IOI2014 中国国家集训队第一次作业

试题泛做

沈洋

浙江省镇海中学

Contents 目录

ACM-ICPC World Final 2003

ACM-ICPC World Final 2004

ACM-ICPC World Final 2005

ACM-ICPC World Final 2006

ACM-ICPC World Final 2007

ACM-ICPC World Final 2008

ACM-ICPC World Final 2009

ACM-ICPC World Final 2010

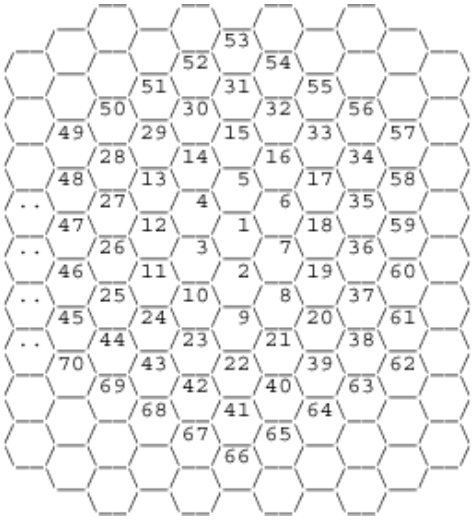
ACM-ICPC World Final 2011

ACM-ICPC World Final 2012

ACM-ICPC World Final 2013

1999 A - Bee Breeding

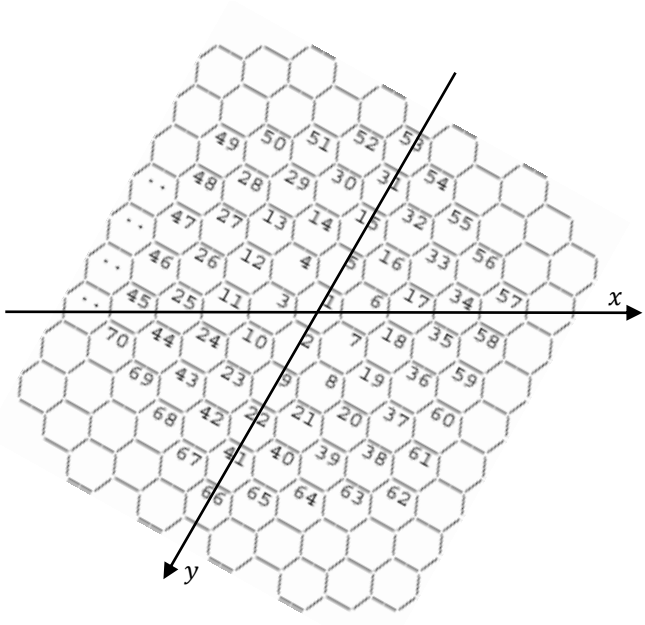
题目大意



有一六边形网格按如图所示的方法标号。在某一个格子上时，我们可以移动到与其有公共边的六个格子中的一个。a号格子到b号格子的最短路定义为从a号格子移动到b号格子所需的最少步数。给定一些(a,b)，求a号格子到b号格子的最短路。

算法讨论

本题考查数学。



如图旋转网格并建立坐标系，因此每个格子得到如下坐标：

编号	坐标
1	(0,0)
2	(0,1)
3	$(-1,0)$
4	$(-1,-1)$
5	$(0,-1)$
6	(1,0)
7	(1,1)
8	(1,2)
…	…

在这个坐标系下，(x,y)可向$(x-1,y)$、$(x+1,y)$、$(x,y-1)$、$(x,y+1)$、$(x-1,y-1)$或$(x+1,y+1)$移动。记起点为(x_0,y_0)，终点为(x_1,y_1)，不妨设$y_0 \le y_1$，因此我们只需考虑$(x-1,y)$、$(x+1,y)$、$(x,y+1)$和$(x+1,y+1)$四种移动。不难发现最小步数如下：

$$ans=\begin{cases}(x_0-x_1)+(y_1-y_0), & x_1 \le x_0\\ y_1-y_0, & x_0 < x_1 \le x_0+(y_1-y_0)\\ x_1-x_0, & x_1 > x_0+(y_1-y_0)\end{cases}$$

时间复杂度

$O(\max(a,b))-O(1)$

空间复杂度

$O(\max(a,b))$

1999 C - A Dicey Problem

题目大意

给定一个骰子地图以及骰子初始放置位置，可按照一定规则在地图中上下左右移动骰子，求一条路径使得骰子从起始位置开始移动最终回到起始位置。

算法讨论

本题考查图论建模和 BFS。

首先注意到根据骰子顶面和面向观察者的那面（不妨称为正面）写着的数字不同，骰子共有 24 种不同的状态。我们对这些状态进行编号如下：

编号	顶面	正面
0	1	5
1	1	4
2	1	2
3	1	3
4	2	6
5	2	3
6	2	1
7	2	4
8	3	6
9	3	5
10	3	1
11	3	2
12	4	6
13	4	2
14	4	1
15	4	5
16	5	6
17	5	4
18	5	1
19	5	3
20	6	5
21	6	3
22	6	2
23	6	4

进而我们可以计算出每个状态在地图上向上、向右、向下以及向左移动得到的新状态。结果如下表：

编号	向上	向右	向下	向左
0	16	9	6	15
1	12	17	10	7
2	4	13	18	11
3	8	5	14	19
4	20	12	2	8
5	9	21	13	3
6	0	10	22	14
7	15	1	11	23
8	23	4	3	16
9	17	20	5	0
10	1	18	21	6
11	7	2	19	22
12	21	16	1	4
13	5	22	17	2
14	3	6	23	18
15	19	0	7	20
16	22	8	0	12
17	13	23	9	1
18	2	14	20	10
19	11	3	15	21
20	18	15	4	9
21	10	19	12	5
22	6	11	16	13
23	14	7	8	17

我们把每个格子 (x,y) 拆成 24 个点 (x,y,z)，其中 $z = 0 \dots 23$，表示骰子以 z 状态经过格子 (x,y)。设 (x,y,z) 向上下左右某个方向移动之后的状态为 (x',y',z')，若 (x',y') 格子上写数与 z 状态下骰子顶面的数相同，或者 (x',y') 格子上画着星星图案，我们就建边 $(x,y,z) \to (x',y',z')$。

根据题目给出的初始状态下骰子顶面与正面的数，我们可以算出骰子的初始状态 $z0$，这样我们就确定了起点 $S = (x0,y0,z0)$；最终我们需要骰子回到 $(x0,y0)$，因此可行的终点集合为 $T = \{(x0,y0,z) \mid z = 0 \dots 23\}$。于是题目就转化为：求一条路径，起点为$S$，终点$\in T$。我们从$S$开始对图进行 BFS 即可解决这个问题。

时间复杂度

$O(R \cdot C)$

空间复杂度

$O(R \cdot C)$

1999 D - The Fortified Forest

题目大意

有n棵树，其中每棵树有四个参数x_i, y_i, l_i, v_i，分别表示这棵树的横坐标、纵坐标、砍掉它能制作的栅栏的长度以及它的价值。现在要砍掉其中一些树使得砍掉的这些树制成的栅栏能够包围剩下的树，要求砍掉的树价值最小，求砍树方案。若有多个方案可行，求砍掉的树的个数最少的方案。对于这个方案，计算最多能剩下多少栅栏。

算法讨论

本题考查枚举、凸包。

我们枚举砍掉了哪些树，那么要包围剩下的树所需的栅栏长度的最小值即为剩余这些点的凸包周长。这样我们只需简单地比较砍掉的树所能制造的栅栏长度与凸包周长的大小即可知道方案是否可行以及剩余栅栏长度。在所有可行方案中按题目要求取最优方案即可。若每次求凸包时重新排序则时间复杂度为$O(2^n \cdot n \log n)$，我们可以通过调整枚举顺序或进行一定的预处理来使时间复杂度降为$O(2^n \cdot n)$。

时间复杂度

$O(2^n \cdot n)$

空间复杂度

$O(n)$

1999 E - Trade on Verweggistan

题目大意

有n堆物品，第i堆物品有m_i个，每个物品都有它特定的价格，你可以买一些物品，然后以每个 10 元的价格卖出，从而获利。对于某一堆物品，你可以从上到下买任意数量的物品（买了某个物品的话必须把它上面的物品都买下），堆与堆之间互不影响。求最大获利，以及达到最大获利时可能买了多少个物品。

算法讨论

本题考查贪心。

因为堆与堆之间是无关的，我们只需要考虑某一堆的情况。我们可以先从上向下扫描一遍算出最大获利，再扫描一遍算出可能买了多少个物品，只保留前 10 个可行的方案。

之后我们要合并所有堆的方案，我们只考虑如何合并两堆的方案，然后依次合并所有堆即可。对于两堆的合并，设第一堆的某种方案为a_i，第二堆的某种方案为b_j，我们暴力算出所有的$a_i + b_j$，然后排序去重并保留前 10 个即可。当然我们也可以用堆来做到更好的复杂度，不过此题范围较小，并无必要。

时间复杂度

$O(nm)$

空间复杂度

$O(m)$

1999 H - Flooded!

题目大意

有一容器底面被分成$n \times m$小格，每小格都是$10\ m \times 10\ m$，并且有自己的海拔。现向容器中注入$V\ m^3$水，问水面海拔以及有百分之多少的底面积被覆盖。

算法讨论

本题考查排序、模拟。

我们首先对所有区域按海拔从低到高进行排序，然后从小到大枚举水淹到了哪个区域。设当前枚举到第i低的区域，我们可以计算出若水面恰好与第$i+1$低的区域平齐时水的体积，当它达到或超过目标体积时停止。于是水的海拔就在第i低的区域的海拔与第$i+1$低的区域的海拔之间，便容易计算出答案了。

时间复杂度

$O(nm \log nm)$

空间复杂度

$O(nm)$

2000 A - Abbott's Revenge

题目大意

给定一个网格图，每次你可以向上下左右中的某些方向走一步，可以走哪些方向与你进入这个节点的方向有关，并且在输入中给定。已知起点 S 和你第一步的方向，求 S 到 T 的最短路径。

算法讨论

本题考查 BFS。

我们把原图中的一个点i拆成四个点i_N, i_S, i_E, i_W，分别表示从对应方向进入该节点。拆点之后按照输入给定的方案连边，即可得到一张新图。我们发现现在起点并没有对应新图中的某个点，但是第一步走到的点在新图中是确定的，因此我们可以直接从第一步走到的点开始 BFS，直到访问到目标位置对应的四个点中的某一个时停止，即可找出一条最短路。

时间复杂度

$O(n)$，其中n表示输入中给出的点数。

空间复杂度

$O(n)$，其中n表示输入中给出的点数。

2000 B - According to Bartjens

题目大意

给定一个数字串，要求在其中添加至少一个运算符符号，使得运算结果等于 2000。另外还有如下规定：

1.数字没有前导零。例如 2*10*0100 就是不可行的。

2.表示 0 的时候不允许多个 0。例如 2*1000+000 就是不可行的。

3.只用二元运算符，不用取负。所以 2*-100*-10+0=也不合法。

4.只用+、-、*，不用/和括号。

5.这些算式按照正常的优先级顺序计算。

按照字典序输出所有可能的方案。若没有方案输出 IMPOSSIBLE。

算法讨论

本题考查搜索。

我们按位搜索每个数字之后是否添加符号，以及添加了什么符号，注意搜索时需要记录当前位置是否为算式中某个数的第一位，如果是第一位而且是 0 的话，其后面一定要放运算符（当然，也可以是等号）。这样搜索有一个问题，即不能保证至少放入一个运算符，但是我们发现不放运算符就可以等于 2000 的情况只有原数字串就是 2000 这一种，因此我们可以特判一下这种情况。最后要按照字典序输出方案，我们可以在搜索的时候注意搜索顺序，按照“*”、“+”、“-”、“不放”的顺序搜索，或者直接搜出所有结果之后排序即可。

时间复杂度

$O(4^n \cdot n)$

空间复杂度

$O(n)$

2000 C - Cutting Chains

题目大意

给定一张包含n个点的图，你可以选择一些点并修改它们的连边情况（修改某个点的连边情况是指添加和删除一些与其相连的边），使得整个图变成一条链，求最少需要选择的点数。

算法讨论

本题考查枚举。

由于本题n比较小，我们可以枚举修改了哪些点的连边情况。假设我们选择了m个点，考虑从原图中删去这m个点以及与其相连的边之后形成的新图，不难发现当且仅当新图中连通块的个数不大于$m+1$，并且每个连通块都是一条链或者一个点时，这种选法是可行的。我们按照选择的点数从少到多枚举所有可能的选法，找到第一种可行的选法即为答案。

时间复杂度

$O(2^n \cdot n^2)$

空间复杂度

$O(n^2)$

2000 E - Internet Bandwidth

题目大意

给定一个n个点m条边的流网络，边的正反向容量相等，求S到T的最大流。

算法讨论

本题考查网络流。

使用任意一种最大流算法即可。我选用的是 SAP 算法。

时间复杂度

$O(n^2m)$，实际效果较好，很难达到这个上界。

空间复杂度

$O(n+m)$

2000 F - Page Hopping

题目大意

给定一个n个点m条边的有向图，每条边的边权都是 1，已知该图的任意两个点之间均连通，求所有点对的最短路的平均值（自己到自己不计入答案）。

算法讨论

本题考查最最短路。

由于本题中边权都为 1，因此我们可以从每个点开始 BFS 求出该点到其他点的最短路，并计算所有最短路的平均值。当然，直接使用 Floyd 算法求出任意两点间的最短路也是可行的。

时间复杂度

$O(nm)$或$O(n^3)$

空间复杂度

$O(n+m)$或$O(n^2)$

2001 A - Airport Configuration

题目大意

一个机场有上下两排门，各n个。从上排第i个门到下排第j个门的距离$d_{i,j}$为$|i-j|+1$。该机场接待n个城市之间互相转机的客流，并且已知任意两个城市之间在该机场转机的平均客流量$a_{i,j}$。现假设上排第i个门对应城市p_i的到达客流，下排第j个门对应城市q_j的出发客流，求$\sum_i^n \sum_j^n d_{i,j} \cdot a_{p_i,q_j}$。

算法讨论

本题考查模拟。

我们只需枚举i,j后计算$d_{i,j} \cdot a_{p_i,q_j}$，再简单相加即可。注意将题目中未给出的客流量设为0，并且最后排序输出的时候使用双关键字排序。

时间复杂度

$O(n^2)$

空间复杂度

$O(n^2)$

2001 B - Say Cheese

题目大意

在一个三维空间直角坐标系中，你现在位于点(x_s, y_s, z_s)，目标是点(x_t, y_t, z_t)，一般情况下，你的移动速度是每十秒一个单位长度。在这个空间内有一些球(x, y, z, r)，表示球心坐标为(x, y, z)，半径为r，你在这些球的内部移动是不需要时间的。球于球之间允许相交。求你到达目标所需的最短时间。

算法讨论

本题考查最短路。

我们可以把起点和终点当作半径为 0 的球，定义两个球(x_i, y_i, z_i, r_i)与(x_j, y_j, z_j, r_j)之间的距离为在空间只有这两个球的情况下，从一个走到另一个的最短时间，即

$$d_{i,j} = 10 \max\left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} - r_i - r_j, 0\right)$$

接下来我们只需应用 Dijkstra 算法，求出起点和终点对应的两个球之间的最短路即可。

时间复杂度

$O(n^2)$

空间复杂度

$O(n)$

2001 F - A Major Problem

题目大意

给定两个大调的起始音符，要求把第一个大调中的某些音转换成第二个大调中对应位置的音，或判断出给定的音符不是第一个大调中的音符。

算法讨论

本题考查模拟。

首先列出 CDEFGAB 的相对音高（相对于 C，以一个半音为步长，$mod\ 12$）：

音名	C	D	E	F	G	A	B
音高	0	2	4	5	7	9	11

每个音的升调或降调之后，其相对音高对应变成原来的音高$+1$或者-1，即如下表：

音名	C	D	E	F	G	A	B
音高	0	2	4	5	7	9	11
#	1	3	5	6	8	10	0
b	11	1	3	4	6	8	10

于是我们对于某个大调的起始音查表即可得到它的相对音高，依次$+2, +2, +1, +2, +2, +2$即可得到该大调其他音的相对音高；字母方面由于要求一个大调中的字母都不相同，所以从起始音的字母开始，依次向后推一个字母即可得到每个音对应的字母。接下来根据字母和音高确定每个字母是升调、降调还是不变，即可推出该大调完整的七个音。

这样，我们分别推出两个大调中完整的七个音，找到待转换音符在第一个大调中的位置，并输出第二个大调中的对应音符即可。

时间复杂度

$O(1)$

空间复杂度

$O(1)$

2001 H - Professor Monotonic's Network

题目大意

给定一个包含n个变量，m个比较器的比较网络，问：

- 该网络是否是排序网络。
- 每个比较器运行需要 1 单位时间，无拓扑关系的比较器可以并行运行，求该排序网络的运行时间。

算法讨论

本题考查排序网络。

对于第一问，根据排序网络的 0-1 原则，我们枚举所有的可能的长度为n的 01 序列并检查是否都能正确排序，若能则该比较网络是排序网络，否则不是。

对于第二问，由于输入给定的是一个比较器的拓扑序，因此对于某个比较器(a_x, a_y)我们可以确定a_x和a_y上一次出现的比较器即为该比较器的两个前驱，因此该比较器执行完成的时间就是两个前驱执行完成时间的较大值+1。因此我们只需按执行顺序扫描比较器并维护当前每个变量最后出现在哪个比较器中，即可依次推出每个比较器执行完成的时间。整个排序网络的运行时间即为所有比较器运行完成时间的最大值。

时间复杂度

$O(2^n \cdot m)$

空间复杂度

$O(n + m)$

2002 A - Balloons in a Box

题目大意

给你一个盒子和n个可以放气球的位置，每个气球可以看做一个球体。每次你可以选择一个还未选择过的位置放入气球，并充气使其半径增大到不能增大为止，求盒内剩余体积的最小值。不选某些位置也是被允许的。

算法讨论

本题考查搜索。

我们可以先通过 DFS 的方式枚举气球放入的顺序，之后题目转化为一道模拟题。

对于某个正准备放入的气球i，限制来自于盒子的壁和已经放入的气球两个方面。对于盒壁，它对球半径的限制是球的半径不能大于球心到任意一个盒壁的距离；对于已经放入的某个气球j，设其半径为r_j，球i与求j的距离为$d_{i,j}$，那么它对气球i的半径的限制是球半径不能大于$d_{i,j}-r_j$，若$d_{i,j}<r_j$，则表明气球i无法被放入。我们对所有这些限制取最小值，即可得到气球i最大能扩展到的半径。

最后对于所有放入顺序，取盒内剩余体积最小值即可。

时间复杂度

$O(n!\cdot n^2)$

空间复杂度

$O(n^2)$

2002 C - Crossing the Desert

题目大意

一个沙漠被抽象成二维平面，沙漠中有n个点，其中点 1 是你现在所在的位置，点n是你的目标，其余点是绿洲。走路是要消耗水和食物的，每走 1 个单位需要消耗 1 个单位的水和食物。食物只能在点 1 买到，但你可以把食物储存在任意一个点上，水在任意一个点上免费无限量供应。你在走路过程中所能携带的最大的水和食物的总和为lim。求在满足以上条件的前提你能否到达目标点，若能，你最少需要购买多少食物。

