Lecture 22: Hashing
CSCI 700 - Algorithms I

Andrew Rosenberg

December 1, 2009
Last Time

- Graphs
Hashing
A dictionary supports (minimally) **Insert**, **Search**, and **Delete**.

Today: Hash Tables – another dictionary data structure.
Limitations of arrays

- Comparison sort and comparison search on an open set is bound by $\Omega(n \log n)$
- Counting sort showed that if we have a closed domain of data (size $O(n)$), we can sort in linear time, $O(n)$.
- We can search a closed domain (size $O(n)$) in constant time, $O(1)$.
Searching in Constant Time

- Given a domain of size $O(n)$, construct an $O(n)$ element $T$ containing the elements in $A$.

- Write a `lookup` function to map the elements of the domain to indices in $T$.
  - `lookup` might be a case statement, an enumeration, or nested ifs depending on language support.
  - Regardless of implementation `lookup` is $O(1)$.

Search($T, x$)

```plaintext
return $T[\text{lookup}(x)]$
```

Insert($T, x$)

```plaintext
$T[\text{lookup}(x)] \leftarrow x$
```

Delete($T, x$)

```plaintext
$T[\text{lookup}(x)] \leftarrow \emptyset$
```
Constructing a table $T$ with size equal to the number of keys $U$ you want to index might be impractical.

- Say, if you want to index any strings with less than 32k characters.
  - Minimally $32,000^{27}$
Constructing a table $T$ with size equal to the number of keys $U$ you want to index might be impractical.

- Say, if you want to index any strings with less than 32k characters.
- Minimally $32,000^{27}$

The solution: Map an open set $U$ onto a closed set $K$ which is much smaller than $U$.

- This mapping is performed using a hash function.
  - Let $|K| = m$
  - $h : U \rightarrow \{0, 1, \ldots, m - 1\}$

Hashing allows for a dictionary with $\text{INSERT}$, $\text{DELETE}$ and $\text{SEARCH}$ with expected runtimes of $O(1)$. 
Hash Table

$U$ (universe of keys)

$K$

- $k_1$
- $k_2$
- $k_3$
- $k_4$
- $k_5$

$T$

- $0$
- $h(k_3)$
- $h(k_1)$
- $h(k_2)$
- $h(k_5)$
- $h(k_4)$
- $m-1$
Using a **hash function**, we can store elements of an open set in a small data structure.

For example:

- `INSERT("Andrew")`
- `INSERT("Michael")`
- `INSERT("John")`
- `SEARCH("Michael") = "Michael"
- `SEARCH("Sally") = ∅`
Hashing as an associative data structure

In practice, hash tables are used to store key/value pairs.

For example: Names (strings) are keys, Ages (integers) are values.

- **Insert**("Andrew", 30)
- **Insert**("Michael", 33)
- **Insert**("John", 15)
- **Search**("Michael") = 33
- **Search**("Sally") = ∅
- **Search**("Andrew") = 30
Hashing as an associative data structure

This allows a user to index a data structure by an element of an open set.

Arrays

Hash Tables
- $H["Andrew"] = 30$
- $H["Michael"] = 33$
- $H["John"] = 15$
Hashing

$U$ (universe of keys)

$K$

$k_1$, $k_2$, $k_3$, $k_4$, $k_5$

$h(k_1)$, $h(k_2)$, $h(k_3)$, $h(k_4)$, $h(k_5)$

$T$

$0$, $v_1$, $v_2$, $v_3$, $v_4$, $v_5$, $m-1$
What’s the catch?
The catch

What’s the catch?

Collisions
Figure 11.2 Cormen
Since $U$ is much larger than $m$, the size of the hash table, there are multiple elements in $U$ that have the same hash value $h(k_1) = h(k_2)$.

- Pigeon hole principle: if $n$ items are put into $m$ pigeon holes with $n > m$, then at least one pigeon hole must contain more than one item.
**Problem**: More than one key needs to occupy a single hash table entry.

**Solution**: Allow each hash table entry to hold more than one key.
Problem: More than one key needs to occupy a single hash table entry.

Solution: Allow each hash table entry to hold more than one key.

- Each element of the hash table is a list.
- **INSERT**($k, v$):
  Insert $v$ at the head of list $T[h(k)]$
- **SEARCH**($k$):
  Search for an key $k$ at the head of list $T[h(k)]$
- **DELETE**($k$):
  Delete $k$ from list $T[h(k)]$
How much space is required to store $N$ elements in a hash table with $m$ entries with chaining?
How much space is required to store $N$ elements in a hash table with $m$ entries with chaining?

What is the worst case runtime for \textsc{Insert}, \textsc{Search} and \textsc{Delete}?
Performance of a hash table with Chaining

How much space is required to store $N$ elements in a hash table with $m$ entries with chaining?

What is the worst case runtime for \texttt{INSERT}, \texttt{SEARCH} and \texttt{DELETE}?

The best case?
How much space is required to store $N$ elements in a hash table with $m$ entries with chaining?

What is the worst case runtime for Insert, Search and Delete?

The best case?

What makes the difference?
Performance of a hash table with Chaining

How much space is required to store $N$ elements in a hash table with $m$ entries with chaining?

What is the worst case runtime for Insert, Search and Delete?

The best case?

What makes the difference?

Load factor: $\alpha = \frac{n}{m}$
Not all hash functions are equally good.

Let $s \in U$ be the set of all strings with $< 32k$ characters. Consider the following hash functions.

- $h : U \rightarrow 1$
Identifying a **good** hash function

Not all hash functions are equally good.

Let $s \in U$ be the set of all strings with $< 32k$ characters. Consider the following hash functions.

- $h : U \rightarrow 1$
- $h : U \rightarrow \text{int}(s[1])$

The more evenly distributed $h(k)$ is the better.
Not all hash functions are equally good.

Let $s \in U$ be the set of all strings with $< 32k$ characters. Consider the following hash functions.

- $h : U \rightarrow 1$
- $h : U \rightarrow \text{int}(s[1])$
- $h : U \rightarrow \sum_i^n \text{int}(s[i])$

The more evenly distributed $h(k)$ is the better.
h(k) must be bound by $m$ the size of the hash table.

How do we guarantee that?

**Division Method** $h(k) = k \mod m$

- As a rule of thumb, $m$ is selected to be prime and far from a power of 2.
  - If $m = 2^p$, then the hash is just the lower $p$ bits of $k$.
  - This is probably *not* evenly distributed.
Identifying a good size for a hash table

h(k) must be bound by $m$ the size of the hash table.

How do we guarantee that?

**Multiplication Method** Choose a constant $A$, such that $0 < A < 1$.

$$h(k) = \lfloor m(kA \mod 1) \rfloor$$

- $(kA \mod 1)$ - the fractional part of $kA$.

The distribution is independent of $m$. Allowing hash tables with sizes $m = 2^p$.

- Knuth 1973 - The Art of Programming vol. 3: $A \approx (\sqrt{5} - 1)/2$ works reasonably well for most keys.
- There are other machine considerations that can be taken into account.
Next time (12/3)
- NP-completeness
  - Read Cormen Chapter 34