数据读取问题

陈 可卿
浙江省 绍兴一中

摘要

本文给出了一个原创题，讨论并研究了题目的部分分解法，以及一个优秀的标准算法。对题目的数据生成，及作者对分数分布的预测进行了详细分析，并做了总结。

关键字

BFS DFS 并查集 线段树 平行世界 离散化 BFS序列

目录

章节1

问题描述

1.1 题目描述

平行宇宙，或者叫做多重宇宙论，这是一种在物理学里尚未被证实的理论。

我们假想，在某个事件点(以时间为轴)之后，宇宙的运行轨迹会出现许多可能，而这些可能的宇宙是平行的。举例来说，从我们现在存在的这个宇宙开始，每过一个时刻，宇宙就会分成很多个，在这些宇宙中，会有成为警察的你，会有成为总理的你，等等。而这些不同的宇宙是相互平行的，且在之后的发展中也是平行，不会相交。

现在我们进入正题。

我们正在使用计算机读数据，数据有K行，每行一个非负整数。我们需要按如下方式读取数据:

1、首先读入第一个数，需要支付1的代价。

2、我们假设读入的数是x，那么我们需要读入接下来x个数。

3、如果文件已经读完，则读入结束；否则我们接着再读一个数(需要支付1的代价)，然后转2。

数据保证任何一个读入的x，在他后面至少还有x个数字。虽然按照上面的方法一定可以恰好读完数据，但是这么做支付的代价不一定是最小的，你可以修改读入的那个x，可以把x修改为$x+y$或者$x-y$，不过必须保

证，值仍然是非负整数，且接下来有不少于$x \pm y$个数。而我们需要支付的代价就是y。

相信你已经猜到我们的问题了，那就是恰好读完所有数据，需要支付的最少代价是多少。

等等，似乎还缺了什么。没错，我们并不知道这些数据是什么，但我们知道这些数据可能是什么，就像薛定谔的猫。在没有读入这个数字之前，它什么都是，一旦读入了这个数，根据结果，我们就进入了不同的平行世界。

现在我将告诉你宇宙可能出现的轨迹，希望你计算出所有不同的结果(读完数据需要支付的最小代价)。

1.2 输入格式

为了方便起见，我们把读入一个数看成一个事件，我们用$1 \ldots N$把所有可能的事件编号。

第一行，读入一个整数N，表示可能的事件的个数。

接下来N行，第$i+1$行描述第i个事件。第i个事件，将会告知这些参数$x\ m\ a_1\ a_2\ a_3\ \cdots$：

x 表示第i个事件的数据值是多少。

m 表示这个事件之后有多少种可能出现的事件，编号是$a_1\ a_2\ a_3\ \cdots$。

其中第1个事件，它的数据值一定是-1，因为这时第一个数还未读入。也就是说，第1个事件所相关的事件$a_1\ a_2\ a_3\ \cdots$，就是读入的第一个数所有的可能值。且如果某个事件$m=0$，那么这个所读入的数x，就是数据中的最后一个数。

1.3 输出格式

输出M行，M是最终所有可能的不同宇宙的个数。

每行输出一个最小的代价值，按照代价值的大小，从小到大输出。

1.4 输入样例

```
7
-1 1 2
1 2 3 7
3 1 4
0 1 5
0 1 6
0 0
0 0
```

1.5 输出样例

```
1
3
```

1.6 注意事项

数据保证所有可能出现的事件序列，都满足按任意读入的x，之后至少还有x个数字。

提交文件名为`rdata.*`，输入输出文件名为`rdata.in/out`。

每个测试点的时间限制都为4秒[1]，空间限制以机器为准。

1.7 数据规模

数据保证有20%的数据$N \leq 2000$；

保证有60%的数据$N \leq 200000$;

