从一类单调性问题看算法的优化

湖南省长沙市第一中学　汤泽

【关键字】

数据关系　　队列　　单调性

【摘要】

充分挖掘数据关系，往往是构造出优秀算法的关键因素。本文从单调性入手，详细讨论了允许在表的尾端进行插入，而在两端删除元素的特殊队列对一类单调性问题的优化方法，并以此说明充分利用数据关系对构造优秀算法的重要性。

【正文】

对于很多问题，如果我们充分挖掘问题当中隐含的数据关系，并对某些简单的数据结构作出相应变形，应用于这些数据关系，就能以较低的编程复杂度来实现算法的优化。本文将通过一种特殊队列在一类单调性问题中的运用，来讨论这种思想的具体应用。

队列是一种我们非常熟悉的数据结构。最常见的队列是一种先进先出的线性表：它只允许在表的一端进行插入，而在另一端删除元素。我们对这种常见队列稍作变形，构造出一个特殊队列：它允许在表的尾端进行插入，而在两端删除元素。对于一些问题，如果能够挖掘出问题中隐含的单调关系，这种特殊队列能够很好地帮助我们完成算法的优化。

一、 在动态规划问题中的应用

运用单调性和这种特殊队列进行优化的例子最常见于动态规划问题当中。有些动态规划问题，可以利用决策的单调性，运用这种特殊队列来实现“降一维”。下面是一个具体的问题。

【问题一】锯木场选址（CEOI2004）

从山顶上到山底下沿着一条直线种植了 n 棵老树。当地的政府决定把他们砍

下来。为了不浪费任何一棵木材，树被砍倒后要运送到锯木厂。

木材只能按照一个方向运输：朝山下运。山脚下有一个锯木厂。另外两个锯木厂将新修建在山路上。你必须决定在哪里修建两个锯木厂，使得传输的费用总和最小。假定运输每公斤木材每米需要一分钱。

任务

你的任务是写一个程序：

从标准输入读入树的个数和他们的重量与位置

计算最小运输费用

将计算结果输出到标准输出

输入

输入的第一行为一个正整数n——树的个数（$2 \leqslant n \leqslant 20\,000$）。树从山顶到山脚按照 1，2……$n$标号。接下来$n$行，每行有两个正整数（用空格分开）。第$i$+1 行含有：$w_i$——第$i$棵树的重量（公斤为单位）和 d_i——第i棵树和第i+1 棵树之间的距离，$1 \leqslant w_i \leqslant 10\ 000$，$0 \leqslant d_i \leqslant 10\ 000$。最后一个数$d_n$，表示第n棵树到山脚的锯木厂的距离。保证所有树运到山脚的锯木厂所需要的费用小于 2000 000 000 分。

输出

输出只有一行一个数：最小的运输费用。

样例

输入

```
9
1 2
2 1
3 3
1 1
3 2
1 6
2 1
1 2
1 1
```

输出

```
26
```

在解决这一问题时，首先我们要明确，将锯木厂建立在相邻两棵树之间是没有任何意义的，否则我们可以将这样的锯木厂上移到最近的一棵树处，此时运送上方树木的费用减少，运送下方树木的费用没有变化，总费用降低。

为了方便讨论，我们先作如下定义：

假设山脚锯木场处也有一棵树，编号为 $n+1$，并且 $w[n+1]=d[n+1]=0$。

$sumw[i]$ 表示第 1 棵树到第 i 棵树的质量和，即 $sumw[i]=\sum_{j=1}^{i} w[j]$。

$sumd[i]$ 表示第 1 棵树到第 i 棵树的距离，即 $sumd[i]=\sum_{j=1}^{i-1} d[j]$。特别的，有 $sumd[1]=0$，表示第 1 棵树到自己的距离为 0。

$c[i]$ 表示在第 i 棵树处建一个锯木厂，并且将第 1 到第 i 棵树全部运往这个伐木场所需的费用。显然有 $c[i]=c[i-1]+sumw[i-1]\times d[i-1]$。特别的，有 $c[1]=0$。

$w[i,j]$ 表示在第 i 棵树处建一个锯木场，并且将第 $j+1$ 到第 i 棵树全部运往这个锯木场所需的费用。则 $w[i,j]=c[i]-c[j]-sumw[j]\times(sumd[i]-sumd[j])$。特别的，当 $i=j$ 时 $w[i,j]=c[i]$。

综上可知，求出所有 $sumw[i], sumd[i]$ 与 $c[i]$ 的时间复杂度为 $O(n)$，此后求任意 $w[i,j]$ 的时间复杂度都为 $O(1)$。

