Course: Advanced Algorithm and Problem Solving

WEEK 6 - Advanced Data Structures

Lemlem Kassa (Ph.D.)

Addis Ababa Science and Technology University, Ethiopia

April ,2025

Week 6: Advanced Data Structures

Contents

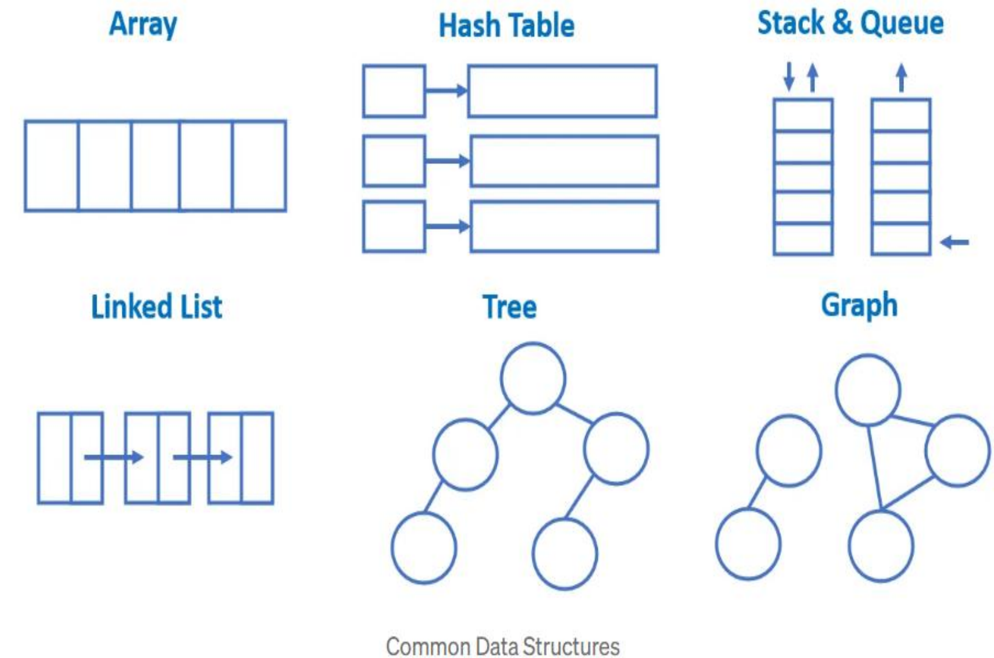
- Introduction to Trees
  - Tree Terminology and Rules
  - Importance of Tree
- Types of Trees
- Applications of Trees
- Introduction to Hashing
  - Components of Hashing
- Types of Hash Functions
- Collision in Hashing

Lecture Learning Outcome

- Understand Tree data structure , components and significance
- Differentiate types of Trees in Data Structure
- Understand applications of Trees
- Understand Hashing components and significance
- Understand how does Hashing work
- Differentiate types of Hash Functions
- Understand the Collision handling strategies in Hashing
- Understand Applications of Hash Table
- Understand the advantage and disadvantage of Hashing

1. Introduction to Trees

- The Tree data structure is similar to Linked Lists in that each node contains data and can be linked to other nodes.
- Linear structures means that each element follows directly after another in a sequence.
- In a Tree, a single element can have multiple 'next' elements, allowing the data structure to branch out in various directions (hierarchical).
- Understanding tree helps in choosing the right data structure for different problems, making algorithms more efficient and effective.

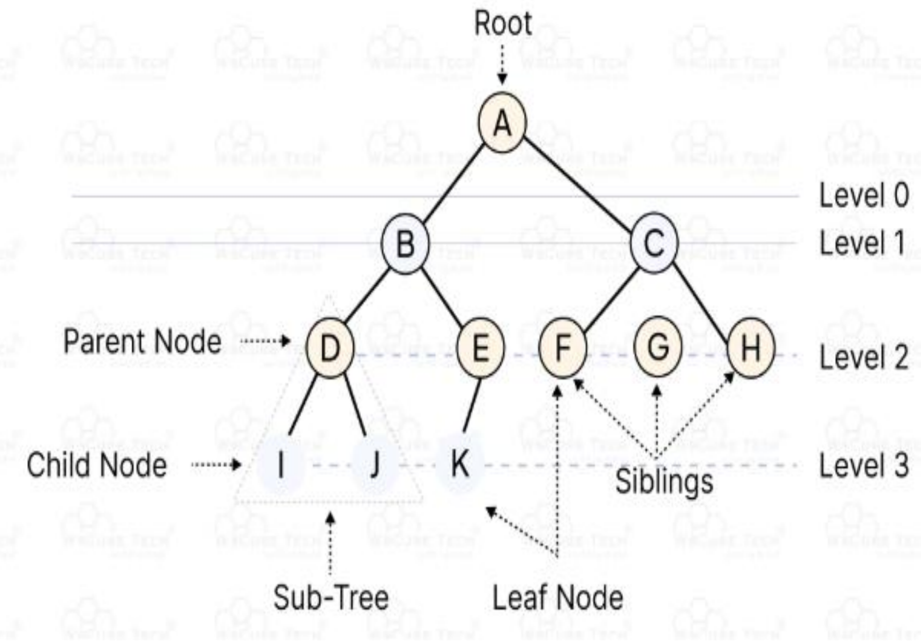


https://medium.com/codex/a-dummys-guide-to-linked-lists-part-1-44469f35f65a\

[1]. W3Schools, DSA Trees,

1. Introduction to Trees -Tree Terminology and Rules

- The first node in a tree is called the root node. A link connecting one node to another is called an edge.
- A parent node has links to its child nodes. Another word for a parent node is internal node. A node can have zero, one, or many child nodes. And it can only have one parent node.
- Nodes without links to other child nodes are called leaves, or leaf nodes. The tree height is the maximum number of edges from the root node to a leaf node.
- The height of a node is the maximum number of edges between the node and a leaf node. The tree size is the number of nodes in the tree.



https://www.wscubetech.com/resources/dsa/tree-data-structure

[2]. WsCube Tech, Tree Data Structure, https://www.wscubetech.com/resources/dsa/tree-data-structure

Introduction to Trees

Importance of Tree

The Tree data structure can be useful in many cases:

- Hierarchical Data: File systems, organizational models, etc.
- Databases: Used for quick data retrieval.
- Routing Tables: Used for routing data in network algorithms.
- Sorting/Searching: Used for sorting data and searching for data.
- Priority Queues: Priority queue data structures are commonly implemented using trees, such as binary heaps.

