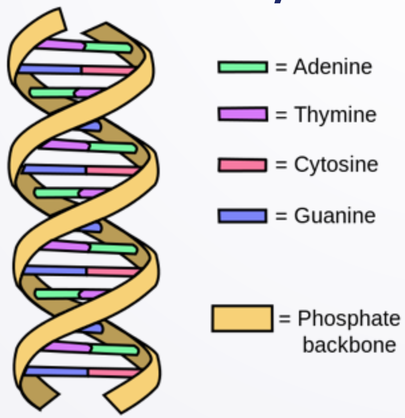
Using and Improving GladLibs

HashMap

Duke
UNIVERSITY

New Structures

- GladLib program works!
  - Design flaws, but can be modified
- Counting frequencies, 'CGAT' to 'AB..YZ'
  - Extended to words with two ArrayLists
  - From 4 to 26 to 5,280* counters!



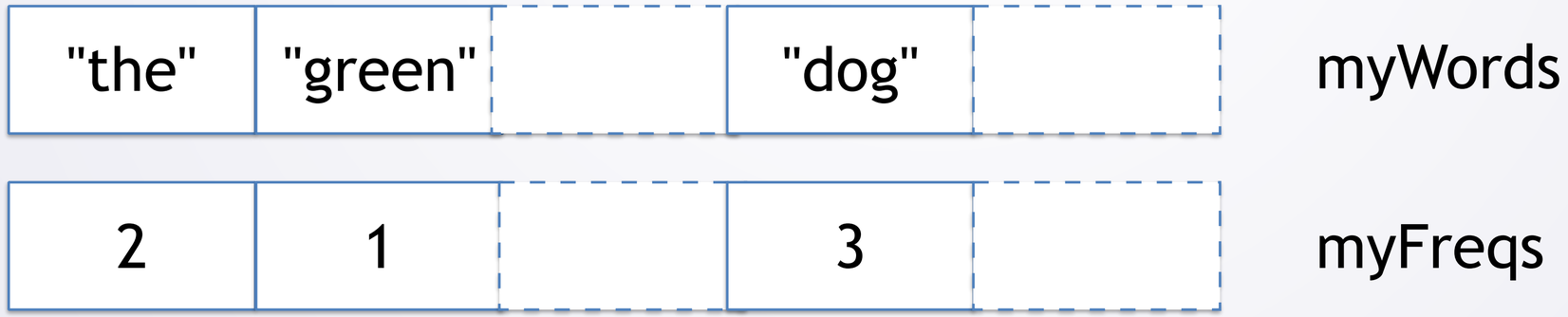
DNA

New Structures

- GladLib program works!
  - Design flaws, but can be modified
- Counting frequencies, 'CGAT' to 'AB..YZ'
  - Extended to words with two ArrayLists
  - From 4 to 26 to 5,280* counters!

New Structures

- GladLib program works!
  - Design flaws, but can be modified
- Counting frequencies, 'CGAT' to 'AB..YZ'
  - Extended to words with two ArrayLists
  - From 4 to 26 to 5,280* counters!



New Structures

- GladLib program works!
  - Design flaws, but can be modified
- Counting frequencies, 'CGAT' to 'AB..YZ'
  - Extended to words with two ArrayLists
  - From 4 to 26 to 5,280* counters!
- From parallel ArrayLists to HashMap
  - Code in GladLib easier to modify
  - Much faster to count word frequencies

Starting with Parallel ArrayLists

- Seen code using parallel arrays to count word occurrences, calls **`.indexOf(s)`**

```
public void findUnique(){
        FileResource resource = new FileResource();
        for(String s : resource.words()){
            s = s.toLowerCase();
            int index = myWords.indexOf(s);
            if (index == -1){
                myWords.add(s);
                myFreqs.add(1);
            }
            else {
                int freq = myFreqs.get(index);
                myFreqs.set(index,freq+1);
            }
        }
 }
```

