Assignment No. 7

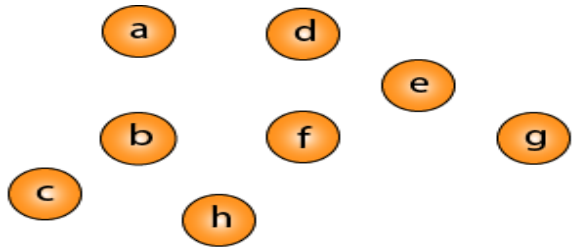
Aim: To implement the travelling salesman problem using C++.

Theory:

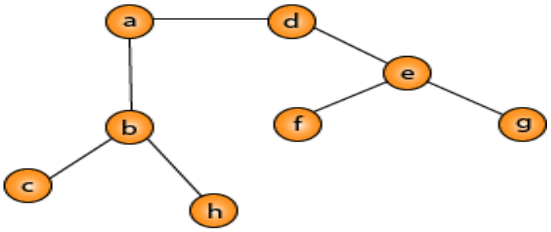
- **Travelling Salesman Problem:**

In the traveling salesman Problem, a salesman must visits n cities. We can say that salesman wishes to make a tour or Hamiltonian cycle, visiting each city exactly once and finishing at the city he starts from. There is a non-negative cost c (i, j) to travel from the city i to city j. The goal is to find a tour of minimum cost. We assume that every two cities are connected. Such problems are called Traveling-salesman problem (TSP). We can model the cities as a complete graph of n vertices, where each vertex represents a city. It can be shown that TSP is NPC. If we assume the cost function c satisfies the triangle inequality, then we can use the following approximate algorithm.

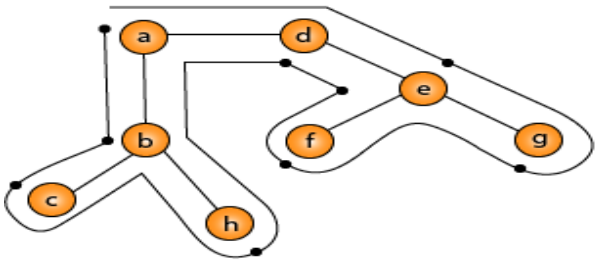
- **Example:**



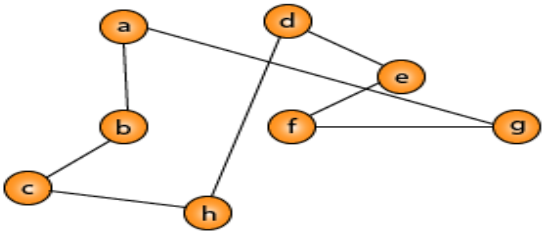
(1) A given set of points



(2) MST T



(3) Full tree walk on T.



(4) A preorder sequence gives a tour H.

Intuitively, Approx-TSP first makes a full walk of MST T, which visits each edge exactly two times. To create a Hamiltonian cycle from the full walk, it bypasses some vertices (which corresponds to making a shortcut)

Algorithm for travelling salesman problem:

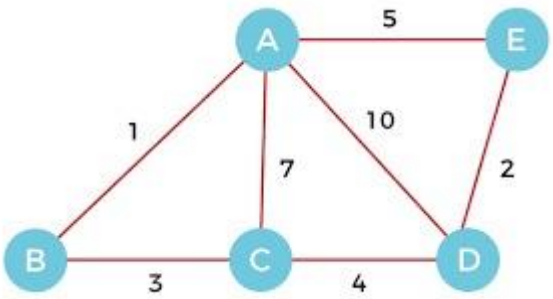
1. Approx-TSP (G= (V, E))
2. {
3. 1. Compute a MST T of G;
4. 2. Select any vertex r is the root of the tree;

5. 3. Let L be the list of vertices visited in a preorder tree walk of T;
6. 4. Return the Hamiltonian cycle H that visits the vertices in the order L;
7. }

- **Approach using Dynamic Programming:**

The dynamic programming or DP method guarantees finding the best answer to TSP. However, its time complexity would exponentially increase with the number of cities. The time complexity with the DP method asymptotically equals $N^2 \times 2^N$ where N is the number of cities. To give you a hint of how this time complexity increases, let me share my experiments. TSP with 10 cities can be solved by a DP method in almost 0.2 seconds using intel core i7. This number increases to almost 13 seconds (~60 times greater) with 15 cities. That is, the time complexity significantly increases even with a small increment in the number of cities. To optimize the DP method, I could have used the memorization technique. However, the memorization technique with a large number of cities needs a $2^N \times 2^N$ matrix that cannot be easily handled in memory.

Example 1:



Solution:

Program:

Output:

Conclusion: