DR. RaedIbraheemHamed
Associate Professor of Information Systems,
College of Science and Technology, University of Human Development
Sulaymaniyah, Iraq,

2015 – 2016

Java Program to Implement Sorted Array

This Java program is to Implement Sorted array. A sorted array is an array data structure in which each element is sorted in numerical, alphabetical, or some other order, and placed at equally spaced addresses in computer memory.

…………………………………………………………………………………….

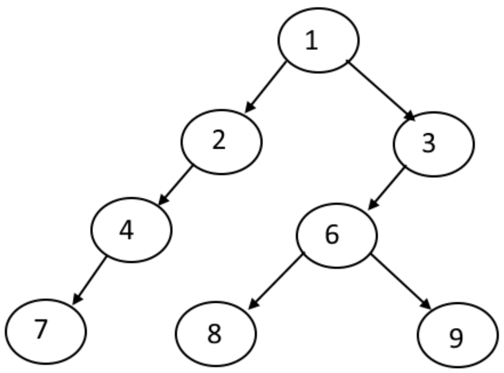
```
importjava.util.Arrays;

public class SortedArray<T>
{
private T[] array;

public SortedArray(T[] array)
   {
this.array = array;
   }

public void sort()
   {
Arrays.sort(array);
   }

public T[] getArray()
   {
return array;
   }

public static void main(String...arg)
   {
Integer[] inums = {10,9,8,7,6, 2, 1, 3, 999};
Float[] fnums = {23.9f,5.5f,10.8f,2.5f,82.0f};
Double[] dnums = {12.5,244.92,1.9,98.3,35.2};
      String[] strings = {"banana","pineapple","apple","mango","jackfruit"};
```

```
System.out.println("The Values Before sorting");
System.out.println();

System.out.println("Integer Values");
for (inti = 0; i<inums.length; i++)
System.out.print(inums[i] + "\t");

System.out.println();
System.out.println("Floating Values");
for (inti = 0; i<fnums.length; i++)
System.out.print(fnums[i] + "\t");

System.out.println();
System.out.println("Double Values");

for (inti = 0; i<dnums.length; i++)
System.out.print(dnums[i] + "\t");

System.out.println();
System.out.println("String Values");

for (inti = 0; i<strings.length; i++)
System.out.print(strings[i] + "\t");

        SortedArray<Integer> integer = new SortedArray<Integer>(inums);
        SortedArray<Float> floating = new SortedArray<Float>(fnums);
        SortedArray<Double> doubles = new SortedArray<Double>(dnums);
        SortedArray<String> string = new SortedArray<String>(strings);

integer.sort();
floating.sort();
doubles.sort();
string.sort();

inums = integer.getArray();
fnums = floating.getArray();
dnums = doubles.getArray();
strings = string.getArray();

System.out.println();
System.out.println("The Values After sorting");
System.out.println();
System.out.println("Integer Values");
for (inti = 0; i<inums.length; i++)
System.out.print(inums[i] + "\t");
System.out.println();
System.out.println("Floating Values");
for (inti = 0; i<fnums.length; i++)
System.out.print(fnums[i] + "\t");

System.out.println();
System.out.println("Double Values");
for (inti = 0; i<dnums.length; i++)
System.out.print(dnums[i] + "\t");

System.out.println();
System.out.println("String Values");
for (inti = 0; i<strings.length; i++)
System.out.print(strings[i] + "\t");
    }
}
```

…………………………………………………………………………………

………………………………………………………………………………

In a Binary Tree, Check if Two nodes has the same parent or are siblings



Node 1 and 2 are Siblings?? False
Node 2 and 3 are Siblings?? True
Node 4 and 3 are Siblings?? False
Node 4 and 6 are Siblings?? False
Node 8 and 9 are Siblings?? True

Approach:

- Given, root, Node x, Node y.
- Check if x and y are childs of root. (*root.left==x && root.right==y)* ||*root.left==y && root.right==x*)
- if yes then return true.
- Else make a recursive call to root.left and root.right

…………………………………………..

Objective: In a Binary Tree, Check if Two nodes has the same parent or are siblings

Input: A binary tree and two nodes

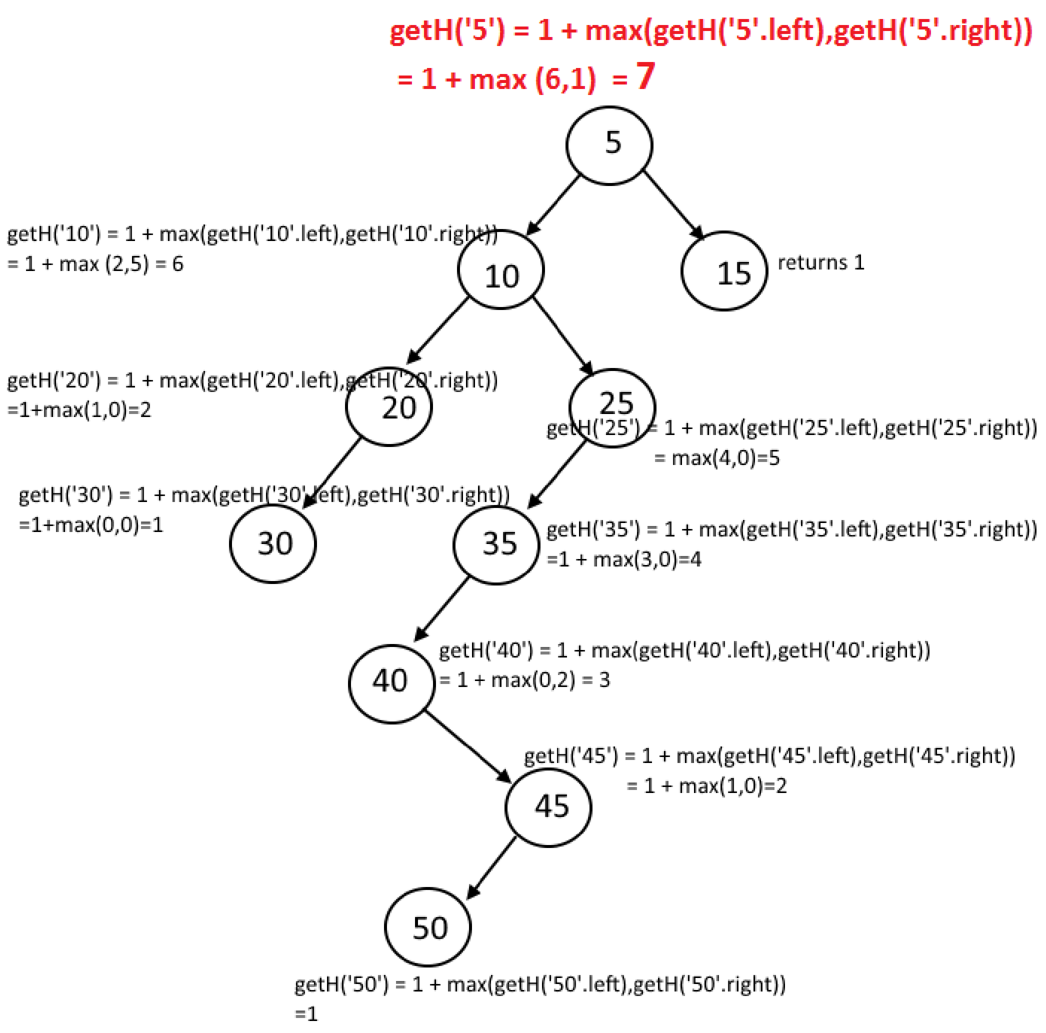
```
	public class CheckIfTwoNodesAreSiblings {
		publicbooleansameParents(Node root, Node x, Node y){
			if(root==null) return false;
			return ((root.left==x &&root.right==y) ||root.left==y 
&&root.right==x ||sameParents(root.left,x,y) || sameParents(root.right,x,y));
		}
		public static void main (String[] args) throws java.lang.Exception
		{
			Node root = new Node(1);
			Node x1 = new Node(2);
			Node y1 = new Node(3);
			root.left = x1;
			root.right = y1;
			root.left.left = new Node(4);
			root.right.left = new Node(6);
			Node x2 = new Node(7);
			Node y2 = new Node(9);
			root.right.left.left = new Node(8);
			root.right.left.right = y2;
			root.left.left.left = x2;
			CheckIfTwoNodesAreSiblingsi  = new 
CheckIfTwoNodesAreSiblings();
			System.out.println("Node " + x1.data + " and Node " + y1.data + " 
are siblings??? " + i.sameParents(root, x1, y1));
			System.out.println("Node " + x2.data + " and Node " + y2.data + " 
are siblings??? " + i.sameParents(root, x2, y2));
		}
}
class Node{
        int data;
        Node left;
        Node right;
        public Node(int data){
                this.data = data;
                this.left = null;
                this.right = null;
        }
}
```

…………………………………………………………

Output:

```
Node 2 and Node 3 are siblings??? true

Node 7 and Node 9 are siblings??? false
```

……………………………………………………………………………………
……………………………………………………………………………………………
….


