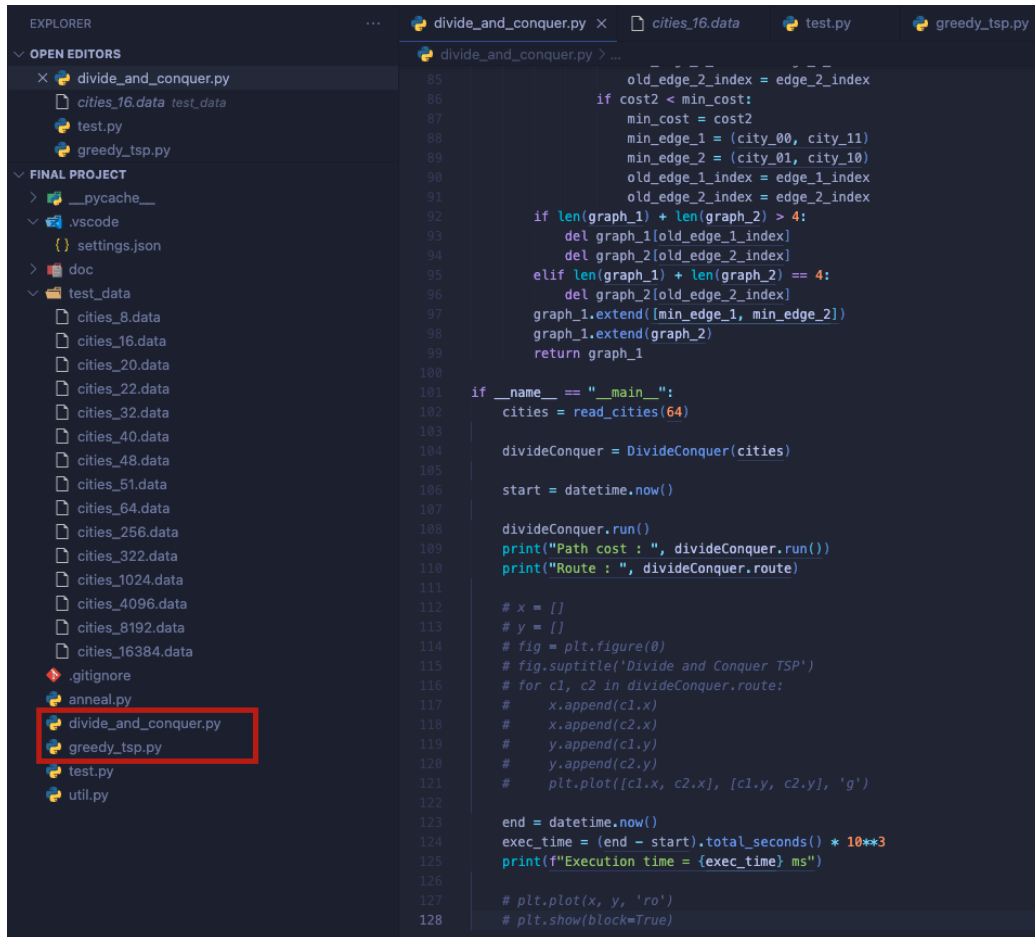


1. Running the Traveling Salesman Problem using 2 algorithms

Step 1 : Pick 1 out of 2 algorithms you want to execute the Travelling salesman problem.



```
85         old_edge_2_index = edge_2_index
86         if cost2 < min_cost:
87             min_cost = cost2
88             min_edge_1 = (city_00, city_11)
89             min_edge_2 = (city_01, city_10)
90             old_edge_1_index = edge_1_index
91             old_edge_2_index = edge_2_index
92         if len(graph_1) + len(graph_2) > 4:
93             del graph_1[old_edge_1_index]
94             del graph_2[old_edge_2_index]
95         elif len(graph_1) + len(graph_2) == 4:
96             del graph_2[old_edge_2_index]
97             graph_1.extend([min_edge_1, min_edge_2])
98             graph_1.extend(graph_2)
99             return graph_1
100
101 if __name__ == "__main__":
102     cities = read_cities(64)
103
104     divideConquer = DivideConquer(cities)
105
106     start = datetime.now()
107
108     divideConquer.run()
109     print("Path cost : ", divideConquer.run())
110     print("Route : ", divideConquer.route)
111
112     # x = []
113     # y = []
114     # fig = plt.figure(0)
115     # fig.suptitle('Divide and Conquer TSP')
116     # for c1, c2 in divideConquer.route:
117     #     x.append(c1.x)
118     #     x.append(c2.x)
119     #     y.append(c1.y)
120     #     y.append(c2.y)
121     #     plt.plot([c1.x, c2.x], [c1.y, c2.y], 'g')
122
123     end = datetime.now()
124     exec_time = (end - start).total_seconds() * 10**3
125     print(f"Execution time = {exec_time} ms")
126
127     # plt.plot(x, y, 'ro')
128     # plt.show(block=True)
```

