Data Structures

Goh Shan Lun & Cosmi Wai Hang

TOC

1. Introduction

Overview and Objectives

Overview of Data Structures

- Used to **organise and store data** in programs and systems
- **Arrange data** on a computer so that it can be **accessed and updated efficiently**
- **Provide methods for searching, sorting, and processing data efficiently**
- Help break down complex problems into manageable components
- Using the appropriate data structure for a program can **enhance its performance**

Objectives of the Workshop

- Learn **concepts** of **various types of data structures** like
  - Arrays
  - Linked List
  - Stack
  - Queue
  - Trees
  - Graphs
  - Hash Tables and Maps

- Learn how to **implement** some data structures from **scratch** and using the **Standard Template Library (STL)**

2. Data Structure (DS)

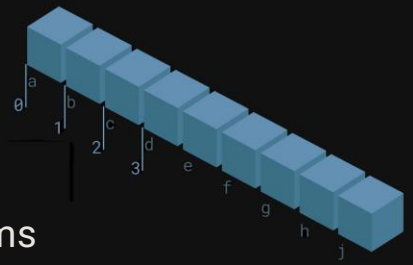
Array, Linked List, Stack and Queue

Array

1D array vs ND array

1D Array

- A one-dimensional array used to store a collection of elements of the same data type in a contiguous block of memory
- Can only store items of the **same type**
- Stored in contiguous blocks of memory
- **Fixed size**, **very fast** search, **slow** addition and deletion of items



1D Array Demo

ND Array

- A multi-dimensional array which provides a convenient and efficient way to store and manipulate multi-dimensional data

- For example
  - 2D Array: Collection of 1D arrays
  - 3D Array: Collection of 2D arrays

- Common Use Cases:
  - Scientific Computing
  - Image Processing
  - Machine Learning

ND Array Demo

1D Array vs ND Array

1D Array	ND Array
Storing and manipulating one-dimensional data	Storing and manipulating multi-dimensional data
Takes up lesser memory	Takes up more memory
Limited to one-dimensional data	Convenient and efficient for multi-dimensional data
	Can take advantage of vectorization and parallel processing

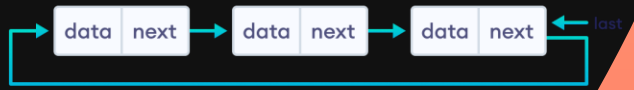
Linked List

Singly Linked List, Double Linked List etc.

Linked List

- A chain of nodes
- Every node has a reference to the next node
- Not stored in contiguous memory which allows it to have dynamic sizing
- More efficient for large or dynamic lists

- Some Common Types Of Linked List:
  - Singly Linked List: the basic linked list
  - Doubly Linked List: every node has a pointer to the previous node too!
  - Circular Linked List: a circular linked list



Singly Linked List

- Every node contains a pointer to the next node in the sequence
- Can only traverse in one direction from head to tail (left -> right)
- Head: First Node
- Tail: Last Node

- Adding Node:
  - Update reference in new node to point to next node
  - Update the previous node to point to the new node
- Removing Node:
  - Need to update the reference in the previous node to point to the next node
  - Free memory associated with the node to be removed

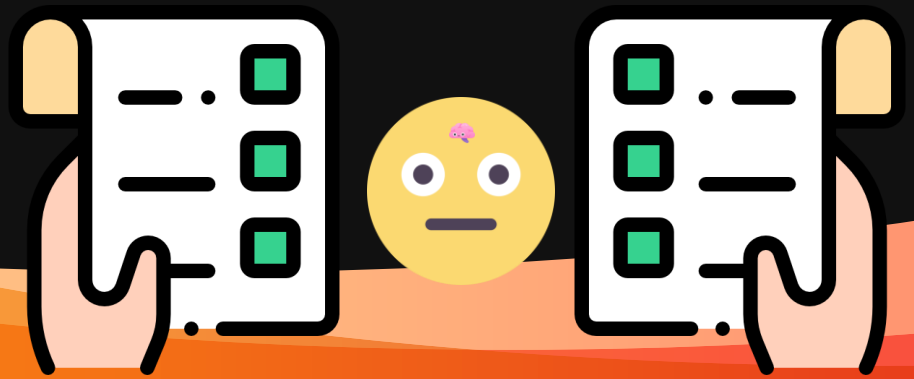
Singly Linked List Demo

Doubly Linked List

- Every node contains pointers to the previous node and the next node in the sequence
- The list can be traversed in both directions

