1. cStatic and Dynamic Arrays
2. Linked Lists (Single, Double, Circular)
3. Stack - Last In First Out
4. Queue - First In First Out
5. Binary Tree - is a tree data structure in which each node has at most two children,
6. Binary Search Tree - The left subtree contains only nodes with keys lesser than the node's key and right subtree has nodes with greater value. Inorder (^) traversal to print BST in ascending

Usage

1. Used to implement simple sorting algorithms.
2. Can be used as priority queues.
3. Used in many search applications where data are constantly entering and leaving.
4. Complete binary tree - A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, It can be represented in array . Left ((2 \* i) + 1, Right ((2 \* i) + 2) Parent (i - 1) / 2
5. A Binary Heap- is a complete Binary Tree with heap invariant. It’s a complete tree .
6. Min Heap, Max Heap - Min Binary Heap, the key at root must be minimum among all keys and reverse is in Max.
7. PriorityQueue (queue with comparator) Implemented with binary heap. The root is the first element to pop out.
8. Balanced binary tree - A balanced binary tree is a binary tree structure in which the left and right subtrees of every node differ in height by no more than 1
9. Self-Balanced Binary Tree - a self-balancing (or height-balanced) binary search tree is any node-based binary search tree that automatically keeps its height (maximal number of levels below the root) small in the face of arbitrary item insertions and deletions [ AVL Tree, red/black tree, splay tree ]
10. Tries – Each trie is a tree with 26 children representing 26 letters. Each children have another 26 and so on. A Boolean value isEndofWord decides the word exist or not.
11. AVL Tree AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.

1) Left Left Case [ rightRotate(node)]

2) Left Right Case [ rightRotate(node)]

3) Left Right Case [ node.left = leftRotate(node.left); rightRotate(node) ]

4) Right Left Case [ node.right = rightRotate(node.right); leftRotate(node) ]

Usage

1. Used in situations where frequent insertions are involved.
2. Used in Memory management subsystem of the Linux kernel to search memory regions of processes during preemption.
3. Red/Black tree : A red-black tree is a kind of self-balancing binary search tree where each node has an extra bit, and that bit is often interpreted as the color. Rules

1)Every node has a color either red or black.

2)The root of the tree is always black.

3)There are no two adjacent red nodes (A red node cannot have a red parent or red child).

4)Every path from a node (including root) to any of its descendants NULL nodes has the same number of black nodes.

Insertion

1)Insert as you would into a BST, coloring the node red.

2)If the parent of the node you just inserted was red, you have a double-red problem which you must correct.

3)Color the root node black.

A double red problem is corrected with zero or more recolorings followed by zero or one restructuring. Recolor whenever the sibling of a red node's red parent is red:

Restructure whenever the red child's red parent's sibling is black or null. There are four cases:

* Right
* Left
* Right-Left
* Left-Right

When you restructure, the root of the restructured subtree is colored black and its children are colored red.

Usage

1. As a base for data structures used in computational geometry.
2. Used in the Completely Fair Scheduler used in current Linux kernels.
3. Used in the epoll system call implementation of Linux kernel.
4. Splay tree : Splay Tree is a self - adjusted Binary Search Tree in which every operation on element rearranges the tree so that the element is placed at the root position of the tree.

Usage

1. Used to implement caches
2. Used in garbage collectors.
3. Used in data compression
4. B-tree B tree is a self-balancing search tree and contains multiple nodes which keep data in sorted order. Each node has 2 or more children and consists of multiple keys.

Usage

* 1. Used in database indexing to speed up the search.
  2. Used in file systems to implement directories.

1. N-ary tree. If a tree is a rooted tree in which each node has no more than N children, it is called N-ary tree. Trie is one of the most frequently used N-ary trees.
2. Index Priority Queue- In an Indexed Priority Queue, data is stored just like standard priority queue and along with this, the value of a data can be updated using its key. It is called “indexed” because a hash map can be used to store the index in container using the key of key-value pair input as the key of hash map.
3. Hash Tables – First hash the keys, then % with array size. A linked list is connected to each array index. Each node has a key and value.

Trees

The AVL trees are more balanced compared to Red-Black Trees, but they may cause more rotations during insertion and deletion. So if your application involves frequent insertions and deletions, then Red-Black trees should be preferred.

Tree Traversal

Inorder (^) print BST in ascending

Left ,root,right

If(node ==null)

Return

Inorder(node.left)

Process node.data

Inorder(node.right)

Preorder <. create a copy of the tree

Root,left,right

Process data

Preorder(node.left)

Preorder(node.right)

Postorder > used to delete the tree  
left,right,root

Postorder(node.left)

Postorder(node.right)

Process(node.data)

Binary Indexed Tree  A Fenwick tree or binary indexed tree is a data structure that can efficiently update elements and calculate prefix sums in a table of numbers.

Segment Tree A **segment tree** is a data structure used to store information about array segments and answer segment queries efficiently. There are two main operations performed on a segment tree:

1. **range(i, j)**: gives the sum of the array elements starting at index i and ending at index j.
2. **update(i, val)**: updates the value at index i to the val in the original array and updates the segment tree accordingly.

Both range(i, j) and update(i, val) take log(n)*log*(*n*) time, where n is the number of elements in the segment tree.

Algorithms

1. BFS and DFS, and know the difference between inorder, postorder and preorder.
2. Bitwise operations

Bit Manipulations

1. Odd or Even - The expression n&1 return value 1 or 0 depending on whether n is odd or even
2. Opposite signs. The expression x^y is negative if numbers have opposite signs.
3. -~x will add 1 to integer x. ~ of a number is – of that number+1. ~4 =-5.
4. Swap without using third variable. a=a^b; b = a^b; a= a^b;
5. Turn off k’th bit in a number – The idea is to use bitwise <<,& and ~ operators. Using the expression ~(1<<(k-1)) , we get a number with all its bits set except kth bit. If we do this result with a number n, that is n & ~(1<<(k-1)) , we get a number which has all bits same as n , except kth bit which is 0.
6. Turn on kth bit in a number. Same as above, without ~. Then we get a number with only kth bit set to 1. We do | with n and we get kth bit set.
7. Check if k’th bit is set for a number – Use the number from 6. First & it with n. n&(1<<(k-1). If the answer is non zero, then kth bit is set.
8. Toggle the k’th bit – Use the same as above , XOR with n. n^&(1<<(k-1).
9. LSB. Unset the rightmost set bit of a number- n&(n-1) . This will return 0 if n is a power of 2 as it will have only one bit set.
10. Find the position of the rightmost set bit – n& (-n) will give the position of LSB.
11. Find the absolute value of an integer . (n+mask)^mask where mask is n>>31. 0 for positive and 1 for negative

<https://medium.com/techie-delight/bit-manipulation-interview-questions-and-practice-problems-27c0e71412e7>