getH('5') = 1 + max(getH('5'.left),getH('5'.right))
= 1 + max (6,1) = 7
5
getH('10') = 1 + max(getH('10'.left),getH('10'.right))
= 1 + max (2,5) = 6
10
15
returns 1
getH('20') = 1 + max(getH('20'.left),getH('20'.right))
=1+max(1,0)=2
20
25
getH('25') = 1 + max(getH('25'.left),getH('25'.right))
= max(4,0)=5
getH('30') = 1 + max(getH('30'.left),getH('30'.right))
=1+max(0,0)=1
30
35
getH('35') = 1 + max(getH('35'.left),getH('35'.right))
=1 + max(3,0)=4
40
getH('40') = 1 + max(getH('40'.left),getH('40'.right))
= 1 + max(0,2) = 3
getH('45') = 1 + max(getH('45'.left),getH('45'.right))
= 1 + max(1,0)=2
45
50
getH('50') = 1 + max(getH('50'.left),getH('50'.right))
=1

………

Find the Height of a Binary Tree

Objective: Given a binary tree, find the height of it

Input: A Binary Tree

Output: Height of a binary tree

```
public class TreeHeight {
	public int treeHeight(Node root){
		if(root==null)return 0;
		return (1+
Math.max(treeHeight(root.left),treeHeight(root.right)));
	}
	public static void main (String[] args) throws
java.lang.Exception
	{
		Node root = new Node(5);
		root.left = new Node(10);
		root.right = new Node(15);
		root.left.left = new Node(20);
		root.left.right = new Node(25);
		root.left.left.left =new Node(30);
		root.left.right.left = new Node(35);
		root.left.right.left.left = new Node(40);
		root.left.right.left.left.right = new Node(45);
			root.left.right.left.left.right.left = new
Node(50);
		TreeHeight i  = new TreeHeight();
		System.out.println(i.treeHeight(root));
	}
}
class Node{
	int data;
	Node left;
	Node right;
	public Node(int data){
```

```
                this.data = data;
                this.left = null;
                this.right = null;
        }
}
```

Output:

```
Height of the Tree is 7
```

……………………………………………………………

Find the Height of a tree without Recursion

Objective: - Find the Height of a tree without Recursion.
In our earlier post "Height of tree" we had used recursion to find it. In this post we will see how to find it without using recursion.

Approach:

1. Approach is quite similar to ***Level Order Traversal*** which uses ***Queue***.
2. Take int ***height*** =0.
3. Here we will use NULL as a marker at every level, so whenever we encounter null, we will increment the height by 1.
4. First add root to the ***Queue*** and add NULL as well as its marker.
5. Extract a node from Queue.
6. Check it is NULL, it means either we have reached to the end of a level OR entire tree is traversed.
7. So before adding null as marker for the next level, check if queue is empty, which means we have traveled all the levels and if not empty then add NULL as marker and increase the height by 1.
8. If Extracted node in Step 6, is NOT NULL add the children of extracted node to the Queue.
9. Repeat Steps from 5 to 8 until Queue is Empty.
10. See the Code for better explanation.

Code:

```
import java.util.LinkedList;
import java.util.Queue;

public class T_TreeHeightWithOutrecursion {

	// use NULL as a marker at every level, so whenever we encounter null, we
	// will increment the height by 1
	public int getHeight(Node root) {
		Queue<Node> q = new LinkedList<Node>();
		int height = 0;
		// add root to the queue
		q.add(root);
		// add null as marker
		q.add(null);
		while (q.isEmpty() == false) {
			Node n = q.remove();
			// check if n is null, if yes, we have reached to the end of the
```

```
                                        // current level, increment the height by 1, and add the
another
                                        // null as marker for next level
                                        if (n == null) {
                                                // before adding null, check if queue is empty,
which means we
                                                // have traveled all the levels
                                                if(!q.isEmpty()){
                                                        q.add(null);
                                                }
                                                height++;
                                        }else{
                                                // else add the children of extracted node.
                                                if (n.left != null) {
                                                        q.add(n.left);
                                                }
                                                if (n.right != null) {
                                                        q.add(n.right);
                                                }
                                        }
                                }
                                return height;
                }

                public static void main(String[] args) {
                                Node root = new Node(1);
                                root.left = new Node(2);
                                root.right = new Node(3);
                                root.left.left = new Node(4);
                                root.left.right = new Node(5);
                                root.left.left.right = new Node(8);

                                T_TreeHeightWithOutrecursion i = new
T_TreeHeightWithOutrecursion();
                                System.out.println("Tree Height " + i.getHeight(root));

                }
}

class Node {
                int data;
                Node left;
                Node right;

                public Node(int data) {
                                this.data = data;
```

```
            }
}



Output:

Tree Height 4
```

……..

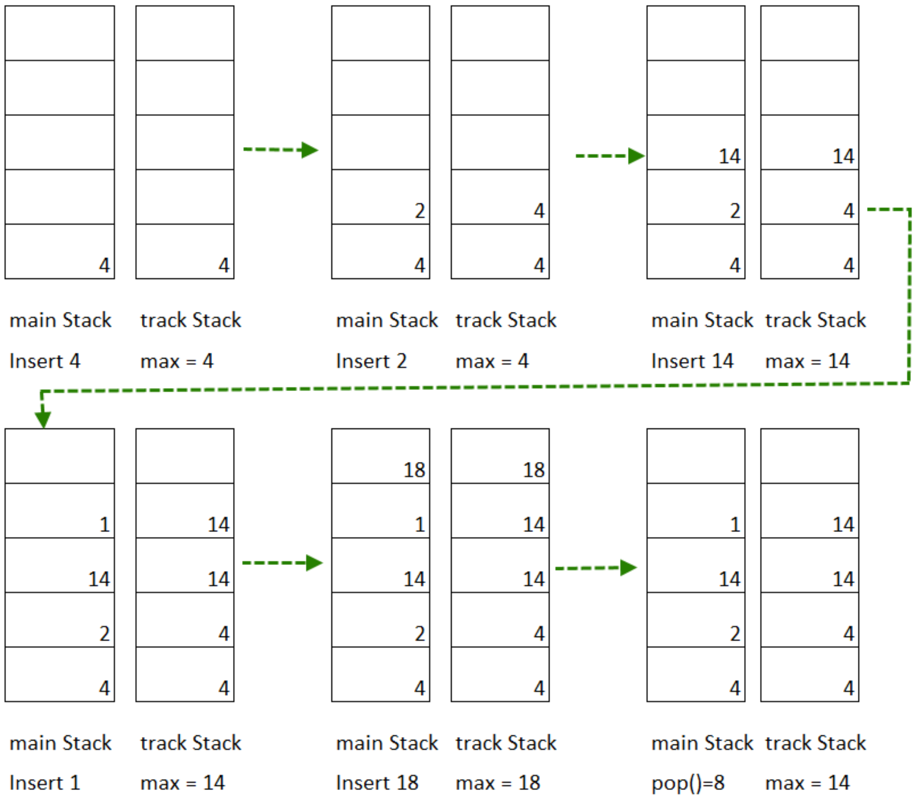
Track the Maximum Element in a Stack.

Objective: In a Stack, keep track of maximum value in it. It might be the top element in the stack but once it is poped out, the maximum value should be from the rest of the elements in the stack.

Approach:

- Create another another Stack(call it as ***track***) which will keep track of maximum in the given Stack(call it as***main***).
- When you insert an element in the ***main stack*** for the first time (means it is empty), insert it in the ***track Stack*** as well.
- Now onwards when you insert a new element(say it is ***x***) in the ***main Stack***, peek() the element from the track Stack (say it is '***a***'). Compare ***x and a*** and which ever is greater, insert it into ***track Stack***.
- When you pop the element from the ***main stack***, pop from the ***track Stack*** as well
- So to get to know the ***maximum*** element in the ***main Stack***, **peek the element in the track Stack**. .

See Example below.



Track the Maximum

THE PROGRAM

```
import java.util.Stack;

public class TrackMaxInStack {

        // objective here is to keep track of maximum value in a stack of integers
        // create another another Stack which will keep track of maximum
        Stack<Integer> main = new Stack<>();
        Stack<Integer> track = new Stack<>();

        public void insert(int x) {
                if (main.isEmpty()) { // if stack is empty, insert the number in both
                                                        // stacks
                        main.add(x);
                        track.add(x);
                } else {
                        // check if number in Stack(track) is bigger than x
```

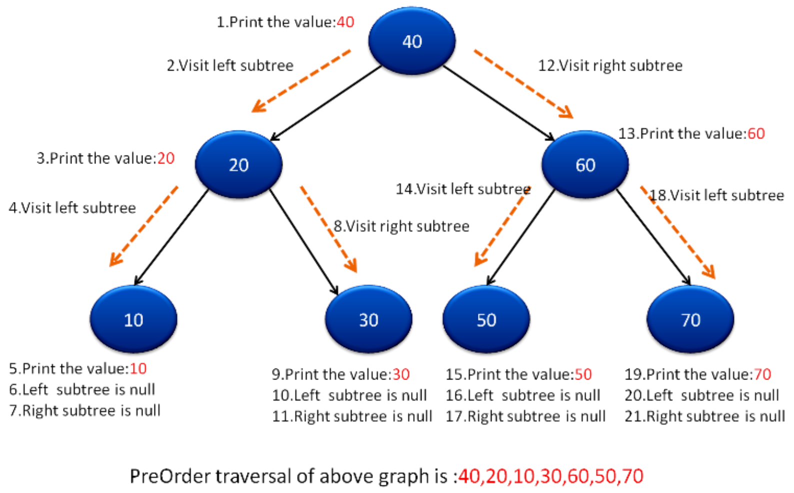
```
                    // which ever is bigger, insert it into Stack

                    int a = track.peek();
                    track.add(Math.max(a, x));
                    main.add(x); // insert it into main stack.
            }
    }

    public int remove() {
            if (!main.isEmpty()) { // pop the top elements
                    track.pop();
                    return main.pop();
            }
            return 0;
    }

    public int getMax() {
            return track.peek();
    }

    public static void main(String[] args) {
            TrackMaxInStack i = new TrackMaxInStack();
            i.insert(5);
            i.insert(6);
            i.insert(8);
            i.insert(1);
            i.insert(9);
            System.out.println("Max Element is " + i.getMax());
            System.out.println("Removing Element " + i.remove());
            System.out.println("Max Element is " + i.getMax());
    }

}
```

…………………………………………………………………………………

```
Max Element is 9

Removing Element 9

Max Element is 8


Process completed.
```

……………………………………………………………………………………………
…..…………………………….…pre- order…………………………………………..…..

