1. There are 3n piles of coins of varying size, you and your friends will take piles of coins as

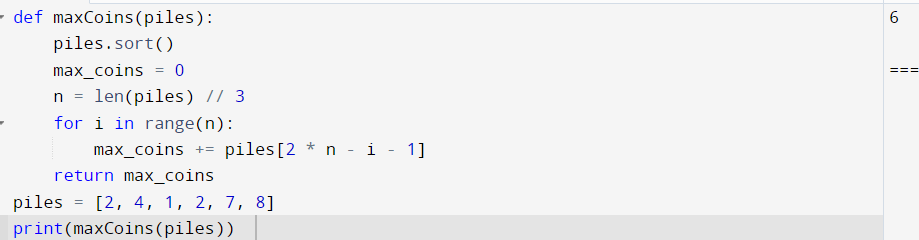
follows: In each step, you will choose any 3 piles of coins (not necessarily consecutive). Of

your choice, Alice will pick the pile with the maximum number of coins. You will pick the

next pile with the maximum number of coins. Your friend Bob will pick the last pile. Repeat

until there are no more piles of coins. Given an array of integers piles where piles[i] is the

number of coins in the ith pile. Return the maximum number of coins that you can have.



2. You are given a 0-indexed integer array coins, representing the values of the coins available,

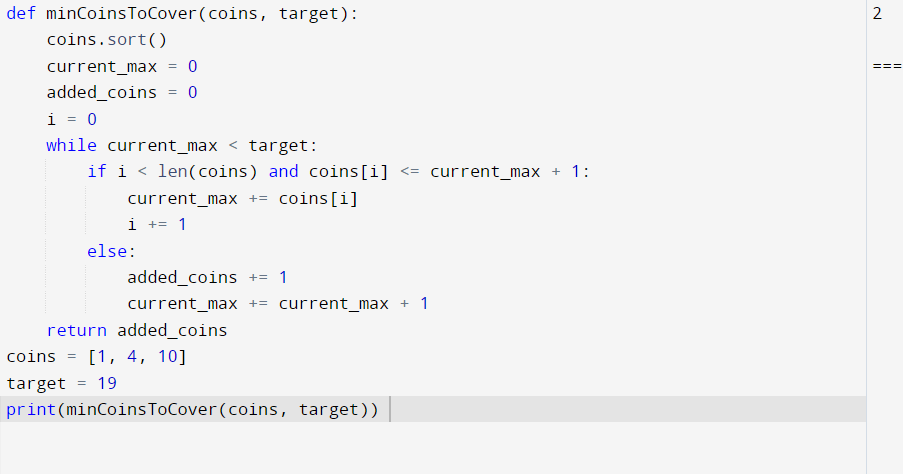
and an integer target. An integer x is obtainable if there exists a subsequence of coins that

sums to x. Return the minimum number of coins of any value that need to be added to the

array so that every integer in the range [1, target] is obtainable. A subsequence of an array is

a new non-empty array that is formed from the original array by deleting some (possibly

none) of the elements without disturbing the relative positions of the remaining elements.



3. You are given an integer array jobs, where jobs[i] is the amount of time it takes to complete

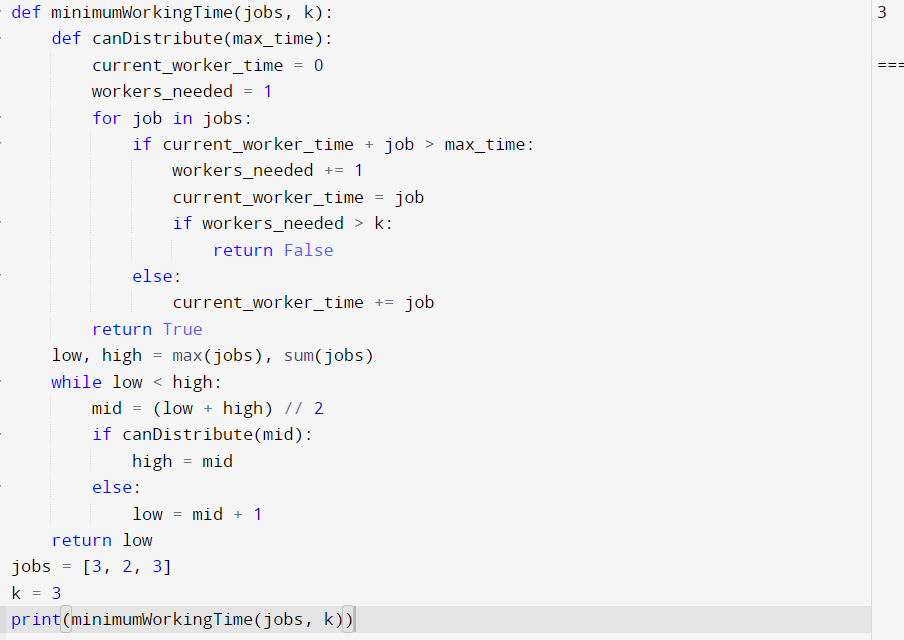
the ith job. There are k workers that you can assign jobs to. Each job should be assigned to

