CS109B Project 2: Instant Insanity and Generalizations

Due Wednesday, June 7, 1995

There is a puzzle called *instant insanity* consisting of four cubes, with all faces colored white, red, green, or blue. The object is to stack the cubes so that on each of the four sides of the stack, there is one face of each color. We'll bring the cubes in for show and tell, since this description may not be adequately clear.

You can fool around with the cubes and (maybe) find the solution. With a computer, you could "brute-force" it. Assuming that one of the four sides of the stack is "front," we have six choices of the front face for each cube, and that chosen, the rear face is determined. However, by rotating the cube around the front-rear axis, you have four different selections for the left and right sides. Thus, there are "only" $24^4 = 331{,}776$ possible solutions. In fact, since any solution has four different orientations, found by rotating the stack, there are really only 82,944 possibilities if we organize things properly.

A Little Graph Theory

However, there is a graph representation of the problem that makes it surprisingly simple for a human to "eyeball" the problem. The graph representing the "standard" instant insanity puzzle has four nodes, one for each of the colors. Number the cubes 1–4. If cube i has a pair of opposite faces with colors X and Y, then connect nodes X and Y by an edge labeled i. The representation of standard II is shown in Fig. 1.

Note that this graph differs in two respects from the graphs we covered in Ch. 9:

1. "Loop" edges, connecting a node to itself, are permitted.
2. There can be more than one edge between two nodes. We could instead have allowed edge labels to be lists of numbers, but it appears easier to "eyeball" as we have shown the graph.

Finding a Solution for the Front and Rear Faces

Now what in this graph constitutes a solution. Suppose, e.g., that the top (4th) cube had red in front. Then in the rear that cube would have some color that was connected to the node R by an edge labeled 4 (either R or G). Suppose we chose G; i.e., the rear face of cube 4 is green.

There must be some other cube whose front face is green, because each color must appear on each side. Thus, there must be some edge out of G with a label other than 4. There are actually four such edges; let us choose the edge labeled 1 leading to W. That says cube 1 has front face green and rear face white.

Now think of where the white face is on the front. We need to pick an edge out of node W not labeled by one of the cubes (1 and 4) we have already used. Let us pick the edge labeled 3 from W to B. That is, the front face of cube 3 is white and its rear face blue.

Now, we need to find a place for blue in the front, and we have only cube 2. We also need to find red in the rear, and we also must find that on cube 2. Fortunately, there is an edge between B and R labeled 2, so we have "solved" part of the puzzle; i.e., we have

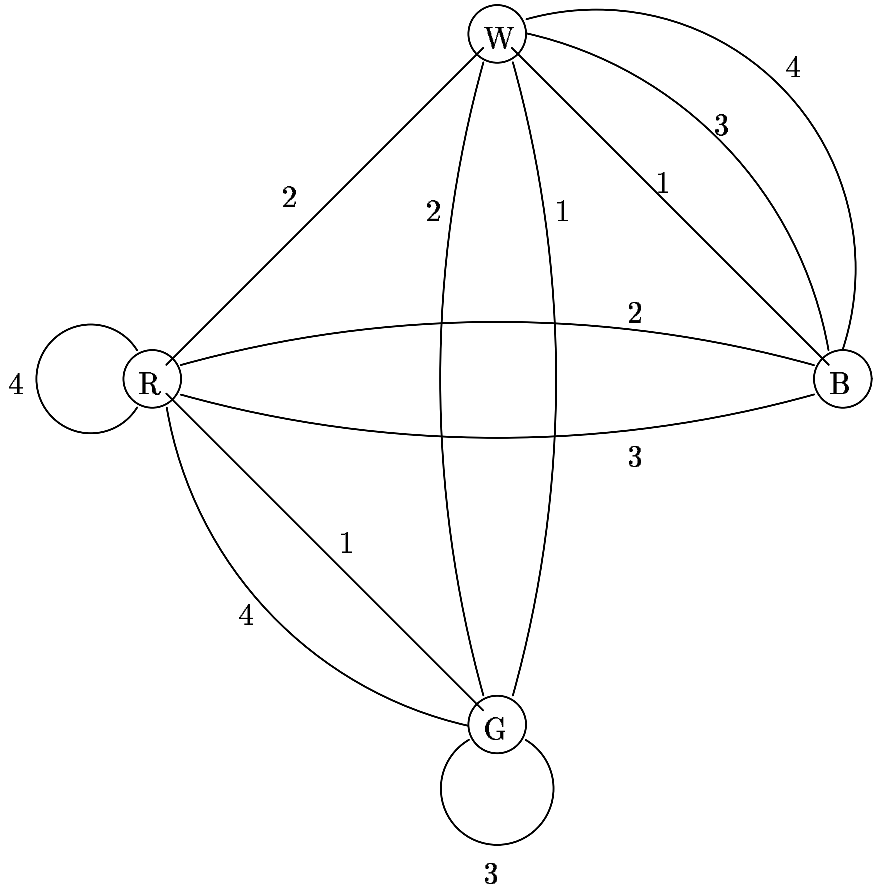

Fig. 1. The instant-insanity graph.

picked front and rear faces for each of the four cubes that gives us each color on each of those two sides.

