**Experiment - 10**

**Name: Armaan UID: 22BCS12007**

**Aim:** To solve leet code problems

1. Problem : Pascal’s Triangle

Code:

class Solution {

public List<List<Integer>> generate(int numRows) {

List<List<Integer>> res = new ArrayList<>();

res.add(List.of(1));

for (int i = 0; i < numRows - 1; i++) {

List<Integer> dummyRow = new ArrayList<>();

dummyRow.add(0);

dummyRow.addAll(res.get(res.size() - 1));

dummyRow.add(0);

List<Integer> row = new ArrayList<>();

for (int j = 0; j < dummyRow.size() - 1; j++) {

row.add(dummyRow.get(j) + dummyRow.get(j + 1));

}

res.add(row);

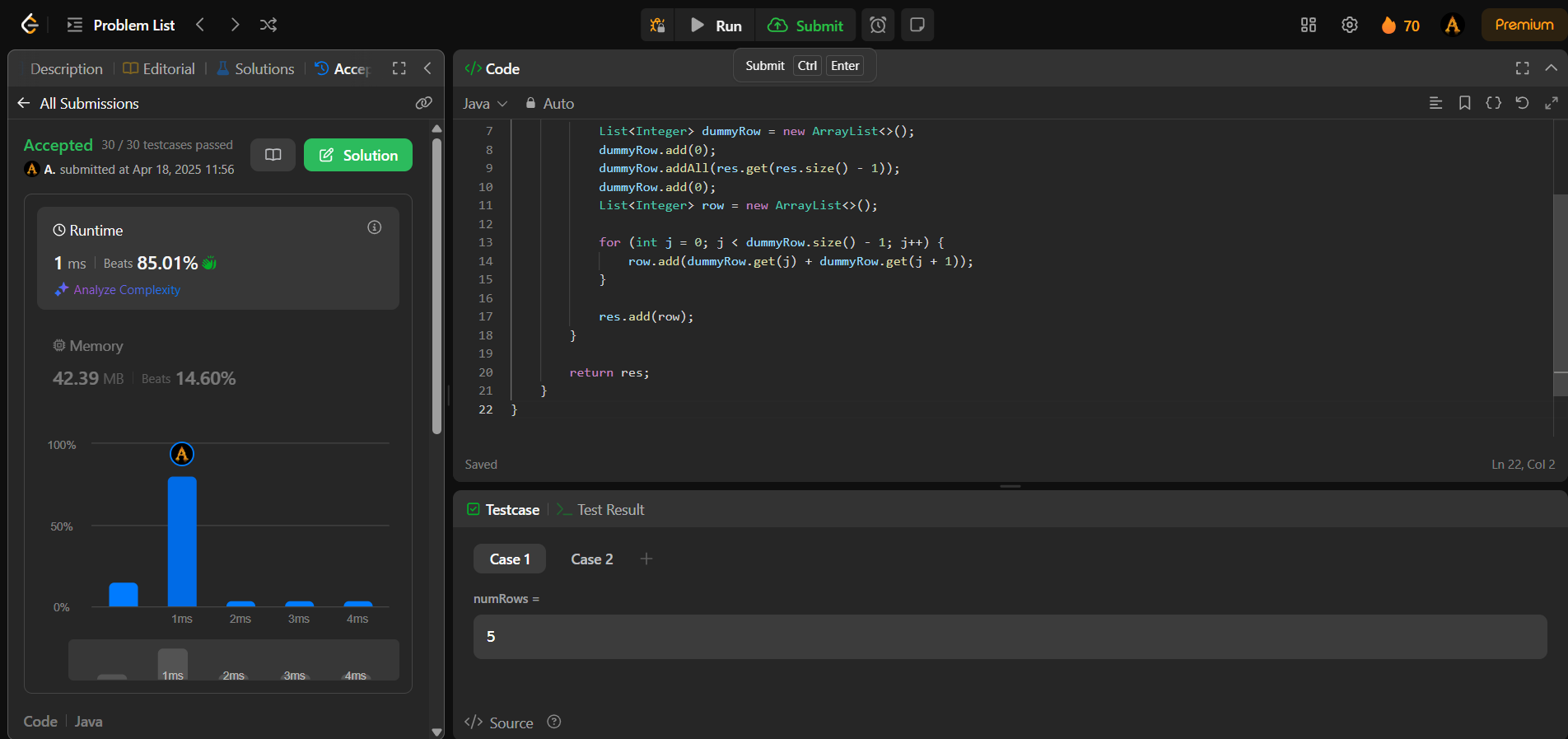
}

return res;

}

}

Output :