[1]. W3Schools, DSA Trees, https://www.w3schools.com/dsa/dsa_theory_trees

Types of Trees

Types of Trees in Data Structure

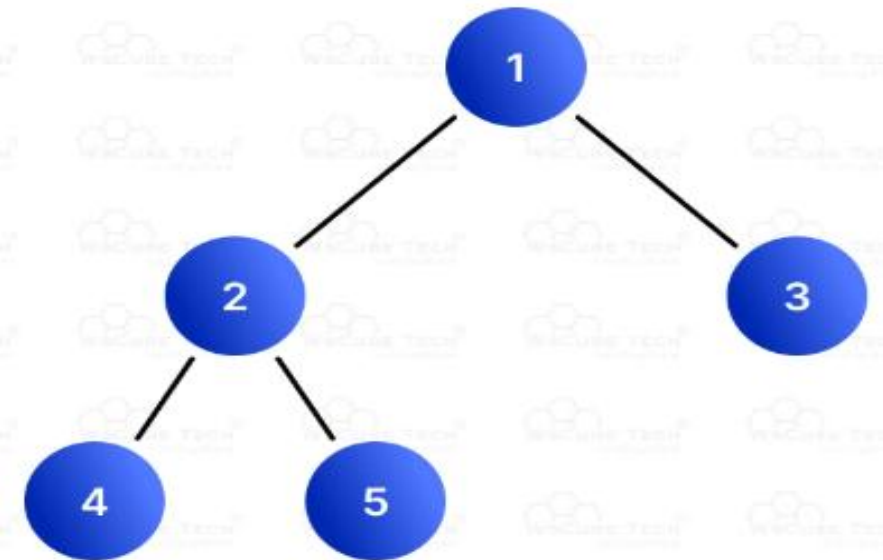
1. Binary Tree

- A binary tree is a tree data structure where **each node has at most two children**, referred to as the left child and the right child.

Properties:

- Each node can have zero, one, or two children.
- The left and right subtrees are also binary trees.

Example



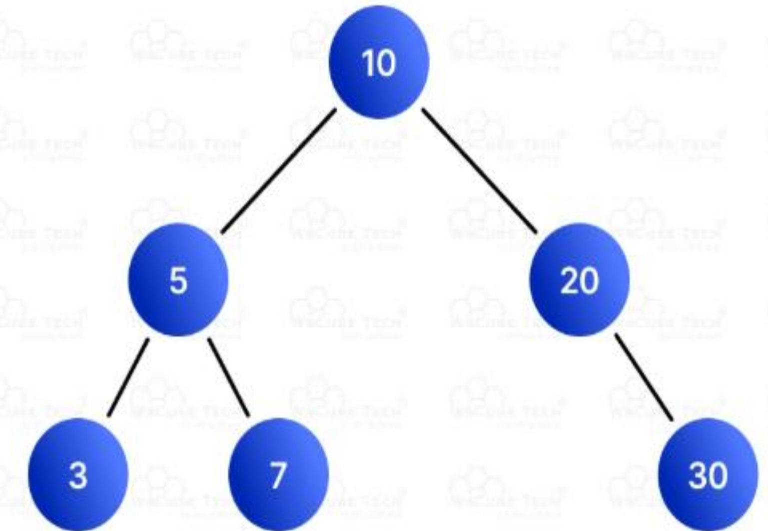
[2]. WsCube Tech, Tree Data Structure, https://www.wscubetech.com/resources/dsa/tree-data-structure

Types of Trees …(cont'd)

2. Binary Search Tree (BST)

- A binary search tree is a binary tree where each node has a value, and the left child's value is less than the parent's value, while the right child's value is greater than the parent's value.
- The high of BST maybe O(logn) or O(n), the high is under control as it depends how the key elements are inserted. This is the drawback of binary search tree

Example



Properties:

- Allows efficient searching, insertion, and deletion.
- In-order traversal of a BST gives a sorted sequence of values.

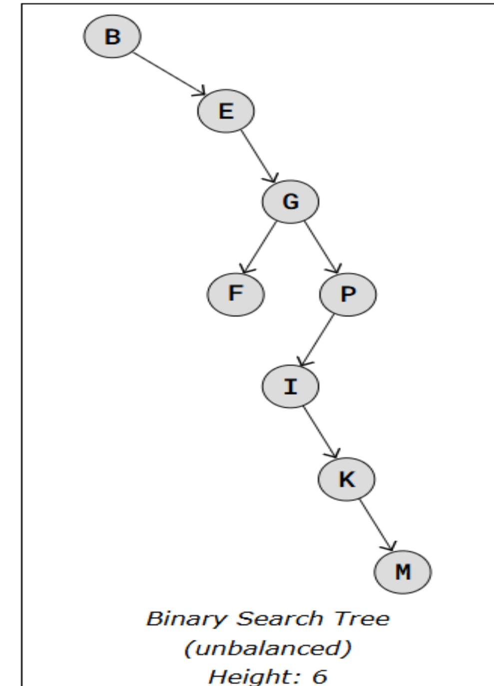
[2]. WsCube Tech, Tree Data Structure, https://www.wscubetech.com/resources/dsa/tree-data-structure

Types of Trees …(cont'd)

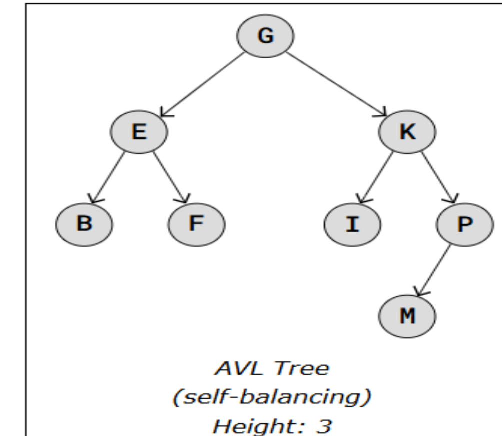
AVL Tree

- AVL trees are self-balancing, which means that the tree height is kept to a minimum so that a very fast runtime is guaranteed for searching, inserting and deleting nodes, with time complexity O(logn).
- The only difference between a regular Binary Search Tree and an AVL Tree is that AVL Trees do rotation operations in addition, to keep the tree balance.
- A Binary Search Tree is in balance when the difference in height between left and right subtrees is less than 2.
- By keeping balance, the AVL Tree ensures a minimum tree height, which means that search, insert, and delete operations can be done really fast.