另有20%的数据，输出的个数$M \leq 50$。

所有数据$N \leq 1000000$。

[1]测试环境为Inter® Core(TM)2 Duo CPU T8300 @ 2.40GHz 2GB RAM

章节2

出题背景

2.1 题目灵感

由于今年集训队作业从论文改成了出题，所以就有这个题目。`:-p`

题目的灵感来自PYC同学和我说的某个题，以及CEOI2009的某个题目。现在成形的题目有着精妙的模型转化，以及解题技巧。

2.2 题目背景

这个着实让我头痛了一把，最终修订版经过数人的检验，基本已无歧义等问题。

章节3

问题分析

3.1 模型简化

题目给出一颗N个点的树，从树的根节点到树的每个叶子节点所形成的每条链，都按如下做法：

花费1的代价读一个数x，然后修改为$x \pm y$，修改代价为y；接着读$x \pm y$个数；一直执行这个过程，直至恰好读完。

分别求出最小代价，最后排序输出。

3.2 部分分思路

3.2.1 一条链的做法

我们先来考虑数据中的特殊情况(同样也是题目的子问题)，也就是一条链的情况。

有一个长度为n的串，元素的值是$a_1\ a_2\ \cdots\ a_n$。

我们按复杂度的高低来简述一下各类算法。

$O(n^2)$

首先我们来添加一个点，也就是a_{n+1}，目的是为了方便，当我们达到

第$n+1$个点的时候，也就是我们读完数据的时候。

我们用$f[i]$来表示，到达第i个点(还未读取数据)的最小代价值。很显然$f[1]=0$，然后有如下转移：

$$f[i] = \min_{j<i}\{f[j]+1+|j+a_j+1-i|\}$$

其中加1，表示读取这个值的代价，那么最后$f[n+1]$就是我们要求的值。

$O(n\log n)$

同理先添加a_{n+1}。仍然用$f[i]$来表示，到达第i个点(还未读取数据)的最小代价值。观察一下刚才那个方程：

$$f[i] = \min_{j<i}\{f[j]+1+{\color{red}|j+a_j+1-i|}\}$$

我们可以对绝对值分类讨论一下：

$$f[i] = \min_{j\ ,\forall j+a_j+1\geq i}\{f[j]+1+(j+a_j+1)\}-i$$

$$f[i] = \min_{j\ ,\forall j+a_j+1<i}\{f[j]+1-(j+a_j+1)\}+i$$

如上所示，我们只要按照$j+a_j+1$的值来建立一棵线段树[2][4](其他数据结构也可以)，来维护$f[j]+1+(j+a_j+1)$和$f[j]+1-(j+a_j+1)$的最小值。然后对于$f[i]$，我们只要求出$j+a_j+1<i$和大于等于$j+a_j+1\geq i$的最小值就可以了。

包括修改，以及询问操作在内，所有单次操作的复杂度是$O(\log n)$，而总共进行的操作次数是$O(n)$，所以总的复杂度是$O(n\log n)$。

$O(n)$

上述算法都是在动态规划的思想下进行的，现在让我们换个思路。

还是那个序列$a_1\ a_2\ \cdots\ a_n$，读入1个数a_i，然后我们可以直接转跳到第$i+a_i+1$个数，但我们还可以选择到第$(i+a_i+1)\pm y$个数。换个思路，我们不是选择直接从i转跳到$(i+a_i+1)\pm y$，而是先到达$i+a_i+1$，再由$i+a_i+1$到达$i+a_i+1\pm y$。

让我们更直观的描述一下，我们建立一个有向带权图。点i向点$i+a_i+1$连权值为1的边，表示i可以花费1的代价转跳到$i+a_i+1$。然后点i向点$i\pm 1$都连权值为1的边，表示某个可以转跳到i的点x，可以选择转跳到$i\pm 1$，而代价就是1，显然这里$i\neq 1$。

建完图之后，我们就要求最短路了。上述图的信息，以1为源点，到达每个点的最短路就是前两个算法中所提到的$f[i]$的值。再仔细观察，我们发现每条边的权值都是1，所以我们可以用宽度优先搜索(BFS)来代替最短路。最后我们求出到达第$n+1$个点的最短路的值，就是我们所求的最小值。

由于每个点连出去的边不超过3条，而BFS的复杂度是$O(E+V)$，所以这个算法的复杂度是$O(n)$。

3.2.2 一棵树的做法

看完了一条链的三种做法，相信你已经对一棵树的做法初有想法了。

$O(N^3)$，$O(N^2\log N)$，$O(N^2)$

将树中所有的链条取出，然后用一条链的方法来分别求出答案。

$O(N^2)$

树是由多条链合并而成的，但是就算是不同的两条链，它们也有着公共部分，对于它们公共部分，我们可以一起计算。

我们在每个叶子节点后面添加一个空点(理由同上)，然后用$1\ldots N$来给所有的点标号。我们用$f[i]$来表示，到达节点i(还未读取数据)的最小代价值。有如下转移：

$$f[i]=\min_{j}\{f[j]+1+|a_j+1-dist(i,j)|\}$$

其中j是i的每个祖先，$dist(i,j)$是两个点之间的距离，而最后答案，显然就是我们添加的那些点的$f[]$值。由于每个点最多有$O(N)$个祖先，所以这个算法的复杂度是$O(N^2)$。