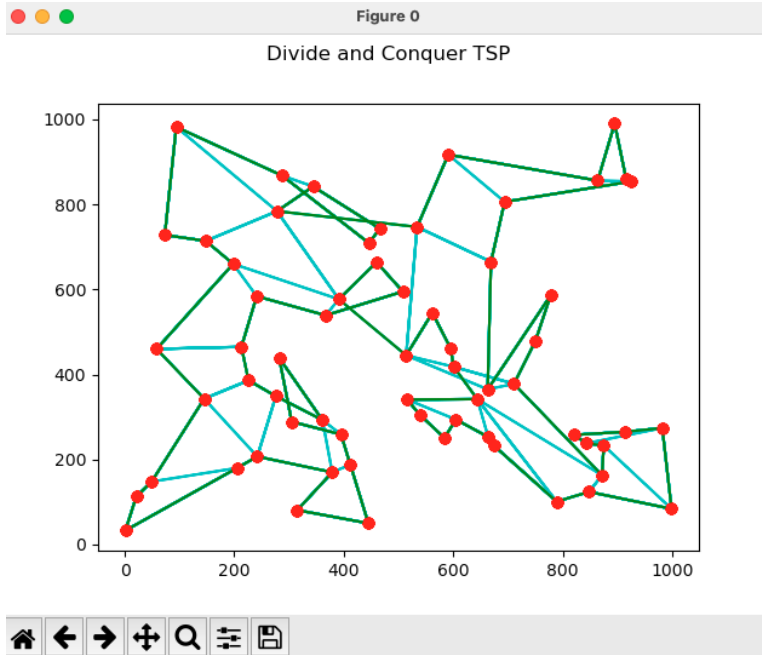
Step 2 : We provided the user with the number of cities they want to solve during the traveling salesman problem. Therefore, they can run it using both algorithms. In addition, to insert the city data using the template (“coordinate (space) coordinate”). The user is required to enter the coordinates of both the arrival and departure cities in each data entry for the algorithm to run smoothly without any interference.

```
85         old_edge_2_index = edge_2_index
86         if cost2 < min_cost:
87             min_cost = cost2
88             min_edge_1 = (city_00, city_11)
89             min_edge_2 = (city_01, city_10)
90             old_edge_1_index = edge_1_index
91             old_edge_2_index = edge_2_index
92         if len(graph_1) + len(graph_2) > 4:
93             del graph_1[old_edge_1_index]
94             del graph_2[old_edge_2_index]
95         elif len(graph_1) + len(graph_2) == 4:
96             del graph_2[old_edge_2_index]
97         graph_1.extend([min_edge_1, min_edge_2])
98         graph_1.extend(graph_2)
99         return graph_1
100
101 if __name__ == "__main__":
102     cities = read_cities(64)
103
104     divideConquer = DivideConquer(cities)
105
106     start = datetime.now()
107
108     divideConquer.run()
109     print("Path cost : ", divideConquer.run())
110     print("Route : ", divideConquer.route)
111
112     # x = []
113     # y = []
114     # fig = plt.figure(0)
115     # fig.suptitle('Divide and Conquer TSP')
116     # for c1, c2 in divideConquer.route:
117     #     x.append(c1.x)
118     #     x.append(c2.x)
119     #     y.append(c1.y)
120     #     y.append(c2.y)
121     #     plt.plot([c1.x, c2.x], [c1.y, c2.y], 'g')
122
123     end = datetime.now()
124     exec_time = (end - start).total_seconds() * 10**3
125     print(f"Execution time = {exec_time} ms")
126
127     # plt.plot(x, y, 'ro')
128     # plt.show(block=True)
```

