树上莫队

括号序树上莫队

一般的莫队只能处理线性问题，我们要把树强行压成序列。

我们可以将树的括号序跑下来，把括号序分块，在括号序上跑莫队。

具体怎么做呢?

dfs 一棵树，然后如果 dfs 到 x 点，就 `push_back(x)` ，dfs 完 x 点，就直接 `push_back(-x)` ，然后我们在挪动指针的时候，

- 新加入的值是 x ---> `add(x)`
- 新加入的值是 - x ---> `del(x)`
- 新删除的值是 x ---> `del(x)`
- 新删除的值是 - x ---> `add(x)`

这样的话，我们就把一棵树处理成了序列。

> **例题「WC2013」糖果公园 [https://uoj.ac/problem/58]**
>
> 题意：给你一棵树，每个点有颜色，每次询问
>
> $\sum_c val_c \sum_{i=1}^{cnt_c} w_i$
>
> 其中： val 表示该颜色的价值， cnt 表示颜色出现的次数， w 表示该颜色出现 i 次后的价值

先把树变成序列，然后每次添加/删除一个点，这个点的对答案的的贡献是可以在 $O(1)$ 时间内获得的，即 $val_c \times w_{cnt_{c+1}}$

发现因为他会把起点的子树也扫了一遍，产生多余的贡献，怎么办呢?

