Notes on Number Theory and Computation

Cardcaptor

March 10, 2010

1 Divisions

- If a and b are integers with $a \neq 0$, we say that a *divides* b if there is an integer c such that $b = ac$.

  When a divides b, we say that a is a *factor* of b and that b is a *multiple* of a.

  We use the notation $a \mid b$ to denote the fact that a divides b.

  For examples, $2 \mid 6$ and $7 \mid 14$.

- Let a and b be integers. Then

  1. if $a \mid b$ and $b \mid c$, then $a \mid c$;
  2. if $a \mid b$, then $a \mid bc$ for all integer c;
  3. if $a \mid b$ and $a \mid c$, then $a \mid (b + c)$.

- **Division Algorithm:** Let a be an integer and d a positive integer. Then there are unique integers q and r such that $a = qd + r$ and $0 \leq r < d$.

  We call q the *quotient* and r the *remainder* of dividing a with d. For example, since $4649 = 110 \times 42 + 29$, we have that 110 is the quotient of dividing 4649 with 42, and 29 is the remainder.

  We use the notation $a \bmod b$ to denote the remainder of dividing a by b; i.e., $4649 \bmod 42 = 29$.

  With modern CPUs, you can compute quotients and remainders in $O(1)$ time. However, computing remainders is consider a very slow operating, consuming quite a lot of CPU cycles.

- An integer a is said to be a *common divisor* of b and c if $a \mid b$ and $b \mid c$.

  For example, 15 is a common divisor of 30 and 45.

- The *greatest common divisor* (GCD) of integers a and b, one of which is not zero, is the largest positive integer that is a common divisor of both a and b. We denote the GCD of a and b with the symbol $\gcd(a, b)$.

  For examples, $\gcd(30, 45) = 15$, $\gcd(2, 7) = 1$, and $\gcd(42, 39) = 3$.

- The GCD have the following properties:

  1. $\gcd(a, b) = \gcd(b, a)$;
  2. $\gcd(a, b) = \gcd(-a, b)$;
  3. $\gcd(a, b) = \gcd(|a|, |b|)$;
  4. $\gcd(a, 0) = |a|$;
  5. $\gcd(a, ka) = |a|$ for all integer k;
  6. (**) $\gcd(a, b)$ is the smallest positive integer in the set $\{ax + by : x, y \in \mathbb{Z}\}$ of *linear combinations* of a and b;

7. if d is a common divisor of a and b, then $d \mid \gcd(a, b)$;
8. $\gcd(na, nb) = n \gcd(a, b)$ for all positive integer n;
9. $\gcd(a, b) = \gcd(a, b + ax)$ for all x;
10. for all positive integer n, a, and b, if $n \mid ab$ and $\gcd(a, n) = 1$, then $n \mid b$.

- *Exercise:* Let us prove the last property of GCD above.

  Since $\gcd(a, n) = 1$, we can find x and y such that $ax + ny = 1$. Now, $n = \gcd(n, ab) \leq \gcd(n, axb) \leq n$. So, it must be the case that $\gcd(n, axb) = n$ as well. Using Property 9, we have that

  $$n = \gcd(n, axb) = \gcd(n, axb + nyb) = \gcd(n, (ax + ny)b) = \gcd(n, b).$$

  This implies that $n \mid b$.

- **GCD Recursion Theorem:** If $b > 0$, then $\gcd(a, b) = \gcd(b, a \bmod b)$.

- **Euclid's algorithm** is an algorithm that computes the GCD of two non-negative integers. It makes use of the GCD recursion theorem to compute the GCD as follows:

  EUCLID(a, b)
  1 **if** $b = 0$
  2 **then return** a
  3 **else return** EUCLID$(b, a \bmod b)$

  For example,

  $$\begin{aligned}\text{EUCLID}(4649, 42) &= \text{EUCLID}(42, 29) = \text{EUCLID}(29, 13) = \text{EUCLID}(13, 3) \\ &= \text{EUCLID}(3, 1) = \text{EUCLID}(1, 0) = 1.\end{aligned}$$

  What is the time complexity of this algorithm? We know that, each time it is called,EUCLID spends $O(1)$ time deciding whether $b = 0$ and calculating $a \bmod b$. Thus, the running time depends on the number of times EUCLID is called recursively.

