Advanced Programming LAB II

EXPERIMENT - 10

Submitted by,

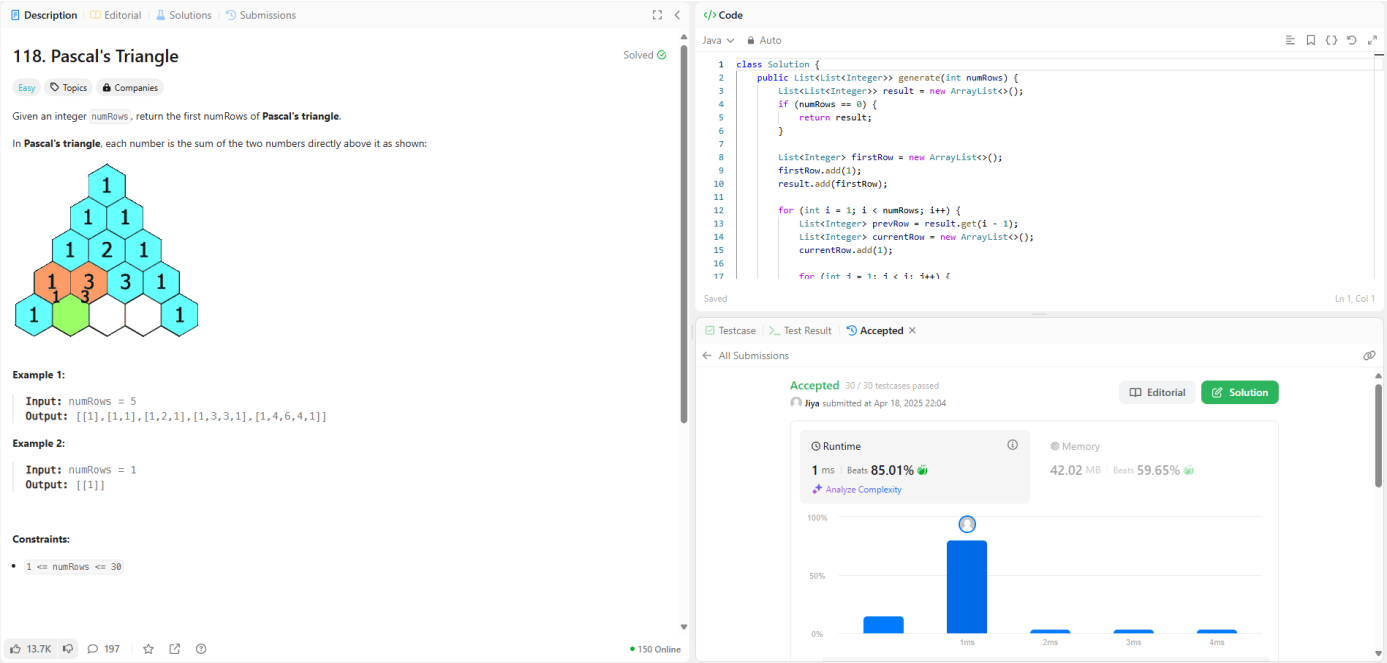
Jiya | 22BCS14856

22BCS_FL_IOT-601 (A)

