CS 70 Discrete Mathematics and Probability Theory
Spring 2016 Rao and Walrand HW 6

Due Thursday March 3rd at 10PM

Warmup

1. **(2/2/2/2/2) Reviewing Lagrange**

   (a) Prove the following: If p is a prime and $y_1,\dots,y_n \in \mathbb{N}$ are all different from 0 modulo p, then $y_1 \times \cdots \times y_n$ is also different from 0 modulo p.

   Answer: Since $y_1,\dots,y_n \in \mathbb{N}$ are all different from 0 modulo p, p does not factor any of them. Thus the prime factorization of $y_1 \times \cdots \times y_n$ does not include p, so p does not divide $y_1 \times \cdots \times y_n$. So

   $$y_1 \times \cdots \times y_n \not\equiv 0 \pmod{p}.$$

   (b) Prove the following: Given a prime p and two integers a,b, it is always possible to find a polynomial $f(x)$ of degree at most 1 such that $f(0) \equiv a \pmod{p}$ and $f(1) \equiv b \pmod{p}$.

   Answer: Let $f(x) = a + (b-a)x$. Then $f(0) = a \equiv a \pmod{p}$ and $f(1) = a + (b-a) = b \equiv b \pmod{p}$.

   (c) You are given a prime p and a positive number $n < p$. Show how to find a polynomial $f(x)$ of degree at most n satisfying $f(0) \equiv f(1) \equiv \cdots \equiv f(n-1) \equiv 0 \pmod{p}$ and $f(n) \equiv 1 \pmod{p}$. In other words, the polynomial f should be congruent to zero at the points $x = 0,\dots,n-1$; at $x = n$ the polynomial should be 1 mod p.
   Hint: Consider $F(x) = (x-0)(x-1)(x-2)\cdots(x-(n-1))$; what can you say about it?

   Answer: Let $F(x) = (x-0)(x-1)(x-2)\cdots(x-(n-1))$, and define $a = F(n) \bmod p$. Note that $a \equiv n! \pmod{p}$ is invertible modulo p, since $n < p$ (we have $a^{-1} \equiv n^{-1} \times (n-1)^{-1} \times \cdots \times 1^{-1} \pmod{p}$, and each of $1,\dots,n$ are invertible modulo p since they are less than p and thus relatively prime to p). Let $b = a^{-1} \bmod p$. Therefore, we may take

   $$f(x) = bF(x).$$

   We will have $f(0) \equiv f(1) \equiv \cdots \equiv f(n-1) \equiv 0 \pmod{p}$, since $F(0) = \cdots = F(n-1) = 0$. Also, we will have $f(n) \equiv F(n)^{-1}F(n) \equiv 1 \pmod{p}$. Finally, F has degree n since it has n terms in its definition, and so this choice of f has degree at most n. Consequently, this choice of f satisfies all the requirements.

   (d) You are given p and n as before, but now you are also given an index j with $0 \le j \le n$. Show how to find a polynomial $g_j(x)$ of degree at most n satisfying

   $$g_j(i) \equiv \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \pmod{p} \qquad \text{for each } i = 0,1,\dots,n.$$

In other words, the polynomial g_j should be congruent to zero at the points $x = 0, \dots, n$, except that at $x = j$ it should be congruent to 1 mod p.

Answer: Use the same idea as in part (c). Define $G_j(x) = (x-0)\cdots(x-(j-1))(x-(j+1))\cdots(x-n)$ and $g_j(x) = (G_j(j)^{-1} \bmod p)G_j(x)$. As before, $g_j(i) = 0$ if $i \neq 0$, $g_j(j) \equiv 1 \pmod{p}$, and g_j has degree at most n, so this satisfies all the requirements.

(e) You are given a prime p, a number n with $0 < n < p$, and a sequence of values $a_0, a_1, \dots, a_n \pmod{p}$. Describe an efficient algorithm to find a polynomial $h(x)$ of degree at most n satisfying $h(0) \equiv a_0 \pmod{p}$, $h(1) \equiv a_1 \pmod{p}$, $\dots$, $h(n) \equiv a_n \pmod{p}$.

Hint: What can you say about the polynomial $3g_0(x) + 7g_1(x)$, where $g_0(x), g_1(x)$ are as defined in part (d)? Does this give you any ideas?

