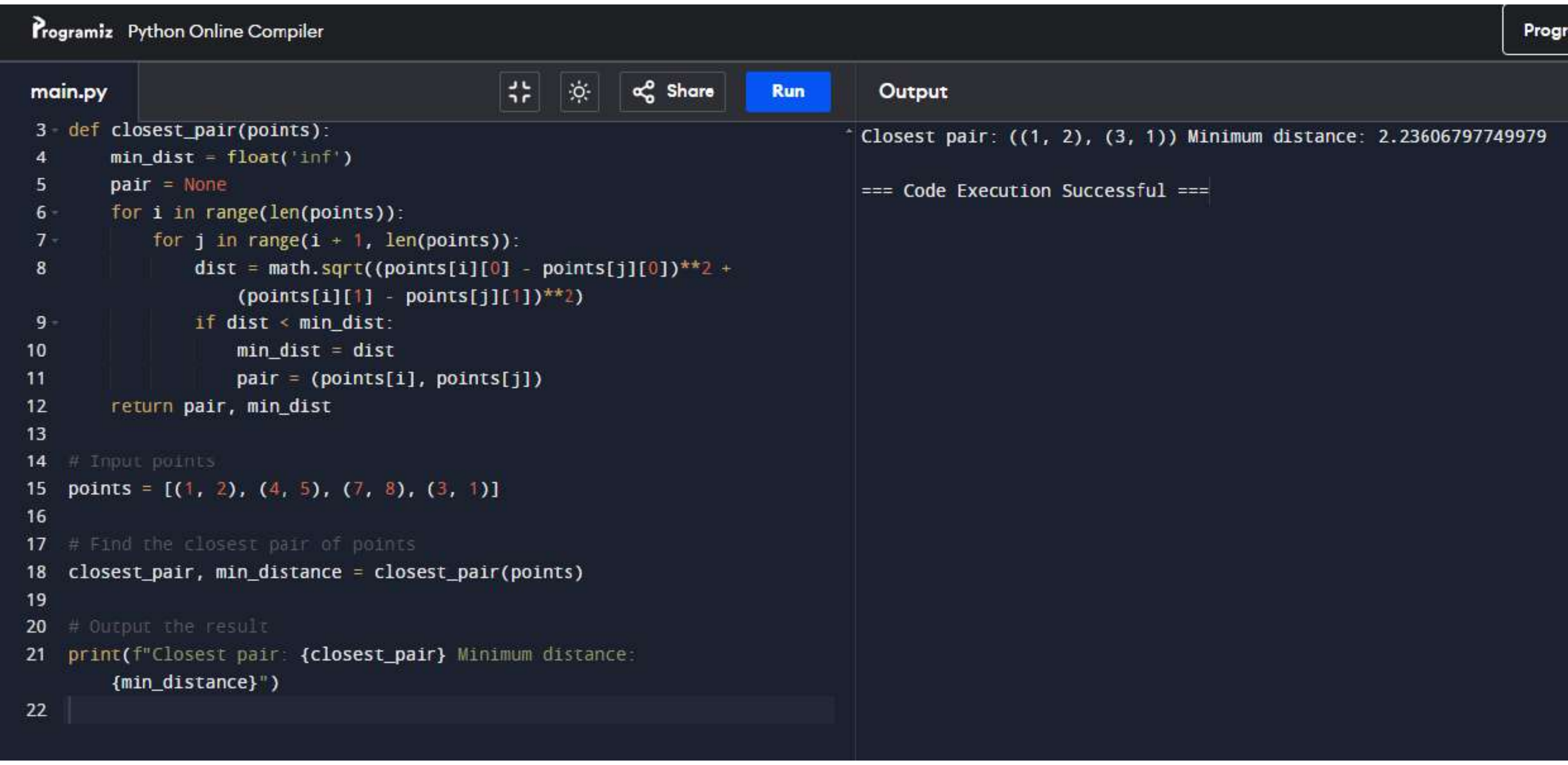
