动态DP

动态DP，是将DP转移方程写为矩阵形式后运用数据结构维护来减少更新信息以及操作次数的优秀算法。通常可以配合线段树/平衡树/分块等维护序列信息或者用来解决树上的DP问题，同时支持点权（边权）修改操作。对于每一次修改，都不需要重新计算树上每个结点的答案，而是更新路径上相关点的答案，以均摊$O(logn)$的复杂度完成修改。

前置知识

矩阵 / 树链剖分 /树形dp

广义矩阵乘法

定义广义矩阵乘法为$A\times B=C$为$C_{i,j}=max_{k=1}^{n}(A_{i,k}+B_{k,j})$

相当于普通的矩阵乘法中的乘变为加，加变为max操作。

假设我们知道DP方程

$$\begin{cases} f_{i,0}=g_{i,0}+max(f_{son_i,0},f_{son_i,1}) \\ f_{i,1}=g_{i,1}+f_{son_i,0} \end{cases}$$

就可以构造出矩阵形如

$$\begin{bmatrix} g_{i,0} & g_{i,0} \\ g_{i,1} & -inf \end{bmatrix}\times\begin{bmatrix} f_{son_i,0} \\ f_{son_i,1} \end{bmatrix}=\begin{bmatrix} f_{i,0} \\ f_{i,1} \end{bmatrix}$$

可以发现，每次修改操作我们只需要修改$g_{i,1}$和向上的重链即可。

具体思路

- DFS预处理求出$f_{i,0/1}$和$g_{i,0/1}$
- 对这棵树进行书剖（注意，因为我们对一个点进行询问需要计算从该点到该点所在的重链末尾的区间矩阵乘，所以对于每一个点记录End_i表示i所在的重链末尾编号），每一条重链建立线段树，线段树维护g矩阵和g矩阵区间乘积。

- 修改时首先修改$g_{i,1}$和线段树中i结点的矩阵，计算top_i矩阵的变化量，修改到fa_{top_i}矩阵
- 查询时就是1到其所在的重链末尾的区间乘，最后取一个max即可。

