**CSE 101**
**数据结构和算法介绍编程作业4**

在这项作业中，你将创建一个计算器，用于执行矩阵运算，利用其矩阵操作数的（预期）稀疏性。如果一个$n\times n$的正方形矩阵的非零条目数（这里缩写为NNZ）与总条目数$n^2$相比是很小的，那么就可以说它是*稀疏的*。其结果将是一个能够进行快速矩阵运算的C语言程序，即使是对非常大的矩阵，只要它们是稀疏的。

给定$n\times n$矩阵$A$和$B$，它们的乘积$C=A\cdot B$是$n\times n$矩阵，其$ij^{\text{th}}$条目由以下公式给出

$$C_{ij} = \sum_{k=1}^{n} {}_{A}i_{kBkj}。$$

因此，$C$的$i^{\text{th}}$行和$j^{\text{th}}$列中的元素是$A$的$i^{\text{th}}$行与$B$的$j^{\text{th}}$列的向量点积。如果我们认为实数的加法和乘法是我们的基本操作，那么上述公式的计算时间为$\Theta(n^3)$，这对于矩阵大小$n$*超过*几千的情况来说是不现实的。如果$A$和$B$*恰好是*稀疏的，那么这些算术运算中就有很多涉及到加法或乘法的零，因此是不必要的。

$A$和$B$的和$S$和差$D$是$n\times n$矩阵，有$ij^{\text{th}}$条目。

$$_{ij}S = A_{ij} + B_{ij} \qquad \text{和} \qquad _{ij}D = A_{ij} - B_{ij}$$

实数$x$与$A$*的*标量乘积表示为$xA$，其$ij^{\text{th}}$条目$(xA)_{ij} = x \cdot A_{ij}$。$A$的转置，表示为$A^T$，是一个矩阵，其$ij^{th}$条目是$A$的$ji^{th}$条目：$(A^T)_{ij} = A_{ji}$。换句话说，$A$的行是$A^T$的列，$A$的列是$A^T$的行。这些操作都可以在$\Theta(n^2)$的时间内计算完成，就像乘法一样，当$A$和$B$是稀疏的时候，它们的成本可以得到明显的改善。

正如人们所期望的那样，矩阵运算的成本在很大程度上取决于用于表示其矩阵操作数的数据结构的选择。有几种方法来表示一个实数项的方形矩阵。标准的方法是使用一个二维的$n\times n$的双数数组。这种表示方法的优点是，上述所有的矩阵操作都可以通过嵌套循环来直接实现。然而，本项目将使用一种非常不同的表示方法。在这里，你将把矩阵表示为一个一维的Lists数组。每个List将代表矩阵的一行，但只有非零条目会被存储。因此，List元素不仅要存储矩阵的条目，还要存储这些条目所在的列索引。例如，下面的矩阵将以列表数组的形式表示，如下。

$$M = \begin{bmatrix} 1.0 & 0.0 & 2.0 \\ 3.0 & 0.0 & 0.0 \\ 0.0 & 4.0 & 5.0 \end{bmatrix}$$

列表的阵列。
[1: (1, 1.0) (3, 2.0)
2: (1, 3.0)
3: (2, 4.0) (3, 5.0)

当矩阵是稀疏的时候，这种方法显然可以节省大量空间。此外，上面定义的标准矩阵操作可以在稀疏矩阵上更有效地执行。不过，正如你所看到的，使用这种表示方法实现矩阵运算要困难得多。这样一来，稀疏矩阵的空间和时间效率都得到了提高，但代价是更复杂的

执行标准矩阵操作的算法。用我们的List
ADT操作来设计这些算法将构成你在这项作业中的大部分工作。

首先有必要对你的List
ADT从pa1开始做一些小的修改。首先，你必须将你的ADT从一个ints的列表转换成一个通用指针的列表。这需要将某些字段类型、声明语句、方法参数和返回类型从`int`改为`void*`。这些List元素所引用的对象将被定义在下面指定的Matrix
ADT中。第二，有必要取消List的操作equals()和copy()。这些函数的问题在于，List不再知道它是一个什么列表，因此只能执行这些操作的 "浅层"版本，即比较或复制指针，而不是它们所指向的东西。出于同样的原因，函数printList()不再是必需的（但你可能希望包括它，只是为了诊断的目的）。函数cat()（可选）也可以被包括在内。pa1的所有其他List操作将被保留。

List客户中的一个 "典型循环"（如之前pa1项目描述的第3页所示）现在可能显示为

```
            type x; // 这里的 "type "是指void的任何数据类型。
                                    // List中的指针所指向的是，由
           // 客户端，但不是由List.moveFront(L)。
while( index(L)>=0 ){
   // 获取当前元素，并将其转换为正确的
   // 指针，跟随指针，将其值分配给x。x = *(type*)get(L)。

   //对x做一些事情，然后moveNext(L)
   。
}
```

