Kick Start 2017 - Round F

Analysis: Cake

Eat Cake: Analysis

Small dataset

Let f(**N**) be the minimum number of cakes that have to be eaten such that the total combined area of the eaten cakes is **N**. To compute f(**N**), we can start by checking all possible sizes that we could use for the first cake that we eat. If the first eaten cake has an area of A × A, then we need to eat remaining cake(s) with a total combined area of **N** - A × A, which requires at least f(**N** - A × A) cakes to be eaten. Therefore, f(**N**) can be recursively computed as follows:

```
f(N)
  if (N = 0)
    return 0
  ans = infinity
  for i in [1, sqrt(N)]
    ans = min(ans, f(N - i * i) + 1)
  return ans
```

The algorithm above is fast enough to solve the Small dataset.

Large dataset

We need to use dynamic programming (DP) / memoization to solve the large dataset. If the value of f(**N**) has been recursively computed before, then the next time we need to know that value, we can simply use the previously computed value. Since the DP table of values has size O(**N**), and computing each value of f(**N**) takes O($\mathbf{N}^{0.5}$) time, the total time for this algorithm is O($\mathbf{N}^{1.5}$).

```
f(N)
  if (N = 0)
    return 0
  if (dp[N])
    return dp[N]
  dp[N] = infinity
  for i in [1, sqrt(N)]
    dp[N] = min(dp[N], f(N - i * i) + 1)
  return dp[N]
```

Alternative solution

There is another solution that does not even require a recursive strategy. From Lagrange's four-square theorem, we know that f(**N**) cannot be larger than 4. Therefore, for each k (1 ≤ k ≤ 3), we can simply have k nested loops of possible cake areas to determine whether k cakes are enough. If 3 cakes are still not enough, then the answer must be 4 (and we do not actually need to determine which cakes are used).

```
f(N)
  for i in [1, sqrt(N)]
    if (i * i = N)
```

```
    return 1
for i in [1, sqrt(N)]
  for j in [i, sqrt(N)]
    if (i * i + j * j = N)
      return 2
for i in [1, sqrt(N)]
  for j in [i, sqrt(N)]
    for k in [j, sqrt(N)]
      if (i * i + j * j + k * k = N)
        return 3
return 4
```

Since each layer of the triply nested loop adds O($\mathbf{N}^{0.5}$) time, the total time for this algorithm is also O($\mathbf{N}^{1.5}$).