EX3: Design Review

Due date: Friday April 29, 2022 at 11:59 pm PDT

Latest turn-in date: Monday May 2nd, 2022 at 11:59 pm PDT

Instructions:

High-level collaboration is allowed, but exercises are to be completed and submitted individually. Submit your responses to the “EX3: Design Review” assignment on Gradescope here: https://www.gradescope.com/courses/379339/assignments/2016783/. Make sure to log in to your Gradescope account using your UW email to access our course.

Two students in a 373 class have been given a design problem for their midterm 😊. They were given a scenario and asked to design a solution using the ADT and data structure of their choice. Below is the scenario those students were given and their design submissions. After reading the students’ designs, fill in the review questions on Gradescope analyzing the effectiveness of these designs.

1. Fridge Catalog

TA Rahul loves to cook, but often forgets what he has in his fridge when he’s at the grocery store. To solve this problem, he wants to catalog all of the ingredients in his fridge, and how much of each ingredient he has. That way, he can check what to buy once he’s at the store

You are tasked with designing a system for Rahul to use when he is at home and at the grocery store, implementing these methods:

- `addItem(ingredient, quantity)`: record in your system that Rahul currently has this ingredient in stock, and how much of it he has. If the ingredient is already in the system, add the two quantities together.
- `hasItem(ingredient)`: Check how much of a specific ingredient Rahul has at home

Use the following page of reference materials to select an appropriate ADT and data structure for your design.

List
get(index) return item at index
set(item, index) replace item at index
append(item) add item to end of list
insert(item, index) add item at index
delete(index) delete item at index
size() count of items

Queue
add(item) add item to back
remove() remove and return item at front
peek() return item at front
size() count of items
isEmpty() count of items is 0?

Stack
push(item) add item to top
pop() return and remove item at top
peek() look at item at top
size() count of items
isEmpty() count of items is 0?

Map/Dictionary
put(key, item) add item to collection indexed with key
get(key) return item associated with key
containsKey(key) return if key already in use
remove(key) remove item and associated key
size() return count of items

Data Structure Options

- Array - Indexed based storage with fixed capacity
- Linked List - Null terminated singularly linked nodes. Must describe fields of Node as part of design.
- Separate Chaining Hash Table - Hash table leveraging a Linked List for each bucket to resolve collisions
- Linear Probe Hash Table - Hash table leveraging open addressing to resolve collisions
- Binary Search Tree - Binary tree with binary search invariant applied
- AVL Tree - Binary tree with both binary search and balance invariants applied

Student A’s Design

1. What ADT best fits the required functionality for this scenario?

   List

2. Describe in 1 to 2 sentences why this ADT is a good fit for this scenario, including any assumptions about how users will behave in this scenario.

   This ADT will allow us to store all of the ingredients together in one list.

3. What Data Structure implementation of your chosen ADT is optimal for this scenario?

   Linked List

4. Implementation and Optimization

   a. Describe how your chosen data structure would be used to implement each of the functions of the program. (A few sentences per function).
      - For addItem(), we create a LinkedList Node that has two fields, the ingredient, and the quantity. We simply add this node to the end of our Linked List.
      - For hasItem(), we traverse the list until we find the item we wanted

   b. Describe which functions of the program are optimized by your selected data structure, e.g. What is the most efficient task your ADT accomplishes?
      - This data structure optimizes the addItem() function, as it runs in constant time, only ever adding to the end of the list.

   c. Describe which functions (if any) of the program are not optimized by your selected data structure, e.g. What tasks are not fully efficient with your ADT?
      - This data structure does not optimize the hasItem() perfectly, but it is still relatively fast.

5. Describe the best-case scenario for your design, including discussion of how this impacts each of the ADT’s functions.

   Best case scenario is when only one item is added to the list. This mean that you don’t need to travel very far when you search for that ingredient.

6. What is the Tight Big O of the best-case scenario for each of the ADT’s functions you described in part 5?

   For both functions, the runtime is O(1).

7. Describe the worst-case scenario for your design, including discussion of how this impacts each of the ADT’s functions.

   The worst-case scenario for this design is when you add a lot of items to the list, since you’ll have to search through a lot of nodes.

8. What is the Tight Big O of the worst-case scenario for each of the ADT’s functions you described in part 5?

   addItem() is still O(1), but hasItem() is O(n).

Student B's Design

1. What ADT best fits the required functionality for this scenario?
   Map

2. Describe in 1 to 2 sentences why this ADT is a good fit for this scenario, including any assumptions about how users will behave in this scenario.
   This ADT will allow us to map ingredients as keys and quantities as values. We are assuming that the order of the ingredients added doesn't matter

3. What Data Structure implementation of your chosen ADT is optimal for this scenario?
   Chaining Hash Table

4. Implementation and Optimization
   a. Describe how your chosen data structure would be used to implement each of the functions of the program. (A few sentences per function).
      - For addItem(), we create a key-value pair object using the ingredient as key and quantity as value. We will use a hash function on the key to generate an index in the array for it to be placed, adding this object to the chain at that index, resizing if needed. If the key is already there, we simply increase the corresponding quantity.
      - For hasItem(), we just check if the key is in the map by applying the same hash function and searching through the corresponding chain at that index. Then we check if it has a quantity greater than zero.

   b. Describe which functions of the program are optimized by your selected data structure, e.g. What is the most efficient task your ADT accomplishes?
      This data structure optimizes both the addItem() and hasItem() calls, as both would run in constant time by using the Chaining Hash Table's put() and get() calls.

   c. Describe which functions (if any) of the program are not optimized by your selected data structure, e.g. What tasks are not fully efficient with your ADT?
      This data structure optimizes both functions of the program. However, in the event that we reach a load factor of 1, we must resize by creating a new array, rehashing into it, which is a runtime of O(n).

5. Describe the best-case scenario for your design, including discussion of how this impacts each of the ADT's functions.
   Best case scenario for this design is when all ingredients added to the map hash to different values. This would mean that each bucket in the array contains only one ingredient, without any collisions.

6. What is the Tight Big O of the best-case scenario for each of the ADT's functions you described in part 5?
   For both functions, the runtime is O(1).

7. Describe the worst-case scenario for your design, including discussion of how this impacts each of the ADT's functions.
   The worst-case scenario for this design is if all ingredients hash to the same value, meaning they would all be placed within the same chain in the array, leading to n collisions, essentially creating a linked list with a length of n.

8. What is the Tight Big O of the worst-case scenario for each of the ADT's functions you described in part 5?
   This would slow both functions down to an O(n) runtime.