Tree and Sets 解题报告

杭州学军中学 王逸松

1 试题来源

原创

2 试题大意

有一棵n个结点的仙人掌（每条边最多属于一个简单环的无向连通图），每条边有个长度len_i $(1 \le len_i \le 10000)$。

有q个点集，每个点集给定两个整数u, d $(1 \le u \le n, 1 \le d \le 10^9)$，表示所有与$u$的距离不超过$d$的点（包括$u$）在这个点集中。

两个点之间的距离为它们之间的最短路径的长度。

要求构造一个DAG（有向无环图），满足：

1、这个DAG至少有$n+q$个点，至多有1200000(1.2×10^6)个点和2400000(2.4×10^6)条边

2、对于每一条边$u \to v$，$u > n$ 且$u \ne v$

3、对于第i $(1 \le i \le q)$个点集的每一个点x，第$n+i$个点到第x个点都有且仅有一条路径

3 数据范围

测试点编号	n	m	q
1	1000	$m = n - 1$	1000
2	10000	$m = n - 1$	10000
3	10000	$m = n - 1$	10000
4	9000	$m = n - 1$	9000
5	10000	$m = n - 1$	10000
6	1000	$n - 1 \le m \le 2n - 2$	1000
7	10000	$n - 1 \le m \le 2n - 2$	10000
8	10000	$n - 1 \le m \le 2n - 2$	10000
9	10000	$n - 1 \le m \le 2n - 2$	10000
10	10000	$n - 1 \le m \le 2n - 2$	10000

第2个测试点的生成方式：

```
for i in range(2,10001):
    addedge(i,i/2)
```

第3个测试点的生成方式：

```
for i in range(2,5000):
    addedge(i,i-1)
for i in range(5000,10001):
    addedge(i,randint(1,i-1))
```

（其中`range(l,r)`表示区间$[l,r)$中的所有数，`randint(l,r)`返回一个在$[l,r]$内的随机整数）

4 算法介绍

首先我们要搞清楚这道题是在干啥。

有一棵n个点的仙人掌，和q个点集，要求构造一个DAG，对于第i个点集和该点集中的任意的x，第$n+i$个点到第x个点都有唯一一条路径。

从数据结构的角度考虑，就是有一棵仙人掌，每次询问一个点集中的点的元素和，然后要用DAG来表示出求这些元素的和的过程。

4.1 算法一

对每个询问，向对应的点暴力连边。

时间复杂度$O(nq)$，构造出来的图的点数$O(n+q)$，边数$O(nq)$，能通过第1个测试点和第6个测试点。

4.2 算法二

有一个非常良心的测试点，这棵树是一棵以1号点为根的完全二叉树，深度为$O(\log n)$，且每个点的度数不超过3。

我们可以对于每个i，将i子树内的所有点按照离i的距离排序，然后预处理排序后的前缀和。询问时只要从询问点开始不断往上走并将另一侧的子树的贡献统计进去即可。

由于树的深度是$O(\log n)$的，所以总的点数是$O(n\log n+q)$的，边数是$O(n\log n+q\log n)$的，时间复杂度是$O(n\log n+q\log^2 n)$的，可以通过第2个测试点。

4.3 算法三

还有一个非常良心的测试点，这棵树是一条长度为5000的链，和5000个随机父亲的点。

可以证明，这棵树每个点的度数是期望$O(1)$的。

考虑一个经典问题：给一棵边带权的树，每次询问与某个点u的距离不超过d的点的**个数**。

经典的做法是点分治，然后将每个分治的重心的每个子树内的所有点按照离重心的距离排序，然后询问的时候，对于每一层分治，统计重心的所有子树减去询问点所在子树的满足条件的点的个数。

对于这道题，我们统计的信息没有逆元，但是每个点的度数是期望$O(1)$的，所以可以在点分治的每一层暴力统计其他子树的信息。

时间复杂度$O((n+q)\log^2 n)$，构造出来的图的点数$O(n\log n+q)$，边数$O((n+q)\log n)$，可以通过第3个测试点。

4.4 算法四

我们仍然考虑点分治。

对于一个分治重心，它可能有$O(n)$个子树，所以暴力算法是不可行的。

我们可以对这些子树建一棵“线段树”。对于线段树上每个区间$[l,r]$，我们将第l棵到第r棵子树的所有点按照离分治重心的距离排序，然后预处理前缀和。

由于线段树的深度是$O(\log n)$的，所以每棵子树在每一层点分治中被复制了$O(\log n)$次，所以不考虑询问，构造出来的图的点数是$O(n\log^2 n)$的，边数也是$O(n\log^2 n)$的。

