rogramiz

Python Online Compiler



main.py

Share Run

```python
def maxCoins(piles):
    piles.sort()
    total_coins = 0
    n = len(piles) // 3
    for i in range(n):
        total_coins += piles[-(2 + i)]
    return total_coins

# Example usage
print(maxCoins([2, 4, 1, 2, 7, 8]))
print(maxCoins([2, 4, 5]))
```

Output

```
11
4

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

https://www.programiz.com/python-programming/online-compiler/

Programiz
Python Online Compiler



main.py

Share Run

```python
def min_coins_to_add(coins, target):
    coins.sort()
    current_sum = 0
    count = 0
    for coin in coins:
        while current_sum + 1 < coin and current_sum < target:
            count += 1
            current_sum += current_sum + 1
        current_sum += coin
    while current_sum < target:
        count += 1
        current_sum += current_sum + 1
    return count

# Example usage
coins = [1, 4, 10]
target = 19
print(min_coins_to_add(coins, target))  # Output: 2

```

Output Clear

```
2

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

```python
# Minimum Maximum Working Time of Workers

def minimumTimeRequired(jobs, k):
    def canDistribute(maxTime):
        workers = [0] * k
        return backtrack(0, workers, maxTime)

    def backtrack(i, workers, maxTime):
        if i == len(jobs):
            return True
        for j in range(k):
            if workers[j] + jobs[i] <= maxTime:
                workers[j] += jobs[i]
                if backtrack(i + 1, workers, maxTime):
                    return True
                workers[j] -= jobs[i]
            if workers[j] == 0:  # No need to try further if this worker is still idle
                break
        return False

    left, right = max(jobs), sum(jobs)
    while left < right:
        mid = (left + right) // 2
        if canDistribute(mid):
            right = mid
        else:
            left = mid
    return left

# Example usage
jobs = [3, 2, 3]
k = 3
print(minimumTimeRequired(jobs, k))  # Output: 3
```

```
3

=== Code Execution S
```

```python
def jobScheduling(startTime, endTime, profit):
    jobs = sorted(zip(startTime, endTime, profit), key=lambda x: x[1])
    dp = [0] * (len(jobs) + 1)

    def binarySearch(index):
        low, high = 0, index - 1
        while low <= high:
            mid = (low + high) // 2
            if jobs[mid][1] <= jobs[index][0]:
                low = mid + 1
            else:
                high = mid - 1
        return high

    for i in range(1, len(jobs) + 1):
        incl_profit = jobs[i - 1][2]
        l = binarySearch(i - 1)
        if l != -1:
            incl_profit += dp[l + 1]
        dp[i] = max(incl_profit, dp[i - 1])

    return dp[-1]

# Example usage
startTime = [1, 2, 3, 3]
endTime = [3, 4, 5, 6]
profit = [50, 10, 40, 70]
print(jobScheduling(startTime, endTime, profit))
```

Output

```
120

=== Code Execution
```

om/online_python_compiler

Run | Debug | Stop | Share | Save | {} Beautify

main.py

```python
import sys

def dijkstra(graph, source):
    n = len(graph)
    distances = [sys.maxsize] * n
    distances[source] = 0
    visited = [False] * n

    for _ in range(n):
        min_distance = sys.maxsize
        min_index = -1

        for v in range(n):
            if not visited[v] and distances[v] < min_distance:
                min_distance = distances[v]
                min_index = v

        visited[min_index] = True

        for v in range(n):
            if (graph[min_index][v] > 0 and not visited[v] and
                    distances[min_index] + graph[min_index][v] < distances[v]):
                distances[v] = distances[min_index] + graph[min_index][v]

    return distances

# Test Case 1
n = 5
graph = [[0, 10, 3, float('inf'), float('inf')],
         [float('inf'), 0, 1, 2, float('inf')],
         [float('inf'), 4, 0, 8, 2],
         [float('inf'), float('inf'), float('inf'), 0, 7],
         [float('inf'), float('inf'), float('inf'), 9, 0]]
source = 0

output = dijkstra(graph, source)
print(output)
```

input

```
[0, 7, 3, 9, 5]


...Program finished with exit code 0
Press ENTER to exit console.
```

main.py

Share Run

```python
import heapq

def dijkstra(n, edges, source, target):
    graph = {i: [] for i in range(n)}
    for u, v, w in edges:
        graph[u].append((v, w))
        graph[v].append((u, w))

    min_heap = [(0, source)]
    distances = {i: float('inf') for i in range(n)}
    distances[source] = 0

    while min_heap:
        current_distance, current_vertex = heapq.heappop(min_heap)

        if current_vertex == target:
            return current_distance

        if current_distance > distances[current_vertex]:
            continue

        for neighbor, weight in graph[current_vertex]:
            distance = current_distance + weight

            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(min_heap, (distance, neighbor))

    return distances[target]

# Test Case
n = 6
edges = [(0, 1, 7), (0, 2, 9), (0, 5, 14), (1, 2, 10), (1, 3, 15),
         (2, 3, 11), (2, 5, 2), (3, 4, 6), (4, 5, 9)]
source = 0
target = 4
print(dijkstra(n, edges, source, target))
```

