French-Azerbaijani University

Network Algorithms

Optimal Path for Delivery Services

Yusif Ibrahimov — Saida Sideif-zada — Fahmin Guliyev — Gulshan Mustafayeva

January 3, 2021

Abstract

The objective of this project is to study the network of restaurants in Baku with the help of the Graph Theory.. To perform the analysis analysis, we need to construct a network of restaurants and calculate the metrics of the graph. The purpose of analysis is to find optimal paths for delivery services, so that one can visit as much as possible number of restaurants within short distance.Together with that, another interesting problem in graph theory called Travelling Salesman Problem can be proposed using by the delivery companies in order to keep the efficiency.

1 Visualization of the Dataset

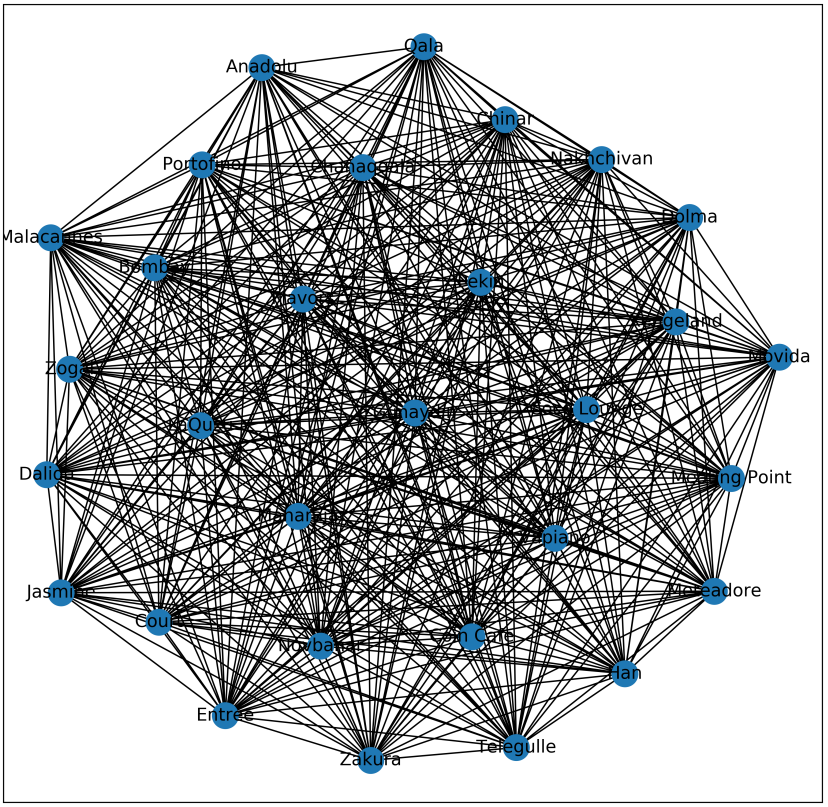
Simple data set is generated by ourselves containing the source restaurant, target restaurant and distance between them in kilometers referring to the list of the google maps.
Data set contains 30 different restaurants located in the Baku city. Here is the top 10 elements of our data set.

	source	target	dist
0	Coin Cafe	Jasmine	4.25
1	Coin Cafe	Telegulle	4.92
2	Coin Cafe	Bombay	3.58
3	Coin Cafe	Movida	2.59
4	Coin Cafe	Nakhchivan	2.57

According to the data set, the complete graph constructed with the help of the "networkx" library.

The graph contains 30 vertices. Since, it's undirected and complete graph, there is maximum the number of edges should be 435 ($C_n^2 = \frac{n*(n-1)}{2} = \frac{30*29}{2} = 435$)

Mathematical demonstration of the graph: $G = (V, E)$, where :
V => set of vertices: V = { 'Anadolu','Bombay','Chanaqqala','Chinar','Coco Lounge','Coin Cafe','Dalida','Dolm
'Entree','Flavors','Gunaydin','Han','Jasmine','LaQuzu','Maharaja','Malacannes','Meating Point','Meteadore',
'Movida','Nakhchivan','Novbahar','Pekin','Portofino','Qala','Telegulle','Vapiano','Xengeland','Zakura','Zogal'}
$|V| = 30$, E=> is set of all edges and $|E| = 435$
Here is the visualization of the graph.



