

目录

1	单调栈 + 单调队列	1
1.1	单调栈	1
1.2	单调队列	1
2	树状数组	1
2.1	lowbit 之和	1
2.2	区间加减 + 区间和查询	2
2.3	统计前后顺序不同数字对个数（三维偏序问题）	2
3	线段树	3
3.1	ZKW 线段树	3
3.1.1	开局	3
3.1.2	单点修改 + 区间查询	4
3.1.3	单点修改 + 区间查询最大字段和	4
3.1.4	区间加减 + 单点查询	5
3.1.5	区间加减 + 区间最值查询（lazy 标记）	5
3.2	普通线段树	6
3.2.1	单点修改 + 区间查询	6
3.2.2	区间修改 + 区间查询	7
3.2.3	区间染色	8
3.2.4	区间修改 + 区间查询：矩阵	9
4	普通平衡树 Treap	12
5	树链剖分	14

1 单调栈 + 单调队列

1.1 单调栈

```
1 int s[N], top=0;
2 for(int i=1;i<=n;i++)
3 {
4     while(top&&a[s[top]]>a[i]) top--;
5     if(s[top]==a[i]) s[top]=i; //去重复操作，视情况而定
6     s[++top]=i;
7 }
```

1.2 单调队列

```
1 int l=0,r=1;
2 int q[N];
3 for(int i=1;i<=n;i++)
4 {
5     while(l<=r&&a[q[r]]>a[i]) r--;
6     q[++r]=i;
7     while(l<=r&&q[l]<i-m+1) l++; //判断条件看情况
8     ans[i]=a[q[l]];
9 }
```

2 树状数组

2.1 lowbit 之和

```
1 /*
2     HDU5975 有n个集合，第 i 个集合的数是[i-lowbit(i)+1,i-1]+i
3     第一种方式是集合[a,b]共有多少个数=lowbit之和? (sol1)
4     第二种方式是数字x在几个集合里面? (sol2)
5     long long型注意!
6 */
7 #define ll long long
8 ll lowbit(ll x){
9     return x&(-x);
10 }
11 ll sum(ll x){ //计算[1,x]中所有数lowbit(k)之和
12     ll ans=0ll;
13     for(ll p=1ll;p<=x;p<=<1){
14         ans+=(x/p-x/(p<<1))*p;
15     }
```

```

16     return ans;
17 }
18 ll sol1(ll x, ll y){ //计算[x,y]中所有树的lowbit(k)之和
19     return sum(y)-sum(x-1ll);
20 }
21 ll sol2(ll x, ll n){
22     ll res=0ll;
23     while(x<=n){
24         res++;
25         x+=lowbit(x);
26     }
27     return res;
28 }

```

2.2 区间加减 + 区间和查询

```

1 #define LL long long
2 const int SIZE=10005;
3 ll T1[SIZE], T2[SIZE], T[SIZE]; //T[]用于存结点值，用过一次即仍
4 int lowbit(int k){
5     return k&-k;
6 }
7 void update(ll x, ll N, ll w){ //把x位置之后所有数的值+w
8     for(ll i=x; i<=N; i+=lowbit(i)){
9         T1[i]+=w; T2[i]+=w*(x-1);
10    }
11 }
12 void range_update(ll l, ll r, ll v){ //在[l,r]上加v
13     update(l, v); update(r+1, -v);
14 }
15 ll sum(ll x){
16     ll ans=0;
17     for(ll i=x; i>0; i-=lowbit(i)){
18         ans+=x*T1[i]-T2[i];
19     }
20     return ans;
21 }
22 ll range_ask(ll l, ll r){ //返回[l,r]的和
23     return sum(r)-sum(l-1);
24 }
25 void init(int N){
26     for(int i=1; i<=N; i++){
27         update(i, T[i]-T[i-1]);
28     }
29 }

