格式：英文/公式两边接汉字注意空格，有些地方未使用公式，如 k□$[l,r]$□

概述

树套树，就是在一个树型数据结构上，每个点不再是一个节点，而是另外一个树形数据结构。

经常应用在一些普通的数据结构外套上区间操作或者动态操作时候。没有固定的套路，根据题目来选不同的树型数据结构组合。

下面介绍一些常用的树套树

树状数组套线段树

例题 [CQOI2011]动态逆序对

现在给出 $1~n $ 的一个排列，按照某种顺序依次删除 m 个元素，你的任务是在每次删除一个元素**之前**统计整个序列的逆序对数。

我们可以建立树状数组，第 i 位维护 $a[i-lowbit(i)+1] ~ a[i]$ 的权值线段树。插入和普通的求逆序对方法相同，删除 x 的时候查询位置在后面比 x 大的数有多少即可。

为了防止内存爆找，线段树采用动态开点。

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cctype>
#include<cstring>
#define ll long long
using namespace std;

inline int read()
{
    int k=0,f=1;char c=getchar();
    while(!isdigit(c)) {if(c=='-') f=-1;c=getchar();}
    while(isdigit(c)) k=k*10+c-'0',c=getchar();return f*k;
}

const int N=100055;
int root[N],lch[N*91],rch[N*91],sum[N*91],cnt;
int n,m,pos[N];
ll anss;

void add(int &k,int l,int r,int x)
{
    if(!k) k=++cnt;sum[k]++;
    if(l==r) return ;
    int mid=l+r>>1;
```

```
    if(x<=mid) add(lch[k],l,mid,x);
    else add(rch[k],mid+1,r,x);
}

void del(int k,int l,int r,int x)
{
    if(!k) return ;
    sum[k]--;
    if(l==r) return ;
    int mid=l+r>>1;
    if(x<=mid) del(lch[k],l,mid,x);
    else del(rch[k],mid+1,r,x);
}

int query(int k,int l,int r,int a,int b)
{
    if(!k) return 0;
    if(a<=l&&b>=r) return sum[k];
    int mid=l+r>>1,ans=0;
    if(a<=mid) ans+=query(lch[k],l,mid,a,b);
    if(b>mid)  ans+=query(rch[k],mid+1,r,a,b);
    return ans;
}

int main()
{
    n=read();m=read();
    for(int i=1;i<=n;i++)
    {
        int a=read();pos[a]=i;
        for(int j=i;j<=n;j+=j&-j)
        add(root[j],1,n,a);
        for(int j=i;j;j-=j&-j)
        anss+=query(root[j],1,n,a+1,n);
    }
    for(int i=1;i<=m;i++)
    {
        printf("%lld\n",anss);
        int a=read();
        if(a!=n)
        {
            for(int j=pos[a];j;j-=j&-j) anss-=query(root[j],1,n,a+1,n);
        }
        if(a!=1)
        {
            for(int j=n;j;j-=j&-j) anss-=query(root[j],1,n,1,a-1);
            for(int j=pos[a]-1;j;j-=j&-j) anss+=query(root[j],1,n,1,a-1);
        }
        for(int j=pos[a];j<=n;j+=j&-j) del(root[j],1,n,a);
    }
```

