目录

[一、 数据结构](#_Toc29993_WPSOffice_Level1) [3](#_Toc29993_WPSOffice_Level1)

[1. ST表（RMQ问题，无修改）](#_Toc27910_WPSOffice_Level2) [3](#_Toc27910_WPSOffice_Level2)

[2. 树状数组&&线段树（经典带修改前缀和、区间操作等）](#_Toc2857_WPSOffice_Level2) [3](#_Toc2857_WPSOffice_Level2)

[3. 主席树（权值线段树的可持久化）](#_Toc5563_WPSOffice_Level2) [3](#_Toc5563_WPSOffice_Level2)

[3.1静态主席树（区间第K大为例）](#_Toc27910_WPSOffice_Level3) [3](#_Toc27910_WPSOffice_Level3)

[3.2带单点修改的主席树（信息可加性，差分思想）](#_Toc2857_WPSOffice_Level3) [4](#_Toc2857_WPSOffice_Level3)

[4. 归并树（维护区间排序后的序列）](#_Toc30146_WPSOffice_Level2) [5](#_Toc30146_WPSOffice_Level2)

[5. 划分树（查询区间第K大元素，空间略优）](#_Toc2665_WPSOffice_Level2) [6](#_Toc2665_WPSOffice_Level2)

[6. Treap](#_Toc21104_WPSOffice_Level2) [7](#_Toc21104_WPSOffice_Level2)

[7. Splay（支持翻转、分裂、插入序列等操作）](#_Toc15314_WPSOffice_Level2) [8](#_Toc15314_WPSOffice_Level2)

[8. LCT（动态维护树形态、连通性）](#_Toc252_WPSOffice_Level2) [10](#_Toc252_WPSOffice_Level2)

[9. 笛卡尔树](#_Toc7297_WPSOffice_Level2) [12](#_Toc7297_WPSOffice_Level2)

[10. 左偏树（可并堆，支持删除任意结点）](#_Toc31768_WPSOffice_Level2) [13](#_Toc31768_WPSOffice_Level2)

[11. 并查集](#_Toc11598_WPSOffice_Level2) [13](#_Toc11598_WPSOffice_Level2)

[12. 对顶堆](#_Toc13929_WPSOffice_Level2) [14](#_Toc13929_WPSOffice_Level2)

[二、 树论](#_Toc27910_WPSOffice_Level1) [14](#_Toc27910_WPSOffice_Level1)

[1. 树链剖分（注意点权和边权区别）](#_Toc24271_WPSOffice_Level2) [14](#_Toc24271_WPSOffice_Level2)

[2. dsu on tree（O(nlogn)处理子树相关静态查询问题）](#_Toc3241_WPSOffice_Level2) [15](#_Toc3241_WPSOffice_Level2)

[3. LCA](#_Toc30567_WPSOffice_Level2) [16](#_Toc30567_WPSOffice_Level2)

[3.1树上倍增](#_Toc5563_WPSOffice_Level3) [16](#_Toc5563_WPSOffice_Level3)

[3.2欧拉序+rmq](#_Toc30146_WPSOffice_Level3) [17](#_Toc30146_WPSOffice_Level3)

[3.3离线Tarjan算法](#_Toc2665_WPSOffice_Level3) [18](#_Toc2665_WPSOffice_Level3)

[三、 字符串](#_Toc2857_WPSOffice_Level1) [18](#_Toc2857_WPSOffice_Level1)

[1. Trie树](#_Toc24054_WPSOffice_Level2) [19](#_Toc24054_WPSOffice_Level2)

[2. 后缀数组](#_Toc27359_WPSOffice_Level2) [19](#_Toc27359_WPSOffice_Level2)

[3. 回文树](#_Toc9105_WPSOffice_Level2) [20](#_Toc9105_WPSOffice_Level2)

[4. 最小表示法](#_Toc14247_WPSOffice_Level2) [21](#_Toc14247_WPSOffice_Level2)

[四、图论](#_Toc5563_WPSOffice_Level1) [21](#_Toc5563_WPSOffice_Level1)

[1. 费用流](#_Toc27542_WPSOffice_Level2) [22](#_Toc27542_WPSOffice_Level2)

[1.1 Dijkstra费用流](#_Toc21104_WPSOffice_Level3) [22](#_Toc21104_WPSOffice_Level3)

[1.2 Spfa费用流](#_Toc15314_WPSOffice_Level3) [23](#_Toc15314_WPSOffice_Level3)

[1.3 ZKW费用流](#_Toc252_WPSOffice_Level3) [25](#_Toc252_WPSOffice_Level3)

[2. 匈牙利算法求二分图最小点覆盖方案](#_Toc16171_WPSOffice_Level2) [26](#_Toc16171_WPSOffice_Level2)

[3. 割点、桥](#_Toc30324_WPSOffice_Level2) [28](#_Toc30324_WPSOffice_Level2)

[4. 边双连通分量](#_Toc31213_WPSOffice_Level2) [29](#_Toc31213_WPSOffice_Level2)

[5. 点双连通分量](#_Toc30003_WPSOffice_Level2) [29](#_Toc30003_WPSOffice_Level2)

[6. 强连通分量](#_Toc11580_WPSOffice_Level2) [30](#_Toc11580_WPSOffice_Level2)

[7. 2-SAT](#_Toc5827_WPSOffice_Level2) [31](#_Toc5827_WPSOffice_Level2)

[7.1 dfs染色求最小字典序解](#_Toc13929_WPSOffice_Level3) [31](#_Toc13929_WPSOffice_Level3)

[7.2 Tarjan强连通分量求法](#_Toc24271_WPSOffice_Level3) [32](#_Toc24271_WPSOffice_Level3)

[五、数论](#_Toc30146_WPSOffice_Level1) [33](#_Toc30146_WPSOffice_Level1)

[1. 素数筛法](#_Toc14264_WPSOffice_Level2) [33](#_Toc14264_WPSOffice_Level2)

[1.1 埃氏筛法（最大质因数）](#_Toc3241_WPSOffice_Level3) [33](#_Toc3241_WPSOffice_Level3)

[1.2 欧拉筛法（最小质因数）](#_Toc30567_WPSOffice_Level3) [34](#_Toc30567_WPSOffice_Level3)

[1.3 Miller Rabin素数测试](#_Toc24054_WPSOffice_Level3) [34](#_Toc24054_WPSOffice_Level3)

[1.4大区间素数筛选（区间长度较小）](#_Toc27359_WPSOffice_Level3) [34](#_Toc27359_WPSOffice_Level3)

[2. 欧几里德算法](#_Toc17642_WPSOffice_Level2) [35](#_Toc17642_WPSOffice_Level2)

[2.1欧几里德算法（最大公约数）](#_Toc9105_WPSOffice_Level3) [35](#_Toc9105_WPSOffice_Level3)

[2.2扩展欧几里德算法（解不定方程ax+by=gcd(a,b)）](#_Toc14247_WPSOffice_Level3) [35](#_Toc14247_WPSOffice_Level3)

[2.3类欧几里得算法](#_Toc27542_WPSOffice_Level3) [35](#_Toc27542_WPSOffice_Level3)

[六、其他](#_Toc2665_WPSOffice_Level1) [39](#_Toc2665_WPSOffice_Level1)

[1. 线性基](#_Toc29458_WPSOffice_Level2) [40](#_Toc29458_WPSOffice_Level2)

[2. 手动加栈](#_Toc13712_WPSOffice_Level2) [41](#_Toc13712_WPSOffice_Level2)

[3. 高效位运算](#_Toc23714_WPSOffice_Level2) [41](#_Toc23714_WPSOffice_Level2)

[4. 产生随机数](#_Toc23415_WPSOffice_Level2) [41](#_Toc23415_WPSOffice_Level2)

[5. Cdq分治](#_Toc966_WPSOffice_Level2) [41](#_Toc966_WPSOffice_Level2)

[6. 莫队算法](#_Toc22945_WPSOffice_Level2) [43](#_Toc22945_WPSOffice_Level2)

[6.1普通莫队](#_Toc31213_WPSOffice_Level3) [43](#_Toc31213_WPSOffice_Level3)

[6.2带修莫队（单点修改）](#_Toc30003_WPSOffice_Level3) [43](#_Toc30003_WPSOffice_Level3)

[6.3树上莫队（序列+LCA）](#_Toc11580_WPSOffice_Level3) [45](#_Toc11580_WPSOffice_Level3)

[6.4树上带修莫队](#_Toc5827_WPSOffice_Level3) [48](#_Toc5827_WPSOffice_Level3)

[七、 计算几何](#_Toc21104_WPSOffice_Level1) [52](#_Toc21104_WPSOffice_Level1)

[1. 动态凸包](#_Toc29739_WPSOffice_Level2) [52](#_Toc29739_WPSOffice_Level2)

1. **数据结构**
2. **ST表（RMQ问题，无修改）**

int dp\_max[max\_logn][maxn],mm[maxn];

void init()

{

mm[0]=0;

for(int i=1;i<maxn;++i){

mm[i]=mm[i-1];

if(i==1<<(mm[i]+1)) ++mm[i];

