The Web is a Graph

- Directed Graph of Nodes and Arcs
  - Nodes = web pages
  - Arcs = hyperlinks from a page to another
- A graph can be explored
- A graph can be indexed

Creating Word Index: Counting Frequencies

```
i am sam
i am sam
i am sam
sam i am
that sam i am
do not like
do not like
do not like
do not like
that sam i am
i do not like
green eggs and ham
would you like them
here or there
i would not like them
in a house
i would not like them
in a house
i would not like them
in a boat
i would not like them
in a car
i would not like them
in a box
i would not like them
in a box
i would not like them
in a box
i would not like them
in a box
```

```
i do not like them
in a house
i do not like them
with a mouse
i do not like them
here or there
i do not like them
anywhere
i do not like them
green eggs and ham
would you like them
here or there
i would not like them
in a house
i would not like them
in a house
```

```
try 4
will 21
with 19
would 16
you 34
```

How Google (and the other search engines) Work

- Crawl the web
- Create word index
- Rank results
- Search engine servers
- Index & Freq's

Challenges in counting English words

- The English language has half-a-million terms
- Any given text, however, has only a few thousand words
- Keeping around an array of 500K words "just in case" is not a good idea
- Hashing is the concept that order is determined by some function of the value of the element to be stored
- Like throwing darts on a board...
Let’s play darts (aka: let’s “hash the keys”)

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brian</td>
<td>1</td>
</tr>
<tr>
<td>Stella</td>
<td>5</td>
</tr>
<tr>
<td>Ellen</td>
<td>4</td>
</tr>
<tr>
<td>Takis</td>
<td>6</td>
</tr>
<tr>
<td>Mark</td>
<td>12</td>
</tr>
<tr>
<td>Lyn</td>
<td>11</td>
</tr>
</tbody>
</table>
```

Hash Table

Any problems with our hash?

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brian</td>
<td>1</td>
</tr>
<tr>
<td>Stella</td>
<td>5</td>
</tr>
<tr>
<td>Ellen</td>
<td>4</td>
</tr>
<tr>
<td>Takis</td>
<td>6</td>
</tr>
<tr>
<td>Mark</td>
<td>12</td>
</tr>
<tr>
<td>Lyn</td>
<td>11</td>
</tr>
<tr>
<td>Sohie</td>
<td>?</td>
</tr>
</tbody>
</table>
```

Pros and Cons

Pros
- Searching: $O(1)$
- Inserting: $O(1)$
- Deleting: $O(1)$

Cons
- You cannot keep adding new elements for ever!
  - Table size is fixed (like an array)
  - Needs expansion capabilities ( $O(1)$ )
- Would be nice to have a perfect hashing function but many items may end up on same location
- Collisions need resolution policy

What are the Pros and Cons of using Hashing?
Load Factor: When M is large enough?

- \( \frac{N}{M} = \text{load factor} \) of a hashtable
- number of entries \( N \) in table
- divided by the table capacity \( M \).

Heuristics:
- If you know \( N \), make \( M = 1.5 \times N \)
- If you do not know \( N \), provide for dynamic resizing
- Create larger Hash Table
  - Insert old elements into new

Hash Functions: Division

- Good:
  \[ h(\text{hashCode}) = \text{hashCode} \mod M \]
  
  \( M \): prime

- Better:
  \[ h(\text{hashCode}) = ((a \times \text{hashCode} + b) \mod p) \mod M \]
  
  \( p \): prime >> \( N \)
  
  \( a, b \): positive integers

Hashing Functions: Mid-square

- The key is multiplied by itself and then
  "extract" some digits from the middle of the result
- For example, if our key is 4321
  - Multiply the key by itself yielding 18671041
  - Extract the needed three digits
- It is critical that the same three digits by extract each time
- We may also extract bits and then reconstruct an index from the bits

Even Object has its own hashing function in Java!

- The java.lang.Object class defines a method called hashcode
  that returns an integer based on the memory location of the object
  - This is generally not very useful
- Classes derived from Object often override the inherited definition of hashcode
  to provide their own version
- For example,
  String and Integer define their own hashcode methods
  - These more specific hashcode functions are more effective
Resolving Collisions

- If we are able to develop a perfect hashing function, then we do not need to be concerned about collisions or table size.
- However, often we do not know the size of the dataset and are not able to develop a perfect hashing function.
- In these cases, we must decide how to handle collisions.

