

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

```
1  import numpy as np
2  #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。
3  from operator import itemgetter
4  #排序时使用
5  class sparse_matrix(object):
6  #创建稀疏矩阵类，内置了一些参数及函数
7      def __init__(self):
8          self.data = []
9          self.rpos = []
10         self.mu = 0
11         self.nu = 0
12         self.tu = 0
13         self.juzhen = [[]]
14         #初始化
15         def RLSM(self,s):
16             #由列表s对稀疏矩阵类中的一些参数进行初始化
17         def sortdata(self):
18             #对对象自己的data进行按照先行后列的排序
19         def RLSMself(self):
20             #当对象只存在data或只存在juzhen时，对其他参数进行赋值
21     def addMatrix(M,N):
22         #矩阵加法运算
23     def subMatrix(M,N):
24         #矩阵减法运算
25     def searchrpos(Q,i):
26         #返回稀疏矩阵某一行非零元素的个数
27     def multMatrix(M,N):
28         #矩阵乘法运算
29     def printMatrix(Q):
30         #以矩阵的形式输出打印稀疏矩阵
31     def str_to_tuple(str):
32         #将input读入的字符串转换为元组(tuple)
```

三、详细设计

xishujuzhen.py

```
1  import numpy as np
2  from operator import itemgetter
3  class sparse_matrix(object):
4      def __init__(self):
5          self.data = []
6          self.rpos = []
7          self.mu = 0
8          self.nu = 0
9          self.tu = 0
10         self.juzhen = [[]]
11     def RLSM(self,s):
12         data1 = []
13         for m in s:
14             data1.append(m)
15         data = sorted(data1,key=itemgetter(0,1),reverse=False)
16         #print(data)
17         self.data = data
18         n = 0
19         mu = 1
20         nu = 1
21         for i in range(0,len(data)):
22             if (data[i][0] > mu):
23                 mu = data[i][0]
24             if (data[i][1] > nu): nu = data[i][1]
25             n += 1
26         self.mu = mu
27         self.nu = nu
28         self.tu = n
29         rpos = np.zeros(mu+1,dtype=int)
30         juzhen = np.zeros((mu+1,nu+1))
31         k = 0
32         for d in data:
33             # print(d)
34             t = d[0]
35             k += 1
36             if (rpos[t] == 0):
37                 rpos[t] = k
38             juzhen[d[0]][d[1]] = d[2]
39         self.rpos = rpos
40
41         self.juzhen = juzhen
42     def sortdata(self):
43         self.data = sorted(self.data, key=itemgetter(0, 1), reverse=False)
44     def RLSMself(self):
45         if(self.data != []):
46             data = self.data
47             n = 0
48             mu = 1
49             nu = 1
50             for i in range(0, len(data)):
51                 if (data[i][0] > mu):
```

```

52         mu = data[i][0]
53         if (data[i][1] > nu): nu = data[i][1]
54         n += 1
55     self.mu = mu
56     self.nu = nu
57     self.tu = n
58     rpos = np.zeros(mu + 1, dtype=int)
59     juzhen = np.zeros((mu + 1, nu + 1))
60     k = 0
61     for d in data:
62         # print(d)
63         t = d[0]
64         k += 1
65         if (rpos[t] == 0):
66             rpos[t] = k
67         juzhen[d[0]][d[1]] = d[2]
68     self.rpos = rpos
69     self.juzhen = juzhen
70     elif(self.juzhen != [[]]):
71         self.mu = self.juzhen.shape[0] - 1
72         self.nu = self.juzhen.shape[1] - 1
73         for i in range(self.mu):
74             for j in range(self.nu):
75                 if(self.juzhen[i][j] != 0):
76                     self.data.append((i, j, self.juzhen[i][j]))
77 def addMatrix(M, N):
78     Q = sparse_matrix()
79     if(M.mu != N.mu or M.nu != N.nu): return "Error"
80     else:
81         for i in range (len(M.data)-1):
82             for mdata in M.data:
83                 for ndata in N.data:
84                     if (mdata[0] == ndata[0] and mdata[1] == ndata[1]):
85                         Q.data.append((mdata[0], mdata[1], mdata[2] + ndata[2]))
86                         M.data.remove(mdata)
87                         N.data.remove(ndata)
88                     break
89         for mdata in M.data:
90             Q.data.append(mdata)
91         for ndata in N.data:
92             Q.data.append(ndata)
93     Q.RLSMself()
94     return Q
95 def subMatrix(M, N):
96     Q = sparse_matrix()
97     if (M.mu != N.mu or M.nu != N.nu):
98         return "Error"
99     else:
100         for i in range (len(M.data)-1):
101             for mdata in M.data:
102                 for ndata in N.data:
103                     if (mdata[0] == ndata[0] and mdata[1] == ndata[1]):
104                         Q.data.append((mdata[0], mdata[1], mdata[2] - ndata[2]))

