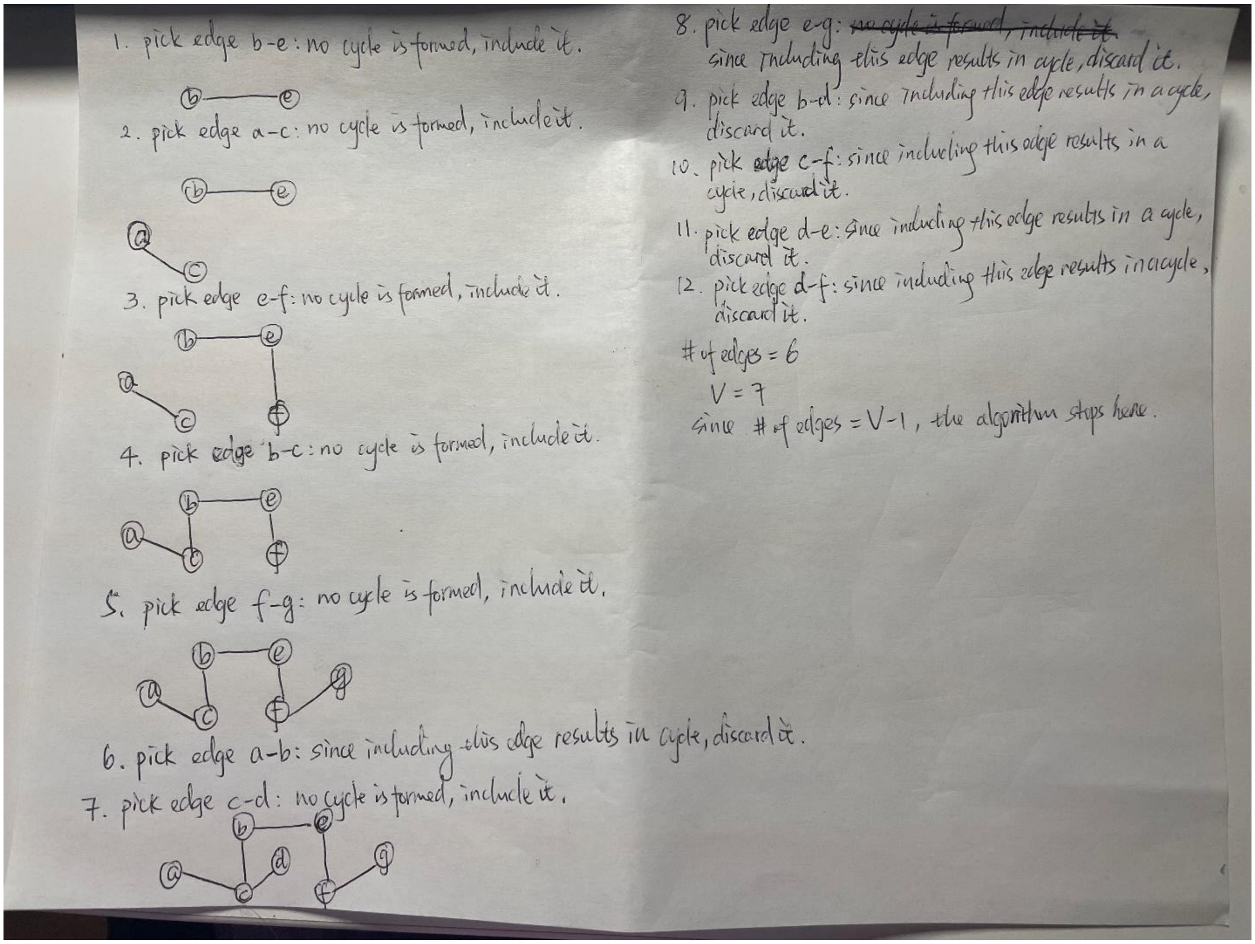
Step 1: Study 1135 Connecting Cities With Minimum Cost (local copy) – Medium

```
class Solution:
    def minimumCost(self, n: int, connections: List[List[int]]) -> int:
        r = [i for i in range(n + 1)]
        w = [1] * (n + 1)
        connections = sorted(connections, key=lambda c: c[2])

        total_branches = 0
        min_weight = 0

        def ref(n):
            while r[n] != n:
                n = r[n]

            return n

        for i, j, c in connections:
            ref_i, ref_j = ref(i), ref(j)
            if ref_i == ref_j:
                continue

            total_branches += 1
            min_weight += c

            if w[ref_i] < w[ref_j]:
                r[ref_i] = ref_j
                w[ref_j] += w[ref_i]
            else:
                r[ref_j] = ref_i
                w[ref_i] += w[ref_j]


        return min_weight if total_branches == n - 1 else -1
```

