数据结构

2024年7月8日 13:09

算法的质量：正确性、可读性、健壮性（鲁棒性/容错性）、时间效率和空间效率
计算机科学的恒等式：数据结构+算法=程序
数据结构包括：数据的存储结构和数据的逻辑结构

数据结构的基本概念：
数据：对客观事物的符号表示（数值数据、非数值数据{字符、图形、图像、音频、视频}）
数据元素：表示现实世界中一个实体的一组数据，是数据结构的基本组成单位
数据项：数据元素中有独立含义的不可分割的最小单位
数据对象：性质相同的数据元素的集合
数据结构：相互之间存在一种或多种特定关系的数据元素的集合

数据结构的种类：
集合结构:元素之间的关系是属于关系
线性结构：一个元素最多有一个前驱和一个后驱
树状结构：数据元素可以有多个后继者（一对多的关系）
图状结构：数据元素可以有多个前驱和后驱（多对多的关系）

数据的逻辑结构：
（D,S）定义：Data_Structure=(D,S):D表示数据元素的集合，S表示数据元素之间关系的集合

数据元素之间的关系表示：
数据元素之间的关系在计算机内存中有两种不同的表示方法，即顺序存储和链式存储
顺序存储：将数据元素按一定的规则存放在一组编号连续的存储单元中
链式存储：在数据元素后附加一个地址域，通过指示相邻元素的存储地址来体现数据元素的逻辑关系

线性数据结构的顺序存储就是将数据元素依次存放到一维数组中

算法和算法分析：

算法：对特定问题求解步骤的一种描述，是指令的有限序列

算法的描述工具有：自然语言、框图、PAD图、伪码、程序设计语言等

（在java中，算法一般用类的一个方法进行描述）

算法的时间效率分析：

同样的算法，书写程序的语言级别越高，执行效率越低，运行时间越长

分析时的做法：从算法中选取一种对于所研究的问题来说是基本操作的原操作，以该基本操作重复执行的次数为基准度量算法的时间效率。

统一规定：计算时间复杂度以最坏的情况为准。

算法的空间效率分析：

执行一个算法所需实现的存储空间包括3部分：

（1）程序指令占用的存储空间

（2）输入数据占用的存储空间

（3）实现数据处理任务所必须的辅助存储空间（除输入数据外所需要用到的辅助变量所占用的内存{基本数据类型的字节数}，常量阶的辅助变量个数空间复杂度为O（1））

线性表

2024年7月8日 16:18

定义：由n个类型相同的数据元素组成的有限序列
线性表中数据元素的个数n称为线性表的长度，前者是后者的直接前驱，后者是前者的直接后继。

链式存储——>链表：用一组任意的存储单元（编号可以不连续）存储线性表中的数据元素，需要在数据元素的后面附加一个地址域，存储它的后继元素的地址，指示它后继元素的存储位置。
优点：零碎的空间得以充分利用
结点：数据部分（data）+地址部分（next）
第一个结点的地址成为链表的首地址，用head表示
（为了便于实现基本操作，可以在第一个结点之前增加一个头结点，头结点的类型与其他结点一样，头结点的数据部分可以为空或存放线性表的元素个数）

栈和队列

2024年7月10日 10:33

栈和队列也是线性结构，线性表、栈和队列这3种数据结构的数据元素以及数据元素之间的逻辑关系完全相同，区别在于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；队列的插入操作只能在表的一端进行——>将栈和队列称为操作受限的线性表

栈的定义：

栈（stack）是只允许在表的尾端进行插入和删除操作的线性表（在插入数据元素时，新插入的数据元素e只能处于线性表的表尾，在删除元素时，只能删除线性表的表尾元素）

栈顶和栈底：
表尾端为栈顶（top），表头端为栈底（bottom）

进栈和出栈：
栈的插入操作成为进栈或入栈（push），栈的删除操作称为出栈或退栈（pop）

LIFO：last in first out

在java语言的层面上讨论顺序栈（栈的顺序存储）时，栈的顺序存储是将栈的数据元素自栈底至栈顶放在一维数组中，同时，附设一个top指示器指向栈顶元素，top的值就是栈顶元素在数组中的下标，也可以让top指示器指向栈顶元素的下一个位置。

