CS 70 Discrete Mathematics and Probability Theory
Spring 2021

HW 7 (Optional)

1 Count It!

For each of the following collections, determine and briefly explain whether it is finite, countably infinite (like the natural numbers), or uncountably infinite (like the reals):

(a) The integers which divide 8.

(b) The integers which 8 divides.

(c) The functions from $\mathbb{N}$ to $\mathbb{N}$.

(d) The set of strings over the English alphabet. (Note that the strings may be arbitrarily long, but each string has finite length. Also the strings need not be real English words.)

(e) Computer programs that halt. *Hint: How can we represent a computer program?*

(f) The set of finite-length strings drawn from a countably infinite alphabet, $\mathscr{A}$.

(g) The set of infinite-length strings over the English alphabet.

Solution:

(a) Finite. They are $\{-8,-4,-2,-1,1,2,4,8\}$.

(b) Countably infinite. We know that there exists a bijective function $f:\mathbb{N}\to\mathbb{Z}$. Then the function $g(n)=8f(n)$ is a bijective mapping from $\mathbb{N}$ to integers which 8 divides.

(c) Uncountably infinite. We use Cantor's Diagonalization Proof:

Let $\mathscr{F}$ be the set of all functions from $\mathbb{N}$ to $\mathbb{N}$. We can represent a function $f\in\mathscr{F}$ as an infinite sequence $(f(0),f(1),\cdots)$, where the i-th element is $f(i)$. Suppose towards a contradiction that there is a bijection from $\mathbb{N}$ to $\mathscr{F}$:

$$\begin{aligned}
0 &\longleftrightarrow (f_0(0),f_0(1),f_0(2),f_0(3),\dots)\\
1 &\longleftrightarrow (f_1(0),f_1(1),f_1(2),f_1(3),\dots)\\
2 &\longleftrightarrow (f_2(0),f_2(1),f_2(2),f_2(3),\dots)\\
3 &\longleftrightarrow (f_3(0),f_3(1),f_3(2),f_3(3),\dots)\\
&\vdots
\end{aligned}$$

Consider the function $g : \mathbb{N} \to \mathbb{N}$ where $g(i) = f_i(i) + 1$ for all $i \in \mathbb{N}$. We claim that the function g is not in our finite list of functions. Suppose for contradiction that it were, and that it was the n-th function $f_n(\cdot)$ in the list, i.e., $g(\cdot) = f_n(\cdot)$. However, $f_n(\cdot)$ and $g(\cdot)$ differ in the n-th argument, i.e. $f_n(n) \neq g(n)$, because by our construction $g(n) = f_n(n) + 1$. Contradiction!

(d) Countably infinite. 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.

(e) Countably infinite. The total number of programs is countably infinite, since each can be viewed as a string of characters (so for example if we assume each character is one of the 256 possible values, then each program can be viewed as number in base 256, and we know these numbers are countably infinite). So the number of halting programs, which is a subset of all programs, can be either finite or countably infinite. But there are an infinite number of halting programs, for example for each number i the program that just prints i is different for each i. So the total number of halting programs is countably infinite. (Note also that this result together with the previous one in (g) implies that not every function from $\mathbb{N}$ to $\mathbb{N}$ can be written as a program.)

(f) Countably infinite. 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.) 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

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

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.

(g) Uncountably infinite. We can use a diagonalization argument. First, for a string s, define $s[i]$ as the i-th character in the string (where the first character is position 0), where $i \in \mathbb{N}$ because the strings are infinite. Now suppose for contradiction that we have an enumeration of strings s_i for all $i \in \mathbb{N}$: then define the string s' as $s'[i] =$ (the next character in the alphabet after $s_i[i]$), where the character after z loops around back to a. Then s' differs at position i from s_i for all $i \in \mathbb{N}$, so it is not accounted for in the enumeration, which is a contradiction. Thus, the set is uncountable.

Alternative 1: The set of all infinite strings containing only as and bs is a subset of the set we're counting. We can show a bijection from this subset to the real interval $\mathbb{R}[0, 1]$, which proves the uncountability of the subset and therefore entire set as well: given a string in $\{a, b\}^*$, replace the as with 0s and bs with 1s and prepend $'0.'$ to the string, which produces a unique binary number in $\mathbb{R}[0, 1]$ corresponding to the string.

