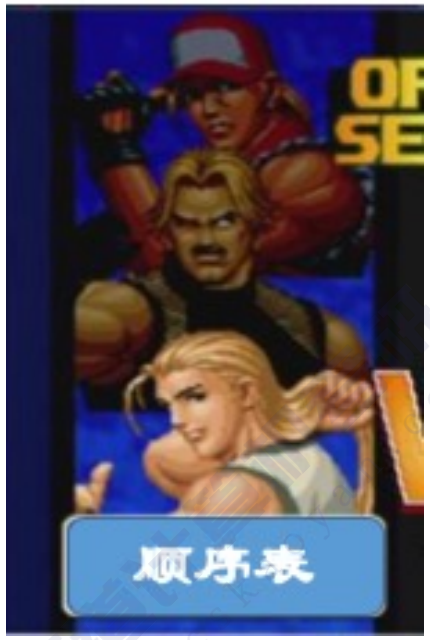
本节内容

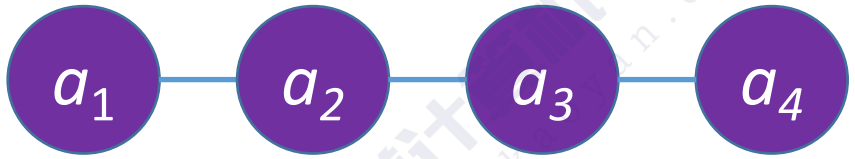
顺序表
V.S.
链表

知识总览

- 顺序表 V.S. 链表
  - Round 1 — 逻辑结构
  - Round 2 — 物理结构/存储结构
  - Round 3 — 数据的运算/基本操作
  - 如何抉择？❤️到底爱谁？💖

