1. Design and implement a data structure for a Least Recently Used (LRU) cache.

import java.util.\*;

public class Main {

public static void main(String[] args) {

LRUCache cache = new LRUCache(2);

cache.set(1, 10);

cache.set(2, 20);

System.out.println("Value for the key 1 is " + cache.get(1)); // returns 10

// Evicts key 2 and stores a key (3) with value 30 in the cache

cache.set(3, 30);

System.out.println("Value for the key 2 is " + cache.get(2)); // returns -1

// Evicts key 1 and stores a key (4) with value 40 in the cache

cache.set(4, 40);

System.out.println("Value for the key 1 is " + cache.get(1)); // returns -1

System.out.println("Value for the key 3 is " + cache.get(3)); // returns 30

System.out.println("Value for the key 4 is " + cache.get(4)); // returns 40

}

}

class LRUCache {

private HashMap<Integer, Integer> cache;

private LinkedHashSet<Integer> usage;

private int capacity;

public LRUCache(int capacity) {

this.capacity = capacity;

this.cache = new HashMap<>();

this.usage = new LinkedHashSet<>();

}

public int get(int key) {

if (!cache.containsKey(key)) return -1;

// Move accessed key to the end to show that it was recently used

usage.remove(key);

usage.add(key);

return cache.get(key);

}

public void set(int key, int value) {

if (cache.containsKey(key)) {

// Remove the old to update its position in usage

usage.remove(key);

} else if (cache.size() >= capacity) {

int lruKey = usage.iterator().next();

usage.remove(lruKey);

cache.remove(lruKey);

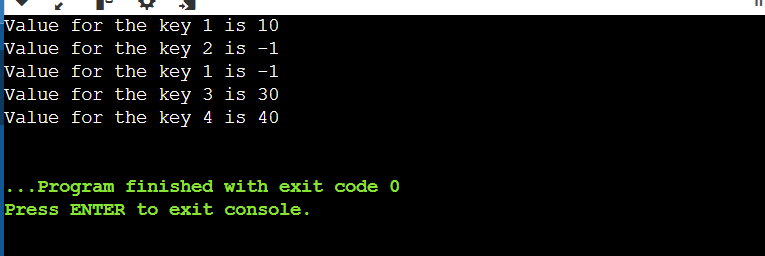
}

cache.put(key, value);

usage.add(key);

}

}



1. ConcurrentModificationException

import java.util.\*;

public class Main{

public static void main(String args[])

{

ArrayList<String> arr = new ArrayList<String>();

arr.add("One");

arr.add("Two");

arr.add("Three");

arr.add("Four");

try {

Iterator<String> iter = arr.iterator();

while (iter.hasNext()) {

System.out.print(iter.next() + ", ");

}

// No exception is raised as

// a modification is done

// after the iteration

System.out.println( "\n\nTrying to add"+ " an element in+ "betweeniteration: "+ arr.add("Five"));

// Printing the elements

System.out.println( "\nUpdated ArrayList: ");

iter = arr.iterator();

while (iter.hasNext()) {

System.out.print(iter.next() + ", ");

}

}

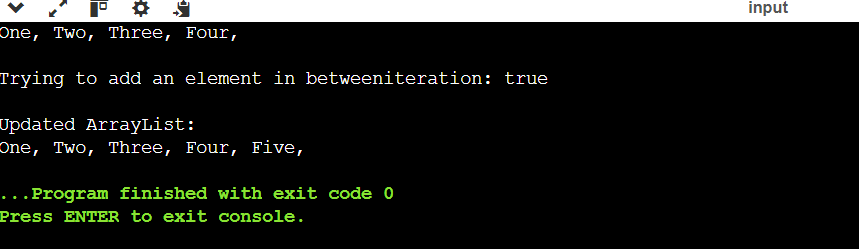
catch (Exception e) {

System.out.println(e);

}

}

}



if while iterating over the collection, we directly try to modify that collection, then the given will throw this ConcurrentModificationException.

To avoid this exception,

* Simply we can do the modifications once the iteration is done.
* Implement the concept of the synchronized block or method.

