

小小青蛙听风就是雨

HiedanoAkyuu、Oneman233、KR12138

2019 年 10 月 18 日

目录

1 字符串	1		
1.1 KMP	1		
1.2 EX-KMP	1		
1.3 Manacher	1		
1.4 串的最小表示	1		
1.5 后缀数组	1		
1.5.1 倍增 SA	1		
1.5.2 DC3	1		
1.6 回文自动机	1		
1.7 AC 自动机	1		
1.7.1 多模匹配	1		
1.7.2 自动机上 DP	2		
1.8 后缀自动机	2		
2 计算几何	2		
2.1 二维几何	2		
2.2 三维几何	3		
3 图论	3		
3.1 最短路	3		
3.1.1 Dijkstra	3		
3.1.2 SPFA	3		
3.1.3 Floyd	3		
3.1.4 负环	3		
3.1.5 差分约束	3		
3.2 最小生成树	3		
3.2.1 Prim	3		
3.2.2 Kruskal	3		
3.2.3 最小生成树计数	3		
3.2.4 次小生成树	3		
3.2.5 最小乘积生成树	3		
3.3 树的直径	3		
3.4 LCA	3		
3.4.1 Tarjan 离线	3		
3.4.2 倍增 LCA	3		
3.5 无向图与有向图联通性	4		
3.5.1 割点	4		
3.5.2 桥	4		
3.5.3 e-DCC	4		
3.5.4 v-DCC	4		
3.5.5 SCC	4		
3.5.6 2-SAT	4		
3.5.7 支配树	4		
3.6 二分图	4		
3.6.1 最大匹配-匈牙利	4		
3.6.2 带权匹配-KM	4		
3.7 网络流	4		
3.7.1 最大流-Dinic	4		
3.7.2 最小费用最大流-Dij+Dinic	5		
3.7.3 上下界流	5		
3.8 欧拉路	5		
3.9 Prufer 序列	5		
4 数据结构	5		
4.1 树状数组	5		
4.2 线段树	5		
4.2.1 带优先级线段树	5		
4.2.2 吉司机线段树	5		
4.2.3 线段树维护扫描线	6		
4.3 RMQ	6		
4.3.1 一维	6		
4.3.2 二维	6		
4.4 树链剖分	6		
4.4.1 点剖分	6		
4.4.2 边剖分	8		
4.5 平衡树	8		
4.5.1 Treap	8		
4.5.2 Splay	9		
4.6 动态树	9		
4.7 主席树	9		
4.8 树套树	9		
4.8.1 线段树套 Treap	9		
4.8.2 树状数组套线段树	12		
4.9 K-D Tree	12		
4.10 分治	12		
4.10.1 CDQ	12		
4.10.2 点分治	12		
4.10.3 dsu on tree	12		
4.10.4 整体二分	13		
4.11 分块	13		
4.11.1 普通分块	13		
4.11.2 莫队	13		
4.12 线性基	13		
4.13 珂朵莉树	13		
4.14 跳舞链	13		
5 动态规划	13		
5.1 SOS	13		
5.2 动态 DP	13		
5.3 插头 DP	13		
6 数学	13		
6.1 矩阵类	13		
6.2 质数筛	13		
6.2.1 埃筛	13		
6.2.2 线筛	13		
6.3 质数判定	13		
6.3.1 Miller Rabin	13		
6.4 质因数分解	13		
6.4.1 Pollard-Rho	13		
6.5 逆元	13		
6.5.1 EX-GCD 求逆元	13		
6.5.2 线性筛逆元	13		
6.5.3 阶乘逆元	13		
6.6 欧拉函数	13		
6.6.1 欧拉线筛	13		
6.6.2 求单个数的欧拉函数	13		
6.6.3 欧拉降幂	13		
6.6.4 一般积性函数求法	13		
6.7 EX-GCD	13		
6.8 CRT	13		
6.9 N 次剩余	13		
6.10 数论分块	13		
6.11 高斯消元	13		
6.11.1 普通消元	13		
6.11.2 异或方程组消元	13		
6.12 莫比乌斯反演	13		
6.12.1 莫比乌斯函数	13		
6.12.2 杜教筛	13		
6.12.3 洲阁筛	13		
6.12.4 min25 筛	13		
6.13 BSGS	13		
6.14 FFT	13		
6.15 FWT	13		
6.16 NTT	13		
6.17 数值计算	13		
6.17.1 辛普森	13		
6.17.2 自适应辛普森	13		
6.18 康拓展开	13		
6.19 卢卡斯定理	13		

7 其他13

7.1 快读快写13

7.2 约瑟夫环13

7.3 悬线法13

7.4 蔡勒公式13

7.5 三角公式13

7.6 海伦公式13

7.7 匹克定理13

7.8 组合计数13

7.8.1 计数原理13

7.8.2 卡特兰数13

7.8.3 Polya13

7.8.4 二项式反演公式13

7.8.5 斯特林反演公式13

7.8.6 组合数恒等式13

1 字符串

1.1 KMP

1.2 EX-KMP

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5  ex[i]: s1[i..l1-1]与s2的最大公共前缀长度
6  exnext[i]: s2[i..l2-1]与s2的最大公共前缀长度
7  get_exnext(s2) 求exnext[]
8  exkmp(s1,s2) 求ex[]
9 */
10
11 const int N=50005;
12 char s1[N],s2[N];
13 int ex[N],exnext[N];
14
15 void get_exnext(char s[N])
16 {
17     int n=strlen(s),p=1,j,i;
18     exnext[0]=n;
19     for(i=0;i<n-1&&s[i]==s[i+1];i++);
20     exnext[1]=i;
21     for(i=2;i<n;i++)
22         if(exnext[i-p]+i<p+exnext[p])
23             exnext[i]=exnext[i-p];
24         else
25         {
26             j=exnext[p]+p-i;
27             if(j<0) j=0;
28             while(i+j<n&&s[j]==s[i+j]) j++;
29             exnext[i]=j;
30             p=i;
31         }
32 }
33
34 void exkmp(char s1[N],char s2[N])
35 {
36     int l1=strlen(s1),l2=strlen(s2),p=0,i,j;
37     get_exnext(s2);
38     for(i=0;i<l1&&i<l2&&s1[i]==s2[i];i++);
39     ex[0]=i;
40     for(int i=1;i<l1;i++)
41     {
42         if(exnext[i-p]+i<p+ex[p])
43             ex[i]=exnext[i-p];
44         else
45         {
46             j=ex[p]+p-i;
47             if(j<0) j=0;
48             while(i+j<l1&&s1[i+j]==s2[j]) j++;
49             ex[i]=j;
50             p=i;
51         }
52     }
53 }

```

1.3 Manacher

1.4 串的最小表示

1.5 后缀数组

1.5.1 倍增 SA

1.5.2 DC3

1.6 回文自动机

1.7 AC 自动机

1.7.1 多模匹配

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5  trie静态开点+trie图优化
6 */
7
8 int sz,hd=1,nxt[1000005][26],fail[1000005],id
   [1000005],n;
9 char s[1000005];
10
11 void trie_clean()
12 {
13     sz=1;
14     memset(nxt,0,sizeof(nxt));
15     memset(fail,0,sizeof(fail));
16     memset(id,0,sizeof(id));
17 }
18
19 void trie_insert(int head,char s[],int len,int idx)
20 {
21     int p=head;
22     for(int i=0;i<len;i++)
23     {
24         int c=s[i]-'a';
25         if(!nxt[p][c]) nxt[p][c]=++sz;
26         p=nxt[p][c];
27     }
28     id[p]+=idx;
29 }
30
31 void acatm_build(int head)
32 {
33     int p,tp;
34     queue<int> q;
35     q.push(head);
36     fail[head]=0;
37     while(!q.empty())
38     {
39         p=q.front();
40         q.pop();
41         for(int i=0;i<26;i++)
42             if(nxt[p][i])
43             {
44                 fail[nxt[p][i]]=p==head?head:nxt[fail[p]
   ][i];
45                 q.push(nxt[p][i]);
46             }
47         else
48             nxt[p][i]=p==head?head:nxt[fail[p]][i];
49     }

