

小小青蛙听风就是雨

HiedanoAkyuu、Oneman233、KR12138

2019 年 10 月 19 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	2
1.6	回文自动机	2
1.7	AC 自动机	2
1.7.1	多模匹配	2
1.7.2	自动机上 DP	3
1.8	后缀自动机	3
2	计算几何	3
2.1	二维几何	3
2.2	三维几何	4
3	图论	4
3.1	最短路	4
3.1.1	Dijkstra	4
3.1.2	SPFA	4
3.1.3	Floyd	4
3.1.4	负环	4
3.1.5	差分约束	4
3.2	最小生成树	4
3.2.1	Prim	4
3.2.2	Kruskal	4
3.2.3	最小生成树计数	4
3.2.4	次小生成树	4
3.2.5	最小乘积生成树	4
3.3	树的直径	4
3.4	LCA	4
3.4.1	Tarjan 离线	4
3.4.2	倍增 LCA	4
3.5	无向图与有向图联通性	5
3.5.1	割点	5
3.5.2	桥	5
3.5.3	e-DCC	5
3.5.4	v-DCC	5
3.5.5	SCC	5
3.5.6	2-SAT	5
3.5.7	支配树	5
3.6	二分图	5
3.6.1	最大匹配-匈牙利	5
3.6.2	带权匹配-KM	5
3.7	网络流	5
3.7.1	最大流-Dinic	5
3.7.2	最小费用最大流-Dij+Dinic	5
3.7.3	上下界流	6
3.8	欧拉路	6
3.9	Prufer 序列	6
4	数据结构	6
4.1	树状数组	6
4.2	线段树	6
4.2.1	带优先级线段树	6
4.2.2	吉司机线段树	6
4.2.3	线段树维护扫描线	7
4.3	RMQ	7
4.3.1	一维	7
4.3.2	二维	7
4.4	树链剖分	7
4.4.1	点剖分	7
4.4.2	边剖分	8
4.5	平衡树	8
4.5.1	Treap	8
4.5.2	Splay	10
4.6	动态树	10
4.7	主席树	10
4.8	树套树	11
4.8.1	线段树套 Treap	11
4.8.2	树状数组套线段树	13
4.9	K-D Tree	14
4.10	分治	14
4.10.1	CDQ	14
4.10.2	点分治	14
4.10.3	dsu on tree	14
4.10.4	整体二分	14
4.11	分块	14
4.11.1	普通分块	14
4.11.2	莫队	14
4.12	线性基	14
4.13	珂朵莉树	14
4.14	跳舞链	14
5	动态规划	14
5.1	SOS	14
5.2	动态 DP	15
5.3	插头 DP	15
6	数学	15
6.1	矩阵类	15
6.2	质数筛	15
6.2.1	埃筛	15
6.2.2	线筛	15
6.3	质数判定	15
6.3.1	Miller Rabin	15
6.4	质因数分解	15
6.4.1	Pollard-Rho	15
6.5	逆元	15
6.5.1	EX-GCD 求逆元	15
6.5.2	线性筛逆元	15
6.5.3	阶乘逆元	15
6.6	欧拉函数	15
6.6.1	欧拉线筛	15
6.6.2	求单个数的欧拉函数	15
6.6.3	欧拉降幂	15
6.6.4	一般积性函数求法	15
6.7	EX-GCD	15
6.8	CRT	15
6.9	N 次剩余	15
6.10	数论分块	15
6.11	高斯消元	15
6.11.1	普通消元	15
6.11.2	异或方程组消元	15
6.12	莫比乌斯反演	15
6.12.1	莫比乌斯函数	15
6.12.2	杜教筛	15
6.12.3	洲阁筛	15
6.12.4	min25 筛	15
6.13	BSGS	15
6.14	FFT	15
6.15	FWT	15
6.16	NTT	15
6.17	数值计算	15
6.17.1	辛普森	15
6.17.2	自适应辛普森	15
6.18	康拓展开	15
6.19	卢卡斯定理	15

