**Data Structure && Algorithm**

### 复杂度

##### 概念

###### 引言

复杂度：complexity，从字面意思理解就是用一种度量单位对复杂程度的一种描述。复杂度(Complexity, CPX)，指的是在给定样本中不同DNA 序列的总长度，是一件事物的复杂性可以用描写这事物所需的计算机语言的长度来衡量。[来自百度文科]

###### 动机（Motivate）

1. 复杂度的概念首先是由Kolmgorov提出来的。简明说就是一件事物的复杂性可以用描写这事物所用的计算机语言的长度来衡量。一般认为描述一件事物的计算机语言的长度越长，该事物就越复杂。70年代Lemple等在信息理论的研究中对随机序列复杂性给出了定义， 认为复杂性反映了一个时间序列随其长度的增加出现新模式的速率， 表现了序列接近随机的程度。80年代末期Kasper 等对随机序列Lem-Ziv意义下的复杂度进行了研究，提出了随机序列复杂性测度的具体算法。这套算法得到的复杂性测度被称为Kc复杂度，并且指出此算法比Lyapunov指数优越。由于复杂度分析方法对序列的长度要求不严格，因此在信号处理领域应用较广。

###### 意图（Intent）

1. [时间复杂度](https://baike.baidu.com/item/%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)是指在[计算机科学](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6)与工程领域完成一个[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95)所需要的时间，是衡量一个算法优劣的重要[参数](https://baike.baidu.com/item/%E5%8F%82%E6%95%B0)。[时间复杂度](https://baike.baidu.com/item/%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)越小，说明该[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95)效率越高，则该算法越有价值。
2. [空间复杂度](https://baike.baidu.com/item/%E7%A9%BA%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)是指[计算机科学](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6)领域完成一个[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95)所需要占用的[存储空间](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E7%A9%BA%E9%97%B4)，一般是输入参数的函数。它是[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95)优劣的重要度量指标，一般来说，[空间复杂度](https://baike.baidu.com/item/%E7%A9%BA%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)越小，算法越好。我们假设有一个[图灵机](https://baike.baidu.com/item/%E5%9B%BE%E7%81%B5%E6%9C%BA)来解决某一类语言的某一问题，设有X个字（word）属于这个问题，把X放入这个图灵机的输入端，这个图灵机为解决此问题所需要的工作带格子数总和称为**空间**。

###### 结构图（Structure）

###### 组成

##### How

###### 计算复杂度

1. 计算Kc之前，首先将要处理的序列进行粗粒化，在此对随机序列进行二值化处理，就是将序列的每一个点都由一个比特位来代表，于是就可以将所研究的信号信息粗粒化形成一个“0，1”序列。假设要处理的时间传输序列为{xi)(i=1，2，…，n)，求得平均值。如果xi≥平均值，令xi=1；如果xi<平均值，令xi=0；然后将这些0，1点组成原来序列的简化序列。
2. Kc的计算即找出序列x中所含的模式数，具体方法是通过一个“0，1”时间序列中的一串字符s(s1，s2，．．．，s。)后再加一个或一串字符Q，看字符Q是否属于SQv(SQv是SQ字符串中减去最后一个字符而得到的)，如果出现的字句在前面已经有过，即Q是sQ咛的一个子串，则该字符称为“复制”，认为这个过程没有新模式出现，把该字符加在串的后面，继续增长Q，再进行判断；若它没有出现过，那么对这个字符进行“插入”，“插入”时用一个“·”把前后字符分开，认为出现了一个新的模式：然后把最后一个“·”前面所有字符看成s，重新构造Q，再重复上述操作直到该序列结束并计算发现的模式数的总和。
3. 例如序列(0010)的复杂度可由下面的步骤得出：
   1. 第一个字符永远是插入0·；
   2. S=0，Q=0，SQ=00，SQv=0，Q属于字句SQv，0·0；
   3. s=0，Q=01，sQ=001，sQv=00，Q不属于字句sQv，0·01·；
   4. S=001，Q=0，SQ=0010，SQv=001，Q属于字句SQv，0·01·0。
   5. 于是得出该序列的模式数为3，即复杂度c(4)=3。符号序列0000…应是最简单的，0·000…，c(n)=2。另外，如010101…应是0.1.0101…，c(n)=3。
   6. 如上所述，用“·”将字符串分成了段，段的数目就定义为复杂度c(n)，几乎所有的“0，1”序列的c(n)都趋向于一个定值，即：limc(n)=b(n) = n/ln(n)
   7. 所以，b(n)是随机序列的渐进行为，可利用它使c(n)归一化，成为相对复杂度：C(n) = c(n)/b(n)
   8. 用这种函数来表达时间序列的复杂变化，可以看出完全随机的序列的C(n)趋于1，其他规律的和周期的运动则趋于0，而不完全随机序列的c(n)介于二者之间。

###### 实现要点

1. 要点
2. 优点
3. 缺点
4. 场景

##### Refer

###### 场景：

##### Summary

### 数据结构

##### 概念

###### 引言

数据结构是计算机存储、组织数据的方式。数据结构是指相互之间存在一种或多种特定关系的数据元素的集合。在任何问 题中，数据元素之间都不是孤立的，而是存在着一定的关系，这种关系称为结构 (Structure)。通常情况下，精心选择的数据结构可以带来更高的运行或者存储效率。数据结构往往同高效的检索算法和索引技术有关。

###### 动机（Motivate）

1. 数据结构时人根据数据在存储或者传输时的展现的特征的一种描述。不同的数据结构在传输或者存储的时间复杂度和空间复杂度是不一样的。
2. 数据的逻辑结构反映数据元素之间的逻辑关系的数据结构，其中的逻辑关系是指数据元素之间的前后件关系，而与他们在计算机中的存储位置无关。

###### 意图（Intent）

1. 根据不同的数据传输、存储需求对不同的数据结构进行正确的选择和使用已达到最优效果
2. 数据结构的主要任务是通过分析数据对象的结构特征，包括逻辑结构及数据对象之间的关系，然后把逻辑结构表示成计算机课实现的物理结构，从而便于计算机处理。

###### 结构图（Structure）

1. 数据（data）

是能被计算机处理的符号或符号集合，含义广泛，可理解为“原材料”。如字符、图片、音视频等。

1. 数据元素（data element）

是数据的基本单位。例如一张学生统计表。

1. 数据项（data item）、

组成数据元素的最小单位。例如一张学生统计表，有编号、姓名、性别、籍贯等数据项

1. 数据对象（data object）

是性质相同的数据元素的集合，是数据的一个子集。例如正整数N={1，2，3，····}。

1. 数据结构（data structure）

是数据的组织形式，数据元素之间存在的一种或多种特定关系的数据元素集合。

1. 数据类型（data type）

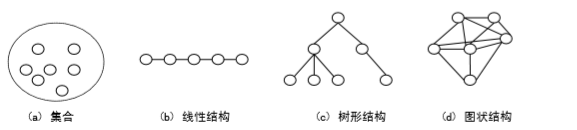
是按照数据值的不同进行划分的可操作性。在C语言中还可以分为原子类型和结构类型。原字类型是不可以再分解的基本类型，包括整型、实型、字符型等。结构类型是由若干个类型组合而成，是可以再分解的。