Recursive solution:
Recursive solution is very straight forward. Below diagram will make you understand recursion better.



java program for Pre-Order traversal:

……………………………………………………………………….

```
package org.arpit.java2blog;
import java.util.Stack;
public class BinaryTree {
 public static class TreeNode
 {
  int data;
  TreeNode left;
  TreeNode right;
  TreeNode(int data)
  {
   this.data=data;
  }
 }
        // Recursive Solution
 public void preorder(TreeNode root) {
    if(root !=  null) {
   //Visit the node-Printing the node data
      System.out.printf("%d ",root.data);
      preorder(root.left);
```

```
     preorder(root.right);
   }
 }
 // Iterative solution
 public void preorderIter(TreeNode root) {
        if(root == null)
            return;
        Stack<TreeNode> stack = new Stack<TreeNode>();
        stack.push(root);
        while(!stack.empty()){
            TreeNode n = stack.pop();
            System.out.printf("%d ",n.data);
            if(n.right != null){
                stack.push(n.right);
            }
            if(n.left != null){
                stack.push(n.left);
            }
        }
    }
public static void main(String[] args)
{
 BinaryTree bi=new BinaryTree();
 // Creating a binary tree
 TreeNode rootNode=createBinaryTree();
 System.out.println("Using Recursive solution:");

 bi.preorder(rootNode);
 System.out.println();
 System.out.println("-------------------------");
 System.out.println("Using Iterative solution:");
 bi.preorderIter(rootNode);
}
public static TreeNode createBinaryTree()
{
 TreeNode rootNode =new TreeNode(40);
 TreeNode node20=new TreeNode(20);
 TreeNode node10=new TreeNode(10);
 TreeNode node30=new TreeNode(30);
 TreeNode node60=new TreeNode(60);
 TreeNode node50=new TreeNode(50);
 TreeNode node70=new TreeNode(70);

 rootNode.left=node20;
 rootNode.right=node60;
```

```
  node20.left=node10;
  node20.right=node30;

  node60.left=node50;
  node60.right=node70;

  return rootNode;
 }
}
```
……………………………………output……..……………………………………….

```
  Using Recursive solution:
40 20 10 30 60 50 70
-------------------------
  Using Iterative solution:
40 20 10 30 60 50 70
```

………………………………………………………………………..
……………………………………………………………………

Code for recursion will be:

```
public void preorder(TreeNode root) {
    if(root !=  null) {
   //Visit the node by Printing the node data
      System.out.printf("%d ",root.data);
      preorder(root.left);
      preorder(root.right);
    }
  }
```

Iterative solution:

For recursion, we use implicit stack. So here to convert recursive solution to iterative, we will use explicit stack.

Steps for iterative solution:

1. Create empty stack and pust root node to it.
2. Do the following when stack is not empty
   - Pop a node from stack and print it
   - Push right child of popped node to stack
   - Push left child of popped node to stack

We are pushing right child first, so it will be processed after left subtree as Stack is LIFO.

```
public void preorderIter(TreeNode root) {

        if(root == null)
            return;

        Stack<TreeNode> stack = new Stack<TreeNode>();
        stack.push(root);

        while(!stack.empty()){

            TreeNode n = stack.pop();
            System.out.printf("%d ",n.data);


            if(n.right != null){
                stack.push(n.right);
            }
            if(n.left != null){
                stack.push(n.left);
            }

        }

    }
```

……………………………………………………………………………………………………
………………………………………………..………………………………………………

,,,

Graph traversal algorithms: Depth First Search youtube

…………………………………………………………………………………………

Java Program to Traverse Graph using DFS

This Java program performs the DFS traversal on the given graph represented by an adjacency matrix. The DFS traversal makes use of a stack.

```
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.Stack;
```

```
public class DFS
{
   private Stack<Integer> stack;

   public DFS()
   {
      stack = new Stack<Integer>();
   }

   public void dfs(int adjacency_matrix[][], int source)
   {
      int number_of_nodes = adjacency_matrix[source].length - 1;

      int visited[] = new int[number_of_nodes + 1];
      int element = source;
      int i = source;
      System.out.print(element + "\t");
      visited[source] = 1;
      stack.push(source);

      while (!stack.isEmpty())
      {
         element = stack.peek();
         i = element;
            while (i <= number_of_nodes)
            {
               if (adjacency_matrix[element][i] == 1 && visited[i] == 0)
               {
               stack.push(i);
               visited[i] = 1;
               element = i;
               i = 1;
               System.out.print(element + "\t");
                  continue;
            }
            i++;
            }
         stack.pop();
      }
   }

   public static void main(String...arg)
   {
      int number_of_nodes, source;
      Scanner scanner = null;
         try
```

```
        {
            System.out.println("Enter the number of nodes in the graph");
          scanner = new Scanner(System.in);
          number_of_nodes = scanner.nextInt();

            int adjacency_matrix[][] = new int[number_of_nodes + 1][number_of_nodes +
1];
            System.out.println("Enter the adjacency matrix");
            for (int i = 1; i <= number_of_nodes; i++)
              for (int j = 1; j <= number_of_nodes; j++)
              adjacency_matrix[i][j] = scanner.nextInt();

            System.out.println("Enter the source for the graph");
          source = scanner.nextInt();

          System.out.println("The DFS Traversal for the graph is given by ");
          DFS dfs = new DFS();
          dfs.dfs(adjacency_matrix, source);
        }catch(InputMismatchException inputMismatch)
        {
          System.out.println("Wrong Input format");
        }
        scanner.close();
    }
}
```

……………………………………output…………………………………………..

```
Enter the number of nodes in the graph
4
Enter the adjacency matrix
0 1 0 1
0 0 1 0
0 1 0 1
0 0 0 1
Enter the source for the graph
1
The DFS Traversal for the graph is given by
1       2       3       4
```

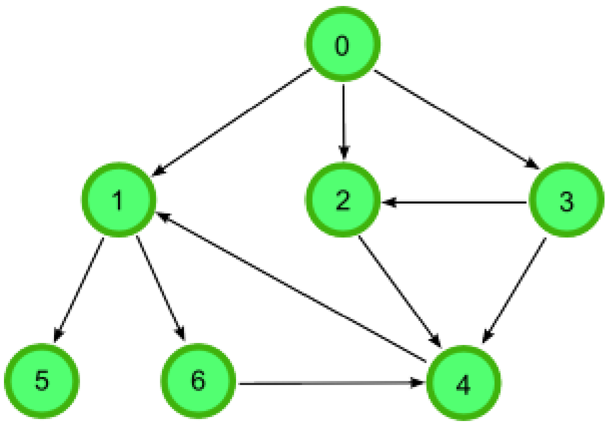
…………………………………...…………………………………………………..

…………………………………………………………………………………………………….

Depth First Search –Java implementation

1. Depth First Search – Graph example

In this example we will have a look at the *Depth First Search* (DFS) algorithm in Java. We will first store the graph below in the **adjacency list** representation.



The adjacency list representation looks as follows:

```
0: [1, 2, 3]
1: [5, 6]
2: [4]
3: [2, 4]
4: [1]
5: []
6: [4]
```

If we run DFS by hand the vertices are visited in the following order: 0, 1, 5, 6, 4, 2, 3. We expect our Java program to print this order.

