# 数据结构

## 树状数组

一组数据，要求能够区间修改和区间询问．同时要完成两种操作，只要维护两个树状数组就好了．一个差分数组，另一个是对差分数组进行求和的数组．

$$\begin{aligned} Sum(n) = \sum_{i=1}^{n} a_i &= a_n + a_{n-1} + \ldots + a_1 \\ &= (d_n + \ldots + d_1) + (d_{n-1} + \ldots + d_1) + \ldots + d_1 \\ &= (n-(n-1)) * d_n + (n-(n-2)) * d_{n-1} + \ldots + (n-0) * d_1 \\ &= n * \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} d_i * (i-1) \end{aligned}$$

```
ll lowbit(ll x){    return x&(-x);}

void update(ll i,ll val)
{
    ll x=i;
    while(i<=N){
        a[i]+=val;
        b[i]+=(val*(x-1));
        i+=lowbit(i);
    }
}

ll getsum(ll i)
{
    ll res=0,x=i;
    while(i>0){
        res+=(a[i]*x-b[i]);
        i-=lowbit(i);
    }
    return res;
}

int main()
{
    ll Q,num1,num2=0;
    scanf ("%lld %lld",&N,&Q);
    for (int i=1;i<=N;i++){
        scanf ("%lld",&num1);
        update(i,num1-num2);
        num2=num1;
    }e
    for (int i=1;i<=Q;i++)
    {
        char op=getchar();
        ll x,y,k;
        if (op=='C') {
            scanf ("%lld %lld %lld",&x,&y,&k);
            update(x,k);
            update(y+1,-k);
        }
        else if (op=='Q') {
```

```
            scanf("%lld %lld",&x,&y);
            printf("%lld\n",getsum(y)-getsum(x-1));
        }
        else i--;
    }
    return 0;
}
```

## ST表

```
int Min(int a,int b)
{
    return s[a]<=s[b]?a:b;
}

void getST()
{
    len=strlen(s);
    memset(ST,0,sizeof(ST));
    for (int j=0;j<=11;j++){
        for (int i=0;i+bin[j]-1<len;i++){
            if (j==0) ST[i][j]=i;
            else ST[i][j]=Min(ST[i][j-1],ST[i+bin[j-1]][j-1]);
        }
    }
}

int STsearch(int x,int y)
{
    if (x==y) return x;
    int t=Log[y-x+1];
    return Min(ST[x][t],ST[y-bin[t]+1][t]);
}
```

## 线段树标记

```
void build(int o,int l,int r)
{
    lazy1[o]=1;
    if (l==r) { sum[o]=num[l]%p;    return ;}
    build(lo,l,M);
    build(ro,M+1,r);
    sum[o]=(sum[lo]+sum[ro])%p;
}

void pushdown(int o,int l,int r)
{
    lazy1[lo]*=lazy1[o];    lazy1[lo]%=p;
    lazy1[ro]*=lazy1[o];    lazy1[ro]%=p;
    lazy2[lo]=lazy2[lo]*lazy1[o]+lazy2[o];  lazy2[lo]%=p;
    lazy2[ro]=lazy2[ro]*lazy1[o]+lazy2[o];  lazy2[ro]%=p;
    sum[lo]=((sum[lo]*lazy1[o])+lazy2[o]*len(l,M))%p;
    sum[ro]=((sum[ro]*lazy1[o])+lazy2[o]*len(M+1,r))%p;
    lazy1[o]=1;
    lazy2[o]=0;
}
```

```
ll getsum(int o,int l,int r)
{
    ll res=0;
    if (x<=l && r<=y) return sum[o]%p;
    pushdown(o,l,r);
    if ( M >= x) {
        res+=getsum(lo,l,M);
        res%=p;
    }
    if ( M < y) {
        res+=getsum(ro,M+1,r);
        res%=p;
    }
    return res%p;
}

void addval(int o,int l,int r)
{
    if (x<=l && r<=y) {
        sum[o]=(sum[o]+k*len(l,r))%p;
        lazy2[o]+=k;    lazy2[o]%=p;
        return ;
    }
    pushdown(o,l,r);
    if ( M >= x ) addval(lo,l,M);
    if ( M < y ) addval(ro,M+1,r);
    sum[o]=sum[lo]+sum[ro];
    sum[o]%=p;
}

void multival(int o,int l,int r)
{
    if (x<=l && r<=y) {
        sum[o]=(sum[o]*k)%p;
        lazy1[o]*=k;lazy1[o]%=p;
        lazy2[o]*=k;lazy2[o]%=p;
        return ;
    }
    pushdown(o,l,r);
    if ( M >= x) multival(lo,l,M);
    if ( M < y) multival(ro,M+1,r);
    sum[o]=sum[lo]+sum[ro];
    sum[o]%=p;
}
```