1. Problem: Hamming Distance

Code:

class Solution {

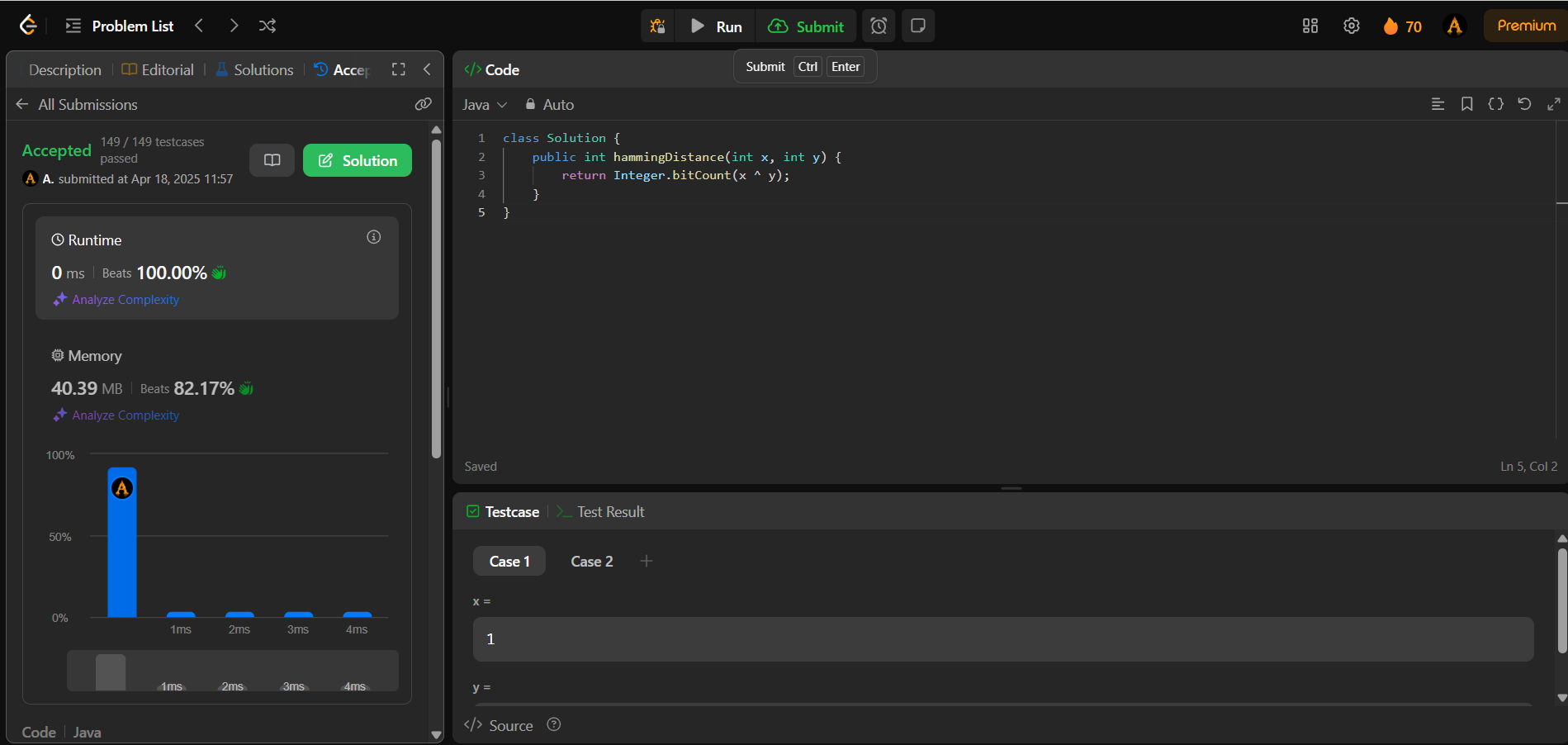
public int hammingDistance(int x, int y) {

return Integer.bitCount(x ^ y);

}

}

Output:



1. Problem: Task Scheduler

Code:

public class Solution {

public int leastInterval(char[] tasks, int n) {

int[] counter = new int[26];

int max = 0;

int maxCount = 0;

for(char task : tasks) {

counter[task - 'A']++;

if(max == counter[task - 'A']) {

maxCount++;

}

else if(max < counter[task - 'A']) {

max = counter[task - 'A'];

maxCount = 1;

}

}

int partCount = max - 1;

int partLength = n - (maxCount - 1);

int emptySlots = partCount \* partLength;

int availableTasks = tasks.length - max \* maxCount;

