CS 70 Discrete Mathematics and Probability Theory
Fall 2023 Rao, Tal

DIS 7A

1 Hilbert's Hotel

Note 11

You don't have any summer plans, so you decide to spend a few months working for a magical hotel with a countably infinite number of rooms. The rooms are numbered according to the natural numbers, and all the rooms are currently occupied. Assume that guests don't mind being moved from their current room to a new one, so long as they can get to the new room in a finite amount of time (i.e. guests can't be moved into a room infinitely far from their current one).

(a) A new guest arrives at the hotel. All the current rooms are full, but your manager has told you never to turn away a guest. How could you accommodate the new guest by shuffling other guests around? What if you instead had k guest arrive, for some fixed, positive $k \in \mathbb{Z}$?

(b) Unfortunately, just after you've figured out how to accommodate your first $k+1$ guests, a countably infinite number of guests arrives in town on an infinitely long train. The guests on the train are sitting in seats numbered according to the natural numbers. How could you accommodate all the new guests?

(c) Thanks to a (literally) endless stream of positive TripAdvisor reviews, word of the infinite hotel gets around quickly. Soon enough you find out that a countably infinite number of trains have arrived in town. Each is of infinite length, and carries a countably infinite number of passengers. How would you accommodate all the new passengers?

Solution:

(a) Shift all guests into the room number that is k greater than their current room number. So for a guest in room i move him/her to room $i+k$. Then place the k new guests in the k first rooms in the hotel which will now be unoccupied.

(b) Place all existing guests in room $2i$ where i is their current room number. Place all the new guests in room $2j+1$ where j is their seat number on the train.

(c) **Solution 1**: We first set up a bijection between the newly arriving guests and the set $\mathbb{N} \times \mathbb{N}$. Notice that each guest has an "address": his/her train number i and his/her seat number j. Let this guest be mapped to (i, j). It is clear that this is a bijection.

We know from Lecture Note 10 that the set $\mathbb{N} \times \mathbb{N}$ is countable (via the spiral method) and hence there is a bijection from $\mathbb{N}^2$ to $\mathbb{N}$. Thus the newly arriving guests can be enumerated and considered as if arriving in a single infinite length train with their corresponding seat

numbers given by the enumeration. This reduces to the same exact problem as the previous part! Therefore, we can accommodate these guests.

Solution 2: Place all existing guests in room 2^i where i is their current room number. Assign the $(k+2)$th prime, p_{k+2}, to the kth train (e.g. the 0th train will be assigned the 2nd prime, 3). We then place each new guest in room p_{k+2}^{j+1}, where j is the seat number of the new guest on that train.

This works because any power of a prime p will not have any prime factors other than p.

Yes, there will be plenty of empty rooms, but that's okay because every guest will still have somewhere to stay.

2 Counting Functions

Note 11

Are the following sets countable or uncountable? Prove your claims.

(a) The set of all functions f from $\mathbb{N}$ to $\mathbb{N}$ such that f is non-decreasing. That is, $f(x) \leq f(y)$ whenever $x \leq y$.

(b) The set of all functions f from $\mathbb{N}$ to $\mathbb{N}$ such that f is non-increasing. That is, $f(x) \geq f(y)$ whenever $x \leq y$.

Solution:

(a) Uncountable: Let us assume the contrary and proceed with a diagonalization argument. If there are countably many such function we can enumerate them as

	0	1	2	3	...
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	...
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	...
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	...
f_3	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Now go along the diagonal and define f such that $f(x) > f_x(x)$ and $f(y) > f(x)$ if $y > x$, which is possible because at step k we only need to find a number $\in \mathbb{N}$ greater than all the $f_j(j)$ for $j \in \{0, \ldots, k\}$; for example, we could define such a function using

$$f(x) = \begin{cases} f_0(0) + 1 & x = 0 \\ \max(f_x(x), f(x-1)) + 1 & x > 0 \end{cases}$$

This function differs from each f_i and therefore cannot be on the list, hence the list does not exhaust all non-decreasing functions. As a result, there must be uncountably many such functions.

Alternative Solution: Look at the subset $\mathscr{S}$ of strictly increasing functions. Any such f is uniquely identified by its image which is an infinite subset of $\mathbb{N}$. But the set of infinite subsets of $\mathbb{N}$ is uncountable. This is because the set of all subsets of $\mathbb{N}$ is uncountable, and the set of all finite subsets of $\mathbb{N}$ is countable. So $\mathscr{S}$ is uncountable and hence the set of all non-decreasing functions must be too.