## 扫描线

```
void do_odd(ll o, ll L, ll R)
{
    ll all = num[R + 1] - num[L];
    tree[o].sum2 = (all - tree[o].sum2);
}
void pushup(ll o, ll L, ll R)
{
    if (tree[o].lazy)
        tree[o].sum1 = num[R + 1] - num[L];
```

```
    else
        tree[o].sum1 = tree[lo].sum1 + tree[ro].sum1;
}
void pushdown(ll o, ll L, ll R)
{
    if (tree[o].lazy2) {
        ll mid = (L + R) >> 1;
        do_odd(lo, L, mid);
        do_odd(ro, mid + 1, R);
        tree[lo].lazy2 ^= 1;
        tree[ro].lazy2 ^= 1;
        tree[o].lazy2 = 0;
    }
}
void addval(ll o, ll L, ll R)
{
    if (num[R + 1] <= ql || num[L] >= qr)
        return ;
    if (ql <= num[L] && num[R + 1] <= qr) {
        tree[o].lazy += k;
        do_odd(o, L, R);
        tree[o].lazy2 ^= 1;
        pushup(o, L, R);
        return ;
    }
    pushdown(o, L, R);
    ll M = (L + R) / 2;
    addval(lo, L, M);
    addval(ro, M + 1, R);
    pushup(o, L, R);
    tree[o].sum2 = tree[lo].sum2 + tree[ro].sum2;
}
```

## 树链剖分

```
void dfs1(int x, int f, int d)
{
    fa[x] = f;
    siz[x] = 1;
    dep[x] = d;
    int maxson = -1;
    for (int i = first[x]; i; i = inext[i]) {
        if (to[i] == f)
            continue;
        dfs1(to[i], x, d + 1);
        siz[x] += siz[to[i]];
        if (siz[to[i]] > maxson)
            maxson = siz[to[i]], son[x] = to[i];
    }
}
void dfs2(int x, int topf)
{
    id[x] = ++cur;
    top[x] = topf;
    w2[cur] = w1[x];
    if (!son[x])
        return ;
```

```
    dfs2(son[x], topf);
    for (int i = first[x]; i; i = inext[i]) {
        if (to[i] != fa[x] && to[i] != son[x])
            dfs2(to[i], to[i]);
    }
}
int lss(int beg, int end)
{
    int ans = 0;
    while (top[beg] != top[end]) {
        if (dep[top[beg]] < dep[top[end]])
            swap(beg, end);
        ans +=sum(1, 1, n, id[top[beg]], id[beg]);
        beg = fa[top[beg]];
    }
    if (dep[beg] > dep[end])
        swap(beg, end);
    ans += sum(1, 1, n, id[beg], id[end]);
    return ans ;
}
int imax(int beg, int end)
{
    int ans = -inf;
    while (top[beg] != top[end]) {
        if (dep[top[beg]] < dep[top[end]])
            swap(beg, end);
        ans = max(ans, query(1, 1, n, id[top[beg]], id[beg]));
        beg = fa[top[beg]];
    }
    if (dep[beg] > dep[end])
        swap(beg, end);
    ans = max(ans, query(1, 1, n, id[beg], id[end]));
    return ans ;
}
```

