CST2550 - SOFTWARE ENGINEERING MANAGEMENT AND DEVELOPMENT

Coursework 2 – Take Away Restaurant System Report

Written by:

Bhavna Chummun | M00942163

Shreya Dosieah | M00953441

Harshvardhan V Doyal | M00953762

Sneha L Gunput | M00939847

Nathan A Kagoro | M00910099

Submitted on:

21st April 2024

Tutors:

Mr. Aditya Santokhee

Mr. Karel Veerabudren

Table of Contents

Table of Figures

Table of Tables

Introduction

Project Description

The efficiency of food service operations, especially in terms of take-away restaurants, heavily relies on the effectiveness of the menu management system. To address this issue, it is imperial to have a flexible and user-friendly system which is able to properly handle taking orders and managing the menus. In response to these demands, the development of an intuitive menu management system has been carried out which aims to significantly facilitate the taking order process with greater accuracy and better customer service.

The system makes use of a hash table to store and retrieve menu items, allowing the restaurant staff to take orders and edit the menu. It enables the staff to not only browse and search through the different menus, but also add and remove menu items in order to modify the menus according to circumstantial changes or changing customer preferences.

Report Layout

The restaurant system will be the focal point of this report. After having described the restaurant system and knowing its purpose, it is important to discuss the design of the system, including justification of choosing a hash table as data structure and analysis of algorithms using pseudocode. Following the design section, there is a testing section whereby the testing approach is assessed along with a table of test cases. Lastly, the system is summarized and critically analyzed to find any limitations and their causes, as well as how these would be avoided in the future.

Design

Justification of Data Structures and Algorithms

A Hash Table was the chosen data structure, and the separating collision resolution technique was used, combined with a hash function to store and retrieve the menu items effectively.

A hash table is a data structure that stores data in an associative manner, using key-value pairs. It is designed to enable fast retrieval of data by using a hash function to compute an index into an array, where the value corresponding to a given key is stored. The hash function takes a key and produces an index in the array, which is used to find the appropriate bucket where the key-value pair resides. (PCMAG, n.d.) (Khalilstemmler, 2022)

Separate chaining is a collision resolution technique used in hash tables. A collision occurs when two different keys produce the same index after being processed by the hash function. Since a hash table has a finite number of buckets, collisions are inevitable, especially as the number of stored elements grows. (Cincotti, 2022)

Using hash table makes it suitable for a dynamic environment where menu items need to be frequently searched, added and modified. Separate chaining involves using linked list at each index of the hash table's underlying array to store colliding elements, allowing multiple values to exist at the same index without overwriting each other.

A hash table, which is in a class, has been used to implement the Restaurant Takeaway system in order to store and retrieve the menu items. Additionally, several functions have been used to perform different tasks such as insertion, sorting, and retrieval of menu items. Menu items are loaded from a CSV file and changes are saved back to it through the program, ensuring that menu data is persistently stored between program executions.

Algorithm Analysis

1. Input Menu

```
function loadItemsFromCSV(filename) {
  file = openFile(filename)
  if file is not open {
    print "Error opening file: " + filename
    showMenu()
  } else {
    print "File input successful"
  }
  while (line = readLineFromFile(file)) {
    split line by ','
    if successfully split {
      name = first part of line
      ingredients = second part of line
      price = third part of line
      insertMenuItem(name, ingredients, price)
    } else {
      print "Failed to parse line: " + line
    }
  }
  closeFile(file)
}
```

Figure 1: Add Menu Pseudocode

Function: loadItemsFromCSV
Purpose: Loads items from a CSV file into the system.
Operations:
Read each line: O(1) per line
Split and insert each line: O(1) per operation
Overall Complexity: O(n), where n is the number of lines in the CSV file.
Justification: Each line is processed individually and independently, leading to a linear relationship between the number of lines and the execution time.

2. Display Menu

```
function displayMenuItems() {
  for each index i in tableSize {
    if table[i] is not empty {
      for each item in table[i] {
        print "Item Key: " + i
        print "Item Name: " + item.name
        print "Ingredients: " + item.ingredients
        print "Price: " + item.price
      }
    }
  }
}
```

Figure 2: Display Menu Pseudocode

Function: displayMenuItems
Purpose: Displays all the menu items stored in the data structure.
Operations:
Iterate through all elements in the data structure: $O(1)$ per element
Print details for each item: $O(1)$ per item
Overall Complexity: $O(n)$, where n is the total number of items in the data structure.
Justification: Each item is retrieved and shown once, resulting in a direct link between the number of items and execution time.

3. Search Menu

```
function searchMenuItem() {
  print "Enter the item name or ingredients you want to search for: "
  keyword = readLineFromConsole()
  keyword = toLowercase(keyword)
  found = false

  for each index i in tableSize {
    for each item in table[i] {
      if toLowercase(item.name) contains keyword or
        toLowercase(item.ingredients) contains keyword
      {
        print "Item Found"
        print "Item Key: " + i
        print "Item Name: " + item.name
        print "Ingredients: " + item.ingredients
        print "Price: " + item.price
        found = true
      }
    }
  }

  if not found {
    print "Item not found in the menu."
  }
}
```

Figure 3: Search Menu Pseudocode

Function: searchMenuItem
Purpose: Searches for a menu item by name or ingredients within the stored data.
Operations:
Linear search through each item: $O(1)$ per item check
Comparison of search keyword against item properties (name, ingredients): $O(1)$ per comparison
Overall Complexity: $O(n)$, where n is the total number of items in the data structure.
Justification: The function checks each item until it finds a match or exhausts the list, therefore the number of operations increases linearly with the number of things.

