问题中的变与不变

陈雪

长沙市雅礼中学 410007

【摘要】

信息学竞赛中很多试题本质上都是对变量进行求解或者维护的过程。而有时候变量有很多种变化，单纯的维护变量往往导致时空复杂度的低下。而如果能在变量的变化里找出其中“不变”的常量，往往能将问题迎刃而解。本文简单的介绍了几种在变量中找出“不变”的技巧，使得问题得到简化。

【关键字】

变 变量 不变 常量 维护

【引言】

变量在信息学中是最常见的，最基本的有循环变量，累计变量，决策变量。维护这些变量也是信息学中最基本的操作。信息学中关于变量的维护也有很多数据结构，像平衡二叉树，栈，队列等等。

然而有些问题规模很大，仅仅是所需要的变量进行维护的总操作数就会达到无法承受的时空复杂度。于是我们就要对变量进行化简，找出其中的有用信息，从而使得维护这些变量的操作数更少，甚至将一些变量化成“常量”。

下面就让我们具体看看这类技巧在信息学中的应用。

【正文】

将有用的变量放在一起看成一个集合，利用特殊的性质，进行一系列的操作，比如加发，乘法，是将变量转化成常量的最常见的方法。下面我们就看这样的一道问题。

例 1：蚂蚁 ants[1]

一些蚂蚁以 1cm/s 的速度在长度为 *l*cm 的线段上爬行，爬到线段端点就会掉下去。当两只蚂蚁相遇，就会立刻掉头返回。已知 *l* 和一开始每只蚂蚁的位置但不知道它们的方向，求它们最早何时全部掉落，最迟何时全部掉落。

最多 1,000,000 只蚂蚁。

分析：蚂蚁一出来我们只知道位置，不知道方向。如果单纯枚举方向的话，光是各种方案的可能性就有 $2^{1000000}$ 种，这个一个无法承受的数字。因此不可能去枚举每个蚂蚁的方向，不如先从简单的问题着手。假设我们已知了蚂蚁初始的方向后接下来的任务就是模拟所有的蚂蚁相遇和掉落的过程，求出最后一个掉落的时间。在这个过程中，最复杂的变量就是每只蚂蚁的方向了，明显在最坏情况下相遇次数可能达到 N^2 级别，仅仅是维护每只蚂蚁的方向和位置都是不能满足题目

的时间复杂度的要求的。

我们从细节开始考虑问题：

首先考虑题目中最基本的一个操作也就是 2 只蚂蚁碰头的情况：

假设一只蚂蚁 A 从左向右爬行，在 X 点的时候遇到了另一只从右向左的蚂蚁 B 后返回。同时蚂蚁 B 也从 X 向右返回。

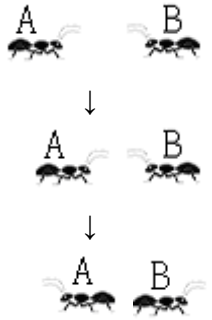

图 1 2 只蚂蚁相撞后的情况

将这 2 只蚂蚁有用的信息也就是速度 V_i（V_i 是向量)和当前位置 W_i 记录下来一起构成集合 U，也就是 $U=\{(V_a,W_a),(V_b,W_b)\}$。

在蚂蚁 A 和蚂蚁 B 相遇前瞬间一刻，$U=\{(1,X),(-1,X)\}$

而在蚂蚁 A 和蚂蚁 B 相遇后瞬间一刻，$U'=\{(-1,X),(1,X)\}$

我们发现 U=U’，也就是说虽然对于每只蚂蚁，它的速度发生了改变，但是把这 2 只蚂蚁放在一个集合看的话，这个集合并没有发生变化。或者说，在同一个集合内的蚂蚁相遇，这个集合是不会变的。

