CS 4820, Summer 2010 Out: August 06. **Due: Tuesday, August 10, 8:30am**

Homework 9

Reading: Sections 8.1–8.7 from the textbook.

Problem 1. Suppose you're acting as a consultant for the Port Authority of a small Pacific Rim nation. They're currently doing a multi-billion dollar business per year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Handling hazardous materials adds additional complexity to what is, for them, an already complicated task. Suppose a convoy of ships arrives in the morning and delivers a total of n canisters (denoted by set C, hence $|C| = n$), each containing a different kind of hazardous material. Standing on the dock is a set of m trucks (denoted by T, hence $|T| = m$), each of which can hold up to ℓ containers.

Here are two related problems, which arise from different types of constraints that might be placed on the handling of hazardous materials. For each of the two problems, give one of the following two answers:

- A polynomial-time algorithm to solve it; or
- A proof that it is NP-complete.

(a) For each canister i, there is a specified subset T_i of the trucks in which it may be safely carried. Is there a way to load all n canisters into the m trucks so that no truck is overloaded, and each container goes in a truck that is allowed to carry it?

(b) In this different version of the problem, any canister can be placed in any truck; however, there are certain pairs of canisters that cannot be placed together in the same truck. (The chemicals they contain may react explosively if brought into contact.) Is there a way to load all n canisters into the m trucks so that no truck is overloaded, and no two canisters are placed in the same truck when they are not supposed to be?

Problem 2. Suppose you've just compiled the final version of your sweet new computer game, "Ninja Monkeys vs. Pirate Penguins: The Final Encounter". Despite its inherent awesomeness, the number of people who have actually downloaded your game is disappointingly small. Those who have played it give rave reviews, but most people still haven't heard of it.

To increase the game's visibility, you decide to do a little viral marketing. Your goal is to make sure everyone on campus has either tried the game themselves, or at least is friends with someone else who has. Obviously you could do this by convincing everyone on campus to try the game, but this would take forever. The question is, can you meet your goal while only taking the time to convince k people to try NMvPP (we'll go with the slightly optimistic assumption that anyone can be convinced to play the game eventually)?

We can model the Viral Marketing problem more formally as follows. Given a graph $G = (V, E)$ (in which nodes represent people and edges connect friends) and a positive integer k, we would like to know whether there a set of nodes $S \subseteq V$ containing at most k nodes such that every node $x \in V$ is either itself in S or is adjacent to at least one node in S (or both). Such a set S will be called a Viral Set.

(a) Give an example of a graph G and a positive integer k such that G has a Viral Set of size k, but does not have an Vertex Cover of size k.

(b) Prove that the Viral Marketing problem is NP-Complete. Note that the input to the Viral-Set problem is $\langle G, k\rangle$ and the input is a "yes" instance if there is a Viral-Set of size at most k and a "no" instance otherwise.

Problem 3. The mapping of genomes involves a large array of difficult computational problems. At the most basic level, each of an organism's chromosomes can be viewed as an extremely long string (generally containing millions of symbols) over the four letter alphabet $\{A, C, G, T\}$). One family of approaches to

genome mapping is to generate a large number of short, overlapping snippets from a chromosome, and then to infer the full long string representing the chromosome from this set of overlapping substrings.

While we won't go into these string assembly problems in full details, here's a simplified problem that suggests some of the computational difficulty one encounters in this area. Suppose, we have a set $S = \{s_1, s_2, \ldots, s_n\}$ of short DNA strings over a q-letter alphabet Σ (that is $|\Sigma| = q$); and each string s_i has length 2ℓ for some $\ell \geq 1$. We also have a library of additional strings $T = \{t_1, t_2, \ldots, t_m\}$ over the same alphabet; each of these also has length 2ℓ. In trying to assess whether the string s_b might come directly after the string s_a in chromosome, we will look to see whether the library T contains a string t_k so that the first ℓ symbols in t_k are equal to the last ℓ symbols in s_a, and the last ℓ symbols in t_k are equal to the first ℓ symbols in s_b. If this is possible, we will say that t_k *corroborates* the pair (s_a, s_b). (In other words, t_k could be a snippet of DNA that straddled the region in which s_b directly followed s_a.)

Now, we would like to concatenate all the strings in S in some order, one after the other with no overlaps, so that each consecutive pair is corroborated by some string in the library T. That is, we would like to order the strings in S as $s_{i_1}, s_{i_2}, \ldots, s_{i_n}$, where $i_1, i_2, \ldots, i_n$ is a permutation of $\{1, 2, \ldots, n\}$, so that for each $j = 1, 2, \ldots, n-1$, there is a string t_k that corroborates the pair $(s_{i_j}, s_{i_{j+1}})$. (The same string t_k can be used for more than one consecutive pair in the concatenation.) If this is possible, we will say that the set S has a *perfect assembly* with respect to T.

Given set S and T, the *Perfect Assembly Problem* asks: Does S have a perfect assembly with respect to T? Prove that Perfect Assembly is NP-complete.

Example: Suppose the alphabet is $\{A, C, G, T\}$ (the four bases), the set $S = \{AG, TC, TA\}$, and the set $T = \{AC, CA, GC, GT\}$ (so each string has length $2\ell = 2$.) Than the answer to this instance of Perfect Assembly is yes: We can concatenate the three strings in S in the order $TCAGTA$ (so $s_{i_1} = s_2$, $s_{i_2} = s_1$, and $s_{i_3} = s_3$). In this order, the pair (s_{i_1}, s_{i_2}) is corroborated by the string CA in the library T, and the pair (s_{i_2}, s_{i_3}) is corroborated by the string GT in the library T.