CSCI-201: Principles of Software Development - Fall 2020

PA2: Project Last Light

Concepts Covered:

- Arrays
- Java Classes
- Sorting
- Linked Lists
- Dictionaries
- Amortized Array Based Dictionaries
- JSON Formatting and Reading

Prelude

It's funny how we worried so much about artificial intelligence becoming self-aware and end up wiping out the human race back in the early 21st century. The premise of the cult-classic Matrix movies, Terminator, as well as constant warnings from the late Elon Musk and Jeff Bezos all revolve around the dangers of AI. As a result, AI stayed strictly in the consumer space and never made its way into the government or military.

However, the fourth industrial revolution could not be stopped and automation inevitably swept the world by storm. Gone are the days of soldiers risking their lives on the frontlines operating primitive machinery and weapons - a new breed of war machines were born and the way humans fought was forever changed. Patented and manufactured by Pharaoh Automated Products Incorporated, their war robots are packed with highly lethal weaponry designed to deliver maximum efficiency. Their mother-bots even have the ability to manufacture robot units in the field to replenish any combat losses on the fly. Mother-bots also have absolute control over the units they produced by default.

Since AI has been banned from the military, the new war machines are limited to rudimentary software that accepts commands remotely. Cybersecurity measures are of paramount levels to prevent any form of exploit or software intrusion. In the case of a severe communication disruption, the robots are programmed to return to their base at any cost to prevent capture, including activating their biomass conversion function if their energy cells become depleted and using lethal force against any hostile combatants standing in their way. A fleet of remote-controlled unhackable dumb killers with a failsafe that follows incredibly basic directives and avoiding the use of AI entirely - surely they cannot become self-aware and turn on us?

Fate, it seems, is not without a sense of irony.

We never learned the exact time of the incident because Pharaoh Automated Products Incorporated tried very hard to cover it up, but by the time world leaders became aware it was already too late to take any military action. During field testing, a corrupted network packet sent to the mother-bot caused an internal firmware glitch and quickly led to code corruption that turned on its robot-manufacturing functionality. Even worse, the corruption damaged its authentication firmware, causing the mother-bot to lose its ability to authenticate incoming commands and know which base it belonged to. With its ability to communicate damaged, the mother-bot started to follow its basic failsafe directive: return to base. Yet, it didn't even know which base to return to anymore and thus began to wander pointlessly. Any attempts of physical restraint were seen as a hostile act and met with lethal force. Any attempts at regaining software control were futile due to its cybersecurity measures. The only way to shut them down was via a decryption code that could only be brute-forced, which was way beyond the capabilities of a single private company. Pharaoh has made the ultimate war machine - an actively replicating, biomass-consuming, out-of-control killer robot that won't stop until the biosphere is completely destroyed.

In the midst of progressing global destruction, Project Last Light was conceived by world leaders to join forces and brute-force the decryption code necessary to bypass authentication and shut down the mother-bot. Since the mother-bot has absolute control over all units it produced by default, shutting down the mother-bot would also shut down every single rogue robot it produced. The project was named "Last Light" because if the project does not yield the decryption code in time, there will be no more humans, no more animals, and no more life on Earth. There will only be countless Pharaoh robots in hibernation, waiting to get their hands on more biomass to further replicate.

You are a member of Project Last Light and you are here to devise an ***amortized array-based dictionary*** *to aid the brute-force efforts.*

The Assignment

You are to create an amortized array-based dictionary from the given starter code. `public class Dictionary<AnyType extends Comparable<AnyType>> implements DictionaryInterface<AnyType>`
using a linked list of sorted arrays. Specifically, you will maintain a linked list of the following nodes:

```
private static class Node {
  private int power;
  private Comparable[] array;
  private Node next;
}
```

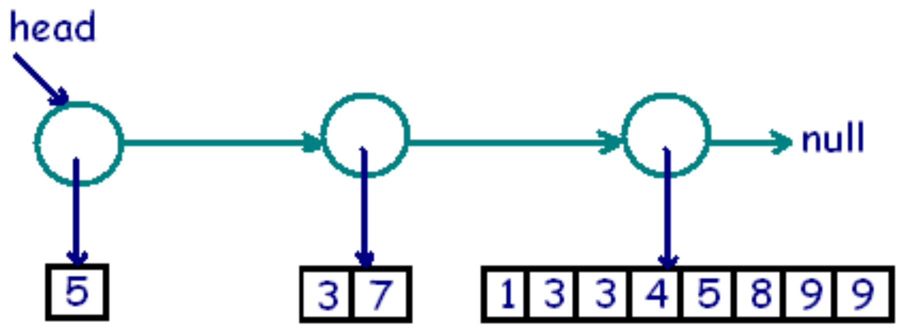
These nodes are organized in the ascending order of array sizes - there are no two nodes containing arrays of the same size. This is because if there were, they would be immediately merged, as is shown in the example in the next section.