Example



Binary Search Tree (unbalanced) Height: 6



AVL Tree (self-balancing) Height: 3

[2]. WsCube Tech, Tree Data Structure, https://www.wscubetech.com/resources/dsa/tree-data-structure

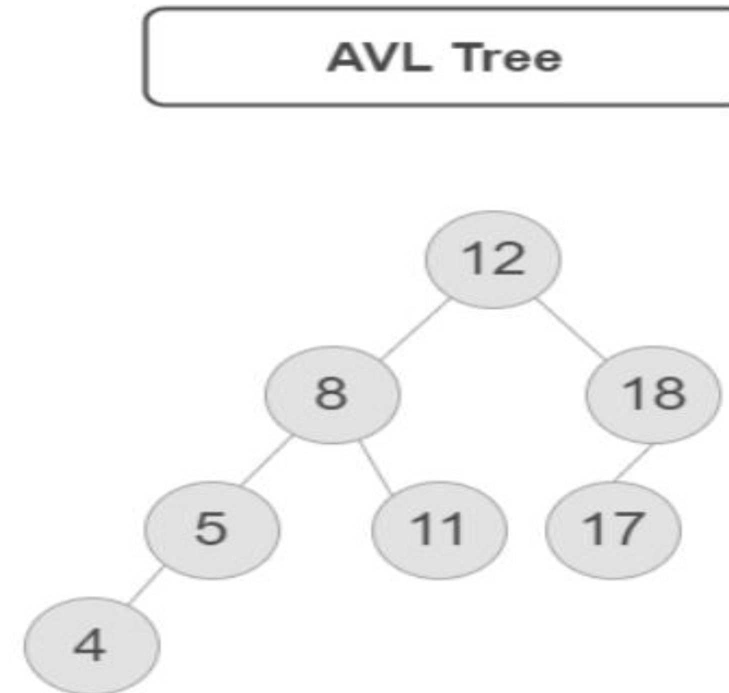
Types of Trees …(cont'd)

AVL Tree

- The absolute difference between the heights of the left subtree and the right subtree for any node is known as the **balance factor** of the node. The balance factor for all nodes must be less than or equal to 1.

Example of an AVL Tree:

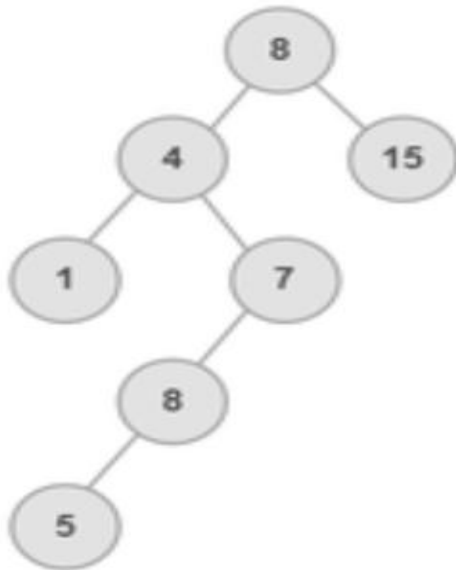
- The balance factors for different nodes are : 12 :1, 8:1, 18:1, 5:1, 11:0, 17:0 and 4:0. Since all differences are less than or equal to 1, the tree is an AVL tree.



[3]. GeeksforGeeks, AVL Tree Data Structure.
https://www.geeksforgeeks.org/introduction-to-avl-tree/

Types of Trees …(cont'd)

- **Example of a BST which is NOT AVL:**
- The Below Tree is **NOT an AVL Tree** as the balance factor for nodes 8, 4 and 7 is more than 1.



- The main advantage of an AVL Tree is, the time complexities of all operations (search, insert and delete, max, min, floor and ceiling) become O(Log n). This happens because height of an AVL tree is bounded by O(Log n).
- In case of a normal BST, the height can go up to O(n).

[3]. GeeksforGeeks, AVL Tree Data Structure. https://www.geeksforgeeks.org/introduction-to-avl-tree/

Types of Trees ...(cont'd)

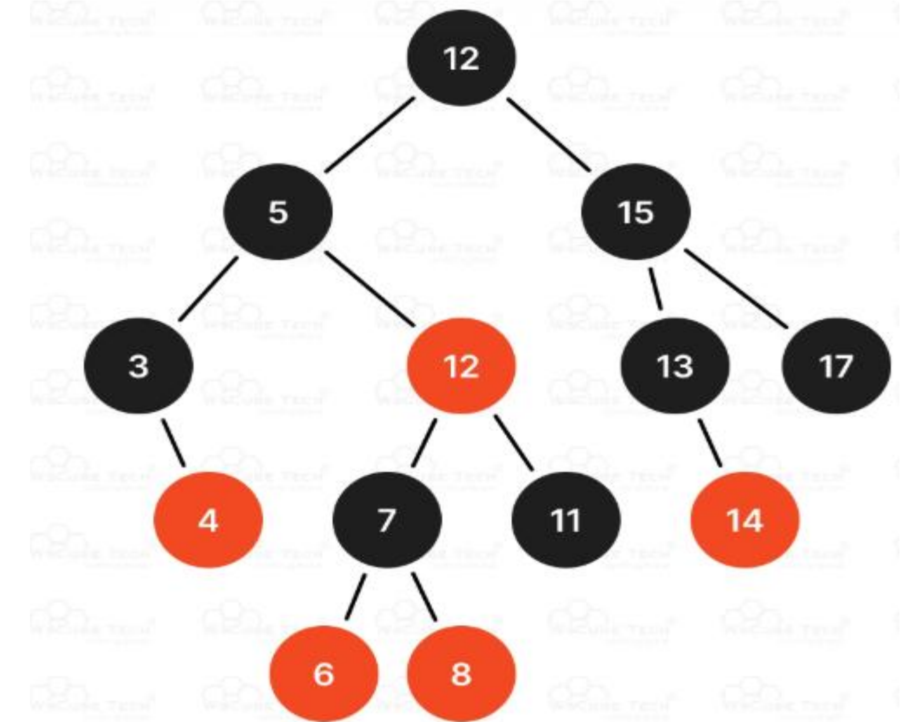
Example

4. Red-Black Tree

- A red-black tree is a self-balancing binary search tree where each node contains an extra bit for color (red or black) to ensure the tree remains balanced.