2 Countability Proof Practice

(a) A disk is a 2D region of the form $\{(x, y) \in \mathbb{R}^2 : (x - x_0)^2 + (y - y_0)^2 \leq r^2\}$, for some $x_0, y_0, r \in \mathbb{R}$, $r > 0$. Say you have a set of disks in $\mathbb{R}^2$ such that none of the disks overlap. Is this set always countable, or potentially uncountable?
(*Hint*: Attempt to relate it to a set that we know is countable, such as $\mathbb{Q} \times \mathbb{Q}$)

(b) A circle is a subset of the plane of the form $\{(x, y) \in \mathbb{R}^2 : (x - x_0)^2 + (y - y_0)^2 = r^2\}$ for some $x_0, y_0, r \in \mathbb{R}$, $r > 0$. Now say you have a set of circles in $\mathbb{R}^2$ such that none of the circles overlap. Is this set always countable, or potentially uncountable?
(*Hint*: The difference between a circle and a disk is that a disk contains all of the points in its interior, whereas a circle does not.)

(c) Is the set containing all increasing functions $f : \mathbb{N} \to \mathbb{N}$ (i.e., if $x \geq y$, then $f(x) \geq f(y)$) countable or uncountable? Prove your answer.

(d) Is the set containing all decreasing functions $f : \mathbb{N} \to \mathbb{N}$ (i.e., if $x \geq y$, then $f(x) \leq f(y)$) countable or uncountable? Prove your answer.

Solution:

(a) Countable. Each disk must contain at least one rational point (an (x,y)-coordinate where $x,y \in \mathbb{Q}$) in its interior, and due to the fact that no two disks overlap, the cardinality of the set of disks can be no larger than the cardinality of $\mathbb{Q} \times \mathbb{Q}$, which we know to be countable.

(b) Possibly uncountable. Consider the circles $C_r = \{(x,y) \in \mathbb{R}^2 : x^2 + y^2 = r\}$ for each $r \in \mathbb{R}$. For $r_1 \neq r_2$, C_{r_1} and C_{r_2} do not overlap, and there are uncountably many of these circles (one for each real number).

(c) Suppose that there is a bijection between $\mathbb{N}$ and the set of all increasing functions $\mathbb{N} \to \mathbb{N}$:

$$\begin{aligned}
0 &\mapsto (f_0(0), f_0(1), f_0(2), \ldots) \\
1 &\mapsto (f_1(0), f_1(1), f_1(2), \ldots) \\
2 &\mapsto (f_2(0), f_2(1), f_2(2), \ldots) \\
&\vdots
\end{aligned}$$

We will use a diagonalization argument to prove that there is a function f which is not in the above list. Define

$$f(n) = 1 + \sum_{i=1}^{n} f_i(n).$$

First, we will show that f is increasing. Indeed, if $m \leq n$, then

$$f(m) = 1 + \sum_{i=1}^{m} f_i(m) \leq 1 + \sum_{i=1}^{n} f_i(m) \leq 1 + \sum_{i=1}^{n} f_i(n) = f(n).$$

The first inequality is because each function is non-negative; the second inequality is because the f_i are increasing.

To show that f is not in the list, note that

$$f(n) = 1 + \sum_{i=1}^{n} f_i(n) \geq 1 + f_n(n) > f_n(n).$$

Since $f(n) > f_n(n)$ for each $n \in \mathbb{N}$, f cannot be any of the functions in the list. Therefore, the set of increasing functions $f : \mathbb{N} \to \mathbb{N}$ is uncountable.

(d) Given any function that begins with $f(0) = n$, consider the number of indices in which the function decreases in output: the set of i such that $f(i) < f(i-1)$. The range of f is a subset of $\mathbb{N}$ so by the well-ordering principle there must be a least element. Call this element a. Then there are only at most $n - a$ transition points. We can set a bijection for any function with $f(0) = n$ to a "word" of indices at which the function decreases. Therefore, the set of decreasing functions $\mathbb{N} \to \mathbb{N}$ has the same cardinality as the set of finite bit strings from a countably infinite alphabet, which is countable. Therefore, the set of all decreasing functions is countable.

