Obligatory assignment report
INF-1101

Sera Madeleine Elstad

March 02, 2022

1 Introduction

For this assignment in INF-1101, two sets of ADTs will be implemented and compared. The ADT is known as an ordered set because it is based on element ordering and supports iterating over its elements in that order. The operations that should be supported by the set are:

- Adding an element to the set
- Getting the current size of the set
- Checking whether a specific element is contained in the set
- Getting the union, the intersection, and the set difference of the set and other sets
- Iterating over the elements of the set, in sorted order.

A spam filter will use the algorithm to filter out spam emails. This will be accomplished by constructing a set of spam-related words. To begin, some pre-code is provided.

2 Technical Background

2.1 The big-O complexity

When calculating the efficiency of an algorithm the Big O notation is used to describe the complexity of the algorithm, which in this case refers to how well the algorithm scales with the size of the dataset [1]. Figure 1 depicts how the complexities of various operations change as more elements are added.

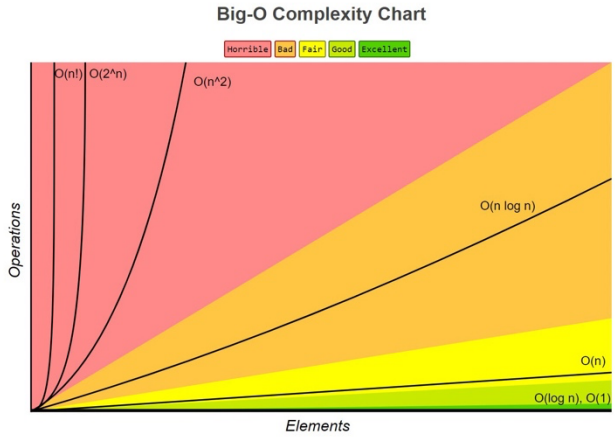

Figure 1, the complexity

2.2 Sets (ADT)

Sets are simply a group of distinct objects that can contain any group of items. Every item in the set is referred to as an element. A set is distinguished by the fact that it does not contain two elements that are identical.

2.2.1 Union, intersection, and difference

Image 1 shows how intersection, union, and difference between two sets work. The union of two sets (A and B) is a new set containing all the elements of the two original sets. If there are two elements in A and B that are equal, the new set will only contain one such element [2]. The intersection of two sets returns a new set containing all elements of A that also belongs to B [3]. To find the difference between two sets a new set that contains all the elements in A which is not in set B is created, and vice versa [4].

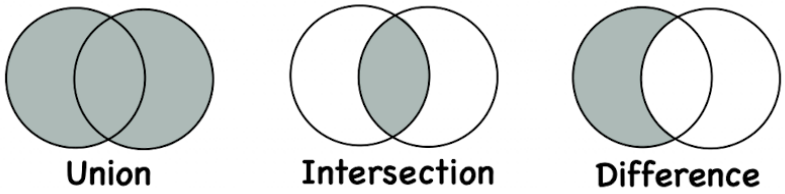

Figure 2,

The union, intersection, and difference between two sets A and B

2.3 Singly and doubly linked lists

Linked lists are a type of data structure that consists of a series of linked nodes, each of which stores some data as well as the address of the next node. The address of the HEAD is stored in the first node of a linked list, which tells the program where the list begins. A pointer at the end of the list points to NULL, indicating to the program that it has reached the end of the list.

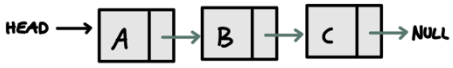

Figure 3, singly linked list

With a single liked list, the program only has the NEXT pointer, which allows the program to traverse forward (see figure 3). In many ways, doubly linked lists are equivalent to singly linked lists, but they also have a PREV node pointing to the previous node (see figure 4) [5].

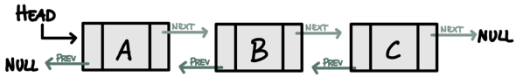

Figure 4, doubly linked lists

The complexity of inserting or deleting a node in a linked list, whether singly or doubly, is O(1), and the complexity of searching and accessing a node is O(n) [6]. Although their general complexity suggests they are similar, the doubly linked list is more complex and thus less efficient because adding and removing nodes requires multiple operations. This is because when adding or removing a node in a doubly linked list, both the NEXT and the PREV must be changed, whereas in a singly linked list, only the NEXT must be changed. Although doubly linked lists are more complicated, they are preferred when searching.

2.4 Array

An array is a data structure that arranges elements sequentially. An array is denoted by a memory address, which is the memory address of the first element. Accessing or overwriting elements in an array takes very little time; the time complexity is O(1). This is because, unlike a list, all elements in an array can be found by their index number, whereas a list must be traversed to find the given element. The complexity of inserting or deleting an element in an array is O(n) [7].This means that because of the index number, arrays are faster for accessing elements, but linked lists are faster for inserting or deleting elements.