```

```

50 }
51
52 int acatm_match(int head,char s[],int len)
53 {
54     int p=head,ret=0;
55     for(int i=0;i<len;i++)
56     {
57         int c=(int)s[i]-'a';
58         p=nxt[p][c];
59         for(int tp=p;tp;tp=fail[tp])
60             if(id[tp]) ret++;
61     }
62     return ret;
63 }

```

1.7.2 自动机上 DP

1.8 后缀自动机

2 计算几何

2.1 二维几何

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 #define db double
5 const db EPS=1e-9;
6 inline int sign(db a){return a<-EPS?-1:a>EPS;}
7 inline int cmp(db a,db b){return sign(a-b);}
8 struct P
9 {
10     db x,y;
11     P(){}
12     P(db x,db y):x(x),y(y){}
13     P operator+(P p){return {x+p.x,y+p.y};}
14     P operator-(P p){return {x-p.x,y-p.y};}
15     P operator*(db d){return {x*d,y*d};}
16     P operator/(db d){return {x/d,y/d};}
17     bool operator<(P p) const
18     {
19         int c=cmp(x,p.x);
20         if(c) return c==1;
21         return cmp(y,p.y)==1;
22     }
23     bool operator==(P o) const
24     {
25         return cmp(x,o.x)==0&&cmp(y,o.y)==0;
26     }
27     db distTo(P p){return (*this-p).abs();}
28     db alpha(){return atan2(y,x);}
29     void read(){scanf("%lf%lf",&x,&y);}
30     void write(){printf("%.10f,%.10f\n",x,y);}
31     db abs(){return sqrt(abs2());}
32     db abs2(){return x*x+y*y;}
33     P rot90(){return P(-y,x);}
34     P unit(){return *this/abs();}
35     int quad() const {return sign(y)==1||((sign(y)==0&&
36         sign(x)>=0);}
37     db dot(P p){return x*p.x+y*p.y;}
38     db det(P p){return x*p.y-y*p.x;}
39     P rot(db an){return {x*cos(an)-y*sin(an),x*sin(an)
40         +y*cos(an)}};

```

```

40 //For segment
41 #define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.
42     x-p1.x)*(p2.y-p1.y))
43 #define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
44
45 bool chkLL(P p1,P p2,P q1,P q2) //0:parallel
46 {
47     db a1=cross(q1,q2,p1),a2=-cross(q1,q2,p2);
48     return sign(a1+a2)!=0;
49 }
50
51 P isLL(P p1,P p2,P q1,P q2) //crossover point if
52     chkLL()
53 {
54     db a1=cross(q1,q2,p1),a2=-cross(q1,q2,p2);
55     return (p1*a2+p2*a1)/(a1+a2);
56 }
57
58 bool intersect(db l1,db r1,db l2,db r2)
59 {
60     if(l1>r1) swap(l1,r1);if(l2>r2) swap(l2,r2);
61     return !(cmp(r1,l2)==-1||cmp(r2,l1)==-1);
62 }
63
64 bool isSS(P p1,P p2,P q1,P q2)
65 {
66     return intersect(p1.x,p2.x,q1.x,q2.x)&&intersect(
67         p1.y,p2.y,q1.y,q2.y)&&
68         crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<=0&&crossOp(q1
69             ,q2,p1)*crossOp(q1,q2,p2)<=0;
70 }
71
72 bool isSS_strict(P p1,P p2,P q1,P q2)
73 {
74     return crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<0
75         &&crossOp(q1,q2,p1)*crossOp(q1,q2,p2)<0;
76 }
77
78 bool isMiddle(db a,db m,db b)
79 {
80     return sign(a-m)==0||sign(b-m)==0||((a<m!=b<m);
81 }
82
83 bool isMiddle(P a,P m,P b)
84 {
85     return isMiddle(a.x,m.x,b.x)&&isMiddle(a.y,m.y,b.y
86         );
87 }
88
89 bool onSeg(P p1,P p2,P q)
90 {
91     return crossOp(p1,p2,q)==0&&isMiddle(p1,q,p2);
92 }
93
94 bool onSeg_strict(P p1,P p2,P q)
95 {
96     return crossOp(p1,p2,q)==0&&sign((q-p1).dot(p1-p2)
97         )*sign((q-p2).dot(p1-p2))<0;
98 }
99
100 P proj(P p1,P p2,P q)
101 {
102     P dir=p2-p1;
103     return p1+dir*(dir.dot(q-p1)/dir.abs2());

```

```

99 }
100
101 P reflect(P p1,P p2,P q)
102 {
103     return proj(p1,p2,q)*2-q;
104 }
105
106 db nearest(P p1,P p2,P q)
107 {
108     P h=proj(p1,p2,q);
109     if(isMiddle(p1,h,p2))
110         return q.distTo(h);
111     return min(p1.distTo(q),p2.distTo(q));
112 }
113
114 db disSS(P p1,P p2,P q1,P q2) //dist of 2 segments
115 {
116     if(isSS(p1,p2,q1,q2)) return 0;
117     return min(min(nearest(p1,p2,q1),nearest(p1,p2,q2)),
118                min(nearest(q1,q2,p1),nearest(q1,q2,p2)));
119 }
120
121 db rad(P p1,P p2)
122 {
123     return atan2l(p1.det(p2),p1.dot(p2));
124 }
125
126 db area(vector<P> ps)
127 {
128     db ret=0;
129     for(int i=0;i<ps.size();i++)
130         ret+=ps[i].det(ps[(i+1)%ps.size()]);
131     return ret/2;
132 }
133
134 int contain(vector<P> ps,P p) //2:inside,1:on_seg,0:
135     outside
136 {
137     int n=ps.size(),ret=0;
138     for(int i=0;i<n;i++)
139     {
140         P u=ps[i],v=ps[(i+1)%n];
141         if(onSeg(u,v,p)) return 1;
142         if(cmp(u.y,v.y)<=0) swap(u,v);
143         if(cmp(p.y,u.y)>0||cmp(p.y,v.y)<=0) continue;
144         ret^=crossOp(p,u,v)>0;
145     }
146     return ret*2;
147 }
148
149 vector<P> convexHull(vector<P> ps)
150 {
151     int n=ps.size();if(n<=1) return ps;
152     sort(ps.begin(),ps.end());
153     vector<P> qs(n*2);int k=0;
154     for(int i=0;i<n;qs[k++]=ps[i++])
155         while(k>1&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
156             --k;
157     for(int i=n-2,t=k;i>=0;qs[k++]=ps[i--])
158         while(k>t&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
159             --k;
160     qs.resize(k-1);
161     return qs;
162 }

```

