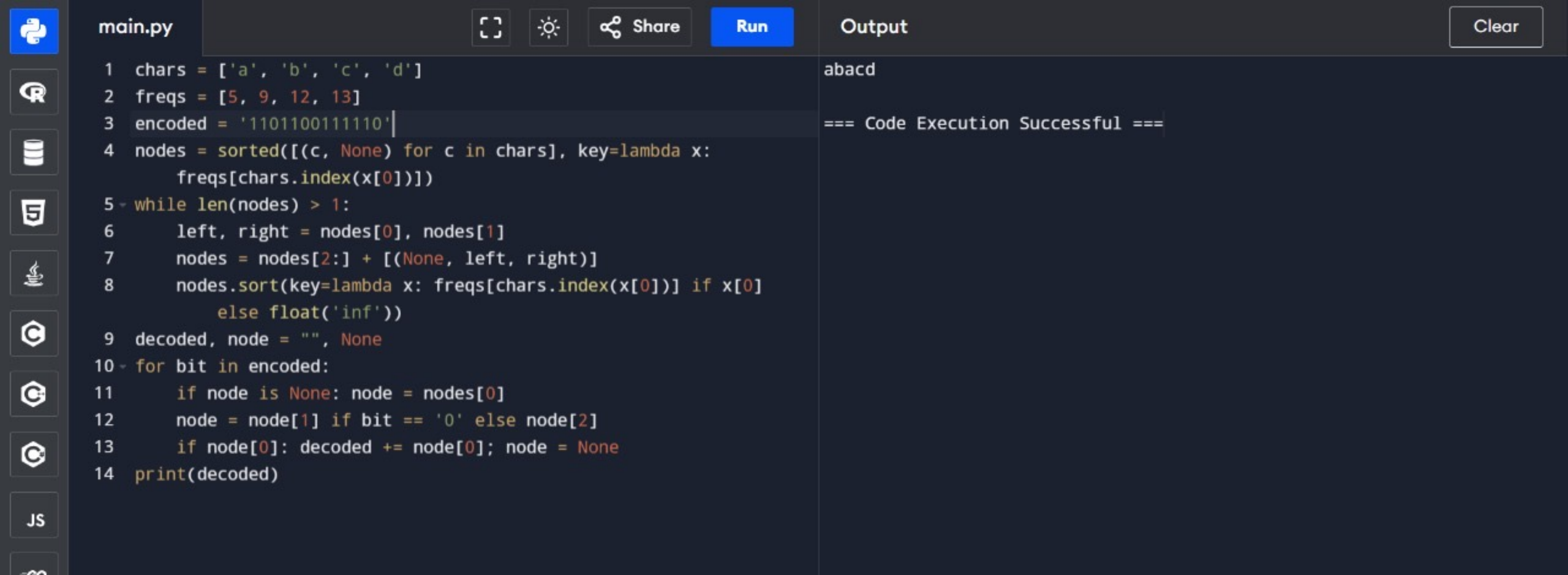
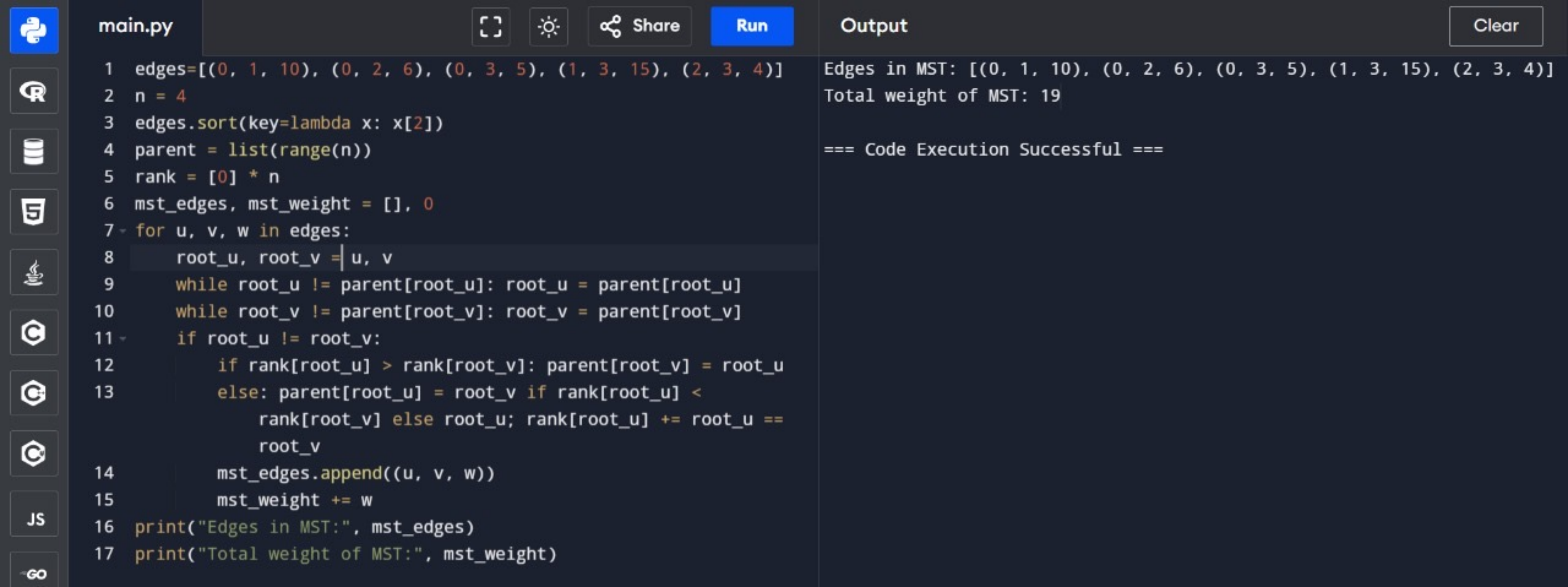


