位运算技巧

jask

10/05/2024

在集合论中，有交集 ∩∩、并集 ∪∪、包含于 ⊆⊆ 等等概念。如果编程实现「求两个哈希表的交集」，需要一个一个地遍历哈希表中的元素。那么，有没有效率更高的做法呢？

该二进制登场了。

集合可以用二进制表示，二进制从低到高第 ii 位为 11 表示 ii 在集合中，为 00 表示 ii 不在集合中。例如集合 {0,2,3}{0,2,3} 可以用二进制数 1101(2)1101(2) 表示；反过来，二进制数 1101(2)1101(2) 就对应着集合 {0,2,3}{0,2,3}。

例如集合 {0,2,3}{0,2,3} 可以压缩成 20+22+23=1320+22+23=13，也就是二进制数 1101(2)。

利用位运算「并行计算」的特点，我们可以高效地做一些和集合有关的运算。按照常见的应用场景，可以分为以下四类：

集合与集合
集合与元素
遍历集合
枚举集合

遍历集合

设元素范围从 00 到 n−1n−1，枚举范围中的元素 ii，判断 ii 是否在集合 ss 中。

```
for(int i=0;i<n;i++){
  if((s>>i)&1){//i 在 s 中

  }
}
```

也可以直接遍历集合 ss 中的元素：不断地计算集合最小元素、去掉最小元素，直到集合为空。

```
for(int t=s;t;t&=t-1){
  int i=__builtin_ctz(i);
}
```

枚举集合

设元素范围从 00 到 n−1n−1，从空集 ∅∅ 枚举到全集 UU：

```
for(int s=0;s< (1<<n);s++){

}
```

枚举非空子集

设集合为 ss，从大到小枚举 ss 的所有非空子集 subsub：

```
for (int sub = s; sub; sub = (sub - 1) & s) {
    // 处理 sub 的逻辑
}
```

枚举超集

如果 TT 是 SS 的子集，那么称 SS 是 TT 的超集（superset）。

枚举超集的原理和上文枚举子集是类似的，这里通过或运算保证枚举的集合 SS 一定包含集合 TT 中的所有元素。

枚举 SS，满足 SS 是 TT 的超集，也是全集 U={0,1,2,…,n−1}U={0,1,2,…,n−1} 的子集。

调用这些函数的时间复杂度都是 $\mathcal{O}(1)$。

术语	Python	Java	C++	Go
集合大小	`s.bit_count()`	`Integer.bitCount(s)`	`__builtin_popcount(s)`	`bits.OnesCount(s)`
二进制长度	`s.bit_length()`	`32-Integer.numberOfLeadingZeros(s)`	`__lg(s)+1`	`bits.Len(s)`
集合最大元素	`s.bit_length()-1`	`31-Integer.numberOfLeadingZeros(s)`	`__lg(s)`	`bits.Len(s)-1`
集合最小元素	`(s&-s).bit_length()-1`	`Integer.numberOfTrailingZeros(s)`	`__builtin_ctz(s)`	`bits.TrailingZeros(s)`

Figure 1: 常见操作

```
for (int s = t; s < (1 << n); s = (s + 1) | t) {
    // 处理 s 的逻辑
}
```

2595 奇偶位数

给你一个正整数 n 。

用 even 表示在 n 的二进制形式（下标从 0 开始）中值为 1 的偶数下标的个数。

用 odd 表示在 n 的二进制形式（下标从 0 开始）中值为 1 的奇数下标的个数。

返回整数数组 answer ，其中 answer = [even, odd] 。

常规解法

```
class Solution {
public:
    vector<int> evenOddBit(int n) {
        vector<int> ans(2);
        for(int i=0;n;i^=1,n>>=1){
            ans[i]+=n&1;
        }
        return ans;
    }
};
```

