DATA STRUCTURES AND ALGORITHMS BUT SIMPLER

How I Taught Myself Linked Lists

Breaking down the definition of linked lists part by part.

Cleon W · Follow

Published in Towards Data Science

7 min read · May 31, 2020

What is a linked list?

Linked lists are amongst the simplest and most common data structures. If you have ever attempted to learn more about basic data structures, it is likely that you've come across linked lists and have read definitions similar to the following:

> *A linked list is a data structure where the objects are arranged in a linear order. Unlike an array, however, in which the linear order is determined by the array indices, the order in a linked list is determined by a pointer in each object.*

Based on personal experience when learning about data structures, the above probably makes no sense to the absolute beginner. As such, this post aims to show that linked lists are conceptually simple by breaking down its definition in an intuitive and layman way to you, as though you were a complete beginner.

We will leave the "So what?" of studying linked lists to the end of this post, as I believe that we will be able to ask more meaningful 'so whats' when equipped with a better understanding of linked lists. Note that we will be referencing the above definition throughout the article.

So let's get started.

What is a data structure?

> *A linked list is a data structure...*

Since linked list is a type of data structure, it pays to briefly understand what is a data structure.

Simply put, a data structure is how data is stored and organised.

Then, depending on how the data is stored and organised, different operations can be applied to the data.

What differentiates data structures from each other can either be how the data is stored and organised, what can be done with and/or to the data, or both. A bulk of the problems in computer science requires some sort of data, and our need to efficiently retrieve information from the data depending on the problem type is what motivates the creation of the different data structures that we have today.

For example, a linked list stores and organises data in a linear order with the help of *pointers* (this is the "how data is organised"), and these pointers allow us to easily insert and remove elements without reorganising the entire data structure (this is the "what can be done"). This makes a linked list a *linear data structure* (as opposed to a non-linear one).

What is linear order?

> *... where the objects are arranged in a linear order.*

Storing elements in a linear order means organising them in a sequential order such that there is:

- Only one first element and it has only one next element
- Only one last element and it has only one previous element
- All other elements have a previous and next element

For example, apart from linked lists, *arrays* are also linear data structures that store data in a linear order.

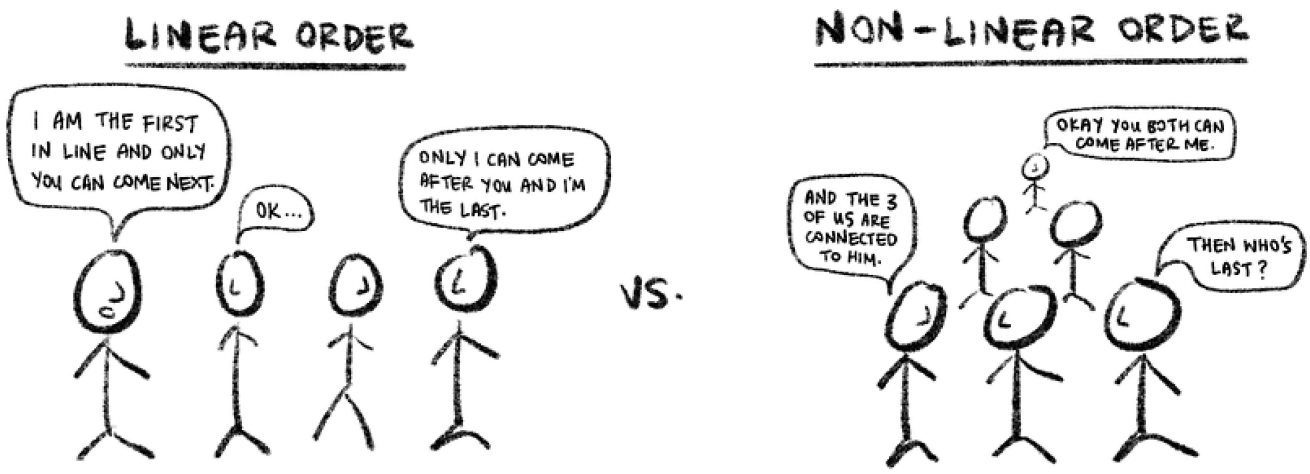


Image drawn by author.

Again, whether linear or non-linear data structures are better depends on your use case. Both types have their pros and cons, as usual.

Okay, but what's wrong with arrays?

> *... Unlike an array, however, ...*

By now, you should have noticed that linked lists are most commonly compared to arrays. What is wrong (or right) with arrays depends on your use case. As with any other data structure, an array does have its own pros and cons.

While arrays offer very fast indexed access and assignment of elements, they have fixed, immutable lengths. That means that they cannot grow or shrink, and it is impossible to append or insert items. Consequently, we must know how many elements we want to store when creating an array, which might cause certain issues depending on the problem at hand.

Dynamic arrays (e.g. Python's **list**) have variable lengths and this is done by reserving additional memory space. However, inserting an element at the start of a dynamic array can be really expensive compared to inserting it at the end. This might pose as a huge blocker, again depending on the problem that you are trying to address.