118. Pascal's Triangle

https://leetcode.com/problems/pascals-triangle/

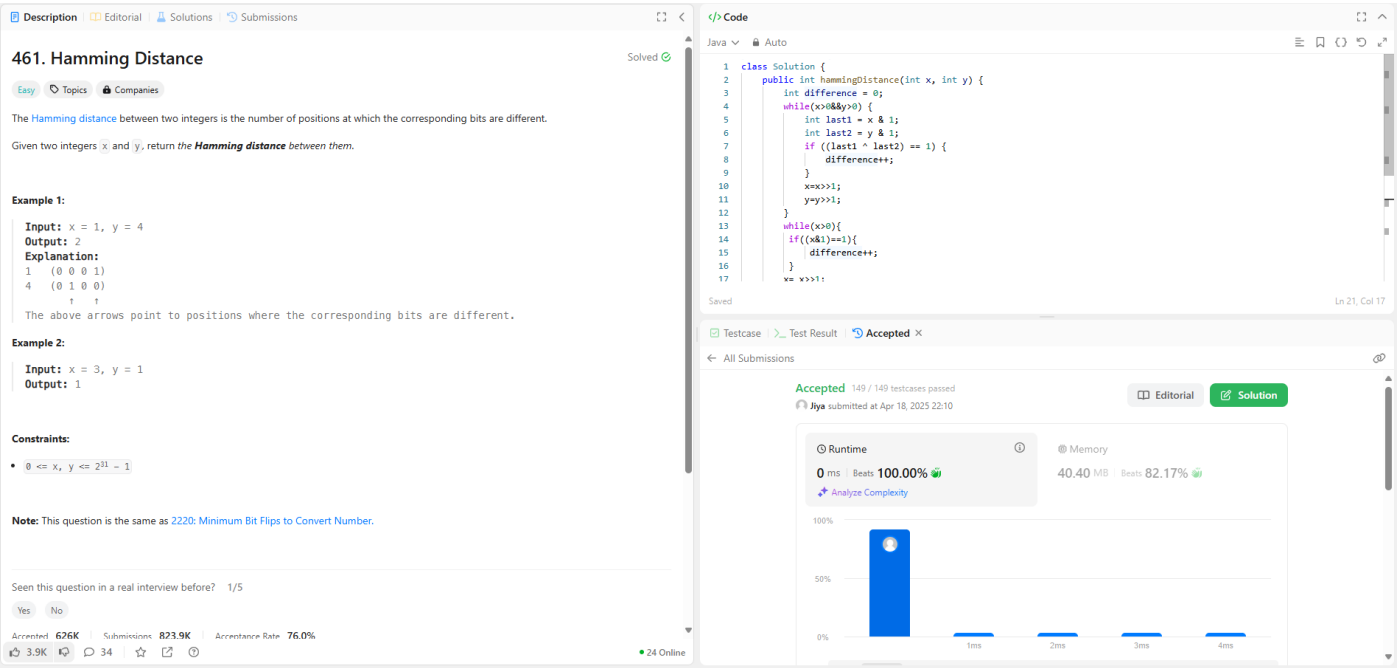
```java
import java.util.ArrayList;
import java.util.List;

class Solution {
    public List<List<Integer>> generate(int numRows) {
        List<List<Integer>> result = new ArrayList<>();
        if (numRows == 0) {
            return result;
        }

        List<Integer> firstRow = new ArrayList<>();
        firstRow.add(1);
        result.add(firstRow);

        for (int i = 1; i < numRows; i++) {
            List<Integer> prevRow = result.get(i - 1);
            List<Integer> currentRow = new ArrayList<>();
            currentRow.add(1);

            for (int j = 1; j < i; j++) {
                currentRow.add(prevRow.get(j - 1) + prevRow.get(j));
            }

            currentRow.add(1);
            result.add(currentRow);
        }

        return result;
    }
}
```



461. Hamming Distance

https://leetcode.com/problems/hamming-distance/

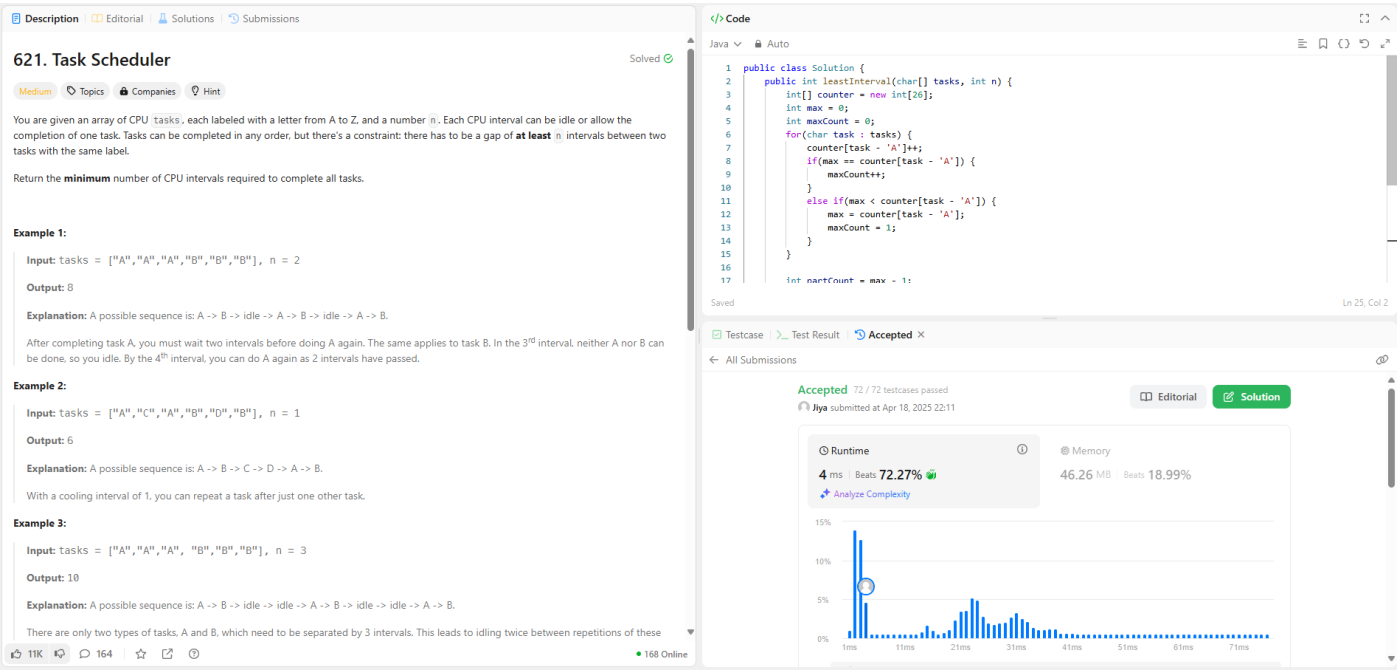
```
class Solution {
    public int hammingDistance(int x, int y) {
        int difference = 0;
        while(x>0&&y>0) {
            int last1 = x & 1;
            int last2 = y & 1;
            if ((last1 ^ last2) == 1) {
                difference++;
            }
            x=x>>1;
            y=y>>1;
        }
        while(x>0){
         if((x&1)==1){
             difference++;
         }
        x= x>>1;
        }
        while(y>0){
            if((y&1)==1){
                difference++;
            }
            y=y>>1;
        }
      return difference;
    }
}
```



