1. 课程介绍

* 1. 什么是栈/队列;（了解）
* 2. 为什么需要栈/队列;（了解）
* 3. 栈/队列的操作和实现;（掌握）
* 4. 双端队列的含义及操作.(了解)

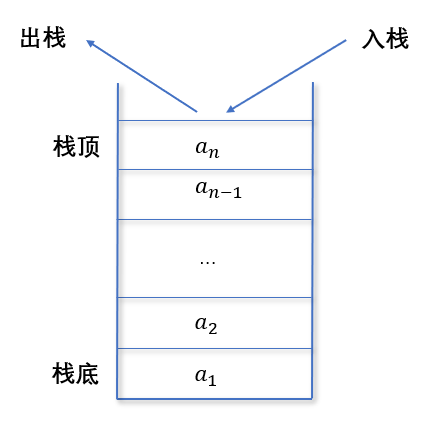
1. 栈与队列的基本概念

栈与队列是两种重要的线性结构,其特殊性在于,它们的**操作**要受到一定的**限制**. 现实生活中,许多问题分解的步骤和求解的步骤次序往往相反,这是栈的应用背景. 排队无处不在,队列结构的应用背景就来自排队问题.

2.1 什么是栈？

**栈(stack)**,又称**堆栈**,是保证元素**后进先出(Last In First Out)**关系的线性表.

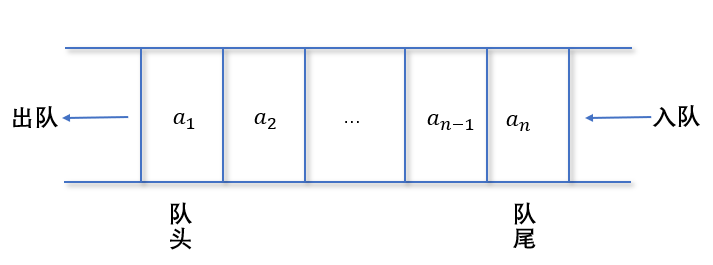
* + - 1. 栈中元素只有**一种**关系:到来的先后顺序.
      2. 栈是一种**容器**,可存入,访问和删除栈中的数据.
      3. 栈顶与栈底,入栈与出栈
      4. **只能在栈顶存入,访问和删除数据**.



* 1. 什么是队列?

**队列(queue)**是保证元素**先进先出**(First In First Out)关系的线性表.

* + - 1. 队列中没有**位置**的概念.
      2. 队列是一种容器,可存入,访问和删除队列中的数据
      3. 队头和队尾,入队和出队
      4. 队列只支持默认方式的元素存入和取出.
      5. **队列是只能在一端存入元素,在另一端删除操作的线性表.**(不允许在其他位置进行操作)



* 1. 栈的实现

我们可以用python程序来实现栈这一数据结构. 同顺序表一样,我们将栈定义成类.

1. #!/usr/bin/python
2. # -\*- coding: utf-8 -\*-
3. """
4. 基于顺序表技术实现的栈类
5. """
6. **class** SStack():
7. **def** \_\_init\_\_(self):
8. self.\_\_list = [] #生成一个容器
10. **def** is\_empty(self):
11. """判断是否为空"""
12. **return** self.\_\_list == []
14. **def** top(self):
15. """访问栈顶元素"""
16. **if** self.\_\_list == []:
17. **return** None
18. **return** self.\_\_list[-1]
20. **def** push(self, elem):
21. """向栈顶存入元素"""
22. self.\_\_list.append(elem) # 将列表(顺序表)尾部作为栈顶
24. **def** pop(self):
25. """从栈顶删除元素"""
26. **if** self.\_\_list == []:
27. **return** None
28. **return** self.\_\_list.pop()
30. **def** size(self):
31. """求元素个数"""
32. **return** len(self.\_\_list)
34. **if** \_\_name\_\_ == "\_\_main\_\_":
35. st1 = SStack()
36. st1.push(3)
37. st1.push(5)
38. st1.push(7)
39. **print** st1.pop()
40. **print** st1.pop()
41. **print** st1.pop()
    1. 栈的操作

1. 判断是否为空: is\_empty()

2. 求栈的元素个数: size()

3. 入栈: push(item)

4. 出栈 : pop()

5. 访问栈顶元素: top()

* 1. 队列的实现

用Python代码实现队列:

1. #!/usr/bin/python
2. # -\*- coding: utf-8 -\*-
3. """
4. 基于顺序表技术实现的队列类
5. """
6. **class** Queue():
7. """队列"""
8. **def** \_\_init\_\_(self):
9. self.\_\_list = [] #生成一个容器
11. **def** is\_empty(self):
12. """判断是否为空"""
13. **return** self.\_\_list == []
15. **def** enqueue(self, item):
16. """入队:将item添加到队尾"""
17. self.\_\_list.append(item) # 顺序表尾部作为队尾
18. #self.\_\_list.insert(0,item) # 顺序表头部作为队尾
20. **def** dequeue(self):
21. """出队: 从队头删除一个元素"""
22. **if** self.\_\_list == []:
23. **return** None
24. **return** self.\_\_list.pop(0)  # 顺序表尾部作为队尾
25. #return self.\_\_list.pop()  # 顺序表头部作为队尾
26. **def** size(self):
27. """返回队列的大小"""
28. **return** len(self.\_\_list)
30. **if** \_\_name\_\_ == "\_\_main\_\_":
31. q1 = Queue()
32. q1.enqueue(3)
33. q1.enqueue(5)
34. q1.enqueue(7)
35. **print** "size of queue:", q1.size()
36. **print** q1.dequeue()
37. **print** q1.dequeue()
38. **print** q1.dequeue()
39. **print** "size of queue:",q1.size()
    1. 双端队列

队列的扩展:双端队列

* + 1. 什么是双端队列(deque)?