请注意，equals()、copy()和printList()操作仍然可以被执行，但现在必须从客户层面进行。在你尝试实现下面定义的Matrix ADT之前，请务必测试这个修改过的List
ADT。一个名为ListClient.c的修改后的List
ADT的测试（非常弱）将发布在该类网页的例子部分。

**文件格式**

本项目的顶级客户模块将被称为Sparse.c。它将接受两个命令行参数，分别给出输入和输出文件的名称。输入文件将以一行开始，包含三个整数$n$、$a$和$b$，以空格分隔。第二行将是空白，接下来的$a$行将指定$n\times n$矩阵$A$的非零项。每一行都将包含一个以空格分隔的三个数字列表：两个整数和一个双数，给出相应矩阵条目的行、列和值。在另一个空白行之后，将有$b$行指定$n\times n$矩阵$B$的非零条目。例如，两个矩阵

$$A = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \\ 7.0 & 8.0 & 9.0 \end{bmatrix} \quad \text{和} \quad B = \begin{bmatrix} 1.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix}$$

是由以下输入文件编码的。

```
3 9 5

1 1 1.0
1 2 2.0
1 3 3.0
2 1 4.0
2 2 5.0
2 3 6.0
3 1 7.0
3 2 8.0
3 3 9.0

1 1 1.0
1 3 1.0
3 1 1.0
3 2 1.0
3 3 1.0
```

你的程序将读取上述输入文件，初始化并建立矩阵$A$和$B$的数组表示，然后计算并打印下列矩阵到输出文件。$A$, $B$, $(1.5)A$, $A + B$, $A + A$, $B$ - $A$, $A$ - $A$, $A^T$ , $AB$ 和 $B^2$ 。下面的例子说明了输出文件的格式，它与上述输入文件相对应。

```
A有9个非零条目。
1: (1, 1.0) (2, 2.0) (3, 3.0)
2: (1, 4.0) (2, 5.0) (3, 6.0)
3: (1, 7.0) (2, 8.0) (3, 9.0)

B有5个非零条目。
1: (1, 1.0) (3, 1.0)
3: (1, 1.0) (2, 1.0) (3, 1.0)

(1.5)*A =
1: (1, 1.5) (2, 3.0) (3, 4.5)
2: (1, 6.0) (2, 7.5) (3, 9.0)
3: (1, 10.5) (2, 12.0) (3, 13.5

) a+b =
1: (1, 2.0) (2, 2.0) (3, 4.0)
2: (1, 4.0) (2, 5.0) (3, 6.0)
3: (1, 8.0) (2, 9.0) (3, 10.0)

A+A=
1: (1, 2.0) (2, 4.0) (3, 6.0)
2: (1, 8.0) (2, 10.0) (3, 12.0)

3: (1, 14.0) (2, 16.0) (3, 18.0

) b-a=
1: (2, -2.0) (3, -2.0)
2: (1, -4.0) (2, -5.0) (3, -6.0)
3: (1, -6.0) (2, -7.0) (3, -8.0)


A-A =
```

```
转置(A) =
1: (1, 1.0) (2, 4.0) (3, 7.0)
2: (1, 2.0) (2, 5.0) (3, 8.0)
3: (1, 3.0) (2, 6.0) (3, 9.0)

A*B=
1: (1, 4.0) (2, 3.0) (3, 4.0)
2: (1, 10.0) (2, 6.0) (3, 10.0)
3: (1, 16.0) (2, 9.0) (3, 16.0)

B*B=
1: (1, 2.0) (2, 1.0) (3, 2.0)
3: (1, 2.0) (2, 1.0) (3, 2.0)
```

请注意，行将以列排序的方式打印，而零行将被完全跳过。一个零的矩阵将导致没有输出被打印出来，正如上面的矩阵*A* - *A*所见。注意，与输出文件不同，输入文件可以以任何顺序给出矩阵条目。

**Matrix ADT规格**

除了主程序Sparse.c和来自pa1的经过修改的List.c之外，你将在一个名为Matrix.c的文件中实现一个Matrix ADT。这个ADT将包含一个私有的内部结构（类似于List ADT中的Node），封装了与矩阵条目对应的列和值信息。你可以给这个内部结构起任何名字，但我在这里将它称为Entry。因此 Entry 是一个指向名为 EntryObj 的结构的指针，该结构有两个字段，类型分别为 int 和 double。你的Matrix ADT将把一个矩阵表示为Entry的Lists数组。由于Entry本身是一个指针，它可以被分配给一个类型为`void*`的字段。要求每个List of Entries都以列排序的方式维护。你的矩阵ADT将输出以下操作。

```
// newMatrix()
// 返回一个零状态的新nXn矩阵对象的引用。矩阵 newMatrix(int n)

// freeMatrix()
//释放与*pM相关的堆内存，将*pM设为NULL。 void freeMatrix(Matrix*
pM)。


// 访问功能

// size()
// 返回正方形矩阵M的大小，int size(Matrix M)
。

// NNZ()
// 返回M中非零元素的数量。 int NNZ(Matrix M);

// equals()
// 如果矩阵A和B相等，返回真（1），否则返回假（0）。 int equals(Matrix A, Matrix
B);
```