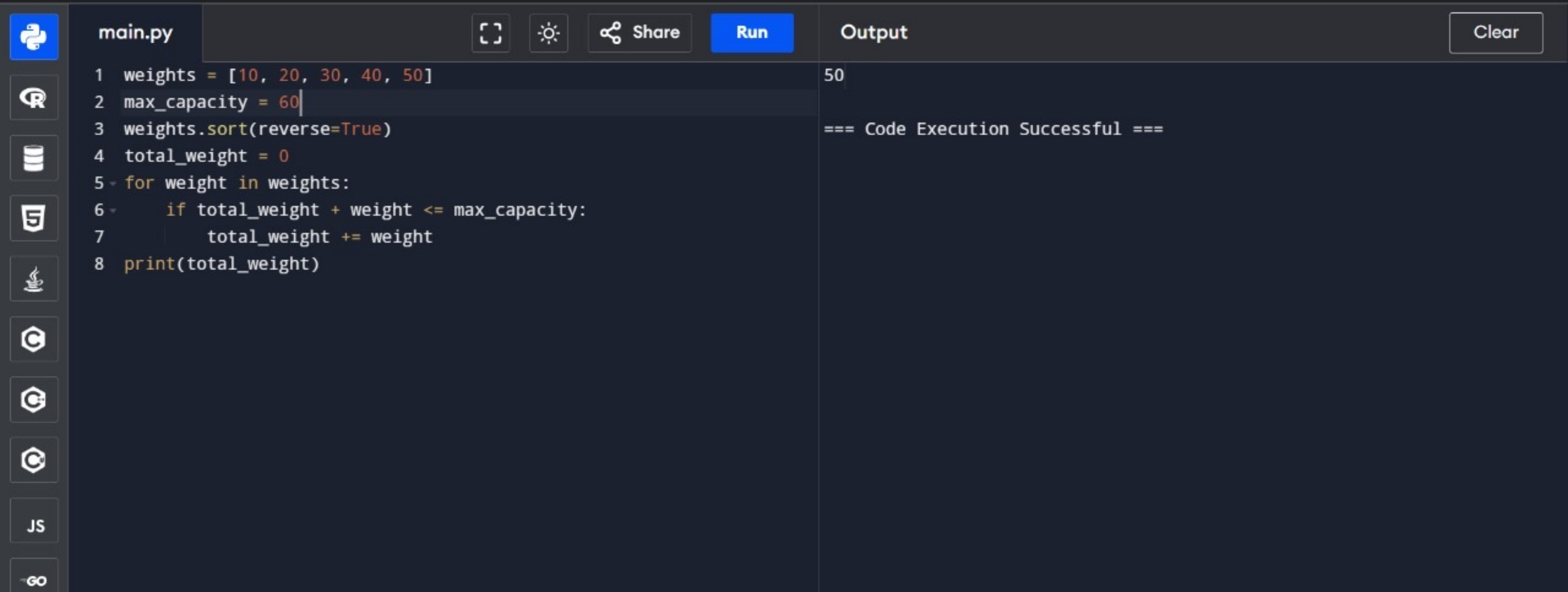
```python
chars = ['a', 'b', 'c', 'd']
freqs = [5, 9, 12, 13]
encoded = '1101100111110'
nodes = sorted([(c, None) for c in chars], key=lambda x:
    freqs[chars.index(x[0])])
while len(nodes) > 1:
    left, right = nodes[0], nodes[1]
    nodes = nodes[2:] + [(None, left, right)]
    nodes.sort(key=lambda x: freqs[chars.index(x[0])] if x[0]
        else float('inf'))
decoded, node = "", None
for bit in encoded:
    if node is None: node = nodes[0]
    node = node[1] if bit == '0' else node[2]
    if node[0]: decoded += node[0]; node = None
print(decoded)
```

```
abacd

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



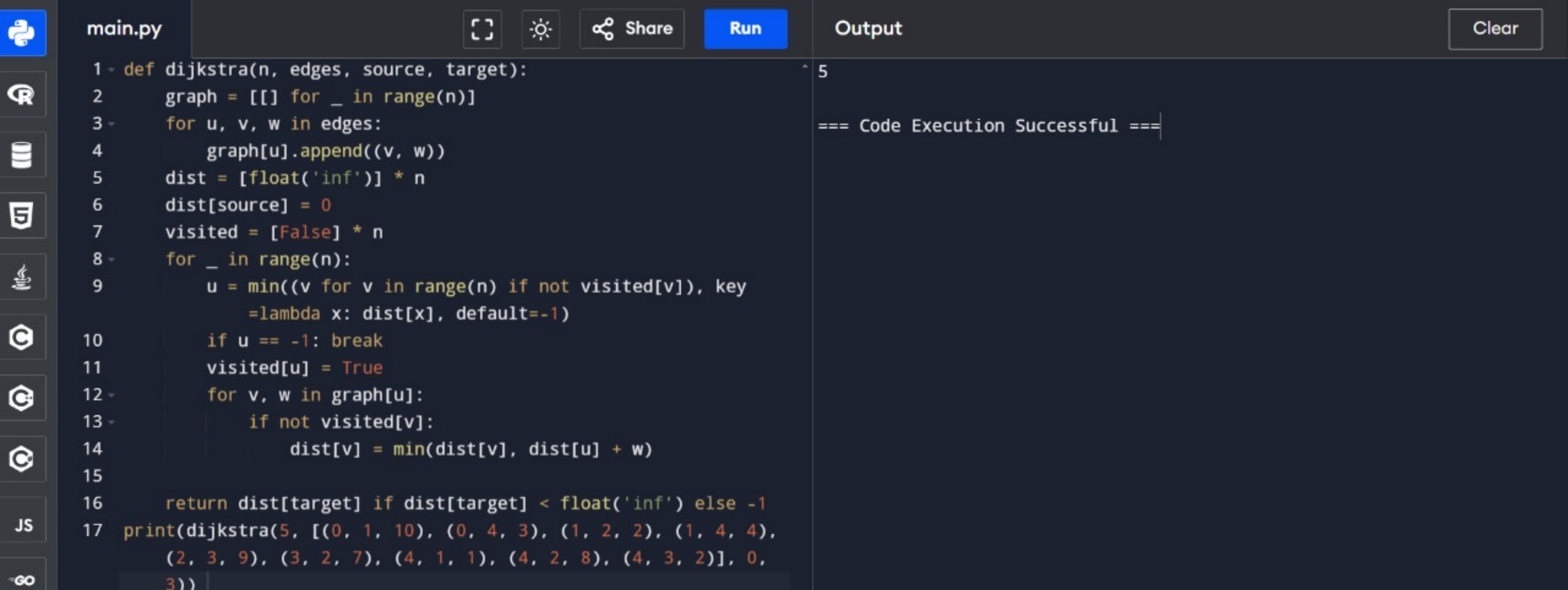
```python
edges=[(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
n = 4
edges.sort(key=lambda x: x[2])
parent = list(range(n))
rank = [0] * n
mst_edges, mst_weight = [], 0
for u, v, w in edges:
    root_u, root_v = u, v
    while root_u != parent[root_u]: root_u = parent[root_u]
    while root_v != parent[root_v]: root_v = parent[root_v]
    if root_u != root_v:
        if rank[root_u] > rank[root_v]: parent[root_v] = root_u
        else: parent[root_u] = root_v if rank[root_u] < 
            rank[root_v] else root_u; rank[root_u] += root_u == 
            root_v
        mst_edges.append((u, v, w))
        mst_weight += w
print("Edges in MST:", mst_edges)
print("Total weight of MST:", mst_weight)
```

