数据结构
Data Structure

2017年秋季学期

刘鹏远

(线性)表之顺序表

- 概念
- (逻辑结构)ADT定义与使用
- 存储结构及操作实现
- 应用(链表实现后再讲)

概念

- 线性表是线性结构的原型结构，也最基础，基本特点是：
  - 结构含有限个数据元素
  - 数据元素之间为一一相邻关系

可具体描述为，在有限数据集内：

① 存在一个唯一的被称为“第一个”的数据元素；

② 存在一个唯一的被称为“最后一个”的数据元素；

③ 除第一元素外，每元素均有唯一一个直接前驱；

④ 除最后一元素外，每元素均有唯一一个直接后继。

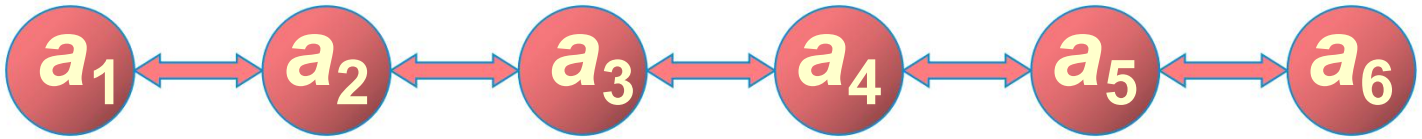
线性表是数据结构的基础，必须掌握实现基本结构

线性表(Linear List)：是由n(n≥0)个数据元素(结点)a1, a2, … an组成的有限序列。该序列中的所有结点具有相同的数据类型。其中数据元素的个数n称为线性表的长度。

第一个(首)结点，后一个(尾)结点

a1, a2,…ai-1都是ai(2≤i≤n)的前驱，其中ai-1是ai的直接前驱;
ai+1, ai+2, …an都是ai(1≤i ≤n-1)的后继，其中ai+1是ai的直接后继。

当n=0时，称为空表。（重点：从编号一始，不太科学）



节点(数据元素)可以是什么？

- 单数据项的数据元素:

  26个英文字母组成的字母表： {A，B，C、…、Z}

  1个骰子各面的点数{1,2,3,4,5,6}

- 多数据项的数据元素：

  学生基本情况：{(‘201414101’，‘张XX’，‘男’，06/24/1993)，(‘201414102’，‘刘XX’，‘男’，08/12/1994) …， (‘201414102’，‘李XX’，‘女’，08/12/1994) }

- 其他数据的抽象：

  带结构的数据元素、图片，视频，音频等

如何使线性表更通用(不受节点类型限制)?

1、先写好以Elem_type作为节点类型的各类实现。
函数原型：int do_sth(Elem_type...); 及相应函数实现。
2、typedef 例如：
- **节点是int的**
在自定义头文件中：typedef int Elem_type;
- **节点是任意类型x_type**
在自定义头文件中：
  - 定义一个x_type类型(struct student{int age;char sex;...});
  - typedef x_type Elem_type;

(逻辑结构)
ADT定义与使用

线性表的ADT（与具体存储结构无关）

ADT List

{

数据对象：

D={ai | ai∈ElemSet, i=1,2,…,n, n≥0}

数据关系：

R={<ai-1,ai>|ai-1,ai∈D, i=2,3,…,n }

基本操作：

init(*L)

操作结果：构造一个空的线性表L

destroy(*L)

初始条件：线性表L已存在

操作结果：销毁线性表L

clear(*L)

初始条件：线性表L已存在

操作结果：将线性表L重置为空表

is_empty(L)
初始条件：线性表L已存在
操作结果：若L为空表，返TRUE，否则FALSE
length(L)
初始条件：线性表L已存在
操作结果：返回线性表L中数据元素的个数
get_elem(L, i ,*e)
初始条件：线性表L已存在，1≤i≤length(L)
操作结果：用e返回L中第i个数据元素的值

locate_elem(L, e, compare( ))

