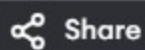




main.py



Run

Output

Clear

```
1 def closest_pair(points):
2     min_distance = float('inf')
3     closest_points = ()
4     for i in range(len(points)):
5         for j in range(i + 1, len(points)):
6             distance_squared = (points[j][0] - points[i][0])**2 +
                                (points[j][1] - points[i][1])**2
7
8             if distance_squared < min_distance:
9                 min_distance = distance_squared
10                closest_points = (points[i], points[j])
11    return closest_points, min_distance**0.5
12 points = [(1, 2), (4, 5), (7, 8), (3, 1)]
13 pair, distance = closest_pair(points)
14 print("Closest pair:", pair[0], "-", pair[1])
15 print("Minimum distance:", distance)
16
```

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

=== Code Execution Successful ===

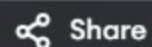


JS





main.py



Run

Output

Clear



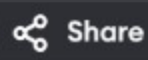
```
1 def euclidean_distance(point1, point2):
2     return ((point2[0] - point1[0]) ** 2 + (point2[1] - point1[1]
3         ) ** 2) ** 0.5
4
5 def closest_pair(points):
6     min_distance = float('inf')
7     closest_points = ()
8     for i in range(len(points)):
9         for j in range(i + 1, len(points)):
10            distance = euclidean_distance(points[i], points[j])
11            if distance < min_distance:
12                min_distance = distance
13                closest_points = (points[i], points[j])
14    return closest_points, min_distance
15
16 points = [(1, 2), (4, 5), (7, 8), (3, 1)]
17 pair, distance = closest_pair(points)
18 print("Closest pair:", pair[0], "-", pair[1])
19 print("Minimum distance:", distance)
```

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

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



main.py



Run

Output

Clear



JS



```
1 def distance(c1, c2):
2     return ((c1[0] - c2[0]) ** 2 + (c1[1] - c2[1]) ** 2) ** 0.5
3 def tsp(cities):
4     n, min_dist, best_path = len(cities), float('inf'), []
5     def visit(path, visited):
6         nonlocal min_dist, best_path
7         if len(path) == n:
8             path.append(path[0])
9             dist = sum(distance(path[i], path[i + 1]) for i in
10                        range(n))
11             if dist < min_dist: min_dist, best_path = dist,
12                                path[:]
13             path.pop()
14             return
15         for i in range(n):
16             if i not in visited:
17                 visit(path + [cities[i]], visited | {i})
18 visit([], set())
19 return min_dist, best_path
20 for i, cities in enumerate([(1, 2), (4, 5), (7, 1), (3, 6)], 1):
```

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

Output

Clear

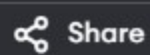
```
1 def total_value(selected, values):
2     return sum(values[i] for i in selected)
3 def is_feasible(selected, weights, capacity):
4     return sum(weights[i] for i in selected) <= capacity
5 def knapsack(weights, values, capacity):
6     best_value, best_selection = 0, []
7     def search(selected):
8         nonlocal best_value, best_selection
9         if is_feasible(selected, weights, capacity):
10             value = total_value(selected, values)
11             if value > best_value:
12                 best_value, best_selection = value, selected[:]
13             for i in range(len(weights)):
14                 if i not in selected:
15                     search(selected + [i])
16     search([])
17     return best_selection, best_value
18 cases=[([2, 3, 1], [4,5,3],4),([1, 2, 3, 4], [2, 4, 6, 3], 6)]
19 for i, (w, v, c) in enumerate(cases, 1):
20     sel, val = knapsack(w, v, c)
```

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

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



main.py



Run

Output

Clear

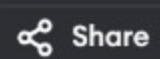
```
1 def is_left_turn(p1, p2, p3):
2     return (p2[0] - p1[0]) * (p3[1] - p1[1]) - (p3[0] - p1[0])
        * (p2[1] - p1[1]) > 0
3 def convex_hull(points):
4     leftmost = min(points)
5     hull = [leftmost]
6     while True:
7         p1 = hull[-1]
8         next_point = points[0]
9         for p2 in points:
10             if next_point == p1 or is_left_turn(p1, next_point,
                p2):
11                 next_point = p2
12             if next_point == leftmost:
13                 break
14             hull.append(next_point)
15     return hull
16 points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
17 print("Convex Hull:", convex_hull(points))
18
```

