第二讲 线性表

郑渤龙

线性表

- 线性表是零个或多个元素的有穷序列，通常可以表示为$e_0, e_1, e_2, \ldots, e_{n-1}\ (n \geq 1)$
  - 节点（表目）：线性表中的元素
  - 索引（下标）：i称为节点e_i的索引
  - 表的长度：线性表中所含元素的个数n
  - 空表：长度为0的线性表$(n = 0)$

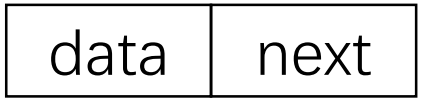
基础线性表

■ 数组（Array）

- 按索引值从小到大存放在一片连续的内存区域
- 紧凑结构，存储密度为1

■ 链表（Linked List）

- 通过指针把一串存储结点（通常在内存中不连续）链接成一个链
- 存储结点由两部分组成：数据域+指针域（后继节点）



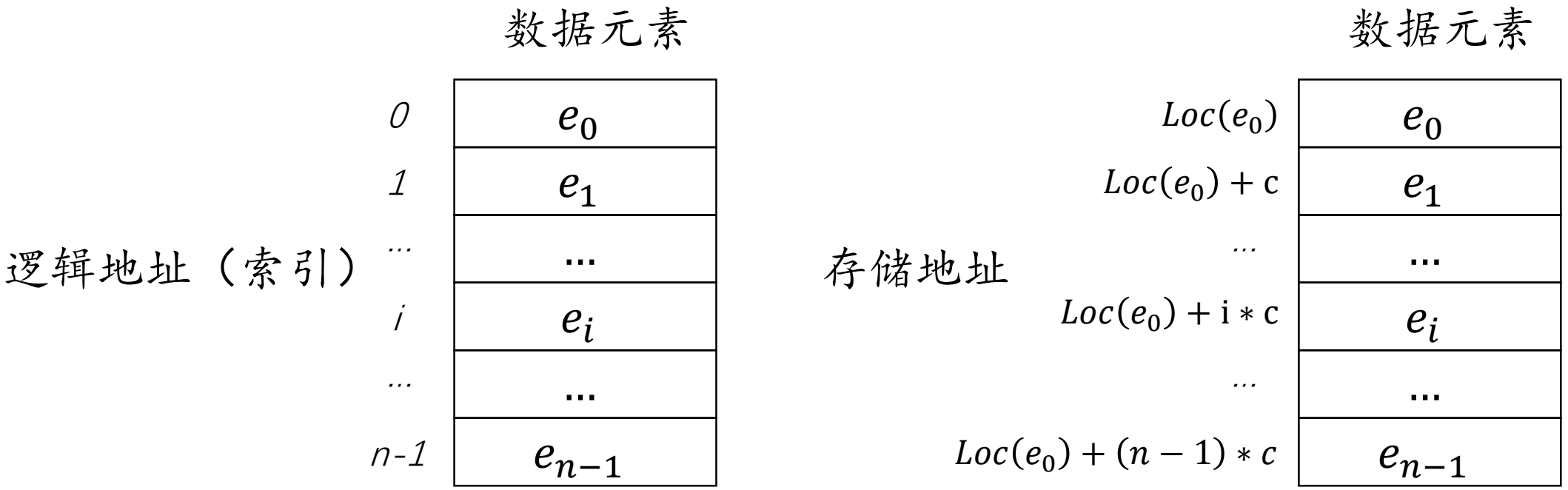
2.1 数组

数组（Array）

- 又称顺序表、向量，采用定长的一维数组存储结构
- 主要特性
  - 元素的类型相同
  - 元素顺序地存储在连续内存空间中，每个元素有唯一的索引值
  - 使用常数作为长度
- 读写其元素很方便，通过下标即可指定位置
  - 只要确定了首地址，线性表中任意数据元素都可以随机存取