7 其他15

7.1 快读快写15

7.2 约瑟夫环15

7.3 悬线法15

7.4 蔡勒公式15

7.5 三角公式15

7.6 海伦公式15

7.7 匹克定理15

7.8 组合计数15

7.8.1 计数原理15

7.8.2 卡特兰数15

7.8.3 Polya15

7.8.4 二项式反演公式15

7.8.5 斯特林反演公式15

7.8.6 组合数恒等式15

1 字符串

1.1 KMP

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAXN=1000005;
5 char s1[MAXN],s2[MAXN];
6 int nxt[MAXN];
7
8 /*
9     nxt[i] s2[i-x..i-1]=s2[0..x-1]且x最大
10     即s2[0..i]的真前缀与真后缀的最大匹配
11     "ABAAB\0"=>[-1 0 0 1 1 2]
12 */
13
14 void get_fail(char *s,int l)
15 {
16     int i=0,j;
17     j=nxt[0]=-1;
18     while(i<l)
19     {
20         while(~j&&s[j]!=s[i]) j=nxt[j];
21         nxt[++i]=++j;
22     }
23 }
24
25 void kmp(char *s1,char *s2,int l1,int l2)
26 {
27     int i=0,j=0;
28     get_fail(s2,l2);
29     while(i<l1)
30     {
31         while(~j&&s1[i]!=s2[j]) j=nxt[j];
32         i++,j++;
33         if(j>=l2); //匹配上了
34     }
35 }
36
37 int main()
38 {
39     scanf("%s%s",s1,s2);
40     int l1=strlen(s1),l2=strlen(s2);
41     kmp(s1,s2,l1,l2);
42     for(int i=0;i<=l2;i++)
43         printf("%d ",nxt[i]);
44     return 0;
45 }

```

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=500005;
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 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5     str[0..len-1] 原串
6     sa[1..len] 排名第i的后缀的下标[1..len]
7     Rank[1..len] 从i开始的后缀的排名[1..len]
8     height[1..len] 排名第i的后缀与排名第i-1的后缀的lcp
9     i开始的后缀与j开始的后缀的lcp (Rank[i]<Rank[j])
10     min{height[Rank[i]+1..Rank[j]]}
11 */
12
13 const int MAXN=100005;
14 const int inf=0x3f3f3f3f;

```

```

15 int wa[MAXN],wb[MAXN],wv[MAXN],wz[MAXN],sa[MAXN],Rank
    [MAXN],height[MAXN];
16 char str[MAXN];
17
18 inline bool cmp(int *r,int a,int b,int l){return r[a
    ]==r[b]&&r[a+1]==r[b+1];}
19
20 void da(const char r[],int sa[],int n,int m)
21 {
22     int i,j,p,*x=wa,*y=wb,*t;
23     for(i=0;i<m;i++) wz[i]=0;
24     for(i=0;i<n;i++) wz[x[i]=r[i]]++;
25     for(i=1;i<m;i++) wz[i]+=wz[i-1];
26     for(i=n-1;i>=0;i--) sa[--wz[x[i]]]=i;
27     for(j=1,p=1;p<n;j*=2,m=p)
28     {
29         for(p=0,i=n-j;i<n;i++) y[p++]=i;
30         for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
31         for(i=0;i<n;i++) wv[i]=x[y[i]];
32         for(i=0;i<m;i++) wz[i]=0;
33         for(i=0;i<n;i++) wz[wv[i]]++;
34         for(i=1;i<m;i++) wz[i]+=wz[i-1];
35         for(i=n-1;i>=0;i--) sa[--wz[wv[i]]]=y[i];
36         for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
37             x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
38     }
39 }
40
41 void calheight(const char *r,int *sa,int n)
42 {
43     int i,j,k=0;
44     for(i=1;i<=n;i++) Rank[sa[i]]=i;
45     for(i=0;i<n;height[Rank[i+1]]=k)
46         for(k?k--:0,j=sa[Rank[i]-1];r[i+k]==r[j+k];k++);
47     for(int i=n;i>=1;--i) sa[i]++,Rank[i]=Rank[i-1];
48 }
49
50 int main()
51 {
52     scanf("%s",str);
53     int len=strlen(str);
54     da(str,sa,len+1,130); //字符的值域
55     calheight(str,sa,len);
56     for(int i=1;i<=len;i++)
57         printf("sa[%d] %d\n",i,sa[i]);
58     for(int i=1;i<=len;i++)
59         printf("Rank[%d] %d\n",i,Rank[i]);
60     for(int i=1;i<=len;i++)
61         printf("height[%d] %d\n",i,height[i]);
62     return 0;
63 }

```

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&&sign(x)>=0);}
36     db dot(P p){return x*p.x+y*p.y;}
37     db det(P p){return x*p.y-y*p.x;}
38     P rot(db an){return {x*cos(an)-y*sin(an),x*sin(an)+y*cos(an)};}
39 };
40
41 //For segment
42 #define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.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);

```

