栈 Stack

1 基本信息

1.1 介绍

栈是一个序列，其中元素只能在**一端(顶部)**被添加和删除

并且只能访问当前位于**顶部**的元素

特点：**先进后出型 First In Last Out**

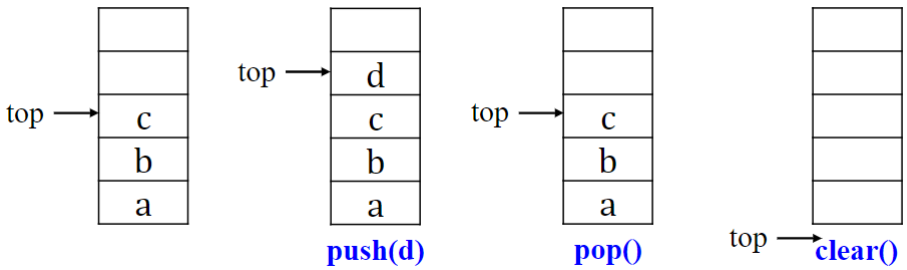
1.2 操作

基本操作

操作	解释	示例
压栈push	将一个元素添加到栈的顶部	stack.push(item)
出栈pop	弹出并删除栈顶部的元素	stack.pop()

其它操作

操作	解释	示例
偷看peak	查看栈顶的元素	stack.peek()
是否为空isEmpty	查看栈是否为空	stack.isEmpty()
是否满了is Full	查看栈是否已满	stack.isFull()
清空clear	清空当前栈	stack.clear()
栈的大小size	返回当前栈的大小	stack.size()



所有的访问只能使用**top**指针

1.3 应用

确保分隔符(括号)是平衡的

计算算术表达式

内存中的运行时栈

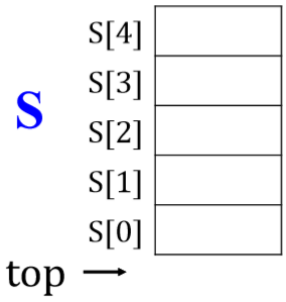
2 栈的实现

2.1 基于数组

由于数组的内存开销规定好了，所以栈开始就有最大值

初始状态

- MAX_SIZE = n
- top = -1
- Array S



伪代码：PUSH

```
Algorithm:PUSH(item):
//数组要考虑栈满了的情况
if top = MAX_SIZE - 1 then
    return false
else
    top++
    S[top] ← item
    return true
```

伪代码：POP

```
Algorithm:POP():
//考虑栈是空的情况
if top = -1 then
    return null
else
    returnVal ← S[top]
    top--
    return returnVal
```

2.2 基于链表

链表的内存不会被规定，所以在不限制MAX_SIZE的前提下是可以无限增加的

伪代码：PUSH

```
Algorithm:PUSH(Node):
if isEmpty() then:
    top ← Node
else
    top.next ← Node
    Node.prev ← top
    top ← top.next
return true
```

伪代码：POP

```
Algorithm:POP():
if isEmpty() then:
    return null
else if size = 1 then
    returnNode ← top
    top ← null
    return returnNode
else
    returnNode ← top
    top ← top.prev
    returnNode.prev ← null
    top.next ← null
    return returnNode
```

2.3 括号匹配问题

思路

如果遇到 { [(等左括号，则把它们压栈

如果遇到 }]) 等右括号，则再弹出栈顶元素，查看是否与该括号匹配

- 如果匹配，则可以继续操作
- 如果不匹配，说明括号不匹配

伪代码

```
Algorithm:Brackets Balance(Brackets[])
n = len(Brackets[])
create new stack

for i ← 0 to n-1
    switch Brackets[i]:
        case leftBracket
            stack.push(Brackets[i])
        case rightBracket
            if stack.pop() match Brackets[i]
                continue
            else
                return false
end

//最后要判断栈是否是空的
if stack.size = 0 then
```

```
    return true
else
    return false
```

2.4 算术表达式问题

前缀和/中缀和/后缀和

算术表达式

- 操作符 + - * /
- 操作数 a b c d 1 2 3

中缀

- $a + b * c$

前缀

- $+a * bc$

后缀

- $abc * +$

后缀表达式的好处

不需要括号即可完成运算

❓问题：中缀和转换

> 已知中缀和 5 * ((9 + 3) * (4*2) + 7)
>
> 将其写成前缀和和后缀和的形式

前缀表达

后缀表达 5 9 3 + 4 2 * * 7 + *

思路

一次读取一个符号

如果它是**操作数**，则把它**压栈**

如果它是**操作符**

- 弹出站内两个操作数
- 对它使用操作符进行操作
- 把得到的结果再压栈

❓问题：后缀运算表达

> 写出5 9 3 + 4 2 * * 7 + * 的运算操作

读取	栈的操作	栈内元素
5	push(5)	5
9	push(9)	5 9
3	push(3)	5 9 3
+	push(pop()+pop())	5 12
4	push(4)	5 12 4
2	push(2)	5 12 4 2
*	push(pop()*pop())	5 12 8
*	push(pop()*pop())	5 96
7	push(7)	5 96 7
+	push(pop()+pop())	5 103
*	push(pop()+pop())	515