$O(N\log N)$

利用公共部分来加速算法，同样可以在线段树的方法中使用。

试想，我们用深度优先搜索(BFS)来遍历整棵树，当我们遍历到节点i的时候，还存在在栈中的节点就是i的所有祖先。然后我们观察一下一条链的第二个算法，当我们试图求出第j个节点的$f[j]$值的时候，我们需要用到的信息就是就只有j之前的点(在树中就是所有祖先)。所以我们仍然用$f[i]$来表示，到达第i个点(还未读取数据)的最小代价值。

$$f[i]=\min_{j<i}\{f[j]+1+|j+a_j+1-i|\}$$

然后按$j+a_j+1$的值来建立一棵线段树，维护两个最小值。当我们遍历到节点i时，通过线段树中所存储的信息，得到了$f[i]$的值，然后将$f[i]$信息添加入线段树中；在i退出栈的时候，我们将这个信息从线段树中删除。如此，我们得到了一个复杂度为$O(N\log N)$的算法。

值得注意的是，由于$j+a_j+1$的值会重复，而现在又涉及到了删除操作，所以我们要将$j+a_j+1$值先离散化，以便进行线段树操作。

3.2.3 其他做法

上面提及的部分分做法，都是按照数据规模，或者数据特殊性而设计的。现在让我们来看看其他的做法。

骗分

我并不喜欢骗分，不过这里还是提一下。

由于数据保证了“任何一个读入的x，在他后面至少还有x个数字”，所以我们可以直接做，不考虑修改x的值。

合并

还是这句话，我们发现如果某事件x之后可能出现两个事件y,z且$y=z$，那么我们可以合并这两个事件。这可以缩小题目规模，使得一些本来会超时的算法通过测试数据。

3.3 优秀算法

接下来介绍一个复杂度十分优秀的算法，也是标程使用的算法。

3.3.1 算法构思

回想一下一条链做法的最后一个算法，这是一个基于BFS的最短路算法。我们来考虑，是否可以将这个算法运用到一棵树上。

连边方式依然和上面说的一样：点i向它的父亲和儿子们都连权值为1的边，表示某个可以转跳到i的点x，可以选择转跳到$i\pm 1$；点i向它子树中深度为$deep(i)+a_i+1$的点连权值为1的边，表示i可以直接到这些点。

可是问题出现了，虽然连边不是问题，但是点i子树中深度为$deep(i)+a_i+1$的点的个数可以到达$O(N)$，也就是说我们连的边数可以到达$O(N^2)$。这迫使我们寻求算法优化的途径。

3.3.2 算法优化

所连的两类边中，第一类我们可以确定它们的数量是$O(N)$。所以我们要优化的也就只有第二类边。

在叙述优化算法之前，我们先引入一个东西，这个就是树的BFS序。树的BFS序，也就是在BFS遍历整棵树的时候，存于队列中节点编号的序列。为什么要说这个东西？因为这个BFS序，有一些特殊的性质，它会把

同一层(深度相同)的节点并在一起，且会把同一个子树中的同一层节点也并在一起。

现在我们要连的第二类边是从点i，连向其子树中所有深度为$deep(i)+a_i+1$的点，也就是说他连向的节点在BFS序中是连续的一段。

有了这个强力武器之后，我们就可以优化我们的算法了。算法流程如下：

```
将所有可能成为第一个元素的点都加入Queue;
while Queue非空 do
    取出Queue的首元素i;
    if i_father未被访问过 then
        将i_father加入Queue;
    end
    for j = i每个儿子 do
        if j未被访问过 then
            将j加入Queue;
        end
    end
    for j = i可以跳到的区间中还未被访问过的元素 do
        if j未被访问过 then
            将j加入Queue;
        end
    end
end
```

Algorithm 1: 最终算法

在算法流程中我们看到，我们需要访问i可以跳到的区间中还未被访问过的元素。如果我们可以用$O(\beta)$的复杂度，做到只找到并访问，未被访问过的元素，那么由于每个点最多只被访问到1次，所以第二类边的复杂度就可以降低到$O(N\beta)$。

如何做到只访问我们需要的元素呢？我们可以用并查集[1]来实现：

我们以每个点的BFS序来作为集合标号，以开始每个元素都指向自己。如果一个元素x已经被访问过了，那么我们就把它指向$x-1$，这样我们就只要按如下方法就可以遍历我们需要的点了：

需要遍历的区间从$Left$到$Right$;
for $i = FindSet(Right)$; $i \geq Left$; $i = FindSet(i)$ **do**
    将i加入$Queue$;
    删除i，把i指向$i-1$;
end

