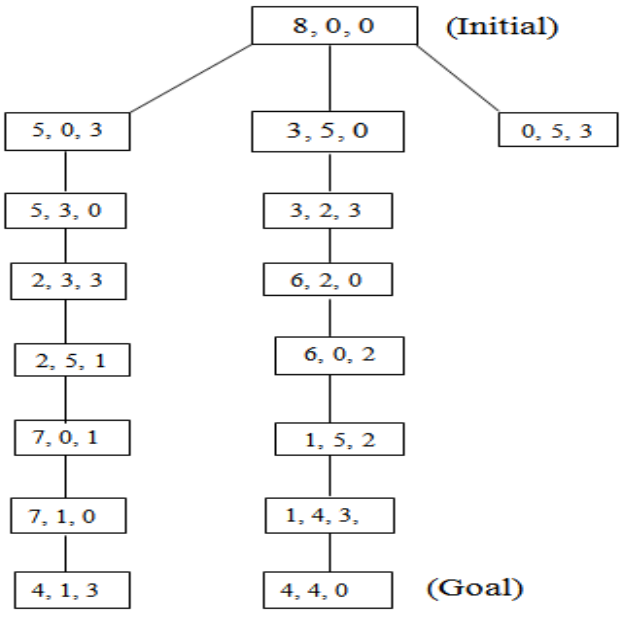
(1, 4, 3) → (1, 4, 3)

Step7:

(7, 1, 0) → (4, 1, 3)

(1, 4, 3) → (4, 4, 0) (Goal)

So finally the state will be (4, 4, 0) that means jug A and jug B contains 4 liters of water each which is our goal state. The tree of the water jug problem can be like:



Missionaries and Carnivals Problem

Definition: In Missionaries and Carnivals Problem, initially there are some missionaries and some carnivals will be at a side of a river. They want to cross the river. But there is only one boat available to cross the river. Thecapacity of the boat is 2 and no one missionary or no Carnivals can cross the river together. So for solvingthe problem and to find out the solution on different states is called the Missionaries and Carnival Problem.

Procedure:

Let us take an example. Initially a boatman, Grass, Tiger and Goat is present at the left bank of the river and want to cross it. The only boat available is one capable of carrying 2 objects of portions at a time. The condition of safe crossing is that at no time the tiger present with goat, the goat present with the grass at the either side of the river. How they will cross the river?

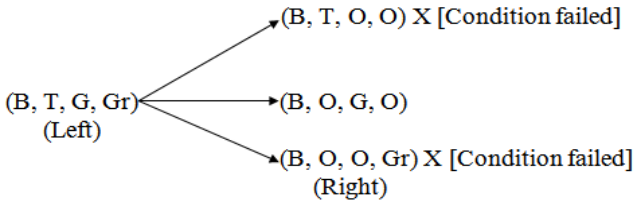
The objective of the solution is to find the sequence of their transfer from one bank of the river to theother using the boat sailing through the river satisfying these constraints.

Let us use different representations for each of the missionaries and Carnivals as follows.

B: Boat T: Tiger

G: Goat Gr: Grass

Step 1: According to the question, this step will be (B, T, G, Gr) as all the Missionaries and the Carnivals are at one side of the bank of the river. Different states from this state can be implemented as



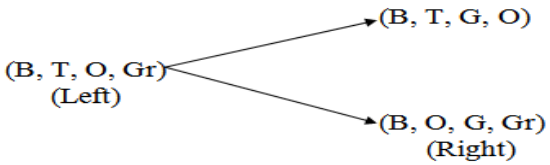
The states (B, T, O, O) and (B, O, O, Gr) will not be countable because at a time the Boatman and theTiger or the Boatman and grass cannot go. (According to the question).

Step 2: Now consider the current state of step-1 i.e. the state (B, O, G, O). The state is the right side of the river. So on the left side the state may be (B, T, O, Gr)

i.e.(B, O, G, O) → (B, T, O, Gr)

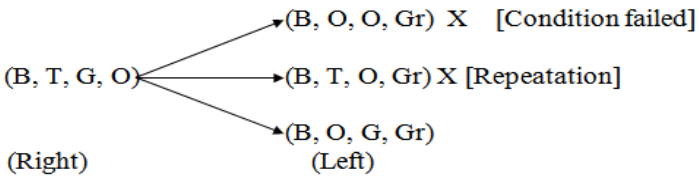
(Right) (Left)

Step 3: Now proceed according to the left and right sides of the river such that the condition of the problem mustbe satisfied.

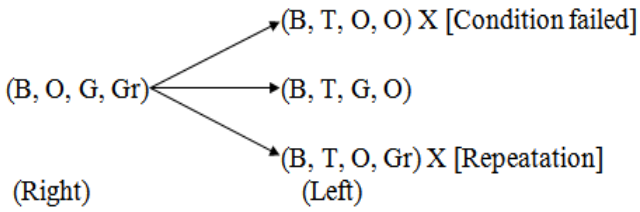


Step 4:

First, consider the first current state on the right side of step 3 i.e.

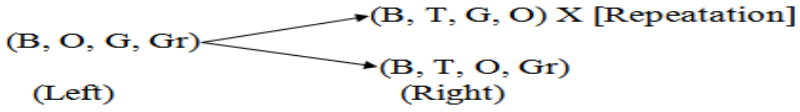


Now consider the second current state on the right side of step-3 i.e.

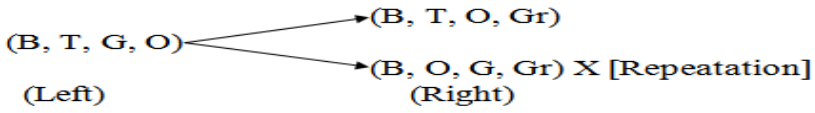


Step 5:

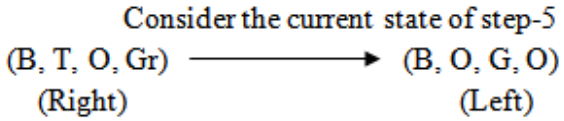
Now first consider the first current state of step 4 i.e.



Now consider the 2nd state of step-4 i.e.



Step 6:



Step 7:

From ftep-6, the state will be

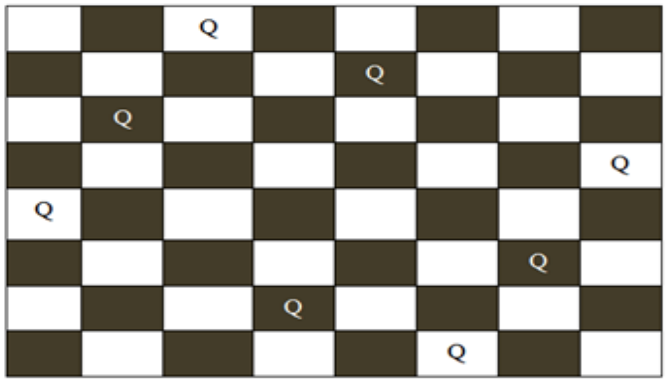
(B, O, G, O) ⟶ (B, T, G, Gr)

(Left) (Right)

Hence the final state will be (B, T, G, Gr) which are on the right side of the river.