```
    return 0;
}
```

线段树套平衡树

例题 洛谷P3380 二逼平衡树

让你维护一个有序数列，有以下操作：

1.查询k在区间内的排名

2.查询区间内排名为k的值

3.修改某一位值上的数值

4.查询k在区间内的前驱

5.查询k在区间内的后继

就是平衡时问题加上了区间限制。我们外层建线段树，每一个节点维护该节点包含区间的平衡树。

操作一和操作三对线段树上包含区间的节点全部进行操作。

操作四，五，外层线段树区间查询，对每个包含[l,r]的节点得到的前驱/后继求一个最大值/最小值即可。

操作二 我们二分答案，和操作一类似，利用小于某数的个数进行二分。复杂度多一个 log

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cctype>
using namespace std;

inline int read()
{
    int k=0,f=1;char c=getchar();
    while(!isdigit(c)) {if(c=='-') f=-1;c=getchar();}
    while(isdigit(c)) k=k*10+c-'0',c=getchar();return f*k;
}
const int N=100005,inf=2147483647;
struct T
{
    int ch[2],fa,size,cnt,v;
}tr[N*200];
int tot,rt[N*20],n,m,a[N];

#define l(x) tr[x].ch[0]
#define r(x) tr[x].ch[1]
void pu(int x)
```

```
{
    tr[x].size=tr[l(x)].size+tr[r(x)].size+tr[x].cnt;
}

void rotate(int x)
{
    int y=tr[x].fa,z=tr[y].fa,k=tr[y].ch[1]==x;
    tr[z].ch[tr[z].ch[1]==y]=x;tr[x].fa=z;
    tr[y].ch[k]=tr[x].ch[k^1];tr[tr[x].ch[k^1]].fa=y;
    tr[y].fa=x;tr[x].ch[k^1]=y;
    pu(y);pu(x);
}

void splay(int x,int r,int pos)
{
    while(tr[x].fa!=pos)
    {
        int y=tr[x].fa,z=tr[y].fa;
        if(z!=pos) (l(z)==y)^(l(y)==x)?rotate(x):rotate(y);
        rotate(x);
    }
    if(!pos) rt[r]=x;
}

void ins(int x,int r)
{
    int u=rt[r],f=0;
    if(!u)
    {
        tr[++tot].v=x;tr[tot].size=tr[tot].cnt=1;rt[r]=tot;return ;
    }
    while(u&&tr[u].v!=x) f=u,u=tr[u].ch[tr[u].v<x];
    if(u) {tr[u].cnt++;tr[u].size++;splay(u,r,0);return;}
    u=++tot;
    tr[u].fa=f;if(f) tr[f].ch[tr[f].v<x]=u;
    tr[u].v=x;tr[u].size=tr[u].cnt=1;splay(u,r,0);
}

void find(int x,int r)
{
    int u=rt[r];
    while(tr[u].v!=x&&tr[u].ch[tr[u].v<x])
    u=tr[u].ch[tr[u].v<x];
    splay(u,r,0);
}

int find_max(int x)
{
    while(r(x)) x=r(x);return x;
}
```

```
int lxt(int x,int r)
{
    int u=rt[r],ans=-inf;
    while(u)
    {
        if(tr[u].v<x) ans=max(ans,tr[u].v),u=r(u);
        else u=l(u);
    }
    return ans;
}

int rxt(int x,int r)
{
    int u=rt[r],ans=inf;
    while(u)
    {
        if(x<tr[u].v) ans=min(ans,tr[u].v),u=l(u);
        else u=r(u);
    }
    return ans;
}

void replace(int x,int y,int r)
{
    find(x,r);
    int u=rt[r];
    if(tr[u].cnt>1) tr[u].size--,tr[u].cnt--;
    else if(!tr[u].ch[0]) rt[r]=r(u),tr[r(u)].fa=0;
    else if(!tr[u].ch[1]) rt[r]=l(u),tr[l(u)].fa=0;
    else
    {
        splay(find_max(l(u)),rt[r],u);
        tr[tr[u].ch[0]].ch[1]=tr[u].ch[1];
        tr[r(u)].fa=l(u);tr[l(u)].fa=0;
        rt[r]=l(u);pu(l(u));
    }
    ins(y,r);
}

int ran(int x,int r)
{
    int u=rt[r],ans=0;
    while(u)
    {
        if(tr[u].v>x) u=l(u);
        else if(tr[u].v<x) ans+=tr[l(u)].size+tr[u].cnt,u=r(u);
        else {ans+=tr[l(u)].size;return ans;}
    }
    return ans;
}
```

```
#define lson k<<1,l,mid
#define rson k<<1|1,mid+1,r
void build(int k,int l,int r)
{
    for(int i=l;i<=r;i++) ins(a[i],k);
    if(l==r) return ;
    int mid=l+r>>1;
    build(lson);build(rson);
}

int sol1(int k,int l,int r,int a,int b,int x)
{
    if(a<=l&&b>=r)
    {
        return ran(x,k);
    }
    int mid=l+r>>1,ans=0;
    if(a<=mid) ans+=sol1(lson,a,b,x);
    if(b>mid)  ans+=sol1(rson,a,b,x);
    return ans;
}

int sol2(int l,int r,int k)
{
    int L=0,R=1e8;
    while(R>L)
    {
        int mid=L+R>>1;
        if(sol1(1,1,n,l,r,mid)<k) L=mid+1;
        else R=mid;
    }
    return L-1;
}

void sol3(int k,int l,int r,int c,int x)
{
    replace(a[c],x,k);
    if(l==r) {a[c]=x;return;}
    int mid=l+r>>1;
    if(c<=mid) sol3(lson,c,x);
    else sol3(rson,c,x);
}

int sol4(int k,int l,int r,int a,int b,int x)
{
    if(a<=l&&b>=r) return lxt(x,k);
    int mid=l+r>>1,ans=-inf;
    if(a<=mid) ans=max(ans,sol4(lson,a,b,x));
    if(b>mid) ans=max(ans,sol4(rson,a,b,x));
    return ans;
```

```
}

int sol5(int k,int l,int r,int a,int b,int x)
{
    if(a<=l&&b>=r)
    {
        return rxt(x,k);
    }
    int mid=l+r>>1,ans=inf;
    if(a<=mid) ans=min(ans,sol5(lson,a,b,x));
    if(b>mid) ans=min(ans,sol5(rson,a,b,x));
    return ans;
}

int main()
{
    int opt,b,c,d;
    n=read();m=read();
    for(int i=1;i<=n;i++) a[i]=read();
    build(1,1,n);
    for(int i=1;i<=m;i++)
    {
        opt=read();b=read();c=read();if(opt!=3) d=read();
        if(opt==1)      printf("%d\n",sol1(1,1,n,b,c,d)+1);
        else if(opt==2) printf("%d\n",sol2(b,c,d));
        else if(opt==3) sol3(1,1,n,b,c);
        else if(opt==4) printf("%d\n",sol4(1,1,n,b,c,d));
        else            printf("%d\n",sol5(1,1,n,b,c,d));
    }
    return 0;
}
```

