



中国计算机学会  
China Computer Federation

# 简单数据结构



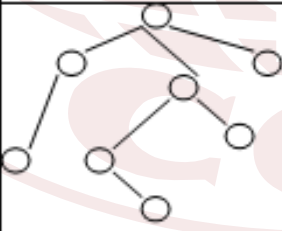
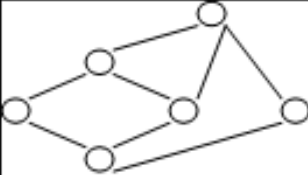
成都七中 蔺洋

# 什么是数据结构？

- **数据**是一切能输入计算机中的信息的总和，**结构**是指数据之间的关系。
- **数据结构**就是将数据及其之间的关系有效地存储在计算机中并进行基本操作。
- **算法**是对特定问题求解步骤的一种描述，通俗讲就是解决问题的方法和策略。
- Niklaus Wirth教授所说的：**“数据结构 + 算法 = 程序”**

# 什么是数据结构？

- 存储结构：顺序存储结构（数组）和链式存储结构（链表）
- 逻辑结构：集合、线性结构、树形结构、图结构

	数据间的关系特征	图示	备注
集合	数据元素“属于”或“不属于”集合。二者必居其一。		一般用其他数据结构代替
线性结构	数据元素存在一个对一个的关系		操作简单，编程复杂度低。
树形结构	数据元素存在一个对多个的关系		平均操作效率很高。广泛应用于查找。
图结构	数据元素存在多个对多个的关系		结构复杂、应用广泛，是常见的数学模型。有许多经典算法。

# 线性的数据结构

- 数组
- 栈
- 队列
- 链表
- .....



# 数组

- 数组 ( Array ) 是有序的元素序列。是一种线性表**数据结构**。是由相同类型的元素 ( element ) 的集合所组成的数据结构，它用**一组**连续的内存空间,来存储**一组具有相同类型的数据**。利用元素的索引 ( index ) 可以计算出该元素对应的存储地址。

如： `int a[10]`

- 注意：C++语言中，每个数组第一个元素的下标都是**0**，因此第一个元素为第**0**个数组元素。
- 访问  $O(1)$
- 插入  $O(n)$
- 删除  $O(n)$

# 动态数组

- STL中的vector是动态数组，可变长度的数组。

定义：`vector<数据类型>` 数组名

如：`vector<int> v;`

- `v.push_back();` // 数组后面增加一个元素。
- `v.pop_back();` // 删除数组里的最后一个元素。
- `v.size();` // 返回v的长度。