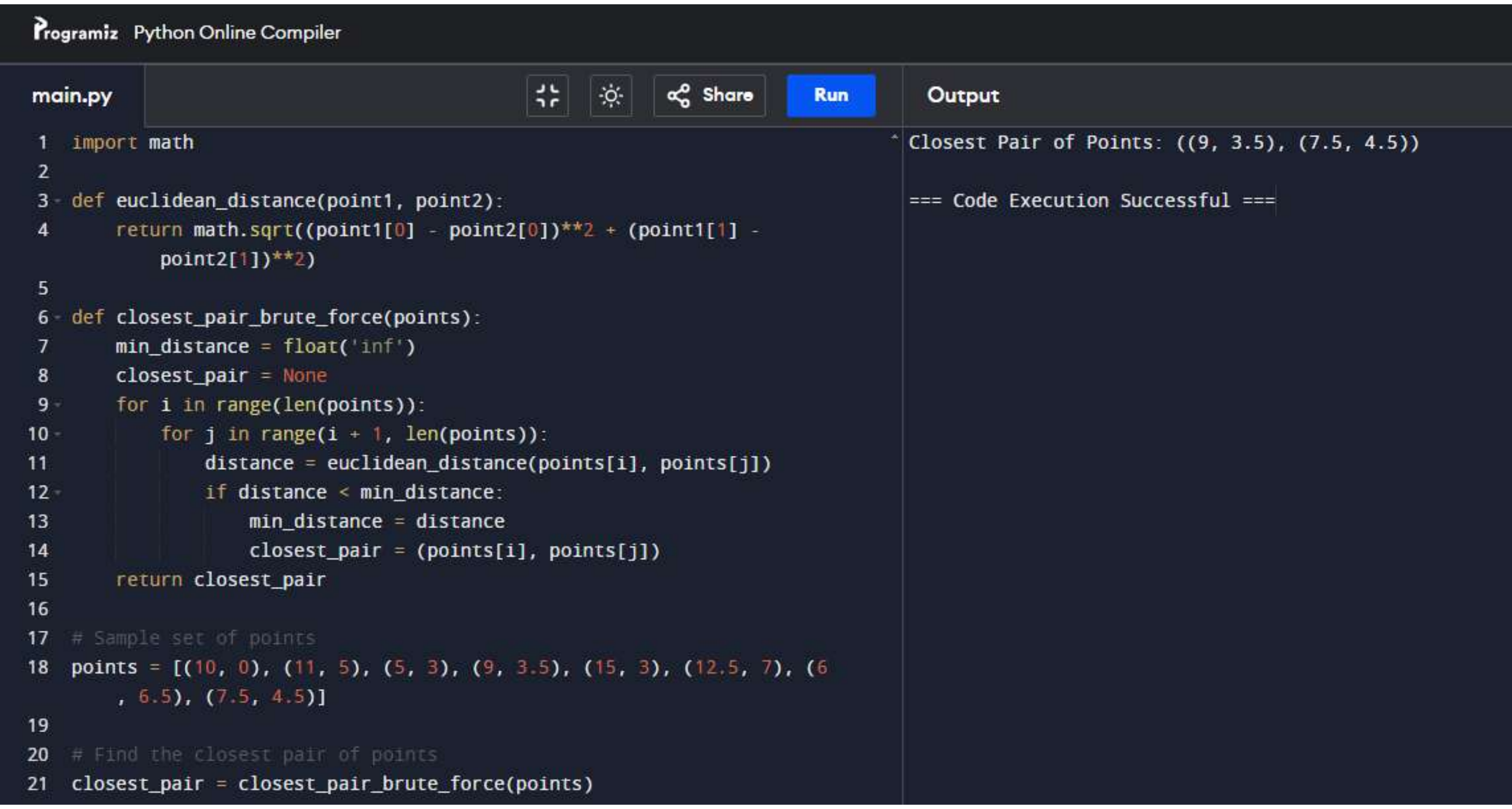