2. DFS – Java implementation

2a. Recursive DFS- Java implementation

```
import java.util.ArrayList;

public class DepthFirstSearch {

  // recursive dfs
  private static void dfs_rec(ArrayList<ArrayList<Integer>> adjLists, boolean[] visited, int v){
    visited[v] = true;
    System.out.print(v + " ");
    for(int w : adjLists.get(v)){
      if(!visited[w]){
        dfs_rec(adjLists, visited, w);
      }
    }
  }

  // Usually dfs_rec() would be sufficient. However, if we don't want to pass
  // a boolean array to our function, we can use another function dfs().
// We only have to pass the adjacency list and the source node to dfs(),
// as opposed to dfs_rec(), where we have to pass the boolean array additionally.
public static void dfs(ArrayList<ArrayList<Integer>> adjLists, int s){
    int n = adjLists.size();
    boolean[] visited = new boolean[n];
    dfs_rec(adjLists, visited, s);
}


// ---------------------------------------------------------------------
// Testing our implementation
public static void main(String[] args) {

    // Create adjacency list representation
    ArrayList<ArrayList<Integer>> adjLists = new ArrayList<ArrayList<Integer>>();
    final int n = 7;

    // Add an empty adjacency list for each vertex
    for(int v=0; v<n; v++){
        adjLists.add(new ArrayList<Integer>());
    }

    // insert neighbors of vertex 0 into adjacency list for vertex 0
    adjLists.get(0).add(1);
    adjLists.get(0).add(2);
    adjLists.get(0).add(3);

    // insert neighbors of vertex 1 into adjacency list for vertex 1
    adjLists.get(1).add(5);
    adjLists.get(1).add(6);

    // insert neighbors of vertex 2 into adjacency list for vertex 2
    adjLists.get(2).add(4);

    // insert neighbors of vertex 3 into adjacency list for vertex 3
    adjLists.get(3).add(2);
    adjLists.get(3).add(4);

    // insert neighbors of vertex 4 into adjacency list for vertex 4
    adjLists.get(4).add(1);

    // insert neighbors of vertex 5 into adjacency list for vertex 5
    // -> nothing to do since vertex 5 has no neighbors

    // insert neighbors of vertex 6 into adjacency list for vertex 5
    adjLists.get(6).add(4);

    // Print vertices in the order in which they are visited by dfs()
    dfs(adjLists, 0);

}
```

```
}
```

The output of the program is: 0 1 5 6 4 2 3

2b. Iterative DFS version – Java implementation

```
import java.util.ArrayList;

import java.util.Stack;

 public class DepthFirstSearch_Iterative {

    // Use a stack for the iterative DFS version
    public static void dfs_iterative(ArrayList<ArrayList<Integer>> adj, int s){
        boolean[] visited = new boolean[adj.size()];
        Stack<Integer> st = new Stack<Integer>();
        st.push(s);
        while(!st.isEmpty()){
            int v = st.pop();
            if(!visited[v]){
                visited[v] = true;
                System.out.print(v + " ");
                // auxiliary stack to visit neighbors in the order they appear
                // in the adjacency list
                // alternatively: iterate through ArrayList in reverse order
                // but this is only to get the same output as the recursive dfs
                // otherwise, this would not be necessary
                Stack<Integer> auxStack = new Stack<Integer>();
                for(int w : adj.get(v)){
                    if(!visited[w]){
                        auxStack.push(w);
                    }
                }
                while(!auxStack.isEmpty()){
                    st.push(auxStack.pop());
                }
            }
        }
        System.out.println();
    }

      // ----------------------------------------------------------------------
    // Testing our implementation
    public static void main(String[] args) {

        // Create adjacency list representation
        ArrayList<ArrayList<Integer>> adjLists = new ArrayList<ArrayList<Integer>>();
        final int n = 7;
        // Add an empty adjacency list for each vertex
        for(int v=0; v<n; v++){
           adjLists.add(new ArrayList<Integer>());
        }

        // insert neighbors of vertex 0 into adjacency list for vertex 0
        adjLists.get(0).add(1);
        adjLists.get(0).add(2);
        adjLists.get(0).add(3);

        // insert neighbors of vertex 1 into adjacency list for vertex 1
        adjLists.get(1).add(5);
        adjLists.get(1).add(6);

        // insert neighbors of vertex 2 into adjacency list for vertex 2
        adjLists.get(2).add(4);

        // insert neighbors of vertex 3 into adjacency list for vertex 3
        adjLists.get(3).add(2);
        adjLists.get(3).add(4);

        // insert neighbors of vertex 4 into adjacency list for vertex 4
        adjLists.get(4).add(1);

        // insert neighbors of vertex 5 into adjacency list for vertex 5
        // -> nothing to do since vertex 5 has no neighbors

        // insert neighbors of vertex 6 into adjacency list for vertex 5
        adjLists.get(6).add(4);

        // Print vertices in the order in which they are visited by dfs()
        dfs_iterative(adjLists, 0);
    }
}
```

....

Again, the output of the program is: 0 1 5 6 4 2 3.

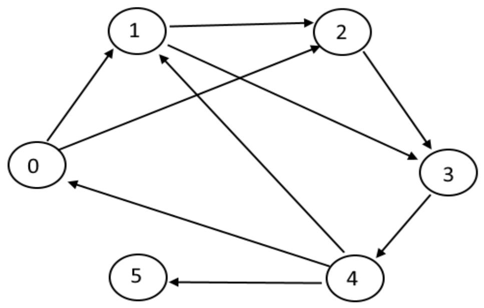
....

Breadth First Search

Approach:

1. For Graph as well we will use the Queue for performing the BFS.
2. We will use the **boolean[]** to keep a track of the nodes because unlike tree during traversal we might keep moving into the circles by visiting same nodes repeatedly.
3. In our example we are using adjacency List for the Graph Representation.

Graph



BFS starting from Node 0

0 2 1 3 4 5

Breadth-First Search (Traversal) in a Graph is quite similar to Binary Tree.

……………………………………………………………………………………………..

```
import java.util.LinkedList;
import java.util.Queue;
public class GraphBFS {
	public static void main(String args[]) {
		Graph g = new Graph(6);
		g.addEdge(0, 2);
		g.addEdge(0, 1);
		g.addEdge(1, 2);
		g.addEdge(1, 3);
		g.addEdge(3, 4);
		g.addEdge(2, 3);
		g.addEdge(4, 0);
		g.addEdge(4, 1);
		g.addEdge(4, 5);
		g.BFS(0);
	}
}
class Node {
	int dest;
	Node next;
	public Node(int d) {
		dest = d;
		next = null;
	}
}
class adjList {
	Node head;
}
class Graph {
        int V;
        adjList[] array;
        public Graph(int V) {
                this.V = V;
                array = new adjList[V]; // linked lists = number of Nodes in Graph
                for (int i = 0; i < V; i++) {
                        array[i] = new adjList();
                        array[i].head = null;
                }
        }
        public void addEdge(int src, int dest) {
                Node n = new Node(dest);
                n.next = array[src].head;
                array[src].head = n;
        }
        public void BFS(int startVertex) {
                boolean[] visited = new boolean[V];
                Queue<Integer> s = new LinkedList<Integer>();
                s.add(startVertex);
                while (s.isEmpty() == false) {
                        int n = s.poll();
                        System.out.print(" " + n);
                        visited[n] = true;
                        Node head = array[n].head;
                        while (head != null) {
                                if (visited[head.dest] == false) {
                                        s.add(head.dest);
                                        visited[head.dest] = true;
                                }
                                head = head.next;
                        }
                }
        }
}
```

....
....

Algorithm:

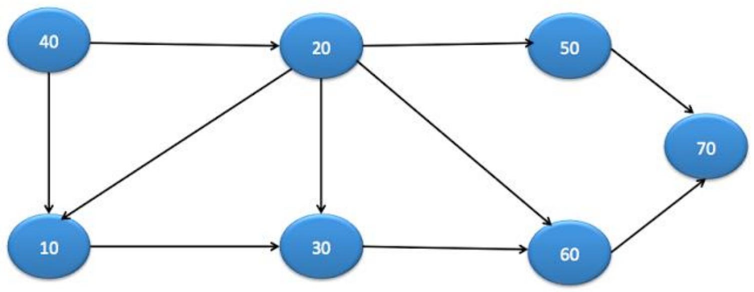
Steps for Breadth first search:

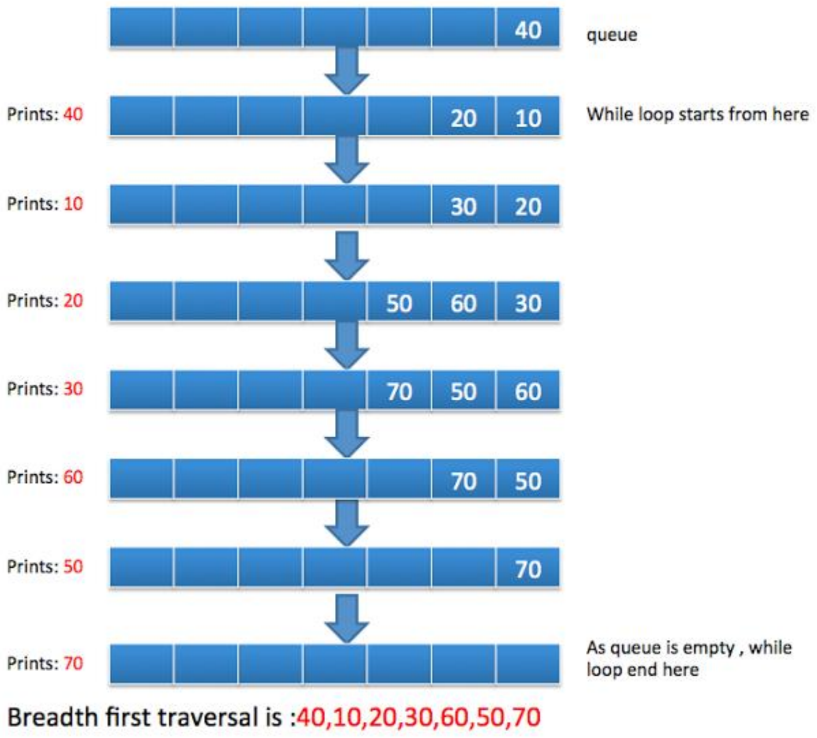
1. Create empty queue and push root node to it.

