# 基础数据结构

## 单调队列

内部元素有序的队列，有序性一般依靠题目的单调性维护。

- 模板：滑动窗口
    - 描述：给出一数列 $a_n$ ，对每个 $i \in [1,n]$ ，求 $[i,i+k]$ 的最大值，$k$ 为常数。
    - 将 $[1,k]$ 中的元素放入队列，若队列中存在 $a_i \le a_{i+j}$，则 $a_i$ 一定不是 $[1,k]$ 的唯一最大值，也不可能是之后任何一个区间的唯一最大值，我们将其删除。最终得到的队列一定是单调的，队首即为答案。然后我们删除小于 $a_{k+1}$ 的元素，将 $a_{k+1}$ 加入，删除$a_1$（如果还存在的话），以此类推。
- [NOIP2016蚯蚓]
    - 题目描述：$n$ 只蚯蚓，每次选最长的截成长度比为 $u:v$ 的两段（取整），同时其他蚯蚓增加长度 $q$ ，求 $m$ 天内每次被切的蚯蚓的长度，以及最终每条蚯蚓的长度。$n=1e5,m=7e6$
    - 其他蚯蚓增加长度 $q$ ，相当于被切的蚯蚓减少 $q$ ，然后所有蚯蚓加 $q$ 。

      直接使用堆存储蚯蚓，每次取堆顶截断后放回堆里即可。复杂度 $O(m\log(n+m))$ ，肯定过不去，但有 85 pts。
    - 正解：先产生的小蚯蚓，一定比后产生的长。所以可以用单调队列维护。

      具体地，将初始的蚯蚓排序得到 q1，新蚯蚓较大的一条按先后顺序放入q2，较小的放入q3，则三个队列都单调，队首即为极值。
- [P2627]
    - 给定数列 $a_n$ 从中选择一些数，且不能连续选择超过 $K$ 个，求选择数的和的最大值。
    - $f_i$ 表示按要求从前 $i$ 个数中选择的最大方案，则 $f_i=\max\limits_{i-K\le j\le i}\{sum_i-sum_j+f_{j-1}\}$

      即 $f_i=sum_i+\max\limits_{i-K\le j\le i}\{f_{j-1}-sum_j\}$ ，用单调队列维护 $f_{j-1}-sum_j$ 即可。

## 离散化

算是一个前置知识。

当使用树状数组或线段树时，我们有时要用到和数据值域等大的空间，而值域有时是 $1e9$ 甚至更大，空间会不够用。但是数据量其实不可能达到 $1e9$ ，也就是说很多值是无用的。

这时我们可以将数值重新编号，将值域缩回 $n$ 。

比如 1,2,4,7,7,20 $\rightarrow$ 1,2,3,4,4,5 。

写起来也很简单：先将所有数字记录在数组 reg 中，然后对 reg 排序并去重，之后用到某个数字，就在 reg 中二分出这个数字的下标，用下标替代数值计算。

```
sort(a+1,a+n+1);
int cnt=unique(a+1,a+n+1)-a-1;

v=lower_bound(a+1,a+cnt+1,v)-a;
```

# 树状数组

- 单点修改，前缀和查询。

  令第 $i$ 个节点的值表示序列上区间 $[i - lowbit(i) + 1, i]$ 的和。

  则修改单个位置的时候同时要修改包含它的节点。

  事实上不需要真正理解它的含义，背代码就行。

```
void add(int p,int x){
    for(;p<=n;p+=lowbit(p))
        a[p]+=x;
}
```

  前缀和即将前缀拆分为若干形如 $[i - lowbit(i) + 1, i]$ 即可。

```
int sum(int p){
    int ans=0;
    for(;p;p-=lowbit(p))
        ans+=a[p];
    return ans;
}
```

- 区间修改，单点查询。

  考虑差分：$d_i = a_i - a_{i-1}$

  $[l, r]$ 增加 $k$ 相当于 $d_l$ 增加 $k$ ，$d_{r+1}$ 减少 $k$；求 $a_i$ 相当于求 $d_i$ 的前缀和。

  回到第一种情况。

- 插入/删除数据，询问 $[1, r]$ 区间内的数据个数。

  设 $a_i$ 表示数值等于 $i$ 的数据个数。

  插入数据 $x$ 相当于 $a_x$ 增加一，询问区间数据个数相当于求 $a_i$ 前缀和。

- 建立与清空

  - 每个位置有初始值时，不必依次加入，只需每次向后推一位。

    ```
    void init() {
        for (int i=1;i<=n; ++i) {
            t[i] += a[i];
            int j = i + lowbit(i);
            if (j <= n) t[j] += t[i];
        }
    }
    ```