Properties:

- No two consecutive red nodes.
- If the node is red the child must be black (but not vice versa)
- The root of the tree must be black
- Every path from the root to a leaf has the same number of black nodes.
- Ensures O(log n) time complexity for insertion, deletion, and search operations.



[2]. WsCube Tech, Tree Data Structure, https://www.wscubetech.com/resources/dsa/tree-data-structure

Types of Trees …(cont'd)

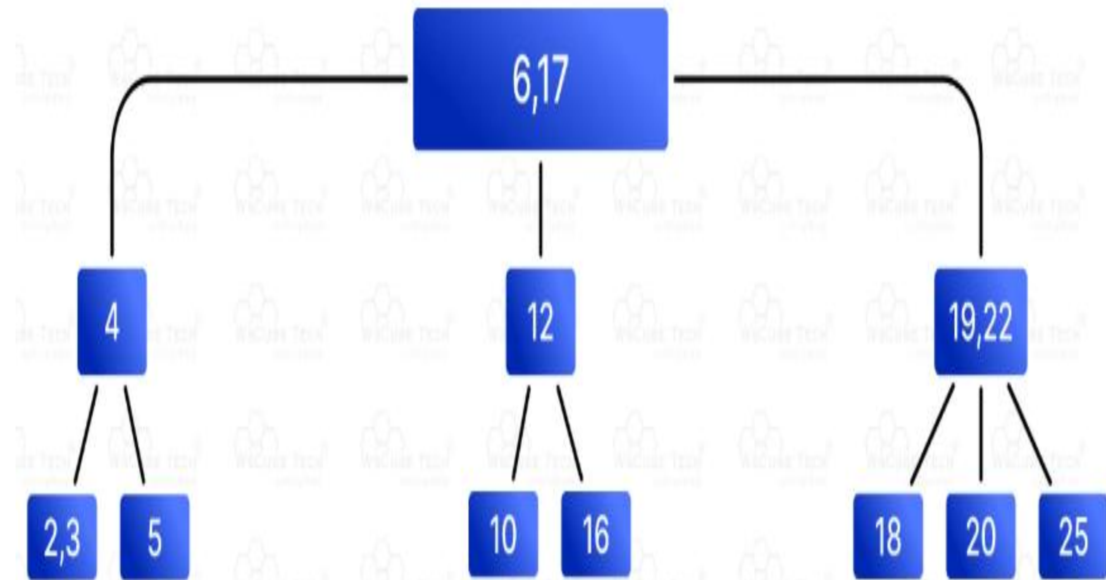
6. B-Tree

- A B-tree is a self-balancing search tree in which nodes can have multiple children. It is commonly used in databases and file systems.

Properties:

- Ensures balanced height.
- Nodes can have more than two children.
- Efficiently supports insertion, deletion, and search operations.

Example



[2]. WsCube Tech, Tree Data Structure, https://www.wscubetech.com/resources/dsa/tree-data-structure

Types of Trees ...(cont'd)

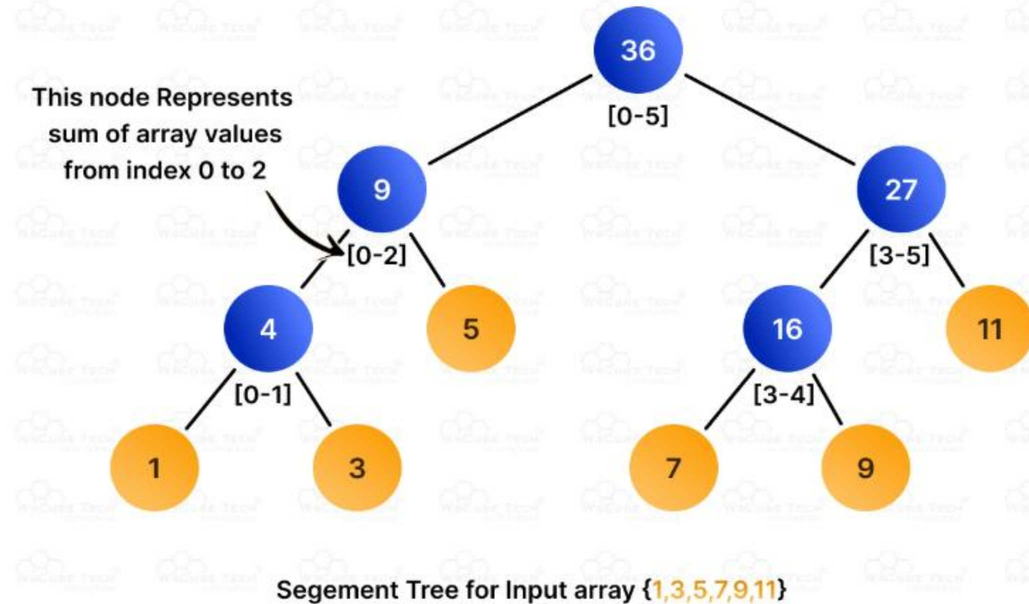
7. Segment Tree

- A segment tree is a binary tree used for storing intervals or segments.
- It allows querying which segments contain a given point.

Properties:

- Efficiently supports range queries and updates.
- Each node represents an interval.

Example



[2]. WsCube Tech, Tree Data Structure, https://www.wscubetech.com/resources/dsa/tree-data-structure

Applications of Trees

- Trees are versatile data structures that play a crucial role in various applications across computer science and related fields:

1. **Efficient Searching and Sorting**
    - Binary Search Trees (BST) are used to maintain a sorted sequence of elements and provide efficient search, insertion, and deletion operations.
    - Example: Database indexing systems use BSTs to quickly locate records.
2. **Priority Queues**
    - Heap trees are used to implement priority queues where the highest (or lowest) priority element is accessed first.
    - Example: Task scheduling systems use heaps to manage tasks by priority.
3. **Arithmetic Expressions Evaluation**
    - Expression trees represent arithmetic expressions. The leaf nodes contain operands, and the internal nodes contain operators. E.g.: Parsing and evaluating mathematical expressions in compilers.

Applications of Trees …(cont'd)

4. Data Compression

- Huffman trees are used to generate optimal prefix codes for characters based on their frequencies, reducing the overall size of the data.
- **Example:** Compression algorithms like Huffman coding use these trees to compress data efficiently.

