TOPIC-5: GREEDY

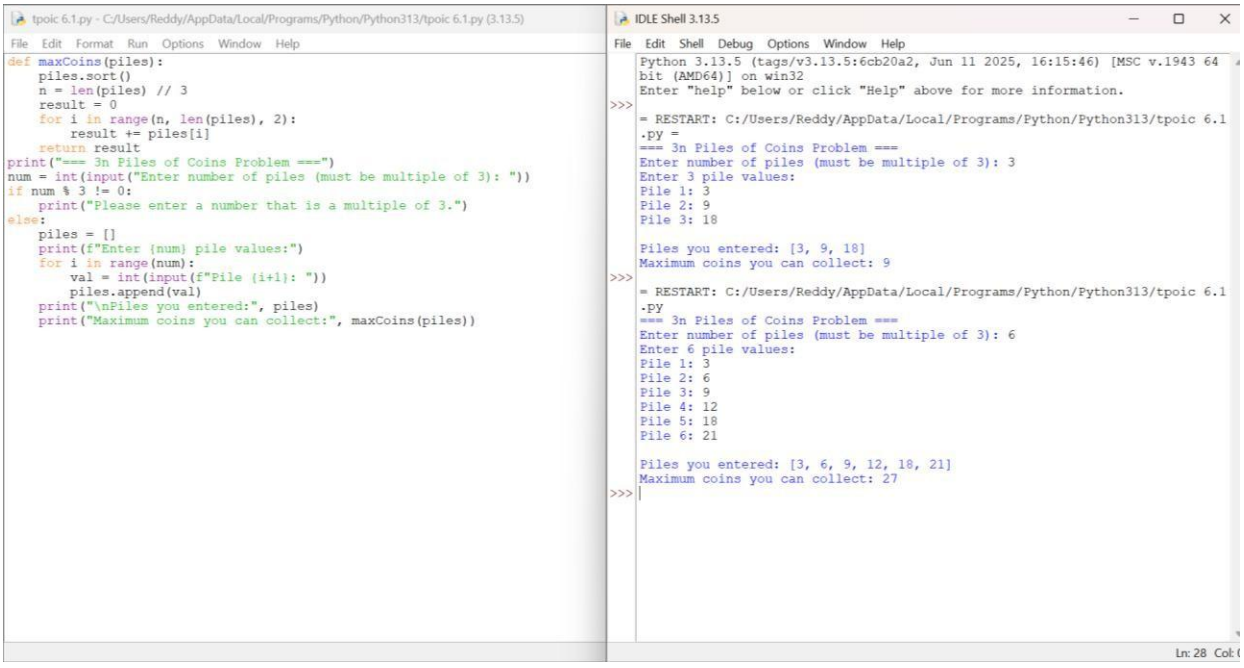
1.Maximum of Coins

Aim: To finds the maximum number of coins you can collect from 3n piles of coins**.**

Algorithm:

1. Start the program.
2. Read the number of piles n (must be a multiple of 3).
3. Input all the pile values (number of coins in each pile).
4. Sort the pile values in ascending order.
5. Initialize a variable result = 0 to store your total coins
6. Stop the program.

Input & Output: -



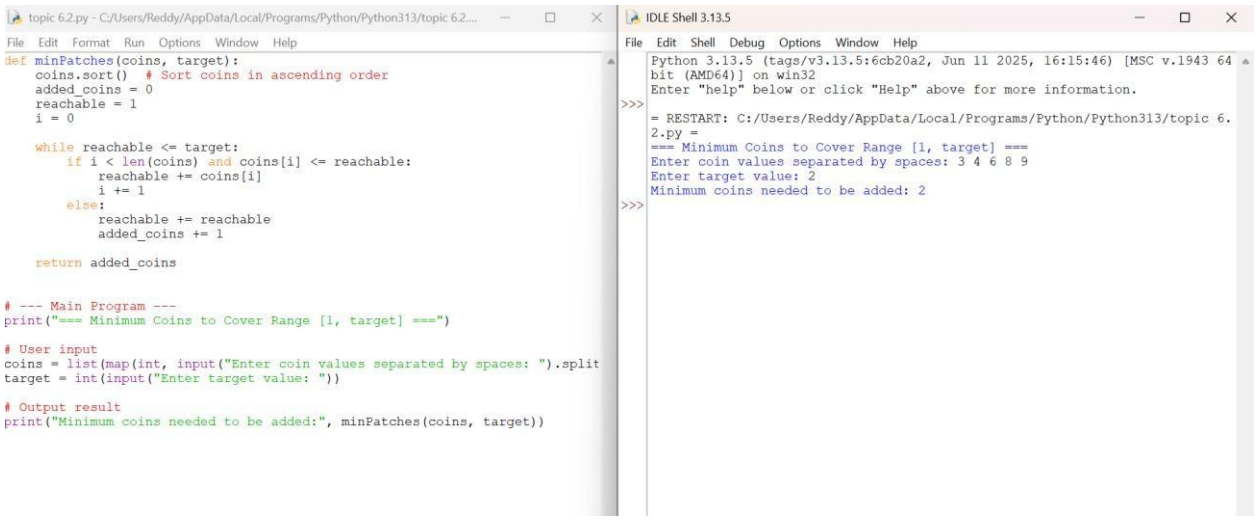
2.Minimum of Coins

Aim: To write a Python program that finds the minimum number of coins

Algorithm:

1. Start the program.
2. Input the list of existing coin values and the target value.
3. Sort the list of coins in ascending order.
4. Traverse through the coin list:
5. Continue this process until reachable > target.
6. Print the value of added coins as the minimum number of coins to be added.
7. Stop the program.

