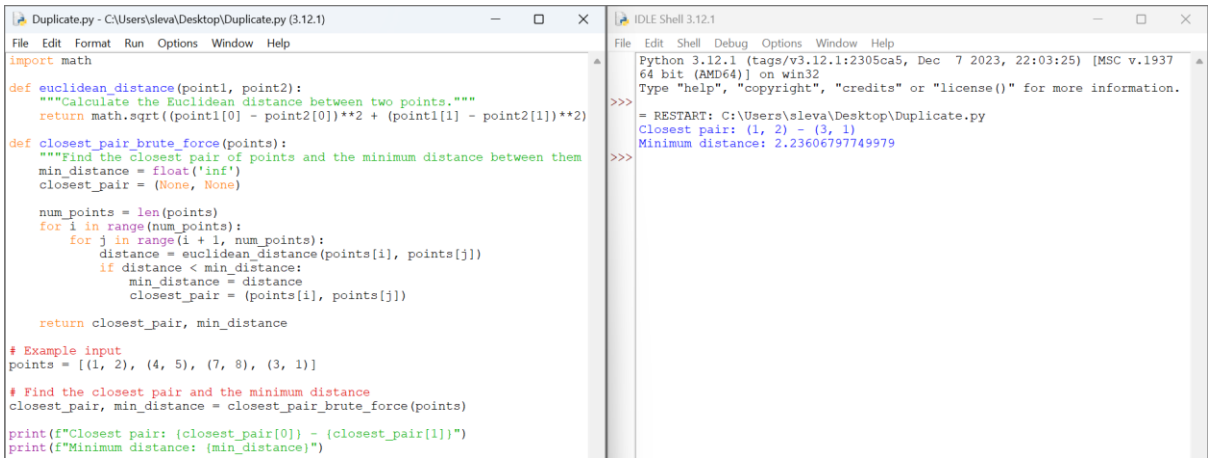
1. Closest pair of points



Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

File Edit Format Run Options Window Help

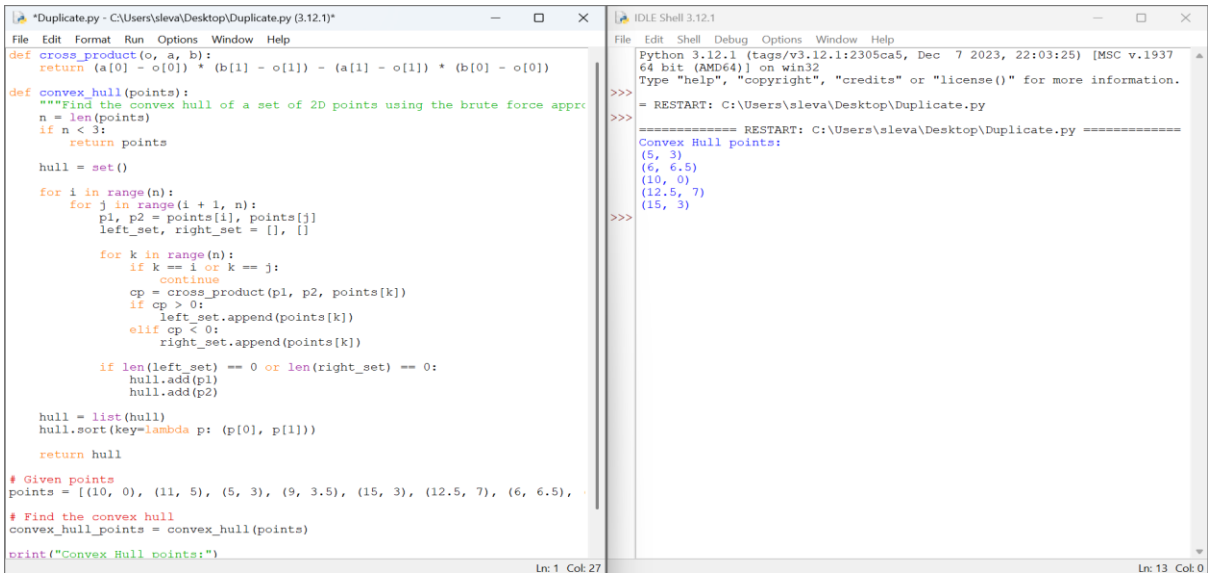
```
import math

def euclidean_distance(point1, point2):
    """Calculate the Euclidean distance between two points."""
    return math.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)

def closest_pair_brute_force(points):
    """Find the closest pair of points and the minimum distance between them
    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):
            distance = euclidean_distance(points[i], points[j])
            if distance < min_distance:
                min_distance = distance
                closest_pair = (points[i], points[j])

    return closest_pair, min_distance

# Example input
points = [(1, 2), (4, 5), (7, 8), (3, 1)]

# Find the closest pair and the minimum distance
closest_pair, min_distance = closest_pair_brute_force(points)

print(f"Closest pair: {closest_pair[0]} - {closest_pair[1]}")
print(f"Minimum distance: {min_distance}")
```