3 Finite and Infinite Graphs

The graph material that we learned in lecture still applies if the set of vertices of a graph is infinite. We thus make a distinction between finite and infinite graphs: a graph $G = (V,E)$ is finite if V and E are both finite. Otherwise, the graph is infnite. As examples, consider the graphs

- $G_1 = (V = \mathbb{Z},\ E = \{(i,j) \in \mathbb{Z} \times \mathbb{Z} |\ |i-j| = 1\})$
- $G_2 = (V = \mathbb{Z},\ E = \{(i,j) \in \mathbb{Z} \times \mathbb{Z} |\ i < j\})$
- $G_3 = (V = \mathbb{Z}^2,\ E = \{((i,j),(k,l)) \in \mathbb{Z}^2 \times \mathbb{Z}^2 |\ (i = k \wedge |j-l| = 1) \vee (j = l \wedge |i-k| = 1)\})$

Observe that G_1 is a line of integers, G_2 is a complete graph over all integers, and G_3 is an grid of integers. Prove whether the following sets of graphs are countable or uncountable

(a) The set of all finite graphs $G = (V,E)$, for $V \subseteq \mathbb{N}$

(b) The set of all infinite graphs over a fixed, countably infinite set of vertices (in other words, they all have the same vertex set).

(c) The set of all graphs over a fixed, countably infinite set of vertices, the degree of each vertex is exactly two. For instance, every vertex in G_1 (defined above) has degree 2.

(d) We say that graphs $G = (V,E)$ and $G' = (V',E')$ are isomorphic if the exists some bijection $f : V \to V'$ such that $(u,v) \in V$ iff $(f(u), f(v)) \in V'$. Such a bijection f is called a **graph isomorphism**. Suppose we consider two graphs to be the equivalent if they are isomorphic. The idea is that if we relabel the vertices of a graph, it is still the same graph. Using this definition of "being the same graph", can you conclude that the set of trees over countably infinite vertices is countable?
(*Hint:* Begin by showing that for any graph isomorphism f, and any vertex v, $f(v)$ and v have the same degree)

Solution:

(a) Countable. Let A be the set of graphs we are counting. Let A_k be the set of all graphs $G = (V,E)$, where $V \subseteq \{1, 2, \ldots k\}$. A_k is finite because there are only 2^k possible subsets of vertices that is a subset of $\{1,2,...k\}$. For a particular vertex set of size $q \leq k$, there are $\binom{q}{2}$ possible edges over that particular vertex set. Since the number of possible graphs over the vertex set is the power set of all the possible edges to choose from, the number of possible graphs on q vertices is at most $2^{\binom{q}{2}} \leq 2^{k^2}$. There are 2^k ways of choosing a vertex set $V \subseteq \{1,2,\ldots,k\}$, so the number of graphs $|A_k|$ of at most k vertices is bounded at most $2^k \cdot 2^{k^2} = 2^{k^2+k}$. Since each graph's vertex set is a subset of $\mathbb{N}$, the graph must be must be contained in A_k for some k. Thus, $A = \cup_{k=1}^{\infty} A_k$. We can simply enumerate A by enumerating each A_k. Note we are double counting some graphs but for it purpose of showing countability, it's okay.

(b) Uncountable. The set of possible edges in a graph of countably infinite vertices is clearly infinite. The power set of any infinte set is uncountable.

(c) Uncountable.

Recall from lectures that the number of infinite-length binary strings is uncountably infinite. We will construct an injection from this set to the given set of graphs.

First observe that since the number of vertices is countable, we can label them with the positive integers: $V = \{v_i | \, i \in \{1,2,3,\ldots\}\}$. Now consider an infinite binary string $s = b_1 b_2 b_3 \ldots \in \{0,1\}^{\infty}$, where $b_i \in \{0,1\}$ is the ith digit of the string. We can encode the first digit by creating a simple cycle of length $b_1 + 3$ out of vertices $v_1, v_2, \ldots, v_{b_1+2}$ (i.e. we make a chain $v_1 \to v_2 \to \cdots \to v_{b_1+2} \to v_1$). We can also encode b_2 by creating a simple cycle of length $b_2 + 3$ out of the vertices $v_{b_1+3}, v_{b_1+4}, \ldots, v_{b_1+b_2+5}$. Proceeding in this way, we encode the ith digit by making a simple cycle of length $b_i + 3$ out of the vertices $v_{f_s(i)}$ through $v_{f_s(i)+b_i+1}$, where $f_s(i) = 3(i-1) + \sum_{j=1}^{i-1} b_j$. Each vertex in V will be part of exactly one such cycle, and so each vertex will have degree exactly two.