Round 1：逻辑结构

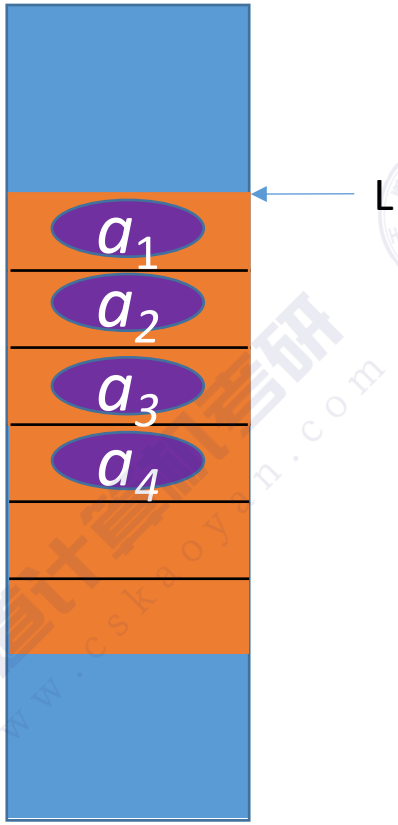


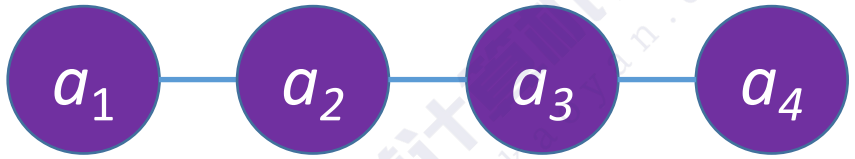


都属于线性表，都是线性结构

Round 2：存储结构

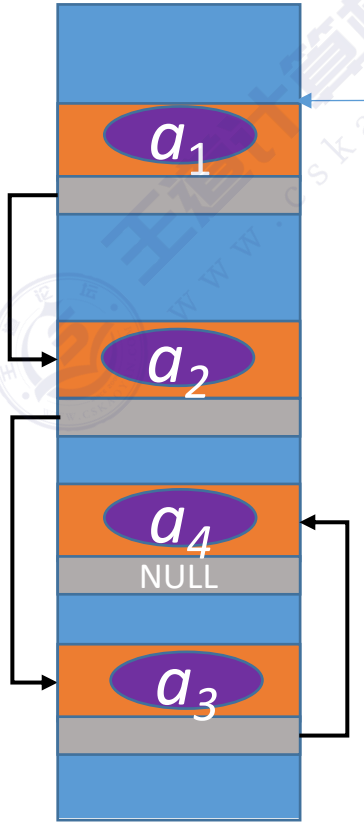
顺序表
（顺序存储）





都属于线性表，都是线性结构

链表
（链式存储）



优点：支持随机存取、存储密度高
缺点：大片连续空间分配不方便，改变容量不方便

优点：离散的小空间分配方便，改变容量方便
缺点：不可随机存取，存储密度低

Round 3：基本操作

复习回忆思路：

创销、增删改查

Round 3：基本操作

创

顺序表
（顺序存储）

L

需要预分配大片连续空间。若分配空间过小，则之后不方便拓展容量；若分配空间过大，则浪费内存资源

链表
（链式存储）

L

只需分配一个头结点（也可以不要头结点，只声明一个头指针），之后方便拓展

静态分配：静态数组

容量不可改变

动态分配：动态数组（malloc、free）

容量可改变，但需要移动大量元素，时间代价高

Round 3：基本操作

销

顺序表
（顺序存储）

修改 Length = 0

L

静态分配：静态数组

系统自动回收空间

动态分配：动态数组（malloc、free）

需要手动 free

```
typedef struct{
    ElemType *data;
    int MaxSize;
    int length;
} SeqList;
```

malloc 和 free
必须成对出现

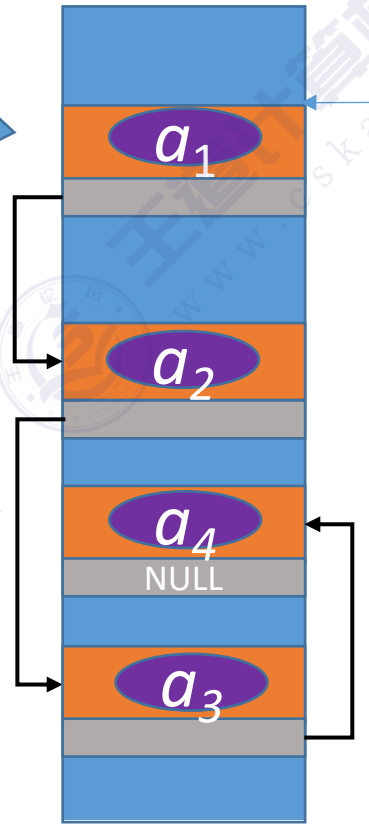
```
L.data = (ElemType *) malloc (sizeof(ElemType) * InitSize);
```

