SFU CMPT-307 2008-2 Lecture: Week 8

Ján Maňuch

E-mail: jmanuch@sfu.ca

Lecture on June 24, 2008, 5.30pm-8.20pm

Universal hashing

Overcomes problem of bad inputs by using **randomization**

Mentioned: if **adversary** chooses keys to be hashed by **fixed** hash function, he can choose n keys that **all** hash into same slot

All fixed hash functions have this problem

Universal hashing:

- design **class** of parametrized hash functions $\mathcal{H}$
- pick **random** $h \in \mathcal{H}$ at beginning of execution (select at random parameters)

Let $\mathcal{H}$ be a finite collection of hash functions that map given universe U into $\{0, 1, \ldots, m-1\}$

$\mathcal{H}$ is said to be **universal** if for each pair $k, \ell \in U$ of keys, the number of hash functions $h \in \mathcal{H}$ with $h(k) = h(\ell)$ is at most $|\mathcal{H}|/m$, i.e., the chance of a collision is at most

$$\frac{|\mathcal{H}|/m}{|\mathcal{H}|} = \frac{1}{m}$$

In other words: with h randomly chosen from $\mathcal{H}$, the chance of collision between arbitrary distinct keys k and ℓ is just the same $(1/m)$ as a chance of collision if $h(k)$ and $h(\ell)$ were randomly and independently chosen from $\{0, \ldots, m-1\}$

Properties: average-case

Let $\mathcal{H}$ be a universal collection of hash functions

Theorem. Choose $h \in \mathcal{H}$ randomly to hash n keys (set K) into table T of size m, using chaining to resolve collisions.

1. If key k is not in the table, then expected length $E[n_{h(k)}]$ of list that k hashes to is at most α.
2. If key k is in the table, then expected length $E[n_{h(k)}]$ of list containing k is at most $1 + \alpha$.

Proof. *Important:* expectations over the choice of a hash function, do not depend on assumptions regarding distribution of keys!

For each pair of distinct keys k, ℓ define a random variable $X_{k\ell}$ with $X_{k\ell} = 1$ iff $h(k) = h(\ell)$, i.e, collision

By definition, $P(X_{k\ell} = 1) \leq 1/m$, and thus $E[X_{k\ell}] \leq 1/m$

Now define for each key k a random variable

$$Y_k = \sum_{\substack{\ell \in K \\ \ell \neq k}} X_{k\ell},$$

i.e. the number of keys other than k that hash to same slot as k. Clearly,

$$E[Y_k] = E\left[\sum_{\substack{\ell \in K \\ \ell \neq k}} X_{k\ell}\right] = \sum_{\substack{\ell \in K \\ \ell \neq k}} E[X_{kl}] \leq \sum_{\substack{\ell \in K \\ \ell \neq k}} \frac{1}{m} = |\{\ell : \ \ell \in K \wedge \ell \neq k\}|/m$$

It remains to count the number of $\ell \in K$ with $\ell \neq k$

Case 1: If $k \notin K$, then $n_{h(k)} = Y_k$ and

$$|\{\ell : \ \ell \in K \wedge \ell \neq k\}| \ = \ n$$

Thus $E[n_{h(k)}] = E[Y_k] \leq n/m = \alpha$

Case 2: If $k \in K$, key k appears in $T[h(k)]$. Also, count Y_k does not include k itself. Thus $n_{h(k)} = Y_k + 1$ and

$$|\{\ell : \ \ell \in K \wedge \ell \neq k\}| \ = \ n - 1$$

Thus

$E[n_{h(k)}] = E[Y_k] + 1 \leq (n-1)/m + 1 = 1 + \alpha - 1/m < 1 + \alpha$ □

Universal hashing: sequence of operations

Corollary. Using the universal hashing and chaining in a table with m slots, it takes expected time $\Theta(k)$ to handle any sequence of k Insert, Search, and Delete operations, containing $O(m)$ Insert operations.

Proof.

- the number of Insertions is $O(m)$
- thus the number of elements in the hash table at any time is $n = O(m)$ and $\alpha = O(1)$.
- Insert and Delete take constant time, Search takes expected constant time (due to $\alpha = O(1)$),
- by linearity of expectation, expected time for sequence of k operations is $O(k)$.