  The analysis of the running time of Euclid's algorithm involves the Fibonacci number F_k, defined as follow: $F_0 = 0$, $F_1 = 1$, and $F_k = F_{k-1} + F_{k-2}$ for all $k \geq 2$.

  Lemma 1. *If $a > b \geq 1$ and* EUCLID(a, b) *performs $k \geq 1$ recursive calls, then $a \geq F_{k+2}$ and $b \geq F_{k+1}$.*

  Proof. The proof is by induction on k. In the base case, $k = 1$, if EUCLID(a, b) performs 1 recursive call, then $b \neq 0$. So, $b \geq 1 = F_2$. Since $a > b$, we have that $a \geq 2 = F_3$. The base case is established.

  Inductively, assume the lemma is true if at least $k - 1$ calls are made. Consider a and b such that EUCLID(a, b) makes at least k recursive calls. This implies that EUCLID$(b, a \bmod b)$ makes at least $k - 1$ recursive calls. This implies that $b \geq F_{k+1}$ and $(a \bmod b) \geq F_k$. Since $a = qb + (a \bmod b)$ for some $q \geq 1$, we have that

  $$a \geq b + (a \bmod b) \geq F_{k+1} + F_k = F_{k+2}.$$

  We have established the fact that $a \geq F_{k+2}$ and $b \geq F_{k+1}$. So, by induction, the lemma is true for all $k \geq 1$. □

  The contrapositive of the above lemma is the following theorem:

  Theorem 2 (Lamé's Theorem)**.** *Let $k \geq 1$. If $a > b \geq 1$ and $b < F_{k+1}$, then* EUCLID(a, b) *performs fewer than k recursive calls.*

So what's the running time of EUCLID(a, b)? It is $O(k)$ where k is the smallest integer such that $F_{k+1} > b$. We know that $F_k \approx \phi^k/\sqrt{5}$, where $\phi = (1 + \sqrt{5})/2$. So, $k = O(\log b)$. That is, Euclid's algorithm runs in time *linear* in the number of bits used to represent the inputs. In other words, it is linear in the size of the input.

- **Extended Euclid's Algorithm:** We shall see later that it is sometimes useful not only to compute $\gcd(a, b)$ but to also compute x and y such that $\gcd(a, b) = ax + by$.

  Doing so by hand is quite easy. Let us compute $\gcd(48, 30)$:

$$\begin{aligned} 48 &= 1(30) + 18, \\ 30 &= 1(18) + 12, \\ 18 &= 1(12) + 6, \\ 12 &= 2(6). \end{aligned}$$

  Looking at two lines before the last, we have that $6 = 1(18) - 1(12)$ and $12 = 1(30) - 1(18)$. Substituting, we have that $6 = 1(18) - 1\big(1(30) - 1(18)\big) = -1(30) + 2(18)$. Looking at the first line, we have that $18 = 1(48) - 1(30)$. So, $6 = -1(30) + 2\big(1(48) - 1(30)\big) = 2(48) - 3(30)$. So $x = 2$ and $y = -3$.

  Let us codify the process we just went through a little bit. When we compute $\gcd(a, b)$ where $b \neq 0$, we compute $\gcd(b, r)$ where $r = a \bmod b$. Let us be wishful and assume that $\gcd(b, r)$ return x' and y' such that $bx' + ry' = \gcd(b, r) = \gcd(a, b)$. Then, let q be such that $a = qb + r$. We then have $r = a - qb$, and so

$$\gcd(a, b) = bx' + ry' = bx' + (a - qb)y' = ay' + (x' - qy')b.$$

  Thus, we can return $x = y'$ and $y = x' - qy'$.

  The remaining case is when $b = 0$. In this case, it is safe to return $x = 1$ and $y = 0$.

