CS 70 Discrete Mathematics and Probability Theory
Summer 2020 Course Notes

Homework 2

1 Exam Policy and Practice

Please read the exam policy carefully before proceeding. This question is designed to familiarize you with some of the things you will have to do during the exam.

1. Fill out the following Google Form to submit the Zoom link you will be using:

   You must use this Zoom link for this assignment, as well as for the exams.

2. Start the Zoom call whose link you provided above. Turn on your microphone and webcam. Turn off your speaker. Share your entire desktop (not just a particular window).

3. Start recording via Zoom. Record locally so that in the event of your internet connection dying your meeting still continues to be recorded.

4. Hold your CalID next to your face and record yourself saying your name into the webcam. Both your face and your entire CalID should be visible in the video. We should be able to read your name and SID. This step should take **at least** 3 seconds. See figure 1. *If you do not have a CalID for some reason, please hold up some document which has an image of you and proves your identity, such as a driver's license.*

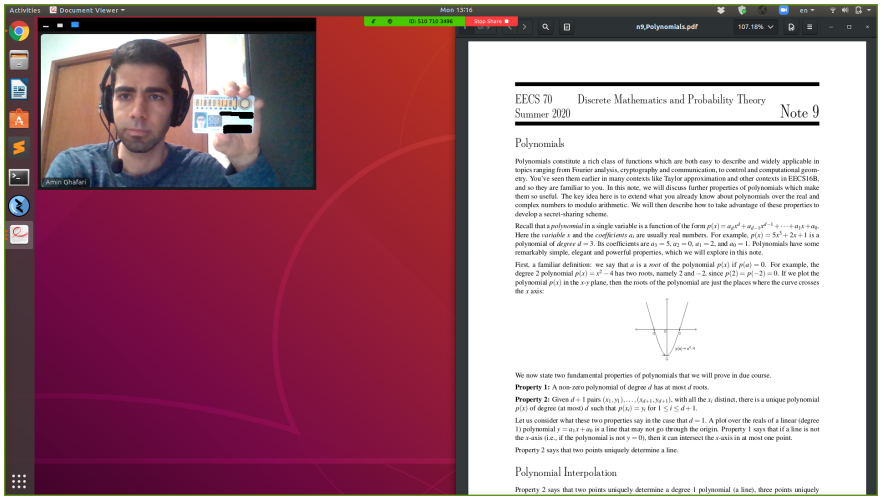


Figure 1: ID card demonstration. Do not actually black out your SID and name.

5. Turn your webcam/laptop around 360° **slowly** so that we can see your entire room clearly. There should be no uncovered screens anywhere in the room during your exam. Only admin TAs and instructors will be able to see your videos (both for this assignment and for the actual exams).

6. Position your webcam/laptop in such a way that we can see your upper body from your head to your hands. Your face and hands must be in the frame at all times. Your phone should also be in the frame at all times, turned upside down. If you do not have sufficient space to include your head, then please make sure to include your hands, phone, and as much of your torso as possible. See figure 2.

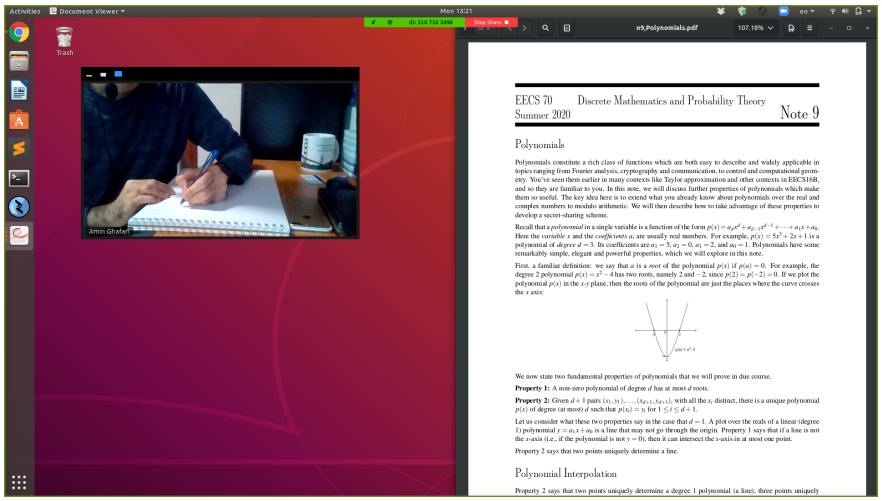


Figure 2: Demonstration of taking your exam. Your setup should look like this while you are taking the exam.

