课程实践 20181119实习四：习题集p136 4.1

郭宸 17081511

一、需求分析

稀疏矩阵是指那些多数元素为零的矩阵。利用“稀疏”特点进行存储和计算可以大大节省存储空间，提高计算效率。实现一个能进行稀疏矩阵基本运算的运算器。 以“带行逻辑链接信息”的三元组顺序表表示稀疏矩阵，实现两个矩阵相加、相减和相乘的运算，稀疏矩阵的输入形式采用三元组表示，而运算结果的矩阵则以通常的阵列形式 列出。

二、概要设计

```
import numpy as np
#python常用的矩阵运算工具包。本代码中不使用其已经封装好的矩阵运算函数。
from operator import itemgetter
#排序时使用
class sparse_matrix(object):
#创建稀疏矩阵类，内置了一些参数及函数
    def __init__(self):
        self.data = []
        self.rpos = []
        self.mu = 0
        self.nu = 0
        self.tu = 0
        self.juzhen = [[]]
    #初始化
    def RLSM(self,s):
    #由列表s对稀疏矩阵类中的一些参数进行初始化
    def sortdata(self):
    #对对象自己的data进行按照先行后列的排序
    def RLSMself(self):
    #当对象只存在data或只存在juzhen时，对其他参数进行赋值
def addSMatrix(M,N):
#矩阵加法运算
def subtMatrix(M,N):
#矩阵减法运算
def searchrpos(Q,i):
#返回稀疏矩阵某一行非零元素的个数
def multSMatrix(M,N):
#矩阵乘法运算
def printMatrix(Q):
#以矩阵的形式输出打印稀疏矩阵
def str_to_tuple(str):
#将input读入的字符串转换为元组(tuple)
```

三、详细设计

xishujuzhen.py

```
import numpy as np
from operator import itemgetter
class sparse_matrix(object):
    def __init__(self):
        self.data = []
        self.rpos = []
        self.mu = 0
        self.nu = 0
        self.tu = 0
        self.juzhen = [[]]
    def RLSM(self,s):
        data1 = []
        for m in s:
            data1.append(m)
        data = sorted(data1,key=itemgetter(0,1),reverse=False)
        #print(data)
        self.data = data
        n = 0
        mu = 1
        nu = 1
        for i in range(0,len(data)):
            if (data[i][0] > mu):
                mu = data[i][0]
            if (data[i][1] > nu): nu = data[i][1]
            n += 1
        self.mu = mu
        self.nu = nu
        self.tu = n
        rpos = np.zeros(mu+1,dtype=int)
        juzhen = np.zeros((mu+1,nu+1))
        k = 0
        for d in data:
            # print(d)
            t = d[0]
            k += 1
            if (rpos[t] == 0):
                rpos[t] = k
            juzhen[d[0]][d[1]] = d[2]
        self.rpos = rpos

        self.juzhen = juzhen
    def sortdata(self):
        self.data = sorted(self.data, key=itemgetter(0, 1), reverse=False)
    def RLSMself(self):
        if(self.data != []):
            data = self.data
            n = 0
            mu = 1
            nu = 1
            for i in range(0, len(data)):
                if (data[i][0] > mu):
```

```
                    mu = data[i][0]
                if (data[i][1] > nu): nu = data[i][1]
                n += 1
            self.mu = mu
            self.nu = nu
            self.tu = n
            rpos = np.zeros(mu + 1,dtype=int)
            juzhen = np.zeros((mu + 1, nu + 1))
            k = 0
            for d in data:
                # print(d)
                t = d[0]
                k += 1
                if (rpos[t] == 0):
                    rpos[t] = k
                juzhen[d[0]][d[1]] = d[2]
            self.rpos = rpos
            self.juzhen = juzhen
        elif(self.juzhen != [[]]):
            self.mu = self.juzhen.shape[0] - 1
            self.nu = self.juzhen.shape[1] - 1
            for i in range(self.mu):
                for j in range(self.nu):
                    if(self.juzhen[i][j] != 0):
                        self.data.append((i,j,self.juzhen[i][j]))
def addSMatrix(M,N):
    Q = sparse_matrix()
    if(M.mu != N.mu or M.nu != N.nu): return "Error"
    else:
        for i in range (len(M.data)-1):
            for mdata in M.data:
                for ndata in N.data:
                    if (mdata[0] == ndata[0] and mdata[1] == ndata[1]):
                        Q.data.append((mdata[0], mdata[1], mdata[2] + ndata[2]))
                        M.data.remove(mdata)
                        N.data.remove(ndata)
                        break
        for mdata in M.data:
            Q.data.append(mdata)
        for ndata in N.data:
            Q.data.append(ndata)
    Q.RLSMself()
    return Q
def subtMatrix(M,N):
    Q = sparse_matrix()
    if (M.mu != N.mu or M.nu != N.nu):
        return "Error"
    else:
        for i in range (len(M.data)-1):
            for mdata in M.data:
                for ndata in N.data:
                    if (mdata[0] == ndata[0] and mdata[1] == ndata[1]):
                        Q.data.append((mdata[0], mdata[1], mdata[2] - ndata[2]))
```

```
                        M.data.remove(mdata)
                        N.data.remove(ndata)
                        break
        for mdata in M.data:
            Q.data.append(mdata)
        for ndata in N.data:
            Q.data.append((ndata[0],ndata[1],-ndata[2]))
    Q.RLSMself()
    return Q
def searchrpos(Q,i):
    if(i < Q.mu):
        if(Q.rpos[i+1] != 0):
            return Q.rpos[i+1] - Q.rpos[i]
        else:
            for j in range(i+1,Q.mu+1):
                if(Q.rpos[j] == 0):
                    continue
                else:
                    return Q.rpos[j] - Q.rpos[i]
    else:
        return Q.tu - Q.rpos[i] + 1
def multSMatrix(M,N):
    if(M.nu != N.mu): return "Error"
    else:
        Q = sparse_matrix()
        juzhen = np.zeros((M.mu+1,N.nu+1))
        for mdata in M.data:
            t = searchrpos(N,mdata[1])
            for i in range(N.rpos[mdata[1]],N.rpos[mdata[1]] + t):
                juzhen[mdata[0]][N.data[i-1][1]] += N.data[i-1][2]*mdata[2]
        Q.juzhen = juzhen
        Q.RLSMself()
        return Q
def printMatrix(Q):
    for i in range(1,Q.mu+1):
        if(i == 1):print(" ⌈",end="")
        else: print("|",end="")
        for j in range(1,Q.nu+1):
            print(Q.juzhen[i][j],end=" ")
        if(i == Q.mu): print("⌋ ")
        else: print("|")
```

