0.1 Link Cut Tree

```cpp
#include <iostream>
#include <stdio.h>
using namespace std;
const int N = 5e5 + 50;
int ch[N][2], fa[N], sum[N], val[N], rev[N], laz[N], sze[N];
/*
    ch[N][2] 左右儿子
    fa[N] 父亲指向
    sum[N] 路径权值和
    val[N] 点权
    rev[N] 翻转标记
    laz[N] 权值标记
    sze[N] 辅助树上子树大小
*/
struct LCT
{
#define lc(x) ch[x][0]
#define rc(x) ch[x][1]
    inline void Reverse(int x)
    {
        swap(lc(x), rc(x)), rev[x] ^= 1;
    }
    inline void Pushup(int x)
    {
        sum[x] = sum[lc(x)] ^ sum[rc(x)] ^ val[x];
    }
    inline void Pushdown(int x)
    {
        if (rev[x])
        {
            if (lc(x))
                Reverse(lc(x));
            if (rc(x))
                Reverse(rc(x));
            rev[x] = 0;
        }
    }
    inline bool WhichChild(int x)
    {
        return x == rc(fa[x]);
        //return 0 lc
        //return 1 rc
    }
    inline bool isRoot(int x)
    {
        return lc(fa[x]) != x && rc(fa[x]) != x;
    }
    void Update(int x)
    {
        if (!isRoot(x))
            Update(fa[x]);
        Pushdown(x);
    }
    void Rotate(int x)
    {
        int y = fa[x], z = fa[y], which = WhichChild(x);
        if (!isRoot(y)) //虚边的存在
            ch[z][WhichChild(y)] = x;
        ch[y][which] = ch[x][which ^ 1];
        fa[ch[x][which ^ 1]] = y;
        ch[x][which ^ 1] = y;
        fa[y] = x, fa[x] = z;
        Pushup(y), Pushup(x);
    }
```

```
65      void Splay(int x)
66      {
67          Update(x);
68          for (int f; f = fa[x], !isRoot(x); Rotate(x))
69              if (!isRoot(f))
70                  Rotate(WhichChild(x) == WhichChild(f) ? f : x);
71      }
72      void Access(int x)
73      {
74          for (int pre = 0; x; pre = x, x = fa[x])
75              Splay(x), rc(x) = pre, Pushup(x);
76      }
77      void MakeRoot(int x)
78      {
79          Access(x), Splay(x), Reverse(x);
80      }
81      int FindRoot(int x)
82      {
83          Access(x), Splay(x);
84          while (lc(x))
85              Pushdown(x), x = lc(x);
86          Splay(x); //防止卡链 保证时间复杂度 例如根在链的远处却多次找根
87          return x;
88      }
89      void Link(int x, int y)
90      {
91          MakeRoot(x);
92          if (FindRoot(y) == x)
93              return;
94          fa[x] = y;
95          //if Find(y)==x 在同一颗树中 不合法 合法连虚边
96      }
97      void Cut(int x, int y)
98      {
99          /*
100         操作合法
101         MakeRoot(x), Access(y), Splay(y);
102         fa[x] = lc(y) = 0;
103         Pushup(y);
104         */
105         MakeRoot(x);
106         if (FindRoot(y) != x || fa[y] != x || lc(y))
107             return;
108         fa[y] = rc(x) = 0;
109         Pushup(x);
110     }
111     void Split(int x, int y)
112     {
113         MakeRoot(x), Access(y), Splay(y);
114     }
115 } lct;
116 int main()
117 {
118     int n, m;
119     scanf("%d%d", &n, &m);
120     for (int i = 1; i <= n; i++)
121         scanf("%d", &val[i]);
122     for (int i = 1; i <= m; i++)
123     {
124         int opt, x, y;
125         scanf("%d%d%d", &opt, &x, &y);
126         if (opt == 0)
127             lct.Split(x, y), printf("%d\n", sum[y]);
128         else if (opt == 1)
