华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析　　**年级：**22 级　　**上机实践成绩：**

指导教师：金澈清　　**姓名：**郭夏辉

上机实践名称：Strassen 算法　　**学号：**10211900416　　**上机实践日期：**2023 年 3 月 9 日

上机实践编号：No.2　　**组号：**1-416

一、目的

1. 熟悉算法设计的基本思想
2. 掌握 Strassen 算法的基本思想，并且能够分析算法性能

二、内容与设计思想

1. 设计一个随机数矩阵生成器，输入参数包括 N, s, t；可随机生成一个大小为 N*N、数值范围在[s, t]之间的矩阵。
2. 编程实现普通的矩阵乘法；
3. 编程实现 Strassen’s algorithm；
4. 在不同数据规模情况下（数据规模 N=2^4, 2^8, 2^9, 2^10, 2^11）下，两种算法的运行时间各是多少；
5. 思考题：修改 Strassen’s algorithm，使之适应矩阵规模 N 不是 2 的幂的情况；
6. 改进后的算法与 2 中的算法在相同数据规模下进行比较。

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. 随机数矩阵生成器

为了后续的实验方便，我将该生成器设计成了输出到文件的形式。

get-random.cpp

```cpp
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <ctime>
using namespace std;
int n,s,t;
int main() {
    srand(time(NULL));
    ofstream fout("data.txt");
    cin>>n>>s>>t;
    fout<<n<<endl;
    for(int i=0;i<n;i++) {
```

```
        for (int j=0;j<n;j++) {
            fout<< s+rand()%(t-s+1)<<" ";
            fout<< s+rand()%(t-s+1)<<" ";
        }
        fout<<endl;
    }
    for(int i=0;i<n;i++) {
        for (int j=0;j<n;j++) {
            fout<< s+rand()%(t-s+1)<<" ";
        }
        fout<<endl;
    }
    fout.close();
    return 0;
}
```

2. 普通矩阵乘法

这里涉及到一个问题，我们以什么顺序进行运算？根据《深入理解计算机系统》(CSAPP 第三版，机械工业出版社)Page448-450,我们可以知道采用 kij 或者 ikj 版本的不命中率最低，时间效率最高。我在这个问题中采用 kij 版本书写代码。

normal.cpp

```
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
#include<ctime>
#include<cstdio>
#include<cstring>
#define MAXN 2050
using namespace std;
int A[MAXN][MAXN],B[MAXN][MAXN],C[MAXN][MAXN];
//A,B 作为输入矩阵 C 作为结果矩阵
int n;
int main(){
    std::ios::sync_with_stdio(false);
    ifstream fin("data.txt");
    ofstream fout("result.txt");
    fin>>n;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++)fin>>A[i][j];
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++)fin>>B[i][j];
    }
    //这里涉及到一个问题，我们以什么顺序进行运算？
```

```
    //根据《深入理解计算机系统》(CSAPP 第三版，机械工业出版社)Page448-450
    //我们可以知道采用 kij 或者 ikj 版本的不命中率最低，时间效率最高
    //我在这个问题中采用 kij 版本书写代码
    int tmp;
    clock_t start, stop;
    start=clock();
    for(int k=0;k<n;k++){
        for(int i=0;i<n;i++){
            tmp=A[i][k];
            for(int j=0;j<n;j++){
                C[i][j]+=tmp*B[k][j];
            }
        }
    }
    stop=clock();
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            fout<<C[i][j]<<" ";
        }
        fout<<endl;
    }
    cout<<"Time: "<<1000*((double)(stop - start) / CLOCKS_PER_SEC)<<"ms"<<
endl;
    fin.close();
    fout.close();
    return 0;
}
```