初始条件：

线性表L已存在，compare( )是数据元素的判定函数

操作结果：

返回L中第1个与e满足关系compare( )的数据元素的位序。若不存在，则返回值为0

prior_elem(L, cur_e ,*pre_e)

初始条件：线性表L已存在

操作结果：若cur_e是L的数据元素，且不是第一个，则用pre_e返回它的前驱，否则操作失败

next_elem(L, cur_e ,*next_e)

初始条件：线性表L已存在

操作结果：若cur_e是L的数据元素，且不是最后一个，则用next_e返回它的后继，否则操作失败

insert(*L, i, e)

初始条件：线性表L已存在，1≤i≤length(L)+1

操作结果：在L中第i个位置之前插入新的数据元素e，L的长度加1

delete(*L, i, &e)

初始条件：线性表L已存在，1≤i≤length(L)

操作结果：删除L的第i个数据元素，并用e返回其值，L的长度减1

traverse(L, visit( ))

初始条件：线性表L已存在

操作结果：依次对L的每个数据元素调用函数visit( )。一旦visit( )失败，则操作失败

}//ADT定义由类C语言实现

假设上述ADT定义及基本操作均已经实现。

任务1：集合合并

利用两个线性表La和Lb，分别表示两个集合A和B，现要求一个新的集合A＝A∪B（线性表中的数据元素即为集合中的成员）

int union(*La, Lb)

输入：线性表La、线性表Lb

输出：更新后的La

处理方法：

扩大线性表La，将存在于线性表Lb中而不存在于线性表La中的数据元素插入到线性表La中去。

步骤：

1. 从线性表LB中依次取得每个数据元素;
2. 依值在线性表LA中进行查找;
3. 若不存在，则插入LA。

```
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尾部
    La_len = length(La);
    Lb_len = length(Lb);
    int i;
    for (i = 1;  i <= Lb_len;  i++)
    {
        get_elem(Lb, i, *e);
        if(!locate_elem(La, e, equal))
            insert(La, ++La_len, e);
    }
}
```

该算法的大O=？

使用2：合并有序（假设正序）表LA,LB，生成LC保持有序。如何做？

归并法：

```
void merge_list(List La, List Lb, List *Lc)
{
        Init(Lc);
        int i=j=k=1;
        La_len = La.length;
        Lb_len = Lb.length;
```

```
While ((i<=La_len)&&(j<=Lb_len))
{
    get_elem(La,i,a);      get_elem(Lb,j,b);
    if (a<=b)   {
            insert(Lc, k++, a);
            i++;
    }
    else {
            insert(Lc, k++, b);
            j++;
    }
}
```

```
    while (i<=La_len) {
        get_elem(La,i++,a); insert(Lc, k++, a);
    }
    while (j<=Lb_len) {
        get_elem(Lb,j++,b); insert(Lc,k++, b);
    }
}
```

