小小青蛙听风就是雨

HiedanoAkyuu、Oneman233、KR12138

2019 年 10 月 18 日

目录

1 字符串

1.1 KMP

1.2 EX-KMP

```cpp
#include <bits/stdc++.h>
using namespace std;

/*
    ex[i]: s1[i..l1-1]与s2的最大公共前缀长度
    exnext[i]: s2[i..l2-1]与s2的最大公共前缀长度
    get_exnext(s2) 求exnext[]
    exkmp(s1,s2) 求ex[]
*/

const int N=50005;
char s1[N],s2[N];
int ex[N],exnext[N];

void get_exnext(char s[N])
{
    int n=strlen(s),p=1,j,i;
    exnext[0]=n;
    for(i=0;i<n-1&&s[i]==s[i+1];i++);
    exnext[1]=i;
    for(i=2;i<n;i++)
        if(exnext[i-p]+i<p+exnext[p])
            exnext[i]=exnext[i-p];
        else
        {
            j=exnext[p]+p-i;
            if(j<0) j=0;
            while(i+j<n&&s[j]==s[i+j]) j++;
            exnext[i]=j;
            p=i;
        }
}

void exkmp(char s1[N],char s2[N])
{
    int l1=strlen(s1),l2=strlen(s2),p=0,i,j;
    get_exnext(s2);
    for(i=0;i<l1&&i<l2&&s1[i]==s2[i];i++);
    ex[0]=i;
    for(int i=1;i<l1;i++)
    {
        if(exnext[i-p]+i<p+ex[p])
            ex[i]=exnext[i-p];
        else
        {
            j=ex[p]+p-i;
            if(j<0) j=0;
            while(i+j<l1&&s1[i+j]==s2[j]) j++;
            ex[i]=j;
            p=i;
        }
    }
}
```

1.3 Manacher

1.4 串的最小表示

1.5 后缀数组

1.5.1 倍增 SA

1.5.2 DC3

1.6 回文自动机

1.7 AC 自动机

1.7.1 多模匹配

```cpp
#include <bits/stdc++.h>
using namespace std;

/*
    trie静态开点+trie图优化
*/

int sz,hd=1,nxt[1000005][26],fail[1000005],id
    [1000005],n;
char s[1000005];

void trie_clean()
{
    sz=1;
    memset(nxt,0,sizeof(nxt));
    memset(fail,0,sizeof(fail));
    memset(id,0,sizeof(id));
}

void trie_insert(int head,char s[],int len,int idx)
{
    int p=head;
    for(int i=0;i<len;i++)
    {
        int c=s[i]-'a';
        if(!nxt[p][c]) nxt[p][c]=++sz;
        p=nxt[p][c];
    }
    id[p]+=idx;
}

void acatm_build(int head)
{
    int p,tp;
    queue<int> q;
    q.push(head);
    fail[head]=0;
    while(!q.empty())
    {
        p=q.front();
        q.pop();
        for(int i=0;i<26;i++)
            if(nxt[p][i])
            {
                fail[nxt[p][i]]=p==head?head:nxt[fail[p
                    ]][i];
                q.push(nxt[p][i]);
            }
            else
                nxt[p][i]=p==head?head:nxt[fail[p]][i];
    }
```

```
}

int acatm_match(int head,char s[],int len)
{
    int p=head,ret=0;
    for(int i=0;i<len;i++)
    {
        int c=(int)s[i]-'a';
        p=nxt[p][c];
        for(int tp=p;tp;tp=fail[tp])
            if(id[tp]) ret++;
    }
    return ret;
}
```