```

2.3 统计前后顺序不同数字对个数（三维偏序问题）

```

1 #define ll long long
2 const int SIZE=200000+50;
3 ll arr[3][SIZE]; //数据存放为[1,n]的范围
4 ll tree[SIE];
5 int n; //n为每组数字个数
6 int lowbit(int k){
7     return k&-k;
8 }
9 void add(int x,int k){
10     while(x<=n){
11         tree[x]+=k;
12         x+=lowbit(x);
13     }
14 }
15 ll sum(int x){
16     ll ans=0;
17     while(x!=0){
18         ans+=tree[x];
19         x-=lowbit(x);
20     }
21     return ans;
22 }
23 int pos[SIE];
24 ll CountInversions(int x,int y){
25     memset(tree,0,sizeof(tree));
26     for(int i=1;i<=n;i++){
27         pos[arr[x][i]]=i;
28     }
29     ll ans=0;
30     for(int i=n;i>=1;i--){
31         ans+=sum(pos[arr[y][i]]);
32         add(pos[arr[y][i]],1);
33     }
34     return ans;
35 }
36
37 //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)
38 ll invers=(CountInversions(0,1)+CountInversions(1,2)+CountInversions(2,0))/2ll;
39 ll tot=((ll)n*(ll)(n-1))/2ll; //这里一定要加ll!!不然会爆
40 printf("%lld\n",tot-invers);

```

3 线段树

3.1 ZKW 线段树

3.1.1 开局

```

1 for(M=1;M<=n;M<=1); //获得层数M

```

```

2  for(int i=1;i<=M<<1;i++)//初始化
3  {
4      T[i]=0;
5  }
6  for(int i=1,x;i<=n;i++)//建树
7  {
8      scanf("%d",&x);
9      modify(i,x);
10 }

```

3.1.2 单点修改 + 区间查询

```

1  void modify(int n,int v)//单点修改
2  {
3      for(T[n+=M]+=v,n>>=1;n;n>>=1)
4          T[n]=T[n+n]+T[n+n+1];
5  }
6  ll query(int l, int r)//区间查询和，可调为最大
7  {
8      ll ans=0;
9      for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
10     {
11         if(~l & 1) ans+=T[l^1];
12         if(r & 1) ans+=T[r^1];
13     }
14     return ans;
15 }

```

3.1.3 单点修改 + 区间查询最大字段和

```

1  struct Node{
2      ll pre,suf,max,sum;
3  }T[200000],null;
4  Node merge(Node l, Node r)
5  {
6      Node res;
7      res.sum=l.sum+r.sum;//区间和
8      res.pre=max(l.pre,l.sum+r.pre);//最大前缀
9      res.suf=max(r.suf,l.suf+r.sum);//最大
10     res.max=max(l.max,r.max);//最大子段和
11     res.max=max(res.max,l.suf+r.pre);
12     return res;
13 }
14 void modify(int n,int v)//单点修改
15 {
16     for(T[n+=M]={v,v,v,v},n>>=1;n;n>>=1)
17         T[n]=merge(T[n+n],T[n+n+1]);
18 }

```

```

19 ll query(int l, int r)//查询
20 {
21     Node resl(null),resr(null);
22     // resl.pre=resl.suf=resl.sum=0;
23     // resr.pre=resr.suf=resr.sum=0;
24     resl.max=resr.max=-inf;
25     for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
26     {
27         if(~l & 1) resl=merge(resl,T[l^1]);
28         if(r & 1) resr=merge(T[r^1],resr);
29     }
30     return merge(resl,resr).max;
31 }

```

3.1.4 区间加减 + 单点查询

```

1 void add(int l,int r,int v)//区间加减
2 {
3     for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
4     {
5         if(~l & 1) T[l^1]+=v;
6         if(r & 1) T[r^1]+=v;
7     }
8 }
9 ll query(int n)//单点查询
10 {
11     ll ans=0;
12     for(n+=M;n>>=1) ans+=T[n];
13     return ans;
14 }

```

3.1.5 区间加减 + 区间最值查询 (lazy 标记)

```

1 void add(int L,int R,int v)//区间修改
2 {
3     L+=M-1,R+=M+1;
4     for(int l=L,r=R;l^r^1;l>>=1,r>>=1)
5     {
6         if(~l & 1) lazy[l^1]+=v,T[l^1]+=v;
7         if(r & 1) lazy[r^1]+=v,T[r^1]+=v;
8     }
9     for(int l=L>>1,r=R>>1;l;l>>=1,r>>=1)
10    {
11        T[l]=max(T[l+l],T[l+l+1])+lazy[l];
12        T[r]=max(T[r+r],T[r+r+1])+lazy[r];
13    }
14 }
15 int query(int l, int r)//区间查询

