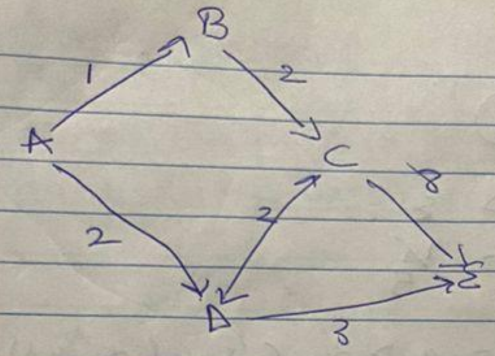
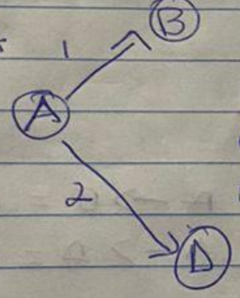
Bellman – Ford's



5 vertices = 4 iterations

Cycle 1

* A

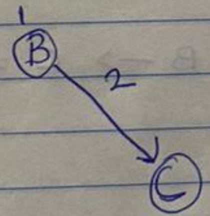


	1		2	
0	~~∞~~	∞	~~∞~~	∞
A	B	C	D	E

* Since 0+1 = 1 < ∞, B's value = 1

" 0 + 2 = 2 < ∞, D's value = 2

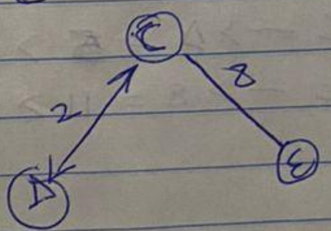
* B



		3		
0	1	~~∞~~	2	∞
A	B	C	D	E

Since 1+2 = 3 < ∞, C is value will change to 3

* C

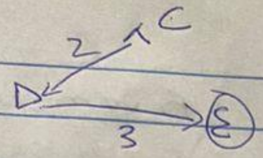


				11
0	1	3	2	~~∞~~
A	B	C	D	E

- Since 3 + 2 = 5 > 2, D's value will not ch

- " 3 + 8 = 11 < ∞, E's value will change

* D



A	B	C	D	E
0	1	3	2	~~11~~ 5

- Since 2 + 2 = 4 > 3, C's value will not change
- Since 2 + 3 = 5<11 E's value will chang = 5

cycle 2:

* A:

A	B	C	D	E
0	1	3	2	5

A -> B = 1
A -> D = 2

Nothing changes

* B:

A	B	C	D	E
0	1	3	2	5

B -> C = 3

Nothing changes

* C:

A	B	C	D	E
0	1	3	2	5

C -> D = 5 > 2
C -> E = 11 > 5

Nothing changes

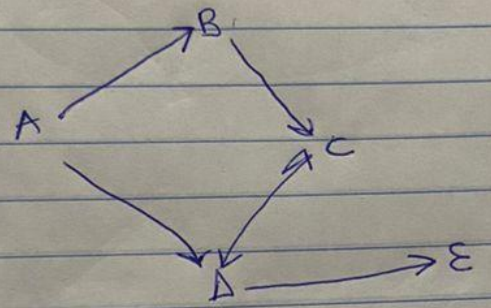
* D:

