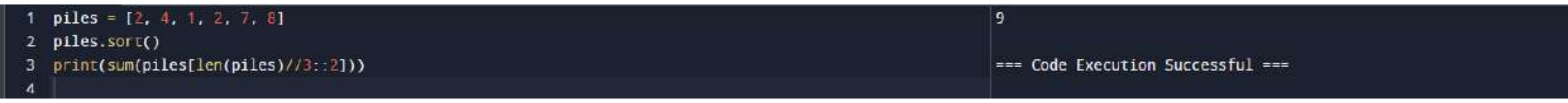
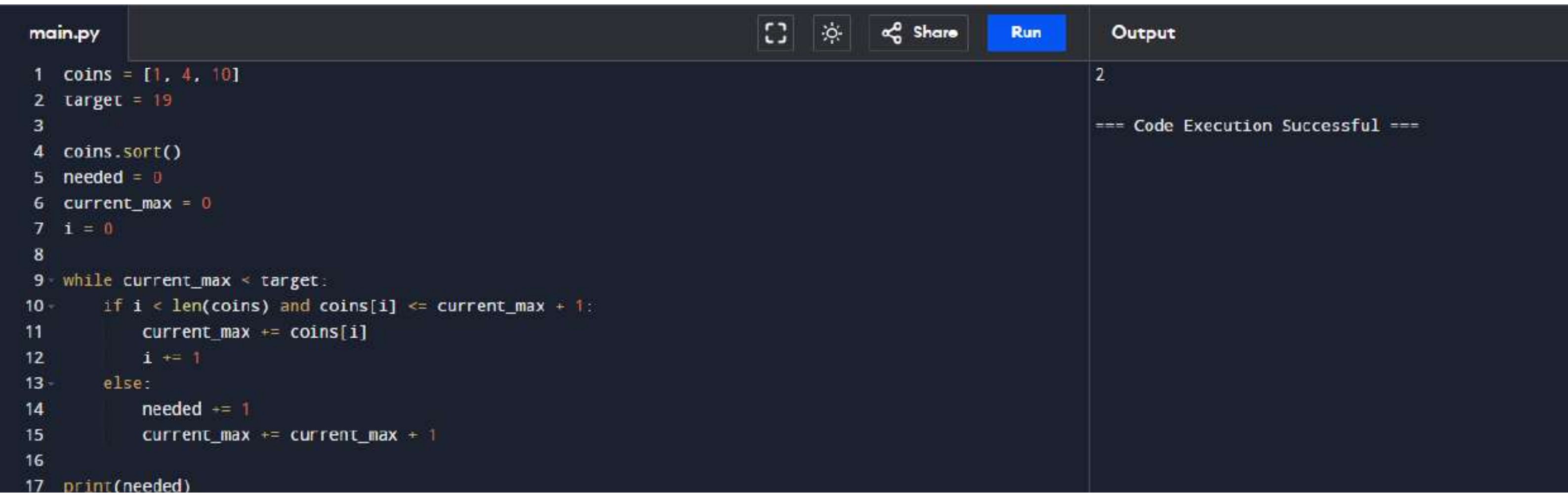


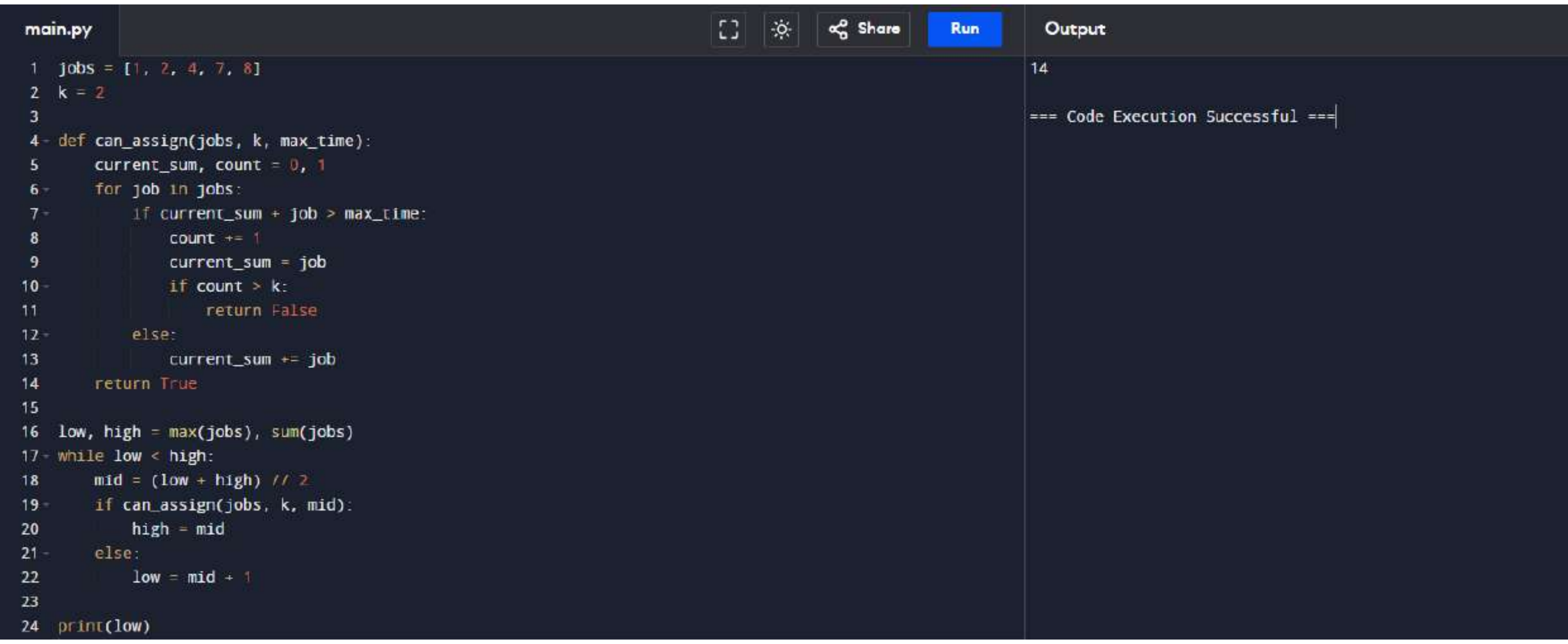
```python
piles = [2, 4, 1, 2, 7, 8]
piles.sort()
print(sum(piles[len(piles)//3::2]))
```

```
9

=== Code Execution Successful ===
```



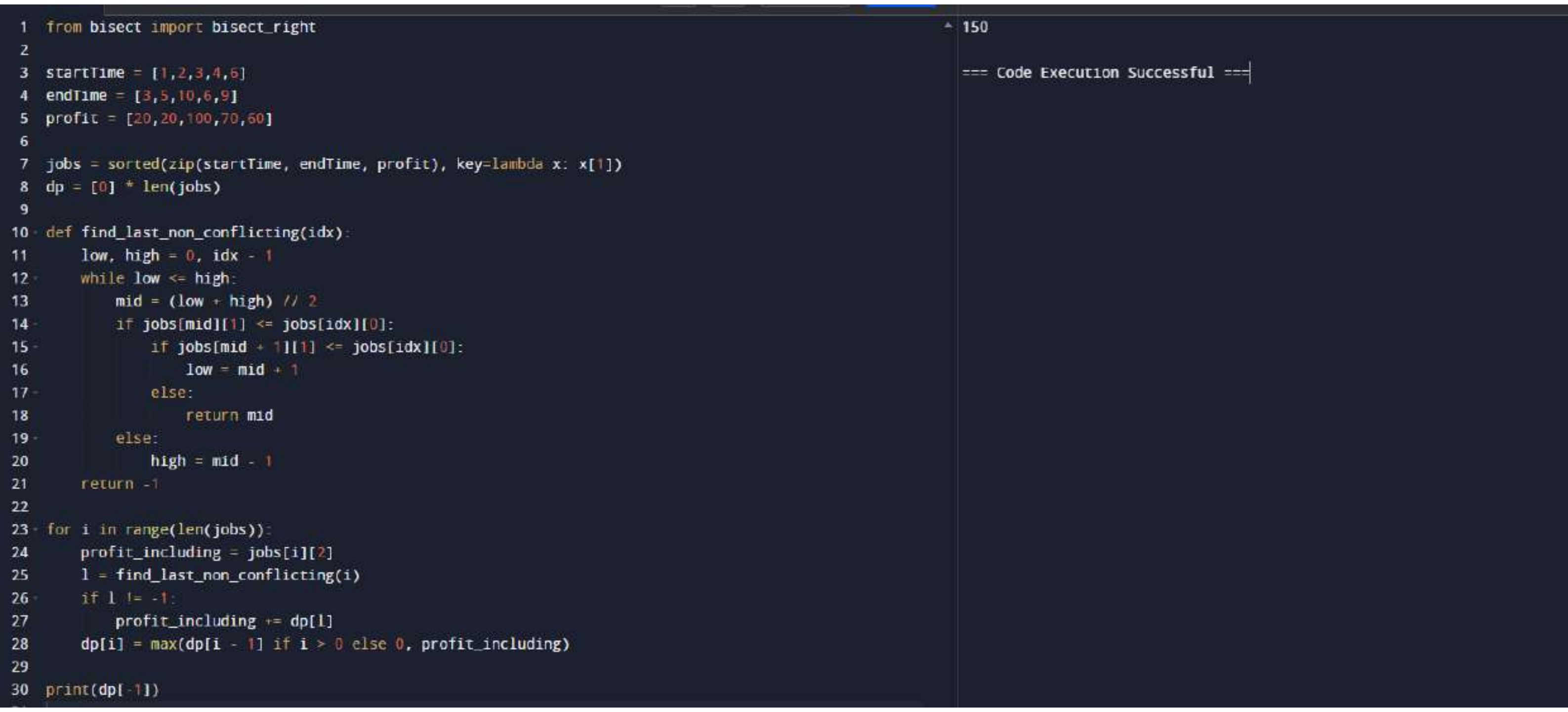
```python
coins = [1, 4, 10]
target = 19

coins.sort()
needed = 0
current_max = 0
i = 0

while current_max < target:
    if i < len(coins) and coins[i] <= current_max + 1:
        current_max += coins[i]
        i += 1
    else:
        needed += 1
        current_max += current_max + 1

print(needed)
```

```
2

=== Code Execution Successful ===
```



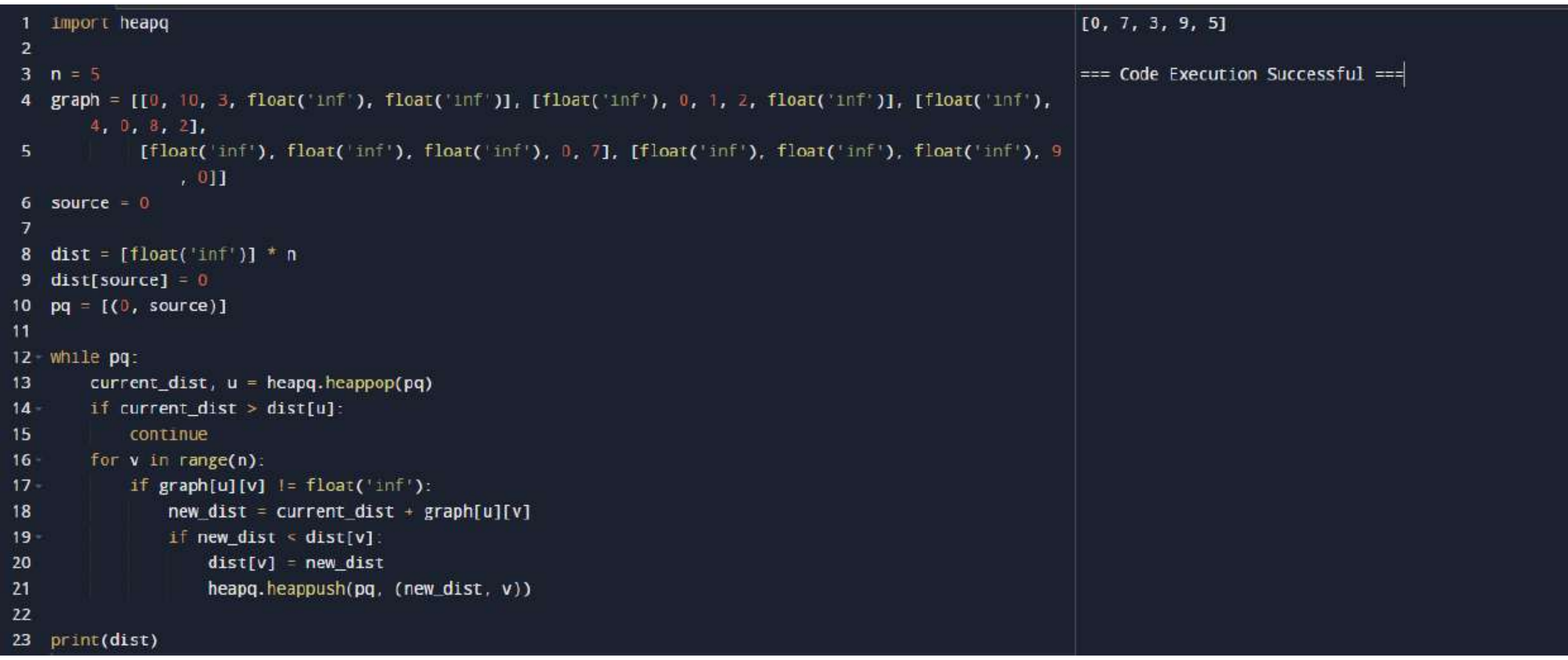
```python
jobs = [1, 2, 4, 7, 8]
k = 2

def can_assign(jobs, k, max_time):
    current_sum, count = 0, 1
    for job in jobs:
        if current_sum + job > max_time:
            count += 1
            current_sum = job
            if count > k:
                return False
        else:
            current_sum += job
