3rd: Greedy

手写题

Distributing Candy Game

(1) Modeling

- 问题理解：在游戏中，教师和每个孩子都有两个整数，分别写在左右手上。游戏的目标是通过调整队伍的排列顺序，使得获得最多糖果的孩子所得到的糖果数量尽可能减少。每个孩子的糖果数量依赖于在他们面前的所有人的左手数字的乘积，除以他们自己右手数字的值，向下取整。
- 根据题目的要求，小朋友1和小朋友2位置交换的必要条件是：小朋友2在小朋友1前面时所获得的糖果数应更小。为此，考虑只有两个小朋友的情况，分析两种排列的糖果数。用a表示左手数字，b表示右手数字，$a0$、$b0$分别表示老师左右手的数字。如果小朋友1在前，他们的糖果数为$\frac{a_0}{b_0}$和 $\frac{a0 \cdot a1}{b2}$；如果小朋友2在前，他们的糖果数为 $\frac{a0}{b2}$ 和 $\frac{a0 \cdot a2}{b1}$。可以约去公式中的 $a0$，将问题转化为比较 $\max\left(\frac{1}{b1}, \frac{a1}{b2}\right)$ 和 $\max\left(\frac{1}{b2}, \frac{a2}{b1}\right)$。由 $a > 0$ 且 a 为整数可得：

  1. $\frac{a1}{b2} \geq \frac{1}{b2}$
  2. $\frac{1}{b1} \leq \frac{a2}{b1}$

  假设 $\frac{1}{b1}$ 是最大值，则有 $\frac{1}{b1} \geq \frac{a2}{b1}$，只在两边相等时成立，此时进一步得出 $\frac{1}{b2} \leq \frac{a2}{b1}$。因此，两种排列的最大值相等，无需交换。同理，如果 $\frac{1}{b2}$是最大值，也无需交换。
  最终，只需比较 $\frac{a1}{b2}$ 和 $\frac{a2}{b1}$ 的大小。当 $\frac{a1}{b2} > \frac{a2}{b1}$ 时，需进行交换。由于这可变形为 $a1 \cdot b1 > a2 \cdot b2$。相邻两个小朋友的顺序既不会影响他们前面人的糖果数，也不会影响他们后面人的糖果数，只影响他们两个人的糖果数量。因此，为了优化排列，应按 $a \cdot b$ 的值从小到大进行排序

(2) Algorithm description

- 贪心策略：按照 $a \cdot b$ 的值从小到大排列，即可让获得最多糖果数的小朋友所获得的糖果数量尽可能少。遍历可得到获得最多糖果数的小朋友所获取的糖果数及小朋友的位置。

Minimum Number of Straights

(3) Time complexity

- 排序复杂度：由于算法的主要步骤是对小朋友的左右手数字进行排序，使用快速排序或归并排序的平均时间复杂度为 $O(n \log n)$，其中 n 是小朋友的数量。
- 糖果数量计算复杂度：遍历一次小朋友数组来计算每个小朋友的糖果数量，时间复杂度为 $O(n)$。
  综合考虑，整体时间复杂度为：
  $O(n \log n) + O(n) = O(n \log n)$

(4) Space complexity

- 存储空间：算法中使用的额外空间主要用于存储小朋友的左右手数字以及排序后的结果。在最坏情况下，排序算法可能会使用 $O(n)$ 的额外空间。

Array Partition

(1) Modeling

我们先对数组进行排序。由于每两个数，我们只能选择当前小的一个进行累加。因此我们猜想应该从第一个位置进行选择，然后隔一步选择下一个数。这样形成的序列的求和值最大。

(2) Algorithm description

【自然语言描述】

```
class Solution {
    public int arrayPairSum(int[] nums) {
        int n = nums.length;
        Arrays.sort(nums);
        int ans = 0;
        for (int i = 0; i < n; i += 2) ans += nums[i];
        return ans;
    }
}
```

(3) Time complexity

$O(n\log n)$

(4) Space complexity

$O(\log n)$

Delete Number Game

(1) Modeling

- 题目要求的是删除特定位数的数字后，剩下的数字按照左右顺序构成的新数最小。反过来想，即只保留$n-k$位数字使得构成的新数最小。
- 贪心策略：从原来的数中取$n-k$位数字使得构成的数字最小。从左往右取数，因为左边的数字是新数的高位，要使高位尽可能地小，因此枚举第一位数字所有可能的位置（因为要保证它后面还有足够的数字可以供后续取数），取最小且最靠前的一个数字（即如果最小的数字有多个，取最靠前的一个），这样就保证了第一位数字的最优性。第二位数在第一位数之后所有可能的位置内枚举，同样取最小且最靠前的一个。之后的取数方法也同上，直到取完所有要取的数。

