DATA STRUCTURES

GR-TECHMIND

INTRODUCTION TO DATA STRUCTURES

What are data structures?

Data structures are like the building blocks of programming. They are organized ways of storing, managing, and accessing data in a computer's memory.

Example

Think of data structures as containers - like a bag, a box, or a shelf - that hold different types of items neatly.

INTRODUCTION TO DATA STRUCTURES

Importance of Data Structures in Programming

Data structures are crucial because they determine how efficiently we can perform operations on our data. Choosing the right data structure can significantly impact the speed and efficiency of a program.

Example

Imagine you're organizing a library. If you arrange books by genre, it's much faster for readers to find what they want compared to randomly scattered books.

BUILT-IN DATA STRUCTURES IN PYTHON

Basic to know first!!

📋 Lists [ ]

🔄 Tuples ( )

🔮 Sets { }

📚 Dictionaries { }

📜 Strings (' ') or (" ")

BUILT-IN FUNCTIONS

LISTS FUNCTION [11]

append ( )
clear ( )
copy ( )
count ( )
extend ( )
index ( )
insert ( )
pop ( )
remove ( )
reverse ( )
sort ( )

SETS FUNCTION {17}

add ( )
clear ( )
copy ( )
difference ( )
difference_update ( )
discard ( )
intersection ( )
intersection_update ( )
isdisjoint ( )
issubset ( )
issuperset ( )
pop ( )
remove ( )
symmetric_difference ( )
symmetric_difference-update ( )
union ( )
update ( )

DICTIONARIES FUNCTION {11}

clear ( )
copy ( )
fromkeys ( )
get ( )
items ( )
keys ( )
pop ( )
popitem ( )
setdefault ( )
update ( )
values ( )

TUPLE FUNCTION (2)

count ( )
index ( )

GR-TECHMIND

BUILT-IN FUNCTIONS

STRING FUNCTION (41)

- capitalize ( )
- casefold ( )
- center ( )
- count ( )
- encode ( )
- endswith ( )
- expandtabs ( )
- find ( )
- format ( )
- index ( )
- isalnum ( )
- isalpha ( )
- isdecimal ( )
- isdigit ( )
- isidentifier ( )
- islower ( )
- isnumeric ( )
- isprintable ( )
- isspace ( )
- istitle ( )
- isupper ( )
- ljust ( )
- rjust ( )
- strip ( )
- lstrip ( )
- rstrip ( )
- partition ( )
- rpartition ( )
- rfind ( )
- rindex ( )
- lower ( )
- upper ( )
- replace ( )
- join ( )
- split ( )
- rsplit ( )
- splitlines ( )
- startswith ( )
- swapcase ( )
- title ( )
- zfill ( )

BUILT-IN DATA STRUCTURES IN PYTHON

📋 LISTS [] MUTABLE

Lists are like dynamic arrays that can store a collection of items, allowing you to add, remove, and modify elements easily.

Example

Think of a list like a shopping list where you can add or remove items as needed.

BUILT-IN DATA STRUCTURES IN PYTHON

“REAL TIME EXAMPLES“

GPS Navigation: *A linked list of map data can be used to store and manage a list of locations and routes, allowing users to easily navigate to their destination.*

To-Do Lists: *To-do lists are a simple example of a list that many people use in their daily lives to keep track of tasks and goals.*

Shopping Lists: *Shopping lists are another example of a simple list that people use to keep track of items they need to buy.*

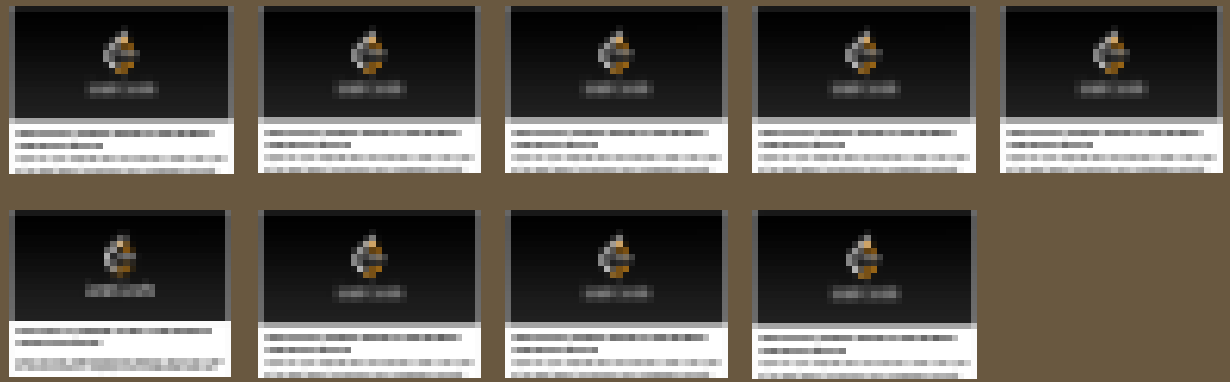
Music Playlists: *Music playlists are a type of list that people use to organize and play their favorite songs in a specific order.*

Contact Lists: *Contact lists are a type of list that people use to store and manage their contacts' information, such as names, phone numbers, and email addresses.*

"Solve More Problems, Grow More Smart – Give It a try!"

LEETCODE PROBLEMS (Provided links)

LISTS



GR-TECHMIND

BUILT-IN DATA STRUCTURES IN PYTHON

🔄 TUPLES () IMMUTABLE

Tuples are similar to lists but are immutable, meaning their contents cannot be changed after creation.

Example

A tuple is like a sealed envelope; once you've put something in it, you can't alter its contents without breaking the seal.

BUILT-IN DATA STRUCTURES IN PYTHON

“REAL TIME EXAMPLES“

Immutable Data: *Tuples can be used to store data that should not be changed, such as a tuple of (year, month, day) for a date*

Data Analysis: *Tuples can be used to store data in a structured way, such as a tuple of (name, age, gender) for a list of people, which can be useful for data analysis*

Database Queries: *Tuples can be used to represent the results of a database query, where each tuple represents a row of data*

Function Return Values: *Tuples can be used to return multiple values from a function, such as a tuple of (min, max) for a function that calculates the minimum and maximum values in a list*

BUILT-IN DATA STRUCTURES IN PYTHON

🔮 SETS {} UNIQUE

Sets are collections of unique elements, making them ideal for tasks like removing duplicates or testing membership in a group.

Example

Imagine a set of keys - you can't have duplicates, and you can quickly check if a key is present or not.

BUILT-IN DATA STRUCTURES IN PYTHON

“REAL TIME EXAMPLES“

Kitchen Sets: *In the kitchen, we often use sets of utensils, plates, and cups that are kept separately for easy access.*

Shopping Malls: *Shopping malls often have sets of items that are grouped together, such as sets of clothes, shoes, and accessories.*

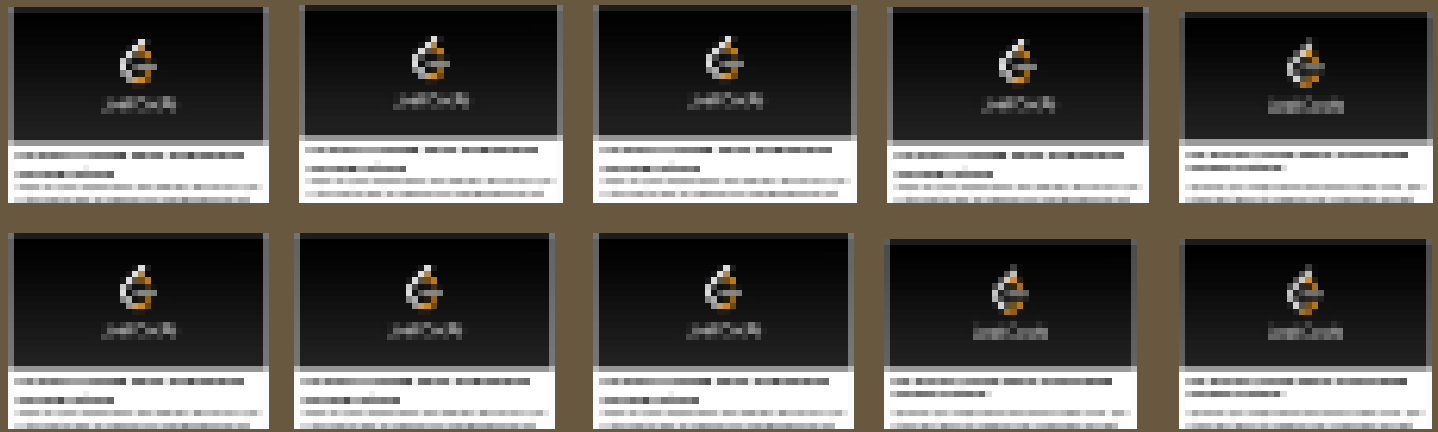
Your Favorite Playlist: *Playlists are a type of set that people use to organize and play their favorite songs in a specific order.*

Grade-Level School Books: *In schools, students are often given sets of books that are specific to their grade level.*

"Solve More Problems, Grow More Smart – Give It a try!"

LEETCODE PROBLEMS (Provided links)

S E T S



GR-TECHMIND

BUILT-IN DATA STRUCTURES IN PYTHON

📚 DICTIONARIES {} KEY , VALUE

Dictionaries are key-value pairs, allowing you to store and retrieve data using a unique key.

Example

A dictionary is like a real-world dictionary, where you look up a word (the key) to find its definition (the value).