于是把所有的蚂蚁看成一个大集合 $V=\{(V_1,W_1),(V_2,W_2)...(V_n,W_n)\}$。

根据上面说提到的，在这个集合 V 内的任何相遇对于集合 V 都没有影响。

同时我们通过集合 V 可以知道，虽然每只蚂蚁掉落的时间 Ti 无法求出，但是所有的 Ti 所构成的集合 T 我们是可以求出来的。

事实上就是 T={第 I 只蚂蚁按初始方向走到一端点的时间|1≤i≤N}。

得到了这个有效的结论后，我们回到原来的问题：它们最早何时全部掉落，最迟何时全部掉落。

先考虑最早何时全部掉落：根据 T={第 I 只蚂蚁按初始方向走到一顶点的时间|1≤i≤N}，我们贪心的让每只蚂蚁都朝离自己最近的端点爬行，即可保证 T 中最大值最小。

同理最迟何时全部掉落的情况就是每只蚂蚁朝离自己最远的端点爬行，即可保证 T 中最大值最大。

最终我们得到了一个时间复杂度为 O(N)的算法了。这已经是理论的下限了。

小结

通过集合的概念，我们将原来要维护每个蚂蚁的方向,位置这些规模庞大的变量转化成一些“常量”。而通过这些“常量”不用维护就可以直接得出最终结果。

集合操作在这类变量转化成“常量”的问题有极广泛地运用，常常可以利用题目给定的条件，来构造集合，然后利用集合内元素之间的特殊性质来判断无解或者优化算法。

通过这道题目我们也能看到，仔细观察，从细节入手，寻找变量之间的联系，才能找到将变量转化成“常量”的方法。

例 2：Navigation Game[2]

这里有一个 N 行 M 列的方阵，第 N 行象征着这里，第 1 行象征着海那边的彼岸。这中间的 N-2 行象征着你所期盼的大海。

你的目标是，控制一艘船，从这里的任意一个停泊处（用“H”表示），经过最短的航行时间到达对岸的任意一个停泊处。只有这样，你才可以通过这个通道

你的船只能向左，向右或向上前进，一次一格，而且除非登陆（这时你必须到达一个停泊处，从而结束你的游戏），你是不能驶向陆地的。

记住，人生没有回头路。因此，一旦你离开一个格子，就永远也无法返回。

向着目标航行永远是一件令人愉快的事情。因此向上航行只需要消耗一个单位的时间。但是，看似原地打转的左右方向的航行会让人厌倦。如果某次左右航行之前你已经连续进行了 x 次左右航行，你这次左右航行所消耗的时间就是 x+1 个时间单位。

