参考：https://learnku.com/docs/byte-of-python/2018 (https://learnku.com/docs/byte-of-python/2018)
《A Byte of Python》的中文译本，由社区维护，每年更新

看看当前的Python版本

In [2]:
```
from platform import python_version

print(python_version())
```

```
3.7.4
```

第一个Python语句；

注意 print 总是以一个不可见的 「新的一行」 字符（\n）作为结尾， 除非给他指定一个end

In [3]:
```
print ("Hello World!")
print("Tom")
print ("Hello World!",end=' ')
print("Tom", end=' ')
```

```
Hello World!
Tom
Hello World! Tom
```

超级计算器,超级大的数都没问题.

In [4]:
```
print(2**100)
```

```
1267650600228229401496703205376
```

In [5]:
```
2**100
```

Out[5]:
```
1267650600228229401496703205376
```

不可变类型的每次赋值操作都生成一个新对象

In [6]:
```
a=5; print(id(a))
a=6; print(id(a))
```

```
4458439952
4458439984
```

元组不能够进行元素赋值

In [7]:
```
a=3,4,5
a[1]=6
```

```
---------------------------------------------------------------------
---------
TypeError                                 Traceback (most recent c
all last)
<ipython-input-7-989b38129c3a> in <module>
      1 a=3,4,5
----> 2 a[1]=6

TypeError: 'tuple' object does not support item assignment
```

可变类型可以改变，可以元素赋值；相同的变量可能指向同一个对象，在对象上的操作两个变量都可见。

In [ ]:
```
l=[1,2,3]
m=l
print(l)
l[1]="hahahahaha"
print(l)
print(m)
```

Python在版本3.0以后对整数的大小不再进行限制，maxint没有了；但对list或str的索引下标仍有限制：sys.maxsize

In [ ]:
```
import sys
print(sys.maxsize)
print(sys.maxint)
```

常用的数学函数如sqrt/sin/cos等都在math模块中

In [ ]:
```
import math
print(math.sqrt(2))
```

字符串

Python字符串不可修改，只能生成新的字符串

In [ ]:
```
s='hello';
print(s)
print(id(s))
s += " world"
print(s)
print(id(s))
s[2]='x'
```

用[start:end:step]的方式从字符串中提取一部分。

In [ ]:
```
s='1234567'
print(s[1])
print(s[0:4])
print(s[-1:1:-1])
```

高级操作：split, join

In [ ]:
```
print('You are my best friend'.split())
'-'.join(["One", "Two", "Three"])
```

容器类型：列表和元组

In [ ]:
```
s='abc'
l=list(s)
print(l, id(l))
l.reverse()
print(l, id(l))
```

单元素元组的表示

In [ ]:
```
print((2)*4)
print((2,)*4)
print((2,3,4)*4)
```

集合的元素是不重复的，必须是可散列的

In [ ]:
```
aset=set('abcd')
print(aset)
print('x' in aset)
print('a' in aset)
aset.add('d')
print(aset)
aset.add([1,2,3])
```

什么东西是hashable的?

In [ ]:
```
print("hash(100) is {}".format(hash(100)))
print("hash('xzm') is {}".format(hash('xzm')))
print("hash((1,2,3)) is {}".format(hash((1,2,3))))
```

字典的key也必须是可散列的

In [8]:
```
a={}
a[1]=23
a['xzm']=[1,2,3]
a[(1,2,3)]='PKU'
a[[1,2,3]]='PKU'
```

```
---------------------------------------------------------------------
---------
TypeError                                 Traceback (most recent c
all last)
<ipython-input-8-2b2b8eb8054e> in <module>
      3 a['xzm']=[1,2,3]
      4 a[(1,2,3)]='PKU'
----> 5 a[[1,2,3]]='PKU'

TypeError: unhashable type: 'list'
```

字符串的format()方法

In [ ]:
```
age = 20
name = 'Swaroop'

print('{} was {} years old when he wrote this book'.format(name, age))
print('{0} was {1} years old when he wrote this book'.format(name, age))
print('Why is {0} playing with that python?'.format(name))
print('{author} write "{book}"'.format(author='Swaroop', book='a byte of python'))
```

format()可读性优于字符串拼接的方式

In [ ]:
```
print(name + ' is ' + str(age) + ' years old')
```

format其他的排版功能

取十进制小数点后的精度为 3 ， 得到的浮点数为 '0.333'

In [ ]:
```
print('{0:.3f}'.format(1.0/3))
```

填充下划线 (_) ， 文本居中 将 '***hello***' 的宽度扩充为 11

In [ ]:
```
print('{0:_^11}'.format('hello'))
```

对元组的format方式，使用'%'的话，需要特别注意

In [ ]:
```
name=(1,2,3)
print("hi there %s" % (name,))
print("hi there {}".format(name))
print("hi there %s" % name)
```

for-loop中else-block的作用

In [ ]:
```
for i in range(10):
    print(i,end=' ')
    if i==15:
        break
else:
    print('fini')

for i in range(20):
    print(i,end=' ')
    if i==15:
        break
else:
    print('fini')
```

Exception的处理

In [ ]:
```
def execute(func):
    try:
        print('try...')
        func()
    except TypeError as e:
        print('TypeError:', e)
    except ZeroDivisionError as e:
        print('ZeroDivisionError:', e)
    else:
        print('no error!')
    finally:
        print('finally...')
    print('END')

def f1():
    23/0
execute(f1)

def f2():
    '.'.join([1,2])
execute(f2)

def f3():
    3+4
execute(f3)

def f4():
    l=[1,2]
    l[34]=0
execute(f4)
```

函数本身 vs. 函数返回值

In [ ]:
```
def sum_list(alist):    #定义一个带参数的函数
    sum_tmp = 0
    for i in alist:
        sum_tmp += i
    return sum_tmp      #返回求和值

print(sum_list)
my_list = [23, 45, 67, 89, 100]
# 调用函数，将返回值赋值给 my_sum
my_sum = sum_list(my_list)
print("sum of my list is {}".format(my_sum))
```

没有return语句的函数返回None

In [ ]:
```
def sayhello(name):
    print("How are you, {}!".format(name))
c=sayhello("Tom")
print(c)
```

缺省参数的使用

In [ ]:
```
def func(a, b= 0, c= "abc"):
    print("{}:{}".format(c, a+b))

func(2)
func(2,4)
func(2,4, "shanghai")
func(2, c="beijing")
func(2, "guangzhou")
```

Support for type hints

让代码更加可读、友好，但python runtime没管它，属于“君子协定”；第三方软件可能用到它。
参考：https://docs.python.org/3/library/typing.html (https://docs.python.org/3/library/typing.html)
https://www.python.org/dev/peps/pep-0484/ (https://www.python.org/dev/peps/pep-0484/)

In [ ]:
```
def hello(name: int)-> str:
    for i in name:
        print(i)
    return 23

print(hello('xzm'))

from typing import List, Tuple

def top3(grades: List[Tuple[str, int]])->List[Tuple[str, int]]:
    grades.sort(reverse=True, key=lambda x:x[1])
    return grades[:3]

print(top3([("xzm", 4), ("abc", 7), ("ccc", 6), ("ddd", 10) ]))
```

Tuple assignment

In [ ]:
```
a, b= 1, 2
a, b = b, a
print(a, b)
##########################
a, b = 1, 2
a = b
b = a
print(a, b)
```

class的使用

In [ ]:
```
class Force: #二维力
    def __init__(self, x, y):
        self.fx, self.fy = x, y
    def show(self):
        print("Force<{},{}>".format(self.fx, self.fy))
    def add(self, force2):
        x = self.fx + force2.fx
        y = self.fy + force2.fy
        return Force(x, y)
    """
    def __add__(self, other):
        print("__add__ is called")
        return self.add(other)
    """
    __add__ = add

    def __str__(self):
        print("__str__ is called")
        return "Force<{},{}>".format(self.fx, self.fy)

    def __mul__(self, n):
        print("__mul__ is called")
        x = self.fx * n
        y = self.fy * n
        return Force(x, y)
    def __eq__(self, other):
        print("__eq__ is called")
        return self.fx == other.fx and self.fy == other.fy

#调用特殊操作符__add__()
(Force(1,3)+Force(2,4)).show()
#调用特殊操作符__mul__()
(Force(1,3)*5).show()
#调用特殊操作符__str__()
print(Force(1,3))
#调用特殊操作符__eq__()
if Force(1,3)*5 == Force(5,15):
    print("equal!")
```

Python中类的继承，父类与子类

In [ ]:
```
#
#car.py
#
class Car:
    def __init__(self,name):
        self.name = name
        self.remain_mile = 0
    def fill_fuel(self, miles): #加燃料里程
        self.remain_mile += miles
    def run(self, miles):   #行驶miles里程
        print(self.name, end=":")
        if self.remain_mile >= miles:
            self.remain_mile -= miles
            print("run {} miles".format(miles))
        else:
            print("fuel out!")
class GasCar(Car):
    def fill_fuel(self,gas): #加汽油gas升
        print("加油{}升".format(gas))
        super().fill_fuel(gas * 6.0) #每升油跑6miles
class ElecCar(Car):
    def fill_fuel(self, power): #充电power度
        print("充电{}度".format(power))
        super().fill_fuel(power * 3.0) #每度电跑3miles

mycar = ElecCar("唐")
mycar.fill_fuel(5)
mycar.run(10)
mycar.run(10)
```

