算法设计基础

陈锋

sukhoeing@gmail.com, QQ(23543620)

20180701

最优装载问题

给出n个物体，第i个物体重量为wi。选择尽量多的物体，使得总重量不超过C。

【分析】

1. 只关心物体的数量，装重的没有轻的划算。
2. 所有物体按重量从小到大排序，依次选择每个物体，直到装不下为止。
3. 典型的贪心算法，它只顾眼前，但却能得到最优解。

部分背包问题

有n个物体，第i个物体的重量为wi，价值为vi。在总重量不超过C的情况下让总价值尽量高。每一个物体都可以只取走一部分，价值和重量按比例计算。

【分析】

1. 本题在上一题的基础上增加了价值，所以不能简单先拿轻的(轻的可能价值也小)，也不能先拿价值大的(可能它特别重)，综合考虑两个因素。
2. 直观的贪心策略是：优先拿“价值/重量”最大的，直到重量和正好为C。
3. 物体可以只拿一部分，一定可以让总重量恰好为C(或全部重量不足C)，除了最后一个以外，要么不拿，要么拿走全部。

乘船问题

有n个人，第i个人重量为wi。每艘船的最大载重量均为C，且最多只能乘两个人。用最少的船装载所有人。

【分析】

考虑最轻的人i，i应该和谁一起坐？如果每个人和i坐不下，唯一方案就是每人坐一艘船(WHY)。否则i应该选择能和他一起坐的人中最重的j。方法是贪心的，只是让“眼前”的浪费最少。反证法说明：

1. 情况1：i不和任何人同坐，把j拉过来和他一起坐，总船数不增加(可能会减少)。
2. 情况2：i和k同船。由贪心策略，j是“可以和i同坐”中最重的。因此k比j轻。把j和k交换后k所在的船仍然不会超重(k比j轻)，而i,j所在的船也不会超重，所得新解不会更差。
3. 由此可见，贪心不会丢失最优解。
4. 程序实现上，比j更重的只能每人坐一艘。只需用i,j分别表示当前考虑的最轻的人和最重的人，每次先将j往左移，直到i和j可以共坐。然后将i加1，j减1，并重复上述操作。
5. 时间复杂度仅为O(n)，是最优算法(输入也需要O(n))。

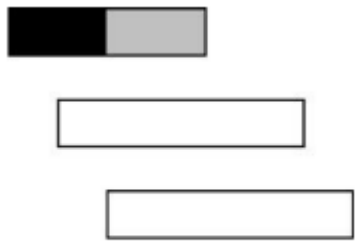
选择不相交区间

数轴上有n个开区间(ai,bi)。选择尽量多个区间，使得这些区间两两没有公共点。

【分析】

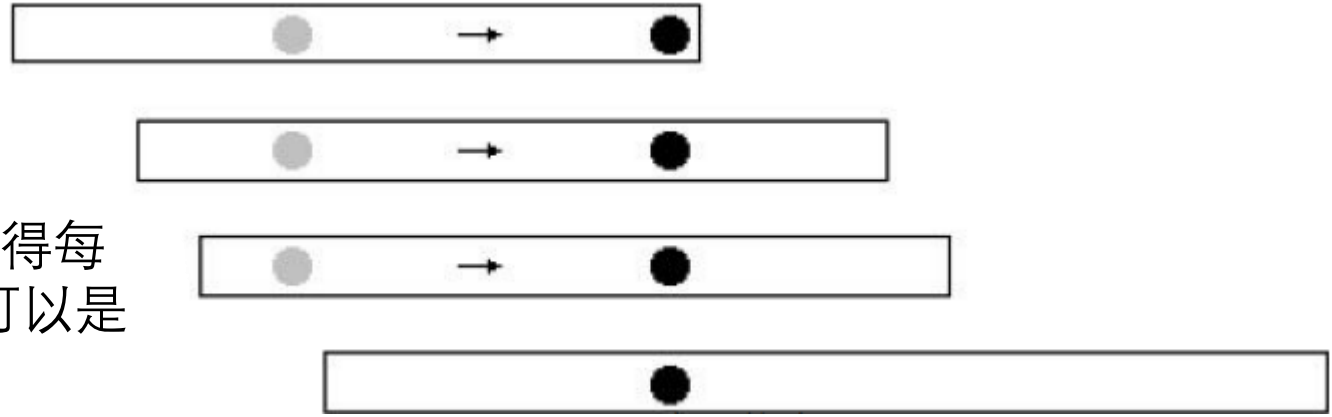
1. 假设有两个区间x,y，x完全包含y。选x是不划算的，因为x和y最多只能选一个，选x还不如选y，不仅数目不会减少，且给其他区间留出更多位置。接下来，按照bi从小到大给区间排序。贪心策略是：一定要选第一个区间。为什么？现在已经是b1≤b2≤b3…了，考虑a1和a2的大小关系。
2. 情况1：a1>a2，如图(a)所示，区间2包含1。一定不会选择区间2。不仅区间2如此，只要有a1>ai，i都不要选。
3. 情况2：排除了情况1，一定有a1≤a2≤a3≤…，如图(b)所示。如果区间1,2完全不相交，那么没有影响(一定要选1)，否则1和2最多只能选一个。如果不选2，黑色部分其实不影响的（它不会挡住任何一个区间），区间1的有效部分变成灰色部分，被2所包含！区间2不能选。依此类推，不能因为选任何区间而放弃区间1，因此选择区间1是明智的。
4. 选择区间1后，把所有和1相交的区间排除在外，需要记录上一个被选择的区间编号。排序后只需扫描一次即可完成贪心过程，得到正确结果。

（a） $a_1>a_2$



（b） $a_1<a_2<a_3$

区间选点问题



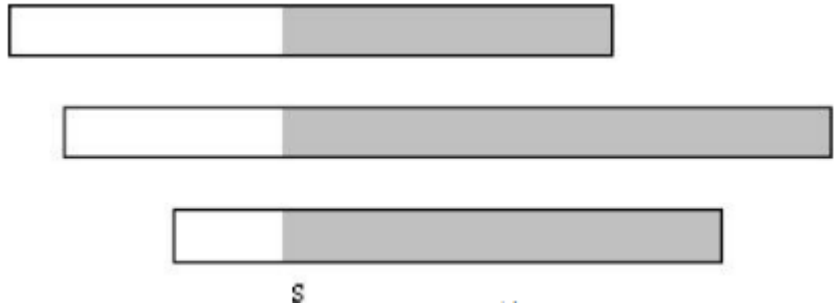
数轴上有n个闭区间[ai,bi]。取尽量少的点，使得每个区间内都至少有一个点(不同区间内含的点可以是同一个）。

【分析】

1. 如果区间i内已经有一个点被取到，则称区间i已被满足。先讨论区间包含。小区间被满足时大区间一定也被满足，这种情况下大区间不需考虑。
2. 把所有区间按b从小到大排序(b相同时a从小到大排)，则如果出现区间包含，小区间一定排在前面。第一个区间应该取哪一个点呢？此处的贪心策略是：取最后一个点，如图所示。
3. 根据刚才的讨论，所有需要考虑的区间的a也是递增的，如果区间1取中间的灰色点，把它移动到最后，被满足的区间增加，原先被满足的区间现在一定被满足。不难看出，这样的贪心策略是正确的。

区间覆盖问题

有n个闭区间[ai,bi]，选择尽量少的区间覆盖一条指定线段[s,t]