8- Queen Problem

Definition: "We have 8 queens and an 8x8 Chess board having alternate black and white squares. The queens are placed on the chessboard. Any queen can attack any other queen placed on same row, or column ordiagonal. We have to find the proper placement of queens on the Chess board in such a way that no queenattacks other queen".



Procedure: In figure, the possible board configuration for 8-queen problem has been shown. The board has alternative black and white positions on it. The different positions on the board hold the queens. Theproduction rule for this game is you cannot put the same queens in a same row or same column or in samediagonal. After shifting a single queen from its position on the board, the user have to shift other queensaccording to the production rule. Starting from the first row on the board the queen of their correspondingrow and column are to be moved from their original positions to another position. Finally the player hasto be ensured that no rows or columns or diagonals of on the table is same.

8- Puzzle Problem

Definition: "It has set off a 3x3 board having 9 block spaces out of which 8 blocks having tiles bearing number from1 to 8. One space is left blank. The tile adjacent to blank space can move into it. We have to arrange the tiles in a sequence for getting the goal state".

Procedure: The 8-puzzle problem belongs to the category of "sliding block puzzle" type of problem. The 8-puzzle isa square tray in which eight square tiles are placed. The remaining ninth square is uncovered. Each tile inthe tray has a number on it. A tile that is adjacent to blank space can be slide into that space. The gameconsists of a starting position and a specified goal position. The goal is to transform the starting positioninto the goal position

by sliding the tiles around. The control mechanisms for an 8-puzzle solver mustkeep track of the order in which operations are performed, so that the operations can be undone one at atime if necessary. The objective of the puzzles is to find a sequence of tile movements that leads from astarting configuration to a goal configuration such as two situations given below.

The operations are thepermissible moves up, down, left, right. Here at each step of the problem a function f(x) will be defined

1	2	3
4	5	6
	7	8

1	2	3
4	5	6
7	8	-

which is the combination of g(x) and h(x).i.e. $F(x)=g(x) + h(x)$

Where

g (x): how many steps in the problem you have already done or the current state from the initial state.

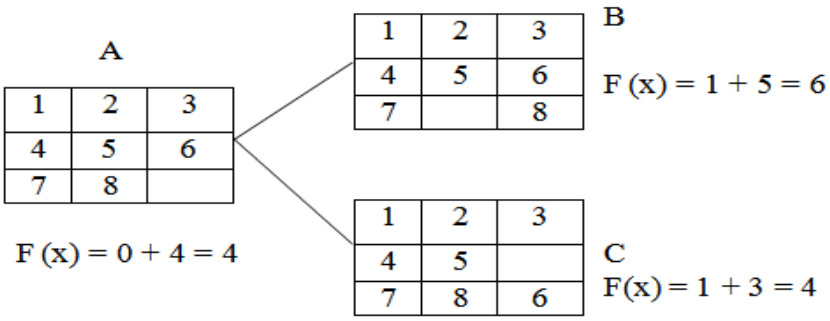
h (x): Number of ways through which you can reach at the goal state from the current state or h (x)is the heuristic estimator that compares the current state with the goal state note down how many states are displaced from the initial or the current state.

After calculating the f (x) value at each step finally take the smallest f (x) value at every step and choose that as the next current state to get the goal state. Let us take an example.

Figure (Initial State) (Goal State)

1	2	3
4	5	6
7	8	

1	2	3
4	8	5
	7	6

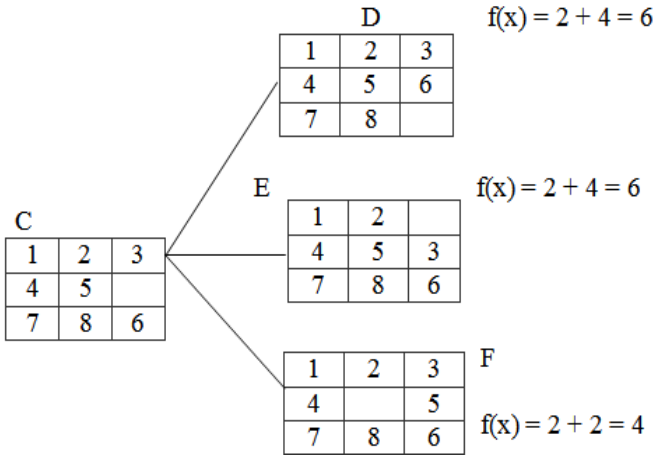


Step1: f (x) is the step required to reach at the goal state from the initial state. So in the tray either 6 or 8 canchange their portions to fill the empty position. So there will be two possible current states namely B and

C. The f (x) value of B is 6 and that of C is 4. As 4 is the minimum, so take C as the current state to the next state.

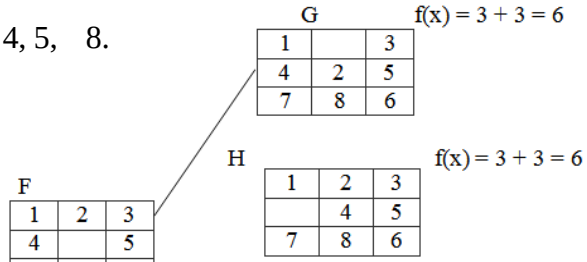
Step 2:

In this step, from the tray C three states can be drawn. The empty position will contain either 5 or 3 or 6.So for three different values three different states can be obtained. Then calculate each of their f (x) and take the minimum one.



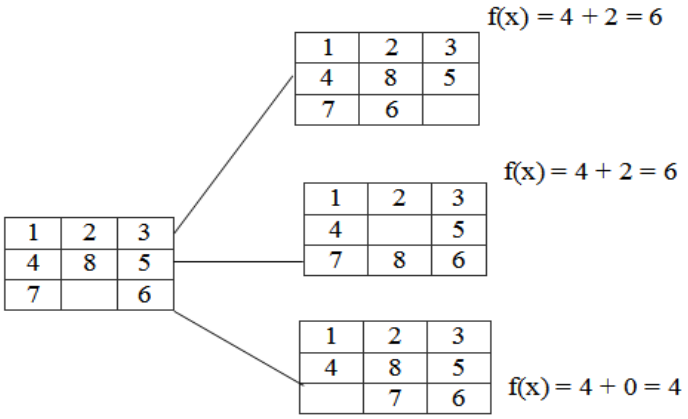
Here the state F has the minimum value i.e. 4 and hence take that as the next current state.

Step 3: The tray F can have 4 different states as the empty positions can be filled with b4 values i.e.2, 4, 5, 8.



Step 4:

In the step-3 the tray I has the smallest f (n) value. The tray I can be implemented in 3 different states because the empty position can be filled by the members like 7, 8, 6.



Hence, we reached at the goal state after few changes of tiles in different positions of the trays.

Monkey Banana Problem

Definition: "A monkey is in a room. A bunch of bananas is hanging from the ceiling. The monkey cannot reach thebananas directly. There is a box in the corner of the room. How can the monkey get the bananas?"