2. 普通 Strassen(只能用于数据规模 N 是 2 的幂次方)

strassen.cpp

```
//code of strassen is complex, but I can make it!
//在完成这个实验的过程中，我通过构造结构体包装相关的功能
//这个只能适应矩阵规模 N 是 2 的幂的情况
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
#include<ctime>
#include<cstdio>
#include<cstring>
using namespace std;
struct Matrix {
    int row, column;//row*column 的矩阵
    int** m; //用来存那个数组
    Matrix(int r, int c) {//初始化
```

```
        row=r;
        column=c;
        m=(int**)malloc(sizeof(int*)*r);
        for (int i = 0; i < r; i++)m[i] = (int*)malloc(sizeof(int) * c);
    }

    ~Matrix(){ //析构函数，销毁时用
        if (m != NULL) {
            for (int i=0;i<row;i++) {
                delete[] m[i];
            }
            delete[] m;
        }
    }
    Matrix(const Matrix& mat) {
        row=mat.row;
        column=mat.column;
        m=(int**)malloc(sizeof(int*)* mat.row);
        for (int i=0;i<mat.row;i++)
            m[i]=(int*)malloc(sizeof(int) * mat.column);
        for (int i=0;i<row;i++){
            for (int j=0;j<column;j++){
                m[i][j]=mat.m[i][j];
            }
        }
    }

    Matrix& operator = (const Matrix& mat) {
        //重载=
        if (this != &mat) {
            row = mat.row;
            column = mat.column;
            m = (int**)malloc(sizeof(int*) * mat.row);
            for (int i = 0; i < mat.row; i++)
                m[i] = (int*)malloc(sizeof(int) * mat.column);
            for (int i = 0; i < row; i++){
                for (int j = 0; j < column; j++){
                    m[i][j] = mat.m[i][j];
                }
            }
        }
        return *this;
    }
```

```
    Matrix operator + (const Matrix& mat) {
        //重载+
        Matrix result=Matrix(row,column);
        for(int i=0;i<row;i++){
            for(int j=0;j<column;j++){
                result.m[i][j]=(*this).m[i][j]+mat.m[i][j];
            }
        }
        return result;
    }

    Matrix operator - (const Matrix& mat) {
        //重载-
        Matrix result=Matrix(row,column);
        for(int i=0;i<row;i++){
            for(int j=0;j<column;j++){
                result.m[i][j]=(*this).m[i][j]-mat.m[i][j];
            }
        }
        return result;
    }


};
Matrix matMini(Matrix mat, int st1, int st2, int ed1, int ed2) {
    //from(st1,st2)to(ed1,ed2)
    Matrix matC=Matrix(ed1-st1+1, ed2-st2+1);
    for (int i=0;i<=(ed1-st1);i++){
        for (int j=0;j<=(ed2-st2);j++) {
            matC.m[i][j]=mat.m[st1+i][st2+j];
        }
    }
    return matC;
}
Matrix combine(Matrix mat1, Matrix mat2, Matrix mat3, Matrix mat4) {
    Matrix newmat=Matrix(mat1.row+mat3.row,mat1.column+mat2.column);
    for (int i=0;i<mat1.row;i++){
        for (int j=0;j<mat1.column;j++){
            newmat.m[i][j]=mat1.m[i][j];