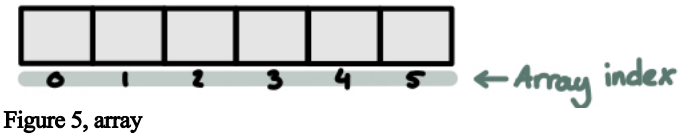

Figure 5, array

2.5 Sorting algorithms

2.5.1 Merge Sort

Merge Sort is based on the divide and conquer algorithm, which is a strategy for breaking down a large problem into smaller ones, solving the smaller ones, and then combining them. As illustrated in Figure 6, mega sort divides the array in half until it is left with a subarray of size one. Following that, the function combines one and one subarray, sorting it as it goes until the array is complete. Whether the array is sorted or not, the time complexity of Merge Sort is O (n log n) [8].

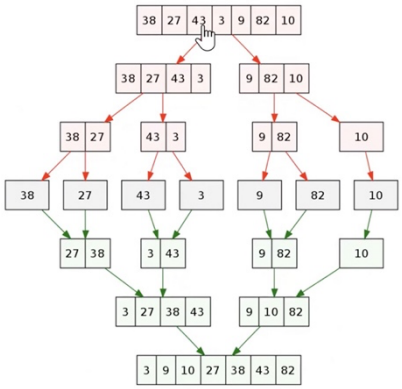

Figure 6, Merge Sort

2.5.2 Selection Sort

Selection Sort sorts an array by repeatedly finding the minimum element. To begin with, the first element is put as the minimum element. This element is compared with the next element. If the second element is smaller, they change places, and the minimum element is updated. If the element is bigger, the algorithm jumps to the next element and compares it with the first. This will be done until the algorithm reaches the last element of the array. The process that is described above is repeated until the program runs through the array one time without swapping. Selection Sort has Ω(n^2) as the best, the average is Θ(n^2) and the worst is O(n^2) [6].

3 Design and implementation

3.1 A set implemented as a linked list

The first implementation for the set is a linked list. To make the linked list implementation the functions from the linked list files were used to make several of the new functions for the set. This was done by calling on the list function and pointing from the set to the list. In the set.h file the functions that should be created are defined. The main difference between the pre-code linked list and the set is that the set is sorted and has some more functions than the pre-code. For example, does the set have functions like union, intersection, and difference, that takes in two sets and gives one set back. The set implementation also has a copy function that makes it possible to copy a given set.

3.1.1 Union

To create the union function for two sets, a new empty set and two iterators are created, one for each original set. A while-loop running as long as there are elements left in set A is created. Within this while-loop, all of the elements in A are iterated over and added to the set if they do not already exist. This loop produces a set that contains all of the elements from set A. The same method is applied to set B, and the elements from B that do not already exist in the set from A are added. The iterators are then destroyed to free memory, and a set containing one of each element from A and B is returned.

3.1.2 Intersection

For the intersection of two sets, a new empty set and two iterators are created, one for each original set. Then a while-loop is created. The while-loop runs as long as set A has elements left, and for each element in A, it checks to see if set B has the same element. If A and B both have the element, it will be added to the new set that contains the elements that A and B both have. The iterators are then destroyed to free memory, and the intersection of A and B is returned.

3.1.3 Difference

The function difference returns a set containing the elements found in A but not in B. A new set and two iterators are created as a result of this. Another while-loop is created, and within this loop, the program checks to see if set A contains an element that set B does not. If so, the element is added to the new set. The iterators are then destroyed, and the result is a function that returns the difference of two sets.

3.1.4 Copy

To duplicate a given set, an iterator and a new set are created to store all of the elements. A while-loop is used because the program does not know the length of the set. The loop continues indefinitely as long as there are elements left in the given set. The element is added for each element that does not already exist in the new set. The result is a duplicate of the original set.

3.2 A set implemented as an array

For the second implementation, the plan was to make an array. Due to the time that was spent on the first implementation and the spam filter, there was simply not enough time to complete this. If there had been time left the way the array would have been implemented is quite like the solution from the first non-mandatory assignment. The biggest difference would be the newly added functions for the sets, which are union, intersection, difference, and copy. For the array, another sorting algorithm, Selection Sort, would have been used. This is because this would give an even bigger difference for the runtime.

3.3 Spamfilter

To begin, all of the files in the various directories are found and the names of these files are returned as a list of strings using find files. Then, for each directory, an iterator is created. In the assignment, a formula for identifying spam emails is provided:

$$(M \cap ((S1 \cap S2 \cap \ldots \cap Sn) - (N1 \cup N2 \cup \ldots \cup Nm)) \neq \emptyset).$$

First, the program must locate the intersection of all of the spam emails provided. A while-loop is created to allow this to be done with multiple sets. Tokenize is used to generate a set of unique words found in the given file, which the program then uses to find the intersection. Outside the loop, a set A that contains the first tokenized set is created. Another set, B, is created inside the while-loop. This set contains the following set from the specified directory. After locating both sets, the intersection function is applied and set equal to set A. When the while-loop is finished the result will be a set containing only words found in all the emails.

