CS/Math 240: Intro to Discrete Math 3/3/2011

Solutions to Homework 4

Instructor: Dieter van Melkebeek

Problem 1

We denote by Zeros(S) the number of 0's that appear in sequence S. Similarly, Ones(S) denotes the number of 1's in S.

(a) We use strong structural induction to prove the given statement. The induction is on the number of times the constructor rule is used for creating a sequence. We define $P(n)$: "In every good sequence that can be created using n applications of the constructor rule, the number of 0's and 1's is equal."

Proof. For the base case, we consider the sequence obtained without any application of the constructor rule. There is only one such good sequence, S, obtained using the foundation rule. This is the empty sequence which has no 0's and 1's. Hence, $\text{Zeros}(S) = \text{Ones}(S)$, which proves the base case.

Next we assume that $P(m)$ holds for all natural numbers $m \leq n$. We show that $P(n+1)$ holds. Consider a good sequence S created using $n+1$ applications of the constructor rule. There are two cases that we need to analyze depending on the first character of the sequence.

Case 1: The first character of S is 0. This means that S is of the type $0s1t$. Since S was created using $n+1$ applications of the constructor rule, both s and t, were created by at most n applications of the constructor rule. Therefore, by our induction hypothesis,

$$\text{Zeros}(s) = \text{Ones}(s) \tag{1}$$
$$\text{Zeros}(t) = \text{Ones}(t). \tag{2}$$

By construction of S, we have that:

$$\text{Zeros}(S) = 1 + \text{Zeros}(s) + \text{Zeros}(t) \tag{3}$$
$$\text{Ones}(S) = 1 + \text{Ones}(s) + \text{Ones}(t). \tag{4}$$

Substituting (1) and (2) into (3), we get

$$\text{Zeros}(S) = 1 + \text{Ones}(s) + \text{Ones}(t) \tag{5}$$
$$= \text{Ones}(S). \tag{6}$$

Case 2: The first character of S is 1. This means that S is of the type $1s0t$. This case can be proved using reasoning similar to as above, swapping the roles of 0 and 1. We omit the details.

This completes the inductive step. Hence, $P(n)$ holds true for all integers $n \geq 0$. □

(b) We again make use of structural induction to prove the given statement. This time the induction is on the number of 0's (equivalently, the number of 1's) in the sequence. Let's define $P(n)$: "Every sequence with n 0's and n 1's is a good sequence."

Proof. The base base is $n = 0$. In that case the sequence does not contain any 0's or 1's, meaning it is the empty sequence, which is good by the foundation rule. Hence, the base case holds.

For the induction step, we prove that for every integer $n \geq 0$, $(\bigwedge_{k=0}^{n} P(k)) \Rightarrow P(n+1)$. Let S be sequence of 0's and 1's such that $\text{Zeros}(S) = \text{Ones}(S) = n+1$. As in (a), we have two cases that we need to analyze depending on the first character of S.

Case 1: Assume that the first character of S is 0. Consider a plot of the number of 0's minus the number of 1's as we move over S from left to right. For $S = 00101101110001$ this gives us the plot in Figure 1.

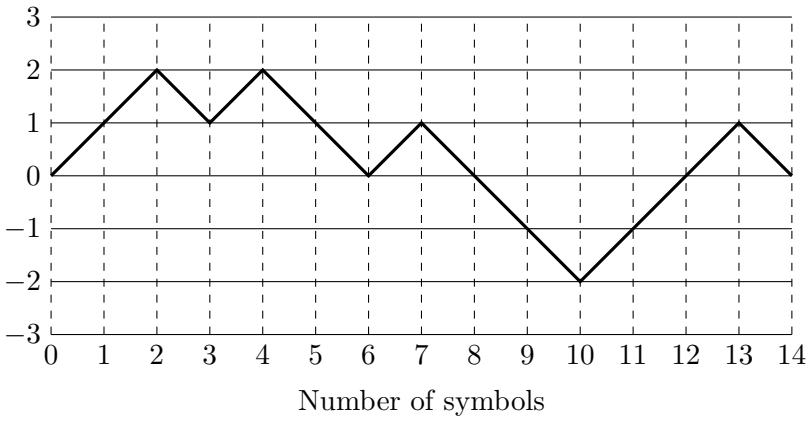

Figure 1: An example for the sequence $S = 00101101110001$.

