数据结构模板

1. Trie字典树
2. 并查集之食物链
3. 树状数组
4. 线段树合集(最大数，最大连续子段和，区间最大公约数，pushdown操作，扫描线)
5. 可持续化Trie，可持续化线段树
6. 平衡树
7. Splay

1.Trie字典树

```
#include<stdio.h>
#include<iostream>
#include<algorithm>
using namespace std;
const int N=100005;
int son[N][26],cnt[N],idx;
char str[N];
// son[节点序号][字母]=下一个儿子的序号
void inset_trie(char *str)
{
  int p=0;
  for(int i=0;str[i];i++)
  {
    int u=str[i]-'a';

    if(son[p][u]==0)
    son[p][u]=++idx;

    p=son[p][u];
  }
  cnt[p]++;
}
int query_trie(char *str)
{
  int p=0;
  for(int i=0;str[i];i++)
  {
    int u=str[i]-'a';

    if(son[p][u]==0)
    return 0;
    else
    {
      p=son[p][u];
    }
  }
  return cnt[p];
}
int main()
{
  int m;
  char op[2];
  scanf("%d",&m);
```

```
  while(m--)
  {
    scanf("%s%s",op,str);
    if(*op=='I')
    inset_trie(str);
    else
    printf("%d\n",query_trie(str));
  }
}
```

2.食物链

```
#include<iostream>
#include<algorithm>
using namespace std;
const int N=100005,M=N*31;
int son[M][2],idx;
int n;
void insert(int x)
{
  int p=0;
  for(int i=30;i>=0;i--)
  {
    int u=x>>i&1;
    if(son[p][u]==0)
    son[p][u]=++idx;

    p=son[p][u];
  }
}
int query(int x)
{
  int p=0,res=0;
  for(int i=30;i>=0;i--)
  {
    int u=x>>i&1;
    if(son[p][u^1]!=0)
    {
      p=son[p][u^1];
      res=res*2+1;
    }
    else
    {
      p=son[p][u];
      res=res*2;
    }
  }
  return res;
}
int main()
{
  scanf("%d",&n);
  int res=0;
  for(int i=1;i<=n;i++)
  {
    int x;
    scanf("%d",&x);
    insert(x);

    int t=query(x);
    res=max(res,t);
  }
  printf("%d",res);
}
```

3.树状数组

```
int d[N]
int lowbit(int x)
{
  return x&-x;
}
int query(int x)//查询某个数的前缀和
{
  int res=0;
  while(x)
  {
    res+=d[x];
    x-=lowbit(x);
  }
return res;
}
void add(int x,int c)//对a[x]加上c
{
  while(x<n)
  {
    d[x]+=c;
    x+=lowbit(x);
  }
}
```

4.线段树合集

最大数

```
#include<iostream>
#include<algorithm>
using namespace std;
const int N=200005;
int p;
struct Node
{
  int l,r;
  int max;
}tr[N*4];
void pushup(int u)
{
  tr[u].max=max(tr[u<<1].max,tr[u<<1|1].max);
}
void build(int u,int l,int r)
{
  if(l==r)
  tr[u]={l,r};
  else
  {
    tr[u]={l,r};
    int mid=l+r>>1;

    build(u<<1,l,mid);
    build(u<<1|1,mid+1,r);
  }
}
int query(int u,int l,int r)
{
  if(tr[u].l>=l&&tr[u].r<=r)
  return tr[u].max;
  else
  {
    int sum=0;
    int mid=tr[u].l+tr[u].r>>1;
```

```
    if(l<=mid) sum=query(u<<1,l,r);
    if(r>mid) sum=max(sum,query(u<<1|1,l,r));

    return sum;
  }
}
void modify(int u,int x,int d)
{
  //printf("u=%d \n",u);
  if(tr[u].l==x&&tr[u].r==x)
  tr[u].max=d;
  else
  {
    int mid=tr[u].l+tr[u].r>>1;

    if(x<=mid) modify(u<<1,x,d);
    else modify(u<<1|1,x,d);

    pushup(u);
  }
}
int main()
{
  int m;
  scanf("%d%d",&m,&p);
  int a=0;
  int n=0;
  build(1,1,m);
  while(m--)
  {
    char op[2];
    int t;
    scanf("%s%d",op,&t);
    if(*op=='Q')
    {
      a=query(1,n-t+1,n);
      printf("%d\n",a);
    }
    else
    {
      n++;
      modify(1,n,(t+a)%p);
    }
  }
}
```

