Top 8 Common Data Structures You Need to Know

40+ years later, that equation still holds true. That's why software engineering candidates have to demonstrate their understanding of data structures along with their applications.

Almost all problems require the candidate to demonstrate a deep understanding of data structures. It doesn't matter whether you have just graduated (from a university or coding bootcamp), or you have decades of experience.

Sometimes interview questions explicitly mention a data structure, for example, "given a binary tree." Other times it's implicit, like "we want to track the number of books associated with each author."

Learning data structures is essential even if you're just trying to get better at your current job. Let's start with understanding the basics.

What is a Data Structure?

Simply put, a data structure is a container that stores data in a specific layout. This "layout" allows a data structure to be efficient in some operations and inefficient in others. Your goal is to understand data structures so that you can pick the data structure that's most optimal for the problem at hand.

Why do we need Data Structures?

As data structures are used to store data in an organized form, and since data is the most crucial entity in computer science, the true worth of data structures is clear.

No matter what problem are you solving, in one way or another you have to deal with data — whether it's an employee's salary, stock prices, a grocery list, or even a simple telephone directory.

Based on different scenarios, data needs to be stored in a specific format. We have a handful of data structures that cover our need to store data in different formats.

Commonly used Data Structures

Let's first list the most commonly used data structures, and then we'll cover them one by one:

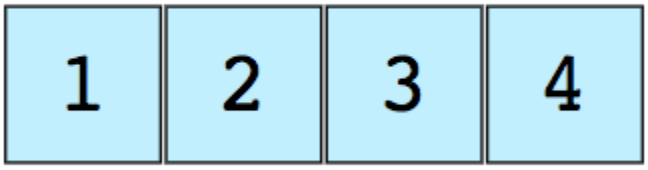
1. Arrays
2. Stacks

3. Queues
4. Linked Lists
5. Trees
6. Graphs
7. Tries (they are effectively trees, but it's still good to call them out separately).
8. Hash Tables

Arrays

An array is the simplest and most widely used data structure. Other data structures like stacks and queues are derived from arrays.

Here's an image of a simple array of size 4, containing elements (1, 2, 3 and 4).



Each data element is assigned a positive numerical value called the Index, which corresponds to the position of that item in the array. The majority of languages define the starting index of the array as 0.

The following are the two types of arrays:

- One-dimensional arrays (as shown above)
- Multi-dimensional arrays (arrays within arrays)

Basic Operations on Arrays

- Insert — Inserts an element at a given index
- Get — Returns the element at a given index
- Delete — Deletes an element at a given index
- Size — Gets the total number of elements in an array

Commonly asked Array interview questions

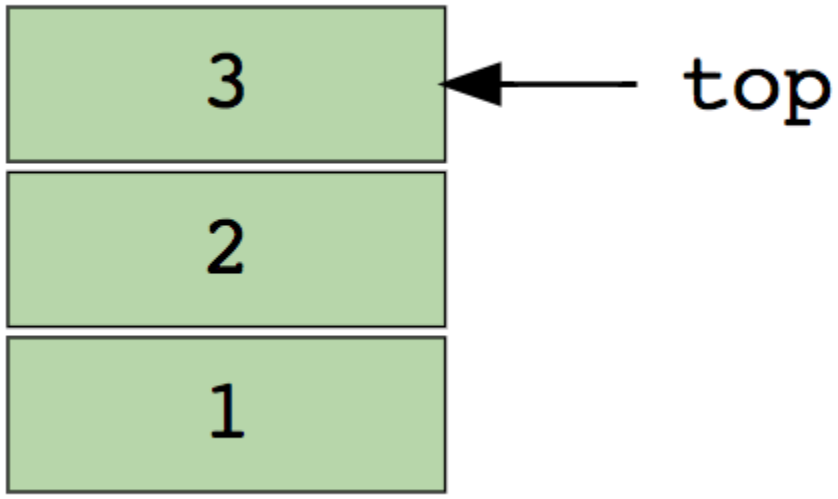
- Find the second minimum element of an array
- First non-repeating integers in an array
- Merge two sorted arrays
- Rearrange positive and negative values in an array

Stacks

We are all familiar with the famous Undo option, which is present in almost every application. Ever wondered how it works? The idea: you store the previous states of your work (which are limited to a specific number) in the memory in such an order that the last one appears first. This can't be done just by using arrays. That is where the Stack comes in handy.

A real-life example of Stack could be a pile of books placed in a vertical order. In order to get the book that's somewhere in the middle, you will need to remove all the books placed on top of it. This is how the LIFO (Last In First Out) method works.

