CSSE 220 Day 28

Markov

Checkout *Markov* project from SVN

Questions

Presentations

- Wednesday, 10:30 – 2:00 in the Union lobby
  - You have a 15-min time slot where your whole team can be there
  - You'll demo on a projector; anyone can watch
  - If it's too sunny, then go to a nearby (PA?) room
- Each person will
  - talk for ~1 minute about a technical facet of the program to which they contributed
  - be prepared to answer questions about the project
- Be professional!
  - Be prepared
  - Dress nicely

Presentation times

Time	Team
3rd hour	
10:35	8
4th hour	
10:50	2
11:05	4
11:20	3
5th hour	
11:45	6
12:00	7
12:15	9
6th hour	
12:40	5
7th hour	
1:35	1

Vector Graphics Team Evaluations

- Complete survey on ANGEL by start of class Thursday
  - Lessons → Project Forms → Team Performance Evaluations

- Failure to complete the evaluations will result in a letter grade deduction on your individual team project score

Announcements

- Due to Wednesday's presentations, Friday's class will be optional
- But for those who are here, it will be a great time to work on the Markov project, especially if you are working with a partner

Markov Chaining

- Details

Markov Chain Progam

- Input: a text file

```
the skunk jumped over the stump
the stump jumped over the skunk
the skunk said the stump stunk
and the stump said the skunk stunk
```

- Output: a randomly generated list of words that is “like” the original input in a well-defined way

Markov Chain Process

- Gather statistics on word patterns by building an appropriate data structure
- Use the data structure to generate random text that follows the discovered patterns

Markov Example, n = 1

- Input: a text file

the skunk jumped over the stump
the stump jumped over the skunk
the skunk said the stump stunk
and the stump said the skunk stunk

Prefix	Suffixes
NONWORD	the
the	skunk (4), stump (4)
skunk	jumped, said, stunk, the
jumped	over (2)
over	the (2)
stump	jumped, said, stunk, the
said	the (2)
stunk	and, NONWORD
and	the

Markov Example, n = 2

- Input: a text file

the skunk jumped over the stump
the stump jumped over the skunk
the skunk said the stump stunk
and the stump said the skunk stunk

Prefix	Suffixes
NW NW	the
NW the	skunk
the skunk	jumped, said, the, stunk
skunk jumped	over
jumped over	the
over the	stump, skunk
the stump	the, jumped, stunk, said
…	

Output

- n=1:

`the skunk the skunk jumped over the skunk stunk`

`the skunk stunk`

- n=2:

`the skunk said the stump stunk and the stump jumped over the skunk jumped over the skunk stunk`

- Note: it's also possible to hit the max before you hit the last nonword.

Markov Data structures

- For the prefixes?
- For the set of suffixes?
- To relate them?

Prefix	Suffixes
NW NW	the
NW the	skunk
the skunk	jumped, said, the, stunk
skunk jumped	over
jumped over	the
over the	stump, skunk
the stump	the, jumped, stunk, said
…	

Fixed-Length Queue and Markov

- FixedLengthQueue: a specialized data structure, useful for Markov problem
- Check out **`FixedLengthQueue`**
  - Working alone? See your individual repo.
  - Working with a partner? See your new Markov repo.

- Work to implement it in the next 25 minutes or so
- When you finish, read the (long) Markov description and start coding
- We will only do milestone 1 (so no text justification)

Teams

01, breenjw,runchemr
02, hugheyjm,weavergg
03, hannumed,woodhaal
04, labarpr,
05, macshake,mcgeevsa
06, pedzindm,parasby
07, eatonmi,
08, correlbn,shinnsm
09, smebaksg,
10, moravemj,wanstrnj
11, cheungkt,ngop
12, duganje,
13, carvers,krachtkq
14, lemmersj,
15, popenhjc,
16, beaversr,davidsac
17, amanb,
18, foltztm,
19, sheetsjr,
20, walthagd,

