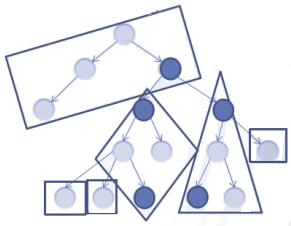
# **[树上分块](https://www.cnblogs.com/hua-dong/p/8275227.html)**



#### **树上的有些问题是可以用树剖或者动态树解决的，但是他们有一个共同点就是：不连通。**

* 比如求u到v的路径权值和，或者最大值：

                u到v可能对应了多个链，这多个链在对应的数据结构（假设是线段树）上面对应不同的区间。但是线段树上这几个区间的不连续并不影响我们得到答案。

                (~~当然求子树的信息话是连续区间)~~

那么如果我们遇到的问题要求区间连续呢，比如求**u到v的路径上点的权值有多少种**？如果不连续就得处理链与链之间的关系。显然这些点得待在一起，如果树剖很难维护链与链之间的关系。

#### **树分块，大概有这样的一些方法：**

* **王室联邦分块法**：可以保证每个块的大小和直径都不超过2√N−1，但是不保证块联通
* **DFS序分块法**：首先是好写（毕竟转化成了序列问题），严格保证块大小√N，但是不保证直径，也不保证联通。处理子树信息比较方便
* **size分块**：检查当前节点的父亲所在块的大小，如果小于√N就把当前节点加入进去，不然新开块。块大小最坏√N，保证块内联通，还保证直径，多么优美啊可惜不能保证块个数（一个菊花图就死了）

#### 

#### **王室联邦分块，假设每个块的数量个数为[B,3B].**

     如何分组可以看,裸题 BZOJ1086 [王室联邦](https://vjudge.net/problem/HYSBZ-1086" \t "https://www.cnblogs.com/hua-dong/p/_blank)。

# BZOJ1086——树上分块

Description   
　　“余”人国的国王想重新编制他的国家。他想把他的国家划分成若干个省，每个省都由他们王室联邦的一个成员来管理。他的国家有n个城市，编号为1..n。一些城市之间有道路相连，任意两个不同的城市之间有且仅有一条直接或间接的道路。为了防止管理太过分散，每个省至少要有B个城市，为了能有效的管理，每个省最多只有3B个城市。每个省必须有一个省会，这个省会可以位于省内，也可以在该省外。但是该省的任意一个城市到达省会所经过的道路上的城市（除了最后一个城市，即该省省会）都必须属于该省。一个城市可以作为多个省的省会。聪明的你快帮帮这个国王吧！

Input   
　　第一行包含两个数N，B（1<=N<=1000, 1 <= B <= N）。接下来N－1行，每行描述一条边，包含两个数，即这条边连接的两个城市的编号。

Output   
　　如果无法满足国王的要求，输出0。否则输出数K，表示你给出的划分方案中省的个数，编号为1..K。第二行输出N个数，第I个数表示编号为I的城市属于的省的编号，第三行输出K个数，表示这K个省的省会的城市编号，如果有多种方案，你可以输出任意一种。

Sample Input   
8 2

1 2

2 3

1 8

8 7

8 6

4 6

6 5   
Sample Output   
3

2 1 1 3 3 3 3 2

2 1 8

###### **这道题可谓是树上分块的模板，由于树上分块对于树上莫队算法有极其重要的作用，所以在这里进行介绍。**

###### **我们在树上将点进行分类，每一块的大小都为B到3B。要使每一块的大小均为B至3B，只需要开一个栈，在我们遍历每一个点时搜寻它目前在栈中的子树大小，如果大于B，我们就将他以及他的儿子放在一个块里，并且这个点就是该题中的省会。**

###### **在我们dfs完之后，还会剩下一些点，这些点肯定少于B个，所以我们将他归入最后一个分块即可。（最后一个分块一定与它相连）**

## **分析**

一次dfs即可解决。做法如下：

* 任意选取一个点开始dfs
* 对一个点，搜索它的所有儿子
* 如果搜索完一个儿子后发现栈中元素大于B个，那么把栈中的元素分成一个块
* 否则继续搜索
* 一个点dfs结束时把它本身入栈