2. Do the following when queue is not empty
    - Pop a node from queue and print it.
    - Find neighbours of node with the help of adjacency matrix and check if node is already visited or not.
    - Push neighbours of node into queue if not null

Lets understand with the help of example:

Lets say graph is:




40
queue
Prints: 40
20 10
While loop starts from here
Prints: 10
30 20
Prints: 20
50 60 30
Prints: 30
70 50 60
Prints: 60
70 50
Prints: 50
70
Prints: 70
As queue is empty , while loop end here
Breadth first traversal is :40,10,20,30,60,50,70

```java
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

public class BreadthFirstSearchExample
{

 private Queue<Node> queue;
 static ArrayList<Node> nodes=new ArrayList<Node>();
 static class Node
 {
  int data;
  boolean visited;

  Node(int data)
  {
   this.data=data;

  }
 }

 public BreadthFirstSearchExample()
 {
  queue = new LinkedList<Node>();
 }

 // find neighbors of node using adjacency matrix
 // if adjacency_matrix[i][j]==1, then nodes at index i and index j are connected
 public ArrayList<Node> findNeighbours(int adjacency_matrix[][],Node x)
 {
  int nodeIndex=-1;

  ArrayList<Node> neighbours=new ArrayList<Node>();
  for (int i = 0; i < nodes.size(); i++) {
   if(nodes.get(i).equals(x))
   {
    nodeIndex=i;
    break;
   }
  }

  if(nodeIndex!=-1)
  {
   for (int j = 0; j < adjacency_matrix[nodeIndex].length; j++) {
    if(adjacency_matrix[nodeIndex][j]==1)
    {
     neighbours.add(nodes.get(j));
    }
   }
  }
  return neighbours;
 }

 public  void bfs(int adjacency_matrix[][], Node node)
 {
  queue.add(node);
  node.visited=true;
  while (!queue.isEmpty())
  {

   Node element=queue.remove();
   System.out.print(element.data + "\t");
   ArrayList<Node> neighbours=findNeighbours(adjacency_matrix,element);
   for (int i = 0; i < neighbours.size(); i++) {
    Node n=neighbours.get(i);
    if(n!=null && !n.visited)
    {
     queue.add(n);
     n.visited=true;

    }
   }

  }
 }

 public static void main(String arg[])
 {

   Node node40 =new Node(40);
   Node node10 =new Node(10);
   Node node20 =new Node(20);
   Node node30 =new Node(30);
   Node node60 =new Node(60);
   Node node50 =new Node(50);
   Node node70 =new Node(70);

   nodes.add(node40);
   nodes.add(node10);
   nodes.add(node20);
   nodes.add(node30);
   nodes.add(node60);
   nodes.add(node50);
   nodes.add(node70);
   int adjacency_matrix[][] = {
    {0,1,1,0,0,0,0},  // Node 1: 40
    {0,0,0,1,0,0,0},  // Node 2 :10
    {0,1,0,1,1,1,0},  // Node 3: 20
    {0,0,0,0,1,0,0},  // Node 4: 30
            {0,0,0,0,0,0,1},  // Node 5: 60
            {0,0,0,0,0,0,1},  // Node 6: 50
            {0,0,0,0,0,0,0},  // Node 7: 70
          };
          System.out.println("The BFS traversal of the graph is ");
          BreadthFirstSearchExample bfsExample = new BreadthFirstSearchE
xample();
          bfsExample.bfs(adjacency_matrix, node40);

        }
      }
```

When you run above program, you will get below output:

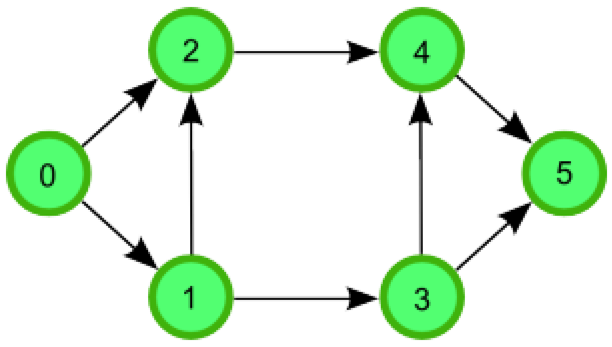
```
The BFS traversal of the graph is
40 10 20 30 60 50 70
```

....

....

Adjacency list representation of a graph

Here, the adjacency list representation of a graph. Take for example the graph below.



For each vertex v we will store a list that contains the neighbors of v:

```
0: [1, 2]
1: [2, 3]
2: [4]
```

```
3: [4, 5]
4: [5]
5: []
```

Here, 0: [1,2] means vertex 0 has the neighbors 1,2. Similarly, 5:[] means vertex 5 has no neighbors.

a) <u>Java implementation using an ArrayList of ArrayLists</u>

```
import java.util.*;
public class AdjacencyList {
    public static void main(String[] args) {
        // empty ArrayList
        ArrayList<ArrayList<Integer>> adjLists = new
ArrayList<ArrayList<Integer>>();
        // insert n=6 ArrayLists
        int n = 6;
        for(int i=0; i<n; i++){
            adjLists.add(new ArrayList<Integer>());
        }

        // insert neighbors into list for vertex 0
        adjLists.get(0).add(1);
        adjLists.get(0).add(2);

        // insert neighbors into list for vertex 1
        adjLists.get(1).add(2);
        adjLists.get(1).add(3);

        // insert neighbors into list for vertex 2
        adjLists.get(2).add(4);

        // insert neighbors into list for vertex 3
        adjLists.get(3).add(4);
        adjLists.get(3).add(5);

        // insert neighbors into list for vertex 4
        adjLists.get(4).add(5);

        // insert neighbors into list for vertex 5
        // -> nothing to do since 5 has no neighbors

        // testing
        System.out.println("Neigbors of vertex 0: " + adjLists.get(0));
        System.out.println("Neigbors of vertex 5: " + adjLists.get(5));
        System.out.println("\nPrint all adjacency lists with corresponding
vertex:");
        for(int v=0; v<n; v++){
            System.out.println(v + ": " + adjLists.get(v));
        }
    }
}
```

....

The output of that program is:

```
    Neigbors of vertex 0: [1, 2]
    Neigbors of vertex 5: []
     Print all adjacency lists with corresponding vertex:
    0: [1, 2]
    1: [2, 3]
    2: [4]
    3: [4, 5]
    4: [5]
    5: []
```

b) <u>Java implementation using a dictionary (HashMap)</u>

```
import java.util.*;
 public class AdjacencyList_Dict {
    public static void main(String[] args) {
        // empty dictionary
        HashMap<Integer, ArrayList<Integer>> adjLists_dict = new
HashMap<Integer, ArrayList<Integer>>();

        // insert empty lists for each node
        int n = 6;
        for(int v=0; v<n; v++){
            adjLists_dict.put(v, new ArrayList<Integer>());
        }

        // insert (vertex, list) pairs into dictionary
        // insert neighbors into list for vertex 0
        adjLists_dict.get(0).add(1);
        adjLists_dict.get(0).add(2);

        // insert neighbors into list for vertex 1
        adjLists_dict.get(1).add(2);
        adjLists_dict.get(1).add(3);

        // insert neighbors into list for vertex 2
        adjLists_dict.get(2).add(4);

        // insert neighbors into list for vertex 3
        adjLists_dict.get(3).add(4);
        adjLists_dict.get(3).add(5);

        // insert neighbors into list for vertex 4
        adjLists_dict.get(4).add(5);

        // insert neighbors into list for vertex 5
        // -> nothing to do since 5 has no neighbors

         // testing
        System.out.println("Neighbors of vertex 0: " + adjLists_dict.get(0));
        System.out.println("Neighbors of vertex 3: " + adjLists_dict.get(3));

        System.out.println("\nPrint all adjacency lists with corresponding
vertex:");
        for(int v=0; v<n; v++){
           System.out.println(v + ": " + adjLists_dict.get(v));
        }

    }

}
```
……………………………………………………………………………………………

The output is:

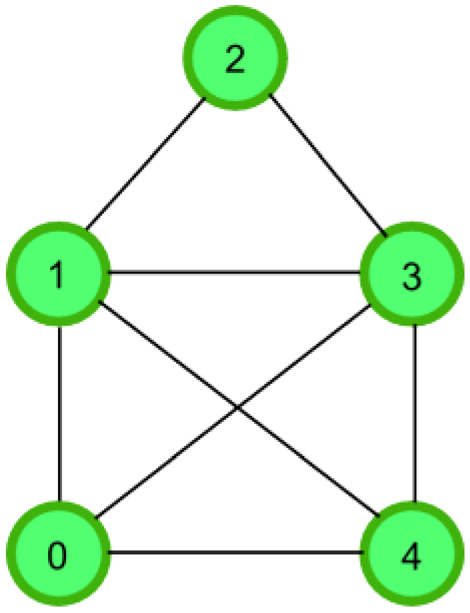
```
Neighbors of vertex 0: [1, 2]
Neighbors of vertex 3: [4, 5]

Print all adjacency lists with corresponding vertex:
0: [1, 2]
1: [2, 3]
2: [4]
3: [4, 5]
```

4: [5]
5: []
………
………

Form the adjacency matrix and adjacency lists from the edges

We will describe how to form the **adjacency matrix** and **adjacency list** representation if a list of all edges is given.

