CS 70 Discrete Mathematics and Probability Theory
Summer 2018 Sinho Chewi and Vrettos Moulos

HW 4

Sundry

Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. (In case of homework party, you can also just describe the group.) How did you work on this homework? Working in groups of 3-5 will earn credit for your "Sundry" grade.

Please copy the following statement and sign next to it:

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up. (*signature here*)

1 Countability Introduction

(a) Do $(0,1)$ and $\mathbb{R}_+ = (0,\infty)$ have the same cardinality? If so, give an explicit bijection (and prove that it's a bijection). If not, then prove that they have different cardinalities.

(b) Is the set of English strings countable? (Note that the strings may be arbitrarily long, but each string has finite length.) 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. Consider the bijection $f : (0,1) \to (0,\infty)$ given by

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

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.

f is onto: take any $y \in \mathbb{R}_+$. Let $x = 1/(1+y) \in (0,1)$. 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 $\mathbb{R}_+$, so they have the same cardinality. (In fact, they are both uncountable.)

(b) Countable.

We can enumerate the strings as follows: list all strings of length 1, and then all strings of length 2, and then strings of length 3, and so forth. At each step, there are only finitely many strings of a particular length ℓ, so this algorithm never gets "stuck" at any length.

(c) No, the strings are still countable, 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

$$a, aa, aaa, \ldots$$

also does not end.

Let $\mathscr{A} = \{a_1, a_2, \ldots\}$ denote the alphabet. (We are making use of the fact that the alphabet is countably infinite when we assume there is such an enumeration.) 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.

(c) List all strings containing only characters in $\{a_1, a_2, a_3\}$ which are of length at most 3.

(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.

In fact, the above idea should look somewhat familiar to you: this is the same method we used to show that $\mathbb{Z} \times \mathbb{Z}$ (and therefore $\mathbb{Q}$) is countable. We produce a table

	$\{a_1\}$	$\{a_1, a_2\}$	$\cdots$
1	↙	↙	↙
2	↙	↙	
$\vdots$	↙		

where the column represents which characters we are allowed to use in the string, and the row represents the maximum length of the string. To enumerate all of the strings, we proceed diagonally through the table.

2 Counting Cartesian Products

For two sets A and B, define the Cartesian Product as $A \times B = \{(a,b) : a \in A, b \in B\}$.

(a) Given two countable sets A and B, prove that $A \times B$ is countable.

(b) Given a finite number of countable sets $A_1, A_2, \ldots, A_n$, prove that $A_1 \times A_2 \times \cdots \times A_n$ is countable.

(c) Consider an infinite number of countable sets: $B_1, B_2, \ldots$. Under what condition(s) is $B_1 \times B_2 \times \cdots$ countable? Prove that if this condition is violated, $B_1 \times B_2 \times \cdots$ is uncountable.

Solution:

(a) As shown in lecture, $\mathbb{N} \times \mathbb{N}$ is countable by creating a zigzag map that enumerates through the pairs: $(0,0), (1,0), (0,1), (2,0), (1,1), \ldots$. Since A and B are both countable, there exists a bijection between each set and a subset of $\mathbb{N}$. Thus we know that $A \times B$ is countable because there is a bijection between a subset of $\mathbb{N} \times \mathbb{N}$ and $A \times B : f(i,j) = (A_i, B_j)$. We can enumerate the pairs (a,b) similarly.

(b) Proceed by induction.
Base Case: $n = 2$. We showed in part (a) that $A_1 \times A_2$ is countable since both A_1 and A_2 are countable.
Induction Hypothesis: Assume that for some $n \in \mathbb{N}$, $A_1 \times A_2 \times \cdots \times A_n$ is countable.
Induction Step: Consider $A_1 \times \cdots \times A_n \times A_{n+1}$. We know from our hypothesis that $A_1 \times \cdots \times A_n$ is countable, call it $C = A_1 \times \cdots \times A_n$. We proved in part (a) that since C is countable and A_{n+1} are countable, $C \times A_{n+1}$ is countable, which proves our claim.

