181. 超过经理收入的员工

181. 超过经理收入的员工

难度 简单 254 收藏 分享 切换为英文 关注 反馈

SQL架构 ›

`Employee` 表包含所有员工，他们的经理也属于员工。每个员工都有一个 Id，此外还有一列对应员工的经理的 Id。

```
+----+-------+--------+-----------+
| Id | Name  | Salary | ManagerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | NULL      |
| 4  | Max   | 90000  | NULL      |
+----+-------+--------+-----------+
```

给定 `Employee` 表，编写一个 SQL 查询，该查询可以获取收入超过他们经理的员工的姓名。在上面的表格中，Joe 是唯一一个收入超过他的经理的员工。

```
+----------+
| Employee |
+----------+
| Joe      |
+----------+
```

```sql
SELECT
    *
FROM
    Employee AS a,
    Employee AS b
WHERE
    a.ManagerId = b.Id
        AND a.Salary > b.Salary
;
```

182. 查找重复的电子邮箱

182. 查找重复的电子邮箱

难度 简单 189 收藏 分享 切换为英文 关注 反馈

SQL架构 ›

编写一个 SQL 查询，查找 `Person` 表中所有重复的电子邮箱。

示例：

```
+----+---------+
| Id | Email   |
+----+---------+
| 1  | a@b.com |
| 2  | c@d.com |
| 3  | a@b.com |
+----+---------+
```

根据以上输入，你的查询应返回以下结果：

```
+---------+
| Email   |
+---------+
| a@b.com |
+---------+
```

说明： 所有电子邮箱都是小写字母。

```sql
select Email from
(
  select Email, count(Email) as num
  from Person
  group by Email
) as statistic
where num > 1
;
```

183. 从不订购的客户

183. 从不订购的客户

难度 简单 150 收藏 分享 切换为英文 关注 反馈

SQL架构 ›

某网站包含两个表，`Customers` 表和 `Orders` 表。编写一个 SQL 查询，找出所有从不订购任何东西的客户。

`Customers` 表：

```
+----+-------+
| Id | Name  |
+----+-------+
| 1  | Joe   |
| 2  | Henry |
| 3  | Sam   |
| 4  | Max   |
+----+-------+
```

`Orders` 表：

```
+----+------------+
| Id | CustomerId |
+----+------------+
| 1  | 3          |
| 2  | 1          |
+----+------------+
```

例如给定上述表格，你的查询应返回：

```
+-----------+
| Customers |
+-----------+
| Henry     |
| Max       |
+-----------+
```

```sql
select customers.name as 'Customers'
from customers
where customers.id not in
(
    select customerid from orders
);
```

184. 部门工资最高的员工

184. 部门工资最高的员工

难度 中等 231 收藏 分享 切换为英文 关注 反馈

SQL架构 ›

`Employee` 表包含所有员工信息，每个员工有其对应的 Id, salary 和 department Id。

```
+----+-------+--------+--------------+
| Id | Name  | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 70000  | 1            |
| 2  | Henry | 80000  | 2            |
| 3  | Sam   | 60000  | 2            |
| 4  | Max   | 90000  | 1            |
+----+-------+--------+--------------+
```

`Department` 表包含公司所有部门的信息。

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

编写一个 SQL 查询，找出每个部门工资最高的员工。例如，根据上述给定的表格，Max 在 IT 部门有最高工资，Henry 在 Sales 部门有最高工资。

```
+------------+----------+--------+
| Department | Employee | Salary |
+------------+----------+--------+
| IT         | Max      | 90000  |
| Sales      | Henry    | 80000  |
+------------+----------+--------+
```

```sql
SELECT
    Department.name AS 'Department',
    Employee.name AS 'Employee',
    Salary
FROM
    Employee
        JOIN
    Department ON Employee.DepartmentId = Department.Id
WHERE
    (Employee.DepartmentId , Salary) IN
    (   SELECT
            DepartmentId, MAX(Salary)
        FROM
            Employee
        GROUP BY DepartmentId
    )
;
```

185. 部门工资前三高的所有员工

185. 部门工资前三高的所有员工

难度 困难 317 收藏 分享 切换为英文 关注 反馈

SQL架构 ›

`Employee` 表包含所有员工信息，每个员工有其对应的工号 `Id`，姓名 `Name`，工资 `Salary` 和部门编号 `DepartmentId` 。

```
+----+-------+--------+--------------+
| Id | Name  | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 85000  | 1            |
| 2  | Henry | 80000  | 2            |
| 3  | Sam   | 60000  | 2            |
| 4  | Max   | 90000  | 1            |
| 5  | Janet | 69000  | 1            |
| 6  | Randy | 85000  | 1            |
| 7  | Will  | 70000  | 1            |
+----+-------+--------+--------------+
```

`Department` 表包含公司所有部门的信息。

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

编写一个 SQL 查询，找出每个部门获得前三高工资的所有员工。例如，根据上述给定的表，查询结果应返回：

187. 重复的DNA序列

187. 重复的DNA序列

难度 中等 88 收藏 分享 切换为英文 关注 反馈

所有 DNA 都由一系列缩写为 A，C，G 和 T 的核苷酸组成，例如：“ACGAATTCCG”。在研究 DNA 时，识别 DNA 中的重复序列有时会对研究非常有帮助。

