入选理由

1. 一道难度简单的堆题目，大家来练练手

2. 这道题能很好地体现堆的**动态**求极值特点。

标签

- 堆

难度

- 简单

题目地址（1046.最后一块石头的重量）

https://leetcode-cn.com/problems/last-stone-weight/

题目描述

```
有一堆石头，每块石头的重量都是正整数。

每一回合，从中选出两块 最重的 石头，然后将它们一起粉碎。假设石头的重量分别为 x 和 y，且 x <= y。那么粉碎的可能结果如下：

如果 x == y，那么两块石头都会被完全粉碎；
如果 x != y，那么重量为 x 的石头将会完全粉碎，而重量为 y 的石头新重量为 y-x。
最后，最多只会剩下一块石头。返回此石头的重量。如果没有石头剩下，就返回 0。



示例：

输入：[2,7,4,1,8,1]
输出：1
解释：
先选出 7 和 8，得到 1，所以数组转换为 [2,4,1,1,1]，
再选出 2 和 4，得到 2，所以数组转换为 [2,1,1,1]，
接着是 2 和 1，得到 1，所以数组转换为 [1,1,1]，
最后选出 1 和 1，得到 0，最终数组转换为 [1]，这就是最后剩下那块石头的重量。
```

```
提示：

1 <= stones.length <= 30
1 <= stones[i] <= 1000
```

前置知识

- 堆

思路

读完题目可以发现，核心是每次取石头堆中最重的两个石头进行粉碎，如果直接实现排序后，粉碎后的石头重新插入数组，则需要再次排序。因此，不难想到插入删除复杂度都为 logN 的堆。主要步骤如下：

- 把石头构大顶堆（Java 默认小顶堆，需重写比较器）
- 每次取出前两个（前提 size≥2）进行判断，若重量相同，则抛弃，否则将重量差入堆。
- 当堆中元素不足 2 个则停止并返回相应结果。

代码

代码支持：Java，Python3，JS

Java Code：

```java
class Solution {

    public int lastStoneWeight(int[] stones) {

        PriorityQueue<Integer> pq = new PriorityQueue<>((x, y) -> y - x);

        for (int i : stones)
            pq.offer(i);

        while (pq.size() >= 2) {

            int x = pq.poll();
            int y = pq.poll();

            if (x > y)
                pq.offer(x - y);
        }

        return pq.size() == 1 ? pq.peek() : 0;
    }
}
```

Python3 Code:

```python
class Solution:
    def lastStoneWeight(self, stones: List[int]) -> int:
        h = [-stone for stone in stones]
        heapq.heapify(h)

        while len(h) > 1:
            a, b = heapq.heappop(h), heapq.heappop(h)
            if a != b:
                heapq.heappush(h, a - b)
        return -h[0] if h else 0
```

JS Code:

```javascript
var lastStoneWeight = function (stones) {
  const pq = new MaxPriorityQueue();
  for (const stone of stones) {
    pq.enqueue("x", stone);
  }

  while (pq.size() > 1) {
    const a = pq.dequeue()["priority"];
    const b = pq.dequeue()["priority"];
    if (a > b) {
      pq.enqueue("x", a - b);
    }
  }
  return pq.isEmpty() ? 0 : pq.dequeue()["priority"];
};
```

复杂度分析

令石头个数为 N

- 时间复杂度：$O(N\log N)$

- 空间复杂度：$O(N)$

上一页 下一页