Problem Set 5 Solutions

Due: Monday, March 7 at 9 PM in Room 32-044

Problem 1. An undirected graph G has ***width*** w if the vertices can be arranged in a sequence

$$v_1,\ v_2,\ v_3,\ \ldots,\ v_n$$

such that each vertex v_i is joined by an edge to at most w preceding vertices. (Vertex v_j *precedes* v_i if $j < i$.) Use induction to prove that every graph with width at most w is $(w+1)$-colorable.

(Recall that a graph is k-*colorable* iff every vertex can be assigned one of k colors so that adjacent vertices get different colors.)

Solution. We use induction on n, the number of vertices. Let $P(n)$ be the proposition that every graph with width w is $(w + 1)$ colorable.

Base case: Every graph with $n = 1$ vertex has width 0 and is $0 + 1 = 1$ colorable. Therefore, $P(1)$ is true.

Inductive step: Now we assume $P(n)$ in order to prove $P(n+1)$. Let G be an $(n+1)$-vertex graph with width w. This means that the vertices can be arranged in a sequence

$$v_1, v_2, v_3, \ldots, v_n, v_{n+1}$$

such that each vertex v_i is connected to at most w preceding vertices. Removing vertex v_{n+1} and all incident edges gives a graph G' with n vertices and width at most w. (If original sequence is retained, then removing v_{n+1} does not increase the number of edges from a vertex v_i to a preceding vertex.) Thus, G' is $(w + 1)$-colorable by the assumption $P(n)$. Now replace vertex v_{n+1} and its incident edges. Since v_{n+1} is joined by an edge to at most w preceding vertices, we can color v_{n+1} differently from all of these. Therefore, $P(n + 1)$ is true.

The theorem follows by the principle of induction.

Problem 2. In this problem, we walk you through a proof of the following theorem:

Theorem 1. *A graph G is bipartite if and only if it contains no odd cycle.*

As is common with "if and only if" assertions, the proof has two parts. Part (a) asks you to prove that the left side implies the right side. Parts (b), (c) and (d) help you prove that the right side implies the left.

(Recall that a graph is *bipartite* iff it is 2-colorable.)

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

Solution. Select a 2-coloring of G. Consider an arbitrary cycle with vertices $v_1, v_2, \ldots, v_k$. Then the vertices v_i must be one color for all even i and the other color for all odd i. Since v_1 and v_k must be colored differently, k must be even. Thus, the cycle has even length.

(b) Now we're going to prove that the right side implies the left. As a preliminary step, explain why there is a 2-coloring of any tree T.

Solution. Select a "root" vertex r arbitrarily. Color a vertex v red if the distance in T from r to v is even, and color v black if the distance from r is odd.

To see that this defines a valid 2-coloring of the tree T, consider an arbitrary edge x—y. There is a unique path from x to r and a unique path from y to r. One of these paths must contain the edge x—y; otherwise, this edge together with portions of the other two paths would form a cycle.

If x—y is on the unique path from x to r, then y is exactly one edge closer than x to the root r. Similarly, if x—y is on the unique path from y to r, then x is one edge closer than y to the root. In either case, one vertex is at an odd distance from r and the other is at an even distance from r, so the two vertices are differently-colored.

(c) Suppose that G is a *connected* graph with no odd cycle, and let T be a spanning tree of G. Show that a valid 2-coloring of T also defines a valid 2-coloring of G.

Solution. Vertices connected by an edge in T have different colors in a 2-coloring of T, so we need only show that an edge x—y *not* in the tree must connect different-colored vertices.

Let z be the first common vertex on the path in T from x to the root v_0 and the path in T from y to v_0. The path from x to z, followed by the path from z to y, followed by the edge y—x form a cycle which, by hypothesis, must be of even length. Since the length of the cycle is one plus the sum of the lengths of the two paths, one of the paths must have odd length and the other even length. This implies that of the paths from x to v_0 and from y to v_0, one has even length and the other odd length. From the way we 2-colored T, it follows that x and y must be colored differently.

(d) Now assume the right side of Theorem **??**; namely, G is a graph with no odd cycle, but is not necessarily connected. Prove that G is bipartite.

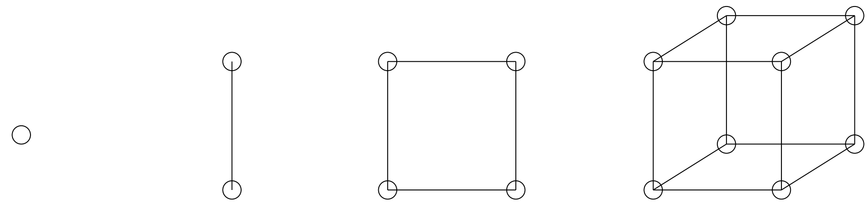
Solution. Color each connected component of G with 2 colors one at a time.

