1.3算法和算法分析

DS/SS

算法

- 内涵：

  是对特定问题求解步骤的一种描述，是指令的有限序列，其中每一条指令表示一个或多个操作。

- 特性：

  *有穷形：有穷步+有穷时间/每一步

  *确定性：指令的语义无二义性

  *可行性：算法能用基本操作完成

  *输入：零个或多个输入

  *输出：一个或多个输出

1.3算法和算法分析（续）

算法的描述方式

- 自然语言
- 流程图
- 类高级语言（类C语言、类Pascal语言等）
- 程序设计语言

1.3算法和算法分析（续）

算法设计的要求

- 正确性（Correctness）
- 可读性（Readablity）
- 健壮性（Robustness）
- 高时间效率与低存储量需求

1.3算法和算法分析（续）

DS/SS

算法选择时效率的考虑

虽然我们希望所选的算法占用额外空间小，运行时间短，其他性能也好，但计算机时间和空间这两大资源往往相互抵触。所以，一般算法选择的原则是：

对于反复使用的算法应选择运行时间短的算法；而使用次数少的算法可力求简明、易于编写和调试；对于数据量较大的算法可从如何节省空间的角度考虑。

1.3算法和算法分析（续）

算法时间效率的度量（一）

- 程序运行消耗时间取决于下列因素

  *算法策略

  *问题规模

  *语言层次

  *编译程序所产生的机器代码的质量

  *机器执行指令的速度

- 算法时间效率度量

  算法时间效率在软硬件环境相同的情况下取决于问题的规模，即$T(n)=f(n)$

1.3算法和算法分析（续）

算法时间效率的度量（二）

- 算法时间效率度量的基本做法

在算法中选取一种对于所研究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作为算法的时间度量。

一般而言，这个基本操作是最深层循环内的语句中的原操作。

- 算法时间复杂度

T(n)=O (f(n)) 称为算法的渐近时间复杂度，简称时间复杂度。

1.3算法和算法分析（续）

算法时间效率的度量（三）

- **算法语句频度与时间复杂度的关系**

一般算法消耗的实际时间为算法中每条语句频度之和，是n的函数T(n)。当n趋于无穷大时，T(n)的同阶无穷大即是算法时间复杂度。

```
for (i=1; i<=n; ++i)                          ---------- n+1
    for (j=1;  j<=n; ++j){                    ---------- n(n+1)
        c[i][j] = 0;                          ---------- n^2
        for (k=1; k<=n; ++k)                  ---------- n^2 (n+1)
            c[i][j] += a[i][k]*b[k][j];       ---------- n^3
    }
```

总时间消耗：$T(n)=2n^3+3n^2+2n+1$

时间复杂度：$T(n)=O(n^3)$

1.3算法和算法分析（续）

算法时间效率的度量（四）

- 大O的运算规则

*加法准则

a）前提：$T_1(m)=O(f(m));T_2(n)=O(g(n))$

结论：$T(n)=T_1+T_2=O(max(f(m),g(n)))$

b）前提：$T_1(n)=O(f(n));T_2(n)=O(g(n))$

结论：$T(n)=T_1+T_2=O(f(n)+g(n))$

*乘法准则

前提：$T_1(n)=O(f(n));T_2(n)=O(g(n))$

结论：$T(n)=T_1*T_2=O(f(n)*g(n))$

1.3算法和算法分析（续）

算法存储空间的度量

- **算法存储空间度量的基本做法**

  用程序执行中需要的辅助空间的消耗作为存储空间度量的依据，是问题规模n的函数。而程序执行中本身需要的工作单元不能算。

- **算法空间复杂度**

  S(n)=O (f(n)) 称为算法的空间复杂度。

教学内容---第二章

DS/SS

1. 绪论
2. 线性表
3. 栈、队列和串
4. 数组
5. 广义表
6. 树和二叉树
7. 图
8. 动态存储管理
9. 查找
10. 内部排序
11. 外部排序
12. 文件

2.1线性表的逻辑结构

DS/SS

线性数据结构的特点

在数据元素的非空有限集中

- 存在唯一的一个被称作“第一个”的数据元素；
- 存在唯一的一个被称作“最后一个”的数据元素；
- 除第一个之外，集合中的每一个数据元素均只有一个前驱；
- 除最后一个之外，集合中每一个数据元素均只有一个后继。