```

```

16 {
17     int lmax=-inf,rmax=-inf;
18     for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
19     {
20         if(lazy[l]&&lmax!=-inf) lmax+=lazy[l];
21         if(lazy[r]&&rmax!=-inf) rmax+=lazy[r];
22         if(~l & 1) lmax=max(lmax,T[l^1]);
23         if(r & 1) rmax=max(rmax,T[r^1]);
24     }
25     for(;l;l>>=1,r>>=1)
26     {
27         lmax+=lazy[l],rmax+=lazy[r];
28     }
29     return max(lmax,rmax);
30 }

```

3.2 普通线段树

3.2.1 单点修改 + 区间查询

```

1 #define lson l, mid, rt << 1
2 #define rson mid + 1, r, rt << 1 | 1
3 const int SIZE = 5e5+50;
4 struct node{
5     int l,r;int val;
6 }T[SIZE*3];
7 int arr[SIZE];// 用于存从[1,n]中的数组
8 void build(int l,int r,int rt){
9     T[rt].l = l;T[rt].r = r;
10    if(l == r){
11        T[rt].val = arr[l];
12        return ;
13    }
14    int mid = (T[rt].l + T[rt].r)>>1;
15    build(lson); build(rson);
16    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
17 }
18 void update(int tar,int rt,int k){// 单点修改将tar改为k
19     if(T[rt].l==T[rt].r){
20         T[rt].val += k;return ;
21     }
22     int mid = (T[rt].l + T[rt].r)>>1;
23     if(tar <= mid) update(tar,rt<<1,k);
24     else update(tar,rt<<1|1,k);
25     T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
26 }
27 int ans = 0;
28 void Find(int rt,int l,int r){
29     if(T[rt].l>=l && T[rt].r<=r){

```

```

30     ans += T[rt].val;return ;
31 }
32 if(T[rt<<1].r>=l) Find(rt<<1,l,r);
33 if(T[rt<<1|1].l<=r) Find(rt<<1|1,l,r);
34 }

```

3.2.2 区间修改 + 区间查询

```

1  #define ll long long
2  const int SIZE = 1e5+5;
3  struct node{
4      int l,r;int val;
5  }T[SIZE*3];
6  int lazy[SIZE*3];
7  int arr[SIZE];// arr数组用于记录[1,n]的数据
8  void build(int l,int r,int rt){
9      T[rt].l = l;T[rt].r = r;
10     if(l==r){
11         T[rt].val = arr[l];return ;
12     }
13     int mid = (l+r)>>1;
14     build(l,mid,rt<<1);
15     build(mid+1,r,rt<<1|1);
16     T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
17 }
18 // l,r为指针, L,R是要修改的区间[L,R],为整个区间加上x
19 void update(int L,int R,int x,int rt){
20     if(L<=T[rt].l && R>=T[rt].r){
21         T[rt].val += x*(T[rt].r-T[rt].l+1);
22         lazy[rt] += x;
23         return ;
24     }
25     int mid = (T[rt].l + T[rt].r)>>1;
26     if(lazy[rt]){ // 下推标记更新一波
27         T[rt<<1].val += lazy[rt]*(mid-T[rt].l+1);
28         T[rt<<1|1].val += lazy[rt]*(T[rt].r-mid);
29         lazy[rt<<1] += lazy[rt];
30         lazy[rt<<1|1] += lazy[rt];
31         lazy[rt] = 0;
32     }
33     if(L <= mid)update(L,R,x,rt<<1);
34     if(R >mid)update(L,R,x,rt<<1|1);
35     T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
36 }
37
38 ll query(int L,int R,int rt){
39     // printf("rt=%d",rt);
40     if(L<=T[rt].l && R>=T[rt].r)return T[rt].val;
41     int mid = (T[rt].l + T[rt].r)>>1;

```



```

42     if(lazy[rt]){
43         T[rt<<1].val += lazy[rt]*(mid-T[rt].l+1);
44         T[rt<<1|1].val += lazy[rt]*(T[rt].r-mid);
45         lazy[rt<<1] += lazy[rt];
46         lazy[rt<<1|1] += lazy[rt];
47         lazy[rt] = 0;
48     }
49     ll sum=0;
50     if(L <= mid)sum += query(L,R,rt<<1);
51     if(R > mid)sum += query(L,R,rt<<1|1);
52     return sum;
53 }