创

```
free(L.data);
```

销

链表
（链式存储）



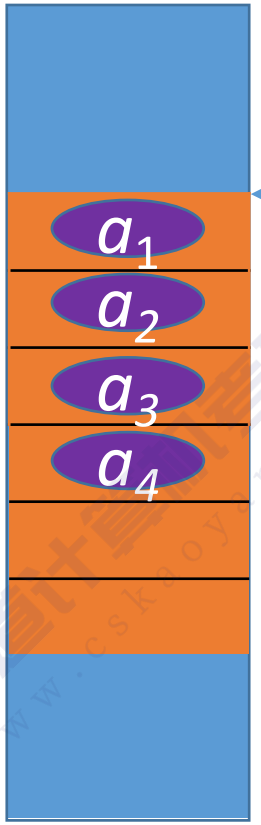
L

依次删除各个
结点（free）

Round 3：基本操作

增 删

顺序表
（顺序存储）



插入/删除元素要将后续元素都后移/前移

时间复杂度 O(n)，时间开销主要来自移动元素

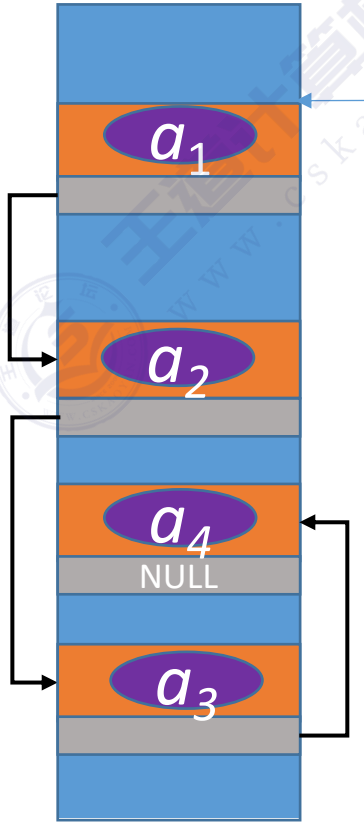
若数据元素很大，则移动的时间代价很高

插入/删除元素只需修改指针即可

时间复杂度 O(n)，时间开销主要来自查找目标元素

查找元素的时间代价更低

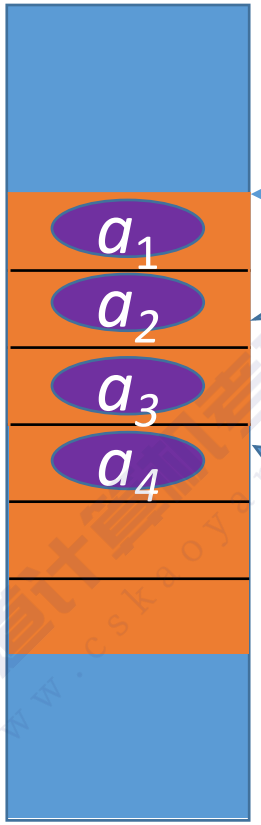
链表
（链式存储）



Round 3：基本操作

查

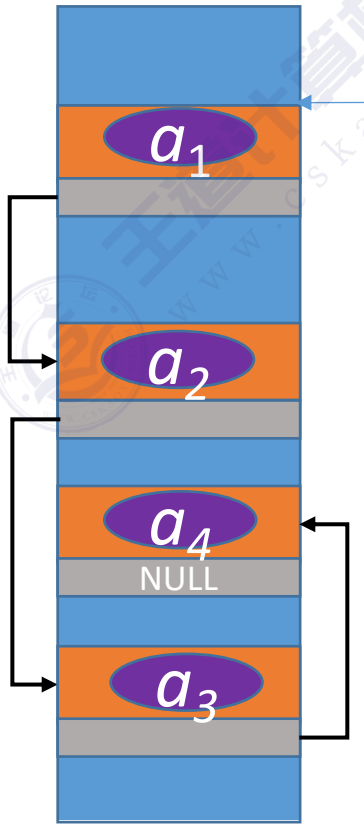
顺序表
（顺序存储）



按位查找：O(1)

按值查找：O(n)
若表内元素有序，可在
$O(\log_2 n)$ 时间内找到

链表
（链式存储）



按位查找：O(n)

按值查找：O(n)

用顺序表 or 链表？

	顺序表	链表
弹性（可扩容）	😭	😄
增、删	😭	😄
查	😄	😭

表长难以预估、经常要增加/删除元素 ——链表

表长可预估、查询（搜索）操作较多 ——顺序表

知识回顾与重要考点

开放式问题的答题思路：

问题：　请描述顺序表和链表的 bla bla bla...
　　实现线性表时，用顺序表还是链表好？

顺序表和链表的逻辑结构都是线性结构，都属于线性表。
但是二者的存储结构不同，顺序表采用顺序存储...(特点，带来的优点缺点)；链表采用链式存储...（特点、导致的优缺点）。
由于采用不同的存储方式实现，因此基本操作的实现效率也不同。当初始化时...；当插入一个数据元素时...；当删除一个数据元素时...；当查找一个数据元素时...