C++，Java 和二分三分

郭炼

哈尔滨工业大学
计算学部

2020 年 7 月 27 号

哈爾濱工業大學

目录

目录

时间安排

此次集训采用线上训练的模式

- 7.27 — 8.28 共 5 周，其中
  - 周一到周五：

上午	9:00 — 11:30	专题授课
下午	12:30 — 17:30	专题测试
晚上	18:30 —	讲题

  - 周六：

下午	12:30 — 17:30	阶段测试
晚上	18:30 —	讲题

Rating 计算

- 单场得分 = 本人本场解题数 / SUM * (50 – min (rank, 25)) * k * 20
- 单场补题得分 = 本人本场补题数 / SUM * 1000
- SUM = 所有人本场解题数之和 (补题不计)
- rank = 本人当场排名 (第一名 rank = 0)
- k = 本场比赛得分系数

Rating 得分系数

各个测试对应的得分系数

- 周一到周五 1.5
- 周六比赛 9.5
- 终测 60.0

最终得分 = 单场得分最高的 25 场比赛得分之和

额外加分

- Additional：至少参加 5 场 Codeforces Contest，并且有至少 3 场在 7 月 15 日之后完成的账号可以计算加分
- 加分公式为系数 * (Max Rating -1300)，其中 Max Rating 为本账号截止到暑假集训结束的历史最高分

注意

- 单场比赛当前未签到，得分计 0 分
- 终测会尽可能在线下进行，如果在线上进行，会适当降低其得分系数

诚信问题

以下行为会被认定为作弊：

- 代码与网上的代码高度相似
- 在测试期间查阅 CSDN 等博客
- 使用小号，或是直接在原 OJ 处提交

后果

直接开除集训资格
视情节严重情况可能会上报教务处

第一周

- C++，JAVA 代码基础练习，杂题
- STL，队列，栈，并查集
- 数论初步
- 第一阶段测试

第二周

- 搜索
- 动态规划基础
- 树状数组 + 线段树
- 平衡树
- HASH，KMP，马拉车，trie
- 第二阶段测试

第三周

- AC 自动机
- 图论基础，欧拉回路，最短路
- 强连通分量相关
- 基础最大流，费用流
- 二分图最大带权匹配
- 第三阶段测试

第四周

- 区间，树 dp
- 计算几何基础
- 凸包，半平面交，旋转卡壳
- lca 和 rmq
- 数位 dp 和概率 dp
- 第四阶段测试

第五周

- 状压 DP，优化 DP
- 网络流建图
- 数论
- 博弈
- 2-SAT
- 后缀三连（8 月 31 日）
- 终测

目录

C++ 与 C 的区别

诸位都是学过 C 或 C++ 的人。在程序设计竞赛中，大部分选手使用的是 C++。

C++ 几乎是 C 的超集。C++ 有一些额外的功能：

- 输入：C++ 可以使用 cin/cout 来输入/输出
- 临时变量：C++ 可以在任意位置声明临时变量
- 模板：便于复用和编写板子
- 最重要的，STL 库的使用

Java

Java 在程序设计竞赛中一般用于编写高精度相关的题目。

输入/输出

```java
import java.util.*;
public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (scanner.hasNext()) {
      int a = scanner.nextInt();
      int b = scanner.nextInt();
      System.out.println("a + b = " + a + b);
    }
  }
}
```

高精度

高精度整数 BigInteger 类，需要引入对应的包。

```
import java.math.BigInteger;
public class Main {
  public static void main(String[] args) {
    BigInteger a = BigInteger.valueOf(100);
    BigInteger b = BigInteger.valueOf("
        100000000000000000000000000");
    System.out.println("a + b = " + a.add(b));
  }
}
```

BigInteger 常用函数

- BigInteger add(BigInteger other)
- BigInteger subtract(BigInteger other)
- BigInteger multiply(BigInteger other)
- BigInteger divide(BigInteger other)
- BigInteger mod(BigInteger other)
- int compareTo(BigInteger other)
- new BigInteger(scanner.next(),base); //读入 base 进制
- System.out.println(p.toString(base)); //输出成 base 进制

