CS 70 Fall 2022 | Discrete Mathematics and Probability Theory Satish Rao and Babak Ayazifar | HW 08

Due: Saturday, 10/22, 4:00 PM
Grace period until Saturday, 10/22, 6:00 PM

## Sundry

Before you start writing your final homework submission, state briefly how you worked on it. Who else did you work with? List names and email addresses. (In case of homework party, you can just describe the group.)

I did not work with anybody to complete this homework. I went to office hours to receive guidance on problem 6 and 3. For problem 4(a), I read up on the Wikipedia article about Dovetailing, then wrote up my own solution based on my understanding of the concept. The citation (i.e. hyperlink) to the article is listed in my solution.

## 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.

**Solution:** The numbers are $\{\pm 1, \pm 2, \pm 4, \pm 8\}$ which is obviously finite.

(b) The integers which 8 divides.

**Solution:** Call $S$ the set of numbers which 8 divides. Then, we can take $\frac{S}{8}$ (i.e. dividing every element in the set by 8), which gives us the natural numbers. Therefore, this set is countably infinite, since the naturals are countably infinite.

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

**Solution:** This set is uncountably infinite, since we can imagine each function as outputting an infinite string of numbers based upon its input, for instance $f_n(1), f_n(2), f_n(3)$ and so on. Now, suppose we enumerate every natural number with a function. We can generate a new function $g$, where for every input $k$, $g(k) = f_k(k) + 1$. Therefore $g$ is different from every function $f_n$ in at least one spot, so it is not in our original enumeration. Therefore, this set is uncountably infinite.

(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.)

**Solution:** This set is finite, since having a finite length means that there are a finite number of letters and a finite length, so eventually we will run out of possible strings to write down, and thus we only reuqire a finite number of numbers to enumerate this list.

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

**Solution:** Because the alphabet is countably infinite, we can enumerate all letters in $\mathscr{C}$ with a unique natural number. Then, our problem becomes simply choosing a string of finitely many of them, which is clearly countable since the natural numbers are countable.

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

**Solution:** This set is uncountably infinite. We can form a diagonalization construction as follows: Suppose we have enumerated all infinite length strings over the English alphabet. Now, construct a new string which takes letters along the diagonal, where every letter is changed to the letter which precedes it (i.e. A $\to$ B, B $\to$ C, etc., and Z $\to$ A). Therefore, we've created a new string which is different from every other string in at least one location! Therefore, this set is uncountably infinite.

Another way to see this is the fact that the real numbers also have a finite alphabet (i.e. 0-9) but infinite in length, which is uncountable due to diagonalization. Therefore the set of infinite-length strings over a finite alphabet is always uncountable.

## 2 Countability Proof Practice

(a) A disk is a 2D region of the form $\{(x,y) \in \mathbb{R}^2 : (x-x_0)^2 + (y-y_0)^2 \le r^2\}$, 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 $\mathbb{Q} \times \mathbb{Q}$.)

**Solution:** This set is countable, because every pair of rational numbers $(a,b) \in \mathbb{Q}^2$ exists in only one disk, since the disks do not overlap. Therefore, each rational pair $(a,b)$ is uniquely within a single disk. Since $\mathbb{Q}^2 = \mathbb{Q} \times \mathbb{Q}$ is countable, then this set of disks is also countable.

(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.)

**Solution:** Consider the set of concentric circles centered at $(0,0)$, that have radius $r$. Since $r$ can be any real number (a circle can have radius equal to any real number), then for every real number we can create a non-overlapping circle (since they're all centered at $(0,0)$), and so we've created a bijection between every circle and the real numbers. Since the real numbers are uncountable, then this set is also potentially uncountable.

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

**Solution:** Let's enumerate all the functions in the following manner:

| $\mathbb{N}$ | $f : \mathbb{N} \to \mathbb{N}$ |
|---|---|
| 1 | $f_1(1), f_1(2), f_1(3), \dots$ |
| 2 | $f_2(1), f_2(2), f_2(3), \dots$ |
| 3 | $f_3(1), f_3(2), f_3(3), \dots$ |
| $\vdots$ | $\vdots$ |

with the restriction that $f_k(n) \le f_k(n+1)$ for all $k$ and $n$. Now, we can create a new function $g(x)$ as follows:

$$\begin{aligned} g(1) &= f_1(1) + f_2(1) + f_3(1) + \cdots \\ g(2) &= f_1(2) + f_2(2) + f_3(2) + \cdots \\ &\vdots \end{aligned}$$

Since for every function $f$ we know that $f_k(n) \le f_k(n+1)$, then we also know that $g$ is strictly increasing. However, now we've created a new function which is different from every $f$ in at least one location, since every input of $g(n)$ is created as a sum of all $f(n)$. Therefore, this set is uncountable.

