Cpt S 223 Fall 2013
Homework 06
Due 12-12-2013
Novella (100 points)

Write a program called "**novella**" to search and index text. Your program should read a file specified as an argument on the command line.

novella **myNovel.txt**

The program should allow the user to: ***search*** for a keyword to find whole and partial sub-string matches within the text.

Novel File Structure

1. The file format will be ASCII text [0-9][a-zA-Z] including punctuation.
2. Lines should be parsed by newline character \n
3. Pages should be counted every 20 lines. i.e. every 20 lines you read, you should increment the page number. You can assume this number to be fixed.
4. Lines should be relative to page numbers. i.e. You should have Page 1, line 0, Page 2, line 0, Page N, line 0.

Requirements

Ignore common words - Your program should ignore the top 100 most common words in the English language (see the end of the assignment for list). You may use an STL map to check each word before inserting into the Trie and Suffix Tree.

Non-Alphanumeric Characters – Your program should ignore non-alphanumeric characters such as punctuation, hyphens, etc., such as "dog's", or "top-dog". Only [0-9][a-zA-Z] must be considered when matching.

Case sensitivity – Your program should ignore case sensitivity. Searching for DoG should return DOG or dog. You may convert all text to upper or lower case to make this assignment easier. Alternatively, no bonus points will be given for maintaining case.

PREFIX Search Command – The program should have a search mode that looks for prefixes of words. i.e. prefix dog would return the positions in the text where dog was listed. The positions should be given by page number, line number. You should also print the surrounding text where the keyword was found with the ellipsis as shown below:

prefix dog

dog found on:

Page 15, Line 7

…the big **dog** was here…

Page 16, Line 8,

…gone away. **Dog's** once ruled…

Page 16, Line 9

…there my **Dog** was spotted…

You should print two words before and after the keyword. In the Above example the keyword **dog** was found on three pages. Your program should not return "g-dog", as it is prefixed by a **g.**

Suffix Search Command – The program should have a search mode that looks for patterns in the text. This should list where in the text the pattern was listed. The positions should be given by page number, line number. You should also print the surrounding text where the keyword was found with the ellipsis as shown below:

suffix dog

Page 15, Line 7

…he was bird**dog**ged when he…

Page 16, Line 8,

…gone away. **Dog's** once ruled…

Page 16, Line 9

…there my **Dog** was spotted…

You should print two words before and after the keyword. In the Above example the keyword **dog** was found on three pages. You don't have to worry about showing words on multiple lines. Although the program should now show:

Page 16, Line 16 & 17

…there my

dog was sitting…

Bonus

5 bonus points if you print the keyword in colored text. Prefix and suffix commands should only highlight the keyword searched (it's ok if you include punctuations, hyphens, etc., e.g. main-**dog-**s)

http://en.wikipedia.org/wiki/ANSI_escape_code#graphics
http://stackoverflow.com/questions/2616906/how-do-i-output-coloured-text-to-a-linux-terminal

Grading

C - (starting at 72/100)

Prefix finds the text and prints the pages and line numbers

Prefix finds the text and prints the pages, line numbers, and surrounding text up to N characters

Prefix Finds the text and prints the pages, line numbers, and surrounding two words (left and right)

B - (starting at 82/100)

Suffix finds the pattern and prints the pages and line numbers

Suffix finds the pattern and prints the pages, line numbers, and surrounding text, up to N characters

A - (starting at 92/100)

Suffix finds the pattern and prints the pages, line numbers, and surrounding two words (left and right)

Bonus + 10

1. **Compare the space requirements for a trie to that of a standard array**
2. **Compare the time it takes to search for a keyword pattern using the naïve approach discussed in class to using a suffix array**
3. **If you don't store the entire file for printing pre- and post-words after the pattern.**

Bonus + 5

1. **Colored text as mentioned above**

Max

Max points = 110/100

Fail

You don't use a trie
Your code does not unzip
Your code does not compile
If you only do the prefix command, you will receive half of the points for each section.

Rank	Word
1	the
2	be
3	to
4	of
5	and
6	a
7	in
8	that
9	have
10	I
11	it
12	for
13	not
14	on
15	with
16	he
17	as
18	you
19	do
20	at
21	this
22	but
23	his
24	by
25	from
26	they
27	we
28	say
29	her
30	she
31	or
32	an
33	will
34	my
35	one
36	all
37	would
38	there
39	their
40	what
41	so
42	up
43	out
44	if
45	about

46	who
47	get
48	which
49	go
50	me
51	when
52	make
53	can
54	like
55	time
56	no
57	just
58	him
59	know
60	take
61	people
62	into
63	year
64	your
65	good
66	some
67	could
68	them
69	see
70	other
71	than
72	then
73	now
74	look
75	only
76	come
77	its
78	over
79	think
80	also
81	back
82	after
83	use
84	two
85	how
86	our
87	work
88	first
89	well
90	way
91	even
92	new
93	want

94	because
95	any
96	these
97	give
98	day
99	most
100	us

Guidance

This program is not as big as it may look. Consider what you have to do. Start first with pen and paper. Build the concept first **BEFORE PROGRAMMING!!!!!!!** Or you are destined to do poorly.

Data structures

1. Trie
    a. compressed or standard trie
2. Suffix Tree
    a. This is a compressed trie
3. Pattern
4. Terminal Node
    a. Store page #
    b. Store line #
    c. You could use two more pointers to reduce space requirements for your software.
5. You could store each line
    a. Given Page ID, Line ID, return vector of words

Building

1. Trie / Suffix Construction
    a. Word Ignorer
        i. Given a word, should it be inserted in the trie?
    b. File Reading
        i. Parsing words, split on newline, split on space,
        ii. Page / Line Identifier Algorithm
    c. Word Tracker
        i. Given a line ID, return all words
2. Algorithm
    a. Get User Input Command and Pattern
    b. Find all matches for pattern
        i. Prefix or Suffix – think about it, is there much of a difference in what they return?
    c. For each match
        i. Lookup the Line
        ii. Get the pre-words before pattern
        iii. Get the post-words after pattern
        iv. Print (pre-words, word, post-words)

Define Your Tasks

1. Define your test cases
2. Define your objects you will build
3. Write each test and object and test each feature first.
4. Then integrate the above.