CS 70 Discrete Mathematics and Probability Theory
Summer 2023 Huang, Suzani, and Tausik

HW 2

1 Countability Practice

Note 11

(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.

2 Counting Functions

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 10, 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 10 (via spiral method). Now the set of all non-increasing functions is $\cup_{i \in \mathbb{N}} D_n$, and thus countable.

3 Countability and the Halting Problem

Note 11
Note 12

Using methods from countability, we will prove the Halting Problem is undecidable.

a) What is a reasonable representation for a computer program? Using this definition, show that the set of all programs are countable. *(Hint: Machine Code)*

b) The Halting Problem only considers programs which take a finite length input. Show that the set of all finite-length inputs is countable.

c) Assume that you have a program that tells you whether or not a given program halts on a specific input. Since the set of all programs and the set of all inputs are countable, we can enumerate them and construct the following table.

	x_1	x_2	x_3	x_4	...
p_1	H	L	H	L	...
p_2	L	L	L	H	...
p_3	H	L	H	L	...
p_4	L	H	L	L	...
⋮	⋮	⋮	⋮	⋮	⋱

An H (resp. L) in the ith row and jth column means that program p_i halts (resp. loops) on input x_j. Now write a program that is not within the set of programs in the table above.

d) Find a contradiction in part a and part c to show that the halting problem can't be solved.

Solution:

a) As in discussion and lecture, we represent a computer programs with a set of finite-length strings (which, in turn, can be represented by a set of finite length binary strings). The set of finite length binary strings are countably infinite. Therefore the set of all programs is countable.

b) Notice that all inputs can also be represented by a set of finite length binary strings. The set of finite length binary strings are countably infinite, as proved in Note 11. Therefore the set of all inputs is countable.

c) For the sake of deriving a contradiction in part (d), we will use the following program:

```
procedure P'(x_j)
    if P_j(x_j) halts then
        loop
    else
        halt
    end if
end procedure
```

d) If the program you wrote in part c) exists, it must occur somewhere in our complete list of programs, P_n. This cannot be. Say that P_n has source code x_j (i.e. its source code corresponds to column j). What is the (i, j)th entry of the table? If it's H, then $P_n(x_j)$ should loop forever, by construction; if it's L, then $P_n(x_j)$ should halt. In either case, we have a contradiction.

4 Tenfold

Note 12

(a) Suppose we have a program TenTimes which takes in two programs, F and G, some input y, as well as an integer z.

TenTimes(F, G, y, z) will run F and G - both with input y - at the same time, and start a timer as soon as they start running. If after exactly z seconds, F is at line ℓ and G is at line 10ℓ, TenTimes returns True. Otherwise, the program returns False. (If F or G halts before x seconds, then it stays on the line in which it halts.) Show that TenTimes is decidable.

(b) We now consider a program TenTimes2, which takes just the two programs, F and G, and some input y. If F and G are both run at the same time with input y, and at some point F is at line ℓ while G is simultaneously at line 10ℓ, TenTimes2 will return True. If this never occurs, TenTimes2 will return False. (Again, if F or G halts before x seconds, then it stays on the line in which it halts.) Show that TenTimes2 is undecidable.

Solution:

(a) One can simply run TenTimes. Since there is a time limit (the finite variable z), we can effectively wait until that limit has been reached to see what happens. Regardless of whether or not F or G halted, they will both be on some line once z seconds have passed, so a decision can be made on what TenTimes should output. Therefore, this program is decidable.

(b) TenTimes2 is undecidable, for otherwise we could solve the halting problem as so: define P' as P, but shift all lines starting at 10 up by one. Also, if there are any instances of line k in P, and $k \geq 10$, then replace that value with $k+1$. This way, P' will have no code on line 10.

