11 Hash Tables

11.1 Direct-address tables

11.1-1

> Suppose that a dynamic set S is represented by a direct-address table T of length m. Describe a procedure that finds the maximum element of S. What is the worst-case performance of your procedure?

As the dynamic set S is represented by the direct-address table T, for each key k in S, there is a slot k in T points to it. If no element with key k in S, then $T[k] = \text{NIL}$. Using this property, we can find the maximum element of S by traversing down from the highest slot to seek the first non-NIL one.

```
MAXIMUM(S)
    return TABLE-MAXIMUM(T, m - 1)
```

```
TABLE-MAXIMUM(T, l)
    if l < 0
        return NIL
    else if DIRECT-ADDRESS-SEARCH(T, l) != NIL
        return l
    else return TABLE-MAXIMUM(T, l - 1)
```

The TABLE-MAXIMUM procedure gest down and checks 1 sloc at a time, linearly approaches the solution. In the worst case where S is empty, TABLE-MAXIMUM examines m slots. Therefore, the worst-case performance of MAXIMUM is $O(m)$, where m is the length of the direct-address table T.

11.1-2

> A ***bit vector*** is simply an array of bits (0s and 1s). A bit vector of length m takes much less space than an array of m pointers. Describe how to use a bit vector to represent a dynamic set of distinct elements with no satellite data. Dictionary operations should run in $O(1)$ time.

Using the bit vector data structure, we can represent keys less than m by a string of m bits, denoted by $V[0..m-1]$, in which each position that occupied by the bit 1, corresponds to a key in the set S. If the set contains no element with key k, then $V[k] = 0$. For instance, we can store the set $\{2, 4, 6, 10, 16\}$ in a bit vector of length 20:

$$00101010000100000010000$$

```
BITMAP-SEARCH(V, k)
    if V[k] != 0
        return k
    else return NIL
```

```
BITMAP-INSERT(V, x)
    V[x] = 1
```

```
BITMAP-DELETE(V, x)
    V[x] = 0
```

Each of these operations takes only $O(1)$ time.

11.1-3

> Suggest how to implement a direct-address table in which the keys of stored elements do not need to be distinct and the elements can have satellite data. All three dictionary operations (INSERT, DELETE, and SEARCH) should run in $O(1)$ time. (Don't forget that DELETE takes as an argument a pointer to an object to be deleted, not a key.)

Assuming that fetching an element should return the satellite data of all the stored elements, we can have each key map to a doubly linked list.

- INSERT: appends the element to the list in constant time
- DELETE: removes the element from the linked list in constant time (the element contains pointers to the previous and next element)
- SEARCH: returns the first element, which is a node in a linked list, in constant time

11.1-4 ⋆

> We wish to implement a dictionary by using direct addressing on a *huge* array. At the start, the array entries may contain garbage, and initializing the entire array is impractical because of its size. Describe a scheme for implementing a direct-address dictionary on a huge array. Each stored object should use $O(1)$ space; the operations SEARCH, INSERT, and DELETE should take $O(1)$ time each; and initializing the data structure should take $O(1)$ time. (Hint: Use an additional array, treated somewhat like a stack whose size is the number of keys actually stored in the dictionary, to help determine whether a given entry in the huge array is valid or not.)

The additional data structure will be a stack S.

Initially, set S to be empty, and do nothing to initialize the huge array. Each object stored in the huge array will have two parts: the key value, and a pointer to an element of S, which contains a pointer back to the object in the huge array.

- To insert x, push an element y to the stack which contains a pointer to position x in the huge array. Update position $A[x]$ in the huge array A to contain a pointer to y in S.
- To search for x, go to position x of A and go to the location stored there. If that location is an element of S which contains a pointer to $A[x]$, then we know x is in A. Otherwise, $x \notin A$.
- To delete x, invalidate the element of S which is pointed to by $A[x]$. Because there may be "holes" in S now, we need to pop an item from S, move it to the position of the "hole", and update the pointer in A accordingly. Each of these takes $O(1)$ time and there are at most as many elements in S as there are valid elements in A.