队列的定义：
队列是一种只允许在表尾端进行插入，在表头端进行删除的线性表

队头（front）即为表头，队尾（end）即为表尾

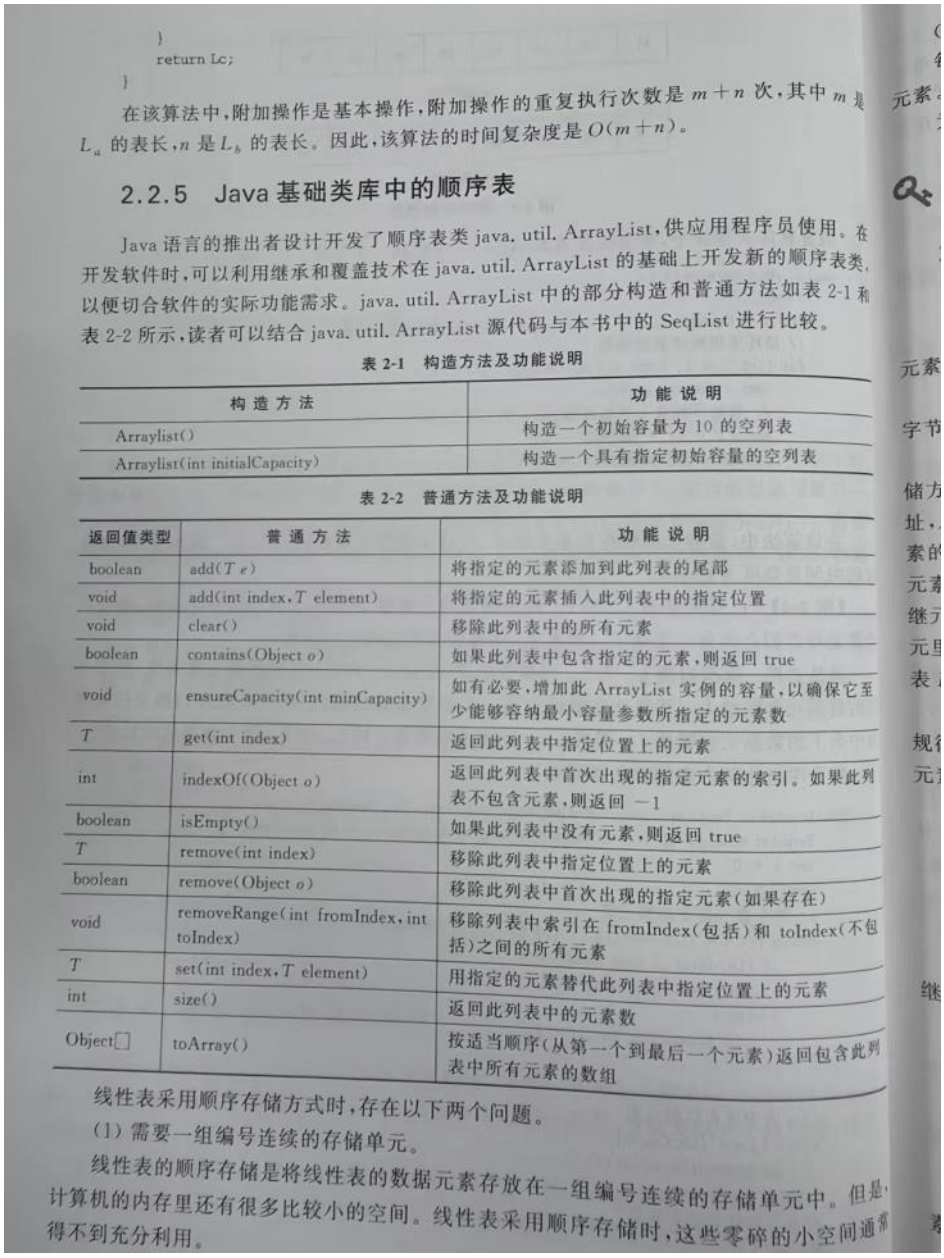
FIFO：first in first out

头结点的地址用front来表示，称front为队头引用
为了提高插入操作的效率，附设一个引用rear指示队尾元素的地址，称rear为队尾引用

Java基础类库

2024年7月10日　20:54

顺序表：



```
    }
    return Lc;
}
```

在该算法中，附加操作是基本操作，附加操作的重复执行次数是 $m+n$ 次，其中 m 是 L_a 的表长，n 是 L_b 的表长。因此，该算法的时间复杂度是 $O(m+n)$。