Procedure: The solution of the problem is of course that the monkey must push the box under the bananas, then standon the box and grab the bananas. But the solution procedure requires a lot of planning algorithms. The purpose of the problem is to raise the question: Are monkeys intelligent? Both humans and monkeys havethe ability to use mental maps to remember things like where to go to find shelter or how to avoid danger.They can also remember where to go to gather food and water, as well as how to communicate with each other. Monkeys have the ability not only to remember how to hunt and gather but they also have theability to learn new things, as is the case with the monkey and the bananas. Even though that monkey maynever have entered that room before or had only a box for a tool to gather the food available, that monkeycan learn that it needs to move the box across the floor, position it below the bananas and climb the box toreach for them.

Initially, the monkey is at location 'A', the banana is at location 'B' and the box is at location 'C'. Themonkey and box have height "low"; but if the monkey climbs onto the box will have height "High", thesame as the bananas.

The action available to the monkey include:

"GO" from one place to another.

"PUSH" an object from one place to another.

"Climb" onto an object.

"Grasp" an object.

Grasping results in holding the object if the monkey and the object are in the same place at the same height. So the solution to the planning problem may be of following

- GO (A,C)
- PUSH (Box, C, B, Low)
- Climb Up (Box , B)
- Grasp (banana, B, High)
- Climb down (Box)
- Push (Box, B, C, Low)

Tower of Hanoi Problem

Definition: "We are given a tower of eight discs (initially) four in the applet below, initially stacked in increasing sizeon one of three pegs. The objective is to transfer the entire tower to one of the other pegs (the right mostone in the applet below), moving only one disc at a time and never a larger one onto a smaller".

Procedure: The objective of the puzzle is to move the entire stack to another rod, obeying the following rules.

- Only one disc can be moved at a time.
- Each move consist of taking the upper disc from one of the rods and sliding it onto another rod,on top of the other discs that may already be present on that rod.
- No disc may be placed on the top of a smaller disk.

Thefollowing solution is a very simple method to solve the tower of Hanoi problem.

- Alternative moves between the smallest piece and a non-smallest piece. When moving thesmallest piece, always move it in the same direction (to the right if starting number of pieces iseven, to the left if starting number of pieces is odd).
- If there is no tower in the chosen direction, move the pieces to the opposite end, but then continue to move in the correct direction, for example if you started with three pieces, you would move the smallest piece to the opposite end,

then continue in the left direction after that.

- When the turn is to move the non-smallest piece, there is only one legal move.

Finally, the user willreach at the goal. A recursive solution for tower of Hanoi problem is as follows.

A key to solving this problem is to recognize that it can be solve by breaking the problem down into thecollection of smaller problems and further breaking those problems down into even smaller problems until a solution is reached. The following procedure demonstrates this approach.

- Label the pegs A, B, C - these levels may move at different steps.
- Let n be the total number of disks.
- Number of disks from 1 (smallest, topmost) to n (largest, bottommost). To move n disks from peg A to peg C.

a) Move n-1 disks from A to B. This leaves disk #n alone on peg A.

b) Move disk #n from A to C.

c) Move n-1 disks from B to C so they sit on disk #n.

To carry out steps a and c, apply the same algorithm again for n-1. The entire procedure is a finite numberof steps, since at most point the algorithm will be required for n = 1. This step, moving a single disc frompeg A to peg B, is trivial.

Cryptarithmatic Problem

Definition: "It is an arithmetic problem which is represented in letters. It involves the decoding of digit representedby a character. It is in the form of some arithmetic equation where digits are distinctly represented bysome characters. The problem requires finding of the digit represented by each character. Assign adecimal digit to each of the letters in such a way that the answer to the problem is correct. If the same letter occurs more than once, it must be assigned the same digit each time. No two different letters may be assigned the same digit".

Procedure: Cryptarithmatic problem is an interesting constraint satisfaction problem for which different algorithmshave been developed. The different constraints of defining a cryptarithmatic problem are as follows.

1) Each letter or symbol is represented by only one and a unique digit throughout the

problem.

2) When the digits replace letters or symbols, the resultant arithmetical operation must be correct.

Consider that, the base of the number is 10. Then there must be at most 10 unique symbols or letters inthe problem. Otherwise, it would not possible to assign a unique digit to unique letter or symbol in theproblem. To be semantically meaningful, a number must not begin with a 0. So, the letters at thebeginning of each number should not correspond to 0. Also one can solve the problem by a simple blindsearch. But a rule based searching technique can provide the solution in minimum time.

Now, let us solve a simple cryptarithmatic puzzle given below.

```
  S END
+ MORE
-------
M ONEY
```

Step 1: In the given problem, M must be 1. You can visualize that, this is an addition problem. The sum of twofour digit numbers cannot be more than 10,000. Also M cannot be zero according to the rules, since it is the first letter. So now you have the problem like

```
  SEN D
+1 0 R E
---------
 1 0 N Y
```

Step 2: Now in the column s10, s+1 ≥ 10. S must be 8 because there is a 1 carried over from the column EON or O must be 0 (if s=8 and there is a 1 carried or s = 9 and there is no 1 carried) or 1 (if s=9 and there is a 1 carried). But 1 is already taken, so O must be 0.

```
  S END
+ 1 0 RE
----------
1 0 NEY
```

Step 3: There cannot be carry from column EON because any digit +0 < 10, unless there is a carry from thecolumn NRE, and E=9; But this cannot be the case because then N would be 0 and 0 is already taken. SoE < 9 and there is no carry from this column. Therefore S=9 because 9+1=10.

Step 4:

```
  9 E N D
+ 1 O R E
-----------
1 O N E Y
```

In the column EON, E cannot be equal to N. So there must be carry from the column NRE; E+1=N. We now look at the column NRE, we know that E+1=N. Since we know that carry from this column,N+R=1E (if there is no carry from the column DEY) or N+R+1=1E (if there is a carry from the column DEY).

Let us see both the cases:

No carry: $N + R = 10 + (N - 1) = N + 9$

$R = 9$

But 9 is already taken, so this will not work

Carry: $N + R + 1 = 9$

$R = 9 - 1 = 8$. This must be solution for R

```
  9 E N D
+ 1 0 8 E
-----------
1 O N E Y
```

Step 5:

Now just think what are the digits we have left? They are 7, 6, 5, 4, 3 and 2. We know there must be a carry from the column DEY. So D + E > 10.N = E + 1, So E cannot be 7 because then N would 8

```
  9 5 N D
+ 1 0 8 5
-----------
1 0 N 5 Y
```

which is already taken. D is almost 7, so E cannot be 2 because then D + E < 10 and E cannot be 3 because then D + E = 10 and Y = 0, but 0 is already taken. Also E cannot be 4 because if D > 6,D + E < 10 and if D = 6 or D = 7 then Y = 0 or Y = 1, which are both taken. So E is 5 or 6. If E = 6, then D = 7 and Y = 3. So this part will work but look the column N8E. Point that there is a carry from the column D5Y.N + 8 + 1 = 16(As there is a carry from this column). But then N=7 and 7 is taken by D therefore E=5.