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

```
#include<bits/stdc++.h>
using namespace std;

#define db double
const db EPS=1e-9;
inline int sign(db a){return a<-EPS?-1:a>EPS;}
inline int cmp(db a,db b){return sign(a-b);}
struct P
{
    db x,y;
    P(){}
    P(db x,db y):x(x),y(y){}
    P operator+(P p){return {x+p.x,y+p.y};}
    P operator-(P p){return {x-p.x,y-p.y};}
    P operator*(db d){return {x*d,y*d};}
    P operator/(db d){return {x/d,y/d};}
    bool operator<(P p) const
    {
        int c=cmp(x,p.x);
        if(c) return c==-1;
        return cmp(y,p.y)==-1;
    }
    bool operator==(P o) const
    {
        return cmp(x,o.x)==0&&cmp(y,o.y)==0;
    }
    db distTo(P p){return (*this-p).abs();}
    db alpha(){return atan2(y,x);}
    void read(){scanf("%lf%lf",&x,&y);}
    void write(){printf("(%.10f,%.10f)\n",x,y);}
    db abs(){return sqrt(abs2());}
    db abs2(){return x*x+y*y;}
    P rot90(){return P(-y,x);}
    P unit(){return *this/abs();}
    int quad() const {return sign(y)==1||(sign(y)==0&&
        sign(x)>=0);}
    db dot(P p){return x*p.x+y*p.y;}
    db det(P p){return x*p.y-y*p.x;}
    P rot(db an){return {x*cos(an)-y*sin(an),x*sin(an)
        +y*cos(an)};}
};

//For segment
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.
    x-p1.x)*(p2.y-p1.y))
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))

bool chkLL(P p1,P p2,P q1,P q2) //0:parallel
{
    db a1=cross(q1,q2,p1),a2=-cross(q1,q2,p2);
    return sign(a1+a2)!=0;
}

P isLL(P p1,P p2,P q1,P q2) //crossover point if
    chkLL()
{
    db a1=cross(q1,q2,p1),a2=-cross(q1,q2,p2);
    return (p1*a2+p2*a1)/(a1+a2);
}

bool intersect(db l1,db r1,db l2,db r2)
{
    if(l1>r1) swap(l1,r1);if(l2>r2) swap(l2,r2);
    return !(cmp(r1,l2)==-1||cmp(r2,l1)==-1);
}

bool isSS(P p1,P p2,P q1,P q2)
{
    return intersect(p1.x,p2.x,q1.x,q2.x)&&intersect(
        p1.y,p2.y,q1.y,q2.y)&&
    crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<=0&&crossOp(q1
        ,q2,p1)*crossOp(q1,q2,p2)<=0;
}

bool isSS_strict(P p1,P p2,P q1,P q2)
{
    return crossOp(p1,p2,q1)*crossOp(p1,p2,q2)<0
    &&crossOp(q1,q2,p1)*crossOp(q1,q2,p2)<0;
}

bool isMiddle(db a,db m,db b)
{
    return sign(a-m)==0||sign(b-m)==0||(a<m!=b<m);
}

bool isMiddle(P a,P m,P b)
{
    return isMiddle(a.x,m.x,b.x)&&isMiddle(a.y,m.y,b.y
        );
}

bool onSeg(P p1,P p2,P q)
{
    return crossOp(p1,p2,q)==0&&isMiddle(p1,q,p2);
}

bool onSeg_strict(P p1,P p2,P q)
{
    return crossOp(p1,p2,q)==0&&sign((q-p1).dot(p1-p2)
        )*sign((q-p2).dot(p1-p2))<0;
}

P proj(P p1,P p2,P q)
{
    P dir=p2-p1;
    return p1+dir*(dir.dot(q-p1)/dir.abs2());
```

```
}

P reflect(P p1,P p2,P q)
{
    return proj(p1,p2,q)*2-q;
}

db nearest(P p1,P p2,P q)
{
    P h=proj(p1,p2,q);
    if(isMiddle(p1,h,p2))
        return q.distTo(h);
    return min(p1.distTo(q),p2.distTo(q));
}

db disSS(P p1,P p2,P q1,P q2) //dist of 2 segments
{
    if(isSS(p1,p2,q1,q2)) return 0;
    return min(min(nearest(p1,p2,q1),nearest(p1,p2,q2)