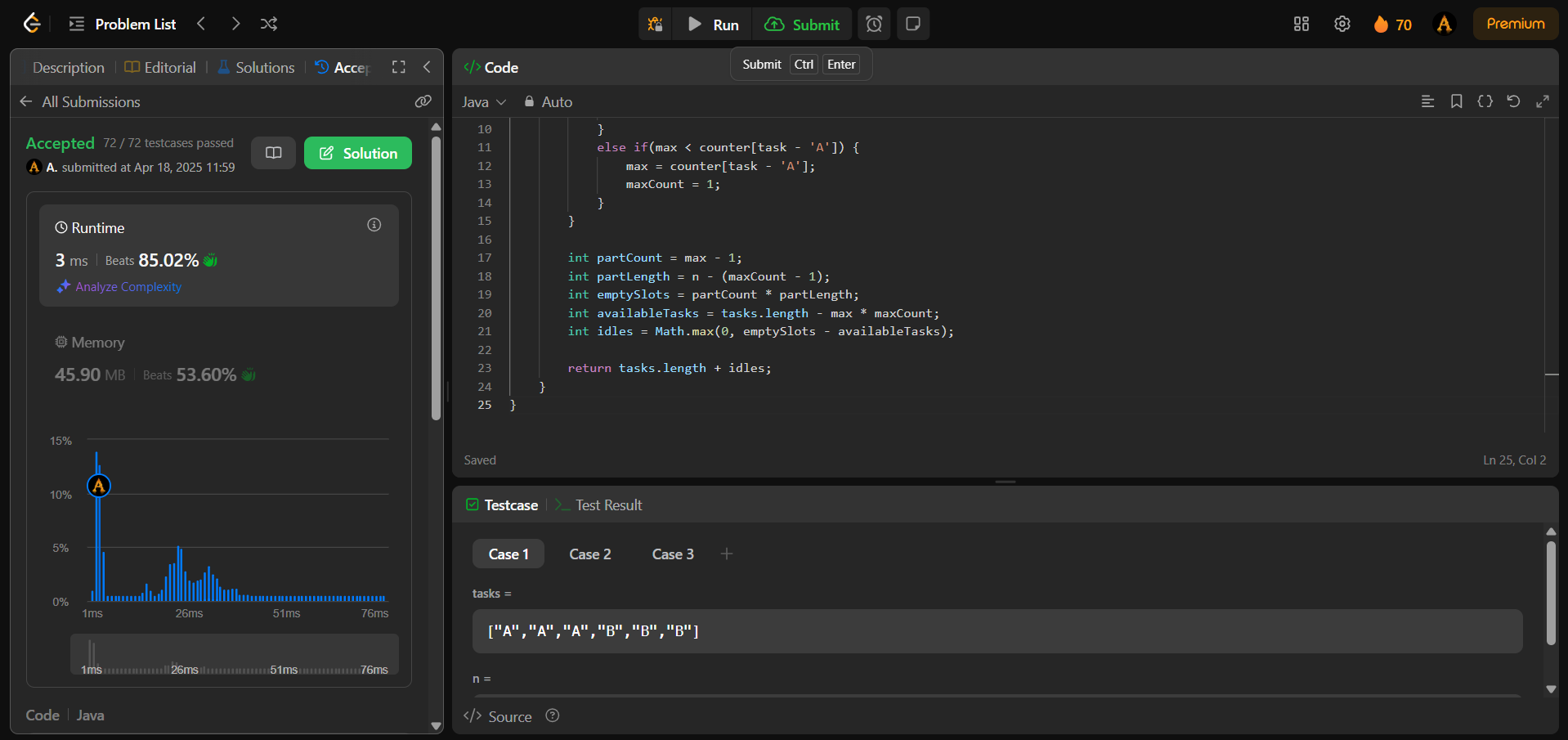
int idles = Math.max(0, emptySlots - availableTasks);

return tasks.length + idles;

}

}

Output:



1. Problem: Number of 1 bits

Code:

class Solution {

public int hammingWeight(int n) {

String x = Integer.toBinaryString(n);

int count =0;

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

if(x.charAt(i)=='1'){

count++;

