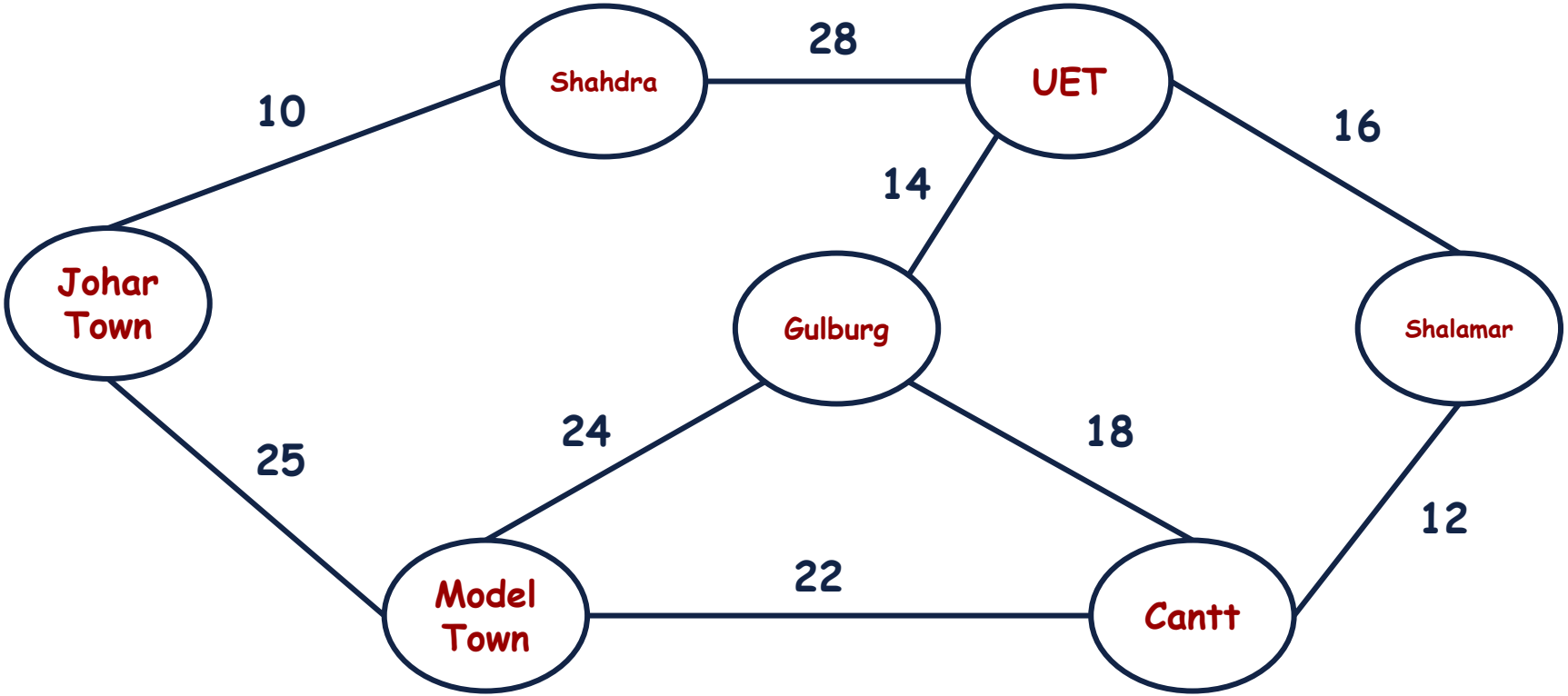
Minimum Spanning Tree

Graphs: Problem

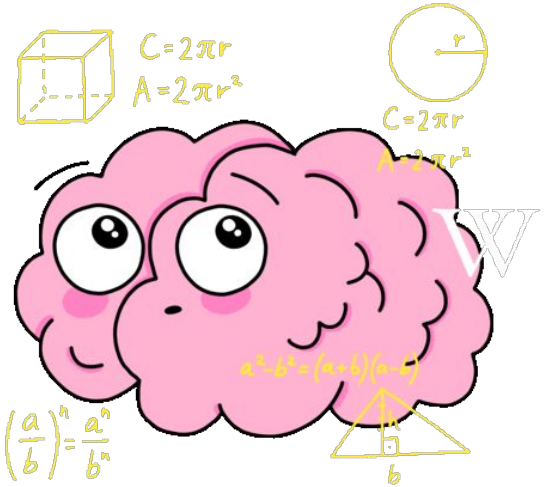
You are working in the PTCL broadband company. You have to provide Internet connection to the new neighborhood. Cables should be placed underground so that they are not damaged by environmental Conditions. Since you are a problem solver, you have made a graph of the neighborhood and cost of placing the internet cables is mentioned.
Now, your task is to provide internet facility to all the neighborhood with minimum cost.

Graphs: Problem



Graphs: Problem

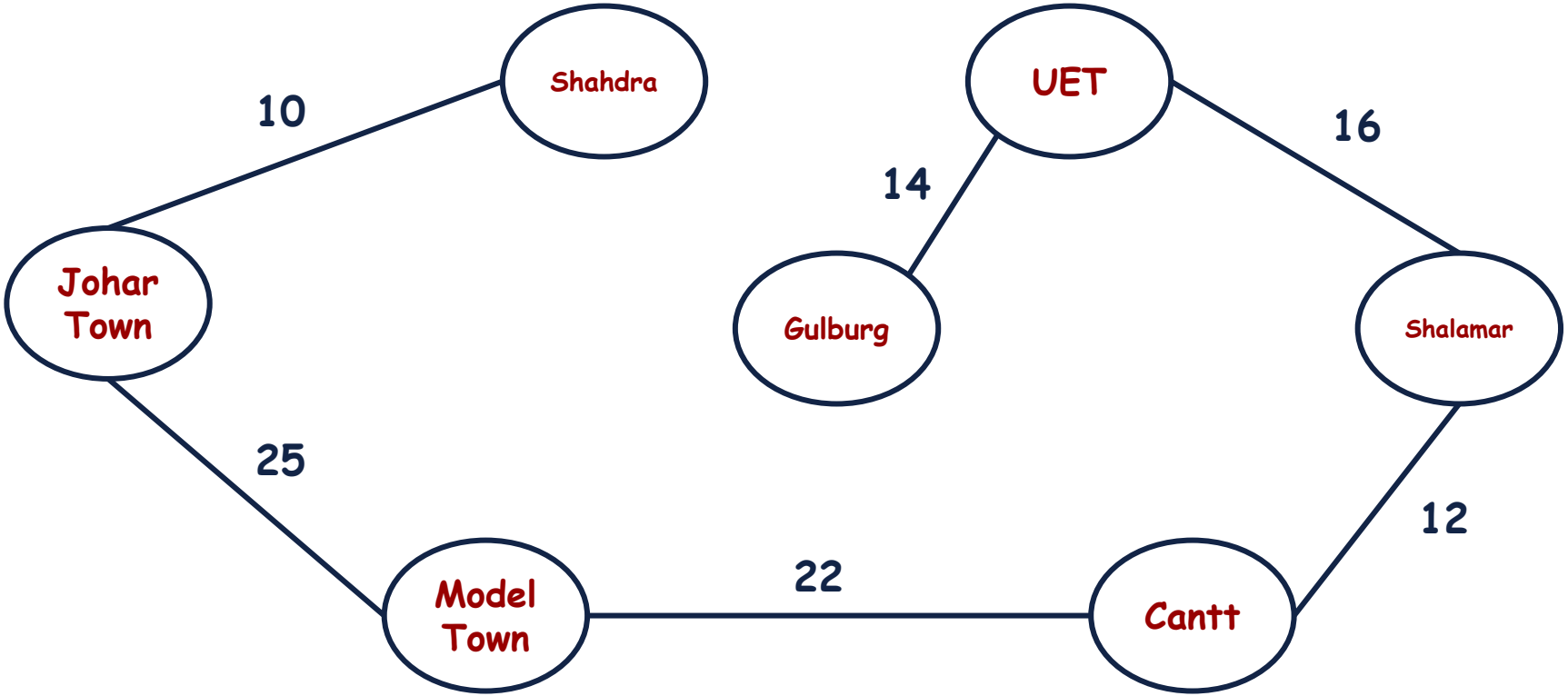
How can we do that?



Graphs: Problem

- One thing is clear that we have to include every Vertex.
- Now, we have to include only those edges that has no cycles but still connects every vertex.
- And we have to choose those edges that have the minimum weight.

Graphs: Problem

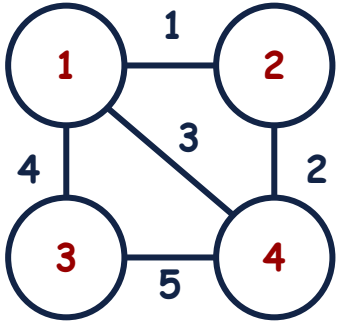


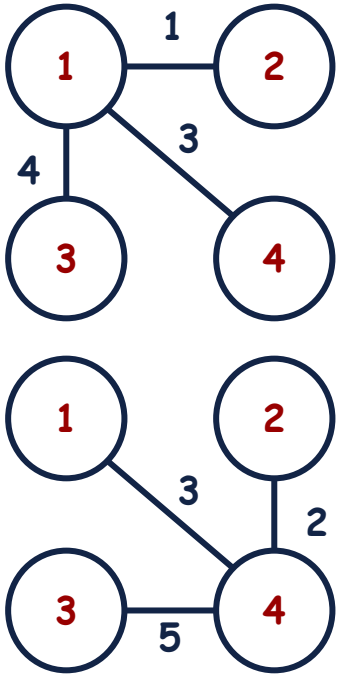
Graphs: Spanning Tree

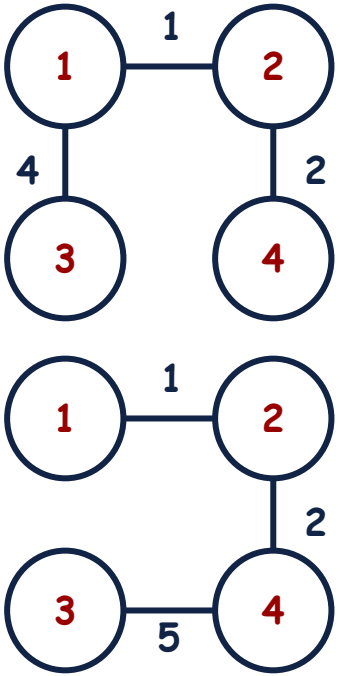
This is a spanning tree, which is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges.

Graphs: Spanning Tree

Let's understand the Spanning Trees with a simple example first.

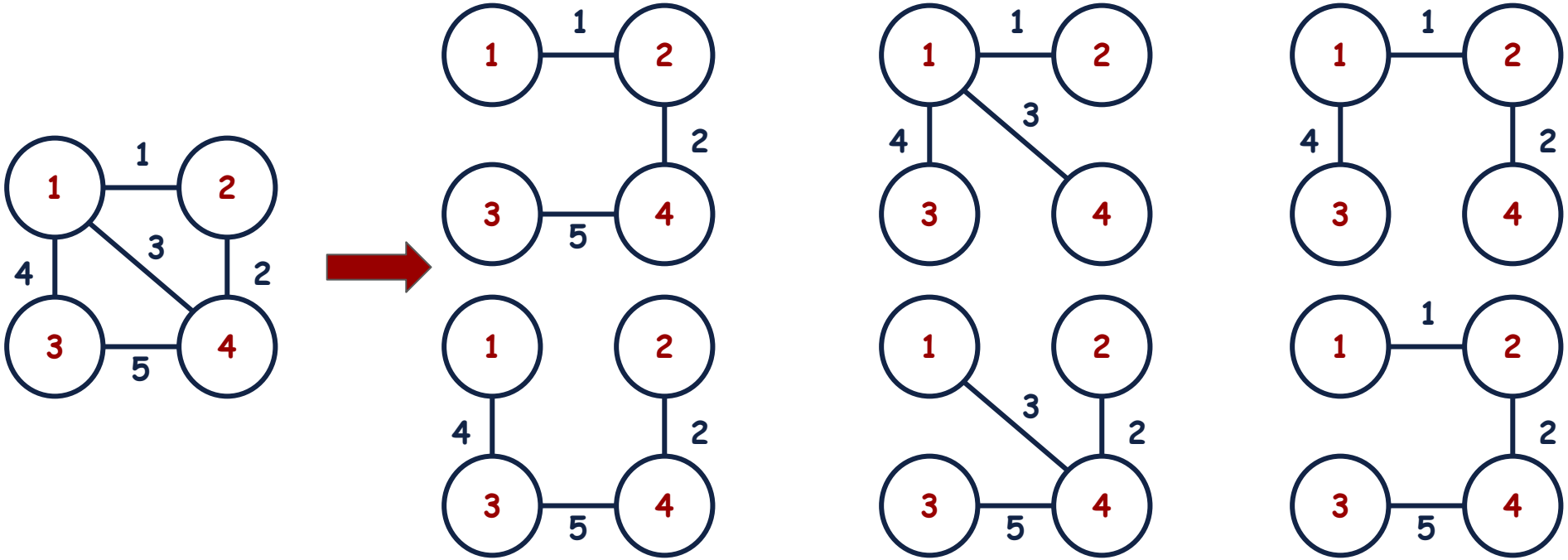






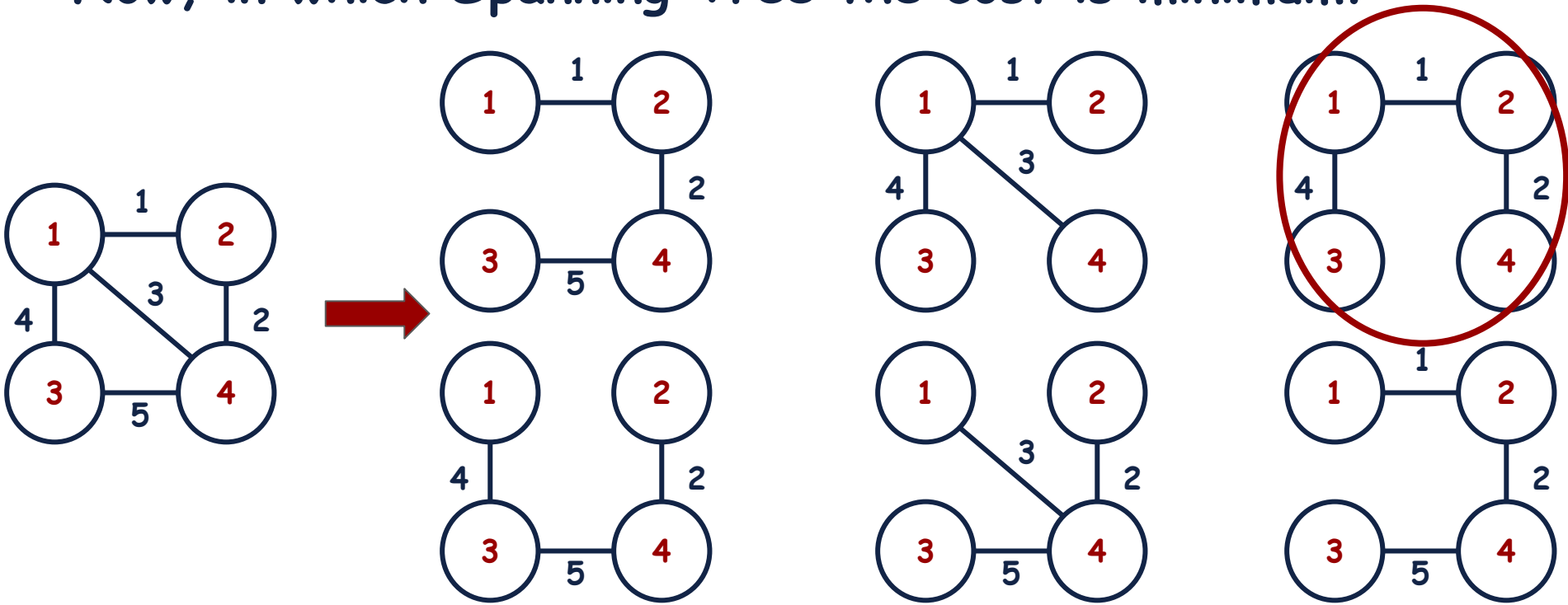
Graphs: Spanning Tree

Now, in which Spanning Tree the cost is minimum?



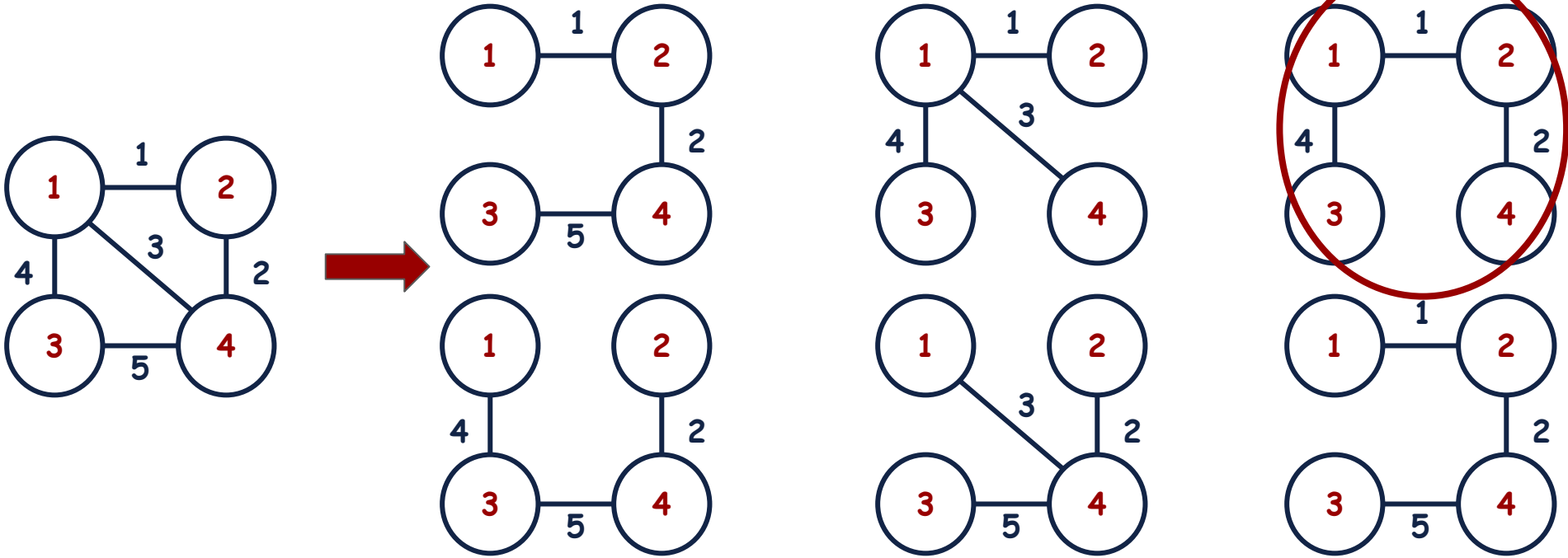
Graphs: Spanning Tree

Now, in which Spanning Tree the cost is minimum?



Graphs: Minimum Spanning Tree

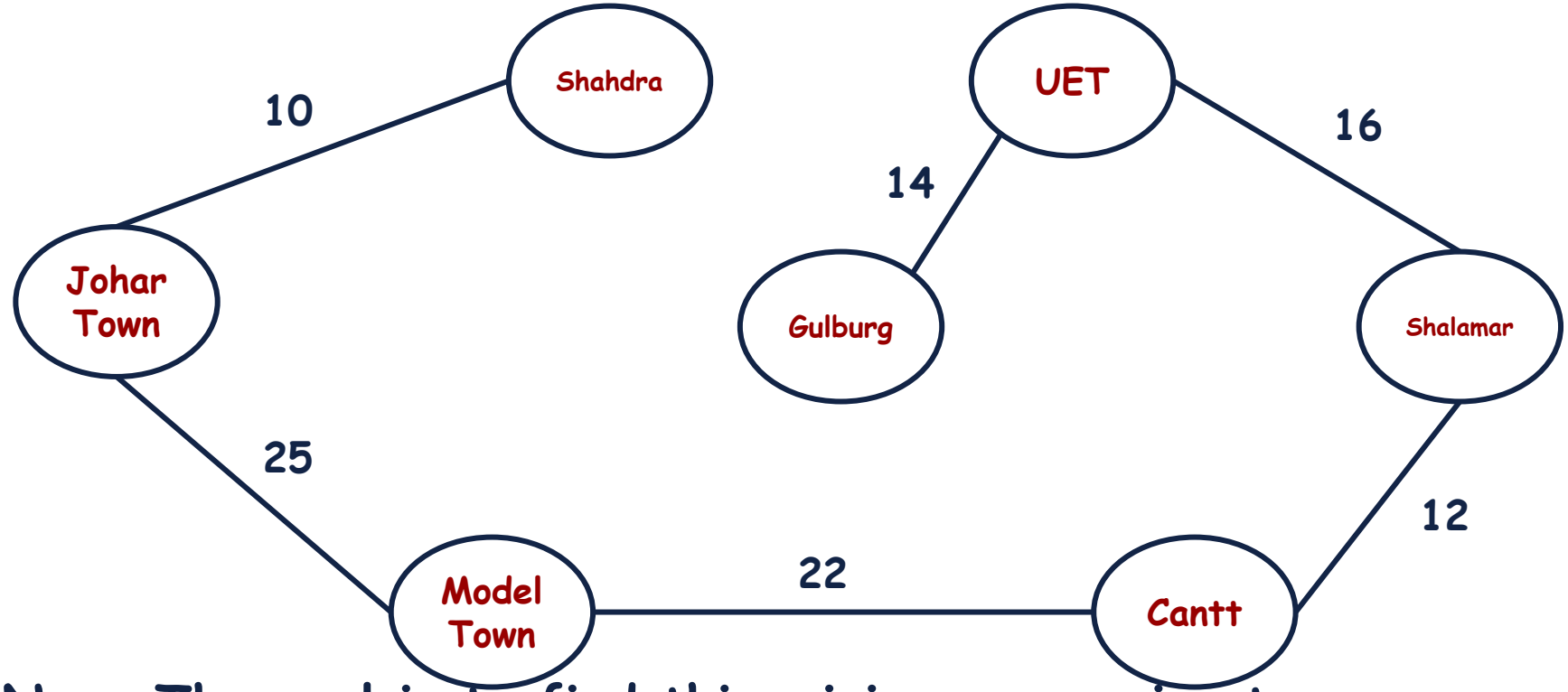
A MST is a spanning tree in which the sum of the weight of the edges is as minimum as possible.



Graphs: Minimum Spanning Tree

Since, we are choosing the edges having the minimum weight in our PTCL broadband Example, therefore it is called **Minimum Spanning Tree**.

