Program Design and Performance

5

The objective of this chapter is to discuss performance as another program design criteria.

5.1 Preconditions

The following should be true prior to starting this chapter.

- You understand three program design criteria: separation of concerns, design for reuse, and design only what is needed. You've evaluated program code using these criteria.
- If you are learning about structured design, then you know that a hierarchy chart may be used to model the structure of your program design.
- If you are learning about object-oriented design, then you know a class diagram may be used to model the structure of your program design.
- You know that a Nassi-Shneiderman diagram and a statechart are models used to illustrate the behavior of your program design.

5.2 Concepts and Context

The performance of software should consider runtime and memory usage.

D. P. Voorhees, *Guide to Efficient Software Design*, Texts in Computer Science,
https://doi.org/10.1007/978-3-030-28501-2_5

5.2.1 Performance: Time

All of the ABA Chaps. 3 and 4 program solutions use a sequence data structure to store the address book data. The Python and C++ solutions use a list data structure while Java uses an ArrayList data structure. When the code needs to determine whether a contact name is already in the data structure, a sequential search is performed of the data structure. When the number of entries in the address book is large (i.e., think hundreds of thousands and larger), the linear time (e.g., $O(n)$) behavior of a sequential search could have an observable affect on the responsiveness of the ABA.

It may be impractical to think of an address book as containing hundreds of thousands, millions, or even billions of contacts. It may also be unreasonable to assume that a search taking a few seconds means that the application is no longer useful to its users; having one poor performing software application on a computing device may be acceptable to a user. But if software developers ignore performance considerations, then users would end up using many poor performing applications. Given the desire to have computing platforms capable of operating many software applications simultaneously, having many poorly performing applications would compound the responsiveness issues for a user. Even with the increase in hardware computing power, the *time complexity* (i.e., time performance) of software applications is still something that software programmers and designers should pay attention to.

When looking to correct poor performing software, there are a handful of choices a software designer may opt to implement:

- Change the algorithm to one that has better performance.
- Change the data structure to one that has better performance.
- Implement multithreading and move the poor performing code to a separate thread.
- Target a computing device that has significantly more processing capabilities.

We will focus on the first two items in the above list.

5.2.1.1 Algorithm Performance

What if we change the sequential search to a binary search? A binary search exhibits $O(\log_2 n)$ time, which is significantly faster than $O(n)$ for large n. For example, for n=10 a linear search would take 10 time units while a binary search would take about 4.3 time units. However, when n=1000 a binary search would only take about 11 time units! A potential issue with doing a binary search is that the values being searched must be in sort order. To answer the question about switching to a binary search, we need to look at each programming language.

The Python list data structure supports indexing (using the [ ] operator) and has a sort method that sorts the list contents in place (i.e., it does not use any extra memory to perform the sort). Python also has a built-in function, named sorted, that will sort the contents into a new list object. The indexing operator exhibits constant time $O(1)$ performance while both the sort method and the sorted() function have $O(n \log n)$ time performance [1].

With Python lists, we would first do a binary search to see if the contact name is already in the ABA. This would have $O(\log_2 n)$ time performance. When the contact name is not in the ABA, we would add the new contact name to the end of the list data structure (constant time $O(1)$ performance) and then sort the list (with $O(n \log n)$ performance) to ensure the next binary search works correctly. Thus, switching from a sequential search to a binary search in the Python solution takes us from $O(n)$ to a worse time performance of at least $O(n \log n)$. (When n=1000, $n \log n$ is 3000.)

The Java ArrayList data structure supports indexing (using the get method) and the Java Collections class has a static sort method that will sort an ArrayList. The ArrayList get method exhibits constant time performance while the Collections sort method has $O(n \log n)$ time performance [2]. Thus, switching from a sequential search to a binary search in the Java solution also takes us from $O(n)$ to $O(n \log n)$ performance.

The C++ list data structure is a doubly linked list data structure that does not support indexing. Adding an item to the middle of the list exhibits linear performance while adding an item to the front or back of the list exhibits constant time performance. With no support for indexing, sorting a C++ list data structure is an expensive operation that would result in overall time performance that is worse than linear time.

In the case of these three programming languages and the data structures being used, the faster binary search algorithm would not improve the overall time performance of the ABA.

5.2.1.2 Data Structure Performance

What if we change the type of data structure being used to store the contact names? Would this improve the performance? To answer the question about switching the type of data structure, we need to look at each programming language.

In the Python solution, we can switch from a list data structure to a dictionary data structure. Since a Python dictionary is implemented as a hash table, the hash function used to locate the key value (e.g., the contact name) within the dictionary represents the major impact on time performance. When there are no collisions generated by the hash function, the time performance for storing and retrieving a contact name would be $O(1)$ constant time. However, when the hash function generates many collisions for the different contact names then the time performance would get closer to linear time [1]. Thus, the worst-case performance when using a dictionary matches the best-case performance when using a list.

