Hashing: Static Methods

- Course Notes -

*** Hashing is a method of Information Retrieval - typically used for database management systems, other systems in which rapid storage and retrieval of information is necessary.**

*** Typical problem - Search for a record/object in a database that is associated with some key**

*** Hashing is done with a hash function such that -**

H: KEY------> INDEX or ADDRESS

*** Hash function:**

H(Primary Key) = External Key

The problem that occurs:

$H(K_1) = I_K = H(K_2)$

That is, 2 different keys hash to the same external location! This is called a COLLISION.

*** Hashing takes a potentially huge range of values and maps it to a much smaller range of values -**

E.G.s -

1) Students here at Chico State -

Large potential # of SSNs -

10 digits ---> 10^9 possible combinations

yet only about 15,000 actual students, so

H : SSN ----> Student File is "many to few"

2) All possible C++ identifiers (program names) and the compiler will store these in a symbol table by hashing for rapid retrieval:

If limited to 32 characters, only use alpha for first (all caps), alphanumeric (all caps) for others:

$$26*(36)^{31}$$

*** Sample Hash Functions:**

1) Division Method (MODULO arithmetic):

Note: '%' is the C++ MODULO operator

H: Key ----> Integer Index

E.g. - Table size of 100

3 Digit numbers are the keys

999 possible items

Indices 0..99 on the table

999 % 100 = 99 (100 is Table size)

524 % 100 = 24

199 % 100 = 99 (COLLISION)

2) Mid-Square Method - Concat, Square and Remove the Middle!

E.G. 32 character identifiers being hashed -

Table size [0..99]

A..Z ---> 1,2, ...26

0..9 ----> 27,...36

Identifier: CS1 --->3+19+28 (concat) = 31,928

$(31,928)^2$ = 1,019,397,184 - 10 digits

extract middle 2 digits (5th and 6th)

get 39, so:

H(CS1) = 39

3) Folding Method:

a) break key up into binary segments (ASCII)

b) XOR these together

c) Calculate the numeric integer equivalent

Hashing Examples:

1) Basic Division Method-

H(Key) = Key % 15,

Values to be hashed all > 0

0 indicates null value - or nothing there

Results after hashing 41, 58, 12, 92, 50 and 91:

Index Key

0	0
1	91

2	92
3	0
4	0
5	50
6	0
7	0
8	0
9	0
10	0
11	41
12	12
13	58
14	0

This is a nice distribution of values, no collisions!

2) Same hash function -

Now with values 10, 20, 30, 40, 50, 60, 70 -

Index Value

Overflow

0	30	60 (collision)
1	0	
2	0	
3	0	
4	0	
5	20	50 (collision)
6	0	
7	0	
8	0	
9	0	
10	10	40, 70 (collisions)
11	0	
12	0	
13	0	
14	0	

Conclusion: % 15 is a BAD HASH FUNCTION for this particular set of values!

In general: Choose the nearest prime number 1.5 times greater than the size of the data set you are hashing!

3) H(Key) = Key % 11

Same values as in last e.g. above:

Index Value

0	0
1	0
2	0
3	0
4	70
5	60
6	50
7	40
8	30
9	20
10	10

*** Handling Collisions - Techniques:**

Two Major Strategies:

1) Open Addressing - Find another spot in the "Table" (same contiguous address space)

2) Chaining - Find another spot outside the "Table"

***Open Addressing Techniques:**

a) Linear Probing - search sequentially (with wrap-around) until you find the first vacant slot

b) Quadratic Probing -

Hashed value to index i - slot i is occupied!

1st try after i ---> try i+1

2nd try after i ---> try i + 2^2

3rd try after i ---> try i + 3^2

(Always % tablesize, of course)

ETC.

c) Rehashing: When see spot is occupied, hash original key over with a second hash function - this to find another spot in the table.

*** Chaining Techniques:**

This technique "Chains" the item that collided to a location outside the "Table" - to another block of memory

(You'll do this with Dynamic, Extendible Hashing Techniques)

Problems with both Open Addressing and Chaining - can have very long searches for an item that collided a bunch with other items!

E.g.s -

1) Open Addressing with Linear Probing

Clustering can occur :

Suppose keys 160, 204, 219, 119, 412, 390, 263 are loaded and H is biased for returning 38-40!

Index Value

Hash values

0		
1		
2		
....	.....	.....
38	160	H(160)=38
39	204	H(204)=38
40	219	H(219)=38
41	119	H(119)=39
42	412	H(412)=39
43	390	H(390)=39
44	263	H(263)=40
....	.....	......
size		

Conclusion: Clustering can occur due to a biased hash function with linear probing as a collision resolution technique!

2) Quadratic Probing:

H(Key) = Key %11, Hashing values 13, 3, 24, 46, 90:

Index Value

0	46
1	
2	13
3	3
4	
5	
6	24
7	90

8	
9	
10	

Note the wraparound calculations!

Also, quadratic probing may never "get anywhere"-

H(K) = K % 8 ......

Index Value

0	
1	
2	
3	X - Initial hash position, plus 4th, 8th, 12th.... probes
4	1st, 3rd, 5th, 7th, 9th.... probes after collision at position 3
5	
6	
7	2nd, 6th, 10th.... probes after collision at position 3

Also, can have hashing to "Buckets" - More like the database situation where a "Bucket" is the size of a disk block that can fit n records/objects of size k, say:

E.g. - A Bucket Size of 3, H(K) = K%10

Index Slot 1 Slot 2 Slot 3

0	record with key 400	310	20
1	record with key 501	211	Empty
...			

...			
9	record with key 89	Empty	Empty

How does this contrast with chaining?

E.g. of Chaining -

Index Hashtable Chains

0 400 ---->310 ----->20 ----->50

1 501 ---->211

.....

9 89

****

Hash Table General Class Methods:

- **Construct a Hash Table**
- **Destroy a Hash Table**
- **Insert a Data Element into a Hash Table identified by a key**
- **Delete a Data Element from a Hash Table identified by a key**
- **Search for a Data Element in a Hash Table identified by a key**
- **Retrieve a Data Element in a Hash Table identified by a key**
- **Print a Data Element or Elements in a Hash Table identified by a key or keys**
- **

Go to Hashing: Dynamic Techniques - Course Notes