```

3.2.3 区间染色

```

1  const int SIZE=1e5+50;
2  const int col_SIZE=1e5+50;
3  bool col_vis[col_SIZE];
4  struct node{
5      int l,r;int col;
6  }T[SIZE*3];
7
8  void build(int root,int l,int r){
9      T[root].l = l;
10     T[root].r = r;
11     if(l == r)return ;
12     int mid=(T[root].l + T[root].r)>>1;
13     build(root<<1,l,mid);
14     build((root<<1)+1,mid+1,r);
15 }
16 void update(int root,int l,int r,int col){
17     if(T[root].l>=l && T[root].r<=r){
18         T[root].col = col;return ;
19     }
20     else{
21         if(T[root].col > 0){
22             T[root<<1].col = T[root].col;
23             T[(root<<1)+1].col = T[root].col;
24             T[root].col = 0;
25         }
26         int mid = (T[root].l + T[root].r)>>1;
27         if(l>mid){
28             update((root<<1)+1,l,r,col);
29         }
30         else if(r<=mid){
31             update(root<<1,l,r,col);
32         }
33         else{
34             update(root<<1,l,mid,col);

```

```

35         update((root<<1)+1,mid+1,r,col);
36     }
37 }
38 }
39
40 void Find(int root,int l,int r){
41     if(T[root].l==0 || T[root].r==0)return ;
42     if(T[root].col > 0 ){
43         col_vis[T[root].col] = true;
44     }
45     else{
46         int mid = (T[ root ].l + T[ root ].r ) >> 1;
47         if(l>mid){
48             Find((root<<1)+1,l,r);
49         }
50         else if(r<=mid){
51             Find(root<<1,l,r);
52         }
53         else{
54             Find(root<<1,l,mid);
55             Find((root<<1)+1,mid+1,r);
56         }
57     }
58 }
59 void init(int N){//N为最大的颜色
60     for(int i=0;i<=N;i++){
61         col_vis[i]=0;
62     }
63 }
64
65 int tot(int l,int r,int N){//统计[l,r]内颜色范围为[1,N]共有几种
66     init(N);
67     Find(1,l,r);
68     int tot=0;
69     for(int i=0;i<=N;i++){
70         if(col_vis[i])tot++;
71     }
72     return tot;
73 }

```

3.2.4 区间修改 + 区间查询：矩阵

```

1  /*
2     本模板需要配合矩阵快速幂食用qwq
3  */
4  typedef long long ll;
5  int n;
6  mat A(2,2),B(2,2);
7  const int SIZE=1e5+50;

```

```

8 char str[SIZE];
9 struct node{
10     int l,r;
11     bool lazy=0;
12 }T[SIZE*3];
13 mat ma[SIZE*3][2];//0是正序, 1是反序
14 void push_up(int root){
15     ma[root][0]=ma[root<<1][0]*ma[(root<<1)|1][0];
16     ma[root][1]=ma[root<<1][1]*ma[(root<<1)|1][1];
17     return ;
18 }
19 void build(int root,int l,int r){
20     T[root].l=l;T[root].r=r;
21     if(l==r){
22         if(str[l]=='A'){
23             ma[root][0].n = A.n;ma[root][0].m = A.m;
24             for(int i=0;i<A.n;i++)
25                 for(int j=0;j<A.m;j++)
26                     ma[root][0].v[i][j]=A.v[i][j];
27             //-----
28             ma[root][1].n = B.n;ma[root][1].m = B.m;
29             for(int i=0;i<B.n;i++)
30                 for(int j=0;j<B.m;j++)
31                     ma[root][1].v[i][j]=B.v[i][j];
32         }
33         else{
34             ma[root][0].n = B.n;ma[root][0].m = B.m;
35             for(int i=0;i<B.n;i++)
36                 for(int j=0;j<B.m;j++)
37                     ma[root][0].v[i][j]=B.v[i][j];
38             //-----
39             ma[root][1].n = A.n;ma[root][1].m = A.m;
40             for(int i=0;i<A.n;i++)
41                 for(int j=0;j<A.m;j++)
42                     ma[root][1].v[i][j]=A.v[i][j];
43         }
44         return ;
45     }
46     int mid=(T[root].l+T[root].r)>>1;
47     build(root<<1,l,mid);
48     build((root<<1)|1,mid+1,r);
49     push_up(root);
50     return ;
51 }
52
53 void push_down(int root){
54     if(T[root].lazy){
55         swap(ma[root<<1][0],ma[root<<1][1]);
56         swap(ma[(root<<1)|1][0],ma[(root<<1)|1][1]);
57         T[root<<1].lazy=!T[root<<1].lazy;