算法讨论

本题考查最短路。

不难发现发现最后的方案一定类似于这样：先从点 1 买好全部所需的食物，然后不断往返于点 1 与第一个停靠点之间，直到搬运完所有的食物，接着再往返于第一个与第二个停靠点之间，将剩余食物都搬运至第二个停靠点……最终到达目标点。考虑某个停靠点u，设它前一个停靠点为v，假设我们需要搬运x个单位的食物到u，我们考虑至少需要搬运多少食物到v。设u与v的距离为d，因为每次最多能带lim单位的食物和水，因此搬运c次并让自己从u到v最多可以搬运c$(lim-3d)+d$单位食物，因此搬运x单位食物至少需要搬运max$\left(\left\lceil\frac{x-d}{lim-3d}\right\rceil,1\right)$次（注意$lim\le 3d$的情况需要特判），因此点$v$至少需要有$x+2d\max\left(\left\lceil\frac{x-d}{lim-3d}\right\rceil,1\right)-d$单位的食物。设$f[i]$表示若要从点$i$走到点$n$，点$i$上至少需要的食物数，不难得到$f[n]=0$，转移方程：

$$f[i]=\min\left\{f[j]+2d_{i,j}\max\left(\left\lceil\frac{x-d_{i,j}}{lim-3d_{i,j}}\right\rceil,1\right)-d_{i,j}\;\middle|\;1\le j\le n,j\ne i\right\}$$

（其中$d_{i,j}$表示点i与点j的直线距离）。注意到该转移方程具有后效性，因此不可以进行动态规划，需要使用某种最短路算法来计算$f[i]$。于是最终答案即为$[f[1]]$。注意到本题中“边权”非负，因此可以用 Dijkstra 算法作为本题的最短路算法。

时间复杂度

$O(n^2)$

空间复杂度

$O(n)$

2002 E - Island Hopping

题目大意

有一个群岛包含n个岛屿，已知它们的坐标和岛上的人数，其中岛 1 已经接入互联网。现要在岛与岛之间连接电缆，让所有岛都能通过一些电缆直接或间接与岛 1 相连以接入互联网，并且电缆总长度最小（两个岛屿之间的电缆长度为这两个岛屿的直线距离）。现在，所有电缆同时按照该方案开始建造，每条电缆的建造速度都是 1 单位长度每单位时间，求每个人连上互联网的时间的平均值。

算法讨论

本题考查最小生成树。

我们首先在岛屿之间两两连边，边权为这两点之间的距离，那么，最优方案就是该图的最小生成树。接下来，岛i上的人接入互联网的时间就是岛i到岛 1 的路径上最长的电缆修建完成的时间，也就是最小生成树中岛i到岛 1 的路径上的最大边权。于是我们只要求出任意一棵最小生成树，并求出每个岛到岛 1 的路径上边权的最大值，按岛上的人数求加权平均数即为答案。最小生成树可能有多棵，但是由于最小生成树的性质，两点x,y之间的最大边权在任意两棵最小生成树上都相同，因此我们任取一棵计算不影响答案的正确性。

时间复杂度

$O(n^2)$

空间复杂度

根据实现的不同可做到$O(n)$或$O(n^2)$

2002 H - Silly Sort

题目大意

给定n个两两不同的整数$a_1 \dots a_n$，我们需要对它进行升序排序。在排序过程中，我们每次可以交换两个数的位置。每次交换代价是被交换的两个数的和。求最小代价。

算法讨论

本题考查贪心。

对于每个数我们算出它的目标位置b_i，并从i向b_i连边。我们发现这些边一定形成了一些环。对于每个环我们的目标是把每个数都放到环上相邻的位置。那么有如下两种决策：

1. 不断将环上最小的数和当前指向它的数交换，直到最小的数沿环绕了一圈到达了它的目标位置。

2. 先将全局最小的数与环上最小的数交换，再按 1 的方式交换，最后将原本环上最小的那个数与全局最小的数再次交换。

设环长为l，环上最小的数是ml，全局最小的数是mg，环上的数的和是sum，决策 1 的代价为$sum + ml \cdot (l - 2)$，决策 2 的代价为$sum + mg \cdot (l + 1) + ml$，只需简单地选择一种代价较小的决策即可。

时间复杂度

$O(n \log n)$

空间复杂度

$O(n)$

2003 B - Light Bulbs

题目大意

有一排n个灯泡，对应n个开关，使用第i个开关反转第$i-1,i,i+1$个灯泡的亮暗状态（第 1 个和第n个开关只对应两个灯泡）。给定灯泡的初始状态和目标状态，求最少使用开关的个数，以及字典序最小的使用方案。

算法讨论

本题考查模拟、数学。

注意到我们枚举第 1 个开关是否使用之后，能控制第 1 个灯泡的开关只有第 2 个开关了，那么第 2 个开关是否使用就确定了，接下来只有开关 3 能控制灯泡 2，于是开关 3 是否使用也可以确定下来……最后我们能确定每个开关是否使用，检查第n个开关现在的状态是否与目标状态相符即可判断这种方案可不可行。我们在这两种可能的方案里面找出可行的方案并取使用开关数最少的方案输出即可。

时间复杂度

$O(n)$，但十进制与二进制互化需要$O(n^2)$时间。

空间复杂度

$O(n)$

2003 F - Combining Images

题目大意

有一种基于四分树的黑白图像编码方法如下：首先检查当前图像是否为全黑或全白，若为全黑，则在当前编码串末尾添上 11，若为全白，则添上 10，否则，首先在当前编码串末尾添上一个 0，再将当前图像平均分成左上、右上、左下、右下四个子正方形，并依次递归地进行编码。现给定两个大小相同的正方形图像的编码，求它们的交的编码。所谓两个图像A, B的交C是指，对于C的任意一个像素点$C_{x,y}$，$C_{x,y}$为黑色当且仅当$A_{x,y}, B_{x,y}$都为黑色。

算法讨论

本题考查模拟。

首先我们定义该四分树的存储结构。一个四分树的节点应该包含五个域：一个整数col记录当前节点的颜色是黑色、白色或黑白相间，四个指针ul, ur, dl, dr指向其四个孩子的节点。我们先根据读入的两个编码分别建出对应的四分树。接下来我们设计两个四分树合并的算法。设要合并的两个四分树的根为x, y，那么如果x和y都是黑白相间的，那么我们递归地合并x和y的四个子树，并返回一个黑白相间的点，其四个子树为x和y的对应子树合并的结果，需要注意的是若我们发现合并出来的四个子树都是纯白色的，那么我们应该直接返回一个白色节点；否则，不妨设纯色的点是x，当x是黑色时，我们直接返回y，否则直接返回一个白色的点。这样一来我们就得到了代表两个图像的交的四分树，接下来按照题目要求的编码方式输出它即可。

时间复杂度

$O(n)$，其中n表示输入串和输出串的总长。

空间复杂度

$O(n)$，其中n表示输入串和输出串的总长。

2003 H - A Spy in the Metro

题目大意

在一个有n个地铁的地铁站，地铁在地铁站i和地铁站$i+1$之间运行需要t_i单位的时间。有ns班地铁分别在$s_1, s_2, \dots s_{ns}$时刻从地铁站 1 向地铁站n开出；有ne班地铁分别在$e_1, e_2, \dots, e_{ne}$时刻从地铁站n向地铁站 1 开出。现在是时刻 0，你在地铁站 1，你的目标是在T时刻到达地铁站n，你需要适当选乘地铁来使你在站台上等待的时间最小（在T时刻之前到达也是允许的，但到达时刻到T时刻的这段时间也计入在站台上等待的时间），问是否可能完成目标，若能，求最小等待时间。

算法讨论

本题考查动态规划。

设$l_{i,j}$表示i时刻在地铁站j是否有开往地铁站 1 方向的地铁，$r_{i,j}$表示i时刻在地铁站j是否有开往地铁站n点方向的地铁，这可以通过给定的地铁时刻表很方便地求出。设$f[i][j]$表示i时刻时你在地铁站j的情况下，你已经在站台上等待的时间的最小值。若在i时刻无法到达地铁站j，则$f[i][j] = inf$。初始时$f[0][1] = 0$。考虑主动转移，对于某个状态$f[i][j] < inf$，若$i + 1 \le T$则可以尝试用$f[i][j] + 1$更新$f[i+1][j]$，若$l_{i,j} = true$并且$i + t_{i-1} \le T$，则可以尝试用$f[i][j]$更新$f[i + t_{i-1}][j-1]$，若$r_{i,j} = true$并且$i + t_i \le T$，则可以尝试用$f[i][j]$更新$f[i + t_i][j+1]$。最终若$f[T][n] < inf$，则答案为$f[T][n]$，否则答案为 impossible。

时间复杂度

$O(nT + n(ns + ne))$

空间复杂度

$O(nT)$

2003 I - The Solar System

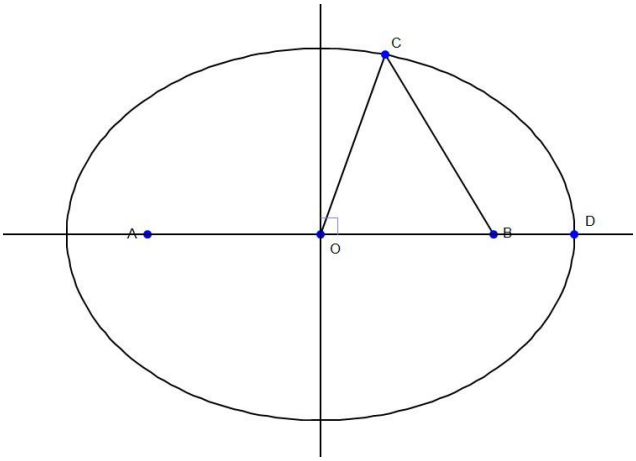
题目大意

给定太阳系中某行星 A 的运行轨道信息（半长轴、半短轴和周期），以及另一颗行星 B 的运行轨道半长轴和半短轴，求行星 B 从近日点开始，经过给定的时间运行到的位置。

算法讨论

本题考查数学。

设 B 行星的运动轨道的参数方程为$\begin{cases}x = a\cos\theta \\ y = b\sin\theta\end{cases}$，如下图所示，则其在$x$轴非负一侧的焦点 B 的坐标为$\left(\sqrt{a^2-b^2},0\right)$。设其中某一点 C 的坐标为$(a\cos\theta_1, b\sin\theta_1)$。



考虑区域 CBD 的面积S_{CBD}，容易计算$S_{CBD} = S_{COD} - S_{COB} = \frac{1}{2}ab\theta_1 - \frac{1}{2}\sqrt{a^2-b^2}\cdot b\sin\theta_1$。

再观察题目给定的条件，我们可以计算出行星 B 的运行周期$T_2 = t1\sqrt{\left(\frac{a_2}{a_1}\right)^3}$，进而问题转化为解方程$t\frac{S_2}{T_2} = S_{CBD}$，即求解关于$\theta_1$的方程 $t\frac{\pi a_2 b_2}{t1\sqrt{\left(\frac{a_2}{a_1}\right)^3}} = \frac{1}{2}a_2 b_2\theta_1 - \frac{1}{2}\sqrt{{a_2}^2-{b_2}^2}\cdot b_2\sin\theta_1$，该方程右边关于$\theta_1$单调递增，因此我们二分$\theta_1$即可。最后我们可以根据$(a\cos\theta_1, b\sin\theta_1)$算出答案。

时间复杂度

$O\left(\log\frac{ans}{eps}\right)$

空间复杂度

$O(1)$

2003 J - Toll

题目大意

给定一个n个点m条边的图，每个点有可能是村庄或是城镇，你要在这张图上运输货物。进入某个村庄时，你需要支付 1 个单位的货物作为过路费；进入一个城市时，设你携带了p个单位的货物，那么你需要支付$\left\lceil \frac{p}{20} \right\rceil$个单位的货物作为过路费。给定起点$S$和终点$T$，你需向终点运送至少$t$单位的货物，问在起点上至少需要多少货物。

算法讨论

本题考查最短路。

考虑相邻的两个点i,j，你正要从i向j运送p单位的货物，那么不难得到你在i点需要的货物数量为$\begin{cases} p+1, & j\text{为村庄} \\ \left\lceil \frac{20}{19}p \right\rceil, & j\text{为城市} \end{cases}$，令其为$cost(j,p)$。

设$d[i]$表示从i点运货到终点，使得终点至少有t单位货物时，i点至少需要多少货物。于是不难得到$d[T]=t$，转移方程为：

$$d[i]=\min\{cost(j,d[j]) \mid 1\le j\le n\}$$

注意到该方程是有后效性的，因此我们需要选择一种最短路算法来计算$d[i]$。最终答案即为$d[S]$。注意到“边权”非负，我们可以选择 Dijkstra 算法作为本题中的最短路算法。

时间复杂度

$O(n^2)$

空间复杂度

$O(n^2)$

2004 E - Intersecting Dates

题目大意

给定n个日期区间，这些区间所包含的全部日期组成了日期集合A，以同样的方式给定另外m个区间，它们组成了日期集合B，求包含B但不包含于A的区间集合，并以不相交且不连续的区间形式输出（输出的区间按起始日期从小到大排序，相邻两个区间不能相交或连续）。

算法讨论

本题考查模拟。

首先我们对日期编号，比如我们可以计算 1700/1/1 到某个日期中间经过的天数作为日期的编号。这样我们适当预处理一些结果之后就可以在$O(1)$时间内计算出任意日期的编号，也能通过编号反推出日期。这样编号的好处是相邻的日期在编号上是连续的，于是问题就转化为了整数区间上的问题。我们将这$n+m$个区间的左右端点放在一起排序之后即可得到$n+m$个关键点，在相邻两个关键点之间的数都具有同一种状态（是否存在于A集合，是否存在于B集合）。接下来我们只要从左到右扫描关键点并判断相邻关键点之间数的状态，如果某一段数存在于B集合但不存在于A集合，那么我们就输出这个区间（注意合并连续的区间，特判没有元素的区间和只有一个元素的区间）。

时间复杂度

$O(TotalDates) - O\big((n+m)\log(n+m)\big)$，其中$TotalDates$表示题目允许的日期范围内包含的总天数。

空间复杂度

$O(TotalDates+n+m)$，其中$TotalDates$表示题目允许的日期范围内包含的总天数。

2004 H - Tree-Lined Streets

题目大意

平面上有n条线段，代表n条路。线段与线段的交点代表十字路口。现在要在路边种树，要求如下：

- 同一条路上相邻两棵树的距离不小于$50m$
- 任意一棵树距离十字路口的距离不得小于$25m$

求最多能种下多少树。

算法讨论

本题考查计算几何、模拟。

不难发现两条路上的树互不影响，因此我们分别考虑每条路。我们求出其他路与这条路的交点，并把这些交点按照与路的起点间的距离排序，这样这些交点就把整条路分成了许多段，不难发现段与段之间也是互不影响的。若整条路只有一段，那么这条路上最多能种$\left\lfloor\frac{len}{50}\right\rfloor+1$棵树；若这条路被分为至少两段，那么第一段和最后一段至多能种$\left\lfloor\frac{len+25}{50}\right\rfloor$棵树，中间每一段至多能种$\left\lfloor\frac{len}{50}\right\rfloor$棵树。之后我们要简单的累加每条路的答案即可。

时间复杂度

$O(n^2 \log n)$

空间复杂度

$O(n)$

2004 I - Suspense!

题目大意

两栋建筑分别高n层和m层，间距为d，每一层住着小猫或者小鸟，并且猫和鸟都站在窗台上，现在要在两栋楼之间建一座桥。猫可以至多向上跳 0.5m 或向下跳至多 3m，这样，建桥之后猫就有可能通过桥抓到鸟。我们要防止这种情况的发生。桥的位置是由悬挂桥的绳子确定的。绳子的两段分别挂在两栋建筑最高层窗户的下沿，绳子的长度由你决定，绳子自然下垂呈抛物线的形状，桥的高度比绳子的最低点低 1m。楼的层高是 3m，每层的窗户下沿比该层地面高 1m。另外桥的高度还需要满足比两栋楼最高点窗户下沿至少低 2m，比地面至少高 1m。求最大可能的绳长。

算法讨论

本题考查贪心、数学。

我们首先分析哪些高度的桥面能够被猫达到。设某只猫的高度为h，那么它能达到的高度区间就是$(h-3, h+0.5)$，我们对于所有这样的区间求并，即可算出猫能达到的高度集合，记为C。接下来我们分析在在哪些高度的桥上的猫会抓到鸟。设某只鸟的高度为h，那么可能抓到它的桥的高度的区间就是$(h-0.5, h+3)$，我们同样对所有这些区间求并，记为B。这样一来所有不可行的桥的高度集合就是$A = B \cap C$。我们要求绳长最长，也就是要求桥的高度最低，因此我们找满足比两栋楼最高点窗户下沿至少低 2m，比地面至少高 1m 的不属于集合A的最小高度即可。

假设求出的桥面高度为$h-1$，那么对应抛物线的最低点高度就为h，设抛物线左右两悬挂点的高度分别为h_1+h, h_2+h，以该最低点为原点建立坐标系，设抛物线方程为$y=ax^2$，两悬挂点坐标为$(x_1, h_1), (x_2, h_2)$，不难得到$x1 = -\sqrt{\frac{h_1}{a}}, x_2 = \sqrt{\frac{h_2}{a}}$，根据$x_2 - x_1 = d$，解方程可知$a = \left(\frac{\sqrt{h_1}+\sqrt{h_2}}{d}\right)^2$。

对于抛物线$y=ax^2$，我们计算它$[0, x_0]$这一段的长度

$$l(x_0) = \int_0^{x_0} \sqrt{1+(2ax)^2} \cdot dx = \frac{1}{2}x_0\sqrt{1+(2ax_0)^2} + \frac{1}{4a}\sinh^{-1}(2ax_0)$$

因此答案即为$l\left(\sqrt{\frac{h_1}{a}}\right) + l\left(\sqrt{\frac{h_2}{a}}\right)$。

时间复杂度

$O(n+m)$

空间复杂度

$O(n+m)$

2005 C - The Traveling Judges Problem

题目大意

给定一张nc个点的带权无向图，求一个子图满足：

1. 给定的nj个点都能到达某个给定的点dc。
2. 在满足 1 的前提下，最小化边权和；若边权和相等，则最小化子图的点数；若点数也相等，则最小化选择的点的字典序。

输出最小边权和，以及每个点到dc的路径。

算法讨论

本题考查斯坦纳树与最小生成树。

不难看出该问题就是求包含dc以及另外给定的nj个点的最小斯坦纳树。本题与经典问题不同的地方在于，在满足边权和最小的情况下还需要保证点数最少，字典序最小。

我们可以通过扩展权值的定义来解决这个问题。我们用一个二元组(val, sta)来表示一棵树的权值，其中val表示其边权和，sta是一个二进制数，表示每个节点是否存在于这棵树中。两个这样的权值的比较方法是先比较val，若相等再比较sta中 1 的个数，若还相等则比较sta所代表的点集的字典序，通过一些预处理或 builtin 函数，这个比较可以在$O(1)$时间完成。定义两个权值的加法为$(val_1, sta_1) + (val_2, sta_2) = (val_1 + val_2, sta_1 \vee sta_2)$。这样一来一条边的权值就变为了$(\text{该边的长度}, \text{该边两端点对应的二进制数})$。我们直接根据这些定义求最小斯坦纳树即可。

最后我们需要构造建树方案。一种方法是 DP 时记录每个点是从哪里转移的，最后反向推出建树方案；另一种更为简洁的方法是，设最小斯坦纳树的权值为(val, sta)，那么我们直接求点集sta的最小生成树即为一种可行的建树方案。