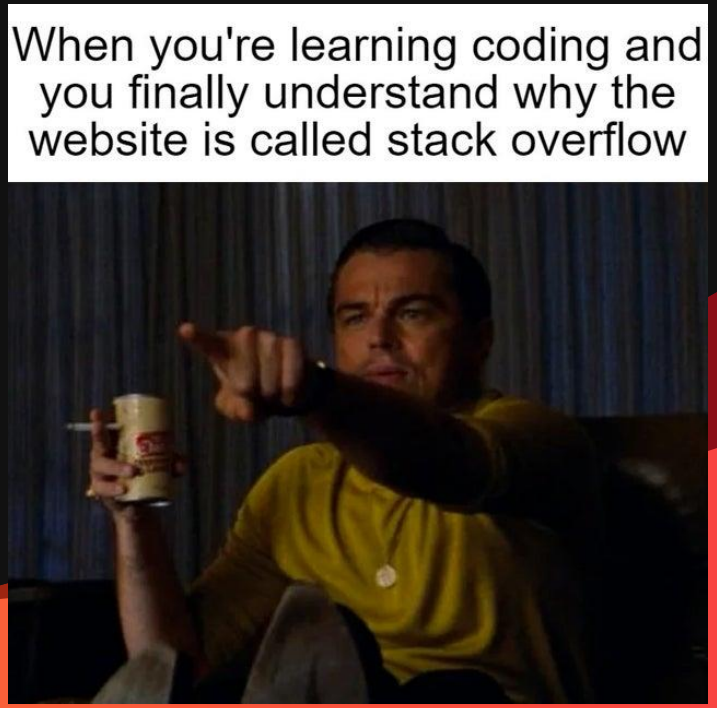
- Adding Node:
  - Update the references in the previous and the next nodes to point to the new node
  - Update the references in the new node to point to the previous and next nodes
- Removing Node:
  - Update the references in the previous and the next nodes to point to each other
  - Free the memory of the deleted node

Doubly Linked List Demo

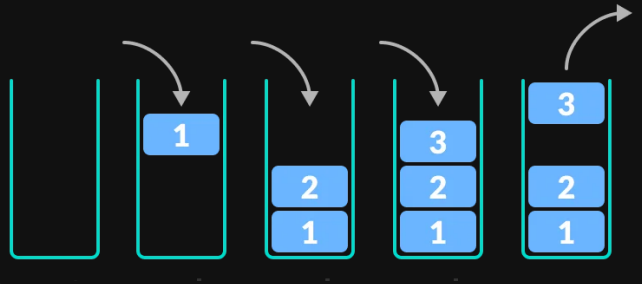


Stack

Stack Abstract Data Type (ADT)



Stack

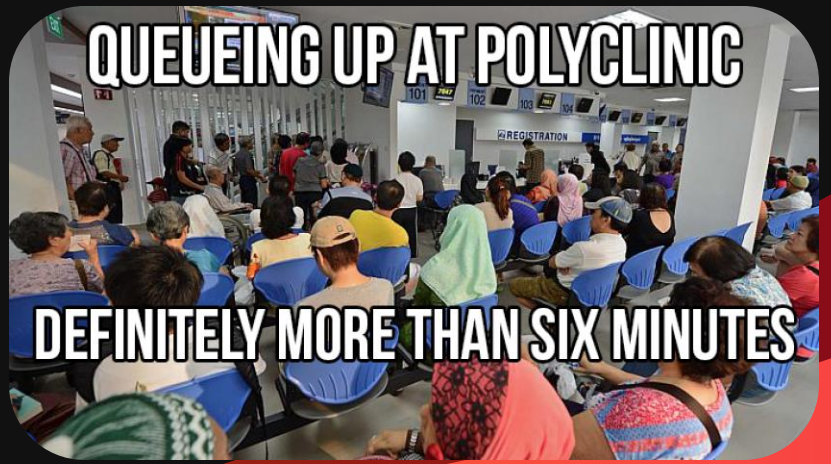


- It follows the Last-In-First-Out (LIFO) principle
- Array Implementation:
  - The stack is a fixed-size collection of elements
  - Push/Pop: Update an index variable that points to top of stack
- Linked List Implementation:
  - The stack is a dynamic collection of elements
  - Push (add): Adding nodes to head of list
  - Pop (remove): Remove nodes from head of list
- Other useful functions:
  - Peek: Check what is at the top of the stack without removing
  - isEmpty: Check if stack is empty
  - And Many More……

