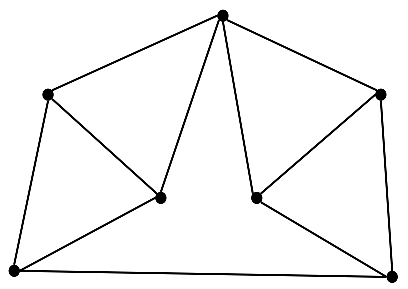
Solutions to In-Class Problems Week 7, Mon.

Problem 1.
Let G be the graph below[1]. Carefully explain why $\chi(G) = 4$.



Solution. Four colors are sufficient, so $\chi(G) \leq 4$.

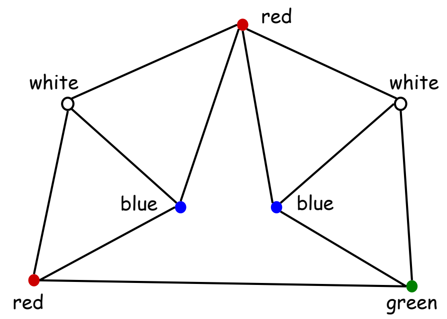


Figure 1: A 4-coloring of the Graph

Now assume $\chi(G) = 3$. We may assume the top vertex is colored red. The top two triangles require 3 colors each, and since they share the top red vertex, they must have the other two colors, white and blue, at their bases, as in Figure 1. Now the bottom two vertices are both adjacent to vertices colored white and blue, and cannot have the same color since they are adjacent, so there is no alternative but to color one with a third color and the other with a fourth color, contradicting the assumption that 3 colors are enough. Hence, $\chi(G) > 3$. This together with the coloring of Figure 1 implies that $\chi(G) = 4$. ■

[1]From *Discrete Mathematics*, Lovász, Pelikan, and Vesztergombi. Springer, 2003. Exercise 13.3.1

Problem 2.
A portion of a computer program consists of a sequence of calculations where the results are stored in variables, like this:

	Inputs:	a, b
Step 1.	c	$= a + b$
2.	d	$= a * c$
3.	e	$= c + 3$
4.	f	$= c - e$
5.	g	$= a + f$
6.	h	$= f + 1$
	Outputs:	d, g, h

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. Programming language compilers face the problem of assigning each variable in a program to a register. Computers usually have few registers, however, so they must be used wisely and reused often. This is called the *register allocation* problem.

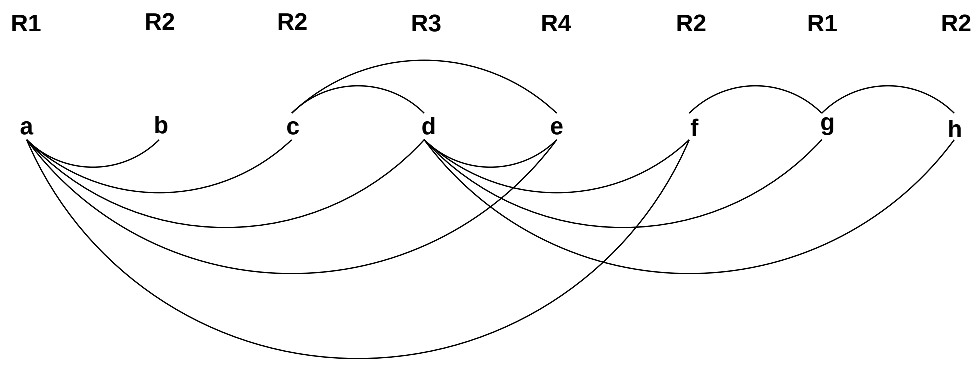
In the example above, variables a and b must be assigned different registers, because they hold distinct input values. Furthermore, c and d must be assigned different registers; if they used the same one, then the value of c would be overwritten in the second step and we'd get the wrong answer in the 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.

Solution. There is one vertex for each variable. An edge between two vertices indicates that the values of the variables must be stored in different registers.

We can classify each appearance of a variable in the program as either an *assignment* or a *use*. In particular, an appearance is an assignment if the variable is on the left side of an equation or on the "Inputs" line. An appearance of a variable is a use if the variable is on the right side of an equation or on the "Outputs" line. The *lifetime of a variable* is the segment of code extending from the initial assignment of the variable until the last use.[2] There is an edge between two variables if their lifetimes overlap. This rule generates the following graph:

[2]This definition is for the case that each variable is assigned at most once (see part (c)).



■

(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?

Solution. Four registers are needed.

One possible assignment of variables to registers is indicated in the figure above. In general, coloring a graph using the minimum number of colors is quite difficult; no efficient procedure is known. However, the register allocation problem always leads to an *interval graph*, and optimal colorings for interval graphs are always easy to find. This makes it easy for compilers to allocate a minimum number of registers. ■

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

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

How might you cope with this complication?