1. @LogExecutionTime

public void serve() throws InterruptedException {

Thread.sleep(2000);

}



1. A Java program to demonstrate serialization and deserialization of Binary Tree

import java.util.\*;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) { val = x; }

}

class Main {

TreeNode root;

// Encodes a tree to a single string.

public static String serialize(TreeNode root)

{

if (root == null) {

return null;

}

Stack<TreeNode> s = new Stack<>();

s.push(root);

List<String> l = new ArrayList<>();

while (!s.isEmpty()) {

TreeNode t = s.pop();

// If current node is NULL, store marker

if (t == null) l.add("#");

else {

// Else, store current node

// and recur for its children

l.add("" + t.val);

s.push(t.right);

s.push(t.left);

}

}

return String.join(",", l); //used to convert the list of strings l into a single string

}

static int t;

// Decodes your encoded data to tree.

public static TreeNode deserialize(String data)

{

if (data == null)

return null;

t = 0;

String[] arr = data.split(",");

return helper(arr);

}

public static TreeNode helper(String[] arr)

{

if (arr[t].equals("#"))

return null;

// Create node with this item

// and recur for children

TreeNode root = new TreeNode(Integer.parseInt(arr[t]));

t++;

root.left = helper(arr);

t++;

root.right = helper(arr);

return root;

}

// A simple inorder traversal used

// for testing the constructed tree

static void inorder(TreeNode root)

{

if (root != null) {

inorder(root.left);

System.out.print(root.val + " ");

inorder(root.right);

}

}

// Driver code

public static void main(String args[])

{

// Construct a tree shown in the above figure

Main tree = new Main();

tree.root = new TreeNode(20);

tree.root.left = new TreeNode(8);

tree.root.right = new TreeNode(22);

tree.root.left.left = new TreeNode(4);

tree.root.left.right = new TreeNode(12);

tree.root.left.right.left = new TreeNode(10);

tree.root.left.right.right = new TreeNode(14);

String serialized = serialize(tree.root);

System.out.println("Serialized view of the tree:");

System.out.println(serialized);

System.out.println();

// Deserialize the stored tree into root1

TreeNode t = deserialize(serialized);

System.out.println(

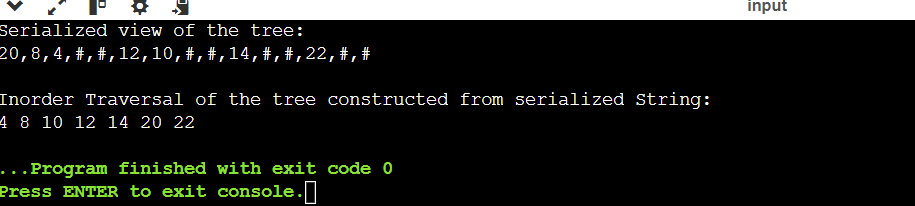
"Inorder Traversal of the tree constructed"

+ " from serialized String:");

inorder(t);

}

}



1. Implement a trie with insert, search, and startsWith methods.

import java.util.\*;

class Trie {

class Node {

char data;

boolean isTerminal;

HashMap<Character, Node> child = new HashMap<>();

}

private Node root;

public Trie() {

Node nn = new Node();

nn.data = '\*';

root = nn;

}

public void insert(String word) {

Node curr = root;

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

char ch = word.charAt(i);

if (curr.child.containsKey(ch)) {

curr = curr.child.get(ch);

} else {

Node nn = new Node();

nn.data = ch;

curr.child.put(ch, nn);

curr = nn;

}

}

curr.isTerminal = true;

}

public boolean search(String word) {

Node curr = root;

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

char ch = word.charAt(i);

if (curr.child.containsKey(ch)) {

curr = curr.child.get(ch);

} else {

return false;

}

}

return curr.isTerminal;

}

public boolean startsWith(String prefix) {

Node curr = root;

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

char ch = prefix.charAt(i);

if (curr.child.containsKey(ch)) {

curr = curr.child.get(ch);

} else {

return false;

}

}

return true;

}

}

public class Main {

public static void main(String[] args) {

Trie t = new Trie();

t.insert("hello");

t.insert("world");

System.out.println(t.search("hello")); // true

System.out.println(t.search("hell")); // false

System.out.println(t.search("world")); // true

System.out.println(t.search("wor")); // false

System.out.println(t.startsWith("hell")); // true

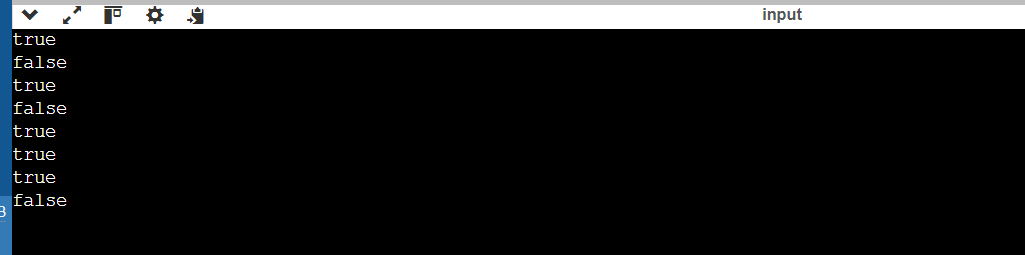
System.out.println(t.startsWith("wor")); // true

