Data Structures

Basil Udoudoh

MEETING OBJECTIVES

01

MERCURY

Mercury is the closest planet to the Sun

02

VENUS

Venus is the second planet from the Sun

AGENDA

01

Introduction

Give a little to get a little (data that is)

02

Course Rules

The “Structure” of our course

03

Getting Started with Repl.IT

I promise it will work this time, mostly……

AGENDA

Linear Data Structures

This Data is pretty “straightforward”

04

05

Hierarchical Data Structures

Bigger data, higher learning

Closing Time

No puns here, just time to leave

03

Introduction

Hey everyone, I’m Basil.

I’m a Software Architect/Lead from Slalom. I’ve been building software on multiple platforms (web, mobile, desktop, cloud) for 15 years

When I’m not working, I love playing sports and video games

I’m a big fan of pecan pie

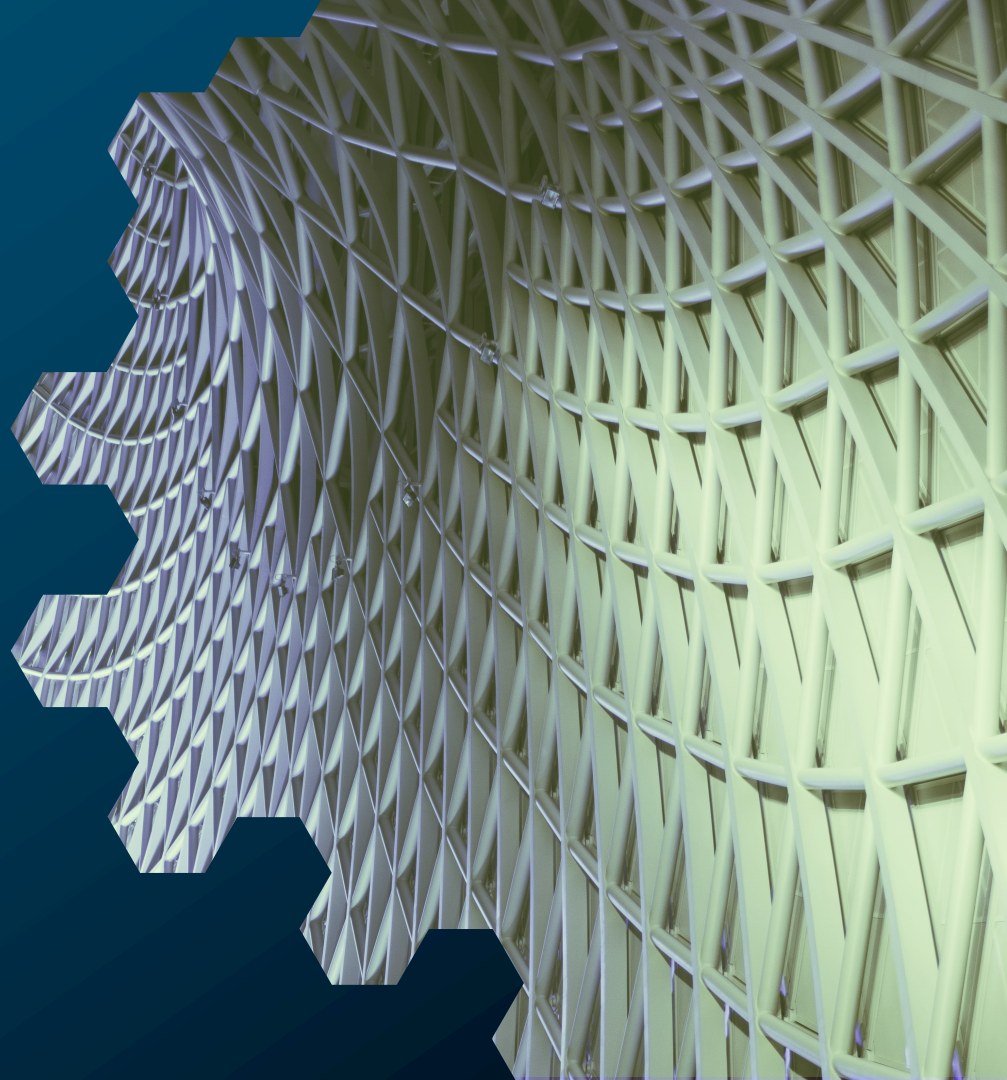
Tell me about you

Hey everyone, who are you?