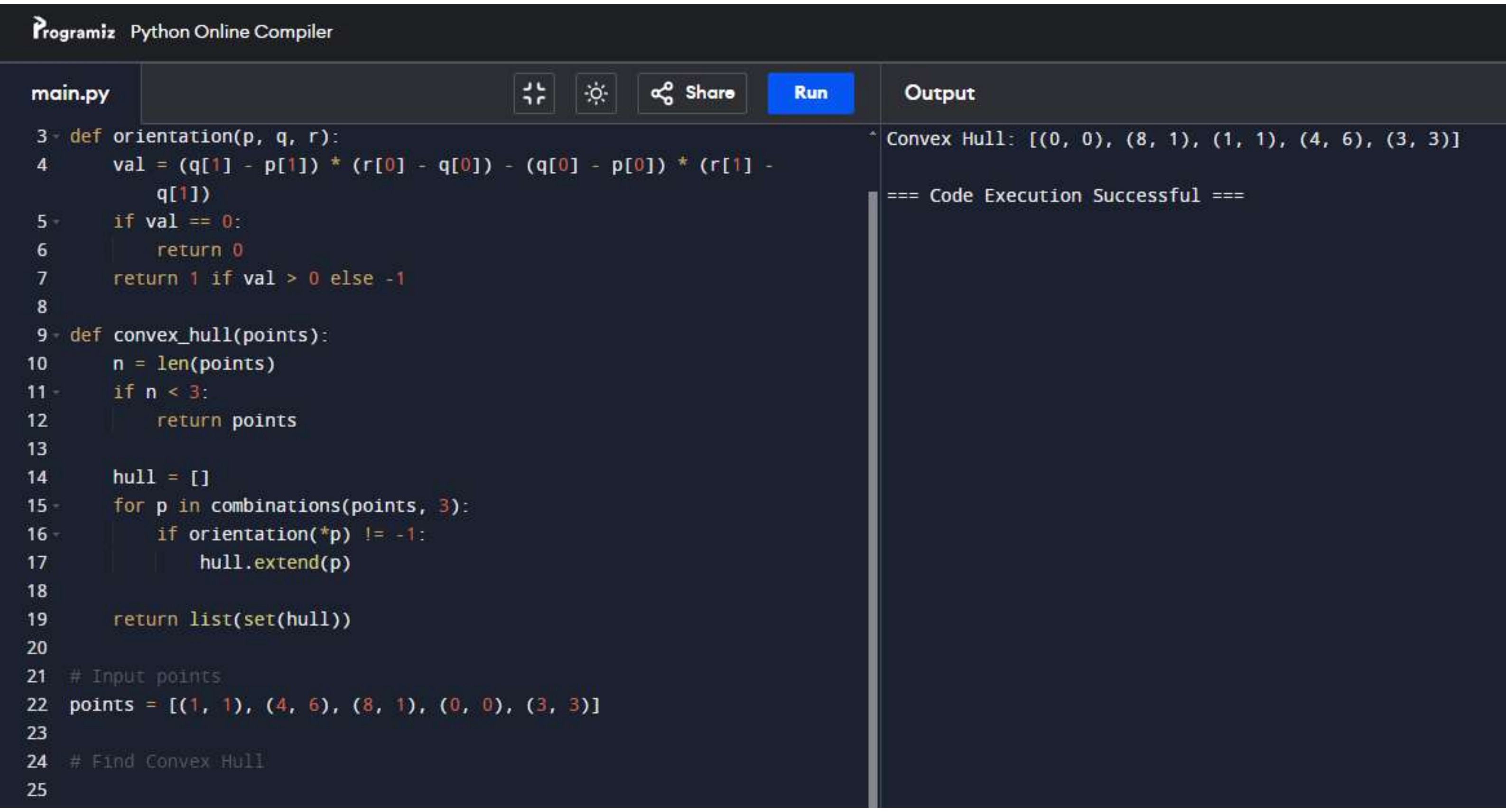
```python
def closest_pair(points):
    min_dist = float('inf')
    pair = None
    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            dist = math.sqrt((points[i][0] - points[j][0])**2 + (points[i][1] - points[j][1])**2)
            if dist < min_dist:
                min_dist = dist
                pair = (points[i], points[j])
    return pair, min_dist