exactly one worker. The working time of a worker is the sum of the time it takes to complete

all jobs assigned to them. Your goal is to devise an optimal assignment such that the

maximum working time of any worker is minimized. Return the minimum possible

maximum working time of any assignment.



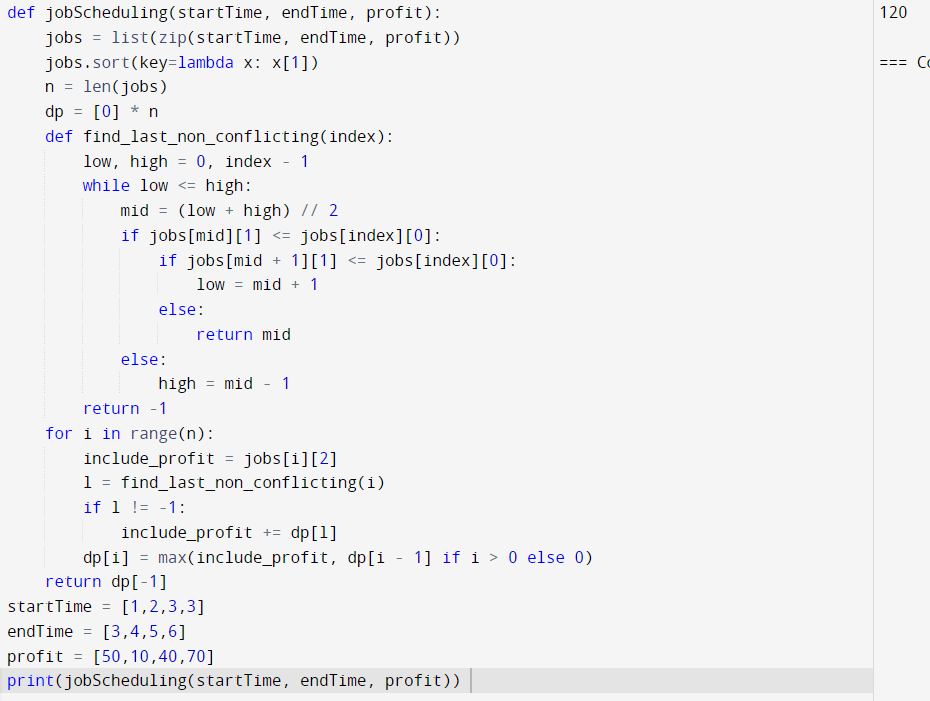
4. We have n jobs, where every job is scheduled to be done from startTime[i] to endTime[i],

obtaining a profit of profit[i]. You're given the startTime, endTime and profit arrays, return

the maximum profit you can take such that there are no two jobs in the subset with

overlapping time range. If you choose a job that ends at time X you will be able to start

another job that starts at time X.



