https://www.luogu.com.cn/problem/P4719

https://www.luogu.com.cn/problem/P5024

给定一有点权的树，要求选择点使相邻两点中至少有一个被选择，且每次询问令点a,b必须选/不选，求满足每个询问要求时的最小代价

⇔ 给定一棵有点权的树，多次询问，每次修改点权或询问相邻两点中至少有一个被选择时最小代价

不考虑修改有：

$$dp[i][0]=\sum_{j\in son[i]}dp[j][1]$$
$$dp[i][1]=value[i]+\sum_{j\in son[i]}min(dp[j][0],dp[j][1])$$

注意到修改后只有点x到根路径上的dp值会发生变化，考虑树链剖分，使得在$O(log\ n)$的时间里求出重链头的dp值(带修) 在$O(1)$的时间通过轻边转移

如果单纯的一条链而没有其他多余的轻儿子 那么可以把方程写成$(min,+)$矩乘：

$$[\ dp[i][0]\quad dp[i][1]\ ]=[\ dp[sson[i]][0]\quad dp[sson[i]][1]\ ]\times\begin{bmatrix}inf & value[i]\\ 0 & value[i]\end{bmatrix}$$

每次修改某一个节点对应的矩阵，头部的dp矩阵就是 尾部的dp矩阵×所有矩阵顺序乘起来的矩阵，后者可以用线段树维护

```
struct node{
    int ma[2][2];
    node operator *(const node &x)const{
        node ret;
        ret.ma[0][0]=ret.ma[1][0]=ret.ma[0][1]=ret.ma[1][1]=inf;
        for(int i=0;i<2;i++){
            for(int j=0;j<2;j++){
                for(int k=0;k<2;k++)ret.ma[i][j]=min(ret.ma[i][j],ma[i]
[k]+x.ma[k][j]);
            }
        }
        return ret;
    }
```

```
}h[MAX<<2];

inline void pushup(int rt){h[rt]=h[rt<<1|1]*h[rt<<1];}
node query(int rt,int l,int r,int ql,int qr){
    if(ql<=l && r<=qr)return h[rt];
    int mid=(l+r)>>1;
    node ret;
    ret.ma[0][0]=ret.ma[1][1]=0;
    ret.ma[0][1]=ret.ma[1][0]=inf;
    if(mid<qr)ret=ret*query(rt<<1|1,mid+1,r,ql,qr);
    if(ql<=mid)ret=ret*query(rt<<1,l,mid,ql,qr);
    return ret;
}
void modify(int rt,int l,int r,int aim){
    if(l==r){
        h[rt].ma[0][0]=inf;
        h[rt].ma[0][1]=h[rt].ma[1][1]=valuel[l];
        h[rt].ma[1][0]=0;
        return ;
    }
    int mid=(l+r)>>1;
    if(aim<=mid)modify(rt<<1,l,mid,aim);
    else modify(rt<<1|1,mid+1,r,aim);
    pushup(rt);
}
```

考虑加入多余的轻儿子，换一种转移顺序：先将所有轻儿子都递归做完，用$ldp[i][0/1]$表示当前点不选/选时只考虑轻儿子时转移出来的答案，再转移单纯的一条重链：

$$ldp[i][0]=\sum_{j\in lson[i]} dp[j][1]$$
$$ldp[i][1]=value[i]+\sum_{j\in lson[i]} min(dp[j][0],dp[j][1])$$
$$dp[i][0]=dp[sson[i]][1]+ldp[i][0]$$
$$dp[i][1]=ldp[i][1]+min(dp[sson[i]][0],dp[sson[i]][1])$$

你单看这玩意发现并没有任何用，但是这意味着我们成功的将**树上**问题转化为了**序列**问题，这是树链剖分的基本思想；如果我们可以使用线段树维护重链上的dp值，就可以支持快速修改；把方程写成矩乘，就可以类似类似链的情况了：

$$[\,dp[i][0] \quad dp[i][1]\,] = [\,dp[son[i]][0] \quad dp[son[i]][1]\,] \times \begin{bmatrix} inf & ldp[i][1] \\ ldp[i][0] & ldp[i][1] \end{bmatrix}$$

具体来讲，当我们修改一个点的值的时候，我们在对应的重链的线段树上进行修改，此时会改变重链顶部的dp值，然后会改变重链顶的father的ldp值，然后又对应了线段树上的单点修改，此时又会改变另一个重链顶的dp值，

