## 0.1 并查集

### 带权并查集

```
int find(int x)
{
    if(x!=fa[x])
    {
        int t=fa[x];fa[x]=find(fa[x]);
        v[x]=(v[x]+v[t])%2;
    }
    return fa[x];
}
void lianjie(int x,int y,int s)
{
    int px=find(x),py=find(y);
    fa[px]=py;
    v[px]=(-v[x]+v[y]+s+2)%2;
}
```

## 0.2 单调栈

定义函数$f(i)$代表数列中第$i$个元素之后第一个大于$a_i$的元素的下标。若不存在，则$f(i)=0$。

```
for(int i=n;i>=1;i--)
{
    while(!s.empty() && a[s.top()]<=a[i]) s.pop();
    if(s.size()==0) f[i]=0;
    else f[i]=s.top();
    s.push(i);
}
```

## 0.3 单调队列

有一个长为$n$的序列$a$，以及一个大小为$k$的窗口。现在这个从左边开始向右滑动，每次滑动一个单位，求出每次滑动后窗口中的最大值。

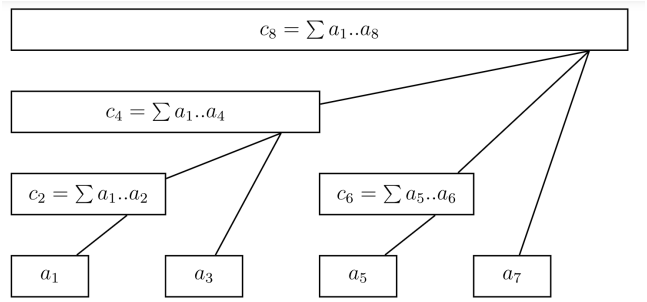
```
deque<int>q;
for(int i=1;i<=n;i++)
```

```
{
    if(!q.empty() && q.front()<=i-k) q.pop_front();
    while(!q.empty() && a[q.back()]<=a[i]) q.pop_back();
    q.push_back(i);
    if(i>=k) printf("%lld\n",a[q.front()]);
}
```

## 0.4 ST表

```
void ST()
{
    for(int j=1;j<21;j++)
    {
        for(int i=1;i+(1<<(j-1))<=n;i++)
        {
            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
        }
    }
}
int query(int l,int r)
{
    int k=log2(r-l+1);
    return max(f[l][k],f[r-(1<<k)+1][k]);
}
```

## 0.5 树状数组




单点修改

```
void update(int num,int x)
{
    for(int i=num;i<=n;i+=lowbit(i)) tre[i]+=x;
}
ll query(int num)
{
    ll sum=0;
    for(int i=num;i>0;i-=lowbit(i)) sum+=tre[i];
    return sum;
}
```

### 区间修改

维护序列$a$的差分数组$b$，此时我们对$a$的一个前缀$r$求和，即$\sum_{i=1}^{r} a_i$，由差分数组定义得$a_i = \sum_{j=1}^{i} b_j$。

$$\begin{aligned}
&\sum_{i=1}^{r} a_i \\
=&\sum_{i=1}^{r}\sum_{j=1}^{i} b_j \\
=&\sum_{i=1}^{r} b_i \times (r-i+1) \\
=&\sum_{i=1}^{r} b_i \times (r+1) - \sum_{i=1}^{r} b_i \times i
\end{aligned}$$

区间和可以用两个前缀和相减得到，因此只需要用两个树状数组分别维护$\sum b_i$和$\sum i \times b_i$，就能实现区间求和。

```
//tre1表示b的前缀和，tre2表示b*i的前缀和，b表示原数组的差分数组
void update(int num,int x)
{
    for(int i=num;i<=n;i+=lowbit(i)) tre1[i]+=x,tre2[i]+=num*x;
}
void update(int l,int r,int x)
{
    update(l,x);update(r+1,-x);
}
ll query(ll tre[],int x)
{
    ll sum=0;
    for(int i=x;i>0;i-=lowbit(i)) sum+=tre[i];
```

```
    return sum;
}
ll query(int x)
{
    return query(tre1,x)*(x+1)-query(tre2,x);
}
```

#### 查询第$k$小/大元素

在此处只讨论第$k$小，第$k$大问题可以通过简单计算转化为第$k$小问题。

找到最大的$x$满足$\sum_{i=1}^{x} a_i < k$，那么$x+1$就是我们的答案。在树状数组中，节点是根据2的幂划分的，每次可以扩大2的幂的长度。令$sum$表示当前的$x$所代表的前缀和，有如下算法找到最大的：