BUILT-IN DATA STRUCTURES IN PYTHON

"REAL TIME EXAMPLES"

Phone Book: *A phone book is a common example of a dictionary, where names are mapped to phone numbers.*

Online Shopping: *Online shopping websites often use dictionaries to store product information, where each product has a unique identifier (key) and a set of attributes (values).*

Real Estate Listings: *Real estate websites often use dictionaries to store property information, where each property has a unique identifier (key) and a set of attributes (values).*

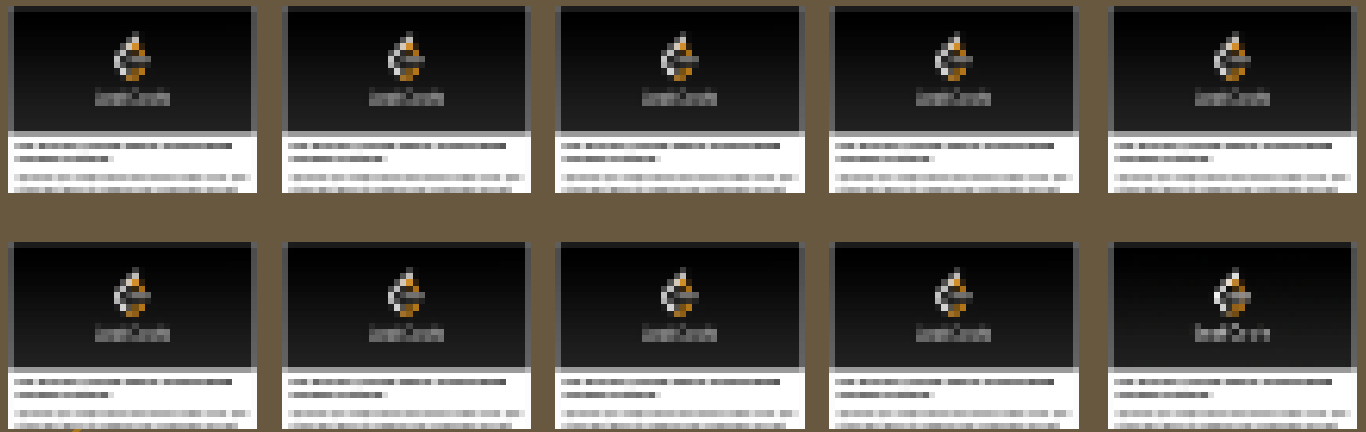
Student Records: *Schools and universities often use dictionaries to store student records, where each student has a unique identifier (key) and a set of attributes (values) such as name, address, and grades.*

Library Catalogs: *Library catalogs often use dictionaries to store book information, where each book has a unique identifier (key) and a set of attributes (values) such as author, title, and publication date.*

"Solve More Problems, Grow More Smart – Give It a try!"

LEETCODE PROBLEMS (Provided links)

DICTIONARIES



GR-TECHMIND

BUILT-IN DATA STRUCTURES IN PYTHON

📜 STRINGS SINGLE (' ') OR DOUBLE (" ") QUOTES

Strings are sequences of characters, enclosed in either single (' ') or double (" ") quotes, used to represent textual data in Python. They offer various operations for manipulation, like concatenation, slicing, and searching for substrings.

Example

```
str1 = 'Hello, '
str2 = "World!"
```

BUILT-IN DATA STRUCTURES IN PYTHON

“REAL TIME EXAMPLES“

Date and Time Display: *In a digital clock or calendar application, strings are used to display the current date and time, such as, "Friday, September 9, 2023, 14:30:00."*

Timestamps in Emails: *When you receive an email, the date and time of the email's arrival are displayed as a string, such as "Sent on September 9, 2023, 08:45 AM."*

"Solve More Problems, Grow More Smart – Give It a try!"

LEETCODE PROBLEMS (Provided links)

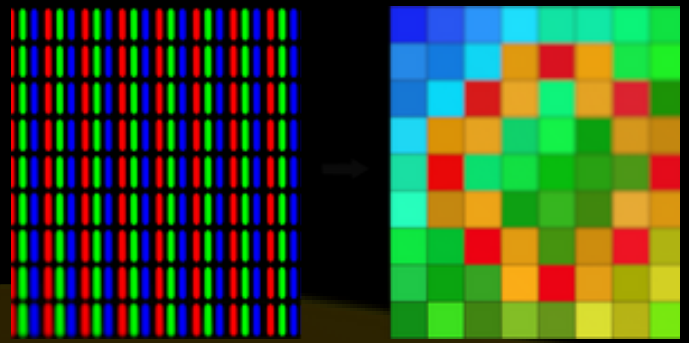
STRINGS

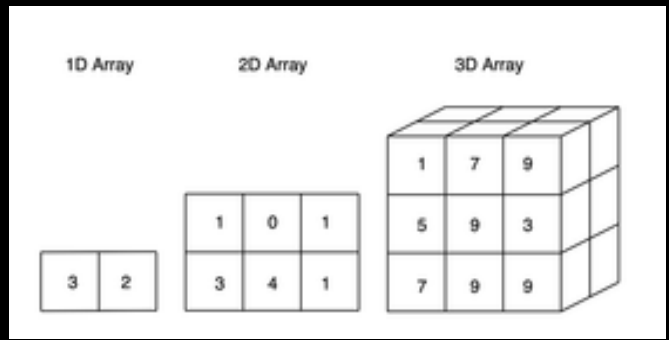
GR-TECHMIND

ARRAYS

INTRODUCTION | 1D ARRAY | 2D ARRAY | 3D ARRAY

EXAMPLES | LEETCODE PROBLEMS





GR-TECHMIND

INTRODUCTION TO ARRAYS

📊 What are arrays?

An array is a collection of elements, all of the same data type, arranged in a fixed-size sequence. Each element is accessed by its index.

Example

*Think of an array like an **ice cream** cone holder; it can hold multiple scoops (elements), and each scoop is placed in a specific slot (index).*

ARRAYS

🛠️ Array Operations

Accessing Elements: Retrieving elements at specific indices.
Slicing: Extracting a portion of the array.
Concatenation: Combining two or more arrays.
Reversing: Reversing the order of elements in an array.
Sorting: Arranging elements in ascending or descending order.
Adding Elements: Appending, inserting, or extending elements into an array.
Removing Elements: Removing elements by value or by index.
Copying Arrays: Creating a copy of an array.
Finding Length: Determining the number of elements in an array.
Checking Membership: Checking if an element is present in the array.

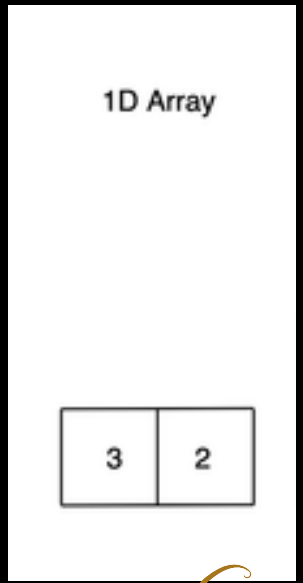
ARRAYS

🛠️ Array Methods

append(): Adds an element to the end of the array.
extend(): Appends elements from another iterable to the array.
insert(): Inserts an element at a specified position.
remove(): Removes the first occurrence of a specified element.
pop(): Removes and returns an element at a specified position.
clear(): Removes all elements from the array.
index(): Returns the index of the first occurrence of a specified element.
count(): Returns the number of occurrences of a specified element.
reverse(): Reverses the order of elements in the array.
sort(): Sorts the array in ascending or descending order.
copy(): Creates a copy of the array.
isArray(): Checks if an object is an array.

ARRAYS

1D ARRAY(One-Dimensional Array)



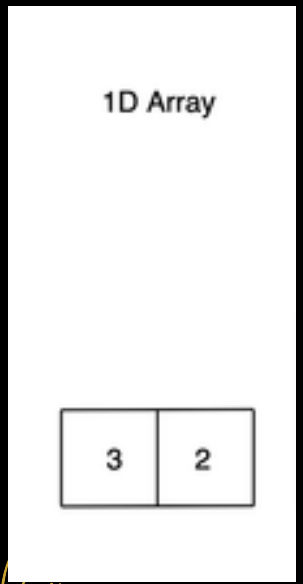
A 1D array, also known as a one-dimensional array, is a linear data structure that stores a collection of elements in a single row or a single line. Each element in a 1D array is identified by its index, starting from 0 for the first element. 1D arrays are often used to represent sequences of data, such as lists of numbers or characters.