2.1线性表的逻辑结构（续）

DS/SS

基本概念和术语

- **线性表**：n个数据元素的有限序列（线性表中的数据元素在不同环境下具体含义可以不同，但在同一线性表中的元素性质必须相同）。
- **表长**：线性表中元素的个数n(n>=0)。
- **空表**：n=0时的线性表称为空表。
- **位序**：非空表中数据元素 a_i 是此表的第 i 个元素，则称 i 为 a_i 在线性表中的位序。

2.1线性表的逻辑结构（续）

线性表的抽象数据类型定义

ADT List {

数据对象：D={a_i | a_i属于ElemSet, i = 1,2, … ,n, n>=0}

数据关系：R1={< a_{i-1}, a_i >| a_{i-1}, a_i属于D, i =2,3, …, n}

基本操作：

InitList(&L)

DestroyList(&L)

ClearList(&L)

ListLength(L)

GetElem(L, i ,&e)

初始条件：L存在；
1<=i<=ListLength(L)

操作结果：用e返回L中第i个数据元素的值

} ADT List

LocateElem(L,e,compare()) ---------- 查找

初始条件：L存在； compare()是判定条件

操作结果：返回第1个与e满足关系compare()的数据元素位序，若不存在，则返回0

ListInsert(&L, i ,e) ---------- 插入

初始条件：L存在；1<=i<=ListLength(L)+1

操作结果：第i个位置之前插入元素e，长度加1

ListDelete(&L, i ,&e) ---------- 删除

初始条件：L存在；非空；1<=i<=ListLength(L)

操作结果：删除第i个元素，e返回值，长度减1

2.1线性表的逻辑结构（续）

DS/SS

示例

问题：两个集合A,B,求A = A ∪ B

算法求解：

```
Void union(List &La, List Lb) {
   //将所有在线性表Lb中但不在La中的数据元素插入到La中
   La_len = ListLength(La); Lb_len = ListLength(Lb);
   for (i = 1; i <= Lb_len; i++) {
        GetElem(Lb, i, e);
        if (!LocateElem(La, e, equal))
              ListInsert(La, ++La_len, e);
   }
}
```

