

- 块状数组
- 块状数组的区间修改和区间查询
- 并查集启发式合并
- ST表
- 莫队
- 笛卡尔树
- BIT
- Segment tree

## 块状数组

```
num = sqrt(n);
for (int i = 1; i <= num; i++)
    st[i] = n / num * (i - 1) + 1, ed[i] = n / num * i;
ed[num] = n;
for (int i = 1; i <= num; i++) {
    for (int j = st[i]; j <= ed[i]; j++) {
        belong[j] = i;
    }
    size[i] = ed[i] - st[i] + 1;
}
```

## 块状数组的区间修改和区间查询

```
void Sort(int k) {
    for (int i = st[k]; i <= ed[k]; i++) t[i] = a[i];
    sort(t + st[k], t + ed[k] + 1);
}

void Modify(int l, int r, int c) {
    int x = belong[l], y = belong[r];
    if (x == y) // 区间在一个块内就直接修改
    {
        for (int i = l; i <= r; i++) a[i] += c;
        Sort(x);
        return;
    }
    for (int i = l; i <= ed[x]; i++) a[i] += c; // 直接修改起始段
    for (int i = st[y]; i <= r; i++) a[i] += c; // 直接修改结束段
    for (int i = x + 1; i < y; i++) delta[i] += c; // 中间的块整体打上标记
    Sort(x);
    Sort(y);
}
```

```

}

int Answer(int l, int r, int c) {
    int ans = 0, x = belong[l], y = belong[r];
    if (x == y) {
        for (int i = l; i <= r; i++)
            if (a[i] + delta[x] >= c) ans++;
        return ans;
    }
    for (int i = l; i <= ed[x]; i++)
        if (a[i] + delta[x] >= c) ans++;
    for (int i = st[y]; i <= r; i++)
        if (a[i] + delta[y] >= c) ans++;
    for (int i = x + 1; i <= y - 1; i++)
        ans +=
            ed[i] - (lower_bound(t + st[i], t + ed[i] + 1, c - delta[i]) - t) + 1;
    // 用 lower_bound 找出中间每一个整块中第一个大于等于 c 的数的位置
    return ans;
}

```

## 并查集启发式合并

```

struct dsu {
    vector<size_t> pa, size;

    explicit dsu(size_t size_) : pa(size_), size(size_, 1) {
        iota(pa.begin(), pa.end(), 0);
    }

    void unite(size_t x, size_t y) {
        x = find(x), y = find(y);
        if (x == y) return;
        if (size[x] < size[y]) swap(x, y);
        pa[y] = x;
        size[x] += size[y];
    }
};

```

## ST表

```

void prepare(){
    logn[2]=1;
    logn[1]=0;
    for(int i=3;i<MAXN;i++){
        logn[i]=logn[i/2]+1;
    }
}

```

```

}
for(int i=1;i<=21;i++){
    for(int j=1;j+(1<<i)-1<=n;j++){
        dp[j][i]=max(dp[j][i-1],dp[j+(1<<(i-1))][i-1]);
    }
}
int s=logn[y-x+1];
printf("%d\n",max(dp[x][s],dp[y-(1<<s)+1][s]));

```

## 莫队

```

struct query {
    int l, r, id;
} q[maxn];
int cmp(query a, query b) {
    return (belong[a.l] ^ belong[b.l]) ? belong[a.l] < belong[b.l] :
((belong[a.l] & 1) ? a.r < b.r : a.r > b.r);
} //对排序的优化
void add(int pos) {
    if(!cnt[aa[pos]]) ++now;
    ++cnt[aa[pos]];
}
void del(int pos) {
    --cnt[aa[pos]];
    if(!cnt[aa[pos]]) --now;
}
size = sqrt(n);
bnum = ceil((double)n / size);
for(int i = 1; i <= bnum; ++i)
    for(int j = (i - 1) * size + 1; j <= i * size; ++j) {
        belong[j] = i;
    }
while(l < q1) del(l++);
while(l > q1) add(--l);
while(r < qr) add(++r);
while(r > qr) del(r--);

while(l < q1) now -= !--cnt[a[l++]];
while(l > q1) now += !cnt[a[--l]]++;
while(r < qr) now += !cnt[a[++r]]++;
while(r > qr) now -= !--cnt[a[r--]]; //对修改操作的优化