7. Your microphone should be on at all times. We should be able to see the time on your desktop at all times.

8. Write down following the honor code in this state. Then read it out loud. This will ensure you understand the positioning requirement, and have enabled audio.

   I pledge to uphold the university's honor code: to act with honesty, integrity, and respect for others, including their work. By signing, I ensure that all written homework I submit will be in my own words, that I will acknowledge any collaboration or help received,and that I will neither give nor receive help on any examinations.

9. Stop the recording. Upload your video to Google drive and submit the link to the video using this Google Form. You must make sure that the link sharing permissions are set so that we may view the video. Also, please submit your zoom link to this this google form. Afterwards, please write down the magic words from each google form to indicate you have down so.

Link for policy:
https://docs.google.com/document/d/1kj83rPvxgoLuaafkJLcZsmIslWjrhxDujfng4K5oSPI/edit?usp=sharing
Link to upload video:
https://forms.gle/emBA5xzK83wn1uRH8
Link to share zoom link:
https://forms.gle/Co9Rdhx6QV413PFG6

2 Countability Practice

(a) Do $(0,1)$ and $\mathbb{R}_+ = (0,\infty)$ have the same cardinality? If so, either give an explicit bijection (and prove that it is a bijection) or provide an injection from $(0,1)$ to $(0,\infty)$ and an injection from $(0,\infty)$ to $(0,1)$ (so that by Cantor-Bernstein theorem the two sets will have the same cardinality). If not, then prove that they have different cardinalities.

(b) Is the set of strings over the English alphabet countable? (Note that the strings may be arbitrarily long, but each string has finite length. Also the strings need not be real English words.) If so, then provide a method for enumerating the strings. If not, then use a diagonalization argument to show that the set is uncountable.

(c) Consider the previous part, except now the strings are drawn from a countably infinite alphabet $\mathscr{A}$. Does your answer from before change? Make sure to justify your answer.

Solution:

(a) Yes, they have the same cardinality.
Explicit bijection: Consider the bijection $f : (0,1) \to (0,\infty)$ given by

$$f(x) = \frac{1}{x} - 1.$$

We show that f is a bijection by proving separately that it is one-to-one and onto. The function f is one-to-one: suppose that $f(x) = f(y)$. Then,

$$\begin{aligned} \frac{1}{x} - 1 &= \frac{1}{y} - 1, \\ \frac{1}{x} &= \frac{1}{y}, \\ x &= y. \end{aligned}$$

Hence, f is one-to-one.

The function f is onto: take any $y \in (0,\infty)$. Let $x = 1/(1+y)$. Note that $x \in (0,1)$. Then,

$$f(x) = \frac{1}{1/(1+y)} - 1 = 1 + y - 1 = y,$$

so f maps x to y. Hence, f is onto.

We have exhibited a bijection from $(0,1)$ to $(0,\infty)$, so they have the same cardinality. (In fact, they are both uncountable.)

Indirect bijection: The injection from $(0,1)$ to $(0,\infty)$ is trivial; consider the function $f : (0,1) \to (0,\infty)$ given by

$$f(x) = x.$$

It is easy to see that f is injective.

For the other way, consider the function $g : (0,\infty) \to (0,1)$ given by

$$g(x) = \frac{1}{x+1}.$$

To see that g is injective, suppose $g(x) = g(y)$. Then

$$\frac{1}{x+1} = \frac{1}{y+1} \implies x = y.$$

Hence g is injective. Thus we have an injective function from $(0,1)$ to $(0,\infty)$ and an injective function from $(0,\infty)$ to $(0,1)$. By Cantor-Bernstein theorem there exists a bijection from $(0,1)$ to $(0,\infty)$ and hence they have the same cardianlity.

(b) Countable. The English language has a finite alphabet (52 characters if you count only lower-case and upper-case letters, or more if you count special symbols – either way, the alphabet is finite).

