Homework 1: Introduction to Proofs – Solutions

Due: February 3, 2026

This homework must be typed in LaTeX. Please use the code released as part of the homework, remove these instructions and fill in the solutions.

Please ensure that your solutions are complete, concise, and communicated clearly. Use full sentences and plan your presentation before you write. Except where indicated, consider every problem as asking for a proof.

Your solutions must be submitted using Gradescope. When doing so, please correctly mark the pages with the solutions to each problem.

Problem 1. Let $f : A \to B$ and $g : B \to C$ be functions. Prove or disprove the following statement:

If $g \circ f$ is bijective, then f is injective and g is surjective.

Solution. This statement is $\boxed{\text{true}}$. If you want a non-trivial example of two functions satisfying f and g (at least one that doesn't make f and g both bijective), let $A = C = \mathbb{R}$ and $B = \mathbb{R}^2$. Then, with f and g defined as follows we have that,

$$f(x) := (x, 0)$$
$$g(x, y) := x$$

$g \circ f$ is the identity map. However, showing this example isn't enough to show that this is *always* the case, but it does provide nice intuition!

Proving f is injective. Suppose, for the sake of contradiction that f is not injective. Then, there exists two values $x, y \in A$ such that $x \neq y$ but $f(x) = f(y)$. Therefore, we have that

$$(g \circ f)(x) = g(f(x)) = g(f(y)) = (g \circ f)(y)$$

Therefore, $g \circ f$ is not injective, which contradicts the assumption that the map is bijective. Thus, f must be injective.

Proving g is surjective. Suppose, for the sake of contradiction that g is not surjective. Then, there exists a value $c \in C$ such that for all $b \in B$, $g(b) \neq c$. Therefore, there exists no $a \in A$ such that $(g \circ f)(a) = c$, which contradicts the assumption that the map is bijective. Thus g must be surjective. □

Problem 2. Prove or disprove the following statement:

The square for every odd number can be expressed as $8k + 1$ for some integer k.

Solution. The statement is $\boxed{\text{true}}$. Let n be odd: this means that it can be expressed $n = 2m + 1$, for some integer m. Squaring,

$$n^2 = (2m + 1)^2 = 4m^2 + 4m + 1.$$

Factoring out the 4, we can rewrite this as

$$n^2 = 4(m^2 + m) + 1.$$

Consider $m^2 + m$: this can be factored as $m(m+1)$, which is a product of two consecutive integers. Because they are consecutive, this is a product of an even and odd integer, which is always even. Therefore, we can find some integer k so that $m(m + 1) = 2k$. Plugging this back in,

$$n^2 = 4(m^2 + m) + 1$$

$$n^2 = 4(2k) + 1$$

$$n^2 = 8k + 1$$

which is exactly the statement we want to prove. □

Problem 3. You are given a rectangular chocolate bar with $m \times n$ squares of chocolate, and our task is to divide it into mn individual squares. You are only allowed to split one piece of chocolate at a time using a vertical or a horizontal break. For example, suppose that the chocolate bar is 2×2. The first split makes two pieces, both 2×1. Each of these pieces requires one more split to form single squares. This gives a total of three splits. Use an induction argument to prove the correctness of the following statement:

$mn - 1$ splits are sufficient to divide a rectangular chocolate bar with $m \times n$ squares into individual squares.

Solution. Strong induct on the size of the chocolate bar, mn. Let $P(m, n)$ be the proposition that a chocolate bar with $m \times n$ squares requires at most $mn - 1$ splits.

Base Case: The base case is $mn = 1$, or $m = n = 1$. This holds trivially because we already start with an individaul square, so $0 = 1 \cdot 1 - 1$ splits are needed.

Inductive Hypothesis: Suppose that P holds on all m, n satisfying $mn \leq k$.

Induction Step: We want to show that P holds for all m, n satisfying $mn = k+1$. Since $k+1 > 1$, one of m, n must be greater than 1: without loss of generality, take $m > 1$. Then, split along any horizontal break to split the bar into a $p \times n$ and $q \times n$ piece, where $p + q = m$ and $p, q > 1$. Now, we have two chocolate bars with size $pn, qn < mn = k + 1$. This is the same as $pn, qn \leq k$, so we can apply the inductive hypothesis and conclude that it is possible to split the $p \times n$ and $q \times n$ chocolate bar in at most $pn - 1$ and $qn - 1$ moves, respectively. We now have a way to split the $m \times n$ bar in at most

$$1 + (pn - 1) + (qn - 1) = (p + q)n - 1 = mn - 1$$

splits. Thus, we have shown $P(m, n)$ is true for all m, n satisfying $m \times n = k + 1$, which completes the induction.

□

Problem 4. Consider the algorithm given as pseudocode below:

1. What is the output of the algorithm? Provide an informal but precise description.
2. Prove the correctness of the algorithm.
3. Analyze the running time of the algorithm.

Algorithm 1 ?-?

```
Input: An $n$-vertex graph represented by an adjacency matrix $\mathbf{D}$, where $\mathbf{D}[i][j]$ is
the non-negative weight of the edge from vertex $i$ to vertex $j$, for $0 \le i, j < n$.
If there is no edge connecting $i$ and $j$, $\mathbf{D}[i][j] = \infty$.
Output: ?
for $i \leftarrow 0$ to $n-1$ do                          ▷ Initialization solution
    for $j \leftarrow 0$ to $n-1$ do
        $\mathbf{S}[i][j] \leftarrow \mathbf{D}[i][j]$
for $k \leftarrow 0$ to $n-1$ do
    for $i \leftarrow 0$ to $n-1$ do
        for $j \leftarrow 0$ to $n-1$ do
            $\mathbf{S}[i][j] \leftarrow \min\{\mathbf{S}[i][j], \mathbf{S}[i][k] + \mathbf{S}[k][j]\}$
```

Solution.