In [ ]:
```
mycar = GasCar("悍马")
mycar.fill_fuel(4)
mycar.run(10)
```

使用自己实现的模块

In [ ]:
```
import os
import sys
module_path = os.path.abspath(os.path.join('code'))
if module_path not in sys.path:
    sys.path.append(module_path)
import foo
import importlib
importlib.reload(foo)
foo.hello("小苹果")
print()
```

module是怎么定义的, __name__是模块的一个属性,被python直接运行时__name__=="**main**"
被其他module import时，__name__就是模块名/文件名

In [ ]:
```
# code/foo.py
def hello(yourname):
    print("Hello! {}, my name is {}".format(yourname, __name__))

print("从其他地方import或者python foo.py，我都会执行")

if __name__ == "__main__":
    hello("汤姆")
```

算法分析

获取系统当前时间：从1970年一月一日零时开始的秒数

In [ ]:
```
import time
help(time.time)
print(time.time())
```

求和并计时

In [ ]:
```
import time
def sumofN2(n):
    start = time.time()      #记住开始时间
    theSum = 0
    #产生1, 2, 3, ... n-1, n的序列
    for i in range(1, n+1):
        theSum += i
    end = time.time()        #记住结束时间
    return theSum, end-start

for i in range(200000,1000000,200000):
    print("Sum({}) is {}, requires {:.7f} seconds".format(i, *sumof
N2(i)))
for i in range(2000000,10000000,2000000):
    print("Sum({}) is {}, requires {:.7f} seconds".format(i, *sumof
N2(i)))
```

测试in操作的时间

In [ ]:
```
import timeit
import random
print("size,   list_time,  dict_time")
for i in range(10000, 1000001,20000):   #从一万到一百万
    t = timeit.Timer("random.randrange({}) in x".format(i),
                     "from __main__ import random, x")
    x = list(range(i))
    lst_time = t.timeit(number=1000)
    x = {j:None for j in range(i)}
    d_time = t.timeit(number=1000)
    print("{}, {:.7f}, {:.7f} ".format(i, lst_time, d_time))
```

使用line_profiler分析语句频度T(n)

In [ ]:
```
from line_profiler import LineProfiler
def foo(n):
    a = 5
    b = 6
    c = 10
    for i in range(n):
        for j in range(n):
            x = i * i
            y = j * j
            z = i * j
    for k in range(n):
        w = a * k + 45
        v = b * b
    d = 33

if __name__ == "__main__":
    lprofiler = LineProfiler(foo)
    lprofiler.run('foo(10)')
    lprofiler.print_stats()
```

通过特殊函数实现对int()的支持

In [ ]:
```
class Uncle:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __int__(self):
        return self.age

print(int(Uncle("John", 48)))
```

数学画图模块的使用

In [ ]:
```
import matplotlib.pyplot as plt
number = int(input("please input a number:"))
plt.plot(range(number),range(number), label="linear")
plt.plot(range(number),[x*x for x in range(number)], label="square"
)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('Two or more lines on same plot with suitable legends ')
plt.legend()
plt.show()
```

测试列表index()操作和[]操作的时间复杂度

In [ ]:
```
import timeit
import random
start, end, step = (int(x) for x in input("Please input 3 numbers(s
tart, end, stop):").split(maxsplit=3))
print(start, end, step)
index_time = []
sub_time = []
rand_time = []
for N in range(start, end, step):
    t_index = timeit.Timer("mylist.index(random.randrange({}))".for
mat(N)
                           , "import random; from __main__ import my
list")
    t_sub = timeit.Timer("mylist[random.randrange({})]".format(N)
                         , "import random; from __main__ import mylist
")
    t_rand = timeit.Timer("random.randrange({})".format(N)
                          , "import random")
    mylist = list(range(N))
    t1 = t_index.timeit(number=1000)
    t2 = t_sub.timeit(number=1000)
    t3 = t_rand.timeit(number=1000)
    print(t1, t2, t3)
    index_time.append(t1)
    sub_time.append(t2)
    rand_time.append(t3)

import matplotlib.pyplot as plt
#plt.plot(range(start, end, step), index_time, label="index")
plt.plot(range(start, end, step), sub_time, label="sub")
plt.plot(range(start, end, step), rand_time, label="rand")
plt.xlabel('size of list')
plt.ylabel('1000*time')
plt.title('compare index() and [] operations on list')
plt.legend()
plt.show()
```

当N较小的时候，目标操作的执行时间较小，测量更加容易受到其他因素的干扰；
sub和rand操作用时较少，也容易收到干扰

讨论比特币挖矿中的算法复杂度问题

In [ ]:
```
import hashlib
def bthash(unicode):
    return hashlib.sha256(unicode.encode("utf8")).hexdigest()

hash_result = bthash("""戏剧中叔父克劳迪谋害了丹麦国王--哈姆雷特的父亲，篡了王位，并娶了国王的遗孀葛簇特；
王子哈姆雷特因此为父王之死向叔父复仇。剧本细致入微地刻画了伪装的、真实的疯癫 —— 从悲痛欲绝到假装愤怒 —— 探索了背叛、复仇、乱伦、堕落等主题""")
print("hash of《哈姆雷特介绍》：{}".format(hash_result))

difficulty_bits = 4
difficulty = 2 ** difficulty_bits
target = 2 ** (256 - difficulty_bits)
print("Difficulty is {}({} bits),\nless than:{:>64x}".format(
    difficulty, difficulty_bits, target))
#
# 核心问题，求解nonce，满足条件：
#     Curr_hash = hash(transactions + nonce + Prev_hash) < target
#
def validate(transactions, nonce):
    block = transactions + [nonce] + ["前一个块哈希:13b1b06...76d3d"]
    hash_result = bthash(str(block))
    print("nonce {:>3}:{}".format(nonce, hash_result))
    if int(hash_result, 16) < target:
        return True
    else :
        return False

def find_nonce(transactions):                #挖矿过程
    ntries = 256
    for nonce in range(ntries):              #随机数
        if validate(transactions, nonce):
            print("A new block mined with nonce {}".format(nonce))
            return nonce
    else:
        print("Failed in ntries:{}".format(ntries))
        return None

if __name__ == "__main__":
    transactions = ["Alice sends .5 coins to Bob",
             "Bob sends 2 coins to John",
             "Alice send .1 coins to Kate",
             "John send 1 coins to Miselle",
             "Miselle send 2 coins to Alice",
               "矿工(光头强)被奖励一个coin"]
    nonce = find_nonce(transactions)
    if nonce:
        print("光头强成功开采出了一个区块，获得一个比特币奖励。")
```

最新的比特币区块：https://btc.com/ (https://btc.com/)

比特币挖矿难度变更趋势：https://btc.com/stats/diff (https://btc.com/stats/diff)

其中的难度和难度对数，分别对应上面代码中的difficulty和difficulty_bits 当difficulty_bits最大取256时，difficulty数超过已知宇宙中的原子数，远远超过佛教中的“恒河沙数”。

比特币价格变化趋势：https://www.coindesk.com/price/bitcoin (https://www.coindesk.com/price/bitcoin)

SHA256函数的更多信息：https://en.wikipedia.org/wiki/SHA-2 (https://en.wikipedia.org/wiki/SHA-2)

思考

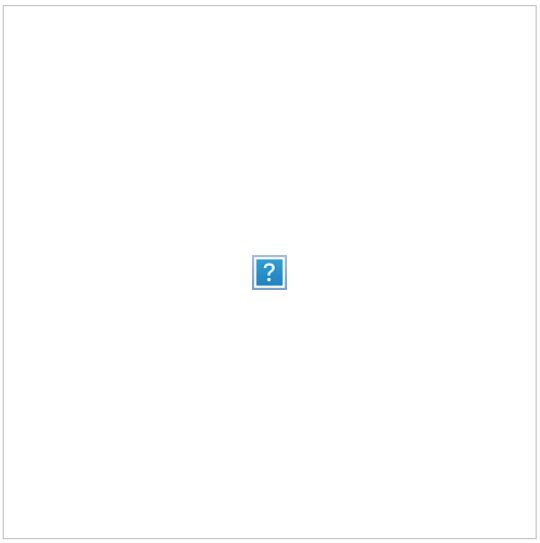
1：上面validate和find_nonce的复杂度分别是多少？它们是什么关系？分别属于NP还是P?
2: 如果想把“Alice send .1 coins to Kate”改成“Alice send 100 coins to Kate”，计算一个新nonce来保持“块哈希”值不变，算法复杂度是多少?