时间复杂度

$O(3^{nj} \cdot nc^2)$

空间复杂度

$O(3^{nj} \cdot nc)$

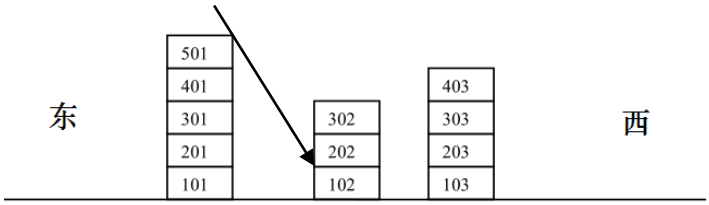
2005 E - Lots of Sunlight

题目大意

已知每间公寓的宽为w，高为h，有n幢公寓楼从东向西排列，第i幢有c_i层。某一天，太阳在5:37:00从东边地平线升起，以恒定角速度扫过天空，直到18:17:00在西边地平线落下。若某个时刻一间公寓的东面墙或西面墙被阳光完整地照射，或太阳正好垂直于地面，则我们称此时该时刻该公寓被太阳直射。现在询问一些公寓被直射的时间段。

算法讨论

本题考查数学、模拟。



由于每个公寓被直射的时段都是一个连续的区间，因此我们只需关注开始时间与结束时间，因此我们的目标就是找出能直射该公寓的第一缕阳光与最后一缕阳光。首先考虑第一缕阳光，不难发现它一定是经过某幢楼的右上角并射到目标公寓左侧墙面的最低点上的光线（图中的箭头画出了直射公寓 202 的第一缕阳光），那么我们直接枚举它经过所求公寓左边的哪一幢楼，并计算光线与底面的夹角，取最大值就能得到第一缕阳光与底面的夹角。最后一缕阳光的求解方法与第一缕阳光类似。

得到角度之后，由于太阳高度角变化的角速度恒定，我们可以根据日出日落时间求出太阳的角速度。这样我们便可根据之前求出的第一缕阳光与最后一缕阳光与地面的夹角确定对应的时刻。

时间复杂度

$O(Qn)$

空间复杂度

$O(Qn)$

2005 G - Tiling the Plane

题目大意

给定一个每个角都是直角的多边形，问是否能通过它不断复制、平移（不能旋转或翻转）来不重不漏的覆盖无限的二维平面。

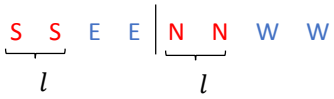
算法讨论

本题考查枚举和字符串匹配。

注意到题面上提供的有关“棋盘覆盖”和“蜂巢覆盖”的信息，我们不难把此题转化成一道字符串匹配题。

首先，我们将原描述串展开（将一个字母重复多次的表达方式展开为字符串序列，如“S 2 E 2 N 2 W 2”展开得到“SSEENNWW”），设展开串的长度为n，在之后的说明中，我们都将使用展开串来描述多边形或其边界上的一段。

假设我们现在检测该字符串是否满足“棋盘覆盖”，即检测是否能依次找到分割点 A,B,C,D，使得 A→B 与 D→C 匹配，B→C 与 A→D 匹配。注意到需要匹配的两个串方向相反，因此我们需要将其中一串倒置并将其中的每个方向替换为相反的方向（N 换为 S，S 换为 N，E 换为 W，W 换为 E）再进行匹配。假如我们现在检测“SENE”与“WSWN”是否匹配，我们先把“WSWN”反向得到“NWSW”，再将每个方向反向得到“SENE”，我们发现这与另一匹配串“SENE”是相等的，则说明“SENE”与“WSWN”是匹配的。接下来就是枚举并检查匹配。暴力枚举 A,B,C,D 点并暴力匹配的时间复杂度达到了$O(n^5)$，这是我们所不能接受的。但稍作考虑我们即可发现，A→C 和 C→A 一定是长度相等的两个串，并且 A→B 与 D→C 长度相等，B→C 与 A→D 长度相等。这样我们在枚举时可以先枚举将序列分割成长度相等的两段的分割方案，这样便确定了 A、C 的位置，再枚举 A→B（D→C）的长度l，就能确定 B、D 的位置。这样就把枚举量减小到$O(n^2)$，再乘上匹配所需的$O(n)$，总时间复杂度降为了$O(n^3)$。



图：枚举分割方案后枚举 A→B 串长度

图中两个红色部分互相匹配，两个蓝色部分互相匹配，该多边形可以铺满平面

我们仍然可以继续优化这个算法。设原串为 A，我们先处理出原串整个串倒置并将每个方向反向的字符串 B，这样检查原串中两段是否能匹配的问题就转化为了检查 A 的某个子串和 B 的

某个子串是否相等的问题了。我们预处理$f(i,j)$表示从 A 的第i位开始，B 的第j位开始的最长匹配长度，即$\mathrm{A}[i \ldots n]$与$\mathrm{B}[j \ldots n]$的最长公共前缀，它可以利用以下递推式在$\mathrm{O}(n^2)$时间内求出：

$$f(i,j)=\begin{cases} f(i+1,j+1)+1, & \mathrm{A}[i]=B[j] \\ 0, & \mathrm{A}[i]\neq B[j] \end{cases}$$

这样我们就可以在$\mathrm{O}(1)$时间内检查 A、B 的两个子串是否相等，检测该字符串是否满足“棋盘覆盖”便能够在$\mathrm{O}(n^2)$时间内完成。

接下来考虑“蜂巢覆盖”的情况。这种情况与“棋盘覆盖”的情况实质上没有太大区别。我们同样首先枚举将序列分割成等长两段的分割方案，即确定了 A、D 的位置、之后我们发现，它与棋盘覆盖唯一的不同只是有 A→B 与 E→D、B→C 与 F→E、C→D 与 A→F 三段需要匹配。我们先枚举 A→B 的长度$l1$，即确定了 B、E 的位置，再枚举 B→C 的长度$l2$，即确定了 C、F 的位置，最后用与“棋盘覆盖”同样的方法验证是否匹配即可。这样总枚举量是$\mathrm{O}(n^3)$，总时间复杂度也是$\mathrm{O}(n^3)$。由于大多数情况下，在枚举完$l1$之后已经发现不匹配，不会再进行对$l2$的枚举，因此这个算法的实际效果较为不错。

有没有与周长无关只与顶点数有关的算法呢？答案是肯定的。我们发现顶点的本质是展开序列中，字母发生变化的位置，也就是说一个顶点到下一个顶点之后的一系列字母都是相同的。注意到我们枚举$l1$实际上就是枚举 A 的对应点 E 的位置，因此我们可以只枚举 E 之后第一个顶点的位置，由于 E 到该顶点之间的字母都是相同的，我们只要看 A 之后有多少与之相同的字母即可确定 E 的位置。枚举$l2$时同理。设多边形的顶点数为m，这样对于每个分割方案的枚举量就减小为$\mathrm{O}(m^2)$。而枚举分割方案时，容易发现分割方案一定过某个顶点，这样分割方案数为$\mathrm{O}(m)$，总枚举量便降为$\mathrm{O}(m^3)$。如果我们预处理匹配结果，使检查匹配能在$\mathrm{O}(1)$时间内完成的话，整个算法的时间复杂度就为$\mathrm{O}(m^3)$。

时间复杂度

$\mathrm{O}(m^3)$，其中m表示顶点数。

空间复杂度

$\mathrm{O}(m^2)$，其中m表示顶点数。

2005 H - The Great Wall Game

题目大意

在一个$n \times n$的棋盘上有n个棋子，位置分别为(x_i, y_i)。你每次可以将其中一个棋子向上下左右中的一个方向移动一格，问将棋子排成一行、一列或一对角线所需的最小移动次数。

算法讨论

本题考查贪心、二分图带权权匹配。

首先考虑移动到一行或一列的情况，不妨考虑一行的情况。不难发现在这种情况下，横向和纵向的移动是无关的——横向上只要把棋子移动到不同的列上，纵向上只要把棋子全部移到同一行上即可。对于横向移动，我们对横坐标排序，设得到的的序列为$x_1, x_2, \dots, x_n$，那么把x_1移到 1 上，x_2移到 2 上……x_n移到n上的代价一定最小；纵向移动则是经典问题，可以证明将棋子移动到纵坐标的中位数的代价一定最小（若n为偶数，则可以移动到排在中间的两个数之间的任意位置）。

接下来考虑移动到一个对角线的情况，不妨考虑移动到$(1,1), (2,2), \dots, (n,n)$。建立二分图，左部$n$个点代表$n$个棋子的初始位置，右部$n$个点代表目标位置，我们在所有初始位置和所有目标位置之间两两连边，权值为这两点的曼哈顿距离，那么最小移动次数就是该二分图的最小权匹配，若使用 KM 算法计算，时间复杂度为$O(n^3)$。

时间复杂度

$O(n^3)$

空间复杂度

$O(n^2)$

2005 I - Workshops

题目大意

有n个会议，每个会议人数为a_i，需要的时间为b_i；有m个房间，每个房间可容纳的人数为c_i，最长可使用时间为d_i。现在要把这些会议安排到这些房间里面，可能会发生有些会议无法被安排的情况，要求最小化不能安排的会议数，在会议数相同的情况下，最小化不能安排会议的人数和。

算法讨论

本题考查贪心。

我们按人数从大到小考虑每个会议。每次我们从当前可选的房间中选择可使用时间最短的房间分配给这个会议，若当前无房间可选，则将这个会议视为不能安排的会议。容易证明此贪心的正确性。

实现时，我们可以将可容纳人数大于当前考虑的会议的房间放入一个 multiset 中，按可用时间排序，这样每次查找、删除的复杂度都是$O(\log m)$，总时间复杂度为$O\big((n+m)\log m\big)$

时间复杂度

$O\big((n+m)\log m\big)$

空间复杂度

$O(n+m)$

2005 J - Zones

题目大意

有n个基站，第i个基站总共服务a_i个用户。两个基站服务的区域可能有重复，重复区域共有c个，第i个重复区域有p_i个用户，这个区域同时属于第$b_1, b_2 \dots b_{k_i}$个基站。现在要求删除一些基站，使得保留的m个基站服务人数最多（当然，被多个基站同时服务的人只能算一次）。若有多种选法，要求字典序最小。

算法讨论

本题考查枚举。

首先，根据题目给出的数据不难算出单由第i个基站服务的人数s_i。接下来我们枚举留下了哪些基站，那么不难计算留下的基站服务的人数就是它们单独服务的人数加上与它们有交的公共区域的人数。按题目要求取最优值即可。

时间复杂度

$O(2^n + C_n^m \cdot c)$

空间复杂度

$O(nc)$

2006 A - Low Cost Air Travel

题目大意

有n个城市，在这些城市之间有m条飞机航线，第i条航线依次经过城市$a_{i,1}, a_{i,2} \dots a_{i,c_i}$，其机票价格为$p_i$。乘坐飞机时，我们只能从某一条航线的起点城市上飞机，但可以从该航线经过的任意一个城市下飞机，注意，在哪个城市下飞机都需要支付全额机票。

现在有q个询问，每个询问是一个目标序列$t_1, t_2 \dots t_l$，表示你需要从t_1出发乘飞机依次访问城市$t_2 \dots t_l$（不一定要连续访问），求最小花费，以及最小花费下依次使用了哪些机票。

算法讨论

本题考查最短路。

我们对于每个询问分别处理。首先拆点，将一个城市i拆成l个点$d_{i,1}, d_{i,2}, \dots, d_{i,l}$，分别表示到达该点时已经访问了目标序列的前$1,2, \dots, l$个城市。对于某个点$d_{i,j}$，考虑所有起点为$i$的航线$k$，枚举在哪个城市下飞机，并计算沿途访问的城市能使目标序列的访问进度推进到哪里，设下飞机的城市为x，在x下飞机时目标序列的前y个城市已经被访问，则我们连边$d_{i,j} \to d_{x,y}$，权值为p_k。这样，最终答案即为$d_{t_1,1}$到$d_{t_l,l}$的最短路。设$C = \sum_i^m c_i$，则该图的总点数为nl，总边数为Cl，若使用二叉堆优化的 Dijkstra 作为最短路算法，则时间复杂度为$\mathrm{O}\big((nl + Cl)\log nl\big)$。

注意到$d_{i,j}$的后继$d_{x,y}$都满足$y \geq j$，因此该图是一个分层图。我们把所有j相同的点看作一层，在层内做最短路，层间 DP，可将时间复杂度降至$\mathrm{O}\big((nl + Cl)\log n\big)$

时间复杂度

$\mathrm{O}\big((nl + Cl)\log nl\big)$

空间复杂度

$\mathrm{O}(nl + C)$

2006 B - Remember the A La Mode!

题目大意

有n种 A 类物品，每种有a_i个，同时有m种 B 类物品，每种有b_i个。一些种类的 A 类物品和一些种类的 B 类物品可以配对，记$c_{i,j}$为一个第i种 A 类物品与一个第j种 B 类物品配对的代价，若$c_{i,j}=-1$则表示这两种物品不能配对。先要求把所有的 A 类和 B 类物品两两配对，求总代价的最大值和最小值。

算法讨论

本题考查费用流。

我们把每种物品都看成点，并设源点S和汇点T。设A_i表示第i种 A 类物品对应的点，B_i表示第i种 B 类物品对应的点。对于每个A_i，我们建边$S \to A_i$，容量为a_i，费用为0；对于每个B_i，我们建边$B_i \to T$，容量为b_i，费用为 0；对于每个$c_{i,j} \neq -1$，我们建边$A_i \to B_j$，容量$+\infty$，费用$c_{i,j}$。此时，该网络的一个可行的最大流就代表了一种可行的配对方案，其中每条边$A_i \to B_j$的流量代表有多少个第i种 A 类物品与第j种 B 类物品配对。因此该网络的最小费用最大流即为总代价的最小值，最大费用最大流即为总代价的最大值。

此题如果使用实数费用，需要注意比较时的精度问题。

时间复杂度

$O(CostFlow(n+m,nm))$

空间复杂度

$O(nm)$

2006 D - Bipartite Numbers

题目大意

我们把由n个a和m个b连接而成的数叫做二段数，其中$n,m,a>0,b\geq 0,a\neq b$。现给定正整数x，求最小的二段数y，使得y是x的倍数，且$x\neq y$。

算法讨论

本题考查枚举。

设$f(l)=\underbrace{111\ldots1}_{l个1}$，那么一个二段数就可以表示为

$$i\cdot f(l_1)-j\cdot f(l_2)\ (1\leq i,j\leq 9,i-j\geq 0,l_1>l_2)$$

或

$$i\cdot f(l_1)+j\cdot f(l_2)\ (1\leq i,j\leq 9,i+j\leq 9,l_1>l_2)$$

由于$f(l)\ mod\ x$一定存在长度不超过x的循环节，所以可以证明若答案存在则其位数不会超过$2x$。我们考虑枚举l_1和i，设$g(p)=\{j\mid$存在l满足$l<l_1,1\leq j\leq 9$且$j\cdot f(l)\equiv p\ (mod\ x)\}$，这可以在枚举的过程中很方便地维护。于是对于第一种二段数的验证，我们只需检查$g(i\cdot f(l_1))$与$[1,i]$是否有交集即可，对于第二种二段数的验证，我们只需检查$g(-i\cdot f(l_1))$与$[1,9-i]$是否有交集即可。如果我们用位二进制压位来实现$g(p)$集合的话，就可以在每次$O(1)$时间内完成修改和查询。设数字种类数为B（在此问题中$B=10$），那么枚举验证的时间复杂度就为$O(Bx)$。此时我们只能确定对于当前l_1和i，存在一个解，但并不能知道解具体是什么，因此我们需要再枚举l_2和j来确定具体答案是什么。在枚举时注意枚举顺序，就可以保证第一次枚举到的一定是最小解。另外，如果x本身就是一个二段数，那么第一个枚举到的解会是它本身，然而这个解是不符合题目要求的，因此在这种情况下我们需要输出次小解。

时间复杂度

$O(Bx)$，其中B表示数字种类数，在此问题中$B=10$。

空间复杂度

$O(x)$

2006 E - Bit Compressor

题目大意

有一种压缩数据的方法如下：将二进制数据中每一段连续的k个 1（这一段连续的 1 左右必须都是 0 或没有东西）替换成k的二进制表示（只有当替换之后串长减小才进行替换）。现给定压缩后的串，问是否存在一个长度为l，1 的个数为c的原串，若存在，问原串是否唯一。

算法讨论

本题考查搜索。

由于本题数据范围较小，因此合法的展开方案不是很多。因此我们可以通过暴力搜索来枚举所有的展开方案并判断是否满足要求。搜索时注意以下几点：

- 若要展开一个子串，那么这个子串的前后必须是 0 或者没有字符。
- 若有 3 个及以上的 1 连续出现，则必须展开。
- 若当前已经找到两种方案，则可以停止搜索。
- 若当前长度或 1 的个数已经超过目标，则剪枝。

时间复杂度

$O\left(2^{\frac{n}{2}}\right)$，其中$n$表示压缩串的长度，实际复杂度远达不到此上界。

空间复杂度

$O(n)$

2006 G - Pilgrimage

题目大意

有一个旅行团队，他们在旅行时所有人的钱全部由一个人统一管理。有一本旅行日志，记录了他们这次旅行某一天整个团队的钱的变化情况。有下面四种记录：

- **IN** k：团队中新加入了k个人。此时新加入的这些人每人都会交钱，使得他们加入前后，团队中平均每个人持有的钱数不变。
- **OUT** k：k个人离开了团队。离开的人每个人都会分到一些钱，使得他们离开前后，团队中平均每个人持有的钱数不变。
- **COLLECT** k：平均每个人的钱数增加了k。
- **PAY** k：整个团队产生了k的开销。

然而，神奇的是，在这一天旅行过程中，在所有的“**IN** k”或“**OUT** k”操作发生时，团队平均每个人持有的钱数都是整数。求这一天刚开始时这个团队可能有多少人。

算法讨论

本题考察模拟、数学。

容易发现开始时可能的人数只受制于“**人数变化发生时，总钱数是人数的倍数**”以及“**任意时刻团队中至少有一个人**”，我们进一步分析那些记录来探究这两个条件是如何限制总人数的。

由于这一天刚开始的时候团队持有的总钱数可以是任意值，因此不必担心 **PAY** 操作时钱不够用的问题，因此所有的 **COLLECT** 操作都是可以忽略的。同时，第一个人数变化操作前的总钱数一定可以是人数的倍数，也就是说第一次人数变化之前的所有操作都是可以忽略的。并且，在最后一次人数变化操作之后的任何操作一定是可行的，因此也可以忽略它们。接下来，两次相邻的 **PAY** 操作合并在一起并不会造成任何影响，因此我们可以合并所有相邻的 **PAY** 操作了。这样一来，我们的旅行日记上就只剩下了人数变化操作和 **PAY** 操作，并且 **PAY** 操作只可能单独出现在两次人数变化操作之间。

这时我们考虑剩下的 **PAY** k操作。因为 **PAY** k操作前是人数变化操作，因此 **PAY** 之前总钱数是总人数的倍数，**PAY** k操作之后也是人数变化操作，因此 **PAY** 之后的总钱数也是总人数的倍数，因此k是总人数的倍数。反过来说，此时可能的总人数一定是k的约数。根据这个 **PAY** 操作之前的人数变化操作，我们不难推出在这个 **PAY** 的限制下，开始时可能的人数集合。我们对于所有的 PAY 操作算出开始时可能的人数集合并取交集，即可得到满足条件“**人数变化发生时，总钱数是人数的倍数**”的可能的起始人数。

“**任意时刻团队中至少有一个人**”这个条件比较简单，我们只需从前往后扫一遍，计算出团队人数最少的时刻，此时团队人数至少为 1 人，然后向前推知团队的起始人数，即可得到起始人数的最小值，大于等于这个值的起始人数都会满足这个条件。

最后，同时满足以上两个条件的起始人数即为可能的起始人数。

时间复杂度

$O(n\sqrt{k})$

空间复杂度

$O(n+\sqrt{nk})$

2006 I - Degrees of Separation

题目大意

给定一个n个点m条边的无向图，每条边的边权都是 1，问该图是否连通，若连通，则求所有点对间最短路的最大值。

算法讨论

本题考查最最短路。

由于本题中边权都为 1，因此我们可以从每个点开始 BFS 求出该点到其他点的最短路，并在所有最短路中取最大值。当然，直接使用 Floyd 算法求出任意两点间的最短路也是可行的。