```

54     return (p1*a2+p2*a1)/(a1+a2);
55 }
56
57 bool intersect(db l1,db r1,db l2,db r2)
58 {
59     if(l1>r1) swap(l1,r1);if(l2>r2) swap(l2,r2);
60     return !(cmp(r1,l2)==-1||cmp(r2,l1)==-1);
61 }
62
63 bool isSS(P p1,P p2,P q1,P q2)
64 {
65     return intersect(p1.x,p2.x,q1.x,q2.x)&&intersect(
66         p1.y,p2.y,q1.y,q2.y)&&
67     crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<=0&&crossOp(q1
68         ,q2,p1)*crossOp(q1,q2,p2)<=0;
69 }
70
71 bool isSS_strict(P p1,P p2,P q1,P q2)
72 {
73     return crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<0
74     &&crossOp(q1,q2,p1)*crossOp(q1,q2,p2)<0;
75 }
76
77 bool isMiddle(db a,db m,db b)
78 {
79     return sign(a-m)==0||sign(b-m)==0||(a<m!=b<m);
80 }
81
82 bool isMiddle(P a,P m,P b)
83 {
84     return isMiddle(a.x,m.x,b.x)&&isMiddle(a.y,m.y,b.y
85         );
86 }
87
88 bool onSeg(P p1,P p2,P q)
89 {
90     return crossOp(p1,p2,q)==0&&isMiddle(p1,q,p2);
91 }
92
93 bool onSeg_strict(P p1,P p2,P q)
94 {
95     return crossOp(p1,p2,q)==0&&sign((q-p1).dot(p1-p2)
96         )*sign((q-p2).dot(p1-p2))<0;
97 }
98
99 P proj(P p1,P p2,P q)
100 {
101     P dir=p2-p1;
102     return p1+dir*(dir.dot(q-p1)/dir.abs2());
103 }
104
105 P reflect(P p1,P p2,P q)
106 {
107     return proj(p1,p2,q)*2-q;
108 }
109
110 db nearest(P p1,P p2,P q)
111 {
112     P h=proj(p1,p2,q);
113     if(isMiddle(p1,h,p2))
114         return q.distTo(h);
115     return min(p1.distTo(q),p2.distTo(q));
116 }
117
118 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 db rad(P p1,P p2)
121 {
122     return atan2l(p1.det(p2),p1.dot(p2));
123 }
124
125 db area(vector<P> ps)
126 {
127     db ret=0;
128     for(int i=0;i<ps.size();i++)
129         ret+=ps[i].det(ps[(i+1)%ps.size()]);
130     return ret/2;
131 }
132
133 int contain(vector<P> ps,P p) //2:inside,1:on_seg,0:
134     outside
135 {
136     int n=ps.size(),ret=0;
137     for(int i=0;i<n;i++)
138     {
139         P u=ps[i],v=ps[(i+1)%n];
140         if(onSeg(u,v,p)) return 1;
141         if(cmp(u.y,v.y)<=0) swap(u,v);
142         if(cmp(p.y,u.y)>0||cmp(p.y,v.y)<=0) continue;
143         ret^=crossOp(p,u,v)>0;
144     }
145     return ret*2;
146 }
147
148 vector<P> convexHull(vector<P> ps)
149 {
150     int n=ps.size();if(n<=1) return ps;
151     sort(ps.begin(),ps.end());
152     vector<P> qs(n*2);int k=0;
153     for(int i=0;i<n;qs[k++]=ps[i++])
154         while(k>1&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
155             --k;
156     for(int i=n-2,t=k;i>=0;qs[k++]=ps[i--])
157         while(k>t&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
158             --k;
159     qs.resize(k-1);
160     return qs;
161 }
162
163 db convexDiameter(vector<P> ps)
164 {
165     int n=ps.size();if(n<=1) return 0;
166     int is=0,js=0;
167     for(int k=1;k<n;k++) is=ps[k]<ps[is]?k:is,js=ps[js]
168         <ps[k]?js:k;
169     int i=is,j=js;
170     db ret=ps[i].distTo(ps[j]);
171     do{
172         if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j])
173             >=0) (++j)%=n;
174         else (++i)%=n;
175         ret=max(ret,ps[i].distTo(ps[j]));
176     }while(i!=is||j!=js);
177     return ret;
178 }

```

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 inline void add_edge(int x,int y,int f)

