# 基本数据结构--队列目录

[一、 Team Queue](#_Toc5120_WPSOffice_Level1) [1](#_Toc5120_WPSOffice_Level1)

[二、 蚯蚓](#_Toc29844_WPSOffice_Level1) [2](#_Toc29844_WPSOffice_Level1)

[三、 双端队列](#_Toc11693_WPSOffice_Level1) [5](#_Toc11693_WPSOffice_Level1)

[四、 最大子序和 （单调队列）](#_Toc31049_WPSOffice_Level1) [6](#_Toc31049_WPSOffice_Level1)

**基本数据结构--链表与邻接表目录**

1. 邻值查找.............................................................................8
2. 邻接表..............................................................................10

### Team Queue

#include <bits/stdc++.h>

**using** **namespace** std;

**const** **int** maxn = 1005;

**int** n , num , x;

**bool** vis[maxn]; //判断是否有小队的人在队列种中

map<**int**,**int**> mp; //<id , team\_num>

queue<**int**> q[maxn];

**void** init(){

**for**(**int** i = 0 ; i < maxn ; ++ i)

**while**(!q[i].empty()) q[i].pop();

**memset**(vis, 0 ,**sizeof** vis);

    mp.clear();

}

**int** main(){

**int** cas = 0;

**while**(cin >> n){

**if**(!n) **break**;

        init();

        cas ++;

        cout << "Scenario #" << cas << endl;

**for**(**int** i = 1 ; i <= n ; i ++){

            cin >> num;

**for**(**int** j = 1 ; j <= num ; ++ j){

                cin >> x;

                mp[x] = i;

            }

        }

        string s;

**while**(cin >> s){

**if**(s == "STOP") **break**;

**else** **if**(s == "ENQUEUE"){

                cin >> x;

                //cout << x << endl;

**if**(!vis[mp[x]]){

                    q[0].push(mp[x]);

                    vis[mp[x]] = 1;

                }

                q[mp[x]].push(x);

            }**else**{

**int** team = q[0].front();

**int** now = q[team].front();

                q[team].pop();

**if**(q[team].empty()){

                    q[0].pop();

                    vis[team] = 0;

                }

                cout << now << endl;

            }

        }

        cout << endl;