The plot starts at 0, and, since S starts with a 0, next goes up to 1. Since S contains an equal number of 0's and 1's, the plot has to end at 0. Thus, there is a first time after the start that the plot returns to zero. Let k denote the number of 0's seen up to then. At that point, the plot has to go down from 1 to 0 due to a character 1. This is because in each step the plot can only go up or down by 1, and has to remain positive until the first time we hit 0 after the start. Let us denote by s the subsequence of S after the initial 0 and before the 1 that brings the plot back to 0 for the first time. Let us denote by t the subsequence of S after that 1. We can write S as $S = 0s1t$.

We have the following equalities:

$$\text{Zeros}(s) = \text{Ones}(s) = k - 1 \tag{7}$$

$$\text{Zeros}(S) = 1 + \text{Zeros}(s) + \text{Zeros}(t) \tag{8}$$

$$\text{Ones}(S) = 1 + \text{Ones}(s) + \text{Ones}(t) \tag{9}$$

Substituting (7) into (8) and (9), we get

$$\text{Zeros}(t) = \text{Ones}(t) = n + 1 - k \tag{10}$$

Since $1 \leq k \leq n$, we have that $k-1 \leq n$ and $n+1-k \leq n$. Therefore, by our strong induction hypothesis, both s and t are good. Combining with the constructor rule, we conclude that S is good.

Case 2: We would assume that the first character of S is a 1. We can argue as in Case 1—we only need to interchange 0 and 1. We omit the proof for this case.

This completes the inductive step. Hence, $P(n)$ holds true for all integers $n \geq 0$. □

Problem 2

(a) We begin by defining any puzzle state by its sequence of tiles in-order, starting from the top left corner and proceeding in a left-right, top-down fashion. So our initial board state is

$$(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 14, \square),$$

and our desired end state is

$$(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, \square),$$

where $\square$ denotes the empty tile. Recall that with our 8-puzzle example from class, any column moves resulted in the rearrangement of the relative order of three numbered tiles and the empty tile, so that the consecutive series $\square abc$ was transformed into $cab\square$, or vice versa:

$$\square abc \leftrightarrow cab\square.$$

It can be seen similiarly that a column move of the 16-puzzle will rearrange the relative order of four numbered tiles and the empty tile:

$$\square abcd \leftrightarrow dabc\square.$$

Such a move changes the relative order of exactly three pairs of numbered tiles, namely $\{a, d\}$, $\{b, d\}$, and $\{c, d\}$. Each of these changes either decreases or increases the total number of pairwise errors in the overall sequence by one. For example, if our move was from $(....\square, 5, 7, 1, 4, ...)$ to $(...., 4, 5, 7, 1, \square, ...)$, the total number of pairwise errors in the sequence would decrease by 1 [$(5, 4) \rightarrow (4, 5)$ removes an error, $(7, 4) \rightarrow (4, 7)$ also removes an error, and $(1, 4) \rightarrow (4, 1)$ adds an error]. In general, the total change from a single column move is describable as $\Delta_{errors} = x + y + z$, where $x, y, z, \in \{-1, 1\}$. Since Δ_{errors} is the sum of three odd integers, this sum is necessarily an odd integer as well. So every column move changes the total number of pairs in error by an odd number [More specifically, by one of -3, -1, 1, or 3—but it is not necessary to be this specific].

We now make the following statement: for all $k \geq 0$, after $2k$ column moves there is an odd number of errors in our 16-puzzle. We refer to the latter statement as $P(k)$. The base case $P(0)$ holds as out start state contains exactly one pair in error. Assuming $P(k)$ holds for some natural number k, we show that $P(k+1)$ holds.

