# 数据结构与算法专题实践1

## 1．问题描述/需求分析

1. 构建并管理计算机缓冲区模拟器
2. 组织与管理文件中的大型二维数组乘法
3. 统计运算过程中的缓存区缺失次数
4. 绘制缓存区缺失次数与缓存区大小关系图像
5. 绘制缓存区缺失次数与矩阵大小关系图像

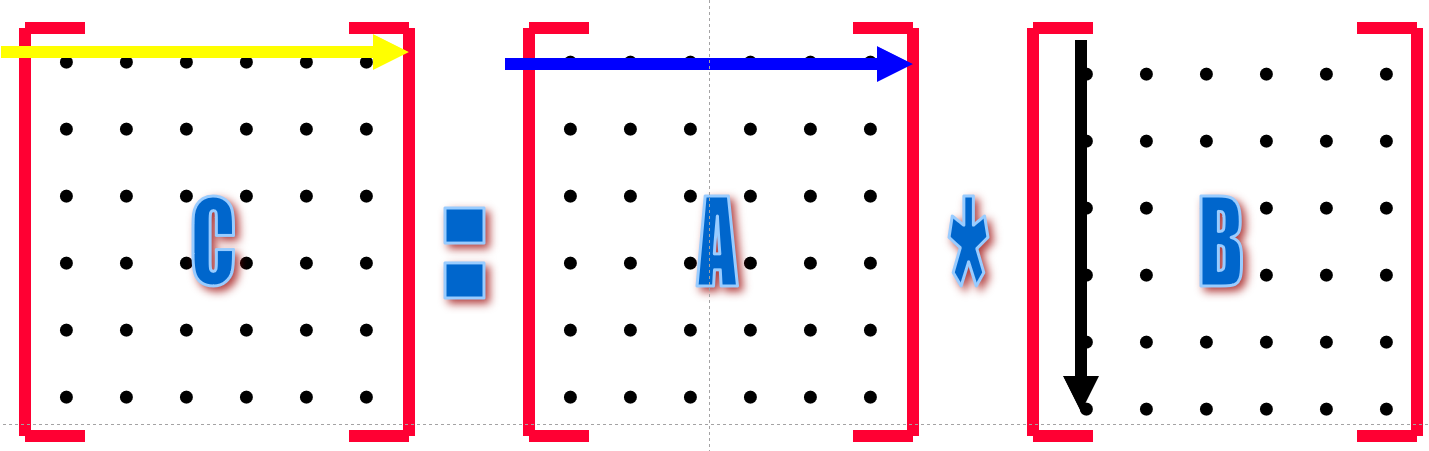
## 2．系统结构/算法思想

### 基本思路：

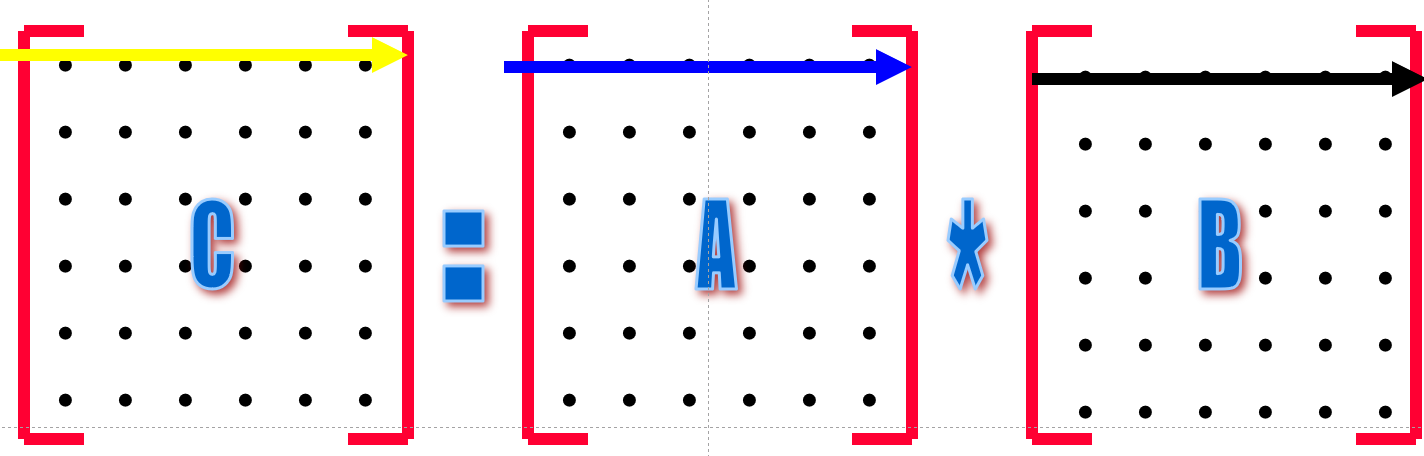
基本思路为：运算层按照操作层提供的指令运算通过缓冲区获取的数据。

运算层包含两种算法：ijk矩阵乘法算法以及ikj矩阵乘法算法

ijk矩阵乘法算法的数据访问过程如图所示：



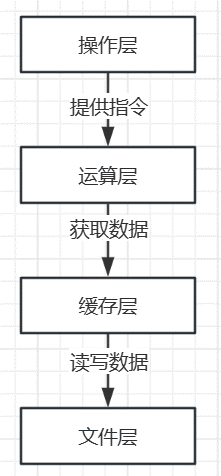
ikj矩阵乘法算法的数据访问过程如图所示：