    }

**return** 0;

}

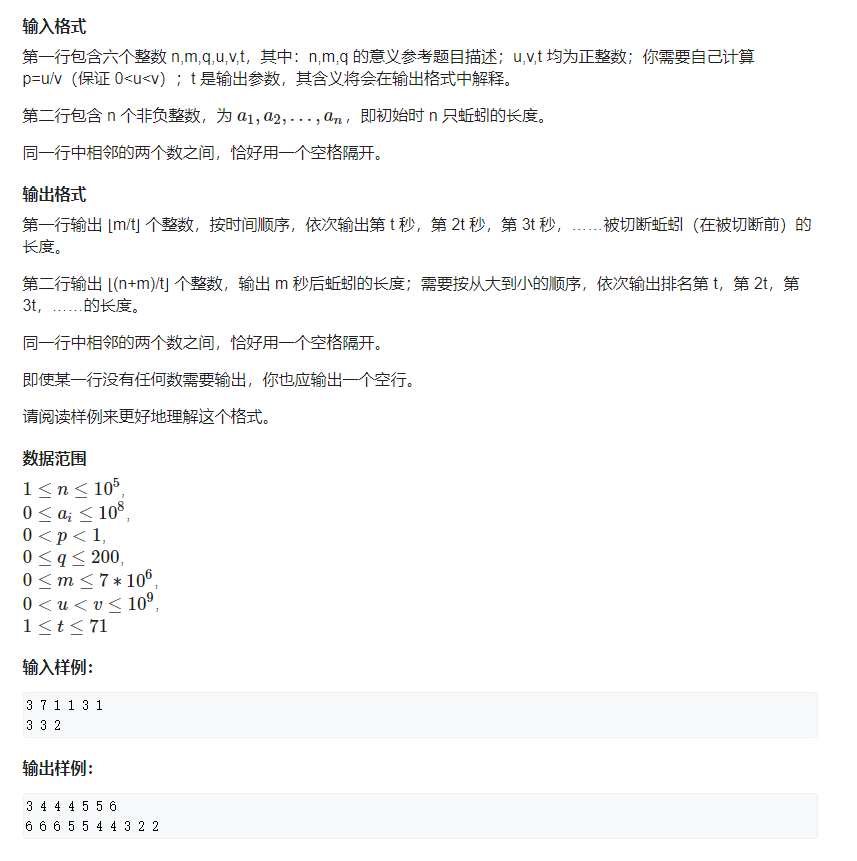
### 蚯蚓

题意：

有n个线段，第i个蚯蚓的长度为ai , ai >= 0。每一秒,选择最长的一段蚯蚓，切成 ⌊p\*x⌋ 和 x−⌊p\*x⌋, 特殊地，如果这两个数的其中一个等于0，则这个长度为0的线段也会被保留 。此外，除了刚刚产生的两只新线段，其余线段的长度都会增加一个非负整数 q 。现在希望知道：

1）m 秒内，每一秒被切断的线段被切断前的长度，共有 m 个数。

2）m 秒后，所有线段的长度，共有 n+m 个数。



分析：

1. 原序列从大到小排序 a1 a2 a3 a4 a5 ...

2. left序列

3. right序列

发现如果蚯蚓i被切断后,然后蚯蚓j被切断（ai >= aj）,可以证明 ,i蚯蚓的切后两段的长度,都会大于j蚯蚓的切后的两段的长度,因此这里有单调递减的性质.

即left[i] >= left[j] , right[i] >= right[j]。

找到了性质,那么可以通过三个队列模拟优先队列,一个队列维护切后的第一段left[],一个队列维护切后的第二段right[],

另外一个队列,里面存储蚯蚓长度,记住长度是从高到低,排好序的长度,那么每一次将被切断的蚯蚓,肯定是这三个队列的队头,

因为这道题目具有单调递减的性质,所以其实这道题目三个队列,都隐藏着单调队列的性质.

另外存一个偏移量delta，三个队列中存相对值，不需要每秒每个数都加q，只需要每秒更新delta。

每次切找三个队列的队头元素,然后将切的两段分别放入left[]和right[]

#include <bits/stdc++.h>

**using** **namespace** std;

**const** **int** maxn = 100005, maxm = 7000005;

**typedef** **long** **long** ll;

ll n ,m ,q ,u ,v ,t ,a[maxn];

queue<ll> Left , Right , arr;

**bool** cmp(ll x, ll y){

**return** x > y;

}

ll delta; // 偏移量

//求得最大值并删去 O(1)

ll get\_max(){

    ll ret = LONG\_LONG\_MIN;

**int** pos = 0; //记录从哪个队列中删 arr:pos=1,left:pos=2,right:pos=3

**if**(!arr.empty() && arr.front() > ret){

        ret = arr.front();

        pos = 1;

    }

**if**(!Left.empty() && Left.front() > ret){

        ret = Left.front();

        pos = 2;

    }

**if**(!Right.empty() && Right.front() > ret){

        ret = Right.front();

        pos = 3;

    }

**if**(pos == 1) arr.pop();

**else** **if**(pos == 2) Left.pop();

**else** Right.pop();

**return** ret;

}

**int** main(){

    ios::sync\_with\_stdio(**false**);

    cin >> n >> m >> q >> u >> v >> t;

**for**(**int** i = 1 ; i <= n ; ++ i) cin >> a[i];

    sort(a + 1 , a + 1 + n ,cmp);

**for**(**int** i = 1 ; i <= n ; ++ i) arr.push(a[i]);

**for**(**int** i = 1 ; i <= m ; ++ i){

        ll x = get\_max();

        x += delta;

        ll l = x \* 1ll \* u / v;

        ll r = x - l;

