Lecture 16: More hashing (11.3-11.4)

CS303: Algorithms

Last update: February 13, 2014

1 Review

- Idea of hashing: hashing data into indexes of the hashing table
- Goal: maintain near constant time efficiency ($\Theta(1+\alpha)$) for average case.

2 Division method

First some exercise about the division hashing $h(k) = k\ mod\ p$.

We want k to be as random as possible though we can't guarantee that.

e.g. student records, key = SSN. Hash function: $h(k) = k\ mod\ p$ where p is some integer (typically, prime)

Q:/if p = 1009, where is record with SSN= 314159265 stored?

A:/ h(314159265)=314159265 mod 1009=52 $\rightarrow$ hash value and also the index of the entry in T that stores this student's record

e.g. a fool and his money are soon parted

$h(k)$ = radix representation of the words mod 13. The reason for radix representation is to make the keys as random as possible.

In the String class, for example, the key code k of a string s of length n is calculated as radix $2^p - 1$ where p can be 5 for example.

if $p=5$, $2^5-1=31$, a length of n, table T has $m=13$ entries

Thus $k = S[0]*31^{n-1} + S[1]*31^{n-2} + \cdots + S[n-1]$

$key(A) = 65 * 31^0 = 65$ $h(A)=0$

$key(fool) = 3459660$ $h(fool)=9$

$key(and) = 102948$ $h(and)=1$

$key(his) = 109971$ $h(his)=4$

$key(money) = 118048025$ $h(money)=4$

$key(are) = 103077$ $h(are)=0$

$key(soon) = 3885646$ $h(soon)=11$

$key(parted) = -431301372$ $h(parted) = -8 \xrightarrow{+13} = 5$

Be careful here since the values can be pretty big.

Usually we pick m as a prime that isn't close to powers of 2.

3 Multiplication method (generally better than division)

Multiplication is usually easier to do than division

Let$m = 2^r$ and computers has w-bit words

Then $h(k) = (A * k\ mod\ 2^w)$ right shift $(w-r)$
A is usually an odd integer between 2^{m-1} and 2^m, not close to powers of 2.
e.g.

$m = 8 = 2^3$, $w = 7$

$A = 1011001$

if $k = 1101011$. (both A & k are of length w)

$$
\begin{array}{r}
1011001 \\
\times \quad 1101011 \\
\hline
1001010\underline{0\underbrace{110011}_{\text{our hash value}}}
\end{array}
$$

mod 2^w means only take the last 7 bit

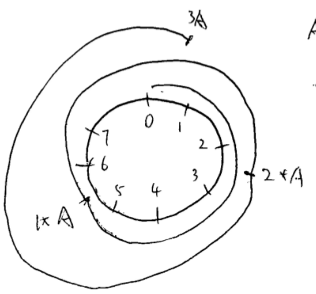
right shift $7-3=4$ bits

Thus $h(A) = 3$

e.g. the code for generating hashing values.

One common A for 32 bit machines is 2654435769 which makes $\frac{A}{2^{32}}$ close to $\frac{\sqrt{5}-1}{2}$
Q:/ Why don't we want A to be powers of 2?
A:/ Imagine if A in the previous example is 0001000. Then k*A will basically shift k to the left 3 bits. Then we will be looking at the bits from k at positions 2,3,and 4 from the right. This isn't random. All ks whose bits 2-4 are the same will be projected into the same entry in the hashing table.
e.g. if k is 1101011, our our result will be 101 which are the bits of 2,3,and 4 from the right of k.

Another way to think about it is in modular wheel. Imagine if A is 1011011 and we have a modula wheel, our wheel won't repeat that often.



A is about $1011011/2^7 \approx \frac{5.5}{8}$ th of a circle.

This means if k*A and k is big, we have a wheel of fortune event

Note: both division and multiplication methods are heuristic methods meaning there might be a set of K that make lots of collisions.

4 Another way to resolve collisions: open addressing /closed hashing

Idea: no storage for links (save some space)

If I hashed to a slot and there is something there. I hash again and again until an empty slot is found.

When searching, I follow the same order of probing. If successful, I found an element. If unsuccessful, found an empty slot

Then the hashing function will take two arguments (k and probe number) and map it to a number between 0 and m-1

Note:

- probe sequence should be a permutation
- table may fill up ($n \leq m$)
- deletion is difficult (be careful)

e.g. insert k=496

Probe $h(496, 0)$

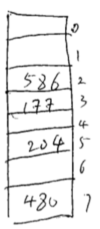
$= 3$

probe again

$h(496, 1) = 2$

probe again

$h(496, 2) = 4$



So insert 496 if I do insertion. if I am searching, then return Nil.

Q:/ How to design probing sequence?

A:/

1. one of the easiest is linear $h(k, i) = (h(k, 0) + i)\ mod\ m$

   what it means is you look at the next element in the table in the next probe. e.g.

Key	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
h(k)	1	9	6	10	7	11	11	12

If we use closed hashing, the results are

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

This suffers from local clustering because if one area is full in the table and it takes a long time to search/insert/delete elements in that area.

2. quadratic probing

   $h(k,i) = (h'(k) + c_1 i + c_2 i^2)\ mod\ m$. c_1 and c_2 are constants and h' is called an auxiliary hashing function.
   The behavior is basically to jump around a little bit instead of arranging the elements sequentially.

3. double hashing

   $h(k,i) = (h_1(k) + i \times h_2(k))\ mod\ m$. i is probing sequence starting with 0. Thus the first probing is purely based on h_1. Afterwards, h_2 comes in place.