Queue abstraction data type & Python implementation

队列抽象数据类型及Python实现

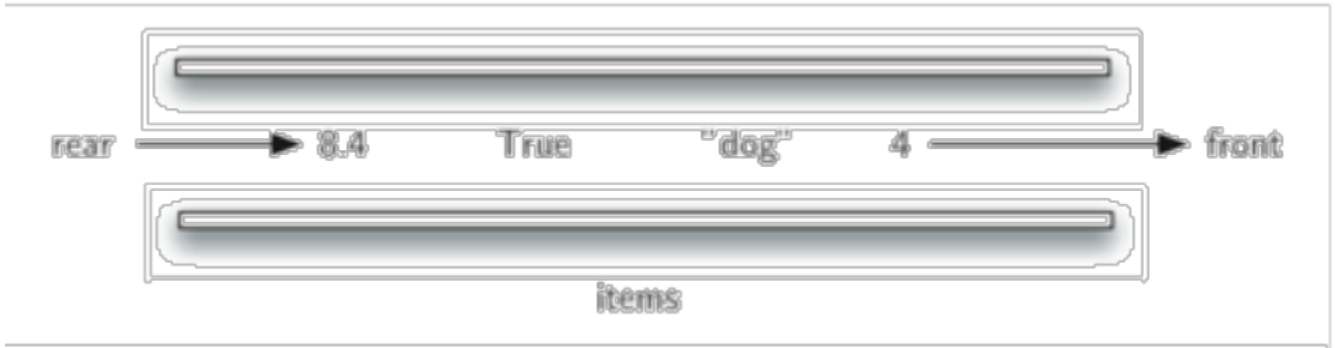
Queue: What is a queue?
队列Queue：什么是队列？

A queue is a sequential collection of data characterized by
队列是一种有次序的数据集合，其特征是

The addition of new data items always occurs at one end (usually the tail rear end)
新数据项的添加总发生在一端(通常称为“尾rear”端)

The removal of existing data items always occurs on the other end (usually the first front end)
而现存数据项的移除总发生在另一端(通常称为“首front”端)

When the data item joins the queue, it first appears at the end of the queue, and then it gradually approaches the team head with the removal of the first data item.
当数据项加入队列，首先出现在队尾，随着队首数据项的移除，它逐渐接近队首。



Queue: What is a queue?
队列Queue：什么是队列？

New data items must wait at the end of the data set, and the longest waiting data item is the queue head. The principle of this order arrangement is called (FIFO: First-in first-out) first-in-out

新加入的数据项必须在数据集末尾等待，而等待时间最长的数据项则是队首，这种次序安排的原则称为(FIFO:First-infirst-out)先进先出

Or "first come, first service first-come first-served"

或“先到先服务first-comefirst-served”

Examples of queues appear in every aspect of our daily life: queuing that has only one entrance and one exit

队列的例子出现在我们日常生活的方方面面：排队队列仅有一个入口和一个出口

Data items are not allowed to insert directly into the queue, or to remove data items from the middle

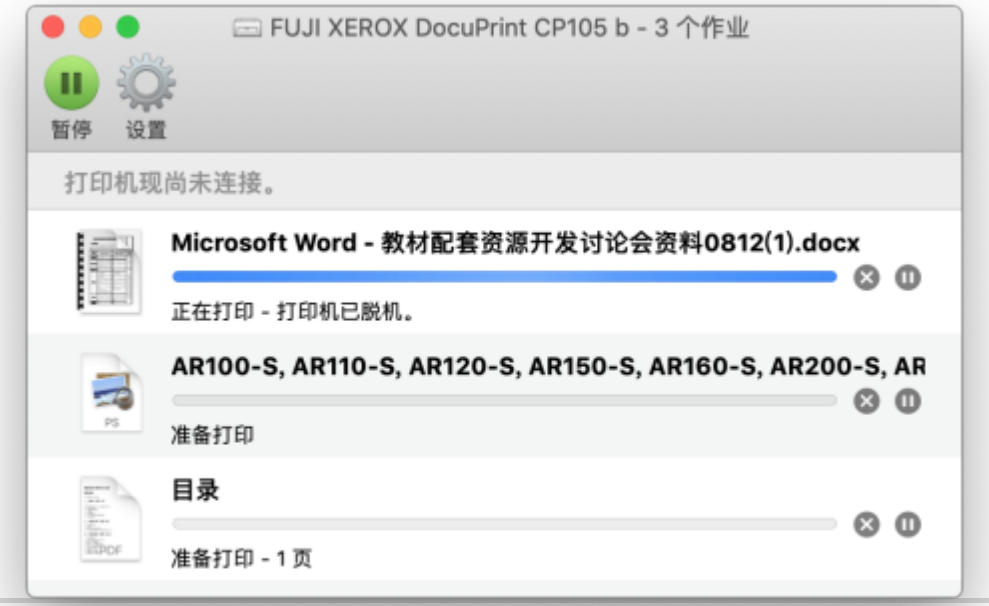
不允许数据项直接插入队中，也不允许从中间移除数据项

Example of a queue in computer science: Print a queue
计算机科学中队列的例子：打印队列

A single printer serves multiple users / programs
一台打印机面向多个用户/程序提供服务

The printing speed is much slower than the print request submission speed. When a task is printing, the later print requests are queued up, waiting to be processed as a FIFO.

打印速度比打印请求提交的速度要慢得多，有任务正在打印时，后来的打印请求就要排成队列，以FIFO的形式等待被处理。



Example of a queue in computer science:process scheduling
计算机科学中队列的例子：进程调度

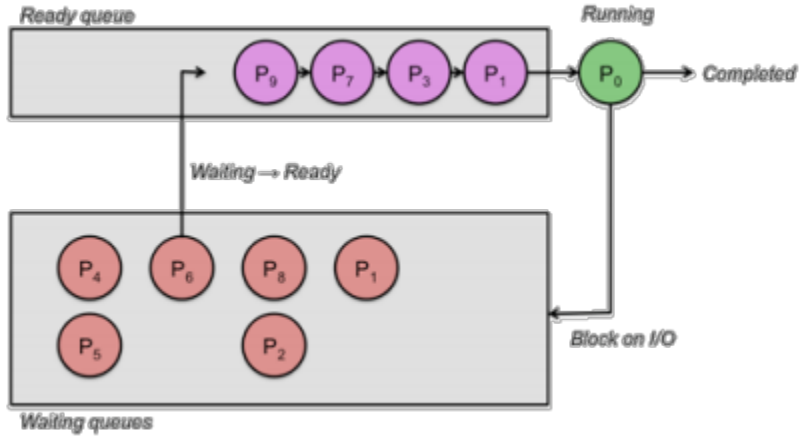
The operating system core uses multiple queues to schedule processes running simultaneously in the system

操作系统核心采用多个队列来对系统中同时运行的进程进行调度

- The number of processes is far more than the number of CPU cores
  进程数远多于CPU核心数
- Some processes still wait for different types of I/O events
  有些进程还要等待不同类型I/O事件

The scheduling principle integrates the two starting points of "first come, first come service" and "full utilization of resources"

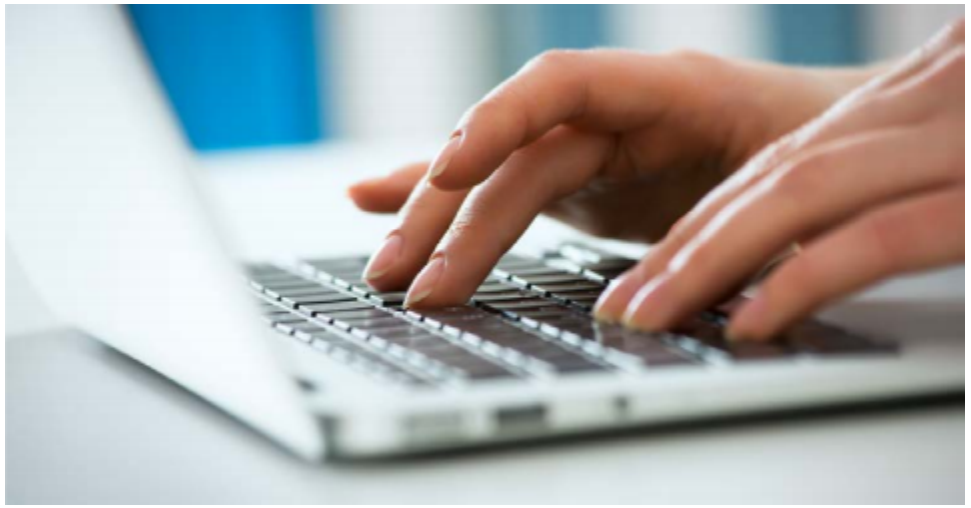
调度原则综合了“先到先服务”及“资源充分利用”两个出发点



Example of a queue in computer science:keyboard buffering
计算机科学中队列的例子：键盘缓冲

Keyboard tapping is not immediately displayed on the screen
键盘敲击并不马上显示在屏幕上

You need a queue buffer to hold tapping characters that are not yet displayed,
需要有个队列性质的缓冲区，将尚未显示的敲击字符暂存其中，
The first-in-first-out nature of the queue ensures the consistency of the input and display order of the characters.
队列的先进先出性质则保证了字符的输入和显示次序一致性。



The Abstract Datatype, Queue
抽象数据类型Queue

The abstract data type Queue is a sequential data set

抽象数据类型Queue是一个有次序的数据集合

Data items are added to the “tail rear” end only

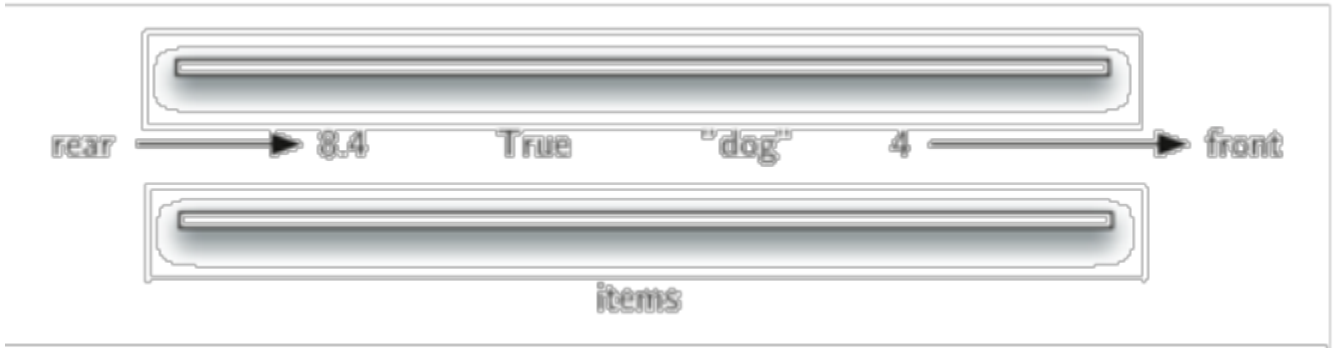
数据项仅添加到“尾rear”端

And only removed from the “first front” end

而且仅从“首front”端移除

The Queue has the order of operations for the FIFO

Queue具有FIFO的操作次序



The Abstract Datatype, Queue
抽象数据类型Queue

The abstract data type Queue is defined by the following actions:

抽象数据类型Queue由如下操作定义：

①Queue(): Create an empty queue object with a return value of the Queue object;

Queue()：创建一个空队列对象，返回值为Queue对象；

②enqueue (item): Add the data item item to the end of the team with no return value;

enqueue(item)：将数据项item添加到队尾，无返回值；

③dequeue(): Remove the data item from the queue head, the return value is the team first data item, and the queue is modified;

dequeue()：从队首移除数据项，返回值为队首数据项，队列被修改；

④isEmpty(): Test for an empty queue, returned as a Boolean value

isEmpty()：测试是否空队列，返回值为布尔值

⑤size(): Return the number of data items in the queue.

size()：返回队列中数据项的个数。

The Abstract Datatype, Queue
抽象数据类型Queue

Cohort operation 队列操作	Queue content 队列内容	returned value 返回值
q=Queue()	[]	Queueobject
q.isEmpty()	[]	True
q.enqueue(4)	[4]	
q.enqueue('dog')	['dog',4]	
q.enqueue(True)	[True,'dog',4]	
q.size()	[True,'dog',4]	3
q.isEmpty()	[True,'dog',4]	False
q.enqueue(8.4)	[8.4,True,'dog',4]	
q.dequeue()	[8.4,True,'dog']	4
q.dequeue()	[8.4,True]	'dog'
q.size()	[8.4,True]	2

The Python implements the ADT Queue

Python实现ADTQueue

The List was used to accommodate the data items of the Queue

采用List来容纳Queue的数据项

Use the List head as the queue rear

将List首端作为队列尾端

The end of the List was used as the queue head

List的末端作为队列首端

The enqueue () complexity is O(n)

enqueue()复杂度为O(n)

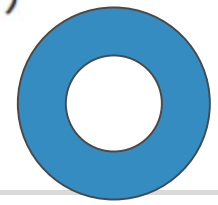
The dequeue () complexity is O(1)

dequeue()复杂度为O(1)