```

160 db convexDiameter(vector<P> ps)
161 {
162     int n=ps.size();if(n<=1) return 0;
163     int is=0,js=0;
164     for(int k=1;k<n;k++) is=ps[k]<ps[is]?k:is,js=ps[js]
165         <ps[k]?js:k;
166     int i=is,j=js;
167     db ret=ps[i].distTo(ps[j]);
168     do{
169         if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j])
170             >=0) (++j)%=n;
171         else (++i)%=n;
172         ret=max(ret,ps[i].distTo(ps[j]));
173     }while(i!=is||j!=js);
174     return ret;
175 }

```

2.2 三维几何

3 图论

3.1 最短路

3.1.1 Dijkstra

3.1.2 SPFA

3.1.3 Floyd

3.1.4 负环

3.1.5 差分约束

3.2 最小生成树

3.2.1 Prim

3.2.2 Kruskal

3.2.3 最小生成树计数

3.2.4 次小生成树

3.2.5 最小乘积生成树

3.3 树的直径

3.4 LCA

3.4.1 Tarjan 离线

3.4.2 倍增 LCA

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5     预处理 O(nlogn)
6     单次查询 O(logn)
7 */
8
9 const int MAXN=500005;
10 int n,q,dep[MAXN],s,lg[MAXN],fa[MAXN][32];
11 vector<int> e[MAXN];
12
13 void dfs(int now,int pa)
14 {
15     dep[now]=dep[pa]+1;
16     fa[now][0]=pa;

```

```

17     for(int i=1;(1<i)<=dep[now];i++)
18         fa[now][i]=fa[fa[now][i-1]][i-1];
19     for(auto to:e[now])
20         if(to!=pa) dfs(to,now);
21 }
22
23 int lca(int x,int y)
24 {
25     if(dep[x]<dep[y]) swap(x,y);
26     while(dep[x]>dep[y]) x=fa[x][lg[dep[x]-dep[y]]-1];
27     if(x==y) return x;
28     for(int i=lg[dep[x]]-1;i>=0;i--)
29         if(fa[x][i]!=fa[y][i])
30             x=fa[x][i],y=fa[y][i];
31     return fa[x][0];
32 }
33
34 int main()
35 {
36     for(int i=1;i<MAXN;i++)
37         lg[i]=lg[i-1]+(1<<lg[i-1]==i);
38     scanf("%d%d%d",&n,&q,&s);
39     for(int i=0,x,y;i<n-1;i++)
40     {
41         scanf("%d%d",&x,&y);
42         e[x].push_back(y),e[y].push_back(x);
43     }
44     dep[0]=0;
45     dfs(s,0);
46     for(int i=0,x,y;i<q;i++)
47     {
48         scanf("%d%d",&x,&y);
49         printf("%d\n",lca(x,y));
50     }
51     return 0;
52 }

```

3.5 无向图与有向图联通性

3.5.1 割点

3.5.2 桥

3.5.3 e-DCC

3.5.4 v-DCC

3.5.5 SCC

3.5.6 2-SAT

3.5.7 支配树

3.6 二分图

3.6.1 最大匹配-匈牙利

3.6.2 带权匹配-KM

3.7 网络流

3.7.1 最大流-Dinic

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 /*

```

```

6     s,t 超级源、超级汇
7     cur[] 当前弧优化
8     时间复杂度 O(n^2*m)
9 */
10
11 const int MAXN=10005;
12 const ll inf=0x3f3f3f3f3f3f3f11;
13 int n,m,s,t,tot,dis[MAXN],cur[MAXN];
14 struct edge
15 {
16     int to,cap,rev;
17     edge(){}
18     edge(int to,int cap,int rev):to(to),cap(cap),rev(
19         rev){}
20 };
21 vector<edge> E[MAXN];
22
23 inline void add_edge(int x,int y,int f)
24 {
25     E[x].emplace_back(y,f,E[y].size());
26     E[y].emplace_back(x,0,E[x].size()-1);
27 }
28
29 int bfs()
30 {
31     for(int i=1;i<=n;i++) dis[i]=0x3f3f3f3f;
32     dis[s]=0;
33     queue<int> q;
34     q.push(s);
35     while(!q.empty())
36     {
37         int now=q.front();q.pop();
38         for(int i=0;i<E[now].size();i++)
39         {
40             edge &e=E[now][i];
41             if(dis[e.to]>dis[now]+1&&e.cap)
42             {
43                 dis[e.to]=dis[now]+1;
44                 if(e.to==t) return 1;
45                 q.push(e.to);
46             }
47         }
48     }
49     return 0;
50 }
51
52 ll dfs(int now,ll flow)
53 {
54     if(now==t) return flow;
55     ll rest=flow,k;
56     for(int i=cur[now];i<E[now].size();i++)
57     {
58         edge &e=E[now][i];
59         if(e.cap&&dis[e.to]==dis[now]+1)
60         {
61             cur[now]=i;
62             k=dfs(e.to,min(rest,(long long)e.cap));
63             e.cap-=k;
64             E[e.to][e.rev].cap+=k;
65             rest-=k;
66         }
67     }
68     return flow-rest;
69 }

```

```

70 ll dinic()
71 {
72     ll ret=0,delta;
73     while(bfs())
74     {
75         for(int i=1;i<=n;i++) cur[i]=0;
76         while(delta=dfs(s,inf)) ret+=delta;
77     }
78     return ret;
79 }

```

3.7.2 最小费用最大流-Dij+Dinic

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef pair<int,int> pii;
4
5  /*
6   第一遍跑的spfa,然后是加上势函数的dij,玄学
7   h[] 势函数
8   cur[] 当前弧优化
9   msmf 最大流时的最小费用
10  s,t 超级源、超级汇
11  时间复杂度 O(n^2*m)
12  */
13
14 const int MAXN=2005;
15 const int inf=0x3f3f3f3f;
16 int msmf,s,t,cur[MAXN],dis[MAXN],vis[MAXN],h[MAXN];
17 struct edge
18 {
19     int to,val,cap,rev;
20     edge(){}
21     edge(int to,int cap,int val,int rev):to(to),cap(
22         cap),val(val),rev(rev){}
23 };
24 vector<edge> E[MAXN];
25
26 inline void add_edge(int x,int y,int f,int cost)
27 {
28     E[x].emplace_back(y,f,cost,E[y].size());
29     E[y].emplace_back(x,0,-cost,E[x].size()-1);
30 }
31
32 int dij()
33 {
34     fill(dis,dis+t+1,inf);
35     priority_queue<pii,vector<pii>,greater<pii>> q;
36     q.emplace(0,s);dis[s]=0;
37     while(!q.empty())
38     {
39         pii p=q.top();q.pop();
40         int now=p.second;
41         if(dis[now]<p.first) continue;
42         for(int i=0;i<E[now].size();i++)
43         {
44             edge &e=E[now][i];
45             if(e.cap>0&&dis[e.to]>p.first+e.val+h[now]-
46                 h[e.to])
47             {
48                 dis[e.to]=p.first+e.val+h[now]-h[e.to];
49                 q.emplace(dis[e.to],e.to);
50             }
51         }
52     }
53 }