位掩码 + 库函数

利用位掩码 0x55555555（二进制的 010101），取出偶数下标比特和奇数下标比特，分别用库函数统计 1 的个数。

本题由于 n 范围比较小，取 0x5555 作为位掩码。

```
class Solution {
public:
    vector<int> evenOddBit(int n) {
        constexpr int MASK=0x5555;
        return {__builtin_popcount(n&MASK),__builtin_popcount(n&(MASK>>1))};
    }
};
```

476 数字的补数

对整数的二进制表示取反（0 变 1 ，1 变 0）后，再转换为十进制表示，可以得到这个整数的补数。

例如，整数 5 的二进制表示是 "101" ，取反后得到 "010" ，再转回十进制表示得到补数 2 。

给你一个整数 num ，输出它的补数。

```
class Solution {
public:
    int findComplement(int num) {
        return ~num&((1L<<(32-__builtin_clz(num)))-1);
    }
};
```

338 比特位统计

给你一个整数 n ，对于 0 <= i <= n 中的每个 i ，计算其二进制表示中 1 的个数，返回一个长度为 n + 1 的数组 ans 作为答案。

动态规划做法

```
class Solution {
 public:
  vector<int> countBits(int n) {
```

```
    vector<int> bits(n + 1);
    for (int i = 1; i <= n; i++) {

      bits[i] = bits[i >>1] + (i&1);
    }
    return bits;
  }
};
```

库函数做法

```
class Solution {
public:
    vector<int> countBits(int n) {
        vector<int> ans;
        for(int i=0;i<=n;i++){
            ans.emplace_back(__builtin_popcount(i));
        }
        return ans;
    }
};
```

3226 使两个整数相等的位更改次数

给你两个正整数 n 和 k。

你可以选择 n 的二进制表示中任意一个值为 1 的位，并将其改为 0。

返回使得 n 等于 k 所需要的更改次数。如果无法实现，返回 -1。

```
class Solution {
public:
    int minChanges(int n, int k) {
        return (n&k)!=k?-1: (__builtin_popcount(n^k));
    }
};
```

461 汉明距离

两个整数之间的汉明距离指的是这两个数字对应二进制位不同的位置的数目。

给你两个整数 x 和 y，计算并返回它们之间的汉明距离。

```
public:
    int hammingDistance(int x, int y) {
        return __builtin_popcount(x^y);
    }
};
```

868 二进制间距

给定一个正整数 n，找到并返回 n 的二进制表示中两个相邻 1 之间的最长距离。如果不存在两个相邻的 1，返回 0 。

如果只有 0 将两个 1 分隔开（可能不存在 0 ），则认为这两个 1 彼此相邻。两个 1 之间的距离是它们的二进制表示中位置的绝对差。例如，“1001”中的两个 1 的距离为 3 。

```
class Solution {
public:
    int binaryGap(int n) {
        int last=-1;
        int ans=0;
        for(int i=0;n;i++){
            if(n&1){
                if(last!=-1){
```

```
                    ans=max(ans,i-last);
                }
                last=i;
            }
            n>>=1;
        }
        return ans;
    }
};
```

3211 生成不含相邻零的二进制字符串

给你一个正整数 n。

如果一个二进制字符串 x 的所有长度为 2 的子字符串中包含至少一个“1”，则称 x 是一个有效字符串。

返回所有长度为 n 的有效字符串，可以以任意顺序排列。

```
class Solution {
public:
    vector<string> validStrings(int n) {
        vector<string> ans;
        int mask=(1<<n)-1;
        for(int i=0;i<(1<<n);i++){
            int x=mask^i;
            if(((x>>1)&x)==0){
                ans.push_back(bitset<18>(i).to_string().substr(18-n));
            }
        }
        return ans;
    }
};
```

2917 找出数组中的 K-or 值

给你一个整数数组 nums 和一个整数 k 。让我们通过扩展标准的按位或来介绍 K-or 操作。在 K-or 操作中，如果在 nums 中，至少存在 k 个元素的第 i 位值为 1 ，那么 K-or 中的第 i 位的值是 1 。