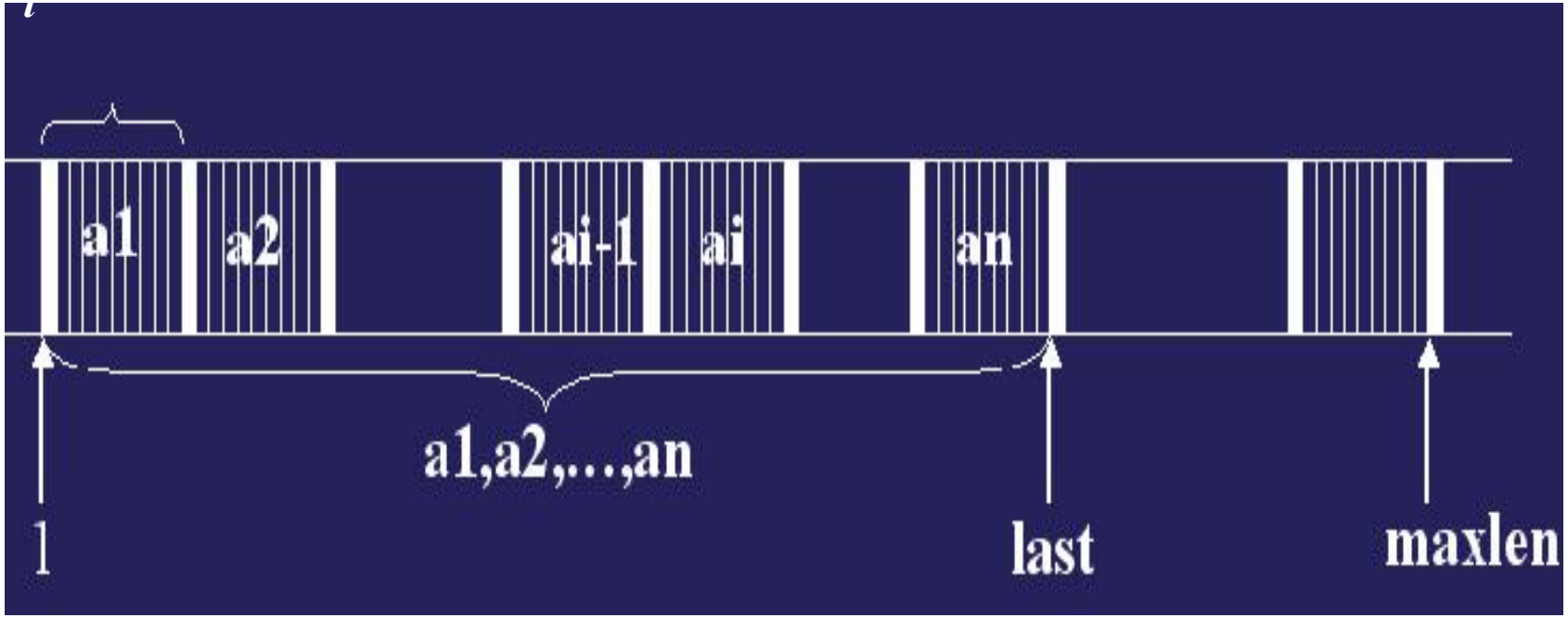
//if [get] and [insert] is 1 step time，大O=？？

是否还有其他方法来完成这个任务？

存储结构之
顺序存储结构

顺序存储/表示

- 定义：用一组地址连续的存储单元依次存储线性表中的数据元素
  --顺序表（顺序表是线性表的一种）
  - 逻辑相邻关系由物理相邻表示
  - 相邻存储单元之间没有空间
  - 随机存取（知起始位置，则任意元素均可算出地址，直接存取）
    （为什么知道地址就可以直接存取？）