Notice: Impossible for adversary to pick sequence of operations that forces the worst-case time

This is now **without** the assumption of *simple uniform hashing*

Construction of a universal class of hash functions

Simple solution. take all functions mapping U into $\{0, 1, \ldots, m-1\}$

disadvantages:

size of the collection: $m^{|U|}$

number of random bits needed to pick a random function: $|U| \log m$ — a lot of preprocessing needed

how to encode functions?

we have to store the chosen function — space needed: $|U| \log m$

we need a much smaller collection of hash functions

Assignment Problem 8.1. (deadline: July 8, 5:30pm)
Prove that the class of all functions from U to $\{0, 1, \ldots, m-1\}$ is universal.

Number-theoretical solution.

Let p be a prime number and let

$$\begin{aligned} \mathbb{Z}_p &= \{0, 1, \ldots, p-1\} \\ \mathbb{Z}_p^* &= \{1, 2, \ldots, p-1\} \end{aligned}$$

Lemma. For any $a \in \mathbb{Z}_p^*$, $b, c \in \mathbb{Z}_p$, the equation

$$ax + b \equiv c \mod p$$

has a unique solution $x \in \mathbb{Z}_p$.

(see Chapter 31.4 for a proof)

Let p be a prime, large enough such that every possible key k is in $0, \ldots, p-1$, inclusive.

for all $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$ define hash functions $h_{a,b}$:

$$\boxed{h_{a,b}(k) = ((ak + b) \bmod p) \bmod m}$$

(linear transformation followed by reductions modulo p and then modulo m)

Note: $p > m$ since by the assumption, the size of universe of keys U is greater than the number of slots
the function $h_{a,b}$ maps $U \subseteq \mathbb{Z}_p$ into $\mathbb{Z}_m$

Consider the universal class of hash functions:

$$\boxed{\mathcal{H}_{p,m} = \{h_{a,b} : \ a \in \mathbb{Z}_p^*, \ b \in \mathbb{Z}_p\}}$$

Note: size of class: $p(p-1)$
p can be chosen such that $p < 2|U|$ (Bertrand's postulate)
picking randomly from $\mathcal{H}_{p,m}$ means picking random a, b
number of bits needed to pick a function from $\mathcal{H}_{p,m}$: $\mathcal{O}(\log |U|)$

Theorem. Class $\mathcal{H}_{p,m}$ of hash functions is universal.

Proof. Take two distinct keys k and ℓ.

How many functions are there in $\mathcal{H}_{p,m}$ mapping k and ℓ to the same slot?

Consider $h_{a,b} \in \mathcal{H}_{p,m}$. Let

$$r = (ak + b) \bmod p$$
$$s = (a\ell + b) \bmod p$$

Is it possible that $r = s$?

By **Lemma**: "No".
If $r = s$ then k and ℓ are solutions to equation

$$ax + b \equiv r \mod p$$

and by uniqueness of solution, $k = \ell$.

"no collisions at modulo p level"

There are $p(p-1)$ functions in $\mathcal{H}_{p,m}$
and there are $p(p-1)$ possible pairs (r, s) such that $r \neq s$. Is there a one-to-one correspondence?

"Yes" — Values r and s determine the function $h_{a,b}$ (remember k and ℓ are still fixed):

$$r - s \equiv a(k - \ell) + 0 \mod p$$

by **Lemma** (unknown a), a is unique

$$r \equiv 1.b + ak \mod p$$

by **Lemma** (unknown b), b is unique

implies $h_{a,b}$ uniquely determined by r and s

Conclusion: If we pick $h_{a,b} \in \mathcal{H}_{p,m}$ uniformly at random, the resulting pair (r, s) is equally likely to be any pair of distinct values from $\mathbb{Z}_p$.

When a **collision** between k and ℓ happens?

k and ℓ collide when $r \equiv s \mod m$

Hence, probability that k and ℓ collide is equal to probability that $r \equiv s \mod m$ when r and s are randomly chosen as distinct values from $\mathbb{Z}_p$.