}

for(int i=1;i<=n;++i) dp[0][i]=w[i];

for(int j=1;j<=mm[n];++j){

for(int i=1;i+(1<<j)-1<=n;++i){

dp\_max[j][i]=max(dp\_max[j-1][i],dp\_max[j-1][i+(1<<(j-1))];

}

}

}

int Rmq(int l,int r)

{

int k=mm[r-l+1];

return max(dp\_max[k][l],dp\_max[k][r-(1<<k)+1]);

}

1. **树状数组&&线段树（经典带修改前缀和、区间操作等）**
2. **主席树（权值线段树的可持久化）**

**3.1静态主席树（区间第K大为例）**

int rt[maxn],ls[maxn\*30],rs[maxn\*30],cnt[maxn\*30],tot;

void build(int l,int r,int &rt)

{

rt=++tot;

ls[rt]=rs[rt]=cnt[rt]=0;

if(l==r) return;

int mid=l+r>>1;

build(l,mid,ls[rt]);

build(mid+1,r,rs[rt]);

}

void update(int last,int pos,int l,int r,int &rt)

{

rt=++tot;

ls[rs]=ls[last],rs[rt]=rs[last],cnt[rt]=cnt[last]+1;

if(l==r) return;

int mid=l+r>>1;

if(pos<=mid) update(ls[last],pos,l,mid,ls[rt]);

else update(rs[last],pos,mid+1,r,rs[rt]);

}

int Query(int rtx,int rty,int l,int r,int k)

{

if(l==r) return l;

int mid=l+r>>1,sum=cnt[ls[rty]]-cnt[ls[rtx]];

if(sum>=k) return Query(ls[rtx],ls[rty],l,mid,k);

return Query(rs[rtx],rs[rty],mid+1,r,k-sum);

}

**3.2带单点修改的主席树（信息可加性，差分思想）**

int rt[maxn],ls[maxn\*30],rs[maxn\*30],cnt[maxn\*30],tot;

void build(int l,int r,int &rt)

{

rt=++tot;

ls[rt]=rs[rt]=cnt[rt]=0;

if(l==r) return;

int mid=l+r>>1;

build(l,mid,ls[rt]);

build(mid+1,r,rs[rt]);

}

void update(int last,int pos,int l,int r,int &rt)

{

rt=++tot;

ls[rs]=ls[last],rs[rt]=rs[last],cnt[rt]=cnt[last]+1;

if(l==r) return;

int mid=l+r>>1;

if(pos<=mid) update(ls[last],pos,l,mid,ls[rt]);

else update(rs[last],pos,mid+1,r,rs[rt]);

}

int Query(int rtx,int rty,int l,int r,int k)

{

if(l==r) return l;

int mid=l+r>>1,sum=cnt[ls[rty]]-cnt[ls[rtx]];

if(sum>=k) return Query(ls[rtx],ls[rty],l,mid,k);

return Query(rs[rtx],rs[rty],mid+1,r,k-sum);

}

void Add(int x,int pos,int val)

{

while(x<=n){

update(S[x],pos,val,1,n,S[x]);

x+=x&-x;

}

}

int Sum(int x)

{

int ans=0;

while(x){

ans+=cnt[ls[use[x]]];

x-=x&-x;

}

return ans;

}

int Query(int L,int R,int k)

{

int rty=rt[R],rtx=rt[L-1],l=1,r=n,mid;

for(int i=L-1;i;i-=i&-i) use[i]=S[i];

for(int i=R;i;i-=i&-i) use[i]=R[i];

while(l<r){

mid=l+r>>1;

int sum=cnt[ls[rty]]-cnt[ls[rtx]]+Sum(R)-Sum(L-1);

if(sum>=k){

r=mid;

rtx=ls[rtx],rty=ls[rty];

for(int i=L-1;i;i-=i&-i) use[i]=ls[use[i]];

for(int i=R;i;i-=i&-i) use[i]=ls[use[i]];

}

else{

k-=sum;

l=mid+1;

rtx=rs[rtx],rty=rs[rty];

for(int i=L-1;i;i-=i&-i) use[i]=rs[use[i]];

for(int i=R;i;i-=i&-i) use[i]=rs[use[i]];

}

}

return l;

}

1. **归并树（维护区间排序后的序列）**

vector<int>T[maxn<<2];

void build(int l,int r,int rt)

{

T[rt].resize(r-l+1);

if(l==r){

T[rt][0]=a[l];

return;

}

int mid=l+r>>1;

build(l,mid,rt<<1);

build(mid+1,r,rt<<1|1);

merge(T[rt<<1].begin(),T[rt<<1].end(),T[rt<<1|1].begin(),T[rt<<1|1].end(),T[rt].begin());

}

1. **划分树（查询区间第K大元素，空间略优）**

int T[max\_logn][maxn],Num[max\_logn][maxn],a[maxn];

//sort(a+1,a+1+n);

void build(int l,int r,int dep)

{

if(l==r) return;

int mid=l+r>>1,same=mid-l+1,p=l,q=mid+1;

for(int i=l;i<=r;++i) if(T[dep][i]<a[mid]) --same;

for(int i=l;i<=r;++i){

if(T[dep][i]<a[mid]) T[dep+1][p++]=T[dep][i];

else if(T[dep][i]==a[mid]&&same) T[dep+1][p++]=T[dep][i],--same;

else T[dep+1][q++]=T[dep][i];

Num[dep][i]=Num[dep][l-1]+p-l;

}

build(l,mid,dep+1);

build(mid+1,r,dep+1);

}

//查询区间是[l,r],LR是结点对应区间

int Query(int l,int r,int L,int R,int dep,int k)

{

if(l==r) return T[dep][l];

int mid=L+R>>1,cnt=Num[dep][r]-Num[dep][l-1];

if(cnt>=k){

int nl=L+Num[dep][l-1]-Num[dep][L-1];

int nr=nl+cnt-1;

return Query(nl,nr,L,mid,dep+1,k);

}

else{

int nr=r+Num[dep][R]-Num[dep][r];

int nl=nr-(r-l-cnt);

return Query(nl,nr,mid+1,R,dep+1,k-cnt);

}

}

1. **Treap**

int ch[maxn][2],key[maxn],pri[maxn],sz[maxn],cnt[maxn],rt,tot;

void pushup(int r)

{

sz[r]=sz[ch[r][0]]+sz[ch[r][1]]+cnt[r];

}

void newNode(int &r,int val)

{

r=++tot;

ch[r][0]=ch[r][1]=0;

key[r]=val;

pri[r]=rand();

sz[r]=cnt[r]=1;

}

//将结点r的儿子往d方向旋上去,0左1右

void rotate(int &r,int d)

{

int c=ch[r][d^1];

ch[r][d^1]=ch[c][d];

ch[c][d]=r;

pushup(r);

r=c;

}

void insert(int &r,int x)

{

if(!r){newNode(r,x);return;}

if(key[r]==x){++sz[r],++cnt[r];return;}

int d=x>key[r];

insert(ch[r][d],x);

if(pri[ch[r][d]]>pri[r]) rotate(r,d^1);

pushup(r);

}

void del(int &r,int x)

{

if(!r) return;

if(key[r]==x){

if(cnt[r]>1){--cnt[r],--sz[r];return;}

if(!ch[r][0]) r=ch[r][1];

else if(!ch[r][1]) r=ch[r][0];

else{

int d=pri[ch[r][0]]>pri[ch[r][1]];

rotate(r,d);

del(ch[r][d],x);

}

}

else del(ch[r][x>key[r]],x);

pushup(r);

}

int kth(int k)

{

if(k<0||k>sz[rt]) return inf;

int r=rt;

while(1){

if(sz[ch[r][0]]>=k) r=ch[r][0];

else if(sz[ch[r][0]]+cnt[r]<k){

k-=sz[ch[r][0]]+cnt[r];

r=ch[r][1];

}

else break;

}

return key[r];

}

1. **Splay（支持翻转、分裂、插入序列等操作）**

#define kw ch[ch[rt][1]][0]

int ch[maxn][2],fa[maxn],sz[maxn],rev[maxn],rt;

int key[maxn],mx[maxn],add[maxn];

void newNode(int &r,int p,int id)

{

r=id;

ch[r][0]=ch[r][1]=add[r]=rev[r]=0;

fa[r]=p;

sz[r]=1;

mx[r]=key[r]=w[r];

}

void update\_rev(int r)

{

if(!r) return;

swap(ch[r][0],ch[r][1]);

rev[r]^=1;

}

void update\_add(int r,int val)

{

if(!r) return;

key[r]+=val;

mx[r]+=val;

add[r]+=val;

}

void pushdown(int r)

{

if(add[r]){

update\_add(ch[r][0],add[r]);

update\_add(ch[r][1],add[r]);

add[r]=0;

}

if(rev[r]){

update\_rev(ch[r][0]);

update\_rev(ch[r][1]);

rev[r]=0;

}

}

void pushup(int r)

{

sz[r]=sz[ch[r][0]]+sz[ch[r][1]]+1;

mx[r]=max(key[r],max(mx[ch[r][0]],mx[ch[r][1]]));

}

int getson(int r)

{

return ch[fa[r]][1]==r;

}

void rotate(int r)

