

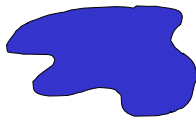
Hash Tables

Weiss, Chapter #5

Information Retrieval

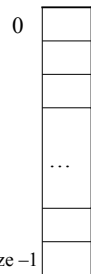
Hash Tables

- Constant time accesses!
- A **hash table** is an array of some fixed size, usually a prime number.
- General idea:



hash func.
 $h(K)$

hash table

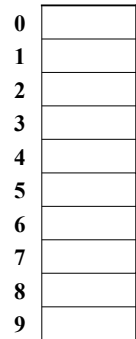


key space (e.g., integers, strings)

TableSize - 1

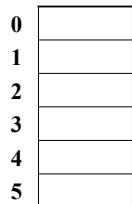
Example

- key space = integers
- TableSize = 10
- $h(K) = K \bmod 10$
- **Insert:** 7, 18, 41, 34
- How do we **find** them?



Another Example

- key space = integers
- TableSize = 6
- $h(K) = K \bmod 6$
- **Insert:** 7, 18, 41, 34
- How do we **find** them?



Hash Functions

1. **simple/fast** to compute,
2. Avoid **collisions**
3. have keys distributed **evenly** among cells.

Perfect Hash function:

Sample Hash Functions:

- key space = strings
- $s = s_0 s_1 s_2 \dots s_{k-1}$

1. $h(s) = s_0 \text{ mod TableSize}$

2. $h(s) = \left(\sum_{i=0}^{k-1} s_i \right) \text{ mod TableSize}$

3. $h(s) = \left(\sum_{i=0}^{k-1} s_i \cdot 37^i \right) \text{ mod TableSize}$

4/4/2005

CS 216, Spring 2005

7

Collision Resolution

Collision: when two keys map to the same location in the hash table.

Two ways to resolve collisions:

- Separate Chaining
- Open Addressing (linear probing, quadratic probing, double hashing)

4/4/2005

CS 216, Spring 2005

8

Separate Chaining

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Insert:

10
22
107
12
42

- Separate chaining:** All keys that map to the same hash value are kept in a list (or "bucket").

4/4/2005

CS 216, Spring 2005

9

Analysis of find

- Defn:** The **load factor**, λ , of a hash table is the ratio: $\frac{N}{M}$ ← no. of elements / table size

For separate chaining, λ = average # of elements in a bucket

- unsuccessful: λ
(avg. length of a list at hash(k))
- successful: $1 + (\lambda/2)$
(one node, plus half the avg. length of a list (not including the item)).

4/4/2005

CS 216, Spring 2005

10

How big should the hash table be?

- For Separate Chaining:

4/4/2005

CS 216, Spring 2005

11

Open Addressing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Insert:

38
19
8
109
10

- Linear Probing:** after checking spot $h(k)$, try spot $h(k)+1$, if that is full, try $h(k)+2$, then $h(k)+3$, etc.

4/4/2005

CS 216, Spring 2005

12

Open Addressing

Examine cells in the order:

$$h_0(k), h_1(k), h_2(k), \dots$$

where: $h_i(k) = (\text{hash}(k) + f(i)) \bmod \text{TableSize}$

Linear Probing

$$f(i) = i$$

- After searching spot $\text{hash}(k)$ in the array, look in $\text{hash}(k) + 1$, $\text{hash}(k) + 2$, etc.

Examine cells in the order:

$$h_0(k), h_1(k), h_2(k), \dots$$

where: $h_i(k) = (\text{hash}(k) + f(i)) \bmod \text{TableSize}$

Quadratic Probing

- $f(i) = i^2$
- After searching spot $\text{hash}(k)$, look in the 1st, 4th, 9th, etc. spots after $\text{hash}(k)$.
- Less likely to encounter primary clustering.

Quadratic Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Insert:

89
18
49
58
69

Examine cells in the order:

$$h_0(k), h_1(k), h_2(k), \dots$$

where: $h_i(k) = (\text{hash}(k) + f(i)) \bmod \text{TableSize}$

Double Hashing

- $f(i) = i \cdot \text{hash}_2(k)$
- Items that hash to the same location with $\text{hash}(k)$ won't have the same probe for $\text{hash}_2(k)$.

Rehashing

Idea: When the table gets too full, create a bigger table and hash all the items from the original table into the new table.

- When to rehash?
 - half full ($\lambda = 0.5$)
 - when an insertion fails
 - some other threshold
- Cost of rehashing

Hash Table example

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

+ marking deleted items
+ choice of table size