11.2 Hash tables

11.2-1

> Suppose we use a hash function h to hash n distinct keys into an array T of length m. Assuming simple uniform hashing, what is the expected number of collisions? More precisely, what is the expected cardinality of $\{\{k, l\} : k \neq l \text{ and } h(k) = h(l)\}$?

Under the assumption of simple uniform hashing, we will use linearity of expectation to compute this.

Suppose that all the keys are totally ordered $\{k_1, \ldots, k_n\}$. Let X_i be the number of ℓ's such that $\ell > k_i$ and $h(\ell) = h(k_i)$. So X_i is the (expected) number of times that key k_i is collided by those keys hashed afterward. Note, that this is the same thing as $\sum_{j>i} \Pr(h(k_j) = h(k_i)) = \sum_{j>i} 1/m = (n-i)/m$. Then, by linearity of expectation, the number of collisions is the sum of the number of collisions for each possible smallest element in the collision. The expected number of collisions is

$$\sum_{i=1}^{n} \frac{n-i}{m} = \frac{n^2 - n(n+1)/2}{m} = \frac{n^2 - n}{2m}.$$

11.2-2

> Demonstrate what happens when we insert the keys $5, 28, 19, 15, 20, 33, 12, 17, 10$ into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \mod 9$.

Let us number our slots $0, 1, \ldots, 8$.

Then our resulting hash table will look like following:

h(k)	keys
$0 \mod 9$	
$1 \mod 9$	$10 \to 19 \to 28$
$2 \mod 9$	20
$3 \mod 9$	12
$4 \mod 9$	
$5 \mod 9$	5
$6 \mod 9$	$33 \to 15$
$7 \mod 9$	
$8 \mod 9$	17

11.2-3

> Professor Marley hypothesizes that he can obtain substantial performance gains by modifying the chaining scheme to keep each list in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions, and deletions?

- Successful searches: no difference, $\Theta(1 + \alpha)$.
- Unsuccessful searches: faster but still $\Theta(1 + \alpha)$.
- Insertions: same as successful searches, $\Theta(1 + \alpha)$.
- Deletions: same as before if we use doubly linked lists, $\Theta(1)$.

11.2-4

> Suggest how to allocate and deallocate storage for elements within the hash table itself by linking all unused slots into a free list. Assume that one slot can store a flag and either one element plus a pointer or two pointers. All dictionary and free-list operations should run in $O(1)$ expected time. Does the free list need to be doubly linked, or does a singly linked free list suffice?

The flag in each slot of the hash table will be 1 if the element contains a value, and 0 if it is free. The free list must be doubly linked.

- Search is unmodified, so it has expected time $O(1)$.
- To insert an element x, first check if $T[h(x.key)]$ is free. If it is, delete $T[h(x.key)]$ and change the flag of $T[h(x.key)]$ to 1. If it wasn't free to begin with, simply insert $x.key$ at the start of the list stored there.
- To delete, first check if $x.prev$ and $x.next$ are NIL. If they are, then the list will be empty upon deletion of x, so insert $T[h(x.key)]$ into the free list, update the flag of $T[h(x.key)]$ to 0, and delete x from the list it's stored in. Since deletion of an element from a singly linked list isn't $O(1)$, we must use a doubly linked list.
- All other operations are $O(1)$.

11.2-5

> Suppose that we are storing a set of n keys into a hash table of size m. Show that if the keys are drawn from a universe U with $|U| > nm$, then U has a subset of size n consisting of keys that all hash to the same slot, so that the worst-case searching time for hashing with chaining is $\Theta(n)$.

Suppose the $m - 1$ slots contains at most $n - 1$ elements, then the remaining slot should have

$$|U| - (m - 1)(n - 1) > nm - (m - 1)(n - 1) = n + m - 1 \geq n$$

elements, thus U has a subset of size n.