###### 组成

1. 数据的逻辑结构

逻辑结构（logical structure）是指在数据中数据元素之间的相互关系。

* 1. **集合结构**: 该结构中的数据元素除了存在“同属于一个集 合”的关系外，不存在任何其它关系。
  2. **线性结构**:线性的数据元素结构关系是一对一的，并且是一种先后的次序，就像a-b-c-d被一根线穿起来
  3. **树形结构**:树形的数据元素结构关系是一对多的，这就像公司的部门级别
  4. **图形结构**:图的元素结构关系是多对多的。就是我们常见的各大城市的铁路图



1. 数据的存储（物理）结构

存储结构（storage structure）也称为物理结构（physical structure），指的是数据的逻辑结构在计算机中的存储形式。数据的存储结构一般可以反映数据元素之间的逻辑关系，分为顺序存储和链式存储结构

* 1. 顺序存储结构：是把数据元素存放在一组存储地址连续的存储单元里，其数据元素间的逻辑关系和物理关系是一致的。
  2. 链式存储结构：是把数据元素存放在任意的存储单元里，这组存储单元可以是连续的，也可以是不连续的，数据元素的存储关系并不能反映其逻辑关系，因此需要借助指针来表示数据元素之间的逻辑关系

1. 抽象数据类型
   1. 抽象数据类型（Abstract Data Type）是描述具有某种逻辑关系的数据模型，并对在数学模型上进行的一组操作。
   2. 抽象数据类型描述的是一组逻辑上的的特性，与在计算机内部表示无关，计算机中的整数数据类型是一个抽象数据类型，不同处理器可能实现方法不同，但其逻辑特性相同
   3. “抽象”的意思是数据类型的数学抽象特性而不是指它们的实现方法。抽象数据类型体现了程序设计中的问题分解、抽象、信息隐藏等特性，可以把现实中的大问题分解为多个规模小且容易处理的小问题，然后建立起一个能被计算机处理的数据，并把每个功能模块的实现细节作为一个独立的单元，从而使具体实现过程隐藏起来。
2. 任何一个算法的设计取决于选定的数据逻辑结构，而算法的实现依赖于所采用的存储结构

##### How

###### 实现方法

###### 实现要点

1. 要点
   1. 数据结构呈线性关系。
      1. 必须存在唯一的被称为“第一个”的元素数据
      2. 必须存在唯一的被称为“最后一个”的数据元素
      3. 除第一个元素外，每个元素都有且只有一个前驱元素
      4. 除最后一个元素外，每个元素都有且只有一个后继元素
   2. 所有数据元素在同一个线性表中必须是相同的数据类型。
   3. 线性表（按存储结构划分）
      1. 顺序表（用顺序存储结构（依次存放）存储的线性表成为顺序表）
      2. 链表（用链式存储结构存储的线性表称为链表）
   4. 从数据类型和数据结构的关系
      1. 数据类型可以看 作是简单的数据结构。
      2. 数据的取值范围可以看作是数据元素的有限集合
      3. 而对数 据进行操作的集合可以看作是数据元素之间关系的集合。
2. 优点
3. 缺点
4. 场景

##### Refer

###### .NET 场景：

##### Summary

### 算法

##### 概念

###### 引言

在数据类型建立起来之后，就要对这些数据类型进行操作，建立起运算的集合即程序。运算的建立、方法好坏直接决定着计算机程序原型效率的高低。算法是对某一特定类型的问题的求解步骤的一种描述，是指令的 有限序列。其中的每条指令表示一个或多个操作

###### 动机（Motivate）

###### 意图（Intent）

1. 算法（algorithm）是解决特定问题求解步骤的描述，在计算机中表现为有限的操作序列。

###### 结构图（Structure）

###### 组成

1. 算法的五大特性
   1. **有穷性(Finity):** 是指算法在执行有限的步骤之后，自动结束而不是出现无限循环，并且每一个步骤在可接受的时间内完成。
   2. **确定性(Unambiguousness):** 是指算法执行的每一步骤在一定条件下只有一条执行路径，也就是相同输入只能有一个唯一的输出结果。
   3. **可行性(Realizability):** 是指算法每一步骤都必须可行，能够通过有限的执行次数完成。
   4. **输入(Input):** 是指算法具有零个或多个输入。
   5. **输出(Output):** 是指算法至少有一个或多个输出

##### How

###### 实现方法

1. 算法的评价标准
   1. 正确性(Correctness)

算法的执行结果应当满足预先规定的功能和性能的要求， 这是评价一个算法的重要也是基本的标准。算法的正确性还包括对于输入、 输出处理的明确而无歧义的描述。

* 1. 可读性(Readability)

算法主要是为了人阅读和交流，其次才是机器的执行。 所以，一个算法应当思路清晰、层次分明、简单明了、易读易懂。即使算法已转 变成机器可执行的程序，也需要考虑人能较好地阅读理解。同时，一个可读性强 的算法也有助于对算法中隐藏错误的排除和算法的移植。

* 1. 健壮性(Robustness)

一个算法应该具有很强的容错能力，当输入不合法的数 据时，算法应当能做适当的处理，使得不至于引起严重的后果。健壮性要求表明 算法要全面细致地考虑所有可能出现的边界情况和异常情况，并对这些边界情况 和异常情况做出妥善的处理，尽可能使算法没有意外的情况发生。

* 1. 运行时间(Running Time)

运行时间是指算法在计算机上运行所花费的时间， 它等于算法中每条语句执行时间的总和。对于同一个问题如果有多个算法可供选 择，应尽可能选择执行时间短的算法。一般来说，执行时间越短，性能越好

* 1. 占用空间(Storage Space)
     1. 占用空间是指算法在计算机上存储所占用的存储空 间，包括存储算法本身所占用的存储空间、算法的输入及输出数据所占用的存储 空间和算法在运行过程中临时占用的存储空间。算法占用的存储空间是指算法执 行过程中所需要的大存储空间，对于一个问题如果有多个算法可供选择，应尽 可能选择存储量需求低的算法。实际上，算法的时间效率和空间效率经常是一对 矛盾，相互抵触。我们要根据问题的实际需要进行灵活的处理，有时需要牺牲空 间来换取时间，有时需要牺牲时间来换取空间。
     2. 通常把算法在运行过程中临时占用的存储空间的大小叫算法的空间复杂度 (Space Complexity)。算法的空间复杂度比较容易计算，它主要包括局部变量所占 用的存储空间和系统为实现递归所使用的堆栈占用的存储空间。