Here's an image of stack containing three data elements (1, 2 and 3), where 3 is at the top and will be removed first:



Basic operations of stack:

- Push — Inserts an element at the top
- Pop — Returns the top element after removing from the stack
- isEmpty — Returns true if the stack is empty
- Top — Returns the top element without removing from the stack

Commonly asked Stack interview questions

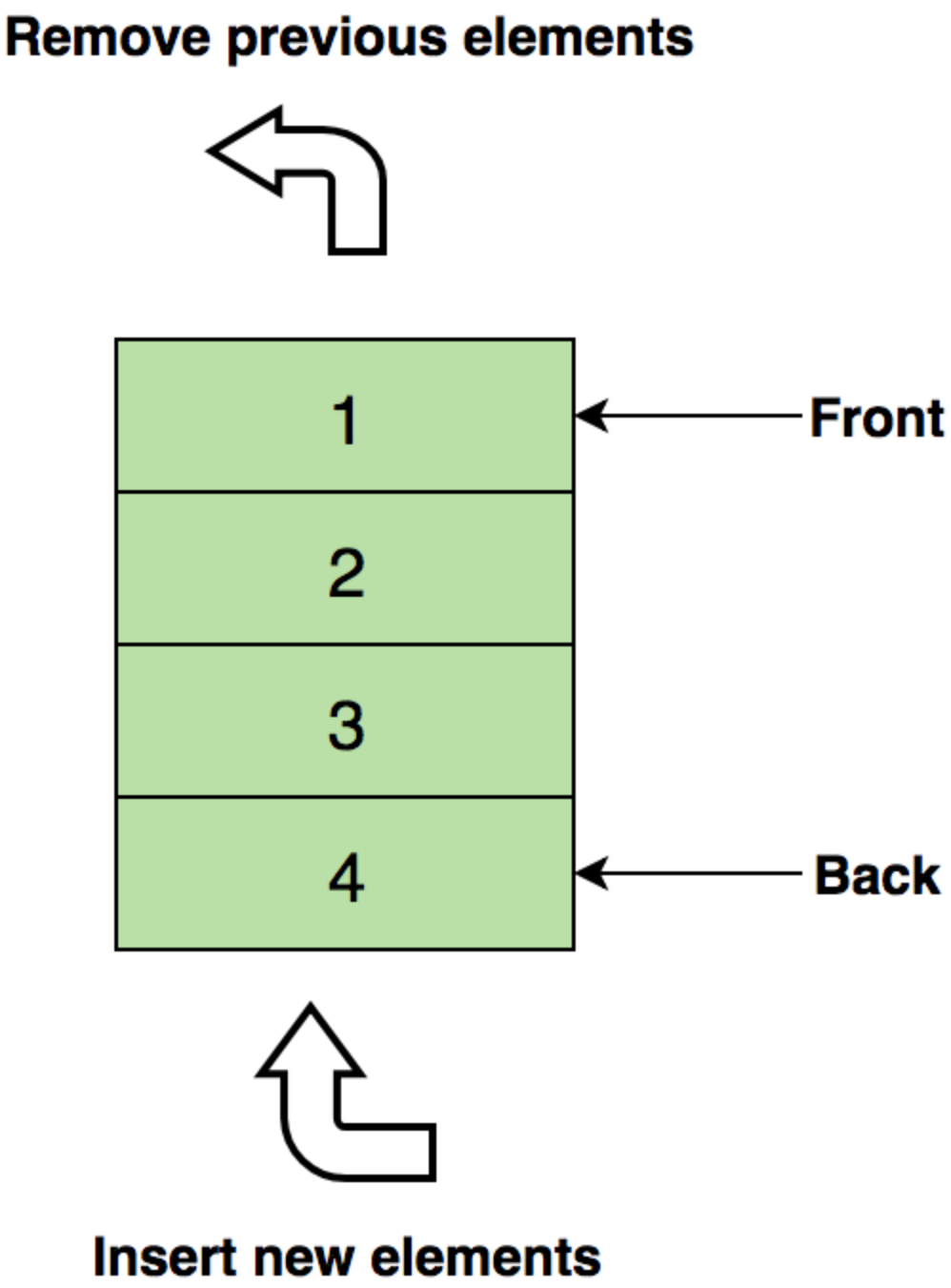
- Evaluate postfix expression using a stack
- Sort values in a stack
- Check balanced parentheses in an expression

Queues

Similar to Stack, Queue is another linear data structure that stores the element in a sequential manner. The only significant difference between Stack and Queue is that instead of using the LIFO method, Queue implements the FIFO method, which is short for First in First Out.