  Let us the above description into pseudocode.

```
EXTENDED-EUCLID(a, b)
1  if b = 0
2     then return (a, 1, 0)
3     else q ← ⌊a/b⌋
4          r ← a mod b
5          (d, x', y') ← EXTENDED-EUCLID(b, r)
6          return (d, y', x' − qy')
```

2 Modular Arithematic

- A *group* $(S, \oplus)$ is a set S together with a binary operation $\oplus : S \times S \to S$ with the following properties:

  1. *Identity:* There exists an element $e \in S$ called the *identity* of the group such that $e \oplus s = s \oplus e$ for all $s \in S$.
  2. *Associativity:* For all $a, b, c \in S$, we have $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.
  3. *Inverses:* For all $a \in S$, there exists $b \in S$. called the *inverse* of a such that $a \oplus b = b \oplus a = e$.

  Examples of groups includes $(\mathbb{Z}, +)$ and $(\mathbb{R} - \{0\}, \times)$. What is the identity of $(\mathbb{R} - \{0\}, \times)$? What is the inverse of a in $(\mathbb{Z}, +)$?

- A group $(S, \oplus)$ is called an *abelian group* if, for all $a, b \in S$, $a \oplus b = b \oplus a$. In other words, $(S, \oplus)$ is abelian if $\oplus$ is commutative.

  $(\mathbb{Z}, +)$ and $(\mathbb{R}, \times)$ are abelian groups.

- Let n be a natural number. Let us consider the binary operation $+_n$ defined as follows: for any $a, b \in \mathbb{Z}$, $a +_n b = (a + b) \bmod n$.

  The operator $+_n$ and the set $\{0, 1, 2, \ldots, n-1\}$ forms a group called the *additive group modulo* n, and it is denoted by the symbol $\mathbb{Z}_n$.

  For example, in $\mathbb{Z}_6$, we have that $1 +_6 5 = 0$ and $4 +_6 5 = 3$.

- Two integers a and b are said to be *congruent modulo* m, for some positive integer m, if $m \mid (a - b)$. This fact is denoted by the symbol $a \equiv b \pmod{m}$.

  For examples, $10 \equiv 1 \pmod{3}$ and $-1 \equiv 14 \pmod{5}$.

- Congruence modulo m has the following properties:

  1. $a \equiv b \pmod{m}$ if and only if $b \equiv a \pmod{m}$;
  2. if $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m}$, then $a \equiv c \pmod{m}$;
  3. $a \equiv b \pmod{m}$ if and only if $a \bmod m = b \bmod m$;
  4. if $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then $a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$;
  5. if $a \equiv b \pmod{m}$ then $a^k \equiv b^k \pmod{m}$ for all non-negative integer k.

  Congruence is pretty much like equality. There's a catch though. In general, if $ac = bc$ and $c \neq 0$, then we know that $a = b$. However, this needs not be true in congruences. For example, we have that $2 \times 4 \equiv 4 \times 4 \pmod{8}$, but $2 \not\equiv 4 \pmod{8}$.

- **Recursive Squaring:** You are given an integer a, a non-negative integer k, and a positive integer m. You are asked to find b such that $0 \leq b < m$ and $a^k \equiv b \pmod{m}$; in other words, $a^k \bmod m$. What's an efficienty to compute this?

  Observe that

$$a^k \bmod m = \begin{cases} 1 & \text{if } k = 0, \\ (a^{k/2} \bmod m)^2 \bmod m, & \text{if } k \text{ is even}, \\ a(a^{(k-1)/2} \bmod m)^2 \bmod m, & \text{if } k \text{ is odd}. \end{cases}$$

  So, we can compute $a^k \bmod m$ by computing $a^{\lfloor k/2 \rfloor}$, squaring it, and multiply by a if k is odd. Putting the idea into pseudocode, we have

```
EXPONENT(a, k, m)
1  if k = 0
2      then return 1
3  elseif k is even
4      then e = EXPONENT(a, k/2, m)
5           return e^2 mod m
6  elseif k is even
7      then e = EXPONENT(a, (k − 1)/2, m)
8           return (ae^2) mod m
```

  The running time of this algorithm is $O(\log k)$.

- What's the solutions to the equation $ax \equiv b \pmod{n}$ for $a > 0$ and $n > 0$?

  First, let us determine if the equation has any solution at all. The equation is equivalent to the statement $n \mid (ax - b)$; in other words, there exist an integer y such that $ny = ax - b$ or $b = ax - ny$. That is, b is in the set of linear combinations of a and n. Using the properties of GCD two pages above, we conclude that b must be divisible by $\gcd(a, n)$.

