Analysis of Algorithm

Amypo Technologies Pvt Ltd

Concept:

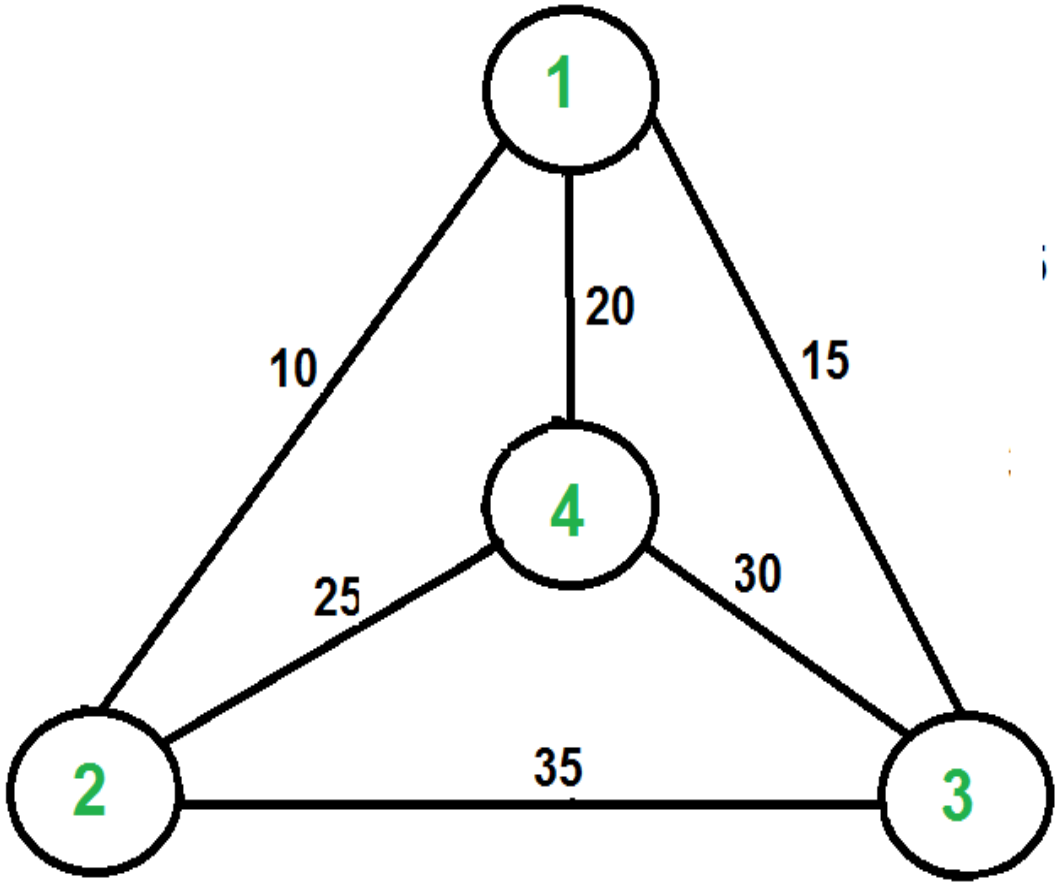
- Travelling Salesman Problem
- 0-1 Knapsack
- Longest Common Subsequence

Travelling Salesman Problem (TSP):

The Travelling Salesman Problem is a classic problem in computer science and optimization. It involves finding the shortest possible route that visits each city exactly once and returns to the origin city. This problem is NP-hard, meaning that as the number of cities increases, finding the optimal solution becomes increasingly difficult and time-consuming.