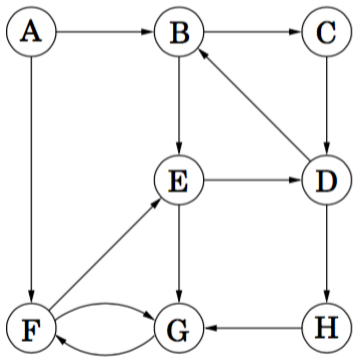
```
TestHalt(P, x):
  F(x):
    Loop on Line 1 (this line) indefinitely
  G(x):
    P'(x)
    Go to Line 10 of P'(x) and halt
  if TenTimes2(F, G, x):
    return true
  else:
    return false
```

If TenTimes2 exists, TestHalt must also exist by this reduction. However, since we know TestHalt cannot exist, TenTimes2 therefore cannot exist. In other words, it is undecidable.

5 Graph Basics

Note 5

In the first few parts, you will be answering questions on the following graph G.



(a) What are the vertex and edge sets V and E for graph G?

(b) Which vertex has the highest in-degree? Which vertex has the lowest in-degree? Which vertices have the same in-degree and out-degree?

(c) What are the paths from vertex B to F, assuming no vertex is visited twice? Which one is the shortest path?

(d) Which of the following are cycles in G?

i. $(B,C),(C,D),(D,B)$
ii. $(F,G),(G,F)$
iii. $(A,B),(B,C),(C,D),(D,B)$
iv. $(B,C),(C,D),(D,H),(H,G),(G,F),(F,E),(E,D),(D,B)$

(e) Which of the following are walks in G?

i. (E,G)
ii. $(E,G),(G,F)$
iii. $(F,G),(G,F)$
iv. $(A,B),(B,C),(C,D),(H,G)$
v. $(E,G),(G,F),(F,G),(G,C)$
vi. $(E,D),(D,B),(B,E),(E,D),(D,H),(H,G),(G,F)$

(f) Which of the following are tours in G?

i. (E,G)
ii. $(E,G),(G,F)$
iii. $(F,G),(G,F)$
iv. $(E,D),(D,B),(B,E),(E,D),(D,H),(H,G),(G,F)$
v. $(B,C),(C,D),(D,H),(H,G),(G,F),(F,E),(E,D),(D,B)$

In the following three parts, let's consider a general undirected graph G with n vertices ($n \geq 3$). If true, provide a short proof. If false, show a counterexample.

(g) True/False: If each vertex of G has degree at most 1, then G does not have a cycle.

(h) True/False: If each vertex of G has degree at least 2, then G has a cycle.

(i) True/False: If each vertex of G has degree at most 2, then G is not connected.

Solution:

(a) A graph is specified as an ordered pair $G = (V, E)$, where V is the vertex set and E is the edge set.

$$V = \{A, B, C, D, E, F, G, H\},$$
$$E = \{(A,B), (A,F), (B,C), (B,E), (C,D), (D,B), (D,H), (E,D), (E,G), (F,E), (F,G), (G,F), (H,G)\}.$$

(b) G has the highest in-degree (3). A has the lowest in-degree (0).

$\{B, C, D, E, F, H\}$ all have the same in-degree and out-degree. H and C have in-degree (out-degree) equal to 1 and the other four have in-degree (out-degree) equal to 2.

(c) There are three paths:
$(B,C), (C,D), (D,H), (H,G), (G,F)$
$(B,E), (E,D), (D,H), (H,G), (G,F)$
$(B,E), (E,G), (G,F)$
The first two have length 5, while the last one has length 3, so the last one is the shortest path.

(d) A cycle is a path that starts and ends at the same point. This means that (iii) is not a cycle, since it starts at A but ends at B. In addition, all the vertices $\{v_1, \ldots, v_n\}$ in the cycle $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$ should be distinct, so (iv) is not a cycle. The correct answers are (i) and (ii).

(e) A walk consists of any sequence of edges such that the endpoint of each edge is the same as the starting vertex of the next edge in the sequence. Example (iv) does not fit this definition–even though it uses only valid edges, the endpoint of the second to last edge in D, while the start point of the next edge is H. Example (v) also is not a walk, since it tries to walk from G to C as its last step, but there is no such edge. All the rest are walks.

