Assignment 5 – Towers Report Template

Nithin Duvvuru

CSE 13S – Winter 24

Purpose

Audience for this section: Pretend that you are working in industry, and write this paragraph for your boss. You are answering the basic question, "What does this thing do?". This section can be short. A single paragraph is okay.

Do not just copy the assignment PDF to complete this section, use your own words.

This assignment introduces us to more abstract data types through linked lists, associative arrays, and hash tables. We are meant to implement and optimize these structures, emphasizing memory management and performance. They develop a hash table API, optimize for efficiency, and create a real-world application, "uniqq," to count unique words.

Questions

Please answer the following questions before you start coding. They will help guide you through the assignment. To make the grader's life easier, please do not remove the questions, and simply put your answers below the text of each question.

- In Part I, you implemented garbage collection—routines that clean up dynamically-allocated memory. How did you make sure the memory was all cleaned up? How did you check?

  Memory cleanup will be ensured through implementing the list_destroy function, which traverses the linked list, freeing allocated memory for each node. To confirm memory cleanup, Valgrind will be used to detect and address any memory leaks or issues.

- In Part II, you made a major optimization to the linked list optimization. What was it, and why do you think it changed the performance of bench1 so much?

  A major optimization to the linked list could be implementing a more efficient insertion algorithm. This change likely will improve performance significantly by reducing the time complexity of insertion operations, making it faster to add elements to the list.

- In Part III, you implemented hash tables. What happens to the performance of bench2 as you vary the number of buckets in your hash table? How many buckets did you ultimately choose to use?

  Performance improves as the number of buckets increases initially, but eventually plateaus. We ultimately chose a number of buckets that provided optimal performance, balancing memory usage and lookup efficiency based on tests.

- How did you make sure that your code was free from bugs? What did you test, and how did you test it? In particular, how did you create inputs and check the output of uniqq?

  Code functionality is going to be ensured through lots of testing. Unit tests will validate individual functions, while integration tests will assess module interactions. Boundary tests will handle extreme inputs. For uniqq, diverse input files are generated and outputs will be compared against expected results and Valgrind will be used for memory leaks.

Testing

List what you will do to test your code. Make sure this is comprehensive. [1] Be sure to test inputs with delays and a wide range of files/characters.

Testing will include unit tests for individual functions, integration tests for module interactions, and boundary tests for extreme inputs. Diverse input files will validate uniqq outputs. Automated scripts and manual inspection, will ensure code works as specs intend.

How to Use the Program

Audience: Write this section for the user of your program. You are answering the basic question, "How do I use this thing?". Don't copy the assignment exactly; explain this in your own words. This section will be longer for a more complicated program and shorter for a less complicated program. You should show how to compile and run your program. You should also describe any optional flags or inputs that your program uses, and what they do.

To show "code font" text within a paragraph, you can use `\lstinline{}`, which will look like this: `text`.

For a code block, use `\begin{lstlisting}` and `\end{lstlisting}`, which will look like this:

```
Here is some code in lstlisting.
```

And if you want a box around the code text, then use `\begin{lstlisting}[frame=single]` and `\end{lstlisting}`

which will look like this:

```
Here is some framed code (lstlisting) text.
```

Want to make a footnote? Here's how.[2]

Do you need to cite a reference? You do that by putting the reference in the file `bibtex.bib`, and then you cite your reference like this[1][2][3].

First, some command-line flags let the user control the program behavior such as -h for help or -v for verbose output. After execution, uniqq will count unique lines or words in the text file, displaying the result.

Program Design

Audience: Write this section for someone who will maintain your program. In industry you maintain your own programs, and so your audience could be future you! List the main data structures and the main algorithms. You are answering the basic question, "How is this thing organized so that I can have a chance of fixing it?". This section will be longer for a more complicated program and shorter for a less complicated program.

[1] This question is a whole lot more vague than it has been the last few assignments. Continue to answer it with the same level of detail and thought.

[2] This is my footnote.

The main data structures include linked lists for managing key-value pairs and hash tables implemented with arrays of linked lists. Algorithms involve hashing for efficient key mapping, collision resolution techniques, insertion and lookup strategies, and memory management for memory allocation and deallocation to prevent memory leaks and optimize resource usage.

