Important Algorithms

Contents

Binary Search

```java
public class BinarySearch {

 /**
  * Expectation : Array should be sorted
  *
  * if the num is found then print the index
  *
  * else print that the number is not found
  * @param arr
  * @param left
  * @param right
  * @param num
  *
  * TC : O(logn)
  *
  */
 public static void search(int[] arr, int left, int right, int num){

   //Base condition
   if(left > right){
     System.out.println("Number : "+ num +" can't be found in the array");
     return ;
   }

   int mid = left + (right - left)/2 ; // (left+right)/2

   if(arr[mid] == num){
     System.out.println(num + " is found at the index "+ mid);
   }
   else if ( arr[mid] > num){
     search(arr, left, mid-1, num);
   }else{
     search(arr, mid+1, right, num);
   }
 }


 public static void main(String[] args) {
   int[] arr = {1,2,3,4,5,6,7,8};

   search(arr, 0, arr.length-1 , 7);
 }
}
```

Insertion Sort

```java
// Java program for implementation of Insertion Sort
class InsertionSort {
    /*Function to sort array using insertion sort*/
    void sort(int arr[])
    {
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int j = i - 1;

            /* Move elements of arr[0..i-1], that are
            greater than key, to one position ahead
```

```
            of their current position */
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }

    /* A utility function to print array of size n*/
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");

        System.out.println();
    }

    // Driver method
    public static void main(String args[])
    {
        int arr[] = { 12, 11, 13, 5, 6 };

        InsertionSort ob = new InsertionSort();
        ob.sort(arr);

        printArray(arr);
    }
} /* This code is contributed by Rajat Mishra. */
```

Selection Sort

```
// Java program for implementation of Selection Sort
class SelectionSort
{
    void sort(int arr[])
    {
        int n = arr.length;

        // One by one move boundary of unsorted subarray
        for (int i = 0; i < n-1; i++)
        {
            // Find the minimum element in unsorted array
            int min_idx = i;
            for (int j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;

            // Swap the found minimum element with the first
            // element
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
    }

    // Prints the array
    void printArray(int arr[])
    {
        int n = arr.length;
```

```
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }

    // Driver code to test above
    public static void main(String args[])
    {
        SelectionSort ob = new SelectionSort();
        int arr[] = {64,25,12,22,11};
        ob.sort(arr);
        System.out.println("Sorted array");
        ob.printArray(arr);
    }
}
/* This code is contributed by Rajat Mishra*/
```

Search in a sorted and rotated array

```
/* Java program to search an element in
sorted and rotated array using
single pass of Binary Search*/

class Main {
    // Returns index of key in arr[l..h]
    // if key is present, otherwise returns -1
    static int search(int arr[], int l, int h, int key)
    {
        if (l > h)
            return -1;

        int mid = (l + h) / 2;
        if (arr[mid] == key)
            return mid;

        /* If arr[l...mid] first subarray is sorted */
        if (arr[l] <= arr[mid]) {
            /* As this subarray is sorted, we
            can quickly check if key lies in
            half or other half */
            if (key >= arr[l] && key <= arr[mid])
                return search(arr, l, mid - 1, key);
            /*If key not lies in first half subarray,
        Divide other half into two subarrays,
        such that we can quickly check if key lies
        in other half */
            return search(arr, mid + 1, h, key);
        }

        /* If arr[l..mid] first subarray is not sorted,
        then arr[mid... h] must be sorted subarry*/
        if (key >= arr[mid] && key <= arr[h])
            return search(arr, mid + 1, h, key);

        return search(arr, l, mid - 1, key);
    }

    // main function
    public static void main(String args[])
    {
```

```
        int arr[] = { 4, 5, 6, 7, 8, 9, 1, 2, 3 };
        int n = arr.length;
        int key = 6;
        int i = search(arr, 0, n - 1, key);
        if (i != -1)
            System.out.println("Index: " + i);
        else
            System.out.println("Key not found");
    }
}
```

Array Rotation

```
// Java program to rotate an array by
// d elements

class RotateArray {
    /*Function to left rotate arr[] of size n by d*/
    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[n-1] = temp;
    }

    /* utility function to print an array */
    void printArray(int arr[], int n)
    {
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
    }

    // Driver program to test above functions
    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);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Remove Duplicates from a Sorted LL

```
// Java program for the above approach

import java.io.*;

import java.util.*;
```

```
class Node
{
    int data;
    Node next;
    Node()
    {
        data = 0;
        next = null;
    }
}
class GFG
{