xishujuzhentest.py

```
import xishujuzhen as xsjz
from xishujuzhen import sparse_matrix as sm
def str_to_tuple(str):
    tupleresult=[]
    t = str.split('),(')
    for r in t:
        temp = r.replace('(', '').replace(')', '')
        a = tuple([int(i) for i in temp.split(',')])
```

```
        #print(a)
        tupleresult.append(a)
    return tuple(tupleresult)
o = '#'
while(o != 'e'):
    print("矩阵加法：1")
    print("矩阵减法：2")
    print("矩阵乘法：3")
    print("退出：e")
    o = input("请选择将要进行的矩阵运算")
    if (o == '1'):
        a = input("以三元组形式输入第一个矩阵，回车结束")
        b = input("以三元组形式输入第二个矩阵，回车结束")
        A = sm()
        B = sm()
        # print(str_to_tuple(a)))
        # print(str_to_tuple(b)))
        A.RLSM(str_to_tuple(a))
        B.RLSM(str_to_tuple(b))
        Q = xsjz.addSMatrix(A, B)
        print("运算结果为：————————————————————————————————————————")
        xsjz.printMatrix(Q)
        print("————————————————————————————————————————————————————")
    if (o == '2'):
        a = input("以三元组形式输入被减矩阵，回车结束")
        b = input("以三元组形式输入减矩阵，回车结束")
        A = sm()
        B = sm()
        A.RLSM(str_to_tuple(a))
        B.RLSM(str_to_tuple(b))
        Q = xsjz.subtMatrix(A, B)
        print("运算结果为：————————————————————————————————————————")
        xsjz.printMatrix(Q)
        print("————————————————————————————————————————————————————")
    if (o == '3'):
        a = input("以三元组形式输入第一个矩阵，回车结束")
        b = input("以三元组形式输入第二个矩阵，回车结束")
        A = sm()
        B = sm()
        A.RLSM(str_to_tuple(a))
        B.RLSM(str_to_tuple(b))
        Q = xsjz.multSMatrix(A, B)
        print("运算结果为：————————————————————————————————————————")
        xsjz.printMatrix(Q)
        print("————————————————————————————————————————————————————")
'''
(1,1,10),(2,3,9),(3,1,-1)
(2,3,-1),(3,1,1),(3,3,-3)
(1,1,10),(2,2,9),(3,1,-1)
(2,2,-1),(3,1,1),(3,2,-3)
(1,1,4),(1,2,-3),(1,5,1),(2,4,8),(3,3,1),(4,5,70)
(1,1,3),(2,1,4),(2,2,2),(3,2,1),(4,1,1),(5,3,0)
'''
```