From the above argument, the change in the number of pairs in error after any single column move is odd, i.e., expressible as $2n+1$ for some integer n. The total change in error after two column moves is then $(2m+1) + (2n+1) = 2(m+n+1)$ for two integers m and n. This

change is necessarily even. The sum of an even and odd number is necessarily odd, so if $P(k)$ holds, $P(k+1)$ must as well.

We have now shown that there can be no solution for the 16-puzzle that uses an even number of column moves. This is equivalent to showing that all solutions for the 16-puzzle must use an odd number of column moves.

(b) We begin by defining any puzzle state by R, the number of rows that the empty square is away from the fourth row. Initially, $R = 0$. We note that any column move changes the row position of the empty square by 1 or -1. So after any two column moves, there is a total change in R of 2, 0, or -2. We make the following statement: for all $k \geq 0$, after $2k+1$ moves R is odd. We denote the latter statement by $Q(k)$.

Our base case $Q(0)$ corresponds to the puzzle state after one column move; in this case $R = 1$, which is odd. Assuming $Q(k)$ holds for some natural number k [$R_k = 2n+1$ for some integer n], we see that R_{k+1} is one of either $2n+3, 2n+1$, or $2n-1$. All of these are odd integers. Thus, $Q(k+1)$ holds.

Since at the end of the puzzle $R = 0$, which is even, we have shown that there can be no solution for the 16-puzzle that uses an odd number of column moves. This is equivalent to showing that all solutions for the 16-puzzle must use an even number of column moves.

(c) Since any any solution requires both an odd number of column moves and an even number of column moves, no solution can exist.

Problem 3

(a) Let $P(k)$ be the proposition: The kth time we reach the test of the while loop, $0 \leq i \leq n$ and x does not appear in $A[0..(i-1)]$. We need to prove that the proposition holds for all positive integer values of k.

Proof. Let's denote by i_k, the value of variable i the kth time we reach the test of the while loop. We prove that $0 \leq i_k \leq n$ and x does not appear in $A[0..(i_k - 1)]$. We use induction on k.

The base case is $P(1)$. The only program statement to have executed so far is line 1 which sets i to 0, i.e., $i_1 = 0$. Moreover, $n \geq 0$ by the input precondition. So $0 \leq i_1 \leq n$ holds. As per the stated convention, $A[0..-1]$ is the empty array. Since the empty array contains no element, x does not appear in $A[0..(i_1 - 1)]$. Hence the base case holds.

For the inductive step, assume that $P(k)$ holds for some integer $k \geq 1$. Assuming we reach the test of the while loop for the $(k+1)$st time, the loop condition evaluated to true the kth time around, which leads us to conclude that

$$i_k < n. \tag{11}$$

Also, during the kth iteration of the loop, the return statement in line 3 was not executed. Otherwise, we would not reach the test of the while loop for the $(k+1)$st time. From this we infer that the if statement evaluated to false, i.e.,

$$A[i_k] \neq x. \tag{12}$$

During the kth iteration, line 4 has executed. Therefore,

$$i_{k+1} = i_k + 1. \tag{13}$$

Combining (11) and (13), we get that $i_{k+1} \leq n$. As per the induction hypothesis, $i_k \geq 0$; therefore $i_{k+1} \geq 0$. Again, x does not appear in $A[0..(i_k - 1)]$. Combining this with (12), we get x does not appear in $A[0..i_k]$ which is same as $A[0..(i_{k+1} - 1)]$. Hence, $P(k+1)$ holds true as well.

Therefore, the loop invariant is maintained throughout. □

(b) *Partial Correctness*: Assume that the program terminates. There are only two possible return points for the program.

First, the return statement in line 3 got executed. The output in this case would be the value of variable i. The execution of the return statement means that the if condition evaluated to true i.e. $A[i] = x$. By our loop invariant $A[0..(i-1)]$ does not contain x, therefore i is the smallest index, such that $A[i] = x$.