1. The algorithm returns a matrix S, where $S[i][j]$ represents the weight of the shortest path between i and j in the graph represented by D (where we view weights here as distance).

   Remark: This algorithm is known as the **Floyd Warshall** algorithm.

2. Here, we choose to induct on k in the outermost for-loop in the main logic. The claim is that after the mth iteration, $S[i][j]$ will represent the length of the shortest path from i to j that uses only the vertices $\{0, \cdots, m-1\}$ as intermediate vertices. Formally, if our path is $\{p_0, p_1, \cdots, p_d\}$, where $p_0 = i$ and $p_d = j$, then $p_1, \cdots, p_{d-1} \in \{0, \cdots m-1\}$.

   For the base case of $k = 1$, we wish to show that $S[i][j]$ is the shortest path from i to j passing through only 0 as an intermediate vertex. This is true: if the path uses 0 as an intermediate vertex, it must look like $i \to 0 \to j$, so this is represented by $S[i][0] + S[0][j]$. Then, we take the minimum with the current value of $S[i][j]$, which is the weight of the edge from i to j, meaning $S[i][j]$ now holds the length of the shortest path from i to j with all intermediate vertices in the set $\{0\}$.

   Now suppose the hypothesis holds for the mth iteration. We wish to show that it holds after the $m+1$th iteration. If we don't pass through vertex m at all, then the shortest path is precisely $S[i][j]$, since now the intermediate vertex set is limited to $\{0, \cdots, m-1\}$. Suppose now that the path passes through m. It is clear that we don't want any cycles in this path, so the path from i to m and the path from m to j must not pass through m again. However, this means that the intermediate vertices from $i \to m$ and $m \to j$ are precisely in the set $\{0, \cdots, m-1\}$! This means we can use the inductive hypothesis and conclude the shortest length path from $i \to m$ is $S[i][m]$ and from $m \to j$ is $S[m][j]$. Thus, in the case that we pass through m, the shortest distance is $S[i][m] + S[m][j]$. Therefore, setting $S[i][j] = \min\{S[i][j], S[i][m]+S[m][j]\}$ properly represents the shortest path from i to j using only intermediate vertices from $\{0, \cdots, m\}$, completing the induction.

To finish, we note that after all the iterations of the outermost for-loop, $S[i][j]$ represents the shortest path between i and j using intermediate vertices in $\{0, \cdots, n-1\}$. This is now the entire vertex set in our graph, so it is the same as the shortest path between i and j in the graph, as claimed in (a).

3. The running time of the algorithm is $O(n^3)$: the initialization of S requires two nested for loops, which is $O(n^2)$, and the main logic of the algorithm is a triple nested for loop with an $O(1)$ operation in each loop, which means it is in $O(n^3)$. Thus, the overall runtime is $O(n^2) + O(n^3) \in O(n^3)$.

$\square$

Problem 5. A *bit*, as you probably know, is a value that is either 0 or 1. A nonnegative integer can be written in binary, which means base two, and the binary representation can be interpreted as a sequence of bits. For example, the number thirteen is represented in binary as 1011. The rightmost position is the one's position, the second-to-rightmost position is the two's position, the third-to-rightmost position is the four's position, and so on.
A *bitwise operator* operates on nonnegative integers by interpreting the integers as sequences of bits.

- AND (): The AND of two nonnegative integers is the integer whose binary representation obeys the following: for each bit position, the output bit is the AND of the two input bits in the same position.
  Example: `1100 & 1010 = 1000`
- OR (): The OR of two nonnegative integers is the integer whose binary representation obeys the following: for each bit position, the output bit is the OR of the two input bits in the same position.
  Example: `1100   1010 = 1110`
- Exclusive OR (aka XOR, , ^): The XOR of two nonnegative integers is the integer whose binary representation obeys the following: for each bit position, the output bit is the XOR of the two input bits in the same position.
  Example: `1100 ^ 1010 = 0110`

Bitwise AND, OR, and XOR are commutative and associative.

Here are two more operators that interpret values as binary:

- Bit right shift (): The left operand value, interpreted as a sequence of bits, is shifted right by the number n of bits specified by the right operand value. Because the output is required to be an integer, the n lowest-order bit are discarded.
  Example: `11010 >> 2 = 110`
- Bit left shift (): The left operand value, interpreted as a sequence of bits, is shifted left by the number n of bits specified by the right operand value. Zeros are inserted and become the rightmost n bits.
  Example: `1011 << 2 = 101100`

1. Evaluate the following expressions:
   (i) 1000111111
   (ii) (1111115)5
   (iii) 100110111000
2. What are the identity elements of bitwise AND, OR, and XOR?
3. Identify the bitwise operation `x ? y` equivalent to the following operations in Python:
   (i) `x * (2 ** y)`
   (ii) `x // (2 ** y)`
4. Say we have a large positive integer x. How can we find x mod 2^4 without using mod (instead using a binary operation)?

Solution. .

1. (a) $1000 \oplus 111 \oplus 11 \oplus 1 = 1101$
   (b) $(111111 \gg 5) \ll 5 = 100000$
   (c) $1001\&1011|1000 = 1001$
2. (a) **AND:** *n-bit numbers* of all 1s
   (b) **OR:** *n-bit numbers* of all 0s
   (c) **XOR:** *n-bit numbers* of all 0s
3. (a) $x \ll y$
   (b) $x \gg y$
4. To find $x \bmod 2^4$ without using mod, we can take AND with 1111, since it is the identity on the last 4 digits.

□