COMP 480/580 — Probabilistic Algorithms and Data Structure Sept 1 2022

Lecture 4: Analysis of Hashing, Chaining and Probing

Lecturer: Anshumali Shrivastava *Scribe By: Fangfei Yang*

1 Universal hashing family

Definition 1 *k-universal hashing family H for a set $x_1, x_2, \ldots, x_k$, and h from H, $h(x_1), h(x_2), \ldots, h(x_k)$ are independent random variables. Or, $Pr\left(h(x_1) = h(x_1) = \cdots = h(x_k)\right) \leq \frac{1}{n^{k-1}}$*

$ax + b \ mod \ P \ \rightarrow \ P\left(h(x) = h(y)\right) \ \leq \ 1/N$ gives you 2-universal family, but for k-universal, we need polynomial in k. That is to say, $ax + b \ mod \ P \ \rightarrow \ P(h(x) = h(y) = h(z)) \not\leq 1/N^2$, while $a_1x^2 + a_2xb \ mod \ P \rightarrow P(h(x) = h(y) = h(z)) \leq 1/N^2 = P(h(x) = h(y)) \times P(h(y) = h(z))$.

And, $a_kx^k + a_{k-1}x^{k-1} + \cdots + a_1x + a_0 \ mod \ P \ mod \ R$ is k-universal.

Thus, it's harder to achieve higher independence as the cost for computation and memory increased.

2 Hash table collision

2.1 Chaining

Put m objects into array of n. The expected length of chain $\leq 1 + m - 1/m$.

Definition 2 *load factor α*

$$\alpha = m/n. \tag{1}$$

Search time is $1 + \alpha$, in worst case it's m.

Good case? $\log N$.

What is the probability of existing a chain of $size \geq \log N$.

Theorem 1 *For the special case with $m = n$, with probability at least $1 - 1/n$, the longest list if $O(\ln n/\ln\ln n)$.*

Proof Let $X_{i,k} = indicator of key$ hash to slot k, $Pr(X_{i,k} = 1) = 1/n$

We can calculate the probability that a particular slot k receives $> K$, assuming independent.

$$\binom{n}{K} 1/m^k = \binom{n}{K} 1/n^k < 1/k! \tag{2}$$

If we choose $K = 3 * \ln n/\ln\ln n$, then $K! > n^2$ and $1/k! < 1/n^2$. Thus the probability that any n slots receives $> K$ keys is $< 1/n$.

Definition 3 *Power of two (multiple) choices The bad event happens with probability with p, happens in both worlds (assume independent), is rare: $p^2, p^3, \ldots, p^n$. If we define the good event is not all world has the bad event.*

Use two hash functions, insert at the location with smaller chain.

Assignment Using $m = n$ slots, with probability ar least $1 - 1/n$, the longest list if $O(\log\log n)$.

Do independent things in parallel pick the best.

2.2 Linear Probing

Probing sequence

- 0^{th} **probe** $= h(k)\ mod\ TableSize$
- 0^{th} **probe** $= h(k) + 1\ mod\ TableSize$
- 0^{th} **probe** $= h(k) + 2\ mod\ TableSize$
- ...

History

- 1954 linear probing introduced as subroutine for an assembler
- 1962 - $n - independence$ the probing steps is constant
- 2005 - $5 - independence$ the probing steps is constant
- 2007 - $2 - independence$ the probing steps is constant

In practices, linear probingis one of the fastest general-purpose hashing strategies available.

Reasons

- Low memory overhead: array and a hash function
- Excellent locality: when collisions occur, we only search in adjacent location
- Great cache performance: a combination of the above of two

Analyze Analyzing linear probingis hard because insertion in any location is going to effect other insertion with different hash result while chaining only rely on its own location k.

Assume a load factor $\alpha = \frac{m}{n} = 1/3$.

- What happens to linear probing of $\alpha \geq 1$.
- Contrast with chaining

Definition 4 *Region a region R of size m is consecutive set of m locations in the hash table.*

An element q hashes into region R if $h(q) \in R$, though q may not be placed in R.

On expectation, a region of size 2^s has at most $1/3 * 2^s$ elements hash to it.

It would be very unlikely if a region has twice as many as elements in it as expected. A region of size 2^s is overloaded if at least $2/3 * 2^s$ elements hash to it.

Theorem 2 *The probability that the query element q ends up between 2^S and 2^{S+1} steps from its home location is upper-bounded by $c \cdot Pr[\text{the region of size } 2^s \text{centered on } h(q) \text{ is overloaded}]$ for some fixed constant c independent of S.*

Donating the $Pr[\text{the region of size } 2^s\text{centered on } h(q) \text{ is overloaded}]$ as $Pr[R_{2^S} > 2/3 * 2^S]$, where X_{2^S} is the random variable for the number of elements in any R^S region.

Applying Markov's inequality, we get:

$$Pr\left[R_{2^S} > 2/3 * 2^S\right] \tag{3}$$
$$\leq \frac{E\left[R_{2^S}\right]}{2/3 * 2^S} \tag{4}$$
$$= \frac{2^S * \alpha}{2/3 * 2^S} \tag{5}$$
$$= 1/2 \tag{6}$$

This gives us a bound for $E[step] \leq \sum_S^{log(n)} 2^S * c * Pr[R_{2^S} > 2/3 * 2^S] = O(n)$.

3 Cuckoo Hashing

Worst case of both chaining and probing is $O(n)$. Expected is $O(1)$, for both insertion and searching. It utilized two hash tables T_1 and T_2 with theirs own hash functions h_1 and h_2.

Cuckoo Hashing sacrifice insertions for worst case $O(1)$ searching.

Algorithm 1 Cuckoo Hashing

Require: $i = 1, 2,$ Hash function f_1 and f_2, Hash table T_1 and T_2.

```
function INSERT(i, x)
    y ← T_i[h_i(x)]
    T_i[h_i(x)] ← x
    if y is not empty then
        Insert(y, 3 − i)
    end if
end function
function LOOK-UP(x)
    a ← T_1[h_1(x)]
    b ← T_2[h_2(x)]
    if a is x then
        return a
    end if
    if b is x then
        return b
    end if
    return ∅
end function
function DELETION(x)
    if T_1[h_1(x)] is x then
        T_1[h_1(x)] ← ∅
    else if T_2[h_2(x)] is x then
        T_2[h_2(x)] ← ∅
    end if
end function
```

However, this algorithm 1 could failed and there are two cases of it:

1. There is no enough space
2. The chain is too long

Both cases can be detected easily. And when the chain is too long (or infinitely), we just need to pick up two new hash function f_1, f_2 and re-hash the whole table again.

This algorithm has an overall $20 - 30\%$ overhead compared to linear probing.