1. Edge list as two arrays
Suppose we are given the graph below:



The graph with n=5 nodes has the following edges:

1 (0,1), (0,3), (0,4),
2 (1,0), (1,2), (1,3), (1,4)
3 (2,1), (2,3),
4 (3,0), (3,1), (3,2), (3,4)
5 (4,0), (4,1), (4,3)

We can store the edges in two arrays edge_u[] and edge_v[], where (edge_u[i], edge_v[i]) represents an edge between two nodes. The arrays have the length M, where M is the number of edges.

They look like this:

```
edge_u = [0, 0, 0, 1, 1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 4]
edge_v = [1, 3, 4, 0, 2, 3, 4, 1, 3, 0, 1, 2, 4, 0, 1, 3]
```

For instance, `(edge_u[1], edge_v[1]) = (0, 3)` indicates that there is an edge from vertex 0 to vertex 3. Another example is `(edge_u[15], edge_v[15]) = (4, 3).`

2. Adjacency matrix from edge list

By scanning the arrays edge_u and edge_v we can form the adjacency matrix.

1. **Java implementation of Adjacency matrix from edge list**

```
public class Edgelist_To_Adjmatrix {

   public static void printMatrix(int[][] matrix){
      for(int i=0; i<matrix.length; i++){
         for(int k=0; k<matrix[0].length; k++){
            System.out.print(matrix[i][k] + " ");
         }
         System.out.println();
      }
      System.out.println();
   }

   // ----------------------------------------------------------------

   public static void main(String[] args) {
      int[] edge_u = {0, 0, 0, 1, 1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 4};
      int[] edge_v = {1, 3, 4, 0, 2, 3, 4, 1, 3, 0, 1, 2, 4, 0, 1, 3};

   // our graph has n = 5 nodes
   final int n = 5;

   // adjacency matrix - initialized with 0 by default in Java
   int[][] adjMatrix = new int[n][n];

   // scan the arrays edge_u and edge_v
```

```
    int m = edge_u.length;
    for(int i=0; i<m; i++){
        int u = edge_u[i];
        int v = edge_v[i];
        adjMatrix[u][v] = 1;
    }

    // check matrix
    System.out.println("Adjacency matrix:");
    printMatrix(adjMatrix);


    }
}
```
……………………………………………………………..

The output is:

```
Adjacency matrix:
0 1 0 1 1
1 0 1 1 1
0 1 0 1 0
1 1 1 0 1
1 1 0 1 0
```

…………………………………………………………………………………………………

2. Java implementation Adjacency lists from edge_list

………………………………………………………………………………………………

```
import java.util.ArrayList;

public class Edgelist_To_AdjList {

    public static void printAdjList(ArrayList<ArrayList<Integer>> adjList){
        int n = adjList.size();
        for(int i=0; i<n; i++){
            System.out.println(i + ": " + adjList.get(i));
        }
    }

    // ---------------------------------------------------------------

    public static void main(String[] args) {
        int[] edge_u = {0, 0, 0, 1, 1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 4};
        int[] edge_v = {1, 3, 4, 0, 2, 3, 4, 1, 3, 0, 1, 2, 4, 0, 1, 3};

    // our graph has n = 5 nodes
    final int n = 5;

    // create empty adjacency lists - one for each node
    ArrayList<ArrayList<Integer>> adjList = new ArrayList<ArrayList<Integer>>();
    for(int i=0; i<n; i++) adjList.add(new ArrayList<Integer>());
    // scan the arrays edge_u and edge_v
    int m = edge_u.length;
    for(int i=0; i<m; i++){
        int u = edge_u[i];
        int v = edge_v[i];
        adjList.get(u).add(v);
    }

    // check matrix
    System.out.println("Adjacency list:");
    printAdjList(adjList);

    }
}
```

……………………………………………………………………………….

The output is:

```
Adjacency list:
0: [1, 3, 4]
1: [0, 2, 3, 4]
2: [1, 3]
3: [0, 1, 2, 4]
4: [0, 1, 3]
```

…………………………………………………………………………………………
…………………………………………………………………………………………

Second Semester

……………………………………………………………………………..

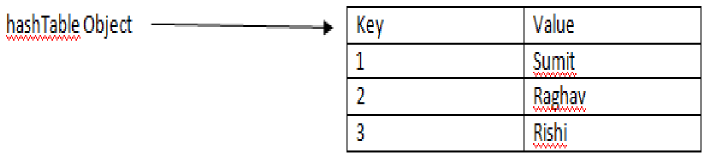
Hash Table Implementation

Objective : To implement a Hash Table

Input: A set of pairs of keys and values

Approach:

- Create a Hash Table
  - Hashtable<Integer, String> ht = new Hashtable<Integer, String>();
- Insert values in hash table using put(key,value)
  - ht.put(key, value);
- Get values from hash table using get(key)
  - ht.get(key);



Key	Value
1	Sumit
2	Raghav
3	Rishi

Complete Code:

```
package interviewQuestion;
import java.util.Hashtable;
public class SimpleHashTable {

        int [] a = new int[5];
        String [] arrNames = new
String[]{"Sumit","Jain","Raghav","Garg","Gaurav","Rishi"};

        Hashtable<Integer, String> ht = new Hashtable<Integer, String>();

        public void insertValues(){
                for(int i=0;i<arrNames.length;i++ ){
                        ht.put(i+1,arrNames[i]);
                }
        }
        public String getValue(int key){
                return ht.get(key);
        }
        public static void main (String [] args){
                SimpleHashTable sht = new SimpleHashTable();
                sht.insertValues();
                System.out.println("All values inserted");
                System.out.println("Employee with ID 1 is "+ sht.getValue(1));
                System.out.println("Employee with ID 3 is "+ sht.getValue(6));
        }
}
```

Output:

```
All values inserted
Employee with ID 1 is Sumit
Employee with ID 3 is Rishi
```

…………………………………………………………………………………

…………………………………………………………………………………..

Hashtable examples in Java with a set of operations:

1. How to put object into Hashtable?
2. How to retrieve object from Hashtable in Java?
3. How to reuse Hashtable by using clear()?
4. How to check if Hastable contains a particular value?
5. How to check if Hashtable contains a particular key?
6. How to traverse Hashtable in Java?
7. How to check if Hashtable is empty in Java?
8. How to find size of Hashtable in Java?
9. How to get all values form hashtable in Java?
10. How to get all keys from hashtable in Java?

Here is complete code example of all above hashtable example or exercises:

……………………………………………………………………………………………

```
import java.util.Collection;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Set;

public class HashtableDemo {

public static void main(String args[]) {

// Creating Hashtable for example
Hashtable companies = new Hashtable();

// Java Hashtable example to put object into Hashtable
// put(key, value) is used to insert object into map
companies.put("Google", "United States");
companies.put("Nokia", "Finland");
companies.put("Sony", "Japan");

// Java Hashtable example to get Object from Hashtable
// get(key) method is used to retrieve Objects from Hashtable
companies.get("Google");

// Hashtable containsKey Example
// Use containsKey(Object) method to check if an Object exits as key in
// hashtable
System.out.println("Does hashtable contains Google as key: "
+ companies.containsKey("Google"));


// Hashtable containsValue Example
// just like containsKey(), containsValue returns true if hashtable
// contains specified object as value
System.out.println("Does hashtable contains Japan as value: "
+ companies.containsValue("Japan"));

// Hashtable enumeration Example
// hashtabl.elements() return enumeration of all hashtable values
Enumeration enumeration = companies.elements();

while (enumeration.hasMoreElements()) {
System.out
.println("hashtable values: " + enumeration.nextElement());
}

// How to check if Hashtable is empty in Java
// use isEmpty method of hashtable to check emptiness of hashtable in
// Java
System.out.println("Is companies hashtable empty: "
+ companies.isEmpty());

// How to find size of Hashtable in Java
// use hashtable.size() method to find size of hashtable in Java
System.out.println("Size of hashtable in Java: " + companies.size());

// How to get all values form hashtable in Java
// you can use keySet() method to get a Set of all the keys of hashtable
// in Java
Set hashtableKeys = companies.keySet();

// you can also get enumeration of all keys by using method keys()
Enumeration hashtableKeysEnum = companies.keys();

// How to get all keys from hashtable in Java
// There are two ways to get all values form hashtalbe first by using
// Enumeration and second getting values ad Collection

Enumeration hashtableValuesEnum = companies.elements();

Collection hashtableValues = companies.values();

// Hashtable clear example
// by using clear() we can reuse an existing hashtable, it clears all
// mappings.
companies.clear();
}
}
```
………

Output:

```
Does hashtable contains Google as key: true
Does hashtable contains Japan as value: true
```

```
hashtable values: Finland
hashtable values: United States
hashtable values: Japan
Is companies hashtable empty: false
Size of hashtable in Java: 3
```

…………………………………………………………………………………
……………………………………………………………………………….

Java Program to Implement Merge Sort

This is a Java Program to implement Merge Sort on an integer array

…………………………………………………

Merge sort is a divide and conquer algorithm, conceptually, a merge sort works as follows:

1) Divide the unsorted list into n sublists, each containing 1 element (a list of 1 element is considered sorted).
2) Repeatedly merge sublists to produce new sublists until there is only 1 sublist remaining. This will be the sorted list.