We will now enumerate the strings in such a way that each string appears exactly once in the list. We will use the same trick as used in Lecture note 10 to enumerate the elements of $\{0,1\}^*$ We get our bijection by setting $f(n)$ to be the n-th string in the list. List all strings of length 1 in lexicographic order, and then all strings of length 2 in lexicographic order, and then strings of length 3 in lexicographic order, and so forth. Since at each step, there are only finitely many strings of a particular length ℓ, any string of finite length appears in the list. It is also clear that each string appears exactly once in this list.

(c) No, the strings are still countable. Let $\mathscr{A} = \{a_1, a_2, \dots\}$ denote the alphabet. (We are making use of the fact that the alphabet is countably infinite when we assume there is such an enumeration.) We will provide two solutions:

Alternative 1: We will enumerate all the strings similar to that in part (b), although the enumeration requires a little more finesse. Notice that if we tried to list all strings of length 1, we would be stuck forever, since the alphabet is infinite! On the other hand, if we try to restrict our alphabet and only print out strings containing the first character $a \in \mathscr{A}$, we would also have a similar problem: the list

$$a, aa, aaa, \dots$$

also does not end.

The idea is to restrict *both* the length of the string and the characters we are allowed to use:

(a) List all strings containing only a_1 which are of length at most 1.

(b) List all strings containing only characters in $\{a_1, a_2\}$ which are of length at most 2 and have not been listed before.

(c) List all strings containing only characters in $\{a_1, a_2, a_3\}$ which are of length at most 3 and have not been listed before.

(d) Proceed onwards.

At each step, we have restricted ourselves to a finite alphabet with a finite length, so each step is guaranteed to terminate. To show that the enumeration is complete, consider any string s of length ℓ; since the length is finite, it can contain at most ℓ distinct a_i from the alphabet. Let k denote the largest index of any a_i which appears in s. Then, s will be listed in step $\max(k,\ell)$, so it appears in the enumeration. Further, since we are listing only those strings that have not appeared before, each string appears exactly once in the listing.

Alternative 2: We will encode the strings into ternary strings. Recall that we used a similar trick in Lecture note 10 to show that the set of all polynomials with natural coefficients is countable. Suppose, for example, we have a string: $S = a_5a_2a_7a_4a_6$. Corresponding to each of the characters in this string, we can write its index as a binary string: $(101, 10, 111, 100, 110)$. Now, we can construct a ternary string where "2" is inserted as a separator between each binary string. Thus we map the string S to a ternary string: 101210211121002110. It is clear that this mapping is injective, since the original string S can be uniquely recovered from this ternary string. Thus we have an injective map to $\{0,1,2\}^*$. From Lecture note 10, we know that the set $\{0,1,2\}^*$ is countable, and hence the set of all strings with finite length over $\mathscr{A}$ is countable.

3 Fixed Points

Consider the problem of determining if a function F has any fixed points; that is, we want to know if there is any input x such that $F(x)$ outputs x. Prove that this problem is undecidable.

Solution:

We can prove this by reducing from the Halting Problem. Suppose we had some function `FixedPoint`(F) that solved the fixed-point problem. We can define `TestHalt`(F,x) as follows:

```
def TestHalt(F, x):
  def F_prime(y):
    F(x)
    return y
  return FixedPoint(F_prime)
```

If $F(x)$ halts, we have that $F'(y)$ will always just return y, so every input is a fixed point. On the other hand, if $F(x)$ does not halt, F' won't return anything for any input y, so there can't be any fixed points. Thus, our definition of `TestHalt` must always work, which is a contradiction; this tells us that `FixedPoint` cannot exist.

4 The Complexity Hierarchy

The complexity hierarchy is a monument to our collective understanding of computation and its limitations. In fact, you may already be familiar with the classes P and NP from CS61B. In this

problem, we will focus on decision problems like the Halting Problem, where the output is Yes (True) or No (False), and explore the classes RE, coRE, and R.

(a) A problem is recursively enumerable (RE) if there exists a program P that can print out all the inputs for which the answer is Yes, and no inputs for which the answer is No. The program P can print out a given input multiple times, so long as every input gets printed eventually. The program P can run forever, so long as every input which should be printed is at a finite index in the printed output.

