2. Lookup Table

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class LookUpTable {
    public static void main(String[] args) {
        DictionaryLookUp dict = new DictionaryLookUp();

        dict.insertItem(2, 3);
        dict.insertItem(3, 4);
        dict.insertItem(4, 5);
        dict.insertItem(5, 6);
        dict.insertItem(6, 7);

        int removedElement = dict.removeItem(4);
        System.out.println("Item removed for key = 4: " + removedElement);
        System.out.println("Found 4 at position " + dict.findItem(3));

        List<Integer> keys = dict.getKeys();
        keys.forEach(key -> System.out.print(key + " "));
        System.out.println();

        List<List<Integer>> dictionary = dict.getElement();
        for(int i=0;i<dictionary.size();i++) {
            for(int j=0;j<2;j++) {
                System.out.print(dictionary.get(i).get(j) + " ");
            }
            System.out.println();
        }

        System.out.println("Size of Dictionary: " + dict.size());
    }
}
class DictionaryLookUp {
    List<List<Integer>> a;
    int size;

    DictionaryLookUp() {
        size = 0;
        a = new ArrayList<>();
```

```
}

void insertItem(int key, int value) {
    List<Integer> toAdd = new ArrayList<>(Arrays.asList(key, value));
    if(isEmpty()) {
        a.add(toAdd);
        size++;
        return;
    }
    int left = 0;
    int right = a.size() - 1;
    int ans = a.size();
    while(left <= right) {
        int mid = left + (right - left)/2;
        if(a.get(mid).get(0) >= key) {
            ans = mid;
            right = mid - 1;
        }else {
            left = mid + 1;
        }
    }
    a.add(ans, toAdd);
    size++;
}

boolean findItem(int key) {
    if(isEmpty())
        return false;
    int left = 0;
    int right = size - 1;

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

        if(a.get(mid).get(0) == key) {
            return true;
        }else if(a.get(mid).get(0) < key) {
            left = mid + 1;
        }else {
            right = mid - 1;
        }
    }
    return false;
}
```

```
int removeItem(int key) {
    if(isEmpty())
        return -1;
    int left = 0;
    int right = size - 1;

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

        if(a.get(mid).get(0) == key) {
            size--;
            int element = a.get(mid).get(1);
            a.remove(mid);
            return element;
        }else if(a.get(mid).get(0) < key) {
            left = mid + 1;
        }else {
            right = mid - 1;
        }
    }
    return -1;
}

int size() {
    return this.size;
}

boolean isEmpty() {
    return this.size == 0;
}

List<Integer> getKeys() {
    List<Integer> ans = new ArrayList<>();
    if(isEmpty())
        return ans;
    for(int i=0;i<a.size();i++) {
        ans.add(a.get(i).get(0));
    }
    return ans;
}

List<List<Integer>> getElement() {
    return a;
```

```
    }
}
```

3:. Program to implement different applications using Divide and Conquer and Computation of time complexity-Binary Search, Ternary Search, Merge-sort, Quick-sort.

a)binary

```
#include <iostream>
using namespace std;
int binarySearch(int a[], int beg, int end, int val)
{
    int mid;
    if(end >= beg)
    {
        mid = (beg + end)/2;
        if(a[mid] == val)
        {
            return mid+1;
        }
        else if(a[mid] < val)
        {
            return binarySearch(a, mid+1, end, val);
        }
    else
        {
            return binarySearch(a, beg, mid-1, val);
        }
    }
    return -1;
}
int main() {
  int a[] = {1,5,7,10, 12,15,20 ,24,30,35, 40, 41, 46, 70};
  int val = 40;
  int n = sizeof(a) / sizeof(a[0]);
  int res = binarySearch(a, 0, n-1, val);
  cout<<"The elements of the array are - ";
  for (int i = 0; i < n; i++)
  cout<<a[i]<<" ";
  cout<<"\nElement to be searched is - "<<val;
  if (res == -1)
  cout<<"\nElement is not present in the array";
  else
```

```
  cout<<"\nElement is present at "<<res<<" position of array";
  return 0;
}
```