时间复杂度

$O(nm)$或$O(n^3)$

空间复杂度

$O(n+m)$或$O(n^2)$

2006 J - Routing

题目大意

给定一个n个点m条边的有向图，要求从中选出一个包含点 1 和点 2 的子图，使点 1 和点 2 在该子图中互相可达，求子图最小点数。

算法讨论

本题考查最短路。

令$d_{i,j}\ (i \neq j)$表示从点i到点j至少需要经过的点数（不包括点i,j），我们首先从每个点开始 BFS 一次即可求出d。设最优情况下点 1 到点 2 的路径如下：

$$1 \to A_1 \to A_2 \to \cdots \to A_{l1} \to 2$$

那么对应的点 2 到点 1 的路径一定是：

$$2 \to \mathrm{B}_1 \to \cdots \to B_i \to A_{s1} \to \cdots \to A_{e1} \to B_{i+1} \to \cdots \to B_j \to A_{sk} \to \cdots \to A_{ek} \to B_{j+1} \to \cdots \to 1$$

并且

$$e1 \geq s1 > e2 \geq s2 > \cdots > ek \geq sk$$

换句话说就是，从 2 到 1 的路径中有一段或多段与 1 到 2 的路径共用，并且这些段在两条路径中出现的次序完全相反。若不是完全相反的话，就说明这两条路径必定在同样两点u,v之间走了不同的路，那么若让它们改走相同的路答案一定更优，因此原路径一定不是最优解。如果我们将原图中的每条边反向建立原图的反图的话，那么原图中 2 到 1 的路径就是反图中 1 到 2 的路径，相应的上述性质就变为这些公共段必定以相同次序出现，但每一段内部的顺序相反。令$a_{i.j}$表示原图中走到位置i，反图中走到位置j，那么$a_{i,j}$可能的出边有：

- 设e为原图中i的一个后继，那么我们连边$a_{i,j} \to a_{e,j}$，权值为$equ(e,j)$，表示i向后扩展一步走到点e，其中$equ(e,j) = \begin{cases} 0, & e = j \\ 1, & e \neq j \end{cases}$。
- 设e为原图中j的一个后继，那么我们连边$a_{i,j} \to a_{i,e}$，权值为$equ(e,i)$，表示j向后扩展一步走到点e。
- 若$i \neq j$，则连边$a_{i,j} \to a_{j,i}$，权值为$d_{i,j}$，表示两条路径共用了一段。

这样一来，$a_{1,1}$到$a_{2,2}$的最短路即为答案。注意到该图共有n^2个点$\mathrm{O}(nm + n^2)$条边，直接使用堆优化的 Dijkstra 算法计算最短路的时间复杂度为$\mathrm{O}\big((nm + n^2)\log n\big)$。但是我们注意到答案

是不大于n的，因此在计算最短路的过程中涉及到的dis均不会超过n（dis超过n的状态可以直接忽略），于是我们可以使用在每种dis上挂链表的方法来实现优先队列，这样最短路的时间复杂度就降为了$O(nm+n^2)$。

时间复杂度

$O(nm+n^2)$

空间复杂度

$O(n^2+m)$

2007 A - Consanguine Calculations

题目大意

给定一个三口之家，已知父亲、母亲和孩子三人中某两个人的血型（ABO 型以及 RH 型），问第三个人可能的血型。

算法讨论

本题考查枚举。

血型是本质上是由三种显性基因是否存在决定的，也就是说一个人血液中存在的显性基因集合确定了这个人的血型。下表表示了每种血型对应的显性基因集合：

血型	**O-**	**O+**	**B-**	**B+**	**A-**	**A+**	**AB-**	**AB+**
显性基因集合	$\emptyset$	{+}	{B}	{B, +}	{A}	{A, +}	{A, B}	{A, B, +}

而父母血型与子女血型的关系是：若父母中存在 A，则子女中可能存在 A；若父母中存在 B，则子女中可能存在 B；若父母中存在+，则子女中可能存在+，特别地，若父母中有一方同时含有 A、B，则子女中至少存在 A 和 B 中的一个，若有一方不含有 A 和 B，则子女至多存在 A 和 B 中的一个。换句话说就是，子女血型是父母血型并集的子集，并且当父母中有人同时包含 A、B 时，子女至少会包含 A 和 B 中的一个，当父母中有人不包含 A 和 B 时，子女至多能包含 A 和 B 中的一个。因此我们只需枚举未知的那个人的血型并验证是否满足上述关系即可。

时间复杂度

O(1)

空间复杂度

O(1)

2007 B - Containers

题目大意

给你一个只包含大写字母的字符序列，你需要从左向右依次处理这些字符，每次你可以将正在处理的字符压入已有的某个栈或者新建一个栈将它压入，要求处理完全部字符之后，可以做到先弹出所有 A，再弹出所有 B，再弹出所有 C……依此类推。求最少需要几个栈。

算法讨论

本题考查贪心。

显然，要满足最后的条件，我们需要保证每个栈自顶向底是不降的，也就是说，如果一个栈已经压入了某个字符 x，那么之后它就只能压入小于等于 x 的字符了。于是，我们每次将正在处理的字符压入栈顶字符最小的可行栈（如果没有可行栈则新建）即可得到最优解。容易证明此贪心的正确性。

朴素的做法时间复杂度为$O(n \cdot ans)$，注意到所有的栈栈顶字符肯定是不同的，并且字符集大小只有 26，因此我们可以用一个二进制数表示存在哪些栈顶字符。我们可以通过一些位运算来支持寻找大于等于某个字符的最小字符、删除一个字符、插入一个字符，因此该算法可以在$O(n)$时间内实现。

时间复杂度

$O(n)$

空间复杂度

$O(n)$

2007 G - Network

题目大意

有n条信息，第i条信息的长度为len_i字节，它们被拆成m块按顺序送入缓存中，其中第i块包含了第no_i条信息的第s_i到e_i个字节，而缓存的作用就是将信息们按原来的顺序连续输出（同一条信息必须按顺序连续输出，但先输出哪条信息是任意的）。缓存是这样工作的：收到一个信息块时，缓存可以决定直接输出这个信息块或者存储这个信息块，被存储的信息块可以在一个由缓存决定的时间输出。存储一个信息块所需的空间与信息块的大小相同，问缓存至少需要有多大的存储空间。

算法讨论

本题考查枚举、贪心。

首先考虑只有一条信息的情况。我们对信息块按照它们应该被输出的次序排序，设第i个块的排名是$rank_i$。那么下面的贪心显然是正确的：设当前已经输出了排名前u的信息块，当信息块i进入时，若$rank_i = u + 1$，则将该信息块直接输出并将缓存中的块尽可能多的输出，否则将信息块i放入缓存。我们可以用一个布尔数组in_i表示排名为i的信息块当前是否在缓存中，这样一来上述算法就能够在$O(m)$的时间内实现。

多条信息的情况也不难处理。我们枚举信息的输出顺序，这样一来每个信息块输出的次序就确定了，于是我们直接套用上面的算法即可。总共有$n!$种情况需要枚举，因此这一部分的时间复杂度为$O(n!\,m)$，再加上预处理时对信息块进行排序的时间复杂度$O(m \log m)$，总时间复杂度为$O(n!\,m + m \log m)$。

时间复杂度

$O(n!\,m + m \log m)$

空间复杂度

$O(n + m)$

2007 I - Water Tanks

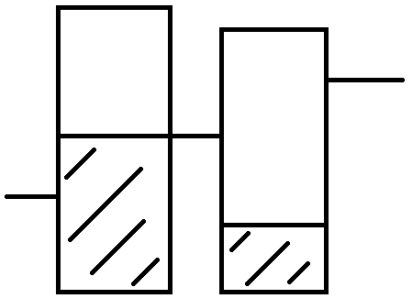
题目大意

给你n个底面积为 1 平方米的容器，高度分别为$h_1, h_2, \ldots, h_n$，第 1 个容器上方与大气连通，第 2 个到第n个容器密闭。在相邻两个容器之间有管子连接这两个容器，第i个容器与第$i+1$个容器之间的管子高度为hp_i，保证管子的高度递增。问若从第一个容器开口处缓慢注水，最多能够注入多少水。

算法讨论

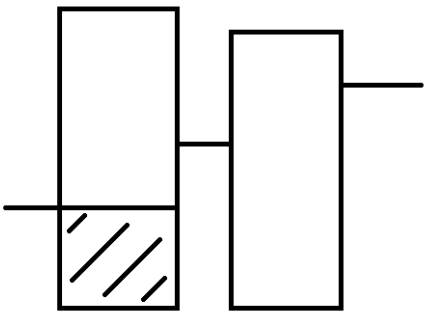
本题考查物理、数学。

我们不妨假设容器按$1 \ldots n$从左向右放置。若无特别说明，以下说明中长度均以m为单位，压强均以atm为单位。



一个重要的事实是：假设现在第i个容器已经注水到水面高度与其右侧的管子（与第$i+1$个容器相连的管子）相平，若继续注水，由于注水缓慢，新注入的水不会导致第i个容器的水面高度上涨，只会导致第$i+1$个容器的水面升高，如上图所示，直到第$i+1$个容器的水面与左边管子平齐。此时第i个容器上方的气体与第$i+1$个容器以及之后的容器中的气体是连通的。

根据这个事实，不难发现我们一定可以注水使得第 2 个容器的水面与其左边管子平齐。我们不妨从这个状态开始考虑。



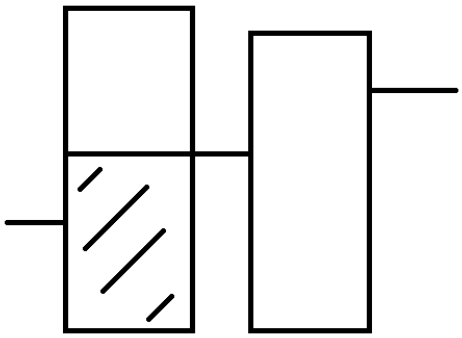
如上图所示，假设第i个容器水面已与左边管子相平，我们现在判断是否能使它上升至与右边管子相平。设此时第i个容器及其之后容器中气体的体积为V_r，压强为p_r，水面升至与右边管子平齐时气体的体积为V_n，则此时气体压强为$p_n = p_r \frac{V_r}{V_n}$，因此该容器底面压强$p_{bot} = p_n + p_{水} = p_r \frac{V_r}{V_n} + 0.097hp_i$。由于此时该容器内的水体与第 1 个容器中的水体是连通的，因此此时第 1 个容器底部的压强也应当为p_{bot}，又第一个管子底部最大可能压强$p_{max} = 0.097h_1 + 1$，故如果$p_{bot} \leq p_{max}$，则水面可以升至与右边管子平齐，否则水面无法升至与右边管子平齐。

若水面无法升至与右边管子平齐，那么第i个容器最终的水面高度h满足方程：

$$\frac{V_r p_r}{V_r + hp_{i-1} - h} + 0.097h = p_{max}$$

解出h即可。此时我们称注水工作在容器i结束。

若水面可以升至与右边管子平齐，则根据上面的事实，继续注入的水不会导致第i个容器的水面高度上涨，只会导致第$i+1$个容器的水面升高，因此我们接下来需要判断是否能使第$i+1$个容器的水面与其左边的管子平齐。



此时的情况如上图所示。同样设此时第i个容器及其之后容器中气体的体积为V_r，压强为p_r，第$i+1$个容器水面升至与左边管子平齐时气体的体积为V_n，则此时气体压强为$p_n = p_r \frac{V_r}{V_n}$，此时这两个容器底面压强均为$p_{bot} = p_n + p_{水} = p_r \frac{V_r}{V_n} + 0.097hp_i$。同样地，我们判断$p_{bot}$与$p_{max}$的大小关系，如果$p_{bot} \leq p_{max}$，则第$i+1$个容器的水面可以升至与左边管子平齐，否则水面无法升至与左边管子平齐。

若水面无法升至与左边管子平齐，那么第i个容器最终的水面高度为hp_i，第$i+1$个容器最终的水面高度h满足方程：

$$\frac{V_r p_r}{V_r - h} + 0.097hp_i = p_{max}$$

解出h即可。此时我们也称注水工作在容器i结束。

若水面可以升至与左边管子平齐，那么此时第i个容器上方的气体被密闭起来，我们记下其体积与压强的乘积为$pV_i = p_n(h_i - hp_i)$，然后继续对下一个容器进行上述两个判断。

最后我们需要算出总注水量。设注水工作是在容器i结束的，那么容器$i \ldots n$最后的水面高度要么为已经算出，要么为 0，而第 1 个容器的水面高度必定为h_i，因此我们需要计算容器$2 \ldots i-1$最终的水面高度。由于这些容器中的水体都与第 1 个容器中的水体连通，因此它们的底部压强均是p_{max}。那么对于容器j，其最终水面高度h满足方程：

$$\frac{pV_j}{h_j - h} + 0.097h = p_{max}$$

解出h即可。

最后，简单的将各容器水面高度相加即为总注水量。

时间复杂度

$O(n)$

空间复杂度

$O(n)$

2007 J - Tunnels

题目大意

给定一个$n+1$个点m条边的无向图，间谍要从 1 号点出发逃跑到 0 号点，你可以随时观察到间谍的逃跑路线并且任何时候都可以选择摧毁一些道路，使得间谍无法逃跑到 0 号点。问最坏情况下你最少需要摧毁几条道路。

算法讨论

本题考查最小割。

我们直接给出算法：

1. 设$d_i = MinCut(i, 0)$，其中$MinCut(S, T)$表示S到T的最小割。
2. 设当前没有被删除的点中，d_i的最小值为min，我们从图中删除所有满足$d_i = min$的点，并在此基础上用$min + MinCut(i, 0)$来更新剩下的d_i（此时的$MinCut(S, T)$亦指删点之后的最小割）。
3. 重复 2 过程直到图中所有点被删完。
4. 此时d_i表示若间谍从点i出发，要阻止其逃走的最小代价，因此d_1即为答案。

下面简要证明该算法的正确性。首先，要阻止间谍逃跑，我们的策略一定是不断摧毁某些隧道并引导间谍走向某一（某些）点上，然后摧毁切断该点所在连通块与外界的一切联系，而摧毁点i与点 0 的一切联系的最小代价就是$MinCut(i, 0)$。也就是说，无论如何决策，最后都要摧毁某个点到外界的一切联系，因此对于$MinCut(i, 0)$最小的一些点来说，即使走到其他点答案也不会比直接摧毁这个点与外界的联系优，因此对于这些点$d_i = MinCut(i, 0)$一定是成立的。接下来考虑其他的d_i。对于某个点i，一种显然正确的做法是，枚举允许从它走到的点的集合S，对于S内的点，我们让间谍走到其中的某个点之后再决策，这样带来的最大可能代价是$\max\{d_j | j \in S\}$（假设d_j已经正确的求出）；对于不在S内的点，我们切断间谍通过它们走到外界的一切路径，这样的代价等于删除了S的图中的$MinCut(i, 0)$。对于所有枚举到的S集合，我们用$\max\{d_j | j \in S\} + MinCut(i, 0)$来更新$d_i$，那么一定能得到正确的$d_i$。根据这个代价的式子不难发现，枚举所有集合是不必要的，因为一旦集合中某个d_j达到了$\max\{d_j | j \in S\}$，那么我们可以把所有d_j小于等于这个值的点j也选入集合而不会让解变差。因此我们在枚举集合的时候可以只枚举允许的最大值，然后将d_j不大于这个最大值的点j都加入S集合，并只用这些可能的S集合来更新d_i。这样一来我们回到上面的算法，当某个d_i被选为当前最小点时，它一定已经被所有小于它的d_j更新过了，这样一来就相当于枚举了允许的最大值然后将所有d_j小于它的点j都加入S集合并用这个S集合更新了d_i。因此若之前已经计算出的d_j都是正确的，那么此时的d_i也是正确

的。又因为前面已经证明，对于那些$MinCut(i,0)$最小的点$d_i = MinCut(i,0)$一定是成立的，根据数学归纳法，不难得到最后所有的d_i都是正确的。

时间复杂度

$O(n^2 \cdot MaxFlow(n,m))$

空间复杂度

$O(n+m)$

2008 E - Huffman Codes

题目大意

给定n个字符的 Huffman 编码，编码时两个节点合并时，要求字频小的作为左孩子，求这n个字符可能的词频分布（每个字符的字频为i%，i为整数且大于 0）。注意，两个仅仅顺序不同的字频分布（例如一个是 30%、70%，另一个是 70%、30%）算作同一种。

算法讨论

本题考查搜索。

由于 Huffman 树是每次选择两个字频最小的节点进行合并，不难得到以下两条性质：

- 同一层中节点的字频从左向右不降。
- 第i层节点的最小字频不小于第$i+1$层节点的最大字频。

同时也不难说明，满足这两条性质的树一定是一棵符合条件的 Huffman 树。因此我们可以按这两个条件来搜索字频分布。我们首先求出这可 Huffman 树的 BFS 序，并且 BFS 时我们每次先访问右孩子，这样上述两个性质就转化成 BFS 序中后一个节点的字频不大于前一个节点。根节点的字频是 100，我们从它开始搜索，每碰到一个有孩子的节点时枚举两个孩子的字频分配并注意只枚举满足性质的情况，即可通过此题。

时间复杂度

上界为$O(50^n)$，实际复杂度接近$O(ans)$

空间复杂度

$O(n)$

2008 F - Glenbow Museum

题目大意

一个只包含 90 度角和 270 度角的多边形的形状可以用它的角序列粗略地表示。一个包含L个角的多边形的角序列是一个由L个字母组成的字符串，这个字符串从多边形的某个角开始依次描述了多边形的L个角，其中字母R代表一个 90 度角，字母O代表一个 270 度角。一个满足要求的角序列必须能代表一个合法的多边形并且它所代表的多边形内部至少存在一个点能看到整个多边形的内部。求长度为L的满足要求的角序列的个数。

算法讨论

本题考查数学。

设$f(x) = RORO\ldots ROR\ (x个O，\ x+1\ 个R)$，那么不难发现将满足要求的角序列首尾相接形成一个环之后一定是$f(a)+f(b)+f(c)+f(d)$的形式。因为$f(x)$的长度可以取遍所有正奇数，所以满足要求的角序列的个数等于将一个长度为L的环分割成长度为奇数的四段的方案数（循环同构算多次），计算可得这个方案数为$\sum_{i=1}^{L-3} C^3_{\left\lfloor\frac{L-i-3}{2}\right\rfloor+3}$。继续化简可得：

$$\sum_{i=1}^{L-3} C^3_{\left\lfloor\frac{L-i-3}{2}\right\rfloor+3} = \sum_{i=3}^{\left\lfloor\frac{L+2}{2}\right\rfloor} C_i^3 + \sum_{i=3}^{\left\lfloor\frac{L+1}{2}\right\rfloor} C_i^3 = C^4_{\left\lfloor\frac{L+2}{2}\right\rfloor+1} + C^4_{\left\lfloor\frac{L+1}{2}\right\rfloor+1} = C^4_{\frac{L}{2}+2} + C^4_{\frac{L}{2}+1}$$

于是答案即为$C^4_{\frac{L}{2}+2} + C^4_{\frac{L}{2}+1}$。注意$L$为奇数时，答案应为 0。

时间复杂度

O(1)

空间复杂度

O(1)