数组访问

- 元素地址计算如下所示
  - 元素$Loc(e_i) = Loc(e_0) + i * c, c = sizeof(ELEMENT)$



常见操作复杂度

	插入	删除
首		
尾		
中间		

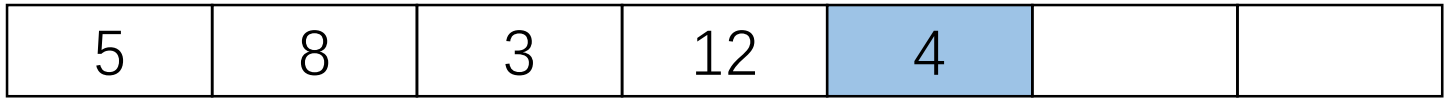
常见操作复杂度

	插入	删除
首		
尾		
中间		

5	8	3	12			

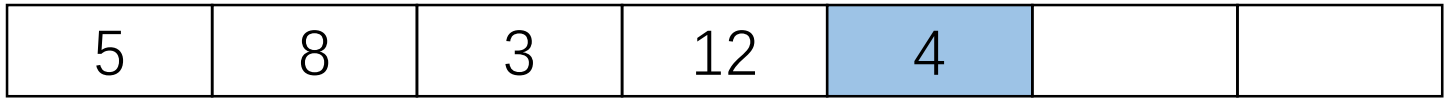
常见操作复杂度

	插入	删除
首		
尾		
中间		



常见操作复杂度

	插入	删除
首		
尾	$O(1)$	
中间		



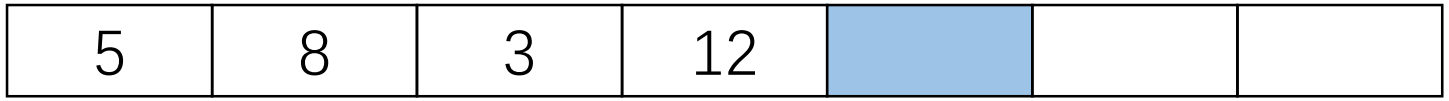
常见操作复杂度

	插入	删除
首		
尾	$O(1)$	
中间		

5	8	3	12	4		

常见操作复杂度

	插入	删除
首		
尾	$O(1)$	
中间		



常见操作复杂度

	插入	删除
首		
尾	$O(1)$	$O(1)$
中间		

5	8	3	12			

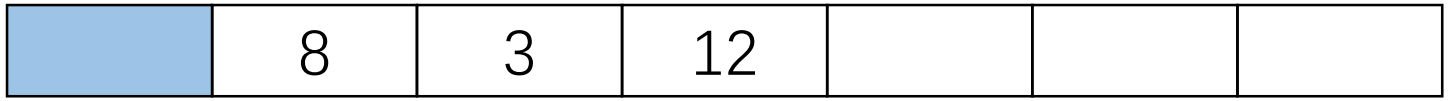
常见操作复杂度

	插入	删除
首		
尾	$O(1)$	$O(1)$
中间		

5	8	3	12			

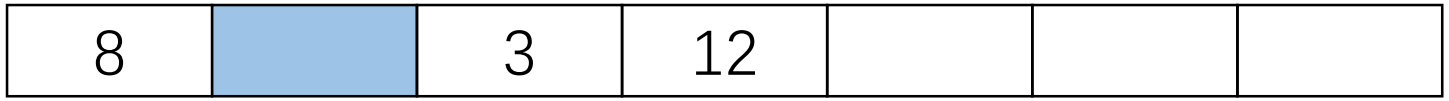
常见操作复杂度

	插入	删除
首		
尾	$O(1)$	$O(1)$
中间		



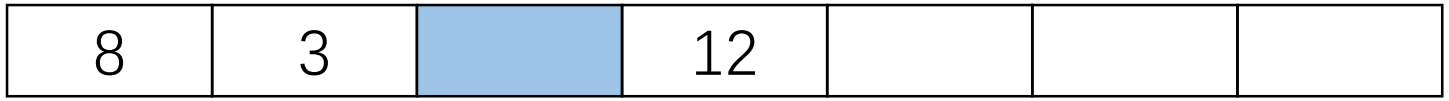
常见操作复杂度

	插入	删除
首		
尾	$O(1)$	$O(1)$
中间		



常见操作复杂度

	插入	删除
首		
尾	$O(1)$	$O(1)$
中间		



常见操作复杂度

	插入	删除
首		
尾	$O(1)$	$O(1)$
中间		

常见操作复杂度

	插入	删除
首		$O(n)$
尾	$O(1)$	$O(1)$
中间		

8	3	12				

常见操作复杂度

	插入	删除
首		$O(n)$
尾	$O(1)$	$O(1)$
中间		

8	3	12				

常见操作复杂度

	插入	删除
首	$O(n)$	$O(n)$
尾	$O(1)$	$O(1)$
中间		

8	3	12				
8	3		12			
8		3	12			
	8	3	12			
5	8	3	12			

常见操作复杂度

	插入	删除
首	$O(n)$	$O(n)$
尾	$O(1)$	$O(1)$
中间	???	???