11.2-6

> Suppose we have stored n keys in a hash table of size m, with collisions resolved by chaining, and that we know the length of each chain, including the length L of the longest chain. Describe a procedure that selects a key uniformly at random from among the keys in the hash table and returns it in expected time $O(L \cdot (1 + 1/\alpha))$.

☑ = p(1 + \alpha) + (1 – p)(1 + E[X])sincewe′dexpecttotake1 + \alpha stepstoreachanelementonthelist, andsinceweknowhowmanyelementsareoneachlist, iftheelementdoesn′texistwe′llknowrightaway. Thenwehave E[X] = \alpha + 1 / p. Theprobabilityofpickingaparticularelementisn / mL = \alpha / L$. Therefore, we have

$$ \begin{aligned} ☑ & = \alpha + L / \alpha \\ & = L(\alpha/L + 1/\alpha) \\ & = O(L(1 + 1/\alpha)) \end{aligned} $$

since $\alpha \leq L$.

11.3 Hash functions

11.3-1

> Suppose we wish to search a linked list of length n, where each element contains a key k along with a hash value $h(k)$. Each key is a long character string. How might we take advantage of the hash values when searching the list for an element with a given key?

If every element also contained a hash of the long character string, when we are searching for the desired element, we'll first check if the hashvalue of the node in the linked list, and move on if it disagrees. This can increase the runtime by a factor proportional to the length of the long character strings.

11.3-2

> Suppose that we hash a string of r characters into m slots by treating it as a radix-128 number and then using the division method. We can easily represent the number m as a 32-bit computer word, but the string of r characters, treated as a radix-128 number, takes many words. How can we apply the division method to compute the hash value of the character string without using more than a constant number of words of storage outside the string itself?

```
sum = 0
for i = 1 to r
    sum = (sum * 128 + s[i]) % m
```

Use `sum` as the key.

11.3-3

> Consider a version of the division method in which $h(k) = k \mod m$, where $m = 2^p - 1$ and k is a character string interpreted in radix 2^p. Show that if we can derive string x from string y by permuting its characters, then x and y hash to the same value. Give an example of an application in which this property would be undesirable in a hash function.

We will show that each string hashes to the sum of it's digits $\mod 2^p - 1$. We will do this by induction on the length of the string.

- Base case

  Suppose the string is a single character, then the value of that character is the value of k which is then taken $\mod m$.

- Inductive step.

  Let $w = w_1 w_2$ where $|w_1| \geq 1$ and $|w_2| = 1$. Suppose $h(w_1) = k_1$. Then,
  $h(w) = h(w_1)2^p + h(w_2) \mod 2^p - 1 = h(w_1) + h(w_2) \mod 2^p - 1$. So, since $h(w_1)$ was the sum of all but the last digit $\mod m$, and we are adding the last digit $\mod m$, we have the desired conclusion.

11.3-4

> Consider a hash table of size $m = 1000$ and a corresponding hash function $h(k) = \lfloor m(kA \mod 1) \rfloor$ for $A = (\sqrt{5} - 1)/2$. Compute the locations to which the keys 61, 62, 63, 64, and 65 are mapped.

- $h(61) = \lfloor 1000(61 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 700$.
- $h(62) = \lfloor 1000(62 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 318$.
- $h(63) = \lfloor 1000(63 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 936$.
- $h(64) = \lfloor 1000(64 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 554$.
- $h(65) = \lfloor 1000(65 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 172$.

11.3-5 ⋆

> Define a family $\square$ of hash functions from a finite set U to a finite set B to be ϵ-***universal*** if for all pairs of distinct elements k and l in U,
>
> $$\Pr\{h(k) = h(l)\} \leq \epsilon,$$
>
> where the probability is over the choice of the hash function h drawn at random from the family $\square$. Show that an ϵ-universal family of hash functions must have

$$\epsilon \geq \frac{1}{|B|} - \frac{1}{|U|}.$$

As a simplifying assumption, assume that $|B|$ divides $|U|$. It's just a bit messier if it doesn't divide evenly.

