解题报告

题目名称	皇后游戏	旅行计划	维修机器人
输入文件名	game.in	plan.in	repair.in
输出文件名	game.out	plan.out	repair.out
每个测试点时限	1 秒	2 秒	1 秒
内存限制	512MB	512MB	512MB
测试点数目	20	10	20
每个测试点分值	5	10	5
是否有部分分	否	否	否
题目类型	传统	传统	传统

皇后游戏

- **算法1**
  暴力枚举顺序,时间复杂度$O(n \times n!)$或$O(n!)$可以得到20分.做法很简单,这里就不讲了.

注意到$\max\{c_i\}$ $(1 \le i \le n)$,一定是c_n,这个可以从c_i的定义中很容易发现c_i是单调递增的.

- **算法2**
  考虑状态压缩的动态规划,记$dp[S][i]$表示已经安排好的数对组成的集合为S,最后一个数对为第i对,c_i的最小值为$dp[S][i]$.时间复杂度$O(2^n n^2)$,可以得到40分.

- **算法3**
  其实,从样例中我们可以发现,满足条件的排列方案并不唯一,因此对于前40分数据,只需要每次随机一个排列,多次随机并卡时取最优解,也可以通过.

- **算法4**
  当$a_i = b_i$时,任意一种排列方式都是最优解,这里不给出证明.

- **算法5**
  当$b_i = a_i + 1$时,按照a_i递增的方式排列可以得到最优解,这里不给出证明.

至此,结合上面几部分算法,可以得到60分.

- **算法6(满分算法)**
  考虑$n = 2$时的情况:假定两个数对分别为(a, b),(c, d),则当且仅当$\min(a, d) \le \min(b, c)$时,把$(a, b)$放在前面更优,否则把$(c, d)$放在前面更优,这里不给出证明.(具体证明可以参考2014 年北京市高考理科数学第20题第2问,或后文.)

类似地,我们考虑交换第i个数对与第$i+1$个数对,注意到如果我们可以让c_{i+1}变得更小,则这是对后面有利的.

记$sum = \sum_{j=1}^{i-1} a_j$.我们考虑初值<交换后的值则有:

$$\max\{\max\{c_{i-1}, sum + a_i\} + b_i, sum + a_i + a_{i+1}\} + b_{i+1}$$
$$<$$
$$\max\{\max\{c_{i-1}, sum + a_{i+1}\} + b_{i+1}, sum + a_{i+1} + a_i\} + b_i$$

将所有值都放进max里,上式变为:

$$\max\{c_{i-1} + b_i + b_{i+1}, sum + a_i + b_i + b_{i+1}, sum + a_i + a_{i+1} + b_{i+1}\}$$
$$<$$
$$\max\{c_{i-1} + b_{i+1} + b_i, sum + a_{i+1} + b_{i+1} + b_i, sum + a_{i+1} + a_i + b_i\}$$

注意到第一项的值是相同的,因此我们只需要:

$$\begin{gathered}\max\{sum+a_i+b_i+b_{i+1},sum+a_i+a_{i+1}+b_{i+1}\}\\<\\\max\{sum+a_{i+1}+b_{i+1}+b_i,sum+a_{i+1}+a_i+b_i\}\end{gathered}$$

约去sum以后得到:

$$\begin{gathered}\max\{a_i+b_i+b_{i+1},a_i+a_{i+1}+b_{i+1}\}\\<\\\max\{a_{i+1}+b_{i+1}+b_i,a_{i+1}+a_i+b_i\}\end{gathered}$$

令$d=a_i+b_i+a_{i+1}+b_{i+1}$,则上式等价于:

$$\max\{d-a_{i+1},d-b_i\}<\max\{d-a_i,d-b_{i+1}\}$$

这也就是说:

$$\max\{-a_{i+1},-b_i\}<\max\{-a_i,-b_{i+1}\}$$

即:(有点绕,max变成了min,写几个值,自行脑补一下)

$$\begin{gathered}-\min\{a_{i+1},b_i\}<-\min\{a_i,b_{i+1}\}\\\min\{a_i,b_{i+1}\}<\min\{a_{i+1},b_i\}\end{gathered}$$

于是我们定义第i个数对比第j个数对小,当且仅当:

$$\min\{a_i,b_j\}<\min\{a_j,b_i\}$$

按照这个规则排序以后扫描一遍即可得到最优解,时间复杂度$O(n\log n)$.