编写一个函数来查找 DNA 分子中所有出现超过一次的 10 个字母长的序列（子串）。

示例：

```
输入：s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"
输出：["AAAAACCCCC", "CCCCCAAAAA"]
```

```
class Solution {
```

```
public:
    vector<string> findRepeatedDnaSequences(string s) {
        unordered_map<string,int> hash;
        vector<string> res;

        for(int i = 0; i + 10 <= s.size();i ++)
        {
            string str = s.substr(i,10);
            hash[str] ++;
            if(hash[str] == 2) res.push_back(str);
        }
        return res;
    }
};
```

188. 买卖股票的最佳时机 IV

188. 买卖股票的最佳时机 IV

难度 困难 209 收藏 分享 切换为英文 关注 反馈

给定一个数组，它的第 *i* 个元素是一支给定的股票在第 *i* 天的价格。

设计一个算法来计算你所能获取的最大利润。你最多可以完成 **k** 笔交易。

注意: 你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

示例 1:

```
输入: [2,4,1], k = 2
输出: 2
解释: 在第 1 天 (股票价格 = 2) 的时候买入，在第 2 天 (股票价格 = 4) 的时候卖出，这笔交易所能获得利润 = 4-2 = 2 。
```

示例 2:

```
输入: [3,2,6,5,0,3], k = 2
输出: 7
解释: 在第 2 天 (股票价格 = 2) 的时候买入，在第 3 天 (股票价格 = 6) 的时候卖出，这笔交易所能获得利润 = 6-2 = 4 。
     随后，在第 5 天 (股票价格 = 0) 的时候买入，在第 6 天 (股票价格 = 3) 的时候卖出，这笔交易所能获得利润 = 3-0 = 3 。
```

```
class Solution {
public:
    int maxProfitII(vector<int>& prices) {
        int n = prices.size();
        int f, g;
        f = 0;
        g = -1000000000;
        for (int i = 0; i < n; i++) {
            int last_f = f, last_g = g;
            f = max(last_f, last_g + prices[i]);
            g = max(last_g, last_f - prices[i]);
        }
        return f;
```

```
    }
    int maxProfit(int k, vector<int>& prices) {
        int n = prices.size();
        if (k >= n / 2)
            return maxProfitII(prices);

        vector<vector<int>> f(2, vector<int>(k + 1, -1000000000));
        vector<vector<int>> g(2, vector<int>(k + 1, -1000000000));
        f[0][0] = 0;
        for (int i = 1; i <= n; i++)
            for (int j = 0; j <= k; j++) {
                f[i & 1][j] = f[i - 1 & 1][j];
                g[i & 1][j] = max(g[i - 1 & 1][j], f[i - 1 & 1][j] - prices[i -
1]);
                if (j > 0)
                    f[i & 1][j] = max(f[i & 1][j], g[i - 1 & 1][j - 1] +
prices[i - 1]);
            }
        int ans = 0;
        for (int i = 0; i <= k; i++)
            ans = max(ans, f[n & 1][i]);
        return ans;
    }
};
```

189. 旋转数组

189. 旋转数组

难度 简单 560 收藏 分享 切换为英文 关注 反馈

给定一个数组，将数组中的元素向右移动 *k* 个位置，其中 *k* 是非负数。

示例 1:

```
输入: [1,2,3,4,5,6,7] 和 k = 3
输出: [5,6,7,1,2,3,4]
解释:
向右旋转 1 步: [7,1,2,3,4,5,6]
向右旋转 2 步: [6,7,1,2,3,4,5]
向右旋转 3 步: [5,6,7,1,2,3,4]
```

示例 2:

```
输入: [-1,-100,3,99] 和 k = 2
输出: [3,99,-1,-100]
解释:
向右旋转 1 步: [99,-1,-100,3]
向右旋转 2 步: [3,99,-1,-100]
```

说明:

- 尽可能想出更多的解决方案，至少有三种不同的方法可以解决这个问题。
- 要求使用空间复杂度为 O(1) 的 **原地** 算法。

```cpp
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        k %= nums.size();
        reverse(nums.begin(), nums.end());
        reverse(nums.begin(), nums.begin() + k);
        reverse(nums.begin() + k, nums.end());
    }
};
```

190. 颠倒二进制位

190. 颠倒二进制位

难度 简单 159 收藏 分享 切换为英文 关注 反馈

颠倒给定的 32 位无符号整数的二进制位。

示例 1：

输入： 00000010100101000001111010011100
输出： 00111001011110000010100101000000
解释： 输入的二进制串 00000010100101000001111010011100 表示无符号整数 43261596，
因此返回 964176192，其二进制表示形式为 00111001011110000010100101000000。

示例 2：

输入： 11111111111111111111111111111101
输出： 10111111111111111111111111111111
解释： 输入的二进制串 11111111111111111111111111111101 表示无符号整数 4294967293，
因此返回 3221225471 其二进制表示形式为 10101111110010110010011101101001。

提示：

- 请注意，在某些语言（如 Java）中，没有无符号整数类型。在这种情况下，输入和输出都将被指定为有符号整数类型，并且不应影响您的实现，因为无论整数是有符号的还是无符号的，其内部的二进制表示形式都是相同的。
- 在 Java 中，编译器使用二进制补码记法来表示有符号整数。因此，在上面的 **示例 2** 中，输入表示有符号整数 `-3`，输出表示有符号整数 `-1073741825`。

进阶:
如果多次调用这个函数，你将如何优化你的算法?