A	B	C	D	E
0	1	3	2	5

D -> C = 4 > 3
D - E = 5

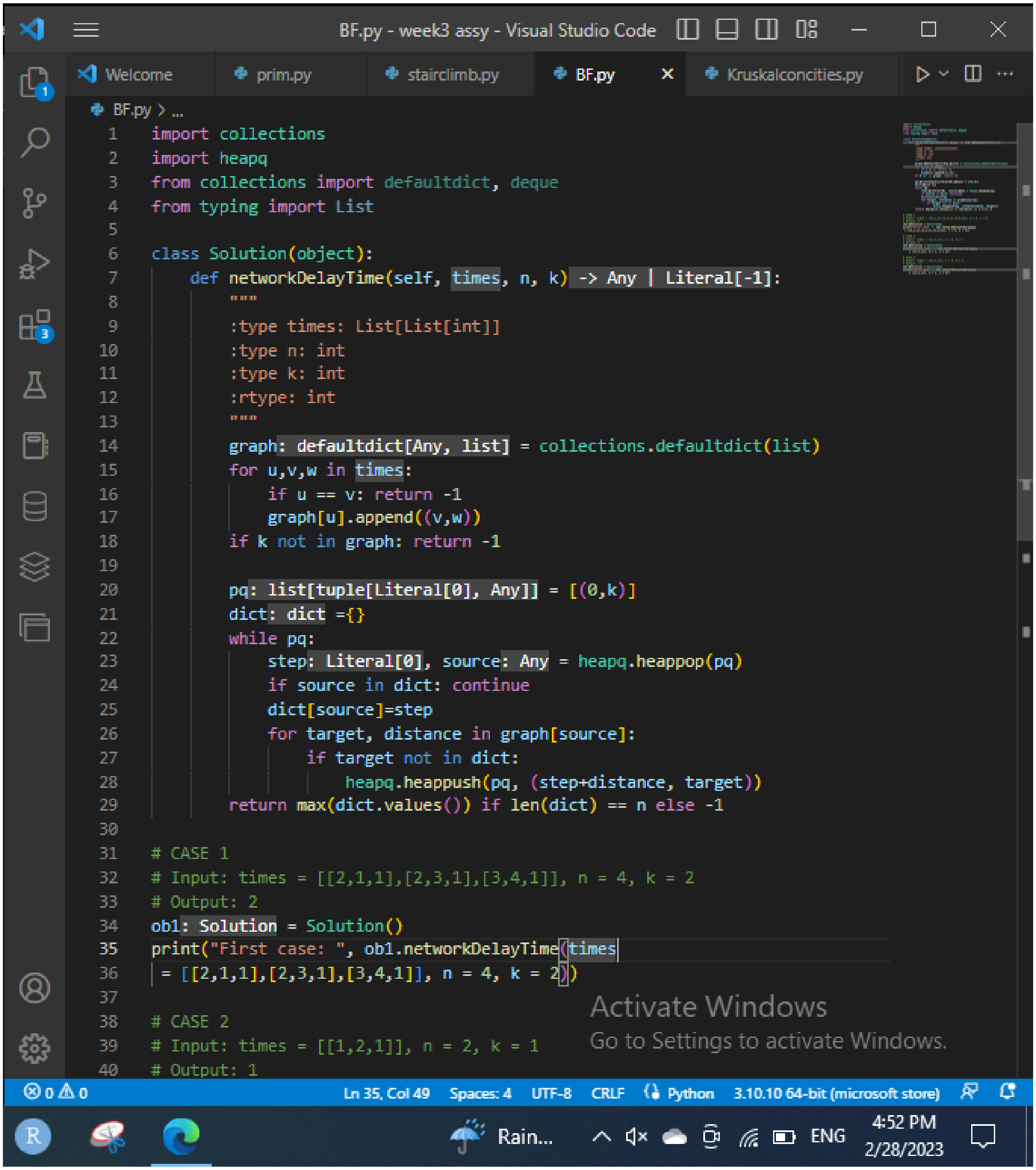
Nothing changes

[illegible]: The process ends at cycle 2 because none of the vertices is change.



c -> is not included to avoid loop.