Proposition 3. *The equation* $ax \equiv b \pmod{n}$ *is solvable if and only if* $\gcd(a, n) \mid b$.

Next, if the equation is solvable, what are the solutions then?

Let $d = \gcd(a, n)$. We can use EXTENDED-EUCLID(a, n) to find x' and y' such that $ax' + ny' = d$. Multiplying both sides by b/d, we have $a((b/d)x') + n((b/d)y') = b$, which means $a((b/d)x') \equiv b \pmod{n}$. Hence, $x = (b/d)x'$ is a solution.

If the equation is solvable, then there are infinitely many solutions. For each integer i, observe that

$$a((b/d)x' + i(n/d)) + n((b/d)y' - i(a/d)) = a((b/d)x') + n((b/d)y') + an/d - an/d = b.$$

So, $(b/d)x' + i(n/d)$ is a solution for all i. We shall show that these are all the solutions.

Observe that if x is a solution, then $x + i(n/d)$ is also a solution. Thus, if there are solutions other than $(b/d)x' + i(n/d)$, then there must be one solution which is equal to $(b/d)x' + k$, where $k < n/d$. So, assume by way of contradiction that such a k exists. We have that $a((b/d)x' + k) \equiv b \pmod{n}$, which implies $ak \equiv 0 \pmod{n}$; in other words, $n \mid ak$. would imply that n/d has to divide $(a/d)k$. Since $\gcd(n/d, a/d) = 1$, n/d must divides k. However, $k < n/d$ so this is impossible. Contradiction.

Theorem 4. *Let* x' *and* y' *are integers such that* $\gcd(a, n) = ax' + ny'$. *Suppose that the equation* $ax \equiv b \pmod{n}$ *is solvable. Then, all the solutions of this equation can be written as* $(b/d)x' + i(n/d)$ *for some* $i \in \mathbb{Z}$.

Collorary 5. *If* $\gcd(a, n) = 1$, *then the equation* $ax \equiv 1 \pmod{n}$ *has a unique solution modulo* n. *(In other words, there exists one and only one* x *such that* $0 \leq x < n$ *and* x *satisfies the equation.)*

- For integer a and positive integer n, a *multiplicative inverse modulo* n of a is an integer such b that $ab \equiv 1 \pmod{n}$.

  From the above corollary, we know that the multiplicative inverse is unique modulo n. We let a^{-1} denotes the unique inverse modulo n of a.

- Two integers a and b, both not 0, are *relatively prime* if $\gcd(a, b) = 1$.

  Integers $a_1, a_2, \ldots, a_k$, none of them 0, are *pairwisely relatively prime* if $\gcd(a_i, a_j) = 1$ for all i, j.

  For example, 6, 5, and 143 are pairwisely relatively prime.

  The nice property of mutually relatively prime set of integers is that, if you pick any two numbers from the set, say x and y, then you can always find x^{-1} modulo y and y^{-1} modulo x. More generally, you can partition the set into two sets, and the products of the numbers in the two sets are relatively prime.

- Consider the following question. A brigade of marines has x soldiers. If the commander makes them line up so that there are 3 soldiers in each row, the last row has 2 soldiers. If the commander makes them line up so that there are there are 4 soldiers in each row, the last row has 1 soldiers. If the commander makes them line up so that there are 5 soldiers in each row, the last row has 3 soldiers. What's the minimum number of soldiers in this brigade?

  This type of question can be answered by making use of the following theorem:

  Theorem 6. *(Chinese Remainder Theorem) Let* $n_1, n_2, \ldots, n_k$ *be pairwisely relatively prime numbers. The system of equation*

  $$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

  has a unique solution modulo $n_1 n_2 \cdots n_k$.

Proof. Let us proof first that there exists at most one solution modulo $n_1 n_2 \cdots n_k$. Suppose that there are x_1 and x_2 that satisfies the above system of equations. Then, $x_1 - x_2 \equiv 0 \pmod{n_i}$ for all i. This is only possible if $x_1 - x_2 = 0$. So $x_1 = x_2$.