队列 Queue

1 基本信息

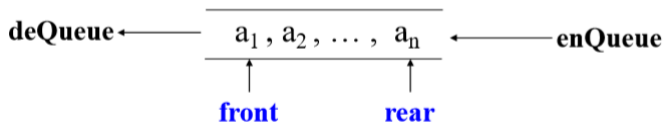
1.1 介绍

队列是一个序列，其中元素只能在**头部**被移除或者**尾部**被添加

并且只能访问当前位于**顶部**的元素

特点：**先进先出型 First In First Out**

Illustration



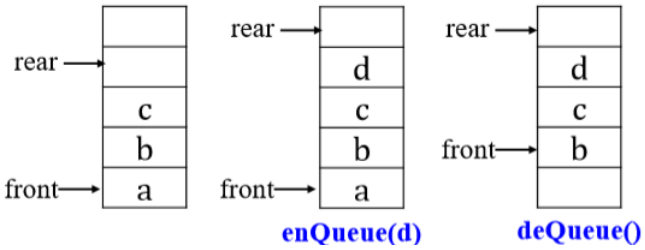
1.2 操作

基本操作

操作	解释	示例
入队enQueue	将一个元素添加到队列的尾部	queue.enQueue(item)
出队deQueue	出队并删除队列头部的元素	queue.deQueue

其它操作

操作	解释	示例
偷看front	查看队列头的元素	queue.front()
是否为空isEmpty	查看队列是否为空	queue.isEmpty()
是否满了is Full	查看队列是否已满	queue.isFull()
清空clear	清空当前队列	queue.clear()
队列的大小size	返回当前队列的大小	queue.size()



入队只能使用**rear**指针，出队只能使用**front**指针

1.3 应用

先进先出(FIFO)库存控制

- 操作系统调度:进程、打印作业、数据包等
- 图的宽度优先遍历或二叉树的层次顺序遍历

实际应用

- iTunes播放列表。TiVo
- 数据缓冲区(iPod)
- 异步数据传输(文件IO、管道、套接字)

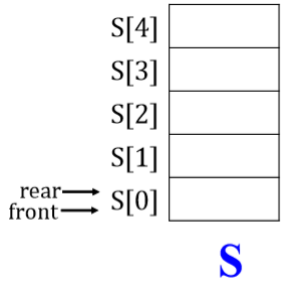
2 队列的实现

2.1 基于数组

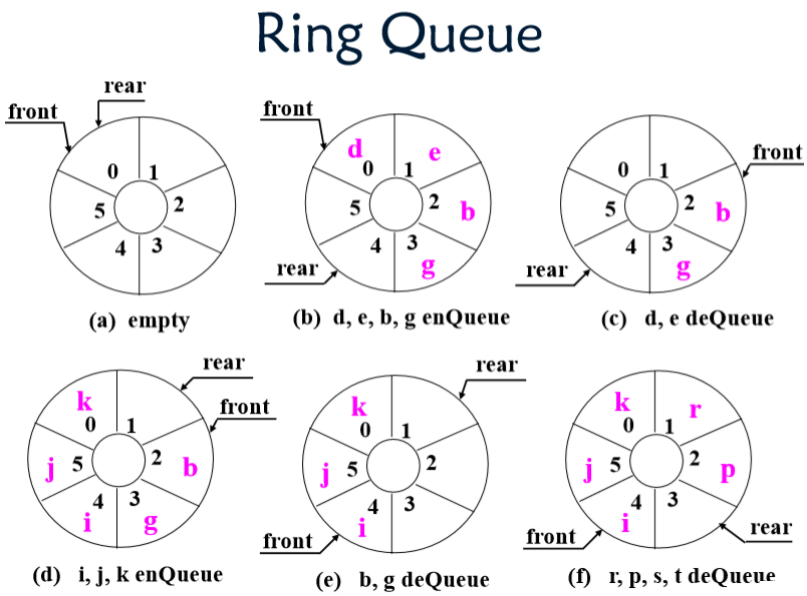
由于数组的内存开销规定好了，所以队列开始就有最大值

初始状态

- MAX_SIZE = n
- front = 0
- rear = 0
- Array S



需要注意当指针指到数组顶的情况，指针应该回移从0开始



队满：Q.front == (Q.rear + 1) % MAXSIZE

队空：Q.front == Q.rear

入队：Q.rear = (Q.rear + 1) % MAXSIZE

2.2 基于链表

链表的内存不会被规定，所以在不限制MAX_SIZE的前提下是可以无限增加的

❗伪代码：enQueue

```
Algorithm:enQueue(Node):
if isEmpty() then:
    front ← Node
    rear ← Node
    return true
else:
    rear.next ← Node
    Node.prev ← prev
    rear ← Node
    return true
```

❗伪代码：deQueue

```
Algorithm:deQueue():
if isEmpty() then:
    return null
else if size = 1 then
    returnNode ← front
    rear ← null
    front ← null
    return returnNode
else
    returnNode ← front
    front ← front.next
    front.prev ← null
    returnNode.next ← null
    return returnNode
```

栈和队列

1 对比

	Stack	Queue
In-Out	FILO	FIFO
Application	function runtime	OS scheduling
Operations	push pop	enQueue, deQueue
Ops Time Complexity	O(1)	O(1)
Implementation	Array-based, Linked-based	Array-based, Linked-based