

1. Dijkstra's Algorithm

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

def dijkstra(graph, source):
    V = len(graph)
    pq = []
    heapq.heappush(pq, (0, source)) # (distance, vertex)
    dist = [float('inf')] * V
    dist[source] = 0
    while pq:
        # Extract the vertex with minimum distance from pq
        current_dist, u = heapq.heappop(pq)

        # Visit each neighbor of u
        for neighbor, weight in graph[u]:
            new_dist = current_dist + weight

            # If a shorter path to v is found
            if new_dist < dist[v]:
                dist[v] = new_dist
                heapq.heappush(pq, (new_dist, v))

    return dist

# Example usage
if __name__ == "__main__":
    # Graph represented as an adjacency list
    # Each element is a list of tuples (neighbor, weight)
    graph = {
        0: [(1, 4), (2, 1)],
        1: [(3, 1)],
        2: [(1, 2), (3, 5)],
        3: []
    }

    source = 0
    distances = dijkstra(graph, source)
    print(f"Shortest distances from vertex {source}: {distances}")

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 from vertex 0: [0, 3, 1, 4]
>>>
```

2. Huffman Algorithm

```
*Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)*
File Edit Format Run Options Window Help
import heapq
from collections import defaultdict, Counter

class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    # Define comparison operators for priority queue
    def __lt__(self, other):
        return self.freq < other.freq

def build_huffman_tree(frequency):
    # Create a priority queue (min-heap)
    heap = [Node(char, freq) for char, freq in frequency.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        # Extract the two nodes with the smallest frequency
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)

        # Create a new internal node with these two nodes as children
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right

        # Add the new node to the heap
        heapq.heappush(heap, merged)

    # The remaining node is the root of the Huffman tree
    return heap[0]

def build_codes(node, prefix="", codebook=None):
    if codebook is None:
        codebook = {}

    if node.char is not None:
        # Leaf node, store the code
        codebook[node.char] = prefix

    if node.left:
        build_codes(node.left, prefix + "0", codebook)

    if node.right:
        build_codes(node.right, prefix + "1", codebook)

    return codebook

# Example usage
if __name__ == "__main__":
    # Sample text
    text = "this is an example for huffman encoding"

    # Calculate character frequencies
    frequency = Counter(text)

    # Build Huffman tree
    huffman_tree = build_huffman_tree(frequency)

    # Build codebook
    codebook = build_codes(huffman_tree)

    # Encode the text
    encoded_data = ""
    for char in text:
        encoded_data += codebook[char]

    # Decode the data
    decoded_data = ""
    i = 0
    while i < len(encoded_data):
        # Find the character corresponding to the current bit sequence
        for char, code in codebook.items():
            if encoded_data[i:i+len(code)] == code:
                decoded_data += char
                i += len(code)
                break

    print("Encoded data:", encoded_data)
    print("Decoded data:", decoded_data)
```

3. Container loading

```
*Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)*
File Edit Format Run Options Window Help
class Item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight
        self.ratio = value / weight

def container_loading(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
    total_weight = 0.0
    loaded_items = []

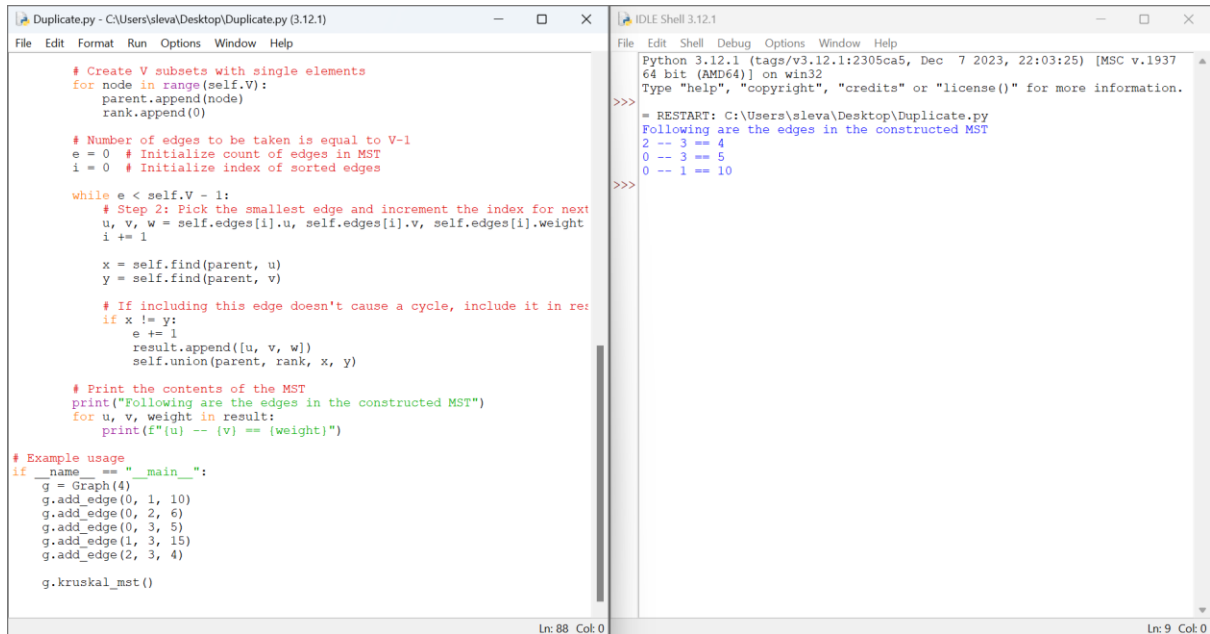
    for item in items:
        if total_weight + item.weight <= capacity:
            # Take the whole item
            total_weight += item.weight
            total_value += item.value
            loaded_items.append(item)
        else:
            # Take the fraction of the remaining capacity
            remaining_capacity = capacity - total_weight
            total_value += item.ratio * remaining_capacity
            total_weight += remaining_capacity
            loaded_items.append(Item(item.value * remaining_capacity / item.value, remaining_capacity))
            break

    return total_value, loaded_items

# Example usage
if __name__ == "__main__":
    items = [
        Item(60, 10),
        Item(100, 20),
        Item(120, 30)
    ]
    capacity = 50
    max_value, loaded_items = container_loading(items, capacity)
    print(f"Maximum value in the container: {max_value:.2f}")
    print("Items loaded:")
    for item in loaded_items:
        print(f"Value: {item.value}, Weight: {item.weight}")