Using Python's list object, we can run the following code to illustrate this:

```
n = 500000
example_list1 = []
example_list2 = []
```

```
# Slow insertion of at the front of Python lists
for i in range(0, n):
    example_list1.insert(0, i)

# Fast insertion of at the end of Python lists
for i in range(0, n):
    example_list2.insert(i, i)
```

Recording and printing the time taken to completely insert 500,000 elements, we get:

```
Insertion at start took: 110.1042 seconds
Insertion at end took: 0.1853 seconds
```

With a dynamic array, in order to insert an element in the first position (or first index), you would need to move all of the other elements ‘to the right’ to make room for this new element. Naturally, you can imagine that if you have an array with a large number of elements this would be really inefficient. Furthermore, there is a possibility of you needing to allocate more memory space to the array before shifting the elements.

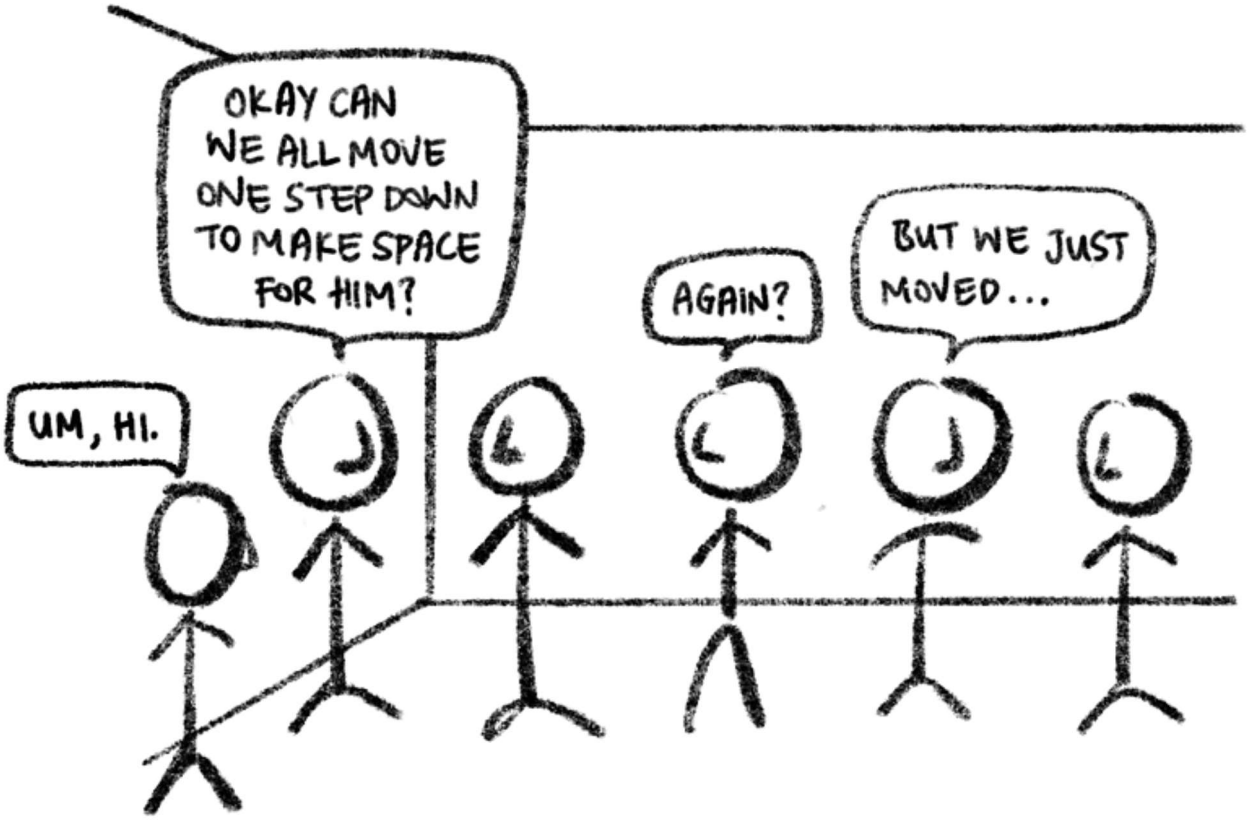


Image drawn by author.

This is where linked lists can do better.

Right, so how are linked lists different from arrays?

> *...the order in a linked list is determined by a pointer in each object.*

Pointers, pointers, pointers.

Despite both linked lists and arrays being linear data structures, a linked list maintains the linear order of its elements using pointers (as compared to the indices in arrays). The use of pointers allows us to insert or remove nodes at arbitrary positions more efficiently than we could with arrays.

In its most basic form (i.e. containing only the required attributes), linked lists are made up of:

1. Objects called **Nodes** with the attributes: (a) **Item**, which is assigned the data that it stores *(required)*, (b) **Next pointer**, which points to the next Node in the linear order *(required)* and (c) **Previous pointer**, which points to the previous Node in the linear order *(not required)*.
2. A pointer to the head of the list (required)
3. A pointer to the tail of the list (not required)

To see how a linked list allows us to efficiently insert objects at the head and tail, we will implement a linked list with 3 nodes holding the integers 14, 5 and 22 as data in linear order. Each node has a pointer to the next node and the linked list has both a pointer to the head of the list and a pointer to its tail.

