

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/>

```
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;
    }
}
```

Description

Editorial

Solutions

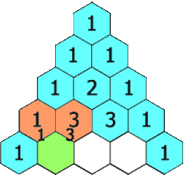
Submissions

118. Pascal's Triangle

Easy Topics Companies

Given an integer `numRows`, return the first `numRows` of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:



Example 1:
Input: `numRows = 5`
Output: `[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]`

Example 2:
Input: `numRows = 1`
Output: `[[1]]`

Constraints:

- `1 <= numRows <= 30`

Solved

Code

```
1 class Solution {
2     public List<List<Integer>> generate(int numRows) {
3         List<List<Integer>> result = new ArrayList<>();
4         if (numRows == 0) {
5             return result;
6         }
7
8         List<Integer> firstRow = new ArrayList<>();
9         firstRow.add(1);
10        result.add(firstRow);
11
12        for (int i = 1; i < numRows; i++) {
13            List<Integer> prevRow = result.get(i - 1);
14            List<Integer> currentRow = new ArrayList<>();
15            currentRow.add(1);
16
17            for (int j = 1; j < i; j++) {
```

Ln 1, Col 1

Testcase

Test Result

Accepted

All Submissions

Accepted 30 / 30 testcases passed

jiya submitted at Apr 18, 2025 22:04

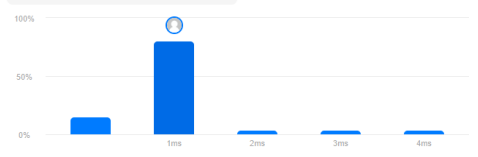
Runtime

Memory

1 ms | Beats: 85.01%

42.02 MB | Beats: 59.65%

Analyze Complexity



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;
    }
}
```

461. Hamming Distance

The **Hamming distance** between two integers is the number of positions at which the corresponding bits are different.

Given two integers x and y , return the **Hamming distance** between them.

Example 1:

Input: $x = 1, y = 4$
Output: 2
Explanation:
1 (0 0 0 1)
4 (0 1 0 0)
1 1
The above arrows point to positions where the corresponding bits are different.

Example 2:

Input: $x = 3, y = 1$
Output: 1

Constraints:

- $0 \leq x, y \leq 2^{31} - 1$

Note: This question is the same as 2220: Minimum Bit Flips to Convert Number.

Seen this question in a real interview before? 1/5

Accepted: 626K | Submissions: 823.9K | Acceptance Rate: 76.0%

3.9K | 34 | 24 Online

Code

```
1 class Solution {
2     public int hammingDistance(int x, int y) {
3         int difference = 0;
4         while(x>0&&y>0) {
5             int last1 = x & 1;
6             int last2 = y & 1;
7             if ((last1 ^ last2) == 1) {
8                 difference++;
9             }
10            x=x>>1;
11            y=y>>1;
12        }
13        while(x>0){
14            if((x&1)==1){
15                difference++;
16            }
17            x= x>>1;
18        }
19    }
20 }
```

Accepted 149 / 149 testcases passed
Jiya submitted at Apr 18, 2025 22:10

Runtime 0 ms | Beats 100.00%
Memory 40.40 MB | Beats 82.17%

100%
50%
0%

1ms 2ms 3ms 4ms

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;
    }
}
```

DescriptionEditorialSolutionsSubmissions

621. Task Scheduler

MediumTopicsCompaniesHint

You are given an array of CPU `tasks`, each labeled with a letter from A to Z and a number `n`. Each CPU interval can be idle or allow the completion of one task. Tasks can be completed in any order, but there's a constraint: there has to be a gap of **at least** `n` intervals between two tasks with the same label.

Return the **minimum** number of CPU intervals required to complete all tasks.

Example 1:

Input: `tasks = ["A","A","A","B","B","B"], n = 2`

Output: 8

Explanation: A possible sequence is: A -> B -> idle -> A -> B -> idle -> A -> B.

After completing task A, you must wait two intervals before doing A again. The same applies to task B. In the 3rd interval, neither A nor B can be done, so you idle. By the 4th interval, you can do A again as 2 intervals have passed.

Example 2:

Input: `tasks = ["A","C","A","B","D","B"], n = 1`

Output: 6

Explanation: A possible sequence is: A -> B -> C -> D -> A -> B.

With a cooling interval of 1, you can repeat a task after just one other task.