算法实现的复杂度取决于基本操作的复杂度以及算法的控制流程

2.2线性表的顺序存储结构

DS/SS

顺序表---线性表的顺序存储

内涵：

线性表的顺序存储指用一组地址连续的存储单元依次存储线性表的数据元素。这称为顺序表。

特点：

* 存储单元地址连续（需要一段连续空间）
* 逻辑上相邻的数据元素其物理位置也相邻
* 随机存取
* 存储密度最大（100%）

2.2线性表的顺序存储结构（续）

DS/SS

顺序表的随机存取

对线性表L,设每个元素占k个存储单元，则有：

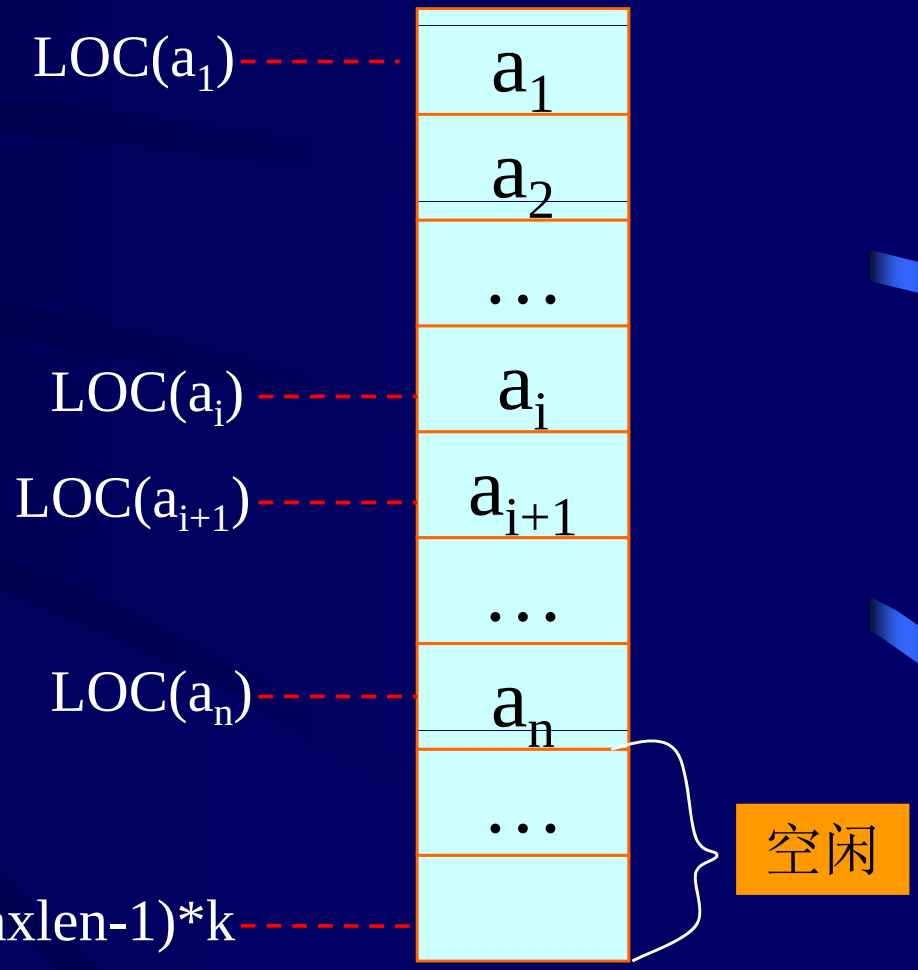
递推关系：

$LOC(a_{i+1})=LOC(a_i)+k$

任一元素a_{i+1}的存储位置：

$LOC(a_i)=LOC(a_1)+(i-1)*k$

(其中1<= i <=ListLength(L))



2.2线性表的顺序存储结构（续）

DS/SS

顺序表的数据类型描述

用高级语言中的数组类型描述线性表的顺序存储

```
// 动态分配
Status InitList_Sq(SqList &L)
    L.elem=(ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (! L.elem) exit(overflow);
    L.length=0;
    L.listsize = LIST_INIT_SIZE ;
    return  OK;
} // InitList_Sq
} SqList
```

2.2线性表的顺序存储结构（续）

DS/SS

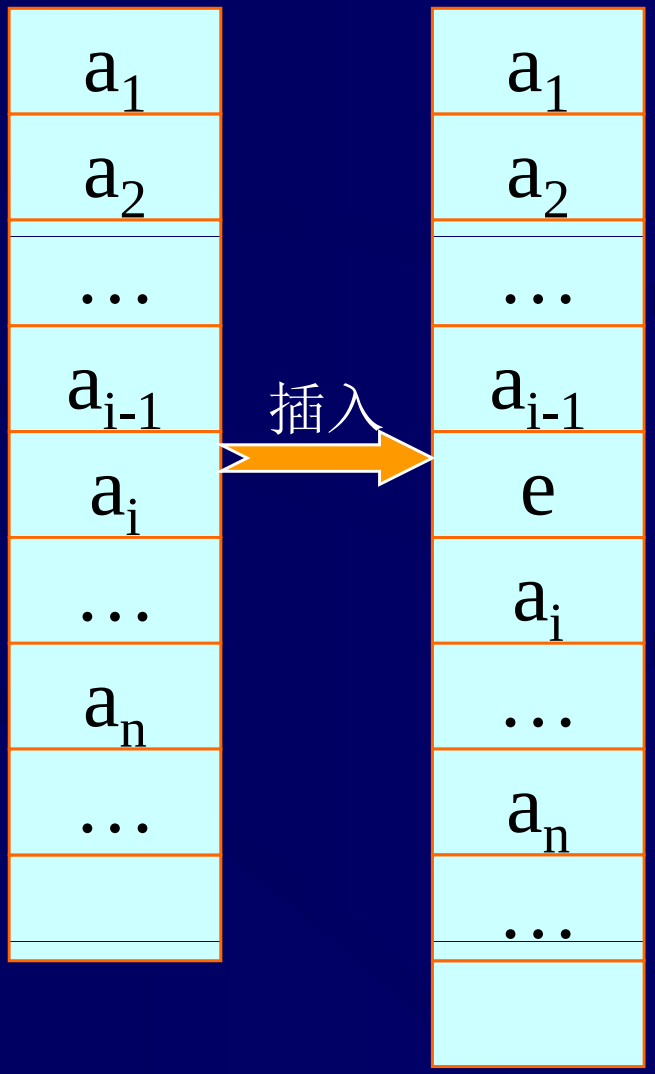
顺序表上插入运算的实现

$(a_1, \ldots, a_i, a_{i+1}, \ldots, a_n)$ 表长为n

ListInsert(&L, i ,e)

$(a_1, \ldots, a_{i-1}, e, a_i, \ldots, a_n)$ 表长为n+1

```
Status ListInsert_Sq(SqList &L, int i, ElemType e) {
    第一步：判断参数是否合法合理，否则出错；
    第二步：在物理空间中找到插入位置；
    第三步：插入前的准备工作;
    第四步：插入；
} //ListInsert_Sq
```



