Most Asked
Data Structure
Interview QnA

@curious_.programmer

1. What is an array?

- arrays are the collection of similar types of data stored at contiguous memory locations.
- It is the simplest data structure where the data element can be accessed randomly just by using its index number

arr → array variable

[0] → Index of element to be accessed

@curious_.programmer

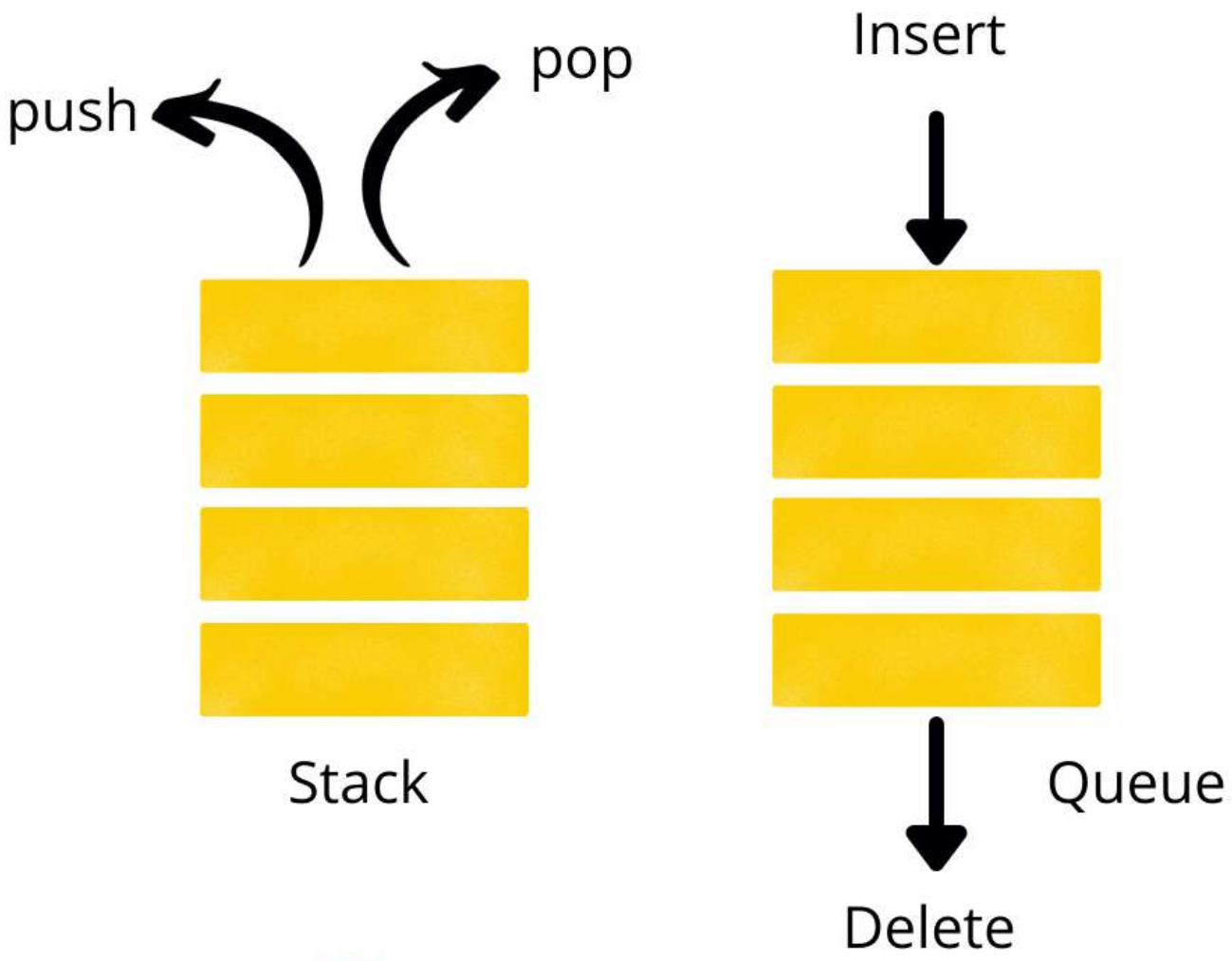
2. What is a linked list?

- A linked list is a data structure that has a sequence of nodes where every node is connected to the next node by means of a reference pointer.
- The elements are not stored in adjacent memory locations. They are linked using pointers to form a chain. This forms a chain-like link for data storage.
- Each node element has two parts:
  - a data field
  - a reference (or pointer) to the next node.

@curious_.programmer

3. How is a stack different from a queue?

1. **In a stack, the item that is most recently added is removed first whereas in queue, the item least recently added is removed first.**



@curious_.programmer

4. What is a stack? What are the applications of stack?

1. Stack is a linear data structure that follows LIFO (Last In First Out) approach for accessing elements.
2. Push, pop, and top (or peek) are the basic operations of a stack

applications of a stack:

- Check for balanced parentheses in an expression
- Evaluation of a postfix expression
- Problem of Infix to postfix conversion
- Reverse a string

@curious_.programmer

5. What is a multidimensional array?

- Multi-dimensional arrays are those data structures that span across more than one dimension.
- This indicates that there will be more than one index variable for every point of storage. This type of data structure is primarily used in cases where data cannot be represented or stored using only one dimension. Most commonly used multidimensional arrays are 2D arrays.
- 2D arrays emulates the tabular form structure which provides ease of holding the bulk of data that are accessed using row and column pointers.

@curious_.programmer

6. What is a queue? What are the applications of queue?

1. A queue is a linear data structure that follows the FIFO (First In First Out) approach for accessing elements.
2. Dequeue from the queue, enqueue element to the queue, get front element of queue, and get rear element of queue are basic operations that can be performed.

applications of queue are:

- CPU Task scheduling
- BFS algorithm to find shortest distance between two nodes in a graph.
- Website request processing
- Used as buffers in applications like MP3 media player, CD player, etc.
- Managing an Input stream

@curious_.programmer

7. Can you explain the difference between file structure and storage structure

- File Structure: Representation of data into secondary or auxiliary memory say any device such as a hard disk or pen drive that stores data that remains intact until manually deleted is known as a file structure representation.
- Storage Structure: In this type, data is stored in the main memory i.e RAM, and is deleted once the function that uses this data gets completely executed.
- The difference is that the storage structure has data stored in the memory of the computer system, whereas the file structure has the data stored in the auxiliary memory.

@curious_.programmer

8. How are linked lists more efficient than arrays?

- **Insertion and Deletion:** The insertion and deletion process is expensive in an array as the room has to be created for the new elements and existing elements must be shifted.
- **Dynamic Data Structure**: The linked list is a dynamic data structure which means there is no need to give an initial size at the time of creation as it can grow and shrink at runtime by allocating and deallocating memory. Whereas, the size of an array is limited as the number of items is statically stored in the main memory.
- **No wastage of memory**: As the size of a linked list can grow or shrink based on the needs of the program, there is no memory wasted because it is allocated in runtime.

@curious_.programmer

9. Explain the process behind storing a variable in memory.

Declaration: Before a variable can be stored in memory, it must first be declared. This involves specifying the data type of the variable and giving it a name.

Memory allocation: Once the variable is declared, the program needs to allocate memory to store it. This is typically done automatically by the compiler or interpreter. The amount of memory allocated depends on the data type of the variable. For example, an integer typically requires 4 bytes of memory.

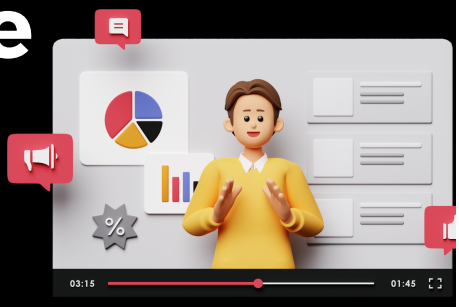
Assignment: After memory has been allocated for the variable, you can assign a value to it. This is done using the assignment operator "=".

@curious_.programmer

100 Data Structure Interview QnA

Made By:

Yadneyesh (Curious Coder)
CodWithCurious.com



1. What is data structure?

A data structure is a way of organizing and storing data in a computer's memory or storage system. It provides a systematic approach to managing and manipulating data efficiently. Examples of data structures include arrays, linked lists, stacks, queues, trees, and graphs.

2. What is an array?

An array is a data structure that stores a fixed-size sequence of elements of the same type. It provides random access to its elements using an index. Arrays are commonly used for storing and manipulating collections of data, such as a list of integers or characters.

3. What is a linked list?

A linked list is a data structure in which each element, called a node, contains a value and a reference to the next node in the sequence. Unlike arrays, linked lists do not require contiguous memory allocation, allowing for efficient insertion and deletion operations. However, accessing elements in a linked list requires traversing the list from the beginning.