5. Database and File Systems

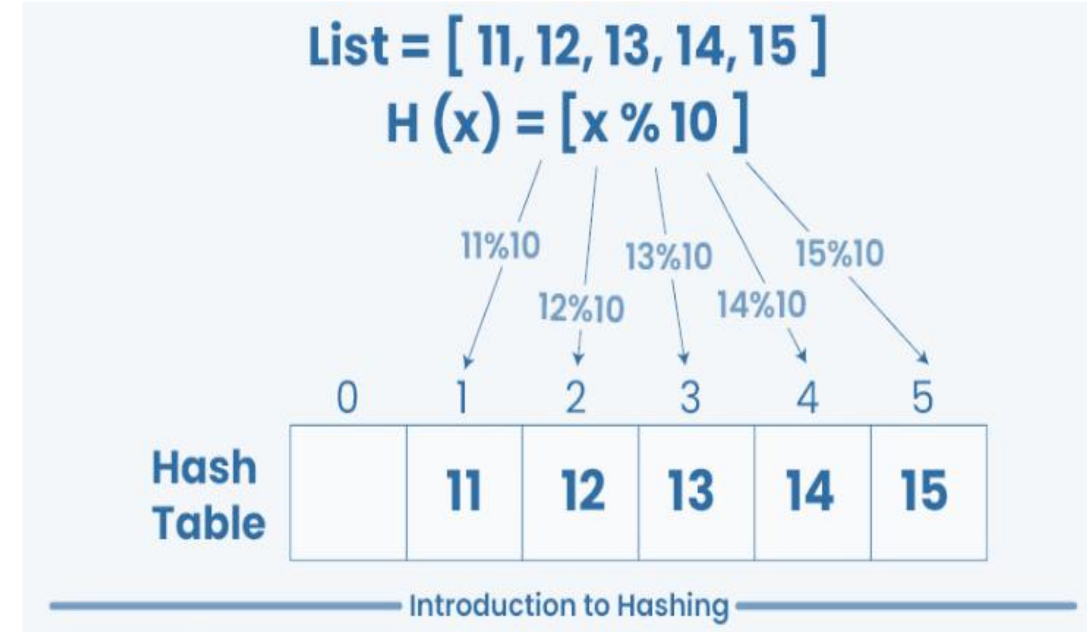
- B-Trees are self-balancing search trees that maintain sorted data and allow searches, sequential access, insertions, and deletions in logarithmic time.
- **Example:** Used in databases and file systems to manage large blocks of data.

6. Efficient Retrieval of Keys

- Tries are used to store a dynamic set of strings where keys are usually strings. They provide efficient search and retrieval operations. E.g. Implementing dictionaries and autocomplete features.

Introduction to Hashing

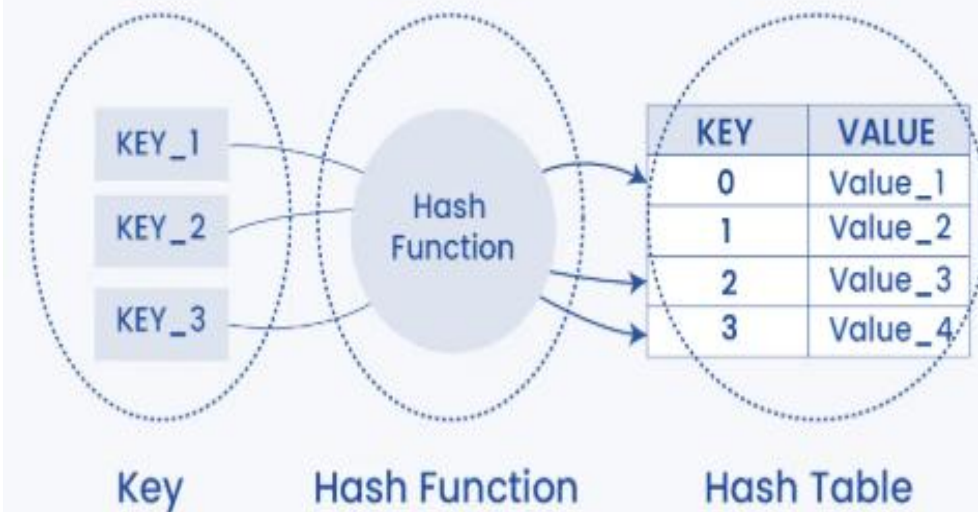
- Hashing refers to the process of generating a small sized output (that can be used as index in a table) from an input of typically large and variable size.
- Hashing uses mathematical formulas known as hash functions to do the transformation.
- This technique determines an index or location for the storage of an item in a data structure called Hash Table.



Introduction to Hashing …(cont'd)

Components of Hashing

- **Key:** can be anything string or integer ***which is fed as input in the hash function*** the technique that determines an index or location for storage of an item in a data structure.
- **Hash Function:** Receives the input key and returns the index of an element in an array called a hash table. The index is known as the hash index.
- **Hash Table:** It is typically an array of lists. It stores values corresponding to the keys.



[4] . GeeksforGeeks, Introduction to Hashinghttps://www.geeksforgeeks.org/introduction-to-hashing

Introduction to Hashing …(cont’d)

- Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects.
- Some examples of how hashing is used in our lives include:
  - In universities, each student is assigned a unique roll number that can be used to retrieve information about them.
  - In libraries, each book is assigned a unique number that can be used to determine information about the book, such as its exact position in the library or the users it has been issued to etc.
- In both these examples the students and books were hashed to a unique number

Introduction to Hashing …(cont'd)

What is Hash Table?

- A hash table is a special data structure that helps store and find data quickly using a key. Think of it like a big table where each row has a unique label (key) and a piece of information (value).
- When we want to store something, we give it a key, and the hash table uses a special math function, called a hash function, to figure out where to put it in the table.
- Later, when we want to find that piece of information, we just use the same key, and the hash table quickly tells you where it is.

Introduction to Hashing …(cont'd)

Hash function

- It is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table.
- The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes.
- basic requirements of a good hash function with the following
  - **Easy to compute**: It should be easy to compute and must not become an algorithm in itself.
  - **Uniform distribution**: It should provide a uniform distribution across the hash table and should not result in clustering.
  - **Less collisions**: Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided.

[5]. HackerEarth, Basics of Hash Tables, Prateek Garg.
https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial

How does Hashing work?