        ),min(nearest(q1,q2,p1),nearest(q1,q2,p2)));
}

db rad(P p1,P p2)
{
    return atan2l(p1.det(p2),p1.dot(p2));
}

db area(vector<P> ps)
{
    db ret=0;
    for(int i=0;i<ps.size();i++)
        ret+=ps[i].det(ps[(i+1)%ps.size()]);
    return ret/2;
}

int contain(vector<P> ps,P p) //2:inside,1:on_seg,0:
    outside
{
    int n=ps.size(),ret=0;
    for(int i=0;i<n;i++)
    {
        P u=ps[i],v=ps[(i+1)%n];
        if(onSeg(u,v,p)) return 1;
        if(cmp(u.y,v.y)<=0) swap(u,v);
        if(cmp(p.y,u.y)>0||cmp(p.y,v.y)<=0) continue;
        ret^=crossOp(p,u,v)>0;
    }
    return ret*2;
}

vector<P> convexHull(vector<P> ps)
{
    int n=ps.size();if(n<=1) return ps;
    sort(ps.begin(),ps.end());
    vector<P> qs(n*2);int k=0;
    for(int i=0;i<n;qs[k++]=ps[i++])
        while(k>1&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
            --k;
    for(int i=n-2,t=k;i>=0;qs[k++]=ps[i--])
        while(k>t&&crossOp(qs[k-2],qs[k-1],ps[i])<=0)
            --k;
    qs.resize(k-1);
    return qs;
}

db convexDiameter(vector<P> ps)
{
    int n=ps.size();if(n<=1) return 0;
    int is=0,js=0;
    for(int k=1;k<n;k++) is=ps[k]<ps[is]?k:is,js=ps[js
        ]<ps[k]?js:k;
    int i=is,j=js;
    db ret=ps[i].distTo(ps[j]);
    do{
        if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j])
            >=0) (++j)%=n;
        else (++i)%=n;
        ret=max(ret,ps[i].distTo(ps[j]));
    }while(i!=is||j!=js);
    return ret;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

/*
    s,t 超级源、超级汇
    cur[] 当前弧优化
    时间复杂度 O(n^2*m)
*/

const int MAXN=10005;
const ll inf=0x3f3f3f3f3f3f3f3fll;
int n,m,s,t,tot,dis[MAXN],cur[MAXN];

struct edge
{
    int to,cap,rev;
    edge(){}
    edge(int to,int cap,int rev):to(to),cap(cap),rev(rev){}
};
vector<edge> E[MAXN];

inline void add_edge(int x,int y,int f)
{
    E[x].emplace_back(y,f,E[y].size());
    E[y].emplace_back(x,0,E[x].size()-1);
}

int bfs()
{
    for(int i=1;i<=n;i++) dis[i]=0x3f3f3f3f;
    dis[s]=0;
    queue<int> q;
    q.push(s);
    while(!q.empty())
    {
        int now=q.front();q.pop();
        for(int i=0;i<E[now].size();i++)
        {
            edge &e=E[now][i];
            if(dis[e.to]>dis[now]+1&&e.cap)
            {
                dis[e.to]=dis[now]+1;
                if(e.to==t) return 1;
                q.push(e.to);
            }
        }
    }
    return 0;
}

ll dfs(int now,ll flow)
{
    if(now==t) return flow;
    ll rest=flow,k;
    for(int i=cur[now];i<E[now].size();i++)
    {
        edge &e=E[now][i];
        if(e.cap&&dis[e.to]==dis[now]+1)
        {
            cur[now]=i;
            k=dfs(e.to,min(rest,(long long)e.cap));
            e.cap-=k;
            E[e.to][e.rev].cap+=k;
            rest-=k;
        }
    }
    return flow-rest;
}

ll dinic()
{
    ll ret=0,delta;
    while(bfs())
    {
        for(int i=1;i<=n;i++) cur[i]=0;
        while(delta=dfs(s,inf)) ret+=delta;
    }
```

```
    return ret;
}
```