Answer: Let

$$h(x) = a_0 g_0(x) + a_1 g_1(x) + \cdots + a_n g_n(x),$$

with the g_j's defined as above. Then h has degree at most n, since the g_j's do, and moreover $h(i) \equiv a_0 g_0(i) + \cdots + a_n g_n(i) \equiv 0 + \cdots + 0 + a_i \cdot 1 + 0 + \cdots + 0 \equiv a_i \pmod{p}$, as desired.

Note that this h may be computed efficiently. Multiplying a polynomial of degree d by $(x-i)$ modulo p requires d multiplications and d additions modulo p, so we can compute each $G_j(x)$ in $O(n^2(\lg p)^2)$ time. Inverting $G_j(j)$ modulo p can be done in $O((\lg p)^3)$ time, as we saw in class. Thus we can compute all the g_j's in $O(n^3(\lg p)^2 + n(\lg p)^3)$ time, and then multiplying by the a_i's and adding gives us a total runtime of $O(n^3(\lg p)^2 + n^2(\lg p)^2 + n(\lg p)^3)$ to compute h modulo p. Since the input is $n \lg p$ bits long, this shows that the running time of the algorithm is polynomial in the input size (in fact, at worst cubic) and thus can be considered efficient.

Polynomials

2. **(2/2/2/2/2) Representing Polynomials**

Let f be a polynomial of degree at most d. The *coefficient representation* of f is the sequence $(a_0, a_1, \dots, a_d)$ of coefficients of f. A *point-value representation* of f is a collection $\{(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_t, f(x_t))\}$ of the values of f at any t points $x_1, x_2, \dots, x_t$, where $t \geq d+1$. (Recall from Lecture Note 7 that a polynomial of degree d is completely determined by its values at any $d+1$ points. Note that t may be greater than $d+1$, so more points than necessary may be given.)

In the following questions, let f and g by any two real polynomials of degree at most d.

(a) What is the maximum degree of the product polynomial fg?

(b) Given coefficient representations of f and g, explain how to compute the coefficient representation of fg using $O(d^2)$ arithmetic operations (additions/subtractions/multiplications/divisions) over real numbers.

(c) Now suppose that f and g are specified by point-value representations at t points for some $t \geq d+1$, i.e., f is specified as $(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_t, f(x_t))$, and g as $(x_1, g(x_1)), (x_2, g(x_2)), \dots, (x_t, g(x_t))$. With a suitable value of t (which you should specify), show how to compute a point-value representation of fg using only $O(d)$ arithmetic operations.

(d) Suppose that polynomial g divides polynomial f, and that f, g are given in point-value representation as in part (c) with $t = d+1$. Show how to compute a point-value representation for the quotient f/g using $O(d)$ arithmetic operations, and justify your algorithm carefully.

(e) Suppose you are given f in coefficient representation, and you want to compute a point-value representation for f at $t = d+1$ points. Show how to do this using $O(d^2)$ arithmetic operations. [HINT: Show how to evaluate f at one point using $O(d)$ operations; to do this, consider writing f in the form $f(x) = a_0 + xh(x)$, where h is a polynomial of degree at most $d-1$, and iterating.]

Answer:

(a) Suppose the polynomial $f(x) = a_0 + a_1x + \cdots a_kx^k$ and $g(x) = b_0 + b_1x + \cdots b_lx^l$ where k and l are at most d. The coefficient c_s of x^s in the product polynomial fg is given by:

$$c_s = a_0b_s + a_1b_{s-1} + \cdots + a_sb_0 = \sum_{i+j=s} a_ib_j \tag{1}$$

The expression shows that the coefficient c_s equals 0 if $s > k+l$. The maximum possible degree of fg is therefore $k+l \le 2d$.

(b) The expression (3) for the coefficients of fg shows that c_s can be computed with at most $s+1$ multiplications and s additions. The number of arithmetic operations to compute one coefficient of fg is at most $2(s+1) = O(d)$ as $s \le 2d$ by part (a). There are at most $2d+1$ non-zero coefficients c_s, so all the coefficients of fg can be computed with $O(d^2)$ arithmetic operations.