Prove that the Halting Problem belongs in RE. Namely, prove that it is possible to write a program P which:

- runs forever over all possible programs M and inputs x, and prints out strings to the console,
- for every (M,x), if $M(x)$ halts, then P eventually prints out (M,x),
- for every (M,x), if $M(x)$ does NOT halt, then P never prints out (M,x).

In this context, P is called an *enumerator. (Hint: Consider the tail of a dove.)*

(b) An equivalent definition of RE is as follows: A problem belongs in RE if there exists a program P' that will output Yes when given an input x for which the answer is Yes. If the answer is No, then P'(x) may output No or loop forever. As an optional exercise, you should be able to convince yourself that this is indeed an equivalent definition.

Prove that the Halting Problem belongs in RE using this equivalent definition. In this context, P' is called a *recognizer.*

(c) As you might suspect, a problem is co-recursively enumerable (coRE) if its complement is in RE. The complement of a decision problem A is another problem A' where A'(x) is Yes iff A(x) is No, and A'(x) is No iff A(x) is Yes. State the complement of the Halting Problem.

(d) Finally, a problem belongs in the class R if it is computable, meaning there exists a program P that answers Yes when the answer is Yes, and answers No when the answer is No. By definition then, the problem is a computable function if it is computable.

We know that the TestHalt is not computable, and that the Halting Problem belongs in RE. Prove by contradiction that the Halting Problem cannot belong in coRE.

Solution:

(a)
```
for l = 1 to ∞:
  foreach input x where len(x) < l:
    foreach program M where len(M) < l:
      simulate M(x) for at most l steps
        if M(x) has halted, print out (M,x)
```

What we've essentially done is defined a program that avoids the problem of getting stuck simulating some M on some x indefinitely, and instead limits the number of steps in order to eventually get to every pair (M,x). The limit l is gradually increased, so that every $M(x)$ that does terminate will get its chance to run to completion.

(b) Consider the program $P'(M,x)$ as follows:

```
run M(x)
return Yes
```

This program P' meets the requirements since it outputs Yes when $M(x)$ halts, and loops forever otherwise.

(c) The complement of the Halting Problem is the decision problem where the desired output for input (M,x) is No if M halts on x, and Yes if M does not halt on x.

(d) Assume toward contradiction the Halting Problem is in both RE and coRE. That means there exists a recognizer Q for the Halting Problem, and a recognizer Q' for its complement. Then we can define $TestHalt(P,x)$ as follows:

```
Alternate between simulating a step of Q(P,x) and Q'(P,x)
  if Q(P,x) returns Yes, return Yes
  if Q'(P,x) returns Yes, return No
```

By construction, this will eventually print out Yes if $P(x)$ halts, and No otherwise. In other words, we are guaranteed that one of Q or Q' will eventually return Yes. However, we know that *TestHalt* is not computable, so we have a contradiction. Therefore the Halting Problem cannot be in coRE.

5 Build-Up Error?

What is wrong with the following "proof"? In addition to finding a counterexample, you should explain what is fundamentally wrong with this approach, and why it demonstrates the danger build-up error.

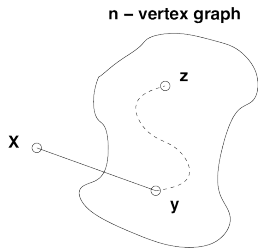
False Claim: If every vertex in an undirected graph has degree at least 1, then the graph is connected.

Proof: We use induction on the number of vertices $n \geq 1$.

Base case: There is only one graph with a single vertex and it has degree 0. Therefore, the base case is vacuously true, since the if-part is false.

Inductive hypothesis: Assume the claim is true for some $n \geq 1$.

Inductive step: We prove the claim is also true for $n+1$. Consider an undirected graph on n vertices in which every vertex has degree at least 1. By the inductive hypothesis, this graph is connected. Now add one more vertex x to obtain a graph on $(n+1)$ vertices, as shown below.



All that remains is to check that there is a path from x to every other vertex z. Since x has degree at least 1, there is an edge from x to some other vertex; call it y. Thus, we can obtain a path from x to z by adjoining the edge $\{x,y\}$ to the path from y to z. This proves the claim for $n+1$.

