Masai Profitable Schemes: Editorial

$\sim Omkar$

February 22, 2024

1 Problem Statement

In "Masai Profitable Schemes," the challenge is to calculate the number of ways a group of n individuals can perform various schemes to achieve at least a $minProfit$. Each scheme requires a certain number of group members to participate and yields a specific amount of profit. The objective is to use a dynamic programming approach to find the total number of combinations that meet or exceed the minimum profit requirement.

2 Summary

This problem is a complex one that intertwines the elements of group size, minimum profit requirements, and the individual schemes characterized by their profit and group size requirements. The essence is to find all possible combinations of schemes that cumulatively achieve or surpass a specified profit threshold.

3 Dynamic Programming Approach

Dynamic Programming is utilized, employing a two-dimensional dynamic programming array $dp[i][j]$, where i represents the number of group members used and j denotes the accumulated profit. This approach methodically explores all scheme combinations, updating the dp array to reflect the number of ways to achieve at least j profit with i members.

State Transition: The dp array is iteratively updated for each combination of group size and profit, including the current scheme or excluding it, applying modulo MOD to manage large numbers.

3.1 Pseudocode

```
// Initialize the dynamic programming array
Create a 2D array dp with dimensions (n+1) x (minProfit+1) and set all values to 0
Set dp[0][0] to 1 to represent one way to achieve 0 profit with 0 members

// Iterate over each scheme
For each scheme k from 0 to group.length - 1:
  Retrieve the group size g and profit p for the current scheme

  // Update the dp array from higher to lower to avoid overcounting
  For each possible number of members i from n down to g:
    For each possible profit j from minProfit down to 0:
      Update dp[i][j] by adding the number of ways without the current scheme (dp[i-g][max(0, j-p)])
      Apply modulo MOD to the result to avoid integer overflow

// Calculate the total number of profitable schemes
Initialize a variable totalSchemes to 0
For each possible number of members i from 0 to n:
  Add the number of ways to achieve at least minProfit using i members to totalSchemes
  Apply modulo MOD to totalSchemes

// Return the total number of profitable schemes
Return totalSchemes
```

Listing 1: Detailed Pseudocode

4 Implementations

4.1 Java Implementation

```java
import java.util.Scanner;

public class Main {
    public static int profitableSchemes(int n, int minProfit, int[]
        group, int[] profit) {
        int MOD = 1000000007;
        int[][] dp = new int[n + 1][minProfit + 1];
        dp[0][0] = 1;

        for (int k = 0; k < group.length; k++) {
            for (int i = n; i >= group[k]; i--) {
                for (int j = minProfit; j >= 0; j--) {
                    dp[i][j] = (dp[i][j] + dp[i - group[k]][Math.max(0,
                        j - profit[k])]) % MOD;
                }
            }
        }

        int result = 0;
        for (int i = 0; i <= n; i++) {
            result = (result + dp[i][minProfit]) % MOD;
        }

        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int minProfit = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        int[] group = Arrays.stream(scanner.nextLine().split(" ")).
            mapToInt(Integer::parseInt).toArray();
        int[] profit = Arrays.stream(scanner.nextLine().split(" ")).
            mapToInt(Integer::parseInt).toArray();

        System.out.println(profitableSchemes(n, minProfit, group,
            profit));
    }
}
```

Listing 2: Java Code

4.2 C++ Implementation

```cpp
#include <iostream>
#include <vector>
using namespace std;

int profitableSchemes(int n, int minProfit, vector<int>& group, vector<
    int>& profit) {
    const int MOD = 1e9 + 7;
    vector<vector<int>> dp(n + 1, vector<int>(minProfit + 1, 0));
    dp[0][0] = 1;

    for (int k = 0; k < group.size(); k++) {
        for (int i = n; i >= group[k]; i--) {
            for (int j = minProfit; j >= 0; j--) {
                dp[i][j] = (dp[i][j] + dp[i - group[k]][max(0, j -
                    profit[k])]) % MOD;
            }
        }
    }

    int result = 0;
    for (int i = 0; i <= n; i++) {
        result = (result + dp[i][minProfit]) % MOD;
    }

    return result;
}

int main() {
    int n, minProfit;
    cin >> n >> minProfit;
    vector<int> group(n), profit(n);
    for (int& g : group) cin >> g;
    for (int& p : profit) cin >> p;

    cout << profitableSchemes(n, minProfit, group, profit) << endl;
    return 0;
}
```

Listing 3: C++ Code

4.3 Python Implementation

```python
def profitableSchemes(n, minProfit, group, profit):
    MOD = 10**9 + 7
    dp = [[0] * (minProfit + 1) for _ in range(n + 1)]
    dp[0][0] = 1

    for g, p in zip(group, profit):
        for i in range(n, g - 1, -1):
            for j in range(minProfit, -1, -1):
                dp[i][j] = (dp[i][j] + dp[i - g][max(0, j - p)]) % MOD

    return sum(dp[i][minProfit] for i in range(n + 1)) % MOD

n, minProfit = map(int, input().split())
group = list(map(int, input().split()))
profit = list(map(int, input().split()))

print(profitableSchemes(n, minProfit, group, profit))
```

Listing 4: Python Code

4.4 JavaScript Implementation

```javascript
function profitableSchemes(n, minProfit, group, profit) {
    const MOD = 10**9 + 7;
    let dp = Array.from({length: n + 1}, () => Array(minProfit + 1).
        fill(0));
    dp[0][0] = 1;

    for (let k = 0; k < group.length; k++) {
        for (let i = n; i >= group[k]; i--) {
            for (let j = minProfit; j >= 0; j--) {
                dp[i][j] = (dp[i][j] + dp[i - group[k]][Math.max(0, j -
                    profit[k])]) % MOD;
            }
        }
    }

    let result = 0;
    for (let i = 0; i <=n; i++) {
        result = (result + dp[i][minProfit]) % MOD;
    }

    return result;
}

const n = parseInt(prompt("Enter the number of members: "), 10);
const minProfit = parseInt(prompt("Enter the minimum profit: "), 10);
const groupInput = prompt("Enter the group sizes separated by space: ")
    ;
const profitInput = prompt("Enter the profits separated by space: ");
const group = groupInput.split(" ").map(Number);
const profit = profitInput.split(" ").map(Number);

console.log(profitableSchemes(n, minProfit, group, profit));
```

Listing 5: JavaScript Code

5 Complexity Analysis

The time complexity for this solution is $O(group.length \times n \times minProfit)$, due to iterating over all schemes and the two-dimensional DP array update. The space complexity is $O(n \times minProfit)$ for storing the DP states. This highlights the algorithm's efficiency in handling problems with multiple constraints and objectives.