目录

1 单调栈 + 单调队列

1.1 单调栈

```
int s[N], top=0;
for(int i=1;i<=n;i++)
{
    while(top&&a[s[top]]>a[i]) top--;
    if(s[top]==a[i]) s[top]=i;//去重复操作，视情况而定
    s[++top]=i;
}
```

1.2 单调队列

```
int l=0,r=1;
int q[N];
for(int i=1;i<=n;i++)
{
    while(l<=r&&a[q[r]]>a[i]) r--;
    q[++r]=i;
    while(l<=r&&q[l]<i-m+1) l++;//判断条件看情况
    ans[i]=a[q[l]];
}
```

2 树状数组

2.1 lowbit 之和

```
/*
    HDU5975 有n个集合，第 i 个集合的数是[i-lowbit(i)+1,i-1]+i
    第一种方式是集合[a,b]共有多少个数=lowbit之和？(sol1)
    第二种方式是数字x在几个集合里面？(sol2)
    long long型注意！
*/
#define ll long long
ll lowbit(ll x){
    return x&(-x);
}
ll sum(ll x){           //计算[1,x]中所有数lowbit(k)之和
    ll ans=0ll;
    for(ll p=1ll;p<=x;p<<=1){
        ans+=(x/p-x/(p<<1))*p;
    }
```

```
    return ans;
}
ll sol1(ll x,ll y){     //计算[x,y]中所有树的lowbit(k)之和
    return sum(y)-sum(x-1ll);
}
ll sol2(ll x,ll n){
    ll res=0ll;
    while(x<=n){
        res++;
        x+=lowbit(x);
    }
    return res;
}
```

2.2 区间加减 + 区间和查询

```
#define LL long long
const int SIZE=10005;
ll T1[SIZE],T2[SIZE],T[SIZE];//T[]用于存结点值，用过一次即仍
int lowbit(int k){
    return k&-k;
}
void update(ll x,ll,N,ll w){//把x位置之后所有数的值+w
    for(ll i=x;i<=N;i+=lowbit(i)){
        T1[i]+=w;T2[i]+=w*(x-1);
    }
}
void range_update(ll l,ll r,ll v){//在[l,r]上加v
    update(l,v);update(r+1,-v);
}
ll sum(ll x){
    ll ans=0;
    for(ll i=x;i>0;i-=lowbit(i)){
        ans+=x*T1[i]-T2[i];
    }
    return ans;
}
ll range_ask(ll l,ll r){//返回[l,r]的和
    return sum(r)-sum(l-1);
}
void init(int N){
    for(int i=1;i<=N;i++){
        update(i,T[i]-T[i-1]);
    }
}
```

2.3 统计前后顺序不同数字对个数（三维偏序问题）

```cpp
#define ll long long
const int SIZE=200000+50;
ll arr[3][SIZE];//数据存放为[1,n]的范围
ll tree[SIZE];
int n;//n为每组数字个数
int lowbit(int k){
    return k&-k;
}
void add(int x,int k){
    while(x<=n){
        tree[x]+=k;
        x+=lowbit(x);
    }
}
ll sum(int x){
    ll ans=0;
    while(x!=0){
        ans+=tree[x];
        x-=lowbit(x);
    }
    return ans;
}
int pos[SIZE];
ll CountInversions(int x,int y){
    memset(tree,0,sizeof(tree));
    for(int i=1;i<=n;i++){
        pos[arr[x][i]]=i;
    }
    ll ans=0;
    for(int i=n;i;i--){
        ans+=sum(pos[arr[y][i]]);
        add(pos[arr[y][i]],1);
    }
    return ans;
}

//求三组数中有多少对数的前后顺序在三组数中都相同(三维偏序问题)
ll invers=(CountInversions(0,1)+CountInversions(1,2)+CountInversions(2,0))/2ll;
ll tot=((ll)n*(ll)(n-1))/2ll;//这里一定要加ll!!不然会爆
printf("%lld\n",tot-invers);
```

3 线段树

3.1 ZKW 线段树

3.1.1 开局

```cpp
for(M=1;M<=n;M<<=1);//获得层数M
```

```
for(int i=1;i<=M<<1;i++)//初始化
{
    T[i]=0;
}
for(int i=1,x;i<=n;i++)//建树
{
    scanf("%d",&x);
    modify(i,x);
}
```