类似地，Java 还有高精度小数 BigDecimal

注意事项

- 一般比赛中 Java 拥有 2 倍时限（但可能还是不够用
- IO 效率缓慢（可以使用 Buffer 加速
- 通常仅限于高精度题目

目录

- 今天，我们来讲一点简单的内容（大佬轻虐）。
- 大家平时如何查纸制辞典？

基本思想

二分法的基本思想

二分算法是基于函数的单调性，如果一个函数 $f(x) : \mathbb{R}(\text{或}\mathbb{Z}) \to \{T, F\}$，满足

$$\forall x \leq x_0, f(x) = T, \forall x > x_0, f(x) = F$$

那么我们就可以使用二分法来求解 x_0。

基本思想

二分法的基本思想

二分算法是基于函数的单调性，如果一个函数 $f(x) : \mathbb{R}(\text{或}\mathbb{Z}) \to \{T, F\}$，满足

$$\forall x \leq x_0, f(x) = T, \forall x > x_0, f(x) = F$$

那么我们就可以使用二分法来求解 x_0。

- 我们来具一个 simple 的例子。

二分查找

二分查找

给定一个长度为 n 的序列 $A_1, A_2, \ldots, A_n$，保证 $A_1 \leq A_2, \ldots, \leq A_n$。查询这个序列中是否存在一个数 x

思考

- 一个 $O(n)$ for 循环做法是十分显然的。

思考

- 一个 $O(n)$ for 循环做法是十分显然的。
- 有没有更快的做法呢？

思考

- 一个 $O(n)$ for 循环做法是十分显然的。
- 有没有更快的做法呢？
- 考虑原数列的性质

思考

- 一个 $O(n)$ for 循环做法是十分显然的。
- 有没有更快的做法呢？
- 考虑原数列的性质
- 如果 $A_i \le x$，那么 $\forall j < i, A_j \le A_i \le x$。

思考

- 一个 $O(n)$ for 循环做法是十分显然的。
- 有没有更快的做法呢？

- 考虑原数列的性质
- 如果 $A_i \le x$，那么 $\forall j < i, A_j \le A_i \le x$。
- 同理，如果 $A_i > x$，那么 $\forall j > i, A_j \ge A_i > x$。

解法

- 于是我们每次只需要找到当前询问区间的中点（mid）

解法

- 于是我们每次只需要找到当前询问区间的中点（mid）
- 如果 x 在中点左侧（即 $A_{mid} \geq x$）则舍弃中点右侧的所有值，
- 反之舍弃中点左侧的所有值

解法

- 于是我们每次只需要找到当前询问区间的中点（mid）
- 如果 x 在中点左侧（即 $A_{mid} \geq x$）则舍弃中点右侧的所有值，
- 反之舍弃中点左侧的所有值
- 由于每次都将问题规模缩小一半，时间复杂度 $O(\log_2 n)$

模板

二分的写法很多

```
auto binary = [&]() {
  int l = 1, r = n;
  while (l < r) {
    int mid = (l + r + 1) / 2;
    if (check(mid)) {
      l = mid;
    } else {
      r = mid - 1;
    }
  }
  return l;
};
```

```
auto binary = [&]() {
  int l = 1, r = n, ans;
  while (l <= r) {
    int mid = (l + r) / 2;
    if (check(mid)) {
      ans = mid;
      l = mid + 1;
    } else {
      r = mid - 1;
    }
  }
  return ans;
};
```

二分答案

二分答案

有一类题目，我们直接计算问题的答案是复杂的，但是验证一个答案是否合法是简单的，并且问题满足可二分性。我们可以考虑二分答案 + 验证的方法来求解。

二分答案

二分答案

有一类题目，我们直接计算问题的答案是复杂的，但是验证一个答案是否合法是简单的，并且问题满足可二分性。我们可以考虑二分答案 + 验证的方法来求解。

- 常见于最大值最小化（最小值最大化）问题。

$A \times B$ 问题

$A \times B$ 问题

给定两个正数数组 A 和 B，定义 $A \times B =$ 数组$\{A_i \times B_j\}$。求出 $A \times B$ 中的 k 大的数。

解法

- 答案在 $[min(A) \times min(B), max(A) \times B]$，我们在这个区间内二分答案

解法

- 答案在 $[min(A) \times min(B), max(A) \times B]$，我们在这个区间内二分答案
- 利用排序 + 二分查找，我们可以快速的算出 x 的排名
- 时间复杂度 $O(\log_2 \Delta \times |A| \log_2 |B|)$

分披萨

分披萨

现在有 N 块披萨，每块披萨的大小是 A_i。现在有 M 个人要分这些披萨，每个人只能从某一大块披萨中切一小块（必须完整），问怎么分配才能使得分到的披萨最小的人的披萨最大。

解法

- 这一个最小值最大化问题

解法

- 这一个最小值最大化问题
- 考虑二分最小的人拿到的大小

解法

- 这一个最小值最大化问题
- 考虑二分最小的人拿到的大小
- 我们强制让所有人取得的大小都一样
- 现在的问题转变成了：有 M 块披萨，每个人都需要拿走大小为 x 的部分，问披萨够不够分

