Assignment 2
Data Mining, 2001
Due Date: November 23

An index for free text searching

The purpose of this programming assignment is to implement the technique of free text searching using suffix arrays.

Assume that an input file named tobe.txt consists of the following text (where _ means blank and $ means an end-of-file character):

To_be_or_not_to_be$

Then, this text file contains the following suffixes (at positions 0–17):

0:To_be_or_not_to_be$
1:o_be_or_not_to_be$
2:_be_or_not_to_be$
3:be_or_not_to_be$
4:e_or_not_to_be$
5:_or_not_to_be$
6:or_not_to_be$
7:r_not_to_be$
8:_not_to_be$
9:not_to_be$
10:ot_to_be$
11:t_to_be$
12:_to_be$
13:to_be$
14:o_be$
15:_be$
16:be$
17:e$

If we ignore all suffixes not starting with a letter, we are left with 13 suffixes. We can now sort these suffixes in alphabetic order. When comparing strings, we do not differentiate between upper and lower case letters. This is why the suffix to_be can be placed before To_be or... We also ignore non-letter characters, this is why the suffix To_be or... is placed before t_to... Altogether, we get the sorted set of suffixes shown on the next page.
16: be$
3: be_or_not_to_be$
17: e$
4: e_or_not_to_be$
9: not_to_be$
14: o_be$
1: o_be_or_not_to_be$
6: or_not_to_be$
10: ot_to_be$
7: r_not_to_be$
13: to_be$
0: To_be_or_not_to_be$
11: t_to_be$

We can now represent in a fairly compact way an efficient search structure for all suffixes by storing their positions in a suffix array, which can easily be written to a file for future use:

16 3 17 4 9 14 1 6 10 7 13 0 11

If we use this array to search for the string "otto", a binary search will give the following result

compare at pos 1: otto > o_be_or_not_to_be
compare at pos 7: otto < r_not_to_be
compare at pos 6: otto > or_not_to_b
compare at pos 10: otto = ot_to_be

and we find the query string at position 10.

The programming assignment

The assignment consists of two parts:

1. Write a program that takes a text file as input and produces a file containing a sorted set of positions for all suffixes starting by a character, as illustrated above. Note that the output file should only consist of integers. The generation should be done in a fairly efficient way. Suggestion:

(a) Generate an array containing only the relevant suffixes. In the example above, this would be:

0 1 3 4 6 7 9 10 11 13 14 16 17

(b) Sort this array with some sorting algorithm. Use your own comparison function which takes as input two positions in the text.

2. Write a program that takes as input a text file and a suffix file, and which lets the user write query strings. The program shall read query strings from standard input, one per input line. For each query string, the program shall return the number of hits and, if the number of hits is at most 10, their positions. When searching for a string, non-letter characters should be ignored both in the query string and in the text file. Also, no difference should be made between upper and lower case characters. The input string Q cannot be searched as it is used to end the session.
An example of a session using the second of the programs is shown below:

> mySearch tobe.txt tobe.suf
Welcome to my free text searching in the file tobe.txt.
Write a query string (end by Q): TO
2 hits found at 13 0
Write a query string (end by Q): o
4 hits found at 14 1 6 10
Write a query string (end by Q): Tobbe
No hit
Write a query string (end by Q): Beo
1 hit found at 3
Write a query string (end by Q): Q
Goodbye!
>

Note
Your program should work efficiently and correctly when given a big text file (e.g., 10MBytes) as input.