129             lct.Link(x, y);
130         else if (opt == 2)
```

```
131             lct.Cut(x, y);
132         else if (opt == 3)
133             lct.Splay(x), val[x] = y;
134     }
135     //system("pause");
136     return 0;
137 }
```

0.2 Splay Tree

应用：区间翻转

```
#include <iostream>
#include <stdio.h>
using namespace std;
const int N = 1e5 + 50;
int rt, tot, fa[N], ch[N][2];
int val[N], cnt[N], sze[N];
bool rev[N];
int data[N];
int n, m;
struct Splay_Tree
{
#define lc(x) ch[x][0]
#define rc(x) ch[x][1]
    inline void Pushup(int x)
    {
        sze[x] = sze[lc(x)] + sze[rc(x)] + cnt[x];
    }
    inline bool WhichChild(int x)
    {
        return x == rc(fa[x]);
        //return 0 lc
        //return 1 rc
    }
    inline void Pushdown(int x)
    {
        if (rev[x])
        {
            rev[lc(x)] ^= 1, rev[rc(x)] ^= 1;
            swap(lc(x), rc(x));
            rev[x] = 0;
        }
    }
    void Rotate(int x)
    {
        int y = fa[x], z = fa[y], which = WhichChild(x);
        Pushdown(y), Pushdown(x);
        ch[y][which] = ch[x][which ^ 1];
        fa[ch[x][which ^ 1]] = y;
        ch[x][which ^ 1] = y;
        fa[y] = x, fa[x] = z;
        if (z)
            ch[z][y == ch[z][1]] = x;
        Pushup(y), Pushup(x);
    }
    void Splay(int x, int goal) //x:the son of goal
    {
        for (int f; (f = fa[x]) != goal; Rotate(x))
            if (fa[f] != goal)
                Rotate(WhichChild(x) == WhichChild(f) ? f : x);
        if (!goal)
            rt = x;
    }
    int Kth(int x)
    {
        int cur = rt;
        while (1)
        {
            Pushdown(cur);
            if (lc(cur) && x <= sze[lc(cur)])
                cur = lc(cur);
            else
            {
                x -= cnt[cur] + sze[lc(cur)];
```

```
64                  if (x <= 0)
65                      return cur;
66                  cur = rc(cur);
67              }
68          }
69      }
70      int Build(int f, int l, int r)
71      {
72          if (l > r)
73              return 0;
74          int cur = ++tot;
75          int mid = (l + r) >> 1;
76          val[cur] = data[mid], fa[cur] = f, rev[cur] = 0;
77          sze[cur] = cnt[cur] = 1;
78          lc(cur) = Build(cur, l, mid - 1);
79          rc(cur) = Build(cur, mid + 1, r);
80          Pushup(cur);
81          return cur;
82      }
83      void DFS(int x)
84      {
85          if (x)
86          {
87              Pushdown(x);
88              DFS(lc(x));
89              if (val[x] >= 1 && val[x] <= n)
90                  printf("%d ", val[x]);
91              DFS(rc(x));
92          }
93      }
94      void Reverse(int l, int r)
95      {
96          int x = Kth(l);
97          int y = Kth(r + 2);
98          Splay(x, 0);
99          Splay(y, x);
100         int target = lc(rc(rt));
101         rev[target] ^= 1;
102     }
103 } splay;
104 int main()
105 {
106     scanf("%d%d", &n, &m);
107     for (int i = 0; i <= n + 1; i++)
108         data[i] = i;
109     rt = splay.Build(0, 0, n + 1);
110     int l, r;
111     for (int i = 0; i < m; i++)
112     {
113         scanf("%d%d", &l, &r);
114         splay.Reverse(l, r);
115     }