4. What is a stack?

A stack is an abstract data type that follows the Last-In-First-Out (LIFO) principle. It supports two main operations: push (inserting an element onto the top of the stack) and pop (removing the topmost element from the stack). Stacks are often used for managing function calls, expression evaluation, and undo mechanisms.

5. What is a queue?

A queue is an abstract data type that follows the First-In-First-Out (FIFO) principle. It supports two primary operations: enqueue (adding an element to the end of the queue) and dequeue (removing the element at the front of the queue). Queues are commonly used in scenarios where data needs to be processed in the order it arrives, such as scheduling tasks or handling requests.

6. What is a tree?

A tree is a hierarchical data structure consisting of nodes connected by edges. It has a root node at the top and child nodes below it, forming a branching structure. Trees are used to represent hierarchical relationships, such as file systems, organization structures, and decision-making processes.

7. What is a graph?

A graph is a non-linear data structure consisting of nodes (vertices) and edges that connect them. It is a powerful tool for representing relationships between objects. Graphs can be directed (edges have a specific direction) or undirected (edges have no direction). They are widely used in network analysis, social networks, and pathfinding algorithms.

8. What is the difference between an array and a linked list?

The main difference between an array and a linked list is their underlying structure and the operations they support. Arrays have contiguous memory allocation and provide direct access to elements using an index, allowing for fast random access. Linked lists, on the other hand, use nodes with references to the next element, providing efficient insertion and deletion at any position but slower access time.

9. What is the difference between a stack and a queue?

The key difference between a stack and a queue lies in their order of operations. A stack follows the Last-In-First-Out (LIFO) principle, where the last element inserted is the first one to be removed. In contrast, a queue adheres to the First-In-First-Out (FIFO) principle, where the first element inserted is the first one to be removed. Stacks are like a pile of plates, while queues resemble a line of people waiting.

10. What is the difference between a tree and a graph?

While both trees and graphs are hierarchical structures, the main difference lies in their level of organization. A tree is a type of graph that does not contain cycles, meaning there are no loops or circular dependencies among the nodes. In contrast, a general graph can have cycles and arbitrary connections between nodes, allowing for more complex relationships.

11. What is the difference between breadth-first search (BFS) and depth-first search (DFS)?

Breadth-first search (BFS) and depth-first search (DFS) are graph traversal algorithms that visit all the nodes in a graph. The key difference is the order in which they explore the nodes. BFS visits all the neighbors of a node before moving to the next level, resembling a wave expanding from the starting point. DFS explores as far as possible along each branch before backtracking, going deeper into the graph.

12. What is the time complexity of inserting an element into an array?

The time complexity of inserting an element into an array depends on the position where the insertion needs to occur. If the element is inserted at the beginning, all existing elements must be shifted to make room, resulting in a time complexity of O(n), where n is the number of elements in the array. If the insertion happens at the end, the time complexity is constant, O(1).

13. What is the time complexity of searching for an element in an array?

The time complexity of searching for an element in an array depends on the search algorithm used. The simplest approach is linear search, which has a time complexity of O(n), where n is the number of elements in the array. Binary search, on the other hand, has a time complexity of O(log n) if the array is sorted, as it repeatedly divides the search space in half.

14. What is the time complexity of inserting an element into a linked list?

Inserting an element into a linked list typically involves updating the references of the adjacent nodes. If the insertion happens at the beginning or end of the linked list, the time complexity is constant, O(1), as it requires updating only a few references. However, inserting in the middle of a linked list requires traversing it until the desired position, resulting in a time complexity of O(n), where n is the number of elements in the linked list.

15. What is the time complexity of searching for an element in a linked list?

The time complexity of searching for an element in a linked list is O(n), where n is the number of elements in the linked list. Since linked lists do not provide random access, we need to traverse the list from the beginning until we find the desired element or reach the end. This linear traversal makes the search time proportional to the size of the linked list.

16. What is a binary search tree (BST)?

A binary search tree (BST) is a binary tree data structure in which each node has a key/value and follows a specific property: the key of any node in the left subtree is less than the key of the node itself, and the key of any node in the right subtree is greater. This property allows for efficient searching, insertion, and deletion operations, with an average time complexity of O(log n), where n is the number of nodes in the tree.

17. What is a heap data structure?

A heap is a complete binary tree data structure that satisfies the heap property: for a max heap, the key of each node is greater than or equal to the keys of its children; for a min heap, the key of each node is smaller than or equal to the keys of its children. Heaps are commonly used to implement priority queues and efficient sorting algorithms like heap sort.

18. What is a hash table?

A hash table, also known as a hash map, is a data structure that uses a hash function to map keys to values. It provides efficient insertion, deletion, and retrieval operations with an average time complexity of O(1). Hash tables are widely used for fast data lookup, such as implementing dictionaries or symbol tables.

19. What is the difference between an array and a hash table?

Arrays and hash tables differ in their underlying structure and the operations they support. Arrays provide direct access to elements using an index, allowing for fast random access. In contrast, hash tables use a hash function to map keys to values, providing efficient insertion, deletion, and retrieval operations, but without direct index-based access.

20. What is the time complexity of inserting an element into a hash table?

The time complexity of inserting an element into a hash table is typically O(1), assuming a well-designed hash function and an evenly distributed hash table. The hash function calculates the index where the element will be stored, and the element is inserted at that position. In the best case, insertion can be constant time. However, in the worst case, when collisions occur and chaining is used to resolve them, the time complexity can be O(n), where n is the number of elements in the hash table.

21. What is the time complexity of searching for an element in a hash table?

The time complexity of searching for an element in a hash table is typically O(1), assuming a well-designed hash function and an evenly distributed hash table. The hash function calculates the index of the element, and a lookup is performed at that position. In the best case, the element is found immediately. However, in the worst case, when collisions occur and chaining is used, the time complexity can be O(n), where n is the number of elements in the hash table.

22. What is a trie data structure?

A trie, also known as a prefix tree, is a tree-based data structure used to efficiently store and search for strings. Each node in the trie represents a common prefix of multiple strings, and the edges represent individual characters. Tries are particularly useful for tasks such as autocomplete, spell checking, and IP routing.

23. What is the time complexity of inserting a string into a trie?

The time complexity of inserting a string into a trie is proportional to the length of the string, denoted as O(m), where m is the length of the string. During insertion, the algorithm traverses the trie, creating new nodes as necessary until the entire string is inserted. The efficiency of tries lies in their ability to provide fast prefix-based searches.

24. What is the time complexity of searching for a string in a trie?

The time complexity of searching for a string in a trie is proportional to the length of the string, denoted as O(m), where m is the length of the string. The algorithm follows the

characters of the string, traversing the trie from the root to the corresponding leaf node. If the string exists in the trie, the search operation terminates at the leaf node. Otherwise, it reaches a point where the string is not present.

25. What is dynamic programming?

Dynamic programming is a problem-solving technique that breaks down complex problems into smaller overlapping subproblems, solving each subproblem only once and storing the results for future use. It is often used when the subproblems exhibit optimal substructure, meaning the optimal solution to the main problem can be constructed from optimal solutions to its subproblems. Dynamic programming can significantly improve the efficiency of algorithms by avoiding redundant computations.

26. What is memoization in dynamic programming?

Memoization is a technique used in dynamic programming to optimize recursive algorithms by storing the results of expensive function calls and returning the cached result when the same inputs occur again. It avoids redundant computations and improves the overall efficiency of the algorithm. Memoization is commonly implemented using arrays, hash tables, or data structures like memoization tables.

27. What is a greedy algorithm?

A greedy algorithm is an algorithmic paradigm that follows the problem-solving heuristic of making the locally optimal choice at each stage, with the hope of finding a global optimum. Greedy algorithms make decisions based on the current best option without considering the overall consequences. While they are relatively simple to design and efficient, greedy algorithms do not guarantee the optimal solution for all problems.

28. What is a divide and conquer algorithm?

A divide and conquer algorithm breaks down a problem into smaller, more manageable subproblems, solves them independently, and combines the solutions to obtain the final solution. It follows the recursive structure of dividing the problem, solving the subproblems, and merging the results. Divide and conquer algorithms are often used in sorting (e.g., merge sort, quicksort) and searching (e.g., binary search) problems.

29. What is a dynamic array?

A dynamic array, also known as a resizable array, is a data structure that provides the flexibility of resizing the array during runtime. It starts with a fixed initial capacity and dynamically allocates more memory when needed. Dynamic arrays combine the benefits of arrays, such as constant-time random access, with the ability to grow or shrink the array as necessary.

30. What is the time complexity of appending an element to a dynamic array?

The time complexity of appending an element to a dynamic array depends on the implementation. In most cases, appending an element to the end of the array requires constant time on average, denoted as O(1). However, in scenarios where the array needs

to be resized due to insufficient capacity, the time complexity can be O(n), where n is the number of elements in the array, as all elements may need to be copied to the new memory location.

31. What is the time complexity of accessing an element in a dynamic array?

The time complexity of accessing an element in a dynamic array is constant, denoted as O(1). Since dynamic arrays use contiguous memory allocation, elements can be accessed directly using an index. This provides fast random access, similar to traditional arrays.

