Dictionary Operations

- Dictionary operations: insert, search, delete
  - Search means...
  - Was the key inserted into the table?
  - Set membership
  - To find the value of satellite data associated with the key
  - Should be efficient. Hopefully $O(1)$

Overview

- Hash Functions Chapter 11.3
- Chaining
- Open Addressing (Chapter 11.5)
- Hash Tables
Hash-Tables

- Reduce storage requirements but still maintain $O(1)$ access time
- When $|Y| \ll |\Omega|$
• Hash Functions (Chapter 11.3)
• Open Addressing (Chapter 11.5)
• Chaining

Overview

Collisions

• Two keys might hash to the same value
• Will hopefully map keys randomly across the hash table
• Hash function is deterministic: $h(k)$ is always same value
• Try to avoid collisions as much as possible
• Can happen since size of universe $\ll$ size of hash table

Overview

Collisions
Efficiency of Dictionary Operations

- **Insert(T,x)** - Insert x at the head of list
  - If we do not check if x.key is already in list: \(O(1)\)

- **Search(T,k)** - Search for an element with key k in list
  - Worst case: proportional to length of list
  - \(O(n)\)

- **Delete(T,x)** - Delete x from list
  - \(O(1)\) time since we already have pointer to element and it is doubly-linked

Collision Resolution by Chaining

\(T\) (universe of keys)
\(K\) (actual keys)
\(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8\)

\(n\) (number of keys)

\(x, y\) (actual keys)

\(T\) (universe of keys)

\(x\) (key of x)

\(y\) (key of y)

(left) Insert(x)

(right) Delete(T,x)
Theorem 11.1

In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1 + \alpha)$ under simple uniform hashing assumption.

Proof:

Under assumption of simple uniform hashing, any key $k$ not already stored in the table is equally likely to hash to any of the $m$ slots. The expected number of elements examined in an unsuccessful search is $\alpha$, and total time including computing $h(k)$ is $\Theta(1 + \alpha)$.

Load factor $\alpha = \frac{n}{m}$ where $n$ is keys stored, and $m$ is size of table.

Analysis of Hashing with Chaining

- Load factor $\alpha = \frac{n}{m}$ where $n$ is keys stored, and $m$ is size of table.
- Worse-case for search is $\Theta(n)$ for one linked list for all elements.
- Average-case performance of hashing depends on how well $h$ distributes keys among $m$ slots on average.
- Load factor $\alpha = \frac{n}{m}$ where $n$ is keys stored, and $m$ is size of table.

Assume hash of key is independent of keys already inserted.
Implications

- Load factor $\alpha = \frac{n}{m}$ where $n$ is keys stored and $m$ is size of table
- Search time depends on load factor
- If number of keys used is proportional to size of table
  + Search is $O(1)$ time

Successful Search

Theorem 11.2
In a hash table in which collisions are resolved by chaining, a
successful search takes average-case time $O(\alpha + 1)$, under simple
uniform hashing.

Proof:
We assume element being searched for is equally likely to be any of the $n$
elements stored in the table.
How far $x$ is from front of list?
How many other elements are in the same list?
How many other elements are equally likely to be $x$?

We have $\Theta(1 + \alpha)$ for the search.

To be $O(1 + \alpha)$ time,
What Makes a Good Hash Function?

- Make hash function independent of any patterns that might exist in data

  Good approach

+ Don't use suffixes or prefixes

- Hashing English words: be careful of the 'a' and 's'

  + Good hash function?

  - Hashing in range 0 ≤ k < l

  - If we know keys are random real numbers \( k \) independently and uniformly distributed

  - Domain knowledge might help in designing hash function

  - But really, know the probability distribution from which keys are drawn

  - Each key is equally likely to hash to any of the \( m \) slots regardless of what else has hashed

  Should satisfy assumption of simple uniform hashing

Overview

- Open Addressing (Chapter 11.5)
- Hash Functions (Chapter 11.3)
- Chaining
- Hash Tables
Multiplication Method

- First multiply key $k$ by a constant $A$ in the range $0 < A < 1$ and extract the fractional part of $kA$
- Then multiple it by $m$ and take the floor of the result

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

Advantage

- Reduces dependency on $m$

Division Method

- Assume keys are in the set of natural numbers
- Otherwise find way to map them to numbers
- Take remainder of $k$ modulo $m$: $h(k) = k \mod m$
- Otherwise find way to map them to numbers
- Prime near $2000/3$, not near any power of 2
- If $m = 2000$, and $0 = 0 \mod 2$, seems reasonable. Can pick $m = 701$ since $701$ is a prime not close to a power of 2, seems to work well
- Even using $m = 2^p - 1$ is problematic (see textbooks)
- Better if depend on all of the bits of the key
- Otherwise just using the lowest-order bits of $k$
- Do not use $m$ as a power of 2
- Seems that we pick size of table so that it works well for hashing
- Allows you to map onto all of the slots in the table
- Alternatively find way to map them to numbers
- Otherwise keys are in the set of natural numbers
Example

• Some values of \( A \) work better than others
  - Knuth suggests \( A \approx \left( \sqrt{5} - 1 \right) / 2 = 0.6180339887 \).

Example

\[ k = 123456 \]
\[ p = 14 \]
\[ m = 2 \]
\[ 327700222976 = 16384 \]
\[ w = 32 \]
\[ A = s/2 \]
\[ 2^{32} = 0.6180339887 \approx \sqrt{5} - 1 \]

Some values of \( A \) work better than others.