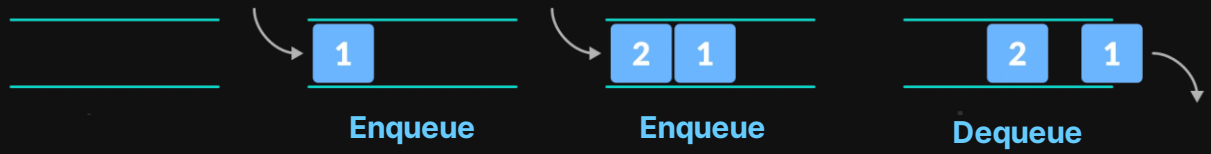
Stack Demo

Queue

Queue Abstract Data Type (ADT)



Queue



- It follows the First-In-First-Out (FIFO) Principle
- Array Implementation:
  - The queue is a fixed-size collection of elements
  - Enqueue/Dequeue: Update two index variable that to the front and back of the queue respectively
- Linked List Implementation:
  - The queue is a dynamic collection of elements
  - Enqueue: The back pointer is updated to point to the new element
    (If empty: front pointer is also pointed to the new element)
  - Dequeue: The front pointer is updated to point to the next element in the queue
    (If empty: set both front and back pointers to null)

Queue Demo

3. Advanced DS

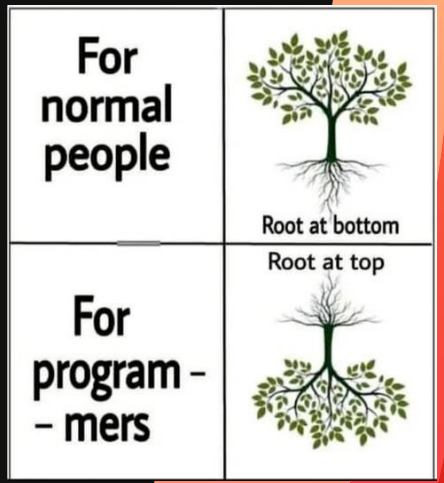
Tree, Graph, Hash Table and Map

Trees

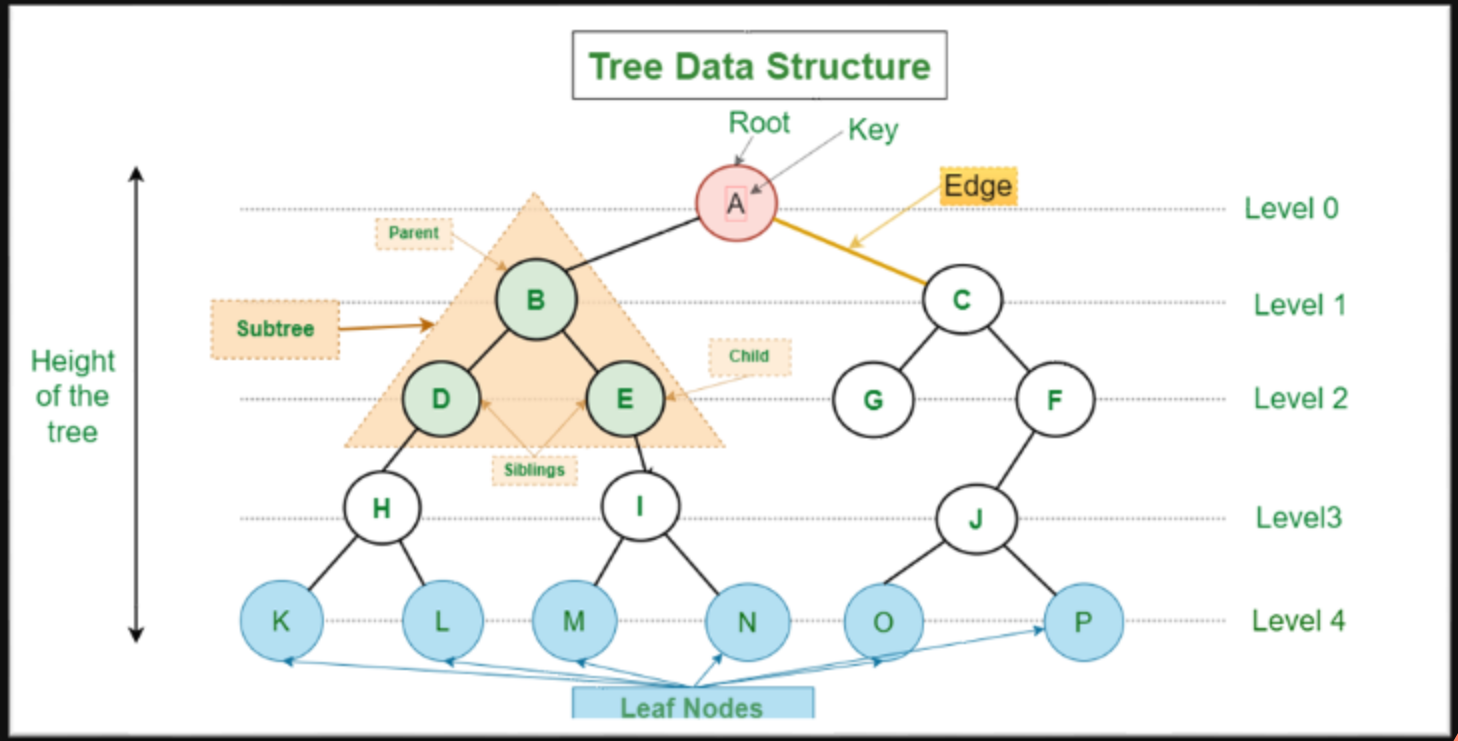
Binary Tree, AVL Tree, B-Tree, B+ Tree etc.