例题

luoguP4719 【模板】"动态 DP"&动态树分治

给定一颗树，点带点点权，每次操作将点x的权值修改为y，并输出这棵树当前的最大权独立集的权值大小。

```
#include<bits/stdc++.h>
#define inf 0x3f3f3f3f
#define int long long
#define ls (p<<1)
#define rs (p<<1|1)
#define mid ((l+r)>>1)
using namespace std;
const int N=1e5+7;

int n,m,val[N],vc[N],dp[N][4],flag=0;
char type[10];
vector<int>G[N];

int sz[N],dep[N],son[N],fa[N],dfn[N],idfn[N],bot[N],bel[N],idx=0;
void dfs1(int x){
    sz[x]=1;
    for(auto to:G[x]){
        if(to==fa[x]) continue;
        fa[to]=x;
        dep[to]=dep[x]+1;
        dfs1(to);
        sz[x]+=sz[to];
        if(sz[son[x]]<sz[to]) son[x]=to;
    }
}

void dfs2(int x,int tp){  //树链剖分
    dfn[x]=++idx;
    idfn[idx]=x;
    bel[x]=tp;
    if(son[x]) dfs2(son[x],tp),bot[x]=bot[son[x]];
    else bot[x]=x;
    for(auto to:G[x]){
        if(to==fa[x]||to==son[x]) continue;
        dfs2(to,to);
    }
}

void ddp(int x,int f){  //先预处理dp值
    dp[x][1]=val[x]; //选用该点的子树的最大独立集
    dp[x][0]=0; //不选用该点的子树的最大独立集
    if(vc[x]==-1&&vc[f]==-1) flag=1;
    for(auto to:G[x]){
        if(to==f) continue;
        ddp(to,x);
```

```
        dp[x][0]+=max(dp[to][0],dp[to][1]);
        dp[x][1]+=dp[to][0];
    }
}

struct Matrix{
    int s[2][2];
}tr[N<<2],tmp[N];

Matrix operator *(Matrix a,Matrix b){ //广义矩阵乘法
    Matrix res;
    res.s[0][0]=max(a.s[0][0]+b.s[0][0],a.s[0][1]+b.s[1][0]);
    res.s[0][1]=max(a.s[0][0]+b.s[0][1],a.s[0][1]+b.s[1][1]);
    res.s[1][0]=max(a.s[1][0]+b.s[0][0],a.s[1][1]+b.s[1][0]);
    res.s[1][1]=max(a.s[1][0]+b.s[0][1],a.s[1][1]+b.s[1][1]);
    return res;
}

void build(int p,int l,int r){ //按dfs序构建线段树
    if(l==r){
        int u=idfn[l],g0=0,g1=val[u];
        for(auto to:G[u]){
            if(to!=fa[u]&&to!=son[u]) g0+=max(dp[to][0],dp[to][1]),g1+=dp[to]
[0];
        }
        tmp[l]=tr[p]=(Matrix){g0,g0,g1,-inf};
        return;
    }
    build(ls,l,mid);build(rs,mid+1,r);
    tr[p]=tr[ls]*tr[rs];
}

void modify(int p,int l,int r,int pos){ //线段树单点修改
    if(l==r){
        tr[p]=tmp[l];
        return;
    }
    if(pos<=mid) modify(ls,l,mid,pos);
    else modify(rs,mid+1,r,pos);
    tr[p]=tr[ls]*tr[rs];
}

Matrix query(int p,int l,int r,int ql,int qr){ //返回区间所表示的矩阵
    if(ql==l&&r==qr) return tr[p];
    if(qr<=mid) return query(ls,l,mid,ql,qr);
    if(ql>mid) return query(rs,mid+1,r,ql,qr);
    return query(ls,l,mid,ql,mid)*query(rs,mid+1,r,mid+1,qr);
}

Matrix GetMat(int p){return query(1,1,n,dfn[bel[p]],dfn[bot[p]]);}

void modify(int u,int w){ //修改点权
    tmp[dfn[u]].s[1][0]+=w-val[u],val[u]=w;
    while(u){
        Matrix a=GetMat(bel[u]);
        modify(1,1,n,dfn[u]);
        Matrix b=GetMat(bel[u]);
        u=fa[bel[u]];
```

```
        if(!u) break;
        tmp[dfn[u]].s[0][1]=(tmp[dfn[u]].s[0][0]+=max(b.s[0][0],b.s[1][0])-
max(a.s[0][0],a.s[1][0]));
        tmp[dfn[u]].s[1][0]+=b.s[0][0]-a.s[0][0];
    }
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>val[i];
    int u,v,w;
    for(int i=1;i<n;i++){
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs1(1);
    dfs2(1,1);
    ddp(1,0);
    build(1,1,n);
    while(m--){
        cin>>u>>w;
        modify(u,w);
        Matrix ans=GetMat(1);
        cout<<max(ans.s[0][0],ans.s[1][0])<<"\n";
    }
    return 0;
}
```