## 可持久化并查集

```
inline int find(int root, int x)
{
    node now = Tree[root];
    if (now.l == now.r)
        return now.fa;
    int mid = (now.l + now.r) >> 1;
    if (x <= mid)
        return find(now.ls,x);
    else
        return find(now.rs,x);
}

inline int getfa(int root, int x)
{
    int fa = find(root, x);
    if (fa == x)
        return x;
    else
        return getfa(root, fa);
}
```

```
inline int update(const int pre, int loc, int val)
{
    int now = tot++;
    copy(now, pre);
    if (Tree[now].l == Tree[now].r ) {
        Tree[now].fa = val;
        return now;
    }
    int mid = (Tree[now].l + Tree[now].r) >> 1;
    if (loc <= mid)
        Tree[now].ls = update(Tree[now].ls,loc,val);
    else
        Tree[now].rs = update(Tree[now].rs,loc,val);
    return now;
}
```

## SBT

```
void rotate(int &rt, int k)
{
    int nrt = son[rt][!k]; //new root
    son[rt][!k] = son[nrt][k];
    son[nrt][k] = rt;
    siz[nrt] = siz[rt];
    siz[rt] = siz[son[rt][0]] + siz[son[rt][1]] + 1;
    rt = nrt;
}

void maintain(int &rt, bool k)
{
    if (siz[son[son[rt][k]][k]] > siz[son[rt][!k]])
        rotate(son[rt][k], !k);
    else if (siz[son[son[rt][k]][!k]] > siz[son[rt][!k]])
        rotate(son[rt][k], k), rotate(rt, !k);
    else
        return ;
    maintain(son[rt][0], 0);
    maintain(son[rt][1], 1);
    maintain(rt, 0);
    maintain(rt, 1);
}

void insert(int &rt, int val)
{
    if (rt == 0) {
        rt = ++tails;
        datas[rt] = val;
        son[rt][0] = son[rt][1] = 0;
        siz[rt] = 1;
        return ;
    }
    ++siz[rt];
    if (val <= datas[rt])
        insert(son[rt][0], val);
    else
```

```
        insert(son[rt][1], val);
    maintain(rt, datas[rt] > val);
}

void del(int &rt, int val)
{
    if (datas[rt] != val) {
        del(son[rt][datas[rt] < val], val);
        siz[rt] = siz[son[rt][0]] + siz[son[rt][1]] + 1;
        return ;
    }
    --siz[rt];
    if (son[rt][0] == 0)
        rt = son[rt][1];
    else if (son[rt][1] == 0)
        rt = son[rt][0];
    else {
        int p = son[rt][1];
        while (son[p][0] != 0)
            p = son[p][0];
        datas[rt] = datas[p];
        del(son[rt][1], datas[p]);
    }
}

int pre (int val)
{
    int cur = sbt, ans = 0;
    while (cur != 0) {
        if (datas[cur] < val)
            ans = cur, cur = son[cur][1];
        else
            cur = son[cur][0];
    }
    return datas[ans];
}

int nxt (int val)
{
    int cur = sbt, ans = 0;
    while (cur != 0) {
        if (datas[cur] > val)
            ans = cur, cur = son[cur][0];
        else
            cur = son[cur][1];
    }
    return datas[ans];
}

int xth(int x)
{
    int cur = sbt;
    while (siz[son[cur][0]] + 1 != x) {
        if (siz[son[cur][0]] + 1 < x)
            x -= (siz[son[cur][0]] + 1), cur = son[cur][1];
        else
            cur = son[cur][0];
    }
```

```
    return datas[cur];
}

int ranks(int val)
{
    int cur = sbt, ans = 0;
    while (cur != 0) {
        if (datas[cur] < val)
            ans += (siz[son[cur][0]] + 1), cur = son[cur][1];
        else
            cur = son[cur][0];
    }
    return ans + 1;
}
```