    return True

low, high = max(jobs), sum(jobs)
while low < high:
    mid = (low + high) // 2
    if can_assign(jobs, k, mid):
        high = mid
    else:
        low = mid + 1

print(low)
```

```
14

=== Code Execution Successful ===
```



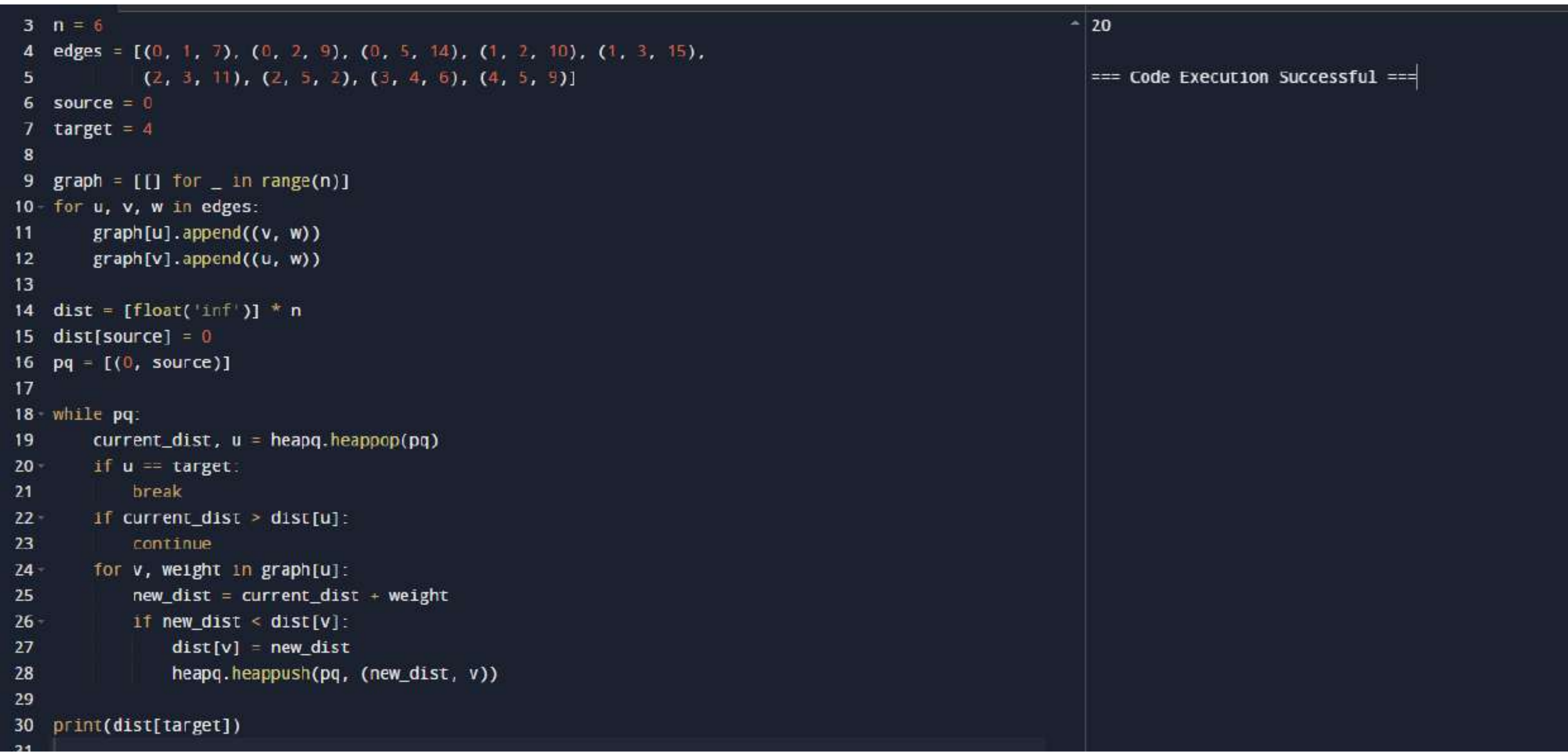
```python
from bisect import bisect_right

startTime = [1,2,3,4,6]
endTime = [3,5,10,6,9]
profit = [20,20,100,70,60]

jobs = sorted(zip(startTime, endTime, profit), key=lambda x: x[1])
dp = [0] * len(jobs)

def find_last_non_conflicting(idx):
    low, high = 0, idx - 1
    while low <= high:
        mid = (low + high) // 2
        if jobs[mid][1] <= jobs[idx][0]:
            if jobs[mid + 1][1] <= jobs[idx][0]:
                low = mid + 1
            else:
                return mid
        else:
            high = mid - 1
    return -1

for i in range(len(jobs)):
    profit_including = jobs[i][2]
    l = find_last_non_conflicting(i)
    if l != -1:
        profit_including += dp[l]
    dp[i] = max(dp[i - 1] if i > 0 else 0, profit_including)