```

## 笛卡尔树

```

int a[N]; //存数据
stack<int> q;
int ls[N], rs[N]; //存左右儿子

```

```

void create_tree1(){
    fr(i,n){
        int k=0;
        while(!q.empty()&&q.top()>a[i]){
            k=q.top();
            q.pop();
        }
        if(!q.empty())rs[q.top()]=a[i];
        if(k>0)ls[a[i]]=k;
        q.push(a[i]);
    }
}

void create_tree2(){
    fr(i,n){
        int k=0;
        while(!q.empty()&&a[q.top()]>a[i]){
            k=q.top();
            q.pop();
        }
        if(!q.empty())rs[q.top()]=i;
        if(a[k]>0)ls[i]=k;
        q.push(i);
    }
}

```

# BIT

```

struct BIT{
    vec tree;
    void build(int n){
        tree.assign(n,0);
    }
    void init(int n){
        for(int i=0;i<=n;i++)tree[i]=0;
    }
    void add(int x,int v,int n){
        for(;x<=n;x+=lowbit(x)){
            tree[x]+=v;
        }
    }
    int query(int x,int n){
        int res=0;
        for(;x;x-=lowbit(x)){
            res+=tree[x];
        }
        return res;
    }
}

```

# Segment tree

```
struct seg{
    int sum,l,r,add;
}tr[4*N];
int a[N];
inline void pushup(int x){
    tr[x].sum+=tr[2*x].sum+tr[2*x+1].sum;
}
inline void pushdown(int x){//将懒标记下放到子节点
    if(tr[x].add){
        tr[2*x].add+=tr[x].add;
        tr[2*x+1].add+=tr[x].add;
        tr[2*x].sum+=tr[x].add*(tr[2*x].r-tr[2*x].l+1);
        tr[2*x+1].sum+=tr[x].add*(tr[2*x+1].r-tr[2*x+1].l+1);
        tr[x].add=0;
    }
}
void build(int x,int l,int r){
    tr[x].l=l,tr[x].r=r;
    if(l==r){
        tr[x].sum=a[l];
        return;
    }
    int mid=(l+r)>>1;
    build(2*x,l,mid);
    build(2*x+1,mid+1,r);
    pushup(x);
}
void change(int now,int x,int val){// 单点修改
    if(tr[now].l==tr[now].r){
        tr[now].sum+=val;
        return;
    }
    int mid=(tr[now].l+tr[now].r)>>1;
    if(x<=mid){
        change(now*2,x,val);
    }
    else {
        change(now*2+1,x,val);
    }
    pushup(now);
}
int query(int x,int l,int r){//区间查询
    if(l<=tr[x].l&&r>=tr[x].r)return tr[x].sum;
    pushdown(x);
    int mid=(tr[x].l+tr[x].r)>>1;
    int sum=0;
    if(l<=mid){
        sum+=query(2*x,l,r);
    }
    if(r>mid){
        sum+=query(2*x+1,l,r);
    }
}
```

```
        return sum;
    }
    void update(int l,int r,int x,int k){//区间修改
        if(l<=tr[x].l&&tr[x].r){
            tr[x].sum+=k*(tr[x].r-tr[x].l+1);
            tr[x].add+=k;
        }
        else {
            pushdown(x);
            int mid=(tr[x].l+tr[x].r)>>1;
            if(l<=mid){
                update(l,r,2*x,k);
            }
            if(r>mid){
                update(l,r,2*x+1,k);
            }
            pushup(x);
        }
    }
}
```