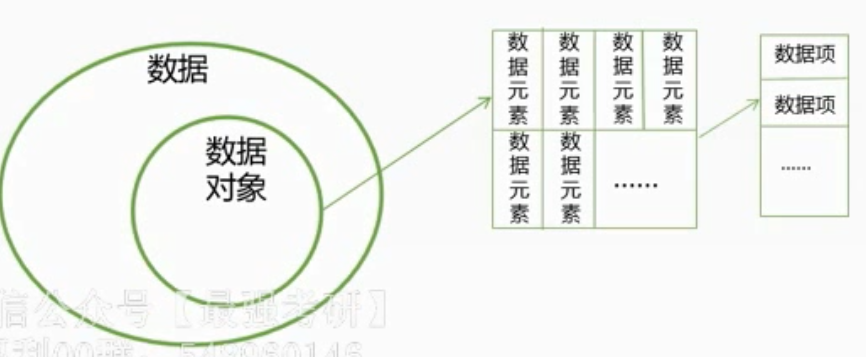
**数据结构**

### 一、绪论



1. 基本概念及术语
2. 数据：信息的载体，是描述客观事物属性的数、字符及所有能输入到计算机中并被计算机程序识别和处理的符号的集合。
3. 数据对象：具有相同性质的数据元素的集合，是数据的一个子集。
4. 数据元素：数据的基本单位，通常作为一个整体进行考虑和处理。
5. 数据项：构成数据元素的不可分割的最小单位。



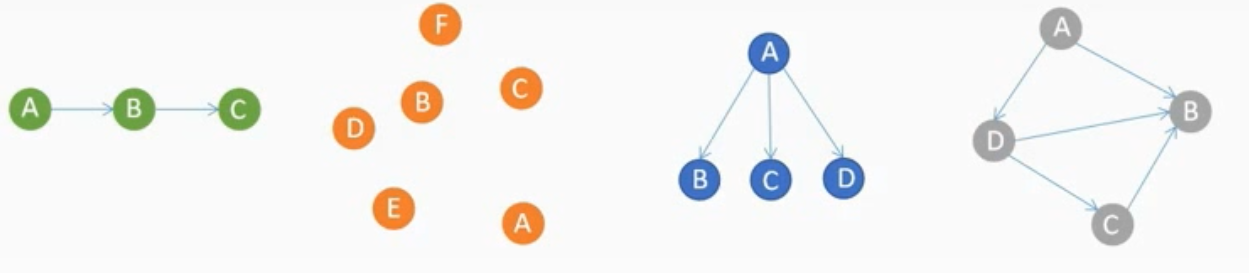


1. 结构：数据不是孤立，他们存在某种关系，这种相互关系即结构。
2. 数据结构是相互之间存在一种或多种特定关系的数据元素的集合。
3. 数据结构**三要素**：逻辑结构、物理结构、数据的运算

