Problem A

Find a minor

题意：给定某无向图 G，定义一次收缩操作为：对于图中某条边 e={u,v},建立新的结点代替 u,v 并把 u,v 连出的除 e 外的边连到新的结点上，删去 e。问能否通过收缩操作，删点操作，删边操作使得 G 包含 K_n(结点个数为 n 的完全图)或 $K_{n,m}$（两边结点个数分别为 n,m 的完全二分图）为 G 的子图。其中 G 的结点个数 V<=12。

算法梗概：由数据规模便可判断此题能够用搜索解决。不过此题的搜索也不好想，要么是复杂度实在大得可怕，要么是缺少优化的算法。此题搜索方法多样，这里仅提出一种作为参考。

首先对题目进行深入的分析，如果 G 能通过操作最后包含目标图，那么目标图中的每一个结点在原图中均是一个连通的点集，而这个点集必然是可以表示成一棵树的，也就是说我们可以对每个结点搜索与它相连的哪条边被收缩，进而求出每个连通块再进行判断，然而这样对二分图的判断会有点麻烦，并且不难想到这样搜出来的情况会有很多重复。而我们注意到每棵树中的每个点都可以当作根，我们不妨令结点编号最小的点为根，我们把整个图中的所有根搜出来，再对其他结点进行搜边的操作，其中搜根的时候还可以保证我们搜出的根递增（如果是二分图就在两边分别递增），在 n 或者 n+m 很大的时候这将减去很多冗余的搜索。第二步搜边，搜边的时候可以加些优化，由于我们默认搜出的根是该连通块中编号最小的，所以当有编号更小的点连向根时，可以剪掉，当然我们用并查集等类似数据结构记录已经连向当前点的最小点，并当当前点连向根的时候作出判断，然而这样会比较麻烦，并且会增加常数，而只判断当前点，就已经能收到很好的效果了。另外还有一些优化诸如剩下的边比目标图少就剪去。判断直接暴力即可，因为 V 很小，不会浪费很多时间。

还有一个并不是搜索中的优化，如果 n>=2 且 m>=2（如果有 m 的话）或 n>=3(没有 m)时，原图中的度为 1 的点可以直接去掉，因为它既不能成为目标图中的一个单独的点，它唯一的边的收缩也不会给其他点带来任何变化，所以直接去掉不会影响正确性。

时空复杂度：时间复杂度 $O(V^v)$，空间复杂度 O(V)。

Problem B

Bricks

题意：给定很多长方体的位置，他们高度都一样，分为两层叠放，其中上层一个长方体是稳定的当且仅当底面在四个方向上的半个长方形均有其他某个下层的长方体的底面与其重叠（接触不算），初始状态是稳定的，问最多能从下层移走多少长方体使上层保持稳定。移动的时候只能朝四个方向之一移，并且不能转弯，只有这个方向上不会与其他长方体相撞才能移动。两层的长方体数分别为m、n，m,n<=10;

算法梗概：看数据规模似乎又可以搜索，但事实上有更优的算法，对于某个下层长方体的取舍情况，后面的移动显然跟前面的移动毫无关系，所以这是满足无后效性的，于是想到集合 DP，只需要用 2^n 个状态来记录下层的某种取舍情况可不可行即可，可以使用记忆化搜索来实现。对于稳定的判定问题，可以直接把这 2^n 种情况的是否稳定在一开始就处理好，对于不稳定的直接判掉，对于能否移动的问题，也可以在预处理中解决，由于 n 非常小，直接暴力 $O(n^2)$即可。

时空复杂度：时间复杂度 $O(2^n*n^2)$，空间复杂度 $O(2^n)$

Problem C

Color Squares

题意：在 3*3 的方格中，初始时方格全是白色，然后有一系列染色规则：

1：蓝色可以直接染。

2：与蓝色格子相邻的格子可以染红色

3：同时与蓝色格子和红色格子相邻的格子可以染绿色。

4：同时与前三种颜色的格子相邻的格子可以染黄色。

其中不同的颜色有不同的分值，染色会覆盖原有颜色，问至少进行多少次染色操作使总分值>=某个常数 w。

算法梗概：非常简单的一道题，直接可以用 $0\sim5^9-1$ 来表示所有的颜色状态，预处理时直接用 bfs 搜出所有状态到达的最少染色次数，并预处理出各种状态中的各个颜色的格子的数目，并且对每组数据暴力求最小的染色次数即可。