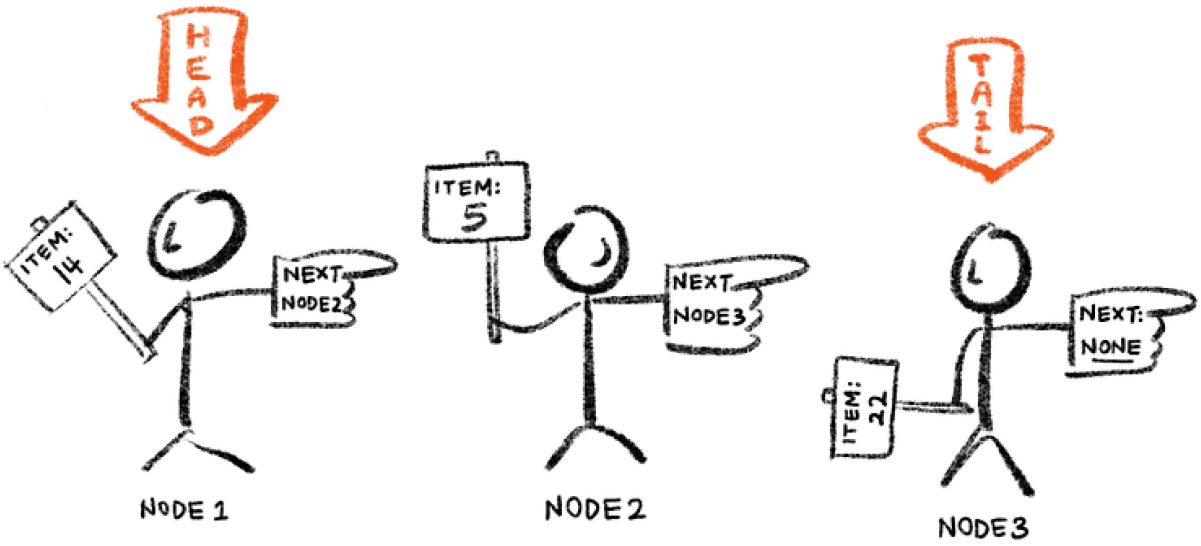


Image drawn by author.

Since the position of a node in the list is stored in the pointer of the previous node, the nodes of a linked list do not necessarily have to be organised consecutively in memory. In fact, it can be stored anywhere in memory. Hence, it would look more like this:

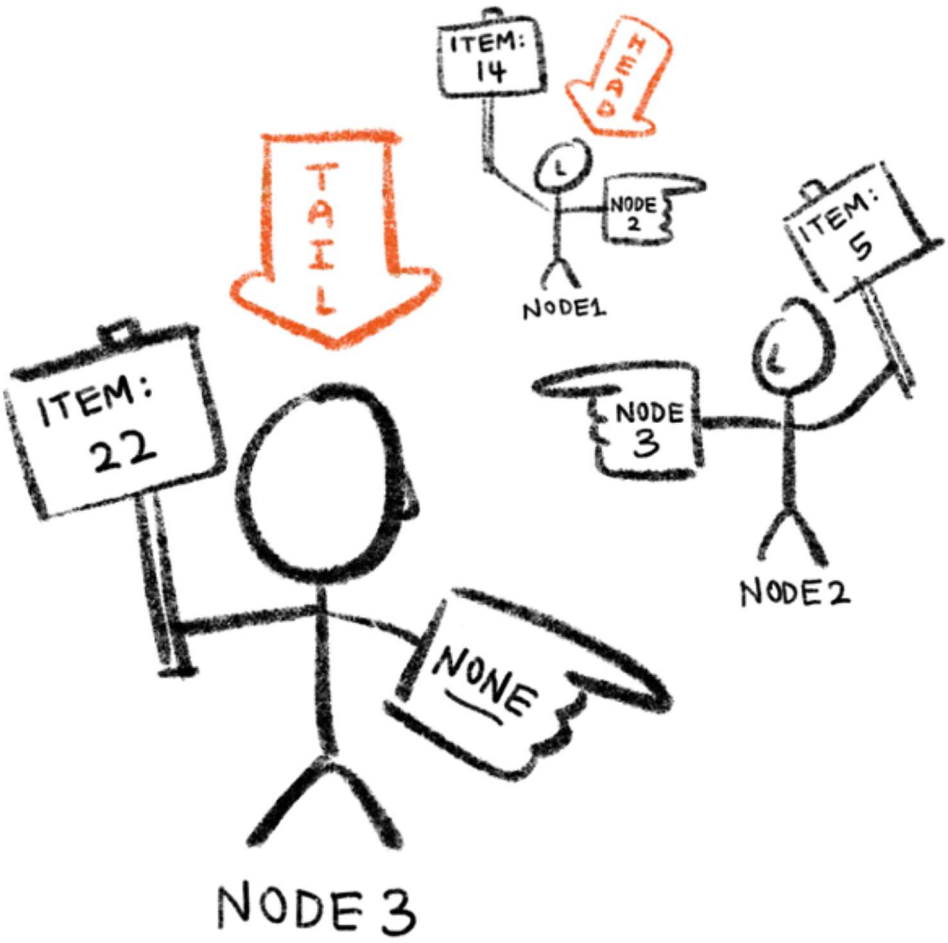


Image drawn by author.