print(dp[-1])
```

```
150

=== Code Execution Successful ===
```



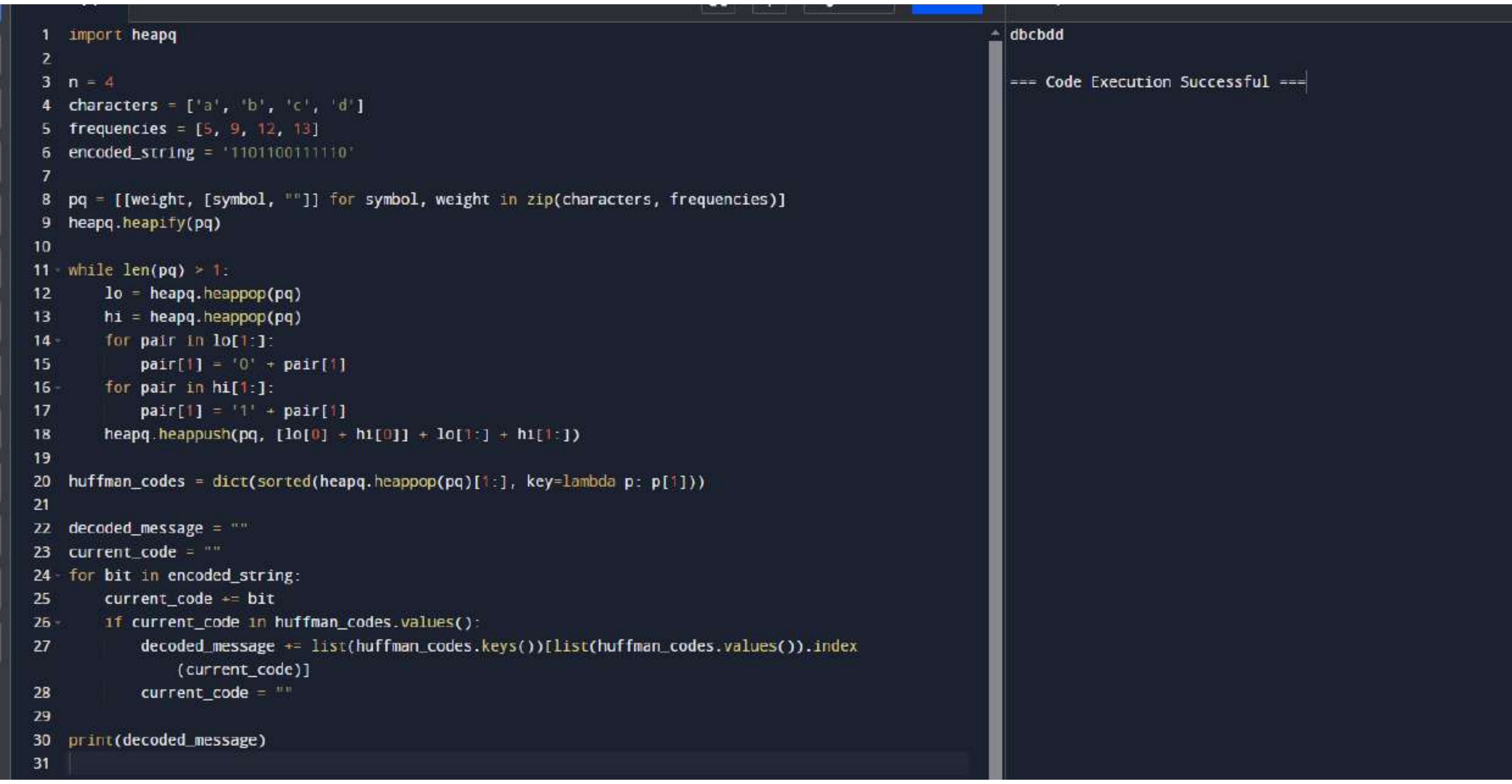
```python
import heapq

n = 5
graph = [[0, 10, 3, float('inf'), float('inf')], [float('inf'), 0, 1, 2, float('inf')], [float('inf'), 4, 0, 8, 2],
         [float('inf'), float('inf'), float('inf'), 0, 7], [float('inf'), float('inf'), float('inf'), 9, 0]]
source = 0

dist = [float('inf')] * n
dist[source] = 0
pq = [(0, source)]

while pq:
    current_dist, u = heapq.heappop(pq)
    if current_dist > dist[u]:
        continue
    for v in range(n):
        if graph[u][v] != float('inf'):
            new_dist = current_dist + graph[u][v]
            if new_dist < dist[v]:
                dist[v] = new_dist
                heapq.heappush(pq, (new_dist, v))

print(dist)
```

```
[0, 7, 3, 9, 5]

=== Code Execution Successful ===
```



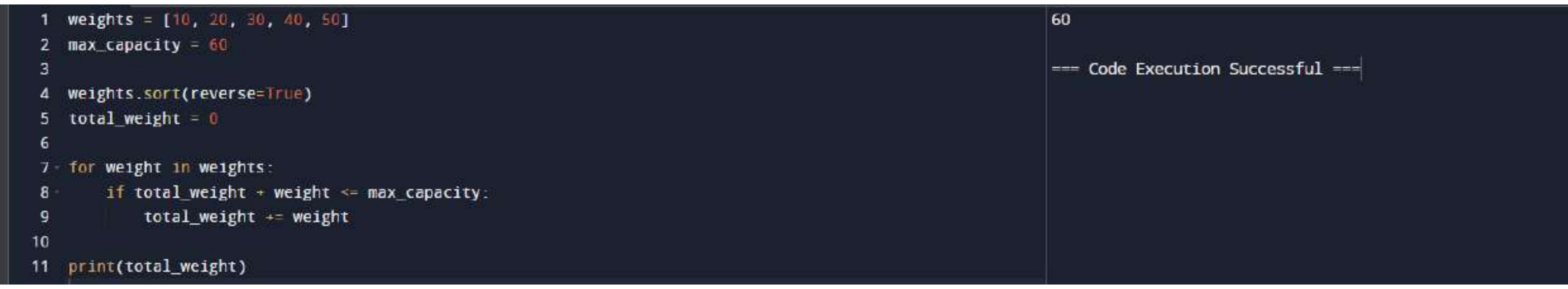
```
n = 6
edges = [(0, 1, 7), (0, 2, 9), (0, 5, 14), (1, 2, 10), (1, 3, 15),