3.7.2 最小费用最大流-Dij+Dinic

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;

/*
    第一遍跑的spfa,然后是加上势函数的dij,玄学
    h[] 势函数
    cur[] 当前弧优化
    msmf 最大流时的最小费用
    s,t 超级源、超级汇
    时间复杂度 O(n^2*m)
*/

const int MAXN=2005;
const int inf=0x3f3f3f3f;
int msmf,s,t,cur[MAXN],dis[MAXN],vis[MAXN],h[MAXN];
struct edge
{
    int to,val,cap,rev;
    edge(){}
    edge(int to,int cap,int val,int rev):to(to),cap(
        cap),val(val),rev(rev){}
};
vector<edge> E[MAXN];

inline void add_edge(int x,int y,int f,int cost)
{
    E[x].emplace_back(y,f,cost,E[y].size());
    E[y].emplace_back(x,0,-cost,E[x].size()-1);
}

int dij()
{
    fill(dis,dis+t+1,inf);
    priority_queue<pii,vector<pii>,greater<pii>> q;
    q.emplace(0,s);dis[s]=0;
    while(!q.empty())
    {
        pii p=q.top();q.pop();
        int now=p.second;
        if(dis[now]<p.first) continue;
        for(int i=0;i<E[now].size();i++)
        {
            edge &e=E[now][i];
            if(e.cap>0&&dis[e.to]>p.first+e.val+h[now]-
                h[e.to])
            {
                dis[e.to]=p.first+e.val+h[now]-h[e.to];
                q.emplace(dis[e.to],e.to);
            }
        }
    }
    return dis[t]!=inf;
}

int dfs(int now,int flow)
{
    if(now==t) return flow;
    int rest=flow,k;
    vis[now]=1;
    for(int i=cur[now];i<E[now].size();i++)
    {
        edge &e=E[now][i];
        if(e.cap&&dis[now]+e.val+h[now]-h[e.to]==dis[e
            .to]&&!vis[e.to])
        {
            cur[now]=i;
            k=dfs(e.to,min(e.cap,rest));
            e.cap-=k;
            E[e.to][e.rev].cap+=k;
            rest-=k;
            msmf+=k*e.val;
        }
    }
    vis[now]=0;
    return flow-rest;
}

int dinic()
{
    int ret=0,delta;
    while(dij())
    {
        for(int i=s;i<=t;i++) cur[i]=0;
        while(delta=dfs(s,inf)) ret+=delta;
        for(int i=s;i<=t;i++) h[i]+=(dis[i]==inf)?0:
            dis[i];
    }
    return ret;
}
```

3.7.3 上下界流

3.8 欧拉路

3.9 Prufer 序列

4 数据结构

4.1 树状数组

4.2 线段树

4.2.1 多操作线段树

4.2.2 吉司机线段树

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

/*
    modify 将区间大于x的数变成x
    query 询问区间和
    单次复杂度 O(log^2(n))
*/

const ll INF=0xc0c0c0c0c0c0c0c0ll;
const int MAXN=200005;
ll seg[MAXN<<2],m1[MAXN<<2],m2[MAXN<<2],cnt[MAXN<<2],
    tag[MAXN<<2],a[MAXN];
int n,q;

void pushdown(int rt)
```

```
{
    if(!tag[rt]) return;
    ll y=m1[rt];
    if(y<m1[rt<<1])
    {
        tag[rt<<1]=1;
        seg[rt<<1]-=(m1[rt<<1]-y)*cnt[rt<<1];
        m1[rt<<1]=y;
    }
    if(y<m1[rt<<1|1])
    {
        tag[rt<<1|1]=1;
        seg[rt<<1|1]-=(m1[rt<<1|1]-y)*cnt[rt<<1|1];
        m1[rt<<1|1]=y;
    }
    tag[rt]=0;
}

void pushup(int rt)
{
    seg[rt]=seg[rt<<1]+seg[rt<<1|1];
    if(m1[rt<<1]==m1[rt<<1|1])
    {
        m1[rt]=m1[rt<<1];
        cnt[rt]=cnt[rt<<1]+cnt[rt<<1|1];
        m2[rt]=max(m2[rt<<1],m2[rt<<1|1]);
    }
    else if(m1[rt<<1]>m1[rt<<1|1])
    {
        m1[rt]=m1[rt<<1];
        cnt[rt]=cnt[rt<<1];
        m2[rt]=max(m2[rt<<1],m1[rt<<1|1]);
    }
    else
    {
        m1[rt]=m1[rt<<1|1];
        cnt[rt]=cnt[rt<<1|1];
        m2[rt]=max(m2[rt<<1|1],m1[rt<<1]);
    }
}

void build(int rt,int l,int r)
{
    tag[rt]=0;
    if(l==r)
    {
        seg[rt]=m1[rt]=a[l];
        cnt[rt]=1;
        m2[rt]=INF;
        return;
    }
    int m=l+r>>1;
    if(l<=m) build(rt<<1,l,m);
    if(m<r) build(rt<<1|1,m+1,r);
    pushup(rt);
}

void modify(int rt,int l,int r,int L,int R,ll y)
{
    if(y>=m1[rt]) return;
    if(L<=l&&r<=R&&y>m2[rt])
    {
        tag[rt]=1;
        seg[rt]-=(m1[rt]-y)*cnt[rt];
        m1[rt]=y;
        return;
    }
    pushdown(rt);
    int m=l+r>>1;
    if(L<=m) modify(rt<<1,l,m,L,R,y);
    if(m<R) modify(rt<<1|1,m+1,r,L,R,y);
    pushup(rt);
}

ll query(int rt,int l,int r,int L,int R)
{
    if(L<=l&&r<=R) return seg[rt];
    int m=l+r>>1;
    pushdown(rt);
    ll ret=0;
    if(L<=m) ret+=query(rt<<1,l,m,L,R);
    if(m<R) ret+=query(rt<<1|1,m+1,r,L,R);
    pushup(rt);
    return ret;
}
```