设 $f[i]$ 表示在第 i 棵树处建立第二个锯木场的最小费用，则有 $f[i]=\min_{1\le j\le i-1}\{c[j]+w[i,j]+w[n+1,i]\}$。直接用这个式子计算的时间复杂度为 $O(n^2)$，由于问题规模太大，直接使用这一算法必然超时，因此我们必须对算法进行优化。在讨论如何进行优化以前，我们首先证明下面这一猜想。

[猜想]

如果在位置 i 建设第二个锯木厂, 第一个锯木厂的位置是 j 时最优。那么如果在位置 $i+1$ 建设第二个锯木厂, 第一个锯木厂的最佳位置不会小于 j。

证明：

令 $s[k,i]$ 表示决策变量取 k 时 $f[i]$ 的值，即 $s[k,i]=c[k]+w[i,k]+w[n+1,i]$。

设 $k1<k2$，则有

$$s[k1,i]-s[k2,i]=sumw[k2]\times(sumd[i]-sumd[k2])-sumw[k1]\times(sumd[i]-sumd[k1])$$

若 $s[k1,i]-s[k2,i]<0$，则有

$$sumw[k2]\times(sumd[i]-sumd[k2])-sumw[k1]\times(sumd[i]-sumd[k1])<0$$

将含 K 的项移到不等式左边，整理得

$$(sumw[k1]\times sumd[k1]-sumw[k2]\times sumd[k2])/(sumw[k1]-sumw[k2])>sumd[i]$$

我们令 $g[k1,k2]=$不等式左边，当 $g[k1,k2]>sumd[i]$ 时 $s[k1,i]-s[k2,i]<0$。

因为 $d[k]\geq 0$，所以对于 $j\leq i$ 有 $g[k1,k2]>sumd[i]\geq sumd[j]$。

证毕。

由上面的证明，可以说明决策变量 j 是单调的，此时问题就好解决了。我们可以维护一个特殊的队列 k，这个队列只能从队尾插入，但是可以从两端删除。这个队列满足 $k1<k2<k3<...<kn$ 以及 $g[k1,k2]<g[k2,k3]<...<g[kn-1,kn]$。

1．计算状态 $f[i]$ 前，若 $g[k1,k2]<sumd[i]$，表示 $s[k1,i]-s[k2,i]>0$，即决策 $k1$ 没有 $k2$ 优，应当删除，直至 $g[k1,k2]>sumd[i]$。

2．计算状态 $f[i]$ 时，有 $f[i]=c[k1]+w[i,k1]+w[n+1,i]$。

3．计算状态 $f[i]$ 后向队列插入新的决策 i。若 $g[kn-1,kn]>g[kn,i]$，表示在 kn 比 $kn-1$ 优之前 i 就将比 kn 优，kn 没必要保存。因此删除 kn，直至 $g[kn-1,kn]<g[kn,i]$。

每个状态 $f[i]$ 计算的复杂度都为 $O(1)$，维护队列 k 的总复杂度为 $O(n)$，因此整个算法的时间复杂度为 $O(n)$。问题得到解决！

本例直接利用决策的单调性，运用这一特殊的队列成功地将算法的复杂度降低，而有的动态规划问题，虽然表面上看起来决策不单调，无法直接套用上面介绍的优化方法，但是只要充分挖掘条件中隐含的单调关系，并对数据作出相应的调整变形，仍然可以利用类似的方法实现时间上的“降维”。

【问题二】货币兑换（BOI2003）

你每天会收到一些货币 A，可能正也可能负，银行允许你在某天将手头所有

的货币 A 兑换成 B。第 i 天的兑换比率是 1 A ： i B，同时你必须再多付出 t B 被银行收取。在 N 天你必须兑换所有持有的货币 A。

任务

找一个方案使第 N 天结束时能得到最多的货币 B

输入文件

第一行是整数 N,T

第二行是 N 个整数 Ai，表示第 i 天开始时得到 Ai 的 A 币

输出文件

将最终得到的最多的 B 货币数写到文件

数据范围

- 1 ≤ N ≤ 34 567
- 0 ≤ T ≤ 34 567

样例输入

7 1

-10 3 -2 4 -6 2 3

样例输出

17

我们很容易得到问题的基本解决方法：

定义 $suma[i]$ 表示第一天到第 i 天总共得到了多少 A 币，特别的，有 $suma[0]=0$

$w[i,j]$ 表示以第 i 天作为一个分割点，并且将第 $j+1$ 天到第 i 天得到的 A 币进行一次性兑换可以得到的 B 币数。则 $w[i,j]=(suma[i]-suma[j])*i$，特别的，当 $i=j$ 时，有 $w[i,j]=a[i]*i$。