Suppose to a contradiction that $\epsilon > \frac{1}{|B|} - \frac{1}{|U|}$. This means that $\forall$ pairs $k, \ell \in U$, we have that the number $n_{k,\ell}$ of hash functions in $\square$ that have a collision on those two elements satisfies $n_{k,\ell} \leq \frac{|\square|}{|B|} - \frac{|\square|}{|U|}$. So, summing over all pairs of elements in U, we have that the total number is $\leq \frac{|\square||U|^2}{2|B|} - \frac{|\square||U|}{2}$.

Any particular hash function must have that there are at least $|B|\binom{|U|/|B|}{2} = |B|\frac{|U|^2-|U||B|}{2|B|^2} = \frac{|U|^2}{2|B|} - \frac{|U|}{2}$ colliding pairs for that hash function, summing over all hash functions, we get that there are at least $|\square|\left(\frac{|U|^2}{2|B|} - \frac{|U|}{2}\right)$ colliding pairs total. Since we have that there are at most some number less than this many, we have a contradiction, and so must have the desired restriction on ϵ.

11.3-6 ⋆

Let U be the set of n-tuples of values drawn from $\mathbb{Z}_p$, and let $B = \mathbb{Z}_p$, where p is prime. Define the hash function $h_b : U \to B$ for $b \in \mathbb{Z}_p$ on an input n-tuple $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ from U as

$$h_b(\langle a_0, a_1, \ldots, a_{n-1} \rangle) = \left(\sum_{j=0}^{n-1} a_j b^j\right) \mod p,$$

and let $\square = \{h_b : b \in \mathbb{Z}_p\}$. Argue that $\square$ is $((n-1)/p)$-universal according to the definition of ϵ-universal in Exercise 11.3-5. (Hint: See Exercise 31.4-4.)

Fix $b \in \mathbb{Z}_p$. By exercise 31.4-4, $h_b(x)$ collides with $h_b(y)$ for at most $n-1$ other $y \in U$. Since there are a total of p possible values that h_b takes on, the probability that $h_b(x) = h_b(y)$ is bounded from above by $\frac{n-1}{p}$, since this holds for any value of b, $\square$ is $((n-1)/p)$-universal.

11.4 Open addressing

11.4-1

Consider inserting the keys $10, 22, 31, 4, 15, 28, 17, 88, 59$ into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h'(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \mod (m-1))$.

We use T_t to represent each time stamp t starting with $i = 0$, and if encountering a collision, then we iterate i from $i = 1$ to $i = m - 1 = 10$ until there is no collision.

- **Linear probing**:

$h(k,i) = (k+i) \mod 11$	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
$0 \mod 11$		22	22	22	22	22	22	22	22
$1 \mod 11$								88	88
$2 \mod 11$									
$3 \mod 11$									
$4 \mod 11$				4	4	4	4	4	4
$5 \mod 11$					15	15	15	15	15
$6 \mod 11$						28	28	28	28
$7 \mod 11$							17	17	17
$8 \mod 11$									59
$9 \mod 11$			31	31	31	31	31	31	31
$10 \mod 11$	10	10	10	10	10	10	10	10	10

- **Quadradic probing**, it will look identical until there is a collision on inserting the fifth element:

$h(k, i) = (k + i + 3i^2) \mod 11$	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
$0 \mod 11$		22	22	22	22	22	22	22	22
$1 \mod 11$									
$2 \mod 11$								88	88
$3 \mod 11$							17	17	17
$4 \mod 11$				4	4	4	4	4	4
$5 \mod 11$									
$6 \mod 11$						28	28	28	28
$7 \mod 11$									59
$8 \mod 11$					15	15	15	15	15
$9 \mod 11$			31	31	31	31	31	31	31
$10 \mod 11$	10	10	10	10	10	10	10	10	10

Note that there is no way to insert the element 59 now, because the offsets coming from $c_1 = 1$ and $c_2 = 3$ can only be even, and an odd offset would be required to insert 59 because $59 \mod 11 = 4$ and all the empty positions are at odd indices.