}

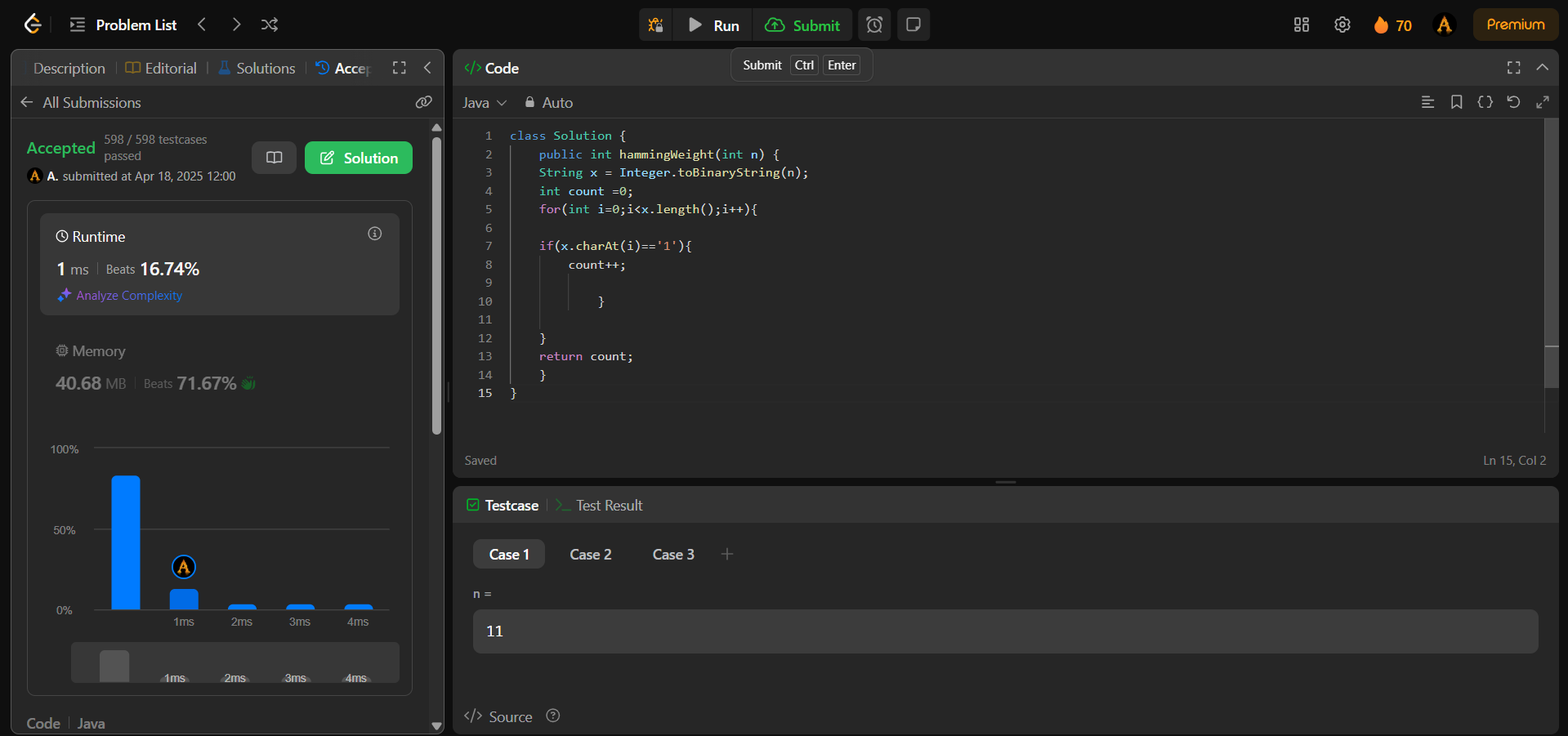
}

return count;

}

}

Output:



1. Problem: Divide two Integers

Code:

class Solution {

public int divide(int dividend, int divisor) {

if (dividend == Integer.MIN\_VALUE && divisor == -1) return Integer.MAX\_VALUE; //Cornor case when -2^31 is divided by -1 will give 2^31 which doesnt exist so overflow

boolean negative = dividend < 0 ^ divisor < 0; //Logical XOR will help in deciding if the results is negative only if any one of them is negative

dividend = Math.abs(dividend);

divisor = Math.abs(divisor);

int quotient = 0, subQuot = 0;

while (dividend - divisor >= 0) {

for (subQuot = 0; dividend - (divisor << subQuot << 1) >= 0; subQuot++);

quotient += 1 << subQuot; //Add to the quotient

dividend -= divisor << subQuot; //Substract from dividend to start over with the remaining