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()

    def size(self):
        return len(self.items)
```

The beginning and end are upside down, and the complexity is also upside down

首尾倒过来的实现，复杂度也倒过来



Application of the queue: Print task

队列的应用：打印任务

Simulation algorithm: Print the task
模拟算法：打印任务

Many people share a printer and adopt a "first come, first service" queue policy to perform printing tasks. In this setting, a primary problem is:

多人共享一台打印机，采取“先到先服务”的队列策略来执行打印任务在这种设定下，一个首要的问题就是：

How big is this printing job system in capacity?

这种打印作业系统的容量有多大？

How many users can the system accommodate and how often to submit print tasks during an acceptable waiting time?

在能够接受的等待时间内，系统能容纳多少用户以及多高频率提交打印任务？

Simulation algorithm: Print the task
模拟算法：打印任务

A specific instance configuration is given as follows:
一个具体的实例配置如下：

In a laboratory, about 10 students are present in any hour, and each person initiates about two prints, each lasting 1 to 20 pages
一个实验室，在任意的一个小时内，大约有10名学生在场，这一小时中，每人会发起2次左右的打印，每次1~20页

The printer's performance is:
打印机的性能是：

Print in draft mode, 10 pages per minute,
以草稿模式打印的话，每分钟10页，

Print in normal mode, the print quality is good, but the speed drops to 5 pages per minute.
以正常模式打印的话，打印质量好，但速度下降为每分钟5页。

Simulation algorithm: Print the task
模拟算法：打印任务

The question is: how do you set the printer mode so that everyone won't wait too long to improve the printing quality?

问题是：怎么设定打印机的模式，让大家都不会等太久的前提下尽量提高打印质量？

This is a typical decision support problem, but it cannot be calculated directly through the rules. We will use a program to simulate this printing task scenario, and then analyze the program running results to support the decision on the printer mode setting.

这是一个典型的决策支持问题，但无法通过规则直接计算，我们要用一段程序来模拟这种打印任务场景，然后对程序运行结果进行分析，以支持对打印机模式设定的决策。

How to model the problem?
如何对问题建模？

First, abstract the problem and determine the relevant objects and process

首先对问题进行抽象，确定相关的对象和过程

Abandon away the gender, age, printer model, print content, print size and other details that are unrelated to the essence of the problem

抛弃那些对问题实质没有关系的学生性别、年龄、打印机型号、打印内容、纸张大小等等众多细节

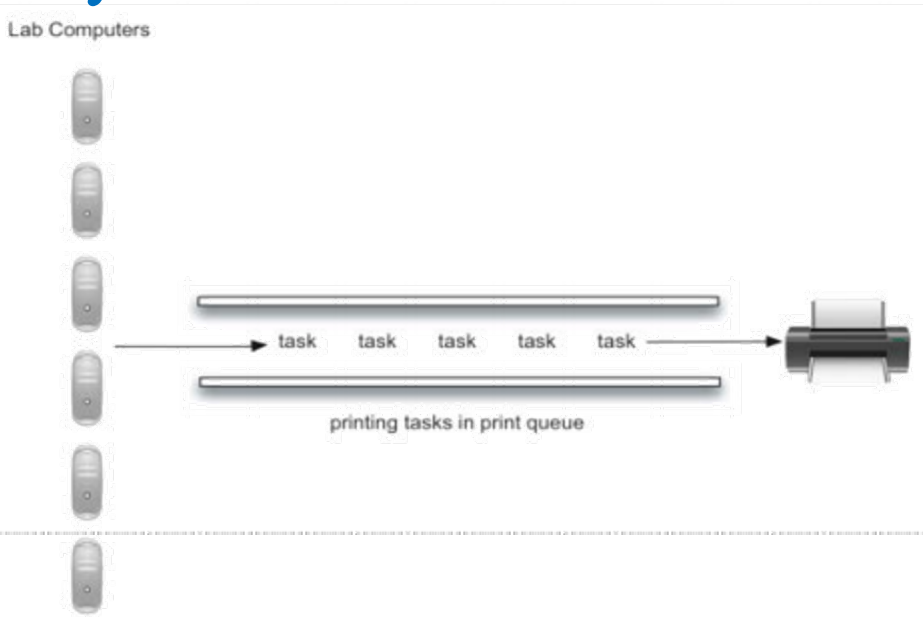
How to model the problem?
如何对问题建模？

Object: Print task, print queue, printer
对象：打印任务、打印队列、打印机

Attributes of the print task: commit time, and the number of printed pages
打印任务的属性：提交时间、打印页数

Attributes of the print queue: a print task queue with a FIFO nature
打印队列的属性：具有FIFO性质的打印任务队列

Printer attributes: print speed, whether busy or not
打印机的属性：打印速度、是否忙



How to model the problem?
如何对问题建模？

Process: Generate and submit print tasks
过程：生成和提交打印任务

Determine the generation probability: The instance is 20 assignments submitted by 10 students per hour, so that 1 job is generated and submitted every 180 seconds, with a probability of 1 / 180 per second.

确定生成概率：实例为每小时会有10个学生提交的20个作业，这样，概率是每180秒会有1个作业生成并提交，概率为每秒1/180。

Determine the number of printed pages: the instance is 1~20 pages, then the probability between 1 and 20 pages is the same.

确定打印页数：实例是1~20页，那么就是1~20页之间概率相同。

$$\frac{20\ tasks}{1\ hour} \times \frac{1\ hour}{60\ minutes} \times \frac{1\ minute}{60\ seconds} = \frac{1\ task}{180\ seconds}$$

How to model the problem?
如何对问题建模？

Process: Implement printing
过程：实施打印

Current print job: Printing job

当前的打印作业：正在打印的作业

Countdown to printing end: The countdown starts when the new task starts printing. Returning to 0 means that the printing is finished and the next task can be processed

打印结束倒计时：新作业开始打印时开始倒计时，回0表示打印完毕，可以处理下一个作业

simulated time:
模拟时间：

Unified time frame: Set the end time to synchronize all processes with the time that elapses uniformly in the smallest unit (seconds) : process the two processes of generating a print task and implementing a print once each

统一的时间框架：以最小单位(秒)均匀流逝的时间，设定结束时间同步所有过程：在一个时间单位里，对生成打印任务和实施打印两个过程各处理一次

Print task issues: Simulation process
打印任务问题：模拟流程

Create a print queue object

创建打印队列对象

Time passes in units of seconds

时间按照秒的单位流逝

- Generate the print task by probability and join the print queue

  按照概率生成打印作业，加入打印队列

- If the printer is idle and the queue is not empty, remove the first task to print it and record this task's waiting time

  如果打印机空闲，且队列不空，则取出队首作业打印，记录此作业等待时间

- If the printer is busy, print at the speed for 1 second

  如果打印机忙，则按照打印速度进行1秒打印

- If the current printing task is completed, the printer enter the idle state

  如果当前作业打印完成，则打印机进入空闲

Time runs out, and start counting the average waiting time

时间用尽，开始统计平均等待时间

Print task issues: Simulation process
打印任务问题：模拟流程

Waiting time for the task
作业的等待时间

When generating a task, record the generated timestamp

生成作业时，记录生成的时间戳

When starting printing, using the current time subtract the generation time

开始打印时，当前时间减去生成时间即可

Print time of the job
作业的打印时间

Records the number of pages for the task

生成作业时，记录作业的页数

When you start printing, divide the number of pages by the print speed

开始打印时，页数除以打印速度即可

Print task issue: Python code 1
打印任务问题：Python代码1

```
from pythonds.basic.queue import Queue

import random

class Printer:
    def __init__(self, ppm):
        self.pagerate = ppm
        self.currentTask = None
        self.timeRemaining = 0

    def tick(self):
        if self.currentTask != None:
            self.timeRemaining = self.timeRemaining - 1
            if self.timeRemaining <= 0:
                self.currentTask = None

    def busy(self):
        if self.currentTask != None:
            return True
        else:
            return False

    def startNext(self,newtask):
        self.currentTask = newtask
        self.timeRemaining = newtask.getPages() \
                             * 60/self.pagerate
```

打印速度

打印任务

任务倒计时

打印1秒

打印忙？

打印新作业

Print task issue: Python code 2

打印任务问题：Python代码2

```
class Task:
    def __init__(self,time):
        self.timestamp = time
        self.pages = random.randrange(1,21)

    def getStamp(self):
        return self.timestamp

    def getPages(self):
        return self.pages

    def waitTime(self, currenttime):
        return currenttime - self.timestamp

    def newPrintTask():
        num = random.randrange(1,181)
        if num == 180:
            return True
        else:
            return False
```

生成时间戳

打印页数

等待时间

1/180概率生成作业

Print task issue: Python code 3
打印任务问题：Python代码3

```
def simulation(numSeconds, pagesPerMinute):

    labprinter = Printer(pagesPerMinute)
    printQueue = Queue()
    waitingtimes = []

    for currentSecond in range(numSeconds):

      if newPrintTask():
         task = Task(currentSecond)
         printQueue.enqueue(task)

      if (not labprinter.busy()) and \
                (not printQueue.isEmpty()):
        nexttask = printQueue.dequeue()
        waitingtimes.append( \
            nexttask.waitTime(currentSecond))
        labprinter.startNext(nexttask)

      labprinter.tick()

    averageWait=sum(waitingtimes)/len(waitingtimes)
    print("Average Wait %6.2f secs %3d tasks remaining."\
                    %(averageWait,printQueue.size()))
```

模拟

时间流逝

Print Task Issues: Run and Analyze 1
打印任务问题：运行和分析1

Run the simulation 10 times at 5PPM and 1 hour
按5PPM、1小时的设定，模拟运行10次

The total average waiting time was 93.1 seconds, the longest average wait was 164 seconds, and the shortest average wait was 26 seconds, with 3 simulations and some tasks had not started printing

总平均等待时间93.1秒，最长的平均等待164秒，最短的平均等待26秒，有3次模拟，还有作业没开始打印

```
>>> for i in range(10):
    simulation(3600,5)


Average Wait  67.00 secs   0 tasks remaining.
Average Wait  26.00 secs   0 tasks remaining.
Average Wait  46.00 secs   2 tasks remaining.
Average Wait 115.00 secs   0 tasks remaining.
Average Wait  53.00 secs   0 tasks remaining.
Average Wait 121.00 secs   0 tasks remaining.
Average Wait 164.00 secs   1 tasks remaining.
Average Wait 136.00 secs   0 tasks remaining.
Average Wait 122.00 secs   2 tasks remaining.
Average Wait  81.00 secs   0 tasks remaining.
```

Print Task Issues: Run and Analyze 2

打印任务问题：运行和分析2

Increase the printing speed to 10PPM, 1 hour setting

提升打印速度到10PPM、1小时的设定

The total average waiting time is 12 seconds, the longest average waiting time is 35 seconds, the shortest average waiting time is 0 seconds, and moreover, all the tasks are printed

总平均等待时间12秒，最长的平均等待35秒，最短的平均等待0秒，就是一提交就打印了，而且，所有作业都打印了

```
>>> for i in range(10):
    simulation(3600,10)


