二分算法

定义

二分算法（Binary Search），也称折半查找，是算法竞赛中最为核心的工具之一。

二分算法是一种在有序（具有单调性）的数据集合中，通过不断将搜索区间对半缩小，从而快速定位目标元素的算法。

- 核心思想：与其一个一个检查（线性查找），不如每次检查中间的元素。如果中间元素不是目标，就根据大小关系排除掉一半的可能性。
- 效率：其时间复杂度为 $O(\log N)$。

> 二分算法的使用有一个铁律：数据必须是有序的，或者说问题的解空间必须具有单调性 。

- 升序/降序：例如数组 [1, 3, 5, 7, 9]。
- 判定性质：例如“大于等于 x 的所有数都满足条件，小于 x 的都不满足”。

二分算法的分类：

- 整数二分
- 二分答案

整数二分

整数二分用于在一个整数区间内，寻找满足某种性质的边界值。由于整数是不连续的（1 和 2 之间没有整数），在缩小区间时，我们需要精确决定 mid 归属于左半部分还是右半部分。

寻找左侧边界

> 第一个满足条件的数

例如：在 [1, 2, 2, 2, 3] 中找第一个 2 的下标。

```cpp
#include <iostream>
using namespace std;

int main() {
    // 定义有序数组（非递减顺序）
    int a[] = {1, 2, 2, 2, 3};
    // 数组长度
    int n = 5;
    // 查找的目标值
    int target = 2;

    // 二分查找的左右边界指针
    int l = 0;        // 左边界指针，初始指向数组起始位置
    int r = n - 1;    // 右边界指针，初始指向数组末尾位置

    // 二分查找循环，当左指针小于右指针时继续
    while (l < r) {
        // 计算中间位置，使用位运算>>1相当于除以2（向下取整）
        // 注意：l + r >> 1 等同于 (l + r) / 2，但优先级要注意
        int mid = l + r >> 1;

        // 如果中间值大于等于目标值，说明目标值在左半部分（包括mid）
```

```
        // 将右边界移动到mid位置（因为我们要找的是第一个>=target的位置）
        if (a[mid] >= target) r = mid;
        // 否则，中间值小于目标值，说明目标值在右半部分
        // 将左边界移动到mid+1位置（mid位置肯定不是目标）
        else l = mid + 1;
    }

    // 循环结束后，l和r相等，指向第一个>=target的元素下标
    cout << "左边界下标: " << l << endl;

    // 验证：检查找到的位置是否确实等于目标值
    // 注意：这个算法找到的是第一个>=target的位置，不一定等于target
    // 例如如果target=4，会找到最后一个元素的下标（4），但a[4]=3<4

    return 0;
}
```

寻找右侧边界

最后一个满足条件的数

例如：在 [1, 2, 2, 2, 3] 中找最后一个 2 的下标。

注意： 此时计算 mid 需要补上 + 1，防止死循环。

```
#include <iostream>
using namespace std;

int main() {
    // 定义有序数组（非递减顺序）
    int a[] = {1, 2, 2, 2, 3};
    // 数组长度
    int n = 5;
    // 查找的目标值
    int target = 2;

    // 二分查找的左右边界指针
    int l = 0;        // 左边界指针，初始指向数组起始位置
    int r = n - 1;    // 右边界指针，初始指向数组末尾位置

    // 二分查找循环，当左指针小于右指针时继续
    // 注意：这是查找右边界的二分法，循环条件为l<r
    while (l < r) {
        // 计算中间位置，使用位运算>>1相当于除以2
        // 注意这里加了1：l + r + 1 >> 1，这是为了避免死循环
        // 当l和r相邻时，如果不加1，mid会等于l，可能导致无限循环
        int mid = l + r + 1 >> 1;

        // 如果中间值小于等于目标值，说明目标值在右半部分（包括mid）
        // 将左边界移动到mid位置（因为我们要找的是最后一个<=target的位置）
        if (a[mid] <= target) l = mid;
        // 否则，中间值大于目标值，说明目标值在左半部分
        // 将右边界移动到mid-1位置（mid位置肯定大于target）
        else r = mid - 1;
    }

    // 循环结束后，l和r相等，指向最后一个<=target的元素下标
    // 注意：如果数组中所有元素都大于target，那么l会等于0（第一个元素下标）
    cout << "右边界下标: " << l << endl;

    // 验证：检查找到的位置是否确实等于目标值
    // 注意：这个算法找到的是最后一个<=target的位置，不一定等于target
```

```
    return 0;
}
```