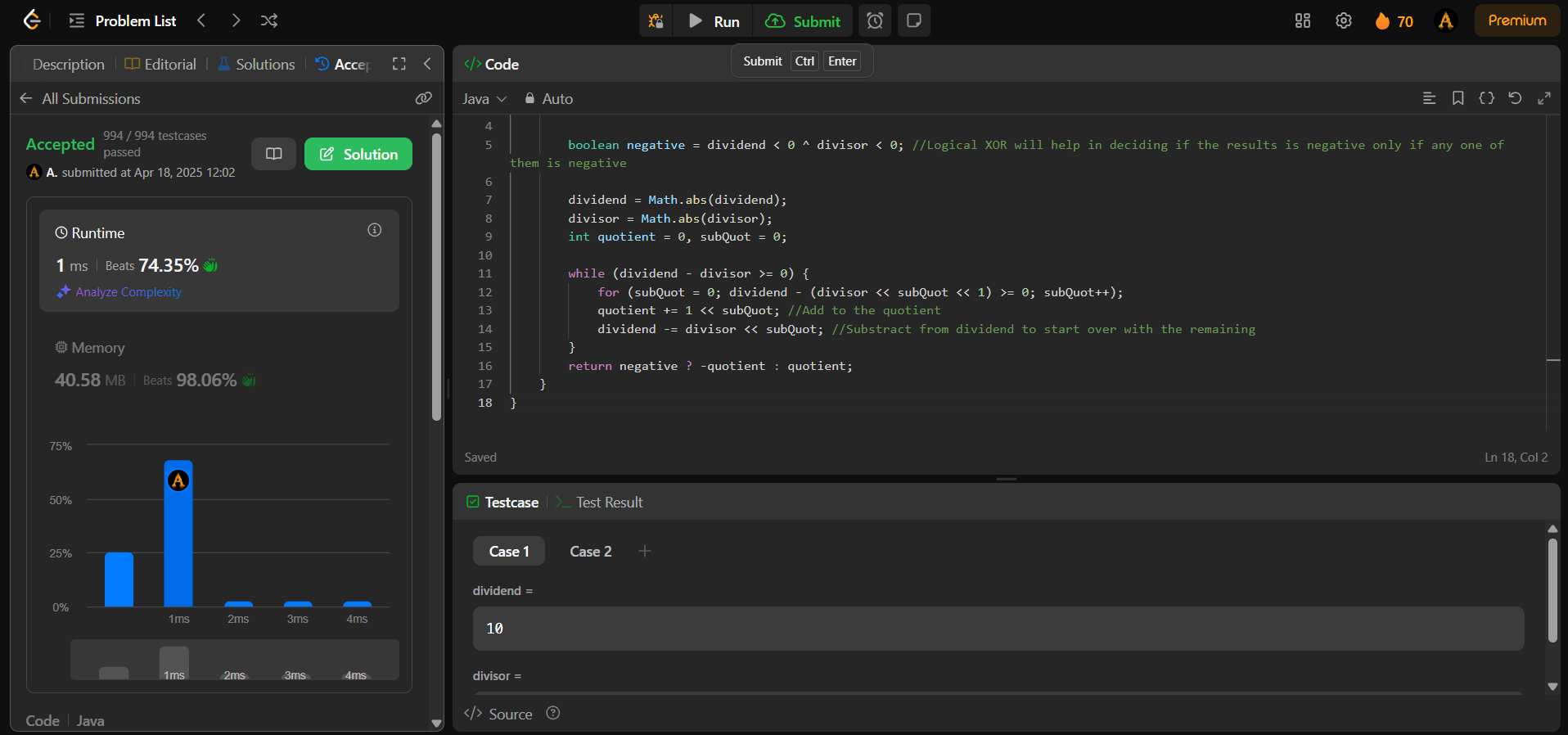
}

return negative ? -quotient : quotient;

}

}

Output:



1. Problem : Trapping Rainwater

Code:

class Solution {

public int trap(int[] height) {

int left = 0;

int right = height.length - 1;

int leftMax = height[left];

int rightMax = height[right];

int water = 0;

while (left < right) {

if (leftMax < rightMax) {

left++;

leftMax = Math.max(leftMax, height[left]);

water += leftMax - height[left];

} else {

right--;

rightMax = Math.max(rightMax, height[right]);

water += rightMax - height[right];