解法

- 这一个最小值最大化问题
- 考虑二分最小的人拿到的大小
- 我们强制让所有人取得的大小都一样
- 现在的问题转变成了：有 M 块披萨，每个人都需要拿走大小为 x 的部分，问披萨够不够分
- 所以只需要判断 $\sum \lfloor \frac{A_i}{x} \rfloor \geq N$ 即可

砍树

砍树

给定 N 棵树，共需要 M 米长的木材。伐木机工作过程如下：伐木机升起一个巨大的锯片到高度 H，并锯掉所有的树比 H 高的部分。问 H 最高为多少，使得可以有 M 米长的木材。

$1 \le N \le 1000000, 1 \le M \le 2000000000, H \in \mathbb{N}^+$

解法

- 若 H 减少，那么我们获得的木材不会减少

解法

- 若 H 减少，那么我们获得的木材不会减少
- 有了单调性，我们就可以二分了

解法

- 若 H 减少，那么我们获得的木材不会减少
- 有了单调性，我们就可以二分了
- 判断 $\sum \max(tree_i - H, 0) \geq M$

跳石头

NOIP 2015 跳石头

以两块岩石作为比赛起点和终点。在起点和终点之间，有 N 块岩石，位置为 L_i（不含起点和终点的岩石）。现在至多从起点和终点之间移走 M 块岩石（不能移走起点和终点的岩石），使两块岩石间的最短距离尽可能长。询问最短距离的最大值。

$0 \le M \le N \le 50,000, 1 \le L \le 1,000,000,000$

解法

- 看到“最短距离”“最大值”，又是最小值最大化问题

解法

- 看到“最短距离”“最大值”，又是最小值最大化问题
- 考虑二分答案,Check 本答案需要移走的石子数量即可

Castle Defense

Codeforces 954G Castle Defense

给出一个长度为 n 的序列 $A_1, A_2, \ldots, A_n$，给出值 r, k，定义 $sum[i]$ 表示区间 $[i-r, i+r]$ 中 A_j 的和。你可以对这个序列进行不超过 k 次操作，每次操作选择一个 A_i 使其变为 $A_i + 1$。求 $\min(sum[i])$ 的最大值。

如给出长度为 5 的序列 $5, 4, 3, 4, 9$，给定 $r = 0, k = 6$，那么序列可以变为 $6, 6, 5, 5, 9$，得出最优解 $\min(sum[i]) = 5$

$1 \le r \le n \le 500000, k \le 10^{18}$

解法

- 询问的实际上是使得最小值最大，因此想到二分答案
- 显然 k 次操作会用完
- 对于每一个答案判断是否合法即可
- 考虑一个点如果需要补, 必然补在最右侧
- 差分一下即可

kotori 的设备

洛谷 P3743 kotori 的设备

kotori 共有 n 个可同时使用的设备，每个设备连续均匀地使用能量，速率为 a_i，初始时包含 b_i 的能量。kotori 有一个充电宝，每秒可以给接通的设备充能 p 个单位，充能也是连续的，设备切换时间不计。在充电器的作用下，她最多能将这些设备一起使用多久。