返回 nums 的 K-or 值。

```
class Solution {
public:
    int findKOr(vector<int>& nums, int k) {
        int ans=0;
        for(int i=0;i<32;i++){
            int count=0;
            for(auto num: nums){
                if(num&(1<<i)){
                    count++;
                }
            }
            if(count>=k){
                ans+=1<<i;
            }
        }
        return ans;
    }
};
```

693 交替位二进制数

给定一个正整数，检查它的二进制表示是否总是 0、1 交替出现：换句话说，就是二进制表示中相邻两位的数字永不相同。

```c++
class Solution {
public:
    bool hasAlternatingBits(int n) {
        bool tail=n&1;
        bool last=n&1;
        n>>=1;
        while(n){
            tail=(last^(n&1));
            if(!tail){
                return false;
            }
            last=n&1;
            n>>=1;
        }
        return tail;
    }
};
```

2657 找到两个数组的前缀公共数组

注意__builtin_popcountll是对于 long long 类型的

```c++
  class Solution {
public:
    vector<int> findThePrefixCommonArray(vector<int>& A, vector<int>& B) {
        vector<int> ans;
        long long setA=0;
        long long setB=0;
        int n=A.size();
        for(int i=0;i<n;i++){
            setA|=1LL<<A[i];
            setB|=1LL<<B[i];
            ans.push_back(__builtin_popcountll(setA&setB));
        }
        return ans;
    }
};
```

1486 数组异或操作

给你两个整数，n 和 start 。

数组 nums 定义为：nums[i] = start + 2*i（下标从 0 开始）且 n == nums.length 。

请返回 nums 中所有元素按位异或（XOR）后得到的结果。

利用异或的性质

```c++
class Solution {
public:
    vector<int> decode(vector<int>& encoded, int first) {
        int n=encoded.size()+1;
        vector<int> ans(n);
        ans[0]=first;
        for(int i=1;i<n;i++){
            ans[i]=ans[i-1]^encoded[i-1];
        }
        return ans;
    }
};
```

2433 找出前缀异或的原始数组

给你一个长度为 n 的整数数组 pref 。找出并返回满足下述条件且长度为 n 的数组 arr ：

pref[i] = arr[0] ^ arr[1] ^ ... ^ arr[i].

注意 ^ 表示按位异或（bitwise-xor）运算。

可以证明答案是唯一的。

```
class Solution {
public:
    vector<int> findArray(vector<int>& pref) {
        int n=pref.size();
        vector<int> ans(n);
        ans[0]=pref[0];
        int first=0;
        for(int i=0;i<n;i++){
            int tmp=first^pref[i];
            ans[i]=tmp;
            first ^=tmp;
        }
        return ans;
    }
};
```

1310 子数组异或查询

有一个正整数数组 arr，现给你一个对应的查询数组 queries，其中 queries[i] = [Li, Ri]。

对于每个查询 i，请你计算从 Li 到 Ri 的 XOR 值（即 arr[Li] xor arr[Li+1] xor ⋯xor arr[Ri]）作为本次查询的结果。

并返回一个包含给定查询 queries 所有结果的数组。

```
class Solution {
public:
    vector<int> xorQueries(vector<int>& arr, vector<vector<int>>& queries) {
        int n=arr.size();
        vector<int> xors(n+1);
        for(int i=0;i<n;i++){
            xors[i+1]=xors[i]^arr[i];
        }
        int m=queries.size();
        vector<int> ans(m);
        for(int i=0;i<m;i++){
            ans[i]=xors[queries[i][0]]^xors[queries[i][1]+1];
        }
        return ans;
    }
};
```