EXAMPLE

TEMPERATURES = [72, 74, 75, 76, 78, 79, 80]

LEETCODE PROBLEMS

1D ARRAY(One-Dimensional Array)



TWO SUM (PROBLEM #1)
https://leetcode.com/problems/two-sum/

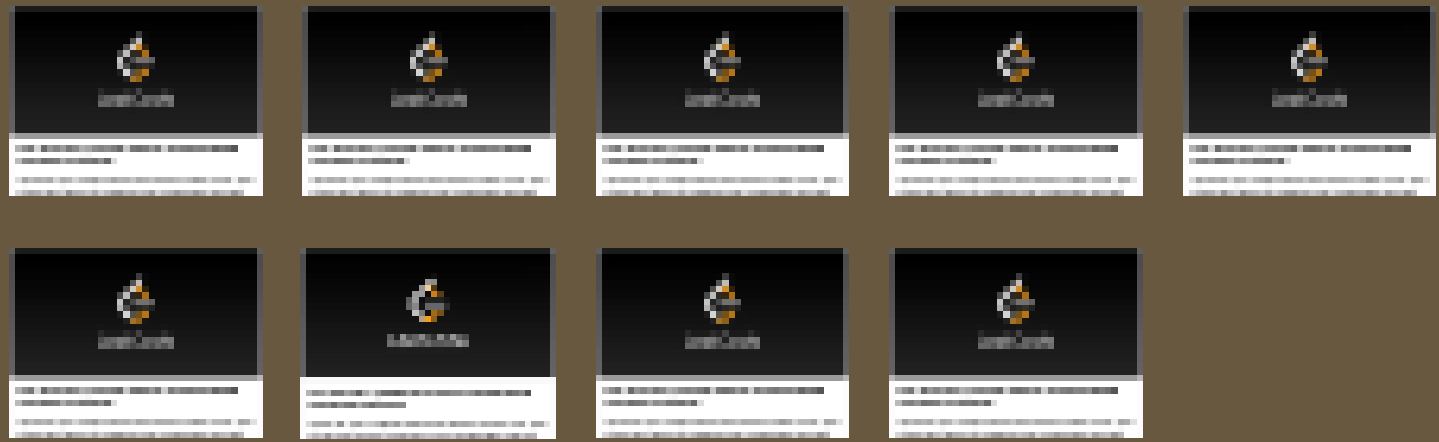
BEST TIME TO BUY AND SELL STOCK (PROBLEM #121)
https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

FIND ALL NUMBERS DISAPPEARED IN AN ARRAY (PROBLEM #448)
https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/

"Solve More Problems, Grow More Smart – Give It a try!"

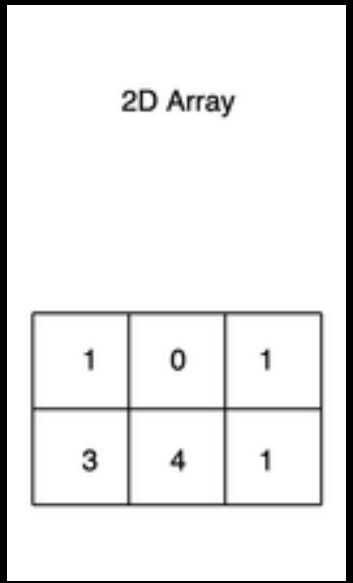
LEETCODE PROBLEMS (Provided links)

1D ARRAY(One-Dimensional Array)



GR-TECHMIND

ARRAYS



2D ARRAY(Two-Dimensional Array)

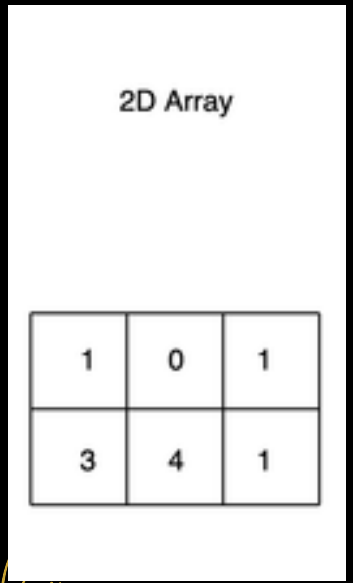
A 2D array, also known as a two-dimensional array, is a data structure that stores data in a grid or table format. It consists of rows and columns, where each element is identified by its row and column index. 2D arrays are often used to represent matrices, tables, or grids of data.

EXAMPLE

```
TIC_TAC_TOE_BOARD = [ ['X', 'O', 'X'],
                      ['O', 'X', 'O'],
                      ['X', 'X', 'O'] ]
```

LEETCODE PROBLEMS

2D ARRAY(Two-Dimensional Array)



Matrix Diagonal Sum (Problem #1572)
Link to problem

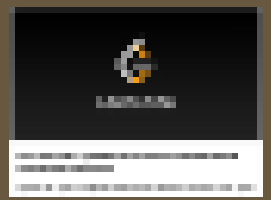
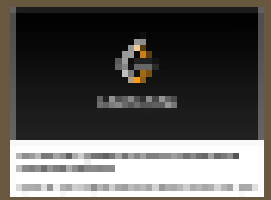
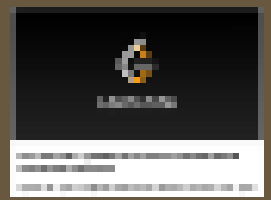
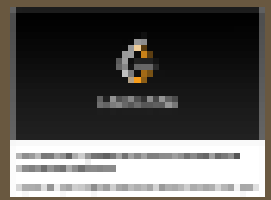
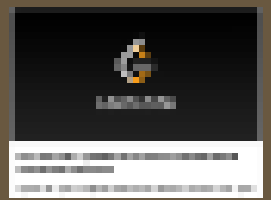
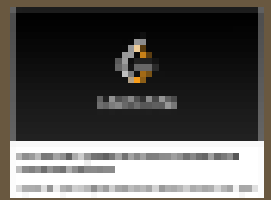
Image Rotation (Problem #48)
Link to problem

Spiral Matrix (Problem #54)
Link to problem

"Solve More Problems, Grow More Smart – Give It a try!"

LEETCODE PROBLEMS (Provided links)

2D ARRAY(Two-Dimensional Array)

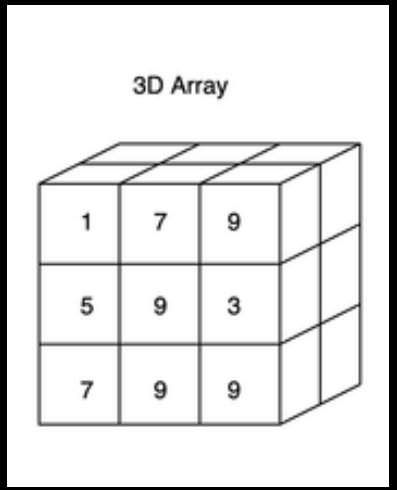







1446 ARRAY PROBLEMS ON LEETCODE 👇

GR-TECHMIND

ARRAYS

3D ARRAY(Three-Dimensional Array)



A 3D array, also known as a three-dimensional array, is an extension of the 2D array and is used to store data in a cube-like structure with three dimensions: rows, columns, and depth. It's a collection of 2D arrays stacked on top of each other. Each element in a 3D array is identified by its row, column, and depth index.

EXAMPLE

```
CUBE = [
  [ # LAYER 1
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
  ],
  [ # LAYER 2
    [10, 11, 12],
    [13, 14, 15],
    [16, 17, 18]
  ],
  [ # LAYER 3
    [19, 20, 21],
    [22, 23, 24],
    [25, 26, 27]
  ]
]
```

GR-TECHMIND

TASK TIME !!

FIND OUT SOME EXAMPLES

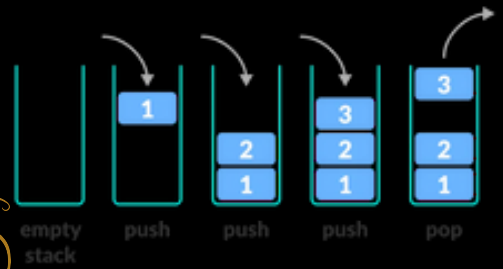
ON 3D ARRAYS

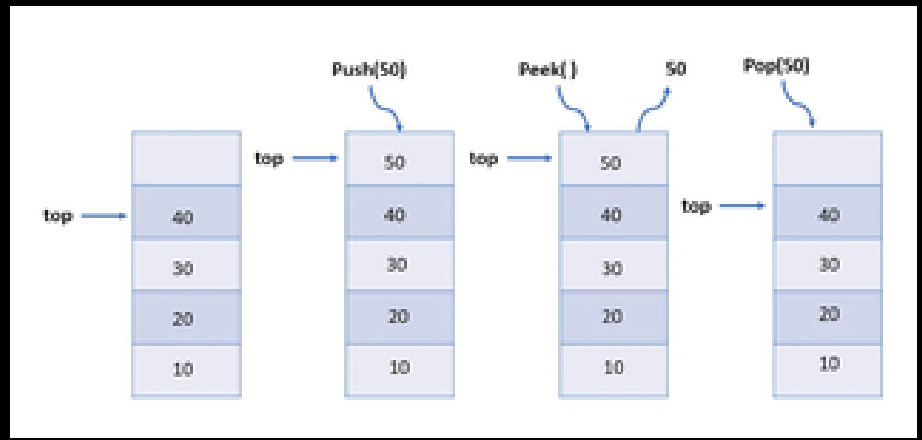
GR-TECHMIND

STACKS

INTRODUCTION | OPERATIONS | METHODS

EXAMPLES | LEETCODE PROBLEMS





GR-TECHMIND

INTRODUCTION TO STACKS

📚 What are Stacks? Last-In-First-Out (LIFO)

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. In a stack, elements are added and removed from the same end, known as the "top" of the stack. The last element added to the stack is the first one to be removed. Stacks are used for managing data in a way that allows efficient adding and removing of elements, making them useful in various computer science and real-world applications.

Example

Text Editors' Undo Functionality, Undo in Graphics Software, Backtracking Algorithms, Maze Solving Algorithms

STACKS

🛠️ Stack Operations

Push ( ): Add an element to the top of the stack.

Pop ( ): Remove and return the element at the top of the stack.

Peek (or Top): Retrieve the element at the top of the stack without removing it.

IsEmpty ( ): Check if the stack is empty. Returns a boolean indicating whether the stack has any elements.

Size (or Length): Get the number of elements in the stack.

STACKS

🛠️ Stack Methods

push(item): Adds an item to the top of the stack.

pop(): Removes and returns the item at the top of the stack.

peek(): Returns the item at the top of the stack without removing it.

is_empty(): Checks if the stack is empty and returns True if it is, or False if it's not.

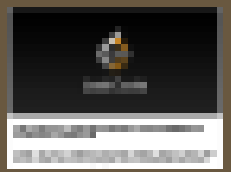
size(): Returns the number of elements currently in the stack.

clear(): Removes all elements from the stack, making it empty.

