CSI 4105 Design and Analysis of Algorithms II — Winter 2011
Computer Science — University of Ottawa

Homework Assignment #2 (100 points, weight 5%)
Due: Friday March 18, by 3:30 p.m. (drop under my office door)

---

1. (25 points) Chapter 8 - Exercise 22 page 517 (on independent set).

Suppose that someone gives you a black-box algorithm $\mathcal{A}$ that takes an undirected graph $G = (V, E)$, and a number k, and behaves as follows.

- If G is not connected, it simply returns "G is not connected."
- If G is connected and has an independent set of size at least k, it returns "yes."
- If G is connected and does not have an independent set of size at least k, it returns "no."

Suppose that the algorithm $\mathcal{A}$ runs in time polynomial in the size of G and k.

Show how, using calls to $\mathcal{A}$, you could then solve the Independent Set Problem in polynomial time: Given an arbitrary undirected graph G, and a number k, does G contain an independent set of size at least k?

2. (25 points) Chapter 10, exercise 3, page 596 (Solving Hamiltonian Path Problem in time $O(2^n p(n))$).

3. Suppose we are given a directed graph $G = (V, E)$, with $V = \{v_1, v_2, \ldots, v_n\}$, and we want to decide whether G has a Hamiltonian path from v_1 to v_n. (That is, is there a path in G that goes from v_1 to v_n, passing through every other vertex exactly once?)

Since the Hamiltonian Path Problem is NP-complete, we do not expect that there is a polynomial-time solution for this problem. However, this does not mean that all nonpolynomial-time algorithms are equally "bad." For example, here's the simplest brute-force approach: For each permutation of the vertices, see if it forms a Hamiltonian path from v_1 to v_n. This takes time roughly proportional to $n!$, which is about 3×10^{27} when $n = 20$.

Show that the Hamiltonian Path Problem can in fact be solved in time $O(2^n \cdot p(n))$, where $p(n)$ is a polynomial function of n. This is a much better algorithm for moderate values of n; for example, 2^n is only about a million when $n = 20$.

3. (50 points) Backtracking for Latin squares

   (a) A Latin square is an $n \times n$ array with entries in $\{1, 2, \ldots, n\}$ such that every symbol occurs exactly once in each row and in each column. The next example shows Latin squares of order 3:

1	2	3
3	1	2
2	3	1

1	2	3
2	3	1
3	1	2

   Write a backtracking algorithm to generate all Latin squares of order n. For larger values of n, please do not output the Latin squares, but simply how many are there. Check online at the encyclopedia of integer sequences for the correct number for verification: check `http://oeis.org/` and search for "latin squares". Compute the number of Latin squares for $n = 1, 2, 3, 4, 5, \ldots$, until a value of n that it is still computationally reasonable to solve the problem.

   (b) Use Knuth's method for estimating the size of a backtracking tree, but use the variation in which you assign weights to the nodes. Assign weights accordingly and show how to estimate the number of Latin squares for the next value of n by applying Knuth's method on a sufficient number of probes. Please, show how the estimation progresses as the number of probes grow.

Note: this assignment does not include questions on PSPACE, but the next midterm test will.