* 逻辑结构

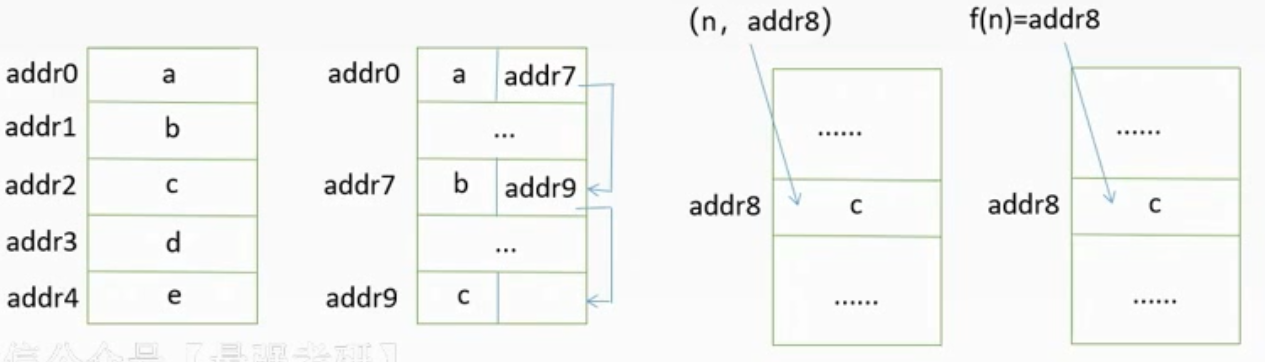
线性结构

非线性结构：集合、树形机构、图状结构



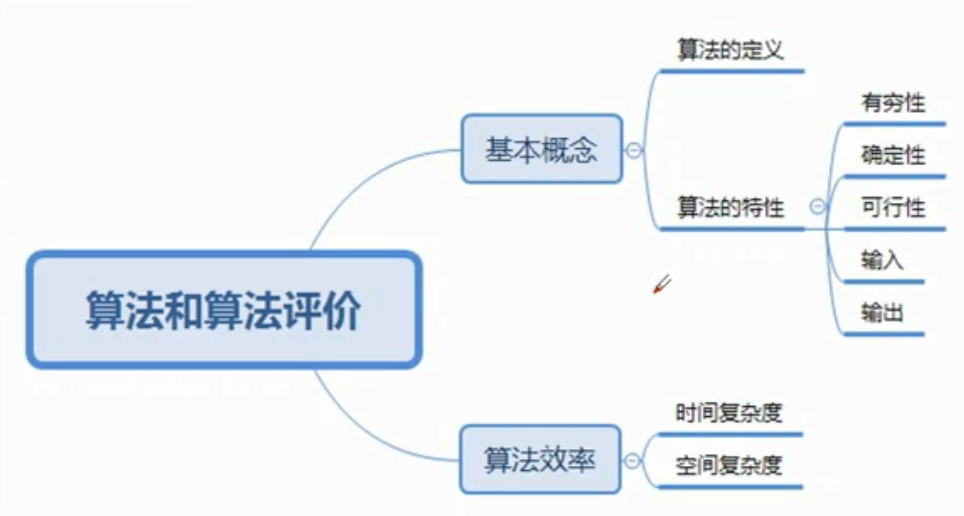
* 存储结构（物理结构）

顺序存储、链式存储、索引存储【其中有索引表(关键字和地址)】、散列存储（哈希存储）



* 数据的运算

运算包括运算的定义和实现，运算的定义针对逻辑结构，运算的实现针对存储结构。



1. 算法

* 对特定问题求解步骤的一种描述，他是指令的有限序列，其中的每条指令表示一个或多个操作。
* 特性：有穷性、可行性、确定性、输入、输出
* 程序 vs 算法

1. 程序是某种程序设计语言对算法的具体实现。
2. 算法必须是有穷的，程序可以是无穷的。
3. 算法必须是正确的，程序可以是错误的。
4. 算法可以用伪代码、程序语言等描述，程序只能用程序语言编写并执行。