Average Wait  35.00 secs   0 tasks remaining.
Average Wait   8.00 secs   0 tasks remaining.
Average Wait  29.00 secs   0 tasks remaining.
Average Wait   0.00 secs   0 tasks remaining.
Average Wait  13.00 secs   0 tasks remaining.
Average Wait   5.00 secs   0 tasks remaining.
Average Wait   0.00 secs   0 tasks remaining.
Average Wait   8.00 secs   0 tasks remaining.
Average Wait  17.00 secs   0 tasks remaining.
Average Wait   5.00 secs   0 tasks remaining.
```

Print Task Question: Discussion
打印任务问题：讨论

To make decisions on the printing mode setting, we evaluated the task waiting time with a simulation program

为了对打印模式设置进行决策，我们用模拟程序来评估任务等待时间

Through the analysis of the simulation results in two cases, we realize that

通过两种情况模拟仿真结果的分析，我们认识到

If so many students rush to class with the printed program source code

如果有那么多学生要拿着打印好的程序源代码赶去上课的话

Then, the printing quality must be sacrificed to improve the printing speed.

那么，必须得牺牲打印质量，提高打印速度。

Using simulation system to simultate the reality

模拟系统对现实的仿真

Sometimes atuthentic experiments can not be carried out without consuming real resources. It can help us make decisions using different settings and repeated simulations.

在不耗费现实资源的情况下—有时候真实的实验是无法进行的，可以以不同的设定，反复多次模拟来帮助我们进行决策。

Print Task Question: Discussion
打印任务问题：讨论

Print task simulation program can also be added to different settings to make a richer simulation

打印任务模拟程序还可以加进不同设定，来进行更丰富的模拟

- What happens when the number of students doubles?

  学生数量加倍了会怎么样？

- What if students don't have to rush to class on weekends and accept longer waiting times?

  如果在周末，学生不需要赶去上课，能接受更长等待时间，会怎么样？

- What if you switch to Python programming, greatly reducing the source code and the number of pages you print?

  如果改用Python编程，源代码大大减少，打印的页数减少了，会怎么样？

Print Task Question: Discussion
打印任务问题：讨论

More realistic simulations, derived from more refined modeling of the problem, and from setting and running with real data, can also be extended to other similar decision support problems

更真实的模拟，来源于对问题的更精细建模，以及以真实数据进行设定和运行，也可以扩展到其它类似决策支持问题

For example: the table setting of the restaurant makes the customer queuing time shorter

如：饭馆的餐桌设置，使得顾客排队时间变短

Application of the queue: Hot potatoes
队列的应用：热土豆

Hot potato problem (Josephus problem)
热土豆问题(约瑟夫问题)

The potato version of "Drum pass flowers"
“击鼓传花”的土豆版本

Pass hot potatoes, when the drums stop, the children with potatoes will stand out of the queue.

传烫手的热土豆，鼓声停的时候，手里有土豆的小孩就要出队列。

Hot potato problem (Joseph problem)

热土豆问题(约瑟夫问题)

If the drums were removed and the fixed number of people was passed through, it became the "modern" Joseph problem

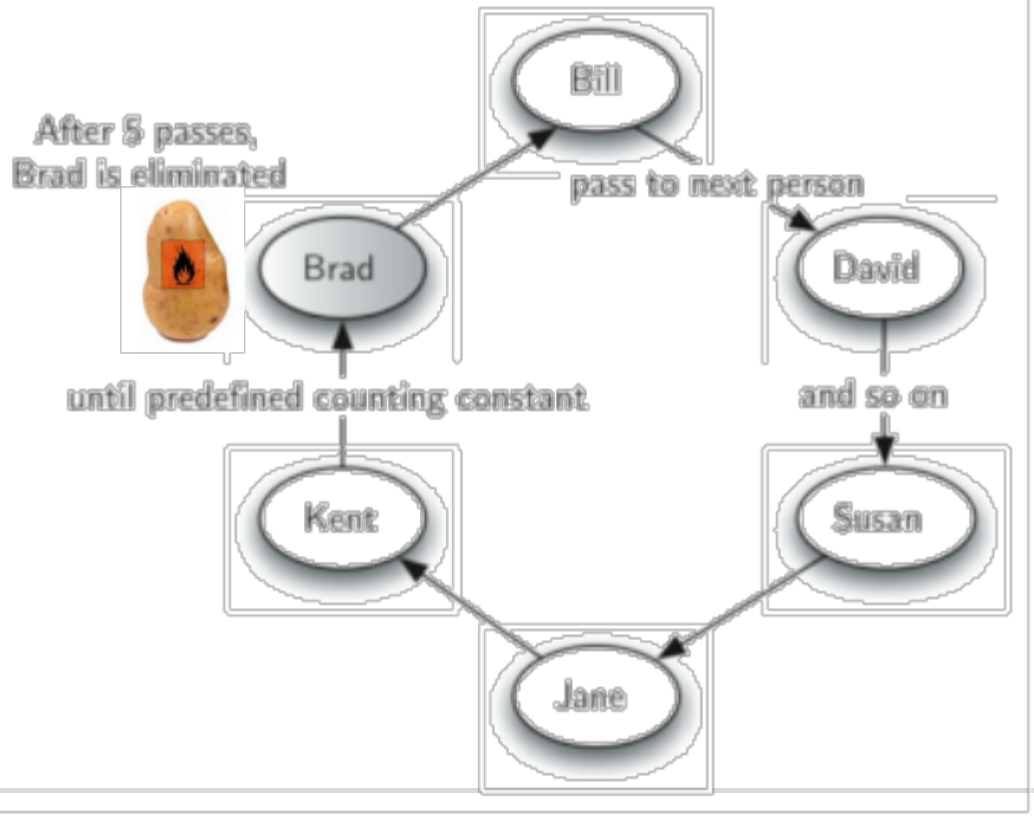
如果去掉鼓，改为传过固定人数，就成了“现代版”的约瑟夫问题

著名犹太历史学家Josephus有过以下的故事：在罗马人占领乔塔帕特后，39 个犹太人与Josephus及他的朋友躲到一个洞中，39个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式，41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止。然而Josephus 和他的朋友并不想遵从。首先从一个人开始，越过k-2个人（因为第一个人已经被越过），并杀掉第k个人。接着，再越过k-1个人，并杀掉第k个人。这个过程沿着圆圈一直进行，直到最终只剩下一个人留下，这个人就可以继续活着。问题是，给定了和，一开始要站在什么地方才能避免被处决。Josephus要他的朋友先假装遵从，他将朋友与自己安排在第16个与第31个位置，于是逃过了这场死亡游戏。

Hot potato problem : the algorithm
热土豆问题：算法

An Algorithm for the Hot Potato Problem using Queues: the list of names participating in the game, the number of potato transmission num, the algorithm returns the last remaining names

用队列来实现热土豆问题的算法：参加游戏的人名列表，以及传土豆次数num，算法返回最后剩下的人名



Hot potato problem : the algorithm
热土豆问题：算法

The simulation program uses queues to store the names of all the participants, following the direction of passing potatoes from the front to the back of the queue

模拟程序采用队列来存放所有参加游戏的人名，按照传递土豆方向从队首排到队尾

In the game, the first team is always holding potatoes

游戏时，队首始终是持有土豆的人

At the beginning of the simulation game, only need to leave the queue who is at the head of the queue, and then join the queue at the end of the queue, which is a transfer of potatoes.

模拟游戏开始，只需要将队首的人出队，随即再到队尾入队，算是土豆的一次传递

After passing the num times, remove the people at the queue head, and no longer join the team.

Repeat this, until there is only 1 remaining person in the queue

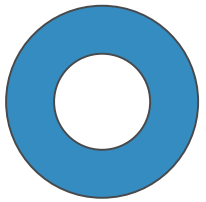
传递了num次后，将队首的人移除，不再入队，如此反复，直到队列中剩余1人

Hot potato problem : Code
热土豆问题：代码

```
from pythonds.basic.queue import Queue

def hotPotato(namelist, num):
    simqueue = Queue()
    for name in namelist:
        simqueue.enqueue(name)

    while simqueue.size() > 1:
        for i in range(num):
            simqueue.enqueue(simqueue.dequeue())

        simqueue.dequeue()

    return simqueue.dequeue()

print(hotPotato(["Bill","David","Susan","Jane","Kent","Brad"],7))
```

A pass

一次传递



Two-end queue abstraction data type and Python implementation

双端队列抽象数据类型及Python实现

Dual-end queue (Deque): What is Deque?
双端队列Deque：什么是Deque

The two-end queue Deque is an ordered dataset,

双端队列Deque是一种有次序的数据集，

Similar to queues, both ends can be called "front" and "rear"

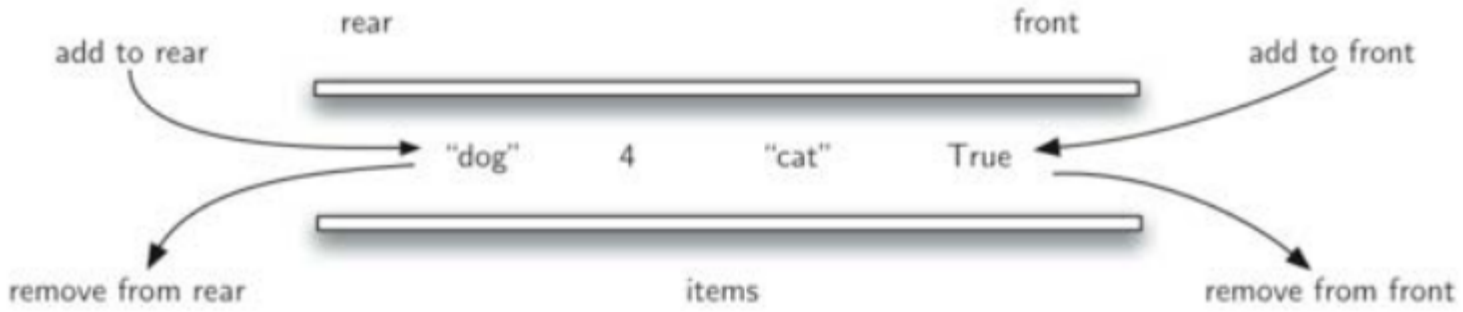
跟队列相似，其两端可以称作“首”“尾”端，

However, data items in deque can be added either from the head or from the end; data items can also be removed from both ends.

但deque中数据项既可以从队首加入，也可以从队尾加入；数据项也可以从两端移除。

In a sense, two-end queues integrate the capabilities of the stack and the queue

某种意义上说，双端队列集成了栈和队列的能力



Dual-end queue Deque: What is Deque?

双端队列Deque：什么是Deque

However, the two-end queue does not have intrinsic LIFO or FIFO features

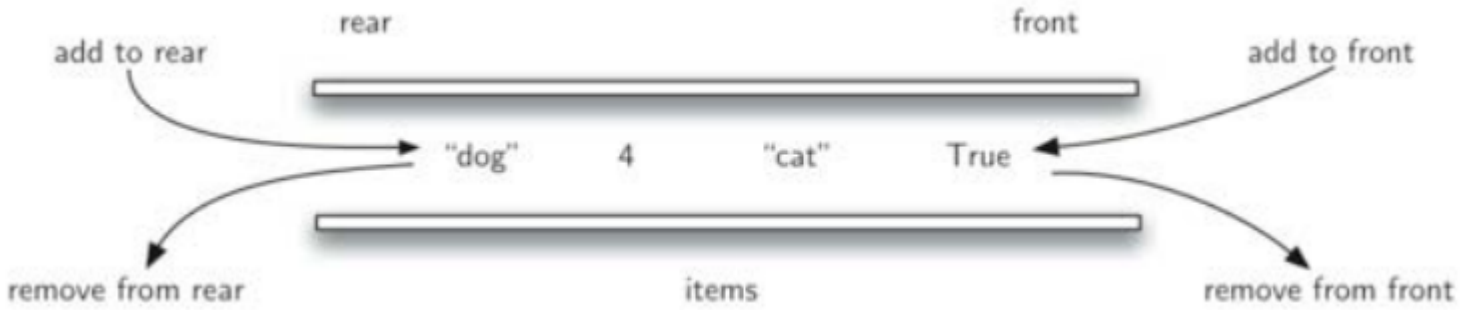
但双端队列并不具有内在的LIFO或者FIFO特性

If a two-end queue is used to simulate the stack or the queue

如果用双端队列来模拟栈或队列

The consistency of the operation needs to be maintained by the user himself

需要由使用者自行维护操作的一致性



The Abstract Datatype Deque

抽象数据类型Deque

The deque operation is defined as follows:

deque定义的操作如下：

①Deque (): Create an empty two-end queue

Deque()：创建一个空双端队列

②addFront (item): Add item to the queue head

addFront(item)：将item加入队首

③addRear (item): Add item to the queue end

addRear(item)：将item加入队尾

④removeFront (): Remove the data item from the queue head, and the return value is the removed data item

removeFront()：从队首移除数据项，返回值为移除的数据项

⑤removeRear (): Remove the data item from the end of the queue, and the return value is the removed data item

removeRear()：从队尾移除数据项，返回值为移除的数据项

⑥isEmpty (): Return whether the deque is empty

isEmpty()：返回deque是否为空

⑦size (): Return the number of data items included in the deque

size()：返回deque中包含数据项的个数

The Abstract Datatype Deque

抽象数据类型Deque

Dual-end queue operation 双端队列操作	Dual-end queue content 双端队列内容	returned value 返回值
d=Deque()	[]	Dequeobject
d.isEmpty()	[]	True
d.addRear(4)	[4]	
d.addRear('dog')	['dog',4,]	
d.addFront('cat')	['dog',4,'cat']	
d.addFront(True)	['dog',4,'cat',True]	
d.size()	['dog',4,'cat',True]	4
d.isEmpty()	['dog',4,'cat',True]	False
d.addRear(8.4)	[8.4,'dog',4,'cat',True]	
d.removeFront()	['dog',4,'cat']	True

The Python implements the ADT Deque
Python实现ADTDeque

implementation by List
采用List实现

- List subscript 0 is used as the tail end of the deque; List subscript -1 is used as the head of the deque
  List下标0作为deque的尾端List下标-1作为deque的首端

Operational complexity
操作复杂度

- addFront/removeFrontO(1)
- addRear/removeRearO(n)

```
class Deque:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def addFront(self, item):
        self.items.append(item)

    def addRear(self, item):
        self.items.insert(0,item)

    def removeFront(self):
        return self.items.pop()

    def removeRear(self):
        return self.items.pop(0)