(f) A tour is a walk that has the same start and end vertex; examples (iii) and (v) satisfy this definition. Note in part (d), we already said that (iii) was a cycle–and indeed, all cycles are also tours.

(g) True. In order for there to be a cycle in G starting and ending at some vertex v, we would need at least two edges incident to v: one to leave v at the start of the cycle, and one to return to v at the end. If every vertex has degree at most 1, no vertex has two or more edges incident on it, so no vertex is capable of acting as the start and end point of a cycle.

(h) True. Consider starting a walk at some vertex v_0, and at each step, walking along a previously untraversed edge, stopping when we first visit some vertex w for the second time. If this process terminates, the part of our walk from the first time we visited w until the second time is a cycle. Thus, it remains only to argue this process always terminates.

Each time we take a step from some vertex v, since we are not stopping, we must have visited that vertex exactly once and not yet left. It follows that we have used at most one edge incident with v (either we started at v, or we took an edge into v). Since v has degree at least 2, there must be another edge leaving v for us to take.

(i) False. For example, a 3-cycle (triangle) is connected and every vertex has degree 2.

6 Coloring Countries

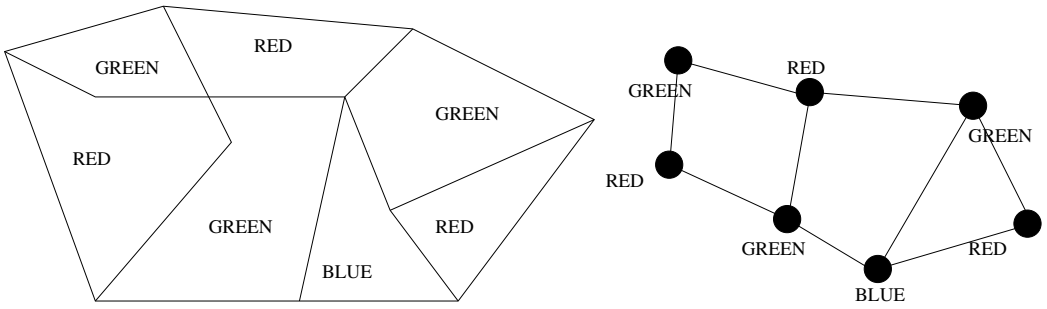


Figure 1: An island map that has been colored using three colors, and the corresponding graph.

Explorers have just discovered several new islands in the Pacific ocean! Each island is divided into several countries. As chief map-maker, your job is to make a map of each island, giving a color to each country so that *no two neighboring countries have the same color*. For example, the left side of Figure 1 shows a map that has been colored using three colors.

Unfortunately, you haven't been to the mapmaking store in a while, and so you only have six colors to work with: red, green, blue, purple, orange and almond toast. Fortunately, that's enough to color any map, as we shall see!

The right side of Figure 1 shows another way of looking at a map: make a vertex for each country, and draw an edge between two nodes if the countries are neighbors. The graph you get will be *planar*, meaning it can be laid out so that none of the edges cross each other.

(a) In order to color the map in Figure 1 so that no neighbors have the same color, you need at least three different colors. (To see why, try to color it using only red and green and see what happens.) Draw a map that needs at least *four* different colors, and then draw the corresponding planar graph.

(b) In order to see that six colors will be enough to color any map, prove the following theorem:

Theorem: *Every planar graph can be colored with six colors, in such a way that no two neighboring vertices have the same color.*

You may find the following lemma useful:

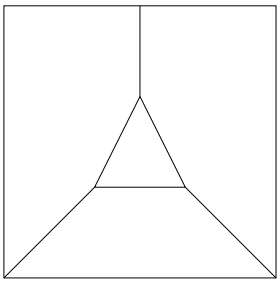
Lemma: *Every planar graph (with at least one node) has a node with at most five neighbors. (We say the node has* degree *at most five.)*

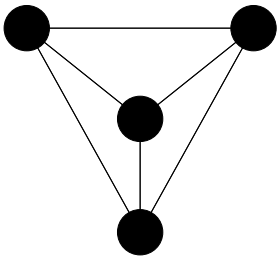
You don't need to prove the lemma, but you may assume it is true when proving the theorem.