海上你可能会遇到：

O：障碍物。障碍物占据的格子，你永远也不会到达。

F：命运之轮。经过这里，你的命运会从此逆转。我了解你的命运有多么不幸。因此，你必须在航行途中经过奇数次命运之轮，才能安全到达彼岸。

B：祝福石。走到这里是不需要时间的。

S：暴风雨的咒符。走到这里所需的时间是正常情况的两倍。

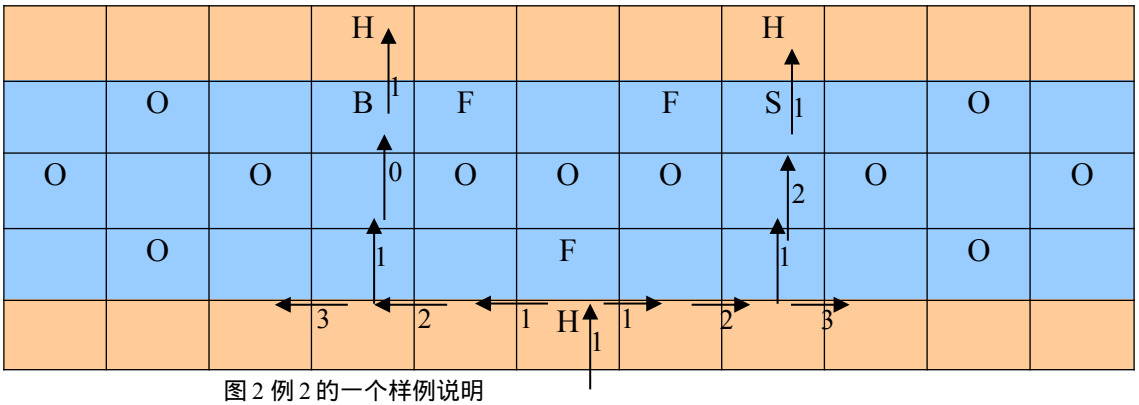

图 2 例 2 的一个样例说明

数据范围约定: N,M≤1000。

分析：根据题目的要求，不能走回头路，同时只能向上或者向右或者向左，于是我们考虑使用动态规划：设 $f_{t,i,j}$ 表示从起点到(I,J)且走过的命运之轮的次数为 T 的最少的代价(T 表示奇偶性)。设 $C_{i,j}$ 表示从(I+1,J)到(I,J)的代价。

于是得到状态转移方程:

$$f_{t,i,j} = Min\{f_{t',i+1,k} + C_{i,k} + \cos t_{i,k,j}\}$$

（其中 $\cos t_{i,k,j}$ 表示从(I,K)移动到(I,J)按照题目所给要求的代价，且(I,K)

到(I,J)之间没有障碍物,T'=T xor (I,K)到(I,J)之间 F 个数）

假设如果没有题目所限制 O,S 的限制，那么 $\cos t_{i,k,j} = \frac{|j-k|(|j-k|+1)}{2}$。

首先我们只考虑从左往右走。那么状态转移方程可以写成:

$$f_{t,i,j} = Min\{f_{t',i+1,k} + C_{i,k} + \frac{(j-k)(j-k+1)}{2}\}$$

不妨另 $U_j = f_{t,i,j}, V_k = f_{t',i+1,k} + C_{i,k}$。状态转移方程就变成了

$$U_i = Min\{V_j + \frac{(i-j)(i-j+1)}{2}\}$$

考虑 Ui 的最优决策 K，那么满足(i>k>l)：

$$V_k + \frac{(i-k)(i-k+1)}{2} < V_l + \frac{(i-l)(i-l+1)}{2} \longrightarrow \frac{(2V_k + k(k-1)) - (2V_l + l(l-1))}{2(k-l)} < i$$

另 $X_i = 2V_i + i(i-1), Y_i = 2i$。把 X_i, Y_i 画在平面坐标轴上，根据上面的式子有(X_k, Y_k)-(X_l, Y_l)之间的斜率小于 i 的时候，决策 k 比决策 L 更优。事实上这是一个经典的斜率优化动态规划的模型[3]。根据一般性我们知道维护一个队列表示(X_i, Y_i)的凸包中的下凸部分即可在均摊 O(n)的时间内求出 U_i 最优值。具体操作如下：用 stack 队列来表示这个凸包的下凸部分。同时设队首指针 st 和队尾指针 en。根据 Stack 队列维护的是凸包的下凸部分，那么满足下面的性质 2.1:

1. $stack_1 < stack_2 < \ldots < stack_{en}$
2. $stack_1 - stack_2$ 的斜率< $stack_2 - stack_3$ 的斜率. < $stack_{en-1} - stack_{en}$ 的斜率

具体操作如下：

从小到大枚举 I。对于当前 I，如果满足 $stack_{st} - stack_{st+1}$ 的斜率<i，则不断删除队首元素。此时队首元素就是当前 I 的最优决策。同时把 I 加入队列队尾，维护队尾满足性质 2.1。同样我们也可以用类似的方法求出从右向左走的最小值。

但是这道题目有个很独特条件：经过 S 不用时间和经过 B 时间加倍。

由于从不同的起点经过一个 S 或者 B 产生的代价的改变是不同的，这个变化值决定了上面所说的方法不能够直接搬到这个问题来。

回到原来的方程 $f_{t,i,j} = Min\{f_{t',i+1,k} + C_{i,k} + \cos t_{i,k,j}\}$ 上。

还是先考虑从左往右走，即 j>k。显然，无论(I,J)-(I,K)之间无论是 B 或者 S，变量 $\cos t_{i,k,j}$ 都能表述成一个关于(J-K)的二次函数。不妨设 $\cos t_{i,k,j}=\frac{(j-k+1)(j-k)}{2}+a(j-k)+b$。当然 a 和 b 都是需要维护的变量。