```

```

58         T[(root<<1)|1].lazy=!T[(root<<1)|1].lazy;
59         T[root].lazy=0;
60     }
61 }
62 void update(int root,int l,int r){
63     if(l<=T[root].l && T[root].r<=r){
64         T[root].lazy = !T[root].lazy;
65         swap(ma[root][0],ma[root][1]);
66         return ;
67     }
68     push_down(root);
69     int mid = (T[root].l + T[root].r)>>1;
70     if(l <= mid)update(root<<1,l,r);
71     if(r > mid)update((root<<1)|1,l,r);
72     push_up(root);
73 }
74
75 mat tmp(2,2);// 对最后的答案矩阵进行修改
76 void find(int root,int l,int r){
77     if(l <= T[root].l && T[root].r<=r){
78         tmp = tmp*ma[root][0];
79         return;
80     }
81     push_down(root);
82     int mid = (T[root].l + T[root].r)>>1;
83     if(l <= mid)find(root<<1,l,r);
84     if(mid < r)find((root<<1)|1,l,r);
85     return;
86 }
87 int main(){
88     A.v[0][0]=1;A.v[0][1]=0;A.v[1][0]=1;A.v[1][1]=1;
89     B.v[0][0]=1;B.v[0][1]=1;B.v[1][0]=0;B.v[1][1]=1;
90     int q;scanf("%d",&n,&q);
91     scanf("%s",str+1);
92     build(1,1,n);
93     while(q--){
94         int op;scanf("%d",&op);
95         if(op==1){
96             int L,R;scanf("%d%d",&L,&R);
97             update(1,L,R);
98         }
99         else{
100             int L,R;ll AA,BB;scanf("%d%d%lld%lld",&L,&R,&AA,&BB);
101             tmp.v[0][0]=1;
102             tmp.v[0][1]=0;
103             tmp.v[1][0]=0;
104             tmp.v[1][1]=1;
105             mat qwq(1,2);
106             qwq.v[0][0]=AA%mod;qwq.v[0][1]=BB%mod;
107             find(1,L,R);

```

```

108         qwq = qwq*tmp;
109         qwq.display();
110     }
111 }
112 }

```

4 普通平衡树 Treap

```

1  #define inf 0x3f3f3f3f
2  const int maxn = 1e6+5;
3  int ch[maxn][2];
4  int val[maxn],dat[maxn];
5  int sz[maxn],cnt[maxn];
6  int tot,root;
7  int New(int v){// 辅助函数
8      val[++tot] = v;
9      dat[tot] = rand();
10     sz[tot] = 1;
11     cnt[tot] = 1;
12     return tot;
13 }
14 void pushup(int id){// 辅助函数
15     sz[id] = sz[ch[id][0]] + sz[ch[id][1]] + cnt[id];
16 }
17 void build(){// 辅助函数
18     root = New(-INF),ch[root][1] = New(INF);
19     pushup(root);
20 }
21 void Rotate(int &id,int d){// 辅助函数
22     int temp = ch[id][d ^ 1];
23     ch[id][d ^ 1] = ch[temp][d];
24     ch[temp][d] = id;
25     id = temp;
26     pushup(ch[id][d]),pushup(id);
27 }
28 void insert(int &id,int v){// 插入一个数值为v
29     if(!id){
30         id = New(v);return ;
31     }
32     if(v == val[id])cnt[id]++;
33     else{
34         int d = v < val[id] ? 0 : 1;
35         insert(ch[id][d],v);
36         if(dat[id] < dat[ch[id][d]])Rotate(id,d ^ 1);
37     }
38     pushup(id);
39 }
40 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)