缓冲区采用页替换模式，出现未命中情况后以缺失位置为基准更新整个缓冲区。

由图可得，ikj矩阵乘法算法数据访问更加规律，因此更容易命中缓冲区。

### 系统框架：



### 描述各模块功能及其关系：

**操作模块：**包含main函数，向矩阵运算模块提供操作指令

**矩阵运算模块：**包含两种矩阵乘法算法，按照操作模块提供的指令进行运算，从缓存模块获取数据。

**缓存模块：**为矩阵运算模块提供数据，在数据缺失时自动从文件中读取数据，保障数据一致性，使得矩阵运算模块如同直接访问文件一般访问数据。

**文件模块：**存储矩阵内容以及相关信息。

## 3．功能模块设计

### 模块设计思想：

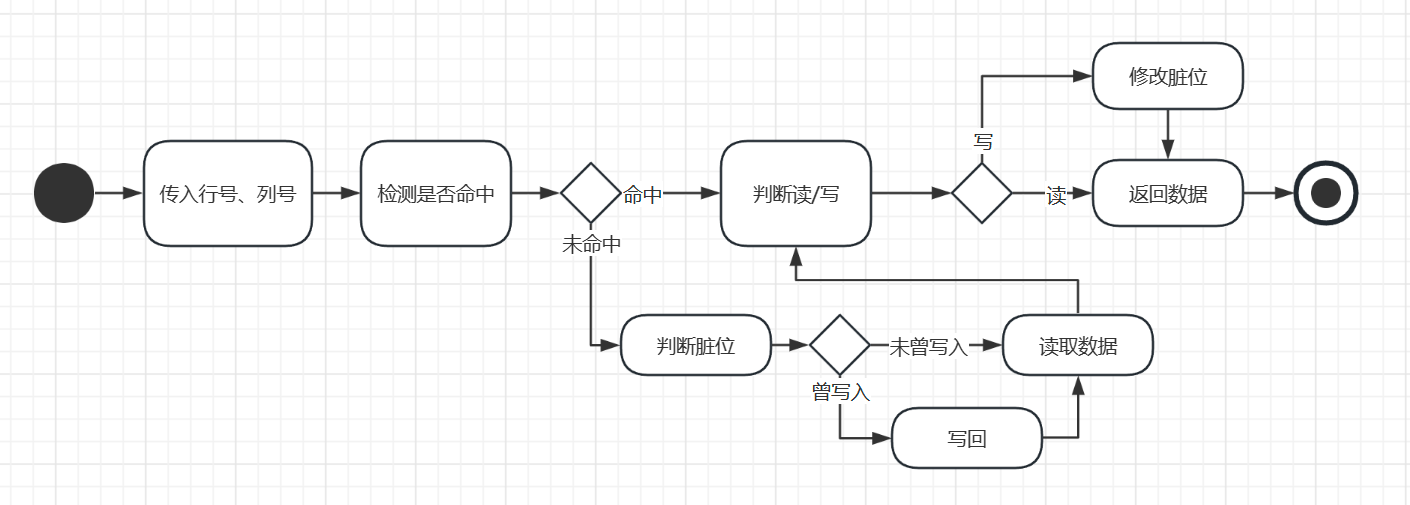
**操作模块：**包含main函数，向矩阵运算模块提供操作指令，将运算结果通过屏幕进行图形化输出，同时输出缓存分析。