Second, the return statement in line 5 got executed. This means that the condition $i < n$ failed. Since $0 \leq i \leq n$ holds by our invariant, this means that $i = n$. The second part of our loop invariant then states that $A[0..(n-1)]$ does not contain x. The program returns -1 in this case, which is as per the specification.

Termination: In each complete iteration of the while loop the value of i is increased by one. Since the loop ends as soon as $i \leq n$, there can only be a finite number of iterations.

Problem 4

The given algorithm is known as Euclid's algorithm for the greatest common divisor. We restate the algorithm below for completeness.

Algorithm 1: GCD(a, b)

Input: $a, b \in \mathbb{N}$, at least one of a, b is nonzero
Output: $\gcd(a, b)$

(1) **if** $a \leq b$ **then** $(x, y) \leftarrow (a, b)$
(2) **else** $(x, y) \leftarrow (b, a)$
(3) **while** $x > 0$ **do**
(4) $r \leftarrow y - \lfloor y/x \rfloor \cdot x$
(5) $y \leftarrow x$
(6) $x \leftarrow r$
(7) **return** y

Part a

Recall the following lemma from class.

Lemma 1. *Let x, y be positive integers. Then the following are true.*

(i) If $x = y$, $\gcd(x, y) = x$

(ii) If $x < y$, $\gcd(x, y) = \gcd(x, y - x)$

(iii) If $x > y$, $\gcd(x, y) = \gcd(x - y, y)$

We can extend part (ii) of this lemma to the following.

Lemma 2. *Let $x, y \in \mathbb{N}$ such that $0 < x \leq y$, and let $r = y - \lfloor y/x \rfloor \cdot x$. Then* $\gcd(x, y) = gcd(x, r)$.

Proof. Just apply part (ii) of Lemma 1 $\lfloor y/x \rfloor$ times. □

Now we prove partial correctness of Euclid's algorithm. To that end, we prove the following loop invariant.

Theorem 1. *Each time we reach the test of the while loop on line 3, $0 \leq x \leq y$ and* $\gcd(x, y) = \gcd(a, b)$.

Proof. Let x_k, y_k, r_k be the values of x, y and r the kth time we reach the test of the while loop. We show by induction on k that $0 \leq x_k \leq y_k$ and $\gcd(x_k, y_k) = \gcd(a, b)$.

For the base case $k = 1$, we need to find what x_1 and y_1 are. We do this by cases.

Case 1: $a \leq b$. In this case, the condition on the first line is true, so we set $x_1 = a$ and $y_1 = b$. Then $x_1 \leq y_1$. Hence, $\gcd(x_1, y_1) = \gcd(a, b)$ by initialization. Since the specification says a and b are non-negative, $x_1 \geq 0$ holds too.

Case 2: $a > b$. In this case, the condition on the first line is false, so we set $x_1 = b$ and $y_1 = a$. Then $x_1 < y_1$ and $\gcd(x_1, y_1) = \gcd(b, a) = \gcd(a, b)$ by initialization. Since the specification says a and b are non-negative, $x_1 \geq 0$.

This completes the proof of the base case.

Now suppose that $\gcd(x_k, y_k) = \gcd(a, b)$ and $0 \leq x_k \leq y_k$, and suppose we reach the test of the while loop for the $(k + 1)$st time. The latter implies that the kth interation of the while loop completed, so $x_k > 0$. In this case we have that $r_k = y_k - \lfloor \frac{y_k}{x_k} \rfloor \cdot x_k$. Since $\lfloor \frac{y_k}{x_k} \rfloor \leq \frac{y_k}{x_k} < \lfloor \frac{y_k}{x_k} \rfloor + 1$ and $x_k > 0$, it follows that

$$\left\lfloor \frac{y_k}{x_k} \right\rfloor \cdot x_k \leq \frac{y_k}{x_k} \cdot x_k = y_k < \left(\left\lfloor \frac{y_k}{x_k} \right\rfloor + 1 \right) \cdot x_k = \left\lfloor \frac{y_k}{x_k} \right\rfloor \cdot x_k + x_k.$$