....

```
import java.util.Scanner;
/* Class MergeSort */
public class MergeSort
{
   /* Merge Sort function */
   public static void sort(int[] a, int low, int high)
   {
      int N = high - low;
      if (N <= 1)
         return;
      int mid = low + N/2;
      // recursively sort
      sort(a, low, mid);
      sort(a, mid, high);
      // merge two sorted subarrays
      int[] temp = new int[N];
      int i = low, j = mid;
      for (int k = 0; k < N; k++)
```

```
        {
            if (i == mid)
                temp[k] = a[j++];
            else if (j == high)
                temp[k] = a[i++];
            else if (a[j]<a[i])
                temp[k] = a[j++];
            else
                temp[k] = a[i++];
        }
        for (int k = 0; k < N; k++)
            a[low + k] = temp[k];
    }
    /* Main method */
    public static void main(String[] args)
    {
        Scanner scan = new Scanner( System.in );
        System.out.println("Merge Sort Test\n");
        int n, i;
        /* Accept number of elements */
        System.out.println("Enter number of integer elements");
        n = scan.nextInt();
        /* Create array of n elements */
        int arr[] = new int[ n ];
        /* Accept elements */
        System.out.println("\nEnter "+ n +" integer elements");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextInt();
        /* Call method sort */
        sort(arr, 0, n);
        /* Print sorted Array */
        System.out.println("\nElements after sorting ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i]+" ");
        System.out.println();
    }
}
```
…………………………………………………………………………………………..

Example Sort Test

Enter number of integer elements
5
Enter 5 integer elements
20 1 100 30 2
Elements after sorting

1 2 20 30 100

……………………………………………………………………………

……………………………………………………………………………

Merge Two Sorted Arrays

………………………….

1) For each element in first array, compare it with the first element in the second array
2) If the element in first array is greater than the first element in second array, swap the two.
3) When all elements in first array are done, copy the second array contents to the result.

………………………..

```
    A  = {2,4,6,8,444};
    B  = {1,3,5,7,10,11,143};
```

array3 = {1 2 3 4 5 6 7 8 10 11 143 444**}**

Write a Java Code to merge two arrays of orders into one sorted array.

……………………….

Merge two sorted Arrays

Let 'a' and 'b' are two sorted arrays of size 'm' and 'n'. Merge arrays 'a' and 'b' to array 'c' .

………………………………………………………………………….

```
public class MergeArrays {

  static int[] merge(int a[], int b[]){
    int m = a.length;
    int n = b.length;
    int aCount=0, bCount=0;
    int i=0;
    int c[] = new int[m+n];

    while(aCount<m && bCount<n){
      if(a[aCount] < b[bCount]){
        c[i] = a[aCount];
        aCount++;
```

```
            }
            else{
                c[i] = b[bCount];
                bCount++;
            }
            i++;
        }

        if(aCount==m){
          for(int j=bCount; j<n; j++)
              c[i++] = b[j];
        }
        else{
          for(int j=aCount; j<m; j++)
            c[i++] = a[j];
        }
        return c;
    }

    public static void main(String args[]){
        int a[] = {2,4,6,8,444};
        int b[] = {1,3,5,7,10,11,143};

        int c[] = merge(b,a);
        for(int i=0; i<c.length; i++){
            System.out.print(c[i] +" ");
        }
    }
}
--------------------Output--------------------

1 2 3 4 5 6 7 8 10 11 143 444
```

. . . .

. . . .

Java Program to implement QuickSort Algorithm

Here is a Java program to sort an array of integers using **QuickSort algorithm**. It is an in-place, recursive implementation of QuickSort. Logic is encapsulated in QuickSort class, and method quickSort(int low, int high). This method is called recursively to sort the array.

```
import java.util.Arrays;

/**
 * Test class to sort array of integers using Quicksort algorithm in Java.
*/
public class QuickSortDemo{

   public static void main(String args[]) {

      // unsorted integer array
      int[] unsorted = {6, 5, 3, 1, 8, 7, 2, 4};
      System.out.println("Unsorted array :" + Arrays.toString(unsorted));

      QuickSort algorithm = new QuickSort();

      // sorting integer array using quicksort algorithm
      algorithm.sort(unsorted);

      // printing sorted array
      System.out.println("Sorted array :" + Arrays.toString(unsorted));

   }

}

/**
 * Java Program sort numbers using QuickSort Algorithm. QuickSort is a divide
 * and conquer algorithm, which divides the original list, sort it and then
 * merge it to create sorted output.
 *
*/
class QuickSort {

   private int input[];
   private int length;

   public void sort(int[] numbers) {

      if (numbers == null || numbers.length == 0) {
         return;
      }
      this.input = numbers;
      length = numbers.length;
      quickSort(0, length - 1);
   }
```

```
/*
 * This method implements in-place quicksort algorithm recursively.
 */
private void quickSort(int low, int high) {
    int i = low;
    int j = high;

    // pivot is middle index
    int pivot = input[low + (high - low) / 2];

    // Divide into two arrays
    while (i <= j) {
        /**
         * As shown in above image, In each iteration, we will identify a
         * number from left side which is greater then the pivot value, and
         * a number from right side which is less then the pivot value. Once
         * search is complete, we can swap both numbers.
         */
        while (input[i] < pivot) {
            i++;
        }
        while (input[j] > pivot) {
            j--;
        }
        if (i <= j) {
            swap(i, j);
            // move index to next position on both sides
            i++;
            j--;
        }
    }

    // calls quickSort() method recursively
    if (low < j) {
        quickSort(low, j);
    }

    if (i < high) {
        quickSort(i, high);
    }
}

private void swap(int i, int j) {
    int temp = input[i];
    input[i] = input[j];
    input[j] = temp;
    }
}
```

Output :

Unsorted array :[6, 5, 3, 1, 8, 7, 2, 4]

Sorted array :[1, 2, 3, 4, 5, 6, 7, 8]

…………………………………………………………………………………
……………………………………………………………………………..

Queue provides two sets of method for similar task e.g. **add() and offer()** for adding elements, **poll() and remove()** for removing head element from PriorityQueue and **peek() and element()** for returns the highest-priority element but does not modify the queue,

A priority queue must at least support the following operations:

- *insert_with_priority*: add an element to the queue with an associated priority.
- *pull_highest_priority_element*: remove the element from the queue that has the *highest priority*, and return it.

• •

In this example, you will learn: -

- how to add elements into PriorityQueue,
- how to remove elements from PriorityQueue,

By default, elements are sorted on increasing order and head contains the smallest elements.

Java Program to implement PriorityQueue Algorithm

…………………………………………………………………..

```
import java.util.ArrayList;
import java.util.Scanner;

public class Heaps2 {

    public void  PQ_isert(ArrayList<Integer> y, int v) {

           System.out.println("befor insert "+v+" --> "+    y.toString());

           System.out.println(" ");
        y.add(v);

        int c = y.size() - 1;

        int p = (c - 1) / 2;

        int temp = 0;
        while (p >= 0 && y.get(p) > y.get(c)) {

          System.out.println("  inserted "+v+" --> "+    y.toString());
        System.out.println(" ");

            temp = y.get(p);

            y.set(p, y.get(c));
            y.set(c, temp);

            c = p;
            p = (c - 1) / 2;

        }
       System.out.println("after insert --> "+    y.toString() +"\n" );


    }

    public void PQ_delete(ArrayList<Integer> y) {

        if (y.size()==0) {
            System.out.println("array is empty");
            return;
        }
        System.out.println("befor delete   --> "+    y.toString());
        y.set(0, y.get(y.size() - 1));
        y.remove(y.size() - 1);

        int p = 0;
        int leftChild = (2 * p) + 1;
        int rightChild = leftChild + 1;
        int minChild = 0;

        int temp = 0;

        while (true) {

            if (rightChild >= y.size() || leftChild >= y.size()) {
                break;
            }

            if (y.get(leftChild) < y.get(rightChild)) {
                minChild = leftChild;
            }

            if (y.get(leftChild) > y.get(rightChild)) {
                minChild = rightChild;
            }

            if (y.get(p) < y.get(minChild)) {
                break;
            }

            if (y.get(p) > y.get(minChild)) {

                temp = y.get(p);

                y.set(p, y.get(minChild));
                y.set(minChild, temp);
    System.out.println("after delete first number  --> "+    y.toString());
                p = minChild;

                leftChild = 2 * p + 1;
                rightChild = leftChild + 1;

            }

        }
```

```
   }

      static ArrayList<Integer> y = new ArrayList<>();

   public static void main(String[] args) {

       Heaps2 o=new Heaps2();

       ArrayList<Integer> x = new ArrayList<>();

       System.out.println("insert 8 number  \n");

       Scanner s=new Scanner(System.in);

            for (int i = 0; i < 8; i++) {

        int v=s.nextInt();

          o.PQ_isert(y, v);
       }


          o.PQ_delete(y);

   }

}
```

……………………………………… OUTPUT …………………………………..

```
insert 8 number

5 7 8 9 1 2 12 3
before insert 5 --> []
after insert --> [5]
before insert 7 --> [5]
 after insert --> [5, 7]
before insert 8 --> [5, 7]
 after insert --> [5, 7, 8]
before insert 9 --> [5, 7, 8]
 after insert --> [5, 7, 8, 9]
before insert 1 --> [5, 7, 8, 9]
   inserted 1 --> [5, 7, 8, 9, 1]
   inserted 1 --> [5, 1, 8, 9, 7]
 after insert --> [1, 5, 8, 9, 7]
before insert 2 --> [1, 5, 8, 9, 7]
```

```
   inserted 2 --> [1, 5, 8, 9, 7, 2]
 after insert --> [1, 5, 2, 9, 7, 8]
before insert 12 --> [1, 5, 2, 9, 7, 8]
 after insert --> [1, 5, 2, 9, 7, 8, 12]
before insert 3 --> [1, 5, 2, 9, 7, 8, 12]
   inserted 3 --> [1, 5, 2, 9, 7, 8, 12, 3]
   inserted 3 --> [1, 5, 2, 3, 7, 8, 12, 9]
 after insert --> [1, 3, 2, 5, 7, 8, 12, 9]
before delete   --> [1, 3, 2, 5, 7, 8, 12, 9]
after delete first number  --> [2, 3, 9, 5, 7, 8, 12]
after delete first number  --> [2, 3, 8, 5, 7, 9, 12]
```

....
....