最大连续子段和

> 给定一个长度为 N 的数列 A，以及 M 条指令，每条指令可能是以下两种之一：
>
> 1. `C l r d`，表示把 $A[l], A[l+1], \ldots, A[r]$ 都加上 d。
> 2. `Q l r`，表示询问数列中第 $l \sim r$ 个数的和。

```
#include<iostream>
#include<algorithm>
#include<string.h>
using namespace std;
const int N=500005;
int n,m;
int w[N];
struct Node
{
  int l,r;
```

```
  int sum,lmax,rmax,tmax;//总和，最大前缀和，最大后缀和，最大连续字段和
}tr[N*4];
void pushup(Node &u,Node &l,Node &r)
{
  u.sum=l.sum+r.sum;
  u.lmax=max(l.lmax,l.sum+r.lmax);
  u.rmax=max(r.rmax,r.sum+l.rmax);
  u.tmax=max(max(l.tmax,r.tmax),l.rmax+r.lmax);
}
void pushup(int u)
{
  pushup(tr[u],tr[u<<1],tr[u<<1|1]);
}
void build(int u,int l,int r)
{
  if(l==r) tr[u]={l,l,w[l],w[l],w[l],w[l]};
  else
  {
    tr[u].l=l;
    tr[u].r=r;
    int mid=l+r>>1;
    build(u<<1,l,mid);
    build(u<<1|1,mid+1,r);

    pushup(u);
  }
}
void modify(int u,int x,int v)
{
  if(tr[u].l==x&&tr[u].r==x)
  tr[u]={x,x,v,v,v,v};
  else
  {
    int mid=tr[u].l+tr[u].r>>1;
    if(x<=mid) modify(u<<1,x,v);
    else modify(u<<1|1,x,v);

    pushup(u);
   }
}
Node query(int u,int l,int r)
{
  if(tr[u].l>=l&&tr[u].r<=r)//树被区间覆盖
  return tr[u];
  else
  {
    int mid=tr[u].l+tr[u].r>>1;
    if(r<=mid) return query(u<<1,l,r);
    else if(l>mid) return query(u<<1|1,l,r);
    else
    {
      auto left=query(u<<1,l,r);
      auto right=query(u<<1|1,l,r);
      Node res;
      pushup(res,left,right);
      return res;
    }
  }
}
int main()
{
  scanf("%d%d",&n,&m);
  for(int i=1;i<=n;i++)
  scanf("%d",&w[i]);

  build(1,1,n);
  while(m--)
  {
    int k,x,y;
    scanf("%d%d%d",&k,&x,&y);
    if(k==1)
    {
      if(x>y)
      swap(x,y);
```

```
      printf("%d\n",query(1,x,y).tmax);
    }
    else
    {
      modify(1,x,y);
    }
  }
  return 0;
}
```