```
Edges in MST: [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
Total weight of MST: 19

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



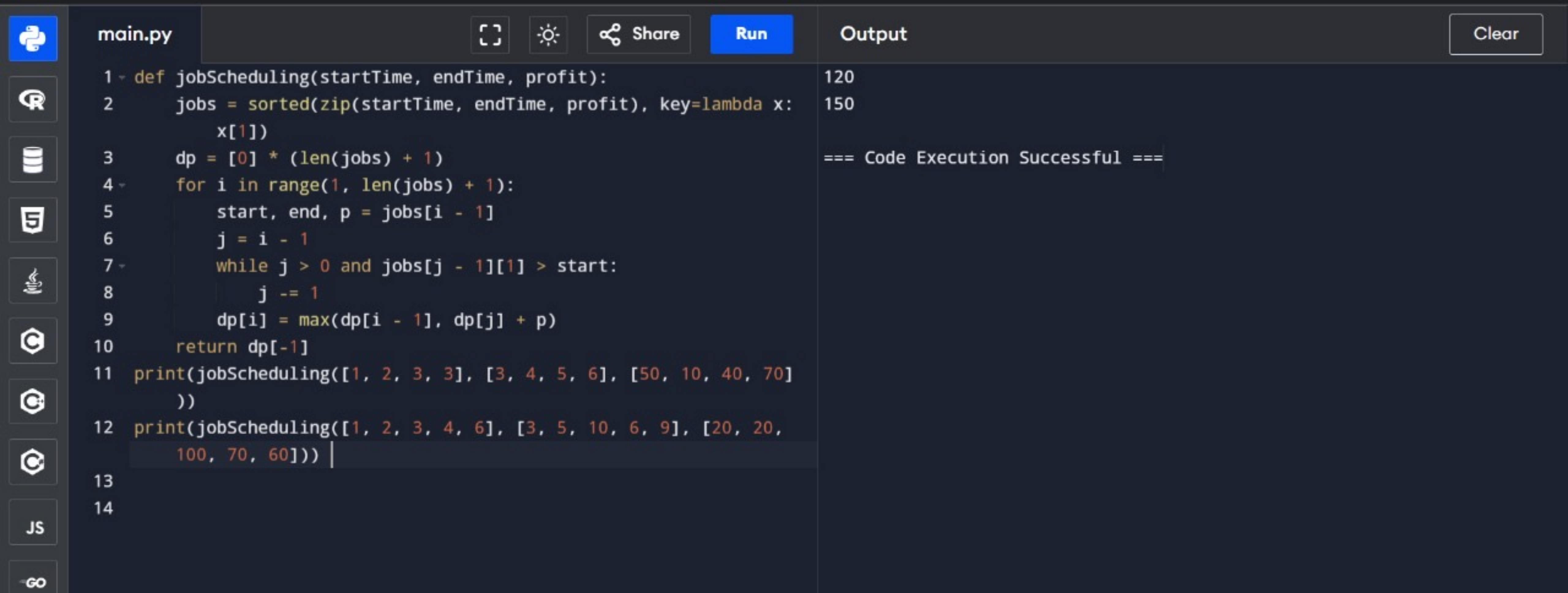
```python
weights = [10, 20, 30, 40, 50]
max_capacity = 60
weights.sort(reverse=True)
total_weight = 0
for weight in weights:
    if total_weight + weight <= max_capacity:
        total_weight += weight
print(total_weight)
```

```
50


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



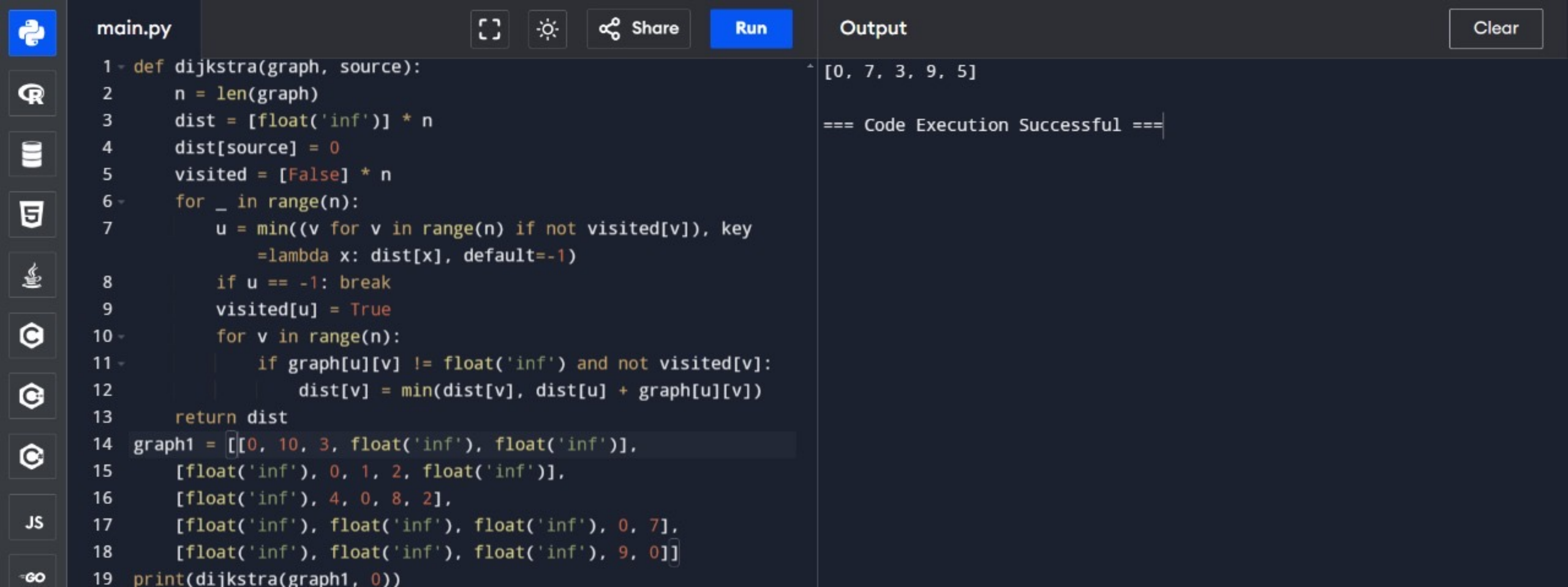
```python
def dijkstra(n, edges, source, target):
    graph = [[] for _ in range(n)]
    for u, v, w in edges:
        graph[u].append((v, w))
    dist = [float('inf')] * n
    dist[source] = 0
    visited = [False] * n
    for _ in range(n):
        u = min((v for v in range(n) if not visited[v]), key
            =lambda x: dist[x], default=-1)
        if u == -1: break
        visited[u] = True
        for v, w in graph[u]:
            if not visited[v]:
                dist[v] = min(dist[v], dist[u] + w)

    return dist[target] if dist[target] < float('inf') else -1
print(dijkstra(5, [(0, 1, 10), (0, 4, 3), (1, 2, 2), (1, 4, 4),
    (2, 3, 9), (3, 2, 7), (4, 1, 1), (4, 2, 8), (4, 3, 2)], 0,
    3))
```