Next, we have to show that there exists at least one solution. Let $N = n_1 n_2 \cdots n_k$, and let $N_i = N/n_i$ for all i. Since N_i is relatively prime to n_i, there exists a number I_i such that $N_i I_i \equiv 1 \pmod{n_i}$.

Consider the number $M = N_1 I_1 a_1 + N_2 I_2 a_2 + \cdots + N_k I_k a_k$. We have that

$$\begin{aligned} M &\equiv N_1 I_1 a_1 + N_2 I_2 a_2 + \cdots + N_k I_k a_k && \pmod{n_i} \\ &\equiv N_i I_i a_i + n_i((N_1/n_i) I_1 a_1 + (N_2/n_i) I_2 a_2 + \cdots + (N_k/n_i) I_k a_k) && \pmod{n_i} \\ &\equiv N_i I_i a_i && \pmod{n_i} \\ &\equiv a_i && \pmod{n_i} \end{aligned}$$

for all i. So there exists at least one solution. We are done. □

3 Primes

- A positive integer $p > 1$ is called prime if its only factors are 1 and itself.

  2, 3, 5, 7, 11, 13, 17 are the seven smallest prime numbers.

- Let S be a set, and $\oplus$ and $\otimes$ be binary operations on S. The tuple $(S, \oplus, \otimes)$ is called a *field* if the following properties hold:

  1. *Associativity:* For any $a, b, c \in S$, we have that $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ and $(a \otimes b) \otimes c = a \otimes (b \otimes c)$.
  2. *Commutativity:* For any $a, b \in S$, we have that $a \oplus b = b \oplus a$ and $a \otimes b = b \otimes a$.
  3. *Distributivity:* For any $a, b, c \in S$, we have that $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$.
  4. *Identity:* There exists e and i in S such that $a \oplus e = e \oplus a = a$ and $a \otimes i = i \otimes a = a$ for all a. Here, e is called the *additive identity* and *multiplicative identity*.
  5. *Additive Inverse:* For every $a \in S$, there exists a number b, called the *addtive inverse* of a and denoted by $-a$, such that $a \oplus b = e$.
  6. *Multiplicative Inverse:* For every $a \neq e$ in S, there exists a number b, called the *multiplicative inverse* of a and denoted by a^{-1}, such that $a \otimes b = i$.

  $(\mathbb{R}, +, \times)$ and $(\mathbb{Q}, +, \times)$ are fields. What are their identities?

- Consider the group $\mathbb{Z}_p$ where p is a prime number. Define operator $\cdot_p$ as follows:

  $$a \cdot_p b = (ab) \bmod p.$$

  We have that $(\mathbb{Z}_p, +_p, \cdot_p)$ is a field. We often use $\mathbb{Z}_p$ to denote the field without mentioning the two binary operations. What are the identities of $\mathbb{Z}_p$?

- Given $a \in \mathbb{Z}_p$ such that $a \neq 0$, how does one calculate a^{-1}?

  Well, by using Extended-Euclid.

  Remember that a^{-1} is a number such that $aa^{-1} \equiv 1 \pmod{p}$. Therefore, we can run Extended-Euclid(a, p) to get x and y such that $ax + py = 1$. It follows that $a^{-1} \equiv x \pmod{p}$.

- The following lemma leads to a fundamental theorem in mathematics:

  Lemma 7. *Let a and b be integers and p be a prime. If $p \mid ab$, then $p \mid a$ or $p \mid b$.*

Proof. Assume that p divides ab and p does not divide a. Let $ab = cp$. We have that $\gcd(a, p) = 1$, so there exist x and y such that $ax + py = 1$. This implies that $b = b(ax + py) = bax + bpy = cpx + bpy = p(cx + by)$. Hence, p divides b. □

Theorem 8. *(Fundamental Theorem of Arithemetic) Every non-zero integer n can be expressed as*

$$n = \pm p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k},$$

where the p_i are distinct primes and the e_i are positive integers. Moreover, this expression is unique up to a reordering of the primes.