    def size(self):
        return len(self.items)
```

"Palindrome word" judgment

“回文词”判定

"Palindromic word" refers to words with both the same forward and reverse reading

“回文词”指正读和反读都一样的词

Such as radar, madam, toot

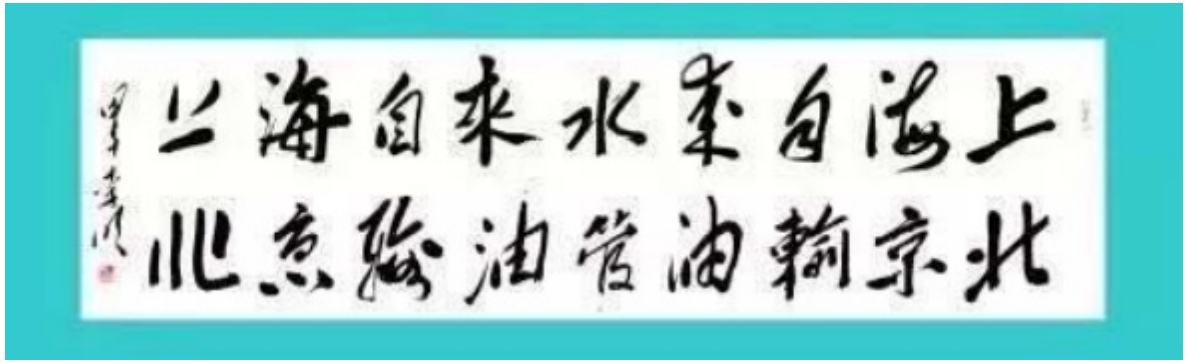
如radar、madam、toot

The Chinese "上海自来水来自海上"

中文“上海自来水来自海上”

"山东落花生花落东山"

“山东落花生花落东山”



"Palindrome word" judgment

“回文词”判定

The "palindromic" problem is easily solved with two-ended queues

用双端队列很容易解决“回文词”问题

First add the word that needs to be judged to the deque from the end of the queue

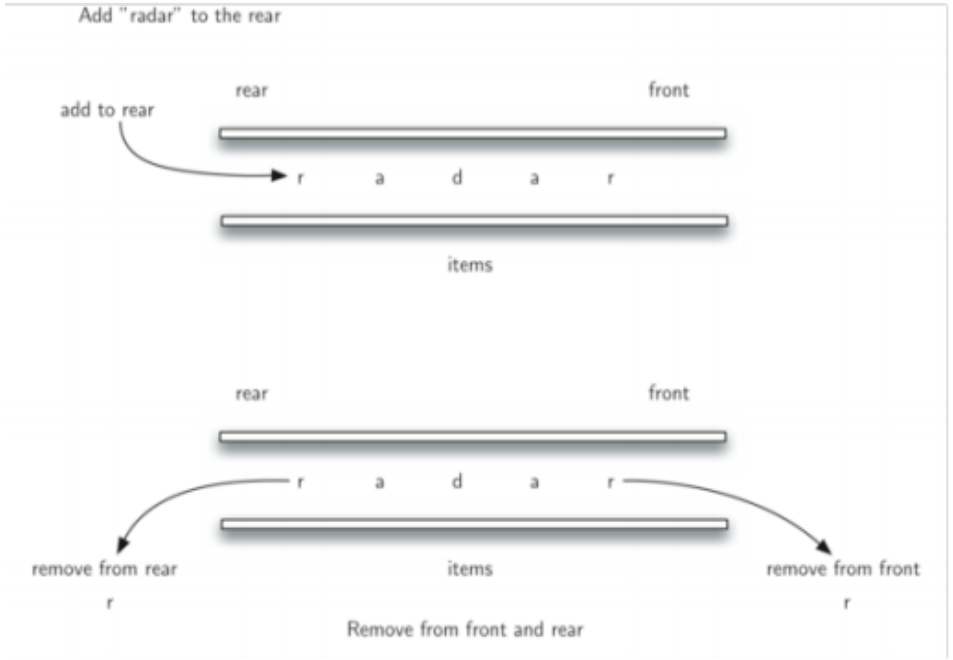
先将需要判定的词从队尾加入deque

Then remove the characters from both ends to determine whether they are the same or not,

再从两端同时移除字符判定是否相同，

Until either 0 or 1 character is left in the deque

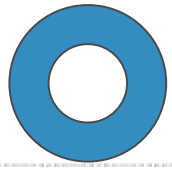
直到deque中剩下0个或1个字符



"Palindrome word" judgment : code

“回文词”判定:代码

```
from pythonds.basic.deque import Deque

def palchecker(aString):
    chardeque = Deque()

    for ch in aString:
        chardeque.addRear(ch)

    stillEqual = True

    while chardeque.size() > 1 and stillEqual:
        first = chardeque.removeFront()
        last = chardeque.removeRear()
        if first != last:
            stillEqual = False

    return stillEqual

print(palchecker("lsdkjfskf"))
print(palchecker("radar"))
```



List: Unordered list to abstract data types and Python implementation

无序表抽象数据类型及Python实现

List : What is a list?

列表List：什么是列表？

In the previous discussion of basic data structures, we adopted PythonList to implement multiple linear data structures, and List is a simple yet powerful dataset structure that provides rich operational interfaces

在前面基本数据结构的讨论中，我们采用PythonList来实现了多种线性数据结构，列表List是一种简单强大的数据集结构，提供了丰富的操作接口

But not all programming languages provide List data types, sometimes requiring programmers to implement them themselves.

但并不是所有的编程语言都提供了List数据类型，有时候需要程序员自己实现。

List : What is a list?
列表List：什么是列表？

A dataset where data items are stored in relative locations

一种数据项按照相对位置存放的数据集

In particular, it is known as the "unordered list"

特别的，被称为“无序表unorderedlist”

The data items are only indexed according to the storage location, such as number 1 and number 2 ……the last one, etc.

其中数据项只按照存放位置来索引，如第1个、第2个……、最后一个等。

(For simplicity, assume no duplicates in the table)

(为了简单起见，假设表中不存在重复数据项)

Like a collection of test scores "54,26,93,17,77, and 31"

如一个考试分数的集合“54,26,93,17,77和31”

If represented by an unordered table, it is [54,26,93,17,77,31]

如果用无序表来表示，就是[54,26,93,17,77,31]

Abstract data type: the unordered table, List

抽象数据类型：无序表List

The operation of the unordered table List is as follows:

无序表List的操作如下：

①List(): Create an empty list

List()：创建一个空列表

②Add(item): Add a data item to the list, assuming that item does not originally exist in the list

add(item)：添加一个数据项到列表中，假设item原先不存在于列表中

③remove(item): Remove the item from the list, the list is modified, and the item should originally exist in the table

remove(item)：从列表中移除item，列表被修改，item原先应存在于表中

④search(item): Find the item in the list and return a Boolean type value

search(item)：在列表中查找item，返回布尔类型值

⑤isEmpty(): Whether the return list is empty or not

isEmpty()：返回列表是否为空

⑥The size(): How many data items the return list contains

size()：返回列表包含了多少数据项

Abstract data type: the unordered table, List

抽象数据类型：无序表List

The operation of the unordered table List is as follows:

无序表List的操作如下：

⑦append (item): Add a data item to the end of the table, assuming that the item did not originally exist in the list

append(item)：添加一个数据项到表末尾，假设item原先不存在于列表中

⑧index (item): Returns the position of data items in the table

index(item)：返回数据项在表中的位置

⑨insert (pos, item): Inserts data items into the position pos, assuming that item does not originally exist in the list, and the original list has enough data items to allow item to occupy the position pos

insert(pos,item)：将数据项插入到位置pos，假设item原先不存在与列表中，同时原列表具有足够多个数据项，能让item占据位置pos

⑩The pop (): Remove data items from the end of the list, assuming that the original list has at least one data item

pop()：从列表末尾移除数据项，假设原列表至少有1个数据项

The pop (pos): Remove the data item with pos and assuming that location pos exists in the original list

pop(pos)：移除位置为pos的数据项，假设原列表存在位置pos

Unordered table and linked list implementation

无序表的链表实现

Implement unordered tables using linked list
采用链表实现无序表

To realize the disordered table data structure, the scheme of linked lists can be adopted.
为了实现无序表数据结构，可以采用链接表的方案。

Although the list data structure requires maintaining the relative position of the front and rear data items, this retention of the front and rear positions does not require the data items to be stored in a continuous storage space in turn
虽然列表数据结构要求保持数据项的前后相对位置，但这种前后位置的保持，并不要求数据项依次存放在连续的存储空间

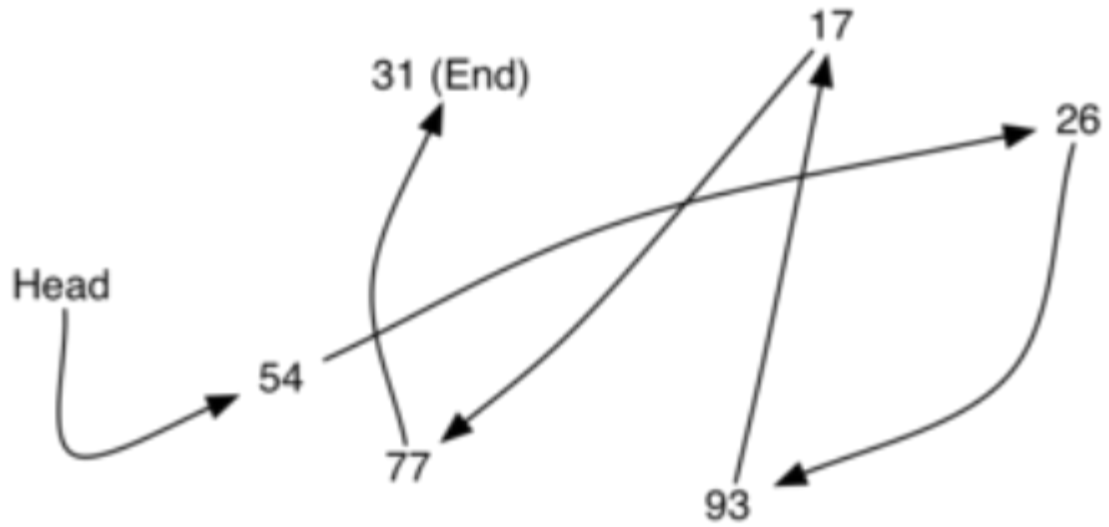
Implement unordered tables using linked list
采用链表实现无序表

As shown in the figure below, there is no rule for the location of the data items, but if a link pointing is established between the data items, you can maintain the relative position

如下图，数据项存放位置并没有规则，但如果在数据项之间建立链接指向，就可以保持其前后相对位置

The first and last data items need to be explicitly marked out, one is the head and the other is the end, and there is no data behind them.

第一个和最后一个数据项需要显式标记出来，一个是队首，一个是队尾，后面再无数据了。



Linked list implementation: Node

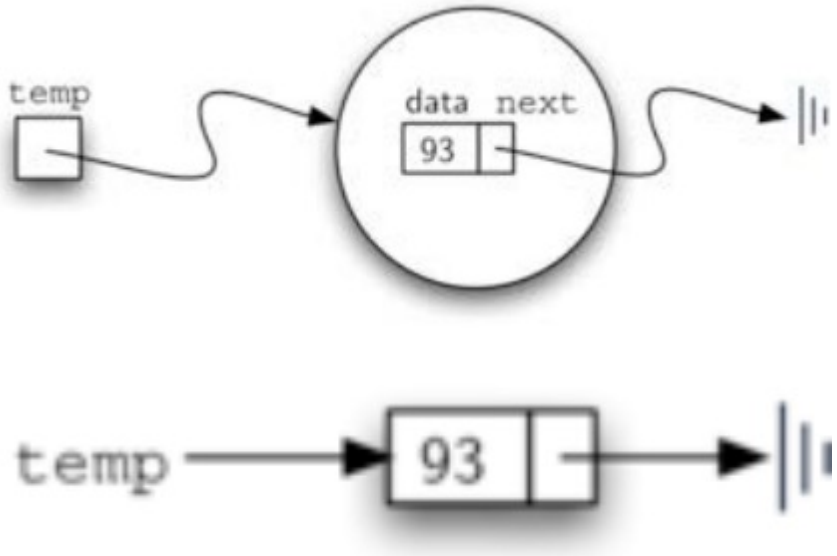
链表实现：节点Node

The most basic element of the linked list implementation is the node

链表实现的最基本元素是节点Node

Each node should contain at least two pieces of information: the data item itself, and the reference information pointing to the next node. Note that next is a None means that there is no node behind, which is of great significance

每个节点至少要包含2个信息：数据项本身，及指向下一个节点的引用信息，注意next为None的意义是没有下一个节点了，这个很重要



Linked list implementation: Node

链表实现：节点Node

```
class Node:
    def __init__(self,initdata):
        self.data = initdata
        self.next = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self,newdata):
        self.data = newdata

    def setNext(self,newnext):
        self.next = newnext
```

```
>>> temp= Node(93)
>>> temp.getData()
93
>>>
```

Linked list implementation: Unordered table

链表实现：无序表UnorderedList

Datasets can be built by linking nodes to implement disordered tables

可以采用链接节点的方式实现无序表

The first and last nodes of the linked list are the most important

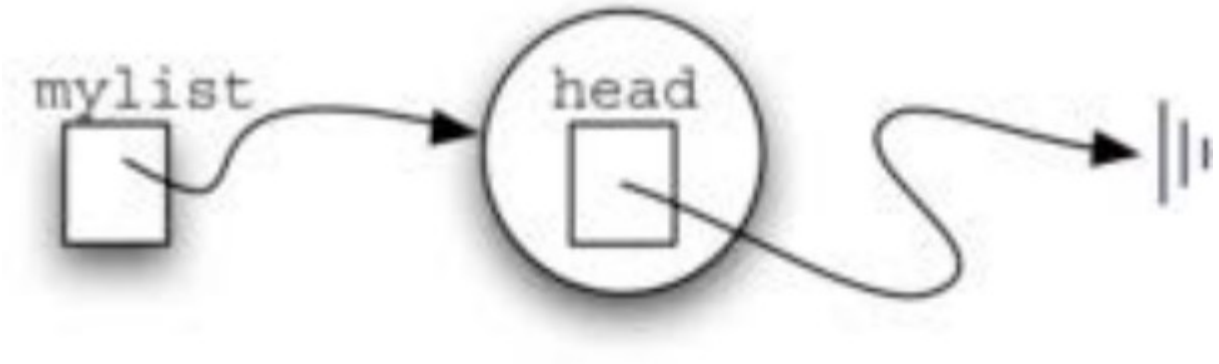
链表的第一个和最后一个节点最重要

If you want to access all the nodes in the linked table, you must go from the first

如果想访问到链表中的所有节点，就必须从第一

The nodes start going through the link

个节点开始沿着链接遍历下去



Linked list implementation: Unordered table UnorderedList

链表实现：无序表UnorderedList

So the unordered table must have a reference to the first node

所以无序表必须要有对第一个节点的引用信息

Set up a attributive head that saves the reference to the first node

设立一个属性head，保存对第一个节点的引用

The head of the empty table is None

空表的head为None

