1

main.py | Share | Run | Output

```
import itertools
import math
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_pair, min_distance = min(((p1, p2), math.dist(p1, p2)) for p1, p2 in
    itertools.combinations(points, 2))
print(f"Closest pair: {closest_pair} Minimum distance: {min_distance}")

```

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

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

2

main.py | Share | Run | Output

```
from itertools import combinations
from math import dist

def closest_pair(points):
    min_dist = float('inf')
    closest = None
    for pair in combinations(points, 2):
        d = dist(pair[0], pair[1])
        if d < min_dist:
            min_dist = d
            closest = pair
    return closest

points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5), (7.5, 4
    .5)]
result = closest_pair(points)
print(result)
```

```
((9, 3.5), (7.5, 4.5))

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

3

main.py | Share | Run | Output

```
from itertools import combinations
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, q in combinations(points, 2):
        side = [r for r in points if orientation(p, q, r) == 1]
        if not any(orientation(p, q, r) == -1 for r in points):
            hull.extend(side)
    return list(set(hull))
points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
convex_hull_points = convex_hull(points)
print("Convex Hull:", convex_hull_points)
```

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

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

4

main.py

```python
from itertools import combinations
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, q in combinations(points, 2):
        side = [r for r in points if orientation(p, q, r) == 1]
        if not any(orientation(p, q, r) == -1 for r in points):
            hull.extend(side)
    return list(set(hull))
points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
convex_hull_points = convex_hull(points)
print("Convex Hull:", convex_hull_points)
```

Output

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

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

5

main.py

```python
from itertools import permutations
def total_cost(assignment, cost_matrix):
    return sum(cost_matrix[worker][task] for worker, task in enumerate(assignment
    ))
def assignment_problem(cost_matrix):
    num_workers = len(cost_matrix)
    min_cost = float('inf')
    optimal_assignment = None
    for perm in permutations(range(num_workers)):
        current_cost = total_cost(perm, cost_matrix)
        if current_cost < min_cost:
            min_cost = current_cost
            optimal_assignment = perm
    return min_cost, optimal_assignment
cost_matrix_simple = [[3, 10, 7], [8, 5, 12], [4, 6, 9]]
min_cost_simple, optimal_assignment_simple = assignment_problem(cost_matrix_simple)
print("Simple Case - Minimum Cost:", min_cost_simple)
print("Simple Case - Optimal Assignment:", optimal_assignment_simple)
cost_matrix_complex = [[15, 9, 4], [8, 7, 18], [6, 12, 11]]
min_cost_complex, optimal_assignment_complex = assignment_problem
    (cost_matrix_complex)
print("Complex Case - Minimum Cost:", min_cost_complex)
print("Complex Case - Optimal Assignment:", optimal_assignment_complex)
```

Output

```
Simple Case - Minimum Cost: 16
Simple Case - Optimal Assignment: (2, 1, 0)
Complex Case - Minimum Cost: 17
Complex Case - Optimal Assignment: (2, 1, 0)

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

6

main.py

```python
def total_value(items, values):
    return sum(values[i] for i in items)
def is_feasible(items, weights, capacity):
    return sum(weights[i] for i in items) <= capacity
def knapsack(items, weights, values, capacity):
    n = len(items)
    max_value = 0
    optimal_selection = []
    for i in range(1 << n):
        selected_items = [j for j in range(n) if (i & (1 << j))]
        if is_feasible(selected_items, weights, capacity):
            total = total_value(selected_items, values)
            if total > max_value:
                max_value = total
                optimal_selection = selected_items
    return optimal_selection, max_value
items1 = 3
weights1 = [2, 3, 1]
values1 = [4, 5, 3]
capacity1 = 4
items2 = 4
weights2 = [1, 2, 3, 4]
values2 = [2, 4, 6, 3]
capacity2 = 6
optimal_selection1, total_value1 = knapsack(range(items1), weights1, values1,
    capacity1)
optimal_selection2, total_value2 = knapsack(range(items2), weights2, values2,
    capacity2)
```

Output

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

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

```python
optimal_selection2, total_value2 = knapsack(range(items2), weights2, values2,
    capacity2)
print("Test Case 1:")
print("Optimal Selection:", optimal_selection1)
print("Total Value:", total_value1)
print("Test Case 2:")
print("Optimal Selection:", optimal_selection2)
print("Total Value:", total_value2)
```