(2) Algorithm description

- 定义参数：设原整数的长度为 n，需要删除的位数为 k，则需要保留的数字个数为 $m = n - k$。
- 初始化：从原整数的第一个数字开始，创建一个空的结果字符串，用于保存最终的最小数字。
- 从左到右选择数字：
  - 选择第一位数字：设索引从0开始，遍历从索引 0 到 $n - m$ 的所有数字。这是因为选择第一位数字时，后面必须留有至少 $m - 1$ 位数字供选择。对于这段范围内的每一个数字，找出其中的最小数字。如果有多个最小数字，选择最靠前的那个数字，并将其添加到结果字符串中。更新选择范围：选择完第一位数字后，更新索引范围为从上一个选定数字的下一个位置到 $n - m + 1$，以保证后面还有足够的数字可以选择。
  - 选择后续位数字：对于后续每一位数字，继续在更新后的范围内进行选择，具体范围为从上一个选定数字的下一个位置到 $n - m + i$（其中 i 是已经选择的数字位数）。在这个范围内，寻找最小数字，并同样选择最靠前的那个数字添加到结果字符串中。
- 最终结果：重复上述步骤，直到选择完所有的 m 个数字。最终结果字符串即为通过删除 k 个数字所能构成的最小整数。结果可以包含前导零。

(3) Time complexity

共需要进行m次遍历，每次遍历的时间复杂度最坏情况下为$O(k)$，因此总的时间复杂度为$O(mk) = O(k(n-k)) = O(nk)$。

(4) Space complexity

主要的空间用于保存最终选择的m个数字，因此空间复杂度为$O(m)$。

Ex. Container Balancing Operations

(1) Modeling

容易判断出当容器的数量不能整除所有容器的物品之和时，则不能物品数量不能平衡，反之都可以，目标市使每个容器物品数量相等。

题目要求每次只能将一个物品从一个容器移动到相邻的容器中。为了使所有容器的物品数量达到 平均个数Avg（即每个容器物品相等），我们需要考虑每个容器的最大不平衡量。我们定义 $Max = max(container[i])$，即所有容器中最多的物品数量。为了让最多物品的容器减少到 Avg，最少需要 $Max - Avg$ 次移动。

为了实现物品的平衡，考虑容器之间的累积不平衡。设sum为前i 个容器物品总和，若 $sum - i * Avg > 0$，则表示前 i 个容器的物品多，需要向后面的容器移动；若 $sum - i * Avg < 0$，表示前 i个容器的物品少，后面的容器需要将物品移动过来。

所需的最小操作次数等于每台洗衣机的累积不平衡量和移动到目标 Avg 的最大值。

(2) Algorithm description

1. 条件检查：首先判断所有容器的物品总数是否可以被容器数量整除，若不能，返回1。
2. 定义目标值：每台容器的目标物品数量为 $Avg = sum(container[i])/n$。
3. 最少操作数：
    - 每台容器的最大移动次数为 $Max - Avg$，其中Max 是最多物品的容器数量。
    - 通过累积不平衡量 running_diff，记录前 i 个容器的累积差值，决定是否需要向后或向前移动物品。
4. 最终最少操作数：取每个容器的累积差值 |running_diff| 和最大差值 $Max - Avg$的最大值作为最少操作次数。

(3) Time complexity

时间复杂度:$O(n)$

(4) Space complexity

空间复杂度: $O(1)$

OJ题

Sorrowful Cows

(1) Modeling

- 要求总长度最短，就必须使被浪费的长度尽量小，即cow与cow之间的距离。
- 注意到，在第一个cow和最后一个cow之间，一共会有$M-1$个距离被浪费。
- 因此，对每两个cow之间的距离进行排序，然后用第一个cow到最后一个cow之间的距离依次减去最大的$M-1$个距离，即是答案。

(2) Algorithm description

1. 计算每两个cow之间被浪费的距离$cow[i]-cow[i-1]-1$，并按降序排序。
2. 计算第一个cow到最后一个cow之间的距离$cow[C]-cow[1]+1$。
3. 第2步得到的距离依次减去前$M-1$个第1步得到的距离，即是答案。