**if**(i % t == 0) cout << x << ' ';

        delta += q;

        Left.push(l - delta);

        Right.push(r - delta);

    }

    cout << endl;

**for**(**int** i = 1 ; i <= m + n; ++ i){

        ll x = get\_max();

**if**(i % t == 0) cout << x + delta << ' ';

    }

    cout << endl;

**return** 0;

}

### 双端队列

性质:

1.每个deque都是连续的一段

2.最终deque内元素的下标为“单谷”形

#include <bits/stdc++.h>

**using** **namespace** std;

**const** **int** maxn = 200005;

**struct** node{

**int** val , id;

}a[maxn];

**bool** cmp(node x, node y){

**if**(x.val == y.val) **return** x.id < y.id;

**return** x.val < y.val;

}

**int** n;

**int** main(){

    cin >> n;

**for**(**int** i = 1 ; i <= n ; ++ i){

        cin >> a[i].val;

        a[i].id = i;

    }

    sort(a + 1 , a + 1 + n ,cmp);

**int** cnt = 1 , last = INT\_MAX;

**bool** dir = 0; //下降:dir = 0   上升:dir=1;

**for**(**int** i = 1 , j = i; i <= n ; i = j){

**while**(j <= n && a[j].val == a[i].val) j ++;  //寻找相等的一段数 [i , j - 1]

**int** minp = a[i].id , maxp = a[j - 1].id;

**if**(!dir){ //如果当前序列为下降趋势

**if**(last > maxp) last = minp; //能接上

**else** last = maxp , dir = 1; 　//接不上，当前last为谷　趋势转为上升

        }**else**{ //上升

**if**(minp  > last) last = maxp; //能接上

**else** {　//接不上，当前last为当前deque的最后一位 新开一个deque 趋势设为下降

                dir = 0 ;

                cnt ++ ;

                last = minp;

            }

        }

    }

    cout << cnt << endl;

**return** 0;

}

### 最大子序和 （单调队列）

第i为终点的最大的答案：S[i] - S[j] (j ：m ~ i - 1)

S[j]越小 ， S[i] - S[j]越大

我们需要维护一个**下标单调递增、对应前缀和S也递增**的序列。

（前缀和S单调递增的单调队列）

原因：当k < j < i 并且S[k] >= S[j]时，终点为i的最大答案选j一定比k优，因此需要及时淘汰S[k]位置。

步骤：

1. 判断队头决策与i的距离是否超出M的范围，若超出则出队。（遍历从左到右，当前i超出M个，之后也超出M个，因此及时淘汰超出M的位置）。
2. 此时对头就是右端点为i时，左端点j的最优选择 （即在M范围内，队首就是终点为i的最大答案）。
3. 不断删除队尾决策，直到队尾对应的S值小于S[i]。然后把i作为一个新的决策入队。（入队要保证队列中元素单调递增）

#include  <bits/stdc++.h>

**using** **namespace** std;

**typedef** **long** **long** ll;

**const** **int** maxn = 300005;

ll q[maxn] ,sum[maxn] ,ans = 0;

**int** main(){

**int** n , m;

    cin >> n >> m;

**for**(**int** i = 1 ; i <= n ; ++ i){

        cin >> sum[i];

        sum[i] += sum[i - 1];

    }

**int** l = 1 ,r = 1;

    q[1] = 0;

**for**(**int** i = 1 ; i <= n ; ++ i){

**while**(l <= r && q[l] < i - m) l ++; //step1

        ans = max(ans , sum[i] - sum[q[l]]); //step2

**while**(l <= r && sum[q[r]] >= sum[i]) r--; //step3

        q[++ r] = i;

    }

    cout << ans << endl;

}

### 邻值查找

做法一：使用STL中的set（在线）

#pragma GCC optimize(2)

#include <bits/stdc++.h>

**using** **namespace** std;

set<pair<**int**,**int**> > s;