1. 求出$depth = \lfloor \log_2 n \rfloor$
2. 计算$t = \sum_{i=x+1}^{x+2^{depth}} a_i$
3. 如果$sum + t < k$，则此时扩展成功，将$2^{depth}$累加到$x$上；否则扩展失败，对$x$不进行操作
4. 将$depth$减1，回到步骤2，直至$depth$为0

```
// 权值树状数组查询第小k
int kth(int k)
{
    int cnt=0,ret=0;
    for(int i=log2(n);~i;--i) //i与上文depth含义相同
    {
        ret+=1<<i; //尝试扩展
        if(ret>=n||cnt+tre[ret]>=k) //如果扩展失败
            ret-=1<<i;
        else cnt+=tre[ret]; //扩展成功后,要更新之前求和的值
    }
    return ret+1;
}
```

## 0.6 线段树

### 0.6.1 单点修改

```
struct tree
{
    int l,r,sum;
    int mid(){return (l+r)/2;}
}tre[maxn<<2];
void pushup(int num)
{
    tre[num].sum=tre[2*num].sum+tre[2*num+1].sum;
}
void build(int num,int l,int r)
{
    tre[num].l=l,tre[num].r=r;
    if(l==r)
    {
        tre[num].sum=a[l];return;
    }
    int mid=tre[num].mid();
    build(2*num,l,mid);build(2*num+1,mid+1,r);
    pushup(num);
}
int query(int num,int l,int r)
{
    if(tre[num].l==l && tre[num].r==r)
    return tre[num].sum;
    int mid=tre[num].mid();
    if(l>mid) return query(2*num+1,l,r);
    else if(r<=mid) return query(2*num,l,r);
    else return(query(2*num,l,mid)+query(2*num+1,mid+1,r));
}
void add(int num,int n,int k)
{
    if(tre[num].l==tre[num].r) tre[num].sum+=k;return;
    int mid=tre[num].mid();
    if(n<=mid) add(2*num,n,k);
    else add(2*num+1,n,k);
    pushup(num);
}
```

### 0.6.2 区间修改

```
struct tree
```

```
{
    ll l,r,sum,lazy;
    ll mid(){return (l+r)/2;}
}tre[4*maxn];
void build(ll num,ll l,ll r)
{
    tre[num].l=l;tre[num].r=r;
    tre[num].lazy=0;
    if(l==r)
    {
        tre[num].sum=a[l];
        return;
    }
    ll mid=tre[num].mid();
    build(2*num,l,mid);build(2*num+1,mid+1,r);
    tre[num].sum=tre[2*num].sum+tre[2*num+1].sum;
}
void pushdown(ll num)
{
    tre[2*num].sum+=(tre[2*num].r-tre[2*num].l+1)*tre[num].lazy
        ;
    tre[2*num+1].sum+=(tre[2*num+1].r-tre[2*num+1].l+1)*tre[num
        ].lazy;
    tre[2*num].lazy+=tre[num].lazy;
    tre[2*num+1].lazy+=tre[num].lazy;
    tre[num].lazy=0;
}
ll query(ll num,ll l,ll r)
{
    if(tre[num].l==l && tre[num].r==r) return tre[num].sum;
    pushdown(num);
    ll mid=tre[num].mid();
    if(l>mid) return query(2*num+1,l,r);
    else if(r<=mid) return query(2*num,l,r);
    else return query(2*num,l,mid)+query(2*num+1,mid+1,r);
}
void update(ll num,ll l,ll r,ll k)
{
    if(tre[num].l==l && tre[num].r==r)
    {
        tre[num].sum+=(tre[num].r-tre[num].l+1)*k;
        tre[num].lazy+=k;
```

```
42          return;
43      }
44      pushdown(num);
45      ll mid=tre[num].mid();
46      if(l>mid) update(2*num+1,l,r,k);
47      else if(r<=mid) update(2*num,l,r,k);
48      else
49      {
50          update(2*num,l,mid,k);update(2*num+1,mid+1,r,k);
51      }
52      tre[num].sum=tre[2*num].sum+tre[2*num+1].sum;
53  }
```