Input & Output: -



3.Maximum Working time

Aim:

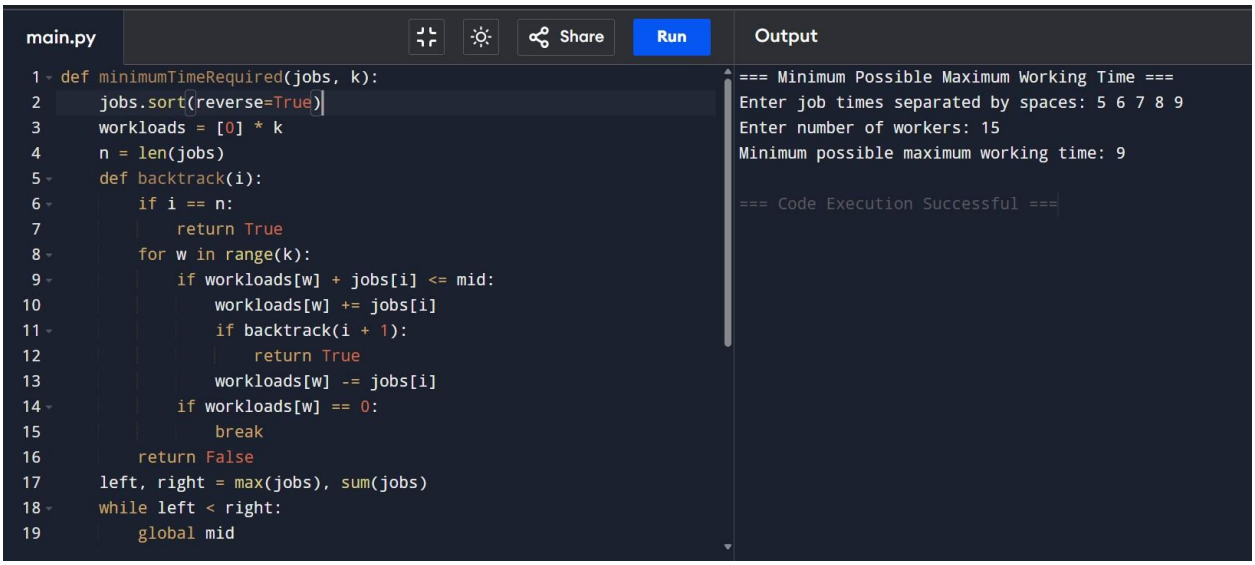
To write a Python program that assigns n jobs among k each job is assigned to exactly one worker.

Algorithm:

1. Start the program.
2. Input the list of job times and the number of workers (k).
3. Define a helper function.
4. Use Binary Search on the answer.
5. For each mid-value between low and high:

6. Stop the program.

Input & Output: -



4.Maximum Profit

Aim: To find the maximum profit subset of jobs such that no two jobs overlap.

Algorithm

1. Start the program.
2. Input arrays start Time, end Time, and profit.
3. Combine jobs into tuples and sort by end Time.
4. Initialize a DP array and an end times list.
5. Stop the program.

Input & Output

main.py

```python
from bisect import bisect_right
def jobScheduling(startTime, endTime, profit):
    jobs = sorted(zip(startTime, endTime, profit), key=lambda x: x[1]
        )
    n = len(jobs)
    dp = [0] * n
    end_times = [job[1] for job in jobs]
    for i in range(n):
        incl_profit = jobs[i][2]
        idx = bisect_right(end_times, jobs[i][0]) - 1
        if idx != -1:
            incl_profit += dp[idx]
        dp[i] = max(incl_profit, dp[i-1] if i > 0 else 0)
    return dp[-1]
startTime = [1, 2, 3, 3]
endTime = [3, 4, 5, 6]
profit = [50, 10, 40, 70]
print("Maximum Profit:", jobScheduling(startTime, endTime, profit))
```

Output

```
Maximum Profit: 120

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

4.Shortest Path

Aim: To find the shortest path from a given source vertex to all other vertices in a weighted graph represented by an adjacency matrix using Dijkstra's Algorithm.

1. Start the program.
2. Input the adjacency matrix graph and the source vertex.
3. Initialize distance array with infinity for all vertices, except the source vertex which is 0.
4. Initialize a visited array to keep track of visited vertices.
5. Print the distance array.
6. Stop the program.

Input & Output

main.py

```
def dijkstra(graph, src):
    n = len(graph)
    distance = [float('inf')] * n
    distance[src] = 0
    visited = [False] * n
    for _ in range(n):
        min_dist = float('inf')
        u = -1
        for i in range(n):
            if not visited[i] and distance[i] < min_dist:
                min_dist = distance[i]
                u = i
        if u == -1:
            break
        visited[u] = True
        for v in range(n):
            if graph[u][v] != float('inf') and not visited[v]:
                if distance[u] + graph[u][v] < distance[v]:
                    distance[v] = distance[u] + graph[u][v]
```

Output

```
Vertex  Distance from Source
0   0
1   4
2   12
3   19
4   21
5   11
6   9
7   8
8   14

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

5.Shortest Path (vertex to target)

Aim: To find the shortest path from a given source vertex to a target vertex in a weighted graph represented as an edge list using Dijkstra's Algorithm.

Algorithm:

1. Start the program.
2. Input the edge list, number of vertices, source vertex src, and target vertex tgt.
3. Convert the edge list into an adjacency list for efficient lookup.
4. Initialize distance array with infinity for all vertices, except the source vertex which is 0.
5. Use a priority queue (min-heap) to select the vertex with the minimum distance at each step while the priority queue is not emp.

Input & Output