Step 3 : After picking one algorithm and how many cities the user want to solve, we can execute the algorithm

```
85         old_edge_2_index = edge_2_index
86         if cost2 < min_cost:
87             min_cost = cost2
88             min_edge_1 = (city_00, city_11)
89             min_edge_2 = (city_01, city_10)
90             old_edge_1_index = edge_1_index
91             old_edge_2_index = edge_2_index
92         if len(graph_1) + len(graph_2) > 4:
93             del graph_1[old_edge_1_index]
94             del graph_2[old_edge_2_index]
95         elif len(graph_1) + len(graph_2) == 4:
96             del graph_2[old_edge_2_index]
97         graph_1.extend([min_edge_1, min_edge_2])
98         graph_1.extend(graph_2)
99         return graph_1
100
101 if __name__ == "__main__":
102     cities = read_cities(64)
103
104     divideConquer = DivideConquer(cities)
105
106     start = datetime.now()
107
108     divideConquer.run()
109     print("Path cost : ", divideConquer.run())
110     print("Route : ", divideConquer.route)
111
112     # x = []
113     # y = []
114     # fig = plt.figure(0)
115     # fig.suptitle('Divide and Conquer TSP')
116     # for c1, c2 in divideConquer.route:
117     #     x.append(c1.x)
118     #     x.append(c2.x)
119     #     y.append(c1.y)
120     #     y.append(c2.y)
121     #     plt.plot([c1.x, c2.x], [c1.y, c2.y], 'g')
122
123     end = datetime.now()
124     exec_time = (end - start).total_seconds() * 10**3
125     print(f"Execution time = {exec_time} ms")
126
127     # plt.plot(x, y, 'ro')
128     # plt.show(block=True)
```

Step 4: This window will pop off when the user executes the algorithm successfully.



In the terminal we also provide the user with the route and also the execution time to run the algorithm.

```
Route : [(1.0, 33.0), (21.0, 113.0), (1.0, 33.0), (205.0, 180.0), (21.0, 113.0), (49.0, 148.0), (49.0, 148.0), (145.0, 343.0), (205.0, 180.0), (242.0, 207.0), (276.0, 349.0), (276.0, 349.0), (361.0, 293.0), (445.0, 50.0), (315.0, 81.0), (445.0, 50.0), (412.0, 188.0), (315.0, 81.0), (379.0, 170.0), (412.0, 188.0), (395.0, 259.0), (395.0, 259.0), (305.0, 289.0), (305.0, 289.0), (282.0, 439.0), (361.0, 293.0), (282.0, 439.0), (145.0, 343.0), (58.0, 460.0), (226.0, 386.0), (213.0, 465.0), (58.0, 460.0), (198.0, 660.0), (213.0, 465.0), (241.0, 584.0), (241.0, 584.0), (367.0, 539.0), (367.0, 539.0), (508.0, 595.0), (391.0, 577.0), (460.0, 664.0), (460.0, 664.0), (508.0, 595.0), (198.0, 660.0), (148.0, 714.0), (148.0, 714.0), (73.0, 728.0), (73.0, 728.0), (94.0, 982.0), (278.0, 784.0), (344.0, 842.0), (94.0, 982.0), (287.0, 868.0), (287.0, 868.0), (446.0, 709.0), (344.0, 842.0), (466.0, 743.0), (446.0, 709.0), (466.0, 743.0), (391.0, 577.0), (514.0, 445.0), (278.0, 784.0), (534.0, 747.0), (584.0, 251.0), (605.0, 294.0), (584.0, 251.0), (539.0, 305.0), (539.0, 305.0), (516.0, 341.0), (605.0, 294.0), (664.0, 253.0), (516.0, 341.0), (644.0, 342.0), (789.0, 101.0), (674.0, 233.0), (674.0, 233.0), (664.0, 253.0), (789.0, 101.0), (848.0, 124.0), (848.0, 124.0), (999.0, 84.0), (872.0, 163.0), (875.0, 233.0), (875.0, 233.0), (843.0, 238.0), (999.0, 84.0), (982.0, 274.0), (821.0, 258.0), (843.0, 238.0), (821.0, 258.0), (913.0, 265.0), (913.0, 265.0), (982.0, 274.0), (644.0, 342.0), (601.0, 418.0), (872.0, 163.0), (711.0, 378.0), (601.0, 418.0), (595.0, 462.0), (514.0, 445.0), (562.0, 545.0), (595.0, 462.0), (562.0, 545.0), (663.0, 365.0), (778.0, 586.0), (711.0, 378.0), (750.0, 478.0), (750.0, 478.0), (778.0, 586.0), (663.0, 365.0), (669.0, 665.0), (669.0, 665.0), (694.0, 806.0), (534.0, 747.0), (591.0, 917.0), (694.0, 806.0), (926.0, 854.0), (591.0, 917.0), (864.0, 856.0), (926.0, 854.0), (916.0, 860.0), (864.0, 856.0), (895.0, 990.0), (916.0, 860.0), (895.0, 990.0)]
Execution time = 7930.929 ms
```