Starting with Parallel ArrayLists

- Seen code using parallel arrays to count word occurrences, calls **`.indexOf(s)`**

```
public void findUnique(){
        FileResource resource = new FileResource();
        for(String s : resource.words()){
            s = s.toLowerCase();
            int index = myWords.indexOf(s);
            if (index == -1){
                myWords.add(s);
                myFreqs.add(1);
            }
            else {
                int freq = myFreqs.get(index);
                myFreqs.set(index,freq+1);
            }
        }
 }
```

Starting with Parallel ArrayLists

- Seen code using parallel arrays to count word occurrences, calls **`.indexOf(s)`**

```
public void findUnique(){
        FileResource resource = new FileResource();
        for(String s : resource.words()){
            s = s.toLowerCase();
            int index = myWords.indexOf(s);
            if (index == -1){
                myWords.add(s);
                myFreqs.add(1);
            }
            else {
                int freq = myFreqs.get(index);
                myFreqs.set(index,freq+1);
            }
        }
 }
```

Starting with Parallel ArrayLists

- Seen code using parallel arrays to count word occurrences, calls **`.indexOf(s)`**

```
public void findUnique(){
        FileResource resource = new FileResource();
        for(String s : resource.words()){
            s = s.toLowerCase();
            int index = myWords.indexOf(s);
            if (index == -1){
                myWords.add(s);
                myFreqs.add(1);
            }
            else {
                int freq = myFreqs.get(index);
                myFreqs.set(index,freq+1);
            }
        }
 }
```

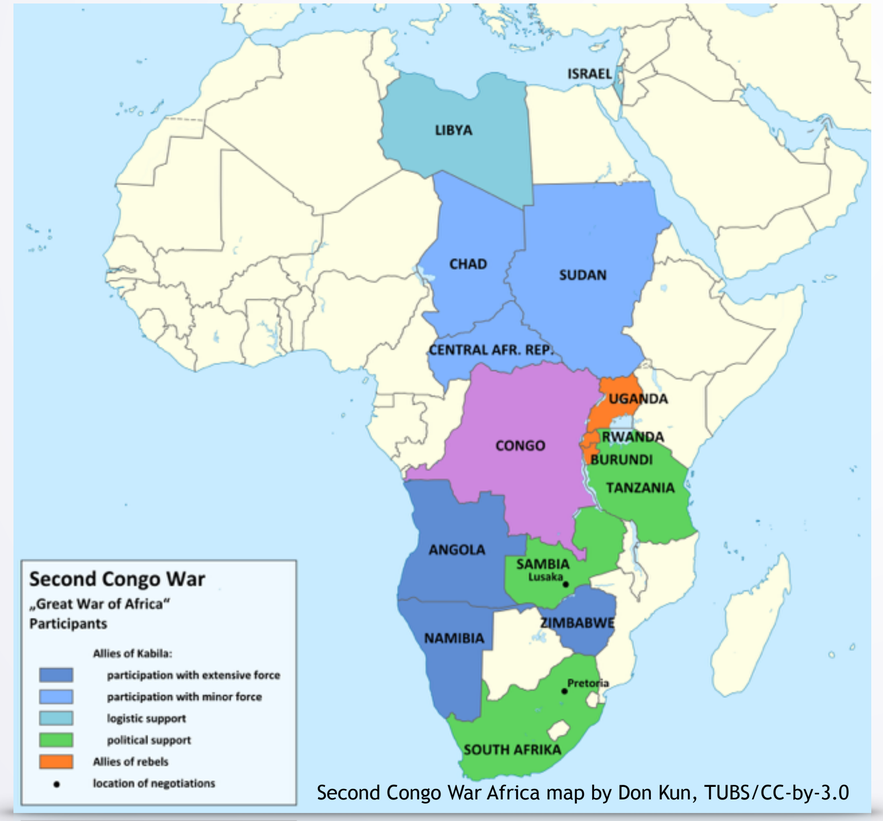
Starting with Parallel ArrayLists

- Seen code using parallel arrays to count word occurrences, calls **`.indexOf(s)`**

```
public void findUnique(){
        FileResource resource = new FileResource();
        for(String s : resource.words()){
            s = s.toLowerCase();
            int index = myWords.indexOf(s);
            if (index == -1){
                myWords.add(s);
                myFreqs.add(1);
            }
            else {
                int freq = myFreqs.get(index);
                myFreqs.set(index,freq+1);
            }
        }
 }
```