例1：给定一个长度为N的数列，有M次操作，每次操作有以下两种之一：

- 1、修改数列中的一个数
- 2、求数列中某位置的值

$1 \leq N, M \leq 1000000$

例2：给定一个长度为N的数列，有M次操作，每次操作有以下两种之一：

- 1、修改数列中的一个数
- 2、求数列中某位置某次操作后的值

$1 \leq N, M \leq 100000$



# 链表

- 链表是一种物理存储单元上**非连续、非顺序**的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。

访问 $O(n)$

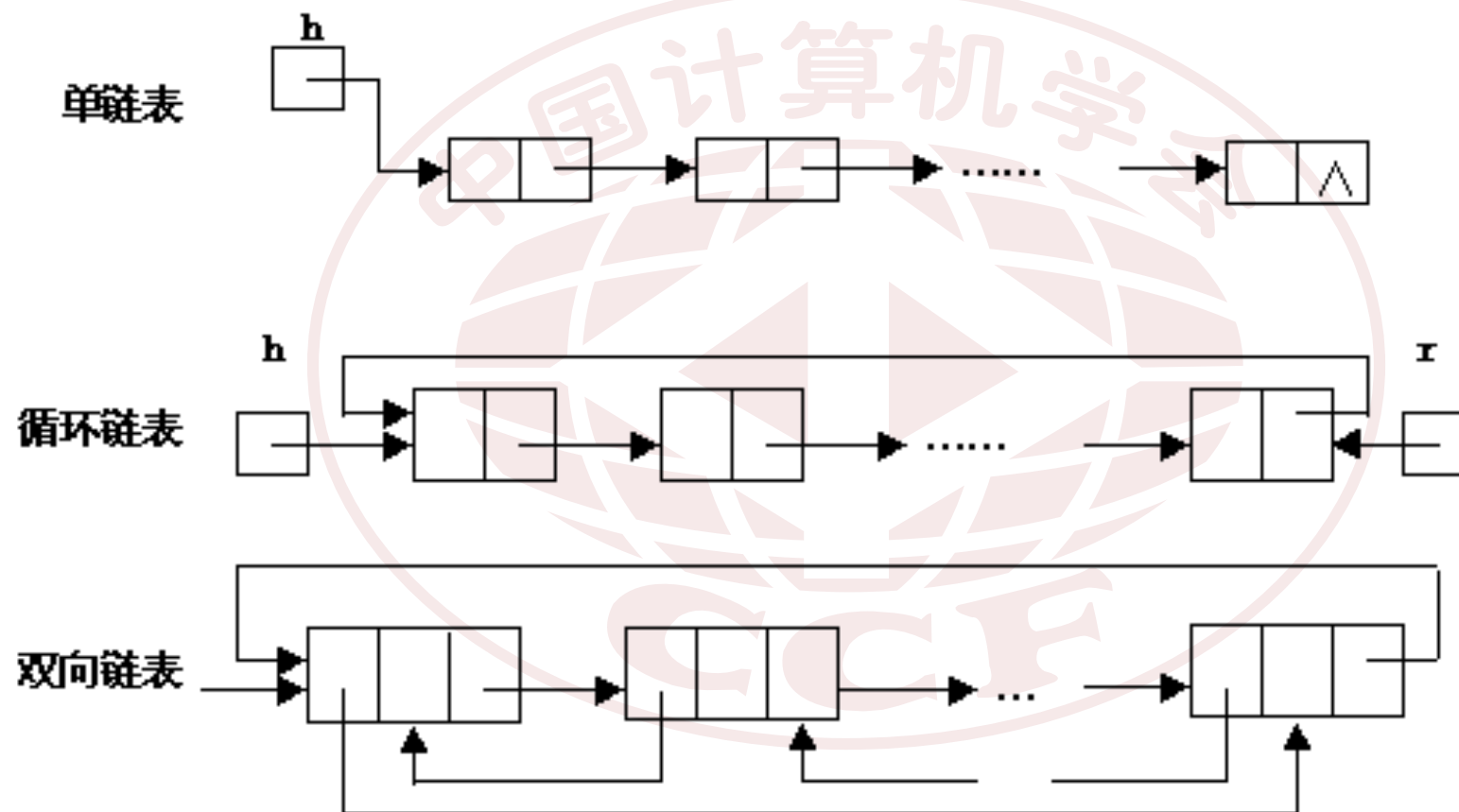
插入 $O(1)$

删除 $O(1)$

链表：插入删除迅速 $O(1)$ ，不支持随机下标访问

数组：插入删除是 $O(n)$ ，支持随机下标访问

# 常见的链表形态



**例3：约瑟夫环：**已知 $n$ 个人（以编号1，2，3... $n$ 分别表示）围坐在一张圆桌周围。从编号为 $k$ 的人开始报数，数到 $m$ 的那个人出列；他的下一个人又从1开始报数，数到 $m$ 的那个人又出列；依次规律重复下去，直到圆桌周围的人全部出列。输出出圈序列。