(c) Given the tuples $(x_i, f(x_i))$ and $(x_i, g(x_i))$, using one multiplication we can compute $fg(x_i) = f(x_i) \cdot g(x_i)$. Using part (a), it is sufficient to have $2d+1$ tuples in a point-value representation for fg. Hence, provided that $t \ge 2d+1$, a point-value representation of fg can be computed with $2d+1 = O(d)$ multiplications.

(d) Given tuples $(x_i, f(x_i))$ and $(x_i, g(x_i))$, using one division we can compute $[f/g](x_i) = f(x_i)/g(x_i)$ if $g(x_i) \ne 0$. The computation is invalid if and only if the point x_i is a root of g. The number of roots of g is at most $deg(g)$, so $f(x_i)/g(x_i)$ can be evaluated using division on at least $d+1-deg(g)$ points. As the polynomial f is divisible by g we have $d+1-deg(g) \ge deg(f)+1-deg(g) = 1+deg(f/g)$.

Thus, a point-value representation for f/g can be found using $deg(f/g)+1 = O(d)$ divisions by computing $[f/g](x_i) = f(x_i)/g(x_i)$ for all x_i such that $g(x_i) \ne 0$.

(e) We note that $f(x) = a_0 + a_1x + \cdots + a_kx^k = a_0 + x.(a_1 + \cdots a_kx^{k-1})$, showing that a degree-$k$ polynomial can be written as $f(x) = a_0 + x.h(x)$ where the degree of $h(x)$ is $k-1$. It follows that a degree-k polynomial can be evaluated at point x by evaluating a polynomial of degree $k-1$ at x along with two additional arithmetic operations.

The number of arithmetic operations $A(k)$ required to evaluate a degree k polynomial therefore satisfies the recurrence $A(k) = 2 + A(k-1)$. Since a degree-0 polynomial a_0 can be evaluated with zero operations we have $A(0) = 0$, and by induction on k it follows that $A_k = 2k$.

Finally, computing a point-value representation for a polynomial of degree d requires evaluating the polynomial at $d+1$ points. The number of arithmetic operations required is therefore $2d(d+1) = O(d^2)$.

Error Correcting Codes

3. **(5/5) Error-correcting codes: an optimization**

In class, we saw an error-correcting code where the n message packets $m_1, \ldots, m_n$ are encoded to the $n+k$ encoded packets $c_1, \ldots, c_{n+k}$ by setting $P(x) = m_nx^{n-1} + \cdots + m_2x + m_1$, then defining $c_i = P(i)$

(all this is in $GF(q)$, where q is prime and larger than $n+k$, so each packet is a number in the range $0 \ldots q-1$). However, one possible criticism of this error-correcting code is that decoding always requires a Lagrange interpolation step, even if no packets are lost.

(a) In this part, you will develop a scheme that addresses this criticism. Let's preserve the basic approach where $c_i = Q(i)$ (for $i = 1, 2, \ldots, n+k$), for some appropriately chosen polynomial $Q(x)$ which encodes the entire message, and which has degree at most $n-1$. (As before, we'll work in $GF(q)$, where $q > n+k$ and q is prime.) At the same time, let's ensure $c_1 = m_1$, $c_2 = m_2$, ..., $c_n = m_n$, so that if no packets are lost, we can just use the first n encoded packets to immediately read off the message. Describe how to choose $Q(x)$ with this desired property, given $m_1, \ldots, m_n$. In other words, describe an efficient algorithm we can use for encoding.

Answer: Since $c_1 = m_1 = Q(1), \ldots, c_n = m_n = Q(n)$, we can efficiently construct $Q(x)$ using Lagrange interpolation. Specifically,

$$Q(x) = m_1\Delta_1(x) + m_2\Delta_2(x) + \cdots + m_n\Delta_n(x).$$

(b) For your scheme from part 1, if some packets are lost, the recipient can use Lagrange interpolation to recover $Q(x)$. Describe how the recipient could recover $m_1, \ldots, m_n$ from $Q(x)$.

Answer: Since $Q(1) = c_1 = m_1, \ldots, Q(n) = c_n = m_n$, we can recover the message $m_1, \ldots, m_n$ from $Q(x)$ simply by evaluating it at $1, 2, \ldots, n$ respectively.