Example 3:

Input: `tasks = ["A","A","A","B","B","B"], n = 3`

Output: 10

Explanation: A possible sequence is: A -> B -> idle -> idle -> A -> B -> idle -> idle -> A -> B.

There are only two types of tasks, A and B, which need to be separated by 3 intervals. This leads to idling twice between repetitions of these

11K164168 Online

Solved

```
1 public class Solution {
2     public int leastInterval(char[] tasks, int n) {
3         int[] counter = new int[26];
4         int max = 0;
5         int maxCount = 0;
6         for(char task : tasks) {
7             counter[task - 'A']++;
8             if(max == counter[task - 'A']) {
9                 maxCount++;
10            }
11            else if(max < counter[task - 'A']) {
12                max = counter[task - 'A'];
13                maxCount = 1;
14            }
15        }
16
17        int partCount = max - 1;
18        int partLength = n - (maxCount - 1);
19        int emptySlots = partCount * partLength;
20        int availableTasks = tasks.length - max * maxCount;
21        int idles = Math.max(0, emptySlots - availableTasks);
22
23        return tasks.length + idles;
24    }
25 }
```

Ln 25, Col 2

TestcaseTest ResultAccepted

All Submissions

Accepted 72 / 72 testcases passed

jiya submitted at Apr 18, 2025 22:11


Runtime

4 msBeats 72.27%

Memory

46.26 MBBeats 18.99%

Analyze Complexity



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;
    }
}
```

The screenshot displays the LeetCode problem page for '191. Number of 1 Bits'. The left sidebar contains the problem description, examples, and constraints. The main area shows the code editor with a Java solution. The bottom right panel displays the test results, indicating that the solution is 'Accepted' with a runtime of 0 ms and memory usage of 40.52 MB.

191. Number of 1 Bits Solved ✓

Easy Topics Companies

Given a positive integer n , write a function that returns the number of **set bits** in its binary representation (also known as the **Hamming weight**).

Example 1:
Input: $n = 11$
Output: 3
Explanation:
The input binary string **1011** has a total of three set bits.

Example 2:
Input: $n = 128$
Output: 1
Explanation:
The input binary string **10000000** has a total of one set bit.

Example 3:
Input: $n = 2147483645$
Output: 30
Explanation:
The input binary string **1111111111111111111111111111101** has a total of thirty set bits.

Constraints:

6.8K 179

Code

```
1 class Solution {
2     public static int hammingWeight(int n) {
3         int ones = 0;
4         while(n!=0) {
5             ones = ones + (n & 1);
6             n = n>>>1;
7         }
8         return ones;
9     }
10 }
```

Accepted 598 / 598 testcases passed
jiya submitted at Apr 18, 2025 22:14

Runtime: 0 ms | Beats: 100.00%
Memory: 40.52 MB | Beats: 85.13%

Testcase Test Result Accepted

All Submissions

Editorial Solution

Runtime: 0 ms | Beats: 100.00%
Memory: 40.52 MB | Beats: 85.13%

Bar chart showing runtime performance across different test cases. The first bar (0 ms) is significantly higher than the others (1ms, 2ms, 3ms, 4ms).

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();
    }
}
```

20. Valid Parentheses

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

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:
Input: `s = "()"`
Output: `true`

Example 2:
Input: `s = "()[]{}"`
Output: `true`

Example 3:
Input: `s = "(]"`
Output: `false`

Example 4:
Input: `s = "([)]"`
Output: `false`

Constraints:
1 ≤ s.length ≤ 10⁴

Code:

```
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();
    }
}
```

Accepted 100 / 100 testcases passed
Jiya submitted at Apr 18, 2025 22:16

Runtime: 2 ms | Beats: 97.43%
Memory: 41.65 MB | Beats: 82.21%

Bar Chart:

Runtime (ms)	Percentage
1ms	~1%
2ms	~55%
3ms	~25%
4ms	~1%

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;
    }
}
```

DescriptionEditorialSolutionsSubmissions

29. Divide Two Integers

MediumTopicsCompanies

Given two integers `dividend` and `divisor`, divide two integers **without** using multiplication, division, and mod operator.

The integer division should truncate toward zero, which means losing its fractional part. For example, `8.345` would be truncated to `8`, and `-2.7335` would be truncated to `-2`.

Return the **quotient** after dividing `dividend` by `divisor`.

Note: Assume we are dealing with an environment that could only store integers within the **32-bit** signed integer range: $[-2^{31}, 2^{31} - 1]$. For this problem, if the quotient is **strictly greater than** $2^{31} - 1$, then return $2^{31} - 1$, and if the quotient is **strictly less than** -2^{31} , then return -2^{31} .

Example 1:

Input: dividend = 10, divisor = 3
Output: 3
Explanation: 10/3 = 3.33333.. which is truncated to 3.

Example 2:

Input: dividend = 7, divisor = -3
Output: -2
Explanation: 7/-3 = -2.33333.. which is truncated to -2.

Constraints:

- $-2^{31} \leq \text{dividend}, \text{divisor} \leq 2^{31} - 1$
- `divisor != 0`

Seen this question in a real interview before? 1/5

5.6K 309 128 Online

Code

JavaAuto

```
1 class Solution {
2     public int divide(int dividend, int divisor) {
3         if(dividend == divisor) return 1;
4         if (dividend == Integer.MIN_VALUE && divisor == -1) {
5             return Integer.MAX_VALUE;
6         }
7         if(divisor == 1) return dividend;
8         if(dividend == -1) return -dividend;
9         int sign = 1;
10        if(dividend>0 && divisor<0) sign = -1;
11        if(dividend<0 && divisor>0) sign = -1;
12
13        long n = Math.abs((long)dividend);
14        long d = Math.abs((long)divisor);
15        int ans = 0;
16        while(n>=d)
17        {
18            int p = 0;
19            while(n >= d<<p)
20                p++;
21
22            p--;
23            n -= d<<p;
24            ans += 1<<p;
25        }
26        if(ans>=Math.pow(2,31) && sign==1) return Integer.MAX_VALUE;
27        if(ans>=Math.pow(2,31) && sign==-1) return Integer.MIN_VALUE;
28
29        return ans*sign;
30    }
31 }
```

TestcaseTest ResultAccepted


All Submissions

Accepted 994 / 994 testcases passed
jiya submitted at Apr 18, 2025 22:22

Runtime
1 ms | Beats: 74.35%

Memory
40.70 MB | Beats: 95.47%

Analyze Complexity



1ms2ms3ms4ms

0%25%50%75%

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;
    }
}
```


DescriptionEditorialSolutionsSubmissions

42. Trapping Rain Water

HardTopicsCompanies

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:



Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

Input: height = [4,2,0,3,2,5]
Output: 9

Constraints:

- $n == \text{height.length}$
- $1 \leq n \leq 2 \times 10^4$
- $0 \leq \text{height}[i] \leq 10^5$

33.8K377512 Online

Code

JavaAuto

```
1 class Solution {
2     public int trap(int[] height) {
3         int i=0, left_max=height[0], sum=0;
4         int j=height.length-1, right_max=height[j];
5         while (i<j)
6         {
7             if(left_max <= right_max)
8             {
9                 sum+=(left_max-height[i]);
10                i++;
11                left_max=Math.max(left_max,height[i]);
12            }
13            else
14            {
15                sum+=(right_max-height[j]);
16                j--;
17                right_max=Math.max(right_max,height[j]);
18            }
19        }
20        return sum;
21    }
22 }
```