```
5


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



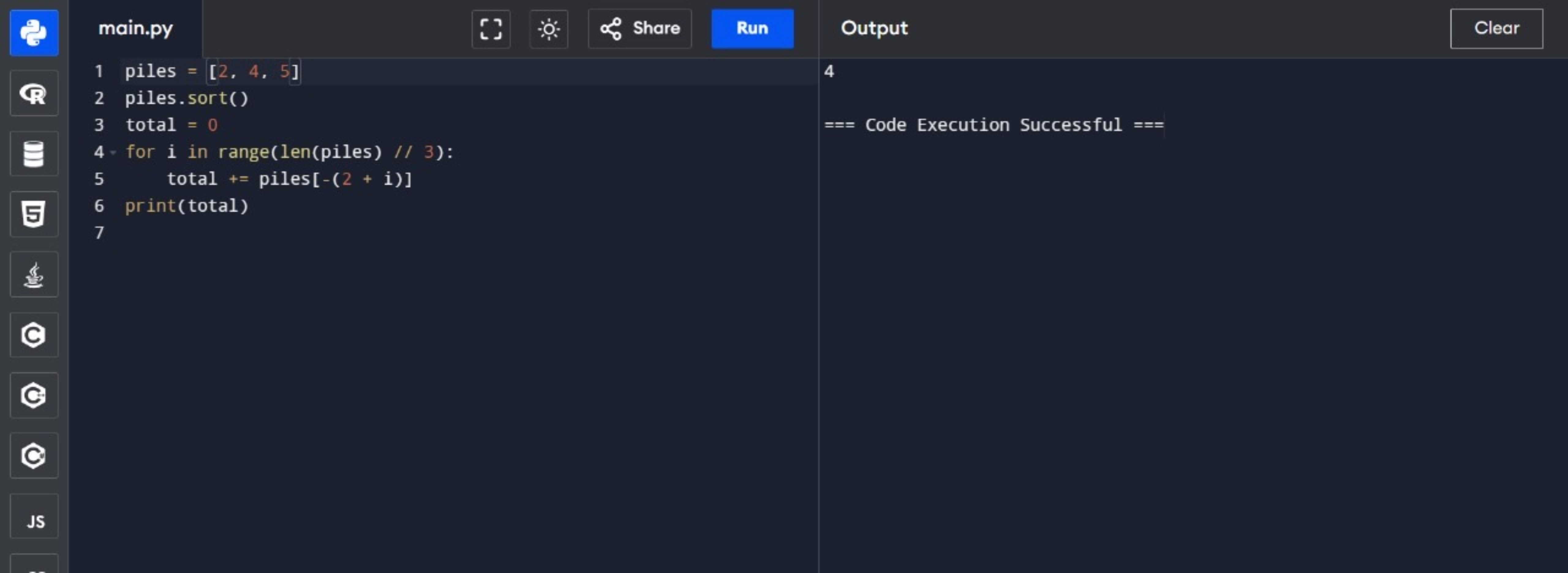
```python
def jobScheduling(startTime, endTime, profit):
    jobs = sorted(zip(startTime, endTime, profit), key=lambda x:
        x[1])
    dp = [0] * (len(jobs) + 1)
    for i in range(1, len(jobs) + 1):
        start, end, p = jobs[i - 1]
        j = i - 1
        while j > 0 and jobs[j - 1][1] > start:
            j -= 1
        dp[i] = max(dp[i - 1], dp[j] + p)
    return dp[-1]
print(jobScheduling([1, 2, 3, 3], [3, 4, 5, 6], [50, 10, 40, 70]
    ))
print(jobScheduling([1, 2, 3, 4, 6], [3, 5, 10, 6, 9], [20, 20,
    100, 70, 60]))
```

```
120
150

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



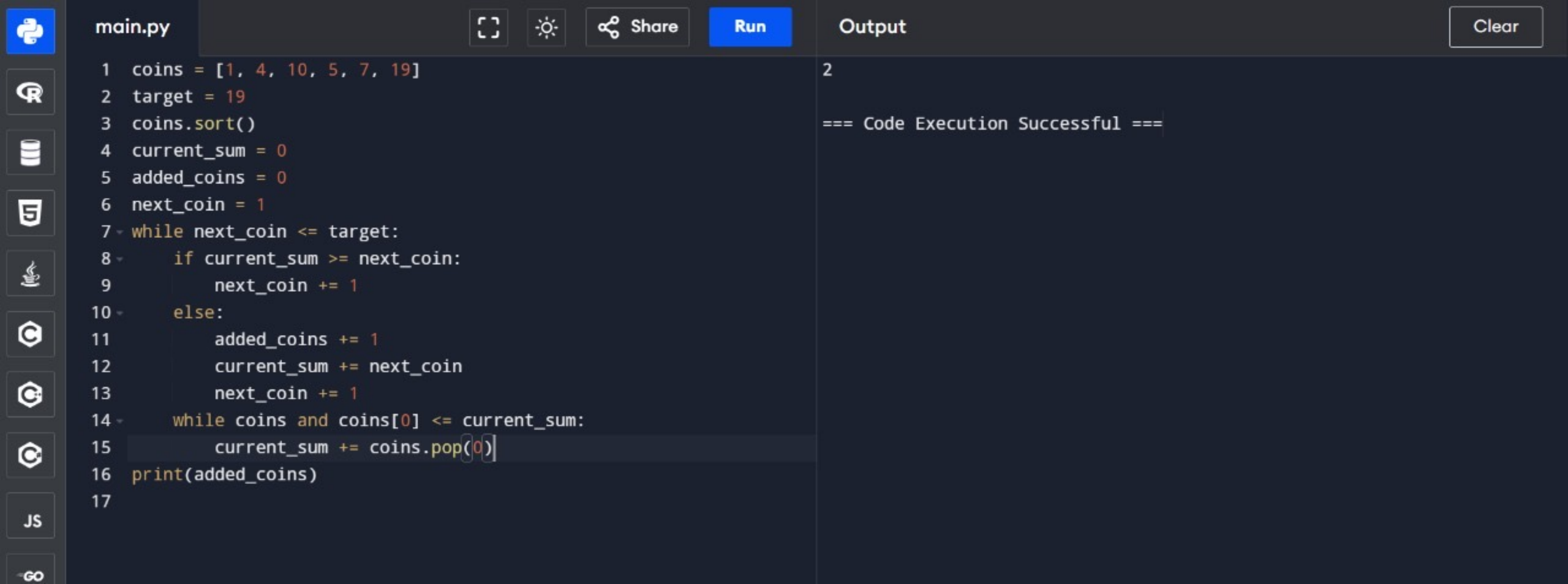
```python
def dijkstra(graph, source):
    n = len(graph)