{

int p=fa[r],d=getson(r)^1;

pushdown(p);

pushdown(r);

ch[p][d^1]=ch[r][d];

fa[ch[r][d]]=p;

if(fa[p]) ch[fa[p]][getson(p)]=r;

fa[r]=fa[p];

ch[r][d]=p;

fa[p]=r;

pushup(p);

}

void splay(int r,int goal)

{

for(int p;fa[r]!=goal;rotate(r)){

if(fa[p=fa[r]]!=goal)

getson(p)==getson(r)?rotate(p):rotate(r);

}

pushup(r);

if(!goal) rt=r;

}

void build(int l,int r,int&rt,int p)

{

if(l>r) return;

int mid=l+r>>1;

newNode(rt,p,mid);

build(l,mid-1,ch[rt][0],rt);

build(mid+1,r,ch[rt][1],rt);

pushup(rt);

}

void init()

{

rt=0;

ch[0][0]=ch[0][1]=f[0]=sz[0]=key[0]=add[0]=rev[0]=0;

mx[0]=-INF;

newNode(rt,0,n+1);

newNode(ch[rt][1],rt,n+2);

build(1,n,kw,ch[rt][1]);

pushup(ch[rt][1]);

pushup(rt);

}

1. **LCT（动态维护树形态、连通性）**

int ch[maxn][2],fa[maxn],rev[maxn],stk[maxn];

inline bool isrt(int r){return ch[fa[r]][0]!=r&&ch[fa[r]][1]!=r;}

inline int getson(int r){return ch[fa[r]][1]==r;}

void update\_rev(int r)

{

if(!r) return;

swap(ch[r][0],ch[r][1]);

rev[r]^=1;

return;

}

void pushdown(int r)

{

if(rev[r]){

update\_rev(ch[r][0]);

update\_rev(ch[r][1]);

rev[r]=0;

}

return;

}

void pushup(int r){}

void rotate(int x)

{

int y=fa[x],z=fa[y],c=getson(x);

if(!isrt(y)) ch[z][getson(y)]=x;

fa[x]=z;

ch[y][c]=ch[x][c^1],fa[ch[y][c]]=y;

ch[x][c^1]=y,fa[y]=x;

pushup(y);

return;

}

void splay(int x)

{

int top=0;stk[++top]=x;

for(int i=x;!isrt(i);i=fa[i]) stk[++top]=fa[i];

while(top) pushdown(stk[top--]);

for(int y=fa[x];!isrt(x);rotate(x),y=fa[x]){

if(!isrt(y)) getson(x)^getson(y)?rotate(x):rotate(y);

}

pushup(x);

return;

}

void Access(int x)

{

for(int y=0;x;x=fa[y=x]){

splay(x);

ch[x][1]=y;

//ch[x][1]变为虚子树，y变为实子树

pushup(x);

}

return;

}

void Makeroot(int x)

{

Access(x);

splay(x);

update\_rev(x);

return;

}

int Findroot(int x)

{

Access(x);

splay(x);

while(ch[x][0]) x=ch[x][0];

return x;

}

void split(int x,int y)

{

Makeroot(x);

Access(y);

splay(y);

return;

}

void Link(int x,int y)

{

Makeroot(x);

//维护子树信息时把y也makeroot一下

fa[x]=y;

return;

}

void Cut(int x,int y)

{

split(x,y);

ch[y][0]=fa[x]=0;

pushup(y);

return;

}

//有根树CUT，y是x父亲

inline void Cut(int x,int y)

{

Access(x);

splay(y);

ch[y][1]=fa[x]=0;

pushup(y);

return;

}

void init(){}

1. **笛卡尔树**

//按下标顺序构造O(n)，大根堆，ch[0][1]为根

ch[0][1]=top=0;

for(int i=1;i<=n;++i){

scanf("%d",a+i);

while(top&&a[i]>a[stk[top]]) --top;

ch[i][1]=0;

ch[i][0]=ch[stk[top]][1];

ch[stk[top]][1]=i;

stk[++top]=i;

}

ans=1;

dfs(ch[0][1]);

1. **左偏树（可并堆，支持删除任意结点）**

//以大根堆为例

int rt[maxn],ls[maxn],rs[maxn],dis[maxn],w[maxn],par[maxn];

int merge(int x,int y)

{

if(!x||!y) return x+y;

if(w[x]<w[y]) swap(x,y);

rs[x]=merge(rs[x],y);

if(rs[x]) par[rs[x]]=x;

if(dis[ls[x]]<dis[rs[x]]) swap(ls[x],rs[x]);

dis[x]=dis[rs[x]]+1;

return x;

}

int pop(int x)//删除任意结点,返回值是树根

{

int l=ls[x],r=rs[x],q=par[x];

par[x]=ls[x]=rs[x]=dis[x]=0;

int p=merge(l,r);

par[p]=q;

if(!q) return p;

if(ls[q]==x) ls[q]=p;

else rs[q]=p;

while(q){

if(dis[ls[q]]<dis[rs[q]]) swap(ls[q],rs[q]);

if(dis[rs[q]]+1==dis[q]) break;

dis[q]=dis[rs[q]]+1;

q=par[q];

}

return rt;

}

1. **并查集**

//路径压缩

int find(int x){return fa[x]==x?x:fa[x]=find(fa[x]);}

//按秩合并（保留树形态，且树高logn）

int find(int x)

{

while(fa[x]!=x) x=fa[x];

return x;

}

int unite(int x,int y)

{

int fx=find(x),fy=find(y);

if(fx==fy) return 0;

if(rk[fx]>rk[fy]) swap(fx,fy);

fa[fx]=fy;

if(rk[fx]==rk[fy]) ++rk[fy];

return fx;

}

//将返回值存栈可以实现撤销操作

1. **对顶堆**

处理动态中位数等问题，灵活运用了堆的性质，本质是维护两个堆。

大根堆Q1：维护集合中较小值的部分的最大值。

小根堆Q2：维护集合中较大值的部分的最小值。

注意到两个堆中的元素各自是单调的，两个堆间也是单调的。也就是说，Q1中的任何一个元素都不大于Q2中的任何一个元素。

//插入和调整

if(Q2.empty()||x<-Q2.top()) Q1.push(x);

else Q2.push(-x);

while(Q1.size()>Q2.size()+1) Q2.push(-Q1.top()),Q1.pop();

while(Q2.size()>Q1.size()) Q1.push(-Q2.top()),Q2.pop();

1. **树论**
2. **树链剖分（注意点权和边权区别）**

//以边权为例

int top[maxn],son[maxn],sz[maxn],fa[maxn],depth[maxn],id[maxn],vs[maxn];

void dfs1(int u,int p,int d)

{

fa[u]=p;

depth[u]=d;

son[u]=0,sz[u]=1;

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(v==p) continue;

dfs1(v,u,d+1);

sz[u]+=sz[v];

if(!son[u]||sz[v]>sz[son[u]]) son[u]=v;

}

}

void dfs2(int u,int sp)

{

top[u]=sp;

id[u]=++tot;

vs[tot]=u;

if(son[u]) dfs2(son[u],sp);

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(v==fa[u]||v==son[u]) continue;

dfs2(v,v);

}

}

void change(int u,int v,int val)

{

int fx=top[u],fy=top[v];

while(fx!=fy){

if(depth[fx]<depth[fy]) swap(fx,fy),swap(u,v);

update(id[fx],id[u],val,1,n,1);

u=fa[fx];

fx=top[u];

}

if(u==v) return; //u==v时无边，返回

if(depth[u]>depth[v]) swap(u,v);

update(id[son[u]],id[v],val,1,n,1);

}

int Qsum(int u,int v)

{

int fx=top[u],fy=top[v],ans=0;

while(fx!=fy){

if(depth[fx]<depth[fy]) swap(fx,fy),swap(u,v);

ans+=Query(id[fx],id[u],1,n,1);

u=fa[fx];

fx=top[u];

}

if(u==v) return ans;

if(depth[u]>depth[v]) swap(u,v);

ans+=Query(id[son[u]],id[v],1,n,1);

return ans;

}

1. **dsu on tree（O(nlogn)处理子树相关静态查询问题）**

//以查询u子树往下走k步到达结点数为例

int fa[max\_logn][maxn],son[maxn],sz[maxn],cnt[maxn],ans[maxn];

bool vis[maxn];

vector<pair<int,int> >vec[maxn];

inline void update(int u,int d,int val)

{

cnt[d]+=val;

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(v==fa[0][u]||vis[v]) continue;

update(v,d+1,val);

}

}

inline void dfs2(int u,int d,int keep)

{

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(v==fa[0][u]||v==son[u]) continue;

dfs2(v,d+1,0);

}

if(son[u]) dfs2(son[u],d+1,1),vis[son[u]]=1;

update(u,d,1);

for(auto elem:vec[u]){

ans[elem.second]=cnt[d+elem.first]-1;

}

vis[son[u]]=0;

if(!keep) update(u,d,-1);

}

1. **LCA**

**3.1树上倍增**

//预处理O(nlogn),查询O(logn)

//dfs预处理

int fa[max\_logn][maxn],depth[maxn],n;

void init()