实现中有一个问题，对于一个儿子，它在栈中积累了不到B个点，而在下一个搜索的儿子的下层达到了B个点，那么这样两个块就会不连通，出现问题。所以我们每次设置一个bottom，意为对于这个儿子它的栈底是哪里，这样就可以保证不会弹出之前的节点，从而保证连通性。每次弹栈的时候把当前搜索的节点设为省会，就一定能够符合要求。最后搜索完之后栈中会剩下一些节点，而这些节点必定与最后一个块连通，而最后一个块的节点个数和剩下的节点个数和sum∈(2B,3B]，所以可以归进同一个块内。代码实现非常简单。

**这道题是树分块方法的模版题，树分块在很多树上询问问题中都有应用。**

#include<cstdio>

#include<iostream>

#include<algorithm>

#include<cstring>using namespace std;

int const maxn=2010;

int q[maxn],group[maxn],rt[maxn],top;int Laxt[maxn],Next[maxn],To[maxn],cnt,ans;

int n,B;void init()

{

ans=0; top=0; cnt=0;

memset(Laxt,0,sizeof(Laxt));

}void add(int u,int v)

{

Next[++cnt]=Laxt[u];

Laxt[u]=cnt;

To[cnt]=v;

}

void dfs(int u,int pre)

{

int Now=top;//记下栈顶指针

for(int i=Laxt[u];i;i=Next[i])//枚举子树

if(To[i]!=pre){

dfs(To[i],u);

if(top-Now>=B) {//新的一块，块编号为ans，省会为U

rt[++ans]=u; while(top!=Now) group[q[top--]]=ans;}

}

}

q[++top]=u;

}

int main()

{

while(~scanf("%d%d",&n,&B)){

int u,v;

init();

for(int i=1;i<n;i++){

scanf("%d%d",&u,&v);

add(u,v);add(v,u);

}

dfs(1,0);

while(top) group[q[top--]]=ans;

printf("%d\n",ans);

for(int i=1;i<=n;i++) printf("%d ",group[i]);printf("\n");

for(int i=1;i<=ans;i++) printf("%d ",rt[i]);printf("\n");

} return 0;

}

#### 

#### **再稍微介绍一下王室联邦分块是干嘛的：**

     我们dfs，把子树中大于B的分为一组，剩余的（肯定小于B）上传分到父亲那组。由于父亲那组大于B，加进去小于3B。每一组即比较平均了，B的大小会影响空间和时间的优劣，需要根据题目给定的时间和空间，时间多空间小就B大，空间多时间少就B小。从而来决定B的大小。

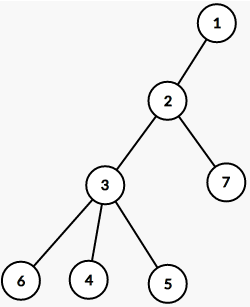
     这样分块是为了莫队的排序，而不是预处理保存信息。比如，(u,v) 转移到(a,b)，由于u和a或在一个组里面，即距离不太远，转移时间不太大。

# **[SPOJ COT2 Count on a tree II 树上莫队算法](https://www.cnblogs.com/AOQNRMGYXLMV/p/5289778.html)**

#### **题意：**

给出一棵n(n≤4×104)个节点的树，每个节点上有个权值，和m(m≤105)个询问。  
每次询问路径u→v上有多少个权值不同的点。

分析：

欧拉序

操作：当访问到一个点的时候，把它加进序列。当离开这个点的时候，把它加进序列。

举个栗子：

这棵树的欧拉序就是：1,2,3,6,6,4,4,5,5,3,7,7,2,1

我们已经会序列莫队了，那么如果能把树上问题转成序列问题，问题就解决了，那就是欧拉序！

我们不妨记i第一次出现位置为in[i]，第二次为out[i]。那么在询问区间中出现了两次的数必定是不会产生贡献的，根据欧拉序的性质，x和y之间路径不会经过它。则用一个桶记录编号为i的节点已经出现了几次，出现两次的节点需要减去它的贡献。

