第四章 数据结构目录

1、划分树,查询区间第 K 大

```
/*划分树(查询区间第 k 大)  */
const int MAXN = 100010;
int tree[20][MAXN];        //  表示每层每个位置的值
int sorted[MAXN];          //  已经排序好的数
int toleft[20][MAXN];     //  toleft[p][i]表示第 i 层从 1 到 i 有数分入左边
void build(int l, int r, int dep)
{
    if (l == r)       return;
    int mid = (l + r) >> 1;
    int same = mid - l + 1;              //  表示等于中间值而且被分入左边的个数
    for (int i = l; i <= r; i++){       //  注意是 l,不是 one
        if (tree[dep][i] < sorted[mid])       same--;
    }
    int lpos = l;
    int rpos = mid + 1;
    for (int i = l; i <= r; i++){
        if (tree[dep][i] < sorted[mid])     tree[dep + 1][lpos++] = tree[dep][i];
        else if (tree[dep][i] == sorted[mid] && same > 0){
            tree[dep + 1][lpos++] = tree[dep][i];
            same--;
        }
        else     tree[dep + 1][rpos++] = tree[dep][i];
        toleft[dep][i] = toleft[dep][l - 1] + lpos - l;
    }
    build(l, mid, dep + 1);
    build(mid + 1, r, dep + 1);
    return ;
}
//  查询区间第 k 大的数,[L,R]是大区间,[l,r]是要查询的小区间
int query(int L, int R, int l, int r, int dep, int k){
    if(l == r)       return tree[dep][l];
    int mid = (L + R) >> 1;
    int cnt = toleft[dep][r] - toleft[dep][l - 1];
    if (cnt >= k){
        int newl = L + toleft[dep][l - 1] - toleft[dep][L - 1];
        int newr = newl + cnt - 1;
        return query(L, mid, newl, newr, dep + 1, k);
    }
    else{
        int newr = r + toleft[dep][R] - toleft[dep][r];
        int newl = newr - (r - l - cnt);
        return query(mid + 1, R, newl, newr, dep + 1, k - cnt);
```

```
    }
}
/*int tree[20][MAXN];        //   表示每层每个位置的值
  int sorted[MAXN];           //   已经排序好的数
  int toleft[20][MAXN];     //   toleft[p][i]表示第 i 层从 1 到 i 有数分入左边*/
int main(){
    int n, m; //n 个数 , m 次查询
    while (scanf("%d%d", &n, &m) == 2){
        memset(tree, 0, sizeof(tree));
        for (int i = 1; i <= n; i++){
            scanf("%d", &tree[0][i]);
            sorted[i] = tree[0][i];
        }
        sort(sorted + 1, sorted + n + 1);
        build(1, n, 0);
        int s, t, k;
        while(m--){
            scanf("%d%d%d", &s, &t, &k);
            printf("%d\n", query(1, n, s, t, 0, k));
        }
    }
    return 0;
}
```

2、伸展树,区间子序列最小值

```
/*   伸展树(Splay Tree)
 *   题目:维修数列。
 *   经典题,插入、删除、修改、翻转、求和、求和最大的子序列*/
#define Key_value ch[ch[root][1]][0]
const int MAXN = 500010;
const int INF = 0x3f3f3f3f;
int pre[MAXN], ch[MAXN][2], key[MAXN], size[MAXN];
int root, tot1;
int sum[MAXN], rev[MAXN], same[MAXN];
int lx[MAXN], rx[MAXN], mx[MAXN];
int s[MAXN], tot2;          //   内存池和容量
int a[MAXN];
int n, q;
//   debug Start**********************************
void Treavel(int x){
    if (x){
        Treavel(ch[x][0]);
        printf("结点:%2d: 左儿子 %2d 右儿子 %2d 父结点 %2d size = %2d\n", x, ch[x][0],
```

```
ch[x][1], pre[x], size[x]);
            Treavel(ch[x][1]);
        }
        return ;
}
void debug(){
        printf("root:%d\n", root);
        Treavel(root);
        return ;
}
//    debug End**********************************
void NewNode(int &r, int father, int k){
        if (tot2)    r = s[tot2--];    //    取的时候是 tot2--,存的时候就是++tot2
        else      r = ++tot1;
        pre[r] = father;
        ch[r][0] = ch[r][1] = 0;
        key[r] = k;
        sum[r] = k;
        rev[r] = same[r] = 0;
        lx[r] = rx[r] = mx[r] = k;
        size[r] = 1;
        return ;
}
void Update_Rev(int r){
        if (!r)      return ;
        swap(ch[r][0], ch[r][1]);
        swap(lx[r], rx[r]);
        rev[r] ^= 1;
        return ;
}
void Update_Same(int r, int v)
{
        if (!r)      return ;
        key[r] = v;
        sum[r] = v * size[r];
        lx[r] = rx[r] = mx[r] = max(v, v * size[r]);