```

```

41     if(!id)return ;
42     if(v == val[id]){
43         if(cnt[id] > 1){cnt[id]--,pushup(id);return ;}
44         if(ch[id][0] || ch[id][1]){
45             if(!ch[id][1] || dat[ch[id][0]] > dat[ch[id][1]]){
46                 Rotate(id,1),Remove(ch[id][1],v);
47             }
48             else Rotate(id,0),Remove(ch[id][0],v);
49             pushup(id);
50         }
51         else id = 0;
52         return ;
53     }
54     v < val[id] ? Remove(ch[id][0],v) : Remove(ch[id][1],v);
55     pushup(id);
56 }
57 int get_rank(int id,int v){//
58     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)
59     if(!id)return 0;
60     if(v == val[id])return sz[ch[id][0]] + 1;
61     else if(v < val[id])return get_rank(ch[id][0],v);
62     else return sz[ch[id][0]] + cnt[id] + get_rank(ch[id][1],v);
63 }
64 int get_val(int id,int rank){// 查询排名为rank的数
65     if(!id)return INF;
66     if(rank <= sz[ch[id][0]])return get_val(ch[id][0],rank);
67     else if(rank <= sz[ch[id][0]] + cnt[id])return val[id];
68     else return get_val(ch[id][1],rank - sz[ch[id][0]] - cnt[id]);
69 }
70 int get_pre(int v){// 求v的前驱(前驱定义为小于v，且最大的数)
71     int id = root,pre;
72     while(id){
73         if(val[id] < v)pre = val[id],id = ch[id][1];
74         else id = ch[id][0];
75     }
76     return pre;
77 }
78 int get_next(int v){// 求v的后继(后继定义为大于v，且最小的数)
79     int id = root,next;
80     while(id){
81         if(val[id] > v)next = val[id],id = ch[id][0];
82         else id = ch[id][1];
83     }
84     return next;
85 }
86 int main(){
87     build();int x;
88     int shk;scanf("%d",&shk);
89     while(shk--){
90         int op;scanf("%d",&op);

```

```

90     switch(op){
91         case 1:scanf("%d",&x);insert(root,x);break;
92         case 2:scanf("%d",&x);Remove(root,x);break;
93         case 3:scanf("%d",&x);printf("%d\n",get_rank(root,x)-1);break;
94         case 4:scanf("%d",&x);printf("%d\n",get_val(root,x+1));break;
95         case 5:scanf("%d",&x);printf("%d\n",get_pre(x));break;
96         case 6:scanf("%d",&x);printf("%d\n",get_next(x));break;
97     }
98 }
99 }

```

5 树链剖分

```

1  int sum[MAXM*4],add[MAXM*4];
2  int a[MAXM],n;
3  int idx;
4  int first[MAXM];
5  struct edge{
6      int v,next;
7  }e[MAXM*2];
8  int f[MAXM],son[MAXM],size[MAXM],dfn[MAXM],dep[MAXM],top[MAXM],seq[MAXM];
9  int cnt;
10 int mod;
11 /*
12 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.
13 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.
14 dep是结点的深度
15 */
16
17 //—————加边和预处理—————
18 void eadd(int a,int b)
19 {
20     e[idx].v = b;
21     e[idx].next = first[a];
22     first[a] = idx++;
23 }
24 void init()
25 {
26     memset(first,-1,sizeof(first));
27     idx = 1;
28     cnt = 0;
29 }
30 void dfs1(int u,int fa,int depth)
31 {
32     f[u] = fa; size[u] = 1; dep[u] = depth;
33     int maxson = -1;
34     for(int i = first[u];i != -1;i = e[i].next){
35         int v = e[i].v;

