CSCI 104
Hash Tables Intro

Mark Redekopp
David Kempe
Sandra Batista
Motivation

Suppose a company has a unique 3-digit ID for each of its 1000 employees.

• We want a data structure that, when given an employee ID, efficiently brings up that employee’s record.

How should we implement this?

• An array gives $O(1)$ access time!

Alright, how do we obtain this runtime when the keys are no longer so nicely ordered??
Dictionaries/Maps

• An array maps **integers** to values
  – Given i, array[i] returns the value in O(1)

• Dictionaries map **keys** to values
  – Given key, k, map[k] returns the associated value
  – Key can be anything provided...
    • It has a '<' operator defined for it (C++ map) or some other comparator functor
    • Most languages implementation of a dictionary implementation require something similar to operator< for key types

Arrays associate an integer with some arbitrary type as the value (i.e. the key is always an integer)

C++ maps allow any type to be the key

map<string, double>

Pair<string, double>

"Tommy" 2.5

"Jill" 3.45
Dictionary Implementation

• A dictionary/map can be implemented with a balanced BST
  – Insert, Find, Remove = $O(\text{______________})$

```
Map::find("Greg")

"Frank" Student object
  "Anne" Student object
  "Greg" Student object

"Jordan" Student object

"Percy" Student object
  "Tommy" Student object

Map::find("Mark")
```
Dictionary Implementation

- A dictionary/map can be implemented with a balanced BST
  - Insert, Find, Remove = $O(\log_2 n)$
- Can we do better?
  - Hash tables (unordered maps) offer the promise of $O(1)$ access time

```
Map::find("Greg")

"Jordan" Student object
  "Frank" Student object
  "Anne" Student object
  "Greg" Student object

Map::find("Mark")

"Percy" Student object
  "Tommy" Student object
```
Unordered Maps / Hash Tables

- Can we use non-integer keys but still use an array?
- What if we just convert the non-integer key to an integer.
  - For now, make the unrealistic assumption that each unique key converts to a unique integer
- This is the idea behind a hash table
- The conversion function is known as a hash function, \( h(k) \)
  - It should be fast/easy to compute
    - \( O(x) \) – where \( x \) is the length of the key
  - It should consistently output the same result when given the same input
  - It should distribute keys well
    - We'd like every unique key to map to a different index, but that turns out to be almost impossible.

---

### Example

<table>
<thead>
<tr>
<th>Name</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bo</td>
<td>3.2</td>
</tr>
<tr>
<td>Tom</td>
<td>2.7</td>
</tr>
<tr>
<td>&quot;Jill&quot;</td>
<td>3.45</td>
</tr>
<tr>
<td>Tim</td>
<td>3.8</td>
</tr>
<tr>
<td>Lee</td>
<td>4.0</td>
</tr>
</tbody>
</table>

**Conversion function**

\[ h(k) \]

**A map implemented as a hash table**

(key = name, value = GPA)
Unordered_Maps / Hash Tables

• A hash table implements a map ADT
  – Add(key, value)
  – Remove(key)
  – Lookup/Find(key) : returns value

• In a BST the keys are kept in order
  – A Binary Search Tree implements an ORDERED MAP

• In a hash table keys are evenly distributed throughout the table (unordered)
  – A hash table implements an UNORDERED MAP

A map implemented as a hash table (key=name, value = GPA)
C++11 Implementation

• C++11 added new container classes:
  – unordered_map
  – unordered_set
• Each uses a hash table for average complexity to insert, erase, and find in $O(1)$
• Must compile with the `-std=c++11` option in g++
• Provides hash functions for basic types: int, string, etc. but for any other type you must provide your own hash function (like the operator $<$ for BSTs)
Hash Tables

- A hash table is an array that stores key, value pairs
  - Usually smaller than the size of possible set of keys, |S|
    - USC ID's = $10^{10}$ options
  - But larger than the expected number of keys to be entered (defined as $n$)
- The table is coupled with a function, $h(k)$, that maps keys to an integer in the range $[0..tableSize-1]$ (i.e. $[0..m-1]$)
- What are the considerations...
  - How big should the table be?
  - How to select a hash function?
  - What if two keys map to the same array location? (i.e. $h(k1) == h(k2)$)
    - Known as a collision
    - The probability of this should be low
Hash Tables are Awesome!

Hash Tables provide a very lucrative potential runtime. However, they are probabilistic.

• There was a similar problem with Splay Trees: they had a good average runtime, but a poor worst-case runtime.

As of this moment, we do not have the necessary mathematical framework to analyze either of these structures.

• We’re going to start remedying that... now.