## 0.7 势能线段树

区间进行lowbit操作

```
1   struct tree
2   {
3       ll l,r,sum,lazy,flag;
4       ll mid(){return (l+r)/2;}
5   }tre[4*maxn];
6   int check(int num)
7   {
8       int cnt=0;
9       for(int i=32;i>=0;i--) if(tre[num].sum>>i&1) cnt++;
10      if(cnt==1)
11      {
12          tre[num].sum%=mod;
13          return 1;
14      }
15      return 0;
16  }
17  void pushdown(ll num)
18  {
19      int lazy=tre[num].lazy;
20      if(lazy==0) return;
21      tre[2*num].sum=(tre[2*num].sum*p[lazy])%mod;
22      tre[2*num+1].sum=(tre[2*num+1].sum*p[lazy])%mod;
23      tre[2*num].lazy+=tre[num].lazy;
24      tre[2*num+1].lazy+=tre[num].lazy;
```

```
25      tre[num].lazy=0;
26  }
27  void pushup(int num)
28  {
29      tre[num].sum=(tre[2*num].sum+tre[2*num+1].sum)%mod;
30      tre[num].flag=tre[2*num].flag&tre[2*num+1].flag;
31  }
32  void build(ll num,ll l,ll r)
33  {
34      tre[num].l=l;tre[num].r=r;
35      tre[num].lazy=0;tre[num].sum=0;tre[num].flag=0;
36      if(l==r)
37      {
38          tre[num].sum=a[l];tre[num].flag=check(num);
39          return;
40      }
41      ll mid=tre[num].mid();
42      build(2*num,l,mid);build(2*num+1,mid+1,r);
43      pushup(num);
44  }
45  ll query(ll num,ll l,ll r)
46  {
47      if(tre[num].l==l && tre[num].r==r)
48      {
49          return tre[num].sum%mod;
50      }
51      pushdown(num);
52      ll mid=tre[num].mid();
53      if(l>mid) return query(2*num+1,l,r);
54      else if(r<=mid) return query(2*num,l,r);
55      else return (query(2*num,l,mid)+query(2*num+1,mid+1,r))%mod
            ;
56  }
57  void update(ll num,ll l,ll r)
58  {
59      ll mid=tre[num].mid();
60      if(tre[num].l==l && tre[num].r==r)
61      {
62          if(tre[num].flag)
63          {
64              tre[num].sum=(tre[num].sum*2)%mod;tre[num].lazy++;
65          }
```

```
        else
        {
            if(l==r)
            {
                tre[num].sum=lowbit(tre[num].sum)+tre[num].sum;
                tre[num].flag=check(num);
            }
            else
            {
                update(2*num,l,mid);update(2*num+1,mid+1,r);
                pushup(num);
            }

        }
        return;
    }
    pushdown(num);
    if(l>mid) update(2*num+1,l,r);
    else if(r<=mid) update(2*num,l,r);
    else
    {
        update(2*num,l,mid);update(2*num+1,mid+1,r);
    }
    pushup(num);
}
```

## 0.8 主席树

区间第$k$大

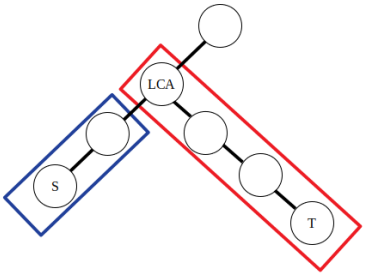
```
int build(int l,int r)
{
    int pos=++cnt;
    lch[pos]=0,rch[pos]=0,sum[pos]=0;
    if(l<r)
    {
        int mid=l+r>>1;
        lch[pos]=build(l,mid);
        rch[pos]=build(mid+1,r);
    }
    return pos;
```

```
}
int update(int pre,int l,int r,ll x)
{
    int pos=++cnt;
    lch[pos]=lch[pre],rch[pos]=rch[pre],sum[pos]=sum[pre];
    if(l<r)
    {
        int mid=l+r>>1;
        if(x<=mid) lch[pos]=update(lch[pre],l,mid,x);
        else rch[pos]=update(rch[pre],mid+1,r,x);
        sum[pos]=sum[lch[pos]]+sum[rch[pos]];
    }
    else sum[pos]++;
    return pos;
}
int query(int x,int y,int l,int r,int k)
{
    if(l==r) return l;
    int mid=l+r>>1;
    int nn=sum[lch[y]]-sum[lch[x]];
    if(nn<k) return query(rch[x],rch[y],mid+1,r,k-nn);
    else return query(lch[x],lch[y],l,mid,k);
}
```

## 0.9 树上差分

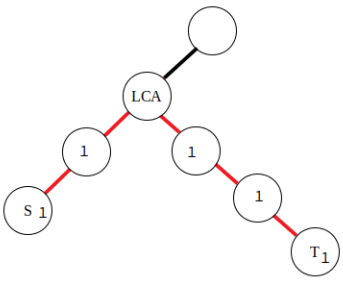
### 0.9.1 点差分

$$d_s \leftarrow d_s + 1$$
$$d_{lca} \leftarrow d_{lca} - 1$$
$$d_t \leftarrow d_t + 1$$
$$d_{f(lca)} \leftarrow d_{f(lca)} - 1$$




