Some Useful Algorithms

Boboge

ZUCC ACM Group

2022.01.23

Contents

Contents

Introduction

- 概念非常简单，代码量基本很小。

Introduction

- 概念非常简单，代码量基本很小。
- 眼睛看，脑子会，一上手，就报废。

Introduction

- 概念非常简单，代码量基本很小。
- 眼睛看，脑子会，一上手，就报废。
- 通过讲题直接搞懂一般套路。

Introduction

- 概念非常简单，代码量基本很小。
- 眼睛看，脑子会，一上手，就报废。
- 通过讲题直接搞懂一般套路。
- Practice! Practice! Practice!

Contents

Why Binary Search?

爱上🦕涩图bot 2021年07月01日10:41:00

最高の命令：二分探索を理解することを学ぶ

[圣经]

RE: Binary Search

工作原理

- 以在一个升序数组中查找一个数为例。
- 它每次考察数组当前部分的中间元素，如果中间元素刚好是要找的，就结束搜索过程
- 如果中间元素小于所查找的值，那么左侧的只会更小，不会有所查找的元素，只需到右侧查找
- 如果中间元素大于所查找的值同理，只需到左侧查找
- 由于每次将搜索范围减小至原先的一半以内，最多需要 $log_2(n) + 1$ 次查询，即可找到我们需要的元素。

RE: Binary Search

代码实现

```
int l = 1, r = n;
while (l <= r) {
    int mid = (l + r) / 2;
    if (a[mid] <= key) {
        l = mid + 1;
        pos = mid;
    } else {
        r = mid - 1;
    }
}
```

OR

```
lower_bound(a + 1, a + 1 + n, key) - a;
```

obviously $O(log_2 n)$ time & $O(1)$ space

RE: Binary Search - STL

STL 自带基于二分的函数

- lower_bound(begin, end, key); 可以找到升序数组中第一个值大于等于 key 的位置，注意，他返回的是变量的地址，比如数组的第几个位置。
- upper_bound(begin, end, key); 可以找到升序数组中第一个值大于 key 的位置。
- 注意，set 和 map 中使用这两个函数需要使用容器自带的函数，否则时间复杂度将超过 log 级别，可以认为是线性的。

```
vector<int> v;
auto position = lower_bound(v.begin(), v.end(), key);
set<int> s;
auto position = s.lower_bound(key);
```

Essence Of Binary Search

- 实际上二分是一种思想。
- 很多题目在特定条件下满足单调性，我们都可以使用二分。
- (关于什么是单调性，通过题目详细展开)
- 一千个人可以有一千种二分的方式，我个人推荐大家可以去理解上一页中的模版。

Sample: Guess the Number

- 交互题
- 选手提交的程序发起特定询问，评测程序返回询问的结果。
- 一般会限制询问次数以及格式。
- 你的程序需要通过评测程序的回答推出题目的答案并返回给评测程序。
- 注意！每次输出完询问或者回答后，需要手动清理缓冲区。
- std::endl; fflush(stdout); std::flush; 都可以完成这个需求。

Sample: Guess the Number

- Problem - Guess the Number
- 评测程序现有一个整数 $x(1 \leq x \leq 10^6)$，你需要把它的值猜出来。
- 你可以猜（也就是询问）一个值，系统会返回比答案大或者是小于等于。
- 你需要在最后输出这个值。
- 现在我们限定，你最多可以询问 25 次。

Solution: Guess the Number

- 注意到我们最多只能询问 25 次，可以反应过来限定的是 log 级的询问次数。

Solution: Guess the Number

- 注意到我们最多只能询问 25 次，可以反应过来限定的是 log 级的询问次数。
- 其实相当于，维护这个值可能存在区间，初始是 $[1, 10^6]$。
- 利用二分的思想，我们每次询问区间中间的值，根据程序返回的结果，我们可以知道起码有一半的区间是非法的。

Solution: Guess the Number

- 注意到我们最多只能询问 25 次，可以反应过来限定的是 log 级的询问次数。
- 其实相当于，维护这个值可能存在区间，初始是 $[1, 10^6]$。
- 利用二分的思想，我们每次询问区间中间的值，根据程序返回的结果，我们可以知道起码有一半的区间是非法的。
- 详细地说就是，若 mid 小于等于目标数字则 mid 右半边的区间非法，否则左半边的区间非法。

Binary Search for Answer

- 二分答案是基于二分思想的一种解题套路。

Binary Search for Answer

- 二分答案是基于二分思想的一种解题套路。
- 有时我们会通过枚举答案的值，然后 check 答案的合法性，以求出答案。

Binary Search for Answer

- 二分答案是基于二分思想的一种解题套路。
- 有时我们会通过枚举答案的值，然后 check 答案的合法性，以求出答案。
- 考虑二分需要基本性质：单调。
- 如果答案区间是单调的（比如大于某个值就合法之类的），我们就可以通过二分答案来求解出这个区间的边界。

