

201. 数字范围按位与

201. 数字范围按位与

难度 中等

104

收藏

分享

切换为英文

关注

反馈

给定范围 $[m, n]$ ，其中 $0 \leq m \leq n \leq 2147483647$ ，返回此范围内所有数字的按位与（包含 m, n 两端点）。

示例 1:

输入: $[5, 7]$

输出: 4

示例 2:

输入: $[0, 1]$

输出: 0

```
class Solution {
public:
    int rangeBitwiseAnd(int m, int n) {
        int ans = 0;
        for (int i = 30; i >= 0; i--) {
            if ((m >> i & 1) ^ (n >> i & 1))
                break;
            ans |= (m >> i & 1) << i;
        }
        return ans;
    }
};
```

202. 快乐数

202. 快乐数

难度 简单

👍 282

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

💡 反馈

编写一个算法来判断一个数 `n` 是不是快乐数。

「快乐数」定义为：对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和，然后重复这个过程直到这个数变为 1，也可能是 **无限循环** 但始终变不到 1。如果 **可以变为** 1，那么这个数就是快乐数。

如果 `n` 是快乐数就返回 `True`；不是，则返回 `False`。

示例：

```
输入：19
输出：true
解释：
 $1^2 + 9^2 = 82$ 
 $8^2 + 2^2 = 68$ 
 $6^2 + 8^2 = 100$ 
 $1^2 + 0^2 + 0^2 = 1$ 
```

```
class Solution {
public:
    bool isHappy(int n) {
        unordered_set<int> got;
        int m = n; // m是每次用来被拆解的数
        while(true){
            int sum = 0;
            while(m != 0){
                sum += (m % 10) * (m % 10);
                m /= 10;
            }
            if(sum == 1)
                return true;
            else if(got.find(sum)!=got.end())
                return false;
            else
                got.insert(sum);
            m = sum;
        }
    }
};
```

203. 移除链表元素

203. 移除链表元素

难度 简单

👍 377

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

删除链表中等于给定值 *val* 的所有节点。

示例:

输入: 1->2->6->3->4->5->6, *val* = 6
输出: 1->2->3->4->5

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val) {
        ListNode *dummy = new ListNode(-1);
        dummy->next = head;
        for (ListNode *p = dummy; p;)
        {
            if (p->next && p->next->val == val)
                p->next = p->next->next;
            else
                p = p->next;
        }
        return dummy->next;
    }
};
```

204. 计数质数

204. 计数质数

难度 简单

👍 334

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

统计所有小于非负整数 *n* 的质数的数量。

示例:

输入: 10
输出: 4
解释: 小于 10 的质数一共有 4 个, 它们是 2, 3, 5, 7 。

```
class Solution {
public:
    int countPrimes(int n) {
        vector<int> primes;
        vector<bool> st(n + 1);
```

```

        for (int i = 2; i < n; i ++ )
        {
            if (!st[i]) primes.push_back(i);
            for (int j = 0; i * primes[j] < n; j ++ )
            {
                st[i * primes[j]] = true;
                if (i % primes[j] == 0) break;
            }
        }
        return primes.size();
    }
};

```

205. 同构字符串

205. 同构字符串

难度 简单

👍 187

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定两个字符串 s 和 t ，判断它们是否是同构的。

如果 s 中的字符可以被替换得到 t ，那么这两个字符串是同构的。

所有出现的字符都必须用另一个字符替换，同时保留字符的顺序。两个字符不能映射到同一个字符上，但字符可以映射自己本身。

示例 1:

输入: $s = \text{"egg"}, t = \text{"add"}$
输出: true

示例 2:

输入: $s = \text{"foo"}, t = \text{"bar"}$
输出: false

示例 3:

输入: $s = \text{"paper"}, t = \text{"title"}$
输出: true

说明:

你可以假设 s 和 t 具有相同的长度。

```

class Solution {
public:
    bool isIsomorphic(string s, string t) {
        unordered_map<char, char> st, ts;
        if (s.size() != t.size()) return false;
        for (int i = 0; i < s.size(); i ++ )
        {
            if (st.count(s[i]))
            {
                if (st[s[i]] != t[i]) return false;
            }
            else st[s[i]] = t[i];
        }
    }
};

```