A perfect real-life example of Queue: a line of people waiting at a ticket booth. If a new person comes, they will join the line from the end, not from the start — and the person standing at the front will be the first to get the ticket and hence leave the line.

Here's an image of Queue containing four data elements (1, 2, 3 and 4), where 1 is at the top and will be removed first:



Basic operations of Queue

- Enqueue() — Inserts an element to the end of the queue
- Dequeue() — Removes an element from the start of the queue
- isEmpty() — Returns true if the queue is empty
- Top() — Returns the first element of the queue

Commonly asked Queue interview questions

- Implement stack using a queue
- Reverse first k elements of a queue
- Generate binary numbers from 1 to n using a queue

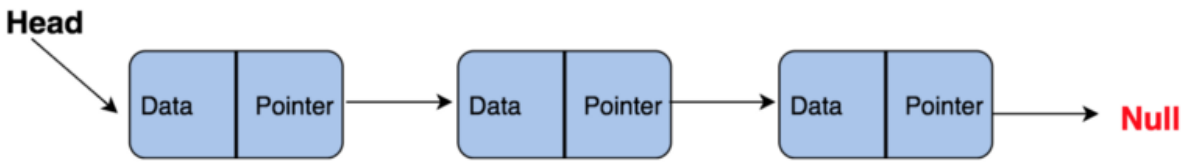
Linked List

A linked list is another important linear data structure which might look similar to arrays at first but differs in memory allocation, internal structure and how basic operations of insertion and deletion are carried out.

A linked list is like a chain of nodes, where each node contains information like data and a pointer to the succeeding node in the chain. There's a head pointer, which points to the first element of the linked list, and if the list is empty then it simply points to null or nothing.

Linked lists are used to implement file systems, hash tables, and adjacency lists.

Here's a visual representation of the internal structure of a linked list:



Following are the types of linked lists:

- Singly Linked List (Unidirectional)
- Doubly Linked List (Bi-directional)

Basic operations of Linked List:

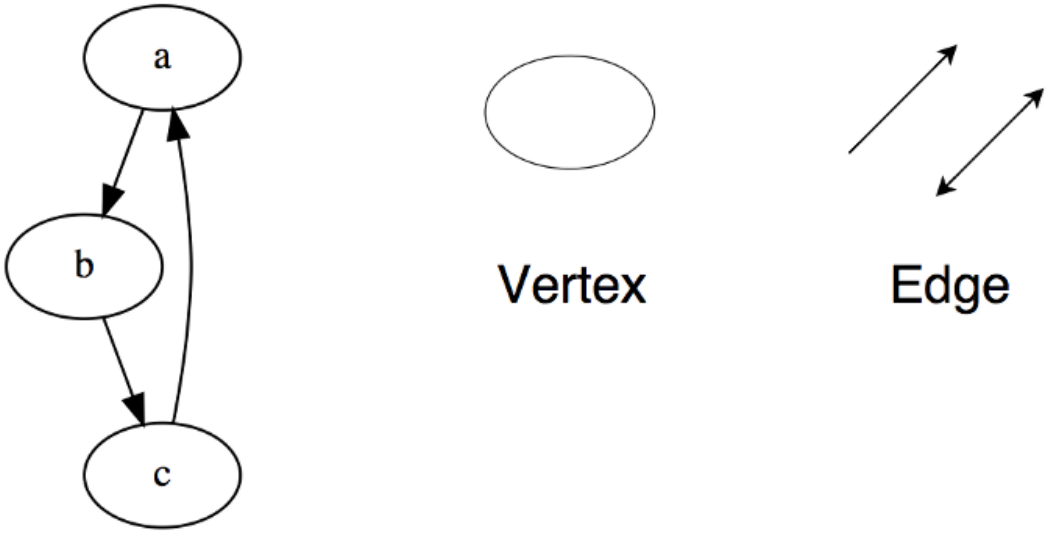
- *InsertAtEnd* — Inserts a given element at the end of the linked list
- *InsertAtHead* — Inserts a given element at the start/head of the linked list
- *Delete* — Deletes a given element from the linked list
- *DeleteAtHead* — Deletes the first element of the linked list
- *Search* — Returns the given element from a linked list
- *isEmpty* — Returns true if the linked list is empty

Commonly asked Linked List interview questions

- Reverse a linked list
- Detect loop in a linked list
- Return Nth node from the end in a linked list
- Remove duplicates from a linked list

Graphs

A graph is a set of nodes that are connected to each other in the form of a network. Nodes are also called vertices. A pair(x,y) is called an edge, which indicates that vertex x is connected to vertex y. An edge may contain weight/cost, showing how much cost is required to traverse from vertex x to y.