Motivating HashMap

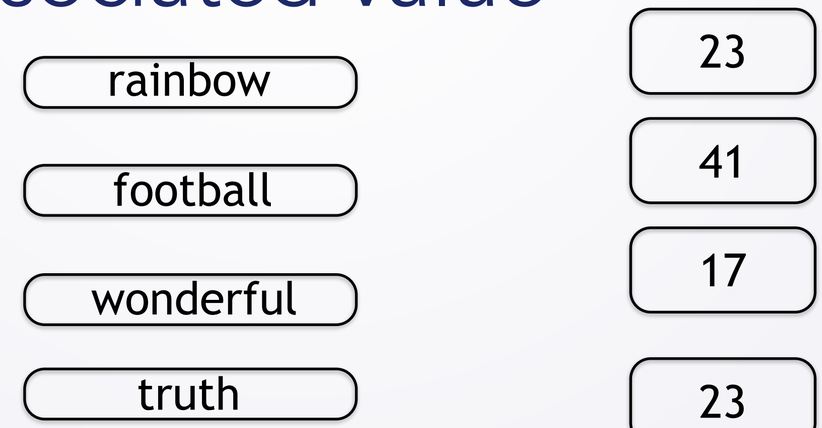
- A HashMap is a class that associates keys with values, generally called a map



Second Congo War Africa map by Don Kun, TUBS/CC-by-3.0

Motivating HashMap

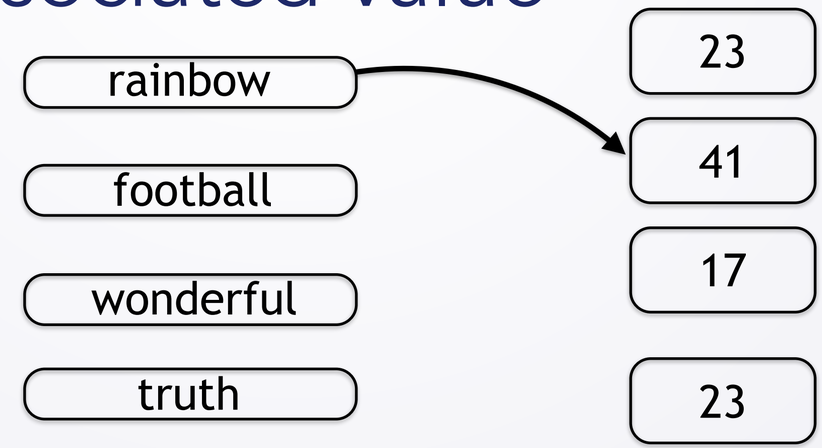
- A HashMap is a class that associates keys with values, generally called a map
  - More mathematical than geographical
  - Key is element in domain, value is what key maps to in range
- Look up key, get associated value



Motivating HashMap

- A HashMap is a class that associates keys with values, generally called a map
  - More mathematical than geographical
  - Key is element in domain, value is what key maps to in range
- Look up key, get associated value

