CPSC 031 — Mathematics Review for CPSC 413

Exercise #1 — Mathematical Induction (With Hints)

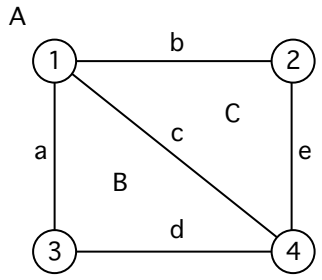
September, 2000

The problems given here are the same as the ones on the original problem set, "Exercise 1," except that they include some additional information to help you get started.

1. A *planar graph* is a graph that can be drawn (on a planar surface, like a blackboard) so that no two edges intersect except at endpoints. A graph is *connected* if there is a path (along edges) between any pair of distinct vertices in the graph.

   Prove *Euler's formula*: If G is a nonempty connected planar graph with v vertices, e edges, and f faces, then $v + f = e + 2$.

   For example, the connected planar graph shown below has $v = 4$ vertices, numbered 1–4. It has $e = 5$ edges, labelled a–e. It has $f = 3$ faces (including an exterior face) labelled A–C. In this case, $v + f = 7 = e + 2$, as expected.



   Getting Started: When there is more than one parameter to consider, as in this example, then it might not be clear what to induct on. This problem can likely be answered in several different ways, and some approaches are probably more difficult than others. It turns out that if you use induction on the number of *faces,* then the solution is reasonably straightforward, except for one more little trick: You must either use a fact about directed acyclic graphs ("trees,") without proof, or you must use a *second* induction to prove it.

   Since there will be at least one face associated with any graph, the case $f = 1$ is the first case to consider (in a "basis") in a proof by induction on the number of faces. In this case, you wish to prove that if G is a nonempty connected graph with v vertices, e edges, and one face, then $v + 1 = e + 2$ — that is, $v = e + 1$. However, the fact that G is connected and has only one face implies that G is a "directed acyclic graph" — that is, G is a tree.

Now, to complete the "basis" for your induction on f, you must prove that every nonempty tree has exactly one more vertex than it has edges. This is easy to prove, either using induction on the number of edges in the tree, or using induction on the number of vertices.

Once you've done that, you must complete the "inductive step." In order to do this, let f be greater than or equal to one, suppose that $v + f = e + 2$ whenever G is a nonempty connected planar graph with v vertices, f faces and e edges, and consider a nonempty connected acyclic graph with $f + 1$ faces.

Look for a way to reduce the number of faces by one, by reducing the number of edges by one, while leaving the number of vertices unchanged.

If you can do this, then you can solve the problem. (Why?)

2. A node of a binary tree is an *internal node* if it has at least one child, and it is a *leaf* otherwise. A binary tree is a *complete* binary tree if every internal node has exactly two children.

   Prove that every nonempty complete binary tree has exactly one more leaf than it has internal nodes.

   Getting Started: Try to use induction on the number of internal nodes in the tree. The basis will be easy — there is only one (shape for a) nonempty tree that has zero internal nodes: It consists only of the root, and this is a leaf.

   Now, think about a way to take a complete binary tree that has at least one internal node, and modify it, by deleting vertices, so that you reduce both the number of leaves and the number of internal nodes by exactly one (possibly turning an existing "internal node" into a leaf, in the process).

3. Lines in the plane are in *general position* if no two lines are parallel and no three (or more) lines intersect at a common point. Find the number $R(n)$ of regions created by n lines in general position, and prove your answer.

   Getting Started: Note that if there are zero lines then there is one region — the entire plane — so that $R(0) = 1$. If there is one line then there are two regions, one on each "side" of the line, so that $R(1) = 2$. If there are two lines then, since they are not parallel, they produce four regions (draw the picture if this is not clear), so that $R(2) = 4$.

   Drawing similar pictures you should be able to confirm that $R(3) = 7$ and $R(4) = 11$.

   Now, look for a pattern in the sequence $R(0), R(1), R(2), \ldots$. If you can't find one, consider sums of consecutive elements; products of consecutive elements; differences between consecutive elements; and so on, until you can find a pattern in one of *these* sequences.

   If you do this correctly then you should be able to *find* a pattern and use this to guess what $R(n)$ is, in general.

   Next, you will need to prove that your guess is correct. For the inductive step, you will need to consider what happens when you add a new line to a drawing that has n lines in it already. If the lines are "in general position," how many of the existing regions does the new line travel through (noting that something happens whenever the new line intersects one of the old ones)? What happens to the existing regions when the new line is added?