区间最大公约数

```
#include<iostream>
#include<algorithm>
#include<string.h>
#include<cmath>
using namespace std;
typedef long long ll;
const int N=500005;
ll w[N];
struct Node
{
  int l,r;
  ll sum,d;
}tr[4*N];
ll gcd(ll a,ll b)
{
  return b?gcd(b,a%b):a;
}
void pushup(Node &u,Node &l,Node &r)
{
  u.sum=l.sum+r.sum;
  u.d=gcd(l.d,r.d);
}
void pushup(int u)
{
  pushup(tr[u],tr[u<<1],tr[u<<1|1]);
}
void build(int u,int l,int r)
{
  if(l==r)
  {
    ll b=w[l]-w[l-1];
    tr[u]={l,r,b,b};
  }
  else
  {
    int mid=l+r>>1;
    tr[u]={l,r};
    build(u<<1,l,mid);
    build(u<<1|1,mid+1,r);
    pushup(u);
  }

}
void modify(int u,int x,ll c)
{
  if(tr[u].l==x&&tr[u].r==x)
  {
    tr[u].sum+=c;
    tr[u].d+=c;
  }
  else
  {
    int mid=tr[u].l+tr[u].r>>1;

    if(x<=mid) modify(u<<1,x,c);
```

```
    else modify(u<<1|1,x,c);
    pushup(u);
  }
}
Node query(int u,int l,int r)
{
  if(tr[u].l>=l&&tr[u].r<=r)
  return tr[u];

  int mid=tr[u].l+tr[u].r>>1;
  if(r<=mid) return query(u<<1,l,r);
  else if(l>mid) return query(u<<1|1,l,r);
  else
  {
    Node left=query(u<<1,l,r);
    Node right=query(u<<1|1,l,r);
    Node res;
    pushup(res,left,right);

    return res;
  }

}
int main()
{
  int n,m;
  scanf("%d%d",&n,&m);

  for(int i=1;i<=n;i++)
  scanf("%lld",&w[i]);

  build(1,1,n);
  char op[2];
  int l,r;
  while(m--)
  {
    scanf("%s%d%d",op,&l,&r);
    if(*op=='C')
    {
      ll d;
      scanf("%lld",&d);
      modify(1,l,d);
      if(r+1<=n) modify(1,r+1,-d);
    }
    else
    {
      printf("%lld\n",abs(gcd(query(1,1,l).sum,query(1,l+1,r).d)));
    }
  }
}
```

进行pushdown操作的

243. 一个简单的整数问题2

题目 | 提交记录 | 讨论 | 题解

给定一个长度为 N 的数列 A，以及 M 条指令，每条指令可能是以下两种之一：

1. `C l r d`，表示把 $A[l], A[l+1], \ldots, A[r]$ 都加上 d。
2. `Q l r`，表示询问数列中第 $l \sim r$ 个数的和。

对于每个询问，输出一个整数表示答案。

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<cstdio>
using namespace std;
typedef long long ll;
const int N=100005;
int n,m;
int w[N];
struct Node
{
  int l,r;
  ll sum,add;
}tr[4*N];
void pushup(int u)
{
  tr[u].sum=tr[u<<1].sum+tr[u<<1|1].sum;
}
void pushdown(int u)
{
  auto &root=tr[u],&left=tr[u<<1],&right=tr[u<<1|1];
  if(root.add)
  {
    left.add+=root.add , left.sum+=(ll)(left.r-left.l+1)*root.add;
    right.add+=root.add , right.sum+=(ll)(right.r-right.l+1)*root.add;
    root.add=0;
  }
}
void build(int u,int l,int r)
{
  if(l==r)
  {
    tr[u]={l,r,w[l],0};
  }
  else
  {
    int mid=l+r>>1;
    tr[u]={l,r};
    build(u<<1,l,mid);
    build(u<<1|1,mid+1,r);
    pushup(u);
  }
}
ll query(int u,int l,int r)
{
  if(tr[u].l>=l&&tr[u].r<=r)
  return tr[u].sum;
  else
  {
    pushdown(u);
    ll sum=0;
```

```
    int mid=tr[u].l+tr[u].r>>1;
    if(l<=mid) sum=query(u<<1,l,r);
    if(r>mid) sum+=query(u<<1|1,l,r);
    return sum;
  }
}
void modify(int u,int l,int r,int d)
{
  if(tr[u].l>=l&&tr[u].r<=r)
  {
    tr[u].sum+=(ll)(tr[u].r-tr[u].l+1)*d;
    tr[u].add+=d;
  }
  else
  {
    pushdown(u);

    int mid=tr[u].l+tr[u].r>>1;
  //  printf("u=%d\n",u);
  //  printf("%d %d %d\n",tr[u].l,tr[u].r,mid);
    //printf("l=%d r=%d\n\n",l,r);
    if(l<=mid) modify(u<<1,l,r,d);
    if(r>mid) modify(u<<1|1,l,r,d);

    pushup(u);
  }
}
int main()
{
  scanf("%d%d",&n,&m);
  for(int i=1;i<=n;i++)
  scanf("%d",&w[i]);