(c) Trivially, if any of the B_i are empty, then the infinite Cartesian Product is also empty. Hence, we assume that none of the B_i are empty.

A necessary and sufficient condition for the infinite Cartesian Product to be countable is for all but finitely many of the B_i to have exactly one element.

Necessary: Suppose not; then, infinitely many B_i have at least two elements. Notice that for any B_i with only one element, every element of the infinite Cartesian Product must necessarily use the single element of B_i. Therefore, we can ignore each B_i with only one element and assume that every B_i has at least two elements. Proceed with a diagonalization argument by assuming for the sake of contradiction that $B_1 \times B_2 \times \cdots$ is countable and its elements can be

enumerated in a list:

$$(b_{1,1}, b_{2,1}, b_{3,1}, b_{4,1}, \dots)$$
$$(b_{1,2}, b_{2,2}, b_{3,2}, b_{4,2}, \dots)$$
$$(b_{1,3}, b_{2,3}, b_{3,3}, b_{4,3}, \dots)$$
$$(b_{1,4}, b_{2,4}, b_{3,4}, b_{4,4}, \dots)$$
$$\vdots$$

where $b_{i,j}$ represents the item from set B_i that is included in the jth element of the Cartesian Product. Now consider the element $(\overline{b_{1,1}}, \overline{b_{2,2}}, \overline{b_{3,3}}, \overline{b_{4,4}}, \dots)$, where $\overline{b_{i,j}}$ represents any item from set B_i that differs from $b_{i,j}$ (i.e. any other element in the set). This is a valid element that should exist in the Cartesian Product $B_1, B_2, \dots$, yet it is not in the enumerated list. This is a contradiction, so $B_1, B_2, \dots$ must be uncountable.

Notice that this relies on the fact that each set B_i has some other item we can choose when constructing our diagonal element.

Sufficient: If all but finitely many of the B_i have one element, then there are only a finite number of B_i which have more than one element. Again, we can ignore the B_i with only one element. By the previous part, we showed that the Cartesian Product of a finite number of countable sets is countable, so the infinite Cartesian Product in this case is countable.

3 Impossible Programs

Show that none of the following programs can exist.

(a) Consider a program P that takes in any program F, input x and output y and returns true if $F(x)$ outputs y and returns false otherwise.

(b) Consider a program P that takes in any program F and returns true if $F(F)$ halts and returns false if it doesn't halt.

(c) Consider a program P that takes in any programs F and G and returns true if F and G halt on all the same inputs and returns false otherwise.

Solution:

(a) If P exists, we can solve the halting problem. We show this by constructing $\texttt{HALT}(F,x)$ where F is a program and x is the input. We will use P as a subroutine to derive a contradiction.

```
def HALT(F, x):
  y = 0 # arbitrarily chosen
  def F_prime(x):
    F(x)
```

```
    return y
  return P(F_prime, x, y)
```

We modify F to create F that runs $F(x)$ and if $F(x)$ halts it outputs an arbitrarily chosen output y. We then call $P(f,x,y)$ and if it returns true then $F(x)$ halts and if it returns false then $F(x)$ must not halt. Therefore we have solved the halting problem. This is a contradiction because the halting problem is uncomputable. Therefore the program P cannot exist.

(b) If P exists, we can solve the halting problem. We show this by constructing $\texttt{HALT}(F,x)$ where F is a program and x is the input. We will use P as a subroutine to derive a contradiction.

```
def HALT(F, x):
  def F_prime(ignore):
    return F(x)
  return P(F_prime)
```

We construct a function F which ignores its input and simply runs $F(x)$. We then call $P(F)$. If $P(F')$ returns true then $F(x)$ must have halted, otherwise $P(F')$ will have returned false. Therefore, we have solved the halting problem. This is a contradiction because the halting problem is uncomputable. Therefore, the program P cannot exist.