        }
    }

    for (int i=0;i<mat2.row;i++){
        for (int j=0;j<mat2.column;j++){
            newmat.m[i][j+mat1.column]=mat2.m[i][j];
```

```
        }
    }

    for (int i=0;i<mat3.row;i++){
        for (int j=0;j<mat3.column;j++){
            newmat.m[mat1.row+i][j]=mat3.m[i][j];
        }
    }

    for (int i=0;i<mat4.row;i++){
        for (int j=0;j<mat4.column;j++){
            newmat.m[mat1.row+i][mat1.column+j]=mat4.m[i][j];
        }
    }
    return newmat;
}

Matrix strassen(Matrix* matA, Matrix* matB) {
    if ((*matA).row == 1 && (*matA).column == 1 && (*matB).row == 1 &&
(*matB).column == 1) {
        Matrix matC=Matrix(1, 1);
        matC.m[0][0]=(*matA).m[0][0]*(*matB).m[0][0];
        return matC;
    }
    int mid1=(*matA).row/2 - 1;
    int mid2=(*matA).column/2 - 1;

    Matrix a11=matMini((*matA),0,0,mid1,mid2);
    Matrix a12=matMini((*matA),0, mid2+1,mid1,(*matA).column- 1);
    Matrix a21=matMini((*matA),mid1+1,0,(*matA).row-1,mid2);
    Matrix
a22=matMini((*matA),mid1+1,mid2+1,(*matA).row-1,(*matA).column-1);

    Matrix b11=matMini((*matB),0,0,mid1,mid2);
    Matrix b12=matMini((*matB),0, mid2+1,mid1,(*matB).column- 1);
    Matrix b21=matMini((*matB),mid1+1,0,(*matB).row-1,mid2);
    Matrix
b22=matMini((*matB),mid1+1,mid2+1,(*matB).row-1,(*matB).column-1);

    Matrix s1=b12-b22;
    Matrix s2=a11+a12;
    Matrix s3=a21+a22;
    Matrix s4=b21-b11;
    Matrix s5=a11+a22;
```

```
    Matrix s6=b11+b22;
    Matrix s7=a12-a22;
    Matrix s8=b21+b22;
    Matrix s9=a11-a21;
    Matrix s10=b11+b12;


    Matrix p1=strassen(&a11,&s1);
    Matrix p2=strassen(&s2,&b22);
    Matrix p3=strassen(&s3,&b11);
    Matrix p4=strassen(&a22,&s4);
    Matrix p5=strassen(&s5,&s6);
    Matrix p6=strassen(&s7,&s8);
    Matrix p7=strassen(&s9,&s10);


    Matrix c11=p5+p4-p2+p6;
    Matrix c12=p1+p2;
    Matrix c21=p3+p4;
    Matrix c22=p5+p1-p3-p7;

    return combine(c11,c12,c21,c22);
}

int n;
int main() {
    std::ios::sync_with_stdio(false); //关闭同步，提高效率
    ifstream fin("data.txt");
    ofstream fout("result.txt");
    fin>>n;
    Matrix m1=Matrix(n,n);
    Matrix m2=Matrix(n,n);
    Matrix mr=Matrix(n,n);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            fin>>m1.m[i][j];
        }
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            fin>>m2.m[i][j];
        }
    }
    clock_t start, stop;
    start=clock();
```

```
    mr=strassen(&m1,&m2);
    stop=clock();
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            fout<<mr.m[i][j]<<" ";
        }
        fout<<endl;
    }
    cout<<"Time: "<<1000*((double)(stop - start) / CLOCKS_PER_SEC)<<"ms"<<
endl;
    fin.close();
    fout.close();
    return 0;
}
```