In [ ]:
```
l=[1,2,3]
r=['xzm', 'aa', 'bb']
[x for x in zip(l,r)]
```

线性表

Python list类型是顺序表，存储的实际是指针

In [ ]:
```
mylist = [1, 2, 45, 45]
print([id(i) for i in mylist])
mylist.insert(2, 5)
print(mylist)
print([id(i) for i in mylist])
```



用Python list来实现stack ADT，实现后的stack是一个数据类型

通常在限制少的数据类型上加一些限制，来实现限制多的数据类型

In [9]:
```
#成员的方式实现
class Stack:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[-1]
    def size(self):
        return len(self.items)
    def __str__(self):
       return "[ {} *".format(", ".join([str(i) for i in self.items
]))
```

In [10]:
```
s = Stack()
print(s.isEmpty())
```

```
True
```

In [11]:
```
s.push(4)
s.push("Dog")
print(s)
print(s.peek())
```

```
[ 4, Dog *
Dog
```

In [12]:
```
s.push(True)
print(s.size())
print(s.isEmpty())
```

```
3
False
```

In [13]:
```
s.push(8.4)
print(s.pop())
print(s.pop())
print(s.size())
print(s)
```

```
8.4
True
2
[ 4, Dog *
```

In [14]:
```
### 继承list的方式实现stack
class stack(list):
    def isEmpty(self):
        return self==[]
    def push(self, item):
        self.append(item)
    #def pop(self):
    #   return super().pop()
    #此处省略完整实现...
ss = stack()
print(ss.isEmpty())
ss.push("xzm")
print(ss.pop())
```

```
True
xzm
```

In [15]:
```
#括号匹配
def parChecker(symbolString):
    s = Stack()
    for c in symbolString:
        if c=='(':
            s.push(c)
        else : # 输入的字符，不是'('，就当成')'了
            if s.isEmpty():
                return False
            else:
                s.pop()
    else:
        return s.isEmpty()
print(parChecker("((()))"))
print(parChecker("(()"))
print(parChecker("())"))
```

```
True
False
False
```

In [16]:
```
#多种括号匹配(),[],{}
pars={'(':')','[':']','{':'}'}
def parChecker(symbolString):
    s = Stack()
    for c in symbolString:
        if c in pars:
            s.push(c)
        else :
            if s.isEmpty():
                return False
            elif pars[s.peek()] == c:
                s.pop()
            else:
                return False
    else:
        return s.isEmpty()
print(parChecker("([{}])"))
print(parChecker("([{}]a)"))
```

```
True
False
```

In [17]:
```
#中缀转后缀代码
prec = {"*":3,
        "/":3,
        "+":2,
        "-":2}
def infixToPostfix(infixexpr):
    opStack = Stack()
    postfixList = []
    tokenList = infixexpr.split()
    for token in tokenList:
        if token in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" \
          or token in "0123456789":  #操作数的处理
            postfixList.append(token)
        elif token == '(':           #标记子表达式开始
            opStack.push(token)
        elif token == ')':           #子表达式结束
            while opStack.peek() != '(':
                postfixList.append(opStack.pop())
            else:
                opStack.pop()        #弹出'('
        else:                        #操作符
            while (not opStack.isEmpty()
                   and opStack.peek() != '('
                   and prec[opStack.peek()] >= prec[token]):
                postfixList.append(opStack.pop())
            opStack.push(token)     #所有操作符都必须进栈等待
    while not opStack.isEmpty():
        postfixList.append(opStack.pop())
    return " ".join(postfixList)

print(infixToPostfix("A + B * 5"))
print(infixToPostfix("( A + B ) * 5"))
```

```
A B 5 * +
A B + 5 *
```

In [18]:
```
class ListNode:
     def __init__(self, val=0, next=None):
         self.val = val
         self.next = next

def next(l):
    return l.next, l.val

l1=ListNode(20)
l1, num = next(l1)
print(l1, num)
```

```
None 20
```

比较“==”与“=”的优先级

In [19]:
```
f = 4==5
print(f)
```

```
False
```

特殊函数的调用，不会出现多重歧义。

In [20]:
```
class foo:
    def __init__(self, num):
        self.num = num
    def __int__(self):
        return self.num
    #def __str__(self):
    #   return "hello"

b = foo(24)
print(b)
```

```
<__main__.foo object at 0x7fd392412650>
```

Python list实现Queue

In [53]:
```
class Queue_on_list:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def enqueue(self, item):
        self.items.insert(0, item) #0为队尾
    def dequeue(self):
        return self.items.pop()
    def size(self):
        return len(self.items)
```

热土豆问题

In [22]:
```
Queue = Queue_on_list
def hotPotato(namelist, num):
    que =Queue()
    for name in namelist:
        que.enqueue(name)
    while que.size() > 1:
        for i in range(num):
            que.enqueue(que.dequeue())
        que.dequeue()  #杀掉一个
    return que.dequeue()

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

```
Susan
```

打印任务问题

In [23]:
```
import random
class Task:
    def __init__(self, time):
        self.arrive_time = time
        self.pages = random.randrange(1,21)
    def waitTime(self, current_time):
        return current_time - self.arrive_time

Queue = Queue_on_list

class Printer:
    def __init__(self,ppm):
        self.pagerate = ppm     #打印速度，每分钟几页
        self.currentTask = None
        self.timeRemaining = 0 #任务倒计时
    def tick(self):
        if self.currentTask != None:
            self.timeRemaining -= 1
            if self.timeRemaining <= 0:
                self.currentTask = None
    def busy(self):
        return self.currentTask != None
    def startNext(self, newtask):
        self.currentTask = newtask
        self.timeRemaining = newtask.pages * 60 / self.pagerate

def newPrintTask():
    num = random.randrange(1,181)
    return num == 180

def simulation(numSeconds, pagesPerMinute):
    labprinter = Printer(pagesPerMinute)
    printQueue = Queue()
    waitingtimes = []
    for currentSecond in range(numSeconds):
        if newPrintTask():
            printQueue.enqueue(Task(currentSecond))
        if (not labprinter.busy()) and \
           (not printQueue.isEmpty()):
            nexttask = printQueue.dequeue()
            waitingtimes.append(nexttask.waitTime(currentSecond))
            labprinter.startNext(nexttask)
        labprinter.tick()
    #模拟结束，统计结果
    averageWait = sum(waitingtimes)/len(waitingtimes)
    print("Average Wait {:.3f} secs {} tasks remaining.".format(
        averageWait, printQueue.size()))

for i in range(10):
    simulation(7200,10)
```

```
Average Wait 22.864 secs 0 tasks remaining.
Average Wait 20.592 secs 0 tasks remaining.
Average Wait 18.538 secs 0 tasks remaining.
Average Wait 20.105 secs 0 tasks remaining.
Average Wait 17.698 secs 0 tasks remaining.
Average Wait 23.426 secs 0 tasks remaining.
Average Wait 21.018 secs 0 tasks remaining.
Average Wait 11.524 secs 0 tasks remaining.
Average Wait 41.891 secs 0 tasks remaining.
Average Wait 31.105 secs 0 tasks remaining.
```

Python list实现Deque

In [24]:
```
class Deque_on_list:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def addFront(self, item):
        self.items.append(item)
    def addRear(self, item):   # 0位置当尾巴
        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)
```

“回文词”判定

In [25]:
```
Deque = Deque_on_list

def isPalindromic(str):
    dq = Deque()
    for c in str:
        dq.addFront(c)
    while dq.size()>1:
        if dq.removeFront() != dq.removeRear():
            return False
    else:
        return True

print("lasdsal is {}palindromic".format("" if isPalindromic("lasdsa
l") else "NOT "))
print("lasdsalaaa is {}palindromic".format("" if isPalindromic("las
dsalaaa") else "NOT "))
```

```
lasdsal is palindromic
lasdsalaaa is NOT palindromic
```

链表的实现

In [26]:
```
class Node:
    def __init__(self, initdata=None):
        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)
print(temp.getData())

class UnorderedList:
    def __init__(self):
        self.head = None
    def add(self, item):
        temp = Node(item)
        temp.setNext(self.head)
        self.head = temp

class UnorderedListWithNulHead:
    def __init__(self):
        self.head = Node()
```

```
93
```

In [27]:
```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade
    def __lt__(self, other):
        return self.grade > other.grade
    def __str__(self):
        return "({},{})".format(self.name, self.grade)
    __repr__ = __str__

s = []
s.append(Student("Jack", 80))
s.append(Student("Cook", 90))
s.sort()
print(s)
s.sort(key=lambda x:x.name, reverse=True)
print(s)
```

```
[(Cook,90), (Jack,80)]
[(Jack,80), (Cook,90)]
```

双链表的append和remove

小心判断边界条件和保持不变量

In [28]:
```
class DNode:
    def __init__(self, initdata=None):
        self.data = initdata
        self.next = None
        self.prev = None

class UnorderedList:
    def __init__(self):
        self.head = self.tail = None
    def __str__(self):
        ds, cur = [], self.head
        while cur != None:
            ds.append(cur.data)
            cur = cur.next
        return "{}".format(ds)
    def append(self, data):
        temp = DNode(data)
        if self.head is None:                    #插入空表
            self.head = self.tail = temp
        else:
            self.tail.next = temp                #从表尾插入
            temp.prev = self.tail
            self.tail = temp
    def remove(self, data):
        curr = self.head
        while curr != None:
            if curr.data == data:
                break
            curr = curr.next
        else:
            return                               #即使空表，也在这里返回
        #if self.tail == self.head:              #唯一元素   为什么这个地方要单独判断?
         #   self.tail = self.head = None
        if self.tail == curr:                    #删尾
            self.tail = self.tail.prev                  # ---> 看这里。
            self.tail.next = None
        elif self.head == curr:
            self.head = self.head.next
            self.head.next = None
        else:
            curr.prev.next = curr.next
            curr.next.prev = curr.prev
        return

dl = UnorderedList()
dl.append(1)
```

```
dl.append(2)
dl.append(3)
print(dl)
dl.remove(2); print(dl)
dl.remove(2); print(dl)
dl.remove(1); print(dl)
dl.remove(3); print(dl)
```