  - 清空树状数组，可以不进行 memset。

    考虑给每个数据标记时间，若数据的时间与当前时间不符，则清除该数据。

```
void add(int p,int x){
    for(;p<=n;p+=lowbit(p)){
        if(tim[p]!=cur)
            tim[p]=cur,a[p]=0;
        a[p]+=x;
    }
}
//sum同理
```

清空就只需 `++cur` 就行。

- 求第 $k$ 小。

  依然是设 $a_i$ 表示数值等于 $i$ 的数据个数。

  则我们需要求出一个最小的 $x$ 满足 $\sum\limits_{i=1}^{x-1} a_i = k-1$ 。

  树状数组中的节点包含的区间长度都是 $2^m$ ，我们考虑从大到小枚举 $m$ ，贪心地考虑每一层能否加入。

```
int kth(int k){
    int cnt=0,ans=0;
    for(int m=log2(n);~m;--m){
        ans+=1<<m;
        if(ans>=n||cnt+a[ans]>=k)
            ans-=1<<m;
        else cnt+=a[ans];
    }
    return ans+1;
}
```

应用：

- 逆序对：求给出一组数据，求 $a_i > a_j, i < j$ 的数对 $(i,j)$ 个数。

  从 $1$ 到 $n$ 考虑，对 $a_i$ ，先查找 $[1, a_i - 1]$ 范围内的数字个数，再插入数据 $a_i$ .

- 给出一长度为 $n$ 的数列 $a_n$ ，求其长度为 $M$ 的严格递增子序列的个数，$M \le n \le 1000$。

  设 $f_{i,j}$ 表示以 $a_j$ 结尾的数列，长度为 $i$ 的严格递增子序列个数。

  则 $f_{i,j} = \sum\limits_{k<j, a_k<a_j} f_{i-1,k}$ ，特殊地，令 $a_0 = -\infty$ .

  需要快速查找 $a_k < a_j$ 的 $f_{i-1,k}$ 之和的数据结构。

  可以使用树状数组，在 $a_k$ 的位置加上 $f_{i-1,k}$ ，然后查询 $[1, a_j)$ 间所有数的和。

```
for(int i=1;i<=m;++i){
    ++cur;//清空
    add(a[0],f[i-1][0]);
    for(int j=1;j<=n;++j){
        f[i][j]=sum(a[j]-1);
        add(a[j],f[i-1][j]);
    }
}
```

- 给出长度为 $n$ 的整数数列 $a_n$ 和整数 $x,y$，求满足 $\sum\limits_{l\le i\le r} a_i \le x+y(r-l+1)$ 的数对 $(l,r)$ 个数。
  - 设 $a_n$ 前缀和为 $s_n$，则 $s_r-s_{l-1}<x+y(r-l+1)$，即 $s_r-yr-x\le s_{l-1}-y(l-1)$。

  先离散化，再求依次将 $s_{l-1}-y(l-1)$ 插入树状数组，查找比 $s_r-yr-x$ 大的数的个数即可。
  - 即求 $\sum\limits_{l\le i\le r}(a_i-y)\le x$，对前者求前缀和得 $t_n$，即 $t_r-x\le t_{l-1}$，将 $t_i$ 插入树状数组，依次查找小于 $t_r-x$ 的数的个数即可。

# 线段树

对每个节点，设其代表区间 $[l,r]$ 的信息。其左儿子代表 $[l,mid]$ 的信息，右儿子代表 $[mid+1,r]$ 的信息。

其中 $mid=\lfloor\frac{(l+r)}{2}\rfloor$，$l=r$ 时该节点为叶节点。

一般我们按完全二叉树的标准建立线段树，即节点 $x$ 的左儿子为 $2x$，右儿子为 $2x+1$。

可以证明，这样得到的线段树，总节点数不超过区间长度的 $4$ 倍。

- 建立：递归建立即可

```
#define ls (rt<<1)
#define rs ((rt<<1)+1)
void build(int l=1,int r=n,int rt=1){
    siz[rt]=r-l+1;
    if(l==r){
        /*init*/
        return;
    }
    int mid=(l+r)>>1;
    build(l,mid,ls);
    build(mid+1,r,rs);
    /*push_up*/
}
```