Graphs: Minimum Spanning Tree



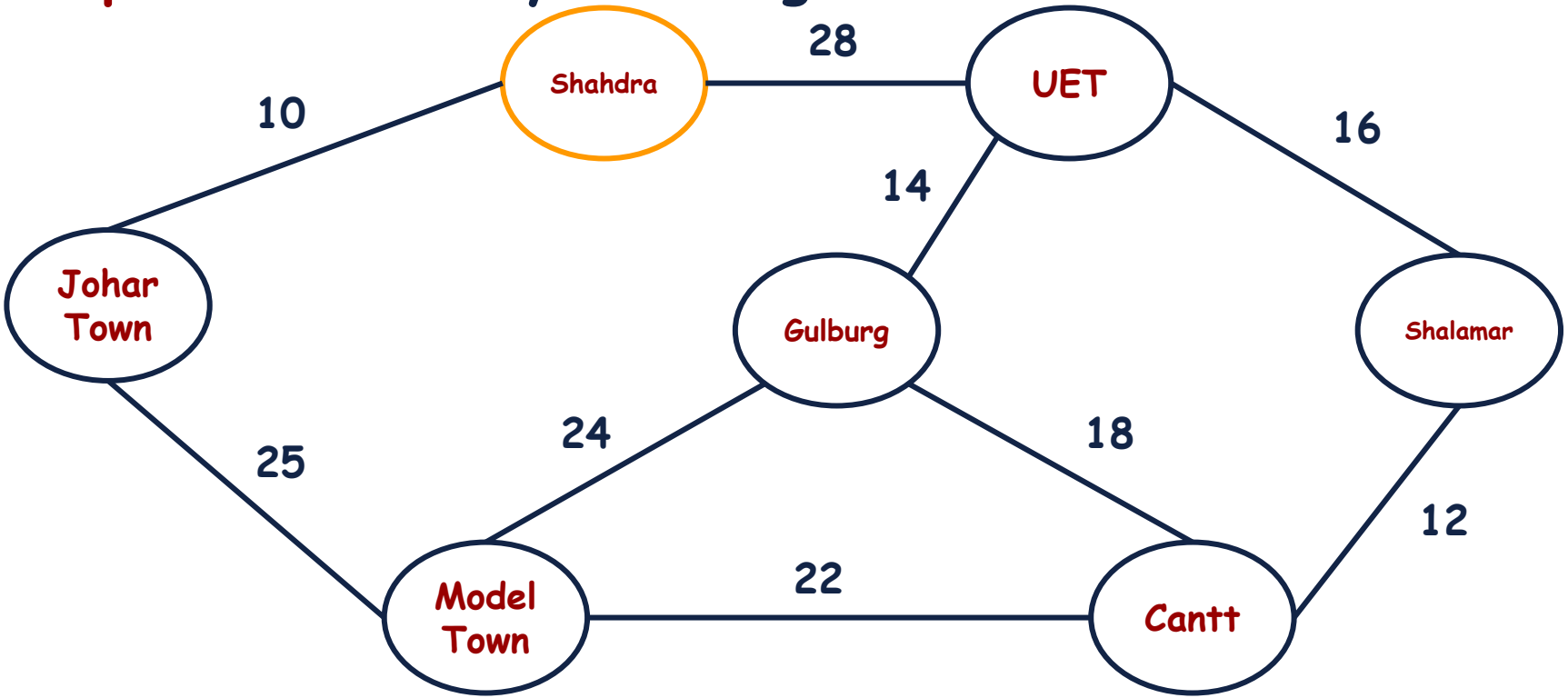
Now, The goal is to find this minimum spanning tree.

Minimum Spanning Tree

What is the Algorithm to find the MST?

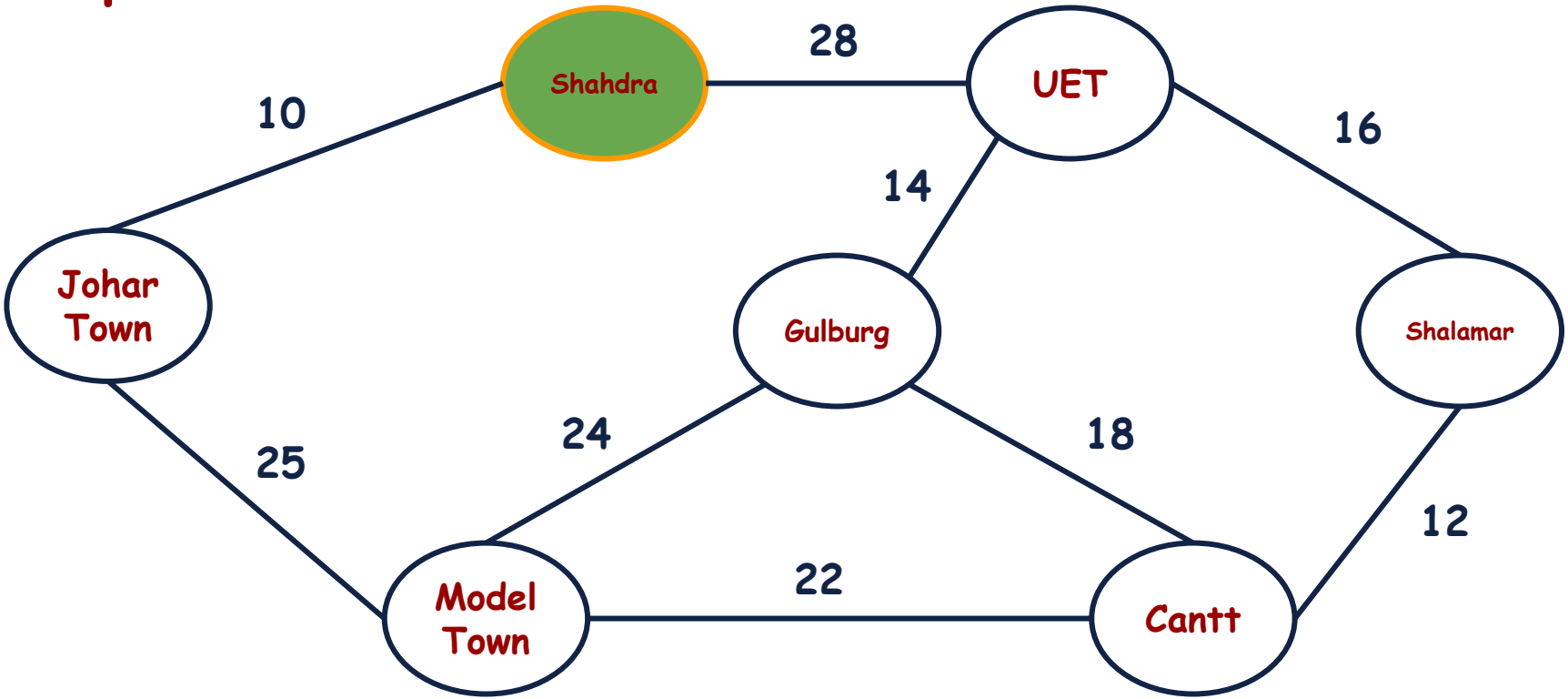
Minimum Spanning Tree: Solution

Step 1: Choose any Starting Vertex.



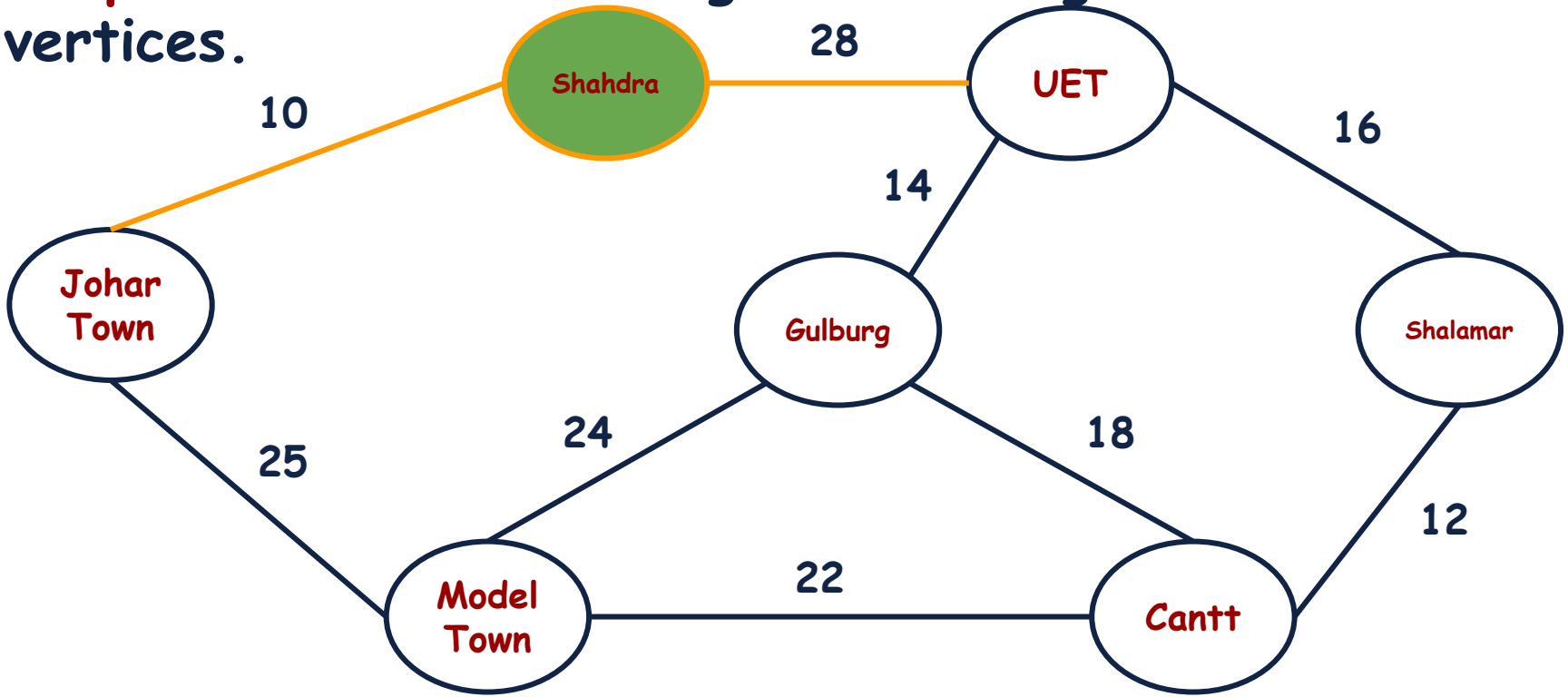
Minimum Spanning Tree: Solution

Step 2: Mark it as Visited.



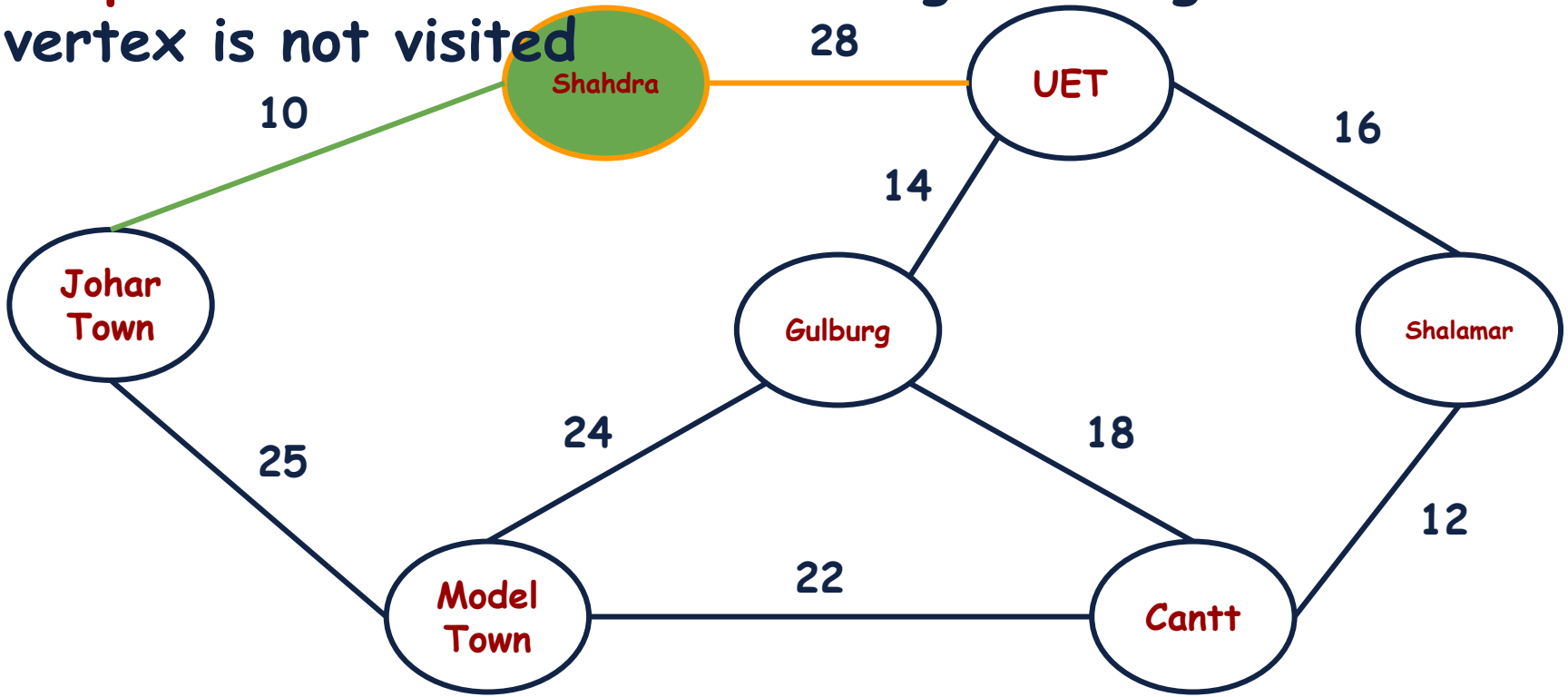
Minimum Spanning Tree: Solution

Step 3: Check all the edges extending from the visited vertices.



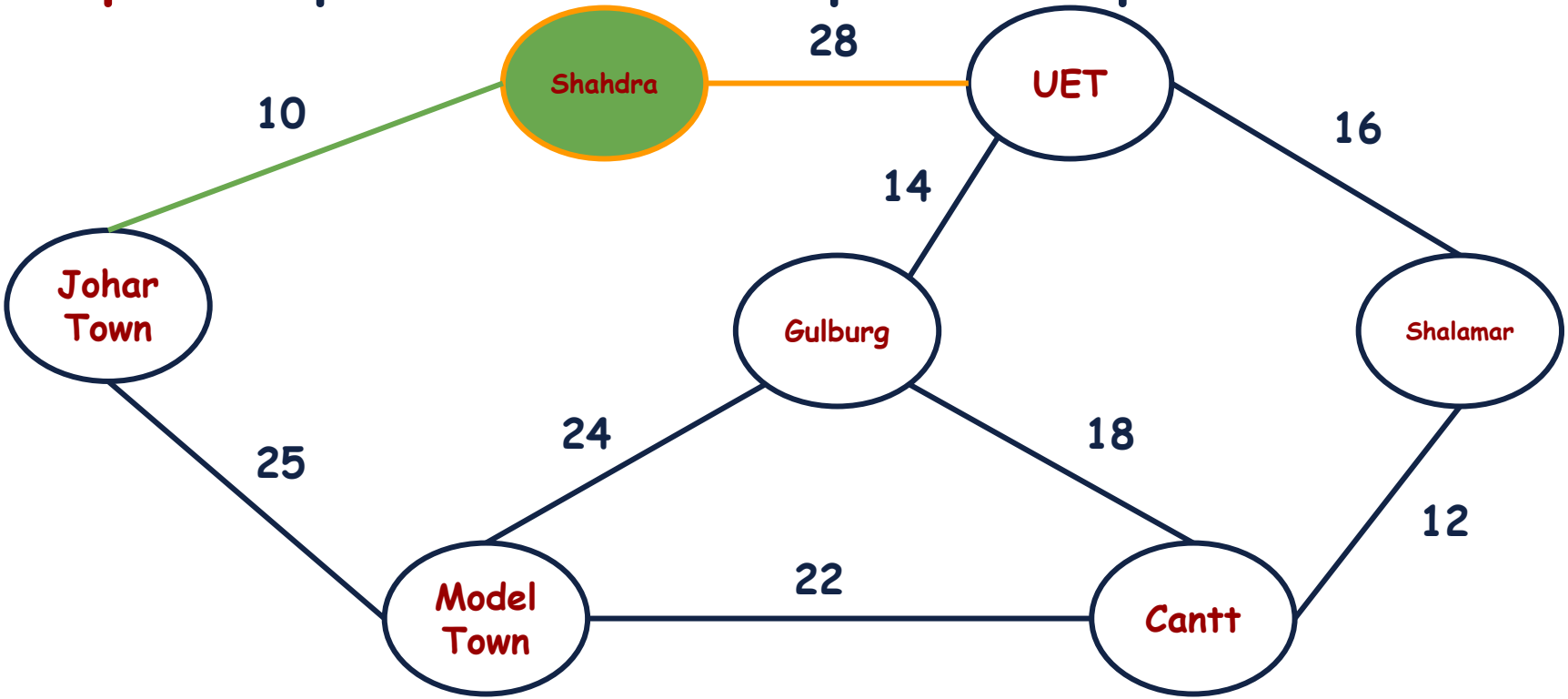
Minimum Spanning Tree: Solution

Step 4: Choose the minimum weighted edge whose vertex is not visited



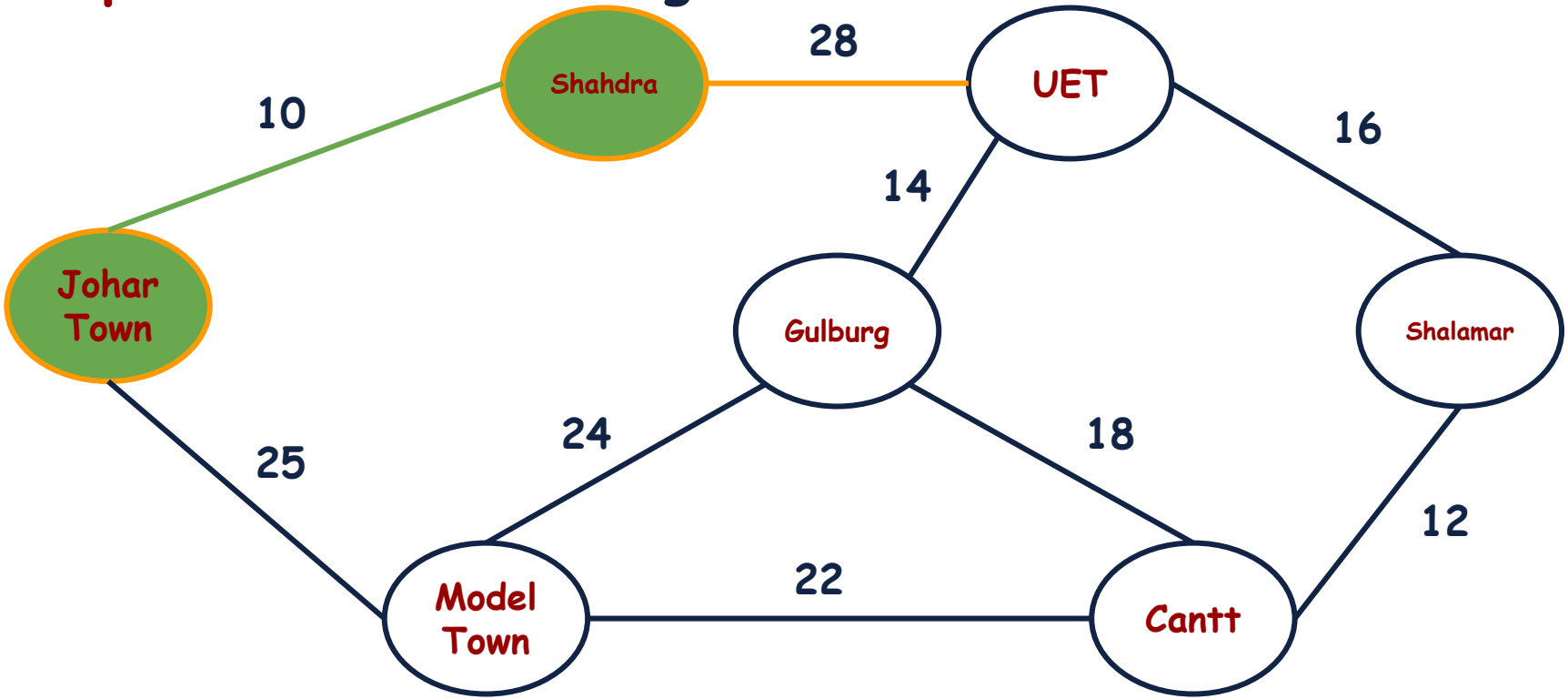
Minimum Spanning Tree: Solution

Step 5: Repeat from the step 2 to step 4.



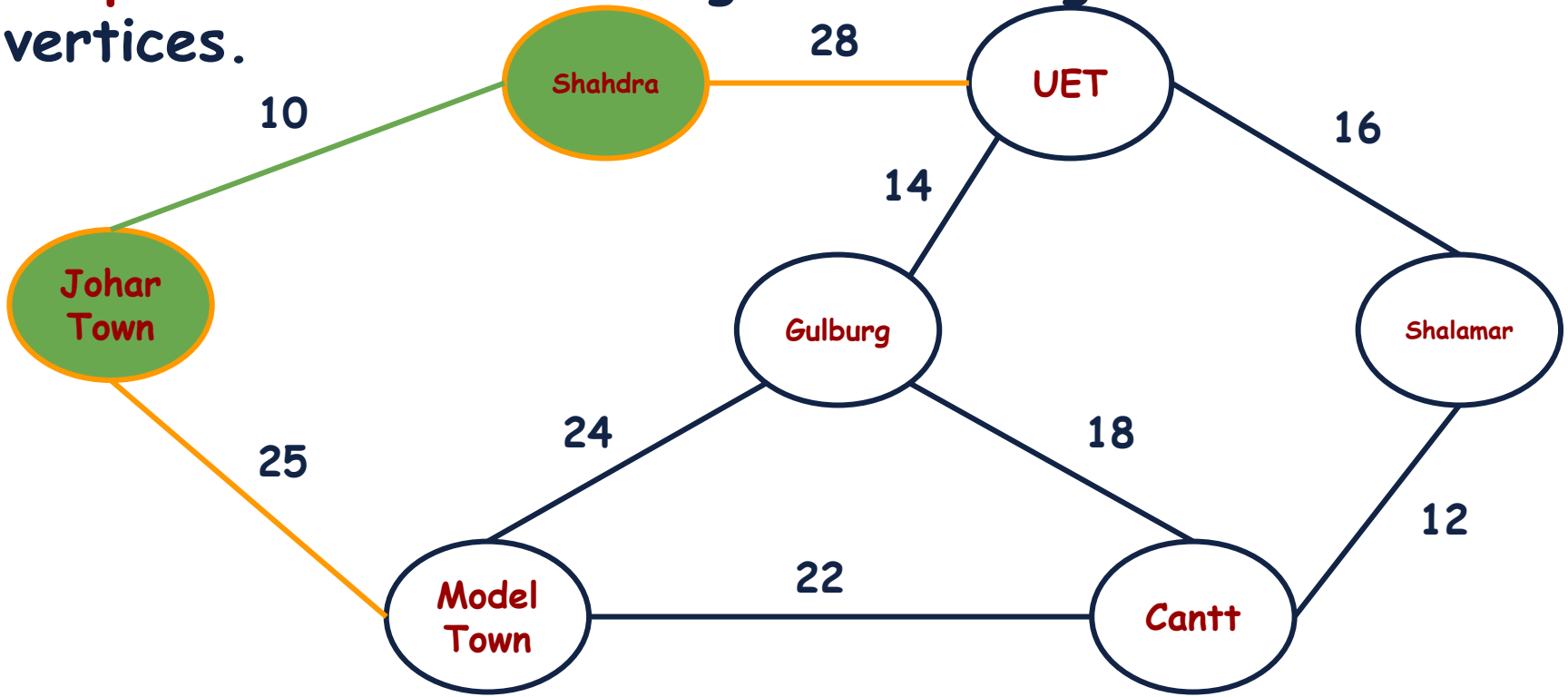
Minimum Spanning Tree: Solution

Step 2: Marked the edge visited.