先考虑如何维护 $\cos t_{i,k,j}$。

假设 K 已经固定，当前 $\cos t_{i,k,j}=\frac{(j-k+1)(j-k)}{2}+a(j-k)+b$。下面观察 $\cos t_{i,k,j+1}$。

1.(I,J+1)上不是 B 也不是 S:

有 $\cos t_{I,K,J+1}=\cos t(I,K,J)+(J-K+1)=\frac{(j-k+2)(j-k+1)}{2}+a(j-k+1)+b-a$

2.(I,J+1)上是 B,不需要时间:

有 $\cos t_{I,K,J+1}=\cos t(I,K,J)=\frac{(j-k+2)(j-k+1)}{2}+(a-1)(j-k+1)+b-a$

3.(I,J+1)上是 S,时间要加倍:

有 $\cos t_{I,K,J+1}=\cos t(I,K,J)+2(J-K+1)=\frac{(j-k+2)(j-k+1)}{2}+(a+1)(j-k+1)+b-a$

根据上面所说，我们及时更新 a,b 就能得出新的 $\cos t_{i,k,j+1}$。同时我们也能看到 a 的意义就是在(I,K+1)-(I,J)这一段中，$a=\sum s-\sum b$。

我们看另一种情况:如果 J 固定，而 K 发生变化的时候，$\cos t_{i,k,j}$ 的维护。

重新扫描一遍肯定会导致算法效率低下。而且 $\cos t_{i,k,j}$ 变量一共有 $O(n^3)$的级别，即使我们能在均摊 O(1)的时间内算出所有 $\cos t_{i,k,j}$ 时间复杂度依然是 $O(n^3)$的。于是考虑优化算法，尽量在短时间内算出对我们有用的信息。

假设当前已知 $\cos t_{i,k,j}=\frac{(j-k+1)(j-k)}{2}+a(j-k)+b$，而且已知 $\cos t_{i,k,l}=\frac{(l-k+1)(l-k)}{2}+a'(l-k)+b'(l>k)$ 的 a'和 b'。下面我们要求 $\cos t_{i,l,j}$。既然重新求 $\cos t_{i,l,j}$ 没有太好的方法，我们干脆做一个大胆的猜想那就是希望改变一些变量值使得 $\cos t_{i,k,j}$ 的 a 和 b 能直接满足 $\cos t_{i,l,j}$ 的要求。把 $\cos t_{i,k,j}$ 的代价分为 2 个部分，一部分表示从(I,K)出发在 (I,K)-(I,L)这一段的代价，另一部分表示在(I,L+1)-(I,J)这一段的代价。而第 1 部分的代价显然就是 $\cos t_{i,k,l}$。第

二部分当中既有对我们有用的代价 $\cos t_{i,l,j}$,同时也有一部分 $\Delta\cos t$ 的多余代价(从 L 出发比 K 节省的时间)。于是我们得到

$$f_{t',i+1,l}+C_{i,l}+\cos t_{i,l,j}=f_{t',i+1,l}+C_{i,l}+\cos t_{i,k,j}-\Delta\cos t-\cos t_{i,k,l}$$

$$=f_{t',i+1,l}+C_{i,l}+\frac{(j-l+1)(j-l)}{2}+a(j-l)+b-b'$$

于是我们只要给 $f_{t',i+1,l}+C_{i,l}-b'$ 就无需改变 a 和 b。而这样做会不会对 $\cos t_{i,l,j'}$ 产生影响？我们分析由 $\cos t_{i,l,j}$ -> $\cos t_{i,l,j'}$ 的过程，就能知道这样做是没有任何影响的。而这样做的好处就是能把 $\cos t_{i,k,j}$ 用一种更具体的形式给出而不是一个虚拟的变量。实际上为了方便处理我们可以另 $g_{t',i,j}=f_{t',i+1,j}+c_{i,j}+b$。(b 就是上文 $\cos t_{i,l,j}$ 中的 b',也可以看成是 $f_{t,i,j}$ 的最优决策 k， $\cos t_{i,k,j}$ 中的那个 b)

