CSE 373: Hash Tables

Chapter 5

Motivation

*Goal:* The ability to store and retrieve information in $O(1)$ time

*Current Solutions:*
Hash Table Insight

We can already do: 0 1 2 ...

We'd like to do: “brad” “maria” “sunliang” ...

n-1 “ethompson”

Hash Table Approach

“brad” “maria” “sunliang” “ethompson"
Hashing Conflicts

Hashing Integers

What’s a simple scheme for hashing integers?

```c
int HashInteger(int key, int tablesize) {

}
```

- advantages?
- disadvantages?
General Strategies

- Selecting a good hash function often depends on the set of possible keys
- Using a hash table whose size is a prime number tends to help reduce conflicts

Hashing Strings

General approach:
- convert string to integer
- “mod” integer by table size

Naive approach:

```c
int HashString(char *key, int tablesize) {
    int total = 0;
    while (*key) {
        total += *key;
    }
    return total % tablesize;
}
```
Problem with Naive Approach

\[
\text{HashString(“bat”, n)} = \text{HashString(“tab”, n)}; \\
= \text{HashString(“rad”, n)};
\]

Total Number of Possibilities \( \approx 127 \times 12 = 1524 \)
Useful Number of Possibilities \( = 27 \times 12 = 324 \)

Probably not good if hash table size or number of keys is greater than this...

Improved Approach

Read string as base 27 number:

\[
\begin{align*}
1 & \quad 27 & 729 \\
b & \quad a & \quad t = 2 \times 1 + 1 \times 27 + 20 \times 729 = 14,609 \\
t & \quad a & \quad b = 20 \times 1 + 1 \times 27 + 2 \times 729 = 1,505 \\
r & \quad a & \quad d = 18 \times 1 + 1 \times 27 + 4 \times 729 = 2,961 
\end{align*}
\]

Advantages?
Disadvantages?
Other Ideas

Hash using only a subset of the characters…
- first three?
- last three?
- middle three?
- first, middle, last?
- etc.

Hash Function Design Goals

- Hash to all slots in your table
- Avoid collisions
- Hash as evenly as possible
- Hash as quickly as possible

(Again, note that much of this may depend on the set of possible keys…)
Harsh Hash Reality

- No matter how good your hash function is, collisions will probably occur
- Thus, we also need a collision resolution strategy...
  - separate chaining
  - resizalbe hash table
  - open addressing

Hash Table Operations

- Main Operations:
  ```c
  void Insert(HashTable, HashType);
  Position Find(HashTable, HashType);
  void Delete(HashTable, Position);
  ```
- Normal Creation/Deletion Operations
- No iteration, `FindMin()`/`Max()`, etc.
  (why?)
Operator Analysis

<table>
<thead>
<tr>
<th>Hash Table</th>
<th>List</th>
<th>BST</th>
</tr>
</thead>
<tbody>
<tr>
<td>problem size</td>
<td>space</td>
<td></td>
</tr>
<tr>
<td>Insert()</td>
<td>Find()</td>
<td>Delete()</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Flashback to Day 1

```c
const int num_courses = 7000;
const int num_students = 33000;

typedef int registry[ num_students ][ num_classes ];
```