5	8	3	12			

常见操作复杂度

	插入	删除
首	$O(n)$	$O(n)$
尾	$O(1)$	$O(1)$
中间	$O(n)$	$O(n)$

5	8	3	12			

数组ADT接口

操作	功能	适用对象
size()	返回数组当前的规模（元素总数）	数组
get(i)	获取第i个元素	数组
put(i,e)	用e替换第i个元素的数值	数组
insert(i,e)	将e插入到i位置，后继元素依次后移	数组
remove(i)	删除第i个元素，返回该元素中原存放的对象	数组
disordered()	判断所有元素是否已按非降序排列	数组
sort()	调整各元素的位置，使之按非降序排列	数组
find(e)	查找目标元素e	数组
search(e)	查找目标元素e，返回不大于e且位置最靠近e的元素	有序数组
deduplicate()	剔除重复元素	数组
uniquify()	剔除重复元素	有序数组
traverse()	遍历数组并统一处理所有元素，处理方法由函数对象指定	数组

问题

- 在一个初始为空的数组上依次执行：insert(0, 2), insert(1, 6), put(0, 1), remove(1), insert(0, 7) 后的结果是？

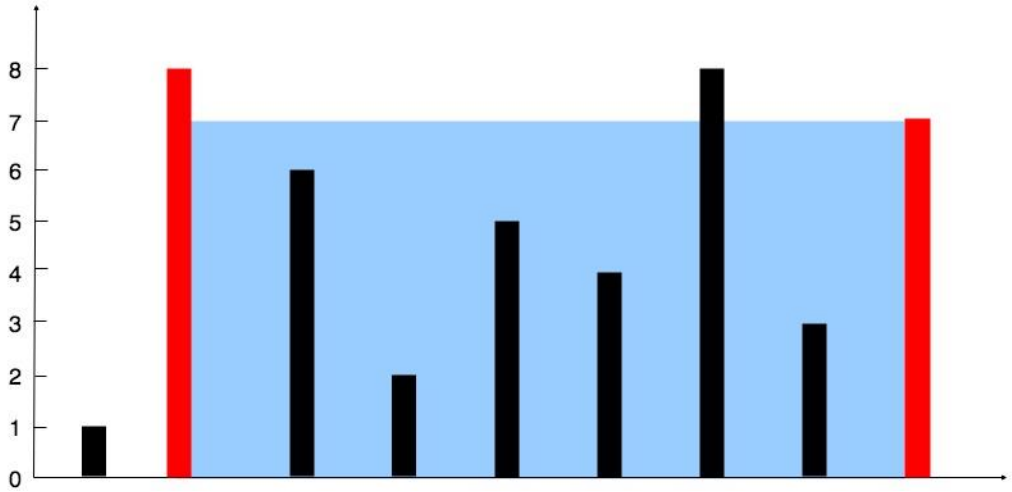
问题

- 在一个初始为空的数组上依次执行：insert(0, 2), insert(1, 6), put(0, 1), remove(1), insert(0, 7) 后的结果是？

{7, 1}

问题：盛最多水的容器

- 给定n个非负整数$a_1, a_2, \ldots, a_n$，每个数代表坐标中的一个点(i, a_i)。在坐标内画n条垂直线，垂直线i的两个端点分别为(i, a_i)和$(i, 0)$。找出其中的两条线，使得它们与x轴共同构成的容器可以容纳最多的水。
- 说明：你不能倾斜容器，且n的值至少为2。



- 输入：[1, 8, 6, 2, 5, 4, 8, 3, 7]
- 输出：49

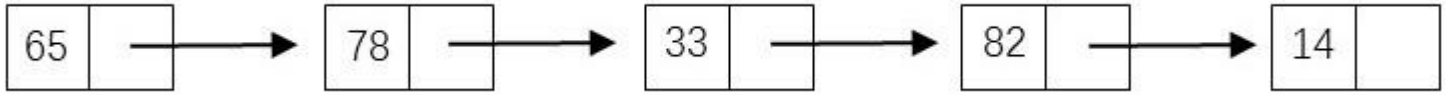
2.2 链表

链表（linked list）

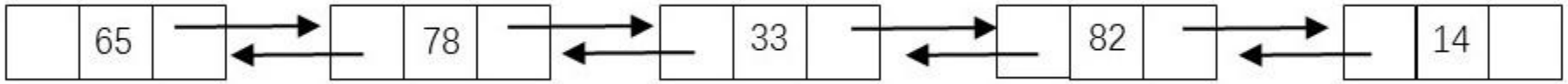
- 链表是一个存储节点的序列：
  - 每个节点中的信息存储在连续的内存单元中
  - 每个节点保存了指向后继节点（前驱节点）的指针。指向某个节点的指针，实际是该节点在内存中的地址
    - 我们将指向后继节点的指针，称为后指针；指向前驱节点的指针，称为前指针
- 链表的第一个节点称为表头，最后一个节点称为表尾

