浅析信息学竞赛中一类与物理有关的问题

杭州学军中学 方戈

摘要

目前，信息学竞赛中出现许多与其他学科有关联的问题，这也是信息学竞赛发展到一定阶段的必然趋势。而物理，作为一种实用性很强的学科，与信息学也有着越来越紧密的联系，许多信息学竞赛中的问题都或多或少跟物理有联系。而这类与物理有关的问题，正是本文研究的重点。首先，本文将阐述研究与物理有关的问题的重要性，并提出些典型的有物理特色的问题类型。然后本文将重点解析几道典型而有趣的问题。最后，本文将简单介绍作者在研究这类问题时碰到的困难以及一些心得与体会。希望本文能对同学们更好地理解此类问题有一定帮助。

关键字

信息学竞赛 物理 平衡

目录

正文

序言

虽然我们讨论的是有关物理的题目，但事实上他们并不需要很多物理的知识，实际上它们中有很多只是用物理做了下背景而已。这些题目的种类很多，比如像一类“倒水”的问题，又比如牵扯到“平衡”或“稳定”等字眼的一类问题，而随便引入几个物理概念，就又能编出一类完全不同的题目了，可谓是五花八门，千奇百怪。

但经观察又可以发现，这些题目还是有些共性的，比如，它们的背景一般是三维的，这就需要我们对空间有很好的感觉，至少我们要能想象出题目描述的是怎么样的情况，这需要我们对题目有很好的感性认识。

而然后，这类题目的做法也不是非常好想的，得到算法是一个盘旋上升的过程，而这中间需要很严谨的思维，对于每一个结论，也必须有严密的证明，这就需要我们对题目有很好的理性认识。

因此说，这类问题是需要感性和理性的结合的。它们考察的是一个人的各方面综合素质，而从另一个角度上，它们的确涉猎了各个方面的算法——计算几何，数学、动态规划、搜索、模拟、枚举，而考察各方面能力也正是信息学问题的发展趋势，这也是目前此类问题非常重要的原因之一。

同时，由于背景是三维的，这些问题也就更贴近实际，更具有实用价值，而不像很多数学题，动态规划题，只是思维的游戏，它们在这个三维的世界上有着更好的用处，而这，就是此类问题非常重要的另一个原因。

这类问题通常是非常有趣的，并且它们的算法实现非常简单，程序不怎么长，但他们的算法设计需要的是一个很长的一个过程，其间更需要的是有条理的分析和很好的耐性，下面就来看一些例题吧。

例一**[Ars Longa](World Final 2006 Problem C)**

题目大意

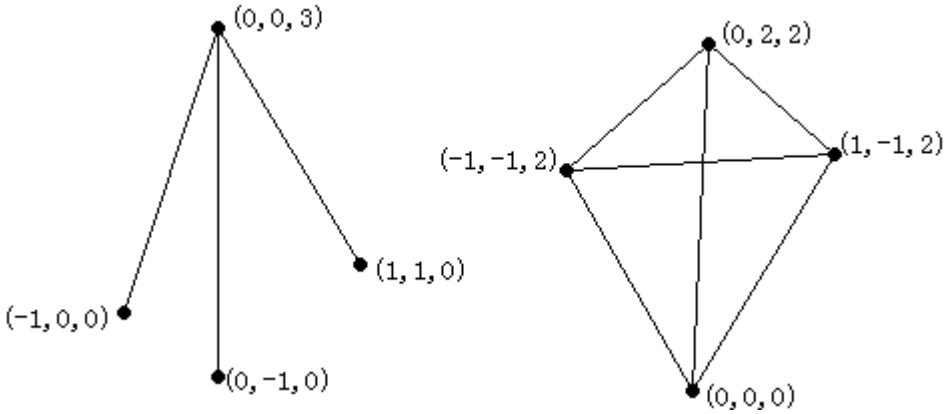
在空间中有一个结构，由 N 个点和一些连接他们的边组成，给出 N 个点的位置，点质量均为 1KG，点的 z 坐标是非负的，z 坐标是 0 的点是粘在地上不能移动的，而 z 坐标大于 0 的点可以随意地移动。给出 M 条边，故略边的质量，边十分牢固，可以承受无限大的力，忽略边之间的作用力。每条边连接两个点，边可以绕着点无摩擦地转动，长度即为连接的两个点的初始位置之间的距离。边对点的作用力可以是推，也可以是拉。

题目的保证不会有这样的退化数据，如一个点只收到水平的边的作用，这会造成一些很难解决的情况。

称这个结构是平衡的，如果这个结构在重力作用下不会移动。

称这个结构是稳定[1]的，如果这个结构是平衡的，并且无论怎么给结构中的点施加怎么样的力（可以给多点同时施力，并且力可以不同）也不会使雕塑移动。

现在要确定这个雕塑是否平衡或稳定。其中 N≤100，M≤100。



上面是两个例子，括号中是点的坐标，其中左图是稳定的，就像一个三角架，而右图是平衡而非稳定的，上面三点的重心刚好在地面点的正上方，因此它能保持平衡，但显然只要推它一下，它就倒了。

初步分析

这是一道非常贴近实际的题，因此我们对它会有许多的感性认识。显然的，这需要用到物理中有关物体平衡的简单定理:

保持静止的物体是平衡的。

平衡物体所受到的合力为 **0**。

这样点的物理模型就出现了，然后是边的，注意到题目中的一句重要的话: 边可以绕着点无摩擦转动，这个实际上说明了边对点的作用力是在边所在直线上的。边是理想化无重力的模型，因此边对两个点的作用力是相等并且相反的

那么，整道题的物理模型就构建好了，然而此物理模型中包含着复杂多样的情况，我们无法用计算机对此物理模型进行分析和计算，因此我们需要对模型进行一些转化。

物理模型的转化

所受合力为 0，这样的判断句给我们的第一感受就是可以转化为等式。事实上，如果把每个边所承受的力用未知数表示的话，那么对于某个在空中的点，它的 x，y，z 坐标均可以建立一个平衡，这实际上就把平衡物体的物理模型等价转化为了三个等式——数学模型。

[1] 原题中所说的是“稳定(stable)”，但在物理学中，更严谨的讲法是“刚性(rigid)”，下同

其中边的力也就是未知数的数值，而边对两点的推和拉也就是未知数的正负符号。

若空中点数有 P 个，那么对于每个空中的点都这样转化就会出现 3*P 方程，这些方程有 M 个未知数，这些方程组成一个 M 元一次方程组。至此物理模型就完全转化为了数学的模型。具体转化如下：

设方程组为：

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \cdots a_{1,M}x_M = 0 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + \cdots a_{2,M}x_M = 0 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + \cdots a_{3,M}x_M = 1 \\ \vdots \\ a_{3P-2,1}x_1 + a_{3P-2,2}x_2 + a_{3P-2,3}x_3 + \cdots a_{3P-2,M}x_M = 0 \\ a_{3P-1,1}x_1 + a_{3P-1,2}x_2 + a_{3P-1,3}x_3 + \cdots a_{3P-1,M}x_M = 0 \\ a_{3P,1}x_1 + a_{3P,2}x_2 + a_{3P,3}x_3 + \cdots a_{3P,M}x_M = 1 \end{cases}$$

其中第 3i－2，3i－1，3i 个方程分别表示第 i 个悬空点的 x，y，z 方向上的平衡。关于 z 方向平衡的方程式右式为 1（只要是一个相同的非 0 数即可）。