4. **(5/5) List decoding**

(a) Consider a n character message encoded into m characters over the field $GF(p)$ using polynomials. Consider that one recieves $n-1$ of the m packets. Give a method to find a list of size at most p of possible messages. Your running time must be polynomial in terms of p, m, and n.

(b) Consider a n character message encoded into $m = n+2k$ characters over the field $GF(p)$ using polynomials. Consider that $k+1$ of the m received packets are corrupted. Give a method to find a list of possible messages which contains the original message. What is the size of the list for your scheme. It should be a small polynomial in p. Your running time must be polynomial in terms of p, m, and n.

Answer:

(a) Since we are trying to encode an n character message using polynomials, we are going to fit our message into a degree $n-1$ polynomial and then encode our message into the length m message $[P(0)P(1)\ldots P(m-1)]$. Now, we receive $n-1$ of these characters; suppose without loss of generality that the character at position k, $P(k)$ was not received. Now, we know $n-1$ points of the polynomial $P(x)$ - but knowing these $n-1$ points gives us no information about $P(k)$ since it is possible for us to construct a degree n polynomial that goes through the $n-1$ known points no matter what the value of $P(k)$ is. However, suppose we fix the value of $P(k)$ - then it turns out that there is exactly one polynomial that goes through the $n-1$ known points and $P(k)$, since a degree $n-1$ polynomial is uniquely determined by n of its points. We use this to deduce that there are at most p different polynomials $P(x)$ that could possibly be our encoding polynomial, one for each possible value of $P(k)$ - and thus there are at most p different possible messages that were originally sent. We would generate the p possible messages in the same way:

```
foreach value of l in the range 0 to p-1
    P(x) = interpolate(n-1 known points, P(k) = l)
    generate the possible message [P(0) P(1) ... P(m-1)]
end
```

(b) We can use a similar approach to above; we know that we have $k+1$; if we knew in advance that we were going to have $k+1$ errors we would have sent $n+2k+2$ packets in order to make sure that we could perform error correction using the Berlekamp-Welsh method to decode to the correct message. However, we ended up only sending $n+2k$ packets. If we knew the (correct) values of two more packets, then we could use Berlekamp-Welsh to decode the message. Since we do not know the values of two more packets, we can do what we did in part (a) and just guess what they are to generate possible messages. Since for the value of each packets there are p possible values, at most we will generate p^2 possible messages, and the real message will be included.

```
foreach value of a in the range 0 to p-1
     foreach value of b in the range 0 to p-1
         Use Berlekamp-Welsh with the known values and R(n + 2k + 1) = a
         and R(n + 2k + 2) = b
     end
end
```

5. **(3/3/4) Reed-Solomon and Reliable Computation**

In this question, we will see how error correction can help with faulty computations. Let us first establish two useful facts.

a) For a communication system that uses Reed-Solomon codes, what is the minimum number of additional packets to transmit for an intended message of n packets, knowing that the communication channel will corrupt at most a fraction $0 \leq f < \frac{1}{2}$ of the packets? (e.g. If $f = \frac{1}{4}$, that means that 3 out of 4 transmitted packets will be received flawlessly, but one quarter of them might have their contents corrupted in an arbitrary manner.)

Answer: A Reed-Solomon code which accounts for k general errors with intended message length n is $n+2k$ packets long. By definition, if the channel corrupts at most a fraction f of the packets, then there are at most $(n+2k)f$ corrupted packets. In order to account for these corrupted packets, we must have $k \geq (n+2k)f$, which we can rewrite as:

$$k \geq \frac{nf}{1-2f}$$

And hence, we need at least $2 \times \lfloor \frac{nf}{1-2f} \rfloor$ additional packets along with the intended n packets. Notice that if the channel corrupts more than half of the transmitted packets, no Reed-Solomon code can correct this.

b) Suppose we are using a Reed-Solomon code over $GF(p)$ guarding for k transmission errors. Let $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ be two n-packet messages. Show that the Reed-Solomon codeword for message $a + b = (a_1 + b_1, \ldots, a_n + b_n)$ is the same as the sum of the Reed-Solomon codewords of a and b. In other words, the RS codeword of the element-wise sum is the element-wise sum of the RS codewords.