// 操纵程序

// makeZero()
// 将M重新设置为零矩阵状态。 void
makeZero(Matrix M);

// changeEntry()
//将M的第i行，第j列改为x值。
// Pre: 1<=i<=size(M), 1<=j<=size(M)
void changeEntry(Matrix M, int i, int j, double x)。

// 矩阵算术操作

// copy()
// 返回一个新的Matrix对象的引用，该对象的条目与A相同。 Matrix copy(Matrix A)。

// 转置()
// 返回一个代表转置的新矩阵对象的引用。
A的//。
矩阵转置(矩阵A)。

// scalarMult()
// 返回一个代表xA的新矩阵对象的引用。Matrix scalarMult(double x,
Matrix A);

// sum()
// 返回一个代表A+B的新矩阵对象的引用。
//预：size(A)==size(B)
矩阵sum（矩阵A，矩阵B）。

// diff()
// 返回一个代表A-B的新矩阵对象的引用。
//预：size(A)==size(B)
Matrix diff(Matrix A, Matrix B)。

// product()
// 返回一个代表AB的新矩阵对象的引用
//预：size(A)==size(B)
矩阵乘积（矩阵A，矩阵B）。

// printMatrix()
// 将矩阵M的字符串表示法打印到文件流中。零行
//不被打印。每一个非零行被表示为一行，包括
行号的//，后面是冒号，一个空格，然后是一个空格隔开。
// 列表中的"(col, val) "给出了列号和非零值。
//在该行。双重值将被四舍五入到小数点后1位。 void printMatrix(FILE* out,
Matrix M);

要求您的程序能有效地执行这些操作。让$n$为$A$中的行数，让$a$和$b$分别表示$A$和$B$中的非零条目数。那么上述函数的最坏情况下的运行时间应该有以下渐进增长率。

| | |
|---|---|
| changeEntry(A, i, j, x)。 | $\Theta(a)$ |
| copy(A): | $\Theta(n+a)$ |
| 转置(A)。 | $\Theta(n+a)$ |
| scalarMult(x, A)。 | $\Theta(n+a)$ |
| sum(A, B): | $\Theta(n+a+b)$ |
| diff(A, B): | $\Theta(n+a+b)$ |
| 积(A, B)。 | $\Theta(n^2+a\cdot b)$ |

在Matrix.c中加入一个私有函数将是很有帮助的，其签名为

```
double vectorDot(List P, List Q)
```

该函数计算由列表P和Q代表的两个矩阵行的向量点积，与函数transpose()一起使用有助于实现product()。类似的sum()和diff()操作的辅助函数也会很有用，强烈推荐。

**要交的东西**

你的项目将由三个文件构成。Sparse.c、Matrix.c和List.c（还有头文件Matrix.h和List.h）。主程序Sparse.c将处理输入和输出文件，是Matrix ADT的客户端，而Matrix ADT本身也是修改后的List ADT的客户端。请注意，Sparse本身不是List的直接客户，因为它不需要调用任何List操作。你还将编写单独的客户端模块ListTest.c和MatrixTest.c来单独测试List和Matrix ADTs。学生们经常问这些测试文件的内容应该是什么。在每种情况下，包括足够多的ADT操作的调用，以使评分者相信你确实在大项目中使用List和Matrix ADT模块之前对它们进行了隔离测试。做到这一点的最好方法是将*它们实际用于这一目的*。至少它们应该至少调用一次各自ADT模块中的每个公共函数。网页链接 Examples/pa4 将包含文件MatrixClient.c 和 ListClient.c。这些应该被认为是对你的 Matrix 和 List 模块的弱测试，并不适合你的测试目的。一些匹配的输入/输出文件对也将被包括在内，同时还包括一个用于创建随机输入文件的Python脚本。

同时提交一个README文件和一个Makefile 文 件 ，它可以创建一个名为Sparse 的可执行文件。(Examples/pa4中也有一个Makefile，你可以根据你的需要进行修改）。因此，总共将提交九个文件。

| | |
|---|---|
| Sparse.c | 由你写的 |
| 矩阵.c | " " " |
| 矩阵.h | " " " |
| MatrixTest.c | " " " |
| 列表.c | " " " |
| 列表.h | " " " |
| ListTest.c | " " " |
| README | " " " |
| 制作文件 | 提供的，如果你喜欢的话，可以修改 |

在到期日之前，将这些文件推送到git.ucsc.edu上你的git存储库中的pa4文件夹。像往常一样，尽早开始并提出问题。