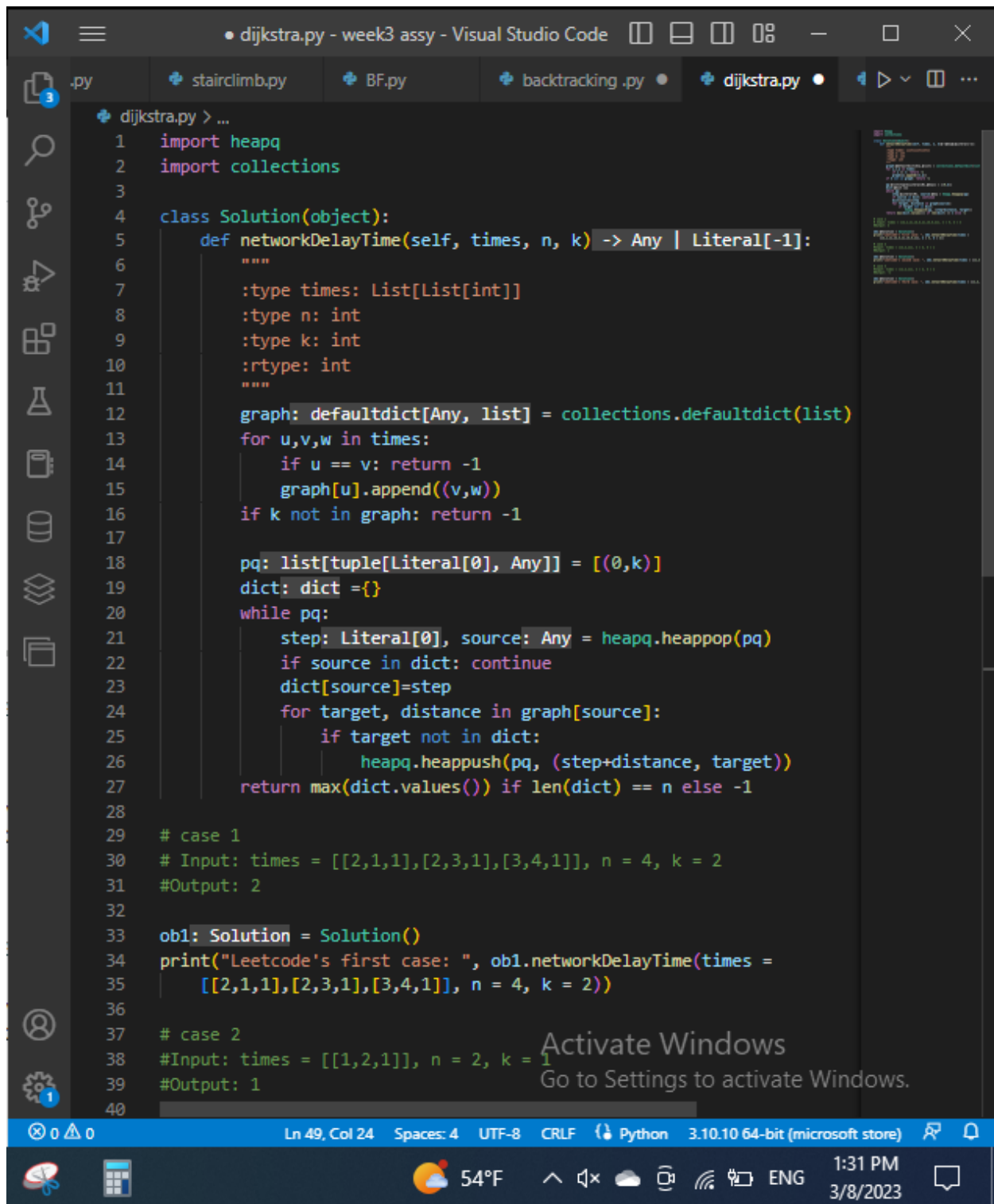


Vertex (accumulated path)	Initial () Next Step V_A	Step1 V_A (V_A) Next Step V_B	Step2 V_B (V_A, V_B) Next Step V_C	Step3 V_C (V_A, V_B, V_C) Next Step V_D	Step4 V_D (V_A, V_B, V_C, V_D) Next Step V_E	Step5 V_E (V_A, V_B, V_C, V_D, V_E) End at V_9
V_A	0	0	\emptyset	\emptyset	\emptyset	\emptyset
V_B	∞	1	1	4	4	4
V_C	∞	∞	3	3	3	3
V_D	∞	2	2	2	2	2
V_E	∞	∞	∞	11	5	5



```
dijkstra.py > ...
1 import heapq
2 import collections
3
4 class Solution(object):
5     def networkDelayTime(self, times, n, k) -> Any | Literal[-1]:
6         """
7         :type times: List[List[int]]
8         :type n: int
9         :type k: int
10        :rtype: int
11        """
12        graph: defaultdict[Any, list] = collections.defaultdict(list)
13        for u,v,w in times:
14            if u == v: return -1
15            graph[u].append((v,w))
16        if k not in graph: return -1
17
18        pq: list[tuple[Literal[0], Any]] = [(0,k)]
19        dict: dict = {}
20        while pq:
21            step: Literal[0], source: Any = heapq.heappop(pq)
22            if source in dict: continue
23            dict[source]=step
24            for target, distance in graph[source]:
25                if target not in dict:
26                    heapq.heappush(pq, (step+distance, target))
27        return max(dict.values()) if len(dict) == n else -1
28
29 # case 1
30 # Input: times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2
31 #Output: 2
32
33 ob1: Solution = Solution()
34 print("Leetcode's first case: ", ob1.networkDelayTime(times =
35     [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2))
36
37 # case 2
38 #Input: times = [[1,2,1]], n = 2, k = 1
39 #Output: 1
40
```

dijkstra.py - week3 assy - Visual Studio Code

dijkstra.py

stairclimb.py

BF.py

backtracking.py

dijkstra.py

dijkstra.py > ...

```
28
29 # case 1
30 # Input: times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2
31 #Output: 2
32
33 ob1: Solution = Solution()
34 print("Leetcode's first case: ", ob1.networkDelayTime(times =
35     [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2))
36
37 # case 2
38 #Input: times = [[1,2,1]], n = 2, k = 1
39 #Output: 1
40
41 ob1: Solution = Solution()
42 print("Leetcode's second case: ", ob1.networkDelayTime(times = [[1,2,1
