ST 表

一、RMQ 问题

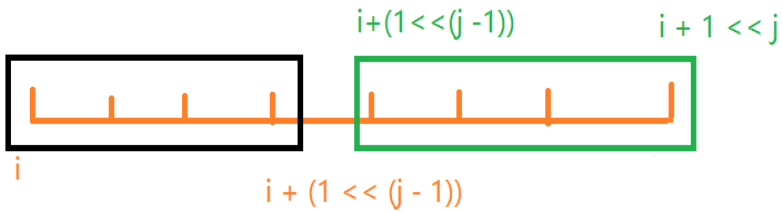
RMQ（Range Minimum/Maximum Query），即区间最值查询 。是指这样一个问题：对于长度为 n 的数列 a，回答若干询问 RMQ（A,i,j）(i, j<=n)，返回数列 a 中下标在 i，j 之间的最小/大值。

二、ST 表

ST（Sparse Table）算法是一个非常有名的在线处理 RMQ 问题的算法，它可以在 O(nlogn)时间内进行预处理，然后在 O(1)时间内回答每个查询。

ST 表预处理采取了动态规划、倍增的思想，设 f[i][j]表示从 i 点开始 2^j 个长度范围内的最值。可以得到以下的状态转移方程：

```
f[i][j] = min(f[i][j - 1], f[i + (1 << (j - 1))][j - 1])
```



对于查询操作，有两种方式，首先介绍 O(1)的查询方式。设查询区间为[l,r]，则该区间长度为 r-l+1，找到一个 k 值，使得 2^k 刚好小于 r-l+1，于是答案即为：

```
min(f[l][k],f[r - (1 << k) + 1][k])
```

显然，k=log(r-l+1)。

另外一种查询方式是利用倍增原理，通过拼凑的方式组合出区间答案。如长度为 13 的区间，可以先区前面长度 8 的区间，然后是长度 4，长度 1。根据的是 13 的二进制位上的每一位是否有 1，参考代码：

```
int rmq(int l,int r){
    int k=r-l+1;
    int ans=-1;
    for(int i=19;i>=0;i--)
        if(k&(1<<i)){
            ans=max(ans,f[l][i]);
            l=l+(1<<i) ;
        }
    return ans;
}
```

三、实例

例 1：A2100. 「模板」ST 表

题意：长度为 N 的整数序列 A ，有 Q 次询问，每次询问区间 [l,r]最大值。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e6+10;
```