【分析】

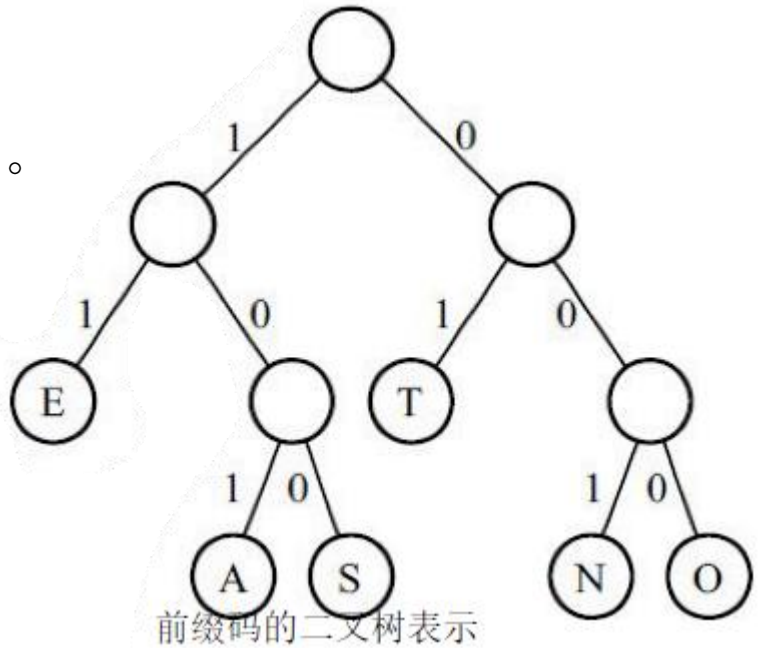
1. 突破口仍然是区间包含和排序扫描，
2. 先进行预处理。每个区间在[s,t]外的部分都应该预先被切掉。相互包含时，小区间无需考虑。
3. 按照a递增排序。如果1起点不是s，无解(其他起点更大，不可能覆盖s)，否则选择起点在s的最长区间 [ai,bi]，之后新起点设置为bi，并且忽略所有区间在bi之前的部分，就像预处理一样。
4. 一次扫描(如图)。s为当前起点(之前被覆盖)，应该选择区间2。

最优编码问题

给出n个字符的频率ci，给每个字符赋予一个01编码串，使得任意一个字符的编码不是另一个字符编码的前缀，而且编码后总长度(每个字符的频率与编码长度乘积的总和)尽量小。

【分析】

1. 任何一个前缀编码都可以表示成每个非叶结点恰好有两个子结点的二叉树。每个非叶结点与左子结点的边上写1，与右子结点的边上写0。每个叶子对应一个字符，编码为从根到该叶子的路径上的01序列。在图中，N的编码为001，而E的编码为11。证明两个结论：
2. 结论1：n个叶子的二叉树一定对应一个前缀码。如果编码a为编码b的前缀，则a所对应的结点一定为b所对应结点的祖先。而两个叶子不会有祖先后代的关系。
3. 结论2：最优前缀码一定可以写成二叉树。逐个字符构造即可。每拿到一个编码，都可构造出从根到叶子的一条路径，沿着已有结点走，创建不存在的结点。这样得到的二叉树不可能有单子结点，因为如果存在，只要用这个子结点代替父结点，得到的仍然是前缀码，且总长度更短。



前缀码的二叉树表示

唯一的雪花(Unique snow flakes, UVa11572)

输入一个长度为n($n \leqslant 10^6$)的序列A，找到一个尽量长的连续子序列A_L~A_R，使得该序列中没有相同的元素。

【分析】

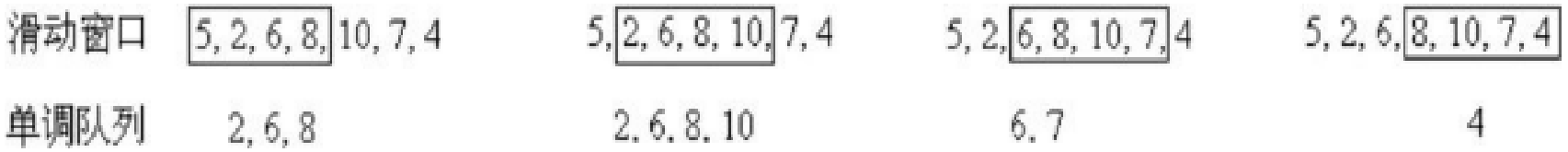
1. 假设序列元素从0开始编号，所求连续子序列为[L,R]。
2. 首先考虑L=0。从R=0开始不断增加R，相当于把所求序列的右端点往右延伸。当无法延伸(A[R+1] in A[L~R])时，增大L，继续延伸R。既然当前的A[L ~ R]是可行解，L增大之后必然还是可行解，所以不必减少R，继续增大即可。
3. 不难发现这个算法是正确的，时间复杂度? 暂时先不考虑“判断是否可以延伸”这个部分，每次要么把R加1，要么把L加1，而L和R最多从0增加到n-1，所以指针增加的次数是O(n)。
4. 考虑“判断是否可以延伸”这个部分。容易想到用STL的set保存A[L ~ R]中元素集合，R增大时判断A[R+1]是否在set中出现，R++时把A[R+1]插入到set，L+1时把A[L]删除。set的插入删除和查找都是O(logn)，总时间复杂度为O(nlogn)。
5. 也可用map维护last[i]，就是i的“上一个相同元素的下标”。例如，输入为3241323，当前区间是[1,3](即元素2,4,1)，是否可以延伸呢？下一个数是A[5]=3，它的“上一个相同位置”是下标0（A[0]=3），不在区间中，因此可以延伸。
6. map所有操作都是O(logn)，总时间复杂度也是O(nlogn)。

滑动窗口的最小值问题

输入正整数k和一个长度为n的整数序列$A_1,A_2,A_3,\cdots,A_n$。定义f(i)表示从元素i开始的连续k个元素的最小值，即$f(i)=\min\{A_i,A_{i+1},\cdots,A_{i+k-1}\}$。要求计算f(1),f(2),f(3),…,f(n-k+1)。例如，对于序列5,2,6,8,10,7,4，k=4，则f(1)=2,f(2)=2,f(3)=6,f(4)=4。

【分析】

1. 使用定义，每个f(i)都需要O(k)时间计算，总时间复杂度为O((n-k)k)，太大了。
2. 计算f(1)时，需要求k个元素最小值——这是一个“窗口”。计算f(2)时，这个窗口向右滑动了一个位置，计算f(3)和f(4)时，窗口各滑动一个位置，窗口中的元素“出去”了一个，又“进来”了一个。往右滑动时需要删除一个，然后插入，还需要取最小值。这不就是优先队列吗？
3. 还可以做得更好。假设窗口中有两个元素1和2，且1在2的右边，2在离开窗口之前永远不可能成为最小值。这个2是无用的，应当及时删除。删除无用元素后，滑动窗口中的有用元素从左到右递增。称为单调队列队首元素就是最小值。
4. 窗口滑动时，首先删除滑动前窗口的最左边元素(如果是有用元素)，然后把新元素加入单调队列。注意，比新元素大的元素都变得无用了，应当从右往左删除。
5. 单调队列和普通队列不同，右端既可以插入删除，在代码中通常用一个数组和front、rear两个指针来实现，或者用STL中的deque。插入时可能会删除多个元素，但每个元素最多被删除一次，所以仍为O(n)，达到了理论下界（至少需要O(n)来检查每个元素)。



例题1 勇者斗恶龙(The Dragon of Loowater, UVa11292)