那么有这么两种情况：

…x…x…y…y…询问[out[x],in[y]]

…x…y…y…x…询问[in[x],in[y]]

但是仔细想想欧拉序的遍历方法，好像又有点不对，因为在第一种情况下，lca并不会出现在它们两之间的欧拉序列之中，特判加入即可。

#### **分析：**

* 树分块

首先将树分块，每块的大小为√n左右。  
然后将询问离线处理，按照区间上的莫队算法将询问按块排序。  
[这里](http://www.lydsy.com:808/JudgeOnline/problem.php?id=1086)有一道裸的树分块的题目。

* 树上的路径转移

定义S(u,v)表示路径u→v上的点集，定义⨁为集合的对称差，类似于异或运算。  
那么有S(u,v)=S(root,u)⨁S(root,v)⨁LCA(u,v)，有一个LCA不方便处理。  
再定义一个T(u,v)=S(root,u)⨁S(root,v)  
T(u1,v1)⨁T(v1,v2)=S(root,u1)⨁S(root,v1)⨁S(root,v1)⨁S(root,v2)  
消去中间两项得到：T(u1,v1)⨁T(v1,v2)=S(root,u1)⨁S(root,v2)=T(u1,v2)

从结论可以看出，由T(u1,v1)到T(u1,v2)只需要⨁一个T(v1,v2)。  
由于对称差⨁运算满足交换律和结合律，所以再⨁一个T(u1,u2)就得到T(u2,v2)。

假设上次查询的路径为u→v，我们维护点集T(u,v)的信息：in(u)点u是否在集合中，cnt(x)集合中权值为x点的个数，diff权值不同的点数。  
查询的话如果LCA(u,v)的权值没有出现过，答案就是diff+1，否则就是diff。

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

const int maxn = 40000 + 10;

const int maxq = 100000 + 10;

struct Edge

{

int v, nxt;

Edge() {}

Edge(int v, int nxt): v(v), nxt(nxt) {}

};

int ecnt, head[maxn];

Edge edges[maxn \* 2];

void AddEdge(int u, int v) {

edges[ecnt] = Edge(v, head[u]);

head[u] = ecnt++;

}

int n, m;

int a[maxn], b[maxn], tot;

int anc[maxn][20], dep[maxn];

int group[maxn], blocks, sz;

int S[maxn], top;

void dfs(int u) {

int cur = top;

for(int i = head[u]; ~i; i = edges[i].nxt) {

int v = edges[i].v;

if(v == anc[u][0]) continue;

anc[v][0] = u;

dep[v] = dep[u] + 1;

dfs(v);

if(top - cur >= sz) {

blocks++;

while(top != cur) group[S[top--]] = blocks;

}

}

S[++top] = u;

}

struct Query

{

int u, v, id;

bool operator < (const Query& t) const {

return group[u] < group[t.u] || (group[u] == group[t.u] && group[v] < group[t.v]);

}

}q[maxq];

void preprocess() {//倍增

for(int j = 1; (1 << j) < n; j++)

for(int i = 1; i <= n; i++) if(anc[i][j-1])

anc[i][j] = anc[anc[i][j-1]][j-1];

}

int LCA(int u, int v) {

if(dep[u] < dep[v]) swap(u, v);

int log;

for(log = 0; (1 << log) < dep[u]; log++);

for(int i = log; i >= 0; i--)

if(dep[u] - (1<<i) >= dep[v]) u = anc[u][i];

if(u == v) return u;

for(int i = log; i >= 0; i--)

if(anc[u][i] && anc[u][i] != anc[v][i])

u = anc[u][i], v = anc[v][i];

return anc[u][0];