32. What is the time complexity of removing an element from a dynamic array?

The time complexity of removing an element from a dynamic array depends on the position of the element. If the element is removed from the end of the array, the time complexity is constant, O(1), as it only requires updating the array's size. However, if the element is removed from the middle, all subsequent elements need to be shifted, resulting in a time complexity of O(n), where n is the number of elements in the array.

33. What is a red-black tree?

A red-black tree is a self-balancing binary search tree that maintains balanced properties, ensuring efficient insertion, deletion, and search operations. It achieves balance by coloring each node either red or black and applying specific rotation and color-flipping operations during insertion and deletion. Red-black trees are used in various applications, including C++ STL's set and map implementations.

34. What is the time complexity of inserting an element into a red-black tree?

The time complexity of inserting an element into a red-black tree is O(log n), where n is the number of nodes in the tree. The balancing operations performed during insertion take logarithmic time because the tree height remains balanced, thanks to the red-black tree properties. The self-balancing nature ensures that the worst-case height of the tree remains proportional to log n.

35. What is the time complexity of searching for an element in a red-black tree?

The time complexity of searching for an element in a red-black tree is O(log n), where n is the number of nodes in the tree. Similar to other balanced binary search trees, the height of the red-black tree remains balanced due to its properties. As a result, the search operation efficiently narrows down the search space, leading to a logarithmic time complexity.

36. What is a B-tree?

A B-tree is a self-balancing tree data structure designed to efficiently store and retrieve large amounts of data on disk or other secondary storage devices. It allows for efficient operations by minimizing the number of disk accesses required. B-trees are commonly used in databases and file systems, where data is organized in blocks or pages.

37. What is the time complexity of inserting an element into a B-tree?

The time complexity of inserting an element into a B-tree depends on the height of the tree. For a B-tree with a balanced structure, the height is logarithmic, resulting in an average time complexity of O(log n), where n is the number of elements in the tree. The balancing properties of B-trees ensure that the height remains balanced, leading to efficient insertions.

38. What is the time complexity of searching for an element in a B-tree?

The time complexity of searching for an element in a B-tree is similar to the insertion complexity and depends on the height of the tree. For a balanced B-tree, the height is logarithmic, resulting in an average time complexity of O(log n), where n is the number of elements in the tree. The balanced structure ensures efficient search operations by narrowing down the search space.

39. What is a priority queue?

A priority queue is an abstract data type that maintains a set of elements, each associated with a priority. It allows for efficient retrieval of the element with the highest (or lowest) priority. Priority queues are commonly implemented using binary heaps or balanced binary search trees. They find applications in scheduling, Dijkstra's algorithm, and Huffman coding, among others.

40. What is the difference between a priority queue and a regular queue?

The main difference between a priority queue and a regular queue lies in the ordering of elements. In a regular queue, elements are stored and retrieved in a First-In-First-Out (FIFO) order. However, in a priority queue, elements are associated with priorities and retrieved based on the priority order. The element with the highest (or lowest) priority is dequeued first.

41. What is the time complexity of inserting an element into a priority queue implemented with a binary heap?

The time complexity of inserting an element into a priority queue implemented with a binary heap is O(log n), where n is the number of elements in the heap. During insertion, the element is appended to the end of the heap, and then it "bubbles up" by swapping with its parent until the heap property is restored. The maximum number of swaps required is proportional to the height of the heap, which is logarithmic.

42. What is the time complexity of accessing the maximum element in a priority queue implemented with a binary heap?

The time complexity of accessing the maximum element in a priority queue implemented with a binary heap is O(1). The maximum element is always located at the root of the heap, providing direct access without the need for traversal or comparison with other elements.

43. What is the time complexity of removing the maximum element from a priority queue implemented with a binary heap?

The time complexity of removing the maximum element from a priority queue implemented with a binary heap is O(log n), where n is the number of elements in the heap. The removal process involves swapping the root with the last element, "bubbling down" the new root to its proper position, and restoring the heap property. The number of swaps required is proportional to the height of the heap, which is logarithmic.

44. What is the time complexity of sorting elements using heap sort?

The time complexity of sorting elements using heap sort is O(n log n), where n is the number of elements in the input array. Heap sort involves building a binary heap from the array (O(n)), repeatedly removing the maximum element from the heap (O(log n)) and placing it in the sorted portion of the array. The overall time complexity is dominated by the O(log n) removal operation, performed n times.

45. What is a graph traversal algorithm?

A graph traversal algorithm explores all the nodes or vertices of a graph in a systematic manner. It enables visiting each node and performing necessary operations, such as marking the node as visited or collecting information. Common graph traversal algorithms include depth-first search (DFS) and breadth-first search (BFS).

46. What is the difference between BFS and DFS graph traversal algorithms?

The main difference between breadth-first search (BFS) and depth-first search (DFS) lies in the order in which they explore nodes in a graph. BFS visits all the neighbors of a node before moving to the next level, resembling a wave expanding from the starting point. DFS explores as far as possible along each branch before backtracking, going deeper into the graph. As a result, BFS typically finds the shortest path, while DFS explores paths deeply.

47. What is the time complexity of BFS in a graph?

The time complexity of breadth-first search (BFS) in a graph is O(V + E), where V is the number of vertices (nodes) and E is the number of edges in the graph. BFS visits each vertex once and examines all its adjacent edges, resulting in a linear time complexity.

48. What is the time complexity of DFS in a graph?

The time complexity of depth-first search (DFS) in a graph is O(V + E), where V is the number of vertices (nodes) and E is the number of edges in the graph. DFS visits each vertex once and examines all its adjacent edges recursively, resulting in a linear time complexity.

49. What is a topological sort?

A topological sort is an ordering of the vertices in a directed acyclic graph (DAG) such that for every directed edge (u, v), vertex u comes before vertex v in the ordering.

Topological sorting is commonly used in tasks such as task scheduling, dependency resolution, and determining the order of events.

50. What is the time complexity of topological sort in a directed acyclic graph?

The time complexity of topological sort in a directed acyclic graph (DAG) is O(V + E), where V is the number of vertices (nodes) and E is the number of edges in the graph. The algorithm performs a depth-first search (DFS) with some modifications, resulting in a linear time complexity.

51. What is a linked list?

A linked list is a linear data structure consisting of nodes, where each node contains a value and a reference (or pointer) to the next node in the sequence. Linked lists allow for efficient insertion and deletion at any position, but accessing elements requires traversing the list from the beginning.

52. What is the time complexity of inserting an element at the beginning of a linked list?

The time complexity of inserting an element at the beginning of a linked list is O(1). Since the new element becomes the head of the list, it simply requires updating the head pointer to point to the new node.

53. What is the time complexity of inserting an element at the end of a linked list?

The time complexity of inserting an element at the end of a linked list is O(n), where n is the number of nodes in the list. To insert at the end, we need to traverse the entire list to reach the last node and then update its reference to point to the new node.

54. What is the time complexity of searching for an element in a linked list?

The time complexity of searching for an element in a linked list is O(n), where n is the number of nodes in the list. In the worst case, we may need to traverse the entire list to find the desired element.

55. What is the time complexity of removing an element from a linked list?

The time complexity of removing an element from a linked list depends on the position of the element. If the element is at the beginning, the removal operation can be done in O(1) time by updating the head pointer. If the element is in the middle or at the end, it requires traversing the list to find the element (O(n)) and updating the references accordingly.

56. What is a stack?

A stack is an abstract data type that follows the Last-In-First-Out (LIFO) principle. It can be visualized as a vertical stack of elements, where insertion and deletion occur only at one end, known as the top. The last element inserted is the first one to be removed.

57. What is the time complexity of inserting an element into a stack?

The time complexity of inserting (pushing) an element into a stack is O(1). It involves adding the element to the top of the stack by updating the top pointer.

58. What is the time complexity of removing an element from a stack?

The time complexity of removing (popping) an element from a stack is O(1). It involves removing the element from the top of the stack by updating the top pointer.

59. What is the time complexity of accessing the top element of a stack?

The time complexity of accessing (peeking) the top element of a stack is O(1). It involves retrieving the element from the top of the stack without modifying the stack itself.

60. What is a queue?

A queue is an abstract data type that follows the First-In-First-Out (FIFO) principle. It can be visualized as a horizontal line of elements, where insertion occurs at one end (rear) and removal occurs at the other end (front). The first element inserted is the first one to be removed.

61. What is the time complexity of inserting an element into a queue?

The time complexity of inserting (enqueueing) an element into a queue is O(1). It involves adding the element to the rear of the queue.

62. What is the time complexity of removing an element from a queue?

The time complexity of removing (dequeueing) an element from a queue is O(1). It involves removing the element from the front of the queue.

63. What is the time complexity of accessing the front element of a queue?

The time complexity of accessing (peeking) the front element of a queue is O(1). It involves retrieving the element from the front of the queue without modifying the queue itself.

