# Lab-06

# a. To Implement a program to solve traveling salesman problems

## Objectives:

- To compute Z-transform of discrete time signals.
- To compute inverse Z-transform of discrete time signals.

## Apparatus:

- **Hardware Requirement**
  Personal computer.
- **Software Requirement**
  Anaconda.

## Theory:

The traveling salesman problem is a classic problem in combinatorial optimization. This problem is to find the shortest path that a salesman should take to traverse through a list of cities and return to the origin city. The list of cities and the distance between each pair are provided.

TSP is useful in various applications in real-life such as planning or logistics. For example, a concert tour manager who wants to schedule a series of performances for the band must determine the shortest path for the tour to ensure reducing traveling costs and not making the band unnecessarily exhausted.

This is an NP-hard problem. In simple words, it means you cannot guarantee to find the shortest path within a reasonable time limit. This is not unique to TSP though. In real-world optimization problems, you frequently encounter problems for which you must find sub-optimal solutions instead of optimal ones.

### Traveling Salesman Problem (TSP) Implementation

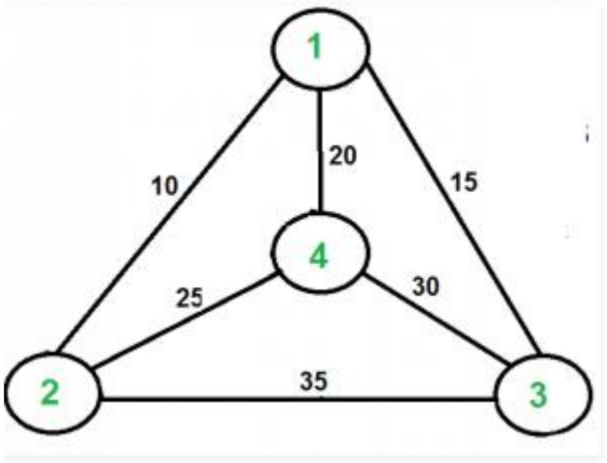
**Travelling Salesman Problem (TSP):**

Given a set of cities and distances between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Note the difference between Hamiltonian Cycle and TSP. The Hamiltonian cycle problem is to find if there exists a tour that visits every city exactly once. Here we know that the Hamiltonian Tour exists (because the graph is complete) and in fact, many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.

**For example,** consider the graph shown in the figure on the right side. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.

The problem is a famous NP-hard problem. There is no polynomial-time known solution for this problem.



Examples:

Output of Given Graph:

minimum weight Hamiltonian Cycle :

10 + 25 + 30 + 15 := 80

**Discussion of the implementation**

1. Consider city 1 as the starting and ending point. Since the route is cyclic, we can consider any point as a starting point.
2. Generate all (n-1)! permutations of cities.
3. Calculate the cost of every permutation and keep track of the minimum cost permutation.
4. Return the permutation with minimum cost.

**Lab Task:**

Implement the discussion of TSP in python.

## b. To implement the tower of Hanoi problem

### Objectives:

- To understand the concept of problem solving agent.
- To implement the tower of hanoi problem

### Apparatus:

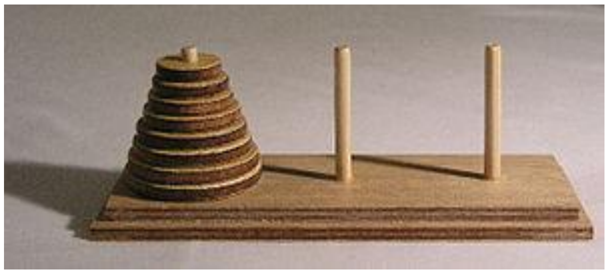
- Hardware Requirement
  Personal computer.
- Software Requirement
  Anaconda

### Theory:

The Tower of Hanoi (also called the Tower of Brahma or Lucas' Tower and sometimes pluralized as Towers) is a mathematical game or puzzle. It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, 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 or on an empty rod.
No larger disk may be placed on top of a smaller disk.
With 3 disks, the puzzle can be solved in 7 moves. The minimal number of moves required to solve a Tower of Hanoi puzzle is 2n − 1, where n is the number of disks.



## Solution

The puzzle can be played with any number of disks, although many toy versions have around 7 to 9 of them. The minimal number of moves required to solve a Tower of Hanoi puzzle is 2n − 1, where n is the number of disks.This is precisely the nth Mersenne number.

