

Nama : Muhammad Fakhrol Amin

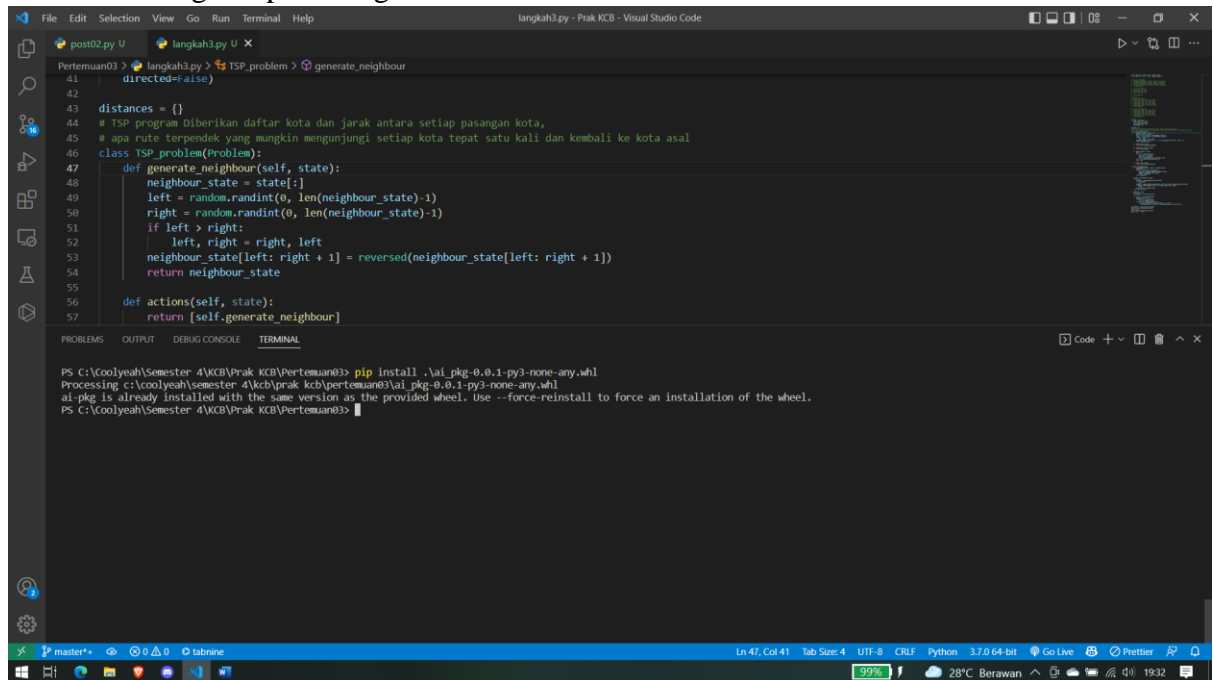
NIM : 2000018277

Slot : Sabtu 07.30

LANGKAH PRAKTIKUM KECERDASAN BUATAN

PERTEMUAN 3

1. Lakukanlah tugas 1 pada langkah 1

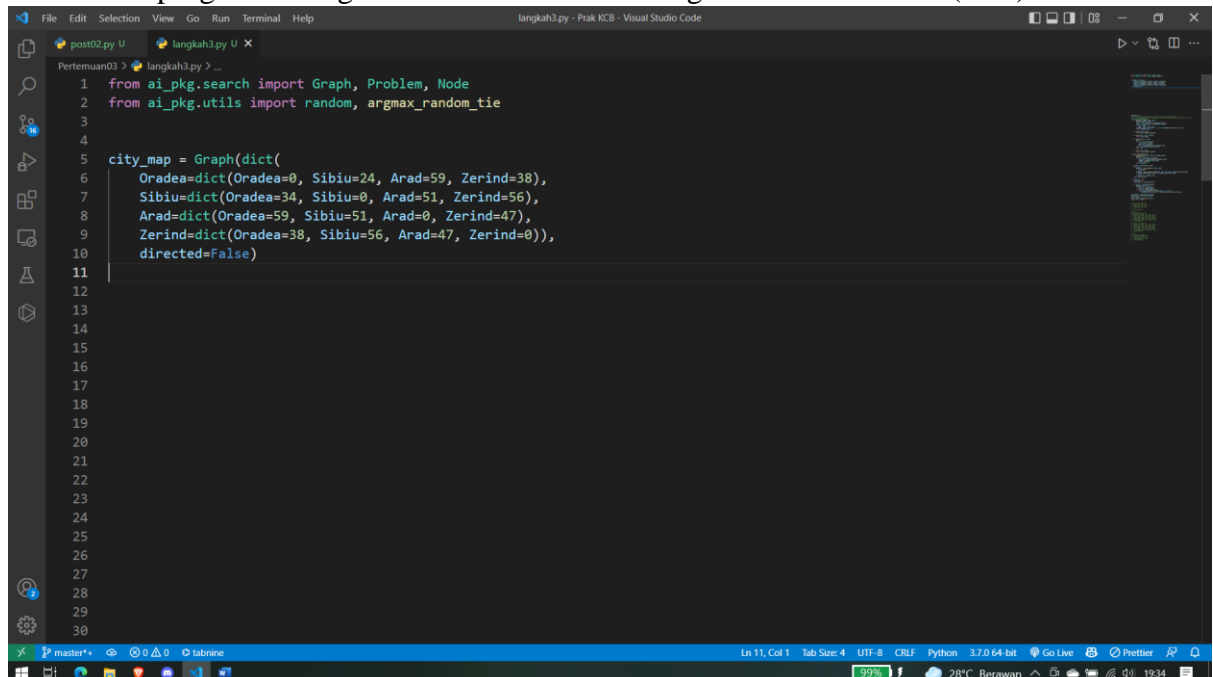


The screenshot shows a Visual Studio Code editor with a file named `langkah3.py` open. The code defines a `TSP_problem` class with methods `generate_neighbour` and `actions`. The `generate_neighbour` method generates a new state by randomly selecting two cities in the current state and reversing the path between them. The `actions` method returns a list containing the generated neighbour state. Below the code editor, the terminal window shows the command `pip install .\ai_pkg-0.0.1-py3-none-any.whl` being executed, which successfully installs the package.

```
41 def generate_neighbour(self, state):
42     directed=False
43
44     distances = {}
45     # TSP program Diberikan daftar kota dan jarak antara setiap pasangan kota,
46     # apa rute terpendek yang mungkin mengunjungi setiap kota tepat satu kali dan kembali ke kota asal
47     class TSP_problem(problem):
48         def generate_neighbour(self, state):
49             neighbour_state = state[:]
50             left = random.randint(0, len(neighbour_state)-1)
51             right = random.randint(0, len(neighbour_state)-1)
52             if left > right:
53                 left, right = right, left
54             neighbour_state[left:right+1] = reversed(neighbour_state[left:right+1])
55             return neighbour_state
56
57         def actions(self, state):
58             return [self.generate_neighbour(state)]
```

```
PS C:\Coolyeh\Semester 4\KCB\Prak KCB\Peraturan03> pip install .\ai_pkg-0.0.1-py3-none-any.whl
Processing c:\cool\yeh\semester 4\kcb\prak kcb\peraturan03\ai_pkg-0.0.1-py3-none-any.whl
ai_pkg is already installed with the same version as the provided wheel. Use --force-reinstall to force an installation of the wheel.
PS C:\Coolyeh\Semester 4\KCB\Prak KCB\Peraturan03>
```