Inserting at the tail

To insert a node that carries integer 50 at the tail of the linked list, we do the following:

1. Create a new node with its **item** = 50
2. Let its **next pointer** point to *None*
3. Let the current tail node point to this new node instead of *None*
4. Reset the **tail pointer** of the linked list to point to this newly created node

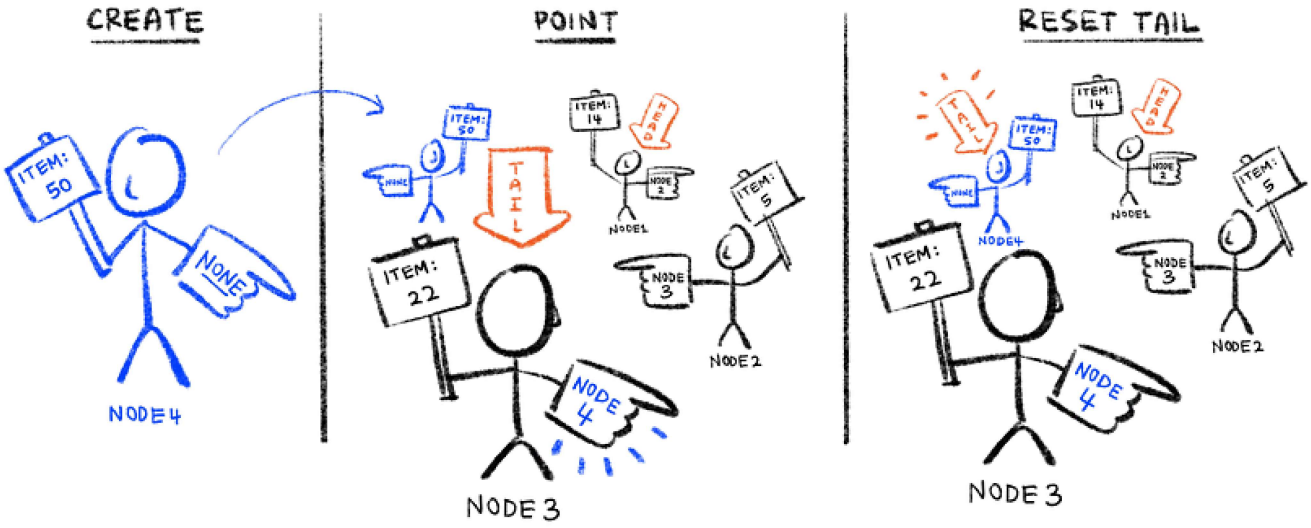


Image drawn by author.

Inserting at the head

Similarly, to add an item at the head of the linked list, we do the following:

1. Create a new Node
2. Let its **next pointer** point to the current head of the linked list
3. Reset the **head pointer** of the linked list to point to this newly created node

That essentially is a linked list in its simplest form.

So what?

As promised, so what?

We can see that the main benefit of a linked list as compared to a (dynamic) array is that it allows for fast insertion (and removal) of items. Unlike arrays, there is no need to restructure the entire data structure and reallocate memory for that restructuring. This stems from the use of pointers to maintain the linear order of the data, which removes the need for the data items to be stored in a consecutive manner in memory. In other words, linked lists allow the insertion and removal of items using a fixed number of operations (as listed in the steps above), regardless of the length of the list.

Using these fundamental concepts of a node, how it stores data and points to its neighbour(s), we can build other (more complicated) types of linked lists depending on our needs, such as *doubly linked lists* and *circular linked lists*.

More excitingly, understanding linked lists and their nodes can lead us to exploring and understanding other data structures and even algorithms. A linked list is one of the many ways to implement other abstract data types such as stacks, queues, hash tables and graphs, which we will sink our teeth into another day.

For the record, this is how the above singly linked list will look like in a conventional textbook, without the colours of course:

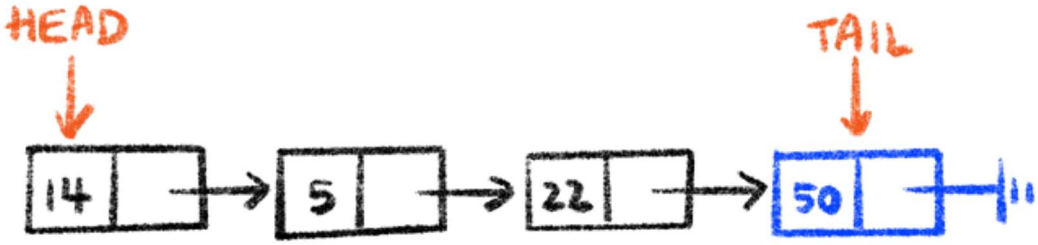


Image drawn by author.

