Collections, Part Three
Lexicon
Lexicon

• A **Lexicon** is a container that stores a collection of words.

• The Lexicon is designed to answer the following question efficiently:

  *Given a word, is it contained in the Lexicon?*

• The Lexicon does *not* support access by index. You can’t, for example, ask what the 137\textsuperscript{th} English word is.

• However, it *does* support questions of the form “does this word exist?” or “do any words have this as a prefix?”
Tautonyms

- A *tautonym* is a word formed by repeating the same string twice.
  - For example: murmur, couscous, papa, etc.
- What English words are tautonyms?
Some Aa

http://upload.wikimedia.org/wikipedia/commons/f/f1/Aa_large.jpg
One Bulbul
More than One Caracara

http://www.greglasley.net/images/CO/Crested-Caracara-F3.jpg
Introducing the Dikdik
And a Music Recommendation
Time-Out for Announcements!
Assignment 2

- Assignment 2 (Fun with Collections) goes out today. It’s due a week from this Monday.
  - Explore mathematical models of crystal formation!
  - Build a program that (almost) always wins at Hangman.
- We’ve provided a suggested timetable for completing this assignment on the front page of the handout. Aim to stick to this timeline; you’ve got plenty of time to complete things if you start early.
- **You must complete this assignment individually.** Working in pairs is not permitted yet.
YEAH Hours

• We’ll be holding YEAH (Your Early Assignment Help) hours for this assignment today at 3:30PM in Shriram 104.

• Can’t make it? No worries! The slides will be up on the course website.
LaIR Closure

• The LaIR will be closed on Sunday in observance of Dr. Martin Luther King, Jr. Day.

• The LaIR will, however, be open on Monday during the usual 7PM – 11PM time slot.
Set
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Set

- The Set represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Set

- The Set represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Operations on Sets

- You can add a value to a set by writing
  ```
  set += value;
  ```
- You can remove a value from a set by writing
  ```
  set -= value;
  ```
- You can check if a value exists by writing
  ```
  set.contains(value)
  ```
- Many more operations are available (union, intersection, difference, subset, etc.), so be sure to check the documentation.
Map
Map

- The Map class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.
Map

- The **Map** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.
Map

- The Map class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

<table>
<thead>
<tr>
<th>CS106B</th>
<th>Hello!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dikdik</td>
<td>Cute!</td>
</tr>
</tbody>
</table>
Map

- The **Map** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

<table>
<thead>
<tr>
<th>CS106B</th>
<th>Hello!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dikdik</td>
<td>Cute!</td>
</tr>
<tr>
<td>This Slide</td>
<td>Self Referential</td>
</tr>
</tbody>
</table>
Map

• The **Map** class represents a set of key/value pairs.
• Each key is associated with a unique value.
• Given a key, can look up the associated value.
Using the Map

- You can create a map by writing
  \[\text{Map}<\text{KeyType}, \text{ValueType}> \text{ map};\]
- You can add or change a key/value pair by writing
  \[\text{map}[\text{key}] = \text{value};\]
  If the key doesn't already exist, it is added.
- You can read the value associated with a key by writing
  \[\text{map}[\text{key}]\]
  If the key doesn't exist, it is added and associated with a default value.
- You can check whether a key exists by calling
  \[\text{map}.\text{containsKey}(\text{key})\]
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```
Map Autoinsertion

Map<string, int> freqMap;

while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

```
"Hello"
```
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

```cpp
{
    text = "Hello"
}
```
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

```
text    "Hello"
```
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Oh no! I don't know what that is!
Map Autoinsertion

Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

Let’s pretend I already had that key here.
Map Autoinsertion

Map<string, int> freqMap;

while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

The values are all ints, so I'll pick zero.
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Phew! Crisis averted!
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```
Map Autoinsertion

Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
Map Autoinsertion

Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

Cool as a cucumber.  
⊂(▀¯▀⊂)
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

 freqMap

```
Hello

1

"Hello"
```

Cool as a cucumber.

⊂ (▀¯▀⊂)
Map Autoinsertion

Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap
{ "Hello" 1
    text "Hello"
}
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

```
"Hello" 1
```
Map Autoinsertion

Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap

{ "Hello" 1 }
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

```
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>1</td>
</tr>
</tbody>
</table>
```

text "Goodbye"
Map autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

```
"Hello" 1
```

```cpp
text "Goodbye"
```
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

- "Hello" : 1

- text : "Goodbye"
Map Autoinsertion

Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap
  { "Hello" : 1 }

Oh no, not again!
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```
Map autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

```
freqMap

<table>
<thead>
<tr>
<th>Text</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Hello&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;Goodbye&quot;</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Text
"Goodbye"
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

<table>
<thead>
<tr>
<th>Text</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>1</td>
</tr>
<tr>
<td>Goodbye</td>
<td>0</td>
</tr>
</tbody>
</table>

Text: "Goodbye"
Map Autoinsertion

Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Hello&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;Goodbye&quot;</td>
<td>0</td>
</tr>
</tbody>
</table>

Chillin' like a villain.

C(uciones)
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

```
Hello 1
Goodbye 1
```

Chillin' like a villain.

\[C(\text{ifice})\]
Map Autoinsertion

Map<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap

"Hello" 1
"Goodbye" 1

text "Goodbye"
Map Autoinsertion

```cpp
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

```
freqMap
  "Hello" 1
  "Goodbye" 1
```
Sorting by First Letters
Lexicon english("EnglishWords.txt");