Step 6: Now we have gotten this important digit, it gets much simpler from here. N+8+1=15, N=6

```
  9 5 6 D
+ 1 0 8 5
-----------
1 0 6 5 Y
```

Step 7:

The digits left are 7, 4, 3 and 2. We know there is carry from the column D5Y, so the only pair that worksis D=7 and Y= 2.

```
  9 5 6 7
+ 1 0 8 5
-----------
1 0 6 5 2
```

Which is final solution of the problem.

Production systems

Since search forms the core of many intelligent processes, it is useful to structure AI programs in a way that facilitates describing and performing the search process. Production systems provide such structures. It is a model of computation that can be applied to implement search algorithms andmodel human problem solving. Such problem solving knowledge can be packed up in the form of productions. A production is a rule consisting of a situation recognition part and an actionpart. A production is a situation-action pair in which the left side is a list of things to watch for and theright side is a list of things to do so. A production system consists of following components.

(a) A set of production rules, which are of the form A→B. Each rule consists of left hand side constituent that represent the current problem state and a right hand side that represent an output state. A rule is applicable if its left hand side matches with the current problem state.

(b) A database, which contains all the appropriate information for the particular task. Some part of the database may be permanent while some part of this may pertain only to the solution of thecurrent problem.

(c) A control strategy that specifies order in which the rules will be compared to the

database of rulesand a way of resolving the conflicts that arise when several rules match simultaneously.

(d) A rule applier, which checks the capability of rule by matching the content state with the left hand side of the rule and finds the appropriate rule from database of rules.

The important roles played by production systems include a powerful knowledge representation scheme. A production system not only represents knowledge but also action. It acts as a bridge between AI andexpert systems. Production system provides a language in which the representation of expert knowledgeis very natural. We can represent knowledge in a production system as a set of rules of the form

If (condition) THEN (condition)

along with a control system and a database. The control system serves as a rule interpreter and sequencer. The database acts as a context buffer, which records the conditions evaluated by the rules and informationon which the rules act.

Control Strategies

We need to choose the appropriate control structure for the production system so that the search can be as efficient as possible. It deals with how to decide which rule to apply next during the process of searching for a solution to a problem. This question arise since often more than one rule will have its left side match the current state. It is clear that such decisions made will have a crucial impact on how quickly a problem is finally solved.

A good control strategy should have the following requirement: The **first** requirement is that it causes motion. In a game playing program the pieces move on the board and in the water jug problem wateris used to fill jugs. Suppose in water jug problem we implement a simple strategy of starting each time at the top of the list of rules and choosing the first applicable one, we would never solve the problem. We would continue indefinitely filling 4 gallon jug with water. Control strategies that do not cause motion will never lead to a solution.

The **second** requirement is that it is systematic, this is a clear requirement for it would not be sensible to fill a jug and empty it repeatedly nor in a game would it be advisable to move a piece round and round the board in a cyclic way. Simple control strategy for the water jug problem is: On each cycle, choose at random from among the applicable rules. This strategy is better than the first. It causes motion. It will lead to a solution eventually.

But we are likely to arrive at the same state several times during the process and to use many more steps than are necessary. Because the control strategy is not systematic, we may explore a particular useless sequence of operators several times before we finally find a solution. Hence there is a need for systematic control strategy.

Problem Characteristics

In order to choose the most appropriate method (or combination of methods) for a particular problem, it is necessary to analyze the problem along several keydimensions. Some of the main key features of a problem are given below.

- Is the problem decomposable into set of sub problems?
- Can the solution step be ignored or undone?
- Is the problem universally predictable?
- Is a good solution to the problem obvious without comparison to all the possible solutions?
- Is the desire solution a state of world or a path to a state?
- Is a large amount of knowledge absolutely required to solve the problem?
- Will the solution of the problem required interaction between the computer and the person?

The above characteristics of a problem are called as 7-problem characteristics under which the solutionmust take place.

1. Is the problem decomposable?

A very large and composite problem can be easily solved if it can be broken into smaller problems and recursion could be used. Suppose we want to solve.

Ex:- $\int x2 + 3x+\sin2x \cos 2x \, dx$

This can be done by breaking it into three smaller problems and solving each by applying specific rules. Adding the results the complete solution is obtained.

2. Can solution steps be ignored or undone?

Problem fall under three classes ignorable, recoverable and irrecoverable. This classification is with reference to the steps of the solution to a problem. Consider theorem proving. We may later find that it is of no help. We can still proceed further, since nothing is lost by this redundant step. This is an example of ignorable solutions steps.

Now consider the 8 puzzle problem tray and arranged in specified order. While moving from the start state towards goal state, we may make some stupid move and consider theorem

proving. We may proceed by first proving lemma. But we may backtrack and undo the unwanted move. This only involves additional steps and the solution steps are recoverable.
Lastly consider the game of chess. If a wrong move is made, it can neither be ignored nor be recovered. The thing to do is to make the best use of current situation and proceed. This is an example of an irrecoverable solution steps. A knowledge of these will help in determining the control structure.

3.. Is the Universal Predictable?

Problems can be classified into those with certain outcome (eight puzzle and water jug problems) and those with uncertain outcome (playing cards). In certain – outcome problems, planning could be done to generate a sequence of operators that guarantees to a lead to a solution. Planning helps to avoid unwanted solution steps. For uncertain outcome problems, planning can at best generate a sequence of operators that has a good probability of leading to a solution. The uncertain outcome problems do not guarantee a solution and it is often very expensive since the number of solution and it is often very expensive since the number of solution paths to be explored increases exponentially with the number of points at which the outcome cannot be predicted. Thus one of the hardest types of problems to solve is the irrecoverable, uncertain – outcome problems (Ex:- Playing cards).

4. Is good solution absolute or relative?

There are two categories of problems. In one, like the water jug and 8 puzzle problems, we are satisfied with the solution, unmindful of the solution path taken, whereas in the other category not just any solution is acceptable. We want the best, like that of traveling sales man problem, where it is the shortest path. In any – path problems, by heuristic methods we obtain a solution and we do not explore alternatives. For the best-path problems all possible paths are explored using an exhaustive search until the best path is obtained.

5. Is the solution a state or a path?

Consider the problem of finding a consistent interpretation for the sentence.
The bank president ate a dish of pasta salad with the fork.
There are several components of this sentence, each of which, in isolation, may have more than one interpretation, But the components must form a coherent whole, and so they constrain each other’s interpretations. Some of the sources of ambiguity in this sentence are the following:

The word "bank" may refer either to a financial institution or to a side of a river. But only one of these may have a president.

- The word "dish" is the object of the verb "eat." It is possible that a dish was eaten. But it is more likely that the pasta salad in the dish was eaten.
- Pasta salad is a salad containing pasta. But there are other ways meanings can be formed from pairs of nouns. For example, dog food does not normally contain dogs.

Because of the interaction among the interpretations of the constituents of this sentence, some search may be required to find a complete interpretation for the sentence. But to solve the problem of finding the interpretation we need to produce only the interpretation itself. No record of the processing by which the interpretation was found is necessary.

Contrast this with the water jug problem. Here it is not sufficient to report that we have solved the problem and that the final state is (2. 0). For this kind of problem, what we really must report is not the final state but the path that we found to that state. Thus a statement of a solution to this problem must be a sequence of operations that produces the final state.