Data Structures | Computer Science | Data Science | Programming | Algorithms

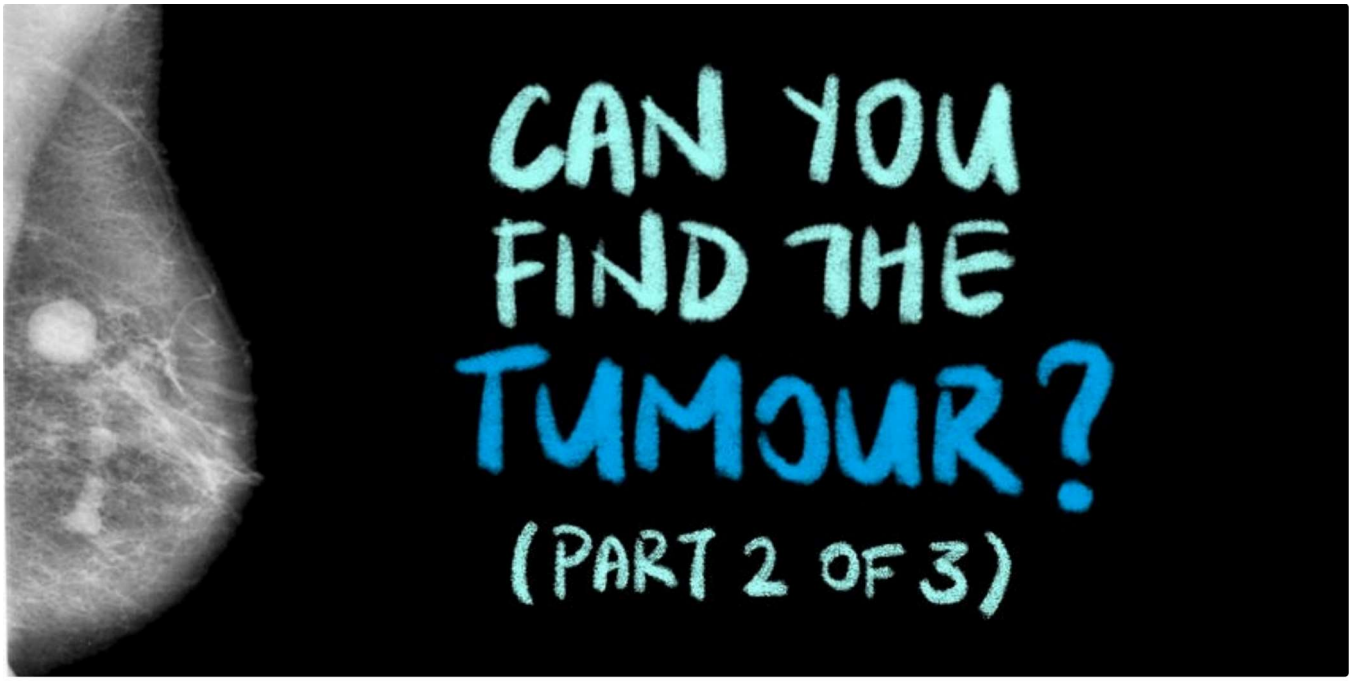
Written by Cleon W

125 Followers · Writer for Towards Data Science

Ex-founder, Product @ Crypto.com. I like to write. https://cleonwong.com

Follow

More from Cleon W and Towards Data Science

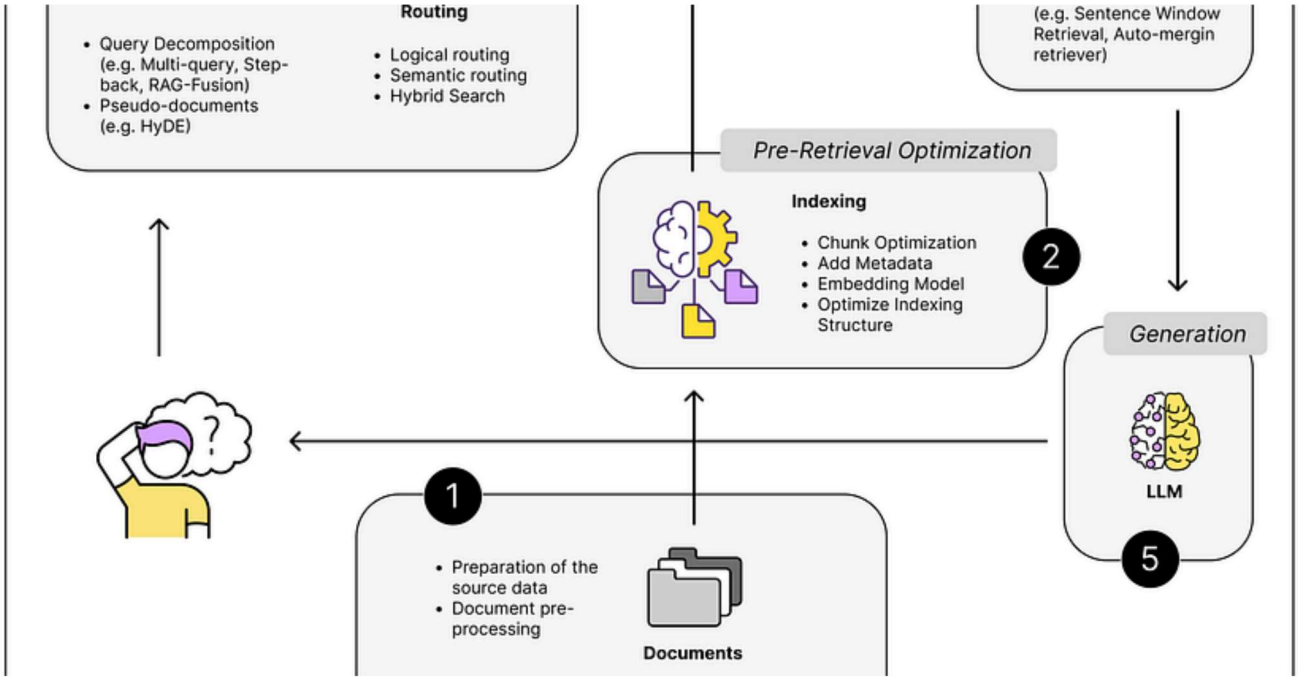


Cleon W in Towards Data Science

Can You Find the Breast Tumours? (Part 2 of 3)

A step-by-step guide to implementing a deep learning semantic segmentation pipeline on real-world mammograms in Tensorflow 2.

Mar 4, 2021 111