         (2, 3, 11), (2, 5, 2), (3, 4, 6), (4, 5, 9)]
source = 0
target = 4

graph = [[] for _ in range(n)]
for u, v, w in edges:
    graph[u].append((v, w))
    graph[v].append((u, w))

dist = [float('inf')] * n
dist[source] = 0
pq = [(0, source)]

while pq:
    current_dist, u = heapq.heappop(pq)
    if u == target:
        break
    if current_dist > dist[u]:
        continue
    for v, weight in graph[u]:
        new_dist = current_dist + weight
        if new_dist < dist[v]:
            dist[v] = new_dist
            heapq.heappush(pq, (new_dist, v))

print(dist[target])
```

```
20

=== Code Execution Successful ===
```



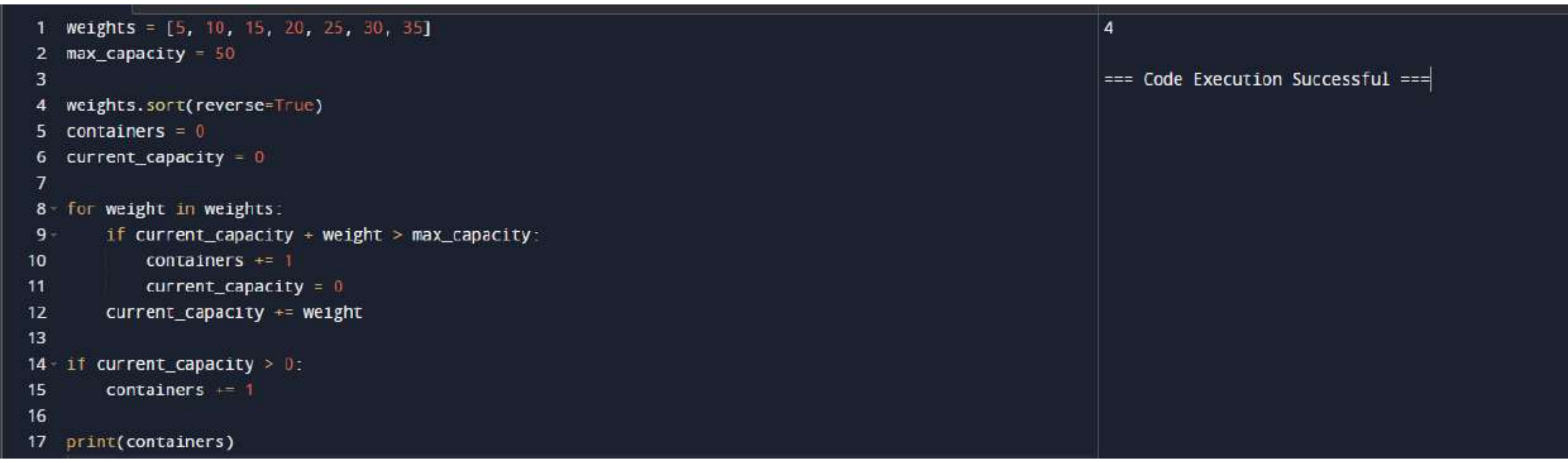
```python
import heapq

n = 4
characters = ['a', 'b', 'c', 'd']
frequencies = [5, 9, 12, 13]
encoded_string = '1101100111110'

pq = [[weight, [symbol, ""]] for symbol, weight in zip(characters, frequencies)]
heapq.heapify(pq)

while len(pq) > 1:
    lo = heapq.heappop(pq)
    hi = heapq.heappop(pq)
    for pair in lo[1:]:
        pair[1] = '0' + pair[1]