```

        if (ts.count(t[i]))
        {
            if (ts[t[i]] != s[i]) return false;
        }
        else ts[t[i]] = s[i];
    }
    return true;
}
};

```

206. 反转链表

206. 反转链表

难度 **简单**  923  收藏  分享  切换为英文  关注  反馈

反转一个单链表。

示例:

输入: 1->2->3->4->5->NULL
输出: 5->4->3->2->1->NULL

进阶:

你可以迭代或递归地反转链表。你能否用两种方法解决这道题?

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if(!head) return NULL;
        if(!head->next) return head;
        ListNode *p = reverseList(head->next);
        head->next->next = head;
        head->next = NULL;
        return p;
    }
};

```

207. 课程表

207. 课程表

难度 中等

305

收藏

分享

切换为英文

关注

反馈

你这个学期必须选修 `numCourse` 门课程，记为 `0` 到 `numCourse-1`。

在选修某些课程之前需要一些先修课程。例如，想要学习课程 `0`，你需要先完成课程 `1`，我们用一个匹配来表示他们：`[0,1]`

给定课程总量以及它们的先决条件，请你判断是否可能完成所有课程的学习？

示例 1:

输入: 2, [[1,0]]

输出: true

解释: 总共有 2 门课程。学习课程 1 之前，你需要完成课程 0。所以这是可能的。

示例 2:

输入: 2, [[1,0],[0,1]]

输出: false

解释: 总共有 2 门课程。学习课程 1 之前，你需要先完成课程 0；并且学习课程 0 之前，你还应先完成课程 1。这是不可能的。

提示:

1. 输入的先决条件是由 **边缘列表** 表示的图形，而不是邻接矩阵。详情请参见图的表示法。
2. 你可以假定输入的先决条件中没有重复的边。
3. $1 \leq \text{numCourses} \leq 10^5$

```
class Solution {
public:
    bool canFinish(int numCourses, vector<pair<int, int>>& prerequisites) {
        vector<vector<int>> graph(numCourses);
        vector<int> in_degree(numCourses, 0);
        for (int i = 0; i < prerequisites.size(); i++) {
            in_degree[prerequisites[i].first]++;
            graph[prerequisites[i].second].push_back(prerequisites[i].first);
        }

        queue<int> q;
        vector<bool> vis(numCourses, false);

        for (int i = 0; i < numCourses; i++)
            if (in_degree[i] == 0)
                q.push(i);
        while (!q.empty()) {
            int sta = q.front();
            q.pop();
            vis[sta] = true;
            for (int i = 0; i < graph[sta].size(); i++) {
                in_degree[graph[sta][i]]--;
                if (in_degree[graph[sta][i]] == 0)
                    q.push(graph[sta][i]);
            }
        }

        for (int i = 0; i < numCourses; i++)
            if (!vis[i])
                return false;
        return true;
    }
};
```

```

    }
}

for (int i = 0; i < numCourses; i++)
    if (vis[i] == false)
        return false;
return true;
}
};

```

208. 实现 Trie (前缀树)

208. 实现 Trie (前缀树)

难度 **中等** 278 收藏 分享 切换为英文 关注 反馈

实现一个 Trie (前缀树)，包含 `insert`、`search` 和 `startsWith` 这三个操作。

示例:

```

Trie trie = new Trie();

trie.insert("apple");
trie.search("apple");   // 返回 true
trie.search("app");     // 返回 false
trie.startsWith("app"); // 返回 true
trie.insert("app");
trie.search("app");     // 返回 true

```

说明:

- 你可以假设所有的输入都是由小写字母 `a-z` 构成的。
- 保证所有输入均为非空字符串。

```

class Trie {
public:
    struct Node
    {
        bool is_end;
        Node *son[26];
        Node()
        {
            is_end = false;
            for(int i = 0; i < 26; i++) son[i] = NULL;
        }
    }*root;
    /** Initialize your data structure here. */
    Trie() {
        root = new Node();
    }

    /** Inserts a word into the trie. */
    void insert(string word) {
        auto p = root;
        for(auto c : word)
        {