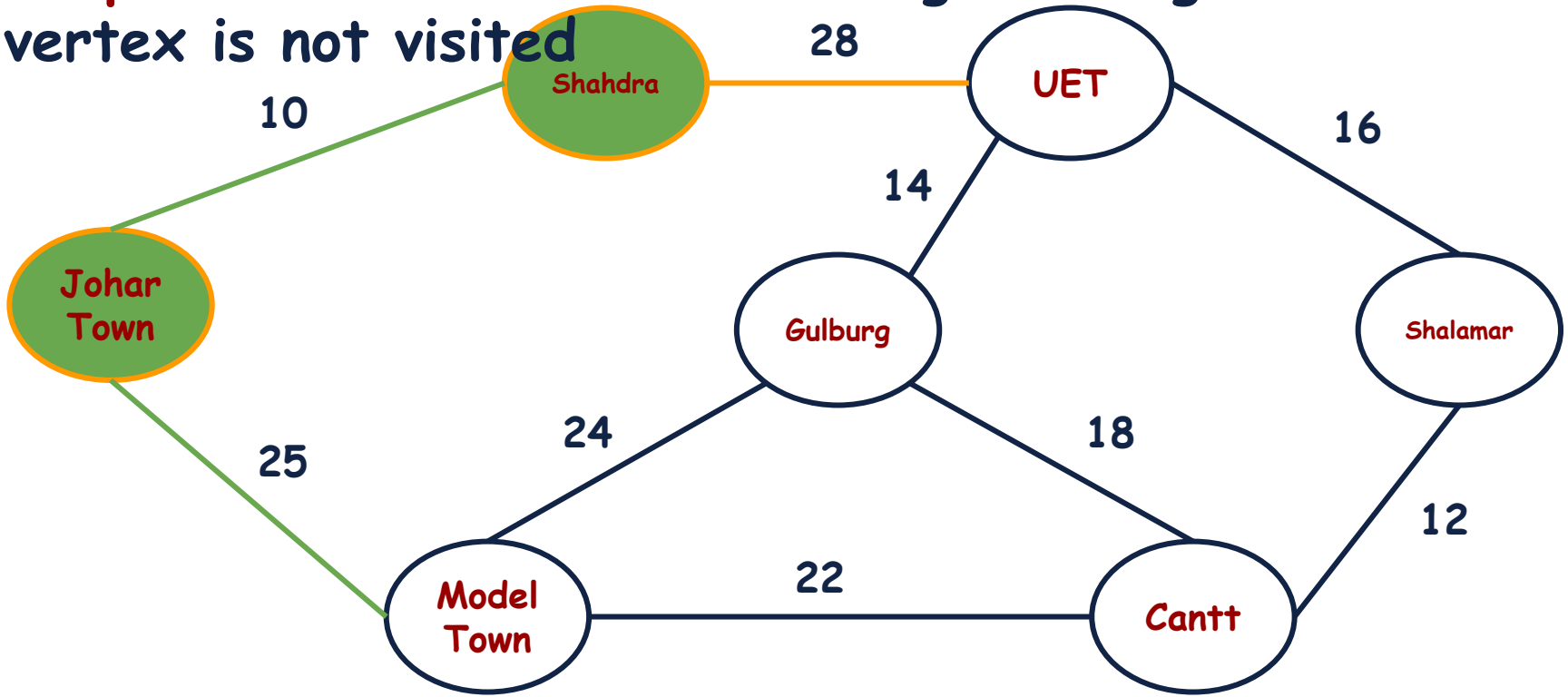
Minimum Spanning Tree: Solution

Step 3: Check all the edges extending from the visited vertices.



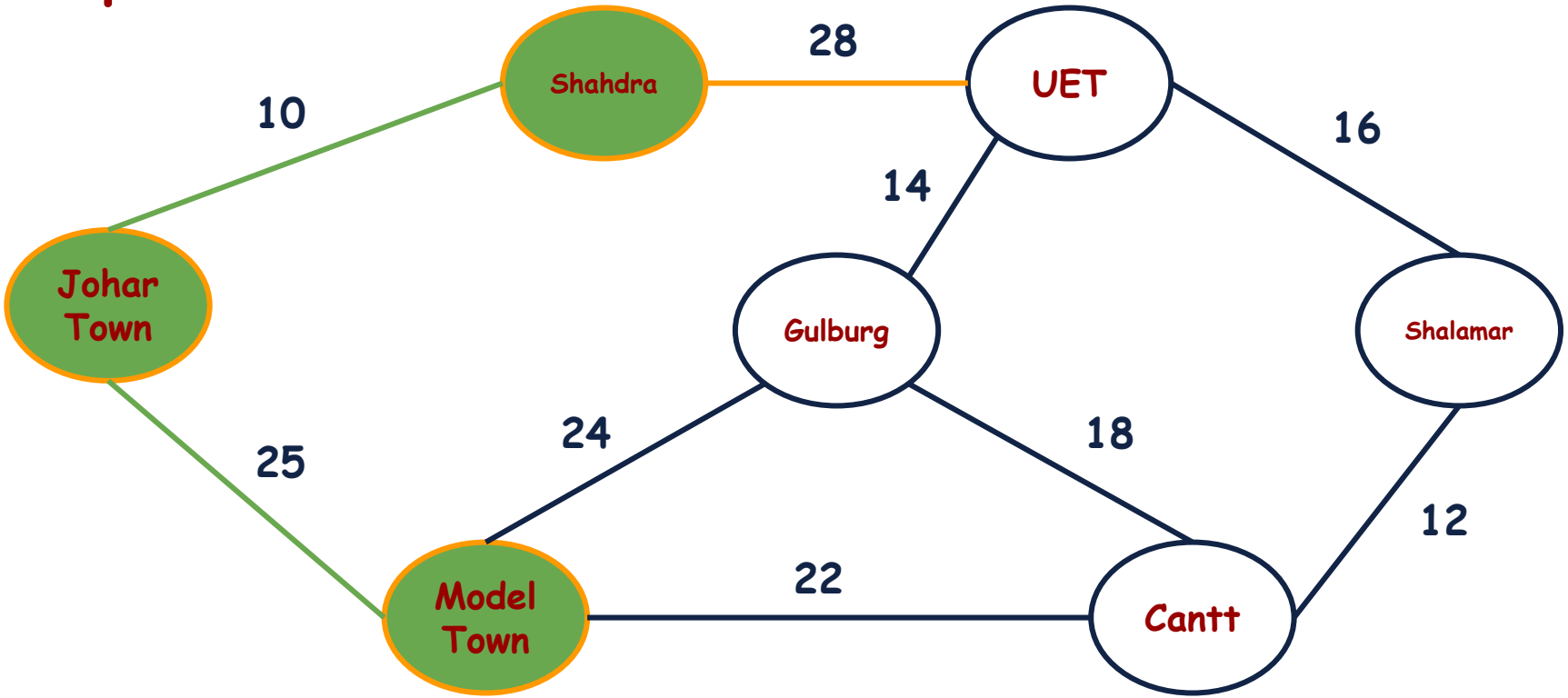
Minimum Spanning Tree: Solution

Step 4: Choose the minimum weighted edge whose vertex is not visited



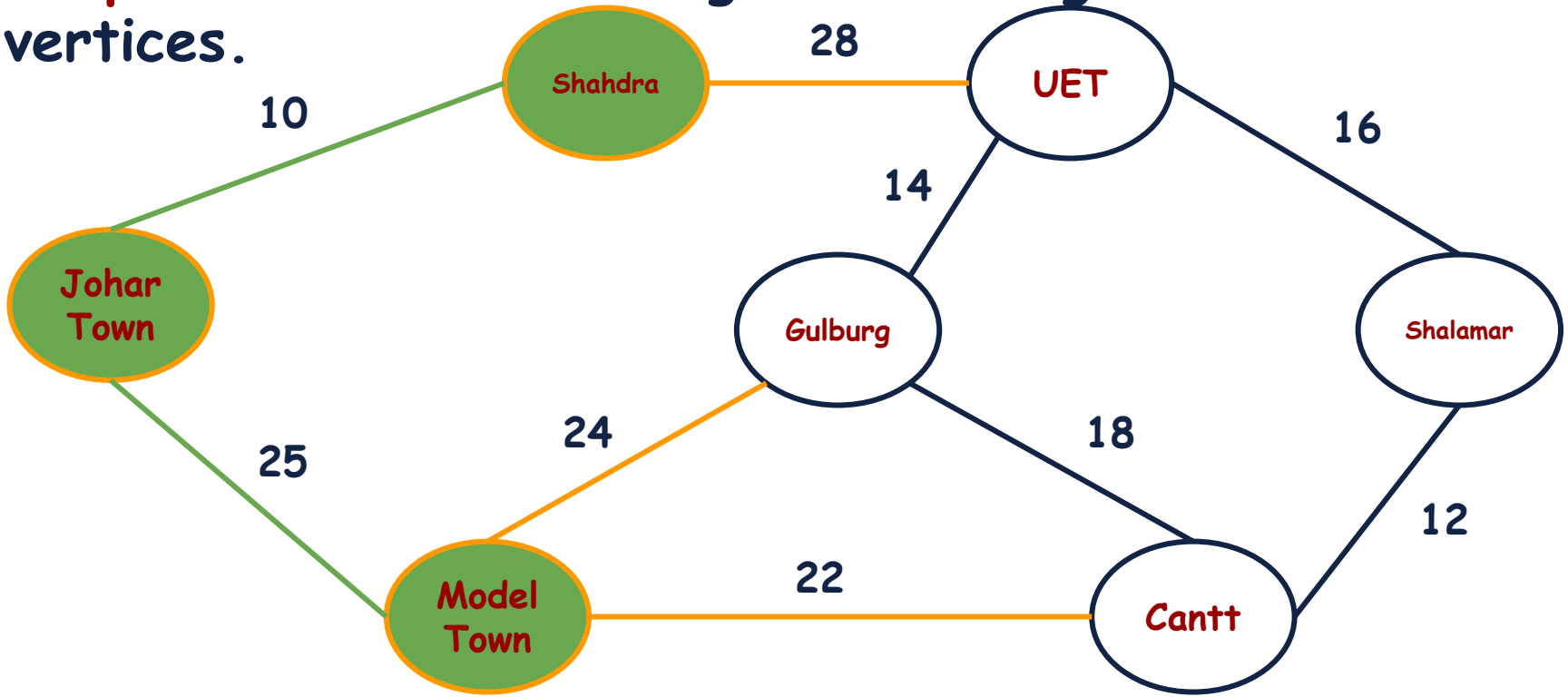
Minimum Spanning Tree: Solution

Step 2: Mark the vertex as visited.



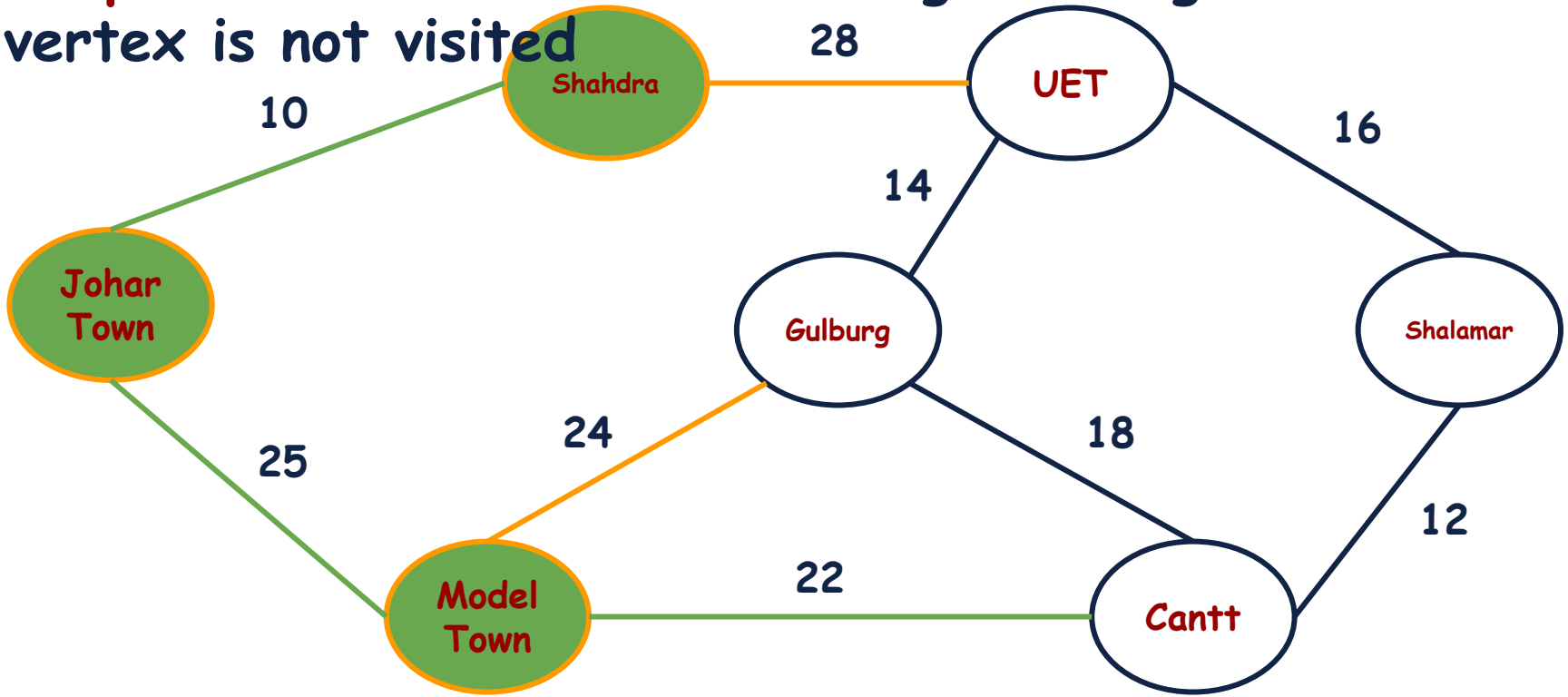
Minimum Spanning Tree: Solution

Step 3: Check all the edges extending from the visited vertices.



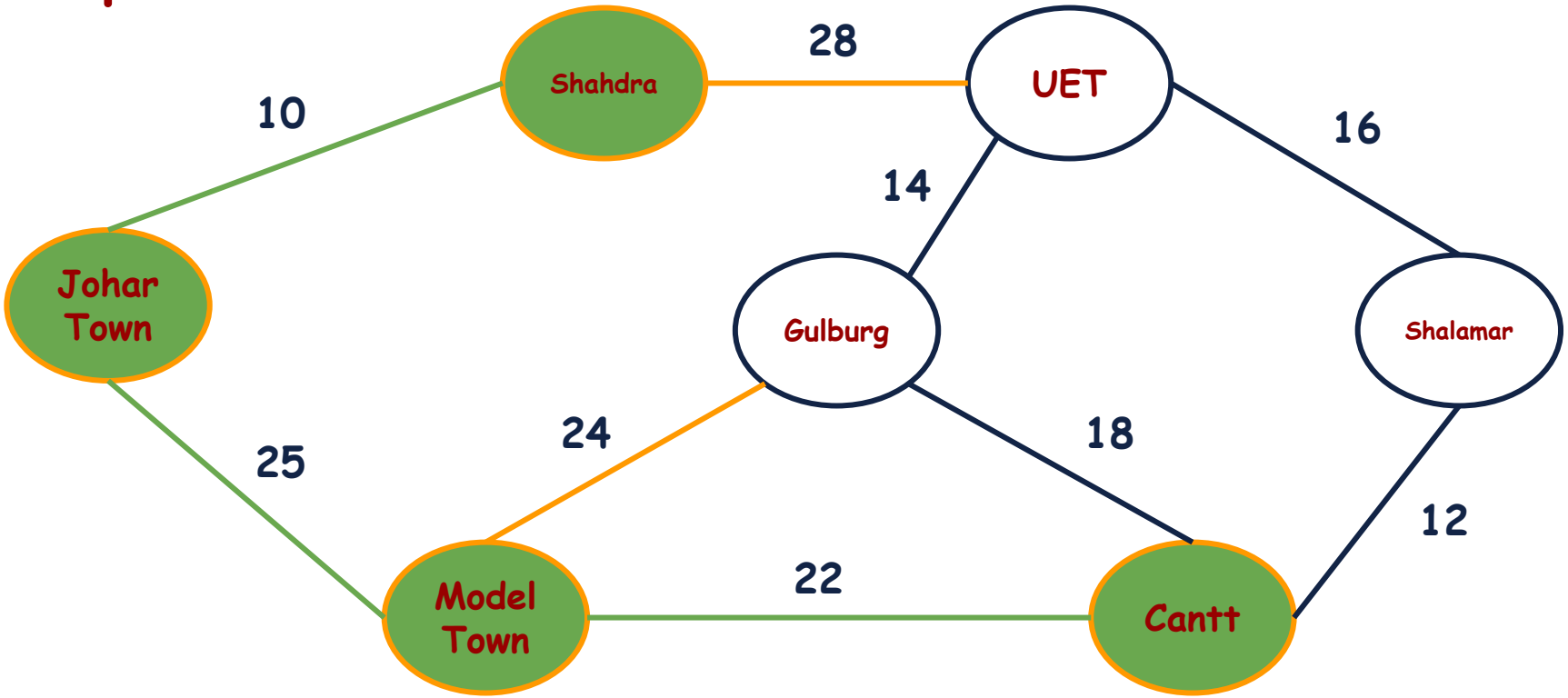
Minimum Spanning Tree: Solution

Step 4: Choose the minimum weighted edge whose vertex is not visited



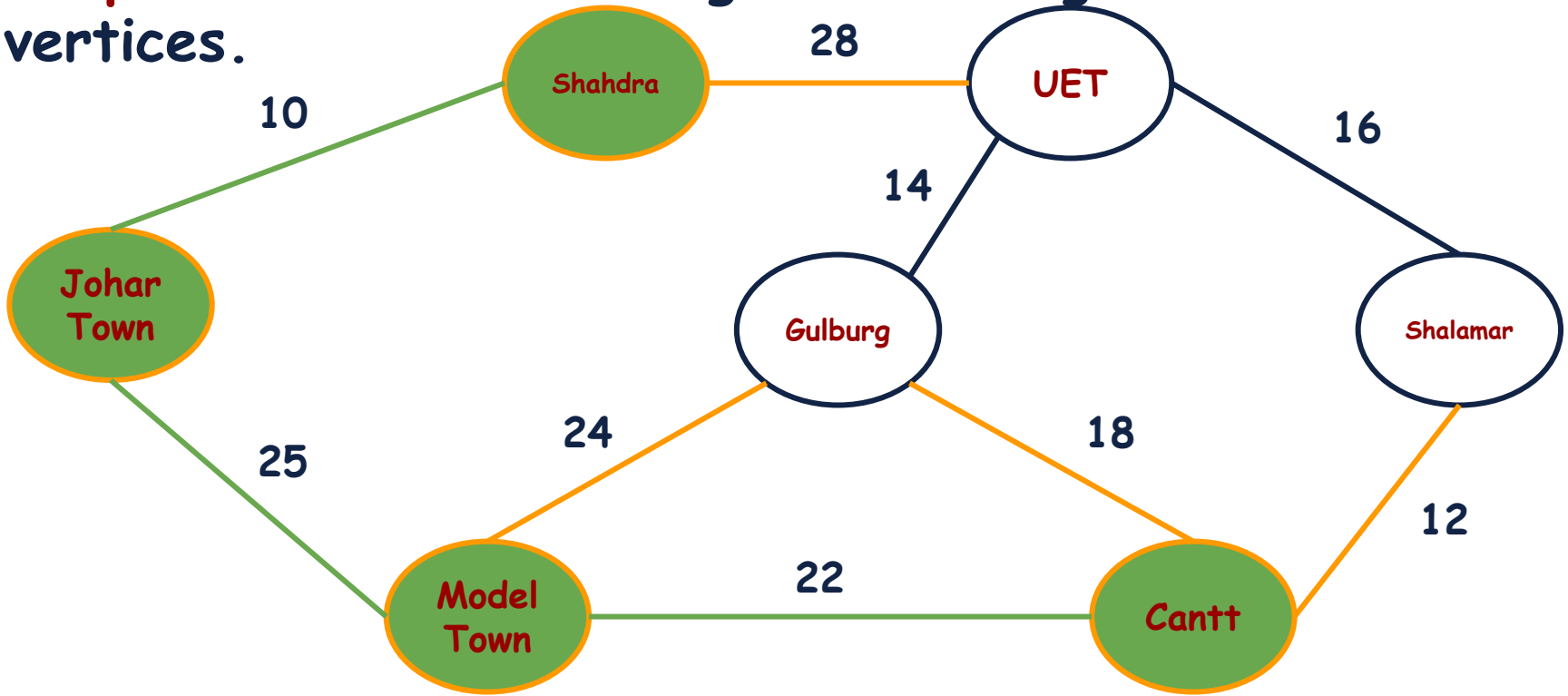
Minimum Spanning Tree: Solution

Step 2: Marked the vertex as Visited.



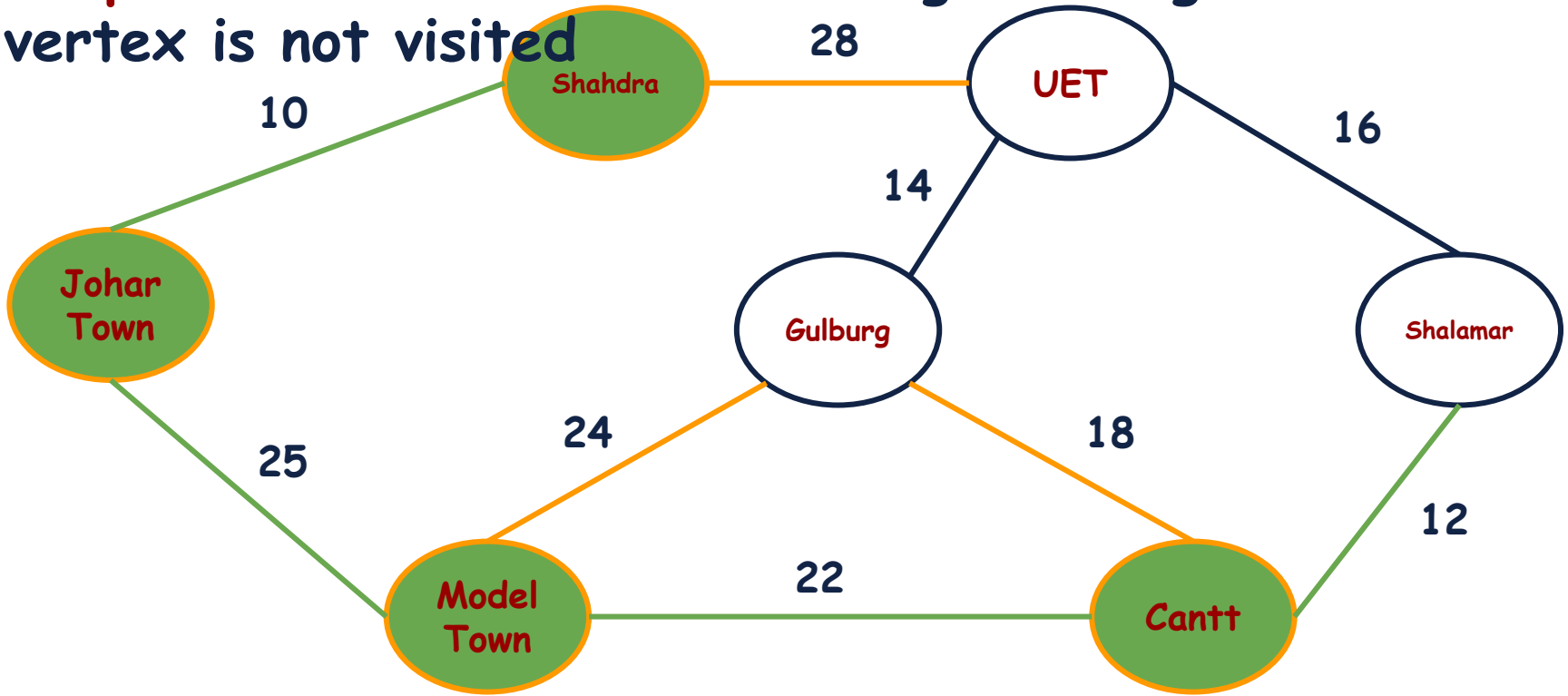
Minimum Spanning Tree: Solution

Step 3: Check all the edges extending from the visited vertices.



