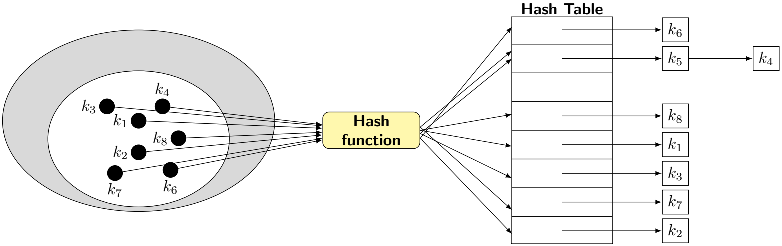
Hashing by Chaining



Implemented as an array of pointers to a linked list.

Hashing by Chaining

- For inserting an element perform following steps:
  1. Compute the hash value of the element
  2. Access the pointer in the array indexed by hash value, prepend to the list.
- Collisions resolved by chaining the elements in a linked list.
- For a deletion/search perform following steps:
  1. Obtain the hash value
  2. Access the corresponding chain to find the value.

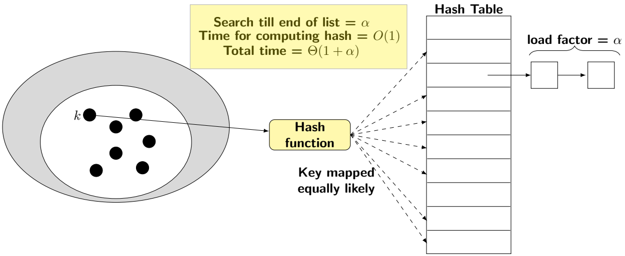
Hashing by Chaining

- Simple uniform hash function means that each key is equally likely to be hashed into any slot.
- Let $P(k)$ be the probability that k is represented in the table.
- Distributiveness means each slot $j = 0, 1, \ldots, m-1$ equally likely to be occupied:

$$\sum_{k|h(k)=j} P(k) = \frac{1}{m}.$$

- The expected length of any chain = $\frac{n}{m}$ which is called load factor and denoted by α.

Hashing by Chaining



- For an unsuccessful search, the number links traversed is α excluding the NULL.
- For successful search it is: $1 + \alpha/2$.
  - One link has to be traversed any way.
  - In an average half the links will be traversed.

Pseudo Code Initialization

```
typedef struct hTnode {
      int val;
      struct node * next;
} node;

// Initialization of pointer array for hash table
void initializeHT(node * hashTable[], int m) {
      int i;
      for (i=0; i < m; i++)
         hashTable[i] = NULL;

}
```

Pseudo Code for Search

```
node *searchKey (node *hashTable[], int k) {
    node *p;
    p = hashTable[h(k)];
    while ((p != NULL) && (p->val != k))
        p = p->next;
    if (p->val == k)
        return p;
    else
        return NULL;
}
```

Pseudo Code for Insert

```
void insertKey(node * hashTable[], int k) {
    node * newNode;
    node *ptr = searchKey(hashTable,k);
    if (ptr == NULL) {
        newNode = (node *) malloc(sizeof(node));
        newNode->val = k;
        newNode->next = hashTable[h(k)]
        hashTable[h(k)] = newNode;
    }
}
```

Pseudo Code for Delete

```
void deleteKey (node *hashTable[], int k) {
    node *save, *p;
    save = NULL;
    p = hashTable[h(k)];
    while ( p!=NULL ) {
        save = p;
        p = p->next;
    }
    if (p != NULL) {
        save->next = p->next;
        free(p);
    } else
        print("value %d not found\n", k);
}
```

Universal Hash Function

- Universal hashing defines a family of hash functions $\mathcal{H}$.
- A randomly chosen hash is picked from $\mathcal{H}$ to mapp the keys.
- The idea is that a good hashing scheme may emerge through a competition among the rival developers.
  - Apart from hashing programs being tested against a bench mark suite, they can also be tested by the rivals.
  - The rivals would create test cases to defeat each other's hashing schemes.
- Hashing scheme is called universal, as it will work against any adversary with the promised expectation.

Universal Hash Function

- The only way one can win is to prevent an adversary from gaining an insight by using randomization.
- So, choose one at random out of several hash functions.
- An adversary can examine your code, but does not exactly know which hash will be used.
- It guarantees that for any two distinct keys x, and y the probability of collision is: $1/m$, where m is the table size.

Universal Hash Function

Definition

Let U be a universe of keys, and let $\mathcal{H}$ be a finite collection of hash functions mapping U to $\{0, 1, \ldots, m-1\}$.

Definition

$\mathcal{H}$ is universal, if for all $x \neq y$, $|\{h \in \mathcal{H} : h(x) = h(y)\}| = \frac{|\mathcal{H}|}{m}$.

From definiton 2, if h chosen randomly from $\mathcal{H}$ we have:

$$\frac{\text{\# functions mapping } x \text{ and } y \text{ to same location}}{\text{Total \# of functions}} = \frac{\frac{|\mathcal{H}|}{m}}{|\mathcal{H}|} \leq \frac{1}{m}$$

Universal Hash Function

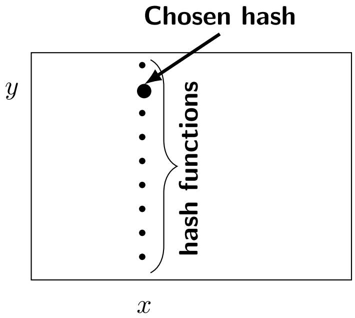
Theorem

Suppose n keys to be hashed into a table of size m, then choose a hash function h randomly from the set $\mathcal{H}$, Under the stated conditions, the expected number of collisions with any key x is given by:

$$E(\text{\# of collision with } x) = \frac{n}{m}$$

$\frac{n}{m} = \alpha$ is known as load factor.