# Input points
points = [(1, 2), (4, 5), (7, 8), (3, 1)]

# Find the closest pair of points
closest_pair, min_distance = closest_pair(points)

# Output the result
print(f"Closest pair: {closest_pair} Minimum distance: {min_distance}")
```

```
Closest pair: ((1, 2), (3, 1)) Minimum distance: 2.23606797749979

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



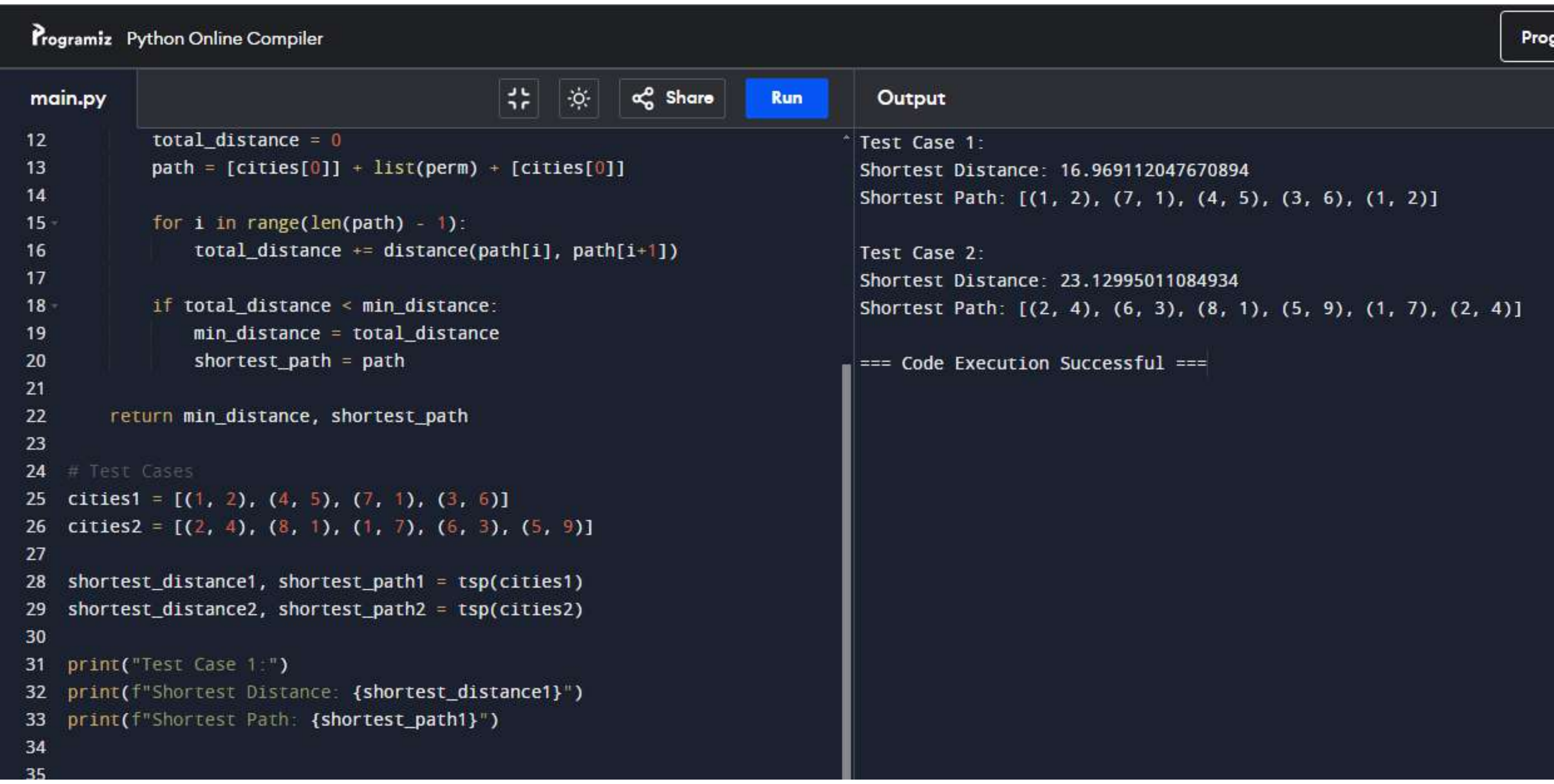
```python
import math