因为扫的过程中起点的子树里的点肯定会被扫两次，但贡献为 0。

所以可以开一个 vis 数组，每次扫到点 x，就把 vis_x 异或上 1。

如果 $vis_x = 0$，那这个点的贡献就可以不计。

所以可以用树上莫队来求。

修改的话，加上一维时间维即可，变成带修改树上莫队。

然后因为所包含的区间内可能没有 LCA，对于没有的情况要将多余的贡献删除，然后就完事了。

参考代码

```cpp
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;

const int maxn = 200010;

int f[maxn], g[maxn], id[maxn], head[maxn], cnt,
last[maxn], dep[maxn],
    fa[maxn][22], v[maxn], w[maxn];
int block, index, n, m, q;
int pos[maxn], col[maxn], app[maxn];
bool vis[maxn];
long long ans[maxn], cur;

struct edge {
  int to, nxt;
} e[maxn];
int cnt1 = 0, cnt2 = 0;  // 时间戳

struct query {
  int l, r, t, id;
  bool operator<(const query &b) const {
    return (pos[l] < pos[b.l]) || (pos[l] == pos[b.l] &&
pos[r] < pos[b.r]) ||
           (pos[l] == pos[b.l] && pos[r] == pos[b.r] && t
< b.t);
  }
} a[maxn], b[maxn];

inline void addedge(int x, int y) {
  e[++cnt] = (edge){y, head[x]};
  head[x] = cnt;
```

```
}

void dfs(int x) {
  id[f[x] = ++index] = x;
  for (int i = head[x]; i; i = e[i].nxt) {
    if (e[i].to != fa[x][0]) {
      fa[e[i].to][0] = x;
      dep[e[i].to] = dep[x] + 1;
      dfs(e[i].to);
    }
  }
  id[g[x] = ++index] = x;  // 括号序
}

inline int lca(int x, int y) {
  if (dep[x] < dep[y]) swap(x, y);
  if (dep[x] != dep[y]) {
    int dis = dep[x] - dep[y];
    for (int i = 20; i >= 0; i--)
      if (dis >= (1 << i)) dis -= 1 << i, x = fa[x][i];
  }  // 爬到同一高度
  if (x == y) return x;
  for (int i = 20; i >= 0; i--) {
    if (fa[x][i] != fa[y][i]) x = fa[x][i], y = fa[y][i];
  }
  return fa[x][0];
}

inline void add(int x) {
  if (vis[x])
    cur -= (long long)v[col[x]] * w[app[col[x]]--];
  else
    cur += (long long)v[col[x]] * w[++app[col[x]]];
  vis[x] ^= 1;
}

inline void modify(int x, int t) {
  if (vis[x]) {
    add(x);
    col[x] = t;
    add(x);
  } else
    col[x] = t;
}  // 在时间维上移动

int main() {
  scanf("%d%d%d", &n, &m, &q);
  for (int i = 1; i <= m; i++) scanf("%d", &v[i]);
  for (int i = 1; i <= n; i++) scanf("%d", &w[i]);
```

```
  for (int i = 1; i < n; i++) {
    int x, y;
    scanf("%d%d", &x, &y);
    addedge(x, y);
    addedge(y, x);
  }
  for (int i = 1; i <= n; i++) {
    scanf("%d", &last[i]);
    col[i] = last[i];
  }
  dfs(1);
  for (int j = 1; j <= 20; j++)
    for (int i = 1; i <= n; i++)
      fa[i][j] = fa[fa[i][j - 1]][j - 1];  // 预处理祖先
  int block = pow(index, 2.0 / 3);
  for (int i = 1; i <= index; i++) {
    pos[i] = (i - 1) / block;
  }
  while (q--) {
    int opt, x, y;
    scanf("%d%d%d", &opt, &x, &y);
    if (opt == 0) {
      b[++cnt2].l = x;
      b[cnt2].r = last[x];
      last[x] = b[cnt2].t = y;
    } else {
      if (f[x] > f[y]) swap(x, y);
      a[++cnt1] = (query){lca(x, y) == x ? f[x] : g[x],
f[y], cnt2, cnt1};
    }
  }
  sort(a + 1, a + cnt1 + 1);
  int L, R, T;  // 指针坐标
  L = R = 0;
  T = 1;
  for (int i = 1; i <= cnt1; i++) {
    while (T <= a[i].t) {
      modify(b[T].l, b[T].t);
      T++;
    }
    while (T > a[i].t) {
      modify(b[T].l, b[T].r);
      T--;
    }
    while (L > a[i].l) {
      L--;
      add(id[L]);
    }
    while (L < a[i].l) {
```

```
132         add(id[L]);
133         L++;
134       }
135       while (R > a[i].r) {
136         add(id[R]);
137         R--;
138       }
139       while (R < a[i].r) {
140         R++;
141         add(id[R]);
142       }
143       int x = id[L], y = id[R];
144       int llca = lca(x, y);
145       if (x != llca && y != llca) {
146         add(llca);
147         ans[a[i].id] = cur;
148         add(llca);
149       } else
150         ans[a[i].id] = cur;
151     }
152     for (int i = 1; i <= cnt1; i++) {
        printf("%lld\n", ans[i]);
      }
      return 0;
    }
```

真·树上莫队

上面的树上莫队只是将树转化成了链，下面的才是真正的树上莫队。

由于莫队相关的问题都是模板题，因此实现部分不做太多解释

询问的排序

首先我们知道莫队的是基于分块的算法，所以我们需要找到一种树上的分块方法来保证时间复杂度。

条件：

- 属于同一块的节点之间的距离不超过给定块的大小
- 每个块中的节点不能太多也不能太少
- 每个节点都要属于一个块

- 编号相邻的块之间的距离不能太大

了解了这些条件后，我们看到这样一道题 「SCOI2005」王室联邦 [https://loj.ac/problem/2152] 。

在这道题的基础上我们只要保证最后一个条件就可以解决分块的问题了。

> **思路**
>
> 令 lim 为希望块的大小，首先，对于整个树 dfs，当子树的大小大于 lim 时，就将它们分在一块，容易想到：对于根，可能会剩下一些点，于是将这些点分在最后一个块里。

做法：用栈维护当前节点作为父节点访问它的子节点，当从栈顶到父节点的距离大于希望块的大小时，弹出这部分元素分为一块，最后剩余的一块单独作为一块。

最后的排序方法：若第一维时间戳大于第二维，交换它们，按第一维所属块为第一关键字，第二维时间戳为第二关键字排序。

指针的移动

容易想到，我们可以标记被计入答案的点，让指针直接向目标移动，同时取反路径上的点。

但是，这样有一个问题，若指针一开始都在 x 上，显然 x 被标记，当两个指针向同一子节点移动（还有许多情况）时，x 应该不被标记，但实际情况是 x 被标记，因为两个指针分别标记了一次，抵消了。

如何解决呢?