{

for(int j=1;1<<j<n;++j){

for(int i=1;i<=n;++i){

fa[j][i]=fa[j-1][fa[j-1][i]];

}

}

}

int Lca(int u,int v)

{

if(depth[u]>depth[v]) swap(u,v);

for(int d=depth[v]-depth[u],i=0;d;d>>=1,++i){

if(d&1) v=fa[i][v];

}

if(u==v) return u;

for(int i=max\_logn-1;i>=0;--i){

if(fa[i][u]!=fa[i][v]){

u=fa[i][u];

v=fa[i][v];

}

}

return fa[0][u];

}

**3.2欧拉序+rmq**

//预处理O(nlogn),查询O(1)

//[firstu,firstv]过程中，回溯到深度最小的结点就是u，v的LCA

//maxm=2\*maxn

int id[maxn],vs[maxm],depth[maxm],dp[max\_logm][maxm],mm[maxm],tot;

void dfs(int u,int p,int d)

{

id[u]=++tot;

vs[tot]=u;

depth[tot]=d;

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(v==p) continue;

dfs(v,u,d+1);

vs[++tot]=u;

depth[u]=d;

}

}

void init(int n)

{

mm[0]=0;

for(int i=1;i<maxm;++i){

mm[i]=mm[i-1];

if(i==1<<(mm[i]+1)) ++mm[i];

}

dfs(1,0,0);

int N=n<<1;

for(int i=1;i<=N;++i) dp[0][i]=i;

for(int j=1;j<=mm[N];++j){

for(int i=1;i+(1<<j)-1<=N;++i){

int x=dp[j-1][i],y=dp[j-1][i+(1<<(j-1))];

if(depth[x]<depth[y]) dp[j][i]=x;

else dp[j][i]=y;

}

}

}

int Rmq(int l,int r)

{

int k=mm[r-l+1],x=dp[k][l],y=dp[k][r-(1<<k)+1];

return depth[x]<depth[y]?x:y;

}

int Lca(int u,int v)

{

if(id[u]>id[v]) swap(u,v);

return vs[Rmq(id[u],id[v])];

}

**3.3离线Tarjan算法**

//离线算法，复杂度O(q)

//需要并查集

struct Ask

{

int v,lca,id,rev;

};

vector<Ask>vec[maxn];

bool vis[maxn];

int fa[maxn];

void dfs(int u,int p)

{

vis[u]=1;

int v;

for(int i=head[u];i!=-1;i=edge[i].next){

v=edge[i].to;

if(v==p) continue;

dfs(v,u);

fa[v]=u;

}

for(auto &elem:vec[u]){

v=elem.v;

if(vis[v]){

elem.lca=vec[v][elem.rev].lca=find(v);

}

}

}

1. **字符串**
2. **Trie树**

struct Trie

{

int ch[maxn][26],en[maxn],tot,rt;

void init(){tot=0;rt=newNode();}

int newNode()

{

++tot;

memset(ch[tot],0,sizeof(ch[tot]));

en[tot]=0;

return tot;

}

void Insert(char \*s)

{

int len=strlen(s),r=rt;

for(int i=0;i<len;++i){

int c=s[i]-'a';

if(!ch[r][c]) ch[r][c]=newNode();

r=ch[r][c];

}

en[r]=1;

return;

}

bool Query(char \*s)

{

int len=strlen(s),r=rt;

for(int i=0;i<len;++i){

int c=s[i]-'a';

if(!ch[r][c]) return 0;

r=ch[r][c];

}

return en[r];

}

};

1. **后缀数组**

/\*字符串下标0~n-1，maxn为字符串长

int r[maxn],c[maxn],t0[maxn],t1[maxn],sa[maxn],rk[maxn],height[maxn];

char s[maxn];

void da(int n,int m)

{

++n;

int i,j,p,\*x=t0,\*y=t1;

for(i=0;i<m;i++) c[i]=0;

for(i=0;i<n;i++) ++c[x[i]=r[i]];

for(i=1;i<m;i++) c[i]+=c[i-1];

for(i=n-1;i>=0;i--) sa[--c[x[i]]]=i;

for(j=1;j<=n;j<<=1)

{

p=0;

for(i=n-j;i<n;i++) y[p++]=i;

for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;

for(i=0;i<m;i++) c[i]=0;

for(i=0;i<n;i++) ++c[x[i]];

for(i=1;i<m;i++) c[i]+=c[i-1];

for(i=n-1;i>=0;i--) sa[--c[x[y[i]]]]=y[i];

swap(x,y);

p=1;x[sa[0]]=0;

for(i=1;i<n;i++)

x[sa[i]]=y[sa[i-1]]==y[sa[i]]&&y[sa[i-1]+j]==y[sa[i]+j]?p-1:p++;

if(p>=n) break;

m=p;

}

--n;

int k=0;

for(i=0;i<=n;i++) rk[sa[i]]=i;

for(i=0;i<n;i++)

{

if(k) --k;

j=sa[rk[i]-1];

while(r[i+k]==r[j+k]) ++k;

height[rk[i]]=k;

}

return;

}

1. **回文树**

struct PTree

{

int ch[maxn][26],len[maxn],fail[maxn],s[maxn],tot,n,last;

void init()

{

n=tot=last=0;

fail[0]=1,s[0]=-1;

newNode(0),newNode(-1);

}

int newNode(int \_len)

{

memset(ch[tot],0,sizeof(ch[tot]));

len[tot]=\_len;

return tot++;

}

int gfail(int x,int l)

{

while(s[l-len[x]-1]!=s[l]) x=fail[x];

return x;

}

void add(int c)

{

s[++n]=c;

int cp=gfail(last,n);

if(!ch[cp][c])

{

int nn=newNode(len[cp]+2);

fail[nn]=ch[gfail(fail[cp],n)][c];

ch[cp][c]=nn;

Addedge(fail[nn],nn);

}

last=ch[cp][c];

}

};

1. **最小表示法**

//s起始地址，len为字符串长度，采用2倍空间或模len

inline int minpos(char \*s,int len)

{

int i=0,j=1,k=0;

for(;i<len&&j<len&&k<len;){

if(s[(i+k)%len]!=s[(j+k)%len]){

if(s[(i+k)%len]>s[(j+k)%len]){

i=max(j,i+k+1);

j=i+1;

}

else j+=k+1;

k=0;

}

else ++k;

}

return i;

}

**四、图论**

1. **费用流**

**1.1 Dijkstra费用流**

struct Dflow {

struct edge {

int to, capacity, cost, rev;

edge() {}

edge(int to, int \_capacity, int \_cost, int \_rev) :to(to), capacity(\_capacity), cost(\_cost), rev(\_rev) {}

};

int V, H[maxn + 5], dis[maxn + 5], PreV[maxn + 5], PreE[maxn + 5];

vector<edge> G[maxn + 5];

//调用前初始化

void Init(int n) {

V = n;

for (int i = 0; i <= V; ++i)G[i].clear();

}

//加边

void Add\_Edge(int from, int to, int cap, int cost) {

G[from].push\_back(edge(to, cap, cost, G[to].size()));

G[to].push\_back(edge(from, 0, -cost, G[from].size() - 1));

}

//flow是自己传进去的变量，就是最后的最大流，返回的是最小费用

int Min\_cost\_max\_flow(int s, int t, int f, int& flow) {

int res = 0; fill(H, H + 1 + V, 0);

while (f) {

priority\_queue <pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> > q;

fill(dis, dis + 1 + V, INF);

dis[s] = 0; q.push(pair<int, int>(0, s));

while (!q.empty()) {

pair<int, int> now = q.top(); q.pop();

int v = now.second;

if (dis[v] < now.first)continue;

for (int i = 0; i < G[v].size(); ++i) {

edge& e = G[v][i];

if (e.capacity > 0 && dis[e.to] > dis[v] + e.cost + H[v] - H[e.to]) {

dis[e.to] = dis[v] + e.cost + H[v] - H[e.to];

PreV[e.to] = v;

PreE[e.to] = i;

q.push(pair<int, int>(dis[e.to], e.to));

}

}

}

if (dis[t] == INF)break;

for (int i = 0; i <= V; ++i)H[i] += dis[i];

int d = f;

for (int v = t; v != s; v = PreV[v])d = min(d, G[PreV[v]][PreE[v]].capacity);

f -= d; flow += d; res += d\*H[t];

for (int v = t; v != s; v = PreV[v]) {

edge& e = G[PreV[v]][PreE[v]];

e.capacity -= d;

G[v][e.rev].capacity += d;

}

}

return res;

}

};

**1.2 Spfa费用流**

struct E

{

int to,next,cap,flow,cost;

}edge[maxm];

int head[maxn],tol;

inline void Addedge(int u,int v,int w,int c)

{

edge[tol].to=v;edge[tol].cap=w;edge[tol].flow=0;edge[tol].cost=c;

edge[tol].next=head[u];head[u]=tol++;

edge[tol].to=u;edge[tol].cap=edge[tol].flow=0;edge[tol].cost=-c;

edge[tol].next=head[v];head[v]=tol++;

return;

}

struct Spfaflow

{

int d[maxn],cur[maxn],s,t,n;

bool vis[maxn];

queue<int>Q;

inline void init(){memset(head,-1,sizeof(head));tol=0;}

inline bool spfa()