SavedLn 23, Col 2

TestcaseTest ResultAccepted

All Submissions


Accepted324 / 324 testcases passed
jiya submitted at Apr 18, 2025 22:23

EditorialSolution

Runtime0 msBeats 100.00%

Memory46.91 MBBeats 8.30%

Analyze Complexity



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;
    }
}
```

Description

Editorial

Solutions

Submissions

2071. Maximum Number of Tasks You Can Assign

Hard

You have n tasks and m workers. Each task has a strength requirement stored in a 0-indexed integer array `tasks`, with the i th task requiring `tasks[i]` strength to complete. The strength of each worker is stored in a 0-indexed integer array `workers`, with the j th worker having `workers[j]` strength. Each worker can only be assigned to a single task and must have a strength **greater than or equal to** the task's strength requirement (i.e. `workers[j] >= tasks[i]`).

Additionally, you have `pills` magical pills that will **increase a worker's strength** by `strength`. You can decide which workers receive the magical pills, however, you may only give each worker **at most one** magical pill.

Given the 0-indexed integer arrays `tasks` and `workers` and the integers `pills` and `strength`, return the **maximum** number of tasks that can be completed.

Example 1:

Input: `tasks = [3,2,1]`, `workers = [0,3,3]`, `pills = 1`, `strength = 1`

Output: 3

Explanation:

We can assign the magical pill and tasks as follows:

- Give the magical pill to worker 0.
- Assign worker 0 to task 2 (`0 + 1 >= 1`).
- Assign worker 1 to task 1 (`3 >= 2`).
- Assign worker 2 to task 0 (`3 >= 3`).

Example 2:

Input: `tasks = [5,4]`, `workers = [0,0,0]`, `pills = 1`, `strength = 5`

Output: 1

Explanation:

We can assign the magical pill and tasks as follows:

- Give the magical pill to worker 0.
- Assign worker 0 to task 0 (`0 + 5 >= 5`).

Example 3:

Input: `tasks = [10,15,30]`, `workers = [0,10,10,10]`, `pills = 3`, `strength = 10`

Output: 2

Code

Java

Auto

```
1 class Solution {
2     public int maxTaskAssign(int[] tasks, int[] workers, int pills, int strength) {
3         Arrays.sort(tasks);
4         TreeMap<Integer, Integer> map = new TreeMap<>();
5         for (int i : workers)
6             map.put(i, map.getOrDefault(i, 0) + 1);
7         int res = 0, left = 0, right = Math.min(tasks.length, workers.length) - 1;
8         while (left <= right) {
9             int mid = (left + right) / 2;
10            if (validate(tasks, (TreeMap<Integer, Integer>)map.clone(), pills, strength, mid))
11                res = left = mid + 1;
12            else
13                right = mid - 1;
14        }
15        return res;
16    }
17    boolean validate(int[] tasks, TreeMap<Integer, Integer> map, int pills, int strength, int pos) {
18        for (; pos >= 0; pos--) {
19            int maxStrength = map.lastKey(), t = tasks[pos];
20            if (pills > 0 && strength + maxStrength < t || pills == 0 && maxStrength < t)
21                return false;
22            if (maxStrength < t) {
23                t -= strength;
24                pills--;
25            }
26            int matchStrength = map.ceilingKey(t);
27            if (map.get(matchStrength) > 1)
28                map.put(matchStrength, map.get(matchStrength) - 1);
29            else
30                map.remove(matchStrength);
31        }
32        return true;
33    }
34 }
```

Testcase

Test Result

Case 1

Case 2

Case 3

tasks =

[3,2,1]

workers =

[0,3,3]

pills =

1

strength =

1

566

6

7 Online

</> Source

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));
```

DescriptionEditorialSolutionsSubmissions

297. Serialize and Deserialize Binary Tree

HardTopicsCompanies

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.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

Clarification: The input/output format is the same as [how LeetCode serializes a binary tree](#). You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

Example 1:

```
graph TD
    1((1)) --> 2((2))
    1 --> 3((3))
    3 --> 4((4))
    3 --> 5((5))
```

Input: root = [1,2,3,null,null,4,5]
Output: [1,2,3,null,null,4,5]

Example 2:

Input: root = []
Output: []

10.6K89130 Online

</>Code

JavaAuto

```
1 /**
2  * Definition for a binary tree node.
3  * public class TreeNode {
4  *     int val;
5  *     TreeNode left;
6  *     TreeNode right;
7  *     TreeNode(int x) { val = x; }
8  * }
9  */
10 public class Codec {
11
12     // Encodes a tree to a single string.
13     static TreeNode res;
14     public String serialize(TreeNode root) {
15         res=root;
16         return "";
17     }
18 }
```

SavedLn 17, Col 6

TestcaseTest ResultAccepted X

All Submissions

Accepted53 / 53 testcases passed
jiya submitted at Apr 18, 2025 22:20

EditorialSolution

Runtime0 ms | Beats: 100.00%

Memory44.92 MB | Beats: 98.67%

Analyze Complexity

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;
    }
}
```

146. LRU Cache

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"]
[[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

```
97 ++count;
98
99 if(count > capacity){
100     // pop the tail
101     DLinkedNode tail = this.popTail();
102     this.cache.remove(tail.key);
103     --count;
104 }
105 }else{
106     // update the value.
107     node.value = value;
108     this.moveToHead(node);
109 }
110 }
111
112 }
```

Saved

Ln 112, Col 2

Testcase Test Result Accepted X

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%

