CPSC 629: Analysis of Algorithms, Fall 2003

Solutions to Homework 6

Solution to 1 (32.2-2)
Assume each pattern has length m. We can follow the Rabin-Karp algorithm to compute the numerical value of every m-substring modulo p, where p is a randomly selected prime number. If any of these values is congruent to the numerical value of any pattern modulo p, then we further check if the m-substring is a real occurrence of a pattern.

If the patterns have different lengths, the algorithm is similar to the above, except that now we need to compute the numerical values of subsets of different lengths.
□

Solution to 2 (31.7-1)
Solving the equation

$$e \cdot d \equiv 1 \pmod{\phi(n)},$$

where $e = 3$, $\phi(n) = 280$, we have a unique solution $d = 187$, which is the secret key. The encryption for the message 100 is:

$$P(100) = 100^3 \pmod{319} = 254.$$

Comments on common errors: Keys and encryptions are positive integers. □

Solution to 3 (31-1)

1. Let $\alpha = \gcd(a/2, b/2)$. Then let $a/2 = \beta\alpha$ and $b/2 = \gamma\alpha$, where β, γ are integers that are relatively prime. This implies that $a = 2\alpha\beta$ and $b = 2\alpha\gamma$. Since β, γ are relatively prime, 2α is the greatest common divisor of a and b.

2. Let $\alpha = \gcd(a, b/2)$. Then let $a = \beta\alpha$ and $b/2 = \gamma\alpha$, where β, γ are integers that are relatively prime. This implies that $b = 2\gamma\alpha$. Since β, γ are relatively prime and β is odd, $\beta, 2\gamma$ are also relatively prime. Therefore, α is the greatest common divisor of a and b.

3. Let $\alpha = \gcd(a, b)$. Then let $a = \beta\alpha$ and $b = \gamma\alpha$, where β, γ are integers that are relatively prime. This implies that $a - b = (\beta - \gamma)\alpha$. Since β, γ are both odd, $\beta - \gamma = 2\delta$ for some integer δ. Hence, $(a - b)/2 = \alpha(\beta - \gamma)/2 = \alpha\delta$. Because β, γ are relatively prime, so are $\beta - \gamma, \gamma$. Moreover, because $\beta - \gamma$ is even and γ is odd, we have that $(\beta - \gamma)/2, \gamma$ are relatively prime. Therefore, α is the greatest common divisor of $(a - b)/2$ and b.

4. The binary gcd algorithm is simply a recursive algorithm that is based on the above three cases. We bound the number of recursive calls by a recurrence relation that is based on the product of a and b. Let $f(c)$ be the number of recursive calls to the binary gcd algorithm to solve $\gcd(a, b)$, where $c = a \cdot b$. Then we have the following recurrence relation: $f(c) = f(c') + d$, where $c' \leq c/2$, and d is a constant. Solving the recursive relation with the base case $f(1) = O(1)$, we have $f(c) = O(\log c) \leq O(\log a^2) = O(\log a)$.

□

Solution to 4 (31-3)

1. Let $f(i)$ be the time to compute the i-th Fibonacci number. Then the recurrence (3.21) implies that $f(n) = f(n-1) + f(n-2) + c$, where c is a constant. Solving the recurrence relation with the base case $f(1) = O(1)$, we have $f(n) = O(\phi^n)$, where $\phi = (1 + \sqrt{5})/2$. The running time is exponential in n.

2. Using memoization, the algorithm for computing the n-th Fibonacci number is simply filling in a table of size n, where the i-th element is the i-th Fibonacci number. Let $f(i)$ be the time needed to fill in the 1 to i-th element of the table. Then $f(n) = f(n-1) + c$, where c is a constant. Solving the recursive relation with the base case $f(1) = O(1)$, we have $f(n) = O(n)$.

3. Let $X = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. Then it is easy to verify that $F_i = (X^i)_{0,1}$. Therefore, in order to compute n-th Fibonacci number F_n, it suffices to compute X^n. To do so, we compute a set of matrices

$$\mathcal{S} = \{X^2, X^4, \ldots, X^{\lfloor \log n \rfloor}\}.$$

Let $n = 2^{n_1} + 2^{n_2} + \ldots + 2^{n_k}$, where $n_1 > n_2 > \ldots > n_k \geq 0$. We can compute X^n by computing

$$X^n = X^{2^{n_1}} \cdot X^{2^{n_2}} \cdot \ldots \cdot X^{2^{n_k}}.$$

Because the matrix multiplications in this algorithm cost constant time, X^n can be computed in time $O(\log n)$, and hence the n-th Fibonacci number can be computed in time $O(\log n)$.

4. In this model, the recurrence relation of the first method becomes $f(n) = f(n-1) + f(n-2) + cn$. Solving the recurrence relation, we have $f(n) = O(\phi^n)$, and hence, the running time is still exponential in n.

The recurrence relation of the second method becomes $f(n) = f(n-1) + cn$, and hence, the running time becomes $O(n^2)$.

In the third method, to compute the n-th Fibonacci number, we need to compute the set

$$\mathcal{S} = \{X^{2^1}, X^{2^2}, \ldots, X^{2^k}\},$$

where $k = \lfloor \log n \rfloor$. For any i, $1 \leq i \leq k$, X^{2^i} is computed by multiplying $X^{2^{i-1}}$ with itself. The entries in $X^{2^{i-1}}$ are Fibonacci numbers $F_{2^{i-1}-1}, F_{2^{i-1}}, F_{2^{i-1}+1}$, which have lengths $O(2^{i-1})$. Therefore, it takes time $O(2^{2(i-1)})$ to compute X^{2^i}. In total, it takes time $O(2^2 + 2^4 + \ldots + 2^{2(k-1)}) = O(2^{2k}) = O(n^2)$ to compute the set $\mathcal{S}$. Finally, to compute F_n, we need to multiply a subset of $\mathcal{S}$. In the worst case, we need to multiply all elements in $\mathcal{S}$ together. The entries in X^{2^i} have length 2^i, for $1 \leq i \leq k$. Multiplying a subset of $\mathcal{S}$ takes time $O(2^4 + 2^6 + \ldots + 2^{2k}) = O(2^{2k}) = O(n^2)$. Therefore, the third method takes time $O(n^2)$ to compute the n-th Fibonacci number in this more reasonable model.

Comments on common errors: In part (a), in order to prove that the running time is exponential in n, you need to show that the algorithm runs in time $2^{\Omega(n)}$. In part (d), the length β is related to n. Pay attention to how the lengths of numbers are calculated in each method.

□

Note: The solutions given here are terse, and in some cases, incomplete. Your answers should be complete and have more details. But you will lose points if they are too long or too complex.