Dominik Polzer in Towards Data Science

17 (Advanced) RAG Techniques to Turn Your LLM App Prototype into a Production-Ready Solution

A collection of RAG techniques to help you develop your RAG app into something robust that will last

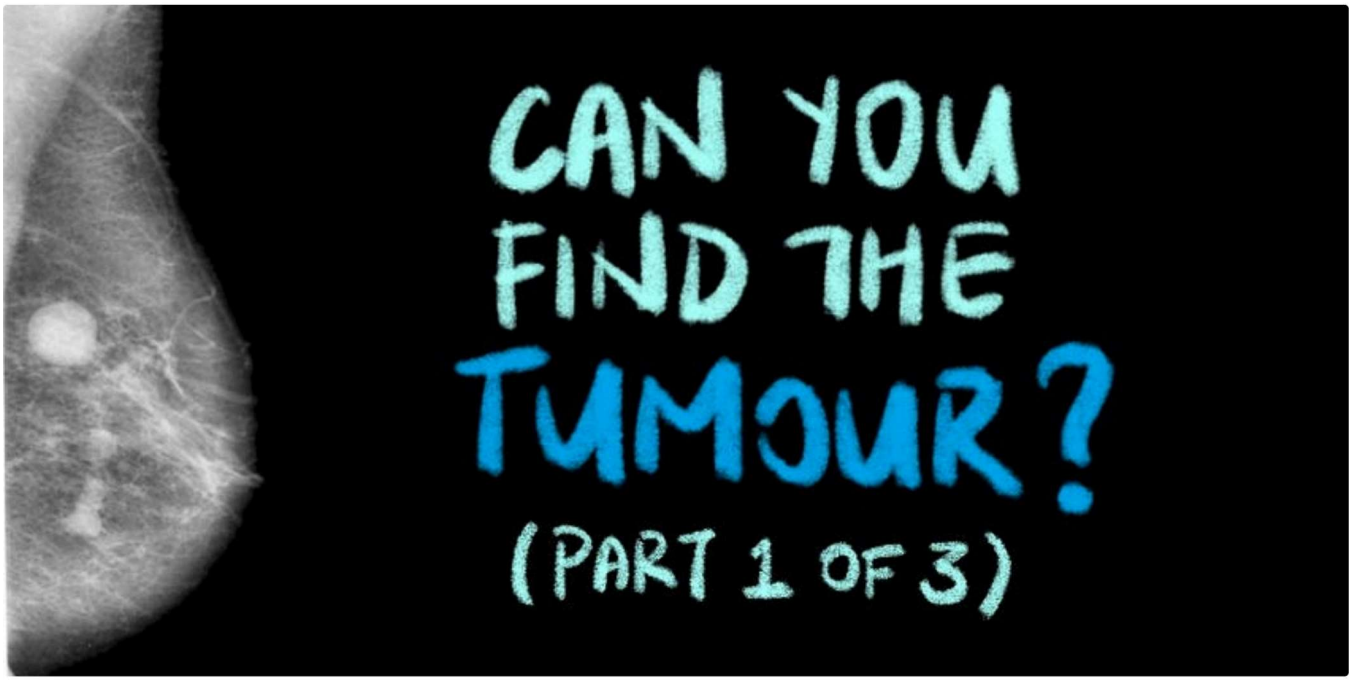
Jun 26 1.94K 20

Mauro Di Pietro in Towards Data Science

GenAI with Python: RAG with LLM (Complete Tutorial)

Build your own ChatGPT with multimodal data and run it on your laptop without GPU

Jun 28 714 13



Cleon W in Towards Data Science

Can You Find the Breast Tumours? (Part 1 of 3)