    dist = [float('inf')] * n
    dist[source] = 0
    visited = [False] * n
    for _ in range(n):
        u = min((v for v in range(n) if not visited[v]), key
            =lambda x: dist[x], default=-1)
        if u == -1: break
        visited[u] = True
        for v in range(n):
            if graph[u][v] != float('inf') and not visited[v]:
                dist[v] = min(dist[v], dist[u] + graph[u][v])
    return dist
graph1 = [[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]]
print(dijkstra(graph1, 0))
```

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

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



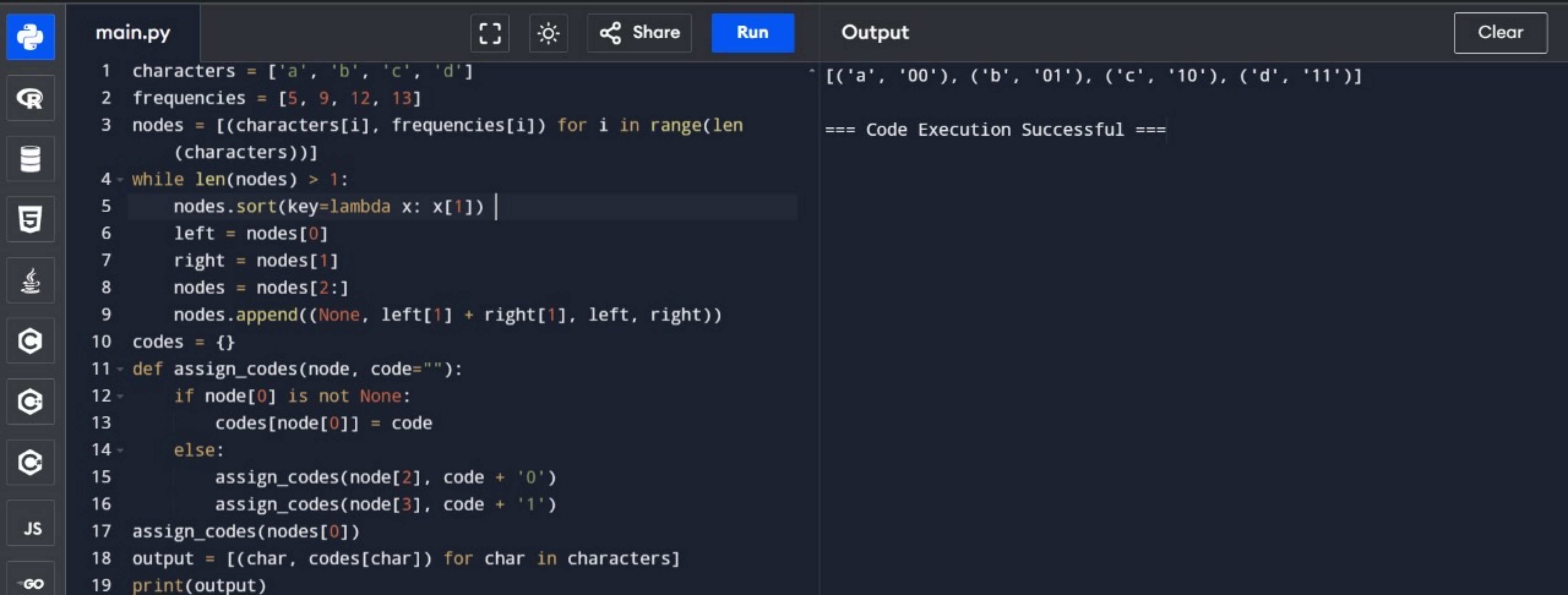
```python
piles = [2, 4, 5]
piles.sort()
total = 0
for i in range(len(piles) // 3):
    total += piles[-(2 + i)]
print(total)
```

