数据结构

单调队列

原题链接：AcWing 154. 滑动窗口

滑动窗口求最值的问题都是用单调队列来解决的

单调队列的元素从队尾**插入**，从队头**取出**，为保证整个队列的单调性，元素在队尾进行**调整**

也就是每次从对头取出的元素一定是整个单调队列当中的最值，而每次都会从队尾插入元素，与此同时也会在队尾调整元素

用代码描述这一整个过程（先后顺序不能乱）就是：

- 在队尾删除元素以保证整个队列的单调性
- 在队尾插入一个新的元素
- 在对头取出元素，此元素便是整个单调队列的最值

在滑动窗口求最值的问题中，需要在最前面加上队头元素是否会离开滑动窗口，但不管怎么样，上面的三个顺序不能变

关于单调队列的代码实现：

我们定义 hh 表示队头指针，tt 表示队尾指针，初始时 $hh = 0, tt = -1$ ，队列当中存储的是各个元素的下标

判断队列是否为空：$hh \leq tt$

队尾插入：$h[++tt] = k$

队头删去：$h++$

在保证队列单调性的部分，我们需要考虑的是删掉的元素与**当前遍历到的元素** $a[i]$ 之间的大小关系

如果我们期望队列严格单调递增，那么我们需要删去所有小于等于 $a[i]$ 的的元素

如果我们期望队列单调递增，那么需要删去所有小于 $a[i]$ 的元素，即队列当中允许存在等于 $a[i]$ 的元素

就本题而已，由于需要求的只是最大和最小值，因此有没有等号都可以

关于队头元素何时出队的问题，我们需要明确**单调队列中元素个数是多少**，即**单调队列最左边的数的下标是什么**

就本题来说，单调队列中只有 k 个数，因此最远的合法下标为 $i-k+1$ ，当 $q[hh] < i-k+1$ 时表示队头元素已经出队

完整代码：

```
#include <iostream>

using namespace std;

const int N = 1e6 + 10;
int a[N], q[N];
int n, k;

int main()
{
    cin >> n >> k;
    for(int i = 1; i <= n; i++) cin >> a[i];
    int hh = 0, tt = -1;
    for(int i = 1; i <= n; i++)
    {
        if(hh <= tt && i - q[hh] > k - 1) hh++;//判断队头元素是否离开窗口内
        while(hh <= tt && a[q[tt]] >= a[i]) tt--;//保证队列内元素严格递减，因此需要将所以大于等于a[i]全部删去
        q[++tt] = i;//先将元素插入到队列中，之后才能从队头取数
        if(i >= k) cout << a[q[hh]] << " ";//只有遍历到的数大于窗口长度，就可以输出
    }
    cout << endl;
    hh = 0, tt = -1;
    for(int i = 1; i <= n; i++)
    {
        if(hh <= tt && i - q[hh] > k - 1) hh++;
        while(hh <= tt && a[q[tt]] <= a[i]) tt--;
        q[++tt] = i;
        if(i >= k) cout << a[q[hh]] << " ";
    }
    return 0;
}
```

KMP

原题链接：AcWing 831. KMP字符串

模板：

```
#include <iostream>

using namespace std;

const int N = 1e6 + 10;

char p[N], s[N];
int ne[N];
```

```
int n, m;

int main()
{
    cin >> n >> p + 1 >> m >> s + 1;
    for(int i = 2, j = 0; i <= n; i ++)
    {
        while(j && p[i] != p[j + 1]) j = ne[j];
        if(p[i] == p[j + 1]) j++;
        ne[i] = j;
    }

    for(int i = 1, j = 0; i <= m; i ++)
    {
        while(j && s[i] != p[j + 1]) j = ne[j];
        if(s[i] == p[j + 1]) j++;
        if(j == n)
        {
            cout << i - n << " ";
            j = ne[j];
        }
    }
    return 0;
}
```

原题链接：[AcWing 141. 周期]

本题实际上是结论题：

- 对于长度为 i 的字符串而言，如果满足 i 那么该字符串存在**最小循环节**，最小循环节长度为 $i - next[i]$ ，循环次数 $K = i/next[i]$
- 任意一个循环节的长度都是最小循环节长度的整数倍
- 如果 i 不能整除 $i - next[i]$ 那么该字符串**没有最小循环节**

下面一一证明这些结论：

$next[i]$ 表示长度为 i 字符串的**最长相等前后缀**，设 $T = i - next[i]$ ，我们保证 T 始终大于 0 ，即 $i > next[i]$

对于字符串 S 而言，有：$S[1 \sim next[i]] = S[T + 1 \sim i]$ ，即 S 中**后** $next[i]$ **个字符**与将 S 向右偏移 T 个单位后的**前** $next[i]$ **个字符相同**

因此有 $S[1 \sim T] = S[T + 1 \sim 2T]$

如此这般，我们有 $S[1 \sim T] = S[T + 1 \sim 2T] = S[2T + 1 \sim 3T] = \cdots = S[i - T + 1, i]$

上述等式**成立的条件为** i **能够整除** T

下面我们证明 T 为最小循环节：

假设存在另一循环节 T' ，满足 $T' < T$

在 S 中除第一个循环节外剩余长度为 $i - T$ ，并且有 $next[i] = i - I$

由于 $T' < T$ ，因此 $i - T' > i - T$ ，即 $next[i]' > next[i]$

由于 $next[i]$ 为最长相等前后缀，即不存在另一个相等前后缀比 $next[i]$ 更大，因此假设矛盾

到此为止，我们证明了 $T = i - next[i]$ 为最小循环节。现在有一个问题是，如果 T 不能整除 i ，是否存在一个循环节 T' ，且 $T' > T$ ，满足 i 能够整除 T' 成立

也就是最小循环节 T 不能成为 S 的一个循环节，但是否存在另一个循环节 T' 是 S 的一个循环节?

假定 T 为最小循环节，T' 为另一循环节，且 $T' > T, T \nmid T'$

设 $d = gcd(T, T')$ ，由于 $T \nmid T'$ 且 $T < T'$ 因此 $d < T$

由裴蜀定理，一定存在一对整数 x, y 使得 $d = xT + yT'$ ，不妨设 $x > 0$

由于 T 与 T' 均为循环节，因此
$S_i = S_{i+T} = S_{i+2T} = \cdots = S_{i+xT} = S_{i+xT+T'} = S_{i+xT+2T'} = \cdots = S_{i+xT+yT'} = S_d$

因此 d 也是 S 的一个循环节，由于 $d < T$ ，而 T 为 S 的最小循环节，因此产生矛盾

所以 S 的**任意循环节均是最小循环节的整数倍**

完整代码:

```
#include <iostream>

using namespace std;

const int N = 1e6 + 10;

int ne[N];
char str[N];
int n;

int main()
{
    int T = 1;
    while(cin >> n, n)
    {
        cin >> str + 1;
        for(int i = 2, j = 0; i <= n; i ++)
        {
            while(j && str[i] != str[j + 1]) j = ne[j];
            if(str[i] == str[j + 1]) j++;
            ne[i] = j;
        }

        cout << "Test case #" << T++ << endl;

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

```
            int t = i - ne[i];
            if(i % t == 0 && i / t > 1) printf("%d %d\n", i, i / t);
        }
        cout << endl;
    }
    return 0;
}
```