- Another consequence of $\mathbb{Z}_p$ being a field is the following famous theorem:

  Theorem 9. *(Wilson's Theorem) If p is a prime, then*

  $$(p-1)! \equiv -1 \pmod{p}.$$

  Proof. Consider the set $\{1, 2, \ldots, p-1\}$. Every number except 1 and $p-1$ can be paired with its multiplicative inverse. Hence, $(p-1)! \equiv 1(-1) \equiv -1 \pmod{p}$. □

- We also have another useful theorem:

  Theorem 10. *(Fermat's Little Theorem) If a is an integer and p a prime, then*

  $$a^p \equiv a \pmod{p}.$$

  Proof. The claim is trivially true for all a such that $p \mid a$. So let us assume that a is not a multiple of p. Consider the set $S = \{a, 2a, 3a, \ldots, (p-1)a\}$. We have that there are no two different elements ia and ja in S such that $ia \equiv ja \pmod{p}$ because that would imply that $i \equiv j \pmod{p}$. Note also that none of the elements in S is divisible by p. So, the elements of S, when reduced into $\mathbb{Z}_p$, is equal to the set $\{1, 2, \ldots, p-1\}$. Therefore,

  $$\begin{aligned} a(2a)(3a)\cdots((p-1)a) &\equiv (p-1)! \pmod{p} \\ a^{p-1}(p-1)! &\equiv (p-1)! \pmod{p} \\ a^{p-1} &\equiv 1 \pmod{p} \end{aligned}$$

  It follows that $a^p \equiv a \pmod{p}$. □

4 Hashing

- The *dictionary* ADT supports the following operations:
  - $insert(x)$: Insert x into the dictionary.
  - $remove(x)$: Remove x from the dictionary.
  - $find(x)$: Check whether x is in the dictionary.
- A *hash table* is an implementation of the dictionary ADT where the items to manipulate are integers or can be mapped to the integers. Let us suppose that the items comes from the set $[U] = \{0, 1, \ldots, U-1\}$ for some positive integer U.

  A hash table consists of an array a with M slots, and a *hash functions* $h : [U] \to [m]$. The array a starts empty. The three operations can be implemented as follows:
  - $insert(x)$: Compute $h(x)$ and put x in $a[h(x)]$.

– $remove(x)$: Compute $h(x)$ and remove x from $a[h(x)]$ if it is there.
– $find(x)$: Compute $h(x)$ and check whether x is in $a[h(x)]$.

- The simple implementation above has the problem of *collision*: there may be $x_1, x_2 \in [U]$ such that $x_1 \neq x_2$ and $h(x_1) \neq h(x_2)$. Collision is unaviodable if $U > m$.

- There are many ways to resolve collision. The most simple way is called *chaining*: instead of each slot storing only one item, it stores a linked list of items placed there. It is clear that the performance of hasing with chaining depends on the distribution of items into slots. Any operation involving items that go into a particular slot takes time linear to the number of items already in the slot.

- We are interested in the problem of building *static hash tables*. A fixed set of n items $\{x_1, x_2, \ldots, x_n\} \subseteq [U]$ is given to us. We would like to build a data structure that can perform $find(x)$ — i.e., testing whether x belongs to the set — very fast. An example of an application of static hash table is the English language dictionary.

  We describe a scheme invented by Fredman, Komlós, Szemerédi, which builds a static hash table for n items in $O(n)$ expected time using $O(n)$ space and each $find(x)$ operation takes $O(1)$ worst case time.

- The mathematical tool used in the scheme is the *universal family of hash functions*.

  Definition 11. *A set of hash functions $\mathcal{H}$ is called a* universal family of hash function *if, for all $x, y \in [U]$ such that $x \neq y$,*

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(y)] = \frac{O(1)}{m}.$$

  In other words, if we pick a random hash function out of h, the probability that any two fixed items collide is a constant over the size of the hash table.

