Hash Tables

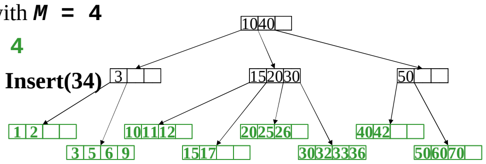
Chapter 5 in Weiss

CSE 326
Data Structures
Ruth Anderson

Today's Outline

- **Announcements**
  - **Project 2B due Wednesday, 2/10 at 11pm**
  - **Midterms returned and discussed in section Thurs**
  - **Written Homework #4 due Friday 2/12**
- **Today's Topics:**
  - **Hash Tables**

B-Tree with ***M* = 4**
and ***L* = 4**
Perform Insert(34)



(Only showing keys, but leaves also have data!)

Information Retrieval

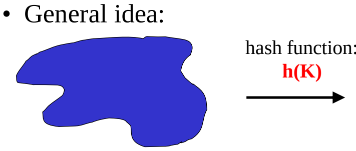
Implementations So Far

Student Activity

Hash Tables

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

hash table



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

TableSize –1

Example

- key space = integers
- TableSize = 10
- **h**(K) = K mod 10
- **Insert**: 7, 18, 41, 94

Another Example

- key space = integers
- TableSize = 6
- **h**(K) = K mod 6
- **Insert**: 7, 18, 41, 34, 32

Student Activity

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 = $s_0\ s_1\ s_2 \ldots s_{k-1}$

1. h(s) = s_0 mod TableSize
2. h(s) = $\left(\sum_{i=0}^{k-1} s_i\right)$ mod TableSize
3. h(s) = $\left(\sum_{i=0}^{k-1} s_i \cdot 37^i\right)$ mod TableSize

Designing a Hash Function for web URLs

s = $s_0\ s_1\ s_2 \ldots s_{k-1}$

Issues to take into account:

h(s) =

Collision Resolution

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

Two ways to resolve collisions:

1. Separate Chaining
2. Open Addressing (linear probing, quadratic probing, double hashing)

Separate Chaining

Insert:
10
22
107
12
42

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

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:
- successful:

How big should the hash table be?

- For Separate Chaining:

tableSize: Why Prime?

- Suppose
  - data stored in hash table: 7160, 493, 60, 55, 321, 900, 810
  - tableSize = 10
    data hashes to 0, 3, <u>0</u>, 5, 1, <u>0</u>, <u>0</u>
  - tableSize = 11
    data hashes to 10, 9, 5, 0, 2, <u>9</u>, 7

Real-life data tends to have a pattern

Being a multiple of 11 is usually *not* the pattern ☺

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.

Terminology Alert!

"**Open** Hashing" equals "Separate Chaining"

"Closed Hashing" equals "**Open** Addressing"

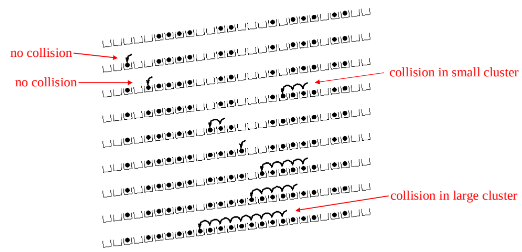
Weiss

Linear Probing

$f(i) = i$

- Probe sequence:
  - 0^{th} probe = h(k) mod TableSize
  - 1^{th} probe = (h(k) + 1) mod TableSize
  - 2^{th} probe = (h(k) + 2) mod TableSize
  - . . .
  - i^{th} probe = (h(k) + i) mod TableSize

Linear Probing – Clustering



[R. Sedgewick]

Load Factor in Linear Probing

- For *any* $\lambda < 1$, linear probing *will* find an empty slot
- Expected # of probes (for large table sizes)
  - successful search: $\frac{1}{2}\left(1+\frac{1}{(1-\lambda)}\right)$
  - unsuccessful search: $\frac{1}{2}\left(1+\frac{1}{(1-\lambda)^2}\right)$
- Linear probing suffers from ***primary clustering***
- Performance quickly degrades for $\lambda > 1/2$

Quadratic Probing

Less likely to encounter Primary Clustering

$f(i) = i^2$

- Probe sequence:
  - 0^{th} probe = h(k) mod TableSize
  - 1^{th} probe = (h(k) + 1) mod TableSize
  - 2^{th} probe = (h(k) + 4) mod TableSize
  - 3^{th} probe = (h(k) + 9) mod TableSize
  - . . .
  - i^{th} probe = (h(k) + i^2) mod TableSize

Quadratic Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Insert:
89
18
49
58
79

Quadratic Probing:

- h(k) = k mod 7
- Perform these inserts:
  - Insert(65)
  - Insert(10)
  - Insert(47)

0	
1	
2	93
3	
4	
5	40
6	76