孤独(alone)

摸大三文鱼 出题组

0 写在题解之前

本题要求一个图的最大独立集，这是一个NP完全问题。

在造题过程中，笔者的数据测试状况是根据出题组成员们的电脑情况得出。题解中可能引用了一些来自维基百科的定理和算法（感谢计算机科学界的前辈们），在此统一注明出处为《维基百科·最大独立集》，《维基百科·最大团》。介于现在的笔者水平有限，部分证明选手们可以自行查阅百科上的论文。如有谬误，请多多包涵。如有爆标的好方法，请大方分享（我有预感会被爆标）！

做完这题，相信大家都会有丰富的体验。但是目前，寻找最大团的算法仅能达到大概$O(1.1888^n)$。笔者希望，此题能成为各位通往更高处的垫脚石，让未来的计算机科学领域群星璀璨！

1 启航，一些部分分！

1.1 第0和第1个数据是暴力搜索最大独立集。

1.2 第2和第3个数据点是$m = n - 1$。它们是连通图，所以可以轻松用dp解决。

1.3 第4和第5个数据是二分图。但是观察给出的部分分表，要求输出的Ai和Bi非常小。因此可以对于每条增广路贪心构造。如果构造不出来可以对于每条增广路的起点随机调整。

1.4 第6和第7个数据是基环树，可以用类似仙人掌上的树形dp解决。

因为这些数据点的n都没有太大，可以使用bitset储存方案（或者手写bst压位）解决。

2 如何继续?

2.0 到第8和第9个数据点，我们发现它没有明显的规律。介于n比较小，我们不妨尝试直接使用1.0.1中的暴力搜索；但在笔者的实现上，这样的效率其低无比。我们先观察第9个数据点，发现边数只有800左右，远小于所有可能出现的边数总数（5460）。结合题面中奇怪的输入方式（分给出的边和断掉的边两种输入），我们不妨尝试**将出现和未出现的边反置**， 观察如何继续。

2.1 对这种建边方式，重新搜索第0和第1个点。我们发现这样搜出的每一个 合法集合 都是

重建图后的完全图。 这种图也称为原图的补图，完全图也称为团。 那么我们得出：

|最大独立集| = |其补图的最大团|

证明是容易的，对每一个原图中的最大独立集，集中点两两间都未连边，而补图中两两间就必然有边，是完全图。而最大的独立集，也就是最大团。

这样搜索效率明显提高。考虑到电脑有2个或4个线程，本地上可加编译开关，这样还是比较快的。

2.2 接下来的第10和11个点对效率提出了更高的要求。

如何优化决策总数？我们首先观察到在接下来的选择中，两个之间无连边的点不能同时选。那么假如x, y间无连边，先选x，再尝试y，或反之，会消耗一些时间。那么我们采用手写bitset实时维护决策，可以一定程度上减少决策数。

更进一步地，我们发现如果不选x ,那么我们必然要选择 y 们中的一个。因此我们可以将x 作为“中枢点",当前的决策集合变为$S \bigcap x$的非邻居们 。这样大大减少决策数，起到了很好的优化效果。

2.3 另一个方向上，我们尝试优化状态总数。由于一个状态可能搜索阶乘遍，我们钦定最大团内选择的元素单调递增。 这样每个最大团只会被搜索一次，也起到了很好的优化效果。

注意： 优化2.2会影响2.3的单调性，因此不能同时使用。经笔者测试，2.3理论上会更快些，可实际上有时2.2会更快。

3 “极大团”

3.0 事实上，我们刚刚做的事情是找出了一些“极大团”以统计出最大团，即无法再添加任何一个点的团。能否精准找到极大团?

3.1 这里有一种以空间换时间的的方法：

假设我们统计出了1~n的极大团，现在要加入一个点n+1，我们开始思考：包含n+1这个点的极大团长什么样?

明显地，这个团中属于1~n的部分无法再添加任何一个<=n的点让它更大（包含了n+1这个点的连边情况）。这个定义和先前<=n的极大团的定义是类似的，只差“要和n+1这个点也联通“这一条!

仍然可以使用手写bst，笔者使用队列保存之前的状态，每次按度数从大到小暴力添加点（并对bst取并），这样就可以在非常快的时间内得出单个极大集，在最大度数极小时表现优异。通过第12和第13个测试点绰绰有余。

4 脑洞时间!

4.0 "尽管随机算法下的最大团处理是困难的，然而这题原题是最大独立集啊！我是不是可以随便乱选，隔一个选一个？ 哈哈，出题人你没想到吧，我啪地一下打个随机，很快啊！"

（在惨痛的经历后，）笔者构造了一番，只要选手的脑洞不是太大应该不能获得很高的分数（尤其是ty=2）；但是实际上，随机算法找出最大团是很困难的，找出一个的较大独立集却比较容易，很多情况下它与答案都比较贴近，选手可以尝试用随机算法贴近正确答案，再使用搜索或dp局部调整，尝试达到最优解。

4.1 “可以粗略地计算上界，然后简单剪枝？”

这个想法是可行的，然而上界要 精确计算/达到 还是很难的，因此这个剪枝（笔者实现）应该没啥用。

4.2 “诶，我已经搞出了两种算法了，它们各有特点：2.3适合决策数小的，3.1适合决策数大的；那我能不能……”

恭喜你：我们可以尝试对度数（或者点数）平衡规划一下；类似于折半，拿度数很大或很小的点们去跑3.1（因为此时极大集个数少，而且这个时间复杂度是可控的）,其他去跑2.3（或者根据具体情况调整）；实测这样跑的比单纯2.3或者3.1快多了!

我们甚至可以折三半或者四半，有兴趣的选手可以尝试一下。

5 终章

5.0 “出题人，为何你第14个点又是基环树？？”

第14个点对后面5个点起提示作用：请将链与联通分量分开处理！

5.1 我们可以发现，数据的后面大部分是有规律的链，前部则是杂乱的连边。进一步观察，我们会发现这些边的两个端点之差好像在一个固定的范围内，并且隔一段就会改变这个范围。处理出这些范围来，发现这些范围一般在几十左右，最大的也就一百多。

初步的想法便是对于长条的链，使用1.2中的算法，先使用tarjan对强连通分量缩店，对于强连通分量使用2或3或4.2中的算法（推荐使用4.2）。

值得注意的是要 **判断度数为零的点** ，以及只挂在上面一长条的链。请注意点数多边数也多的块尽量用2中算法。运行时间太长时可用clock()剪枝。注意合理的常数优化！

最终，我们就能AC此题！

5.2 然而，其实这题只是一道提答题。大家亦可以针对数据使用贪心或dp等方法进行构造，只要超过答案即可。欢迎大家分享自己的方法！

6 最后的最后

这道提答题，虽看似平平无奇，但每一个数据都在日夜更替间以心血酿造。笔者为这道题，总共写了20多k代码，弄得我的老奔腾三四次蓝屏，~~矢子成吨的作业~~ 。但不管怎么说，还是希望大家做题愉快。

虽然我们每个人孤独地来到这个世界上，拥有孤独而独特的使命要完成，但，这并不意味这我们前路孤独。笔者从一开始拥有的想法到题目的完工，在山穷水尽时得到了师长和朋友们无私的帮助与支持。衷心感谢之余，亦可见我们在征程中并不孤独，就像算法2.1中孤独的独立集能转化为团结的最大团那般，孤独与否亦能互相转换。

至暗的黑夜，有时亦是最闪耀的星空。