这样原状态转移方程就可以写成

$f_{t,i,j}=Min\{g_{t',i,k}+\cos t_{i,k,j}\}=Min\{g_{t',i,k}+\frac{(j-k+1)(j-k)}{2}+a(j-k)+b\}$ (虽然 a,b 是变化的，但是对于单独的一个 $f_{t,i,j}$，a,b 是固定的)。

仿照没有 B 和 S 时的思路，设最优决策 K，同时观察决策 L。

$$g_{t',i,k}+\frac{(j-k)(j-k+1)}{2}+a(j-k)<g_{t',i,l}+\frac{(j-l)(j-l+1)}{2}+a(j-l)$$

$$\longrightarrow\frac{(2g_{t',i,k}+k(k-1))-(2g_{t',i,l}+l(l-1))}{2(k-l)}<j-\frac{a}{2}$$

另 $X_k=2g_{t',i,k}+k(k-1)$， $Y_k=2k$。还是仿照上面的思路维护一个队列表示 (X_i,Y_i) 所构成的凸包的下凸部分。具体操作同没有 B 和 S 的情况类似，这里就不重复了。但是这样是否依然能保证正确性？

每次 J 会增加 1，而无论(I,J)上是 B 或者 S，a 只会发生 1 的变化。于是右边的改变量 ΔJ 依然能保证是非负的。所以正确性显然。

而从左到右反过来考虑即可。

总的时间复杂度:O(行数*(列数+队列的操作数))=O(NM)。

小结

这道题目中，由于特殊的条件的约束，如果使用其他方法维护 cost(I,K,J)中的 a,b 反而令问题更加复杂。于是大胆猜想，通过调整其他变量使 a,b 不变，从而在后面的问题中，利用队列和斜率将算法优化到 O(NM)。

例三：Circular Railway[4]

在一条长度为 L 的环形轨道上有 N 个 A 类点，N 个 B 类点。现在要你安排一种方案使得这 N 个 A 类点和 N 个 B 类点一一匹配。并且所有匹配点之间的最短距离（指在轨道上）和最小。

数据范围: $1 \le n \le 50000, 2 \le L \le 10^9$

分析：首先将 N 个 A 类点按顺时针排序后设他们的坐标为 $A_1, A_2 \cdots A_n$。

将 N 个 B 类点按顺时针排序后设他们的坐标为 $B_1, B_2 \cdots B_n$。(同时另 $B_{i+n} = B_i$)

显然问题实际上就是一个二分图的最小权匹配。但是最小权匹配最坏情况下是 $O(n^4)$的时间复杂度，对于题目的数据范围，肯定是不能满足要求的。

仔细分析我们就能发现最小权匹配时有一条重要的性质:

性质 3.1: 2 条匹配边肯定不会相交。

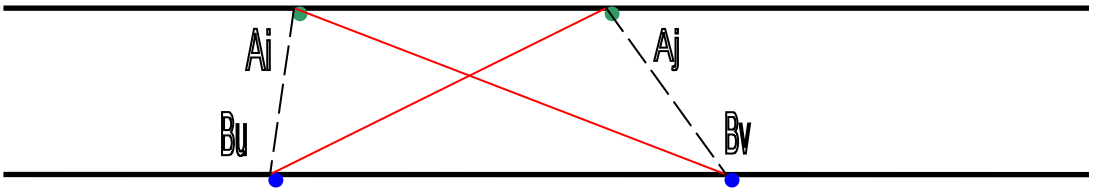

图 3 性质 3.1 的证明

反证：如果出现了 2 条匹配边 A_i - B_v 和 A_j - B_u 相交，那么我们交换 A_i, A_j 的匹配使得 A_i - B_u 和 A_j - B_v 肯定可以让匹配的总权值最小。（如图 3 所示）