有一个很显然的性质：这些点肯定是某些 LCA，因为 LCA 处才有可能被重复撤销导致撤销失败。

所以我们每次不标记 LCA，到需要询问答案时再将 LCA 标记，然后再撤销。

```
//取反路径上除LCA以外的所有节点
void move(int x, int y) {
  if (dp[x] < dp[y]) swap(x, y);
  while (dp[x] > dp[y]) update(x), x = fa[x];
  while (x != y) update(x), update(y), x = fa[x], y = fa[y];
```

```
  // x!=y保证LCA没被取反
}
```

对于求 LCA，我们可以用树剖，然后我们就可以把分块的步骤放到树剖的第一次 dfs 里面，时间戳也可以直接用第二次 dfs 的 dfs 序。

```
int bl[100002], bls = 0;  // 属于的块，块的数量
unsigned step;            // 块大小
int fa[100002], dp[100002], hs[100002] = {0}, sz[100002] =
{0};
//父节点，深度，重儿子，大小
stack<int> sta;
void dfs1(int x) {
  sz[x] = 1;
  unsigned ss = sta.size();
  for (int i = head[x]; i; i = nxt[i])
    if (ver[i] != fa[x]) {
      fa[ver[i]] = x;
      dp[ver[i]] = dp[x] + 1;
      dfs1(ver[i]);
      sz[x] += sz[ver[i]];
      if (sz[ver[i]] > sz[hs[x]]) hs[x] = ver[i];
      if (sta.size() - ss >= step) {
        bls++;
        while (sta.size() != ss) bl[sta.top()] = bls,
sta.pop();
      }
    }
  sta.push(x);
}
// main
if (!sta.empty()) {
  bls++;  // 这一行可写可不写
  while (!sta.empty()) bl[sta.top()] = bls, sta.pop();
}
```

时间复杂度

重点到了，这里关系到块的大小取值。

设块的大小为 $unit$：

- 对于 x 指针，由于每个块中节点的距离在 $unit$ 左右，每个块中 x 指针移动 $unit^2$ 次（$unit \times dis_max$），共计 $n \times unit$ （$unit^2 \times (n \div unit)$