```

```

        int u = c - 'a';
        if(p->son[u] == NULL) p->son[u] = new Node();
        p = p->son[u];
    }
    p->is_end = true;
}

/** Returns if the word is in the trie. */
bool search(string word) {
    auto p = root;
    for(auto c : word)
    {
        int u = c - 'a';
        if(p->son[u] == NULL) return false;
        p = p->son[u];
    }
    return p->is_end;
}

/** Returns if there is any word in the trie that starts with the given
prefix. */
bool startswith(string prefix) {
    auto p = root;
    for(auto c : prefix)
    {
        int u = c - 'a';
        if(p->son[u] == NULL) return false;
        p = p->son[u];
    }
    return true;
}

};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startswith(prefix);
 */

```

209. 长度最小的子数组

209. 长度最小的子数组

难度 中等

👁 248

💖 收藏

🗨 分享

🌐 切换为英文

🔔 关注

💡 反馈

给定一个含有 n 个正整数的数组和一个正整数 s ，找出该数组中满足其和 $\geq s$ 的长度最小的连续子数组。如果不存在符合条件的连续子数组，返回 0。

示例:

输入: $s = 7$, $nums = [2,3,1,2,4,3]$

输出: 2

解释: 子数组 $[4,3]$ 是该条件下的长度最小的连续子数组。

进阶:

如果你已经完成了 $O(n)$ 时间复杂度的解法, 请尝试 $O(n \log n)$ 时间复杂度的解法。

```
class Solution {
public:
    int minSubArrayLen(int s, vector<int>& nums) {
        int n = nums.size();
        int start = 0, end = 0, tot = 0, ans = n + 1;
        while (end < n) {
            tot += nums[end];
            while (tot >= s) {
                ans = min(ans, end - start + 1);
                tot -= nums[start];
                start++;
            }
            end++;
        }
        if (ans == n + 1)
            ans = 0;
        return ans;
    }
};
```

210. 课程表 II

210. 课程表 II

难度 中等

124

收藏

分享

切换为英文

关注

反馈

现在你总共有 n 门课需要选，记为 0 到 $n-1$ 。

在选修某些课程之前需要一些先修课程。例如，想要学习课程 0 ，你需要先完成课程 1 ，我们用一个匹配来表示他们: $[0, 1]$

给定课程总量以及它们的先决条件，返回你为了学完所有课程所安排的学习顺序。

可能会有多个正确的顺序，你只要返回一种就可以了。如果不可能完成所有课程，返回一个空数组。

示例 1:

输入: 2, $[[1,0]]$

输出: $[0,1]$

解释: 总共有 2 门课程。要学习课程 1，你需要先完成课程 0。因此，正确的课程顺序为 $[0,1]$ 。

示例 2:

输入: 4, $[[1,0],[2,0],[3,1],[3,2]]$

输出: $[0,1,2,3]$ or $[0,2,1,3]$

解释: 总共有 4 门课程。要学习课程 3，你应该先完成课程 1 和课程 2。并且课程 1 和课程 2 都应该排在课程 0 之后。

因此，一个正确的课程顺序是 $[0,1,2,3]$ 。另一个正确的排序是 $[0,2,1,3]$ 。

说明:

1. 输入的先决条件是由**边缘列表**表示的图形，而不是邻接矩阵。详情请参见图的表示法。
2. 你可以假定输入的先决条件中没有重复的边。

提示:

1. 这个问题相当于查找一个循环是否存在于有向图中。如果存在循环，则不存在拓扑排序，因此不可能选取所有课程进行学习。
2. 通过 DFS 进行拓扑排序 - 一个关于Coursera的精彩视频教程 (21分钟)，介绍拓扑排序的基本概念。
3. 拓扑排序也可以通过 BFS 完成。

```
class Solution {
public:
    vector<int> findOrder(int numCourses, vector<pair<int, int>>& prerequisites)
    {
        vector<vector<int>> graph(numCourses);
        vector<int> in_degree(numCourses, 0);
        for (int i = 0; i < prerequisites.size(); i++) {
            in_degree[prerequisites[i].first]++;
            graph[prerequisites[i].second].push_back(prerequisites[i].first);
        }

        queue<int> q;
        vector<bool> vis(numCourses, false);
        vector<int> ans;

        for (int i = 0; i < numCourses; i++)
            if (in_degree[i] == 0)
```

```

        q.push(i);
    while (!q.empty()) {
        int sta = q.front();
        q.pop();
        ans.push_back(sta);
        vis[sta] = true;
        for (int i = 0; i < graph[sta].size(); i++) {
            in_degree[graph[sta][i]]--;
            if (in_degree[graph[sta][i]] == 0)
                q.push(graph[sta][i]);
        }
    }

    for (int i = 0; i < numCourses; i++)
        if (vis[i] == false)
            return vector<int>{};
    return ans;
}
};