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.

The code will be broken down into sections for linked lists, hash tables, and application logic. Linked list functions will handle node insertion, deletion, and traversal. Hash table functions will implement hashing, collision resolution, and key-value pair management. Application logic will include input parsing, error handling, and unique word counting.

The linked list functions will be implemented using a node-based structure where each node contains a key-value pair. Insertion will involve creating a new node and updating pointers to maintain the list's integrity. Deletion will require adjusting pointers to bypass the node to be removed. Traversal will involve iterating through the list, accessing each node's data as needed.

The hash table, will use an array of linked lists, with each list storing key-value pairs hashed to the same index. A hash function will be used to map keys to indices, aiming to distribute entries evenly across the array. In case of collisions, chaining will resolve conflicts by appending colliding key-value pairs to the linked list at the corresponding index.

The logic will include parsing input files to extract words or lines, then inserting them into the hash table. Error handling will involve checking for memory allocation failures and memory cleanup. Finally, unique word counting will involve iterating through the hash table and tallying unique entries.

Function Descriptions

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

- The inputs of every function (even if it's not a parameter)
- The outputs of every function (even if it's not the return value)
- The purpose of each function, a brief description about a sentence long.
- For more complicated functions, include pseudocode that describes how the function works
- For more complicated functions, also include a description of your decision making process; why you chose to use any data structures or control flows that you did.

Do not simply use your code to describe this. This section should be readable to a person with little to no code knowledge. **DO NOT JUST PUT THE FUNCTION SIGNATURES HERE. MORE EXPLANATION IS REQUIRED.**

```
Function: create_linked_list()
Outputs: Pointer to the head of the created linked list
Purpose: Creates an empty linked list.

Function: insert_node()
Inputs: Pointer to the head of the linked list, key and value to be inserted
Outputs: Pointer to the head of the updated linked list
Purpose: Inserts a new node with the provided key-value pair at the beginning of the linked list.
```

```
Function: delete_node()
Inputs: Pointer to the head of the linked list, key to be deleted
Outputs: Pointer to the head of the updated linked list
Purpose: Deletes the node with the specified key from the linked list.

Function: search_node()
Inputs: Pointer to the head of the linked list, key to be searched
Outputs: Value corresponding to the key if found, otherwise NULL
Purpose: Searches for a node with the specified key in the linked list and returns its value.

Function: list_remove()
Inputs: Pointer to the head of the linked list, key to be removed
Outputs: Pointer to the head of the updated linked list
Purpose: Removes the node with the specified key from the linked list.

Function: list_destroy()
Inputs: Pointer to a pointer to the head of the linked list
Outputs: None
Purpose: Deallocates memory for all nodes in the linked list and sets the head pointer to NULL

Function: hash_create()
Inputs: None
Outputs: Pointer to the created hash table
Purpose: Allocates memory for and initializes a hash table data structure.

Function: hash_put()
Inputs: Pointer to the hash table, key, and value to be inserted
Outputs: Boolean indicating success or failure
Purpose: Inserts a key-value pair into the hash table. If the key already exists, updates its value
    .

Function: hash_get()
Inputs: Pointer to the hash table, key to be searched
Outputs: Pointer to the value associated with the key, or NULL if key not found
Purpose: Retrieves the value associated with the specified key from the hash table.

Function: hash_destroy()
Inputs: Pointer to the hash table
Outputs: None
Purpose: Deallocates memory for the hash table and its contents.
```

Results

Follow the instructions on the pdf to do this. In overleaf, you can drag an image straight into your source code to upload it. You can also look at `https://www.overleaf.com/learn/latex/Inserting_Images`

References

[1] Wikipedia contributors. C (programming language) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/C_(programming_language), 2023. [Online; accessed 20-April-2023].

[2] Robert Mecklenburg. *Managing Projects with GNU Make, 3rd ed.* O'Reilly, Cambridge, Mass., 2005.

[3] Walter R. Tschinkel. Just scoring points. *The Chronicle of Higher Education*, 53(32):B13, 2007.