  build(1,1,n);
  int l,r,d;
  char op[2];
  while(m--)
  {
    scanf("%s%d%d",op,&l,&r);
    if(*op=='C')
    {
      scanf("%d",&d);
      modify(1,l,r,d);
    }
    else
    {
      printf("%lld\n",query(1,l,r));
    }
  }
  return 0;
}
```

扫描线

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<cstdio>
#include<vector>
using namespace std;
const int N=100005;
int n;
struct segment//储存每一个线段的信息
{
  double x,y1,y2;
  int k;//区分它是前面线段还是后面线段
  bool operator < (const segment&t)const
  {
```

```
    return x<t.x;
  }
 }seg[N*2];
struct Node
{
  int l,r;
  int cnt;//记录这段区间出现的次数
  double len;//记录这段区间的长度
 }tr[N*8];
vector<double>ys;//用于离散化纵坐标
int find(double y)
{
  return lower_bound(ys.begin(),ys.end(),y)-ys.begin();
}
void pushup(int u)
{
  if(tr[u].cnt) tr[u].len=ys[tr[u].r+1]-ys[tr[u].l];
  else if(tr[u].l!=tr[u].r)
  {
    tr[u].len=tr[u<<1].len+tr[u<<1|1].len;
  }
  else
  tr[u].len=0;//可能cnt>0，后面又变为=0，这时候len就要更新成=0
 }
void build(int u,int l,int r)
{
  tr[u]={l,r,0,0};
  if(l!=r)
  {
    int mid=l+r>>1;
    build(u<<1,l,mid);
    build(u<<1|1,mid+1,r);

    //反正最初你pushup也是0，所以就不需要加这句话了
  }
}
void modify(int u,int l,int r,int d)
{
  if(tr[u].l>=l&&tr[u].r<=r)
  {
    tr[u].cnt+=d;
    pushup(u);
  }
  else
  {
    int mid=tr[u].l+tr[u].r>>1;
    if(l<=mid) modify(u<<1,l,r,d);
    if(r>mid) modify(u<<1|1,l,r,d);

    pushup(u);
  }
 }
int main()
{
  int T=1;
  while(scanf("%d",&n),n)
  {
    ys.clear();
    int j=0;//代表有几个x下标
    for(int i=0;i<n;i++)
    {
      double x1,y1,x2,y2;
      scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
      seg[j++]={x1,y1,y2,1};//前面x的线段
      seg[j++]={x2,y1,y2,-1};//后面x的线段
      ys.push_back(y1),ys.push_back(y2);//将y轴纵坐标出现的点存进ys，进行离散化
    }
    sort(seg,seg+j);//按x的先后进行排序

    sort(ys.begin(),ys.end());//按y的大小进行排序
    ys.erase(unique(ys.begin(),ys.end()),ys.end());

    build(1,0,ys.size()-2);//从第0个区间到第几个y-2区间（第几个y-1即最后一个y的下标）
```

```
    double res=0;
    for(int i=0;i<2*n;i++)
    {
      //tr[1].len为此时有效线段长度
      if(i>0) res+=tr[1].len*(seg[i].x-seg[i-1].x);

      modify(1,find(seg[i].y1),find(seg[i].y2)-1,seg[i].k);
    }

    printf("Test case #%d\n",T++);
    printf("Total explored area: %.2lf\n\n",res);
  }
 }
```