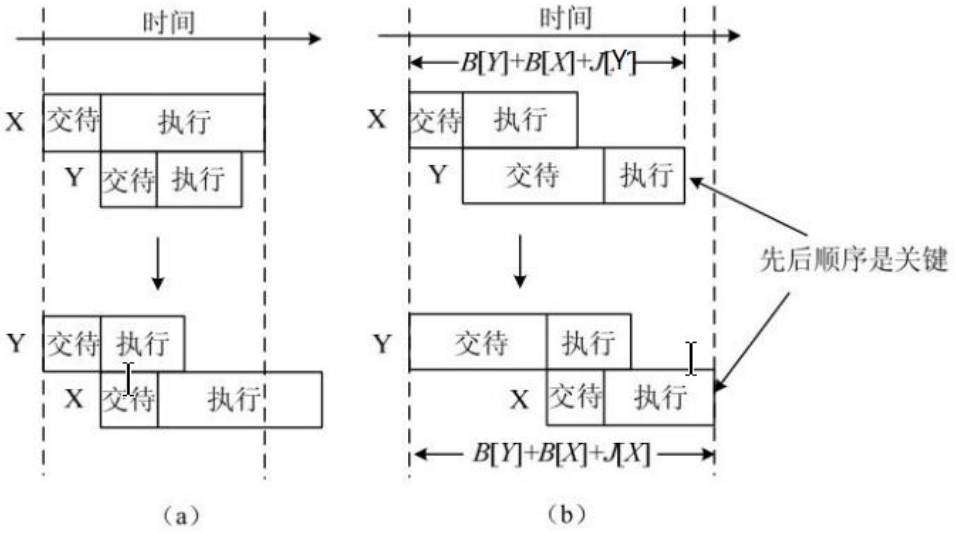
王国里有一条 n 个头的恶龙，希望雇一些骑士把它杀死（砍掉所有头）。有 m 个骑士可以雇佣。一个能力值为x的骑士可以砍掉一个直径不超过x的头，需要支付x个金币。如何雇佣骑士，才能砍掉恶龙的所有头，并且需要支付的金币尽量少？注意，一个骑士只能砍一个头（且不能被雇佣两次）。

【分析】

能力强的骑士开价高是合理的，但如果被你派去砍一个很弱的头，就是浪费人才了。 换句话说，如果你把雇佣来的骑士按照能力从小到大排序，所有头按照直径递增排序， 一个一个砍就可以了——当然，不能砍掉“当前需要砍的头”的骑士就不要雇佣了。

例题2 突击战(Commando War, UVa11729)

有n个部下，每个部下需要完成一项任务。第i个部下需要你花B_i分钟交待任务， 然后他会立刻独立无间断执行J_i分钟后完成任务。需要选择交待任务的顺序，使得所有任务尽早执行完毕（即最后一个执行完的任务尽早结束）。注意，不能同时给两个部下交待任务，但部下们可以同时执行各自任务。



【分析】

直觉上执行久的任务应该先交待。贪心算法： 按照J递减对任务排序，依次交待。

为什么是对的呢？假设我们交换两个相邻的任务X和 Y（以前X在Y之前， 交换后Y在X之前），不难发现其他任务的完成时间没有影响，那么这两个任务呢？分情况讨论

1. 交换前Y先结束，如图(a)。交换后 X 结束时间延后，Y结束时间提前，最终答案不会变好。
2. 交换前X先结束。因此交换后答案变好的充要条件是：交换后X结束比交换前 Y 结束早（交换后Y结束时间肯定变早了），如图(b)。这个条件可以写成 $B_Y+B_X+J_X<B_X+B_Y+J_Y \rightarrow J_X<J_Y$，这就是贪心依据，J更大的放前面。

例题3 分金币(Spreading the Wealth, UVa11300)

圆桌上坐着n(n≤1000000)个人，每人有一定数量的金币，金币总数能被n整除。每个人可以给他左右相邻的人一些金币，最终使得每个人的金币数目相等。你的任务是求出被转手的金币数量的最小值。比如，n=4，四个人的金币数量分别为1,2,5,4，则只需转移4个金币：第三给第二2个金币，第二个人和第四个人分别给第一个人1个金币。

【分析】

首先，最终每个人的金币数量=金币总数除以n。记为M。

假设有四个人，按顺序编号为 1, 2, 3, 4。假设1号给2号3 枚金币，然后2号又给1号5 枚金币，实际上等价于2号给1号2枚金币，而1号什么也没给 2 号。用x_2 表示2号给1 号的金币数量。x_2<0，表示1号给2号-x_2个金币。x_1 , x_3和x_4的含义类似。由于是环形，x_1指的是1号给4号多少金币。

现在假设编号为 i 的人初始有A_i个金币。1号来说，他给4号x_1枚金币，还剩A_i-x_1枚；但因为2号给了他x_2枚金币，所以最后还剩A_1-x_1+x_2枚金币。根据题设，它等于M。A_1-x_1 +x_2=M。 同理，对于第二个人，A_2-x_2+x_3 =M。可以得到 n 个方程。一共有n个变量，但不能直接解方程组了呢？因为从前 n-1 个方程可以推导出最后一个方程（why?）。只有n-1个方程有用。

用 x_1表示出其他的x_i，则本题就变成了 单变量的极值问题。具体来说：

$A_1-x_1+x_2=M\rightarrow x_2=M-A_1+x_1=x_1-C_1$(规定$C_1=A_1-M$，以后类似)

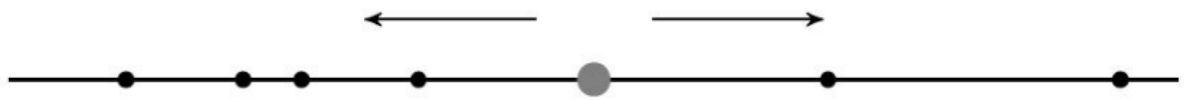
$A_2-x_2+x_3=M\rightarrow x_3=M-A_2+x_2=2M-A_1-A_2+x_1=x_1-C_2$

$A_3-x_3+x_4=M\rightarrow x_4=M-A_3+x_3=3M-A_1-A_2-A_3+x_1=x_1-C_3$

…

$A_n-x_n+x_1=M$。这是一个多余的等式，并不能给我们更多的信息(想一想，为什么）。

希望所有x_i的绝对值之和尽量小，即$|x_1|+|x_1-C_1|+|x_1-C_2|+\cdots+|x_1-C_{n-1}|$要最小。$|x_1-C_i|$的几何意义是数轴上点$x_1$到$C_i$的距离。问题变成给n个点，找一个点，到它们的距离之和尽量小。最优的x_1就是这些数的“中位数”（排序以后位于中间的数），因此只需排序就可以了。要证明是：给定数轴上的n个点，中位数离所有顶点的距离之和最小。凡是能转化为这个模型的题目都可以用中位数求解。



随便找一个点，比如上面的灰点。它左边有四个输入点，右边有两个输入点。把它“稍 微往左移动一点”——不要移得太多，以免碰到输入点。假设移动了d单位距离，则左边四点到它的距离各减少d，右边的两个点到它的距离各增加d，总的来说，距离之和减少了2d。

如果灰点的左边有两个点，右边有四个点，道理类似，只不过应该往右移动。换句话说，只要灰点左右的输入点不一样多，就不是最优解。什么情况下左右的输入点一样多呢？ 如果输入点一共有奇数个，则灰点必须和中间的那个点重合（中位数！）；如果有偶数个，则灰点可以位于最中间的两个点之间的任意位置（还是中位数！）。证毕。

例题5 蚂蚁(Piotr’s Ants, UVa10881)