    /* Function to insert a node at
the beginging of the linked
* list */
    static Node push(Node head_ref, int new_data)
    {

        /* allocate node */
        Node new_node = new Node();

        /* put in the data */
        new_node.data = new_data;

        /* link the old list off
        the new node */
        new_node.next = (head_ref);

        /* move the head to point
        to the new node */
        head_ref = new_node;
        return head_ref;
    }
```

```
/* Function to print nodes
in a given linked list */
static void printList(Node node)
{
    while (node != null)
    {
        System.out.print(node.data + " ");
        node = node.next;
    }
}

// Function to remove duplicates
static void removeDuplicates(Node head)
{
    HashMap<Integer, Boolean> track = new HashMap<>();
    Node temp = head;

    while(temp != null)
    {
        if(!track.containsKey(temp.data))
        {
            System.out.print(temp.data + " ");
        }
        track.put(temp.data , true);
        temp = temp.next;
    }
}

// Driver Code
public static void main (String[] args)
{
    Node head = null;
```

```
        /* Created linked list will be
        11->11->11->13->13->20 */
        head = push(head, 20);
        head = push(head, 13);
        head = push(head, 13);
        head = push(head, 11);
        head = push(head, 11);
        head = push(head, 11);
        System.out.print("Linked list before duplicate removal ");
        printList(head);
        System.out.print("\nLinked list after duplicate removal ");
        removeDuplicates(head);
    }
}

// This code is contributed by avanitrachhadiya2155
```

Merge two Sorted Linked Lists

```
class GFG
{
    public Node SortedMerge(Node A, Node B)
    {

        if(A == null) return B;
        if(B == null) return A;

        if(A.data < B.data)
        {
            A.next = SortedMerge(A.next, B);
            return A;
        }
        else
        {
            B.next = SortedMerge(A, B.next);
            return B;
        }

    }
}

// This code is contributed by Tuhin Das
```

Detect and Remove Loop in Linked List

```
// Java program to detect and remove loop in a linked list
import java.util.*;
```

```java
public class LinkedList {

    static Node head; // head of list

    /* Linked list Node*/
    static class Node {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    /* Inserts a new Node at front of the list. */
    static public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    // Function to print the linked list
    void printList(Node node)
    {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }

    // Returns true if the loop is removed from the linked
    // list else returns false.
    static boolean removeLoop(Node h)
    {
        HashSet<Node> s = new HashSet<Node>();
        Node prev = null;
        while (h != null) {
            // If we have already has this node
            // in hashmap it means their is a cycle and we
            // need to remove this cycle so set the next of
            // the previous pointer with null.

            if (s.contains(h)) {
                prev.next = null;
                return true;
            }

            // If we are seeing the node for
            // the first time, insert it in hash
            else {
                s.add(h);
                prev = h;
                h = h.next;
```

```
            }
        }

        return false;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        LinkedList llist = new LinkedList();

        llist.push(20);
        llist.push(4);
        llist.push(15);
        llist.push(10);

        /*Create loop for testing */
        llist.head.next.next.next.next = llist.head;

        if (removeLoop(head)) {
            System.out.println("Linked List after removing loop");
            llist.printList(head);
        }
        else
            System.out.println("No Loop found");
    }
}

// This code is contributed by Animesh Nag.
```

Delete Last Node in Linked List

```
// Java program to remove last node of
// linked list.
class GFG {

    // Link list node /
    static class Node {
        int data;
        Node next;
    };

    // Function to remove the last node
    // of the linked list /
    static Node removeLastNode(Node head)
    {
        if (head == null)
            return null;

        if (head.next == null) {
            return null;
        }

        // Find the second last node
        Node second_last = head;
        while (second_last.next.next != null)
            second_last = second_last.next;

        // Change next of second last
        second_last.next = null;
```

```
        return head;
    }

    // Function to push node at head
    static Node push(Node head_ref, int new_data)
    {
        Node new_node = new Node();
        new_node.data = new_data;
        new_node.next = (head_ref);
        (head_ref) = new_node;
        return head_ref;
    }

    // Driver code
    public static void main(String args[])
    {
        // Start with the empty list /
        Node head = null;

        // Use push() function to con
        // the below list 8 . 23 . 11 . 29 . 12 /
        head = push(head, 12);
        head = push(head, 29);
        head = push(head, 11);
        head = push(head, 23);
        head = push(head, 8);

        head = removeLastNode(head);
        for (Node temp = head; temp != null; temp = temp.next)
            System.out.print(temp.data + " ");
    }
}