```

```

23 {
24     E[x].emplace_back(y,f,E[y].size());
25     E[y].emplace_back(x,0,E[x].size()-1);
26 }
27
28 int bfs()
29 {
30     for(int i=1;i<=n;i++) dis[i]=0x3f3f3f3f;
31     dis[s]=0;
32     queue<int> q;
33     q.push(s);
34     while(!q.empty())
35     {
36         int now=q.front();q.pop();
37         for(int i=0;i<E[now].size();i++)
38         {
39             edge &e=E[now][i];
40             if(dis[e.to]>dis[now]+1&&e.cap)
41             {
42                 dis[e.to]=dis[now]+1;
43                 if(e.to==t) return 1;
44                 q.push(e.to);
45             }
46         }
47     }
48     return 0;
49 }
50
51 ll dfs(int now,ll flow)
52 {
53     if(now==t) return flow;
54     ll rest=flow,k;
55     for(int i=cur[now];i<E[now].size();i++)
56     {
57         edge &e=E[now][i];
58         if(e.cap&&dis[e.to]==dis[now]+1)
59         {
60             cur[now]=i;
61             k=dfs(e.to,min(rest,(long long)e.cap));
62             e.cap-=k;
63             E[e.to][e.rev].cap+=k;
64             rest-=k;
65         }
66     }
67     return flow-rest;
68 }
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

```



```

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 二维

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[
17     MAXN],num[MAXN],fa[MAXN];
18 vector<int> e[MAXN];
19
20 void dfs1(int now,int f)
21 {
22     dep[now]=dep[f]+1;
23     fa[now]=f;
24     num[now]=1;
25     son[now]=0;
26     for(auto to:e[now])
27     {
28         if(to==f) continue;
29         dfs1(to,now);
30         num[now]+=num[to];
31         if(num[to]>num[son[now]]) son[now]=to;
32     }
33 }
34
35 void dfs2(int now,int f)
36 {
37     id[now]=++cnt;
38     top[now]=f;
39     if(son[now]) dfs2(son[now],f);
40     for(auto to:e[now])
41     {
42         if(to!=fa[now]&&to!=son[now])
43             dfs2(to,to);
44     }
45 }

```

```

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;
155             modify(1,1,n,id[x],id[x]+num[x]-1,z);
156         }
157     }
158     return 0;
159 }

```

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;

```

```

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     }
86     else
87     {
88         _insert(bst[rt].rc,val);
89         if(bst[rt].dat<bst[bst[rt].rc].dat) zag(rt)
90             ;
91     }
92     pushup(rt);
93 }
94
95 void _delete(int &rt,int val)
96 {
97     if(rt==0) return;
98     if(bst[rt].val==val)
99     {
100         if(bst[rt].cnt>1)
101         {
102             bst[rt].cnt--;
103             pushup(rt);
104             return;
105         }
106         if(bst[rt].rc||bst[rt].lc)
107         {
108             if(bst[rt].rc==0||bst[bst[rt].rc].dat<
109                 bst[bst[rt].lc].dat)
110                 zig(rt),_delete(bst[rt].rc,val);
111             else
112                 zag(rt),_delete(bst[rt].lc,val);
113             pushup(rt);
114         }
115         else rt=0;
116         return;
117     }
118     if(val<bst[rt].val) _delete(bst[rt].lc,val);
119     else _delete(bst[rt].rc,val);
120     pushup(rt);
121 }
122
123 int getPrev(int val)
124 {
125     int ret=1,rt=root;
126     while(rt)
127     {
128         if(bst[rt].val==val)
129         {
130             if(bst[rt].lc)
131             {
132                 rt=bst[rt].lc;
133                 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[rt].lc.
        siz+1;
169     if(val<bst[rt].val) return getRank(bst[rt].lc,
        val);
170     else return 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[rt].lc.siz>=k) return getVal(bst[
        rt].lc,k);
177     if(bst[rt].lc.siz+bst[rt].cnt>=k) return
        bst[rt].val;
178     return getVal(bst[rt].rc,k-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--)
```

```

189     {
190         scanf("%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