## 莫队 回滚莫队

```
for (int k = 0; k <= blockn; ++k)
{
    int l = ranger[k] + 1, r = ranger[k];
    memset(cnt, 0, sizeof(cnt));
    ll temp = 0, now = 0;
    for (; block[q[i].l] == k; ++i) {
        int ql = q[i].l, qr = q[i].r;
        if (block[q[i].l] == block[q[i].r]) {
            temp = 0;
            for (int j = ql; j <= qr; ++j)
                cnt2[id[j]] = 0;
            for (int j = ql; j <= qr; ++j) {
                if (!cnt2[id[j]])
                    cnt2[id[j]] = j;
                else
                    temp = max(temp, 1ll * abs(j - cnt2[id[j]]));
            }
            ans[q[i].id] = temp;
            continue;
        }
        while (r < qr) {
            if (!cnt[id[++r]][0])
                cnt[id[r]][1] = cnt[id[r]][0] = r;
            else {
                cnt[id[r]][1] = r;
                now = max(now, 1ll * abs(r - cnt[id[r]][0]));
            }
        }
        temp = now;
        while (ql < l) {
            if (!cnt[id[--l]][1])
                cnt[id[l]][1] = l;
            else
                temp = max(temp, 1ll * abs(l - cnt[id[l]][1]));
        }
        ans[q[i].id] = temp;
        while (l < ranger[k] + 1) {
            if (cnt[id[l]][1] < ranger[k] + 1)
                cnt[id[l]][1] = 0;
            ++l;
```

```
        }
    }
}
```

# 字符串

## KMP

```
void getnextarr(char* str)
{
    int n=strlen(str);
    nexts[0]=-1,nexts[1]=0;
    for (int i=2;i<=n;i++)
    {
        int k=i-1;
        while (1){
            if (nexts[k]==-1 || str[i-1]==str[nexts[k]])
            {
                nexts[i]=nexts[k]+1;
                break;
            }
            else k=nexts[k];
        }
    }
    return ;
}

void kmp()
{
    int len1=strlen(str1);
    int len2=strlen(str2);
    int i=0,j=0;
    getnextarr(str2);
    while (i<len1)
    {
        if (str1[i]==str2[j]){
                if (j!=len2-1) i++,j++;
            else {
                ans[cnt++]=i-j+1;
                j=nexts[j];
            }
        }
        else if (nexts[j]==-1) i++;
        else j=nexts[j];
    }
    for (int t=0;t<cnt;t++) printf("%d\n",ans[t]);
    for (int t=0;t<len2;t++) {
         printf("%d ",nexts[t+1]);
    }
    return ;
}
```

## AC自动机

```
struct node{
    int f;
    int book;
    int next[26];
}tree[1000051];
struct imark{
    int num,str;
}ans[1000051];
bool cmp(imark a,imark b){
    if (a.num==b.num) return a.str<b.str;
    else return a.num>b.num;
}
char a[155][1000];
void clean(int x){
    memset(tree[x].next,0,sizeof(tree[x].next));
    tree[x].f=0;
    tree[x].book=0;
}
char s[1000506];
int ta=0;
void build(int x){
    int cur=0,l=strlen(a[x]);
    for (int i=0;i<l;++i){
        if (!tree[cur].next[a[x][i]-'a']) {
            tree[cur].next[a[x][i]-'a']=++ta;
            clean(ta);
        }
        cur=tree[cur].next[a[x][i]-'a'];
    }
    tree[cur].book=x;
}
```

## 马拉车

```
string Mannacher(string s)
{
    //插入"#"
    string t="$#";
    for(int i=0;i<s.size();++i)
    {
        t+=s[i];
        t+="#";
    }

    vector<int> p(t.size(),0);
    //mx表示某个回文串延伸在最右端半径的下标，id表示这个回文子串最中间位置下标
    //resLen表示对应在s中的最大子回文串的半径，resCenter表示最大子回文串的中间位置
    int mx=0,id=0,resLen=0,resCenter=0;
     //建立p数组
    for(int i=1;i<t.size();++i)
    {
        p[i]=mx>i?min(p[2*id-i],mx-i):1;
        //遇到三种特殊的情况，需要利用中心扩展法
        while(t[i+p[i]]==t[i-p[i]])++p[i];
```

```
        //半径下标i+p[i]超过边界mx，需要更新
        if(mx<i+p[i]){
            mx=i+p[i];
            id=i;
        }
        //更新最大回文子串的信息，半径及中间位置
        if(resLen<p[i]){
            resLen=p[i];
            resCenter=i;
        }
    }
    //最长回文子串长度为半径-1，起始位置为中间位置减去半径再除以2
    return s.substr((resCenter-resLen)/2,resLen-1);
}
```