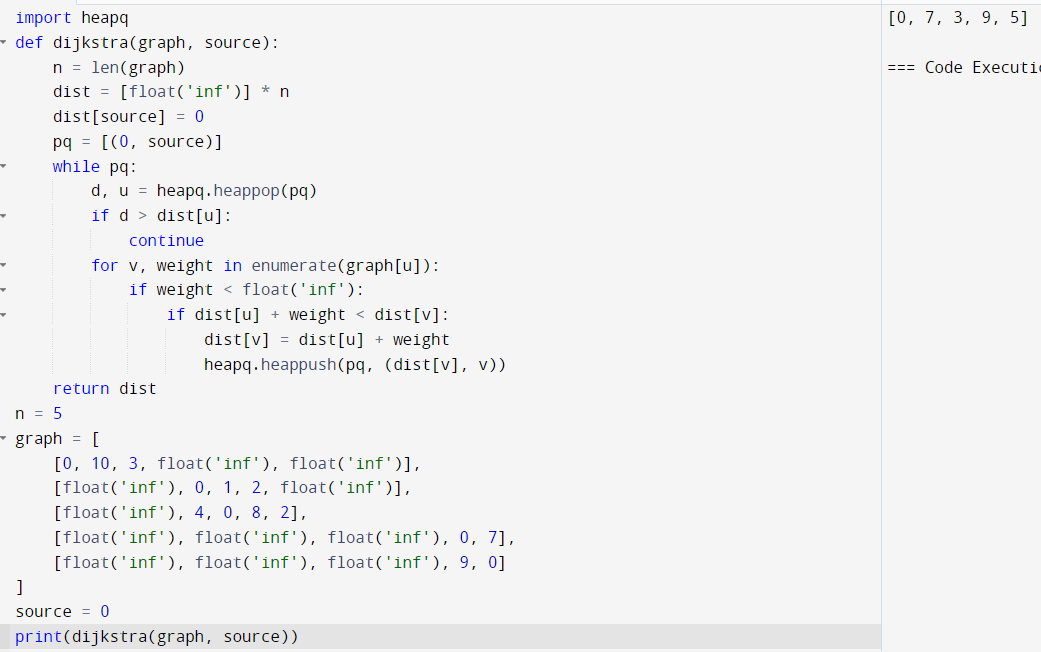
5. Given a graph represented by an adjacency matrix, implement Dijkstra's Algorithm to

find the shortest path from a given source vertex to all other vertices in the graph. The

graph is represented as an adjacency matrix where graph[i][j] denote the weight of the

edge from vertex i to vertex j. If there is no edge between vertices i and j, the value is

Infinity (or a very large number).

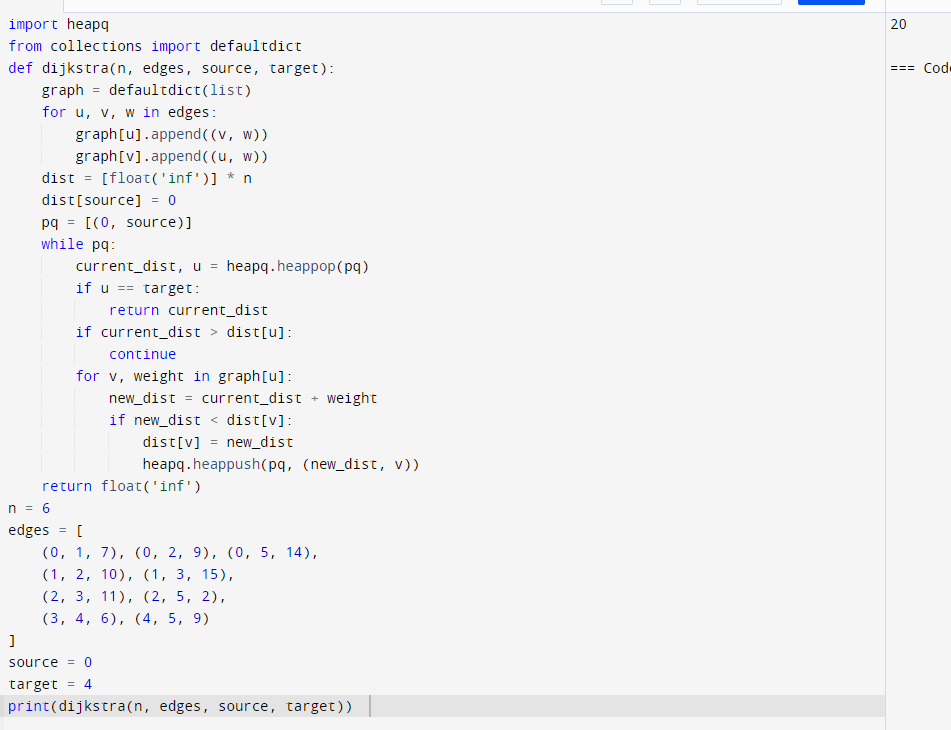


6. Given a graph represented by an edge list, implement Dijkstra's Algorithm to find the

shortest path from a given source vertex to a target vertex. The graph is represented as a

list of edges where each edge is a tuple (u, v, w) representing an edge from vertex u to

vertex v with weight w.