## Iterative solution

Animation of an iterative algorithm solving 6-disk problem.
A simple solution for the toy puzzle is to alternate moves between the smallest piece and a non-smallest piece. When moving the smallest piece, always move it to the next position in the same direction (to the right if the starting number of pieces is even, to the left if the starting number of pieces is odd). If there is no tower position in the chosen direction, move the piece 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. Doing this will complete the puzzle in the fewest moves

## Recursive solution

Illustration of a recursive solution for the Towers of Hanoi puzzle with 4 disks

The key to solving a problem recursively is to recognize that it can be broken down into a collection of smaller sub-problems, to each of which that same general solving procedure that we are seeking applies, and the total solution is then found in some simple way from those sub-problems' solutions. Each of thus created sub-problems being "smaller" guarantees that the base case(s) will eventually be reached. Thence, for the Towers of Hanoi:

label the pegs A, B, C,
let n be the total number of disks,
number the disks from 1 (smallest, topmost) to n (largest, bottom-most).

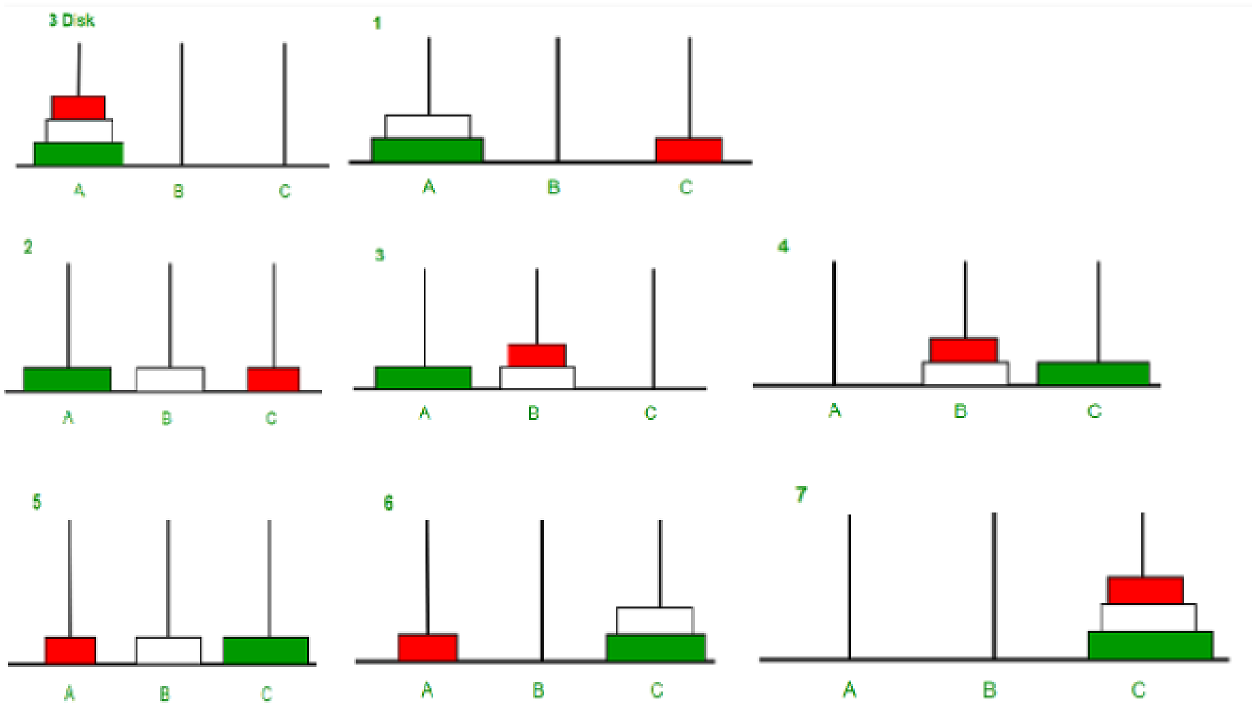
## Final Approach to implement :

Take an example for 2 disks :
Let rod 1 = 'A', rod 2 = 'B', rod 3 = 'C'.

Step 1 : Shift first disk from 'A' to 'B'.
Step 2 : Shift second disk from 'A' to 'C'.
Step 3 : Shift first disk from 'B' to 'C'.

The pattern here is :
Shift 'n-1' disks from 'A' to 'B'.
Shift last disk from 'A' to 'C'.
Shift 'n-1' disks from 'B' to 'C'.

Image illustration for 3 disks :



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

**Lab task:**

Implement the discussed approach for tower of Hanoi in Python Language.