64. What is a hash table?

A hash table is a data structure that implements an associative array abstract data type. It uses a hash function to map keys to array indices, allowing for efficient insertion, deletion, and retrieval of key-value pairs. Hash tables provide constant-time average case complexity for these operations.

65. What is a hash function?

A hash function is a function that takes an input (such as a key) and returns a fixed-size numerical value, known as a hash code or hash value. The hash function is designed to evenly distribute the hash codes across the available indices of the hash table, minimizing collisions and maximizing efficiency.

66. What is collision handling in a hash table?

Collision handling in a hash table refers to the process of dealing with situations where two or more keys result in the same hash code, leading to a collision. Common collision handling techniques include chaining (using linked lists or arrays to store multiple values at the same index) and open addressing (probing for alternative locations when a collision occurs).

67. What is the time complexity of inserting an element into a hash table?

The time complexity of inserting an element into a hash table is typically O(1) on average. However, in the worst case, when collisions are frequent and extensive chaining or probing is required, the time complexity can increase to O(n), where n is the number of elements in the hash table.

68. What is the time complexity of retrieving an element from a hash table?

The time complexity of retrieving an element from a hash table is typically O(1) on average. However, in the worst case, when collisions are frequent and extensive chaining or probing is involved, the time complexity can increase to O(n), where n is the number of elements in the hash table.

69. What is the time complexity of removing an element from a hash table?

The time complexity of removing an element from a hash table is typically O(1) on average. However, in the worst case, when collisions are frequent and extensive chaining or probing is required, the time complexity can increase to O(n), where n is the number of elements in the hash table.

70. What is a binary search tree (BST)?

A binary search tree (BST) is a binary tree data structure in which each node has a key greater than all the keys in its left subtree and smaller than all the keys in its right subtree. This property enables efficient searching, insertion, and deletion operations. In-order traversal of a BST yields a sorted sequence of keys.

71. What is the time complexity of searching for an element in a binary search tree (BST)?

The time complexity of searching for an element in a binary search tree (BST) is O(h), where h is the height of the tree. In a balanced BST, the height is logarithmic (h = log n, where n is the number of nodes), resulting in an average case time complexity of O(log n). However, in the worst case, when the tree is skewed and resembles a linked list, the height is linear (h = n), leading to a time complexity of O(n).

72. What is the time complexity of inserting an element into a binary search tree (BST)?

The time complexity of inserting an element into a binary search tree (BST) is O(h), where h is the height of the tree. In a balanced BST, the height is logarithmic (h = log n, where n is the number of nodes), resulting in an average case time complexity of O(log

n). However, in the worst case, when the tree is skewed and resembles a linked list, the height is linear (h = n), leading to a time complexity of O(n).

73. What is the time complexity of removing an element from a binary search tree (BST)?

The time complexity of removing an element from a binary search tree (BST) is O(h), where h is the height of the tree. In a balanced BST, the height is logarithmic (h = log n, where n is the number of nodes), resulting in an average case time complexity of O(log n). However, in the worst case, when the tree is skewed and resembles a linked list, the height is linear (h = n), leading to a time complexity of O(n).

74. What is a self-balancing binary search tree?

A self-balancing binary search tree is a binary search tree that automatically maintains a balanced structure during insertions and deletions. It achieves this balance by performing rotations or other operations to ensure that the height of the tree remains logarithmic, optimizing the time complexity of search, insert, and delete operations.

75. What is an AVL tree?

An AVL tree is a self-balancing binary search tree named after its inventors, Adelson-Velsky and Landis. It maintains the balance factor (the height difference between left and right subtrees) of each node, ensuring that it is always in the range of -1, 0, or 1. AVL trees perform rotations to maintain balance and achieve efficient operations with a worst-case time complexity of O(log n).

76. What is a red-black tree?

A red-black tree is a self-balancing binary search tree with an additional color attribute for each node, either red or black. The color properties and rotations maintain a balance between the left and right subtrees, ensuring that the longest path is no more than twice the length of the shortest path. Red-black trees offer efficient operations with a worst-case time complexity of O(log n).

77. What is a heap?

A heap is a complete binary tree data structure that satisfies the heap property. In a max heap, for every node, the value of the node is greater than or equal to the values of its children. In a min heap, the value of each node is smaller than or equal to the values of its children. Heaps are commonly used to implement priority queues and heap sort.

78. What is the time complexity of finding the maximum (or minimum) element in a heap?

The time complexity of finding the maximum (or minimum) element in a heap is O(1). The maximum (or minimum) element is always located at the root of the heap, allowing for direct access without the need for traversal or comparison with other elements.

79. What is the time complexity of inserting an element into a heap?

The time complexity of inserting an element into a heap is O(log n), where n is the number of elements in the heap. The insertion process involves adding the element to the next available position in the heap and "bubbling up" by swapping it with its parent until the heap property is satisfied. The number of swaps required is proportional to the height of the heap, which is logarithmic.

80. What is the time complexity of removing the maximum (or minimum) element from a heap?

The time complexity of removing the maximum (or minimum) element from a heap is O(log n), where n is the number of elements in the heap. The removal process involves swapping the root with the last element, removing the last element, and "bubbling down" the new root by swapping it with its larger (or smaller) child until the heap property is satisfied. The number of swaps required is proportional to the height of the heap, which is logarithmic.

81. What is a trie?

A trie, also known as a prefix tree, is a tree-based data structure commonly used for efficient string searching and retrieval operations. It stores a set of strings, where each node represents a prefix or a complete string. Trie nodes typically have multiple child pointers, each associated with a character. Tries are useful in applications such as autocomplete, spell-checking, and IP routing.

82. What is the time complexity of searching for a string in a trie?

The time complexity of searching for a string in a trie is O(m), where m is the length of the string. The search process involves traversing the trie from the root to the leaf node corresponding to the last character of the string. The number of comparisons required is proportional to the length of the string.

83. What is the time complexity of inserting a string into a trie?

The time complexity of inserting a string into a trie is O(m), where m is the length of the string. The insertion process involves traversing the trie based on the characters of the string and creating new nodes as necessary. The number of operations is proportional to the length of the string.

84. What is a graph?

A graph is a non-linear data structure consisting of a set of vertices (nodes) connected by edges. Graphs can be used to represent various real-world relationships or networks. They can be directed (edges have a specific direction) or undirected (edges have no direction). Graphs are widely used in areas such as social networks, transportation networks, and computer networks.

85. What is a weighted graph?

A weighted graph is a graph in which each edge is assigned a weight or cost. The weight represents some value associated with the edge, such as the distance between two vertices or the cost of traversing the edge. Weighted graphs are used to model scenarios where edges have different significance or cost.

86. What is a directed acyclic graph (DAG)?

A directed acyclic graph (DAG) is a directed graph that does not contain any directed cycles. In other words, it is impossible to traverse from a vertex and return back to it by following the directions of the edges. DAGs are used in various applications, including task scheduling, dependency resolution, and representing precedence relationships.

87. What is a minimum spanning tree (MST)?

A minimum spanning tree (MST) is a subset of the edges of a weighted undirected graph that connects all the vertices with the minimum possible total edge weight. MSTs are used to find the most cost-effective way to connect a set of nodes. Common algorithms for finding MSTs include Prim's algorithm and Kruskal's algorithm.

88. What is Dijkstra's algorithm?

Dijkstra's algorithm is a graph traversal algorithm used to find the shortest path between a starting vertex and all other vertices in a weighted graph with non-negative edge weights. It maintains a priority queue to continuously select the vertex with the smallest distance from the starting vertex and updates the distances of adjacent vertices accordingly. Dijkstra's algorithm guarantees the shortest paths when all edge weights are non-negative.

89. What is the time complexity of Dijkstra's algorithm?

The time complexity of Dijkstra's algorithm depends on the data structure used to implement the priority queue. When implemented with a binary heap or Fibonacci heap, the time complexity is $O((V + E) \log V)$, where V is the number of vertices and E is the number of edges in the graph.

90. What is the difference between a breadth-first search (BFS) and a depth-first search (DFS)?

BFS and DFS are graph traversal algorithms with different exploration strategies. BFS explores all the vertices at the current depth level before moving to the next depth level, while DFS explores as far as possible along each branch before backtracking. BFS uses a queue data structure, while DFS uses a stack or recursion.

91. What is dynamic programming?

Dynamic programming is a problem-solving technique that solves complex problems by breaking them down into overlapping subproblems and solving each subproblem only once, storing the results in a table (memoization) for future use. It is particularly useful

when the problem exhibits optimal substructure and overlapping subproblems. Dynamic programming can significantly improve the efficiency of recursive algorithms.

92. What is memoization in dynamic programming?

Memoization is a technique used in dynamic programming to store the results of expensive function calls and avoid redundant computations. It involves caching the computed values of subproblems in a lookup table or an array, allowing subsequent calls to retrieve the stored results instead of recomputing them. Memoization can greatly reduce the time complexity of recursive algorithms.

