Massachusetts Institute of Technology — Handout 10
6.042J/18.062J: Mathematics for Computer Science — March 4, 1997
Professor David R. Karger

Problem Set 5

Due: March 11, 1997.

Reading: State machine handout.
Remember that correctness involves termination plus the right answer!

Problem 1. Read the description of the RSA cryptosystem from the textbook, pp 146-148.

(a) Given primes $p = 173$ and $q = 149$ generate encryption and decryption keys, e and d respectively, such that $e \cdot d = 1 \bmod (p-1)(q-1)$. For "security," make $e, d \geq 20$.

(b) Convert the word "MIT" to a number by first mapping each letter to its position in the alphabet, e.g. A = 0, B = 1, C = 2, ..., Z = 25, and then gathering the numbers (M, I, T) into a single number $26^2 * M + 26 * I + T$. Find the RSA-encrypted version of "MIT" and show that it decrypts properly.

(c) What happens when you apply the encrypt/decrypt operation to "YOU"? Why?

Problem 2. Consider the following algorithm PolyEval, which takes a list of arguments $a_0, \ldots, a_n$ and an argument x, and is supposed to compute the value of the polynomial

$$\sum_{i=0}^{n} a_i x^i.$$

Prove that the algorithm is correct.

PolyEval($a_0, \ldots, a_n, x$)

$s = 0$
for $i = n$ downto 0
    $s = x * s + a_i$
return s

Problem 3. Write a (pseudocode) program that when given a natural number n returns the binary representation of n. (the output of the program must be a string over the alphabet $\{0, 1\}$). Use standard primitive operations. Prove that your program is correct.

Problem 4. Consider the following algorithm to find the greatest common divisor of a collection of numbers $a_1, \ldots, a_n$. It uses algorithm Euclid for finding the GCD of two numbers. Prove it is correct (you might want to prove a lemma about GCD(a,GCD($b_1, \ldots, b_k$))).

MultiEuclid($a_1, \ldots, a_n$)

$r = a_1$
for $i = 2$ to n
    $r =$Euclid(r, a_i)
return r

Problem 5. The following code estimates the value of $\frac{1}{1-x/2}$ to within an additive error of $1/1024$ for $0 < x < 1$. Prove that it is correct.

```
estimate(x)
{
  Fprev = 0;
  Fnext = 1;
  while (Fnext - Fprev > 1/1024) {
    Fprev = Fnext;
    Fnext = (Fprev * x / 2) +1;
  }
  return(Fnext);
}
```

Problem 6. The mode of a list is an element appearing as frequently in the list as any other. Let $A[i..j]$ denote the sublist of A with indices between i and j, inclusive. Consider the following algorithm.

Mode(A)
  if (size[A] = 1)
    return(A[1])
  if ($A[1] =$ Mode($A[2..size[A]]$))
    return(A[1])
  else
    return(Mode($A[2..(size[A])]$))

What is the fallacy in the following argument that Mode returns the mode of A? Explain the fallacy, and give a counterexample showing the algorithm fails.

Proof by induction on the length of the input. The base case is trivial. So assume the hypothesis true for inputs of length n. Consider an input of length $n+1$. Then by induction, each of the two recursive calls properly returns the mode of its argument. Thus, if $A[1]$ is the mode, the if test succeeds, and the function returns the correct value. If $A[1]$ is not the mode, then clearly removing $A[1]$ doesn't change the mode, so the second recursive call succeeds and returns the correct value.

Problem 7. A *search tree* is either:

- An *empty tree* with no nodes
- A *root node* with *key* k, and
  - a *left subtree,* all of whose nodes have keyes less than k
  - a *right subtree,* all of whose nodes have keys exceeding k.

The following fallacious algorithm Find, when called with key k and (a, b) search tree T, should print "found it" if and only if k is the key of a node in the search tree.

Find(k, T)

```
c =key(root-node(T))
if (k = c)
    print "found it"
else if (k < c)
    Find(k,left-subtree(T))
else
    Find(k,right-subtree(T))
```

(a) (3 points) What is wrong with the above algorithm? Add a starting if-test that will fix the problem.

(b) (7 points) Prove that your (fixed) algorithm does what it should.

Problem 8. Consider a coin exchanger described by the following state machine.

The state set Q consists of values for state components:

$total \in \mathbb{N}$
$total' \in \mathbb{N}$
$cents \in \mathbb{N}$
$dimes \in \mathbb{N}$
$quarters \in \mathbb{N}$.

Here *total* is an initial amount of money (in cents) given as input to the machine. All other variables are initially set to 0. When the machine terminates (no more transitions are possible) the variables *cents*, *dimes* and *quarters* should be the correct change corresponding to the input *total*. $total'$ is an auxiliary variable.

The transition function δ is defined by the following rules.

give-quarter
  Precondition:
    $total - total' \geq 25$
  Effect:
    $total' := total' + 25$
    $quarters := quarters + 1$

give-dime
  Precondition:
    $25 > total - total' \geq 10$
  Effect:
    $total' := total' + 10$
    $dimes := dimes + 1$

give-cent
  Precondition:
    $10 > total - total' > 0$
  Effect:
    $total' := total' + 1$
    $cents := cents + 1$

Prove using the invariant method that the algorithm is correct, that is, when the machine terminates the total amount of money described by the variables *cents*, *dimes*, *quarters* is exactly *total*.

Problem 9. (optional)

Consider the following procedure Bump, which operates on an infinite array of bits $b[0], b[1], b[2], \ldots$ that are all initially 0.

```
Bump(k)

    if b[k] = 0 then
        b[k] = 1
    else
        b[k] = 0
        Bump(k + 1)
    end if
```

What will the $b[i]$ contain if Bump(0) is called n times? Prove it.