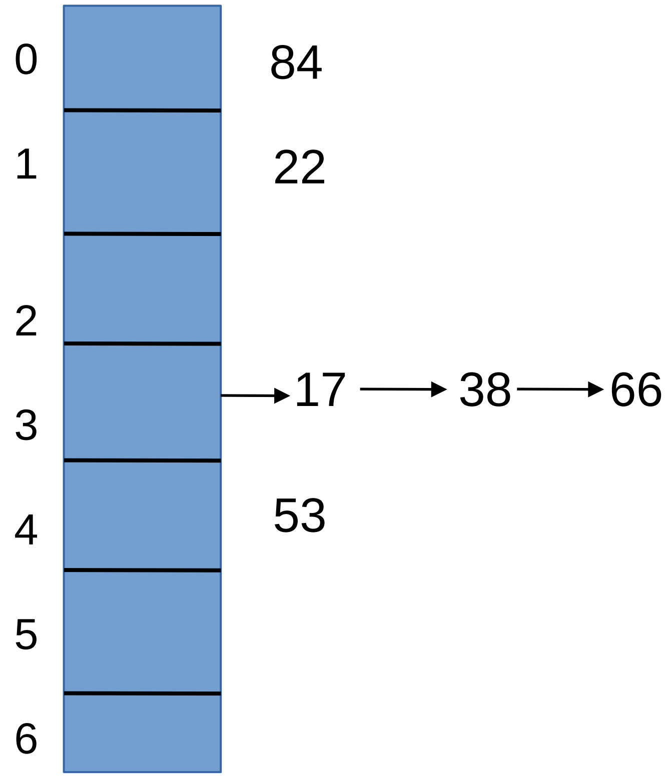
Universal Hashing

Quick Introduction to Hashing

$U = [1..100]$

$S = \{17, 22, 53, 84, 38, 66\}$

$h(x) = x \bmod 7$



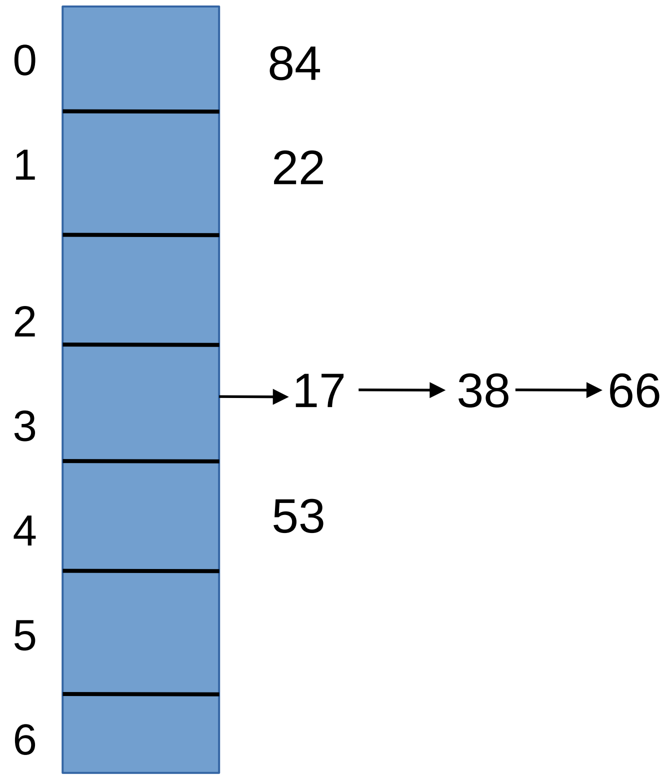
- Consider the simple technique of hashing.
- Each element u from a subset S of a universe U is mapped to an index in a table called the hash table.
- The map is called as the hash function denote h().

Quick Introduction to Hashing

U = [1..100]

S = {17, 22, 53, 84, 38, 66}

h(x) = x mod 7



- It is clear by now that when the domain of h is U and the range is a small set of indices of T, the map can never be one-to-one.
  - In other words, x, y in U such that h(x) = h(y).
  - This situation is called as a collision.

Quick Introduction to Hashing

- How to handle collisions?
- There are several techniques.
- Chaining is one of them.
  - All the elements of S that have a collision under a given hash function h() are linked in a singly linked list at the corresponding index of T.