4. Consider the algorithm, *Convert-to-Binary*, that is shown below.

Algorithm *Convert-to-Binary*

Input: A positive integer n.

Output: An array B of bits corresponding to the binary representation of n.

```
1.  begin
2.    t := n
3.    k := 0
4.    while t > 0 do
5.      k := k + 1
6.      B[k] := t mod 2
7.      t := t div 2
8.    end while
9.  end
```

Note that (a **mod** b) and (a **div** b) are, respectively, the remainder and quotient you would get when you divide a positive integer a by a positive integer b, so that (a **mod** b) is always between 0 and $b-1$, inclusive, and so that it is always true that

$$a = (a \textbf{ div } b) \times b + (a \textbf{ mod } b).$$

Prove that when the above algorithm terminates, the binary representation of the integer n is stored in the array B.

Getting Started: The specification of the input and output is a little bit unclear — it isn't clear in which order the bits of the binary representation are to be stored. Start by tracing execution of the algorithm on one or two small examples — say, for example, 4 (whose binary representation is 100_2) and 6 (whose binary representation is 110_2). This should allow you to complete the specification of the algorithm: On termination,

- k is the number of bits in the binary representation of n.
- Values $B[1], B[2], \ldots, B[k]$ are defined and are each equal to either zero or one, while $B[i]$ is undefined if $i > k$.
- $n = \sum_{i=1}^{k} B[i]2^{i-1}$.

You *might* be able to prove this by induction on n, but it might be difficult. It's often helpful, when you are trying to prove correctness of a program, to break it into pieces, and identify (and prove) properties of the state of the program's variables, when each piece of the program has been completed.

For this program, let's split it up into a piece containing the first two assignment statements, and the **while** loop that follow them. Clearly, after the first two statements have been executed, $t = n$, $k = 0$, and $B[i]$ has not been defined, yet, for any positive integer i.

In order to prove something about the effect of the loop, it will be useful to establish a "loop invariant" — a useful property of the state of the program that is true just before each

execution of the loop body, and also just after the final execution (that is, each time the loop's test is executed).

To discover a useful "loop invariant," trace execution of the program on a small (but nontrivial) value of n — say, $n = 100$. Write down the value of the program's variables — including the value of $B[i]$ for each integer i such that $B[i]$ has been defined — at the end of each execution of the loop body (and, to start, just before the loop body is first executed). You should find that the value of k is related to the number of times the loop body is executed (call this number i); t is a function of n and i; and some number (also depending on i) of the bits of the binary representation of n have been written into B.

In particular, you should discover, among other things, that

$$n = 2^k t + \sum_{j=1}^{k} B[j]2^{j-1}.$$

Now, you should be able to use induction on the number of executions of the loop body to prove that your "loop invariant" really is correct. That is, it is always satisfied, just before the loop body is executed, and just after the loop terminates.

Once you've done this, you should be able to argue that the program is correct, by noting that the loop invariant is satisfied, and $t \leq 0$ (since the loop test failed) when the program terminates.

Note as well that the program *does* terminate: You should be able to discover a bound on the number of executions of the loop body that is a (slow growing) function of n, as part of this process.

5. Find a positive constant c such that the following is true for every positive integer n and prove your result.

$$\sum_{i=1}^{n} i^2 \leq cn^3.$$

Getting Started: There are at least two ways to do this. One way is to compare the values of $\sum_{i=1}^{n} i^2$ and n^3 for the first few values of n (say, $n = 1, 2, 3, \dots$) in order to discover a value of the constant c that would be a reasonable "guess." Then, replace the unknown constant by your guess to get an inequality that only depends on n, and try to prove this inequality using induction on n.

Another method, that eliminates the guesswork, is to write down the proof by induction on n, leaving c as an unknown symbolic constant. Keep track of all the conditions that c must satisfy in order for the proof to be correct. For example, you will need to assume that $c \geq 1$ in order for the inequality to be satisfied when $n = 1$. Finally, select a single constant c that satisfies all your conditions, and observe that you would have a correct proof by induction if you had initially guessed that value for c..

6. Once again, consider the above algorithm *Convert-to-Binary*. Find positive constants c and d such that, given any positive integer n as input, the number of statements executed by the algorithm is at most $c \log_2 n + d$. Prove that your choices are correct.

Getting Started: If you've done the rest, this should be reasonably straightforward!