2008 H - Painter

题目大意

平面上有n个三角形，问三角形是否两两没有公共点，若是，求三角形最大嵌套层数。

算法讨论

本题考查扫描线、平衡树。

首先我们对所有坐标进行变换$(x,y)\to(x+y\cdot eps)$以避免垂直线段的影响。判断平面上是否存在线段相交是一个经典问题，做法是扫描线+平衡树。具体来说，我们按x轴左到右扫描并用平衡树实时维护在当前扫描线上的线段的位置关系，不难说明线段在相交之前一定是在扫描线上相邻的，于是我们只需要在插入或删除线段时检查新产生的相邻线段是否相交即可。需要注意的是，对于本题来自同一个三角形的线段是允许相交的，需要特判。

在确定没有线段相交之后，我们就可以来确定嵌套层数了，做法依然是扫描线+平衡树。注意到若一个三角形在当前扫描线上，必定意味着该三角形有两条边在当前扫描线上，并且一上一下。我们把每个三角形上方的边看做左括号，下方的边看做右括号的话，当前扫描线上的线段就形成了一个括号序列，而一个三角形被嵌套的层数就是它对应的括号被嵌套的层数。这样我们就可以在每个三角形对应的一对括号被插入时确定该三角形被嵌套的层数。具体来说，我们寻找当前插入的三角形对应的左括号的前一个括号，若该括号也是左括号，那么说明新插入的括号被该括号嵌套，因此当前三角形被嵌套的层数等于该括号所属三角形被嵌套的层数+1；若该括号是右括号，那么说明新插入的括号与该括号平级（被嵌套于同一个括号中），因此因此当前三角形被嵌套的层数等于该括号所属三角形被嵌套的层数。这样扫描一遍之后我们就得到了每个三角形被嵌套的层数，简单地取最大值即为答案。为了方便实现，我们一开始可以把线段$(10^{-9},10^{9})\to(10^{9},10^{9})$和线段$(10^{-9},-10^{9})\to(10^{9},-10^{9})$加入平衡树作为画布。另外，由于判断相交和计算嵌套层数的算法具有高度相似性，实现时可以同时完成。

时间复杂度

$O(n\log n)$

空间复杂度

$O(n)$

2008 I - Password Suspects

题目大意

给定n个字符串，问长度为l并且包含了所有这n个字符串的字符串有多少个。如果个数不多于 42，则输出它们。

算法讨论

本题考查字符串匹配、AC 自动机、动态规划。

首先建立这n个字符串的 AC 自动机。设$f[i][j][S]$表示长度为i，在 AC 自动机上走到节点j，已经匹配的字符串集合是S的方案数。我们考虑主动转移，$f[i][j][S]$能够转移到的的状态是$f[i+1][son(j,ch)]\left[S \cup EndSet\left(son(j,ch)\right)\right]$，其中$ch$是任意小写字母，$son(j,ch)$表示 AC 自动机上$j$点添加一个字母$ch$之后到达的节点，$EndSet(x)$是一个集合，表示 AC 自动机上的节点$x$以及它在 Fail 树中的父亲是哪些字符串的结尾。设U是字符串的全集，$Nodes$是 AC 自动机的点集，于是最终答案就是$\sum_{j \in Nodes} f[l][j][U]$。

关于输出方案，我们可以从最终有用的状态向前推，每次向贡献了答案的上层节点走即可推出所有可行的字符串。

时间复杂度

$O(2^n \cdot l \cdot \Sigma \cdot |Nodes|)$，$\Sigma$表示字符集大小。

空间复杂度

$O(2^n \cdot l \cdot |Nodes|)$

2008 J - The Sky is the Limit

题目大意

给你n个底边在x轴上的等腰三角形，求它们的并的上轮廓线的长度。

算法讨论

本题考查计算几何。

我们把每个等腰三角形的上轮廓线拆成两条线段来处理，问题就转变成了求最上方的线段组成的折线的长度。我们先计算出所有可能出现线段相对位置发生变化的位置（线段出现、消失以及相交的位置），不难发现在这些“关键位置”之间，位置最高的线段一定是同一条。于是我们从左向右扫描，依次计算在每个“关键位置”上所有线段的高度，并选出最高的那一个，再将所有这些最高点相连，形成的折线长度即为所求的轮廓线长度。需要注意的是某些关键点之间没有任何线段的情况是不能被计入答案的。这样，关键点的个数最多能达到$O(n^2)$，每次取最大值需要$O(n)$的时间，因此该算法的时间复杂度为$O(n^3)$，已经可以通过所有数据，并且十分容易实现。

事实上我们还可以优化这个算法。我们在扫描时维护当前存在的线段在当前位置的排序结果，并用堆维护当前次序相邻的两条线段的相交时间，这样我们就可以通过堆中记录的信息查找下一个关键位置以及线段间次序的变化情况了。由于维护了线段的排序结果，我们可以在$O(1)$的时间内查询最大值，而从一个关键位置跳到下一个时，维护堆中的信息需要花费$O(\log n)$的时间，因此该算法的总时间复杂度为$O(n^2 \log n)$。前一个算法由于预处理了所有关键位置，空间复杂度为$O(n^2)$，而这个算法中关键位置是实时计算的，空间复杂度仅为$O(n)$。

时间复杂度

$O(n^3)$ 或 $O(n^2 \log n)$

空间复杂度

$O(n^2)$ 或 $O(n)$

2008 K - Steam Roller

题目大意

有一张$n \times m$的网格图，每个点与上下左右四个方向的点之间可能存在边，每条边的权值已知。走一条边的代价是这么计算的：如果你上一步走的边的方向与下一步走的边的方向都与这一步相同，则代价为这条边的权值，否则代价为这条边权值的两倍。你走的第一条边和最后一条边的代价均为那条边代价的两倍。你现在要从(x_s, y_S)走到(x_e, y_e)，求最小代价。

算法讨论

本题考查最短路。

我们把原图中的每个点(x, y)拆成一个能向四个方向走的点$a_{x,y,0}$和四个只能向特定方向走的点$a_{x,y,1} \dots a_{x,y,4}$，其中 1 表示只能向左走，2 表示向右，3 表示向上，4 表示向下，同时设$d_{x,y,z}$表示原图中(x, y)向z方向的边的权值。对于每个点(x, y)和方向z，令(x', y')表示(x, y)向z方向走到达的点，我们连边$(a_{x,y,0} \to a_{x',y',0}, 2d_{x,y,z})$、$(a_{x,y,0} \to a_{x',y',z}, 2d_{x,y,z})$、$(a_{x,y,z} \to a_{x',y',0}, 2d_{x,y,z})$、$(a_{x,y,z} \to a_{x',y',z}, d_{x,y,z})$，其中$(x \to y, z)$表示一条从$x$向$y$，长度为$z$的有向边。于是问题的答案即为新图上$a_{x_s,y_s,0}$到$a_{x_e,y_e,0}$的最短路。使用 Dijkstra 算法求解该最短路即可。

时间复杂度

$O(nm \log nm)$

空间复杂度

$O(nm)$

2009 A - A Careful Approach

题目大意

有n架飞机，第i架可以在$[a_i, b_i]$这段时间内起飞。要求安排每架飞机的起飞时间，使得最小的时间间隔最大，求这个时间间隔。

算法讨论

本题考查二分答案、状态压缩的动态规划。

这是一道典型的使最小值最大的问题，我们考虑二分答案。设当前考虑的答案为lim，也就是说，两架飞机的起飞间隔至少为lim，考虑如何判断是否存在一种安排方案满足该要求。我们用$f[S]$表示已经起飞了的飞机集合为S时，S中最后一架飞机的最早起飞时间，若为inf则表示没有这样的方案。不难得到如下转移方程：

$$f[S] = \min \begin{cases} \max(f[S-i] + lim,\ a_i), & i \in S \text{且} f[S-i] + lim \le b_i \\ inf \end{cases}$$

设U为飞机的全集，则我们只需判断$f[U]$是否小于inf即可。

此题最后要求四舍五入到秒，因此只需以 0.5 秒为单位进行整数二分即可。

时间复杂度

$O(2^n \cdot n \log ans)$

空间复杂度

$O(2^n)$

2009 B - My Bad

题目大意

给定由ni个输入，ng个逻辑门，no个输出组成的逻辑电路，其中有至多一个门坏掉了，坏掉的方式可能是该门只输出 0、只输出 1 或总与正确的输出相反。现给定nt个输入以及它们对应的输出，你要确定这些输出是否都是正确的，若不是，则确定哪个门以什么方式坏掉了。如果有多种坏掉的方式都能符合这些输入和输出，则输出无法确定。

算法讨论

本题考查枚举、模拟。

我们首先将给定的电路图进行拓扑排序，于是对于给定的某个输入我们只需按照拓扑序依次计算各个门的结果即可得到相应的输出。这样我们就可以检查某一个电路是否满足所有给定的输入输出了。

我们先检查原电路是否已经满足了所有输入输出，若不满足再枚举所有门可能的三种坏法并分别验证，若只有一种坏法满足所有输入输出则输出这种坏法，否则输出无法确定。虽然门的种类达到了 10 种（与、反向的与、或、反向的或、异或、反向的异或、非、反向的非、输出 0、输出 1），但每一种都不难处理。

时间复杂度

$O(ng^2 \cdot nt)$

空间复杂度

$O\big(nt(ni+no)\big)$

2009 F - Deer-Proof Fence

题目大意

平面上有n个点(x_i, y_i)，现在要造一个或多个围栏将每个点围住。围栏的要求是与其围住的任意一个点的距离不小于r，一个围栏可以围住一个点或多个点。求围栏总距离的最小值。

算法讨论

本题考查凸包、动态规划。

设点的全集为U，首先考虑将U的某个子集S用一个围栏围住的情况下该围栏长度的最小值。可以发现若$r = 0$，最优方案就是该点集的凸包；当$r > 0$时，我们只要将凸包的每条边都沿垂直于该边的方向向外移动r单位，并将转角处用圆心为转角处的对应顶点，半径为r的圆弧连接即可得到最优解。这样，设$C(S)$表示点集S的凸包的周长，那么用一个围栏围住这个点集的最优解即为$C(S) + 2\pi r$。

接下来的任务是将全集U划分成一个或多个子集，使得总代价最小。考虑动态规划，令$f[S]$表示使用一个或多个围栏围住点集S所需的最小围栏长度，那么不难得到转移方程：

$$f[S] = \min\begin{cases} C(S) + 2\pi r \\ f[T] + f[\complement_U S], \quad T \subset S, T \neq \emptyset \end{cases}$$

答案即为$f[U]$。

若对凸包周长的处理顺序稍加设计，则处理所有凸包周长的时间复杂度可做到$O(2^n \cdot n)$，动态规划的时间复杂度为$O(3^n)$，因此总时间复杂度为$O(2^n \cdot n + 3^n)$

时间复杂度

$O(2^n \cdot n + 3^n)$

空间复杂度

$O(2^n)$

2009 H - The Ministers' Major Mess

题目大意

有n个提案，m个议员，第i个议员对k_i $(1 \le k_i \le 4)$个议案投了票（支持或是反对）。现在要决定每个议案是否被通过，问是否存在方案使得每个议员的投票中有多于一半被满足，若存在这样的方案，则确定每个提案是否一定会通过或一定不会通过。

算法讨论

本题考查 2-SAT。

首先我们把每个议案拆成两个点，分别表示这个议案通不通过。考虑议员投票对点的选择的限制，此时第i个议员的投票可以表达表示为他建议选择$p_1, p_2, \dots, p_{k_i}$。因为每个议员的建议要满足一半以上，那么当$k_i \le 2$时，该议员的建议就必须全部满足，换句话说此时他的建议对点的限制是“**必须选择某些点**”；当$k_i > 2$时，该议员的建议最多不满足一个，设与$p_1, p_2, \dots, p_{k_i}$相对的分别是$p_1', p_2', \dots, p_{k_i}'$，那么也就是在$p_1', p_2', \dots, p_{k_i}'$中最多只能选择一个点，即$p_1', p_2', \dots, p_{k_i}'$**两两互相矛盾**。

这时我们就得到了一些形如**必须选择某些点**和**某些点对之间是矛盾的**这样的限制关系。对于前者，我们可以先将其确定下来然后在图中删除相关节点来处理，不过更简单的做法是直接将其转化为它的对应点与提案 1 的通过或不通过均矛盾。接下来就可以套用经典 2-SAT 问题的做法了。设点对i, j间有矛盾，它们的对应点分别为i', j'，则建边$i \to j', j \to i'$，这样一条边$i \to j$就表示如果选了i就必须选j。我们分别从每个点开始推导出其他相关的组，并检查是否有冲突（某一个议案的两个点同时被选中）。之后，如果存在某个议案，从它通过和不通过的两个节点开始推导的结果都是存在冲突则表示 2-SAT 无解，即无法找到满足条件的方案，否则问题有解。在有解的情况下，对于每个议案，若它的两个节点都不能推导出冲突，则不能确定这个议案是否通过，否则没有冲突的那个点所代表的意义即为这个议案的决策。

时间复杂度

$O(nm)$

空间复杂度

$O(n + m)$

2009 I - Struts and Springs

题目大意

给出平面上一些不相交的矩形，两两不相交（要么包含，要么相离），保证有一个大矩形包含了所有矩形。矩形的的长宽和位置由硬杆（定长）或弹簧（变长）确定。更具体的，对于不是最外层矩形的任意一个矩形，它的长、宽、上边距、下边距、左边距以及右边距都分别由一根硬杆或弹簧确定。

对于某个矩形来说，当直接包含它的矩形被改变的时候，它会根据规则“硬杆长度不变，弹簧长度等比例放缩”随之变化。

每次修改最外层大矩形的长宽，求修改后每个矩形的长宽和位置。

算法讨论

本题考查模拟。

如果矩形 A 包含矩形 B，并且不存在矩形 C 使得 A 包含 C，C 包含 B，那么我们称 A 是 B 的父亲。这样一来我们注意到矩形之间形成了一种树结构，某个矩形的形状和位置仅跟其父亲有关。按照题目所述的规则确定，根据父亲的形状和位置确定儿子的形状和位置是容易的。因此对于每个修改操作，我们只需修改最外层的大矩形，再按照从父亲到儿子的顺序（拓扑序）依次重新计算每个矩形即可。

以下事实能帮助我们实现算法：

- 将矩形按面积从大到小排序即可得到一个可行的拓扑序
- 一个矩形的父亲一定是包含它的矩形中面积最小的

时间复杂度

$O(nwin^2 + nwin \cdot nresize)$

空间复杂度

$O(nwin)$

2010 B - Barcodes

题目大意

Code-11 编码的编码方式如题面所示。现给定对于某个 Code-11 编码的扫描结果（以每个区域的宽度表示），问该编码是否合法，若合法，问原串内容。由于扫描和打印的误差，测量的数据可能有正负5%的误差，并且可能是反向的。

算法讨论

本题考查模拟

首先我们需要识别每一个区域是宽的还是窄的。我们可以找出这些区域当中最宽的宽度记为max，最窄的宽度记为min，接下来如果一个区域与max更接近则认为它宽的，反之则认为它是窄的。接下来验证数据是否满足误差是否小于5%，也就是说是否能找到一个w，使得对于所有的窄区域，其宽度w_i满足$0.95w \le w_i \le 1.05w$，对于所有的宽域，其宽度w_i满足$1.9w \le w_i \le 2.1w$。设最窄的窄区域宽度为$smin$，最宽的窄区域宽度为$smax$，最窄的宽区域宽度为$bmin$，最宽的宽区域宽度为$bmax$，则w必须满足如下不等式：

$$\begin{cases} \frac{smax}{1.05} \le w \le \frac{smin}{0.95} \\ \frac{bmax}{2.1} \le w \le \frac{bmin}{1.9} \end{cases}$$

若该不等式有解，则表示宽窄区域识别成功，若无解则说明这是一个 bad code。

接下来判定方向。我们可以直接根据前 5 位是 00110 还是 01100 来判断方向，若都不是，则说明这是一个 bad code。

确定方向之后，我们只需六位一组分别解码即可。注意字符数量应该大于 4，首位字符应该是开始/结束符号，所有编码都应有对应的字符，开始/结束符号不应在串中间出现，每一组的第六位应该是 0，不满足以上任意一条都会导致 bad code。

接下来按照题目要求计算 C 和 K 并验证是否与编码串中相同即可。

时间复杂度

$O(n)$

空间复杂度

$O(n)$

2010 C - Tracking Bio-bots

题目大意

有一个$n \times m$的网格，格子从$(0,0)$到$(n-1,m-1)$标号，其中有w个矩形内的格子有障碍物，是不能走的。有一种机器人可以在网格上行走，设机器人在点(x,y)，那么它下一步可以走向$(x+1,y)$或$(x,y+1)$（当然目标位置必须是可以走的）。问有多少个初始位置(x,y)满足从此位置出发的机器人不能走到点$(n-1,m-1)$（本来就不能走的格子不计入答案）。

算法讨论

本题考查 BFS。

由于坐标范围较大，我们首先对地图进行离散化，并令新地图中每个格子的权值表示它代表的原地图中的多少个格子。这样一来，坐标范围就缩小到$O(w)$级别。不难发现求有多少个位置不能能够走到$(n-1,m-1)$就是求从$(n-1,m-1)$反向走不能走到的点的个数，也就是在新地图中，不能走到的点的权值和。我们从$(n-1,m-1)$反向 BFS 一遍，标记一下哪些格点可以走到，然后再求出没有障碍但不能走到的格点的权值和即可。

时间复杂度

$O(w^2)$

空间复杂度

$O(w^2)$

2010 D - Castles

题目大意

有n个城堡，它们之间通过$n-1$条无向边连接形成一棵树，你现在要攻占这些城堡。攻占城堡i时你的军队至少要有a_i人，并且攻占之后你的军队会减少d_i人。现在你需要选择一个起点开始攻占，攻占完一个城堡之后你可以带上剩下的部队继续攻占相邻的城堡，注意每到一个没有攻占过的城堡是就必须攻占它，而一个已经被攻占了的城堡可以安全经过。为保证安全，一条路的同一方向不能走两次。问军队至少需要多少人才能攻占所有城堡。

算法讨论

本题考查动态规划。

不难发现攻占城堡的实质就是找一种方式遍历整棵树，使得代价最小。我们首先枚举起点S，以起点为根将无根树转化为有根树。考虑树形 DP，设$f[x]$表示攻占x及其子树至少需要带多少人，$g[x]$表示攻占x及其子树会减少多少人，考虑如何转移。设x有孩子$s_1, s_2, \dots, s_m$，那么求解$f[x]$的过程实际上就是要确定一个$s_1, s_2, \dots, s_m$的排列，使得按该排列攻占子树所需的人数最少。事实上，我们将这些孩子按$f[s_i]-g[s_i]$从大到小排序而得到的排列即为最优排列。简单说明如下：对于某个排列$v_1, v_2, \dots, v_m$，不难计算该排列对应的最少需要的军队人数为$\max\{(f[v_i]-g[v_i])+\sum_{j=1}^{i-1} g[v_j] \mid i=1 \dots m\}$，要使最大值最小，就要让越大的$f[v_i]-g[v_i]$对应的$\sum_{j=1}^{i-1} g[v_j]$越小，而$\sum_{j=1}^{i-1} g[v_j]$是随$i$不降的，因此把越大的$f[v_i]-g[v_i]$放在越前面一定是优的。设在这个代价下攻占了所有子树需要的最小人数为c，不难得出$f[x]=\max(c+d_x, a_x)$，$g[x]=d_x+\sum_{i=1}^{m} g[s_i]$。于是，以$S$为起点的最少人数即为$f[S]$。对所有起点算得的答案取最小值即可。事实上，对于不同的S，若某个点x的在这两种情况下的父亲是相同的，那么在这两种情况下$f[x]$和$g[x]$也一定分别相同。对于这些相同的结果我们不需要重复求解，这样本质不同的状态就只有$O(n)$个了，因此总时间复杂度为$O(n \log n)$。