2 Traversing the graph

Graph traversing is nothing more than moving all the nodes. For undirected graphs we follow all the edges, for directed graph, we follow only the out edges.We generaaly use two types of the traversing:

- Breadth First Search: **BFS**
- Depth First Search: **DFS**

2.1 Breadth First Search

We start from the root node (which we initialize) or some arbitrary node of a graph,,and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.
Here's the pseudocode of the BFS algorithm

```
BFS(graph,s)
   queue = {s}
   visited = {s}
   while queue is not empty:
```

```
u = enque()
print(u)
for every adj of u:
     if adj is not visited
     visited.append(adj)
     queue.append(adj)
```

BFS can be used to solve many problems including finding single source shortest path, solving puzzle games like Rubic's cube and etc. If we apply the BFS to our graph, we will get:

'Coin Cafe => Jasmine => Telegulle => Bombay => Movida => Nakhchivan => Portofino => Meteadore => Malacannes => Xengeland => Dolma => Entree => Novbahar => Pekin => Anadolu => Meating Point => Chanaqqala => Chinar => Gunaydin => Zakura => Flavors => Qala => Zogal => Coco Lounge => Han => Vapiano => Maharaja => Dalida => LaQuzu => Court'

2.2 Depth First Search

It starts from the starting node and goes possible down in a given path, and then backtracks until it finds an unvisited node, and then explores it. The algorithm does this until the graph has been visited completely.
Here's the pseudocode of the DFS algorithm

```
DFS(graph,s)
   stack = {s}
   visited = {s}
   while stack is not empty:
       u = pop()
       print(u)
       for every adj of u:
            if adj is not visited
            visited.append(adj)
            stack.append(adj)
```

DFS can be used to solve many problems like, sudoku, maze, some scheduling problems like topological sorting and etc.
If we apply the DFS algorithm to our graph, we will get the following output:

'Coin Cafe => Court => LaQuzu => Dalida => Maharaja => Vapiano => Han => Coco Lounge => Zogal => Qala => Flavors => Zakura => Gunaydin => Chinar => Chanaqqala => Meating Point =>

Anadolu => Pekin => Novbahar => Entree => Dolma => Xengeland => Malacannes => Meteadore => Portofino => Nakhchivan => Movida => Bombay => Telegulle => Jasmine'

3 Common metrics of centrality

In graph theory, we generally use the **centrality** to identify the important vertices of the graph. There are some metrics that help us to measure the centrality. They are:

- Degree Connectivity
- Degree Centrality
- Closeness Centrality
- Betweenness Centrality
- Network Density
- Network Diameter
- Network Average Path Length

3.1 Degree Connectivity

The degree of connectivity to a node in a network is a measure of the number of in and out links the node has to other nodes.
The value for our graph is: **29**
As the node connectivity is equal to the minimum number of nodes that must be removed to disconnect G, having complete graph, we got the value of node connectivity one less than the number of nodes.

3.2 Degree Centrality

Degree Centrality metric defines importance of a node in a graph as being measured based on its degree i.e the higher the degree of a node, the more important it is in a graph. Mathematically, Degree Centrality is defined as D(i) for a node "i" as below:

$$D(i) = \sum_j m(i,j), m(i,j) = 1 \tag{1}$$

if there's a link from i to j.

The result for our graph is:
'Coin Cafe': 1.0, 'Jasmine': 1.0, 'Telegulle': 1.0, 'Bombay': 1.0, 'Movida': 1.0, 'Nakhchivan': 1.0,

'Portofino': 1.0, 'Meteadore': 1.0, 'Malacannes': 1.0, 'Xengeland': 1.0, 'Dolma': 1.0, 'Entree': 1.0, 'Novbahar': 1.0, 'Pekin': 1.0, 'Anadolu': 1.0, 'Meating Point': 1.0, 'Chanaqqala': 1.0, 'Chinar': 1.0, 'Gunaydin': 1.0, 'Zakura': 1.0, 'Flavors': 1.0, 'Qala': 1.0, 'Zogal': 1.0, 'Coco Lounge': 1.0, 'Han': 1.0, 'Vapiano': 1.0, 'Maharaja': 1.0, 'Dalida': 1.0, 'LaQuzu': 1.0, 'Court': 1.0

The higher a value of network centrality, the more centered is a node. Every node of our graph has the maximum network centrality, as they are all of the same high importance. The values are adjusted in the scale from 0 to 1, as the maximum value is 29, for each node the value of network centrality is divided to the maximum value, so all nodes having 29 edges got centrality equal to 1.