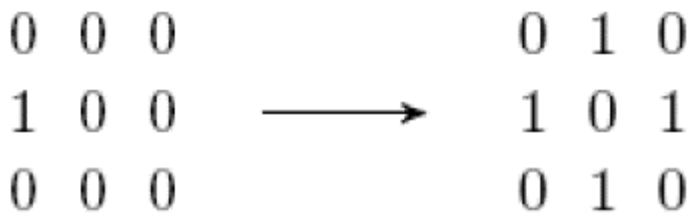
一根长度为L厘米的木棍上有n只蚂蚁，每只蚂蚁要么朝左爬，要么朝右爬，速度为1厘米/秒。当两只蚂蚁相撞时，二者同时掉头（掉头时间忽略不计）。给出每只蚂蚁的初始位置和朝向，计算T秒之后每只蚂蚁的位置。

【分析】

1. 假设你在远处观察这些蚂蚁的运动，会看到什么？一群密密麻麻的小黑点在移动。由于黑点太小，当蚂蚁因碰撞而掉头时，看上去和两个点“对穿而过”没有任何区别——换句话说，如果把蚂蚁看成是没有区别的小点，那么只需要独立计算每只蚂蚁在 T 时刻的位置即可。比如有三只蚂蚁。蚂蚁 1=(1, R)，蚂蚁 2= (3, L)，蚂蚁 3=(4, L)，则两秒钟之后，三只蚂蚁分别为(3,R), (1,L), (2,L)。
2. 从整体上讲，“掉头”等价于“对穿而过”，对于每只蚂蚁而讲并不是。
    1. 蚂蚁 1的初始状态为(1,R)，一定有一只蚂蚁在2秒之后处于(3,R)，但这只蚂蚁却不一定是蚂蚁 1！，需要搞清楚目标状态中“谁是谁”。
3. 所有蚂蚁的相对顺序保持不变的，因此把所有目标位置从小到大排序，则从左到右的每个位置对应于初始状态下从左到右的每只蚂蚁。
    1. 由于原题中蚂蚁不一定按照从左到右的顺序输入，还需要预处理计算出输入中的第 i 只蚂蚁的序号 order[i]。

例题7. 偶数矩阵(Even Parity, UVa11464)

给一个n*n(1≤N≤15)的01矩阵(元素非0即1)，把尽量少的0变成1，使每个元素的上下左右的元素(如果存在)之和均为偶数。例如：下图左边的矩阵至少要把三个0变成1。

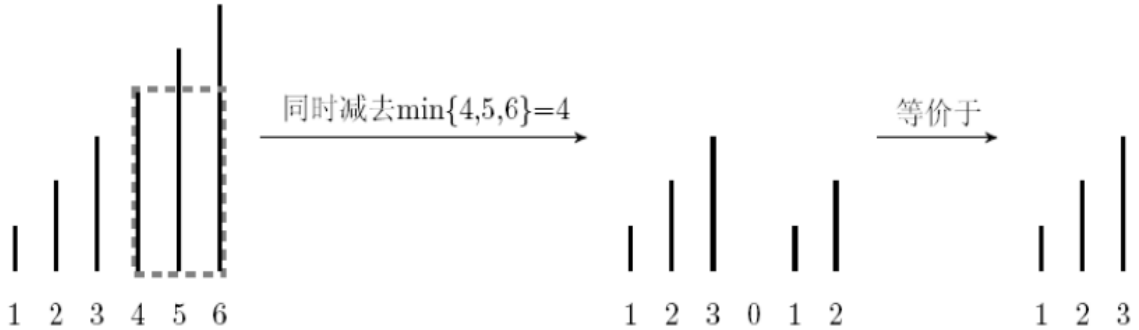


【分析】

1. 暴力方法是枚举每个数字“变”还是“不变”，判断整个矩阵是否满足条件。最多需要枚举2^{255} ≈$5*10^{67}$种情况，肯定TLE。
2. N≤15，第一行有不超过 2^{15}=32768 种可能，第一行暴力枚举。根据第一行可以完全计算出第二行(要让第1行满足)，第二行又能计算出第三行(要让第2行条件满足)……总时间复杂度是$O(2^n*n^2)$。

例题10 正整数序列(Help is needed for Dexter, UVa11384)

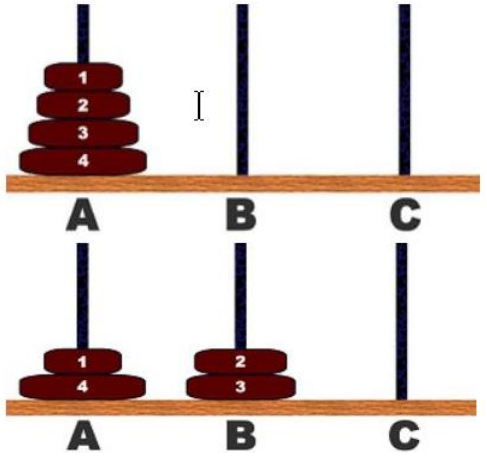
给定正整数 n，你的目标是用最小操作次数把序列 1, 2, …, n 中的所有数都变成 0。每次操作是从序列中选择一个或多个整数，同时减去一个相同的正整数。比如 1,2,3 可以把 2和 3 同时减小 2，得到 1,0,1。



【分析】

1. 首先自己试一试。经过若干次尝试和总结后，不难发现第一步的最好这样。
    1. 当 n=6 的时候留下 1, 2, 3，而把 4,5,6 同时减去 min{4,5,6}=4，得到序列 1, 2, 3, 0, 1, 2，它等价于 1, 2, 3（想一想，为什么）。换句话说，f(6)=f(3)+1。
2. 一般的，为了平衡，我们保留 1 到 n/2，把剩下的数同时减去 n/2+1，得到序列 1, 2, …,n/2, 0,1,…,(n-1)/2，它等价于 1, 2, …, n/2，因此 f(n)=f(n/2)+1。边界是 f(1)=1。

例题11 新汉诺塔问题(A Different Task, UVa10795)



标准的汉诺塔上有 n(1≤n≤60)个大小各异的盘子。给定一个初始局面，求它到给定目标局面至少需要多少步。移动规则如下：一次只能移动一个盘子；在移动一个盘子之前，必须把压在上面的其他盘子先移走；编号大的盘子不得压在编号小的盘子上。

【分析】

1. 考虑最大的盘子。如果这个盘子在初始局面和目标局面中位于同一根柱子上，根本不需要移动。移动了，反而不可能是最优解（WHY?）。这样，我们可以找出在初始局面和目标局面中所在柱子不同的盘子中，编号最大的一个，设为 k，k 必须移动。
2. 考虑移动k之前的一瞬间，柱子上的情况。假设k需要从柱子1移动到2。编号比k小的盘子既不能在1上，也不能在2上，因此只能在3上。换句话说，这时柱子1只有盘子k，柱子 2 为空，柱子 3 从上到下依次是盘子 1, 2, 3, …, k-1（再次提醒：已经忽略编号大于k 的）。
3. 把这个局面称为参考局面。 由于盘子移动可逆，根据对称性，只需要求出初始和目标移动成参考局面的步数之和，然后加一移动盘子k的一步即可。

1. 需要写一个函数 f(P,i, final)，表示已知各盘子的初始柱子编号数组为P（具体来说，P[i]代表盘子i的柱子编号），把盘子 1,2,… i 全部移到柱子final所需的步数，则答案就是f(start, k-1, 6-start[k]-finish[k]) + f(finish, k-1, 6-start[k]-finish[k]) + 1，其中 start[i]和 finish[i]是i 的初始和目标柱子， k “必须移动的最大盘子”。
    1. 柱子编号为1,2,3，所以“除了柱子x,y之外的那个柱子”编号为6-x-y。
2. 如何计算 f推理和刚才类似。假设 P[i]=final，那么 f(P,i,final)=f(P,i-1,final)；否则需要先把前i-1个盘子挪到 6-P[i]-final做中转，然后把i移动到final，最后把前i-1个盘子从中转的柱子移到final。注意，最后一个步骤是把 i-1 个盘子从一个柱子整体移到另一个柱子，根据汉诺塔问题的经典结论，这个步骤需要 2^{i-1}-1步，加上移动盘子i的那一步，一共 2^{i}-1 步。 p[i]≠final，f(P,i,final)=f(P,i-1,6-P[i]-final)+2^{i}-1。
3. 注意答案要用long long。