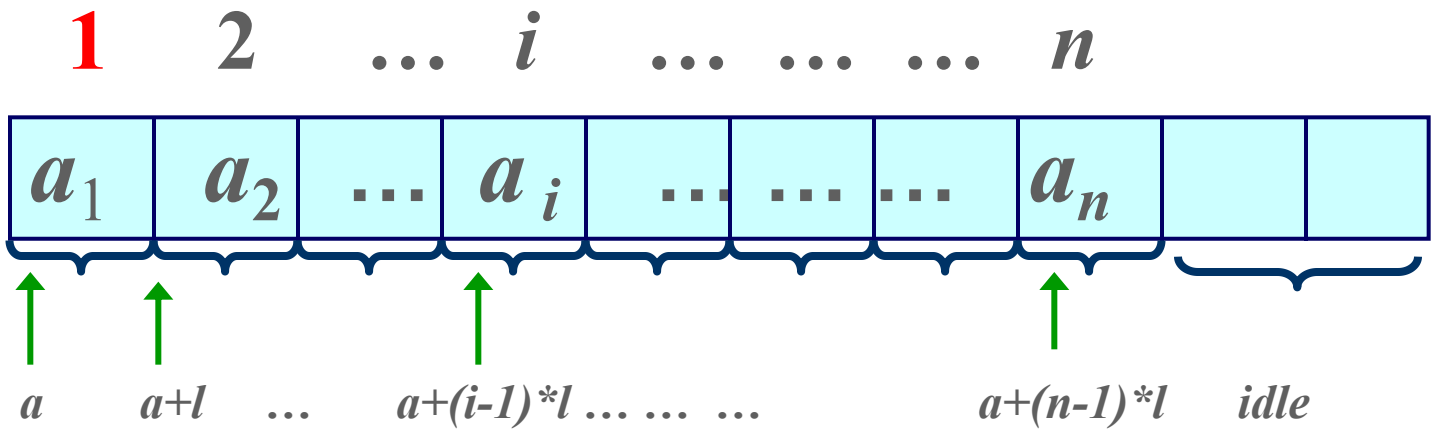


顺序存储/表示

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

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

基地址base



- 可直接利用高级语言中已经实现的数组
- 实现方案一(静态)：

```
#define  LIST_SIZE  100
   typedef  struct{
      Elem_type elem[LIST_SIZE];   /* 存储空间*/
      int      length;                    /* 当前长度*/
   }seq_list_static;
```

评价：

1) LIST_SIZE过小，则会导致顺序表上溢；
2) LIST_SIZE过大，则会导致空间的利用率不高

- 实现方案二（动态，P22）：

```
#define  INIT_SIZE  50    /* 存储空间的初始分配量*/
#define  INCREMENT  10       /* 存储空间的分配增量     */
  typedef  struct{
     Elem_type *elem;              /* 存储空间的基址base    */
     int              length;     /* 当前长度               */
     int              size; /* 当前分配的存储容量     */
  }seq_list;
```

该方案解决“上溢”问题和“空间利用率不高”问题，但需要：

1) 记载当前线性表的实际分配的空间大小；
2) 实现语言能提供空间的动态分配与释放管理。

基本操作列表：

初始化，清空，求长度（元素个数），销毁，判断空表，取表中第i个元素，遍历，查找某元素位置，插入元素，删除元素。可根据基本操作，形成更多操作

尽量避免直接修改属性而提供访问函数，实现良好封装

程序头可加如下定义代码

```
#define  OK   1
#define  ERROR   -1
#define  OVERFLOW   -2
#define  MAX_SIZE  100
typedef  int  Status ;.....................
```

```
Status init(seq_list *L)
{//初始化一个空的顺序表，大小为INIT_SIZE
        L->elem =
(Elem_type*)malloc(INIT_SIZE*sizeof(Elem_type));
        if(L->elem == NULL) exit(OVERFLOW);
        L->length = 0;
        L->size = INIT_SIZE;
        return OK;
} //时间复杂度O（1）
```

```
Status  clear( seq_list *L)
{
      L->length = 0;
      return OK;
  } //时间复杂度O（1）
```

- 销毁　　利用free　自行实现
- 求表长　利用L.length　　自行实现
- 判断空表　　利用L.length　　自行实现

- 取表中第i个元素(随机存取)

```
Status get_elem(seq_list L, int i ,Elem_type* e)
{
    if(i<1||i>L.length ) return ERROR;        //边界条件
    *e = *(L.elem + i-1);
    //*e = L.elem[i-1]
    return OK;
}//O(1)
```

- 按值查找1：在顺序表中从头查找结点值等于给定值 x 的结点位置

```
int find_elem(seq_list L ,Elem_type e) {
    int i;
    for(i=0;i<L.length;i++)  {
        if(e==L.elem[i])
            return i+1;
    }
    return ERROR;
}//O(？）比较次数 最好，最坏，平均
```