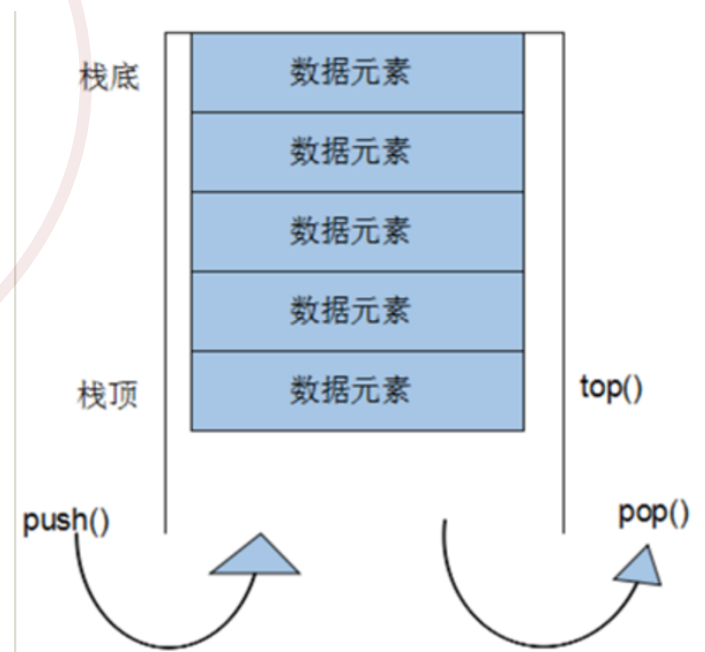
# 数组模拟链表

例4：一群小朋友排队，排成一排，每个人都只记得自己右手边站的是谁。如何利用这些信息，恢复出他们的排队顺序。

如：1-4， 3-2， 4-3， 2-5， 5-none

# 线性结构（栈）

- 限定只能在表的一端进行插入和删除操作的线性表。
- 栈顶(top)：允许插入和删除的一端。
- 栈底(bottom)：不允许插入和删除的另一端。
- 空栈：不含元素的空表。
- 栈是先进后出 FILO
- 每次操作是 $O(1)$



# 线性结构（栈）

- 栈的程序实现

**//用数组来实现栈**

**const int N=10005;**

**int s[N],top;**

**top=0; //初始化,置空栈**

**return (top==0); //判栈是否为空**

**s[++top]=x; //栈顶插入一个元素**

**s[top--]=0; //删除栈顶元素**

**x=s[top]; //取栈顶元素**

**//用STL中stack来实现栈**

**stack <int> s; //定义一个栈即空栈**

**return s.empty(); //判栈是否为空**

**s.push(x); //在栈顶插入一个元素**

**s.pop(); //删除栈顶元素**

**s.size(); //栈里有多少个元素**

**x=s.top(); //取栈顶元素**

# 线性结构（栈）

例 5：

1、今有一空栈S，对下列待进栈的元素序列a,b,c,d,e,f一次进行进栈，进栈，出栈，进栈，进栈，出栈的操作，此操作完成后，栈S的栈顶元素为（ ）。

A. f      B. c      C. a      D. b

2. 对于入栈顺序为a, b, c, d, e, f, g 的序列，下列( )不可能是合法的出栈序列。

A. a,b,c,d,e,f,g      B. a,d,c,b,e,g,f  
C. a,d,b,c,g,f,e      D. g,f,e,d,c,b,a

# 线性结构（栈）

- 栈的应用

只要问题满足LIFO 原则，均可使用栈做数据结构。

例如：

- DFS（深度优先搜索）
- 数制转换、
- 判断回文数、
- 括号匹配、
- 栈与递归
- 表达式求值、
- 后序表达式



# 栈的应用

## 例6：进制转换

输入格式

输入一个十进制数 $N$ 与需要转换的进制 $d$

输出格式

输出转换后的 $d$ 进制数

# 栈的应用

## 例7：括号匹配：

给定一个字符串，这个字符串由() { }这六个字符构成。如果所有的括弧都可以匹配上，那么就说这个字符串合法，否则非法。下面给一些例子：

( )或( (  ))

合法

( )或( ( ))或(( ) )