Algorithm 2: 遍历-并查集

其中$FindSet(x)$是找x集元的一个函数。

如此我们就将所有操作的总复杂度控制在了近似$O(N)$这个级别。

3.3.3 实现技巧

整个算法流程在上面已经叙述清楚了，这里主要介绍一下实现技巧。

我们需要求出每个点i，可以到达的区间$Left$和$Right$。我们可以用如下实现：

Input: u
$DeepCount[deep(u)] = DeepCount[deep(u)] + 1$;
$Left_u = DeepCount[deep(u) + a_u + 1] + 1$;
for $i = u$的每个儿子 **do**
    $DFS(i)$;
end
$Right_u = DeepCount[deep(u) + a_u + 1]$;

Algorithm 3: 求可行区间

主要思想是利用DFS的性质，具体需要模拟堆栈。

这样我们就确定了i所到达的区间，在深度为$deep(i)+a_i+1$的所有节点中第$Left_i$到第$Right_i$个被访问到的点。

3.3.4 算法复杂度

我们使用BFS算法，一类边数量$O(N)$，二类边数量$O(N)$，访问复杂度均摊为$O(\alpha(N))$[1] [1]。这样总体复杂度为$O(N\alpha(N))$。

空间复杂度为$O(N)$。

[1] $\alpha(x)$为Ackerman函数的某个反函数，它可以近似看成小于5的常数[3]

章节4

数据设计

4.1 测试点1

人工手造小数据，$N = 10$。

4.2 测试点2

普通的随机生成的树，在连边的时候保证点i只可能是$i-1$或者$i-2$的儿子,$N = 2000$。

4.3 测试点3

满二叉树，$N = 65535$。这个点使用$O(N^2)$的算法可以通过，因为每个点的祖先的个数不超过$O(\log N)$，所以实际复杂度为$O(N \log N)$。

4.4 测试点4

扫把型的树，也就是1到$\lfloor N/2 \rfloor$是一条链，而从$\lfloor N/2 \rfloor$开始，后面是一棵满二叉树，$N = 131069$。

这个测试点是希望使用$O(N \log N)$算法的程序可以通过。

4.5 测试点5

双头扫把型的树，形状与上面那个类似，不过这次是有两个扫把头，$N = 200000$。

4.6 测试点6

这是一条链的数据，$N = 200000$。是希望使用链型$O(N \log N)$算法的程序可以通过。

4.7 测试点7

链型数据，不过最后有一个小分叉，范围略大$N = 1000000$，这是希望使用链型$O(N)$，也就是BFS算法的程序通过。

4.8 测试点8

扫把型的树，不过范围略大$N = 1000000$，只希望标准算法通过。

4.9 测试点9

双头扫把型的树，$N = 1000000$。

4.10 测试点10

扫把型的树，不过最后的扫把是一棵满三叉树，$N = 1000000$。

章节5

选手分布估计

5.1 算法得分分析

$O(N^2)$

虽然在数据规模的说明中，只给出了20%的数据，但是这个算法实际可以得到30%的分数。

$O(N \log N)$

按照数据规模中的说明，这个算法可以得到60%的分数，实际也是如此。

骗分

可以得到10%的分数。不过，不排除存在可以得到更高分数的骗分方法。

链条$O(N)$

求出所有链后，用$O(N)$的方法来做。这个可以得到20%的分数。

各类结合

按照出题人的设想，在不使用标准算法的情况下，最多可以得到70～80%的分数。

标准算法

可以得到100%的分数。(因为时限相对宽限，不存在因实现方式而导致的超时问题)

5.2 选手得分分析

对于大部分的选手，根据出题人的设想，可以想出$O(N^2)$的方法。然后部分选手可以想到一条链情况$O(N\log N)$的算法，而其中小部分的选手可以准确无误的写出树形的$O(N\log N)$的算法。

如果选手可以想出一条链的$O(N)$的BFS算法，那么至少可以得到50%的分数。而可以将这个BFS算法升级到树形的$O(N\alpha(N))$算法，且正确实现，相信只有极少数人。

5.3 分布估计

得分	估计得分人数(百分比)
0	2%
10	5%
20	5%
30	35%
40	5%
50	20%
60	5%
70	10%
80	6%
90	5%
100	2%

章节6

总结

这次出题让我收获了不少，这是一种宝贵的经验。

题目总的难点在于思维复杂度高。标程所涉及到的算法都是很简单的，BFS、DFS、并查集...，这些都是我们常用的基本算法，所以难点并不在此；而题目所涉及到的模型转化，一些基础算法的特殊应用，才是题目要考察选手的地方。在题目分析中，所提及的其他一些方法，也都很有启发性。

参考文献

[1] Robert E. Tarjan and Jan van Leeuwen. *Worst-case analysis of set union algorithms*. Journal of the ACM, 31(2):245 – 281, 1984.

[2] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*: An Introduction. Springer-Verlag, 1985

[3] 刘汝佳, 黄亮. 《算法艺术与信息学竞赛》. 清华大学出版社, 2004

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7