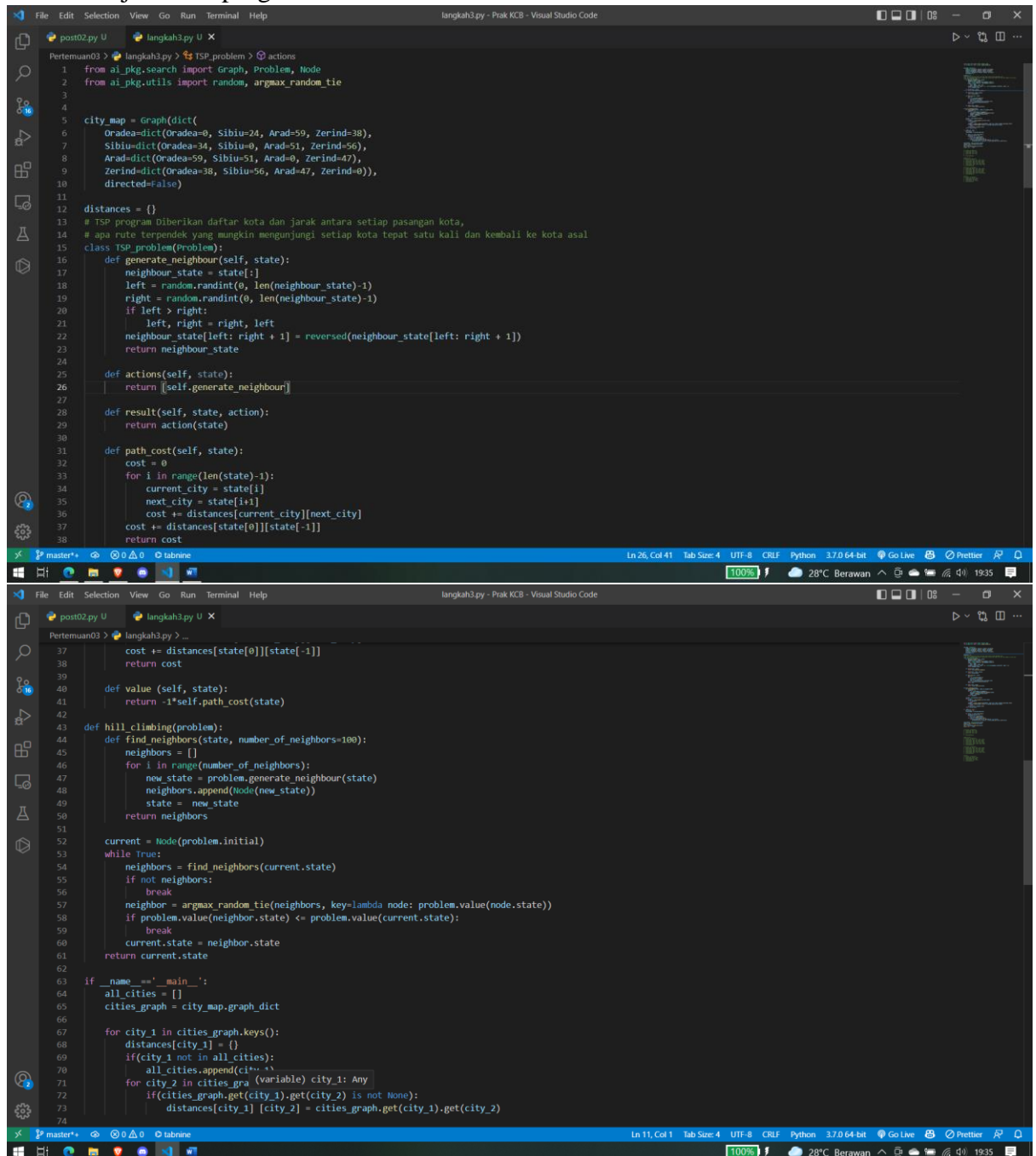
2. Ketik kode program listing 3.1 untuk kasus Travelling Salesman Problem(TSP)



The screenshot shows a Visual Studio Code editor with a file named `langkah3.py` open. The code imports `Graph`, `Problem`, and `Node` from `ai_pkg.search` and `random`, `argmax_random_tie` from `ai_pkg.utils`. It then defines a `city_map` as a `Graph` object with a dictionary of cities and their distances. The cities are Oradea, Sibiu, Arad, and Zerind, with their respective distances to each other.

```
1 from ai_pkg.search import Graph, Problem, Node
2 from ai_pkg.utils import random, argmax_random_tie
3
4
5 city_map = Graph(dict(
6     Oradea=dict(Oradea=0, Sibiu=24, Arad=59, Zerind=38),
7     Sibiu=dict(Oradea=34, Sibiu=0, Arad=51, Zerind=56),
8     Arad=dict(Oradea=59, Sibiu=51, Arad=0, Zerind=47),
9     Zerind=dict(Oradea=38, Sibiu=56, Arad=47, Zerind=0)),
10    directed=False)
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

3. Jalankan program Travelling Salesman Problem dengan cara:
 - a. Unduh custom package melalui URL yang diberikan asisten dan instal package tersebut dan import modul yang ada pada package untuk menggunakannya dalam program.
 - b. Tuliskan program yang ada pada contoh kasus gambar 3.1 dan tambahkan listing 3,4 untuk menjalankan program.