System.out.println(t.startsWith("hello")); // true

System.out.println(t.startsWith("helloworld")); // false

}

}



1. Given a string containing just the characters '(', ')', '{', '}', '[', and ']', determine if the input string is valid.

import java.util.\*;

public class Main

{

public static void main(String[] args) {

Scanner cs = new Scanner(System.in);

String s = cs.next();

System.out.println(isValid(s));

}

public static boolean isValid(String s) {

Stack<Character> st = new Stack<>();

for(int c:s.toCharArray()){

if(c=='(') st.push(')');

else if(c=='[') st.push(']');

else if(c=='{') st.push('}');

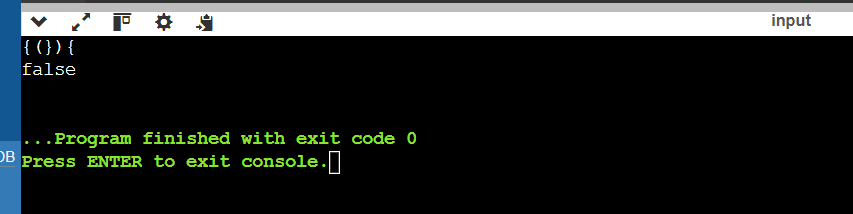
else if(st.isEmpty() || c!=st.pop()) return false;

}

return st.isEmpty();

}

}



1. Container With Most Water

import java.util.\*;

public class Main

{

public static void main(String[] args) {

Scanner cs = new Scanner(System.in);

int n = cs.nextInt();

int[] arr = new int[n];

for(int i=0;i<n;i++)

arr[i] = cs.nextInt();

System.out.println(maxArea(arr));

}

public static int maxArea(int[] arr) {

int start = 0;

int end = arr.length-1;

int maxarea = 0;

int area = 0;

while(start<end){

area = Math.min(arr[start],arr[end])\*(end-start);

if(area>maxarea) maxarea = area;

if(Math.min(arr[start],arr[end])==arr[start]) start++;

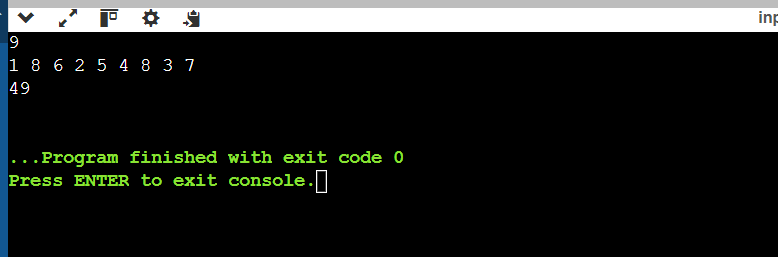
else end--;

}

return maxarea;

}

}



1. Find the kth largest element in an unsorted array.

import java.util.\*;

public class Main

{

public static void main(String[] args) {

Scanner cs = new Scanner(System.in);

int n = cs.nextInt();

int[] arr = new int[n];

for(int i=0;i<n;i++)

arr[i] = cs.nextInt();

int k = cs.nextInt();

System.out.println(findKthLargest(arr,k));

}

public static int findKthLargest(int[] nums, int k) {

PriorityQueue<Integer> q = new PriorityQueue<Integer>();

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

q.add(nums[i]);

}

for(int i=k;i<nums.length;i++){

if(nums[i]>q.peek()){

q.remove();

q.add(nums[i]);

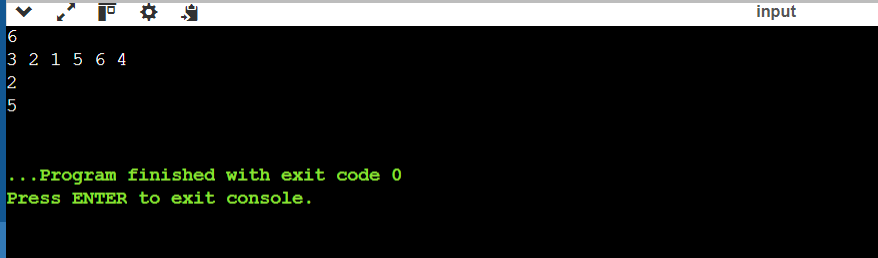
}

}

return q.peek();

}

}



1. Design an interval tree to efficiently find all intervals that overlap with a given interval.

Import java.util.\*;

public class Main {

static class Interval {

int low, high;

public Interval(int low, int high) {

this.low = low;

this.high = high;

}

public String toString() {

return "[" + this.low + "," + this.high + "]";

}

}

static class Node {

Interval range;

Node left, right;

public Node(Interval range) {

this.range = range;