// This code is contributed by Arnab Kundu
```

Middle Element in a Linked List

```
// Java program to find middle of linked list
class LinkedList
{
    Node head; // head of linked list

    /* Linked list node */
    class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    /* Function to print middle of linked list */
    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");
        }
    }

    /* Inserts a new Node at front of the list. */
    public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    /* This function prints contents of linked list
    starting from the given node */
    public void printList()
    {
        Node tnode = head;
        while (tnode != null)
        {
            System.out.print(tnode.data+"->");
            tnode = tnode.next;
        }
        System.out.println("NULL");
    }

    public static void main(String [] args)
    {
        LinkedList llist = new LinkedList();
        for (int i=5; i>0; --i)
        {
            llist.push(i);
            llist.printList();
            llist.printMiddle();
        }
    }
}
// This code is contributed by Rajat Mishra
```

Implement Stack using single Queue

```
// Java program to implement stack using a
// single queue

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

public class stack
```

```
{
    Queue<Integer> q = new LinkedList<Integer>();

    // Push operation
    void push(int val)
    {
        // get previous size of queue
        int size = q.size();

        // Add current element
        q.add(val);

        // Pop (or Dequeue) all previous
        // elements and put them after current
        // element
        for (int i = 0; i < size; i++)
        {
            // this will add front element into
            // rear of queue
            int x = q.remove();
            q.add(x);
        }
    }

    // Removes the top element
    int pop()
    {
        if (q.isEmpty())
        {
            System.out.println("No elements");
            return -1;
        }
        int x = q.remove();
        return x;
    }

    // Returns top of stack
    int top()
    {
        if (q.isEmpty())
            return -1;
        return q.peek();
    }

    // Returns true if Stack is empty else false
    boolean isEmpty()
    {
        return q.isEmpty();
    }

    // Driver program to test above methods
    public static void main(String[] args)
    {
        stack s = new stack();
        s.push(10);
        s.push(20);
        System.out.println("Top element :" + s.top());
        s.pop();
        s.push(30);
        s.pop();
        System.out.println("Top element :" + s.top());
```

```
    }
}

// This code is contributed by Rishabh Mahrsee
```

Design a Stack with operations on Middle Element

```
/* Java Program to implement a stack that supports
findMiddle() and deleteMiddle in O(1) time */

public class GFG {
    /* A Doubly Linked List Node */
    class DLLNode {
        DLLNode prev;
        int data;
        DLLNode next;
        DLLNode(int d) { data = d; }
    }

    /* Representation of the stack data structure that
    supports findMiddle() in O(1) time. The Stack is
    implemented using Doubly Linked List. It maintains
    pointer to head node, pointer to middle node and
    count of nodes */
    class myStack {
        DLLNode head;
        DLLNode mid;
        int count;
    }

    /* Function to create the stack data structure */
    myStack createMyStack()
    {
        myStack ms = new myStack();
        ms.count = 0;
        return ms;
    }

    /* Function to push an element to the stack */
    void push(myStack ms, int new_data)
    {

        /* allocate DLLNode and put in data */
        DLLNode new_DLLNode = new DLLNode(new_data);

        /* Since we are adding at the beginning,
        prev is always NULL */
        new_DLLNode.prev = null;

        /* link the old list off the new DLLNode */
        new_DLLNode.next = ms.head;

        /* Increment count of items in stack */
        ms.count += 1;

        /* Change mid pointer in two cases
        1) Linked List is empty
        2) Number of nodes in linked list is odd */
        if (ms.count == 1) {
            ms.mid = new_DLLNode;
```

```
    }
    else {
        ms.head.prev = new_DLLNode;

        if ((ms.count % 2)
            != 0) // Update mid if ms->count is odd
            ms.mid = ms.mid.prev;
    }

    /* move head to point to the new DLLNode */
    ms.head = new_DLLNode;
}

/* Function to pop an element from stack */
int pop(myStack ms)
{
    /* Stack underflow */
    if (ms.count == 0) {
        System.out.println("Stack is empty");
        return -1;
    }

    DLLNode head = ms.head;
    int item = head.data;
    ms.head = head.next;

    // If linked list doesn't become empty, update prev
    // of new head as NULL
    if (ms.head != null)
        ms.head.prev = null;

    ms.count -= 1;

