复习：前缀和

1 对于序列A[i] 的前缀和S[i]计算公式如下：

$$S[i] = \sum_{j=1}^{i} A[j]$$

在O（n）的时间复杂度内递推得到答案S数组

2 支持O(1)完成指定区间求和：

$$\text{sum}[l, r] = S[r] - S[l-1]$$

复习：差分

1 给定序列A[i]的任意相邻2数的差值

$$B[1] = A[1], \quad B[i] = A[i] - A[i-1] \ \ (2 \le i \le n)$$

在O（n）的时间复杂度内递推得到答案B数组

2 支持O(1)完成指定区间修改：（区间修改变单点修改）

对区间[left,right] +k （k为实数）：

B[left]+k , B[right+1] –k

信息学

树上差分

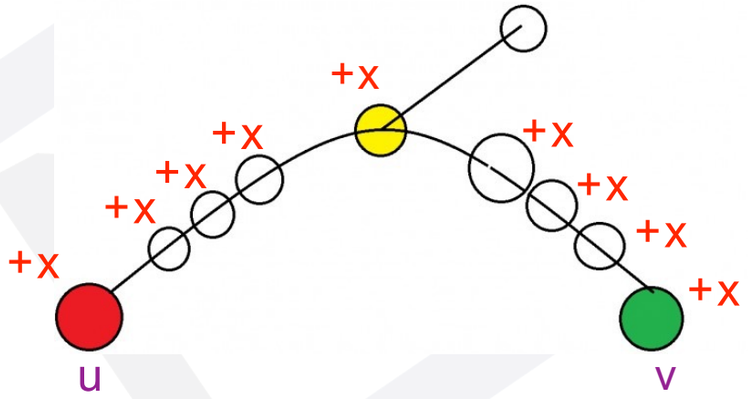
复习：前缀和与差分关系

前缀和与差分是一对互逆运算

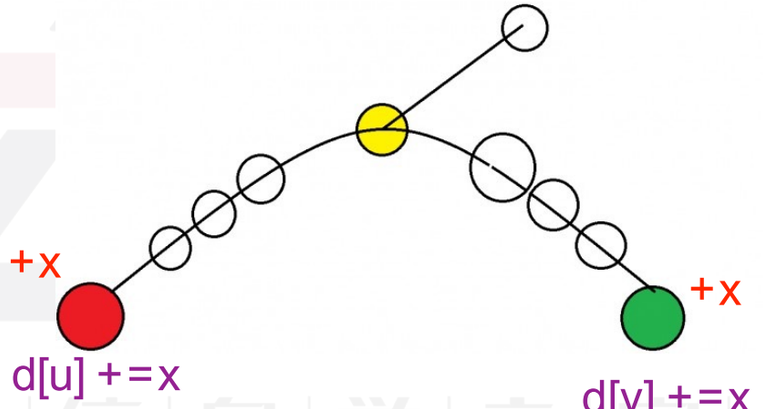
再对修改后的差分数组求前缀和可得修改后的数组A

新问题：链上节点整体修改 O(1)