Universal Hash Function



- The theorem essentially implies that if a set of universal hash function exists then choosing a hash function from this set ensures that keys are evenly distributed.

Universal Hash Function

Proof.

Let C_x be the random variable denoting the number of keys in table T colliding with x. Define

$$c_{xy} = \begin{cases} 1, \text{ if } h(x) = h(y) \\ 0, \text{ otherwise} \end{cases}$$

Then $E(c_{xy}) = 1/m$ and $C_x = \sum_{y \in T - \{x\}} c_{xy}$ □

Universal Hash Function

Proof (contd).

Now derive $E(C_x)$:

$$\begin{aligned}
E(C_x) &= E\left(\sum_{y \in T-\{x\}} c_{xy}\right) \\
&= \sum_{y \in T-\{x\}} E(c_{xy}), \text{ by linearity of expectations} \\
&\leq \sum_{y \in T-\{x\}} \frac{1}{m} = \frac{n-1}{m}
\end{aligned}$$

$\square$

Universal Hash Function

Proof (contd).

- In the above expression we only considered the cases when x and y are distinct.
- Since x collides with itself 1 more probe will necessary for x to account for all keys that collide with x.
- So, the expected number of probes will be $\leq 1 + \alpha$.

$\square$

Constructing a Universal Hash Function

- Works when m is prime.
- Every key is decomposed into $r+1$ digits of base m, where $0 \leq k_i \leq m-1$ (where m is table size).
- For example, let size m = 11, and key=46793.
- key is represented as vector: $\langle 4, 6, 7, 9, 3 \rangle$ and its value is $3 * 11^0 + 9 * 11^2 + 7 * 11^3 + 6 * 11^4 + 4 * 11^5$.
- Then pick a random vector $a = \langle a_0, a_1, \ldots, a_r \rangle$, where $0 \leq a_i \leq m-1$.
  - Picking vector a actually means picking of a random hash function. In other words, a serves as an index for picking a random hash functions.
- Compute $h_a(k) = \left(\sum_{0 \leq i \leq r} a_i k_i \right) \mod m$

Size of Set of Hash Functions

- How many vectors of length $r + 1$ can be there, where each value can be a m base digit?
  - It will be m^{r+1}.
- So there are m^{r+1} hash functions or possible choices for vectors $\langle a_0, a_1, \ldots, a_r \rangle$.
- Now we have to prove that these hash functions form a universal set of hash functions.

Finite Fields: A Digression from Hashing

- Consider a result from finite field before actual proof.
- For any prime m, the set of integers

$$\mathcal{Z}_m = \{0, 1, \ldots, m-1\}$$

  with modulo m operations (+, *) defines a field.
- In a field every nonzero element has a unique multiplicative inverse.

Finite Field

For example consider $m = 7$, the elements of field are {0, 1, 2, 3, 4, 5, 6}.

z	1	2	3	4	5	6
z^{-1}	1	4	5	2	3	6

- Note that m has to be prime to become a field with modulo operation.
- Let us take $m = 10$, then elements of field: $\{1, 2, \ldots 9\}$.
- Clearly, 2 does not have any inverse in $\mathcal{Z}_{10}$.

Universal Hashing

Theorem

The construction of family of hash functions as specified by random choice of $\langle a_0, a_1, \ldots, a_r \rangle$ is universal.

Proof.

- We need to show that for any two distinct keys x and y, $Pr[h_a(x) = h_a(y)] \leq \frac{1}{m}$
- Given that x are y distinct, decompose each as a $(r+1)$-digit base m integer.
- We should have $x_i \neq y_i$ at least at one position $0 \leq i \leq r$.
- WLOG assume that $x_0 \neq y_0$.
- If they differ in another position arguments remain same.

$\square$

Proof for Construction of Universal Hashing

Proof (contd).

$$h_a(x) = \sum_0^r a_i x_i, \text{ and } h_a(y) = \sum_0^r a_i y_i$$

Therefore,

$$\sum_0^r a_i(x_i - y_i) \equiv 0 \pmod{m}$$

$$a_0(x_0 - y_0) + \sum_1^r a_i(x_i - y_i) \equiv 0 \pmod{m}$$

$$a_0(x_0 - y_0) \equiv -\sum_1^r a_i(x_i - y_i) \pmod{m}$$

□

Proof for Construction of Universal Hashing

Proof (contd).

- Since, $x_0 \neq y_0$, $\exists$, $(x_0 - y_0)^{-1}$ in $\mathcal{Z}_m$.
- Now multiply both side of above modulo expression by the inverse $(x_0 - y_0)^{-1}$.
- We get

$$a_0 \equiv \left(-\sum_{1}^{r} a_i(x_i - y_i) \right) (x_0 - y_0)^{-1}$$

- Which implies a_0 is a fixed value computed from a function of other a_i values.

□

Proof for Construction of Universal Hashing

Proof (contd).

- So, once a set of a_i's , for $i > 0$, has been fixed, only one value of a_0 is possible.
- The number of possible choices of a_i's can be m^r which produces m^r different values of a_0's.
- So, the possibility of a clash in $h_a(x)$ and $h_a(y)$ is:

$$\frac{m^r}{m^{r+1}} = \frac{1}{m}.$$

- Therefore, construction as suggested is universal.

$\square$