链滴

贪心算法原理及其应用

作者：vcjmhg

原文链接：https://ld246.com/article/1604387967604

来源网站：链滴

概述

贪心算法应该算是那种“只闻其声不见其人”的算法，我们可能在好多地方都会听到贪心算法这一概，并且它的算法思想也比较简单就是说算法只保证局部最优，进而达到全局最优。但我们实际编程的程中用的并不是很多，究其原因可能是贪心算法使用的条件比较苛刻，所要解决的问题必须满足**贪心择性质**---所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心算法行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。

贪心算法

由于贪心算法本身的特殊性，我们在使用贪心算法之前必须要进行证明，保证算法满足贪心选择性质具体的证明方法无外乎就是通过数学归纳法来进行证明。但大部分人可能并不喜欢枯燥的公式，因而这里提供一个使用贪心算法的小技巧。由于贪心算法某种程度上算是动态规划算法的特例，使用条件较苛刻，因而能够用动态规划解决的问题**尽量都是用动态规划来进行先解决**，如果在用完动态规划之，提交时发现问题超时，并且进行状态压缩之后仍然**超时**，此时我们就可以**考虑使用贪心算法来进解决。**最后强调一下，我们在使用贪心算法之前，如果要保证解法的绝对正确，一定要对问题进行明，切记，切记！！

下边我们以区间调度问题为例，来讲一下贪心算法到底该如何取用。

区间调度问题

问题描述：

给你很多形如 [start, end] 的闭区间，请你设计一个算法，算出这些区间中最多有几个互不相交的区。

举个例子，intvs = [[1,3], [2,4], [3,6]]，这些区间最多有 2 个区间互不相交，即 [[1,3], [3,6]]，你的法应该返回 2。注意边界相同并不算相交。

这个问题大眼一看好像有很多贪心策略可供选择，比如我们可以选择区间最短的？或者选择开始最早？。。。

但是上面几种策略，我们都可以比较容易的举出反例来排除，同时这也是贪心算法的另一个**小技巧**--然好多时候直接证明贪心策略的正确性很难，**但是我们可以从反证法入手，对贪心策略进行证伪**，排许多错误的贪心策略。□smile□
mile