Note: doesn't depend on k and ℓ anymore

pick any r, there are at most $\lceil p/m \rceil$ choices of s such that $r \equiv s \mod m$, one of them is $s = r$

hence, there are at most $\lceil p/m \rceil - 1$ choices for s such that $s \neq r$ and $r \equiv s \mod m$

hence, probability that $r \equiv s \mod m$ and $r \neq s$ is at most

$$\frac{\lceil p/m \rceil - 1}{p-1} \leq \frac{(p+m-1)/m - 1}{p-1}$$
$$= \frac{(p-1)/m}{p-1} = 1/m$$

Final conclusion: $\mathcal{H}_{p,m}$ is universal.

Open addressing

the second method for dealing with collisions — recall, the first method was "chaining"

- elements are stored directly in hash table (not in linked lists)
- hence: the **load** α is always at most 1 (but the table can overflow)
- for every key we have a sequence of slots, called the **probe sequence**
  - when inserting we try (**probe**) the first slot, if used, the second slot, etc., until we find an empty slot
  - when searching we go over slots in sequence until we either find the searched element or empty slot
- no pointers used — we can use more memory for the hash table instead, which results in fewer collisions, and so, faster searching

Hash functions have now two parameters:

- the key $k \in U$
- the probe number $i \in \{0, 1, \ldots, m-1\}$

Furthermore, the hash function

$$h : U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$$

must satisfy that for every key k, the **probe sequence**

$$\langle h(k, 0), h(k, 1), \ldots, h(k, m-1) \rangle$$

is a permutation of $\langle 0, 1, \ldots, m-1 \rangle$

Now: every position is eventually considered as a slot for a new key (the algorithm fails with "overflow" only when there is no place in the entire table to insert a key)

Assume that an empty slot contains the value NIL

Hash-Insert(T, k)

```
1:  i ← 0
2:  repeat
3:      j ← h(k, i)
4:      if T[j] = NIL then
5:          T[j] ← k
6:          return j
7:      else
8:          i ← i + 1
9:      end if
10: until i = m
11: error "hash table overflow"
```

Idea: probing the probe sequence of k until we find an empty slot

Hash-Search(T, k) probes the same sequence, therefore the search can be terminated when an empty slot is found

assuming that we didn't delete anything from the table

Hash-Search(T, k)

```
1:  i ← 0
2:  repeat
3:      j ← h(k, i)
4:      if T[j] = k then
5:          return j
6:      else
7:          i ← i + 1
8:      end if
9:  until T[j] = NIL or i = m
10: return NIL
```

returns j where slot j contains the key k, or NIL, if k is not in the table

Hash-Delete(T, k) — difficult operation:

- when we delete a key from the table, we cannot just mark the slot containing the key by NIL (empty)
  [we could disconnect a sequence of occupied elements leading to the key]

Solution. mark the place with the special value DELETED (instead of NIL)

- modify **Hash-Insert**(T, k) so that it treats slot containing DELETED as empty
- **Hash-Search**(T, k) will treat such a slot as occupied, which we want, so no modification is needed

Problem. the search time doesn't depend on load $\alpha = \frac{\#\text{elements}}{\#\text{slots}}$, but rather on $\frac{\#\text{elements}+\#\text{DELETED}}{\#\text{slots}}$, which can be quite bad

Conclusion. when **Delete** operation is allowed, prefer to use hashing by **chaining**

Assignment Problem 8.2. (deadline: July 8, 5:30pm)
Write a pseudo-code for the modified **Hash-Insert** able to handle also the special value DELETED, and for the procedure **Hash-Delete**.

Uniform hashing

– generalization of *simple uniform hashing*
– very strong conditions, in practice just some approximations are used
– useful for analysis

assumption:

each key is equally likely to have any of $m!$ permutations of $\langle 0, 1, \ldots, m-1 \rangle$ as its probe sequence

requires that all $m!$ possible permutations (probe sequence) could be generated

Common techniques used are much worse:

- linear probing (only m distinct probe sequences)
- quadratic probing (only m distinct probe sequences)
- double hashing (m^2 distinct probe sequences)
  the best results

Linear probing

needs an ordinary hash function $h' : U \to \{0, 1, \ldots, m-1\}$, called an **auxiliary hash function**

hash function:

$$h(k, i) = (h'(k) + i) \bmod m$$

hence, for a key k, the probe sequence is

$$\langle h'(k), h'(k)+1, \ldots, n-1, 0, \ldots, h'(k)-1 \rangle$$

– we start at the slot given by auxiliary hash function, and then probe consecutive slots

– first probe determines the entire sequence $\implies$ only m distinct probe sequences

Problem. **primary clustering**

– long clusters (sequences) of occupied slots – increasing average search time

– why?