```
The elements of the array are - 1 5 7 10 12 15 20 24 30 35 40 41 46 70
Element to be searched is - 40
Element is present at 11 position of array
```

b) ternary

```
#include <bits/stdc++.h>
using namespace std;
int ternarySearch(int l, int r, int key, int ar[])
{
    if (r >= l) {
        int mid1 = l + (r - l) / 3;
        int mid2 = r - (r - l) / 3;
        if (ar[mid1] == key) {
            return mid1;
        }
        if (ar[mid2] == key) {
            return mid2;
        }
        if (key < ar[mid1]) {
            return ternarySearch(l, mid1 - 1, key, ar);
        }
        else if (key > ar[mid2]) {
            return ternarySearch(mid2 + 1, r, key, ar);
        }
        else {
            return ternarySearch(mid1 + 1, mid2 - 1, key, ar);
        }
    }
    return -1;
}
int main()
{
    int l, r, p, key;
    int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11,12,13,14,15,16};
    l = 0;
    r = 9;
    key = 7;
    p = ternarySearch(l, r, key, ar);
    cout << "Index of " << key
         << " is " << p << endl;
```

```
    key = 50;
    p = ternarySearch(l, r, key, ar);
    cout << "Index of " << key
         << " is " << p << endl;
}
```

```
Index of 7 is 6
Index of 50 is -1
```

c)merge sort

```
#include <iostream>
using namespace std;
void merge(int a[], int beg, int mid, int end)
{
    int i, j, k;
    int n1 = mid - beg + 1;
    int n2 = end - mid;
    int LeftArray[n1], RightArray[n2];
    for (int i = 0; i < n1; i++)
    LeftArray[i] = a[beg + i];
    for (int j = 0; j < n2; j++)
    RightArray[j] = a[mid + 1 + j];

    i = 0;
    j = 0;
    k = beg;
    while (i < n1 && j < n2)
    {
        if(LeftArray[i] <= RightArray[j])
        {
            a[k] = LeftArray[i];
            i++;
        }
        else
        {
            a[k] = RightArray[j];
            j++;
        }
        k++;
    }
    while (i<n1)
    {
        a[k] = LeftArray[i];
```

```
        i++;
        k++;
    }

    while (j<n2)
    {
        a[k] = RightArray[j];
        j++;
        k++;
    }
}

void mergeSort(int a[], int beg, int end)
{
    if (beg < end)
    {
        int mid = (beg + end) / 2;
        mergeSort(a, beg, mid);
        mergeSort(a, mid + 1, end);
        merge(a, beg, mid, end);
    }
}
void printArray(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout<<a[i]<<" ";
}

int main()
{
    int a[] = { 11, 30,5,34,65,13,46,67,23,55,14,8,79};
    int n = sizeof(a) / sizeof(a[0]);
    cout<<"Before sorting array elements are - \n";
    printArray(a, n);
    mergeSort(a, 0, n - 1);
    cout<<"\nAfter sorting array elements are - \n";
    printArray(a, n);
    return 0;
}
```

```
Before sorting array elements are -
11 30 5 34 65 13 46 67 23 55 14 8 79
After sorting array elements are -
5 8 11 13 14 23 30 34 46 55 65 67 79
```

d)quick sort

```
#include <bits/stdc++.h>
using namespace std;
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
int partition(int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i
        = (low- 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high)
{
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
```

```
}
int main()
{
    int arr[] = { 10,9,4,5,6,3,2,7,6,5,8,0,1,5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "array: \n";
    printArray(arr, n);
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

```
array:
10 9 4 5 6 3 2 7 6 5 8 0 1 5
Sorted array:
0 1 2 3 4 5 5 5 6 6 7 8 9 10
```

