CS 106B, Lecture 7
Sets, Maps, and Lexicons
Today’s Topics

• Sets (and Lexicons)
  – A new kind of ADT
  – countUniqueWords redux

• Maps
  – An ADT for pairs of data
  – wordCount example
  – Where2Eat
ADT Soup

Start

How many dimensions of data do I have?

Two

Grid

One

Which elements do I need to access?

Frequent looping or middle elements

Vector

First element

Queue

Last element

Stack
CountUniqueWords

• One basic statistic about a text is the number of unique words it has
  – Linguists and computer scientists frequently start analysis with the number of unique words
  – Good indication of vocabulary

• Problem: how can we determine the number of unique words in a file?

File? ladygaga.txt
There are 774 unique words in ladygaga.txt
Sets
Sets

- Only answers question of membership
  - No duplicates
- Operations
  - contains($elem$)
  - add($elem$)
  - remove($elem$)
- Comparison to Vector
  - Does not maintain order
  - No duplicates
  - Really fast at finding membership
Looping over Sets

• Sets don’t have indices, so we use a for-each loop
• Iterates in sorted order (alphabetical order for strings)
• Can’t edit while we iterate

Set<string> friends;
friends.add("Shreya");
friends.add("Leland");

// prints in alphabetical order
for (string myFriend : friends) {
    cout << "Hi, " << myFriend << endl;
    cout << "Let's get dinner." << endl;
}
Good Operations to Know

- **Union**: set1 + set2
- **Difference**: set1 - set2
- **Intersection**: set1 * set2
## Sets – Method List

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.add(value)</code></td>
<td>$O(\log N)$</td>
<td>Adds an element to this set, if it was not already there</td>
</tr>
<tr>
<td><code>s.clear()</code></td>
<td>$O(N)$</td>
<td>Removes all elements from this set</td>
</tr>
<tr>
<td><code>s.contains(value)</code></td>
<td>$O(\log N)$</td>
<td>Returns <code>true</code> if <code>value</code> is in this set</td>
</tr>
<tr>
<td><code>s.equals(set)</code></td>
<td>$O(N)$</td>
<td>Returns <code>true</code> if the two sets contain the same elements</td>
</tr>
<tr>
<td><code>s.first()</code></td>
<td>$O(\log N)$</td>
<td>Returns the first value in the set in order</td>
</tr>
<tr>
<td><code>s.isEmpty()</code></td>
<td>$O(1)$</td>
<td>Returns <code>true</code> if the set contains no elements</td>
</tr>
<tr>
<td><code>s.isSubsetOf(s2)</code></td>
<td>$O(N)$</td>
<td>Returns <code>true</code> if all the elements in the set are also in <code>s2</code></td>
</tr>
<tr>
<td><code>s.remove(value)</code></td>
<td>$O(\log N)$</td>
<td>Removes an element from this set</td>
</tr>
<tr>
<td><code>s.size()</code></td>
<td>$O(1)$</td>
<td>Returns the number of elements in this set</td>
</tr>
<tr>
<td><code>s.toString()</code></td>
<td>$O(N)$</td>
<td>Converts the set to a printable string representation</td>
</tr>
</tbody>
</table>
Lexicons

- Set where the only type is string
- Can do everything a Set does
- Also answers the question – do any words start with this prefix?
  - `lexicon.containsPrefix(prefix)`
- Used to store dictionaries
- We’ll talk about lexicons more later
ADT Soup Expanded

Start

How many dimensions of data do I have?

One

Two

Grid

Do I only care about membership?

No - need duplicates or order

Which elements do I need to access?

Yes

Set

Frequent looping or middle elements

First element

Vector

Last element

Queue

Stack
Announcements

• Assignment 1 due **today at 5PM**
• Assignment 2 comes out today, due **Wednesday, July 11 at 5PM**
• Don't forget to answer the debugging questions in LaIR
Maps
Maps

- **Stores pairs of information**
  - First half of the pair is called a **key**, and the second half is the associated **value**
  - Find a value by looking up its associated key
  - Useful when you will only have half the information available later in the program
  - Keys must be unique (just like elements in a Set!)

- **Comparison with Vector**
  - Vectors look up elements by **index**, Maps look them up by **key**
  - Need to have two types (for the key and the value)
  - Ordered by key, not index
Map Syntax

• `map.put(key, value)`
  – `map[key] = value`
  – Adds the key if it wasn’t already in the map
  – Otherwise edits its value

• `map.get(key)`
  – `map[key]`
    • This alternate syntax will create a key with the `default` value in the map

• `map.remove(key)`
  – No effect if the key isn’t in the map
Map Example: Dictionary

```cpp
ifstream file;
promptUserForFile(file, "Where is your dictionary?");
Map<string, string> dictionary;
string word;

while (getline(file, word)) {
    string definition;
    getline(file, definition);
    dictionary[word] = definition;
}

while (true) {
    string query = getLine("Word to look up?");
    if (dictionary.containsKey(query)) {
        cout << "The definition is " << dictionary[query] << endl;
    } else {
        cout << "I don't know that word!" << endl;
    }
}
```
Looping over Maps

- Maps also don’t have indices, so we use a for-each loop over the keys
- Iterates in sorted order over the keys (alphabetical order for strings)
- Can’t edit the keys while we iterate (can edit values)

Map<string, int> phonebook;
phonebook["Ashley"] = 5551234;
phonebook["Shreya"] = 5559876;

// prints in alphabetical order
for (string name: phonebook) {
    int phoneNumber = phonebook[name];
    cout << "I’m going to call " << name;
    cout << " at " << phoneNumber << endl;
}
We’ve found the number of unique words in a file. Another statistic is how frequently each word is used.

Given a text file and a user-inputted word, how frequently is that word used in the file?

<table>
<thead>
<tr>
<th>File? tiny.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word? to</td>
</tr>
<tr>
<td>&quot;to&quot; appears 2 times</td>
</tr>
<tr>
<td>Word? or</td>
</tr>
<tr>
<td>&quot;or&quot; appears 1 times</td>
</tr>
</tbody>
</table>

to be or not to be
ADT Soup

Start

How many dimensions of data do I have?

One

Is my data in pairs?

Yes

Map

No

Two

Grid

Do I only care about membership?

Yes

Set

No

No - need duplicates or order

Which elements do I need to access?

Frequent looping or middle elements

First element

Vector

Last element

Queue

Stack
Problem: we want to schedule a dinner with some group of our friends

We have a text file with all our friends' dinner preferences

Given a group of friends going to a dinner, where should we eat to maximize happiness?
  • We might not be able to find a place that makes everyone happy – such is life

Which ADT(s) should we use?

<table>
<thead>
<tr>
<th>Ashley</th>
<th>In n Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipotle</td>
<td>Axe and Palm</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shreya</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipotle</td>
</tr>
<tr>
<td>Bytes Cafe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Karel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes Café</td>
</tr>
<tr>
<td>Forbes Cafe</td>
</tr>
</tbody>
</table>