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5 离散化+区间k小
6 */
7
8 const int MAXN=200005;
9 int n,m,a[MAXN],tmp[MAXN],org[MAXN],root[MAXN],tot=0;
10 struct tree
11 {
12     int cnt,lc,rc;
13 }seg[30*MAXN];
14
15 int build(int l,int r)
16 {
17     int p=tot++;
18     if(l==r)
19     {
20         seg[p].cnt=0;
21         return p;
22     }
23     int m=l+r>>1;
24     seg[p].lc=build(l,m);
25     seg[p].rc=build(m+1,r);
26     seg[p].cnt=seg[seg[p].lc].cnt+seg[seg[p].rc].cnt;
27     return p;
28 }
29
30 int modify(int rt,int l,int r,int x)
31 {
32     int p=tot++;
33     seg[p]=seg[rt];
34     if(l==r)
35     {
36         seg[p].cnt++;
37         return p;
38     }
39     int m=l+r>>1;
40     if(x<=m) seg[p].lc=modify(seg[rt].lc,l,m,x);
41     else seg[p].rc=modify(seg[rt].rc,m+1,r,x);
42     seg[p].cnt=seg[seg[p].lc].cnt+seg[seg[p].rc].cnt;
43     return p;
44 }
45
```

```

46 int query(int p,int q,int l,int r,int k)
47 {
48     if(l==r) return l;
49     int m=l+r>>1;
50     int lc=seg[seg[q].lc].cnt-seg[seg[p].lc].cnt;
51     if(lc>=k) return query(seg[p].lc,seg[q].lc,l,m,k);
52     else return query(seg[p].rc,seg[q].rc,m+1,r,k-lc);
53 }
54
55 int main()
56 {
57     scanf("%d",&n,&m);
58     for(int i=1;i<=n;i++)
59         scanf("%d",a+i),tmp[i]=a[i];
60     sort(tmp+1,tmp+n+1);
61     root[0]=build(1,n);
62     for(int i=1;i<=n;i++)
63     {
64         int k=lower_bound(tmp+1,tmp+n+1,a[i])-tmp;
65         org[k]=a[i];
66         a[i]=k;
67         root[i]=modify(root[i-1],1,n,a[i]);
68     }
69     while(m--)
70     {
71         int x,y,k;
72         scanf("%d%d%d",&x,&y,&k);
73         printf("%d\n",org[query(root[x-1],root[y],1,n,k)]);
74     }
75     return 0;
76 }

```

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 const int inf=2147483647;
11 const int MAXN=50005;
12 int root[MAXN<<2],n,m,a[MAXN];
13 struct Treap
14 {
15     int tot;
16     struct node
17     {
18         int lc,rc,dad,val,cnt,siz;
19     }bst[MAXN*4*20];
20
21     int newnode(int v)
22     {
23         bst[++tot].val=v;
24         bst[tot].dad=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].siz;
54     if(bst[rt].rc) bst[rt].siz+=bst[bst[rt].rc].siz;
55 }
56
57 int build()
58 {
59     int rt=newnode(-inf);
60     bst[rt].rc=newnode(inf);
61     pushup(rt);
62     return rt;
63 }
64
65 void _delete(int &rt,int x)
66 {
67     if(bst[rt].val==x)
68     {
69         if(bst[rt].cnt>1)
70         {
71             bst[rt].cnt--;
72             pushup(rt);
73             return;
74         }
75         if(bst[rt].lc||bst[rt].rc)
76         {
77             if(bst[rt].rc==0||bst[bst[rt].rc].dad<bst[bst[rt].lc].dad)
78                 zig(rt),_delete(bst[rt].rc,x);
79             else
80                 zag(rt),_delete(bst[rt].lc,x);
81             pushup(rt);
82         }
83         else rt=0;
84         return;
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         }
151         if(bst[rt].val<x&&bst[rt].val>ret) ret=bst[
152             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[
166             rt].val;
167         if(x<bst[rt].val) rt=bst[rt].lc;
168         else rt=bst[rt].rc;
169     }
170     return ret;
171 }
172 }treap;
173
174 void build(int rt,int l,int r)
175 {
176     root[rt]=treap.build();
177     if(l==r) return;
178     int m=l+r>>1;
179     build(rt<<1,l,m);
180     build(rt<<1|m,m+1,r);
181 }
182
183 void modify(int rt,int l,int r,int x,int v,int y)
184 {
185     if(y==-1) treap._delete(root[rt],v);
186     else treap._insert(root[rt],v);
187     if(l==r) return;
188     int m=l+r>>1;
189     if(x<=m) modify(rt<<1,l,m,x,v,y);
190     else modify(rt<<1|m,m+1,r,x,v,y);
191 }
192
193 int query(int rt,int l,int r,int op,int L,int R,int x
194 )
195 {
196     if(L<=l&&R<=R)
197     {
198         if(op==1) return treap.get_rank(root[rt],x)-2;
199         if(op==2) return treap.get_num(root[rt],x)-1;
200         if(op==4) return treap.get_prev(root[rt],x);
201         if(op==5) return treap.get_nxt(root[rt],x);
202     }
203     int m=l+r>>1,ret;
204     if(op==1||op==2)
205     {
206         ret=0;
207         if(L<=m) ret+=query(rt<<1,l,m,op,L,R,x);
208         if(m<R) ret+=query(rt<<1|m,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 );
213     if(m<R) ret=max(ret,query(rt<<1|1,m+1,r,op,L,R
214 ,x));
215 }
216 if(op==5)
217 {
218     ret=inf;
219     if(L<=m) ret=min(ret,query(rt<<1,l,m,op,L,R,x)
220 );
221     if(m<R) ret=min(ret,query(rt<<1|1,m+1,r,op,L,R
222 ,x));
223 }
224 return ret;
225 }
226
227 int main()
228 {
229     srand(time(0));
230     scanf("%d%d",&n,&m);
231     build(1,1,n);
232     for(int i=1;i<=n;i++)
233     {
234         scanf("%d",a+i);
235         modify(1,1,n,i,a[i],1);
236     }
237     while(m--)
238     {
239         int op,l,r,k,pos;
240         scanf("%d",&op);
241         if(op==1)
242         {
243             scanf("%d%d%d",&l,&r,&k);
244             printf("%d\n",query(1,1,n,op,l,r,k)+1);
245         }
246         else if(op==2)
247         {
248             scanf("%d%d%d",&l,&r,&k);
249             int L=-inf,R=inf,mid;
250             while(L<R)
251             {
252                 mid=(L+R+1)>>1;
253                 if(query(1,1,n,1,l,r,mid)+1>k) R=mid-1;
254                 else L=mid;
255             }
256             printf("%d\n",L);
257         }
258         else if(op==3)
259         {
260             scanf("%d%d",&pos,&k);
261             modify(1,1,n,pos,a[pos],-1);
262             a[pos]=k;
263             modify(1,1,n,pos,k,1);
264         }
265         else
266         {
267             scanf("%d%d%d",&l,&r,&k);
268             printf("%d\n",query(1,1,n,op,l,r,k));
269         }
270     }
271     return 0;