4.2.3 扫描线

4.3 RMQ

4.3.1 一维

4.3.2 两维

4.4 树链剖分

4.4.1 点剖分

4.4.2 边剖分

4.5 平衡树

4.5.1 Treap

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN=1e5+5;
const int inf=0x7fffffff;
int n,op,x;

/*
    树内初始化时有无穷大和无穷小两个结点
    _delete(root,x) 删除一个x
    _insert(root,x) 插入一个x
    getRank(root,x) 返回x的排名+1(包含了无穷小)
    getVal(root,x+1) 返回排名为x的数
    getPrev(x) x的前驱
    getNext(x) x的后继
*/

namespace Treap
{
    int tot,root;
    struct node
    {
        int cnt,val,dat,siz,lc,rc;
    }bst[MAXN];

    inline void pushup(int rt)
    {
        bst[rt].siz=bst[rt].cnt;
```

```
    if(bst[rt].lc) bst[rt].siz+=bst[bst[rt].lc].siz;
    if(bst[rt].rc) bst[rt].siz+=bst[bst[rt].rc].siz;
}

inline void zig(int &rt)
{
    int p=bst[rt].lc;
    bst[rt].lc=bst[p].rc;
    bst[p].rc=rt;
    rt=p;
    pushup(bst[rt].rc);pushup(rt);
}

inline void zag(int &rt)
{
    int p=bst[rt].rc;
    bst[rt].rc=bst[p].lc;
    bst[p].lc=rt;
    rt=p;
    pushup(bst[rt].lc);pushup(rt);
}

int new_node(int val)
{
    bst[++tot].val=val;
    bst[tot].dat=rand();
    bst[tot].siz=bst[tot].cnt=1;
    bst[tot].lc=bst[tot].rc=0;
    return tot;
}

void build()
{
    new_node(-inf);new_node(inf);
    root=1,bst[1].rc=2;
    pushup(1);
}

void _insert(int &rt,int val)
{
    if(rt==0)
    {
        rt=new_node(val);
        return;
    }
    if(bst[rt].val==val)
    {
        bst[rt].cnt++;
        pushup(rt);
        return;
    }
    if(val<bst[rt].val)
    {
        _insert(bst[rt].lc,val);
        if(bst[rt].dat<bst[bst[rt].lc].dat) zig(rt);
    }
    else
    {
        _insert(bst[rt].rc,val);
        if(bst[rt].dat<bst[bst[rt].rc].dat) zag(rt);
    }
    pushup(rt);
}

void _delete(int &rt,int val)
{
    if(rt==0) return;
    if(bst[rt].val==val)
    {
        if(bst[rt].cnt>1)
        {
            bst[rt].cnt--;
            pushup(rt);
            return;