###### 实现要点

1. 要点
2. 优点
3. 缺点
4. 场景

##### Refer

###### 数据结构和算法的关系：

1. 两者基友联系又有区别。联系是程序=算法+数据结构。数据结构是算法实现的基础，算法总是要依赖某种数据结构来实现的。算法的操作对象是数据结构。区别是数据结构关注的是数据的逻辑结构、存储结构有一集基本操作，而算法更多的是关注如何在数据结构的基本上解决实际问题。算法是编程思想，数据结构则是这些思想的基础。

##### Summary

### 集合(collection、set)

##### 概念

###### 引言

无序且不重复的数据放在一起，数据结构中最基础的一种，其他都是由该结构演变过来的。

###### 动机（Motivate）

1. 数据结构时人根据数据在存储或者传输时的展现的特征的一种描述。不同的数据结构在传输或者存储的时间复杂度和空间复杂度是不一样的。

###### 意图（Intent）

1. 根据不同的数据传输、存储需求对不同的数据结构进行正确的选择和使用已达到最优效果

###### 结构图（Structure）

###### 组成

1. **集合：**数据结构中的元素之间除了“同属一个集合” 的相互关系外，别无其他关系。
2. **线性结构**：数据结构中的元素存在一对一的相互关系。
3. **树形结构**：数据结构中的元素存在一对多的相互关系。
4. **图形结构**：数据结构中的元素存在多对多的相互关系。

##### How

###### 实现方法

1. 在很多场合都需要把数字转换成中文，我们就可以使用解释器来实现该功能，把给定的数字解释成符合语法规范的汉字表示法。
2. **Component**
   1. ）

###### 实现要点

1. 要点
   1. 使用Interpreter模式来表示文法规则，从而可以使用面向对象技巧方便地“扩展”文法。
   2. Interpreter模式比较适合简单的文法表示，对于复杂的文法表示，Interpreter模式会产生比较大的类层次结构，需要求助于语法分析生成器这样的标准工具。
2. 优点
   1. 易于改变和扩展文法。
   2. 每一条文法规则都可以表示为一个类，因此可以方便地实现一个简单的语言。
   3. 实现文法较为容易。在抽象语法树中每一个表达式节点类的实现方式都是相似的，这些类的代码编写都不会特别复杂，还可以通过一些工具自动生成节点类代码。
   4. 增加新的解释表达式较为方便。如果用户需要增加新的解释表达式只需要对应增加一个新的终结符表达式或非终结符表达式类，原有表达式类代码无须修改，符合“开闭原则”。
3. 缺点
   1. 对于复杂文法难以维护。在解释器模式中，每一条规则至少需要定义一个类，因此如果一个语言包含太多文法规则，类的个数将会急剧增加，导致系统难以管理和维护，此时可以考虑使用语法分析程序等方式来取代解释器模式。
   2. 执行效率较低。由于在解释器模式中使用了大量的循环和递归调用，因此在解释较为复杂的句子时其速度很慢，而且代码的调试过程也比较麻烦
4. 场景
   1. Interpreter模式的应用场合是Interpreter模式应用中的难点，只有满足“业务规则频繁变化，且类似的模式不断重复出现，并且容易抽象为语法规则的问题”才适合使用Interpreter模式。
   2. 当一个语言需要解释执行，并可以将该语言中的句子表示为一个抽象语法树的时候，可以考虑使用解释器模式（如XML文档解释、正则表达式等领域）
   3. 一些重复出现的问题可以用一种简单的语言来进行表达。
   4. 一个语言的文法较为简单.
   5. 当执行效率不是关键和主要关心的问题时可考虑解释器模式（注：高效的解释器通常不是通过直接解释抽象语法树来实现的，而是需要将它们转换成其他形式，使用解释器模式的执行效率并不高。

##### Refer

###### .NET 场景：

1. 正则表达式就是一个典型的解释器。ASP.NET中，把aspx文件转化为dll时，会对html语言进行处理，这个处理过程也包含了解释器的模式在里面。Interpreter模式其实有Composite模式的影子，但它们解决的问题是不一样的。

##### Summary

对

### 线性表(linear list)

##### 概念

###### 引言

严格按照插入顺序排列的数据,可以认为是集合按照顺序排列变化之后的结果。线性表是简单、基本、常用的数据结构。线性表是线性结构的抽象 (Abstract)，线性结构的特点是结构中的数据元素之间存在一对一的线性关系。这 种一对一的关系指的是数据元素之间的位置关系，即：（1）除第一个位置的数据 元素外，其它数据元素位置的前面都只有一个数据元素；（2）除后一个位置的 数据元素外，其它数据元素位置的后面都只有一个元素。也就是说，数据元素是 一个接一个的排列。因此，可以把线性表想象为一种数据元素序列的数据结构

###### 定义（Definition）

1. 线性表(List)是由 n(n≥0)个相同类型的数据元素构成的有限序列。
2. 有限:指的是线性表中的数据元素的个数是有限的
3. 有序:指的是线性表中的数据元素的位置是有顺序的
4. 相同类型:指的是线性表中的所有元素都属于同一种类型
5. 表示：L=(a1,a2,…,ai-1,ai, ai+1,…,an) L:List, L中包含n个数据元素，下标表示数据元素在线性表中的位置。a1是线性 表中第一个位置的数据元素，我们称作第一个元素。an是线性表中后一个位置 的数据元素，我们称作后一个元素。n为线性表的表长，n=0 时的线性表被称 为空表（Empty List）。
6. 线性表中的数据元素之间存在着前后次序的位置关系，将ai-1称为ai的直接前 驱，将ai称为ai+1的直接后继。除a1外，其余元素只有一个直接前驱，因为a1是第 一个元素，所以它没有前驱。除an外，其余元素只有一个直接后继，因为an是 后一个元素，所以它没有后继。

###### 行为（Action）

1. 任何位置插入元素
2. 在末尾追加数据元素
3. 删除任何位置元素
4. 获得任何位置元素
5. 读取任何位置元素值
6. 修改任何位置元素值
7. 定位数据元素所在位置
8. 清除表所有数据
9. 重新初始化表