Therefore, $0 \leq r_k = y_k - \lfloor \frac{y_k}{x_k} \rfloor \cdot x_k < x_k$. Since $x_{k+1} = r_k$ we have that

$$0 \leq x_{k+1} < x_k, \tag{14}$$

and since $y_{k+1} = x_k$, that $0 \leq x_{k+1} \leq y_{k+1}$, which establishes the first part of the invariant. As for the second part, we also have $\gcd(x_{k+1}, y_{k+1}) = \gcd(r_k, x_k)$. By Lemma 2, the latter is $\gcd(x_k, y_k)$, and the induction hypothesis tells us that $\gcd(x_k, y_k) = \gcd(a, b)$. It follows that $\gcd(x_{k+1}, y_{k+1}) = \gcd(a, b)$. □

Since Theorem 1 tells us that $x \geq 0$, and the loop terminates only if $x \leq 0$, we see that when the loop terminates, $x = 0$. In that case $\gcd(x, y) = y$ because every number divides zero and y is the largest divisor of y. By the invariant from Theorem 1, $\gcd(x, y) = \gcd(a, b)$ at that point, which gives us partial correctness of the algorithm.

Now we show that the algorithm terminates. Consider the situation the kth time we reach the test of the while loop. First, if $x_k \leq 0$, the loop terminates and the algorithm returns right after that. When $x_k > 0$, we observe that $x_{k+1} < x_k$ because of (14). Thus, x is a nonnegative integer which decreases by at least one in each iteration of the loop unless it's zero already. Since x is initialized to be $\min(a, b)$, after at most $\min(a, b)$ iterations of the loop, $x = 0$, the loop terminates, and the program returns.

This completes the proof that Euclid's algorithm for the greatest common divisor is correct.

Here we also remark that Algorithm 1 is an improvement over the algorithm discussed in class since it works in all cases in which $\gcd(a, b)$ is defined. Unlike the algorithm from lecture, this one also works when one of the arguments is zero.

Part b

Now we show that the kth time we reach the test of the while loop, there exist integers (possibly negative) w_k, z_k, u_k, v_k such that $x_k = w_k a + z_k b$ and $y_k = u_k a + v_k b$. We prove this invariant by inductionk on k.

For the base case, we have two cases.

Case 1: $a \leq b$. Then $x_1 = a$ and $y_1 = b$, so let $w_1 = v_1 = 1$, and $z_1 = u_1 = 0$.

Case 2: $a > b$. Then $x_1 = b$ and $y_1 = a$, so let $z_1 = u_1 = 1$ and $w_1 = v_1 = 0$.

This completes the proof of the base case.

Now suppose that the kth time we reach the test of the while loop, there are integers w_k, z_k, u_k, v_k such that $x_k = w_k a + z_k b$ and $y_k = u_k a + v_k b$. If $x_k = 0$, there is no next iteration of the loop and thus nothing to prove. So assume that $x_k > 0$. A careful examination of the proof of Lemma 2 gives us the values of our four integers the $(k+1)$st time we reach the test of the while loop. Since $x_{k+1} = r_k = y_k - \lfloor \frac{y_k}{x_k} \rfloor x_k$, we can write $x_{k+1} = (u_k a + v_k b) - \lfloor \frac{y_k}{x_k} \rfloor \cdot (w_k a + z_k b) = (u_k - \lfloor \frac{y_k}{x_k} \rfloor w_k)a + (v_k - \lfloor \frac{y_k}{x_k} \rfloor z_k)b$, so we can pick $w_{k+1} = u_k - \lfloor \frac{y_k}{x_k} \rfloor w_k$ and $z_{k+1} = v_k - \lfloor \frac{y_k}{x_k} \rfloor z_k$. Since $y_{k+1} = x_k$, we can write $y_{k+1} = w_k a + z_k b$ and pick $u_{k+1} = w_k$ and $v_{k+1} = z_k$. All choices of the coefficients are valid as they are all integers.