如下图所示：

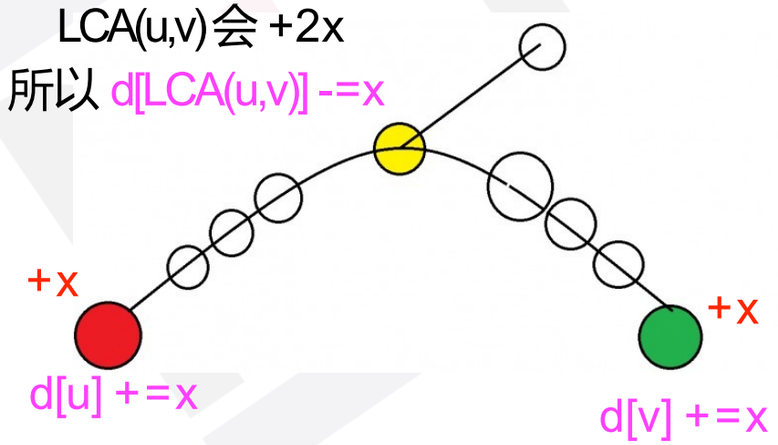


用数组d
记录修改



如何根据d数组还原修改？

问题分析：



如何根据d数组复原修改信息？

某个点的最终权值
=
该点原始权值
+
该点代表的子树节点的差分数组之和。

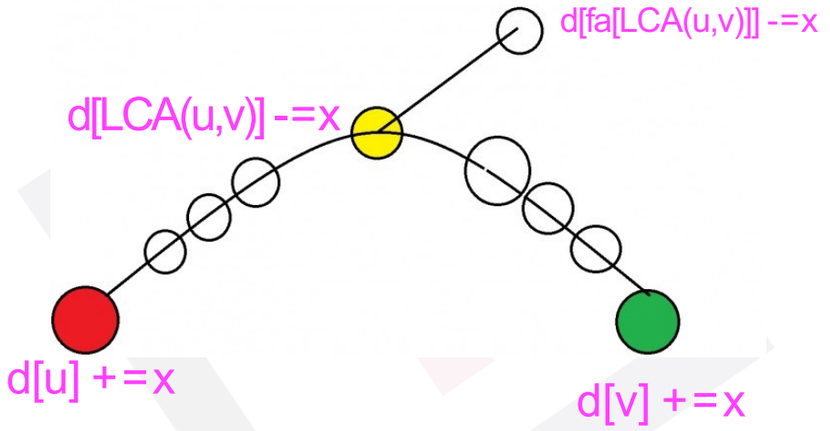
Dfs从子节点向上更新

（这样增加的x就可以向上传递到每个节点上）

所以见左图

这里的d数组，就是树的差分数组

新知：树上差分（点差分）



问题：（多次操作） 把u和v之间路径上所有点权+x

差分：

```
d[u]+=x; d[v]+=x;
d[ lca(u,v) ]-=x; d[ fa[lca(u,v)] ]-=x;
```

新知：树上差分（点差分）

如何求前缀和？

结点的最终权值=该结点所在子树的所有差分数组和+该结点原始值。

Dfs回溯求和即可。

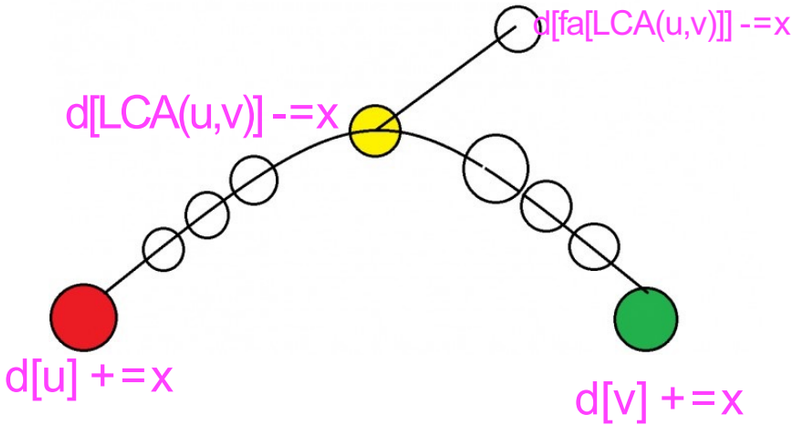
小结：树上差分（点差分）

1 问题：给定一棵树T，指定2点之间路径进行整体修改。（例如：路径上所有点+k，k为实数）

1. 线性区间操作==树上路径操作
2. 前缀和== 子树和

2 解释1.2：树上差分数组前缀和：

某个点的最终权值 = 该点原始点权 + 该点代表的子树内节点差分数组之和。



A. 问题：

（多次操作）把u和v之间路径上所有点权+x

B. 差分：

```
d[u]+=x; d[v]+=x;
d[ lca(u,v) ]-=x; d[ fa[lca(u,v)] ]-=x;
```

C. 求和：

dfs回溯求和。

例 ：USACO15DEC MAX FLOW P

FJ给他的牛棚的N(2≤N≤50,000)个隔间之间安装了N-1根管道，隔间编号从1到N。所有隔间都被管道连通了。

FJ有K(1≤K≤100,000)条运输牛奶的路线，第i条路线从隔间si运输到隔间ti。一条运输路线会给它的两个端点处的隔间以及中间途径的所有隔间带来一个单位的运输压力，你需要计算压力最大的隔间的压力是多少。

例 ：USACO15DEC MAX FLOW P

FJ给他的牛棚的N(2≤N≤50,000)个隔间之间安装了N-1根管道，隔间编号从1到N。所有隔间都被管道连通了。
FJ有K(1≤K≤100,000)条运输牛奶的路线，第i条路线从隔间si运输到隔间ti。一条运输路线会给它的两个端点处的隔间以及中间途径的所有隔间带来一个单位的运输压力，你需要计算压力最大的隔间的压力是多少。

分析：

该算法复杂度至少O(n logn)实现。最好O(n)