{

memset(d,0x3f,sizeof(int)\*n);

memset(vis,0,sizeof(bool)\*n);

d[s]=0;

Q.push(s);

while(!Q.empty()){

int u=Q.front();Q.pop();

vis[u]=0;

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(edge[i].cap>edge[i].flow&&d[v]>d[u]+edge[i].cost){

d[v]=d[u]+edge[i].cost;

if(!vis[v]){

vis[v]=1;

Q.push(v);

}

}

}

}

return d[t]!=INF;

}

inline int dfs(int u,int a)

{

if(u==t) return a;

int f=0;

vis[u]=1;

for(int &i=cur[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(!vis[v]&&edge[i].cap>edge[i].flow&&d[v]==d[u]+edge[i].cost){

int tmp=dfs(v,min(a,edge[i].cap-edge[i].flow));

if(tmp){

edge[i].flow+=tmp;

edge[i^1].flow-=tmp;

f+=tmp;

a-=tmp;

if(!a) break;

}

}

}

return f;

}

inline int Maxflow(int s,int t,int n,int&cost)

{

this->s=s,this->t=t,this->n=n;

int ans=0;

while(spfa()){

memcpy(cur,head,sizeof(int)\*n);

while(1){

memset(vis,0,sizeof(bool)\*n);

int tmp=dfs(s,INF);

if(!tmp) break;

ans+=tmp;

cost+=tmp\*d[t];

}

}

return ans;

}

};

**1.3 ZKW费用流**

//适用于流量大，费用小，增广路径短的图，每次至少增加一条可行边

struct E

{

int to,next,cap,flow,cost;

}edge[maxm];

int head[maxn],tol;

inline void Addedge(int u,int v,int w,int c)

{

edge[tol].to=v;edge[tol].cap=w;edge[tol].flow=0;edge[tol].cost=c;

edge[tol].next=head[u];head[u]=tol++;

edge[tol].to=u;edge[tol].cap=edge[tol].flow=0;edge[tol].cost=-c;

edge[tol].next=head[v];head[v]=tol++;

return;

}

struct Zkwflow

{

int d[maxn],cur[maxn],t,n;

bool vis[maxn];

inline void init(){memset(head,-1,sizeof(head));tol=0;}

inline int dfs(int u,int a)

{

if(u==t) return a;

vis[u]=1;

for(int &i=cur[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(!vis[v]&&edge[i].cap>edge[i].flow&&d[v]==d[u]+edge[i].cost){

int tmp=dfs(v,min(a,edge[i].cap-edge[i].flow));

if(tmp){

edge[i].flow+=tmp;

edge[i^1].flow-=tmp;

return tmp;

}

}

}

return 0;

}

inline bool mdf()

{

int Min=INF;

for(int u=0;u<n;u++){

if(!vis[u]) continue;

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(edge[i].cap>edge[i].flow&&!vis[v]&&Min>d[u]-d[v]+edge[i].cost){

Min=d[u]-d[v]+edge[i].cost;

}

}

}

if(Min==INF) return 0;

for(int i=0;i<n;i++){

if(vis[i]) d[i]-=Min;

}

return 1;

}

inline int Maxflow(int s,int t,int n,int &cost)

{

this->t=t,this->n=n;

memset(d,0,sizeof(int)\*n);

int ans=0;

do{

memcpy(cur,head,sizeof(int)\*n);

while(1){

memset(vis,0,sizeof(bool)\*n);

int tmp=dfs(s,INF);

if(!tmp) break;

ans+=tmp;

cost+=-d[s]\*tmp;

}

}while(mdf());

return ans;

}

};

1. **匈牙利算法求二分图最小点覆盖方案**

struct E

{

int to,next;

}edge[maxm];

int head[maxn],tol;

inline void Addedge(int u,int v)

{

edge[tol].to=v;edge[tol].next=head[u];head[u]=tol++;

}

int a[maxn];

int lin[maxn],rit[maxn];

bool visx[maxn],visy[maxn];

vector<int>S,T;

inline bool dfs(int u)

{

visx[u]=1;

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(!visy[v]){

visy[v]=1;

if(lin[v]==-1||dfs(lin[v])){

lin[v]=u;

rit[u]=v;

return 1;

}

}

}

return 0;

}

inline int hungry()

{

int ans=0;

memset(lin,-1,sizeof(lin));

memset(rit,-1,sizeof(rit));

for(auto u:S){

memset(visx,0,sizeof(visx));

memset(visy,0,sizeof(visy));

if(dfs(u)) ++ans;

}

return ans;

}

inline void mincover()

{

memset(visx,0,sizeof(visx));

memset(visy,0,sizeof(visy));

for(auto u:S){

if(rit[u]==-1) dfs(u);

}

vector<int>Ans;

for(auto u:S) if(!visx[u]) Ans.push\_back(a[u]);

for(auto v:T) if(visy[v]) Ans.push\_back(a[v]);

for(int i=0;i<Ans.size();++i) printf("%d%c",Ans[i],i==Ans.size()-1?'\n':' ');

return;

}

1. **割点、桥**

struct E

{

int to,next,cut;

}edge[maxm];

int head[maxn],tol;

inline void Addedge(int u,int v)

{

edge[tol].to=v;edge[tol].cut=0;

edge[tol].next=head[u];head[u]=tol++;

return;

}

int dfn[maxn],low[maxn],cut[maxn],tot;

inline void Tarjan(int u,int p)

{

dfn[u]=low[u]=++tot;

int pcnt=0,son=0;

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(v==p&&!pcnt){++pcnt;continue;}

if(!dfn[v]){

++son;

Tarjan(v,u);

if(low[v]<low[u]) low[u]=low[v];

if(low[v]>=dfn[u]) cut[u]=1;

if(low[v]>dfn[u]){

edge[i].cut=edge[i^1].cut=1;

}

}

else if(dfn[v]<low[u]){

low[u]=dfn[v];

}

}

if(!p&&son==1) cut[u]=0;

return;

}

1. **边双连通分量**

struct E

{

int to,next;

}edge[maxm];

int head[maxn],tol;

inline void Addedge(int u,int v)

{

edge[tol].to=v;edge[tol].next=head[u];head[u]=tol++;

return;

}

int dfn[maxn],low[maxn],bel[maxn],stk[maxn],top,blocks,tot;

inline void Tarjan(int u,int p)

{

dfn[u]=low[u]=++tot;

stk[top++]=u;

int pcnt=0,v;

for(int i=head[u];i!=-1;i=edge[i].next){

v=edge[i].to;

if(v==p&&!pcnt){++pcnt;continue;}

if(!dfn[v]){

Tarjan(v,u);

if(low[v]<low[u]) low[u]=low[v];

}

else if(dfn[v]<low[u]){

low[u]=dfn[v];

}

}

if(low[u]==dfn[u]){

++blocks;

do{

v=stk[--top];

bel[v]=blocks;

}while(v!=u);

}

return;

}

1. **点双连通分量**

struct E

{

int to,next;

}edge[maxm];

int head[maxn],tol;

inline void Addedge(int u,int v,int id)

{

edge[tol].to=v;edge[tol].next=head[u];head[u]=tol++;

return;

}

int dfn[maxn],low[maxn],bel[maxn],stk[maxm],top,blocks,tot;

vector<int>bcc[maxn];

inline void Tarjan(int u,int p)

{

dfn[u]=low[u]=++tot;

int pcnt=0;

for(int i=head[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(v==p&&!pcnt){++pcnt;continue;}

if(!dfn[v]){

stk[top++]=i;

Tarjan(v,u);

if(low[v]<low[u]) low[u]=low[v];

if(low[v]>=dfn[u]){

bcc[++blocks].clear();

do{

E&e=edge[stk[--top]];

if(bel[e.from]!=blocks){

bel[e.from]=blocks;

bcc[blocks].push\_back(e.from);

}

if(bel[e.to]!=blocks){

bel[e.to]=blocks;

bcc[blocks].push\_back(e.to);

}

if(e.from==u&&e.to==v) break;

}while(1);

}

}

else if(dfn[v]<dfn[u]){

//这里要dfn[v]<dfn[u],确保这个点双连通分量求出前加入该边

stk[top++]=i;

low[u]=min(low[u],dfn[v]);

}

}

return;

}

1. **强连通分量**

struct E

{

int to,next;

}edge[maxm];

int head[maxn],tol;

inline void Addedge(int u,int v)

{

edge[tol].to=v;edge[tol].next=head[u];head[u]=tol++;

return;

}

int dfn[maxn],low[maxn],stk[maxn],bel[maxn],tot,top,blocks;

inline void Tarjan(int u)

{

dfn[u]=low[u]=++tot;

stk[top++]=u;

int v;

for(int i=head[u];i!=-1;i=edge[i].next){

v=edge[i].to;

if(!dfn[v]){

Tarjan(v);

if(low[v]<low[u]) low[u]=low[v];

}

else if(!bel[v]&&dfn[v]<low[u]) low[u]=dfn[v];

//bel[v]==0确保v在当前scc中

}

if(low[u]==dfn[u]){

++blocks;

do{

v=stk[--top];

bel[v]=blocks;