(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:** Consider an arbitrary function $f : \mathbb{N} \to \mathbb{N}$. We know that since it is strictly decreasing, the function either equals zero or remains a constant value as $x \to \infty$. Furthermore, because $f$'s output is in $\mathbb{N}$, then this means that there are a *finite* number of decreases that $f$ can make before it reaches zero.

Now, we can enumerate every $f$ as follows: for every function $f$, let it be represented by a string consisting of its starting value and the inputs where $f$ decreases immediately followed by how much it decreases. For instance, if $f$ produces the sequence:

$$f : f(0) = 5, f(1) = 3, f(2) = 2, f(3) = 1, f(4) = 0, \dots$$

then we could enumerate $f$ by writing the string "(5)(1,2)(2,1)(3,1)(4,1)" (Note the starting value is 5, then (1, 2) denotes that at $f(1)$, $f$ decreases by 2, and so on.)

I added the parentheses to be visually clear, but we can just write this string as "512213141" as well. In any case, this string must be finitely long, since $f$ can only decreases finitely many times before reaching zero. Therefore, this is effectiely the same as choosing a finite string from a countably infinite alphabet, which from problem 1e we know is countable. Therefore, the set of all decreasing functions from $\mathbb{N} \to \mathbb{N}$ is countable.

## 3 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.

**Solution:** Define a test program `test(x, y)` which evaluates $F(x)$ and checks whether it outputs $y$ on the input $x$. In other words, it evaluates $F(x)$, then if $F(x) = y$ then it returns true, or false otherwise.

Then, we can define $P$ to be `Halt(x, y)` which returns `test(x, y)`. Then, by this construction, `Halt(x, y)` knows whether $F(x) = y$ or not, which is impossible since the halting problem is impossible to solve. Therefore, $P$ cannot exist.

(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:** Define a test program `testF(x, y)` which evaluates $F$ for a given input $x$, and returns true or false depending on whether it equals $y$.. Similarly, define another program `testG(x, y)` which evaluates $G$ at an input $x$ and returns true or false depending on whether it equals $y$.

Now, we can define another function `testFG(x, y)` which executes `testF (x, y)` and `testG(x, y)`, and returns true if both `testF` and `testG` return true. Otherwise, it returns false.

Finally, we can define $P$ to be `Halt(x, y)` which returns `testFG(x, y)`. However, this would then imply that `Halt(x, y)` knows the output of `testFG`, which is impossible since a function cannot know the output of another function. Therefore $P$ 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.)

**Solution:** We can write $P$ as follows, which takes in a list of functions $M$ and an input $x$:

```
def P:
    Generates an arbitrary function M(x) and stores it in a list
      Evaluates the next line of every program in the list of M
      If an M(x) returns an output:  print("(M, x)")
    Otherwise:  return to line 2 and generate another M
```

In this formulation, if $M(x)$ halts, then it will print out $(M,x)$ based on the conditional if statement. This way, every program $M(x)$ is also reached, since each $M(x)$ is being evaluated line by line simultaneously, so if $M$ does halt then $(M,x)$ will be printed. Otherwise, nothing will be printed since no output is reached. My solution here is based upon reading up on the Wikipedia page about Dovetailing, as suggested by the hint. Here's a link to the article: https://en.wikipedia.org/wiki/Dovetailing_(computer_science)

(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. Namely, prove that it is possible to write a program $P'$ which:

- takes as input a program $M$ and input $x$.

- if $M$ halts on input $x$, then $P'$ should print "Yes".
- if $M$ does not halt on input $x$, then $P'$ may output "No" or loop forever.

In this context, $P'$ is called a *recognizer*.