对于第 i 条边，假如它连接了第 p 个悬空点和第 q 个悬空点（如果它连接了非悬空点，则忽略相应的操作），并且两点之间的向量为(x_0, y_0, z_0)，那么 $a_{3p-2,i}=x_0$，$a_{3p-1,i}=y_0$，$a_{3p,i}=z_0$，$a_{3q-2,i}=-x_0$，$a_{3q-1,i}=-y_0$，$a_{3q,i}=-z_0$，其余的 $a_{*,i}$ 均为 0。这些赋值保证了该边对两点的力的方向与杆的方向一致，并且对两点的力相反，因此 x_0，y_0，z_0 都成倍增加或减少也是成立的，若表现在方程上，成倍增加或减少也不会影响方程的有解性。

可以看到，这里方程的个数是 O(N)的，未知数的个数是 O(M)的。

平衡及稳定的判定

由于我们的等价转化，结构的平衡实际上与我们得到方程组的有解是等价的，因此，我们只需要解方程，即可判定结构是否平衡。解方程就可以用高斯消元法来做了。[2]

高斯消元消到最后，会出现一些（也有可能没有）左边系数全为 0 的方程，如果这些方程中某一个的右边是非 0 数，显然方程无解，否则，因为我们对解没有限制，所以一定有解。这实际上就把不平衡的情况判定出来了。

稳定的判定看上去比较复杂，但经过分析也会发现这只是一只纸老虎。

稳定的定义，转化到方程上就是对方程右边的数稍加调整，能否还保持有解。我们首先的想法就是模拟，对每个方程给它变一变，看有没有产生影响。然而模拟的代价是巨大的。每次都要重新地解一次方程来看是否有无解的情况，这会使最终的复杂度成倍增加。有没有不需要再解方程就判定稳定的方法呢？如果仔细琢磨，不难发现以下规律：

高斯消元后，若存在左边系数全为 **0** 的方程，则不稳定，否则就稳定。

证明：若存在左边系数全为 0 的方程，那么若调大或调小原来此方程等号右边的数值，会造成消元后此方程等号右边的数不等于 0，即不平衡，因此也

[2] 关于高斯消元法，可以参见以前的国家集训队论文。

就不稳定。

若不存在左边系数全为 0 的方程，那么无论右边系数怎么变化，左边系数在高斯消元后还是不变的，因此在任何情况下都不存在左边系数全为 0 的方程，即在任何情况下都平衡，也就是稳定。

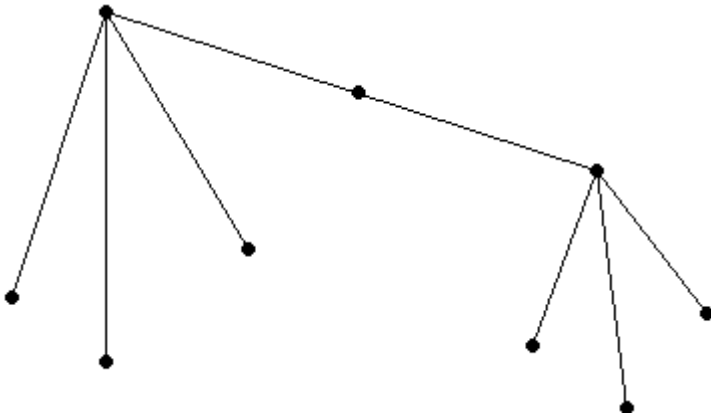
证毕。

至此，问题解决，此算法的复杂度，即为高斯消元法的复杂度，对于 O(N)个方程，O(M)个未知数，高斯消元的复杂度为 O(N*M*M)。对于 N=100,M=100 的极限数据，此复杂度还是非常宽松的。

对于退化数据的一些研究

在研究有物理特色的问题时，严谨是必要的，对于此题，虽然原题中尽量避免出现退化数据，但只要仔细琢磨，还是能想像出一些会退化的数据：

其实只要是某个点，与它相邻的边全在某一个不与地面平行的平面上，就很容易出现退化的情况。比如说像这样的一种简单例子：



上图中在空中有 3 个点，在旁边的两个点由于有地面上三角架的支撑是稳定的，这时候中间的点如果动的话，那么连接它的两边至少有一个会发生长度上的变化，这是不可能的，而它自己，由于两个边方向相同且不在垂直方向上，它又受到了重力的作用，它是不平衡的，它会移动，这就矛盾了。

对于这个矛盾，我是这样理解的，某条边对该点的拉伸，除了在此边方向上的拉伸，再分出一个竖直方向的 0 向量，由于拉伸可以是无穷大的，而无穷大乘以 0 为一常数，那么可以用这一常数把该点支撑。或者也可以这样理解，在边上的力趋向于无穷大时，点移动的距离会越来越小，趋向于 0，因此当边上的力为无穷大时，点的移动为 0。点不移动的话，就可以说此结构是平衡的，进而也可以说它是稳定的。

然而由于实际中，边上的力是无法达到无穷大的，因此，我更觉得题目中“边可以承受无限大的力”应该“改成边可以承受足够大的力”，而出数据时应该杜绝这样的退化数据，因为对于这样的数据，还是不能有很好的算法来分析其最终的情况，比如按我们的算法，最终对于上面提出的例子就会算出不平衡。即使有，我认为这些讨论也破坏了算法本身的优美。事实上要杜绝这样的数据，只要规定任何点所连的所有边，不能在同一个平面上，以下情形例外：该平面垂直于地面，或者这些边在与过该点某平面的同一边。

小结及拓展

反观解题的全过程，从物理模型到数学模型是最为重要的一步，对于大多数物理问题，物理模型建立是很简单的，但是对于这样抽象的模型是无法提出有效算法的的，因此势必要对其进行转化。本题的算法及结论可谓美妙，通过转化，我们也发现了高斯消元法在解决实际问题中的价值。而对此题退化情况的分析，让我们对此题的解决更为严谨。

此题也不乏有向多方面拓展的潜力，如原题中，若对每条边所承受的力的最大值加以限制，那么表现在方程中又会是怎么样一个情形呢？此时平衡和稳定与方程的关系又是怎么样的呢？这样的限制加了以后，问题就更加实际，也更加需要往深处分析了，这也是此类问题的魅力之一吧。

例二**[water tanks](World Final 2007 Problem I)**

题目大意

有 N 个圆柱形容器，它们的底面完全相同——均为一单位面积，它们排成一列，并且已知它们的高度 col_H，第一个管子的上端是开口的，其余的均为密封，任两个圆柱中间有管子相连，管子与地面平行（如下页图），已知第 i 个容器和第 i + 1 个容器之间的管子的高度为 pipe_H[i]，保证 pipe_H[i] ≠pipe_H[i + 1]。管子足够粗使水以及空气能够顺利通过管子，管子足够细使得不必计算它们中间的水的体积。现缓慢往第一个管子倒水，过程中一切满足以下规律（理想化的物理模型）：

·开放的空气中气压为一个大气压。

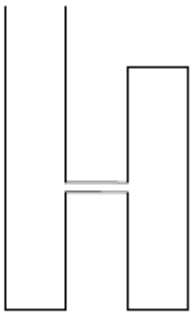
·水不能被压缩，空气能被压缩。

·密封的空间内处处气压相等，密封空间的体积改变时，气压改变满足以下规律 $P_1V_1=P_2V_2$,其中 P_1，V_1 为压缩前的气压和体积，P_2，V_2 为压缩后的。

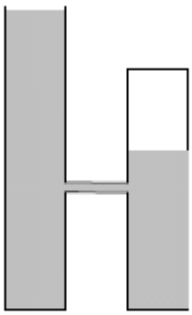
·一单位高度的水柱产生的压力为 KP=0.097。

·相连的水在同一个高度的水压是相等的。

现问：第一个管子注满水时，共倒了多少单位体积的水?