These two examples, natural language understanding and the water jug problem, illustrate the difference between problems whose solution is a state of the world and problems whose solution is a path to a state.

6. What is the role of Knowledge?

Though one could have unlimited computing power, the size of the knowledge base available for solving the problem does matter in arriving at a good solution. Take for example the game of playing chess, just the rules for determining legal moves and some simple control mechanism is sufficient to arrive at a solution. But additional knowledge about good strategy and tactics could help to constrain the search and speed up the execution of the program. The solution would then be realistic. Consider the case of predicting the political trend. This would require an enormous amount of knowledge even to be able to recognize a solution, leave alone the best.

7. Does the task requires interaction with the person.

The problems can again be categorized under two heads.

1. Solitary in which the computer will be given a problem description and will produce an answer, with no intermediate communication and with the demand for an explanation of the reasoning process. Simple theorem proving falls under this category, given the basic rules

and laws, the theorem could be proved, if one exists. Ex:- theorem proving (give basic rules & laws to computer)

2. Conversational, in which there will be intermediate communication between a person and the computer, weather to provide additional assistance to the computer or to provide additional informed information to the user, or both problems such as medical diagnosis fall under this category, where people will be unwilling to accept the verdict of the program, if they cannot follow its reasoning. Ex:- Problems such as medical diagnosis.

AI problems with 7 problem characteristics

Chess

Problem characteristic	**Satisfied**	**Reason**
Is the problem decomposable?	No	One game have Single solution
Can solution steps be ignored or undone?	No	In actual game(not in PC) we can't undo previous steps
Is the problem universe predictable?	No	Problem Universe is not predictable as we are not sure about move of other player(second player)
Is a good solution absolute or relative?	absolute	Absolute solution: once you get one solution you do need to bother about other possible solution. Relative Solution: once you get one solution you have to find another possible solution to check which solution is best (i.e low cost). By considering this **chess is absolute**
Is the solution a state or a path?	Path	For natural language understanding, some of the words have different interpretations, therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only, the workings are not necessary (i.e path to solution is not necessary). So In chess winning state(goal state) describe path to state
What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	No	Conversational in which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or ***to provide additional information to the user***, or both. In chess additional assistance is not required

Water jug

Problem characteristic	**Satisfied**	**Reason**

Is the problem decomposable?	No	One Single solution
Can solution steps be ignored or undone?	Yes	
Is the problem universe predictable?	Yes	Problem Universe is predictable bcoz to solve this problem it require only one person we can predict what will happen in next step
Is a good solution absolute or relative?	absolute	**Absolute solution**: water jug problem may have number of solution , but once we found one solution, no need to bother about other solution **Bcoz it doesn't effect on its cost**
Is the solution a state or a path?	Path	Path to solution
What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	Yes	additional assistance is required. Additional assistance, like to get jugs or pump

8 puzzle

Problem characteristic	**Satisfied**	**Reason**
Is the problem decomposable?	No	One game have Single solution
Can solution steps be ignored or undone?	Yes	We can undo the previous move
Is the problem universe predictable?	Yes	Problem Universe is predictable bcoz to solve this problem it require only one person, we can predict what will be **position of blocks** in next move
Is a good solution absolute or relative?	absolute	8 puzzle **is absolute**
Is the solution a state or a path?	Path	In 8 puzzle winning state(goal state) describe path to state
What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	No	In 8 puzzle additional assistance is not required

Travelling Salesman (TSP)

Problem characteristic	**Satisfied**	**Reason**
Is the problem decomposable?	No	One game have Single solution
Can solution steps be ignored or undone?	Yes	
Is the problem universe predictable?	Yes	
Is a good solution absolute or relative?	absolute	TSP **is absolute**
Is the solution a state or a path?	Path	In TSP (goal state) describe path to state
What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.

Does the task require human-interaction?	No	In chess additional assistance is not required

Missionaries and cannibals

Problem characteristic	**Satisfied**	**Reason**
Is the problem decomposable?	No	One game have Single solution
Can solution steps be ignored or undone?	Yes	
Is the problem universe predictable?	Yes	Problem Universe is not predictable as we are not sure about move of other player(second player)
Is a good solution absolute or relative?	absolute	
Is the solution a state or a path?	Path	In winning state(goal state) describe path to state
What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	Yes	Additional assistance is required to move **Missionaries to other side of river of other assistance is required**

Tower of Hanoi

Problem characteristic	**Satisfied**	**Reason**
Is the problem decomposable?	No	One game have Single solution
Can solution steps be ignored or undone?	Yes	
Is the problem universe predictable?	Yes	
Is a good solution absolute or relative?	absolute	Tower of Hanoi is **absolute**
Is the solution a state or a path?	Path	In tower of Hanoi winning state(goal state) describe path to state
What is the role of knowledge?		lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	No	In tower of Hanoi additional assistance is not required

Production System Characteristics

1. Can production systems, like problems, be described by a set of characteristics that shed some light on how they can easily be implemented?

2. If so, what relationships are there between problem types and the types of production systems best suited to solving the problems?

The answer to the first question is yes. Consider the following definitions of classes of production systems. A **monotonic production system** is a production system in which the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected. A **non-monotonic** production system is one in which this is not true. A **partially commutative** production system is a production system with the property that if the application of a particular sequence of rules transforms state x into state y, then any permutation of those rules that is allowable also transforms state x into state y. A **commutative** production system is a production system that is both monotonic and partially commutative.

Coming at the second question above, which asked whether there is an interesting relationship between classes of production systems and classes of problems. For any solvable problem, there exist an infinite number of production systems that describe ways to find solutions. Some will be more natural or efficient than others. Any problem that can be solved by any production system can be solved by a commutative one, but the commutative one may be so unwieldy as to be practically useless. It may use individual states to represent entire sequences of applications of rules of a simpler, noncommutative system. So in a formal sense, there is no relationship between kinds of problems and kinds of production systems since all problems can be solved by all kinds of systems. But in a practical -sense, there definitely is such a relationship between kinds of problems and the kinds of systems that lend themselves naturally to describing those problems. To see this, let us look at a few examples.

There are 4 categories of production systems, monotonic versus nonmonotonic and partially commutative versus non-partially commutative. Partially commutative, monotonic production systems are useful for solving ignorable problems.eg. theorem proving. Nonmonotonic, partially commutative systems, on the other hand, are useful for problems in which changes occur but can be reversed and in which order of operations is not critical. Eg. Robot navigation.

Production systems that are not partially commutative are useful for many problems in which irreversible changes occur. For example, chemical synthesis. Non-partially commutative production systems are less likely to produce the same node many times in the search process. When dealing with ones that describe irreversible processes, it is particularly

important to make correct decisions the first time, although if the universe is predictable, planning can be used to make that less important. Eg bridge.

Issues in the design of search programs

Every search process can be viewed as a traversal of a tree structure in which each node represent, a problem state and each arc represents a relationship between the states represented by the nodes it connects. For example, below Fig. shows part of a search tree for a water jug problem. The arcs have not been labeled in the Fig., but they correspond to particular water-pouring operations. The search process must find a path or paths through the tree that connect an initial state with one or more final stales. The tree that must be searched could, be constructed in its entirety from the rules that define allowable move, in the problem space.