…………………………………………………………………………
……………………………………………………………………..

We have implemented a class that should compress set of inputs using run-length encoding. We have implemented to techniques of encoding and decoding.

Java Program to implement Run Length Encoding Algorithm

……………………………………………………………………..

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RunLengthEncoding {

   public static String encode(String source) {
      StringBuffer dest = new StringBuffer();
      for (int i = 0; i < source.length(); i++) {
         int runLength = 1;
         while (i+1 < source.length() && source.charAt(i) == source.charAt(i+1)) {
            runLength++;
            i++;
         }
         dest.append(runLength);
         dest.append(source.charAt(i));
      }
      return dest.toString();
   }
```

```
    public static String decode(String source) {
        StringBuffer dest = new StringBuffer();
        Pattern pattern = Pattern.compile("[0-9]+|[a-zA-Z]");
        Matcher matcher = pattern.matcher(source);
        while (matcher.find()) {
            int number = Integer.parseInt(matcher.group());
            matcher.find();
            while (number-- != 0) {
                dest.append(matcher.group());
            }
        }
        return dest.toString();
    }

    public static void main(String[] args) {
        String example = "FFFFooff88ccfgdoyuuuueeee";

        System.out.print("    Encode -------->   ");
        System.out.println(encode(example));
        System.out.print("    Decode ------->    ");
        System.out.println(decode("3r6u8o5A"));
    }
}
```

………………………….. Output ………………………………

```
Encode -------->   4F2o2f282c1f1g1d1o1y4u4e

Decode ------->    rrruuuuuuooooooooAAAAA
```

…………………………………………………………………………………
……………………………………………………………………………..

Java Program for union-find algorithm to detect cycle in an Undirected Graph

………………………………………

A ***disjoint-set data structure*** is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. A ***union-find algorithm*** is an algorithm that performs two useful operations on such a data structure:

Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.

Union: Join two subsets into a single subset.
In this post, we will discuss an application of Disjoint Set Data Structure. The application is to check whether a given graph contains a cycle or not.

Union-Find Algorithm can be used to check whether an undirected graph contains cycle or not. Note that we have discussed an ***algorithm to detect cycle.*** This is another method based on *Union-Find*. This method assumes that graph doesn't contain any self-loops. We can keeps track of the subsets in a 1D array, lets call it parent[].
Let us consider the following graph:

```
0
| \
|   \
1-----2
```

For each edge, make subsets using both the vertices of the edge. If both the vertices are in the same subset, a cycle is found.

Initially, all slots of parent array are initialized to -1 (means there is only one item in every subset).

```
0   1   2

-1 -1  -1
```

Now process all edges one by one.

Edge 0-1: Find the subsets in which vertices 0 and 1 are. Since they are in different subsets, we take the union of them. For taking the union, either make node 0 as parent of node 1 or vice-versa.

```
0   1   2    <----- 1 is made parent of 0 (1 is now representative of subset {0, 1})

1  -1  -1
```

Edge 1-2: 1 is in subset 1 and 2 is in subset 2. So, take union.

```
0   1   2    <----- 2 is made parent of 1 (2 is now representative of subset {0, 1, 2})

1   2  -1
```

Edge 0-2: 0 is in subset 2 and 2 is also in subset 2. Hence, including this edge forms a cycle.
How subset of 0 is same as 2?
0->1->2 // 1 is parent of 0 and 2 is parent of 1

……………………………………………………………….

```
import java.util.*;
import java.lang.*;
import java.io.*;

class Graph
{
   int V, E;    // V-> no. of vertices & E->no.of edges
   Edge edge[]; // /collection of all edges

   class Edge
   {
      int src, dest;
   };

   // Creates a graph with V vertices and E edges
   Graph(int v,int e)
   {
      V = v;
```

```
    E = e;
    edge = new Edge[E];
    for (int i=0; i<e; ++i)
        edge[i] = new Edge();
}

// A utility function to find the subset of an element i
int find(int parent[], int i)
{
    if (parent[i] == -1)
        return i;
    return find(parent, parent[i]);
}

// A utility function to do union of two subsets
void Union(int parent[], int x, int y)
{
    int xset = find(parent, x);
    int yset = find(parent, y);
    parent[xset] = yset;
}


// The main function to check whether a given graph
// contains cycle or not
int isCycle( Graph graph)
{
    // Allocate memory for creating V subsets
    int parent[] = new int[graph.V];

    // Initialize all subsets as single element sets
    for (int i=0; i<graph.V; ++i)
        parent[i]=-1;

    // Iterate through all edges of graph, find subset of both
    // vertices of every edge, if both subsets are same, then
    // there is cycle in graph.
    for (int i = 0; i < graph.E; ++i)
    {
        int x = graph.find(parent, graph.edge[i].src);
```

```
            int y = graph.find(parent, graph.edge[i].dest);

            if (x == y)
                return 1;

            graph.Union(parent, x, y);
        }
        return 0;
    }

    // Driver Method
    public static void main (String[] args)
    {
        /* Let us create following graph
         0
        | \
        |   \
        1-----2 */
        int V = 3, E = 3;
        Graph graph = new Graph(V, E);

        // add edge 0-1
        graph.edge[0].src = 0;
        graph.edge[0].dest = 1;

        // add edge 1-2
        graph.edge[1].src = 1;
        graph.edge[1].dest = 2;

        // add edge 0-2
        graph.edge[2].src = 0;
        graph.edge[2].dest = 2;

        if (graph.isCycle(graph)==1)
            System.out.println( "Graph contains cycle" );
        else
            System.out.println( "Graph doesn't contain cycle" );
    }
}
```
………………………… output …………………………

Graph contains cycle

…………………………………………………………………………
………………………………………………………………………

Java Program for Rotation algorithm of Treaps Randomized Data Structures

…………………………………………………………..

The program used with Binary Tree Rotation (Left or Right)

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Tree;

/**
 *
 * @author sasan
 */
class Bst {

    Node root;

    public Bst() {
        root = null;
    }

    public Node addNode(int value, Node n) {
        if (n == null) {
            n = new Node(value);
        } else if (value <= n.data) {
            n.left = addNode(value, n.left);
        } else {
            n.right = addNode(value, n.right);
        }
        return n;
    }

    private void inorder(Node r) {
```

```
    if (r != null) {
        inorder(r.left);
        System.out.print(r.data + " ");
        inorder(r.right);
    }
}

private void preorder(Node r) {
    if (r != null) {
        System.out.print(r.data + " ");
        preorder(r.left);
        preorder(r.right);
    }
}

private void postorder(Node r) {
    if (r != null) {
        postorder(r.left);
        postorder(r.right);
        System.out.print(r.data + " ");
    }
}

public void leftRotation(Node p) {
    Node q = p.right;
    p.right = q.left;
    q.left = p;
    root = q;

}

public void rightRotation(Node q) {
    Node p = q.left;
    q.left = p.right;
    p.right = q;
    root = p;

}

public static void main(String[] args) {
    Bst tree = new Bst();
    tree.root = tree.addNode(23, tree.root);
    tree.root = tree.addNode(55, tree.root);
    tree.root = tree.addNode(80, tree.root);
    tree.root = tree.addNode(40, tree.root);
    tree.root = tree.addNode(11, tree.root);
```

```
   // System.out.println(tree.root.data);
        tree.leftRotation(tree.root);
     //   System.out.println(tree.root.left.right.data);
        //tree.rightRotation(tree.root);
        //System.out.println(tree.root.data);
        System.out.println("inorder");
        tree.inorder(tree.root);
        System.out.println("right Rotation");
        tree.rightRotation(tree.root);
        tree.inorder(tree.root);
        System.out.println("left Rotation");
        tree.leftRotation(tree.root);
        System.out.println("");
        tree.inorder(tree.root);

        /*tree.root=tree.addNode(9, tree.root);
         tree.root=tree.addNode(10, tree.root);*/
        /*tree.postorder(tree.root);
         System.out.println("");

         */
    }

    class Node {

        int data;
        Node right, left;

        public Node(int data) {
            this.data = data;
            left = right = null;
        }
    }
}
```

………………………………………………………………………………..

…………………………………………………………………………………