CS1110 lec 12: Analysis of lies using recursion   07 Oct 2010

Today: recursion in an interdisciplinary application:
computer science/computational linguistics, psychology, history/politics

Have your iClickers out.

Reading for next lecture (casting about): Secs 4.2 & 4.3

- *Prelim, 7:30-9pm today*
  - Last name A-K: go to Olin 155
  - Last name L-Z: go to Olin 255
- A4 due Saturday Oct. 16
- No labs next week (Tue-Wed Oct 11-12), due to fall break
- No office/consulting hours Friday through Tuesday inclusive (Oct 8-12), due to fall break

Lies, damned lies, and statistics

James Pennebaker et al., “Lying words: predicting deception from linguistic styles”, 2003:

Claim: deceptive communication is characterized by (among others):
- fewer 1st-person singular and 3rd-person pronouns (“I”, “they”)
- more negative emotion words (“hate”, “enemy”)
- fewer “complex/exclusive” words (“but”, “except”, “without”)

Research question (1): What really are the best cues? (or models)
Sub-question: is there a more realistic, convenient source of “lies”?

The “Iraq War Card False Statements Database”
http://projects.publicintegrity.org/WarCard/Search/Default.aspx

Where we are, and why
[besides automatic lie detection being inherently cool]

Research question 1: What are the best linguistic lie cues (or models)?

Research question 2: are the "true" and "false" statements regarding Iraq by top Bush administration officials distinguishable? This would imply something about their beliefs.

- Demonstration of interdisciplinary research involving computer science, psychology/linguistics, politics and history

- Demonstration of methodology in approaching a programming problem
  - stepwise refinement, writing and reading specs carefully, String manipulation, recursion, testing, etc.

(Lecture loosely based on joint work with CS grad student Cristian Danescu-Niculescu-Mizil and CS undergrad Haden Lee, in consultation with Comm./IS Prof. Jeff Hancock.)

Formulating the task: might something be a lie cue?

Given
- a target word *w* (e.g., “**they**”)
- a file containing “lie span mark-up”:

… with respect to iraq, the problem is quite simple. **<span style="background-color: yellow;">** we suspect ***they*** are developing weapons of mass destruction. we more than suspect it; we know it.**</span>** **they** could…

spanStart

spanEnd

Consider these two statistics for *w*:
- number of *hits*: occurrences of *w* in a “lie” span
- number of misses*:* occurrences of *w* not in a “lie” span

Write a class LieData with method counts(*w*,…) that will tell us the number of hits and the number of misses for *w* in a specific text (file).

To write: a class LieData with method counts(*w*).
We need to track the source file’s text, spanStart, spanEnd, and the target word *w*.
Assume we want separate results for separate texts e.g., (comparing lab vs. life)

Q: How should we declare the relevant entities?

(A) *text and the span delimiters are stored in individual objects*
```
private String text; private String spanStart; private String spanEnd;
public String counts(String w) {…}
```

(B) *everything is a parameter to static method counts*
```
public static String counts(String textFileName, String spanStart,
    String spanEnd, String w) {…}
```

(C) *this mixture of objects and static:*
```
private String text; private String spanStart; private String spanEnd
public static String counts(String w) {…}
```

1. Get data from source file into Java-manageable format
   write a constructor LieData(<file>) using In.java to initialize *text*
2. Get target word *w*
   parameter for method counts(*w*)
3. Process each occurrence of *w* in *text*
   - Is it a hit or a miss?

4. Report relevant statistics
   output of counts(*w*): “hits <h>; misses <m>; ratio: <h>/<m>”

The core recursive idea: how many hits past an index *i?*

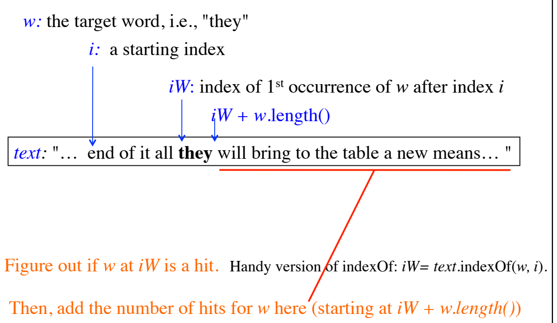

w: the target word, i.e., "they"
i: a starting index
iW: index of 1st occurrence of *w* after index *i*
iW + *w*.length()
text: "… end of it all **they** will bring to the table a new means… "

Figure out if *w* at *iW* is a hit. Handy version of indexOf: *iW*= *text*.indexOf(*w*, *i*).

Then, add the number of hits for *w* here (starting at *iW* + *w.length()*)

We need the output of helper method countsFrom(String w, int i) to encode *both* the numbers of hits *and* the number of misses in a way such that we can extract or add to them. But a function can only return one thing…

Q: What format should countsFrom's output be?

(A) an int, the difference between the # of hits and the # of misses

(B) a String "hits: <# of hits>; misses: <#misses>"

(C) a double, using decimal as separator. Example: 4.5 means 4 hits, 5 misses

(D) an object that maintains hits and misses (we would have to write the class)

A helper class for storing hit/miss pairs

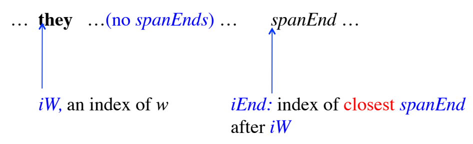
```
/** An instance maintains a number of hits and a number of misses */
public class TallyPair {
    /** number of hits */
    public int hits;
    /** number of misses */
    public int misses;

    /** Constructor: a TallyPair with 0 hits and 0 misses */
    public TallyPair() {
        ;
    }
    /** = hits: <# of hits>; misses <# of misses> */
    public String toString() {
        return "hits: " + hits + "; misses: " + misses + "; "
                + "ratio: " + (double)hits/misses;
    }
}
```

(A better choice would be to make this a private static nested class: see pp 348—350.)

How do we determine if *w* at index *iW* is a hit or not?

boolean procedure isHit(String *w*, int *iW*)


… **they** …(no *spanEnds*) … *spanEnd* …
iW, an index of *w*
iEnd: index of closest *spanEnd* after *iW*

Handy version of lastIndexOf:
return *iEnd* != - 1 && *text*.lastIndexOf(*spanStart*, *iEnd*) < *iW*;

We wish you luck on tonight's prelim …

… and that's no lie!