Map<\texttt{char}, \texttt{Lexicon}> wordsByFirstLetter;
\textbf{for} (\texttt{string word: english}) {
    \texttt{wordsByFirstLetter[word[0]].add(word);} 
}
Map Autoinsertion

Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;

for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

word  "first"
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

"first"
Lexicon english("EnglishWords.txt");

Map<\texttt{char}, \texttt{Lexicon}> \texttt{wordsByFirstLetter};
for (\texttt{string word: english}) {
    \texttt{wordsByFirstLetter[word[0]]}.add(word);
}

\texttt{wordsByFirstLetter}

\texttt{word} \quad \texttt{"first"}
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

Oops, no f's here.
Lexicon english("EnglishWords.txt");

Map<
char,
Lexicon>
wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
{'f'}

Let's insert that key.

word "first"
Lexicon english("EnglishWords.txt");

Map<Char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

I’ll give you a blank Lexicon.
Map Autoinsertion

Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

{  }

word
"first"
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;

for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

{ "first" }

'f'

word "first"
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

word

"first"
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
    'f'
    { "first" }
Map Autoinsertion

```java
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

\{'f'\}
\{"first"\}
```
Map Autoinsertion

Lexicon english("EnglishWords.txt");

Map< char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

word

"foremost"
Lexicon english("EnglishWords.txt");

Map<
char,
Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

'f' { "first" }

word "foremost"
Map Autoinsertion

Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter['f'] = { "first", "foremost" };
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter['f'] { "first" }

"foremost"

Easy peasy.
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

'f' { "first" }

word "foremost"
Lexicon english("EnglishWords.txt");

Map<\texttt{char}, \texttt{Lexicon}> wordsByFirstLetter;
\textbf{for} (string \texttt{word: english}) {
    wordsByFirstLetter[\texttt{word[0]}].add(\texttt{word});
}

wordsByFirstLetter
\{ 'f' \{ "first", "foremost" \} \}

\texttt{word} \ "foremost"
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter { 'f' { "first", "foremost" } }
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

'f' { "first", "foremost" }
Lexicon english("EnglishWords.txt");

Map<
char, Lexicon> wordsByFirstLetter;

for (string word: english) {
  wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

{ 'f': { "first", "foremost" } }

word

"initial"
Map Autoinsertion

Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter['f'] { "first", "foremost" }

word "initial"
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

| 'f'         | { "first", "foremost" } |
| 'i'         | {}                      |

word "initial"
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
    'f' { "first", "foremost" }
    'i' {}
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

```
| 'f'          | { "first", "foremost" } |
| 'i'          | { "initial" }           |
```

word  "initial"
Anagrams

- Two words are *anagrams* of one another if the letters in one can be rearranged into the other.

- Some examples:
  - “Senator” and “treason.”
  - “Praising” and “aspiring.”
  - “Arrogant” and “tarragon.”

- *Question for you:* does this concept exist in other languages? If so, please send me examples!
Anagrams

- **Nifty fact:** two words are anagrams if you get the same string when you write the letters in those words in sorted order.

- For example, “praising” and “aspiring” are anagrams because, in both cases, you get the string “aiignprs” if you sort the letters.
Anagram Clusters

• Let’s group all words in English into “clusters” of words that are all anagrams of one another.

• We’ll use a Map<string, Lexicon>.
  • Each key is a string of letters in sorted order.
  • Each value is the collection of English words that have those letters in that order.
Next Time

• **Thinking Recursively**
  • How can you best solve problems using recursion?
  • What techniques are necessary to do so?
  • And what problems yield easily to a recursive solution?
Extra Content: How to Sort a String
Order in Range-Based for Loops

- When using the range-based for loop to iterate over a collection:
  - In a `Vector`, `string`, or array, the elements are retrieved in order.
  - In a `Map`, the `keys` are returned in sorted order.
  - In a `Set` or `Lexicon`, the values are returned in sorted order.
Counting Sort
Counting Sort

banana
Counting Sort

banana

letterFreq
for (char ch: input) {
    letterFreq[ch]++;
}

Counting Sort

banana

letterFreq
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

banana

letterFreq
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;  
}

letterFreq

banana
Counting Sort

```
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

`banana`

```
letterFreq
```

```
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

```java
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

```java
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq

banana

b 1
a 2
n 1
a 3
Counting Sort

```java
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

```java
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq

banana

\[
\begin{array}{c|c}
 a & 3 \\
 b & 1 \\
 n & 2 \\
\end{array}
\]
Counting Sort

banana

letterFreq

<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
</tr>
</tbody>
</table>
Counting Sort

```
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```
Counting Sort

```java
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```
Counting Sort

For (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}

\[
\begin{array}{c|c}
\text{b} & 1 \\
\text{a} & 3 \\
\text{n} & 2 \\
\end{array}
\]
Counting Sort

banana

letterFreq

for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
Counting Sort

**banana**

```java
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

**letterFreq**

```
a 3
b 1
n 2
```
Counting Sort

Count the occurrences of each letter in the word "banana" and then output the letters in order of their frequency.

```java
for (char ch : letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

letterFreq

```
+---+----+
| a | 3  |
| b | 1  |
| n | 2  |
```

Output: `aaaab`
Counting Sort

banana

letterFreq

<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
</tr>
</tbody>
</table>

for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}

aaabbn
Counting Sort

banana

letterFreq

a 3
b 1
n 2

aaabbnns