- Suppose we have a set of strings {“ab”, “cd”, “efg”} and we would like to store it in a table.
  - **Step 1:** We know that hash functions (which is some mathematical formula) are used to calculate the hash value which acts as the index of the data structure where the value will be stored.
  - **Step 2:** So, let’s assign
    - “a” = 1,
    - “b”=2, .. etc, to all alphabetical characters.
  - **Step 3:** Therefore, the numerical value by summation of all characters of the string:

“ab” = 1 + 2 = 3,
“cd” = 3 + 4 = 7 ,
“efg” = 5 + 6 + 7 = 18

[4]. GeeksforGeeks, Introduction to Hashing. https://www.geeksforgeeks.org/introduction-to-hashing

How does Hashing work? …(cont'd)

- **Step 4:** Now, assume that we have a table of size 7 to store these strings. The hash function that is used here is the sum of the characters in ***key mod Table size*** We can compute the location of the string in the array by taking the sum(string) mod 7 .
- **Step 5:** So we will then store

Mapping Key with indices of Array

0	1	2	3	4	5	6
cd			ab	egf		

"ab" in 3 mod 7 = 3,
"cd" in 7 mod 7 = 0, and
"efg" in 18 mod 7 = 4.

- The above technique enables us to calculate the location of a given string by using a simple hash function and rapidly find the value that is stored in that location.
- Therefore, the idea of hashing seems like a great way to store (key, value) pairs of the data in a table.

[4]. GeeksforGeeks, Introduction to Hashing. https://www.geeksforgeeks.org/introduction-to-hashing/

Types of Hash Functions

The following are different types of hash functions that use numeric or alphanumeric keys.

- Division Method.
- Multiplication Method
- Mid-Square Method
- Folding Method
- Cryptographic Hash Functions
- Universal Hashing
- Perfect Hashing

[4]. GeeksforGeeks, Introduction to Hashinghttps://www.geeksforgeeks.org/introduction-to-hashing

Types of Hash Functions …(cont'd)

1. Division Method

- The division method involves dividing the key by a prime number and using the remainder as the hash value.

  h(k)=k mod m **,** Where k is the key and ?m is a prime number.

E.g. If the key is 123 and the table size is 10, the hash value would be 123 % 10 = 3. So, the index is 3.

2. Multiplication Method

- In the multiplication method, a constant ?A (0 < A < 1) is used to multiply the key.

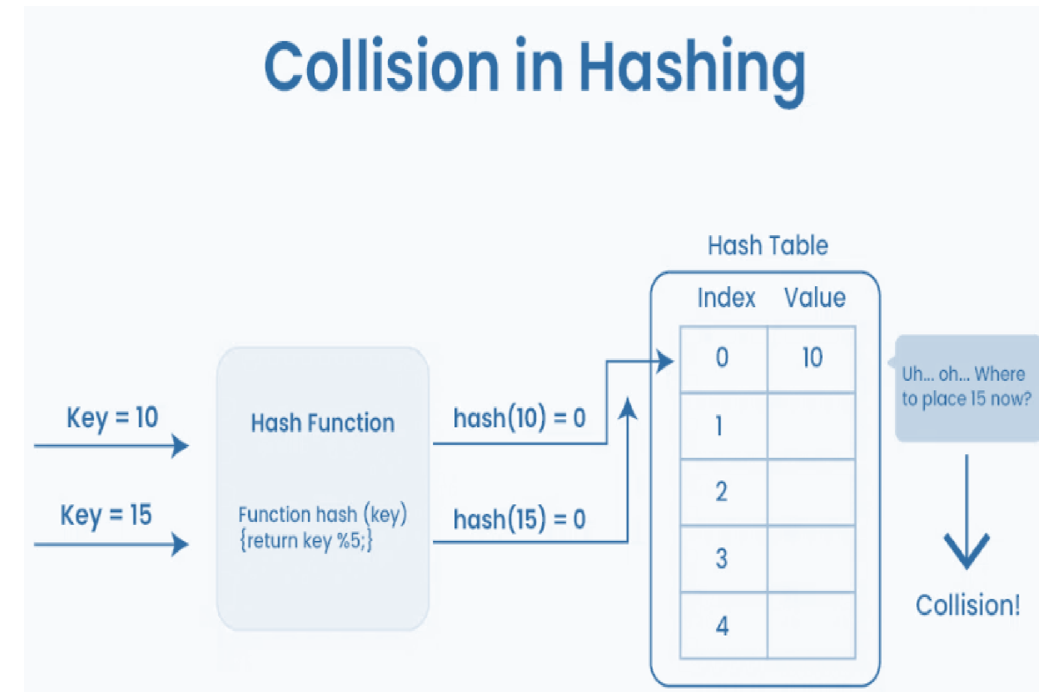
The fractional part of the product is then multiplied by ?m to get the hash value.

h(k)=[m(kAmod1)] , Where [ ] denotes the floor function.

- E.g. If the key is 123, the table size is 10, and A is 0.618, then calculate (123 * 0.618) % 1, get the fractional part, multiply by 10, and take the floor value to get the index.

Collision in Hashing …(cont'd)

- When two or more keys have the same hash value, a collision happens.
- If we consider the example in the given finger , the hash function we used is the sum of the letters, but if we examined the hash function closely then the problem can be easily visualised that for different strings same hash value is being generated by the hash function.
- Example: {"ab", "ba"} both have the same hash value, and string {"cd","be"} also generate the same hash value, etc.
- This is known as **collision** and it creates problem in searching, insertion, deletion, and updating of value.