```

```

50 }
51 return dis[t]!=inf;
52 }
53
54 int dfs(int now,int flow)
55 {
56     if(now==t) return flow;
57     int rest=flow,k;
58     vis[now]=1;
59     for(int i=cur[now];i<E[now].size();i++)
60     {
61         edge &e=E[now][i];
62         if(e.cap&&dis[now]+e.val+h[now]-h[e.to]==dis[
63             e.to]&&!vis[e.to])
64         {
65             cur[now]=i;
66             k=dfs(e.to,min(e.cap,rest));
67             e.cap-=k;
68             E[e.to][e.rev].cap+=k;
69             rest-=k;
70             msmf+=k*e.val;
71         }
72     }
73     vis[now]=0;
74     return flow-rest;
75 }
76
77 int dinic()
78 {
79     int ret=0,delta;
80     while(dij())
81     {
82         for(int i=s;i<=t;i++) cur[i]=0;
83         while(delta=dfs(s,inf)) ret+=delta;
84         for(int i=s;i<=t;i++) h[i]+=(dis[i]==inf)?0:
85             dis[i];
86     }
87     return ret;
88 }

```

3.7.3 上下界流

3.8 欧拉路

3.9 Prufer 序列

4 数据结构

4.1 树状数组

4.2 线段树

4.2.1 带优先级线段树

4.2.2 吉司机线段树

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  /*
6   modify 将区间大于x的数变成x
7   query 询问区间和
8   单次复杂度 O(log^2(n))
9  */

```



```

10
11 const ll INF=0xc0c0c0c0c0c0c0c0ll;
12 const int MAXN=200005;
13 ll seg[MAXN<<2],m1[MAXN<<2],m2[MAXN<<2],cnt[MAXN<<2],
    tag[MAXN<<2],a[MAXN];
14 int n,q;
15
16 void pushdown(int rt)
17 {
18     if(!tag[rt]) return;
19     ll y=m1[rt];
20     if(y<m1[rt<<1])
21     {
22         tag[rt<<1]=1;
23         seg[rt<<1]-=(m1[rt<<1]-y)*cnt[rt<<1];
24         m1[rt<<1]=y;
25     }
26     if(y<m1[rt<<1|1])
27     {
28         tag[rt<<1|1]=1;
29         seg[rt<<1|1]-=(m1[rt<<1|1]-y)*cnt[rt<<1|1];
30         m1[rt<<1|1]=y;
31     }
32     tag[rt]=0;
33 }
34
35 void pushup(int rt)
36 {
37     seg[rt]=seg[rt<<1]+seg[rt<<1|1];
38     if(m1[rt<<1]==m1[rt<<1|1])
39     {
40         m1[rt]=m1[rt<<1];
41         cnt[rt]=cnt[rt<<1]+cnt[rt<<1|1];
42         m2[rt]=max(m2[rt<<1],m2[rt<<1|1]);
43     }
44     else if(m1[rt<<1]>m1[rt<<1|1])
45     {
46         m1[rt]=m1[rt<<1];
47         cnt[rt]=cnt[rt<<1];
48         m2[rt]=max(m2[rt<<1],m1[rt<<1|1]);
49     }
50     else
51     {
52         m1[rt]=m1[rt<<1|1];
53         cnt[rt]=cnt[rt<<1|1];
54         m2[rt]=max(m2[rt<<1|1],m1[rt<<1]);
55     }
56 }
57
58 void build(int rt,int l,int r)
59 {
60     tag[rt]=0;
61     if(l==r)
62     {
63         seg[rt]=m1[rt]=a[l];
64         cnt[rt]=1;
65         m2[rt]=INF;
66         return;
67     }
68     int m=l+r>>1;
69     if(l<=m) build(rt<<1,l,m);
70     if(m<r) build(rt<<1|1,m+1,r);
71     pushup(rt);
72 }
73

```

```

74 void modify(int rt,int l,int r,int L,int R,ll y)
75 {
76     if(y>m1[rt]) return;
77     if(L<=l&&r<=R&&y>m2[rt])
78     {
79         tag[rt]=1;
80         seg[rt]-=(m1[rt]-y)*cnt[rt];
81         m1[rt]=y;
82         return;
83     }
84     pushdown(rt);
85     int m=l+r>>1;
86     if(L<=m) modify(rt<<1,l,m,L,R,y);
87     if(m<R) modify(rt<<1|1,m+1,r,L,R,y);
88     pushup(rt);
89 }
90
91 ll query(int rt,int l,int r,int L,int R)
92 {
93     if(L<=l&&r<=R) return seg[rt];
94     int m=l+r>>1;
95     pushdown(rt);
96     ll ret=0;
97     if(L<=m) ret+=query(rt<<1,l,m,L,R);
98     if(m<R) ret+=query(rt<<1|1,m+1,r,L,R);
99     pushup(rt);
100     return ret;
101 }

```

4.2.3 线段树维护扫描线

4.3 RMQ

4.3.1 一维

4.3.2 两维

4.4 树链剖分

4.4.1 点剖分

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 /*
6     轻重链剖分 单次复杂度  $O(\log^2(n))$ 
7     a[i] 表示dfs标号为i的点的值,而非点i的值
8     1 x y z 表示将树从x到y结点最短路径上所有节点值都加上z
9     2 x y 表示求树从x到y结点最短路径上所有节点值之和
10    3 x z 表示将以x为根节点的子树内所有节点值都加上z
11    4 x 表示求以x为根节点的子树内所有节点值之和
12 */
13
14 const int MAXN=100005;
15 ll mod,lazy[MAXN<<2],seg[MAXN<<2],a[MAXN],tmp[MAXN];
16 int n,q,r,cnt,tot,dep[MAXN],top[MAXN],id[MAXN],son[
    MAXN],num[MAXN],fa[MAXN];
17 vector<int> e[MAXN];
18
19 void dfs1(int now,int f)
20 {
21     dep[now]=dep[f]+1;
22     fa[now]=f;
23     num[now]=1;
24     son[now]=0;

