Code Jam 2011 - Round 1B

Analysis: House of Kittens

There are three different tasks you need to work through in order to solve this problem:

- How do you transform the input into a more convenient format?
- How do you efficiently calculate **C**?
- How do you efficiently find a catnip assignment with exactly **C** flavors?

An implementation would certainly start with the first task, but when you're just thinking about the problem, it's better to focus on the high-level algorithm first. How could you know what data format is convenient until you know what you want to do with it?

Finding an Optimal Assignment

So let's start with **C**. The most important observation is also one of the simplest. Let **m** be the minimum number of vertices in a single room. Kittens in that room have access to at most **m** flavors of catnip, so it must be that **C** ≤ **m**.

In fact, it turns out that **C** is always equal to **m**. Proving this is pretty much equivalent to the third sub-task: we need to give a method for assigning flavors that always work with **C** = **m**. Here it is:

- Choose an arbitrary room. Assign flavors to its vertices in such a way that all **C** flavors are used, and no two adjacent vertices use the same flavor.
- Choose a room adjacent to the starting one. This will have two different flavors fixed for two adjacent vertices. Fill in the remaining vertices as before: all **C** flavors are used, and no two adjacent vertices use the same flavor.
- Choose another room adjacent to one of the rooms previously considered. Again, it will have two different flavors fixed for two adjacent vertices, and nothing else. Proceed as before.
- Continue in exactly the same way until all rooms are complete.

There are two keys that make this work.

Key 1: It really is possible to assign valid flavors to the vertices of a single room, even after fixing distinct flavors for two adjacent vertices. Start with those two adjacent vertices, assign the remaining flavors to the next **C** - 2 vertices, and then just avoid equal neighbors for the remaining vertices. This is always possible since **C** ≥ 3.

Key 2: When we get to a new room, only two adjacent vertices will ever be fixed. To see why, let's say you just got to a room R by crossing some wall W. Then, W divides the house into two disjoint parts, so this will be the first time you are touching any room on the same side of W as R. In particular, this means the only vertices that will be fixed are the ones that belong to W.

That's it. Just be glad we are asking for an arbitrary assignment of flavors, instead of the lexicographically first one, or something equally evil!

Handling the Input

The main technical challenge in implementing this algorithm is figuring out where all the rooms are in the first place, and how they are connected.

One approach is to maintain a list of lists, representing the vertices in each room. We start off with just a single room: [[1, 2, ..., **N**]]. For each wall inside the house, we scan through the rooms until we find the one that has both endpoints of the wall, and then split that room into two. We then have to be a little careful about what order we process the rooms in. One option would be to start with an arbitrary room, and then only process future rooms once they have two vertices set. This runs in O($\mathbf{N}^2$) time.

There are also some fancier (almost) linear-time solutions. For each vertex, record the edges coming out of the vertex, sorted by the opposite endpoint. Now you can start with one face, trace along all of its edges, and then recursively proceed to the faces across each edge.

Either method works, so you can use whichever you prefer.