这样反复几次，我们就可以在$O(log^2n)$时间内完成单点修改

```
inline void solve(int x,int value) {     //value只是个指示，看到底是删除x的贡献
还是 增加x的贡献
    node bef;
    while (top[x] != 1) {
        bef = dp[bot[top[x]]] * query(1, 1, n, dfsn[top[x]],
dfsn[bot[top[x]]] - 1);
        x = fa[top[x]];
        ldp[x][0] += value * bef.ma[0][1];
        ldp[x][1] += value * min(bef.ma[0][0], bef.ma[0][1]);
        if (value == 1)modify(1, 1, n, dfsn[x]);
    }
}
//...
signed main() {
    //...
    for (int x, y, i = 1; i <= m; i++) {
        scanf("%d %d", &x, &y);
        solve(x, -1);    //删除x的贡献
        ldp[x][1] += y - a[x], dp[x].ma[0][1] += y - a[x];     //修改该点的权
值
        a[x] = y;
        modify(1, 1, n, dfsn[x]);   //修改所在重链
        solve(x, 1);   //增加x的贡献
        rec = dp[bot[1]] * query(1, 1, n, dfsn[1], dfsn[bot[1]] - 1);     //
求答案
        printf("%d\n", max(rec.ma[0][0], rec.ma[0][1]));
    }
    return 0;
}
```