Your main goal is to implement the functions described in `DictionaryInterface.java`:

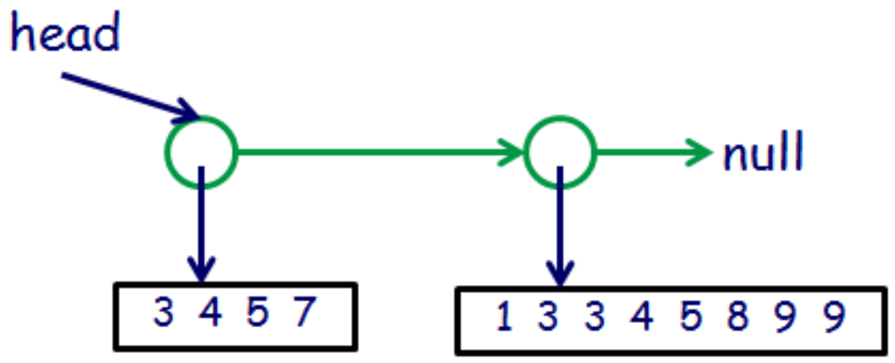
- `add()`:
  - You create an array of size one and insert it into the linked list. Next, you traverse the list and merge sorted arrays of the same size until at most one array remains of each size.
- `combine()`:
  - Combining two dictionaries is based on recursive merging sorted arrays of the same size. The procedure has two phases. In the first phase you merge the roots of two linked lists into one linked list that is sorted by array size in monotonically increasing order. In the second phase you merge arrays of equal size until at most one array remains of each size.
- `contains()`:
  - Since each array in the list is in sorted order, you run Java's binary search on each of them.
- `frequency()`:
  - This function reports the number of times a specific entry appears in your data structure.
- `remove()`:
  - This is the most cumbersome operation which can be implemented in more than one way. We advise you to follow the algorithm outlined by Nathan in his PPT presentation. In general terms, removing an item from an array of size `2k` results in splitting an array into `k` smaller arrays of sizes 1, 2, 4, and so on until `2k-1` Then, you traverse the list and merge arrays of the same size.
  - NOTE: This function should only delete one instance of the object to remove, as reptition is allowed in this data structure
- After it has been completed, you should create a terminal user interface for the program. This should be done in the Main.java class, and is better explained in the testing portion.

Amortized Array-Based Dictionary

One of the most important structures in computer science are the dictionary data structures that support fast insert and lookup operations. Dozens of different data structures have been proposed for implementing dictionaries including hash tables, skip lists, search trees and others. In this programming assignment you will implement a dictionary based on linked lists and sorted arrays. This structure combines a fast lookup on sorted arrays with ease of linked list-insertion. Note that a sorted array is good for lookups (think of a binary search) but bad for inserts. A linked list is good for inserts but bad for lookups (they can take linear time). The idea of this data structure is as follows. We have a linked list of arrays, where array `k` has size `2k`, and each array is in sorted order. However, there will be no relationship between the items in different arrays. Whether arrays are full, or empty is based on the binary representation of the number of items we are storing. For example, with 11 items our dictionary might look like this (11 = 1 + 2 + 8):



In general, we need at most `Ceiling(log(n))` arrays to store `n` items. How do we insert into this data structure? We create an array of size one and add to the linked list. Since each array must have a different length, insertion requires merging arrays of the same size. As an example, consider inserting 4 into the dictionary in the above figure. We get two arrays of size one, thus, we have to merge them. After merging, the dictionary will have two arrays of size two: `[4, 5]` and `[3,7]`. We have to merge them into an array of size four. The following figure demonstrates the final dictionary.



Note that the larger array stayed the exact same, this is because the newly inserted size 1 array merged with the other size 1, then with the size 2, thus resulting in a new array of size 4. If you were to insert another value, there would simply be a new size 1 array appended as the new head of the amortized array based dictionary.

In the worst-case we may merge all `O(log(n))` sorted arrays. Each pair of sorted arrays can be merged in linear time. The total cost model is the following: creating the initial array of size 1 costs 1, and merging two arrays of size `k` costs `2k`. The total cost of the above insert is 1+2+4 = 7. In the general case, the worst-case runtime complexity of a single insert is `O(n)`. We can prove (but we wont) that the amortized dictionary data structure has amortized cost O(log n) per insert, hence the name "Amortized Array-Based Dictionary" (AAD).

Testing

- We provide `TestingDriver.java` to help test your implementation. It will run some simple tests on your code and provide feedback.
- After you have tested your dictionary class, you are expected to create a runner (main) class. This class will essentially provide a menu to your program.
- The menu will need to provide for choice inputs for users.
    - 1: Get the frequency of a certain code(value in your dictionary). Essentially, given a certain value, output the number of times that that value appears in your data structure.
    - 2: Check if a code was guessed. Just determine whether or not the provided value is stored in the data structure.
    - 3: Remove a code: remove the selected value from your dictionary (if it exists).
        - Note: When calling remove here, it should remove EVERY value that is the same as the argument, not just the first.
    - 4: Quit: Quit the program.
- The "codes" that are referred to, are the decryption codes for the machines you are trying to regain control of. Essentially, these codes are the values stored in your Amortized Array-based Dictionary.
- Here is an example input to the program.
- The inputs are in the form of JSON (see `input.txt`).
- An example output of the program is attached (see `sample-output.txt`).
- **Note: It should be printed directly to your terminal, not to an output file.**

Starter Code

- Three files are attached: `TestingDriver.java`, `Dictionary.java`, `DictionaryInterface.java`.

Submission

- Export your Eclipse project and submit it on DEN. It is of paramount importance that you export correctly. ***If the graders cannot import your project or if it contains errors after importing, you will get a zero.***

Grading Criteria

- `add()` 25%:
  - 10%: Successfully creates array and inserts into list.
  - 15%: Successfully merges sorted arrays so that one array remains of each size.
- `combine()` 20%:
  - 5%: Utilize recursion when merging.
  - 5%: Merge two linked lists into one.
  - 5%: Have merged array sorted by array size (increasing order).
  - 5%: Merge arrays of equal size until at most one array remains of each size.
- `contains()` 10%:
  - 10%: Correctly runs binary search to return correct values.
- `frequency()` 20%:
  - 20%: Correctly outputs the number of occurences of the provided argument in the data structure.
- `remove()` 20%:
  - 10%: Successfully removes the designated item.
  - 10%: Successfully merge arrays of the same size created due to remove.
- Runner/Main Class 5%:
  - 5%: The runner class provides the correct outputs for each function.