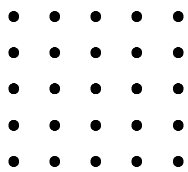
CPSC-320 Intermediate Algorithm Design and Analysis Winter 2000

Practice Homework # 1

These practice problems will be discussed at the tutorials during the week of September 11. If at all possible, work on them over the weekend, and identify before the tutorials what seems confusing or difficult. Solutions will be posted on Friday, September 15. Good luck!

1. Consider an $n \times n$ square of dots, such as the following.

   How many *tilted subsquares* are hidden in this square? By a tilted subsquare I mean a square whose perimeter is formed by dots lying on *diagonals* of the given $n \times n$ square. (Note that a tilted square with i dots per side won't have a total of i^2 dots. Also, titled subsquares are at a 45 degree angle to the grid axes.)

   Suggestion: Express the answer as a summation and then find a closed form expression for the summation, as a function of n.

2. Consider again the golf balls problem described in the first lecture. The algorithm presented in class for 12 golf balls uses exactly three weighings to determine which is the heavy ball. There is an alternative algorithm that never uses more than three weighings, but "when lucky" uses less than three weighings. This algorithm is a beautiful illustration of a class of algorithms called *randomized* or *probabilistic* algorithms. (Quicksort is another example of such an algorithm, as we will see later in the course.) In this problem, your task is to discover such an algorithm, and also to consider randomized algorithms for the golf ball problem with 13 balls.

   The problem will serve to introduce some notation about randomized algorithms that will be useful throughout the course. We illustrate this notation via the simpler problem with 4 golf balls.

```
1.   Choose two balls at random.
2.   Place one chosen ball on each side of the balance.
3.   If one of the chosen balls is the heavy ball, we are done.
4.   Otherwise,
5.       place one of the remaining two balls on each side of the balance.
6.       The heavy ball is determined from this weighing.
```

   This algorithm is *randomized* because the choice of which balls to place on the balance is made at random in step 1. The probability of "being lucky," i.e. that the heavy ball is one of the two balls chosen in step 1, is 1/2. In this case, only one weighing is needed to solve the problem. In the unlucky case, two weighings are needed. The *expected number* of weighings is therefore $1/2 \times 1 + 1/2 \times 2 = 1.5$.

(a) Design an algorithm for the 12 golf ball problem with an expected number of weighings equal to 2.5. Explain why the number of weighings is 2.5 for your algorithm.

(b) Consider the problem with 13 golf balls. Can you describe how to find the heavy ball using an expected number of weighings that is *less than* 2.5?

3. Graph colouring problems have a long history and are relevant to many practical computational problems. Before describing the graph colouring problem, we start with an application problem, for motivation.

   Suppose that you want to schedule a set of courses into a limited number, say T, of time slots. There are conflicts between some pairs of courses, for example if one student wants to take both, in which case those two courses cannot be assigned to the same time slot. A basic task would be as follows: Given a number of timeslots, a list of courses, and a list of pairs of conflicting courses, determine if the courses can be assigned to time slots so as to ensure that no two conflicting courses get the same time slot. (A more complicated-sounding task would be to find an assignment of classes to time-slots that "minimizes" the number of conflicts. In general a good strategy when faced with the task of designing an algorithm for such a tricky problem is to first simplify the problem as much as possible and try to find an algorithm for the simpler problem, then go back to the more complicated problem later.)

   If the courses are represented as nodes of a graph, and an (undirected) edge is placed between each pair of conflicting nodes, then the task above is equivalent to determining if one can colour the nodes of the graph with at most T colours, where each node is assigned exactly one colour, such that no two adjacent nodes (i.e. nodes that are connected by an edge) are assigned the same colour. (Can you see the relationship between graph colouring and course scheduling?)

   (a) Consider the case where $T = 3$; this is called the *graph 3-colouring problem*. Design an algorithm that takes as input an undirected graph and determines if the graph is 3-colourable, that is, every node of the the graph can be assigned one of three colours (say either red, blue, or green), such that no two adjacent nodes have the same colour. Keep your algorithm as simple as possible. What can you say about the number of steps (i.e. the "running time") of your algorithm? You may define a "step" of your algorithm in whatever way seems most reasonable to you.

   (Hint: the simplest algorithm I can think of requires a number of steps that grows roughly as 3^n, where n is the number of nodes of the graph.)

   (b) Assume that one step of your algorithm takes only 1 nanosecond on your post-state-of-the-art new PC. How long will it take your algorithm to run on a graph with 10 nodes? 50 nodes? 100 nodes?