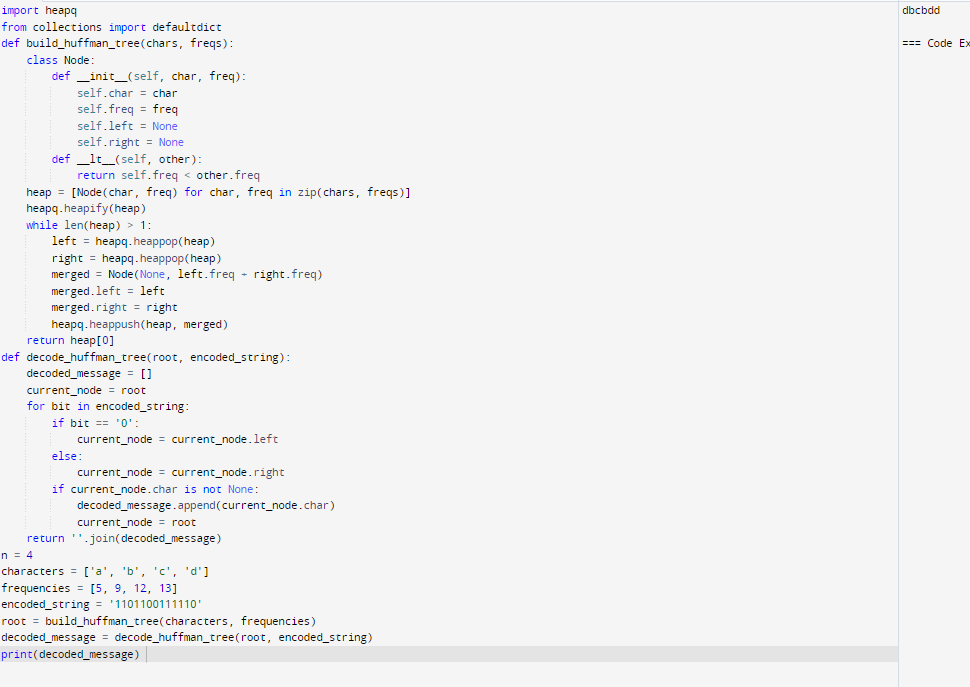
7. Given a set of characters and their corresponding frequencies, construct the Huffman

Tree and generate the Huffman Codes for each character.



8. Given a Huffman Tree and a Huffman encoded string, decode the string to get the

original message.

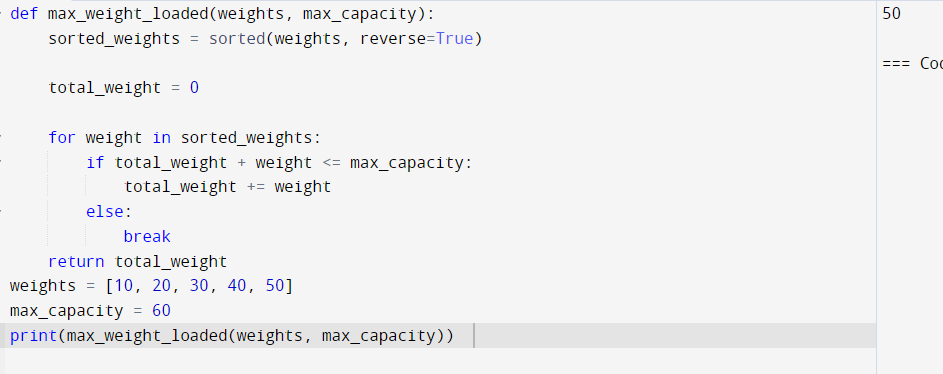


9. Given a list of item weights and the maximum capacity of a container, determine the

maximum weight that can be loaded into the container using a greedy approach. The

greedy approach should prioritize loading heavier items first until the container reaches

its capacity.