```

211. 添加与搜索单词 - 数据结构设计

211. 添加与搜索单词 - 数据结构设计

难度 中等

115

收藏

分享

切换为英文

关注

反馈

设计一个支持以下两种操作的数据结构：

```

void addWord(word)
bool search(word)

```

search(word) 可以搜索文字或正则表达式字符串，字符串只包含字母 `.` 或 `a-z`。`.` 可以表示任何一个字母。

示例:

```

addWord("bad")
addWord("dad")
addWord("mad")
search("pad") -> false
search("bad") -> true
search(".ad") -> true
search("b..") -> true

```

说明:

你可以假设所有单词都是由小写字母 `a-z` 组成的。

```

class wordDictionary {
public:
    struct Node
    {
        bool is_end;
        Node *next[26];
        Node()

```

```

    {
        is_end = false;
        for (int i = 0; i < 26; i ++ )
            next[i] = 0;
    }
};

Node *root;

/** Initialize your data structure here. */
wordDictionary() {
    root = new Node();
}

/** Adds a word into the data structure. */
void addWord(string word) {
    Node *p = root;
    for (char c : word)
    {
        int son = c - 'a';
        if (!p->next[son]) p->next[son] = new Node();
        p = p->next[son];
    }
    p->is_end = true;
}

/** Returns if the word is in the data structure. A word could contain the
dot character '.' to represent any one letter. */
bool search(string word) {
    return dfs(word, 0, root);
}

bool dfs(string &word, int k, Node *u)
{
    if (k == word.size()) return u->is_end;
    if (word[k] != '.')
    {
        if (u->next[word[k] - 'a']) return dfs(word, k + 1, u->next[word[k]
- 'a']);
    }
    else
    {
        for (int i = 0; i < 26; i ++ )
            if (u->next[i])
            {
                if (dfs(word, k + 1, u->next[i])) return true;
            }
    }
    return false;
}
};

/**
 * Your wordDictionary object will be instantiated and called as such:
 * wordDictionary obj = new wordDictionary();
 * obj.addWord(word);
 * bool param_2 = obj.search(word);
 */

```

212. 单词搜索 II

212. 单词搜索 II

难度 **困难**

👍 143

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个二维网格 **board** 和一个字典中的单词列表 **words**，找出所有同时在二维网格和字典中出现的单词。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母在一个单词中不允许被重复使用。

示例:

输入:

```
words = ["oath","pea","eat","rain"] and board =  
[  
  ['o','a','a','n'],  
  ['e','t','a','e'],  
  ['i','h','k','r'],  
  ['i','f','l','v']  
]
```

输出: ["eat","oath"]

说明:

你可以假设所有输入都由小写字母 `a-z` 组成。

提示:

- 你需要优化回溯算法以通过更大数据量的测试。你能否早点停止回溯？
- 如果当前单词不存在于所有单词的前缀中，则可以立即停止回溯。什么样的数据结构可以有效地执行这样的操作？散列表是否可行？为什么？前缀树如何？如果你想学习如何实现一个基本的前缀树，请先查看这个问题： [实现Trie（前缀树）](#)。

```
class Solution {  
public:  
  
    struct Node  
    {  
        int id;  
        Node *next[26];  
        Node()  
        {  
            id = -1;  
            for (int i = 0; i < 26; i ++ )  
                next[i] = 0;  
        }  
    };  
  
    Node *root;  
  
    void insert(string word, int id) {  
        Node *p = root;  
        for (char c : word)  
        {
```

```

        int son = c - 'a';
        if (!p->next[son]) p->next[son] = new Node();
        p = p->next[son];
    }
    p->id = id;
}

unordered_set<string> hash;
vector<string> ans;
vector<vector<bool>> st;
vector<string> _words;
int n, m;

vector<string> findWords(vector<vector<char>>& board, vector<string>& words)
{
    if (board.empty()) return ans;
    _words = words;
    root = new Node();
    for (int i = 0; i < words.size(); i++) insert(words[i], i);
    n = board.size(), m = board[0].size();
    st = vector<vector<bool>>(n, vector<bool>(m, false));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            dfs(board, i, j, root->next[board[i][j]-'a']);
    return ans;
}

void dfs(vector<vector<char>>& board, int x, int y, Node *u)
{
    if (!u) return;
    st[x][y] = true;
    if (u->id != -1)
    {
        string match = _words[u->id];
        if (!hash.count(match))
        {
            hash.insert(match);
            ans.push_back(match);
        }
    }
    int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
    for (int i = 0; i < 4; i++)
    {
        int a = x + dx[i], b = y + dy[i];
        if (a >= 0 && a < n && b >= 0 && b < m && !st[a][b])
        {
            char c = board[a][b];
            dfs(board, a, b, u->next[c-'a']);
        }
    }
    st[x][y] = false;
}
};

```

213. 打家劫舍 II

213. 打家劫舍 II

难度 中等 246 收藏 分享 切换为英文 关注 反馈

你是一个专业的小偷，计划偷窃沿街的房屋，每间房内都藏有一定的现金。这个地方所有的房屋都围成一圈，这意味着第一个房屋和最后一个房屋是紧挨着的。同时，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组，计算你**在不触动警报装置的情况下**，能够偷窃到的最高金额。

示例 1:

输入: [2,3,2]

输出: 3

解释: 你不能先偷窃 1 号房屋 (金额 = 2)，然后偷窃 3 号房屋 (金额 = 2)，因为他们是相邻的。

示例 2:

输入: [1,2,3,1]

输出: 4

解释: 你可以先偷窃 1 号房屋 (金额 = 1)，然后偷窃 3 号房屋 (金额 = 3)。
偷窃到的最高金额 = 1 + 3 = 4。




```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        if (n == 0) return 0;
        if (n == 1) return nums[0];
        if (n == 2) return max(nums[0], nums[1]);

        int ans = 0, f, g;
        // choose the first one
        f = nums[2]; g = 0;
        for (int i = 3; i < n; i++) {
            int last_f = f, last_g = g;
            f = last_g + nums[i];
            g = max(last_f, last_g);
        }
        ans = g + nums[0];

        // not choose the first one
        f = nums[1]; g = 0;
        for (int i = 2; i < n; i++) {
            int last_f = f, last_g = g;
            f = last_g + nums[i];
            g = max(last_f, last_g);
        }
        ans = max(ans, max(f, g));
        return ans;
    }
};
```

214. 最短回文串

214. 最短回文串

难度 **困难**  137  收藏  分享  切换为英文  关注  反馈

给定一个字符串 s ，你可以通过在字符串前面添加字符将其转换为回文串。找到并返回可以用这种方式转换的最短回文串。

示例 1:

输入: "aacecaaa"
输出: "aaacecaaa"

示例 2:

输入: "abcd"
输出: "dcbabcd"

```
class Solution {
public:
    string shortestPalindrome(string s) {
        int n = s.length();
        if (n == 0)
            return "";
        string t(s.rbegin(), s.rend());
        vector<int> nxt(n);
        nxt[0] = -1;
        int j = -1;
        for (int i = 1; i < n; i++) {
            while (j > -1 && s[i] != s[j + 1]) j = nxt[j];
            if (s[i] == s[j + 1])
                j++;
            nxt[i] = j;
        }

        j = -1;
        for (int i = 0; i < n; i++) {
            while (j > -1 && t[i] != s[j + 1]) j = nxt[j];
            if (t[i] == s[j + 1])
                j++;
        }
        return t + s.substr(j + 1, n - j - 1);
    }
};
```

215. 数组中的第K个最大元素

215. 数组中的第K个最大元素

难度 中等

454

收藏

分享

切换为英文

关注

反馈

在未排序的数组中找到第 k 个最大的元素。请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。

示例 1:

