**LINKED DATA STRUCTURES \***

Chekanin, Vladislav A; Chekanin, Alexander V.

INTRODUCTION

Arrays and Lists are one of the many ways data can be stored, accessed, and manipulated using a programming interface on the memory of your computer. They are quite similar with one another, such that in essence, they are storing memory addresses for the data they hold. The array is the fundamental sequentially-allocated data structure. It is fixed-sized, can be dynamically reallocated, and is efficiently indexed. Each item in an array is able to be referenced, by its index, in constant time. The list is the fundamental linked data structure. Therefore, a list is stored as distinct blocks of memory bound together by pointers (Jerome, 2017).

Linked data structures use a combination of very effective data structures and is used in many algorithms. The implementation of a linked data structure includes an organize set of elements having links to each other. One of the greatest advantages of a linked data structure is that its elements can be located on different memory allocations contrary to a linear array. Since stack, queue and array are linear data structures, we shall establish how these data structures can be implemented, and that includes the operations as well (Chekanin, Chekanin 2017). For this journal review, the data structure to be discussed is about the Stack Data Structure.

STATEMENT OF THE PROBLEM

In general, the process of indexing an array takes more time to execute than does the process of accessing the contents of a pointer. In fact, this is one of the main reasons why pointers are used to access the elements of an array (Kochan 2014). The main issue to be discussed is the optimization of a program, one aspect is through the use of arrays in accessing data or its elements, by looping over every element, in another perspective, the speed of insertion and deletion of elements are also concerned. For small-scale arrays, this sort of method will not affect the overall time consumed, but when this array becomes the size of a thousand or more, this will be a bottleneck for the processing time.

LINK

https://search.proquest.com/pqrl/docview/2190324803/fulltextPDF/4ABB9BBA9C4E472BPQ/1?accountid=34542  
August 31, 2019  
GMT+8 - 8:00 AM

CONCEPTS AND THEORIES

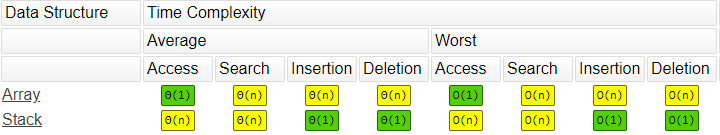
Data in a program, whether stored in an array or a list, is stored in addressable block of memory so that it can be accessed later. Items in either arrays or lists may be referred to as nodes, items, elements, records, or entities. We’ve been using the term “node” to describe the items linked together in lists. In the case of arrays, we’ll use the term “item”, which is conventional. Keep in mind, however, that all of these terms can be used interchangeably (Jerome, 2017).

A list is an ordered collection of elements that controls its elements by the position at which a new element is inserted to or deleted from. The process of inserting is fairly simple, where a new element, inserted at an index, will take that location, if there exist an element at that index, that element and the proceeding elements are moved by an index. In that case, the list will have to do extra effort, in which once in a while, the insertion operation will take time. This concept is the same with the deletion of elements in a list, where each of the elements of the list are adjusted correspondingly (Manifesto 2015).

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top (Nitesh, Rahul 2014). A stack is an example of a linked data structure where the elements of the stack have a memory address to other stack elements, usually having the terms, next and/or prev, meaning the next item and the previous item respectively. To implement a stack, at least two pointers must be available, one for the bottom of the stack, and one for the top of the stack.

SOLUTION

Stack, as it is implemented as a linked data structure, effectively reduces the time constraints of insertion and deletion of elements. Time complexity must also be taken into consideration, through which, we can consider the data structure to be used for a certain program, with optimization in mind.

Green - > Excellent Yellow -> Good

We can see that for the Stack, there are two green areas and two yellow areas (average and worst case). As for the Array, there are three yellow areas. The main advantage of the stack data structure is in the insertion and deletion process.

Stack, its Applications and Implementation.

* Simplest example for a stack is to reverse a word.
* The undo mechanism in text editors, and many other applications.
* Backtracking, a process when you need to access the most recent element in a series of elements.

(Nitesh, Rahul 2014).

CONCLUSION

Although it is true that we can use any data structure we desire to use, as a good developer, for applications or scripts, optimization must always be taken into consideration as well. Data structures such as arrays, list, arraylist, linkedlist, stack, queue, tree, and of others have their own advantages and disadvantages. It is our duty as developers to use what is appropriate for the application in development. Yes, a list can do everything a stack can do, but if we know to what extent an application is used for, then maybe we can adjust what data structure to be used.

In conclusion, knowing the operations of the data structures helps us realize more of the underlying principles of programming and memory allocation. Through this knowledge, in doing low-level or high-level programming, can help us determine what should and shouldn’t be done in practice. We must keep in mind, even if it works, does not necessarily mean that it is a good practice, this sort of practice is coined with the term spaghetti code, where the mindset is simply, ‘if it works, then it works’. In knowing the time complexity of the operations in the different data structures, we should then be able to weigh the factors in decision making regarding to what data structure to be implemented.

REFERENCES

Adam Drozdek. Data Structures and Algorithms in C++, Third Edition. Thomson Course Technology, 2005. ISBN 0 - 534-49182-0. Chapter 4: Stacks and Queues, pp. 137–169.

Chekanin, V. A., & Chekanin, A. V. (2017). LINKED DATA STRUCTURES \*. *International Journal of Computer Research, 24*(2), 237-267. Retrieved August 31, 2019, from https://search.proquest.com/docview/2190324803?accountid=34542

Chekanin, V. A., Chekanin, A. V. (2016). Основы алгоритмизации и программирования на языках C и C++; ФГБОУ ВО МГТУ “СТАНКИН”: Москва, 2016, pp. 265-278. [Basics of algorithmization and programming on languages C and C++; MSTU “STANKIN”: Moscow, 2016, pp. 265-278].

Donald Knuth. The Art of Computer Programming, Volume 1: Fundamental Algorithms, Third Edition. Addison- Wesley, 1997. ISBN 0-201-89683-4. Section 2.2.1: Stacks, Queues, and Deques, pp. 238–243.

Jerome, D. L. (June 30, 2017). Lists Part III: Lists vs Arrays. Retrieved September 1, 2019 from https://medium.com/@dljerome/lists-part-iii-lists-vs-arrays-60bdced64eed.

Kochan, S. (August 18, 2014). Programming in C (4th Edition). Developer’s Library.

Manifesto, Mario. (January 16, 2015). Java lists: Arraylist vs LinkedList. Retrieved September 4, 2019 from https://manifesto.co.uk/java-lists-arraylist-vs-linkedlist/

Nitesh, Manbir S., Rahul Y. (2014). RESEARCH PAPER ON STACK AND QUEUE. Volume 1 Issue 7. JIRT.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 10.1: Stacks and queues, pp. 200–204.

Weiss, M. A. Data Structures and Algorithm Analysis in C++; Pearson Education: Boston, MA, 2014, pp. 77-116.

N.A. Know Thy Complexities. Retrieved September 2, 2019, from https://www.bigocheatsheet.com/

William Ford, William Topp. Data Structures with C++ and STL, Second Edition. Prentice Hall, 2002. ISBN 0-13- 085850-1. Chapter 8: Queues and Priority Queues, pp. 386–390.