```
4


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



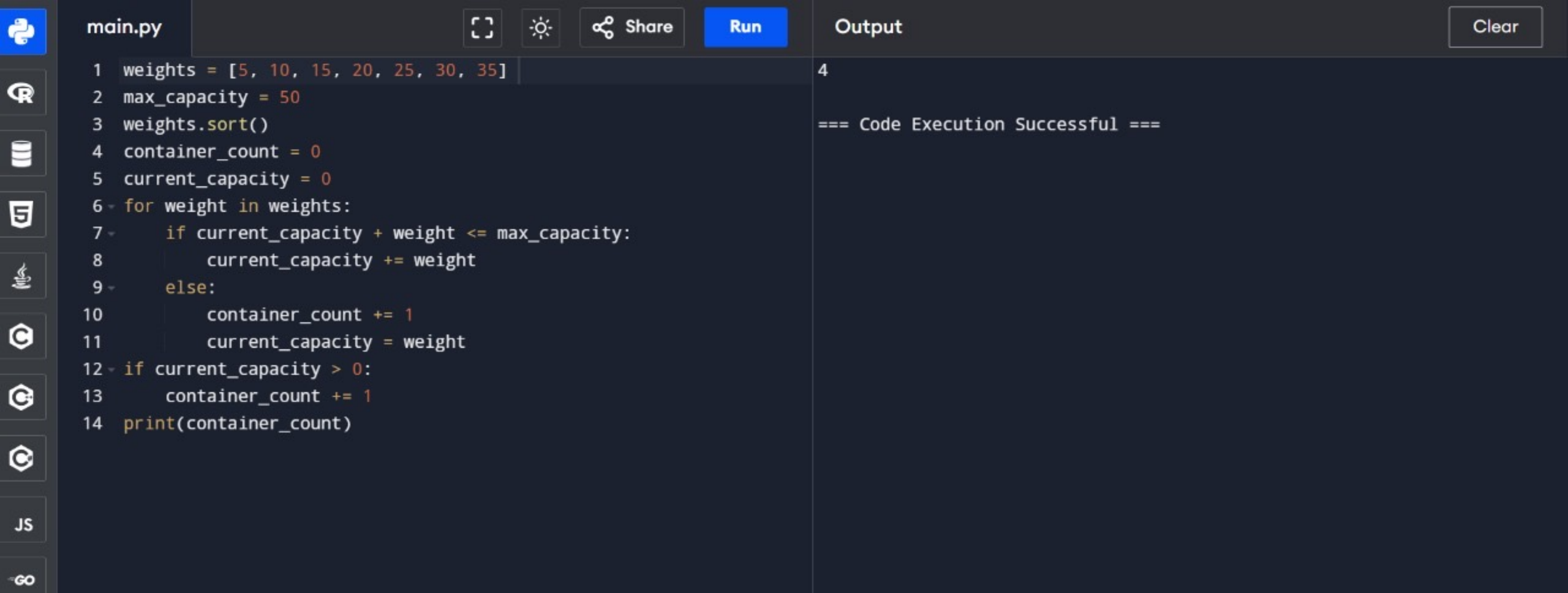
```python
coins = [1, 4, 10, 5, 7, 19]
target = 19
coins.sort()
current_sum = 0
added_coins = 0
next_coin = 1
while next_coin <= target:
    if current_sum >= next_coin:
        next_coin += 1
    else:
        added_coins += 1
        current_sum += next_coin
        next_coin += 1
    while coins and coins[0] <= current_sum:
        current_sum += coins.pop(0)
print(added_coins)
```

```
2


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



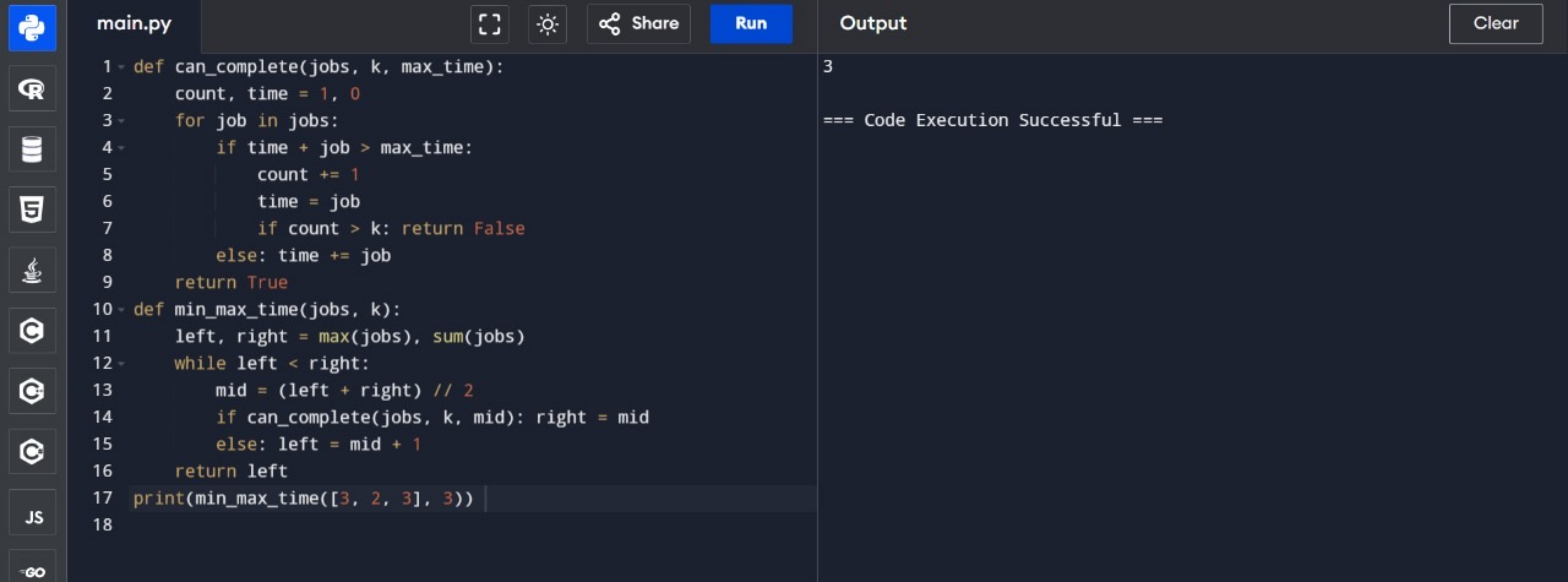
```python
characters = ['a', 'b', 'c', 'd']
frequencies = [5, 9, 12, 13]
nodes = [(characters[i], frequencies[i]) for i in range(len
    (characters))]
while len(nodes) > 1:
    nodes.sort(key=lambda x: x[1])
    left = nodes[0]
    right = nodes[1]
    nodes = nodes[2:]
    nodes.append((None, left[1] + right[1], left, right))
codes = {}
def assign_codes(node, code=""):
    if node[0] is not None:
        codes[node[0]] = code
    else:
        assign_codes(node[2], code + '0')
        assign_codes(node[3], code + '1')
assign_codes(nodes[0])
output = [(char, codes[char]) for char in characters]
print(output)
```