"Solve More Problems, Grow More Smart – Give It a try!"

LEETCODE PROBLEMS (Provided links)

STACKS

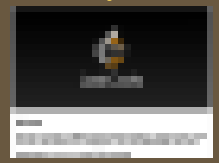









147 STACK PROBLEMS ON LEETCODE 👇

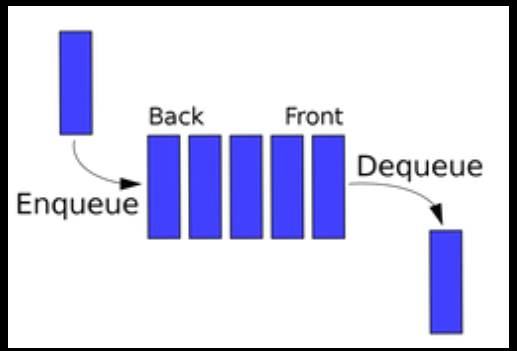


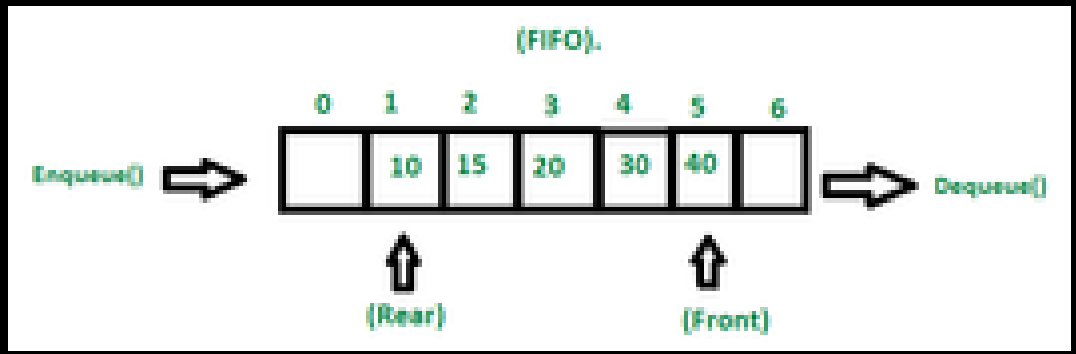
GR-TECHMIND

QUEUES

INTRODUCTION | OPERATIONS | METHODS

EXAMPLES | LEETCODE PROBLEMS





GR-TECHMIND

INTRODUCTION TO QUEUES

📚 What are Queues? First-In-First-Out (FIFO)

A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. In a queue, elements are added at one end called the "rear," and they are removed from the other end called the "front." The element that has been in the queue the longest is the first one to be removed. Queues are used for various applications where tasks or data need to be processed in a specific order.

Example

Call Center Systems: *Incoming customer support calls are placed in a queue and answered by available agents in the order they were received.*

QUEUES

🛠️ Queue Operations

Enqueue (or Push): Add an element to the rear (end) of the queue.

Dequeue (or Pop): Remove and return the element from the front of the queue.

Front (or Peek): Retrieve the element at the front of the queue without removing it.

IsEmpty: Check if the queue is empty. Returns a boolean indicating whether the queue has any elements.

Size (or Length): Get the number of elements in the queue.

QUEUES

🛠️ Queue Methods

enqueue(item): Adds an item to the rear (end) of the queue.

dequeue(): Removes and returns the item from the front of the queue.

front(): Returns the item at the front of the queue without removing it.

is_empty(): Checks if the queue is empty and returns True if it is, or False if it's not.

size(): Returns the number of elements currently in the queue.

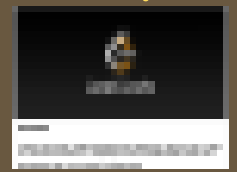
clear(): Removes all elements from the queue, making it empty.

"Solve More Problems, Grow More Smart – Give It a try!"

LEETCODE PROBLEMS (Provided links)

QUEUES

40 QUEUE PROBLEMS ON LEETCODE 👇

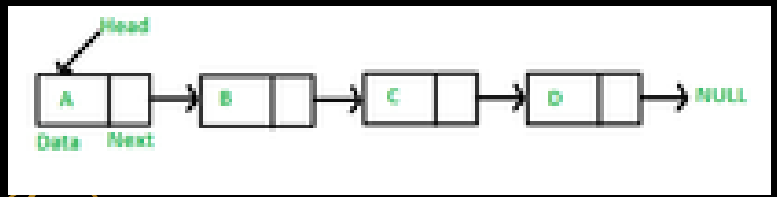


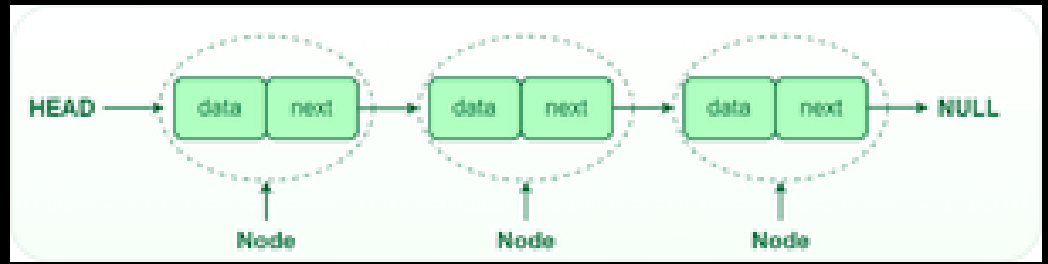
GR-TECHMIND

LINKED LISTS

INTRODUCTION | TYPES | OPERATIONS | METHODS

EXAMPLES | LEETCODE PROBLEMS





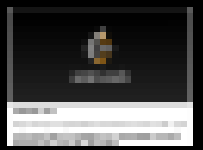
GR-TECHMIND

INTRODUCTION TO LINKED LISTS

📚 What are Linked Lists?

Linked lists are fundamental data structures in computer science and programming. They provide a dynamic way to organize and store data, especially when the number of elements is not fixed or known in advance. Linked lists consist of a series of nodes, where each node holds both data and a reference (or link) to the next node in the sequence.

72 LINKED LISTS PROBLEMS ON LEETCODE 👇



GR-TECHMIND

LINKED LISTS

🛠️ Linked Lists Operations and Methods

1. Insertion:
Insert at the Beginning
Insert at the End
Insert at a Specific Position

2. Deletion:
Delete from the Beginning
Delete from the End
Delete at a Specific Position
Delete by Value

3. Traversal:
Iterative Traversal

4. Searching:
Search by Value

5. Modification:
Update Node Value

6. Length and Empty Check:
Get Length

7. Reversal:
Reverse Linked List
Concatenate Linked Lists

9. Splitting:
Split Linked List

10. Merge and Sort:
Merge Linked Lists
Sort Linked List

11. Cycle Detection:
Detect Cycles

GR-TECHMIND

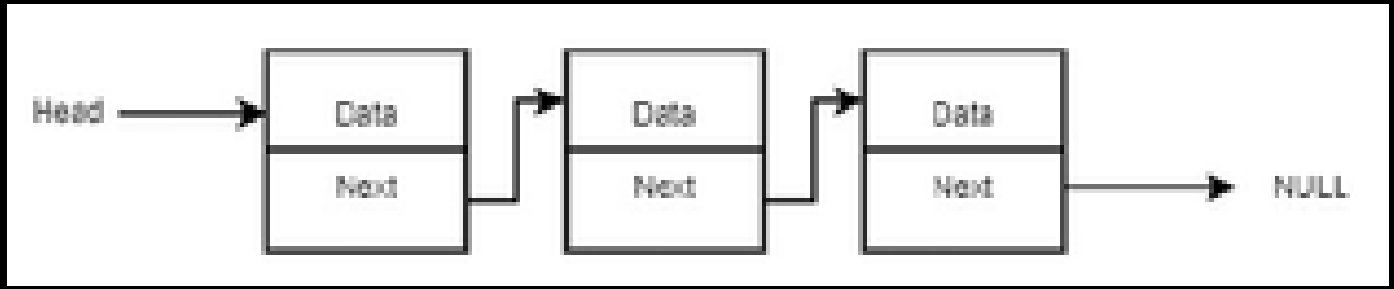
TYPES OF LINKED LISTS

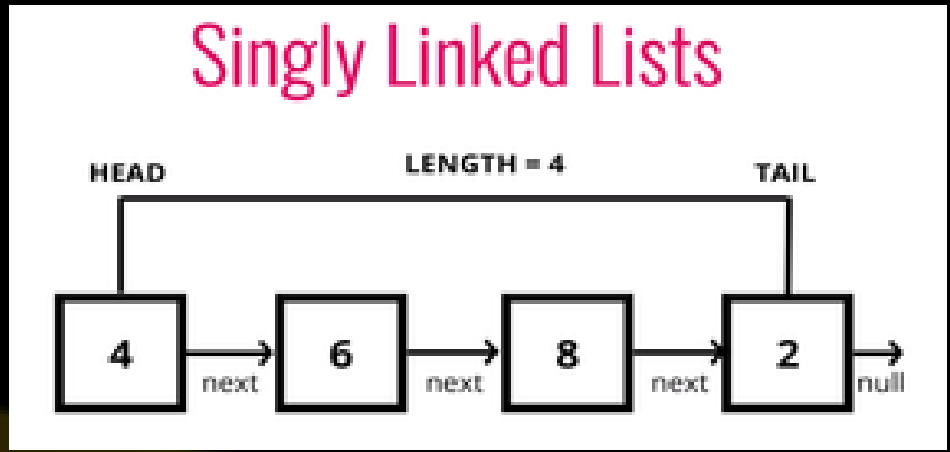
Singly Linked Lists:
A singly linked list is a linear data structure where each element (node) contains a data element and a reference (or link) to the next node in the sequence.
Example: Think of a singly linked list as a chain of train cars. Each car (node) holds cargo (data) and is connected to the next car, forming a train.

