DAA LAB - 07 REPORT

**Aim- In this lab, we would be solving problems based on graphs.

**Explanation-

1. **Kruskal's algorithm :** Kruskal's algorithm creates a minimum spanning tree from a weighted undirected graph by adding edges in ascending order of weights till all the vertices are contained in it. Kruskal's algorithm gets greedy as it chooses edges in increasing order of weights. The algorithm makes sure that the addition of new edges to the spanning tree does not create a cycle within it.

*Algorithm:

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree.

* Pseudocode:

```
KRUSKAL(G):
A = ∅
For each vertex v ∈ G.V:
   MAKE-SET(v)
For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
   if FIND-SET(u) ≠ FIND-SET(v):
   A = A ∪ {(u, v)}
   UNION(u, v)
return A
```

1.Prim's Algorithm:

Prim's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which form a tree that includes every vertex has the minimum sum of weights among all the trees that can be formed from the graph.

*Algorithm:

1) Create a set *min_span_tree_set* that keeps track of vertices already included in MST.
2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE.
3)Assign key value as 0 for the first vertex so that it is picked first.
4) While min_span_tree_set doesn't include all vertices
a) Pick a vertex *u* which is not there in min_span_tree_set and has minimum key value.
b) Include *u* to min_span_tree_set.
c)Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u*-*v* is less than the previous key value of *v*, update the key value as weight of *u*-*v*
The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

* Pseudocode:

```
T = ∅;
U = { 1 };
while (U ≠ V)
   let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
   T = T ∪ {(u, v)}
   U = U ∪ {v}
```

3. Dijkstra's Algorithm:

Dijkstra's algorithm allows us to find the shortest path between any two *vertices of a graph.*

It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

*Algorithm:

1) Create a set *Set* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

3) While *Set* doesn't include all vertices

a) Pick a vertex u which is not there in *Set* and has minimum distance value.

b) Include u to *Set*.

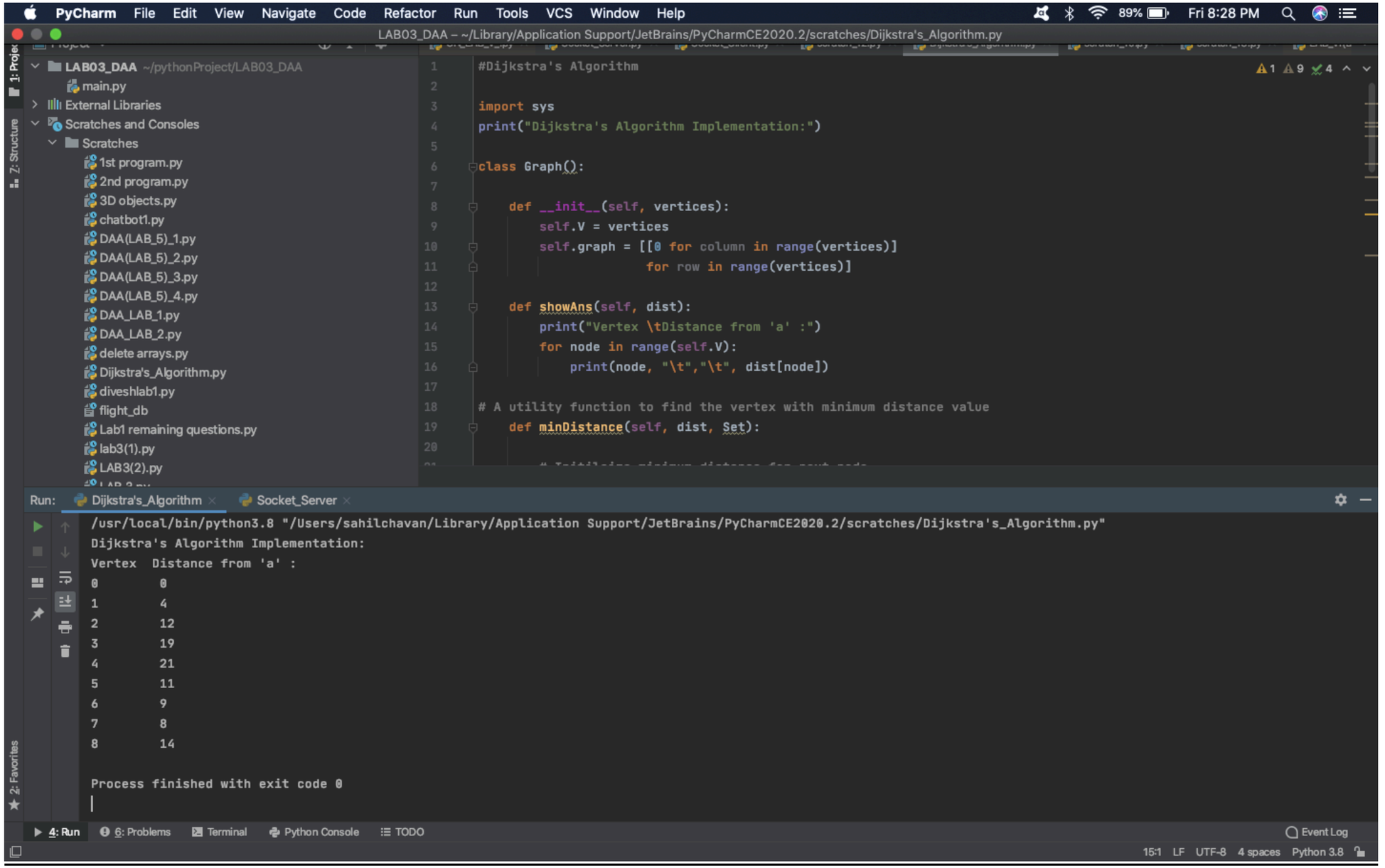
c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