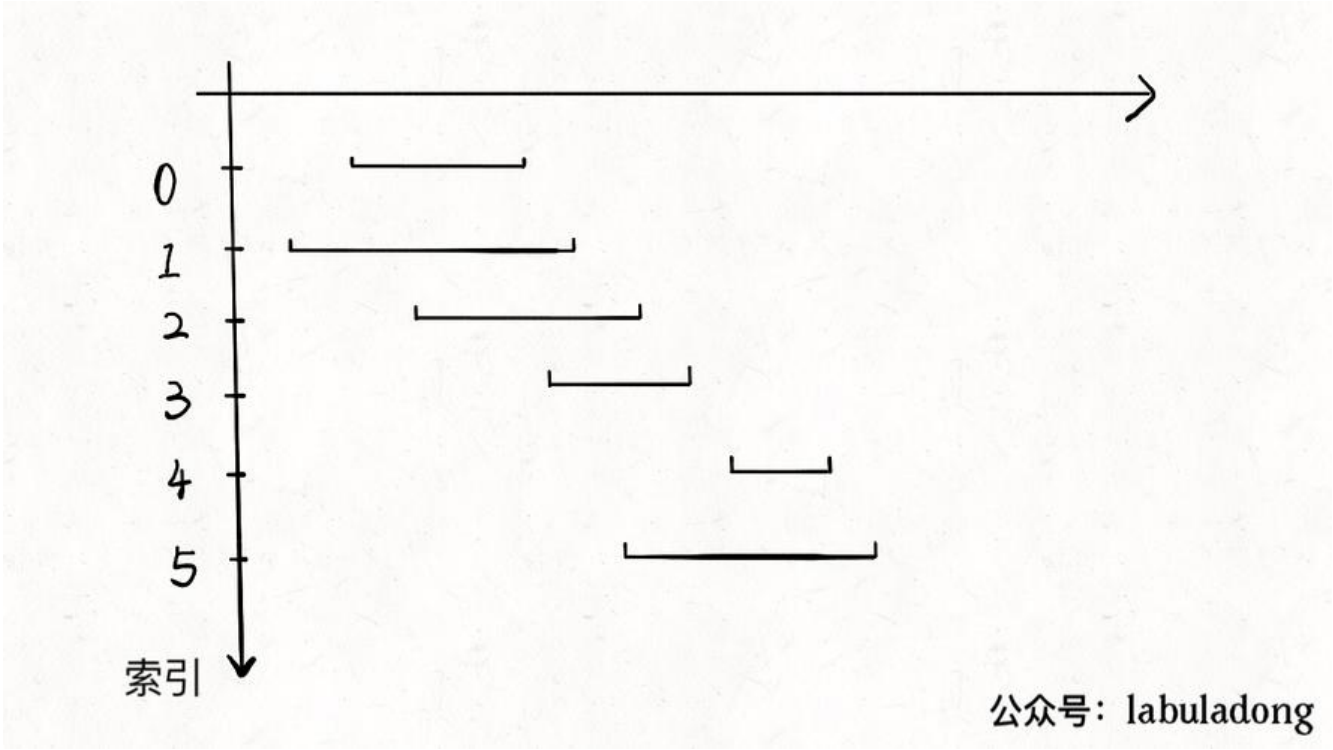
好了，说了这么多，那针对该问题正确的贪心策略到底是哪个？

其实正确的思路也比较简单，可以分成下面三步：

1. 从区间集合中选择一个区间x，这个x是所有区间中结束最早的（end最小）。
2. 把所有与x区间相交的区间从区间集合中删除掉。
3. 重复1和2，直到区间集合为空。之前选出的那些x的集合就是最大的不想交子集。

这个思路实现成算法的话，可以按照每个区间的end数值进行升序排序，因为这样处理以后实现步骤和步骤2就会容易很多。

我们通过下面这个动图来辅助理解其整个过程。



由于我们在计数之前进行了排序，所以所有与x相交的区间必然会和x的end相交；如果一个区间不想与的end相交，它的start必须要大于或者等于x的end。

具体实现的代码如下：

```
public int eraseOverlapIntervals(int[][] intervals) {
    if (intervals.length == 0) {
      return 0;
    }
    Arrays.sort(
        intervals,
```

```
        new Comparator<int[]>() {
          @Override
          public int compare(int[] o1, int[] o2) {
            return o1[1] - o2[1];
          }
        });
    //排序后的第一个必然可用
    int count = 1;
    int x_end = intervals[0][1];
    for (int[] interval : intervals) {
      if (interval[0] >= x_end) {
        count++;
        x_end = interval[1];
      }
    }
    return count;
}
```

应用

如果学会了上面的区间调度问题的话，leetCode上边有两个题目，我们便都可以拿下了。

452. 用最少数量的箭引爆气球

难度 中等 196 收藏 分享 切换为英文 接收动态 反馈

在二维空间中有许多球形的气球。对于每个气球，提供的输入是水平方向上，气球直径的开始和结束坐标。由于它是水平的，所以纵坐标并不重要，因此只要知道开始和结束的横坐标就足够了。开始坐标总是小于结束坐标。