621. Task Scheduler

https://leetcode.com/problems/task-scheduler/description/

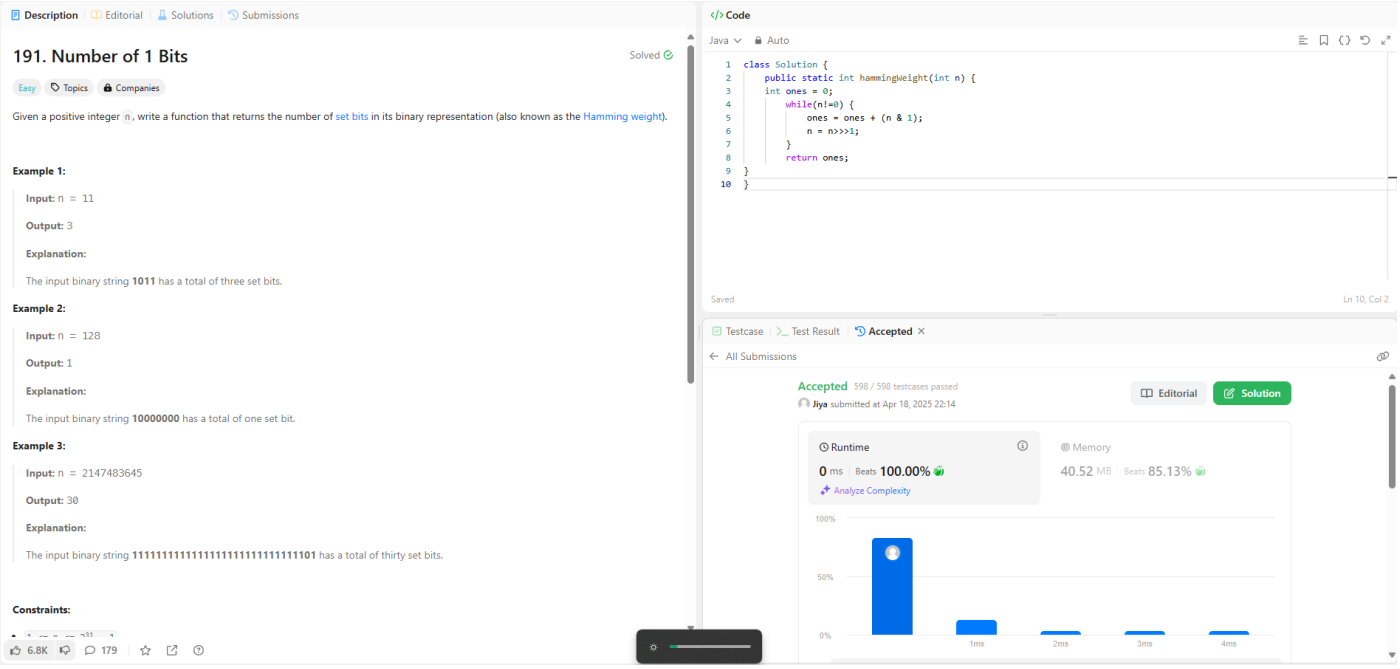
```
public class Solution {
    public int leastInterval(char[] tasks, int n) {
        int[] counter = new int[26];
        int max = 0;
        int maxCount = 0;
        for(char task : tasks) {
            counter[task - 'A']++;
            if(max == counter[task - 'A']) {
                maxCount++;
            }
            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;
    }
}
```