Solution:

The mistake is in the argument that *"every $(n+1)$-vertex graph with minimum degree 1 can be obtained from an n-vertex graph with minimum degree 1 by adding 1 more vertex"*. Instead of starting by considering an arbitrary $(n+1)$-vertex graph, this proof only considers an $(n+1)$-vertex graph that you can make by starting with an n-vertex graph with minimum degree 1. As a counterexample, consider a graph on four vertices $V = \{1,2,3,4\}$ with two edges $E = \{\{1,2\},\{3,4\}\}$. Every vertex in this graph has degree 1, but there is no way to build this 4-vertex graph from a 3-vertex graph with minimum degree 1.

More generally, this is an example of *build-up error* in proof by induction. Usually this arises from a faulty assumption that every size $n+1$ graph with some property can be "built up" from a size n graph with the same property. (This assumption is correct for some properties, but incorrect for others, such as the one in the argument above.)

One way to avoid an accidental build-up error is to use a *"shrink down, grow back"* process in the inductive step: start with a size $n+1$ graph, remove a vertex (or edge), apply the inductive hypothesis $P(n)$ to the smaller graph, and then add back the vertex (or edge) and argue that $P(n+1)$ holds.

Let's see what would have happened if we'd tried to prove the claim above by this method. In the inductive step, we must show that $P(n)$ implies $P(n+1)$ for all $n \geq 1$. Consider an $(n+1)$- vertex graph G in which every vertex has degree at least 1. Remove an arbitrary vertex v, leaving an n-vertex graph G' in which every vertex has degree... uh-oh! The reduced graph G' might contain a vertex of degree 0, making the inductive hypothesis $P(n)$ inapplicable! We are stuck —— and properly so, since the claim is false!

6 Connectivity

Consider the following claims regarding connectivity:

(a) Prove: If G is a graph with n vertices such that for any two non-adjacent vertices u and v, it holds that $\deg u + \deg v \geq n-1$, then G is connected.

[*Hint:* Show something more specific: for any two non-adjacent vertices u and v, there must be a vertex w such that u and v are both adjacent to w.]

(b) Give an example to show that if the condition $\deg u + \deg v \geq n-1$ is replaced with $\deg u + \deg v \geq n-2$, then G is not necessarily connected.

(c) Prove: For a graph G with n vertices, if the degree of each vertex is at least $n/2$, then G is connected.

(d) Prove: If there are exactly two vertices with odd degrees in a graph, then they must be in the same connected component (meaning, there is a path connecting these two vertices).

[*Hint:* Proof by contradiction.]

Solution:

(a) If u and v are two adjacent vertices, they are connected by definition. Then, consider non-adjacent u and v. Then, there must be a vertex w such that u and v are both adjacent to w. To see why, suppose this is not the case. Then, the set of neighbors of u and v has $n-1$ elements, but there are only $n-2$ other vertices. (This is the Pigeonhole Principle.) We have proven that for any non-adjacent u and v, there is a path $u \to w \to v$, and thus G is connected.

(b) Consider the graph formed by two disconnected copies of K_2. For non-adjacent u, v, it holds that $\deg u + \deg v = 2 = 4-2 = n-2$, but the graph is not connected.

(c) If each vertex's degree is at least $n/2$, then for any two non-adjacent vertices u, v,

$$\deg u + \deg v \geq \frac{n}{2} + \frac{n}{2} = n > n-1.$$

Then by part (a), the graph is connected.

(d) Suppose that they are not connected to each other. Then they must belong to two different connected components, say G_1 and G_2. Each of them will only have one vertex with odd degree. This leads to a contradiction since the sum of all degrees should be an even number.

7 Triangular Faces

Suppose we have a connected planar graph G with v vertices and e edges such that $e = 3v-6$. Prove that in any planar drawing of G, every face must be a triangle; that is, prove that every face must be incident to exactly three edges of G.

Solution:

Suppose for the sake of contradiction that we have found a planar drawing of G such that one of the faces is incident on more than three edges. Choose an arbitrary vertex on that face to call v_0, and number the other vertices around the face $v_1, v_2, \ldots, v_k$ proceeding clockwise from v_0. Since this face has at least 4 sides, we know that v_0 and v_2 do not have an edge between them. Furthermore, we know that we can add this edge to the planar drawing of G without having it cross any existing edges by just letting it cross the face. Thus, adding an edge between v_0 and v_2 results in a planar graph with v vertices and $e+1 = 3v-5$ edges. But we know that a planar graph can have at most $3v-6$ edges, so this is a contradiction. Thus, we must have that no such face exists; that is, we must have that every face in G is incident on exactly 3 edges.