(c) If P exists, we can solve the halting problem. We show this by constructing $\texttt{HALT}(F,x)$ where F is a program and x is the input. We will use P as a subroutine to derive a contradiction.

```
def HALT(F, x):
  def F_prime(y):
    F(x)
    while x != y:
      pass
    return

  def G(y):
    while x != y:
      pass
    return
  return P(F_prime, G)
```

We construct functions G and F'. Both functions loop forever unless the input is x. Additionally, F' runs $F(x)$ and so only halts if $F(x)$ halts. We then call $P(F',G)$, and if the answer is true, then the the F halts on x. Otherwise, it does not halt on x. Therefore, we have solved the halting problem. This is a contradiction because the halting problem is uncomputable. Therefore, the program P cannot exist.

4 Printing All x where $M(x)$ Halts

(a) Prove that it is possible to write a program P which:

- takes as input M, a Java program,
- runs forever, and prints out strings to the console,
- for every x, if $M(x)$ halts, then $P(M)$ eventually prints out x,
- for every x, if $M(x)$ does NOT halt, then $P(M)$ never prints out x.

(b) Lexicographical ordering of strings means (1) shorter strings are in front of longer strings and (2) for two strings of the same length, they are sorted in alphabetical order.

Prove that it's impossible to solve the above problem if we require the output be in lexicographical order.

Solution:

(a)
```
let S = {}
for i = 1 to ∞:
    let N_i = a new machine loaded with program M and input i
    S = S∪{N_i}

    simulate every machine in S for 1 cycle

    foreach N_x in S that has halted:
        print out x
        remove N_x from S
```

Consider any x, such that $M(x)$ halts after n steps. For some k, at stage k, $M(x)$ is added to S. At stage $k+n+1$, $M(x)$ halts, and x is then printed out.

For any x where $M(x)$ does not halt, x is never printed out.

What we've essentially done is defined a program that takes in a program M and staggers all the possible inputs on different machines. It runs $M(x_1)$ one step (and prints x_1 if halted), then runs $M(x_1)$ one more step and runs $M(x_2)$ one step (and prints x_1 if $M(x_1)$ has halted, x_2 if $M(x_2)$ has halted), and so on, iterating through each additional x input one additional step and printing out all of the x_s that have halted at that point.

(b) Let M be any Java program. We show how to construct a program which can decide whether $M(x)$ halts for any x. Consider the program M' defined as follows:

```
M'(2k)   = simulate M(k)
M'(2k+1) = halt
```

Now, suppose there is a lexicographical enumerator for M'. Call this enumerator E.

To decide whether $M(x)$ halts, we run E until we see $2k+1$ printed on the output tape. It must eventually be printed since $M'(2k+1)$ halts.

Then, we see whether $2k$ was printed out the tape already.

If so, we know that $M'(2k)$ halted, meaning that $M(k)$ halts.

If not, we know that $M(k)$ loops forever.

5 Counting, Counting, and More Counting

The only way to learn counting is to practice, practice, practice, so here is your chance to do so. For this problem, you do not need to show work that justifies your answers. We encourage you to leave your answer as an expression (rather than trying to evaluate it to get a specific number).

(a) How many ways are there to arrange n 1s and k 0s into a sequence?

(b) A bridge hand is obtained by selecting 13 cards from a standard 52-card deck. The order of the cards in a bridge hand is irrelevant.
How many different 13-card bridge hands are there? How many different 13-card bridge hands are there that contain no aces? How many different 13-card bridge hands are there that contain all four aces? How many different 13-card bridge hands are there that contain exactly 6 spades?

(c) Two identical decks of 52 cards are mixed together, yielding a stack of 104 cards. How many different ways are there to order this stack of 104 cards?

(d) How many 99-bit strings are there that contain more ones than zeros?

(e) An anagram of FLORIDA is any re-ordering of the letters of FLORIDA, i.e., any string made up of the letters F, L, O, R, I, D, and A, in any order. The anagram does not have to be an English word.
How many different anagrams of FLORIDA are there? How many different anagrams of ALASKA are there? How many different anagrams of ALABAMA are there? How many different anagrams of MONTANA are there?

(f) How many different anagrams of ABCDEF are there if: (1) C is the left neighbor of E; (2) C is on the left of E.