树上路径修改点值，只涉及到点值。不涉及到中途的查询。

板子题

例：伪代码

思考一下开搞过程：

1. 建树
2. 预处理
    1. LCA倍增数组
    2. LCA的父亲数组fa[][]
    3. LCA的深度数组depth
    4. 差分数组diff
3. 读入询问修改差分数组（求LCA）
4. dfs求节点值

例：伪代码

思考一下开搞过程：

1. 建树
2. 预处理
   1. LCA倍增数组
   2. LCA的父亲数组fa[][]
   3. LCA的深度数组depth
   4. 差分数组diff
3. 读入询问修改差分数组（求LCA）
4. dfs求节点值

```
#include<iostream>
#include<cmath>
using namespace std;
#define maxn 50010
#define ll long long
struct Node{
    int to,next;
};
Node edge[maxn<<2]; //链式前向星要多开几倍数组
int head[maxn<<2],diff[maxn],n,m,depth[maxn],fa[maxn][30],ans,num;
inline int read(){ //快读 int
    s=0;
    char c=getchar();
    while (c<'0' || c>'9') c=getchar();
    while (c>='0' && c<='9') {s=s*10+c-'0';c=getchar();} //注意{}  return
    s;
}
//链式前向星
inline void add(int x,int  y){edge[++num].to=y;edge[num].next=head[x];head[x]=num;}
```

例：伪代码

思考一下开搞过程：

1. 建树
2. 预处理
   1. LCA倍增数组
   2. LCA的父亲数组fa[][]
   3. LCA的深度数组depth
   4. 差分数组diff
3. 读入询问修改差分数组（求LCA）
4. dfs求节点值

```
//接下来是初始化
inline void init(int u,int fath){ //fath是u的爸爸
    depth[u]=depth[fath]+1;          //预处理深度数组
    fa[u][0]=fath;                   //记录fa信息 表示u节点上面的第2^j次方的爸爸
    for (int i=0;fa[u][i];++i)  //类似区间合并的思想
        fa[u][i+1]=fa[fa[u][i]][i];
    for (int i=head[u];i;i=edge[i].next){ //拓展到其他儿子节点。 int
        e=edge[i].to;
        if (e!=fath) init(e,u); //加个if 避免反复横跳
    }
}
```

例：伪代码

思考一下开搞过程：

1. 建树
2. 预处理
    1. LCA倍增数组
    2. LCA的父亲数组fa[][]
    3. LCA的深度数组depth
    4. 差分数组diff
3. 读入询问修改差分数组（求LCA）
4. dfs求节点值

```
//倍增求LCA，当然你可以tarjian求LCA 干掉这个log的复杂度。
inline int Lca(int u,int v){
    if (depth[u]>depth[v]) swap(u,v);
    for (int i=20;i>=0;--i)
        if (depth[u]<=depth[v]-(1<<i)) v=fa[v][i];
    if (u==v) return u;
    for (int i=20;i>=0;--i)
        if (fa[u][i]!=fa[v][i]) {u=fa[u][i];v=fa[v][i];}
    return fa[u][0];
}
/dfs求子树和

    for (int i=head[u];i;i=edge[i].next){
        int e=edge[i].to;
        if (e==fath) continue;
        dfs(e,u);
        diff[u]+=diff[e]; //回溯到时候求答案即可。
    }
    ans=max(ans,diff[u]);
}

int main(){
    n=read();m=read();
    int x,y;
    for (int i=1;i<n;++i){ //建树
        x=read();y=read();
        add(x,y); add(y,x);
    }
    init(1,0); //初始化
    for (int i=1; i<=m; ++i){ //处理m个区间修改（区间变单点）
        x=read();y=read();
        int lca=Lca(x,y);
        ++diff[x];++diff[y];--diff[lca];--diff[fa[lca][0]]; //树上差分
    }

    //dfs求子树前缀和，顺便根据题意打个擂台求答案
    dfs(1,0);//节点从1开始编号，规定0号节点是根节点的爸爸（方便程序运行）
    printf("%d\n",ans);
    return 0;
}
```