）次；

- 对于 y 指针，每个块中最多移动 $O(n)$ 次，共计 $n^2 \div unit$ （$n \times (n \div unit)$）次。

加起来大概在根号处取得最小值（由于树上莫队块的大小不固定，所以不一定要严格按照）。

例题「WC2013」糖果公园

由于多了时间维，块的大小取到 $0.6n$ 的样子就差不多了。

参考代码

```
#include <bits/stdc++.h>
using namespace std;
inline int gi() {
  register int x, c, op = 1;
  while (c = getchar(), c < '0' || c > '9')
    if (c == '-') op = -op;
  x = c ^ 48;
  while (c = getchar(), c >= '0' && c <= '9')
    x = (x << 3) + (x << 1) + (c ^ 48);
  return x * op;
}
int head[100002], nxt[200004], ver[200004], tot = 0;
void add(int x, int y) {
  ver[++tot] = y, nxt[tot] = head[x], head[x] = tot;
  ver[++tot] = x, nxt[tot] = head[y], head[y] = tot;
}
int bl[100002], bls = 0;
unsigned step;
int fa[100002], dp[100002], hs[100002] = {0}, sz[100002]
= {0}, top[100002],
                              id[100002];
stack<int> sta;
void dfs1(int x) {
  sz[x] = 1;
  unsigned ss = sta.size();
  for (int i = head[x]; i; i = nxt[i])
    if (ver[i] != fa[x]) {
      fa[ver[i]] = x, dp[ver[i]] = dp[x] + 1;
      dfs1(ver[i]);
      sz[x] += sz[ver[i]];
      if (sz[ver[i]] > sz[hs[x]]) hs[x] = ver[i];
```

```
      if (sta.size() - ss >= step) {
        bls++;
        while (sta.size() != ss) bl[sta.top()] = bls,
sta.pop();
      }
    }
  sta.push(x);
}
int cnt = 0;
void dfs2(int x, int hf) {
  top[x] = hf, id[x] = ++cnt;
  if (!hs[x]) return;
  dfs2(hs[x], hf);
  for (int i = head[x]; i; i = nxt[i])
    if (ver[i] != fa[x] && ver[i] != hs[x]) dfs2(ver[i],
ver[i]);
}
int lca(int x, int y) {
  while (top[x] != top[y]) {
    if (dp[top[x]] < dp[top[y]]) swap(x, y);
    x = fa[top[x]];
  }
  return dp[x] < dp[y] ? x : y;
}
struct qu {
  int x, y, t, id;
  bool operator<(const qu a) const {
    return bl[x] == bl[a.x] ? (bl[y] == bl[a.y] ? t < a.t
: bl[y] < bl[a.y])
                            : bl[x] < bl[a.x];
  }
} q[100001];
int qs = 0;
struct ch {
  int x, y, b;
} upd[100001];
int ups = 0;
long long ans[100001];
int b[100001] = {0};
int a[100001];
long long w[100001];
long long v[100001];
long long now = 0;
bool vis[100001] = {0};
void back(int t) {
  if (vis[upd[t].x]) {
    now -= w[b[upd[t].y]--] * v[upd[t].y];
    now += w[++b[upd[t].b]] * v[upd[t].b];
  }
```

```
  a[upd[t].x] = upd[t].b;
}
void change(int t) {
  if (vis[upd[t].x]) {
    now -= w[b[upd[t].b]--] * v[upd[t].b];
    now += w[++b[upd[t].y]] * v[upd[t].y];
  }
  a[upd[t].x] = upd[t].y;
}
void update(int x) {
  if (vis[x])
    now -= w[b[a[x]]--] * v[a[x]];
  else
    now += w[++b[a[x]]] * v[a[x]];
  vis[x] ^= 1;
}
void move(int x, int y) {
  if (dp[x] < dp[y]) swap(x, y);
  while (dp[x] > dp[y]) update(x), x = fa[x];
  while (x != y) update(x), update(y), x = fa[x], y =
fa[y];
}
int main() {
  int n = gi(), m = gi(), k = gi();
  step = (int)pow(n, 0.6);
  for (int i = 1; i <= m; i++) v[i] = gi();
  for (int i = 1; i <= n; i++) w[i] = gi();
  for (int i = 1; i < n; i++) add(gi(), gi());
  for (int i = 1; i <= n; i++) a[i] = gi();
  for (int i = 1; i <= k; i++)
    if (gi())
      q[++qs].x = gi(), q[qs].y = gi(), q[qs].t = ups,
q[qs].id = qs;
    else
      upd[++ups].x = gi(), upd[ups].y = gi();
  for (int i = 1; i <= ups; i++) upd[i].b = a[upd[i].x],
a[upd[i].x] = upd[i].y;
  for (int i = ups; i; i--) back(i);
  fa[1] = 1;
  dfs1(1), dfs2(1, 1);
  if (!sta.empty()) {
    bls++;
    while (!sta.empty()) bl[sta.top()] = bls, sta.pop();
  }
  for (int i = 1; i <= n; i++)
    if (id[q[i].x] > id[q[i].y]) swap(q[i].x, q[i].y);
  sort(q + 1, q + qs + 1);
  int x = 1, y = 1, t = 0;
  for (int i = 1; i <= qs; i++) {
```

```
      if (x != q[i].x) move(x, q[i].x), x = q[i].x;
      if (y != q[i].y) move(y, q[i].y), y = q[i].y;
      int f = lca(x, y);
      update(f);
      while (t < q[i].t) change(++t);
      while (t > q[i].t) back(t--);
      ans[q[i].id] = now;
      update(f);
    }
    for (int i = 1; i <= qs; i++) printf("%lld\n", ans[i]);
    return 0;
  }
```

评论