    for pair in hi[1:]:
        pair[1] = '1' + pair[1]
    heapq.heappush(pq, [lo[0] + hi[0]] + lo[1:] + hi[1:])

huffman_codes = dict(sorted(heapq.heappop(pq)[1:], key=lambda p: p[1]))

decoded_message = ""
current_code = ""
for bit in encoded_string:
    current_code += bit
    if current_code in huffman_codes.values():
        decoded_message += list(huffman_codes.keys())[list(huffman_codes.values()).index
            (current_code)]
        current_code = ""

print(decoded_message)

```

```
dbcbdd

=== Code Execution Successful ===
```



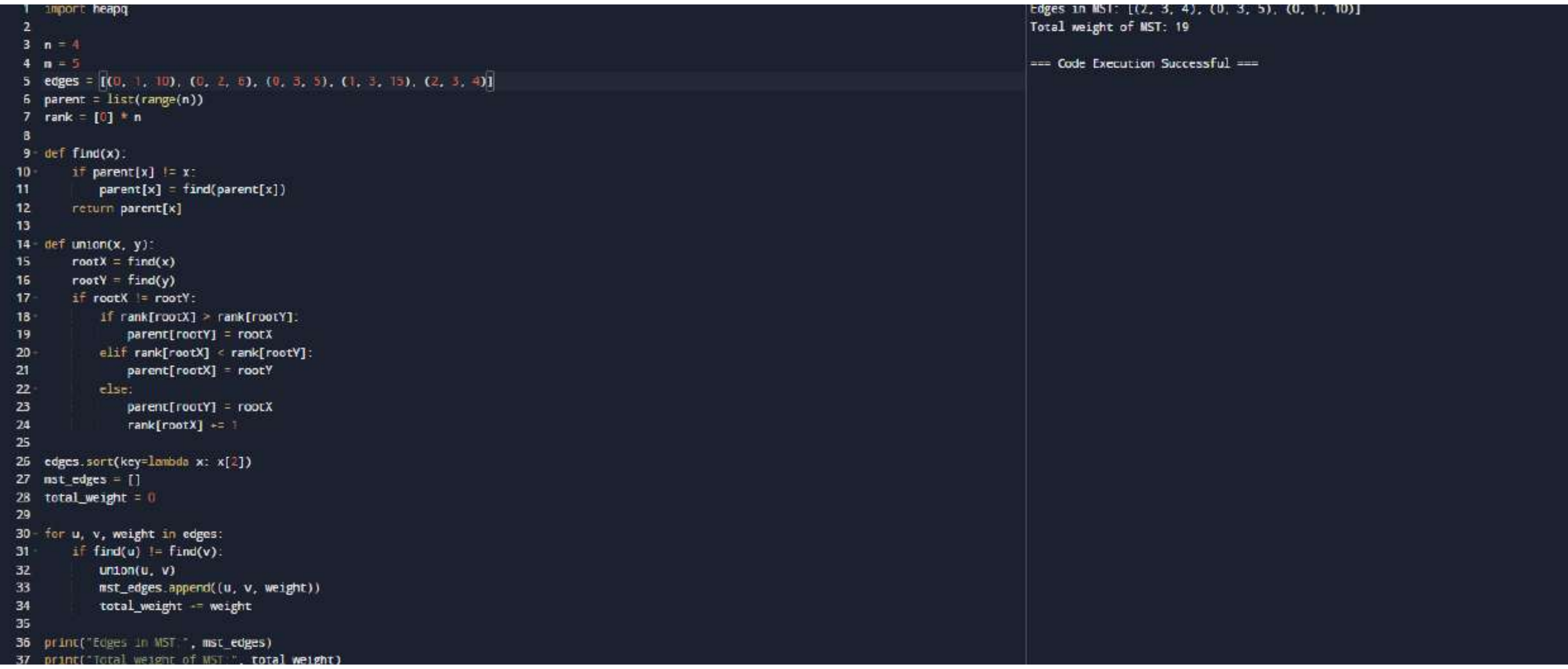
```python
weights = [10, 20, 30, 40, 50]
max_capacity = 60

weights.sort(reverse=True)
total_weight = 0

for weight in weights:
    if total_weight + weight <= max_capacity:
        total_weight += weight

print(total_weight)
```

```
60

=== Code Execution Successful ===
```



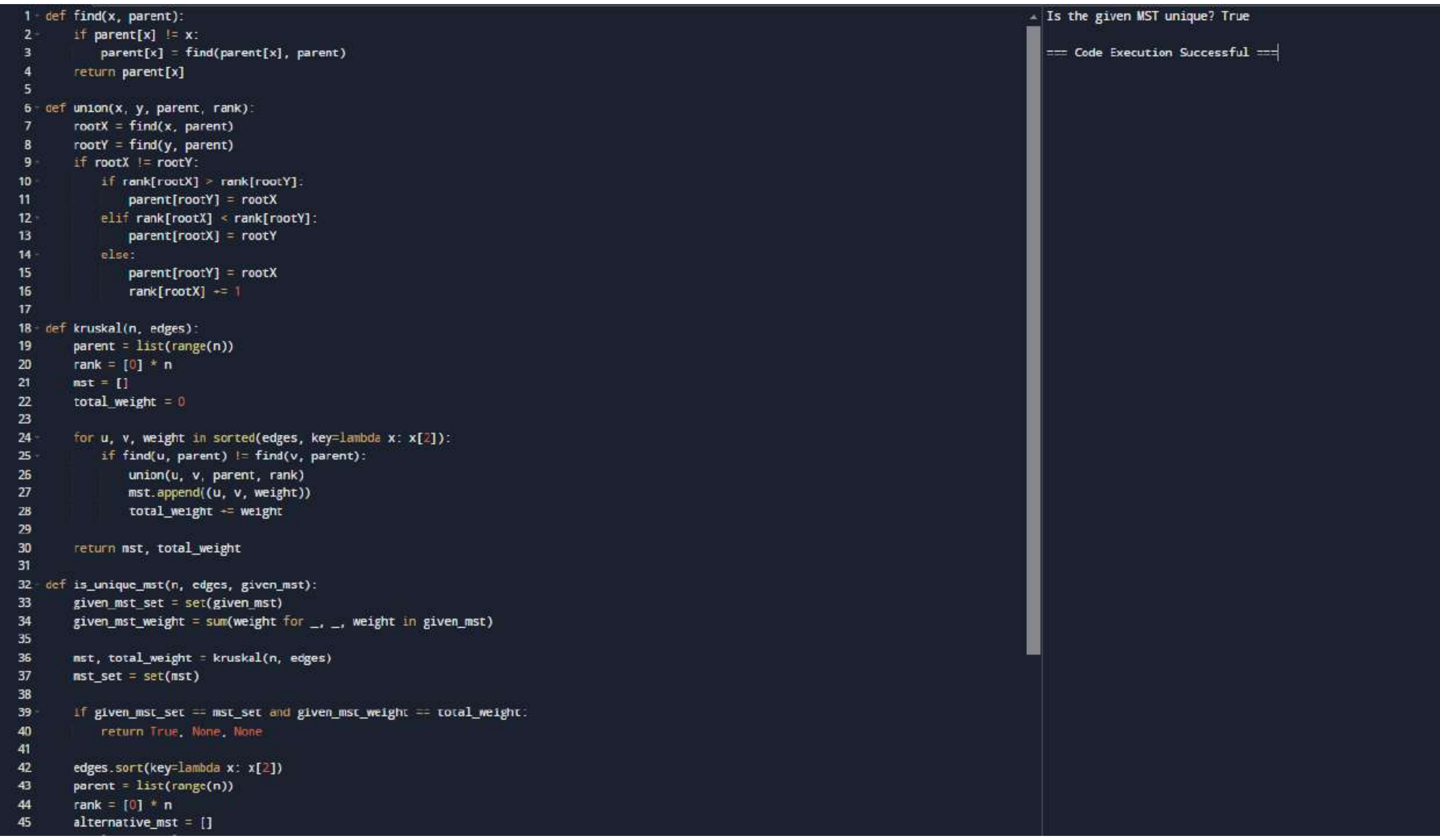
```python
weights = [5, 10, 15, 20, 25, 30, 35]
max_capacity = 50

weights.sort(reverse=True)
containers = 0