        same[r] = 1;
        return ;
}
void push_up(int r){
        int lson = ch[r][0], rson = ch[r][1];
        size[r] = size[lson] + size[rson] + 1;
        sum[r] = sum[lson] + sum[rson] + key[r];
        lx[r] = max(lx[lson], sum[lson] + key[r] + max(0, lx[rson]));
```

```
    rx[r] = max(rx[rson], sum[rson] + key[r] + max(0, rx[lson]));
    mx[r] = max(0, rx[lson]) + key[r] + max(0, lx[rson]);
    mx[r] = max(mx[r], max(mx[lson], mx[rson]));
    return ;
}
void push_down(int r){
    if (same[r]){
        Update_Same(ch[r][0], key[r]);
        Update_Same(ch[r][1], key[r]);
        same[r] = 0;
    }
    if(rev[r]){
        Update_Rev(ch[r][0]);
        Update_Rev(ch[r][1]);
        rev[r] = 0;
    }
    return ;
}
void Build(int &x, int l, int r, int father){
    if (l > r)      return ;
    int mid = (l + r) / 2;
    NewNode(x, father, a[mid]);
    Build(ch[x][0], l, mid - 1, x);
    Build(ch[x][1], mid + 1, r, x);
    push_up(x);
    return ;
}
void Init(){
    root = tot1 = tot2 = 0;
    ch[root][0] = ch[root][1] = size[root] = pre[root] = 0;
    same[root] = rev[root] = sum[root] = key[root] = 0;
    lx[root] = rx[root] = mx[root] = -INF;
    NewNode(root, 0, -1);
    NewNode(ch[root][1], root, -1);
    for (int i = 0; i < n; i++)     scanf("%d", &a[i]);
    Build(Key_value, 0, n - 1, ch[root][1]);
    push_up(ch[root][1]);
    push_up(root);
}
//  旋转,0 为左旋,1 为右旋
void Rotate(int x,int kind){
    int y = pre[x];
    push_down(y);
    push_down(x);
```

```
    ch[y][!kind] = ch[x][kind];
    pre[ch[x][kind]] = y;
    if (pre[y])
        ch[pre[y]][ch[pre[y]][1]==y] = x;
    pre[x] = pre[y];
    ch[x][kind] = y;
    pre[y] = x;
    push_up(y);
}
//  Splay 调整,将 r 结点调整到 goal 下面
void Splay(int r, int goal){
    push_down(r);
    while (pre[r] != goal){
        if (pre[pre[r]] == goal){
            push_down(pre[r]);
            push_down(r);
            Rotate(r, ch[pre[r]][0] == r);
        }
        else{
            push_down(pre[pre[r]]);
            push_down(pre[r]);
            push_down(r);
            int y = pre[r];
            int kind = ch[pre[y]][0] == y;
            if (ch[y][kind] == r){
                Rotate(r, !kind);
                Rotate(r, kind);
            }
            else{
                Rotate(y, kind);
                Rotate(r, kind);
            }
        }
    }
    push_up(r);
    if (goal == 0)      root = r;
    return ;
}
int Get_kth(int r, int k){
    push_down(r);
    int t = size[ch[r][0]] + 1;
    if (t == k)     return r;
    if (t > k)      return Get_kth(ch[r][0], k);
    else       return Get_kth(ch[r][1], k - t);
```

```
}
//   在第 pos 个数后面插入 tot 个数
void Insert(int pos, int tot){
      for (int i = 0; i < tot; i++)       scanf("%d",&a[i]);
      Splay(Get_kth(root, pos + 1), 0);
      Splay(Get_kth(root, pos + 2), root);
      Build(Key_value, 0, tot - 1, ch[root][1]);
      push_up(ch[root][1]);
      push_up(root);
      return ;
}
//   删除子树
void erase(int r){
      if (!r)       return ;
      s[++tot2] = r;
      erase(ch[r][0]);
      erase(ch[r][1]);
      return ;
}
//   从第 pos 个数开始连续删除 tot 个数
void Delete(int pos, int tot){
      Splay(Get_kth(root, pos), 0);
      Splay(Get_kth(root, pos + tot + 1), root);
      erase(Key_value);
      pre[Key_value] = 0;
      Key_value = 0;
      push_up(ch[root][1]);
      push_up(root);
      return ;
}
//   将从第 pos 个数开始的连续的 tot 个数修改为 c
void Make_Same(int pos, int tot, int c){
      Splay(Get_kth(root, pos), 0);
      Splay(Get_kth(root, pos + tot + 1), root);
      Update_Same(Key_value, c);
      push_up(ch[root][1]);
      push_up(root);
      return ;
}