    // update the mid pointer when we have even number
    // of elements in the stack, i,e move down the mid
    // pointer.
    if (ms.count % 2 == 0)
        ms.mid = ms.mid.next;

    return item;
}

// Function for finding middle of the stack
int findMiddle(myStack ms)
{
    if (ms.count == 0) {
        System.out.println("Stack is empty now");
        return -1;
    }
    return ms.mid.data;
}

// Driver program to test functions of myStack
public static void main(String args[])
{
    GFG ob = new GFG();
    myStack ms = ob.createMyStack();
    ob.push(ms, 11);
    ob.push(ms, 22);
    ob.push(ms, 33);
    ob.push(ms, 44);
```

```
        ob.push(ms, 55);
        ob.push(ms, 66);
        ob.push(ms, 77);

        System.out.println("Item popped is " + ob.pop(ms));
        System.out.println("Item popped is " + ob.pop(ms));
        System.out.println("Middle Element is "
                           + ob.findMiddle(ms));
    }
}

// This code is contributed by Sumit Ghosh
```

Implement Queue Using Stack

```
// Java program to implement Queue using
// two stacks with costly enQueue()
import java.util.*;

class GFG
{
static class Queue
{
    static Stack<Integer> s1 = new Stack<Integer>();
    static Stack<Integer> s2 = new Stack<Integer>();

    static void enQueue(int x)
    {
        // Move all elements from s1 to s2
        while (!s1.isEmpty())
        {
            s2.push(s1.pop());
            //s1.pop();
        }

        // Push item into s1
        s1.push(x);

        // Push everything back to s1
        while (!s2.isEmpty())
        {
            s1.push(s2.pop());
            //s2.pop();
        }
    }

    // Dequeue an item from the queue
    static int deQueue()
    {
        // if first stack is empty
        if (s1.isEmpty())
        {
            System.out.println("Q is Empty");
            System.exit(0);
        }

        // Return top of s1
        int x = s1.peek();
        s1.pop();
        return x;
    }
```

```
};

// Driver code
public static void main(String[] args)
{
    Queue q = new Queue();
    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);

    System.out.println(q.deQueue());
    System.out.println(q.deQueue());
    System.out.println(q.deQueue());
}
}

// This code is contributed by Prerna Saini
```

Generate Binary Numbers from 1 to n - Using Queue

```
// Java program to generate binary numbers from 1 to n

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

public class GenerateBNo {
    // This function uses queue data structure to print
    // binary numbers
    static void generatePrintBinary(int n)
    {
        // Create an empty queue of strings
        Queue<String> q = new LinkedList<String>();

        // Enqueue the first binary number
        q.add("1");

        // This loops is like BFS of a tree with 1 as root
        // 0 as left child and 1 as right child and so on
        while (n-- > 0) {
            // print the front of queue
            String s1 = q.peek();
            q.remove();
            System.out.println(s1);

            // Store s1 before changing it
            String s2 = s1;

            // Append "0" to s1 and enqueue it
            q.add(s1 + "0");

            // Append "1" to s2 and enqueue it. Note that s2
            // contains the previous front
            q.add(s2 + "1");
        }
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        int n = 10;
```

```
        generatePrintBinary(n);
    }
}
// This code is contributed by Sumit Ghosh
```

BFS for a graph

```
// Java program to print BFS traversal from a given source vertex.
// BFS(int s) traverses vertices reachable from s.
import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency list
// representation
class Graph
{
    private int V; // No. of vertices
    private LinkedList<Integer> adj[]; //Adjacency Lists

    // Constructor
    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }

    // Function to add an edge into the graph
    void addEdge(int v,int w)
    {
        adj[v].add(w);
    }

    // prints BFS traversal from a given source s
    void BFS(int s)
    {
        // Mark all the vertices as not visited(By default
        // set as false)
        boolean visited[] = new boolean[V];

        // Create a queue for BFS
        LinkedList<Integer> queue = new LinkedList<Integer>();

        // Mark the current node as visited and enqueue it
        visited[s]=true;
        queue.add(s);

        while (queue.size() != 0)
        {
            // Dequeue a vertex from queue and print it
            s = queue.poll();
            System.out.print(s+" ");

            // Get all adjacent vertices of the dequeued vertex s
            // If a adjacent has not been visited, then mark it
            // visited and enqueue it
            Iterator<Integer> i = adj[s].listIterator();
            while (i.hasNext())
            {
                int n = i.next();
```

