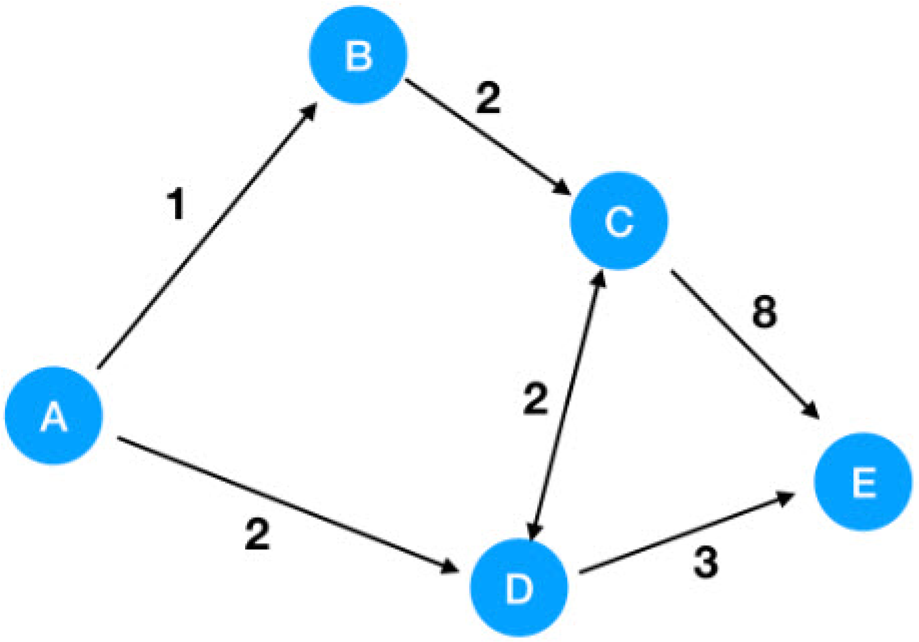
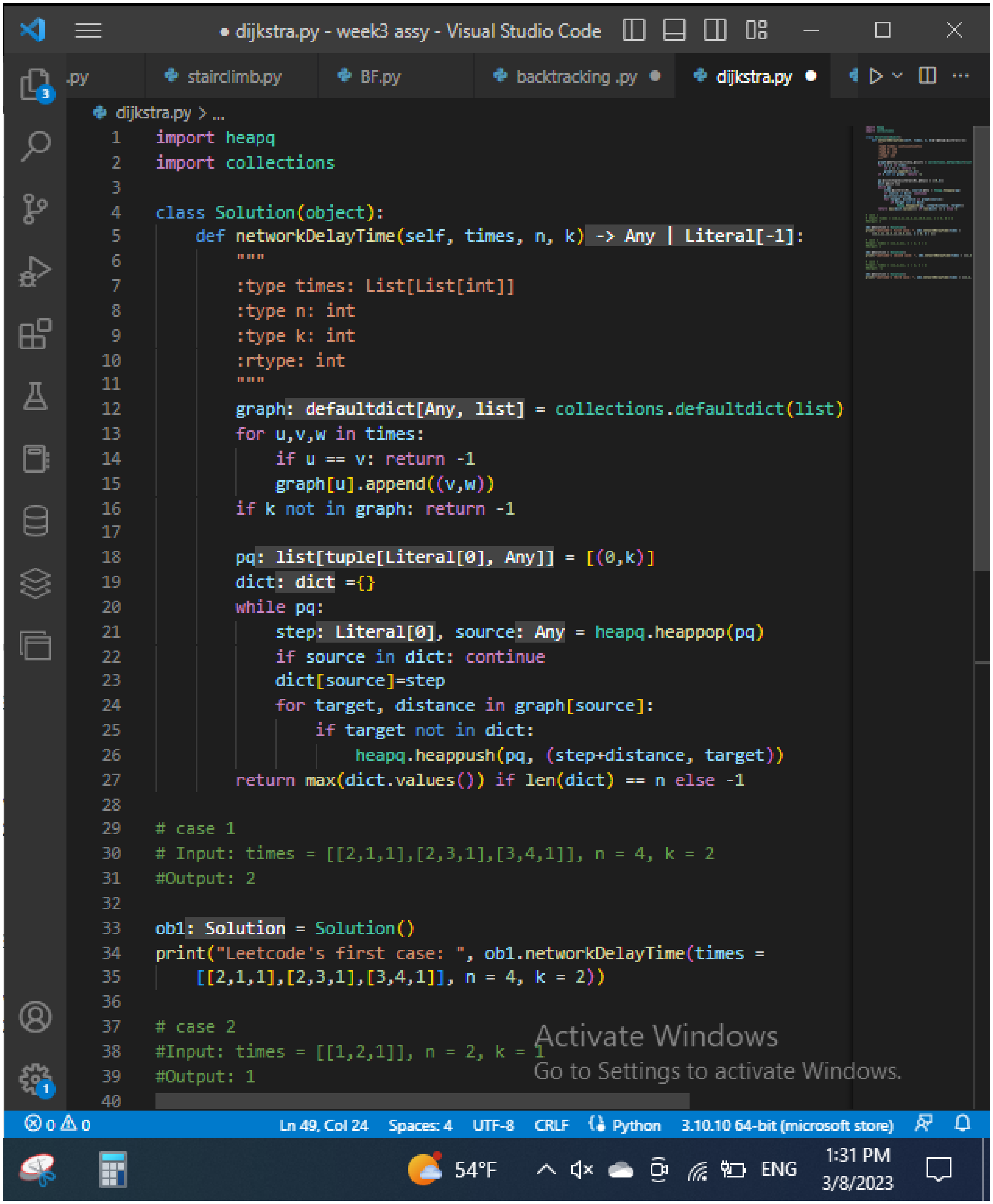
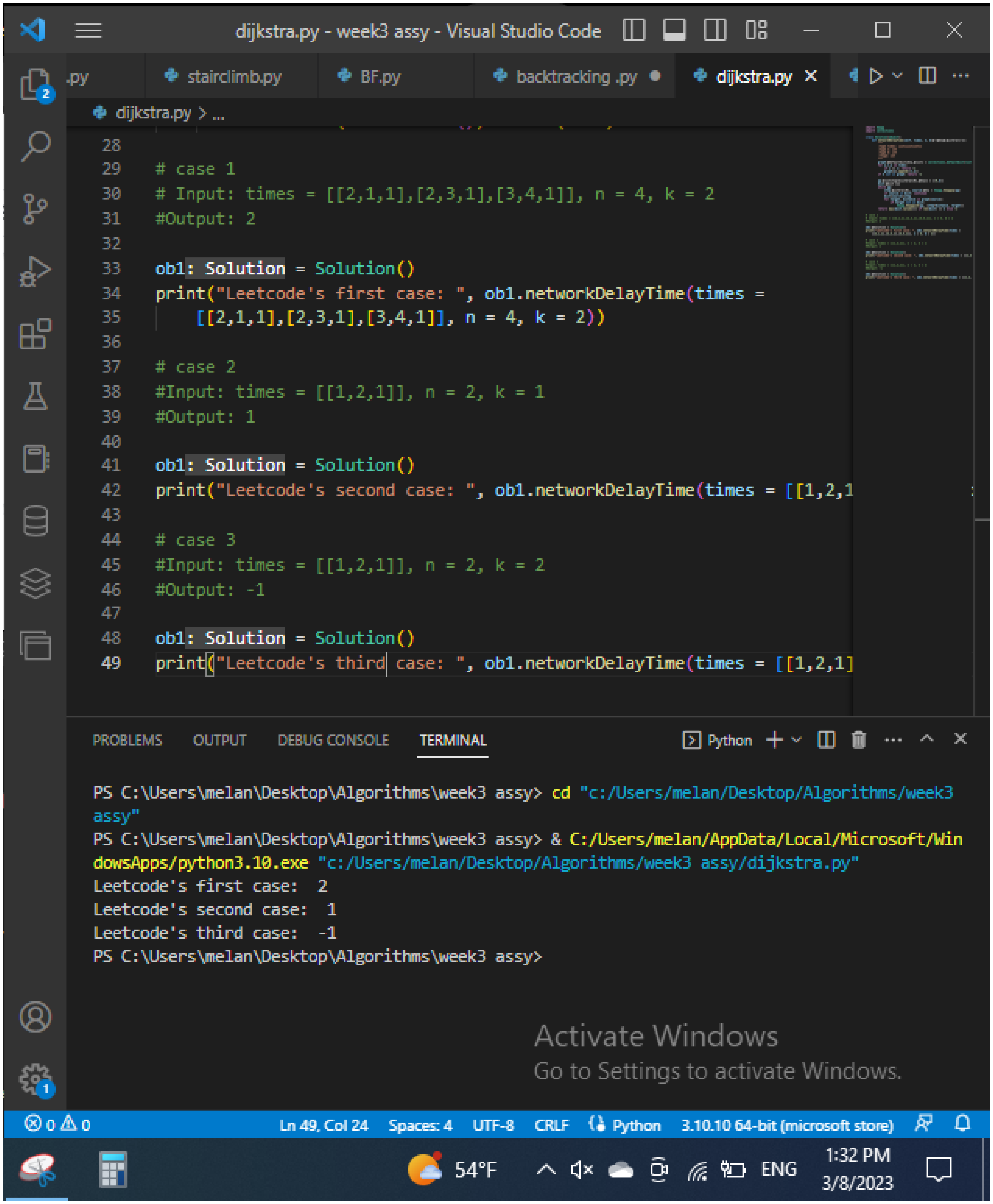


Vertex (accumulated path)	Initial ( ) Next Step V_A	Step1 V_A (V_A) Next Step V_B	Step2 V_B (V_A, V_B) Next Step V_C	Step3 V_C (V_A, V_B, V_C) Next Step V_D	Step4 V_D (V_A, V_B, V_C, V_D) Next Step V_E	Step5 V_E (V_A, V_B, V_C, V_D, V_E) End at V_9
V_A	0	0	~~0~~	~~0~~	~~0~~	~~0~~
V_B	∞	1	1	~~1~~	~~1~~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



```python
import heapq
import collections

class Solution(object):
    def networkDelayTime(self, times, n, k) -> Any | Literal[-1]:
        """
        :type times: List[List[int]]
        :type n: int
        :type k: int
        :rtype: int
        """
        graph: defaultdict[Any, list] = 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: list[tuple[Literal[0], Any]] = [(0,k)]
        dict: dict ={}
        while pq:
            step: Literal[0], source: Any = 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 = 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
```



```python

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

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

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

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

```
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/Win
dowsApps/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>
```

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))
```