Tanks before filling

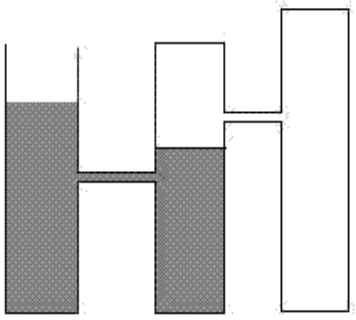


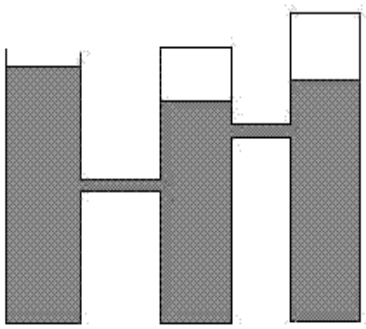
Tanks after filling

上图为一个只有两个管子的例子，N≤1000000。[3]

初步分析

这题看上去感觉并不复杂，对于原来的规模，只要按照那些性质直接模拟即可。我们的感性认识告诉我们，对于每个管子，只需要在气压和水压上建立一个平衡，就可以了，但事实上，对这题的感性认识骗了我们自己，即使是模拟，由于很多量在变化，每个容器的气压涉及到很多因素，其间需要很多的讨论和判断。比如说下面的这种的情况。





当第二个容器中的水低于第三个容器时，第二个容器的剩余空气与第三个容器中的空气是连通的，所以当第二个容器的水升高的时候，第二个容器和第三个容器的气压是共同升高的，而当我们继续倒水变成右上图所示情况的时候：两个容器的空气就隔绝了，此时两个容器的气压就需要分别处理。而显然，当容器数增多的时候，情况就远不止这样了。本题的难点就在于如何处理这些情况。

一些简单的性质

本题情况多样，对于本题，一种经典而又朴素的方案便是繁琐的讨论了，

[3] 此题与原题有些区别，条件限制和数据规模上都有所加强，原题有个限制是横管的高度是从左到右递增的，并且数据规模只有 10，详见附录中的英文原题。

但是刚切入题目便讨论，必然会对很多特殊情况漏判或者误判，很难做到严谨，而这正是这类题目所最为需要的。我们需要的另一种简洁明了的方案，而得到方案的路必然是盘旋上升的，想一步登天是不可行的，即使是讨论，也需要对本题的一些性质和规律有深入的了解，所以首先看一些简单的结论。

设第 i 个容器中水面高度为 H[i]，那么有：

性质一：**H[i] < pipe_H[i]**时，**H[i + 1] = 0,**其中 **i ≠N**。

性质二：**H[i + 1] < pipe_H[i]**时，左右连通。否则，左右不连通。

性质三：第 **i** 个容器若和左右都不连通时，它上方空气的体积和气压的乘积就确定下来了，此时 **H[i]**仅和最高水柱高度即 **H[1]**有关，可单独考虑。