Typical approach

• Choose \( m \) be power of 2 (\( m = 2^p \))
• Suppose word size of machine is \( m \) bits and \( k \) fits into a single word
• \( A = s/2^w \) where \( s \) is integer
• First multiple \( k \) by \( s \)-bit integer
  \[ (s \cdot k) \bmod m \]
• Result is \( 2^w \) bit value
• \( r_1 = 2^{32} = 2^{14} = 16384 \)
• \( r_0 = 123456 \)
• Hash value is \( p \) most significant bits of \( r_0 \times s \)

\[ d = (\phi \bmod 1) \]
Open Addressing

- Hash table can fill up
- Each table entry contains either an element of dynamic set or "Nil"
- Load factor $\alpha$ cannot exceed 1
- When searching, systematically examine table slots until:
  - find desired element
  - ascertain element is not in table
- Do not use chaining to a linked-list for collisions

Overview

- Hash Functions
- Chapter 11.3
- Chaining
- Hash Tables
- Open Addressing (Chapter 11.5)
Deletion

Deletion from an open address table is difficult

- Open table not commonly used when deletion is needed
- Search times no longer depend on load factor α
- When insertion viewed as having a value
  - When searching, viewed as having a value
- Can add a special value 'Deleted'
  - When searching, viewed as having a value
  - When inserting, viewed as nil

- Search times no longer depend on load factor α
- Open table not commonly used when deletion is needed
- Any advantage of chaining with a linked-list?

Insertion

- Successively probe table until you find an empty slot to put the key
- Rather than probe starting at 0 (would require Θ(n) probes)
  - Hash function takes input key and probe number
  - Sequence of probes depends on key being inserted
  - Successful probe table until you find an empty slot to put the key

- Hash function takes input key and probe number
  - Sequence of probes depends on key being inserted
  - Rather than probe starting at 0 (would require Θ(n) probes)
  - Hash function takes input key and probe number
  - Sequence of probes depends on key being inserted
  - Rather than probe starting at 0 (would require Θ(n) probes)

- Hash function takes input key and probe number
  - Sequence of probes depends on key being inserted
  - Rather than probe starting at 0 (would require Θ(n) probes)
  - Hash function takes input key and probe number
  - Sequence of probes depends on key being inserted
  - Rather than probe starting at 0 (would require Θ(n) probes)
Linear Probing

Let $h': U \rightarrow \{0, 1, \ldots, m-1\}$ be an ordinary hash function. Referred to as auxiliary hash function.

Hash function: $h(k, i) = (h'(k) + i) \mod m$

Initial probe determines sequence: only $m$ distinct probe sequences.

Primary Clustering

If there are $i$ slots filled in a role, odds are $i/m$ that hash function will do initial hash to it, and cause cluster to grow by one.

- Will increase search times.
- Initial hash to it and cause cluster to grow by one.

Uniform Hashing

- Uniform Hashing generalizes simple uniform hashing.
- Probe sequence of each key is equally likely to be any of the $m!$ permutations of $\{0, 1, \ldots, m-1\}$.
- Do not guarantee that each table entry is included.
- Difficult to implement, usually approximated.

Uniform Hashing

- Generalizes simple uniform hashing.
Double Hashing

\[ h(k, i) = (h_1(k) + ih_2(k)) \mod m \]

• Initial probe depends on \( h_1 \)
• Successive probes are offset by \( h_2 \)

- We can use \( O(\sqrt{m}) \) and \( \Theta(m) \) probe sequences.
- Either approach gives \( \Theta(m^2) \) probe sequences.
- Let \( m \) be a prime number, \( h_2(k) \) is a positive number less than \( m \).
- Let \( m \) be a power of 2, and make \( h_2(k) \) always return odd numbers.
- Either approach gives \( \Theta(m^2) \) probe sequences.
- \( h_2(k) \) must be relatively prime to the hash-table size \( m \) for entire hash table to be searched.
- Now keys with same initial probe will not follow same probe sequence.
- \( h_2(k) \) must be relatively prime to the hash-table size \( m \).

- Let \( m \) be a power of 2 and make \( h_2(k) \) always return odd numbers.
- If \( m \) is prime, \( h_2(k) \) must be relatively prime to the hash-table size \( m \).
- \( h_2(k) \) must be relative prime to the hash-table size \( m \). For \( h_2(k) \) to be relatively prime to the hash-table size \( m \), \( h_2(k) \) must be relatively prime to \( m \).
- \( h_2(k) \) must be relatively prime to the hash-table size \( m \).
- \( h_2(k) \) must be relatively prime to the hash-table size \( m \).
- \( h_2(k) \) must be relatively prime to the hash-table size \( m \).
- \( h_2(k) \) must be relatively prime to the hash-table size \( m \).

Quadratic Probing

\[ h(k, i) = (h'_1(k) + ci + (\eta'h)\eta) \mod m \]

where \( h'_1(k) \) is an auxiliary hash function.

- \( c_1 \) and \( c_2 \) are positive auxiliary constants.
- \( \eta \) is an auxiliary hash function.
- \( h' \) is an auxiliary hash function.

- \( \Theta \), \( O \), and \( \Omega \) are used for asymptotic analysis.
Theorem 11.6
Given an open-address hash table with load factor $\alpha = n/m < 1$ (and no deletions) and uniform hashing, expected number of probes in an unsuccessful search is at most
\[ \frac{1}{1 - \alpha}. \]

Inituition
Always make a first probe.
Make a second probe if first probe is unsuccessful.
Make a third probe if second probe is unsuccessful.
Make a fourth probe if third probe is unsuccessful.

For chaining, $1 + \alpha$ probes in an unsuccessful search is at most $1/1 - \alpha$.

Given an open-address hash table with load factor $\alpha$, expected number of probes in an unsuccessful search is at most $1/1 - \alpha$.