Global Trend Assessment Questions ..

NAME: “PRIYANSHU KUMAR”;

“STUDENT OF B.TECH 3RD YEAR”

1. Design and implement a data structure for a Least Recently Used (LRU) cache. It should support the following operations: get and put.

get(key): Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

put(key, value): Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least recently used item before inserting a new item.

Ans = To design and implement a Least Recently Used (LRU) cache, we can use a combination of a hash map (dictionary) and a doubly linked list. The hash map provides O(1) time complexity for get and put operations, while the doubly linked list maintains the order of elements based on their usage.

import java.util.HashMap;

class LRUCache {

private class Node { //This inner class represents a node in the doubly linked list. It stores the key-value pair and pointers to the previous and next nodes.

int key, value;

Node prev, next;

Node(int key, int value) { // constructor of class node

this.key = key;

this.value = value;

}

}

private int capacity;

private HashMap<Integer, Node> cache;

private Node head, tail;

public LRUCache(int capacity) {

this.capacity = capacity;

this.cache = new HashMap<>();

this.head = new Node(0, 0);

this.tail = new Node(0, 0);

head.next = tail;

tail.prev = head;

}

public int get(int key) { // getter and setters are initialized

if (cache.containsKey(key)) {

Node node = cache.get(key);

remove(node);

add(node);

return node.value;

}

return -1;

}

public void put(int key, int value) {

if (cache.containsKey(key)) {

remove(cache.get(key));

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

cache.remove(tail.prev.key);

remove(tail.prev);

}

Node newNode = new Node(key, value);

add(newNode);

cache.put(key, newNode);

}

private void remove(Node node) { //Removes a node from the doubly linked list.

node.prev.next = node.next;

node.next.prev = node.prev;

}

private void add(Node node) { //Adds a node right after the head.

node.next = head.next;

node.prev = head;

head.next.prev = node;

head.next = node;

}

public static void main(String[] args) { // Main function where all functions are called.

LRUCache lruCache = new LRUCache(2);

lruCache.put(1, 1);

lruCache.put(2, 2);

System.out.println(lruCache.get(1)); // returns 1

lruCache.put(3, 3); // evicts key 2

System.out.println(lruCache.get(2)); // returns -1 (not found)

lruCache.put(4, 4); // evicts key 1

System.out.println(lruCache.get(1)); // returns -1 (not found)

System.out.println(lruCache.get(3)); // returns 3

System.out.println(lruCache.get(4)); // returns 4

}

}

2. Write a Java program that demonstrates the ConcurrentModificationException. Explain why the exception is thrown and how to handle it properly.

Ans = A ConcurrentModificationException in Java occurs when a collection (such as an ArrayList, HashSet, or HashMap) is modified while it is being iterated over in a way that is not allowed by the iterator. This often happens when one thread is iterating over a collection while another thread modifies it, or when you modify a collection directly within its iteration loop.

An example is as follows:

import java.util.ArrayList; // these three are import statements from the library

import java.util.Iterator;

import java.util.List;

public class ConcurrentModificationExample {

public static void main(String[] args) {

List<String> list = new ArrayList<>();// making a new list of type ArrayList.. it’s a data structure type..

list.add("one"); //these are adding statements for string type elements

list.add("two");

list.add("three");

for (String item : list) { // now a for each loop is used , loops or iterative methods are used to running a statement as many time as the condition goes true..

if ("two".equals(item)) {

list.remove(item); // This line will cause a ConcurrentModificationException

}

}

}

}

Now cure

There are 3 methods are as follows..

1.Use Iterator's ‘remove’ Method:

Instead of removing elements directly from the list, use the iterator's remove method.

2. Use ‘CopyOnWriteArrayList’:

For thread-safe operations where read operations are more frequent than write operations, you can use CopyOnWriteArrayList.

3. Synchronized Blocks:

If multiple threads are modifying the collection, you can synchronize the block of code that modifies the collection.

Now there representation is as follows:

1.import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

public class ConcurrentModificationExample {

public static void main(String[] args) {

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

list.add("one");

list.add("two");

list.add("three");

Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {

String item = iterator.next();

if ("two".equals(item)) {

iterator.remove();

}

}

}

}

//..

//2.

import java.util.Iterator;

import java.util.List;

import java.util.concurrent.CopyOnWriteArrayList;

public class ConcurrentModificationExample {

public static void main(String[] args) {

List<String> list = new CopyOnWriteArrayList<>();

list.add("one");

list.add("two");

list.add("three");

for (String item : list) {

if ("two".equals(item)) {

list.remove(item);

}

}

}

}

//..

//3.

import java.util.ArrayList;

import java.util.Collections;

import java.util.Iterator;

import java.util.List;

public class ConcurrentModificationExample {

public static void main(String[] args) {

List<String> list = Collections.synchronizedList(new ArrayList<>());

list.add("one");

list.add("two");

list.add("three");

synchronized (list) {

Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {

String item = iterator.next();

if ("two".equals(item)) {

iterator.remove(); // This is safe

}

}

}

}

}

//..

3. Create a custom annotation @LogExecutionTime to log the execution time of annotated methods. Implement an annotation processor to handle this annotation.

Ans = first creating

import java.lang.annotation.\*;

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface LogExecutionTime {

} //..

