数据结构第二次作业解题报告

梁俊邦

建议每题在自己写了个超时（但是剩下的要对）的程序之后，想了1h以上都想不出正解解法再来看题解。这样才会有比较大的提高。

Stock

【题目大意】

给出N条互不包含的在第一象限且与x轴平行的线段。令f(x)为穿过x的线段中y的最大值。求x从0到所有线段末端最大值的所有f(x)的平均值。N不超过10^7。

【关键字】

单调队列

【解】

画图可以容易看到题目相当于要求线段与x轴“包围”成的图形面积。我们可以让x从0开始扫一遍（就像读入的一样），每次遇到一个事件点（线段插入或删除），统计前一段的结果，接着处理一下新事件即可。

如何统计前一段结果？我们需要知道这个时刻，所有存在着的线段的最大高度。我们可以发现，考虑分别线段i和线段j，i比j先插入，如果有height[i]<height[j]，那么在插入了j之后，由于它比j先删除，所以永远不可能是最大值，也就是说它已经没用了。这样我们就可以维护一个插入时间递增，高度递减的保存着“有用”的线段的双端队列。这个时刻最大的高度就是队列头的高度。如果需要插入一个高度为h的线段，那么则把队列里面的所有高度不比h大的线段删除（从队列尾开始一个个删）再在队尾插入该线段；如果有某一条线段被删除，则检查队列头的线段是否需要被删。是的话删除队列头即可。

因为每条线段插入一次，最多被删除一次，所以时间复杂度为O(N)。

【细节】

需要用gets(char\*)来读入否则会超时。第一次写单调队列的话注意一下插入与删除两种操作。不要做过多的判断，比如删除时从队列尾开始删而不要从头开始for一遍，删除队列头的时候不要把整个队列都移动一格，只需要++head就好。

Zerone

【题目大意】

有N个01串。每次两个玩家轮流操作。第i次操作可以去掉所有第i位为0或1的字符串。谁操作之后没有串剩余则判负。两个人都采取最佳操作。对于从1到n的i，求前i个串作为初始状态时的胜负情况。N不超过10^6，串的长度不超过64。读入数据不超过35MB。

【关键字】

字母树

【解】

一个大的思路是，每次加入一个串，然后快速判断是否为先手必胜。

那么对于N个01串，我们考虑一棵由它们形成的二叉树。边的权值为0或1，从根开始往下走到叶子节点即为其中一个01串。那么对于每次的操作p，在树中就是对应于沿着1-p的那条边往下走。初始状态则对应根节点，操作之后没有剩余的串即对应于在树中由于已经到了叶子节点而没得走。那么原问题即转化为一棵二叉树，从根节点开始轮流往下走，谁到了叶子节点就输，问是否先手必胜的一个问题。解法很显然：

win[node] = !win[son1] || !win[son2]

win[leaf] = false

稍微解释一下第一个转移方程：一个节点是否为先手必胜，等价于是否有一个儿子节点是先手必败。

对于加入串的操作，我们很容易通过在树中走相应的路径实现。注意到除了那条路径之外，其他节点的胜负状态是不变的，所以我们只需要修改路径上的win值即可。

时间复杂度为O(NL)，L为每个串的最大长度。

【细节】

空间复杂度为O(NL)，需要精细地估计空间的使用。其实也有一个不需要这么多空间的做法：注意到在树中儿子数量为2的节点只有O(N)个，我们可以把儿子数为1的节点合并起来从而压缩字母树。此时空间复杂度降为O(N)。不过实现起来比较麻烦。

Behtree

【题目大意】

有一段XML格式的文档。其中的语法已给出。要求转成规定的JSON格式的文档。

【关键字】

模拟

【解】

没什么可说的……读懂题之后直接字符串处理即可。这里介绍一下一个比较简便的做法。注意到输入里面每对双引号都对应着一个node，而每一个斜杠都对应着最近的node的结束。我们可以不断读入，直到遇到双引号或者斜杠为止，然后作相应的处理。推荐用递归的方法边读入边输出，这样写很方便。

【细节】

Leaf\_node和Inner\_node的输出有点不一样。这个需要注意一下。还有就是需要注意一下输出格式，不要粗心打错了符号。

Mooney

【题目大意】

一个有向图。每个节点有0或1两个权值。求从0号节点到n-1号节点的：1、最少经过多少个0；2、最多经过多少个不在同一个点的1。

【关键字】

最短路、强连通分量、拓扑序的DP

【解】

这是两个独立的问题。

对于第一个问题，直接套用最短路算法即可。

对于第二个问题，定义强连通分量为一个子图，里面所有点都可以到达其他各点。求出所有极大的强连通分量并把它们缩成一个点之后，我们可以用如下DP：

f[v] = w[v] + max{f[u] | (u, v) in E}

f[u(0)] = w[u(0)]

ans = f[u(n-1)]

其中w[v]表示v这个强连通分量的权值和，u(i)表示i所在的强连通分量。E表示边集。

下面讲强连通分量的求法。有很多种求强连通分量的方法。在这里只介绍一种：tarjan算法。

这里是郭家宝同学在很多年前的介绍强连通分量的一个blog：<https://www.byvoid.com/blog/scc-tarjan/>

在这里简要说明一下。它是对所有未访问过的点进行深搜，对每个点p记录它的dfn值与low值。初始它们都为“已访问点的数量”。并维护一个栈。然后考虑u连出去的点v。如果它在栈中，则用dfn[v]更新low[u]，否则它如果没有被访问过，则递归深搜v，之后用low[v]更新low[u]，否则它是被访问过但是不在栈中，那么它必定和u不在同一个强连通分量。“更新”指的是取最小值。如果搜到最后有dfn[u]=low[u]，那么栈中u及之后的点都是与u在同一个强连通分量中，需要把它们退栈。

Tarjan算法的正确性可以感性理解一下，笔者也不能清楚地证明。有兴趣的同学可以在上述网址学习一下并查找有关文献。

Tarjan算法及DP的时间复杂度均为O(N)。

【细节】

求完强连通分量之后需要重构图。DP用记忆化搜索的形式写比较简便。本题用到的变量个数比较多，注意不要弄混。

Dice

【题目大意】

一个骰子在8\*8的棋盘上。每个面有一个数。求从指定格子到指定格子的一条骰子的滚动路径，使得路径中骰子面朝下所对应的数的总和最小。

【关键字】

最短路

【解】

用最短路算法即可。由于使用的算法不同，时间复杂度可能不同，所以不在此说明。

【细节】

本题主要就是细节麻烦。首先有效的状态数量有8\*8\*24个。分别是位置以及骰子的状态。如果手算24个状态之间如何滚动转移的话，没有一个三阶魔方和足够大的耐性和足够细的心是不可能完成的。因此我们可以把状态数量扩大到8\*8\*6^6个，虽然实际上没有这么多的状态但是这样的话我们可以比较方便地写出每个状态之间是如何转移的。不过这样写起来也非常麻烦，一定要有耐心。一旦哪里写错了就很难再看得出来了。因为这题的状态数非常少，我们不必优化程序的时间效率，只要写对就可以了。