3.3 Closeness Centrality

Closeness Centrality metric defines the importance of a node in a graph as being measured by how close it is to all other nodes in the graph. For a node, it is defined as the sum of the geodesic distance between that node to all other nodes in the network. Mathematically, Closeness Centrality C(i) of a node i in a graph can be defined as below:

$$C(i) = \sum_{j} d(i, j) \tag{2}$$

The answer for our graph is:

'Coin Cafe': 0.33375532282195886, 'Jasmine': 0.2982311805841218, 'Telegulle': 0.35587188612099646, 'Bombay': 0.3005492797181055, 'Movida': 0.426094622392007, 'Nakhchivan': 0.38476847552076415, 'Portofino': 0.30123610678300616, 'Meteadore': 0.329208763764332, 'Malacannes': 0.3092673562973232, 'Xengeland': 0.3398968588842006, 'Dolma': 0.3079537007539557, 'Entree': 0.39482641252552747, 'Novbahar': 0.31449951198351583, 'Pekin': 0.3862032227993074, 'Anadolu': 0.2570921985815603, 'Meating Point': 0.35975685398833884, 'Chanaqqala': 0.2963719979560552, 'Chinar': 0.36574599571194355, 'Gunaydin': 0.33799533799533804, 'Zakura': 0.37903542020650893, 'Flavors': 0.343073465041997, 'Qala': 0.33553164410505604, 'Zogal': 0.30111099574291345, 'Coco Lounge': 0.4059918801623968, 'Han': 0.3542201050445828, 'Vapiano': 0.3614157527417746, 'Maharaja': 0.34540257265364455, 'Dalida': 0.35587188612099646, 'LaQuzu': 0.4001103752759382, 'Court': 0.2889309554647803

From the values obtained we can easily see that the most important node is "Movida" as it has the highest betweenness centrality and thus, is the part of most paths.

3.4 Betweenness Centrality

Betweenness Centrality defines and measures the importance of a node in a network based upon how many times it occurs in the shortest path between all pairs of nodes in a graph.

Mathematically, Betweenness Centrality B(i) of a node i in a graph is defined as below:

$$B(i) = \sum_{a,b} \frac{g_{aib}}{g_{ab}} \tag{3}$$

a and b are the pair of nodes in our graph
g_{aib} is the number of shortest path from a to b passing through i
g_{ab} is the number of shortest path from a to b
The output for our graph is:

'Coin Cafe': 0.022167487684729065, 'Jasmine': 0.0, 'Telegulle': 0.013546798029556651,
'Bombay': 0.007389162561576354, 'Movida': 0.12807881773399016, 'Nakhchivan': 0.02586206896551724,
'Portofino': 0.0024630541871921183,
'Meteadore': 0.014778325123152709, 'Malacannes': 0.0024630541871921183, 'Xengeland': 0.022167487684729065
'Dolma': 0.0,
'Entree': 0.05172413793103448, 'Novbahar': 0.006157635467980296, 'Pekin': 0.08866995073891626, 'Anadolu':
0.0, 'Meating Point': 0.04187192118226601,
'Chanaqqala': 0.009852216748768473, 'Chinar': 0.06403940886699508, 'Gunaydin': 0.01600985221674877,
'Zakura': 0.024630541871921183,
'Flavors': 0.03201970443349754, 'Qala': 0.024630541871921183, 'Zogal': 0.007389162561576354, 'Coco
Lounge': 0.09113300492610837,
'Han': 0.03940886699507389, 'Vapiano': 0.04802955665024631, 'Maharaja': 0.024630541871921183, 'Dalida': 0.04187192118226601,
'LaQuzu': 0.10837438423645321, 'Court': 0.0024630541871921183

3.5 Network Density

Network Density describes the portion of the potential connections in a network that are actual connections. As we have a complete network, it has a density of **1**

3.6 Network Diameter

The diameter of a network refers to the length of the longest of all the computed shortest paths between all pair of nodes in the network. Since our graph is complete, every node is accessible from any other node, so the diameter is 1, which is the highest number of steps neeed to reach next node.

