石子合并

显然对于任意满足 $1 \le i < j \le n$ 的一对 i,j，会在第 i 堆和第 j 堆所属大堆合并时，对答案产生$A_i \times A_j$的贡献，也即无论合并顺序如何，代价和都不变。即 x=read(),ans=ans+(s*x),s=s+x; 注意答案会爆 long long。

寿司

算法一

爆搜，枚举所有走法。可以获得 0 到 20 分不等。

算法二

注意到每种状态实际上是可以压缩的。所以搜索的时候对于较小的数据把状态记录下来，用最小表示法去个重，对于稍大的数据可以用 $IDA*$搜索，对每个状态进行估价来进行加速。

可以获得 20 到 40 分不等。

算法三

一般环上的问题一般都要把序列复制一遍，然后转成了序列上的问题。这道题也不例外。经过观察后我们显然可以得到一个结论：对于最优解，一定有一个断点使得所有的交换都不经过这个点。

于是，我们就可以枚举断点，然后就转变为了一个序列了。我们的目的就变成了使一种颜色的寿司全部靠边。然后可以预处理出每个寿司移动到左边界需要多少步，右边界需要多少步，取 min 后加起来即可。

时间复杂度 $O(n^2)$，可以得到 40 分。

算法四

发现对于一个序列，肯定是左边一部分往左靠，右边一部分往右靠，于是可以每次二分出这个边界点。

时间复杂度 $O(n\log n)$，可以得到 80 到 100 分。

算法五

如果算法四你都想出来了，那么算法五应该顺理成章的出来了。当断点顺时针移动的时候，分界点显然不会逆时针移动。于是这玩意儿是有单调性的，弄一个单调指针扫一扫就可以了。

时间复杂度 $O(n)$，可以得到 100 分。

反击

来源：codeforces633 F

关键字：树形 DP

3.1　15%

直接上大暴力，枚举某个点是否处于最终方案中，每次判断合法性并更新答案。

时间复杂度：O（2^n）

3.2　15%

枚举两条链的起始端点，然后再暴力判断两条链是否相交，更新答案。

时间复杂度：O（n^5）

3.3　30%

考虑枚举 u 的子树的时候，两条链分别在 u 的子树中，和 u 的子树外的情况。u 的子树外 O（n）的 DP 一遍，两条链的答案组合即可得到全局答案。

时间复杂度：O（n^2）

3.4　100%

个人认为，这道题的树形 DP 还是很好想的，就是有很多种情况需要分类讨论，较为繁琐。

题意简单的来说就是一棵每个点都有权值的树，求两条不相交的路径的点权和的最大值。

首先一遍 dfs 一遍，对于每个点 u 求出 chain[u]，表示 u 到叶子结点的路径点权和的最大值，和 whole[u]，表示 u 的子树中一条路经的点权和的最大值。然后我们需要考虑如何讨论两条不相交的路径的最大权值和。

显然，我们一定可以找到一个点 x，和 x 的一个儿子结点 v，使得两条路径，一条处于 v 及其子树中，一条处于剩下的一大块中。相当于是，x 和 v 相连的这条边“隔开”了这两条路经。我们可以考虑对于每个点 x 都去枚举 x 的一个儿子结点 v，进行讨论。

题目转换为：求 v 的子树中一条路经的最优值和剩下部分的最优值。现在我们考虑，如何分别得出两个部分的最优路径和。首先 v 的子树中的最优值我们已经预处理出来了，就是 whole[v]，所以我们的任务就是求整棵树除了 v 的子树之外的剩余部分的最优值。

通过简单地作图即可发现，在剩余部分的这条链的情况只有 2 种。要么这条链由 x 的除了 v 之外的两个儿子结点 v1，v2 的 chain[v1]，chain[v2]加上 x 的点权组成，这种情况是链完全处于 x 的子树中；还有一种情况就是由 x 的除了 v 以外的一个儿子结点 v1 的 chain[v1] 加上 x 的点权再加上 x 往上走的链的最大权值和，我们不妨设这个往上走的最大路径权值和为 up[x]（不包括 x 结点），那么我们需要在对 x 进行讨论之前就计算出 up[x]，所以我们就必须保证讨论 x 之前其深度比 x 小的已经全部讨论完毕。那么我们就可以用一个队列来保证遍历的顺序。而在每次讨论完 x 之后，可以更新 x 的每个儿子结点的 up 值，并把每个儿子结点加入队列。而对于每个 up[vi]（vi 是 x 的儿子结点）的算法也是需要分类讨论：要么是 x 的 up 值加上 x 的权值（即“继承下来”），或者是从 x 的除了 vi 之外的别的儿子结点的 chain 转移过来，二者取一个 max 即可。具体实现的话略显复杂，细节比较多。如果不是很理解的，可以画画图，并看一下我的代码的具体实现。