This completes the proof of the invariant.

Part c

Our proof of the inductive step in the proof from part (b) gives us the modification of the algorithm we need, and we show it as Algorithm 2. This algorithm is known as the *Extended Euclidean algorithm*.

Algorithm 2: EXTENDED-GCD(a, b)

Input: $a, b \in \mathbb{N}$, at least one of a, b is nonzero
Output: $\gcd(a, b)$ and integers u, v such that $\gcd(a, b) = au + bv$

(1) **if** $a \leq b$ **then** $(x, y, w, z, u, v) \leftarrow (a, b, 1, 0, 0, 1)$
(2) **else** $(x, y, w, z, u, v) \leftarrow (b, a, 0, 1, 1, 0)$
(3) **while** $x > 0$ **do**
(4) $q \leftarrow \lfloor y/x \rfloor$
(5) $r \leftarrow y - qx$
(6) $y \leftarrow x$
(7) $x \leftarrow r$
(8) $wNew \leftarrow u - qw$
(9) $zNew \leftarrow v - qz$
(10) $u \leftarrow w$
(11) $v \leftarrow z$
(12) $w \leftarrow wNew$
(13) $z \leftarrow zNew$
(14) **return** (y, u, v)

Problem 5

Fix two positive integers a and b. We prove the two implications separately.

First let's show that if $\gcd(a, b) = 1$, then there is some integer n_0 such that for all $n \geq n_0$, we can realize a postage of n cents using a-cent and b-cent stamps.

Since $\gcd(a, b) = 1$, there exist $x, y \in \mathbb{Z}$ such that $1 = ax + by$ by part (b) of problem 4. This suggests that if we can realize a postage of n cents, say $n = ca + db$ for $c, d \in \mathbb{N}$, then we can realize $n + 1$ cents as $n + 1 = ca + db + ax + by = (c + x)a + (d + y)b$. However, x and/or y may be negative, so we cannot guarantee that $c + x \in \mathbb{N}$ and $d + y \in \mathbb{N}$. In fact, except when a and b are at most 1, one of x or y has to be negative (and the other one positive). Assume without loss of generality that $y < 0$. Then after some number of applications of our strategy, we start using negative numbers of b-cent stamps, which is not valid. Thus, we need to be able to "restart" at some point and increase the number of b-cent stamps we use. We can "restart" at a postage that is a multiple of b. If $n = kb$, we can use k b-cent stamps to realize this postage. For the next $b - 1$ postages, we need to apply our rule before we get to a postage that can be realized using only b-cent stamps again. In those $b - 1$ applications of our rule, we reduce the number of b-cent stamps from k to $k + (b - 1)y$. To be able to do this, we need $k + (b - 1)y \geq 0$, so pick n_0 to be the first k for which this works.

We now argue the other implication. Suppose that $\gcd(a, b) \neq 1$. In general, every value that can be realized using stamps of a and b cents, has to be a multiple of $\gcd(a, b)$. To see this, note that $a = m \cdot \gcd(a, b)$ and $b = n \cdot \gcd(a, b)$ for some $m, n \in \mathbb{N}$. Hence, a postage formed with x a-cent stamps and y b-cent stamps is $xa + yb = xm \cdot \gcd(a, b) + yn \cdot \gcd(a, b) = (xm + yn) \cdot \gcd(a, b)$. Since $\gcd(a, b) \neq 1$, this means that there are infinitely many values that cannot be realized, e.g., all values of the form $k \cdot \gcd(a, b) + 1$, where $k \in \mathbb{N}$. Thus, no matter how large we pick n_0, there is always a value $n \geq n_0$ that cannot be realized.