Then annotation processor

import javax.annotation.processing.\*;

import javax.lang.model.SourceVersion;

import javax.lang.model.element.\*;

import javax.tools.Diagnostic;

import java.util.Set;

@SupportedAnnotationTypes("LogExecutionTime")

@SupportedSourceVersion(SourceVersion.RELEASE\_8)

public class LogExecutionTimeProcessor extends AbstractProcessor {

@Override

public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {

for (Element element : roundEnv.getElementsAnnotatedWith(LogExecutionTime.class)) {

// Check if element is a method

if (element.getKind() == ElementKind.METHOD) {

String methodName = element.getSimpleName().toString();

processingEnv.getMessager().printMessage(Diagnostic.Kind.NOTE,

"Method '" + methodName + "' is annotated with @LogExecutionTime. Logging execution time...");

// Add your logging logic here (e.g., System.currentTimeMillis(), logging framework)

}

}

return true;

}

}//…

4. Design an algorithm to serialize and deserialize a binary tree. Implement serialize(TreeNode root) which converts a tree into a string, and deserialize(String data) which converts a string back to a tree.

Ans = Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

public class Codec {

public String serialize(TreeNode root) {

return serial(new StringBuilder(), root).toString();

}

// Generate preorder string

private StringBuilder serial(StringBuilder str, TreeNode root) {

if (root == null) return str.append("#");

str.append(root.val).append(",");

serial(str, root.left).append(",");

serial(str, root.right);

return str;

}

public TreeNode deserialize(String data) {

return deserial(new LinkedList<>(Arrays.asList(data.split(","))));

}

// Use queue to simplify position move

private TreeNode deserial(Queue<String> q) {

String val = q.poll();

if ("#".equals(val)) return null;

TreeNode root = new TreeNode(Integer.valueOf(val));

root.left = deserial(q);

root.right = deserial(q);

return root;

}

}

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

insert(word): Inserts a word into the trie.

search(word): Returns if the word is in the trie.

startsWith(prefix): Returns if there is any word in the trie that starts with the given prefix.

Ans = Trie : Implementations, Insert , Search & startWith methods are as follows.

Time complexity of all this code is O(n)..

public class Tries {

static class Node {

Node[] children = new Node[26];

boolean eow;

public Node() {

for (int i=0; i<26; i++) {

children[i] = null;

}

}

}

public static Node root = new Node();

public static void insert(String word) { // Word insert function

int level = 0;

int len = word.length();

int idx = 0;

Node curr = root;

for(; level<len; level++) {

idx = word.charAt(level)-'a';

if(curr.children[idx] == null) {

curr.children[idx] = new Node();

}

curr = curr.children[idx];

}

curr.eow = true;

}

public static boolean search(String key) { // Search function

int level = 0;

int len = key.length();

int idx = 0;

Node curr = root;

for(; level<len; level++) {

idx = key.charAt(level)-'a';

if(curr.children[idx] == null) {

return false;

}

curr = curr.children[idx];

}

return curr.eow == true;

}

public static boolean startsWith(String prefix) { // Starts with function

Node curr = root;

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

int idx = prefix.charAt(i)-'a';

if(curr.children[idx] == null) {

return false;

}

curr = curr.children[idx];

}

return true;

}

public static void main(String args[]) {

String words[] = {"the", "a", "there", "their", "any", "thee"};

for (String word : words) {

insert(word);

System.out.println("inserted " + word);

}

System.out.println("thee -> " + search("thee"));

System.out.println("thor -> " + search("thor"));

System.out.println(startsWith("the"));

System.out.println(startsWith("thi"));

}

}

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

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Ans = class Solution {

public boolean isValid(String s) {

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

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

if (c == '(')

stack.push(')');

else if (c == '{')

stack.push('}');

else if (c == '[')

stack.push(']');

else if (stack.isEmpty() || stack.pop() != c)

return false;

}

return stack.isEmpty();

}

}

7. Given n non-negative integers a1, a2, ..., an , where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of the line i are at (i, ai) and (i, 0). Find two lines, which together with the x-axis forms a container, such that the container contains the most water.

Ans = class Solution {

public int maxArea(int[] height) {

int min =0;

int midd = height.length-1;

int max = 0;

while (min < midd) {

int currentArea = Math.min(height[min], height[midd]) \* (midd - min);

max = Math.max(max, currentArea);

if (height[min] < height[midd]) {

min++;

} else {

midd--;

}

}

return max;

}

}

8. Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Ans = public class Solution {

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

Arrays.sort(nums);

return nums[nums.length - k];

}

}

9. Design an interval tree to efficiently find all intervals that overlap with a given interval. Implement the following operations:

insertInterval(int start, int end): Insert a new interval [start, end] into the tree.

deleteInterval(int start, int end): Delete an interval [start, end] from the tree.

findOverlappingIntervals(int start, int end): Return a list of all intervals that overlap with the interval [start, end].

Constraints

The intervals are represented as pairs of integers [start, end] where start ≤ end.

Ans = my apology but that’s I cant solve now but I assure you I will solve after some guidance and some time that’s till now..

10. Write a Java program that checks if a given string is a palindrome. A palindrome is a word, phrase, number, or other sequences of characters that reads the same forward and backward (ignoring spaces, punctuation, and capitalization).

Ans = class Solution {

public boolean isPalindrome(String s) {

if (s.isEmpty()) {

return true;

}

int start = 0;

int last = s.length() - 1;

while(start <= last) {

char currFirst = s.charAt(start);

char currLast = s.charAt(last);

if (!Character.isLetterOrDigit(currFirst )) {

start++;

} else if(!Character.isLetterOrDigit(currLast)) {

last--;

} else {

if (Character.toLowerCase(currFirst) != Character.toLowerCase(currLast)) {

return false;

}

start++;

last--;

}

}

return true;

}

}

//…

MY SINCEARLY APOLOGY FOR ANY INCONVINEANCE..

THAT’S ALL BY MY SIDE , I AM GRATEFUL YOU GIVE ME SOME KNOWLEDGE …