Assignment 5 – Calc Report

Max Ratcliff

CSE 13S – Spring 24

Purpose

The purpose of this program is to design an abstract data type to implement a dictionary like data type that takes a key and returns a value representing that key in order to process large amounts of data.

Questions

draft

Part I

in order to implement `list_remove` I will traverse the list to find the node to remove and then update the pointer of the previous node to point to the next node instead of the node to remove

in order to implement `list_destroy` I will traverse the list backwards freeing each node until I reach the head which I will also free and then set the pointer to NULL. I can then check to make sure the memory was actually freed by running valgrind

Part II

To ensure efficiency I will have a large number of bins so that there are minimal items in each and accessing each item will be faster

Part III

for now I have no plans on changing the hash function and will just use the one provided, but I will use a relatively large number of bins to ensure minimum collision handling

uniqq

to implement `uniqq` I will treat each line as a key and hash it to place each line in a bucket, then I should be able to return the number of buckets which should also return the number of unique lines

final

Part I

to ensure all the memory was cleaned up I iterated backwards through the list freeing every node and then freed the final pointer, I checked that everything was cleaned properly by running valgrind and making sure the allocs matched the frees.

Part II

to optimize the linked list I implemented a tail pointer and removed the while loop so that the whole list doesnt have to be iterated through everything something needs to be added.

part III

It speeds up as I add more buckets until the returns start to become mininal. I chose 1000 buckets just to make sure that it could handle very very large inputs

0.0.1 uniqq

to test my code I passed a variety of inputs and compared the output of uniqq to the output of uniq — wc I passed inputs using the '¡' operator to pass a file through stdin.

How to Use the Program

this program can be treated as a dictionary and used for a variety of purposes, after including it in a C file with include "hash.h", to use uniqq first compile with 'make' and then run with .
uniqq and provide input to stdin, either by typing it or passing a file by running '.
uniqq ¡ inputfile'

Program Design

implementation of an item is in item.h and item.c which is used in linked list to make a linked list of items, the implementation is in ll.h and ll.c, the linked lists are used in the implementation of a hashtable wich is an array of linked lists and is implemented in hash.h and hash.c. we use uniqq.c as an example of how you can use hash.c to implement a program to count unique lines

Pseudocode

Give the reader a top down description of your code! How will you break it down? What features will your code have? How will you implement each function.

Function Descriptions

For each function in your program, you will need to explain your thought process. This means doing the following

0.0.2 list_add

- Inputs: ptr to a linked list, ptr to am item to add
- Output: updated linked list and a success boolean
- adds an item to the end of a linked list and updates the tail

0.0.3 list_destroy

- Inputs: double ptr to a linked list
- Output: all memory freed and ptr set to null
- frees all the allocated memory of the linked list and sets the original pointer to null
- initialize a node to the start of a linked list and a node to null itterate until the end of the list and for each node update the null node to the next node and free the current node and then update the current node to me the next node. at the end of the loop set the head of the linked list to null and free the pointer.

0.0.4 list_remove

- Inputs: ptr to a linked list, ptr to a comparison function, ptr to an item to remove
- Output: updated linked list
- delete an item from a linked list
- initialize a current node and previous node, loop through list comparing each item until the correct item is found, then set the previous node to the next item of the current notde and free the current node

0.0.5 hash_create

- Inputs:
- Output: initialized array of "BUCKET" number linked list
- initializes an array of type hash table where each index is an empty linked list

0.0.6 hash_put

- Inputs: ptr to a hashtable, ptr to a key, ptr to a value to put into the table
- Output: updated hashtable
- hash the key and choose a bucket to put it into, create a value with key and value, add it to the list at the index of the hashed key

0.0.7 hash_get

- Inputs: ptr to a hashtable, ptr to a key
- Output: value of the item from the key
- hash key and find the item in the linked list corresponding to the bucket at the key

0.0.8 hash_destroy

- Inputs: double ptr to a hashtable
- Output: ptr to hashtable set to null
- iterate through each bucket destroying the linked list at each bucket, then free the hashtable and set its pointer to null