非法

# 栈的应用

## 例8：后缀表达式求值

所谓后缀表达式是指这样的一个表达式：式中不再引用括号，运算符号放在两个运算对象之后，所有计算按运算符号出现的顺序，严格地由左而右新进行（不用考虑运算符的优先级）。

如：

$3*(5-2)+7$ 对应的后缀表达式为： $3 \ . \ 5 \ . \ 2 \ . \ - \ 7 \ . \ + \ @$ 。  
‘@’为表达式的结束符号。‘.’为操作数的结束符号。

# 栈的应用

思考：中缀表达式如何求值？

- 1、中缀表达式直接求值
- 2、把中缀表达式转成后缀表达式
  - ①利用栈进行转换
  - ②利用二叉树进行转换

# 栈的应用

- ❖ 中缀表达式  $a + b * (c - d) - e / f$
- ❖ 后缀表达式  $a b c d - * + e f / -$
- ❖ 前缀表达式  $- + a * b - c d / e f$

练一练：

表达式  $a * (b + c) * d$  的后缀形式是()。

- A.  $abcd*+*$     B.  $abc+*d*$     C.  $a*bc+*d$     D.  $b+c*a*d$

# 栈的应用

例9：

题目描述

M 海运公司最近要对旗下仓库的货物进出情况进行统计。目前他们所拥有的唯一记录就是一个记录集装箱进出情况的日志。该日志记录了两类操作：第一类操作为集装箱入库操作，以及该次入库的集装箱重量；第二类操作为集装箱的出库操作。这些记录都严格按时间顺序排列。集装箱入库和出库的规则为**先进后出**，即每次出库操作出库的集装箱为当前在仓库里所有集装箱中最晚入库的集装箱。

出于分析目的，分析人员在日志中随机插入了若干第三类操作——查询操作。分析日志时，每遇到一次查询操作，都要报告出当前仓库中最大集装箱的重量。

输入格式：

包含N+1 行：

第一行为1 个正整数N，对应于日志内所含操作的总数。

接下来的N 行，分别属于以下三种格式之一：

格式1: 0 X //一次集装箱入库操作，正整数X表示该次入库的集装箱的重量

格式2: 1 //一次集装箱出库操作，（就当时而言）最后入库的集装箱出库

格式3: 2 //一次查询操作，要求分析程序输出当前仓库内最大集装箱的重量

当仓库为空时你应该忽略出库操作，当仓库为空查询时你应该输出0。

输出格式：

输出行数等于日志中查询操作的次数。每行为一个正整数，表示查询结果。

# 栈的应用

输入样例#1：

13  
0 1  
0 2  
2  
0 4  
0 2  
2  
1  
2  
1  
1  
2  
1  
2

输出样例#1：

2  
4  
4  
1  
0

说明

对于20%的数据，有 $N \leq 10$ ；

对于40%的数据，有 $N \leq 1000$ ；

对于100%的数据，有 $N \leq 200000$ ， $X \leq 10^8$ 。

# 单调栈

- 单调栈中存放的数据应该是有序的，所以单调栈也分为单调递增栈和单调递减栈
- 单调递增栈：单调递增栈就是从栈顶到栈底数据是从小到大
- 单调递减栈：单调递减栈就是从栈顶到栈底数据是从大到小



# 单调栈

单调递减栈：10，3，7，4，12。

从左到右依次入栈，则如果栈为空或入栈元素值小于栈顶元素值，则入栈；否则，如果入栈则会破坏栈的单调性，则需要把比入栈元素小的元素全部出栈。单调递减的栈反之。

10入栈时，栈为空，直接入栈，栈内元素为10。

3入栈时，栈顶元素10比3大，则入栈，栈内元素为10，3。

7入栈时，栈顶元素3比7小，则栈顶元素出栈，此时栈顶元素为10，比7大，则7入栈，栈内元素为10，7。

4入栈时，栈顶元素7比4大，则入栈，栈内元素为10，7，4。

12入栈时，栈顶元素4比12小，4出栈，此时栈顶元素为7，仍比12小，栈顶元素7继续出栈，此时栈顶元素为10，仍比12小，10出栈，此时栈为空，12入栈，栈内元素为12。

# 单调栈应用

1. 求最长的单调上升、递减区间
  2. 针对每个数，寻找它和它左 / 右边第一个比它大 / 小的数的值，以及相距多少个数。
  3. 多个区间中的最值 / 某个数为最值的最长区间
- .....