```

```

105         M.data.remove(mdata)
106         N.data.remove(ndata)
107         break
108     for mdata in M.data:
109         Q.data.append(mdata)
110     for ndata in N.data:
111         Q.data.append((ndata[0], ndata[1], -ndata[2]))
112     Q.RLSMself()
113     return Q
114 def searchrpos(Q, i):
115     if(i < Q.mu):
116         if(Q.rpos[i+1] != 0):
117             return Q.rpos[i+1] - Q.rpos[i]
118         else:
119             for j in range(i+1, Q.mu+1):
120                 if(Q.rpos[j] == 0):
121                     continue
122                 else:
123                     return Q.rpos[j] - Q.rpos[i]
124     else:
125         return Q.tu - Q.rpos[i] + 1
126 def multSMatrix(M, N):
127     if(M.nu != N.mu): return "Error"
128     else:
129         Q = sparse_matrix()
130         juzhen = np.zeros((M.mu+1, N.nu+1))
131         for mdata in M.data:
132             t = searchrpos(N, mdata[1])
133             for i in range(N.rpos[mdata[1]], N.rpos[mdata[1]] + t):
134                 juzhen[mdata[0]][N.data[i-1][1]] += N.data[i-1][2]*mdata[2]
135         Q.juzhen = juzhen
136         Q.RLSMself()
137         return Q
138 def printMatrix(Q):
139     for i in range(1, Q.mu+1):
140         if(i == 1): print("[", end="")
141         else: print("|", end="")
142         for j in range(1, Q.nu+1):
143             print(Q.juzhen[i][j], end=" ")
144         if(i == Q.mu): print("]")
145         else: print("|")

```

xishujuzhentest.py

```

1 import xishujuzhen as xsjz
2 from xishujuzhen import sparse_matrix as sm
3 def str_to_tuple(str):
4     tupleresult=[]
5     t = str.split('(',')')
6     for r in t:
7         temp = r.replace('(', '').replace(')', '')
8         a = tuple([int(i) for i in temp.split(',')])

```

```

9         #print(a)
10        tupleresult.append(a)
11    return tuple(tupleresult)
12    o = '#'
13    while(o != 'e'):
14        print("矩阵加法: 1")
15        print("矩阵减法: 2")
16        print("矩阵乘法: 3")
17        print("退出: e")
18        o = input("请选择将要进行的矩阵运算")
19        if (o == '1'):
20            a = input("以三元组形式输入第一个矩阵, 回车结束")
21            b = input("以三元组形式输入第二个矩阵, 回车结束")
22            A = sm()
23            B = sm()
24            # print(str_to_tuple(a))
25            # print(str_to_tuple(b))
26            A.RLSM(str_to_tuple(a))
27            B.RLSM(str_to_tuple(b))
28            Q = xsjz.addSMatrix(A, B)
29            print("运算结果为: -----")
30            xsjz.printMatrix(Q)
31            print("-----")
32        if (o == '2'):
33            a = input("以三元组形式输入被减矩阵, 回车结束")
34            b = input("以三元组形式输入减矩阵, 回车结束")
35            A = sm()
36            B = sm()
37            A.RLSM(str_to_tuple(a))
38            B.RLSM(str_to_tuple(b))
39            Q = xsjz.subtMatrix(A, B)
40            print("运算结果为: -----")
41            xsjz.printMatrix(Q)
42            print("-----")
43        if (o == '3'):
44            a = input("以三元组形式输入第一个矩阵, 回车结束")
45            b = input("以三元组形式输入第二个矩阵, 回车结束")
46            A = sm()
47            B = sm()
48            A.RLSM(str_to_tuple(a))
49            B.RLSM(str_to_tuple(b))
50            Q = xsjz.multSMatrix(A, B)
51            print("运算结果为: -----")
52            xsjz.printMatrix(Q)
53            print("-----")
54    '''
55    (1,1,10), (2,3,9), (3,1,-1)
56    (2,3,-1), (3,1,1), (3,3,-3)
57    (1,1,10), (2,2,9), (3,1,-1)
58    (2,2,-1), (3,1,1), (3,2,-3)
59    (1,1,4), (1,2,-3), (1,5,1), (2,4,8), (3,3,1), (4,5,70)
60    (1,1,3), (2,1,4), (2,2,2), (3,2,1), (4,1,1), (5,3,0)
61    '''

```

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

```
PS H:\data structures> python xishujuzhentest.py
矩阵加法: 1
矩阵减法: 2
矩阵乘法: 3
退出: e
请选择将要进行的矩阵运算1
以三元组形式输入第一个矩阵,回车结束(1,1,10),(2,3,9),(3,1,-1)
以三元组形式输入第二个矩阵,回车结束(2,3,-1),(3,1,1),(3,3,-3)
运算结果为:
┌10.0 0.0 0.0 │
│0.0 0.0 8.0 │
│0.0 0.0 -3.0 │
└───────────┘
```

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

```
矩阵加法: 1
矩阵减法: 2
矩阵乘法: 3
退出: e
请选择将要进行的矩阵运算2
以三元组形式输入被减矩阵,回车结束(1,1,10),(2,2,9),(3,1,-1)
以三元组形式输入减矩阵,回车结束(2,2,-1),(3,1,1),(3,2,-3)
运算结果为:
┌10.0 0.0 │
│0.0 10.0 │
│-2.0 3.0 │
└────────┘
```

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

```
PS H:\data_structures> python xishujuzhentest.py
矩阵加法: 1
矩阵减法: 2
矩阵乘法: 3
退出: e
请选择将要进行的矩阵运算3
以三元组形式输入第一个矩阵, 回车结束 (1, 1, 4), (1, 2, -3), (1, 5, 1), (2, 4, 8), (3, 3, 1), (4, 5, 70)
以三元组形式输入第二个矩阵, 回车结束 (1, 1, 3), (2, 1, 4), (2, 2, 2), (3, 2, 1), (4, 1, 1), (5, 3, 0)
运算结果为:
-----
| 0.0 -6.0 0.0 |
| 8.0 0.0 0.0 |
| 0.0 1.0 0.0 |
| 0.0 0.0 0.0 |
-----
--
矩阵加法: 1
矩阵减法: 2
矩阵乘法: 3
退出: e
请选择将要进行的矩阵运算e
PS H:\data_structures>
```

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