以上两条性质均可直接获得，虽然他们看似废话，但他们是我们继续探索的基础。由它们可以继续得到以下的结论：

性质三：**0 < H[i + 1] < pipe_H[i]**时，**H[i] = pipe_H[i]**。

证明：如果 H[i] < pipe_H[i]，与性质一矛盾。

如果 H[i] > pipe_H[i]，水会往右边流，直到左边水面高度等于横管高度或者右边水波面高度不小于横管高度。证毕。

这些简单的信息事实上就高度概括了水流的规律，而只有通过这些规律才能题目开刀。

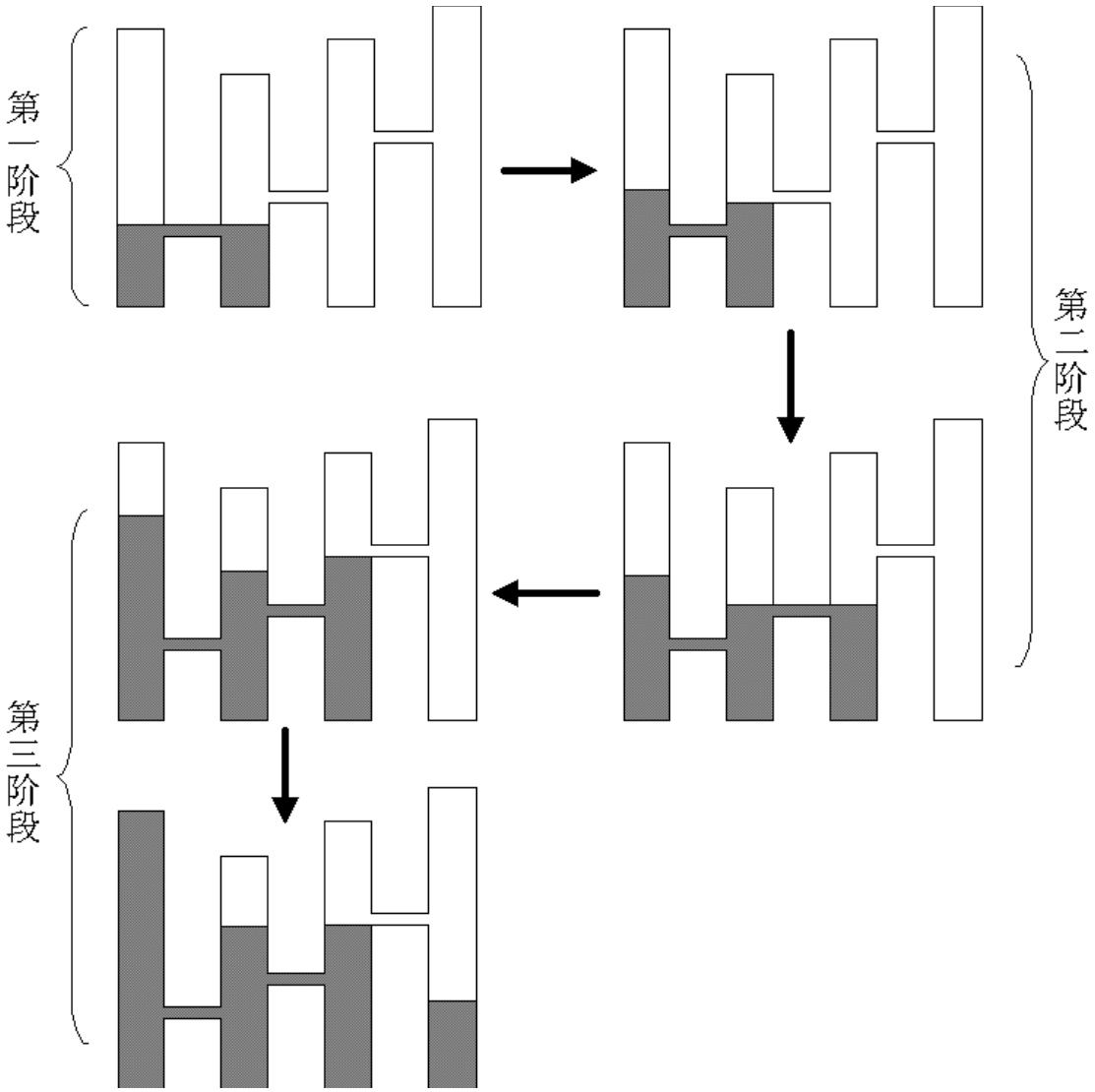
对一类特殊情况的分析

仅仅知道水怎么流是不够的，我们还是很难直接提出算法，我们不妨先来看一个特殊的情况，再想办法从特殊推到一般。

把从原题删去的限制条件加回去，也就是多了这么个限制：横管的高度是从左到右递增的。

这个限制条件的添加事实上给我们的解题提供了很大的便利，从感性认识上讲，就会觉得少了很多情况，事实也正是如此，容器中水的上升情况的规律很容易就能发现：

假设第 1 个容器无限大，那么在往第 1 个容器中倒水的过程中：首先 H[2]到达 pipe_H[2]后，H[3]开始上升，上升到 pipe_H[2]后，容器 2，3 隔绝，容器 2 中的空气达到密封，H[2]可单独考虑，然后 H[3]继续上升，上升到 pipe_H[3]后，H[4]开始上升，上升到 pipe_H[3]后，容器 3，4，隔绝，H[3]可单独考虑……直到最后一个管子 H[N] = pipe_H[N – 1]。



观察整个过程，其中的关键点是相邻的两个容器的隔绝，因为它导致了两部分气压变化的分化，鉴于此，我们称 H[i] = pipe_H[i – 1]时到 H[i + 1] = pipe_H[i – 1]时的变化为第 i 阶段的变化，又因为没有 H[0]，不妨就令第一阶段的变化为一开始到 H[1] = H[2] = pipe_H[1]的变化。上图是某个变化阶段的例子，其中第三阶段的第二步还未完成时，第一个容器已经满了，因此第三阶段结束

对一类特殊情况的算法

然后，一个简明算法就能设计出来了，其中 i 表示目前的阶段数，V 表示目前第 i 个容器右边（不包括第 i 个）的空气的体积，P 表示其气压，其中的 col_H[1]即为最终的水柱高度，用来计算压强。

step 1：初始化 H[1] = H[2] = pipe_H[1]，i = 1，V 赋为总体积-col_H[1]-H[2]，P = 1。

step 2：i ++，若 i = N，计算 H[i]最终高度，跳入 step5。若 H[i]在从 pipe_H[i – 1]上升到 pipe_H[i]的过程中能达到最终水压和气压的平衡，即存在 delH 使得

0≤delH<pipe_H[i] – pipe_H[i – 1]，并且(col_H[1]-H[i] – delH)*KP + 1 = V * P / (V – delH)。那么 H[i] += delH，跳入 step5，否则 H[i] = pipe_H[i]，更新 V，P。

step 3：如果 H[i + 1]在从 0 上升到 pipe_H[i]的过程中能达到最终水压和气压的平衡，即存在 delH 使得 0≤delH<pipe_H[i]，并且(col_H[1] – pipe_H[i]) * KP + 1 = V * P / (V – delH)。那么 H[i + 1] = delH，跳入 step5，否则 H[i + 1] = pipe_H，更新 V，P。

step 4：计算 H[i]的最终高度，跳回 step2。

step 5：统计答案并输出。

其中要注意解一元二次方程的时候用最小的非负解。

算法中 step2~step4 就是对阶段的循环处理，他们的复杂度均为 O(1)，最多 N 个阶段，复杂度为 O(N)，而 step1，step5 的复杂度也为 O(N)，因此总的算法时间复杂度为 O(N)，空间复杂度也为 O(N)，至此这一类特殊情况已解决。[4]

从特殊到一般

回到一般的情况，我们已经知道特殊情况的解决方案，要解决一般的情况，一种方案就是把一般的情况通过某种方式归为特殊的情况。就是说，要从无序的 pipe_H 中造出一个“递增”来。

事实上，这个转化的方式并不难想到。这里引入一个块的定义：

一个块是一段连续的容器 i~j，满足以下性质：

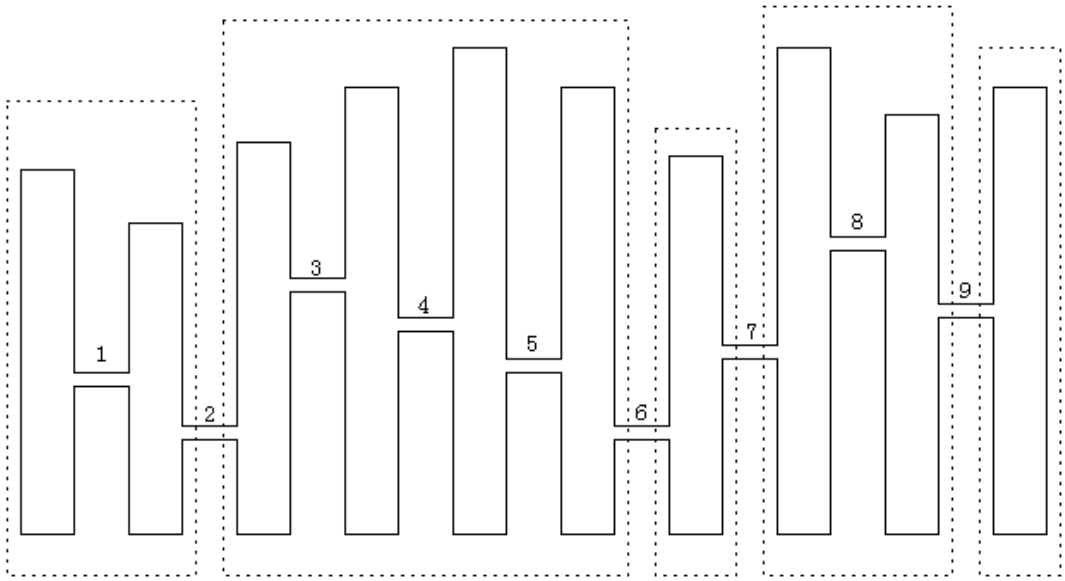
1： **pipe_H[i] > pipe_H[i + 1] > pipe_H[i + 1] >……>pipe_H[j]**。(不妨设 pipe_H[N] = 0)

2：**pipe_H[i] > pipe_H[i – 1]**或 **i = 1**。

3：**pipe_H[j] < pipe_H[j + 1]**或 **j = N**。

对于一般的情况，一个块就相当于特殊情况中的一个容器，而特殊情况相对于一般情况，就是每个容器都是一个块。下图是某个一般情况的块（用虚线框出的部分）：

[4] 程序请参见附录中 tanks_1.cpp。



在一般情况的块中，也没有特别显著的递增，但是它满足任两个块连接处的 pipe_H 是递增的——如上图中 2-3，6-7，7-8，这个的意义在于当前这个块后面的一个容器的 H 到达一定高度时，两个块的空气就隔绝了——如当第三个容器中的水上涨到 2 的高度时，第一个块和第二个快就隔绝了。而我们构造特殊情况所要达到的目的即在于此，因此可以说我们构造出了一个“潜递增”。