例题12 组装电脑(Assemble, NWERC 2007, LA3971)

你有b块钱，要组装一台电脑。给出n个配件各自的种类、品质因子和价格，要求每种类型的配件各买一个，总价格不超过 b，且“品质最差配件”的品质因子应尽量大。输出这个品质因子。

【分析】

1. 解决“最小值最大”的常用方法是二分。
2. 假设答案为 x，如何判断 x 是太小还是太大？删除品质因子小于 x 的所有配件，如果可以组装出一台不超过 b 元的电脑，那么答案 ans>=x，否则 ans<x。
3. 如何判断是否可以组装出满足预算约束的电脑：每一类配件选择最便宜的一个。如果这样都超预算，问题无解。

例题13 派(Pie, NWERC 2006, LA3635)

有 F+1 个人来分N个圆形派，每个人得到的必须是一整块派而不是好几块拼在一起，且面积要相同。问每个人最多能得到多大面积的派(不必是圆形)。

【分析】

1. 不是“最小值最大”问题，仍然可以二分，把问题转化为“是否可以让每人得到一块面积为 x 的派”。这样的转化相当于多了一个条件，求解目标变成“看看这些条件是否相互矛盾”。
2. x 太大就满足不了 F+1 个人。只需要计算共可以切多少份面积为 x 的派，看份数是否≥F+1 即可。派是不可以拼，半径为 r 的派只能切出$[\pi r^2]$个派(其他部分浪费)
3. 所有派能切出的份数加起来即可。

例题14 填充正方形(Fill the Square, UVa11520)

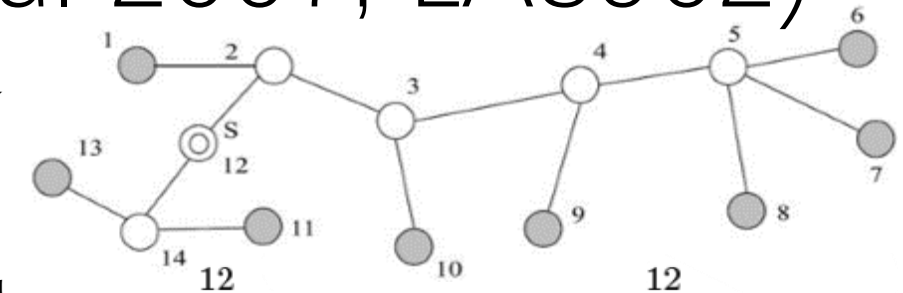
在一个 n*n 网格中填了一些大写字母。你的任务是把剩下的格子中也填满大写字母，使得任意两个相邻格子（即有公共边的格子）中的字母不同。如果有多种填法，按照从上到下从左到右的顺序把所有格子连接起来得到的字符串的字典序应该尽量小。

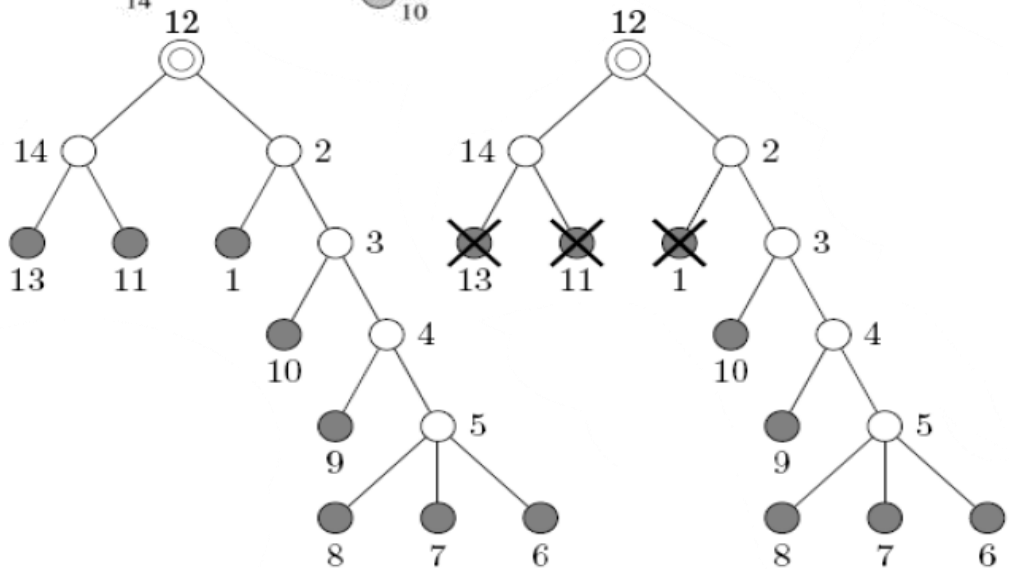
【分析】

1. 当解可能有多个时，为了确保答案惟一，题目通常会加上一些限制条件，“字典序最小”就是一个常见要求。可以用STL的lexicographical_compare。
2. 只有序列才有字典序，题目中的这句“从上到下从左到右”就不难理解了。它的意思是首先把每行看成一个字符串，然后从上到下顺次连接。要求得到的这个长长的字符串的字典序最小。
3. 根据字典序的定义，只需从左到右从上到下依次给所有的空格填上最小可能的字母即可。
4. 如果某个格子把 A 到 Z 的所有字母都尝试完，全部填不了，该怎么办？这意味着必须推翻以前的策，难道要回溯？
5. 每个格子的上下左右一共才 4 个格子，不可能包含 A 到 Z 这 26 个字母。因此，每个空格都能填上字母。

例题15 网络(Network, Seoul 2007, LA3902)

n台机器连成一个树状网络，其中叶结点是客户端，其他是服务器。目前有一台服务器正在提供VOD服务，虽然视频质量本身很不错，但对于那些离它很远的客户端来说，网络延迟难以忍受。 在一些其他服务器上也安装同样的服务，使得每台客户端到最近服务器的距离不超过整数 k。安装服务的服务器台数应尽量小。如下图，当 k=2 时还要在结点 4 放一台服务器。





【分析】

1. 把无根树变成有根树，已经有了一个的root：原始 VOD 服务器。对于那些已经满足条件(到root距离≤k)的点，我们直接当它们不存在
2. 考虑最深的叶子(比如8)。应该在哪里放新的服务器来覆盖它呢(到8的距离≤ k)？只有5,4满足条件。显然4比5划算，因为5 能覆盖的叶子(6, 7, 8)都能被 4 覆盖！

1. 一般对于深度最大的 u，选择u的k 级祖先是最划算的(父亲是 1 级祖先，父亲的父亲是 2 级……)。
2. 每放一个新服务器s，进行一次 DFS，覆盖与它距离不超过 k 的所有结点(这里注意dfs要以s为根)。
3. 注意覆盖叶子而不需要覆盖中间结点，深度不超过 k 的叶子已经被root覆盖，只需要处理深度大于 k 的叶结点。
4. 为了让程序更简单，我们用 nodes 表(深度d→所有深度为d的点)避开了“按深度排序”的操作。

长城守卫(Beijing Guards,CERC2004,LA3177)