```
int n,q,l,r;
int a[N];
int f[N][20];
int rmq(int l,int r){
    int k=log2(r-l+1);
    return max(f[l][k],f[r-(1<<k)+1][k]);
}
int main(){
    scanf("%d",&n);
    for(int i=0;i<=n;i++){
        scanf("%d",&a[i]);
        f[i][0]=a[i];
    }
    for(int j=1;(1<<j)<=n;j++)
        for(int i=0;i<=n;i++)
            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
    scanf("%d",&q);
    for(int i=1;i<=q;i++){
        scanf("%d%d",&l,&r);
        cout<<rmq(l,r)<<"\n";
    }
    return 0;
}
```

例 2：与众不同 P2586

题意：给定一个长度为 n 的序列，m 个询问,每次询问一个区间 [L,R]内最长的没有重复数字的子序列,n 以及 m <= 2 * 10^5

【解析】

方法一:简单暴力,枚举 [L,R]区间内的子序列,然后 O(len) 扫一遍,总的时间复杂度是 O(n^3 * m)

方法二:

二分最长的长度，对于判断，每一次枚举左端点，那么右端点便是固定的，然后借助线段树 O(n log^2(n) * m)

方法三:

首先，记录每一个元素的前一个与它相同的元素的位置是哪一个。

我们发现，对于一个左端点来说，其最长的，没有重复数字的子序列的长度是固定的。

固定左端点后，我们总是希望不停的往右边挪动右端点以求能够有一个最长的子序列。

实际上，我们对于每一个左端点，不停移动右端点(这个右端点只会增加而不会减小)，总的时间复杂度 O(n)。

对于每一个点 i，当它为左端点时不妨设其最长没有重复元素的子段的右端点为:r[i]

同时，我们观察到，得到的 RR 序列是单调不降的，于是就方便处理，用二分.

对于一个查询怎么办?

区间 [L,R] 之间的任意一个点作为左端点

Case 1 左端点对应的最长的没有重复元素的子序列的右端点大于 R，

显然这些左端点只有最左边的那个是"有用"的,同时因为得到的 r[]是单调不降的，我们很容易用二分找到这一个最左边的 r[i]大于 R 的点。

Case2 左端点对应的最长的没有重复元素的子序列的右端点小于等于 R 的，
取最大的 r[i]-i （预处理出 r[i]-i),ST 表即可。
比较上下两种情况即可
总的时间复杂度为:O((m + n)logn)

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2e5+10,M=1e6+10;
int last[M*2],dp[N][33],q[N],st[N],Log[N];
int n,m,l,r,x;
#define in read()
int read(){
    int s=0,f=1;
    char c=getchar();
    while(c>'9' || c<'0'){if(c=='-') f=-1;c=getchar();}
    while(c<='9' && c>='0'){s=(s<<1)+(s<<3)+c-'0';c=getchar();}
    return s*f;
}
void inti(){
    for(int i=1;i<=n;i++)dp[i][0]=q[i];
    for(int j=1;j<=20;j++)
        for(int i=1;i+(1<<j)-1 <=n ; i++)
            dp[i][j]=max(dp[i][j-1] , dp[i+(1<<(j-1))][j-1]);
}
int query(int l,int r){
    int k=Log[r-l+1];
    return max(dp[l][k] ,dp[r-(1<<k)+1][k]);
}
int search(int l,int r){
    if(st[l] == r)return l;
    if(st[r] < l) return r+1;
    int le=l,ri=r;
    while(le<ri){
        int mid=(le+ri)>>1;
        if(st[mid]<l)
            le=mid+1;
        else
            ri=mid;
    }
    return le;
}
```

```
int main(){
    n=in;m=in;
    Log[0]=1;
    for(int i=2;i<=n+10;i++)Log[i]=Log[i>>1]+1;
    for(int i=1;i<=n;i++){
        x=in;
        st[i]=max(st[i-1] , last[x+M] + 1);
        q[i] = i - st[i] +1;
        last[x+M] = i;
    }
    inti();
    while(m--){
        l=in;r=in;
        l++;r++;
        int mid=search(l,r),ans=0;
        if(mid>l)ans=mid-l;
        if(mid<=r)    {
            ans=max(ans,query(mid,r));
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

例 3：超级钢琴 P2613

题意：一个长度为 n 的数字串，选取 k 组长度在 l~r 范围内的子串，是的总和最大

【解析】

我们先考虑较暴力的写法，我们枚举每个点做为左端点，那么每个点有 L~R 种情况，最坏 N^2,然后要在这些里面取 K 个最大的，最暴力做法是排序。最坏 N^2 logN

而这道题，只要前 K 个大的数，有点类似做过的一道堆的题。

我们先处理出这样一个三元组（i，L，R）表示当这个和弦的左端点为 i 时在上限和下限中的最优值。（也就是右端点在[i+l-1，i+r-1]中）

如图，起点为 i 对于的区间在 L,R 之间，最优的是 id 这个位置，那么把这个位置加入答案后，再把剩下的区间[L~ID-1]　[ID+1~R]的最优加入堆中

我们对于第一步的处理，其实有两种方法：

首先我们都需要处理出前缀和这个东西来，然后就是查询区间最值，查完后再加上 i 到 L 的值就好了。对于查区间最值：

①：我们可以用线段树。但是我们的查询次数是（n+k），所以会比较慢。

②：我们可以用 ST，因为数列前缀和是不修改的，用 ST 就可以做到 O（1）的查询，会快很多。

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

```
#define ll long long
#define N 500010
const int INF=1e9;
inline void read(int &x){
    x=0;int f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
    while(c>='0'&&c<='9'){x=x*10+c-'0';c=getchar();}
    x*=f;
}
struct Node{
    int maxn,id,i,l,r;
    bool operator<(const Node&t)const{
        return maxn<t.maxn;
    }
};
priority_queue<Node>q;
int a[N]={},n,K,L,R;
ll ans;
struct T{
    int maxn,id;
    bool operator<(const T&t)const{
        return maxn<t.maxn;
    }
}f[N][20];//f[i][j]表示 i 为起点长度为 2^j 的区间的最右值前缀和大小和位置

void pre(){
    for(int j=1;(1<<j)<=n;j++){
        for(int i=1;i+(1<<j)-1<=n ; i++){
            f[i][j]=max(f[i][j-1],f[i+(1<<j-1)][j-1]);
        }
    }
}
T query(int l,int r){
    T ret;
    int k=0;
    while((1<<k+1) < r-l+1 )k++;
    ret=max(f[l][k],f[r-(1<<k)+1][k]);
    return ret;
}
int main(){
    read(n);read(K);read(L);read(R);
    T t;
    for(int i=1;i<=n;i++){
        read(a[i]);
```