```

```

25     for(auto to:e[now])
26     {
27         if(to==f) continue;
28         dfs1(to,now);
29         num[now]+=num[to];
30         if(num[to]>num[son[now]]) son[now]=to;
31     }
32 }
33
34 void dfs2(int now,int f)
35 {
36     id[now]=++cnt;
37     top[now]=f;
38     if(son[now]) dfs2(son[now],f);
39     for(auto to:e[now])
40         if(to!=fa[now]&&to!=son[now])
41             dfs2(to,to);
42 }
43
44 inline void pushdown(int rt,ll lnum,ll rnum)
45 {
46     if(!lazy[rt]) return;
47     seg[rt<<1]=(seg[rt<<1]+lazy[rt]*lnum%mod)%mod;
48     seg[rt<<1|1]=(seg[rt<<1|1]+lazy[rt]*rnum%mod)%mod;
49     lazy[rt<<1]=(lazy[rt<<1]+lazy[rt])%mod;
50     lazy[rt<<1|1]=(lazy[rt<<1|1]+lazy[rt])%mod;
51     lazy[rt]=0;
52 }
53
54 inline void pushup(int rt)
55 {
56     seg[rt]=(seg[rt<<1]+seg[rt<<1|1])%mod;
57 }
58
59 void build(int rt,int l,int r)
60 {
61     lazy[rt]=0;
62     if(l==r)
63     {
64         seg[rt]=a[l]%mod;
65         return;
66     }
67     int m=l+r>>1;
68     if(l<=m) build(rt<<1,l,m);
69     if(m<r) build(rt<<1|1,m+1,r);
70     pushup(rt);
71 }
72
73 void modify(int rt,int l,int r,int L,int R,ll x)
74 {
75     if(L<=l&&r<=R)
76     {
77         lazy[rt]=(lazy[rt]+x)%mod;
78         seg[rt]=(seg[rt]+x*(r-l+1)%mod)%mod;
79         return;
80     }
81     int m=l+r>>1;
82     pushdown(rt,m-l+1,r-m);
83     if(L<=m) modify(rt<<1,l,m,L,R,x);
84     if(m<R) modify(rt<<1|1,m+1,r,L,R,x);
85     pushup(rt);
86 }
87
88 ll query(int rt,int l,int r,int L,int R)
89 {

```

```

90     if(L<=l&&r<=R) return seg[rt];
91     int m=l+r>>1;
92     ll ret=0;
93     pushdown(rt,m-l+1,r-m);
94     if(L<=m) ret=(ret+query(rt<<1,l,m,L,R))%mod;
95     if(m<R) ret=(ret+query(rt<<1|1,m+1,r,L,R))%mod;
96     pushup(rt);
97     return ret;
98 }
99
100 int main()
101 {
102     scanf("%d%d%d%lld",&n,&q,&r,&mod);
103     for(int i=1;i<=n;i++) scanf("%lld",&tmp[i]);
104     for(int i=1,x,y;i<=n;i++)
105     {
106         scanf("%d%d",&x,&y);
107         e[x].push_back(y),e[y].push_back(x);
108     }
109     num[0]=0,dep[r]=0;
110     dfs1(r,r);
111     dfs2(r,r);
112     for(int i=1;i<=n;i++) a[id[i]]=tmp[i];
113     build(1,1,n);
114
115     while(q--)
116     {
117         int op,x,y,ll z;
118         scanf("%d%d",&op,&x);
119         if(op==4)
120         {
121             printf("%lld\n",query(1,1,n,id[x],id[x]+num
122                 [x]-1));
123             continue;
124         }
125         if(op==1)
126         {
127             scanf("%d%lld",&y,&z);z%=mod;
128             while(top[x]!=top[y])
129             {
130                 if(dep[top[x]]<dep[top[y]]) swap(x,y);
131                 modify(1,1,n,id[top[x]],id[x],z);
132                 x=fa[top[x]];
133             }
134             if(dep[x]>dep[y]) swap(x,y);
135             modify(1,1,n,id[x],id[y],z);
136         }
137         else if(op==2)
138         {
139             scanf("%d",&y);
140             ll ans=0;
141             while(top[x]!=top[y])
142             {
143                 if(dep[top[x]]<dep[top[y]]) swap(x,y);
144                 ans=(ans+query(1,1,n,id[top[x]],id[x]))%
145                     mod;
146                 x=fa[top[x]];
147             }
148             if(dep[x]>dep[y]) swap(x,y);
149             ans=(ans+query(1,1,n,id[x],id[y]))%mod;
150             printf("%lld\n",ans);
151         }
152         else
153         {
154             scanf("%lld",&z);z%=mod;

```

```

153         modify(1,1,n,id[x],id[x]+num[x]-1,z);
154     }
155 }
156 return 0;
157 }

```

4.4.2 边剖分

4.5 平衡树

4.5.1 Treap

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAXN=1e5+5;
5 const int inf=0x7fffffff;
6 int n,op,x;
7
8 /*
9  树内初始化时有无穷大和无穷小两个结点
10  _delete(root,x) 删除一个x
11  _insert(root,x) 插入一个x
12  getRank(root,x) 返回x的排名+1(包含了无穷小)
13  getVal(root,x+1) 返回排名为x的数
14  getPrev(x) x的前驱
15  getNext(x) x的后继
16 */
17
18 namespace Treap
19 {
20     int tot,root;
21     struct node
22     {
23         int cnt,val,dat,siz,lc,rc;
24     }bst[MAXN];
25
26     inline void pushup(int rt)
27     {
28         bst[rt].siz=bst[rt].cnt;
29         if(bst[rt].lc) bst[rt].siz+=bst[bst[rt].lc].siz;
30         if(bst[rt].rc) bst[rt].siz+=bst[bst[rt].rc].siz;
31     }
32
33     inline void zig(int &rt)
34     {
35         int p=bst[rt].lc;
36         bst[rt].lc=bst[p].rc;
37         bst[p].rc=rt;
38         rt=p;
39         pushup(bst[rt].rc);pushup(rt);
40     }
41
42     inline void zag(int &rt)
43     {
44         int p=bst[rt].rc;
45         bst[rt].rc=bst[p].lc;
46         bst[p].lc=rt;
47         rt=p;
48         pushup(bst[rt].lc);pushup(rt);
49     }
50
51     int new_node(int val)

```

```

52     {
53         bst[++tot].val=val;
54         bst[tot].dat=rand();
55         bst[tot].siz=bst[tot].cnt=1;
56         bst[tot].lc=bst[tot].rc=0;
57         return tot;
58     }
59
60     void build()
61     {
62         new_node(-inf);new_node(inf);
63         root=1,bst[1].rc=2;
64         pushup(1);
65     }
66
67     void _insert(int &rt,int val)
68     {
69         if(rt==0)
70         {
71             rt=new_node(val);
72             return;
73         }
74         if(bst[rt].val==val)
75         {
76             bst[rt].cnt++;
77             pushup(rt);
78             return;
79         }
80         if(val<bst[rt].val)
81         {
82             _insert(bst[rt].lc,val);
83             if(bst[rt].dat<bst[bst[rt].lc].dat) zig(rt);
84         }
85         else
86         {
87             _insert(bst[rt].rc,val);
88             if(bst[rt].dat<bst[bst[rt].rc].dat) zag(rt);
89         }
90         pushup(rt);
91     }
92
93     void _delete(int &rt,int val)
94     {
95         if(rt==0) return;
96         if(bst[rt].val==val)
97         {
98             if(bst[rt].cnt>1)
99             {
100                 bst[rt].cnt--;
101                 pushup(rt);
102                 return;
103             }
104             if(bst[rt].rc||bst[rt].lc)
105             {
106                 if(bst[rt].rc==0||bst[bst[rt].rc].dat<bst[bst[rt].lc].dat)
107                     zig(rt),_delete(bst[rt].rc,val);
108                 else
109                     zag(rt),_delete(bst[rt].lc,val);
110                 pushup(rt);
111             }
112             else rt=0;
113         }

```