链表分类

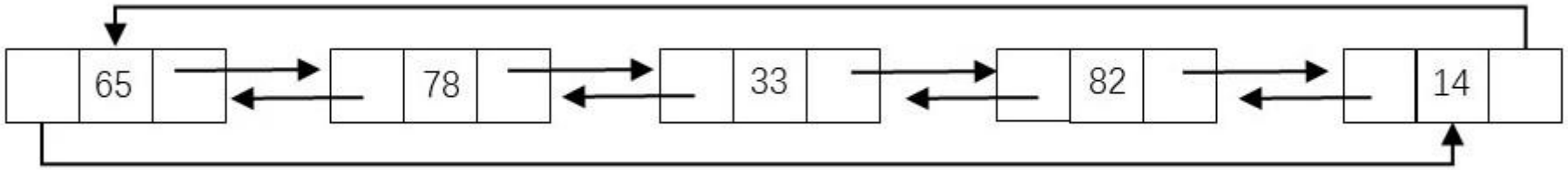
- 单链表



- 双链表

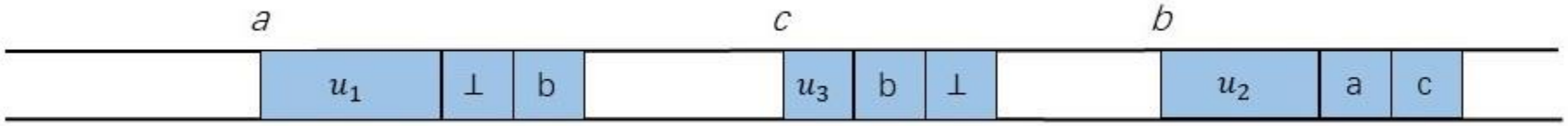


- 循环链表



链表（linked list）

- 下图表示内存中一个链表的三个节点u_1，u_2，u_3的存储状态，他们的地址分别为a，b，c

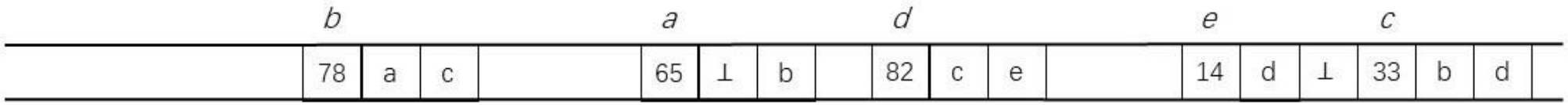


- 节点u_1（表头）的前指针为空，表示为⊥；节点u_2（表尾）的后指针也为空。

每个节点可以存储在内存的任意位置，节点的地址也不需要按照其在链表中的顺序排列。

链表示例

- 下图为内存中一个包含5个整数的链表 {14, 65, 78, 33, 82}



- 逻辑表示为：

链表特性

- 假设我们用链表保存n个整数的集合S，一个节点保存一个整数

1. 链表的空间复杂度为$O(n)$，其占用了$O(n)$内存单元
2. 从表头开始，遍历一遍链表的时间复杂度为$O(n)$

链表操作

- 链表保存集合S支持针对集合的更新操作

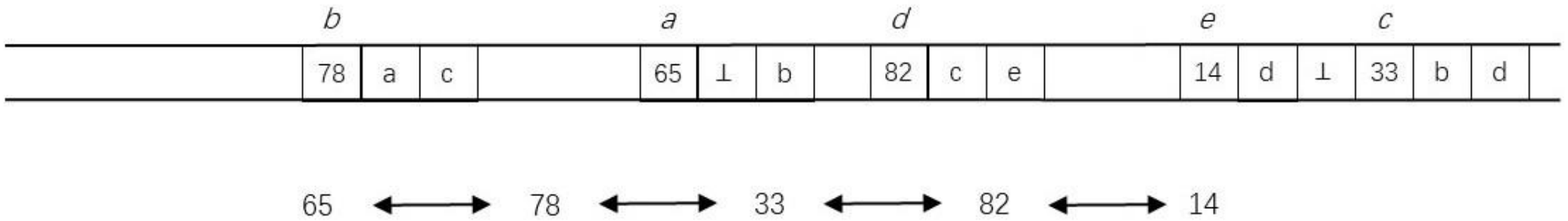
1. 插入：增加一个新的元素到集合S
2. 删除：删除一个集合S中存在的元素

链表插入

- 插入新元素e到集合S中，我们简单考虑在链表表尾插入
  - 找到表尾u
  - 新建一个新的节点u_{new}
  - 将u的后指针指向新节点u_{new}的地址
  - 将u_{new}的前指针指向u的地址
- u_{new}变成新的链表表尾
- 如果已知表尾的地址，此插入操作时间为$O(1)$