Step 2: Use Kruskal approach to find the Minimum Spanning Tree of this question

weight	src	dest
2	b	e
3	a	c
3	e	f
4	b	c
4	f	g
5	a	b
5	c	d
5	e	g
6	b	d

6	c	f
6	d	e
6	d	f



1. pick edge b-e: no cycle is formed, include it.
2. pick edge a-c: no cycle is formed, include it.
3. pick edge e-f: no cycle is formed, include it.
4. pick edge b-c: no cycle is formed, include it.
5. pick edge f-g: no cycle is formed, include it.
6. pick edge a-b: since including this edge results in cycle, discard it.
7. pick edge c-d: no cycle is formed, include it.
8. pick edge e-g: ~~no cycle is formed, include it.~~ since including this edge results in cycle, discard it.
9. pick edge b-d: since including this edge results in a cycle, discard it.
10. pick edge c-f: since including this edge results in a cycle, discard it.
11. pick edge d-e: since including this edge results in a cycle, discard it.
12. pick edge d-f: since including this edge results in a cycle, discard it.

\# of edges = 6
V = 7
since # of edges = V-1, the algorithm stops here.

Step 3: Ask ChatGPT: "How to use Kruskal approach to implement a Python code to solve the LeetCode question: '1135 Connecting Cities With Minimum Cost', with the test data: 'Input: n = 3, connections = [[1, 2, 5],[1,3,6],[2,3,1]], Output: 6'"

To solve the LeetCode question "Connecting Cities With Minimum Cost" using Kruskal's algorithm in Python, you can follow these steps:

1. Create a class, `DisjointSet`, to represent a disjoint set data structure. This class will have two methods: `find` and `union`. The `find` method finds the root of a given element, and the `union` method merges two sets.
2. Create a helper function, `minimum_cost`, to calculate the minimum cost using Kruskal's algorithm. This function takes the total number of cities `n` and the `connections` list as input.
3. Sort the `connections` list in ascending order based on the cost.
4. Create an instance of the `DisjointSet` class.
5. Initialize a variable, `minimum_cost`, to keep track of the total minimum cost.
6. Iterate over each connection in the sorted `connections` list. For each connection `[u, v, cost]`, check if the root of `u` and `v` in the disjoint set is the same. If they are not, it means they are in different sets and can be connected. Add the cost of the connection to the `minimum_cost` and merge the sets using the `union` method.
7. After iterating through all connections, check if all cities are connected by verifying if there is only one set remaining in the disjoint set data structure. If not, return -1 to indicate that it is not possible to connect all cities.
8. Finally, return the `minimum_cost` variable, which represents the minimum cost to connect all cities.

Here's the Python code implementing Kruskal's algorithm for the given test data:

python

Copy code

```python
class DisjointSet:
    def __init__(self, n):
        self.parent = list(range(n + 1))

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        self.parent[self.find(x)] = self.find(y)
```

```python
def minimum_cost(n, connections):
    connections.sort(key=lambda x: x[2])
    disjoint_set = DisjointSet(n)
    minimum_cost = 0

    for u, v, cost in connections:
        root_u = disjoint_set.find(u)
        root_v = disjoint_set.find(v)

        if root_u != root_v:
            disjoint_set.union(u, v)
            minimum_cost += cost

    root = disjoint_set.find(1)
    for i in range(2, n + 1):
        if disjoint_set.find(i) != root:
            return -1

    return minimum_cost
```

```python
# Test data
n = 3
connections = [[1, 2, 5], [1, 3, 6], [2, 3, 1]]
output = minimum_cost(n, connections)
print(output)  # Output: 6
```