**矩阵运算模块：**即Matrix类。主要包含两种矩阵乘法算法，并为操作模块提供打印接口和数据构建接口。按照操作模块提供的指令选择算法进行运算。通过缓存模块提供的接口从中获取数据。

**缓存模块：**即Buff类。为达到快速检索的目的，底层使用哈希图。提供getNum()与setNum()接口为矩阵运算模块提供数据。通过刷新操作在数据缺失时自动从文件中读取数据。通过脏位保障数据一性，在缓冲区内容被修改后设置为脏，从而在刷新数据前将数据写回。使得矩阵运算模块如同直接访问文件一般访问数据。具体内容见流程图

**文件模块：**矩阵以二进制形式(.bin)存储于文件中，前两个数据为整型的行与列，之后是按顺序以双精度浮点型存储的矩阵内容。缓存模块通过文件指针的操作访问数据。

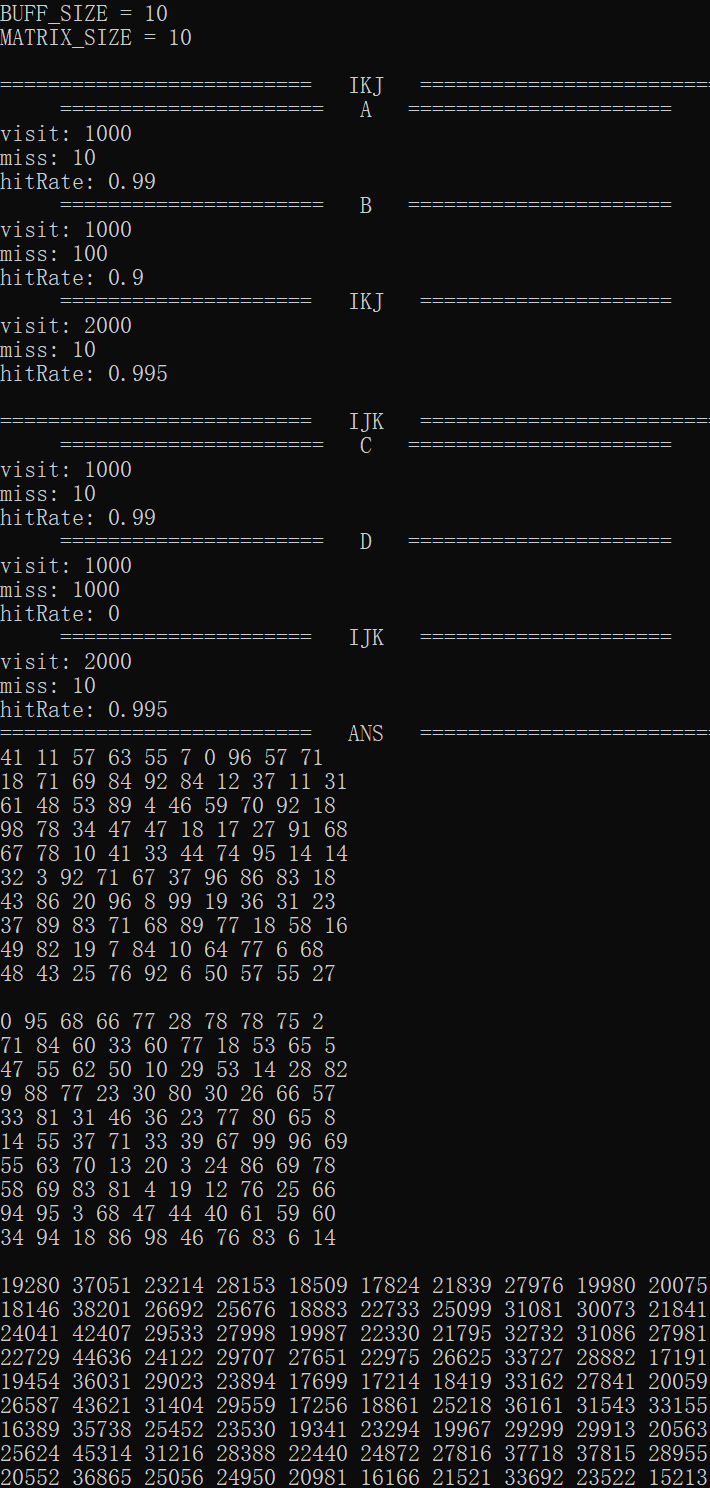
### 缓存流程图：



## 4．测试结果与分析

**测试数据选择或生成办法：**通过rand()随机算法生成指定大小的测试矩阵。

**运行结果截图：**



**性能图：**

如图所示，IKJ算法缺失次数整体小于IJK算法。初始情况下IKJ算法的缺失次数随着缓冲区的增加变化明显。在缓冲区接近列数后两个算法的缺失次数趋于减缓。而当缓冲区大小达到列数后，两者的缺失次数再次出现大幅下降的情况。而当缓冲区大小超过列数后，IJK算法的缺失次数发生骤降后稳定，而IKJ算法的缺失次数开始波动式缓慢下降。

