

Data structures

Srikanth Kandula



April 14, 2020

Contents

[Data Structures 1](#_Toc39004118)

[Arrays 1](#_Toc39004119)

[Linked Lists 1](#_Toc39004120)

[Stacks 2](#_Toc39004121)

[Queues 2](#_Toc39004122)

[Associative Arrays 3](#_Toc39004123)

[Hash Functions 3](#_Toc39004124)

[Hash Table 4](#_Toc39004125)

[Sets 4](#_Toc39004126)

[Trees 5](#_Toc39004127)

[Binary Search Trees (BST) 5](#_Toc39004128)

[Heaps 6](#_Toc39004129)

[Big O Notation 6](#_Toc39004130)

# Data Structures

All the data is stored as a set of 1s and 0s. How does the computer identify different data types or data structures? Are there any specialized bits for each type to indicate them?

Primitive data types. Reference or derived data types.

Arrays

Single dimension array, multi-dimension array, jagged array (length of arrays inside this multi-dimensional array varies).

When you add an item to an array, it can be added to an existing array or a new array is created copying all the existing array contents and adding this new element to the end of it. This implementation depends on the language and has major performance repercussions.

Linear search – searching each item of the array one by one. The time to search is proportional to the size of the input array to search from. This linear search can be done by the programmer or it is done in the background by predefined methods like .indexOf().

## Linked Lists

Linked lists are a linear data structure consisting of different Nodes. Each Node is a combination of a data entry and a pointer which points to next node. The first node in the linked list is called Head. The pointer of the last node in the linked list points to null. Nodes are not stored in contiguous locations.

ArrayLists:

They are linked lists but are actually stored as arrays under the hood. They favor direct access of elements of the list.

The connotation of the term linked lists changes with language. Some infer doubly linked lists, some infer singly linked list, and some refer lists which are actually arrays under the hood.

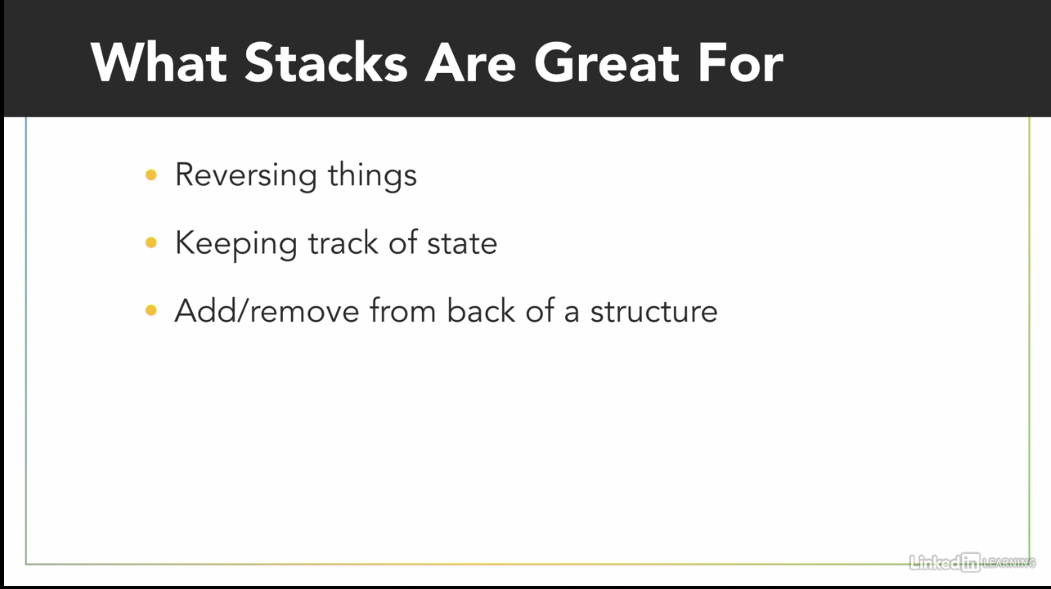
* Merge sort is generally preferred with linked lists.
* Linked lists have slowed random access performance. Because we start at the head and proceed step by step to go a node in the list unlike array where you can directly access by specifying its index.

## Stacks

Stack is a data structure where new elements are added/removed to/from the existing data. This follows LIFO policy.

There is **no indexing** in stacks.

**Runtime stack** maintains track of the state in time and lists the sequence of operations when some error occurs.



Stacks are useful for the operations that need LIFO operations like reversing a string, debug (i.e. keep track of state).

They are not good if items at the beginning or in the middle are to be accessed often.

## Queues

Queue is a data structure where new elements are added only to the end of existing data. Elements can be popped only from the front or beginning of this data structure. Adding is called **Queueing** and removing is called **Dequeuing**.

Queue is an ordered list (i.e. a collection of items where each item holds a relative position) with a specific way of adding and removing items.

Methods like .append() or push() add new element to the back or end of the queue.

There is **no indexing** with queues.

This follows FIFO policy.

**Priority queues:** Each item in the queue is given some priority. The item with highest priority is dequeued first. Items with same priority are dequeued in FIFO order.

**Dequek (**pronounced as deck**)** is a queue in which items can be removed or popped from either ends of the queue but not from anywhere else. It is more like a combination of a stack and a queue.

* Queues are best for operations that require FIFO policy. They are also ideal when items are to be added/removed from the ends (Dequek). Priority queue can be used if the items in the middle are to be accessed occasionally, but they soon turn in-appropriate if number of accesses from the middle increases.

## Associative Arrays

**Abstract data type** – It is a data type whose behavior is specified by its definition but not how it is implemented. For example, int, char, Boolean, arrays, linked lists etc. are all abstract data types as only their behavior is specified not their implementation details like the algorithms used, memory allocation etc. *In other words, it is an idea that can be implemented in many different ways.*

Associative Arrays are abstract data types where each element in it is a key value pair. It doesn’t bother about the storage of its elements. It only cares about assigning value to its respective key. Keys are unique, but values need not be.