###### 结构图（Structure）

###### 组成

###### 实现方法

1. int GetLength();
   1. 获取长度
   2. 初始条件：线性表存在；
   3. 操作结果：返回线性表中所有数据元素的个数。
2. void Clear();
   1. 清空，初始化
   2. 初始条件：线性表存在且有数据元素；
   3. 操作结果：从线性表中清除所有数据元素，线性表为空。
3. bool IsEmpty();
   1. 判断是否为空列表
   2. 初始条件：线性表存在；
   3. 操作结果：如果线性表为空返回 true，否则返回 false
4. void Append(T item);
   1. 附加数据元素
   2. 初始条件：线性表存在且未满；
   3. 操作结果：将值为 item 的新元素添加到表的末尾。
5. void Insert(T item, int index);
   1. 插入数据元素
   2. 线性表存在，插入位置正确()(1≤i≤n+1,n 为插入前的表长)。
   3. 在线性表的第 i 个位置上插入一个值为 item 的新元素，这样使得 原序号为 i,i+1,…,n 的数据元素的序号变为 i+1,i+2,…,n+1，插入后表长=原表长+1。
6. T Delete(int index);
   1. 删除数据元素
   2. 初始条件：线性表存在且不为空，删除位置正确(1≤i≤n,n 为删除前的表 长)。
   3. 操作结果：在线性表中删除序号为 i 的数据元素，返回删除后的数据元素。 删除后使原序号为 i+1,i+2,…,n 的数据元素的序号变为 i,i+1,…,n-1，删除后 表长=原表长-1。
7. T GetElement(int index);
   1. 获取数据元素
   2. 初始条件：线性表存在，所取数据元素位置正确（1≤i≤n，n 为线性表的表 长）；
   3. 操作结果：返回线性表中第 i 个数据元素。
8. int Locate(T value);
   1. 定位数据元素所在位置
   2. 初始条件：线性表存在。
   3. 操作结果：在线性表中查找值为 value 的数据元素，其结果返回在线性表中 首次出现的值为 value 的数据元素的序号，称为查找成功；否则，在线性表中未 找到值为 value 的数据元素，返回一个特殊值表示查找失败。

###### 实现要点

1. 要点
   1. 。
2. 优点
   1. 。
   2. 。
3. 缺点
4. 场景
   1. 。

##### Refer

###### .NET 场景：

1. 。

##### Summary

### 顺序表(sequence list)

##### 概念

###### 引言

在计算机内，保存线性表简单、自然的方式，就是把表中的元素一个接 一个地放进顺序的存储单元，这就是线性表的顺序存储(Sequence Storage)。

###### 定义（Definition）

1. 线性表的顺序存储是指在内存中用一块地址连续的空间依次存放线性表的数据元素，用这种方式存储的线性表叫顺序表(Sequence List)。

###### 行为（Action）

|  |  |  |
| --- | --- | --- |
| 操作 | 说明 | 时间复杂度 |
| Clear | 清空 | O(1) |
| Append(T item) | 追加 | O(1) |
| Insert(T item,int idx) | 前插 | O(n) |
| Delete(int idx) | 删除 | O(n) |
| GetElement(int idx) | 获取节点 | O(1) |
| Locate(T value) | 定位 | O(n) |
| GetLength() | 获取长度 | O(1) |

###### 结构图（Structure）



###### 组成

1. 假设顺序表中的每个数据元素占w个存储单元，设第i个数据元素的存储地址 为Loc(ai)，则有：Loc(ai)= Loc(a1)+(i-1)\*w 1≤i≤n
2. 式中的Loc(a1)表示第一个数据元素a1的存储地址，也是顺序表的起始存储地址， 称为顺序表的基地址(Base Address)。
3. 只要知道顺序表的基地址和每个 数据元素所占的存储单元的个数就可以求出顺序表中任何一个数据元素的存储 地址。并且，由于计算顺序表中每个数据元素存储地址的时间相同，所以顺序表 具有随机存取的特点。
4. C#语言中的数组在内存中占用的存储空间就是一组连续的存储区域，因此， 数组具有随机存取的特点。所以，数组天生具有表示顺序表的数据存储区域的特性。

###### 实现方法

###### 实现要点

1. 要点
   1. 。
2. 优点
   1. 顺序表是用地址连续的存储单元顺序存储线性表中的各个数据元素，逻辑上 相邻的数据元素在物理位置上也相邻。因此，在顺序表中查找任何一个位置上的数据元素非常方便
   2. 如果不需要经常在线性表中进行插入和删除，只是进行查找，那么，线性表应该顺序存储；
3. 缺点
   1. 在对顺序表进行插入和删除时， 需要通过移动数据元素来实现，影响了运行效率
4. 场景
   1. 。

##### Refer

###### .NET 场景：

1. 。

##### Summary

对

### 单链表(linked list)

##### 概念

###### 引言

在计算机内，不要求逻辑上相邻的元素在物理存储逻辑也相邻，这就是线性表的另外一种存储方式：链式存储(Linked Storage)。

###### 定义（Definition）

1. 逻辑上相邻，但是物理存储逻辑上不一定要相邻的线性表叫链表(Linked List)。
2. 链表不要求逻辑相邻的数据元素存储位置上也相邻，所以对链表进行插入和删除时不需要移动元素，但同时也失去了顺序表的可随机存储的优点。
3. 链表是用一组任意的存储单元来存储线性表中的数据元素，这组存储单元可以是连续的，也可以是不连续的，那么如何来表示数据元素之间的线性关系呢？为此，在存储元素时，除了要存储数据元素的本身信息外，还要存储与它相邻的数据元素的存储地址信息。
4. 链表中存储数据元素本身信息和相邻数据元素信息组成该元素的存储映像（Image），称为节点（Node）
5. 把存储元素本身信息的域叫结点的数据域（Data Domain）
6. 把存储元素相邻数据元素的存储地址信息的域叫做结点的引用域（Reference Domain）
7. 如何结点的引用域只存储该结点直接后继结点的存储地址，则该链表叫单链表（Singly Linked List）。把该引用域叫做next

###### 行为（Action）

|  |  |  |
| --- | --- | --- |
| 操作 | 说明 | 时间复杂度 |
| Clear | 清空 | O(1) |
| Append(T item) | 追加 | O(n) |
| Insert(T item,int idx) | 前插 | O(n) |
| InsertPost(T item,int idx) | 后插 | O(n) |
| Delete(int idx) | 删除 | O(n) |
| GetElement(int idx) | 获取节点 | O(n) |
| Locate(T value) | 定位 | O(n) |
| GetLength() | 获取长度 | O(n) |

###### 结构图（Structure）



###### 组成

1. data：数据域
2. next：后继引用域

###### 实现方法

###### 实现要点

1. 要点
   1. 单链表中的按值查找的主要运算是比较，比较的次数与给定值在表中的位置和表长有关。当给定值与第一个结点的值相等时，比较次数为1；当给定值与最后一个结点的值相等时，比较次数为n。所以，平均比较次数为(n+1)/2，时间复杂度为O（n）
   2. 单链表的建立与顺序表的建立不同，它是一种动态管理的存储结构，链表中的每个结点占用的存储空间不是预先分配，而是运行时系统根据需求而生成的。单链表的建立分为在头部插入结点建立单链表和在尾部插入结点建立单链表
2. 优点
   1. 如果线性表需要经常插入和删除，而不经常进行查找，则线性表应该链式存储。
3. 缺点
   1. 查找不方便，因为每一次都要从头部开始遍历，单词查询的时间复杂度是O(n)
4. 场景
   1. 。

##### Refer

###### .NET 场景：

1. 。

##### Summary

对

### 双向链表(doubly list)

##### 概念

###### 引言

单链表允许从一个结点直接访问它的后继结点，所以, 找直接后继结点的时间复杂度是O(1)。但是,要找某个结点的直接前驱结点，只能从表的头引用开始遍历各结点。如果某个结点的Next等于该结点，那么，这个结点就是该结点的直接前驱结点。也就是说，找直接前驱结点的时间复杂度是O(n)，n是单链表的长度。