*Pseudocode:

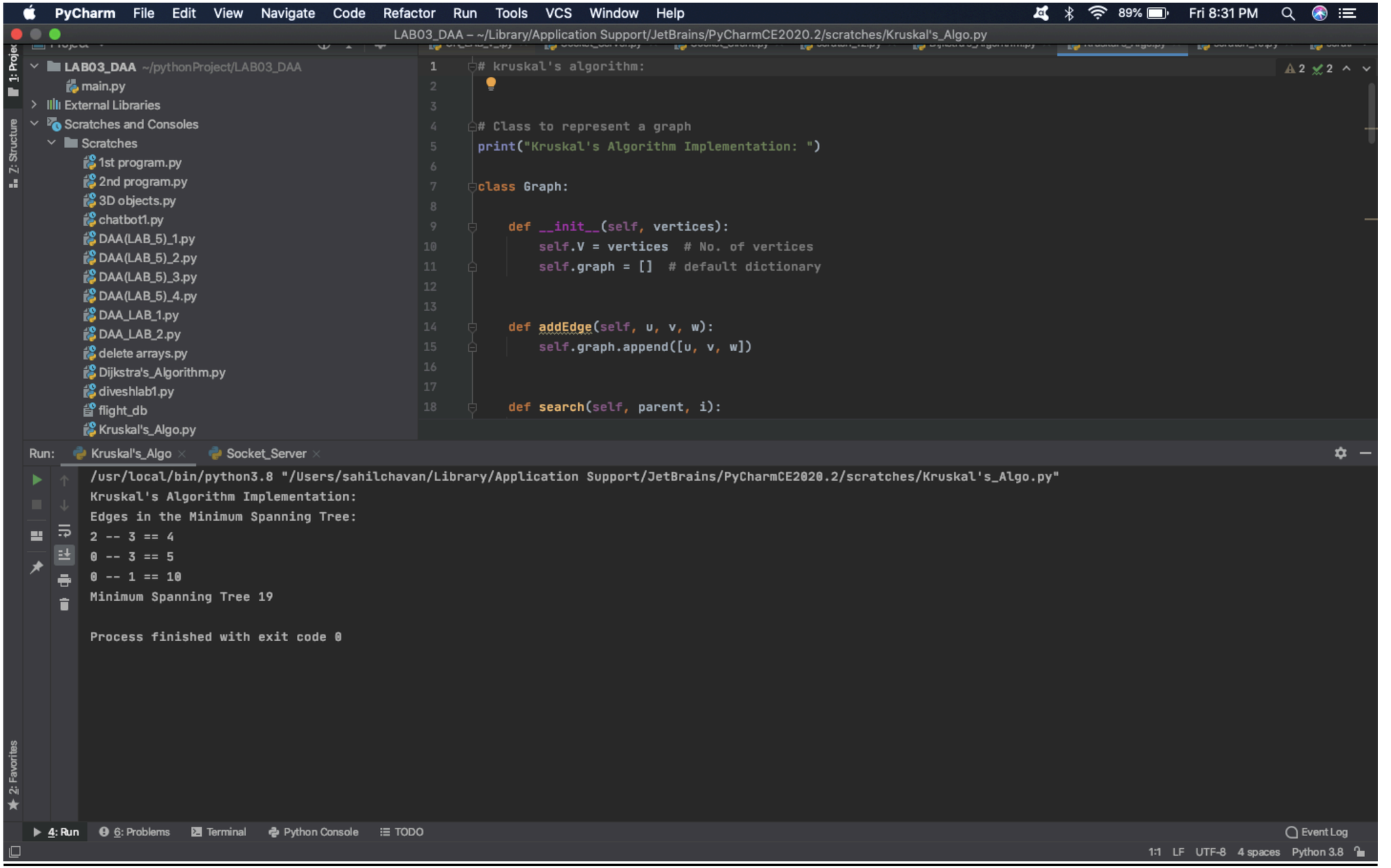
```
function dijkstra(G, S)
   for each vertex V in G
      distance[V] <- infinite
      previous[V] <- NULL
      If V != S, add V to Priority Queue Q
   distance[S] <- 0

   while Q IS NOT EMPTY
      U <- Extract MIN from Q
      for each unvisited neighbour V of U
         tempDistance <- distance[U] + edge_weight(U, V)
         if tempDistance < distance[V]
   return distance[], previous[]
```