### Hash Functions

**Hashing** is the process of taking an input and producing a unique output. The function used for hashing is called **Hash Function**.

The hashed values are unique for each input and there is no way to get the original value from the hashed value. This is where it differentiates itself from Encoding or Encrypting.

When two inputs lead to common hash, it is called as **Collision**.

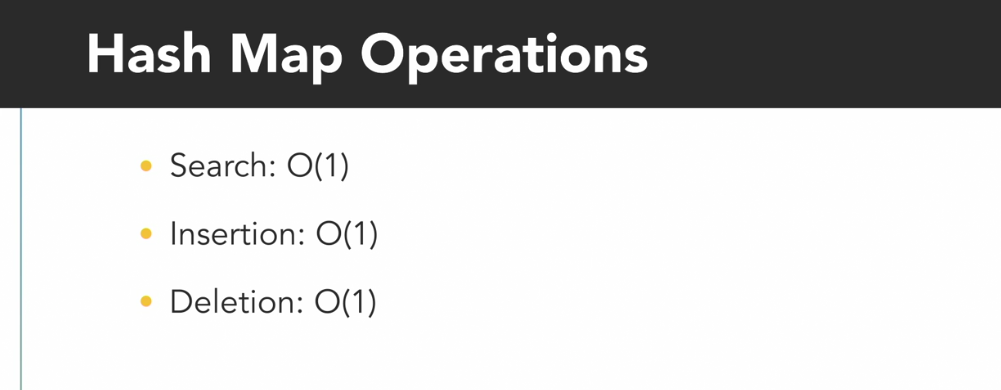
### Hash Table

is an implementation of an associative array. It is implemented using arrays, some modifications are done to get the desired functionality. The place where a key-value pair is added is referred to as **Bucket**. (will only one key value pair be stored in a bucket? - Yes) Always both the key and values are added, only one will never be added.

Once the hashed value of key is obtained, indices are generated using some logic. The indices are used to access the elements. There is no searching or traversing of any sort to obtain data. If there is a collision, we use different techniques like using linked lists to separate values stored at this index. This process is called **Separate Chaining**.

If the values of an object consist of only key and value pairs, it is called Dictionary.

Few languages like Java, Python have predefined hash functions. In JS you need to write own hash functions or use relevant node modules.



## Sets

It is a collection of data that have a similar property. A collection of clothes is a set, for example. A set doesn’t index its elements. Elements are unordered and unique. While using a set, all we care about is whether an element is part of set.

Behind the scenes, set is often implemented using hash tables. A given object is hashed and stored with its index value. To check if an object is present in set, this is hashed again to check if the object is already present. Set is never used for data retrieval.

* **For the data structures like linked lists, associative arrays, sets, trees etc. the way these are stored physically is unimportant. We are speaking of a logical data structure. It is just an idea. How it is actually stored in memory is trivial.**

Sets are ideal where we only care if a given object is present and don’t intend to maintain duplicate copies. Example presence of Mango in a fruit store.

## Trees

It is a collection of nodes. Each node can have **Child** nodes. A node can be both **Parent** and Child node at a time. The starting node is called as **Root** node. Childs to a same parent are called **Sibling** nodes. A node at the end i.e. without any child is called **Leaf** node.

A linked list is like a specialized form a tree where each node has only one child.

## Binary Search Trees (BST)

A binary tree is a tree where each node has a maximum of two child nodes (a left node and a right node). A binary search tree is a binary tree where the value in left node is always less than that of parent node and the value is right node is greater than that of Parent node.

Left Node < Parent Node < Right Node

BST is used behind the scenes two implement different data structures like sets in C++, sorted dictionaries in C#.

BST is always ordered and is quick to search a value.

Average case complexity of Search, Insert, and Delete Operations is **O(log n)**, where n is the number of nodes in the tree.

Simple explanation @ <http://lcm.csa.iisc.ernet.in/dsa/node91.html>

When the number of right nodes is not equal to that of left nodes, the tree is said to be **Un-balanced**.

If your data requires frequent searching, adding, deletion BSTs are ideal. They are always sorted and

Search – O (log N) or O(N) if unbalanced.

Addition – O (log N)

Deletion – O (log N)

With BSTs, there is an overhead to always balance the tree.

## Heaps

A Heap is a binary tree where nodes are filled from top to bottom and left to right. A given level is filled before moving onto next. Generally, Heaps are of two types, **Min-Heap** and **Max-Heap**.

Min-Heap: In Minimum Heap, the least value always occupies the root of the tree.

Filling the tree: 1) insert the first element in the root

2) insert the next element in the left node. If it is lesser than the parent node, swap the parent node value with this node’s value.

3) Fill the tree top to bottom, left to right performing the above check and swapping as and when required.

The values in siblings are never compared. We don’t bother about their values.

The above strategy ensures the Minimum value always bubbles up to the top.

Max Heap is like min heap except that the max value bubbles up to the top.

Heap is not completely ordered like a BST. Heaps are used to implement data structures like Queues, Priority Queues.

Find Min/Max – O(1)

Insertion – O(Log N)

Search – O(N)

Delete – O(N) -> search, delete and swap.

# Big O Notation

This is used to represent time complexity of an algorithm.

**O(1)** indicates that the algorithm would take constant time to perform the specific task regardless of the input. Example, adding an element at a specific index. In other words, the algorithm’s adding time complexity is O(1).

**O(n)** indicates that the time, algorithm takes ,to perform the operation would increase linearly with the input data size.

Generally, Big O Notation is used to represent the worst-case scenario.