3. 拓展 Strassen(能用于数据规模 N 不是 2 的幂次方)

strassen1.cpp

```
//code of strassen is complex, but I can make it!
//在完成这个实验的过程中，我通过构造结构体包装相关的功能
//这个适应了矩阵规模 N 不是 2 的幂的情况
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
#include<ctime>
#include<cstdio>
#include<cstring>
using namespace std;
struct Matrix {
    int row, column;//row*column 的矩阵
    int** m; //用来存那个数组
    Matrix(int r, int c) {//初始化
        row=r;
        column=c;
        m=(int**)malloc(sizeof(int*)*r);
        for (int i = 0; i < r; i++)m[i] = (int*)malloc(sizeof(int) * c);
        /*
        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++){
                m[i][j]=0;
            }
        }
        */
    }
```

```
~Matrix(){ //析构函数，销毁时用
    if (m != NULL) {
        for (int i=0;i<row;i++) {
            delete[] m[i];
        }
        delete[] m;
    }
}
Matrix(const Matrix& mat) {
    row=mat.row;
    column=mat.column;
    m=(int**)malloc(sizeof(int*)* mat.row);
    for (int i=0;i<mat.row;i++)
        m[i]=(int*)malloc(sizeof(int) * mat.column);
    for (int i=0;i<row;i++){
        for (int j=0;j<column;j++){
            m[i][j]=mat.m[i][j];
        }
    }
}

Matrix& operator = (const Matrix& mat) {
    //重载=
    if (this != &mat) {
        row = mat.row;
        column = mat.column;
        m = (int**)malloc(sizeof(int*) * mat.row);
        for (int i = 0; i < mat.row; i++)
            m[i] = (int*)malloc(sizeof(int) * mat.column);
        for (int i = 0; i < row; i++){
            for (int j = 0; j < column; j++){
                m[i][j] = mat.m[i][j];
            }
        }
    }
    return *this;
}

Matrix operator + (const Matrix& mat) {
    //重载+
    Matrix result=Matrix(row,column);
    for(int i=0;i<row;i++){
        for(int j=0;j<column;j++){
            result.m[i][j]=(*this).m[i][j]+mat.m[i][j];
```

```
            }
        }
        return result;
    }

    Matrix operator - (const Matrix& mat) {
        //重载-
        Matrix result=Matrix(row,column);
        for(int i=0;i<row;i++){
            for(int j=0;j<column;j++){
                result.m[i][j]=(*this).m[i][j]-mat.m[i][j];
            }
        }
        return result;
    }


};
Matrix matExtend(Matrix* mat){
    int pos=0,exn;
    int n=(*mat).row;
    while(n){
        pos++;
        n>>=1;
    }
    if(((*mat).row & ((*mat).row - 1)) == 0)exn=(*mat).row;
    else exn=1<<pos;
    Matrix matC=Matrix(exn,exn);

    for (int i=0;i<(*mat).row;i++){
        for(int j=0;j<(*mat).column; j++){
            matC.m[i][j]=(*mat).m[i][j];
        }
    }
    for(int i=(*mat).row;i<exn;i++){
        for(int j=0;j<exn;j++){
            matC.m[i][j]=0;
        }
    }
    for(int i=0;i<(*mat).row;i++){
        for(int j=(*mat).column;j<exn;j++){
            matC.m[i][j]=0;
        }
    }
    return matC;
```

```
}
Matrix matMini(Matrix mat, int st1, int st2, int ed1, int ed2) {
    //from(st1,st2)to(ed1,ed2)
    Matrix matC=Matrix(ed1-st1+1, ed2-st2+1);
    for (int i=0;i<=(ed1-st1);i++){
        for (int j=0;j<=(ed2-st2);j++) {
            matC.m[i][j]=mat.m[st1+i][st2+j];
        }
    }