```
[('a', '00'), ('b', '01'), ('c', '10'), ('d', '11')]

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



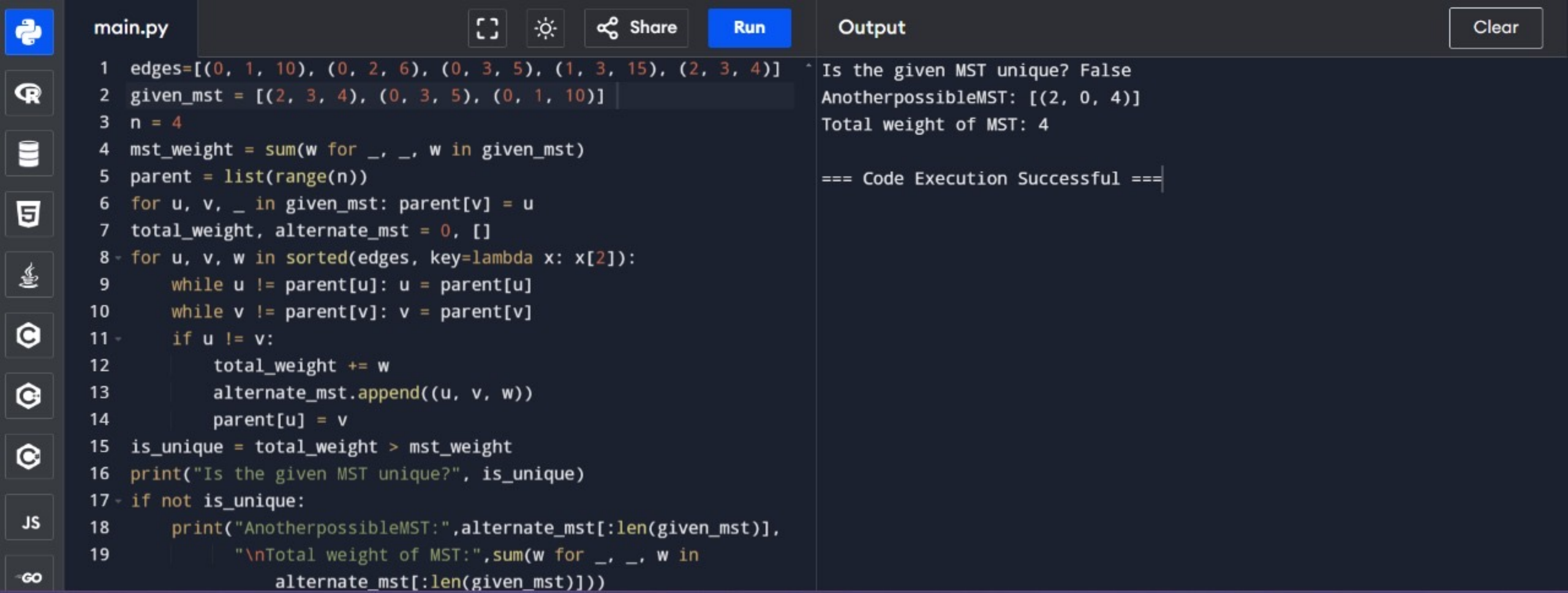
```python
weights = [5, 10, 15, 20, 25, 30, 35]
max_capacity = 50
weights.sort()
container_count = 0
current_capacity = 0
for weight in weights:
    if current_capacity + weight <= max_capacity:
        current_capacity += weight
    else:
        container_count += 1
        current_capacity = weight
if current_capacity > 0:
    container_count += 1
print(container_count)
```

```
4


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



```python
def can_complete(jobs, k, max_time):
    count, time = 1, 0
    for job in jobs:
        if time + job > max_time:
            count += 1
            time = job
            if count > k: return False
        else: time += job
    return True
def min_max_time(jobs, k):
    left, right = max(jobs), sum(jobs)
    while left < right:
        mid = (left + right) // 2
        if can_complete(jobs, k, mid): right = mid
        else: left = mid + 1
    return left
print(min_max_time([3, 2, 3], 3))
```

```
3


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



```python
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)]
n = 4
mst_weight = sum(w for _, _, w in given_mst)
parent = list(range(n))
for u, v, _ in given_mst: parent[v] = u
total_weight, alternate_mst = 0, []
for u, v, w in sorted(edges, key=lambda x: x[2]):
    while u != parent[u]: u = parent[u]
    while v != parent[v]: v = parent[v]
    if u != v:
        total_weight += w
        alternate_mst.append((u, v, w))
        parent[u] = v
is_unique = total_weight > mst_weight
print("Is the given MST unique?", is_unique)
if not is_unique:
    print("AnotherpossibleMST:",alternate_mst[:len(given_mst)],
          "\nTotal weight of MST:",sum(w for _, _, w in
            alternate_mst[:len(given_mst)]))
```

```
Is the given MST unique? False
AnotherpossibleMST: [(2, 0, 4)]
Total weight of MST: 4

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