main.py

Share Run

Output Clear

```python
def closest_pair(points):
    min_distance = float('inf')
    closest_points = ()
    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            distance_squared = (points[j][0] - points[i][0])**2 +
                (points[j][1] - points[i][1])**2

            if distance_squared < min_distance:
                min_distance = distance_squared
                closest_points = (points[i], points[j])
    return closest_points, min_distance**0.5
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
pair, distance = closest_pair(points)
print("Closest pair:", pair[0], "-", pair[1])
print("Minimum distance:", distance)

```

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

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

main.py

Share Run

```python
def euclidean_distance(point1, point2):
    return ((point2[0] - point1[0]) ** 2 + (point2[1] - point1[1]
        ) ** 2) ** 0.5
def closest_pair(points):
    min_distance = float('inf')
    closest_points = ()
    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_points = (points[i], points[j])
    return closest_points, min_distance
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
pair, distance = closest_pair(points)
print("Closest pair:", pair[0], "-", pair[1])
print("Minimum distance:", distance)
```

Output

Clear

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

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

main.py

Share Run

```python
def distance(c1, c2):
    return ((c1[0] - c2[0]) ** 2 + (c1[1] - c2[1]) ** 2) ** 0.5
def tsp(cities):
    n, min_dist, best_path = len(cities), float('inf'), []
    def visit(path, visited):
        nonlocal min_dist, best_path
        if len(path) == n:
            path.append(path[0])
            dist = sum(distance(path[i], path[i + 1]) for i in
                range(n))
            if dist < min_dist: min_dist, best_path = dist,
                path[:]
            path.pop()
            return
        for i in range(n):
            if i not in visited:
                visit(path + [cities[i]], visited | {i})
    visit([], set())
    return min_dist, best_path
for i, cities in enumerate([[(1, 2), (4, 5), (7, 1), (3, 6)],[
```

Output

Clear

```
Test Case 1 : 16.969112047670894 | [(1, 2), (7, 1), (4, 5), (3, 6), (1,
    2)]
Test Case 2 : 23.12995011084934 | [(2, 4), (6, 3), (8, 1), (5, 9), (1,
    7), (2, 4)]

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

main.py

Share Run

```python
def total_value(selected, values):
    return sum(values[i] for i in selected)
def is_feasible(selected, weights, capacity):
    return sum(weights[i] for i in selected) <= capacity
def knapsack(weights, values, capacity):
    best_value, best_selection = 0, []
    def search(selected):
        nonlocal best_value, best_selection
        if is_feasible(selected, weights, capacity):
            value = total_value(selected, values)
            if value > best_value:
                best_value, best_selection = value, selected[:]
        for i in range(len(weights)):
            if i not in selected:
                search(selected + [i])
    search([])
    return best_selection, best_value
cases=[([2, 3, 1], [4,5,3],4),([1, 2, 3, 4], [2, 4, 6, 3], 6)]
for i, (w, v, c) in enumerate(cases, 1):
    sel, val = knapsack(w, v, c)
```

Output Clear

```
Test Case 1 : Selection: [1, 2] Value: 8
Test Case 2 : Selection: [0, 1, 2] Value: 12

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

main.py

Share Run

```python
def is_left_turn(p1, p2, p3):
    return (p2[0] - p1[0]) * (p3[1] - p1[1]) - (p3[0] - p1[0])
        * (p2[1] - p1[1]) > 0
def convex_hull(points):
    leftmost = min(points)
    hull = [leftmost]
    while True:
        p1 = hull[-1]
        next_point = points[0]
        for p2 in points:
            if next_point == p1 or is_left_turn(p1, next_point,
                p2):
                next_point = p2
        if next_point == leftmost:
            break
        hull.append(next_point)
    return hull
points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
print("Convex Hull:", convex_hull(points))