}

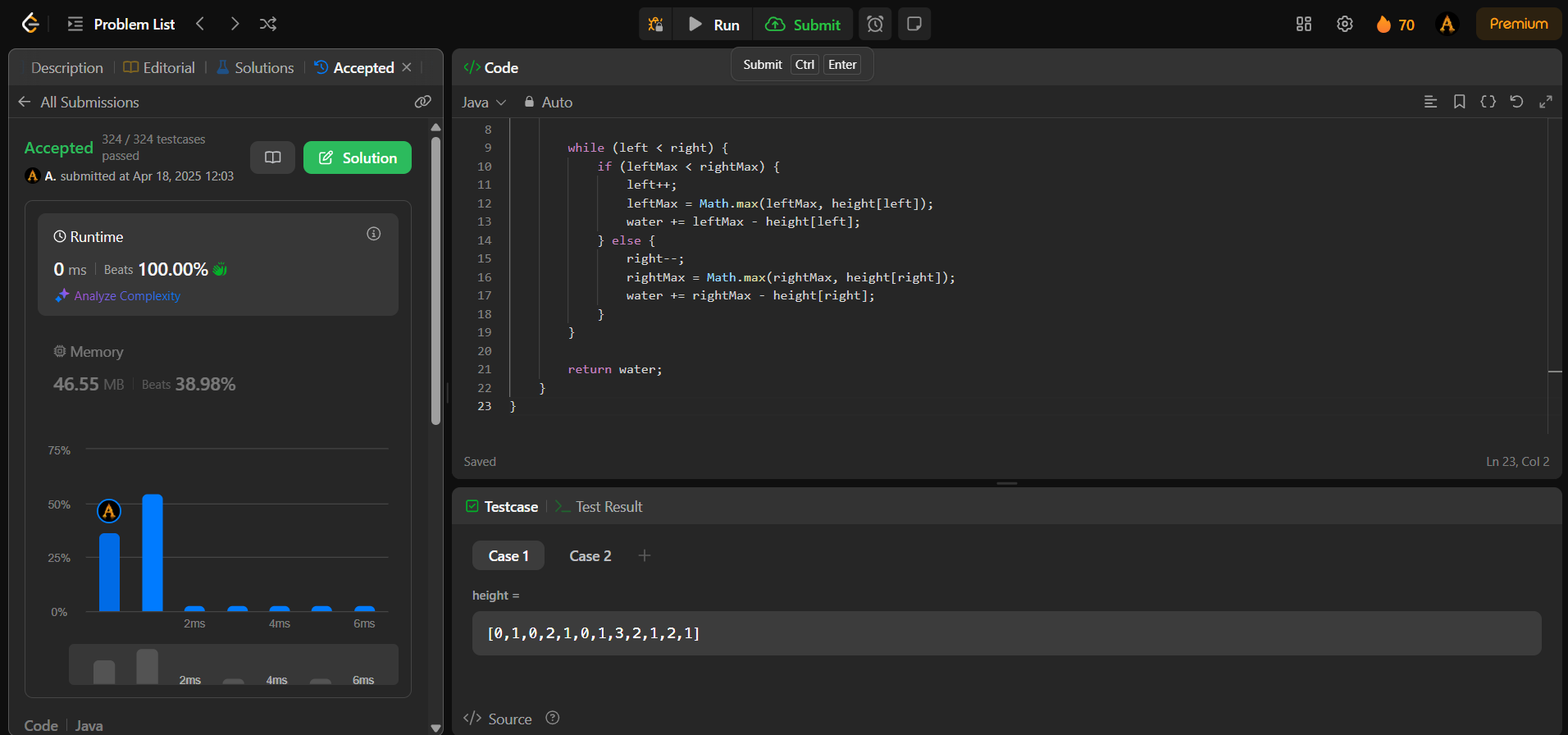
}

return water;

}

}

Output:



1. Problem: Max Number of Tasks You Can Assign

Code:

class Solution {

    public int maxTaskAssign(int[] tasks, int[] workers, int pills, int strength) {

        int left = 0, right = Math.min(tasks.length, workers.length);

        Arrays.sort(tasks);

        Arrays.sort(workers);

        while(left+1<right)

        {

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

            if(canAssign(mid, tasks, workers, pills, strength))

            {

                left = mid;

            }

            else

            {

                right = mid;

            }

        }

        if(canAssign(right, tasks, workers, pills, strength))

        {

            return right;

        }

        else return left;

    }

      public boolean canAssign(int count, int[] tasks, int[] workers, int pills, int strength){

        Deque<Integer> dq = new ArrayDeque<>();

        int ind = workers.length - 1;

        for (int i = count - 1; i >= 0; i--) {

            while(ind>=workers.length-count && workers[ind]+strength>=tasks[i])

            {

              dq.offerLast(workers[ind]);

              ind--;

            }

            if(dq.isEmpty())return false;

            if(dq.peekFirst()>=tasks[i])

            {

                dq.pollFirst();

            }

            else

            {

                dq.pollLast();

                pills--;

                if(pills<0)return false;