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

191. 位1的个数

191. 位1的个数

难度 简单 160 收藏 分享 切换为英文 关注 反馈

编写一个函数，输入是一个无符号整数，返回其二进制表达式中数字位数为 '1' 的个数（也被称为汉明重量）。

示例 1：

```
输入：00000000000000000000000000001011
输出：3
解释：输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。
```

示例 2：

```
输入：00000000000000000000000010000000
输出：1
解释：输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。
```

示例 3：

```
输入：11111111111111111111111111111101
输出：31
解释：输入的二进制串 11111111111111111111111111111101 中，共有 31 位为 '1'。
```

提示：

- 请注意，在某些语言（如 Java）中，没有无符号整数类型。在这种情况下，输入和输出都将被指定为有符号整数类型，并且不应影响您的实现，因为无论整数是有符号的还是无符号的，其内部的二进制表示形式都是相同的。
- 在 Java 中，编译器使用二进制补码记法来表示有符号整数。因此，在上面的 **示例 3** 中，输入表示有符号整数 `-3`。

进阶:
如果多次调用这个函数，你将如何优化你的算法？

```cpp
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int res = 0;
        while (n)
        {
            res += n & 1;
            n >>= 1;
        }
        return res;
    }
};
```

198. 打家劫舍

198. 打家劫舍

难度 简单　755　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，影响你偷窃的唯一制约因素就是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：[1,2,3,1]
输出：4
解释：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 。
```

示例 2:

```
输入：[2,7,9,3,1]
输出：12
解释：偷窃 1 号房屋 (金额 = 2), 偷窃 3 号房屋 (金额 = 9)，接着偷窃 5 号房屋 (金额 = 1)。
     偷窃到的最高金额 = 2 + 9 + 1 = 12 。
```

```cpp
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        vector<int> f(n + 1),g(n + 1);
        for(int i = 1;i <= n;i ++)
        {
            f[i] = max(f[i - 1],g[i - 1]);
            g[i] = f[i - 1] + nums[i - 1];
        }
        return max(f[n],g[n]);
    }
};
```

199. 二叉树的右视图

199. 二叉树的右视图

难度 中等 221 收藏 分享 切换为英文 关注 反馈

给定一棵二叉树，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

示例:

```
输入：[1,2,3,null,5,null,4]
输出：[1, 3, 4]
解释：

   1            <---
 /   \
2     3         <---
 \     \
  5     4       <---
```

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int>res;
        if(!root)
            return res;
        vector<TreeNode*>Nodes;//记录按层遍历的节点的数组
        int l = 0, r = 0;
        Nodes.push_back(root);
        while(l<=r){//l r维护每一层的开始和结尾节点的位置
            res.push_back(Nodes[l]->val);//返回每层的第一个节点
            int cnt = 0;
            for(int i = l;i<=r;i++){
                if(Nodes[i]->right){//优先访问右边节点
                    Nodes.push_back(Nodes[i]->right);
                    cnt++;
                }
                if(Nodes[i]->left){
                    Nodes.push_back(Nodes[i]->left);
                    cnt++;
                }
            }
            l = r+1;
            r = l+cnt-1;
        }
        return res;
    }
};
```

200. 岛屿数量

200. 岛屿数量

难度 中等 545 收藏 分享 切换为英文 关注 反馈

给你一个由 '1'（陆地）和 '0'（水）组成的的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

示例 1:

```
输入：
11110
11010
11000
00000
输出：1
```

示例 2:

```
输入：
11000
11000
00100
00011
输出：3
解释：每座岛屿只能由水平和/或竖直方向上相邻的陆地连接而成。
```

```
class Solution {
public:
    int dx[4] = {0, 1, 0, -1};
    int dy[4] = {-1, 0, 1, 0};
    void dfs(int x, int y, vector<vector<char>>& grid) {
        grid[x][y] = '0';
        for (int i = 0; i < 4; i++) {
            int tx = x + dx[i], ty = y + dy[i];
            if (tx >= 0 && tx < grid.size() && ty >= 0 && ty < grid[0].size() &&
grid[tx][ty] == '1')
                dfs(tx, ty, grid);
        }
    }
    int numIslands(vector<vector<char>>& grid) {
        int n = grid.size();
        if (n == 0)
            return 0;
        int m = grid[0].size();
        int ans = 0;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                if (grid[i][j] == '1') {
                    ans++;
```

```
                    dfs(i, j, grid);
                }
        return ans;
    }
};
```