```
                if (!visited[n])
                {
                    visited[n] = true;
                    queue.add(n);
                }
            }
        }
    }

    // Driver method to
    public static void main(String args[])
    {
        Graph g = new Graph(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Following is Breadth First Traversal "+
                           "(starting from vertex 2)");

        g.BFS(2);
    }
}
// This code is contributed by Aakash Hasija
```

DFS for a graph

```
// Java program to print DFS
// traversal from a given given
// graph
import java.io.*;
import java.util.*;

// This class represents a
// directed graph using adjacency
// list representation
class Graph {
    private int V; // No. of vertices

    // Array of lists for
    // Adjacency List Representation
    private LinkedList<Integer> adj[];

    // Constructor
    @SuppressWarnings("unchecked") Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }

    // Function to add an edge into the graph
    void addEdge(int v, int w)
    {
        adj[v].add(w); // Add w to v's list.
    }
```

```
    // A function used by DFS
    void DFSUtil(int v, boolean visited[])
    {
        // Mark the current node as visited and print it
        visited[v] = true;
        System.out.print(v + " ");

        // Recur for all the vertices adjacent to this
        // vertex
        Iterator<Integer> i = adj[v].listIterator();
        while (i.hasNext()) {
            int n = i.next();
            if (!visited[n])
                DFSUtil(n, visited);
        }
    }

    // The function to do DFS traversal. It uses recursive
    // DFSUtil()
    void DFS()
    {
        // Mark all the vertices as not visited(set as
        // false by default in java)
        boolean visited[] = new boolean[V];

        // Call the recursive helper function to print DFS
        // traversal starting from all vertices one by one
        for (int i = 0; i < V; ++i)
            if (visited[i] == false)
                DFSUtil(i, visited);
    }

    // Driver Code
    public static void main(String args[])
    {
        Graph g = new Graph(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println(
            "Following is Depth First Traversal");

        g.DFS();
    }
}
// This code is contributed by Aakash Hasija
```

Djikstra's Algorithm

```
// A Java program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph
import java.util.*;
import java.lang.*;
import java.io.*;
```

```
class ShortestPath {
    // A utility function to find the vertex with minimum distance value,
    // from the set of vertices not yet included in shortest path tree
    static final int V = 9;
    int minDistance(int dist[], Boolean sptSet[])
    {
        // Initialize min value
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min) {
                min = dist[v];
                min_index = v;
            }

        return min_index;
    }

    // A utility function to print the constructed distance array
    void printSolution(int dist[])
    {
        System.out.println("Vertex \t\t Distance from Source");
        for (int i = 0; i < V; i++)
            System.out.println(i + " \t\t " + dist[i]);
    }

    // Function that implements Dijkstra's single source shortest path
    // algorithm for a graph represented using adjacency matrix
    // representation
    void dijkstra(int graph[][], int src)
    {
        int dist[] = new int[V]; // The output array. dist[i] will hold
        // the shortest distance from src to i

        // sptSet[i] will true if vertex i is included in shortest
        // path tree or shortest distance from src to i is finalized
        Boolean sptSet[] = new Boolean[V];

        // Initialize all distances as INFINITE and stpSet[] as false
        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        // Distance of source vertex from itself is always 0
        dist[src] = 0;

        // Find shortest path for all vertices
        for (int count = 0; count < V - 1; count++) {
            // Pick the minimum distance vertex from the set of vertices
            // not yet processed. u is always equal to src in first
            // iteration.
            int u = minDistance(dist, sptSet);

            // Mark the picked vertex as processed
            sptSet[u] = true;

            // Update dist value of the adjacent vertices of the
            // picked vertex.
            for (int v = 0; v < V; v++)
```

```
                // Update dist[v] only if is not in sptSet, there is an
                // edge from u to v, and total weight of path from src to
                // v through u is smaller than current value of dist[v]
                if (!sptSet[v] && graph[u][v] != 0 && dist[u] !=
Integer.MAX_VALUE && dist[u] + graph[u][v] < dist[v])
                    dist[v] = dist[u] + graph[u][v];
        }

        // print the constructed distance array
        printSolution(dist);
    }

    // Driver method
    public static void main(String[] args)
    {
        /* Let us create the example graph discussed above */
        int graph[][] = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                                      { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                                      { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                                      { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                                      { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                                      { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                                      { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                                      { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                                      { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
        ShortestPath t = new ShortestPath();
        t.dijkstra(graph, 0);
    }
}
// This code is contributed by Aakash Hasija
```