3.1.2 单点修改 + 区间查询

```
void modify(int n,int v)//单点修改
{
    for(T[n+=M]+=v,n>>=1;n;n>>=1)
        T[n]=T[n+n]+T[n+n+1];
}
ll query(int l, int r)//区间查询和，可调为最大
{
    ll ans=0;
    for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
    {
        if(~l & 1) ans+=T[l^1];
        if(r & 1) ans+=T[r^1];
    }
    return ans;
}
```

3.1.3 单点修改 + 区间查询最大字段和

```
struct Node{
    ll pre,suf,max,sum;
}T[200000],null;
Node merge(Node l, Node r)
{
    Node res;
    res.sum=l.sum+r.sum;//区间和
    res.pre=max(l.pre,l.sum+r.pre);//最大前缀
    res.suf=max(r.suf,l.suf+r.sum);//最大
    res.max=max(l.max,r.max);//最大子段和
    res.max=max(res.max,l.suf+r.pre);
    return res;
}
void modify(int n,int v)//单点修改
{
    for(T[n+=M]={v,v,v,v},n>>=1;n;n>>=1)
        T[n]=merge(T[n+n],T[n+n+1]);
}
```

```
ll query(int l, int r)//查询
{
    Node resl(null),resr(null);
//  resl.pre=resl.suf=resl.sum=0;
//  resr.pre=resr.suf=resr.sum=0;
    resl.max=resr.max=-inf;
    for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
    {
        if(~l & 1) resl=merge(resl,T[l^1]);
        if(r & 1) resr=merge(T[r^1],resr);
    }
    return merge(resl,resr).max;
}
```

3.1.4 区间加减 + 单点查询

```
void add(int l,int r,int v)//区间加减
{
    for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
    {
        if(~l & 1) T[l^1]+=v;
        if(r & 1) T[r^1]+=v;
    }
}
ll query(int n)//单点查询
{
    ll ans=0;
    for(n+=M;n;n>>=1) ans+=T[n];
    return ans;
}
```

3.1.5 区间加减 + 区间最值查询（lazy 标记）

```
void add(int L,int R,int v)//区间修改
{
    L+=M-1,R+=M+1;
    for(int l=L,r=R;l^r^1;l>>=1,r>>=1)
    {
        if(~l & 1) lazy[l^1]+=v,T[l^1]+=v;
        if(r & 1) lazy[r^1]+=v,T[r^1]+=v;
    }
    for(int l=L>>1,r=R>>1;l;l>>=1,r>>=1)
    {
        T[l]=max(T[l+l],T[l+l+1])+lazy[l];
        T[r]=max(T[r+r],T[r+r+1])+lazy[r];
    }
}
int query(int l, int r)//区间查询
```

```
{
    int lmax=-inf,rmax=-inf;
    for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
    {
        if(lazy[l]&&lmax!=-inf) lmax+=lazy[l];
        if(lazy[r]&&rmax!=-inf) rmax+=lazy[r];
        if(~l & 1) lmax=max(lmax,T[l^1]);
        if(r & 1) rmax=max(rmax,T[r^1]);
    }
    for(;l;l>>=1,r>>=1)
    {
        lmax+=lazy[l],rmax+=lazy[r];
    }
    return max(lmax,rmax);
}
```

3.2 普通线段树

3.2.1 单点修改 + 区间查询

```
#define lson l, mid, rt << 1
#define rson mid + 1, r, rt << 1 | 1
const int SIZE = 5e5+50;
struct node{
    int l,r;int val;
}T[SIZE*3];
int arr[SIZE];// 用于存从[1,n]中的数组
void build(int l,int r,int rt){
    T[rt].l = l;T[rt].r = r;
    if(l == r){
        T[rt].val = arr[l];
        return ;
    }
    int mid = (T[rt].l + T[rt].r)>>1;
    build(lson); build(rson);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}
void update(int tar,int rt,int k){// 单点修改将tar改为k
    if(T[rt].l==T[rt].r){
        T[rt].val += k;return ;
    }
    int mid = (T[rt].l + T[rt].r)>>1;
    if(tar <= mid) update(tar,rt<<1,k);
    else update(tar,rt<<1|1,k);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}
int ans = 0;
void Find(int rt,int l,int r){
    if(T[rt].l>=l && T[rt].r<=r){
```

```
        ans += T[rt].val;return ;
    }
    if(T[rt<<1].r>=l) Find(rt<<1,l,r);
    if(T[rt<<1|1].l<=r) Find(rt<<1|1,l,r);
}
```