旅行计划

算法一：使用类似 *Floyd* 的算法预来处理任意两点间的答案，记 $ans[i][j]$表示从点 i 走到点 j 最小的最大距离值。每次用 $\max(ans[i][k],ans[k][j])$来更新 $ans[i][j]$。预处理复杂度 $O(n^3)$，单次询问复杂度 $O(1)$。期望得分 20 分。

算法二：注意到测试点 3 只有一次询问，测试点 1 和 2 的点数及询问数都很小。因此可以二分答案以后从 S 使用 BFS 来扩展看能否走到 T，然后调整二分边界。期望得分 30 分。

算法三：易知我们的行走过程一定要在原图的最小生成树上行走，而 n 个点的图共有约 $n^2/2$ 条无向边，求出最小生成树，问题转换成 m 次询问树上两点间最大边权。预处理复杂度 $O(n^2\log n)$或 $O(n^2)$，单次询问复杂度 $O(n)$。期望得分 40 分。

算法四：注意到测试点 5 和 6 所有的点都满足 $x_i = 1$，因此所有的点将构成一条链。每次的行走过程一定是从一个点走到它上面的点或者它下面的点，将所有的点按照纵坐标排序，这样每次询问就变成求给定区间$[L,R]$的最小值，使用 ST 表或线段树等数据结构进行处理。期望得分 20 分。结合算法三。期望得分 60 分。

算法五：我们发现算法的瓶颈主要出现在预处理最小生成树上。对于平面点曼哈顿距离最小生成树有一些性质可以利用。对于一个点 i，我们以它为原点建立平面直角坐标系，并画出直线 $y = x$ 和 $y = –x$。这样整个平面就被坐标轴和这两条直线分成了 8 个部分，可以证明的是点 i 只需要向每个部分离它最近的点连边然后求最小生成树即可。这样边数就变成了 $8n$ 条，可以快速求出最小生成树了。接下来我们考虑如何构造这 $8n$ 条边，如果对每个点都去寻找这 8 个部分离它最近的点，建边复杂度将再次退化为 $O(n^2)$。一个可行的做法是将所有的点按照横坐标为第一关键字纵坐标为第二关键字排序，然后使用线段树或树状数组或平衡树等数据结构维护点之间的关系即可做到 $O(n\log n)$的预处理复杂度。接下来我们考虑每次询问，设树的高度为 h，则单次询问的复杂度就为 $O(h)$。对于前 4 个测试点 n 都较小，对于第 5 和第 6 个测试点可以通过算法四解决，对于第 7 和第 8 个测试点由于数据随机生成，这样树的高度不会特别高。因此结合算法四以后期望得分 80 分。

算法六：使用算法五预处理出最小生成树，我们考虑优化算法五的询问部分。每次询问树上两点间的最大边权，使用树链剖分，预处理复杂度 $O(n\log n)$，单次询问复杂度 $O(\log^2 n)$。期望得分 80~90 分。使用动态树，预处理复杂度 $O(n\log n)$，单次询问复杂度 $O(\log n)$。期望得分 80~90 分。以上两种算法虽然可以高效解决树上的很多询问问题，但由于常数过大等原因无法通过所有测试数据。我们考虑到树的形态固定，边权固定，因此我们可以使用倍增方式预处理点 i 往上走 2^j 条边的最大边权，这样预处理复杂度 $O(n\log n)$，单次询问复杂度 $O(\log n)$，常数很小。期望得分 100 分。

