A Simple Double Mat Library for C

12111224
贾禹帆

矩阵库使用说明

```
#include "Mat.h"
```

浮点型矩阵定义 `dMat`

```
typedef struct dMat {
    size_t rows;
    size_t columns;
    double* data;
} dMat;
```

实现函数

```
创建矩阵
struct dMat createMatrix(size_t _rows, size_t _columns);

创建一个空矩阵(用于做输出赋值)
struct dMat makeNullMatrix();

删除矩阵
void deleteMatrix(dMat* _in);

复制矩阵
void copyMatrix(const dMat* _ori, dMat* _trg);

矩阵相加
void addMatrix(const dMat* _lhs, const dMat* _rhs, dMat* _res);

矩阵相减
void subtractMatrix(const dMat* _lhs, const dMat* _rhs, dMat* _res);

矩阵和标量(对应倍数单位阵)相加
void addScalar(const dMat* _lhs, double _val, dMat* _res);

矩阵和标量(对应倍数单位阵)相减
void subtractScalar(const dMat* _lhs, double _val, dMat* _res);

矩阵相乘
void multiplyMatrix(const dMat* _lhs, const dMat* _rhs, dMat* _res, int _me

/*
Avaiable method:
#define MATMUL_GENERATE 1
#define MATMUL_COPPERSMITHWINOGRAD 2
*/

矩阵乘标量
void multiplyScalar(const dMat* _lhs, double _val, dMat* _res);

求转置矩阵
void transposeMatrix(const dMat* _ori, dMat* _trg);

获得最大元素值
double getMax(const dMat* _mat);

获得最小元素值
double getMin(const dMat* _mat);
```

```
打印整个矩阵
void printMat(const dMat* _ori, unsigned int _totalDigits,
unsigned int _point_digits);
```

宏

```
访问特定元素
#define get(__mat, __row, __col) __mat.data[__row * __mat.columns + __col]
```

具体实现

结构体 `dMat`

行列表示

矩阵由行和列构成,行列数值的最大值直接影响到所能表示矩阵的大小,选用 `size_t` 表示的理由如下

1. 对于各个操作系统，所能使用的最大内存为2^xbytes,x为操作系统位数。size_t的大小与操作系统位数一致,以64位操作系统为例，理论上能使用的最大内存为2^{64}bytes,而 `double` 类型大小为64bits=8bytes,所以矩阵最多能容纳2^{61}个元素(实际无法达到),矩阵单边最大值为2^{61},在 `size_t` 的表示范围内。以上为理论上存储的极限，实际中家用电脑内存一般为8G~128G(2^{33}~2^{37}bytes)，能存储的元素更是少的多
2. 既然限制是2^{61},那么使用 `long long` 也是可行的，但是考虑到行列数 > 0,使用 `size_t` (unsign long long)能够减少一部分出错的概率

> 为什么不用const:由于c结构体完全公开没有封装，行列设为const可以避免不小心修改造成的问题，但是这也直接禁用了浅拷贝和重新分配行列大小的行为,并且不能创建一个默认的空dMat接收结果

```
dMat res=makeNullMatrix();
copyMatrix(&ori,&res);
```

数据存储

数据存储有两种实现方法，一是由一个 `double*` 指针管理，一是使用类似二维数组的方式由 `double**` 管理每行的 `double*` ，各自的优缺点如下

double*

优点

1. 一次性申请内存，若堆空间不足(返回NULL)可以进行异常处理而不必释放先前申请的内存
2. 只需要一次解引用即可访问，速度快(可以使用指针算术快速遍历)

缺点:

1. 不能直接按二维数组表示法访问

double**

优点:

1. 可以直接通过两次operator [] 以数组方式下标直接访问
2. 理论上可以管理2^{65}个元素，上限更高(~~但是没用，没有那么大内存~~)

缺点:

1. 访问较慢(两次解引用)
2. 逐级多次申请堆空间的机制导致一旦出现空间不足需要手动释放之前申请的空间

综上， `double*` 要优于 `double**` ,并且不直观的访问方式也可以通过宏解决

函数

- 由于c不支持try...catch，而返回错误码处理过于繁琐，且大部分问题为矩阵行列数不正确,将会直接导致结果不可控，而且极难定位。因此为了立即定位问题位置，使用 `goto` 跳转打印错误原因后直接终止程序 `abort()`

```
//定义宏简化代码
#define ErrorLabel(_unused)          \
    MemoryLack:                      \
    printf("Lack of Memory\n");      \
    abort();                         \
    ErrorMatFormat:                  \
    printf("Wrong rows/columns\n"); \
    abort();

//usage example:
static inline dMat newMat(size_t rows, size_t columns) {
    if(columns > ULLONG_MAX / rows / sizeof(double))
        goto MemoryLack;
    double* ptr = calloc(rows * columns, sizeof(double));
    if(ptr == NULL)
```

```
        goto MemoryLack;
    dMat res = { rows, columns, ptr };
    return res;
    ErrorLabel(newMat)
}

void multiplyMatrix(const dMat* lhs, const dMat* rhs, dMat* res, int methoc
    if(lhs->rows == 0 || lhs->columns == 0 || rhs->columns == 0)
        goto ErrorMatFormat;
    if(lhs->columns != rhs->rows)
        goto ErrorMatFormat;
    if(res->data != NULL) {
        if(res->rows != lhs->rows)
            goto ErrorMatFormat;
        if(res->columns != rhs->columns)
            goto ErrorMatFormat;
    } else

        *res = newMat(lhs->rows, rhs->columns);
    switch(method) {
        case MATMUL_GENERATE:
            simGenMatMul(lhs->data, rhs->data, res->data,
            lhs->rows, rhs->columns, lhs->columns);
            break;
        case MATMUL_COPPERSMITHWINOGRAD:
            CoppersmithWinogradMatMul(lhs->data, rhs->data, res->data,
            lhs->rows, rhs->columns, lhs->columns, lhs->columns,
                                      rhs->columns, lhs->columns);
            break;
        default:
            printf("Unkonw method, use Generate method instead\n");
            simGenMatMul(lhs->data, rhs->data, res->data,
            lhs->rows, rhs->columns, lhs->columns);
    }
    return;
    ErrorLabel(multiplyMatrix)
}
```

- 导致程序终止的异常:

  i. 创建长或宽等于0的矩阵(createMatrix)
  ii. 输入的矩阵长或宽为0(防止用户乱改行列)
  iii. 申请过大矩阵导致堆空间不足
  iv. 矩阵加减标量时矩阵不是方阵
  v. 矩阵相加减时lhs,rhs行列不相等
  vi. 矩阵相乘不满足lhs's columns = rhs's rows

vii. 用于存储返回值的矩阵非空(NULL)但行列不满足存储的要求

- 矩阵乘法方面依据情况提供了传统的按定义计算和Coppersmith-Winograd法.可根据情况选用。(参考论文和代码),具体算法如下:
  - $C = A \times B$
    $C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{1,2}$
    $C_{1,2} = A1,1B_{1,2} + A_{1,2}B_{2,2}$
    $C_{2,1} = A_{2,1}B_{1,2} + A_{2,2}B_{2,1}$
    $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$
  - 其中，每个矩阵被分成四个部分(行列都为偶数时),然后按上述方法递归运算
  - 若为奇数，则用定义计算

> 计算4000 x 5000 * 5000 x 6000矩阵乘法对比:
> ```
> ---------------------------
> Generate method's time cost: 1130.221052 s
> ---------------------------
> Coppersmith-Winograd method's time cost: 159.794048 s
> ```
> 按定义计算消耗空间最小但速度慢，Coppersmith-Winograd法较快但需要更多内存

- 为了提升性能,可以通过提前存储解引用后的指针来减少不必要的多次解引用
- 观察生成的汇编文件发现每次进入循环时会进行跳转，会保存当前寄存器状态然后压栈进入目标循环，因此在较为极端的情况下，大外循环套小内循环会导致性能的严重下降，因此在不强调内外顺序的循环嵌套中可以判断内外循环大小实现小外循环来提升性能
- 因为c本身不支持函数重载(虽然可以用编译器宏，但是过于麻烦),可变参数列表也必须手动提供长度而不能像C++一样使用size_of展开，加上批量拷贝和删除矩阵的需求不常见,用时可以通过循环完成，故没有实现因
  此 `copyMatrix()` , `deleteMatrix()` 并没有使用可变参数列表