(g) We have 9 balls, numbered 1 through 9, and 27 bins. How many different ways are there to distribute these 9 balls among the 27 bins? Assume the bins are distinguishable (e.g., numbered 1 through 27).

(h) We throw 9 identical balls into 7 bins. How many different ways are there to distribute these 9 balls among the 7 bins such that no bin is empty? Assume the bins are distinguishable (e.g., numbered 1 through 7).

(i) How many different ways are there to throw 9 identical balls into 27 bins? Assume the bins are distinguishable (e.g., numbered 1 through 27).

(j) There are exactly 20 students currently enrolled in a class. How many different ways are there to pair up the 20 students, so that each student is paired with one other student?

(k) How many solutions does $x_0 + x_1 + \cdots + x_k = n$ have, if each x must be a non-negative integer?

(l) How many solutions does $x_0 + x_1 = n$ have, if each x must be a *strictly positive* integer?

(m) How many solutions does $x_0 + x_1 + \cdots + x_k = n$ have, if each x must be a *strictly positive* integer?

Solution:

(a) $\binom{n+k}{k}$

(b) We have to choose 13 cards out of 52 cards, so this is just $\binom{52}{13}$.

We now have to choose 13 cards out of 48 non-ace cards. So this is $\binom{48}{13}$.

We now require the four aces to be present. So we have to choose the remaining 9 cards in our hand from the 48 non-ace cards, and this is $\binom{48}{9}$.

We need our hand to contain 6 out of the 13 spade cards, and 7 out of the 39 non-spade cards, and these choices can be made separately. Hence, there are $\binom{13}{6}\binom{39}{7}$ ways to make up the hand.

(c) If we consider the 104! rearrangements of 2 identical decks, since each card appears twice, we would have overcounted each distinct rearrangement. Consider any distinct rearrangement of the 2 identical decks of 52 cards and see how many times this appears among the rearrangement of 104 cards where each card is treated as different. For each identical pair (such as the two Ace of spades), there are two ways they could be permuted among each other (since $2! = 2$). This holds for each of the 52 pairs of identical cards. So the number 104! overcounts the actual number of rearrangements of 2 identical decks by a factor of 2^{52}. Hence, the actual number of rearrangements of 2 identical decks is $104!/2^{52}$.

(d) **Answer 1:** There are $\binom{99}{k}$ 99-bit strings with k ones and $99-k$ zeros. We need $k > 99-k$, i.e. $k \geq 50$. So the total number of such strings is $\sum_{k=50}^{99}\binom{99}{k}$.
This expression can however be simplified. Since $\binom{99}{k} = \binom{99}{99-k}$, we have

$$\sum_{k=50}^{99}\binom{99}{k} = \sum_{k=50}^{99}\binom{99}{99-k} = \sum_{l=0}^{49}\binom{99}{l}$$

by substituting $l = 99-k$. Now $\sum_{k=50}^{99}\binom{99}{k} + \sum_{l=0}^{49}\binom{99}{l} = \sum_{m=0}^{99}\binom{99}{m} = 2^{99}$. Hence, $\sum_{k=50}^{99}\binom{99}{k} = (1/2)\cdot 2^{99} = 2^{98}$.

Answer 2: Since the answer from above looked so simple, there must have been a more elegant way to arrive at it. Since 99 is odd, no 99-bit string can have the same number of zeros and ones. Let A be the set of 99-bit strings with more ones than zeros, and B be the set of 99-bit strings with more zeros than ones. Now take any 99-bit string x with more ones than zeros i.e. $x \in A$. If all the bits of x are flipped, then you get a string y with more zeros than ones, and so $y \in B$. This operation of bit flips creates a one-to-one and onto function (called a bijection) between A and B. Hence, it must be that $|A| = |B|$. Every 99-bit string is either in A or in B, and since there are 2^{99} 99-bit strings, we get $|A| = |B| = (1/2) \cdot 2^{99}$. The answer we sought was $|A| = 2^{98}$.

