CS3933 Assignment 11
Summer 2007

Due: Tuesday, August 7, in class or by 10:00am in the box on E-level.

- Read chapter 8 a third time.

- I will try to explain the bonus question that I included in assignment 10, but failed to explain adequately.

  You have two bottles that you are allowed to break, and a ladder that you can climb. Your client want to know from which rungs on the ladder a bottle can be dropped and not break, and from which ones it will break. There must be an integer k such that for any rung above k, a dropped bottle will break, but for rung k and below it will not break. Suppose that you only want to drop bottles no more than n times. What is the largest value $g(n)$ such that if $k \leq g(n)$, you can determine k without dropping bottles more than n times?

  This can be solved using a dynamic program.

  I will give you the best algorithm for the case $n = 2$. Drop the first bottle from rung 2. If it breaks, then drop the second bottle from rung 1. If the second bottle breaks, $k = 0$; if the second bottle does not break then $k = 1$. Otherwise the first bottle did not break, in which case you drop the second bottle from rung 3. Now if the second bottle breaks, $k = 2$. If the second bottle does not break, then $k \leq 3$, and you cannot determine k without dropping bottles again, and you are not allowing yourself to do that. Thus $g(2) = 3$.

  Your problem to solve for me, is to determine the first 10 values of $g(n)$, for $n = 0, 1, \ldots, 9$. The main thing to do is to find a recurrence relation for g. You do have to explain why the recurrence is correct, by specifying from which rungs the bottles should be dropped.

- Do question 2 on page 505. Hint: transform (reduce is the text's word) from either Set Cover or Vertex Cover, to prove NP-hardness.

  To solve a dynamic programming problem, you must define a mathematical function *in terms of the inputs*, then give a recurrence relation that it must satisfy (including a base case), and explain how you can calculate the function iteratively.

  To prove a problem P is NP-complete, you must

1. Show that the problem P is in NP.
2. Choose some other known NP-complete problem Q.
3. Reduce Q to P, ie. show that $Q \leq_P P$.