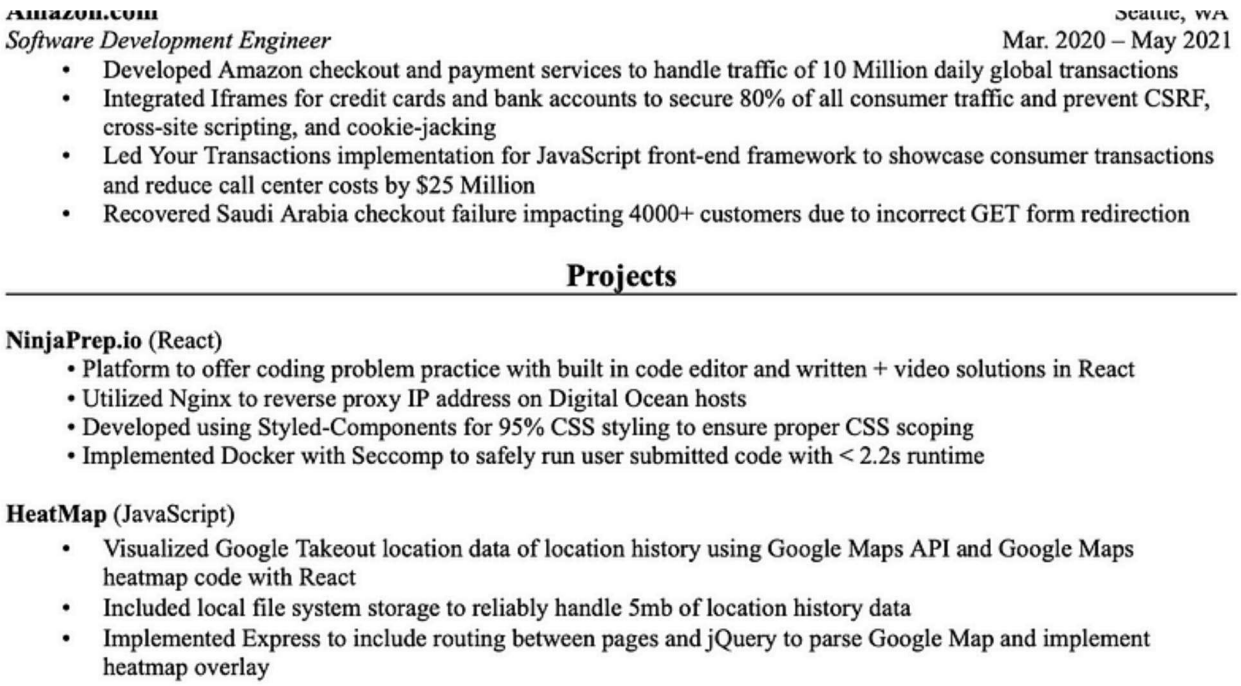
A step-by-step guide to implementing a deep learning semantic segmentation pipeline on real-world mammograms in Tensorflow 2.

Feb 25, 2021 421 3

See all from Cleon W

See all from Towards Data Science

Recommended from Medium



Amazon.com Seattle, WA
Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)
- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)
- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

Jun 1 13.7K 204

nums = [2, 3, 5, 9] k = 2

left, right, mid = 2, 9, 5

Metalesaek

Understanding Dynamic Programming With Leetcode Examples Part-II

Dynamic programming is a highly efficient technique widely used to tackle various challenges. To master this technique, it is essential to...