```

114     return;
115 }
116 if(val<bst[rt].val) _delete(bst[rt].lc,val);
117 else _delete(bst[rt].rc,val);
118 pushup(rt);
119 }
120
121 int getPrev(int val)
122 {
123     int ret=1,rt=root;
124     while(rt)
125     {
126         if(bst[rt].val==val)
127         {
128             if(bst[rt].lc)
129             {
130                 rt=bst[rt].lc;
131                 while(bst[rt].rc) rt=bst[rt].rc;
132                 ret=rt;
133             }
134             break;
135         }
136         if(bst[rt].val<val&&bst[rt].val>bst[ret].val) ret=rt;
137         if(val<bst[rt].val) rt=bst[rt].lc;
138         else rt=bst[rt].rc;
139     }
140     return bst[ret].val;
141 }
142
143 int getNext(int val)
144 {
145     int ret=2,rt=root;
146     while(rt)
147     {
148         if(bst[rt].val==val)
149         {
150             if(bst[rt].rc)
151             {
152                 rt=bst[rt].rc;
153                 while(bst[rt].lc) rt=bst[rt].lc;
154                 ret=rt;
155             }
156             break;
157         }
158         if(bst[rt].val>val&&bst[rt].val<bst[ret].val) ret=rt;
159         if(val<bst[rt].val) rt=bst[rt].lc;
160         else rt=bst[rt].rc;
161     }
162     return bst[ret].val;
163 }
164
165 int getRank(int rt,int val)
166 {
167     if(rt==0) return 0;
168     if(val==bst[rt].val) return bst[bst[rt].lc].siz+1;
169     if(val<bst[rt].val) return getRank(bst[rt].lc,val);
170     else return bst[bst[rt].lc].siz+bst[rt].cnt+getRank(bst[rt].rc,val);
171 }
172
173 int getVal(int rt,int k)

```

```

174 {
175     if(rt==0) return inf;
176     if(bst[bst[rt].lc].siz>=k) return getVal(bst[rt].lc,k);
177     if(bst[bst[rt].lc].siz+bst[rt].cnt>=k) return bst[rt].val;
178     return getVal(bst[rt].rc,k-bst[bst[rt].lc].siz-bst[rt].cnt);
179 }
180 }
181
182 int main()
183 {
184     using namespace Treap;
185     srand(time(0));
186     build();
187     scanf("%d",&n);
188     while(n-->0)
189     {
190         scanf("%d%d",&op,&x);
191         if(op==1) _insert(root,x);
192         else if(op==2) _delete(root,x);
193         else if(op==3) printf("%d\n",getRank(root,x)-1);
194         else if(op==4) printf("%d\n",getVal(root,x+1));
195         else if(op==5) printf("%d\n",getPrev(x));
196         else if(op==6) printf("%d\n",getNext(x));
197     }
198     return 0;
199 }

```

4.5.2 Splay

4.6 动态树

4.7 主席树

4.8 树套树

4.8.1 线段树套 Treap

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5     空间 O(nlogn)
6     单点修改,区间rank,前驱后继(不存在则为±2147483647) 单
7     次 O(log^2(n))
8     区间排名为k的值 单次 O(log^3(n))
9 */
10
11 const int inf=2147483647;
12 const int MAXN=50005;
13 int root[MAXN<<2],n,m,a[MAXN];
14 struct Treap
15 {
16     int tot;
17     struct node
18     {
19         int lc,rc,dat,val,cnt,siz;
20     }bst[MAXN*4*20];
21
22     int newnode(int v)
23     {
24         tot++;
25         bst[tot].val=bst[tot].dat=v;
26         bst[tot].lc=bst[tot].rc=0;
27         bst[tot].cnt=bst[tot].siz=1;
28         return tot;
29     }
30
31     void pushup(int rt)
32     {
33         bst[rt].siz=bst[bst[rt].lc].siz+bst[bst[rt].rc].siz+bst[rt].cnt;
34     }
35
36     void _insert(int &rt,int x)
37     {
38         if(!rt) rt=newnode(x);
39         else if(x<bst[rt].val) _insert(bst[rt].lc,x);
40         else if(x>bst[rt].val) _insert(bst[rt].rc,x);
41         else bst[rt].cnt++;
42         pushup(rt);
43     }
44
45     void _delete(int &rt,int x)
46     {
47         if(!rt) return;
48         if(x<bst[rt].val) _delete(bst[rt].lc,x);
49         else if(x>bst[rt].val) _delete(bst[rt].rc,x);
50         else
51         {
52             if(!bst[rt].lc) bst[rt].rc=rt=bst[rt].rc;
53             else if(!bst[rt].rc) bst[rt].lc=rt=bst[rt].lc;
54             else
55             {
56                 int p=q;
57                 while(bst[p].lc!=0) p=bst[p].lc;
58                 _delete(bst[rt].lc,bst[p].val);
59                 bst[rt].lc=p;
60                 pushup(rt);
61             }
62         }
63     }
64
65     int getRank(int rt,int x)
66     {
67         if(!rt) return 0;
68         if(x<bst[rt].val) return getRank(bst[rt].lc,x);
69         if(x>bst[rt].val) return getRank(bst[rt].rc,x)+bst[rt].cnt+1;
70         return bst[rt].cnt+1;
71     }
72
73     int getVal(int rt,int k)
74     {
75         if(!rt) return inf;
76         if(k>bst[rt].cnt+bst[bst[rt].lc].siz) return getVal(bst[rt].rc,k-bst[rt].cnt-bst[bst[rt].lc].siz);
77         if(k<=bst[bst[rt].lc].siz) return getVal(bst[rt].lc,k);
78         return bst[rt].val;
79     }
80 }
81
82 int main()
83 {
84     int n,m;
85     scanf("%d%d",&n,&m);
86     for(int i=1;i<=n;i++) scanf("%d",&a[i]);
87     Treap t;
88     for(int i=1;i<=m;i++)
89     {
90         int op,x;
91         scanf("%d%d",&op,&x);
92         if(op==1) t._insert(root,x);
93         else if(op==2) t._delete(root,x);
94         else if(op==3) printf("%d\n",t.getRank(root,x)-1);
95         else if(op==4) printf("%d\n",t.getVal(root,x+1));
96         else if(op==5) printf("%d\n",t.getPrev(x));
97         else if(op==6) printf("%d\n",t.getNext(x));
98     }
99 }