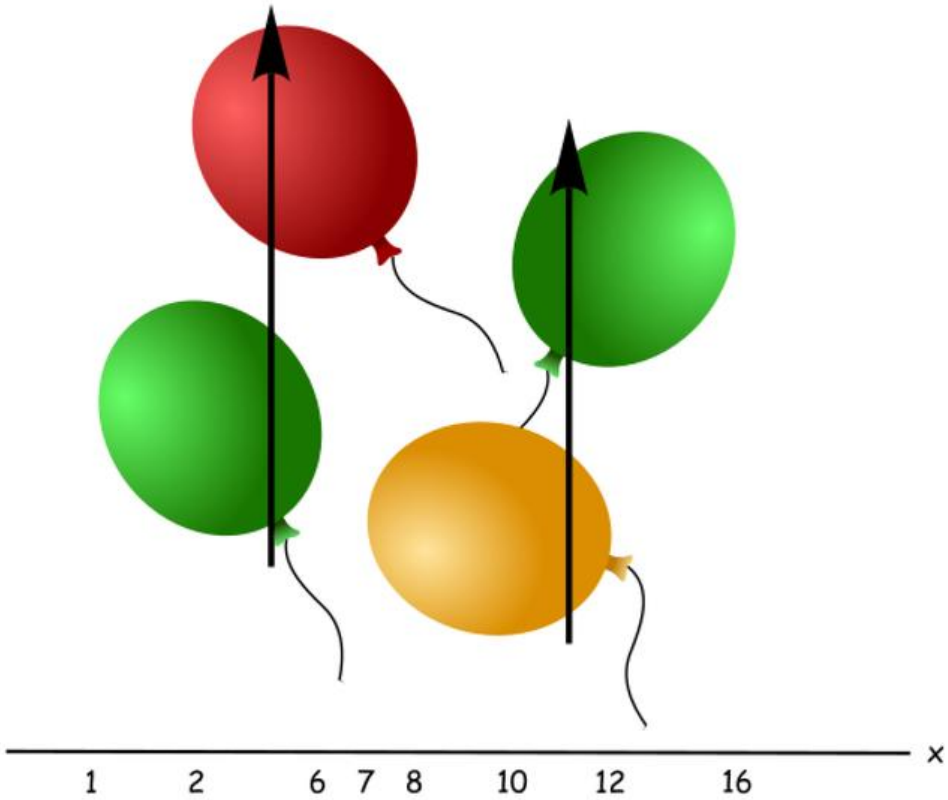
一支弓箭可以沿着 x 轴从不同点完全垂直地射出。在坐标 x 处射出一支箭，若有一个气球的直径的开始和结束坐标为 x_{start}，x_{end}，且满足 $x_{start} \leqslant x \leqslant x_{end}$，则该气球会被引爆。可以射出的弓箭的数量没有限制。 弓箭一旦被射出之后，可以无限地前进。我们想找到使得所有气球全部被引爆，所需的弓箭的最小数量。