- 区间修改：如果有一个节点的区间完全被包含在修改区间内，则标记这个节点，然后返回即可，无需继续递归。$O(\log n)$

```
//示例：区间加
void add(int L,int R,int v,int l=1,int r=n,int rt=1){
    if(l<=L&&R<=r){
        tag[rt]+=v;
        sum[rt]+=siz[rt]*v;
        return;
    }
    push_down(rt);
    int mid=(l+r)>>1;
    if(L<=mid)add(L,R,v,l,mid,ls);
    if(R>mid)add(L,R,v,mid+1,r,rs);
    sum[rt]=sum[ls]+sum[rs];
}
```

- push_down：由于区间修改我们偷了懒，修改后存在一些节点应当被修改，但我们只是在它的祖先节点做了标记，没有真正修改该节点。所以当我们向下递归时，需要先看看左右儿子需不需要被修改。

  ```
  void push_down(int rt){
      if(tag[rt]){
          tag[ls]+=tag[rt],
          tag[rs]+=tag[rt];
          sum[ls]+=siz[ls]*tag[rt],
          sum[rs]+=siz[rs]*tag[rt];
          tag[rt]=0;
      }
  }
  ```

- 区间询问：和区间修改同理。

  ```
  int query(int L,int R,int l=1,int r=n,int rt=1){
      if(l<=L&&R<=r)
          return sum[rt];
      push_down(rt);
      int mid=(l+r)>>1,ans=0;
      if(L<=mid)ans+=query(L,R,v,l,mid,ls);
      if(R>mid)ans+=query(L,R,v,mid+1,r,rs);
      return ans;
  }
  ```

- 应用：
  - 区间求和，区间加，区间乘，区间求极值，区间赋值

    ```
    void push_down(int rt){
        if(asg[rt]){
            asg[ls]=asg[rs]=asg[rt];
            mul[ls]=mul[rs]=add[ls]=add[rs]=0;
            mx[ls]=mx[rs]=asg[rt];
            sum[ls]=siz[ls]*asg[rt];
            sum[rs]=siz[rs]*asg[rt];
            asg[rt]=0;
        }
        if(mul[rt]){
            mul[ls]*=mul[rt],mul[rs]*=mul[rt];
            sum[ls]*=mul[rt],sum[rs]*=mul[rt];
            mx[ls]*=mul[rt],mx[rs]*=mul[rt];
            mul[rt]=1;
        }
        if(add[rt]){
            add[ls]+=add[rt],add[rs]+=add[rt];
            sum[ls]+=add[rt]*siz[ls],sum[rs]+=add[rt]*sum[rs];
            mx[ls]+=add[rt],mx[rs]+=add[rt];
            add[rt]=0;
        }
    }
    //实列代码，不保证正确。
    ```

- 区间加等差数列

  等差数列由首项和公差唯一确定，并且可以分别累加。由此写出 push_down 函数：

```
void push_down(int rt){
    if(d[rt]||fir[rt]){
        d[ls]+=d[rt],d[rs]+=d[rt];
        fir[ls]+=fir[rt],fir[rs]+=fir[rt]+siz[ls]*d[rt];
        sum[ls]+=fir[rt]*siz[ls]+(siz[ls]-1)*siz[ls]*d[rt]/2;
        sum[rs]+=(fir[rt]+siz[ls]*d[rt])*siz[rs]+
(siz[rs]-1)*siz[rs]*d[rt]/2;
        d[rt]=fir[rt]=0;
    }
}
```

- 01序列区间取反，区间计算
- 区间异或，区间求和

  好像这时我们没有办法直接算出 sum 修改后的值。

  于是考虑将数字拆成二进制位，每位分别计算，区间异或就变成区间取反了。

```
void push_down(int rt){
    if(tag[rt]){
        tag[ls]^=tag[rt],tag[rs]^=tag[rt];
        for(int i=0;i<N;++i)
            if(tag[rt]&(1<<i)){
                sum[k][ls]=siz[ls]-sum[k][ls];
                sum[k][rs]=siz[rs]-sum[k][rs];
            }
        tag[rt]=0;
    }
}
```