}while(v!=u);

}

return;

}

1. **2-SAT**

求解n个布尔变量满足合取范式的解，每个变量两个取值，实际上代表两种状态的选择。

一些建边的技巧

（1）同或建边（x,y)，(x^1,y^1)，（y,x），(y^1,x^1)

（2）异或建边（x,y^1），（x^1,y），（y,x^1），（y^1,x）

（3）某些变量固定取值（如x=1），（x^1,x）构造反状态的矛盾即可

**7.1 dfs染色求最小字典序解**

struct E

{

int to,next;

}edge[maxm];

int head[maxn],tol;

inline void Addedge(int u,int v)

{

edge[tol].to=v;edge[tol].next=head[u];head[u]=tol++;

return;

}

bool vis[maxn];

int stk[maxn],top;

inline bool dfs(int u)

{

if(vis[u^1]) return 0;

if(vis[u]) return 1;

vis[u]=1;

stk[top++]=u;

for(int i=head[u];i!=-1;i=edge[i].next){

if(!dfs(edge[i].to)) return 0;

}

return 1;

}

**7.2 Tarjan强连通分量求法**

struct E

{

int to,next;

}edge[maxm];

int head[maxn],tol;

inline void Addedge(int u,int v)

{

edge[tol].to=v;edge[tol].next=head[u];head[u]=tol++;

return;

}

int dfn[maxn],low[maxn],bel[maxn],stk[maxn],tot,top,blocks;

inline void Tarjan(int u)

{

dfn[u]=low[u]=++tot;

stk[top++]=u;

int v;

for(int i=head[u];i!=-1;i=edge[i].next){

v=edge[i].to;

if(!dfn[v]){

Tarjan(v);

if(low[v]<low[u]) low[u]=low[v];

}

else if(!bel[v]&&dfn[v]<low[u]) low[u]=dfn[v];

}

if(low[u]==dfn[u]){

++blocks;

do{

v=stk[--top];

bel[v]=blocks;

}while(v!=u);

}

return;

}

inline void init()

{

memset(head,-1,sizeof(head));

memset(dfn,0,sizeof(dfn));

memset(bel,0,sizeof(bel));

tol=top=tot=blocks=0;

return;

}

inline bool solve(int n)

{

for(int i=0;i<n;i++) if(!dfn[i]) Tarjan(i);

for(int i=0;i<n;i+=2){

if(bel[i]==bel[i^1]) return 0;

}

return 1;

}

**五、数论**

**1. 素数筛法**

**1.1 埃氏筛法（最大质因数）**

bool vis[maxn];

int prime[maxn];

void init()

{

for(int i=2;i<maxn;++i){

if(!vis[i]){

prime[++prime[0]]=i;

vis[i]=1;

for(int j=i\*i;j<maxn;j+=i) vis[j]=1;

}

}

}

**1.2 欧拉筛法（最小质因数）**

int prime[maxn+1];

void init()

{

memset(prime,0,sizeof(prime));

for(int i=2;i<=maxn;++i) {

if(!prime[i]) prime[++prime[0]]=i;

for(int j=1;j<=prime[0]&&prime[j]<=maxn/i;++j) {

prime[prime[j]\*i]=1;

if(i%prime[j]==0) break;

}

}

}

**1.3 Miller Rabin素数测试**

bool millerRabbin(int n) {

if (n < 3) return n == 2;

int a = n - 1, b = 0;

while (a % 2 == 0) a /= 2, ++b;

// test\_time 为测试次数,建议设为不小于 8

// 的整数以保证正确率,但也不宜过大,否则会影响效率

for (int i = 1, j; i <= test\_time; ++i) {

int x = rand() % (n - 2) + 2, v = powmod(x, a, n);

if (v == 1 || v == n - 1) continue;

for (j = 0; j < b; ++j) {

v = (ll)v \* v % n;

if (v == n - 1) break;

}

if (j >= b) return 0;

}

return 1;

}

**1.4大区间素数筛选（区间长度较小）**

//用根号n以内质数去筛掉大区间的合数

inline int solve(int L,int R)

{

memset(vis,0,sizeof(vis));

if(L==1) ++L;

for(int i=1;i<=prime[0]&&(ll)prime[i]\*prime[i]<=R;++i){

int j=L/prime[i]+(L%prime[i]>0);

if(j==1) ++j;

for(;(ll)prime[i]\*j<=R;++j){

vis[prime[i]\*j-L]=1;

}

}

int ans=0;

for(int i=0;i<=R-L;++i) if(!vis[i]) ++ans;

return ans;

}

**2. 欧几里德算法**

**2.1欧几里德算法（最大公约数）**

int gcd(int x,int y){return y?gcd(y,x%y):x;}

**2.2扩展欧几里德算法（解不定方程ax+by=gcd(a,b)）**

int ex\_gcd(int a,int b,int &x,int &y)

{

if(!b){x=1,y=0;return a;}

int g=ex\_gcd(b,a%b,y,x);

y-=a/b\*x;

return g;

}

**2.3类欧几里得算法**

//计算，时间复杂度O(logc)







**2.3.1计算f函数**

inline ll f(ll a,ll b,ll c,ll n)

{

ll ac=a/c,bc=b/c,m=(a\*n+b)/c;

if(!a) return bc\*(n+1)%mod;

if(a>=c||b>=c)

return (n\*(n+1)/2%mod\*ac%mod+(n+1)\*bc%mod+f(a%c,b%c,c,n))%mod;

return (n\*m%mod-f(c,c-b-1,a,m-1)+mod)%mod;

}

**2.3.2计算f,g,h函数**

struct Data

{

ll f,g,h;

};

ll inv2,inv6;

inline ll powmod(ll a,ll b)

{

ll ans=1;

while(b){if(b&1) ans=ans\*a%mod;b>>=1;a=a\*a%mod;}

return ans;

}

inline Data cal(ll a,ll b,ll c,ll n)

{

Data res;

ll ac=a/c,bc=b/c,m=(a\*n+b)/c,n1=n+1,n2=2\*n+1;

if(!a){

res.f=n1\*bc%mod;

res.g=n\*n1%mod\*inv2%mod\*bc%mod;

res.h=n1\*bc%mod\*bc%mod;

return res;

}

if(a>=c||b>=c){

res.f=(n\*n1%mod\*inv2%mod\*ac%mod+n1\*bc%mod)%mod;

res.g=(n\*n1%mod\*n2%mod\*inv6%mod\*ac%mod+n\*n1%mod\*inv2%mod\*bc%mod)%mod;

res.h=(n\*n1%mod\*n2%mod\*inv6%mod\*ac%mod\*ac%mod+n1\*bc%mod\*bc%mod+n\*n1%mod\*ac%mod\*bc%mod)%mod;

Data t=cal(a%c,b%c,c,n);

res.f=(res.f+t.f)%mod;

res.g=(res.g+t.g)%mod;

res.h=(res.h+t.h+2\*bc%mod\*t.f%mod+2\*ac%mod\*t.g%mod)%mod;

return res;

}

res.f=n\*m%mod;

res.g=n\*n1%mod\*m%mod;

res.h=n\*m%mod\*(m+1)%mod;

Data t=cal(c,c-b-1,a,m-1);

res.f=(res.f-t.f+mod)%mod;

res.g=(res.g-t.h+mod-t.f+mod)%mod\*inv2%mod;

res.h=(res.h-2\*t.g%mod+mod-2\*t.f%mod+mod-res.f+mod)%mod;

return res;

}

1. **欧拉函数**

**3.1单个数的欧拉函数求法（）**

int phi(int x)

{

int ans=x;

for(int i=2;i\*i<=x;++i){

if(x%i==0){

ans=ans/i\*(i-1);

while(x%i==0) x/=i;

}

}

if(x>1) ans=ans/x\*(x-1);

return ans;

}

**3.2线性筛**

int phi[maxn],prime[maxn];

void init()

{

phi[1]=1;

for(int i=2;i<maxn;++i){

if(!phi[i]){

phi[i]=i-1;

prime[++prime[0]]=i;

}

for(int j=1;j<=prime[0]&&1ll\*i\*prime[j]<maxn;++j){

if(i%prime[j]) phi[i\*prime[j]]=phi[i]\*(prime[j]-1);

else{

phi[i\*prime[j]]=phi[i]\*prime[j];

break;

}

}

}

}

1. **二次剩余（p为奇素数）**

inline ll powmod(ll a,ll b)

{

ll ans=1;

while(b){if(b&1) ans=ans\*a%mod;b>>=1;a=a\*a%mod;}

return ans;

}

inline pair<ll,ll> pMul(const pair<ll,ll>&x,const pair<ll,ll>&y,ll f)

{

return make\_pair((x.first\*y.first%mod+x.second\*y.second%mod\*f%mod)%mod,

(x.second\*y.first%mod+x.first\*y.second%mod)%mod);

}

inline int Quadratic\_residue(int a){

if(!a) return 0;

if(powmod(a,mod-1>>1)!=1) return -1;

ll x, f;

do{

x=(1ll\*rand()<<14^rand())%mod;