**int** main(){

    ios::sync\_with\_stdio(**false**);

    cin.tie(0) , cout.tie(0);

**int** n;

    cin >> n;

**for**(**int** i = 1 ; i <= n ; ++ i){

**int** x;

        cin >> x;

**if**(i != 1){

            auto it = s.lower\_bound(make\_pair(x , -1));

**if**(it == s.end()){

                it --;

                cout << **abs**(x - it->first) << ' ' << it->second << '\n';

            }**else** **if**(it == s.begin()){

                cout << **abs**(x - it->first) << ' ' << it->second << '\n';

            }**else**{

**int** ans2 = it->first , pos2 = it->second;

                it--;

**int** ans1 = it->first , pos1 = it->second;

**if**(x - ans1 <= ans2 - x) cout << **abs**(ans1 - x) << ' ' << pos1 << '\n';

**else** cout << **abs**(ans2 - x) << ' ' << pos2 << '\n';

            }

        }

        s.insert(make\_pair(x , i));

    }

**return** 0;

}

解法二：双向链表（离线 nlogn）

1. 读取所有数，记录好每个数的下标后从小到达排序.
2. 先从排完序后原本下标为n的数开始找，这个数的前驱和后继就是与之最近的两个数，计算差按需记录答案，用于最后输出，然后删除这个数。（因为其它所有数都在它之前，所以选的数一定满足j<i这个条件）
3. 继续找原本下标为n - 1的数。一直运行步骤2直到算完下标为2的数为止。

#include <bits/stdc++.h>

**using** **namespace** std;

**const** **int** maxn = 100010;

**int** n;

pair<**int**,**int**> a[maxn] , ans[maxn];

**int** p[maxn]; //用于记录点i在链表中的位置

**int** l[maxn] , r[maxn]; //记录点i的前驱和后继

**int** main(){

    ios::sync\_with\_stdio(**false**);

    cin.tie(0); cout.tie(0);

    cin >> n;

**for**(**int** i = 1 ; i <= n ; ++ i){

        cin >> a[i].first;

        a[i].second = i;

    }

    sort(a + 1 , a + 1 + n);

    a[0].first = 1e9 , a[n + 1].first = -1e9;

**for**(**int** i = 1 ; i <= n ; ++ i){ //建立链表

        l[i] = i - 1 , r[i] = i + 1;

        p[a[i].second] = i;

    }

**for**(**int** i = n ; i > 1 ; -- i){

**int** j = p[i] , pre = l[j] , nxt = r[j]; //当前元素在链表中的位置,前驱以及后继

**int** lv = **abs**(a[j].first - a[pre].first);

**int** rv = **abs**(a[j].first - a[nxt].first);

**if**(lv <= rv) ans[i] = {lv , a[pre].second};

**else** **if**(lv > rv) ans[i] = {rv , a[nxt].second};

        //删除操作

        r[pre] = nxt , l[nxt] = pre;

    }

**for**(**int** i = 2 ; i <= n ; ++ i) cout << ans[i].first << ' ' << ans[i].second << '\n';

**return** 0;

}

### 邻接表

**void** add(**int** x , **int** y , **int** z){

    ver[++tot] = y , edge[tot] = z;  //真实数据 ver[]存边的终点 edge[]存边的权值

    next[tot] = head[x] , head[x] = tot; //next[]和ver[]存储的是ver[]的下标

}

//遍历图

**for**(**int** i = head[x] ; i ; i = next[i]){

**int** y = ver[i] , z = edge[i];

    //找到一条有向边(x,y)，权值为z

}

初始化tot = 1 ，这样ver[]和edge[]的下标就会从2开始，这样存储无向边看成两条有向边会成对存储在下标为"2 , 3","4 , 5","6 , 7"......的位置上。

0X01中的“成对变换”对下标进行xor 1的运算就可以方便的直接定位到与当前边反向的边。

e.g. ver[i]是第i条边的终点，那么ver[i xor 1]就是第i条边的起点。