

Module 5 - Hashing

Graham Cooper

June 16th, 2015

Search in a dictionary of n items

- Unsorted array $\Theta(n)$
- sorted array $\Theta(\log n)$
- AVL Tree $\Theta(\log n)$
- B-tree $\Theta(\log n)$

Can we do better than $\Theta(\log n)$?

Comparison Based Model

NO

In a comparison based model

- A search algorithm is a binary decision tree
- Each internal node is a comparison
- each leaf is a set of actions
- each instance (output) requires different actions
- $n+1$ possible outputs.

A binary decision tree with $n+1$ leaves

- The height of the tree (the number of required comparisons) is $\Omega(\log n)$

0.1 Direct accessing

- On implementation of Dictionary A.D.T. we have n item with integer keys which are $< M$

insert(3, John)

insert(5, andrew)

insert(6, shahini)

Hashing

A dictionary of n items with numerical keys

- A hash table of size M , a hash function \rightarrow maps each key to an integer in the range $[0, M]$

- To search, insert, delete look at the mapped index in a hash table

Assume $M = 7$,

- keys are integers and $h(k) = k \bmod 7$ - insert(26, "john") $\rightarrow h(26) = 5$

- insert(39, "hi") $\rightarrow h(39) = 4$

insert(21, "blah") $\rightarrow h(21) = 0$

insert(35, "STFF") $\rightarrow h(35) = 0$

Collision \rightarrow when two keys in dictionary share the same index in the hash table Load factor $\alpha = \frac{n}{M}$

It can be > 1 ≤ 1 , α should be constant (independent of n, m)

A table of size m , n changes by insertion deletions,

- When load factor becomes too large - too many collisions, slower operation

- Too small - space waste In both cases we should do a rehash

Collision Handling - Chaining

$h(k) = k \bmod 7$

insert(39) - $h(39) = 4$

insert(3) - $h(3) = 3$

insert(36) - $h(36) = 1$

insert(6) - $h(6) = 6$

insert(20) - $h(20) = 6$

We create a linked list, the start is 6, then 20

Getting a value from the linked list is still $O(1)$ since the load balance means there are not a lot of collisions

Average length of lists

\exists are n items with m indices

If the hash function distributes items equally between indices, each bucket has $\frac{n}{m} = \alpha$ items

Time complexity of n

search $O(1 + \alpha)$

insert - $O(1)$

delete = $O(1 + \alpha)$

Open addressing

Each key has a set of candidate indices $h(k,0), h(k,1), \dots h(k,i)$

Linear Probing

Add one to the index until we find an open address.

$$h(k,i) = (h(k) + i) \bmod M$$

$$\implies h(k,0) = h(k)$$

$$h(k,1) = h(k) + 1$$

$$M = 7, h(k) = k \bmod 7 \text{ insert}(6)$$

$$\text{insert}(4)$$

$$\text{insert}(0)$$

$$\text{insert}(2)$$

$$\text{insert}(17) - h(17) = 3$$

$$\text{insert}(7) - h(7) = 0$$

$$\text{insert}(7,b) = h(7) = 0 \text{ (is it available, no)}$$

$$h(7,1) = (h(7) + 1) \bmod 7 = 1 \text{ (good)}$$

$$\text{insert}(16) - h(16,0) = h(16) = 2 \text{ (bad)}$$

$$\text{insert}(16,1) - (h(16) + 1) = 3 \text{ (bad)}$$

$$\text{insert}(16,2) - h(16) + 2 = 4 - \text{good!}$$

$$M = 7, h(k) = k \bmod 7 \text{ insert}(2)$$

```

insert(3)
insert(9)
- h(9) = 2
delete(3)
search(9)

```

We need to mark things as deleted so we do not stop at deleted indexes when we are searching for 9, so that we do not think that they key does not exist as we advance

Requires that $n \leq m \implies \alpha \leq 1$

Double Hashing or Two Hash function

```

first candidate h(k,0) = (h1(k)) mod m
Second candidate h(k,1) = (h1(k, 1) + h2(k)) mod m
third candidate h(k, 2) = (h1(k) + 2h2(k)) mod m
...
h(k,i) = (h1(k) + ih2(k)) mod m

```

$m = 7, h_1(k) = k \bmod 7, h_2(k) = (2k) \bmod 7$

```

insert(4)
- h(k,0) = h1(k) = 4 mod 7 = 4
insert(9)
- h(k,0) = h1(9) = 2
insert(13)
- h(13,0) = h1(13) = 6
insert(24)
- h(k,0) = h1(24) = 3
insert(16)
- h(16,0) = h1(16) = 2 THERE IS A COLLISION
- h(16,1) = h1(16) + h2(16) mod 7 = 2 + 32 mod 7 = 2 + 4 = 6 Collision again
- h(16,2) = h1(16) + 2h2(16) = 2 + 2 * 4 = 10 mod 7 = 3 Collision again
- h(16,3) = h1(16) + 3h2(16) = 2 + 3 * 4 = 14 mod 7 = 0 Good to go

```

```

Delete(24)

```

- $h(24,0) = h_1(24) = 24 \bmod 7 = 3$ Found 24, mark as deleted
- Search(16)
- $h(16,0) = 2 \rightarrow 9 \neq 16$ not found
- $h(16,1) = 6 \rightarrow 13 \neq 16$ not found
- $h(16,2) = 3 \rightarrow \text{deleted}$ keep searching!
- $h(16,3) = 0 \rightarrow 16 = 16$ we found it!!

Chaining vs Open Addressing

Ease of implementation

chaining is easier to implement

Space Efficiency, Balance Factor

chaining: α can be larger than 1, so space in the hash table is not wasted, but pointers are taking up memory

Open addressing: α must be less than one so there are always going to be empty spaces in the table but there are no pointers taking up memory

0.1.1 Speed

Double hashing is generally faster than Linear Probing

α	50%	60%	75%	90%
LP - search	1.5	2	3	5.5
LP - Insert	2.5	5	8.5	55.5
DH - Search	1.4	1.6	1.8	2.6
DH - Insert	1.5	2	3	5.5

Cuckoo Hashing

There are two hash functions h_1, h_2 then there are two candidate positions for each key, ie, $h_1(k), h_2(k)$

To insert a key k

- Place k at index $h_1(k)$

- if required remove key k' from $h_1(k)$
- place k' in the other candidate index for it
- this might require removing another k'' and repeating

$$k = 7, h_1(k) = k \bmod 7, h_2(k) = 2k \bmod 7$$

insert(8)

$$- h_1(8) = 1$$

insert(13)

$$- h_1(13) = 6$$

insert(9)

$$- h_1(9) = 2$$

insert(15)

- $h_1(15) = 1$ put 15 at index 1 and kick out 8 - $h_2(8) = 16 \bmod 7 = 2$ place 8 at index 2 and kick out 9

- $h_2(9) = 18 \bmod 7 = 4$ place 9 at index 4

Good for search/delete since there are only two indices to check

Insertion is a bit messy as there might be a loop after kicking out n keys (rehash)

Choosing a Hash Function

Goals:

- scramble the keys
- efficiently computable
- each index is equally likely

If we are hashing our SIN number, the first 3 digits are the same based on certain things such as immigration etc. Last 3 digits would be better because they are seemingly random

Date of birth:

Birth year - very bad! very few indexes

Birth day - better, since there are a possible 365 values

Phone Numbers

First three digits is bad b/c everyone is from ontario, many 519

Last 3 digits is much better

$h(k) = \text{floor}(M(kA - \text{floor}(kA)))$, where A usually is ϕ and M making M to be a power of 2