对于每个询问，只要在每一层点分治中的线段树中查询，然后找到对应的前缀和即可。每个询问边数是$O(\log^2 n)$的。

要注意实现的技巧，我们可以将树和询问放在一起点分治，然后对某个分治重心的第l到第r棵建线段树的过程，就是令$mid = \lfloor \frac{l+r}{2} \rfloor$，先递归处理$[l,mid]$和$[mid+1,r]$这两个区间，然后考虑左半边子树内的点对右半边子树内的询问的贡献，和右半边子树内的点对左半边子树内的询问的贡献。

时间复杂度$O((n+q)\log^2 n)$，构造出来的图的点数$O(n\log^2 n+q)$，边数$O((n+q)\log^2 n)$。

4.5 算法五

我们可以把点分治的过程看作一棵“虚拟树”。在这棵“虚拟树”上，每个分治重心的儿子是这次分治时的每一棵子树的重心。

于是询问的本质就是从虚拟树的某个结点开始，每次走到当前结点的父结点，并计算父结点的其他儿子的贡献。每个询问连边的数量应该是$O($询问点在虚拟树上的结点到虚拟树的根的路径上所有结点的度数之和$)$。

对于第2个和第3个测试点，这棵虚拟树的每个结点的度数是期望$O(1)$的，于是每个询问的连边数是期望$O(\log n)$的。

在算法四中，我们对每个分治重心的所有子树建一棵线段树，就相当于在虚拟树上加若干个“虚拟点”，使虚拟树成为一棵二叉树。这样做虚拟树的深度是$O(\log^2 n)$的，于是每个询问的连边数量是$O(\log^2 n)$。

接下来给出一种$O(\log n)$深度的虚拟树的构造方法。

在每次点分治时，在虚拟树上对分治重心的所有子树建一棵二叉树，满足对于这棵二叉树的任意结点，它的左儿子和右儿子在整棵虚拟树上的子树大小都不超过这个结点在整棵虚拟树上的子树大小的一半。每次将$A_1,A_2,...,A_m$这些

子树建二叉树时，找到一个mid，使

$$\sum_{i=1}^{mid-1} size(A_i) \le \frac{1}{2}\sum_{i=1}^{m} size(A_i) \text{且} \sum_{i=mid+1}^{m} size(A_i) \le \frac{1}{2}\sum_{i=1}^{m} size(A_i) \tag{1}$$

（其中，$size(x)$表示x的子树大小），那么这棵二叉树的根就是A_{mid}。

考虑这种虚拟树的深度。每次从一个二叉树结点走向它的儿子时，子树大小至少减少了一半，这样的次数是$O(\log n)$的。每次从一棵二叉树走向另一棵二叉树，相当于从某一层的分治重心走向下一层，这样的次数也是$O(\log n)$的。于是这种虚拟树的深度是$O(\log n)$。

实现起来也很简单，只要将算法四建线段树的过程改成建上述二叉树即可。

这个算法的时间复杂度是$O((n+q)\log^2 n)$，构造出来的图的点数是$O(n\log n+q)$，边数是$O((n+q)\log n)$，

4.6 算法六

对于原树上度数大于3的点，我们加一些虚点，使所有点的度数不超过3。具体做法是：对每个点x的每个儿子，新建一个虚点，将这个儿子结点跟虚点相连，然后虚点跟上一个虚点（一开始为x）连一条长度为0的边。

然后套用算法三就行了。

复杂度是$O((n+q)\log^2 n)$，构造出来的图的点数是$O(n\log n+q)$，边数是$O((n+q)\log n)$，

4.7 算法七

对于树上问题，另一个经典算法是平衡树的启发式合并。

考虑在树上的每个点，用平衡树维护这个子树内所有结点到该结点的距离，以及这个子树内所有询问的d减去询问点到该结点的距离。

在平衡树的update的同时，在DAG上建出对应的边。

每次加入一棵子树时，统计这棵子树与之前所有子树之间的贡献，即将这种情况的边在DAG上建出来。

具体实现方法是，用较小的平衡树上的每一个点，去在较大的平衡树上询问，然后建出相应的边。

每次从一个点走向父亲时，只要对维护这个子树的平衡树打标记即可。

复杂度分析：

首先将原树上有多于一个询问的结点拆成一条链，使每个结点上只有至多一个询问。此时点数还是$O(n)$的。（可以认为$q = O(n)$）

这样在启发式合并的过程中，一个点子树内询问的个数和结点的个数是同一个级别的。

因此统计某棵子树与其它子树之间的贡献的过程的复杂度跟将这棵子树与其它子树合并的复杂度是一样。

由于每个点在启发式合并中只被插入平衡树$O(\log n)$次，所以总复杂度是$O(n \log^2 n)$的。

构造出来的图的点数和边数也是$O(n \log^2 n)$。

由于常数很大不一定能通过$1 \sim 5$号测试点。

4.8 算法八

对于平衡树的合并，可以做到更优的复杂度。

这里以treap为例。

对于一个treap_node T，有以下属性：

```
T -> key            T的权值
T -> priority       T的优先级，这是一个随机数
T -> left           T的左子树
T -> right          T的右子树
```