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

=== Code Execution Successful ===



main.py



Run

Output

Clear



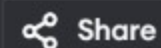
```
5 def visit(path, visited):
6     nonlocal min_dist, best_path
7     if len(path) == n:
8         path.append(path[0])
9         dist = sum(distance(path[i], path[i + 1]) for i in
10                     range(n))
11         if dist < min_dist: min_dist, best_path = dist,
12                             path[:]
13         path.pop()
14         return
15     for i in range(n):
16         if i not in visited:
17             visit(path + [cities[i]], visited | {i})
18 visit([], set())
19 return min_dist, best_path
20 for i, cities in enumerate([[(1, 2), (4, 5), (7, 1), (3, 6)],
21                             [(2, 4), (8, 1), (1, 7), (6, 3), (5, 9)]]):
22     dist, path = tsp(cities)
23     print("Test Case", i + 1, ":", dist, "|", path)
```

```
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



Run

Output

Clear

```
3 def is_feasible(selected, weights, capacity):
4     return sum(weights[i] for i in selected) <= capacity
5 def knapsack(weights, values, capacity):
6     best_value, best_selection = 0, []
7     def search(selected):
8         nonlocal best_value, best_selection
9         if is_feasible(selected, weights, capacity):
10             value = total_value(selected, values)
11             if value > best_value:
12                 best_value, best_selection = value, selected[:]
13         for i in range(len(weights)):
14             if i not in selected:
15                 search(selected + [i])
16     search([])
17     return best_selection, best_value
18 cases=[([2, 3, 1], [4,5,3],4),([1, 2, 3, 4], [2, 4, 6, 3], 6)]
19 for i, (w, v, c) in enumerate(cases, 1):
20     sel, val = knapsack(w, v, c)
21     print("Test Case", i, ": Selection:", sel, "Value:", val)
```

Test Case 1 : Selection: [1, 2] Value: 8

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

=== Code Execution Successful ===



main.py



Share

Run

Output

Clear

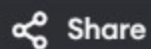
```
1 from itertools import permutations
2 def total_cost(assign, cost_matrix):
3     return sum(cost_matrix[i][assign[i]] for i in range(len
        (assign)))
4 def assignment_problem(cost_matrix):
5     min_cost = float('inf')
6     best_assignment = []
7     for perm in permutations(range(len(cost_matrix))):
8         cost = total_cost(perm, cost_matrix)
9         if cost < min_cost:
10             min_cost, best_assignment = cost, perm
11     return min_cost, best_assignment
12 cost_matrices = [
13     [[3, 10, 7], [8, 5, 12], [4, 6, 9]],
14     [[15, 9, 4], [8, 7, 18], [6, 12, 11]]
15 ]
16 for i, matrix in enumerate(cost_matrices):
17     min_cost, assignment = assignment_problem(matrix)
18     print("Test Case", i + 1, ":", "Total Cost:", min_cost, "|
        Assignment:", assignment)
```

```
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



Run

Output

Clear

```
1 def distance(city1, city2):
2     return ((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1])
3             ** 2) ** 0.5
4
5 def tsp(cities):
6     from itertools import permutations
7     min_dist = float('inf')
8     best_path = []
9     for perm in permutations(cities[1:]):
10        path = [cities[0]] + list(perm) + [cities[0]]
11        dist = sum(distance(path[i], path[i + 1]) for i in
12                    range(len(path) - 1))
13        if dist < min_dist:
14            min_dist = dist
15            best_path = path
16    return min_dist, best_path
17
18 cities = [(0, 0), (1, 2), (4, 3), (6, 1)]
19 min_distance, path = tsp(cities)
20 print("Min Distance:", min_distance)
21 print("Path:", path)
```

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

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