周赛【优先队列】【异或运算】

题目描述：

5736. 单线程 CPU

给你一个二维数组 `tasks` ，用于表示 `n` 项从 `0` 到 `n - 1` 编号的任务。其中 `tasks[i] = [enqueueTime_i, processingTime_i]` 意味着第 `i` 项任务将会于 `enqueueTime_i` 时进入任务队列，需要 `processingTime_i` 的时长完成执行。

现有一个单线程 CPU ，同一时间只能执行 **最多一项** 任务，该 CPU 将会按照下述方式运行：

- 如果 CPU 空闲，且任务队列中没有需要执行的任务，则 CPU 保持空闲状态。
- 如果 CPU 空闲，但任务队列中有需要执行的任务，则 CPU 将会选择 **执行时间最短** 的任务开始执行。如果多个任务具有同样的最短执行时间，则选择下标最小的任务开始执行。
- 一旦某项任务开始执行，CPU 在 **执行完整个任务** 前都不会停止。
- CPU 可以在完成一项任务后，立即开始执行一项新任务。

返回 CPU 处理任务的顺序。

题目分析：[优先队列]

```
class Solution {
    private:
    struct cmp2{
        bool operator()(vector<int>&a,vector<int>&b)
        {
            if(a[0]==b[0])return a[1]>b[1];//小的优先级高，所以是从小到大排序
            else return a[0]>b[0];
        }
    };
```

```
    struct cmp3{
        bool operator()(vector<int>&a,vector<int>&b)
        {
            if(a[1]==b[1])return a[2]>b[2];//小的优先级高，所以是从小到大排序
            else return a[1]>b[1];
        }
    };

public:
    vector<int> getOrder(vector<vector<int>>& tasks) {
        //采用优先队列
        priority_queue <vector<int>,vector<vector<int>>,cmp2 > q;
        //入队
        for(int i=0;i<tasks.size();i++){
            tasks[i].resize(3,i);
            q.push(tasks[i]);
        }
        //开始循环
        vector<int>ans;
        priority_queue <vector<int>,vector<vector<int>>,cmp3 > que;
        //进入队头元素
        que.push(q.top());
        q.pop();
        ans.push_back(que.top()[2]);
        long long int time=que.top()[0]-1,next_time=que.top()
[0]+que.top()[1];
        que.pop();
        while(true){
            //入队元素
            time=next_time;
            while((!q.empty()&&q.top()[0]<=time)){

                que.push(q.top());
                q.pop();
            }

            if(q.empty())break;
            //换下一道工序
            if(time==next_time||que.empty()){
                if(que.empty()){
                    que.push(q.top());
                    q.pop();
                }
                ans.push_back(que.top()[2]);
                next_time=time+que.top()[1];
                que.pop();
            }
        }
```

```
        while(!que.empty()){
            ans.push_back(que.top()[2]);
            que.pop();
        }
        return ans;
    }
};
```

题目描述：

5737. 所有数对按位与结果的异或和

难度 困难 3

列表的 **异或和**（**XOR sum**）指对所有元素进行按位 `XOR` 运算的结果。如果列表中仅有一个元素，那么其 **异或和** 就等于该元素。

- 例如，`[1,2,3,4]` 的 **异或和** 等于 `1 XOR 2 XOR 3 XOR 4 = 4` ，而 `[3]` 的 **异或和** 等于 `3` 。

给你两个下标 **从 0 开始** 计数的数组 `arr1` 和 `arr2` ，两数组均由非负整数组成。

根据每个 `(i, j)` 数对，构造一个由 `arr1[i] AND arr2[j]` （按位 `AND` 运算）结果组成的列表。其中 `0 <= i < arr1.length` 且 `0 <= j < arr2.length` 。

返回上述列表的 **异或和** 。

示例 1：

```
输入：arr1 = [1,2,3], arr2 = [6,5]
输出：0
解释：列表 = [1 AND 6, 1 AND 5, 2 AND 6, 2 AND 5, 3 AND 6, 3 AND 5] = [0,1,2,0,2,1] ，
异或和 = 0 XOR 1 XOR 2 XOR 0 XOR 2 XOR 1 = 0 。
```

示例 2：

```
输入：arr1 = [12], arr2 = [4]
输出：4
解释：列表 = [12 AND 4] = [4] ，异或和 = 4 。
```

题目分析：

1. 
2. ```
   class Solution {
   public:
       int getXORSum(vector<int>& arr1, vector<int>& arr2) {
           int m = arr1.size();
           int n = arr2.size();
           int ans = 0;
           // 依次确定答案二进制表示中的每一位
           for (int k = 30; k >= 0; --k) {
               int cnt1 = 0;
               for (int num: arr1) {
                   if (num & (1 << k)) {
                       ++cnt1;
                   }
               }
               int cnt2 = 0;
               for (int num: arr2) {
                   if (num & (1 << k)) {
                       ++cnt2;
                   }
               }
               // 如果 cnt1 和 cnt2 都是奇数，那么答案的第 k 位为 1
               if (cnt1 % 2 == 1 && cnt2 % 2 == 1) {
                   ans |= (1 << k);
               }
   ```
   ```

```
        }
        return ans;
    }
};
```

作者：LeetCode-Solution
链接：https://leetcode-cn.com/problems/find-xor-sum-of-all-pairs-bitwise-and/solution/find-xor-sum-of-all-pairs-bitwise-and-by-sok6/
来源：力扣（LeetCode）

3. 我们进行如下的推导：

   - 答案的第 k 位为 1

   等价于

   - $cnt_1[k]$ 为奇数且 $cnt_2[k]$ 为奇数

   等价于

   - 数组 arr_1 中二进制表示第 k 位的异或和为 1 且数组 arr_2 中二进制表示第 k 位的异或和为 1

   等价于

   - 数组 arr_1 中二进制表示第 k 位的异或和 $\wedge$ 数组 arr_2 中二进制表示第 k 位的异或和 $= 1$

4.
```
class Solution {
public:
    int getXORSum(vector<int>& arr1, vector<int>& arr2) {
        int tot1 = accumulate(arr1.begin(), arr1.end(), 0,
bit_xor<int>());
        int tot2 = accumulate(arr2.begin(), arr2.end(), 0,
bit_xor<int>());
        return tot1 & tot2;
    }
};
//bit_xor<int>()
//作用进行异或操作然后取其和
```

作者：LeetCode-Solution
链接：https://leetcode-cn.com/problems/find-xor-sum-of-all-pairs-bitwise-and/solution/find-xor-sum-of-all-pairs-bitwise-and-by-sok6/

来源：力扣（LeetCode）

5. accumulate定义在#include<numeric>中，作用有两个，一个是累加求和，另一个是自定义类型数据的处理

1.累加求和

```
int sum = accumulate(vec.begin() , vec.end() , 42);
```

accumulate带有三个形参：头两个形参指定要累加的元素范围，第三个形参则是累加的初值。

accumulate函数将它的一个内部变量设置为指定的初始值，然后在此初值上累加输入范围内所有元素的值。accumulate算法返回累加的结果，其返回类型就是其第三个实参的类型。

可以使用accumulate把string型的vector容器中的元素连接起来：

```
string sum = accumulate(v.begin() , v.end() , string(" "));
```

6.