其中T -> priority满足堆性质，即T -> priority比T的所有子树的priority都小。

treap支持split和join操作。

split(T,x,L,R)表示按照权值是否小于x的权值，将T分为L和R两棵treap。

join(L,R)返回一个由L和R这两棵treap连接而成的treap（L的所有结点的权值都要小于R的所有结点的权值）。

定义merge(T1,T2)，返回由T1和T2中所有元素构成的treap。

merge的伪代码如下：

```
node merge(node T1, node T2)
{
    if (!T1) return T2;
    if (!T2) return T1;
    if (T1 -> priority > T2 -> priority) swap(T1, T2);
    node L, R;
    split(T2, T1, L, R);
    T1 -> left = merge(T1 -> left, L);
    T1 -> right = merge(T1 -> right, R);
    return T1;
}
```

假设T1和T2的大小分别为n和m ($n > m$)，在[1]中证明了上述merge的复杂度为期望$O(m \log \frac{n}{m})$，即$O(\log \binom{n+m}{n})$。这是个理论下界，因为合并两个大小为n和m的集合，要区分$\binom{n+m}{n}$种不同情况，至少要进行$\lceil \log \binom{n+m}{n} \rceil$次比较操作。

可以证明，对于若干个treap，大小分别为$A_1, A_2, ..., A_k$，将这些treap以任意顺序合并的总时间复杂度均为$O(\log \frac{(\sum_{i=1}^{k} A_i)!}{\prod_{i=1}^{k} A_i!})$。特别的，如果$A_1 = A_2 = ... = A_k = 1$，那么时间复杂度为$O(\log(k!)) = O(k \log k)$。

现在考虑原问题的复杂度，每次统计贡献可以用与treap的合并类似的算法，那么统计贡献的复杂度跟合并对应子树的平衡树的复杂度一样，所以只要考虑所有平衡树合并的复杂度。

一共有$O(n)$棵大小为1的treap，最终被合并成2个大小为$O(n)$的treap，所以总时间复杂度是$O(n \log n)$的，空间复杂度是$O(n)$的。

这样构造出来的图的点数和边数都是$O(n \log n)$的，可以通过前5个测试点。

4.9 算法九

对于仙人掌上的问题，可以类比树的点分治，进行“仙人掌分治”。

类似树上的情况，在分治之前，先通过加虚点使每个点的度数为$O(1)$级别。

每次找一个点，使删除这个点和这个点所在环上的所有边后，剩下的连通块的最大大小最小，我们把这个点称为重心。

从重心开始dfs，于是我们得到了若干棵子仙人掌。

对于每棵子仙人掌，我们将其中每个结点到重心的距离排序，然后预处理前缀和。

然后对于每个询问，在其它子仙人掌里查询前缀和并建出相应的边。

对于每个重心所在的环，要处理环上不同子仙人掌中的询问和结点的贡献。类似仙人掌上的DP，我们将环分成两部分，然后问题就转化成二维前缀和问题，可以将环上所有子仙人掌中的结点依次插入一棵线段树来解决。（仙人掌的DP有一些细节，由于篇幅有限略去不讲）

对于删掉重心和重心所在环上的所有边后剩下的每一个连通块，递归进行仙人掌分治。

复杂度分析：

类比树的点分治，可以证明删掉重心和重心所在环上的所有边后，剩下的每个连通块大小都不超过原来的一半。

所以分治的层数是$O(\log n)$的。

每一层分治的时间复杂度是$O(n \log n)$，所以总复杂度为$O(n \log^2 n)$，构造出来的图的点数和边数也是$O(n \log^2 n)$。

由于出题时间有限，出题人并没有对这个算法进行测试。（从理论上来说是可以AC的）

4.10 算法十

我们可以将算法八推广到仙人掌上。

跟算法八一样，每个结点维护两棵平衡树，分别是子仙人掌内所有结点到该结点的距离，和子仙人掌内所有询问的d减去询问结点到该结点的距离。

如果没有环，就跟树上的情况一样，统计这棵子仙人掌与之前所有子仙人掌之间的贡献。

如果有环，就将环分为两部分，然后用平衡树的合并处理前缀和、后缀和，来统计贡献。（这里的细节同仙人掌上的DP）

注意统计完贡献后要平衡树的状态还原，一种可行的方法是，可持久化地进行平衡树的合并。

复杂度分析：

对于环上统计贡献的过程，复杂度跟将环上所有平衡树一起合并是一样的。

所以总时间复杂度还是$O(n \log n)$，构造出来的图的点数和边数也是$O(n \log n)$，可以通过全部测试点。

参考文献

[1] Reid-Miller, Margaret. “Fast Set Operations Using Treaps.” (1998).