3.7 Network Average Path Length

The Network Mean Path Length is the average of the shortest path length, averaged over all pairs of nodes.The answer is **2.9607**

4 Minimum Spanning Trees - MST

Spanning Trees are the trees, (complete, acyclic graphs). They reach all the vertices.Each tree has its own cost, which is the some of all the weights of the tree. MST is the spanning tree where the cost is minimum among all the spanning trees.Note that, MSTs are not unique.
MSTs can be applied on many fields like: Image segmentation, Cluster Analysis , Handwriting recognition and etc.
For our case they are important because there exists access to all the restaurant and cost is the minimum. We can propose some algorithms to find the MST through the graph. They are:

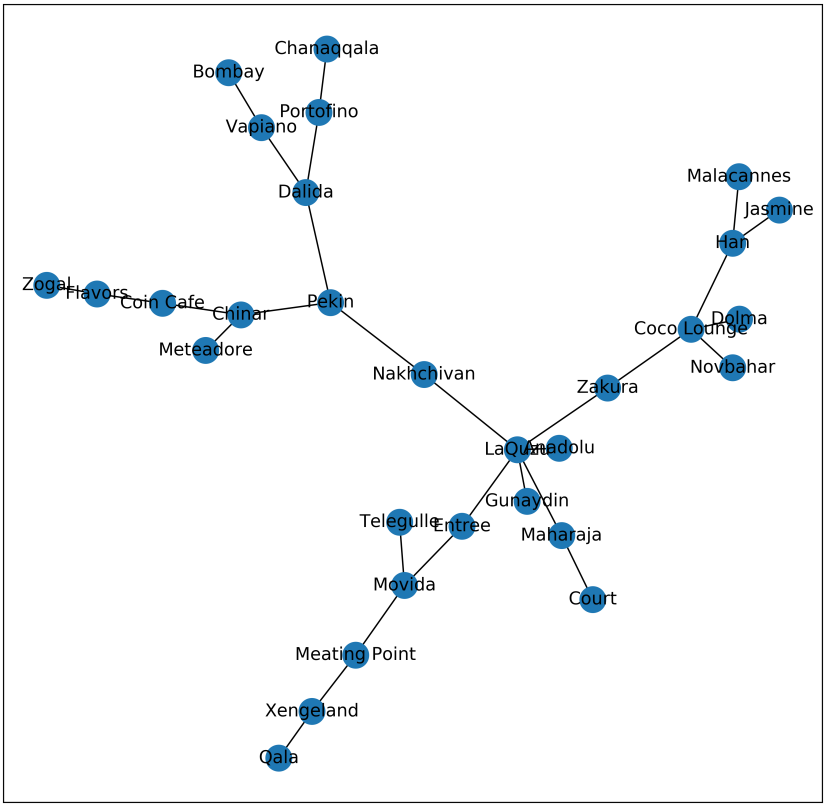
- Prim's Algorithm
- Kruskal's Algorithm

4.1 Prim's MST algorithm

Prim's algorithm is Greedy algorithms used in order to determine the MST of the graph. Greedy algorithm means that, it builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit.
The procedure to apply the Prim's algorithm is:

```
1. Take any vertex.
2. Make all the weights of \textbf{vertices} to infinity except
                        the chosen one.
3. For every adjacent vertices if the current weight of vertex
                      is more than the weight of the edge, then we set
                      it to the current edge.
4. Mark the current vertex as visited
5. Repeat these steps until getting the
                           increasing order of all weight of the vertices
```

If we apply the Prim's MST algorithm to our graph, we will get the following tree diagram.The **cost of the tree is 33.04**