按照性质 3.1，我们可以知道最小权匹配必然满足下面的条件：

不妨另 A_1 匹配的点是 B_j，然后 A_i (1<i≤n)匹配的点就是从 B_j 开始顺时针数到的第 i 个 B 类点。（如图 4 所示，绿色代表 A_i，蓝色代表 B_i）

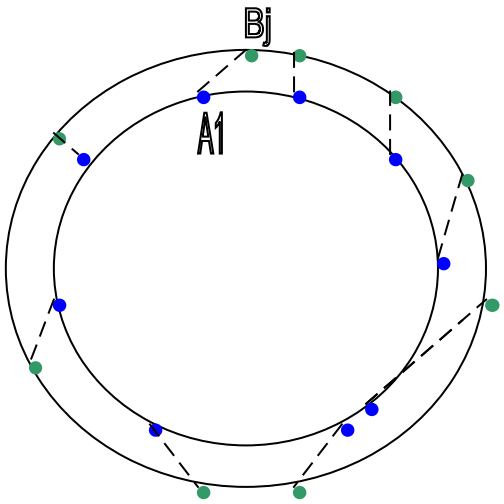

图 4 最小权匹配的方案

既然最小权匹配满足上述条件，我们得到了算法 3.2:枚举与 A_1 匹配的点 B_j，然后根据上面的条件，可以知道 A_i 匹配的点就是 B_{i+j-1}。然后根据每条匹配求出总代价，这当中最小的代价就是我们要求的最小权匹配了。

算法 3.2 的时间复杂度为 $O(n^2)$。尽管对于 n=50000 仍然有些力不从心，但是已经有了很大的进步了。

继续分析假设 A_i 匹配的点 B_{Q_i}，同时设 C_i 表示 A_i 至 B_{Q_i} 的最短距离。那么我们知道在上面所说的算法中，只要维护了变量 C_i，就能得出最优答案。但是仅仅是把所有的 A_i 至 B_{Q_i} 的最短距离求出来就已经 $O(n^2)$，也就是说 Ci 一共有 n^2 种不同的值。这也意味着维护或者预处理计算 Ci 的时间复杂度都是 $O(n^2)$的。为了优化算法，我们要考虑另辟捷径。

换个角度考虑 Ci。下面具体分析 Ci。同时令 j= Q_i。

下面考虑 $C_i = A_i \longrightarrow B_j$ 的几种取值可能。

1. $C_i = A_i \longrightarrow B_j = A_i - B_j$

2. $C_i = A_i \longrightarrow B_j = B_j - A_i$

3. $C_i = A_i \longrightarrow B_j = L + A_i - B_j$

4. $C_i = A_i \longrightarrow B_j = L + B_j - A_i$

下面继续分析最优值分别为每种情况的时候 A_i 和 B_j 关系。

1.当 $C_i = A_i - B_j$ 最优，此时 A_i 和 B_j 要满足条件：

1. A_i > B_j　2. A_i - B_j ≤L/2

2.当 $C_i = B_j - A_i$ 最优，此时 A_i 和 B_j 要满足条件：

1. B_j > A_i　2. B_j - A_i ≤L/2

3．当 $C_i = L + A_i - B_j$ 最优，此时 A_i 和 B_j 要满足条件：

1. B_j - A_i >L/2

4．当 $C_i = L + B_j - A_i$ 最优，此时 A_i 和 B_j 要满足条件：

1. A_i - B_j >L/2

通过上面的分析发现 Ci 只与当前的 A_i 和 B_j 有关，既然关于变量 Ci 无能为力，那么我们不妨把注意力放到 A_i 和 B_j 上来，不妨另 $C_i = f(A_i) + g(B_j)$。下面考虑 A_i 对 Ci 的影响，也就是函数 $f(A_i)$。