Tree

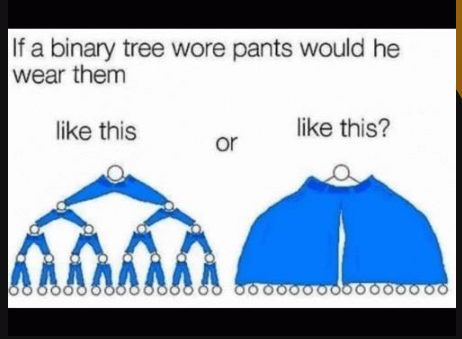
- Hierarchical collection of nodes (vertices) and edges (links) that connect these nodes.
- Used to organize data in a way that is easy to navigate and search
  - **Trie**: Used for dynamic spell checking
  - **Binary Search Tree**
  - **Artificial Intelligence**: In the form of a decision tree



Structure of a Tree



Types of trees



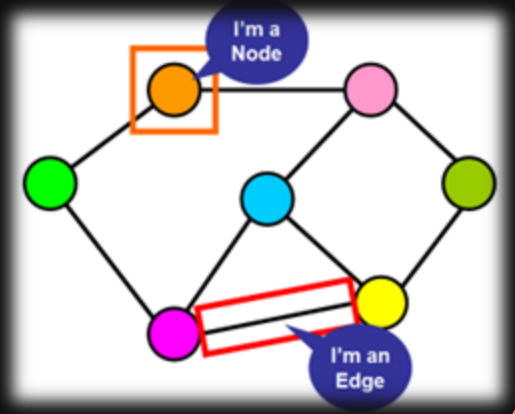
- There are 4 main types of trees
  - **General**: No restriction on the number of nodes; parent node can have any number of child nodes.
  - **Binary**: Every node can have a maximum of 2 child nodes
  - **Balanced**: Height of left sub-tree and right sub-tree do not differ by more than 1
  - **Binary Search**: Rules of Binary trees apply, additionally, the left node value must be less than its parent and the right node value must be greater than its parent

Graph

Directed/Undirected graph, Weighted/Unweighted graph

Definition of Graph

- Collection of nodes (vertices) and edges (links) that connect these nodes.
- Can be used to model complex relationships between entities:
  - Social Networks
  - Road Networks
  - Computer Networks