Doubly Linked Lists:
A doubly linked list is similar to a singly linked list, but each node contains references to both the next and the previous nodes.
Example: Imagine a two-way street. In a doubly linked list, you can move forward (next) and backward (previous) between nodes like navigating a street in both directions.

Circular Linked Lists:
A circular linked list is a variation where the last node points back to the first node, creating a closed loop.
Example: Think of a circular linked list as a circular race track. You can keep running in circles, and there's no end or beginning.

LINKED LISTS - SINGLY



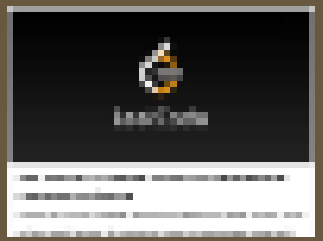
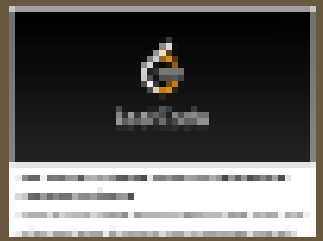
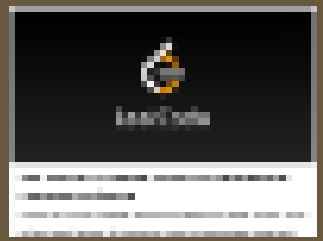


GR-TECHMIND

"Solve More Problems, Grow More Smart – Give It a try!"

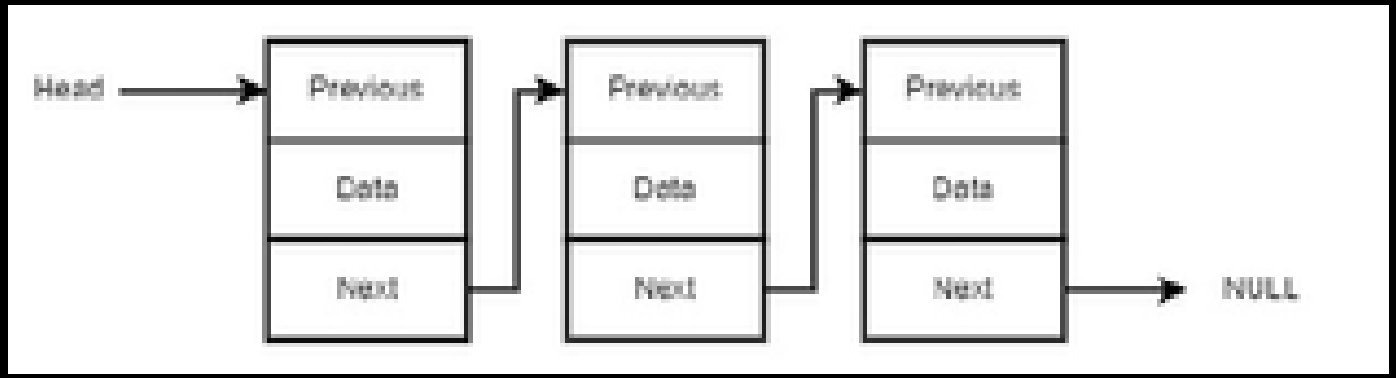
LINKED LISTS - SINGLY

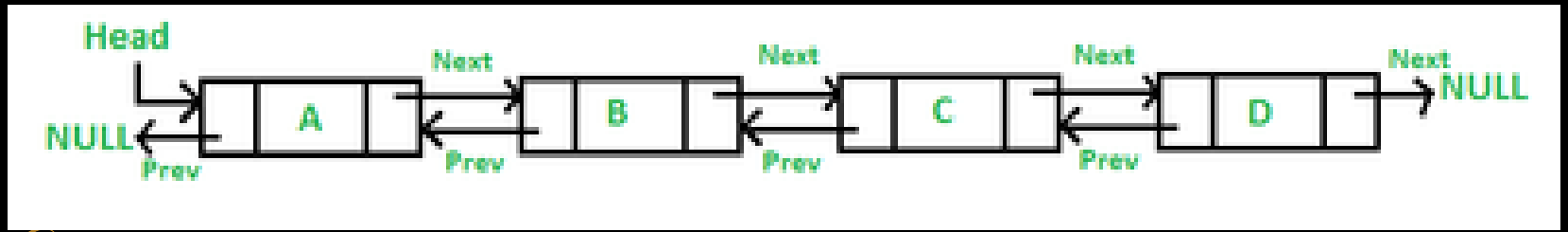
LEETCODE PROBLEMS (Provided links)

GR-TECHMIND

LINKED LISTS - DOUBLY

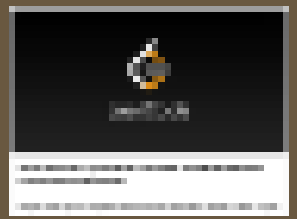
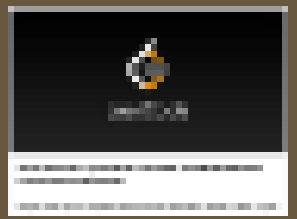
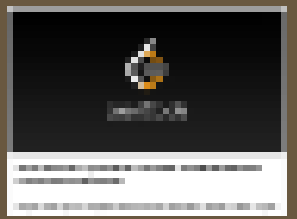
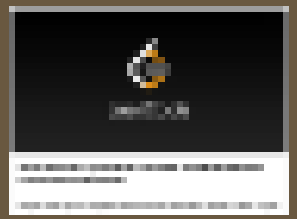
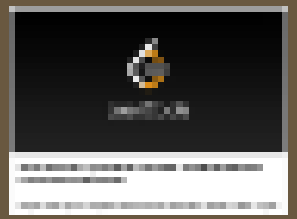




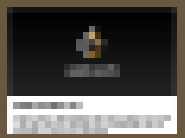
"Solve More Problems, Grow More Smart – Give It a try!"

LINKED LISTS - DOUBLY

LEETCODE PROBLEMS (Provided links)

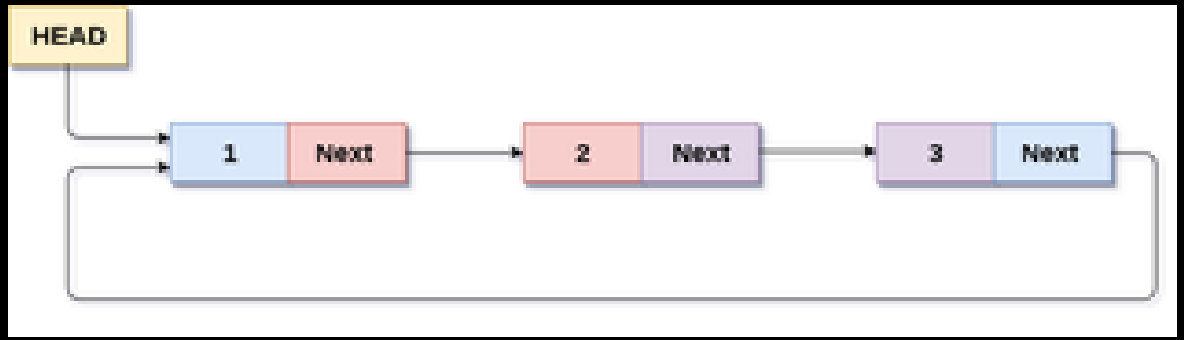






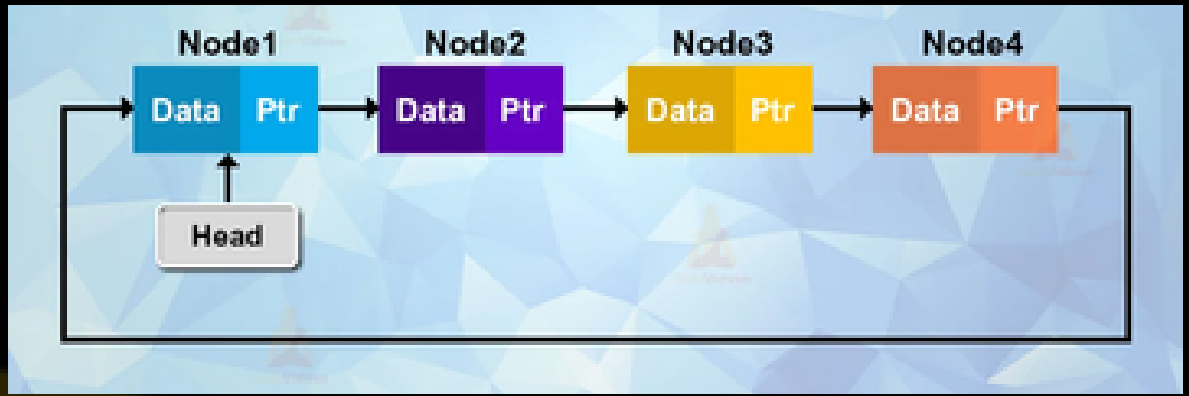
8 DOUBLY LINKED LISTS PROBLEMS ON LEETCODE 👇



GR-TECHMIND

LINKED LISTS - CIRCULAR

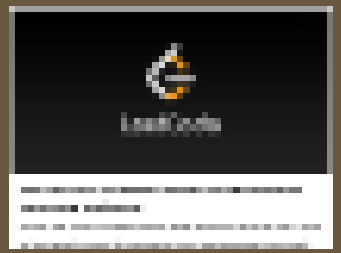


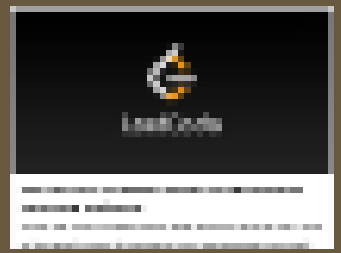


"Solve More Problems, Grow More Smart — Give It a try!"