Step 4: Implement the Python code created by ChatGPT

q1.py > minimum_cost

```python
class DisjointSet:
    def __init__(self, n):
        self.parent = list(range(n + 1))

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        self.parent[self.find(x)] = self.find(y)

```

```python
def minimum_cost(n, connections):
    connections.sort(key=lambda x: x[2])
    disjoint_set = DisjointSet(n)
    minimum_cost = 0

    for u, v, cost in connections:
        root_u = disjoint_set.find(u)
        root_v = disjoint_set.find(v)

        if root_u != root_v:
            disjoint_set.union(u, v)
            minimum_cost += cost

    root = disjoint_set.find(1)
    for i in range(2, n + 1):
        if disjoint_set.find(i) != root:
            return -1

    return minimum_cost

```

```python
# Test data
n = 3
connections = [[1, 2, 5], [1, 3, 6], [2, 3, 1]]
output = minimum_cost(n, connections)
print(output)  # Output: 6

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\MSCS\CS455 Algorithm\w5> & C:/Users/odody/AppData/Local/Programs/Python/Python311/python.exe "d:/MSCS/CS455 Algorithm/w5/q1.py"
6
PS D:\MSCS\CS455 Algorithm\w5>
```

ve Share Column Selection Ln 17, Col 21 Spaces: 4 UTF-8 CRLF {} Python 3.11.3 64-bit Go Live

9:19 PM 6/21/2023

Step 5: Test the Python code with all the test cases provided by 1135 Connecting Cities With Minimum Cost (local copy) q1.py > minimum_cost

```python
class DisjointSet:
    def __init__(self, n):
        self.parent = list(range(n + 1))

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        self.parent[self.find(x)] = self.find(y)

```

```python
def minimum_cost(n, connections):
    connections.sort(key=lambda x: x[2])
    disjoint_set = DisjointSet(n)
    minimum_cost = 0

    for u, v, cost in connections:
        root_u = disjoint_set.find(u)
        root_v = disjoint_set.find(v)

        if root_u != root_v:
            disjoint_set.union(u, v)
            minimum_cost += cost

    root = disjoint_set.find(1)
    for i in range(2, n + 1):
        if disjoint_set.find(i) != root:
            return -1

    return minimum_cost
```

```python
# Test data
n = 3
connections = [[1, 2, 5], [1, 3, 6], [2, 3, 1]]
output = minimum_cost(n, connections)
print(output)  # Output: 6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Python

```
PS D:\MSCS\CS455 Algorithm\w5> & C:/Users/odody/AppData/Local/Programs/Python/Python311/python.exe "d:/MSCS/CS455 Algorithm/w5/q1.py"
6
PS D:\MSCS\CS455 Algorithm\w5>
```

ve Share Column Selection Ln 17, Col 21 Spaces: 4 UTF-8 CRLF {} Python 3.11.3 64-bit Go Live

16° ENG 9:19 PM 6/21/2023

q1.py > minimum_cost

```python
class DisjointSet:
    def __init__(self, n):
        self.parent = list(range(n + 1))

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        self.parent[self.find(x)] = self.find(y)

```

```python
def minimum_cost(n, connections):
    connections.sort(key=lambda x: x[2])
    disjoint_set = DisjointSet(n)
    minimum_cost = 0

    for u, v, cost in connections:
        root_u = disjoint_set.find(u)
        root_v = disjoint_set.find(v)

        if root_u != root_v:
            disjoint_set.union(u, v)
            minimum_cost += cost

    root = disjoint_set.find(1)
    for i in range(2, n + 1):
        if disjoint_set.find(i) != root:
            return -1

    return minimum_cost

# Test data
n = 4
connections = [[1,2,3],[3,4,4]]
output = minimum_cost(n, connections)
print(output)  # Output: -1
```

```
PS D:\MSCS\CS455 Algorithm\w5> & C:/Users/odody/AppData/Local/Programs/Python/Python311/python.exe "d:/MSCS/CS455 Algorithm/w5/q1.py"
-1
PS D:\MSCS\CS455 Algorithm\w5>
```