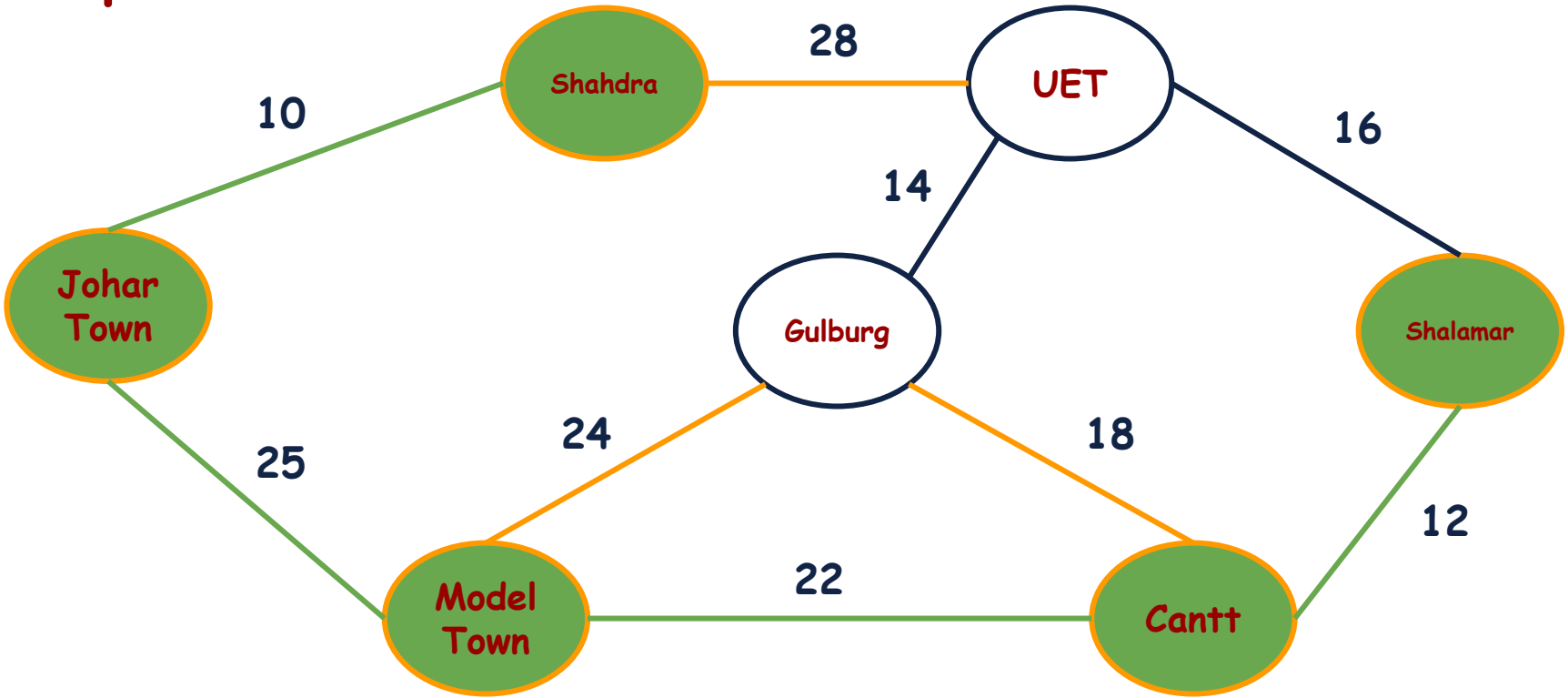
Minimum Spanning Tree: Solution

Step 4: Choose the minimum weighted edge whose vertex is not visited



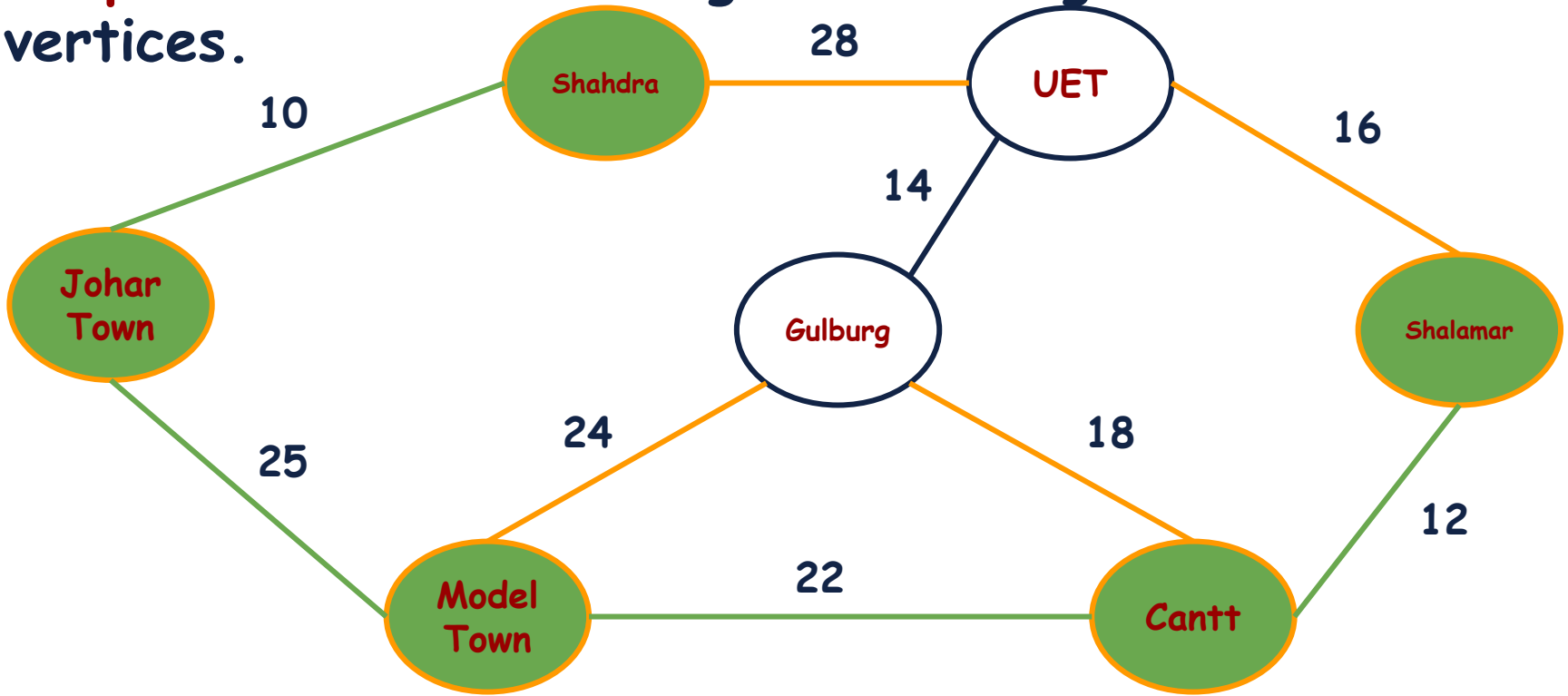
Minimum Spanning Tree: Solution

Step 2: Mark the node as visited.



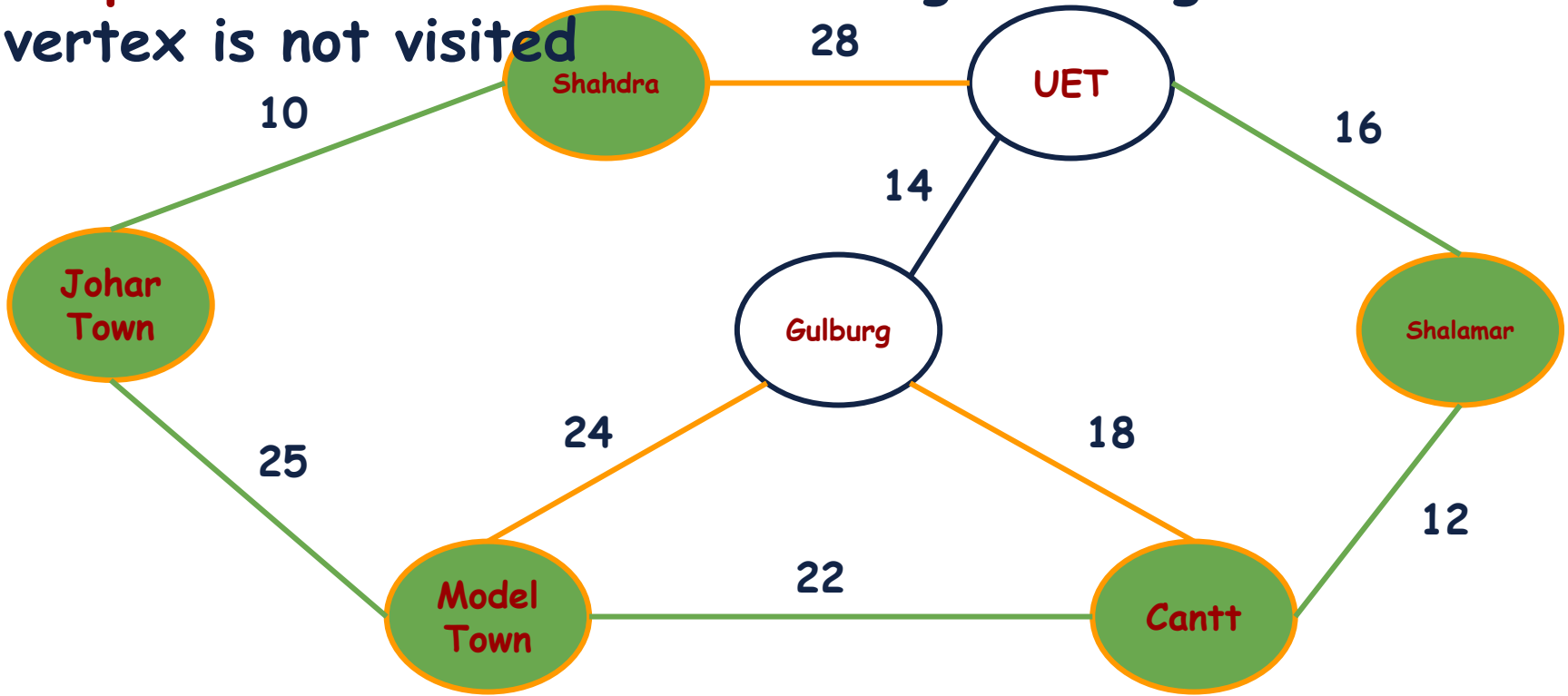
Minimum Spanning Tree: Solution

Step 3: Check all the edges extending from the visited vertices.



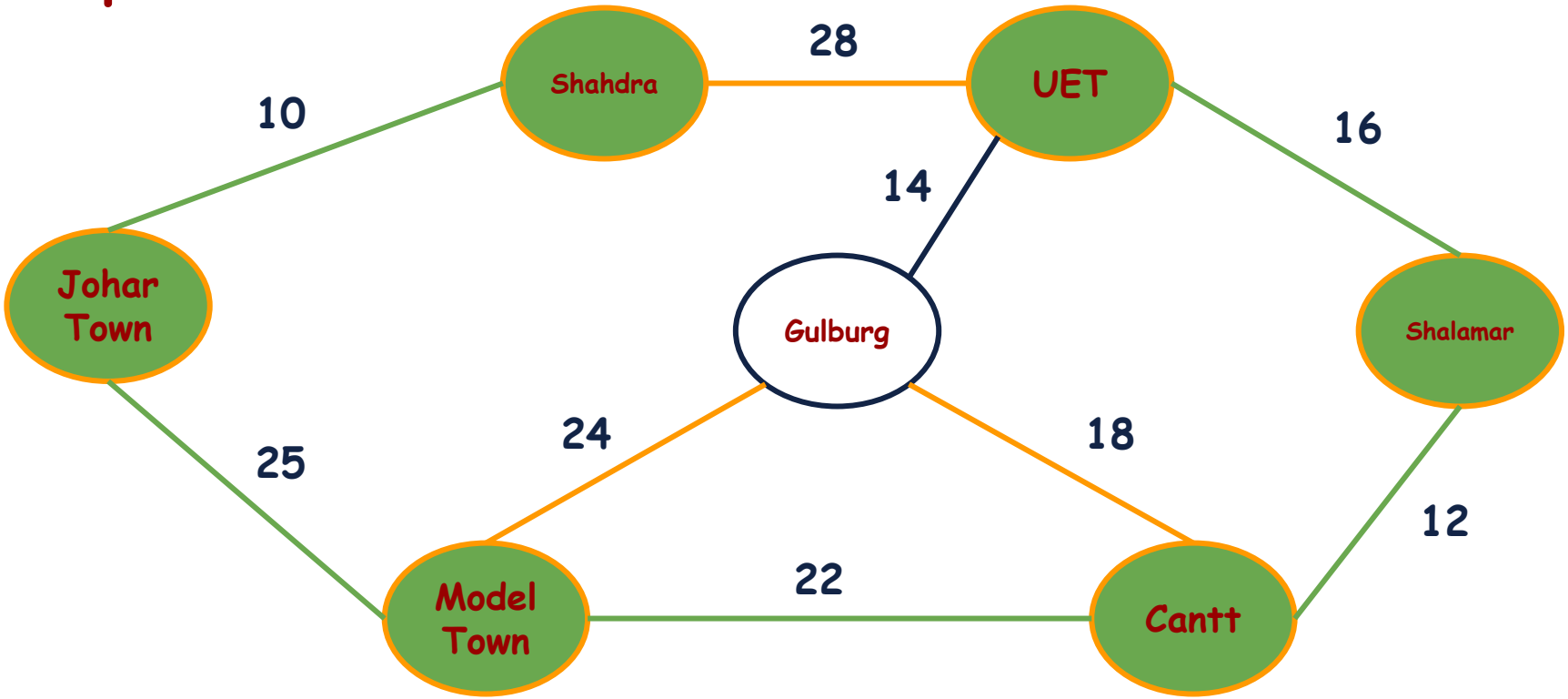
Minimum Spanning Tree: Solution

Step 4: Choose the minimum weighted edge whose vertex is not visited



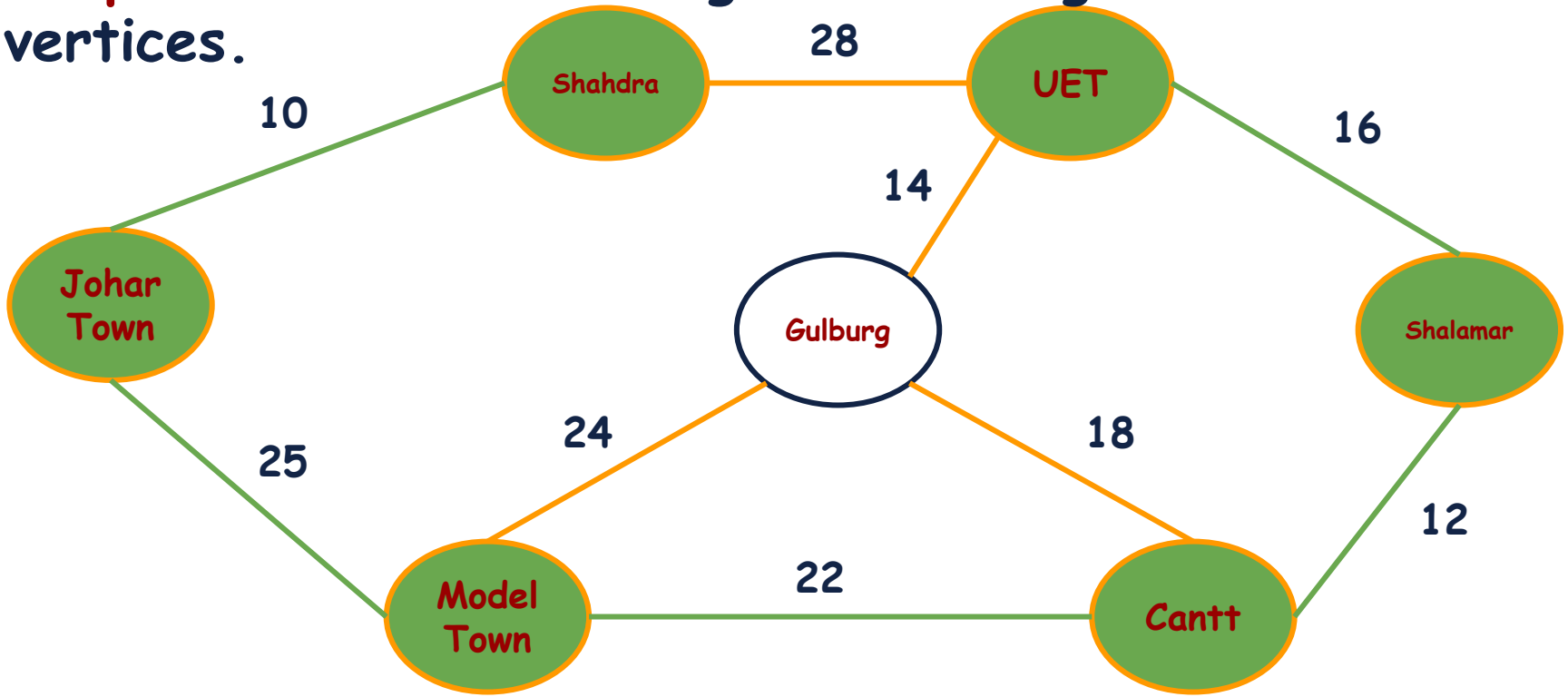
Minimum Spanning Tree: Solution

Step 2: Marked the vertex as visited.



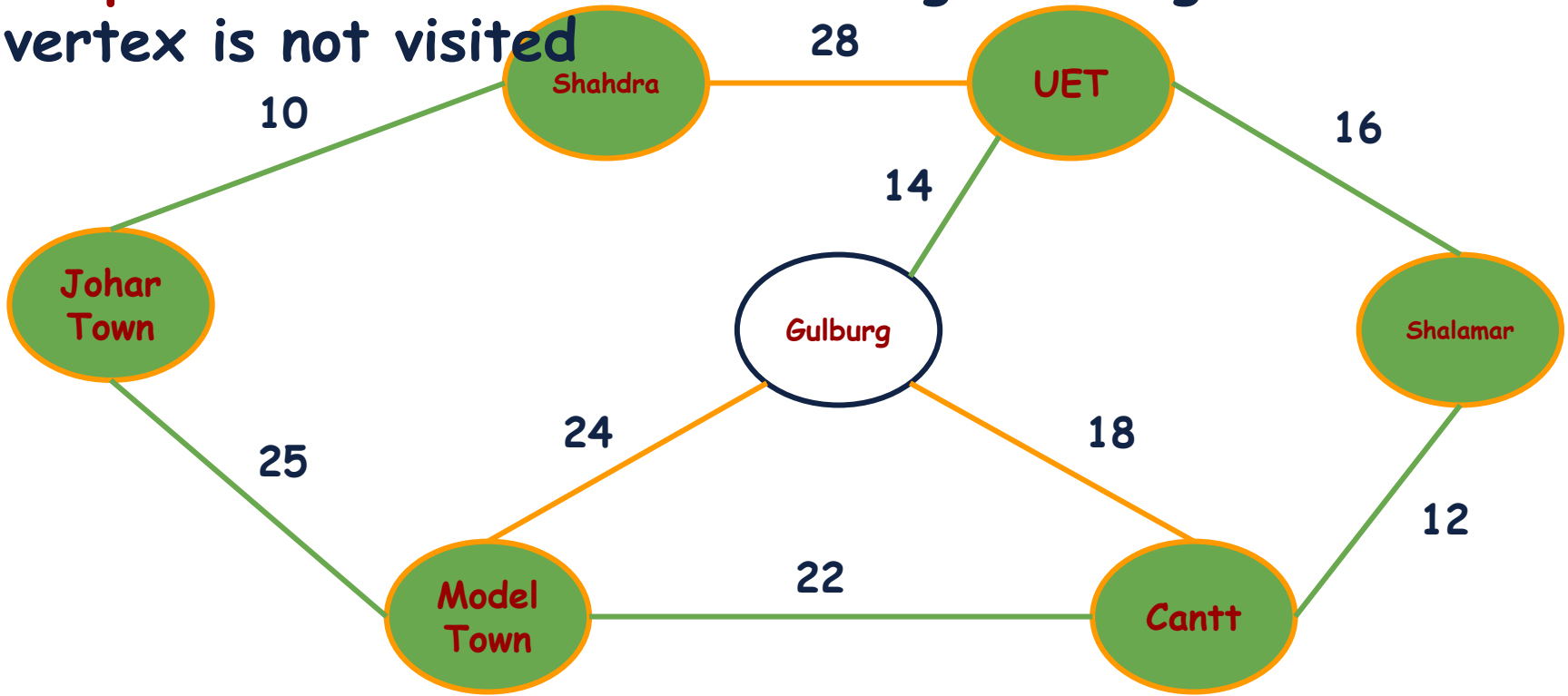
Minimum Spanning Tree: Solution

Step 3: Check all the edges extending from the visited vertices.



