CSE 373: Data Structures and Algorithms

Lecture 17: Hashing II
# Hash versus tree

- Which is better, a hash set or a tree set?

<table>
<thead>
<tr>
<th>Hash</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Implementing Set ADT (Revisited)

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Remove</th>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unsorted array</strong></td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td><strong>Sorted array</strong></td>
<td>$\Theta(\log(n)+n)$</td>
<td>$\Theta(\log(n) + n)$</td>
<td>$\Theta(\log(n))$</td>
</tr>
<tr>
<td><strong>Linked list</strong></td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td><strong>BST (if balanced)</strong></td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td><strong>Hash table</strong></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
Probing hash tables

• Alternative strategy for collision resolution: try alternative cells until empty cell found
  – cells $h_0(x), h_1(x), h_2(x), ...$ tried in succession, where $h_i(x) = (\text{hash}(x) + f(i)) \mod \text{TableSize}$
  – $f$ is collision resolution strategy
  – bigger table needed
Linear probing

- **linear probing**: resolve collisions in slot $i$ by putting colliding element into next available slot ($i+1$, $i+2$, ...)

- Psuedocode for insert:
  
  ```
  first probe = h(value)
  while (table[probe] occupied)
    probe = (probe + 1) % TableSize
  table[probe] = value
  ```

- add 41, 34, 7, 18, then 21, then 57

- lookup/search algorithm modified - have to loop until we find the element or an empty slot
  - what happens when the table gets mostly full?
Linear probing

• $f(i) = i$

• Probe sequence:
  
  $0^{\text{th}}$ probe = $h(x) \mod TableSize$
  
  $1^{\text{th}}$ probe = $(h(x) + 1) \mod TableSize$
  
  $2^{\text{th}}$ probe = $(h(x) + 2) \mod TableSize$
  
  $\ldots$
  
  $i^{\text{th}}$ probe = $(h(x) + i) \mod TableSize$
Deletion in Linear Probing

• To delete 18, first search for 18

• 18 found in bucket 8

• What happens if we set bucket 8 to null?
  – What will happen when we search for 57?
Deletion in Linear Probing (2)

• Instead of setting bucket 8 to null, place a special marker there

• When lookup encounters marker, it ignores it and continues search
  – What should insert do if it encounters marker?

• Too many markers degrades performance – rehash if there are too many
Primary clustering problem

- **clustering**: nodes being placed close together by probing, which degrades hash table's performance
  - add 89, 18, 49, 58, 9

  - now searching for the value 28 will have to check half the hash table! no longer constant time...
Linear probing – clustering

- No collision
- Collision in small cluster
- Collision in large cluster
Alternative probing strategy

• Primary clustering occurs with linear probing because the same linear pattern:
  – if a slot is inside a cluster, then the next slot must either:
    • also be in that cluster, or
    • expand the cluster

• Instead of searching forward in a linear fashion, consider searching forward using a quadratic function
Quadratic probing

- **quadratic probing**: resolving collisions on slot $i$ by putting the colliding element into slot $i+1$, $i+4$, $i+9$, $i+16$, ...
  - add 89, 18, 49, 58, 9
    - 49 collides (89 is already there), so we search ahead by +1 to empty slot 0
    - 58 collides (18 is already there), so we search ahead by +1 to occupied slot 9, then +4 to empty slot 2
    - 9 collides (89 is already there), so we search ahead by +1 to occupied slot 0, then +4 to empty slot 3
  - what is the lookup algorithm?
Quadratic probing in action

\[
\begin{align*}
\text{hash ( 89, 10 )} & = 9 \\
\text{hash ( 18, 10 )} & = 8 \\
\text{hash ( 49, 10 )} & = 9 \\
\text{hash ( 58, 10 )} & = 8 \\
\text{hash ( 9, 10 )} & = 9
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>After insert 89</th>
<th>After insert 18</th>
<th>After insert 49</th>
<th>After insert 58</th>
<th>After insert 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>49</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
</tbody>
</table>
Quadratic probing

• $f(i) = i^2$

• Probe sequence:
  
  $0^{th}$ probe = $h(x) \mod TableSize$
  
  $1^{th}$ probe = $(h(x) + 1) \mod TableSize$
  
  $2^{th}$ probe = $(h(x) + 4) \mod TableSize$
  
  $3^{th}$ probe = $(h(x) + 9) \mod TableSize$
  
  $\ldots$
  
  $i^{th}$ probe = $(h(x) + i^2) \mod TableSize$
Quadratic probing benefit

- If one of $h + i^2$ falls into a cluster, this does not imply the next one will

- For example, suppose an element was to be inserted in bucket 23 in a hash table with 31 buckets
  - The sequence in which the buckets would be checked is:
    23, 24, 27, 1, 8, 17, 28, 10, 25, 11, 30, 20, 12, 6, 2, 0
Quadratic probing benefit