线段树套线段树

例题 P3332 [ZJOI2013]K大数查询

题目大意

有N个位置，M个操作，操作有2种。

- 操作1，1 a b c 在第a个位置到第b个位置，每个位置加入一个数c
- 操作2，2 a b c 询问从第a个位置到第b个位置中第c大的数， $n,m\leq5*10^4,|c|\leq n$

题解

权值线段树套位置线段树。其中第一维记录权值，第二维记录位置。对于第一维线段树上的一个节点维护的权值区间$[L, R]$，它所指向的那颗线段树中记录的是每个位置包含了几个值在$[L, R]$范围内的数。

之所以这题不能像上文一样用线段树套平衡树（外层记录位置，内层维护权值）的原因是：对于套在外面那一层的线段树是很难进行区间操作的，所以这题就需要换一个思路，把位置信息放在内层的线段树中。

PS:以下代码用到了一个小技巧，就是线段树标记永久化，相比与正常线段树的pushdown,我们在路过该节点的时候把修改对答案的影响加上，这样能优化不少常数，~~否则这题你很难卡过去~~

```
#include<bits/stdc++.h>
using namespace std;
#define mem(a,b) memset(a,b,sizeof(a))
typedef long long LL;
typedef pair<int,int> PII;
#define X first
#define Y second
inline int read()
{
    int x=0,f=1;char c=getchar();
    while(!isdigit(c)){if(c=='-')f=-1;c=getchar();}
    while(isdigit(c)){x=x*10+c-'0';c=getchar();}
    return x*f;
}
const int maxn=50010,maxnode=20000005;
int T,tp,n,tot,lc[maxnode],rc[maxnode],ql,qr,v,rt[maxn<<3];;
LL addv[maxnode],sumv[maxnode];
void add(int& o,int L,int R)
{
    if(!o)o=++tot;
    if(ql<=L && R<=qr){sumv[o]+=(R-L+1);addv[o]++;return;}
    int mid=L+R>>1;
    if(ql<=mid)add(lc[o],L,mid);
    if(qr>mid)add(rc[o],mid+1,R);
    sumv[o]=sumv[lc[o]]+sumv[rc[o]]+addv[o]*(R-L+1);
    return;
}
LL query(int o,int L,int R,LL Add)
{
    if(!o)
    {
        int l=max(L,ql),r= min(R,qr);
        return Add*(r-l+1);
    }
    if(ql<=L && R<=qr) return sumv[o]+Add*(R-L+1);
    int mid=L+R>>1;
    LL ans=0;
    if(ql<=mid)ans+=query(lc[o],L,mid,Add+addv[o]);
    if(qr>mid)ans+=query(rc[o],mid+1,R,Add+addv[o]);
    return ans;
}
void update()
{
```

```
    int o=1,L=1,R=n<<1|1;
    while(L<R)
    {
        add(rt[o],1,n);
        int mid=L+R>>1,lo=o<<1,ro=lo|1;
        if(v<=mid)R=mid,o=lo;
        else L=mid+1,o=ro;
    }
    return add(rt[o],1,n);
}
int query()
{
    int o=1,L=1,R=n<<1|1;
    while(L<R)
    {
        int mid=L+R>>1,lo=o<<1,ro=lo|1;
        LL res=query(rt[ro],1,n,0);
        if(v<=res)L=mid+1,o=ro;
        else R=mid,o=lo,v-=res;
    }
    return L;
}
int main()
{
    n=read();T=read();
    while(T--)
    {
        tp=read();ql=read();qr=read();v=read();
        if(tp==1)v+=n+1,update();
        else if(tp==2)printf("%d\n",query()-n-1);
    }
    return 0;
}
```

树状数组套可持久化线段树

话不多说，上题

题目

给定一个含有n个数的序列$a[1],a[2],a[3]……a[n]$□程序必须回答这样的询问：对于给定的i,j,k□在$a[i],a[i+1],a[i+2]……a[j]$中第k小的数是多少$(1≤k≤j-i+1)$□并且，你可以改变一些$a[i]$的值，改变后，程序还能针对改变后的a继续回答上面的问题。