有 n 个人围成一个圈，第i个人想要r_i个不同的礼物。相邻的两个人可以聊天，炫耀自己的礼物，所以如果某两个相邻的人拥有一种相同的礼物，双方都会很不高兴。问：一共需要多少种东西才能满足所有人的需要？假设每种东西有无穷多个，不相邻的两个人不会一起聊天，所以即使拿到相同的礼物也没关系。

比如，一共有 5 个人，每个人都要一个礼物，则至少要 3 种礼物。如果把这三种礼物编号为 1, 2, 3，则 5 个人拿到的礼物分别是：1,2,1,2,3；如果每个人要两个礼物，则至少要5 种礼物。5个人拿到的礼物集合分别是：{1,2},{3,4},{1,5},{2,3},{4,5}。

【分析】

1. n为偶数时，答案为相邻两人r值之和的最大值，即$p=\max\{r_i+r_{i+1}\}$, 规定 $r_{n+1}=r_1$ 。不难看出p是答案的下限，还可以构造出只用p种礼物的方案：对于i, i为奇数发礼物1到r_i；i为偶数则发礼物$p-r_i+1$到p。
2. n为奇数时上述解法无效。需要二分答案：假设已知一共有 p 种东西，如何分配礼物呢？设第1个人的礼物是1~r_1 ，最优分配策略是：编号为奇数的人尽量往前取，偶数尽量往后取。这样，编号为n的人在不冲突的前提下，尽可能地往后取了r_n样东西，最后判定1和n的礼物是否冲突即可。
   1. 例如n=5，R={2,2,5,2,5}，p=8，第1取{1,2}，第2取{3,4}, 第3取{8,7,6,5,2}，第4人取{1, 3}，第5人取{8, 7, 6, 5, 4}，由于第 1 个人与第 5个人不冲突，所以 p = 8 是可行的。
   2. 1: 1 2;- 2: 8,7; - 3: 1,2,3,4,5, 4 : 8,7, 5, 1,2,3,4,5
3. 不要求输出方案，只需记录每个人在[1~r_1]的范围内取了几个，[r_1+1~n]里取了几个（在程序中分别用 left[i]和 right[i]表示），最后判断第 n 个人在[1~r_1]里面是否有取东西即可。

例题17. 年龄排序 (Age Sort, UVa 11462)

给定n个($0<n\leqslant 2*10^6$)居民的年龄（都是1到100之间的整数），把它们按照从小到大的顺序输出。输入可能有25MB。

【分析】

由于数据太大，内存限制太紧，甚至都不能把它们全读进内存，更别说用快速排序了。好在整数范围很小，可以用计数排序。

例题18 开放式学分制(Open Credit System, UVa11078)

给一个长度为 n 的整数序列 $A_0,A_1,\cdots A_{n-1}$，找出两个整数 A_i和$A_j(i<j)$，使得A_i-A_j尽量大。

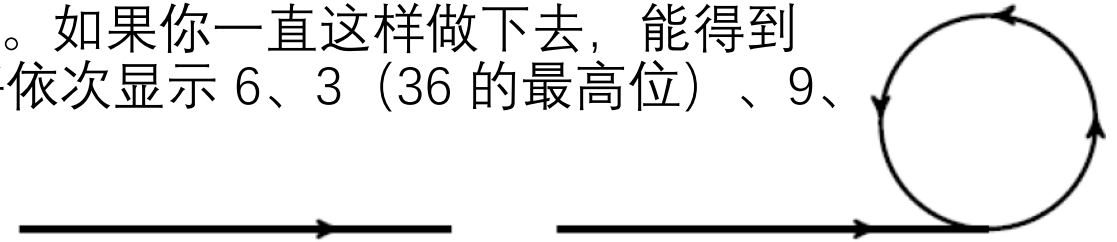
【分析】

1. 暴力的话用二重循环：可惜是$O(n^2)$的，n=100,000，规模太大。对于每个固定的j，应该选择的是i<j且A_i最大的 i，而和 A_j值无关。这样，从小到大枚举 j顺便维护A_i的最大值。
2. 复杂度为O(n)。每次用O(1)时间更新了MaxAi，而不是重新计算。
3. 如果要求输出对应的i和j怎么办？能否不用 A 数组，边读边计算？这样可以让空间复杂度降低到O(1)。

```
int T;
scanf("%d", &T);
while (T--) {
  scanf("%d", &n);
  _for(i, 0, n) scanf("%d",&(A[i]));
  int m = A[0], ans = A[0]-A[1];  //maxAi
  _for(i, 1, n) {
    // m is max{A0, A1, A_{i-1}}
    ans = max(m - A[i], ans);
    m = max(A[i], m);
  }
}
printf("%d\n", ans);
```

计算器谜题(Calculator Conundrum, UVa11549)

有一个老式计算器，只能显示 n 位数字。有一天，你无聊了，于是输入一个整数 k，然后反复平方，直到溢出。每次溢出时，计算器会显示出结果的最高 n 位和一个错误标记。然后你清除错误标记，继续平方。如果你一直这样做下去，能得到的最大数是多少？例如n=1, k=6 时，计算器将依次显示 6、3（36 的最高位）、9、8（81 的最高位）、6（64 的最高位）、3…

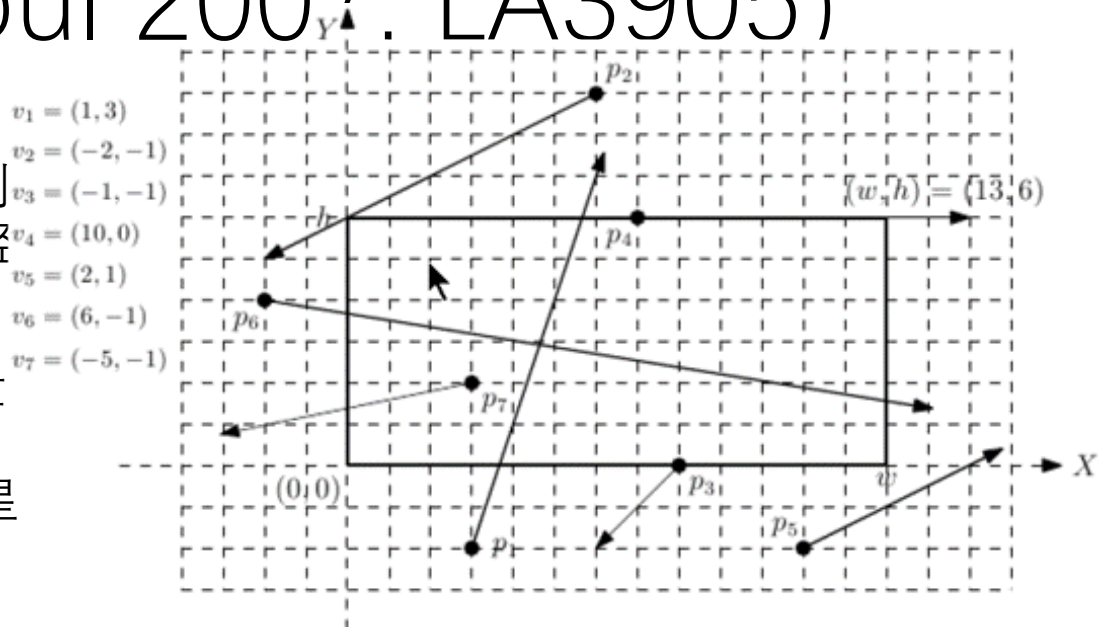


【分析】

1. 题目已经暗示了计算器显示出的数将出现循环(WHY?),
2. 所以不妨一个一个的模拟，每次判断新得到的数是否以前出现过。如何判断呢？一种方法是把所有计算出的数放到一个数组里，然后一一比较。要么太快，要么开不下。
3. 可以利用 STL 的set。也可以用哈希表，但和 set 一样，空间开销都比较大。
4. 想象有两个小孩子在一个“可以无限向前跑”的跑道上赛跑，同时出发，但其中一个小孩的速度是另一个的两倍。如果跑道是直的（如左图），跑得快的小孩永远在前面；但如果跑道有环（如右图），跑得快的小孩将“追上”跑得慢的小孩。
5. 这个算法称为Floyd判圈算法，空间复杂度降为 O(1)。