```

```

23     bst[++tot].val=v;
24     bst[tot].dat=rand();
25     bst[tot].siz=bst[tot].cnt=1;
26     bst[tot].lc=bst[tot].rc=0;
27     return tot;
28 }
29
30 void zig(int &rt)
31 {
32     int p=bst[rt].lc;
33     bst[rt].lc=bst[p].rc;
34     bst[p].rc=rt;
35     rt=p;
36     pushup(bst[rt].rc);
37     pushup(rt);
38 }
39
40 void zag(int &rt)
41 {
42     int p=bst[rt].rc;
43     bst[rt].rc=bst[p].lc;
44     bst[p].lc=rt;
45     rt=p;
46     pushup(bst[rt].lc);
47     pushup(rt);
48 }
49
50 void pushup(int rt)
51 {
52     bst[rt].siz=bst[rt].cnt;
53     if(bst[rt].lc) bst[rt].siz+=bst[bst[rt].lc].
54         siz;
55     if(bst[rt].rc) bst[rt].siz+=bst[bst[rt].rc].
56         siz;
57 }
58
59 int build()
60 {
61     int rt=newnode(-inf);
62     bst[rt].rc=newnode(inf);
63     pushup(rt);
64     return rt;
65 }
66
67 void _delete(int &rt,int x)
68 {
69     if(bst[rt].val==x)
70     {
71         if(bst[rt].cnt>1)
72         {
73             bst[rt].cnt--;
74             pushup(rt);
75             return;
76         }
77         if(bst[rt].lc||bst[rt].rc)
78         {
79             if(bst[rt].rc==0||bst[bst[rt].rc].dat<
80                 bst[bst[rt].lc].dat)
81                 zig(rt),_delete(bst[rt].rc,x);
82             else
83                 zag(rt),_delete(bst[rt].lc,x);
84             pushup(rt);
85         }
86         else rt=0;
87         return;
88     }

```

```

85     }
86     if(x<bst[rt].val) _delete(bst[rt].lc,x);
87     else _delete(bst[rt].rc,x);
88     pushup(rt);
89 }
90
91 void _insert(int &rt,int x)
92 {
93     if(rt==0)
94     {
95         rt=newnode(x);
96         return;
97     }
98     if(bst[rt].val==x) bst[rt].cnt++;
99     else if(x<bst[rt].val)
100     {
101         _insert(bst[rt].lc,x);
102         if(bst[bst[rt].lc].dat>bst[rt].dat) zig(rt)
103             ;
104     }
105     else
106     {
107         _insert(bst[rt].rc,x);
108         if(bst[bst[rt].rc].dat>bst[rt].dat) zag(rt)
109             ;
110     }
111     pushup(rt);
112 }
113
114 int get_rank(int rt,int x)
115 {
116     if(!rt) return 1;
117     if(bst[rt].val==x) return bst[bst[rt].lc].siz
118         +1;
119     if(x<bst[rt].val) return get_rank(bst[rt].lc,x
120         );
121     else return get_rank(bst[rt].rc,x)+bst[bst[rt]
122         ].lc].siz+bst[rt].cnt;
123 }
124
125 int get_num(int rt,int x)
126 {
127     if(!rt) return 0;
128     if(bst[rt].val==x) return bst[bst[rt].lc].siz+
129         bst[rt].cnt;
130     if(x<bst[rt].val) return get_num(bst[rt].lc,x)
131         ;
132     else return get_num(bst[rt].rc,x)+bst[bst[rt].
133         lc].siz+bst[rt].cnt;
134 }
135
136 int get_prev(int rt,int x)
137 {
138     int ret=-inf;
139     while(rt)
140     {
141         if(bst[rt].val==x)
142         {
143             if(bst[rt].lc)
144             {
145                 rt=bst[rt].lc;
146                 while(bst[rt].rc) rt=bst[rt].rc;
147                 ret=bst[rt].val;
148             }
149             break;
150         }

```

```

142     }
143     if(bst[rt].val<x&&bst[rt].val>ret) ret=bst[
        rt].val;
144     if(x<bst[rt].val) rt=bst[rt].lc;
145     else rt=bst[rt].rc;
146 }
147 return ret;
148 }
149
150 int get_nxt(int rt,int x)
151 {
152     int ret=inf;
153     while(rt)
154     {
155         if(bst[rt].val==x)
156         {
157             if(bst[rt].rc)
158             {
159                 rt=bst[rt].rc;
160                 while(bst[rt].lc) rt=bst[rt].lc;
161                 ret=bst[rt].val;
162             }
163             break;
164         }
165         if(bst[rt].val>x&&bst[rt].val<ret) ret=bst[
            rt].val;
166         if(x<bst[rt].val) rt=bst[rt].lc;
167         else rt=bst[rt].rc;
168     }
169     return ret;
170 }
171 }treap;
172
173 void build(int rt,int l,int r)
174 {
175     root[rt]=treap.build();
176     if(l==r) return;
177     int m=l+r>>1;
178     build(rt<<1,l,m);
179     build(rt<<1|1,m+1,r);
180 }
181
182 void modify(int rt,int l,int r,int x,int v,int y)
183 {
184     if(y==-1) treap._delete(root[rt],v);
185     else treap._insert(root[rt],v);
186     if(l==r) return;
187     int m=l+r>>1;
188     if(x<=m) modify(rt<<1,l,m,x,v,y);
189     else modify(rt<<1|1,m+1,r,x,v,y);
190 }
191
192 int query(int rt,int l,int r,int op,int L,int R,int x
        )
193 {
194     if(L<=l&&r<=R)
195     {
196         if(op==1) return treap.get_rank(root[rt],x)-2;
197         if(op==2) return treap.get_num(root[rt],x)-1;
198         if(op==4) return treap.get_prev(root[rt],x);
199         if(op==5) return treap.get_nxt(root[rt],x);
200     }
201     int m=l+r>>1,ret;
202     if(op==1||op==2)
203     {

```

```

204         ret=0;
205         if(L<=m) ret+=query(rt<<1,l,m,op,L,R,x);
206         if(m<R) ret+=query(rt<<1|1,m+1,r,op,L,R,x);
207     }
208     if(op==4)
209     {
210         ret=-inf;
211         if(L<=m) ret=max(ret,query(rt<<1,l,m,op,L,R,x)
            );
212         if(m<R) ret=max(ret,query(rt<<1|1,m+1,r,op,L,R
            ,x));
213     }
214     if(op==5)
215     {
216         ret=inf;
217         if(L<=m) ret=min(ret,query(rt<<1,l,m,op,L,R,x)
            );
218         if(m<R) ret=min(ret,query(rt<<1|1,m+1,r,op,L,R
            ,x));
219     }
220     return ret;
221 }
222
223 int main()
224 {
225     srand(time(0));
226     scanf("%d%d",&n,&m);
227     build(1,1,n);
228     for(int i=1;i<=n;i++)
229     {
230         scanf("%d",a+i);
231         modify(1,1,n,i,a[i],1);
232     }
233     while(m--)
234     {
235         int op,l,r,k,pos;
236         scanf("%d",&op);
237         if(op==1)
238         {
239             scanf("%d%d%d",&l,&r,&k);
240             printf("%d\n",query(1,1,n,op,l,r,k)+1);
241         }
242         else if(op==2)
243         {
244             scanf("%d%d%d",&l,&r,&k);
245             int L=-inf,R=inf,mid;
246             while(L<R)
247             {
248                 mid=(L+R+1)>>1;
249                 if(query(1,1,n,1,l,r,mid)+1>k) R=mid-1;
250                 else L=mid;
251             }
252             printf("%d\n",L);
253         }
254         else if(op==3)
255         {
256             scanf("%d%d",&pos,&k);
257             modify(1,1,n,pos,a[pos],-1);
258             a[pos]=k;
259             modify(1,1,n,pos,k,1);
260         }
261         else
262         {
263             scanf("%d%d%d",&l,&r,&k);
264             printf("%d\n",query(1,1,n,op,l,r,k));

