CSE 326: Data Structures

Topic #10: Hashing (1)

Ashish Sabharwal
Autumn, 2003

Today's Outline

- Admin:
  - Project 2 due Monday night!
  - Midterm: the Monday after, in class
    Syllabus: everything covered so far + Hashing
  - Quick poll for Homework 2 (to be released on Monday)
    - (A) Short homework, due next Fri
      Will give out sample solutions on Fri
    - (B) Normal size homework, due the Wed after midterm
      No sample solutions before midterm
- Finish B-trees
- Start **Hashing**

Reminder: The Search ADT

- Data:
  - unique user-specified *keys*
  - Or: a set of keys
- Operations:
  - Insert (key)
  - Find (key)
    - Checks for membership
  - Remove (key)

insert(Grill) →

- Fry
- Simmer
- Puree
- Braise
- Poach
- Sear
- Stirfry
- Roast

← find(Bake)
NOT FOUND

*The Search ADT is sometimes called the "**Set ADT**"*

Reminder: The Dictionary ADT

- Data:
  - *values* mapped to user-specified *keys*
  - Or: a set of (key, value) pairs
- Operations:
  - Insert (key, value)
  - Find (key)
  - Remove (key)

insert(ashish…) →

- ashish@cs
  Ashish Sabharwal,
  instructor for CSE326,
  hoping to graduate soon!
- ethanpg@cs
  Ethan Phelps-Goodman,
  TA for CSE326,
  new grad student
- awong@cs
  Albert Jongkit Wong,
  TA for CSE326,
  5th year undergrad

← find(ethanpg)
• ethan
Ethan…, new grad…

An easy extension of the Search ADT!

*The Dictionary ADT is sometimes called the "**Map ADT**"*

Implementations So Far

	insert	find	delete
• Unsorted list			
• Sorted list			
• Trees			

How about O(1) insert/find/delete?

Hash Table Goal

We can do:

a[2] = "ethanpg"

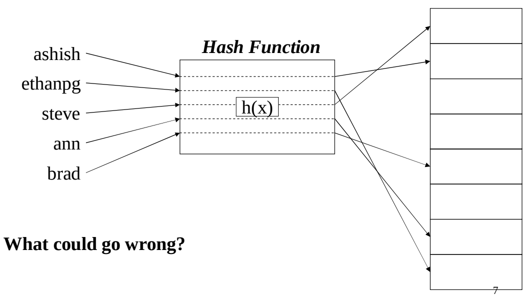
index	
0	
1	
2	ethanpg
3	
⋮	
k-1	

We want to do:

a["ethanpg"] = "Ethan..."

key	
"ann"	
"steve"	
"ethanpg"	ethanpg ...
"ashish"	
⋮	
"brad"	

Hash Table Approach



What could go wrong?

7

Hash Table Code: First Pass

```
Value find(Key k) {
  int index = hash(k) % tableSize;
  return Table[index];
}
```

Key Questions:

1. What should the **hash function** be?
2. How should we resolve **collisions**?
3. What should the **table size** be?

8

A Good Hash Function…

…is easy (fast) to compute
 (O(1) *and* practically fast)

…distributes the data evenly $\Rightarrow$ few collisions
 (ideally, hash(a) % size $\neq$ hash(b) % size $\Rightarrow$ no collision)

…uses the whole hash table
 ($\forall$ k, $0 \leq k <$ size, $\exists$ i such that hash(i) % size = k)

9

Good Hash Function for Integers

Choose
- tableSize to be <u>prime</u>
- hash(i) = i

Example:
- tableSize = 7

insert(4)
insert(17)
find(12)
insert(9)
delete(17)

0	
1	
2	
3	
4	
5	
6	

10

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 J

- More concrete reasons: next lecture!

11

Hash Functions for Strings

Let $s = s_1 s_2 s_3 \ldots s_k$. *Think ASCII values!*

A. $\text{hash}_A(s) = s_1 + s_2 + \ldots + s_k$

B. $\text{hash}_B(s) = \text{hash}(s_1 s_2 s_3) = s_0 + 37\, s_1 + 37^2\, s_2$

C. $\text{hash}_C(s) = s_0 + 37\, s_1 + \ldots + 37^k\, s_k$

Every Java object has a **hashcode() method**.
For strings, hashcode() is similar to hash_C above!

12