Alternative Solution 2: We can inject the set of infinitely long binary strings into the set of non-decreasing functions as follows. For any infinitely long binary string b, let $f(n)$ be equal to the number of 1's appearing in the first n-digits of b. It is clear that the function f so defined is non-decreasing. Also, since the function f is uniquely defined by the infinitely long binary string, the mapping from binary strings to non-decreasing functions is injective. Since the set of infinite binary strings is uncountable, and we produced an injection from that set to the set of non-decreasing functions, that set must be uncountable as well.

(b) Countable: Let D_n be the subset of non-increasing functions for which $f(0) = n$. Any such function must stop decreasing at some point (because $\mathbb{N}$ has a smallest number), so there can only be finitely many (at most n) points $X_f = \{x_1, \ldots, x_k\}$ at which f decreases. Let y_i be the amount by which f decreases at x_i, then f is fully described by $\{(x_1, y_1), \ldots, (x_k, y_k), (-1,0), \ldots, (-1,0)\} \in \mathbb{N}^n = \mathbb{N} \times \mathbb{N} \times \cdots \times \mathbb{N}$ (n times), where we padded the k values associated with f with $n-k$ $(-1,0)$s. In Lecture note 11, we have seen that $\mathbb{N} \times \mathbb{N}$ is countable by the spiral method. Using it repeatedly, we get $\mathbb{N}^{(2^l)}$ is countable for all $l \in \mathbb{N}$. This gives us that $\mathbb{N}^n$ is countably for any finite n (because $\mathbb{N}^n \subset \mathbb{N}^{(2^l)}$ where l is such that $2^l \geq n$). Hence D_n is countable. Since each set D_n is countable we can enumerate it. Map an element of D_n to (n, j) where j is the label of that element produced by the enumeration of D_n. This produces an injective map from $\cup_{n \in \mathbb{N}} D_n$ to $\mathbb{N} \times \mathbb{N}$ and we know that $\mathbb{N} \times \mathbb{N}$ is countable from Lecture note 11 (via spiral method). Now the set of all non-increasing functions is $\cup_{i \in \mathbb{N}} D_n$, and thus countable.

3 Hello World!

Note 12

Determine the computability of the following tasks. If it's not computable, write a reduction or self-reference proof. If it is, write the program.

(a) You want to determine whether a program P on input x prints "Hello World!". Is there a computer program that can perform this task? Justify your answer.

(b) You want to determine whether a program P prints "Hello World!" before running the kth line in the program. Is there a computer program that can perform this task? Justify your answer.

(c) You want to determine whether a program P prints "Hello World!" in the first k steps of its execution. Is there a computer program that can perform this task? Justify your answer.

Solution:

(a) Uncomputable. We will reduce `TestHalt` to `PrintsHW`(P,x).

```
TestHalt(P, x):
  P'(x):
    run P(x) while suppressing print statements
    print("Hello World!")
  if PrintsHW(P', x):
    return true
  else:
    return false
```

If `PrintsHW` exists, `TestHalt` must also exist by this reduction. Since `TestHalt` cannot exist, `PrintsHW` cannot exist.

(b) Uncomputable. Reduce `PrintsHW`(P,x) from part (a) to this program `PrintsHWByK`(P,x,k).

```
PrintsHW(P, x):
  for i in range(len(P)):
    if PrintsHWByK(P, x, i):
      return true
  return false
```

(c) Computable. You can simply run the program until k steps are executed. If P has printed "Hello World!" by then, return true. Else, return false.

The reason that part (b) is uncomputable while part (c) is computable is that it's not possible to determine if we ever execute a specific line because this depends on the logic of the program, but the number of computer instructions can be counted.

4 Code Reachability

Note 12

Consider triplets (M,x,L) where

- `M` is a Java program
- `x` is some input
- `L` is an integer

and the question of: if we execute $M(x)$, do we ever hit line L?

Prove this problem is undecidable.

Solution: Suppose we had a procedure that could decide the above; call it `Reachable(M, x, L)`. Consider the following example of a program deciding whether $P(x)$ halts:

```
def Halt(P, x):
    def M(t):
        run P(x)   # line 1 of M
        return     # line 2 of M
    return Reachable(M, 0, 2)
```

Program M reaches line 2 if and only if $P(x)$ halted. Thus, we have implemented a solution to the halting problem — contradiction.