(In fact, it only ever takes four colors to color a planar graph – but that's much harder to prove. Search for *four color theorem*.)

Solution:

(a)





(b) **Theorem:** *Every planar graph can be colored with six colors, in such a way that no two neighboring vertices have the same color.*

Proof. We proceed by induction on the number of nodes in the graph.

Base case: Clearly graphs with zero or one vertices can be colored.

Inductive hypothesis: Assume that every graph with n vertices can be colored with at most six colors.

Inductive step: Suppose we are given a graph with $n+1$ vertices. According to the lemma, there is a vertex with at most five neighbors. Remove that vertex from the graph, and apply the inductive hypothesis to color the remaining n-vertex graph. Then, put back the vertex you removed, and color it with a different color from its five neighbors. □

7 Binary Trees

Note 5

You may have seen the recursive definition of binary trees from previous classes. Here, we define binary trees in graph theoretic terms as follows (**Note:** here we will modify the definition of leaves slightly for consistency).

- A binary tree of height > 0 is a tree where exactly one vertex, called the **root**, has degree 2, and all other vertices have degrees 1 or 3. Each vertex of degree 1 is called a **leaf**. The **height** h is defined as the maximum length of the path between the root and any leaf.

- A binary tree of height 0 is the graph with a single vertex. The vertex is both a leaf and a root.

(a) Let T be a binary tree of height > 0, and let $h(T)$ denote it's height. Let r be the root in T and u and v be it's neighbors. Show that removing r from T will result in two binary trees, L, R with roots u and v respectively. Also, show that $h(T) = \max(h(L), h(R)) + 1$.

(b) Using the graph theoretic definition of binary trees, prove that the number of vertices in a binary tree of height h is at most $2^{h+1} - 1$.

(c) Prove that all binary trees with n leaves have $2n - 1$ vertices.

Solution:

(a) Since r has degree 2, removing it will break T into two connected components; suppose we call them L and R. By symmetry, we just need to prove that L is a binary tree.

Without loss of generality, suppose $u \in L$. Before removing r, u must have had degree 1 or 3. If u had degree 1, then after removing r, u is a single vertex, and would be a binary tree of height 0, and also is a root. If u had degree 3, then after removing r, u has degree 2, and all other vertices in L have degree 1 or 3. Thus, L is a binary tree with root u; the same reasoning shows that R would also be a binary tree with root v.

To prove that $h(T) = \max(h(L), h(R)) + 1$, we note that because T is a tree, any path from r to a leaf must go through either u or v but not both. This means that the maximum distance from r to any leaf is one plus either the maximum distance from u to any leaf in L (as the path cannot go back through r) or the maximum distance from v to any leaf in R. Formally, if we define $\mathscr{L}(L)$ and $\mathscr{L}(R)$ to be the set of leaves in L and R respectively, and $d(r, l)$ as the length of the path from r to some leaf l, then we have

$$\begin{aligned} h(T) &= 1 + \max\left(\max_{l \in \mathscr{L}(L)} (d(u,l)), \max_{l \in \mathscr{L}(R)} (d(u,l)) \right) \\ &= 1 + \max(h(L), h(R)) \end{aligned}$$

(b) We proceed by induction on the height of the binary tree.

Base Case: a binary tree of height 0 is a singleton and so has $2^1 - 1 = 1$ vertex.

Inductive Hypothesis: assume for all $k < h$, a binary tree of height k has at most $2^{k+1} - 1$ vertices.

Inductive Step: By part (a), we can remove the root from a binary tree and obtain two binary trees L and R of height k and l respectively. Since $h(T) = \max(h(L), h(R)) + 1$, we know that $k, l \leq h - 1$ so we can apply the inductive hypothesis to L and R. Thus, we have that the number of vertices in T is at most

$$\underbrace{(2^{k+1} - 1)}_{\text{from } L} + \underbrace{(2^{l+1} - 1)}_{\text{from } R} + \underbrace{1}_{\text{root}} \leq 2^{(h-1)+1} + 2^{(h-1)+1} - 1 = 2^{h+1} - 1.$$

(c) We proceed by induction on the number of leaves in the binary tree.