5. 可持续化Trie，可持续化线段树

256. 最大异或和

题目　提交记录　讨论　题解　视频讲解

给定一个非负整数序列 a，初始长度为 N。

有 M 个操作，有以下两种操作类型：

1. `A x`：添加操作，表示在序列末尾添加一个数 x，序列的长度 N 增大 1。
2. `Q l r x`：询问操作，你需要找到一个位置 p，满足 $l \leq p \leq r$，使得：$a[p] \ xor \ a[p+1] \ xor \ \ldots \ xor \ a[N] \ xor \ x$ 最大，输出这个最大值。

```
#include<iostream>
#include<algorithm>
using namespace std;
const int N=600005,M=N*25;
int s[N];
int tr[M][2],idx;
int root[N];
int max_id[M];
int n,m;
void insert(int i,int k,int q,int p)
{
  if(k<0)
  {
    max_id[p]=i;
    return ;
  }
  int u=s[i]>>k&1;
  if(q)
  tr[p][u^1]=tr[q][u^1];
  tr[p][u]=++idx;
  insert(i,k-1,tr[q][u],tr[p][u]);

  max_id[p]=max(max_id[tr[p][1]],max_id[tr[p][0]]);
}
int query(int r,int c,int l)
{
  int p=root[r];
  for(int i=23;i>=0;i--)
  {
    int u=c>>i&1;
    if(max_id[tr[p][u^1]]>=l) p=tr[p][u^1];
    else p=tr[p][u];
```

```
  }

  return s[max_id[p]]^c;
}
int main()
{
  scanf("%d%d",&n,&m);
  root[0]=++idx;
  max_id[0]=-1;
  insert(0,23,0,root[0]);

  for(int i=1;i<=n;i++)
  {
    int x;
    scanf("%d",&x);
    s[i]=s[i-1]^x;
    root[i]=++idx;
    insert(i,23,root[i-1],root[i]);
  }
  int l,r,x;
  char op[2];
  while(m--)
  {
    scanf("%s",op);
    if(*op=='A')
    {
      scanf("%d",&x);
      n++;
      s[n]=s[n-1]^x;
      root[n]=++idx;
      insert(n,23,root[n-1],root[n]);
    }
    else
    {
      scanf("%d%d%d",&l,&r,&x);
      printf("%d\n",query(r-1,s[n]^x,l-1));
    }
  }
}
```

