

## Midterm 2 Solutions, CS620, McConnell, Spring '08

1. Why can a vertex never get closer to  $s$  in the residual graph if you augment along a shortest path?

*All new edges created in the residual graph point backward along the shortest path, which means that they point from a farther vertex to a closer vertex. They can't create a shortcut to any vertex.*

2. Derive an  $O(\log n)$  amortized bound on the number of splices in an expose operation.

*A runt is a node whose subtree is at most half the size of the subtree of its parent. There are  $O(\log n)$  runts on any path to the root. Maintain the invariant that each runt that has the solid edge from its parent also has a credit. During an expose, deposit a credit on any runt that steals the solid edge from a sibling, for a total of  $O(\log n)$  credits out-of-pocket. For every non-runt that steals the solid edge from a sibling, the sibling is a runt, so you can use its credit to pay for the splice.*

*When you link two trees, if the joined subtree's root is a runt, add a credit to it. This is subsumed by the  $O(\log n)$  cost of the expose that occurs when you join them. Cutting two trees doesn't require any new credits.*

3. In class, we derived a common-sense algorithm for finding a minimum coloring of an interval graph, given the intervals. Proceed left-to-right. Every time you get to a new interval, assign it the smallest color number that isn't already in use by one of its neighbors.

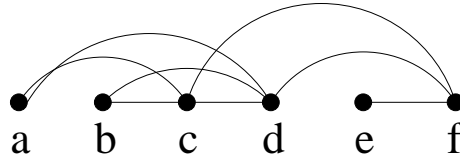
*Get an  $O(n + m \log n)$  algorithm for finding a minimum coloring in a chordal graph by generalizing this algorithm. Show the bound by showing that assigning a color to a vertex  $v$  requires  $O(1 + \deg(v) \log n)$  time. For extra credit, get the bound down to  $O(n + m)$ .*

*The interval-graph coloring algorithm colors the vertices in left-endpoint order, which is the reverse of a perfect elimination ordering. Therefore, to generalize it to chordal graphs, you should work through a perfect elimination ordering in reverse order.*

*When you reach a vertex  $v$ , color it with the smallest color that isn't in use by any neighbor to its right. To figure this out, sort these colors in  $O(1 + \deg(v) \log n)$  time.*

*To do it in  $O(1 + \deg(v))$  time, use the trick of Step 8 on page 90 of Golumbic. Initially create boolean array  $A[1..n]$ . When you reach  $v$ , for each neighbor of  $v$ , if the neighbor has color  $i$ , mark  $A[i]$ . Then traverse  $A[]$  until you find an unmarked index and assign that index as the color of the current vertex. Then for each neighbor, if the neighbor has color  $i$ , unmark  $A[i]$ , leaving  $A$  initialized and ready for the next vertex.*

4. The inductive proof in Golumbic that there exists a clique tree can be turned into an inductive algorithm on a perfect elimination ordering. The following picture depicts a perfect elimination ordering. Draw a picture of the clique tree after each modification of the clique tree in the induction. Label the nodes of the tree with the set of vertices in the corresponding clique.



*Solution:*

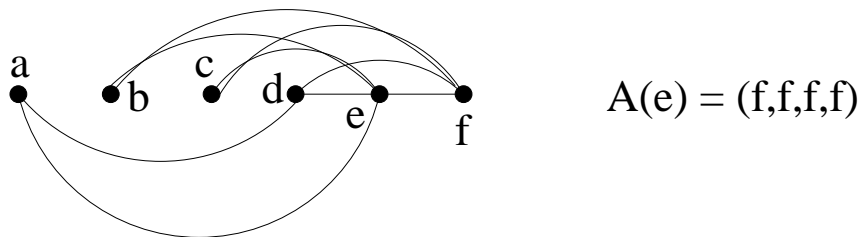
$$\{f\} \quad \{e,f\} \quad \{d,f\} - \{e,f\} \quad \{c,d,f\} - \{e,f\}$$

$$\begin{array}{ccc} \{c,d,f\} - \{e,f\} & & \{a,c,d\} - \{c,d,f\} - \{e,f\} \\ | & & | \\ \{b,c,d\} & & \{b,c,d\} \end{array}$$

*On the last step,  $\{a, c, d\}$  could have been joined to  $\{b, c, d\}$  instead of  $\{c, d, f\}$ .*

5. Checking whether an ordering of vertices is a perfect elimination ordering.
- (a) Show that in the algorithm for doing this check, by the time you reach a vertex  $u$ , a vertex  $v$  can appear multiple times in the list  $A(u)$  of vertices that  $u$  is supposed to be adjacent to.