LINKED LISTS - CIRCULAR

LEETCODE PROBLEMS (Provided links)



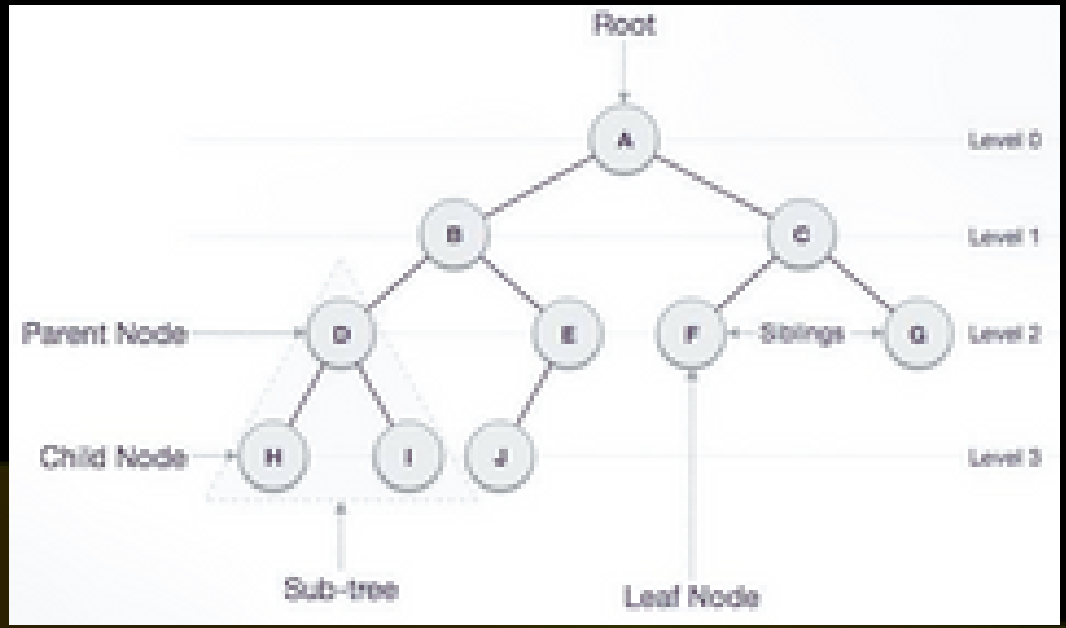


GR-TECHMIND

TREES

INTRODUCTION | TYPES | OPERATIONS | METHODS

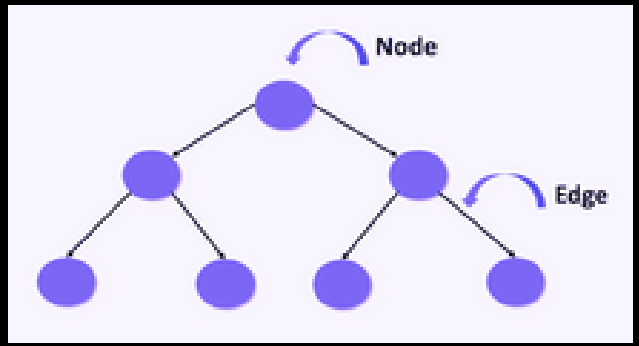
EXAMPLES | LEETCODE PROBLEMS



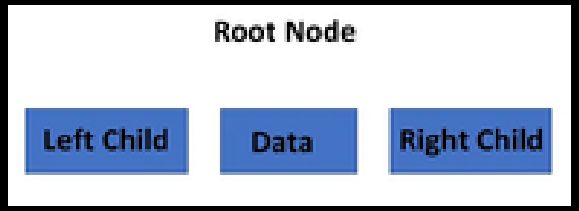
GR-TECHMIND

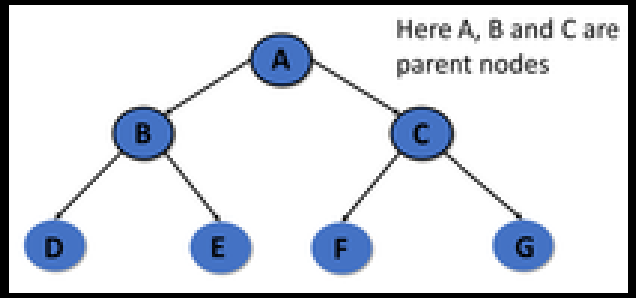
INTRODUCTION TO TREES

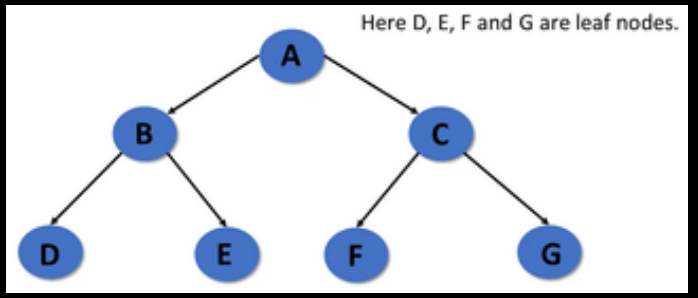
What are Trees?

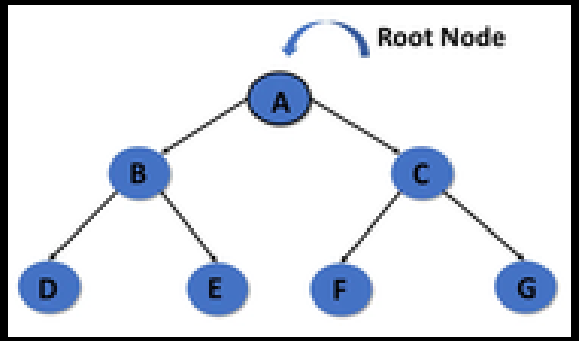


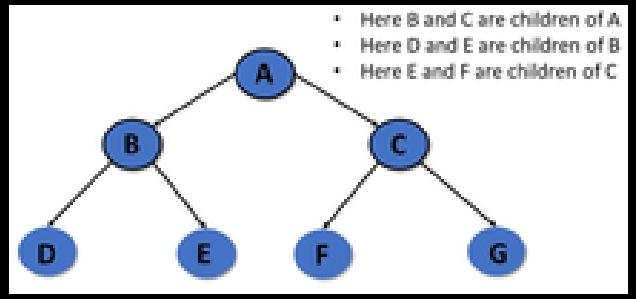
A tree is a hierarchical data structure that stores data in a hierarchical manner.
It comprises a collection of entities known as nodes, which are connected using "edges", both directed and undirected.
Each node in a tree can have at most one parent, except for the root node, which has no parent.

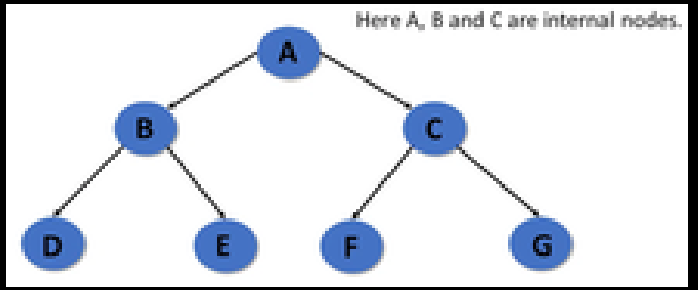

Root Node
Left Child
Data
Right Child

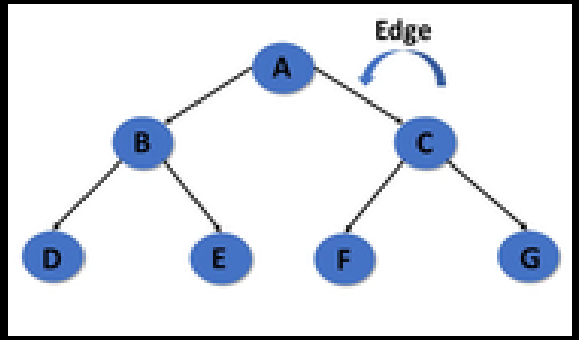

Here A, B and C are parent nodes
A
B
C
D
E
F
G

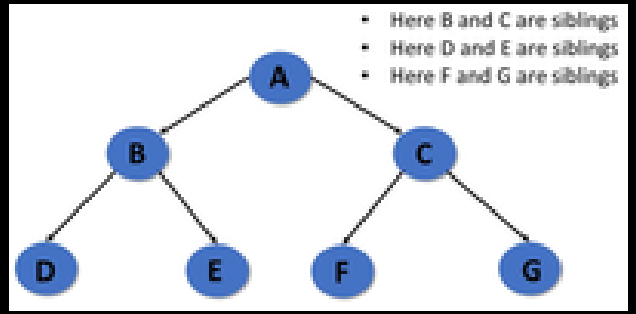

Here D, E, F and G are leaf nodes.
A
B
C
D
E
F
G

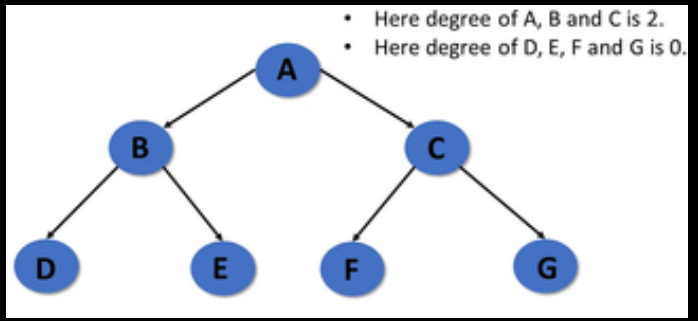

Root Node
A
B
C
D
E
F
G


Here B and C are children of A
Here D and E are children of B
Here E and F are children of C
A
B
C
D
E
F
G