求出所有 $suma[i]$ 的时间复杂度为 $O(n)$，此后求任意 $w[i,j]$ 的时间复杂度都为 $O(1)$。

设 $f[i]$ 表示以第 i 天作为一个分割点，最多可以得到多少个 B 币。则 $f[i]=\min_{0<=j\le i-1}\{f[j]+w[i,j]\}-T$。直接用这个式子计算的时间复杂度为 $O(n^2)$，不能满足要求。

由于本题中 $a[i]$ 可正可负，使得 $suma$ 不满足单调性，只要推倒一下就会发

现，我们不能像上一题一样证明出决策的单调性。因此我们好像无法采用上例中相同的方法来实现“降一维”。

但我们很自然地联想到，如果可以通过一定的变换，将 *Ai* 的值变为全部非负或者非正，问题就迎刃而解了。

注意到这样一个条件：**第 i 天的兑换比率是 1 A ：i B，也就是说，时间越往后，A 币越值钱。也就是说，A 币兑换 B 币的比率是单调的！**

利用这一重要条件作为突破口，考虑一种贪心的思想：如果当前手上的 A 币数为正，那么我们可以不忙着将其换成 B 币，因为留到以后再换，不但可以减少 T 的消耗，还可以使得兑换的比例更高。基于这种思想，我们很自然地得到了如下猜想。

[猜想]

若 $\{a_i\}$ 序列中有一段连续的子序列 a[i],a[i+1].....,a[j](i<j)，满足：

a[i]，$a[i]+a[i+1]$，$a[i]+a[i+1]+a[i+2]$，…… $a[i]+a[i+1]+.....+a[j-1]$，

均为非负数，而 $a[i]+a[i+1]+.....a[j]$ 为负数，则当在第 i 天收到了 $a[i]$ 的货币A后，**一定存在**某个最优方案，使得我们不急于在第 i 天兑换，而是继续在第 $i+1$ 天收到 a_{i+1} 的A币，第 $i+2$ 天收到 a_{i+2} 的A币，直到第 j 天收到 a_j 的A币，再看情况决定是否兑换。

这个猜想可以用反证法得到证明，下面给出证明。

证明：

假设某个最优方案在 $[i, j)$ 区间中，存在一个或多个兑换点，设最早的一个兑换点在第 k 天。

那么设从上次兑换点到第 $i-1$ 天为止，共余下了 x 的 A 币，则：

情况一： x 为非负数：那么显然到第 k 天的兑换点我们余下的钱将不会小于 x，取消这个兑换点，将其并入后一个兑换点一起兑换，既提高了手中 A 币的价值，又省去了一次兑换手续费 T。

情况二： x 为负数：将这个兑换点提前至第 $i-1$ 天，提前脱手了负数的 A 币，相应地可以使付出的 B 币减少，而且将区间 $[i,k]$ 内的非负数归入下一个兑换点，得到的收益不会比原来少。

综上两种情况，无论如何我们都可以通过调整将第 k 天的兑换点去掉，收入却没有减少或者更高，因此假设不成立，前面的猜想得到了证明。

这个猜想告诉我们，如果 $a[1]$ 非负，我们不用急着兑换，而是等到第 2 天，

看 $a[1]+a[2]$，如果还是非负，继续考虑第 3 天，第 4 天……，直到出现最小的 i，满足 $a[1]+a[2].....+a[i]<0$（如果找不到这样的 i，则令 $i=n$）。显然一定存在某个最优方案，使得第 1 到第 $i-1$ 天均不是兑换点。因此，我们可以将序列 $a[1],a[2].....a[i]$ 压缩成一个数 $c[1]$。类似地，我们继续从第 $i+1$ 天考虑起，找到一个最小的 j，满足 $a[i+1]+a[i+2].....+a[j]<0$（如果不存在，则令 $j=n$）。同样的道理，第 $i+1$ 到第 $j-1$ 天均不是兑换点，我们可以继续将 $a[i+1],a[i+2].....a[j]$ 压缩成一个数 $c[2]$。依次类推，用同样的方法，可以将整个 a 序列压缩，变成一个长度为 $m(m<=n)$ 的序列 c。c 序列有个很重要的特征：除了最后一个元素可能非负外，其它所有元素为负数。因此，相应的 $sumc$ 除了最后一个元素外，一定满足单调性。只要我们对最后一个元素做特殊处理，就可以完全套用前面的那个问题的方法来解决了。至此，我们成功地将原算法的时间复杂度降了一维！