191. Number of 1 Bits

https://leetcode.com/problems/number-of-1-bits/description/

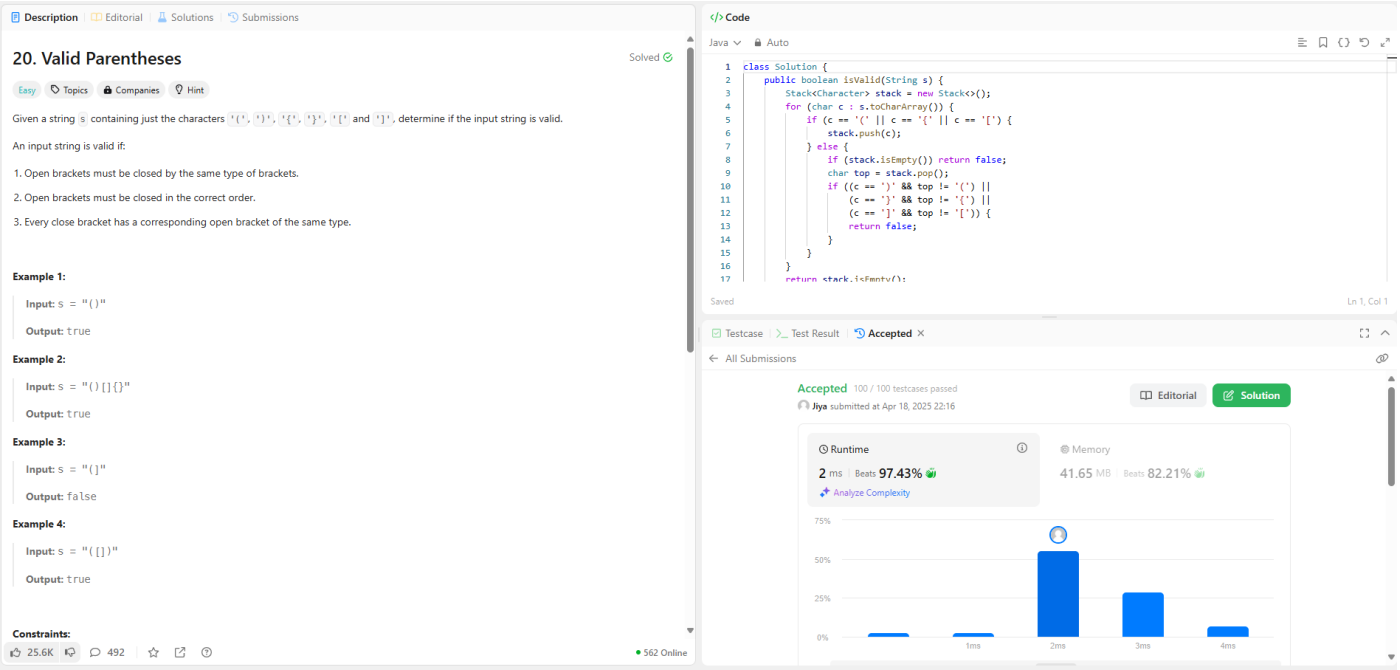
```
class Solution {
    public static int hammingWeight(int n) {
    int ones = 0;
        while(n!=0) {
            ones = ones + (n & 1);
            n = n>>>1;
        }
        return ones;
}
}
```