时空复杂度：O(5^9*5*9+testnum*5^9*5)(testnum 是指数据的组数，把 testnum 包括进来是因为无论 testnum 是多少，bfs 都只要进行一次。空间复杂度 O(5^9*5)。

Problem D
Difficult Melody

题意：给出 n 个串，长度最大为 100，并给出每个串是否是“好的”，对于每个长度为 k 的串，如果它在 n 个串中的 p 个出现过，并且这 p 个串中有 q 个是好的，如果 p 大于等于某个常数 m，那么称 q/p 为这个长度为 k 的串的容易度，需要找出容易度最小的串，如果两个串容易度一样，取相应的 p 更大的，若 p 也一样取字典序最小的。k<=20,n<=100。

算法梗概：碰到这样需要判断某个串是否是另一个串的子串的题，就会立即想出一些类似 KMP，AC 自动机等字符串匹配的算法，这题的匹配串可以理解为有 n*(maxlen-k+1)个（每个串中有 maxlen -k+1 个长度为 k 的子串），被匹配串则有 n 个，而匹配至少要 n 的复杂度，基于匹配的算法的总复杂度就至少有 $O(n^3*maxlen)$，所以匹配算法是不行的。于是又想到另一种关于字符串的算法，后缀数组，我们可以把所有后缀排序，然后对每个匹配串在其他串中二分查找，这个算法的复杂度可以过这题，但似乎比较烦琐。既然基于排序的算法能解决这题，我们何不想一个更简单的排序算法?事实上，我们只要把所有的匹配串拿出来，暴力比较排个序，并从最小的到最大的计算，对每个匹配串排序的时候，就记录下它属于哪个原串，排好后，对于相同的串，我们很容易就能统计出它的 p，q 了，然后直接更新答案。这个算法比后缀排序算法显然要简单，直观的多。

时空复杂度：排序的复杂度为 O(n*maxlen*log(n*maxlen)*k)，求答案的复杂度为 O(n*maxlen*k)，总复杂度为 O(n*maxlen*k*log(n*maxlen))，空间复杂度为 O(n*maxlen)。

Problem E

Expensive Drink

题意：给出 N 个如同 $pl_i<=a_i*x+b_i*y+c_{i*}*z<=pr_i$ 的不等式(其中 x,y,z 为变量, $a_i,b_i,c_i>=0$, N<=100, 0<=x<=y<=z), 给定 a_0,b_0,c_0，$S=a_0*x+b_0*y+c_0*z$ 的最大值。

算法梗概：这是道偏数学的题，这题显然可以用线性规划等方法来做，不过线性规划实现非常麻烦，因此需要找到另一种更为简洁的方法来实现。

首先对题目进行一点小小的转化，令 $a_i=a_i+b_i+c_i$，$b_i=b_i+c_i$, 并把 0<=x<=y<=z 转化为 x,y,z>=0,这会使后面的讨论简化很多。设 x = x_0, y = y_0, z = z_0 时，S 取到最大值，然后分四种情况讨论：x_0 ,y_0 ,z_0

1：x_0 ,y_0 ,z_0 > 0，可以证明，在这种情况下，必然存在三个不同的数, i, j, k，使得对于第 i，第 j，第 k 个不等式有某侧取到等号，并且三个不等式中必有一个是右侧取到等号。

证明：若 x_0 ,y_0 ,z_0 无法使任何不等式取到等号，我们对 x_0 ,y_0 ,z_0 进行调整，可以把 x_0 加上某个 dx,dx 是个足够小的正数使得 x_0 加上 dx 后还是满足所有不等式，则此时 S 的值不会变小，不停地进行此操作直到碰到某个不等式的上界。此时，设碰到第 i 个不等式的上界,则 $a_ix_0+b_iy_0+c_iz_0=pr_i$，如果对于此 x_0 ,y_0 ,z_0 无法使其他 N-1 个不等式取到某侧的等号，则每次把 x_0 加上 $k*b_i$,y_0 减去 $k*a_i$，适当选择 k 的符号，可以使 S 不减，必然存在足够大绝对值的 k 使得新的 x_0,y_0,z_0 使另一个不等式取等号，否则就必然有 x_0,y_0 中某个无穷增加或等于 0，这都不属于这种情况讨论的范畴，用同样的方法同时对 x_0 ,y_0 ,z_0 调整可达到第三个不等式的上界或下界，证毕。