4. Take Order

```
function takeOrder(menu)
  order = empty vector of strings
  choice = 0
  total = 0.00

  repeat
    print "----------Order Menu----------"
    print "1. Add Menu Item to order."
    print "2. Remove Menu Item from order"
    print "3. View Order"
    print "4. View Menu"
    print "5. Calculate bill and end order"
    print "6. Exit"
    print "Enter your choice (1-6): "

    read choice from console
    while choice is not a valid option (1-6)
      print "Invalid choice. Please enter a number between 1 and 6: "
      read choice from console

    if choice is 1
      repeat
        print "Enter item to add to order: "
        read input from console
        if menu.isPresent(input)
          add input to order
          print "Item added successfully."
        else
          print "Item not found."
      until found is true

    else if choice is 2
      print "Enter item to remove from order: "
      read input from console
      find input in order
      if found
        remove input from order
        print "Item removed successfully."
      else
        print "Item not found."

    else if choice is 3
      print "-------------CURRENT ORDER-------------"
      for each item in order
        print item
      print "---------------------------------------"

    else if choice is 4
      print "---------------------------------------"
      menu.displayMenuItems()
      print "---------------------------------------"

    else if choice is 5
      total = 0.00
      for each item in order
        price = menu.getPrice(item)
        total = total + price
      print "Total Price is " + total
      wait for 5 seconds
      print "Exiting..."
      wait for 2 seconds
      clear console
      exit the program

    else if choice is 6
      print "Exiting..."
      wait for 2 seconds
      clear console

    else
      print "Invalid choice. Please try again."

  until choice is 6
end function
```

Figure 4: Take Order Pseudocode

Function: takeOrder
Purpose: Manages the ordering process, allowing users to add, remove items, and finalize the order.
Operations:

Adding/removing items from the order: $O(n)$ on average for search and update operations

Calculating total: $O(m)$, where m is the number of items in the order

Overall Complexity: $O(n)$, assuming n represents the larger of menu items or order items processed during the session.
Justification: Each interaction (add or delete) entails a linear search through the menu or order list, and the overall calculation is linear in relation to the order's contents.

5. *Add Menu Item*

```
function addMenuItems() {
  print "Enter the file name you want to edit: "
  filename = readLineFromConsole()

  print "Enter Menu Item: "
  name = readLineFromConsole()

  print "Enter Ingredients: "
  ingredients = readLineFromConsole()

  print "Enter Price: "
  price = readLineFromConsole()

  insertMenuItem(name, ingredients, price)

  file = openFileForAppending(filename)
  for each index i in tableSize {
    for each item in table[i] {
      writeToFile(file, item.name + "," + item.ingredients + "," + item.price)
    }
  }
  closeFile(file)
}
```

Figure 5: Add Item to Menu Pseudocode

Function: addMenuItems

Purpose: Adds a new menu item to the data structure and updates the CSV file accordingly.
Operations:
Insert new item into the data structure: $O(1)$
Rewrite all items to the CSV file: $O(n)$, where n includes the newly added item
Overall Complexity: $O(n)$, where n is the total number of items after insertion.
Justification: Although adding to the data structure is constant, the necessity to rewrite all things to the file dominates the time complexity, resulting in a linear relationship with the total number of items.

6. *Remove Menu Item*

```
function removeMenuItems() {
  print "Enter the file name you want to edit: "
  filename = readLineFromConsole()
  loadItemsFromCSV(filename)
  displayMenuItems()

  print "Enter the item key you want to remove: "
  key = readIntegerFromConsole()

  if key < 0 or key >= tableSize {
    print "Invalid key. Key must be between 0 and " + (tableSize - 1)
    return
  }

  if table[key] is empty {
    print "No items found at the given key."
    return
  }

  if size of table[key] is 1 {
    clear table[key]
    print "Item removed successfully."
  } else {
    print "Multiple items found at this key."
    print "Enter the name of the item to remove: "
    nameToRemove = readLineFromConsole()
    found = false
    for each item in table[key] {
      if item.name is equal to nameToRemove {
        remove item from table[key]
        found = true
        break
      }
    }

    if not found {
      print "Item name not found at the given key."
      return
    }

    print "Item removed successfully."
  }

  file = openFileForWriting(filename)
  for each list in table {
    for each item in list {
      writeToFile(file, item.name + "," + item.ingredients + "," + item.price)
    }
  }
  closeFile(file)
}
```

Figure 6: Remove Item from Menu Pseudocode

Function: removeMenuItems
Purpose: Removes an item from the data structure based on input and updates the CSV file.
Operations:Search and remove the item: $O(n)$ for linear search and removal
Rewrite all remaining items back to the CSV file: $O(n)$
Overall Complexity:$O(n)$, where$n$ is the number of items before removal.
Justification: Both searching for and deleting the item, as well as rewriting the file, are linear operations, therefore the total complexity stays linear.