43
44 # case 3
45 #Input: times = [[1,2,1]], n = 2, k = 2
46 #Output: -1
47
48 ob1: Solution = Solution()
49 print("Leetcode's third case: ", ob1.networkDelayTime(times = [[1,2,1]
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

Python

PS C:\Users\melan\Desktop\Algorithms\week3 assy> cd "c:/Users/melan/Desktop/Algorithms/week3 assy"

PS C:\Users\melan\Desktop\Algorithms\week3 assy> & C:/Users/melan/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/melan/Desktop/Algorithms/week3 assy/dijkstra.py"

Leetcode's first case: 2

Leetcode's second case: 1

Leetcode's third case: -1

PS C:\Users\melan\Desktop\Algorithms\week3 assy>

Activate Windows

Go to Settings to activate Windows.

0 0 Ln 49, Col 24 Spaces: 4 UTF-8 CRLF Python 3.10.10 64-bit (microsoft store)

54°F 1:32 PM 3/8/2023

CODE

```
import heapq
import collections

class Solution(object):
    def networkDelayTime(self, times, n, k):
        """
        :type times: List[List[int]]
        :type n: int
        :type k: int
        :rtype: int
        """
        graph = collections.defaultdict(list)
        for u,v,w in times:
            if u == v: return -1
            graph[u].append((v,w))
        if k not in graph: return -1

        pq = [(0,k)]
        dict = {}
        while pq:
            step, source = heapq.heappop(pq)
            if source in dict: continue
            dict[source]=step
            for target, distance in graph[source]:
                if target not in dict:
                    heapq.heappush(pq, (step+distance, target))
        return max(dict.values()) if len(dict) == n else -1

# case 1
# Input: times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2
#Output: 2

ob1 = Solution()
print("Leetcode's first case: ", ob1.networkDelayTime(times =
    [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2))

# case 2
#Input: times = [[1,2,1]], n = 2, k = 1
#Output: 1
```

```
ob1 = Solution()  
print("Leetcode's second case: ", ob1.networkDelayTime(times = [[1,2,1]], n = 2, k = 1))
```

```
# case 3
```

```
#Input: times = [[1,2,1]], n = 2, k = 2
```

```
#Output: -1
```

```
ob1 = Solution()
```

```
print("Leetcode's third case: ", ob1.networkDelayTime(times = [[1,2,1]], n = 2, k = 2))
```