93. What is the time complexity of a recursive algorithm with memoization?

The time complexity of a recursive algorithm with memoization depends on the number of distinct subproblems encountered. If there are n subproblems, and the time complexity of solving each subproblem is O(1), the overall time complexity is O(n).

94. What is the difference between an array and a linked list?

An array is a contiguous block of memory that stores elements of the same type. Accessing elements in an array is fast and constant time (O(1)) because they can be accessed directly using their indices. However, inserting or deleting elements in the middle of an array requires shifting subsequent elements, resulting in a time complexity of O(n).

On the other hand, a linked list is a data structure where elements (nodes) are scattered in memory and connected through pointers. Insertion and deletion operations in a linked list can be done in constant time (O(1)) by adjusting pointers, but accessing elements requires traversing the list, resulting in a time complexity of O(n).

95. What is the difference between a stack and a queue?

A stack follows the Last-In-First-Out (LIFO) principle, allowing insertion and deletion only at one end (top). The last element inserted is the first one to be removed.

A queue follows the First-In-First-Out (FIFO) principle, allowing insertion at one end (rear) and deletion at the other end (front). The first element inserted is the first one to be removed.

96. What is the difference between a hash table and a binary search tree?

A hash table uses a hash function to map keys to array indices and provides constant-time average case complexity for insertion, deletion, and retrieval operations. However, hash tables do not naturally maintain order and may experience collisions, affecting performance.

A binary search tree (BST) maintains elements in a sorted order based on their keys. BSTs provide efficient searching, insertion, and deletion operations with a time complexity of O(log n) in balanced trees. However, the time complexity can degrade to O(n) in worst-case scenarios.

97. What is the difference between a graph and a tree?

A graph is a non-linear data structure consisting of a set of vertices connected by edges. It can have cycles and may or may not be connected.

A tree is a type of graph that is acyclic (no cycles) and connected. A tree has a root node and a hierarchical structure where each node has zero or more child nodes. There is a unique path between any two nodes in a tree.

98. What is the difference between a breadth-first search (BFS) and a depth-first search (DFS) in a graph?

BFS and DFS are graph traversal algorithms with different exploration strategies:

- BFS explores all the vertices at the current depth level before moving to the next depth level. It uses a queue to store the vertices and visits them in the order of their discovery.
- DFS explores as far as possible along each branch before backtracking. It uses a stack or recursion to store the vertices and visits them in a depth-first manner.

BFS is typically used to find the shortest path between two vertices or to visit all vertices in a connected component. DFS is useful for tasks such as finding cycles, topological sorting, and exploring paths in a graph.

99. What is the difference between a spanning tree and a minimum spanning tree?

A spanning tree of a graph is a subgraph that includes all the vertices of the graph while forming a tree structure without any cycles. It preserves the connectivity of the original graph.

A minimum spanning tree (MST) is a spanning tree with the minimum possible total edge weight. It connects all the vertices with the least overall cost. MSTs are useful in scenarios such as designing network infrastructure or connecting a set of locations with minimal expenses.

100. What is the difference between an algorithm and a data structure?

An algorithm is a step-by-step procedure or a set of rules for solving a problem or accomplishing a specific task. It defines a sequence of operations or computational steps to transform input data into desired output.

100 Most Asked Data Structure QnA

Made By:

Yadneyesh (Curious Coder)
CodWithCurious.com

1. **What is a data structure?**
   a. A data structure is a way of organizing and storing data in a computer so that it can be accessed and manipulated efficiently.
2. **What is the difference between an array and a linked list?**
   a. An array stores elements of the same type in contiguous memory locations, allowing random access to elements. A linked list stores elements in separate objects called nodes, which are connected through pointers, providing dynamic memory allocation and efficient insertion/deletion at any position.
3. **What is the time complexity of accessing an element in an array and a linked list?**
   a. Accessing an element in an array takes constant time O(1) since the position is known. In a linked list, accessing an element takes linear time O(n) as you need to traverse the list.
4. **What is the difference between a stack and a queue?**
   a. A stack follows the Last-In-First-Out (LIFO) principle, where the last element inserted is the first one to be removed. A queue follows the First-In-First-Out (FIFO) principle, where the first element inserted is the first one to be removed.
5. **What is the difference between a stack and a heap?**
   a. A stack is used for local variables and function calls, and its memory allocation and deallocation are handled automatically. A heap is used for dynamically allocated memory and needs explicit memory management.
6. **What is a binary tree?**
   a. A binary tree is a data structure where each node can have at most two children. The left child is generally smaller, and the right child is greater than the parent.
7. **What is the difference between a binary tree and a binary search tree (BST)?**
   a. A binary tree can have any values in its nodes, whereas a BST follows the property that for any node, all elements in its left subtree are smaller, and all elements in its right subtree are larger.
8. **What is the time complexity of searching for an element in a binary search tree (BST)?**
   a. The time complexity of searching for an element in a BST is O(log n) in the average case and O(n) in the worst case if the tree is skewed.

9. **What is a hash table?**
    a. A hash table is a data structure that stores key-value pairs using a hash function. It provides constant-time average-case complexity for insertion, deletion, and search operations.
10. **What is collision resolution in a hash table?**
    a. Collision resolution is the process of handling situations where two different keys map to the same hash value. Common techniques include chaining (using linked lists) and open addressing (probing neighboring locations).
11. **What is the difference between a stack and a heap memory allocation?**
    a. Stack memory allocation is used for static memory allocation and follows a Last-In-First-Out (LIFO) structure, while heap memory allocation is used for dynamic memory allocation and allows flexible memory management.
12. **What is a doubly linked list?**
    a. A doubly linked list is a type of linked list where each node contains a reference to both the next and previous nodes, allowing traversal in both directions.
13. **What is a circular linked list?**
    a. A circular linked list is a type of linked list where the last node points back to the first node, creating a circular structure. It can be singly or doubly linked.
14. **What is a priority queue?**
    a. A priority queue is an abstract data type that allows elements to be inserted with a priority and removes the highest-priority element first.
15. **What is the difference between a shallow copy and a deep copy?**
    a. A shallow copy creates a new object with references to the same memory locations as the original object. A deep copy creates a new object with its own copy of the data, recursively copying all the referenced objects.
16. **What is the time complexity of various operations in a priority queue implemented as a binary heap?**
    a. The time complexity of insertion and deletion (extracting the minimum or maximum) in a binary heap-based priority queue is O(log n), while searching for an element takes O(n).
17. **What is the difference between BFS (Breadth-First Search) and DFS (Depth-First Search)?**
    a. BFS explores all the vertices of a graph or tree level by level, while DFS explores a branch as far as possible before backtracking.
18. **What is a trie?**
    a. A trie, also known as a prefix tree, is a tree-like data structure used for efficient retrieval of keys that share prefixes. It is often used for implementing dictionary-like structures.
19. **What is the time complexity of searching in a trie?**
    a. The time complexity of searching in a trie is O(m), where m is the length of the search string. It does not depend on the number of keys stored in the trie.
20. **What is an AVL tree?**

a. An AVL tree is a self-balancing binary search tree where the heights of the left and right subtrees of any node differ by at most one. It ensures that the tree remains balanced, providing efficient search, insertion, and deletion operations.

21. **What is the time complexity of searching in an AVL tree?**
    a. The time complexity of searching in an AVL tree is O(log n), as the tree is balanced.
22. **What is a B-tree?**
    a. A B-tree is a self-balancing search tree that maintains sorted data and allows efficient search, insertions, and deletions. It is commonly used in databases and file systems.
23. **What is the difference between a B-tree and a binary search tree?**
    a. A B-tree can have multiple keys per node and multiple children, while a binary search tree can have at most two children per node. B-trees are designed for efficient disk access and can store a larger number of keys per node.
24. **What is the time complexity of searching in a B-tree?**
    a. The time complexity of searching in a B-tree is O(log n), as the tree is balanced and ensures efficient search operations.
25. **What is the difference between a graph and a tree?**
    a. A tree is a connected acyclic graph with a unique root, while a graph can have cycles and may not have a root.
26. **What is a spanning tree?**
    a. A spanning tree of a graph is a subgraph that includes all the vertices of the graph with the minimum possible number of edges. It does not contain cycles.
27. **What are the different types of graph traversals?**
    a. The different types of graph traversals are Breadth-First Search (BFS) and Depth-First Search (DFS).
28. **What is Dijkstra's algorithm?**
    a. Dijkstra's algorithm is a popular algorithm for finding the shortest path between two nodes in a graph with non-negative edge weights.
29. **What is Kruskal's algorithm?**
    a. Kruskal's algorithm is a greedy algorithm used to find a minimum spanning tree in a weighted undirected graph.
30. **What is the time complexity of Dijkstra's algorithm?**
    a. The time complexity of Dijkstra's algorithm is O((V + E) log V), where V is the number of vertices and E is the number of edges in the graph.
31. **What is the time complexity of Kruskal's algorithm?**
    a. The time complexity of Kruskal's algorithm is O(E log E), where E is the number of edges in the graph.
32. **What is memoization?**
    a. Memoization is a technique used to optimize recursive algorithms by caching the results of function calls and reusing them when the same inputs occur again, avoiding redundant computations.