current_capacity = 0

for weight in weights:
    if current_capacity + weight > max_capacity:
        containers += 1
        current_capacity = 0
    current_capacity += weight

if current_capacity > 0:
    containers += 1

print(containers)
```

```
4

=== Code Execution Successful ===
```



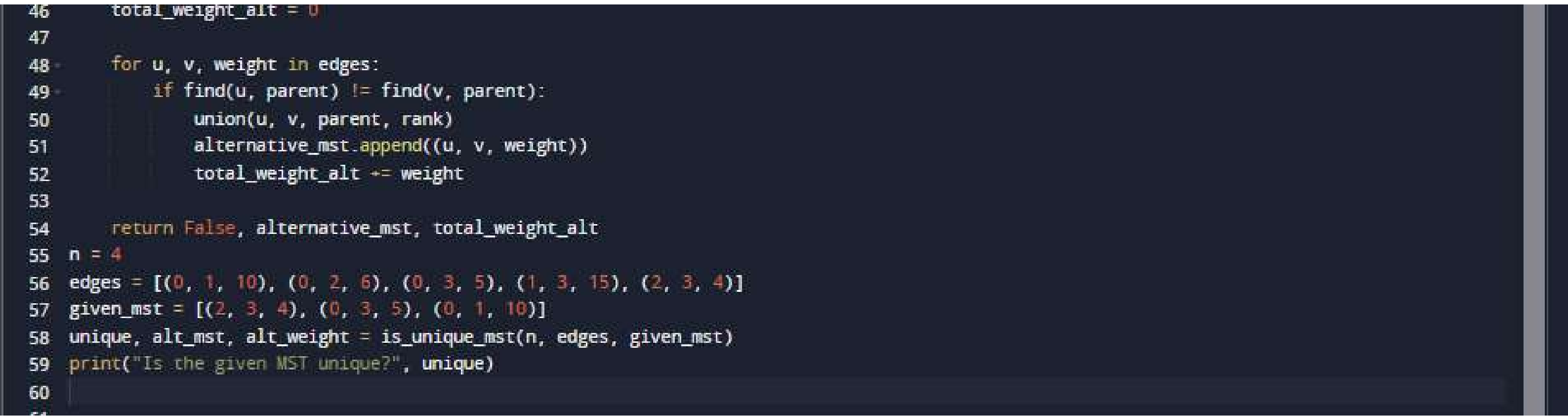
```python
import heapq

n = 4
m = 5
edges = [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
parent = list(range(n))
rank = [0] * n

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

def union(x, y):
    rootX = find(x)
    rootY = find(y)
    if rootX != rootY:
        if rank[rootX] > rank[rootY]:
            parent[rootY] = rootX
        elif rank[rootX] < rank[rootY]:
            parent[rootX] = rootY
        else:
            parent[rootY] = rootX
            rank[rootX] += 1

edges.sort(key=lambda x: x[2])
mst_edges = []
total_weight = 0

for u, v, weight in edges:
    if find(u) != find(v):
        union(u, v)
        mst_edges.append((u, v, weight))