###### 定义（Definition）

1. 我们在结点的引用域中保存直接前驱结点的地址而不是直接后继结点的地址。这样，找直接前驱结点的时间复杂度只有O(1)，但找直接后继结点的时间复杂度是O(n)。如果希望找直接前驱结点和直接后继结点的时间复杂度都是O(1)，那么，需要在结点中设两个引用域，一个保存直接前驱结点的地址，叫prev，一个直接后继结点的地址，叫next，这样的链表就是双向链表(Doubly Linked List)。
2. 双向链表结点的定义与单链表的结点的定义很相似，，只是双向链表多了一个字段prev

###### 行为（Action）

1. 由于双向链表的结点有两个引用，所以，在双向链表中插入和删除结点比单链表要复杂。
2. 双向链表中结点的插入分为在结点之前插入和在结点之后插入，插入操作要对四个引用进行操作

###### 结构图（Structure）



###### 组成

1. prev：前驱引用域
2. next：后继引用域
3. data：数据域

###### 实现方法

###### 实现要点

1. 要点
   1. 。
2. 优点
   1. 。
   2. 。
3. 缺点
4. 场景
   1. 。

##### Refer

###### .NET 场景：

##### Summary

### 循环列表(circular linked list)

##### 概念

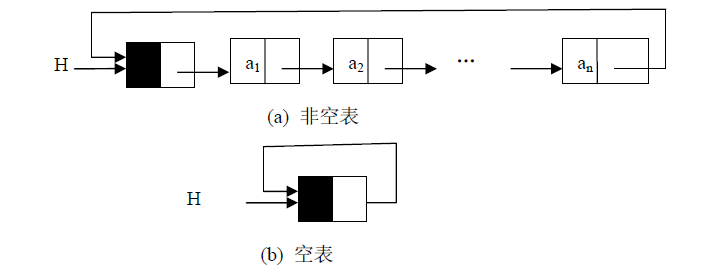
###### 引言

循环链表的基本操作与单链表大体相同，只是判断链表结束的条件并不是判断结点的引用域是否为空，而是判断结点的引用域是否为头引用，其它没有较大的变化。

###### 定义（Definition）

###### 行为（Action）

###### 结构图（Structure）



###### 组成

###### 实现方法

###### 实现要点

1. 要点
   1. 。
2. 优点
   1. 。
   2. 。
3. 缺点
4. 场景
   1. 。

##### Refer

###### .NET 场景：

##### Summary

### 栈(stack)

##### 概念

###### 引言

栈和队列是非常重要的两种数据结构，在软件设计中应用很多。栈和队列也是线性结构，线性表、栈和队列这三种数据结构的数据元素以及数据元素间的逻辑关系完全相同，差别是线性表的操作不受限制，而栈和队列的操作受到限制。栈的操作只能在表的一端进行，队列的插入操作在表的一端进行而其它操作在表的另一端进行，所以，把栈和队列称为操作受限的线性表。

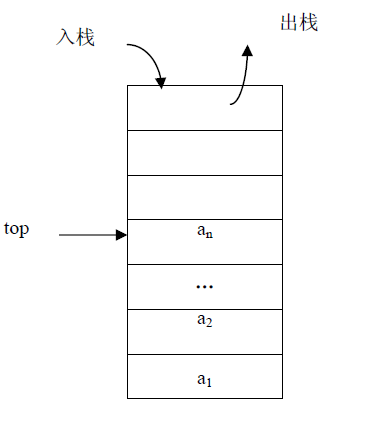
###### 定义（Definition）

1. 栈(Stack)是操作限定在表的尾端进行的线性表。
2. 栈通常记为：S= (a1,a2,…,an)
   1. S是英文单词stack的第1个字母。
   2. a1为栈底元素
   3. an为栈顶元素
3. 这n个数据元素按照a1,a2,…,an的顺序依次入栈，而出栈的次序相反，an第一个出栈，a1最后一个出栈。
4. 栈的操作是按照后进先出(Last In First Out，简称LIFO)或先进后出(First In Last Out，简称FILO)的原则进行的，因此，栈又称为LIFO表或FILO表

###### 行为（Action）

1. 栈的操作只能在表的一端进行
2. 由于栈只能在栈顶进行操作，所以栈不能在栈的任意一个元素处插入或删除元素。
3. 栈的操作是线性表操作的一个子集。
4. 栈的操作主要包括在栈顶插入元素和删除元素、取栈顶元素和判断栈是否为空等。

###### 结构图（Structure）



###### 组成

1. 栈的形式化定义为：S = (D, R)
   1. D是数据元素的有限集合
   2. R是数据元素之间关系的有限集合
2. 顺序栈 Sequence Stack
   1. 用一片连续的存储空间来存储栈中的数据元素，这样的栈称为顺序栈
   2. 类似于顺序表，用一维数组来存放顺序栈中的数据元素。栈顶指示器top设在数组下标为0的端，top随着插入和删除而变化，当栈为空时，top=-1
3. 链栈 Linked Stack
   1. 链栈通常用单链表来表示，它的实现是单链表的简化
   2. 链栈结点的结构与单链表结点的结构一样，为了操作方便，把栈顶设在链表的头部，并且不需要头结点。

###### 实现方法

###### 实现要点

1. 要点
   1. 。
2. 优点
   1. 。
   2. 。
3. 缺点
4. 场景
   1. 。

##### Refer

###### .NET 场景：

##### Summary

### 队列(queue)

##### 概念

###### 引言

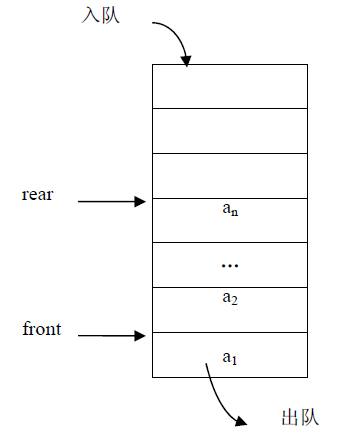
队列(Queue)是插入操作限定在表的尾部而其它操作限定在表的头部进行的线性表。

###### 定义（Definition）

1. 把进行插入操作的表尾称为队尾(Rear)
2. 把进行其它操作的头部称为队头(Front)
3. 当对列中没有数据元素时称为空对列(Empty Queue)
4. 队列通常记为：Q= (a1,a2,…,an)
   1. Q是英文单词queue的第1个字母。
   2. a1为队头元素
   3. an为队尾元素
   4. 这n个元素是按照a1,a2,…,an的次序依次入队的，出对的次序与入队相同，a1第一个出队，an最后一个出队。
5. 对列的操作是按照先进先出(First In First Out)或后进后出( Last In Last Out)的原则进行的，因此，队列又称为FIFO表或LILO表。

