- Use 2 or more threads to read directory entries and analyze the word frequencies of text files,

then compute the Jensen=Shannon distance between the word frequencies of pairs of files

UI:

- will be given the names of 1 or more files and directories

- up to four OPTIONAL arguments specifying program parameters

- file names/directories are used to determine the set of files to analyze

- file names given as arguments are added directly to the set of files

- for each directory, add any files whose names have a particular suffix to the set

-> recursively traverse any subdirectories

- for each (unordered) pair of files in analysis set, output the JSD between their word frequencies

Table 1: Parameters specified by options

Parameter Option Argument Type Default

Directory threads -dN Positive integer 1

File threads -fN Positive integer 1

Analysis threads -aN Positive integer 1

File name suffix -sS String ".txt"

- Comparisons are printed in decreasing order of combined word count(total number of words in both files)

-> Note: that pairs are unordered, so the output will include comparisons of

foo and bar/baz.txt or bar/baz.txt and foo, but not both.

- Take 1 or more regular arguments, MUST be the name( or path to) a file/directory

-> Each file name is added to the set of files to examine

-> Each directory name is added to the set of directories to traverse

- For each directory in the set...

-> read through its entries

-> any file whose name ends with the file name suffix is added to the set of files to examine

-> any directory is added to the set of directories to traverse

-> entries that are NOT directories and do not end with specified suffix are IGNORED

EXCEPTION: Any directory whose name begins with a period is IGNORED

- OPTIONAL AREGUMENTS

- take up to 4 optional arguments, which set program's parameters

-> any argument that begins with a "-" specifies an Option

- first character following dash indicates which Parameter is being specified, rest gives value for that Parameter

- require all optional arguments to occur BEFORE the regular arguments in argument list for simplicity

-> or allow to be intermixed

-> require optional args occur at most once, or allow later ones to override earlier ones

- MUST allow options to be given in any order

- MUST detect any invalid optional arguments and halt with an error message

-> Possible errors are an invalid option (e.g., -x), or an invalid or missing argument (e.g., -f0 or -d)

Note: The file name suffix may be an empty string, which would be indicated by the option -s

OPERATION:

Table 2: Word frequency distribution example

Text: I can’t understand thieves’ cant.

Word Occurrences Frequency

cant 2 0.4

i 1 0.2

thieves 1 0.2

understand 1 0.2

- For each pair of files in this set, it will compute the JSD between the distributions for those files

- frequency of a word is the number of times the word appears divided by the total number of words

- word frequency distribution (WFD) for a file is a list of every word that occurs ATLEAST ONCE in the file

along with its frequency

-> Ignore rounding errors, the frequencies for all the words will add up to 1

- MUST read the file and determine what words it contains

DEFINE WORD: a sequence of word characters, where word characters include letters, numbers, and hyphens

-> separated by whitespace characters

-> other characters, such as punctuation, are ignored

- keep a list of every word found in the file and its count

- each time a word is encountered, look for it in the list and increment its count or add it with count 1

- Once every word has been read, divide the count for each word by the total number of words

- words are CASE INSENSITIVE

- convert words to upper/lower case while reading

DATA STRUCTURE:

- need to maintain a list of mappings between words and counts

- linked list is a good balance of efficiency and simplcity

- reccomended to choose a structure that allows for iteration through the words in alphabetical order

- DO NOT assume maximum word length

-> word storage will be dynamically allocated (str buffer again? :D )

COMPUTING JENSEN-SHANNON DISTANCE (JSD)

- bunch of math characters that I cant type here refer to pdf :)

- Kullbeck-Leibler Divergence (KLD)

- if iterating through word list in alphabetical order its simple to compute JSD using

simultaneous iteration for both lists

- Keep a running total of the KLD for both files

- If you encounter a word appearing in both lists, compute the mean frequency and then add

the scaled frequencies to both running totals

- If word appears in one list but not the other, treat its frequency as 0 in the file where it does not appear

- once all words have been considered, compute the JSD

ah yes, quite simple really.

PROGRAM ORGANIZATION:

- operate in 2 phases

- collection phase recursively traverses directories and reads files to find the WFD for each requested file

- analysis phase computes the JSD for each pair of requested files

- BOTH phases are multithreaded and use synchronized data structures to perform tasks

COLLECTION PHASE:

- uses 2 groups of threads to find the WFD for each file in a set of requested files

- uses 2 synchronized queues to keep track of the directories and that need to be examined

-> nice figure in pdf

- Directory queue -> Contains a list of directories that have been seen but not yet traversed

-> synchronized queue (or stack) that may be bounded or unbounded

- File queue -> Contains a list of files that have been seen, but not yet examined

-> synchronized queue (or stack) that may be bounded or unbounded

- WFD repository -> once a file has been examined, its name, count of tokens, and WFD

is added to this structure

-> This may be any synchronized data structure.

- Main thread -> creates queues and repository

- for each regular argument it determines whether it is a file/directory and adds it to respective queue

- based on the program parameters it starts the requested number of file and directory threads

- once all the file threads have completed, it proceeds to the analysis phase

- Directory thread -> each dir thread is a loop that repeatedly dequeues a dir name from the dir queue

and reads through its directory listing

- Each directory entry is added to the directory queue

- Each non-directory entry that ends with the file name suffix is added to the file queue

- A directory thread will finish when the queue is empty and all other directory threads have

finished or are waiting to dequeue

- File threads -> Each file thread is a loop that repeatedly dequeues a file name from the file queue,

tokenizes the file and computes its WFD, and adds the data for that file to the WFD repository

- A file thread will finish once the queue is empty and all directory threads have finished

Note: working directory is shared by all threads

-> directory threads cannot use chdir()

-> instead, concate directory name and the file/subdirectory name to get a path

-> assume all file/dir names given as arguments are relative to the working directory

Note: need to customize synchronized queue somewhat

-> when directory queue is empty, keep track of the number of threads waiting to dequeue

-> once the last thread tries to dequeue, you can be certain no new directories will be found

-> set a flag indicating that traversal has ended and wake up directory threads so they can terminate

-> file threads should terminate if the file queue is empty and traversal has ended(no more files will be added)

Error Conditions:

- If any file/dir cannot be opened report an error and continue processing

- It is sufficent to call perror() with the name of the file/dir to report the error

-> exit with EXIT\_FAILURE when the program completes

- For unexpected errors, report error and terminate immediately

-> e.g. failure to create threads or lock and unlock mutexes

ANALYSIS PHASE:

- computes JSD and combined word count for each pair of files listed in the WFD repository

- Divides work among one or more analysis threads, as indicated by the optional argument

- Each thread is given a portion of the file pairs and computes the JSD for each pair

- Once all comparisons have been made, the main thread will sort the list of comparisons in descending order

of combined word count and then print the results

Data Structure:

- For each comparison we will need the names of the files being compared, the combined word count, and JSD

- If there are n files, there will be (1/2)n(n+1) comparisons

- Create an array of structs, have each thread write to non-overlapping portions of the array,

and finally sort the array using qsort()

Division of Labor:

- give each thread a roughly-equal portion of the comparison set to compute

-> if there are 30 files and 5 threads, each thread will compute 93 of the 465 comparisons

-> handle casses where number of comparisons does not divide evenly

- Another method, keep track of which comparison needs to be computed next

and have each analysis thread take the next comparison needed

-> reduces possibilit that some threads finish early while other threads have significant amounts of work

to do

Error Conditions:

- If collection phase found fewer than two files, report an error and exit

- Any numeric errors occurring in this phase indicate problems in collection phase

-> can check these using assert()