8 Binary Trees

You have seen the recursive definition of binary trees from lecture and from previous classes. Here, we define binary trees in graph theoretic terms as follows (**Note:** here we will modify the definition of leaves slightly for consistency).

- A binary tree of height > 0 is a tree where exactly one vertex, called the **root**, has degree 2, and all other vertices have degrees 1 or 3. Each vertex of degree 1 is called a **leaf**. The **height** h is defined as the maximum length of the path between the root and any leaf.
- A binary tree of height 0 is the graph with a single vertex. The vertex is both a leaf and a root.

(a) Let T be a binary tree of height > 0, and let $h(T)$ denote it's height. Let r be the root in T and u and v be it's neighbors. Show that removing r from T will result in two binary trees, L, R with roots u and v respectively. Also, show that $h(T) = \max(h(L), h(R)) + 1$

(b) Using the graph theoretic definition of binary trees, prove that the number of vertices in a binary tree of height h is at most $2^{h+1} - 1$

(c) Prove that all binary trees with n leaves have $2n - 1$ vertices

Solution:

(a) Since r has degree 2, removing it will break T into two connected components, call them L and R. By symmetry, we just need to prove that L is a binary tree. Without loss of generality, suppose $u \in L$. Before removing r, u had degree 1 or 3. If u had degree 1, then after removing r, u is a single vertex, and so is a binary tree of height 0, and also is a root. If u had degree 3, then after removing r, u has degree 2, and all other vertices in L have degree 1 or 3. Thus, L is a binary tree with root u.

To prove that $h(T) = \max(h(L), h(R)) + 1$, we note that because T is a tree, any path from r to a leaf must go through either u or v but not both. Thus the maximum distance from r to any leaf is one plus either the maximum distance from u to any leaf in L (as the path cannot go back through r) or the maximum distance from v to any leaf in R. Formally, if we define $\mathscr{L}(L)$ and $\mathscr{L}(R)$ to be the set of leaves in L and R respectively, and $d(r, l)$ as the length of the path from r to some leaf l, then we have

$$\begin{aligned} h(T) &= \max(1 + \max_{l \in \mathscr{L}(L)} (d(u, l)), 1 + \max_{l \in \mathscr{L}(R)} (d(u, l))) \\ &= 1 + \max(h(L), h(R)) \end{aligned}$$

(b) Induction: **Base Case** a binary tree of height 0 is a singleton and so has $2^1 - 1 = 1$ vertex. **Inductive Hypothesis:** assume for all $k < h$, a binary tree of height k has at most $2^{k+1} - 1$ vertices. **Inductive Step:** By part a, we can remove the root from a binary tree and obtain two

binary trees: L, and R of height k and l respectively. Since $h(T) = \max(h(L), h(R)) + 1$, we know that $k, l < h$ so we can apply the inductive hypothesis to L and R. Thus, we have that the number of vertices in T is at most $1 + 2^{k+1} - 1 + 2^{l+1} - 1 \leq 2^h * 2 - 1$.

(c) Induction: **Base Case** if a binary tree has one leaf, it is a singleton and so has $1 = 2 * 1 - 1$ vertices. **Inductive Hypothesis:** assume for all $k < n$, a binary tree with k leaves has $2k - 1$ vertices. **Inductive Step:** For a binary tree, T, with $n > 1$ leaves, remove the root, r and break T into binary trees L and R. Suppose L has a leaves and R has b leaves. Note that all the leaves of T are in L or R, as $n > 1$ implies the root is not a leaf, which means $a + b = n$. By the inductive hypothesis, L has $2a - 1$ vertices, and R has $2b - 1$ vertices, and so the number of vertices in T is $2a - 1 + 2b - 1 + 1 = 2(a + b) - 1 = 2n - 1$.

9 Euclid's Algorithm

(a) Use Euclid's algorithm from lecture to compute the greatest common divisor of 527 and 323. List the values of x and y of all recursive calls.