Quick Introduction to Hashing

- Collisions however can adversely impact the performance of hashing.
- The time taken to search/insert/delete now depends on the length of any list (chain).
- What is the expected length of any list?
- The answer depends on the nature of the hash function used.
- We will assume that the hash function h is such that any element of U is equally likely to hash into any of the slots in T, independent of the other elements and their hash values.
  - This assumption is called as the simple uniform hashing assumption.

Simple Uniform Hashing

- Some notation:
  - Let $|T| = m$.
  - Let n_i denote the number of elements mapped to index i of T for $i = 0,1,2,\ldots, m-1$.
  - $|S| = n$.
  - Therefore, $n = n_0 + n_1 + n_2 + \ldots + n_{m-1}$.
- What is the average value of n_i?
  - $En_i = \Sigma\, En_{ij}$ where n_{ij} is a random variable that takes the value 1 iff the jth element of S hashes to i.
  - Note that $En_{ij} = 1/m$. Why?
  - Now, $E\, n_i = \Sigma\, 1/m = n/m$.
- Note that the quantity n/m is often called as the load factor, denoted a.

Simple Uniform Hashing

- If each list is about n/m long, how much time does it take to search on average?
- Interestingly, the answer depends on a successful search vs. an unsuccessful search.
- The dependence comes from the fact that an unsuccessful search will anyway run through the entire list.
- For a successful search on the other hand, the number of elements traversed in the corresponding list differs as to when the item being searched was inserted.
- The answer in both cases is still $O(1+n/m)$.
- Read CLRS for the detailed proof for the successful case.

Universal Hashing

- Uniform hashing assumes good things happen based on the good nature of the input.
- If S is chosen once h is fixed, then one can always find a bad S where all elements hash to a single index.
  - Called as the adversarial choice of S.
- What if we choose h once S is given.
- Details follow.

Universal Hashing

- Consider a family of hash functions $H = \{ h_1, h_2, \ldots, h_r \}$.
- Each function from H maps keys in U to indices of T.
- The family H is called universal if
  - For every pair of distinct keys k and ℓ from U, the number of hash functions h from H such that $h(k) = h(\ell)$ is at most $|H|/m$.
    - What is m here?
- What does this definition really say?
  - Applies to all pairs of keys from U.
  - The probability that two distinct keys collide under a hash function h chosen uniformly at random from H is the same as the probability of choosing two indices of T uniformly at random.

Universal Hashing

- Three questions about H.

1) What is the benefit of such a family of functions?
2) Do such families exist?
3) How can we use such a family?

- We will take these questions in that order.

Universal Hashing

- Question 2: Do such families of hash functions exist?
- Yes, one such family is given below.
- Let p be a large prime. The universe is [0,p-1].
- Let $Z_p = \{0, 1, \ldots, p-1\}$ and $Z_p^* = \{1,2,\ldots, p-1\}$.
- Note that $p > m$ where $m = |T|$.
- For a in Z_p^* and b in Z_p, define

  $$h_{ab}(k) = ((ak+b) \bmod p) \bmod m.$$
- There are $p(p-1)$ functions in H.

Universal Hashing

- Let p be a large prime. The universe is [0,p-1].
- Let $Z_p = \{0, 1, \ldots, p-1\}$ and $Z_p^* = \{1,2,\ldots, p-1\}$.
- Note that $p > m$ where $m = |T|$.
- For a in Z_p^* and b in Z_p, define

  $h_{ab}(k) = (\ (ak+b) \bmod p\ ) \bmod m$.
- Example: Let p = 23, m = 7, a = 4, and b = 3.
- For k = 20, $h_{4,3}(20) = (\ (4\text{x}20+3) \bmod 23) \bmod 7 = (83 \bmod 23) \bmod 7 = 14 \bmod 7 = 0$.

Universal Hashing

- Theorem: The class H of functions defined earlier is a universal class of hash functions.
- Proof. We will show that for distinct k and ℓ from U, for a hash function from H chosen u.a.r., the probability that h(k) = h(ℓ) is at most 1/m.
- Let r = (ak + b) mod p and s = (aℓ + b) mod p.
- Can r equal s?
- Why?