2.2.5 Java 基础类库中的顺序表

Java 语言的推出者设计开发了顺序表类 java. util. ArrayList，供应用程序员使用。在开发软件时，可以利用继承和覆盖技术在 java. util. ArrayList 的基础上开发新的顺序表类，以便切合软件的实际功能需求。java. util. ArrayList 中的部分构造和普通方法如表 2-1 和表 2-2 所示，读者可以结合 java. util. ArrayList 源代码与本书中的 SeqList 进行比较。

表 2-1　构造方法及功能说明

构造方法	功能说明
Arraylist()	构造一个初始容量为 10 的空列表
Arraylist(int initialCapacity)	构造一个具有指定初始容量的空列表

表 2-2　普通方法及功能说明

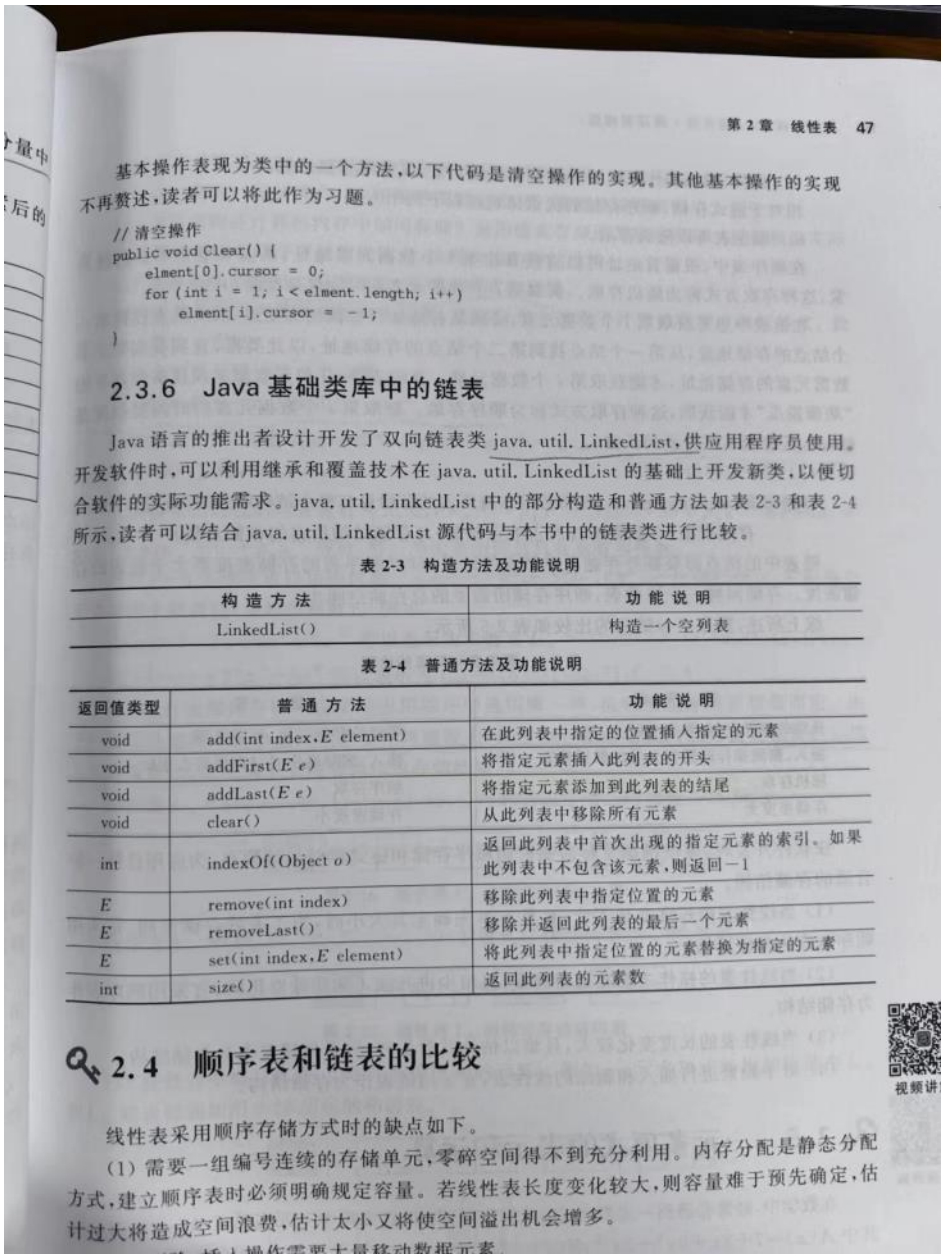
返回值类型	普通方法	功能说明
boolean	add(T e)	将指定的元素添加到此列表的尾部
void	add(int index, T element)	将指定的元素插入此列表中的指定位置
void	clear()	移除此列表中的所有元素
boolean	contains(Object o)	如果此列表中包含指定的元素，则返回 true
void	ensureCapacity(int minCapacity)	如有必要，增加此 ArrayList 实例的容量，以确保它至少能够容纳最小容量参数所指定的元素数
T	get(int index)	返回此列表中指定位置上的元素
int	indexOf(Object o)	返回此列表中首次出现的指定元素的索引。如果此列表不包含元素，则返回 −1
boolean	isEmpty()	如果此列表中没有元素，则返回 true
T	remove(int index)	移除此列表中指定位置上的元素
boolean	remove(Object o)	移除此列表中首次出现的指定元素(如果存在)
void	removeRange(int fromIndex, int toIndex)	移除列表中索引在 fromIndex(包括)和 toIndex(不包括)之间的所有元素
T	set(int index, T element)	用指定的元素替代此列表中指定位置上的元素
int	size()	返回此列表中的元素数
Object[]	toArray()	按适当顺序(从第一个到最后一个元素)返回包含此列表中所有元素的数组