3.2.2 区间修改 + 区间查询

```
#define ll long long
const int SIZE = 1e5+5;
struct node{
    int l,r;int val;
}T[SIZE*3];
int lazy[SIZE*3];
int arr[SIZE];// arr数组用于记录[1,n]的数据
void build(int l,int r,int rt){
    T[rt].l = l;T[rt].r = r;
    if(l==r){
        T[rt].val = arr[l];return ;
    }
    int mid = (l+r)>>1;
    build(l,mid,rt<<1);
    build(mid+1,r,rt<<1|1);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}
// l,r为指针，L,R是要修改的区间[L,R],为整个区间加上x
void update(int L,int R,int x,int rt){
    if(L<=T[rt].l && R>=T[rt].r){
        T[rt].val += x*(T[rt].r-T[rt].l+1);
        lazy[rt] += x;
        return ;
    }
    int mid = (T[rt].l + T[rt].r)>>1;
    if(lazy[rt]){ // 下推标记更新一波
        T[rt<<1].val += lazy[rt]*(mid-T[rt].l+1);
        T[rt<<1|1].val += lazy[rt]*(T[rt].r-mid);
        lazy[rt<<1] += lazy[rt];
        lazy[rt<<1|1] += lazy[rt];
        lazy[rt] = 0;
    }
    if(L <= mid)update(L,R,x,rt<<1);
    if(R >mid)update(L,R,x,rt<<1|1);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}

ll query(int L,int R,int rt){
    // printf("rt=%d",rt);
    if(L<=T[rt].l && R>=T[rt].r)return T[rt].val;
    int mid = (T[rt].l + T[rt].r)>>1;
```

```
    if(lazy[rt]){
        T[rt<<1].val += lazy[rt]*(mid-T[rt].l+1);
        T[rt<<1|1].val += lazy[rt]*(T[rt].r-mid);
        lazy[rt<<1] += lazy[rt];
        lazy[rt<<1|1] += lazy[rt];
        lazy[rt] = 0;
    }
    ll sum=0;
    if(L <= mid)sum += query(L,R,rt<<1);
    if(R > mid)sum += query(L,R,rt<<1|1);
    return sum;
}
```

3.2.3 区间染色

```
const int SIZE=1e5+50;
const int col_SIZE=1e5+50;
bool col_vis[col_SIZE];
struct node{
    int l,r;int col;
}T[SIZE*3];

void build(int root,int l,int r){
    T[root].l = l;
    T[root].r = r;
    if(l == r)return ;
    int mid=(T[root].l + T[root].r)>>1;
    build(root<<1,l,mid);
    build((root<<1)+1,mid+1,r);
}
void update(int root,int l,int r,int col){
    if(T[root].l>=l && T[root].r<=r){
        T[root].col = col;return ;
    }
    else{
        if(T[root].col > 0){
            T[root<<1].col = T[root].col;
            T[(root<<1)+1].col = T[root].col;
            T[root].col = 0;
        }
        int mid = (T[root].l + T[root].r)>>1;
        if(l>mid){
            update((root<<1)+1,l,r,col);
        }
        else if(r<=mid){
            update(root<<1,l,r,col);
        }
        else{
            update(root<<1,l,mid,col);
```

```
            update((root<<1)+1,mid+1,r,col);
        }
    }
}

void Find(int root,int l,int r){
    if(T[root].l==0 || T[root].r==0)return ;
    if(T[root].col > 0 ){
        col_vis[T[root].col] = true;
    }
    else{
        int mid = (T[ root ].l + T[ root ].r ) >> 1;
        if(l>mid){
            Find((root<<1)+1,l,r);
        }
        else if(r<=mid){
            Find(root<<1,l,r);
        }
        else{
            Find(root<<1,l,mid);
            Find((root<<1)+1,mid+1,r);
        }
    }
}
void init(int N){//N为最大的颜色
    for(int i=0;i<=N;i++){
        col_vis[i]=0;
    }
}

int tot(int l,int r,int N){//统计[l,r]内颜色范围为[1,N]共有几种
    init(N);
    Find(1,l,r);
    int tot=0;
    for(int i=0;i<=N;i++){
        if(col_vis[i])tot++;
    }
    return tot;
}
```