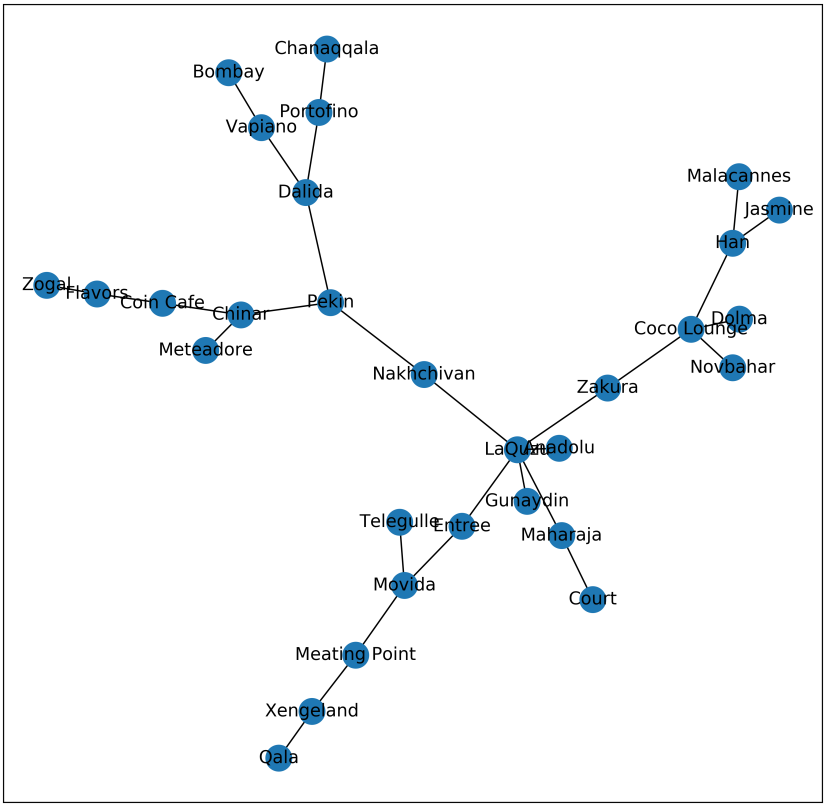


4.2 Kruskal's MST Algorithm

Kruskal's Algorithm is also a greedy algorithm that aims to define the MST The procedure to apply the Kruskal's algorithm is:

1. `Add increasing cost edges at each step`
2. `Avoid any cycle at each step`

If we apply the Kruskal's MST algorithm to our graph, we will get the following tree diagram.The **cost of the tree is 33.04**



4.3 Coclusion

We can see that, the images for both algorithm are not the same. It's not a problem, because we said that the MSTs are not unique and the costs are the same. But for this case, the Prim's Algorithm is much better than the Kruskal's algorithm.The reason is the execution time. Execution time for Prim's algorithm is **2.4ms** and for Kruskal's it's **4.88ms**

5 Shortest Path

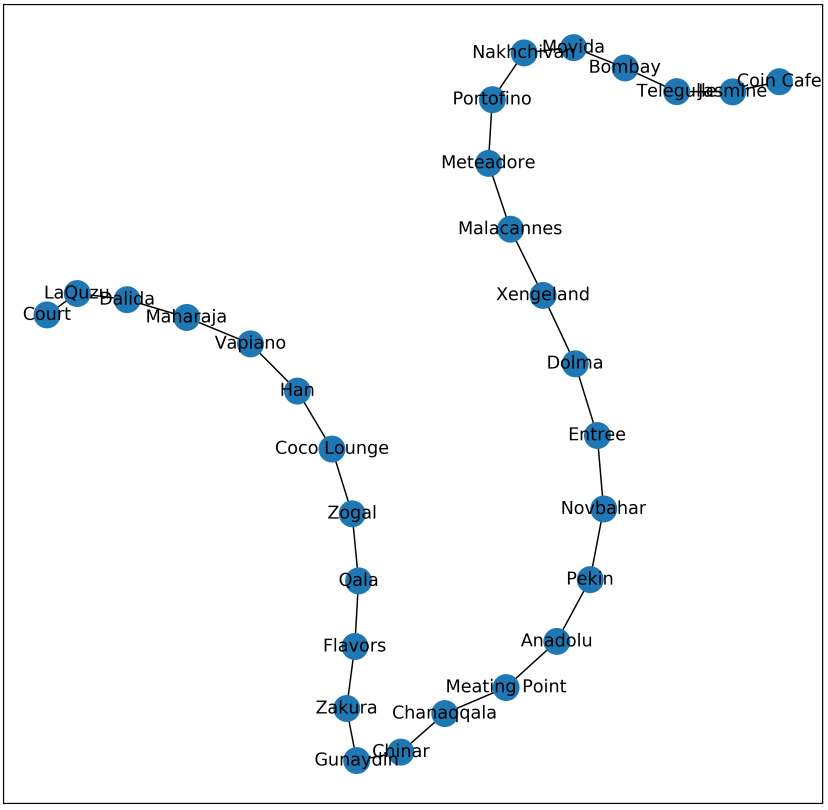
In some cases, we have to be asked about the shortest path between two restaurants. At this case, we can use either **Dijkstra's algorithm** and **Bellman Ford's Algorithm**. Both methods are quite efficient but the second one, is sensitive to the negative weights. But let's calculate shortest for each combination of the nodes and check the execution time. The execution time for the Dijkstra's Algorithm is **15.8** and for Bellman Ford's Algorithm is : **14.9**.But this may vary depending on the execution. However, the results are completely same for two methods.