# 单调栈应用

例10、

题目描述:

$n$ 个人排成一条直线（一排），给出队伍中每个人的身高，每个人只能看到站在他右边且个头比他小没有被其他人挡住（跟他身高相同也会挡出他）的人。请求出所有人可以看到的人数之和。 $1 \leq N \leq 80,000$

input

6

10

3

7

4

12

2

output

5

# 单调栈应用

例11、

题目描述

给定从左到右多个矩形，已知这此矩形的宽度都为 1，长度不完全相等。这些矩形相连排成一排，求在这些矩形包括的范围内能得到的面积最大的矩形，打印出该面积。所求矩形可以横跨多个矩形，但不能超出原有矩形所确定的范围。

输入

输入共一行，第一个数表示矩形的个数。接下来 个数表示矩形的大小。（ $1 \leq n \leq 100000$ ）

输出

输出最大矩形面积。

样例输入

7

2 1 4 5 1 3 3

样例输出

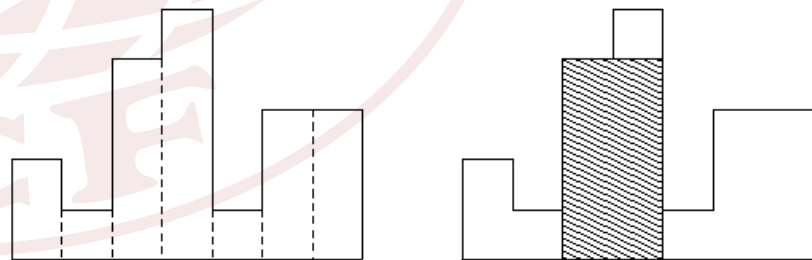
8

数据规模与约定

时间限制：1 s

内存限制：256 M

100% 的数据保证  $1 \leq n \leq 100000$





# 线性结构（队列）

队列同样是一种线性表，与栈不同的是，队列是在一端进行删除，在另一端进行插入的线性表，所以队列又被称为先进先出表（FIFO）。

```
//数组实现queue
const int N=10005;
int q[N],front=0,back=0;//初始化,置空
return (front>back);//判队列是否为空
back++;q[back]=x;//将元素x插入队列
front++;//队头元素弹出
```

```
//STL中的queue
queue<int> q
q.front()返回队首元素
q.back()返回队尾元素
q.push()插入队尾
q.pop()队首出队
q.empty()队列是否为空
q.size()返回队列元素个数
```

入队：

出队：



# 队列应用

- 宽度优先搜索 ( Breadth-first Search )
- 循环队列
  - 队列出现“假溢”的情况，循环的利用队列空间
- 双端队列
  - Sliding Window ( 滑动窗口 )
  - STL中队列，栈等都是用双端队列实现的

# 队列应用

## 例12、题目描述

约瑟夫环是一个数学的应用问题：已知 $n$ 个人（以编号 $1, 2, 3 \dots n$ 分别表示）围坐在一张圆桌周围。从编号为 $k$ 的人开始报数，数到 $m$ 的那个人出列；他的下一个人又从1开始报数，数到 $m$ 的那个人又出列；依此规律重复下去，直到圆桌周围的人全部出列。

输入

8 1 3 ( $n=8$   $k=1$   $m=3$ )

输出

7 (剩下的那个)