- Is it easy to construct a universal family of hash function? Yes. We first pick a prime number $p > U$, and let

$$\mathcal{H}_{p,m} = \{h_{a,b} : a \in \{1, 2, \ldots, p-1\}, b \in \{0, 1, \ldots, p-1\}\},$$

  where

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod m.$$

  Proposition 12. *$\mathcal{H}_{p,m}$ is a universal family of hash function.*

  We prove the following lemma first:

  Lemma 13. *Let $x, y, z, w \in [p]$ be such that $x \neq y$. Then,*

$$\Pr_{(a,b) \leftarrow [p]^2}[(ax + b) \bmod p = z \ \wedge \ (ay + b) \bmod p = w] = \frac{1}{p^2}.$$

  Proof. The fact that $(ax + b) \bmod p = z$ and $(ay + b) \bmod p = w$ is equivalent to the following system of equations:

$$\begin{aligned} ax + b &\equiv z \pmod p \\ ay + b &\equiv w \pmod p \end{aligned}$$

  which is equivalent to the following matrix equation:

$$\begin{bmatrix} x & 1 \\ y & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \equiv \begin{bmatrix} z \\ w \end{bmatrix} \pmod p.$$

  The two-by-two matrix has determinant $x - y$, which is not zero. This means that there is one and only one solution to the system of equation. Since we pick a and b uniformly at random, the probability that a and b make up the solution for the system is $1/p^2$. □

Proof. (Proposition) Fix $x, y \in U$ such that $x \neq y$. We have that $h_{a,b}(x) = h_{a,b}(y)$ if and only if $(ax + b) \bmod p = z$ for some pair of z and w such that $z \equiv w \pmod m$.

We claim that there are at most $4p^2/m$ such (z, w) pairs. To see why, consider the set $0, 1, \ldots, p-1$. It can be partitioned into m subsets based on the value of each element modulo m. Each subset has at most $2p/m$ elements. Any pair of elements in each subset are congruent modulo m, so there are $4p^2/m^2$ pairs per subset. Since there are m subsets, it follows that there are at most $4p^2/m$ pairs.

By the union bound,

$$\begin{aligned}\Pr_{h_{a,b} \leftarrow \mathcal{H}_{p,m}}[h_{a,b}(x) = h_{a,b}(y)] &\leq \sum_{z \equiv w (\bmod m)} \Pr_{(a,b) \leftarrow [p]^2}[(ax+b) \bmod p = z \ \wedge \ (ay+b) \bmod p = w] \\ &= \sum_{z \equiv w (\bmod m)} \frac{1}{p^2} \leq \frac{4p^2}{m}\left(\frac{1}{p^2}\right) = \frac{4}{m}\end{aligned}$$

as desired. □

- We now describe the construction of FKS dictionary. We pick $m > cn$ for some constant c and let $\mathcal{H}$ be a universal family of hash functions. We try pick h from $\mathcal{H}$ at random, and hash $x_1, x_2, \ldots, x_n$ into the table. We stop if there are no more than n collisions. Otherwise, we pick a new hash function an try again.

  We show that the above process takes $O(n)$ expected time. Let I_{ij} be an indicator random variable such that $I_{ij} = 1$ if and only if $x_i = x_j$. Then,

$$E[\#\text{collisions}] = E\left[\sum_{1 \leq i < j < n} I_{ij}\right] = \sum_{1 \leq i < j < n} \mathbf{E}[I_{ij}] = \sum_{1 \leq i < j < n} \frac{O(1)}{cn} < \frac{n^2}{2}\frac{O(1)}{cn} = \frac{O(1)}{2c}n.$$

  We can use c large enough such that the expected number of collisions is less than $n/2$. By Markov's inequality, the probability that the number of collisions is greater than n is less than $1/2$. Thus, the expected number of trials is constant. Since each trial takes linear time, the process takes expected linear time.

  Now, we look to the m slots of the hash table produced above. Let there be n_i elements in slot $a[i]$. For each slot, we create a *collision-free* hash table of size cn_i^2 by the process above: we pick a hash function from h, hash the elements, and try again if there's a collision. We claim that this process takes $O(n_i^2)$ expected time because

$$E[\#\text{collisions}] \leq \frac{n_i^2}{2}\frac{O(1)}{cn_i^2} = \frac{O(1)}{2c}.$$

  Summing the time and space used for all slots together, the total time and space taken is $O(n_0^2) + O(n_1^2) + \ldots + O(n_{m-1}^2) = O(n)$ because the total number of collisions is less than $n_0^2 + n_1^2 + \cdots + n_{p-1}^2$.