luoguP5024 [NOIP2018 提高组] 保卫王国

给定一棵树，点带点权。规定相邻两个点必须至少取一个，并且每次询问规定两个点必须取或必须不取，求每次询问的最小权集。

最小全覆盖集=全集-最大权独立集。因此这道题可以转化为DDP的模板。

强制取点、不取点可以使用把权值改成正无穷或负无穷实现。

```
#include<bits/stdc++.h>
#define int long long
#define mp make_pair
#define pii pair<int,int>
using namespace std;
const int N=2e5+7,inf=1ll<<60;

int read(){ int x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return
x*f;}
int n,q,val[N];
int dp[N][2],g[N][2],fh[N][20][2][2];
string type;
set<pii>st;
```

```
vector<int>G[N];

int fa[N][20],dep[N];
void dfs1(int x,int f,int d){
    fa[x][0]=f,dep[x]=d;
    dp[x][1]=val[x];
    for(auto to:G[x]){
        if(to==fa[x][0]) continue;
        dfs1(to,x,d+1);
        dp[x][0]+=dp[to][1];
        dp[x][1]+=min(dp[to][0],dp[to][1]);
    }
}

void dfs2(int x){
    for(auto to:G[x]){
        if(to==fa[x][0]) continue;
        g[to][0]=g[x][1]+dp[x][1]-min(dp[to][0],dp[to][1]);
        g[to][1]=min(g[x][0]+dp[x][0]-dp[to][1],g[to][0]);
        dfs2(to);
    }
}

int solve(int x,int a,int y,int b){
    if(dep[x]<dep[y]) swap(x,y),swap(a,b);
    int tx[2]={inf,inf},ty[2]={inf,inf};
    int nx[2],ny[2];
    tx[a]=dp[x][a],ty[b]=dp[y][b];
    for(int i=19;~i;i--){
        if(dep[fa[x][i]]>=dep[y]){
            nx[0]=nx[1]=inf;
            for(int j=0;j<2;j++){
                for(int k=0;k<2;k++){
                    nx[j]=min(nx[j],tx[k]+fh[x][i][k][j]);
                }
            }
            tx[0]=nx[0],tx[1]=nx[1],x=fa[x][i];
        }
    }
    if(x==y) return tx[b]+g[x][b];
    for(int i=19;~i;i--){
        if(fa[x][i]!=fa[y][i]){
            nx[0]=nx[1]=ny[0]=ny[1]=inf;
            for(int j=0;j<2;j++){
                for(int k=0;k<2;k++){
                    nx[j]=min(nx[j],tx[k]+fh[x][i][k][j]);
                    ny[j]=min(ny[j],ty[k]+fh[y][i][k][j]);
                }
            }
            tx[0]=nx[0],tx[1]=nx[1],x=fa[x][i];
            ty[0]=ny[0],ty[1]=ny[1],y=fa[y][i];
        }
    }
    int lca=fa[x][0];
    int ans0=dp[lca][0]-dp[x][1]-dp[y][1]+tx[1]+ty[1]+g[lca][0];
    int ans1=dp[lca][1]-min(dp[x][0],dp[x][1])-min(dp[y][0],dp[y]
[1])+min(tx[0],tx[1])+min(ty[0],ty[1])+g[lca][1];
    return min(ans0,ans1);
```

```
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n>>q>>type;
    for(int i=1;i<=n;i++) cin>>val[i];
    for(int i=1,u,v;i<n;i++){
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
        st.insert(mp(u,v));
        st.insert(mp(v,u));
    }
    dfs1(1,0,1);
    dfs2(1);
    for(int i=1;i<=n;i++){
        fh[i][0][0][0]=inf;
        fh[i][0][0][1]=dp[fa[i][0]][1]-min(dp[i][0],dp[i][1]);
        fh[i][0][1][0]=dp[fa[i][0]][0]-dp[i][1];
        fh[i][0][1][1]=dp[fa[i][0]][1]-min(dp[i][0],dp[i][1]);
    }
    for(int j=1;j<=19;j++){
        for(int i=1;i<=n;i++){
            int tmp=fa[i][j-1];
            fa[i][j]=fa[tmp][j-1];
            for(int u=0;u<2;u++){
                for(int v=0;v<2;v++){
                    fh[i][j][u][v]=inf;
                    for(int w=0;w<2;w++){
                        fh[i][j][u][v]=min(fh[i][j][u][v],fh[i][j-1][u]
[w]+fh[tmp][j-1][w][v]);
                    }
                }
            }
        }
    }
    for(int i=1,a,b,x,y;i<=q;i++){
        cin>>a>>x>>b>>y;
        if(!x&&!y&&st.find(mp(a,b))!=st.end()){
            cout<<-1<<"\n";
            continue;
        }else cout<<solve(a,x,b,y)<<"\n";
    }
    return 0;
}
```

参考资料

https://www.luogu.com.cn/blog/zhoutb2333/solution-p5024
https://oi-wiki.org/dp/dynamic/

https://www.cnblogs.com/A2484337545/p/15139132.html

https://zhuanlan.zhihu.com/p/107709816