Computer Science - Data Structures

Topic: Abstract Data Structures

Approach: Computer Science

Miguel Angel Avila Torres - © 2020 All Rights Reserved

Context

This project will try to verify if the student has develop an understanding about the most common abstract[1] data structures.

Motivation

As we know, general data structures (not just those lists, trees, maps, etc.) are itself abstract containers in which we represent certain characteristics, properties and actions/operations that real objects have.

Data structures improves information handling by means using schemes from which is possible to abstract, separate and manipulate some desired characteristics present in the system object of design.

Well, to have in mind, the correct organization of the information, is a crucial factor that effects the response time of any system.

Computer scientists have build very useful/powerful data structures which provides efficient management over any kind of data, these are called "abstract data structures" because different instances of those can operate with different parameterized types even with other data structures, despite those being concrete or abstract (generic or not).

Abstract Data Structures

Stack

- Generalized operations:
  1. Clear
  2. Is empty
  3. Contains
  4. Put at top
  5. Remove (returns the element)
  6. Delete (does not return anything)
  7. Iterator
  8. Size
  9. Resize (if allowed)
- Implementations:
  - Array-Based (not resizeable)
  - Node-Based

[1]Notice that in some languages those structures are not purely abstract, but still are useful. e.g. GO.

Queue

- Generalized operations:
  1. Clear
  2. Is empty
  3. Contains
  4. Put
  5. Remove (returns the element)
  6. Delete (does not return anything)
  7. Iterator
  8. Size
- Implementations:
  - Array-Based implementation
  - Singly linked node based
  - Doubly linked node based
  - Circular doubly linked node based
  - DQ Array Based

Priority Queue

- Generalized operations:
  1. Put
  2. Has pair
  3. Has key
  4. Has value
  5. Is empty
  6. Find min key
  7. Find min value
  8. Find min pair
  9. Find max key
  10. Find max value
  11. Find max pair
  12. Remove min
  13. Remove max
  14. Clear
  15. Key iter
  16. Value iter
  17. Iterator
  18. Size
- Implementations:
  - Unsorted Priority Queue
  - Sorted Priority Queue

List

- Generalized operations:
  1. Add as last
  2. Add as first
  3. Add at index
  4. Add all an iterator at bottom
  5. Add all an iterator at an index
  6. Add an array at the bottom
  7. Add an array at an index
  8. Delete last
  9. Delete first
  10. Delete all in an iterator on first occurrence
  11. Delete all in an iterator on last occurrence
  12. Delete all in an iterator with all occurrences
  13. Delete array content on first occurrence
  14. Delete array content on last occurrences
  15. Delete array content with all occurrences
  16. Replace element at
  17. Remove at index
  18. Contains
  19. Index of
  20. Get at index
  21. Get first
  22. Get last
  23. Set element
  24. Iterator
  25. Size
- Implementations:
  - Doubly Linked List (DLList)
  - Circular Doubly Linked List (CDLList)

Tree

- Generalized operations:
  1. Add
  2. Delete element
  3. Delete all in an iterator
  4. Delete array content
  5. Clear
  6. Is Empty
  7. Inorder
  8. Preorder
  9. Postorder
  10. As list
  11. Iterator
  12. Size
- Implementations:
  - Binary Search Tree
  - Key-Value Binary Search Tree
  - AVL-Tree
  - Key-Value AVL-Tree
  - Red-Black Tree
  - Key-Value Red-Black Tree

Map

- Properties:
  - Entry set
- Generalized operations:
  1. Put
  2. Put all from entry iterator
  3. Put all from other map
  4. Delete all in a key iterator
  5. Delete all in a value iterator
  6. Delete first occurrence from a value iterator
  7. Delete last occurrence from a value iterator
  8. Delete array content
  9. Delete first occurrence from array content
  10. Delete last occurrence from array content
  11. Get value
  12. Get keys
  13. Value iterator
  14. Key iterator
  15. Entries iterator
  16. Size
- Implementations:
  - Unsorted Array Map
  - Sorted Array Map
  - Unsorted Linked Map
  - Sorted Linked Map
  - Key-Value AVL-Tree based
  - Key-Value Red-Black Tree based