(e) This is the number of ways of rearranging 7 distinct letters and is 7!.

In this 6 letter word, the letter A is repeated 3 times while the other letters appear once. Hence, the number 6! overcounts the number of different anagrams by a factor of 3! (which is the number of ways of permuting the 3 A's among themselves). Hence, there are $6!/3!$ different anagrams.

In this 7 letter word, the letter A is repeated 4 times while the other letters appear once. Hence, the number 7! overcounts the number of different anagrams by a factor of 4! (which is the number of ways of permuting the 4 A's among themselves). Hence, there are $7!/4!$ anagrams.

In this 7 letter word, the letter A and N are each repeated 2 times while the other letters appear once. Hence, the number 7! overcounts the number of different anagrams by a factor of $2! \times 2!$ (one factor of 2! for the number of ways of permuting the 2 A's among themselves and another factor of 2! for the number of ways of permuting the 2 N's among themselves). Hence, there are $7!/(2!)^2$ different anagrams.

(f) (1) We consider CE is a new letter X, then the question becomes counting the rearranging of 5 distinct letters, and is 5!. (2) Let A be the set of all the rearranging of ABCDEF with C on the left side of E, and B be the set of all the rearranging of ABCDEF with C on the right side of E. $|A \bigcup B| = 6!, |A \cap B| = 0$. There is a bijection between A and B by construct a operation of exchange the position of C and E. Thus $|A| = |B| = 6!/2$.

(g) Each ball has a choice of which bin it should go to. So each ball has 27 choices and the 9 balls can make their choices separately. Hence, there are 27^9 ways.

(h) **Answer 1:** Since each bin is required to be non-empty, let's throw one ball into each bin at the outset. Now we have 2 identical balls left which we want to throw into 7 distinguishable bins. There are 2 cases to consider:
Case 1: The 2 balls land in the same bin. This gives 7 ways.
Case 2: The 2 balls land in different bins. This gives $\binom{7}{2}$ ways of choosing 2 out of the 7 bins for the balls to land in. Note that it is *not* 7×6 since the balls are identical and so there is no order on them.
Summing up the number of ways from both cases, we get $7 + \binom{7}{2}$ ways.

Answer 2: Since each bin is required to be non-empty, let's throw one ball into each bin at the outset. Now we have 2 identical balls left which we want to throw into 7 distinguishable bins. From class (see notes 10), we already saw that the number of ways to put k identical balls into n distinguishable bins is $\binom{n+k-1}{k}$. Taking $k=2$ and $n=7$, we get $\binom{8}{2}$ ways to do this.
EASY EXERCISE: Can you give an expression for the number of ways to put k identical balls into n distinguishable bins such that no bin is empty?

(i) Since there is no restriction on how many balls a bin needs to have, this is just the problem of throwing k identical balls into n distinguishable bins, which can be done in $\binom{n+k-1}{k}$ ways. Here $k=9$ and $n=27$, so there are $\binom{35}{9}$ ways.

(j) **Answer 1:** Let's number the students from 1 to 20. Student 1 has 19 choices for her partner. Let i be the smallest index among students who have not yet been assigned partners. Then no matter what the value of i is (in particular, i could be 2 or 3), student i has 17 choices for her partner. The next smallest indexed student who doesn't have a partner now has 15 choices for her partner. Continuing in this way, the number of pairings is $19 \times 17 \times 15 \times \cdots \times 1 = \prod_{i=1}^{10}(2i-1)$.

Answer 2: Arrange the students numbered 1 to 20 in a line. There are 20! such arrangements. We pair up the students at positions $2i-1$ and $2i$ for i ranging from 1 to 10. You should be able to see that the 20! permutations of the students doesn't miss any possible pairing. However, it counts every different pairing multiple times. Fix any particular pairing of students. In this pairing, the first pair had freedom of 10 positions in any permutation that generated it, the second pair had a freedom of 9 positions in any permutation that generated it, and so on. There is also the freedom for the elements within each pair i.e. in any student pair (x,y), student x could have appeared in position $2i-1$ and student y could have appeared in position $2i$ and also vice versa. This gives 2 ways for each of the 10 pairs. Thus, in total, these freedoms cause $10! \times 2^{10}$ of the 20! permutations to give rise to this particular pairing. This holds for each of the different pairings. Hence, 20! overcounts the number of different pairings by a factor of $10! \times 2^{10}$. Hence, there are $20!/(10! \cdot 2^{10})$ pairings.