//   将第 pos 个数开始的连续 tot 个数进行反转
void Reverse(int pos, int tot){
      Splay(Get_kth(root, pos), 0);
      Splay(Get_kth(root,pos+tot + 1), root);
      Update_Rev(Key_value);
```

```
        push_up(ch[root][1]);
        push_up(root);
        return ;
}
//    得到第 pos 个数开始的 tot 个数的和
int Get_Sum(int pos, int tot){
        Splay(Get_kth(root, pos), 0);
        Splay(Get_kth(root, pos + tot + 1), root);
        return sum[Key_value];
}
//    得到第 pos 个数开始的 tot 个数中最大的子段和
int Get_MaxSum(int pos, int tot){
        Splay(Get_kth(root, pos), 0);
        Splay(Get_kth(root, pos + tot + 1), root);
        return mx[Key_value];
}
void InOrder(int r){
        if (!r)        return ;
        push_down(r);
        InOrder(ch[r][0]);
        printf("%d ",key[r]);
        InOrder(ch[r][1]);
        return ;
}
int main(){
        //    freopen("in.txt", "r", stdin);
        //    freopen("out.txt", "w", stdout);
        while (scanf("%d%d", &n, &q) == 2){
                Init();
                char op[20];
                int x, y, z;
                while (q--){
                        scanf("%s", op);
                        if (strcmp(op, "INSERT") == 0){
                                scanf("%d%d", &x, &y);
                                Insert(x, y);
                        }
                        else if (strcmp(op, "DELETE") == 0){
                                scanf("%d%d", &x, &y);
                                Delete(x,y);
                        }
                        else if (strcmp(op, "MAKE-SAME") == 0){
                                scanf("%d%d%d", &x, &y, &z);
                                Make_Same(x, y, z);
```

```
            }
            else if (strcmp(op, "REVERSE") == 0){
                scanf("%d%d", &x, &y);
                Reverse(x, y);
            }
            else if (strcmp(op, "GET-SUM") == 0){
                scanf("%d%d", &x, &y);
                printf("%d\n", Get_Sum(x, y));
            }
            else if (strcmp(op, "MAX-SUM") == 0){
                printf("%d\n", Get_MaxSum(1, size[root] - 2));
            }
        }
    }
    return 0;
}
```

3、树状数组(全功能)

```
#define Max 500010
int N;
int data[Max] = {0};
int C[Max] = {0}; //差分数组
int C2[Max] = {0}; // C2[i] = (i-1)*C[i]
int BIT[Max] = {0};
int BIT1[Max] = {0};
int BIT2[Max] = {0};
int lowbit(int x){ return (x)&(-x);}
int getsum(int x){ //  数据数组求和
    int sum = 0;
    for(;x > 0;x-=lowbit(x)){    sum+=BIT[x];   }
    return sum;
}
int getsum1(int x){ //  差分数组求和
    int sum = 0;
    for(;x > 0;x-=lowbit(x)){      sum+=BIT1[x]; }
    return sum;
}
int getsum2(int x){
    int sum = 0;
    for(;x > 0;x-=lowbit(x)){      sum+=BIT2[x]; }
    return sum;
}
void add(int i,int add){ //  数据 BIT 更新
```

```
    for (;i <= N;i+=lowbit(i)){      BIT[i]+=add; }
}
void add1(int i,int add){//  差分 BIT 更新
    for(;i <= N;i+=lowbit(i)){       BIT1[i]+=add;}
}
void add2(int i,int add){
    for(;i <= N;i+=lowbit(i)){      BIT2[i]+=add;}
}
int main(){
    //输入数据到 data , 更新 BIT 数组
    /*  区间修改, 单点查询
      * C[i] = data[i]-data[i-1]
      * add1(i,C[i])
      *  第 x 个数为 getsum1(x)*/
    /*区间修改, 区间查询
      * C2[i] = (i-1)*C[i]
      * add2(i,C2[i])
      *    前 n 项和为 n*getsum1(n)-getsum2(n)*/
    /*cin >> N;
       for(int i = 1;i <= N;i++){     输入数据
          cin >> data[i];
          C[i] = data[i]-data[i-1];
          C2[i] = (i-1)*C[i];
          add1(i,C[i]);
          add2(i,C2[i]);
    }
    int l,r;
    cin >> l >> r;修改区间
    int v;
    cin >> v;
    C[l] +=v;
    C[r+1] -=v;
    add1(l,v);
    add1(r+1,-v);
    int x;
    cin >> x;  查询单点
    cout << getsum1(x) << endl;
    C2[l] += v*(l-1);
    C2[r+1] += (-v)*(r);