(3) Time complexity

第1、2、3步时间复杂度分别是$O(C),O(1),O(M)$，因此整体复杂度为$O(C+M)$。

(4) Space complexity

仅需一个辅助数组记录每两个cow之间的距离，因此空间复杂度为$O(C)$。

Sick Cows

chh

(1) Modeling
- 目标是让最开始感染的牛尽量少，这个问题可以转化为让传染的天数尽量多。
- 如果一只牛在中间，它每天都会传染两边的牛，因此对于连续感染的
x头牛，最多需要传染 $\lfloor \frac{x-1}{2} \rfloor$天。
- 而整群牛最多传染天数是所有连续的牛的最多传染天数中的最少天数。

(2) Algorithm description
- 求出传染天数
m。
- 每只牛在
m天后会感染两边各m头牛，算上它本身总共$2m+1$头牛。
- 为了最初感染的牛最少，每
$2m+1$头牛将中间的那头牛当最初感染的牛。
- 因此，连续感染的

x头牛，最初感染的至少会有$\lceil \frac{x}{2m+1} \rceil$头。
- 边界：连续感染的
x头牛能传染$x-1$天。

(3) Time complexity

$O(N)$

分析：只需在输入的同时扫一遍每头牛的情况即可

(4) Space complexity

$O(N)$

分析：使用数组存储每头牛的感染情况

Minimum Number of Straights

1. Modeling

我们可以将这道题建模为一个关于如何以最小次数使用“顺子”方式来打出所有牌堆中的牌的问题。设有 (n) 堆牌，每堆牌的数量为 (a_i)(1 ≤i ≤ n)。我们的目标是找出最少的手数来将这些牌打出。

2. Algorithm Description

1. 遍历牌堆：
    - 首先，记录每一堆牌的数量 (a_i)。
    - 从左到右遍历所有牌堆，贪心地选择可以组成顺子的部分。
2. 构成顺子：
    - 对于当前牌堆 (i)，如果存在牌 ((a[i] > 0))，尝试与后续的牌堆形成顺子，直到无法再构成新的顺子为止。

- 每次操作时，如果当前堆 (i) 中的牌大于前一堆 (i-1)，则可以用当前堆中的牌填补前一堆的空缺，增加操作次数。

3. 记录操作次数：

   使用一个变量 `ans` 来记录需要的最小操作次数。最后将第一堆的牌数量与 `ans` 相加，得到总的操作次数。

3. Time Complexity

时间复杂度：每次操作都需要遍历所有的牌堆，因此时间复杂度为 (O(n))。在最坏情况下，每堆牌的数量可以高达 100,000，总的操作次数不会超过 (O(n))。

4. Space Complexity

空间复杂度：使用 (O(n)) 的空间来存储每个牌堆的数量。这是因为我们需要一个数组来存储 (n) 堆牌的数量。

Extra Problem

1. **Modeling**

   在这个问题中，我们希望卡车在到达终点的过程中尽量少加油。在保证到达终点的前提下，优先选择能够让卡车行驶最远的加油站。贪心算法的核心思想是每当卡车需要加油时，选择当前能到达的加油站中燃料最多的一个，以尽可能减少加油次数。

2. **Algorithm Description**

   a. 将所有的加油站按距离终点的远近排序，这样我们可以按照顺序检查卡车行驶到每个加油站的情况。

   b. 初始化一个**最大堆**（优先队列），用来保存卡车当前可达的加油站燃料量。在需要加油时，我们总是从最大堆中取出当前最大可用燃料的加油站进行加油。

   c. 模拟前进过程：

      - 每次判断是否能够到达下一个加油站或终点，如果燃料不足，就从最大堆中取出最大燃料的加油站进行加油。
      - 每到达一个加油站，就将这个加油站的燃料量放入最大堆中，以备后续选择。

   d. 终止条件：

      - 如果最大堆中没有燃料可用，但当前燃料不足以继续前行，则无法到达终点，输出 −1。
      - 如果到达终点，输出最少的加油次数。

3. **Time Complexity**

    a. **排序加油站**：需要 $O(NlogN)$ 的时间，其中 N 是加油站的数量。

    b. **遍历和加油**：遍历加油站并使用堆进行加油时，每次加油操作的复杂度为 $O(logN)$，总的复杂度为 $O(NlogN)$ 。

    所以总的时间复杂度为 $O(NlogN)$ 。

4. **Space Complexity**

    - **输入存储**：需要 $O(N)$ 的空间存储每个加油站的位置和燃料量。
    - **最大堆存储**：在最坏情况下，需要存储所有可达的加油站燃料量，因此最大堆的空间复杂度为 $O(N)$ 。

    总的空间复杂度为 $O(N)$。