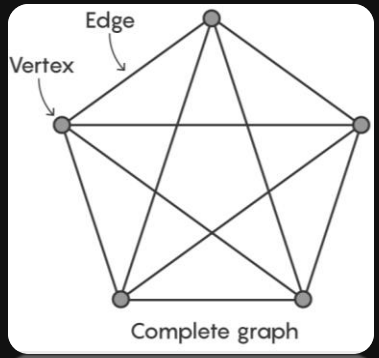


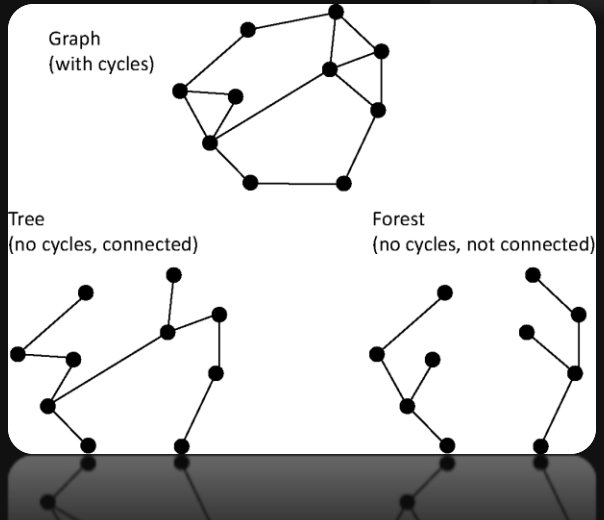
Graph Terminology

- A node is a point in the graph.
- An edge is a line that connects two node.
- A path is a sequence of edges that connects two nodes.
- A cycle is a path that starts and ends at the same nodes.
- A connected graph is a graph in which there is a path between every pair of nodes.

Type of Graph

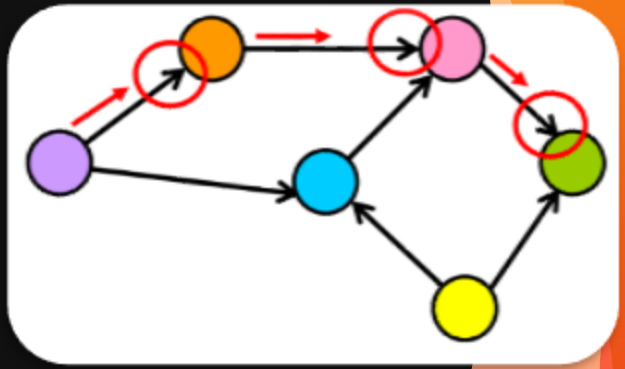
- Directed/Undirected Graph
- Weighted/Unweighted Graph
- Complete graph
- Bipartite graph
- Tree/Forest graph

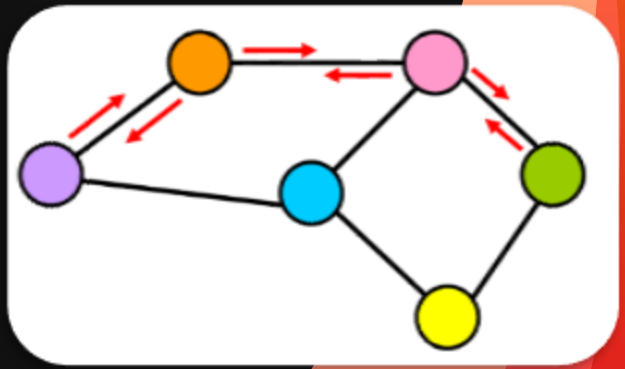




Directed/Undirected Graph

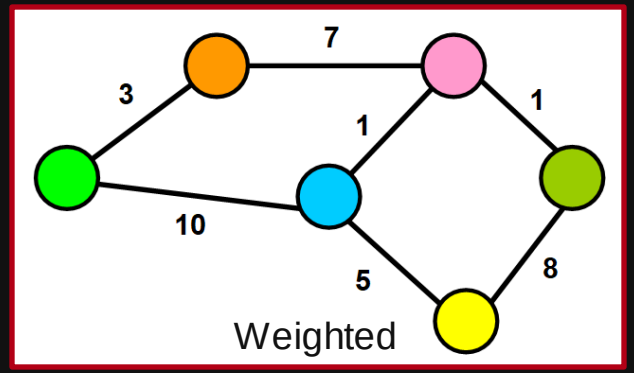
- In a directed graph,
  - edges are unidirectional, and
  - there is a direction associated with each edge.
- In an undirected graph,
  - edges are bidirectional, and
  - there is no direction associated with the edges.

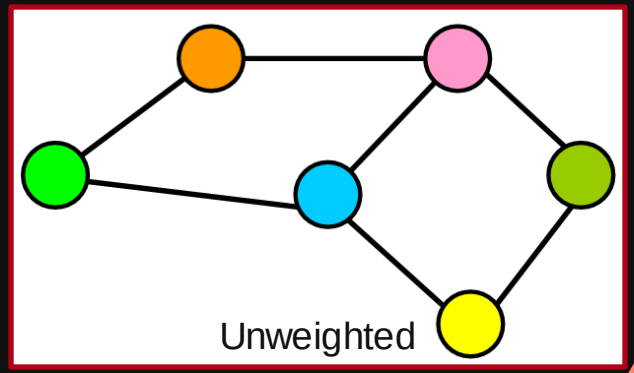




Weighted/Unweighted Graph

- In a weighted graph,
  - edges have different weights or costs associated with them
- In an unweighted graph,
  - all edges have the same weight or cost.





