CS301-Short Notes for Midterm Exams 2021

Linked List

1. LInked List
    - Low-level data structure
    - Represent a sequence of nodes that have pointers to other nodes.
2. 3 basic kinds of linked list
    - Singly Linked List
    - Doubly Linked List
    - Circular Linked List
3. Head
    - No matter how many nodes are present in the linked list, the very first node is called head
4. Tail
    - The last node is called the tail. If there is just one node created then it is called both head and tail.

5. Singly Linked List
    - In this list, the nodes each have one pointer to the next node
6. doubly-linked list
    - In a doubly-linked list contains some data, a link to the next node and a link to the previous node.
7. A node in Singly Linked List contains what?

- Value
- Next Pointer

8. In a linked list

The first node is ... ?
The last node is ... ?

- The first node is Head
- The last node is Tail

9. How do we keep the connection to a linked list?
    - By a variable pointing to the head
10. Simple Pseudo code of Singly LinkedListNode

```
public class LinkedListNode {

public int value;
public LinkedListNode next;

public LinkedListNode(int value) {
this.value = value;
}
}
```

We could build a singly linked list like this:

```
LinkedListNode a = new LinkedListNode(5);
LinkedListNode b = new LinkedListNode(1);
LinkedListNode c = new LinkedListNode(9);

a.next = b;
b.next = c;
```

11. Simple Pseudo code of Doubly LinkedListNode

```
public class LinkedListNode {

public int value;
public LinkedListNode next;
public LinkedListNode previous;

public LinkedListNode(int value) {
this.value = value;
}
}
```

So we could build a doubly linked list like this:

```
LinkedListNode a = new LinkedListNode(5);
LinkedListNode b = new LinkedListNode(1);
LinkedListNode c = new LinkedListNode(9);

// put b after a
a.next = b;
b.previous = a;
```

```
// put c after b
b.next = c;
c.previous = b;
```

12. Time Complexity for Insert /remove from the front
    - Constant
13. Constant Time to insert/remove from the back is when?
    - Tail and Doubly LInked List
14. Constant Time to insert between node when?
    - Doubly Linked List

ADT and Stack

1. Abstraction
    - Representation of an entity with only it's most relevant attributes.
2. Encapsulation
    - Having modules whose state is accessed through its methods only.
3. Implementation
    - Code written in methods
4. Data Hiding
    - Decoupling implementation from interface.
      -Make all fields private except methods that others will use (hide state and make interface available)
      Advantage : Low Coupling (Implementation can change without affecting others)
5. Encapsulation
    - Modularity + Data Hiding
6. Modularity
    - Keep implementation and state in one module (class)
      Advantage: High Cohesion (Everything associated with the call is together)
7. Abstract Data Types (ADT)
    - a data type with values and operations that are not defined in the language by default.
8. Data Structure
    - A programming construct used to implement ADTs.
      -Mechanism for organizing the data that is the ADT is encapsulating
      -Data structures should be hidden by the API(methods) of the ADT
9. Stack
    - A Stack is a Last In First Out (LIFO) data structure. It supports two basic operations called push and pop. The push operation adds an element at the top of the stack, and the pop operation removes an element from the top of the stack.
      The Stack class extends Vector which implements the List interface.

Queues

1. What is a queue?
    - An ADT with FIFO characteristics
2. What does FIFO stand for?
    - First In First Out
3. Where does insertion occur in a queue
    - The rear

4. What are some other words that are used for the queue insertion operation?
    - Put, Add, or Enqueue
5. What are other names for the remove operation in a queue?
    - Get, Delete, or Dequeue
6. Where does removal occur?
    - At the front of the queue
7. Explain the procedure for insertion in a queue
    - If the rear has reached the end of the array, set it equal to -1 2) Increment rear, and set the value of rear equal to the item being inserted 3) Increment the number of items in the queue
8. Explain the procedure for removing an item from a queue
    - Retrieve the current value at the front of the queue into a temporary variable Increment the value of the front 3) If front is now equal to maxsize, set front equal to 0 4) Decrement the number of items in the queue 5) Return the temporary value
9. Explain how to determine if a queue is empty without using an item counter
    - By checking that the rear + 1 is equal to the front or that the front + maxsize - 1 is equal to the rear
10. Explain how to tell if a queue is full without using a counter
    - If the rear + 2 is equal to the front, or if the front + maxsize - 2 is equal to the rear.

Binary Tree

1. Binary tree
    - a structure with a unique starting node, in which each node is capable of having two child nodes, and in which a unique path exist from the root to every other node
2. Root
    - the top node of a tree structure; a node with no parent
3. Leaf node
    - a tree node that has no children
4. Level
    - the distance of a node from the root; the root is level 0
5. Height
    - the maximum level
6. Binary search tree
    - A binary tree in which the key value in any node is greater than the key value in its left child and any of its children and less than the key value in its right child in any of its children
7. keys
    - Represents a value of a node based on which a search operation is to be carried out for a node.
8. Traversing
    - Means passing through nodes in a specific order.
9. Subtree
    - Represents the descendants of a node.

BST

1. Binary Search Tree.
    - a binary tree whose nodes contain elements of some ordered type.

2. What are the methods of the BinarySearchTree class?
    - put(key, value)
    - get(key)
    - delete(key)
    - getSize()
3. Access
    - Average: Θ(log n)
    - Worst: O(n)
4. Search
    - Average: Θ(log n)
    - Worst: O(n)
5. Insertion/Deletion
    - Average: Θ(log n)
    - Worst: O(n)
6. Space
    - Worst: O(n)

Recursion

1. What is Recursion in programming?
    - In Programming, the ability for a function to call itself.
2. What is Recursion?
    - A problem solving concept which can be used with languages that support the /dynamic allocation of memory./
3. What are the 2 requirements for recursive code?
    - There must be an Anchor Point 2. Each recursive call must approach the anchor point (or base case).
4. What is the Anchor point (AKA Base Case?)
    - The stop point of a recursive function.
5. What is the advantage of Recursive programming?
    - Usually a more concise coding solution, some solutions are more easily implemented using recursion
6. What is the disadvantage of Recursive Programming?
    - Less efficient due to overhead involved with function calls

AVL tree

1. What is an AVL tree?
    - A balanced binary search tree
    - Satisfies the height-balance property: ′ For every internal node v of T, the heights of the children of v can differ by at most 1.
2. Insertion in AVL
    - Insertion is as in a BST
      Always done by expanding an external node
3. Maintaining balance
      Remember that an AVL tree needs to be balanced:
    - The height of the subtrees can differ only by 1
    - If inserting a node causes imbalance
    - You need to re-balance the tree by rotating the unbalanced subtree

4. AVL tree rotation
    - Rotations: Restructuring the tree to maintain properties.
    - Inorder traversal must produce the same results on both original and re-structured tree.
    - Types of rotations:
      -Single right rotation
      -Single left rotation
      -Double left-right rotation
      -Double right-left rotation