对于这种情况，一种简单的想法就是枚举出 i,j,k，并且枚举左侧还是右侧取到等号，求出 x_0 ,y_0 ,z_0,再判断是否满足其他不等式，若满足则更新答案，这种方法的时间复杂度高达 $O(N^4)$，显然不够理想。而事实上如果我们只枚举其中的两个达到边界的不等式 i，(默认 b_i*c_j !=b_j*c_i，如果等于的话，由证明我们可以继续调整并找到其他不等式代替)我们就可以把 y,z 关于 x 的表达式求出来，代入到原来的不等式中，求出 x 的范围，如果此范围合法，就把 S 也转化为 x 的一次函数，求出一个上界。这种算法的复杂度为 $O(N^3)$，可以满足题目的要求。

2：x_0 ,y_0 ,z_0 中有且仅有一个等于 0， 这相当于原题的二元版本，用同样方法可以证明必有两个不等式取到等号，我们只需枚举一个不等式，并求范围即可。

3：x_0 ,y_0 ,z_0 中有且仅有一个不等于 0，直接枚举即可。

4：x_0 ,y_0 ,z_0 都等于 0,这种情况仅用来判断原不等式是否有解。

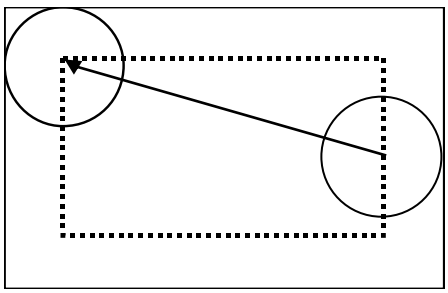
实现时，先把 ans = -1，如果四种情况讨论完 ans 还是-1，则无解，否则，对 i > 0, 如果 c_i 全为 0 而 c_0>0，或 b_i 全为 0 而 b_0>0，或 a_i 全为 0 而 a_0>0 则答案为无穷大，否则答案即为 ans。

时空复杂度：空间复杂度为 O(N), 时间复杂度为 $O(N^3)$。

Problem F

Rotating a Frame

题意：已知一个长方形四个顶点的坐标，一开始有个半径为 R 的圆紧靠着长方形的某个角，然后长方形绕着圆顺时针转动（圆不动），转动的时候长方形与圆之间不滑动，并且长方形始终与圆接触，转动的角速度为 w deg/s，转动的时间为 T s，求 Ts 后长方形四个顶点的位置。



算法梗概：换种角度思考，想象是圆在转而不是长方形在转，先求出圆心运动的轨迹（一个更小的长方形（右图中的虚线），其中圆心的初始坐标为(sx,sy),然后求出圆心的运动速度 v=w*pi/180*R，圆心 Ts 内移动的轨迹长度 L=v*T，圆心 Ts 后的坐标(fx,fy)，然后转换坐标系：求出圆逆时针转过的弧度 s=w*pi/180*T，以圆心为原点把长方形的四个点顺时针转(-s)rad，利用复数求出转动后的坐标，然后把圆归位，把所有点都平移(sx – fx, sy – fy)。

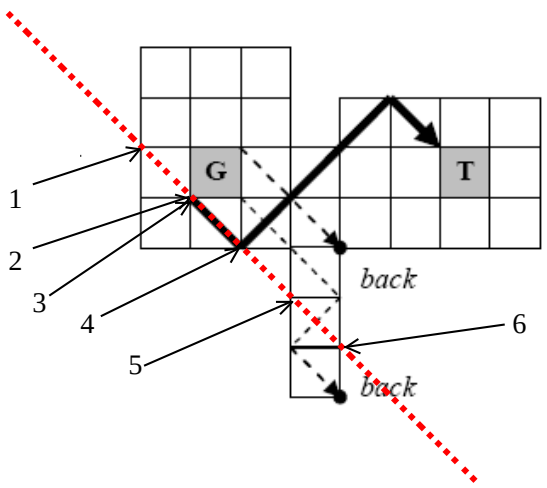
时空复杂度：均为 O(1).