Problem 3. Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be undirected graphs in which every vertex has degree 1. The *union* of graphs G_1 and G_2 is the graph $G = (V, E_1 \cup E_2)$. Prove that G is bipartite.

Solution. Consider any cycle in G. If an edge on the cycle is in E_1, then the edges on either side in the cycle must be in E_2; otherwise, the degree of G_1 would be greater than

1. Thus, the edges around the cycle are alternately in E_1 and E_2. This implies the length of the cycle must be even, so G is bipartite by Theorem **??**.

Problem 4. For each integer $n \geq 0$, we can define a graph called an *n-cube*, which is denoted H_n. The vertices are all n-bit sequences, $\{0,1\}^n$. There is an edge between two vertices if they differ in exactly one bit position.



0–cube **1–cube** **2–cube** **3–cube**

(a) Prove that H_n has an Euler tour for all even $n \geq 2$.

Solution. We prove that H_n has an Euler tour by showing that it is connected and every vertex has even degree.

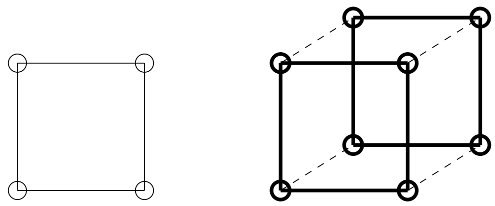
First, consider two arbitrary vertices in an n-cube, $b_1 b_2 \ldots b_n$ and $c_1 c_2 \ldots c_n$. These two vertices are joined by a path which traverses the distinct vertices in the following sequence:

$$
\begin{gathered}
b_1 b_2 b_3 \ldots b_n \\
c_1 b_2 b_3 \ldots b_n \\
c_1 c_2 b_3 \ldots b_n \\
\ldots \\
c_1 c_2 c_3 \ldots c_n
\end{gathered}
$$

Therefore, the graph is connected.

Each vertex v in an n-cube is adjecent to exactly n other vertices, those which can be obtained by flipping one of the n bits of vertex v. Therefore, if n is even, every vertex in an n-cube has even degree. Thus, H_n has an Euler tour for all even $n \geq 2$.

(b) Prove that H_n has a Hamiltonian cycle for all $n \geq 2$. Consider an inductive argument based on the fact that H_{n+1} consists of two subgraphs isomorphic to H_n (bold lines) with corresponding vertices joined (dotted lines):



2–cube **3–cube**

Solution. We use induction on n. Let $P(n)$ be the proposition that an n-cube has a Hamiltonian cycle. A 2-cube is itself a cycle on four vertices, so $P(2)$ is true. Now suppose that $P(n)$ is true for some $n \geq 2$, and consider an $(n+1)$-cube. As noted in the suggestion, this graph consists of two subgraphs isomorphic to the n-cube, with corresponding vertices joined by edges. By induction, there is a Hamiltonian cycle C_n in one of these two subgraphs and a corresponding Hamiltonian cycle C'_n in the other. Deleting an edge u—v from C_n, deleting the corresponding edge u'—v' from C'_n, and adding the edges u—u' and v—v' creates a Hamiltonian cycle in the $(n+1)$-cube. Therefore, $P(n+1)$ is true, and the claim follows by the principle of induction.

Problem 5. 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. 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.

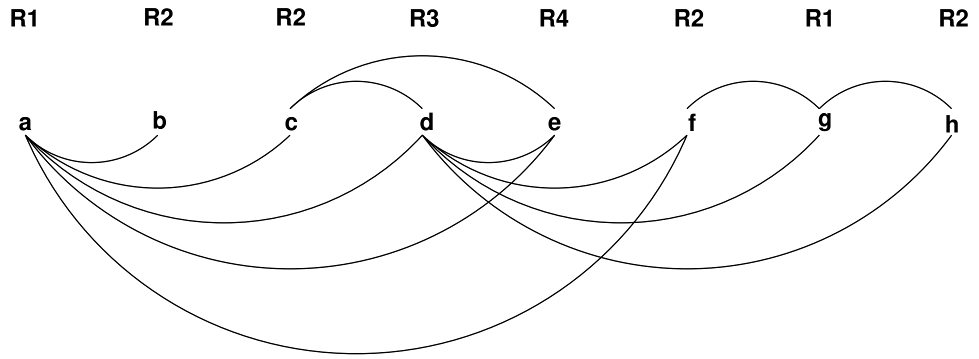
(Assume that the computer carries out each step in the order listed and that each step is completed before the next is begun.)

(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.

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. There is an edge between two variables if their lifetimes overlap. This rule generates the following graph:



(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*. For interval graphs, there are efficient coloring procedures, which can be incorporated into a compiler.

(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