Answer 3: In the first step, pick a pair of students from the 20 students. There are $\binom{20}{2}$ ways to do this. In the second step, pick a pair of students from the remaining 18 students. There are $\binom{18}{2}$ ways to do this. Keep picking pairs like this, until in the tenth step, you pick a pair of students from the remaining 2 students. There are $\binom{2}{2}$ ways to do this. Multiplying all these, we get $\binom{20}{2}\binom{18}{2}\ldots\binom{2}{2}$. However, in any particular pairing of 20 students, this pairing could have been generated in 10! ways using the above procedure depending on which pairs in the pairing got picked in the first step, second step, ..., tenth step. Hence, we have to divide the above number by 10! to get the number of different pairings. Thus there are $\binom{20}{2}\binom{18}{2}\ldots\binom{2}{2}/10!$ different pairings of 20 students.

You may want to check for yourself that all three methods are producing the same integer, even though they are expressed very differently.

(k) $\binom{n+k}{k}$. There is a bijection between a sequence of n ones and k plusses and a solution to the equation: x_0 is the number of ones before the first plus, x_1 is the number of ones between the

first and second plus, etc. A key idea is that if a bijection exists between two sets they must be the same size, so counting the elements of one tells us how many the other has.

(l) $n-1$. It's easy just to enumerate the solutions here. x_0 can take values $1,2,\ldots,n-1$ and this uniquely fixes the value of x_1. So, we have $n-1$ ways to do this. But, this is just an example of the more general question below.

(m) $\binom{(n-(k+1))+k}{k} = \binom{n-1}{k}$. By subtracting 1 from all $k+1$ variables, and $k+1$ from the total required, we reduce it to problem with the same form as the previous problem. Once we have a solution to that we reverse the process, and adding 1 to all the non-negative variables gives us positive variables.

6 Fermat's Wristband

Let p be a prime number and let k be a positive integer. We have beads of k different colors, where any two beads of the same color are indistinguishable.

(a) We place p beads onto a string. How many different ways are there construct such a sequence of p beads with up to k different colors?

(b) How many sequences of p beads on the string are there that use at least two colors?

(c) Now we tie the two ends of the string together, forming a wristband. Two wristbands are equivalent if the sequence of colors on one can be obtained by rotating the beads on the other. (For instance, if we have $k = 3$ colors, red (R), green (G), and blue (B), then the length $p = 5$ necklaces RGGBG, GGBGR, GBGRG, BGRGG, and GRGGB are all equivalent, because these are all rotated versions of each other.)

How many non-equivalent wristbands are there now? Again, the p beads must not all have the same color. (Your answer should be a simple function of k and p.)

[*Hint*: Think about the fact that rotating all the beads on the wristband to another position produces an identical wristband.]

(d) Use your answer to part (c) to prove Fermat's little theorem.

Solution:

(a) k^p. For each of the p beads, there are k possibilities for its colors. Therefore, by the first counting principle, there are k^p different sequences.

(b) $k^p - k$. You can have k sequences of a beads with only one color.

(c) Since p is prime, rotating any sequence by less than p spots will produce a new sequence. As in, there is no number x smaller than p such that rotating the beads by x would cause the pattern to look the same. So, every pattern which has more than one color of beads can be rotated to form $p-1$ other patterns. So the total number of patterns equivalent with some bead sequence is p. Thus, the total number of non-equivalent patterns are $(k^p - k)/p$.

(d) $(k^p - k)/p$ must be an integer, because from the previous part, it is the number of ways to count something. Hence, $k^p - k$ has to be divisible by p, i.e., $k^p \equiv k \pmod{p}$, which is Fermat's Little Theorem.