33. **What is the difference between a linear search and a binary search?**
    a. Linear search sequentially checks each element in a list until a match is found, while binary search divides a sorted list into halves and compares the middle element to the target value, reducing the search space in each iteration.
34. **What is the time complexity of a linear search?**
    a. The time complexity of a linear search is O(n), where n is the number of elements in the list.
35. **What is the time complexity of a binary search?**
    a. The time complexity of a binary search is O(log n), where n is the number of elements in the sorted list.
36. **What is a self-balancing binary search tree?**
    a. A self-balancing binary search tree is a binary search tree that automatically maintains its balance during insertions and deletions, ensuring efficient search operations.
37. **What are some examples of self-balancing binary search trees?**
    a. Examples of self-balancing binary search trees include AVL tree, Red-Black tree, and Splay tree.
38. **What is the time complexity of insertions and deletions in a self-balancing binary search tree?**
    a. The time complexity of insertions and deletions in a self-balancing binary search tree is O(log n), as the tree ensures balance after each operation.
39. **What is the difference between a hash set and a hash map?**
    a. A hash set is a collection of unique elements, while a hash map is a collection of key-value pairs, where each key is unique.
40. **What is the difference between a graph and a tree?**
    a. A tree is a special type of graph with no cycles, while a graph can have cycles.
41. **What is a self-loop in a graph?**
    a. A self-loop is an edge in a graph that connects a vertex to itself.
42. **What is a directed graph?**
    a. A directed graph, also known as a digraph, is a graph where edges have a direction. The edges indicate a one-way relationship between vertices.
43. **What is the difference between a breadth-first search (BFS) and a depth-first search (DFS) in a graph?**
    a. BFS explores all the vertices at the same level before moving to the next level, while DFS explores as far as possible along each branch before backtracking.
44. **What is a cyclic graph?**
    a. A cyclic graph is a graph that contains at least one cycle, i.e., a path that starts and ends at the same vertex.
45. **What is a topological sort?**
    a. A topological sort is an ordering of the vertices of a directed acyclic graph (DAG) such that for every directed edge (u, v), vertex u comes before v in the ordering.
46. **What is the time complexity of finding a cycle in a graph?**

a. The time complexity of finding a cycle in a graph is O(V + E), where V is the number of vertices and E is the number of edges.

47. **What is the time complexity of a depth-first search (DFS) in a graph?**
    a. The time complexity of a depth-first search in a graph is O(V + E), where V is the number of vertices and E is the number of edges.
48. **What is the time complexity of a breadth-first search (BFS) in a graph?**
    a. The time complexity of a breadth-first search in a graph is O(V + E), where V is the number of vertices and E is the number of edges.
49. **What is Huffman coding?**
    a. Huffman coding is a lossless data compression algorithm that assigns variable-length codes to characters based on their frequencies, with more frequent characters having shorter codes.
50. **What is the time complexity of the merge sort algorithm?**
    a. The time complexity of the merge sort algorithm is O(n log n), where n is the number of elements to be sorted.
51. **What is a red-black tree?**
    a. A red-black tree is a self-balancing binary search tree with additional properties that ensure it remains balanced. It guarantees a worst-case time complexity of O(log n) for search, insert, and delete operations.
52. **What is the difference between a binary tree and a binary search tree (BST)?**
    a. In a binary tree, there are no specific rules for the order of elements, while in a BST, the left subtree of a node contains values smaller than the node, and the right subtree contains values greater than the node.
53. **What is a trie and how is it different from a binary search tree (BST)?**
    a. A trie, also known as a prefix tree, is a tree-like data structure primarily used for efficient retrieval of keys that share prefixes. Unlike a BST, a trie does not use comparisons to store or retrieve elements but instead relies on the structure of the keys themselves.
54. **What is a skip list?**
    a. A skip list is a probabilistic data structure that allows efficient search, insert, and delete operations. It consists of multiple layers of linked lists, with higher layers skipping elements, providing faster access to desired elements.
55. **What is the time complexity of searching in a skip list?**
    a. The time complexity of searching in a skip list is O(log n), where n is the number of elements stored in the list.
56. **What is a self-balancing binary search tree?**
    a. A self-balancing binary search tree automatically maintains its balance during insertions and deletions to ensure efficient search operations. Examples include AVL tree, Red-Black tree, and Splay tree.
57. **What is a self-balancing binary search tree rotation?**
    a. A rotation is an operation performed on a self-balancing binary search tree to maintain or restore balance. It involves moving nodes around to restructure the

tree while preserving the order of elements.

58. **What is the difference between a complete binary tree and a full binary tree?**
    a. A complete binary tree is a tree where all levels, except possibly the last, are fully filled, and all nodes are as left as possible. A full binary tree is a tree where all nodes have either 0 or 2 children.
59. **What is an in-order traversal of a binary tree?**
    a. In an in-order traversal, the left subtree is visited first, followed by the current node, and then the right subtree. This traversal visits the nodes in ascending order in a binary search tree.
60. **What is a post-order traversal of a binary tree?**
    a. In a post-order traversal, the left subtree is visited first, followed by the right subtree, and then the current node. This traversal is commonly used to delete nodes from a binary search tree.
61. **What is a pre-order traversal of a binary tree?**
    a. In a pre-order traversal, the current node is visited first, followed by the left subtree and then the right subtree. This traversal is useful for creating a copy of the tree.
62. **What is the difference between BFS (Breadth-First Search) and DFS (Depth-First Search) in a tree?**
    a. In a tree, both BFS and DFS visit each node exactly once. BFS explores the tree level by level, while DFS explores as deep as possible along each branch before backtracking.
63. **What is an adjacency matrix?**
    a. An adjacency matrix is a two-dimensional array that represents a graph. It indicates the presence or absence of an edge between two vertices using a 0 or 1.
64. **What is an adjacency list?**
    a. An adjacency list is a collection of linked lists or arrays used to represent a graph. Each vertex has a list of its neighboring vertices.
65. **What is a heap data structure?**
    a. A heap is a complete binary tree where each node is greater than or equal to (in a max heap) or less than or equal to (in a min heap) its child nodes. It is commonly used for efficient priority queue operations.
66. **What is the difference between a stack and a queue?**
    a. A stack follows the Last-In-First-Out (LIFO) principle, where the last element inserted is the first one to be removed. A queue follows the First-In-First-Out (FIFO) principle, where the first element inserted is the first one to be removed.
67. **What is the difference between a stack and a linked list?**
    a. A stack is an abstract data type that can be implemented using various data structures, including a linked list. A linked list is a linear data structure that consists of nodes, each containing a reference to the next node.
68. **What is the difference between a queue and a linked list?**

a. A queue is an abstract data type that can be implemented using various data structures, including a linked list. A linked list is a linear data structure that consists of nodes, each containing a reference to the next node.

69. **What is a hash table?**
    a. A hash table, also known as a hash map, is a data structure that uses a hash function to map keys to values. It provides fast insertion, deletion, and lookup operations.
70. **What is a collision in a hash table?**
    a. A collision occurs in a hash table when two or more keys map to the same index in the underlying array. Collision resolution techniques are used to handle such cases.
71. **What are some collision resolution techniques used in hash tables?**
    a. Common collision resolution techniques include chaining (using linked lists or arrays to store multiple values at the same index) and open addressing (probing for an alternative index when a collision occurs).
72. **What is a heapify operation in a heap?**
    a. Heapify is an operation that reorders the elements in a heap to maintain the heap property (e.g., max heap or min heap). It ensures that the parent node is greater (or smaller) than its children.
73. **What is the time complexity of heapify operation in a heap?**
    a. The time complexity of the heapify operation in a heap is O(log n), where n is the number of elements in the heap.
74. **What is a disjoint set data structure?**
    a. A disjoint set data structure is a data structure that keeps track of a partitioning of a set into disjoint subsets. It supports efficient operations to merge sets and determine whether elements belong to the same set.
75. **What is the time complexity of the union-find operation in a disjoint set data structure?**
    a. The time complexity of the union-find operation in a disjoint set data structure is typically O(log n), where n is the number of elements.
76. **What is the difference between a doubly linked list and a singly linked list?**
    a. In a doubly linked list, each node contains references to both the next and previous nodes, allowing traversal in both directions. In a singly linked list, each node only has a reference to the next node.
77. **What is a circular linked list?**
    a. A circular linked list is a linked list where the last node points back to the first node, creating a loop. It can be singly or doubly linked.
78. **What is a self-loop in a linked list?**
    a. A self-loop occurs in a linked list when a node points to itself, creating a loop within the list.
79. **What is the time complexity of inserting an element at the beginning of a linked list?**

a. The time complexity of inserting an element at the beginning of a linked list is O(1), as it only involves updating a few references.