例题20 流星(Meteor, Seoul 2007, LA3905)

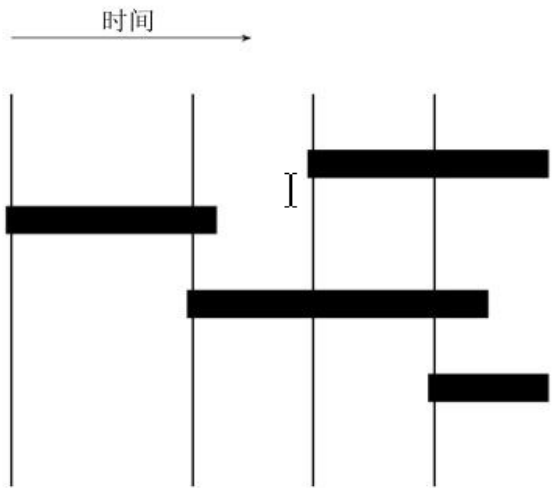
给一个矩形照相机，还有n个流星的初始位置和速度，求能照到的流星最多的时刻。注意，相机边界上的点不会被照到。如下图，流星 2, 3, 4, 5 都不会被照到，因为它们从来没有经过照相机的内部。



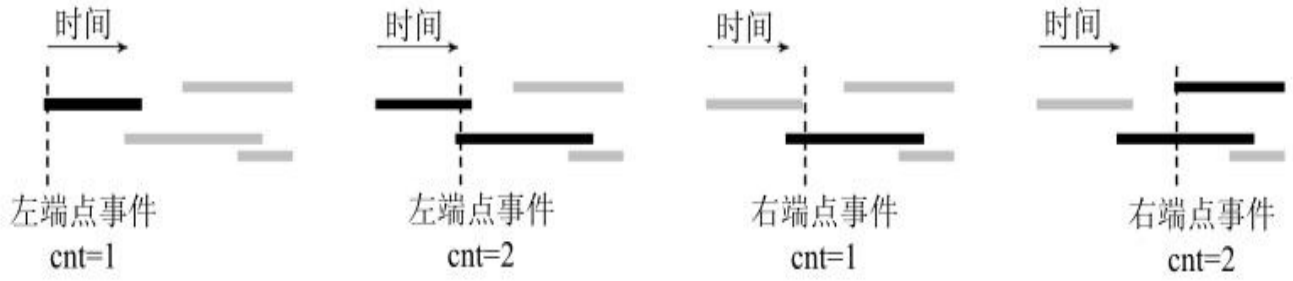
相机的左下角为(0,0)，右上角为(w,h)。每个流星用两个向量p和v表示，p为初始(t=0时)位置，v 为速度。在时刻t(t>=0)的位置是 p+tv。例如若 p=(1,3), v=(-2,5)，则t=0.5 时该流星的位置为(1,3) + 0.5*(-2,5) = (0, 5.5)。

【分析】

1. 流星轨迹是没有直接意义的，有意义的只是在照相机视野内出现的时间段。本题抽象为这样一个问题：给出n个开区间(L_i,R_i)，求一个数t，使得包含它的区间数最多(为什么是开区间?)。
2. 把所有区间画到平行于数轴的直线上（免得相互遮挡，看不清楚），然后想象有一条竖直线从左到右进行扫描，则我们的问题可以转化为：求扫描线在哪个位置时与最多的开区间相交。
3. 当扫描线移动到某个区间左端点的“右边一点点”时最有希望和最多的开区间相交（?）。



1. 为了快速得知在这些位置时，扫描线与多少条线段相交。使用前面提到的技巧：维护信息，而不是重新计算。
2. 把“扫描线碰到一个左端点”和“扫描线碰到一个右端点”看成是事件(event)，则扫描线移动的过程就是从左到右处理各个事件的过程。每遇到一个“左端点事件”，计数器加 1；每遇到一个“右端点事件”，计数器减 1。这里的计数器保存的正是我们要维护的信息：扫描线和多少个开区间相交。如下图：




```
给所有事件从左到右排序
while(还有未处理的事件) {
  选择最左边的事件 E
  if(E是“左端点事件”) {
    cnt++;
    if(cnt > ans) ans = cnt; //  更新计数器和答案
  } else cnt--; //  一定是“右端点事件”
}
```

1. 代码有个问题：如果不同事件的端点相同，哪个排在前面呢？考虑这样一种情况：输入是两个没有公共元素的开区间(L1,R1),(L2,R2)，且R1=L2。在这种情况下，两种排法的结果截然不同：如果先处理L2，结果是 2；如果先处理R1，执行结果是 1——这才是正确答案。
2. 最终版本：先按照从左到右的顺序给事件排序，对于位置相同的事件，把右端点事件排在前面，然后执行上述伪代码的循环部分。不妨把它理解成把所有区间的右端点往左移动了一个极小的距离。

子序列(Subsequence, SEERC2006,LA2678)

有n(10<N<100000)个正整数组成一个序列。给定整数S，求长度最短的连续序列，使它们的和≥S。

【分析】

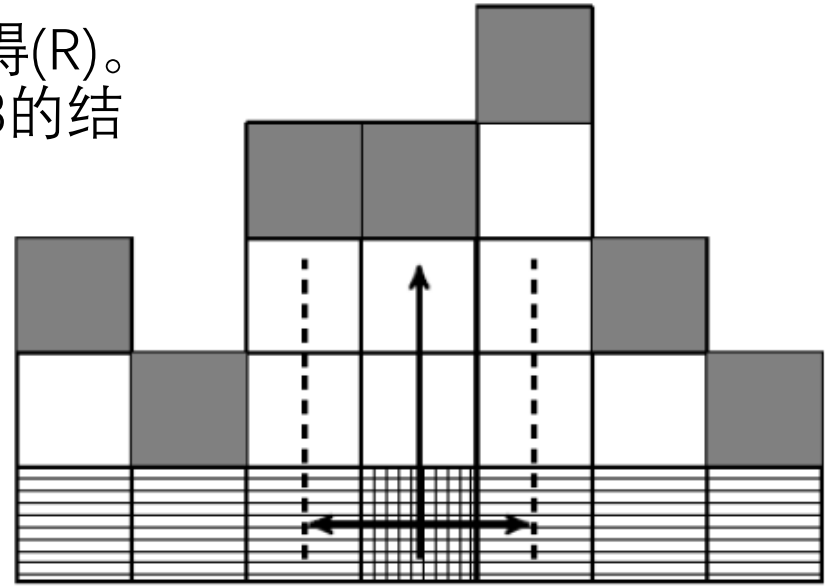
1. 直接的思路是二重循环，枚举子序列的起点终点，时间复杂度O(n^3)的，当 n 达到 100,000 的规模无能为力。使用前缀和技巧计算$\mathrm{B_i} = \sum_{k=1}^{i} A_i$，时间复杂度降为 O($n^2$)，依然TLE。 只要同时枚举起点和终点不可能比 O(n^2)更快，是否可以只枚举一个端点？
2. 试试只枚举终点。对于终点 j，我们的目标是要找到一个让 B_j-B_{i-1}>=S，且i尽量大(子序列长度 j-i+1越小)的 i 。也就是找一个让B_{i-1}≤B_j-S 的最大i。B是单调递增的(Ai>0），用二分查找。使用 STL, i=lower_bound(B, B+j, B[j]-S)。时间复杂度是O(nlogn)。
3. 继续优化到 O(n)：j和B_j都递增，B_{i-1}≤B_j-S 的右边递增。满足条件的i也递增。

```
int ans = n+1, i = 1;
for(int j = 1; j <= n; j++) {
    if(B[i-1] > B[j]-S) continue; // (1)  没有满足条件的 i，换下一个 j
    while(B[i] <= B[j]-S) i++; // (2)  求满足 B[i-1]<=B[j]-S 的最大 i
    ans = min(ans, j-i+1); // (3)
}
```