二、 在非动态规划问题中的应用

对于一些非动态规划的问题，如果能够发现题中隐含的单调关系，同样可以运用一个类似的队列来实现时间上的“降维”。

【问题三】旅行问题（POI2004）

John 打算驾驶一辆汽车周游一个环形公路。公路上总共有 n 车站，每站都有若干升汽油（有的站可能油量为零），每升油可以让汽车行驶一千米。John 必须从某个车站出发，一直按顺时针（或逆时针）方向走遍所有的车站，并回到起点。在一开始的时候，汽车内油量为零，John 每到一个车站就把该站所有的油都带上（起点站亦是如此），行驶过程中不能出现没有油的情况。

任务：

判断以每个车站为起点能否按条件成功周游一周。

输入：

第一行是一个整数 $n(3<=n<=1000000)$，表示环形公路上的车站数。

接下来 n 行，每行两个整数。第 $i+1$ 行含有：$p_i(0<=p_i<=2000000000)$，表示第 i 号车站的存油量；$d_i(0<d_i<=2000000000)$，表示第 i 号车站到下一站的

距离。

输出：

输出共 n 行，如果从第 i 号车站出发，一直按顺时针（或逆时针）方向行驶，能够成功周游一圈，则在第 i 行输出 TAK，否则输出 NIE。

样例输入

```
5
3 1
1 2
5 2
0 1
5 4
```

样例输出

```
TAK
NIE
TAK
NIE
TAK
```

我们探讨一下本题的解决方法。

算法一：

最容易想到的方法是枚举所有车站作为起点，并且模拟汽车的行驶过程，看能否周游一圈，该方法的时间复杂度为 $O(n^2)$，就本题的规模而言，完全无法承受，优化势在必行！

算法二：

简单的枚举法之所以低效，最根本的原因就在于，我们在依次判断各个车站作为起点能否可行的时候，没有充分利用前面得到的某些结果。

由于顺时针和逆时针行走，在本质上是一样的，因此我们暂且只考虑顺时针行走的情况。

原题中带有两个参量 p 和 d，为了减少思考的复杂度，不妨将其压缩成一个。令 $a[i]=p[i]-d[i]$，从第 i 站走到下一站，车内的汽油增加（或减少）了多少。

此外，为了方便讨论，我们将环拆开来，得到一个长度为 $2n-1$ 的链

$$a[1],a[2].....a[n],a[n+1]....a[2n-1]$$

并满足：$a[n+1]=a[1],a[n+2]=a[2].....a[2n-1]=a[n-1]$

设 $suma[i]$ 表示 a 序列中前 i 项的总和，即 $suma[i]=\sum_{j=1}^{i}a[j]$，特别的，有 $suma[0]=0$

此时，我们可以清楚地看到，要判断从某个车站 i 出发，能否成功地周游一圈，实际上就是看 $suma[i]-suma[i-1]$，$suma[i+1]-suma[i-1]$，…… $suma[i+n-1]-suma[i-1]$ 这 n 个数中是否存在负数，如果有负数，则表示中途一定会出现没有油的情况，否则可以顺利行驶一周。

如果我们还是像最朴素的方法一样，通过循环来查找这 n 个数中是否存在负数，算法复杂度没有得到任何优化。但是我们注意到，如果 n 个数中的最小数非负，那么这 n 个数全部非负。求 n 个数中的最小数，很自然的让我们联想到了用堆来处理。用一个大小为 n 的堆保存 $suma[i]$，$suma[i+1]\cdots\cdots suma[i+n-1]$，判断 $suma[i]-suma[i-1]$，$suma[i+1]-suma[i-1]$，…… $suma[i+n-1]-suma[i-1]$ 中是否存在负数，实际上就是判断当前堆中的最小数是否小于 $suma[i-1]$，可以用 $O(1)$ 的复杂度实现。在枚举起始站的时候，随着 i 的增加，我们相应地要将 $suma[i]$ 从堆中删除，并将 $suma[i+n]$ 插入堆中，删除插入的时间复杂度均为 $O(\log_2 n)$，因此整个算法的时间复杂度为 $O(n\log_2 n)$。

这样，我们通过充分利用已经得到的结果，得到了一个比刚刚好得多的算法。但是本题的数据范围极大，$O(n\log_2 n)$ 的时间复杂度仍然不是特别理想，我们期望构造出更好的算法。