Universal Hashing

- Theorem: The class H of functions defined earlier is a universal class of hash functions.
- Proof. We will show that for distinct k and ℓ from U, for a hash function from H chosen u.a.r., the probability that $h(k) = h(\ell)$ is at most $1/m$.
- Let $r = (ak + b) \bmod p$ and $s = (a\ell + b) \bmod p$.
- Can r equal s? NO.
- Since p is prime, there is a unique solution to the above set of equations modulo p.
  - Note $(r - s) = a(k - \ell) \pmod p$. And $k \neq \ell$, $a \neq 0$. So, $a(k - \ell) \neq 0 \bmod p$.

Universal Hashing

- Theorem: The class H of functions defined earlier is a universal class of hash functions.
- Proof. We will show that for distinct k and ℓ from U, for a hash function from H chosen u.a.r., the probability that $h(k) = h(\ell)$ is at most $1/m$.
- Let $r = (ak + b) \bmod p$ and $s = (a\ell + b) \bmod p$.
- Can r equal s? NO.
- So, $r \neq s$, and further for every pair of r, s among the $p(p - 1)$ possible pairs, it can be shown that (r,s) is mapped 1-1 to the pair (a,b).
  - In other words, we can pretend as if we picked a pair (r,s) uniformly at random.

Universal Hashing

- Theorem: The class H of functions defined earlier is a universal class of hash functions.
- Proof. Let $r = (ak + b) \bmod p$ and $s = (a\ell + b) \bmod p$.
- Can r equal s? NO.
- So, $r \neq s$, and further for every pair of r, s among the $p(p - 1)$ possible pairs, it can be shown that (r, s) is mapped 1-1 to the pair (a, b).
- Now, it is still possible that $h(k) = h(m)$ as $r \bmod m$ can equal $s \bmod m$.
- Given an r, the number of s such that $r \bmod m = s \bmod m$ can be counted as $r + m$, $r + 2m$, $r + 3m$, …
- How many such s exist?

Universal Hashing

- Theorem: The class H of functions defined earlier is a universal class of hash functions.
- Proof. Let $r = (ak + b) \bmod p$ and $s = (a\ell + b) \bmod p$.
- So, $r \neq s$.
- Now, it is still possible that $h(k) = h(m)$ as $r \bmod m$ can equal $s \bmod m$.
- Given an r, the number of s such that $r \bmod m = s \bmod m$ can be counted as $r + m$, $r + 2m$, $r + 3m$, …
- How many such s exist?
  - About p/m and $\lceil p/m \rceil - 1$ precisely.
  - Now, $\lceil p/m \rceil - 1 \leq (p+m-1)/m) - 1 = (p - 1)/m$.
- Since s is (as if) chosen u.a.r, the required probability is at most $((p - 1)/m)/(p - 1) = 1/m$.

Using Universal Hashing for Static Keys

- Consider setting where the set of keys, S, is specified at the beginning and does not change.
  - In other words, no further insert and delete operations.
- Examples include the set of keywords in a programming language, the set of files on a read-only device, and the like.
- For such applications, we can use hashing to check whether a given key is in T or not.
- Standard hashing based solution suggests that we should spend O(a) time for each search.
- Can we design an O(1) worst case solution?
  - Of course do not want to spend too much time finding the best hash function.

Using Universal Hashing for Static Keys

- Here is where universal hashing comes to our aid.
- Before we go there, a lemma.
- Let n keys be stored in a hash table using a hash function chosen u.a.r from a universal family of hash functions. The expected number of collisions is ${}^{n}C_{2}$ x 1/m.
- Proof. Use random variables. For every pair of keys, k and ℓ, define an indicator random variable $X_{k\ell}$ with value 1 iff k and ℓ collide under the chosen h.
- $EX_{k\ell}$ is at most 1/m.
- Define X to be a random variable whose value is the number of collisions.

Using Universal Hashing for Static Keys

- Let n keys be stored in a hash table using a hash function chosen u.a.r from a universal family of hash functions. The expected number of collisions is $^nC_2$ x 1/m.
- Proof. Use random variables. For every pair of keys, k and ℓ, define an indicator random variable $X_{k\ell}$ with value 1 iff k and ℓ collide under the chosen h.
- $EX_{k\ell}$ is at most 1/m.
- Define X to be a random variable whose value is the number of collisions.
- $EX = \Sigma_{k\neq\ell} EX_{k\ell} = {}^nC_2 \times 1/m$.