Types of Graphs:

- Undirected Graph
- Directed Graph

In a programming language, graphs can be represented using two forms:

- Adjacency Matrix
- Adjacency List

Common graph traversing algorithms:

- Breadth First Search
- Depth First Search

Commonly asked Graph interview questions

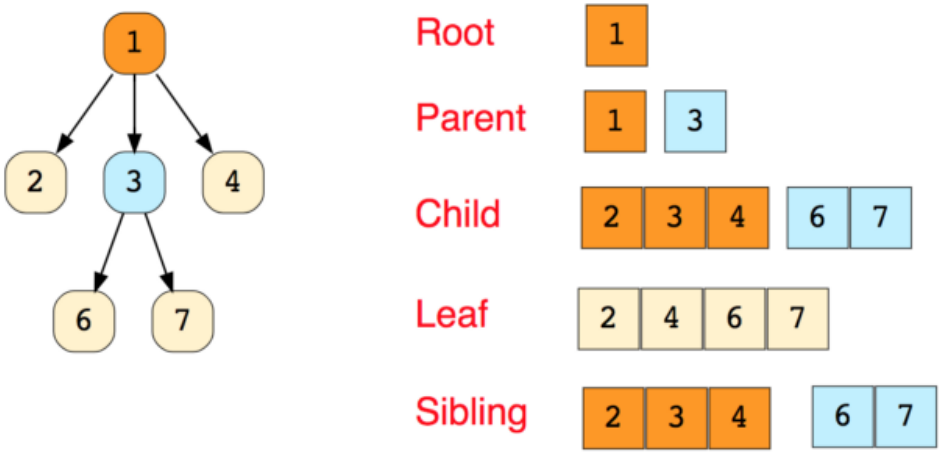
- Implement Breadth and Depth First Search
- Check if a graph is a tree or not
- Count the number of edges in a graph
- Find the shortest path between two vertices

Trees

A tree is a hierarchical data structure consisting of vertices (nodes) and edges that connect them. Trees are similar to graphs, but the key point that differentiates a tree from the graph is that a cycle cannot exist in a tree.

Trees are extensively used in Artificial Intelligence and complex algorithms to provide an efficient storage mechanism for problem-solving.

Here's an image of a simple tree, and basic terminologies used in tree data structure:



The following are the types of trees:

- N-ary Tree

- Balanced Tree
- Binary Tree
- Binary Search Tree
- AVL Tree
- Red Black Tree
- 2–3 Tree

Out of the above, Binary Tree and Binary Search Tree are the most commonly used trees.

