虚树初步

SGColin

目录

1 虚树

1.1 概念

给出一棵树，多次询问，每次给出一个点集，保证询问点集大小之和与总点数呈线性关系。

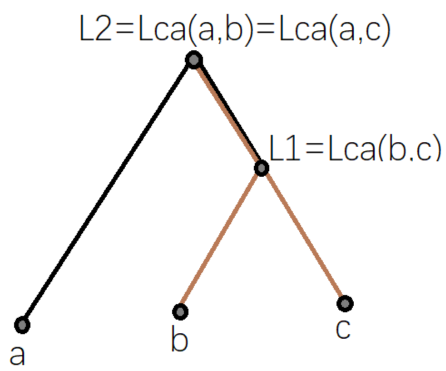
为了使得复杂度是点集大小相关，思想是从原树中提出询问点集大小相关的一个点集，新建一棵树，在这棵树上处理问题。我们称这棵树叫虚树。

可以发现最小化的点集只包含询问点集和他们的 Lca 。

1.2 复杂度证明

可以证明虚树大小和点集大小是线性关系。

考虑询问点集中的三个点 a，b，c 两两求 Lca，至少有两个 Lca 是同一个点。



如果逐个加入点，每次至多会产生一个新的 Lca ，所以虚树中的点数是 $O(2k)$ 级别的。

1.3 一些细节

1. 虚树上节点的父亲不一定是原树上的父亲，虚树上 dfs 的时候注意需要传入 fa 的参数。

2. 关键节点的信息清空也需要点集相关复杂度。如果采用第一种建树方式，虚树内不只是询问点集，所以需要在计算答案 dfs 回溯时顺便清空。如果采用第二种建树方式，可以在计算结束后清空。

1.4 实现

介绍两种写法，复杂度都是 $\mathrm{O}(n\log n)$，复杂度瓶颈在排序，可以用基数排序优化到 $\mathbf{O}(n)$。

1.4.1 方法一

将询问点集按照 dfn 排序，用一个栈维护虚树当前的最右链，模拟 dfs 的过程逐个插入节点。

对于每一个插入的节点 x，设 L = Lca(x, stk[top])，分情况讨论：

(1) L = stk[top]，栈顶元素为当前元素的祖先。此时栈中元素依然在最右链上，退出。

(2) L = stk[top - 1]，L 为栈中维护的元素。将 stk[top] 退栈，与 stk[top - 1] 连边，退出。

(3) dfn[L] > dfn[stk[top - 1]]，L 在原树上 stk[top] 与 stk[top - 1] 之间。此时 L 为 stk[top] 的祖先，将 stk[top] 退栈，与 L 连边，把 L 进栈，退出。

(4) dfn[L] < dfn[stk[top - 1]]，L 为 stk[top - 1] 的祖先。将 stk[top] 退栈，与 stk[top - 1] 连边。

退出循环后将当前点进栈。

退栈合法的原因是该子树中的关键点已经遍历完毕，该节点之后不再在最右链上。

下面是此方法的模板，k 为点集大小，s[1...k] 为询问点集。

```
bool cmp(int x, int y) {return dfn[x] < dfn[y];}
void insert(int u) {
  int l = lca(u, stk[top]);
  if (l == stk[top]) {stk[++top] = u; return;}
  while (top > 1 && dfn[stk[top - 1]] >= dfn[l]) {
    addx(stk[top - 1], stk[top]); --top;
  }
  if (l != stk[top]) addx(l, stk[top]), stk[top] = l;
  stk[++top] = u;
}
void work() {
  k = rd();
  tot = 0;  //清空邻接表
  for (int i = 1; i <= k; ++i) s[i] = rd();
  sort(s + 1, s + 1 + k, cmp);
  stk[top = 1] = 1;  //先放入一个根节点
  for (int i = 1; i <= k; ++i) insert(s[i]);
  while(--top) addx(stk[top], stk[top + 1]);  //构建最右链
  ...  //计算答案
}
```

注意维护的栈可能出现特殊情况，栈中最好先放置一个根节点，代码中放入的是 1 号节点。

插入结束后栈中还保存着最右链，最后要把所有元素退栈，过程中相邻元素连边。

1.4.2 方法二

注意到上一方法讨论集中在新的 Lca 应该放置在什么位置，我们尝试避开繁琐的讨论。

因为新产生的 Lca 都是按顺序两两相邻求得的，所以我们先对点集进行排序，然后两两相邻求 Lca，也都放入这个数组。然后对得到的新点集再进行排序，用 unique 去重，就得到了虚树所有的点，并且已经按照 dfn 排好顺序。