    add2(l,v*(l-1));
    add2(r+1,(-v)*r);
    cin >> l >> r;  查询区间
    cout << r*getsum1(r)-getsum2(r)-((l-1)*getsum1(l-1)-getsum2(l-1)) << endl; */
}
```

4、线段树

```
//tree[]表示每个区间的最小值,sum[]区间和,col[]打标记,表明该区间每个元素增加了多少
//懒标记用在 update 上，意思是可以不用都更新，用到的时候再更新
void build(int node,int begin,int end){
    if(begin==end){
        tree[node]=array[begin];
        sum[node]=array[begin];
        return;
    }
    int mid=begin+(end-begin)
    build(2*node,begin,mid);
    build(2*node+1,mid+1,end);
    //用 tree 记录每个区间的最值
    if(tree[node*2]<=tree[node*2+1]) tree[node]=tree[2*node];
    else    tree[node]=tree[2*node+1];
}
//区间查询
int query1(int node,int begin,int end,int left,int right){//查询 3-5 之间的最小值
    int p1,p2;
    if(left>end || right<begin)    return -1;
    if(begin>=right && end<=right)      return tree[node];
    int mid=begin+(end-begin)/2;
    p1=query(2*node,begin,mid);
    p2=query(2*node+1,mid+1,end);
    if(p1==-1)return p2;
    if(p2==-1)return p1;
    if(p1<=p2)return p1;
    else return p2;
}
//单点更新
void updata(int node ,int begin,int end,int ind,int add){//将哪个点 ind 更新了  add
    if(begin==end)    tree[node]+=add;
    int mid=begin+(end-begin)/2;
    if(ind<=m)    updata(node*2,left,mid,ind,add);
    else      updata(node*2+1,mid+1,right,ind,add);
    tree[node]=min(tree[node*2,node*2+1]);
}
//下放标记
void pushdown(int rt,int m){
    if(col[rt]){
        col[rt<<1]=col[rt];
        col[rt<<1|1]=col[rt];
        sum[rt<<1]=col[rt]*(m-m/2);//这个区间要稍微大一点
```

```
            sum[rt<<1|1]=col[rt]*(m/2);
            col[rt]=0;
        }
}
//区间修改
//以修改区间的值并求整个数组的和为例
void change(int node,int begin,int end,int left,int right,int c){
        if(left<=begin && end >=right){
            col[node]=c;//在该节点上打一个标记 标记打给谁了，就一个么？
            sum[node]=c*(end-begin+1); //做和的时候用了，但是查询的时候不是这个区间了，
这时就用到标记下放
            return;
        }
        pushdown(node,end-begin+1);
        int mid=begin+(end-begin)/2;
        if(left<=mid) change(node*2,begin,end,left,mid,c) ;
        if(right>mid) change(node*2+1,begin,end,mid+1,right,c);
        sum[node]=sum[node*2]+sum[node*2+1];
}
//查询区间之和
int query2(int node,int begin,int end,int left,int right){
        if(left<=begin && end>=right)    return sum[node];     //不需要就不下放了
        Pushdown(node,end-begin+1);//需要的时候再下放
        int ret=0;
        int mid=begin+(end-begin)/2;
        if(left<mid) ret+=query2(node*2,begin,end,left,mid,c) ;
        if(right>mid) ret+=query2(node*2+1,begin,end,mid+1,right,c);
        return ret;
}
```

5、主席树,静态区间第 k 小

```
const int MAXN = 100010;
const int M = MAXN * 30;
int n, q, m, tot;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M], c[M];
void Init_hash(){
        for (int i = 1; i <= n; i++)      t[i] = a[i];
        sort(t + 1, t + 1 + n);
        m = (int)(unique(t + 1, t + 1 + n) - t - 1);
}
int build(int l, int r){
        int root = tot++; c[root] = 0;
```

```
        if (l != r){
            int mid = (l + r) >> 1;
            lson[root] = build(l, mid);
            rson[root] = build(mid + 1, r);
        }
        return root;
}
int hash_(int x){
        return (int)(lower_bound(t + 1, t + 1 + m, x) - t);
}
int update(int root, int pos, int val){
        int newroot = tot++, tmp = newroot;
        c[newroot] = c[root] + val;
        int l = 1, r = m;
        while (l < r){
            int mid = (l + r) >> 1;
            if (pos <= mid){
                lson[newroot] = tot++;
                rson[newroot] = rson[root];
                newroot = lson[newroot];
                root = lson[root];
                r = mid;
            }
            else{
                rson[newroot] = tot++;
                lson[newroot] = lson[root];