http://svn.csse.rose-hulman.edu/repos/csse220-201030-markov-teamXX

Work Time

- Review HW description,

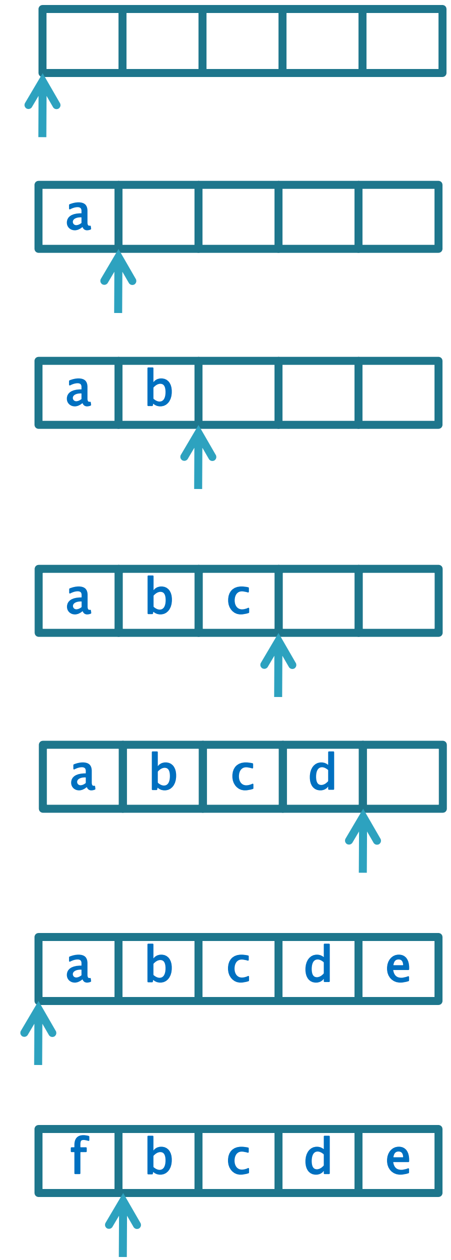
  Work on Markov for rest of class

  The following slides may have some helpful hints

Arrow shows the point at which next to add data

Fixed length queue (FLQ)

- Example to the left shows the queue as elements are added
  - We'll only *add*, no *remove*
- What do you need to implement this?
  - *Array* whose length is the capacity of the FLQ
  - *Index* at which to add the next element to the FLQ
    - This index increases by 1 as you add elements, but "wraps" back to 0 when it reaches the capacity of the FLQ
  - Current *size* of the FLQ
    - As opposed to the capacity of the FLQ



Generating sentences by a Markov chain

<u>Input</u>:
Blessed are the poor for they will be Blessed are the peacemakers for they will find Blessed are meek for they will be Blessed are

Inspired by Matthew 5:3-9

To generate a new phrase, start with NONWORD NONWORD and "follow the chain", but choose *at random* from eligible suffixes

Prefix (n = 2)	Suffix
NONWORD NONWORD	Blessed
NONWORD Blessed	are
Blessed are	the the meek NONWORD
are the	poor peacemakers
the poor	for
poor for	they
for they	will will will
they will	be find
will be	Blessed Blessed
be Blessed	are are
the peacemakers	for
peacemakers for	they
will find	Blessed
find Blessed	are
are meek	for
meek for	they
are NONWORD	NONWORD

What data structures to use?

Use a **Fixed-Length Queue** whose length is n

Use a **MultiSet**

- Stores each word with its multiplicity
- Has:
  - `size()`
  - `findKth(int k)`
- To “pick at random” from a MultiSet, generate a random number, *k*, between 0 and `size()`, then call `findKth(k)` to get the random word

Prefix (n = 2)	Suffix
NONWORD NONWORD	Blessed
NONWORD Blessed	are
Blessed are	the the meek NONWORD
are the	poor peacemakers
the poor	for
poor for	they
for they	will will will
they will	be find
will be	Blessed Blessed
be Blessed	are are
the peacemakers	for
peacemakers for	they
will find	Blessed
find Blessed	are
are meek	for
meek for	they
are NONWORD	NONWORD

The Markov Map

This mapping is what we want to generate new data from the existing data, using a Markov Chain

w_{k-4} w_{k-3} w_{k-2} w_{k-1} w_k $\longrightarrow$ w_{k+1}

- When building the map: the word that follows the given prefix
- When generating from the map: random but according to the data distribution

Implement as a Fixed-Length Queue whose length is *n*

Implement the mapping as a **`HashMap<String, MultiSet>`** where the String is the concatenation of the words in the Fixed-Length Queue, and the MultiSet is the set of words that follow that String in the input

Implement by choosing at random from the mapped MultiSet

Do you see why these are good data structures for this problem?