```
[1, 2, 3]
[1, 3]
[1, 3]
[3]
[3]
```

海龟作图在Jupiter Notebook中不正常

import turtle
t = turtle.Turtle()
w = turtle.Screen()
t.forward(100)
w.exitonclick()

递归法找硬币

In [30]:
```
def recMC(coinValueList, change):
    if change in coinValueList:
        return 1
    return 1+min([recMC(coinValueList, change-c)
                  for c in coinValueList if c <= change],default=chan
ge)

print(recMC([1,5,10,25], 63))
```

```
---------------------------------------------------------------------
---------
KeyboardInterrupt                         Traceback (most recent c
all last)
<ipython-input-30-e3d7602766bb> in <module>
      5                 for c in coinValueList if c <= change],def
ault=change)
      6
----> 7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
```

```
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
```

```
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))
```

```
<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6 
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6 
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6 
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6 
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6 
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
      6 
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],def
ault=change)
```

```
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                 for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
```

```
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                  for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                  for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                  for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                  for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                  for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                  for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                  for c in coinValueList if c <= change],def
ault=change)
      6
      7 print(recMC([1,5,10,25], 63))
```

```
<ipython-input-30-e3d7602766bb> in recMC(coinValueList, change)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],default=change)
      6
      7 print(recMC([1,5,10,25], 63))

<ipython-input-30-e3d7602766bb> in <listcomp>(.0)
      3         return 1
      4     return 1+min([recMC(coinValueList, change-c)
----> 5                   for c in coinValueList if c <= change],default=change)
      6
      7 print(recMC([1,5,10,25], 63))

KeyboardInterrupt:
```

In [ ]:
```
#### 递归法找硬币(带中间结果缓存)
```

In [ ]:
```
def recMC(coinValueList, change, knownResults):
    if knownResults[change] > 0:
        return knownResults[change]
    elif change in coinValueList:
        knownResults[change] = 1
        return 1
    knownResults[change] = 1+min([recMC(coinValueList, change-c, knownResults)
                  for c in coinValueList if c <= change],default=change)
    return knownResults[change]

print(recMC([1,5,10,25], 63, [0]*64))
```

动规找硬币

In [ ]:
```
def dpMC(coinValueList, change, minCoins):
    for cents in range(change+1):
        # minCoins[less than cents] ==> minCoins[cents]
        minCoins[cents] = min([1+minCoins[cents-c]
             for c in coinValueList if c <= cents], default=cents)
    return minCoins[change]

print(dpMC([1,5,10,25], 63, [0]*64))
```

动规找硬币（扩展）

In [ ]:
```
def dpMC(coinValueList, change, minCoins, coinsUsed):
    for cents in range(change+1):
        # minCoins[less than cents] ==> minCoins[cents]
        ll = [(1+minCoins[cents-c],c) for c in coinValueList if c <
= cents]
        minCoins[cents], coinsUsed[cents] = min(ll, key=lambda x:x[
0], default=(cents, 1))
    return minCoins[change]
def printCoins(coinsUsed, change):
    coin = change
    while coin > 0:
        thisCoin = coinsUsed[coin]
        print(thisCoin, end=' ')
        coin -= thisCoin
    print()
coinsUsed = [0]*64
print(dpMC([1,5,10,25], 63, [0]*64, coinsUsed))
printCoins(coinsUsed, 63)
print(dpMC([1,5,10,21, 25], 63, [0]*64, coinsUsed)) #增加了奇葩21
printCoins(coinsUsed, 63)
```

O(mn)的字符串查找算法

In [ ]:
```
def indexOmn(S, P, pos=0):
    i=0                         #P的读写头
    j=pos                         #S的读写头
    while i<len(P) and j<len(S):
        if P[i] == S[j]:        #两个读写头下的字符相等
            i += 1
            j += 1
        else:                   #不等
            j = j - i + 1       #把P右移一格，重新比较
            i = 0
    else:
        if i == len(P):         #找到了一个匹配
            return j-i
        else:
            return None

print(indexOmn("baababababacaca", "ababaca"))
print(indexOmn("baababababacaca", "ababaxa"))
```

求Pattern所有前缀（包括自身）的最长公共前后缀

In [ ]:
```
def partial(pattern):
    ret = [0,0]
    for i in range(1, len(pattern)):
        j = ret[i]
        while j > 0 and pattern[j] != pattern[i]:
            j = ret[j]
        ret.append(j+1 if pattern[j] == pattern[i] else 0)
    return ret

if __name__ == "__main__":
    print(partial('ababac'))
    print(partial('abababa'))
    print(partial('aaaaa'))
```

In [ ]:
```
def indexKMP(S, P, pos=0):
    i=0                         #P的读写头
    j=pos                         #S的读写头
    part=partial(P)             #计算P的partial
    while i<len(P) and j<len(S):
        if P[i] == S[j]:        #两个读写头下的字符相等
            i += 1
            j += 1
        else:                   #不等
            if i == 0:
                j += 1
            else:
                i = part[i]
    else:
        if i == len(P):         #找到了一个匹配
            return j-i
        else:
            return None

if __name__ == "__main__":
    print(indexKMP("baababababacaca", "abababa"))
    print(indexKMP("baababababacaca", "ababaxa"))
```

实测indexOmn和indexKMP的效率

In [ ]:
```
def read_text(fn):
    with open(fn, 'r') as file:
        return file.read().replace('\n', '')

#data = read_text('1618824805.txt')  #红楼梦
data = read_text('10-0.txt')          #圣经(旧约+新约)
print(len(data))
print(data[100000:100200])
print("====================================")
#query="宝哥哥"
#query="宝玉来了"
#query="林姐姐"
#query="空空道人"
#query="林妹妹"
#query="那是个最小性儿又多心的"
#query="所以到底不长命"
query="Kings"
#query="My lord" #****

def searchAll(S, P, fun):
    pos = 0
    while pos is not None:
        pos = fun(S, P, pos)
        if pos :
            print(S[pos:pos+20])
            pos += len(P)

from timeit import Timer
t1 = Timer("searchAll(data, query, indexKMP)", "from __main__ impor
t data, query, searchAll, indexKMP")
print("=========indexKMP cost time is {}".format(t1.timeit(number=1
)))
t2 = Timer("searchAll(data, query, indexOmn)", "from __main__ impor
t data, query, searchAll, indexOmn")
print("=========indexOmn cost time is {}".format(t2.timeit(number=1
)))
```

In [ ]:
```
from line_profiler import LineProfiler
def sumss(n):
    res = 0
    for i in range(n):
        res += i
    return sum
if __name__ == "__main__":
    lprofiler = LineProfiler(sumss)
    lprofiler.run('sumss(5)')
    lprofiler.print_stats()
```

顺序查找无序表

In [ ]:
```
def sequentialSearch(alist, item):
    for elem in alist:
        if elem == item:
            return True
    else:
        return False

if __name__ == "__main__":
    testlist = [1,2,32,8,17,19,42,13,0]
    print(sequentialSearch(testlist, 3))
    print(sequentialSearch(testlist, 13))
```

二分查找的程序执行分析

In [ ]:
```
def binarySearch_loop(alist, item):
    first, last = 0, len(alist)-1
    while first <= last:
        midpoint = (first + last) //2
        if alist[midpoint] == item:
            return True
        elif alist[midpoint] > item:
            last = midpoint -1
        else:
            first = midpoint + 1
    else:
        return False

def binarySearch_recur(alist, item):
    if not alist:
        return False
    midpoint = len(alist) // 2
    if alist[midpoint] == item:
        return True
    elif alist[midpoint] > item:
        return binarySearch_recur(alist[:midpoint], item)
    else:
        return binarySearch_recur(alist[midpoint+1:], item)

from line_profiler import LineProfiler
def lprof(func, code):
    lprof = LineProfiler(func)
    lprof.run(code)
    lprof.print_stats()

if __name__ == "__main__":
    testlist = [0, 1, 2, 8, 13, 17,19, 32, 42]
    print(binarySearch_loop(testlist, 2))
    print(binarySearch_loop(list(range(100)), 2))
    lprof(binarySearch_loop, 'binarySearch_loop(list(range(10000)),
2)')
    lprof(binarySearch_recur, 'binarySearch_recur(list(range(10000)
), 2)')
```

Python内部的hash函数，应用于dict和set

In [ ]:
```
print(hash(1))
print(hash(2))
print(hash(12345678987654321888999111222))
print(hash('xzm'))
hash([1,2,3])
d={[1,2,3]:5}
```

更加高级一点的hash函数

In [ ]:
```
import hashlib
print(hashlib.md5(b"hello world!").hexdigest())
print(hashlib.sha1(b"hello world!").hexdigest())
print(hashlib.sha256(b"hello world!").hexdigest())  # 比特币挖矿算法中出现
```

散列函数的update()方法

In [ ]:
```
import hashlib
m = hashlib.md5()
m.update(b"hello world!")
m.update(b"this is part #2")
m.update(b"this is part #3")
print(m.hexdigest())
print(hashlib.md5((b"hello world!"
                   b"this is part #2"
                   b"this is part #3")).hexdigest()) # 字符串跨行语法
```

桶排序_计数法实现

In [ ]:
```
def BucketSort_counter(alist, ceiling, key=lambda x:x):  #key的取值范围是[0,ceiling)
    blist = [None]*len(alist)           #临时数组
    count = [0]*ceiling                 #初始化计数器
    for i in alist:
        count[key(i)] += 1              #统计每个key出现的次数
    #print(count)
    for i in range(1, len(count)):
        count[i] += count[i-1]          #统计累计计数的key次数(<=key)
                                        #其实就是对应元素应该的排位
    #print(count)
    for i in range(len(blist)-1, -1, -1):   #从尾部开始保持稳定性
        count[key(alist[i])] -= 1
        blist[count[key(alist[i])]] = alist[i]          #重新排位
    return blist