一般情况的解决

一般情况的解决就在于一个块和特殊情况中一个容器的对应，相对于特殊情况中的阶段，我们不妨先把一般情况阶段划分出来：

设第 i 个块的起始容器为 pos[i]，那么 H[pos[i]] = pipe_H[pos[i] – 1]为第 i 个阶段的起点（对于第 1 个阶段，起点为 H[1] = 0），H[pos[i + 1]] = pipe_H[pos[i + 1] – 1]为这个阶段的终点，也就是下个阶段的起点。

套用特殊情况的算法框架后，我们只要把某个阶段的算法从对一个容器的处理改成一个块的处理，也就是要找出一个块中的水面变化规律，由于一个块中的容器之间的 pipe_H 满足递减，这个规律并不难找到，设这个块从第 i 个容器到第 j 个容器，那么有：

1：H[i]从 pipe_H[i – 1]上升到 pipe_H[i]。

2：H[i + 1]从 0 上升到 pipe_H[i + 1]。

……

k：H[i + k – 1]从 0 上升到 pipe_H[i + k – 1]。

……

j – i + 1：H[j]从 0 上升到 pipe_H[j]。

j – i + 2：H[j + 1]从 0 上升到 pipe_H[j]，导致这个块与下一个块的空气分隔。

以上只是水压无限上涨的情况，事实上如果任意一步使得水压和气压达到了平衡，都可以直接跳出。

这里的操作就相当于特殊情况中的 step2 和 step3 了，如果没有跳出，顺利

与下个块隔绝时，就要对这个块进行密封后的处理，相当于特殊情况中的 step4，这个的处理就没有特殊情况中那么简单了，因为块中的空气还会随着水面的上涨而隔绝：

1：H[j]从 pipe_H[j]上升到 pipe_H[j – 1]，如果其间达到平衡，跳出，否则会造成 j 容器中的空气与块中剩余其它空气的隔绝，此时可直接算出 H[j]最终的高度，然后把气压和剩余空气体积更新。

2：同样的方法对 H[j – 1]从 pipe_H[j – 1]上升到 pipe_H[j – 2]进行处理。

……

k：对 H[j – k + 1]从 pipe_H[j – k + 1]上升到 pipe_H[j – k]进行处理。

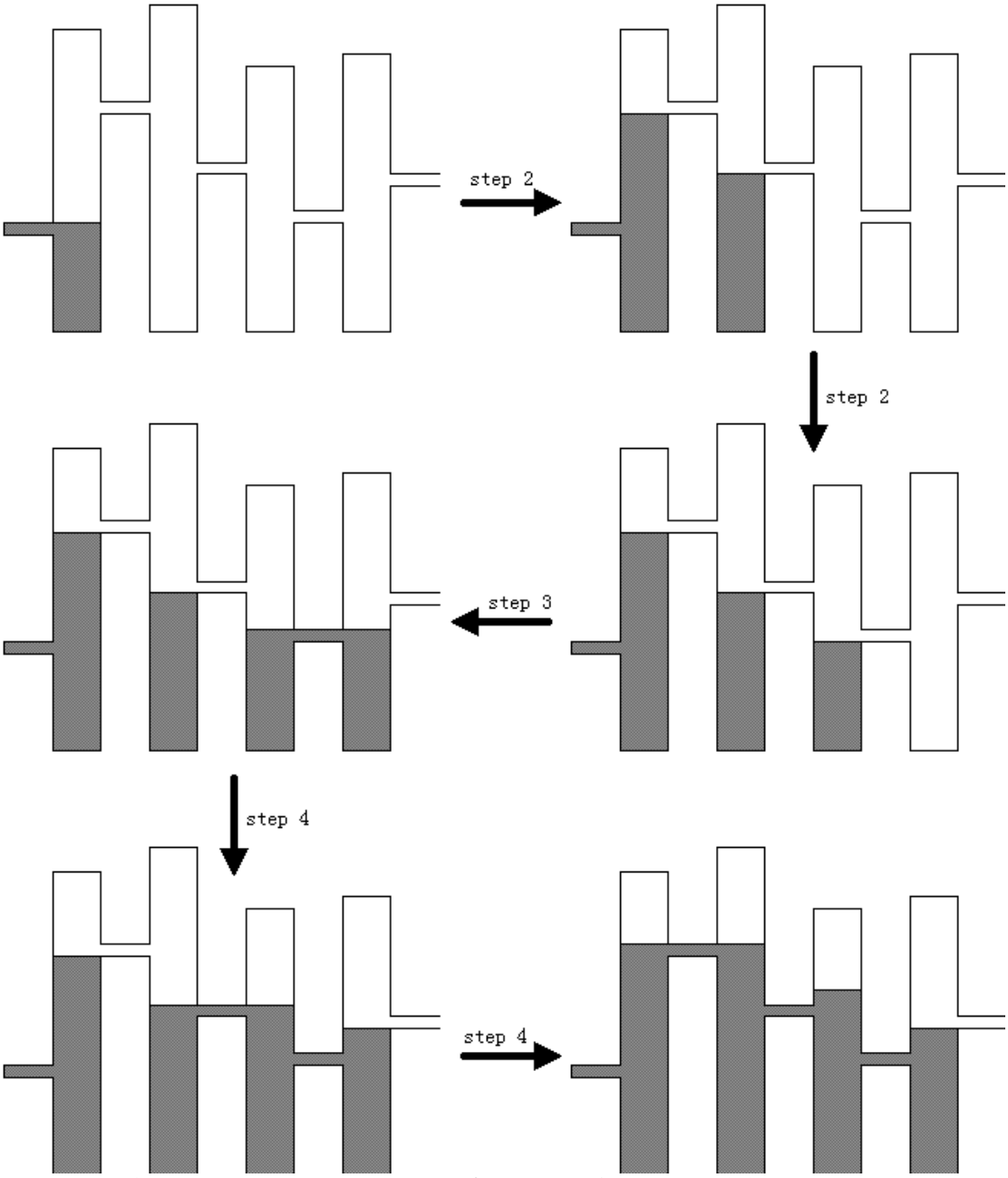
……

j – i：对 H[i + 1]从 pipe_H[i + 1]上升到 pipe_H[i]进行处理。

j – i + 1：此时 i 容器中的空气已隔绝，直接可以求出答案。

这样，相对于特殊情况算法中的 step4 也顺利地解决了。

对某个块的 step2~step4 操作可以从下图中更加直观的看出来：



其中要说明的是，上图是某个局部，默认水压无限增大，上图中的块是前三个容器，最后一个容器是下一个块的，其中箭头上的 step i 表示这步转化相当于一般情况算法中的 step i。

把这些步骤嵌入到特殊情况的算法框架中，就能得到一个适用于一般情况的算法，具体的算法描述这儿就不再赘述了，相信读者自己能够理出来。

分析一下新算法的复杂度：对于一个阶段的处理的复杂度，step2 +step3 = O(j – i)，step4 = O(j – i)，总的复杂度即为 O(块的大小)，因此所有阶段的复杂度，即为 O(所有块的大小的和)即为 O(N)，因此总的时间复杂度还是 O(N)，空间复杂度也是 O(N)，至此原问题也顺利地解决了。[5]