Answer: Let A, B and C be the message polynomials associated with messages a, b and $a+b$ respectively. By definition, A, B and C are of degree at most $n-1$ and are such that $A(i) = a_i$,

$B(i) = b_i$ and $C(i) = a_i + b_i$ for i in range $1, \dots, n$. We have to show that $C(i) = A(i) + B(i)$ for all i in range $1, \dots, n+2k$.

First, we have $C(i) = a_i + b_i = A(i) + B(i)$ for $1 \leq i \leq n$. What about the remaining $2k$ points? $A + B - C$ is a polynomial of degree at most $n-1$ with n distinct roots $1, \dots, n$. Hence $A + B - C$ must be the zero polynomial ($A + B - C = 0$) which we can rewrite as $C = A + B$, thus finishing the proof.

Suppose you have invented a machine for doing additions extremely fast. Your invention takes a list of pairs of numbers as input and returns the list of the pairs sums. Although the machine is blazing fast, it is at the same time prone to mistakes. Luckily, you can bound the number of mistakes: over all of the n outputs returned, you know that at most $\max(1, \lfloor n/4 \rfloor)$ outputs have an error. For example, if we feed the machine $((2,3),(4,3),(0,7),(4,2))$ we might get back output $(5,7,4,6)$, where $4 \neq 0+7$ is a mistake.

You want to sell your invention, but none of your potential clients is interested in an error-prone device like this. They feel the speed benefit does not compensate for the unreliability of the results.

c) Show that you can augment your machine with a Reed-Solomon encoding and decoding scheme such that no wrong outputs are ever returned. Your clients can use the machine the exact same way as before, but they no longer experience erroneous results. More specifically, you need to define functions $E : GF(p)^n \to GF(p)^m$ and $D : GF(p)^m \to GF(p)^n$ such that $D(M(E(a_1,...,a_n),E(b_1,...,b_n)) = (a_1+b_1,...,a_n+b_n)$, even if M makes errors on up to $1/4$ of its additions.

Answer: Let n the number of input pairs $(a_1,b_1),\dots,(a_n,b_n)$. The idea is to first encode $a = (a_1,\dots,a_n)$ into a' and $b = (b_1,\dots,b_n)$ into b' using an $(n,2k)$-Reed-Solomon code, feed the original machine with the encoded pairs $(a'_1,b'_1),\dots,(a'_{n+2k},b'_{n+2k})$ to get the faulty summations, and then decode the obtained message and return the error-free solution. We are indeed guaranteed to get back the actual summations $(a_1+b_1,\dots,a_n+b_n)$ by question b) which shows that the sum of the Reed-Solomon codes corresponds to the Reed-Solomon code of the sums. Now, the only question is to find the number of errors k we need to account for in the Reed-Solomon scheme.

We have to be careful about the max condition in the error bound. There is at least 1 error independently of the input size n, which means we need $k \geq 1$. Furthermore, when $n \geq 4$ at most a fraction $\frac{1}{4}$ of the packets can get corrupted. Hence, by question a), we need:

$$k \geq \frac{n\frac{1}{4}}{1 - 2\frac{1}{4}} = \frac{n}{2}$$

when $n \geq 4$. Combining these two constraints and ignoring the $n \geq 4$ condition (which does not impact the correctness), we get the following bound:

$$k \geq \max(1, \frac{n}{2})$$

so we can use $2\max(1, \lfloor \frac{n}{2} \rfloor)$ additional inputs to make the scheme work.

Secret Sharing

6. **(3/3/4) Secret Sharing Pirate**

After a long and illustrious career as a buccaneer, Captain Flint passed away in the year 1754. He had split the gold accumulated over years of terrorizing ships into two batches, and just before his death he told his five faithful pirates the locations of the the two batches using a secret sharing scheme.

The captain chose polynomials $P(x)$ and $Q(x)$ over $GF(7)$, of degrees 1 and 2 respectively, with the secrets being the values $P(0)$ and $Q(0)$. For $1 \leq i \leq 5$, Pirate i received the two numbers $P(i)$ and $Q(i)$, but was not told which was which. The pirates cursed the Captain as they could not figure out how to recover the secrets, and the treasure lay undiscovered for many years.

On the eve of the 10th anniversary of the Captain's demise, the pirates captured a small vessel and encountered Monsieur Lagrange (who, having forsaken the ennui of land-life in favor of the vicissitudes of the life on the high seas, was himself an aspiring pirate). Lagrange offered his mathematical prowess in solving the pirates' problems, in exchange for a share of the treasure.