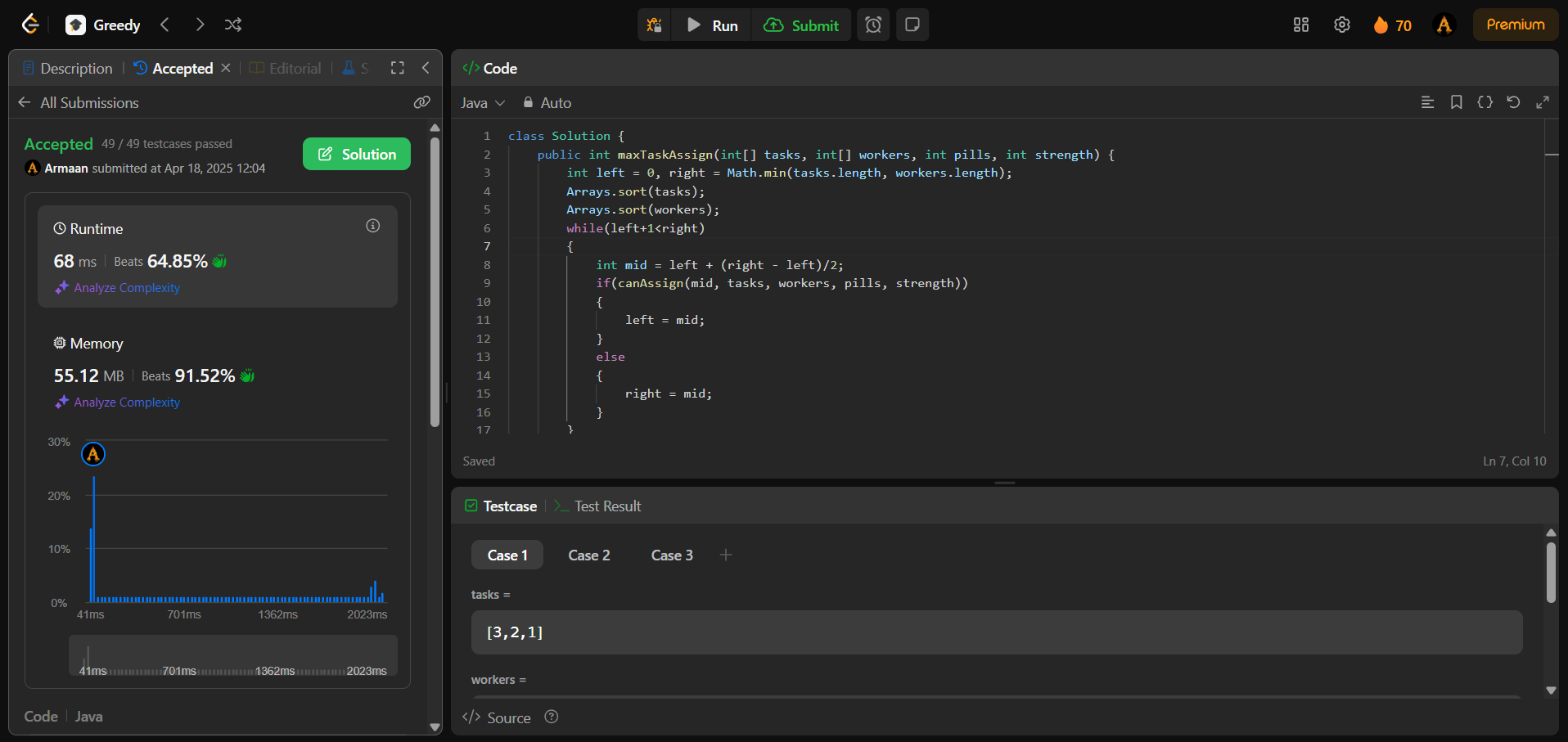
            }

        }

        return true;

    }

}

Output:

1. Problem: Serialize and Deserialize Binary Tree

Code:

public class Codec {

public String serialize(TreeNode root) {

if (root == null) return "#";

return root.val + "," + serialize(root.left) + "," + serialize(root.right);

}

public TreeNode deserialize(String data) {

Queue<String> queue = new LinkedList<>(Arrays.asList(data.split(",")));

return helper(queue);

}

private TreeNode helper(Queue<String> queue) {

String s = queue.poll();

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

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

root.left = helper(queue);