}while(powmod(f=(x\*x%mod-a+mod)%mod,mod-1>>1)==1);

pair<ll,ll>ans=make\_pair(1,0),t=make\_pair(x,1);

for(int i=mod+1>>1;i;i>>=1,t=pMul(t,t,f)) if(i&1) ans=pMul(ans,t,f);

return min(ans.first, mod-ans.first);

}

1. **模线性方程组**

**5.1中国剩余定理**





ll China(int n,ll \*a,ll \*m)

{

ll M=1,ans=0;

for(int i=0;i<n;++i) M\*=m[i];

for(int i=0;i<n;++i){

ll w=M/m[i];

ans=(ans+a[i]\*w%mod\*inv(w,m[i])%mod)%mod;

}

return ans;

}

**5.2扩展中国剩余定理**



//当A[i]=1时，a=lcm(m[1],m[2],...m[n])

ll ex\_china(ll \*A,ll \*B,ll \*m,int n)

{

ll a=1,b=0;

for(int i=0;i<n;++i){

ll m0=A[i]\*a,c=B[i]-A[i]\*b,d=gcd(m[i],m0);

if(c%d) return -1;

ll t=c/d\*inv(m0/d,m[i]/d)%(m[i]/d);

b+=a\*t;

a\*=m[i]/d;

}

return (b%a+a)%a;

}

1. **BSGS**

**//求解离散对数**

6.1普通BSGS（a,p互质）

6.2扩展BSGS（a,p不互质）

**六、其他**

1. **线性基**

struct LB

{

u32 v[SZ];

LB(){init();}

void init(){memset(v,0,sizeof(v));}

void Insert(u32 x)

{

for(int i=SZ-1;i>=0;--i){

if(x>>i&1){

if(v[i]) x^=v[i];

else{

v[i]=x;

break;

}

}

}

return;

}

bool check(u32 x)

{

for(int i=SZ-1;i>=0;--i){

if(x>>i&1){

if(v[i]) x^=v[i];

else return 0;

}

}

return 1;

}

LB operator\*(const LB&A)const //线性基的交

{

LB C=A,D=A,Ans;

for(int i=SZ-1;i>=0;--i){

if(!v[i]) continue;

u32 x=v[i],y=0;

int j=i;

for(;j>=0;--j){

if(x>>j&1){

if(C.v[j]) x^=C.v[j],y^=D.v[j];

else break;

}

}

if(x) C.v[j]=x,D.v[j]=y;

else Ans.Insert(y);

}

return Ans;

}

LB operator+(const LB&A)const //线性基的并

{

LB Ans=A;

for(int i=SZ-1;i>=0;--i){

if(v[i]) Ans.Insert(v[i]);

}

return Ans;

}

};

1. **手动加栈**

#pragma comment(linker, "/STACK:1024000000,1024000000")

1. **高效位运算**

•int \_\_builtin\_ffs (unsigned int x)

返回x的最后一位1的是从后向前第几位，比如7368（1110011001000）返回4。

•int \_\_builtin\_clz (unsigned int x)

返回前导的0的个数。

•int \_\_builtin\_ctz (unsigned int x)

返回后面的0个个数，和\_\_builtin\_clz相对。

•int \_\_builtin\_popcount (unsigned int x)

返回二进制表示中1的个数。

•int \_\_builtin\_parity (unsigned int x)

返回x的奇偶校验位，也就是x的1的个数模2的结果。

此外，这些函数都有相应的unsigned long和unsigned long long版本，只需要在函数名后面加上l或ll就可以了，比如int \_\_builtin\_clzll

1. **产生随机数**

mt19937 gen(time(0));

mt19937\_64 gen(time(0));

1. **Cdq分治**

//三维偏序

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int maxn=200010;

struct BIT

{

int C[maxn],n;

inline void Add(int x,int val){

while(x<=n){

C[x]+=val;

x+=x&-x;

}

}

inline int Sum(int x){

int ans=0;

while(x){

ans+=C[x];

x-=x&-x;

}

return ans;

}

inline void Clear(int x){

while(x<=n){

C[x]=0;

x+=x&-x;

}

}

}bit;

struct Ask

{

int x,y,z,id;

bool operator<(const Ask&P)const{

return y<P.y;

}

}arr[maxn],tmp[maxn];

ll ans=0;

bool cmp(const Ask&A,const Ask&B){return A.x<B.x;}

void Cdq(int l,int r)

{

if(l+1>=r) return;

int mid=l+r>>1,p=l,q=mid,o=0;

Cdq(l,mid),Cdq(mid,r);

while(p<mid&&q<r){

if(arr[p]<arr[q]){

bit.Add(arr[p].z,1);

tmp[o++]=arr[p++];

}

else{

ans+=bit.Sum(arr[q].z-1);

tmp[o++]=arr[q++];

}

}

while(p<mid) tmp[o++]=arr[p++];

while(q<r){

ans+=bit.Sum(arr[q].z-1);

tmp[o++]=arr[q++];

}

for(int i=0;i<o;++i){

arr[l+i]=tmp[i];

bit.Clear(tmp[i].z);

}

return;

}

1. **莫队算法**

**6.1普通莫队**

**6.2带修莫队（单点修改）**

**//**

#include<bits/stdc++.h>

using namespace std;

const int maxn=11010;

const int maxm=1010;

const int maxv=1000010;

int bel[maxn],cnt[maxv],ret=0;

int w[maxn],ans[maxn];

struct Ask

{

int l,r,id,t;

bool operator<(const Ask&p)const

{

if(bel[l]!=bel[p.l]) return l<p.l;

if(bel[r]!=bel[p.r]) return r<p.r;

return t<p.t;

}

}arr[maxn];

pair<int,int>change[maxn];

inline int getint()

{

int ans=0;

char ch=getchar();

if(ch==EOF) return 0;

while(ch<'0'||ch>'9') ch=getchar();

while(ch>='0'&&ch<='9') ans=ans\*10+ch-'0',ch=getchar();

return ans;

}

inline void add(int pos)

{

if(++cnt[w[pos]]==1) ++ret;

}

inline void del(int pos)

{

if(--cnt[w[pos]]==0) --ret;

}

inline void update(const int &l,const int &r,int i)

{

if(change[i].first>=l&&change[i].first<=r)

{

if(--cnt[w[change[i].first]]==0) --ret;

}

swap(change[i].second,w[change[i].first]);

if(change[i].first>=l&&change[i].first<=r)

{

if(++cnt[w[change[i].first]]==1) ++ret;

}

}

int main()

{

freopen("a.txt","r",stdin);

char ch;

int n,m,tim=0,qnum=0,l,r;

scanf("%d%d",&n,&m);

int blocks=int(ceil(pow(n,2.0/3))+0.2);

for(int i=1;i<=n;i++)

{

scanf("%d",&w[i]);

bel[i]=(i-1)/blocks;

}

for(int i=0;i<m;i++)

{

scanf(" %c%d%d",&ch,&l,&r);

if(ch=='Q') arr[qnum]=Ask{l,r,qnum,tim},++qnum;

else change[++tim]=make\_pair(l,r);

}

sort(arr,arr+qnum);

int L=1,R=0,now=0;

for(int i=0;i<qnum;i++)

{

int l=arr[i].l,r=arr[i].r,t=arr[i].t;

while(now<t) update(L,R,++now);

while(now>t) update(L,R,now--);

while(R<r) add(++R);

while(R>r) del(R--);

while(L<l) del(L++);

while(L>l) add(--L);

ans[arr[i].id]=ret;

}

for(int i=0;i<qnum;i++) printf("%d\n",ans[i]);

return 0;

}

**6.3树上莫队（序列+LCA）**

**//**

#include<bits/stdc++.h>

using namespace std;

const int maxn=40010;

const int maxm=100010;

int bel[maxn<<1],cnt[maxn],st[maxn],en[maxn],vs[maxn<<1],vis[maxn],k=0,ret=0;

int f[maxn],top[maxn],son[maxn],sz[maxn],depth[maxn];

int ans[maxm],w[maxn],head[maxn],num=0;

int h[maxn],tot=0;

struct Ask

{

int l,r,id,lca;

bool operator<(const Ask&p)const

{

if(bel[l]==bel[p.l]) return r<p.r;

return l<p.l;

}

}arr[maxm];

struct E

{

int to,next;

}edge[maxn<<1];

inline void add\_edge(int u,int v)

{

edge[num].to=v;

edge[num].next=head[u];

head[u]=num++;

}

void dfs1(int u,int p,int d)

{

f[u]=p;

depth[u]=d;

son[u]=-1;

sz[u]=1;

st[u]=++k;

vs[k]=u;

for(int i=head[u];i!=-1;i=edge[i].next)

{

int v=edge[i].to;

if(v!=p){

dfs1(v,u,d+1);

sz[v]+=sz[u];

if(son[u]==-1||sz[v]>sz[u]) son[u]=v;

}

}

en[u]=++k;

vs[k]=u;

}

void dfs2(int u,int sp)

{

top[u]=sp;

if(son[u]!=-1) dfs2(son[u],sp);

for(int i=head[u];i!=-1;i=edge[i].next)

{

int v=edge[i].to;

if(v!=f[u]&&v!=son[u]) dfs2(v,v);