Binary Search for Answer

- 二分答案是基于二分思想的一种解题套路。
- 有时我们会通过枚举答案的值，然后 check 答案的合法性，以求出答案。
- 考虑二分需要基本性质：单调。
- 如果答案区间是单调的（比如大于某个值就合法之类的），我们就可以通过二分答案来求解出这个区间的边界。
- 常见的一些：最大的最小值，最小的最大值，最大的最小的最大值……

Start with sqrt(x)

- 要求手动实现一个整数开根号。

Start with sqrt(x)

- 要求手动实现一个整数开根号。
- 本质上就是二分答案，此时的答案即开根后的结果，当结果的平方小于等于 x 时合法，我们要找出最大的合法值。
- 大多数情况下没有库函数跑得快，但可以保证精准。

Practice

- P1182 数列分段 Section II
- 简要题意：将一个长度为 n 的数组划分为 m 段，要求使每段和的最大值最小，求这个最小值。
- 例如 $n = 5$, $m = 3$, 将 [4 2 4 5 1] 分为 [4][2 4][5 1] 时，三段和分别为 4, 6, 6, 此时这个值为 6。

Practice

- P1182 数列分段 Section II
- 简要题意：将一个长度为 n 的数组划分为 m 段，要求使每段和的最大值最小，求这个最小值。
- 例如 $n = 5$, $m = 3$, 将 [4 2 4 5 1] 分为 [4][2 4][5 1] 时，三段和分别为 4, 6, 6, 此时这个值为 6。

Solution

显然该答案越大越容易合法，同时若我们知道这个值，一个较为显然的贪心方案为，每当值要超出时，即划分新的一段。若段数量不超过 m 则合法。所以我们对于答案进行二分即可。

Practice

- Problem - Poisoned Dagger
- 简要题意：流血狗每次攻击会流血 k 秒，流血状态下每秒减 1 点 hp，重复的攻击不会叠加而是重置流血效果。给出攻击次数与时间，问能杀死怪物最小的 k。

Practice

- Problem - Poisoned Dagger
- 简要题意：流血狗每次攻击会流血 k 秒，流血状态下每秒减 1 点 hp，重复的攻击不会叠加而是重置流血效果。给出攻击次数与时间，问能杀死怪物最小的 k。

Solution

显然 k 越大越容易杀死怪物。同时，若我们确定 k 的值，非常好 check 是否可以杀死怪物。那么对 k 二分即可。

Practice

- P1577 切绳子
- 简要题意：有 n 条长度不同的绳子，现在需要切出 k 条长度相同的绳子（不能拼接），问最大的长度（浮点数）

Practice

- P1577 切绳子
- 简要题意：有 n 条长度不同的绳子，现在需要切出 k 条长度相同的绳子（不能拼接），问最大的长度（浮点数）

Solution

答案合法性满足单调性，但是浮点数要怎么判断二分的结束条件呢？这里我们可以利用二分 30-100 次可以保证精度足够的技巧解决（浮点数二分三分均可以使用该技巧）。

Contents

Why should we learn Bit?

- 计算机本质是 0/1 组成的。
- 0/1 可以表示是/否两个状态，那么一系列的状态可以用一个 01 串表示，其实也就是一个二进制数字。也就是我们可以用数字 (int) 表示状态。
- 很多状态的变化我们可以通过位运算完成。
- 比赛中会出现很多涉及位运算的题目。
- ~~位运算常数比较小~~

RE: bit operations

- OR 或，AND 与
- XOR 异或
- 左移右移
- NOT 取反（个人经验，用的比较少）

EX: how 32-bits represent integers?

- 学术名：补码
- 第 1-31 位中第 i 位的权值是 2^i 。
- 第 32 位的权制是 -2^{32} 。

EX: how 32-bits represent integers?

- 学术名：补码
- 第 1-31 位中第 i 位的权值是 2^i 。
- 第 32 位的权制是 -2^{32} 。
- small task: 为什么二进制全 1 的 int 实际值是 -1 ？

OR, AND

- OR 对于位：有 1 得 1，否则得 0。
- AND 对于位：全 1 得 1，否则得 0。

OR, AND

- OR 对于位：有 1 得 1，否则得 0。
- AND 对于位：全 1 得 1，否则得 0。
- 对于 int 类型，OR 和 AND 操作都会对每一位（包括符号位）做该操作。

OR, AND

- OR 对于位：有 1 得 1，否则得 0。
- AND 对于位：全 1 得 1，否则得 0。
- 对于 int 类型，OR 和 AND 操作都会对每一位（包括符号位）做该操作。
- 强烈建议不要对负数进行各种位运算，主要是队友大概率会看不懂。

