Hashing
Chapter 5 in Weiss

CSE 373
Data Structures and Algorithms
Ruth Anderson

Today’s Outline

• Announcements
  – Homework #4 due Fri, Nov 5 at the beginning of class.

• Today’s Topics:
  – Disjoint Sets & Dynamic Equivalence
  – Hashing

Hash Tables

• Constant time accesses!
• A hash table is an array of some fixed size, usually a prime number.
• General idea:

  key space (e.g., integers, strings) \[ \text{hash function: } h(K) \] 
  \[ \text{TableSize - 1} \]

Example

• key space = integers
• TableSize = 10
• \( h(K) = K \mod 10 \)
• Insert: 7, 18, 41, 94

Another Example

• key space = integers
• TableSize = 6
• \( h(K) = K \mod 6 \)
• Insert: 7, 18, 41, 34

Student Activity
Hash Functions

1. **simple/fast** to compute,
2. Avoid **collisions**
3. have keys distributed **evenly** among cells.

Perfect Hash function:

Sample Hash Functions:

- key space = strings
- $s = s_0 s_1 s_2 \ldots s_{k-1}$

1. $h(s) = s_0 \mod \text{TableSize}$
2. $h(s) = \left( \sum_{i=0}^{k-1} s_i \right) \mod \text{TableSize}$
3. $h(s) = \left( \sum_{i=0}^{k-1} s_i \cdot 37^i \right) \mod \text{TableSize}$

Designing a Hash Function for web URLs

$s = s_0 s_1 s_2 \ldots s_{k-1}$

Issues to take into account:

$h(s) =$

Collision Resolution

**Collision**: when two keys map to the same location in the hash table.

Two ways to resolve collisions:
1. Separate Chaining
2. Open Addressing (linear probing, quadratic probing, double hashing)

Separate Chaining

<table>
<thead>
<tr>
<th>Insert:</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>107</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>42</td>
</tr>
</tbody>
</table>

- **Separate chaining**: All keys that map to the same hash value are kept in a list ("bucket").

Analysis of find

- The **load factor**, $\lambda$, of a hash table is the ratio:
  
  $\lambda = \frac{\text{no. of elements}}{\text{table size}}$

  For separate chaining, $\lambda = \text{average # of elements in a bucket}$

  - unsuccessful:
  - successful:
How big should the hash table be?

• For Separate Chaining:

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, 2, 7

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

Open Addressing

Insert:

0
1
2
3
4
5
6
7
8
9

• Linear Probing: after checking spot \( h(k) \),
  try spot \( h(k)+1 \), if that is full, try \( h(k)+2 \),
  then \( h(k)+3 \), etc.

Terminology Alert!

“Open Hashing” equals “Separate Chaining”
“Closed Hashing” equals “Open Addressing”

Weiss

Linear Probing

\[ f(i) = i \]

• Probe sequence:
  0th probe = \( h(k) \mod TableSize \)
  1st probe = \( (h(k) + 1) \mod TableSize \)
  2nd probe = \( (h(k) + 2) \mod TableSize \)
  ...
  \( i \)th probe = \( (h(k) + i) \mod TableSize \)

Linear Probing – Clustering

[R. Sedgewick]
Load Factor in Linear Probing

- For any $\lambda < 1$, linear probing will find an empty slot.
- Expected # of probes (for large table sizes):
  - successful search:
    $$\frac{1}{2}\left(1 + \frac{1}{(1-\lambda)}\right)$$
  - unsuccessful search:
    $$\frac{1}{2}\left(1 + \frac{1}{(1-\lambda)^2}\right)$$
- Linear probing suffers from primary clustering.
- Performance quickly degrades for $\lambda > 1/2$.

Quadratic Probing

- $f(i) = i^2$
- Probe sequence:
  - $0^{th}$ probe = $h(k)$ mod TableSize
  - $1^{st}$ probe = $(h(k) + 1)$ mod TableSize
  - $2^{nd}$ probe = $(h(k) + 4)$ mod TableSize
  - $3^{rd}$ probe = $(h(k) + 9)$ mod TableSize
  - $\ldots$
  - $i^{th}$ probe = $(h(k) + i^2)$ mod TableSize.

Quadratic Probing Example

- $h(k) = k$ mod 7
- Perform these inserts:
  - Insert(65)
  - Insert(10)
  - Insert(47)

But... 47%? = 5

Quadratic Probing: Success guarantee for $\lambda < 1/2$

- If size is prime and $\lambda < 1/2$, then quadratic probing will find an empty slot in $\text{size}/2$ probes or fewer.
  - show for all $0 \leq i, j \leq \text{size}/2$ and $i \neq j$:
    - $(h(x) + i^2) \mod \text{size} \neq (h(x) + j^2) \mod \text{size}$
    - by contradiction: suppose that for some $i \neq j$:
      - $(h(x) + i^2) \mod \text{size} = (h(x) + j^2) \mod \text{size}$
      - $(i^2 - j^2) \mod \text{size} = 0$
      - $(i+j)(i-j) \mod \text{size} = 0$
      - BUT size does not divide $(i+j)$ or $(i-j)$.
Quadratic Probing: Properties

- For any $\lambda < \frac{1}{2}$, quadratic probing will find an empty slot; for bigger $\lambda$, quadratic probing may find a slot.
- Quadratic probing does not suffer from primary clustering: keys hashing to the same area are not bad.
- But what about keys that hash to the same spot? – Secondary Clustering!

Double Hashing

\[ f(i) = i \times g(k) \]
where $g$ is a second hash function

- Probe sequence:
  0th probe = $h(k) \mod \text{TableSize}$
  1st probe = $(h(k) + g(k)) \mod \text{TableSize}$
  2nd probe = $(h(k) + 2g(k)) \mod \text{TableSize}$
  3rd probe = $(h(k) + 3g(k)) \mod \text{TableSize}$
  \ldots
  $i$th probe = $(h(k) + i\times g(k)) \mod \text{TableSize}$

Double Hashing Example

\[
\begin{align*}
0 & \quad 76 & 0 & 0 & 0 & 0 & 0 \\
1 & \quad 76 & 93 & 40 & 47 & 10 & 55 \\
2 & \quad 76 & 93 & 2 & 2 & 93 & 2 \\
3 & \quad 76 & 6 & 1 & 1 & 47 & 1 \\
4 & \quad 76 & 93 & 40 & 40 & 40 & 5 \\
5 & \quad 76 & 93 & 40 & 5 & 40 & 5 \\
6 & \quad 76 & 93 & 40 & 5 & 40 & 5 \\
\end{align*}
\]

Insert these values into the hash table in this order. Resolve any collisions with double hashing:

Hash Functions:

- $h(k) = k \mod M$
- $h_2(k) = 1 + ((k/M) \mod (M-1))$
- $M$

Rehashing

**Idea:** When the table gets too full, create a bigger table (usually 2x as large) and hash all the items from the original table into the new table.
- When to rehash?
  - half full ($\lambda = 0.5$)
  - when an insertion fails
  - some other threshold
- Cost of rehashing?

Hashing Summary

- Hashing is one of the most important data structures.
- Hashing has many applications where operations are limited to find, insert, and delete.
- Dynamic hash tables have good amortized complexity.