Professional Background

What do you do for fun besides coding?

What is your favorite desert?



COURSE RULES

PARTICIPATE!

You will all get so much more from this experience if you participate. Try to contribute to all of the discussions.

ASK QUESTIONS

Though it might sound like it when I talk, I don't just want to listen to my voice all day. Ask questions and share experiences whenever you want.

NO CODING NEEDED

There are coding problems in this course, but you don't need to have coding experience to participate. You can follow along with me.

EARLY MORNING JAVA

All Coding Examples are in Java, though the principles apply to all programming languages

GETTING STARTED WITH REPL.IT

https://repl.it/@BasilUdoudoh/data-structures-java

All of the coding problems in this course course can be found here. Each person will have their own space that they can work on the coding examples in.

If you don't already have an account, go ahead and sign up. It's free and a great tool to use to work on code.

If you would rather follow along and not use repl.it, that works too!


LET'S GET STARTED

WHAT IS A DATA STRUCTURE

- Software is all about data!
- Data Structures are memory/program constructs that allow us to organize data so that it can be used effectively
- Modern software processes massive amounts of data, so it's important that we choose carefully when we're loading data into structures.

DATA STRUCTURE FUCTIONS

LOAD

Data has to be loaded into the data structure to be used. Individual elements can be loaded ad-hoc as well.

ACCESS

Once the data is present, the application will need to periodically access individual and multiple data elements

SEARCH

With multi-dimensional data sets, applications will often have to search for data elements that have certain properties

DELETE

Once a data element is no longer valid or useful, I may need to be removed from the data structure without compromising the rest of the data

BIG-O: MORE COMPLEX THAN IT SOUNDS

- Data Structures are judged by how efficient they can perform the functions from the last slide in two dimensions: **Space and Time**
- **Big-O** notation is a measure of complexity (or performance) of a data structure or algorithm and how it performs in Space (memory usage) and Time (CPU usage)
- Big-O describes the average and worst-case performance of a data structure based on how many data elements are held in that structure (usually described as “N” number of elements)
- **Time** is the most important complexity measure for data structures, as space complexity collapses down to the number of elements as the data structure and its number of elements get larger.
- Big-O measures are important to remember for interviews!

