Dense Bags, Markov Text Generation

CMSC 132

Department of Computer Science

University of Maryland, College Park

Overview

- DenseBags
- Markov Chains
- Markov Text Generation
- Project Info

DenseBag

- Data structure similar to a set, but can contain duplicates
- Example

  { 1, 3, 1, 1, 3, 5 }
- Which is the same as

  { 1, 1, 1, 3, 3, 5 }
- You will need to implement this data structure for the project

DenseBag<E> operations

- In addition to the operations supported on any Collection, we need to support:

```
Set<E> getUniqueElements()
int getCount(E e)
E choose(Random r)
```

- Given the DenseBag { 1, 1, 1, 3, 3, 5 }, what would these methods do?

Efficiency

- Most operations on a dense bag should take O(1) average time
  - time is average because it may depend on hashing
- choose(Random r) may take time proportional to the number of unique elements in the bag

Couple of notes

- remove(Object o) will remove only one instance
- uniqueElements returns elements that occur at least once
- getCount(E e) returns 0 if e doesn't occur in the bag at all

Iterators

- Assume we have DenseBag<Integer> = { 1, 1, 1, 3, 3, 5 }
- What do you think an iterator over the dense bag would do?
  - Order in which we do iteration doesn't matter

Markov Chain

- In finite state machines transitions are based on the next input symbol
  - Example: Finite state machine for any number of a's followed by even number of b's
- What if we based the state transition on probabilities instead? Then you have a markov chain

Markov Chain Example

- You could model weather prediction as the following markov chain:
  - If it is sunny today, there is a 90% chance it will be sunny tomorrow, otherwise it will be cloudy.
  - If it is cloudy today, there is a 50% chance it will be sunny tomorrow, otherwise it will be cloudy.

Order-n Markov Chain

- first-order markov chain
  - states are chosen randomly, with the probability distribution for the next state being determined only by the current state
- second-order markov chain
  - states are chosen randomly, with the probability distribution for the next state being determined by the previous two states
- n-order markov chain
  - states are chosen randomly, with the probability distribution for the next state being determined by the previous n states

Markov Text Generation

- **Text generation with the Markov Chain Algorithm**
  - **Read a document (training sequence) and use it to determine the frequency of "what immediately follows what". The result is a transition table.**
  - **Use the transition table to generate random text**

Example

- Consider building a 2nd order markov chain based on the characters in xxxxyyxxz
- Assume we start with xx
- xx occurs 4 times, and is followed by:
  - a x twice
  - a y once
  - a z once
- So when generating random text, xx should be followed by an x 50% of the time, a y 25% of the time, and a z 25% of the time

Questions

- What data structures would be helpful for this?
- How do we handle starting?
  - What characters/words start the randomly generated sequence
- How do we handle ending?
  - When should we stop the generated text?

Project Info

- We want to generate a transition table showing, given a list of the words/characters most recently generated, a DenseBag of the word/character that occurred next in the training documents
- We've provided you with 12 Sherlock Holmes stories, plus the text of Hamlet
- You can generate text from one Sherlock Holmes story, all 12, or even a Sherlock Holmes and Hamlet mashup.
- Various classes provided to read/write text from files and some some automatic generation code
  - will work once you have implemented the project

Project Info

- Creating and Training a MarkovText
- A MarkovText is created with a specific order
- Train it on a sequence of strings
  - Pass the updateMarkovTransitions method an Iterator<String>
- A MarkovText can be trained on multiple sequences