- 单点修改，区间查询

  性质：若一个数取模后值发生变化，则其减少量超过一半。

  证明：$m>\frac{x}{2}时:x \bmod m=x-m<\frac{x}{2},m\le\frac{x}{2}时:x \bmod m<m\le\frac{x}{2}$

  维护区间最大值，若最大值大于模数则递归修改，小于则返回，复杂度 $O(m\log n\log V)$

- 查询区间是否由从 $1$ 开始的连续整数（不重复，不要求顺序）构成

  其实是区间 hash，我们需要构造一种压缩方式，使得集合信息压缩为一个数。

  一个可行的方式：求和 $x$ ，求平方和 $y$ ，判断 $x,y$ 是否与 从 $1$ 开始的连续整数的 $x',y'$ 相等。

- 给出一数列 $a_n$ ，要求支持两种操作：

  - 给定 $l,r$ ，求 $[l,r]$ 内的最大子段和。
  - 修改 $a_n$ 。

  $[l,r]$ 内的最大子段和 $d_{l,r}$ 可能有三种情况：$d_{l,r}=d_{l,mid}$ ，$d_{l,r}=d_{mid+1,r}$ ，$d_{l,r}=R_{l,mid}+L_{mid+1,r}$

  其中 $L,R$ 为从左/右端开始的连续子段和：$L_{l,r}=\max(sum_{l,mid}+L_{mid+1,r},L_{l,mid})$ ，$R$ 同理。

- 权值线段树：

  和权值树状数组类似，设 $cnt_i$ 表示数值等于 $i$ 的数据个数。

  但线段树能做的就多一些。

  - 求第 $k$ 小

    ```
    int kth(int k,int l=1,int r=n,int rt=1){
        if(l==r)return l;
        int mid=(l+r)>>1;
        if(k>cnt[ls])return kth(k-cnt[ls],mid+1,r,rs);
        return kth(k,l,mid,ls);
    }
    ```

  - 求 $x$ 的排名

    ```
    int find(int x,int l=1,int r=n,int rt=1){
        if(l==r)return 1;
        int mid=(l+r)>>1;
        if(x>mid)return cnt[ls]+find(x,mid+1,r,rs);
        return find(x,l,mid,ls);
    }
    ```

# 树链剖分

是一种将树上问题转移到序列问题的办法。

将树划分为若干链，根据划分依据不同，有重链剖分、长链剖分、实链剖分等种类。

主要讲重链剖分。

- 重儿子：子树大小最大的儿子。其余均为轻儿子。
- 重边：父节点与重儿子连的边。
- 轻边：父节点与轻儿子连的边。
- 重链：重边连成的链。

求重儿子：

```
void dfs1(int rt,int f){
    siz[rt]=1,fa[rt]=f,dep[rt]=dep[f]+1;
    int num=0;
    for(int i=head[rt];i;i=nxt[i])
        if(to[i]!=f){
            dfs1(to[i],rt);
            siz[rt]+=siz[to[i]];
            if(siz[to[i]]>num)num=siz[to[i]],son[rt]=to[i];
        }
}
```

遍历整棵树，优先走重边，按访问顺序编号，并处理出每个点所在重链的顶端。

```
void dfs2(int rt,int tp){
    top[rt]=tp,id[rt]=++tim;
    if(son[rt])dfs2(son[rt],tp);
    for(int i=head[rt];i;i=nxt[i])
        if(to[i]!=fa[rt]&&to[i]!=son[rt])
            dfs2(to[i],to[i]);
}
```

应用:

- LCA

```
int LCA(int x,int y){
    int ans=0;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        x=fa[top[x]];
    }
    if(dep[x]>dep[y])swap(x,y);
    return x;
}
```

- 路径修改/查询

```
void Tmodify(int x,int y){
    int ans=0;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        modify(id[top[x]],id[x]);
        x=fa[top[x]];
    }
    if(dep[x]>dep[y])swap(x,y);
    query(id[x],id[y]);
}
```

- 子树修改/查询

```
void Tmodify(int x){
    modify(id[x],id[x]+siz[x]-1);
}
```