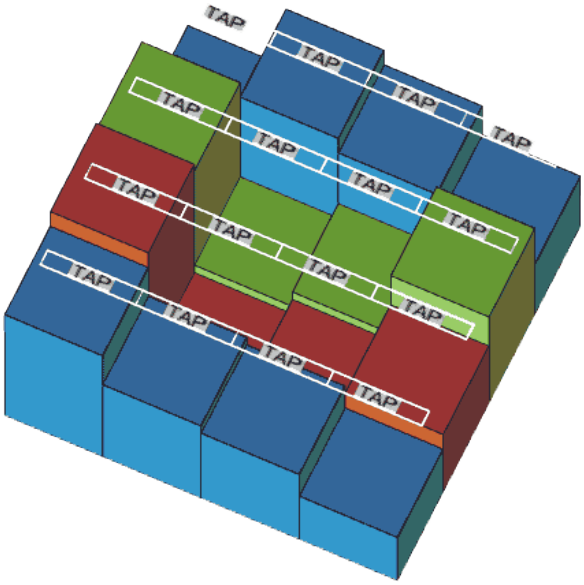
[5] 程序请参见附录中 tanks_2.cpp

小结

通过一步步的分析，我们最后得到了一个简洁高效的算法，达到了原先难以想像的效率，这中间的过程，一步步的衔接也是自然而简单，而不是靠运气。其实此题原题并不难，特别是原先的数据规模只有 10，但要了解此题的精妙所在，就免不了对算法及原题一而再再而三的琢磨。想象如果直接给出改过的题目大概很多人就会手足无措，没法切入。在这种情况下，我们应该由简单的结论想起，从特殊的情况想起，慢慢发掘出事物变化的规律，最终对问题加以解决。对于很大一部分问题，特别是这类有“物理”味的问题，这是一个通用而有效的解决问题的办法。

例三[3D Musical Water-fence](陈启峰，PKU3242)

题目大意



有一个 N*M 的地图（如右图），每格均有一个高度，任两格的高度不同，高度从 1～N*M，周围的高度可以当作 0，现有 T 个有序的倒水的操作，pour(i,j,k)表示往第 i 行第 j 列的格子中倒入 k 单位体积的水，显然，水从高处往低处流，严格的讲，分为以下两种情况：

1：$b_{i,j}$ 中没水且它比某一个或一些相邻格目前的水位高：

这种情况下，k 单位的水会均匀得倒入它的相邻格中，若满足条件的相邻格有 sum 个，那么对每个相邻格都倒入 k/sum。

2：$b_{i,j}$ 中有水或它的高度没有比相邻的单元格水位高：

这种情况下，倒入的 k 单位水会与 $b_{i,j}$ 中原有水及与其相连的水混合导致水位升高直到不能再升高，因为旁边也许有缺口，限制水位的继续升高，而多余的水会从缺口按照第一种情况往下倒。

任务是求出所有操作后浪费的水，即倒在地图外的水。

数据规模如下：N，M≤1000，T≤1000000，k≤1000000。

初步分析

题目中给出了严格的水流方案，这自然而然让我们想到了模拟：按顺序对每一个操作进行实现，递归实现题目描述中的第一种情况，第二种情况则麻烦点，要对整个有相同水位的区域都进行水位提升，具体实现可以见后面描述的第一种算法。在操作过程中，维护目前各个位置的水位情况。这个方案简单并且正确，但它的效率让人大皱眉头。因为这个递归会造成操作指数级增长，不要说数据中最多有 10^6 个操作，即使只有 1，2 个，我们也无能为力——想像这样一种数据，地图的中心是最高点，然后盘旋往下降，就像一座山，如果在山顶浇一点水，那么递归层数实际上就能达到 500 左右，每次即使只增加两倍，也是天文数字中的天文数字了。

而除去模拟，我们也找不到其他切入点了，鉴于这题超级庞大的数据，我们看来需要一段漫长的优化之路了。

第一步优化——大胆猜测

这题是一道非常实际的题，这有一点好处，就是能让人对它有很好的感性认识，对第一步优化，我们需要利用我们对此题的感性认识，大胆地进行一些猜测。

先来分析下我们现有的算法，我们先要想办法提高第一种情况的处理速度，而其中递归是一道必须冲破的障碍，因为即使对其他的因素再优化，我们只要用了递归，就不能高效完成任务了，因此先想一种能代替递归的处理办法是正道。

想像一下这题操作时的图景，我们不难提出这样的一个问题：任意的两个操作，交换其顺序，对情况是否有影响呢？再用感性的认识想想如何会有反例吧，比如方块 1，方块 2（不妨称原始高度为 i 的方块为方块 i，下同）相邻，第一个操作把方块 1 水位提到了 2，第二个操作在方块 2 上倒水，如果先进行第一个操作再进行第二个，那么第二个操作的水不会落到方块 1 上，但如果先进行第二个操作，第二个操作的水就会落到方块 1 上，但再做下去呢？再给方块 1 的倒的水就会满出来，多余的水从高度为 2 的方快上流出去，最终两种操作方式对水位的影响是完全一致的。因此，我们不妨顺着感性思维，大胆提出一下猜想:

对两个操作的交换是对结果没有影响的。[6]

由这个结论，还能够推出下面这个非常重要的推论:

对所有操作的任何顺序的操作都对结果没有影响。这是因为我们只要不断地交换相邻的两个操作，就能达到任何顺序。

这个结论对于本题的优化意义在于：我们能对操作进行汇总处理及延迟处理。比如说，有两个操作都是往方块 6 上倒水，那么这两个操作就能合并成一个；又如，方块 5 和方块 4 都与方块 3 相邻，有两个操作一个往方块 5 上倒水，一个往方块 4 上倒水，两次倒水都会倒到方块 3 上一部分，但我们倒了一部分以后，

[6] 关于此结论的证明，请参照附录。

不需要立刻模拟这部分在方块 3 的下一步会怎样，我们可以在下一部分也倒下来以后一起处理，甚至可以加到原本往方块 3 上倒水的操作中。这就避免了递归让我们能得以对此题的算法进行进一步的优化。

第一种算法

光光能对操作汇总，延迟还是不够的，我们还是无法保证解决第一种情况全部复杂度，比如，有两个方块一个低，一个高并且相邻，那么我们若先处理了低的，再处理高的，那么高的中的水会又流到低的上使得我们不得不再次去处理低的，如果后来又在一个更高的方块上，那么我们不得不对这两个方块都再处理一遍，那对于总方块数高达 10^6 的数据就无法胜任了。因此我们还需要对操作进行必要的排序，这也是上面的结论的另一个优化意义。

事实上，一个简单而有效的排序方法就是把操作按照所倒的方块的高度从大到小排序，因为水不能从低处到高处，如果忽略第二种情况造成的影响，我们的每次对当前方块进行操作后均不用再次操作了，因为后面的方块均比当方块小。由于每次操作的复杂度为 O(1)，最多 O(N*M)个方块，这里的复杂度共为 O(N*M)，这一步能够满足题目的要求了。如果对第二种情况直接模拟的话，我们就能提出一个像样的算法了：

step 1：对所有操作在每个方格上的总倒水量进行统计。

step 2：从高到低的顺序对每个方格处理，处理中顺便统计 ans，即流到边界外的水量。i 表示目前在对方格 i 进行操作，初始 i=N*M，不断进行操作直到 i = 0 结束，跳入 step3。操作包含一下两种情况：

