NOI练习赛 解题报告
fefb6944cc60c32f1ea5833326e27d61

1 捡钱

1.1 10 分算法

Type[i]≤3，A[i]≤1000000。

开辟一数组E[i]记录价值为i 的钱是否在集合中，开辟一变量sum 记录集合元素总和。

操作1：修改E[A[i]]为1，同时令sum 增加i；

操作2：修改E[A[i]]为0，如果原来E[A[i]]是1，那么sum 要减少i；

操作3：直接输出sum。

时间复杂度O(N+A[i].max)。

1.2 20 分算法

没有了A[i]≤1000000 的限制。

预先读入所有可能出现的A[i]离线处理，将它们升序排序，W[i]表示第i 种钱的价值，而E[i]的意义变为第i 种价值的钱是否在集合中。

根据种类id 可以根据W[i]数组迅速查到其价值，根据价值也可以用哈希或二分查找找到其id。

将上述三种操作稍加改动即可。

时间复杂度O(NlogN)。（log 出在二分查找上）

特殊地，也可以直接开一个长为10 亿的bool(ean)数组，不会爆空间。

1.3 40 分算法

多了个询问最小值。

最暴力的方法就是从小到大枚举，时间复杂度O(N2)。

N=500000，看起来会有压力，实际没有：因为操作4 只占总操作的四分之一，并且操作2 的删除基本是什么都删不掉的：因为随机生成，要删除的数本来就存在的概率很小。事实上，可以证明，如果保证数据随机生成，这种算法的时间复杂度的期望是O(NlogN)。这样就可以轻松过掉前4 个点。

1.4 60 分算法

当操作种类加多，第一个想到的当然还是模拟。

操作5：二分查找到A[i]和B[i]对应的种类id，然后for 循环将区间刷成0顺便维护sum；

操作6：二分查找到A[i]和B[i]对应的种类id，然后降序for 循环遇到的第一个1 改成0 维护sum；

操作7：二分查找到A[i]和B[i]对应的种类id，然后for 循环遇到的第一个1 对应的价值输出；

操作8：二分查找到A[i]和B[i]对应的种类id，然后降序for 循环统计名次，并对于名次在[M[i], N[i]]的元素统计求和。

时间复杂度O(N2)。

N=100000，有压力吗？没有！

因为随机生成，会有一半的A[i]¿B[i]的现象，或者M[i]¿N[i]的现象，这些都可以直接跳过。并且操作1 占了一半，而只有操作5 和8 的单次期望复杂度是O(N)，其余的都是O(logN)，所以O(N2)的常数很小。

1.5 80 分算法

对E[i]维护一棵线段树，记录sum，min 和max。

操作1：令E[id(A[i])]增加1；

操作2：令E[id(A[i])]减少1；

操作3：询问E[1..max_id].sum；

操作4：询问E[1..max_id].min；

操作5：修改区间E[id(A[i])..id(B[i])]为0；

操作6：修改E[id(E[id(A[i])..id(B[i])].max)]为0；

操作7：询问E[id(A[i])..id(B[i])].min；

操作8: ……（很麻烦的样子）

时间复杂度O(NlogN)。

另：如果操作8 采用暴力解决，也可以得到100 分。

1.6 100 分算法

对E[i]维护一棵线段树，记录sum 和rank；

其中E[l..r].rank 表示从l 到r 共有多少个元素。则E[1..r]表示不超过r 共有多少个元素。

显然可以在O(log2N)时间询问E 的第i 小元素，记作E.nth(i)；稍作优化（记录中间结果）便可优化到O(logN)。

操作1：令E[id(A[i])]增加1；

操作2：令E[id(A[i])]减少1；

操作3：询问E[1..max_id].sum；

操作4：如果E[1..max_id].rank 为0 则输出0，否则输出W[E.nth(1)]；

操作5：修改区间E[id(A[i])..id(B[i])]为0；

操作6：如果E[id(A[i])..id(B[i])].rank 为0 则什么都不做，否则令E[E.nth(E[1..id(B[i])].rank)]减少1；

操作7：如果E[id(A[i])..id(B[i])].rank 为0 则输出0，否则输出W[E.nth(E[1..id(A[i]-1)].rank+1)]的值；