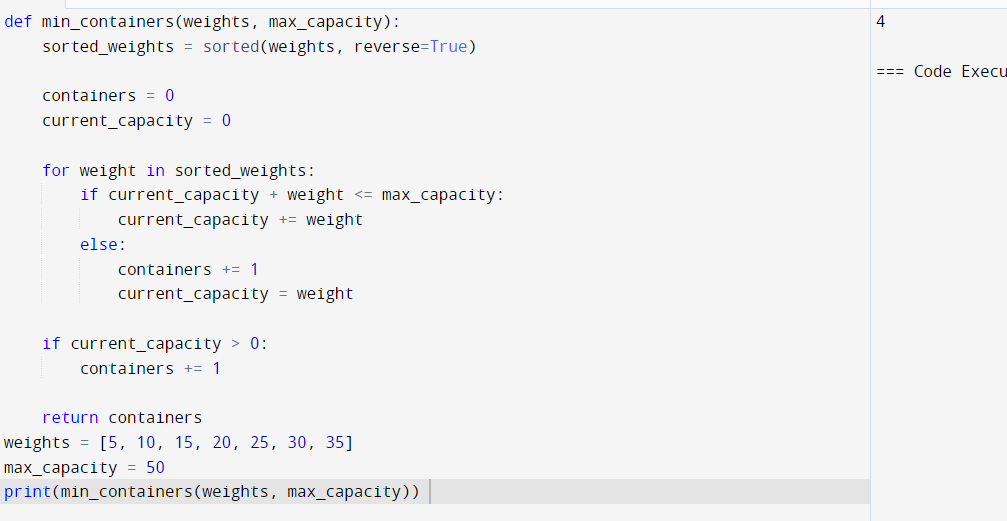


10. Given a list of item weights and a maximum capacity for each container, determine the

minimum number of containers required to load all items using a greedy approach. The

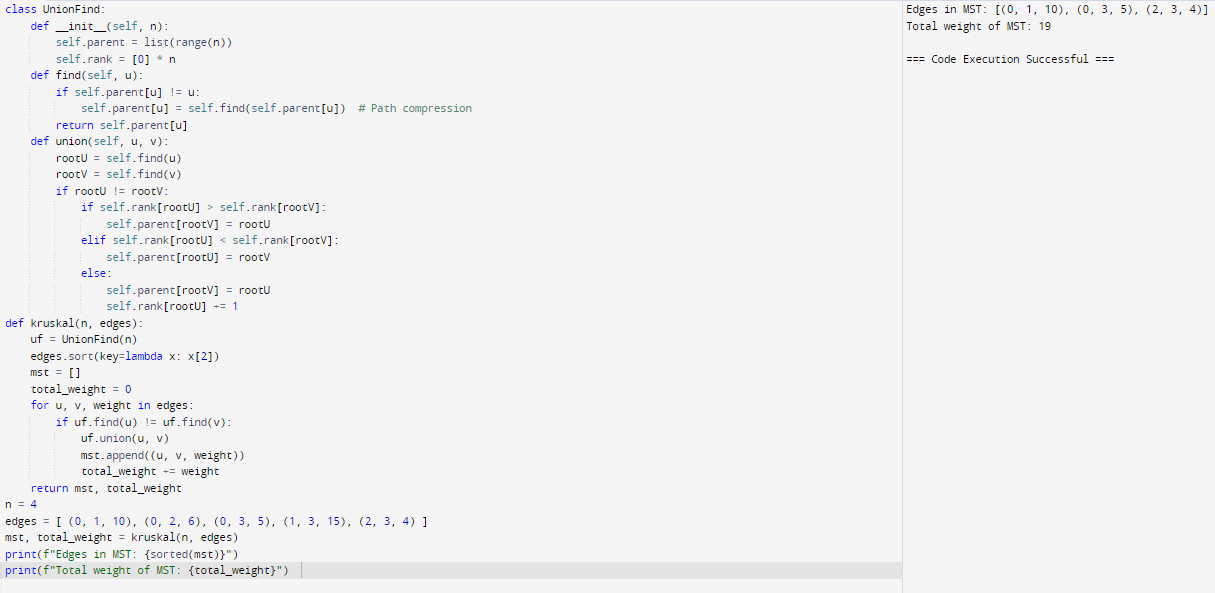
greedy approach should prioritize loading items into the current container until it is full

before moving to the next container.



11. Given a graph represented by an edge list, implement Kruskal's Algorithm to find the

Minimum Spanning Tree (MST) and its total weight.



12. Given a graph with weights and a potential Minimum Spanning Tree (MST), verify if the

given MST is unique. If it is not unique, provide another possible MST.