Solution. Each time a variable is reassigned, we could regard it as a completely new variable. Then we would regard the example as equivalent to the following:

$$\begin{array}{rcl} & \dots & \\ t & = & r+s \\ u & = & t*3 \\ t' & = & m-k \\ v & = & t'+u \\ & \dots & \end{array}$$

We can now proceed with graph construction and coloring as before. ■

Problem 3.
MIT has a lot of student clubs loosely overseen by the MIT Student Association. Each eligible club would like to delegate one of its members to appeal to the Dean for funding, but the Dean will not allow a student to be the delegate of more than one club. Fortunately, the Association VP took 6.042 and recognizes a matching problem when she sees one.

(a) Explain how to model the delegate selection problem as a bipartite matching problem.

Solution. Define a bipartite graph with the student clubs as one set of vertices and everybody who belongs to some club as the other set of vertices. Let a club and a student be adjacent exactly when the student belongs to the club. Now a matching of clubs to students will give a proper selection of delegates: every club will have a delegate, and every delegate will represent exactly one club. ■

(b) The VP's records show that no student is a member of more than 9 clubs. The VP also knows that to be eligible for support from the Dean's office, a club must have at least 13 members. That's enough for her to guarantee there is a proper delegate selection. Explain. (If only the VP had taken 6.046, *Algorithms*, she could even have found a delegate selection without much effort.)

Solution. The degree of every club is at least 13, and the degree of every student is at most 9, so the graph is *degree-constrained* (see the Appendix) which implies there will be no bottlenecks to prevent a matching. Hall's Theorem then guarantees a matching. ■

Problem 4.
A *Latin square* is $n \times n$ array whose entries are the number $1, \ldots, n$. These entries satisfy two constraints: every row contains all n integers in some order, and also every column contains all n integers in some order. Latin squares come up frequently in the design of scientific experiments for reasons illustrated by a little story in a footnote[3]

[3]At Guinness brewery in the eary 1900's, W. S. Gosset (a chemist) and E. S. Beavan (a "maltster") were trying to improve the barley used to make the brew. The brewery used different varieties of barley according to price and availability, and their agricultural consultants suggested a different fertilizer mix and best planting month for each variety.

Somewhat sceptical about paying high prices for customized fertilizer, Gosset and Beavan planned a season long test of the influence of fertilizer and planting month on barley yields. For as many months as there were varieties of barley, they would plant one sample of each variety using a different one of the fertilizers. So every month, they would have all the barley varieties planted and all the fertilizers used, which would give them a way to judge the overall quality of that planting month. But they also wanted to judge the fertilizers, so they wanted each fertilizer to be used on each variety during the course of the season. Now they had a little mathematical problem, which we can abstract as follows.

Suppose there are n barley varieties and an equal number of recommended fertilizers. Form an $n \times n$ array with a column for each fertilizer and a row for each planting month. We want to fill in the entries of this array with the integers 1,. . . ,n numbering the barley varieties, so that every row contains all n integers in some order (so every month each variety is planted and each fertilizer is used), and also every column contains all n integers (so each fertilizer is used on all the varieties over the course of the growing season).

For example, here is a 4×4 Latin square:

1	2	3	4
3	4	2	1
2	1	4	3
4	3	1	2

(a) Here are three rows of what could be part of a 5×5 Latin square:

2	4	5	3	1
4	1	3	2	5
3	2	1	5	4

Fill in the last two rows to extend this "Latin rectangle" to a complete Latin square.

Solution. Here is one possible solution:

2	4	5	3	1
4	1	3	2	5
3	2	1	5	4
1	5	2	4	3
5	3	4	1	2

■

(b) Show that filling in the next row of an $n \times n$ Latin rectangle is equivalent to finding a matching in some $2n$-vertex bipartite graph.

Solution. Construct a bipartite graph as follows. One set of vertices are the columns of the Latin rectangle, and the other set is the numbers 1 to n. Put an edge between a column and a number if the number has *not yet appeared* in the column. Thus, a matching in this graph would associate each column with a distinct number that has not yet appeared in that column. These numbers would form the next row of the Latin rectangle. ■

(c) Prove that a matching must exist in this bipartite graph and, consequently, a Latin rectangle can always be extended to a Latin square.

Solution. Suppose the Latin rectangle has k rows of width n. Then each column-vertex has degree $n-k$ because its edges go to the $n-k$ numbers missing from the column. Also, each number-vertex also has degree $n - k$. That's because each number appears exactly once in each of the k rows and at most once in each column, so each number must be missing from exactly $n - k$ columns.

So the graph is degree-constrained and therefore has a matching. This implies that we can add rows to the Latin rectangle by the procedure described above as long as $k < n$. At that point, we have a Latin square. ■