Staff Solutions to Problem Set 6

STAFF NOTE: Topics:
Connection, Graph Coloring, Bipartite Matching, Recursive Data Types.

Problem 1.
An edge is said to leave a set of vertices if one end of the edge is in the set and the other end is not.

(a) An \( n \)-node graph is said to be mangled if there is an edge leaving every set of \( \lfloor n/2 \rfloor \) or fewer vertices. Prove the following claim.
Claim. Every mangled graph is connected.

Solution. The proof is by contradiction. Assume for the purpose of contradiction that these exists an \( n \)-node graph that is mangled, but not connected. Then the graph must have at least two connected components. However, there can be at most one connected component of size more than \( \lfloor n/2 \rfloor \), since \( 2 (\lfloor n/2 \rfloor + 1) > n \). Therefore, there exists a connected component with \( \lfloor n/2 \rfloor \) or fewer vertices. Since the graph is mangled, there is an edge leaving this component. But this contradicts the definition of a connected component.

An \( n \)-node graph is said to be tangled if there is an edge leaving every set of \( \lceil n/3 \rceil \) or fewer vertices.

(b) Draw a tangled graph that is not connected.

Solution. A tangled but non-connected graph is the following:

(c) Find the error in the proof of the following False Claim. Every tangled graph is connected.

False proof. The proof is by strong induction on the number of vertices in the graph. Let \( P(n) \) be the proposition that if an \( n \)-node graph is tangled, then it is connected. In the base case, \( P(1) \) is true because the graph consisting of a single node is trivially connected.
For the inductive case, assume \( n \geq 1 \) and \( P(1), \ldots, P(n) \) hold. We must prove \( P(n + 1) \), namely, that if an \((n + 1)\)-node graph is tangled, then it is connected.

So let \( G \) be a tangled, \((n + 1)\)-node graph. Choose \( \lceil n/3 \rceil \) of the vertices and let \( G_1 \) be the tangled subgraph of \( G \) with these vertices and \( G_2 \) be the tangled subgraph with the rest of the vertices. Note that since \( n \geq 1 \), the graph \( G \) has at least two vertices, and so both \( G_1 \) and \( G_2 \) contain at least one vertex. Since \( G_1 \) and \( G_2 \) are tangled, we may assume by strong induction that both are connected. Also, since \( G \) is tangled, there is an edge leaving the vertices of \( G_1 \) which necessarily connects to a vertex of \( G_2 \). This means there is a path between any two vertices of \( G \): a path within one subgraph if both vertices are in the same subgraph, and a path traversing the connecting edge if the vertices are in separate subgraphs. Therefore, the entire graph, \( G \), is connected. This completes the proof of the inductive case, and the Claim follows by strong induction.

\[ \blacksquare \]

**Solution.** The error is in the statement, “Let \( G_1 \) be the tangled subgraph . . . .” This makes the implicit assumption that a tangled graph can be split into tangled subgraphs, one of which is of size at most \( \lceil n/3 \rceil \). This assumption is false. To see why, consider the counterexample given in part (b). That graph is tangled and \( \lceil n/3 \rceil = 2 \). But, no matter how we split the graph into two subgraphs of sizes either 2 and 4 or 1 and 5, one of the two is not tangled.

It’s a common blunder to assume that a property of a graph is “inherited” by a subgraph. This is true for many familiar properties such as being \( k \)-colorable, having degrees bounded by a constant \( d \), being planar, being no larger than the whole graph. But many other familiar are not inherited, for example, being a tree, being a simple cycle, requiring more than \( k \) colors, being nonplanar.

\[ \blacksquare \]

**Problem 2.**

My computer program has seven variables, \( t, u, v, w, x, y, z \), and computes in 6 steps. The steps in which each variable is used are as follows: \( t \): steps 1 through 6; \( u \): step 2; \( v \): steps 2 through 4; \( w \): steps 1,3 and 5; \( x \): steps 1 and 6; \( y \): steps 3 through 6; \( z \): steps 4 and 5. Each variable will need to occupy the same index register at each one of the steps in which it is going to be used; however during the steps when the variable is not being used the register may be used by some other variable. How many such registers are needed for my program? Explain how you arrived at the answer.

**Solution.** This problem shows an application of graph coloring to register allocation in a processor. Consider a simple graph \( G \) having the seven variables as its nodes, and an edge between two variables iff there is at least one step of the program during which both variables are going to be used. Figure 1 contains the graph we are talking about.

Clearly, two variables have to occupy different index registers iff they are going to be used simultaneously sometime during the program execution, that is, iff they are connected in \( G \).

To correctly and economically assign variables to registers means to map each variable to one register in a way that does not cause collisions (two simultaneously used variables mapped to the same register) and uses as few registers as possible. But (if we think of registers as colors) this is the same as to color \( G \) in a way that does not cause neighboring vertices of the same color and uses as few colors as possible. So, we have reduced the original problem to the problem of coloring \( G \).
Figure 1:

Figure 2 shows such a coloring, using 5 colors. To see that this is the minimum one, just notice that variables \(t, v, w, y,\) and \(z\) form a 5-vertex subgraph which is complete, and hence cannot be colored with less than 5 colors.

