Code Jam to I/O for Women 2015 - Code Jam to I/O for Women

Analysis: Dreary Design

The small input is small enough to be solved by checking every possible triple of colors to see whether it meets the definition:

```
def solve1(k, v):
  total = 0
  for r in range(0, k+1):
    for g in range(0, k+1):
      for b in range(0, k+1):
        if abs(b-r) <= v and abs(b-g) <= v and abs(g-r) <= v:
          total += 1
  return total
```

The first large input is too big for this, but can still be solved using nested for loops if you incorporate some observations:

1. The tolerance values can be used to bound the range of the for statements.
2. In that case, once R and G are fixed, thereâ€™s no need to look at every possible value of B; you can instead figure out the number of valid values directly.

```
def solve2(maxvalue, tolerance):
  total = 0
  for r in range(0, maxvalue+1):
    for g in range(max(r-tolerance, 0), min(r+tolerance, maxvalue)+1):
      minb = max(r-tolerance, g-tolerance, 0)
      maxb = min(r+tolerance, g+tolerance, maxvalue)
      total += (maxb - minb + 1)
  return total
```

But this is far too slow for the second Large input, in which K can range as high as 2 billion! How can we proceed?

As is often the case in programming contests, the specified limits provide a clue. Thereâ€™s no way to iterate over 2 billion values in time, so there must be a formulaic element to the answer. Indeed, if you look at the numbers of valid values beginning with each possible value for R, a pattern emerges. For example, consider this data for K = 10, V = 3:

R	0	1	2	3	4	5	6	7	8	9	10
Valid colors	16	23	30	37	37	37	37	37	30	23	16

Once weâ€™re at least V away from either end of the range of possible values for R, the number of valid colors is always the same! This makes sense because the number of valid colors is determined by an interaction between V and the boundaries of the range. And once weâ€™re far enough away from the boundaries, that number depends only on V. Moreover, the problem is symmetric. The situation at the beginning of the range is exactly the same as at the end of the range.

These observations suggest a modification that will work for the second Large input:

```
def solve3(k, v):
  total = 0
  for r in range(0, v+1):
```

```
    subtotal = 0
    for g in range(max(r-v, 0), min(r+v, k)+1):
      minb = max(r-v, g-v, 0)
      maxb = min(r+v, g+v, k)
      subtotal += (maxb - minb + 1)
    if r == v:
      total *= 2 # take advantage of symmetry
      total += subtotal * (k - 2*v + 1)
    else:
      total += subtotal
  return total
```

One of our contestants went even farther than that! Please check out this writeup, which derives a simple, elegant formula for the answer.