*A solution:*



- (b) How do we get away with claiming that the running time is  $O(n + m)$  despite the possibility of all these multiple copies?

*The cost of inserting them all was charged to edges out of vertices that were earlier in the ordering that caused them to be inserted. Looking at them a second time doesn't affect the time bound.*

6. The Golumbic handout claims that Lex-BFS takes  $O(n + m)$  time. A tricky point is maintaining the vertices in sorted lexicographic order each time you assign a numerical social rank to a new vertex  $v$ .

Justify that this can be accomplished  $O(1 + \text{deg}(v))$  time, from which the bound will follow. I am looking primarily for a careful description of your data structures and a brief justification of why they make this bound possible.

*Keep a list of groups of nodes in ascending order of social status. Within each group, the nodes have equal status, and they are marked with pointers to the header of their group. Each group is implemented with a doubly-linked list.*

*When you assign a rank to a new node  $x$ , go through its neighbor list, asking the neighbor which group it's in, yank it out of the group (it's a doubly-linked list), and put it in a new group that's the successor of the old group in the list of groups. This requires creating a new successor group the first time a node is yanked from it, and inserting this group as the old group's successor.*

*This takes  $O(1 + \text{deg}(x))$  time, for a total of  $O(n + m)$  for the whole graph.*

7. By how much does Sleator and Tarjan's use of linking and cutting trees speed up Dinic's algorithm if all edges have capacity 1?

*It actually slows it down by a factor of  $\Theta(\log n)$ . Every time you augment along a path in the level graph, all edges of the path disappear. When you retreat from an edge it disappears.*

*When you advance along an edge, it is destined either to disappear during an augmentation or to disappear during a retreat. The time spent per visited edge is  $O(1)$ , but it would be  $O(\log n)$  if you used the linking/cutting trees to "help out."*

8. **Extra Credit** What's wrong with this "proof" that  $P = NP$ ?

Attached is an example of a Sudoku puzzle. We could use a computer to solve it by generating a graph where each square is a vertex, and inserting edges to make each column a clique, each row a clique, and each  $3 \times 3$  box a clique. The problem is now the problem of finding a minimum coloring of the graph. Since this takes 9 colors and the size of a maximum clique is 9, the graph is perfect.

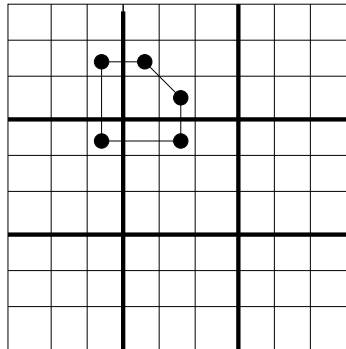
Sudoku puzzles take  $O(1)$  time to solve since their size is  $O(1)$ . To analyze sudoku solvers asymptotically, let's consider the general problem of an  $n^2 \times n^2$  sudoku, where there are  $n^2$  boxes, each of size  $n \times n$ . Generating our graph, we generate a clique of size  $n^2$  for each row, each column, and each box. The object is to color the squares with  $n^2$  colors, which is the size of the maximum clique, and once again, the graph is perfect.

**Proof:** Somebody showed that solving this general Sudoku problem is NP-complete. However, some other people showed that finding a minimum coloring of any perfect takes polynomial time. Therefore,  $P = NP$ . Q.E.D.

Unfortunately, the proof has at least two major flaws. See if you can spot them.

**Answer:** 1. Just because coloring is polynomial doesn't mean that completing an already-started coloring is polynomial. 2. It doesn't follow that the sudoku graph is perfect, since the proof doesn't show that the max-clique/min-coloring property also holds for all induced subgraphs.

The above arguments suffice, because they show that the proof is incomplete. In fact, the claim that the sudoku graph is perfect is wrong, which can be demonstrated with an odd hole:



As for completing a partial coloring on a perfect graph with a minimum number of colors, the partial coloring defeats the well-known polynomial algorithm that minimally colors a perfect graph from scratch.

That doesn't mean that there isn't one. So let me give you another phoney proof, that it is NP-hard. List coloring is known to be NP-hard on perfect graphs. This is the problem of coloring a graph with a minimum number of colors, where each vertex has a list of acceptable colors for it. Completing a coloring is the list-coloring problem where the already-colored nodes have lists of size 1, and the uncolored ones have a list of colors 1 through  $n$ . Therefore, completing a coloring of a perfect graph with a minimum number of colors is NP-hard.

What's wrong with this one?