Data Structure – Arrays Questions

# 1)Rearrange an array such that arr[i] = i

Given an array of elements of length N, ranging from 1 to N. All elements may not be present in the array. If element is not present then there will be -1 present in the array. Rearrange the array such that A[i] = i and if i is not present, display -1 at that place.

**Examples:**

Input : arr = {-1, -1, 6, 1, 9, 3, 2, -1, 4, -1}

Output : [-1, 1, 2, 3, 4, -1, 6, -1, -1, 9]

Input : arr = {19, 7, 0, 3, 18, 15, 12, 6, 1, 8,

11, 10, 9, 5, 13, 16, 2, 14, 17, 4}

Output : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

11, 12, 13, 14, 15, 16, 17, 18, 19]

Soluction :

import java.util.\*;

import java.lang.\*;

class GfG {

    public static int[] fix(int[] A)

    {

        Set<Integer> s = new HashSet<Integer>();

        for(int i = 0; i < A.length; i++)

        {

           s.add(A[i]);

        }

        for(int i = 0; i < A.length; i++)

        {

           if(s.contains(i))

             A[i] = i;

           else

             A[i] = -1;

        }

      return A;

}

public static void main(String[] args)

    {

        int A[] = {-1, -1, 6, 1, 9,

                    3, 2, -1, 4,-1};

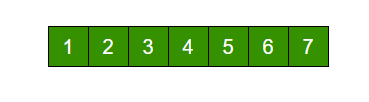
        System.out.println(Arrays.toString(fix(A)));

    }

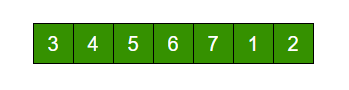
}

=============================================================================

# 2) Program for array rotation

Write a function rotate(ar[], d, n) that rotates arr[] of size n by d elements.  


Rotation of the above array by 2 will make array



Soluction :

class RotateArray {

    void leftRotate(int arr[], int d, int n)

    {

        for (int i = 0; i < d; i++)

            leftRotatebyOne(arr, n);

    }

    void leftRotatebyOne(int arr[], int n)

    {

        int i, temp;

        temp = arr[0];

        for (i = 0; i < n - 1; i++)

            arr[i] = arr[i + 1];

        arr[i] = temp;

    }

    void printArray(int arr[], int n)

    {

        for (int i = 0; i < n; i++)

            System.out.print(arr[i] + " ");

    }

    public static void main(String[] args)

    {

        RotateArray rotate = new RotateArray();

        int arr[] = { 1, 2, 3, 4, 5, 6, 7 };

        rotate.leftRotate(arr, 2, 7);

        rotate.printArray(arr, 7);

    }

}

=============================================================================

# 3) Move all zeroes to end of array

Given an array of random numbers, Push all the zero’s of a given array to the end of the array. For example, if the given arrays is {1, 9, 8, 4, 0, 0, 2, 7, 0, 6, 0}, it should be changed to {1, 9, 8, 4, 2, 7, 6, 0, 0, 0, 0}. The order of all other elements should be same. Expected time complexity is O(n) and extra space is O(1).

**Example:**

Input : arr[] = {1, 2, 0, 4, 3, 0, 5, 0};

Output : arr[] = {1, 2, 4, 3, 5, 0, 0};

Soluction :

import java.io.\*;

class PushZero

{

    static void pushZerosToEnd(int arr[], int n)

    {

        int count = 0;

        for (int i = 0; i < n; i++)

            if (arr[i] != 0)

                arr[count++] = arr[i];

        while (count < n)

            arr[count++] = 0;

    }

    public static void main (String[] args)

    {

        int arr[] = {1, 9, 8, 4, 0, 0, 2, 7, 0, 6, 0, 9};

        int n = arr.length;

        pushZerosToEnd(arr, n);

        System.out.println("Array after pushing zeros to the back: ");

        for (int i=0; i<n; i++)

            System.out.print(arr[i]+" ");

    }

}

=============================================================================

# 4) Program to find largest element in an array

Given an array, find the largest element in it.