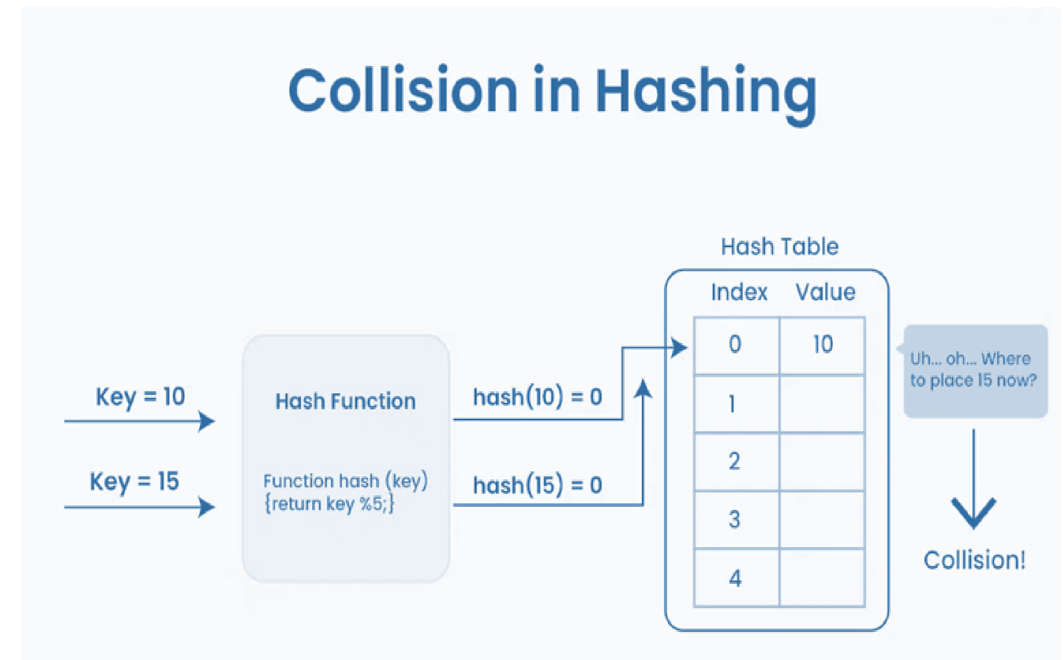


[1]. W3Schools, DSA Trees, https://www.w3schools.com/dsa/dsa_theory_trees

Collision in Hashing ...(cont'd)

Collision Resolution Techniques

- In Hashing, hash functions were used to generate hash values. The hash value is used to create an index for the keys in the hash table.
- The hash function may return the same hash value for two or more keys.
- When two or more keys have the same hash value, a collision happens.
- To handle this collision, we use **Collision Resolution Techniques.**



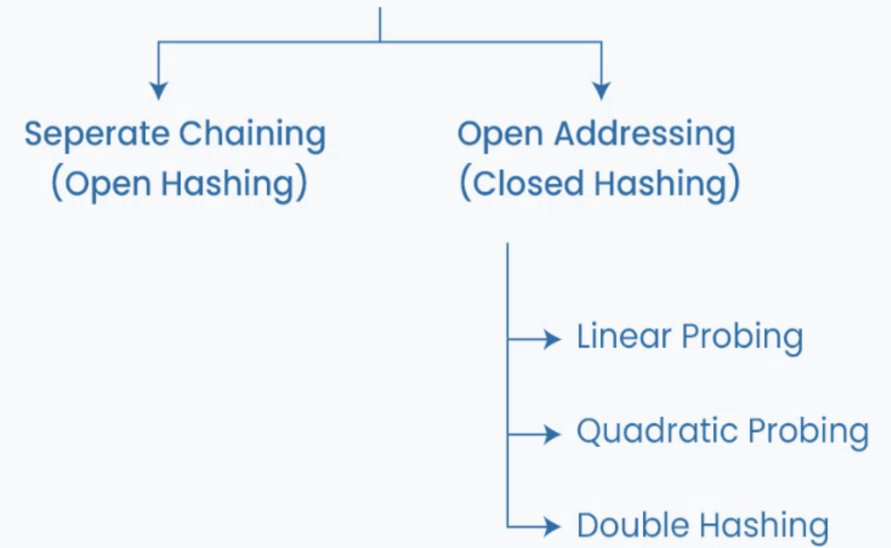
Collision in Hashing …(cont'd)

- Sometimes, two different keys produce the same index; this is called a collision. There are several ways to handle collisions:

1. Open Addressing

- **Linear Probing:** If a collision happens, the algorithm checks the next slot in the table (index + 1) until it finds an empty slot.
- **Quadratic Probing**: Similar to linear probing, but it checks slots at intervals of 1^2, 2^2, 3^2, etc., after a collision.
- **Double Hashing:** Uses a second hash function to determine how far to move when a collision occurs.

2. Chaining : Each index in the hash table points to a linked list. If multiple keys hash to the same index, their values are stored in a linked list at that index.



Collision in Hashing …(cont'd)

Example :Separate Chaining (Open Hashing)

- The idea behind Separate Chaining is to make each cell of the hash table point to a linked list of records that have the same hash function value.
- Chaining is simple but requires additional memory outside the table.

Example:

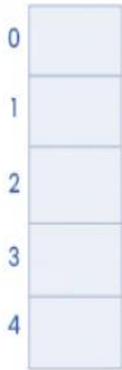
- We have given a hash function and we have to insert some elements in the hash table using a separate chaining method for collision resolution technique.

Hash function = key % 5,
Elements = 12, 15, 22, 25 and 37.

[4]. GeeksforGeeks, Introduction to Hashinghttps://www.geeksforgeeks.org/introduction-to-hashing.

Example :Separate Chaining (Open Hashing)

Slot



Step 01

Empty hash table with range of hash values from **0 to 4** according to the hash function provided.

Slot

Step 02

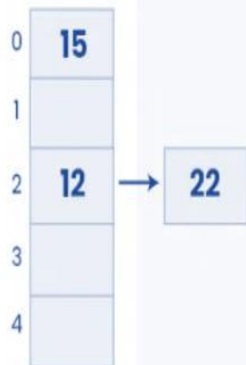
The first key to be inserted is **12** which is mapped to **slot 2 (12%5=2)**.

Slot

Step 03

The next key is **22** which is mapped to **slot 2 (22%5=2)** but **slot 2** is already occupied by **key 12**. Separate chaining will handle collision by creating a linked list to **slot 2.**

Slot



Step 04

The next key is **15** which is mapped to **slot 0 (15%5=0).**

Slot

Step 05

The next key is **25** which is mapped to **slot 0 (25%5=0)**. But slot 0 is already occupied by **key 25**. Again, Separate chaining will handle collision by creating a linked list to **slot 2**.

Collision in Hashing …(cont'd)

2.a) Linear Probing

- In linear probing, the hash table is searched sequentially that starts from the original location of the hash. If in case the location that we get is already occupied, then we check for the next location.

Algorithm:

- Calculate the hash key. i.e. *key = data % size*
- Check, if *hashTable[key]* is empty
- store the value directly by *hashTable[key] = data*
- If the hash index already has some value then
- check for next index using *key = (key+1) % size*
- Check, if the next index is available hashTable[key] then store the value. Otherwise try for next index.
- Do the above process till we find the space.

Collision in Hashing …(cont'd)

Example: Example :Linear Probing

Let us consider a simple hash function as "key mod 5" and a sequence of keys that are to be inserted are 50, 70, 76, 85, 93.

Slot

Slot	Key
0	
1	
2	
3	
4	

Step 01

Empty hash table with range of hash values from **0 to 4** according to the hash function provided.