**Solution:** $P'$ can be written as follows:

```
def P(M, x)
    For all M:
      Evaluates M(x)
    If M(x) returns an output:
      print("Yes")
    else:
      print ("No")
```

In this formulation, if $M$ halts on input $x$, then it will return some output, and thus will print "Yes". Otherwise, if $M$ does not halt, then it will not return a fixed output, and therefore will either print "No" or loop forever.

(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.

**Solution:** The complement of the Halting problem would be that given a program $M$ with an input $x$, if $M$ halts on input $x$ then we return "No" and if $M$ does not halt we return "Yes".

(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:** Suppose that the Halting Problem does belong in coRE. Therefore, its complement would then exist as well. However, as we've shown in part (c), the complement of the Halting Problem is effectively the same as the halting problem itself, except with its outputs reversed. Therefore, if this existed, then the complement would also solve the Halting Problem! Because this is impossible, then therefore the Halting Problem cannot belong in coRE.

## 5 Probability Warm-Up

(a) Suppose that we have a bucket of 30 green balls and 70 orange balls. If we pick 15 balls uniformly out of the bucket, what is the probability of getting exactly $k$ green balls (assuming $0 \leq k \leq 15$) if the sampling is done **with** replacement, i.e. after we take a ball out the bucket we return the ball back to the bucket for the next round?

**Solution:** The probability of drawing a green ball is $\frac{3}{10}$ and the probability for an orange ball is $\frac{7}{10}$. Therefore, the probability of drawing $k$ green balls is

$$\left(\frac{3}{10}\right)^k \left(\frac{7}{10}\right)^{15-k}$$

(b) Same as part (a), but the sampling is **without** replacement, i.e. after we take a ball out the bucket we **do not** return the ball back to the bucket.

**Solution:** Note that each ball picked up now reduces the number of available balls. Therefore, the probaiblity is

$$\mathbb{P} = \prod_{n=0}^{k-1} \left(\frac{30-k}{100-k}\right) \left(\frac{70-(15-k)}{100-k}\right)$$

(c) If we roll a regular, 6-sided die 5 times. What is the probability that at least one value is observed more than once?

**Solution:** We can use the law of total probability to simplify our calculations. Instead of looking at the probability that at least one value is observed more than once, we find instead the probability that no value appears more than once. Let this event be called $E_1$, and the event that one value is observed more than once be called $E_2$. Of the 5 rolls, if we require every roll to be distinct, then there are $6 \cdot 5 \cdots 2 = 6!$ ways of arranging the rolls. Therefore, the probability that every roll is distinct is

$$\mathbb{P}[E_1] = \frac{6!}{6^5} = \frac{5!}{6^4}$$

Then, the probability that at least one value appears more than once is just the complement of this so therefore,

$$\mathbb{P}[E_2] = 1 - \frac{5}{6^4}$$

## 6 Past Probabilified

In this question we review some of the past CS70 topics, and look at them probabilistically.
For the following experiments,

i. Define an appropriate sample space $\Omega$.

ii. Give the probability function $\mathbb{P}[\omega]$.

iii. Compute $\mathbb{P}[E_1]$.

iv. Compute $\mathbb{P}[E_2]$.

(a) Fix a prime $q > 2$, and uniformly sample twice with replacement from $\{0, \dots, q-1\}$ (assume we have two $\{0, \dots, q-1\}$-sided fair dice and we roll them). Then multiply these two numbers with each other in $\pmod q$ space.
$E_1 =$ The resulting product is 0.
$E_2 =$ The product is $(q-1)/2$.

**Solution:**

i. The sample space is the set of pairs $(a,b)|a,b \in \{0,1,\dots,q-1\}$.

ii. Since the sample space is $\{0,1,\dots,q-1\}$, and since $(a,b)$ can be chosen independently of each other, then we know that $|\Omega| = q^2$, since there are $q$ numbers to choose from. Therefore,

$$\mathbb{P}[\omega] = \frac{1}{q^2}$$

iii. The probability of its product being zero is if one of either $a$ or $b$ is zero, then the other is any arbitrary number. Suppose $a = 0$. Then, there are $q$ numbers available for $b$. The same argument goes for $b = 0$. Therefore, there are a total of $2q$ pairings which produce a product of zero. Therefore, we have:

$$\mathbb{P}[E_1] = \frac{2q}{q^2} = \frac{2}{q}$$

iv. We now intend to solve the equation $ab \equiv \frac{q-1}{2} \pmod q$. Therefore, we require the solutions $b = \frac{q-1}{2}a^{-1} \pmod q$. Because $q$ is prime, then the multiplicative inverse of $a$ always exists, and so therefore for every value of $a$, there always exists an appropriate $b$ which will give us the product being $\frac{q-1}{2}$. So therefore there are $q-1$ pairs of numbers (we need to exclude 0) which return the product of $\frac{q-1}{2}$. Since for every $(a,b)$ there also exists a $(b,a)$ which also satisfies the relation, there are $2q$ ways in total to do this. Therefore,

$$\mathbb{P}[E_2] = \frac{2(q-1)}{q^2}$$

(b) Make a graph on $v$ vertices by sampling uniformly at random from all possible edges, (assume for each edge we flip a coin and if it is head we include the edge in the graph and otherwise we exclude that edge from the graph).
$E_1$ = The graph is complete.
$E_2$ = vertex $v_1$ has degree $d$.

**Solution:**

i. The sample space is the list of all possible edges in a graph, as well as the permutation of all labeled vertices.

ii. The size of the sample space is the number of edges in our graph of $v$ vertices. Furthermore, there are $v!$ ways of enumerating the vertices. Therefore we have $|\Omega| = \binom{v}{2}v! = \frac{v(v-1)}{2}v!$, so therefore:

$$\mathbb{P}[\omega] = \frac{1}{|\Omega|} = \frac{2}{v(v-1)v!}$$

iii. There is only one arrangement of vertices which will give a complete graph. Then, there are still $v!$ ways of labelling the vertices, so therefore

$$\mathbb{P}[E_1] = \frac{2v!}{v(v-1)v!} = \frac{2}{v(v-1)}$$

iv. Consider vertex $v_1$. If we require it to have degree $d$, then we require $\binom{v-1}{d}$ of its vertices to be sampled. Then, of the remaining graph of $v-1$ vertices, there are $\frac{(v-1)(v-2)}{2}$ ways of selecting vertices, so therefore:

$$\mathbb{P}[E_2] = \frac{2}{v(v-1)v!} \cdot \binom{v-1}{d} \frac{(v-1)(v-2)}{2} = \binom{v-1}{d} \cdot \frac{(v-2)}{v \cdot v!}$$

(c) Create a random stable matching instance by having each person's preference list be a random permutation of the opposite entity's list (make the preference list for each individual job and each individual candidate a random permutation of the opposite entity's list). Finally, create a uniformly random pairing by matching jobs and candidates up uniformly at random (note that in this pairing, (1) a candidate cannot be matched with two different jobs, and a job cannot be matched with two different candidates (2) the pairing does not have to be stable).
$E_1$ = All jobs have distinct favorite candidates.
$E_2$ = The resulting pairing is the candidate-optimal stable pairing.

**Solution:**

(a) The sample space is the total number of ways of listing job preferences, candidate preferences, and an arbitrary matching.

(b) For the job preferences, there are $n$ candidates, and each candidate can have $n!$ ways of arranging a preference list. The same argument follows from the candidates, since they are independent of each other. Further, there are $n!$ ways of matching a job to a candidate, so therefore we have

$$|\Omega| = (n!)^n \cdot (n!)^n \cdot n! = (n!)^{2n+1}$$

and so

$$\mathbb{P}[\omega] = \frac{1}{(n!)^{2n+1}}$$

(c) For all jobs to have distinct favourite candidates, there are $n!$ ways of arranging the favourite candidates of every job. Furthermore, once the favourite candidates are chosen, there are another $(n-1)!$ ways of arranging the other $n-1$ candidates. Therefore, for the job choices, there are $n!(n-1)!^n$ ways of choosing the arrangement of jobs.

Then, the candidate preferences list remain uncahnged, as well as the matchings. Therefore,

$$\mathbb{P}[E_1] = \frac{n!((n-1)!)^n(n!)^{n+1}}{(n!)^{2n+1}} = \frac{((n-1)!)^n}{(n!)^{2n+1}}$$

(d) There is always only one candidate-optimal stable pairing for any set of job and candidtae preferences. The other quantities remain the same, so therefore

$$\mathbb{P}[E_2] = \frac{(n!)^{2n}}{(n!)^{2n+1}} = \frac{1}{n!}$$