Dictionary ADT and Hash Tables

- Read CLRS § 11.1-4 or Shaffer § 5.4, chap 9.
- Goals:
  - Know the key operations of the Dictionary ADT, and can name its principal implementations.
  - Recognize situations when a Dictionary is required.
  - Able to describe a hash table implementation using *chaining or closed hashing*. 
Scenario: grocery checkout

Requirements:

- Scanners need to map UPC bar code to name and price (look-up).
- Management adds and removes stock numbers.
Scenario: Wi-fi network manager

- OS always listening for nearby SSIDs
- New network names often added or removed.
- If you select a new network, OS must look up its info.
Scenario: login to FaceSpace

- Users of website have a username and password they supply to get access.
- Server checks for match with each login.
- Valid usernames frequently added and removed.
Dictionary Abstract Data Type

- Common themes?
Dictionary Abstract Data Type

Common themes:

- Container of records (keys and satellite data)
- **Add** new keys (would duplicates be ok?)
- **Remove** obsolete keys
- **Look up** info about possible keys (or just test for presence).
One more Container ADT: Dictionary
One more Container ADT: Dictionary
One more Container ADT: Dictionary
Dictionary ADT

- Three typical implementations:
  - Some kind of tree (binary search tree or other)
  - Hash table
  - Sequence (if just a few keys)

- Are duplicate keys allowed? It depends.
  - Often, no. Depends on the application.
  - Not allowed by `java.util.Map` interface

- Are keys stored in sorted order? It depends.
  - `java.util.HashMap`: no
  - `java.util.TreeMap`: yes
Hash Table (chained)

- Store records in a big, semi-empty array (the hash table), denoted $H[0 \ldots M - 1]$.
- Key $k$ maps to an array location (slot) via a hash function $h$. Record is stored in slot $h(k)$.
- Each slot consists of a linked list, storing a chain of zero or more records.
- Typically the hash table is 25% to 50% empty.
Hash Table (chained)

• Grocery store example: each product has a UPC key (a ten-digit integer product ID), and satellite data such as ____ , ____ .
Hash Table (chained)

- Grocery store example: each product has a UPC key (a ten-digit integer product ID), and satellite data such as name, price.
  - 1111068309, “Paprika,” $4.69
  - 2446306116, “Sriracha,” $1.89
  - 1111091444, “Raisins,” $2.25
- Ideally, we could store records in a $M=10000000000$ element array, then index the location by UPC key value.
Hash Table (chained)

- But a 10000000000 element array is too big to be practical.
- Also it is overkill. Store probably stocks only a couple thousand products, tops.
- Instead we will set a reasonable table size of, say, $M=4099$, and define a matching hash function $h(k) = k \mod M$.
- This hash function forces all positive integer keys to map to valid hash table indexes.
Hash Table (chained)

- Examples (for Insert, Find, Delete):
  - \(H[h(2446306116)] = H[2421] \rightarrow \text{("Sriracha,"$1.89)}\)
  - \(H[h(1111068309)] = H[1567] \rightarrow \text{("Paprika,"$4.69)}\)
  - \(H[h(1111091444)] = H[108] \rightarrow \text{("Raisins,"$2.25)}\)

- If you do this long enough, you are bound to map two different keys to the same slot (a collision).
  - UPC code 1111076507 maps to slot 1567 also.
Hash Table (chained)

- Inserting items, size $M = 4099$.
  - Hash table is initially empty: each chain has length zero.
Hash Table (chained)

• Inserting items, size $M = 4099$.
  - Insert record for Sriracha, UPC = 2446306116, cost $1.89$.
  - We use UPC code as the key, so $k = 2446306116$.
  - To insert the record into the table, compute the hash value of the key, $h(k)$.
  - We use $h(k) = k \mod M = 2446306116 \mod 4099$. 

\begin{align*}
\begin{array}{ccccccccccccccc}
\text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} & \text{\textbackslash} \\
\text{0} & \text{1} & \text{2} & \cdots & \text{108} & \cdots & \text{1567} & \cdots & \text{2421} & \cdots & \text{4098} \\
\end{array}
\end{align*}
Hash Table (chained)

