Kick Start 2022 - Coding Practice with Kick Start Session #2

Analysis: Sample Problem

View problem and solution walkthrough video

Solving this problem involves a number of steps. First, we must determine the total number of candies we have available to distribute. We can calculate this by summing together the number of candies in each bag, $total := \sum_{i=1}^{\mathbf{N}} \mathbf{C_i}$. Once we have the total number of candies, we must determine the number of candies that will remain after evenly distributing as many as possible among $\mathbf{M}$ kids. We can calculate this by dividing the total number of candies by the number of kids. $\mathbf{M}$ may not evenly divide $total$, so $(total \div \mathbf{M})$ will have an integer part (the amount each kid receives) and a remainder part (the amount left over). Because we only care about the remainder, we can use the modulo operation which does exactly this.

Example solution:

```
Solve(N, M, C) {
  total = 0
  for i in 1:N {
    total = total + C[i]
  }
  remainder = modulo(total, M)
  return remainder
}
```

Limits and complexity:

With the input limits for this problem, the default integer size for most programming languages will be sufficient to perform all arithmetic without risk of overflow. Specifically, the largest possible value is $total = (\mathbf{N_{max}} \times \mathbf{C_{i_{max}}}) = 10^8$, which is easily stored in a 32-bit signed integer ($int_{max} \approx 2.147 \times 10^9$). If the limits were larger, we could take advantage of the transitive and symmetric properties of modular arithmetic to calculate the remainder provided by each bag, and the running remainder after each addition:

```
Solve(N, M, C) {
  remainder = 0
  for i in 1:N {
    remainder = remainder + modulo(C[i], M)
    remainder = modulo(remainder, M)
  }
  return remainder
}
```

This reduces the largest intermediate value to $2 \times (\mathbf{M_{max}} - 1)$ with only a small amount of extra computation. When solving other problems in this competition, you are likely to encounter limits that will require this kind of optimization.

Computational complexity is also an important factor to consider when deciding how to approach a problem. Both approaches presented here have the same complexity, $O(\mathbf{N})$, which is sufficient to solve the problem within the time limit. Other problems in this competition can often be solved using a variety of approaches, but only some will be sufficiently fast to solve all test sets within the time limit. An algorithm with complexity $O(\mathbf{G}^3)$ may work for a small test set with $\mathbf{G}_{max} = 20$, but that same algorithm will be too slow for a large test set with $\mathbf{G}_{max} = 10^6$.

In general, time and memory limits for the problems in this competition, and the bounds for their test sets, are chosen so that algorithms pass or fail based on their overall time and memory complexity order. In other words, if a problem has a time limit of 60 seconds, it is unlikely that a fast algorithm would take 40 seconds while a slow one took 80 seconds. Instead, you will find that a fast algorithm might take 5 seconds while a slow one takes 5 years. So if you find your solution exceeding the time limit for a problem, take a moment to reevaluate your algorithm. There may be another approach that is many times faster.