if __name__ == "__main__":
    BucketSort = BucketSort_counter
    print(BucketSort([7, 3, 8, 9, 6, 1, 8, 1, 2], 10))
    print(BucketSort([('Mike', 2), ('Jack', 4), ('Alice', 4)
                      ,('John', 5), ('Bob', 3)], 6
                     ,key=lambda x:x[1]))  #什么叫做排序的稳定性?
                                           #或者叫保序。
```

桶排序_容器法实现

In [ ]:
```
def BucketSort_container(alist, ceiling, key=lambda x:x):  #key的取值范围是[0,ceiling)
    #container = [[]]*ceiling          #为什么这样不行?
    container = [ [] for _ in range(ceiling)]
    for i in alist:
        container[key(i)].append(i)    #分配
    blist = []
    for bucket in container:
        blist.extend(bucket)           #回收
    return blist

if __name__ == "__main__":
    BucketSort = BucketSort_container
    print(BucketSort([7, 3, 8, 9, 6, 1, 8, 1, 2], 10))
    print(BucketSort([('Mike', 2), ('Jack', 4), ('Alice', 4)
                      ,('John', 5), ('Bob', 3)], 6
                     ,key=lambda x:x[1]))  #什么叫做排序的稳定性?
                                           #或者叫保序。
```

生成列表的列表，两种不同的方法

In [ ]:
```
[[]]*10
```

In [ ]:
```
[[] for _ in range(10)]
```

In [ ]:
```
l = [1, 2, 3]
print(l,id(l))
l.extend([5,6,7])
l.append([5,6,7])
print(l, id(l))
```

基数排序

In [ ]:
```
def base_sort(alist, base, code_num, BucketSort = BucketSort_container):
    for i in range(code_num):
        print(alist)
        #从低位开始对每一个排序码，调用BucketSort()
        alist = BucketSort(alist, base, key=lambda x:x//base**i%base)
    return alist

if __name__ == "__main__":
    print("result =", base_sort([231, 134, 399, 510, 211, 304, 595], 10, 3))
```

In [ ]:
```
n=2000121
n.bit_length()
```

二叉树的节点链结实现

In [2]:
```
class BinaryTree:
    def __init__(self, rootObj, \
            left=None, right=None):                    #可以同时设置节点的左右子树
        self.key = rootObj
        self.leftChild = left
        self.rightChild = right

    def insertLeft(self, newNode):
        self.leftChild = BinaryTree(newNode, \
                left=self.leftChild)

    def insertRight(self, newNode):
        self.rightChild = BinaryTree(newNode, \
                right = self.rightChild)

    def getRightChild(self):
        return self.rightChild;

    def getLeftChild(self):
        return self.leftChild

    def setRootVal(self, obj):
        self.key = obj

    def getRootVal(self):
        return self.key
```

打印二叉树

In [3]:
```
def _print_t(tree, is_left, offset, depth, buf, label):
    if not tree:
        return 0

    b = "{:^5}".format(label(tree))
    width = 5
    while len(buf)<2*depth+1:                    #让后面的buf[2*depth]访
问有效
        buf.append([])
    left  = _print_t(tree.leftChild,  True, offset,
depth + 1, buf, label);
    right = _print_t(tree.rightChild, False, offset + left + width,
depth + 1, buf, label);

    enlarge = offset+left+width+right-len(buf[2*depth])
    buf[2*depth].extend([' ']*enlarge)
    for i, c in enumerate(b):                    #输出子树根节点的内容
        buf[2*depth][offset+left+i] = c


    if depth > 0:                                #输出子树与其父节点的连线
        if is_left:
            enlarge = offset+left+2*width+right-len(buf[2*depth-1])
            buf[2*depth-1].extend([' ']*enlarge)
            for i in range(width+right):
                buf[2 * depth - 1][offset + left + width//2 + i] =
'_'
            buf[2 * depth - 1][offset + left + width//2] = '+';
            buf[2 * depth - 1][offset + left + width//2 + right + w
idth] = '+';
        else:
            enlarge = offset+left+width+right-len(buf[2*depth-1])
            buf[2*depth-1].extend([' ']*enlarge)
            for i in range(left+width):
                buf[2 * depth - 1][offset - width//2 + i] = '-';
            buf[2 * depth - 1][offset + left + width//2] = '+';
            buf[2 * depth - 1][offset - width//2 - 1] = '+';
    return left + width + right;

def print_t(tree, label=lambda x:x.key):
    buf = []
    _print_t(tree, True, 0, 0, buf, label)
    for l in buf:
        print(''.join(l))
```

全括号表达式转二叉树结构

In [36]:
```
def buildParseTree(fpexp):                          #full parentheses
    fplist = fpexp.split()                          #分解单词：输入表达式中单词需
用空格分开
    pStack = []                                     #python list是多面手，可以
把它当栈用。
    currentTree = eTree = BinaryTree('')    #创建一棵树空树
    #pStack.append(currentTree)                      #这是在干什么？？？？
    for i in fplist[:-1]:
        if i=="(":                                  #子表达式开始
            currentTree.insertLeft('')
            pStack.append(currentTree)      #入栈左下降
            currentTree = currentTree.getLeftChild()
        elif i in ['+', '-', '*', '/']:     #操作符
            currentTree.setRootVal(i)
            currentTree.insertRight('')
            pStack.append(currentTree)      #入栈右下降
            currentTree = currentTree.getRightChild()
        elif i == ')':                              #子表达式结束
            currentTree = pStack.pop()
        elif ord('0')<= ord(i)<= ord('9'): #操作数
            currentTree.setRootVal(int(i))
            currentTree = pStack.pop()      #出栈上升
        else:
            raise ValueError
    return eTree

import operator
def evaluate(parseTree):                            #递归法实现
    opers = {'+':operator.add, '-':operator.sub,
             '*':operator.mul, '/':operator.truediv}
    leftC = parseTree.getLeftChild()        #取子树，缩小规模
    rightC = parseTree.getRightChild()

    if leftC and rightC:
        fn = opers[parseTree.getRootVal()]
        return fn(evaluate(leftC)           #递归调用
                  , evaluate(rightC))
    else:
        return parseTree.getRootVal()       #基本结束条件


if __name__ == "__main__":
    # expr = input("Please input:")
    tree = buildParseTree("( 5 + ( 9 * 2 ) )")
    print_t(tree)
    print(evaluate(tree))
```

```
      +
  +---+---------+
  5             *
            +---+---+
            9       2
23
```

中缀表达式转二叉树

In [ ]:
```
def tokenize(expr):
    """
    支持的token类型
    运算符：‘+’, ‘-’, ‘*’, ‘/’
    操作数：整数，标识符（以字母开头，包含字母或数字）
    括号：‘(’, ‘)’
    """
    for _expr in expr.split():
        for i in _tokenize(_expr):
            yield i
def _tokenize(expr):                        #不用管表达式中的空格了
    buf = []
    for i in expr:
        if i in '+-*/()':
            if buf:
                yield ''.join(buf); buf=[]
            yield i
        elif i.isalnum():
            if buf and buf[0].isdigit() and i.isalpha():
                yield ''.join(buf); buf=[]
            buf.append(i)
        else:
            raise ValueError("Unknown charactor:'{}'".format(i))
    if buf:
        yield ''.join(buf); buf=[]

priority = {'(':0, '+':1, '-':1, '*':2, '/':2}
def buildParseTree(inexp):                  #中缀表达式
    subtree_stack = []
    op_stack = []
    for t in tokenize(inexp):               #从字符串中提取token
        if t in '+-*/':
            while len(op_stack)>0 and priority[t] <= priority[op_st
ack[-1]]:
                if len(subtree_stack) < 2:
                    raise SyntaxError("operand missing")
                subtree_stack.append(BinaryTree(op_stack.pop(),
                                        right = subtree_stack.pop(),
                                        left = subtree_stack.pop()))
            op_stack.append(t)
        elif t == ')':
            while len(op_stack)>0 and op_stack[-1] != '(':
                if len(subtree_stack) < 2:
                    raise SyntaxError("operand missing")
                subtree_stack.append(BinaryTree(op_stack.pop(),
                                       right = subtree_stack.pop(),
                                       left = subtree_stack.pop()))
            else:
```

```
                if len(op_stack)==0:
                    raise SyntaxError("Unexpected ')'")
                else:
                    op_stack.pop()      #pop a '('
            elif t == '(':
                op_stack.append(t)
            else:                               #操作数
                subtree_stack.append(BinaryTree(t))
        while len(op_stack)>0:
            if len(subtree_stack) < 2:
                raise SyntaxError("operand missing")
            subtree_stack.append(BinaryTree(op_stack.pop(),
                                 right = subtree_stack.pop(),
                                 left = subtree_stack.pop()))
        if len(subtree_stack) > 1:
            raise SyntaxError("Unexpected operand '{}'".format(subtree_
stack[-1]))
        return subtree_stack[0]

if __name__ == "__main__":
    expr = input()
    #expr = "(a + b)*h/2"
    for t in tokenlize(expr):
        print(t)
    pt = buildParseTree(expr)
    print_t(pt)
```

函数调用参数列表中函数的执行顺序：从左到右

In [ ]:
```
class foo:
    def __init__(self):
        self.num=0

    def next(self):
        self.num += 1
        return self.num

def bar(first, second):
    print("first={}".format(first))
    print("second={}".format(second))

a = foo()
bar(second=a.next(), first=a.next())
bar(a.next(), a.next())
```

二叉堆的“类”实现

In [37]:
```
class BinHeap:
    def __init__(self, key=lambda x:x):
```