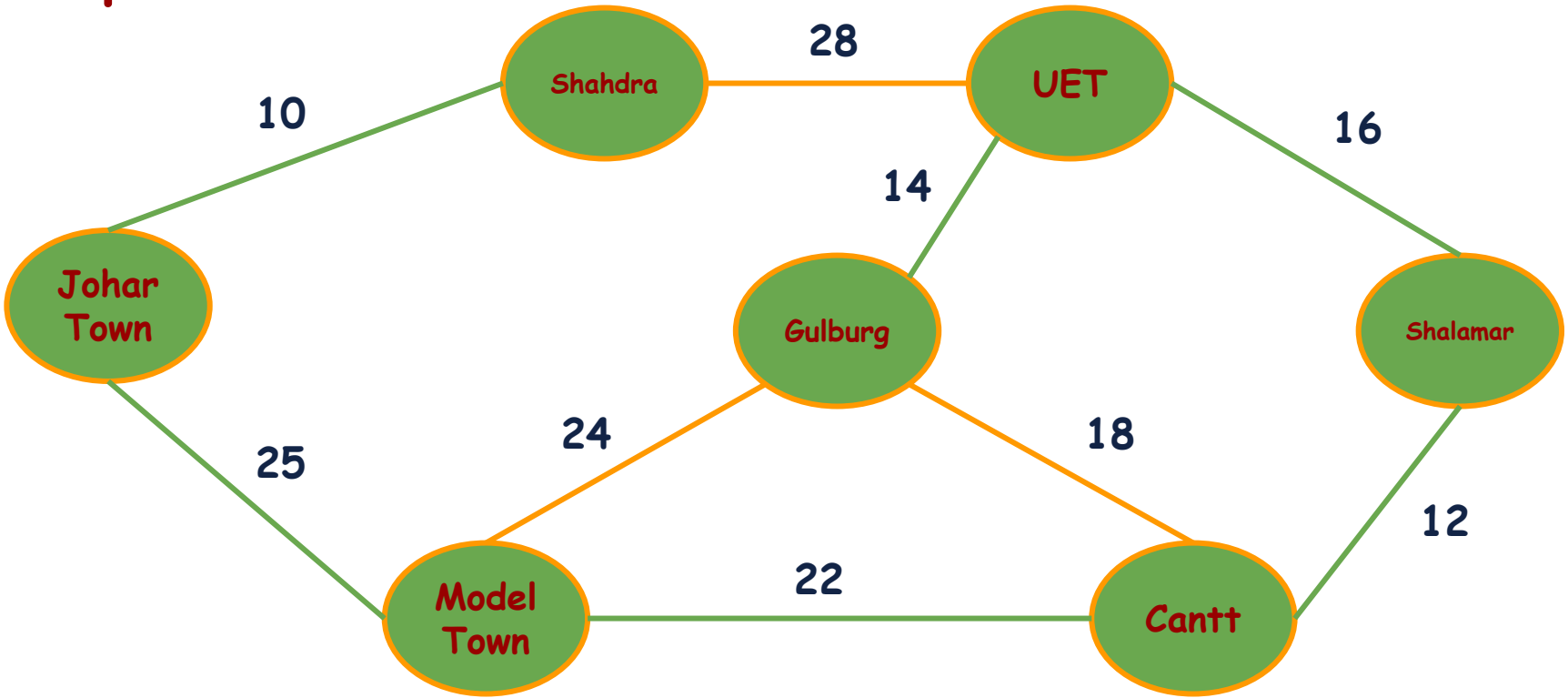
Minimum Spanning Tree: Solution

Step 4: Choose the minimum weighted edge whose vertex is not visited



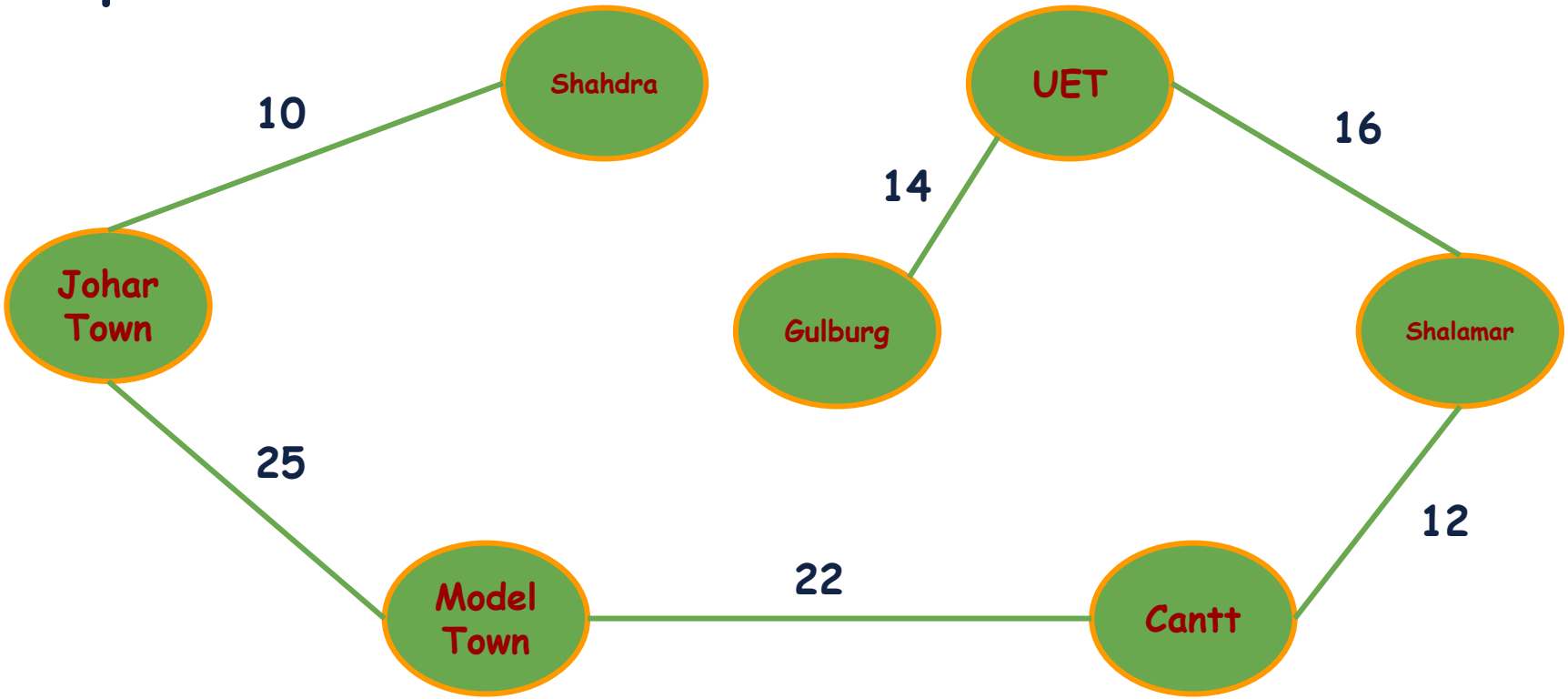
Minimum Spanning Tree: Solution

Step 2: Mark it as visited.



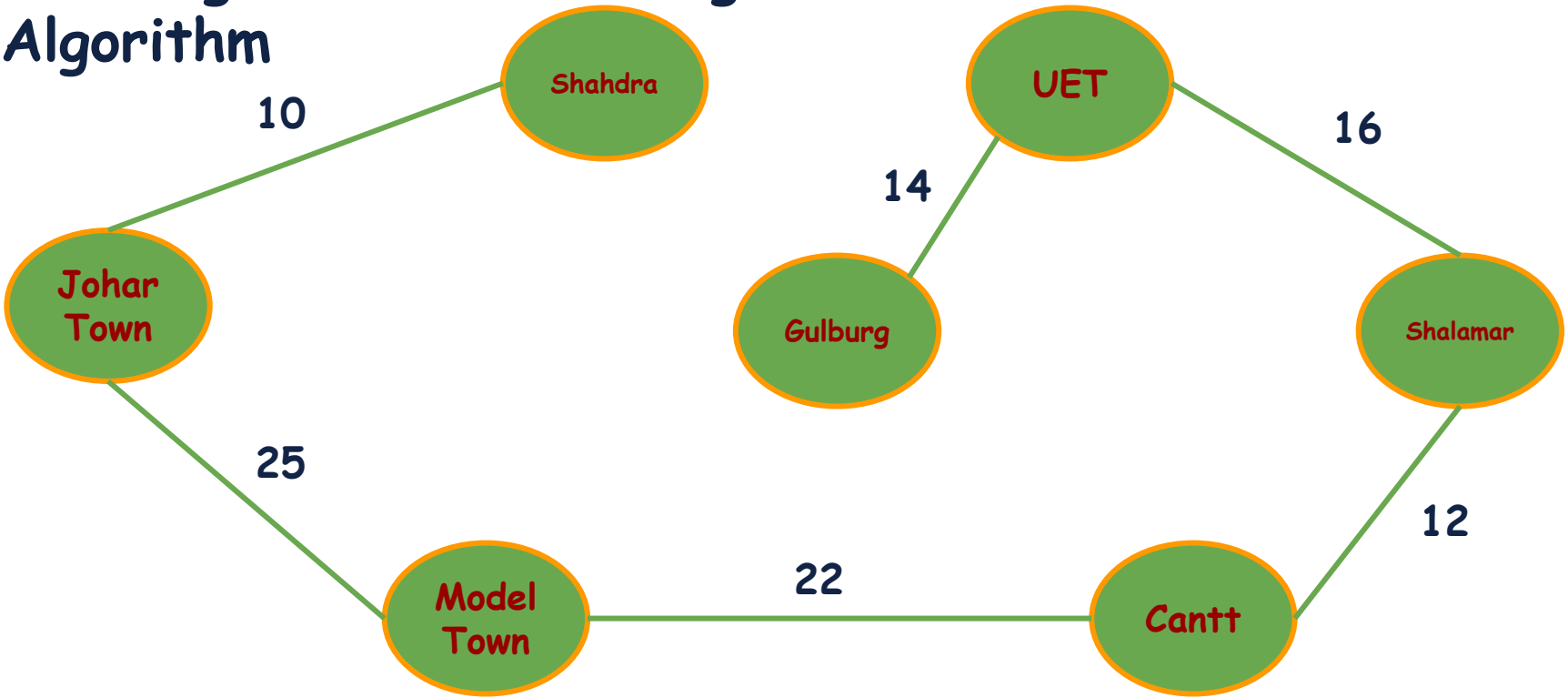
Minimum Spanning Tree: Solution

Stop when all the vertices are visited.



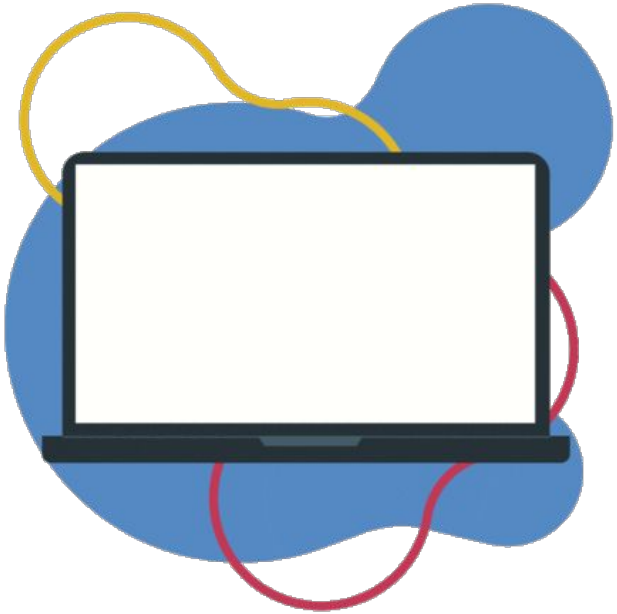
Minimum Spanning Tree: Prim's Algorithm

This Algorithm for finding the MST is called Prim's Algorithm



Prim's Algorithm: Implementation

Lets implement the solution.



Prim's Algorithm: Implementation

```
class Graph
{
    typedef pair<int, string> edgeCost;
    unordered_map<string, vector<edgeCost>> g;

public:

    addEdge(string sourceTown, string destinationTown, int distance)
    {
        g[sourceTown].push_back({distance, destinationTown});
        g[destinationTown].push_back({distance, sourceTown});
    }
```

Prim's Algorithm: Implementation

```
int minimumCostPrims(string start){
        unordered_map<string, bool> visited;
        priority_queue<edgeCost, vector<edgeCost>, greater<edgeCost>> pq;
        int cost = 0;
        pq.push({0, start});
        while (!pq.empty()){
            edgeCost currentTown = pq.top();
            pq.pop();
            if (visited.find(currentTown.second) == visited.end())
            {
                cost = cost + currentTown.first;
                visited[currentTown.second] = true;
                for (auto town : g[currentTown.second])
                {
                    if (visited.find(town.second) == visited.end())
                        pq.push(town);
                }
            }
        }
        return cost;
    }
```

Prim's Algorithm: Implementation

What is the Time Complexity of Prim's Algorithm?

Prim's Algorithm: Implementation

- We are traversing the complete graph.
  $O(|V+E|)$ is the time complexity to traverse the graph.

- We are maintaining the min heap for finding the minimum weighted edge.
  Height of the min heap would be $\log(V)$.

- Therefore, time complexity is $O(|V+E| * \log(|V|))$.

Prim's Algorithm: Implementation

Time Complexity is $O(|E| * \log(|V|))$

Prim's Algorithm: Implementation

Space Complexity is $O(|E| + |V|)$

Prim's Algorithm: Implementation

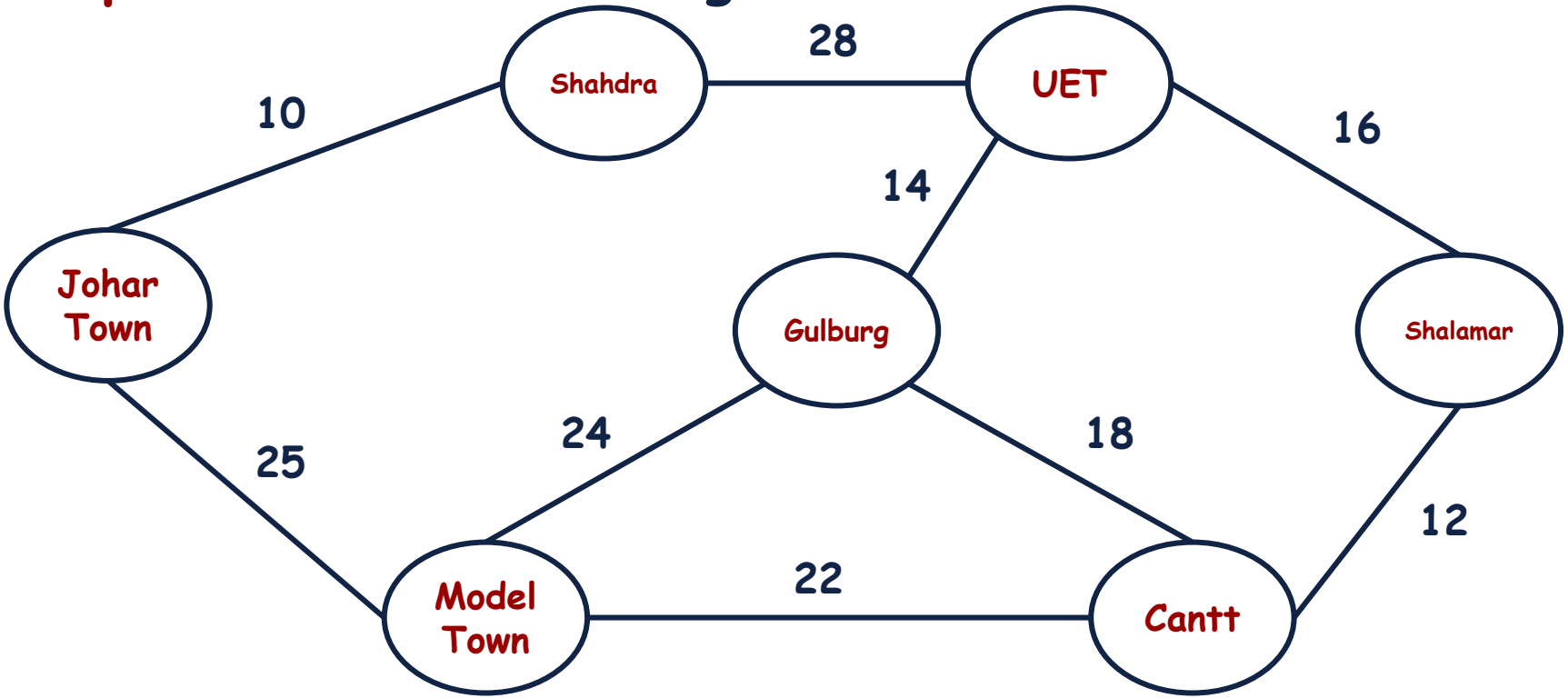
Minimum Spanning Tree	Time Complexity	Space Complexity
	Worst Case	Worst Case
Prim's Algorithm	O(|E| * log(|V|))	O(|E| + |V|)

Minimum Spanning Tree

We have seen Prim's Algorithm for finding the minimum spanning tree. Here is another algorithm.

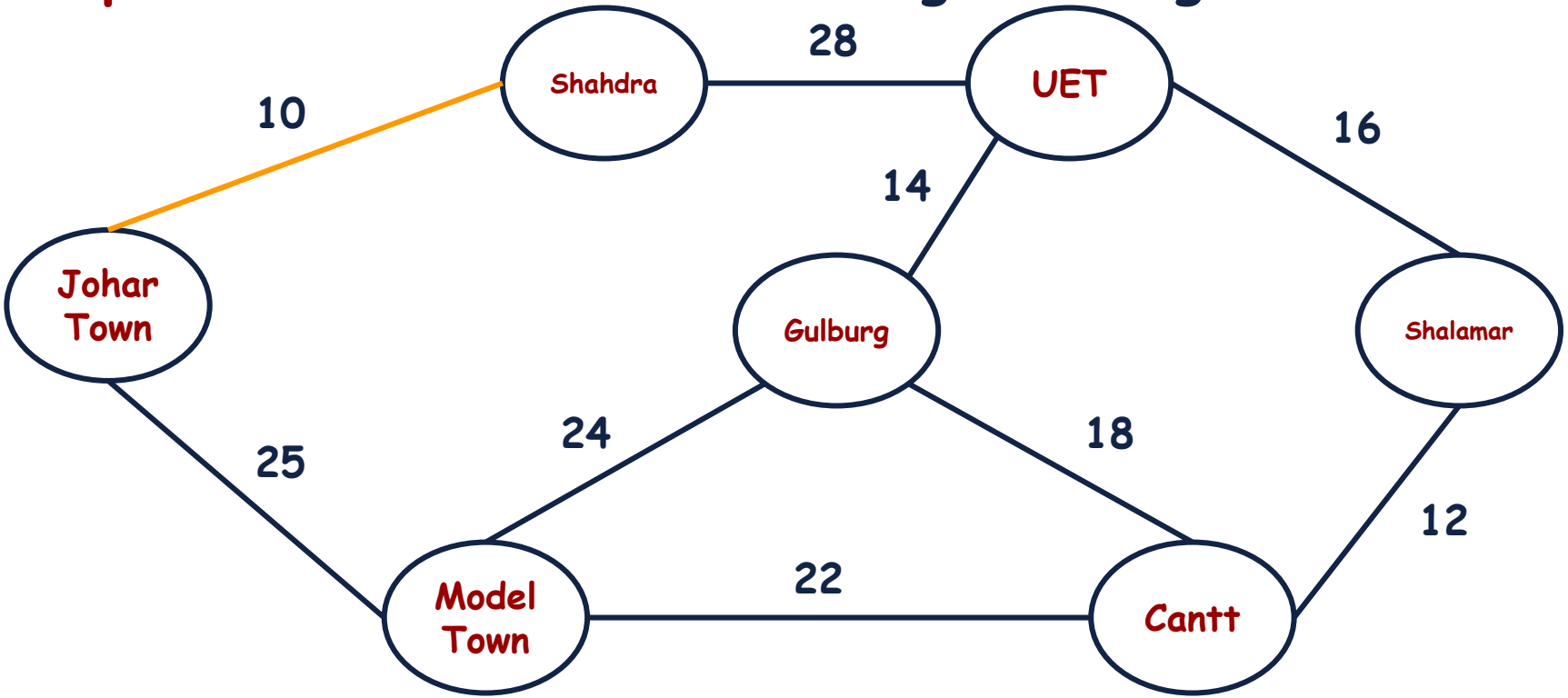
Minimum Spanning Tree: Solution 02

Step 1: Store all the Edges.



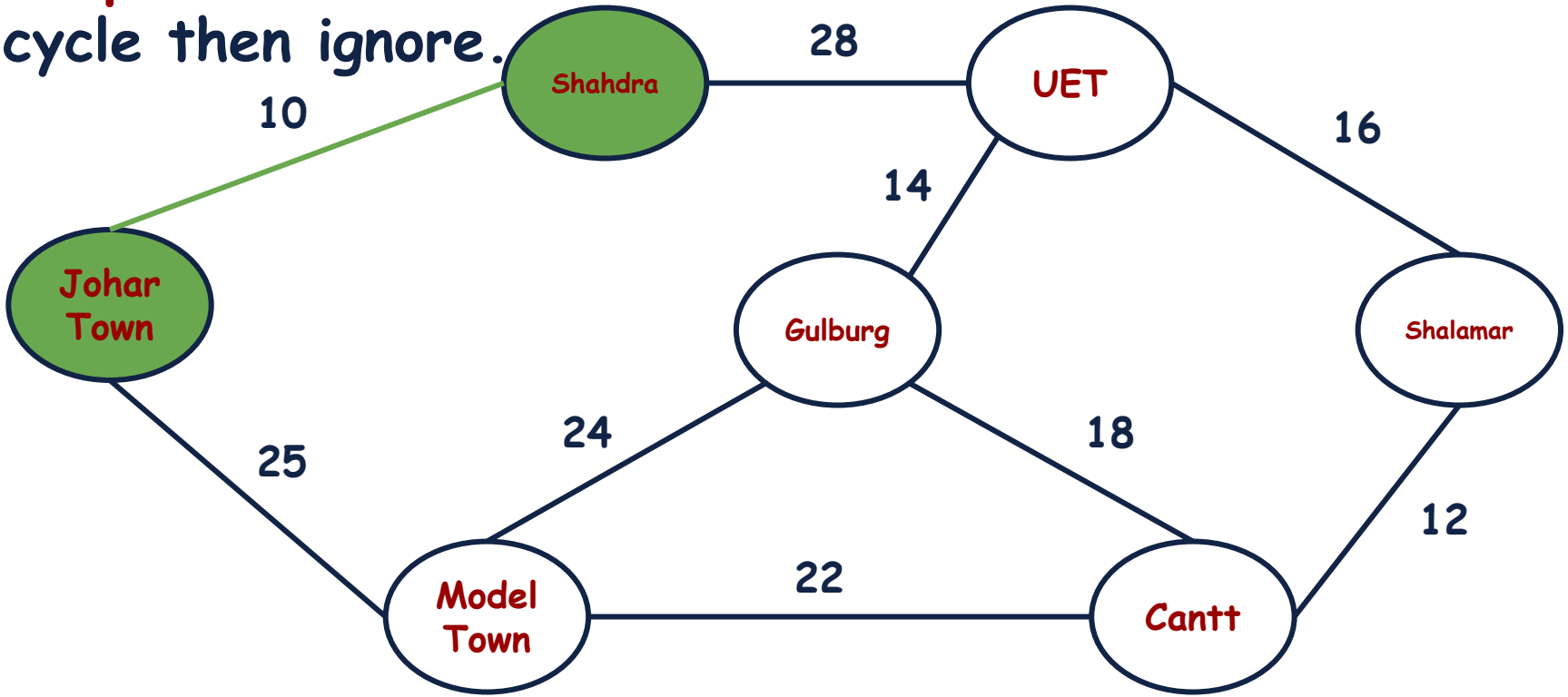
Minimum Spanning Tree: Solution 02

Step 2: Choose the minimum weighted Edge.