完整代码：

```
#include <iostream>
```

```cpp
#include <cstdio>
#define int long long
using namespace std;
const int MAX=1e5+5;
const int inf=999999999999;
int nxt[MAX<<1],to[MAX<<1],head[MAX],cnt;
inline void add(int x,int y){
    ++cnt;
    nxt[cnt]=head[x],to[cnt]=y,head[x]=cnt;
    ++cnt;
    nxt[cnt]=head[y],to[cnt]=x,head[y]=cnt;
}
int a[MAX],n,m,sum;
int
top[MAX],bot[MAX],siz[MAX],son[MAX],fa[MAX],dfsn[MAX],back[MAX],cnt_dfsn;
void dfs1(int now,int father){
    siz[now]=1,fa[now]=father;
    for(int aim,i=head[now];i;i=nxt[i]){
        aim=to[i];
        if(aim!=father){
            dfs1(aim,now);
            siz[now]+=siz[aim];
            if(siz[son[now]]<siz[aim])son[now]=aim;
        }
    }
}
struct node{
    int ma[2][2];
    node operator *(const node &x)const{
        node ret;
        ret.ma[0][0]=ret.ma[1][0]=ret.ma[0][1]=ret.ma[1][1]=inf;
        for(int i=0;i<2;i++){
            for(int j=0;j<2;j++){
                for(int k=0;k<2;k++)ret.ma[i][j]=min(ret.ma[i][j],ma[i]
[k]+x.ma[k][j]);
            }
        }
        return ret;
    }
}h[MAX<<2],dp[MAX];
int ldp[MAX][2];
void dfs2(int now,int start){
    dfsn[now]=++cnt_dfsn,back[cnt_dfsn]=now;
```

```
    top[now]=start,bot[start]=now;
    ldp[now][1]=dp[now].ma[0][1]=a[now];

    if(son[now]){
        dfs2(son[now],start);
        dp[now].ma[0][0]+=dp[son[now]].ma[0][1];
        dp[now].ma[0][1]+=min(dp[son[now]].ma[0][0],dp[son[now]].ma[0][1]);
    }

    for(int aim,i=head[now];i;i=nxt[i]){
        aim=to[i];
        if(aim!=fa[now] && aim!=son[now]){
            dfs2(aim,aim);
            ldp[now][1]+=min(dp[aim].ma[0][0],dp[aim].ma[0][1]);
            ldp[now][0]+=dp[aim].ma[0][1];

            dp[now].ma[0][0]+=dp[aim].ma[0][1];
            dp[now].ma[0][1]+=min(dp[aim].ma[0][0],dp[aim].ma[0][1]);
        }
    }
}
inline void pushup(int rt){h[rt]=h[rt<<1|1]*h[rt<<1];}
node query(int rt,int l,int r,int ql,int qr){
    if(ql<=l && r<=qr)return h[rt];
    int mid=(l+r)>>1;
    node ret;
    ret.ma[0][0]=ret.ma[1][1]=0;
    ret.ma[0][1]=ret.ma[1][0]=inf;
    if(mid<qr)ret=ret*query(rt<<1|1,mid+1,r,ql,qr);
    if(ql<=mid)ret=ret*query(rt<<1,l,mid,ql,qr);
    return ret;
}
void modify(int rt,int l,int r,int aim){
    if(l==r){
        h[rt].ma[0][0]=inf;
        h[rt].ma[0][1]=h[rt].ma[1][1]=ldp[back[l]][1];
        h[rt].ma[1][0]=ldp[back[l]][0];
        return ;
    }
    int mid=(l+r)>>1;
    if(aim<=mid)modify(rt<<1,l,mid,aim);
    else modify(rt<<1|1,mid+1,r,aim);
    pushup(rt);
```

```
}
void build(int rt,int l,int r){
    if(l==r){
        h[rt].ma[0][0]=inf;
        h[rt].ma[0][1]=h[rt].ma[1][1]=ldp[back[l]][1];
        h[rt].ma[1][0]=ldp[back[l]][0];
        return ;
    }
    int mid=(l+r)>>1;
    build(rt<<1,l,mid);
    build(rt<<1|1,mid+1,r);
    pushup(rt);
}
inline void solve(int x,int value) {
    node bef;
    while (top[x] != 1) {
        bef = dp[bot[top[x]]] * query(1, 1, n, dfsn[top[x]],
dfsn[bot[top[x]]] - 1);
        x = fa[top[x]];
        ldp[x][0] += value * bef.ma[0][1];
        ldp[x][1] += value * min(bef.ma[0][0], bef.ma[0][1]);
        if (value == 1)modify(1, 1, n, dfsn[x]);
    }
}
char in[5];
signed main(){
    scanf("%lld %lld %s",&n,&m,in);
    for(int i=1;i<=n;i++)scanf("%lld",&a[i]),sum+=a[i];
    for(int x,y,i=1;i<n;i++)scanf("%lld %lld",&x,&y),add(x,y);
    dfs1(1,1),dfs2(1,1);
    build(1,1,n);
    node rec;
    //for(int i=1;i<=n;i++)printf("%llda%lld\n",ldp[i][0],ldp[i][1]);
    for(int cpy1,cpy2,op1,x,op2,y,ans,i=1;i<=m;i++){
        scanf("%lld %lld %lld %lld",&x,&op1,&y,&op2);
        cpy1=a[x],cpy2=a[y];
        solve(x,-1);
        if(op1==0)ldp[x][1]+=inf-a[x],dp[x].ma[0][1]+=inf-a[x],a[x]=inf;
        else ldp[x][1]+=-inf-a[x],dp[x].ma[0][1]+=-inf-a[x],a[x]=-inf;
        modify(1,1,n,dfsn[x]);
        solve(x,1);

        solve(y,-1);
```

```
        if(op2==0)ldp[y][1]+=inf-a[y],dp[y].ma[0][1]+=inf-a[y],a[y]=inf;
        else ldp[y][1]+=-inf-a[y],dp[y].ma[0][1]+=-inf-a[y],a[y]=-inf;
        modify(1,1,n,dfsn[y]);
        solve(y,1);
        //for(int j=1;j<=n;j++)printf("%lld%c%lld\n",ldp[j][0],'a'+i,ldp[j]
[1]);
        rec=dp[bot[1]]*query(1,1,n,dfsn[1],dfsn[bot[1]]-1);
        ans=min(rec.ma[0][0],rec.ma[0][1])+(op1?cpy1+inf:0)+(op2?
cpy2+inf:0);
        printf("%lld\n",ans>sum?-1:ans);

        solve(x,-1);
        if(op1==0)ldp[x][1]+=cpy1-inf,dp[x].ma[0][1]+=cpy1-inf,a[x]=cpy1;
        else ldp[x][1]+=cpy1+inf,dp[x].ma[0][1]+=cpy1+inf,a[x]=cpy1;
        modify(1,1,n,dfsn[x]);
        solve(x,1);

        solve(y,-1);
        if(op2==0)ldp[y][1]+=cpy2-inf,dp[y].ma[0][1]+=cpy2-inf,a[y]=cpy2;
        else ldp[y][1]+=cpy2+inf,dp[y].ma[0][1]+=cpy2+inf,a[y]=cpy2;
        modify(1,1,n,dfsn[y]);
        solve(y,1);
    }
    return 0;
}
```