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 container: 240.00
Items loaded:
Value: 60, Weight: 10
Value: 100, Weight: 20
Value: 80.0, Weight: 20.0
>>>
```

4. Minimum spanning tree Kruskals algorithm



The screenshot shows a Python IDE with two windows. The left window displays the code for the Kruskal's algorithm, and the right window shows the output of the program.

```
# Create V subsets with single elements
for node in range(self.V):
    parent.append(node)
    rank.append(0)

# Number of edges to be taken is equal to V-1
e = 0 # Initialize count of edges in MST
i = 0 # Initialize index of sorted edges

while e < self.V - 1:
    # Step 2: Pick the smallest edge and increment the index for next
    u, v, w = self.edges[i].u, self.edges[i].v, self.edges[i].weight
    i += 1

    x = self.find(parent, u)
    y = self.find(parent, v)

    # If including this edge doesn't cause a cycle, include it in result
    if x != y:
        e += 1
        result.append([u, v, w])
        self.union(parent, rank, x, y)

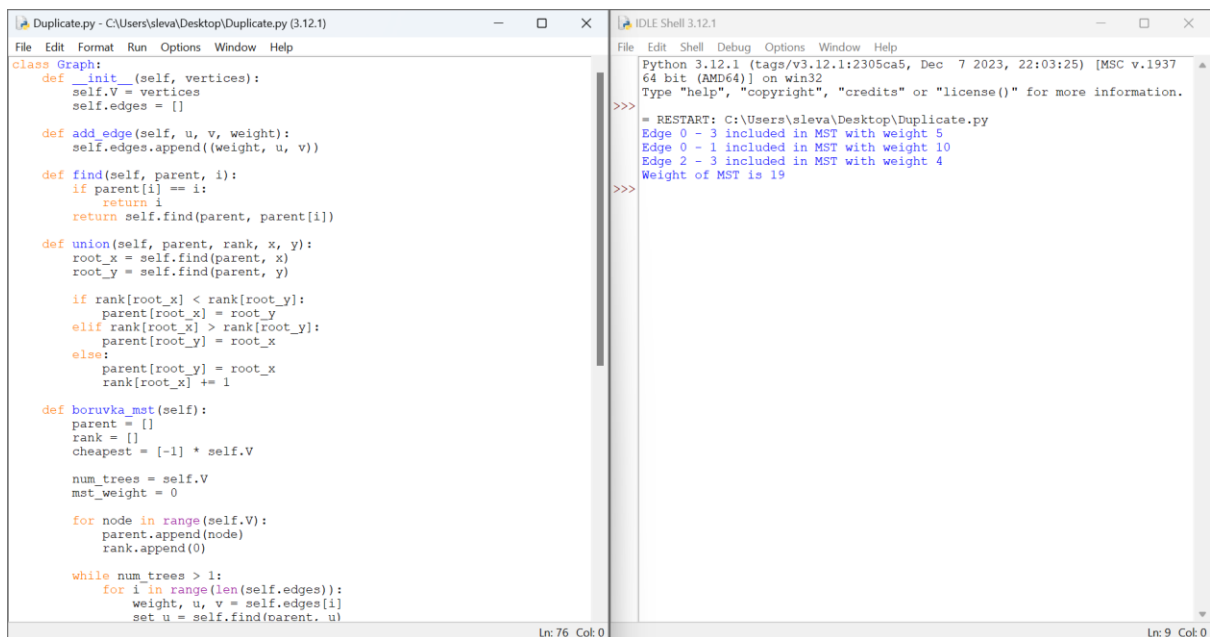
# Print the contents of the MST
print("Following are the edges in the constructed MST")
for u, v, weight in result:
    print(f"{u} -- {v} == {weight}")

# Example usage
if __name__ == "__main__":
    g = Graph(4)
    g.add_edge(0, 1, 10)
    g.add_edge(0, 2, 6)
    g.add_edge(0, 3, 5)
    g.add_edge(1, 3, 15)
    g.add_edge(2, 3, 4)

    g.kruskal_mst()
```

```
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
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
>>>
```

5. Boruvkas algorithm



The screenshot shows a Python IDE with two windows. The left window displays the code for the Boruvka's algorithm, and the right window shows the output of the program.

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.edges = []

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

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def union(self, parent, rank, x, y):
        root_x = self.find(parent, x)
        root_y = self.find(parent, y)

        if rank[root_x] < rank[root_y]:
            parent[root_x] = root_y
        elif rank[root_x] > rank[root_y]:
            parent[root_y] = root_x
        else:
            parent[root_y] = root_x
            rank[root_x] += 1

    def boruvka_mst(self):
        parent = []
        rank = []
        cheapest = [-1] * self.V

        num_trees = self.V
        mst_weight = 0

        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        while num_trees > 1:
            for i in range(len(self.edges)):
                weight, u, v = self.edges[i]
                set_u = self.find(parent, u)
```

```
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
Edge 0 - 3 included in MST with weight 5
Edge 0 - 1 included in MST with weight 10
Edge 2 - 3 included in MST with weight 4
Weight of MST is 19
>>>
```