- consider a cluster with i occupied slots, if the consecutive slot is empty, then it has a probability $(i+1)/m$ that it will be filled during next **Insert** operation

Quadratic probing

hash function:

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

where h' is again an auxiliary hash function, $c_2 \neq 0$

Problems:

- doesn't work for all values of c_1, c_2 and m
- still: a probe sequence for one key is shifted version of the probe sequence for another key $\Longrightarrow$ only m distinct probe sequences
- **secondary clustering** – milder form of clustering (the keys mapped to the same slot will form a cluster)

Assignment Problem 8.3. (deadline: July 8, 5:30pm)
Consider a quadratic probing scheme with constants $m = 2^t$ for some integer t (i.e., m is a power of 2) and $c_1 = c_2 = 1/2$. Prove that the function

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

is indeed a hash function, i.e., prove that all probe sequences are permutations of $\langle 0, 1, \ldots, m - 1 \rangle$.

Hint: Congruence $a/2 \equiv b \mod 2^t$ makes sense only if a is even, and is equivalent to congruence $a \equiv 2b \mod 2^{t+1}$.

Double hashing

hash function:

$$h(k,i) = (h_1(k) + ih_2(k)) \bmod m$$

where h_1 and h_2 are auxiliary hash functions

– first probe doesn't determine the entire sequence, depends also on the value of the second hash function (assuming we don't use $h_2 = h_1$)
– approximates behavior of "uniform hashing", generated permutations have some characteristics of randomly chosen permutation

Requires. The values of $h_2(k)$ have to be relatively prime (coprime) to the size of hash table m

two easy ways:

1. m is power of 2, and h_2 produce only odd numbers
2. m is a prime, and h_2 returns positive values smaller than m

each pair $(h_1(k), h_2(k))$ yields a distinct probe sequence, hence $\Theta(m^2)$ probing sequences are used

Analysis of open-address hashing

- assume that we use "uniform hashing": *a new key has a probe sequence chosen uniformly at random from all possible permutations*
- when we are looking for a key which is already in the table, we assume that every key is equally likely to be searched for
- we will express the expected number of probes in terms of the **load** $\alpha = n/m \leq 1$

Unsuccessful search

Theorem. Given an open-address hash table with load $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$, assuming uniform hashing.

Example.

if $\alpha = 1/2$ (50% of table occupied) then the expected number of probes for unsuccessful search will be $1/(1-1/2) = 2$

if $\alpha = 0.9$ (90% of table occupied), then $1/(1-9/10) = 10$

Proof.

- let X be a random variable denoting the number of probes made in an unsuccessful search
- for $i = 1, \ldots, m$, let A_i be the event that the i-th probe $h(k, i-1)$ is to an occupied slot
- $\{X = i\}$ iff events $A_1, \ldots, A_{i-1}$ happened and event A_i didn't
- event $\{X \geq i\}$ is equivalent to event $A_1 \cap A_2 \cap \cdots \cap A_{i-1}$, hence

$$P(X \geq i) = P(A_1 \cap A_2 \cap \cdots \cap A_{i-1})$$

we are looking for $E[X]$
why do we need $P(X \geq i)$?

answer: Assignment Problem 8.4

Assignment Problem 8.4. (deadline: July 8, 5:30pm)
Consider a random variable X having only positive integer values (i.e., it's mapping the probability space to the set of natural numbers). Prove that

$$E[X] = P(X \geq 1) + P(X \geq 2) + \cdots = \sum_{i=1}^{\infty} P(X \geq i)$$

Hint: events $\{X = 1\}, \{X = 2\}, \{X = 3\}, \ldots,$ are mutually exclusive (disjoint), hence

$$P(X \geq i) = \sum_{j=i}^{\infty} P(X = j)$$

We will use the formula from Assignment Problem 5.2 again:

$$P(A_1 \cap A_2 \cap \cdots \cap A_{i-1}) = \\ P(A_1) \cdot P(A_2|A_1) \cdots P(A_{i-1}|A_{i-2} \cap \cdots \cap A_2 \cap A_1)$$