```
        a[i]+=a[i-1];
        f[i][0].maxn=a[i];//
        f[i][0].id=i;
    }
    pre();
    for(int i=1;i<=n-L+1;i++){
        int l=i+L-1,r=min(i+R-1,n);
        t=query(l,r);
        q.push((Node){t.maxn-a[i-1], t.id ,i,l,r});
        //把区间和，位置，左端点 i，范围 lr 压人堆
    }
    while(K--){
        Node tmp=q.top();q.pop();
        ans+=(ll)tmp.maxn;
        if(tmp.id-1>=tmp.l){//
            t=query(tmp.l, tmp.id-1);
            Node x=(Node){t.maxn-a[ tmp.i-1], t.id ,tmp.i,tmp.l,tmp.id-1};
            q.push(x);
        }
        if(tmp.id+1<=tmp.r){//
            t=query(tmp.id+1, tmp.r);
            Node x=(Node){t.maxn-a[tmp.i-1], t.id ,tmp.i,tmp.id+1,tmp.r};
            q.push(x);
        }
    }
    cout<<ans;
    return 0;
}
```

例 4：萌萌哒 H1109

题意：长度为 n 的数字，m 个限制（l1,r1,l2,r2）对于每个限制满足 S(L1,R1)=S(L2,R2)的 s 数量，m，n<=100000

【解析】

知识点：ST 表，并查集

考虑暴力：对于一个区间与另外一个区间数字相同，则两个区间的每个数字进行合并集合，最后看有多少个集合即可 。

Ans=9*10^(cnt-1)(第一位考虑不能为 0)。

暴力的时间复杂度主要在于每个区间合并时需要合并里面每个数字。分析发现两个区间合并时，可以先不急下放到每个数字，因为区间具备合并性。 所有区间处理完成后再下放合并即可 。

采用倍增法来维护整个过程 。

F[i][j]表示 i 为起点长为 2^j 的区间。将 f[i][j]当做并查集上的一个点 。

设计算法：

1、初始化 f[i][j]（给每个 f[i][j]编号 cnt）记录 cnt 对应的起点

2、对于每个限制条件，二进制拆分区间（从大到小）进行合并（启发式：大编号向小编号合并）

3、将 f[i][j]从长开始下放合并到 f[i][0]

4、统计一共有多少个集合，统计答案

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define mod 1000000007
#define N 100004
int n,m,l1,r1,l2,r2;
int cnt=0,fa[N*18],st[N][20],sta[N*18];//记录左端点
int find(int x){
    if(x==fa[x])return x;
    return fa[x]=find(fa[x]);
}
void merge(int l1,int l2,int k){
    int f1=find(st[l1][k]),f2=find(st[l2][k]);
    if(f1>f2)swap(f1,f2);//因 sta 一定要大的给小的
    fa[f2]=f1;
}
void get(int &x){
    char c = getchar(); x = 0;
    while(c < '0' || c > '9') c = getchar();
    while(c <= '9' && c >= '0') x = x*10+c-48, c = getchar();
}
int main(){
    get(n);get(m);
    if(n==1){
        printf("10");
        return 0;
    }
    for(int j=0;j<=18;j++)
        for(int i=1;i+(1<<j)-1<=n;i++)
            st[i][j]=++cnt,sta[cnt]=i,fa[cnt]=cnt;
    //记录
    for(int i=1;i<=m;i++){
        get(l1);get(r1);get(l2);get(r2);
        for(int j=18;j>=0;j--){
            if(l1+(1<<j)-1<=r1){
                merge(l1,l2,j);
                l1+=(1<<j);l2+=(1<<j);
            }
        }
    }
    //向下传递至 st[i][0]
    for(int j=18;j>=1;j--)
        for(int i=1;i+(1<<j)-1<=n;i++){
```

```
            //如果任意 st[s,t]和 st[i,j]属于同一集合
            // 那 么 st[s,t-1] 与 st[i,j-1] 以 及 st[s+2^(t-1)-1,t-1] 和
st[i+2^(j-1)-1,j-1]都应该属于同一集合
            int f=find(st[i][j]),s=sta[f];
            if(s==i)continue;
            merge(i,s,j-1);
            merge(i+(1<<j-1),s+(1<<j-1),j-1);
        }
    //统计 答案是 9*10^(集合个数-1)
    int ans=9;
    for(int i=2;i<=n;i++)//1 定为集合的根
        if(fa[st[i][0]]==st[i][0])ans=(1LL*ans*10)%mod;
    printf("%d",ans);
    return 0;
}
```