4. Program to implement randomized quick-sort and compute its complexity.

```
#include <bits/stdc++.h>
#define MAX 100

using namespace std;

int Partition(int a[], int low, int high)
{
    int pivot, index, i;
    index = low;
    pivot = high;
    for (i = low; i < high; i++)
    {
        if (a[i] < a[pivot])
        {
            swap(a[i], a[index]);
            index++;
        }
    }
    swap(a[pivot], a[index]);
    return index;
}
void random_shuffle(int arr[], int n)
{
    srand(time(NULL));
    for (int i = n - 1; i > 0; i--)
    {
        int j = rand() % (i + 1);
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
int RandomPivotPartition(int a[], int low, int high)
{
    int pvt, n, temp;
    n = rand();
    pvt = low + n % (high - low + 1);
    swap(a[high], a[pvt]);
    return Partition(a, low, high);
}
void quick_sort(int arr[], int p, int q)
{
    int pindex;
```

```
    if (p < q)
    {
        pindex = RandomPivotPartition(arr, p, q);
        quick_sort(arr, p, pindex - 1);
        quick_sort(arr, pindex + 1, q);
    }
}
int main()
{
    int n=5;
    int arr[n];
    for (int i = 0; i < n; i++)
        arr[i] = i + 1;
    random_shuffle(arr, n);
    quick_sort(arr, 0, n - 1);
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

Output khud laga lo

5.

```
        #include <bits/stdc++.h>
using namespace std;

void countSort(vector<int>& arr)
{
    int max = *max_element(arr.begin(), arr.end());
    int min = *min_element(arr.begin(), arr.end());
    int range = max - min + 1;

    vector<int> count(range), output(arr.size());
    for (int i = 0; i < arr.size(); i++)
        count[arr[i] - min]++;

    for (int i = 1; i < count.size(); i++)
        count[i] += count[i - 1];
```

```
    for (int i = arr.size() - 1; i >= 0; i--) {
        output[count[arr[i] - min] - 1] = arr[i];
        count[arr[i] - min]--;
    }

    for (int i = 0; i < arr.size(); i++)
        arr[i] = output[i];
}

int main()
{
    vector<int> arr = { -5, -10, 0, -3, 8, 5, -1, 10, 4, 8, 5 };
    countSort(arr);

    for(auto i:arr)
        cout<<i<<" ";

    return 0;
}
```

8. Program to find shortest path using Dijkstra’s Algorithm.

```
#include <bits/stdc++.h>
using namespace std;
#define INF 0x3f3f3f3f

typedef pair<int, int> iPair;


class Graph {
        int V;
        list<pair<int, int> >* adj;

public:
        Graph(int V);

        void addEdge(int u, int v, int w);
```

```
	void shortestPath(int s);
};

Graph::Graph(int V)
{
	this->V = V;
	adj = new list<iPair>[V];
}

void Graph::addEdge(int u, int v, int w)
{
	adj[u].push_back(make_pair(v, w));
	adj[v].push_back(make_pair(u, w));
}

void Graph::shortestPath(int src)
{

	priority_queue<iPair, vector<iPair>, greater<iPair> >
		pq;


	vector<int> dist(V, INF);


	pq.push(make_pair(0, src));
	dist[src] = 0;


	while (!pq.empty()) {

		int u = pq.top().second;
		pq.pop();


		list<pair<int, int> >::iterator i;
		for (i = adj[u].begin(); i != adj[u].end(); ++i) {

			int v = (*i).first;
			int weight = (*i).second;

			if (dist[v] > dist[u] + weight) {
				dist[v] = dist[u] + weight;
```

```
                    pq.push(make_pair(dist[v], v));
                }
            }
        }

        printf("Vertex Distance from Source\n");
        for (int i = 0; i < V; ++i)
            printf("%d \t\t %d\n", i, dist[i]);
}

int main()
{
        int V = 9;
        Graph g(V);

        g.addEdge(0, 1, 4);
        g.addEdge(0, 7, 8);
        g.addEdge(1, 2, 8);
        g.addEdge(1, 7, 11);
        g.addEdge(2, 3, 7);
        g.addEdge(2, 8, 2);
        g.addEdge(2, 5, 4);
        g.addEdge(3, 4, 9);
        g.addEdge(3, 5, 14);
        g.addEdge(4, 5, 10);
        g.addEdge(5, 6, 2);
        g.addEdge(6, 7, 1);
        g.addEdge(6, 8, 6);
        g.addEdge(7, 8, 7);

        g.shortestPath(0);

        return 0;
}
```

Output:

```
Vertex Distance from Source
0               0
1               4
2               12
3               19
4               21
5               11
6               9
7               8
8               14
```

9. Program to find All pair shortest path

```
#include <iostream>
#include <iomanip>
#define N 7
#define INF 999
using namespace std;
// Cost matrix of the graph
int costMat[N][N] = {
    {0, 2, 6, INF, INF, INF, INF},
    {3, 1, 2, 1, 9, INF, INF},
    {6, 2, 0, 5, 4, 2, 2},
    {INF, 1, 1, 0, 2, INF, 4},
    {INF, INF, 4, 5, 0, 2, 1},
    {INF, INF, 2, INF, 2, 0, 1},
    {INF, INF, INF, 4, 1, 1, 0}};
void floydWarshal()
{
    int cost[N][N]; // defind to store shortest distance from any N to any
N
    for (int i = 0; i < N; i++)
```

```
        for (int j = 0; j < N; j++)
            cost[i][j] = costMat[i][j]; // copy costMatrix to new matrix
    for (int k = 0; k < N; k++)
    {
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                if (cost[i][k] + cost[k][j] < cost[i][j])
                    cost[i][j] = cost[i][k] + cost[k][j];
    }
    cout << "The matrix:" << endl;
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            cout << setw(3) << cost[i][j];
        cout << endl;
    }
}
int main()
{
    floydWarshal();
}
```

Output:

```
The matrix:
  0  2  4  3  5  6  6
  3  1  2  1  3  4  4
  5  2  0  3  3  2  2
  4  1  1  0  2  3  3
  9  6  4  5  0  2  1
  7  4  2  5  2  0  1
  8  5  3  4  1  1  0
```

10 Implement Topological Sort

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> topoSort(int n, vector<int> adj[])
{
    vector<int> indegree(n, 0);
    for (int i = 0; i < n; i++)
    {
        for (auto it : adj[i])
        {
            indegree[it]++;
        }
    }
    queue<int> q;
    vector<int> ans;
    for (int i = 0; i < n; i++)
    {
        if (indegree[i] == 0)
            q.push(i);
    }
    while (!q.empty())
    {
        int currNode = q.front();
        q.pop();
        ans.push_back(currNode);
        for (auto it : adj[currNode])
        {
            indegree[it]--;
            if (indegree[it] == 0)
                q.push(it);
        }
    }
    return ans;
    // code here
}
int main()
{
    int n;
    cin >> n;
    vector<int> adj[n];
```

```
    int e;
    cin >> e;
    for (int i = 0; i < e; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<int> temp = topoSort(n, adj);
    for (auto it : temp)
        cout << it << " ";
    cout << endl;
}
```

Output:

For Input:

```
3 4
3 0
1 0
2 0
```

Your Output:

```
1 2 3 0
```

11.

```cpp
#include <bits/stdc++.h>
using namespace std;

// Number of vertices in given graph
#define V 6

bool bfs(int rGraph[V][V], int s, int t, int parent[])
{
    bool visited[V] = {};

    queue<int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;

    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        for (int v = 0; v < V; v++)
        {
            if (visited[v] == false && rGraph[u][v] > 0)
            {
                if (v == t)
                {
                    parent[v] = u;
                    return true;
                }
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }
    return false;
}

int fordFulkerson(int graph[V][V], int s, int t)
{
```

```
    int u, v;

    int rGraph[V][V];
    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
            rGraph[u][v] = graph[u][v];

    int parent[V];

    int max_flow = 0;
    while (bfs(rGraph, s, t, parent))
    {

        int path_flow = INT_MAX;
        for (v = t; v != s; v = parent[v])
        {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }

        for (v = t; v != s; v = parent[v])
        {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }

        max_flow += path_flow;
    }
    return max_flow;
}
int main()
{
    int src = 0, sink = 4;
    int graph[V][V] = {{0, 20, 13, 0, 0, 0},
                       {0, 0, 10, 12, 0, 0},
                       {0, 4, 0, 4, 14, 0},
                       {0, 0, 9, 0, 0, 20},
                       {0, 0, 0, 7, 6, 4},
                       {0, 1, 0, 3, 0, 0}};
```

```
    cout << "The maximum possible flow is "
         << fordFulkerson(graph, src, );
}
```