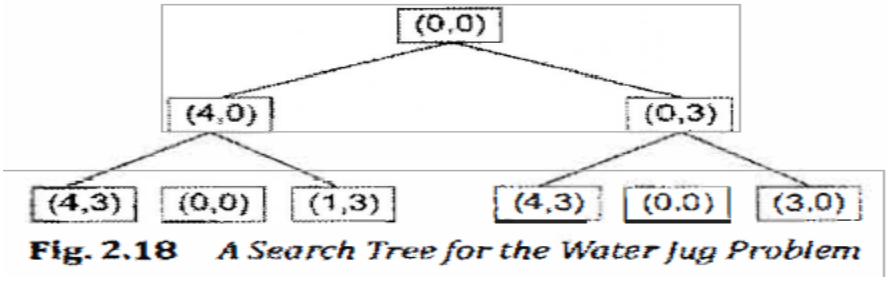


Fig. 2.18 *A Search Tree for the Water Jug Problem*

But, in practice, most of it never is. It is too large and most of it need never be explored. Instead of first building the tree explicitly and then searching it, most search programs represent the tree implicitly in the rules and generate explicitly only those parts that they decide to explore. Some important issues in the design of search programs are:

- The direction in which to conduct the search (forward versus backward reasoning). We can search forward through the state space from the start state to a goal state, or we can search backward from the goal.

- How to select applicable rules. Production systems typically spend most of their time looking for rules to apply, so it is critical to have efficient procedures for matching rules against states.

- How to represent each node of the search process. For problems like chess, a node can be fully represented by a simple array. In more complex problem solving, however, it is inefficient and/or impossible to represent all of the facts in the world and to determine all of the side effects an action may have.

Search trees versus search graphs: We can think of production rules as generating nodes in a search tree. Each node can be expanded in turn, generating a set of successors. This process continues until a node representing a solution is found. Implementing such a procedure requires little bookkeeping. However, this process often results in the same node being generated as part of several paths and so being processed more than once. This happens because the search space may really be an arbitrary directed graph rather than a tree.

For example, in the tree shown in Fig. above, the node (4,3), representing 4-gallons of water in one jug and 3 gallons in the other, can he generated either by first filling the 4-gallon jug and then the 3-gallon one or by filling them in the opposite order. Since the order does not matter, continuing to process both these nodes would be redundant. This example also illustrates another problem that often arises, when the search process operates as a tree walk. On the third level, the node (0, 0) appears. But this is the same as the top node of the tree, which has already been expanded. Those two paths have not gotten Us anywhere. So we eliminate them and continue only along the other branches.

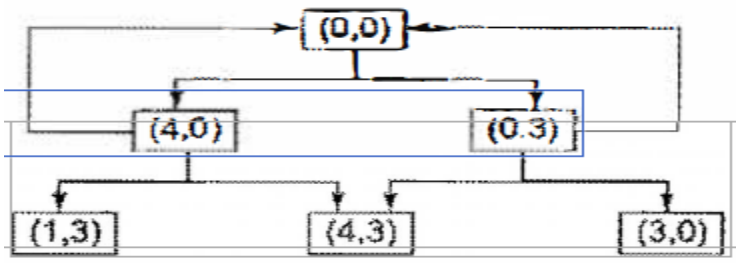


Fig. 2.19 *A Search Graph for the Water Jug Problem*

The waste of effort that arises when the same node is generated more than once can be avoided at the price of additional bookkeeping. Instead of traversing a search tree, we traverse a directed graph. This graph differs from a tree in that several paths may come together at a node. The graph corresponding to the tree is shown in Fig. 2.19. Any tree search procedure that keeps track of all the nodes that have been generated so far can be converted to a graph search procedure by modifying the action performed each time a node is generated.

Treating the search process as a graph search rather than as a tree search reduces the amount of efforts that is spent exploring essentially the same path several times. But it requires additional effort each time a node is generated to see if it has been generated before. Whether this effort is justified depends on the particular problem. If it is very likely that the same

node will he generated in several different ways, then it is more worthwhile to use a graph procedure than if such duplication will happen only rarely. Graph search procedures are especially useful for dealing with partially commutative production systems in which a given set of operations will produce the same result regardless of the order in which the operations are applied. A systematic search procedure will try many of the permutations of these operator, and so will generate the same node many times.

SEARCHING

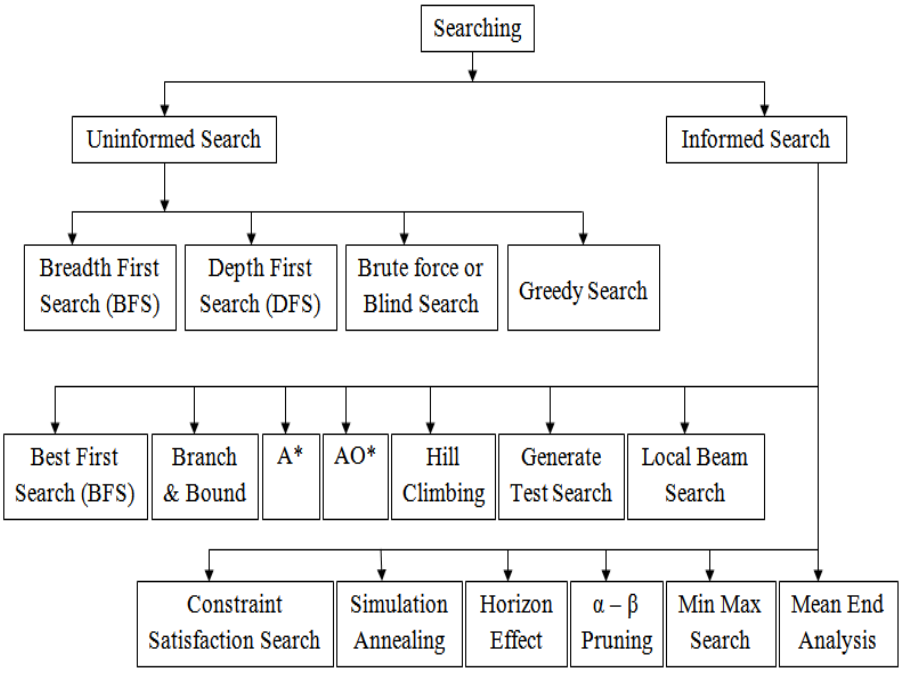
Problem solving in artificial intelligence may be characterized as a systematic search through a range ofpossible actions in order to reach some predefined goal or solution. In AI problem solving by search algorithms is quite common technique. This chapter contains the different search algorithms of AI used in various applications.

A search algorithm takes a problem as input and returns the solution in the form of an action sequence.Once the solution is found, the actions it recommends can be carried out. This phase is called as the execution phase. A problem can be defined by 5 components.

a) **The initial state**: The state from which agent will start.
b) **The goal state**: The state to be finally reached.
c) **The current state**: The state at which the agent is present after starting from the initial state.
d) **Successor function**: It is the description of possible actions and their outcomes.
e) **Path cost**: It is a function that assigns a numeric cost to each path.