**Example:**

Input : arr[] = {10, 20, 4}

Output : 20

Soluction :

class Test

{

     static int arr[] = {10, 324, 45, 90, 9808};

     static int largest()

     {

         int i;

         int max = arr[0];

         for (i = 1; i < arr.length; i++)

             if (arr[i] > max)

                 max = arr[i];

         return max;

     }

     public static void main(String[] args)

     {

         System.out.println("Largest in given array is " + largest());

     }

 }

=============================================================================

# 5) Find Second largest element in an array

class GFG {

    public static void print2largest(int arr[],

                                     int arr\_size)

    {

        int i, first, second;

        if (arr\_size < 2)

        {

            System.out.print(" Invalid Input ");

            return;

        }

        first = second = Integer.MIN\_VALUE;

        for (i = 0; i < arr\_size ; i++)

        {

            if (arr[i] > first)

            {

                second = first;

                first = arr[i];

            }

            else if (arr[i] > second && arr[i] != first)

                second = arr[i];

 }

        if (second == Integer.MIN\_VALUE)

             System.out.print("There is no second largest"+

                                 " element\n");

        else

             System.out.print("The second largest element"+

                                      " is "+ second);

    }

    public static void main(String[] args)

    {

            int arr[] = {12, 35, 1, 10, 34, 1};

            int n = arr.length;

            print2largest(arr, n);

    }

}

=============================================================================

**Linked List implementation**

class LinkedList

{

    Node head;

    class Node

    {

        int data;

        Node next;

        Node(int d) {data = d; next = null; }

    }

    public void push(int new\_data)

    {

        Node new\_node = new Node(new\_data);

        new\_node.next = head;

        head = new\_node;

    }

    public void insertAfter(Node prev\_node, int new\_data)

    {

        if (prev\_node == null)

        {

            System.out.println("The given previous node cannot be null");

            return;

        }

        Node new\_node = new Node(new\_data);

        new\_node.next = prev\_node.next;

        prev\_node.next = new\_node;

    }

    public void append(int new\_data)

    {

        Node new\_node = new Node(new\_data);

        if (head == null)

        {

            head = new Node(new\_data);

            return;

        }

        new\_node.next = null;

        Node last = head;

        while (last.next != null)

            last = last.next;

        last.next = new\_node;

        return;

    }

    public void printList()

    {

        Node tnode = head;

        while (tnode != null)

        {

            System.out.print(tnode.data+" ");

            tnode = tnode.next;

        }

    }

    public static void main(String[] args)

    {

        LinkedList llist = new LinkedList();

        llist.append(6);

        llist.push(7);

        llist.push(1);

        llist.append(4);

        llist.insertAfter(llist.head.next, 8);

        System.out.println("\nCreated Linked list is: ");

        llist.printList();

    }

}

Output :

Created Linked list is: 1 7 8 6 4

**Delete Node :**

    void deleteNode(int key)

    {

        Node temp = head, prev = null;

        if (temp != null && temp.data == key)

        {

            head = temp.next; // Changed head

            return;

        }

        while (temp != null && temp.data != key)

        {

            prev = temp;

            temp = temp.next;

        }

        if (temp == null) return;

        prev.next = temp.next;

    }

**Delete node at specified position :**

    void deleteNode(int position)

    {

        if (head == null)

            return;

        Node temp = head;

        if (position == 0)

        {

            head = temp.next;             return;

        }

        for (int i=0; temp!=null && i<position-1; i++)

            temp = temp.next;

        if (temp == null || temp.next == null)

            return;

        Node next = temp.next.next;

        temp.next = next;

    }

**Function to delete the Linked List :**

    void deleteList()

    {

        head = null;

    }

**Function to find the length of the Linked list**

public int getCount()

    {

        Node temp = head;

        int count = 0;

        while (temp != null)

        {

            count++;

            temp = temp.next;

        }

        return count;

    }

**Search function in linked list :**

public boolean search(Node head, int x)

    {

        Node current = head;

        while (current != null)

        {

            if (current.data == x)

                return true;

            current = current.next;

        }

        return false;

    }

# Write a function to get Nth node in a Linked List

 public int GetNth(int index)

    {

        Node current = head;

        int count = 0;

        while (current != null)

        {

            if (count == index)

                return current.data;

            count++;

            current = current.next;

        }

        /\* if we get to this line, the caller was asking

        for a non-existent element so we assert fail \*/

        assert(false);

        return 0;

    }

**Find the middle of a given linked list in C and Java**

    void printMiddle()

    {

        Node slow\_ptr = head;

        Node fast\_ptr = head;

        if (head != null)

        {

            while (fast\_ptr != null && fast\_ptr.next != null)

            {

                fast\_ptr = fast\_ptr.next.next;

                slow\_ptr = slow\_ptr.next;

            }

            System.out.println("The middle element is [" +

                                slow\_ptr.data + "] \n");

        }

    }

**Detect loop in a linked list**

 static boolean detectLoop(Node h)

    {

        HashSet<Node> s = new HashSet<Node>();

        while (h != null)

        {

            if (s.contains(h))

                return true;

            s.add(h);

            h = h.next;

        }

        return false;

    }

**Function to reverse the linked list**

    Node reverse(Node node) {

        Node prev = null;

        Node current = node;

        Node next = null;

        while (current != null) {

            next = current.next;

            current.next = prev;

            prev = current;

            current = next;

        }

        node = prev;

        return node;

    }