如图所示，IKJ算法缺失次数整体小于IJK算法。两者缺失次数的变化可以用二阶多项式完美拟合(R2>0.99)，拟合IJK算法的二阶多项式系数显著大于IKJ算法，表现为更快的上升。

如图所示，命中率的变化趋势非常接近缺失次数的变化趋势，故不再讨论。

## 5．实验总结

本次实验通过模拟外部大矩阵乘法，探究了矩阵规模、缓存大小对缓存命中率的影响，进而学习算法和缓存之间的相互作用。在本实验中，ikj矩阵乘法算法在大多数情况下都快于ijk矩阵乘法算法，同时，随缓存区大小的增加，ikj矩阵乘法算法的命中率快速上升。

在实验的过程中遇到了以下问题：

问题：文件在读取、写入时会清空原有文件

方案：通过查找资料可知，ofstream默认的ios::out文件操作类型会清空原有文件内容，若使用ios::in|ios::out类型则可以保留文件内容的同时进行IO操作，故需添加ios::in参数。

问题：缓存区是否应当使用最近最久未使用算法LRU算法

方案：通过分析算法特性可得，使用LRU算法不但会增大程序编写难度，更会导致缓存命中率下降，因此舍弃LRU算法。

## 6. 源代码

Matrix.h //矩阵类，内联，包含两种矩阵乘法算法

Buffer.cpp //缓存类实现

Buffer.h //缓存类，包含写入读取以及自动写回

Definition.h //宏定义，缓存大小与矩阵大小的定义

MatrixMultiplication.cpp //主程序文件，包含main函数