}

int cnt[maxn], dif, in[maxn];

void xorvertex(int u) {//异或

if(in[u]) {//偶数次访问点u

cnt[a[u]]--;//权值a[u]数量-1

if(!cnt[a[u]]) dif--;//权值a[u]为0，不同点权数-1

}

else{//奇数次访问点u

if(!cnt[a[u]]) dif++;//原来权值a[u]为0，不同点权数+1

cnt[a[u]]++;//权值a[u]数量-1

}

in[u] ^= 1;//改变访问点u次数的奇偶性

}

void xorpath(int u, int v) {//计算u~v不同点值

if(dep[u] < dep[v]) swap(u, v);

while(dep[u] > dep[v]) { xorvertex(u); u = anc[u][0]; }//不同深度

while(u != v) {//未达lca

xorvertex(u); xorvertex(v);

u = anc[u][0]; v = anc[v][0];

}

}

int ans[maxq];

int main(){

scanf("%d%d", &n, &m);

for(int i = 1; i <= n; i++) {

scanf("%d", a + i);

b[i] = a[i];

}

sort(b + 1, b + 1 + n);

tot = unique(b + 1, b + 1 + n) - b - 1;

for(int i = 1; i <= n; i++)

a[i] = lower\_bound(b + 1, b + 1 + tot, a[i]) - b;

ecnt = 0;

memset(head, -1, sizeof(head));

for(int i = 1; i < n; i++) {

int u, v; scanf("%d%d", &u, &v);

AddEdge(u, v); AddEdge(v, u);

}

sz = (int)sqrt(n);

dfs(1);

while(top) group[S[top--]] = blocks;

preprocess();

for(int i = 1; i <= m; i++) {

scanf("%d%d", &q[i].u, &q[i].v);

q[i].id = i;

if(q[i].u > q[i].v) swap(q[i].u, q[i].v);

}

sort(q + 1, q + 1 + m);

int u = 1, v = 1;//u,v为上次询问的起始点

for(int i = 1; i <= m; i++) {

xorpath(u, q[i].u);

xorpath(v, q[i].v);

u = q[i].u, v = q[i].v;//更新u,v的值。

int lca = LCA(u, v);

ans[q[i].id] = dif;

if(!cnt[a[lca]]) ans[q[i].id]++;

}

for(int i = 1; i <= m; i++) printf("%d\n", ans[i]);

return 0;

}

题意：给一个树图，每个点的点权（比如颜色编号），m个询问，每个询问是一个区间[a,b]，图中两点之间唯一路径上有多少个不同点权（即多少种颜色）。n<40000，m<100000。

思路：无意中看到树上莫队，只是拿来练练，没有想到这题的难点不在于树上莫队，而是判断LCA是否在两点之间的路径上的问题。耗时1天。

　　树上莫队的搞法就是：

　　（1）DFS一次，对树进行分块，分成sqrt(n)块，每个点属于一个块。并记录每个点的DFS序。

　　（2）将m个询问区间用所属块号作为第一关键字，DFS序作为第二关键字进行排序。

　　（3）转移都是差不多的，靠具体问题分析转移方式。

　　这题的搞法：

　　（1）点权可能过大，但是只有4万点权，所以映射到1~4w进行处理。

　　（2）DFS对树进行分块，标记dfs序，记录每个点的深度（倍增用）。

　　（3）LCA预处理，以及求LCA的函数。

　　（4）对m个询问进行排序。

　　（5）莫队开始搞起。

　　如何转移（抄别人的）：

　　（1）定义S(u,v)为u−v路径上的顶点集合，root表示根节点。

　　（2）S(u,v)=S(root,u)△S(root,v)△lca(u,v) (△表示集合中的对称差，相当于xor)

　　（3）定义T(u,v)=S(root,u)△S(root,v)，我们先不管lca的事情

　　（4）如果我们从u−v的路径变成u−v′的路径的话对答案有什么影响呢？

（5）根据定义我们可以得到T(u,v′)=S(root,u)△S(root,v′)

　（6）T(u,v)△T(u,v′)=****S(root,u)△S(root,v)****△****S(root,u)△S(root,v′)****=S(root,v)△S(root,v′)=T(v,v′)。

　　也就是说，(a,b)要转移到(a,c)的话，将a点固定不动，b点逐个点走到c点就行了，肯定是将b走到Lca，c也走到Lca就行了。假设(a,b)路径上的点已经作了标记，只需要在走的时候，(b,c)路径上有标记的点就去掉，无标记的就加标记。

