Problem Solving Techniques in Artificial Intelligence (AI)

Problem-solving is commonly known as the method to reach the desired goal or find a solution to a given situation. In computer science, problem-solving refers to artificial intelligence techniques, including various techniques such as forming efficient algorithms, heuristics, and performing root cause analysis to find desirable solutions. Problem-solving in Artificial Intelligence usually refers to researching a solution to a problem by performing logical algorithms, utilizing polynomial and differential equations, and executing them using modeling paradigms. There can be various solutions to a single problem, which are achieved by different heuristics. Also, some problems have unique solutions. It all rests on the nature of the given problem.

Examples of Problems in Artificial Intelligence

Developers worldwide are using artificial intelligence to automate systems for efficient utilization of time and resources. Some of the most common problems encountered in day-to-day life are games and puzzles. These can be solved efficiently by using artificial intelligence algorithms. Ranging from mathematical puzzles including crypto-arithmetic and magic squares, logical puzzles including Boolean formulas and N-Queens to popular games like Sudoku and Chess, these problem-solving techniques are used to form a solution for all these. Therefore, some of the most prevalent problems that artificial intelligence has resolved are the following:

- **Chess**
- **N-Queen problem**
- **Tower of Hanoi Problem**
- **Travelling Salesman Problem**
- **Water-Jug Problem**

1. Chess Problem:

To find out whether the solutions steps can be ignored or undone. We have three classes of problems mentioned below :-

1. **Ignorable :** In which solution steps can be ignored (e.g., Theorem Proving).

Suppose we are trying to prove a mathematical theorem. First we proceed with proving a lemma that we think will be useful. Later we realize that it is not useful. So, Are we in serious trouble?. No, Still we can prove the theorem, only we need to ignore the first approach and start with another one to prove the theorem.

2. **Recoverable :** In which solution steps can be undone. (e.g., 8-Puzzle Problem).8 Puzzle ProblemSuppose we have a 8-puzzle problem as shown in above figure. While attempting to solve the 8-puzzle problem, mistakenly we make a wrong move and realize that mistake. Now to correct our mistake we need to undo incorrect steps.

To undo incorrect steps the control mechanism for an 8-puzzle solver must keep track of the order in which steps are performed, so that we can backtrack to the initial state and start with some correct move.

3. **Irrecoverable :** In which solution steps cannot be undone. (e.g., Chess).ChessConsider the problem of playing chess. Suppose playing chess program makes a wrong move and realize it after couple of moves. It cannot simply ignore the mistake. Neither it can be undone its move and start the game again. Here, once we make a move we never recover from that step. Only we can try to give the best of the current situation.

N–Queens problem

The N–Queens problem is a classic problem that is often used in discussions of various search strategies. The problem is often defined in terms of a standard 8–by–8 chess board, although it can be defined for any N–by–N board and is solvable for N ³ 4.

Background

As is the case with all such problems, the 8–queens problem is posed as an example to be used in discussing search methods, not as a problem that has value in and of itself. In general, the problem is to place the queens on the board so that no two queens can attack each other. The method for solving the problem depends on the knowledge of the rules

for moving a queen in the game of chess. Specifically, a queen can move in any straight line, either along a row, along a column, or a diagonal. In an N–by–N board, each queen will be located on exactly one row, one column, and two diagonals.

The requirement that no two queens be placed in the same row restricts the number of queens that can be placed on an N–by–N board to N. Thus, a standard 8–by–8 chess board can take at most eight queens. The problem is to find an arrangement that allows placement of N queens.

Tower of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods (A, B, and C) and N disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack to another rod (here considered C), obeying the following simple rules:

Only one disk can be moved at a time.

Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

No disk may be placed on top of a smaller disk.

Examples:

Input: 2

Output: Disk 1 moved from A to B

Disk 2 moved from A to C

Disk 1 moved from B to C

Input: 3

Output: Disk 1 moved from A to C

Disk 2 moved from A to B

Disk 1 moved from C to B

Disk 3 moved from A to C

Disk 1 moved from B to A

Disk 2 moved from B to C

Disk 1 moved from A to C

What is Traveling Salesman Problem (TSP)?

The traveling Salesman Problem (TSP) is a combinatorial problem that deals with finding the shortest and most efficient route to follow for reaching a list of specific destinations.