- **Double hashing**:

$h(k, i) = (k + i(1 + k \mod 10)) \mod 11$	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
$0 \mod 11$		22	22	22	22	22	22	22	22
$1 \mod 11$									
$2 \mod 11$									59
$3 \mod 11$							17	17	17
$4 \mod 11$				4	4	4	4	4	4
$5 \mod 11$					15	15	15	15	15
$6 \mod 11$						28	28	28	28
$7 \mod 11$								88	88
$8 \mod 11$									
$9 \mod 11$			31	31	31	31	31	31	31
$10 \mod 11$	10	10	10	10	10	10	10	10	10

11.4-2

> Write pseudocode for HASH-DELETE as outlined in the text, and modify HASH-INSERT to handle the special value DELETED.

```
HASH-DELETE(T, k)
    i = 0
    repeat
        j = h(k, i)
        if T[j] == k
            T[j] = DELETE
            return j
        else i = i + 1
    until T[j] == NIL or i == m
    error "element not exist"
```

By implementing HASH-DELETE in this way, the HASH-INSERT need to be modified to treat NIL slots as empty ones.

```
HASH-INSERT(T, k)
    i = 0
    repeat
        j = h(k, i)
        if T[j] == NIL or T[j] == DELETE
            T[j] = k
            return j
        else i = i + 1
```

```
    until i == m
    error "hash table overflow"
```

11.4-3

> Consider an open-address hash table with uniform hashing. Give upper bounds on the expected number of probes in an unsuccessful search and on the expected number of probes in a successful search when the load factor is $3/4$ and when it is $7/8$.

- $\alpha = 3/4$,
  - unsuccessful: $\frac{1}{1-\frac{3}{4}} = 4$ probes,
  - successful: $\frac{1}{\frac{3}{4}} \ln \frac{1}{1-\frac{3}{4}} \approx 1.848$ probes.
- $\alpha = 7/8$,
  - unsuccessful: $\frac{1}{1-\frac{7}{8}} = 8$ probes,
  - successful: $\frac{1}{\frac{7}{8}} \ln \frac{1}{1-\frac{7}{8}} \approx 2.377$ probes.

11.4-4 ⋆

> Suppose that we use double hashing to resolve collisions—that is, we use the hash function $h(k, i) = (h_1(k) + ih_2(k)) \mod m$. Show that if m and $h_2(k)$ have greatest common divisor $d \geq 1$ for some key k, then an unsuccessful search for key k examines $(1/d)$th of the hash table before returning to slot $h_1(k)$. Thus, when $d = 1$, so that m and $h_2(k)$ are relatively prime, the search may examine the entire hash table. (Hint: See Chapter 31.)

Suppose $d = \gcd(m, h_2(k))$, the LCM $l = m \cdot h_2(k)/d$.

Since $d | h_2(k)$, then $m \cdot h_2(k)/d \mod m = 0 \cdot (h_2(k)/d \mod m) = 0$, therefore $(l + ih_2(k)) \mod m = ih_2(k) \mod m$, which means $ih_2(k) \mod m$ has a period of m/d.

11.4-5 ⋆

> Consider an open-address hash table with a load factor α. Find the nonzero value α for which the expected number of probes in an unsuccessful search equals twice the expected number of probes in a successful search. Use the upper bounds given by Theorems 11.6 and 11.8 for these expected numbers of probes.

$$\frac{1}{1-\alpha} = 2 \cdot \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$
$$\alpha = 0.71533.$$

11.5 Perfect hashing

11.5-1 ⋆

> Suppose that we insert n keys into a hash table of size m using open addressing and uniform hashing. Let $p(n, m)$ be the probability that no collisions occur. Show that $p(n, m) \leq e^{-n(n-1)/2m}$. (Hint: See equation (3.12).) Argue that when n exceeds $\sqrt{m}$, the probability of avoiding collisions goes rapidly to zero.