Demo

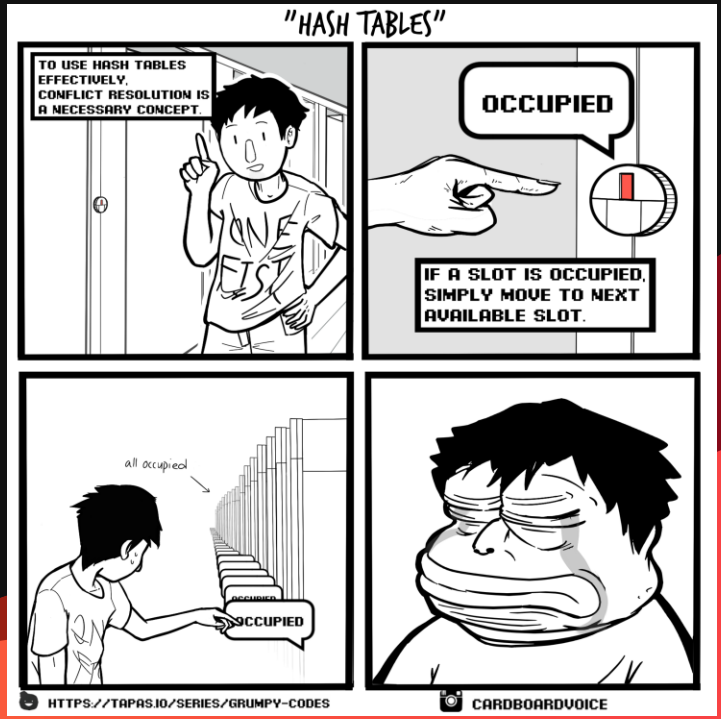
Resources

Topic	Website
Introduction to Graph - Directed/Undirected Graph - Weighted/Unweighted Graph - Cyclic Graph	https://www.freecodecamp.org/news/data-structures-101-graphs-a-visual-introduction-for-beginners-6d88f36ec768/
Data Structure and Algorithm Visualization	https://visualgo.net/en
- Complete Graph - Tree Graph - Forest Graph	https://www.quantamagazine.org/mathematicians-prove-ringels-graph-theory-conjecture-20200219/

Hash Table (HT)

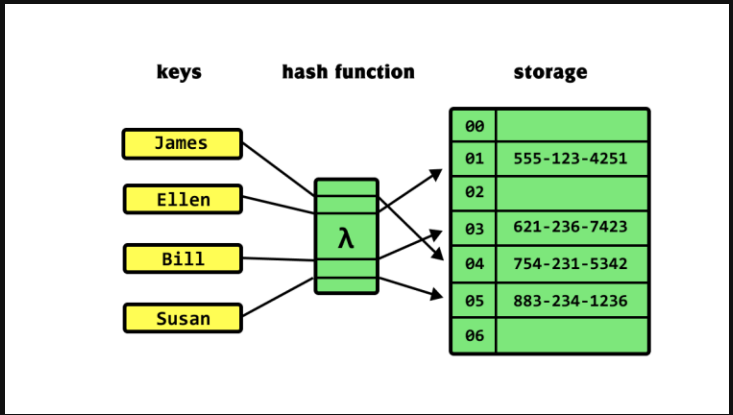
Hash Table, Hash Function,

Collision Resolution Technique etc.



Hash Table (HT)

- A hash table is a data structure that
  - allows for efficient storage and retrieval of key-value pairs.
- It is based on the idea of a hash function,
  - which maps keys to indices in an array.



Why Hash Table

- Hash tables provide
  - fast access to data,
  - with average case time complexity for
    - insertion,
    - access, and
    - deletion operations being O(1).
- They are commonly used in a variety of applications such as
  - databases,
  - compilers, and
  - network routers.

Advantage of Hash Table

- Hash tables allow for constant time access to data,
  - making them faster than other data structures such as
    - arrays or
    - linked lists.
- They also optimize memory usage by storing data in a
  - compact and
  - efficient manner.