def euclidean_distance(point1, point2):
    return math.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)

def closest_pair_brute_force(points):
    min_distance = float('inf')
    closest_pair = None
    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            distance = euclidean_distance(points[i], points[j])
            if distance < min_distance:
                min_distance = distance
                closest_pair = (points[i], points[j])
    return closest_pair

# Sample set of points
points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5), (7.5, 4.5)]

# Find the closest pair of points
closest_pair = closest_pair_brute_force(points)
```

```
Closest Pair of Points: ((9, 3.5), (7.5, 4.5))

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



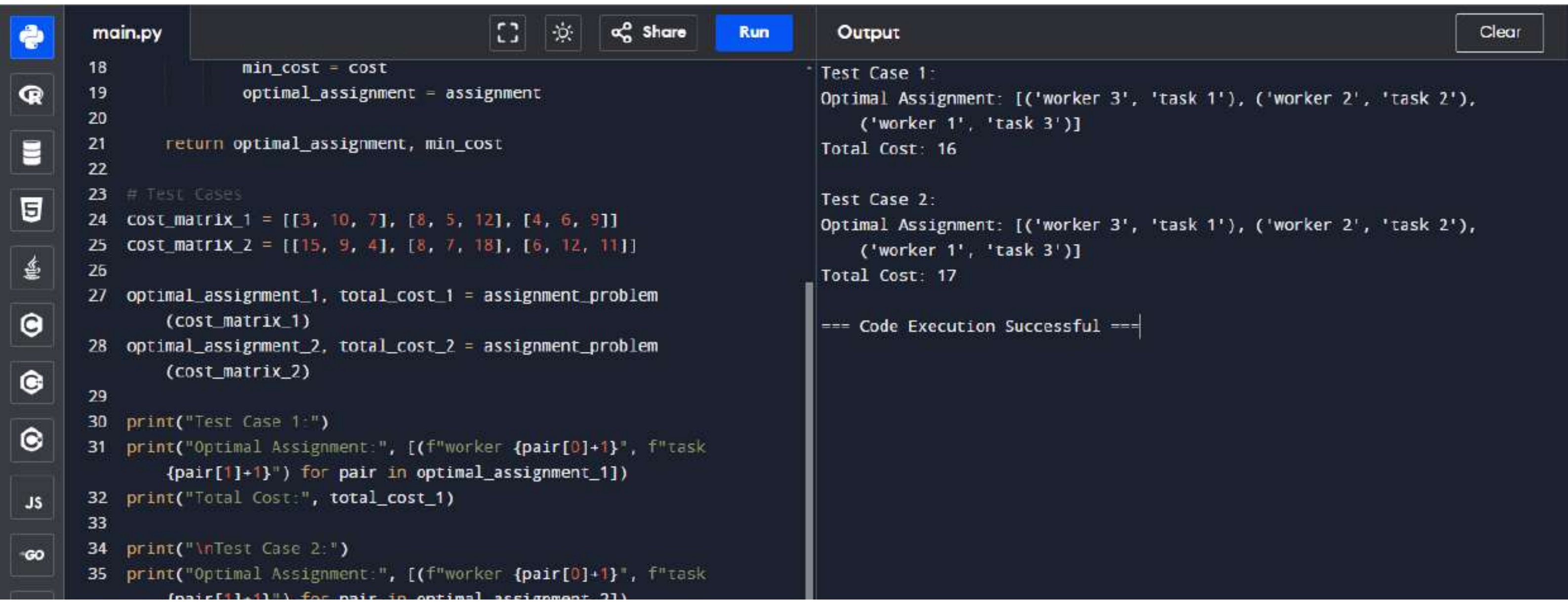
```python
def orientation(p, q, r):
    val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] - q[1])
    if val == 0:
        return 0
    return 1 if val > 0 else -1

def convex_hull(points):
    n = len(points)
    if n < 3:
        return points

    hull = []
    for p in combinations(points, 3):
        if orientation(*p) != -1:
            hull.extend(p)

    return list(set(hull))

# Input points
points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]

# Find Convex Hull

