

1. Bellman-Ford Algorithm

```
*Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)*
File Edit Format Run Options Window Help
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.edges = []

    def add_edge(self, u, v, w):
        self.edges.append((u, v, w))

    def bellman_ford(self, src):
        dist = [float('inf')] * self.V
        dist[src] = 0
        for _ in range(self.V - 1):
            for u, v, w in self.edges:
                if dist[u] != float('inf') and dist[u] + w < dist[v]:
                    dist[v] = dist[u] + w
            for u, v, w in self.edges:
                if dist[u] != float('inf') and dist[u] + w < dist[v]:
                    print("Graph contains negative weight cycle")
                    return
        self.print_solution(dist)

    def print_solution(self, dist):
        print("Vertex Distance from Source")
        for i in range(self.V):
            print(f"{i}\t\t{dist[i]}")

# Example usage
if __name__ == "__main__":
    g = Graph(5)
    g.add_edge(0, 1, -1)
    g.add_edge(0, 2, 4)
    g.add_edge(1, 2, 3)
    g.add_edge(1, 3, 2)
    g.add_edge(1, 4, 2)
    g.add_edge(3, 2, 5)
    g.add_edge(3, 1, 1)
    g.add_edge(4, 3, -3)

    # Print the solution
    g.bellman_ford(0)

IDLE Shell 3.12.1
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\sleval\Desktop\Duplicate.py
Vertex Distance from Source
0          0
1         -1
2          2
3         -2
4          1
>>>
```

2. Warshalls algorithm

```
*Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)*
File Edit Format Run Options Window Help
V = 4
INF = float('inf')
def floyd_warshall(graph):
    # dist will be the output matrix that will finally have the shortest dist
    dist = [[INF] * V for _ in range(V)]
    # Initialize the solution matrix same as input graph matrix
    for i in range(V):
        for j in range(V):
            dist[i][j] = graph[i][j]
    # Adding vertices individually
    for k in range(V):
        for i in range(V):
            for j in range(V):
                # If vertex k is on the shortest path from i to j, then update
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
    # Detect negative cycles
    for i in range(V):
        if dist[i][i] < 0:
            print("Graph contains a negative weight cycle")
            return
    print_solution(dist)

def print_solution(dist):
    print("Shortest distances between every pair of vertices:")
    for i in range(V):
        for j in range(V):
            if dist[i][j] == INF:
                print("INF", end=" ")
            else:
                print(f"{dist[i][j]:7}", end=" ")
        if j == V - 1:
            print()

# Example usage
if __name__ == "__main__":
    graph = [
        [0, 5, INF, 10],
        [INF, 0, 3, INF],
        [INF, INF, 0, 1],
        [INF, INF, INF, 0]
    ]
    floyd_warshall(graph)

IDLE Shell 3.12.1
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\sleval\Desktop\Duplicate.py
Shortest distances between every pair of vertices:
INF      0      5      8      9
INF INF      0      3      4
INF INF INF      0
>>>
```

3. Coin change problem

```
*Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)*
File Edit Format Run Options Window Help
def min_coins(coins, amount):
    # Initialize dp array with infinity for all values except 0
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0

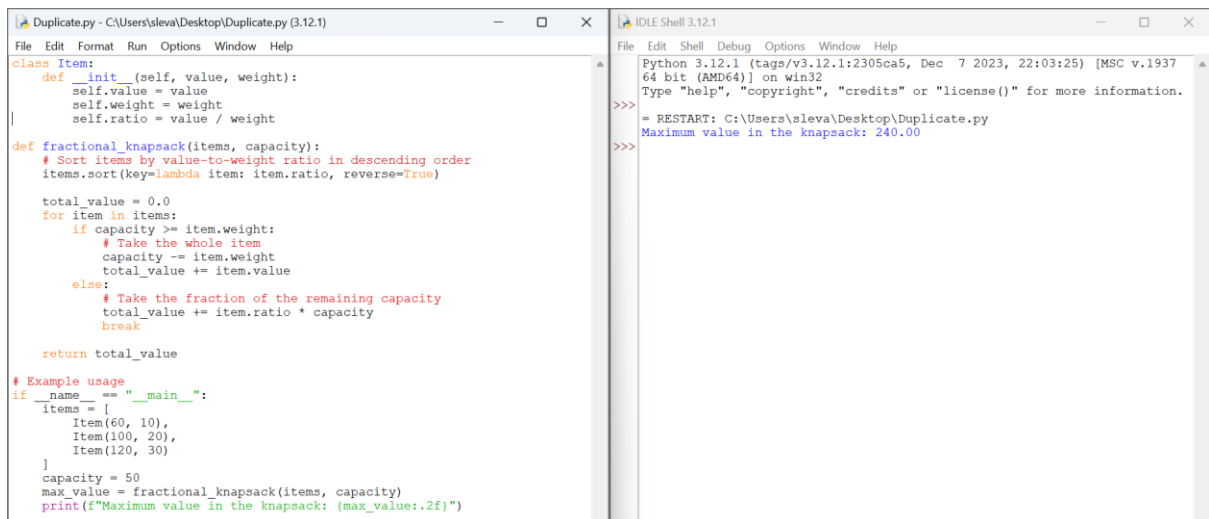
    # Update dp array with minimum coins for each amount
    for i in range(1, amount + 1):
        for coin in coins:
            if i - coin >= 0:
                dp[i] = min(dp[i], dp[i - coin] + 1)

    return dp[amount] if dp[amount] != float('inf') else -1

# Example usage
coins = [1, 2, 5]
amount = 11
print("Coins :", coins[0], coins[1], coins[2])
print("Amount :", amount)
print("Minimum number of coins required:", min_coins(coins, amount))

IDLE Shell 3.12.1
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\sleval\Desktop\Duplicate.py
Minimum number of coins required: 3
>>>
===== RESTART: C:\Users\sleval\Desktop\Duplicate.py =====
Coins : 1 2 5
Amount : 11
Minimum number of coins required: 3
>>>
```