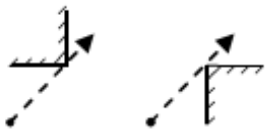
Problem G

Shoot Your Gun!

题意：给出三个边均垂直于坐标轴的多边形（不一定为凸多边形）M,G,T, 其中 G,T 包含在 M 中且不与 M 接触并且不相互接触，你可以在 G 的边界上任选一点，从对角线的四个方向出射，经过 M 的边的反射最后到达 T 边界上的某点，中途不能接触 G，求反射的最小次数。其中碰到拐角时，若形成拐角的两线段在入射直线的同一边，就直接通过这点，不反射，否则就原路返回。其中多边形最多有 MAXN = 50 条边，顶点坐标最多为 MAXLEN = 4000。

算法梗概：首先把图中的无限个点通过转化变为有限个点。把 x,y 坐标均为整数的点称作整点，长度为 1，两端为整点的线段称为单位边，则从某个单位边上（非端点）以相同方向出射的两点，必然会以相同的入射角射到同一个单位边上，并以相同的出射角从该边上反射出去，因此，我们可以只考虑从单位边的中点出发，而不考虑其他点，再把整个图的所有 x,y 放大两倍后，我们便只需要考虑整点出发的情况了。一个简单的想法就是 DP，f[i][j][dir]表示从整点(i,j)出发，方向为 dir，还需要弹几步到 T，而这个 DP 需要的时空复杂度都高达 $O(MAXLEN^2)$，无法满足题目的数据规模而我们注意到它每个状态的来源状态和能推到的状态都只有一个，因此把所有点都拿出来太浪费了些。可以只把 M,G,T 边上的点取出来，并且连上边，最后 bfs 下就能把答案求出。建图连边时，先考虑左上—右下的边，我们把在同一条斜线上的点取出来(如右图的红色虚线)，按照 x 坐标排序，然后按照排序结果 1 连 2,3 连 4,5 连 6……（注意：其中 2,3, 为同一个点，只不过加了两次，这里 2，3 是顺序序号），如果碰到左图的情况，若这个点是 M 的点，就不要算，如果这个点是 G 或 T 的点，则要算两次（如右上图中的 2,3），算两次或不算仅仅是为了排好序后可以直接两两连边，而不需要对种种特殊情况进行讨论。同样的把所有右上-左下的点也这样连边，连好以后以 G 上的所有点为起点，T 上的所有点为终点，Bfs 即可得到答案。因为在 M 边上的点最多只有两条边，这两条边形成一个反射，而我们 bfs 出的最短路中间点均为 M 上的点，它左右两点恰好为它的连出去的不同两点，因此这个算法是正确的。





时空复杂度：设整个边界上有 P 个点，空间复杂度为 O(P)，每个点连出 O(1)条边，因此最多有 O(P)条边，Bfs 的复杂度为 O(P)，而连边时需要对所有点排序，复杂度为 O(P log P)因此总复杂度为 O(P log P),其中 P = O(MAXLEN * MAXN), 由于有三个多边形，我们还把坐标乘了 2，因此 P 理论上可以达到 4000*2*50*3

=1200000,而实际上数据很难出到这个数量级的 P（事实上有些恶心数据可以），因为 G,T 在 M 中，他们不能接触，多边形的很多边长将远远不到 4000，因此一般情况下 P 将远远不到这个数字，O(P log P)还是能够出色地完成原题所描述的任务的。

Problem H
Help Little Laura

题意：给出某有向图，图中不存在双向边，每个边有个权值，并且权值有可能为负数，可以不断在图中选一个圈染色，但染过的无法再染，求最多的染色边的得分和为多少。点数 N<=100，边数 M<=500。

算法梗概：很容易发现这个问题跟网络流的相似之处——容量满的边不能再流，因此，可以把整个图看成一个网络流，有向边的流量为 1，而我们要求的就是能流过的边最大的价值，由于引入了网络流中反向边的概念，我们只需要每次找价值为正数的环，并把这个环流一圈，继续找。而实际操作的时候，一开始先不加反向边，也不管什么残量，找到一个圈，把圈上所有的边反向，价值也取相反数，其实就是保证目前所有的有向边的残量为 1。这个算法正确性的证明可以参照网络流不停增广的正确性的证明。而正价值的圈可以利用 bellman-ford 求，直接求最长距离即可，跟求负权环的 bellman-ford 求最短距离刚好相反。