* 算法效率的度量

正确性、可读性、健壮性、效率与存储量（时间负责度、空间复杂度）

1. 语句频度：该条语句可能重复执行的次数。
2. ：所有语句的频度之和，其中n为问题的规模。
3. 时间复杂度：,其中O表示与在时为同阶无穷大。

最坏时间复杂度、最好时间复杂度

平均时间复杂度：所有输入等概率的情况下时间复杂度的期望

规则：加法规则、乘法规则

通常采用**基本运算频度**（最深层循环所执行的语句的频度）来分析算法时间复杂度。

常见时间复杂度：

1. 空间复杂度：算法消耗的存储空间，记

除本身所用的指令、常数、变量和输入数据外，还需要一些对数据进行操作的工作单元和存储为实现算法所需的一些信息的辅助空间。

算法原地工作时，算法所需辅助空间为常数，。

### 二、线性表

1. 线性表

线性表是具有相同类型的n（n≥0）个元素的有限序列，其中n为表长，当n=0时，该表为空表。

* 特点：

a. 表中元素个数有限。

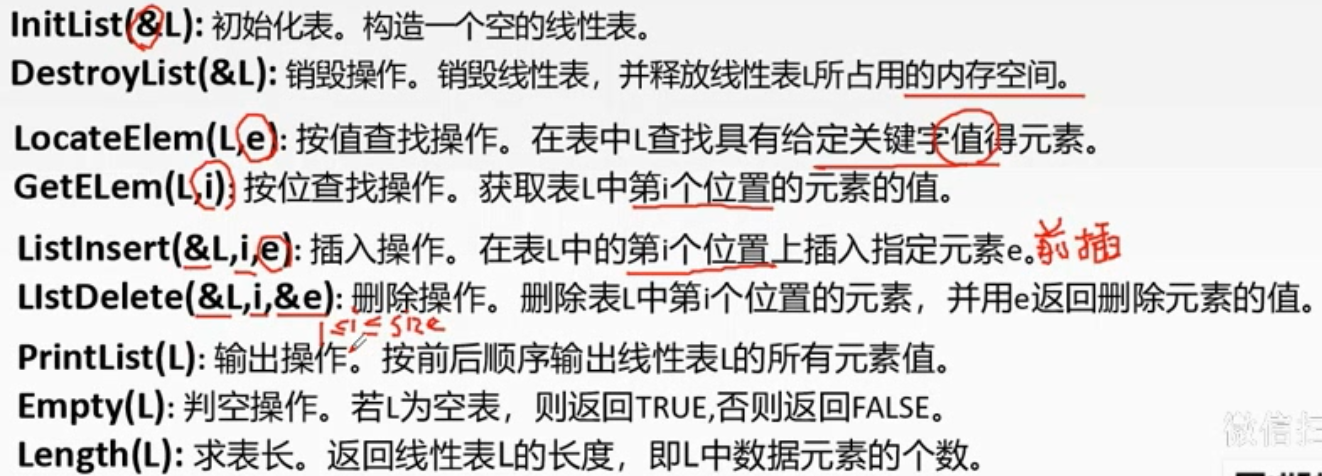
b. 表中元素具有逻辑上的顺序性，在序列中各个元素排列具有先后次序。

c. 表中元素的数据类型都相同，因此每个元素占有相同大小的存储空间。

d. 表中元素具有抽象性，即讨论元素间一对一的逻辑关系，而不考虑元素究竟表示的内容。

e. 线性表是一种逻辑结构，表示元素之间一对一相邻的关系。

* 九种基本操作：



1. 顺序表：线性表的顺序存储又称顺序表。

一组地址连续存放的存储单元依次存放线性表的元素，从而使**逻辑上相邻的两个元素在物理位置上也相邻**。

顺序表可以实现随机存取。

* 顺序表和数组的差别：

a. 数组下标从0开始，顺序表从1开始。

b. 数组容量是不可增加的，顺序表可以增加。

c. 数组可以是多维的，顺序表是一维的。

* 数组静态分配

|  |
| --- |
|  |

* 数组动态分配（也是顺序表，而非链式表）

|  |  |
| --- | --- |
|  |  |

* 基本操作
* 插入操作

|  |
| --- |
|  |

最好时间复杂度：

平均时间复杂度：，即

最坏时间复杂度：

* 删除操作

|  |
| --- |
|  |

最好时间复杂度：

平均时间复杂度：，即

最坏时间复杂度：

* 按值查找

|  |
| --- |
|  |

最好时间复杂度：

平均时间复杂度：，即

最坏时间复杂度：

1. 单链表：线性表的链式存储

单链表通过一组**任意**的存储单元来存储线性表中的数据元素。

【**比顺序存储占用了更多的空间，不能实现随机存取**

