无聊的序列

一道偏简单的性质构造题（希望大部分人都做出来了）。

我们考虑这样一个事实 GCD（a,a+1）= 1，为了让字典序最大，我们需要尽快把相邻的两项清掉，但是删去奇数还是偶数呢？当然是删奇数，因为偶数有共同的 GCD=2，奇数却不一定有共同的 GCD。这样我们就保证了最快使得非 1 元素出现，贪心使得字典序最大。

接下来怎么办呢？

拖

孤独的聚会

千万记住邻接矩阵和位运算是有用的。说实话肉眼可见的状态压缩，重点在于搜索时候的转换，每一次转换时构造一个子完全图（这个不会不知道吧-。-）

我们首先来说一下暴力，直接搜索处理顺序然后一步一步合并（虽然我并不知道你们会用什么方法来合并）然后与当前的最优答案进行判断，解决问题。能打出暴力还是很棒的。

接下来，说一下解决这道题目的关键之处（希望你们能够以后更多的发现题目性质，这样的性质才是解题的关键，就像 2016 的蚯蚓等等）：

1、在最后整张图变成完全图之前，一定有一个点和所有的其余点都有直接邻边连接。也就说明了，当所有的点，都和其余所有点都有直接连边时，状态为完全图。

2、当选定了操作某几个点之后，最后整张图形成的邻接矩阵与操作顺序无关。（你可以类比一下合并石子的过程，看能不能够理解，这个操作就像在收集石子一样）

这样，我们就能规划我们的构成。对于一个状态 2^n 我们存储是否操作这个点，操作为 1 不操作为 0。我们搜索这样一个状态，状态出来之后，开始合并，有上面的性质，我们从左向右选择点合并。

合并过程①：我们设找到第一个是 1 的位置为 k，然后扫描他所连接的点，接下来再枚举其余的点，也就是用 n^2 的操作来操作点 k，这样的总合并复杂度是 n^3 的。总复杂度为 O（$2^n * n^3$）

很容易发现上面做法的不足在于合并时候的枚举，我们可以对邻接矩阵进行状态压缩，这样可以思考一下如何判断两点有连边，如何操作合并节点的步骤。

拖

合并过程②：我们设找到第一个是 1 的位置为 k，然后扫描他所连接的点，我们利用位运算，可以直接用 k 和扫描到的点进行合并，此时的总复杂度为 O（$2^n * n^2$）

我们仍然浪费了部分时间，那部分时间就是枚举 1 点的时间，我们在搜索的时候，当这个点需要选，我们便操作一次，并且传递一个压缩之后的邻接矩阵序列。这样我们就会再少一层 n 的复杂度，并且能在中途直接判断全图是否变成完全图（剪枝）。

这样这道题的最优复杂度为 O（$2^n * n$）

合并过程③：我们枚举到了第 k 位，此时操作第 k 位，我们就扫描他所连接的点，利用位运算，直接合并，并且将新的邻接矩阵表（所有的邻接矩阵都已经被压缩了）利用 dfs 传递到下一级别去。

发现，字典序最小只需要从 1 开始做就行了。（根据推论 2）因为这道题有最优性剪枝，所以可以降低很多复杂度。（众所周知，剪枝的复杂度时玄学的）

难受的购物

题意简单明了，离线区间最小差。

给了 40%的 n^3 的部分分，还有 20%在大数据中的单调数列。(60 分妥妥的)

40%的暴力分随便做，排序都可以。枚举 l 到 r 的序列，排序（或者你直接用 set 找前驱后继）直接出答案。

30%的序列单调，这道题就变成了一道 RMQ（想通了吗），因为答案必定是两个大小连续的数的差值，那么把原序列转化为差值序列，再用 RMQ 算法求取区间最小值，

这 60%的分唾手可得呀。有些同学可能又难受了（#_#）

我的做法：

排序+分块+链表（表述可能有点不清楚，你们可以一起讨论，一边模拟一边理解）

链表的正确性：我们使用链表的时候，在保证顺序的情况下（数值大小顺序），往链表中插入一个数，这个数可以生成新的答案候选，并且如果生成答案，则是当前的最小答案。

☆**(重要的技巧)排序+分块的正确性**：我们对<u>左端点在同一个区块的询问</u>的右端点从大到小排序，当我们处理区块 i 的时候，右端点向左移动，左端点在区块内移动，这样可以保证复杂度在 O（n）。则一个区块处理复杂度为 O（n），区块个数为√n。所以总复杂度为 O（n√n），空间的正确性也容易证明出来。