```

```
Convex Hull: [(0, 0), (8, 1), (1, 1), (4, 6), (3, 3)]

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



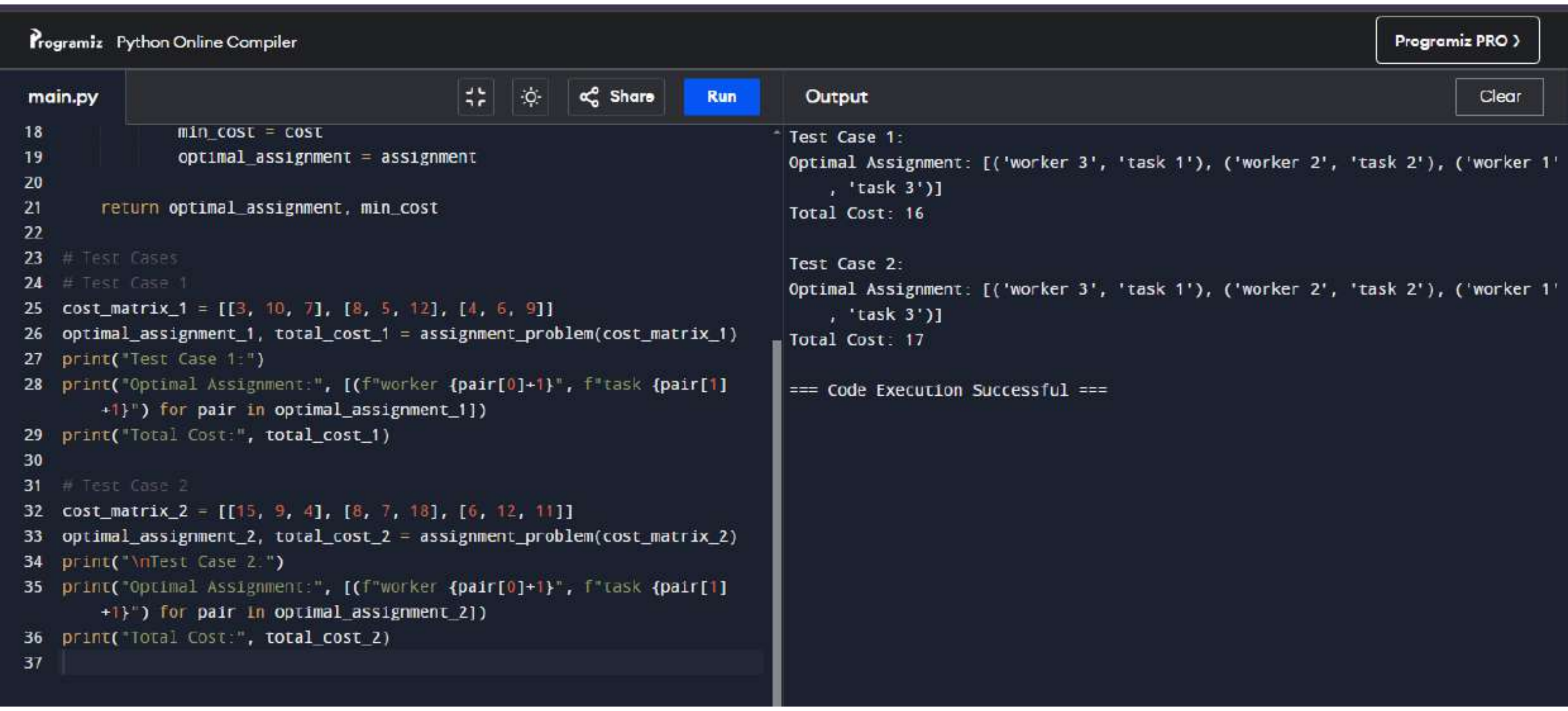
```python
        total_distance = 0
        path = [cities[0]] + list(perm) + [cities[0]]