- Even if two buckets are initially close, the sequence in which subsequent buckets are checked varies greatly
  - Again, with TableSize = 31, compare the first 16 buckets which are checked starting with elements 22 and 23:

```
22  22, 23, 26, 0, 7, 16, 27, 9, 24, 10, 29, 19, 11, 5, 1, 30
23  23, 24, 27, 1, 8, 17, 28, 10, 25, 11, 30, 20, 12, 6, 2, 0
```

- Quadratic probing solves the problem of primary clustering
Quadratic probing drawbacks

• Suppose we have 8 buckets:
  \[ 1^2 \mod 8 = 1, \ 2^2 \mod 8 = 4, \ 3^2 \mod 8 = 1 \]
  – In this case, we are checking bucket \( h(x) + 1 \) twice having checked only one other bucket

• No guarantee that

\[ (h(x) + i^2) \mod TableSize \]
will cycle through 0, 1, ..., \( TableSize - 1 \)
Quadratic probing

• Solution:
  – require that $TableSize$ be prime
  – $(h(x) + i^2) \% TableSize$ for $i = 0, ..., (TableSize - 1)/2$ will cycle through $(TableSize + 1)/2$ values before repeating

• Example with $TableSize = 11$:
  0, 1, 4, 9, 16 ≡ 5, 25 ≡ 3, 36 ≡ 3

• With $TableSize = 13$:
  0, 1, 4, 9, 16 ≡ 3, 25 ≡ 12, 36 ≡ 10, 49 ≡ 10

• With $TableSize = 17$:
  0, 1, 4, 9, 16, 25 ≡ 8, 36 ≡ 2, 49 ≡ 15, 64 ≡ 13, 81 ≡ 13

Note: the symbol ≡ means "$\% TableSize"
Double hashing

• **double hashing**: resolve collisions on slot $i$ by applying a second hash function

• $f(i) = i \times g(x)$
  where $g$ is a second hash function
  – limitations on what $g$ can evaluate to?
  – recommended: $g(x) = R - (x \% R)$, where $R$ prime smaller than $TableSize$

• Psuedocode for double hashing:
  
  ```
  if (table is full) error
  probe = h(value)
  offset = g(value)
  while (table[probe] occupied)
    probe = (probe + offset) % TableSize
  table[probe] = value
  ```
Double Hashing Example

$h(x) = x \mod 7$ and $g(x) = 5 - (x \mod 5)$

<table>
<thead>
<tr>
<th>41</th>
<th>16</th>
<th>40</th>
<th>47</th>
<th>10</th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td>47</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>16</td>
<td>2</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>55</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>40</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
</tbody>
</table>

Probes 1 1 1 2 1 2
Double hashing

- \( f(i) = i \times g(x) \)

- Probe sequence:
  - 0th probe = \( h(x) \mod TableSize \)
  - 1th probe = \( (h(x) + g(x)) \mod TableSize \)
  - 2th probe = \( (h(x) + 2 \times g(x)) \mod TableSize \)
  - 3th probe = \( (h(x) + 3 \times g(x)) \mod TableSize \)
  - \( \ldots \)
  - \( i^{th} \) probe = \( (h(x) + i \times g(x)) \mod TableSize \)
Hashing practice problem

- Draw a diagram of the state of a hash table of size 10, initially empty, after adding the following elements.
  - \( h(x) = x \mod 10 \) as the hash function.
  - Assume that the hash table uses linear probing.

7, 84, 31, 57, 44, 19, 27, 14, and 64
Analysis of linear probing

• the load factor $\lambda$ is the fraction of the table that is full
  
  empty table $\lambda = 0$  
  half full table $\lambda = 0.5$  
  full table $\lambda = 1$

• Assuming a reasonably large table, the average number of buckets examined per insertion (taking clustering into account) is roughly $\frac{1 + \frac{1}{(1-\lambda)^2}}{2}$
  
  • empty table  $\frac{1 + \frac{1}{(1 - 0)^2}}{2} = 1$
  • half full  $\frac{1 + \frac{1}{(1 - 0.5)^2}}{2} = 2.5$
  • 3/4 full  $\frac{1 + \frac{1}{(1 - 0.75)^2}}{2} = 8.5$
  • 9/10 full  $\frac{1 + \frac{1}{(1 - 0.9)^2}}{2} = 50.5$

• as long as the hash function is fair and the table is not too full, then inserting, deleting, and searching are all $O(1)$ operations
Rehashing, hash table size

- **rehash**: increasing the size of a hash table's array, and re-storing all of the items into the array using the hash function
  - can we just copy the old contents to the larger array?
  - When should we rehash? Some options:
    - when load reaches a certain level (e.g., $\lambda = 0.5$)
    - when an insertion fails

- What is the cost (Big-Oh) of rehashing?
- what is a good hash table array size?
  - how much bigger should a hash table get when it grows?
Hashing practice problem

• Draw a diagram of the state of a hash table of size 10, initially empty, after adding the following elements.
  – \( h(x) = x \mod 10 \) as the hash function.
  – Assume that the hash table uses linear probing.
  – Assume that rehashing occurs at the start of an add where the load factor is 0.5.

\[
7, 84, 31, 57, 44, 19, 27, 14, \text{ and } 64
\]

• Repeat the problem above using quadratic probing.