```
class UnorderedList:

    def __init__(self):
        self.head = None

>>> mylist= UnorderedList()
>>> print mylist.head
None
```

Linked list implementation: Unordered table UnorderedList

链表实现：无序表UnorderedList

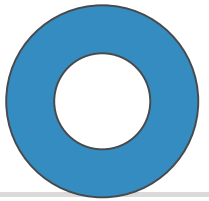
With the addition of the data items, the head of the unordered table always points to the first node in the chain

随着数据项的加入，无序表的head始终指向链条中的第一个节点

pay attention! The unordered table mylist object itself does not contain any data items (the data items are in the node). The head contained is only a reference to the first Node, and isEmpty() for judging an empty table is easy to implement

注意！无序表mylist对象本身并不包含数据项(数据项在节点中)，其中包含的head只是对首个节点Node的引用,判断空表的isEmpty()很容易实现

- returnself.head==None



linked list implementation: Unordered table UnorderedList

链表实现：无序表UnorderedList

Next, consider how to implement adding data items to the unordered table and implement the add method. Since the unordered table does not limit the order between the data items, the new data items can be added to any location in the original table, which should be added to the easiest position.

接下来，考虑如何实现向无序表中添加数据项，实现add方法。由于无序表并没有限定数据项之间的顺序，新数据项可以加入到原表的任何位置，按照实现的性能考虑，应添加到最容易加入的位置上。

Linked list implementation: Unordered table UnorderedList

链表实现：无序表UnorderedList

By the structure of linked list, we know that to access all the data items on the whole chain, you must start from the table header and search along the next link one by one. So the most convenient position to add new data items is the table header, the first position of the whole linked list.

由链表结构我们知道，要访问到整条链上的所有数据项，都必须从表头head开始沿着next链接逐个向后查找，所以添加新数据项最快捷的位置是表头，整个链表的首位置。

Linked list implementation: Unordered table UnorderedList

链表实现：无序表UnorderedList

add method

add 方法

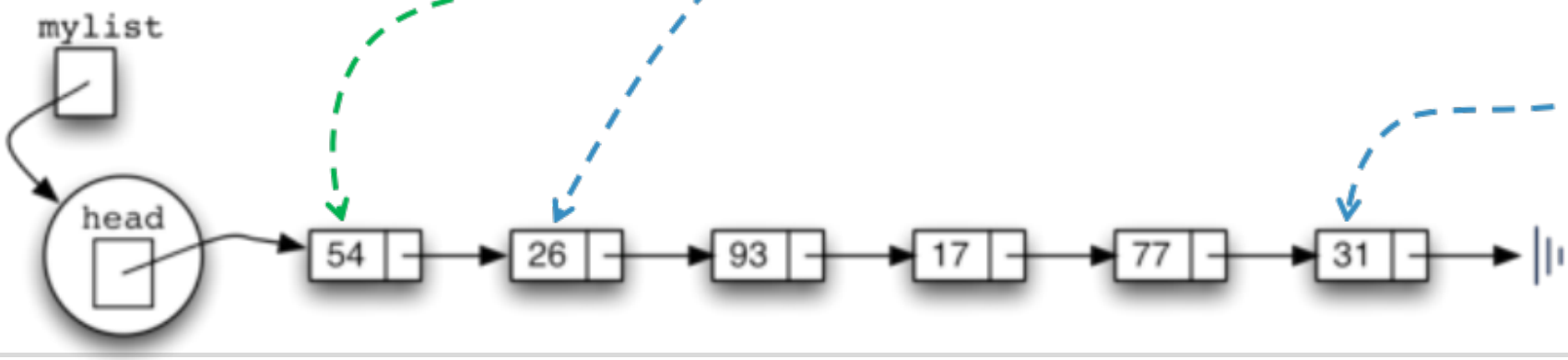
Following the code call in the right figure, the formed linked list is shown below

按照右图的代码调用，形成的链表如下图

31 is the first data item added, so it becomes the last item in the linked list, while 54 is the last added, becoming the first data item in the linked list

31是最先被加入的数据项，所以成为链表中最后一个项，而54是最后被加入的，是链表第一个数据项

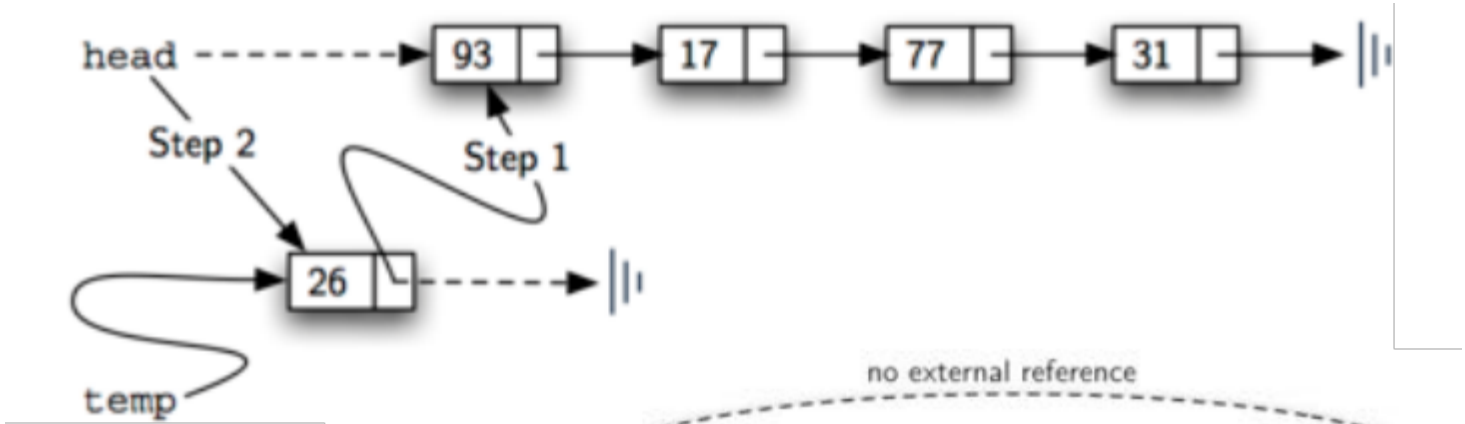
```
mylist.add(31)
mylist.add(77)
mylist.add(17)
mylist.add(93)
mylist.add(26)
mylist.add(54)
```

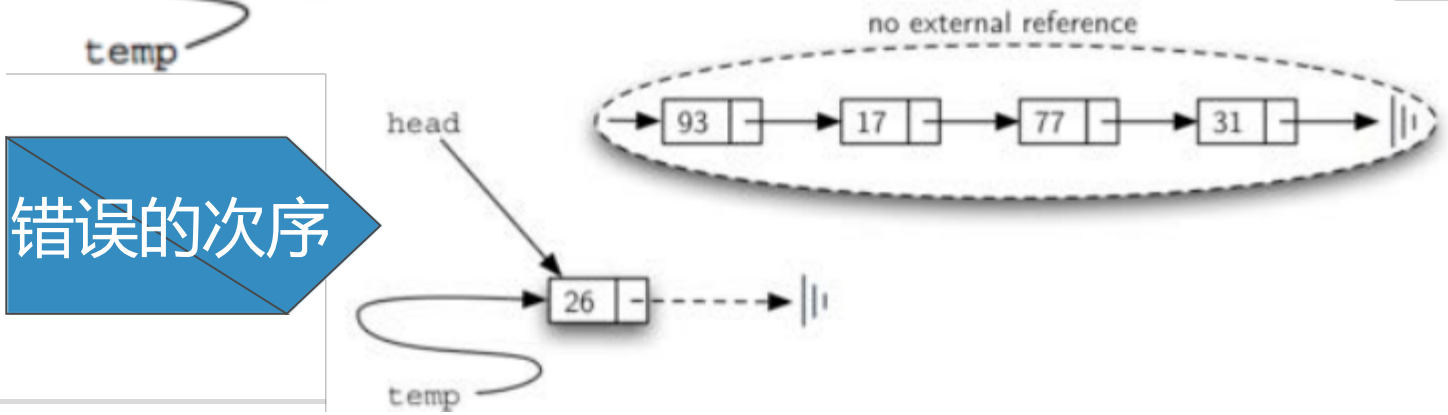


Linked list implementation: the add method implementation
链表实现：add方法实现

Linking order is very important!
Linking order is very important!
链接次序很重要！

```
def add(self,item):
    temp = Node(item)
    temp.setNext(self.head)
    self.head = temp
```

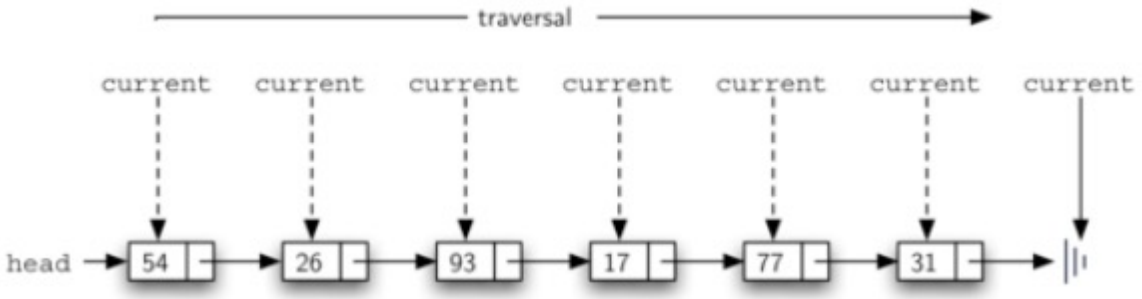




Linked list implementation: size
链表实现：size

size: traverse from the head to the end of the list and use variables to accumulate the number of nodes passed through

size：从链条头head开始遍历到表尾同时用变量累加经过的节点个数。

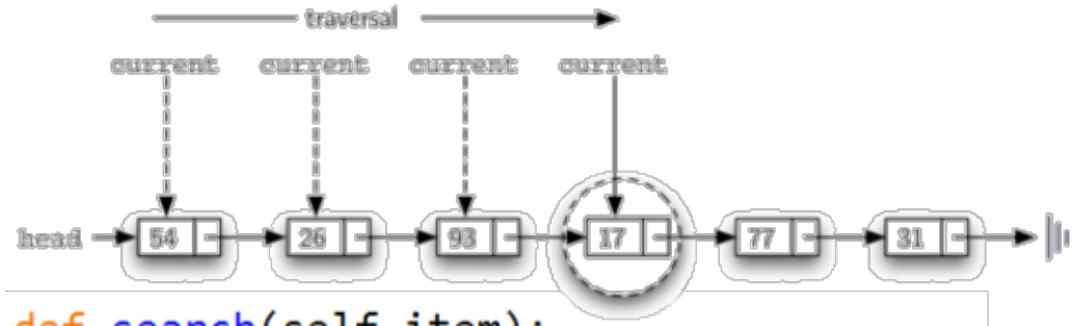



```
def size(self):
    current = self.head
    count = 0
    while current != None:
        count = count + 1
        current = current.getNext()

    return count
```

Linked list implementation: search
链表实现：search

Traverse from the head to the end of the linked list and determine whether the data item of the current node is our target

从链表头head开始遍历到表尾，同时判断当前节点的数据项是否目标




```
def search(self,item):
    current = self.head
    found = False
    while current != None and not found:
        if current.getData() == item:
            found = True
        else:
            current = current.getNext()