XOR

- XOR 对于位：不同得 1，否则得 0。
- 对于 int 类型，XOR 操作会对每一位（包括符号位）做该操作。
- XOR 运算有交换律，结合律。

XOR

- XOR 对于位：不同得 1，否则得 0。
- 对于 int 类型，XOR 操作会对每一位（包括符号位）做该操作。
- XOR 运算有交换律，结合律。

Important theorem

XOR 具有前缀性质。我们可以用维护前缀和的方式维护异或和。即 [l, r] 的区间异或和可以通过 sum[r] XOR sum[l - 1] 得到。这里 sum[i] 代表的是区间 [1, i] 的异或和。进阶：可以由树状数组动态维护。

shift left & shift right

- 左移即 *2，右移即/2。
- 显然，左移后末位会补 0，右移会舍掉末位。
- 可以将一个 O(log) 级的操作降至 O(1)。

shift left & shift right

- 左移即 *2，右移即/2。
- 显然，左移后末位会补 0，右移会舍掉末位。
- 可以将一个 O(log) 级的操作降至 O(1)。
- 不展开左移至溢出的情况，计算机系统原理会讲，算法竞赛中不推荐使用，因为大概率会搞错，而且对队友的血压不友好。

NOT

- 0 变 1，1 变 0。
- 不常用。
- -1 取反为 0。

Some external functions of bit operations

`std::__builtin_popcount(x)` //x的二进制表示中1的个数。
`std::__builtin_parity(x)` //x的二进制表示中1的个数的奇偶性。
`std::__lg(x)` //相当于$log2(x)$，但是返回的是整数，不会出现精度问题。
`x & (-x)` //返回最低位的1的实际值，一般命名为$lowbit$封装成函数。
`x >> i & 1` //判断x的第i位是1还是0。

divide Bits

- 拆位思想，也可以说是一种做题套路。
- 对于一些涉及位运算的题目，我们会观察他的性质，有时我们会发现，每一位对于答案的贡献是可以单独计算的，于是我们就可以枚举每一位去方便地计算答案。

divide Bits

- 拆位思想，也可以说是一种做题套路。
- 对于一些涉及位运算的题目，我们会观察他的性质，有时我们会发现，每一位对于答案的贡献是可以单独计算的，于是我们就可以枚举每一位去方便地计算答案。
- 其实 ACM 中“拆”是一种很常见的套路，很多时候整体考虑会比较困难，我们会把部分东西拆开考虑，然后枚举它，算得上是一种较为通用的思想。

Practice

- B. And It's Non-Zero
- 简要题意：多次询问 [l, r] 内最少删几个数可以使得区间与不等于 0.

Practice

- B. And It's Non-Zero
- 简要题意：多次询问 [l, r] 内最少删几个数可以使得区间与不等于 0.

Solution

考虑某一位与起来不等于 0 即可达成题目要求条件。我们枚举保留的那一位，不考虑其他位的情况，即可知道答案。

Practice

- E. Binary Numbers AND Sum
- 简要题意：现有两个大二进制整数 a，b，重复执行 ans += a & b; b /= 2; 直到 b == 0，求 ans（对 998244353 取模）。

Practice

- E. Binary Numbers AND Sum
- 简要题意：现有两个大二进制整数 a，b，重复执行 ans += a & b; b /= 2; 直到 b == 0，求 ans（对 998244353 取模）。

Solution

由于 b 中的 0 实际上不会产生贡献，我们分别考虑 b 中的每一位 1 实际扫过了多少 a 中的 1，用前缀和计算即可。

Contents

Offline Algo

- 离线算法
- 将所有输入保存后统一处理再输出。

Offline Algo

- 离线算法
- 将所有输入保存后统一处理再输出。

Attention

某些题目会要求强制在线（输入与上一组输出有关）。

Discretization

- 离散化是一种套路
- 需要离线

Discretization with STL

举例：离散化一个数组

```
//index start from 1
sort(a + 1, a + 1 + n);
map<int, int> id;
int cnt = 0;
for (int i = 1; i <= n; ++i) {
    if (id.count(a[i]) continue;
    else id[a[i]] = ++cnt;
}
//or you can use std::unique()
vector<int> vec;
sort(vec.begin(), vec.end());
vec.erase(unique(vec.begin(), vec.end()), vec.end());

int getid(int x) {
    return lower_bound(vec.begin(), vec.end(), x) - vec.begin() + 1;
}
```

Application

现在有 $n \leq 2 * 10^5$ 个区间，每个区间从覆盖 [l, r]$(1 \leq l \leq r \leq 10^9)$，输出被覆盖次数最多的坐标。

Application

现在有 $n \leq 2 * 10^5$ 个区间，每个区间从覆盖 [l, r]$(1 \leq l \leq r \leq 10^9)$，输出被覆盖次数最多的坐标。

Solution

比较显然可以用差分统计，但是我们不可能开 10^9 大小的数组，此时应用离散化即可。