四、调试分析

先输入将要进行的矩阵运算，然后使用input读入矩阵（为字符串类型）。输入矩阵时，要以三元组的形式输入

矩阵加法是直接用data列表进行运算的，不需要对两个矩阵中都为非零元素的参数进行相加运算，减法同理

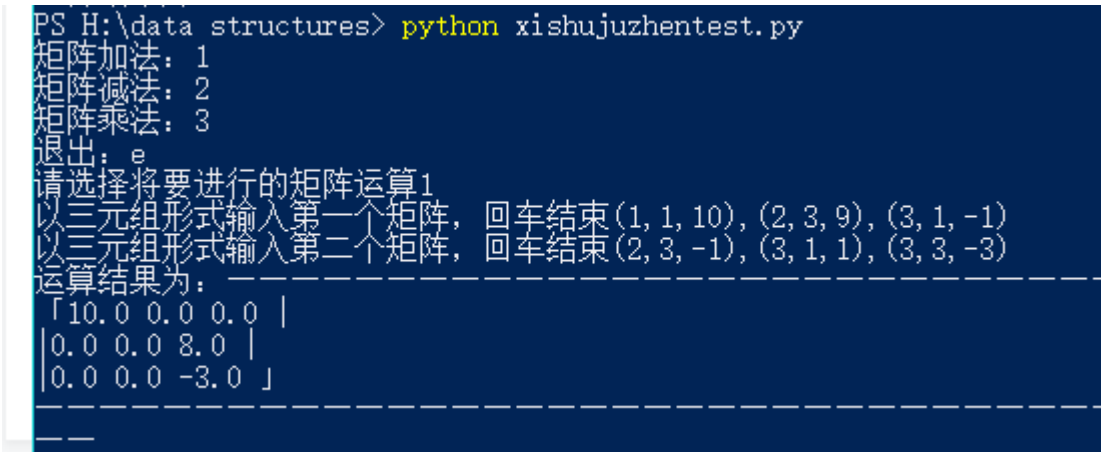
乘法运算仿照课本的伪代码进行实现

五、用户手册

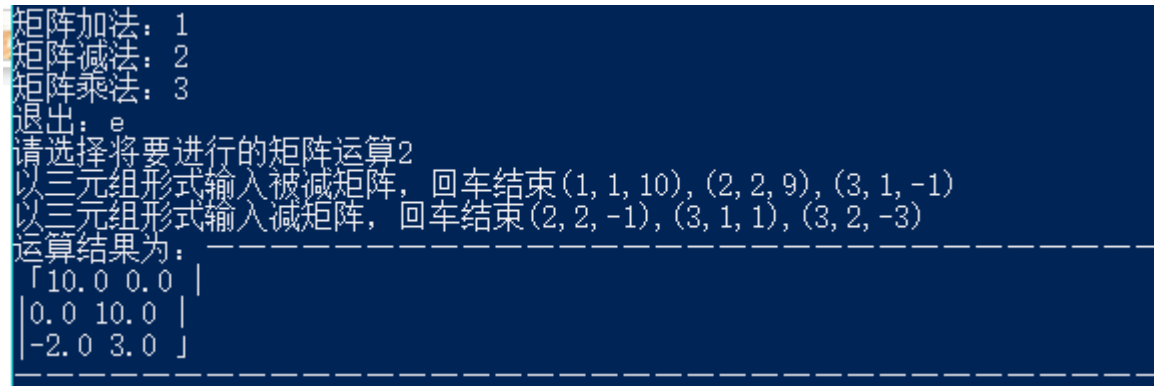
本程序在安装了python3和的所有系统上均可运行。

六、测试结果

1.(1,1,10),(2,3,9),(3,1,-1)与(2,3,-1),(3,1,1),(3,3,-3)相加



2.(1,1,10),(2,2,9),(3,1,-1)与(2,2,-1),(3,1,1),(3,2,-3)相减



3.(1,1,4),(1,2,-3),(1,5,1),(2,4,8),(3,3,1),(4,5,70)与(1,1,3),(2,1,4),(2,2,2),(3,2,1),(4,1,1),(5,3,0)相乘

七、附录

源程序文件名清单：

xishujuzhen.py #主要的类与函数均定义在该程序里

xishujuzhentest.py #测试程序