链表插入示例

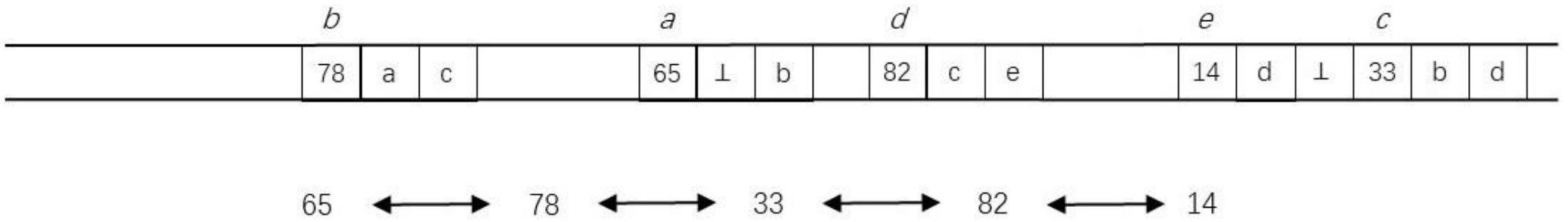
- 插入前



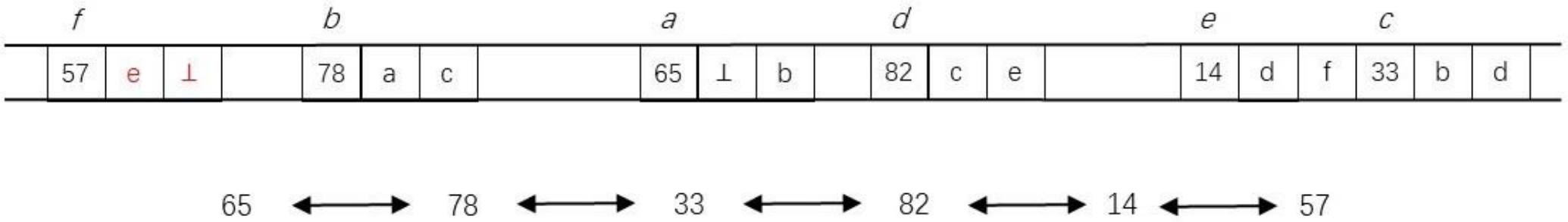
- 插入57后

链表插入示例

- 插入前

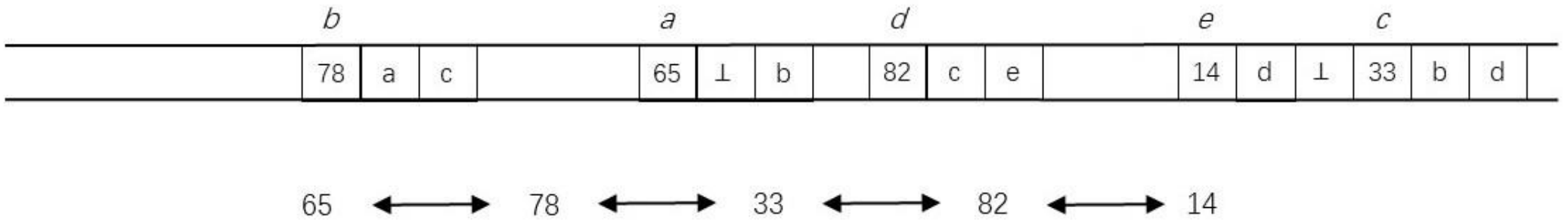


- 插入57后

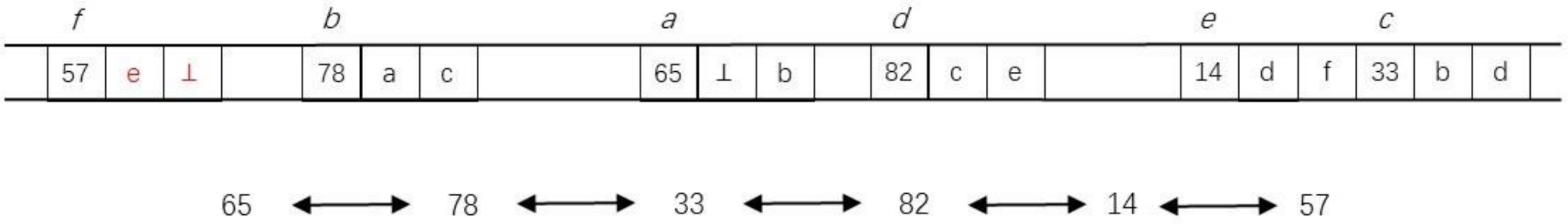


链表插入示例

- 插入前



- 插入57后



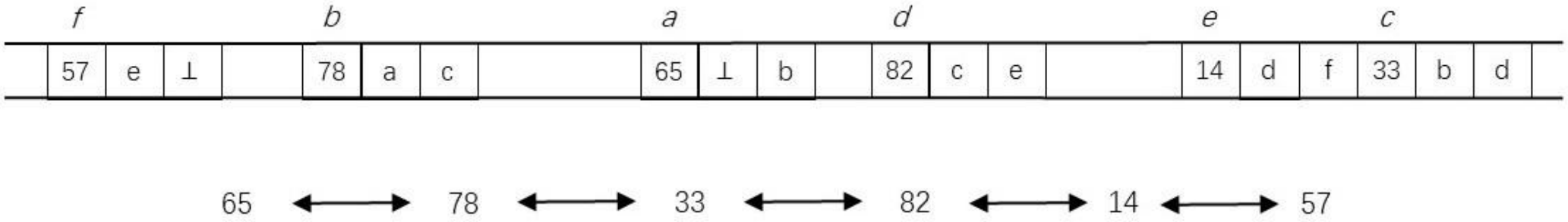
新节点的地址可以是内存的任意地址

链表删除

- 给定一个想要删除的节点u的地址，
  - 找到u的前驱节点u_{prec}
  - 找到u的后续节点u_{succ}
  - 将前驱节点u_{prec}的后指针指向后续节点u_{succ}
  - 将后续节点u_{succ}的前指针指向前驱节点u_{prec}
  - 释放节点u的内存
- 此删除操作时间为$O(1)$

链表删除示例

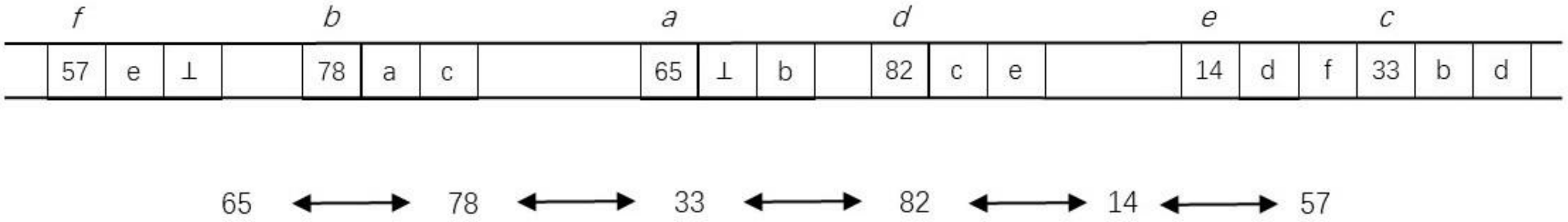
- 删除前



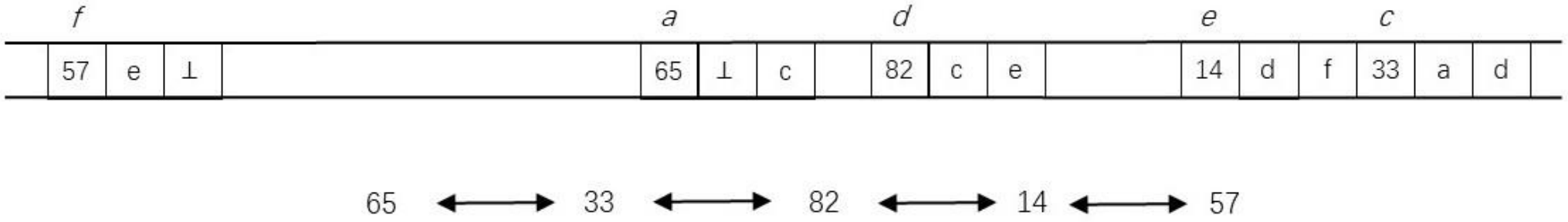
- 删除78后

链表删除示例

- 删除前



- 删除78后



链表运算分析

- 假设我们使用链表保存n个整数的集合S
  - 空间开销$O(n)$
  - 表尾插入时间开销$O(1)$：已知表尾地址
  - 删除时间开销$O(1)$：已知待删除节点地址
  - 遍历链表时间开销$O(n)$：已知表头或者表尾地址

其他操作复杂度

操作	作用	表尾地址未知	表尾地址已知
PushFront	add to front		
Key TopFront	return front item		
PopFront	remove front item		
PushBack(Key)	add to back		
Key TopBack()	return back item		
PopBack()	remove back item		
Boolean Find(Key)	is key in list?		
Erase(Key)	remove key from list		
Boolean Empty()	empty list?		
AddBefore(Node, Key)	adds key before node		
AddAfter(Node, Key)	adds key after node		

单链表操作复杂度

操作	作用	表尾地址未知	表尾地址已知
PushFront	add to front	$O(1)$	
Key TopFront	return front item	$O(1)$	
PopFront	remove front item	$O(1)$	
PushBack(Key)	add to back	$O(n)$	$O(1)$
Key TopBack()	return back item	$O(n)$	$O(1)$
PopBack()	remove back item	$O(n)$	
Boolean Find(Key)	is key in list?	$O(n)$	
Erase(Key)	remove key from list	$O(n)$	
Boolean Empty()	empty list?	$O(1)$	
AddBefore(Node, Key)	adds key before node	$O(n)$	
AddAfter(Node, Key)	adds key after node	$O(1)$	