题解

题目询问的是区间第k小

如果不考虑区间，求第k小的话，用线段维护权值数量即可。求第k大只需在线段树上走，如果左区间权值和$L_v \geq k$就往左走，否则往右走，直到一个叶子节点，即为答案。

现在考虑区间，我们可以将线段树可持久化，就可以支持作差处理，如果要求一段区间$[l,r]$的第k小，我们只需在r处的线段树和$l - 1$处线段树作差后求出第k大。

如果还要支持单点修改，我们线段树的建立就不能单纯的在区间的基础上。如果仍然如此建树，则每次修改区间都要修改该点之后的线段树，数目是$O(n)$的。这个时候我们可以在树状数组的每个点上建树。如此，每个点有关的线段树只有$O(logn)$总复杂度降到$O(nlog^2n)$

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#define REP(i,n) for (int i = 1; i <= (n); i++)
#define lbt(x) (x & -x)
using namespace std;
const int maxn = 10005,maxm = 10000005,INF = 1000000000;
inline int read(){
    int out = 0,flag = 1; char c = getchar();
    while (c < 48 || c > 57) {if (c == '-') flag = -1; c = getchar();}
    while (c >= 48 && c <= 57) {out = (out << 3) + (out << 1) + c - '0'; c
= getchar();}
    return out * flag;
}
int rt[maxn],A[maxn],B[2 * maxn],n,m,tot = 1,siz,N;
int sum[maxm],ls[maxm],rs[maxm],a[2][20];
struct Que{int opt,l,r,k;}Q[maxn];
int getn(int x){
    int l = 1,r = tot,mid;
    while (l <= r){
        mid = l + r >> 1;
        if (B[mid] < x) l = mid + 1;
        else r = mid - 1;
    }
    return l;
}
void update(int& u,int l,int r,int pos,int v){
    if (!u) u = ++siz; sum[u] += v;
    if (l == r) return;
    int mid = l + r >> 1;
    if (mid >= pos) update(ls[u],l,mid,pos,v);
    else update(rs[u],mid + 1,r,pos,v);
}
int query(int l,int r,int k){
    if (l == r) return l;
    int mid = l + r >> 1,t = 0;
    for (int i = 1; i <= a[0][0]; i++) t += sum[ls[a[0][i]]];
    for (int i = 1; i <= a[1][0]; i++) t -= sum[ls[a[1][i]]];
```

```
    if (t >= k){
        for (int i = 1; i <= a[0][0]; i++) a[0][i] = ls[a[0][i]];
        for (int i = 1; i <= a[1][0]; i++) a[1][i] = ls[a[1][i]];
        return query(l,mid,k);
    }else {
        for (int i = 1; i <= a[0][0]; i++) a[0][i] = rs[a[0][i]];
        for (int i = 1; i <= a[1][0]; i++) a[1][i] = rs[a[1][i]];
        return query(mid + 1,r,k - t);
    }
}
void add(int u,int x,int v){while (u <= n) update(rt[u],1,tot,x,v),u +=
lbt(u);}
int solve(int l,int r,int k){
    a[0][0] = a[1][0] = 0;
    for (int i = r; i; i -= lbt(i)) a[0][++a[0][0]] = rt[i];
    for (int i = l - 1; i; i -= lbt(i)) a[1][++a[1][0]] = rt[i];
    return query(1,tot,k);
}
int main(){
    n = read(); m = read(); char c;
    REP(i,n) A[i] = B[++N] = read();
    REP(i,m){
        c = getchar(); while (c != 'Q' && c != 'C') c = getchar();
        if (c == 'Q') Q[i].opt = 0,Q[i].l = read(),Q[i].r = read(),Q[i].k =
read();
        else Q[i].opt = 1,Q[i].l = read(),Q[i].k = B[++N] = read();
    }
    sort(B + 1,B + 1 + N);
    for (int i = 2; i <= N; i++) if (B[i] != B[tot]) B[++tot] = B[i];
    REP(i,n) A[i] = getn(A[i]),add(i,A[i],1);
    REP(i,m){
        if (!Q[i].opt) printf("%d\n",B[solve(Q[i].l,Q[i].r,Q[i].k)]);
        else{
            Q[i].k = getn(Q[i].k);
            add(Q[i].l,A[Q[i].l],-1);
            A[Q[i].l] = Q[i].k;
            add(Q[i].l,A[Q[i].l],1);
        }
    }
    return 0;
}
```

以上就是树状数组套可持久化线段树的应用，通常用于一个二维空间的整体查询单点修改的操作。

再贴一道板题

[TJOI2017]不勤劳的图书管理员