时空复杂度：每次求正权环的复杂度都是 O(NM)，由于题目中 NM 很小，因此求圈是很快的，修改的复杂度是 O(N)，最终复杂度跟总共找的圈的个数 P 有关系，这个个数的上界是非常巨大的，它等于所有中间状态（即所有点满足出度＝入度的状态）的个数，并且很难估计……然而事实上，很难有数据能让 P 达到甚至接近这个上界，特别是当 NM 都很大的时候，因此研究 P 的上界没有很大的意义，总复杂度为 O(PNM)，空间复杂度为 $O(N^2)$。

Problem I
Integer Transmission

题意：给出一个长度为 N 的二进制串，N<=64,这个串的第 i 位将在时间 i 的时候接收到，有个整数 d <=n,，每一位有可能最多延迟 d 收到，对于同一时间收到的位，可以以任意顺序排列，问最多能收到多少种不同的二进制串，其中最小的与最大的是多少。

算法梗概：DP，状态 f[i][j]表示在 i+j+d 的时候，接受到序列的前 i+j 位有 i 个 0,有 j 个 1 的二进制串的个数。由于延迟最多为 d，因此在 T = i+j+d 的时候，原序列的前 i+j 位必然已经收到，因此状态是合法的。递推的时候从 f[i][j]推到 f[i+1][j]和 f[i][j + 1]，若忽略限制条件，正常的递推方程为 f[i+1][j] += f[i][j],f[i][j+1] += f[i][j]。其中 f[0][0] = 1 为递推的边界状态，f[tot[0]][tot[1]]为最终答案，其中 tot[0],tot[1]表示原串中 0 和 1 的个数。

递推中有以下的限制条件：cnt[0],cnt[1]分别表示原串前 i+j+d 位的 0 的个数，1 的个数,如果 cnt[0] = i,则此时 0 已经放满了，不能放第 i +1 个 0,那么 f[i][j]不能推到 f[i+1][j],如果 0,1 都有，但是第 i+1 个 0 在原串中的位置比第 j+1 个 1 在原串中的位置+d 还要大，也就是说第 j+1 个 1 无法 delay 到 i+1 发送的时间，那么此时也不能从 f[i][j]推到 f[i+1][j]，从 f[i][j]推到 f[i][j + 1]的限制条件类似。

求最小值最大值用贪心即可，从左向右确定 Min 的每一位，mincnt[0]和 mincnt[1]表示目前 Min 中有几个 0 和 1，此时的限制条件就相当于把 mincnt[0]代给 i, mincnt[1]代给 j,f[i][j]向外推的限制条件，能添 0 就添 0，否则添 1,Max 的求法类似。

时空复杂度：时间和空间复杂度均为 $O(N^2)$。

Problem J

Jewel Trading

题意：你需要卖一个钻石，你的出价 p 如果大于一个常数 a 那么买家同意的概率是 $1/(1+(p-a)^b)$，其中 b 是一个小于 10 大于 1 的常数，你有 n（1<=n<=100）次机会报价，并且一定要在 n 次中卖掉这个钻石，求价格的期望最大值。

算法梗概：一道比较简单的数学题，方法不难想到。令 f[i]为 i 次报价能得到的最高价格期望，很显然 f[1]=a。并且也很容易写出 f[i]的递推关系式：$f[I]=\max\{p/(1+(p-a)^b)+f[I-1]*(p-a)^b/(1+(p-a)^b)\}$(其中 p 是第 I 次叫价的决策，p 是**整数**)，令 max 括号中的表达式为 g(p)，可以对 g(p)求导，求得 $g'(p)=(p-a)^{b+1}((b-1)*p+a-b*f[I-1])$，由于 p 一定大于 a，$(p-a)^{b+1}$ 是恒大于 0，又由于 b>1，(b-1)*p+a-b*f[I-1]是递增的，因此当且仅当 p=(b*f[I-1]-a)/(b-1)时，g'(p)取 0，分别尝试这个值取上下整，得到最优的 f[I]的值。这样做 n-1 次递推即可得到 f[n]。

时空复杂度：时间复杂度 O(n)，空间复杂度可以达到 O(1)。