    return matC;
}
Matrix combine(Matrix mat1, Matrix mat2, Matrix mat3, Matrix mat4) {
    Matrix newmat=Matrix(mat1.row+mat3.row,mat1.column+mat2.column);
    for (int i=0;i<mat1.row;i++){
        for (int j=0;j<mat1.column;j++){
            newmat.m[i][j]=mat1.m[i][j];
        }
    }

    for (int i=0;i<mat2.row;i++){
        for (int j=0;j<mat2.column;j++){
            newmat.m[i][j+mat1.column]=mat2.m[i][j];
        }
    }

    for (int i=0;i<mat3.row;i++){
        for (int j=0;j<mat3.column;j++){
            newmat.m[mat1.row+i][j]=mat3.m[i][j];
        }
    }

    for (int i=0;i<mat4.row;i++){
        for (int j=0;j<mat4.column;j++){
            newmat.m[mat1.row+i][mat1.column+j]=mat4.m[i][j];
        }
    }
    return newmat;
}

Matrix strassen(Matrix* matA, Matrix* matB) {
    if ((*matA).row == 1 && (*matA).column == 1 && (*matB).row == 1 &&
(*matB).column == 1) {
        Matrix matC=Matrix(1, 1);
        matC.m[0][0]=(*matA).m[0][0]*(*matB).m[0][0];
```

```
        return matC;
    }
    int mid1=(*matA).row/2 - 1;
    int mid2=(*matA).column/2 - 1;

    Matrix a11=matMini((*matA),0,0,mid1,mid2);
    Matrix a12=matMini((*matA),0, mid2+1,mid1,(*matA).column- 1);
    Matrix a21=matMini((*matA),mid1+1,0,(*matA).row-1,mid2);
    Matrix
a22=matMini((*matA),mid1+1,mid2+1,(*matA).row-1,(*matA).column-1);

    Matrix b11=matMini((*matB),0,0,mid1,mid2);
    Matrix b12=matMini((*matB),0, mid2+1,mid1,(*matB).column- 1);
    Matrix b21=matMini((*matB),mid1+1,0,(*matB).row-1,mid2);
    Matrix
b22=matMini((*matB),mid1+1,mid2+1,(*matB).row-1,(*matB).column-1);

    Matrix s1=b12-b22;
    Matrix s2=a11+a12;
    Matrix s3=a21+a22;
    Matrix s4=b21-b11;
    Matrix s5=a11+a22;
    Matrix s6=b11+b22;
    Matrix s7=a12-a22;
    Matrix s8=b21+b22;
    Matrix s9=a11-a21;
    Matrix s10=b11+b12;


    Matrix p1=strassen(&a11,&s1);
    Matrix p2=strassen(&s2,&b22);
    Matrix p3=strassen(&s3,&b11);
    Matrix p4=strassen(&a22,&s4);
    Matrix p5=strassen(&s5,&s6);
    Matrix p6=strassen(&s7,&s8);
    Matrix p7=strassen(&s9,&s10);


    Matrix c11=p5+p4-p2+p6;
    Matrix c12=p1+p2;
    Matrix c21=p3+p4;
    Matrix c22=p5+p1-p3-p7;

    return combine(c11,c12,c21,c22);
}
```

```
int n;
int main() {
    std::ios::sync_with_stdio(false);
    ifstream fin("data.txt");
    ofstream fout("result.txt");
    fin>>n;
    Matrix m1=Matrix(n,n);
    Matrix m2=Matrix(n,n);
    Matrix mr=Matrix(n,n);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            fin>>m1.m[i][j];
        }
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            fin>>m2.m[i][j];
        }
    }
    //我怎样解决输入的矩阵规模 N 不是 2 的幂的情况？
    //就正常思路，把这个一般矩阵拓展成了矩阵规模 N 是 2 的幂的情况就行
    Matrix m1e=matExtend(&m1);
    Matrix m2e=matExtend(&m2);
    clock_t start, stop;
    start=clock();
    mr=strassen(&m1e,&m2e);
    stop=clock();
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            fout<<mr.m[i][j]<<" ";
        }
        fout<<endl;
    }
    cout<<"Time: "<<1000*((double)(stop - start) / CLOCKS_PER_SEC)<<"ms"<<
endl;
    fin.close();
    fout.close();
    return 0;
}
```