DIFFERENT TYPES OF SEARCHING

The searching algorithms can be of various types. When any type of searching is performed, there may some information about the searching or mayn't be. Also it is possible that the searching procedure may dependupon any constraints or rules. However, generally searching can be classified into two types i.e. uninformed searching and informed searching.



UNINFORMED SEARCH

Breadth First Search (BFS)

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. In this type of search the state space is represented in form of a tree. The solution is obtained by traversing through the tree. The nodes of the tree represent the start value or starting state, various intermediate states and the final state. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution if one exists. The solution which is found is always the optional solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level.

Concept:

Step 1: Traverse the root node

Step 2: Traverse all neighbours of root node.

Step 3: Traverse all neighbours of neighbours of the root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: Place the root node inside the queue.

Step 2: If the queue is empty then stops and return failure.

Step 3: If the FRONT node of the queue is a goal node then stop and return success.

Step 4: Remove the FRONT node from the queue. Process it and find all its neighbours that are in readystate then place them inside the queue in any order.

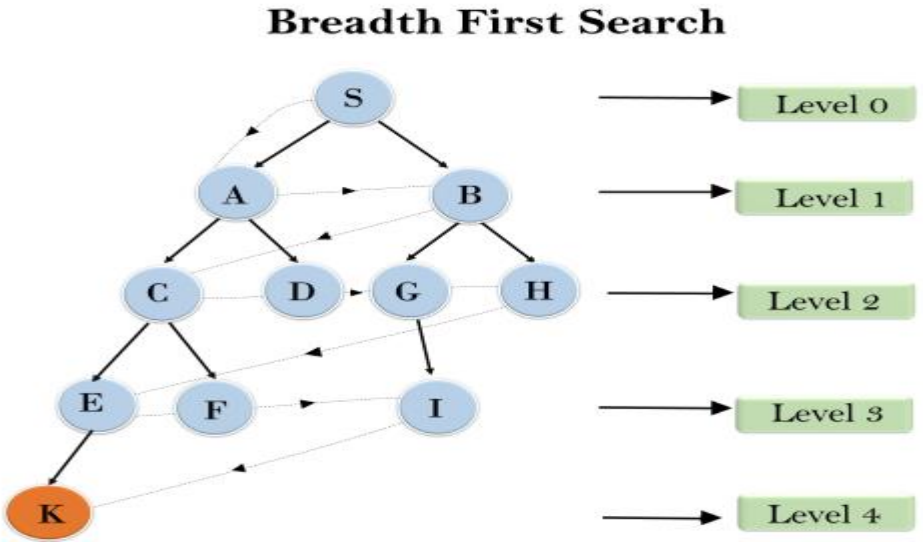
Step 5: Go to Step 3.

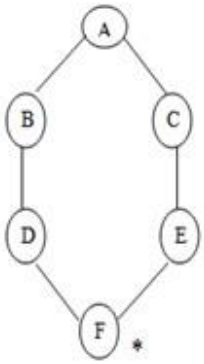
Step 6: Exit.

Implementation:

Let us implement the above algorithm of BFS by taking the following suitable example.

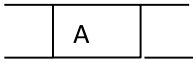
```
S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K
```





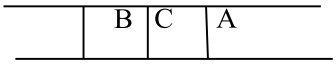
Consider the graph in which let us take A as the starting node and F as the goal node (*)

Step 1: Place the root node inside the queue i.e. A

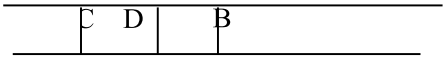


Step 2: Now the queue is not empty and also the FRONT node i.e. A is not our goal node. So move to step 3.

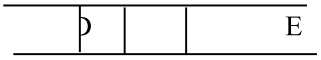
Step 3: So remove the FRONT node from the queue i.e. A and find the neighbour of A i.e. B and C



Step 4: Now b is the FRONT node of the queue. So process B and finds the neighbours of B i.e. D.



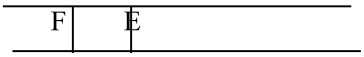
Step 5: Now find out the neighbours of C i.e. E



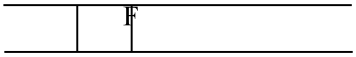
C

Step 6: Next find out the neighbours of D as D is the FRONT node of the queue

Step 7: Now E is the front node of the queue. So the neighbour of E is F which is our goal node.



Step 8: Finally F is our goal node which is the FRONT of the queue. So exit.



Advantages:

- In this procedure at any way it will find the goal.
- It does not follow a single unfruitful path for a long time.
- It finds the minimal solution in case of multiple paths.

Disadvantages:

- BFS consumes large memory space.
- Its time complexity is more.
- It has long pathways, when all paths to a destination are on approximately the same search depth.

Depth First Search (DFS)

DFS visits all the vertices in the graph. This type ofalgorithm always chooses to go deeper into the graph. After DFS visited all the reachable vertices from aparticular sources

vertices it chooses one of the remaining undiscovered vertices and continues the search. DFS reminds the space limitation of breath first search by always generating next a child of the deepestunexpanded node. The data structure stack or last in first out (LIFO) is used for DFS.

Concept:

Step 1: Traverse the root node.

Step 2: Traverse any neighbour of the root node.

Step 3: Traverse any neighbour of neighbour of the root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: PUSH the starting node into the stack.

Step 2: If the stack is empty then stop and return failure.

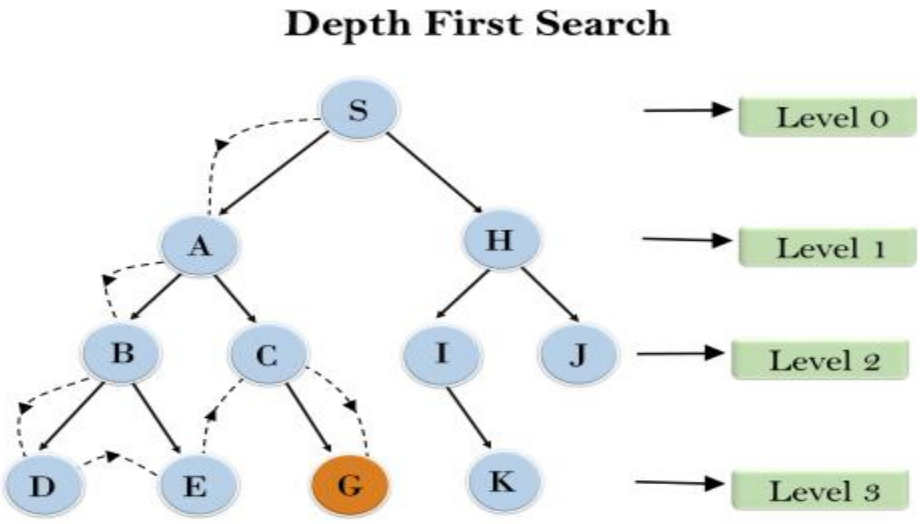
Step 3: If the top node of the stack is the goal node, then stop and return success.