样例输入

8 1 3

样例输出

7

# 队列应用

## 例13、题目描述

小晨的电脑上安装了一个机器翻译软件，他经常用这个软件来翻译英语文章。

这个翻译软件的原理很简单，它只是从头到尾，依次将每个英文单词用对应的中文含义来替换。对于每个英文单词，软件会先在内存中查找这个单词的中文含义，如果内存中有，软件就会用它进行翻译；如果内存中没有，软件就会在外存中的词典内查找，查出单词的中文含义然后翻译，并将这个单词和译义放入内存，以备后续的查找和翻译。

假设内存中有 $M$ 个单元，每单元能存放一个单词和译义。每当软件将一个新单词存入内存前，如果当前内存中已存入的单词数不超过 $M-1$ ，软件会将新单词存入一个未使用的内存单元；若内存中已存入 $M$ 个单词，软件会清空最早进入内存的那个单词，腾出单元来，存放新单词。假设一篇英语文章的长度为 $N$ 个单词。给定这篇待译文章，翻译软件需要去外存查找多少次词典？假设在翻译开始前，内存中没有任何单词。

## 输入格式

输入文件共2行。每行中两个数之间用一个空格隔开。

第一行为两个正整数 $M$ 和 $N$ ，代表内存容量和文章的长度。

第二行为 $N$ 个非负整数，按照文章的顺序，每个数（大小不超过1000）代表一个英文单词。

文章中两个单词是同一个单词，当且仅当它们对应的非负整数相同。

## 输出格式

包含一个整数，为软件需要查词典的次数。

**NOIP2015提高组 机器翻译**



# 单调队列

单调队列：

队列中元素之间的关系具有单调性，而且，队首和队尾都可以进行出队操作，只有队尾可以进行入队操作。

- 对于维护好的单调队列，对内元素是有序的，那么取出最大值（最小值）的复杂度是 $O(1)$
- 可以拿来优化DP

# 单调队列

## 例14、题目描述

有一个长为  $n$  的序列  $a$ ，以及一个大小为  $k$  的窗口。现在这个从左边开始向右滑动，每次滑动一个单位，求出每次滑动后窗口中的最大值和最小值。

输入格式

输入一共有两行，第一行有两个正整数  $n, k$ 。第二行  $n$  个整数，表示序列  $a$

输出格式

输出共两行，第一行为每次窗口滑动的最小值

第二行为每次窗口滑动的最大值

The array is  $[1, 3, -1, -3, 5, 3, 6, 7]$ , and  $k = 3$ .

Window position	Minimum value	Maximum value
[1 3 -1] -3 5 3 6 7	-1	3
1 [3 -1 -3] 5 3 6 7	-3	3
1 3 [-1 -3 5] 3 6 7	-3	5
1 3 -1 [-3 5 3] 6 7	-3	5
1 3 -1 -3 [5 3 6] 7	3	6
1 3 -1 -3 5 [3 6 7]	3	7

# 优先队列

- 最常见的堆，二叉堆，优先队列不是按照进入时间，而是按照某种优先级进行弹出的。C++中priority\_queue定义了一个以权值为优先级的堆。

```
#include<queue>  
priority_queue<数据类型> q;  
优先队列初始默认是权值大优先级高。
```

声明成权值小优先级高：

```
priority_queue<int,vector<int>,greater<int> > q;//小顶堆
```

# 优先队列

## 例15、合并果子

输入格式：

输入文件包括两行，第一行是一个整数  $n$  ( $1 \leq n \leq 10000$ )，表示果子的种类数。第二行包含  $n$  个整数，用空格分隔，第  $i$  个整数  $a_i$  ( $1 \leq a_i \leq 20000$ ) 是第  $i$  种果子的数目。