根据上面的分析，我们知道当

1．当 B_j - A_i >L/2 有，$f(A_i) = A_i$　①

2．当 L/2≥ B_j - A_i >0 有，$f(A_i) = -A_i$　②

3．当 L/2≥ A_i - B_j >0 有，$f(A_i) = A_i$　③

4．当 L/2< A_i - B_j 有，$f(A_i) = L - A_i$　④

我们把注意力放到变量 $f(A_i)$ 上来，通过上面的讨论，可以知道 $f(A_i)$ 虽然还是个变量，但是与 Ci 相比，当 j 从 1 递增到 N 这个过程中 $f(A_i)$ 最多有 4 种情况，事实上 1,4 不会同时出现，只有 3 种。也就是说 $f(A_i)$ 可以看成是几乎不变的“常量”了。

同样的方法分析 $g(B_j)$ 最多也只有 4 种变化。

现在我们从小到大枚举和 A1 匹配的点 Bk 后，其他的点按顺序一一匹配，现在的总代价 $Sum = \sum C_i = \sum (f(A_i) + g(B_j)) = \sum f(A_i) + \sum g(B_i)$。

当 k 变化的时候，我们只看 $Sum = \sum f(A_i) + \sum g(B_i)$ 的变化，把 Ci 放到

一边。根据上面的分析，在我们从 1 到 n 枚举 K 的过程中，每一个 $f(A_i)$，$g(B_i)$ 最多只会变 4 次。而我们可以通过预处理知道对于每个 $f(A_i)$，①变到②是在 K= X_1 变成 K= X_1 +1 时改变的，②变到③是在 K= X_2 变成 K= X_2 +1 时改变的，③变到④是在 K= X_3 变成 K= X_3 +1 时改变的。同理也可以通过预处理知道每个变量 $g(B_i)$ 它发生改变的时间。不妨把 $f(A_i)$，$g(B_i)$ 发生改变看成一次事件，可以知道事件总数肯定不会超过 8n。假设我们已经通过预处理知道每件事件发生的时间（每次事件的时间 x 指事件是在 k=x 变成 k=x+1 发生的）。而通过这些事件我们就可以即时维护 Sum。在 Sum 达到最小值时，就是我们要求的答案了。

整个算法的流程如下：

1.将 Ai 和 Bi 排序。

2.预处理求出每个事件发生的时间。

3.从 1 到 N 枚举 K，同时通过 hash 表找出哪些事件发生在这个时刻，同时维护 Sum。

4.输出 Sum 的最小值。

下面我们考虑预处理。预处理求 X_1，X_2，X_3 实际上就是要我们求对于每个 Ai 情况①②③④所对应的 J 的范围。而根据 A_i 和 B_i 的有序性，我们用指针就能维护情况①②③④所对应的 J 的范围。

时间复杂度分析：O(排序复杂度+预处理+事件总数)=O(nlgn+n+8n)=O(nlgn)

回顾整个过程，我们发现只要牢牢把握住问题的实质，抓住最重要的变量——总代价，进而把每项 Ci 分开看成 2 个不相干的部分，对 2 个“常”量 $f(A_i)$，$g(B_i)$ 分别进行讨论求值，最终将算法优化到了一个满意的时间复杂度。

【总结】

本文简要的举出了 3 道例题，介绍了在信息学中，将变量化成“不变”的技巧。实际上，这种技巧在很多地方中都有建树。本文限于篇幅，介绍了三种比较典型的方法。从这些题目中也能看到，这种技巧的运用远不是简单的化简，有些变量之间的关系存在的很隐蔽，这就需要我们通过仔细观察，大胆猜想，深入分析才能找到合适的“常”量。

【参考文献】

[1] zju online judge 2376 ants. http://acm.zju.edu.cn/show_problem.php?pid=2376

[2] pku online judge 2841 navigation game. http://acm.pku.edu.cn/JudgeOnline/problem?id=2841
[3] 汤泽. 浅析队列在一类单调性问题中的应用.2006 集训队论文.
[4] sgu online judge 313 circular railway. http://acm.sgu.ru/problem.php?contest=0&problem=313
[5] 刘汝佳,黄亮.算法艺术与信息学竞赛. 清华大学出版社.2003