        for i in range(len(path) - 1):
            total_distance += distance(path[i], path[i+1])

        if total_distance < min_distance:
            min_distance = total_distance
            shortest_path = path

    return min_distance, shortest_path

# Test Cases
cities1 = [(1, 2), (4, 5), (7, 1), (3, 6)]
cities2 = [(2, 4), (8, 1), (1, 7), (6, 3), (5, 9)]

shortest_distance1, shortest_path1 = tsp(cities1)
shortest_distance2, shortest_path2 = tsp(cities2)

print("Test Case 1:")
print(f"Shortest Distance: {shortest_distance1}")
print(f"Shortest Path: {shortest_path1}")
```

```
Test Case 1:
Shortest Distance: 16.969112047670894
Shortest Path: [(1, 2), (7, 1), (4, 5), (3, 6), (1, 2)]

Test Case 2:
Shortest Distance: 23.12995011084934
Shortest Path: [(2, 4), (6, 3), (8, 1), (5, 9), (1, 7), (2, 4)]

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



```python
            min_cost = cost
            optimal_assignment = assignment

    return optimal_assignment, min_cost

# Test Cases
cost_matrix_1 = [[3, 10, 7], [8, 5, 12], [4, 6, 9]]
cost_matrix_2 = [[15, 9, 4], [8, 7, 18], [6, 12, 11]]

optimal_assignment_1, total_cost_1 = assignment_problem(cost_matrix_1)
optimal_assignment_2, total_cost_2 = assignment_problem(cost_matrix_2)

print("Test Case 1:")
print("Optimal Assignment:", [(f"worker {pair[0]+1}", f"task {pair[1]+1}") for pair in optimal_assignment_1])
print("Total Cost:", total_cost_1)

print("\nTest Case 2:")
print("Optimal Assignment:", [(f"worker {pair[0]+1}", f"task {pair[1]+1}") for pair in optimal_assignment_2])
```

```
Test Case 1:
Optimal Assignment: [('worker 3', 'task 1'), ('worker 2', 'task 2'),
    ('worker 1', 'task 3')]
Total Cost: 16

Test Case 2:
Optimal Assignment: [('worker 3', 'task 1'), ('worker 2', 'task 2'),
    ('worker 1', 'task 3')]
Total Cost: 17

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



```python
            min_cost = cost
            optimal_assignment = assignment

    return optimal_assignment, min_cost

# Test Cases
# Test Case 1
cost_matrix_1 = [[3, 10, 7], [8, 5, 12], [4, 6, 9]]
optimal_assignment_1, total_cost_1 = assignment_problem(cost_matrix_1)
print("Test Case 1:")
print("Optimal Assignment:", [(f"worker {pair[0]+1}", f"task {pair[1]
    +1}") for pair in optimal_assignment_1])
print("Total Cost:", total_cost_1)

# Test Case 2
cost_matrix_2 = [[15, 9, 4], [8, 7, 18], [6, 12, 11]]
optimal_assignment_2, total_cost_2 = assignment_problem(cost_matrix_2)
print("\nTest Case 2:")
print("Optimal Assignment:", [(f"worker {pair[0]+1}", f"task {pair[1]
    +1}") for pair in optimal_assignment_2])
print("Total Cost:", total_cost_2)
```

```
Test Case 1:
Optimal Assignment: [('worker 3', 'task 1'), ('worker 2', 'task 2'), ('worker 1'
    , 'task 3')]
Total Cost: 16

Test Case 2:
Optimal Assignment: [('worker 3', 'task 1'), ('worker 2', 'task 2'), ('worker 1'
    , 'task 3')]
Total Cost: 17

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