```
        self.heapList = [0]
        self.currentSize = 0
        self.key = key

    def percUp(self, i):
        while i // 2 > 0:
            if self.key(self.heapList[i]) < self.key(self.heapList[
i//2]):
                self.heapList[i//2], self.heapList[i] \
                =self.heapList[i], self.heapList[i//2]  #与父节点进行
交换
            else:
                break                                   #提前停止上浮
            i = i//2                                    #沿路径向上

    def insert(self, elem):
        self.heapList.append(elem)                      #添加到末尾
        self.currentSize += 1
        self.percUp(self.currentSize)                   #新元素按照ke
y上浮

    def percDown(self, i):
        while (i * 2) <= self.currentSize:
            mc = self.minChild(i)
            if self.key(self.heapList[i]) > self.key(self.heapList[
mc]):
                self.heapList[i], self.heapList[mc] \
                =self.heapList[mc], self.heapList[i]    #交换下沉
            else:
                break                                   #提前终止
            i = mc                                      #沿路径向下

    def minChild(self,i):
        if i*2 > self.currentSize:
            return None                                 #没有子节点
        elif i*2 == self.currentSize:                   #只有左子节点
            return i*2
        return i*2 if self.key(self.heapList[i*2])<self.key(self.he
apList[i*2+1]) \
            else i*2+1

    def delMin(self):
        retval = self.heapList[1]                       #移走堆顶
        self.heapList[1] = self.heapList[self.currentSize]
        self.currentSize -= 1
        self.heapList.pop()
        self.percDown(1)                                #新顶下沉
        return retval

    def buildHeap(self, alist):
        i = len(alist) // 2
        self.currentSize = len(alist)
        self.heapList = [0] + alist
```

```
        while i>0:
            self.percDown(i)
            i -= 1
```

通过二叉堆来生成哈夫曼树

In [ ]:
```
from typing import List, Tuple
def haffmanTree(alist: List[Tuple[str, int]]):
    pq = BinHeap(key=lambda x:x[0][1])
    pq.buildHeap([[i,[],[]]for i in alist])          #以字符元素为根，左右子树为空集
    while pq.currentSize >= 2 :
        left = pq.delMin()                           #取出两个最小的元素，
        right = pq.delMin()                          #组成一个新的元素插入优先队列
        pq.insert([(None, left[0][1]+right[0][1]), left, right])
    else:
        return pq.delMin()                           #返回唯一的元素

def _printCode(haff, path):
    if haff[1]==[] and haff[2]==[]:
        #字符所在的叶节点
        print(haff[0], ''.join(path))
    else: #修改path，对左右子树进行递归调用
        _printCode(haff[1], path+['0'])
        _printCode(haff[2], path+['1'])

def printCode(haff):
    curr = haff
    path = []    #用一个栈来保存路径
    _printCode(curr, path)

if __name__ == "__main__":
    l = [("林",1),("妹",2),("哭",6),("了",1)]
    haff = haffmanTree(l)
    print(haff)
    printCode(haff)
```

二叉搜索树的实现

In [4]:
```
class TreeNode:
    def __init__(self, key, val, left=None,
                 right=None, parent=None):
        self.key = key
        self.payload = val
        self.leftChild = left
        self.rightChild = right
```

```
        self.parent = parent              #增加了对parent的指回
        self.balanceFactor = 0            #在BST中用不到，也没什么坏处。

    def hasLeftChild(self):               #这个函数实际没必要，只是为了代
码的讲解更加口语化。
        return self.leftChild

    def hasRightChild(self):
        return self.rightChild

    def isLeftChild(self):
        return self.parent and self.parent.leftChild == self

    def isRightChild(self):
        return self.parent and self.parent.rightChild == self

    def isRoot(self):
        return not self.parent

    def hasAnyChildren(self):
        return self.rightChild or self.leftChild

    def hasBothChildren(self):
        return self.rightChild and self.leftChild

    def isLeaf(self):
        return not self.hasAnyChildren()

    def flat(self):
        flat = lambda x: flat(x.leftChild)+[x.key]+flat(x.rightChil
d) if x else []
        return flat(self)

    def replaceNodeData(self, key, value, lc, rc):
        self.key = key
        self.payload = value
        self.leftChild = lc
        self.rightChild = rc
        if self.hasLeftChild():           #让左右子节点指回父节点
            self.leftChild.parent = self
        if self.hasRightChild():
            self.rightChild.parent = self

    def findSuccessor(self):
        if self.hasRightChild():          #在BinarySearchTree.remove
()中的唯一可能
            return self.rightChild.findMin()
        elif self.isRoot():
            return None
        elif self.isLeftChild():
            return self.parent
        else : #self.isRightChild()       #在parent子树中的最末
            self.parent.rightChild = None #暂时移除自己
```

```
            succ = self.parent.findSuccessor()
            self.parent.rightChild = self #恢复自己
            return succ

    def findMin(self):
        current = self
        while current.hasLeftChild():      #往左下角
            current = current.leftChild
        return current

    def __iter__(self):
        if self:
            if self.hasLeftChild():
                for elem in self.leftChild:
                    yield elem
            yield self.key
            if self.hasRightChild():
                for elem in self.rightChild:
                    yield elem

class BinarySearchTree:                 #函数太多了，增加了“树”类来管理与之有关的部分
    def __init__(self):
        self.root = None
        self.size = 0
    def length(self):
        return self.size
    def __len__(self):                  #供len(bst)调用
        return self.size
    def __iter__(self):
        return self.root.__iter__()

    def display(self):
        print_t(self.root)

    def put(self,key,val=0):
        if self.root:
            self.size += self._put(key,val,self.root)
        else:
            self.root = TreeNode(key, val)
            self.size += 1

    def _put(self, key, val, currentNode):
        if key < currentNode.key:           #递归左子树
            if currentNode.hasLeftChild():
                return self._put(key, val, currentNode.leftChild)
            else:
                currentNode.leftChild = \
                    TreeNode(key, val, parent=currentNode)
                return 1
        elif key > currentNode.key:         #递归右子树
            if currentNode.hasRightChild():
                return self._put(key, val, currentNode.rightChild)
```

```
            else:
                currentNode.rightChild = \
                    TreeNode(key, val, parent=currentNode)
                return 1
        else: #key == currentNode.key         #出现重复key
            currentNode.payload = val
            return 0

    def __setitem__(self, k, v):
        self.put(k,v)

    def get(self, key):
        if self.root:                #这里的判断和_get()内部的判断重复了
            res = self._get(key, self.root)
            if res:
                return res.payload
            else:
                return None
        else:
            return None

    def _get(self, key, currentNode):
        if not currentNode:
            return None
        elif currentNode.key == key:
            return currentNode
        elif currentNode.key > key:
            return self._get(key, currentNode.leftChild)
        else: #currentNode.key < key
            return self._get(key, currentNode.rightChild)

    def __getitem__(self, key):
        return self.get(key)

    def __contains__(self, key):
        if self._get(key, self.root):
            return True
        else:
            return False

    def delete(self, key):
        if self.size > 1:
            nodeToRemove = self._get(key, self.root)
            if nodeToRemove:
                self.remove(nodeToRemove)
                self.size = self.size - 1
            else:
                raise KeyError('Error, key not in tree')
        elif self.size == 1 and self.root.key == key:
            self.root = None           #删除根节点
            self.size = self.size - 1
        else:
            raise KeyError('Error, key not in tree')
```

```
    def __delitem__(self, key):
        self.delete(key)

    def remove(self, currentNode):
        if currentNode.isLeaf():                    #作为叶节点的最简单情况
            #移除叶节点只需要在它的父节点中把它移除
            if currentNode.isLeftChild():
                currentNode.parent.leftChild = None
            elif currentNode.isRightChild():
                currentNode.parent.rightChild = None
        elif currentNode.hasBothChildren():     #有左右两个子节点的复杂情况
            succ = currentNode.findSuccessor() #前驱、后继肯定都有
            currentNode.key = succ.key
            currentNode.payload = succ.payload
            self.remove(succ)                       #succ没有左子节点，为什么？不会出现第二层递归
        else:                                       #只有一个子节点的情况
            # 取得唯一子节点，不关心左右
            child = currentNode.leftChild \
                    if currentNode.hasLeftChild() \
                    else currentNode.rightChild  #可以砍掉一半的源代码
            if currentNode.isLeftChild():      #用子节点替换当前节点
                currentNode.parent.leftChild = child
                child.parent = currentNode.parent
            elif currentNode.isRightChild():
                currentNode.parent.rightChild = child
                child.parent = currentNode.parent
            else :                                  #当前节点是根节点，直接替换成子节点
                currentNode.replaceNodeData(child.key,
                                            child.payload,
                                            child.leftChild,
                                            child.rightChild)


if __name__ == "__main__":
    bst = BinarySearchTree()
    bst.put(70)
    bst.put(31)
    bst.put(93)
    bst.put(94)
    bst.put(14)
    bst.put(23)
    bst.put(73)
    print(bst.root.flat())
    print([x for x in bst])
    bst.display()
```