$$\begin{aligned} p(n, m) &= \frac{m}{m} \cdot \frac{m-1}{m} \cdots \frac{m-n+1}{m} \\ &= \frac{m \cdot (m-1) \cdots (m-n+1)}{m^n}. \end{aligned}$$

$$\begin{aligned} (m-i) \cdot (m-n+i) &= (m - \frac{n}{2} + \frac{n}{2} - i) \cdot (m - \frac{n}{2} - \frac{n}{2} + i) \\ &= (m - \frac{n}{2})^2 - (i - \frac{n}{2})^2 \\ &\leq (m - \frac{n}{2})^2. \end{aligned}$$

$$\begin{aligned} p(n, m) &\leq \frac{m \cdot (m - \frac{n}{2})^{n-1}}{m^n} \\ &= (1 - \frac{n}{2m})^{n-1} . \end{aligned}$$

Based on equation (3.12), $e^x \geq 1 + x$,

$$\begin{aligned} p(n, m) &\leq (e^{-n/2m})^{n-1} \\ &= e^{-n(n-1)/2m} . \end{aligned}$$

Problem 11-1 Longest-probe bound for hashing

Suppose that we use an open-addressed hash table of size m to store $n \leq m/2$ items.

a. Assuming uniform hashing, show that for $i = 1, 2, \ldots, n$, the probability is at most 2^{-k} that the ith insertion requires strictly more than k probes.

b. Show that for $i = 1, 2, \ldots, n$, the probability is $O(1/n^2)$ that the ith insertion requires more than $2 \lg n$ probes.

Let the random variable X_i denote the number of probes required by the ith insertion. You have shown in part (b) that $\Pr\{X_i > 2 \lg n\} = O(1/n^2)$. Let the random variable $X = \max_{1 \leq i \leq n} X_i$ denote the maximum number of probes required by any of the n insertions.

c. Show that $\Pr\{X > 2 \lg n\} = O(1/n)$.

d. Show that the expected length $E[X]$ of the longest probe sequence is $O(\lg n)$.

(Removed)

Problem 11-2 Slot-size bound for chaining

Suppose that we have a hash table with n slots, with collisions resolved by chaining, and suppose that n keys are inserted into the table. Each key is equally likely to be hashed to each slot. Let M be the maximum number of keys in any slot after all the keys have been inserted. Your mission is to prove an $O(\lg n / \lg \lg n)$ upper bound on $E[M]$, the expected value of M.

a. Argue that the probability Q_k that exactly k keys hash to a particular slot is given by

$$Q_k = \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k}.$$

b. Let P_k be the probability that $M = k$, that is, the probability that the slot containing the most keys contains k keys. Show that $P_k \leq nQ_k$.

c. Use Stirling's approximation, equation (3.18), to show that $Q_k < e^k / k^k$.

d. Show that there exists a constant $c > 1$ such that $Q_{k_0} < 1/n^3$ for $k_0 = c \lg n / \lg \lg n$. Conclude that $P_k < 1/n^2$ for $k \geq k_0 = c \lg n / \lg \lg n$.

e. Argue that

$$E[M] \leq \Pr\left\{M > \frac{c \lg n}{\lg \lg n}\right\} \cdot n + \Pr\left\{M \leq \frac{c \lg n}{\lg \lg n}\right\} \cdot \frac{c \lg n}{\lg \lg n}.$$

Conclude that $E[M] = O(\lg n / \lg \lg n)$.

(Removed)

Problem 11-3 Quadratic probing

Suppose that we are given a key k to search for in a hash table with positions $0, 1, \ldots, m - 1$, and suppose that we have a hash function h mapping the key space into the set $\{0, 1, \ldots, m - 1\}$. The search scheme is as follows:

1. Compute the value $j = h(k)$, and set $i = 0$.
2. Probe in position j for the desired key k. If you find it, or if this position is empty, terminate the search.
3. Set $i = i + 1$. If i now equals m, the table is full, so terminate the search. Otherwise, set $j = (i + j) \mod m$, and return to step 2.

> Assume that m is a power of 2.
>
> **a.** Show that this scheme is an instance of the general "quadratic probing" scheme by exhibiting the appropriate constants c_1 and c_2 for equation (11.5).
>
> **b.** Prove that this algorithm examines every table position in the worst case.