2.2线性表的顺序存储结构（续）

顺序表上插入运算效率分析

分析：从算法流程上看，查找插入位置、插入元素的时间花费是常数级，而算法执行时间主要花在插入前元素的移动上，而移动元素的个数与插入位置i有关。

设 p_i 为在第 i 个元素之前插入一个元素的概率，且 $p_1=p_2=\ldots=p_i=\ldots=p_n=1/(n+1)$，则平均移动次数为：

$$E_{is}=\sum_{i=1}^{n+1} p_i\,(n-i+1)=1/(n+1)\sum_{i=1}^{n+1}(n-i+1)=n/2$$

则 $T(n)=O(n)$

2.2线性表的顺序存储结构（续）

DS/SS

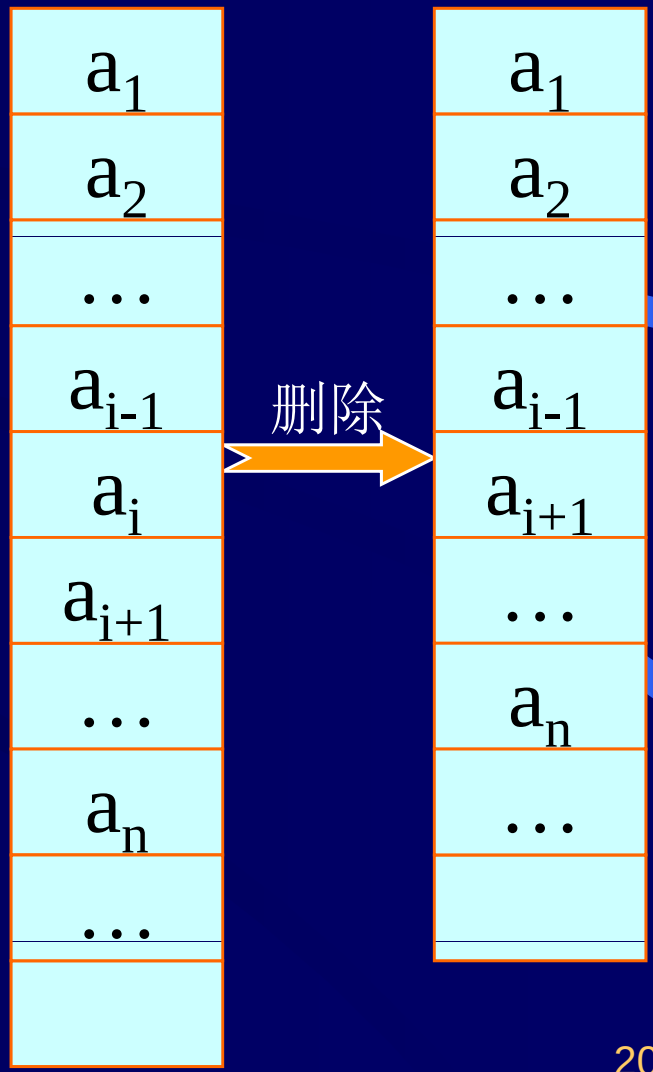
顺序表上删除运算的实现

$(a_1, \ldots, a_i, a_{i+1}, \ldots, a_n)$ 表长为n

ListDelete(&L, i ,&e)

$(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$ 表长为n-1

```
Status ListDelete_Sq(SqList &L, int i, ElemType & e) {
    第一步：判断参数是否合法合理，否则出错；
    第二步：在物理空间中找到删除位置；
    第三步：删除；
    第四步：删除后的善后工作
} // ListDelete_Sq
```



2.2线性表的顺序存储结构（续）

DS/SS

顺序表上删除运算效率分析

分析：从算法流程上看，查找删除位置、删除元素的时间花费是常数级，而算法执行时间主要花在删除后元素的移动上，而移动元素的个数与删除位置i有关。

设 q_i 为删除第 i 个元素的概率，
且 $q_1=q_2=\ldots=q_i=\ldots=q_n=1/n$，则平均移动次数为：

$$E_{de}=\sum_{i=1}^{n} q_i(n-i)=(1/n)\sum_{i=1}^{n}(n-i)=(n-1)/2$$

则 $T(n)=O(n)$

2.3线性表的链式存储结构

DS/SS

顺序表优缺点分析

优点：

* 不需要额外空间来存储元素之间的关系
* 可以随机存取任一元素

缺点：

* 插入和删除运算需要移动大量元素
* 需要一段连续空间
* 预先分配足够大的空间
* 表的容量难以扩充

可以通过使用动态数组数据类型来描述顺序表而改进这两个缺点