Resolving Collisions idea #1: Separate Chaining

Separate Chaining

- Brian hash("Brian") = 1
- Stella
- Ellen
- Lyn
- Mark
- Takis

Separate Chaining

- Brian hash("Brian") = 1
- Stella hash("Stella") = 4
- Ellen
- Lyn
- Mark
- Takis
Separate Chaining

| Brian |hash('Brian')| 1 |
| Stella |hash('Stella')| 4 |
| Ellen |hash('Ellen')| 4 |
| Lyn |
| Mark |
| Tokis |

Separate Chaining

| Brian |hash('Brian')| 1 |
| Stella |hash('Stella')| 4 |
| Ellen |hash('Ellen')| 4 |
| Lyn |hash('Lyn')| 4 |
| Mark |

Resolving Collisions idea #2: Open Addressing

Look for another open position in the table other than the one to which the element is hashed.

• Open addressing \((M >> N)\):

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
\hline
0 & | & | & \\
1 & | & | & \\
2 & | & | & \\
3 & | & | & \\
\end{array}
\]

• How are collisions resolved with this technique?
Resolve Open Addressing Collisions with Linear Probing

- When the index hashed to is occupied by a stranger, probe the next position.
- If that position is empty, we insert the entry, otherwise, we probe the next position and repeat.

```
H A S H I N G I S F U N
8 1 19 8 9 14 7 9 19 6 21 14
0 1 2 3 4 5 6 7 8 9 10
[ ] [ ] [ ] [ ] [ ] [ ]
```

There is a problem though: Clustering

- As the table begins to fill up, more and more entries must be examined before the desired entry is found.
- Insertion of one entry may greatly increase the search time for others.

```
H A S H I N G I S F U N
8 1 19 8 9 14 7 9 19 6 21 14
0 1 2 3 4 5 6 7 8 9 10
[ ] [ ] [ ] [ ] [ ] [ ]
```

The Java Hashtable<K,V> Class

- Located in java.util
- Methods
  - int size() // returns number of keys in table
  - V get(Object key) // returns value to which specified key is mapped in table
  - V put(K key, V value) // maps key to specified value in table
  - boolean containsKey(Object key) // tests if the specified Object is a key in hash table
  - V remove(Object key) // removes key and corresponding value from table
  - ...

```
import java.util.*;

public static void main (String[] args) throws IOException {
    Hashtable wordTable = new Hashtable();
    // set up variables for reading words from a text file
    FileReader fr = new FileReader("GreenEggs.txt");
    BufferedReader br = new BufferedReader(fr);
    StreamTokenizer textwords = new StreamTokenizer(br);
    // process words from the text file
    while (textwords.nextToken() != StreamTokenizer.TT_EOF) {
        String word = textwords.sval; // get the next word
        if (wordTable.containsKey(word)) { // check if word is already in table
            int num = ((Integer)wordTable.get(word)).intValue();
            wordTable.put(word, new Integer(num+1));
        } else { // word is new, so add new entry to table
            wordTable.put(word, new Integer(1));
        }
    }
    // System.out.println(wordTable);
    fr.close();
}
```

Basic Word Frequency code
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new()</td>
<td>Constructs a new empty hash table with a default initial capacity of one, and head size of two, which is 3.</td>
</tr>
<tr>
<td>get(key)</td>
<td>Returns the value currently associated with the specified key, or nil if the key is not found.</td>
</tr>
<tr>
<td>contains()</td>
<td>Returns true if the specified key is in the hash table, false otherwise.</td>
</tr>
<tr>
<td>remove()</td>
<td>Removes the key from the hash table if it is present. Returns false otherwise.</td>
</tr>
<tr>
<td>size()</td>
<td>Returns the number of elements in the hash table.</td>
</tr>
<tr>
<td>total_size()</td>
<td>Returns the total number of slots in the hash table.</td>
</tr>
</tbody>
</table>

**Rehashing:**

- The initial capacity and head size are both doubled whenever the number of elements reaches one-third of the capacity, or when the number of elements is four times the head size.

- The new capacity is a power of two, which is less than one-third of the total capacity. The number of elements is preserved, so the table is fully used after a rehash.

- On rehash, elements are added to the new table, starting with the smallest head size. The items are then reinserted into the new table, starting with the smallest head size again.

- This process continues until the rehash is complete, or the target capacity is reached.

- The rehash process is recursive, with the number of items passed to the rehash function.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.

- The rehash function is called recursively until the number of items is reduced to zero, indicating that the table is fully rehashed.

- The rehash function uses the hash function, which takes the key as input and returns an integer as output.

- The hash function is applied to the key, and the result is used to determine the index in the table where the item will be placed. This index is calculated by taking the remainder of the hash value divided by the total capacity.

- If the index is already occupied, the function checks if the item is found and returns the corresponding value. Otherwise, the item is added to the table at the specified index.

- The function continues until all items have been processed, and the table has been fully rehashed.