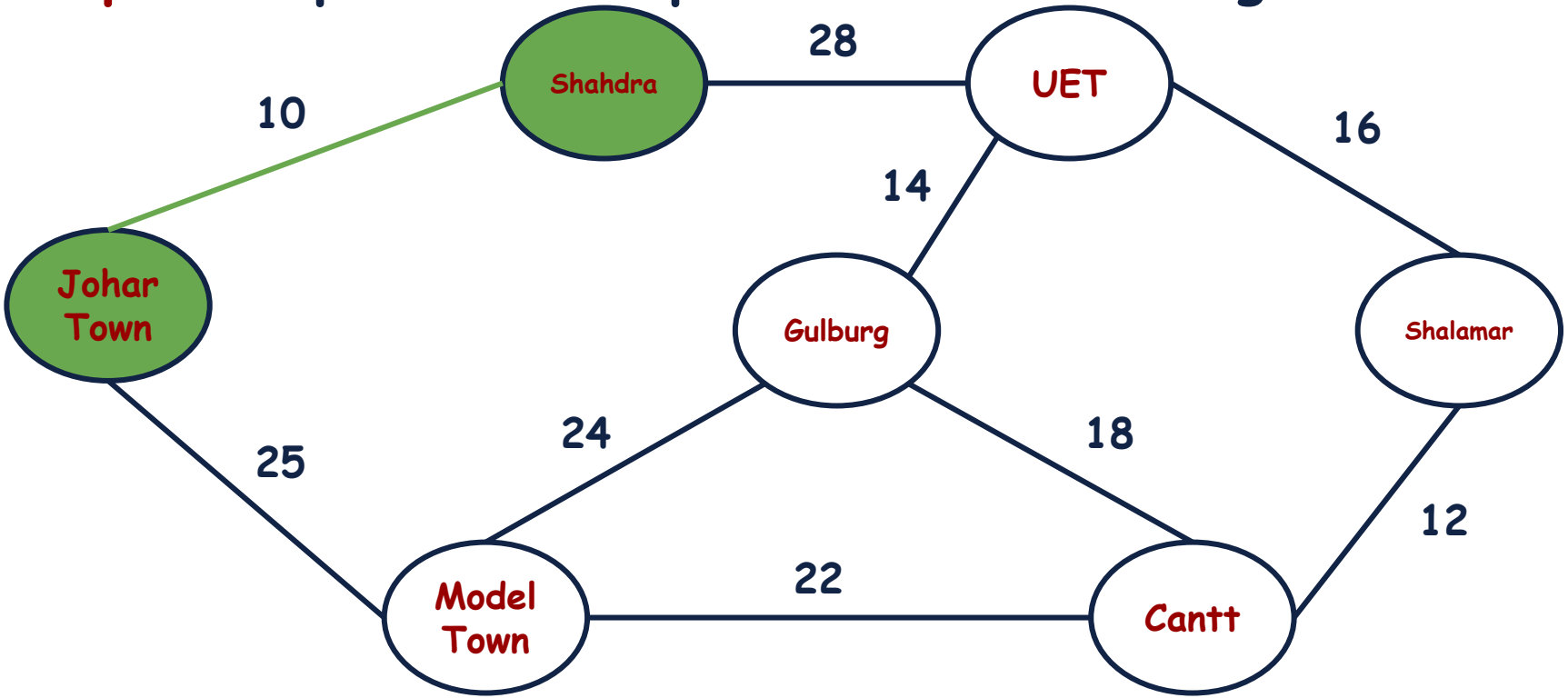
Minimum Spanning Tree: Solution 02

Step 3: Add its vertices to the MST. If it creates a cycle then ignore.



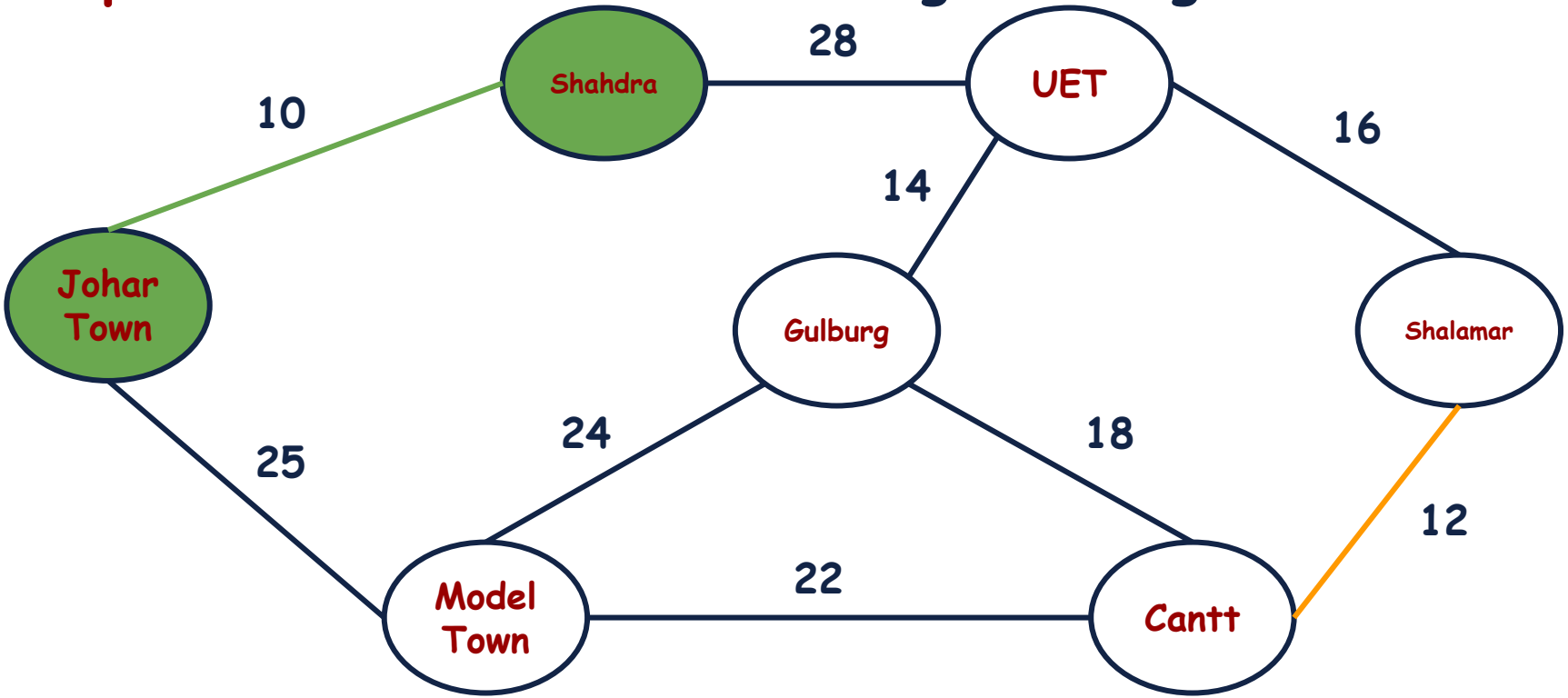
Minimum Spanning Tree: Solution 02

Step 4: Repeat the step 2 and 3 for all edges.



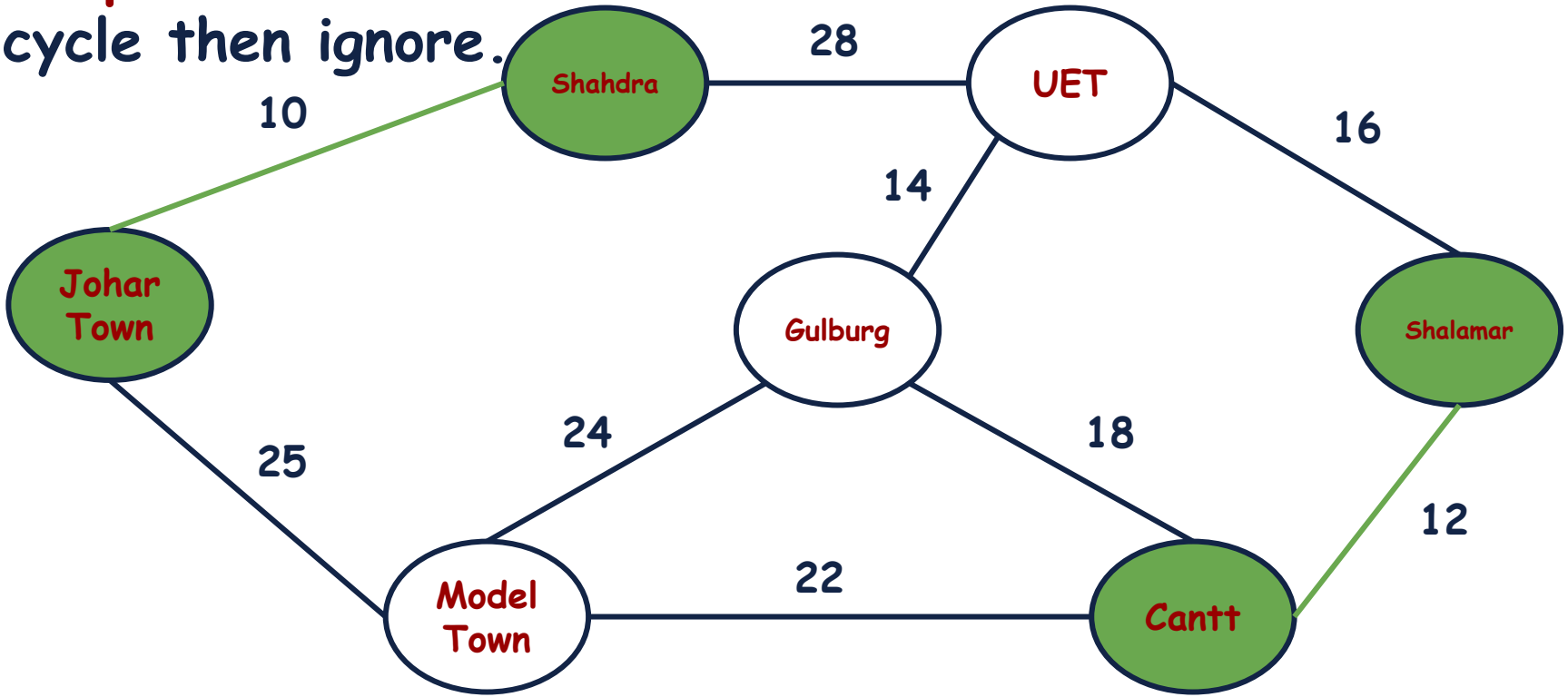
Minimum Spanning Tree: Solution 02

Step 2: Choose the minimum weighted edge



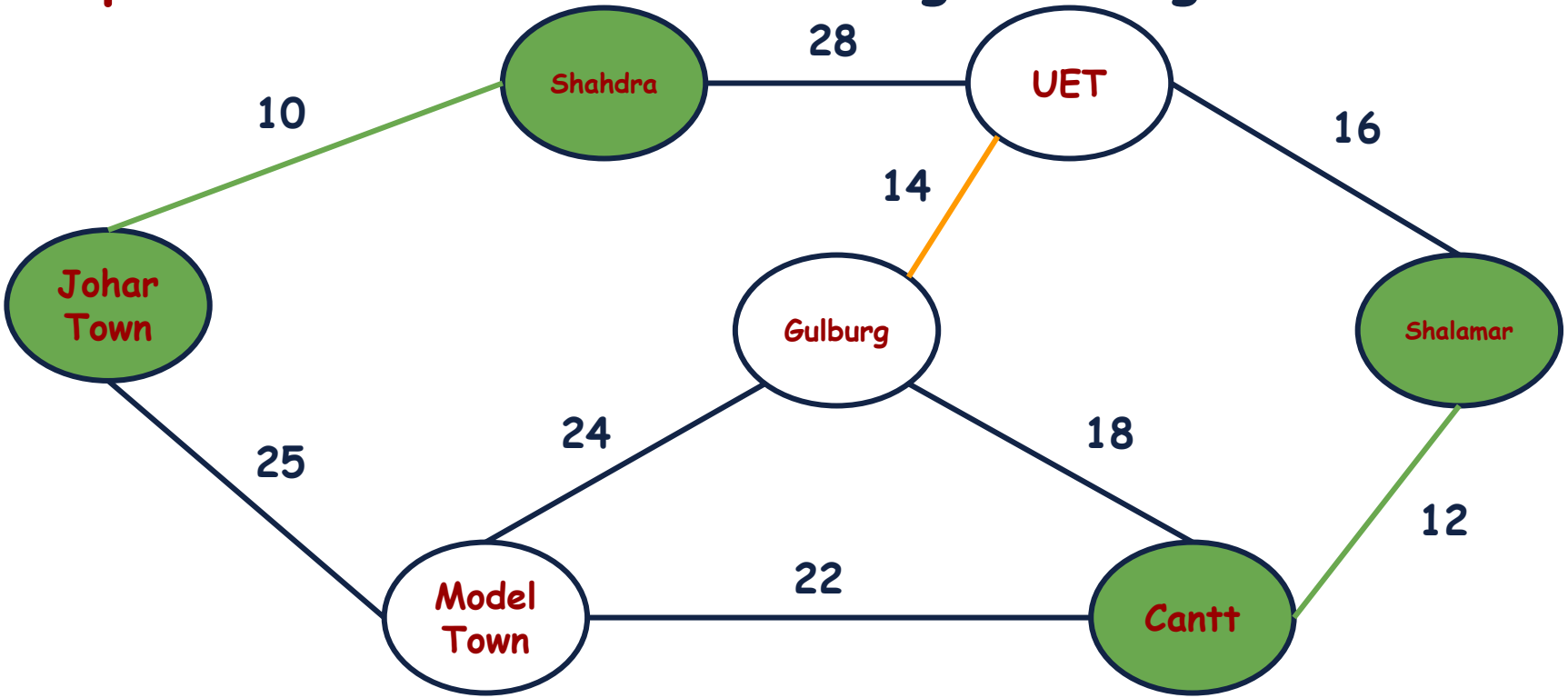
Minimum Spanning Tree: Solution 02

Step 3: Add its vertices to the MST. If it creates a cycle then ignore.



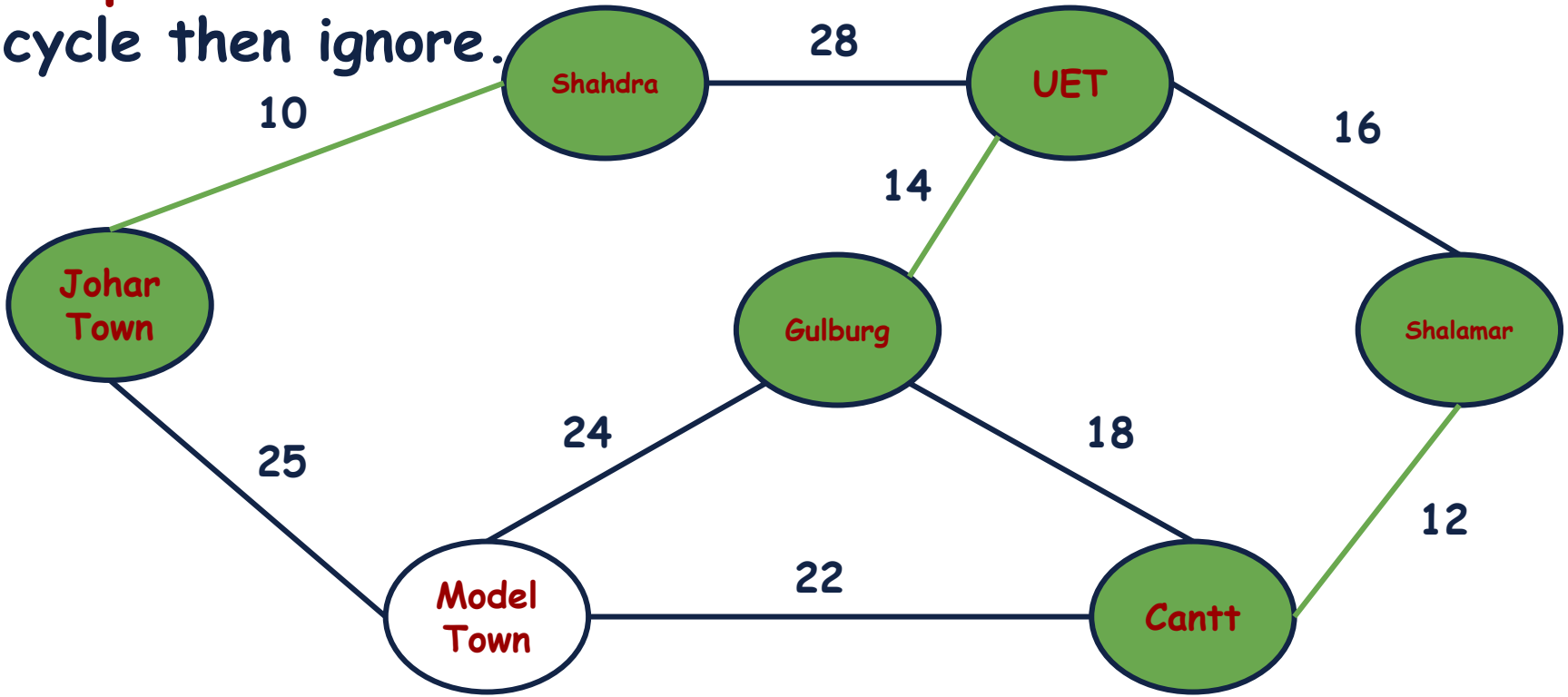
Minimum Spanning Tree: Solution 02

Step 2: Choose the minimum weighted edge.



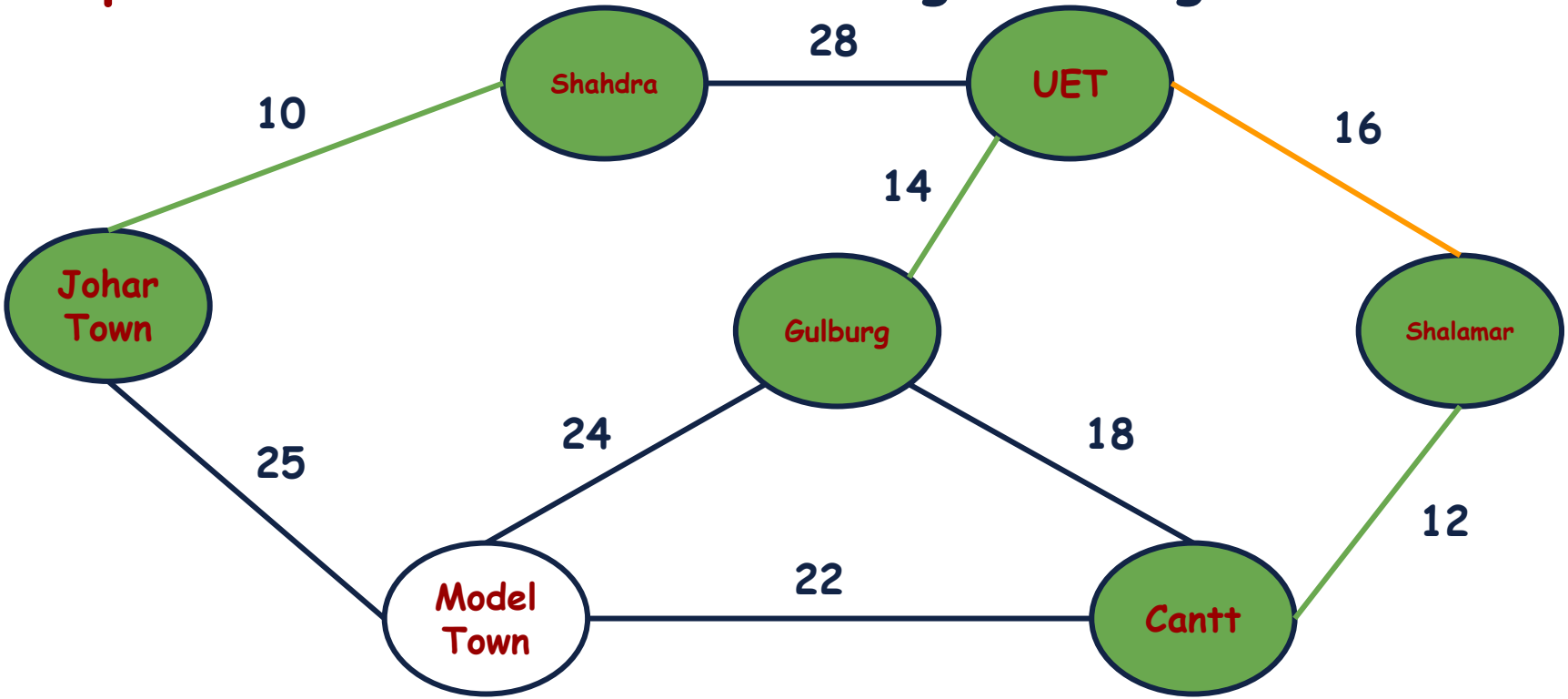
Minimum Spanning Tree: Solution 02

Step 3: Add its vertices to the MST. If it creates a cycle then ignore.



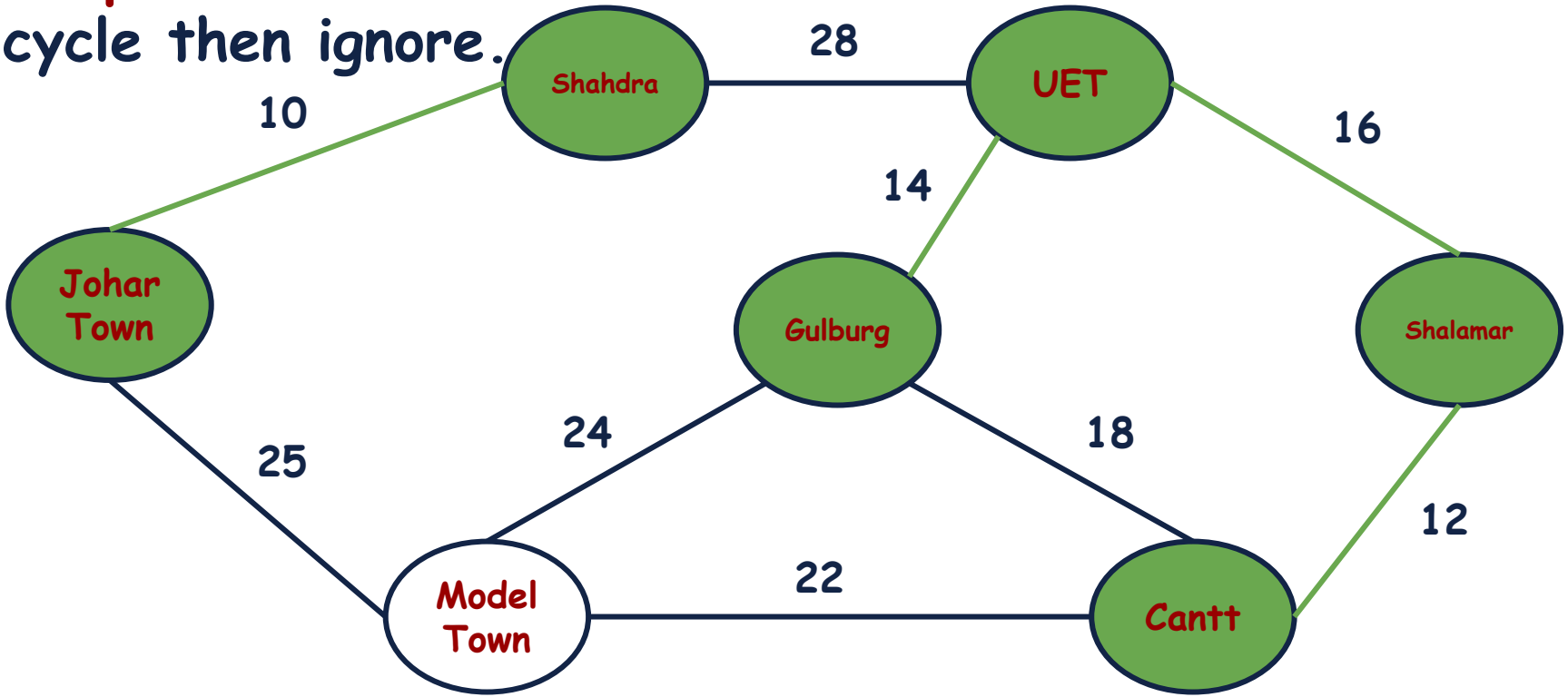
Minimum Spanning Tree: Solution 02

Step 2: Choose the minimum weighted Edge.