接下来问题就是如何建树，我们依旧用一个栈模拟 dfs 。设 $ed\_dfn[u]$ 表示原树中节点 u 的子树 dfn 的最大值，即 $dfn[u]+sz[u]-1$ 。那么如果 $ed\_dfn[stk[top]]<dfn[u]$ ，证明 $stk[top]$ 的子树已经遍历完毕，因此直接将其退栈。

此方法相比上一方法优美很多。连边的形式很简单，因为退栈结束后栈顶一定是当前点在虚树中的父节点，所以进栈的时候连边即可。同时我们可以直接得到虚树的点集，便于预处理和清空。而且此方法不用讨论根节点提前进栈的影响（是否包含于询问点集中）。

```
bool cmp(int u, int v) {return dfn[u] < dfn[v];}

void work() {
  tot = 0;
  sort(a + 1, a + 1 + tota, cmp);
  for (int i = 1, lim = tota; i < lim; ++i) a[++tota] = lca(a[i], a[i + 1]);
  sort(a + 1, a + 1 + tota, cmp);
  tota = unique(a + 1, a + 1 + tota) - a - 1;  //去重
  for (int i = 1; i <= tota; ++i) hd[a[i]] = 0;  //清空头指针
  top = 0;
  for (int i = 1, u; i <= tota; ++i) {
    u = a[i];
    while (top && ed_dfn[stk[top]] < dfn[u]) --top;
    stk[++top] = u;
    if (top >= 2) add(stk[top], stk[top - 1]);  //栈内有当前点父节点，连边
  }
  ...  //计算答案
}
```

此方法需要注意点集数组需要开到 $2n$ 大小。

2 题目小结

2.1 [SDOI 2011] 消耗战

2.1.1 Description

BZOJ 2286　Luogu 2495

给定一棵树，边有边权。

多次询问，每次给出一个不包含 1 号点的点集 S。

最小化使得 S 中的点与 1 号点不连通断边的权值和。

2.1.2 Solution

Code（法一）　Code（法二）

dfs 预处理 mn[i] 表示节点 i 到根的路径上边权最小值。

设 f[u] 表示节点 u 的子树里所有特殊点都与 1 号点不连通的最小代价，那么有

$$f[u] = \begin{cases} mn[u] & \text{if } u \in S \\ \min(mn[u], \sum_{v \in son[u]} f[v]) & \text{if } u \notin S \end{cases}$$

注意第一项会直接 return，子树内的关键点不会遍历到，所以最好在计算后再清空标记。

2.2 [HEOI 2014] 大工程

2.2.1 Description

BZOJ 3611　Luogu 4103

给定一棵树，边有边权，定义两点距离为树上的链长。

多次询问，每次给出一个点集 S。计算点集中两两距离之和，两两距离中的最小值和最大值。

2.2.2 Solution

Code

本题建议采用法二建树，法一还需要讨论一号点是否在询问点集里。

1. 两两距离之和：考虑每条边会被计算多少次，答案是 $sz[v] * (|S| - sz[v]) * w$;

2. 两两距离最小值：设 $g[u]$ 表示 u 距离子树中 S 中的点距离最小值，有 $f[u] = u \in S\ ?\ 0\ :\ \infty$

然后像树的直径那样去 DP 就可以了，先用 $g[u] + g[v] + w$ 更新答案，再用 $g[v] + w$ 更新 $g[u]$。

3. 两两距离最大值：容易发现虚树所有叶节点必定属于 S，因此答案就是虚树的直径。

2.3 [SDOI 2015] 寻宝游戏

2.3.1 Description

BZOJ 3391 Luogu 4103

给定一棵树，边有边权，定义两点距离为树上的链长。

维护一个集合 S，多次操作，每次给出一个点 x，反转 x 是否在 S 中的状态。然后计算从树上任意一点出发，一条路径遍历一遍点集中的点，再回到出发点的最短路径长度。

2.3.2 Solution

Code

可以发现最优的路径形成一个环，经过的每条树边都要访问两次，且每棵子树最多进入一次。

这个过程的一种描述是按照 dfs 序访问，因此我们只需要维护一个按照 dfs 序排好的集合。插入时将答案加上它到前驱和后继的距离和，减掉前驱和后继的距离。删除时将答案减掉它到前驱和后继的距离和，加上前驱和后继的距离。

