# **Program for Tower of Hanoi Algorithm**

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*

## **Tower of Hanoi using Recursion:**

*The idea is to use the helper node to reach the destination using recursion. Below is the pattern for this problem:*

* *Shift ‘N-1’ disks from ‘A’ to ‘B’, using C.*
* *Shift last disk from ‘A’ to ‘C’.*
* *Shift ‘N-1’ disks from ‘B’ to ‘C’, using A.*

**

*Image illustration for 3 disks*

**Time complexity**: O(2N), There are two possibilities for every disk. Therefore, 2 \* 2 \* 2 \* . . . \* 2(N times) is 2N

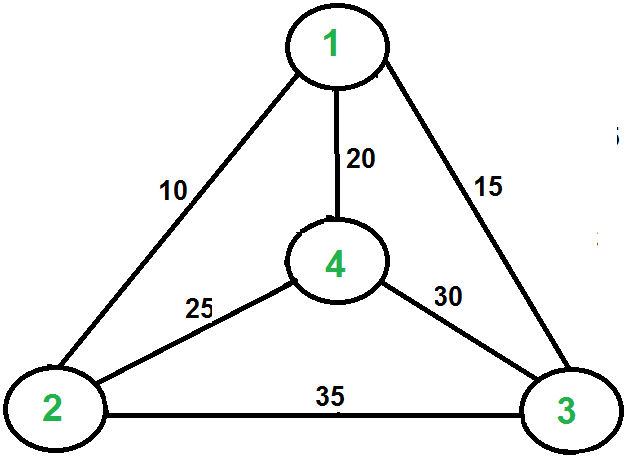
**Auxiliary Space:** O(N), Function call stack space

# **Travelling Salesman Problem using Dynamic Programming**

Last Updated : 19 Apr, 2023

**Travelling Salesman Problem (TSP):**

Given a set of cities and the distance 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](https://www.geeksforgeeks.org/backtracking-set-7-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 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-hardproblem. There is no polynomial-time know solution for this problem. The following are different solutions for the traveling salesman problem.

**Naive Solution:**

1) Consider city 1 as the starting and ending point.

2) Generate all (n-1)! [Permutations](https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/) of cities.

3) Calculate the cost of every permutation and keep track of the minimum cost permutation.

4) Return the permutation with minimum cost.

Time Complexity: ?(n!)

**Dynamic Programming:**

Let the given set of vertices be {1, 2, 3, 4,….n}. Let us consider 1 as starting and ending point of output. For every other vertex I (other than 1), we find the minimum cost path with 1 as the starting point, I as the ending point, and all vertices appearing exactly once. Let the cost of this path cost (i), and the cost of the corresponding Cycle would cost (i) + dist(i, 1) where dist(i, 1) is the distance from I to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This looks simple so far.

Now the question is how to get cost(i)? To calculate the cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems.

Let us define a term *C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i*. We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.

If size of S is 2, then S must be {1, i},

C(S, i) = dist(1, i)

Else if size of S is greater than 2.

C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j != i and j != 1.

Below is the dynamic programming solution for the problem using top down recursive+memoized approach:-

For maintaining the subsets we can use the bitmasks to represent the remaining nodes in our subset. Since bits are faster to operate and there are only few nodes in graph, bitmasks is better to use.

For example: –

10100 represents node 2 and node 4 are left in set to be processed

010010 represents node 1 and 4 are left in subset

# **Approximate solution for Travelling Salesman Problem using MST**

Last Updated : 28 Nov, 2022

We introduced [Travelling Salesman Problem](https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/) and discussed Naive and Dynamic Programming Solutions for the problem in the [previous post](https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/),. Both of the solutions are infeasible. In fact, there is no polynomial time solution available for this problem as the problem is a known NP-Hard problem. There are approximate algorithms to solve the problem though. The approximate algorithms work only if the problem instance satisfies Triangle-Inequality.

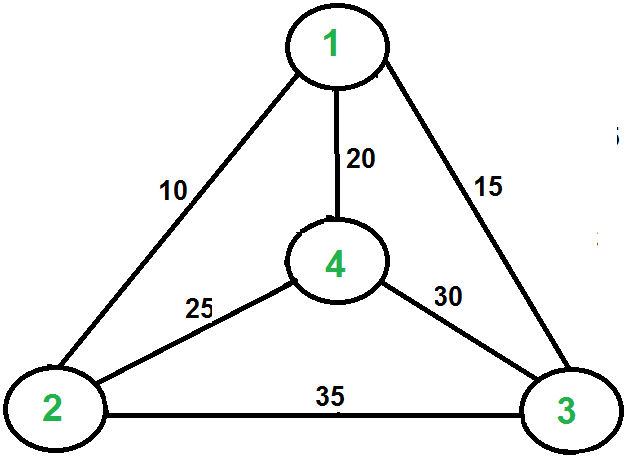
**Triangle-Inequality:** The least distant path to reach a vertex j from i is always to reach j directly from i, rather than through some other vertex k (or vertices), i.e., dis(i, j) is always less than or equal to dis(i, k) + dist(k, j). The Triangle-Inequality holds in many practical situations.