Using Universal Hashing for Static Keys

- Let n keys be stored in a hash table using a hash function chosen u.a.r from a universal family of hash functions. The expected number of collisions is $^nC_2$ x 1/m.
- Proof. Define X to be a random variable whose value is the number of collisions.
- $EX = \Sigma_{k \neq \ell} EX_{k\ell} = {}^nC_2 \times 1/m$.
- Typical values of m.
  - Usually $m = O(n)$. Then, $EX = \Theta(n)$.
  - If $m = n^2$, then, $EX = n(n – 1)/2n < ½$.
- What are some similarities to the above?

Using Universal Hashing for Static Keys

- But $m = n^2$ is an overkill in terms of space.
- Let us try another lemma.
- Let n keys be stored in a hash table using a hash function chosen u.a.r from a universal family of hash functions. Let n_i refer to the number of collisions at index i. Then,

  $$E[\ \Sigma_i\ n_i^2\ ] < 2n.$$

- Proof. Note that for any nonnegative integer x, $x^2 = x + 2\ {}^{x}C_2$.
- So, write the LHS as

Using Universal Hashing for Static Keys

- Let n keys be stored in a hash table using a hash function chosen u.a.r from a universal family of hash functions. Let n_i refer to the number of collisions at index i. Then, $E[\Sigma_i\, n_i^2\,] < 2n$.
- Proof. Note that for any integer $x > 0$, $x^2 = x + 2\ {}^{x}C_2$.
- So, write the LHS as

$$
\begin{aligned}
E[\Sigma_i\, n_i^2\,] &= E[\Sigma_i\ (n_i + 2\ {}^{n_i}C_2\,)\,] \\
&= E[\Sigma_i\ n_i\,] + 2\,E[\Sigma_i\ {}^{n_i}C_2\ ] \\
&= E[n] + 2\,E[\Sigma_i\ {}^{n_i}C_2\ ] \\
&= n + 2\,E[\Sigma_i\ {}^{n_i}C_2\ ]
\end{aligned}
$$

Using Universal Hashing for Static Keys

- Let n keys be stored in a hash table of size n using a hash function chosen u.a.r from a universal family of hash functions. Let n_i refer to the number of collisions at index i. Then, $E[\Sigma_i n_i^2] < 2n$.
- Proof. Note that for any integer $x > 0$, $x^2 = x + 2\ {}^{x}C_2$.
- So, write the LHS as

$$
\begin{aligned}
E[\Sigma_i n_i^2] &= E[\Sigma_i (n_i + 2\ {}^{n_i}C_2)] \\
&= E[\Sigma_i n_i] + 2\ E[\Sigma_i {}^{n_i}C_2] \\
&= E[n] + 2\ E[\Sigma_i {}^{n_i}C_2] \\
&= n + 2\ E[\Sigma_i {}^{n_i}C_2]
\end{aligned}
$$

- The remaining term in the RHS is just the expected number of collisions. Evaluated as ${}^{n}C_2$ X1/m at m = n.
  - Equals n(n – 1)/2n = (n-1)/2.

Using Universal Hashing for Static Keys

- Let n keys be stored in a hash table of size n using a hash function chosen u.a.r from a universal family of hash functions. Let n_i refer to the number of collisions at index i. Then, $E[\Sigma_i n_i^2] < 2n$.
- Proof. So, write the LHS as

$$E[\Sigma_i n_i^2] = E[\Sigma_i (n_i + 2\ {}^{n_i}C_2)]$$
$$= E[\Sigma_i n_i] + 2\,E[\Sigma_i {}^{n_i}C_2]$$
$$= E[n] + 2\,E[\Sigma_i {}^{n_i}C_2]$$
$$= n + 2\,E[\Sigma_i {}^{n_i}C_2]$$

- The remaining term in the RHS is just the expected number of collisions. Evaluated as at most ${}^{n}C_2 \times 1/m$ at m = n. Equals n(n – 1)/2n = (n-1)/2.
- The total is now n + 2(n-1)/2 = 2n – 1 < 2n.