Building the Markov Map

Initially, the FLQ contains NONWORD at all indices and w_{k+1} is the first word of the input

FLQ:

w_{k-4}	w_{k-3}	w_{k-2}	w_{k-1}	w_k

toString

String (key): w_{k-4} w_{k-3} w_{k-2} w_{k-1} w_k

get the MultiSet from the HashMap<String, MultiSet>, using this key

Previous MultiSet

If the MultiSet is null, construct the MultiSet and put it into the HashMap. In any case, add w_{k+1} to the MultiSet

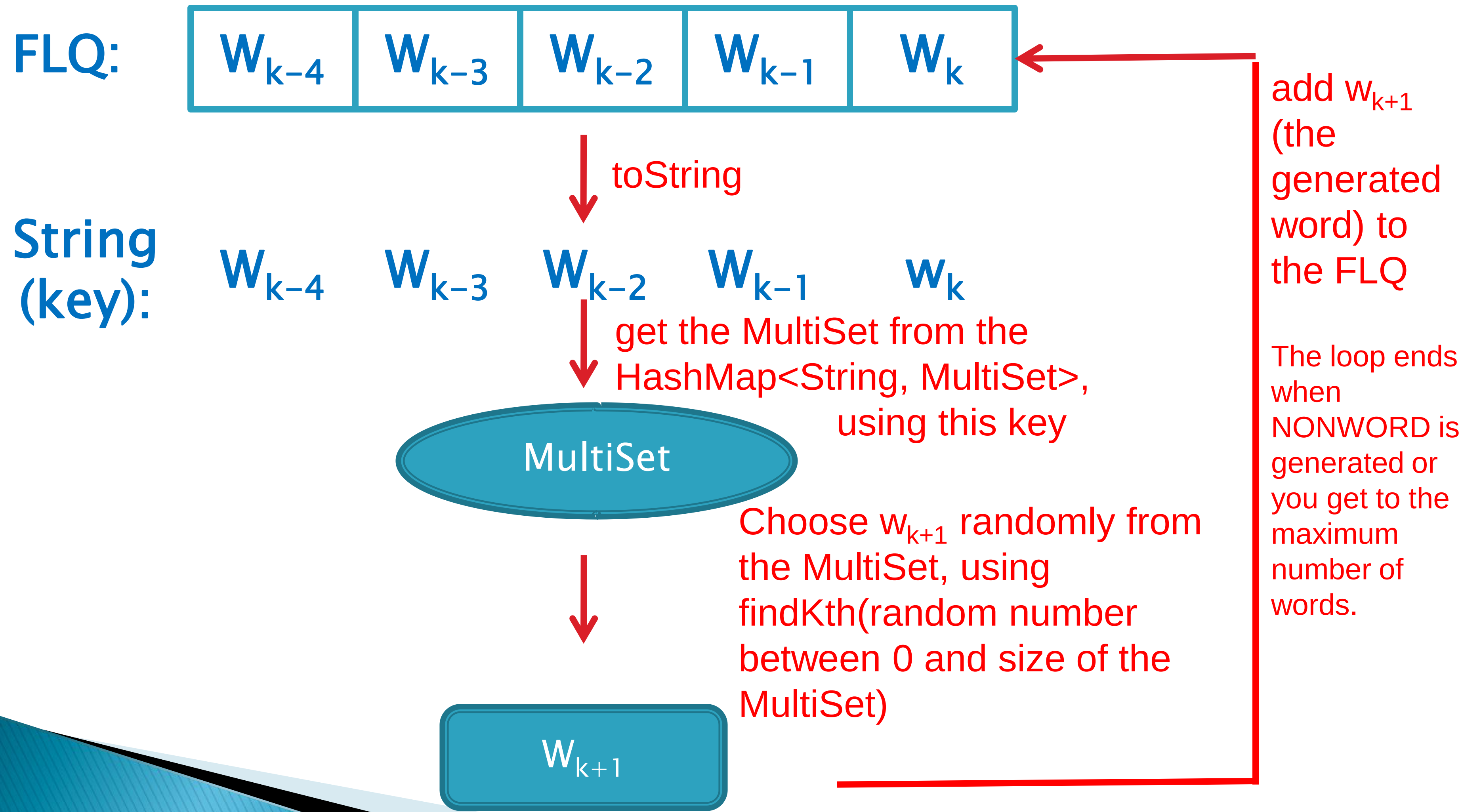
Previous MultiSet plus w_{k+1}

add w_{k+1} (the next word in the input file) to the FLQ

The loop ends when the input file is empty. Follow the loop by putting NONWORD as w_{k+1} n times.

Generating from the Markov Map

Initially, the FLQ contains NONWORD at all indices



Reading words from a file

```
Scanner scanner =
    new Scanner(
        new File (
            this.pathToInputFile)));

while (scanner.hasNext()) {
    String word = scanner.next();
    ...
}
```