　　这样的做法理论上很容易理解，想起来不简单。因为当你走过a,b,c其中两个点的Lca时，你得想清楚这个Lca究竟要不要，而一共有3个Lca可以玩。当然如果某个点在(a,c)路径上，那么肯定是要的，所以任务就是判断这3个Lca是否在这段路径上。这才是难点！

　　虽然有很多种情况，但归类为3种：（1）单纯缩短 （2）单纯伸长 （3）先缩短+再伸长。

　　只要分这3类处理就OK了。而LCA(a,c)肯定是要的，这就可以利用了，判断LCA(b,c)是否在(a,c)上，可以根据从c走到LCA(b,c)遍历的点的顺序，是先到达LCA(a,c)还是LCA(b,c)？如果先到达LCA(b,c)，后达LCA(a,c)，则LCA(b,c)就要，否则就不要。同理这样判断LCA(a,b)要不要。

#include <bits/stdc++.h>

#define pii pair<int,int>

#define INF 0x3f3f3f3f

#define LL long long

using namespace std;

const int N=40010;

struct Que

{

int L, R, pos, ans;

Que(){};

Que(int L,int R,int pos):L(L),R(R),pos(pos){ans=0;};

};

int belongto[N], dfn[N], pre[N], stac[N], w[N];

int depth[N], anc[N][32], inp[N], cnt, bit[101000];

int n, m, block, block\_cnt, stac\_top, dfn\_clock, up;

vector<int> vect[N];

vector<Que> que;

void add\_edge(int from,int to)

{

vect[from].push\_back(to);

vect[to].push\_back(from);

}

inline int cmp(Que a,Que b) //第一关键字：块。第二关键字：DFS时间戳。

{

if(belongto[a.L]==belongto[b.L]) return dfn[a.R]<dfn[b.R];

return belongto[a.L] < belongto[b.L];

}

inline int cmp1(Que a,Que b){return a.pos<b.pos;}

void DFS(int x) //树的分块。

{

dfn[x]=++dfn\_clock; //dfs序

int cur=stac\_top;

for(int i=0; i<vect[x].size(); i++)

{

int t=vect[x][i];

if(!dfn[t])

{

pre[t]=x;

depth[t]=depth[x]+1; //深度

DFS(t);

if(stac\_top-cur >= block) //够block个就组

{

block\_cnt++;

while(--stac\_top>=cur)

{

int p=stac[stac\_top];

belongto[p]=block\_cnt;

}

stac\_top++; //栈顶必须为空

}

}

}

stac[stac\_top++]=x;//栈

}

void pre\_lca(int n) //倍增

{

memset(anc, 0, sizeof(anc));

for(int i=1; i<=n; i++) anc[i][0]=pre[i]; //0是他们自己的父亲。

for(int k=1; (1<<k)<=n; k++) //第2的k次方个父亲。

{

for(int i=1; i<=n; i++)

{

if(anc[i][k-1])

{

int a=anc[i][k-1];

anc[i][k]=anc[a][k-1];

}

}

}

}

int LCA(int a,int b)

{

//提到同层

if(depth[a]<depth[b]) swap(a, b);

int log;

for(log=1; (1<<log)<=depth[a]; log++ );

log--;

for(int i=log; i>=0; i-- ) //一定要从大到小

if( depth[a]-(1<<i)>=depth[b] )

a=anc[a][i];

if(a==b) return a; //b就是lca

for(int i=log; i>=0; i--) //同时往上提。

{

if(anc[a][i] && anc[a][i]!=anc[b][i] )

{

a=anc[a][i];

b=anc[b][i];

}

}

return pre[a];

}

void deal(int x,int d) //将值异或一下。

{

if(d==1) //加上1个

{

if(bit[x]) bit[x]+=1;

else bit[x]=1,cnt++;

}

else //减去1个

{

bit[x]--;

if(bit[x]==0) cnt--;

}

}

void backtofar(int ed,int lca) //但是lca不碰

{

while( ed!=lca )