IDLE Shell 3.12.1

File Edit Shell Debug Options Window Help

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979
>>>
```

2. Convex Hull



Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

File Edit Format Run Options Window Help

```
def cross_product(o, a, b):
    return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) * (b[0] - o[0])

def convex_hull(points):
    """Find the convex hull of a set of 2D points using the brute force appro
    n = len(points)
    if n < 3:
        return points

    hull = set()

    for i in range(n):
        for j in range(i + 1, n):
            p1, p2 = points[i], points[j]
            left_set, right_set = [], []

            for k in range(n):
                if k == i or k == j:
                    continue
                cp = cross_product(p1, p2, points[k])
                if cp > 0:
                    left_set.append(points[k])
                elif cp < 0:
                    right_set.append(points[k])

            if len(left_set) == 0 or len(right_set) == 0:
                hull.add(p1)
                hull.add(p2)

    hull = list(hull)
    hull.sort(key=lambda p: (p[0], p[1]))

    return hull

# Given points
points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5),

# Find the convex hull
convex_hull_points = convex_hull(points)

print("Convex Hull points:")
```

Ln: 1 Col: 27

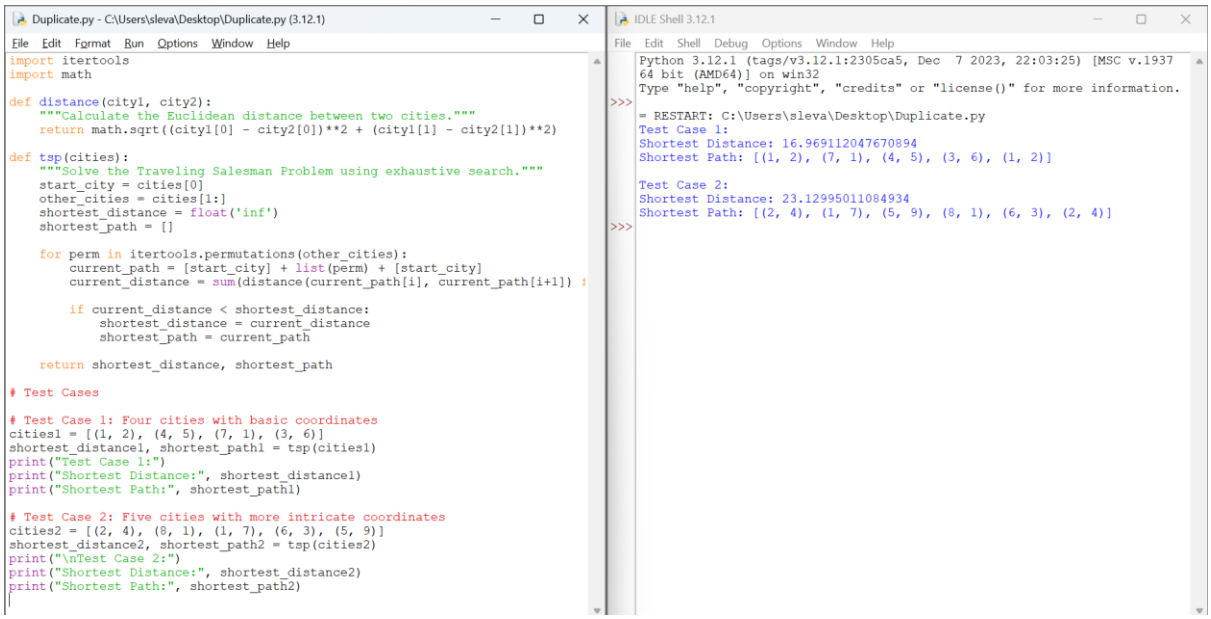
IDLE Shell 3.12.1

File Edit Shell Debug Options Window Help

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
>>>
============ RESTART: C:\Users\sleva\Desktop\Duplicate.py ============
Convex Hull points:
(5, 3)
(6, 6.5)
(10, 0)
(12.5, 7)
(15, 3)
>>>
```