线性表采用顺序存储方式时，存在以下两个问题。

(1) 需要一组编号连续的存储单元。

线性表的顺序存储是将线性表的数据元素存放在一组编号连续的存储单元中。但是，计算机的内存里还有很多比较小的空间。线性表采用顺序存储时，这些零碎的小空间通常得不到充分利用。

链表：



基本操作表现为类中的一个方法，以下代码是清空操作的实现。其他基本操作的实现不再赘述，读者可以将此作为习题。

```
// 清空操作
public void Clear() {
    elment[0].cursor = 0;
    for (int i = 1; i < elment.length; i++)
        elment[i].cursor = -1;
}
```

2.3.6　Java 基础类库中的链表

Java 语言的推出者设计开发了双向链表类 java.util.LinkedList，供应用程序员使用。开发软件时，可以利用继承和覆盖技术在 java.util.LinkedList 的基础上开发新类，以便切合软件的实际功能需求。java.util.LinkedList 中的部分构造和普通方法如表 2-3 和表 2-4 所示，读者可以结合 java.util.LinkedList 源代码与本书中的链表类进行比较。

表 2-3　构造方法及功能说明

构造方法	功能说明
LinkedList()	构造一个空列表

表 2-4　普通方法及功能说明

返回值类型	普通方法	功能说明
void	add(int index,*E* element)	在此列表中指定的位置插入指定的元素
void	addFirst(*E e*)	将指定元素插入此列表的开头
void	addLast(*E e*)	将指定元素添加到此列表的结尾
void	clear()	从此列表中移除所有元素
int	indexOf(Object *o*)	返回此列表中首次出现的指定元素的索引。如果此列表中不包含该元素，则返回－1
E	remove(int index)	移除此列表中指定位置的元素
E	removeLast()	移除并返回此列表的最后一个元素
E	set(int index,*E* element)	将此列表中指定位置的元素替换为指定的元素
int	size()	返回此列表的元素数

2.4　顺序表和链表的比较

线性表采用顺序存储方式时的缺点如下。

(1) 需要一组编号连续的存储单元，零碎空间得不到充分利用。内存分配是静态分配方式，建立顺序表时必须明确规定容量。若线性表长度变化较大，则容量难于预先确定，估计过大将造成空间浪费，估计太小又将使空间溢出机会增多。

顺序栈：

```
}                                                    //返回 tmp
```

(7) 取栈顶元素。

如果顺序栈不为空，则返回栈顶元素的值；否则，返回特殊值，表示栈为空。

取栈顶元素操作的算法实现如下。

```
public T Getop(){
   T tmp ;
   if (IsEmpty()){                                   //T 类型 tmp
      throw new RuntimeException("栈为空，无法删除" );
   }
   tmp = data[top];                                  //栈非空，tmp 等于栈顶元素
   //没有栈顶指示器减 1
   return tmp;                                       //返回 tmp
}
```

