Design Plan - Intersect

James Viner

Project Summary

Intersect is a command-line utility that accepts two or more files as input and prints in-order all words delimited by whitespace that are found in each file at least once.

Features Targeted

Print Uppercase/Lowercase Duplicate Words

The overall structure for storing data is going to be a hash set, generated by hashing a lowercase version of each individual word; collisions are stored as nodes connected by a ’next’ field to form a linked list. Alternate printings of a given word (with punctuation or upper/lowercase letters that differ from the word’s first instance) will be stored as their own nodes attached to an ’alt_next’ field to form a second linked list. If the "-i" option is passed, in the case that a case-insensitive comparison finds a valid word for a given hash, a case-sensitive comparison will be run on that node and all of its alternate-spelling linked nodes and if a match is not found, a node for that spelling will be added to the ’alt_next’ linked list.

Ignore Trailing/Leading Punctuation

When storing words into the hash set, comparisons will be done case insensitively while ignoring any leading or trailing punctuation, but the storage of the word will include these characters for later printing. If the option is not present, these trailing or leading characters will have to be omitted from the print.

Architecture

Data

The overarching data structure for this project is a hash set that will contain pointers to nodes, with each storing one word from the first input file, a pointer to more nodes that make up a linked list of alternate spellings for that word, and a pointer to any nodes with the same hash, but different words. The reason for using this particular data structure is its O(1) time to lookup specific values, given that the most common operation for this program will be comparing words in the data structure.

Significant Functions

`hash_set *load_words(FILE *fo)`

This function takes in a file pointer, loads all of the whitespace-delimited words from it into a newly-created hash set, and returns a pointer to that hash set. Given that the only file according to the project specifications that is guaranteed to fit inside of memory is the first one, that one will be the one always passed to this function.

`void hash_set_add_word(hash_set *set, const char *word)`

In order to handle alternate spellings (capitalization and leading/trailing punctuation) of certain words, a node pointer called 'alt_next' is attached to each node in the hash set. If a node is going to be added, several things will occur: first, the case-insensitive, punctuation-excluded hash of that word will be used to index into the hashtable, then, each node (if present) in the linked list of words that have colliding hashes will have its word field compared against the word to be added (case-insensitively, with punctuation-excluded, again), then, if a match is found, case-sensitive, punctuation-included comparisons will be performed on each node in the linked list of alternate spellings for a given word, and it will be added if not already present. This allows intersect.c to use the set interface the same way regardless of which options are passed.

`static void hash_set_decouple(hash_set *set)`

This is a helper function for hash_to_sorted_list that indexes through the hash set and decouples nodes linked together by the 'next' field and puts orphaned nodes into the first available empty slot in the set.

`struct hash_set *hash_to_sorted_list(hash_set *set, int (*qsort_swap) (const void *, const void *)`

This function converts the hash set into a sorted list by first decoupling any horizontally-linked nodes (nodes connected by the ’next’ pointer rather than the ’alt_next’ pointer) into the empty slots in the array and then invoking qsort on the set using the passed sorting algorithm. This is intended to break normal hash set functionality and should only be called when an ordered list is desired over a hash set; this does not affect the way that the destructor is called or how it functions.

`void print_wrapper(hash_set *set, void (specialized_print) (hash_set *set))`

This function will utilize the hash_set_iterate function to print the loaded words to the console using one of several specialized print functions according to which command-line options were passed.

User Interface

The user will interface with the program by passing files on the command-line to be parsed for matching words and options to modify the printed output.

General Approach

1. Basic hash set infrastructure (adding nodes, freeing hash set, etc)

2. Loading words from file into hash set

3. Printing words to console

4. Hash set sorting infrastructure (decoupling nodes, calling quicksort, etc)

5. Printing **sorted** words to console