Ln: 13 Col: 0

3. Travelling Salesman Problem



Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

File Edit Format Run Options Window Help

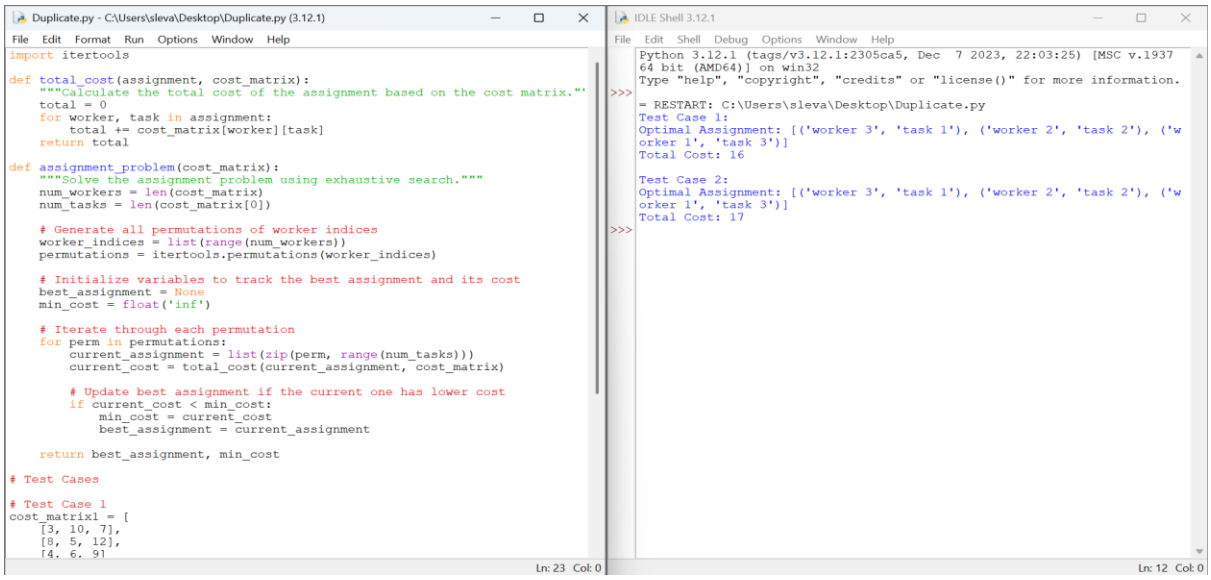
```
import itertools
import math

def distance(city1, city2):
    """Calculate the Euclidean distance between two cities."""
    return math.sqrt((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)

def tsp(cities):
    """Solve the Traveling Salesman Problem using exhaustive search."""
    start_city = cities[0]
    other_cities = cities[1:]
    shortest_distance = float('inf')
    shortest_path = []

    for perm in itertools.permutations(other_cities):
        current_path = [start_city] + list(perm) + [start_city]
        current_distance = sum(distance(current_path[i], current_path[i+1]) f

        if current_distance < shortest_distance:
            shortest_distance = current_distance
            shortest_path = current_path

    return shortest_distance, shortest_path

# Test Cases

# Test Case 1: Four cities with basic coordinates
cities1 = [(1, 2), (4, 5), (7, 1), (3, 6)]
shortest_distance1, shortest_path1 = tsp(cities1)
print("Test Case 1:")
print("Shortest Distance:", shortest_distance1)
print("Shortest Path:", shortest_path1)

# Test Case 2: Five cities with more intricate coordinates
cities2 = [(2, 4), (8, 1), (1, 7), (6, 3), (5, 9)]
shortest_distance2, shortest_path2 = tsp(cities2)
print("\nTest Case 2:")
print("Shortest Distance:", shortest_distance2)
print("Shortest Path:", shortest_path2)
```