(b) Use extended Euclid's algorithm from lecture to compute the multiplicative inverse of 5 mod 27. List the values of x and y and the returned values of all recursive calls.

(c) Find $x \pmod{27}$ if $5x + 26 \equiv 3 \pmod{27}$. You can use the result computed in (b).

(d) Assume a, b, and c are integers and $c > 0$. Prove or disprove: If a has no multiplicative inverse mod c, then $ax \equiv b \pmod{c}$ has no solution.

Solution:

(a) The values of x and y of all recursive calls are (you can get full credits without the column of x mod y):

Function Calls	(x, y)	x mod y
#1	$(527, 323)$	204
#2	$(323, 204)$	119
#3	$(204, 119)$	85
#4	$(119, 85)$	34
#5	$(85, 34)$	17
#6	$(34, 17)$	0
#7	$(17, 0)$	—

Therefore, $\gcd(527, 323) = 17$.

(b) To compute the multiplicative inverse of 5 mod 27, we first call `extended-gcd(27,5)`. Note that `(x div y)` in the pseudocode means $\lfloor x/y \rfloor$. The values of x and y of all recursive calls are (you can get full credits without the columns of x div y and x mod y):

Function Calls	(x,y)	x div y	x mod y
#1	$(27,5)$	5	2
#2	$(5,2)$	2	1
#3	$(2,1)$	2	0
#4	$(1,0)$	—	—

The returned values of all recursive calls are:

Function Calls	(d,a,b)	Returned Values
#4	—	$(1,1,0)$
#3	$(1,1,0)$	$(1,0,1)$
#2	$(1,0,1)$	$(1,1,-2)$
#1	$(1,1,-2)$	$(1,-2,11)$

Therefore, we get $1 = (-2) \times 27 + 11 \times 5$ and

$$1 = (-2) \times 27 + 11 \times 5 \equiv 11 \times 5 \pmod{27},$$

so the multiplicative inverse of 5 mod 27 is 11.

(c)

$$\begin{aligned} 5x + 26 \equiv 3 \pmod{27} &\Rightarrow 5x \equiv 3 - 26 \pmod{27} \\ &\Rightarrow 5x \equiv -23 \pmod{27} \\ &\Rightarrow 5x \equiv 4 \pmod{27} \\ &\Rightarrow 11 \times 5x \equiv 11 \times 4 \pmod{27} \\ &\Rightarrow x \equiv 44 \pmod{27} \\ &\Rightarrow x \equiv 17 \pmod{27}. \end{aligned}$$

(d) False. We can have a counterexample: $a = 3$, $b = 6$, and $c = 12$, so a has no multiplicative inverse mod c (because $a = 3$ and $c = 12$ are not relatively prime). However, $3x \equiv 6 \pmod{12}$ has solutions $x = 2, 6, 10 \bmod 12$.

10 GCD Proof

Let n, x be positive integers. Prove that x has a multiplicative inverse modulo n if and only if $\gcd(n,x) = 1$. (Hint: Remember an iff needs to be proven both directions. The gcd cannot be 0 or negative.)

Solution: If x has a multiplicative inverse modulo n, then $\gcd(n,x) = 1$.

Given that x has a multiplicative inverse modulo n, we can proceed as follows:

Assume for the sake of contradiction that the gcd, d, is greater than 1.

$$\begin{aligned} xa &\equiv 1 \pmod{n} \\ xa &= bn + 1 \\ \frac{xa}{d} &= \frac{bn+1}{d} \\ \frac{xa}{d} &= \frac{bn}{d} + \frac{1}{d} \end{aligned}$$

We've reached a contradiction because xa/d and bn/d must both be integers, however, $1/d$ is not. Therefore we've reached a contradiction, and because the gcd cannot be 0 or negative, it must be 1.

If $\gcd(n,x) = 1$, then x has a multiplicative inverse modulo n. One proof using bijections is already shown in lecture. Here's another one based on egcd:

We know that when we run egcd on n and x, the output is integers $a, b \in \mathbb{Z}$ such that

$$\begin{aligned} an + bx &= 1, \\ bx &\equiv 1 \pmod{n}. \end{aligned}$$

Thus, x has a multiplicative inverse b.

11 Homework Process and Study Group

You must describe your homework process and study group in order to receive credit for this question.