        }
        if(bst[rt].rc||bst[rt].lc)
        {
            if(bst[rt].rc==0||bst[bst[rt].rc].dat<bst[bst[rt].lc].dat)
                zig(rt),_delete(bst[rt].rc,val);
            else
                zag(rt),_delete(bst[rt].lc,val);
            pushup(rt);
        }
        else rt=0;
        return;
    }
    if(val<bst[rt].val) _delete(bst[rt].lc,val);
    else _delete(bst[rt].rc,val);
    pushup(rt);
}

int getPrev(int val)
{
    int ret=1,rt=root;
    while(rt)
    {
        if(bst[rt].val==val)
        {
            if(bst[rt].lc)
            {
                rt=bst[rt].lc;
                while(bst[rt].rc) rt=bst[rt].rc;
                ret=rt;
            }
            break;
        }
        if(bst[rt].val<val&&bst[rt].val>bst[ret].val) ret=rt;
        if(val<bst[rt].val) rt=bst[rt].lc;
        else rt=bst[rt].rc;
    }
    return bst[ret].val;
}

int getNext(int val)
{
    int ret=2,rt=root;
    while(rt)
    {
        if(bst[rt].val==val)
        {
            if(bst[rt].rc)
            {
                rt=bst[rt].rc;
```

```
                while(bst[rt].lc) rt=bst[rt].lc;
                ret=rt;
            }
            break;
        }
        if(bst[rt].val>val&&bst[rt].val<bst[ret].val) ret=rt;
        if(val<bst[rt].val) rt=bst[rt].lc;
        else rt=bst[rt].rc;
    }
    return bst[ret].val;
}

int getRank(int rt,int val)
{
    if(rt==0) return 0;
    if(val==bst[rt].val) return bst[bst[rt].lc].siz+1;
    if(val<bst[rt].val) return getRank(bst[rt].lc,val);
    else return bst[bst[rt].lc].siz+bst[rt].cnt+getRank(bst[rt].rc,val);
}

int getVal(int rt,int k)
{
    if(rt==0) return inf;
    if(bst[bst[rt].lc].siz>=k) return getVal(bst[rt].lc,k);
    if(bst[bst[rt].lc].siz+bst[rt].cnt>=k) return bst[rt].val;
    return getVal(bst[rt].rc,k-bst[bst[rt].lc].siz-bst[rt].cnt);
}
}

int main()
{
    using namespace Treap;
    srand(time(0));
    build();
    scanf("%d",&n);
    while(n--)
    {
        scanf("%d%d",&op,&x);
        if(op==1) _insert(root,x);
        else if(op==2) _delete(root,x);
        else if(op==3) printf("%d\n",getRank(root,x)-1);
        else if(op==4) printf("%d\n",getVal(root,x+1));
        else if(op==5) printf("%d\n",getPrev(x));
        else if(op==6) printf("%d\n",getNext(x));
    }
    return 0;
}
```

4.5.2 Splay

4.6 动态树

4.7 主席树

4.8 树套树

4.8.1 线段树套 Treap

```
#include <bits/stdc++.h>
using namespace std;

/*
    空间 O(nlogn)
    单点修改,区间rank,前驱后继(不存在则为±2147483647) 单次 O(log^2(n))
    区间排名为k的值 单次 O(log^3(n))
*/

const int inf=2147483647;
const int MAXN=50005;
int root[MAXN<<2],n,m,a[MAXN];
struct Treap
{
    int tot;
    struct node
    {
        int lc,rc,dat,val,cnt,siz;
    }bst[MAXN*4*20];

    int newnode(int v)
    {
        bst[++tot].val=v;
        bst[tot].dat=rand();
        bst[tot].siz=bst[tot].cnt=1;
        bst[tot].lc=bst[tot].rc=0;
        return tot;
    }

    void zig(int &rt)
    {
        int p=bst[rt].lc;
        bst[rt].lc=bst[p].rc;
        bst[p].rc=rt;
        rt=p;
        pushup(bst[rt].rc);
        pushup(rt);
    }

    void zag(int &rt)
    {
        int p=bst[rt].rc;
        bst[rt].rc=bst[p].lc;
        bst[p].lc=rt;
        rt=p;
        pushup(bst[rt].lc);
        pushup(rt);
    }

    void pushup(int rt)
    {
        bst[rt].siz=bst[rt].cnt;
        if(bst[rt].lc) bst[rt].siz+=bst[bst[rt].lc].siz;
        if(bst[rt].rc) bst[rt].siz+=bst[bst[rt].rc].
```