The secret shares received by the five pirates were $\{0,5\},\{1,4\},\{3,4\},\{0,4\}$ and $\{0,3\}$ respectively. (So, for example, Pirate 2's share was $\{1,4\}$, meaning that either $P(2)=1$ and $Q(2)=4$, or $P(2)=4$ and $Q(4)=1$.) Trace the following steps to see how M. Lagrange helped the pirates to solve the mystery.

(a) Find $P(0)+Q(0)$.

(b) Find $P(0)Q(0)$.

(c) Using parts (a) and (b), find the two secrets $P(0)$ and $Q(0)$ that were hidden by Captain Flint, assuming that $P(0) < Q(0)$.

Answer:

Summary: The sum $P(0)+Q(0)$ and the product $P(0)\cdot Q(0)$ of the secrets are recovered using Lagrange interpolation to find the polynomials $P(x)+Q(x)$ and $P(x)\cdot Q(x)$ from the given data. The secrets are obtained from the sum and product by solving a quadratic equation.

(a) $P(x)+Q(x)$ is a polynomial of degree 2 and passes through the points $(i, P(i)+Q(i))$. *(Note that we know all of these points exactly from the given data, because to compute $P(i)+Q(i)$ we don't need to know which value is $P(i)$ and which is $Q(i)$.)* We use the three points $(1,5),(2,5),(3,0)$ and write the Lagrange interpolation formula:

$$P(x)+Q(x) = 5\cdot\Delta_1(x) + 5\cdot\Delta_2(x) + 0\cdot\Delta_3(x) \tag{2}$$

The polynomials Δ_i are given by:

$$\begin{aligned}\Delta_1(x) &= \frac{(x-2)(x-3)}{(1-2)(1-3)} = 4(x^2-5x+6)\\ \Delta_2(x) &= \frac{(x-1)(x-3)}{(2-1)(2-3)} = 6(x^2-4x+3)\end{aligned}$$

We compute $P(x)+Q(x)$ by substituting the expressions for Δ_i in the Lagrange interpolation formula (2):

$$\begin{aligned}P(x)+Q(x) &= 5\cdot\Delta_1(x)+5\cdot\Delta_2(x)\\ &= -1(x^2-5x+6)+2(x^2-4x+3) = x^2-3x\end{aligned}$$

The sum of the secrets $P(0)+Q(0)$ is therefore equal to 0.

(b) $P(x) \cdot Q(x)$ is a polynomial of degree 3 and passes through the points $(i, P(i) \cdot Q(i))$. We use the four points $(1,0),(2,4),(4,0),(5,0)$ and write the Lagrange interpolation formula:

$$P(x) \cdot Q(x) = 0 \cdot \Delta_1(x) + 4 \cdot \Delta_2(x) + 0 \cdot \Delta_4(x) + 0 \cdot \Delta_5(x) \tag{3}$$

Since three of these coefficients are zero, it is sufficient to compute the polynomial Δ_2:

$$\Delta_2(x) \quad = \quad \frac{(x-1)(x-4)(x-5)}{(2-1)(2-4)(2-5)} = 6(x^3 - 3x^2 + x + 1)$$

We compute $P(x) \cdot Q(x)$ by substituting the expression for Δ_2 in the Lagrange interpolation formula (3):

$$\begin{aligned} P(x) \cdot Q(x) &= 4 \cdot \Delta_2(x) \\ &= 3x^3 + 5x^2 + 3x + 3 \end{aligned}$$

The product of the secrets $P(0) \cdot Q(0)$ is therefore equal to 3.

(c) The square of the difference of the secrets is given by $(Q(0) - P(0))^2 = (P(0) + Q(0))^2 - 4 \cdot P(0) \cdot Q(0) = 2$. We observe that the two solutions to $x^2 = 2 \mod 7$ are ± 3 hence $Q(0) - P(0) = \pm 3 \mod 7$. Given that $Q(0) > P(0)$ we find that the secrets are $P(0) = 2$ and $Q(0) = 5$.

Counting

7. **(3/3/4) Counting Subsets**

Consider the set S of all (possibly infinite) subsets of $\mathbb{N}$.