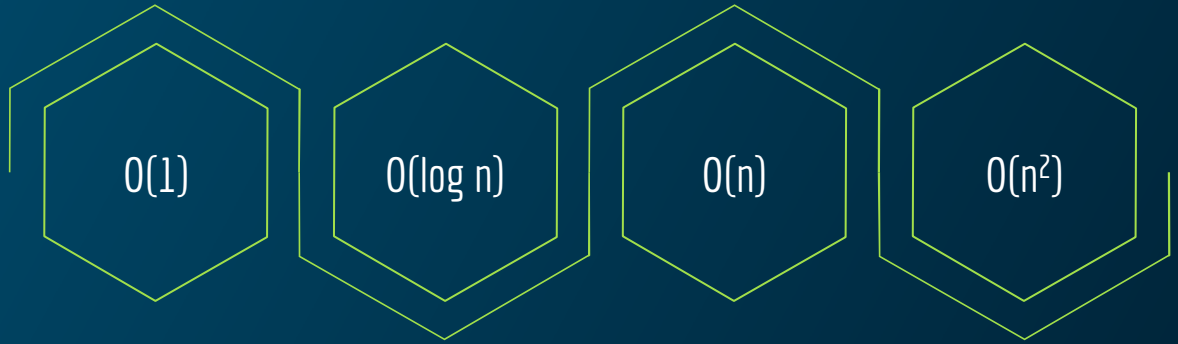
IMPORTANT BIG-O MEASURES

CONSTANT TIME

The best performance possible for a data structure

LINEAR TIME

Not great performance, but sometimes it's the best you can do

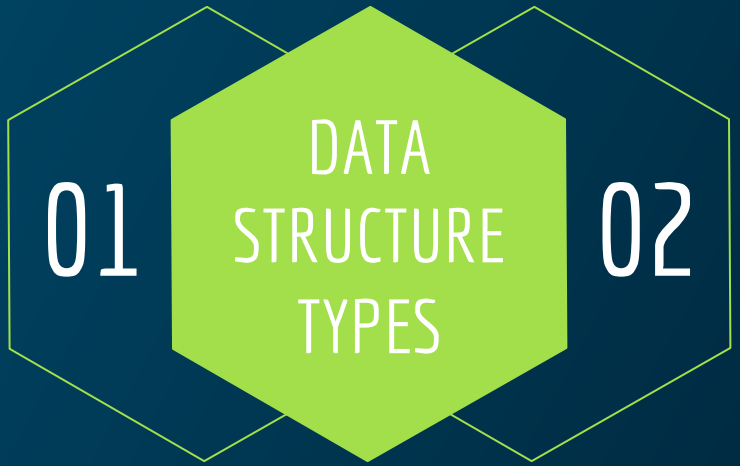


LOGARITHMIC TIME

Very efficient performance, usually based on sorted input

QUADRATIC TIME

Horrible performance for a data structure, usually only found in sorting algorithms, which is the next class



LINEAR DATA STRUCTURES

Data is collected in parsed in a linear fashion

NON-LINEAR/HIERARCHICAL DATA STRUCTURES

Data is collected and parsed in a hierarchical or non-linear fashion

LINEAR DATA STRUCTURES

LINEAR DATA STRUCTURES

ARRAYS

An oldie but goodie

LINKED LISTS

Sometimes all you need is to know where you are and what's next

STACKS

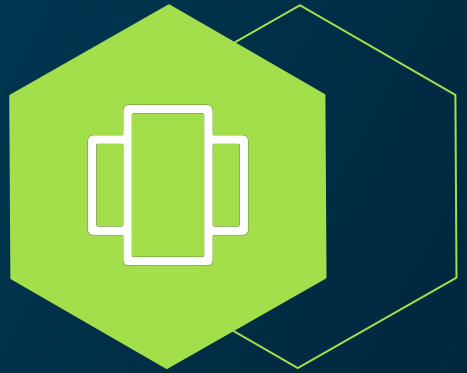
Works for more than just large sums of money

QUEUES

Just like at the DMV, except quicker. Much quicker……

HASH MAPS

This map won't take you forever to find your destination



ARRAYS

An oldie but goodie

ARRAYS - FACT SHEET

An array is a collection of data elements, each of which identified and retrieved by an array index.

LOAD

- Command – Create new then copy
- Complexity – O(n)

SEARCH

- Command – foreach() then compare
- Complexity – O(n)

BEST USE CASES

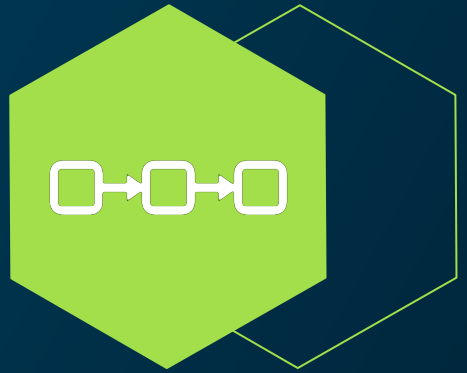
- Static dataset
- Pre-counted dataset

ACCESS

- Command – Array[index]
- Complexity – O(1)

DELETE

- Command – Create then copy
- Complexity – O(n)



LINKED LIST

Sometimes all you need is
to know where you are
and what's next

LINKED LIST - FACT SHEET

A linked list is a collection of nodes, where each node is made up of a data elements and a pointer to the next node in the list.

LOAD

- Command – Get the tail, change the pointer
- Complexity – O(1)

SEARCH

- Command – Get the head, iterate
- Complexity – O(n)

BEST USE CASES

- Dynamic dataset
- Iterative use cases

ACCESS

- Command – Get the head, iterate
- Complexity – O(n)

DELETE

- Command – Remove the pointer
- Complexity – O(1)

STACKS

Works for more than just large sums of money

STACKS - FACT SHEET

A stack is a collection of data elements where each element that is added as well as removed to/from the collection is added to the front of the collection (LIFO).