4. Knapsack problem using greedy



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

class Item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight
        self.ratio = value / weight

def fractional_knapsack(items, capacity):
    # Sort items by value-to-weight ratio in descending order
    items.sort(key=lambda item: item.ratio, reverse=True)

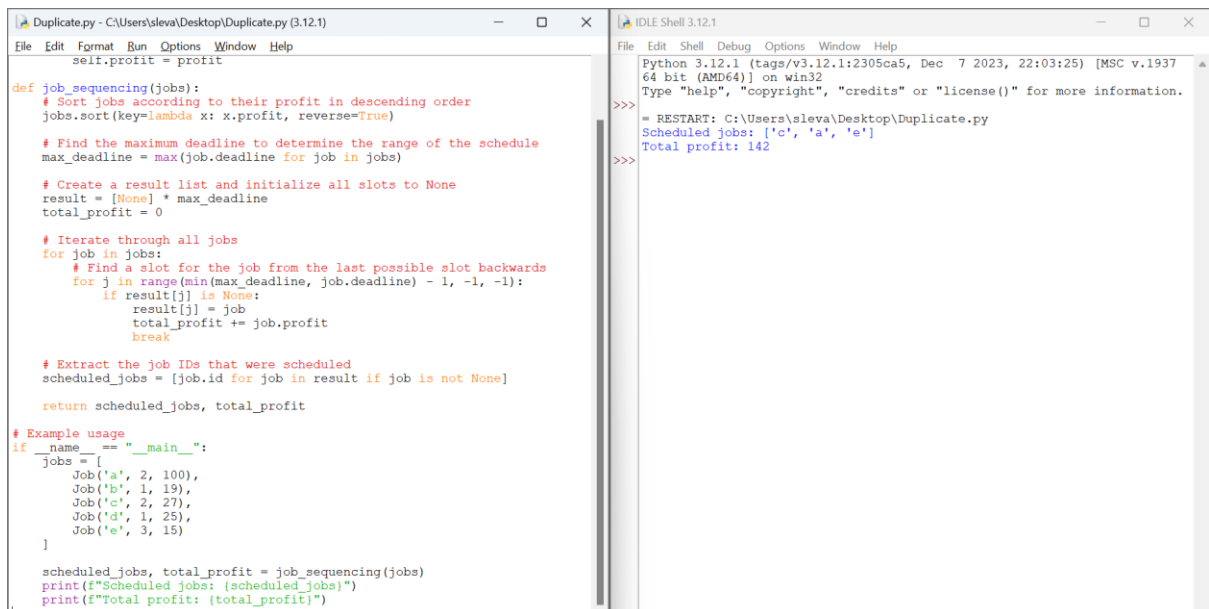
    total_value = 0.0
    for item in items:
        if capacity >= item.weight:
            # Take the whole item
            capacity -= item.weight
            total_value += item.value
        else:
            # Take the fraction of the remaining capacity
            total_value += item.ratio * capacity
            break

    return total_value

# Example usage
if __name__ == "__main__":
    items = [
        Item(60, 10),
        Item(100, 20),
        Item(120, 30)
    ]
    capacity = 50
    max_value = fractional_knapsack(items, capacity)
    print(f'Maximum value in the knapsack: {max_value:.2f}')

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\sleval\Desktop\Duplicate.py
Maximum value in the knapsack: 240.00
>>>
```

5. Job sequence with deadlines



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

self.profit = profit

def job_sequencing(jobs):
    # Sort jobs according to their profit in descending order
    jobs.sort(key=lambda x: x.profit, reverse=True)

    # Find the maximum deadline to determine the range of the schedule
    max_deadline = max(job.deadline for job in jobs)

    # Create a result list and initialize all slots to None
    result = [None] * max_deadline
    total_profit = 0

    # Iterate through all jobs
    for job in jobs:
        # Find a slot for the job from the last possible slot backwards
        for j in range(min(max_deadline, job.deadline) - 1, -1, -1):
            if result[j] is None:
                result[j] = job
                total_profit += job.profit
                break

    # Extract the job IDs that were scheduled
    scheduled_jobs = [job.id for job in result if job is not None]

    return scheduled_jobs, total_profit

# Example usage
if __name__ == "__main__":
    jobs = [
        Job('a', 2, 100),
        Job('b', 1, 19),
        Job('c', 2, 27),
        Job('d', 1, 25),
        Job('e', 3, 15)
    ]

    scheduled_jobs, total_profit = job_sequencing(jobs)
    print(f'Scheduled jobs: {scheduled_jobs}')
    print(f'Total profit: {total_profit}')

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\sleval\Desktop\Duplicate.py
Scheduled jobs: ['c', 'a', 'e']
Total profit: 142
>>>
```