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)

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)

Implementations So Far

insert find delete

• Unsorted list

• Sorted list

• Trees

Hash Table Goal

We can do:

We want to do:

How about O(1) insert/find/delete?
**Hash Table Approach**

**Hash Function**

<table>
<thead>
<tr>
<th>ash</th>
<th>ethan</th>
<th>grego</th>
<th>steve</th>
<th>ann</th>
<th>brad</th>
</tr>
</thead>
</table>

What could go wrong?

**Hash Table Code:**

```java
int 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?

---

**A Good Hash Function…**

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

…distributes the data evenly ⇒ few collisions
(ideally, hash(a) % size ≠ hash(b) % size ⇒ no collision)

…uses the whole hash table
(∀ k, 0 ≤ k < size, ∃ i such that hash(i) % size = k)

**Good Hash Function for Integers**

Choose
- tableSize to be prime
- hash(i) = i

**Example:**
- tableSize = 7
- insert(4)
- insert(17)
- find(12)
- insert(9)
- delete(17)

---

**tableSize: Why Prime?**

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

Real-life data tends to have a pattern
Being a multiple of 11 is usually not the pattern.

- More concrete reasons: next lecture!

---

**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 `hashC` above!