```

```

36         if(v == fa) continue;
37         dfs1(v,u,depth+1);
38         size[u] += size[v];
39         if(size[v]>maxson) son[u] = v,maxson = size[v];
40     }
41 }
42 void dfs2(int u,int t)
43 {
44     top[u] = t;
45     dfn[u] = ++cnt;
46     seq[cnt] = a[u];
47     if(!son[u]) return;
48     dfs2(son[u],t);
49     for(int i = first[u];i != -1;i = e[i].next){
50         int v = e[i].v;
51         if(v != son[u]&&v != f[u]) dfs2(v,v);
52     }
53 }
54
55 //—————线段树—————
56 void pushup(int rt)
57 {
58     sum[rt] = (sum[rt<<1]+sum[rt<<1|1])%mod;
59 }
60 void build(int l,int r,int rt)
61 {
62     if(l == r){
63         sum[rt] = seq[l]%mod;
64         return;
65     }
66     int m = (l+r)>>1;
67     build(l,m,rt<<1);
68     build(m+1,r,rt<<1|1);
69     pushup(rt);
70 }
71 void pushdown(int rt,int ln,int rn)
72 {
73     if(add[rt]){
74         add[rt<<1] = (add[rt<<1] + add[rt])%mod;
75         add[rt<<1|1] = (add[rt<<1|1] + add[rt])%mod;
76         sum[rt<<1] = (sum[rt<<1] + add[rt]*ln%mod)%mod;
77         sum[rt<<1|1] = (sum[rt<<1|1] + add[rt]*rn%mod)%mod;
78         add[rt] = 0;
79     }
80 }
81 void update(int L,int R,int C,int l,int r,int rt)
82 {
83     if(L <= l&&r <= R){
84         sum[rt] = (sum[rt] + C*(r-l+1)%mod)%mod;
85     }

```



```

86         add[rt] = (add[rt] + C)%mod;
87         return;
88     }
89     int m = (l+r)>>1;
90     pushdown(rt,m-l+1,r-m);
91     if(L <= m) update(L,R,C,l,m,rt<<1);
92     if(R > m) update(L,R,C,m+1,r,rt<<1|1);
93     pushup(rt);
94 }
95
96 ll query(int L,int R,int l,int r,int rt)
97 {
98     if(L <= l&&r <= R){
99         return sum[rt];
100     }
101     int m = (l+r)>>1;
102     pushdown(rt,m-l+1,r-m);
103     ll ans = 0;
104     if(L <= m) ans = (ans + query(L,R,l,m,rt<<1))%mod;
105     if(R > m) ans = (ans + query(L,R,m+1,r,rt<<1|1))%mod;
106     return ans;
107 }
108
109
110 //—————树上加、树上求和—————
111 void tadd(int x,int y,int k)
112 {
113     while(top[x]!=top[y]){
114         if(dep[top[x]]<dep[top[y]]) swap(x,y);
115         update(dfn[top[x]],dfn[x],k,1,n,1);
116         x = f[top[x]];
117     }
118     if(dep[x]>dep[y]) swap(x,y);
119     update(dfn[x],dfn[y],k,1,n,1);
120 }
121 ll tsum(int x,int y)
122 {
123     ll ans = 0;
124     while(top[x]!=top[y]){
125         if(dep[top[x]]<dep[top[y]]) swap(x,y);
126         ans = (ans + query(dfn[top[x]],dfn[x],1,n,1))%mod;
127         x = f[top[x]];
128     }
129     if(dep[x]>dep[y]) swap(x,y);
130     ans = (ans + query(dfn[x],dfn[y],1,n,1))%mod;
131     return ans;
132 }
133
134
135 int main()

```

```

136 {
137     int m;
138     int num;//根节点序号
139     scanf("%d%d%d",&n,&m,&num,&mod);
140     init();
141     for(int i = 1;i <= n;i++){
142         scanf("%d",&a[i]);
143     }
144     for(int i = 1;i <= n-1;i++){
145         int u,v;
146         scanf("%d%d",&u,&v);
147         eadd(u,v);
148         eadd(v,u);
149     }
150     dfs1(num,0,1);
151     dfs2(num,num);
152     build(1,n,1);
153     for(int i = 1;i <= m;i++){
154         int op;
155         scanf("%d",&op);
156         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
157             int x,y,z;
158             scanf("%d%d%d",&x,&y,&z);
159             tadd(x,y,z%mod);
160         }
161         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
162             int x,y;
163             scanf("%d%d",&x,&y);
164             printf("%lld\n",tsum(x,y));
165         }
166         else if(op == 3){//将以x为根节点的子树内所有节点值都加上z
167             int x,z;
168             scanf("%d%d",&x,&z);
169             update(dfn[x],dfn[x]+size[x]-1,z%mod,1,n,1);
170         }
171         else if(op == 4){//求以x为根节点的子树内所有节点值之和
172             int x;
173             scanf("%d",&x);
174             printf("%lld\n",query(dfn[x],dfn[x]+size[x]-1,1,n,1)%mod);
175         }
176     }
177     return 0;
178 }

```