LOAD

- Command – push to the top of the stack
- Complexity – O(1)

SEARCH

- Command – pop from stack, iterate
- Complexity – O(n)

BEST USE CASES

- Dynamic dataset
- Iterative use cases
- Single use data structures
- Recursion use cases
- Depth-First Seach

ACCESS

- Command – pop or peek off the top of stack
- Complexity – O(n)

DELETE

- Command – Pop from stack
- Complexity – O(1)

QUEUE

Just like at the DMV, except quicker. Much quicker……

QUEUE - FACT SHEET

A queue is a collection of data elements where each element that is added to the back of the collection and removed from the front of the collection (FIFO).

LOAD

- Command – enqueue in the back of the queue
- Complexity – O(1)

ACCESS

- Command – dequeue from the front of the queue
- Complexity – O(n)

SEARCH

- Command – dequeue from the front of the queue, iterate
- Complexity – O(n)

DELETE

- Command – Dequeue from stack
- Complexity – O(1)

BEST USE CASES

- Dynamic dataset
- Iterative use cases
- Single use data structures
- Breadth-First Search

HASHMAP

This map won’t take you forever to find your destination

HASHMAP - FACT SHEET

A hash map is a collection of data elements where an index for each element can be calculated from a unique key using a hashing function.

LOAD

- Command – insert into hash map
- Complexity – O(1)

SEARCH

- Command – Get from map based on key
- Complexity – O(1)

BEST USE CASES

- Dynamic dataset
- Single Lookup use cases

ACCESS

- Command – get from map based on key
- Complexity – O(1)

DELETE

- Command – Remove from
- Complexity – O(1)

LINEAR DATA STRUCTURES REVIEW

ARRAYS

- Static, fully counted data
- Great for storage and iteration

LINKED LISTS

- Dynamic, ad-hoc data
- Great for storage and iteration

STACKS

- Dynamic, ad-hoc data
- LIFO
- Great for recursion and DFS

QUEUES

- Dynamic, ad-hoc data
- FIFO
- Great for BFS

HASH MAPS

- Dynamic, ad-hoc data
- Great for efficient lookup tables

NON-LINEAR DATA STRUCTURES

NON-LINEAR DATA STRUCTURES

TREES

Think about the children

GRAPHS

Networking and relationships

TREES

Think about the children

TREE - FACT SHEET

A tree is a collection of data elements where data is structured as a hierarchy, with data elements serving as parent nodes and child nodes for each other. A Binary Search Tree is a special tree variant where each parent has at most two nodes, each node is themselves a Binary Search Tree, and the value of the left node is always less than the value of the right node.

LOAD

- Command – insert into tree
- Complexity – O(log n)

SEARCH

- Command – traverse tree
- Complexity – O(log n)

BEST USE CASES

- Dynamic dataset
- Single Lookup use cases
- Hierarchical data

ACCESS

- Command – traverse tree
- Complexity – O(log n)

DELETE

- Command – Remove from tree
- Complexity – O(log n)

GRAPHS

Networking and
relationships

GRAPH - FACT SHEET

A graph is a collection of data elements that are structured in relation to each other, with the data serving as vertices and the relationship serving as edges.

LOAD

- Command – insert node or vertex into graph
- Complexity – O(1)

SEARCH

- Command – query graph
- Complexity – O(n)

BEST USE CASES

- Dynamic dataset
- Relational Data

ACCESS

- Command – query graph
- Complexity – O(n)

DELETE

- Command – Remove from graph
- Complexity – O(1)

NON-LINEAR DATA STRUCTURES REVIEW

TREES

- Dynamic, hierarchical data
- Great for performing operations in logarithmic time

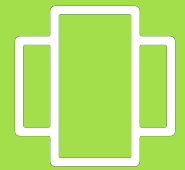
GRAPHS

- Dynamic, relational data
- Great for mapping relationships between data elements

TAKEAWAYS

O(n)

Remember the concept of Big-O and the different time complexities for each data structure type



Use the best data structure for the job you're doing. Arrays aren't always the answer

Graphs and trees are great for mapping relationships between elements

THANKS

All today's code and this presentation are available on github @ https://github.com/budoudoh/data-structures-java