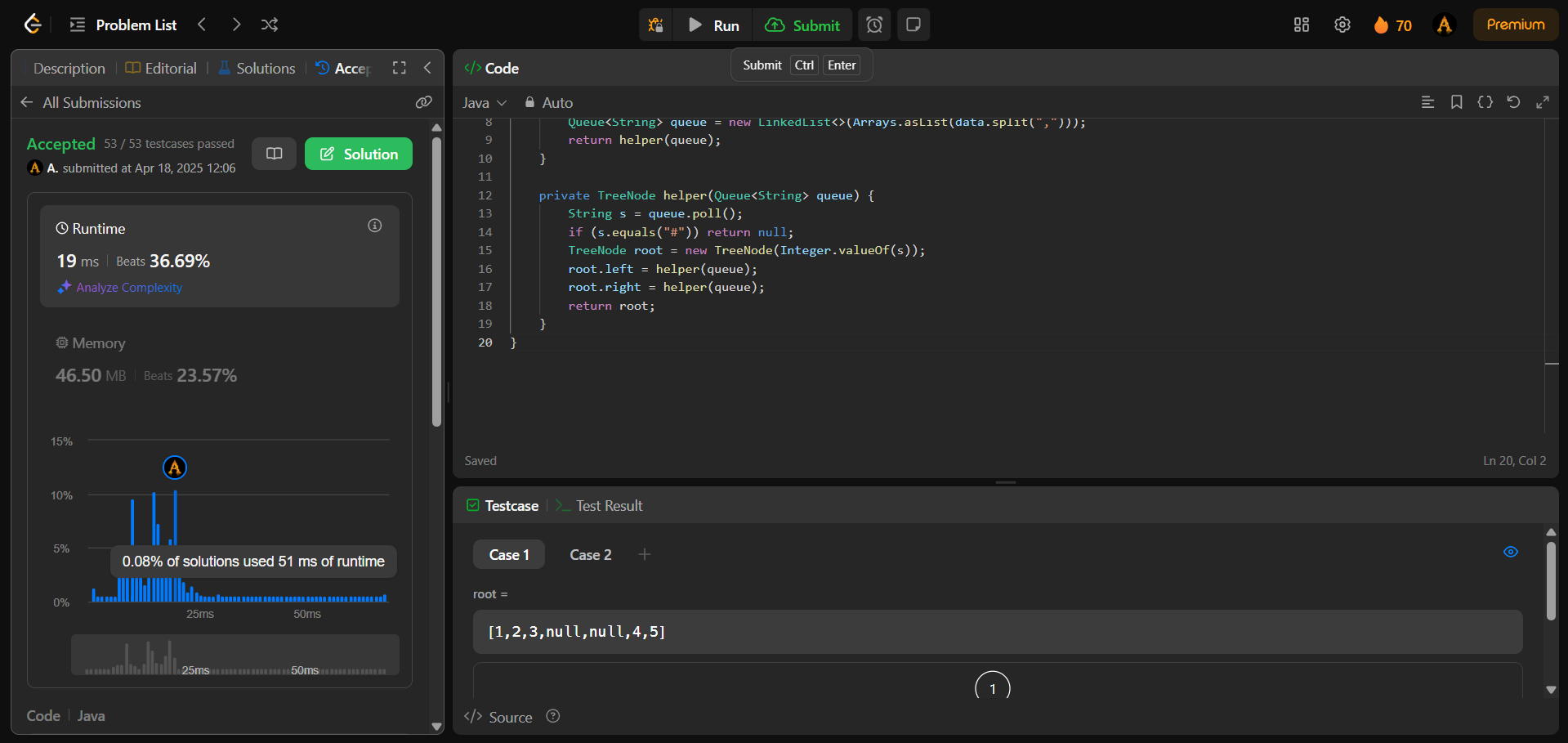
root.right = helper(queue);

return root;

}

}

Output:



1. Problem : LRU cache

Code:

class Node {

int key;

int val;

Node prev;

Node next;

public Node(int key, int val) {

this.key = key;

this.val = val;

this.prev = null;

this.next = null;

}

}

class LRUCache {

private int cap;

private Map<Integer, Node> cache;

private Node oldest;

private Node latest;

public LRUCache(int capacity) {

this.cap = capacity;

this.cache = new HashMap<>();

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

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

this.oldest.next = this.latest;

this.latest.prev = this.oldest;

}

public int get(int key) {

if (cache.containsKey(key)) {

Node node = cache.get(key);

remove(node);

insert(node);

return node.val;

}

return -1;

}

private void remove(Node node) {

Node prev = node.prev;

Node next = node.next;

prev.next = next;

next.prev = prev;

}

private void insert(Node node) {

Node prev = latest.prev;

Node next = latest;

prev.next = next.prev = node;

node.next = next;

node.prev = prev;

}

public void put(int key, int value) {

if (cache.containsKey(key)) {

remove(cache.get(key));

}

Node newNode = new Node(key, value);

cache.put(key, newNode);

insert(newNode);

if (cache.size() > cap) {

Node lru = oldest.next;

remove(lru);

cache.remove(lru.key);

}

}

}

Output :