### 0.9.2 边差分

$$d_s \leftarrow d_s + 1$$
$$d_t \leftarrow d_t + 1$$
$$d_{lca} \leftarrow d_{lca} - 2$$




## 0.10 树链剖分

重链剖分

```
void dfs1(int u,int fa)
{
    sz[u]=1;big[u]=-1;dep[u]=dep[fa]+1;par[u]=fa;
    for(auto nex:v[u])
    {
        if(nex==fa) continue;
        dfs1(nex,u);sz[u]+=sz[nex];
        if(big[u]==-1||sz[big[u]]<sz[nex]) big[u]=nex;
```

```
    }
}
void dfs2(int u,int fa,int t)
{
    top[u]=t;dfn[u]=++tim;rk[tim]=u;
    if(big[u]==-1) return;
    dfs2(big[u],u,t);
    for(auto nex:v[u])
    {
        if(nex==fa||nex==big[u]) continue;
        dfs2(nex,u,nex);
    }
}
struct SegTree
{
    struct tree
    {
        int l,r;ll sum;
        int mid(){return l+r>>1;}
    }tre[maxn<<2];
    void pushup(int num)
    {
        tre[num].sum=tre[2*num].sum+tre[2*num+1].sum;
    }
    void build(int num,int l,int r)
    {
        tre[num].l=l;tre[num].r=r;
        if(l==r)
        {
            tre[num].sum=a[rk[l]];return;
        }
        int mid=tre[num].mid();
        build(2*num,l,mid);build(2*num+1,mid+1,r);
        pushup(num);
    }
    ll query_sum(int num,int l,int r)
    {
        if(tre[num].l==l&&tre[num].r==r)
        {
            return tre[num].sum;
        }
        int mid=tre[num].mid();
```

```
51          if(l>mid) return query_sum(2*num+1,l,r);
52          else if(r<=mid) return query_sum(2*num,l,r);
53          else return query_sum(2*num,l,mid)+query_sum(2*num+1,
                mid+1,r);
54      }
55      void update(int num,int pos,int x)
56      {
57          if(tre[num].l==tre[num].r)
58          {
59              tre[num].sum=x;return;
60          }
61          int mid=tre[num].mid();
62          if(pos<=mid) update(2*num,pos,x);
63          else update(2*num+1,pos,x);
64          pushup(num);
65      }
66  }st;
67  ll query_sum(int x,int y)
68  {
69      ll res=0,fx=top[x],fy=top[y];
70      while(fx!=fy)
71      {
72          if(dep[fx]>=dep[fy]) res+=st.query_sum(1,dfn[fx],dfn[x
                ]),x=par[fx];
73          else res+=st.query_sum(1,dfn[fy],dfn[y]),y=par[fy];
74          fx=top[x],fy=top[y];
75      }
76      if(dfn[x]<dfn[y]) res+=st.query_sum(1,dfn[x],dfn[y]);
77      else res+=st.query_sum(1,dfn[y],dfn[x]);
78      return res;
79  }
```

## 0.11 莫队

区间不同种类元素个数

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int n,m,a[50005],vis[1000005],len;
4  struct node{int l,r,id;}q[200005];
5  int ans[200005],res=0;
```

```
6  bool cmp(node a,node b)
7  {
8          int al=a.l/len,bl=b.l/len;
9          if(al!=bl) return al<bl;
10         return a.r<b.r;
11 }
12 void add(int num)
13 {
14         vis[a[num]]++;
15         if(vis[a[num]]==1) res++;
16 }
17 void del(int num)
18 {
19         vis[a[num]]--;
20         if(vis[a[num]]==0) res--;
21 }
22 int main()
23 {
24         scanf("%d",&n);len=sqrt(n);
25         for(int i=1;i<=n;i++) scanf("%d",&a[i]);
26         scanf("%d",&m);
27         for(int i=1;i<=m;i++)
28         {
29                 scanf("%d%d",&q[i].l,&q[i].r);q[i].id=i;
30         }
31         sort(q+1,q+1+m,cmp);
32         int l=1,r=0;
33         for(int i=1;i<=m;i++)
34         {
35                 int id=q[i].id,ll=q[i].l,rr=q[i].r;
36                 while(r<rr) add(++r);
37                 while(r>rr) del(r--);
38                 while(l<ll) del(l++);
39                 while(l>ll) add(--l);
40                 ans[id]=res;
41         }
42         for(int i=1;i<=m;i++) printf("%d\n",ans[i]);
43 }
```