It is a common algorithmic problem in the field of delivery operations that might hamper the multiple delivery process and result in financial loss. TSP turns out when you have multiple routes available but choosing a minimum cost path is really hard for you or a traveling person.

How difficult is it to solve?

It is quite difficult to solve TSP as it is known as NP-hard, which means there is no polynomial time algorithm to solve it for more numbers of addresses. So, with an increasing amount of addresses, the complexity of solving TSP increases exponentially.

So, it is impossible to find TSP solutions manually. Also, many mathematical algorithms and the fastest computers fail to solve TSP.

However, TSP can be eliminated by determining the optimized and efficient path using approximation algorithms or automated processes.

Common Challenges of Traveling Salesman Problem (TSP)

Being a salesman is not easy, as you need to face various unavoidable challenges in your everyday schedules.

- Firstly, every day, salespeople have to carry out a number of deliveries in a very limited time, so there are a lot of time constraints. To overcome this, you need to plan your routes in a way that you make the most out of them.
- Secondly, there are chances of last-minute changes. Sometimes you get extra and urgent visits to make, while sometimes, some visits are postponed or canceled due to the customer's unavailability.
- Lastly, a math problem, a combinatorial optimization problem, arises. A combinatorial optimization problem is a problem that is mathematically complex to solve as you have to deal with many variables.

These are major challenges in the Traveling Salesman Problem (TSP) as you are required to create a route with the shortest distances using hundreds and thousands of

permutations and combinations that asks for less fuel, fulfill on-time delivery to customers, and are ready to modify routes considering last minute changes.

What are Some Popular Solutions to Traveling Salesman Problem?

These are some of the near-optimal solutions to find the shortest route to a combinatorial optimization problem.

1. Nearest Neighbor (NN)

The Nearest Neighbor Method is probably the most basic TSP heuristic. The method followed by this algorithm states that the driver must start by visiting the nearest destination or closest city. Once all the cities in the loop are covered, the driver can head back to the starting point.

Solving TSP using this efficient method, requires the user to choose a city at random and then move on to the closest unvisited city and so on. Once all the cities on the map are covered, you must return to the city you started from.

2. The Branch and Bound Algorithm

The Branch & Bound method follows the technique of breaking one problem into several little chunks of problems. So it solves a series of problems. Each of these sub-problems may have multiple solutions. The solution you choose for one problem may have an effect on the solutions of subsequent sub-problems.

3. The Brute Force Algorithm

The Brute Force Approach takes into consideration all possible minimum cost permutation of routes using a dynamic programming approach. First, calculate the total number of routes. Draw and list all the possible routes that you get from the calculation. The distance of each route must be calculated and the shortest route will be the most optimal solution.

What are Other Optimal Solutions to the Traveling Salesman Problem?

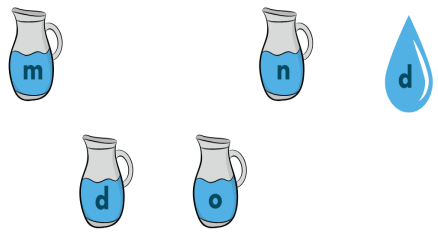
Multi-Agent System: Involves distributing the pair of cities into groups. Then assign a single agent to discover the shortest path, covering all the cities in the assigned group.

- **Zero Suffix Method**: This method solves the classical symmetric TSP and was introduced by Indian researchers.
- **Multi-Objective Evolutionary Algorithm**: This method solves the TSP using NSGA-II
- **Biogeography-based Optimization Algorithm**: This method is based on the migration strategy of animals for solving optimization issues.
- **Meta-Heuristic Multi Restart Iterated Local Search**: This method states that the technique is more efficient compared to genetic algorithms.

What is the Water Jug Problem?

Here is the problem statement for Water Jug Problem:

You are given 2 jugs with the capacity 'm' and 'n' respectively. Initially, they are given empty. There is an unlimited supply of water. You can either fill the whole jug or a quantity that is less than the given capacity of jugs. Now, you are also given a third positive integer 'd'. Using the 2 given jugs, you need to come up with a solution to have 'd' amount of water in them and return the number of steps you took to reach that capacity.



Understanding the Problem

Before beginning the coding part, it is very important for you to understand what the question is asking you to do. Many students just copy the code from web servers and later, when the interviewer asks any question from it, they are stuck and even embarrassed. So, to save yourself from such questions, kindly go through this section carefully.