SIDE NOTE: In this problem we have assumed that a variable must always use the same register, i.e. a variable is assigned a single color. However in a real compiler this is overly restrictive because a variable is backed up into memory every time it is evicted from the register set and then reloaded from memory into the register set when it needs to be used again. When it is reloaded there is no reason why the same register must be used. For example, say we had a variable \(x\) which is used in steps 1,2,3 and then later used in step 6. Then \(x\) could be stored in register \(r_1\) for steps 1 through 3 and some other register \(r_2\) for step 6. We can still use coloring to solve the problem by treating \(x\) as two separate variables: \(x_{123}\) and \(x_6\). Since the number of variables in a computer program can be very large and the register set of a processor is typically much smaller (16-64 registers), register allocation is an important piece of compiling.

Problem 3.
The set \(\text{Supersymm}\) of “super-symmetric strings” is defined recursively as follows:

**Base Case:** The 26 lower case letters of the Roman alphabet, \(a, b, \ldots, z,\) are in \(\text{Supersymm}\).

**Constructor Case:** If \(\alpha\) and \(\beta\) are strings in \(\text{Supersymm}\), then the string \(\alpha\beta\alpha\) is in \(\text{Supersymm}\).

(a) Which of the following are super-symmetric strings? Briefly explain your answers.

(i) \(a\)

**Solution.** Yes, by the Base Case.

(ii) \(aaaba\)
Figure 2:

Solution. No. This string is not of the form $\alpha\beta\alpha$.

(iii) $abcbacabcbab$

Solution. Yes. Let $\beta = aca$, $\alpha = bcb$. Then we have a string of the form $aa\beta\alpha a$.

(iv) $\lambda$, the empty string

Solution. No. A trivial structural induction implies that all super-symmetric strings have positive length.

(v) $abaabcbaaba$

Solution. Yes. Similar reasoning to case (iii) shows that the string $bcb$ is in the middle of the super-symmetric string, with the string $a$ wrapped around it, and with the string $aba$ wrapped around that.

(b) Prove by structural induction that in any super-symmetric string, exactly one letter appears an odd number of times.

Solution. Proof: Define $P(e)$: String $e$ has exactly one letter which appears an odd number of times.

Prove: $\forall e \in SSS \; P(e)$.
1. (Base) $P(a)$ The string $a$ has exactly one letter which appears an odd number of times.
2. (Base) $P(b)$ Same for the string $b$
   
   
   26. (Base) $P(z)$ Still has exactly one letter which appears once.
27. (Inductive Step) \( \forall e, e' \in SSS \ P(e) \land P(e') \rightarrow P(e'e) \)

1. Fix \( e, e' \in SSS \).
2. Assume \( P(e) \land P(e') \). That is, each of \( e \) and \( e' \) has exactly one letter which appears and odd number of times.
3. By the inductive hypothesis, both \( e \) and \( e' \) have exactly one letter which appears an odd number of times. When we form the string \( ee'e \), the letters in \( e \) all get repeated, and thus they all appear an even number of times. By the IH, there is one letter in \( e' \) which appears an odd number of times in \( e' \). Now, even if this letter is also present in \( e \), it will still appear an odd number of times in \( ee'e \) (since the sum of an odd number and an even number is odd). Therefore, we have shown \( P(e'e) \).
4. QED.

Problem 4.
Take a regular deck of 52 cards. Each card has a suit and a value. The suit is one of four possibilities: heart, diamond, club, spade. The value is one of 13 possibilities, \( A, 2, 3, \ldots, 10, J, Q, K \). There is exactly one card for each of the \( 4 \times 13 \) possible combinations of suit and value.

Ask your friend to lay the cards out into a grid with 4 rows and 13 columns. They can fill the cards in any way they’d like. In this problem you will show that you can always pick out 13 cards, one from each column of the grid, so that you wind up with cards of all 13 possible values.

(a) Explain how to model this trick as a bipartite matching problem between the 13 column vertices and the 13 value vertices. Is the graph necessarily degree constrained?

Solution. Create a simple bipartite graph with 13 column vertices and 13 value vertices. Connect a column to a value by a single edge iff a card with that value is contained in that column. A perfect matching would then indicate the value of the card you would choose from each column. The graph may not be degree constrained if any one of the columns contains more than one card with the same value. In the case where the matching indicates a value that appears more than once in the column it is matched to, you can arbitrarily pick any card of that value in that column.

(b) Show that any \( n \) columns must contain at least \( n \) different values and prove that a matching must exist.

Solution. If \( S \) is a set of columns, they contain \( 4|S| \) cards. No card value repeats more than four times, so at least \( |S| \) values must appear among those cards. Thus \( |N(S)| \geq |S| \) and Hall’s theorem gives us a matching.