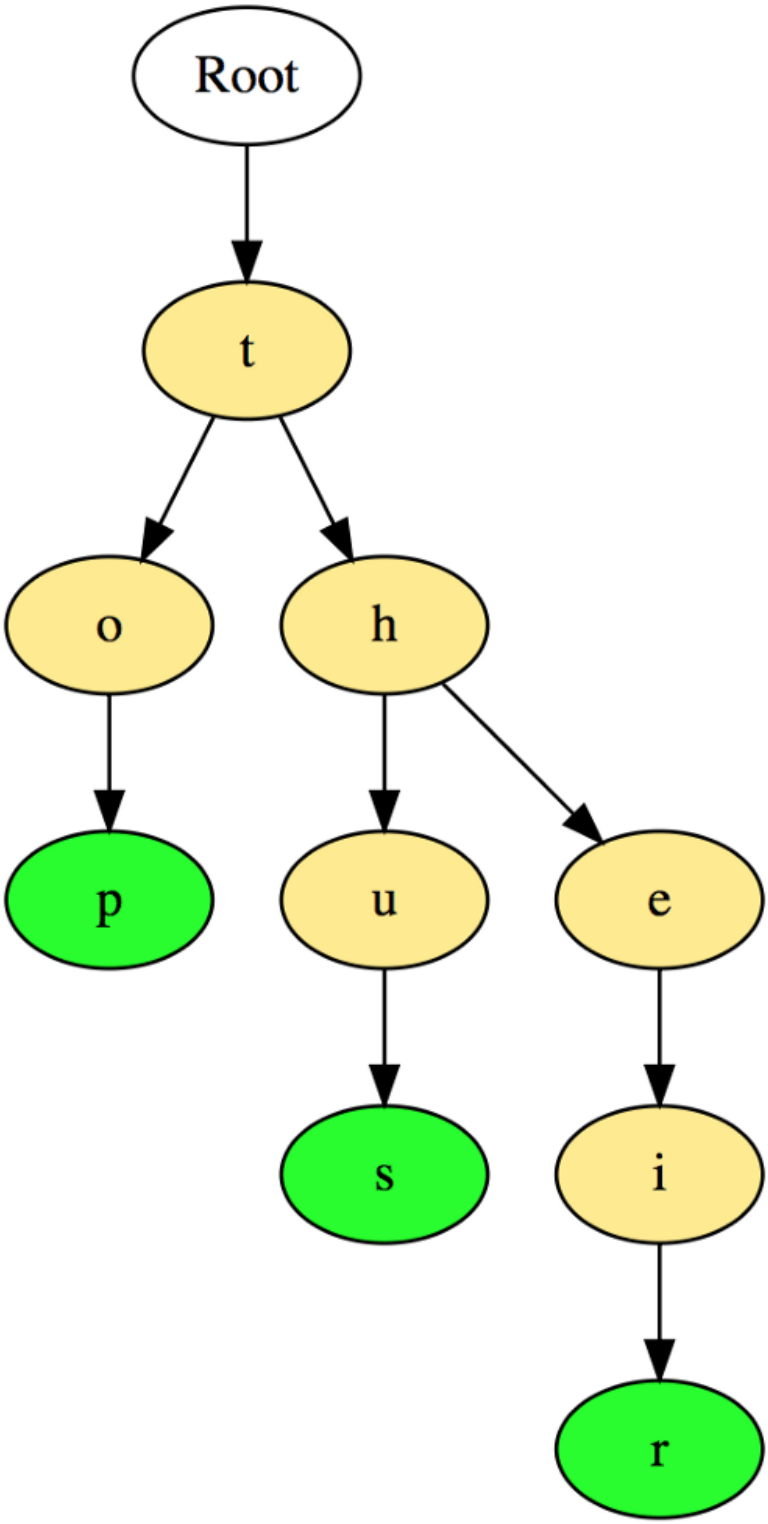
Commonly asked Tree interview questions

- Find the height of a binary tree
- Find kth maximum value in a binary search tree
- Find nodes at “k” distance from the root
- Find ancestors of a given node in a binary tree

Trie

Trie, which is also known as “Prefix Trees”, is a tree-like data structure which proves to be quite efficient for solving problems related to strings. It provides fast retrieval, and is mostly used for searching words in a dictionary, providing auto suggestions in a search engine, and even for IP routing.

Here’s an illustration of how three words “top”, “thus”, and “their” are stored in Trie:


Root
t
o
h
p
u
e
s
i
r

The words are stored in the top to the bottom manner where green colored nodes “p”, “s” and “r” indicates the end of “top”, “thus”, and “their” respectively.

Commonly asked Trie interview questions:

- Count total number of words in Trie
- Print all words stored in Trie
- Sort elements of an array using Trie
- Form words from a dictionary using Trie
- Build a T9 dictionary

Hash Table

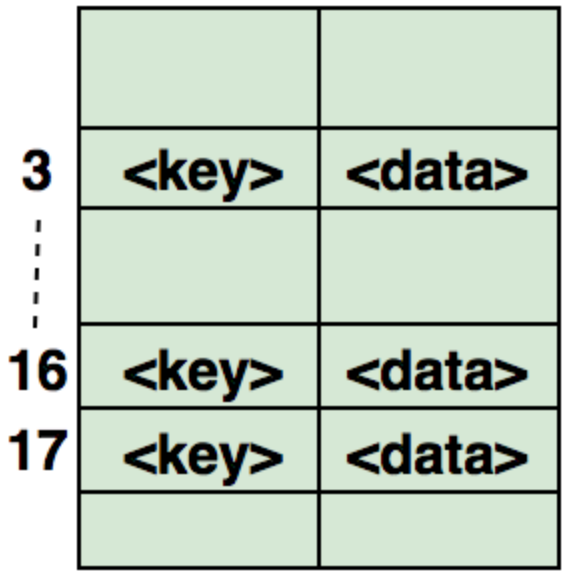
Hashing is a process used to uniquely identify objects and store each object at some pre-calculated unique index called its “key.” So, the object is stored in the form of a “key-value” pair, and the collection of such items is called a “dictionary.” Each object can be searched using that key. There are different data structures based on hashing, but the most commonly used data structure is the hash table.

Hash tables are generally implemented using arrays.

The performance of hashing data structure depends upon these three factors:

- Hash Function
- Size of the Hash Table
- Collision Handling Method

Here’s an illustration of how the hash is mapped in an array. The index of this array is calculated through a Hash Function.



Commonly asked Hashing interview questions

- Find symmetric pairs in an array
- Trace complete path of a journey
- Find if an array is a subset of another array
- Check if given arrays are disjoint

The above are the top eight data structures that you should definitely know before walking into a coding interview.

Original article source at https://www.freecodecamp.org