**为统一操作（插入第一个节点和插入其他节点统一，非空表与空表操作统一），单链表一般会设置一个头结点。**】

* 节点实现

|  |
| --- |
|  |

* 基本操作
* 头插法建立单链表（在已有单链表情况下进行插入，头插法的时间复杂度远低于尾插法）

|  |
| --- |
|  |

* 尾插法建立单链表

|  |
| --- |
|  |

* 按序号查找

|  |
| --- |
|  |

最坏时间复杂度：

* 按值查找

|  |
| --- |
|  |

最坏时间复杂度：

* 插入节点（前插法）

|  |
| --- |
|  |

扩展题：假设给定当前指针p，要在p前以时间复杂度插入一个节点，如何插入？

答：可先通过**后插法**插入节点s，后进行替换p和s的数值域。

* 删除节点

|  |
| --- |
|  |

* 求表长

|  |
| --- |
|  |

1. 特殊链表

* 双向链表
* 数据结构

|  |
| --- |
|  |

* 插入（在双链表的表尾进行插入时，不能保证和其他的统一性）

|  |
| --- |
|  |

* 删除（在双链表的表尾进行删除时，不能保证和其他的统一性）

|  |
| --- |
|  |

* 循环单链表（**插入和删除操作在每个位置都是一样的**）

【注意：循环单链表中仅设尾指针操作效率会比仅设头指针操作效率高。】

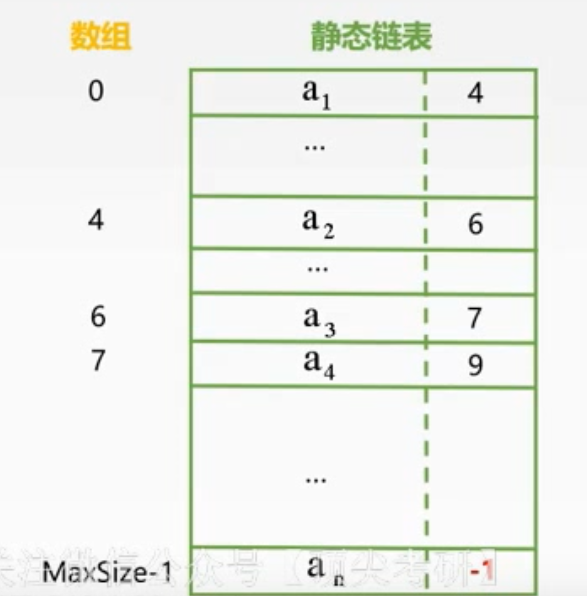
判空条件：L->next = L;

* 循环双链表（**插入和删除操作在每个位置也是一样的**）

判空条件：L->next = L;L->prior = L;

* 静态链表（用数组实现链式存储结构）

指针域存放的是指向下一个节点的数组下标（游标），最后一个节点的指针域存放-1。



* 数据结构

|  |
| --- |
|  |

1. 顺序表 vs 链表
2. 存取方式：顺序表可以实现顺序存取和随机存取；单链表只能实现顺序存取。
3. 逻辑结构和物理结构：顺序表逻辑相邻物理上也相邻，通过相邻表示逻辑关系；单链表逻辑相邻物理上不一定相邻，通过指针表示逻辑结构。
4. 查找：按值查找中，单链表和**无序**顺序表都为O(n)，按序查找中单链表为O(n)，顺序表为O(1)。
5. 内存空间：顺序存储无论静态分配还是非静态都需要预先分配合适的内存空间，静态分配时预分配空间太大会造成浪费，太小会造成溢出；动态分配时虽不会溢出但是扩充需要大量移动元素，操作效率低；链式存储在需要时分配节点空间即可，高效方便，但指针要使用额外空间。
6. 选择存储结构的依据：



1. 三个常用操作

* 求最值

顺序表和单链表时间复杂度都是O(n)