Output

```
The source vertex is 0
The sink vertex is 4
The maximum possible flow is 14
```

12. Red Black Trees

Bro C++ mae karde

//Program 12: Red Black Trees

```
public class RedBlackTree<T extends Comparable<T>> implements Iterable<T> {

  public static final boolean RED = true;
  public static final boolean BLACK = false;

  public class Node {

    // The color of this node. By default all nodes start red.
    public boolean color = RED;

    // The value/data contained within the node.
    public T value;

    // The left, right and parent references of this node.
    public Node left, right, parent;

    public Node(T value, Node parent) {
      this.value = value;
      this.parent = parent;
    }

    public Node(boolean color, T value) {
      this.color = color;
      this.value = value;
    }

    Node(T key, boolean color, Node parent, Node left, Node right) {
      this.value = key;
      this.color = color;

      if (parent == null && left == null && right == null) {
        parent = this;
        left = this;
        right = this;
      }

      this.parent = parent;
      this.left = left;
```

```
        this.right = right;
    }

    public boolean getColor() {
        return color;
    }

    public void setColor(boolean color) {
        this.color = color;
    }

    public T getValue() {
        return value;
    }

    public void setValue(T value) {
        this.value = value;
    }

    public Node getLeft() {
        return left;
    }

    public void setLeft(Node left) {
        this.left = left;
    }

    public Node getRight() {
        return right;
    }

    public void setRight(Node right) {
        this.right = right;
    }

    public Node getParent() {
        return parent;
    }

    public void setParent(Node parent) {
        this.parent = parent;
    }
}
```

```
// The root node of the RB tree.
public Node root;

// Tracks the number of nodes inside the tree.
private int nodeCount = 0;

public final Node NIL;

public RedBlackTree() {
    NIL = new Node(BLACK, null);
    NIL.left = NIL;
    NIL.right = NIL;
    NIL.parent = NIL;

    root = NIL;
}

// Returns the number of nodes in the tree.
public int size() {
    return nodeCount;
}

// Returns whether or not the tree is empty.
public boolean isEmpty() {
    return size() == 0;
}

public boolean contains(T value) {

    Node node = root;

    if (node == null || value == null)
        return false;

    while (node != NIL) {

        // Compare current value to the value in the node.
        int cmp = value.compareTo(node.value);

        // Dig into left subtree.
        if (cmp < 0)
            node = node.left;

        // Dig into right subtree.
```

```
        else if (cmp > 0)
            node = node.right;

        // Found value in tree.
        else
            return true;
    }

    return false;
}

public boolean insert(T val) {
    if (val == null) {
        throw new IllegalArgumentException("Red-Black tree does not allow null values.");
    }

    Node x = root, y = NIL;

    while (x != NIL) {
        y = x;

        if (x.getValue().compareTo(val) > 0) {
            x = x.left;
        } else if (x.getValue().compareTo(val) < 0) {
            x = x.right;
        } else {
            return false;
        }
    }

    Node z = new Node(val, RED, y, NIL, NIL);

    if (y == NIL) {
        root = z;
    } else if (z.getValue().compareTo(y.getValue()) < 0) {
        y.left = z;
    } else {
        y.right = z;
    }
    insertFix(z);

    nodeCount++;
    return true;
}
```

```
private void insertFix(Node z) {
    Node y;
    while (z.parent.color == RED) {
        if (z.parent == z.parent.parent.left) {
            y = z.parent.parent.right;