4. 拓展 Strassen(用于提交到学校 OJ 进行测试)

strassen2.cpp

```
//code of strassen is complex, but I can make it!
```

```cpp
//在完成这个实验的过程中，我通过构造结构体包装相关的功能
//这个通过修改 strassen.cpp,适应了矩阵规模 N 不是 2 的幂的情况
//这个是用来完成 OJ 上题目的版本
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
#include<ctime>
#include<cstdio>
#include<cstring>
using namespace std;
struct Matrix {
    int row, column;//row*column 的矩阵
    int** m; //用来存那个数组
    Matrix(int r, int c) {//初始化
        row=r;
        column=c;
        m=(int**)malloc(sizeof(int*)*r);
        for (int i = 0; i < r; i++)m[i] = (int*)malloc(sizeof(int) * c);
        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++){
                m[i][j]=0;
            }
        }
    }

    ~Matrix(){ //析构函数，销毁时用
        if (m != NULL) {
            for (int i=0;i<row;i++) {
                delete[] m[i];
            }
            delete[] m;
        }
    }
    Matrix(const Matrix& mat) {
        row=mat.row;
        column=mat.column;
        m=(int**)malloc(sizeof(int*)* mat.row);
        for (int i=0;i<mat.row;i++)
            m[i]=(int*)malloc(sizeof(int) * mat.column);
        for (int i=0;i<row;i++){
            for (int j=0;j<column;j++){
                m[i][j]=mat.m[i][j];
            }
```

```
        }
    }

    Matrix& operator = (const Matrix& mat) {
        //重载=
        if (this != &mat) {
            row = mat.row;
            column = mat.column;
            m = (int**)malloc(sizeof(int*) * mat.row);
            for (int i = 0; i < mat.row; i++)
                m[i] = (int*)malloc(sizeof(int) * mat.column);
            for (int i = 0; i < row; i++){
                for (int j = 0; j < column; j++){
                    m[i][j] = mat.m[i][j];
                }
            }
        }
        return *this;
    }

    Matrix operator + (const Matrix& mat) {
        //重载+
        Matrix result=Matrix(row,column);
        for(int i=0;i<row;i++){
            for(int j=0;j<column;j++){
                result.m[i][j]=(*this).m[i][j]+mat.m[i][j];
            }
        }
        return result;
    }

    Matrix operator - (const Matrix& mat) {
        //重载-
        Matrix result=Matrix(row,column);
        for(int i=0;i<row;i++){
            for(int j=0;j<column;j++){
                result.m[i][j]=(*this).m[i][j]-mat.m[i][j];
            }
        }
        return result;
    }


};
Matrix matExtend(Matrix* mat){
```

```
    int pos=0,exn;
    int n=(*mat).row;
    while(n){
        pos++;
        n>>=1;
    }
    if(((*mat).row & ((*mat).row - 1)) == 0)exn=(*mat).row;
    else exn=1<<pos;
    Matrix matC=Matrix(exn,exn);

    for (int i=0;i<(*mat).row;i++){
        for(int j=0;j<(*mat).column; j++){
            matC.m[i][j]=(*mat).m[i][j];
        }
    }
    for(int i=(*mat).row;i<exn;i++){
        for(int j=0;j<exn;j++){
            matC.m[i][j]=0;
        }
    }
    for(int i=0;i<(*mat).row;i++){
        for(int j=(*mat).column;j<exn;j++){
            matC.m[i][j]=0;
        }
    }
    return matC;
}

Matrix matMini(Matrix mat, int st1, int st2, int ed1, int ed2) {
    //from(st1,st2)to(ed1,ed2)
    Matrix matC=Matrix(ed1-st1+1, ed2-st2+1);
    for (int i=0;i<=(ed1-st1);i++){
        for (int j=0;j<=(ed2-st2);j++) {
            matC.m[i][j]=mat.m[st1+i][st2+j];
        }
    }
    return matC;
}
Matrix combine(Matrix mat1, Matrix mat2, Matrix mat3, Matrix mat4) {
    Matrix newmat=Matrix(mat1.row+mat3.row,mat1.column+mat2.column);
    for (int i=0;i<mat1.row;i++){
        for (int j=0;j<mat1.column;j++){