例：伪代码

思考一下开搞过程：

1. 建树
2. 预处理
    1. LCA倍增数组
    2. LCA的父亲数组fa[][]
    3. LCA的深度数组depth
    4. 差分数组diff
3. 读入询问修改差分数组（求LCA）
4. dfs求节点值

```
//dfs求子树和
inline void dfs(int u,int fath){
    for (int i=head[u];i;i=edge[i].next){
        int e=edge[i].to;
        if (e==fath) continue;
        dfs(e,u);
        diff[u]+=diff[e]; //回溯到时候求答案即可。
    }
    ans=max(ans,diff[u]);
}

int main(){
    n=read();m=read();
    int x,y;
    for (int i=1;i<n;++i){ //建树
        x=read();y=read();
        add(x,y); add(y,x);
    }
    init(1,0); //初始化
    for (int i=1; i<=m; ++i){ //处理m个区间修改（区间变单点）
        x=read();y=read();
        int lca=Lca(x,y);
        ++diff[x];++diff[y];--diff[lca];--diff[fa[lca][0]]; //树上差分
    }

    //dfs求子树前缀和，顺便根据题意打个擂台求答案
    dfs(1,0);//节点从1开始编号，规定0号节点是根节点的爸爸（方便程序运行）
    printf("%d\n",ans);
    return 0;
}
```

例：代码


```cpp
#include<iostream>
#include<cmath>
using namespace std;
#define maxn 50010
#define ll long long
struct Node{
    int to,next;
};
Node edge[maxn<<2]; //链式前向星要多开几倍数组
int head[maxn<<2],diff[maxn],n,m,depth[maxn],fa[maxn][30],ans,num;

inline int read(){ //快读
    int s=0;
    char c=getchar();
    while (c<'0' || c>'9') c=getchar();
    while (c>='0' && c<='9') {s=s*10+c-'0';c=getchar();} //注意{}
    return s;
}
//链式前向星
inline void add(int x,int y){edge[++num].to=y;edge[num].next=head[x];head[x]=num;}
//接下来是初始化
inline void init(int u,int fath){ //fath是u的爸爸
    depth[u]=depth[fath]+1;     //预处理深度数组
    fa[u][0]=fath;              //记录fa信息 表示u节点上面的第2^j次方的爸爸
    for (int i=0;fa[u][i];++i)  //类似区间合并的思想
        fa[u][i+1]=fa[fa[u][i]][i];
    for (int i=head[u];i;i=edge[i].next){ //拓展到其他儿子节点。
        int e=edge[i].to;
        if (e!=fath) init(e,u); //加个if 避免反复横跳
    }
}
//倍增求LCA，当然你可以tarjian求LCA 干掉这个log的复杂度。
inline int Lca(int u,int v){
    if (depth[u]>depth[v]) swap(u,v);
    for (int i=20;i>=0;--i)
        if (depth[u]<=depth[v]-(1<<i)) v=fa[v][i];
    if (u==v) return u;
    for (int i=20;i>=0;--i)
        if (fa[u][i]!=fa[v][i]) {u=fa[u][i];v=fa[v][i];}
    return fa[u][0];
}
```

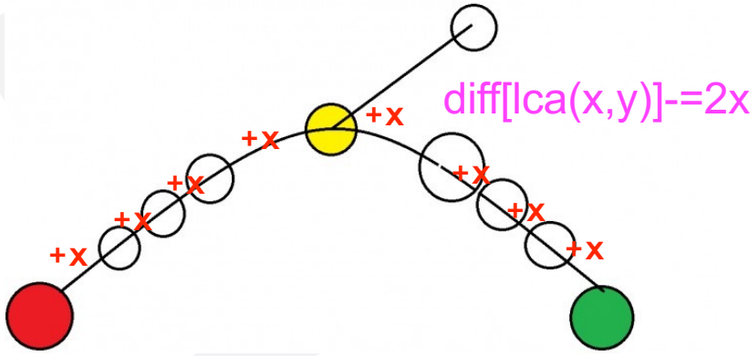
```cpp
//dfs求子树和
inline void dfs(int u,int fath){
    for (int i=head[u];i;i=edge[i].next){
        int e=edge[i].to;
        if (e==fath) continue;
        dfs(e,u);
        diff[u]+=diff[e]; //回溯到时候求答案即可。
    }
    ans=max(ans,diff[u]);
}

int main(){
    n=read();m=read();
    int x,y;
    for (int i=1;i<n;++i){ //建树
        x=read();y=read();
        add(x,y); add(y,x);
    }
    init(1,0); //初始化
    for (int i=1; i<=m; ++i){ //处理m个区间修改（区间变单点）
        x=read();y=read();
        int lca=Lca(x,y);
        ++diff[x];++diff[y];--diff[lca];--diff[fa[lca][0]]; //树上差分
    }

    //dfs求子树前缀和，顺便根据题意打个擂台求答案
    dfs(1,0);//节点从1开始编号，规定0号节点是根节点的爸爸（方便程序运行）
    printf("%d\n",ans);
    return 0;
}
```

