

main.py



Run

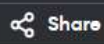
Output

```
3- def closest_pair(points):
4     min_dist = float('inf')
5     pair = None
6     for i in range(len(points)):
7         for j in range(i + 1, len(points)):
8             dist = math.sqrt((points[i][0] - points[j][0])**2 +
9                             (points[i][1] - points[j][1])**2)
9-             if dist < min_dist:
10                min_dist = dist
11                pair = (points[i], points[j])
12     return pair, min_dist
13
14 # Input points
15 points = [(1, 2), (4, 5), (7, 8), (3, 1)]
16
17 # Find the closest pair of points
18 closest_pair, min_distance = closest_pair(points)
19
20 # Output the result
21 print(f"Closest pair: {closest_pair} Minimum distance:
22     {min_distance}")
```

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

=== Code Execution Successful ===

main.py



Share

Run

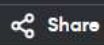
Output

```
1 import math
2
3 def euclidean_distance(point1, point2):
4     return math.sqrt((point1[0] - point2[0])**2 + (point1[1] -
5         point2[1])**2)
6
7 def closest_pair_brute_force(points):
8     min_distance = float('inf')
9     closest_pair = None
10    for i in range(len(points)):
11        for j in range(i + 1, len(points)):
12            distance = euclidean_distance(points[i], points[j])
13            if distance < min_distance:
14                min_distance = distance
15                closest_pair = (points[i], points[j])
16    return closest_pair
17
18 # Sample set of points
19 points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6
20     , 6.5), (7.5, 4.5)]
21
22 # Find the closest pair of points
23 closest_pair = closest_pair_brute_force(points)
```

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

=== Code Execution Successful ===

main.py



Run

Output

```
3 def orientation(p, q, r):
4     val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] -
      q[1])
5     if val == 0:
6         return 0
7     return 1 if val > 0 else -1
8
9 def convex_hull(points):
10     n = len(points)
11     if n < 3:
12         return points
13
14     hull = []
15     for p in combinations(points, 3):
16         if orientation(*p) != -1:
17             hull.extend(p)
18
19     return list(set(hull))
20
21 # Input points
22 points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
23
24 # Find Convex Hull
25
```

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

=== Code Execution Successful ===

main.py



Run

Output

```
12 total_distance = 0
13 path = [cities[0]] + list(perm) + [cities[0]]
14
15 for i in range(len(path) - 1):
16     total_distance += distance(path[i], path[i+1])
17
18 if total_distance < min_distance:
19     min_distance = total_distance
20     shortest_path = path
21
22 return min_distance, shortest_path
23
24 # Test Cases
25 cities1 = [(1, 2), (4, 5), (7, 1), (3, 6)]
26 cities2 = [(2, 4), (8, 1), (1, 7), (6, 3), (5, 9)]
27
28 shortest_distance1, shortest_path1 = tsp(cities1)
29 shortest_distance2, shortest_path2 = tsp(cities2)
30
31 print("Test Case 1:")
32 print(f"Shortest Distance: {shortest_distance1}")
33 print(f"Shortest Path: {shortest_path1}")
34
35
```

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 ===



main.py



Run

```
18     min_cost = cost
19     optimal_assignment = assignment
20
21     return optimal_assignment, min_cost
22
23 # Test Cases
24 cost_matrix_1 = [[3, 10, 7], [8, 5, 12], [4, 6, 9]]
25 cost_matrix_2 = [[15, 9, 4], [8, 7, 18], [6, 12, 11]]
26
27 optimal_assignment_1, total_cost_1 = assignment_problem
28     (cost_matrix_1)
29 optimal_assignment_2, total_cost_2 = assignment_problem
30     (cost_matrix_2)
31
32 print("Test Case 1:")
33 print("Optimal Assignment:", [(f"worker {pair[0]+1}", f"task
34     {pair[1]+1}") for pair in optimal_assignment_1])
35 print("Total Cost:", total_cost_1)
36
37 print("\nTest Case 2:")
38 print("Optimal Assignment:", [(f"worker {pair[0]+1}", f"task
39     {pair[1]+1}") for pair in optimal_assignment_2])
```

Output

Clear

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 ===

main.py



Run

Output

Clear

```
18     min_cost = cost
19     optimal_assignment = assignment
20
21     return optimal_assignment, min_cost
22
23 # Test Cases
24 # Test Case 1
25 cost_matrix_1 = [[3, 10, 7], [8, 5, 12], [4, 6, 9]]
26 optimal_assignment_1, total_cost_1 = assignment_problem(cost_matrix_1)
27 print("Test Case 1:")
28 print("Optimal Assignment:", [(f"worker {pair[0]+1}", f"task {pair[1]+1}") for pair in optimal_assignment_1])
29 print("Total Cost:", total_cost_1)
30
31 # Test Case 2
32 cost_matrix_2 = [[15, 9, 4], [8, 7, 18], [6, 12, 11]]
33 optimal_assignment_2, total_cost_2 = assignment_problem(cost_matrix_2)
34 print("\nTest Case 2:")
35 print("Optimal Assignment:", [(f"worker {pair[0]+1}", f"task {pair[1]+1}") for pair in optimal_assignment_2])
36 print("Total Cost:", total_cost_2)
37
```

```
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 ===
```