Using Universal Hashing for Static Keys

- Let n keys be stored in a hash table of size n using a hash function chosen u.a.r from a universal family of hash functions. Let n_i refer to the number of collisions at index i. Then, $E[\sum_i n_i^2] < 2n$.
- The above suggests that while hashing n keys to a table of size n^2 is an overkill, each set of colliding keys can be rehashed to a bigger table individually.
- This is often called as two level hashing.
- At the first level, there is a hash table of size n for n keys.
- The hash function is chosen u.a.r from a universal family of hash functions.

Using Universal Hashing for Static Keys

- The above suggests that while hashing n keys to a table of size n^2 is an overkill, each set of colliding keys can be rehashed to a bigger table individually.
- This is often called as two level hashing.
- At the first level, there is a hash table of size n for n keys.
- The hash function is chosen u.a.r from a universal family of hash functions.
- At each table index i, if $n_i > 1$, then, we create a secondary table of size n_i^2, pick another hash function h_i u.a.r from a universal family, and rehash these n_i elements.

Using Universal Hashing for Static Keys

- An example follows.
- Take $p = 53$, $m = 11$, and $S = \{11, 19, 4, 62, 17, 28, 33, 51, 45\}$
- Take $a = 13$ and $b = 8$. $h_{13,8}(k) = ((13k + 8) \bmod 53) \bmod 11$.
- $H_{13,8}(11) = 8$, $h_{13,8}(19) = 10$, $h_{13,8}(4) = 7$, $h_{13,8}(62) = 8$, $h_{13,8}(17) = 6$, $h_{13,8}(28) = 1$, $h_{13,8}(33) = 2$, $h_{13,8}(51) = 6$, $h_{13,8}(45) = 10$.
- Now there are two groups of collisions, two elements collide at 8 and two collide at 6.

Using Universal Hashing for Static Keys

- An example follows.
- Take $p = 53$, $m = 11$, and $S = \{11, 19, 4, 62, 17, 28, 33, 51, 45\}$. Take $a = 13$ and $b = 8$. $h_{13,8}(k) = ((13K + 8) \bmod 53) \bmod 11$.
- $H_{13,8}(11) = 1$, $h_{13,8}(19) = 10$, $h_{13,8}(4) = 7$, $h_{13,8}(62) = 8$, $h_{13,8}(17) = 6$, $h_{13,8}(28) = 1$, $h_{13,8}(33) = 2$, $h_{13,8}(51) = 2$, $h_{13,8}(45) = 10$.
- Now there are three groups of collisions, two elements collide at 1, two collide at 10, and two collide at 2.
- For the elements 11 and 28, let $p=7$, $m=4$, and consider the hash function with $a = 5$ and $b = 2$. $h_{5,2}(11) = 1$, and $h_{5,2}(62) = 2$. No collisions!

Using Universal Hashing for Static Keys

- An example follows.
- Take $p = 53$, $m = 11$, and $S = \{11, 19, 4, 62, 17, 28, 33, 51, 45\}$. Take $a = 13$ and $b = 8$. $h_{13,8}(k) = ((13K + 8) \bmod 53) \bmod 11$.
- $H_{13,8}(11) = 8$, $h_{13,8}(19) = 10$, $h_{13,8}(4) = 7$, $h_{13,8}(62) = 8$, $h_{13,8}(17) = 6$, $h_{13,8}(28) = 1$, $h_{13,8}(33) = 2$, $h_{13,8}(51) = 6$, $h_{13,8}(45) = 10$.
- For the elements 11 and 62, consider the hash function with $a = 15$ and $b = 22$. Note however that $m = 4$. $h_{15,22}(11) = 0$, and $h_{15,22}(62) = 3$. No collisions!
- For the elements 17 and 51, consider the hash function with $m = 4$, $a = 32$ and $b = 12$. $h_{32,12}(17) = 0$, and $h_{32,12}(62) = 2$. No collisions!
- For the other two pairs, find similar hash functions.

Using Universal Hashing for Static Keys

- What is the total space used?
  - On expectation O(n) since m = n and $E[\sum_i n_i^2] < 2n$.
  - Plus, space to store the hash functions itself.
- Time per query?
  - O(1)
- What if there are collisions at some secondary table?
  - The expected number of collisions is ½.
  - So, using Markov inequality, the probability that there is at least one collision for a hash function chosen u.a.r is at most ½.
  - Try a couple of times to get a ‘’good’’ hash function.