Hashtables

“Hashtables are arguably the single most important data structure known to humankind.”

- Google.

Hash Table

- Map a key to a value

- Many names:
  - Associative array, HashMap, Dictionary

Example Usage - Pseudocode

graduatedFrom[“DougRyan”] = “ColoradoState”

graduatedFrom[“ShereenKhoja”] = “Lancaster”

graduatedFrom[“ChaddWilliams”] = “WestVirginia”

print (graduatedFrom[“DougRyan”])

Example: C code

```
htInsertKeyAndData (&sHT, pKey, pData, keySize, dataSize);

htFindData (&sHT, pKey, pData, keySize, dataSize);
```

Operation

- A hash table maps keys to a certain location
  - bucket
- A hash function changes the key into an index value
  - hash value

(picture)

Possible Hash Function

Hash Function: Problem

- Create a fast dictionary that we can use to lookup the definition of 2 letter words
  - lower case only
- Total number of possible 2 letter words:
- Function to map each word to a unique integer

(picture)

Hash function

- Create a fast dictionary that we can use to lookup the definition of 5 letter words
  - lower case only
- Total number of possible 5 letter words:
- How many are actually valid words?
- Problems? Solutions?

http://www.thefreedictionary.com/5-letter-words.htm

Solution

Picture

Hash Function

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Collision

- Very difficult to have a hash function that never produces a collision
  - Perfect Hash - each key maps to an empty bucket!
  - very rare

- How should we handle the collision?

Our Problem

- Hash function maps keys to indexes
  - h(K) = M
  - indexes (0 to M-1)

- Problems
  - find suitable function
    - distribute keys evenly across the table
    - minimize collisions
  - find suitable M
  - handle collisions

MidSquare

- Turn the key into an integer
  - square the key
  - take some bits from the center of the square

https://research.cs.vt.edu/AVresearch/hashing/midsquare.php

Code

```
// get middle 8 bits from an int

// assume 4 byte integers
unsigned int key = 0x1231a456;
unsigned int middle;
middle = (key & 0x000ff000) >> 12;
printf("%08x %08x\n", key, middle);
```

http://www.eskimo.com/~scs/cclass/int/sx4ab.html

Division Hashing

- bucket = key % N
- N is length of table AND prime number

Collisions

- Open Addressing
  - find an empty slot
  - Linear probing
  - Quadratic probing
  - re-hash (double hash)
- Separate Chaining
  - each bucket is a linked list!

Open Addressing

- If two keys, K and C, map to the same bucket, we have a collision
- K and C are distinct
- K is inserted first
- Find unoccupied space for C
- Must be able to find C again next time!
- Analysis: (sum of the # of probes to locate each key) / # keys in the table

Open Addressing

- Find another open bucket
- bucket =

Linear Probing

- On collision, use the next empty spot
- On collision at h(n) try:
  - (h(n) +1) % tableSize
  - (h(n) +2) % tableSize
  - etc.
  -

Example

f(i) = i

h(Kn) = n % 11

Insert
M13
G7
Q17
Y25
R18
Z26
F6

Bucket	Data
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Primary Clustering

Quadratic Probing

- If h(n) is occupied, try
  - (h(n) + 1^2) mod table-size,
  - (h(n) + 2^2) mod table-size,
  - and so on until an empty cell is found

Example

f(i) = i^2

h(Kn) = n % 11

Insert
M13
G7
Q17
Y25
R18
Z26
F6

Bucket	Data
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Secondary Clustering

Re-Hashing

- Two hash functions
  - or use the same on twice

- The second hash function has to be chosen with care:
  - The sequence should be able to visit all slots in the table.
  - The function must be different from the first to avoid clustering.
  - It must be very simple to compute.

Double Hash

- f(i) = h2(k) * i
  - second hash function
  - unique probe sequence for every key
  - bucket =
  - h2(k) should be relatively prime to N for all K
    - don’t produce zero

Chaining

- Each bucket is a linked list!
- On collision, add item to list
- How does lookup work?

  How can we make chaining “faster”?

Problem

- Hash the keys M13, G7, Q17, Y25, R18, Z26, and F6 using the hash formula h(Kn) = n mod 9 with the following collision handling technique: (a) linear probing, (b) chaining
- Compute the average number of probes to find an arbitrary key K for both methods.
- avg = (summation of the # of probes to locate each key in the table) / # of keys in the table