In general, what have we found. We selected four edges, one with each label, that form a simple cycle involving all four nodes of the graph. That is what is needed for standard II. If we changed the faces on the cubes, we might have to look for something a bit more general. For example, suppose that in Fig. 1 the label on the edge between R and W were changed from 2 to 1. Then we could use the loop on G to cover green on both front and rear, and the combination of edges $\{W, R\}$ labeled 1, $\{R, B\}$ labeled 2, and $\{B, W\}$ labeled 4 to cover the other three colors on front and rear. The conclusion is that in the general problem we need to find four edges that:

1. Have each of the four labels among them, and

2. Are a set of node-disjoint, simple cycles, i.e., each node appears in exactly one of the cycles. Note, however, that unlike the definition of "simple cycle" in Ch. 9, where we required that it involve at least three nodes, here we must consider a loop to be a simple cycle and a pair of nodes connected by two different edges to be a simple cycle.

For future reference, call such a set of edges a *widget*.

Determining the Left and Right Sides

Having found a widget for the front and rear, we can still orient each cube so any pair of opposite faces are on the left and right. In graph terms, we can select for any cube i either of the two edges labeled i that are not part of the front-rear widget. Thus, we must look for a second widget using edges disjoint from those of the first widget. Any such widget can be used to orient the cubes correctly on left and right, while keeping the same faces front and rear. Thus, II reduces to the following problem:

- Find two edge-disjoint widgets.

Figure 2 shows the two widgets for standard II. Figure 3 shows one widget; it would be legal if the $\{R, W\}$ edge in Fig. 1 were changed to have label 1. It is important to remember that widgets such as these, with two or more cycles, will be necessary to solve some variations of II.

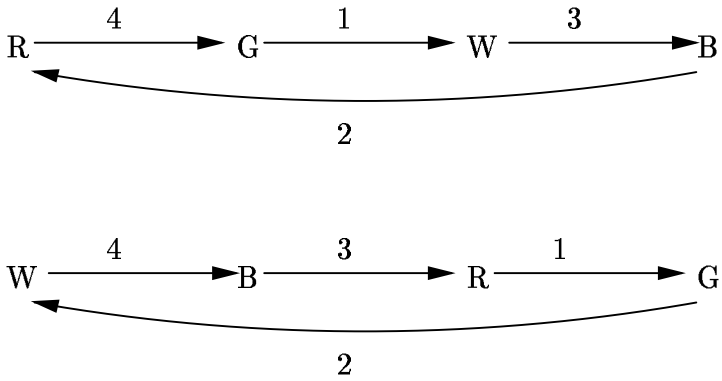

Fig. 2. The solution to standard II.

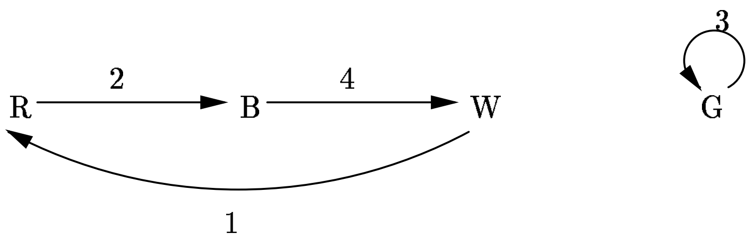

Fig. 3. A hypothetical widget with more than one cycle.

Your Mission

We need a general solution to II. By "general" I mean that the number of cubes in n, not necessarily 4. The number of colors will also be n. A cube is represented by the three pairs of colors on its opposite faces. A solution is a selection of colors for the front, rear, left, and right sides for each cube such that:

1. The selection is physically realizable; i.e., the colors on front and rear of the cube appear on opposite sides and the selection for left and right appear on two other opposite sides.

2. Each color appears exactly once on each of the four sides.

You may solve the problem either by brute force or by generalizing the notion of a widget to allow n nodes/colors and n edge-labels, rather than only 4. The notion of a widget remains the same: a set of node-disjoint simple cycles that include one edge of each label. Remember that some widgets may consist of more than one cycle, like that of Fig. 3. Also remember that you need two widgets with disjoint sets of edges to solve the problem.

Unlike Project 1, where we counted calls to `test` as the measure of running time, here we have the conventional notion of running time: the time taken by the computer's processor (often approximated by wall-clock time if the machine is not loaded). Your solution will be graded on both correctness — the ability to find solutions to some instances of II on which we shall test your program — and on the running time on those instances.

Representing Instances

We shall represent an instance by the following lines:

1. A line on which appear characters representing each of the colors, followed by a newline. You can deduce from this line not only the colors but the value of n, which is both the number of colors and the number of cubes.

2. For each cube, a line of 6 characters, each being one of the characters appearing in line 1. The first two characters are the colors of a pair of opposite faces; the third and fourth characters are a pair of opposite faces, and likewise the last two characters.

For example, the standard II instance could be represented by:

```
RGBW
WGWBRG
RWRBWG
WBRBGG
RRRGWB
```

That is, the colors are R, G, B, W, the first cube has the following pairs of opposite faces: $\{W, G\}$, $\{W, B\}$, and $\{R, G\}$, and so on for the other three cubes.

What to Hand In

Write an ML program with a "main" function `ii()` that reads a file `data` (review Ch. 9, EMLP for how to do this) to get an II instance and solves it, printing out the solution as a list of quadruples giving the colors of each cube, in order, on the front, left, rear, and right faces. For instance, the solution to the standard instance represented as above might be:

```
GRWG
BGRW
WBBR
RWGB
```

Deposit your program in the class `dropoff` directory. Also, hand in a writeup with the following information:

1. A listing of your code.
2. The names of all team members.
3. The name of the file in `dropoff` where your code can be found.
4. Perhaps one page describing the principal ideas behind your algorithm. What strategy are you using? What data structures?