IDLE Shell 3.12.1

File Edit Shell Debug Options Window Help

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
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), (1, 7), (5, 9), (8, 1), (6, 3), (2, 4)]
>>>
```

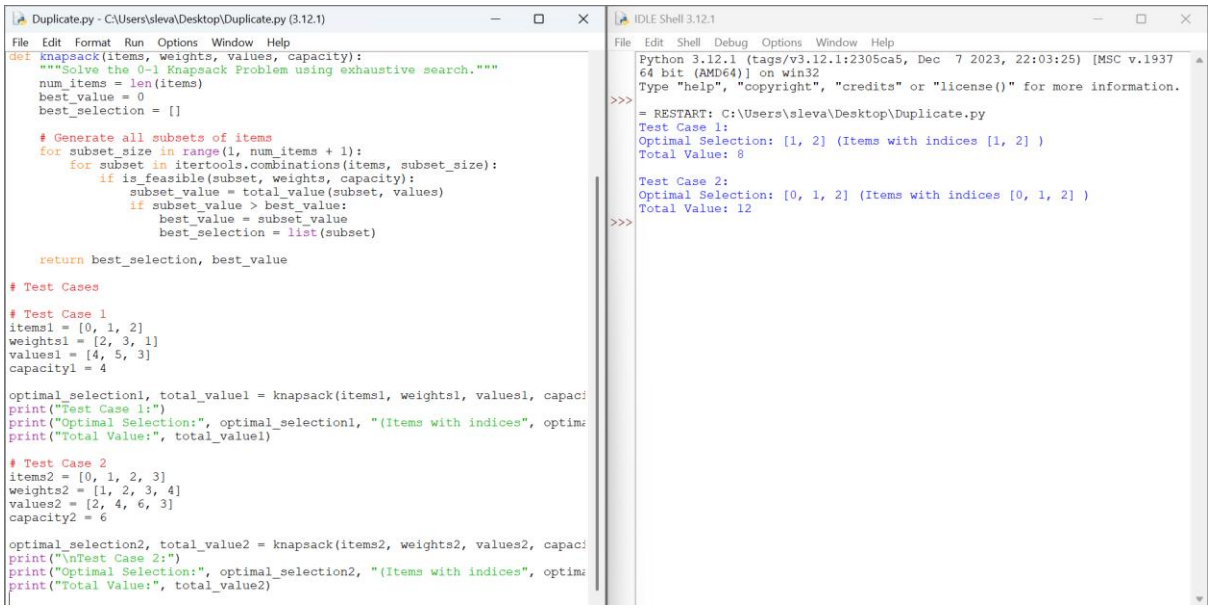
4. Assignment Problem



```
Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)
File  Edit  Format  Run  Options  Window  Help
import itertools

def total_cost(assignment, cost_matrix):
    """Calculate the total cost of the assignment based on the cost matrix."""
    total = 0
    for worker, task in assignment:
        total += cost_matrix[worker][task]
    return total

def assignment_problem(cost_matrix):
    """Solve the assignment problem using exhaustive search."""
    num_workers = len(cost_matrix)
    num_tasks = len(cost_matrix[0])

    # Generate all permutations of worker indices
    worker_indices = list(range(num_workers))
    permutations = itertools.permutations(worker_indices)

    # Initialize variables to track the best assignment and its cost
    best_assignment = None
    min_cost = float('inf')

    # Iterate through each permutation
    for perm in permutations:
        current_assignment = list(zip(perm, range(num_tasks)))
        current_cost = total_cost(current_assignment, cost_matrix)

        # Update best assignment if the current one has lower cost
        if current_cost < min_cost:
            min_cost = current_cost
            best_assignment = current_assignment

    return best_assignment, min_cost

# Test Cases

# Test Case 1
cost_matrix1 = [
    [3, 10, 7],
    [8, 5, 12],
    [4, 6, 9]
Ln: 23  Col: 0
```

```
IDLE Shell 3.12.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
Test Case 1:
Optimal Assignment: [('worker 3', 'task 1'), ('worker 2', 'task 2'), ('w
orker 1', 'task 3')]
Total Cost: 16