80. **What is the time complexity of searching for an element in a linked list?**
    a. The time complexity of searching for an element in a linked list is O(n), where n is the number of elements in the list.
81. **What is the time complexity of deleting an element from a linked list?**
    a. The time complexity of deleting an element from a linked list depends on the position of the element. For deletion at the beginning, it is O(1), while for deletion at the end or a specific position, it is O(n).
82. **What is a self-balancing binary search tree rotation?**
    a. A rotation is an operation performed on a self-balancing binary search tree to maintain or restore balance. It involves rearranging nodes to restructure the tree while preserving the order of elements.
83. **What is a B-tree?**
    a. A B-tree is a self-balancing search tree that maintains sorted data and allows efficient insertions, deletions, and searches. It is commonly used in databases and file systems.
84. **What is the time complexity of searching in a B-tree?**
    a. The time complexity of searching in a balanced B-tree is O(log n), where n is the number of elements in the tree.
85. **What is a priority queue?**
    a. A priority queue is an abstract data type that stores elements with associated priorities and allows retrieval of the highest-priority element. It can be implemented using various data structures, such as heaps or binary search trees.
86. **What is the difference between a singly linked list and a doubly linked list?**
    a. In a singly linked list, each node contains a reference to the next node, while in a doubly linked list, each node contains references to both the next and previous nodes.
87. **What is the time complexity of inserting an element at the end of a linked list?**
    a. The time complexity of inserting an element at the end of a linked list is O(1) if there is a reference to the tail, and O(n) otherwise, as it requires traversing the entire list.
88. **What is the time complexity of reversing a linked list?**
    a. The time complexity of reversing a linked list is O(n), where n is the number of elements in the list, as each node needs to be visited once.
89. **What is a hash function?**
    a. A hash function is a function that takes an input (such as a key) and computes a fixed-size value (hash code or hash value) that represents the input. It is used in hashing-based data structures like hash tables.
90. **What are the characteristics of a good hash function?**
    a. A good hash function should produce a uniform distribution of hash codes, minimize collisions, and be computationally efficient.

91. **What is the difference between linear probing and quadratic probing?**
    a. Linear probing and quadratic probing are collision resolution techniques used in hash tables. Linear probing checks the next available slot in a linear manner, while quadratic probing checks slots based on a quadratic function.
92. **What is a sparse matrix?**
    a. A sparse matrix is a matrix that contains a significant number of zero elements compared to the total number of elements. It is often represented more efficiently using data structures like linked lists or hash tables.
93. **What is a graph traversal?**
    a. Graph traversal is the process of visiting all the vertices or nodes in a graph. Common traversal algorithms include depth-first search (DFS) and breadth-first search (BFS).
94. **What is the difference between a graph and a tree?**
    a. A tree is a type of graph that has no cycles, whereas a graph can have cycles. Additionally, a tree has a single root node, while a graph can have multiple disconnected components.
95. **What is the time complexity of searching in a binary search tree (BST)?**
    a. The time complexity of searching in a balanced binary search tree is O(log n), where n is the number of elements in the tree. In the worst case, an unbalanced BST can have a time complexity of O(n).
96. **What is a topological sort?**
    a. A topological sort is an ordering of the vertices of a directed graph such that for every directed edge (u, v), vertex u comes before vertex v in the ordering. It is used in scheduling and dependency resolution.
97. **What is the difference between a stack and a heap?**
    a. A stack is a region of memory used for local variables and function call information, while a heap is a region of memory used for dynamic memory allocation.
98. **What is the time complexity of inserting an element into a binary search tree (BST)?**
    a. The time complexity of inserting an element into a balanced binary search tree is O(log n), where n is the number of elements in the tree. In the worst case, an unbalanced BST can have a time complexity of O(n).
99. **What is a self-balancing binary search tree?**
    a. A self-balancing binary search tree is a binary search tree that automatically maintains balance after insertions and deletions to ensure efficient search operations. Examples include AVL tree, Red-Black tree, and Splay tree.
100. **What is the time complexity of the merge sort algorithm?**
    - The time complexity of the merge sort algorithm is O(n log n), where n is the number of elements to be sorted.

.

DATA STRUCTURES NOTES

@curious_programmer

What is a Data structure ?

Data can be arranged in a many ways, logical or mathematical arrangedment of a data is called as Data structure.

Examples : Array, linked list, Stack, Queue, Tree Graph, and many more.

What is an Algorithms ?

Sequence of steps performed on the data using efficient data structures to solve a given problem.

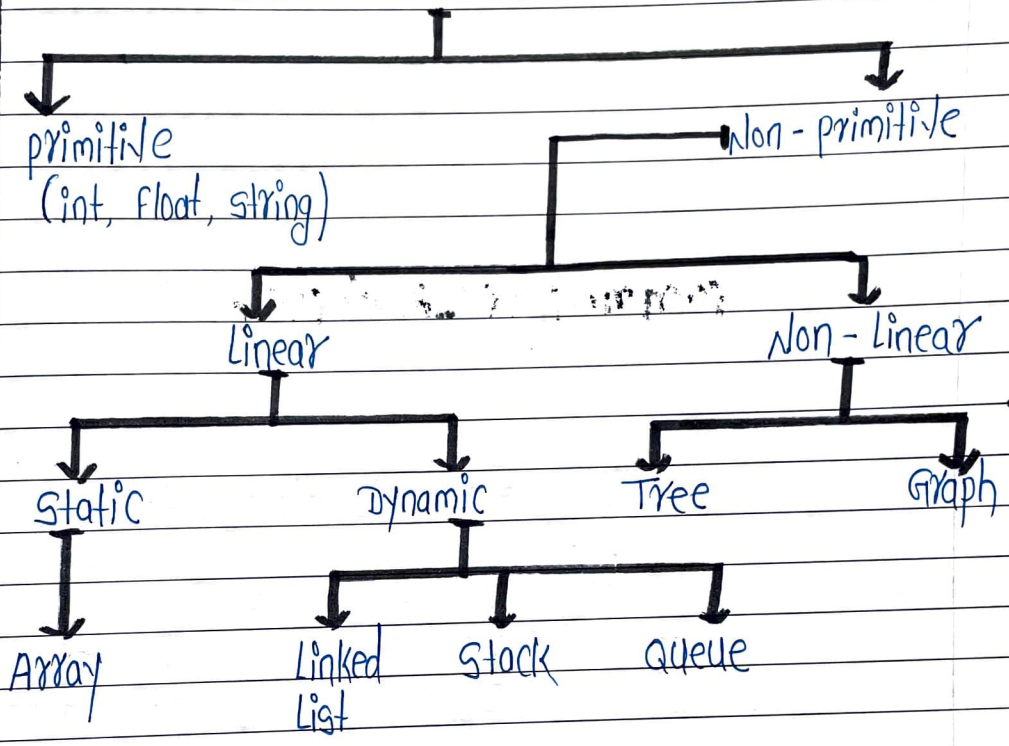
Example : Sorting an Array.

PDF FILE Uploaded on TELEGRAM
(link in Bio)

Page No.:
Date:
YOUVA

Classification of Data structures

@curious_programmer



Types of Data structures

ⓐ primitive and Non-primitive Data structure
ⓑ Static and Dynamic Data structure
ⓒ persistent and ephmeral Data structure.

Non-primitive further divided into two Types

i) Linear Data structure
ii) Non-Linear Data strudure

persistent further divided into three Types

i) partialy persistent
ii) fully persistent
iii) confluently persistent

M T W T F S S
Page No.:
Date:
YOUVA

Data structure operations :

The following four operations play a major role

(1) Traversing :
Accesing each record exactly once so that certain items in the record may be processed

(2) Searching :
Finding the location of the record with a Given key value.

(3) Inserting : Adding a new record to the structure

(4) Deleting : Removing a record from the structure

(5) Merging :
Combining the records in two different sorted files into a single sorted file

(6) Sorting :
Arranging the record in some logical order e.g Alphabetically according to some ~~number~~ Name key or in numerical order according to some number key

PDF file upload on TELEGRAM
(Link in Bio)

M T W T F S S
Page No.:
Date:
YOUVA

Searching Algorithms :

A search algorithm is a step-by-step procedure used to locate specific data among a collection of data.

Types of search algorithms with the complexity

1) Linear Search :

A linear search or sequential search is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched

$C(n) = n/2$ ← complexity of linear search

@curious_programmer

2) Binary search :

In Binary search approach the element is always searched in the middle of a portion of an array. Binary search can be implemented only on a sorted list of items.

If the elements are not sorted already, we need to sort them first.

$C(n) = \log_2 n$ ← complexity of Binary search.