(a) Show that there is a bijection between S and $T = \{f : \mathbb{N} \to \{0,1\}\}$ (the set of all functions that map each natural number to 0 or 1).

(b) Prove or disprove: S is countable.

Answer: Uncountable. Note that such f can be viewed as a binary encoding of a real number between 0 and 1, which exhibits a surjection from V to $[0,1]$.

Let T be a subset of $\mathbb{N}$. Define $f(T) = |T| + \sum_{t \in T} t$. Note that $f(T) \leq (|T|+1)\bar{t}$, where $\bar{t}$ is the largest element in t. Thus, the number of subsets for which $f(T) = a$ is finite for all $a \in \mathbb{N}$. Let $U_a = \{T \mid f(T) = a\}$. Clearly, $S = \bigcup_{a=0}^{\infty} U_a$ is the set of all finite subsets of $\mathbb{N}$. We list the elements of S by listing the elements in U_0, the elements in U_1, and so on. The elements in each U_a are listed in some order (e.g. lexicographically). Since every element of S will appear in this list, S is countable.

(c) Say that a function $f : \mathbb{N} \to \{0,1\}$ has *finite support* if it is non-zero on only a finite set of inputs. Let F denote the set of functions $f : \mathbb{N} \to \{0,1\}$ with finite support.

Prove that F is countably infinite.

Answer: We give a bijection between S and $\mathbb{N}$. We encode an $f \in S$ as a binary number y_f, with the ith position (with $i = 0$ being the least significant digit) set to 1 if $f(i) = 1$. Note that this encoding always has finite length, excluding leading zeros, since the maximum i for which $f(i) = 1$ is finite. Thus the encoding always results in a natural number encoded in binary. This conversion is one-to-one, since each f and f' in S differ on at least one input and therefore y_f and $y_{f'}$ differ in at least one position. The conversion is onto, since every binary number represents a function with finite support. Since the natural numbers are countably infinite, and we have a bijection between S to $\mathbb{N}$, S is countably infinite.

An alternative bijection is between S and the subset of $\mathbb{N}$ that contains only numbers that are the product of distinct primes. Let $\{p_0 = 2, p_1 = 3, p_2 = 5, \cdots\}$ be the set of all primes where p_i is the $(i+1)$th prime. As shown in a previous homework, this set is infinite. Now consider a function $f(x)$ that is 1 exactly on inputs $x_1, x_2, \cdots, x_k$. Encode $f(x)$ as the natural number $p_{x_1} \times p_{x_2} \times \cdots \times p_{x_k}$. In other words, the function f is encoded as the natural number $2^{f(0)} \times 3^{f(1)} \times 5^{f(2)} \times 7^{f(3)} \times 11^{f(4)} \times \cdots$. This encoding is one-to-one, since the prime factorization of a number is unique. The encoding is onto, since every natural number that is composed of distinct primes corresponds to a function in S. Thus this is a bijection, and S is countable.

8. **(2/2/2/2/2) More Countability**

Given:

- A is a countable set, non-empty set. Forall $i \in A$, S_i is an uncountable set.
- B is an uncountable set. Forall $i \in B$, 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) $\cup_{i \in A} S_i$

Answer: Always uncountable. Let a be any elem of A. S_a is uncountable. Thus, $\cup_{i \in A} S_i$, a superset of S_a, is uncountable.

(b) $\cap_{i \in A} S_i$

Answer: Sometimes countable, sometimes uncountable.

Countable: When the S_i are disjoint, the intersection is empty, and thus countable. Formally, let $A = N$, let $S_i = \{i\} \times R = \{(i,x) | x \in R\}$. Then, $\cap_{i \in A} S_i = \emptyset$.

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

(c) $\cup_{i \in B} Q_i$

Answer: Sometimes countable, sometimes uncountable.

Countable: Make all th Q_i identical. Formally, let $B = R$, and $Q_i = N$. Then, $\cup_{i \in B} Q_i = N$, is countable.

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

(d) $\cap_{i \in B} Q_i$

Answer: Always countable. Let b be any elem of B. Q_b is countable. Thus, $\cap_{i \in B} Q_i$, a subset of Q_b, is also countable.

(e) $A \cap B$

Answer: Always countable. $A \cap B = \emptyset$