In the Java solution, we can switch from an ArrayList to a TreeSet. A Java TreeSet implements a red-black tree that guarantees $O(\log_2 n)$ performance when storing and retrieving an item from the data structure [2]. Note that Java also has a HashSet data structure that would exhibit similar time performance as a Python dictionary.

In the C++ solution, we can switch from a list to a set. A C++ set is typically implemented using a red-black tree that guarantees $O(\log_2 n)$ performance when storing and retrieving an item from the data structure [3].

In the case of these three programming languages, other data structures exist that would allow the ABA to exhibit time performance that is better than the linear time associated with the Chaps. 3 (OOP) or 4 (SP) solutions.

5.2.2 Performance: Memory

Another way to evaluate the performance of a program design is to determine its memory usage. Two different solutions to the same set of requirements may use dramatically different amounts of memory. Memory usage can even vary for the same program design implemented in different programming languages.

For example, a Python dictionary is implemented internally as a hash table [4]. A hash table allocates memory for each table entry based on the initial size of the hash table. In Python, the initial size of a hash table is 8 and the collision resolution algorithm uses quadratic open addressing (to avoid looking at contiguous hash table entries for an empty slot). When the hash table is two-thirds full, Python will either quadruple the hash table size (when the hash table has fewer than 50K slots) or double the size (when the hash table has over 50K slots). A Java TreeSet is implemented as a binary search tree (BST). A BST allocates memory for each node in the tree where each node must know if it has child nodes (to allow the BST to be searched quickly).

Both a Python dictionary and a Java TreeSet improve search time, as noted above, but how do these two data structures affect memory usage? Since a hash table has a fixed size, a hash table could have the capacity to store thousands or millions of entries but perhaps there are times when a large hash table is sparsely populated. In this case, one could argue that the hash table is wasting a significant amount of memory. In contrast, a Java TreeSet only has nodes for values that are currently stored in the BST. Thus, a TreeSet makes more efficient use of memory when compared to a sparsely populated hash table.

Parameter passing is another type of memory usage that should be evaluated. This type of memory usage affects the runtime stack, which has a limit to its size. If you've ever written a recursive function that calls itself infinitely many times, you've caused a stack overflow exception that indicates that the size limit to the runtime stack has been exceeded.

In Python and Java, any object (e.g., a Python dictionary or a Java TreeSet) passed as a parameter to a function/method is passed as an object reference. An object reference is a small data value (perhaps 4 bytes in size) that refers to where the object can be found in memory. For example, if we had written the Python displayBook method/function as *def displayBook(book):*, then each call to displayBook would have resulted in a pointer (memory address) value being stored on the runtime stack. This is much more efficient than having to copy the entire data structure onto the runtime stack as part of each method call. Similarly, the Java displayBook method could be written as *public void displayBook(TreeSet book)*. Each call to displayBook would (again) result in a pointer (memory address) value being stored on the runtime stack.

In C++, the default passing mechanism used is *pass by value*. While a C++ set is an object, the default passing mechanism means that a copy of the set will be placed on the runtime stack. This copy is what is used by the method being called. With pass by value, the method does not have access to the original set object. This default behavior can potentially use lots of memory when the data structure contains lots of data. More about this will be discussed in Chap. 7 on structured programming.

5.3 Post-conditions

The following should have been learned after completing this chapter.

- Evaluating the performance of a program design should consider both time and memory usage. More specifically:
  - Adjusting a program design due to poor time performance should look at both the algorithms and data structures being used.
  - Adjusting a program design due to poor memory usage should consider how a programming language does parameter passing and how data structures are being used to store data.

5.4 Next Chapter?

If you are interested in seeing how performance may be applied to a small object-oriented programming solution, continue with Chap. 6. This chapter will continue the Address Book Application case study using Python and Java.

If you are interested in seeing how performance may be applied to a small structured programming solution, continue with Chap. 7. This chapter will continue the Address Book Application case study using Python and C++.

Exercises

Discussion Questions

1. Can you think of a situation where it would be okay to ignore poor time performance in your solution?

2. Can you think of a situation where it would be okay to ignore poor memory performance in your solution?
3. With the fairly rapid increase in processor speed and memory sizes, even in very small computing devices, why is performance still something that should be considered when developing a solution?
4. Cloud computing is a widely used service. Examples of cloud services include email, personal financing, social media, and e-commerce. Since many cloud computing services use large server farms to provide the service, would performance of a cloud-based software app be something that should still be considered? Why?

References

1. Python Software Foundation: TimeComplexity. Python Wiki https://wiki.python.org/moin/TimeComplexity. Accessed 3 Jun 2014
2. Oracle.com: Java Platform Standard Edition 7 API. https://docs.oracle.com/javase/7/docs/api/. Accessed 2 Jun 2014
3. Nyhoff L (2005) ADTs, data structures, and problem solving with C++. Prentice Hall, Upper Saddle River
4. Python.org (2017) Dictionary object implementation using a hash table. http://svn.python.org/projects/python/trunk/Objects/dictobject.c. Accessed 12 Jun 2017