15-451 Algorithms, Fall 2010

Homework # 6 **Due: Dec 2, 2010**

Please hand in each problem on a separate sheet and put your **name** your **andrew id** and your **recitation** (time or letter) at the top of each page. You will be handing each problem into a separate box in lecture, and we will then give homeworks back in recitation.
Remember: Homework is due at the begining of class. If you hand in your homework after class has started you are late. Please cite any sources you use (google, course notes, etc) as well any collaborators. As always each solution must be written in your own words. You must fully understand any solution you write down. You are required to typeset your homework.

Problems:

(50 pts) 1. **Approximately.** The minimum weight vertex cover problem is defined as follows:

Input: A graph G and weights $w_v > 0$ for each vertex v
Goal: Find a vertex cover C of minimum weight, where

$$Weight(C) = \sum_{v \in C} w_v \, .$$

In class you saw a 2-approximation algorithm for regular vertex cover (all vertices have weight 1). In this problem we will look at four different approximation algorithms for weighted min-vertex-cover with positive vertex weights. For each problem, prove the best **approximation ratio** guarantee that you can for the general problem as well as for the special case where all vertex weights are 1. We want you to provide upper and lower bounds. Note: if no approximation guarantee can be made, then a lower bound demonstrating this is sufficient.

A *Super Naive:* Consider all the edges in some order. If the edge $\{u, v\}$ being considered is not covered yet, pick whichever of u or v has less weight (for the case where weights are equal, assume an adversary picks which vertex is used).

B *Naive:* Consider all the edges in some order. If the edge $\{u, v\}$ being considered is not covered yet, pick *both* the vertices u and v.

C *Local Search:* Define two solutions $S \subseteq V$ and $S' \subseteq V$ to be neighbors if S can be obtained from S' by adding, deleting, or swapping a vertex. The local search moves are simple: Start with any solution $S \subseteq V$; if you are at some solution S, move to any neighboring solution S' that has strictly less weight. If you are at a *local optimum* – where all the neighbors have at least as much weight – output this local optimum. (Don't worry about the running time for this algorithm.)

D *LP Rounding:* The standard Vertex-Cover LP is the following: minimize $\sum w_v x_v$ subject to $x_u + x_v \geq 1$ for all edges $\{u, v\} \in E$, and $x \geq 0$. Given a fractional solution for this LP, define $V_\alpha = \{v \in V | x_v \geq \alpha\}$. What value of α ensures that V_α is a vertex cover? What approximation guarantee can you get for such an α?

(50 pts) 2. **RSA Mania** These problems will require you to perform computations with large numbers. This will be extremely difficult to perform by hand. You are welcome to use any program/programming language as an aid. We recommend using mathematica. Some of the problems reference a mathematica notebook. Specific values of public keys, encrypted messages etc... can all be found in this notebook. This file also contains an implementation of RSA. You will notice that the function to encrypt an ASCII string is slightly different from the function used to encrypt an integer, because we must first convert the string to an integer. We perform this conversion by interpreting an ASCII string as a base 256 number. Similarly, we have different functions to decrypt a message. One decrypts the message to an integer value and outputs that integer. The other interprets converts this integer to an ASCII string before outputing.

A Is there an element of order 6 in Z_{17}^*? If so, give one. If not, say why not.

B Harry Q. Bovik forgot exactly how to generate a public key. He randomly generated two very large prime numbers p_h, q_h and computed $N_h = p_h \times q_h$. Finally, he released the following public key: $(N_h, e_h, \phi(N_h))$. The values N_h, e_h, and $\phi(N_h)$ can be found in section B of the mathematica notebook . Your task is to find p_h and q_h. Briefly, explain how you accomplished your goal and why it worked.

C Harry Q. Bovik figures out his mistake and generates a new public key N'_h, e'_h. Harry observes that $(\forall 0 \leq x < N'_h)(N'_h - x)^2 \equiv x^2 \mod N'_h$. After toying around for a while Harry finds two numbers $1 \leq x_h, y_h < N'_h$ such that $x_h^2 \equiv y_h^2 \mod N'_h$ and $x_h \not\equiv \pm y_h \mod N'_h$. Excited, Harry decides to publish his findings. The specific numbers x_h, y_h can be found in section C of the mathematica notebook. Your task is to find p'_h and q'_h the prime factors of N'_h. Briefly explain how you accomplished your goal and why it works. Would your technique still work if $x_h \equiv N'_h - y_h \mod N'_h$?

D Three new employees at a company X have three different public RSA keys: $(N_1, e_1 = 3), (N_2, e_2 = 3), (N_3, e_3 = 3)$ respectively. Their boss just send each of them a encrypted message containing a company password so that they can access secret documents. Specifically, employee i recieved the encrypted message $enc_i = m^3 \mod N_i$. Specific values can be found in part D of the mathematica notebook. Your goal is to recover the original message m and give us the password contained inside. (Hint: The Chinese Remainder Theorem may come in handy. Note: that the original message the boss sent was an ASCII string *str*. It was encrypted with using *EncryptMessageString*$[str, N_i, e_i]$ for $i - 1, 2, 3$. This function views the ASCII string as a base 256 number m and then encrypted.)

E The public key (N_1, e_1) is owned by a bank that Bill Gates uses. Bill Gates has generously offered to give you 100 dollars. To authorize this he encrypts the following string *str*:

```
Transfer Money to CMU Student, Name: Bill Gates.  My authorization
code is *******. Please transfer the following ammount (dollars): d
```

str is encrpted to *encTrans* with the *EncryptMessageString*$[str, N_1, e_1]$ function. Once you present *encTrans* to the bank, the bank will decrypt the message and

check to make sure the authorization code is correct. Assuming the authentication code matches the bank parses *str* for the remaining string *s* after "(dollars): ". In this particular case *s* = "d". The remaining *s* is intpreted as a base 256 number and that is how much money the bank will give you (for example *d* is 100 in ASCII so Bill Gates is offering to give you 100 dollars). In section E, the notebook contains more specific details on how this process works.

You are greedy! You want more than $ 100! Unfortuanately, you don't have the authorization code. Your Task: Using *encTrans* generate a new encrypted string *encTransMore* to get as much money as you can. Remember if the string doesn't parse correctly (or if the string contains the wrong authentication code) you get $ 0. Also, the bank won't authorize transfers in excess of $ 7 million so don't get too greedy.

This illustrates an important issue in cryptography called Malleability. The Cramer-Shoup cryptosystem is an example of a cryptosystem that is provably non-malleable.

(Bonus 25 pts) This problem is very difficult. It will require a lot of work and some additional reading. It will require you to read and understand material on several different topics: lattices, the LLL algorithm, and Coppersmith's method. TA help for this problem will be very limited so we would recommend trying only if you are really interested in the topic.

After their fiasco, the manager at company X decides to generate unique access passwords for each of the employees. His message to employee reads as follows:

```
To get the bonus points all you need to do is get the password at the
end of this message. This will be tough to do! Extra message here
just to pad the length...... The password is: *********
```

where ********** is replaced with the actual password. The encrypted version of this message can be found at the end of the mathematica notebook. The message was encrypted with the EncryptMessageString function. Your task is to recover the password.

Relevant Sources:

- LLL Algorithm
- Coppersmith's Method
- Mathematica - a few of the algorithms you will need are already build into Mathematica