UNIT II - HIERARCHICAL DATA STRUCTURES

Binary Search Trees: Basics – Querying a Binary search tree – Insertion and Deletion- Red Black trees: Properties of Red-Black Trees – Rotations – Insertion – Deletion -B-Trees: Definition of B - trees – Basic operations on B-Trees – Deleting a key from a B-Tree- Heap – Heap Implementation – Disjoint Sets - Fibonacci Heaps: structure – Mergeable-heap operations- Decreasing a key and deleting a node-Bounding the maximum degree.

HEAP – HEAP IMPLEMENTATION

A Binary Heap is a Binary Tree with following properties.

It's a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.

How is Binary Heap represented?

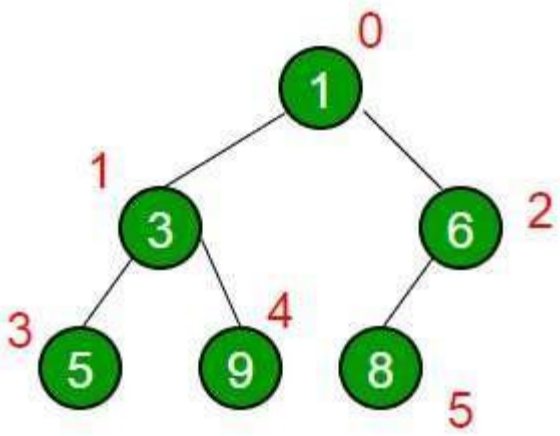
A Binary Heap is a Complete Binary Tree. A binary heap is typically represented as an array.

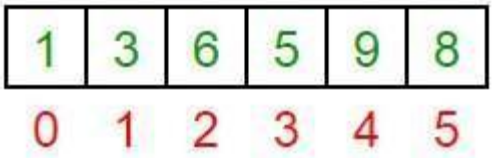
The root element will be at Arr[0].

Below table shows indexes of other nodes for the ith node, i.e., Arr[i]:

Arr[(i-1)/2]	Returns the parent node
Arr[(2*i)+1]	Returns the left child node
Arr[(2*i)+2]	Returns the right child node

The traversal method use to achieve Array representation is Level Order





Applications of Heaps:

Heap Sort: Heap Sort uses Binary Heap to sort an array in O(nLogn) time.

Priority Queue: Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in O(logn) time. Binomoial Heap and Fibonacci Heap are variations of Binary Heap. These variations perform union also efficiently.

Graph Algorithms: The priority queues are especially used in Graph Algorithms like Dijkstra's Shortest Path and Prim's Minimum Spanning Tree.

Many problems can be efficiently solved using Heaps. See following for example.
K'th Largest Element in an array.
Sort an almost sorted array/
Merge K Sorted Arrays.

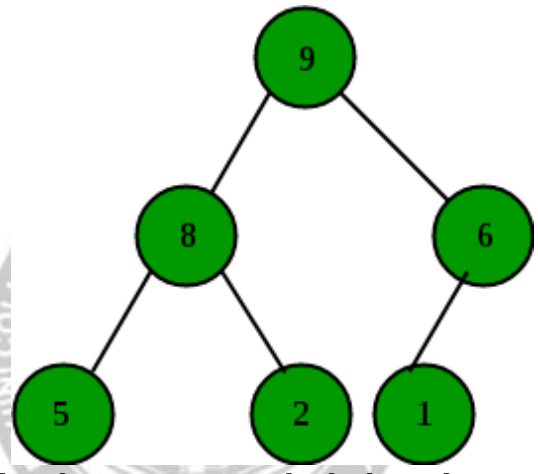
Operations on Min Heap:

- **getMin():** It returns the root element of Min Heap. Time Complexity of this operation is O(1).
- **extractMin():** Removes the minimum element from MinHeap. Time Complexity of this Operation is O(Logn) as this operation needs to maintain the heap property (by calling heapify()) after removing root.
- **decreaseKey():** Decreases value of key. The time complexity of this operation is O(Logn). If the decreased key value of a node is greater than the parent of the node,

then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

- **insert():** Inserting a new key takes O(Logn) time. We add a new key at the end of the tree. IF new key is greater than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.
- **delete():** Deleting a key also takes O(Logn) time. We replace the key to be deleted with minum infinite by calling decreaseKey(). After decreaseKey(), the minus infinite value must reach root, so we call extractMin() to remove the key.

Below is the implementation of basic heap operations.



A max-heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node. Mapping the elements of a heap into an array is trivial: if a node is stored an index k, then its left child is stored at index 2k+1 and its right child at index 2k+2.

How is Max Heap represented?

A-Max Heap is a Complete Binary Tree. A-Max heap is typically represented as an array. The root element will be at Arr[0]. Below table shows indexes of other nodes for the ith node, i.e., Arr[i]:

Arr[(i-1)/2] Returns the parent node. Arr[(2*i)+1] Returns the left child node.

Arr[(2*i)+2] Returns the right child node.

Operations on Max Heap are as follows:

- **getMax():** It returns the root element of Max Heap. The Time Complexity of this operation is O(1).
- **extractMax():** Removes the maximum element from MaxHeap. The Time Complexity of this Operation is O(Log n) as this operation needs to maintain the heap property by calling the heapify() method after removing the root.