Test Case 2:
Optimal Assignment: [('worker 3', 'task 1'), ('worker 2', 'task 2'), ('w
orker 1', 'task 3')]
Total Cost: 17
>>>
Ln: 12  Col: 0
```

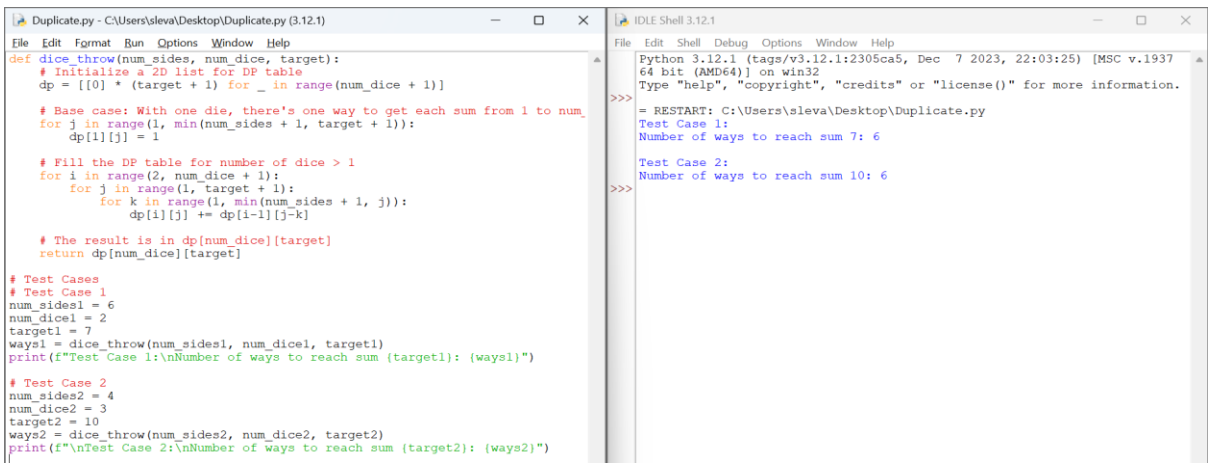
5. Knapsack Problem



```
Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)
File  Edit  Format  Run  Options  Window  Help
def knapsack(items, weights, values, capacity):
    """Solve the 0-1 Knapsack Problem using exhaustive search."""
    num_items = len(items)
    best_value = 0
    best_selection = []

    # Generate all subsets of items
    for subset_size in range(1, num_items + 1):
        for subset in itertools.combinations(items, subset_size):
            if is_feasible(subset, weights, capacity):
                subset_value = total_value(subset, values)
                if subset_value > best_value:
                    best_value = subset_value
                    best_selection = list(subset)

    return best_selection, best_value

# Test Cases

# Test Case 1
items1 = [0, 1, 2]
weights1 = [2, 3, 1]
values1 = [4, 5, 3]
capacity1 = 4

optimal_selection1, total_value1 = knapsack(items1, weights1, values1, capaci
print("Test Case 1:")
print("Optimal Selection:", optimal_selection1, "(Items with indices", optima
print("Total Value:", total_value1)

# Test Case 2
items2 = [0, 1, 2, 3]
weights2 = [1, 2, 3, 4]
values2 = [2, 4, 6, 3]
capacity2 = 6

optimal_selection2, total_value2 = knapsack(items2, weights2, values2, capaci
print("\nTest Case 2:")
print("Optimal Selection:", optimal_selection2, "(Items with indices", optima
print("Total Value:", total_value2)
```

```
IDLE Shell 3.12.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
Test Case 1:
Optimal Selection: [1, 2] (Items with indices [1, 2] )
Total Value: 8

Test Case 2:
Optimal Selection: [0, 1, 2] (Items with indices [0, 1, 2] )
Total Value: 12
>>>
```

6. Dice-throw problem



```
Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)
File  Edit  Format  Run  Options  Window  Help
def dice_throw(num_sides, num_dice, target):
    # Initialize a 2D list for DP table
    dp = [[0] * (target + 1) for _ in range(num_dice + 1)]

    # Base case: With one die, there's one way to get each sum from 1 to num_
    for j in range(1, min(num_sides + 1, target + 1)):
        dp[1][j] = 1

    # Fill the DP table for number of dice > 1
    for i in range(2, num_dice + 1):
        for j in range(1, target + 1):
            for k in range(1, min(num_sides + 1, j)):
                dp[i][j] += dp[i-1][j-k]

    # The result is in dp[num_dice][target]
    return dp[num_dice][target]

# Test Cases
# Test Case 1
num_sides1 = 6
num_dice1 = 2
target1 = 7
ways1 = dice_throw(num_sides1, num_dice1, target1)
print(f"Test Case 1:\nNumber of ways to reach sum {target1}: {ways1}")

# Test Case 2
num_sides2 = 4
num_dice2 = 3
target2 = 10
ways2 = dice_throw(num_sides2, num_dice2, target2)
print(f"\nTest Case 2:\nNumber of ways to reach sum {target2}: {ways2}")
```

```
IDLE Shell 3.12.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
Test Case 1:
Number of ways to reach sum 7: 6

Test Case 2:
Number of ways to reach sum 10: 6
>>>
```