20. Valid Parentheses

https://leetcode.com/problems/valid-parentheses/description/

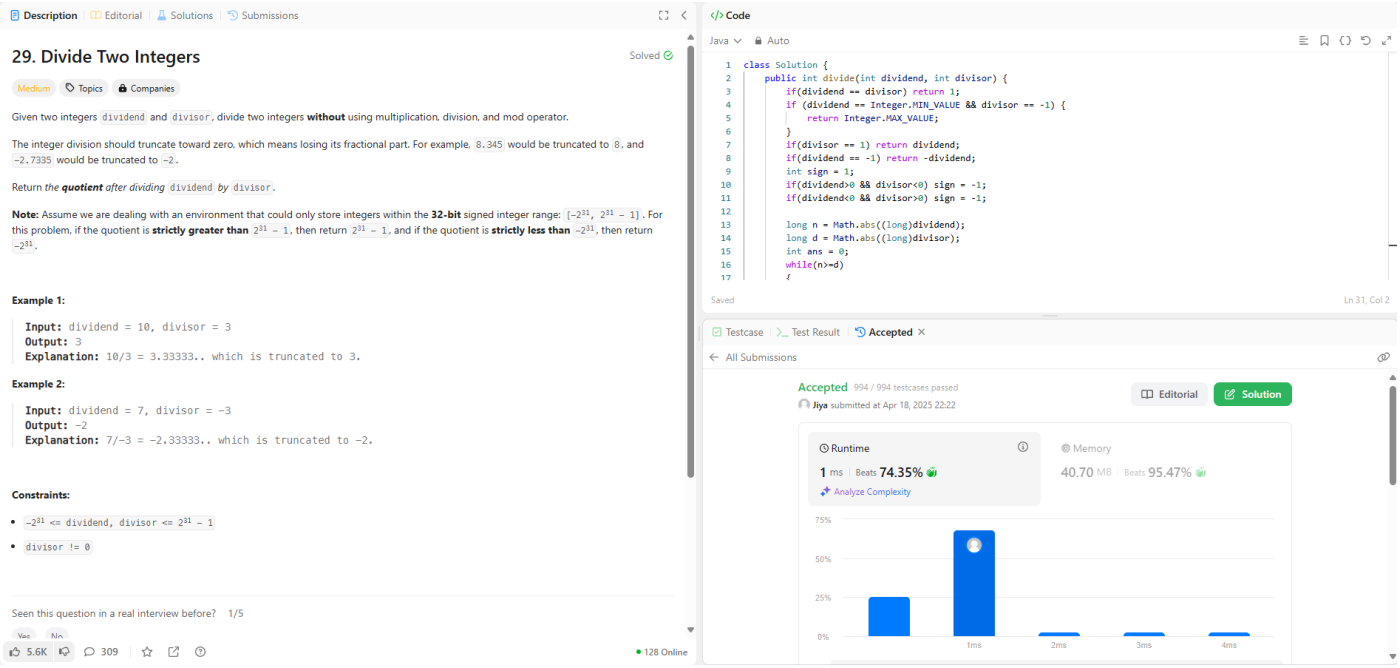
```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        for (char c : s.toCharArray()) {
            if (c == '(' || c == '{' || c == '[') {
                stack.push(c);
            } else {
                if (stack.isEmpty()) return false;
                char top = stack.pop();
                if ((c == ')' && top != '(') ||
                    (c == '}' && top != '{') ||
                    (c == ']' && top != '[')) {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }
}
```



29. Divide Two Integers

https://leetcode.com/problems/divide-two-integers/description/

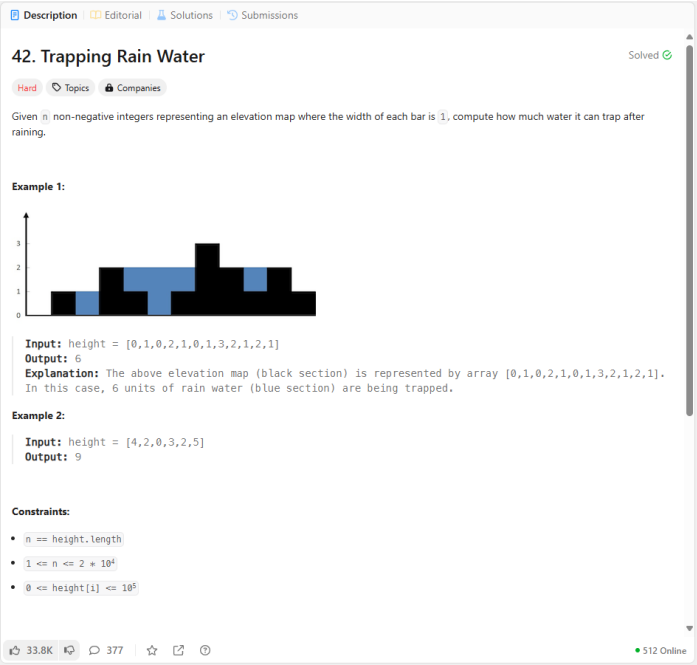
```
class Solution {
    public int divide(int dividend, int divisor) {
        if(dividend == divisor) return 1;
        if (dividend == Integer.MIN_VALUE && divisor == -1) {
            return Integer.MAX_VALUE;
        }
        if(divisor == 1) return dividend;
        if(dividend == -1) return -dividend;
        int sign = 1;
        if(dividend>0 && divisor<0) sign = -1;
        if(dividend<0 && divisor>0) sign = -1;

        long n = Math.abs((long)dividend);
        long d = Math.abs((long)divisor);
        int ans = 0;
        while(n>=d)
        {
            int p = 0;
            while(n >= d<<p)
            p++;

            p--;
            n -= d<<p;
            ans += 1<<p;
        }
        if(ans>=Math.pow(2,31) && sign==1) return Integer.MAX_VALUE;
        if(ans>=Math.pow(2,31) && sign==-1) return Integer.MIN_VALUE;

        return ans*sign;
    }
}
```