时间复杂度

$O(n \log n)$

空间复杂度

$O(n)$

2010 G - The Islands

题目大意

平面上有n个点$(x_1, y_1) \dots (x_n, y_n)$，$x_i$依次递增。两点间的距离定义为它们的直线距离。现要求找到一条路径，从最左边的点 1 出发，一直向右走到点n，再从n往左走，经过所有还没有走过的点回到点 1。特别地，有两个特殊的点a, b，你必须在向左走的过程中访问其中一个，在向右走的过程中访问另一个。求这条路径以及最小距离和。

算法讨论

本题考查动态规划。

我们这条从 1 出发回到 1 的路径拆成两条从左向右的路径来考虑，设$f[i][j][t1][t2]$表示，当前第一条路径最右边的点是i，第二条路径最右边的点是j，两条路径是否访问过特殊点的状态是分别是$t1, t2$（若$t1 = 1$则表示第一条路径访问过特殊点，$t1 = 0$则表示未访问过，$t2$同）时，两条路径长度和的最小值。考虑主动转移，无论是那条路径扩展一个点，那么扩展到的这个点一定是$T = \max(i + 1, j + 1)$，设t'表示T是否是一个特殊的点，$d_{i,j}$表示点i与点j之间的距离，我们可以尝试用$f[i][j][t1][t2] + d_{i,T}$更新$f[T][j][t1 \vee t'][t2]$（第一条路径扩展到点$T$），以及用$f[i][j][t1][t2] + d_{j,T}$更新$f[i][T][t1][t2 \vee t']$（第二条路径扩展到点$T$）。这样，问题的答案即为$\max\{f[n][i][1][1] | 1 < i < n\}$，由于要输出方案，我们还需要记录每个状态是从哪里转移而来的。

时间复杂度

$O(n^2)$

空间复杂度

$O(n^2)$

2010 H - Rain

题目大意

给你一个n个点m条边的平面图，每个区域都是三角形，赋予每个顶点海拔h_i，就形成了一张三维地形图。现在源源不断的大雨从天而降，雨水在该区域的某些低洼除沉积下来，就形成了湖。问总共形成了多少湖以及每个湖湖面的海拔高度。

算法讨论

本题考查计算几何、最短路。

分析题目，我们不难发现以下性质：

性质 1　每个湖至少包含一个顶点。

性质 2　每个顶点至多属于一个湖。

性质 3　水能在某个顶点上积累至海拔h的条件是：$h > h_i$且不存在任意一条从该顶点到边界顶点的路径，满足路径上的点的海拔均小于h。

性质 4　两个顶点属于同一个海拔为h的湖的条件是：水在这两个点上均能积累至海拔h，并且这两点间存在一条路径满足路径上的所有点海拔均小于h。

根据**性质 2** 每个顶点最多属于 1 个湖，显然这个湖的海拔就是满足**性质 3** 的最大的h，我们考虑如何求h。我们发现“满足**性质 3** 的最大的h”可以等价转化为“不存在任意一条从该顶点到边界顶点的路径，满足路径上的点的海拔最大值小于h，并且存在任意一条从该顶点到边界顶点的路径，满足路径上的点的海拔的最大值小于$h+1$”，也就是说

$$h = \min\{\max\{h_u \mid u \in P\} \mid P\text{是该顶点到边界的一条路径}\}$$

这个式子实际上就是短板原理——湖的海拔取决于海拔最低的“挡板”的海拔。记第i个顶点对应的h为d_i，不难得到如下转移方程

$$d_i = \begin{cases} h_i & , \quad i\text{为边界上的顶点} \\ \min\{\max(d_j, h_i) \mid i\text{与}j\text{相邻}\}, & i\text{不为边界上的顶点} \end{cases}$$

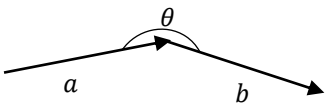
我们可以使用类似于 Dijkstra 算法的方式，初始时设边界上的点的d值为其海拔，其他点为inf，每次选取d值最小的点来更新周围的点，重复$n-1$次即可计算出所有d_i。

这样一来，对于某个顶点i，如果$d_i > h_i$，则表示存在一个包含i的湖，其湖面海拔为d_i。根据**性质 4**，我们从该点开始 BFS，只走海拔小于d_i的顶点，即可求出这个湖的其他顶点。我们依次确定每个顶点所属的湖，即可求出所有的湖。

求解d时，如果我们用堆来实现，那么该方法的时间复杂度即为$\mathrm{O}(n \log n)$。或者我们也可以对于每种海拔挂链表的方式，从小到大枚举海拔高度来实现优先队列，这种方法的时间复杂度为$\mathrm{O}(n+h)$。

接下来考虑如何寻找边界上的顶点。

我们首先寻找区域中最左边的顶点（x最小的顶点），由其出发的倾角最大的边一定在边界上，记这条边为 S。



如上图所示，定义两条连续的边a和b（连续表示a的终点就是b的起点）夹角θ为将b绕两条边的公共点逆时针旋转至于a重合所需旋转角度。我们从 S 出发，每次寻找与当前边夹角最小的边扩展下去，最终一定能按顺时针沿图形边界环绕一周，回到 S。这样我们就确定了边界上有哪些边，也就找出了边界上有哪些点。

时间复杂度

$\mathrm{O}(n \log n)$

空间复杂度

$\mathrm{O}(n)$

2010 J - Sharing Chocolate

题目大意

对于一块$a \times b$的巧克力，我们可以横着或者竖着切一刀，使巧克力变为$x \times b, (a-x) \times b$两块或$a \times y, a \times (b-y)$两块（这里$x, y$均为整数），对切出来的每个小块重复这个过程，我们便能把一大块巧克力分成许多小块。现给定一个$A \times B$的巧克力，问是否存在一种切法，使得最终能得到n个小块，且每个小块的面积分别是$a_1, a_2, \dots, a_n$。

算法讨论

本题考查动态规划或记忆化搜索。

令U表示n个目标小块的全集，设$f[S][x]$表示使用U的某个子集S是否能拼成宽度为x的矩形。考虑如何计算$f[S][x]$。令$Aera(S)$表示S集合包含的小块的面积和，那么若$x \nmid Aera(S)$，则一定不能拼成。若$x | Aera(S)$，首先检查x和$\frac{Aera(S)}{x}$是否都不大于$\max(A, B)$，如果大于的话即使可以拼成也没有意义，因此我们直接认为不能拼成，否则令$y = \frac{Aera(S)}{x}$，我们枚举S的非空真子集T，考虑把这个$x \times y$的矩形横切或者纵切一刀分成T和$\complement_S T$是否可行。若$f[T][x]$与$f[\complement_S T][x]$均可行，那么说明横切是可行的；若$f[T][y]$与$f[\complement_S T][y]$均可行，那么说明纵切是可行的。在我们枚举的所有方案中，只要有一种方案的横切或是纵切是可行的，则说明$f[S][x]$是可行的。我们首先计算出所有S只包含一个小块时的f值，然后通过 DP 即可求解f，最终答案为$f[U][A]$。当然我们也可以通过记忆化搜索来求解，并且相较于 DP，记忆化搜索可以更好的避免无用状态，因此实际效果更好。

时间复杂度

$O(3^n \cdot \max(A, B))$，实际因有用状态数远达不到上限值，因此实际效果较好。另外通过对转移方程的简单修改可以做到$O(3^n \cdot \min(A, B))$，但因为A与B同阶，意义不大。

空间复杂度

$O(2^n \cdot \max(A, B))$

2011 A - To Add or to Multiply

题目大意

某种计算机只能进行$\times m$或者$+a$操作，该计算机上的一个程序是一个由 A 和 M 组成的序列，表示对输入依次进行$+a$或者$\times m$操作。现在给定已知输入在$[l_1, r_1]$范围内，要求输出在$[l_2, r_2]$范围内，求最短的可行的程序。若有多个程序可行，要求字典序最小。

算法讨论

本题考查枚举、数学。

设原数为x，那么显然在一系列操作后它将变为$m^c x + d \cdot a$，其中c, d是与操作序列有关的数。

当$m = 1$时，乘法使用次数一定为 0；当$m > 1$时，乘法使用次数不会超过$\log_m r_2$。因此，我们可以先枚举乘法使用次数。设当前枚举到乘法使用次数为n的情况，我们计算出可能的d的最小值d_l与最大值d_r，那么问题就转化为在$[d_l, d_r]$中选择一个d，将其表示成$p_n m^n + p_{n-1} m^{n-1} + \cdots + p_1 m + p_0$，最小化$n + \sum_{i=0}^{n} p_i$。容易看出对于某个确定的$d$，类似于$m$进制分解的表示方法一定是最优的，那么接下来只需考虑如何在$[d_l, d_r]$中选择一个最优的d。我们将d_l, d_r分别分解，令$d_l = p_n m^n + \cdots + p_1 m + p_0$，$d_r = q_n m^n + \cdots + q_1 m + q_0$，找到最大的$i$使得$p_i \neq q_i$，不难证明最优情况下$d = \begin{cases} p_n m^n + \cdots + p_i m^i & , p_{i-1}, p_{i-2}, \ldots, p_0 \text{均为 } 0 \\ p_n m^n + \cdots + (p_i + 1) m^i & , p_{i-1}, p_{i-2}, \ldots, p_0 \text{不均为 } 0 \end{cases}$，于是我们便得到了乘法使用次数为$n$时的最优解。我们对于每个$n$分别算出最优解，并在其中取最优的那个即可。

时间复杂度

$O(\log_m^2 r_2)$

空间复杂度

$O(\log_m r_2)$

2011 C - Ancient Messages

题目大意

给定一个$n \times m$的二维 01 矩阵，这个矩阵组成了一幅图。要求从中识别出给定的六个图形，并输出每个图形出现了几次。

算法讨论

本题考查图形识别。

观察图像可得每个图形内部的白色连通块个数如下表所示：

图形编号	W	A	K	J	S	D
内部白色连通块个数	0	1	2	3	4	5

我们可以发现图形内部白色连通块的个数两两不同，因此我们可以通过这个特征来识别图形。

我们首先在最外层的白色区域进行 FloodFill 以识别出最外面的空白区域，之后在所有黑色连通块上进行 FloodFill 找出所有图形，并对它们编号。接下来还剩下的区域就是图形内部的白色连通块了。我们对剩下的每一个白色连通块 FloodFill 一遍，并记下与这个白色连通块相邻的任意一个黑色格子所属的图形编号，于是这个白色连通块就是该图形内部的一个白色连通块。这样我们就能知道每个图形中分别有几个白色连通块，也就识别出了这些图形。

时间复杂度

$O(dx \cdot dy \cdot q)$

空间复杂度

$O(dx \cdot dy)$

2011 E - Coffee Central

题目大意

有一平面直角坐标系，其x的范围是$1 \ldots dx$，y的范围是$1 \ldots dy$。其上分布了n个点(x_i, y_i)。定义点(a, b)与点(c, d)的距离为$|a-c|+|b-d|$。现给定一个最大距离m，要求选择一个点(x_0, y_0)，使得与(x_0, y_0)的距离小于等于m的点最多。一共有q个m需要计算，对于每个m输出距离小于等于m的点数以及(x_0, y_0)的坐标。

算法讨论

本题考查枚举。

我们先转换坐标系，将每个点(x, y)的新坐标设为$(x+y, x-y)$，并令新坐标系上的两个点$(a, b), (c, d)$的距离为$\max(|a-c|, |b-d|)$。不难发现同一对点在新旧坐标系下的距离是相等的。此时我们发现，到某个点距离小于等于m的点可能存在的范围变成了一个以该点为中心，边长为$2m$的正方形。我们枚举每个位置并利用前缀和来计算对正方形内的点数，然后取最大的那个即可。

时间复杂度

$O(dx \cdot dy \cdot q)$

空间复杂度

$O(dx \cdot dy)$

2011 F - Machine Works

题目大意

有n件物品，对于第i件物品你可以在第d_i天以p_i元的价格买入，并在任意一天以r_i元的价格卖出，每持有该物品一天可以获得g_i元的收益（买入和卖出的那天不能获得收益），并且你最多只能持有一件物品（但可以在同一天卖出物品并买入新物品）。一开始你有c元钱和d天时间，并且你会在第$d+1$天把所持有的物品（如果存在）卖掉。求最大收入。

算法讨论

本题考查动态规划。

设所有物品已按收益值g排序，不难发现如果当前已经购买了某个物品，那么下一个卖出这个物品的时间一定与购买下一个物品是同一天，并且下一个物品的收益一定大于这个物品。于是令$f[i]$表示在购买物品i的情况下，第d_i天剩余钱数的最大值。我们增设一个 0 号点并令$f[0]=c, g_0=0, r_0=0, d_0=0$，于是不难得到转移方程：

$$f[i]=\max\{f[j]+g_j\cdot(d_i-d_j-1)+r_j-p_i|j=0\dots i-1\}$$

注意到该方程并没有保证$d_i>d_j$，但实际上由于g是递增的，$d_i\le d_j$的决策一定不是最优的，因此可以保证这个转移方程正确性。直接计算f的复杂度是$O(n^2)$的，我们考虑斜率优化。计算$f[i]$时对于两个决策$j,k\ (j>k)$，j比k优的条件是

$$f[j]+g_j\cdot(d_i-d_j-1)+r_j-p_i>f[k]+g_k\cdot(d_i-d_k-1)+r_k-p_i$$

整理得

$$\frac{(f[j]-g_jd_j-g_j+r_j)-(f[k]-g_kd_k-g_k+r_k)}{g_j-g_k}>-d_i$$

于是我们可以将每个决策j看作点$(g_j, f[j]-g_jd_j-g_j+r_j)$并维护它们的上凸壳即可利用二分在每次$O(\log n)$的时间内完成一次决策。最后，$\max\{f[i]+r_i+g_i\cdot(d-d_i)|i=0\dots n\}$即为问题的答案。

时间复杂度

$O(n\log n)$

空间复杂度

$O(n)$

2011 H - Mining Your Own Business

题目大意

给定一张n个点m条边的无向图，你需要在一些点上做标记，使得删除任意一个点之后，剩下的每个点都至少与一个标记连通。问至少需要打几个标记，打标记的方案有几种。

算法讨论

本题考查无向图割点、点双连通分量。

我们首先求出该图的割点以及所有点双连通分量，接下来按割点的存在情况分类讨论。

如果存在割点，那么最优策略一定是在每一个只包含一个割点的双连通分量中的某一个非割点上放标记，这很容易证明：首先这些标记是必要的，因为删去这个连通块对应的割点时，这个连通块的其他点就会独立成一个连通块，而这个连通块内至少需要有一个标记；其次，只标记这些点就可以满足题目要求，若我们删去的点不是割点，那么整张图的连通性不变，显然满足条件，若删去的点是割点，那么剩下的点要么属于只有一个割点的连通块，要么可以通过剩下的割点与只有一个割点的连通块相连，因此所有剩下的点都至少会和一个标记相连。此时需要放置的标记数就是只有一个割点的连通块的个数，而方案数就是这些连通块中非割点数目的乘积。

如果不存在割点，也就是说整个图无论删除哪个点都是连通的，那我们只要在图中任意两个点上放置标记即可。证明如下：首先一个标记显然是不够的，那样的话我们只要删除标记的节点就会导致剩下的节点不与任何标记相连；两个标记是足够的，因为删除任意一个点之后的图依然是连通的，而图中至少还剩下一个标记，那么所有点都可以到达这个标记。

我们可以用 tarjan 算法求出所有割点，再从每个割点开始 BFS 即可求出所有的点双连通分量以及其中的割点和非割点个数。

时间复杂度

$O(n+m)$

空间复杂度

$O(n+m)$

2011 I - Mummy Madness

题目大意

在一个平面直角坐标系上，你在点(0,0)，有n个木乃伊，分别在点(x_i, y_i)。每一步，你可以选择向任意一个相邻的格子移动（8-连通），或者不动，然后所有的木乃伊也会向一个相邻的格子移动（8-连通），或者不动，使得它与你的欧几里得距离最小。当木乃伊与你在同一个格子的时候，你就被抓到了。你需要确定你是否一定会被抓到，如果是，计算最晚被抓时间。

算法讨论

本题考查二分答案、单调队列。

设$m = \max\{\max(|x_i|, |y_i|) \mid 1 \le i \le n\}$，不难发现，若木乃伊在$m$步之后仍没有抓到你，那么它将永远抓不到你。我们考虑如何判断你是否可能在s步之后仍没有被抓到。

我们先考虑一维的情况。在一维情况下，你每一步都可以向左、向右或不动，而木乃伊会朝着你的方向走来，不难发现你最终的位置可能是$[-s, s]$中的任何位置，而木乃伊所在的位置则可能是$[x_i - s, x_i + s]$，因此只要$[-s, s]$中存在某个点不被任何木乃伊区间$[x_i - s, x_i + s]$覆盖，那么逃到这个点上就是安全的，即你可以在s步之后不被抓到。在二维情况下，x方向和y方向的运动是互独立的，在x方向上你可以到达的区间$[-s, s]$是，木乃伊是$[x_i - s, x_i + s]$；y方向上你可以到达的区间是$[-s, s]$，木乃伊是$[y_i - s, y_i + s]$。不被抓到的条件与一维情况有些不同：如果存在点(x, y)，满足对于每个木乃伊i，都有 $x \notin [x_i - s, x_i + s]$或$y \notin [y_i - s, y_i + s]$，则可以在$s$步之后不被抓到。我们枚举$[-s, s]$中的每一个$x$，这样一来所有满足$x \notin [x_i - s, x_i + s]$的木乃伊$i$就无法到达$x$，我们只需考虑剩下的在$x$方向上能到达$x$的木乃伊，也就是所有满足 $x \in [x_i - s, x_i + s]$ 的i。这时问题转换为一维情况：在y方向上，剩余的木乃伊是否能在s步之内抓到你。这个问题实际上就是一个线段覆盖问题，询问给定的一个线段集合是否完整的覆盖了$[-s, s]$这个区间。由于我们之前枚举x的过程中会导致需要考虑的线段集合的增减，因此我们需要一个数据结构，支持插入、删除一个线段，查询某个线段有没有被完整覆盖。线段树明显能够解决这个问题，这样判断一次的时间复杂度为$O(m + n \log n)$。我们枚举的时候，跳过一些线段集合相同的点，复杂度即可降至$O(n \log n)$。

但是，以上的做法没有考虑此问题区间的特殊性——区间长度都是$2s$。首先考虑y方向，注意到覆盖的区间和询问区间长度都是$2s$，并且询问区间固定为$[-s, s]$，这样我们可以把覆盖区间分成两种类型，一种覆盖询问区间的$[-s, r]$这一部分，另一种能覆盖询问区间$[l, s]$这一部分，这样一来，我们只需要维护r的最大值和l的最小值，如果$r_{max} \ge l_{min} - 1$，则$[-s, s]$能被完全覆盖，否则不能。再考虑x方向，区间长度都是$2s$表明，每个木乃伊需要被考虑的时长都相等，也就是说，在线段集合中先插入的区间先删除。这样我们维护r的最大值和l的最小值时，

就可以使用单调队列，达到均摊$O(1)$的时间复杂度。这样我们就能在$O(n)$时间内判断你是否可能在s步之后仍没有被抓到。

这样一来，我们只要二分答案并判定即可。如果答案的步数大于m，则可判定为 never，否则输出答案。二分的次数至多为$O(\log m)$，因此该算法的时间复杂度为$O(n \log m)$。

时间复杂度

$O(n \log m)$

空间复杂度

$O(n)$

2011 J - Pyramids

题目大意

一个底面边长为$n\ (n \geq 2)$的高金字塔需要$\sum_{i=1}^{n} i^2$个石块，一个底面边长为$n\ (n > 2)$的矮金字塔需要的石块数则为$\begin{cases} \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} (2i+1)^2, & n\text{为奇数} \\ \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} (2i)^2, & n\text{为偶数} \end{cases}$，现有$S$个石块，要修建一些金字塔，要求：