Hash Function (HF)

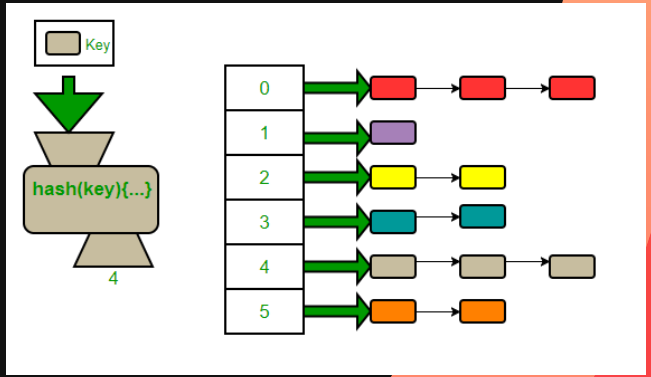
- A hash function is a function that
  - takes in data of any size and
  - returns a fixed-size output.
- Hash functions are commonly used in programming to generate
  - a unique identifier, or
  - hash code,
  - for a given piece of data.
- Good hash functions exhibit uniform distribution and a low collision rate,
  - which can improve the efficiency and effectiveness of hash table operations.

Collision Resolution Techniques (CRT)

- Chaining
- Open Addressing
- Robin Hood Hashing
- Cuckoo Hashing
- Coalesced Hashing

CRT | Chaining

- Chaining is a collision resolution technique used in hash tables.
- Each hash table bucket contains
  - a linked list of elements that have the same hash value.
- When a collision occurs, the new element is
  - added to the linked list at the corresponding bucket.



CRT | Open Addressing

- Open addressing is a collision resolution technique used in hash tables.
- When a collision occurs in open addressing, the hash table looks for the next available slot in the table and inserts the new element in that slot.
- The main advantage of open addressing is that it can
  - avoid the memory overhead associated with chaining.
  - allows all elements to be stored in the hash table itself,
    - which can save memory and improve cache performance.
  - better cache utilization
    - because all elements are stored in contiguous memory locations.

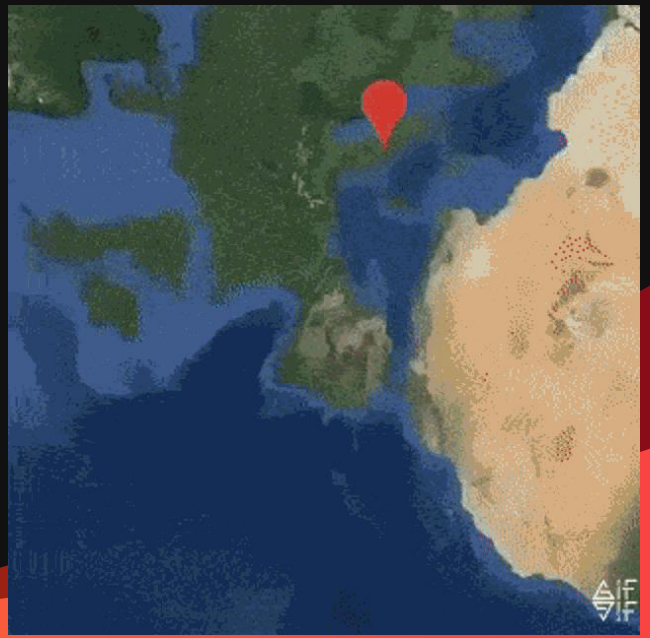
CRT | Type of Open Addressing

- There are 3 main types of open addressing:
  - linear probing,
    - checks the next consecutive slot to resolve collisions.
  - quadratic probing, and
    - uses a quadratic function to determine the next slot to check when a collision occurs.
  - double hashing.
    - uses a second hash function to determine the next slot to check when a collision occurs.

Demo

Map

Dictionary Abstract Data Type (ADT)



Map

- Maps are a data structure that store key-value pairs.
- They allow for efficient storage and retrieval of data.
- In C++, the std::map container provides a built-in implementation of a map.
- Maps are useful for associating a set of values with a corresponding set of keys.
- They offer advantages such as fast search times and efficient memory usage.
- By understanding how maps work and how to use them effectively,
  - We can improve the performance and functionality of their programs.

Note: the concept of a dictionary and a map are generally interchangeable

Demo

Thank you!

Please give us some feedback :D