操作8：咨询A[i]和B[i]所对应的id 和rank 然后特判各种状况（与M[i]，N[i]的大小关系），然后各种分情况乱搞……（具体实在是本弱菜无法用精炼的语言描述得了的……）

一定要特判各种应该输出0 或者不进行任何改动的操作。

时间复杂度O(NlogN)。

2 配对

2.1 算法1

暴力枚举每个数应该向上连还是向下连。

然后，把每个数第一次出现的位置看成左括号，第二次出现的位置看成右括号，那么一个方案是合法的，当且仅当上序列和下序列都是合法的括号序列。一个方案的高度就是两个括号序列的深度之和。

那么暴力检验括号序列是否合法及求出深度就可以了。

时间复杂度：$O(n \cdot 2^n)$。

2.2 算法2

$2n$个数构成了n个区间。称两个区间冲突，当且仅当这两个区间不能放在x轴的同一侧。

考虑两个区间的关系，可能是：没有交集、互相包含、相交（以下均指非包含的相交）。显然，两个区间冲突当且仅当两个区间相交。

把每个区间看成图的顶点，相交的两个区间之间连边。那么原问题有解，当且仅当这个图是二分图。

如果有解，我们再枚举二分图的每个连通块，决定这一块的第一个元素放在x轴上面还是下面（然后连通块的其他元素就确定了）。然后再套用上民的方法检验括号序列的深度。

时间复杂度：$O(n \cdot 2^m)$，其中m是连通块的个数。

2.3 算法3

再考虑上面建出的二分图。

结论：对于两个连通块A和B，如果A中的任何一个区间a被B中的某个区间b包含，那么A中的所有区间都会被b包含。

直观理解是正确的，因为对于A中的任何一个区间x，它一定与a间接相交（x与p相交，p与q相交……最后与a相交），如果x不被b包含，那么这一系列的区间（$x, p, q, \cdots$）一定会有一个与b相交，那么A和B就连通了。

所以不难得出，对于两个连通块A和B，要么A和B的所有区间的并完全没有公共部分，要么有一个被另一个完全包含（即某一个的所有区间都被另一个的某个区间包含）。

为了方便，我们加入一个区间$[0, 2n+1]$，这个区间包含了所有的区间，且单独成一个连通块。这样，所有连通块就按照包含关系，形成了一个以这个连通块为根的有根树。

然后在树上做动态规划。设$dp[A][i]$表示：只考虑连通块A的子树中的区间，要求向上的高度不超过i，这时的向下的高度的最小值。

为了算这个，我们需要以下几个东西：

- A的二分图的每个独立顶点集所对应的区间的最大覆盖高度；（这可以在内部动态规划得出）

- 对于A的每个孩子，要计算对于这个孩子的任何一个区间，A中的包含这个区间的区间个数；

- A的每个孩子的dp值。

转移的时候，首先要枚举A中的每个区间是正放还是反放，然后对于每个孩子枚举这个孩子相对于A是正放还是反放。

算出了所有的dp值以后，枚举在上面放的区间个数算出答案取最小值就可以了。

时间复杂度$O(n^2)$。

3 字符串匹配

3.1 标准算法

考虑$F(n)$的另一种定义：

令$F(0) =$“0”，$F(n|n > 0)$是将$F(n-1)$中所有的“0”替换成“1”，所有的“1”替换成“10”得到的字符串。

容易发现，这样定义的$F(n)$与原定义是完全相同的。

设$f(n,p)$表示$F(n)$中串p的出现次数，那么：

如果p的长度为1，答案是显然的（即某个Fibonacci数）；

否则，如果p中连续出现了子串“00”，那么答案也是显然的（0）；

否则，如果p的最后一位是“0”，那么容易找到字符串q，使得$f(n,p) = f(n-1,q)$，递归即可；

否则，即p的最后一位是“1”，那么容易找到字符串q，使得$f(n,p) = f(n-1, q + “0'') + f(n-1, q + “1'')$，递归即可。

可以证明，这样直接计算$f(n,p)$的时间严格正比于p的长度。