Jun 13 6

Lists

General Coding Knowledge

20 stories · 1395 saves

Predictive Modeling w/ Python

20 stories · 1388 saves

Coding & Development

11 stories · 703 saves

Practical Guides to Machine Learning

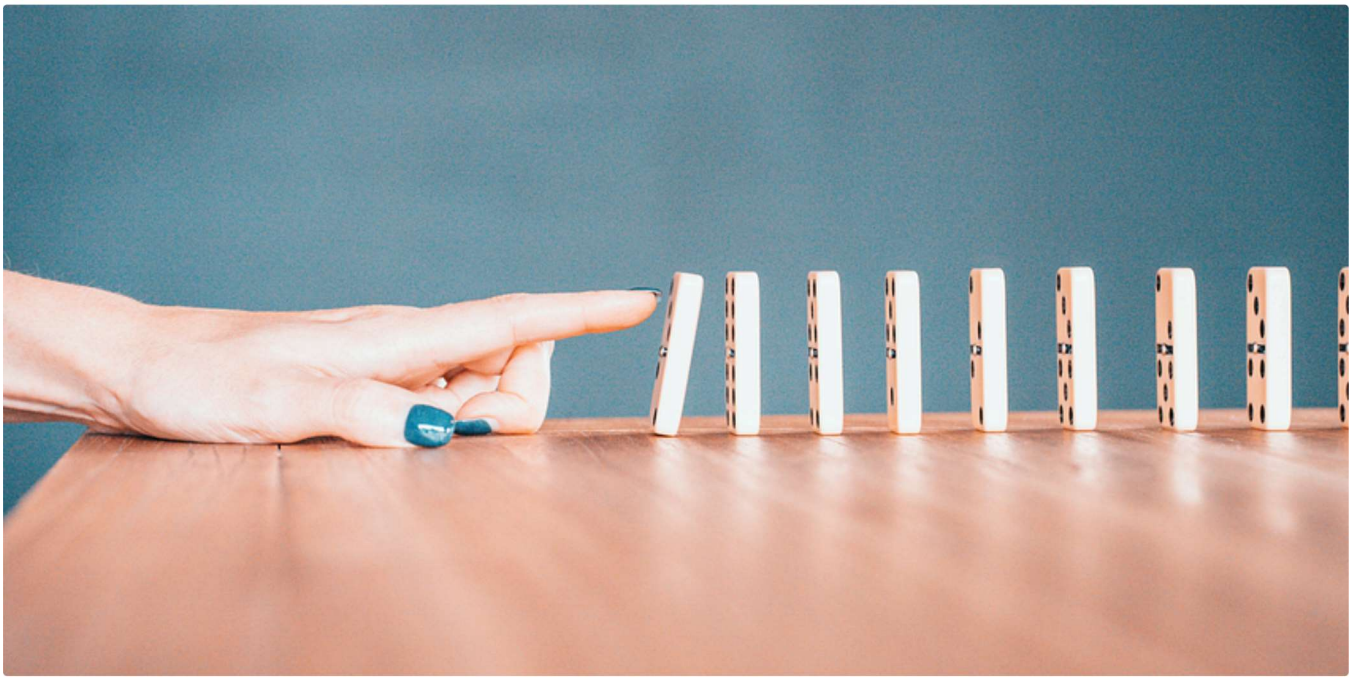
10 stories · 1675 saves

Md Shadekur Rahman in coding interview preparation

My Interview Experience at Meta [E4 Offer]

A recruiter reached out to me via email at the end of Nov 2023 to see if I am interested discussing about a backend role at Meta. I replied...

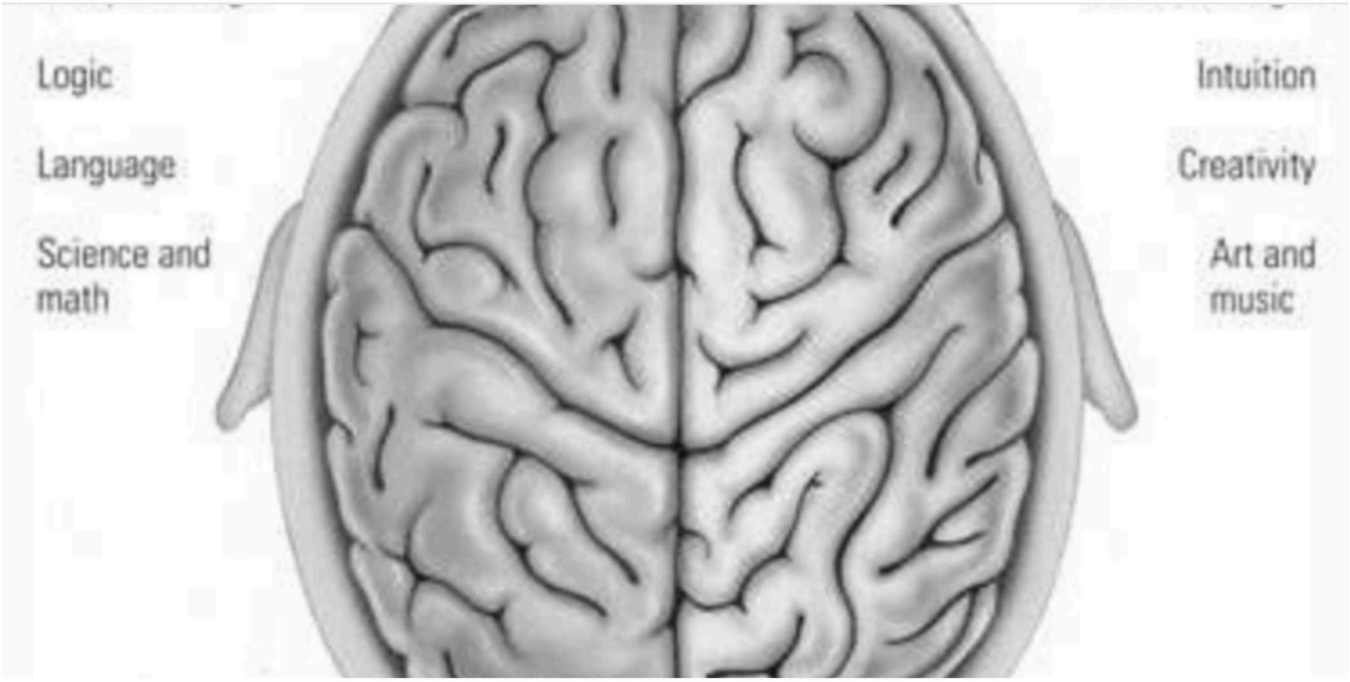
Jul 4



Mark Okafor

Introduction to Data Structures and Algorithm—Algorithms and Big O Notation.

Apr 6



Sufyan Maan, M.Eng in ILLUMINATION

What Happens When You Start Reading Every Day

Think before you speak. Read before you think.—Fran Lebowitz

Mar 12 26K 577

```
*__, a, b, *_ = [1, 2, 3, 4, 5, 6]
print(__, _)
```

What does this print?

- A) Syntax error
- B) `[1] [4, 5, 6]`
- C) `[1, 2] [5, 6]`
- D) `[1, 2, 3] [6]`
- E) `<generator object <genexpr> at 0x1003847c0>`

Liu Zuo Lin

You're Decent At Python If You Can Answer These 7 Questions Correctly

No cheating pls!!