###### 行为（Action）

###### 结构图（Structure）



###### 组成

1. 队列的形式化定义为：Q = (D, R)
   1. D是数据元素的有限集合
   2. R是数据元素之间关系的有限集合
2. 顺序队列
   1. 用一片连续的存储空间来存储队列中的数据元素，这样的队列称为顺序队列(Sequence Queue)。
   2. 类似于顺序栈，用一维数组来存放顺序队列中的数据元素。
   3. 队头位置设在数组下标为0的端，用front表示；队尾位置设在数组的另一端，用rear表示
   4. front和rear随着插入和删除而变化。当队列为空时，front=rear=-1
3. 循环队列
4. 链队列(Linked Queue)
   1. 队列的另外一种存储方式是链式存储，这样的队列称为链队列(Linked Queue)。
   2. 同链栈一样，链队列通常用单链表来表示，它的实现是单链表的简化

###### 实现方法

###### 实现要点

1. 要点
   1. 。
2. 优点
   1. 。
   2. 。
3. 缺点
4. 场景
   1. 。

##### Refer

###### .NET 场景：

##### Summary

### 串(string)

##### 概念

###### 引言

字符串简称串，是一种特殊的线性表，其特殊性在于串中的数据元素是一个个的字符。

###### 定义（Definition）

1. 串(String)由n(n≥0)字符组成的有限序列
2. 串通常记为：S=” c1c2…cn ”(n ≥0)
   1. S是英文单词string的第1个字母，双引号作为串的定界符，用双引号引起来的字符序列是串值。
   2. ci（1≤i≤n）可以是字母、数字或其它字符，n为串的长度，当n=0时，称为空串(Empty String)。
   3. 串中任意个连续的字符组成的子序列称为该串的子串(Substring)。
   4. 包含子串的串相应地称为主串。
   5. 子串的第一个字符在主串中的位置叫子串的位置。如串s1”abcdefg”，它的长度是7，串s2”cdef”的长度是4，s2是s1的子串，s2的位置是3。
   6. 如果两个串的长度相等并且对应位置的字符都相等，则称这两个串相等
3. 由于串中的字符都是连续存储的，而在C#中串具有恒定不变的特性，即字符串一经创建，就不能将其变长、变短或者改变其中任何的字符。

###### 行为（Action）

###### 结构图（Structure）

###### 组成

###### 实现方法

###### 实现要点

1. 要点
   1. 。
2. 优点
   1. 。
   2. 。
3. 缺点
4. 场景
   1. 。

##### Refer

###### .NET 场景：

##### Summary

### 数组(array)

##### 概念

###### 引言

数组是一种常用的数据结构，可以看作是线性表的推广。数组作为一种数据结构，其特点是结构中的数据元素可以是具有某种结构的数据，甚至可以是数组，但属于同一数据类型。数组在许多高级语言里面都被作为固定类型来使用。

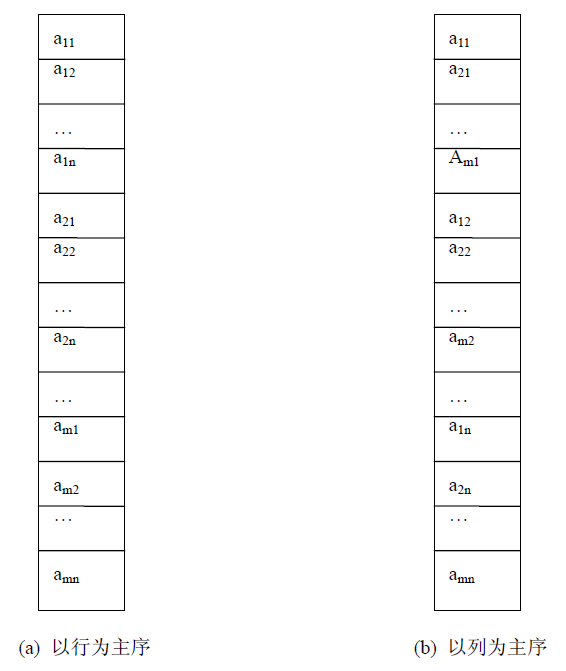
###### 定义（Definition）

1. 数组是n(n≥1)个相同数据类型的数据元素的有限序列
2. 一维数组可以看作是一个线性表，二维数组可以看作是“数据元素是一维数组”的一维数组，三维数组可以看作是“数据元素是二维数组”的一维数组，依次类推。
3. 数组是一个具有固定格式和数量的数据有序集，每一个数据元素通过唯一的下标来标识和访问。
4. 通常，一个数组一经定义，每一维的大小及上下界都不能改变。所以，在数组上不能进行插入、删除数据元素等操作。

###### 行为（Action）

1. 取值操作：给定一组下标，读其对应的数据元素；
2. 赋值操作：给定一组下标，存储或修改与其对应的数据元素；
3. 清空操作：将数组中的所有数据元素清除；
4. 复制操作：将一个数组的数据元素赋给另外一个数组；
5. 排序操作：对数组中的数据元素进行排序，这要求数组中的数据元素是可排序的；
6. 反转操作：反转数组中数据元素的顺序。

###### 结构图（Structure）



###### 组成

1. 数组的内存映像
2. 通常，采用顺序存储结构来存储数组中的数据元素，因为数组中的元素要求连续存放。
3. 本质上，计算机的内存是一个一维数组，内存地址就是数组的下标。
4. 所以，对于一维数组，可根据数组元素的下标得到它的存储地址，也可根据下标来访问一维数组中的元素。
5. 而对于多维数组，需要把多维的下标表达式转换成一维的下标表达式。当行列固定后，要用一组连续的存储单元存放数组中的元素，有一个次序约定问题，这产生了两种存储方式：一种是以行序为主序（先行后列）的顺序存放，另一种是以列序为主序（先列后行）的顺序存放。
6. 数组的存放地址
   1. 当以行序为主序进行存储时，设数组的基址是Loc(a11)，每个数据元素占w个存储单元，则a11的物理地址可由下式计算：
      1. Loc(aij)= Loc(a11)+((i-1)\*n+j-1)\*w
      2. 这是因为数组元素aij的前面有i-1行，每一行有n个数据元素，在第i行中aij的前面还有j-1个元素。
   2. 当以列序为主序进行存储时，则a11的物理地址可由下式计算
      1. Loc(aij)= Loc(a11)+((j-1)\*m+i-1)\*w
      2. 这是因为数组元素aij的前面有j-1列，每一列有m个数据元素，在第j列中aij的前面还有i-1个元素。
7. 由以上的公式可知，数组元素的存储位置是其下标的线性函数，一旦确定了数组各维的长度，就可以计算任意一个元素的存储地址，并且时间相等。所以，存取数组中任意一个元素的时间也相等，因此，数组是一种随机存储结构。