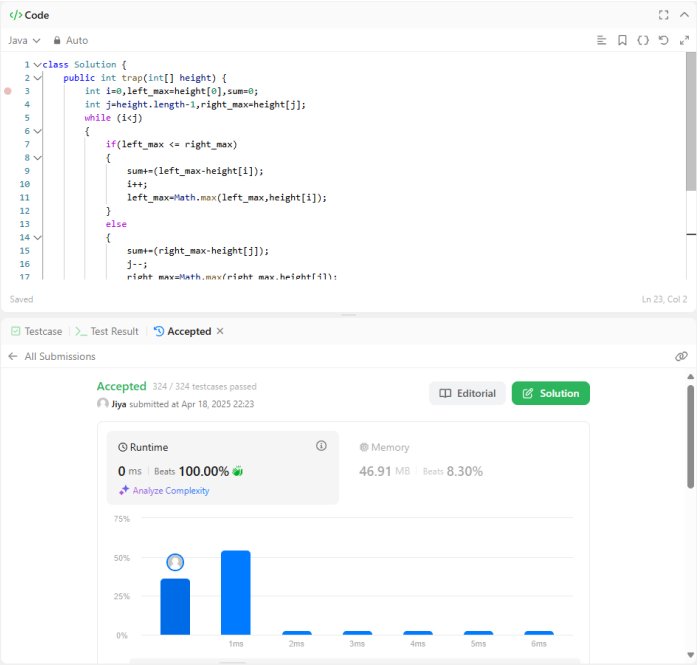


42. Trapping Rainwater

https://leetcode.com/problems/trapping-rain-water/description/

```
class Solution {
    public int trap(int[] height) {
        int i=0,left_max=height[0],sum=0;
        int j=height.length-1,right_max=height[j];
        while (i<j)
        {
            if(left_max <= right_max)
            {
                sum+=(left_max-height[i]);
                i++;
                left_max=Math.max(left_max,height[i]);
            }
            else
            {
                sum+=(right_max-height[j]);
                j--;
                right_max=Math.max(right_max,height[j]);
            }
        }
        return sum;

    }
}
```