3.2.4 区间修改 + 区间查询：矩阵

```
/*
    本模板需要配合矩阵快速幂食用qwq
*/
typedef long long ll;
int n;
mat A(2,2),B(2,2);
const int SIZE=1e5+50;
```

```
char str[SIZE];
struct node{
    int l,r;
    bool lazy=0;
}T[SIZE*3];
mat ma[SIZE*3][2];//0是正序，1是反序
void push_up(int root){
    ma[root][0]=ma[root<<1][0]*ma[(root<<1)|1][0];
    ma[root][1]=ma[root<<1][1]*ma[(root<<1)|1][1];
    return ;
}
void build(int root,int l,int r){
    T[root].l=l;T[root].r=r;
    if(l==r){
        if(str[l]=='A'){
            ma[root][0].n = A.n;ma[root][0].m = A.m;
            for(int i=0;i<A.n;i++)
            for(int j=0;j<A.m;j++)
                ma[root][0].v[i][j]=A.v[i][j];
            //————————————————————————————
            ma[root][1].n = B.n;ma[root][1].m = B.m;
            for(int i=0;i<B.n;i++)
            for(int j=0;j<B.m;j++)
                ma[root][1].v[i][j]=B.v[i][j];
        }
        else{
            ma[root][0].n = B.n;ma[root][0].m = B.m;
            for(int i=0;i<B.n;i++)
            for(int j=0;j<B.m;j++)
                ma[root][0].v[i][j]=B.v[i][j];
            //————————————————————————————
            ma[root][1].n = A.n;ma[root][1].m = A.m;
            for(int i=0;i<A.n;i++)
            for(int j=0;j<A.m;j++)
                ma[root][1].v[i][j]=A.v[i][j];
        }
        return ;
    }
    int mid=(T[root].l+T[root].r)>>1;
    build(root<<1,l,mid);
    build((root<<1)|1,mid+1,r);
    push_up(root);
    return ;
}

void push_down(int root){
    if(T[root].lazy){
        swap(ma[root<<1][0],ma[root<<1][1]);
        swap(ma[(root<<1)|1][0],ma[(root<<1)|1][1]);
        T[root<<1].lazy=!T[root<<1].lazy;
```

```
        T[(root<<1)|1].lazy=!T[(root<<1)|1].lazy;
        T[root].lazy=0;
    }
}
void update(int root,int l,int r){
    if(l<=T[root].l && T[root].r<=r){
        T[root].lazy = !T[root].lazy;
        swap(ma[root][0],ma[root][1]);
        return ;
    }
    push_down(root);
    int mid = (T[root].l + T[root].r)>>1;
    if(l <= mid)update(root<<1,l,r);
    if(r > mid)update((root<<1)|1,l,r);
    push_up(root);
}

mat tmp(2,2);// 对最后的答案矩阵进行修改
void find(int root,int l,int r){
    if(l <= T[root].l && T[root].r<=r){
        tmp = tmp*ma[root][0];
        return;
    }
    push_down(root);
    int mid = (T[root].l + T[root].r)>>1;
    if(l <= mid)find(root<<1,l,r);
    if(mid < r)find((root<<1)|1,l,r);
    return;
}
int main(){
    A.v[0][0]=1;A.v[0][1]=0;A.v[1][0]=1;A.v[1][1]=1;
    B.v[0][0]=1;B.v[0][1]=1;B.v[1][0]=0;B.v[1][1]=1;
    int q;scanf("%d%d",&n,&q);
    scanf("%s",str+1);
    build(1,1,n);
    while(q--){
        int op;scanf("%d",&op);
        if(op==1){
            int L,R;scanf("%d%d",&L,&R);
            update(1,L,R);
        }
        else{
            int L,R;ll AA,BB;scanf("%d%d%lld%lld",&L,&R,&AA,&BB);
            tmp.v[0][0]=1;
            tmp.v[0][1]=0;
            tmp.v[1][0]=0;
            tmp.v[1][1]=1;
            mat qwq(1,2);
            qwq.v[0][0]=AA%mod;qwq.v[0][1]=BB%mod;
            find(1,L,R);
```

```
            qwq = qwq*tmp;
            qwq.display();
        }
    }
}
```