Base Case: if a binary tree has one leaf, it is a singleton and so has $1 = 2 \cdot 1 - 1$ vertices.

Inductive Hypothesis: assume for all $k < n$, a binary tree with k leaves has $2k - 1$ vertices.

Inductive Step: For a binary tree T with $n > 1$ leaves, remove the root, r and break T into binary trees L and R. Suppose L has a leaves and R has b leaves. Note that all the leaves of T are in L or R, as $n > 1$ implies the root is not a leaf, which means $a + b = n$. By the inductive hypothesis, L has $2a - 1$ vertices, and R has $2b - 1$ vertices, and so the number of vertices in T is

$$\underbrace{(2a-1)}_{\text{from } L} + \underbrace{(2b-1)}_{\text{from } R} + \underbrace{1}_{\text{root}} = 2(a+b) - 1 = 2n - 1.$$

8 Does Euler Still Work?

Note 5

In the country of Eulerville, we have k states, where each state can be modeled by a connected planar graph. Furthermore, all these states are connected through a series of non-intersecting highways that go from one state to another (in other words, the whole country can be thought of as a planar graph of planar graphs.)

One day, the leader of Eulerville decides to shutdown all highways that connect the states together, which results in the k states being disconnected from each other. Since Euler's formula, $v - e + f = 2$, requires the graph to be connected, how can we modify this formula to be correct for the now disconnected country of Eulerville? Be sure to prove your answer using induction. (Hint: we can still use Euler's formula for each of the k states, because each individual state is still planar and connected. Assume that after the highways are shutdown, we are left with V vertices, E edges, and F faces in the full planar graph)

Solution: We can our notation as follows: for a state i, the number of vertices is $|v_i|$, the number of edges is $|e_i|$, and the number of faces is $|f_i|$. For each of the k states, we know that Euler's formula still holds. Therefore, we can say $|v_i| - |e_i| + |f_i| = 2$ for every state i.
Now, suppose we add all of these formulas together. This results in

$$\sum_{i=1}^{k} |v_i| - \sum_{i=1}^{k} |e_i| + \sum_{i=1}^{k} |f_i| = 2k$$

Notice that the sum of all vertices and edges are just $|V|$ and $|E|$, respectively. Also, the "infinite" face is counted once by each state, so it gets over counted $k - 1$ times. Therefore, we can rewrite our equation as

$$|V| - |E| + |F| + (k-1) = 2k$$
$$|V| - |E| + |F| = k + 1$$

We now prove by induction that this formula is correct. For our base case, $k = 1$, we see that this is just Euler's formula, so our base case is correct. Now, assume that the formula is true for $k = m$

states. We would like to show that it is true for $k = m+1$ states, as well.
Suppose our country, C is a planar graph of $m+1$ connected states (but not connected to each other), and that C has $V(C)$ vertices, $E(C)$ edges, and $F(C)$ faces. We divide C into C_1, which contains one state, and C_m, which contains m states (and we can use the same notation of $V(C_1)$ being the number of vertices in C_1, $E(C_m)$ being the number of vertices in C_m, and so on). Notice that $V(C) = V(C_1) + V(C_m)$ and $E(C) = E(C_1) + E(C_m)$, but $F(C_1)$ and $F(C_m)$ do share exactly one face, so in order to not over count, we say $F(C) = F(C_1) + F(C-m)$. With these facts, we can now use algebra to say:

$$\begin{aligned} & V(C) - E(C) + F(C) \\ &= V(C_1) + V(C_m) - (E(C_1) + E(C_m)) + F(C_1) + F(C_m) - 1 \\ &= V(C_1) - E(C_1) + F(C_1) + V(C_m) - E(C_m) + F(C_m) - 1 \\ &= 2 + m + 1 - 1 = (m+1) + 1 \end{aligned}$$

This concludes the proof.