            if (y.color == RED) {
                z.parent.color = BLACK;
                y.color = BLACK;
                z.parent.parent.color = RED;
                z = z.parent.parent;
            } else {
                if (z == z.parent.right) {
                    z = z.parent;
                    leftRotate(z);
                }
                z.parent.color = BLACK;
                z.parent.parent.color = RED;
                rightRotate(z.parent.parent);
            }
        } else {
            y = z.parent.parent.left;
            if (y.color == RED) {
                z.parent.color = BLACK;
                y.color = BLACK;
                z.parent.parent.color = RED;
                z = z.parent.parent;
            } else {
                if (z == z.parent.left) {
                    z = z.parent;
                    rightRotate(z);
                }
                z.parent.color = BLACK;
                z.parent.parent.color = RED;
                leftRotate(z.parent.parent);
            }
        }
    }
    root.setColor(BLACK);
    NIL.setParent(null);
}

private void leftRotate(Node x) {
    Node y = x.right;
```

```
    x.setRight(y.getLeft());
    if (y.getLeft() != NIL)
        y.getLeft().setParent(x);
    y.setParent(x.getParent());
    if (x.getParent() == NIL)
        root = y;
    if (x == x.getParent().getLeft())
        x.getParent().setLeft(y);
    else
        x.getParent().setRight(y);
    y.setLeft(x);
    x.setParent(y);
}

private void rightRotate(Node y) {
    Node x = y.left;
    y.left = x.right;
    if (x.right != NIL)
        x.right.parent = y;
    x.parent = y.parent;
    if (y.parent == NIL)
        root = x;
    if (y == y.parent.left)
        y.parent.left = x;
    else
        y.parent.right = x;
    x.right = y;
    y.parent = x;
}

public boolean delete(T key) {
    Node z;
    if (key == null || (z = (search(key, root))) == NIL)
        return false;
    Node x;
    Node y = z; // temporary reference y
    boolean y_original_color = y.getColor();

    if (z.getLeft() == NIL) {
        x = z.getRight();
        transplant(z, z.getRight());
    } else if (z.getRight() == NIL) {
        x = z.getLeft();
        transplant(z, z.getLeft());
```

```
    } else {
        y = successor(z.getRight());
        y_original_color = y.getColor();
        x = y.getRight();
        if (y.getParent() == z)
            x.setParent(y);
        else {
            transplant(y, y.getRight());
            y.setRight(z.getRight());
            y.getRight().setParent(y);
        }
        transplant(z, y);
        y.setLeft(z.getLeft());
        y.getLeft().setParent(y);
        y.setColor(z.getColor());
    }
    if (y_original_color == BLACK)
        deleteFix(x);
    nodeCount--;
    return true;
}

private void deleteFix(Node x) {
    while (x != root && x.getColor() == BLACK) {
        if (x == x.getParent().getLeft()) {
            Node w = x.getParent().getRight();
            if (w.getColor() == RED) {
                w.setColor(BLACK);
                x.getParent().setColor(RED);
                leftRotate(x.parent);
                w = x.getParent().getRight();
            }
            if (w.getLeft().getColor() == BLACK && w.getRight().getColor() == BLACK) {
                w.setColor(RED);
                x = x.getParent();
                continue;
            } else if (w.getRight().getColor() == BLACK) {
                w.getLeft().setColor(BLACK);
                w.setColor(RED);
                rightRotate(w);
                w = x.getParent().getRight();
            }
            if (w.getRight().getColor() == RED) {
                w.setColor(x.getParent().getColor());
```