4 普通平衡树 Treap

```
#define inf 0x3f3f3f3f
const int maxn = 1e6+5;
int ch[maxn][2];
int val[maxn],dat[maxn];
int sz[maxn],cnt[maxn];
int tot,root;
int New(int v){// 辅助函数
    val[++tot] = v;
    dat[tot] = rand();
    sz[tot] = 1;
    cnt[tot] = 1;
    return tot;
}
void pushup(int id){// 辅助函数
    sz[id] = sz[ch[id][0]] + sz[ch[id][1]] + cnt[id];
}
void build(){// 辅助函数
    root = New(-INF),ch[root][1] = New(INF);
    pushup(root);
}
void Rotate(int &id,int d){// 辅助函数
    int temp = ch[id][d ^ 1];
    ch[id][d ^ 1] = ch[temp][d];
    ch[temp][d] = id;
    id = temp;
    pushup(ch[id][d]),pushup(id);
}
void insert(int &id,int v){// 插入一个数值为v
    if(!id){
        id = New(v);return ;
    }
    if(v == val[id])cnt[id]++;
    else{
        int d = v < val[id] ? 0 : 1;
        insert(ch[id][d],v);
        if(dat[id] < dat[ch[id][d]])Rotate(id,d ^ 1);
    }
    pushup(id);
    }
void Remove(int &id,int v){// 删除一个指为v数(若有多个相同的数，因只删除一个)
```

```
    if(!id)return ;
    if(v == val[id]){
        if(cnt[id] > 1){cnt[id]--,pushup(id);return ;}
        if(ch[id][0] || ch[id][1]){
            if(!ch[id][1] || dat[ch[id][0]] > dat[ch[id][1]]){
                Rotate(id,1),Remove(ch[id][1],v);
                }
            else Rotate(id,0),Remove(ch[id][0],v);
            pushup(id);
        }
        else id = 0;
        return ;
    }
    v < val[id] ? Remove(ch[id][0],v) : Remove(ch[id][1],v);
    pushup(id);
}
int get_rank(int id,int v){//
    查询v数的排名(排名定义为比当前数小的数的个数+1。若有多个相同的数，因输出最小的排名)
    if(!id)return 0;
    if(v == val[id])return sz[ch[id][0]] + 1;
    else if(v < val[id])return get_rank(ch[id][0],v);
    else return sz[ch[id][0]] + cnt[id] + get_rank(ch[id][1],v);
}
int get_val(int id,int rank){// 查询排名为rank的数
    if(!id)return INF;
    if(rank <= sz[ch[id][0]])return get_val(ch[id][0],rank);
        else if(rank <= sz[ch[id][0]] + cnt[id])return val[id];
    else return get_val(ch[id][1],rank - sz[ch[id][0]] - cnt[id]);
}
int get_pre(int v){// 求v的前驱(前驱定义为小于v，且最大的数)
    int id = root,pre;
    while(id){
        if(val[id] < v)pre = val[id],id = ch[id][1];
        else id = ch[id][0];
    }
    return pre;
}
int get_next(int v){// 求v的后继(后继定义为大于v，且最小的数)
    int id = root,next;
    while(id){
        if(val[id] > v)next = val[id],id = ch[id][0];
        else id = ch[id][1];
    }
    return next;
}
int main(){
    build();int x;
    int shk;scanf("%d",&shk);
    while(shk--){
        int op;scanf("%d",&op);
```

```
        switch(op){
            case 1:scanf("%d",&x);insert(root,x);break;
            case 2:scanf("%d",&x);Remove(root,x);break;
            case 3:scanf("%d",&x);printf("%d\n",get_rank(root,x)-1);break;
            case 4:scanf("%d",&x);printf("%d\n",get_val(root,x+1));break;
            case 5:scanf("%d",&x);printf("%d\n",get_pre(x));break;
            case 6:scanf("%d",&x);printf("%d\n",get_next(x));break;
        }
    }
}
```

