Discrete Math in CS	**Problem Set 6**
CS 280 Fall 2000	**Due October 27, 2000**

Reading: Rosen, Sections 1.4 – 1.6 and 4.1 – 4.3.

---

(1) Suppose you're monitoring the performance of a link in a communication network, which two *sources* are sharing in order to transmit data. The first source has a sequence of p packets $a_1, a_2, \ldots, a_p$ that it wishes to send, in this order, across the link. The second source has a sequence of q packets $b_1, b_2, \ldots, b_q$ that it wishes to send, in this order, across the link.

The link *multiplexes* these two streams of packets: it takes the two sequences of packets, and interleaves them in an arbitrary fashion in order to send them sequentially across the link. We call the resulting interleaved, ordered sequence of packets a *packet trace*.

How many distinct packets traces are possible, given that the two sources are sending p and q packets respectively? Write a formula for this quantity as a function of p and q, and give a brief explanation for why your formula is correct.

Example: If $p = 2$ and $q = 1$, then the sequences a_1, a_2, b_1 and a_1, b_1, a_2 are two of the possible packet traces. Note that the sequence a_2, b_1, a_1 is not a packet trace; it cannot be obtained by interleaving the two original packet sequences, since a_1 comes before a_2 in one of the original sequences.

(2) Let's consider a variation on the previous question: now there are three sources, again sharing a common link. The first source wishes to send a sequence of p packets, the second a sequence of q packets, and the third a sequence of r packets. A packet trace, by analogy with the previous question, is an interleaving of these sequences.

Write a formula for the number of distinct packet traces that are possible, as a function of p, q, and r; and give a brief explanation for why your formula is correct.

---

(3) Suppose you're monitoring the performance of a distributed system with n distinct processes. Each process can be in one of three possible states: *running*, *waiting*, or *done*. (It is possible for multiple — even all — processes to be runnning simultaneously.)

We say that a *configuration* of the system is a specification of one of these three states for each of the n processes. We say that a configuration is *blocked* if at least one process is waiting, and no process is running. We say that a configuration is *non-blocked* if it is not blocked.

How many distinct non-blocked configurations are there? Write a formula for this quantity as a function of n, and give a brief explanation for why your formula is correct.

---

(4) *(Based on a problem proposed by Joe Polastre.)* The National Hockey League has expanded again (let's suppose), and they've added a new team in the S.F. Bay Area — the Mountain View Venture Capitalists. Unfortunately, the VC's (as their fans affectionately call them) are desperately low on hockey talent, if rather well-funded. In their first season, consisting of 84 games, they lose every single game.

Suppose that in each game the VC's play, both they and their opposing team score between 0 and 10 goals — i.e. each team ends with a final score that is some number between 0 and 10 inclusive. Prove that there are at least two games during the season that the VC's lose by exactly the same final score. (Note: by the final score of a game, we simply mean the final scores of the two teams; e.g. if the VC's score 3 goals and their opponents score 5, we write the final score of the game as 5-3.)

(5) Your friends from out West with the interest in security are back. They've moved beyond trying to implement RSA, and are concerned with designing something they call a *fail-safe system.*

By a fail-safe system, they mean a computer system that is monitored by a set of n independent software agents, who are in charge of making sure the system doesn't "misbehave." We assume n is an even number. It should have the following two properties:

(A) Any subset of the agents of size greater than $n/2$ should be able to collectively shut down the system.

(B) No subset of the agents of size at most $n/2$ should be able to collectively shut down the system.

Thus, requirement (A) is a *positive requirement*: if enough agents believe the system should be shut down, they have the ability to do this. Requirement (B) is a *negative requirement*: it prevents small subsets of agents from conspiring to shut down the system on their own. We say the system is *secure* if it satisfies both requirements (A) and (B).

Here's how your friends propose to design a system like this. They have the n agents, labeled $a_1, a_2, \ldots, a_n$. For some number m, they prepare a set of m secret *passwords* $p_1, p_2, \ldots, p_m$. They give each agent a_i a subset X_i of the passwords; that is, each agent knows some of the secret passwords but not others. Now, suppose a subset S of the agents gets together and tries to shut down the system. The system will ask them to provide each of the m passwords in turn — so in order to succeed in shutting the system, it is required that for each password p_i, there is some agent a_j in the set S who knows p_i; that is, some a_j for which $p_i \in X_j$.

The idea is to carefully assign passwords to agents in such a way that for *any* subset S of more than half the agents, each password is known by some agent in S; but for *any* subset S' of at most half the agents, the agents in S' do not collectively know all the passwords. In this way, requirements (A) and (B) will both be satisfied.

Version 1.0 of their system used just two agents and two passwords. Agent a_1 was given password p_1, and agent a_2 was given password p_2. This way, neither agent on its own could provide both passwords to shut down the system; but acting together they could.

But Version 1.0 is a very simple implementation of a fail-safe system; after all, the only set of size greater than $n/2$ is in fact the full set of $n = 2$ agents. So they're currently designing Version 2.0, which uses 6 agents, and they have a prototype that uses 18 passwords. They're about to start going into the details of how it works — in particular, which agents are assigned which passwords — but in fact you can stop them right here; their prototype must be insecure, as the following statement shows.

Prove that there is no fail-safe system with 6 agents and 18 passwords that satisfies both requirements (A) and (B).