    return found
```

Linked list implementation: remove (item) method
链表实现：remove(item)方法

First find item, which is the same as search, but requires special tricks when deleting nodes

首先要找到item，这个过程跟search一样，但在删除节点时，需要特别的技巧

The *current* points to the node that currently matches the data item

current指向的是当前匹配数据项的节点

While the deletion requires pointing the next of the **previous node** to the **next node** of the current

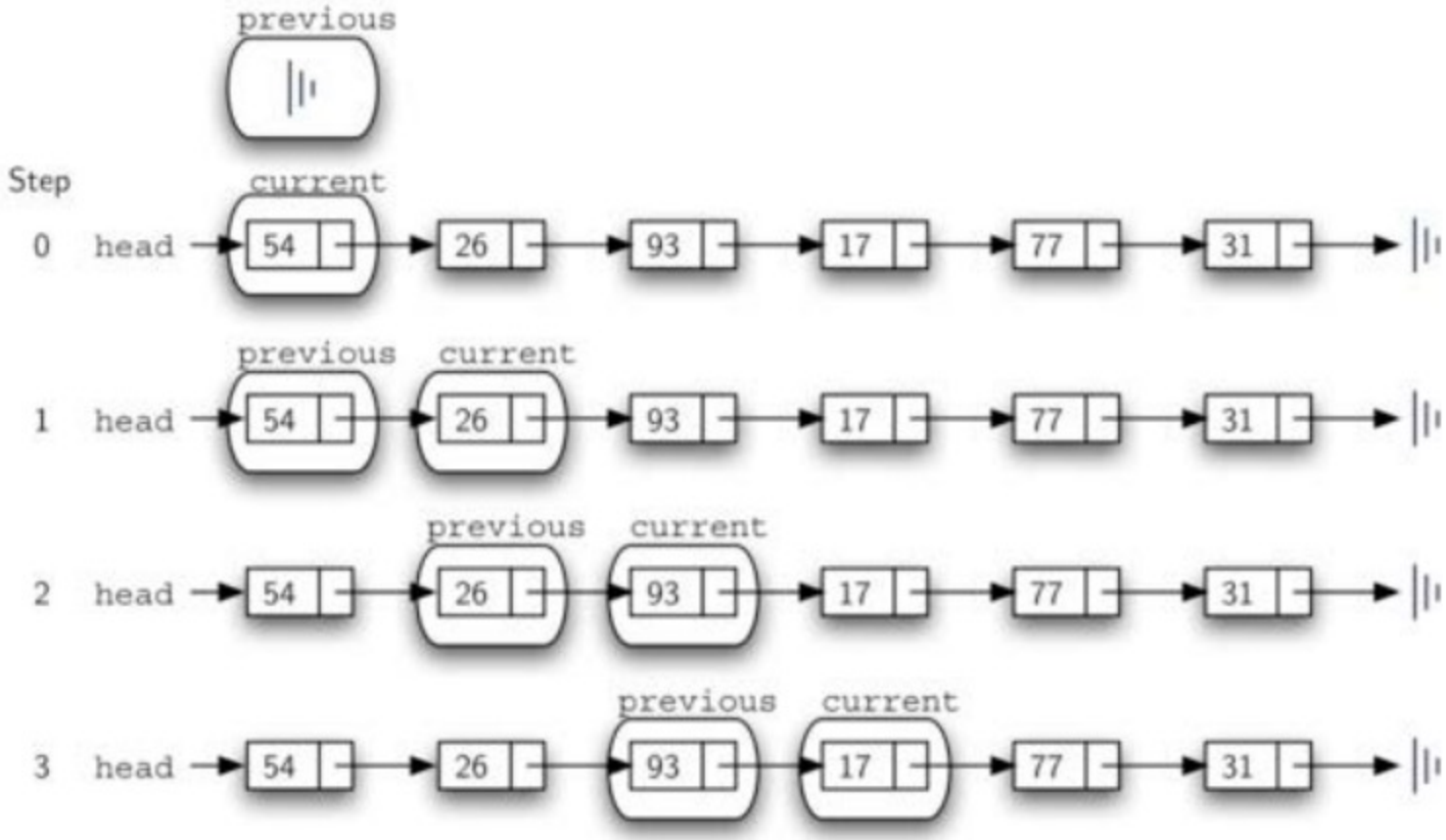
而删除需要把前一个节点的next指向current的下一个节点

So we also maintain references to the (previous) node while search current

所以我们在search current的同时，还要维护前一个(previous)节点的引用

Linked list implementation: remove (item) method

链表实现：remove(item)方法



Linked list implementation: remove (item) method
链表实现：remove(item)方法

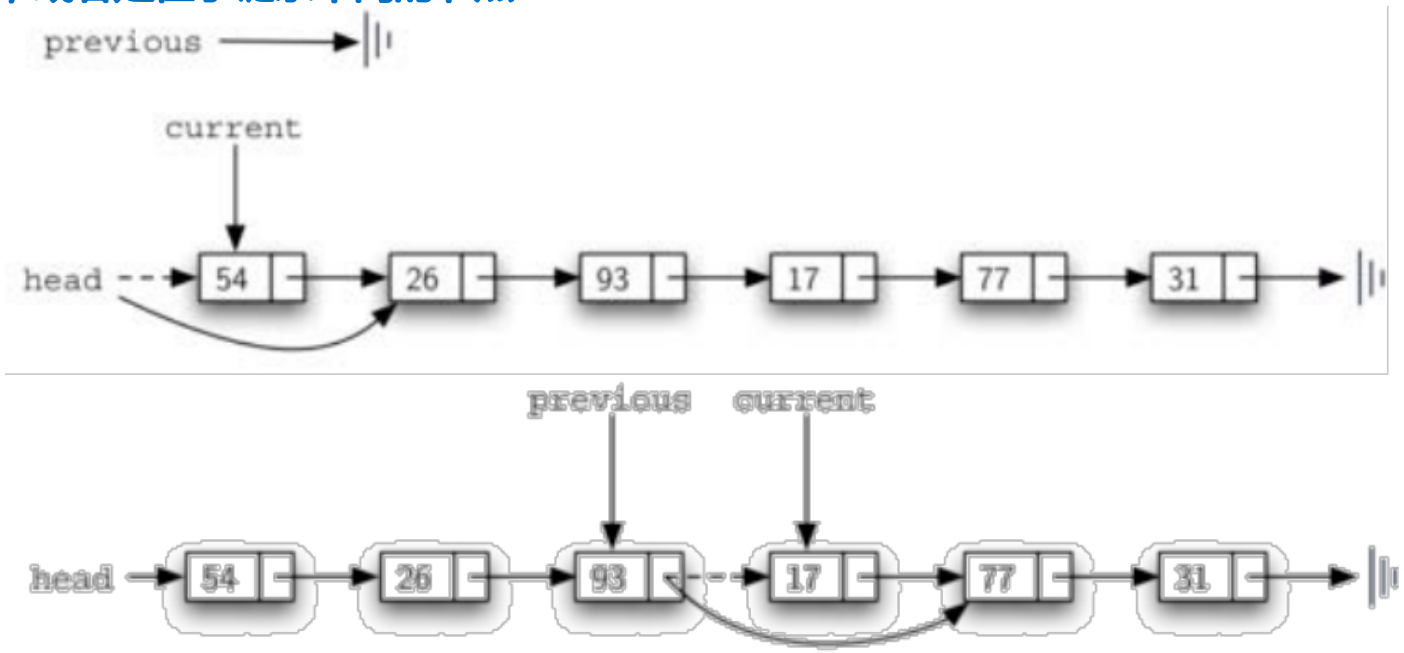
After the item is found, *current* points to the item node, and *previous* points to the previous node, then starts performing the deletion. We need distinguishing between the two cases:

找到item之后，current指向item节点，previous指向前一个节点，开始执行删除，需要区分两种情况：

①The *current* is the first node;

②or a node located in the middle of the chain

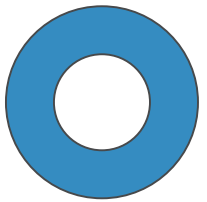
current是首个节点；或者是位于链条中间的节点



Linked list implementation: remove (item) code

链表实现：remove(item)代码

```
def remove(self,item):
    current = self.head
    previous = None
    found = False
    while not found:
        if current.getData() == item:
            found = True
        else:
            previous = current
            current = current.getNext()

    if previous == None:
        self.head = current.getNext()
    else:
        previous.setNext(current.getNext())
```



Order table to abstract data types and Python implementation

有序表抽象数据类型及Python实现

Abstract data type: ordered table OrderedList

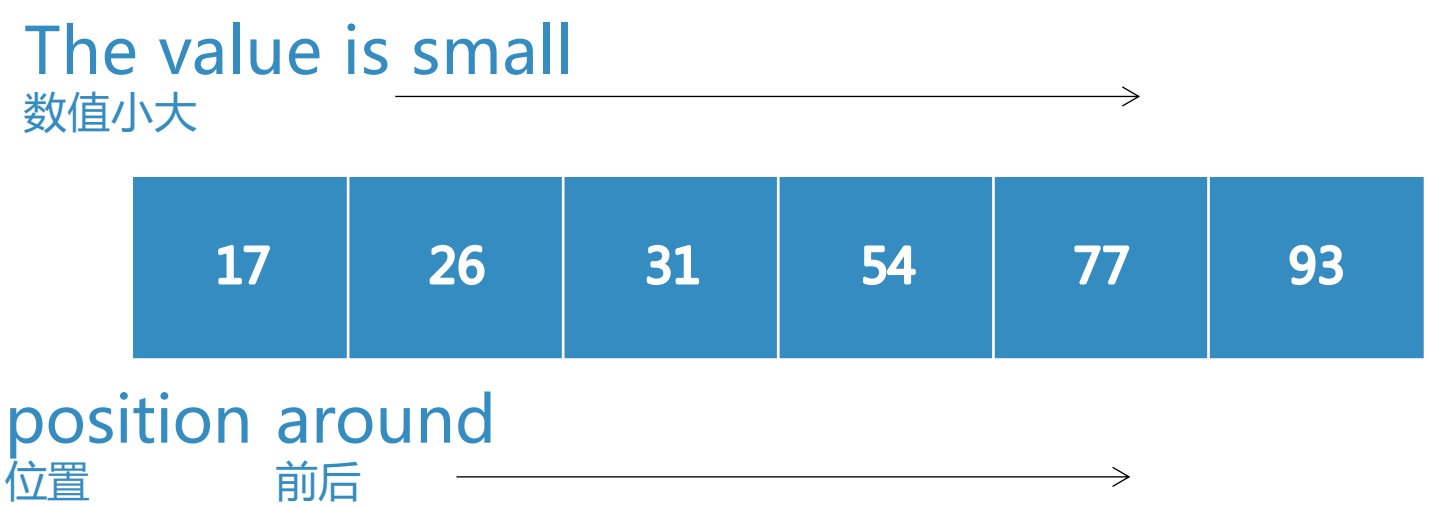
抽象数据类型：有序表OrderedList

An ordered table is a data item that is located in a list according to its comparable attribute (e. g., integer size, alphabet order)

有序表是一种数据项依照其某可比性质(如整数大小、字母表先后)来决定在列表中的位置

The "smaller" data item is, the closer it is to the head of the list

越“小”的数据项越靠近列表的头，越靠“前”



Abstract data type: ordered table OrderedList

抽象数据类型：有序表OrderedList

The actions as defined by the OrderedList are as follows:

OrderedList所定义的操作如下：

①OrderedList(): Create an empty, ordered table

`OrderedList()`：创建一个空的有序表

②Add(item): Add a data item to the table and keep the overall order.

This item does not exist before

`add(item)`：在表中添加一个数据项，并保持整体顺序，此项原不存在

③remove (item): Remove a data item from the ordered table which has already been present.

Order table is modified

`remove(item)`：从有序表中移除一个数据项，此项应存在，有序表被修改

④search (item): Find data items in the ordered table to return the presence

`search(item)`：在有序表中查找数据项，返回是否存在

⑤isEmpty (): Empty table or not

`isEmpty()`：是否空表

Abstract data type: ordered table OrderedList

抽象数据类型：有序表OrderedList

The actions as defined by the OrderedList are as follows:

OrderedList所定义的操作如下：

⑥size(): returns the number of data items in the table

size()：返回表中数据项的个数

⑦index(item): Return the location of the data item in the table that should exist

index(item)：返回数据项在表中的位置，此项应存在

⑧pop(): Remove and return the last item in the ordered table with at least one item present

pop()：移除并返回有序表中最后一项，表中应至少存在一项

⑨pop(pos): Remove and return data items for the specified location in the ordered table,which should exist

pop(pos)：移除并返回有序表中指定位置的数据项，此位置应存在

Ordered table: OrderedList implementation
有序表:OrderedList实现

When implementing the ordered tables, it is vital to keep in mind that the relative location of the data items depends on the "size" comparison between them

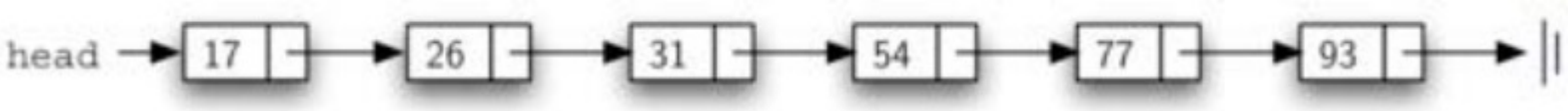
在实现有序表的时候，需要记住的是，数据项的相对位置，取决于它们之间的“大小”比较

Due to the scalability of the Python, the following discussion of data items applies not only to integers, but also to all data types that define the __gt__ method (i.e., the '>' operator)

由于Python的扩展性，下面对数据项的讨论并不仅适用于整数，可适用于所有定义了__gt__方法(即'>'操作符)的数据类型

Taking the integer data items as an example, (17,26,31,54,77,93) form is shown in Fig

以整数数据项为例，(17,26,31,54,77,93)的链表形式如图



Ordered table OrderedList implementation
有序表OrderedList实现

Also implemented using the linked list method
同样采用链表方法实现

The Node definition is the same
Node定义相同

OrderedList also sets a head to save the reference to the linked table header
OrderedList设也置一个head来保存链表表头的引用

```
class OrderedList:
    def __init__(self):
        self.head = None
```

Ordered table OrderedList implementation
有序表OrderedList实现

For the isEmpty / size / remove methods, they are independent of the order of the nodes, so the implementation is the same as for the UnorderedList, wheras search / add method requires modifications

对于isEmpty/ size/ remove这些方法，与节点的次序无关，所以其实现跟UnorderedList是一样的。search/ add方法则需要有修改

Ordered table implementation: the search method
有序表实现：search方法

In the search of the unordered table, if the data item that to be found does not exist, the entire linked list will be searched until the end of it

在无序表的search中，如果需要查找的数据项不存在，则会搜遍整个链表，直到表尾

For the ordered tables, we can take advantage of the orderly arrangement of the linked table nodes to save the search time without the data items

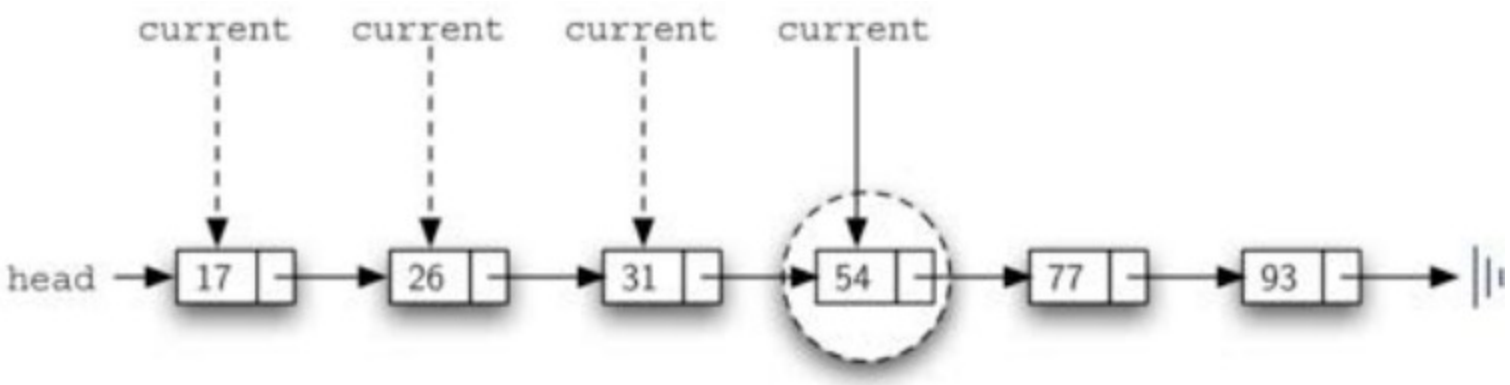
对于有序表来说，可以利用链表节点有序排列的特性，来为search节省不存在数据项的查找时间

Once the data item of the current node is bigger than the data item to be searched, it is impossible to have the data item to be found after the linked list, so it can be returned directly to False

一旦当前节点的数据项大于所要查找的数据项，则说明链表后面已经不可能再有要查找的数据项，可以直接返回False

Ordered table implementation: the search method
有序表实现：search方法

If we want to search the data item of 54 in the figure below
如我们要在下图查找数据项54



Ordered table implementation: the search method
有序表实现：search方法

