

Math 461 Homework #7 Summary

Kurt Luoto

November 30, 2007

1 General comments

1.1 Points

The point assignment for problems in this assignment was: Problem 1 was worth 10 points total: part (a) was 2 points; part (b) was 3 points; part (c) was 5 points. The rest of the problems (2–4) were 5 points each.

1.2 Induction

Mathematical induction is an indispensable tool which should be in every mathematician's toolbox. That said, induction should *not* be the first tool you reach for when solving a problem, at least for the kinds of problems you will typically encounter in this course. True, there are problems for which induction is the only reasonable approach to proof, but for many problems there are alternative approaches, and when there are, usually the alternatives will be easier, shorter, and less prone to error.

For example, none of the problems in this problem set really require induction. Most people who attempted to use induction in their proofs for Problem 1 lost points. Please see discussion of Problem 1 below!

So when you set about proving any problem using induction, take a minute first to ask yourself whether there isn't some simpler way to do it.

1.3 Use those theorems, lemmas, and corollaries!

Several people spent several lines (or pages) effectively re-proving results that have already been proved in lecture and/or in the text. For example, many

people re-proved Theorem 9.4 of the text, the fact that the number of odd-degree vertices of any graph must be odd. Yes, it is a good thing to know the concepts so well that you can prove things from first principles. But for a homework write-up — and especially for an exam problem — it is quite acceptable, not to mention shorter, less error-prone, and less time-consuming to simply refer to the theorem and use it. (It is also easier on us graders!) This goes for any theorem, lemma, corollary, or proposition which has been proved in lecture or in the body of the text which has been covered in the course so far. Just be sure to make clear which theorem you are using, either by number or name or some other clear description.

2 Specific comments

2.1 Problem 1

Several students used arguments based on Prüfer codes for parts (a) and (b). The only problem is that parts (a) and (b) are asking to prove things about unlabeled trees, while Prüfer codes pertain only to labeled trees. One can still leverage facts about Prüfer codes by simply applying a labeling. For example, say something like, “Assume the tree has n vertices. Now label the vertices arbitrarily with labels $\{1, 2, \dots, n\}$, and consider the Prüfer code of the resulting labeled tree. ...” Any facts you can then derive from the Prüfer code that do not depend on how the labeling was performed (such as number of vertices of degree 3) must hold for the original tree. But you do have to make explicit the step of labeling the tree.

Part (a) is pretty much a direct consequence of Theorem 9.4, and should require no more than a sentence or two. (Hence why this part was only worth 2 points.) Using induction to re-prove this is way more complicated than necessary.

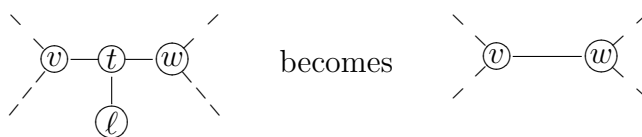
By far the simplest way to prove part (b) is to use vertex degree formulas: If x is the number of degree 1 vertices and y is the number of degree 3 vertices, then you have two equations in two unknowns: $x + y = n$, and $x + 3y = 2(n - 1)$. However an unfortunately large number of people chose to use an inductive argument.

One common inductive error was one which I warned against in an earlier commentary (HW set #5, if I recall correctly); apparently the warning needs repeating. Specifically, many students in their inductive step attempted to

argue that you begin with some tree (of the specified type), with, say, $2k$ vertices, and then add two vertices to it somehow in order to build a tree of the specified type with $2(k + 1)$ vertices. Sometimes they gave some idea of how the vertices were being added, e.g. attaching two new leaves to an existing leaf, but other times they were completely vague. The big problem here is that you must demonstrate you can construct *all* trees of the specified type on $2(k + 1)$ vertices from some tree of the specified type with $2k$ vertices in this fashion. Unfortunately, everyone who took this approach failed to show this.

The rule of thumb here is that it is bad luck in situations such as this to try to build up all of your $n + 1$ case instances from your n case instances. The easier and wiser way is to instead *start* with a generic $n + 1$ case instance and show that you can *reduce* it to a n case instance somehow, typically by *deleting* vertices and edges.

Some students who used induction avoided the above big error, only to stumble over a smaller one. In particular, they argued that you reduce the case of a specified tree on $2(k + 1)$ vertices by deleting two leaves which share a common neighbor. It is true that in a tree with only vertices of degree 3 or 1 such a pair of leaves always exists, but it is not so obvious that it can simply be assumed; it must be proved, and most failed to do so. A more generic reduction is the following: Assuming that there are more than two vertices, a leaf ℓ must be neighbor to a vertex t of degree 3. Vertex t then has neighbors, say, ℓ, v , and w . Reduce by removing ℓ and t and replacing an edge between v and w .



Quite a few students missed the factor $\binom{n}{(n-2)/2}$ in the answer to part (c). I docked 2 points for this.

2.2 Problem 2

Some people had a good start on a proof, passing from labeled trees to Prüfer codes, specifying that one is interested in codes that have $(k - 1)$ occurrences of the vertex n , but then got lost by subsequently mixing up references to

codes with references to neighboring vertices of n in the tree, rather than talking about how to fill in the rest of the Prüfer code. Be careful not to mix apples and oranges!

Terminology tip: Deleting some number of letters (vertices) from a Prüfer code does not leave you with “a smaller Prüfer code” for the tree, nor does it generally leave you with the Prüfer code of some smaller tree. Also, the Prüfer code is a single sequence of letters (vertices), *not* a sequence of pairs of vertices, nor two rows.

2.3 Problem 3

The hint given in the problem statement says, “notice that every pair $\{i, j\} \subset V$ appears as an edge in exactly the same number of trees [as every other pair].” Ideally, one should prove this, but I did not require it. However, I did require that one mention it. I also required that one mention that any edge appears at most once in any given tree, a fact which is used at least implicitly in just about every solution.

Some students resorted to a probabilistic argument. This no doubt is very natural for anyone who has already taken a course in probability or statistics. But we are proving an exact formula here, and the body of theorems developed so far in the course do not include probabilistic methods for proving exact formulas, so probabilistic arguments require additional proof (which, unfortunately, was not provided).

A few students attempted an argument based on Prüfer codes, but it is generally quite difficult to make arguments about the presence of a given edge based on the Prüfer code of a labeled tree. None who tried it were able to make a convincing argument. (So a tip: Prüfer codes are great when dealing with vertex degrees of a labeled tree, but not so easy for dealing with particular edges.) There is one approach using Prüfer codes: Instead of working with the edge $\{1, 2\}$, work with the edge $\{n, n - 1\}$, relying on the fact mentioned in the hint. It turns out that a labeled tree on vertex set $[n]$ contains the edge $\{n, n - 1\}$ if and only if its Prüfer code ends in either n or $n - 1$, from which the desired result easily follows. However, to prove this fact this requires a bit of non-obvious argument, and on the whole is more complex than the argument in the professor’s solution sheet.