The same principle is applied when attempting to find the union of all non-spam emails. The only difference between this loop and the previous one is that the set union function is used instead of the intersection function, and the files tokenized are non-spam emails. As a result, a set containing all of the words found in the non-spam emails is created.

The difference between spam and non-spam emails gives a set that only contains the spam words. To find this the set difference function is used.

Another while-loop is created to run through all the emails in the directory to determine which are spam and which are not. Tokenize is used to create a set for each email, and an iterator is used to loop through the emails. Then, an if statement is used to determine whether the email contains spam words. This is accomplished by determining whether the set size is greater than zero. If the set size is 0, the email is spam-free; otherwise, it contains spam.

4 Evaluation and discussion

Assert and numbers were used to test the first implementation. The result of the numbers was equal to the given expected numbers, and when running assert, no errors appeared. This implies that the implementation of the set is correct.

To test the spamfilter ./spamfilter data/spam data/nonspam data/mail was used. The expected result of the spam filter was specified in the pre-code. The spam filter for this code returns the same results as expected, but the emails are not returned in the correct order. The mails are returned in the following order: mail 2, mail 3, mail 1, mail 4, and mail 5. This could be due to a small error in the iterator or a pointer miscalculation.

Because the second implementation is not completed due to time constraints, the expected complexity is used to compare them. First, the two sorting algorithms will be compared. Merge Sort is the first sorting algorithm, and it is used in the linked list implementation. Selection Sort is the second sorting algorithm, and it is the sorting algorithm that was intended to be used in the array implementation. Figure 6 shows a graph for how long run time different sorting algorisms have when the length of the list increases.

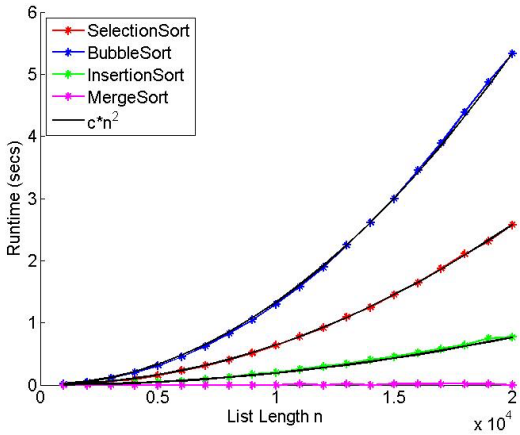

Figure 7, graph for different sorting algorithms

When there are few elements, the Merge Sort (pink) and the Selection Sort (red) are very similar, as shown in Figure 7. When several elements are added to the list, the runtime of the Selection Sort increases significantly more than the runtime for Merge Sort, which more or less remains the same. This allows us to conclude that Merge Sort is faster than Selection Sort when the list contains several elements.

5 Conclusion

The first assignment in INF-1101 is to create two sets of ATDs that support various functions. When the set is implemented, it will be used in a spam filter to determine whether an email contains spam. This is accomplished by creating a set that contains the difference between the intersection of words found in all spam emails and the union of words found in all non-spam emails. The given set is used to determine whether the given email contains any spam words and, if so, to notify the user that the email contains spam.

To complete the task, the set should have had two distinct implementations. Unfortunately, there was only time for one implementation, so the expected results for the implementations were used to compare the complexity. Following the completion of the code and the necessary testing, the result is a spam filter that uses sets to determine whether or not a message contains spam.

References

[1] codepath, “Big O Complexity Analysis,” [Online]. Available: https://guides.codepath.com/compsci/Big-O-Complexity-Analysis. [Accessed February 2022].

[2] Wikipedia, "Union (set theory)," [Online]. Available: https://en.wikipedia.org/wiki/Union_(set_theory). [Accessed February 2022].

[3] Wikipedia, "Intersection (set theory)," [Online]. Available: https://en.wikipedia.org/wiki/Intersection_(set_theory). [Accessed February 2022].

[4] Wikipedia, “Difference set,” [Online]. Available: https://en.wikipedia.org/wiki/Difference_set. [Accessed February 2022].

[5] GeeksforGeeks, “Doubly Linked List | Set 1 (Introduction and Insertion),” 28 January 2022. [Online]. Available: https://www.geeksforgeeks.org/doubly-linked-list/.

[6] Bigocheatsheet, “Know Thy Complexities!,” [Online]. Available: https://www.bigocheatsheet.com/. [Accessed February 2022].

[7] OpenGenus, “Time Complexity Analysis of Array,” [Online]. Available: https://iq.opengenus.org/time-complexity-of-array/. [Accessed February 2022].

[8] Geeksforgeeks, “Merge Sort,” 10 January 2022. [Online]. Available: https://www.geeksforgeeks.org/merge-sort/. [Accessed February 2022].