                newroot = rson[newroot];
                root = rson[root];
                l = mid + 1;
            }
            c[newroot] = c[root] + val;
        }
        return tmp;
}
int query(int left_root, int right_root, int k){
        int l = 1, r = m;
        while ( l < r){
            int mid = (l + r) >> 1;
            if (c[lson[left_root]] - c[lson[right_root]] >= k ){
                r = mid;
                left_root = lson[left_root];
                right_root = lson[right_root];
            }
            else{
```

```
                l = mid + 1;
                k -= c[lson[left_root]] - c[lson[right_root]];
                left_root = rson[left_root];
                right_root = rson[right_root];
            }
        }
        return l;
}
int main(){
        //    freopen("in.txt","r",stdin);
        //    freopen("out.txt","w",stdout);
        while (scanf("%d%d", &n, &q) == 2){
            tot = 0;
            for (int i = 1; i <= n; i++)        scanf("%d", &a[i]);
            Init_hash();
            T[n + 1] = build(1, m);
            for (int i = n; i; i--){
                int pos = hash_(a[i]);
                T[i] = update(T[i + 1], pos, 1);
            }
            while (q--){
                int l, r, k;
                scanf("%d%d%d", &l, &r, &k);
                printf("%d\n", t[query(T[l], T[r + 1], k)]);
            }
        }
        return 0;
}
```

6、主席树,区间有多少不重复的数

```
/*给出一个序列,查询区间内有多少个不相同的数  */
const int MAXN = 30010;
const int M = MAXN * 100;
int n, q, tot;
int a[MAXN];
int T[MAXN], lson[M], rson[M], c[M];
int build(int l, int r){
        int root = tot++;
        c[root] = 0;
        if (l != r){
            int mid = (l + r) >> 1;
            lson[root] = build(l, mid);
            rson[root] = build(mid + 1, r);
```

```
        }
        return root;
}
int update(int root, int pos, int val){
        int newroot = tot++, tmp = newroot;
        c[newroot] = c[root] + val;
        int l = 1, r = n;
        while (l < r){
                int mid = (l + r) >> 1;
                if (pos <= mid){
                        lson[newroot] = tot++;
                        rson[newroot] = rson[root];
                        newroot = lson[newroot];
                        root = lson[root];
                        r = mid;
                }
                else{
                        rson[newroot] = tot++;
                        lson[newroot] = lson[root];
                        newroot = rson[newroot];
                        root = rson[root];
                        l = mid + 1;
                }
                c[newroot] = c[root] + val;
        }
        return tmp;
}
int query(int root, int pos){
        int ret = 0;
        int l = 1, r = n;
        while (pos < r){
                int mid = (l + r) >> 1;
                if (pos <= mid){
                        r = mid;
                        root = lson[root];
                }
                else{
                        ret += c[lson[root]];
                        root = rson[root];
                        l = mid + 1;
                }
        }
        return ret + c[root];
}
```

```
int main(){
    //  freopen("in.txt", "r", stdin);
    //  freopen("out.txt", "w", stdout);
    while (scanf("%d", &n) == 1){
        tot = 0;
        for (int i = 1; i <= n; i++)    scanf("%d", &a[i]);
        T[n + 1] = build(1, n);//T 为树
        map<int,int> mp;
        for (int i = n; i >= 1; i--){
            if (mp.find(a[i]) == mp.end()) T[i] = update(T[i + 1], i, 1);
            else{
                int tmp = update(T[i + 1], mp[a[i]], -1);
                T[i] = update(tmp, i, 1);