输出格式：

输出文件包括一行，这一行只包含一个整数，也就是最小的体力耗费值。输入数据保证这个值小于  $2^{31}$ 。

输入

3

1 2 9

输出

15

备注：

【数据规模】

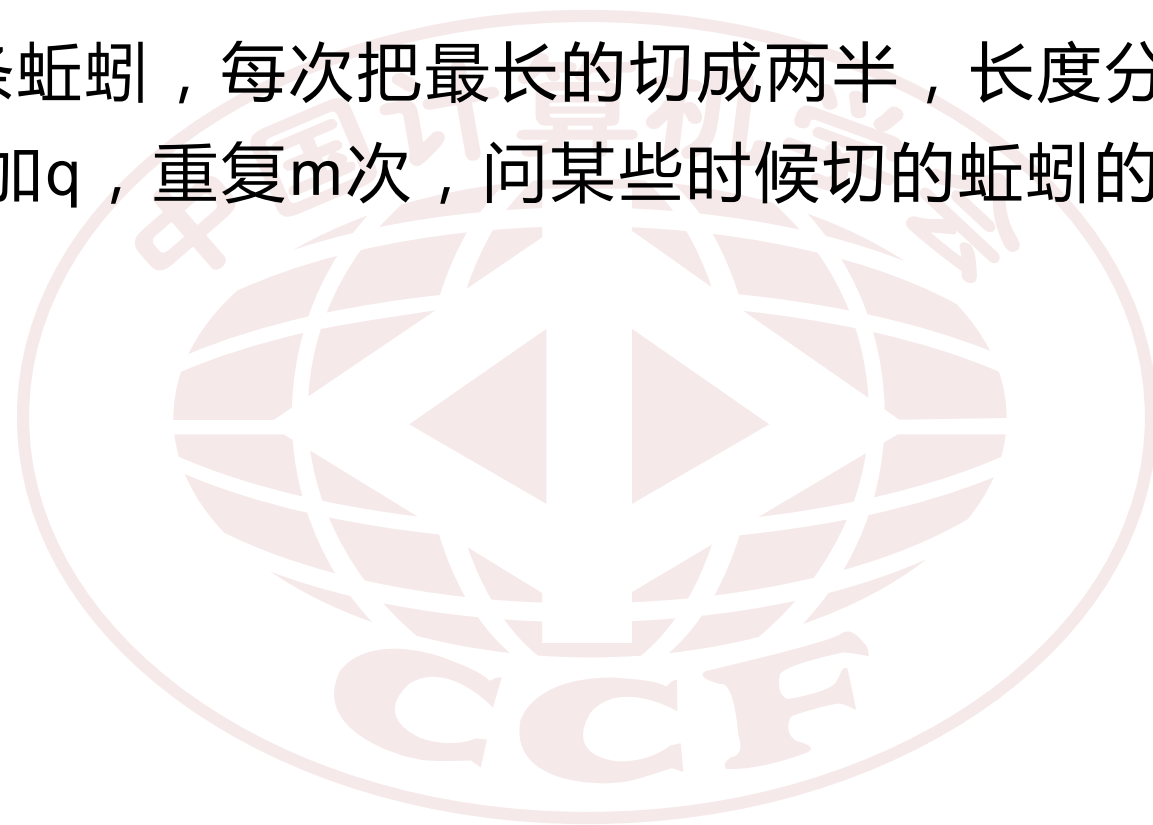
对于 30% 的数据，保证有  $n \leq 1000$ ；

对于 50% 的数据，保证有  $n \leq 5000$ ；

对于全部的数据，保证有  $n \leq 10000$ 。

## 例16、蚯蚓

- 题目大意：有 $n$ 条蚯蚓，每次把最长的切成两半，长度分别为 $\lfloor px \rfloor$ 和 $x - \lfloor px \rfloor$ ，其他的蚯蚓长度加 $q$ ，重复 $m$ 次，问某些时候切的蚯蚓的长度和最后的一些蚯蚓的长度。



# 单调栈和单调队列

- 单调队列：擅长维护区间最大/最小值，最小值对应着递增队列，最大值对应着递减队列
- 单调栈：擅长维护最近大于/小于关系，从左侧先入栈就是维护左侧最近关系，从右侧先入栈就是维护右侧最近关系



# 小结

- 数组
- 链表
- 栈
- 队列
- .....





中国计算机学会  
China Computer Federation



谢谢大家!