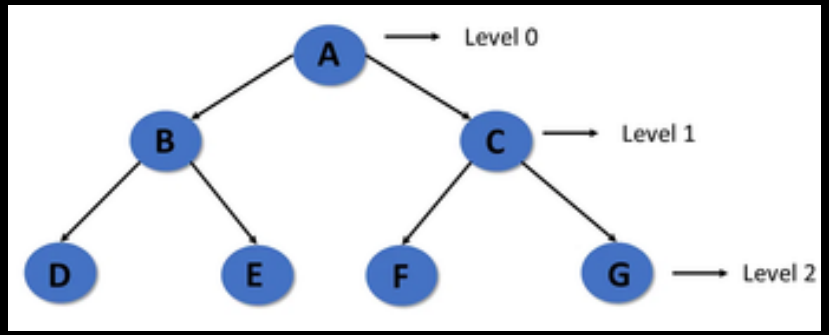

Here A, B and C are internal nodes.
A
B
C
D
E
F
G

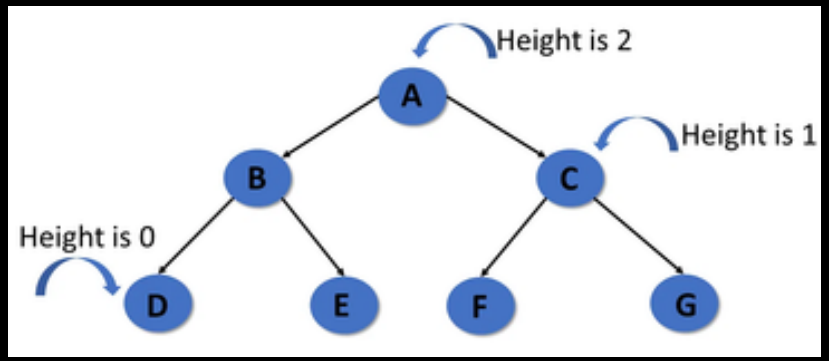

Edge
A
B
C
D
E
F
G

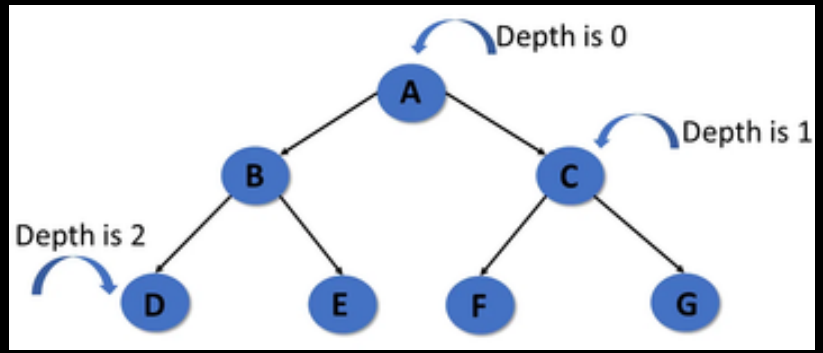

Here B and C are siblings
Here D and E are siblings
Here F and G are siblings
A
B
C
D
E
F
G