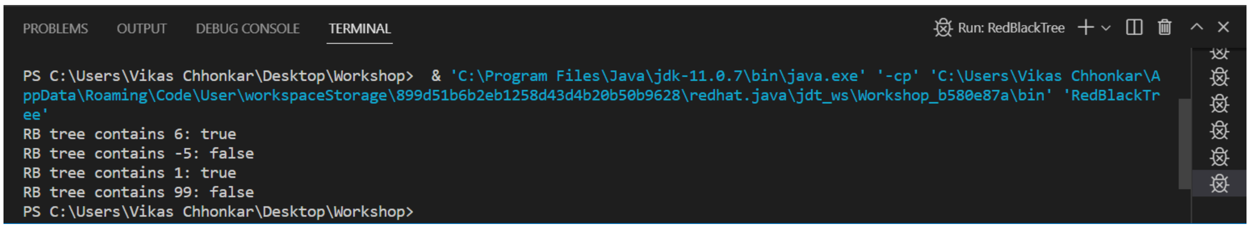
```
                x.getParent().setColor(BLACK);
                w.getRight().setColor(BLACK);
                leftRotate(x.getParent());
                x = root;
            }
        } else {
            Node w = (x.getParent().getLeft());
            if (w.color == RED) {
                w.color = BLACK;
                x.getParent().setColor(RED);
                rightRotate(x.getParent());
                w = (x.getParent()).getLeft();
            }
            if (w.right.color == BLACK && w.left.color == BLACK) {
                w.color = RED;
                x = x.getParent();
                continue;
            } else if (w.left.color == BLACK) {
                w.right.color = BLACK;
                w.color = RED;
                leftRotate(w);
                w = (x.getParent().getLeft());
            }
            if (w.left.color == RED) {
                w.color = x.getParent().getColor();
                x.getParent().setColor(BLACK);
                w.left.color = BLACK;
                rightRotate(x.getParent());
                x = root;
            }
        }
    }
    x.setColor(BLACK);
}

private Node successor(Node root) {
    if (root == NIL || root.left == NIL)
        return root;
    else
        return successor(root.left);
}

private void transplant(Node u, Node v) {
    if (u.parent == NIL) {
```

```
        root = v;
    } else if (u == u.parent.left) {
        u.parent.left = v;
    } else
        u.parent.right = v;
    v.parent = u.parent;
}

private Node search(T val, Node curr) {
    if (curr == NIL)
        return NIL;
    else if (curr.value.equals(val))
        return curr;
    else if (curr.value.compareTo(val) < 0)
        return search(val, curr.right);
    else
        return search(val, curr.left);
}

public int height() {
    return height(root);
}

private int height(Node curr) {
    if (curr == NIL) {
        return 0;
    }
    if (curr.left == NIL && curr.right == NIL) {
        return 1;
    }

    return 1 + Math.max(height(curr.left), height(curr.right));
}

// Returns as iterator to traverse the tree in order.
@Override
public java.util.Iterator<T> iterator() {

    final int expectedNodeCount = nodeCount;
    final java.util.Stack<Node> stack = new java.util.Stack<>();
    stack.push(root);

    return new java.util.Iterator<T>() {
        Node trav = root;
```

```
      @Override
      public boolean hasNext() {
        if (expectedNodeCount != nodeCount)
          throw new java.util.ConcurrentModificationException();
        return root != NIL && !stack.isEmpty();
      }

      @Override
      public T next() {

        if (expectedNodeCount != nodeCount)
          throw new java.util.ConcurrentModificationException();

        while (trav != NIL && trav.left != NIL) {
          stack.push(trav.left);
          trav = trav.left;
        }

        Node node = stack.pop();

        if (node.right != NIL) {
          stack.push(node.right);
          trav = node.right;
        }

        return node.value;
      }

      @Override
      public void remove() {
        throw new UnsupportedOperationException();
      }
    };
  }

  // Example usage of RB tree:
  public static void main(String[] args) {
    int[] values = { 5, 8, 1, -4, 6, -2, 0, 7 };
    RedBlackTree<Integer> rbTree = new RedBlackTree<>();
    for (int v : values)
      rbTree.insert(v);

    System.out.printf("RB tree contains %d: %s\n", 6, rbTree.contains(6));
```

```
        System.out.printf("RB tree contains %d: %s\n", -5, rbTree.contains(-5));
        System.out.printf("RB tree contains %d: %s\n", 1, rbTree.contains(1));
        System.out.printf("RB tree contains %d: %s\n", 99, rbTree.contains(99));
    }
}
```

