Assignment 3, 2012

Theodore S. Norvell

6892 Due Nov 15 10:30

Unless otherwise noted, all complexities are worst case time complexity on a RAM.

Q0 [10] A public key cipher.

Background. Public key cipher system consists of

- A natural number n and a natural number m.
- A large set K of numbers such that $K \subseteq \{0, ..2^m\}$. The elements of K are called the public keys.
- For each k in K, an encryption function $f_k : \{0, ..2^n\} \to \{0, ..2^n\}$. Each encryption function must be invertible, i.e., there must be, for any $k \in K$) a function $f_k^{-1} : \{0, ..2^n\} \to \{0, ..2^n\}$ so that $f_k^{-1}(f_k(p)) = p$, for each $p \in \{0, ..2^n\}$.
- A large set of things S and a public key generation function $g : S \to K$. The elements of S are called the secret keys.
- For each element s of S, a decryption function $d_s : \{0, ..2^n\} \to \{0, ..2^n\}$. Each decryption function must obey $d_s(f_{g(s)}(p)) = p$, for each $p \in \{0, ..2^n\}$

  To be practical the functions f, d, and g should have fast implementations. To be secure, it must be very difficult to compute f_k^{-1} if k is known and even if numerous pairs $(p, f_k(p))$ are also known. In particular, it must be difficult to compute s, given k, *even though g is known.*[0]

Public key cipher systems are used as follows. Every party x chooses (at random) a private key $s_x \in S$, which they keep as a secret. From this they can generate a public key $k_x = g(s_x)$,which they publish. Now suppose Bob wants to send Alice a message p. Bob encrypts p with Alice's public key to obtain a cipher text $c = f_{k_{\text{Alice}}}(p)$. Next Bob transmits c to Alice. (If Bob's message is too long to be encoded as a single n-bit number, he just breaks it up into a sequence of smaller pieces.) Alice can compute $d_{s_{\text{Alice}}}(c)$; the result will be $d_{s_{\text{Alice}}}(c) = d_{s_{\text{Alice}}}(f_{k_{\text{Alice}}}(p)) = d_{s_{\text{Alice}}}(f_{g(s_{\text{Alice}})}(p)) = p$. If Eve obtains c by eavesdropping on the communication link, and if the system is secure, she will not be able to quickly compute p from the information she has: c, k_{Alice}, and a description of the whole cipher system.

(a) Alice has devised a family of public key cipher systems (m, n, K, f, S, g, d), where m and n can be as large as you like. Alice has created an algorithm for encrypting information (i.e., for evaluating f), such that encrypting an n-bit plain text message to an n-bit encrypted message,

[0] Just as an example of how this is (seemingly) possible, in the RSA system, each element s is a pair of fairly large prime numbers and k is their product. Thus to compute s from k you must know how to factor large numbers (at least those that are knwn to be the products of two primes). This factoring problem appears to be difficult (i.e the best algorithms take a very long time) when the number of bits in k is above 1000.

using an m-bit key, requires (in the worst case) $\Theta(n^3 \times m^2)$ one-bit operations. What does the existence of such an algorithm imply about the time complexity of the encryption problem.

(b) The best algorithm that Alice can find to decrypt an n-bit message without knowing s (i.e., to implement f_k^{-1}) takes, in every case, $\Theta(n^3 + 2^m)$ time. Does this imply that Alice's cipher is secure for large m? Explain your answer.

(c) Suppose Alice later proves that her decryption algorithm from part (b) is optimal in the sense that there can be no decryption algorithm with a worst-case time complexity outside of $\Omega(n^3 + 2^m)$. Does this imply that Alice's cipher is secure? Explain your answer. If you answer 'no', explain what would be required to ensure that the cipher is secure.

Q1. Multiplying

Consider the problem of computing the product of two n-bit numbers.

(a) Devise an algorithm to multiply two arrays a and b, where a and b consist of n numbers each 0 or 1,the result should go in an array c of length $2n$. The number represented by the first n bits of an array x is $\text{val}(x, n) = \sum_{i \in \{0,..n\}} x(i) \times 2^i$. Your algorithm should use only bit-level operations (e.g. the multiplication of 2 1-bit numbers or addition of 2 or 3 1-bit numbers) on the data.

(b) What are the exact number (as a function on n) of one bit multiplications and the exact number of one-bit additions your algorithm performs? What then is the asymptotic time complexity of the algorithm. (Use big O, big Θ, or Ω as appropriate.)

(c) What does the existence of your algorithm imply about the complexity of the problem? (Use big O, big Θ, or Ω as appropriate.)

(d) Give a lower bound on the complexity of the problem? (Use big O, big Θ, or Ω as appropriate.) Justify your answer.

Q2. Is Dijkstra's algorithm optimal?

Dijkstra's algorithm has a time complexity of $\Theta(|E| \times \log |V|)$. Consider all algorithms that find the shortest paths from one node to all others that (like Dijkstra's) have the following two properties. (0) The algorithm can be modified to print out the nodes in order from closest to s to farthest from s. (1) The only operations on the weights are (a) to compare two weights or (b) to add the weights of two paths p and q such that p ends at the node where q begins, in order to compute the weight of the combined path.

Show that, among all such algorithms, Dijkstra's algorithm is optimal in the sense that any algorithm for the problem with these two properties will have a worst-case time function in $\Omega(|E| \times \log |V|))$.

Hint. Use an reductio ad absurdum argument; assume that there is an algorithm with a worst-case time function not in $\Omega(|E| \times \log |V|))$ and show that this assumption contradicts what we know to be true about sorting.

Q3 [5]. (For the purpose of this question, the word 'function' means a function from $\mathbb{R}$ to $\mathbb{R}$ that is eventually always positive. Note that the definition of Ω here is different from that in the notes. Your solution will show that this difference does not matter.)

Definition 0 *For a function g, $O(g)$ is the set of all functions f such that*

$$\exists c, m \in \mathbb{R} \cdot c > 0 \wedge (\forall n \in \mathbb{R} \cdot n > m \Rightarrow f(n) \leq cg(n)) .$$

Definition 1 *For a function g, $\Omega(g)$ is the set of all functions f such that*

$$\exists c, m \in \mathbb{R} \cdot c > 0 \wedge (\forall n \in \mathbb{R} \cdot n > m \Rightarrow f(n) \geq cg(n)) .$$

From these definitions prove that, for any functions f and g, $f \in O(g)$ if and only if $g \in \Omega(f)$. Make your proof crystal clear.