新知变形：树上差分（边差分）

修改对象变成了边。如何处理呢？



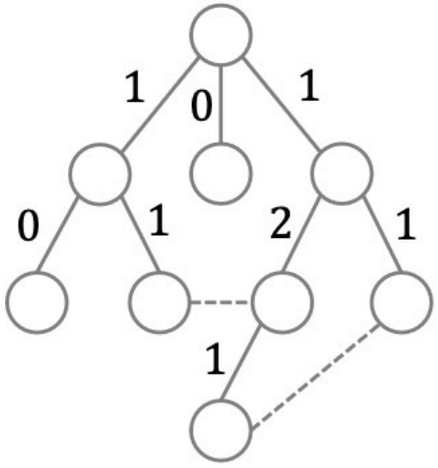
例： 暗之连锁

一个无向图，有 N（$N \le 10^5$）个节点和两类边:

一类称为“主要边”，共 $N-1$ 条，无向图的任意两个节点之间都 存在一条只由主要边构成的路径;

另一类称为“附加边”，共 M（$M \le 2\times10^5$）条。

现可以选择删除一条“主要边”和一条“附加边”，使得图分为不联通的两部分，求方案数量。

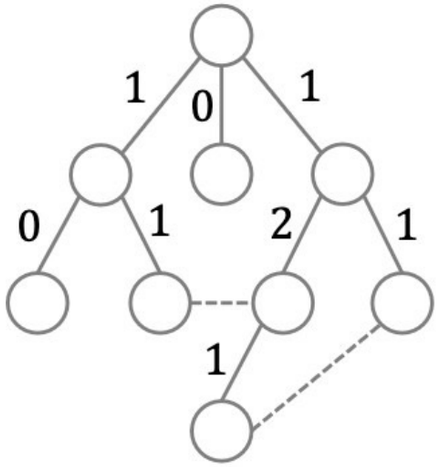


实线：主要边（树边）

虚线：附加边（非树边）

例： 暗之连锁

现可以选择删除一条“主要边”和一条“附加边”，使得图分为不联通的两部分，求方案数量。



实线：主要边（树边）

虚线：附加边（非树边）

主要边构成一棵树：

如果选择一条附加边(x,y)加入树中，构成环记作 $\delta(x, y)$

如果第一步选择删除$\delta(x, y)$ 中的某条主要边，那么下一步必须要删除(x,y)

可以理解为(x,y)把 环$\delta(x, y)$ 上的主要边都联通了一次。

有这样一个规律：

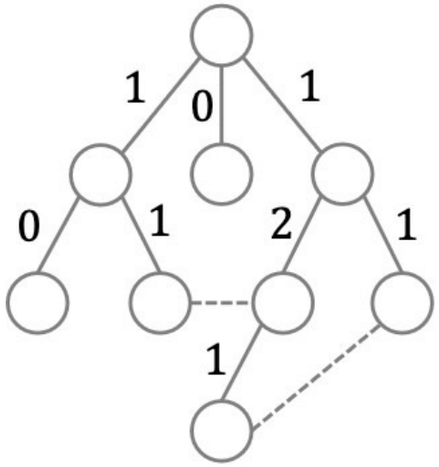
主要边被覆盖0次：任意删除一条附加边均可（方案贡献：+M）

主要边被覆盖1次：删除唯一的一条附加边（方案贡献：+1）

主要边被覆盖2次：无解（方案无贡献）

例： 暗之连锁

现可以选择删除一条“主要边”和一条“附加边”，使得图分为不联通的两部分，求方案数量。



实线：主要边（树边）

虚线：附加边（非树边）

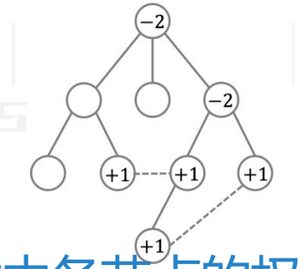
问题转化为：遍历所有主要边，

依次考察删除该边后删附加边的方案数贡献。

关键在于附加边对主要边的覆盖次数！

问题进一步转化为：求解 (u,v) 链上覆盖次数的问题。

树上差分！（边差分）



对路径统计前缀和，即“以 x 为根的子树中各节点的权值 之和”，就是 x 与它父节点之间的“主要边”被覆盖的次 数。 时间复杂度为 $O(N+M)$。

这里的LCA用tarjian处理，如果倍增就是O(NlogN + M)