            }
            mp[a[i]] = i;
        }
        scanf("%d", &q);
        while (q--){
            int l, r;
            scanf("%d%d", &l, &r);
            printf("%d\n", query(T[l], r));
        }
    }
    return 0;
}
```

7、主席树+树状数组,动态区间第 k 大

```
/*树状数组套主席树*/
const int MAXN = 60010;
const int M = 2500010;
int n, q, m, tot;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M],c[M];
int S[MAXN];
struct Query{
    int kind;
    int l, r, k;
} query[10010];
void Init_hash(int k){
    sort(t, t + k);
    m = (int)(unique(t, t + k) - t);
    return ;
}
```

```
int hash_(int x){
	return (int)(lower_bound(t, t + m, x) - t);
}
int build(int l, int r){
	int root = tot++;
	c[root] = 0;
	if (l != r){
		int mid = (l + r) / 2;
		lson[root] = build(l, mid);
		rson[root] = build(mid + 1, r);
	}
	return root;
}
int Insert(int root, int pos, int val){
	int newroot = tot++, tmp = newroot;
	int l = 0, r = m - 1;
	c[newroot] = c[root] + val;
	while (l < r){
		int mid = (l + r) >> 1;
		if (pos <= mid){
			lson[newroot] = tot++;
			rson[newroot] = rson[root];
			newroot = lson[newroot];
			root = lson[root];
			r = mid;
		}
		else{
			rson[newroot] = tot++;
			lson[newroot] = lson[root];
			newroot = rson[newroot];
			root = rson[root];
			l = mid + 1;
		}
		c[newroot] = c[root] + val;
	}
	return tmp;
}
int lowbit(int x){    return x & (-x);}
int use[MAXN];
void add(int x, int pos, int val){
	while (x <= n){
		S[x] = Insert(S[x], pos, val);
		x += lowbit(x);
	}
```

```
    return ;
}
int sum(int x){
    int ret = 0;
    while (x > 0){
        ret += c[lson[use[x]]];
        x -= lowbit(x);
    }
    return ret;
}
int Query(int left, int right, int k){
    int left_root = T[left - 1];
    int right_root = T[right];
    int l = 0, r = m - 1;
    for (int i = left - 1; i; i -= lowbit(i))      use[i] = S[i];
    for (int i = right; i; i -= lowbit(i))        use[i] = S[i];
    while (l < r){
        int mid = (l + r) / 2;
        int tmp = sum(right) - sum(left - 1) + c[lson[right_root]] - c[lson[left_root]];
        if (tmp >= k){
            r = mid;
            for (int i = left - 1; i; i -= lowbit(i))     use[i] = lson[use[i]];
            for (int i = right; i; i -= lowbit(i))     use[i] = lson[use[i]];
            left_root = lson[left_root];
            right_root = lson[right_root];
        }
        else{
            l = mid + 1;
            k -= tmp;
            for (int i = left - 1; i; i -= lowbit(i))     use[i] = rson[use[i]];
            for (int i = right; i; i -= lowbit(i))     use[i] = rson[use[i]];
            left_root = rson[left_root];
            right_root = rson[right_root];
        }
    }
    return l;
}
void Modify(int x, int p, int d){
    while (x <= n){
        S[x] = Insert(S[x], p, d);
        x += lowbit(x);
    }
    return ;
}
```

```
int main(){
    //    freopen("in.txt", "r", stdin);
    //    freopen("out.txt", "w", stdout);
    int Tcase;
    scanf("%d", &Tcase);
    while (Tcase--){
        scanf("%d%d", &n, &q);
        tot = 0;
        m = 0;
        for (int i = 1; i <= n; i++){
            scanf("%d", &a[i]);
            t[m++] = a[i];
        }
        char op[10];
        for (int i = 0; i < q; i++){
            scanf("%s", op);
            if (op[0] == 'Q'){
                query[i].kind = 0;
                scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
            }
            else{
                query[i].kind = 1;
                scanf("%d%d", &query[i].l, &query[i].r);
                t[m++] = query[i].r;
            }
        }
        Init_hash(m);
        T[0] = build(0, m - 1);
        for (int i = 1; i <= n; i++)    T[i] = Insert(T[i - 1], hash_(a[i]), 1);