1. 二重循环：外层 j，内层 i。语句(1)和(2)的执行次数为 n，因为每个不同的 j 执行了一次;
2. (3)的执行次数有些复杂，不同的 j 对应的执行次数不一样。但i 从未减小，一直递增，递增次数一定不超过 n。
3. 时间复杂度为 O(n)。

最大子矩阵(City Game, SEERC 2004, LA 3029)

给定一个 m*n 的矩阵，其中一些格子是空地(F)，其他是障碍(R)。找一个全部由F组成的，面积最大的子矩阵，输出面积乘以3的结果。



j:	0	1	2	3	4	5	6
up:	2	1	3	3	4	2	1
left:	0	0	2	2	4	2	0
right:	0	6	4	4	4	5	6

【分析】

1. 暴力做法：枚举左上角坐标和长、宽，判断这个矩形是否全为空地。需要枚举 $O(m^2n^2)$个，判断还需要 O(mn)时间，总时间复杂度为 $O(m^3n^3)$，太高。虽然是矩形，但仍然可以用扫描法：从上到下扫描。
2. 把每个格子向上延伸的连续空格看成一条悬线，并且用 U(i,j), L(i,j), R(i,j)表示(i,j)的悬线长度、以及向左向右运动的“极限”
    1. 如下图：列3的悬线长度为3，向左向右各能运动一列，因此左右运动极限分别为列2和列4。

每个(i,j)对应着一个以第i行为下界、高度为U(i,j)，左右边界分别为L(i,j),R(i,j)的矩形。不难发现，所有这些矩形中面积最大的就是题目所求(WHY?)。如何快速计算出U,L,R?

1. (i,j)不是空格，U,L,R为0；U(i,j)=U(i-1,j)+1；L(i,j) = max{L(i-1,j), lo+1}
2. lo 是第 i 行中，第j列左边的最近障碍格的列编号。从左到右的计算L(i,j), 很容易维护lo。
3. R同理从右往左计算，因为要维护第 j 列右边最近的障碍格的列编号 ro。

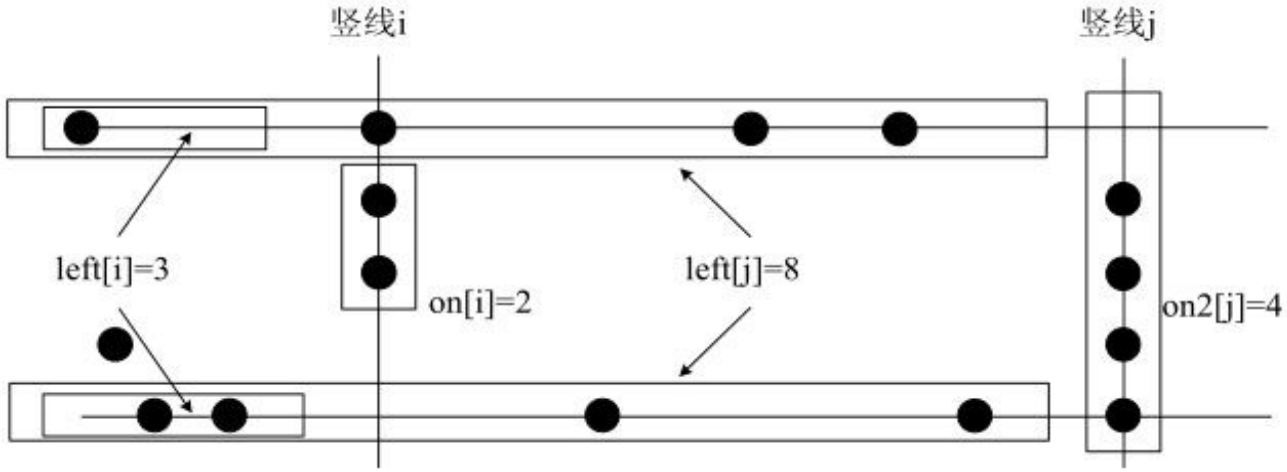
可以用U[j], L[j]和R[j]来保存当前扫描行上的信息。类似于DP中的滚动数组。

遥远的银河(Distant Galaxy, Shanghai2006,LA3695)

给平面上n(n≤100)个点，找一个矩形，边界上包含尽量多的点

【分析】

1. 除非所有输入点都在同一行或者同一列上(答案为n)，最优矩形的四条边都至少有一个点(一个角上的点同时算在两条边上）。可以枚举四条边界所穿过的点，然后统计点数。时间复杂度为 $O(n^5)$（统计点数还需要 O(n)），无法承受。
2. 和《子序列》一题类似，可以考虑只枚举上下边界，用其他方法确定左右边界，过程如图



1. 对于竖线i，用L[i]表示i左边位于上下边界上的点数(不统计竖线上的点)，on[i]和on2[i]都表示竖线上位于上下边界之间的点数，on不统计位于上下边界上的，而on2要统计。
2. 给定左右边界i,j，矩形边界上的点数为L[j]-R[i]+on[i]+on2[j]。j确定时，on[i]-left[i]应最大。
3. 枚举完上下边界后，花O(n)时间从左到右扫描所有点，计算left，on和on2数组，然后枚举右边界 j，同时维护 on[i]-left[i](i<j)的最大值。
    1. 这一步本质上等价于例题《开放式学分制》。

侏罗纪(Jurassic Remains,NEERC2003,LA2965)

给定n(n≤24)个大写字母字符串。选择尽量多的串，使得每个大写字母都出现偶数次。

【样 例 输入】 6 ABD EG GE ABE AC BCD

A	B	C	D	E	F	G	H	…			
1	1	0	1	0	0	0	0	…	A	B	D
0	0	0	0	1	0	1	0	…	E	G	
0	0	0	0	1	0	1	0	…	G	E	
1	1	0	0	1	0	0	0	…	A	B	E
1	0	1	0	0	0	0	0	…	A	C	
0	1	1	1	0	0	0	0	…	B	C	D

【分析】

1. 一个字符串中，每个字符出现的次数本身无关紧要，重要的是其奇偶性，可以用一个二进制位表示一个字母(1表示出现奇数次，0 表示偶数次)。比如样例的 6 个串，写成二进制分别为上图。
2. 问题转化为求尽量多的数，使得它们的 xor值为0。
3. 直接穷举，时间是O(2^n)。注意xor=0的两个整数相等，把字符串集合分成两部分，先计算前n/2个字符串所能得到的所有xor值，保存到一个映射S(xor值→前n/2个字符串的一个子集)中，然后枚举后n/2 个字符串所能得到的所有 xor 值，每次都在 S 中查找。
4. 映射用 STL 的 map 实现，总时间为 O($2^{n/2}$logn)，即 O(1.44^nlogn)，快了很多。
    1. 这样的策略称为中途相遇法（Meet-in-the-Middle）。密码学中著名的中途相遇攻击就是基于这个原理。