        total_weight += weight

print("Edges in MST:", mst_edges)
print("Total weight of MST:", total_weight)
```

```
Edges in MST: [(2, 3, 4), (0, 3, 5), (0, 1, 10)]
Total weight of MST: 19

=== Code Execution Successful ===
```



```python
def find(x, parent):
    if parent[x] != x:
        parent[x] = find(parent[x], parent)
    return parent[x]

def union(x, y, parent, rank):
    rootX = find(x, parent)
    rootY = find(y, parent)
    if rootX != rootY:
        if rank[rootX] > rank[rootY]:
            parent[rootY] = rootX
        elif rank[rootX] < rank[rootY]:
            parent[rootX] = rootY
        else:
            parent[rootY] = rootX
            rank[rootX] += 1

def kruskal(n, edges):
    parent = list(range(n))
    rank = [0] * n
    mst = []
    total_weight = 0

    for u, v, weight in sorted(edges, key=lambda x: x[2]):
        if find(u, parent) != find(v, parent):
            union(u, v, parent, rank)
            mst.append((u, v, weight))
            total_weight += weight

    return mst, total_weight

def is_unique_mst(n, edges, given_mst):
    given_mst_set = set(given_mst)
    given_mst_weight = sum(weight for _, _, weight in given_mst)

    mst, total_weight = kruskal(n, edges)
    mst_set = set(mst)

    if given_mst_set == mst_set and given_mst_weight == total_weight:
        return True, None, None

    edges.sort(key=lambda x: x[2])
    parent = list(range(n))
    rank = [0] * n
    alternative_mst = []
```

```
Is the given MST unique? True

=== Code Execution Successful ===
```



```
    total_weight_alt = 0

    for u, v, weight in edges:
        if find(u, parent) != find(v, parent):
            union(u, v, parent, rank)
            alternative_mst.append((u, v, weight))
            total_weight_alt += weight

    return False, alternative_mst, total_weight_alt
n = 4
edges = [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
given_mst = [(2, 3, 4), (0, 3, 5), (0, 1, 10)]
unique, alt_mst, alt_weight = is_unique_mst(n, edges, given_mst)
print("Is the given MST unique?", unique)

```