```
[14, 23, 31, 70, 73, 93, 94]
[14, 23, 31, 70, 73, 93, 94]
                  70
              +----+---------+
             31              93
  +----------+          +----+----+
 14                    73         94
  +----+
      23
```

二叉搜索树的测试代码

In [ ]:
```
mytree = BinarySearchTree()
mytree[3]="red"
mytree[4]="blue"
mytree[6]="yellow"
mytree[2]="at"

print_t(mytree.root)
print(3 in mytree)
print(mytree[6])
del mytree[3]
print(mytree[2])
for key in mytree:
    print(key, mytree[key])
```

AVL作为BST的子类实现：

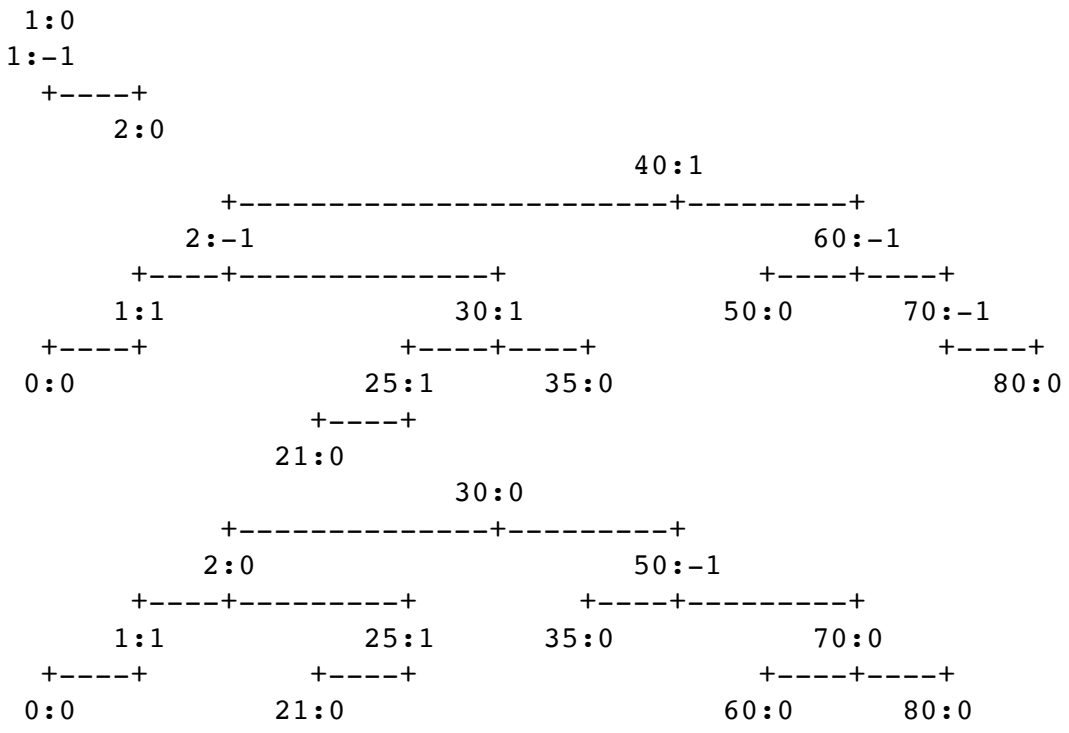
In [5]:
```
class AVL1(BinarySearchTree):
    def _put(self, key, val, currentNode):
        if key < currentNode.key:
            if currentNode.hasLeftChild():
                return self._put(key, val, currentNode.leftChild)
            else:
                currentNode.leftChild = TreeNode(key, val, parent=c
urrentNode)
                self.updateBalance(currentNode.leftChild)  #调整平衡
因子
                return 1
        elif key > currentNode.key:
            if currentNode.hasRightChild():
                return self._put(key, val, currentNode.rightChild)
            else:
                currentNode.rightChild = TreeNode(key, val, parent=
currentNode)
                self.updateBalance(currentNode.rightChild)  #调整平
衡因子
                return 1
```

```
        else: #key == currentNode.key
            currentNode.payload = val          #无新增节点，平衡因子不变
            return 0

    def updateBalance(self, node):
        if node.balanceFactor > 1 or node.balanceFactor < -1: #先看
自己是否要调整
            self.rebalance(node)                          #重新平衡，并
且不会向上传递
            return
        if node.parent != None:                           #更新父节
点平衡因子
            if node.isLeftChild():
                node.parent.balanceFactor += 1
            elif node.isRightChild():
                node.parent.balanceFactor -= 1
            if node.parent.balanceFactor != 0:            #调整父节
点平衡因子
                self.updateBalance(node.parent)           #"=0"也会
阻断递归的传递

    def rotateLeft(self, rotRoot):
        newRoot = rotRoot.rightChild                      #把新根节点
提上来
        rotRoot.rightChild = newRoot.leftChild            #给新根节点
的左子节点重新找位置
        if newRoot.leftChild != None:
            newRoot.leftChild.parent = rotRoot            #    并给它
指定新的parent
        newRoot.parent = rotRoot.parent                   #新根节点完
全取代旧根节点
        if rotRoot.isRoot():
            self.root = newRoot
        else:
            if rotRoot.isLeftChild():
                rotRoot.parent.leftChild = newRoot
            else:
                rotRoot.parent.rightChild = newRoot
        newRoot.leftChild = rotRoot
        rotRoot.parent = newRoot
        #调整新、旧根节点的平衡因子，为什么这样？马上推导。
        rotRoot.balanceFactor = rotRoot.balanceFactor + \
                                1 - min(newRoot.balanceFactor, 0)
        newRoot.balanceFactor = newRoot.balanceFactor + \
                                1 + max(rotRoot.balanceFactor, 0)

    def rotateRight(self, rotRoot):
        newRoot = rotRoot.leftChild
        rotRoot.leftChild = newRoot.rightChild    #56，58新根的右子转
为旧根的左子
        if newRoot.rightChild != None:
            newRoot.rightChild.parent = rotRoot
        newRoot.parent = rotRoot.parent           #59，64/66新根取代
```

```
旧根和父建立关系
        if rotRoot.isRoot():
            self.root = newRoot
        else:
            if rotRoot.isLeftChild():
                rotRoot.parent.leftChild = newRoot
            else:
                rotRoot.parent.rightChild = newRoot
        newRoot.rightChild = rotRoot                #67，68旧根下沉为新
根的子节点
        rotRoot.parent = newRoot
        #调整新、旧根节点的平衡因子，为什么这样？马上推导。
        rotRoot.balanceFactor = rotRoot.balanceFactor - \
                                1 - max(newRoot.balanceFactor, 0)
        newRoot.balanceFactor = newRoot.balanceFactor - \
                                1 + min(rotRoot.balanceFactor, 0)

    def rebalance(self, node):    #"<-1"或">1"才会被updateBalance调用
        if node.balanceFactor < -1:  #右重需要左旋
            if node.rightChild.balanceFactor > 0:
                # 右子节点左重，先对它进行一次右旋
                self.rotateRight(node.rightChild)
            #正常左旋
            self.rotateLeft(node)
        elif node.balanceFactor > 1:
            if node.leftChild.balanceFactor < 0:
                self.rotateLeft(node.leftChild)
            self.rotateRight(node)

    def display(self):
        print_t(self.root, label=lambda x:"{}:{}".format(x.key, x.b
alanceFactor))
```

AVL的测试代码

In [41]:
```
if __name__ == "__main__":
    AVL = AVL2
    tree = AVL()

    tree.put(1)
    tree.display()
    tree.put(2)
    tree.display()
    tree.put(30)
    tree.put(40)
    tree.put(50)
    tree.put(60)
    tree.put(70)
    tree.put(80)
    tree.put(0)
    tree.put(25)
    tree.put(35)
    tree.put(21)
    tree.display()
    del tree[40]
    tree.display()
    """
    command = input().split()
    while command[0] != 'exit':
        if command[0] == 'put':
            tree.put(int(command[1]))
        elif command[0] == 'delete':
            tree.delete(int(command[1]))
        elif command[0] == 'display':
            tree.display()
        command = input().split()
    """
```



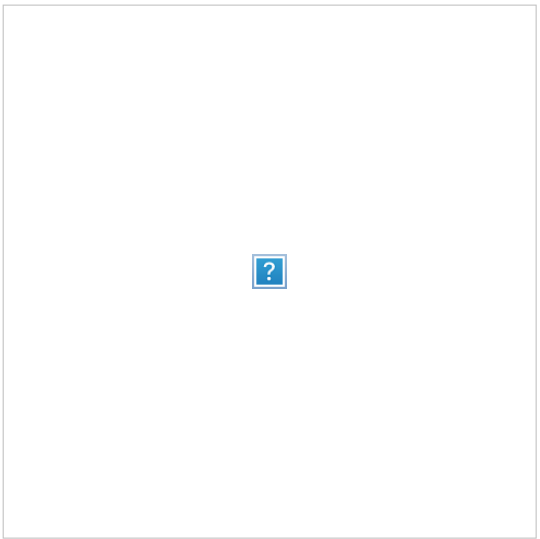
邻接列表的图实现

In [34]:
```
class Vertex1:
    def __init__(self, key):
        self.id = key
        self.connectedTo = {}

    def addNeighbor(self, nbr:Vertex, weight=0):
        self.connectedTo[nbr] = weight

    def __str__(self):
        return str(self.id) + ' connectedTo: ' \
            + str([x.id for x in self.connectedTo])