双端队列是一种具有队列和栈的性质的数据结构.



特征: 双端队列可在任意一端出队和入队.

* + 1. 双端队列的实现

1.双端队列的操作:

创建双端队列

deque()

队列是否为空

is\_empty()

从队头加入元素

add\_front(item)

从队尾加入元素

add\_rear(item)

从队头删除一个元素

rm\_front()

从队尾删除一个元素

rm\_rear()

元素个数(返回双端队列的大小)

size()

2.双端队列的实现(代码示例)

1. #!/usr/bin/python
2. # -\*- coding: utf-8 -\*-
3. """
4. 基于顺序表技术实现的双端队列类
5. """
6. **class** Dequeue():
7. """队列"""
8. **def** \_\_init\_\_(self):
9. self.\_\_list = [] #生成一个容器
11. **def** is\_empty(self):
12. """判断是否为空"""
13. **return** self.\_\_list == []
15. **def** add\_front(self, item):
16. """入队:将item添加到队头"""
17. self.\_\_list.insert(0,item)
19. **def** add\_rear(self, item):
20. """入队:将item添加到队尾"""
21. self.\_\_list.append(item)
23. **def** rm\_front(self):
24. """出队: 从队头删除一个元素"""
25. **if** self.\_\_list == []:
26. **return** None
27. **return** self.\_\_list.pop(0)
29. **def** rm\_rear(self):
30. """出队: 从队尾删除一个元素"""
31. **if** self.\_\_list == []:
32. **return** None
33. **return** self.\_\_list.pop()
35. **def** size(self):
36. """返回队列的大小"""
37. **return** len(self.\_\_list)
39. **if** \_\_name\_\_ == "\_\_main\_\_":
40. q1 = Dequeue()
41. q1.add\_front(3)
42. q1.add\_front(5)
43. q1.add\_front(7)
44. **print** "size of queue:", q1.size()
45. q1.add\_rear(2)
46. q1.add\_rear(4)
47. q1.add\_rear(6)
48. **print** "size of queue:", q1.size()
    1. 应用场景
       1. Python3中的模块queue的使用

(1)一般队列

from queue import Queue # 导入队列模块

q\_example = queue.Queue(maxsize = 500) # 建立一个最大尺寸为500的队列

# 判断队列是否为空

q\_example.empty()

#如果队列为空，返回True,反之False

#判断队列是否已满

q\_example.full() #如果队列满了，返回True,反之False

#加入元素到队列

q\_example.put(100)

# 取出一个元素

q\_example.get()

# 依次取出多个元素

while not q\_example.empty():

q\_example.get()

#返回队列的大小

q\_example.qsize()

(2)栈

from queue import LifoQueue

注意: 在Python2中,该模块的使用方法有些差别.

这些模块中,对于队列等容器,已经有预先定义好的常见操作(方法),如判断是否为空,插入结点,取出结点,判断容器是否已满等.

* + 1. 搜索问题

另外,在搜索问题中,我们往往把待搜索的结点放入队列或栈中,从而可以按照某种顺序来实现对结点的搜索.

1. 课程总结
   1. 重点
      * 1. 掌握栈的操作和实现
        2. 掌握队列的操作和实现
   2. 难点

1. 实现栈和队列时,要考虑线性表的操作的复杂度,以提高代码效率

1. 课后练习

练习**调用**Python3中的queue模块中的队列(Queue)类. 首先自己**建立**具有一定一个容量为16的队列实例,然后依次实现如下操作: 判断队列是否为空并输出结果(True或False),加入元素100到队列,取出队列中的元素并输出结果,依次加入200,300,400,500到队列, 判断队列是否已满(输出结果True,或False),依次取出元素(显示结果),返回队列的大小.

1. **import** queue as Q
2. q = Q.Queue(maxsize = 10)
4. isempty = q.empty()
5. #如果队列为空，返回True,反之False
6. **print**(q.empty())
8. isfull = q.full() #如果队列满了，返回True,反之False
10. q.put(100)
12. # 取出一个元素
13. a = q.get()
14. **print**(a)
16. # 在队列中加入元素
17. li = [200,300,400,500]
18. **for** i **in** li:
19. q.put(i)
21. # 依次取出元素
22. **while** **not** q.empty():
23. **print**(q.get())
25. size = q.qsize() #返回队列的大小
26. **print**(size)
27. 扩展知识或课外阅读推荐
    1. 课外阅读

刘清,张小勇,王琼 <<数据结构>>(第三版),电子工业出版社 (2010)

裘宗燕,<<数据结构与算法>>,机械工业出版社 (2015)