```

Output

Clear

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

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

main.py

Share Run

```
    def visit(path, visited):
        nonlocal min_dist, best_path
        if len(path) == n:
            path.append(path[0])
            dist = sum(distance(path[i], path[i + 1]) for i in
                range(n))
            if dist < min_dist: min_dist, best_path = dist,
                path[:]
            path.pop()
            return
        for i in range(n):
            if i not in visited:
                visit(path + [cities[i]], visited | {i})
    visit([], set())
    return min_dist, best_path
for i, cities in enumerate([[(1, 2), (4, 5), (7, 1), (3, 6)],[
    (2, 4), (8, 1), (1, 7), (6, 3), (5, 9)]]):
    dist, path = tsp(cities)
    print("Test Case", i + 1, ":", dist, "|", path)
```

Output Clear

```
Test Case 1 : 16.969112047670894 | [(1, 2), (7, 1), (4, 5), (3, 6), (1,
    2)]
Test Case 2 : 23.12995011084934 | [(2, 4), (6, 3), (8, 1), (5, 9), (1,
    7), (2, 4)]

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

main.py

Share Run

```python
def is_feasible(selected, weights, capacity):
    return sum(weights[i] for i in selected) <= capacity
def knapsack(weights, values, capacity):
    best_value, best_selection = 0, []
    def search(selected):
        nonlocal best_value, best_selection
        if is_feasible(selected, weights, capacity):
            value = total_value(selected, values)
            if value > best_value:
                best_value, best_selection = value, selected[:]
        for i in range(len(weights)):
            if i not in selected:
                search(selected + [i])
    search([])
    return best_selection, best_value
cases=[([2, 3, 1], [4,5,3],4),([1, 2, 3, 4], [2, 4, 6, 3], 6)]
for i, (w, v, c) in enumerate(cases, 1):
    sel, val = knapsack(w, v, c)
    print("Test Case", i, ": Selection:", sel, "Value:", val)
```

Output

Clear

```
Test Case 1 : Selection: [1, 2] Value: 8
Test Case 2 : Selection: [0, 1, 2] Value: 12

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

main.py

Share Run

```python
from itertools import permutations
def total_cost(assign, cost_matrix):
    return sum(cost_matrix[i][assign[i]] for i in range(len
        (assign)))
def assignment_problem(cost_matrix):
    min_cost = float('inf')
    best_assignment = []
    for perm in permutations(range(len(cost_matrix))):
        cost = total_cost(perm, cost_matrix)
        if cost < min_cost:
            min_cost, best_assignment = cost, perm
    return min_cost, best_assignment
cost_matrices = [
    [[3, 10, 7], [8, 5, 12], [4, 6, 9]],
    [[15, 9, 4], [8, 7, 18], [6, 12, 11]]
]
for i, matrix in enumerate(cost_matrices):
    min_cost, assignment = assignment_problem(matrix)
    print("Test Case", i + 1, ":", "Total Cost:", min_cost, "|
        Assignment:", assignment)
```

Output Clear

```
Test Case 1 : Total Cost: 16 | Assignment: (2, 1, 0)
Test Case 2 : Total Cost: 17 | Assignment: (2, 1, 0)

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

main.py

Share Run

```python
def distance(city1, city2):
    return ((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1])
        ** 2) ** 0.5
def tsp(cities):
    from itertools import permutations
    min_dist = float('inf')
    best_path = []
    for perm in permutations(cities[1:]):
        path = [cities[0]] + list(perm) + [cities[0]]
        dist = sum(distance(path[i], path[i + 1]) for i in
            range(len(path) - 1))
        if dist < min_dist:
            min_dist = dist
            best_path = path
    return min_dist, best_path
cities = [(0, 0), (1, 2), (4, 3), (6, 1)]
min_distance, path = tsp(cities)
print("Min Distance:", min_distance)
print("Path:", path)
```

Output

Clear

```
Min Distance: 14.309535292712578
Path: [(0, 0), (1, 2), (4, 3), (6, 1), (0, 0)]

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