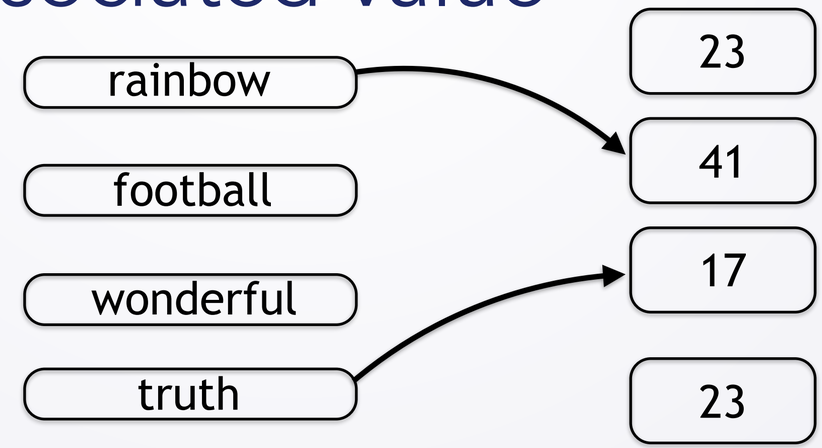
```
map.get("rainbow")
```



Motivating HashMap

- A HashMap is a class that associates keys with values, generally called a map
  - More mathematical than geographical
  - Key is element in domain, value is what key maps to in range
- Look up key, get associated value

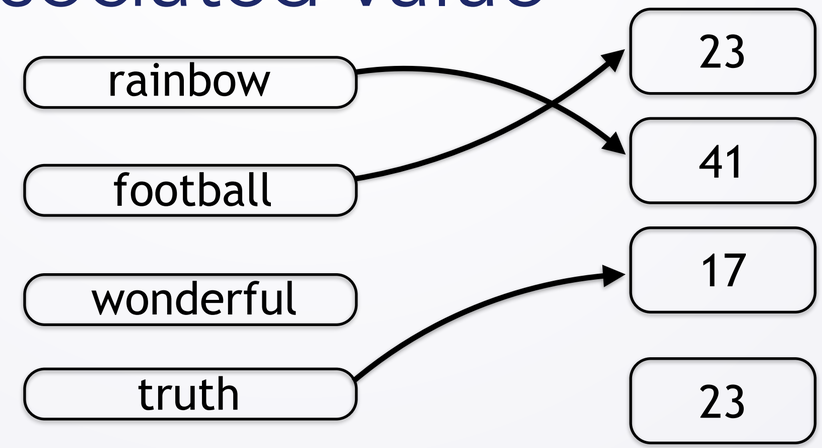
```
map.get("rainbow")
map.get("truth")
```



Motivating HashMap

- A HashMap is a class that associates keys with values, generally called a map
  - More mathematical than geographical
  - Key is element in domain, value is what key maps to in range
- Look up key, get associated value

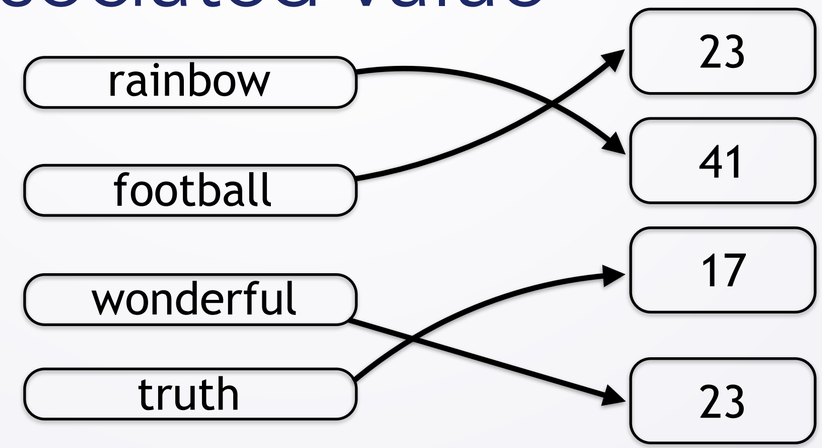
```
map.get("rainbow")
map.get("truth")
map.get("football")
```



