Data Structure

Which data structures are used for BFS and DFS of a graph?

A Queue is used for BFS

A Stack is used for DFS. DFS can also be implemented using recursion.

Which Data Structure Should be used for implementing LRU cache?

We use two data structures to implement an LRU Cache.

A queue which is implemented using a doubly linked list. The maximum size of the queue will be equal to the total number of frames available (cache size). The most recently used pages will be near rear end and least recently pages will be near the front end.

A Hash with page number as key and address of the corresponding queue node as value.

If inorder traversal of a binary tree is sorted, then the binary tree is BST. The idea is to simply do inorder traversal and while traversing keep track of previous key value. If current key value is greater, then continue, else return false. See A program to check if a binary tree is BST or not for more details.

What are linear and nonlinear data Structures?

Linear: A data structure is said to be linear if its elements form a sequence or a linear list. Examples: Array. Linked List, Stacks and Queues
Non-Linear: A data structure is said to be non-linear if traversal of nodes is nonlinear in nature. Example: Graph and Trees.

What are the various operations that can be performed on different Data Structures?

Searching: Find out the location of the data item if it exists in the given collection of data items.
Sorting: Arranging the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

What is BFS/DFS for a Graph?

Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree. Unlike trees, graphs contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. Unlike trees, graphs contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.

Explain what is Skip list?

operation associates a specified key with a new value, while the delete function deletes the specified key.

What is the Difference between ArrayList vs HashMap in Java?

1) ArrayList implements List interface while HashMap implements Map interface in Java.
2) ArrayList only stores one object while HashMap stores two objects key and value.
3) Keys of HashMap must implement equals and hashCode method correctly, ArrayList doesn't have that requirement but its good to have that because contains() method of ArrayList will use equals() method to see if that object already exists or not.
4) ArrayList maintains the order of object, in which they are inserted while HashMap doesn't provide any ordering guarantee.
5) ArrayList allows duplicates but HashMap doesn't allow duplicates key though it allows duplicate values.
6) ArrayList get(index) method always gives an O(1) performance but HashMap get(key) can be O(1) in the best case and O(n) in the worst case.

What is a slow pointer and fast pointer?

Given below is a basic code snippet for moving slow and fast pointers.

```
struct node *slow_ptr, *fast_ptr;

while(!slow_ptr && !fast_ptr && fast_ptr->next != NULL) {
        slow_ptr = slow_ptr->next;  // moves one node ahead at a ti
        fast_ptr = fast_ptr->next->next;  // moves two nodes ahead
}
```

This concept can be used in cases like detecting a loop in a graph, finding the middle node of a linked list (better time complexity), flattening a linked list etc. All these examples use the idea of slow and fast pointers.

```
private static boolean isLoopPresent(Node startNode){
Node slowPointer = startNode; // Initially ptr1 is at starting loc
Node fastPointer = startNode; // Initially ptr2 is at starting loc

while(fastPointer!=null && fastPointer.getNext()!=null){ // If ptr
     slowPointer = slowPointer.getNext(); // ptr1 moving one node
     fastPointer = fastPointer.getNext().getNext(); // ptr2 moving

if(slowPointer==fastPointer) // if ptr1 and ptr2 meets, it means l
     return true;
```

How is an Array different from Linked List?

The size of the arrays is fixed, Linked Lists are Dynamic in size.

Inserting and deleting a new element in an array of elements is expensive, Whereas both insertion and deletion can easily be done in Linked Lists.

Random access is not allowed in Linked Listed.

What is a Queue, how it is different from a stack and how is it implemented?

A queue is a linear structure which follows the order is First In First Out (FIFO) to access elements. Mainly the following are basic operations on queue: Enqueue, Dequeue, Front, Rear
The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added. Both Queues and Stacks can be implemented using Arrays and Linked Lists.

What is a HashTable, and what is the average case and worst case time for each of its operations? How can we use this structure to find all anagrams in a dictionary?

A Hash Table is a data structure for mapping values to keys of arbitrary type. The Hash Table consists of a sequentially enumerated array of buckets of a certain length. We assign each possible key to a bucket by using a hash function – a function that returns an integer number (the index of a bucket) for any given key. Multiple keys can be assigned to the same bucket, so all the (key, value) pairs are stored in lists within their respective buckets.

case time complexity of O(N) (where N is the number of elements in the Hash Table), in practice we have an average time complexity of O(1).

To find all anagrams in a dictionary, we just have to group all words that have the same set of letters in them. So, if we map words to strings representing their sorted letters, we could group words into lists by using their sorted letters as a key.

The hash table stores lists mapped to strings. For each word, we add it to the list at the appropriate key, or make a new list and add it to it then. On average, for a dictionary of N words of length less or equal to L, this algorithm works with an average time complexity of O(N L log L).

What are Red-Black Trees and B-Trees? What is the best use case for each of them?

Both Red-Black Trees and B-Trees are balanced search trees that can be used on items that can have a comparison operator defined on them. They allow operations like minimum, maximum, predecessor, successor, insert, and delete, in O(log N) time (with N being the number of elements). Thus, they can be used for implementing a map, priority queue, or index of a database, to name a few examples.

Red-Black Trees are an improvement upon Binary Search Trees. Binary Search Trees use binary trees to perform the named operations, but the depth of the tree is not controlled – so operations can end up taking a lot more time than expected. Red-Black Trees solve this branch, so each branch is shorter than 2*log_base2(N).

This is the ideal structure for implementing ordered maps and priority queues. B-Trees branch into K-2K branches (for a given number K) rather than into 2, as is typical for binary trees. Other than that, they behave pretty similarly to a binary search tree. This has the advantage of reducing access operations, which is particularly useful when data is stored on secondary storage or in a remote location. This way, we can request data in larger chunks, and by the time we finish processing a previous request, our new request is ready to be handled. This structure is often used in implementing databases, since they have a lot of secondary storage access.

What are Infix, prefix, Postfix notations?

Infix notation: X + Y – Operators are written in-between their operands. This is the usual way we write expressions. An expression such as
A * (B + C) / D
Postfix notation (also known as "Reverse Polish notation"): X Y + Operators are written after their operands. The infix expression given above is equivalent to
A B C + * D/
Prefix notation (also known as "Polish notation"): + X Y Operators are written before their operands. The expressions given above are equivalent to
/ * A + B C D

Codenza