二分答案

与“在数组中找一个数”不同，二分答案是：通过二分“可能的答案范围”，来反向验证这个答案是否可行。

- 普通二分：在一个已有的有序序列里找一个值。
- 二分答案：在一个不存在的、虚拟的“答案区间”里找一个最优值。

如果一个问题满足以下两个特征，通常就可以用二分答案：

1. 答案具有单调性：
- 如果长度为 x 的方案可行，那么长度小于 x 的方案一定都可行（求最大值）。
- 或者：如果速度为 v 满足条件，那么速度大于 v 的一定也满足（求最小值）。
2. 正向求解困难，但反向验证容易：
- 直接计算“最优解”很难，但给你一个确定的数 x，让你判断“x 是否符合要求”很简单。

解题三部曲

根据蓝桥杯的编程大题要求，选手需要优先选择兼具可行性与效率的算法。

1. 第一步：确定答案范围 [L, R]
- 找出答案可能的最小值和最大值。
- 例如：切割绳子，最短可能是 0，最长可能是最长的那根原始绳子。
2. 第二步：编写 check(mid) 函数
- 这是二分答案的核心。
- 逻辑：假设当前的答案是 mid，编写一段代码判断它是否满足题目要求。
- 要求：通常使用贪心或模拟的思想来写 check。
3. 第三步：套用二分模板
- 在 [L, R] 范围内进行二分，根据 check(mid) 的布尔值缩小区间。

例题

假设题目问：有一块 2019×324 的材料，如果要切出 K 个边长为 x 的相同正方形，求 x 的最大值。

1. 范围：x 的范围是 `[1, 324]`。
2. check(x)：
    - 计算横着能切几个：$2019/x$。
    - 计算竖着能切几个：$324/x$。
    - 总数 = $(2019/x) \times (324/x)$。
    - 如果 总数 $\geq K$，返回 `true`，否则返回 `false`。
3. 二分：在 `[1, 324]` 里二分查找满足条件的最大 x。

由于正方形的边长 x 越大，能切出的数量 K 就越少，这满足了单调性，因此我们可以二分搜索 x 的值。

解题思路

- 答案范围：x 最小为 1，最大不会超过材料的最短边（即 324）。
- Check 函数：给定一个边长 mid，计算在 2019×324 的矩形里能切出多少个 $mid \times mid$ 的正方形。数量计算公式为：$(2019/mid) \times (324/mid)$。

- 二分逻辑：如果切出的数量 $\geq K$，说明 mid 可能还能更大，去右半部分找；否则说明 mid 太大了，去左半部分找。

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

long long K; // 需要的最小正方形数量
int H = 2019; // 矩形的高度
int W = 324;  // 矩形的宽度

// 检查函数：判断使用边长为x的正方形是否能切割出至少K个正方形
bool check(int x) {
    if (x == 0) return true; // 边长为0时，可以认为满足条件（边界情况）

    // 计算矩形中可以切割出的边长为x的正方形数量
    // 高度方向可以切割出 H/x 个（整数除法）
    // 宽度方向可以切割出 W/x 个（整数除法）
    // 总数量 = (H/x) * (W/x)
    long long count = (long long)(H / x) * (W / x);

    // 判断数量是否满足需求
    return count >= K;
}

int main() {
    // 读取需要的最小正方形数量K
    if (!(cin >> K)) return 0;

    // 设置二分查找的边界
    // 左边界：至少为1（正方形边长最小为1）
    // 右边界：不能超过矩形的最小边，所以取H和W的最小值
    int l = 1, r = min(H, W);

    int ans = 0; // 存储最终答案（最大边长）

    // 二分查找：寻找满足条件的最大边长x
    while (l <= r) {
        // 计算中间值，防止溢出使用 l + (r - l) / 2
        int mid = l + (r - l) / 2;

        // 检查mid是否满足条件
        if (check(mid)) {
            // 如果满足条件，记录当前mid作为候选答案
            // 然后尝试更大的边长（因为要找最大的满足条件的边长）
            ans = mid;
            l = mid + 1; // 向右搜索，尝试更大的边长
        } else {
            // 如果不满足条件，说明边长太大了
            // 需要尝试更小的边长
            r = mid - 1; // 向左搜索，尝试更小的边长
        }
    }

    // 输出最大的正方形边长
    cout << ans << endl;

    return 0;
}
```