- 按值查找2：（以指针查找）

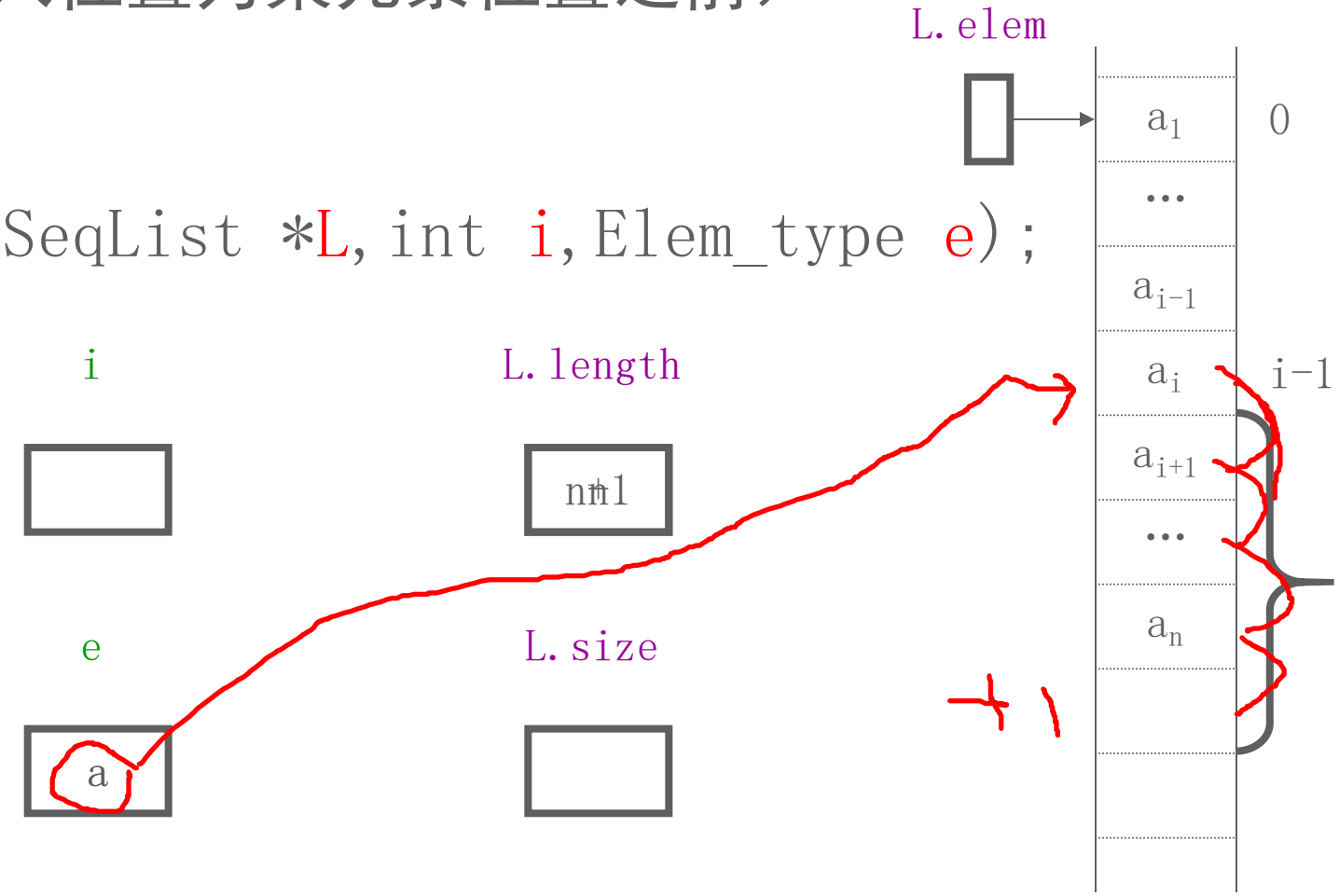
```
int find_elem(seq_list L ,Elem_type e) {
    int i=1;
    Elem_type* p = L.elem;
    while(i<=L.length)        {
        if(e==*p) return i;
        i++;  p++;
    }
    return ERROR;
}O（n）
```

- 遍历

```
Status traverse(seq_list L, Status (*visit)( Elem_type e)) {
    int i;
    for(i=1; i<=L.length; i++)  {
        visit(L.elem[i-1]);
     }
    return OK;
}//O(length)  如不考虑以函数为参数，则直接printf也可
Status Print(Elem_type e){printf("%d\n",e);return OK;}...
traverse(L, &Print(e));
```