```

```

264     }
265 }
266 return 0;
267 }

```

```

56 c1=c2=0;
57 for(int i=1-1;i-=i&-i) X[c1++]=seg.root[i];
58 for(int i=r;i-=i&-i) Y[c2++]=seg.root[i];
59 return seg.query(1,VAL,k);
60 }

```

4.8.2 树状数组套线段树

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5 带单点修区间k小
6 用的时候注意下空间 时空 O(nlog^2(n))
7 外层 add(pos,x,y) 空间上为pos的点且值域上为x的点加上y
8 query(l,r,k) 询问区间[l,r]里k小
9 内层 modify 值域线段树动态开点
10 query 值域线段树区间k小
11 VAL 值域大小
12 */
13
14 const int MAXN=200005;
15 int n,a[MAXN],X[MAXN],Y[MAXN],c1,c2,VAL;
16 struct SEG
17 {
18     int root[MAXN],lc[MAXN*500],rc[MAXN*500],cnt[MAXN
19         *500],tot;
20     void modify(int &rt,int l,int r,int x,int y)
21     {
22         if(rt==0) rt=++tot;
23         cnt[rt]+=y;
24         if(l==r) return;
25         int m=l+r>>1;
26         if(x<=m) modify(lc[rt],l,m,x,y);
27         else modify(rc[rt],m+1,r,x,y);
28     }
29     int query(int l,int r,int k)
30     {
31         if(l==r) return l;
32         int sum=0,m=l+r>>1;
33         for(int i=0;i<c1;i++) sum-=cnt[lc[X[i]]];
34         for(int i=0;i<c2;i++) sum+=cnt[lc[Y[i]]];
35         if(sum>=k)
36         {
37             for(int i=0;i<c1;i++) X[i]=lc[X[i]];
38             for(int i=0;i<c2;i++) Y[i]=lc[Y[i]];
39             return query(l,m,k);
40         }
41         else
42         {
43             for(int i=0;i<c1;i++) X[i]=rc[X[i]];
44             for(int i=0;i<c2;i++) Y[i]=rc[Y[i]];
45             return query(m+1,r,k-sum);
46         }
47     }
48 }seg;
49
50 void add(int pos,int x,int y)
51 {
52     for(;pos<=n;pos+=pos&-pos) seg.modify(seg.root[pos
53         ],1,VAL,x,y);
54 }
55
56 int query(int l,int r,int k)
57 {
58

```

4.9 K-D Tree

4.10 分治

4.10.1 CDQ

4.10.2 点分治

4.10.3 dsu on tree

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 /*
6 统计每颗子树内的出现次数最多的数(们)的和
7 复杂度 O(nlogn)
8 */
9
10 int n,c[100005],cnt[100005],mx,son[100005],siz
11     [100005],hson;
12 ll ans[100005],sum;
13 vector<int> e[100005];
14
15 void dfs1(int now,int fa)
16 {
17     son[now]=0,siz[now]=1;
18     for(auto to:e[now])
19     {
20         if(to==fa) continue;
21         dfs1(to,now);
22         siz[now]+=siz[to];
23         if(siz[to]>siz[son[now]]) son[now]=to;
24     }
25
26     void cal(int now,int fa,int y)
27     {
28         cnt[c[now]]+=y;
29         if(cnt[c[now]]==mx) sum+=c[now];
30         else if(cnt[c[now]]>mx) mx=cnt[c[now]],sum=c[now];
31         for(auto to:e[now])
32             if(to!=fa&&to!=hson) cal(to,now,y);
33     }
34
35     void dfs2(int now,int fa,int keep)
36     {
37         for(auto to:e[now])
38         {
39             if(to==fa||to==son[now]) continue;
40             dfs2(to,now,0);
41         }
42         if(son[now]) dfs2(son[now],now,1);
43         hson=son[now];
44         cal(now,fa,1);
45         hson=0;
46         ans[now]=sum;
47         if(!keep) cal(now,fa,-1),sum=0,mx=0;
48     }
49

```

```

50 int main()
51 {
52     scanf("%d",&n);
53     for(int i=1;i<=n;i++) scanf("%d",c+i);
54     for(int i=1,x,y;i<n;i++)
55     {
56         scanf("%d%d",&x,&y);
57         e[x].push_back(y),e[y].push_back(x);
58     }
59     dfs1(1,1);
60     dfs2(1,1,1);
61     for(int i=1;i<=n;i++) printf("%lld ",ans[i]);
62     return 0;
63 }

```

4.10.4 整体二分

4.11 分块

4.11.1 普通分块

4.11.2 莫队

4.12 线性基

4.13 珂朵莉树

4.14 跳舞链

5 动态规划

5.1 SOS

```

1 for(int i=0;i<(1<<N);i++) dp[i]=a[i];
2 for(int i=0;i<N;i++)
3     for(int mask=0;mask<(1<<N);mask++)
4         if(mask&(1<<i))
5             dp[mask]+=dp[mask^(1<<i)];

```

5.2 动态 DP

5.3 插头 DP

6 数学

6.1 矩阵类

6.2 质数筛

6.2.1 埃筛

6.2.2 线筛

6.3 质数判定

6.3.1 Miller Rabin

6.4 质因数分解

6.4.1 Pollard-Rho

6.5 逆元

6.5.1 EX-GCD 求逆元

6.5.2 线性筛逆元

6.5.3 阶乘逆元

6.6 欧拉函数

6.6.1 欧拉线筛

6.6.2 求单个数的欧拉函数

6.6.3 欧拉降幂

6.6.4 一般积性函数求法

6.7 EX-GCD

6.8 CRT

6.9 N 次剩余

6.10 数论分块

6.11 高斯消元

6.11.1 普通消元

6.11.2 异或方程组消元

6.12 莫比乌斯反演

6.12.1 莫比乌斯函数

6.12.2 杜教筛

6.12.3 洲阁筛

6.12.4 min25 筛

6.13 BSGS

6.14 FFT

6.15 FWT

6.16 NTT

6.17 数值计算

6.17.1 辛普森

6.17.2 自适应辛普森