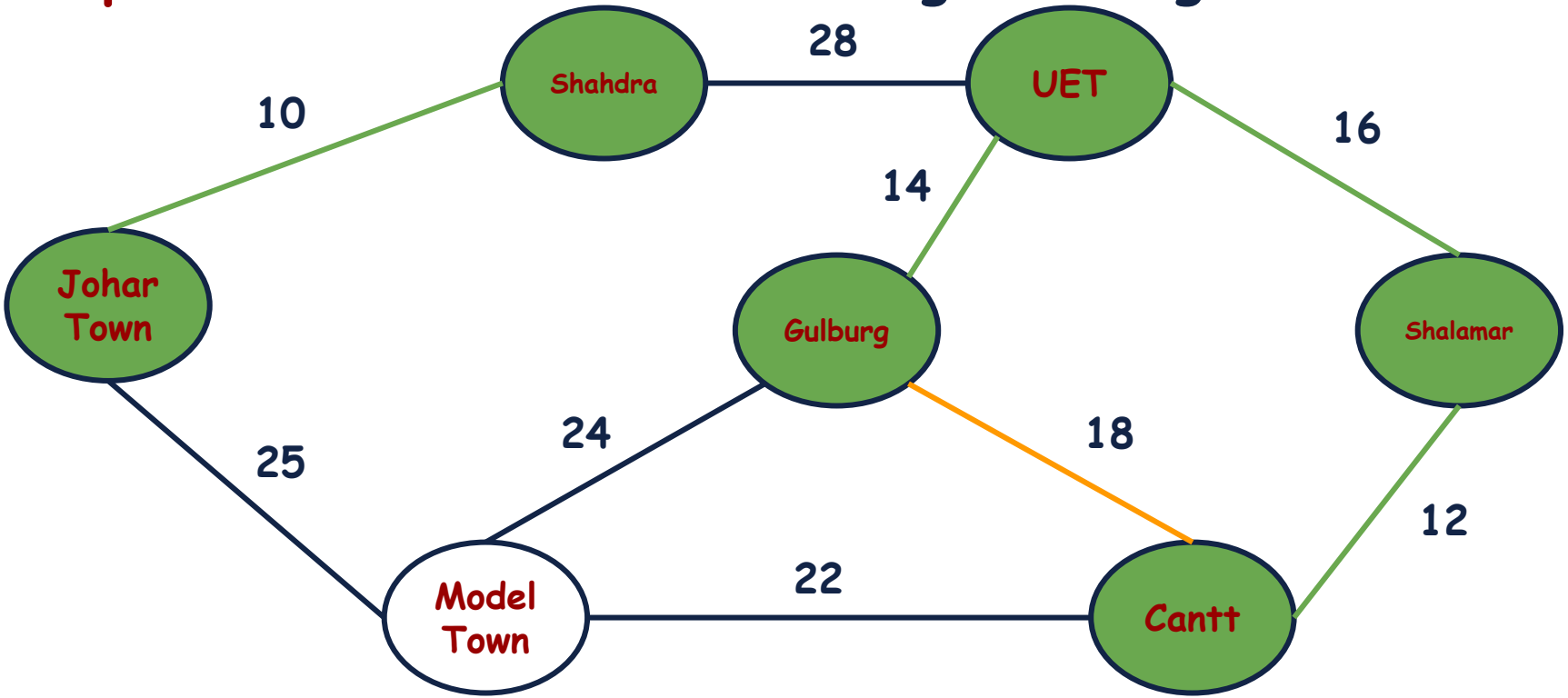
Minimum Spanning Tree: Solution 02

Step 3: Add its vertices to the MST. If it creates a cycle then ignore.



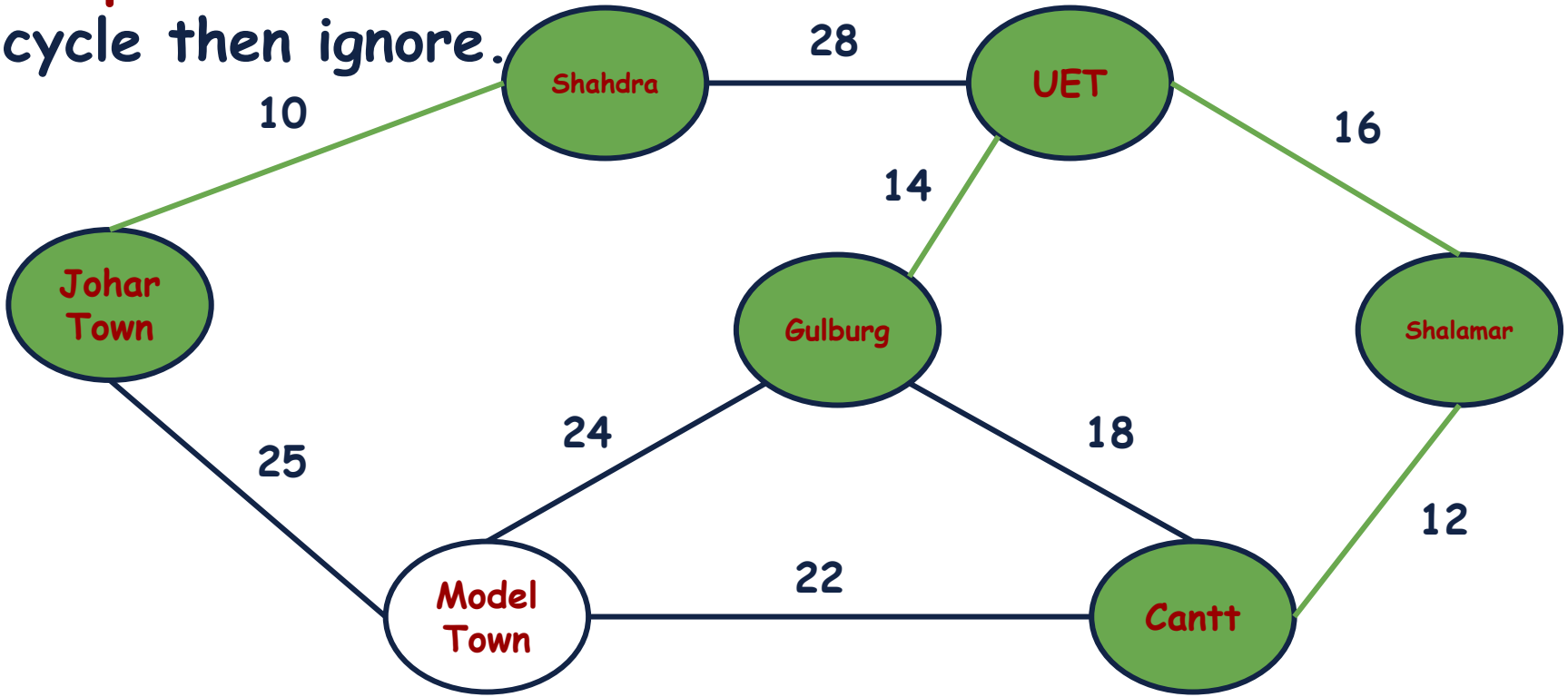
Minimum Spanning Tree: Solution 02

Step 2: Choose the minimum weighted Edge



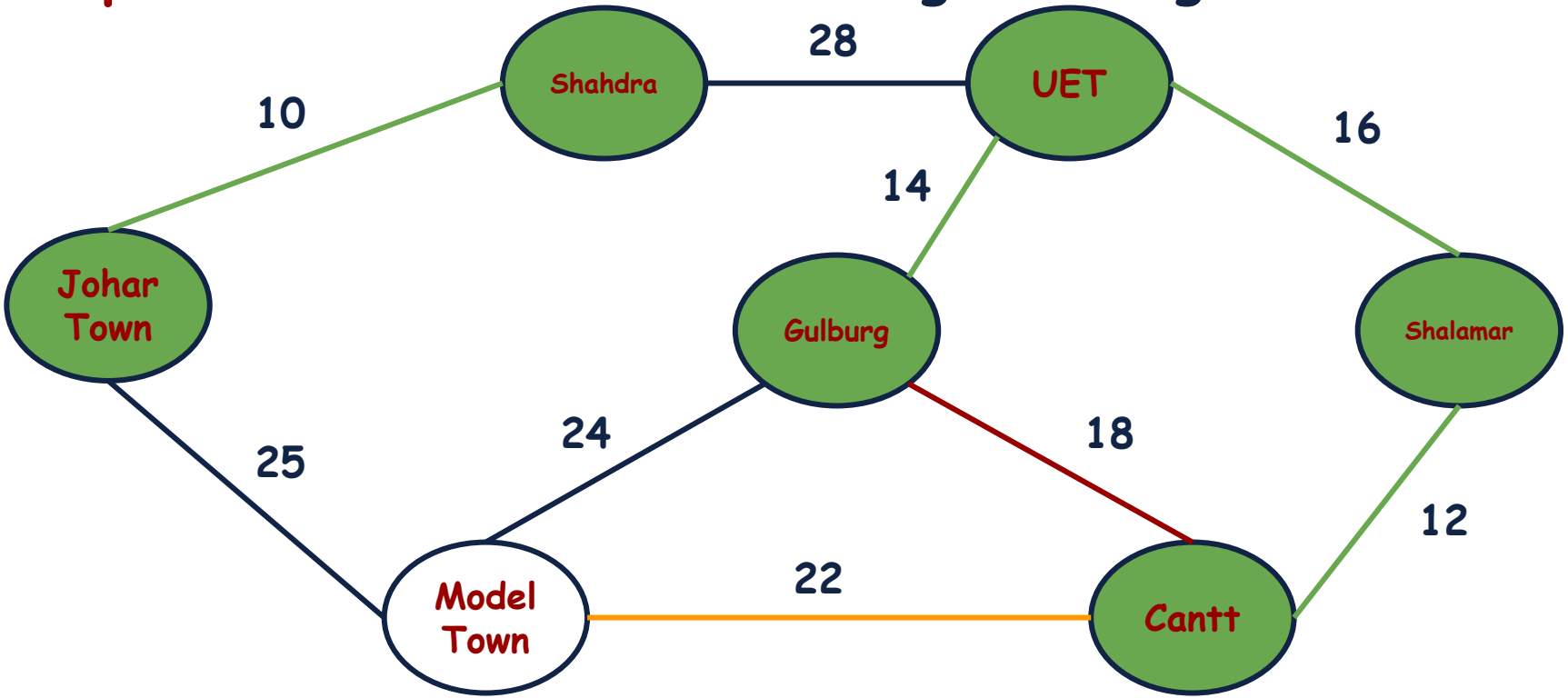
Minimum Spanning Tree: Solution 02

Step 3: Add its vertices to the MST. If it creates a cycle then ignore.



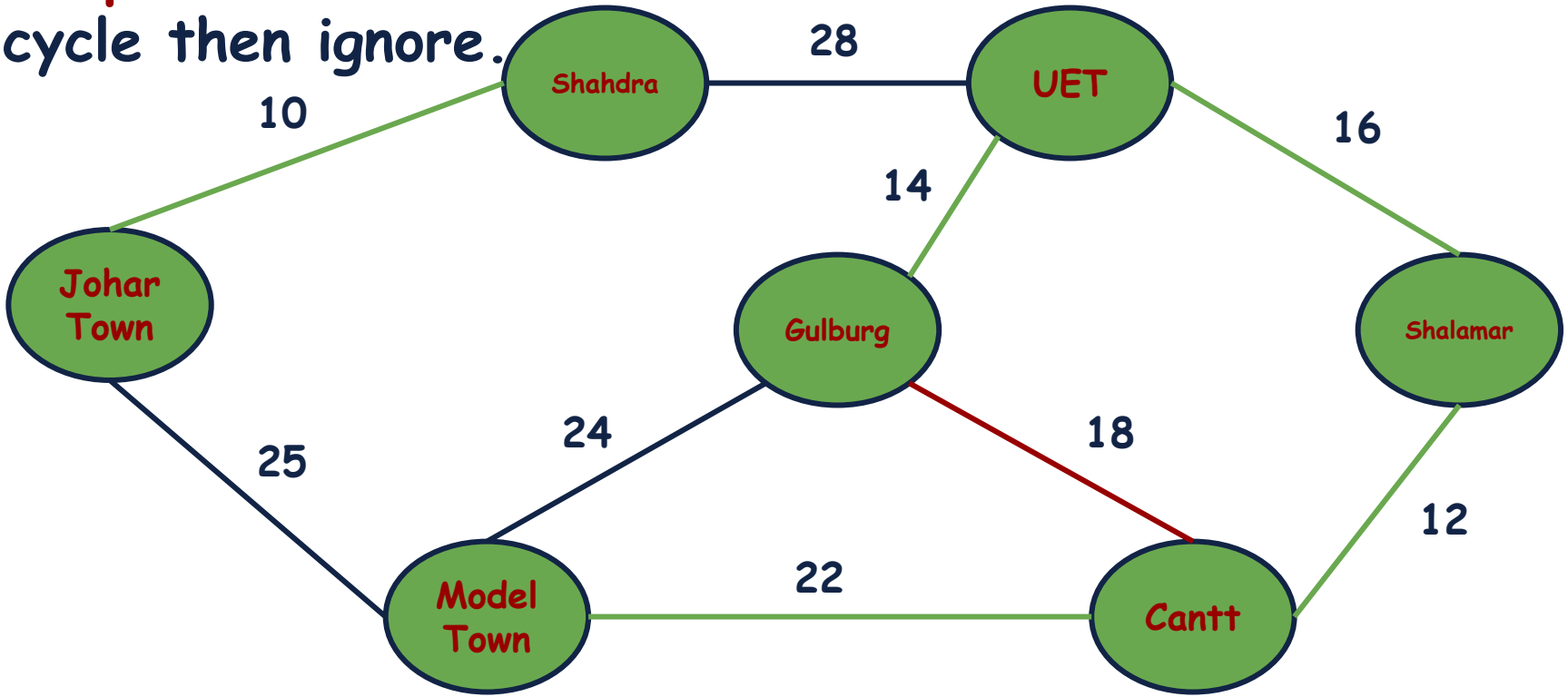
Minimum Spanning Tree: Solution 02

Step 2: Choose the minimum weighted Edge.



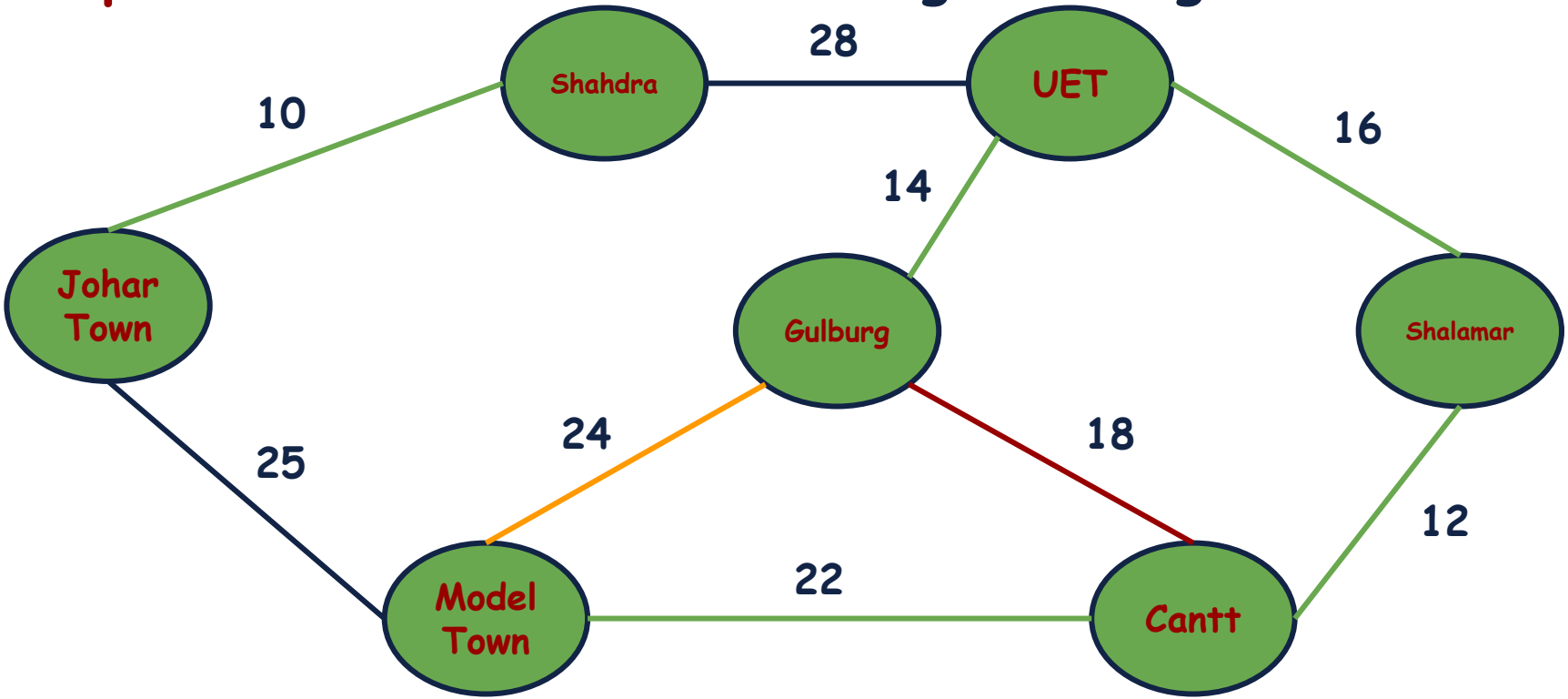
Minimum Spanning Tree: Solution 02

Step 3: Add its vertices to the MST. If it creates a cycle then ignore.



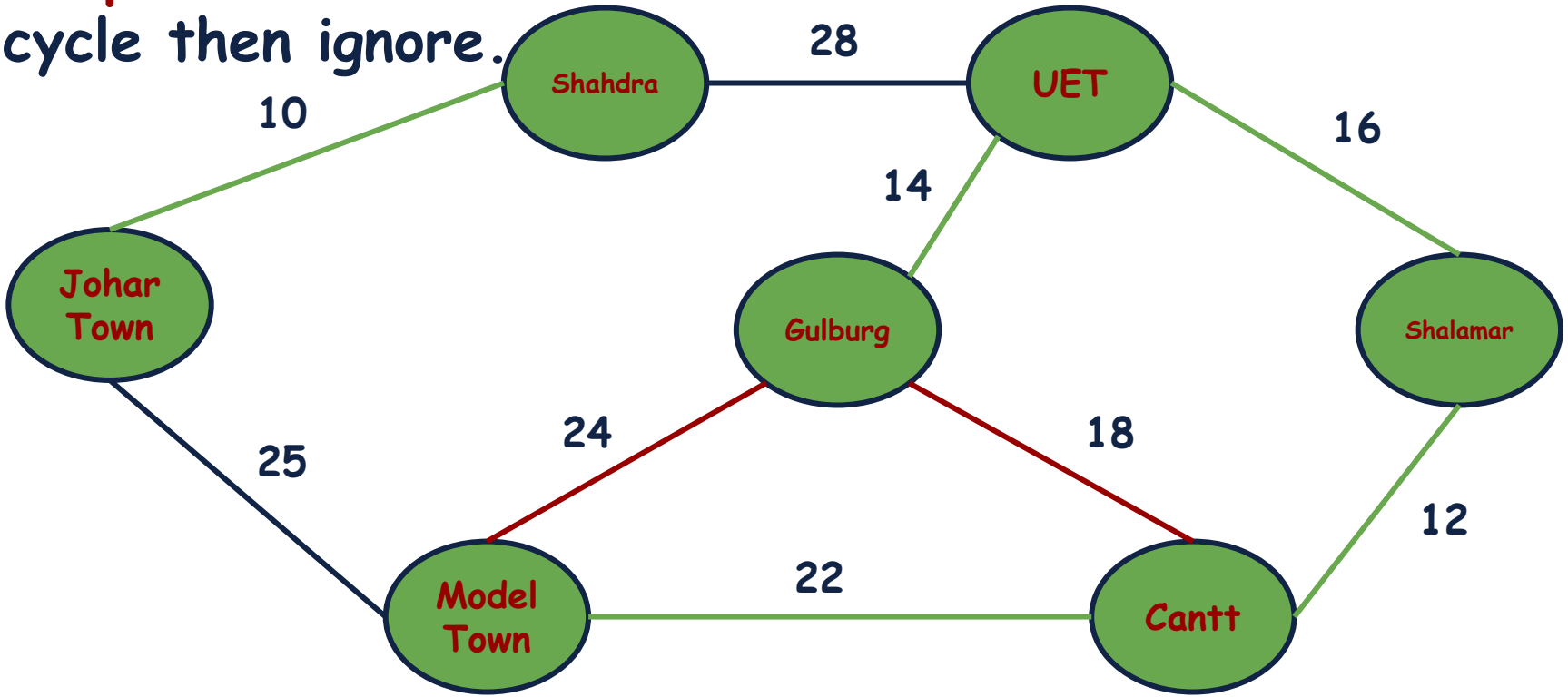
Minimum Spanning Tree: Solution 02

Step 2: Choose the minimum weighted Edge.