###### 实现方法

###### 实现要点

1. 要点
   1. 。
2. 优点
   1. 。
   2. 。
3. 缺点
4. 场景
   1. 。

##### Refer

###### .NET 场景：

##### Summary

### 树(tree)

##### 定义

###### 引言

###### 动机（Motivate）

1. 数据结构时人根据数据在存储或者传输时的展现的特征的一种描述。不同的数据结构在传输或者存储的时间复杂度和空间复杂度是不一样的。
2. 数据的逻辑结构反映数据元素之间的逻辑关系的数据结构，其中的逻辑关系是指数据元素之间的前后件关系，而与他们在计算机中的存储位置无关。

###### 意图（Intent）

1. 根据不同的数据传输、存储需求对不同的数据结构进行正确的选择和使用已达到最优效果

###### 结构图（Structure）

###### 组成

1. **集合：**数据结构中的元素之间除了“同属一个集合” 的相互关系外，别无其他关系。
2. **线性结构**：数据结构中的元素存在一对一的相互关系。
3. **树形结构**：数据结构中的元素存在一对多的相互关系。
4. **图形结构**：数据结构中的元素存在多对多的相互关系。

##### 相关术语

###### 实现方法

###### 实现要点

##### 逻辑表示

###### 实现方法

###### 实现要点

##### 基本操作

###### 实现方法

###### 实现要点

##### 使用场景

###### .NET 场景：

##### 总结

### 二叉树(tree)

##### 定义

###### 引言

###### 动机（Motivate）

1. 数据结构时人根据数据在存储或者传输时的展现的特征的一种描述。不同的数据结构在传输或者存储的时间复杂度和空间复杂度是不一样的。
2. 数据的逻辑结构反映数据元素之间的逻辑关系的数据结构，其中的逻辑关系是指数据元素之间的前后件关系，而与他们在计算机中的存储位置无关。

###### 意图（Intent）

1. 根据不同的数据传输、存储需求对不同的数据结构进行正确的选择和使用已达到最优效果

###### 结构图（Structure）

###### 组成

1. **集合：**数据结构中的元素之间除了“同属一个集合” 的相互关系外，别无其他关系。
2. **线性结构**：数据结构中的元素存在一对一的相互关系。
3. **树形结构**：数据结构中的元素存在一对多的相互关系。
4. **图形结构**：数据结构中的元素存在多对多的相互关系。

##### 二叉树的性质

###### 实现方法

###### 实现要点

##### 二叉树的存储结构

###### 实现方法

###### 实现要点

1. 要点
2. 优点
3. 缺点
4. 场景

##### 二叉链表存储结构的类实现

###### 实现方法

###### 实现要点

##### 二叉树的遍历

##### 使用场景

###### .NET 场景：

##### 总结

### 树与森林(tree)

##### 树、森林与二叉树的转换

###### 引言

###### 动机（Motivate）

1. 数据结构时人根据数据在存储或者传输时的展现的特征的一种描述。不同的数据结构在传输或者存储的时间复杂度和空间复杂度是不一样的。
2. 数据的逻辑结构反映数据元素之间的逻辑关系的数据结构，其中的逻辑关系是指数据元素之间的前后件关系，而与他们在计算机中的存储位置无关。

###### 意图（Intent）

1. 根据不同的数据传输、存储需求对不同的数据结构进行正确的选择和使用已达到最优效果

###### 结构图（Structure）

###### 组成

1. **集合：**数据结构中的元素之间除了“同属一个集合” 的相互关系外，别无其他关系。
2. **线性结构**：数据结构中的元素存在一对一的相互关系。
3. **树形结构**：数据结构中的元素存在一对多的相互关系。
4. **图形结构**：数据结构中的元素存在多对多的相互关系。

##### 树和森林的遍历

###### 实现方法

###### 实现要点

##### 使用场景

###### .NET 场景：

##### 总结

### 哈夫曼树(tree)

##### 定义

###### 引言

###### 动机（Motivate）

1. 数据结构时人根据数据在存储或者传输时的展现的特征的一种描述。不同的数据结构在传输或者存储的时间复杂度和空间复杂度是不一样的。
2. 数据的逻辑结构反映数据元素之间的逻辑关系的数据结构，其中的逻辑关系是指数据元素之间的前后件关系，而与他们在计算机中的存储位置无关。

###### 意图（Intent）

1. 根据不同的数据传输、存储需求对不同的数据结构进行正确的选择和使用已达到最优效果

###### 结构图（Structure）

###### 组成

1. **集合：**数据结构中的元素之间除了“同属一个集合” 的相互关系外，别无其他关系。
2. **线性结构**：数据结构中的元素存在一对一的相互关系。
3. **树形结构**：数据结构中的元素存在一对多的相互关系。
4. **图形结构**：数据结构中的元素存在多对多的相互关系。

##### 哈夫曼树的实现

###### 实现方法

###### 实现要点

##### 哈夫曼编码

###### 实现方法

###### 实现要点

1. 要点

##### 使用场景

###### .NET 场景：

##### 总结

### 图(map)

##### 定义

###### 图的定义

###### 图的基本术语

###### 图的基本操作

##### 图的存储结构

###### 邻接矩阵

###### 邻接表

##### 图的遍历

###### 深度优先的遍历

###### 广度优先的遍历

##### 图的应用

###### 最小生成树

###### 最短路径

###### 拓扑排序

##### 使用场景

###### .NET 场景：

##### 总结

### 排序(order)

##### 定义

###### 引言

###### 动机（Motivate）

1. 数据结构时人根据数据在存储或者传输时的展现的特征的一种描述。不同的数据结构在传输或者存储的时间复杂度和空间复杂度是不一样的。
2. 数据的逻辑结构反映数据元素之间的逻辑关系的数据结构，其中的逻辑关系是指数据元素之间的前后件关系，而与他们在计算机中的存储位置无关。

###### 意图（Intent）

1. 根据不同的数据传输、存储需求对不同的数据结构进行正确的选择和使用已达到最优效果

###### 结构图（Structure）

###### 组成

1. **集合：**数据结构中的元素之间除了“同属一个集合” 的相互关系外，别无其他关系。
2. **线性结构**：数据结构中的元素存在一对一的相互关系。
3. **树形结构**：数据结构中的元素存在一对多的相互关系。
4. **图形结构**：数据结构中的元素存在多对多的相互关系。

##### 简单的排序方法

###### 直接插入排序

###### 冒泡排序

###### 简单选择排序

##### 快速排序

##### 堆排序

##### 归并排序

##### 基数排序

###### 多关键码排序

###### 链式基数排序

##### 各种排序方法的比较与讨论

##### 使用场景

###### .NET 场景：

##### 总结

### 查找(search)

##### 基本概念和术语

###### 引言

###### 动机（Motivate）

1. 数据结构时人根据数据在存储或者传输时的展现的特征的一种描述。不同的数据结构在传输或者存储的时间复杂度和空间复杂度是不一样的。
2. 数据的逻辑结构反映数据元素之间的逻辑关系的数据结构，其中的逻辑关系是指数据元素之间的前后件关系，而与他们在计算机中的存储位置无关。