(Removed)

Problem 11-4 Hashing and authentication

> Let $\square$ be a class of hash functions in which each hash function $h \in \square$ maps the universe U of keys to $\{0, 1, \ldots, m-1\}$. We say that $\square$ is ***k-universal*** if, for every fixed sequence of k distinct keys $\langle x^{(1)}, x^{(2)}, \ldots, x^{(k)} \rangle$ and for any h chosen at random from $\square$, the sequence $\langle h(x^{(1)}), h(x^{(2)}), \ldots, h(x^{(k)}) \rangle$ is equally likely to be any of the m^k sequences of length k with elements drawn from $\{0, 1, \ldots, m-1\}$.
>
> **a.** Show that if the family $\square$ of hash functions is 2-universal, then it is universal.
>
> **b.** Suppose that the universe U is the set of n-tuples of values drawn from $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, where p is prime. Consider an element $x = \langle x_0, x_1, \ldots, x_{n-1} \rangle \in U$. For any n-tuple $a = \langle a_0, a_1, \ldots, a_{n-1} \rangle \in U$, define the hash function h_a by
>
> $$h_a(x) = \left(\sum_{j=0}^{n-1} a_j x_j \right) \mod p.$$
>
> Let $\square = \{h_a\}$. Show that $\square$ is universal, but not 2-universal. (Hint: Find a key for which all hash functions in $\square$ produce the same value.)
>
> **c.** Suppose that we modify $\square$ slightly from part (b): for any $a \in U$ and for any $b \in \mathbb{Z}_p$, define
>
> $$h'_{ab}(x) = \left(\sum_{j=0}^{n-1} a_j x_j + b \right) \mod p$$
>
> and $h' = \{h'_{ab}\}$. Argue that $\square'$ is 2-universal. (Hint: Consider fixed n-tuples $x \in U$ and $y \in U$, with $x_i \neq y_i$ for some i. What happens to $h'_{ab}(x)$ and $h'_{ab}(y)$ as a_i and b range over $\mathbb{Z}_p$?)
>
> **d.** Suppose that Alice and Bob secretly agree on a hash function h form 2-universal family $\square$ of hash functions. Each $h \in \square$ maps from a universe of keys u to $\mathbb{Z}_p$, where p is aprime. Later, Alice sends a message m to Bob over the Internet, where $m \in U$. She authenticates this message to Bob by also sending an authentication tag $t = h(m)$, and Bob checks that the pair (m, t) he receives indeed satisfies $t = h(m)$. Suppose that an adversary intercepts (m, t) en route and tries to fool Bob by replacing the pair (m, t) with a different pair (m', t'). Argue that the probability that the adversary succeeds in fooling Bob into accepting (m', t') is at most $1/p$, no matter how much computing power the adversary has, and even if the adversary knows the family $\square$ of hash functions used.

a. The number of hash functions for which $h(k) = h(l)$ is $\frac{m}{m^2}|\square| = \frac{1}{m}|\square|$, therefore the family is universal.

b. For $x = \langle 0, 0, \ldots, 0 \rangle$, $\square$ could not be 2-universal.

c. Let $x, y \in U$ be fixed, distinct n-tuples. As a_i and b range over $\mathbb{Z}_p$, $h'_{ab}(x)$ is equally likely to achieve every value from 1 to p since for any sequence a, we can let b vary from 1 to $p-1$.

Thus, $\langle h'_{ab}(x), h'_{ab}(y) \rangle$ is equally likely to be any of the p^2 sequences, so $\square$ is 2-universal.

d. Since $\square$ is 2-universal, every pair of $\langle t, t' \rangle$ is equally likely to appear, thus t' could be any value from $\mathbb{Z}_p$. Even the adversary knows $\square$, since $\square$ is 2-universal, then $\square$ is universal, the probability of choosing a hash function that $h(k) = h(l)$ is at most $1/p$, therefore the probability is at most $1/p$.