{

inp[ed]\*=-1;

deal(w[ed], inp[ed]);

ed=pre[ed];

}

}

void biyao(int x,int b) //保证x在inpath上的状态必为b

{

if(inp[x]!=b)

{

inp[x]\*=-1;

deal(w[x], inp[x]);

}

}

int update(int L,int s,int e,int lca,int lca2,int lca3) //将s转移到e

{

if(inp[e]==1) //第一种:缩短

{

backtofar(s, lca);

backtofar(e, lca);

biyao(s, -1);

biyao(lca, -1); //s和e的LCA一定不要，若L到e经过了，那就肯定是他们的LCA，再补回来。

}

else //其他两种

{

int ed, s\_e=INF,L\_s=INF, L\_e=INF, Clock=1; //靠flag判断是：（1）伸长 （2）缩短+伸长

ed=s;

while( ed!=lca ) //lca先不碰

{

if(ed==lca) s\_e=Clock;

if(ed==lca3) L\_s=Clock;

inp[ed]\*=-1; //取反

deal(w[ed], inp[ed]);

ed=pre[ed];

Clock++;

}

ed=e;

int s\_e1=INF;

while( ed!=lca ) //lca先不碰

{

if(ed==lca) s\_e1=Clock;

if(ed==lca2) L\_e=Clock;

inp[ed]\*=-1; //取反

deal(w[ed], inp[ed]);

ed=pre[ed];

Clock++;

}

if(ed==lca) s\_e1=Clock;

if(ed==lca2) L\_e=Clock;

if(L\_s>=s\_e) biyao(lca3, -1);

if(s\_e1>=L\_e) biyao(lca, -1);

if(L\_s<s\_e) biyao(lca3, 1);

if(s\_e1<L\_e) biyao(lca, 1);

}

biyao(L, 1);

biyao(e, 1);

biyao(lca2, 1);

return cnt;

}

void cal(int n,int m)

{

for(int i=1; i<=n; i++) inp[i]=-1; //初始，无妨问过.

DFS(1); //分块

while(stac\_top>=0) //可能有余下的点，另开新块

{

int p=stac[stac\_top--];

belongto[p]=block\_cnt;

}

pre\_lca(n); //倍增处理lca

sort(que.begin(), que.end(), cmp);

int ans=0, L=1, R=1; //先将L和R随便弄到一个区间上，比如(1,1)。

cnt=inp[1]=bit[w[1]]=1;

for(int i=0; i<que.size(); i++) //莫队

{

if(R!=que[i].R) ans=update(L, R, que[i].R, LCA(R, que[i].R), LCA(L,que[i].R), LCA(L,R) ); //左

R=que[i].R;

if(L!=que[i].L) ans=update(R, L, que[i].L, LCA(L, que[i].L), LCA(R,que[i].L), LCA(L,R) ); //右

L=que[i].L;

que[i].ans=ans;

}

sort(que.begin(), que.end(), cmp1); //输出

for(int i=0; i<que.size(); i++) printf("%d\n", que[i].ans);

}

map<int,int> mapp;

void init()

{

block=sqrt(n);

dfn\_clock=block\_cnt=stac\_top=up=0;

mapp.clear();

que.clear();

for(int i=0; i<=n; i++) vect[i].clear();

memset(bit, 0, sizeof(bit));

memset(dfn, 0, sizeof(dfn));

memset(pre, 0, sizeof(pre));

memset(depth, 0, sizeof(depth));

}

int main()

{

freopen("input.txt", "r", stdin);

int a, b, t;

while(cin>>n>>m)

{

init();

for(int i=1; i<=n; i++)

{

scanf("%d", &t); //点权

if(mapp[t]==0) mapp[t]=++up; //映射为小一点的值。

w[i]=mapp[t];

}

for(int i=1; i<n; i++) //树边

{

scanf("%d%d",&a,&b);

add\_edge(a, b);

}

for(int i=0; i<m; i++) //询问

{

scanf("%d%d",&a,&b);

que.push\_back(Que(a,b,i));

}

cal(n,m);

}

return 0;

}