CS 70 Discrete Mathematics and Probability Theory
Fall 2020 DIS 8A

1 Graph Isomorphic

In graph theory, an isomorphism of graphs G and H is a bijection between the vertex sets of G and H

$$f : V(G) \to V(H)$$

such that any two vertices u, v of G are adjacent in G if and only if $f(u), f(v)$ are adjacent in H. Prove the following:

1. The degrees of corresponding nodes $u, f(u)$ are the same.
2. There is a bijection between edges.
3. If G is connected, then H is also connected.

Solution:

(a) by definition, (u,v) in G **if and only if** $(f(u), f(v))$ in H. So they have the same degree.

(b) There is a bijection: (u,v) maps to $(f(u), f(v))$.

(c) If G is connected and H is not connected. Then there exists a pair of points u, v are connected in G by some path, but $f(u), f(v)$ are not connected in H by any path. Let the path be $u->x_1->x_2,\dots,->x_k->v$ where $k \geq 0$ ($k = 0$ means there is an edge between u, v). Since (u, x_1) is connected by some edge in G, then $(f(u), f(x_1))$ is connected by some edge in H. Similar argument holds for any adjacent pairs $(x_i, x_{i+1}), (f(x_i), f(x_{i+1}))$ or $(x_k, v), (f(x_k), f(v))$. So there is a path between $f(u)$ and $f(v)$ in H. Contradiction!

2 Countability Practice

(a) Do $(0,1)$ and $\mathbb{R}_+ = (0,\infty)$ have the same cardinality? If so, either give an explicit bijection (and prove that it is a bijection) or provide an injection from $(0,1)$ to $(0,\infty)$ and an injection from $(0,\infty)$ to $(0,1)$ (so that by Cantor-Bernstein theorem the two sets will have the same cardinality). If not, then prove that they have different cardinalities.

(b) Is the set of strings over the English alphabet countable? (Note that the strings may be arbitrarily long, but each string has finite length. Also the strings need not be real English words.) 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.
Explicit bijection: Consider the bijection $f : (0,1) \to (0,\infty)$ given by

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

We show that f is a bijection by proving separately that it is one-to-one and onto. The function 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.

The function f is onto: take any $y \in (0,\infty)$. Let $x = 1/(1+y)$. Note that $x \in (0,1)$. 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 $(0,\infty)$, so they have the same cardinality. (In fact, they are both uncountable.)

Indirect bijection: The injection from $(0,1)$ to $(0,\infty)$ is trivial; consider the function $f : (0,1) \to (0,\infty)$ given by

$$f(x) = x.$$

It is easy to see that f is injective.

For the other way, consider the function $g : (0,\infty) \to (0,1)$ given by

$$g(x) = \frac{1}{x+1}.$$

To see that g is injective, suppose $g(x) = g(y)$. Then

$$\frac{1}{x+1} = \frac{1}{y+1} \implies x = y.$$

Hence g is injective. Thus we have an injective function from $(0,1)$ to $(0,\infty)$ and an injective function from $(0,\infty)$ to $(0,1)$. By Cantor-Bernstein theorem there exists a bijection from $(0,1)$ to $(0,\infty)$ and hence they have the same cardianlity.

(b) Countable. 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.

(c) No, the strings are still countable. Let $\mathscr{A} = \{a_1, a_2, \dots\}$ 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

$$a, aa, aaa, \dots$$

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.

3 Python Functions

(a) The set $F = \{f : \mathbb{N} \to \{0,1\}\}$ is not countable.

(b) Prove that the set of all python functions that output $\{0,1\}$ is countable. (Python functions have the same power as Turing machines, but people are more familiar with python.)

(c) The set of Python functions that take in input x and output either 0 or 1 appears to be the same as F in (a), but the set of Python function is countable. Why?

Solution:

(a) If F is countable, then there is a list of functions f_i that $F = \{f_i, i \in \mathbb{N}\}$, however $f'(x) = 1 - f_x(x)$ is not in F, contradiction!

(b) A python function can be encoded into a binary string. There is a one-on-one mapping between binary string and integers in the following way:

- A finite binary string can be mapped to a natural number by taking it's binary form.
- That is, a binary string S of length n can be converted into integer $\sum_{i=0}^{n-1} S_i 2^i$.

So there is a bijection between set of finite length binary strings and the natural numbers.

(c) This is a bit subtle. In part (a), we used the fact that a function $f : \mathbb{N} \to 0, 1$ assigns a value to each element of $\mathbb{N}$. While a python program may do the same thing, that is, each python program may assign a value to every elemengt of $\mathbb{N}$, python programs cannot produce all functions. Indeed, the number of functions that python programs can produce cannot be more than the number of python programs. This is countable by previous part. (Intuitively, a python program cannot contain more "information" than its length, whereas a function f contains some information for every natural number which means it contains infinite information. This is basically the same argument.)