维修机器人

首先仍然从最简单的情况开始分析，对于原始数列中三个相邻的数 a、b、c，如果确定 a、c 不进行调整，而可以对 b 进行调整，那怎样调整最优。

可以很直观地想到，只有当 $a > b$ 且 $c > b$ 时，才可能对 b 进行调整，否则将 b 增大不可能减小差值导致的代价，另一方面还有调整本身的代价，那么不妨设 $b < c \le a$。

另外一个显然的结论是，b 调整后不会超过 c，否则的话将 b 的调整量减小 1，调整的代价减小，差值的代价不会增大，整个方案会更优。

所以仅当 $b < a$ 且 $b < c$ 时需要进行调整，设调整后的数为 B，那么 B 的范围为［b, $\min(a,c)$］（将不调整也算入）。调整后的总代价为 $cost(B)=(B-b)2+(a-B)\times C+(c-B)\times C$，这是一个二次函数，$B$ 有确定的取值范围，要对其进行最优化是很容易的。所以，如果一个数两侧的数都确定不进行调整，这个数本身的调整量是可以 $O(1)$计算得出的。

但事实上是存在两个相邻的数都需要调整的情况的，例如数列为 4、1、2、4，倍率为 100，最优的调整方案显然是将中间两个数均修改为 4，当然，也存在连续更多的数都需要调整的情况，不过我们仍然从最简的情况入手：即连续两个数需要调整的情况。

设连续的 4 个数字为 a、b、c、d，其中 a、d 不做调整，b、c 要做调整，b 调整为 B、c 调整为 $C(b < B$、$c < C)$，如果 B、C 不等，不妨设 $B < C$，那么重新将 c 调整为 C-1，此时调整代价减少，差值代价不增，因此总代价变优，故最优的调整方案一定是有 $B = C$ 的。用同样的方法，容易证明如下重要性质：

在最优的调整方案中，如果有一段连续的数字均需要调整，则一定是调整成为相同的数字。

再仿照前文对一个数字做调整时做的分析，不难推出：在最优的调整方案中，如果［i,j］段需要做调整，则必有 $\mathrm{RMQ}(i,j) < \min(a[i-1],a[j+1])$，$\mathrm{RMQ}(i,j)$表示数列中 i 到 j 这一段中的最大值，且具体的调整量同样可以 $O(1)$算出。

注意到在前面的讨论中都没有处理处于数列边界的情况，这是需要特殊处理的，这属于算法的枝节，不在此细说。

现在我们已经有了很强大的性质，可以开始着手设计算法了。不难想到这样一个动态规划算法：$f[i]$表示结点 i 不做调整，调整完前 i 个数的最小代价。$f[i]$可以从 $f[j]$转移，当且仅当 $j+1=i$ 或 $\mathrm{RMQ}(j+1,i-1) < \min(a[i]，a[j])$。

如果直接将这个动态规划实现出来，时间复杂度为 $O(N2)$，无法承受。但直觉告诉我们，真正可以转移的位置是很少的，那么来看一下，具体有多少地方可以进行转移。

如果 $a[j] > a[i]$，显然这样的 j 是只有一个的，即 i 以前第一个比 $a[i]$大的数的位置，在此位置之后 $a[j] \le a[i]$，在此位置之前不可能满足上面制约的等式。故这样的转移只有 $O(N)$个。

如果 $a[j] \le a[i]$，这样的位置 j 是可能有很多个的，但是根据转移的条件，这些位置将来再也不会作为转移的位置了，所以这样的转移也只有 $O(N)$个。

所以实际上只有 $O(N)$次状态转移，关键是怎样快速找到这些位置。根据上面的描述，不难发现这实际上刚好符合单调队列的结构：从前往后扫描，维护单降队列，则每次插入一个元素时，从队首踢出的元素就是第二类转移的位置，踢

完后队首的元素就是第一类转移的位置，而且也不需要专门实现一个 RMQ，每次转移时从队列中踢出的前一个元素就是 RMQ 的值。

至此，本题已经解决，一共只会进行 $O(N)$次转移，每次转移的复杂度为 $O(1)$，故整个算法的复杂度为 $O(N)$。