***OUTPUT :**

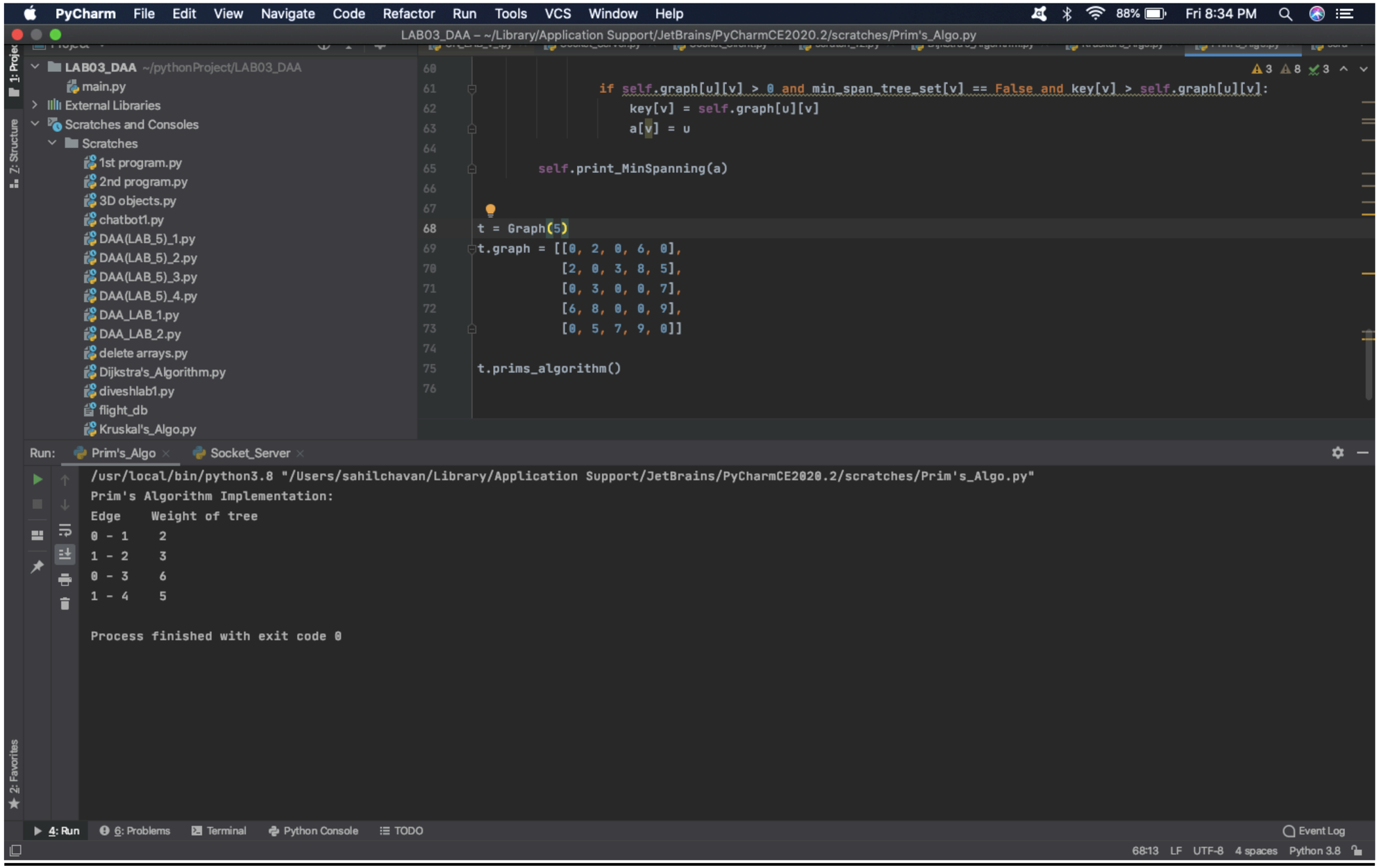
***Dijktra's Algorithm :**



*Kruskal's Algorithm :



*Prim's Algorithm :



__*Conclusion-*__ Therefore, we can implement,

__1) Kruskal's Algorithm__

__2) Prim's Algorithm and__

__3) Dijkstra's Algorithms__

NAME : SAHIL CHAVAN

PRN : 20190802042