5 树链剖分

```
int sum[MAXM*4],add[MAXM*4];
int a[MAXM],n;
int idx;
int first[MAXM];
struct edge{
    int v,next;
}e[MAXM*2];
int f[MAXM],son[MAXM],size[MAXM],dfn[MAXM],dep[MAXM],top[MAXM],seq[MAXM];
int cnt;
int mod;
/*
f是节点的父亲,son是重儿子,size是以该节点为根的子树大小,dfn是给节点重新编上的序号.
seq是与dfn相反的数组,表示标到的这个号表示的原节点,top是目前节点所在链的顶端.
dep是结点的深度
*/

//————————————————加边和预处理————————————————————————
void eadd(int a,int b)
{
    e[idx].v = b;
    e[idx].next = first[a];
    first[a] = idx++;
}
void init()
{
    memset(first,-1,sizeof(first));
    idx = 1;
    cnt = 0;
}
void dfs1(int u,int fa,int depth)
{
    f[u] = fa; size[u] = 1; dep[u] = depth;
    int maxson = -1;
    for(int i = first[u];i != -1;i = e[i].next){
        int v = e[i].v;
```

```
        if(v == fa) continue;
        dfs1(v,u,depth+1);
        size[u] += size[v];
        if(size[v]>maxson) son[u] = v,maxson = size[v];
    }
}
void dfs2(int u,int t)
{
    top[u] = t;
    dfn[u] = ++cnt;
    seq[cnt] = a[u];
    if(!son[u]) return;
    dfs2(son[u],t);
    for(int i = first[u];i != -1;i = e[i].next){
        int v = e[i].v;
        if(v != son[u]&&v != f[u]) dfs2(v,v);
    }
}


//————————————————————线段树————————————————————
void pushup(int rt)
{
    sum[rt] = (sum[rt<<1]+sum[rt<<1|1])%mod;
}
void build(int l,int r,int rt)
{
    if(l == r){
        sum[rt] = seq[l]%mod;
        return;
    }
    int m = (l+r)>>1;
    build(l,m,rt<<1);
    build(m+1,r,rt<<1|1);
    pushup(rt);
}
void pushdown(int rt,int ln,int rn)
{
    if(add[rt]){
        add[rt<<1] = (add[rt<<1] + add[rt])%mod;
        add[rt<<1|1] = (add[rt<<1|1] + add[rt])%mod;
        sum[rt<<1] = (sum[rt<<1] + add[rt]*ln%mod)%mod;
        sum[rt<<1|1] = (sum[rt<<1|1] + add[rt]*rn%mod)%mod;
        add[rt] = 0;
    }
}
void update(int L,int R,int C,int l,int r,int rt)
{
    if(L <= l&&r <= R){
        sum[rt] = (sum[rt] + C*(r-l+1)%mod)%mod;
```

```
        add[rt] = (add[rt] + C)%mod;
        return;
    }
    int m = (l+r)>>1;
    pushdown(rt,m-l+1,r-m);
    if(L <= m) update(L,R,C,l,m,rt<<1);
    if(R >  m) update(L,R,C,m+1,r,rt<<1|1);
    pushup(rt);
}

ll query(int L,int R,int l,int r,int rt)
{
    if(L <= l&&r <= R){
        return sum[rt];
    }
    int m = (l+r)>>1;
    pushdown(rt,m-l+1,r-m);
    ll ans = 0;
    if(L <= m) ans = (ans + query(L,R,l,m,rt<<1))%mod;
    if(R >  m) ans = (ans + query(L,R,m+1,r,rt<<1|1))%mod;
    return ans;
}


//————————————————————————树上加、树上求和————————————————————————
void tadd(int x,int y,int k)
{
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        update(dfn[top[x]],dfn[x],k,1,n,1);
        x = f[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
    update(dfn[x],dfn[y],k,1,n,1);
}
ll tsum(int x,int y)
{
    ll ans = 0;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        ans = (ans + query(dfn[top[x]],dfn[x],1,n,1))%mod;
        x = f[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
    ans = (ans + query(dfn[x],dfn[y],1,n,1))%mod;
    return ans;
}


int main()
```

```
{
    int m;
    int num;//根节点序号
    scanf("%d%d%d%d",&n,&m,&num,&mod);
    init();
    for(int i = 1;i <= n;i++){
        scanf("%d",&a[i]);
    }
    for(int i = 1;i <= n-1;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        eadd(u,v);
        eadd(v,u);
    }
    dfs1(num,0,1);
    dfs2(num,num);
    build(1,n,1);
    for(int i = 1;i <= m;i++){
        int op;
        scanf("%d",&op);
        if(op == 1){//将树从x到y节点最短路径上所有节点的值都加上z
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            tadd(x,y,z%mod);
        }
        else if(op == 2){//求树从x到y节点最短路径上所有节点的值之和
            int x,y;
            scanf("%d%d",&x,&y);
            printf("%lld\n",tsum(x,y));
        }
        else if(op == 3){//将以x为根节点的子树内所有节点值都加上z
            int x,z;
            scanf("%d%d",&x,&z);
            update(dfn[x],dfn[x]+size[x]-1,z%mod,1,n,1);
        }
        else if(op == 4){//求以x为根节点的子树内所有节点值之和
            int x;
            scanf("%d",&x);
            printf("%lld\n",query(dfn[x],dfn[x]+size[x]-1,1,n,1)%mod);
        }
    }
    return 0;
}
```