STEP 2



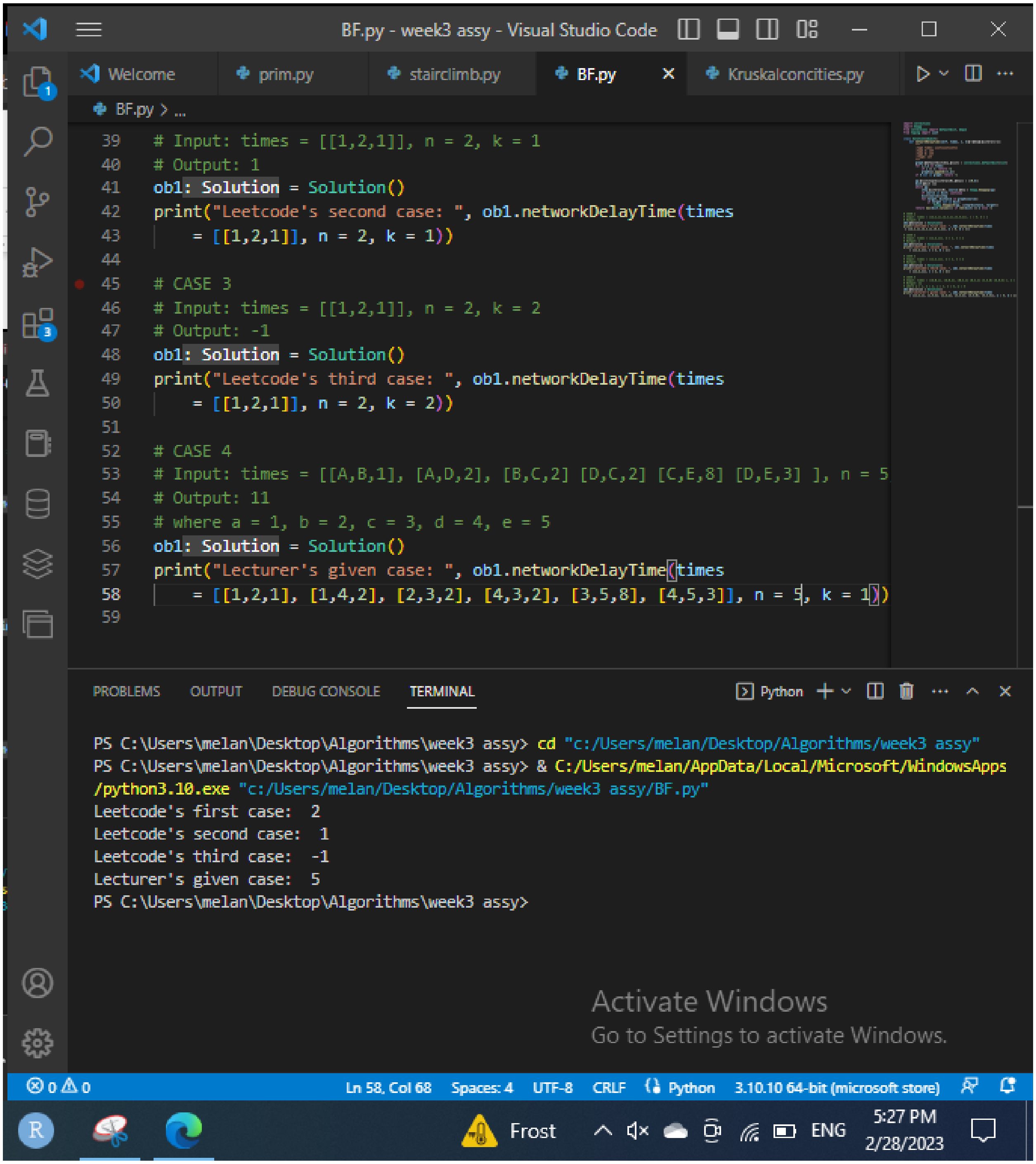
```python
import collections
import heapq
from collections import defaultdict, deque
from typing import List

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("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
# 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]], n = 2, k = 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]], n = 2, k = 2))

# CASE 4
# Input: times = [[A,B,1], [A,D,2], [B,C,2] [D,C,2] [C,E,8] [D,E,3] ], n = 5
# Output: 11
# where a = 1, b = 2, c = 3, d = 4, e = 5
ob1: Solution = Solution()
print("Lecturer's given case: ", ob1.networkDelayTime(times
    = [[1,2,1], [1,4,2], [2,3,2], [4,3,2], [3,5,8], [4,5,3]], n = 5, k = 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/WindowsApps
/python3.10.exe "c:/Users/melan/Desktop/Algorithms/week3 assy/BF.py"
Leetcode's first case:  2
Leetcode's second case:  1
Leetcode's third case:  -1
Lecturer's given case:  5
PS C:\Users\melan\Desktop\Algorithms\week3 assy>
```

CODE

```
import collections
import heapq
from collections import defaultdict, deque
from typing import List

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("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("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("Third case: ", ob1.networkDelayTime(times
    = [[1,2,1]], n = 2, k = 2))
# CASE 4
# Input: times = [[A,B,1], [A,D,2], [B,C,2] [D,C,2] [C,E,8] [D,E,3] ], n = 5, k = A
# Output: 11
# where a = 1, b = 2, c = 3, d = 4, e = 5
ob1 = Solution()
print("Lecturer's given case: ", ob1.networkDelayTime(times
    = [[1,2,1], [1,4,2], [2,3,2], [4,3,2], [3,5,8], [4,5,3]], n = 5, k = 1))
```