如何运用二分查找算法

相关推荐：

- 如何运用贪心思想玩跳跃游戏
- 如何寻找最长回文子串

读完本文，你不仅学会了算法套路，还可以顺便去 LeetCode 上拿下如下题目：

875.爱吃香蕉的珂珂

1011.在D天内送达包裹的能力

二分查找到底有能运用在哪里？

最常见的就是教科书上的例子，在**有序数组**中搜索给定的某个目标值的索引。再推广一点，如果目标值存在重复，修改版的二分查找可以返回目标值的左侧边界索引或者右侧边界索引。

PS：以上提到的三种二分查找算法形式在前文「二分查找详解」有代码详解，如果没看过强烈建议看看。

抛开有序数组这个枯燥的数据结构，二分查找如何运用到实际的算法问题中呢？当搜索空间有序的时候，就可以通过二分搜索「剪枝」，大幅提升效率。

说起来玄乎得很，本文先用一个具体的「Koko 吃香蕉」的问题来举个例子。

一、问题分析

珂珂喜欢吃香蕉。这里有 `N` 堆香蕉，第 `i` 堆中有 `piles[i]` 根香蕉。警卫已经离开了，将在 `H` 小时后回来。

珂珂可以决定她吃香蕉的速度 `K` （单位：根/小时）。每个小时，她将会选择一堆香蕉，从中吃掉 `K` 根。如果这堆香蕉少于 `K` 根，她将吃掉这堆的所有香蕉，然后这一小时内不会再吃更多的香蕉。

珂珂喜欢慢慢吃，但仍然想在警卫回来前吃掉所有的香蕉。

返回她可以在 `H` 小时内吃掉所有香蕉的最小速度 `K`（`K` 为整数）。

示例 1：

```
输入: piles = [3,6,7,11], H = 8
输出: 4
```

示例 2：

```
输入: piles = [30,11,23,4,20], H = 5
输出: 30
```

也就是说，Koko 每小时最多吃一堆香蕉，如果吃不下的话留到下一小时再吃；如果吃完了这一堆还有胃口，也只会等到下一小时才会吃下一堆。在这个条件下，让我们确定 Koko 吃香蕉的**最小速度**（根/小时）。

如果直接给你这个情景，你能想到哪里能用到二分查找算法吗？如果没有见过类似的问题，恐怕是很难把这个问题和二分查找联系起来的。

那么我们先抛开二分查找技巧，想想如何暴力解决这个问题呢？

首先，算法要求的是「`H` 小时内吃完香蕉的最小速度」，我们不妨称为 `speed`，请问 `speed` 最大可能为多少，最少可能为多少呢？

显然最少为 1，最大为 `max(piles)`，因为一小时最多只能吃一堆香蕉。那么暴力解法就很简单了，只要从 1 开始穷举到 `max(piles)`，一旦发现发现某个值可以在 `H` 小时内吃完所有香蕉，这个值就是最小速度：

```
int minEatingSpeed(int[] piles, int H) {
  // piles 数组的最大值
    int max = getMax(piles);
    for (int speed = 1; speed < max; speed++) {
      // 以 speed 是否能在 H 小时内吃完香蕉
        if (canFinish(piles, speed, H))
            return speed;
    }
    return max;
}
```

注意这个 for 循环，就是在**连续的空间线性搜索，这就是二分查找可以发挥作用的标志**。由于我们要求的是最小速度，所以可以用一个**搜索左侧边界的二分查找**来代替线性搜索，提升效率：

```
int minEatingSpeed(int[] piles, int H) {
    // 套用搜索左侧边界的算法框架
    int left = 1, right = getMax(piles) + 1;
    while (left < right) {
        // 防止溢出
        int mid = left + (right - left) / 2;
        if (canFinish(piles, mid, H)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    return left;
}
```

PS：如果对于这个二分查找算法的细节问题有疑问，建议看下前文「二分查找详解」搜索左侧边界的算法模板，这里不展开了。

剩下的辅助函数也很简单，可以一步步拆解实现：

```
// 时间复杂度 O(N)
boolean canFinish(int[] piles, int speed, int H) {
    int time = 0;
    for (int n : piles) {
        time += timeOf(n, speed);
    }
    return time <= H;
}

int timeOf(int n, int speed) {
    return (n / speed) + ((n % speed > 0) ? 1 : 0);
}

int getMax(int[] piles) {
```

```
    int max = 0;
    for (int n : piles)
        max = Math.max(n, max);
    return max;
}
```

至此，借助二分查找技巧，算法的时间复杂度为 O(NlogN)。

二、扩展延伸

类似的，再看一道运输问题：

传送带上的第 `i` 个包裹的重量为 `weights[i]`。每一天，我们都会按给出重量的顺序往传送带上装载包裹。我们装载的重量不会超过船的最大运载重量。

返回能在 `D` 天内将传送带上的所有包裹送达的船的最低运载能力。

示例 1：

输入：weights = [1,2,3,4,5,6,7,8,9,10], D = 5
输出：15
解释：
船舶最低载重 15 就能够在 5 天内送达所有包裹，如下所示：
第 1 天：1, 2, 3, 4, 5
第 2 天：6, 7
第 3 天：8
第 4 天：9
第 5 天：10

请注意，货物必须按照给定的顺序装运，因此使用载重能力为 14 的船舶并将包装分成 (2, 3, 4, 5), (1, 6, 7), (8), (9), (10) 是不允许的。

要在 `D` 天内运输完所有货物，货物不可分割，如何确定运输的最小载重呢（下文称为 `cap`）？

其实本质上和 Koko 吃香蕉的问题一样的，首先确定 `cap` 的最小值和最大值分别为 `max(weights)` 和 `sum(weights)`。

我们要求**最小载重**，所以可以用搜索左侧边界的二分查找算法优化线性搜索：

```
// 寻找左侧边界的二分查找
int shipWithinDays(int[] weights, int D) {
```

```
    // 载重可能的最小值
    int left = getMax(weights);
    // 载重可能的最大值 + 1
    int right = getSum(weights) + 1;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (canFinish(weights, D, mid)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    return left;
}

// 如果载重为 cap，是否能在 D 天内运完货物?
boolean canFinish(int[] w, int D, int cap) {
    int i = 0;
    for (int day = 0; day < D; day++) {
        int maxCap = cap;
        while ((maxCap -= w[i]) >= 0) {
            i++;
            if (i == w.length)
                return true;
        }
    }
    return false;
}
```

通过这两个例子，你是否明白了二分查找在实际问题中的应用?

```
for (int i = 0; i < n; i++)
    if (isOK(i))
        return ans;
```

刷算法，学套路，认准 labuladong，公众号和 在线电子书 持续更新最新文章。

本小抄即将出版，微信扫码关注公众号，后台回复「小抄」限时免费获取，回复「进群」可进刷题群一起刷题，带你搞定 LeetCode。

==其他语言代码==