Case 1，方格 i 存在比它低的邻格：按照题目中对第一种情况的描述直接模拟。i --

Case 2，方格 i 不存在比它低的邻格：bfs 得到水位与方格 i 相同的区域（当然包括这个方格自己），形象地讲，这相当于一个湖的湖面，按照我们的操作，能保证湖面四周的边界方块均高于 i，我们求出区域面积 area，在边界方块中，选一个最低的方格 j——那么使湖面上升到 j 所需要的水量为(j-i)*area，如果当前往方格 i 上倒的总水量为 V，那么若 V < (j-i)*area，就把整个水面区域的水位都提升 V/area 单位，i--，本次操作结束。如果 V>=(j-i)*area，那么把整个水面区域的水位提升到 j，并且把所有当前区域的格子都指向 j，即，当以后有其他格子做 Case 1 操作而导致水落到这个水面上的某个格子上时，就把这些水倒在 j 上，这是等价的，而这会给后面的操作带来方便。然后 i 赋值为 j，并把剩余水量 V-(j-i)*area 倒在方格 j 上。

step 3：输出 ans。

其中 Case 2 即为对第二中情况的模拟，直接讲对数据的处理办法可能比较难以理解，这里不妨来局部模拟一下帮助理解。

（用红色表示当前处理的湖面）

5 12 11 8 15 2 4 14 16 1 3 7 13 10 9 6	5 12 11 8 15 2 4 14 16 1 3 7 13 10 9 6	5 12 11 8 15 2 4 14 16 1 3 7 13 10 9 6	5 12 11 8 15 2 4 14 16 2 3 7 13 10 9 6
初始状态，一开始在 1,2,3,4 上每格倒 3 单位水	经过 4,3,2 的 case1 处理，变为在 1 上倒 12 单位水	1 无法进行 case1 操作，进行 case2:bfs 得湖面即为 1 这格，最低边界为 2，而 12>(2-1)*1，于是 1 的水位升到 2，1 指向 2，2 上要倒 11 单位水，i=2。	2 无法进行 case1 操作，进行 case2:bfs 得湖面为两格，最低边界为 3，1、2 的水位升到 3，1、2 指向 3，3 上倒 9 单位水，i=3。
5 12 11 8 15 3 4 14 16 3 3 7 13 10 9 6	5 12 11 8 15 4 4 14 16 4 4 7 13 10 9 6	5 12 11 8 15 5.5 5.5 14 16 5.5 5.5 7 13 10 9 6	5 12 11 8 15 7 7 14 16 7 7 7 13 10 9 6
同上类似的操作，3 变为 4，4 倒 6 单位水，i=4。	边界为 7，6 <(7-4)*4，4 全变成 5.5 i 继续往下走	这个时候，如果因为其他的原因导致又往 1 上倒了 10 单位的水首先转化为往 4 上倒了 10 单位水，边界为 7，10>(7-5.5)*4，满到 7，多余 4 单位水倒 7 上，i=7。	7 发现有相邻格比它低，于是进行 case1 把 4 单位水倒在边界外。

这有点像在某个碗中倒水，水面面积会越来越大，如果碗有缺口，那么多出来的会漏到碗外。

然而，Case 2 的计算会大大增加计算的复杂度，每次水面的升高将要一次新的 bfs，一种优化办法是对于连续的水面升高，维护当前的水面边界，再加上一些数据结构如堆来优化取最小值的时间，但是即使这样，每次做 Case 2 理论上需要的复杂度还是 O(N*M)，并且它还会使 i 增加，这就连 Case1 的总复杂度 O(N*M)都无法保证了。并且直接优化 Case 2 不但麻烦，还会增加复杂度的常数，不但程序不好打，最终还是满足不了题目中巨大的数据规模。看来我们还是应该再继续分析下去，另辟蹊径来解决 Case 2 的麻烦。

第二步优化——压缩

要优化，就要发现我们作了哪些没用的工作，哪些对最终答案无关的工作，观察我们上个算法对 Case 2 的模拟，会发现水位上升的过程和最终方块 1，2，3，4 的水位情况跟答案并无影响，能不能略去对这些的操作而直接获得答案呢？看上去是不可能的任务，但只要好好想想，其实还是有办法的。比如还是这个例子中，1，2，3，4 就像一个盆地，这个盆地中的水最多升到 7，再上升就会从 7 往外漏了，因此盆地的容量为 18，所以我们不妨把倒在 1，2，3，4 上的水一起算，算出一个落在此盆地的水的总水量 tot，若 tot>18，那么浪费的水量就为 tot-18，可见这里用到一个压缩的思想，我们把一个盆地中的具体水位变化当作冗余的信息压缩了，只保留此盆地溢出的水量。而这对本题 Case 2 的优化的作用，显然是非常巨大的。

为了后面对算法的描述的方便及准确，不妨先对盆地及一些其他概念作更为精确的定义：

方格 **i** 的最终水位 **H[i]**，表示在所有方格上倒无限大的水后，方格 **i** 的水位。

一个海拔为 **i** 的盆地，满足以下性质：

1：它由一些连通的方格组成。

2：所有方格的最终水位等于 **i**，而原来的高度**<i**。

3：方格 **i** 与盆地相邻，除此之外所有与盆地相邻的方格的高度**>i**。

一个海拔盆地为 i 的盆地的容量，即为其方格个数***i**－其方格原高度和，它表示该盆地中最多能倒入多少水而不溢出。

如果方格 **i** 的最终水位为 **i**，那么称方格 i 为海拔为 i 的关卡，因为所有海拔为 i 的盆地均与它相邻，并且它们溢出的水均要通过它往下流。

称一个盆地饱和，如果它的总水量等于它的容量。

称一个盆地过度饱和，如果它的总水量大于它的容量。称这两者的差为此盆地的过度水量。

根据定义，不难发现以下几个与其有关的性质：

每个盆地都有一个且仅有一个相应海拔的关卡与其相邻，一个关卡可能不与任何相应海拔的盆地相邻，也可能与最多 **3** 个相应海拔的盆地相邻。

这个性质可直接从定义中获得。

每个关卡都与至少一个最终水位低于此关卡海拔的方格相邻。这也保证了从任何关卡可以往下倒水。

证明：

若存在关卡 i，它不与任何最终水位低于此关卡海拔的方格相邻，那么考虑关卡 i 及所有相应海拔的盆地所组成的连通块，根据盆地的性质 3 及假设，此连通块相邻的所有方格的高度均>i，那么这个连通块中方格的最终水位不应该是 i，而应该更大，矛盾。假设不成立。

证毕。

任何一个方格，要么在某个盆地中，要么是一个关卡。

证明：

若存在方格既不在盆地中，也不在关卡中。那么由于关卡的定义，这个方格

的最终水位高于原高度，因此与它相邻的方格的最终水位均大于等于它的最终水位，其中必然有方格的最终水位等于它的最终水位，否则它的最终水位应该更高，如果此方格属于盆地，那么与盆地性质第 3 条矛盾，否则继续把它加入连通块，继续找与此连通块连通且最终水位相等的方格，连通块会无限扩大直到找到一个盆地，而地图是有限大的，因此矛盾。假设不成立。