双链表操作复杂度

操作	作用	表尾地址未知	表尾地址已知
PushFront	add to front		
Key TopFront	return front item		
PopFront	remove front item		
PushBack(Key)	add to back		
Key TopBack()	return back item		
PopBack()	remove back item		
Boolean Find(Key)	is key in list?		
Erase(Key)	remove key from list		
Boolean Empty()	empty list?		
AddBefore(Node, Key)	adds key before node		
AddAfter(Node, Key)	adds key after node		

双链表操作复杂度

操作	作用	表尾地址未知	表尾地址已知
PushFront	add to front	$O(1)$	
Key TopFront	return front item	$O(1)$	
PopFront	remove front item	$O(1)$	
PushBack(Key)	add to back	$O(n)$	$O(1)$
Key TopBack()	return back item	$O(n)$	$O(1)$
PopBack()	remove back item	$O(1)$	
Boolean Find(Key)	is key in list?	$O(n)$	
Erase(Key)	remove key from list	$O(n)$	
Boolean Empty()	empty list?	$O(1)$	
AddBefore(Node, Key)	adds key before node	$O(1)$	
AddAfter(Node, Key)	adds key after node	$O(1)$	

问题

- 给定一个链表，删除链表的倒数第n个节点，并且返回链表的头结点。
- 示例：给定一个链表: $1->2->3->4->5$, 和 $n=2$。当删除了倒数第二个节点后，链表变为 $1->2->3->5$。
- 说明：给定的n保证是有效的。

问题

- 给定一个链表，删除链表的倒数第n个节点，并且返回链表的头结点。
- 示例：给定一个链表: $1->2->3->4->5$, 和 $n=2$。当删除了倒数第二个节点后，链表变为 $1->2->3->5$。
- 说明：给定的n保证是有效的。
- 进阶：你能尝试使用一趟扫描实现吗？

基础线性表比较

■ 数组的主要优点

- 没有使用指针，不用花费额外开销
- 数组元素的读访问非常简洁便利

■ 链表的主要优点

- 无需事先了解链表的长度
- 允许链表的长度动态变化
- 能够适应经常插入删除内部元素的情况

■ 总结

- 数组是存储静态数据的不二选择
- 链表是存储动态变化数据的良方

2.2 栈（Stack）

栈

- 使用链表实现栈
- 使用栈保存n个元素的集合S，支持两种操作
  - Push(e)：插入一个新元素到集合S
  - Pop()：删除S中最近一个插入的元素，并返回它

栈

- 使用链表实现栈
- 使用栈保存n个元素的集合S，支持两种操作
  - Push(e)：插入一个新元素到集合S
  - Pop()：删除S中最近一个插入的元素，并返回它
- 换句话说，栈遵循First-In-Last-Out（FILO）原则。也就是，先进后出（后进先出）

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = ?$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = ?$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = ?$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = \{35, 23, 79\}$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = \{35, 23, 79\}$
  - Pop：$S = ?$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = \{35, 23, 79\}$
  - Pop：$S = \{35, 23\}$。返回79，并将其从S中删除

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = \{35, 23, 79\}$
  - Pop：$S = \{35, 23\}$。返回79，并将其从S中删除
  - Pop：$S = ?$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = \{35, 23, 79\}$
  - Pop：$S = \{35, 23\}$。返回79，并将其从S中删除
  - Pop：$S = \{35\}$。返回23，并将其从S中删除

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = \{35, 23, 79\}$
  - Pop：$S = \{35, 23\}$。返回79，并将其从S中删除
  - Pop：$S = \{35\}$。返回23，并将其从S中删除
  - Push(47)：$S = ?$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = \{35, 23, 79\}$
  - Pop：$S = \{35, 23\}$。返回79，并将其从S中删除
  - Pop：$S = \{35\}$。返回23，并将其从S中删除
  - Push(47)：$S = \{35, 47\}$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = \{35, 23, 79\}$
  - Pop：$S = \{35, 23\}$。返回79，并将其从S中删除
  - Pop：$S = \{35\}$。返回23，并将其从S中删除
  - Push(47)：$S = \{35, 47\}$
  - Pop：$S = ?$

栈示例

- 假设栈为空，考虑下列操作
  - Push(35)：$S = \{35\}$
  - Push(23)：$S = \{35, 23\}$
  - Push(79)：$S = \{35, 23, 79\}$
  - Pop：$S = \{35, 23\}$。返回79，并将其从S中删除
  - Pop：$S = \{35\}$。返回23，并将其从S中删除
  - Push(47)：$S = \{35, 47\}$
  - Pop：$S = \{35\}$。返回47，并将其从S中删除