```
        siz;
}

int build()
{
    int rt=newnode(-inf);
    bst[rt].rc=newnode(inf);
    pushup(rt);
    return rt;
}

void _delete(int &rt,int x)
{
    if(bst[rt].val==x)
    {
        if(bst[rt].cnt>1)
        {
            bst[rt].cnt--;
            pushup(rt);
            return;
        }
        if(bst[rt].lc||bst[rt].rc)
        {
            if(bst[rt].rc==0||bst[bst[rt].rc].dat<bst[bst[rt].lc].dat)
                zig(rt),_delete(bst[rt].rc,x);
            else
                zag(rt),_delete(bst[rt].lc,x);
            pushup(rt);
        }
        else rt=0;
        return;
    }
    if(x<bst[rt].val) _delete(bst[rt].lc,x);
    else _delete(bst[rt].rc,x);
    pushup(rt);
}

void _insert(int &rt,int x)
{
    if(rt==0)
    {
        rt=newnode(x);
        return;
    }
    if(bst[rt].val==x) bst[rt].cnt++;
    else if(x<bst[rt].val)
    {
        _insert(bst[rt].lc,x);
        if(bst[bst[rt].lc].dat>bst[rt].dat) zig(rt);
    }
    else
    {
        _insert(bst[rt].rc,x);
        if(bst[bst[rt].rc].dat>bst[rt].dat) zag(rt);
    }
    pushup(rt);
}

int get_rank(int rt,int x)
{
    if(!rt) return 1;
    if(bst[rt].val==x) return bst[bst[rt].lc].siz+1;
    if(x<bst[rt].val) return get_rank(bst[rt].lc,x);
    else return get_rank(bst[rt].rc,x)+bst[bst[rt].lc].siz+bst[rt].cnt;
}

int get_num(int rt,int x)
{
    if(!rt) return 0;
    if(bst[rt].val==x) return bst[bst[rt].lc].siz+bst[rt].cnt;
    if(x<bst[rt].val) return get_num(bst[rt].lc,x);
    else return get_num(bst[rt].rc,x)+bst[bst[rt].lc].siz+bst[rt].cnt;
}

int get_prev(int rt,int x)
{
    int ret=-inf;
    while(rt)
    {
        if(bst[rt].val==x)
        {
            if(bst[rt].lc)
            {
                rt=bst[rt].lc;
                while(bst[rt].rc) rt=bst[rt].rc;
                ret=bst[rt].val;
            }
            break;
        }
        if(bst[rt].val<x&&bst[rt].val>ret) ret=bst[rt].val;
        if(x<bst[rt].val) rt=bst[rt].lc;
        else rt=bst[rt].rc;
    }
    return ret;
}

int get_nxt(int rt,int x)
{
    int ret=inf;
    while(rt)
    {
        if(bst[rt].val==x)
        {
            if(bst[rt].rc)
            {
                rt=bst[rt].rc;
                while(bst[rt].lc) rt=bst[rt].lc;
                ret=bst[rt].val;
            }
            break;
        }
        if(bst[rt].val>x&&bst[rt].val<ret) ret=bst[rt].val;
        if(x<bst[rt].val) rt=bst[rt].lc;
        else rt=bst[rt].rc;
    }
    return ret;
}
}treap;

```

```
void build(int rt,int l,int r)
{
    root[rt]=treap.build();
    if(l==r) return;
    int m=l+r>>1;
    build(rt<<1,l,m);
    build(rt<<1|1,m+1,r);
}

void modify(int rt,int l,int r,int x,int v,int y)
{
    if(y==-1) treap._delete(root[rt],v);
    else treap._insert(root[rt],v);
    if(l==r) return;
    int m=l+r>>1;
    if(x<=m) modify(rt<<1,l,m,x,v,y);
    else modify(rt<<1|1,m+1,r,x,v,y);
}