- Inserting items, size $M = 4099$.
  - Insert record for Sriracha, UPC = 2446306116, cost $1.89$, into table location 2421.

```
0  1  2  ... 108  ... 1567  ... 2421  ... 4098
```

UPC 2446306116
Sriracha, $1.89$
Hash Table (chained)

• Inserting items, size $M = 4099$.
  - Insert record for Paprika, UPC = 1111068309, cost $4.69$, into table location 1567,
  - . . . because $h(k) = 1111068309 \mod 4099 = 1567$. 
Hash Table (chained)

• Inserting items, size $M = 4099$.
  - Insert record for Raisins, UPC = 1111091444, cost $4.69$, into table location 108,
  - . . . because $h(111109144) = 108$. 
Hash Table (chained)

- Inserting items, size $M = 4099$.
  - Insert record for Styling Gel, UPC = 1111076507, cost $2.99$ into location $h(1111076507) = 1567$.
  - Scan the chain,
  - Add to chain.

```
0  1  2  ...  108  ...  1567  ...  2421  ...  4098
```

- UPC 1111076507
  - Styling Gel, $2.99$
- UPC 1111068309
  - Paprika, $4.69$
- UPC 1111091444
  - Raisins, $2.25$
- UPC 2446306116
  - Sriracha, $1.89$
Hash Table (chained)

- In a chained hash table, when collisions occur, you just add the records to the chain.
- Pseudocode for Insert, Remove, Find?
- With a good hash function, average time per Insertion, Removal, Find operations will be $\Theta(1+n/M)$, where $n$ is the number of records.
- Chained hash tables work well for primary memory and can be very efficient.
Keys and Hash functions

- A hash function must always map key $k$ to the same location $h(k)$, or else Find (etc.) won't work.
  - It must be perfectly repeatable (not random).
- An ideal hash function is likely to distribute any two distinct keys $k_1$, $k_2$ to different slots, even if the keys are close in value.
  - We want it to spread the records evenly across the table (“randomly”).
- All kinds of keys (strings, reals) can be hashed.
Hash Tables

- One common setup is to choose a table of prime size $M$, and hash integers modulo $M$.
  - Many more variations on good choices of hash function and table size.
- If the table fills up too much (if it holds $n$ records, with $n/M > 0.5$), you might need to enlarge the table, and re-insert all records.
Closed Hashing

- Another way to set up a hash table is to store records directly in the table, instead of chains.
- What to do about collisions? Good question.
Closed Hashing Example

- Initially empty table
- “Empty” sentinel value

<table>
<thead>
<tr>
<th></th>
<th>UPC = -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1567</td>
<td></td>
</tr>
<tr>
<td>1568</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2421</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>4098</td>
<td></td>
</tr>
</tbody>
</table>
Closed Hashing Example

• Insert Sriracha:
  \[ k = 2446306116, \]
  \[ h(k) = 2421. \]
That cell was empty, so,
copy record into \( H[2421] \).

<table>
<thead>
<tr>
<th>UPC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UPC = -1</td>
</tr>
<tr>
<td>1</td>
<td>UPC = -1</td>
</tr>
<tr>
<td>2</td>
<td>UPC = -1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>108</td>
<td>UPC = -1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1567</td>
<td>UPC = -1</td>
</tr>
<tr>
<td>1568</td>
<td>UPC = -1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2421</td>
<td>UPC 2446306116 Sriracha, $1.89</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>4098</td>
<td>UPC = -1</td>
</tr>
</tbody>
</table>
Closed Hashing Example

- Insert Paprika:
  
  \[ k = 1111068309, \]
  \[ h(k) = 1567. \]
  
  That cell was empty, so,
  
  copy record into \( H[1567] \).

<table>
<thead>
<tr>
<th>UPC</th>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UPC = -1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>UPC = -1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UPC = -1</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>UPC = -1</td>
<td></td>
</tr>
<tr>
<td>1567</td>
<td>UPC 1111068309 Paprika, $4.69</td>
<td></td>
</tr>
<tr>
<td>1568</td>
<td>UPC = -1</td>
<td></td>
</tr>
<tr>
<td>2421</td>
<td>UPC 2446306116 Sriracha, $1.89</td>
<td></td>
</tr>
<tr>
<td>4098</td>
<td>UPC = -1</td>
<td></td>
</tr>
</tbody>
</table>
Closed Hashing Example

- Insert Raisins:
  \[ k = 1111091444, \]
  \[ h(k) = 108. \]
  That cell was empty, so, copy record into \( H[108] \).
Closed Hashing Example

• Insert Styling Gel:
  
  \[ k = 1111076507, \]
  \[ h(k) = 1567. \]
  
  That cell is not empty!