使用链表实现栈

- 在任意时刻，都将集合S中的元素存储在链表L中
- Push(e)：插入e到L的表尾
- Pop：删除L的表尾，同时返回表尾中的元素
- 在任意时刻，都保存表尾的内存地址

1. 空间开销$O(n)$
2. Push时间开销$O(1)$
3. Pop时间开销$O(1)$

问题

- 给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。
- 有效字符串需满足：
    1. 左括号必须用相同类型的右括号闭合。
    2. 左括号必须以正确的顺序闭合。
- 注意空字符串可被认为是有效字符串。

- 示例 1: 输入: "()" 输出: true
- 示例 2: 输入: "()[]{}" 输出: true
- 示例 3: 输入: "(]" 输出: false
- 示例 4: 输入: "([)]" 输出: false
- 示例 5: 输入: "{[]}" 输出: true

2.3 队列（Queue）

队列

- 使用链表实现队列
- 使用队列保存n个元素的集合S，支持两种操作：
  - En-queue(e)：插入一个新元素e到集合S
  - De-queue()：删除S中最后一个插入的元素，并返回它

队列

- 使用链表实现队列
- 使用队列保存n个元素的集合S，支持两种操作：
  - En-queue(e)：插入一个新元素e到集合S
  - De-queue()：删除S中最后一个插入的元素，并返回它
- 队列遵循First-In-First-Out（FIFO）原则。也就是，先进先出（后进后出）

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = ?$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = ?$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = ?$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = \{35, 23, 79\}$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = \{35, 23, 79\}$
  - De-queue：$S = ?$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = \{35, 23, 79\}$
  - De-queue：$S = \{23, 79\}$。返回35，并将其从S中删除

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = \{35, 23, 79\}$
  - De-queue：$S = \{23, 79\}$。返回35，并将其从S中删除
  - De-queue：$S = ?$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = \{35, 23, 79\}$
  - De-queue：$S = \{23, 79\}$。返回35，并将其从S中删除
  - De-queue：$S = \{79\}$。返回23，并将其从S中删除

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = \{35, 23, 79\}$
  - De-queue：$S = \{23, 79\}$。返回35，并将其从S中删除
  - De-queue：$S = \{79\}$。返回23，并将其从S中删除
  - En-queue(47)：$S = ?$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = \{35, 23, 79\}$
  - De-queue：$S = \{23, 79\}$。返回35，并将其从S中删除
  - De-queue：$S = \{79\}$。返回23，并将其从S中删除
  - En-queue(47)：$S = \{79, 47\}$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = \{35, 23, 79\}$
  - De-queue：$S = \{23, 79\}$。返回35，并将其从S中删除
  - De-queue：$S = \{79\}$。返回23，并将其从S中删除
  - En-queue(47)：$S = \{79, 47\}$
  - De-queue：$S = ?$

队列示例

- 假设队列为空，考虑下列操作
  - En-queue(35)：$S = \{35\}$
  - En-queue(23)：$S = \{35, 23\}$
  - En-queue(79)：$S = \{35, 23, 79\}$
  - De-queue：$S = \{23, 79\}$。返回35，并将其从S中删除
  - De-queue：$S = \{79\}$。返回23，并将其从S中删除
  - En-queue(47)：$S = \{79, 47\}$
  - De-queue：$S = \{47\}$。返回79，并将其从S中删除

使用链表实现队列

- 在任意时刻，都将集合S中的元素存储在链表L中
- En-queue(e)：插入e到L的表尾
- De-queue：删除L的表头，同时返回表头中的元素
- 在任意时刻，都保存表头和表尾的内存地址

1. 空间开销$O(n)$
2. Enqueue时间开销$O(1)$
3. Dequeue时间开销$O(1)$

问题

- 设计你的循环队列实现。循环队列是一种线性数据结构，其操作表现基于FIFO（先进先出）原则并且队尾被连接在队首之后以形成一个循环。它也被称为“环形缓冲器”。
- 循环队列的一个好处是我们可以利用这个队列之前用过的空间。在一个普通队列里，一旦一个队列满了，我们就不能插入下一个元素，即使在队列前面仍有空间。但是使用循环队列，我们能使用这些空间去存储新的值。
- 你的实现应该支持如下操作：
- MyCircularQueue(k): 构造器，设置队列长度为 k 。
- Front: 从队首获取元素。如果队列为空，返回 -1 。
- Rear: 获取队尾元素。如果队列为空，返回 -1 。
- enQueue(value): 向循环队列插入一个元素。如果成功插入则返回真。
- deQueue(): 从循环队列中删除一个元素。如果成功删除则返回真。
- isEmpty(): 检查循环队列是否为空。
- isFull(): 检查循环队列是否已满。

总结

1. 数组
2. 链表
3. 栈
4. 队列