255. 第K小数

题目 | 提交记录 | 讨论 | 题解 | 视频讲解

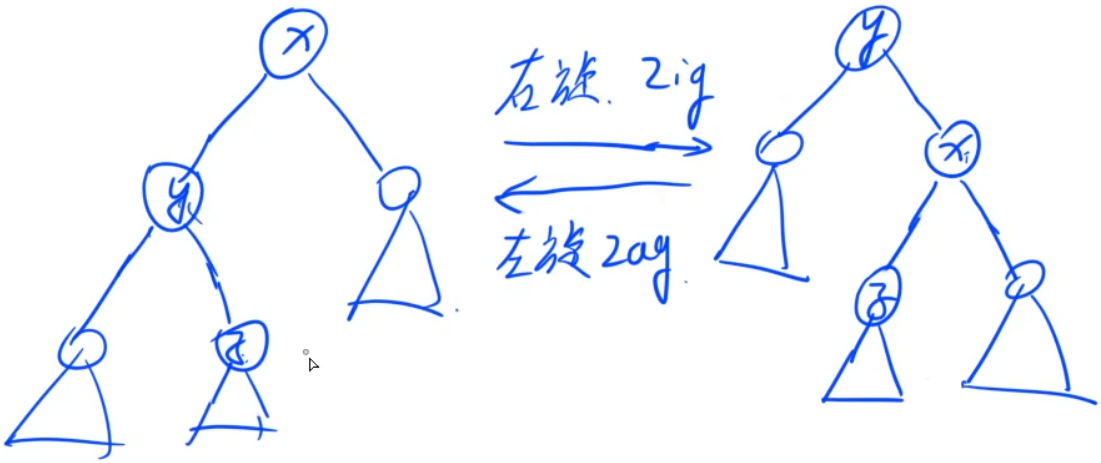
给定长度为 N 的整数序列 A，下标为 $1 \sim N$。

现在要执行 M 次操作，其中第 i 次操作为给出三个整数 l_i, r_i, k_i，求 $A[l_i], A[l_i+1], \ldots, A[r_i]$ (即 A 的下标区间 $[l_i, r_i]$)中第 k_i 小的数是多少。

```
#include<iostream>
#include<algorithm>
#include<string.h>
#include<vector>
using namespace std;
#define debug(x) cout<<"[debug]"#x":"<<x<<endl;
const int N=100005;
int n,m;
int a[N];
int root[N],idx;
vector<int> num;
```

```
struct Node
{
    int l,r;
    int cnt;
}tr[4*N+17*N];
int find(int x)
{
    return lower_bound(num.begin(),num.end(),x)-num.begin();
}
int build(int l,int r)
{
    int p=++idx;
    int mid=l+r>>1;
    if(l==r)
    {
        return p;
    }
    tr[p].l=build(l,mid);
    tr[p].r=build(mid+1,r);
    return p;
}
int insert(int p,int l,int r,int x)
{
    int q=++idx;
    tr[q]=tr[p];
    if(l==r)
    {
        tr[q].cnt++;
        return q;
    }

    int mid=l+r>>1;
    if(x<=mid) tr[q].l=insert(tr[p].l,l,mid,x);
    else tr[q].r=insert(tr[p].r,mid+1,r,x);

    tr[q].cnt=tr[tr[q].l].cnt+tr[tr[q].r].cnt;
    return q;
}
int query(int p,int q,int l,int r,int k)
{
    int cnt=tr[tr[q].l].cnt-tr[tr[p].l].cnt;
    if(l==r) return l;
    if(k<=cnt) return query(tr[p].l,tr[q].l,l,l+r>>1,k);
    else return query(tr[p].r,tr[q].r,(l+r>>1)+1,r,k-cnt);
}
int main()
{
    scanf("%d%d",&n,&m);

    for(int i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
        num.push_back(a[i]);
    }
    sort(num.begin(),num.end());
    num.erase(unique(num.begin(),num.end()),num.end());
    root[0]=build(0,num.size()-1);
    for(int i=1;i<=n;i++)
    {
      //debug(i)
        root[i]=insert(root[i-1],0,num.size()-1,find(a[i]));
    }

    while(m--)
    {
        int l,r,k;
        scanf("%d%d%d",&l,&r,&k);
        printf("%d\n",num[query(root[l-1],root[r],0,num.size()-1,k)]);
    }
}
```

6. 平衡树




```
#include<iostream>
#include<algorithm>
#include<string.h>
using namespace std;
#define debug(x) cout<<"[debug]"#x<<"="<<x<<endl
const int N=100005,INF=10000005;
struct Node
{
  int l,r;
  int key,val;
  int cnt,size;
}tr[N];
int root,idx;
void pushup(int u)
{
  tr[u].size=tr[tr[u].l].size+tr[tr[u].r].size+tr[u].cnt;
}
int get_node(int key)
{
  tr[++idx].key=key;
  tr[idx].cnt=tr[idx].size=1;
  tr[idx].val=rand();
  return idx;
}
void build()
{
  get_node(-INF);get_node(INF);
  root=1,tr[1].r=2;
  pushup(root);

}
void zig(int &p)//右旋(保证仍是当前位置，只是结点编号改变了而已)
{
  int q=tr[p].l;
  tr[p].l=tr[q].r,tr[q].r=p,p=q;
  pushup(tr[p].r);pushup(p);

}
void zag(int &p)//左旋
{
  int q=tr[p].r;
  tr[p].r=tr[q].l,tr[q].l=p,p=q;
  pushup(tr[p].l);pushup(p);
 }
void insert(int &p,int key)
{
```

```
  if(!p) p=get_node(key);
  else if(tr[p].key==key) tr[p].cnt++;
  else if(key<tr[p].key)
  {
    insert(tr[p].l,key);
    if(tr[tr[p].l].val>tr[p].val)
    zig(p);
  }
  else
  {
    insert(tr[p].r,key);
    if(tr[tr[p].r].val>tr[p].val)
    zag(p);
  }
  pushup(p);
}
void remove(int &p,int key)
{
  if(!p) return ;
  else if(key==tr[p].key)
  {
    if(tr[p].cnt>1) tr[p].cnt--;
    else if(tr[p].r||tr[p].l)
    {
      if(!tr[p].r||tr[tr[p].l].val>tr[tr[p].r].val)
      {
        zig(p);
        remove(tr[p].r,key);
      }
      else
      {
        zag(p);
        remove(tr[p].l,key);
      }
    }
    else
    p=0;//因为是调用的地址。所以p=0后其他的指针l和r就指不到它了
  }
  else if(key<tr[p].key)
  remove(tr[p].l,key);
  else if(key>tr[p].key)
  remove(tr[p].r,key);

  pushup(p);
}
int get_rank_by_num(int p,int key)
{
  if(!p)
  return 0;

  if(tr[p].key==key)
  return tr[tr[p].l].size+1;
  else if(key<tr[p].key)
  return get_rank_by_num(tr[p].l,key);
  else
  return tr[tr[p].l].size+tr[p].cnt+get_rank_by_num(tr[p].r,key);
 }
int get_num_by_rank(int p,int rank)
{
  if (!p) return INF;     // 本题中不会发生此情况
  if(rank<=tr[tr[p].l].size) return get_num_by_rank(tr[p].l,rank);
  else if(rank>tr[tr[p].l].size+tr[p].cnt) return get_num_by_rank(tr[p].r,rank-tr[tr[p].l].size-tr[p].cnt);

  return tr[p].key;
}
int get_front(int p,int key)
{
  if (!p) return -INF;
  if(tr[p].key>=key) return get_front(tr[p].l,key);
  else return max(tr[p].key,get_front(tr[p].r,key));//不用看左子树了，因为当前结点<key。而左子树的所有点都会小于当前结点的值
}
int get_next(int p,int key)
{
  if (!p) return INF;
```

```
  if(tr[p].key<=key) return get_next(tr[p].r,key);
  else return min(tr[p].key,get_next(tr[p].l,key));
 }
int main()
{
  build();
  int n;
  scanf("%d",&n);
  while(n--)
  {
    int opt,x;
    scanf("%d%d",&opt,&x);
    if(opt==1) insert(root,x);
    else if(opt==2) remove(root,x);
    else if(opt==3) printf("%d\n",get_rank_by_num(root,x)-1);
    else if(opt==4) printf("%d\n",get_num_by_rank(root,x+1));
    else if(opt==5) printf("%d\n",get_front(root,x));
    else if(opt==6) printf("%d\n",get_next(root,x));
  }
  return 0;
}
```

