

Name =Raja Babu

UID: 22BCS11971

Sec: 614-B

AP-Experiment-10

1. Pascal's Triangle

118. Pascal's Triangle

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`

```
class Solution {
public:
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> a(numRows);
        for(int&& i=0; i<numRows; i++){
            a[i].assign(i+1, 1); // exact allocation once
            for(int&& j=1; j<=i/2; j++){
                a[i][j]=a[i][j]+a[i-1][j-1]+a[i-1][j];
            }
        }
        return a;
    }
};
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

numRows = 5

2. Hamming Distance

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)
   ↑   ↑
The above arrows point to positions where the corresponding bits are different.
```

Example 2:

Input: `x = 3, y = 1`

Output: `1`

Constraints:

- `0 <= x, y <= 2^31 - 1`

```
class Solution {
public:
    int hammingDistance(int x, int y) {
        int ans = x ^ y;
        int count = 0;
        while(ans){
            count += (ans & 1);
            ans >>= 1;
        }
        return count;
    }
};
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

x = 1

3. Task Scheduler

162. Remove Stones to Minimize the Total Solved

Medium Topics Companies Hint

You are given a 0-indexed integer array `piles`, where `piles[i]` represents the number of stones in the i^{th} pile, and an integer `k`. You should apply the following operation **exactly** `k` times:

- Choose any `piles[i]` and **remove** $\text{floor}(\text{piles}[i] / 2)$ stones from it.

Notice that you can apply the operation on the **same** pile more than once.

Return the **minimum** possible total number of stones remaining after applying the `k` operations.

$\text{floor}(x)$ is the **greatest** integer that is **smaller** than or **equal** to `x` (i.e., rounds `x` down).

Example 1:

Input: `piles = [5,4,9]`, `k = 2`
Output: 12
Explanation: Steps of a possible scenario are:
- Apply the operation on pile 2. The resulting piles are `[5,4,5]`.
- Apply the operation on pile 0. The resulting piles are `[3,4,5]`.
The total number of stones in `[3,4,5]` is 12.

Example 2:

Input: `piles = [4,3,6,7]`, `k = 3`
Output: 12

```
int minStoneSum(vector<int>& piles, int k) {
    int n=piles.size();
    priority_queue<int,vector<int>>>pq(piles.begin(),piles.end());
    int ans=accumulate(piles.begin(),piles.end(),0);
    int i=0;
    while(k>0 && !pq.empty())
    {
        int temp=pq.top();
        pq.pop();
        ans-=temp/2;
        pq.push(temp-temp/2);
        k--;
    }
    return ans;
}
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

piles =
[5,4,9]

4. Number of 1 Bits

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:

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int res = 0;
        for (int i = 0; i < 32; i++) {
            if ((n >> i) & 1) {
                res += 1;
            }
        }
        return res;
    }
};
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

n =
11

Problem List < > 🔍

Description Accepted Editorial Solutions Submissions

42. Trapping Rain Water

Hard Topics Companies

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]
```

Solved ✓

C++ Auto

```

7   int rightMax = height[right];
8   int water = 0;
9
10  while (left < right) {
11      if (leftMax < rightMax) {
12          left++;
13          leftMax = max(leftMax, height[left]);
14          water += leftMax - height[left];
15      } else {
16          right--;
17          rightMax = max(rightMax, height[right]);
18          water += rightMax - height[right];
19      }
20  }
21
22  return water;
23
24  };
    
```

Saved

Ln 24, Col 3

Testcase Test Result

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

```
height =
[0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]
```

33.8K 377 482 Online

7. Max Number of Tasks You Can Assign

CU-Assignments/ap-lab-exper... ap-lab-experiment-9-graphs-i... Microsoft Word - AP Experime... Maximum Number of Tasks You...
leetcode.com/problems/maximum-number-of-tasks-you-can-assign/description/?envType=problem-list-v2&envId=greedy

Greedy < > >>

Run Submit

Description Editorial Solutions Accepted Submissions

2071. Maximum Number of Tasks You Can Assign

Hard Topics Companies Hint

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`).

566 6 4 Online

Code

```
C++  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
};  
  
if (pillCountRemaining == 0) break;  
it = workersSet.lower_bound(tasks[i] - strength);  
if (it == workersSet.end()) break;  
pillCountRemaining--;  
workersSet.erase(it);  
  
if (workersSet.empty()) {  
    answer = mid;  
    left = mid + 1;  
} else {  
    right = mid - 1;  
}  
  
return answer;
```

Saved Ln 45, Col 3

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

tasks =
[3,2,1]

8. Serialize and Deserialize Binary Tree

CU-Assignments/ap-lab-exper... ap-lab-experiment-9-graphs-i... Microsoft Word - AP Experime... Serialize and Deserialize Binary...
leetcode.com/problems/serialize-and-deserialize-binary-tree/description/

Problem List < > >>

Run Submit

Description Editorial Solutions Accepted Submissions

297. Serialize and Deserialize Binary Tree

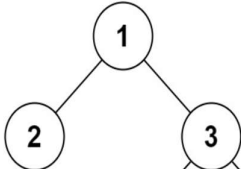
Hard Topics Companies

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:



```
C++  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
};  
  
serialize(root->left, out);  
serialize(root->right, out);  
} else {  
    out << "# ";  
}  
  
TreeNode* deserialize(Istringstream& in) {  
    string val;  
    in >> val;  
    if (val == "#")  
        return nullptr;  
    TreeNode* root = new TreeNode(stoi(val));  
    root->left = deserialize(in);  
    root->right = deserialize(in);  
    return root;  
}
```

Saved Ln 37, Col 3

Testcase Test Result

Accepted Runtime: 3 ms

Case 1 Case 2

Input

root =
[1,2,3,null,null,4,5]

9. LRU Cache

The screenshot shows the LeetCode interface for the '146. LRU Cache' problem. The problem description on the left explains the requirements for an LRU cache, including the `LRUCache` class methods and constraints. A C++ solution is shown in the center, implementing the cache with a doubly linked list and a hash map. The test result on the right shows the solution is 'Accepted'.

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"]
[[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);`

```
62 if (cache.size() > cap) {
63     Node* lru = oldest->next;
64     remove(lru);
65     cache.erase(lru->key);
66     delete lru;
67 }
68 }
69
70 // Destructor to release memory used by the nodes
71 ~LRUCache() {
72     Node* curr = oldest;
73     while (curr != nullptr) {
74         Node* next = curr->next;
75         delete curr;
76         curr = next;
77     }
78 }
79 };
```

Accepted Runtime: 0 ms

Case 1

Input

["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get"]