2928. Distribute Candies Among Children I

Easy Math Combinatorics Enumeration

Problem Description

The task is to find the number of ways to distribute `n` candies among `3` children so that each child receives no more than `limit` candies. We are dealing with combinations and restrictions; it's not just simple combinatorics because of the limit on how many candies a child can receive.

Intuition

The intuition behind the solution is based on combinatorial mathematics and the Principle of Inclusion-Exclusion. The fundamental idea is to consider the distribution of candies as a problem of placing `n` identical items (candies) into 3 distinct buckets (children) with a constraint on the maximum items a bucket can hold (`limit`). Initially, we ignore the constraint and calculate the number of possible distributions as if the buckets had no limit. This is done by creating a visual of `n + 2` objects (which includes the `n` candies plus 2 dividers representing the separations between the 3 children), and choosing 2 spots from the `n + 2` to place the dividers. This is the combinatorial formula `C(n + 2, 2)`.

However, distributions where any child ends up with more than the `limit` of candies should not be counted. To exclude those, we subtract the combinations where at least one child has more than `limit` candies. For one child to exceed the limit, that child must have `limit + 1` candies at least, and the remaining candies must be distributed amongst the other two children, for which there are `C(n - limit + 1, 2)` ways.

We multiplied this count by 3, as any child can be the one to exceed the limit. This subtraction might also remove some combinations more than once (specifically, the cases where two children exceed the limit), so we use the Principle of Inclusion-Exclusion and add those cases back in, using the formula `C(n - 2 * limit, 2)` for the combinations where exactly two children exceed the limit, again multiplied by 3, for each pair of children this could apply to.

Putting it all together, we have the total count as `C(n + 2, 2)` minus `3 * C(n - limit + 1, 2)`, plus `3 * C(n - 2 * limit, 2)`, which gives us the final number of valid distributions.

Solution Approach

The implementation of the solution involves combinatorial mathematics to compute the number of ways to distribute the candies among the children, which aligns with the solution approach outlined in the reference.

Initially, we calculate the total number of ways to distribute `n` candies among `3` children without the limit by using the combination formula `comb(n + 2, 2)`. This is based on the concept of adding two virtual candies to `n` to account for the empty spaces that can be utilized to distribute the candies equally. Those two extra spaces represent the partitions that divide the candies among the three children.

Next, we must exclude the scenarios where one child receives more than the `limit` amount of candies. To do this, we reduce the count by `3 * comb(n - limit + 1, 2)` to discard the invalid distributions for each child. We multiply by 3 since this situation could pertain to each child independently.

However, this exclusion might remove some valid scenarios more than once, specifically where two children receive more than the `limit` at the same time. To correct this, we use the Principle of Inclusion-Exclusion and add back those over-excluded distributions by adding `3 * comb(n - 2 * limit, 2)` to the count.

The combination function `comb(m, k)` is usually implemented using factorials, but most programming languages, including Python, have built-in functions to compute combinations efficiently, avoiding potential overflow issues that can come with factorial calculations.

In the solution, we check whether `n` is greater than `3 * limit`, and in such a case, we directly return 0, since it's impossible to distribute more candies than `3 * limit` amongst three children without one child getting more than `limit` candies.

The given solution leverages these mathematical principles and built-in combination functions to calculate the precise number of valid candy distributions, ensuring that no child ends up with more than the `limit` amount of candies. By approaching the problem with mathematics rather than brute-force enumeration, the solution is both efficient and elegant.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach. Imagine we have `n = 7` candies to distribute among `3` children, and let's say the `limit` for each child is `2` candies.

1. **Without Limits**: Firstly, we ignore the limit and calculate the number of ways to distribute 7 candies. We do this by considering two dividers amongst 7 candies, which can be represented as follows:

   ```
   * * * * * * * | |  (two dividers and seven candies)
   ```

   By using the formula `C(n + 2, 2)`, which translates to `C(7 + 2, 2)`, we get `C(9, 2) = 36` ways.

2. **Subtracting Excess**: We then need to subtract the ways in which any one child gets more than 2 candies. One child getting at least 3 candies means there are `7 - 3 = 4` candies left to distribute to the other two children. With two dividers, the distribution could be something like:

   ```
   * * * | * * | * *  (one child receives 3 candies)
   ```

   For each child that receives 3 or more, we use the formula `C(n - limit + 1, 2)`, which in our case is `C(4 + 1, 2)`. There are `C(5, 2) = 10` such ways for one child. Since there are 3 children, we multiply this number by 3, so we get `3 * 10 = 30` ways to subtract.

3. **Adding Over-Excluded**: However, by doing this subtraction, we have also excluded the combinations where two children have exceeded the limit. This condition is represented by the formula `C(n - 2 * limit, 2)`, which for our example is `C(7 - 2 * 2, 2)`, or `C(3, 2)`, which equals 3. Since there are 3 possible pairs of children this could apply to, we get `3 * 3 = 9` distributions to add back.

Finally, we put it all together to calculate the total number of ways: `Total Ways = C(9, 2) - 3 * C(5, 2) + 3 * C(3, 2)` `Total Ways = 36 - 30 + 9` `Total Ways = 15`

Therefore, there are 15 different ways to distribute 7 candies among 3 children with the condition that no child receives more than 2 candies.