Samples

ErrorSamples.c

罗列了错误样例(注释),可以分别取消注释进行试运行

Demo.c

示例程序，演示了正确的矩阵操作

```
m2 copy from m1,change m2 doesn't affect m1:
m2:
          0.00000             0.00000             0.00000             0.00000             0.00000
          0.00000             0.00000             0.00000             0.00000             0.00000
          0.00000             0.00000             0.00000             0.00000             0.00000
          0.00000             0.00000             0.00000             0.00000             0.00000
          0.00000             0.00000             0.00000             0.00000             0.00000
m1:
    1564178577.56897    -1624825203.66053      983113110.09643      -29909858.42089    -1023408414.66264
    1004423651.78189     -919819089.71214    -1310889260.79342     -559320022.37742    -1869340530.85539
     531587074.60137     1361798350.11460     1601261565.03020     -960693497.94194    -2027210828.98882
     -79161134.05338    -1714529366.19673    -1961582278.97587     -970167153.45496      225385858.28600
    1888524980.78108    -2015109867.95065    -1432455983.45936     2027084300.43536       22301594.32334
-------------------------
m1=m1-114514.1919810
    1564178577.56897    -1624825203.66053      983113110.09643      -29909858.42089    -1023408414.66264
    1004423651.78189     -919819089.71214    -1310889260.79342     -559320022.37742    -1869340530.85539
     531587074.60137     1361798350.11460     1601261565.03020     -960693497.94194    -2027210828.98882
     -79161134.05338    -1714529366.19673    -1961582278.97587     -970167153.45496      225385858.28600
    1888524980.78108    -2015109867.95065    -1432455983.45936     2027084300.43536       22301594.32334
-------------------------
m2=m2+51.4
         51.40000             0.00000             0.00000             0.00000             0.00000
          0.00000            51.40000             0.00000             0.00000             0.00000
          0.00000             0.00000            51.40000             0.00000             0.00000
          0.00000             0.00000             0.00000            51.40000             0.00000
          0.00000             0.00000             0.00000             0.00000            51.40000
-------------------------
m3=m1-m2
    1564064011.97699    -1624825203.66053      983113110.09643      -29909858.42089    -1023408414.66264
    1004423651.78189     -919933655.30413    -1310889260.79342     -559320022.37742    -1869340530.85539
     531587074.60137     1361798350.11460     1601146999.43822     -960693497.94194    -2027210828.98882
     -79161134.05338    -1714529366.19673    -1961582278.97587     -970281719.04694      225385858.28600
    1888524980.78108    -2015109867.95065    -1432455983.45936     2027084300.43536       22187028.73136
-------------------------
m4=m3's transpose
    1564064011.97699     1004423651.78189      531587074.60137      -79161134.05338     1888524980.78108
   -1624825203.66053     -919933655.30413     1361798350.11460    -1714529366.19673    -2015109867.95065
     983113110.09643    -1310889260.79342     1601146999.43822    -1961582278.97587    -1432455983.45936
     -29909858.42089     -559320022.37742     -960693497.94194     -970281719.04694     2027084300.43536
   -1023408414.66264    -1869340530.85539    -2027210828.98882      225385858.28600       22187028.73136
-------------------------
m1's max element is     2027084300.43536
m2's min element is              0.00000
-------------------------
```

> 部分输出截图,详见DemoLog.txt

文件结构

```
./
├── CMakeLists.txt
├── DemoLog.txt                          Demo输出文件
├── header                               引用头文件
│   └── Mat.h
├── img
│   ├── DemoLog.png
│   └── time_cost_comapration.png
├── modules                              源文件
│   └── Mat.c
├── project3.pdf                         project3需求
├── README.md
├── report.md                            报告(markdown)
├── 「report」.pdf                        报告(pdf)
```

```
└── samples                    案例
    ├── Demo.c
    └── ErrorSamples.c
```

总结

这次写了个简陋的C double matrix库，没有构造和析构函数不能实现RAII,需要用户手动释放，同时没有封装也增大了意外修改变量的风险。通过这次project熟悉了C语言，并对C和C++的差别有了更清晰的认识