Motivating HashMap

- A HashMap is a class that associates keys with values, generally called a map
  - More mathematical than geographical
  - Key is element in domain, value is what key maps to in range
- Look up key, get associated value

```
map.get("rainbow")
map.get("truth")
map.get("football")
map.get("wonderful")
```



Updating Values in HashMap

- One HashMap replaces two ArrayLists

```
public void findUnique(){
        FileResource resource = new FileResource();
        for(String s : resource.words()){
            s = s.toLowerCase();
            int index = myWords.indexOf(s);
            if (index == -1){
                myWords.add(s);
                myFreqs.add(1);
            }
            else {
                int freq = myFreqs.get(index);
                myFreqs.set(index,freq+1);
            }
        }
 }
```

Updating Values in HashMap

- One HashMap replaces two ArrayLists
  - Key is String, associated value is Integer

```
public void countWordsMap(){
    FileResource resource = new FileResource();
    HashMap<String,Integer> map = new HashMap<String,Integer>();

    for(String w : resource.words()){
      w = w.toLowerCase();
      if (!map.containsKey(w)){
        map.put(w,1);
      }
      else {
        map.put(w,map.get(w)+1);
      }
}
```

Updating Values in HashMap

- One HashMap replaces two ArrayLists
  - Key is String, associated value is Integer

```
public void countWordsMap(){
    FileResource resource = new FileResource();
    HashMap<String,Integer> map = new HashMap<String,Integer>();

    for(String w : resource.words()){
      w = w.toLowerCase();
      if (!map.containsKey(w)){
        map.put(w,1);
      }
      else {
        map.put(w,map.get(w)+1);
      }
}
```

Updating Values in HashMap

- One HashMap replaces two ArrayLists
  - Key is String, associated value is Integer

```
public void countWordsMap(){
    FileResource resource = new FileResource();
    HashMap<String,Integer> map = new HashMap<String,Integer>();

    for(String w : resource.words()){
      w = w.toLowerCase();
      if (!map.containsKey(w)){
        map.put(w,1);
      }
      else {
        map.put(w,map.get(w)+1);
      }
}
```

Updating Values in HashMap

- One HashMap replaces two ArrayLists
  - Key is String, associated value is Integer
  - Is key new, unseen? put value 1, else update

```
public void countWordsMap(){
    FileResource resource = new FileResource();
    HashMap<String,Integer> map = new HashMap<String,Integer>();

    for(String w : resource.words()){
      w = w.toLowerCase();
      if (!map.containsKey(w)){
        map.put(w,1);
      }
      else {
        map.put(w,map.get(w)+1);
      }
}
```

Updating Values in HashMap

- One HashMap replaces two ArrayLists
  - Key is String, associated value is Integer
  - Is key new, unseen? put value 1, else update

```
public void countWordsMap(){
    FileResource resource = new FileResource();
    HashMap<String,Integer> map = new HashMap<String,Integer>();

    for(String w : resource.words()){
      w = w.toLowerCase();
      if (!map.containsKey(w)){
        map.put(w,1);
      }
      else {
        map.put(w,map.get(w)+1);
      }
}
```

Updating Values in HashMap

- One HashMap replaces two ArrayLists
  - Key is String, associated value is Integer
  - Is key new, unseen? put value 1, else update

```
public void countWordsMap(){
    FileResource resource = new FileResource();
    HashMap<String,Integer> map = new HashMap<String,Integer>();

    for(String w : resource.words()){
      w = w.toLowerCase();
      if (!map.containsKey(w)){
        map.put(w,1);
      }
      else {
        map.put(w,map.get(w)+1);
      }
}
```

Accessing All Values in Map

- Printing all values in parallel arrays uses for loop with index accessing word and freq

```
public void printWords(){
    for(int k=0; k < myFreqs.size(); k++){
      System.out.println(myFreqs.get(k)+"\t"+myWords.get(k));
}
```