Solution Implementation

Python

```python
from math import comb

class Solution:
    def distributeCandies(self, candies: int, limit: int) -> int:
        # If the number of candies is more than 3 times the limit, it's not possible to distribute
        if candies > 3 * limit:
            return 0

        # Calculate the number of ways to distribute candies with no restriction using combinations
        total_ways = comb(candies + 2, 2)

        # Subtract combinations where any child gets more than the limit. We consider only cases
        # where it's possible for a child to exceed the limit (candies > limit).
        if candies > limit:
            total_ways -= 3 * comb(candies - limit + 1, 2)

        # Add back combinations that were subtracted multiple times for cases when more than
        # one child exceeds the limit (candies - 2 >= 2 * limit).
        if candies - 2 >= 2 * limit:
            total_ways += 3 * comb(candies - 2 * limit, 2)

        # Return the final count of ways to distribute the candies
        return total_ways
```

Java

```java
class Solution {
    // Method to distribute candies into a number of gift boxes.
    public int distributeCandies(int candies, int limit) {
        // If the number of candies exceeds 3 times the limit of a gift box, distribution is not possible.
        if (candies > 3 * limit) {
            return 0;
        }
        // Calculate the combinations for the distribution with no limits
        long combinations = combinationPairs(candies + 2);
        // If the number of candies is greater than the limit, adjust the combinations count.
        if (candies > limit) {
            combinations -= 3 * combinationPairs(candies - limit + 1);
        }
        // If the number of candies exceeds twice the limit for a single gift box, adjust again.
        if (candies - 2 >= 2 * limit) {
            combinations += 3 * combinationPairs(candies - 2 * limit);
        }
        // Since the result needs to be an integer, cast the long to an int before returning.
        return (int) combinations;
    }

    // Helper method to calculate the number of pair combinations.
    private long combinationPairs(int n) {
        // Combinatorial formula for the number of combinations of pairs which is n choose 2.
        return 1L * n * (n - 1) / 2;
    }
}
```

C++

```cpp
class Solution {
public:
    // Distributes candies between 1 to 'limit' number of people, ensuring no one
    // gets more than 'limit' candies, and returns the number of ways this can be done.
    int distributeCandies(int candies, int limit) {
        // Helper function to compute the number of combinations of 'candies' taken 2 at a time.
        auto calculateCombinationsForTwo = [](int number) {
            return 1LL * number * (number - 1) / 2;
        };

        // If the number of candies is more than triple the limit, it is impossible
        // to distribute them without someone getting more than 'limit' candies.
        if (candies > 3 * limit) {
            return 0;
        }

        // Calculate initial number of ways to distribute the candies without any limit.
        long long numberOfWays = calculateCombinationsForTwo(candies + 2);

        // If there are more candies than the limit, we need to subtract the combinations
        // where any person would get more than 'limit' candies. We multiply the number
        // of invalid combinations by 3 to account for each person potentially receiving too many.
        if (candies > limit) {
            numberOfWays -= 3 * calculateCombinationsForTwo(candies - limit + 1);
        }

        // If 'candies - 2' is at least twice the limit, we have previously subtracted
        // too many combinations (those where two people got too many candies) and need to
        // add back in the combinations where only one person exceeded the limit.
        if (candies - 2 >= 2 * limit) {
            numberOfWays += 3 * calculateCombinationsForTwo(candies - 2 * limit);
        }

        // The final answer is the number of valid ways to distribute the candies.
        return numberOfWays;
    }
};
```

TypeScript

```typescript
/**
 * Calculate the number of ways to distribute candies.
 *
 * @param {number} candies - The total number of candies to be distributed.
 * @param {number} limit - The maximum number of candies a single person can receive.
 * @return {number} The number of ways to distribute candies within the given limit.
 */
function distributeCandies(candies: number, limit: number): number {
    /**
     * Calculate the combination for choosing 2 items from n items (nC2).
     *
     * @param {number} n - Number of items to choose from.
     * @return {number} Number of combinations for choosing 2 from n items.
     */
    const combinationForTwo = (n: number): number => (n * (n - 1)) / 2;

    // If there are more than 3 times the limit of candies, no distribution is possible.
    if (candies > 3 * limit) {
        return 0;
    }

    // Start with the base number of combinations for distributing candies+2 items.
    let totalWays = combinationForTwo(candies + 2);

    // If there are more candies than the limit, subtract the invalid distributions.
    if (candies > limit) {
        totalWays -= 3 * combinationForTwo(candies - limit + 1);
    }

    // If the number of candies minus 2 is at least double the limit, add back some distributions.
    if (candies - 2 >= 2 * limit) {
        totalWays += 3 * combinationForTwo(candies - 2 * limit);
    }

    // Return the total number of valid distributions.
    return totalWays;
}
```

```python
from math import comb

class Solution:
    def distributeCandies(self, candies: int, limit: int) -> int:
        # If the number of candies is more than 3 times the limit, it's not possible to distribute
        if candies > 3 * limit:
            return 0

        # Calculate the number of ways to distribute candies with no restriction using combinations
        total_ways = comb(candies + 2, 2)

        # Subtract combinations where any child gets more than the limit. We consider only cases
        # where it's possible for a child to exceed the limit (candies > limit).
        if candies > limit:
            total_ways -= 3 * comb(candies - limit + 1, 2)

        # Add back combinations that were subtracted multiple times for cases when more than
        # one child exceeds the limit (candies - 2 >= 2 * limit).
        if candies - 2 >= 2 * limit:
            total_ways += 3 * comb(candies - 2 * limit, 2)

        # Return the final count of ways to distribute the candies
        return total_ways
```

Time and Space Complexity

The time complexity of the code is `O(1)` because the number of operations required to compute the solution does not depend on the size of the input `n` or `limit`. The code involves a fixed number of arithmetic operations and calls to the combination function `comb`, which calculates the binomial coefficient using a constant-time formula. No matter what values `n` and `limit` take, the computation steps inside the function do not change, hence the time complexity is constant.

The space complexity of the code is also `O(1)` because the amount of memory used by the code does not scale with the input size. It uses a fixed number of variables to hold interim results and the final answer. Regardless of the input, the space required remains the same, so the space complexity does not change with the input size.