PDF File Uploaden on TELEGRAM
(link in Bio)

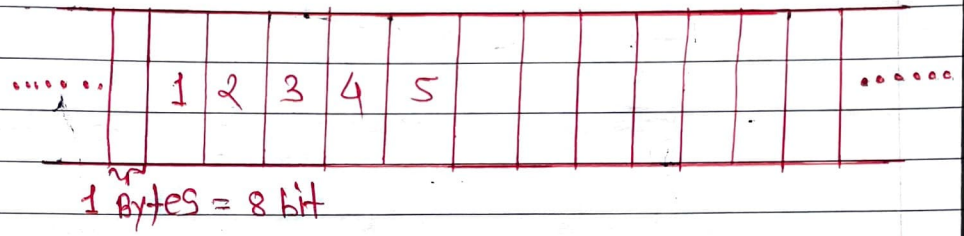
M T W T F S S
Page No.:
Date:
YOUVA

ARRAY :-

Array is a Type of linear Data structure

OR Array is a collection of more than one data but all the data items are same data types, & stored that data in a computer in a contigous memory location.

@curious-programmer



Memory is a long tap of Bytes

Types of Array :

① One Dimentional Array :

The array with only one subscript that array is called as one Dimentional Array.

Example : int a[5]; ——— subscript

② Two Dimentional Array :

The array with two subscript that array is called as Two Dimentional Array.

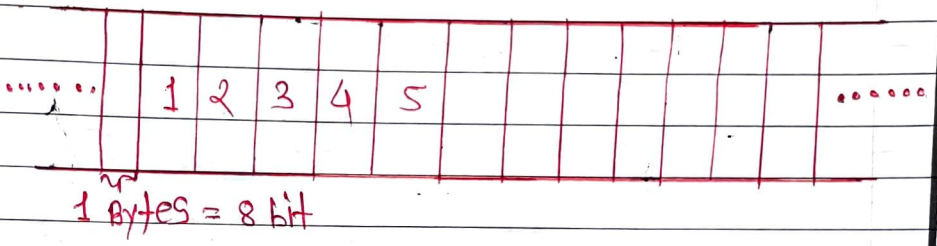
Example : int a[5][5]; ——— subscript

③ Multi-Dimentional Array : The array with more than two subscript.

ARRAY :-

Array is a Type of linear Data structure

OR Array is a collection of more than one data but all the data items are same data types, & stored that data in a computer in a contigous memory location



Memory is a long tap of Bytes

Types of Array :

① One Dimentional Array :
The array with only one subscript that array is called as one Dimentional Array.
Example : int a[5]; — subscript

② Two Dimentional Array :
The array with two subscript that array is called as Two Dimentional Array.
Example : int a[5][5]; — subscript

③ Multi-Dimentional Array : The array with more than two subscript.

PDF File uploaded on TELEGRAM (link in Bio)

M T W T F S S
Page No.:
Date:
YOUVA

LINKED LIST :-

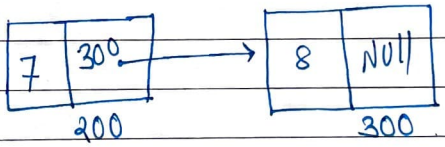
Linked list is a linear Data structure. It is also a collection of more than one data items of a disimilar data type like array but it can not stored it in contigues memory location. It can be stored randomly in a main memory.

So that linkedlist contains two part one for Data and second part for the address of the next data element

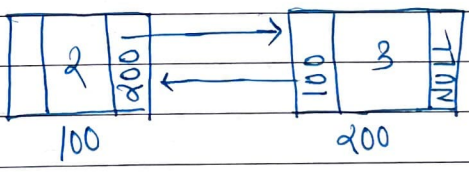
@curious_programmer

Types of linked list

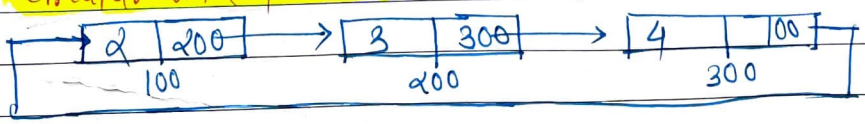
① Singly linked list :



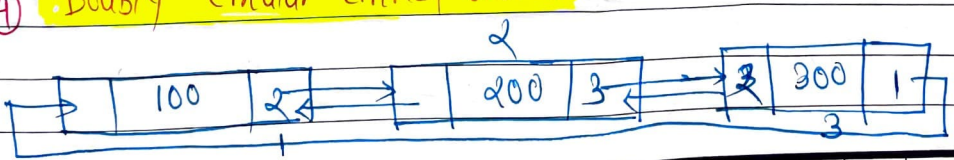
② Doubly linked list :



③ circular linked list :



④ Doubly circular linked list :

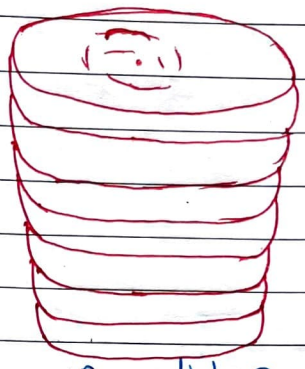


PDF file upload on TELEGRAM
(link in Bio)

M T W T F S S
Page No.:
Date:
YOUVA

STACKS

A stack is a list of elements in which an elements may be inserted or deleted only at one end called the **Top** of the stack.



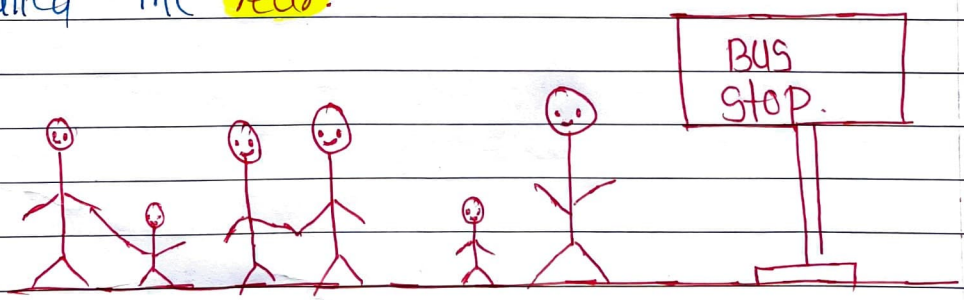
@curious_programmer

Stack of dishes

push ⟶ Insert elements into stack
Delete elements from stack ⟵ pop

QUEUES

A **Queue** is a linear list of elements in which deletions can take place only at one end called **front** and insertions can take place only at the other end called the **rear**.



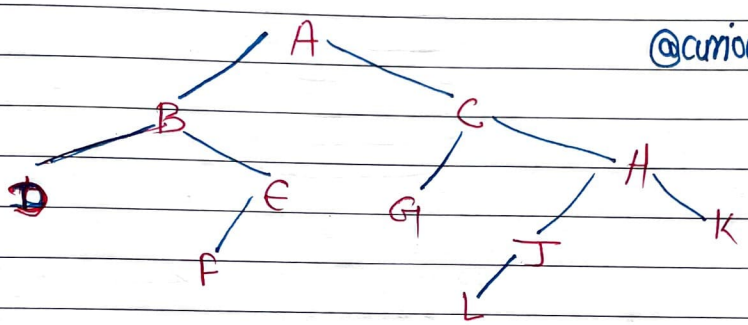
PDF file uploaded on TELEGRAM (link in Bio)

M T W T F S S
Page No.:
Date:
YOUVA

TREES

Trees are non-linear data structure where data are stored or data containing a hierarchical relationship between elements

A binary tree + is defined as a finite set of elements called nodes.



@curious_.programmer

Traversing Binary Trees

There are three ways of traversing a binary tree T with root R.

preorder

1) process the root R
2) Traverse the left subtree of R in preorder
3) Traverse the right subtree in preorder

Inorder

1) Traverse left subtree
2) process the root R
3) Traverse Right subtree

postorder

1) Traverse left subtree
2) Traverse Right subtree
3) process the Root R

PDF File uploaded on TELEGRAM (link in Bio)

M T W T F S S
Page No.:
Date:
YOUVA

GRAPH

Graph is a collection of two set V and E where,

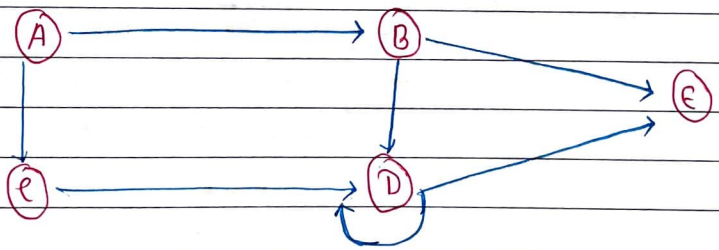
V ⟶ Vertices / Nodes.
E ⟶ Edges

@curious_.programmer

Graph is a mathematical structures that represent pair-wise relationship between objects where nodes are connected with edges.

Vertex ⟶ vertex is nothing but the data element which is also known as Node

Edge ⟶ Edge is a connection link between two vertices



Representation of the graph

(A) Adjacency matrix

(B) Adjacency list