2683 相邻值的按位异或

下标从 0 开始、长度为 n 的数组 derived 是由同样长度为 n 的原始二进制数组 original 通过计算相邻值的按位异或（⊕）派生而来。

特别地，对于范围 [0, n - 1] 内的每个下标 i ：

如果 i = n - 1 ，那么 derived[i] = original[i] ⊕ original[0]
否则 derived[i] = original[i] ⊕ original[i + 1]

给你一个数组 derived ，请判断是否存在一个能够派生得到 derived 的有效原始二进制数组 original 。

如果存在满足要求的原始二进制数组，返回 true ；否则，返回 false 。

二进制数组是仅由 0 和 1 组成的数组。

$$original[i+1] = original[i] \oplus derived[i]$$

那么有

$$\begin{aligned} & original[n-1] \\ = {} & original[n-2] \oplus derived[n-2] \\ = {} & (original[n-3] \oplus derived[n-3]) \oplus derived[n-2] \\ & \vdots \\ = {} & original[0] \oplus derived[0] \oplus derived[1] \oplus \cdots \oplus derived[n-2] \end{aligned}$$

由于

$$original[0] \oplus original[n-1] = derived[n-1]$$

联立得

$$derived[0] \oplus derived[1] \oplus \cdots \oplus derived[n-1] = 0$$

所以如果上式成立，$original$ 必然存在。

Figure 2: 思路

```cpp
class Solution {
public:
    bool doesValidArrayExist(vector<int>& derived) {
        int tmp=0;
        for(int &x: derived){
            tmp^=x;
        }
        return tmp==0;
    }
};
```

2429 最小异或

给你两个正整数 num1 和 num2 ，找出满足下述条件的正整数 x ：

x 的置位数和 num2 相同，且
x XOR num1 的值 最小

注意 XOR 是按位异或运算。

返回整数 x 。题目保证，对于生成的测试用例，x 是唯一确定的。

整数的置位数是其二进制表示中 1 的数目。

基本思路:

x 的置位数和 num2 相同，意味着 x 的二进制表示中有 c2 个 1，我们需要合理地分配这 c2 个 1。

为了让异或和尽量小，这些 1 应当从高位到低位匹配 num1 中的 1；如果匹配完了还有多余的 1，那么就从低位到高位把 0 改成 1。

分类讨论:

如果 c2 n1，x 只能是 2^c2-1，任何其他方案都会使异或和变大；

如果 c2=c1，那么 x=num1；
如果 c2<c1，那么将 num1的最低的 c1-c2 个 1 变成 0，其结果就是 x；
如果 c2>c1，那么将 num1 的最低的 c2-c1 个 0 变成 1，其结果就是 x；

```
class Solution {
public:
    int minimizeXor(int num1, int num2) {
        int c1=__builtin_popcount(num1);
        int c2=__builtin_popcount(num2);
        for(;c2<c1;++c2) num1&=num1-1;
        for(;c2>c1;--c2) num1|=num1+1;
        return num1;
    }
};
```

2997 使数组异或和等于 K 的最少操作次数

给你一个下标从 **0** 开始的整数数组 **nums** 和一个正整数 **k** 。

你可以对数组执行以下操作任意次：

选择数组里的 任意 一个元素，并将它的 二进制 表示 翻转 一个数位，翻转数位表示将 0 变成 1 或者将 1 变成 0 。

你的目标是让数组里所有元素的按位异或和得到 **k** ，请你返回达成这一目标的最少操作次数。

注意，你也可以将一个数的前导 **0** 翻转。比方说，数字 **(101)2** 翻转第四个数位，得到 **(1101)2** 。

思路：

设 **x=s□k**，我们把 **nums** 中的任意数字的某个比特位翻转，那么 **x** 的这个比特位也会翻转。要让 **x=0**，就必须把 **x** 中的每个 **1** 都翻转，所以 **x** 中的 **1** 的个数就是我们的操作次数。

```
class Solution {
public:
    int minOperations(vector<int>& nums, int k) {
        for(auto &e:nums){
            k^=e;
        }
        return __builtin_popcount(k);
    }
};
```