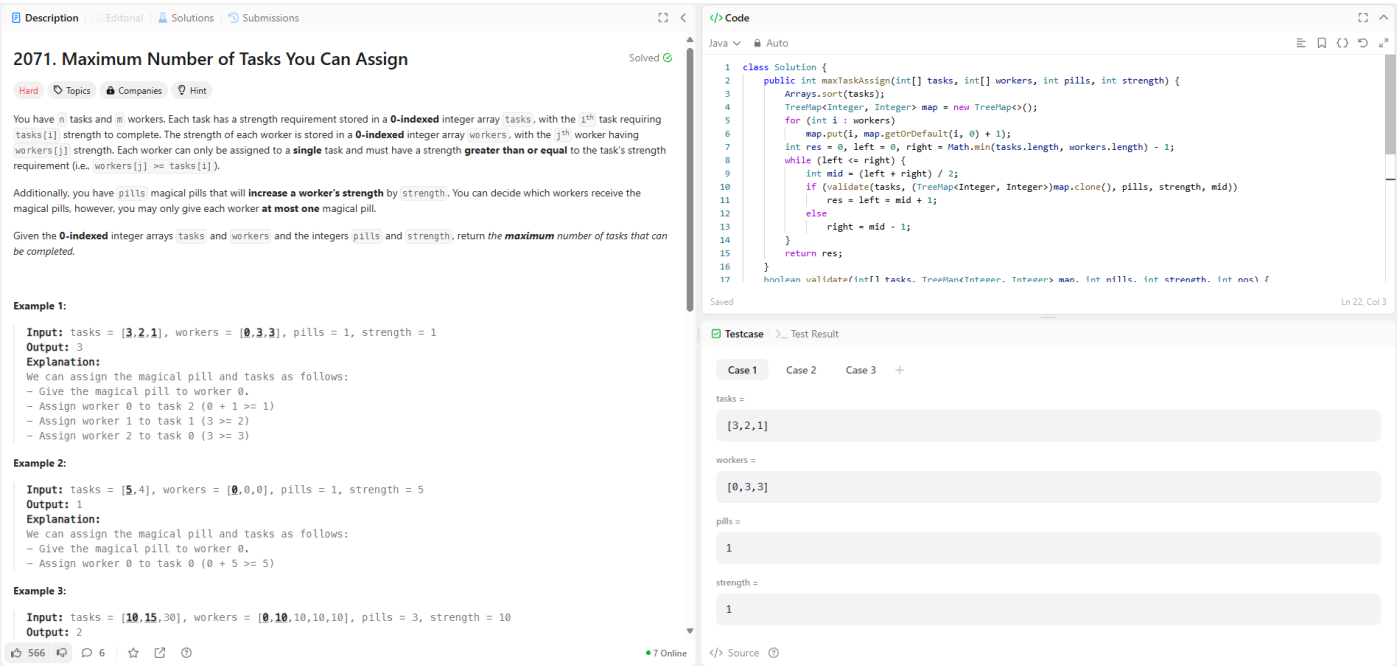




2071. Maximum Number of Tasks You Can Assign

https://leetcode.com/problems/maximum-number-of-tasks-you-can-assign/description/

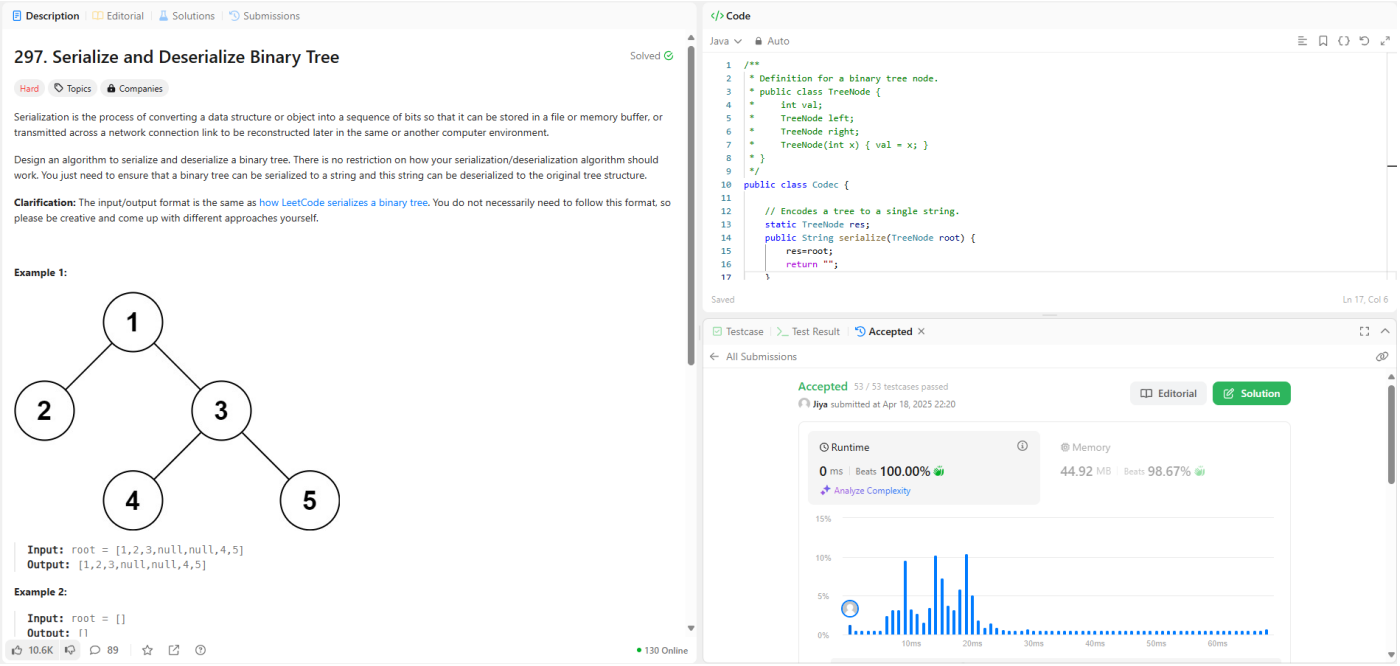
```
class Solution {
    public int maxTaskAssign(int[] tasks, int[] workers, int pills, int strength) {
        Arrays.sort(tasks);
        TreeMap<Integer, Integer> map = new TreeMap<>();
        for (int i : workers)
            map.put(i, map.getOrDefault(i, 0) + 1);
        int res = 0, left = 0, right = Math.min(tasks.length, workers.length) - 1;
        while (left <= right) {
            int mid = (left + right) / 2;
            if (validate(tasks, (TreeMap<Integer, Integer>)map.clone(), pills,
strength, mid))
                res = left = mid + 1;
            else
                right = mid - 1;
        }
        return res;
    }
    boolean validate(int[] tasks, TreeMap<Integer, Integer> map, int pills, int
strength, int pos) {
        for (; pos >= 0; pos--) {
            int maxStrength = map.lastKey(), t = tasks[pos];
            if (pills > 0 && strength + maxStrength < t || pills == 0 && maxStrength
< t)
                return false;
            if (maxStrength < t) {
                t -= strength;
                pills--;
            }
            int matchStrength = map.ceilingKey(t);
            if (map.get(matchStrength) > 1)
                map.put(matchStrength, map.get(matchStrength) - 1);
            else
                map.remove(matchStrength);
        }
        return true;
    }
}
```