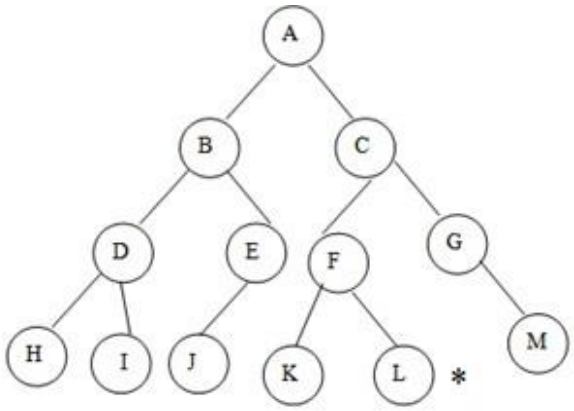
Step 4: Else POP the top node from the stack and process it. Find all its neighbours that are in ready stateand PUSH them into the stack in any order.

Step 5: Go to step 3.

Step 6: Exit.

Implementation:





Consider A as the root node and L as the goal node in the graph figure

Step 1: PUSH the starting node into the stack i.e.

A	

Step 2: Now the stack is not empty and A is not our goal node. Hence move to next step.

Step 3: POP the top node from the stack i.e. A and find the neighbours of A i.e. B and C.

B	C	

A

Step 4: Now C is top node of the stack. Find its neighbours i.e. F and G.

B	F	G	

C

Step 5: Now G is the top node of the stack. Find its neighbour i.e. M

B	F	M	

G

Step 6: Now M is the top node and find its neighbour, but there is no neighbours of M in the graph soPOP it from the stack.

B	F	

M

Step 7: Now F is the top node and its neighbours are K and L. so PUSH them on to the stack.

B	K		

L F

Step 8: Now L is the top node of the stack, which is our goal node.

B	K	

L

Advantages:

- DFS consumes very less memory space.
- It will reach at the goal node in a less time period than BFS if it traverses in a right path.
- It may find a solution without examining much of search because we may get the

desired solutionin the very first go.

Disadvantages:

- It is possible that may states keep reoccurring.
- There is no guarantee of finding the goal node.
- Sometimes the states may also enter into infinite loops.

Difference between BFS and DFS

BFS

- It uses the data structure queue.
- BFS is complete because it finds the solution if one exists.
- BFS takes more space i.e. equivalent to o (b^d) where b is the maximum breath exist in a search tree and d is the maximum depth exit in a search tree.
- In case of several goals, it finds the best one.

DFS

- It uses the data structure stack.
- It is not complete because it may take infinite loop to reach at the goal node.
- The space complexity is O (d).
- In case of several goals, it will terminate the solution in any order.

Depth Bounded DFS (DBDFS)

A depth-bounded DFS search algorithm is similar to depth-first search with a predetermined limit. Depth-bounded 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-bounded search can be terminated with two Conditions of failure:

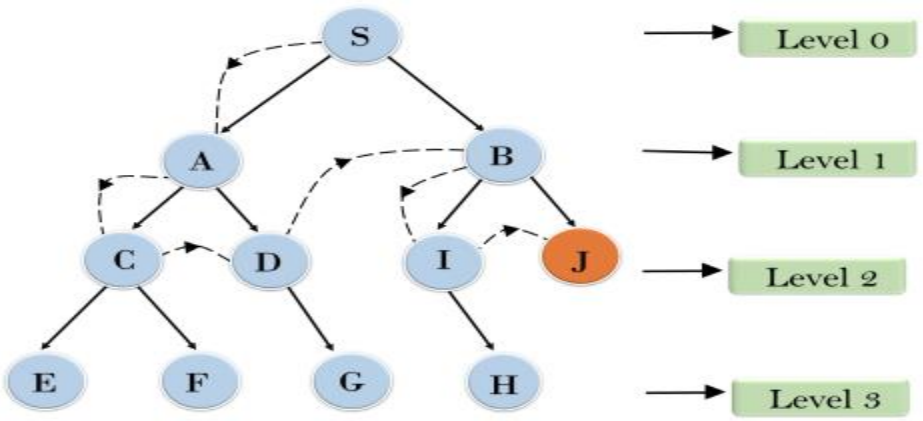
Standard failure value: It indicates that problem does not have any solution.

Cutoff failure value: It defines no solution for the problem within a given depth limit.

Advantages: Depth-bounded search is Memory efficient.

Disadvantages: Depth-bounded search also has a disadvantage of incompleteness.

It may not be optimal if the problem has more than one solution.



Depth First Iterative Deepening (DFID)

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: It combines 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. DFID algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:

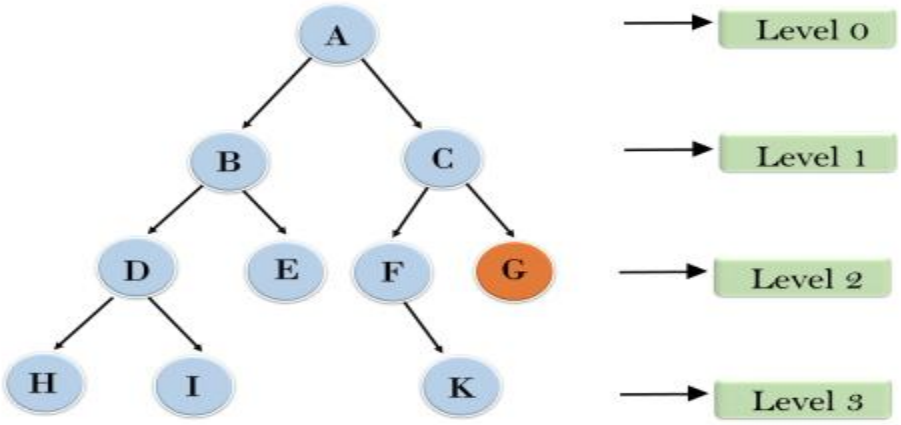
1'st Iteration-----> A

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.



The Traveling Salesman Problem: A salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list. Find the route the salesman should follow for the shortest possible round trip that both starts and finishes at any one of the cities.

A simple, motion-causing and systematic control structure could, in principle, solve this problem. It would simply explore all possible paths in the tree and return the one with the shortest length. This approach will even work in practice for very short lists of cities. But it breaks down quickly as the number of cities grows. If there are N cities. then the number of different paths among them is 1.2 .. (N -· l), or (N – 1)!. The time to examine a single path is proportional to N. So the total time required to perform this search is proportional to N!. Assuming there are only 10 cities, 10! is 3,628,800, which is a very large number. The salesman could easily have 25 cities to visit. To solve this problem would take more time than he would be willing to spend. This phenomenon is ca1led combinatorial explosion. To combat it we need a new control strategy.

We can beat the simple strategy outlined above using a technique called branch- and- bound. Begin generating complete paths, keeping track of the shortest path found so far. Give up exploring any path as soon as its partial length becomes greater than the shortest path found so far. Using this technique, we are still guaranteed to find the shortest path, Unfortunately, although this algorithm is more efficient than the first one, it still requires exponential time. The exact amount of time it saves for a particular problem depends on the order in which the paths are explored. But it is still inadequate for solving large problems.