6 Traveler Salesman Problem

Case study: I want a path that crosses all the vertices exactly once, and again I return my starting point. For example, I am an employee of Wolt delivery, I have to take all the meals from the other restaurants. And also, I want to spend the minimum volume of the fuel. So, I need the minimal distance. Does it looks like the **Hamiltonian path** which crosses all the vertices exact once, and in case of the **cycle** it return the starting point. Maybe, but I want the minimal distance. Let's first take one Hamiltonian path and calculate it's cost;

6.1 Hamiltonian Path

We said that the Hamiltonian path is the path that crosses from all the vertices exactly once. There can be multiple Hamiltonian path, so take one of them.The example is the following with the cost of **156.15**:

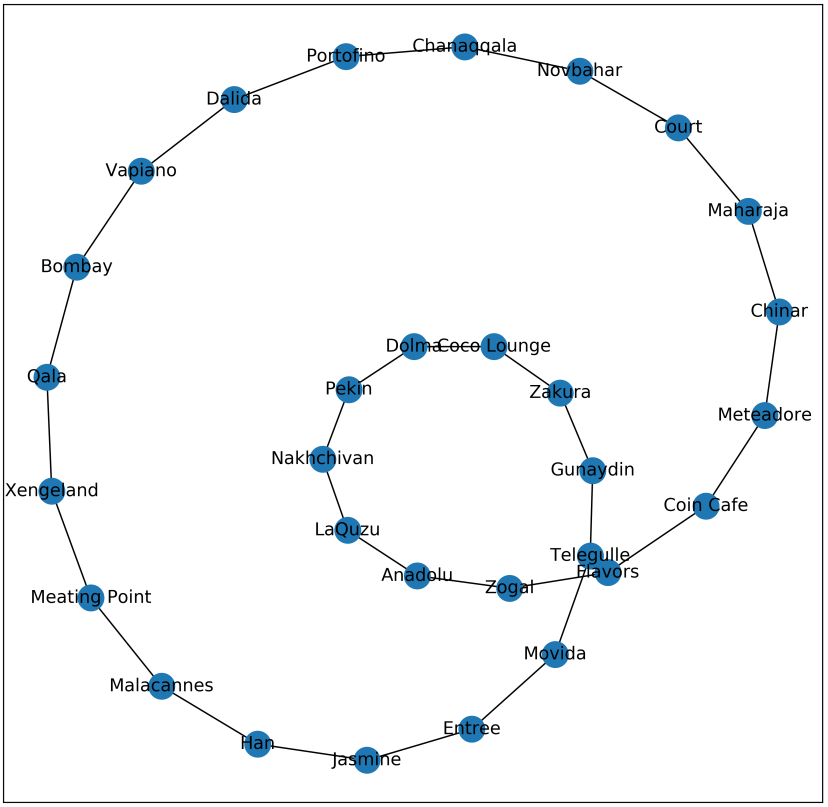


6.2 Traveler Salesman Problem

So we've defined what's the TSP. So, it must be in the minimum of the cost. There are some ways of the solving of the TSP.

- Naive Solution and Dynamic Programming
- Approximate using MST
- Using NP Hard
- Constructive heuristics
- The Algorithm of Christofides and Serdyukov and etc.

So as we said that, there are multiple complex solutions of the TSP problem which are the out of scope for now and they can be prepared as another research topic, however, we can use the written library for the problem called **tsp** library. To use this library, first we must create the destination matrix, then apply the "tsp" library. The result is perfect. The **minimal cost hamiltonian cycle** is the following cycle with a cost of **41.42**



As a result... We can see that, the cost of the TSP is strictly less than the cost of the Hamiltonian Path. So this is the difference.
And also, the difference between **TSP and MST is:** MST asks you to build a tree that connects all cities and has minimum total weight, while the TSP asks you to find a trip that visits all cities with minimum total weight (and possibly coming back to your starting point)