116     splay.DFS(rt);
117     printf("\n");
118     //system("pause");
119     return 0;
120 }
```

0.3 树分治

点分治：树上路径信息统计

```
int head[N], ver[M], nxt[M], edge[M], tot;
int sze[N], dist[N], rt, max_part[N], sum;
bool vis[N];
void calc_sze(int x, int fa)
{
    max_part[x] = 0, sze[x] = 1;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (vis[y] || y == fa)
            continue;
        calc_sze(y, x);
        sze[x] += sze[y];
        max_part[x] = max(max_part[x], sze[y]);
    }
    max_part[x] = max(max_part[x], sum - sze[x]);
    if (max_part[x] < max_part[rt])
        rt = x;
}
void calc_dist(int x, int fa)
{
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (vis[y] || y == fa)
            continue;
        dist[y] = dist[x] + edge[i];
        calc_dist(y, x);
    }
}
void Solve(int x)
{
    vis[x] = true;
    ans += calc(x, 0);
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (vis[y])
            continue;
        ans -= calc(y, edge[i]);//容斥
        sum = max_part[rt = 0] = sze[y];
        calc_sze(y, x), Solve(rt);
    }
}
sum = n;
max_part[rt = 0] = inf;
calc_sze(1, 0);
Solve(rt);
```

0.4 树链剖分

重链剖分：将树上路径剖分为链，配合线性数据结构（线段树）使用。

```
#include <cstring>
#include <iostream>
#include<stdio.h>
using namespace std;
const int N = 1e5 + 50;
const int M = 2e5 + 50;
#define ll long long
int mod;
int temp[N], weight[N];
```

```
int head[N], ver[M], nxt[M], tot;
int fa[N], dep[N], sze[N], son[N], top[N], dfn[N], rnk[N], dfn_tot;
struct SegmentTree
{
    int l, r;
    ll sum, add;
    #define ls(x) (x << 1)
    #define rs(x) (x << 1 | 1)
    #define l(x) tree[x].l
    #define r(x) tree[x].r
    #define sum(x) tree[x].sum
    #define add(x) tree[x].add
} tree[N << 2];
void PushUp(int rt)
{
    sum(rt) = (sum(ls(rt)) + sum(rs(rt))) % mod;
}
void PushDown(int rt)
{
if (add(rt))
{
    sum(ls(rt)) = (sum(ls(rt)) + add(rt) * (r(ls(rt)) - l(ls(rt)) + 1)) % mod;
    sum(rs(rt)) = (sum(rs(rt)) + add(rt) * (r(rs(rt)) - l(rs(rt)) + 1)) % mod;
    add(ls(rt)) = (add(ls(rt)) + add(rt)) % mod;
    add(rs(rt)) = (add(rs(rt)) + add(rt)) % mod;
    add(rt) = 0;
}
}
void SegmentTree_Build(int rt, int l, int r)
{
    l(rt) = l, r(rt) = r;
    if (l == r)
    {
        sum(rt) = weight[l];
        return;
    }
    int mid = (l + r) >> 1;
    SegmentTree_Build(ls(rt), l, mid);
    SegmentTree_Build(rs(rt), mid + 1, r);
    PushUp(rt);
    //向上更新
}
void Update(int rt, int l, int r, int d)
{
    if (l <= l(rt) && r(rt) <= r)
    {
        sum(rt) = (sum(rt) + (r(rt) - l(rt) + 1) * d) % mod;
        add(rt) = (add(rt) + d) % mod;
        return;
    }
    PushDown(rt);
    int mid = (l(rt) + r(rt)) >> 1;
    if (l <= mid)
        Update(ls(rt), l, r, d);
    if (r > mid)
        Update(rs(rt), l, r, d);
    PushUp(rt);
}
ll Query(int rt, int l, int r)
{
    if (l(rt) >= l && r(rt) <= r)
    {
        return sum(rt) % mod;
    }
    PushDown(rt);
    ll ret = 0;
```

```
76      int mid = (l(rt) + r(rt)) >> 1;
77      if (l <= mid)
78          ret = (ret + Query(ls(rt), l, r)) % mod;
79      if (r > mid)
80          ret = (ret + Query(rs(rt), l, r)) % mod;
81      return ret;
82  }
83  /*
84  重链剖分
85      fa[x] 表示节点x在树上的父亲