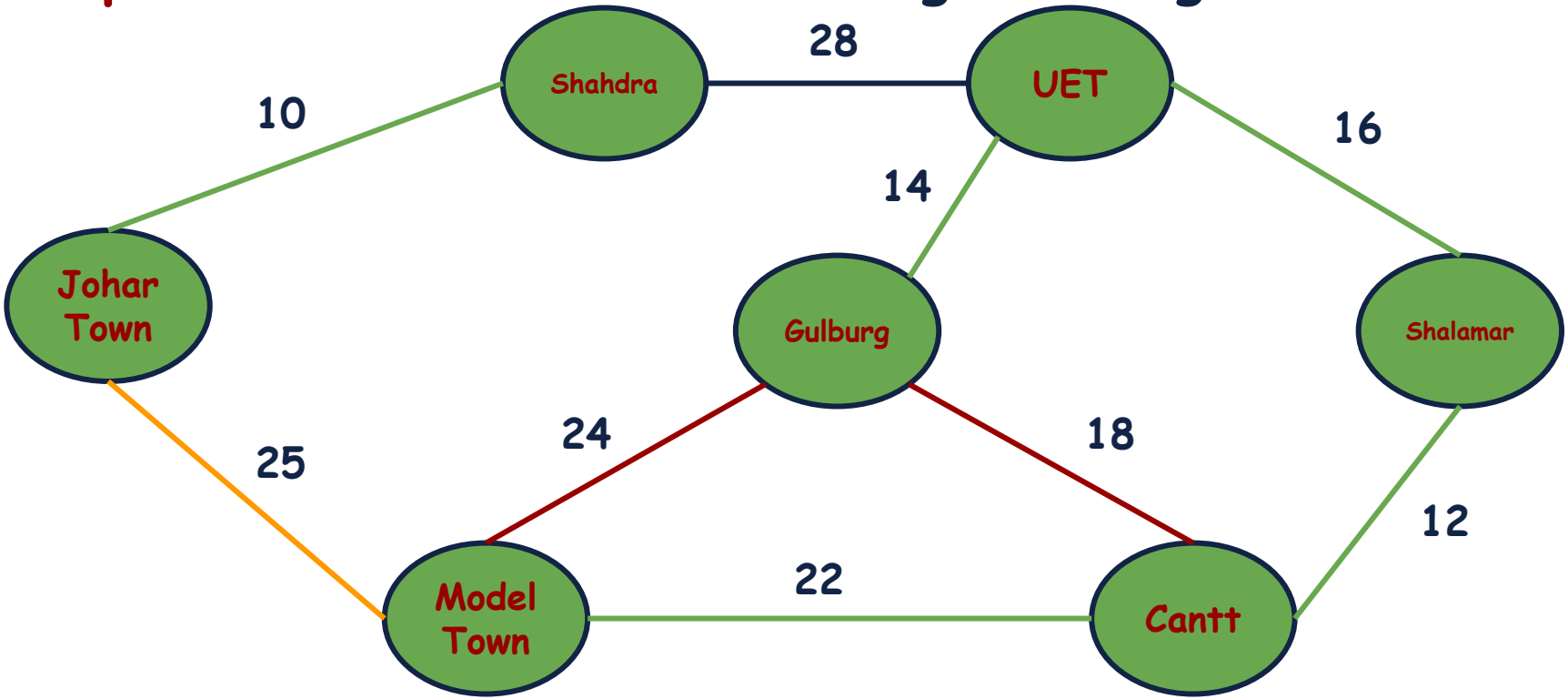
Minimum Spanning Tree: Solution 02

Step 3: Add its vertices to the MST. If it creates a cycle then ignore.



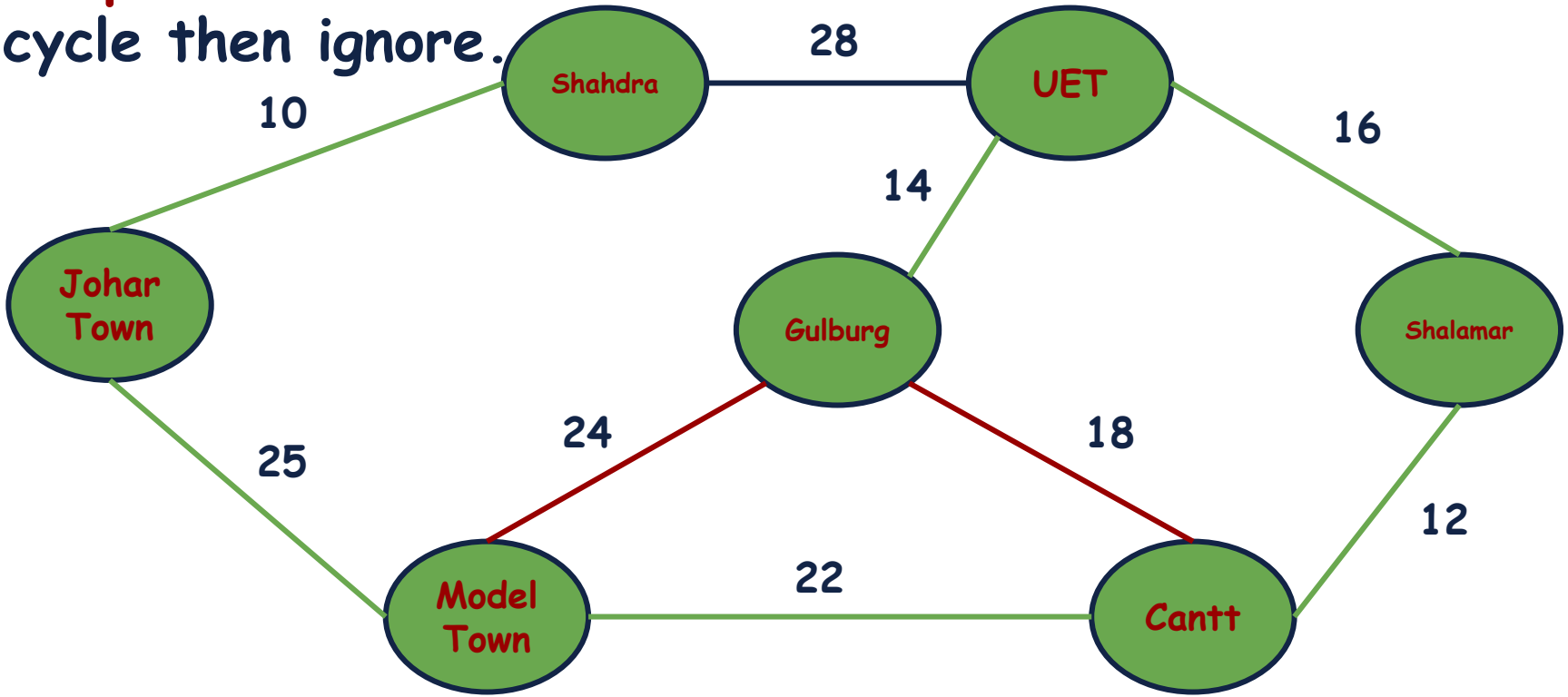
Minimum Spanning Tree: Solution 02

Step 2: Choose the minimum weighted Edge.



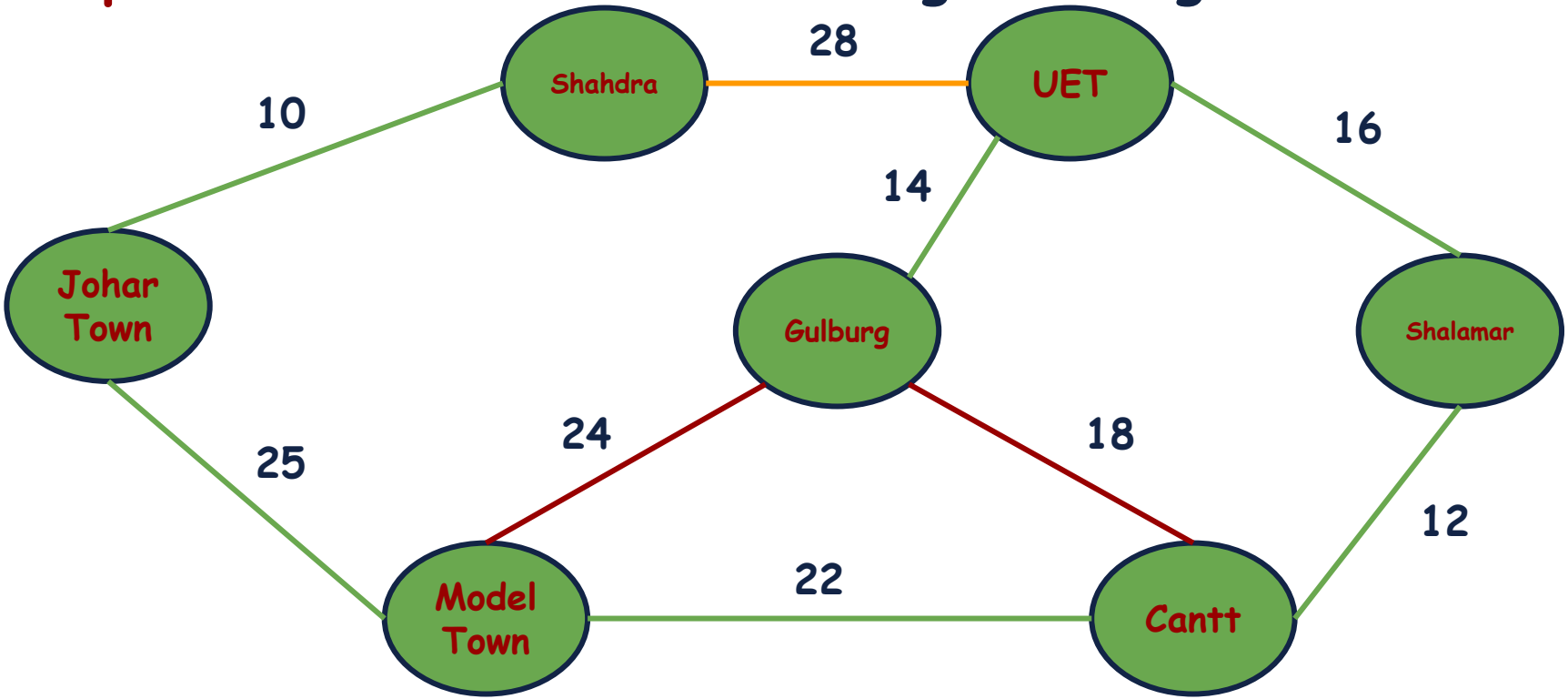
Minimum Spanning Tree: Solution 02

Step 3: Add its vertices to the MST. If it creates a cycle then ignore.



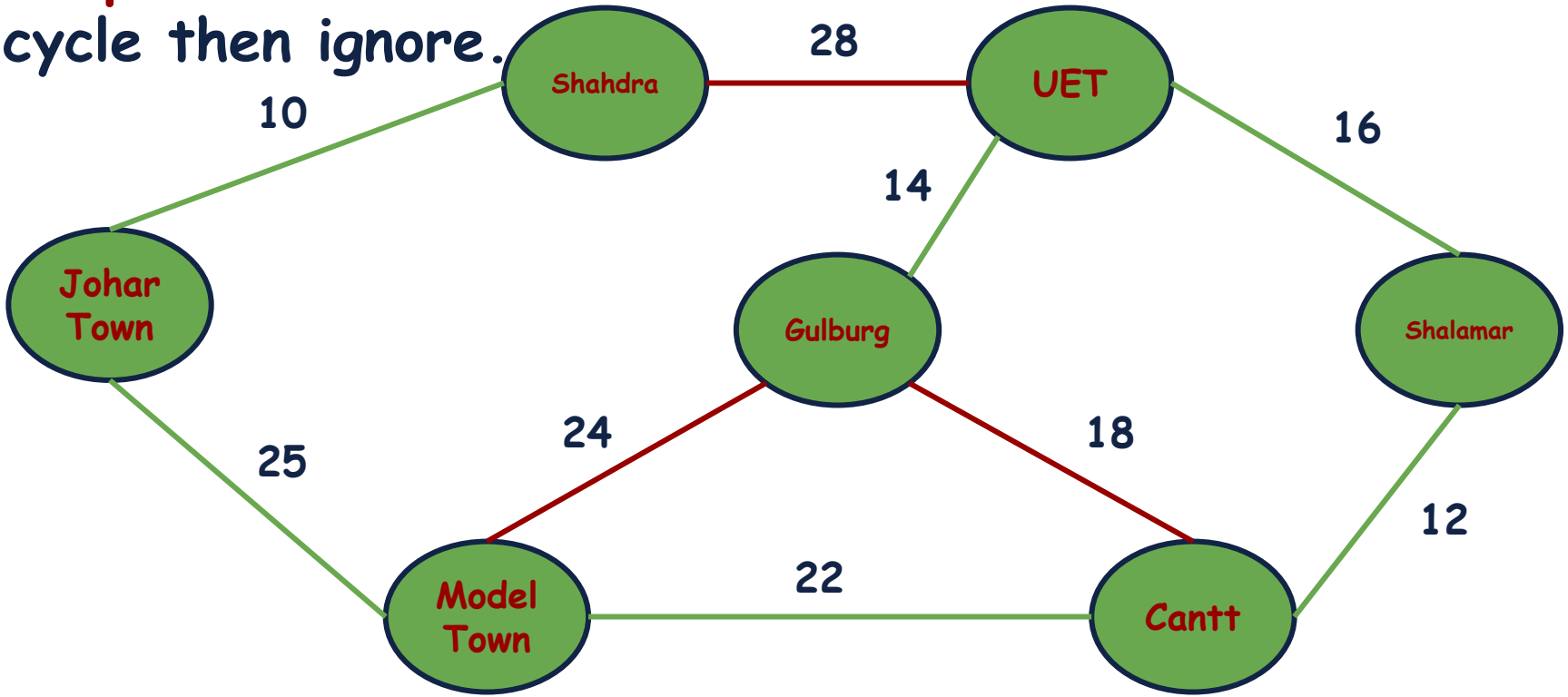
Minimum Spanning Tree: Solution 02

Step 2: Choose the minimum weighted Edge.



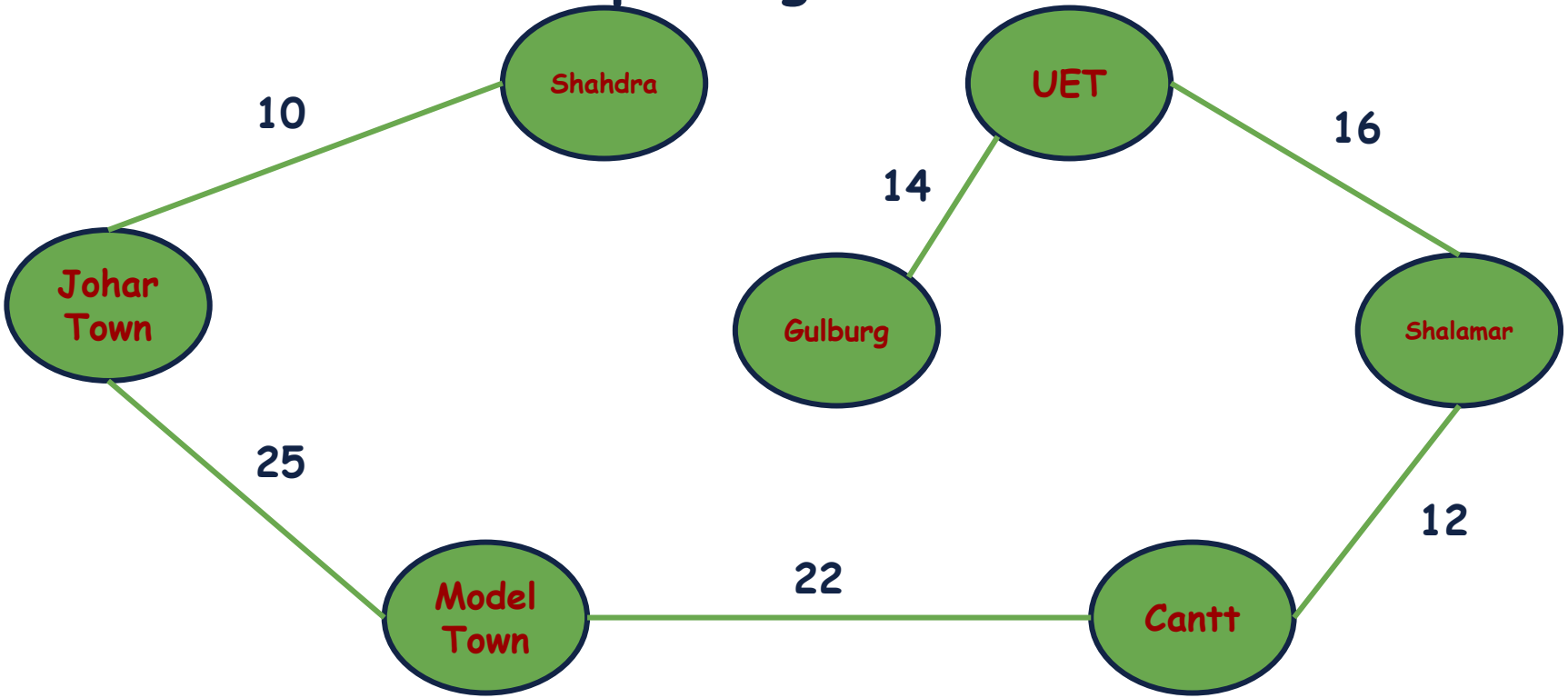
Minimum Spanning Tree: Solution 02

Step 3: Add its vertices to the MST. If it creates a cycle then ignore.



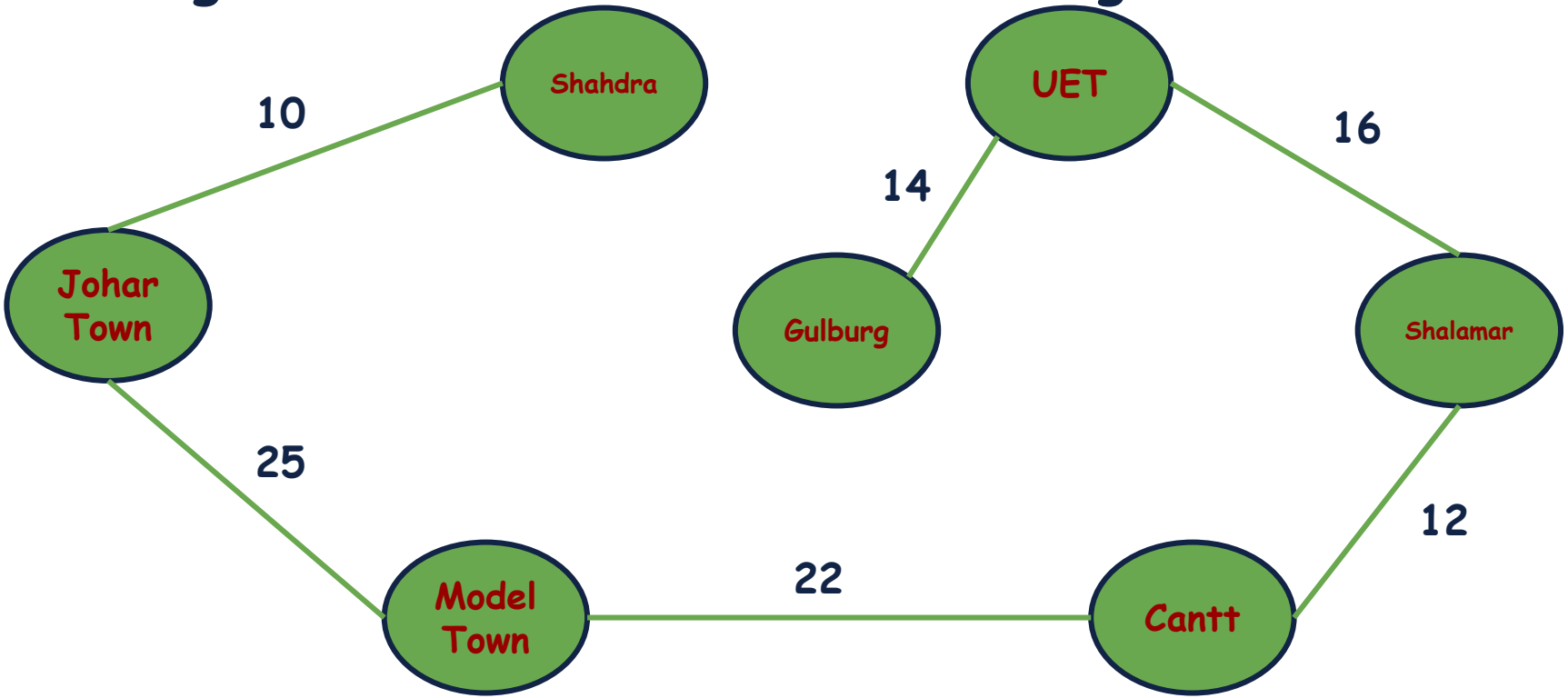
Minimum Spanning Tree: Solution 02

This is our minimum spanning tree.



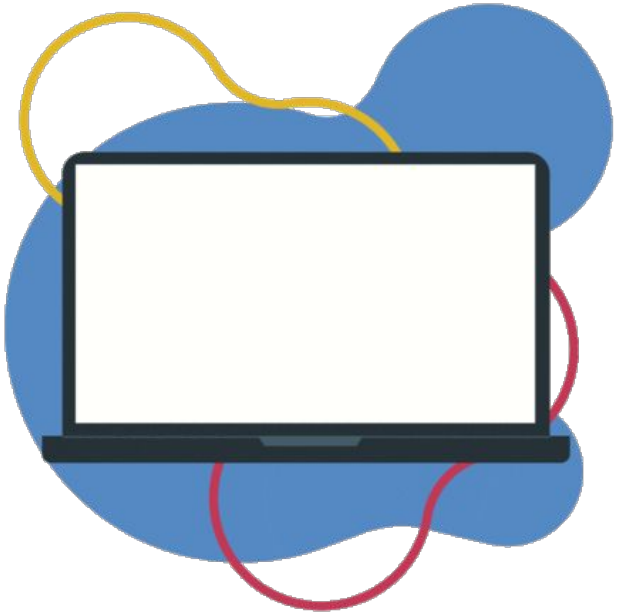
Minimum Spanning Tree: Kruskal's Algorithm

This algorithm is called as Kruskal's Algorithm.



Kruskal's Algorithm: Implementation

Lets implement the solution.



Kruskal's Algorithm: Implementation

```
void makeParents(unordered_map<string, string> &parents)
    {
        for(auto town: g)
            parents[town.first] = town.first;
    }

    string findParent(unordered_map<string, string> parent, string town)
    {
        while(parent[town] != town)
            town = parent[town];
        return town;
    }

    void connect(unordered_map<string, string> &parent, string townA, string townB)
    {
        string townAParent = findParent(parent, townA);
        string townBParent = findParent(parent, townB);
        parent[townAParent] = townBParent;
    }
```

Kruskal's Algorithm: Implementation

```
int minimumCostKruskals(){
 typedef pair<int, pair<string, string>> twoEdgesCost;
 priority_queue<twoEdgesCost, vector<twoEdgesCost>, greater<twoEdgesCost>> pq;
 unordered_map<string, string> parents;
 makeParents(parents);
 int cost = 0;
 for (auto town : g)
   for (auto townEdge : town.second)
         pq.push({townEdge.first, {town.first, townEdge.second}});
 while (!pq.empty())
 {
   twoEdgesCost townEdge = pq.top();
   pq.pop();
   if (findParent(parents ,townEdge.second.first) != findParent(parents, townEdge.second.second))
   {
         cost = cost + townEdge.first;
         connect(parents, townEdge.second.first, townEdge.second.second);
   }
 }
 return cost;
}
```

MST Algorithms

Minimum Spanning Tree	Time Complexity	Space Complexity
	Worst Case	Worst Case
Prim's Algorithm	O(|E| * log(|V|))	O(|E| + |V|)
Kruskal's Algorithm	O(|E| * log(|E|))	O(|E| + |V|)

Learning Objective

Students should be able to **implement Minimum Spanning Tree** from the graphs to solve real life problems.

Conclusion

Sr.No.	Prim's Algorithm	Kruskal's Algorithm
1	This algorithm begins to construct the shortest spanning tree from any vertex in the graph.	This algorithm begins to construct the shortest spanning tree from the vertex having the lowest weight in the graph.
2	To obtain the minimum distance, it traverses one node more than one time.	It crosses one node only one time.
3	In Prim's algorithm, all the graph elements must be connected.	Kruskal's algorithm may have disconnected graphs.
4	When it comes to dense graphs, the Prim's algorithm runs faster.	When it comes to sparse graphs, Kruskal's algorithm runs faster.