插入元素（默认插入位置为某元素位置之前）

```
status ListInsert(SeqList *L,int i,Elem_type e);
```



分三步：

(1) 将线性表L中的第i个至第n个结点后移一个位置。
(2) 将结点e插入到结点ai-1之后（即ai位置）。
(3) 线性表长度加1

```
Status insert(seq_list *L,int i, Elem_type e)
{
        //此处需要判断插入是否能够进行
        int j;
        for(j=L->length-1; j>=i-1; j--)
                L->elem[j+1] = L->elem[j] ;//elem数组起始为0
        L->elem[i-1]=e;
        L->length++;
        return OK;
}
```

- 顺序表插入元素的时间复杂度分析：

  算法的基本操作是移动元素

- 最好
  - O（1）
- 最坏
  - O（n）
- 平均
  - O（n）----参考教材P25
  - 平均移动次数为n/2

判断插入是否能够进行：

- ✡ 插入位置非法

  条件：　　i<1||i>L.length+1

  处理：　　返回ERROR

- ✡ 存储空间不足

  条件：　　L.length>=L.size

  处理：　　增加分配

<---realloc(L.elem,(L.size+INCREMENT)*sizeof(ElemType))

insert实现方法2，见教材P24页

利用指针实现，功能相同

相对复杂但是对理解指针操作比较好

因此请同学们照着自行上机实现。

删除（完整说明/注释示例）

函数头：Status delete(seq_list *L,int i,Elem_type *e)

操作结果：

删除线性表L中的第i个数据元素，并用e返回其值，删除完成后，线性表L的长度减1，其中i的取值范围为：1≤i≤表长，表长>0

头文件中函数原型：

```
Status delete(seqList *,int,ElemType *);
```

思路：

(1)判定删除能否进行：

- 删除位置非法

  条件：　　i<1||i>L.length

  处理：　　返回ERROR

- 线性表为空表

  条件：　　L.length==0

  处理：　　此时对任意i，均有删除位置非法，但实际实现中，可不做这步处理，why?

(2)将被删元素的值赋给e；

(3)移动n-i个元素以实现删除；

(4)表长减1。

- 根据前面思路，自行实现删除指定位置元素的操作：

1、类似前面插入元素的实现

2、参考P24删除算法的实现

思考：插入、删除还有其他方法吗？为什么要这么做？

- 顺序表删除元素的时间复杂度分析：
- 最好
  - O（1）
- 最坏
  - O（n）
- 平均
  - O（n）----参考教材P25
  - 平均移动次数为n-1/2

顺序表小结

逻辑上是线性结构，实现上是顺序存储。

存储结构是采用一片物理地址连续的空间（一般利用数组）

用物理地址的（线性）相邻关系来表示逻辑上的（线性）相邻关系。

优点：

- 由于逻辑顺序与物理顺序保持一致，故而可以随机存取元素，时间复杂度O（1）
- 操作较为简单

- 缺点：
  - 插入、删除操作要移动大量元素；O（n）
  - 存储空间是预分配的、不太灵活、空间浪费；
  - 表的存储空间扩充耗时(有时)；

上机及作业：

顺序表实现　要求：

1、实现顺序表的动态定义与全部操作并进行验证

2、主函数建立一个顺序表

3、插入10个整数，删除1,3,5位置的整数并输出

4、求此时表长

5、遍历此时的顺序表

上机

下午上机实现init_sq, clear_sq, destory_sq,insert_sq, delete_sq, traverse_sq，其余的操作与验证课后实现。

上机及作业：

顺序表实现，要求：

1、实现顺序表的动态定义与操作

2、主函数建立一个顺序表

3、插入10个整数，删除1,3,5位置的整数

4、求此时表长

5、遍历此时的顺序表

下周一凌晨4点模块关闭(你们见过凌晨4点的北京嘛？)