7. splay

```
#include<iostream>
#include<algorithm>
#include<string.h>
using namespace std;
#define debug(x) cout<<"[debug]"#x<<"="<<x<<endl
const int N=100005;
int n,m;
int idx,root;
struct Node
{
    int s[2],p;
    int v;
    int size;
    int flag;//代表是否需要翻转

    void init(int _p,int _v)
    {
        v=_v;
        p=_p;
        size=1;
    }
}tr[N];
void pushup(int u)
{
    tr[u].size=tr[tr[u].s[0]].size+tr[tr[u].s[1]].size+1;
}
void pushdown(int u)
{
    if(tr[u].flag)
    {
        swap(tr[u].s[0],tr[u].s[1]);
        tr[tr[u].s[0]].flag^=1;
        tr[tr[u].s[1]].flag^=1;
        tr[u].flag=0;
    }
}
void output(int u)
{
    pushdown(u);
    if(tr[u].s[0]) output(tr[u].s[0]);
    if(tr[u].v>=1&&tr[u].v<=n) printf("%d ",tr[u].v);
    if(tr[u].s[1]) output(tr[u].s[1]);
```

```
}
void rotate(int x)
{
    int y=tr[x].p,z=tr[y].p;
    int k=(tr[y].s[1]==x);//k为0表示右旋，k为1表示左旋
    tr[z].s[tr[z].s[1]==y]=x,tr[x].p=z;
    tr[tr[x].s[k^1]].p=y,tr[y].s[k]=tr[x].s[k^1];//将B的父节点连到y，并且将y的左孩子变为B
    tr[x].s[k^1]=y,tr[y].p=x;

    pushup(y);
    pushup(x);
}
void splay(int x,int k)
{
    while(tr[x].p!=k)
    {
        int y=tr[x].p,z=tr[y].p;
        if(z!=k)
        {
            if(tr[y].p==z^tr[x].p==y) rotate(x);//折线关系
            else rotate(y);
        }
        rotate(x);
    }
    if(!k) root=x;
}
void insert(int v)
{
    int u=root,p=0;
    while(u)
    {
        p=u;
        u=tr[u].s[v>tr[u].v];
    }
    u=++idx;
    if(p) tr[p].s[v>tr[p].v]=u;
    tr[u].init(p,v);
    splay(u,0);
}
int get_rank(int k)
{
    int u=root;
    while(1)
    {
        pushdown(u);
        if(tr[tr[u].s[0]].size>=k) u=tr[u].s[0];
        else if(tr[tr[u].s[0]].size+1==k) return u;
        else k-=tr[tr[u].s[0]].size+1,u=tr[u].s[1];
    }
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=0;i<=n+1;i++)
    insert(i);

    while(m--)
    {
        int l,r;
        scanf("%d%d",&l,&r);
        l=get_rank(l);
        r=get_rank(r+2);

        splay(l,0);
        splay(r,l);

        tr[tr[r].s[0]].flag^=1;
    }
    output(root);
}
```