$$1 \le n \le 100000, 1 \le p \le 1000001 \le a_i, b_i \le 100000$$

二分答案

解法

- 单调性是显然的

解法

- 单调性是显然的
- 切换的时间不计，所以可以认为充电宝同时为 n 个设备充电，单个设备的瞬时充电功率之和小于 p

解法

- 单调性是显然的
- 切换的时间不计，所以可以认为充电宝同时为 n 个设备充电，单个设备的瞬时充电功率之和小于 p
- 判断所需功率之和与 p 的关系即可

目录

基本思想

- 二分是对一个单调的函数进行的操作

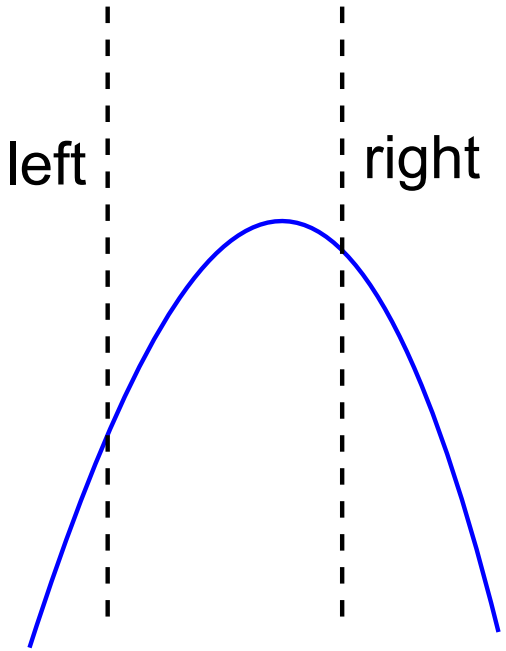
基本思想

- 二分是对一个单调的函数进行的操作
- 那么我们有没有办法对一个单峰的函数进行操作呢

基本思想

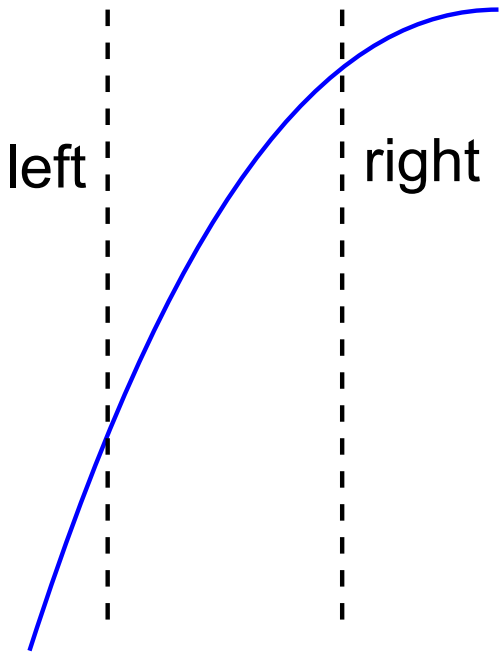
- 二分是对一个单调的函数进行的操作
- 那么我们有没有办法对一个单峰的函数进行操作呢
- 类似二分，我们将区间划分为三段，删除不可能称为答案的部分

基本思想



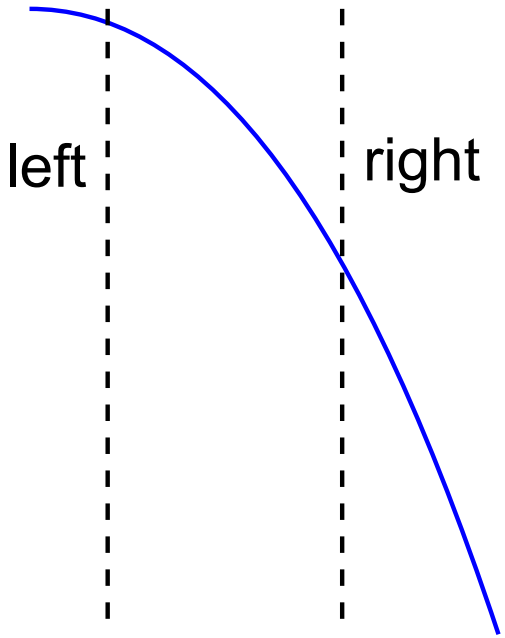
显然左边的部分不可能是答案了

基本思想



显然左边的部分不可能是答案了

基本思想



显然右边的部分不可能是答案了

- 发现共性：左边界和右边界中值较小的那一段一定会被舍去

- 发现共性：左边界和右边界中值较小的那一段一定会被舍去
- 这样我们就可以将问题的规模缩小 $\frac{1}{3}$

基本思想

- 发现共性：左边界和右边界中值较小的那一段一定会被舍去
- 这样我们就可以将问题的规模缩小 $\frac{1}{3}$
- 事实上我们取两次 mid 会好写很多，对复杂度的影响只是一个常系数

Maximize!

Codeforces 939E Maximize!

初始有一个为空的集合，要求支持两种操作

1. 不断向集合中插入一个数，且这个数比集合中所有数都大
2. 在集合中找一个子集，使得找到的子集 S 中的最大值减去子集 S 中元素的平均值的差最大，并输出这个差

操作数 ≤ 500000

解法

- 如何选取子集？

解法

- 如何选取子集？
- 最后插入的这个数是一定要选的，然后再选小的数

解法

- 如何选取子集？
- 最后插入的这个数是一定要选的，然后再选小的数
- 就是一个最大数加上几个用来拉低平均值的小数构成了所需子集

解法

- 如何选取子集？
- 最后插入的这个数是一定要选的，然后再选小的数
- 就是一个最大数加上几个用来拉低平均值的小数构成了所需子集
- 小数一定是从最小值开始连续增加使平均值减小，直到达到一个临界点

解法

- 如何选取子集？
- 最后插入的这个数是一定要选的，然后再选小的数
- 就是一个最大数加上几个用来拉低平均值的小数构成了所需子集
- 小数一定是从最小值开始连续增加使平均值减小，直到达到一个临界点
- 再增加小数就会使平均值增大，易知这是一个单峰函数

解法

- 如何选取子集？
- 最后插入的这个数是一定要选的，然后再选小的数
- 就是一个最大数加上几个用来拉低平均值的小数构成了所需子集
- 小数一定是从最小值开始连续增加使平均值减小，直到达到一个临界点
- 再增加小数就会使平均值增大，易知这是一个单峰函数
- 因此考虑三分选多少小数即可