```
1 from ai_pkg.search import Graph, Problem, Node
2 from ai_pkg.utils import random, argmax_random_tie
3
4
5 city_map = Graph(dict(
6     Oradea=dict(Oradea=0, Sibiu=24, Arad=99, Zerind=38),
7     Sibiu=dict(Oradea=24, Sibiu=0, Arad=51, Zerind=56),
8     Arad=dict(Oradea=99, Sibiu=51, Arad=0, Zerind=47),
9     Zerind=dict(Oradea=38, Sibiu=56, Arad=47, Zerind=0)),
10    directed=False)
11
12 distances = {}
13 # TSP program Diberikan daftar kota dan jarak antara setiap pasangan kota,
14 # apa rute terpendek yang mungkin mengunjungi setiap kota tepat satu kali dan kembali ke kota asal
15 class TSP_problem(Problem):
16     def generate_neighbour(self, state):
17         neighbour_state = state[:]
18         left = random.randint(0, len(neighbour_state)-1)
19         right = random.randint(0, len(neighbour_state)-1)
20         if left > right:
21             left, right = right, left
22         neighbour_state[left:right+1] = reversed(neighbour_state[left:right+1])
23         return neighbour_state
24
25     def actions(self, state):
26         return [self.generate_neighbour]
27
28     def result(self, state, action):
29         return action(state)
30
31     def path_cost(self, state):
32         cost = 0
33         for i in range(len(state)-1):
34             current_city = state[i]
35             next_city = state[i+1]
36             cost += distances[current_city][next_city]
37         cost += distances[state[0]][state[-1]]
38         return cost
39
40     def value(self, state):
41         return -1*self.path_cost(state)
42
43     def hill_climbing(problem):
44         def find_neighbors(state, number_of_neighbors=100):
45             neighbors = []
46             for i in range(number_of_neighbors):
47                 new_state = problem.generate_neighbour(state)
48                 neighbors.append(Node(new_state))
49                 state = new_state
50             return neighbors
51
52         current = Node(problem.initial)
53         while True:
54             neighbors = find_neighbors(current.state)
55             if not neighbors:
56                 break
57             neighbor = argmax_random_tie(neighbors, key=lambda node: problem.value(node.state))
58             if problem.value(neighbor.state) <= problem.value(current.state):
59                 break
60             current.state = neighbor.state
61         return current.state
62
63 if __name__ == '__main__':
64     all_cities = []
65     cities_graph = city_map.graph_dict
66
67     for city_1 in cities_graph.keys():
68         distances[city_1] = {}
69         if city_1 not in all_cities:
70             all_cities.append(city_1)
71         for city_2 in cities_graph[city_1].keys():
72             if city_2 not in all_cities:
73                 all_cities.append(city_2)
74             distances[city_1][city_2] = cities_graph[city_1].get(city_2)
```

- c. Program TSP dapat dijalankan dengan cara membuat objek baru TSP_problem dengan parameter all_cities. Selanjutnya panggil fungsi hil_climbing dengan parameter objek yang baru saja dibuat.

```
60     current.state = neighbor.state
61     return current.state
62
63 if __name__ == '__main__':
64     all_cities = []
65     cities_graph = city_map.graph_dict
66
67     for city_1 in cities_graph.keys():
68         distances[city_1] = {}
69         if(city_1 not in all_cities):
70             all_cities.append(city_1)
71         for city_2 in cities_graph.keys():
72             if(cities_graph.get(city_1).get(city_2) is not None):
73                 distances[city_1][city_2] = cities_graph.get(city_1).get(city_2)
74
75 tsp_problem = TSP_problem(all_cities)
76 result = hill_climbing(tsp_problem)
77 print(r (variable) tsp_problem: TSP_problem)
78 cost = tsp_problem.path_cost(result)
79 print('cost: ', cost)
80
81
82
83
84
85
86
87
88
89
90
```

d. Jalankan program tersebut dan tampilkan hasilnya..

```
60     current.state = neighbor.state
61     return current.state
62
63 if __name__ == '__main__':
64     all_cities = []
65     cities_graph = city_map.graph_dict
66
67     for city_1 in cities_graph.keys():
68         distances[city_1] = {}
69         if(city_1 not in all_cities):
70             all_cities.append(city_1)
71         for city_2 in cities_graph.keys():
72             if(cities_graph.get(city_1).get(city_2) is not None):
73                 distances[city_1][city_2] = cities_graph.get(city_1).get(city_2)
74
75 tsp_problem = TSP_problem(all_cities)
76 result = hill_climbing(tsp_problem)
77 print(r (variable) tsp_problem: TSP_problem)
78 cost = tsp_problem.path_cost(result)
79 print('cost: ', cost)
80
81
82
83
84
85
86
87
88
89
90
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Coolyeah\Semester 4\KCB\Prak KCB> python -u "C:\Coolyeah\Semester 4\KCB\Prak KCB\Pertemuan03\langkah3.py"

['Oradea', 'Sibiu', 'Arad', 'Zerind']

cost: 160

PS C:\Coolyeah\Semester 4\KCB\Prak KCB>