Hash Table

- Generalized operations:
  1. Put
  2. Put all from entry iterator
  3. Put all from other map
  4. Delete all in a key iterator
  5. Delete all in a value iterator
  6. Delete first occurrence from a value iterator
  7. Delete last occurrence from a value iterator
  8. Delete array content
  9. Delete first occurrence from array content
  10. Delete last occurrence from array content
  11. Get value
  12. Get keys
  13. Value iterator
  14. Key iterator
  15. Entries iterator
  16. Size
- Implementations:
  - Array-LinkedNode Based Hash-Table
  - HashMap-Array based
  - AVLTree-Array based

Graphs

- Generalized operations:
  1. Vertices number
  2. Edges number
  3. Get edge
  4. Vertices of

5. Vertex linked to
6. In degrees number of
7. Out degrees number of
8. In degrees iterator of
9. Out degrees iterator of
10. Insert vertex
11. Insert edge
12. Remove vertex
13. Remove edge
14. Vertices iterator
15. Edges iterator

- Implementations[2]:
  - Edge List
  - Adjacency List
  - Adjacency Matrix

Clarifications

In some cases the Generalized public operations between some data structures are in fact the same, but it is only a perspective question, a partial visualization about the real information management that our data structures have.

The number of total operations needed to fulfill the requirements of this project are:

- 9×2 :: Stacks
- 8×5 :: Queues
- 18×2 :: Priority Queues
- 25×2 :: Lists
- 12×6 :: Trees
- 16×6 :: Maps
- 16×3 :: Hash Tables
- 15×3 :: Graphs
- $\sum = 405$ Operations

Adding some other operations and declarations according to the language used and other factors, probably there are approximately 410 to 420 operations, including but not limiting private operations.

It would be better to develop an UML class diagram to provide certain coherence and speeding up the development of this project, it also would be recommendable to create an inheritance model to reduce similar code.

Notes

- You are not allowed to use any native class from any std library present in the program language that you are going to use. Given the case, your real qualification will be recalculated deducting 20% from the original one.

- All the mentioned implementations must be done and tested, the complexity analysis of each mentioned generalized operation must be written on a pdf file, not just the final result, the complete analysis per algorithm[3], the structure will be:
  - Project title
  - Student name
  - Student code
  - Sections, which are the worked data structures
  - Subsections, that adds all implementation (mentioned, not code) related to that data structure
  - Subsection items, containing an operation for which will be written its complexity analysis

- You must use a writing format for the analysis document as APA, Michigan, IEEE, etc., also you may use any online template (check the license type).

- This project has 3-4 weeks to be developed. Ideally coding 15 algorithms per day to finish the project in exactly 4 weeks so that we finish the syllabus in that time and spend the semester learning about further topics.

- No fancy graphics are needed, nor a web application. Just use raw console prints from the view. This is data structures no design class.

  Well clearly I appreciate a great work done with graphics and all, giving some extra points or exonerating from some other work, being more flexible on qualifications etc. but if the data structures does not work then that and nothing are equal.

  You better concentrate in completing the logic core and then (if you want) designing an eye-catching view.

- Questions with the instructor (me). **Do not be shy**, if you do not know how to do something then ask. But first try to make a brief supposition about how to solve the problem.

- Finally, if you use C++ for the development of this project I will be more gentle in your qualification.

Support Material

[1] M. A. Weiss, “DATA STRUCTURES AND ALGORITHM ANALYSIS IN C++” Fourth Edition. Florida International University, Person, 2013.
[2] M. T. Goodrich, R. Tamassia, M. H. Goldwasser “Data Structures & Algorithm Analysis in JAVA™” Fourth Edition. Florida International University, Person, 2013.

[2]For both directed and undirected graphs

[3]If you choose LaTeX it gives you 20% more to your current grade over the maximum qualification (cumulative in further assignments).