输入: [3,2,1,5,6,4] 和 $k = 2$
输出: 5

示例 2:

输入: [3,2,3,1,2,4,5,5,6] 和 $k = 4$
输出: 4

说明:

你可以假设 k 总是有效的，且 $1 \leq k \leq$ 数组的长度。

```
class Solution {
public:
    int solve(int l, int r, vector<int>& nums, int k) {
        if (l == r)
            return nums[l];
        int pivot = nums[l], i = l, j = r;
        while (i < j) {
            while (i < j && nums[j] < pivot) j--;
            if (i < j)
                nums[i++] = nums[j];

            while (i < j && nums[i] >= pivot) i++;
            if (i < j)
                nums[j--] = nums[i];
        }
        if (i + 1 == k)
            return pivot;
        else if (i + 1 > k)
            return solve(l, i - 1, nums, k);
        else
            return solve(i + 1, r, nums, k);
    }
    int findKthLargest(vector<int>& nums, int k) {
        int n = nums.size();
        return solve(0, n - 1, nums, k);
    }
};
```

216. 组合总和 III

216. 组合总和 III

难度 中等

👍 105

📖 收藏

🔗 分享

🌐 切换为英文

👤 关注

💡 反馈

找出所有相加之和为 n 的 k 个数的组合。组合中只允许含有 1 - 9 的正整数，并且每种组合中不存在重复的数字。

说明：

- 所有数字都是正整数。
- 解集不能包含重复的组合。

示例 1:

输入： $k = 3, n = 7$

输出： $[[1,2,4]]$

示例 2:

输入： $k = 3, n = 9$

输出： $[[1,2,6], [1,3,5], [2,3,4]]$

```
class Solution {
public:
    vector<vector<int>> ans;
    vector<int> path;
    vector<vector<int>> combinationSum3(int k, int n) {
        dfs(k,1,n);
        return ans;
    }

    void dfs(int k,int start,int n)
    {
        if(!k)
        {
            if(!n) ans.push_back(path);
            return;
        }

        for(int i = start;i <= 9;i ++){
            path.push_back(i);
            dfs(k - 1,i + 1,n - i);
            path.pop_back();
        }
    }
};
```

217. 存在重复元素

217. 存在重复元素

难度 简单

👍 236

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个整数数组，判断是否存在重复元素。

如果任意一值在数组中出现至少两次，函数返回 `true` 。如果数组中每个元素都不相同，则返回 `false` 。

示例 1:

输入: [1,2,3,1]
输出: true

示例 2:

输入: [1,2,3,4]
输出: false

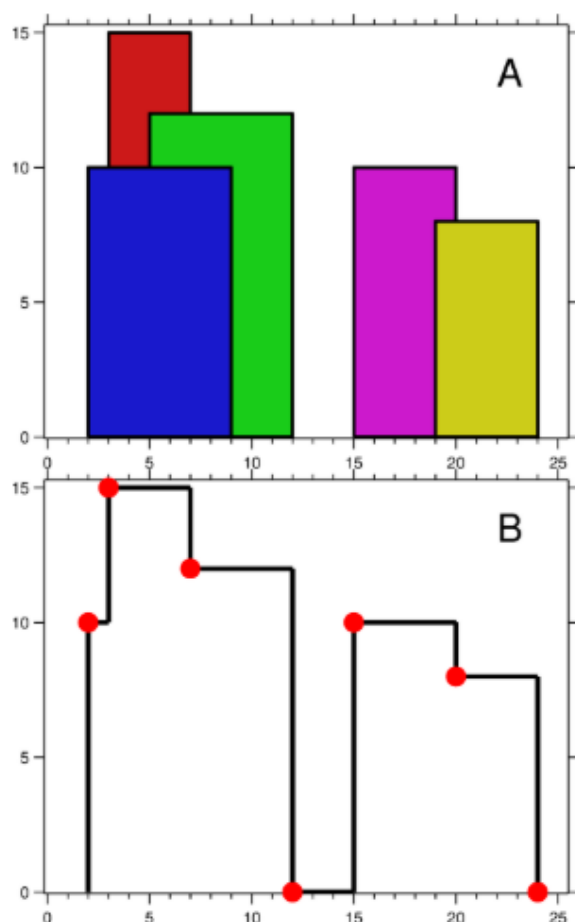
示例 3:

输入: [1,1,1,3,3,4,3,2,4,2]
输出: true

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        unordered_set<int> hash;
        for (int i = 0; i < nums.size(); i++) {
            if (hash.find(nums[i]) != hash.end())
                return true;
            hash.insert(nums[i]);
        }
        return false;
    }
};
```

218. 天际线问题

城市的天际线是从远处观看该城市中所有建筑物形成的轮廓的外部轮廓。现在，假设您获得了城市风光照片（图A）上显示的所有建筑物的位置和高度，请编写一个程序以输出由这些建筑物形成的天际线（图B）。



每个建筑物的几何信息用三元组 $[Li, Ri, Hi]$ 表示，其中 Li 和 Ri 分别是第 i 座建筑物左右边缘的 x 坐标， Hi 是其高度。可以保证 $0 \leq Li, Ri \leq INT_MAX$, $0 < Hi \leq INT_MAX$ 和 $Ri - Li > 0$ 。您可以假设所有建筑物都是在绝对平坦且高度为 0 的表面的完美矩形。

例如，图A中所有建筑物的尺寸记录为： $[[2, 9, 10], [3, 7, 15], [5, 12, 12], [15, 20, 10], [19, 24, 8]]$ 。

输出是以 $[[x1, y1], [x2, y2], [x3, y3], \dots]$ 格式的“关键点”（图B中的红点）的列表，它们唯一地定义了天际线。关键是水平线段的左端点。请注意，最右侧建筑物的最后一个关键点仅用于标记天际线的终点，并始终为零高度。此外，任何两个相邻建筑物之间的地面都应被视为天际线轮廓的一部分。

219. 存在重复元素 II

219. 存在重复元素 II

难度 简单

161

收藏

分享

切换为英文

关注

反馈

给定一个整数数组和一个整数 k ，判断数组中是否存在两个不同的索引 i 和 j ，使得 $\text{nums}[i] = \text{nums}[j]$ ，并且 i 和 j 的差的绝对值至多为 k 。

示例 1:

输入: $\text{nums} = [1,2,3,1]$, $k = 3$
输出: true

示例 2:

输入: $\text{nums} = [1,0,1,1]$, $k = 1$
输出: true

示例 3:

输入: $\text{nums} = [1,2,3,1,2,3]$, $k = 2$
输出: false

```
class Solution {
public:
    bool containsNearbyDuplicate(vector<int>& nums, int k) {
        unordered_map<int, int> hash;
        for (int i = 0; i < nums.size(); i++) {
            if (hash.find(nums[i]) != hash.end()) {
                if (i - hash[nums[i]] <= k)
                    return true;
            }
            hash[nums[i]] = i;
        }
        return false;
    }
};
```

220. 存在重复元素 III

220. 存在重复元素 III

难度 中等

165

收藏

分享

切换为英文

关注

反馈

给定一个整数数组，判断数组中是否有两个不同的索引 i 和 j ，使得 $\text{nums}[i]$ 和 $\text{nums}[j]$ 的差的绝对值最大为 t ，并且 i 和 j 之间的差的绝对值最大为 k 。

示例 1:

输入: $\text{nums} = [1,2,3,1]$, $k = 3$, $t = 0$
输出: true

示例 2:

输入: $\text{nums} = [1,0,1,1]$, $k = 1$, $t = 2$
输出: true

示例 3:

输入: $\text{nums} = [1,5,9,1,5,9]$, $k = 2$, $t = 3$
输出: false

```
#define LL long long
class Solution {
public:
    bool containsNearbyAlmostDuplicate(vector<int>& nums, int k, int t) {
        multiset<LL> hash;
        multiset<LL>::iterator it;
        for (int i = 0; i < nums.size(); i++) {
            it = hash.lower_bound((LL)(nums[i]) - t);
            if (it != hash.end() && *it <= (LL)(nums[i]) + t)
                return true;
            hash.insert(nums[i]);
            if (i >= k)
                hash.erase(hash.find(nums[i - k]));
        }
        return false;
    }
};
```