Two see that this is an injection, consider two distinct strings $s = b_1 b_2 b_3 \ldots$ and $s' = b'_1 b'_2 b'_3 \ldots$ that first differ in their ith digit. By construction, the subgraph formed by the first $f_s(i-1)$ vertices will be identical for the each of the graphs. However, the next $b_i + 3$ vertices will be part of a length-$(b_i + 3)$ simple cycle in the graph for s, while the next $b'_i + 3$ vertices will be part of a length-$(b'_i + 3)$ simple cycle in the graph for s', with $b'_i + 3 \neq b_i + 3$. This holds for all $s, s' \in \{0,1\}^{\infty}$, so we have an injection.

(d) Uncountable. First, to show that if f is a graph isomorphism, then v and $f(v)$ has the same degree, suppose that they do not have the same degree. Then either there is some neighbor, w, of v such that $(f(v), f(w))$ is not in E' or there is some neighbor w', of $f(v)$ such that $(v, f^{-1}(w'))$ is not in E. Either way, f is not an isomorphism.

We will inject the set of infinite bit strings into the set of infinite trees. Specifically, we will construct these trees by adding leaves to the infinite line graph $G = (V, E)$, where $V = \{v_0, v_2, \ldots\}$ and $E = \{(v_i, v_{i+1}) | \; i \in \mathbb{N}\}$. For each bit string, b, where $b(i)$ is the i^{th} bit of b, construct G_b as follows: First, add 10 leaves to v_0. Then, for each i, if $b(i) = 1$, add a leaf to v_i. We will denote each v_i in G_b as v_i^b. Clearly, the resulting graph is a tree, as adding leaves will never disconnect a graph or create cycles. The graph also has a countable number of vertices because we've only added countably many vertices. To show injection, we first note that any graph isopmorphism must map a vertex to a vertex of the same degree. Now suppose $b \neq b'$

but there is an isomorphism, f, from G_b to $G_{b'}$. By the method of construction each G_b has exactly one vertex of degree ≥ 10. Thus, any isomorphism from the vertex set of G_b to that of $G_{b'}$ must map v_0^b to $v_0^{b'}$. Thus, for each i, it must map v_i^b to $v_i^{b'}$. However, because there is some i where $b(i) \neq b'(i)$, there is some i where where $f(v_i^b)$ and $f(v_i^{b'})$ have different degrees.

4 Unions and Intersections

Given:

- A is a countable, non-empty set. For all $i \in A$, S_i is an uncountable set.
- B is an uncountable set. For all $i \in B$, Q_i is a countable set.

For each of the following, decide if the expression is "Always Countable", "Always Uncountable", "Sometimes Countable, Sometimes Uncountable".

For the "Always" cases, prove your claim. For the "Sometimes" case, provide two examples – one where the expression is countable, and one where the expression is uncountable.

(a) $A \cap B$

(b) $A \cup B$

(c) $\bigcup_{i \in A} S_i$

(d) $\bigcap_{i \in A} S_i$

(e) $\bigcup_{i \in B} Q_i$

(f) $\bigcap_{i \in B} Q_i$

Solution:

(a) Always countable. $A \cap B$ is a subset of A, which is countable.

(b) Always uncountable. $A \cup B$ is a superset of B, which is uncountable.

(c) Always uncountable. Let i' be any element of A. $S_{i'}$ is uncountable. Thus, $\bigcup_{i \in A} S_i$, a superset of $S_{i'}$, is uncountable.

(d) Sometimes countable, sometimes uncountable.

Countable: When the S_i are disjoint, the intersection is empty, and thus countable. For example, let $A = \mathbb{N}$, let $S_i = \{i\} \times \mathbb{R} = \{(i, x) \mid x \in \mathbb{R}\}$. Then, $\bigcap_{i \in A} S_i = \varnothing$.