```
def search(self,item):
    current = self.head
    found = False
    stop = False
    while current != None and not found and not stop:
        if current.getData() == item:
            found = True
        else:
            if current.getData() > item:
                stop = True
            else:
                current = current.getNext()

    return found
```

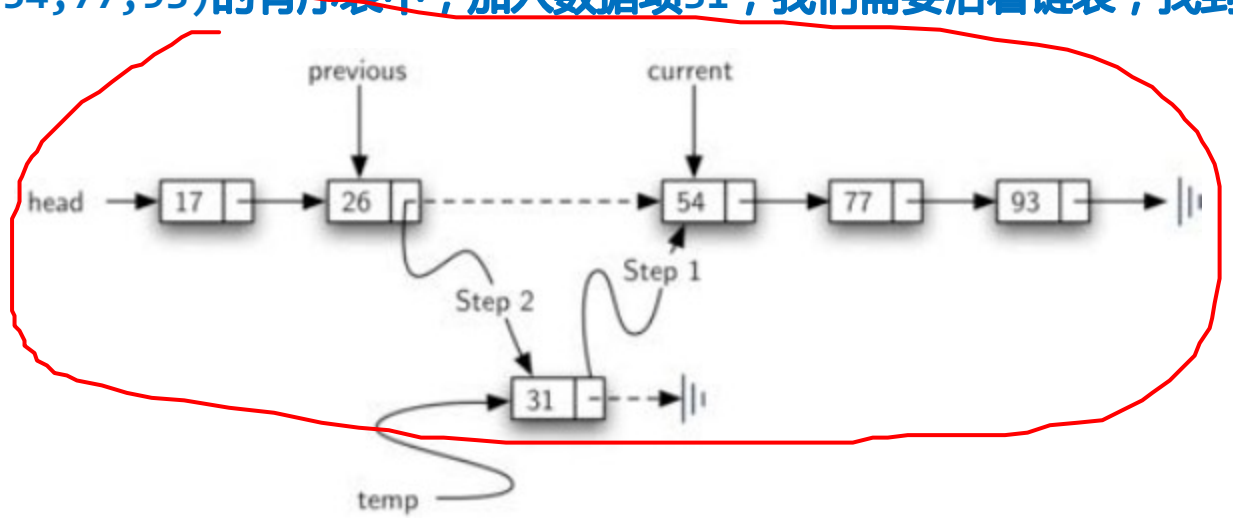
Ordered table implementation: the add method
有序表实现：add方法

Compared to the unordered tables, the biggest way to change is the add, because the add method must ensure that the added data items are added in the appropriate position to maintain the order of the whole linked list.

相比无序表，改变最大的方法是add，因为add方法必须保证加入的数据项添加在合适的位置，以维护整个链表的有序性

For example, in the ordered table of (17,26,54,77,93), to add the data item 31, we need to follow the linked list, find the first data item 54 which is larger than 31, and insert 31 into the front of 54

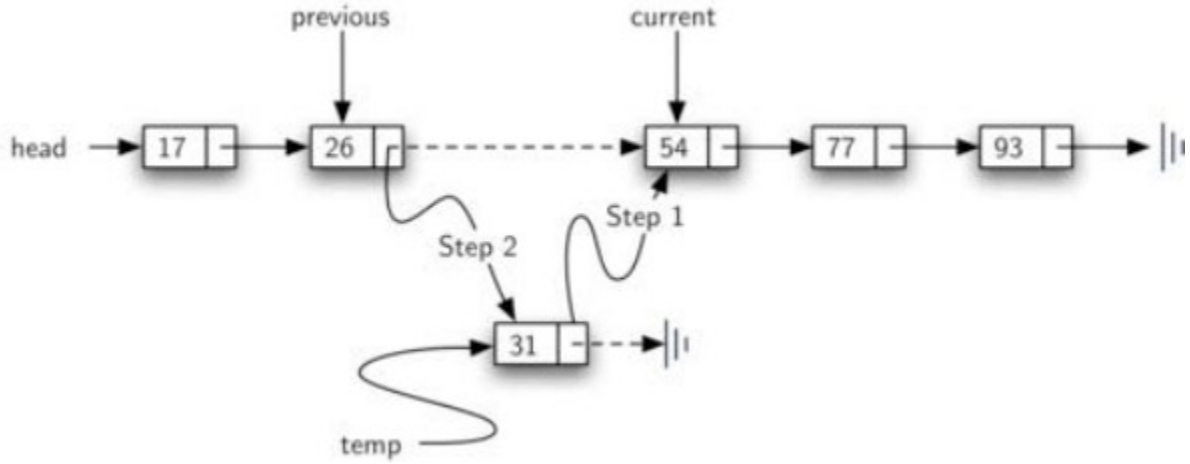
比如在(17,26,54,77,93)的有序表中，加入数据项31，我们需要沿着链表，找到第一个比31大的数据项54，将31插入到54的前面



Ordered table implementation: the add method
有序表实现：add方法

Since the insertion location involved is before the current node, and the linked list cannot get the "front drive" node reference, so similar to the remove method, introduce a previous reference, follow the current node *current*. Once the first data item larger than 31 is found, *previous* comes in handy

由于涉及到的插入位置是当前节点之前，而链表无法得到“前驱”节点的引用，所以要跟remove方法类似，引入一个previous的引用，跟随当前节点current，一旦找到首个比31大的数据项，previous就派上用场了



Ordered table for the OrderedList implementation: the add method
有序表OrderedList实现：add方法

```
def add(self,item):
    current = self.head
    previous = None
    stop = False
    while current != None and not stop:
        if current.getData() > item:
            stop = True
        else:
            previous = current
            current = current.getNext()