    __repr__ = __str__

    def getConnections(self):
        return self.connectedTo.keys()

    def getId(self):
        return self.id

    def getWeight(self, nbr):
        return self.connectedTo[nbr]

Vertex = Vertex1

class Graph:
    def __init__(self):
        self.vertList = {}     #名不符实
        self.numVertices = 0

    def addVertex(self, key):
        self.numVertices += 1
        newVertex = Vertex(key)
        self.vertList[key] = newVertex
        return newVertex

    def getVertex(self, key):
        try:
            return self.vertList[key]
        except KeyError:
            return None

    def __contains__(self, key):
        return key in self.vertList

    def addEdge(self, f, t, weight=0):
        if f not in self:
            self.addVertex(f)
```

```
        if t not in self:
            self.addVertex(t)
        self.vertList[f].addNeighbor(self.vertList[t], weight)

    def getVertices(self):
        return self.vertList.keys()

    def __iter__(self):
        return iter(self.vertList.values())

if __name__=="__main__":
    g = Graph()
    for i in range(6):
        g.addVertex(i)
    for v in g:
        print(v)
    print(g.vertList)
```

```
0 connectedTo: []
1 connectedTo: []
2 connectedTo: []
3 connectedTo: []
4 connectedTo: []
5 connectedTo: []
{0: 0 connectedTo: [], 1: 1 connectedTo: [], 2: 2 connectedTo: [],
3: 3 connectedTo: [], 4: 4 connectedTo: [], 5: 5 connectedTo: []}
```

Graph使用示例



In [22]:
```
if __name__ == "__main__":
    g = Graph()
    edges = [('V0', 'V1', 5), ('V0', 'V5', 2),
             ('V1', 'V2', 4),
            ('V2', 'V3', 9),
            ('V3', 'V4', 7), ('V3', 'V5', 3),
            ('V4', 'V0', 1),
            ('V5', 'V2', 1), ('V5', 'V4', 8)
            ]
    for e in edges:
        g.addEdge(*e)
    for v in g:
        for w in v.getConnections():
            print("{}->{}".format(v.getId(), w.getId()))
```

```
V0->V1
V0->V5
V1->V2
V5->V2
V5->V4
V2->V3
V3->V4
V3->V5
V4->V0
```

词梯问题

In [29]:
```
#### 把词语之间"只差一个字符"的关系转化为图
def buildGraph(wordFile):
    d = {}
    g = Graph()
    with open(wordFile, 'r') as f:
        for line in f:
            word = line[:-1]
            # create buckets of words that differ by one letter
            for i in range(len(word)):
                bucket = word[:i]+'_'+word[i+1:]
                if bucket in d:
                    d[bucket].append(word)
                else:
                    d[bucket] = [word]

    # add vertices and edges for words in the same bucket
    for bucket in d.keys():
        for word1 in d[bucket]:
            for word2 in d[bucket]:
                if word1 != word2:
                    g.addEdge(word1, word2)
    return g

if __name__ == "__main__":
    g = buildGraph("code/vocabulary.txt")
    for i, v in enumerate(g):
        print(v)
        if i>20:                          #为了演示方便，限制一下输出数量
            break;
    print(sum([len(v.connectedTo) for v in g]))
```

```
AAHS connectedTo: ['DAHS', 'FAHS', 'HAHS', 'LAHS', 'AALS']
DAHS connectedTo: ['AAHS', 'FAHS', 'HAHS', 'LAHS', 'DABS', 'DADS',
'DAGS', 'DAIS', 'DAKS', 'DALS', 'DAMS', 'DAPS', 'DAWS', 'DAYS', 'D
AHL', 'DOHS']
FAHS connectedTo: ['AAHS', 'DAHS', 'HAHS', 'LAHS', 'FADS', 'FANS',
'FATS', 'FAYS', 'FEHS']
HAHS connectedTo: ['AAHS', 'DAHS', 'FAHS', 'LAHS', 'HAES', 'HAGS',
'HAMS', 'HAPS', 'HATS', 'HAWS', 'HAYS', 'HAHA', 'HEHS']
LAHS connectedTo: ['AAHS', 'DAHS', 'FAHS', 'HAHS', 'LABS', 'LACS',
'LADS', 'LAGS', 'LAMS', 'LAPS', 'LARS', 'LASS', 'LATS', 'LAVS', 'L
AWS', 'LAYS']
AALS connectedTo: ['AAHS', 'BALS', 'DALS', 'GALS', 'PALS', 'SALS',
'AILS', 'ALLS', 'AWLS']
BALS connectedTo: ['AALS', 'DALS', 'GALS', 'PALS', 'SALS', 'BAAS',
'BADS', 'BAGS', 'BAMS', 'BANS', 'BAPS', 'BARS', 'BASS', 'BATS', 'B
AYS', 'BALD', 'BALE', 'BALK', 'BALL', 'BALM', 'BELS']
DALS connectedTo: ['AALS', 'BALS', 'GALS', 'PALS', 'SALS', 'DABS',
'DADS', 'DAGS', 'DAHS', 'DAIS', 'DAKS', 'DAMS', 'DAPS', 'DAWS', 'D
AYS', 'DALE', 'DELS', 'DOLS']
GALS connectedTo: ['AALS', 'BALS', 'DALS', 'PALS', 'SALS', 'GABS',
'GADS', 'GAES', 'GAGS', 'GAMS', 'GAPS', 'GARS', 'GATS', 'GAYS', 'G
ALA', 'GALE', 'GALL', 'GELS', 'GULS']
PALS connectedTo: ['AALS', 'BALS', 'DALS', 'GALS', 'SALS', 'PACS',
'PADS', 'PAKS', 'PAMS', 'PANS', 'PAPS', 'PARS', 'PASS', 'PATS', 'P
AWS', 'PAYS', 'PALE', 'PALL', 'PALM', 'PALP', 'PALY', 'POLS', 'PUL
S']
SALS connectedTo: ['AALS', 'BALS', 'DALS', 'GALS', 'PALS', 'SABS',
'SACS', 'SAGS', 'SANS', 'SAPS', 'SASS', 'SAWS', 'SAYS', 'SALE', 'S
ALL', 'SALP', 'SALT', 'SELS', 'SOLS']
AILS connectedTo: ['AALS', 'ALLS', 'AWLS', 'AIDS', 'AIMS', 'AINS',
'AIRS', 'AITS', 'FILS', 'MILS', 'NILS', 'OILS', 'TILS']
ALLS connectedTo: ['AALS', 'AILS', 'AWLS', 'ALAS', 'ALBS', 'ALES',
'ALMS', 'ALPS', 'ALTS', 'ELLS', 'ILLS', 'ALLY']
AWLS connectedTo: ['AALS', 'AILS', 'ALLS', 'AWES', 'AWNS', 'OWLS']
ABAS connectedTo: ['AGAS', 'ALAS', 'AMAS', 'ANAS', 'ABOS', 'ABYS']
AGAS connectedTo: ['ABAS', 'ALAS', 'AMAS', 'ANAS', 'AGAR', 'AGES']
ALAS connectedTo: ['ABAS', 'AGAS', 'AMAS', 'ANAS', 'ALAE', 'ALAN',
'ALAR', 'ALBS', 'ALES', 'ALLS', 'ALMS', 'ALPS', 'ALTS']
AMAS connectedTo: ['ABAS', 'AGAS', 'ALAS', 'ANAS', 'AMAH', 'AMIS',
'AMPS', 'AMUS']
ANAS connectedTo: ['ABAS', 'AGAS', 'ALAS', 'AMAS', 'ANAL', 'ANDS',
'ANES', 'ANIS', 'ANTS', 'ANUS']
ABOS connectedTo: ['ABAS', 'ABYS', 'ADOS', 'AVOS']
ABYS connectedTo: ['ABAS', 'ABOS', 'ABYE']
ABBA connectedTo: ['ALBA', 'ABBE']
42600
```

In [50]:
```
def bfs(g, start):
    start.setDistance(0)
    start.setPred(None)
    vertQueue = Queue()
    vertQueue.enqueue(start)
    while (vertQueue.size() > 0):
        currentVert = vertQueue.dequeue()
        for nbr in currentVert.getConnections():
            if (nbr.getColor() == 'white'):
                nbr.setColor('gray')
                nbr.setDistance(currentVert.getDistance() + 1)
                nbr.setPred(currentVert)
                vertQueue.enqueue(nbr)
        currentVert.setColor('black')
```

In [57]:
```
class Vertex2(Vertex1):
    def __init__(self, key):
        self.id = key
        self.connectedTo = {}
        self.color = 'white'

    def getDistance(self):
        return self.distance

    def setDistance(self, dist):
        self.distance = dist

    def setPred(self, pred):
        self.pred = pred

    def getPred(self):
        return self.pred

    def getColor(self):
        return self.color

    def setColor(self, color):
        self.color = color

Vertex = Vertex2

Queue = Queue_on_list

def traverse(y):
    x = y
    while (x.getPred()):
        print(x.getId())
        x = x.getPred()
    print(x.getId())

if __name__ == "__main__":
    g = buildGraph("code/vocabulary.txt")
    bfs(g, g.getVertex('FOOL'))
    word = input("traverse:")
    target = g.getVertex(word)
    if target:
        traverse(target)
    else:
        print("'{}' not found".format(word))
    #print(len(g.getVertices()))
    #print(g.vertices)
```

```
'FAil' not found
```