So, you are given 2 jugs with their capacities. It has also been clearly stated in the question that with the unlimited water supply, you can either fill the whole jug up to its capacity or somewhat less than that. Keep that in mind.

Secondly, we need to find a way to somehow have a 'd' amount of water left in the jugs. Basically, we need to transfer, fill and, empty the jugs to get the required capacity of water in the jugs.

The next most important question is which data structure should you choose for this question. You cannot use arrays as it will be a heavy task for both the compiler(in terms of time and auxiliary space complexity) and you(in terms of long codes).

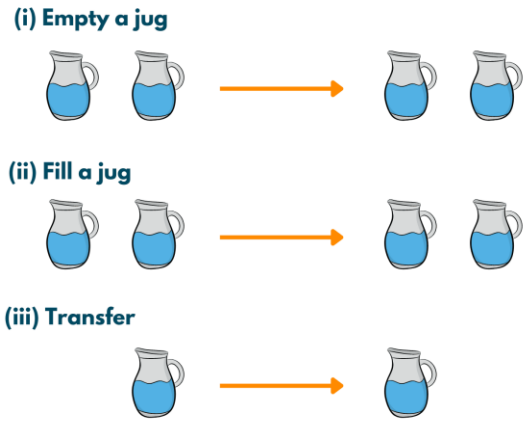
You should choose a structure with which you can manage the content of both jugs simultaneously on the front. The possible and safest option is to use Sets or Pairs.

How to Approach the Solution

Now that we have chosen what to work with, let's understand how you can actually make it work. Well, for starters you can have (a, b) which represents the amount of water currently in jug 1 and jug 2 respectively. Initially, both the components will be (0, 0) since the jugs are empty in the beginning. The final state of the jugs will be either (0, d) or (d, 0) as both add up to give a total of the required quantity.

The following operations can be performed on the jugs:

1. Empty a jug (a, b) -> (0, b)
2. Fill a jug (0, b) -> (a, b)
3. Transfer water from one jug to another.



Example

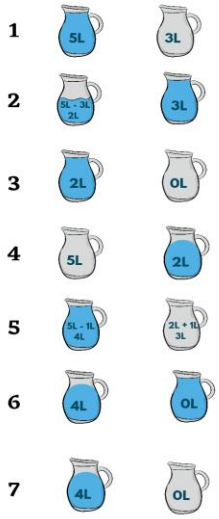
Input: 3, 5, 4

Output: 6

Explanation: The following steps are taken:

1. Fill the 5-liter jug completely.
2. Transfer 3 liters from a 5-liter jug to a 3-liter jug.
3. Empty the 3-liter capacity jug.
4. Transfer the remaining 2 liters from a 5-liter jug to a 3-liter jug.
5. Now, fill the 5-liter jug fully.
6. Pour 1 liter from a 5-liter jug into a 3-liter jug.
7. There! We have 4 liters in the first jug, now empty the 2nd jug.

For more clarity, check the below illustration.



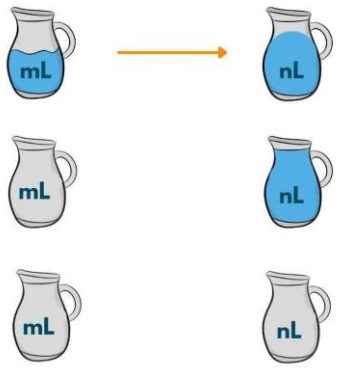
03 Methods to Solve the Water Jug Problem

Now, you know what the problem is and what is the approach behind the solution. Next, discuss some methods with which you can code this problem. Following are the three methods, you can use:

Method 01) Always pour from 1st jug to 2nd jug

1. Fill the m liter jug and empty it into the n liter jug.
2. When the m liter jug becomes empty refill it.
3. When the n-liter jug becomes full, empty it.

4. Repeat the above steps till any one of the jugs contains the required amount of water in them.



Method 02) Always pour from 2nd jug to 1st jug

1. Fill the n-liter jug and empty it into the m-liter jug.
2. When the n liter jug becomes empty refill it.
3. When the m liter jug becomes full, empty it.
4. Repeat the above steps till any of the jugs contain the required amount of water in them.