☆**(我想出来的技巧，不知道其他地方可不可以用)维护链表+分块的正确性**：我们知道，链表的正确性建立在插入的基础上，当我们需要删除的时候，是无法维护正确答案的（也就是，更新的答案无法撤销，除非你能再套一层用来统计的 set），所以，并不能利用滑动窗口的思想来解决这个问题。我们现在的问题集中点在于，统计时不撤销也能够统计到正确的答案。

① 建立在分块基础上的移动，我们要明白，右端点只撤消，不扩增，意思就是，我们要解决右端点向左减少的问题。
② 需要注意的是，我们的排序方式导致了左端点的移动是在块内不规则的，这时候，解决方案来了。能学就学，学不会没关系，我菜菜的算法。
③ 我们设左端点在块 k，那么我们拿出块 k 的最右点设为点 r，以这个点为基础做链表（注意链表以数值大小为链），一直向右拓展，每经过右边的一个点是，按照我们**链表的正确性**有我们可以设经过了 rr，我们可以统计出 r 到 rr 上的答案，并记录它。
④ 这时候，我们再来考虑左端点，当右端点固定之后，我们统计了 r 到 *qes[i].r 的答案，那么按照我们**链表的正确性**，我们一个一个插入 r 到 qes[i].l 这时候再次统计一个答案，再取<u>这次统计的答案</u>与<u>记录在 qes[i].r 上的答案</u>的最优解。
⑤ 有小朋友疑惑了，假如 qes[i].r 比 r 小怎么办啊。暴力啊兄弟。(^_^)

*qes[i]代表第 i 个询问

然后这道题目的官方正解是线段树复杂度在 O（nlog²（n)）在这种数据大小下近

似于我的复杂度，然而我常数太大。

线段树的做法有很多种，我提供一种做法，原始的可以去 http://codeforces.com/blog/entry/50456 F 题

① 我们确定这样一个事实，序列中一个数对（保证前后顺序），要么为正序对要么为逆序对，答案只会出现在这样的序对中，那么，我们正着做一遍正序对，数列翻转做一遍正序对，那么我们就可以求出来答案了。并且，这样的答案是从左到右单调递增的（左边的更优）。

② 扫描（我们先考虑逆序对即 $i<j$ 时 $a[i] \geqslant a[j]$）：

a) 我们不考虑左端点，只考虑右端点，求出当前以 r 为右端点的所有左端点的答案。

b) 右端点向右移动一格，设当前的右端点位置为 r，数值为 x，那么根据上面我所说的，这时候，r 插入的时候，就可以对前面的所有 l 进行更新（此时我们的 l 即左端点是 r 左边的所有点，我们处理出所有的情况）。

c) 我们向左扫，找到第一个大于等于 x 的地方，我们设下标为 i 数值为 y，那么此时，y-x 假如可以更新答案，那么左端点为[1,i]的区间答案都可以更新。我们再次向左扫，找到一个点下标为 j 数值为 z。我们有当满足

   i. $x \leqslant z < y$

   ii. $z - x < y - z$ （想一想为什么）

   这个 j 点是一个可能更新答案的点，此时我们有不等式

$$x \leqslant z < (x+y)/2$$

   可以发现，这是一个二分逼近式，我们可以在 log（ai）的时间内逼近 z 这奠定了时间复杂度的基础。

d) 这样我们建立一棵对 a[i]离散化+排序之后的线段树，线段树的节点存这一个区间的数的最大位置，既然我们需要找到离我们 r 最近的大于 x 的值，我们需要的区间就是[*ls(x),ls((x+y)/2)]这样我们就可以通过 log（n）的时间找到需要的 i 点，然后利用另一棵区间线段树去更新答案。

e) 提取所有以 r 为右端点的询问，在区间线段树上查询答案。

③ 这时候，你会发现，正序是相同的。所以，只需要做一遍即可。