1. 用完所有石块
2. 金字塔个数尽量少
3. 金字塔两两不同
4. 满足以上条件的基础上，用去石块最多的金字塔用掉的石块尽可能多
5. 满足以上条件的基础上，用去石块次多的金字塔用掉的石块尽可能多
6. 依此类推……

问是否能建成，如果能，问最优方案。

算法讨论

本题考查动态规划。

我们先求出所有可能被使用的金字塔，记其个数为m，由$S \leq 10^6$可以计算出$m \leq 320$。接下来问题转化为一个 01 背包问题：给定m个物品，问最少选用多少，能填满一个体积为S的背包，并且满足字典序最大。

回顾经典的背包算法，$f[i]$表示当前状态下体积为i时最小需要的物品数，之后每加入一个体积为V的物品时，使用转移方程$f[i] = \max(f[i-V]+1, f[i])$倒序更新即可。本题在此基础上还要输出决策以及保证方案字典序最大。关于决策的记录，我们使用一个$g[i]$来保存当前$f[i]$对应选了哪些物品，由于此题具有特殊性，打表可得最终选择的物品数不会超过 6，因此可以使用一个 64 位整数来记录选择了哪些物品。关于字典序最大，我们首先将物品按体积从小到大排序然后依次放入“背包”，并且更新时遇到$f[i-V]+1 = f[i]$时也用$f[i-V]+1$去更新$f[i]$，这样就保证了$f[i]$对应的最后一次决策选择的是所有可行的物品中最大的那个，也就保证了整个决策的字典序最大。

时间复杂度

$O(mS)$

空间复杂度

$O(m+S)$

2011 K - Trash Removal

题目大意

在一个二维平面上给定一个多边形，试确定一个旋转角度，使该多边形最左边的点和最右边的点水平距离最小。求这个最小距离。

算法讨论

本题考查凸包、旋转卡壳。

不难发现答案只与点集有关而与图形无关，因此我们只考虑点集。我们求出这个点集的凸包，答案就是凸包的宽（凸多边形的宽度被定义为平行切线间的最小距离）。下面简单说明如何利用旋转卡壳来求凸包的宽。根据宽的定义，不难发现平行切线中的其中一条一定与凸包的某条边重合（否则一定可以通过旋转使得答案更优）。于是我们先确定一条边并找到凸包上与其距离最大的点，并按顺时针依次枚举凸包上的边。注意到对于任意一条边，凸包上其他点到它的距离按它们在凸包上的顺序呈单峰分布，这样我们就可以在枚举边的过程中通过不断将距离最大的点的位置向前进方向移动直到距离将要变小为止，即可在均摊$O(1)$的时间内维护出当前枚举的边对应的最远点。在依次枚举完所有边之后，这些最大距离的最小值即为凸包的宽，也就是本题的答案。

时间复杂度

$O(n \log n)$

空间复杂度

$O(n)$

2012 A - Asteroid Rangers

题目大意

三维空间上有n个点，其初始位置和运动速度已知。两点之间的距离被定义为两点间的直线距离，求这n个点在运动过程中，本质不同的最小生成树的个数。（两个最小生成树本质不同的意思是它们的边集不同）。

算法讨论

本题考查数学、排序、最小生成树、DFS 序。

一个比较显然的结论是，如果所有边两两之间大小关系都不变，那么最小生成树保持不变。而由于所有点的运动速度不变，所以任意两条边的大小关系最多变化两次，我们不妨首先计算出所有大小关系发生变化的事件(t,x,y)，表示在t时刻，边x的长度超过了边y。我们对这些事件按时间排序，然后逐个检查每个事件发生时，最小生成树是否发生变化。具体如下：首先建立初始时的最小生成树，之后每遇到一个事件(t,x,y)时检查y是否能够在当前最小生成树上取代x，设y的两个端点为u,v，则能取代的条件为x在当前生成树中u到v的路径上。若满足条件，我们就可以在当前最小生成树上删去x并加入y，便得到了新的最小生成树；若不满足条件，那么最小生成树显然不会改变。

考虑如何检查是否满足条件并维护最小生成树。我们可以保存当前最小生成树的 DFS 序，这样就可以在$O(1)$时间内检查一个点是否在另一个点的子树内。设p为边x的两个端点中深度较大的那个，那么边x在u到v的路径上就可以等价转化为u和v中有且仅有一个在p的子树内。于是对于某个事件，若最小生成树不发生变化，那么处理该事件仅需要$O(1)$时间，若最小生成树发生变化，由于需要重新计算 DFS 序，因此需要$O(n)$时间。总时间复杂度为$O(n^4 \log n + nAns)$，由于Ans较小所以效果较好。当然，由于能发生取代时x必定是u到v的路径上的最大边，因此使用 Link-Cut Tree 来检查并维护生成树也是可行的，时间复杂度为$O(n^4 \log n)$。

时间复杂度

$O(n^4 \log n + n \cdot Ans)$或$O(n^4 \log n)$

空间复杂度

$O(n^4)$

2012 B - Curvy Little Bottles

题目大意

有一条曲线$f(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$，将曲线的$[x_{bot},x_{top}]$这段绕$x$轴旋转一周就能形成一个曲面。将这个曲面作为一个瓶子的侧面，再将x_{bot}处封死，我们就能得到一个瓶子。接下来你要在这个瓶子上标记刻度，每隔inc体积标记一次，标记满八次后停止。求这个瓶子的容积以及每个标记的位置。

算法讨论

本题考查数学、二分。

首先考虑如何计算容积。不难得到该曲线在x_0处形成的圆的面积为$\pi f^2(x_0)$，由此可以得到该瓶子的容积$\mathrm{V}=\int_{x_{bot}}^{x_{top}}\pi f^2(x)dx$。由于$f(x)$是多项式，因此$f^2(x)$亦是多项式，于是容易算出$\int f^2(x)$，也就可以计算容积了。

接下来考虑如何计算标记的位置。设$V(x)$表示瓶子高度x以下部分的容积，$V(x)$可以通过$\int f^2(x)$很方便地求出，接下来计算每个标记的位置实际上就变成了对于每个标记分别解方程$V(x)=V_{mark}$，注意到$V(x)$关于x单调递增，因此我们可以通过二分x来解这个方程。

时间复杂度

$\mathrm{O}\left(n^2+\log\frac{ans}{eps}\right)$

空间复杂度

$\mathrm{O}(n)$

2012 C - Bus Tour

题目大意

有一张n个点m条边的带权无向图，我们需要从 1 号点出发，按某一顺序访问$2 \dots n-1$号点（允许中间经过某些点但不停下来，这样不算访问了这些点），最后到达n号点，再从n号点出发，按某一顺序访问$2 \dots n-1$号点，最后回到 1 号点。两次访问$2 \dots n-1$的顺序由你确定，但是需要保证第一个顺序中的前$\left\lfloor \frac{n-2}{2} \right\rfloor$个点也是后一个顺序中的前$\left\lfloor \frac{n-2}{2} \right\rfloor$个点。整条线路的代价是其中所有边的边权和，求最小代价。

算法讨论

本题考查最短路、动态规划和枚举。

首先我们预处理出图中任意两点间的最短路$d[i][j]$，这可以用 Floyd 算法实现。

设$f[S][i]$表示从点 1 开始，访问过的点集为S，最后访问的点为i的最小代价，那么不难得到转移方程

$$f[S][i] = \min\{ f[S-i][j] + d[j][i] \mid j \in S, j \neq i \}$$

这样我们就可以在$O(2^n n^2)$时间内计算出f。同时我们设$g[S][i]$表示从点n开始，访问过的点集为S，最后访问的点为i的最小代价。求解g的方法与f基本相同。

接下来我们只需枚举前$\left\lfloor \frac{n-2}{2} \right\rfloor$个经过的点集$S$，第一遍时的第$\left\lfloor \frac{n-2}{2} \right\rfloor$个点$i$，第二遍时的第$\left\lfloor \frac{n-2}{2} \right\rfloor$个点$j$，那么这条路径的代价即为$f[S][i] + g[\complement_U S \cup i][i] + g[S][j] + f[\complement_U S \cup j][j]$。直接枚举并求最小值的时间复杂度是$O(2^n n^2)$，与 DP 同阶，已经可以接受了。然而我们还可以继续优化，不难发现上式中i与j互不相干，因此我们可以分别枚举i和j，对于同一个S分别求出 $f[S][i] + g[\complement_U S \cup i][i]$ 与 $g[S][j] + f[\complement_U S \cup j][j]$ 的最小值然后简单相加即可。这样一来这一步的时间复杂度就降为了$O(2^n n)$。总时间复杂度仍维持不变为$O(2^n n^2)$。

时间复杂度

$O(2^n n^2)$

空间复杂度

$O(2^n n)$

2012 D - Fibonacci Words

题目大意

Fibonacci Words 的定义如下：

$$f(i)=\begin{cases}0, & i=0\\ 1, & i=1\\ f(i-1)+f(i-2), & i\geq 2\end{cases}$$

其中“+”表示字符串连接符。因此我们有$f(2)=10, f(3)=101, f(4)=10110, \ldots$现给定一字符串$s$和非负整数$n$，求$s$在$f(n)$中出现的次数。

算法讨论

本题考查递推、字符串匹配。

设$len(x)$表示字符串x的长度。我们求出满足$len(f(w))\geq len(s)$的最小的w，记$A=f(w)$, $B=f(w+1)$，则$f(w)\ldots f(n)$就都能看成由A,B按某种方式拼接而成的字符串。设c_A表示s在A中出现的次数，c_B表示s在B中出现的次数，c_{AA}表示s在AA中出现的次数，c_{AB}, c_{BA}, c_{BB}的含义依此类推，这六个值可以通过 KMP 等字符串匹配算法在线性时间内求出。假设我们已经知道了$f(i-2)$开头是L_2，结尾是R_2，s在其中出现的次数为a_2，$f(i-1)$开头是L_1，结尾是R_1，s在其中出现的次数为a_1（L_1, R_1, L_2, R_2均是A,B中的一种），那么由于$f(i)=f(i-1)+f(i-2)$，容易推知$f(i)$的开头是L_1，结尾是R_2，s在其中出现的次数为$a_1+a_2+c_{R_1L_2}-c_{R_1}-c_{L_2}$。又我们已经知道$f(w)$的开头是$A$，结尾是$A$，$s$在其中出现的次数为$c_A$，$f(w+1)$的开头是$B$，结尾是$B$，$s$在其中出现的次数为$c_B$，因此我们可以根据上面的递推来求解$s$在$f(n)$中出现的次数。注意特判$len(f(n))<len(s)$的情况。

时间复杂度

$O(len(s))$

空间复杂度

$O(len(s))$

2012 E - Infiltration

题目大意

给定一张有向图，任意两个点之间有且仅有一条边，求该图的最小支配集。

算法讨论

本题考查搜索。

考虑如下贪心：每次选择当前剩余没有被支配的点中能支配的点最多的点加入支配集。由于任意两个点之间都有边，设当前没有被支配的点的个数为n，那么我们本次加入的点就至少可以支配$\left\lceil\frac{n-1}{2}\right\rceil$个点，因此图中至多剩下$\left\lfloor\frac{n-1}{2}\right\rfloor$个点。由于$n \leq 75$，我们可以算出该贪心会产生至多包含 6 个点的支配集（实际上在绝大多数情况下该贪心可以产生包含至多 5 个点的支配集）。接下来我们只需要通过搜索来验证是否存在比贪心答案更小的解即可。若贪心解为 6，那么我们只需验证所有五个点的子集，其个数至多为$C_{75}^{5} = 17259390$，我们在搜索时使用位运算来验证是否支配了所有点，即可在规定时间内出解。但实际上对于该题的测试数据，贪心解为最大为 5，则需要验证的子集个数最多只有$C_{75}^{4} = 1215450$，该算法运行速度很快。

时间复杂度

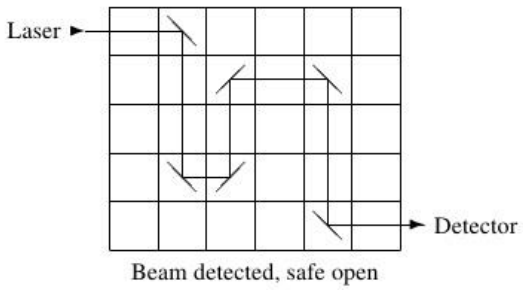
$O(n^{\log n})$

空间复杂度

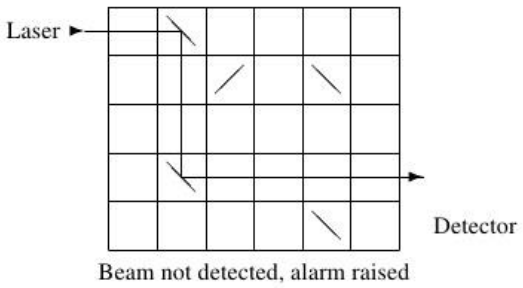
$O(n)$

2012 I - A Safe Bet

题目大意



Beam detected, safe open



Beam not detected, alarm raised

如图，在一个$n \times m$的网格中放置有t面镜子，方向斜向左或者斜向右。问从左上角向右入射（左图中 Laser 位置）的光线是否能从右下角向右射出（抵达 Detector 位置）。若不能，你需要再放置一面镜子使得光线能够到达 Detector，问可行方案的个数以及字典序最小的可行方案。

算法讨论

本题考查排序、二分查找、扫描线、数据结构。

我们首先计算出从 Laser 入射的光路和从 Detector 入射的光路，若此时两条光路重合，即从 Laser 射入的光线已经能够达到 Detector，那么就不需要放置镜子了；否则，根据光路可逆原理，不难发现这两条光路不会有公共边，并且所有交点的位置即为可行的镜子放置位置。

接下来考虑如何计算光路。要计算光路我们只需要解决这么一个子问题——从某个点向某个方向射出的光线到达的第一面镜子在哪里，经过该镜子反射后的出射光线的方向是什么。而这个问题实际上就是问某一行或某一列中小于某数的最大数是什么或大于某数的最小数是什么，可以用排序+二分查找解决。出射光线的方向也可以根据镜子的方向和入射光线的方向判断出来。这样我们通过不断的找下一面镜子即可求出完整的光路。

接下来考虑如何计算交点个数。由于相交的光线必定是一横一竖，我们不妨只考虑如何计算光路一的竖向光线与光路二的横向光线的交点。我们使用扫描线从左到右依次扫描并维护当前扫描线上的横向光线（如果某条横向光线与当前扫描线有交点，我们称这条横向光线在这条扫描线上）。当遇到一条竖向光线的时候，计算它与其他横向光线的交点数实际上就是在计算当前扫描线上某个区间内有多少横向光线，因此我们在扫描过程中用树状数组或线段树等支持区间求和的数据结构来维护横向光线即可。字典序最小的方案也可以在扫描过程中找到第一个解时顺便求出。

时间复杂度

$O(t \log t + t \log m + n + m)$

空间复杂度

$O(n + m + t)$

2012 K - Stacking Plates

题目大意

有n堆盘子，第i堆盘子有h_i个，从上到下第j个盘子的直径是$a_{i,j}$（保证$a_{i,j} \le a_{i,j+1}$），现允许进行两种操作：

- **Split** 将某堆盘子最上面的一些盘子拿出来形成新的一堆。
- **Join** 将某一堆盘子放到另一堆的上面，要求放入的那堆的最下面的盘子直径不大于被放入那堆最上面的盘子的直径。

我们的目标是把所有盘子合并为同一堆，求最小操作次数。

算法讨论

本题考查动态规划。

首先同一堆里面的直径相同的盘子一定是一起移动的，因此我们把它们当做一个盘子来考虑。接下来我们对直径进行离散化，并统计每种直径分别存在于多少个堆以及哪些堆里，设tot表示直径种数，cnt_i表示直径i出现在多少个堆里，$c_{i,j}$表示堆j中是是否存在直径i。考虑最后合并成的那堆，把其中相邻的来自同一个堆的盘子看做一组，统计组数m，那么操作次数就是$2m-n-1$。于是我们的目标就是确定最终的排列方式，使得m最小，这可以通过动态规划实现。令$f[i][j]$表示已经确定了前i种直径的盘子的排列方式，并且最后一个盘子来自第j堆时的最小组数，那么不难的到转移方程：

$$f[i][j]=\begin{cases} inf, & \neg c_{i,j} \\ \min\begin{cases} f[i-1][k]+1, & \neg c_{i,k} \\ f[i-1][k], & c_{i,k} \end{cases}, & c_{i,j} \wedge cnt_i = 1 \\ \min\begin{cases} f[i-1][k]+cnt_i, & \neg c_{i,k} \wedge k \neq j \\ f[i-1][k]+cnt_i-1, & c_{i,k} \wedge k \neq j \end{cases}, & c_{i,j} \wedge cnt_i > 1 \end{cases}$$

最后，最小组数便是$\max\{f[tot][i] \mid i=1 \dots n\}$。

根据转移方程直接计算f的复杂度为$O(n^2h+n \cdot tot)$，简单地利用前缀后缀最小值来优化转移即可做到$O(n \cdot tot)$；如果我们在计算时避免计算$c_{i,j}=false$的状态，那么有用的状态数就只有nh个，配合前后缀最小值，可将计算f的复杂度降为$O(nh+tot)$，又因为$tot<nh$，因此DP的时间复杂度为$O(nh)$。于是，若离散化使用计数排序，则总时间复杂度为$O(nh+MaxD)$，其中$MaxD$表示盘子的最大直径。若$MaxD$较大，我们可以使用快速排序进行离散化，总时间复杂度为$O(nh \log nh)$。

时间复杂度

$O(nh + MaxD)$或$O(nh \log nh)$。

空间复杂度

$O(nh)$，但为方便实现我的程序空间复杂度为$O(MaxDn)$。

2012 L - Takeover Wars

题目大意

有两个人 A 和 B 分别有一个集合，A 的集合里有n个数，B 的集合里有m个数，现在这两个人之间轮流进行游戏，A 先手。每一轮，当前进行游戏的一方可以选择将自己的某两个数合并（删掉这两个数并把它们的和加入自己的集合）或吃掉对方的一个数（若自己的集合里有一个数x，对方的集合里有一个数y，满足$y < x$，则可以把y从对方的集合里删掉）。问在最优策略下，谁会获胜。

算法讨论

本题考查模拟。

首先有两个比较显然的结论：

- 若选择进行合并，则合并的必定是自己集合中最大的两个数。
- 若选择吃掉对方，则必定要吃掉对方最大的数，否则必定选择合并。

有了这两点结论之后不难推出下面两个结论：

- 若对方上一个操作是合并，则尝试吃掉对方，若可以吃掉，自己必然胜利。
- 若对方上一个操作是吃，则自己必然进行合并。

这样一来我们发现，只要第一个操作确定了之后，之后的操作就能依次确定。因此我们只要枚举第一个操作是什么并推演出结果，若存在先手胜的情况则答案为先手胜，否则答案为先手负。

时间复杂度

$O(n)$

空间复杂度

$O(n)$

2013 A - Self-Assembly

题目大意

给定一些形状大小相同的正方形，正方形的四边上标记了形如“X+”或“X-”或“00”的字串，其中 X 是一个字母。两个边界可以拼合的条件是它们的字串为“X+”和“X-”（相同字母），“00”不能与任何边界拼合。现假设每一种正方形有无限多，问在允许旋转和翻转的前提下，能否拼成一个无限大的图形。