        for (int i = 1; i <= n; i++)      S[i] = T[0];
        for (int i = 0; i < q; i++){
            if (query[i].kind == 0){
                printf("%d\n", t[Query(query[i].l, query[i].r, query[i].k)]);
            }
            else{
                Modify(query[i].l, hash_(a[query[i].l]), -1);
                Modify(query[i].l, hash_(query[i].r), 1);
                a[query[i].l] = query[i].r;
            }
        }
    }
    return 0;
}
```

8、左偏树 ,小堆的合并,有序序列合并

```
/*    合并复杂度  O(log N)
  *    INIT: init()读入数据并进行初始化;
  *    CALL: merge()  合并两棵左偏树;
  *              ins()  插入一个新节点;
  *              top()  取得最小结点;
  *              pop()  取得并删除最小结点;
  *              del()  删除某结点;
  *              add()  增/减一个结点的键值;
  *              iroot()  获取结点 i 的根;*/
#define typec int              //    type of key val
const int na = -1;
const int N = 1010;
struct node{
      typec key;
      int l, r, f, dist;
} tr[N];
int iroot(int i){      //    find i's root
      if (i == na)      return i;
      while (tr[i].f != na){      i = tr[i].f;}
      return i;
}
int merge(int rx, int ry){          //    two root:      rx, ry
      if (rx == na)      return ry;
      if (ry == na)      return rx;
      if (tr[rx].key > tr[ry].key)      swap(rx, ry);
      int r = merge(tr[rx].r, ry);
      tr[rx].r = r;
      tr[r].f = rx;
      if (tr[r].dist > tr[tr[rx].l].dist)        swap(tr[rx].l, tr[rx].r);
      if (tr[rx].r == na)    tr[rx].dist = 0;
      else        tr[rx].dist = tr[tr[rx].r].dist + 1;
      return rx;    //    return new root
}
int ins(int i, typec key, int root)
{     //    add a new node(i, key)
      tr[i].key = key;
      tr[i].l = tr[i].r = tr[i].f = na;
      tr[i].dist = 0;
      return root = merge(root, i);      //    return new root
}
int del(int i)
{     //    delete node i
```

```
        if (i == na)        return i;
        int x, y, l, r;
        l = tr[i].l;
        r = tr[i].r;
        y = tr[i].f;
        tr[i].l = tr[i].r = tr[i].f = na;
        tr[x = merge(l, r)].f = y;
        if (y != na && tr[y].l == i)      tr[y].l = x;
        if (y != na && tr[y].r == i)        tr[y].r = x;
        for (; y != na; x = y, y = tr[y].f){
                if (tr[tr[y].l].dist < tr[tr[y].r].dist)    swap(tr[y].l, tr[y].r);
                if (tr[tr[y].r].dist + 1 == tr[y].dist)      break;
                tr[y].dist = tr[tr[y].r].dist + 1;
        }
        if (x != na)        return iroot(x);    //    return new root
        else return iroot(y);
}
node top(int root){      return tr[root];}
node pop(int &root){
        node out = tr[root];
        int l = tr[root].l, r = tr[root].r;
        tr[root].l = tr[root].r = tr[root].f = na;
        tr[l].f = tr[r].f = na;
        root = merge(l, r);
        return out;
}
int add(int i, typec val){      //    tr[i].key += val
        if (i == na)      return i;
        if (tr[i].l == na && tr[i].r == na && tr[i].f == na){
                tr[i].key += val;
                return i;
        }
        typec key = tr[i].key + val;
        int rt = del(i);
        return ins(i, key, rt);
}
void init(int n){
        for (int i = 1; i <= n; i++){
                scanf("%d", &tr[i].key);        //    %d: type of key
                tr[i].l = tr[i].r = tr[i].f = na;
                tr[i].dist = 0;
        }
        return ;
}
```

9、背包相关

```
int dp[SIZE];
int volume[MAXN], value[MAXN], c[MAXN];
int n, v;                    //   总物品数，背包容量
//   01 背包
void ZeroOnepark(int val, int vol){
      for (int j = v ; j >= vol; j--){
            dp[j] = max(dp[j], dp[j - vol] + val);
      }
}
//   完全背包
void Completepark(int val, int vol){