* 逆置

|  |
| --- |
|  |
|  |

数组时间复杂度为O(n)，空间复杂度为O(1)。

单链表时间复杂度为O(n)。

* 合并

|  |
| --- |
|  |
|  |

数组时间复杂度为O(n)。n为合并后的顺序表的长度。

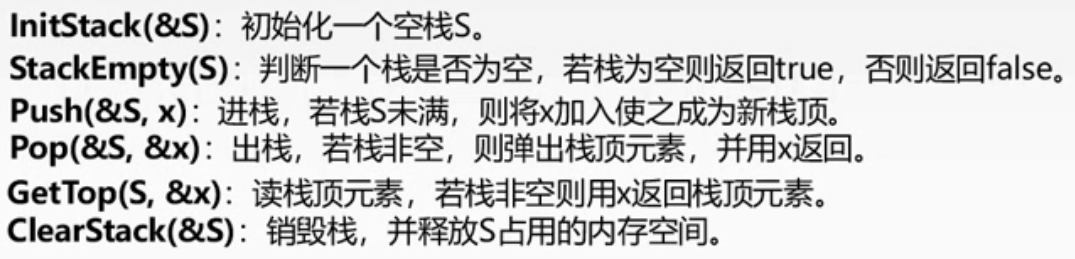
链表时间复杂度为O(n)。

### 三、栈和队列



1. 栈（LIFO）：只允许在一端进行插入或删除操作的线性表。

* 基本操作



* 顺序存储结构
* 基本数据结构

|  |
| --- |
|  |

栈空：s.top == -1;

