Bead Ornaments Challenge

The Bead Ornaments problem was one of the difficult challenges on Saturday's Hackathon. While it was easy to score a few points on some of the challenges (such as Mancala or Leibniz), the Bead Challenge was a "standard" challenge which required finding the solution to pass the test cases.

The first part of the challenge was to figure out exactly what was being asked. The challenge was to find out how many possible ornaments you can create with colored beads (given as input). An ornament consists of multiple tree-structures of beads. Each tree is made up of beads of one color and then the trees are connected together to create an ornament. So you need to figure out how many possible arrangement of trees can be made out of each color, and then how many possible ways there are to connect the trees together.

Jerry Ma ("yuiop" on HackerRank), from Purdue University, was one of the winners of the Back-to-School hackathon, and he kindly agreed to share his solution to the Bead Challenge. This post is based on Jerry's explanation.

Ways to Arrange Each Tree
The first task was figuring out how many possible formations can be created from 'n' beads of one color. I.e. how many ways can you arrange 'n' labeled nodes into a tree? We will call this function T(n).

Jerry was able to find the formula for this by working out some examples, and examining the sample inputs for the challenge:

n | T(n)
- 1 | 1
- 2 | 1
- 3 | 3   (Any one of the nodes can be in the middle.)
- 4 | 16 (Provided in sample case #3.)
- 5 | 125 (Provided in sample case #5, with the equivalent case of 5 beads of different colors.)

The number of possible trees looks like perfect powers of some sort. More specifically, $T(3) = 3^1$, $T(4) = 4^2$, and $T(5) = 5^3$. This pattern leads to the correct formula for the possible number of trees, which is also known as Cayley's Formula : $T(n) = n^{n-2}$.

Combining the Trees
The next part of the challenge is more difficult: How many ways are there to connect the different trees of each color?
Lets say we have two trees Tree 0 and Tree 1. Also, let:

arrange(i) = ways to arrange Tree i

b(n) = beads in Tree n.

If 0 and 1 were the only trees, the connecting formula would be simple. We can connect any node in any configuration of 0 to any node in any configuration of 1. So the formula for all the configurations would simply be:

config(01) = arrange(0) * b(0) * arrange(1) * b(1)

Since we know the formula for arranging trees, the above formula can be simplified to:
config(01) = $b(0)^{b(0)-1} * b(1)^{b(1)-1}$

However, when there are more trees, the challenge becomes more complicated. Let's say you wanted to connect another tree, 2, to the 0-1 tree structure. You can't just multiply config(01) by the number of ways to connect it to 2, since the trees could have been connected to each other in a different manner. In this case, Tree 2 could have been in-between 0 and 1.

Instead, you need a formula to determine the number of ways the colored trees can be arranged. One contest winner, Alexander Ramirez ("adr2370" on HackerRank), derived the formula for this arrangement:

$b(0)*b(1)*\ldots*b(n) * (b(0)+b(1)+\ldots+b(n))^{n-2}$

If you combine this formula with the number of ways to arrange each tree (T(n)), you get this:

$b(0)^{b(0)-1} * b(1)^{b(1)-1} * \ldots *b(n)^{b(n)-1} * (b(0)+b(1)+\ldots+b(n))^{n-2}$.

The formula above for combining two trees is a simplified version of this formula. Alexander realized that the number of ways to connect trees is related to the *sum* of b(i), the total number of beads, since each tree can be connected to any of the other connected beads. You can view Alexander's code for this solution on Github and can read more about his solutions on his blog.

Alternative Approach
What if you weren't able to derive the above formula in time? You still could have written code to solve the problem, which was the approach Jerry took. The question is to determine how many ways the trees can be connected to each other. Each combination of colored trees can be can be considered a single "state". Since there are at most 10 colors, there are at most 1024 (2^{10}) possible states, since each color is either included or excluded.

This information can easily be stored with bitmasks. Each bitmask can represent a state, so if a color is included in the state, the corresponding bit in the mask will be 1. We can then use dynamic programming to generate the answer for large states.

Let's assign the following definitions:

dp[mask] – the total number of configurations possible for the state represented by the mask.

sub-state – Each state has sub-states where the union of two sub-states's colors will result with the original state's colors.

The sub-state can be oriented in any way, so there will be a total of dp[submask_a] * dp[submask_b] possible orientations. In addition, we can combine two substates by joining together one bead from each subtree, for a total of num-beads[submask_a] * num-beads[submask_b] possible links. This leads to the following recurrence:

dp[color] = T(frequency(color))
dp[mask] = sum(dp[submask_a] * dp[submask_b] * num-beads[submask_a] * num-beads[submask_b]) for all submasks a and b.

This formula is the right idea, but it overcounts the number of possible configurations. For one, each pair of submasks gets counted two times. In addition, each configuration of beads gets counted n-1 times.

Therefore, we must divide the number of configurations for each mask by 2*(n-1). We then return the value dp[fulltree] % 1000000007, where fulltree can be expressed as $2^n - 1$.

Jerry's full solution in Java can be viewed on Github.

4
KUDOS

Related Posts

Purdue Hackathon

Functional Friday: Trees and Triangles

Programming Interviews – Techniques & Tips by Gayle Laakmann

HackerRank Hackathon Results

0 Comments HackerRank Login

Sort by Newest Share Favorite

Start the discussion…

Be the first to comment.

Subscribe Add Disqus to your site Privacy

Company
About Us
Careers
Blog

Docs
Scoring
Environment
FAQ

Other
Privacy Policy
Freenode
Contact Us

Facebook - Twitter