Algorithm

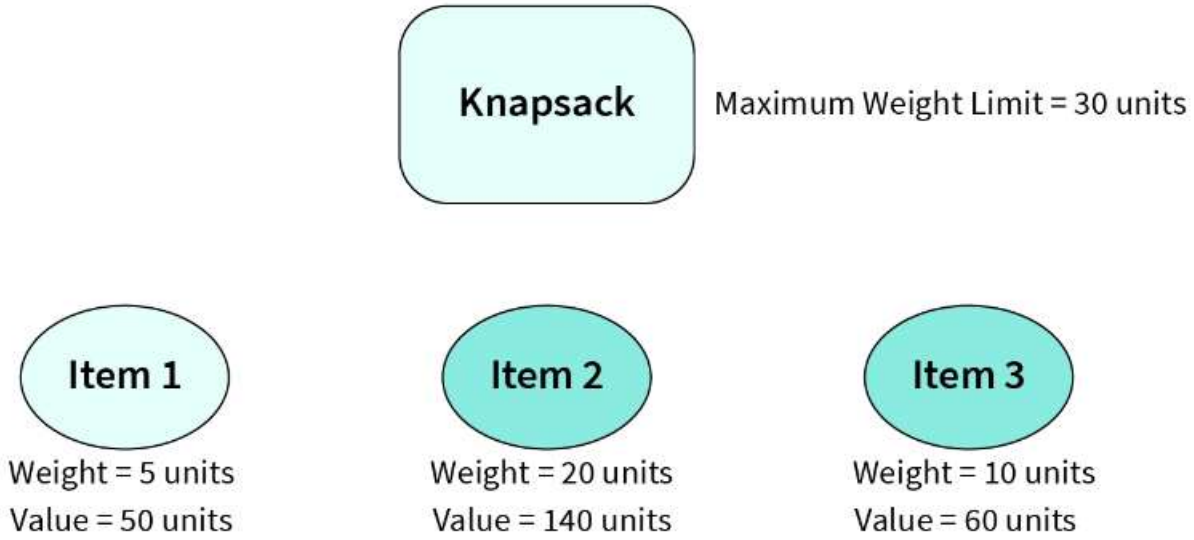
- Travelling salesman problem takes a graph G {V, E} as an input and declare another graph as the output (say G') which will record the path the salesman is going to take from one node to another.
- The algorithm begins by sorting all the edges in the input graph G from the least distance to the largest distance.
- The first edge selected is the edge with least distance, and one of the two vertices (say A and B) being the origin node (say A).
- Then among the adjacent edges of the node other than the origin node (B), find the least cost edge and add it onto the output graph.
- Continue the process with further nodes making sure there are no cycles in the output graph and the path reaches back to the origin node A.
- However, if the origin is mentioned in the given problem, then the solution must always start from that node only. Let us look at some example problems to understand this better.



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 know solution for this problem. The following are different solutions for the traveling salesman problem.

0-1 Knapsack

- The knapsack problem states that – given a set of items, holding weights and profit values, one must determine the subset of the items to be added in a knapsack such that, the total weight of the items must not exceed the limit of the knapsack and its total profit value is maximum.



- The knapsack problem states that − given a set of items, holding weights and profit values, one must determine the subset of the items to be added in a knapsack such that, the total weight of the items must not exceed the limit of the knapsack and its total profit value is maximum.

Maximum Weights →

Items with Weights and Profits ↓

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1 (1, 2)	0	1	1	1	1	1	1	1	1
2 (3,4)	0	1	1	4	6	6	6	6	6
3 (5,7)	0	1	1	4	6	7	9	9	11
4 (7,10)	0	1	1	4	6	7	9	10	(12)

Longest Common Subsequence

A longest common subsequence (LCS) is the longest subsequence common to all sequences in a set of sequences (often just two sequences). It differs from the longest common substring: unlike substrings, subsequences are not required to occupy consecutive positions within the original sequences.

Old revision	New revision
2010-10-31 17:10:03 by admin	2010-10-31 17:31:03 by admin
this is **some** **text** **that** **will** **be** changed	this is **the** changed **text**

Longest Common Subsequence (LCS) means you will be given two strings/patterns/sequences of objects. Among these two sequences/strings, you need to find the longest subsequence of elements in the same order present in both strings or patterns.

R	G	B	G	A	R	G	A
B	G	R	A	R	G		

The LCS is “RARG” with length of 4

Example

For example, there are two strings provided.

Let’s assume that,

Pattern_1 = “RGBGARGA”

Pattern_2 = “BGRARG”

•From pattern_1, sequences can be produced like “RGB”, “RGGA”, ”RGAR”. For creating a sequence, you need to maintain the relative position of each character in the string.

•From pattern_2, we can produce sequences like “BGR”, ”BRAG”, ”RARG”. Sequences can be produced as long as they maintain the original string’s relative position.

We have two options to find the Longest Common Subsequence from the given two sequences or arrays.

- Naive method
- Dynamic Programming Solution: Longest Common Subsequence is also known as LCS.

A naive Solution has larger time complexity and is not the optimal solution. Using Dynamic Programming Solution (DP), we overcome the complexity problem.