}

}

inline int Lca(int u,int v)

{

while(top[u]!=top[v])

{

if(depth[top[u]]<depth[top[v]]) swap(u,v);

u=f[top[u]];

}

return depth[u]<depth[v]?u:v;

}

inline void add(int pos)

{

if(++cnt[h[vs[pos]]]==1) ++ret;

}

inline void del(int pos)

{

if(--cnt[h[vs[pos]]]==0) --ret;

}

inline void update(int pos)

{

vis[vs[pos]]?del(pos):add(pos);

vis[vs[pos]]^=1;

}

int main()

{

freopen("a.txt","r",stdin);

int n,m;

memset(head,-1,sizeof(head));

scanf("%d%d",&n,&m);

for(int i=1;i<=n;i++)

{

scanf("%d",&w[i]);

h[i]=w[i];

}

sort(w+1,w+1+n);

tot=unique(w+1,w+1+n)-w-1;

for(int i=1;i<=n;i++) h[i]=lower\_bound(w+1,w+1+tot,h[i])-w;

int blocks=int(ceil(sqrt(2.0\*n))+0.2);

for(int i=1;i<=n<<1;i++) bel[i]=(i-1)/blocks;

for(int i=1;i<n;i++)

{

int u,v;

scanf("%d%d",&u,&v);

add\_edge(u,v);

add\_edge(v,u);

}

dfs1(1,0,0);

dfs2(1,1);

for(int i=0;i<m;i++)

{

int u,v,t;

scanf("%d%d",&u,&v);

if(st[u]>st[v]) swap(u,v);

t=Lca(u,v);

if(t==u) arr[i].l=st[u],arr[i].r=st[v],arr[i].lca=0;

else arr[i].l=en[u],arr[i].r=st[v],arr[i].lca=st[t];

arr[i].id=i;

}

sort(arr,arr+m);

int L=1,R=0;

for(int i=0;i<m;i++)

{

int l=arr[i].l,r=arr[i].r,t=arr[i].lca;

while(R<r) update(++R);

while(R>r) update(R--);

while(L<l) update(L++);

while(L>l) update(--L);

if(t) update(t);

ans[arr[i].id]=ret;

if(t) update(t);

}

for(int i=0;i<m;i++) printf("%d\n",ans[i]);

return 0;

}

**6.4树上带修莫队**

//

#include<bits/stdc++.h>

using namespace std;

const int maxn=50010;

int bel[maxn],stk[maxn],stop,gs,B;//树分块

int f[maxn],top[maxn],depth[maxn],son[maxn],sz[maxn];//树剖求Lca

int Num[maxn<<1],blocks;//统计答案用

int w[maxn],cnt[maxn<<1],vis[maxn],ans[maxn],N;

struct Ask

{

int u,v,id,t,lca;

bool operator<(const Ask&p)const

{

if(bel[u]!=bel[p.u]) return bel[u]<bel[p.u];

if(bel[v]!=bel[p.v]) return bel[v]<bel[p.v];

return t<p.t;

}

}arr[maxn];

pair<int,int>change[maxn];

struct E

{

int to,next;

}edge[maxn<<1];

int head[maxn],num=0;

inline void add\_edge(int u,int v)

{

edge[num].to=v;

edge[num].next=head[u];

head[u]=num++;

}

inline void dfs1(int u,int p,int d,int bot)

{

f[u]=p;

depth[u]=d;

son[u]=-1;

sz[u]=1;

for(int i=head[u];i!=-1;i=edge[i].next)

{

int v=edge[i].to;

if(v!=p){

dfs1(v,u,d+1,stop);

sz[u]+=sz[v];

if(son[u]==-1||sz[v]>sz[son[u]]) son[u]=v;

if(stop-bot>=B)

{

++gs;

while(stop>bot) bel[stk[stop--]]=gs;

}

}

}

stk[++stop]=u;

}

inline void dfs2(int u,int sp)

{

top[u]=sp;

if(son[u]!=-1) dfs2(son[u],sp);

for(int i=head[u];i!=-1;i=edge[i].next)

{

int v=edge[i].to;

if(v!=f[u]&&v!=son[u]) dfs2(v,v);

}

}

inline int Lca(int u,int v)

{

while(top[u]!=top[v])

{

if(depth[top[u]]<depth[top[v]]) swap(u,v);

u=f[top[u]];

}

return depth[u]<depth[v]?u:v;

}

inline int query()

{

int i=0,j;

for(;i<=N/blocks;i++) if(Num[i]) break;

for(j=i\*blocks;j<(i+1)\*blocks;j++) if(!cnt[j]) break;

return j;

}

inline void add(int x)

{

if(x<=N&&++cnt[x]==1) --Num[x/blocks];

}

inline void del(int x)

{

if(x<=N&&--cnt[x]==0) ++Num[x/blocks];

}

inline void Update(int i)

{

if(vis[change[i].first])

{

del(w[change[i].first]);

}

swap(change[i].second,w[change[i].first]);

if(vis[change[i].first])

{

add(w[change[i].first]);

}

}

inline void rev(int u)

{

vis[u]?del(w[u]):add(w[u]);

vis[u]^=1;

}

inline void mov(int u,int v)

{

int t=Lca(u,v);

while(u!=t) rev(u),u=f[u];

while(v!=t) rev(v),v=f[v];

}

inline int getint()

{

int ans=0;

char ch=getchar();

if(ch==EOF) return 0;

while(ch<'0'||ch>'9') ch=getchar();

while(ch>='0'&&ch<='9') ans=ans\*10+ch-'0',ch=getchar();

return ans;

}

inline void print(int x)

{

if(x>9) print(x/10);

putchar(x%10+'0');

}

int main()

{

// freopen("a.txt","r",stdin);

memset(head,-1,sizeof(head));

int n=getint(),m=getint(),qnum=0,tim=0;

B=int(ceil(pow(n,2.0/3))+0.2);

for(int i=1;i<=n;i++)

{

w[i]=getint();

if(w[i]<maxn<<1) ++cnt[w[i]];

}

for(int i=1;i<n;i++)

{

int u=getint(),v=getint();

add\_edge(u,v);

add\_edge(v,u);

}

dfs1(1,0,0,0);

while(stop) bel[stk[stop--]]=gs;

dfs2(1,1);

for(int i=0;i<m;i++)

{

int op=getint(),x=getint(),y=getint();

if(op){

arr[qnum]=Ask{x,y,qnum,tim,Lca(x,y)};

++qnum;

}

else {

change[++tim]=make\_pair(x,y);

if(y<maxn<<1) ++cnt[y];

}

}

for(N=0;N<maxn<<1;N++) if(cnt[N]==0) break;

memset(cnt,0,sizeof(cnt));

blocks=int(ceil(sqrt(N))+0.2);

for(int i=0;i<=N;i++) ++Num[i/blocks];

sort(arr,arr+qnum);

int u=1,v=1,now=0;

for(int i=0;i<qnum;i++)

{

int nu=arr[i].u,nv=arr[i].v,t=arr[i].t;

while(now<t) Update(++now);

while(now>t) Update(now--);

mov(u,nu);u=nu;

mov(v,nv);v=nv;

rev(arr[i].lca);

ans[arr[i].id]=query();

rev(arr[i].lca);

}

for(int i=0;i<qnum;i++)

{

print(ans[i]);

putchar('\n');

}

return 0;

}

1. **计算几何**
2. 动态凸包

//以下凸壳为例

struct Point

{

static int cmp;

double x,y;

mutable double slop;

Point(){}

Point(double \_x,double \_y,double \_slop):x(\_x),y(\_y),slop(\_slop){}

bool operator<(const Point&P)const{

if(cmp) return slop<P.slop;

return x!=P.x?x<P.x:y<P.y;

}

Point operator-(const Point&P)const{

return Point(x-P.x,y-P.y,0);

}

double operator\*(const Point&P)const{

return x\*P.y-y\*P.x;

}

};

int Point::cmp=0;

set<Point>S[maxn];

void Insert(set<Point>&S,const Point P)

{

Point::cmp=0;

set<Point>::iterator it=S.lower\_bound(P),l,r;

if(it!=S.end()&&it->x==P.x){

S.erase(it);

it=S.lower\_bound(P);

}

l=r=it;

if(l!=S.begin()){

--l;

if(l->x==P.x) return;

if(r!=S.end()&&(\*r-\*l)\*(P-\*l)>=0) return;

}

while(1)

{

it=S.lower\_bound(P);

if(it==S.end()) break;

l=it,it++;

if(it==S.end()) break;

r=it;

if((\*r-P)\*(\*l-P)>=0) S.erase(l);

else break;

}

while(1)

{

it=S.lower\_bound(P);

if(it==S.begin()) break;

it--,r=it;

if(it==S.begin()) break;

it--,l=it;

if((\*l-P)\*(\*r-P)<=0) S.erase(r);

else break;

}

l=r=it=S.insert(P).first;

if(l!=S.begin()){

--l;

l->slop=(P.y-l->y)/(P.x-l->x);

}

++r;

if(r!=S.end()){

it->slop=(r->y-it->y)/(r->x-it->x);

}

else it->slop=INF;

}