- **insert():** Inserting a new key takes O(Log n) time. We add a new key at the end of the tree. If the new key is smaller than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

Disjoint Sets

Disjoint sets are those sets whose intersection with each other results in a null set. In Set theory, sometimes we notice that there are no common elements in two sets or we can say that the intersection of the sets is an empty set or null set. This type of set is called a disjoint set. For example, if we have X = {a, b, c} and Y = {d, e, f}, then we can say that the given two sets are disjoint since there are no common elements in these two sets X and Y. In this article, you will learn what disjoint set is, disjoint set union, Venn diagram, pairwise disjoint set, examples in detail.

Two sets are said to be disjoint when they have no common element. If a collection has two or more sets, the condition of disjointness will be the intersection of the entire collection should be empty.

Yet, a group of sets may have a null intersection without being disjoint. Moreover, while a group of fewer than two sets is trivially disjoint, since no pairs are there to compare, the intersection of a group of one set is equal to that set, which may be non-empty. For example, the three sets {11, 12}, {12, 13}, and {11, 13} have a null intersection but they are not disjoint. There are no two disjoint sets available in this group. Also, the empty family of sets is pairwise disjoint.

Consider an example, {1, 2, 3} and {4, 5, 6} are disjoint sets.Two sets A and B are disjoint sets if the intersection of two sets is a null set or an empty set. In other words, the intersection of a set is empty.

i.e. $A \cap B = \phi$

Properties of Intersection:

- Commutative: $A \cap B = B \cap A$
- Associative: $A \cap (B \cap C) = (A \cap B) \cap C$
- $A \cap \emptyset = \emptyset$
- $A \cap B \subseteq A$
- $A \cap A = A$
- $A \subseteq B$ if and only if $A \cap B = A$

Disjoint Set Union

A disjoint set union is a binary operation on two sets. The elements of any disjoint union can be described in terms of ordered pairs as (x, j), where j is the index that represents

the origin of the element x. With the help of this operation, we can join all the different (distinct) elements of a pair of sets.

A disjoint union may indicate one of two conditions. Most commonly, it may intend the union of two or more sets that are disjoint. Else if they are disjoint, then their disjoint union may be produced by adjusting the sets to obtain them disjoint before forming the union of the altered sets. For example, two sets may be presented as a disjoint set by exchanging each element by an ordered pair of the element and a binary value symbolising whether it refers to the first or second set. For groups of more than two sets, one may likewise substitute each element by an ordered pair of the element and the list of the set that contains it.

The disjoint union is denoted as X U* Y = (X x {0}) U (Y x {1}) = X* U Y*
Assume that, The disjoint union of sets X = (a, b, c, d) and Y = (e, f, g, h) is as follows:
X* = { (a, 0), (b, 0), (c, 0), (d, 0) } and Y* = { (e, 1), (f, 1), (g, 1), (h, 1) }
Then,

X U* Y = X* U Y*

Therefore, the disjoint union set is { (a, 0), (b, 0), (c, 0), (d, 0), (e, 1), (f, 1), (g, 1), (h, 1) }

Pairwise Disjoint sets

We can proceed with the definition of a disjoint set to any group of sets. A collection of sets is pairwise disjoint if any two sets in the collection are disjoint. It is also known as mutually disjoint sets.

Let P be the set of any collection of sets and A and B.
i.e. A, B ∈ P. Then, P is known as pairwise disjoint if and only if A ≠ B. Therefore, A ∩ B = ϕ

Examples:

P = { {1}, {2, 3}, {4, 5, 6} } is a pairwise disjoint set.

P = { {1, 2}, {2, 3} } is not pairwise disjoint, since we have 2 as the common element in two sets.

Are Two Null Sets Disjointed?

We know that two sets are disjoint if they don't have any common elements in the set. When we take the intersection of two empty sets, the resultant set is also an empty set. One can easily prove that only the empty sets are disjoint from itself. The following theorem shows that an empty set is disjoint with itself.

Theorem:

The empty set is disjoint with itself.

Ø ∩ Ø = Ø

Difference between Joint and Disjoint Set

Consider two sets X and Y.

Assume that both the sets X and Y are non-empty sets. Thus, X ⋂ Y is also a non-empty set, the sets are called joint set. In case, if X ⋂ Y results in an empty set, then it is called the disjoint set.

For example: X = {1, 5, 7} and Y = {3, 5, 6}

X ⋂ Y = {5}

Hence, X and Y are joint sets. In case, if

X = {1, 5, 7} and Y = {2, 4, 6}

X ⋂ Y = Ø

Therefore, X and Y are disjoint sets.

Disjoint Sets Examples

Question 1: Show that the given two sets are disjoint sets. A = {5, 9, 12, 14} B = {3, 6}

Solution:

Given: A = {5, 9, 12, 14}, B = {3, 6}

A ⋂ B = {5, 9, 12, 14} ⋂ {3, 6}

Here, set A and B do not have any common element

That is, A ⋂ B = { }

The intersection of set A and set B gives an empty set. Hence, the sets A and B are disjoint sets.

Question 2: Draw a disjoint set Venn diagram that represents the given two sets X = {a, b, c, d, f} and Y = {e, g, h, i}

Solution:

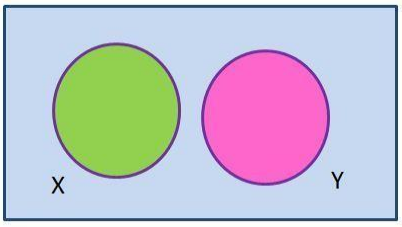
Given: X = {a, b, c, d, f}, Y = {e, g, h, i}

In the given problem, we don't have a common factor.

Therefore, the given sets are disjoint.

i.e. A ⋂ B = { }

The disjoint set Venn diagram is represented as:



The Venn diagram clearly shows that the given sets are disjoint sets.