Output

```
20

=== Code Exec
```

Python Online Compiler

Programiz PRO >

main.py

Share Run

```python
import heapq
from collections import defaultdict

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

    def __lt__(self, other):
        return self.freq < other.freq
def huffman_codes(characters, frequencies):
    heap = [Node(char, freq) for char, freq in zip(characters, frequencies)]
    heapq.heapify(heap)
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)
    codes = {}
    def generate_codes(node, current_code):
        if node:
            if node.char is not None:
                codes[node.char] = current_code
            generate_codes(node.left, current_code + '0')
            generate_codes(node.right, current_code + '1')
    generate_codes(heap[0], '')
    return [(char, codes[char]) for char in characters]

# Test Case
n = 4
characters = ['a', 'b', 'c', 'd']
frequencies = [5, 9, 12, 13]
output = huffman_codes(characters, frequencies)
print(output)
```

Output

Clear

```
[('a', '00'), ('b', '01'), ('c', '10'), ('d', '11')]

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

29°C Partly cloudy

Search

ENG IN

19:15 19-10-2024

main.py

Share Run

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

def build_huffman_tree(characters, frequencies):
    nodes = [Node(char, freq) for char, freq in zip(characters, frequencies)]
    while len(nodes) > 1:
        nodes = sorted(nodes, key=lambda x: x.freq)
        left = nodes[0]
        right = nodes[1]
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        nodes = nodes[2:] + [merged]
    return nodes[0]

def decode_huffman_tree(root, encoded_string):
    decoded_string = []
    current_node = root
    for bit in encoded_string:
        current_node = current_node.left if bit == '0' else current_node.right
        if current_node.char:
            decoded_string.append(current_node.char)
            current_node = root
    return ''.join(decoded_string)

n = 4
characters = ['a', 'b', 'c', 'd']
frequencies = [5, 9, 12, 13]
encoded_string = '1101100111110'

huffman_tree = build_huffman_tree(characters, frequencies)
output = decode_huffman_tree(huffman_tree, encoded_string)
print(output)  # Output: "abacd"
```

Output

```
[('a', '00'), ('b', '01'), ('c', '10'), ('d', '11')]

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

Python Online Compiler



main.py

Share Run

```python
def max_weight(weights, max_capacity):
    weights.sort(reverse=True)
    total_weight = 0

    for weight in weights:
        if total_weight + weight <= max_capacity:
            total_weight += weight

    return total_weight

# Test Case 1
n1 = 5
weights1 = [10, 20, 30, 40, 50]
max_capacity1 = 60
output1 = max_weight(weights1, max_capacity1)
print(output1)  # Output: 50
```

Output

```
60

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

main.py

Share Run Output

```python
def min_containers(weights, max_capacity):
    weights.sort(reverse=True)
    containers = 0
    current_capacity = 0

    for weight in weights:
        if current_capacity + weight > max_capacity:
            containers += 1
            current_capacity = weight
        else:
            current_capacity += weight

    if current_capacity > 0:
        containers += 1

    return containers

# Test Case 1
n = 7
weights = [5, 10, 15, 20, 25, 30, 35]
max_capacity = 50
result = min_containers(weights, max_capacity)
print(result)
```

4

=== Code

```python
class DisjointSet:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n
    def find(self, u):
        if self.parent[u] != u:
            self.parent[u] = self.find(self.parent[u])
        return self.parent[u]
    def union(self, u, v):
        root_u = self.find(u)
        root_v = self.find(v)
        if root_u != root_v:
            if self.rank[root_u] > self.rank[root_v]:
                self.parent[root_v] = root_u
            elif self.rank[root_u] < self.rank[root_v]:
                self.parent[root_u] = root_v
            else:
                self.parent[root_v] = root_u
                self.rank[root_u] += 1

def kruskal(n, edges):
    edges.sort(key=lambda x: x[2])
    ds = DisjointSet(n)
    mst = []
    total_weight = 0

    for u, v, weight in edges:
        if ds.find(u) != ds.find(v):
            ds.union(u, v)
            mst.append((u, v, weight))
            total_weight += weight

    return mst, total_weight
n = 4
edges = [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
mst, total_weight = kruskal(n, edges)
print("Edges in MST:", mst)
print("Total weight of MST:", total_weight)
```

```
4

=== Code Exec
```

main.py

Share

Run

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    def add_edge(self, u, v, w):
        self.graph.append((w, u, v))
    def find_parent(self, parent, i):
        if parent[i] == i:
            return i
        return self.find_parent(parent, parent[i])
    def union(self, parent, rank, x, y):
        xroot = self.find_parent(parent, x)
        yroot = self.find_parent(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1

    def is_unique_mst(self, given_mst):
        parent = []
        rank = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
        mst_weight = 0
        for u, v, w in given_mst:
            mst_weight += w
            self.union(parent, rank, u, v)
        return mst_weight
n = 4
edges = [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
given_mst = [(2, 3, 4), (0, 3, 5), (0, 1, 10)]
g = Graph(n)
for edge in edges:
    g.add_edge(*edge)
unique = g.is_unique_mst(given_mst)
print(f"Is the given MST unique? {unique}")
```

Output

```
Is the given MST unique? 19

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