Output:



13:Program to find Longest Common subsequence using heuristic technique.

```
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

string LCS(string X, string Y)
{
    int m = X.length();
    int n = Y.length();

    int L[m + 1][n + 1];

    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            if (i == 0 || j == 0)
                L[i][j] = 0;

            else if (X[i - 1] == Y[j - 1])
                L[i][j] = L[i - 1][j - 1] + 1;
```

```
            else
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
        }
    }

    int index = L[m][n];

    string lcs(index + 1, ' ');

    int i = m, j = n;
    while (i > 0 && j > 0)
    {
        if (X[i - 1] == Y[j - 1])
        {
            lcs[index - 1] = X[i - 1]; // put current character in result
            i--;
            j--;
            index--;          }

        else if (L[i - 1][j] > L[i][j - 1])
            i--;
        else
            j--;
    }

    return lcs;
}

int main()
{
    string X, Y;

    cout << "Enter first string: ";
    cin >> X;
    cout << "Enter second string: ";
    cin >> Y;
    string lcs = LCS(X, Y);
    cout << "The longest common subsequence is: " << lcs << endl;

    return 0;
}
```

OUTPUT:

```
PS D:\COding\C++ programming> cd "d:\COding\C++ pro
gramming\" ; if ($?) { g++ test11.cpp -o test11 } ;
 if ($?) { .\test11 }
Enter first string: WORKSHOP
Enter second string: WORKS
The longest common subsequence is: WORKS
PS D:\COding\C++ programming>
```

Q14. LCS USING BRUTE FORCE

```
#include <bits/stdc++.h>
  using namespace std;
  int main()
  {

  int t;cin>>t;
  while(t--)
  {
  string s1,s2;
  cin>>s1>>s2;
  int dp[s1.length()+1][s2.length()+1];
  int n=s1.length(),m=s2.length();
  for(int i=0;i<=n;i++)
  {
   for(int j=0;j<=m;j++)
   if(i==0||j==0)
   dp[i][j]=0;
   else
   {
    if(s1[i-1]==s2[j-1])
    dp[i][j]=dp[i-1][j-1]+1;
    else
    dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
   }
```

```
}
cout<<dp[n][m]<<"\n";
}
 return 0;
}
```

```
Output                                                    Clear
/tmp/DMgGOsJQGE.o
6
aabbcc
aabb
4
```

15. Program to find the Longest Common subsequence using Knuth Morris Pratt technique.

```
#include <bits/stdc++.h>

void computeLPSArray(char* pat, int M, int* lps);

// Prints occurrences of txt[] in pat[]
void KMPSearch(char* pat, char* txt)
{
	int M = strlen(pat);
	int N = strlen(txt);

	// create lps[] that will hold the longest prefix suffix
	// values for pattern
	int lps[M];

	// Preprocess the pattern (calculate lps[] array)
	computeLPSArray(pat, M, lps);

	int i = 0; // index for txt[]
	int j = 0; // index for pat[]
	while ((N - i) >= (M - j)) {
		if (pat[j] == txt[i]) {
			j++;
			i++;
		}

		if (j == M) {
			printf("Found pattern at index %d ", i - j);
			j = lps[j - 1];
		}

		// mismatch after j matches
		else if (i < N && pat[j] != txt[i]) {
			// Do not match lps[0..lps[j-1]] characters,
			// they will match anyway
			if (j != 0)
				j = lps[j - 1];
			else
				i = i + 1;
		}
	}
}

// Fills lps[] for given pattern pat[0..M-1]
```

```
void computeLPSArray(char* pat, int M, int* lps)
{
	// length of the previous longest prefix suffix
	int len = 0;

	lps[0] = 0; // lps[0] is always 0

	// the loop calculates lps[i] for i = 1 to M-1
	int i = 1;
	while (i < M) {
		if (pat[i] == pat[len]) {
			len++;
			lps[i] = len;
			i++;
		}
		else // (pat[i] != pat[len])
		{

			if (len != 0) {
				len = lps[len - 1];

				// Also, note that we do not increment
				// i here
			}
			else // if (len == 0)
			{
				lps[i] = 0;
				i++;
			}
		}
	}

int main()
{
	char txt[] = "ABABDABACDABABCABAB";
	char pat[] = "ABABCABAB";
	KMPSearch(pat, txt);
	return 0;
}
```