Hence, to find out $P(X \geq i)$ it's enough to evaluate conditional probabilities $P(A_j|A_{j-1} \cap \cdots \cap A_1)$.

case $j = 1$: $P(A_1) = n/m$

(m slots, n are occupied, uniform hashing, the first probe can be to any of m slots)

case $j > 1$:

– assume $A_1, \ldots, A_{j-1}$ occurred

– hence, first $j-1$ probes were to occupied slots

– j-th probe cannot be to any of these slots (the probe sequence is a permutation)

– hence, we have $m-(j-1)$ slots to choose from, and $n-(j-1)$ of them are occupied

conclusion: probability $P(A_j|A_{j-1} \cap \cdots \cap A_1)$ is

$$(n-j+1)/(m-j+1)$$

We can now calculate $P(X \geq i)$:

$$\begin{aligned} P(X \geq i) &= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdots \frac{n-i+2}{m-i+2} \\ &\leq \left(\frac{n}{m}\right)^{i-1} \\ &= \alpha^{i-1} \end{aligned}$$

Using Assignment Problem 8.4,

$$\begin{aligned} E[X] &= \sum_{i=1}^{\infty} P(X \geq i) \leq \sum_{i=1}^{\infty} \alpha^{i-1} \\ &= \sum_{i=0}^{\infty} \alpha^{i} = \frac{1}{1-\alpha} \end{aligned}$$

- if α is constant, then $1/(1-\alpha)$ is also constant, and so an unsuccessful search runs in time $\mathcal{O}(1)$

Insert

Corollary. Inserting an element into an open-address hash table with load α requires at most $1/(1-\alpha)$ probes on average, assuming uniform hashing.

Proof.

- there must be space in the table, $\alpha < 1$
- inserting a key is equivalent to unsuccessful search, where we insert the key to the final (empty) slot of unsuccessful search
- the same expected number of probes: $1/(1-\alpha)$

Successful search

Theorem. Given an open-address hash table with load $\alpha < 1$, the expected number of probes of successful search is at most

$$\frac{1}{\alpha} \cdot \left(\ln \frac{1}{1-\alpha} + \mathcal{O}(1) \right)$$

assuming uniform hashing and assuming that each key in the table is equally likely to be searched for.

Example.

50% table full: expected number of probes in successful search: at most 1.387

90% table full: 2.559

Proof.

- when looking for a key k, we have to repeat the same probe sequence as when inserting k into the table
- assume that keys were inserted to the table in the order $k_1, k_2, \ldots, k_n$
- hence, if we are looking for the key k_i, the number of probe is the same as the number of probes we needed when inserting k_i to the table
- at that time, table contained $i-1$ elements, and the load factor was $\alpha = (i-1)/m$
- search for k_i requires at most

$$\frac{1}{1-\alpha} = \frac{1}{1-(i-1)/m} = \frac{m}{m-i+1}$$

probes on average

as it can be any of n keys in the table with the same probability, we have to take the average of those expected numbers of probes:

$$
\begin{aligned}
E[\text{\#probes}] &\leq \frac{1}{n}\sum_{i=1}^{n}\frac{m}{m-i+1} \\
&= \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} \\
&= \frac{1}{\alpha}\cdot(H_m - H_{m-n}) \\
&\quad \text{using } H_k = \ln k + \mathcal{O}(1) \\
&= \frac{1}{\alpha}\cdot(\ln m - \ln(m-n) + \mathcal{O}(1)) \\
&= \frac{1}{\alpha}\cdot\left(\ln\frac{m}{m-n} + \mathcal{O}(1)\right) \\
&= \frac{1}{\alpha}\cdot\left(\ln\frac{1}{1-\alpha} + \mathcal{O}(1)\right)
\end{aligned}
$$