Testing

Statement of Testing Approach

In order to ensure that the system is fully functional, efficient and meets the requirements of the restaurant, it is crucial to test it and confirm error-free interactions. Carrying out the different types of testing is important to mitigate the risk of potential errors. These include Unit Testing, Integration Testing, and System Testing where testing pieces of code, combining different functionalities and finally testing the program as a whole allows elimination of bugs from the start of the coding process to its end.

Catch2, a header only testing framework, has been used to write test cases which reflect the possible usage scenarios for the restaurant system. By simply downloading the header file and saving it with the other program files, Catch2 enabled calling and testing the functions (Molssi, n.d.).

Test Cases

The program is built to be almost airtight, containing a wide area of error-handling code to ensure that the user experience is seamless. To ensure that as many possible errors are caught, we ran a wide array of test scenarios using a combination of catch2 testing and manual input testing.

The first set of tests was running the program several times inputting only correct values to make sure that the program runs correctly if all inputs are as demanded. Following that, a series of standard catch2 tests are executed to ensure that there is appropriate error handling. Considering most functions in this program do not return anything, they have to be manually tested by inserting unorthodox inputs. Below are the results of both manual and catch2 tests:

Test	**Desired outcome**	**Test result**
Displaying empty menu	Informs the user that the menu is empty	Displays nothing
Inputting numbers in the lowercase function	Ignores the numbers	Test passed
Inputting spaces in the lowercase function	Ignores the spaces	Test passed
Inputting special characters in the lowercase function	Ignores the special characters	Test passed
Request price of non-existent menu item	Informs the user that the menu item does not exist	Test passed
Inputting nothing into the search function	Returns no program found	The program does not accept nothing as an input
Inputting characters before or after the search	Detects the text to be searched for and returns the correct search	Returns “No item found”

Adding pre-existing data to the menu	Locate that it is a duplicate and inform the user	Duplicates the data onto the menu
Remove data that has already been removed	Inform the user that the data does not exist	Test passed
Inserting special characters into the add	Ignore special characters and adds as normal	Test passed
Inserting special characters into the remove input	Ignore special characters and add/remove keywords entered	Returns “No item found”
Entering a non-existent file path into the add function	Inform the user that the path does not exist and rerun the code	Adds data to local table and carries onward
Entering a non-existent file path into the add function	Inform the user that the path does not exist and rerun the code	Program breaks

Table 1: Test Cases

Conclusion

Summary

In the lineament of the above, the development, implementation, and testing of the take-away restaurant system which uses a custom-designed hash table to manage, make searches to, and modify the menus. Briefly, the functionalities of the system are as follows:

1. Display the menus
2. Take customer orders and process payment
3. Search menus
4. Add items to menus
5. Remove items from menus

These are implemented with the use of the hash table due to its dynamic and anti-collision nature. Through the algorithm analysis, it can be seen that the chosen methods were efficient in tackling the development of the system given the linear complexity trend in the functions, leading to only a small number of errors.

Limitations and Critical Reflection

With relevance to the testing section, it can be seen that the highlighted errors all stem from a lack of robust validation. Although the program is not subject to syntax or compilation errors and correctly performs its core functionalities, these issues still pose a minor setback in terms of customer satisfaction and user-friendliness.

By not being able to search for multiple items, dissatisfaction may arise due to limited search function and time lost in searching. Moreover, the failure of the function in detecting intended search keyword also reduces the flexibility and leads to similar consequences as a limited search. To add on, the absence of duplication detection may tend to a messy and overfilled menu, also resulting in difficulty in comprehending the menu and reducing usability and efficiency.

Future Approaches

To be able to complete a similar project with a different and better approach in the future, the way to start will be to alleviate the above-mentioned issues by doing more thorough testing beforehand and pinpointing the slightest inconvenience for a more intricate application. Continuing from there, the existing functionalities may be further enhanced and expanded upon for a more advanced user experience. For instance, the search functionality can be improved by introducing filters and sorting options. By embracing a solution-oriented and forward-thinking approach, future projects can certainly surpass the current solution in terms of a minimal amount of limitations and better usability.

References

1. Carmen Cincotti (2022). *Separate Chaining FAQ - Hash Tables*. [online] Available at: https://carmencincotti.com/2022-10-03/separate-chaining-faq-hash-tables/ [Accessed 18 Apr. 2024].
2. khalilstemmler.com. (2022). *Hash Tables | What, Why & How to Use Them | Khalil Stemmler*. [online] Available at: https://khalilstemmler.com/blogs/data-structures-algorithms/hash-tables/ [Accessed on 18/04/2024]
3. Molssi. (n.d.). *The Catch2 package*. [online] Available at: https://education.molssi.org/cpp_programming/07-testing/index.html [Accessed: 19 April 2024]
4. PCMAG. (n.d.). *Definition of hash table*. [online] Available at: https://www.pcmag.com/encyclopedia/term/hash-table [Accessed on 18/04/2024]