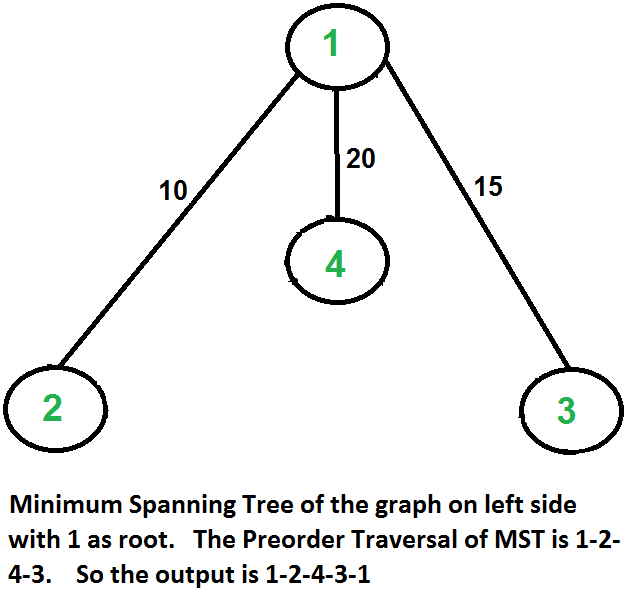
When the cost function satisfies the triangle inequality, we can design an approximate algorithm for TSP that returns a tour whose cost is never more than twice the cost of an optimal tour. The idea is to use **M**inimum **S**panning **T**ree (MST). Following is the MST based algorithm.

**Algorithm:**

1. Let 1 be the starting and ending point for salesman.
2. Construct MST from with 1 as root using [Prim’s Algorithm](https://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-minimum-spanning-tree-mst-2/).
3. List vertices visited in preorder walk of the constructed MST and add 1 at the end.

Let us consider the following example. The first diagram is the given graph. The second diagram shows MST constructed with 1 as root. The preorder traversal of MST is 1-2-4-3. Adding 1 at the end gives 1-2-4-3-1 which is the output of this algorithm.





In this case, the approximate algorithm produces the optimal tour, but it may not produce optimal tour in all cases.

**How is this algorithm 2-approximate?**

The cost of the output produced by the above algorithm is never more than twice the cost of best possible output. Let us see how is this guaranteed by the above algorithm.

Let us define a term ***full walk*** to understand this. A full walk is lists all vertices when they are first visited in preorder, it also list vertices when they are returned after a subtree is visited in preorder. The full walk of above tree would be 1-2-1-4-1-3-1.

Following are some important facts that prove the 2-approximateness.

1. The cost of best possible Travelling Salesman tour is never less than the cost of MST. (The definition of [MST](http://en.wikipedia.org/wiki/Minimum_spanning_tree) says, it is a minimum cost tree that connects all vertices).
2. The total cost of full walk is at most twice the cost of MST (Every edge of MST is visited at-most twice)
3. The output of the above algorithm is less than the cost of full walk. In above algorithm, we print preorder walk as output. In preorder walk, two or more edges of full walk are replaced with a single edge. For example, 2-1 and 1-4 are replaced by 1 edge 2-4. So if the graph follows triangle inequality, then this is always true.

Job Scheduling Algorithm:

====================

Job scheduling algorithm is applied to schedule the jobs on a single processor to maximize the profits.

The greedy approach of the job scheduling algorithm states that, “Given ‘n’ number of jobs with a starting time and ending time, they need to be scheduled in such a way that maximum profit is received within the maximum deadline”.

## Job Scheduling Algorithm

Set of jobs with deadlines and profits are taken as an input with the job scheduling algorithm and scheduled subset of jobs with maximum profit are obtained as the final output.

### Algorithm

Step1 − Find the maximum deadline value from the input set

of jobs.

Step2 − Once, the deadline is decided, arrange the jobs

in descending order of their profits.

Step3 − Selects the jobs with highest profits, their time

periods not exceeding the maximum deadline.

Step4 − The selected set of jobs are the output.

### Examples

Consider the following tasks with their deadlines and profits. Schedule the tasks in such a way that they produce maximum profit after being executed −

| **S. No.** | 1 | 2 | 3 | 4 | 5 |
| --- | --- | --- | --- | --- | --- |
| **Jobs** | J1 | J2 | J3 | J4 | J5 |
| **Deadlines** | 2 | 2 | 1 | 3 | 4 |
| **Profits** | 20 | 60 | 40 | 100 | 80 |

**Step 1**

Find the maximum deadline value, dm, from the deadlines given.

dm = 4.

**Step 2**

Arrange the jobs in descending order of their profits.

| **S. No.** | 1 | 2 | 3 | 4 | 5 |
| --- | --- | --- | --- | --- | --- |
| **Jobs** | J4 | J5 | J2 | J3 | J1 |
| **Deadlines** | 3 | 4 | 2 | 1 | 2 |
| **Profits** | 100 | 80 | 60 | 40 | 20 |

The maximum deadline, dm, is 4. Therefore, all the tasks must end before 4.

Choose the job with highest profit, J4. It takes up 3 parts of the maximum deadline.

Therefore, the next job must have the time period 1.

Total Profit = 100.

**Step 3**

The next job with highest profit is J5. But the time taken by J5 is 4, which exceeds the deadline by 3. Therefore, it cannot be added to the output set.

**Step 4**

The next job with highest profit is J2. The time taken by J5 is 2, which also exceeds the deadline by 1. Therefore, it cannot be added to the output set.

**Step 5**

The next job with higher profit is J3. The time taken by J3 is 1, which does not exceed the given deadline. Therefore, J3 is added to the output set.

Total Profit: 100 + 40 = 140

**Step 6**

Since, the maximum deadline is met, the algorithm comes to an end. The output set of jobs scheduled within the deadline are **{J4, J3}** with the maximum profit of **140**.