Interview Programming Assignment

1. Assume you are given a very large file (> 10G) that contains lines of text. You need to write a program that reads data from the input file and creates two output files:
   1. File 1: each line should print line number with total word count for that line
   2. File 2: unique words (case insensitive) across input file, along with its count

For example:  
First few lines of input file:  
 This is a sample  
 Hacker  
 A this hacker  
  
File 1:  
 1 4 (4 words in line 1)

2 1 (1 word in line 2)

3 3 (3 words in line 3)

File 2:  
 This 2

is 1

a 2

sample 1

hacker 2  
  
You are allowed to use open-source libraries for data structures, such as linked-list, AVL trees and such. Use a programming language that you think fits the need. Provide your own analysis of pros and cons of the data structure you use, time complexity, etc.

Objective is to optimize the execution. At the end of the execution, your program should print the total execution time as well. Your program should scale with more CPUs, i.e., your program should run faster as more CPUs are available.

Attached is an executable ‘createfile’ that you can use to create a large input file. This executable will only run on Linux x86-84 systems. It will create ~10G file (it will take minutes depending on your system). If you have a different OS, you need to first create a program that can generate ~10G file with lines of text.

1. This problem is a variation on the theme of scrabble. You are given two files:

\* A dictionary that contains one word per line. (You can download

this test dictionary.)

\* A "value file" that specifies the values of letters and—here's

the catch—arbitrary strings. Each line of this file contains a letter

or string, some whitespace, and the corresponding point value. For

example:

a 1

q 9

aa 57

qu 12

…

The value file need not list all letters of the alphabet. Letters that

are not listed are defined to have value 0. A word's point value is

the sum of the point values of matching strings in the value file. To

find the word's matching strings, we scan the word from left to right

and identify the longest string that matches the as-yet unmatched part

of the word. For example, given the following value file:

a 1

c 1

e 1

h 1

k 1

r 1

ha 5

ck 5

er 5

hac 15

the value of the word "hacker" is 15 ("hac") + 1 ("k") + 5 ("er") =

21. Similarly, given the value file:

ck 5

ker 15

the value of the word "hacker" is 0 ("h", "a") + 5 ("ck") + 0 ("e",

"r") = 5. Note that the high-value string "ker" does not match because

the "k" in "hacker" has already been matched by "ck" in the

left-to-right scan.

Write a program that takes the names of a dictionary and a value file

on the command line, and prints the highest-value word(s) and their

values, one per line, like this:

ck 5

ker 15

You can use any programming language, but avoid using any in-build data structure that language might provide. Try to implement your own data structure for this problem. Provide your own analysis of pros and cons of the data structure you use, time complexity, etc.

3. IP address clustering

Consider a file that lists IP addresses and the number of

packets seen from each one, one per line, like this:

192.168.3.3 100

10.2.0.9 150

10.1.0.1 120

192.168.5.5 100

10.1.0.2 130

192.168.1.1 100

10.2.0.8 100

192.168.4.4 100

192.168.2.2 100

Given such a list of IP addresses and a threshold T between 1 and 100,

we want to group all the IP addresses into the smallest (i.e., longest

CIDR prefix) non-overlapping subnets that each account for at least T%

of total traffic.

Write a program that takes the name of the IP address file and the

integer T on the command line, and produces the desired list of

subnets and the percentage of traffic that each accounts for, rounded

to the nearest integer and sorted in order of decreasing traffic

volume. If two subnets have the same traffic volume, they should be

sorted in order of increasing IP address. For example, given the IP

traffic data above and T = 25, we would obtain:

./clustering config-clustering-small.txt 25

192.168.0.0/22 300

10.1.0.0/30 250

10.2.0.8/31 250

Your program should scale to an input file with ~100,000 IP addresses. There are two different input files that your program should run against.

config-clustering-small.txt

config-clustering-big.txt