int query(int rt,int l,int r,int op,int L,int R,int x
    )
{
    if(L<=l&&r<=R)
    {
        if(op==1) return treap.get_rank(root[rt],x)-2;
        if(op==2) return treap.get_num(root[rt],x)-1;
        if(op==4) return treap.get_prev(root[rt],x);
        if(op==5) return treap.get_nxt(root[rt],x);
    }
    int m=l+r>>1,ret;
    if(op==1||op==2)
    {
        ret=0;
        if(L<=m) ret+=query(rt<<1,l,m,op,L,R,x);
        if(m<R) ret+=query(rt<<1|1,m+1,r,op,L,R,x);
    }
    if(op==4)
    {
        ret=-inf;
        if(L<=m) ret=max(ret,query(rt<<1,l,m,op,L,R,x)
            );
        if(m<R) ret=max(ret,query(rt<<1|1,m+1,r,op,L,R
            ,x));
    }
    if(op==5)
    {
        ret=inf;
        if(L<=m) ret=min(ret,query(rt<<1,l,m,op,L,R,x)
            );
        if(m<R) ret=min(ret,query(rt<<1|1,m+1,r,op,L,R
            ,x));
    }
    return ret;
}

int main()
{
    srand(time(0));
    scanf("%d%d",&n,&m);
    build(1,1,n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",a+i);
        modify(1,1,n,i,a[i],1);
    }
    while(m--)
    {
        int op,l,r,k,pos;
        scanf("%d",&op);
        if(op==1)
        {
            scanf("%d%d%d",&l,&r,&k);
            printf("%d\n",query(1,1,n,op,l,r,k)+1);
        }
        else if(op==2)
        {
            scanf("%d%d%d",&l,&r,&k);
            int L=-inf,R=inf,mid;
            while(L<R)
            {
                mid=(L+R+1)>>1;
                if(query(1,1,n,1,l,r,mid)+1>k) R=mid-1;
                    else L=mid;
            }
            printf("%d\n",L);
        }
        else if(op==3)
        {
            scanf("%d%d",&pos,&k);
            modify(1,1,n,pos,a[pos],-1);
            a[pos]=k;
            modify(1,1,n,pos,k,1);
        }
        else
        {
            scanf("%d%d%d",&l,&r,&k);
            printf("%d\n",query(1,1,n,op,l,r,k));
        }
    }
    return 0;
}
```

4.8.2 树状数组套线段树

```
#include <bits/stdc++.h>
using namespace std;

/*
    带单点修区间k小
    用的时候注意下空间 时空 O(nlog^2(n))
    外层 add(pos,x,y) 空间上为pos的点且值域上为x的点,加上
        y
        query(l,r,k) 询问区间[l,r]里k小
    内层 modify 值域线段树动态开点
        query 值域线段树区间k小
    VAL 值域大小
*/

const int MAXN=200005;
int n,a[MAXN],X[MAXN],Y[MAXN],c1,c2,VAL;
struct SEG
{
    int root[MAXN],lc[MAXN*500],rc[MAXN*500],cnt[MAXN
        *500],tot;
    void modify(int &rt,int l,int r,int x,int y)
    {
        if(rt==0) rt=++tot;
        cnt[rt]+=y;
        if(l==r) return;
```

```
        int m=l+r>>1;
        if(x<=m) modify(lc[rt],l,m,x,y);
        else modify(rc[rt],m+1,r,x,y);
    }
    int query(int l,int r,int k)
    {
        if(l==r) return l;
        int sum=0,m=l+r>>1;
        for(int i=0;i<c1;i++) sum-=cnt[lc[X[i]]];
        for(int i=0;i<c2;i++) sum+=cnt[lc[Y[i]]];
        if(sum>=k)
        {
            for(int i=0;i<c1;i++) X[i]=lc[X[i]];
            for(int i=0;i<c2;i++) Y[i]=lc[Y[i]];
            return query(l,m,k);
        }
        else
        {
            for(int i=0;i<c1;i++) X[i]=rc[X[i]];
            for(int i=0;i<c2;i++) Y[i]=rc[Y[i]];
            return query(m+1,r,k-sum);
        }
    }
}seg;

void add(int pos,int x,int y)
{
    for(;pos<=n;pos+=pos&-pos) seg.modify(seg.root[pos
        ],1,VAL,x,y);
}

int query(int l,int r,int k)
{
    c1=c2=0;
    for(int i=l-1;i;i-=i&-i) X[c1++]=seg.root[i];
    for(int i=r;i;i-=i&-i) Y[c2++]=seg.root[i];
    return seg.query(1,VAL,k);
}
```