算法讨论

本题考查图论。

很明显能拼成无限大的图形的条件是存在一种由一个或多个正方形拼合而成的结构，使得“X+”和“X-”都存在于这个结构的边界上。我们将这个结构复制多遍然后不断将“X+”和“X-”拼接起来即可得到一个无限大的图形。因为正方形可以旋转和翻转，因此我们可以保证通过翻转和旋转一定能使一系列这样的结构能够拼接起来而不会出现成环或者自交的情况。接下来考虑如何判断这样的结构是否存在。

我们对于每一种字串建立节点，设一个正方形包含的不为“00”的字串是$A_1 \dots A_k$，能与它们相拼接的字串分别是$A'_1 \dots A'_k$（字母相同，符号相反），那么我们就从所有A_i向A'_j $(i \neq j)$连有向边，不难发现若图中存在环，就说明存在上述结构，答案即为 unbounded，否则答案就是 bounded。

时间复杂度

$O(n)$

空间复杂度

$O(n)$

2013 B - Hey, Better Bettor

题目大意

有一种赌博，每次参与需要花费 1 元，若获胜，你能获得 2 元，否则你将不获得任何钱。每次你获胜的概率为$P\%$。赌场为了吸引顾客，作了如下规定：若某一时刻你的总资金减少了，那么你可以赎回你失去的钱的$X\%$，但是你只能赎回一次。当然如果你赢钱了，你可以直接拿钱走人。求最优策略下的期望收益。

算法讨论

本题考查数学、二分。

令$x = 1 - X\%, p = P\%$，令$f(i)$表示当净收益为i时的最大期望总收益。不难列出：

$$f(i) = \max \begin{cases} i \\ ix \\ p \cdot f(i+1) + (1-p) \cdot f(i-1) \end{cases}$$

其意义是：每次要么继续赌博，要么停手拿钱走人。不难发现若对于某个$a\ (a > 0)$我们选择了第一种决策，那么我们就永远不会用到大于a的决策了，即第一种决策只在a处作了一次；同理也存在一个$b\ (b < 0)$，我们对b作了第二种决策，而第二种决策也只在b处作了一次。这样我们就可以发现，对于$a < i < b$，$f(i) = p \cdot f(i+1) + (1-p) \cdot f(i-1)$。现在对于确定的$a, b$，我们已经知道了$f(b)$和$f(a)$，要求的最大期望收益就是$f(0)$。

将上式变形可得$f(i) = \frac{1}{p} f(i-1) + \frac{p-1}{p} f(i-2)$，其特征方程为$x^2 - \frac{1}{p} x - \frac{p-1}{p} = 0$，解得$x_1 = 1,\ x_2 = \frac{1-p}{p}$，于是我们可以知道$f(i)$可以表示为$A + B\left(\frac{1-p}{p}\right)^i$，又$f(b) = bx, f(a) = a$，因此可以算出系数$A, B$，进而根据$f(0) = A + B$计算答案。

我们发现，当a固定时，$f(0)$关于b是单峰的；当b取相应最优值时，$f(0)$关于a也是单峰的，因此我们可以通过二分套二分或者三分套三分的方法来解决此问题。

时间复杂度

$O(\log^3 ans)$

空间复杂度

$O(1)$

2013 C - Surely You Congest

题目大意

一个n个节点，m条边的无向图上有c辆车，每辆车现在的位置已知。现在每辆车都沿最短路向点 1 开去，但由于每条路同时只能有一辆车通过，所以不是辆车都能到达点 1。问在最优方案下最多有多少车能开到点 1。

算法讨论

本题考查最短路、网络流。

设p_i表示i号点现在有多少辆车，d_i表示点 1 到点i的最短路。那么对于点i,j，若$d_i \neq d_j$，则从i出发的车和从j出发的车一定不会互相影响。我们考虑把d相同的点分为一组，然后每组分别处理。对于每一组，建立一张包含原来的n个点并新加入一个汇点T的新图，将这一组的每个点分别向T连边，容量为该点上的车数，另外对于原图中的一条长度为l的边(u,v)，若$d_v = d_u + l$，则在新图上连边$d_u \to d_v$，容量为 1。那么显然这一组能通过的最大车数就是新图上点 1 到点T的最大流。我们可以直接使用 DFS 增广，由于最后答案不会超过c，因此增广次数不会超过$2c$，总时间复杂度为$O(cm)$。

时间复杂度

$O(cm)$

空间复杂度

$O(n+m)$

2013 D - Factors

题目大意

对于任意一个大于 1 的正整数x，都可以用一个或多个质数相乘的形式表示出来。质因子的排列方案有多种，如20就有$2\times2\times5$、$2\times5\times2$、$5\times2\times2$三种。设$f(x)$表示x的质因子的排列方案数，求最小的x，使$f(x)=k$。

算法讨论

本题考查数学。

设p_i表示第i个质数，$x=\prod_{i=1}^{pn}p_{px_i}^{c_i}$，其中$pn$表示$x$不同质因子的个数，$px_i$表示$x$的第$i$个质因子，则不难计算$f(x)=\frac{(\sum_{i=1}^{pn}c_i)!}{\prod_{i=1}^{pn}(c_i!)}$。因此对于两个数$x_1,x_2$，若它们对应的$pn$和$c$数组都相等，那么有$f(x_1)=f(x_2)$。也就是说，对于同一个$c$数组，我们只考虑那个最小的$x$即可，而这个最小的$x$必然满足$px_i=i,c_{i+1}\le c_i$。因此我们利用 DFS 枚举所有可能的$c$数组并计算出对应的最小的$x$，记下所有有用的$x$以及相应的$f(x)$，并用哈希表、平衡树或排序+二分来回答询问即可。注意到$f(x)$可能较大，若直接使用阶乘进行计算分子部分可能超出 64 位整数范围，因此我们在计算时可以将$f(x)$写成组合数相乘的形式，并利用杨辉三角来预处理组合数来避免除法。由于x和$f(x)$均不超过2^{63}，所以可能的c数组不超过四万种，可以在规定时限内轻松完成预处理。

时间复杂度

$O(n)$或$O(n\log n)$，其中n表示可能的c数组的个数。

空间复杂度

$O(n)$，其中n表示可能的c数组的个数。

2013 E - Harvard

题目大意

一种计算机有n个内存库，按$0 \ldots n-1$编号，每个可以储存m个变量。访问 0 号内存库的代价为 1，访问$1 \ldots n-1$号中的某一个内存库i时，若这是第一次访问该内存库，那么代价为 2，若上一次访问该内存库到这一次访问之间未访问过$1 \ldots n-1$号内存库中的任意一个，则这次访问的代价为 1，否则代价为 2。

现给出一个仅包含访问变量和循环的程序，程序运行的代价是指访问变量的代价之和。你需要给程序中的每个变量分配一个内存库中的位置，求该程序运行的最小代价。

算法讨论

本题考查搜索。

首先分析程序。我们可以通过类似于表达式计算的方法，用一个栈线性维护出变量访问的总次数。这样我们就可以把以上所有代价都减一，看作是访问变量的额外代价。

访问 0 号内存库的额外代价为 0，因此要尽可能放满。当变量个数不大于m时，显然将所有变量都放入 0 号内存库是最优的，此时总代价就是访问变量的总次数。当变量个数超过m时，我们首先枚举 0 号内存库中放入了哪些变量，然后无视程序中对这些变量的访问，重新分析程序。令$c_{i,j}$表示此时的程序中，访问变量i之后紧接着又访问变量j的次数，这也可以通过类似表达式计算的方法，用一个栈在线性时间内求出。这样一来我们就可以利用c数组很方便地求出某种分配方案的代价。接下来只要搜索每个内存库的放入了几个变量以及放入了什么变量即可。搜索时注意以下剪枝：

- 当前已分配的变量计算出的额外代价应小于当前最优解。
- 对每个内存库分配的第一个变量一定是当前未分配的最小编号以避免重复搜索。
- 任意两个内存库的大小之和必须大于m。
- 同一内存库的变量编号依次递增以避免重复搜索。

时间复杂度

$O(v^{nm})$，其中v表示变量个数，实际远远达不到该复杂度。

空间复杂度

$O(v^2 + l)$，其中l表示程序长度。

2013 F - Low Power

题目大意

有n个机器，每个机器有两个芯片，每个芯片可以放k个电池。每个芯片能量是k个电池的能量的最小值。定义一个机器的权值为它的两个芯片的能量的差的绝对值。现在有$2nk$个电池，已知它们各自的的能量，我们需要确定一种放置方案，使得所有机器权值的最大值最小。

算法讨论

本题考查二分答案、贪心。

首先二分答案，设当前需要检查的答案为lim，问题就转化为是否存在一种放置方案，满足每个机器上两个芯片的能量差的绝对值均不大于lim。我们称一个机器第一个芯片上能量最小的电池和第二个芯片上能量最小的电池为这个机器的关键电池。我们将电池按能量从小到大排序，得到$a_1, a_2, \dots a_{2nk}$。不难发现一定存在一种最优方案使得每个机器两个关键电池在排序结果中相邻，于是每个机器的权值一定是某个$a_{p+1} - a_p$。接下来不妨设每个机器对应的p依次递增，那么对于第i台机器，我们可以知道机器$i \dots n$中的任何电池的能量不小于a_p，也就是说所有电池中至少需要有$2k(n-i+1)$个电池的能量大于等于a_p，于是我们得到$p \le 2k(i-1)+1$。接下来我们只需从$1 \dots n$枚举所有机器，并对其分配当前最小的可行的p，若分配到某个机器时不满足$p \le 2k(i-1)+1$，则答案lim就是不可行的；若所有的机器都满足$p \le 2k(i-1)+1$，则答案lim是可行的。

时间复杂度

$O(nk \log nk + nk \log ans)$

空间复杂度

$O(nk)$

2013 H - Матрёшка

题目大意

有n个数$a_1, a_2, \ldots, a_n$排成一行，一开始时每个数自成一组，每次可以选择相邻的两组数合并，最终目标是使每一组数都是$1,2,\ldots,m$（对于不同的组m允许不同）。设待合并的两组数升序排列之后为$a_1, a_2, \ldots, a_p$和$b_1, b_2, \ldots, b_q$，并设$\{a_i\}$中小于b_1的数有c_1个，$\{b_i\}$中小于a_1的数有c_2个，那么合并这两组数的代价就是$p+q-c_1-c_2$。问能否达成目标，若能，求最小代价。

算法讨论

本题考查动态规划。

不难看出$a_i \ldots a_j$能被合并为一组的条件是$a_i \ldots a_j$两两互不相同。接下来关于合并$a_i \ldots a_j$的讨论均在它们能够被合并的前提下进行。

设$cost(i,k,j)$表示在$a_i \ldots a_{k-1}$和$a_k \ldots a_j$都已经各自合并为一组的情况下，合并这两组所需的代价，$f[i][j]$表示从初始情况开始将$a_i \ldots a_j$合并为一组所需的最小代价，不难得到如下转移方程：

$$f[i][j] = \begin{cases} \min\{f[i][k-1] + f[k][j] + cost(i,k,j) \mid i < k \le j\}, & i < j \\ 0, & i = j \end{cases}$$

我们发现转移的数量已经达到了$O(n^3)$，因此我们需要考虑如何快速计算$cost(i,k,j)$。我们可以先按$1 \ldots n$的顺序枚举j，再按$j \ldots 2$的顺序枚举k，并利用类似插入排序的算法维护$a_k \ldots a_j$的排序结果，最后按$k-1 \ldots 1$的顺序枚举i。这样一来在枚举i的过程中，$\min(a_k \ldots a_j)$是固定的，因此可以在$O(1)$的时间内求出$a_i \ldots a_{k-1}$中有多少元素小于$\min(a_k \ldots a_j)$；同时$\min(a_i \ldots a_{k-1})$是单调不增的，因此可以在$a_k \ldots a_j$的排序结果中维护一个游标指向第一个小于$\min(a_i \ldots a_{k-1})$的元素，便能在均摊$O(1)$的时间内求出$a_k \ldots a_j$中有多少元素小于$\min(a_i \ldots a_{k-1})$。也就是说，我们可以在均摊$O(1)$的时间内求出一个$cost(i,k,j)$，于是这个动态规划的总时间复杂度即为$O(n^3)$。实现时我们不需要存下所有的$cost(i,k,j)$，只需按照计算$cost(i,k,j)$的顺序来计算$f[i][j]$即可，因此这一步的空间复杂度为$O(n^2)$。

接下来问题就转化成了把$a_1, a_2, \ldots, a_n$分成若干段，使得每一段都是$1,2,\ldots,m$，代价为合并每一段的代价之和，求最小代价。令$g[i]$表示将$1 \ldots i$分成若干段的最小代价，不难得到转移方程：

$$g[i] = \min\left\{ g[j-1] + f[j][i] \,\middle|\, 1 \le j \le i, \{a_j \ldots a_i\} = \{1 \ldots i-j+1\} \right\}$$

最终答案即为$g[n]$。

时间复杂度

$O(n^3)$

空间复杂度

$O(n^2)$

2013 I - Pirate Chest

题目大意

给定一个底面高低不同的$n \times m$的长方形水池，已知每个单元格的水深，在考虑放入物体之后水面升高的前提下，求最大能浸没（物体顶面不得与水面重合）在水中的长方体物体体积（长方体高度可以为任意整数，底面一边长需小于等于a，另一边长需小于等于b，放置时各边需和池塘对应方向的边界平行）。

算法讨论

本题考查单调栈优化的枚举。

如果我们确定了长方体放置的位置(x_1, y_1, x_2, y_2)，容易发现长方体下潜的最大深度受制于该矩形内的最小水深，设该值为d，令长方体底面长$l = x2 - x1 + 1$，宽$w = y2 - y1 + 1$那么容易算出长方体的最大高度为$h_{max} = \left\lfloor \frac{nmd-1}{nm-wl} \right\rfloor$。枚举$x_1, y_1, x_2, y_2$四个变量的枚举量显然太大，我们考虑先枚举$x_1, x_2$，令$g_i = \min(\{d_{j,i} \mid x_1 \le j \le x_2\})$，那么问题降为一维，即在$g$数组中选一段$[l, r]$，最大化$h_{max} \cdot (r - l + 1)$。我们发现$[l, r]$的区间对应的矩形的最大下潜深度受制于这段中$g_i$的最小值，并且在该最小值$d$不变的情况下，$h_{max}$随矩形宽度$w$增加是不降的，因此我们对于每个$i$，只需求出以$g_i$为最小值的情况下向左向右最多能扩展到的最远位置$l_i, r_i$，即可确定以$g_i$为限制的最大子矩形并更新答案，这可以通过类似于建笛卡尔树的方式使用一个单调栈在$O(n)$时间内实现。

接下来考虑底面一边长需小于等于a，另一边长需小于等于b这个限制。不妨设$a \le b$，那么实际上就是当我们枚举矩形的长小于等于a，即$x_2 - x_1 + 1 \le a$时，即使矩形宽左右扩展的长度超过b，矩形的宽也只能为b；当$a < x_2 - x_1 + 1 \le b$时，矩形的宽最大只能是a。我们将求得的最大扩展到的宽度与对应的a或b取较小值当作矩形的宽来更新答案即可。

我们总共需要枚举$O(n^2)$种情况，处理每种情况的时间复杂度为$O(n)$，因此总时间复杂度为$O(n^3)$。

时间复杂度

$O(n^3)$

空间复杂度

$O(n^2)$

2013 J - Pollution Solution

题目大意

给定一个多边形（所有顶点的纵坐标非负）和一个半圆（原点在圆心，半径为r的圆的x轴以上的部分），求它们公共部分的面积。

算法讨论

本题考查计算几何。

我们首先计算出每条线段与圆弧的交点，这样我们就可以提取出公共部分的边界了。设边界上的点逆时针依次为$A_0, A_1, A_2, \dots, A_m$，我们依次扫描边界上的每一条边$(A_i, A_{(i+1)\%m})$，那么有如下两种情况：

- $(A_i, A_{(i+1)\%m})$是一条线段，这种情况下我们给答案加上$\Delta O A_i A_{(i+1)\%m}$的有向面积，即$\frac{1}{2}\left|\overrightarrow{OA_i} \times \overrightarrow{OA_{(i+1)\%m}}\right|$。
- $(A_i, A_{(i+1)\%m})$是一段圆弧，这种情况下我们给答案加上扇形$O A_i A_{(i+1)\%m}$的有向面积，即$\frac{1}{2}r^2(\theta_2 - \theta_1)$，其中$\theta_1$表示$\overrightarrow{OA_i}$与$x$轴正方向的夹角，$\theta_2$表示$\overrightarrow{OA_{(i+1)\%m}}$与$x$轴正方向的夹角。

最后答案即为所求面积。

时间复杂度

$O(n)$

空间复杂度

$O(n)$

2013 K - Up a Tree

题目大意

有如下三段代码：

```
void prePrint(TNode t) {
    output(t.value);
    if (t.left != null) prePrint(t.left);      (1)
    if (t.right != null) prePrint(t.right);    (2)
}
```

```
void inPrint(TNode t) {
    if (t.left != null) inPrint(t.left);       (3)
    output(t.value);
    if (t.right != null) inPrint(t.right);     (4)
}
```

```
void postPrint(TNode t) {
    if (t.left != null) postPrint(t.left);     (5)
    if (t.right != null) postPrint(t.right);   (6)
    output(t.value);
}
```

不难看出这分别是输出一棵树的先序、中序和后序遍历的三个函数。现在由于一些奇怪的问题，代码中(1)…(6)这 6 处 prePrint/inPrint/postPrint 的顺序被打乱了，于是程序产生了错误的输出$s_{pre}, s_{in}, s_{post}$。现在要求根据这三个输出推断出哪些打乱方案是可行的，并求出每种可行方案对应的字典序最小的树。比较两棵树的字典序的方法是先比较它们的先序遍历，再比较它们的中序遍历。

算法讨论

本题考查记忆化搜索。

我们首先枚举打乱方案，总共有$C_6^2 \cdot C_4^2 \cdot C_2^2 = 90$种情况。

接下来我们分别处理每一种情况。令$S = s_{pre} + s_{in} + s_{post}$，$f[a][b][c][len]$表示 prePrint 输出为$S[a \dots a + len - 1]$，intPrint 输出为$S[b \dots b + len - 1]$，postPrint 输出为$S[c \dots c + len - 1]$时字典序最小的树是什么（至少需要记录先序遍历和中序遍历），若a, b, c的某一项或两项是 0 则表示对应的输出不存在。那么最终答案即为$f[1][n + 1][2n + 1][n]$。计算f时我们使用记忆化搜索。对于$f[a][b][c][len]$，我们首先确定其左子树大小，此时需要按a, b, c是否存在分七种情况进行讨论，其中b存在的四种情况可以直接确定左子树大小，而其余三种则需要枚举左子树大小。接下来根据左子树大小确定根以及两个子树的$s_{pre}, s_{in}, s_{post}$然后分别搜索两个子树即

可，最后取字典序最小的答案并记录到$f[a][b][c][len]$。另外，计算的过程中要注意判断无解的情况。

关于时间复杂度，理论上的计算如下：a,b,c均存在的状态总共有$O(n^4)$种，对于这些状态我们都可以直接确定做子树大小，因此处理一个状态的时间复杂度为$O(n)$； a,b,c至少有一个不存在的状态总共有$O(n^3)$种，处理时可能需要枚举左子树大小，因此处理一个状态的时间复杂度为$O(n^2)$。因此，总时间复杂度上限为$O(n^5)$，但实际上由于状态数远没有计算的那么多，并且很多状态可以明显地判断出无解，因此该算法的实际表现比较理想。空间复杂度方面，由于我直接使用数组$f[a][b][c][len]$来记忆化，因此空间复杂度达到了$O(n^4+n_{sta})$，其中n_{sta}表示用到的状态数。

时间复杂度

$O(n^5)$

空间复杂度

$O(n^4+n_{sta})$，其中n_{sta}表示用到的状态数。