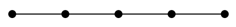
In-Class Problems Week 5, Wed.

Problem 1. Let L_n be the $(n+1)$-vertex *line graph*, which consists of a single simple path of length n. For example, here is L_4:



The line graph L_4.

A graph is *two-ended* if it has exactly two vertices of degree one and all other vertices have degree two. Note that L_n is a two-ended graph for every $n \geq 1$.

(a) Draw a diagram of the smallest two-ended graph that is *not* isomorphic to a line graph.

(b) Identify where the following proof makes a logical error (where something is deduced that didn't follow from the previous results and hypotheses).

False Theorem. *Every two-ended graph with n edges is isomorphic to L_n.*

False proof. We prove the thorem by induction on n, the number of edges in the graph, with the hypothesis

$$P(n) ::= \text{ every two-ended graph with } n \text{ edges is isomorphic to } L_n.$$

Base case, $n = 1$: A graph with one edge has the two vertices connected by that edge and some number of vertices not attached to any edge, that is, vertices of degree zero. A two-ended graph cannot have vertices of degree zero, so the only two-ended graph with one edge consists of that edge and the two vertices it joins, which makes it isomorphic to L_1.

Inductive step: Assume that $n \geq 1$, and let G_n be any two-ended graph with n edges. By hypothesis, G_n is isomorphic to L_n.

Now suppose we have a two-ended graph G_{n+1}. We will show that G_{n+1} is also a line graph. Consider how an edge can be added to the line graph G_n to form a two-ended G_{n+1}:

If an edge with two *new* vertices is added to any line graph, the result is not two-ended because it has four degree one vertices. If an edge is attached to one of the degree 2 ("middle") vertices of the line graph, the result is again not two-ended because it has a vertex of degree 3. So the only way to add an edge to the line graph to get a two-ended graph is to have that edge be incident on one side to one of the degree-one vertices —that is, to one end — and to on the other side to a *new* vertex. But adding such an edge to the end of a line graph yields a line graph that is one longer. So the resulting $(n + 1)$-edge graph, G_{n+1}, is indeed isomorphic to L_{n+1}. This proves $P(n + 1)$.

The Theorem follows by induction. □

Problem 2. In this problem you will prove:

Theorem. *A graph G is 2-colorable iff it contains no odd length cycle.*

As usual with "iff" assertions, the proof splits into two proofs: part (a) asks you to prove that the left side of the "iff" implies the right side. The other problem parts prove that the right side implies the left.

(a) Assume the left side and prove the right side. Three to five sentences should suffice.

(b) Now assume the right side. As a first step toward proving the left side, explain why we can focus on a single connected component H within G.

(c) Choose any 2-coloring of a spanning tree, T, of H. Prove that H is 2-colorable by showing that any edge *not* in T must also connect different-colored vertices.

Problem 3. A portion of a computer program consists of a sequence of calculations where the results are stored in variables, like this:

$$\begin{array}{rrcl}
 & \text{Inputs:} & & a, b \\
\text{Step } 1. & c & = & a + b \\
2. & d & = & a * c \\
3. & e & = & c + 3 \\
4. & f & = & c - e \\
5. & g & = & a + f \\
6. & h & = & f + 1 \\
 & \text{Outputs:} & & d, g, h
\end{array}$$

A computer can perform such calculations most quickly if the value of each variable is stored in a *register*, a chunk of very fast memory inside the microprocessor. Computers usually have few registers, however, so they must be used wisely and reused often. The problem of assigning each variable in a program to a register is called *register allocation*.

In the example above, variables a and b must be assigned different registers, because they hold distinct input values. Furthermore, c and d must be assigned different registers; if they used the same one, then the value of c would be overwritten in the second step and we'd get the wrong answer in the third step. On the other hand, variables b and d may use the same register; after the first step, we no longer need b and can overwrite the register that holds its value. Also, f and h may use the same register; once $f + 1$ is evaluated in the last step, the register holding the value of f can be overwritten.

(Assume that the computer carries out each step in the order listed and that each step is completed before the next is begun.)

(a) Recast the register allocation problem as a question about graph coloring. What do the vertices correspond to? Under what conditions should there be an edge between two vertices? Construct the graph corresponding to the example above.

(b) Color your graph using as few colors as you can. Call the computer's registers $R1$, $R2$, etc. Describe the assignment of variables to registers implied by your coloring. How many registers do you need?

(c) Suppose that a variable is assigned a value more than once, as in the code snippet below:

$$\begin{array}{rcl} & \ldots & \\ t & = & r + s \\ u & = & t * 3 \\ t & = & m - k \\ v & = & t + u \\ & \ldots & \end{array}$$

How might you cope with this complication?