3.2.4 Java 基础类库中的顺序栈

Java 语言的推出者设计开发了顺序栈类 java. util. Stack，供应用程序员使用。在开发软件时，可以利用继承和覆盖技术在 java. util. Stack 的基础上开发新的顺序栈类，以便切合软件的实际功能需求。java. util. Stack 中的部分构造和普通方法如表 3-1 和表 3-2 所示，读者可以结合 java. util. Stack 源代码与本书中的 SeqStack 栈比较。

表 3-1 构造方法及功能说明

构造方法	功能说明
Stack()	创建一个空堆栈

表 3-2 普通方法及功能说明

返回值类型	普通方法	功能说明
boolean	empty()	测试堆栈是否为空
T	peek()	查看堆栈顶部的对象，但不从堆栈中移除它
T	pop()	移除堆栈顶部的对象，并作为此函数的值返回该对象
T	push(*T* item)	将数据元素压入堆栈顶部
int	search(Object *o*)	返回对象在堆栈中的位置，以 1 为基数

3.3 栈的链式存储

3.3.1 栈的链式存储定义

栈的另一种存储方式是链式存储，即将栈中的数据元素存放在一组任意的存储单元中，简称为链栈(linked stack)。链栈通常用单链表来表示，它的实现是单链表的简化。因此，链栈结点的结构与单链表结点的结构相同，如图 3-8 所示。

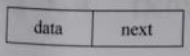


图 3-8 链栈结点

由于链栈的操作只是在一端进行，因此为了操作方便，将栈顶设在链表的头部，并且不需要头结点。栈(a_1,a_2,a_3,a_4,a_5,a_6)的链式存储结构如图 3-9 所示。

队列：

```
public T GetFront() {
    if (IsEmpty()) {
        throw new RuntimeException("队列为空");
    }
    return data[front];
}
```

3.8　Java 基础类库中的队列

Java 语言的推出者设计开发了 java.util.Queue，供应用程序员使用。在 Java 基础类库中，实现该接口的类有 AbstractQueue、ArrayBlockingQueue、ArrayDeque、ConcurrentLinkedQueue、DelayQueue、LinkedBlockingDeque、LinkedBlockingQueue、LinkedList、PriorityBlockingQueue、PriorityQueue、SynchronousQueue。在开发软件时，也可以开发新类实现该接口，以便切合软件的实际功能需求。java.util.Queue 中的部分接口方法如表 3-5 所示。

表 3-5　接口方法及功能说明

返回值类型	接 口 方 法	功 能 说 明
boolean	add(*T e*)	将指定的元素插入此队列(如果立即可行且不会违反容量限制)。如果成功，则返回 true；如果当前没有可用的空间，则抛出 IllegalStateException
T	element()	获取但不移除此队列的头
boolean	offer(*T e*)	将指定的元素插入此队列(如果立即可行且不会违反容量限制)。当使用有容量限制的队列时，此方法通常要优于 add(*T*)，后者可能无法插入元素而只是抛出一个异常
T	peek()	获取但不移除此队列的头。如果此队列为空，则返回 null
T	poll()	获取并移除此队列的头。如果此队列为空，则返回 null
T	remove()	获取并移除此队列的头


视频讲解

3.9　队列的应用举例

队列的典型应用是通过编程判断一个字符串是否为回文。回文是指一个字符序列以中间字符为基准，其两边字符完全相同，如字符序列“ACBDEDBCA”是回文。

算法思想：判断一个字符序列是否为回文，即将第一个字符与最后一个字符相比较，第二个字符与倒数第二个字符比较，以此类推，直至第 i 个字符与第 $n-i$ 个字符比较。如果每次比较都相等，则该序列为回文；如果某次比较不相等，则不是回文。因此，可以将字符序列分别入队列和栈，然后逐个出队列和出栈，并比较出队列的字符和出栈的字符是否相等。如果字符比较全部相等，则该字符序列是回文；否则，该序列不是回文。

算法中的队列和栈可以采用任意存储结构，本例采用循环顺序队列和顺序栈来实现，其他的情况读者可作为习题。算法中假设输入的都是英文字符而没有其他字符，对于输入其他字符情况的处理，读者可以自行实践。使用循环顺序队列和顺序栈来判断一个字符串是