Uncountable: When the S_i are identical, the intersection is uncountable. Let $A = \mathbb{N}$, let $S_i = \mathbb{R}$ for all i. $\bigcap_{i \in A} S_i = \mathbb{R}$ is uncountable.

(e) Sometimes countable, sometimes uncountable.

Countable: Make all the Q_i identical. For example, let $B = \mathbb{R}$, and $Q_i = \mathbb{N}$. Then, $\bigcup_{i \in B} Q_i = \mathbb{N}$ is countable.

Uncountable: Let $B = \mathbb{R}$. Let $Q_i = \{i\}$. Then, $\bigcup_{i \in B} Q_i = \mathbb{R}$ is uncountable.

(f) Always countable. Let b be any element of B. Q_b is countable. Thus, $\bigcap_{i \in B} Q_i$, a subset of Q_b, is also countable.

5 Unprogrammable Programs

Prove whether the programs described below can exist or not.

(a) A program $P(F,x,y)$ that returns true if the program F outputs y when given x as input (i.e. $F(x) = y$) and false otherwise.

(b) A program P that takes two programs F and G as arguments, and returns true if F and G halt on the same set of inputs (or false otherwise).

Solution:

(a) P cannot exist, for otherwise we could solve the halting problem:

```
def Halt(F, x):
  def Q(x):
    F(x)
    return 0
  return P(Q, x, 0)
```

`Halt` defines a subroutine `Q` that first simulates F and then returns 0, that is `Q`(x) returns 0 if $F(x)$ halts, and nothing otherwise. Knowing the output of `P`(F,x,0) thus tells us whether F(x) halts or not.

(b) We solve the halting problem once more:

```
def Halt(F, x):
  def Q(y):
    loop
  def R(y):
    If y = x:
      F(x)
    Else:
      loop
  return not P(Q, R)
```

`Q` is a subroutine that loops forever on all inputs. `R` is a subroutine that loops forever on every input except x, and runs $F(x)$ on input x when handed x as an argument. Knowing if `Q` and `R` halt on the same inputs is thus tantamount to knowing whether F halts on x (since that is the only case in which they could possibly differ). Thus, if $P(Q,R)$ returns "True", then we know they behave the same on all inputs and F must not halt on x, so we return `not P(Q,R)`.

6 Computability

Decide whether the following statements are true or false. Please justify your answers.

(a) The problem of determining whether a program halts in time 2^{n^2} on an input of size n is undecidable.

(b) There is no computer program `Line` which takes a program P, an input x, and a line number L, and determines whether the L^{th} line of code is executed when the program P is run on the input x.

Solution:

(a) False. You can simulate a program for 2^{n^2} steps and see if it halts.

Generally, we can always run a program for any fixed *finite* amount of time to see what it does. The problem of undecidability arises when no bounds on time are available.

(b) True.

We implement `Halt` which takes a program P, an input x and decides whether $P(x)$ halts, using `Line` as follows. We take the input P and modify it so that each exit or return statement jumps to a particular new line. Call the resulting program P'. We then hand that program to `Line` along with the input x and the number of the new line. If the original program halts than `Line` would return true, and if not `Line` would return false.

This contradicts the fact that the program `Halt` does not exist, so `Line` does not exist either.

At a high level, you can show the undecidability of a problem by using your program which solves the problem as a subroutine to solve a different problem that we know is undecidable. Alternatively, you can do a diagonalization proof like we did for `Halt`. The first approach is natural for computer programmers and flows from the fact that you are given P as text! Therefore you can look at it and modify it. This is what the solution above does.

7 Computations on Programs

(a) Is it possible to write a program that takes a natural number n as input, and finds the shortest arithmetic formula which computes n? For the purpose of this question, a formula is a sequence consisting of some valid combination of (decimal) digits, standard binary operators ($+$, $\times$, the "^" operator that raises to a power), and parentheses. We define the length of a formula as the number of characters in the formula. Specifically, each operator, decimal digit, or parentheses counts as one character.

(*Hint:* Think about whether it's possible to enumerate the set of possible arithmetic formulas. How would you know when to stop?)

(b) Now say you wish to write a program that, given a natural number input n, finds another program (e.g. in Java or C) which prints out n. The discovered program should have the minimum execution-time-plus-length of all the programs that print n. Execution time is measured by the number of CPU instructions executed, while "length" is the number of characters in the source code. Can this be done?

