main.py

Share

Run

```python
import math
def calculate_distance(point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 - (point1[1] - point2[1]) ** 2)
def closest_pair(points):
    min_distance = float('inf')
    closest_points = (None, None)
    for i in range(len(points)):
        for j in range(i - 1, len(points)):
            distance = calculate_distance(points[i], points[j])
            if distance < min_distance:
                min_distance = distance
                closest_points = (points[i], points[j])
    return closest_points, min_distance
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_points, min_distance = closest_pair(points)
print(f"Closest pair: {closest_points[0]} - {closest_points[1]}")
print(f"Minimum distance: {min_distance}")
```

Output

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

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

main.py Share **Run**

```python
import math
from itertools import combinations
def 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, None)
    num_points = len(points)
    for i in range(num_points):
        for j in range(i + 1, num_points):
            d = distance(points[i], points[j])
            if d < min_distance:
                min_distance = d
                closest_pair = (points[i], points[j])
    return closest_pair, min_distance
def on_the_left(p1, p2, p):
    return (p2[0] - p1[0]) * (p[1] - p1[1]) - (p[0] - p1[0]) * (p2[1] - p1[1]) > 0
def convex_hull_brute_force(points):
    hull = []
    for (p1, p2) in combinations(points, 2):
        left_set = []
        right_set = []
```

Output

```
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979
Convex hull: [(5, 3), (6, 6.5), (10, 0), (12.5, 7), (15, 3)]

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

```python
        right_set = []
        for p in points:
            if p != p1 and p != p2:
                if on_the_left(p1, p2, p):
                    left_set.append(p)
                else:
                    right_set.append(p)
        if len(left_set) == 0 or len(right_set) == 0:
            if p1 not in hull:
                hull.append(p1)
            if p2 not in hull:
                hull.append(p2)
    return hull
points_closest_pair = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_pair, min_distance = closest_pair_brute_force(points_closest_pair)
print(f"Closest pair: {closest_pair[0]} - {closest_pair[1]}")
print(f"Minimum distance: {min_distance}")
points_convex_hull = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7),
    (6, 6.5), (7.5, 4.5)]
convex_hull = convex_hull_brute_force(points_convex_hull)
convex_hull.sort(key=lambda p: (p[0], p[1]))
print("Convex hull:", convex_hull)
```

```
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979
Convex hull: [(5, 3), (6, 6.5), (10, 0), (12.5, 7), (15, 3)]

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

main.py

Share Run

```python
from itertools import combinations
def on_the_left(p1, p2, p):
    return (p2[0] - p1[0]) * (p[1] - p1[1]) - (p[0] - p1[0]) * (p2[1] - p1[1]) > 0
def convex_hull_brute_force(points):
    hull = []
    for (p1, p2) in combinations(points, 2):
        left_set = []
        right_set = []
        for p in points:
            if p != p1 and p != p2:
                if on_the_left(p1, p2, p):
                    left_set.append(p)
                else:
                    right_set.append(p)
        if len(left_set) == 0 or len(right_set) == 0:
            if p1 not in hull:
                hull.append(p1)
            if p2 not in hull:
                hull.append(p2)
    hull.sort(key=lambda p: (p[0], p[1]))
    return hull
points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
convex_hull = convex_hull_brute_force(points)
print("Convex Hull:", convex_hull)
```

Output

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

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

main.py

Share

Run

```python
import itertools
import math
def distance(city1, city2):
    return math.sqrt((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1]) ** 2)
def tsp(cities):
    min_distance = float('inf')
    shortest_path = []
    for perm in itertools.permutations(cities[1:]):
        current_path = [cities[0]] + list(perm) + [cities[0]]
        current_distance = 0
        for i in range(len(current_path) - 1):
            current_distance += distance(current_path[i], current_path[i + 1])
        if current_distance < min_distance:
            min_distance = current_distance
            shortest_path = current_path
    return min_distance, shortest_path
def test_tsp():
    test_cases = [
        ([(1, 2), (4, 5), (7, 1), (3, 6)], "Test Case 1"),
    ]
    for cities, description in test_cases:
        min_distance, shortest_path = tsp(cities)
        print(f"{description}:")
        print(f"Shortest Distance: {min_distance}")
        print(f"Shortest Path: {shortest_path}")
        print()
test_tsp()
```

Output

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


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

main.py

Share Run

```python
import itertools
def total_cost(assignment, cost_matrix):
    total = 0
    for worker, task in enumerate(assignment):
        total += cost_matrix[worker][task]
    return total
def assignment_problem(cost_matrix):
    num_workers = len(cost_matrix)
    min_cost = float('inf')
    optimal_assignment = []
    for perm in itertools.permutations(range(num_workers)):
        current_cost = total_cost(perm, cost_matrix)
        if current_cost < min_cost:
            min_cost = current_cost
            optimal_assignment = perm
    optimal_assignment = [(worker + 1, task + 1) for worker, task in enumerate(optimal_assignment)]
    return optimal_assignment, min_cost
def test_assignment_problem():
    test_cases = [
        ([[3, 10, 7], [8, 5, 12], [4, 6, 9]], "Test Case 1"),
    ]
    for cost_matrix, description in test_cases:
        optimal_assignment, min_cost = assignment_problem(cost_matrix)
        print(f"Optimal Assignment: {optimal_assignment}")
        print(f"Total Cost: {min_cost}")
        print()
test_assignment_problem()
```

Output

```
Optimal Assignment: [(1, 3), (2, 2), (3, 1)]
Total Cost: 16


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

main.py

Share Run

```python
import itertools
def total_value(selected_items, values):
    return sum(values[i] for i in selected_items)
def is_feasible(selected_items, weights, capacity):
    return sum(weights[i] for i in selected_items) <= capacity
def knapsack_problem(weights, values, capacity):
    num_items = len(weights)
    max_value = 0
    optimal_selection = []
    for r in range(num_items + 1):
        for combination in itertools.combinations(range(num_items), r):
            if is_feasible(combination, weights, capacity):
                current_value = total_value(combination, values)
                if current_value > max_value:
                    max_value = current_value
                    optimal_selection = combination
    return list(optimal_selection), max_value
def test_knapsack_problem():
    test_cases = [
        ([2, 3, 1], [4, 5, 3], 4, "Test Case 1"),
    ]
    for weights, values, capacity, description in test_cases:
        optimal_selection, max_value = knapsack_problem(weights, values, capacity)
        print(f"Optimal Selection: {optimal_selection}")
        print(f"Total Value: {max_value}")
        print()
test_knapsack_problem()
```

Output

```
Optimal Selection: [1, 2]
Total Value: 8


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