Shannon approximated the statistical structure of a piece of text using a simple mathematical model known as a Markov model. A Markov model of order 0 predicts that each letter in the alphabet occurs with a fixed probability. We can fit a Markov model of order 0 to a specific piece of text by counting the number of occurrences of each letter in that text, and using these frequencies as probabilities. For example, if the input text is "gagggagagggagaaaa", the Markov model of order 0 predicts that each letter is 'a' with probability 7/17, 'c' with probability 1/17, and 'g' with probability 9/17 because these are the fraction of times each letter occurs. The following sequence of characters is a typical example generated from this model. A Markov model of order 0 assumes that each letter is chosen independently. This independence does not coincide with the actual English text because there is a high correlation among letters in English. For example, 'w' is more likely to be followed with 'e' than with 'u', while 'q' is more likely to be followed with 'i'.
Agenda

- Miscellaneous updates about LFSR and Guitar Hero
- Structural overview of Markov Model
- Details about k-grams and frequency counts
- Code and data structures
Misc. Updates

• Guitar
  • To be released tonight or tomorrow afternoon
  • 31 submissions got +0.5pts
  • 29 submissions got +1.0pts

• LFSR rubric feedback:
  - Recurring expression that is not defined as IV (seeds.length rather than having a n IV)
  - Constant value/IV could be final [grader: indicate which IV's you are talking about]
  - Recurring expression instead of existing IV [grader: indicate which IV's you are talking about]
  - A lot of intermediate variables [grader: indicate a specific example from student code]

private final double CONCERT_A = 440.0;
private final double ENERGY_DECAY = 0.996;
Live Questions on Sli.do

- QUESTIONS ARE WELCOME during this class meeting
- Go to https://sli.do
- Enter meeting code: #N700
- You can:
  - ASK your own questions
  - UPVOTE questions you are interested in hearing the answer of
- Will monitor and select questions
- Please be kind!
OVERVIEW
Overview

- Two parts
  - Train linguistic model on large corpora with MarkovModel
  - Generate similar random text
  - Training is by counting the frequency of occurrences of characters

TextGenerator

```
MarkovModel("banana", 2)
```

```
k = 2
```

- `ab` →
  - `a` → 1
  - `b` → 0
  - `n` → 0
- `an` →
  - `a` → 2
  - `b` → 0
  - `n` → 0
- `ba` →
  - `a` → 0
  - `b` → 0
  - `n` → 1
- `na` →
  - `a` → 0
  - `b` → 1
  - `n` → 1

random(...)
constructor
High-Level Steps

1. Pick a text and an order $k$

2. Read text in k-grams (sequences of $k$ adjacent characters), and create a frequency table of the characters that follow each k-gram

3. Use the frequency table to, given a k-gram, produce a random character

4. **TextGenerator**: Once you have trained model, and from starting string, repeatedly produce random characters and append to randomly generated string
Obama's DNC 2004 speech

Thank you so much. Thank you so much. Thank you. Thank you. Thank you.


Thank you. Thank you, Dick Durbin. You make us all proud.

On behalf of the world.

Now -- Now let me be clear. Let me be clear. Let me be clear. We have made over two hundred years ago: "We hold these truths to be rich to achieve unless we raise their problem
DETAILS
Live Questions on Sli.do

- QUESTIONS ARE WELCOME during this class meeting
- Go to https://sli.do
- Enter meeting code: #N700
- You can:
  - ASK your own questions
  - UPVOTE questions you are interested in hearing the answer of
- Will monitor and select questions
- Please be kind!
What You Need To Know

• What are characteristics + performance of symbol tables (or “associative arrays”, or “mappings” or “dictionaries”)?

• How to data structures with generics?

• How to combine several data structures together?

• How to sample from a probability distribution.
An example of k-grams (k=3)

they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
An example of $k$-grams ($k=3$)

they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
deep in the sand, they
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
they sold seashells on the shore
An Example ($k=2$)

<table>
<thead>
<tr>
<th>b:</th>
<th>o 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>f:</td>
<td>a 1e 1o 1</td>
</tr>
<tr>
<td>j:</td>
<td>e 2</td>
</tr>
<tr>
<td>m:</td>
<td>o 1</td>
</tr>
<tr>
<td>a:</td>
<td>f 2</td>
</tr>
<tr>
<td>an:</td>
<td>a 1n 1</td>
</tr>
<tr>
<td>be:</td>
<td>r 1</td>
</tr>
<tr>
<td>bo:</td>
<td>b 1n 1</td>
</tr>
<tr>
<td>ee:</td>
<td>f 1</td>
</tr>
<tr>
<td>ef:</td>
<td>i 1</td>
</tr>
<tr>
<td>em:</td>
<td>y 6</td>
</tr>
<tr>
<td>er:</td>
<td>e 6</td>
</tr>
<tr>
<td>fa:</td>
<td>n 1</td>
</tr>
<tr>
<td>fe:</td>
<td>e 1r 1</td>
</tr>
<tr>
<td>fi:</td>
<td>1</td>
</tr>
</tbody>
</table>

| fo: | f 1 |
| i: | m 1 |
| je: | r 3 |
| me: | r 1 |
| mo: | m 1 |
| my: | 5j 1 |
| na: | 2n 1 |
| nn: | a 1 |
| ob: | e 1 |
| of: | e 1 |
| om: | e 1 |
| on: | a 1 |
| re: | m 6 |
| y: | b 2f 1j 2 |
| yj: | e 1 |