Accessing All Values in Map

- Printing all values in parallel arrays uses for loop with index accessing word and freq
- Printing all values in map requires looping over keys, get value associated with key

```
public void printWords(){
    for(String s : myMap.keySet()){
      System.out.println(myMaps.get(s)+"\t"+s);
}
```

Accessing All Values in Map

- Printing all values in parallel arrays uses for loop with index accessing word and freq
- Printing all values in map requires looping over keys, get value associated with key
  - Iterable **`.keySet()`**, similar to **`.words()`** or **`.lines()`** or **`.data()`**

```
public void printWords(){
    for(String s : myMap.keySet()){
      System.out.println(myMaps.get(s)+"\t"+s);
}
```

Accessing All Values in Map

- Printing all values in parallel arrays uses for loop with index accessing word and freq
- Printing all values in map requires looping over keys, get value associated with key
  - Iterable **`.keySet()`**, similar to **`.words()`** or **`.lines()`** or **`.data()`**

```
public void printWords(){
    for(String s : myMap.keySet()){
      System.out.println(myMaps.get(s)+"\t"+s);
}
```

Maps are Very Efficient!

- When files are large, efficiency matters

	Total Words	*Different Words*	*Time ArrayList*	*Time HashMap*
Julius Caesar	*20,869*	*4,443*	*0.25*	*0.03*
Confucius	*34,582*	*6,558*	*0.4*	*0.05*
Scarlet Letter	*85,754*	*13,543*	*1.3*	*0.1*
King James Bible	*823,135*	*32,675*	*20.8*	*0.48*

Maps are Very Efficient!

- When files are large, efficiency matters

	Total Words	*Different Words*	*Time ArrayList*	*Time HashMap*
Julius Caesar	*20,869*	*4,443*	*0.25*	*0.03*
Confucius	*34,582*	*6,558*	*0.4*	*0.05*
Scarlet Letter	*85,754*	*13,543*	*1.3*	*0.1*
King James Bible	*823,135*	*32,675*	*20.8*	*0.48*

Maps are Very Efficient!

- When files are large, efficiency matters

	Total Words	*Different Words*	*Time ArrayList*	*Time HashMap*
Julius Caesar	*20,869*	*4,443*	*0.25*	*0.03*
Confucius	*34,582*	*6,558*	*0.4*	*0.05*
Scarlet Letter	*85,754*	*13,543*	*1.3*	*0.1*
King James Bible	*823,135*	*32,675*	*20.8*	*0.48*

Maps are Very Efficient!

- When files are large, efficiency matters

	Total Words	*Different Words*	*Time ArrayList*	*Time HashMap*
Julius Caesar	*20,869*	*4,443*	*0.25*	*0.03*
Confucius	*34,582*	*6,558*	*0.4*	*0.05*
Scarlet Letter	*85,754*	*13,543*	*1.3*	*0.1*
King James Bible	*823,135*	*32,675*	*20.8*	*0.48*

Maps are Very Efficient!

- When files are large, efficiency matters

	Total Words	*Different Words*	*Time ArrayList*	*Time HashMap*
Julius Caesar	*20,869*	*4,443*	*0.25*	*0.03*
Confucius	*34,582*	*6,558*	*0.4*	*0.05*
Scarlet Letter	*85,754*	*13,543*	*1.3*	*0.1*
King James Bible	*823,135*	*32,675*	*20.8*	*0.48*

Maps are Very Efficient!

- When files are large, efficiency matters
  - Look up in map is independent of number of keys! ArrayList requires looking at all elements

	Total Words	*Different Words*	*Time ArrayList*	*Time HashMap*
Julius Caesar	*20,869*	*4,443*	*0.25*	*0.03*
Confucius	*34,582*	*6,558*	*0.4*	*0.05*
Scarlet Letter	*85,754*	*13,543*	*1.3*	*0.1*
King James Bible	*823,135*	*32,675*	*20.8*	*0.48*