• What to do??

<table>
<thead>
<tr>
<th>UPC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UPC = -1</td>
</tr>
<tr>
<td>1</td>
<td>UPC = -1</td>
</tr>
<tr>
<td>2</td>
<td>UPC = -1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
| 108 | UPC 1111091444  
     | Raisins, $2.25 |
| ... | ...         |
| 1567| UPC 1111068309  
     | Paprika, $4.69 |
| 1568| UPC = -1    |
| ... |             |
| 2421| UPC 2446306116  
     | Sriracha, $1.89 |
| ... |             |
| 4098| UPC = -1    |
Closed Hashing Example

- Insert Styling Gel:
  \( k = 1111076507, \)
  \( h(k) = 1567. \)
  That cell is not empty!

- What to do?
  - Most basic strategy:
  - “Linear probing,” i.e.,
  - find next empty spot.

<table>
<thead>
<tr>
<th>UPC</th>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111091444</td>
<td>Raisins, $2.25</td>
<td></td>
</tr>
<tr>
<td>1111068309</td>
<td>Paprika, $4.69</td>
<td></td>
</tr>
<tr>
<td>2446306116</td>
<td>Sriracha, $1.89</td>
<td></td>
</tr>
</tbody>
</table>
### Closed Hashing Example

- **Insert Styling Gel:**
  
  \[ k = 1111076507, \]
  
  \[ h(k) = 1567. \]
  
  That cell is not empty!

- **What to do?**
  - Most basic strategy:
  - "Linear probing," i.e.,
  - find next empty spot.

<table>
<thead>
<tr>
<th>UPC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>108</th>
<th>1567</th>
<th>1568</th>
<th>2421</th>
<th>4098</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raisins,</td>
<td></td>
<td></td>
<td></td>
<td>1111091444</td>
<td></td>
<td>1111068309</td>
<td></td>
<td>2446306116</td>
</tr>
<tr>
<td>$2.25</td>
<td></td>
<td></td>
<td></td>
<td>Raisins, $2.25</td>
<td></td>
<td>Paprika, $4.69</td>
<td></td>
<td>Sriracha, $1.89</td>
</tr>
<tr>
<td>Paprika,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4.69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Styling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gel, $2.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Closed Hashing

• Another way to set up a hash table is to store records directly in the table, instead of chains.

• What to do about collisions?
  - _Linear probing_: search for the next empty slot.
  - Leads to _primary clustering_ (a large, growing blob of full slots), which can hurt performance.
  - In general, can create a fancy _probe function_ \( p(k,i) \) that guides you where to try on the \( i \)-th probe. Example: quadratic function \( p(k,i) = i^2 \).
Closed Hashing and Deletion

- Your Dictionary application might need to perform deletion.
- What could go wrong if you just erase the record from a closed hash table, making the slot empty again?
Closed Hashing and Deletion

- Your Dictionary application might need to perform deletion.
- What could go wrong if you just erase the record from a closed hash table, making the slot empty again?
  - An empty slot marks the end of a probe sequence.
  - You might break a probe sequence that leads to some other record. That would make other records inaccessible.
- Example: if you delete Paprika, you can't find the Gel.
Closed Hashing and Deletion

• Solution?
Closed Hashing and Deletion

- **Solution?**
  - When we delete a record, we mark the slot with a “tombstone” status.
  - Each slot has status in \{empty, busy, tombstone\}.
  - Find operation should not stop on a tombstone.
  - Insert operation can reuse a tombstone slot.

- **Only guaranteed way to get rid of them is to restructure the table.**
Hashing Summary

• Fast way to implement dictionaries in memory (chaining) or on disk (closed hashing).
  – Nearly constant-time operations, if all is well.
  – If the table gets too full, performance gets worse.
• Records are not stored in any particular order.
• Needs some engineering judgment, to choose proper size and a good hash function.