栈长：s.top + 1；

栈满：s.top == MaxSize – 1；

* 初始化栈

|  |
| --- |
|  |

* 入栈

|  |
| --- |
|  |

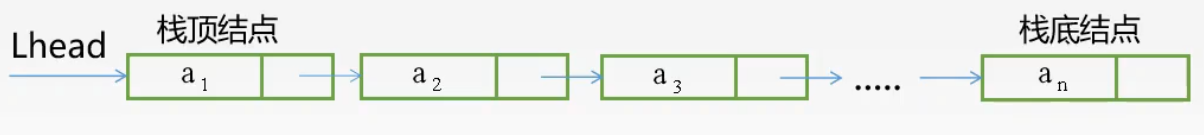
* 出栈

|  |
| --- |
|  |

* 共享栈：



* 链式存储结构（所有的操作都在表头执行）



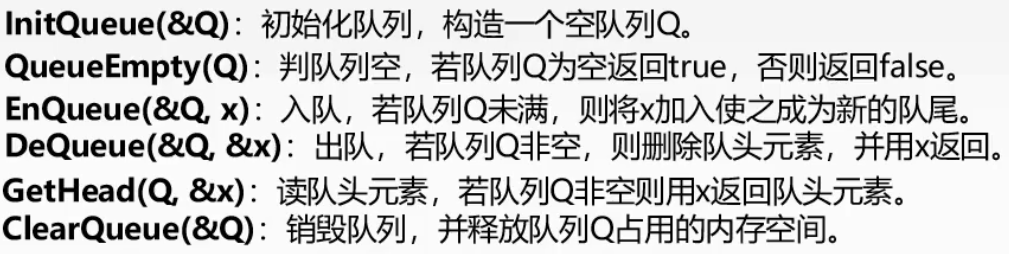
* 基本数据结构：

|  |
| --- |
|  |

1. 队列

先入先出FIFO，只允许在表的一端进行插入，表的另一端进行删除操作的线性表。（队头出队，队尾入队）

* 基本操作



为避免产生假溢出的状况，队列一般都使用循环队列。

循环队列：把存储队列的顺序队列在逻辑上视为一个环。

* 顺序存储结构
* 基本数据结构（front指向队首元素，rear执行队尾元素的下一个元素）

初始化时：规定front == rear == 0

|  |
| --- |
|  |

判空条件：Q.front == Q.rear

front指针移动：Q.front = (Q.front + 1) % MaxSize

rear指针移动：Q.rear = (Q.rear + 1) % MaxSize

队列长度：(Q.rear + MaxSize – Q.front) % MaxSize

判断队满的常用方法：

a. 牺牲一个存储单元：(Q.rear + 1) % MaxSize == Q.front【最常用的】

b. 增加一个变量代表元素的个数：Q.size == MaxSize

c. 增加一个tag标识，用tag标志是入队还是出队。tag = 0为出队，tag = 1为入队。

队空条件：Q.front == Q.rear && tag = 0

队满条件：Q.front == Q.rear && tag = 1

* 初始化队列

|  |
| --- |
|  |

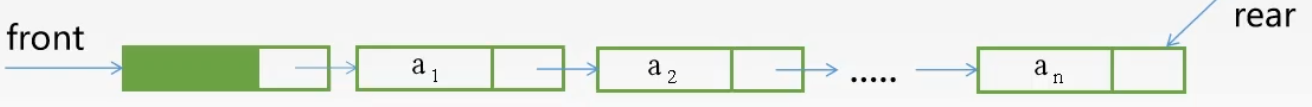
* 入队

|  |
| --- |
|  |

* 出队

|  |
| --- |
|  |

* 链式存储结构



* 基本数据类型

|  |
| --- |
|  |

* 初始化队列

|  |
| --- |
|  |

* 判断队列是否为空

|  |
| --- |
|  |

* 入队（尾插法）

|  |
| --- |
|  |

* 出队

|  |
| --- |
|  |

1. 双端队列

* 输出序列

栈的连续输出序列与正好是输入队列的逆序；队列的连续输出队列即输入队列。

队列的非连续输出序列与连续输出序列相同。

栈的非连续输出：出栈序列中每一个元素后面所有比它小的元素要组成一个递减序列。

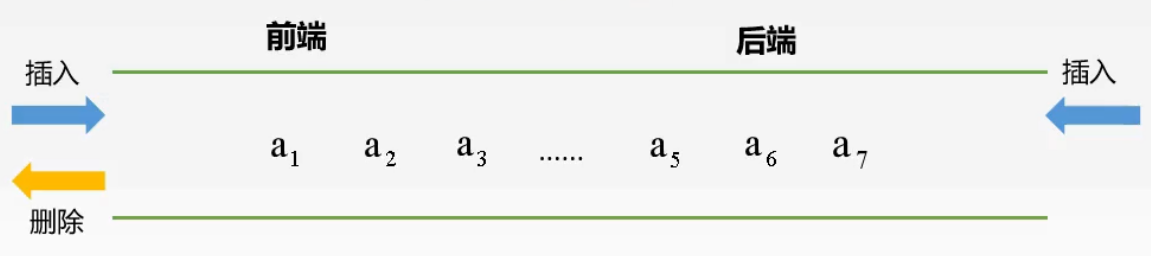
n个元素的栈的合法出栈序列的个数：

计算出栈序列中最后一个是k的出栈序列的个数。我们可以将出栈序列分为三个部分：，则以k为最后一个的合法出栈序列f(k-1)\*f(n-k)。则所有的合法出栈序列[最后一个分别为1~n]，等于f(n) = f(0)\*f(n-1) + f(1)\*f(n-2)+…+f(n-1)\*f(n)。而f(0) = f(1) = 1

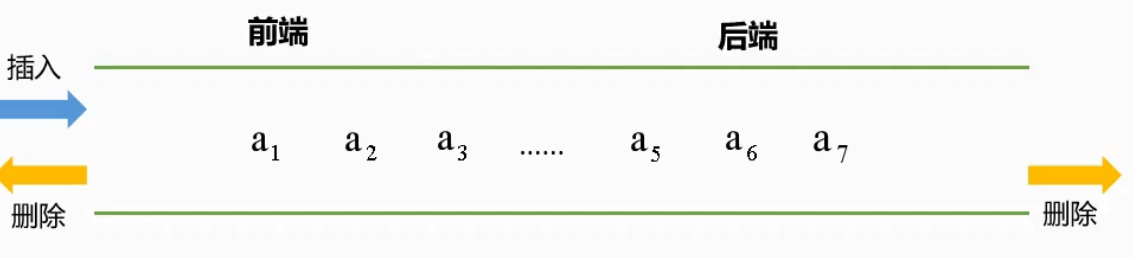
* 双端队列：允许两端都可以进行入队以及出队操作的队列

无论前端还是后端，先出的元素都在序列前，后出的元素都在序列后。

输出受限的双端队列（只能在一端进行删除）



输入受限的双端队列（只能在一端进行插入）



1. 数组（线性表的推广）

数组是由n (n ≥ 1)个相同类型的数据元素构成的有限序列，每个数据元素称为一个数组元素，每个元素受n个线性关系的约束，每个元素在n个线性关系中的序号称为该元素的下标，并称该数组为n维数组。