给你一个数组 points ，其中 points [i] = [x_{start},x_{end}] ，返回引爆所有气球所必须射出的最小弓箭数。

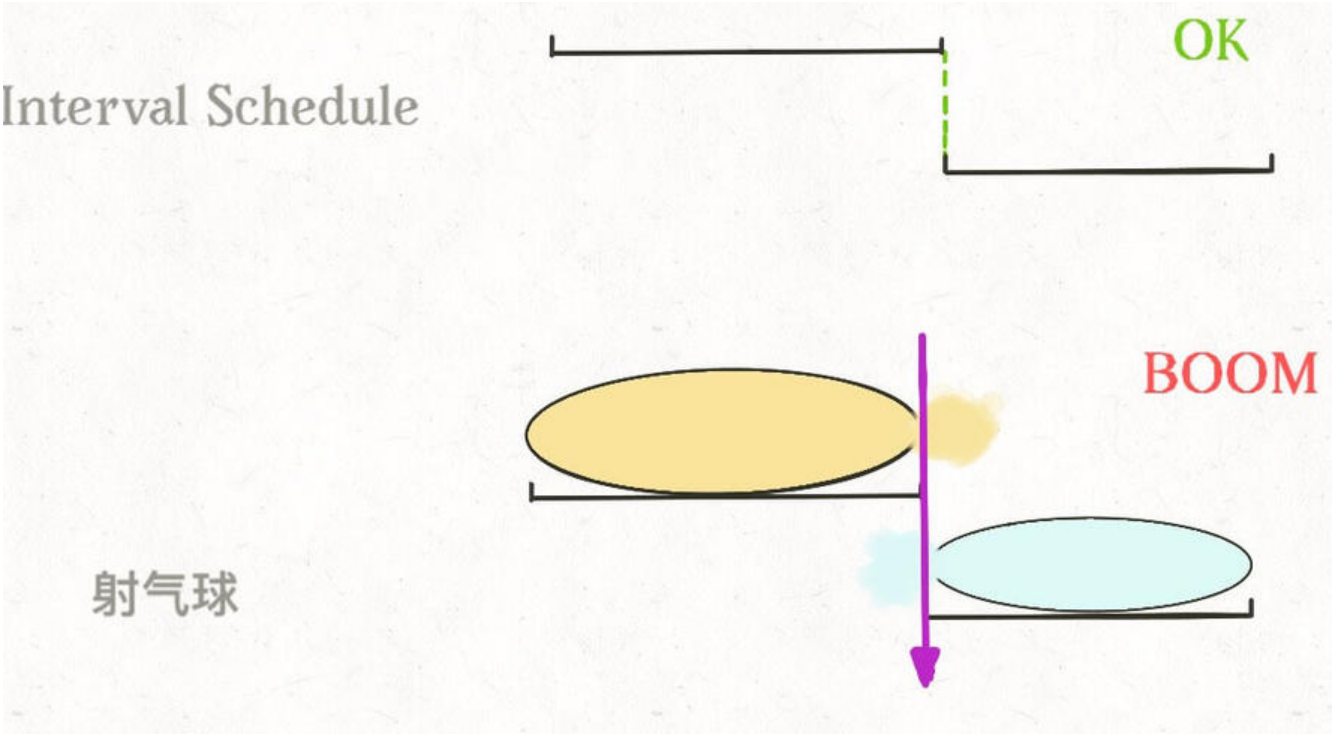
示例 1：

输入： points = [[10,16],[2,8],[1,6],[7,12]]
输出： 2
解释： 对于该样例，x = 6 可以射爆 [2,8],[1,6] 两个气球，以及 x = 11 射爆另外两个气球

这个问题大眼一看好像和我们之前讲的那个区间调度问题毫不相关，但仔细分析一下，好像是一模一的问题，如果最多有 n 个不重叠的区间，那么就至少需要 n 个箭头穿透所有区间。



因而问题也就转化成了，寻找不重叠区间的个数，但我们要注意的一点是，在 intervalSchedule 算中，如果两个区间的边界触碰，不算重叠；而按照这道题目的描述，箭头如果碰到气球的边界气球也爆炸，所以说相当于区间的边界触碰也算重叠。



代码实现如下：

```
public int findMinArrowShots(int[][] points) {
    if (points.length <= 0) {
```

原文链接：贪心算法原理及其应用

```
    return 0;
  }
  // 在排序的过程中要考虑溢出情况的发生
  Arrays.sort(points, (a, b) -> Integer.compare(a[1], b[1]));
  int count = 1;
  int x_end = points[0][1];
  for (int[] point : points) {
    if (point[0] > x_end) {
      count++;
      x_end = point[1];
    }
  }
  return count;
}
```

总结

本文主要结合一个例子，讲了贪心算法的使用方式。

贪心算法实现起来容易，但难在证明。因而文中提供了两个小窍门辅助判断是否使用贪心算法：

1. 在使用考虑贪心算法之前，先考虑使用动态规划（考虑状态压缩）解决该问题，如果问题依然超时则考虑使用贪心算法。

2. 在确定贪心策略之前，先用一些特殊的例子验证贪心策略的正确性。对于正确的贪心策略，为了保算法的绝对正确，要通过数学归纳法进行验证。