(*Hint:* Is it possible to tell whether a program halts on a given input within t steps? What can you say about the execution-time-plus-length of the program if you know that it does not halt within t steps?)

Solution:

(a) Yes it is possible to write such a program.

We already know one way to write a formula for n, which is to just write the number n (with no operators). Let the length of this formula in characters be l. In order to find the *shortest* formula we simply need to search among formulae that have length at most l.

Since there are a finite number of formulas of length at most l, we can write a program that iterates over all of them. For example, if we treat each character as a byte or an 8-bit number, the whole formula becomes a binary integer of length at most $8l$, so we can simply iterate over all binary numbers up to 2^{8l} and for each one check if it is a valid formula.

For each formula that we encounter we can compute its value in finite time (since there are no loop/control structures in formula). Therefore we can check whether it computes n, and then among those that do compute n we find the smallest one.

(b) Yes. Again it is possible to write such a program.

As before, given a number n, there is one program that we know can definitely write n, which is the program that prints the digits of n one by one. Let the length plus running time of this program be l. We only need to check programs that have a length of at most l and a running time of at most l, since otherwise their running time plus length would be bigger than l.

Similar to the previous part, we can iterate over all programs of length at most l (by treating each one as a large binary integer and checking each one's validity by e.g. compiling it). For each such program, we then run it for at most l steps. If it takes more time, we stop executing it and go to the next program, otherwise in at most l steps we see its output and we can check whether it is equal to n or not.

Now among all programs that have length at most l and execute for at most l steps and print n we find the one that has the shortest length plus execution time.

8 Kolmogorov Complexity

Compressing a bit string x of length n can be interpreted as the task of creating a program of fewer than n bits that returns x. The Kolmogorov complexity of a string $K(x)$ is the length of an optimally-compressed copy of x; that is, $K(x)$ is the length of shortest program that returns x.

(a) Explain why the notion of the "smallest positive integer that cannot be defined in under 280 characters" is paradoxical.

(b) Prove that for any length n, there is at least one string of bits that cannot be compressed to less than n bits.

(c) Say you have a program K that outputs the Kolmogorov complexity of any input string. Under the assumption that you can use such a program K as a subroutine, design another program P that takes an integer n as input, and outputs the length-n binary string with the highest Kolmogorov complexity. If there is more than one string with the highest complexity, output the one that comes first lexicographically.

(d) Let's say you compile the program P you just wrote and get an m bit executable, for some $m \in \mathbb{N}$ (i.e. the program P can be represented in m bits). Prove that the program P (and consequently the program K) cannot exist.

(*Hint*: Consider what happens when P is given a very large input n.)

Solution:

(a) Since there are only a finite number of characters then there are only a finite number of positive integers that can be defined in under 280 characters. Therefore there must be positive integers that are not definable in 280 characters and by the well-ordering principle there is a smallest member of that set. However the statement "the smallest positive integer not definable in under 280 characters" defines the smallest such an integer using only 67 characters (including spaces). Hence, we have a paradox (called the Berry Paradox).

(b) The number of strings of length n is 2^n. The number of strings shorter than length n is $\sum_{i=0}^{n-1} 2^i$. We know that sum is equal to $2^n - 1$ (remember how binary works). Therefore the cardinality of the set of strings shorter than n is smaller than the cardinality of strings of length n. Therefore there must be strings of length n that cannot be compressed to shorter strings.

(c) We write such a program as follows:

```
def P(n):
  complex_string = "0" * n
  for j in range(1, 2 ** n):
    # some fancy Python to convert j into binary
    bit_string = "0:b".format(j)
    # length should now be n characters
    bit_string = (n - len(bit_string)) * "0" + bit_string
    if K(bit_string) > K(complex_string):
      complex_string = bit_string
   return complex_string
```

(d) We know that for every value of n there must be an incompressible string. Such an incompressible string would have a Kolmogorov complexity greater than or equal to its actual length. Therefore our program P must return an incompressible string. However, suppose we choose size n_k such that $n_k \gg m$. Our program $P(n_k)$ will output a string x of length n_k that is not compressible meaning $K(x) \geq n_k$. However we have designed a program that outputs x using fewer bits than n_k. This is a contradiction. Therefore K cannot exist.