* 特点

数组具有维度。

数组的维度和维界是不可变的。

存储结构使用顺序存储。

* 存储地址计算

一维数组：

二维数组按行优先存储：

二维数组按列优先存储：

1. 特殊矩阵的压缩存储

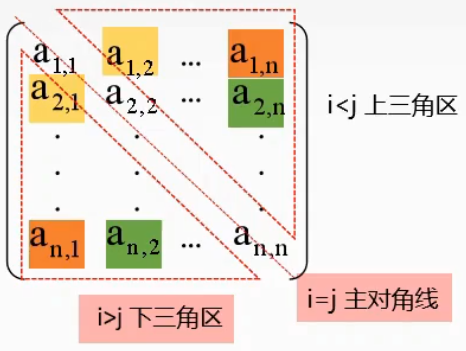
压缩存储：指多个**值相同**的元素只分配**一个**存储空间，对零元素不分配存储空间。

特殊矩阵：指具有许多相同矩阵元素或零元素，并且这些相同矩阵元素或零元素的分布有一定规律性的矩阵。

特殊矩阵的压缩存储：找出特殊矩阵中值相同的矩阵元素的分布规律，把那些呈现规律性分布、值相同的多个矩阵元素压缩到一个存储空间上。

* 对称矩阵（使用一维数组保存）

若对一个n阶方阵A[1…n][1…n]中的任意元素,则称其为对称矩阵。（只需要存储下三角区和主对角线即可：）

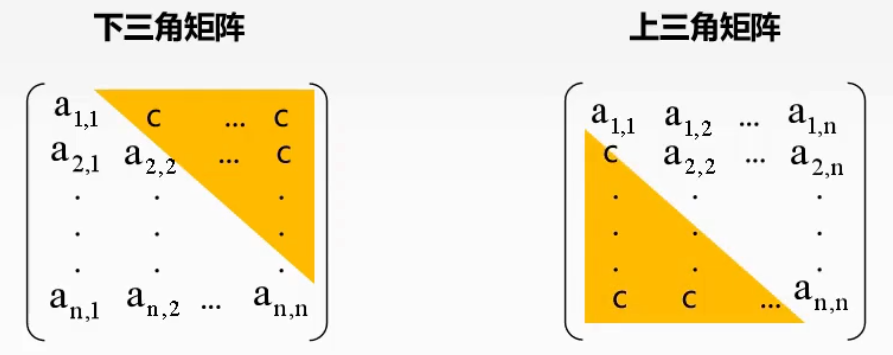


保存在数组下标

则

* 三角矩阵（使用一维数组保存）

若对一个n阶方阵A[1…n][1…n]中上（下）三角区元素均为同一常量，则称为下（上）三角矩阵。（存放元素个数，最后一个为c：）

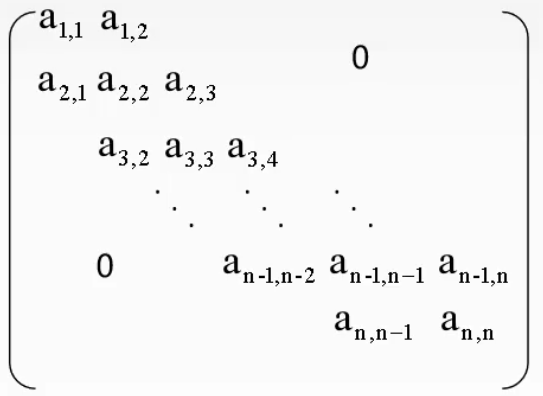


下三角矩阵：保存在数组下标

上三角矩阵：保存在数组下标

* 三对角矩阵（使用一维数组保存）

若对于一个n阶方阵A中的任意元素，当，有，则称为三对角矩阵。



保存在数组下标

若已知数组下标k，

* 稀疏矩阵（使用三元组保存）

矩阵元素个数s相对于矩阵中非零元素的个数t来说非常多，即的矩阵称为稀疏矩阵。

三元组（行标，列标，值）

缺点：稀疏矩阵压缩存储后失去了随机存储的特征。

1. 串