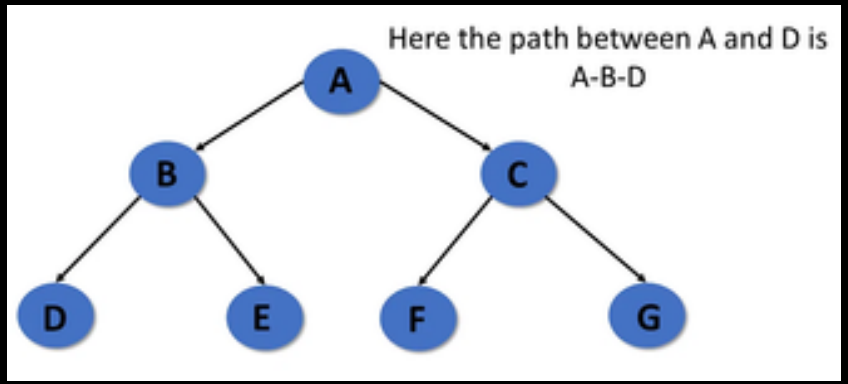

Here degree of A, B and C is 2.
Here degree of D, E, F and G is 0.
A
B
C
D
E
F
G

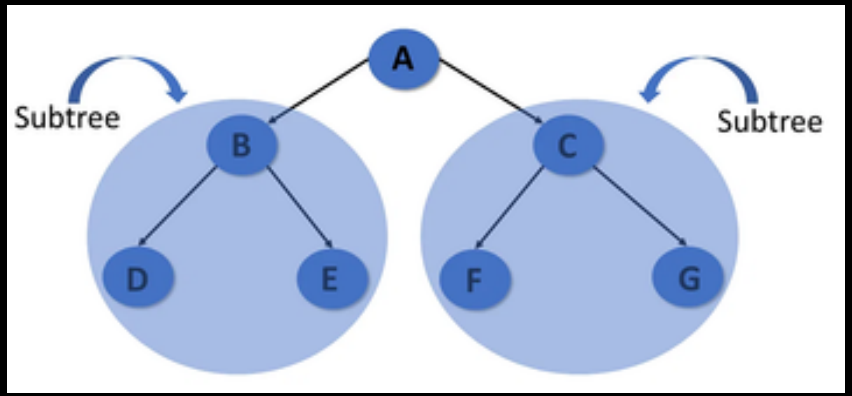
GR-TECHMIND


A
Level 0
B
C
Level 1
D
E
F
G
Level 2


Height is 2
A
Height is 1
B
C
Height is 0
D
E
F
G


Depth is 0
A
Depth is 1
B
C
Depth is 2
D
E
F
G

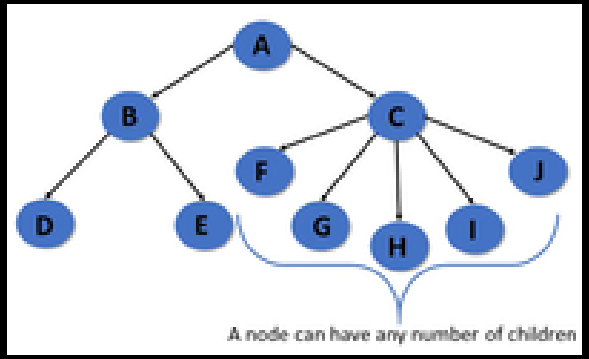

Here the path between A and D is
A-B-D
A
B
C
D
E
F
G


A
Subtree
Subtree
B
C
D
E
F
G

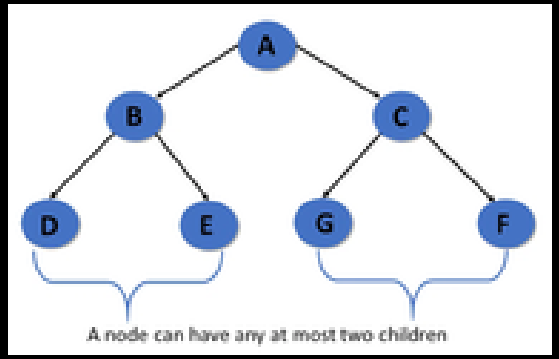
GR-TECHMIND

TYPES OF TREES

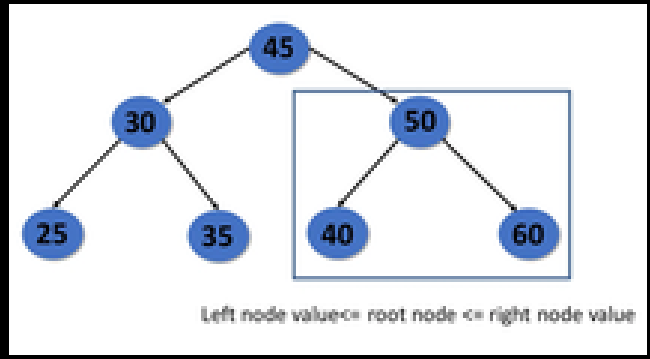
GENERAL TREE



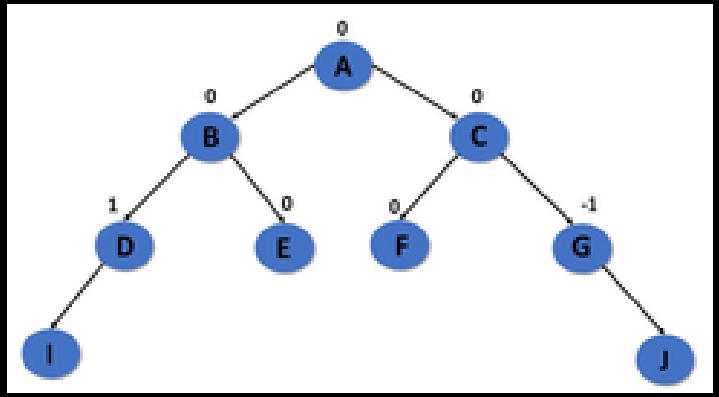
BINARY TREE



BINARY SEARCH TREE



AVL TREE

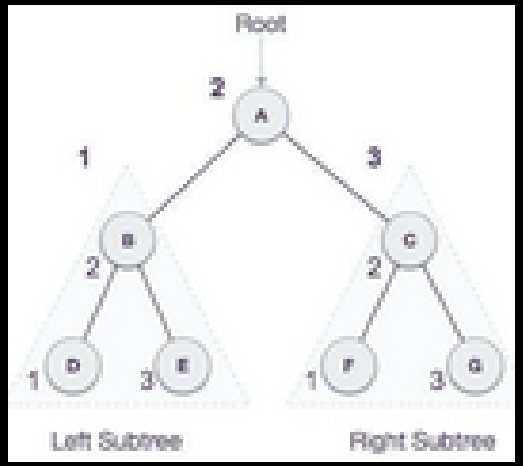


GR-TECHMIND

TYPES OF TREES

TREE TRAVERSAL

IN-ORDER TREE TRAVERSAL



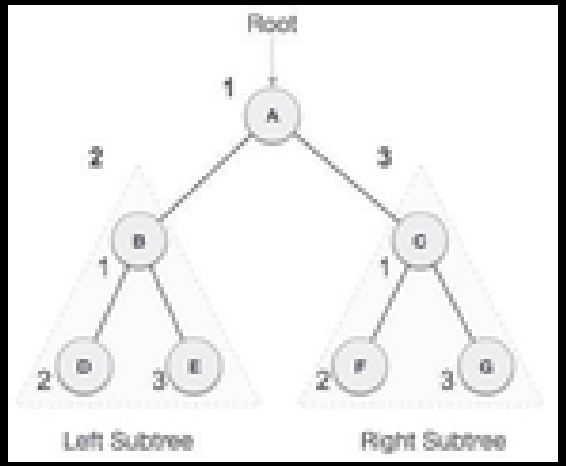
D → B → E → A → F → C → G

Step 1 – Recursively traverse left subtree.

Step 2 – Visit root node.

Step 3 – Recursively traverse right subtree.

PRE-ORDER TREE TRAVERSAL



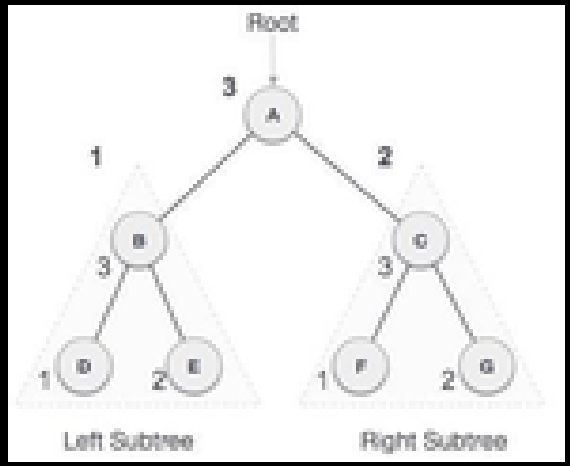
A → B → D → E → C → F → G

Step 1 – Visit root node.

Step 2 – Recursively traverse left subtree.

Step 3 – Recursively traverse right subtree.

POST-ORDER TREE TRAVERSAL



D → E → B → F → G → C → A

Step 1 – Recursively traverse left subtree.

Step 2 – Recursively traverse right subtree.

Step 3 – Visit root node.

GR-TECHMIND

TASK TIME !!

FIND OUT SOME MORE TYPES

ON TREES

GR-TECHMIND

OVERVIEW

ON TREES

Root: 🌱 *The topmost node in a tree, from which all other nodes descend. It serves as the starting point for traversing the tree.*

Node: 🫛 *A fundamental building block of a tree, representing a single entity within the tree. Each node contains data and can have zero or more child nodes.*

Child: 🌿 *A node directly connected to another node when moving away from the root. Each node can have multiple children, depending on the tree type.*

Parent: 🌳 *A node that has one or more child nodes. It is situated higher in the hierarchy.*

Leaf: 🍂 *A node that has no children. It's situated at the end of a branch.*

Siblings: 👯 *Nodes that share the same parent node.*

Depth: 📏 *The level of a node within a tree. The root node is at depth 0, its children are at depth 1, and so on.*

Height: 📐 *The maximum depth of a tree, i.e., the longest path from the root to a leaf node.*

Subtree: 🎄 *A tree formed by selecting a node and all its descendants, excluding the rest of the nodes in the original tree.*

Binary Tree: 0️⃣🌳 *A tree in which each node has at most two children, referred to as the left child and right child.*

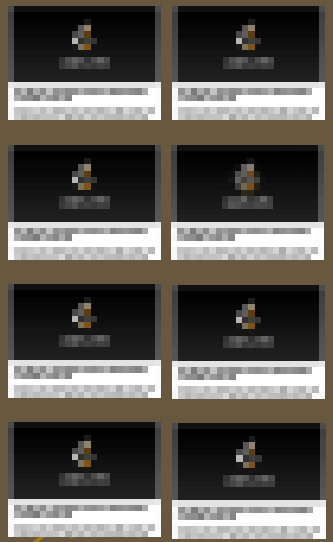
Binary Search Tree (BST): 🔍🌳 *A binary tree in which the left child contains values less than the node, and the right child contains values greater than the node. This property allows for efficient searching.*

Balanced Tree: ⚖️🌳 *A tree in which the height of the left and right subtrees of every node differ by at most one.*

"Solve More Problems, Grow More Smart – Give It a try!"

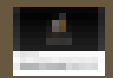
TREES | TYPES OF TREES

LEETCODE PROBLEMS (Provided links)

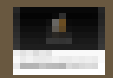


217 TREES PROBLEMS ON LEETCODE

170 BINARY TREES PROBLEMS ON LEETCODE



40 BINARY SEARCH TREES PROBLEMS ON LEETCODE



29 BINARY INDEXED TREES PROBLEMS ON LEETCODE

35 SEGMENT TREES PROBLEMS ON LEETCODE

5 MINIMUM SPANNING TREES PROBLEMS ON LEETCODE

220 BFS PROBLEMS ON LEETCODE

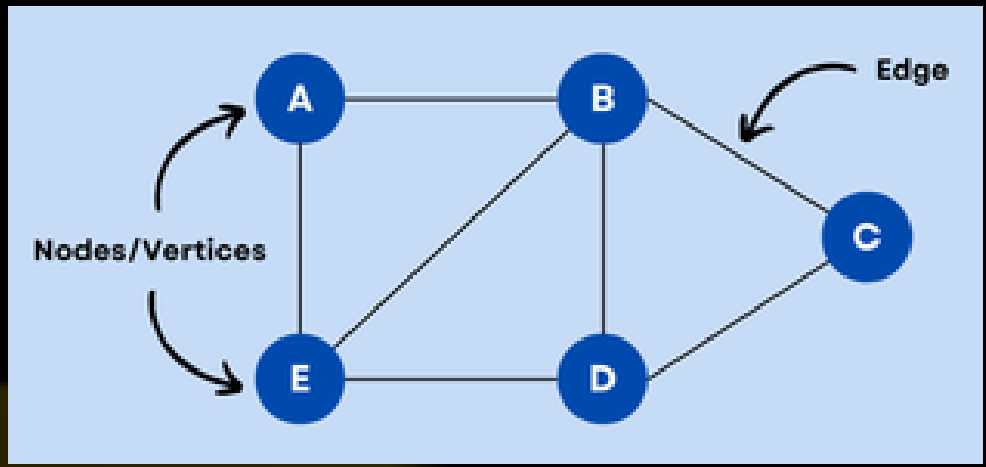
278 DFS PROBLEMS ON LEETCODE

GR-TECHMIND

GRAPHS

INTRODUCTION | TYPES | OPERATIONS | METHODS

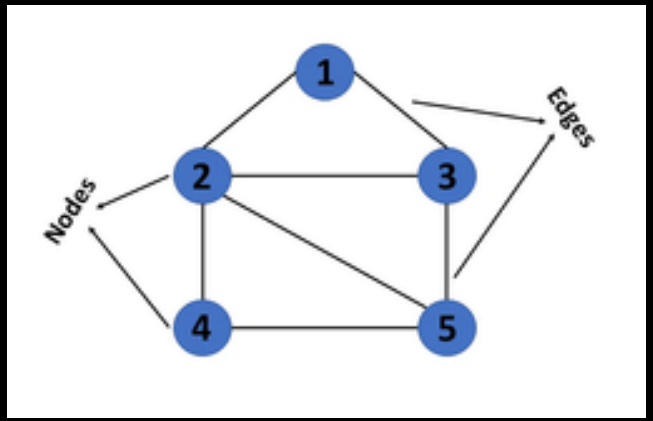
EXAMPLES | LEETCODE PROBLEMS



GR-TECHMIND

INTRODUCTION TO TREES

What are Graphs?



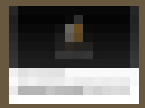
- Graphs are versatile data structures used to represent relationships and connections between entities.
- They consist of nodes (vertices) and edges (connections) that link these nodes.
- Graphs can be directed (edges have a specific direction) or undirected (edges have no direction).

"Solve More Problems, Grow More Smart – Give It a try!"

GRAPHS

LEETCODE PROBLEMS (Provided links)

220 BFS PROBLEMS ON LEETCODE 

278 DFS PROBLEMS ON LEETCODE 

GR-TECHMIND

Thank you!!

Hope you liked it.

gangavarapuramya15@gmail.com