86      dep[x]表示节点x在树上的深度
87      sze[x]表示节点x的子树的节点个数
88      son[x]表示节点x的重儿子
89      top[x]表示节点x所在重链的顶部节点（深度最小）
90      dfn[x]表示节点x的DFS序 ，也是其在线段树中的编号
91      rnk[x]表示DFS序所对应的节点编号，有rnk[dfn[x]]=x
92  */
93  void Add(int x, int y)
94  {
95      ver[++tot] = y, nxt[tot] = head[x], head[x] = tot;
96  }
97  void Build(int x, int pre, int depth)
98  {
99      son[x] = -1, sze[x] = 1, dep[x] = depth;
100     for (int i = head[x]; i; i = nxt[i])
101     {
102         int y = ver[i];
103         if (y == pre)
104             continue;
105         Build(y, x, depth + 1);
106         sze[x] += sze[y], fa[y] = x;
107         if (son[x] == -1 || sze[son[x]] < sze[y])
108             son[x] = y;
109     }
110 }
111 void Decomposition(int x, int t)
112 {
113     top[x] = t, dfn[x] = ++dfn_tot, rnk[dfn_tot] = x;
114     if (son[x] == -1)
115         return;
116     Decomposition(son[x], t);
117     for (int i = head[x]; i; i = nxt[i])
118     {
119         int y = ver[i];
120         if (y != son[x] && y != fa[x])
121             Decomposition(y, y);
122     }
123 }
124 ll Query_Path(int x, int y)
125 {
126     ll ret = 0;
127     while (top[x] != top[y])
128     {
129         if (dep[top[x]] < dep[top[y]])
130             swap(x, y);
131         ret = (ret + Query(1, dfn[top[x]], dfn[x])) % mod;
132         x = fa[top[x]];
133     }
134     if (dep[x] > dep[y])
135         swap(x, y);
136     ret = (ret + Query(1, dfn[x], dfn[y])) % mod;
137     return ret;
138 }
139 void Update_Path(int x, int y, int d)
140 {
141     while (top[x] != top[y])
```

```
142     {
143         if (dep[top[x]] < dep[top[y]])
144             swap(x, y);
145         Update(1, dfn[top[x]], dfn[x], d);
146         x = fa[top[x]];
147     }
148     if (dep[x] > dep[y])
149         swap(x, y);
150     Update(1, dfn[x], dfn[y], d);
151 }
152 int main()
153 {
154     int n, m, r;
155     scanf("%d%d%d%d", &n, &m, &r, &mod);
156     for (int i = 1; i <= n; i++)
157         scanf("%d", &temp[i]);
158     int x, y;
159     for (int i = 1; i <= n - 1; i++)
160         scanf("%d%d", &x, &y), Add(x, y), Add(y, x);
161     Build(r, 0, 1);
162     Decomposition(r, r);
163     for (int i = 1; i <= n; i++)
164         weight[i] = temp[rnk[i]] % mod;
165     SegmentTree_Build(1, 1, dfn_tot);
166     for (int i = 1; i <= m; i++)
167     {
168         int op;
169         scanf("%d", &op);
170         int x, y, z;
171         if (op == 1)
172         {
173             //路径修改
174             scanf("%d%d%d", &x, &y, &z);
175             Update_Path(x, y, z);
176         }
177         else if (op == 2)
178         {
179             //路径和
180             scanf("%d%d", &x, &y);
181             printf("%lld\n", Query_Path(x, y));
182         }
183         else if (op == 3)
184         {
185             //子树修改
186             scanf("%d%d", &x, &z);
187             Update(1,dfn[x],dfn[x]+sze[x]-1,z);
188         }
189         else
190         {
191             //子树和
192             scanf("%d", &x);
193             printf("%lld\n", Query(1,dfn[x],dfn[x]+sze[x]-1));
194         }
195     }
196     //system("pause");
197     return 0;
198 }
```