    temp = Node(item)
    if previous == None:
        temp.setNext(self.head)
        self.head = temp
    else:
        temp.setNext(current)
        previous.setNext(temp)
```

Found the plug, into the location
发现插入位置

Insert in the header
插入在表头

Insert in table
插入在表中

Algorithmic analysis of the linked list implementation
链表实现的算法分析

For the analysis of the linked list complexity, mainly to see whether the corresponding method involves the traversal of the linked list

对于链表复杂度的分析，主要是看相应的方法是否涉及到链表的遍历

For a linked table containing nodes of n

对于一个包含节点数为n的链表

isEmpty is O (1) because only you need to check whether head is None

isEmpty是O(1)，因为仅需要检查head是否为None

The size is O (n) because there is no way to know the number of nodes except traverse to the end of the linked list

因为除了遍历到表尾，没有其它办法得知节点的数量

The search / remove and the add method of the ordered table is O(n), because it involves the traverse of the linked table. According to the probability, the average number of operations is n/2

search/remove以及有序表的add方法，则是O(n)，因为涉及到链表的遍历，按照概率其平均操作的次数是n/2

The add method for an unordered table is O (1), because it only needs to be inserted into the table header

无序表的add方法是O(1)，因为仅需要插入到表头

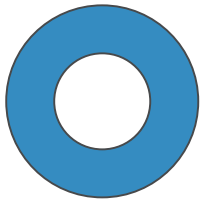
Algorithmic analysis of the linked list implementation
链表实现的算法分析

The List of the linked list implemented is different from the time complexity of the list data type built into the Python in some of the same methods

链表实现的List，跟Python内置的列表数据类型，在有些相同方法的实现上的时间复杂度不同

Mainly because the built-in list data types in Python are implemented based on sequential storage and are optimized

主要是因为Python内置的列表数据类型是基于顺序存储来实现的，并进行了优化



Linear structure summary
线性结构小结

Linear structure summary
线性结构小结

Linear data structure Linear DS organizes the data items in some linear order

线性数据结构LinearDS将数据项以某种线性的次序组织起来

Stack maintains the order of LIFO(last in, first out)

栈Stack维持了数据项后进先出LIFO的次序

The basic operations of stack include push, pop, isEmpty

stack的基本操作包括push,pop,isEmpty

Queue maintains the order of FIFO(first in, first out)

队列Queue维持了数据项先进先出FIFO的次序

The basic operations of queue include enqueue, dequeue, and isEmpty

queue的基本操作包括enqueue,dequeue,isEmpty

Linear structure summary
线性结构小结

Three methods of writing expression are prefix, infix and suffix

书写表达式的方法有前缀prefix、中缀infix和 后 缀 suffix三 种

Because the stack structure has the attribute of order reversal, so the stack structure is suitable

由于栈结构具有次序反转的特性，所以栈结构适

Developing the algorithms for expression evaluation and transformation

合用于开发表达式求值和转换的算法

The "simulation system" can be used through an abstract modeling of a real-world problem, and add random numbers to dynamicly operate, providing a reference for the decisions of complex problems

“模拟系统”可以通过一个对现实世界问题进行抽象建模，并加入随机数动态运行，为复杂问题的决策提供各种情况的参考

Queue can be used for the development of simulation systems

队列queue可以用来进行模拟系统的开发

Linear structure summary
线性结构小结

Dual-end Deque can have both the capabilities of stack and queue

双端队列Deque可以同时具备栈和队列的功能

The main operations of deque include addFront, addRear, removeFront, removeRear, isEmpty

deque的主要操作包括addFront,addRear,removeFront,removeRear,isEmpty

The List is a dataset in which a data item maintains a relative position

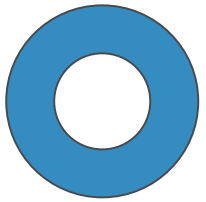
列表List是数据项能够维持相对位置的数据集

The implementation of the linked list can maintains the characteristics of relative position without the need for continuous storage space

链表的实现，可以保持列表维持相对位置的特点，而不需要连续的存储空间

When implemented, its various methods for linked list head require special processing

链表实现时，其各种方法，对链表头部head需要特别的处理



Discuss

1）In a chain queue, assuming that the queue head pointer is ***front***, the queue tail pointer is ***rear***, and the element pointed to by ***x*** needs to be queued, the operation to be performed is ( ).

在一个链队列中，假设队头指针为front，队尾指针为rear，x所指向的元素需要入队，则需要执行的操作为（ ）。

A.`front=x,fron=fron->next`

B.`x->next=front->next,front=x`

C.`rear->next=x,rear=x`

D.`rear->next=x,x->next=null,rear=x`

插入操作时，先将结点x插入到链表尾部，再让rear指向这个结点x。C的做法不够严密，因为是队尾，所以队尾x->next必须置为空。

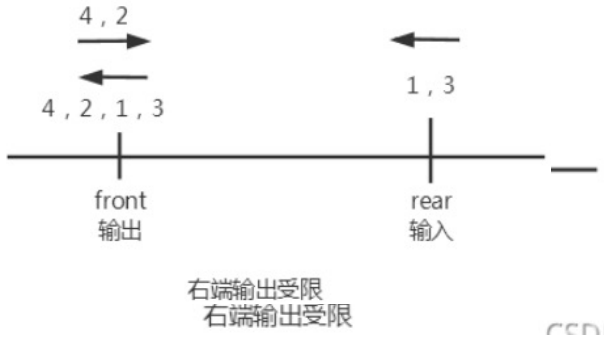
2）If 1, 2, 3 and 4 are taken as the input seque
not be obtained from the double ended queue
output is ( ).

若以1,2,3,4作为双端队列的输入序列，则既不能由
输出序列是（ ）。

A.1，2，3，4　　B.4，1，3，2　　C.4，2，3，1　　D.4，2，1，3

使用排除法。先看可由输入受限的双端队列产生的序列：设右端输入受限，1,2,3,4依次左入，则依次左出可得4,3,2,1，排除A；右出、左出、右出、右出可得到4,1,3,2，排除B；再看可由输出受限的双端队列产生的序列：设右端输出受限，1,2,3,4依次左入、左入、右入、左入，依次左出可得到4,2,1,3，排除D，故选C。

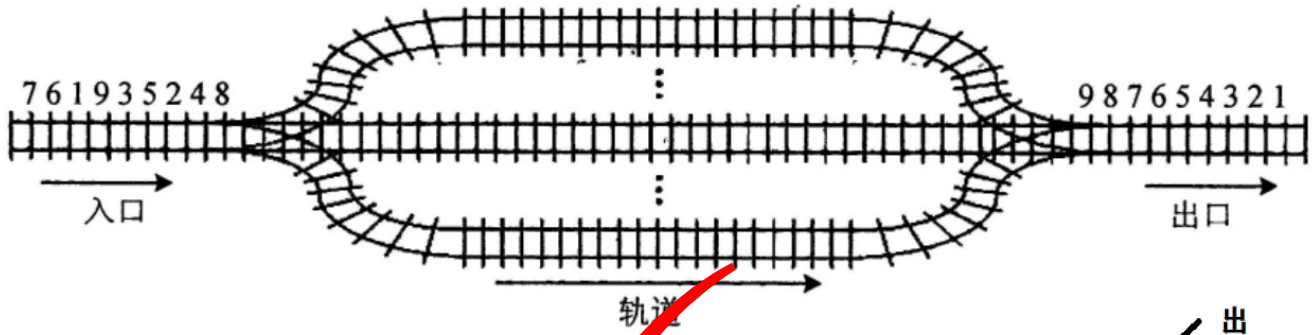
选项B：



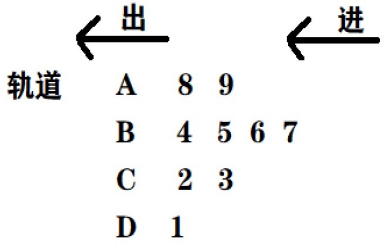
Discuss

3）There are train tracks as shown in the figure below. There are ***N*** tracks between the entrance and the exit. The train travels from left to right. The train can enter any track. There are 9 trains numbered 1-9. The sequence of entry is 8, 4, 2, 5, 3, 9, 1, 6 and 7. If the order of expected driving out is 1 ~ 9, then n is at least (　).

设有如下图所示的火车车轨，入口到出口之间有n条轨道，列车的行进方向均为从左至右，列车可驶入任意一条轨道。现有编号为1~9的9列列车，驶入的次序依次是8,4,2,5,3,9,1,6,7。若期望驶出的次序依次为1~9，则n至少是（　）。



A.2　　B.3　　C.4　　D.5



根据题目从入口到出口有多个队列（轨道），且每个队列（轨道）可容纳多个元素（列车），若需实现出队顺序为1~9，则需先入队元素小于后入队元素

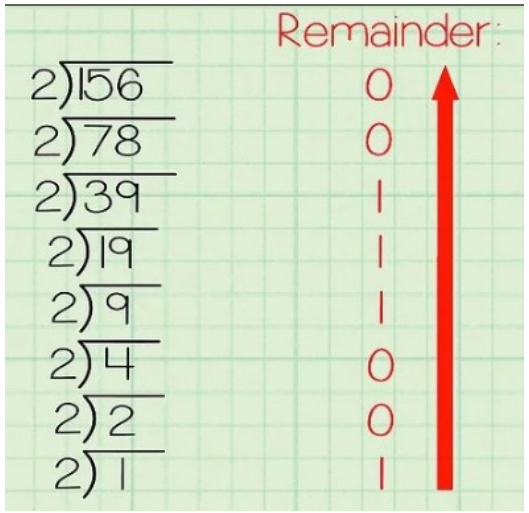
Discuss

1）Use the divide by 2 algorithm to convert the following values to binary numbers. Lists the remainder of the conversion process.

使用“除以2”算法将下列值转换成二进制数。列出转换过程中的余数。

- 17
- 45
- 96

参考：



Discuss

make some noise

2）Read the definition of the circular queue and answer the relevant questions:

The sequential queue is assumed to be a circular space, that is, the table storing the queue elements is logically regarded as a ring, which is called a circular queue. When the team leader pointer ***Q.front = maxsize-1***, it will automatically reach 0 when it advances one position. This can be achieved by dividing and taking the remainder (%).

阅读循环队列的定义并做相关讨论：

将顺序队列臆造为一个环状的空间，即把存储队列元素的表从逻辑上视为一个环，称为循环队列。当队首指针`Q.front=MaxSize-1`后，再前进一个位置就自动到0，这可以利用除法取余运算(%)来实现。

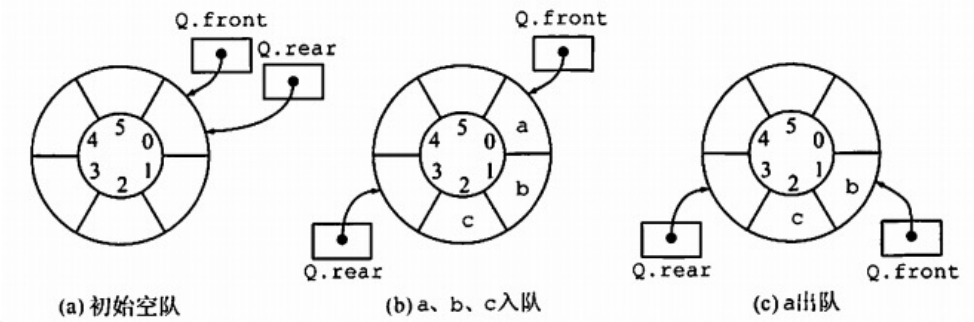
初始时：`Q.front=Q.rear=0`

队首指针进1：`Q.front=(Q.front+1)%MaxSize`

队尾指针进1：`Q.rear=(Q.rear+1)%MaxSize`

队列长度：`(Q.rear+MaxSize-Q.front)%MaxSize`

出队入队时：指针都按顺时针方向进1 When leaving the team and entering the team: the pointer moves clockwise



Discuss

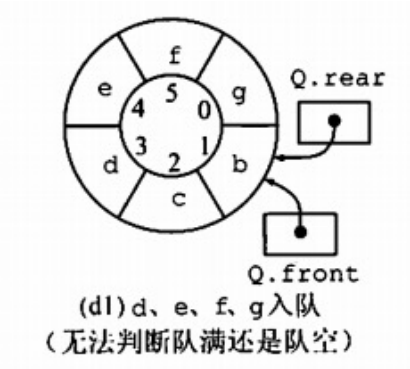
make some noise

那么,循环队列队空和队满的判断条件是什么呢?显然,队空的条件是Q.front==Q.rear。若入队元素的速度快于出队元素的速度，则队尾指针很快就会赶上队首指针，如图d1所示，此时可以看出队满时也有Q.front==Q.rear。

为了区分是队空还是队满的情况，有三种处理方式：

Then, what are the judgment conditions for the empty and full queue of the circular queue? Obviously, the condition that the team is empty is ***Q.front==Q.rear.*** If the speed of the entry element is faster than the speed of the exit element, the pointer at the end of the team will soon catch up with the pointer at the head of the team, as shown in the figure. At this time, it can be seen that there is ***Q.front==Q.rear*** when the team is full.

In order to distinguish between empty and full teams, there are three processing methods:



(d1) d、e、f、g入队
（无法判断队满还是队空）

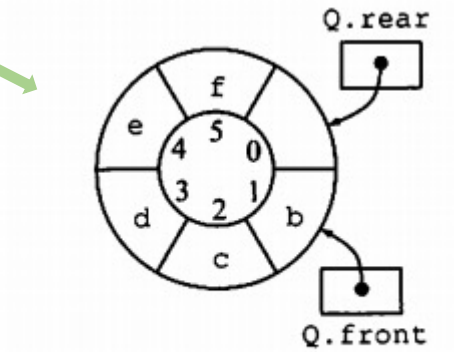
1. 牺牲一个单元来区分队空和队满，入队时少用一个队列单元，这是一种较为普遍的做法，约定以“**队头指针在队尾指针的下一位置作为队满的标志**”，如图d2所示。

It is a common practice to sacrifice one unit to distinguish between empty and full teams, and use one less queue unit when entering the team. It is agreed that "the next position of the team head pointer at the end of the team pointer is the sign of full teams", as shown in Figure ***d2***.

队满条件：`(Q.rear+1)%MaxSize==Q.front`

对空条件：`Q.front==Q.rear`

队列中元素的个数：`(Q.rear-Q.front+MaxSize) % MaxSize`



(d2) d、e、f入队
（牺牲一个存储单元）

Discuss

make some noise

2. 类型中增设表示元素个数的数据成员。这样，队空的条件为***Q.size==0***；队满的条件为***Q.size==MaxSize***。这两种情况都有***Q.front==Q.rear***。

Data members indicating the number of elements are added to the type. Thus, the condition of empty queue is ***Q.size==0***; The condition of full queue is ***Q.size==MaxSize***. In both cases, ***Q.front==Q.rear***.

3. 类型中增设***tag***数据成员，以区分是队满还是队空。tag等于0时，若因删除导致***Q.front==Q.rear***，则为队空；tag等于1时，若因插入导致***Q.front==Q.rear***，则为队满。

Add ***tag*** data members in the type to distinguish whether the team is full or empty. When ***tag*** is equal to ***0***, if ***Q.front==Q.rear*** due to deletion, the queue is empty; When ***tag*** is equal to ***1***, if ***Q.front==Q.rear*** is caused by insertion, the queue is full.

Discuss

3）If you want to make use of all the elements in the cyclic queue, you need to set a flag field ***tag***, and use the value of tag as ***0*** or ***1*** to distinguish whether the queue state is "***empty***" or "***full***" when the queue head pointer front and the queue tail pointer rear are the same. Try to write the queue in and queue out algorithms corresponding to this structure.

若希望循环队列中的元素都能得到利用，则需设置一个标志域tag，并以tag的值为0或1来区分队头指针front和队尾指针rear相同时的队列状态是“空”还是“满”。试编写与此结构相应的入队和出队算法。

在循环队列的类型结构中，增设一个tag的整型变量，进队时置tag为1，出队时置tag为0（因为只有入队操作可能导致队满,也只有出队操作可能导致队空）。队列Q初始时，置tag=0、front=rear=0。这样队列的4要素如下：

队满条件：`Q.front==Q.rear`且`Q.tag==1`

对空条件：`Q.front==Q.rear`且`Q.tag==0`

进队操作：`Q.data[Q.rear]=x; Q.rear=(Q.rear+1)%MaxSize; Q.tag=1;`

出队操作：`x=Q.data[Q.front]; Q.front=(Q.front+1)%MaxSize; Q.tag=0;`

In the type structure of the circular queue, an integer variable of ***tag*** is added. When entering the queue, set ***tag*** to ***1*** and when leaving the queue, set ***tag*** to ***0*** (because only entering the queue may cause the queue to be full, and only leaving the queue may cause the queue to be empty). At the beginning of queue ***Q***, set ***tag=0*** and ***front=rear=0***. The four elements of such a queue are as follows.

Discuss

1. 设“tag”法循环队列入队算法：*An algorithm of "tag" method for round robin queue entry*

```
int EnQueue1(SqQueue &Q,ElemType x){
    if(Q.front==Q.rear && Q.tag==1)
        return 0;                    //两个条件都满足时则队满
    Q.data[Q.rear]=x;
    Q.rear=(Q.rear+1)%MaxSize;
    Q.tag=1;                         //可能队满
    return 1;
}
```

2. 设“tag”法循环队列出队算法：*An algorithm of "tag" method for queue out of cyclic queue*

```
int DeQueue1(SqQueue &Q,ElemType &x){
    if(Q.front==Q.rear && Q.tag==0)
        return 0;                    //两个条件都满足时则队空
    x=Q.data[Q.front];
    Q.front=(Q.front+1)%MaxSize;
    Q.tag=0;                         //可能队空
    return 1;
}
```

Discuss

4）Modify the potato transfer simulation program to allow random counting, so that the results of each round are unpredictable.

修改传土豆模拟程序，允许随机计数，从而使每一轮的结果都不可预测。

```
def hotpotato(namelist):
    simqueue = Queue()
    for name in namelist:
        simqueue.enqueue(name)

    while simqueue.size() > 1:
        num = random.randrange(simqueue.size()*2)
        for i in range(num):
            simqueue.enqueue(simqueue.dequeue())
        simqueue.dequeue()
    return simqueue.dequeue()

def hotpotatotest():
    print(hotpotato(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']))
```

Discuss

make some noise

5）Since each node has only one reference to the following node, the implementation of the linked list given in this chapter is called a one-way linked list. Another implementation is called bidirectional linked list. In this implementation,
each node has a reference to the next node (usually called ***next***) and a reference to the previous node (usually called ***back***). The header reference also has two references, one pointing to the first node in the linked list and the other pointing to the last node. Please use ***python*** to implement bidirectional linked list.

由于每个节点都只有一个引用指向其后的节点，因此本章给出的链表实现称为单向链表。另一种实现称为双向链表。在这种实现中，每一个节点都有指向后一个节点的引用（通常称为next）和指向前一个节点的引用（通常称为back)。头引用同样也有两个引用，一个指向链表中的第一个节点，另一个指向最后一个节点。请用python实现双向链表。

```
class DoubleNode(Node):
    def __init__(self, initdata):
        super().__init__(initdata)
        self.pre = None

    def getpre(self):
        return self.pre

    def setpre(self, newpre):
        self.pre = newpre
```

Discuss

make some noise

```
class DoubleList(UnorderedList):
    def __init__(self):
        super().__init__()

    def add(self, item):
        temp = DoubleNode(item)
        temp.setNext(self.head)
        if self.head:
            self.head.setpre(temp)
        self.head = temp
```

```
def remove(self, v):
    cur = self.head
    pre = None
    found = False
    while not found:
        if cur.getData() == v:
            found = True
        else:
            pre = cur
            cur = cur.getNext()
    if not pre:
        self.head = cur.getNext()
        cur.getNext().setpre(None)
    else:
        pre.setNext(cur.getNext())
        pre.getNext().setpre(pre)
```

Discuss

```
def DoubleListtest():
    dl = DoubleList()
    dl.add(4)
    dl.add(3)
    dl.add(2)
    dl.add(1)
    print(str(dl))
    dl.remove(3)
    print(str(dl))
```