Jeremy Jeremy Boberemy Bonana Fanna Foferemy Feefi Momeremy Jeremy
CODE and DATA STRUCTURES
Symbol Tables ("Associative Arrays")

From book site:

- Associate any value to another value
- Covered in precept

This implementation uses a balanced binary search tree. It requires that the key type implements the `Comparable` interface and calls the `compareTo()` method to compare two keys. [...] The `put`, `contains`, `remove`, [...] operations each take logarithmic time in the worst case.
Arrays vs. Symbol Tables

Examples

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 0 | → | ...
| 1 | → | ...
| 2 | → | ...

...  

N → ...

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 0 | → | ...
| 6 | → | ...
| 10 | → | ...

...  

"other" → ...

N → ...

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 0 | → | ...
| 6 | → | ...
| 10 | → | ...

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 0 | → | ...
| 6 | → | ...
| 10 | → | ...

"symbol" → ...

"table" → ...

"word" → ...

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| "other" | → | ...
| 12 | → | ...

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| "symbol" | → | ...
| "table" | → | ...
| "word" | → | ...

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| "symbol" | → | ...
| "table" | → | ...
| "word" | → | ...

Declaration

int[] array = new int[N];

ST<String, String> st = new ST<String, String>();

ST<Integer, String> st = new ST<Integer, String>();

Ranges

Contiguous ranges of numbers from 0 to N (some number)

Map set of values from any data type (that is comparable)

Sets of sparse numbers

Performance

Constant time  
Most efficient storage

Logarithmic time (in number of keys)

Less efficient storage
The MarkovModel object has two levels of mappings:

- Mapping k-grams to frequency tables
- Mapping characters to frequencies
- Choice of data structures is important

**MarkovModel("banana", 2)**

<table>
<thead>
<tr>
<th>k = 2</th>
<th>ab</th>
<th>an</th>
<th>ba</th>
<th>na</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a → 1</td>
<td>a → 2</td>
<td>a → 0</td>
<td>a → 0</td>
</tr>
<tr>
<td></td>
<td>b → 0</td>
<td>b → 0</td>
<td>b → 0</td>
<td>b → 1</td>
</tr>
<tr>
<td></td>
<td>n → 0</td>
<td>n → 0</td>
<td>n → 1</td>
<td>n → 1</td>
</tr>
</tbody>
</table>
Iterating over keys of a ST

- While we don't emphasize it, it is possible to iterate over the keys of a symbol table (in alphabetical order):

  ```java
  ST<String, int> st = new ST<String, int>();
  st.put("a", 0); st.put("c", 0); st.put("b", 1);
  for(String key : st.keys())
    StdOut.println(key)
  // will print "a", "b", "c"
  ```
StringBuilder (1/2)

• We've told you that String concatenation is not efficient (precept, class meeting, assignment deductions) **because Java creates copies of Strings**

• StringBuilder is a Java class to address this issue:
  • StringBuilder sb = new StringBuilder("initial string"); // initial can be empty
  • sb.append("  concatenated");
  • StdOut.println(sb.toString()); // print "initial string concatenated"
  • sb.insert(0, "start: ");
  • StdOut.println(sb.toString()); // print "start: initial string concatenated"

• Operations more efficient than with straight concatenation
StringBuilder (2/2)

From javapapers.com (by Joe) see http://bit.ly/JavaPapersStringBuilder
Random characters

Frequencies  "y_": b 2  f 1  j 2
2 + 1 + 2 = 5
(normalization)

Probabilities  "y_": b 2/5  f 1/5  j 2/5

```java
int[] frequencies = { 2, 1, 2 }; StdRandom.discrete(frequencies);
// will return
// 0  with probability 2/5
// 1  with probability 2/5
// 2  with probability 2/5

double[] probabilities = { 2/5.0, 1/5.0, 2/5.0 }; StdRandom.discrete(frequencies);
// will return
// 0  with probability 2/5
// 1  with probability 2/5
// 2  with probability 2/5
```

- Either use an array of frequencies/probabilities
- Or draw a random number and iterate over all your possible characters ("alias method")
Thank you!