2.4 [北京集训 2018] 小奇的危机

2.4.1 Description

给定一棵树，边有边权，定义两点距离为树上的链长。

多次询问，每次给定 l, r, x，求一个点 y，满足 $l \le y \le r$，且最小化 $dis(x, y)$。

2.4.2 Solution

Code

这里是离线的做法，时间复杂度 $\mathrm{O}((n+q)\log^2(n+q))$。我们以点的编号为下标建一棵线段树，把每个询问在线段树上拆成 log 个区间，取这些区间的最优解作为该区间的答案。

那么就可以知道每个线段树节点上有哪些询问了。我们**对每个线段树节点**都建一棵虚树，包含其代表区间里的所有点，以及节点上的询问点。在这棵树上我们可以通过换根 dp 求出每个点到关键点的最短距离，然后更新答案即可。

关于 dp 有一些细节。我们设 $g[u]$ 表示节点 u 到子树里的关键点最近的距离，$h[u]$ 表示节点 u 到子树以外的关建点的最近距离。

预处理之后 $g[u]$ 的求法直接 dfs 一遍。$h[u]$ 的更新按照常理应该是分**父节点以外的关键点**和**兄弟节点子树内的关键点**两部分讨论，但是取 min 操作不支持分离贡献。因此做法是把所有子节点记录成一个序列，正反各扫描一遍，分别取前缀 / 后缀更新当前答案。

```
void dp2(int u, int fa) {
  for (int i = hd[u], v; i; i = e[i].nxt)  //记录子节点序列
    if ((v = e[i].to) != fa) ch[++cntch] = v, chw[cntch] = e[i].w;
  int mn = inf;
  for (int i = 1, v, w; i <= cntch; ++i) {  //正序扫描
    v = ch[i]; w = chw[i];
    h[v] = min(h[v], min(h[u] + w, mn + w));  //分父节点以外和兄弟节点更新
    mn = min(mn, g[v] + w);
  }
  mn = inf;
  for (int i = cntch, v, w; i; --i) {  //逆序扫描
    v = ch[i]; w = chw[i];
    h[v] = min(h[v], mn + w);
    mn = min(mn, g[v] + w);
  }
  for (int i = hd[u], v; i; i = e[i].nxt)
    if ((v = e[i].to) != fa) dp2(v, u);
}
```

其实有更简单的写法。之所以用兄弟节点更新时正反顺序都扫描一遍，是为了避免自己更新自己的情况。而此题我们更新询问答案的时候用的是 $\min(g[u], h[u])$，而自己更新自己时有 $h[u] \geq 2 \times g[u]$，所以不会影响最后的答案。因此可以直接用 $g[fa] + w$ 更新 $h[u]$。

```
void dp2(int u, int fa, int w) {
  //分父节点以外和兄弟节点更新
  if (fa) h[u] = min(h[u], min(g[fa], h[fa]) + w);
  for (int i = hd[u], v; i; i = e[i].nxt)
    if ((v = e[i].to) != fa) dp2(v, u, e[i].w);
}
```

此题有在线的做法，时间复杂度不变，可以看 这里。

2.5 [LNOI 2014] LCA

2.5.1 Description

BZOJ 3626　Luogu 4211

给定一棵以 1 号节点为根的有根树，定义深度为这个节点到根的距离 +1。

多次询问，每次给出 l, r, x ，询问 $\sum_{i=l}^{r} deep(lca(i, x))$。

2.5.2 Solution

Code

这里给出一个虚树的做法，**思路和上一题相同**，时间复杂度 $\mathrm{O}((n+q)\log^2(n+q))$。

类似的把询问离线，每个询问拆成 log 个区间，询问的答案即为这 log 个询问答案的和。然后对每个线段树节点都建一棵虚树，包含其代表区间里的所有点，以及节点上的询问点。然后任务就是 dp 求出每一个点和代表区间里的点对应的 Lca 深度之和，这个可以通过换根 dp 求出。

下面我们称**原本在线段树代表区间**内的点为“关键点”，设 $sz[u]$ 表示节点 u 子树里的关键点个数。设 $g[u]$ 表示节点 u 子树里的关键点与 u 的 Lca 深度之和。$h[u]$ 表示节点 u 子树以外的关键点与 u 的 Lca 深度之和。显然的有 $g[u] = sz[u] \times deep[u]$ 。

考虑 $h[u]$ 的构成，与父节点以外的子树求出的 Lca 一定是和父节点结果相同，所以这部分取 $h[fa]$ 。兄弟子树之间的 Lca 一定是父节点，所以这部分为 $(sz[fa] - sz[u]) \times deep[fa]$。

在线的做法同上一题。