###### 意图（Intent）

1. 根据不同的数据传输、存储需求对不同的数据结构进行正确的选择和使用已达到最优效果

###### 结构图（Structure）

###### 组成

1. **集合：**数据结构中的元素之间除了“同属一个集合” 的相互关系外，别无其他关系。
2. **线性结构**：数据结构中的元素存在一对一的相互关系。
3. **树形结构**：数据结构中的元素存在一对多的相互关系。
4. **图形结构**：数据结构中的元素存在多对多的相互关系。

##### 静态查找表

###### 顺序查找

###### 有序表的折半查找

###### 索引查找

##### 动态查找表

##### 哈希表

###### 哈希表的基本概念

###### 常用的哈希函数构造方法

###### 处理冲突的方法

##### 使用场景

###### .NET 场景：

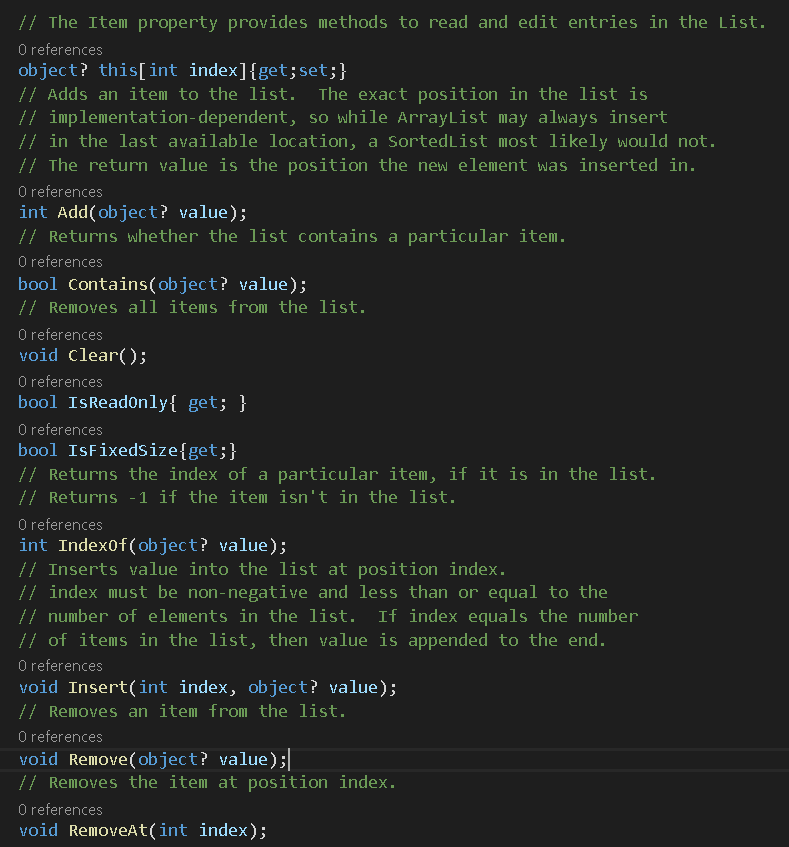
##### 总结

### C#数据结构

##### 概念

###### C#线性表

1. Ilist
   1. 接口表示一个集合，该集合中的项可被排序且可按索引任意访问。
   2. 是所有线性表的基接口
   3. 分成三类
      1. 只读的：不能修改表中的项，也不能在表中插入或删除项
      2. 大小不变：可以修改表中的项，但是不能在表中插入或删除项
      3. 大小可变：既可以修改表中的项，也可以在表中插入或删除项
   4. IList接口声明



1. ICollection

###### 栈和队列

1. 栈和队列是计算机中常用的两种数据结构，是操作受限的线性表。
2. 栈的插入和删除等操作都在栈顶进行，它是先进后出的线性表。
3. 队列的删除操作在队头进行，而插入、查找等操作在队尾进行，它是先进先出的线性表。
4. 与线性表一样，栈和队列有两种存储结构，顺序存储的栈称为顺序栈，链式存储的栈称为链栈。顺序存储的队列称为顺序对列，链式存储的队列称为链队列。
5. 为解决顺序队列中的假溢出问题，采用循环顺序队列，但出现队空和队满的判断条件相同的问题，判断条件都是：front==rear。采用少用一个存储单元来解决该问题。此时，队满的判断条件是：(rear+1)%maxsize==front，判断队空的条件是：rear==front。
6. 栈适合于具有先进后出特性的问题，如括号匹配、表达式求值等问题；
7. 队列适合于具有先进先出特性的问题，如排队等问题。

###### C#字符串

1. 在C#中，一个String表示一个恒定不变的字符序列集合。
2. String类型是封闭类型，所以，它不能被其它类继承，而它直接继承自object。因此，String是引用类型，不是值类型，在托管堆上而不是在线程的堆栈上分配空间。
3. String类型还继承了IComparable、ICloneable、IConvertible、IComparable<string>、IEnumerable<char>、IEnumerable和IEquatable<string>等接口。String的恒定性指的是一个串一旦被创建，就不能将其变长、变短或者改变其中任何的字符。所以，当我们对一个串进行操作时，不能改变字符串，，串连接、串插入和串删除等操作的结果都是生成了新串而没有改变原串。
4. C#也提供了StringBuilder类型来支持高效地动态创建字符串。

###### C#数组

1. C#支持一维数组、多维数组及交错数组（数组的数组）。
2. Array还继承了ICloneable、IList、ICollection、IEnumerable等接口。
3. C#中的数组一般是0基数组（最小索引为0），这是为了和其它语言共享代码。C#也支持非0基数组。C#除了能创建静态数组外，还可以创建动态数组，这通过使用Array的静态方法CreateInstance方法来实现。

##### 原理

###### C#字符串

1. 在C#中，创建串不能用new操作符，而是使用一种称为字符串驻留的机制。这是因为C#语言将String看作是基元类型。
2. 基元类型是被编译器直接支持的类型，可以在源代码中用文本常量(Literal)来直接表达字符串。
3. 当C#编译器对源代码进行编译时，将文本常量字符串存放在托管模块的元数据中。
4. 而当CLR初始化时，CLR创建一个空的散列表，其中的键是字符串，值为指向托管堆中字符串对象的引用。当JIT编译器编译方法时，它会在散列表中查找每一个文本常量字符串。如果找不到，就会在托管堆中构造一个新的String对象（指向字符串），然后将该字符串和指向该字符串对象的引用添加到散列表中；如果找到了，不会执行任何操作。

###### C# 数组

1. 所有的数组类型都隐含继承自System.Array。Array是一个抽象类，本身又继承自System.Object。所以，数组总是在托管堆上分配空间，是引用类型。
2. 任何数组变量包含的是一个指向数组的引用，而非数组本身。当数组中的元素的值类型时，该类型所需的内存空间也作为数组的一部分而分配；当数组的元素是引用类型时，数组包含是只是引用。