算法三：

受到算法二的启发，我们注意到，本题还可以是这样理解：

给定一个数列 $suma[1],suma[2],.....,suma[2n-1]$，以及 n 个区间 $[L_1,R_1],[L_2,R_2].....[L_{2n-1},R_{2n-1}]$，每个区间 $[L_i,R_i]$ 满足 $1<=Li<=n,Li+n-1=Ri$。

要求你对于每个区间，给出 $suma[Li]...suma[Ri]$ 中最小值的编号（如果存在多个这样的编号，任意输出一个即可）。

很明显，这个问题可以套用标准的的 RMQ 算法得到解决。这样算法的时间复

杂度就成功的降为了$O(n)$。问题到此似乎得到了完美解决，但是标准 RMQ 算法的编程复杂度很高，我们很自然地会期望得到一个更容易实现的高效算法。

注意一下这n个区间，发现它们的左右端点都是**严格递增**的！因此本题和一般的 RMQ 问题又存在着些许不同之处，或许利用区间的的单调性，可以像问题二一样得到一个高效且容易实现的算法。

算法四：

受到算法三的启发，我们尝试像例一一样，猜想n个最小值编号是满足单调性的。

其实只要再回过头去分析一下算法二的执行过程，注意到，如果某一次删去的$suma[i]$和插入的$suma[i+n]$都很大，对堆中的最小数是不够成影响的。顺着这个思路往下想，我们很惊喜地发现如下规律：

如果在某一段$suma[i],suma[i+1].....suma[i+n-1]$中，存在$j<k$，使得$suma[j]>suma[k]$，那么$suma[j]$在以后的枚举中无论如何都不可能成为堆中最小的元素（因为$suma[j]$肯定要比$suma[k]$先删除出堆，而$suma[j]$又比$suma[k]$大），直到它被删除出堆。

这样刚刚的猜想得到了证明！其实真是因为保存过多对求解无意义的数据，直接导致了算法二的低效。

至此，我们发现此时的问题和问题一、二有着很大的相似性，因此我们完全可以采用类似的方法解决：

1．维护一个特殊的队列k，表示$suma[k1],suma[k2].....suma[kn]$有可能在今后的枚举中成为某一段的最小值。这个队列只能从队尾插入，但可以从两端删除，并且要满足$k1<k2<k3.....kn$。

2．判断以i作为起点站前，若$k1=i-1$，直接将$k1$从队列中删除，接下来判断，若$suma[kn]>=suma[i+n-1]$，则表示$suma[kn]$在今后不可能成为某一段的最小值，是多余的，将其从队列中删除，直到队列为空或$suma[kn]<suma[i+n-1]$，最后将$i+n-1$插入队列。

3．若$suma[k1]>=suma[i-1]$，则表示以i作为起点的这一段中不会出现负数，标记从i出发，可以成功行驶一周。

判断每个i作为出发点可行的时间复杂度为$O(1)$，维护队列的时间复杂度为

$O(N)$，因此整个算法的时间复杂度为$O(N)$。这个算法的时空效率在理论上和算法三差不多，不过实现难度要比算法三简单得多，甚至比用堆优化的算法二还要容易！

至此，本题得到了完美的解决！

下面是对四个算法的比较：

	空间复杂度	时间复杂度	实现难度
算法一	O(N)	O(N^2)	很容易
算法二	O(N)	O(Nlog2N)	简单
算法三	O(N)	O(N)	难
算法四	O(N)	O(N)	简单

三、总结：

本文的三个例子，都是通过充分挖掘问题当中隐含的单调关系，运用特殊的队列，成功地实现的算法的优化。而且由于采用的数据结构简单，优化过后的编程难度在最初算法的基础上几乎没有增加多少。由此可见，有时候要构造出一个优秀的算法，不一定非得运用多么高深的数据结构。通过不断挖掘问题的隐含条件，也许较简单的数据结构就能够胜任算法的优化！

当然，在竞赛当中，通过寻找隐含数据关系，灵活运用数据结构的问题远不止文中提到的这些类型，有的问题思考、处理起来都要难得多。但是只要我们善于发现，勇于创新，就能够构造出更加完美的算法。

【感谢】

衷心感谢曹利国老师在写这篇论文时给我的指导和帮助。

衷心感谢周戈林同学给我的帮助。

【参考文献】

《算法艺术与信息学竞赛》——刘汝佳、黄亮，清华大学出版社

2005 年信息学国家集训队队员周源的论文