Slot

Slot	Key
0	50
1	
2	
3	
4	

Step 02

The first key to be inserted is **50** which is mapped to **slot 0** (50%5=0)

Slot

Slot	Key
0	50
1	70
2	
3	
4	

Step 03

The next key is **70** which is mapped to **slot 0** (70%5=0) but **50** is already at **slot 0** so, search for the next empty slot and insert it.

Slot

Slot	Key
0	50
1	70
2	76
3	
4	

Step 04

The next key is **76** which is mapped to **slot 1** (76%5=1) but **70** is already at **slot 1** so, search for the next empty slot and insert it.

Slot

Slot	Key
0	50
1	70
2	76
3	85
4	

Step 05

The next key is **85** which is mapped to **slot 0** (85%5=0), but **50** is already at **slot number 0** so, search for the next empty slot and insert it. So insert it into **slot number 3.**

Slot

Slot	Key
0	50
1	70
2	76
3	85
4	93

Step 06

The next key is **93** which is mapped to **slot 3** (93%5=3), but **85** is already at **slot 3** so, search for the next empty slot and insert it. So insert it into **slot number 4.**

Collision in Hashing …(cont'd)

2.b) Quadratic Probing

- Quadratic probing is an open addressing scheme in computer programming for resolving hash collisions in hash tables.
- Quadratic probing operates by taking the original hash index and adding successive values of an arbitrary quadratic polynomial until an open slot is found.
- An example sequence using quadratic probing is:

H + 1^ 2 , H + 2^ 2 , H + 3^ 2 , H + 4^2 ………………….. H + k^ 2

[4]. GeeksforGeeks, Introduction to Hashinghttps://www.geeksforgeeks.org/introduction-to-hashing.

Collision in Hashing …(cont'd)

- **Example:** Let us consider table Size = 7, hash function as Hash(x) = x % 7 and collision resolution strategy to be f(i) = i ^2 . Insert = 22, 30, and 50.

Quadratic Probing (Open Addressing)

Slot

0
1
2
3
4
5
6

Step 01

Empty hash table with range of hash values from **0 to 6** according to the hash function provided.

Slot

0
1 22
2
3
4
5
6

Step 02

The first key to be inserted is **22** which is mapped to **slot 1 (22%7=1)**

Slot

0
1 22
2 30
3
4
5
6

Step 03

The next key is **30** which is mapped to **slot 2 (30%7=2)**

Slot

0
1 22 ← 1+0
2 30 ← $1+1^2$
3
4
5 50 ← $1+2^2$
6

Step 04

The next key is **50** which is mapped to **slot 1 (50%7=1)** but slot 1 is already occupied. So, we will search **slot 1+1^2**, i.e. **1+1 = 2**. Again slot 2 is occupied, so we will search cell **1+2^2**, i.e.**1+4 = 5**,

Applications of Hash Table

- **Database Indexing**: Quickly retrieves data using key-value pairs.
- **Caches:** Stores frequently accessed data for fast retrieval.
- **Symbol Tables in Compilers:** Maps variable names to information about them.
- **Dictionaries in Programming Languages**: Implements associative arrays for key-value data storage.
- **Routing Tables in Networks**: Maps IP addresses to routes for efficient packet delivery.

[5]. WsCube Tech, Hash Table. https://www.wscubetech.com/resources/dsa/hash-table

Applications of Hash Table …(cont'd)

- **Password Verification**: Stores hashed passwords for secure authentication.
- **Counting Frequency of Items**: Tracks occurrences of elements efficiently.
- **Spell Checking:** Quickly looks up words in a dictionary.
- **Data Deduplication:** Identifies and removes duplicate data entries.
- **Load Balancing:** Distributes tasks evenly across resources.
- **Associative Arrays:** Provides a flexible key-based data access mechanism.
- **Cryptography (Hash Functions):** Secures data through hash-based algorithms.
- **String Matching Algorithms:** Quickly finds patterns within strings.
- **File System Management:** Manages files and directories with unique keys.
- **Memory Management**: Tracks memory allocation and deallocation for efficient usage.

Advantages of Hash Tables …(cont’d)

- **Fast Data Retrieval:** Provides constant-time complexity for search operations.
- **Efficient Insertions and Deletions:** Enables quick updates to data.
- **Flexible Data Size:** Can dynamically handle varying amounts of data.
- **Handles Large Datasets Well**: Efficiently manages large collections of data.
- **Direct Access Using Keys:** Allows immediate access to data using unique keys.

[6]. WsCube Tech, Hash Table. https://www.wscubetech.com/resources/dsa/hash-table

Disadvantages of Hash Tables …(cont’d)

- **Potential for Collisions**: Keys may hash to the same index, causing conflicts.
- **Complexity of Hash Functions**: Requires well-designed hash functions to avoid collisions.
- **Wasted Memory (Due to Overhead):** May use more memory due to empty slots.
- **Difficult to Iterate in Order**: Not suitable for ordered data traversal.
- **Performance Degrades with Poor Hash Functions**: Efficiency depends on the quality of the hash function.
- **Not Suitable for Small Data Sets:** May not be as efficient for small datasets.

[6]. WsCube Tech, Hash Table. https://www.wscubetech.com/resources/dsa/hash-table

Summary

- The Tree data structure is similar to Linked Lists in that each node contains data and can be linked to other nodes
- The Tree data structure can be useful in many cases such as Hierarchical Data, Databases, Routing Tables, Sorting/Searching, and Priority Queues.
- Hashing refers to the process of generating a small sized output (that can be used as index in a table) from an input of typically large and variable size.
- There are many hash functions that use numeric or alphanumeric keys, such as Division Method, Multiplication Method, Mid-Square Method, Folding Method, Cryptographic Hash Functions, Universal Hashing, Perfect Hashing
- When two or more keys have the same hash value, a collision happens. Thus, to handle this collision, we use Collision Resolution Techniques such as Open Addressing and Chaining to handle collisions in hashing

References

1. W3Schools, DSA Trees. https://www.w3schools.com/dsa/dsa_theory_trees
2. WsCube Tech, Tree Data Structure. https://www.wscubetech.com/resources/dsa/tree-data-structure
3. GeeksforGeeks, AVL Tree Data Structure. https://www.geeksforgeeks.org/introduction-to-avl-tree/
4. GeeksforGeeks, Introduction to Hashing. https://www.geeksforgeeks.org/introduction-to-hashing.
5. HackerEarth, Basics of Hash Tables, Prateek Garg. https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial
6. WsCube Tech, Hash Table. https://www.wscubetech.com/resources/dsa/hash-table

Thank You!

For your attention