证毕。

第二种算法

好了，有了这些定义和性质，我们就能对 Case 2 进行优化，提出一个新的算法了：

step 1：计算出所有方格的最终水位。根据最终水位判断此方格是否是关卡或所在盆地的编号，并算出所有盆地的容量。对所有关卡及盆地的初始总水量进行统计。

step 2：对所有关卡从大到小进行处理：

对关卡 i 的处理如下：

1：把关卡中的水往下倒，倒的水按相应方格各种情况计入不饱和的相应海拔的盆地的水量、低于其海拔的盆地的水量、低于其海拔的关卡以及答案。并把关卡的水量赋为 0。

2：把所有过度饱和的相应海拔盆地的过度水量反倒在关卡 i 上。

3：如果关卡 i 的水量为 0，跳出，否则跳入 1。

step 3：输出答案。

其中 step 1 中算最终水位和得到盆地过于笼统，事实上这也不是一个非常简单的问题，在实际的操作中，是直接得到盆地的，最终水位其实只是一个为了便于叙述而创造的概念。下面来看得到盆地的算法：

从高度从小到大的顺序判断相应高度的格子是否是关卡并求出盆地，在操作时把已经确定的点标记为 visited。

假如目前在操作高度 i 的点：

1：首先看 i 是否在边界上或与某个已访问的点相连，如果是，那么 i 是关卡，标记 i，否则直接跳出操作 i+1 的。

2：对每个 i 相邻的格子，如果未访问，并且高度<i，那么就以它为七点 bfs，bfs 时也是相同的限制条件，bfs 出的点即为一个盆地，把它们全部标记了并顺便计算它的容量。

下面来证明这个算法的正确性：

证明：

先证明对高度 i 的操作把最终水位为 i 的点全标记了出来，也就是说每次操作 i 前，剩余格的最终水位均≥i，操作后均>i。用归纳法来对此进行证明。

一开始，剩余格即为所有格，最终水位均≥1，满足条件。

假如对于高度 i 的操作，满足操作前剩余格最终水位均≥i，那么：

如果 i 不在边界上也不与某个已访问的点相连，那么 i 所在的未访问连通块的边界上的方格高度必然都>i，那么这个连同块的最小水位即为最低边界格>i，因此 i 的最终水位不为 i，i 不是关卡，显然也就不存在海拔为 i 的盆地。此

时不标记任何点，但是满足剩余格最终水位均>i。

否则，i 的最终水位必然为 i，因为它不能贮存水，它与一个最种水位比它的高度低的方格相连，倒在它上面的水一定为往下流。而对盆地进行 bfs 也是按照盆地的定义，bfs 后，最终水位为 i 的点全被标记了，满足剩余格的最终水位均>i。

证毕。

现在我们已经得到一个具体而又正确的算法，来分析一下复杂度。step 1，求盆地和关卡中每个点最多 bfs 到一次，而每个点最多往外延伸 4 个点，因此 bfs 的总复杂度为 O(N*M)，除 bfs 外的操作均是 O(1)的，最多有 O(N*M)个操作，因此这步的总复杂度为 O(N*M)。step 2，对某个关卡的处理看似会出现死循环，但事实上可以证明这个循环的次数是有限并且很小的，每次循环，要么跳出，要么多一个饱和的盆地——不跳出意味着有至少一个盆地从过度饱和变为饱和，因此循环次数最多为最大盆地个数＋1 次，而一个关卡最多连接三个盆地，因此循环次数最多为 4 次，每次循环都是 O(1)的，因此对某个高度的操作是 O(1)的，而这个算法能保证操作的高度递降而不回升，因此操作总数最多 O(N*M)次，复杂度为 O(N*M)。step 3 也是 O(1)的，因此总复杂度为 O(N*M) + O(N*M) + O(1) = O(N*M)。空间复杂度也为 O(N*M)，这表明此算法能够胜任题目中给出的巨大数据。[7]

小结

这题数据规模巨大，一开始提出的模拟的想法总让人感觉是不可能有希望的，然而，在这样的情况下，我们大胆地提出猜想对其进行第一次优化，又仔细分析后提出压缩的思想而对其进行了第两次的优化，每次优化都使算法变优很多，使时间复杂度从一开始的不可能到最终的很优。而最终算法的实现也变得非常明了简洁。这又一次证明了对于这样的一类问题，只要肯抓住其特点一步一步好好地分析，最终一定能得到胜利。

总结

回顾文中这类具有物理特色的问题，我们能非常清楚地看到它们具有的一种鲜明的特征：它们都是实际的，这让我们一开始就对它们有一个程度的感性认识；它们中隐藏了许多优美的规律，要发掘这些规律，就需要我们利用现有的感性认识，经过一步一步理性的分析，找出一个个突破点，而最终达到我们的目的。可以看到最终它们的算法都是条理清晰，而实现它们也并不困难。

对这类问题的分析还需要有严谨的精神，正像对物理的分析需要严谨的精神一样——一开始也许会觉得严谨只是使分析复杂化，但只有通过严谨的分析，才能够发现问题深处透出的优美，从而更好地理解问题的意义。

这类问题是非常迎合当今的问题发展趋势的，相信以后会有更多更好的有物理特色的问题会出现，而当我们面对这类问题时，只要发挥丰富的想象力，

[7] 程序请参加附录中 3D_Music.cpp

保持冷静的头脑，不中途而废，相信问题最终一定会迎刃而解，而我们从中也一定能获得一些意想不到的收获。

附录

附录一：关于[例三]中的证明

这里仅证明交换两个操作对最终结果不会产生影响，而事实上，也可以证明一个更强的结论——交换两个操作对所有方格的水位都不会影响，有兴趣的读者可以试着证明。

证明：

这儿引入第二步操作中盆地和关卡的概念来对其进行证明。采用归纳法：

1：两个操作所在盆地或关卡的高度均＝0（可以认为它们倒在边界外）时成立。

2：若对于两个操作所在盆地或关卡高度均≤k(k≥0)时成立，那么，要证明对≤k+1 时也均成立，只要证明下面几种情况。

Case 1：其中一个操作所在盆地或关卡高度≤k，另一个＝k+1，前一个操作无论先做后做，均不会影响后一个操作对盆地水位变化的影响，也不会对方格 k+1 往下倒的水量有影响，而方格 k+1 往下倒的水会落在几个最终水位高度≤k 的方格上，而根据假设，这几个操作是可以与前一个操作互换的，因此可以说这两个操作也是可以互换的。

Case 2：两个操作均在 k+1 的关卡上或均在 k+1 的盆地上，那么这两个操作可合并，显然互换也是可以的。

Case 3：一个操作在 k+1 的盆地上，一个操作在 k+1 的关卡上。如果两个操作不能使该盆地饱和，那么交换是可以的，无论先后关卡的水都往此盆地流一部分，而此盆地的水流不出去。如果能使该盆地饱和，两个操作共同的作用无论先后均等价于用总水量让盆地饱和后往 k+1 的关卡上倒剩余的水，也是等价的。

证毕。

附录二：附件

ars longa 英文原题：ars longa.pdf
ars longa 的程序：ars.cpp
water tanks 的英文原题：water tanks.pdf
water tanks 的第一个程序：tanks_1.cpp
water tanks 的第二个程序：tanks_2.cpp
3D Musical Water-fence 的英文原题：3D_Music.doc
3D Musical Water-fence 的程序：3D_Music.cpp

附录三：供测试的网址

http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=3563
http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=3810
http://acm.pku.edu.cn/JudgeOnline/problem?id=3242

感谢

感谢让我走上 **OI** 之路，为我启蒙，带我比赛的沈晓恬老师。

感谢为我写此论文提供莫大帮助，提供资料，纠正错误的刘汝佳教练。

感谢伴随我一路走来的，给予我鼓励的 **4** 个朋友。