```

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 void add(int pos,int x,int y)
50 {
51     for(;pos<=n;pos+=pos&-pos) seg.modify(seg.root[pos]
52     ,1,VAL,x,y);
53 }
54 int query(int l,int r,int k)
55 {
56     c1=c2=0;
57     for(int i=l-1;i--&i) X[c1++]=seg.root[i];
58     for(int i=r;i--&i) Y[c2++]=seg.root[i];

```



```

59     return seg.query(1,VAL,k);
60 }

```

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
    [100005],hson;
11 ll ans[100005],sum;
12 vector<int> e[100005];
13
14 void dfs1(int now,int fa)
15 {
16     son[now]=0,siz[now]=1;
17     for(auto to:e[now])
18     {
19         if(to==fa) continue;
20         dfs1(to,now);
21         siz[now]+=siz[to];
22         if(siz[to]>siz[son[now]]) son[now]=to;
23     }
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

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const double eps=1e-12;
4
5  /*
6   调用 asr(l,r,simpson(l,r))
7  */
8
9  inline double f(double x)
10 {
11     return x; //被积函数
12 }
13
14 double simpson(double l,double r)
15 {
16     double mid=(l+r)/2;
17     return (f(l)+4*f(mid)+f(r))*(r-l)/6;
18 }
19
20 double asr(double l,double r,double ans)
21 {
22     double mid=(l+r)/2;
23     double l1=simpson(l,mid),r1=simpson(mid,r);
24     if(fabs(l1+r1-ans)<eps) return l1+r1;
25     return asr(l,mid,l1)+asr(mid,r,r1);
26 }
27
28 int main()
29 {
30
31     return 0;
32 }

```

6.18 康拓展开

6.19 卢卡斯定理

7 其他

7.1 快读快写

7.2 约瑟夫环

7.3 悬线法

7.4 蔡勒公式

7.5 三角公式

7.6 海伦公式

7.7 匹克定理

7.8 组合计数

7.8.1 计数原理

7.8.2 卡特兰数

7.8.3 Polya

7.8.4 二项式反演公式

7.8.5 斯特林反演公式

7.8.6 组合数恒等式