297. Serialize and Deserialize Binary Tree

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/description/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Codec {

    // Encodes a tree to a single string.
    static TreeNode res;
    public String serialize(TreeNode root) {
        res=root;
        return "";
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        return res;
    }
}

// Your Codec object will be instantiated and called as such:
// Codec ser = new Codec();
// Codec deser = new Codec();
// TreeNode ans = deser.deserialize(ser.serialize(root));
```



146. LRU Cache

https://leetcode.com/problems/lru-cache/description/

```
public class LRUCache {
class DLinkedNode {
  int key;
  int value;
  DLinkedNode pre;
  DLinkedNode post;
}
/**
 * Always add the new node right after head;
 */
private void addNode(DLinkedNode node) {
  node.pre = head;
  node.post = head.post;

  head.post.pre = node;
  head.post = node;
}
/**
 * Remove an existing node from the linked list.
 */
private void removeNode(DLinkedNode node){
  DLinkedNode pre = node.pre;
  DLinkedNode post = node.post;

  pre.post = post;
  post.pre = pre;
}
/**
 * Move certain node in between to the head.
 */
private void moveToHead(DLinkedNode node){
  this.removeNode(node);
  this.addNode(node);
}
// pop the current tail.
private DLinkedNode popTail(){
  DLinkedNode res = tail.pre;
  this.removeNode(res);
  return res;
}
private Hashtable<Integer, DLinkedNode>
cache = new Hashtable<Integer, DLinkedNode>();
private int count;
private int capacity;
private DLinkedNode head, tail;

public LRUCache(int capacity) {
  this.count = 0;
  this.capacity = capacity;

  head = new DLinkedNode();
  head.pre = null;

  tail = new DLinkedNode();
  tail.post = null;

  head.post = tail;
  tail.pre = head;
}
public int get(int key) {
  DLinkedNode node = cache.get(key);
  if(node == null){
    return -1; // should raise exception here.
  }
  this.moveToHead(node);
  return node.value;
}
```

Description | Editorial | Solutions | Submissions

146. LRU Cache

Solved

Medium | Topics | Companies

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `-1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

The functions `get` and `put` must each run in `O(1)` average time complexity.

Example 1:

Input
```
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
```
Output
```
[null, null, null, 1, null, -1, null, -1, 3, 4]
```

Explanation
```
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1);    // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2);    // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
lRUCache.get(1);    // return -1 (not found)
lRUCache.get(3);    // return 3
lRUCache.get(4);    // return 4
```

Constraints:

21.7K | 291 | 606 Online

Code

Java | Auto

```java
    ++count;

    if(count > capacity){
      // pop the tail
      DLinkedNode tail = this.popTail();
      this.cache.remove(tail.key);
      --count;
    }
  }else{
    // update the value.
    node.value = value;
    this.moveToHead(node);
  }
}

}
```

Saved | Ln 112, Col 2

Testcase | Test Result | Accepted

All Submissions

Accepted 23 / 23 testcases passed

Jiya submitted at Apr 18, 2025 22:29

Editorial | Solution

Runtime: 44 ms | Beats 79.06%

Analyze Complexity

Memory: 111.08 MB | Beats 98.27%