}

public String toString() {

return "[" + this.range.low + ", "

+ this.range.high + "]\n";

}

}

public static Node insert(Node root, Interval x) {

if (root == null) {

return new Node(x);

}

if (x.low < root.range.low) {

root.left = insert(root.left, x);

} else {

root.right = insert(root.right, x);

}

return root;

}

public static Node deleteInterval(Node root, Interval x) {

if (root == null) {

return null;

}

if (x.low < root.range.low) {

root.left = deleteInterval(root.left, x);

} else if (x.low > root.range.low) {

root.right = deleteInterval(root.right, x);

} else {

if (root.left == null) {

return root.right;

} else if (root.right == null) {

return root.left;

}

Node minNode = findMin(root.right);

root.range = minNode.range;

root.right = deleteInterval(root.right, minNode.range);

}

return root;

}

public static Node findMin(Node root) {

while (root.left != null) {

root = root.left;

}

return root;

}

public static void inOrder(Node root) {

if (root == null) {

return;

}

inOrder(root.left);

System.out.print(root);

inOrder(root.right);

}

public static Interval isOverlapping(Node root, Interval x) {

if (root == null) {

return new Interval(-1, -1);

}

if ((x.low >= root.range.low && x.low <= root.range.high)

|| (x.high >= root.range.low && x.high <= root.range.high)) {

return root.range;

} else if (root.left != null) {

return isOverlapping(root.left, x);

} else {

return isOverlapping(root.right, x);

}

}

public static void main(String[] args) {

Node root = insert(null, new Interval(15, 20));

root = insert(root, new Interval(10, 30));

root = insert(root, new Interval(17, 19));

root = insert(root, new Interval(5, 20));

root = insert(root, new Interval(12, 15));

root = insert(root, new Interval(30, 40));

System.out.println("Inorder traversal of constructed Interval Tree is");

inOrder(root);

System.out.println();

Interval i = new Interval(6, 7);

System.out.println("Searching for interval " + i);

System.out.println("Overlaps with " + isOverlapping(root, i));

System.out.println();

System.out.println("Deleting interval [10, 30]");

root = deleteInterval(root, new Interval(10, 30));

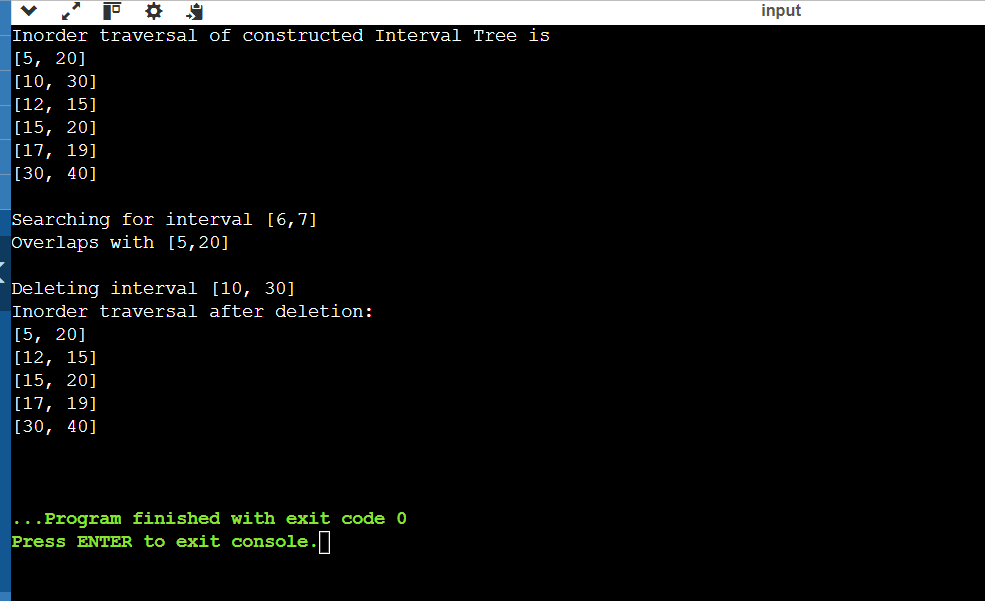
System.out.println("Inorder traversal after deletion:");

inOrder(root);

System.out.println();

}

}



1. Write a Java program that checks if a given string is a palindrome.

import java.util.\*;

public class Main

{

public static void main(String[] args) {

Scanner cs = new Scanner(System.in);

String s = cs.next();

System.out.println(isPalindrome(s));

}

public static boolean isPalindrome(String s){

int i=0;

int j = s.length()-1;

while(i<j){

if(s.charAt(i)!=s.charAt(j)) return false;

i++;

j--;

}

return true;

}

}