      for (int j = vol; j <= v; j++){
            dp[j] = max(dp[j], dp[j - vol] + val);
      }
}
//   多重背包
void Multiplepark(int val, int vol, int amount){
      if (vol * amount >= v)      Completepark(val, vol);
      else{
            int k = 1;
            while (k < amount){
                  ZeroOnepark(k * val, k * vol);
                  amount -= k;
                  k <<= 1;
            }
            if (amount > 0){
                  ZeroOnepark(amount * val, amount * vol);
            }
      }
}
int main(){
      while (cin >> n >> v){
            for (int i = 1 ; i <= n ; i++){
                  cin >> volume[i] >> value[i] >> c[i];          // 费用，价值，数量
            }
            memset(dp, 0, sizeof(dp));
            for (int i = 1; i <= n; i++){
                  Multiplepark(value[i], volume[i], c[i]);
            }
            cout << dp[v] << endl;
      }
      return 0;}
```

10、并查集

```
/*    INIT: makeset(n);
  *    CALL: findset(x); unin(x, y);*/
const int N = 1010;
struct lset{
      int p[N], rank[N], sz;
      void link(int x, int y){
            if (x == y)      return ;
            if (rank[x] > rank[y])      p[y] = x;
            else        p[x] = y;
            if (rank[x] == rank[y])      rank[y]++;
            return ;
      }
      void makeset(int n){
            sz = n;
            for (int i = 0; i < sz; i++){
                  p[i] = i;
                  rank[i] = 0;
            }
            return ;
      }
      int findset(int x){
            if (x != p[x])        p[x] = findset(p[x]);
            return p[x];
      }
      void unin(int x, int y){
            link(findset(x), findset(y));
            return ;
      }
      void compress(){
            for (int i = 0; i < sz; i++)        findset(i);
            return ;
      }
};
```

11、使序列有序的最小交换次数

```
int IT_MAX = 1 << 19;
int MOD = 1000000007;
const int INF = 0x3f3f3f3f;
const ll LL_INF = 0x3f3f3f3f3f3f3f3f;
const db PI = acos(-1);
const db ERR = 1e-10;
const int MAX_N = 100005;
bool cmp (int a , int b){
/*交换任意两数的本质是改变了元素位置，
 *  故建立元素与其目标状态应放置位置的映射关系*/
int getMinSwaps(vector<int> &A){     //排序
    vector<int> B(A);
    sort(B.begin(), B.end());
    map<int, int> m;
    int len = (int)A.size();
    for (int i = 0; i < len; i++)       m[B[i]] = i; //建立每个元素与其应放位置的映射关系
    int loops = 0;          //  循环节个数
    vector<bool> flag(len, false);
    //  找出循环节的个数
    for (int i = 0; i < len; i++){
        if (!flag[i]){
            int j = i;
            while (!flag[j]){
                flag[j] = true;
                j = m[A[j]];       //  原序列中 j 位置的元素在有序序列中的位置
            }
            loops++;
        }
    }
    return len - loops;
}
vector<int> nums;
int main(){
    nums.push_back(1);
    nums.push_back(2);
    nums.push_back(4);
    nums.push_back(3);
    nums.push_back(5);
    int res = getMinSwaps(nums);
    cout << res << '\n';
    return 0;
}
```

12、使序列有序的最小交换区间次数

```
/*  默认目标映射关系是 key 1 => val 1 …… key n => val n
 *  如果序列不是 1~n 可以通过 map 建立新的目标映射关系
 *  交换任意区间的本质是改变了元素的后继，故建立元素与其初始状态后继的映射关系*/
const int MAXN = 30;
int n;
int vis[MAXN];
int A[MAXN], B[MAXN];
int getMinSwaps(){
    memset(vis, 0, sizeof(vis));
    for (int i = 1; i <= n; i++)      B[A[i]] = A[i % n + 1];
    for (int i = 1; i <= n; i++)    B[i] = (B[i] - 2 + n) % n + 1;
    int cnt = n;
    for (int i = 1; i <= n; i++){
        if (vis[i])      continue;
        vis[i] = 1;
        cnt--;
        for (int j = B[i]; j != i; j = B[j])      vis[j] = 1;
    }
    return cnt;
}
int main(){
    cin >> n;
    for (int i = 1; i <= n; i++)    cin >> A[i];
    int res = getMinSwaps();
    cout << res << '\n';
    return 0;
}
```