            newmat.m[i][j]=mat1.m[i][j];
        }
```

```
    }

    for (int i=0;i<mat2.row;i++){
        for (int j=0;j<mat2.column;j++){
            newmat.m[i][j+mat1.column]=mat2.m[i][j];
        }
    }

    for (int i=0;i<mat3.row;i++){
        for (int j=0;j<mat3.column;j++){
            newmat.m[mat1.row+i][j]=mat3.m[i][j];
        }
    }

    for (int i=0;i<mat4.row;i++){
        for (int j=0;j<mat4.column;j++){
            newmat.m[mat1.row+i][mat1.column+j]=mat4.m[i][j];
        }
    }
    return newmat;
}

Matrix strassen(Matrix* matA, Matrix* matB) {
    if ((*matA).row == 1 && (*matA).column == 1 && (*matB).row == 1 &&
(*matB).column == 1) {
        Matrix matC=Matrix(1, 1);
        matC.m[0][0]=(*matA).m[0][0]*(*matB).m[0][0];
        return matC;
    }
    int mid1=(*matA).row/2 - 1;
    int mid2=(*matA).column/2 - 1;

    Matrix a11=matMini((*matA),0,0,mid1,mid2);
    Matrix a12=matMini((*matA),0, mid2+1,mid1,(*matA).column- 1);
    Matrix a21=matMini((*matA),mid1+1,0,(*matA).row-1,mid2);
    Matrix
a22=matMini((*matA),mid1+1,mid2+1,(*matA).row-1,(*matA).column-1);

    Matrix b11=matMini((*matB),0,0,mid1,mid2);
    Matrix b12=matMini((*matB),0, mid2+1,mid1,(*matB).column- 1);
    Matrix b21=matMini((*matB),mid1+1,0,(*matB).row-1,mid2);
    Matrix
b22=matMini((*matB),mid1+1,mid2+1,(*matB).row-1,(*matB).column-1);
```

```
    Matrix s1=b12-b22;
    Matrix s2=a11+a12;
    Matrix s3=a21+a22;
    Matrix s4=b21-b11;
    Matrix s5=a11+a22;
    Matrix s6=b11+b22;
    Matrix s7=a12-a22;
    Matrix s8=b21+b22;
    Matrix s9=a11-a21;
    Matrix s10=b11+b12;


    Matrix p1=strassen(&a11,&s1);
    Matrix p2=strassen(&s2,&b22);
    Matrix p3=strassen(&s3,&b11);
    Matrix p4=strassen(&a22,&s4);
    Matrix p5=strassen(&s5,&s6);
    Matrix p6=strassen(&s7,&s8);
    Matrix p7=strassen(&s9,&s10);


    Matrix c11=p5+p4-p2+p6;
    Matrix c12=p1+p2;
    Matrix c21=p3+p4;
    Matrix c22=p5+p1-p3-p7;

    return combine(c11,c12,c21,c22);
}

Matrix normal(Matrix* matA, Matrix* matB){
    //这里涉及到一个问题，我们以什么顺序进行运算？
    //根据《深入理解计算机系统》(CSAPP 第三版，机械工业出版社)Page448-450
    //我们可以知道采用 kij 或者 ikj 版本的不命中率最低，时间效率最高
    //我在这个问题中采用 kij 版本书写代码
    Matrix matC=Matrix(matA->row,matB->column);
    int tmp;
    for(int k=0;k<(matA->column);k++){
        for(int i=0;i<(matA->row);i++){
            tmp=(*matA).m[i][k];
            for(int j=0;j<(matB->column);j++){
                matC.m[i][j]+=tmp*((*matB).m[k][j]);
            }
        }
    }
    return matC;
```

```
}
int T,n;
int main() {
    std::ios::sync_with_stdio(false);
    cin>>T>>n;
    Matrix m1=Matrix(n,n);
    Matrix m2=Matrix(n,n);
    Matrix mr=Matrix(n,n);
    //我怎样解决输入的矩阵规模 N 不是 2 的幂的情况？
    //就正常思路，把这个一般矩阵拓展成了矩阵规模 N 是 2 的幂的情况就行
    if(n<=512){
        while(T){
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    cin>>m1.m[i][j];
                }
            }
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    cin>>m2.m[i][j];
                }
            }
            mr=normal(&m1,&m2);
            for(int i=0;i<n;i++){
                for(int j=0;j<n-1;j++){
                    cout<<mr.m[i][j]<<" ";
                }
                cout<<mr.m[i][n-1]<<endl;
            }
            T--;
        }
    }else{
        while(T){
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    cin>>m1.m[i][j];
                }
            }
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    cin>>m2.m[i][j];
                }
            }
            Matrix m1e=matExtend(&m1);
```

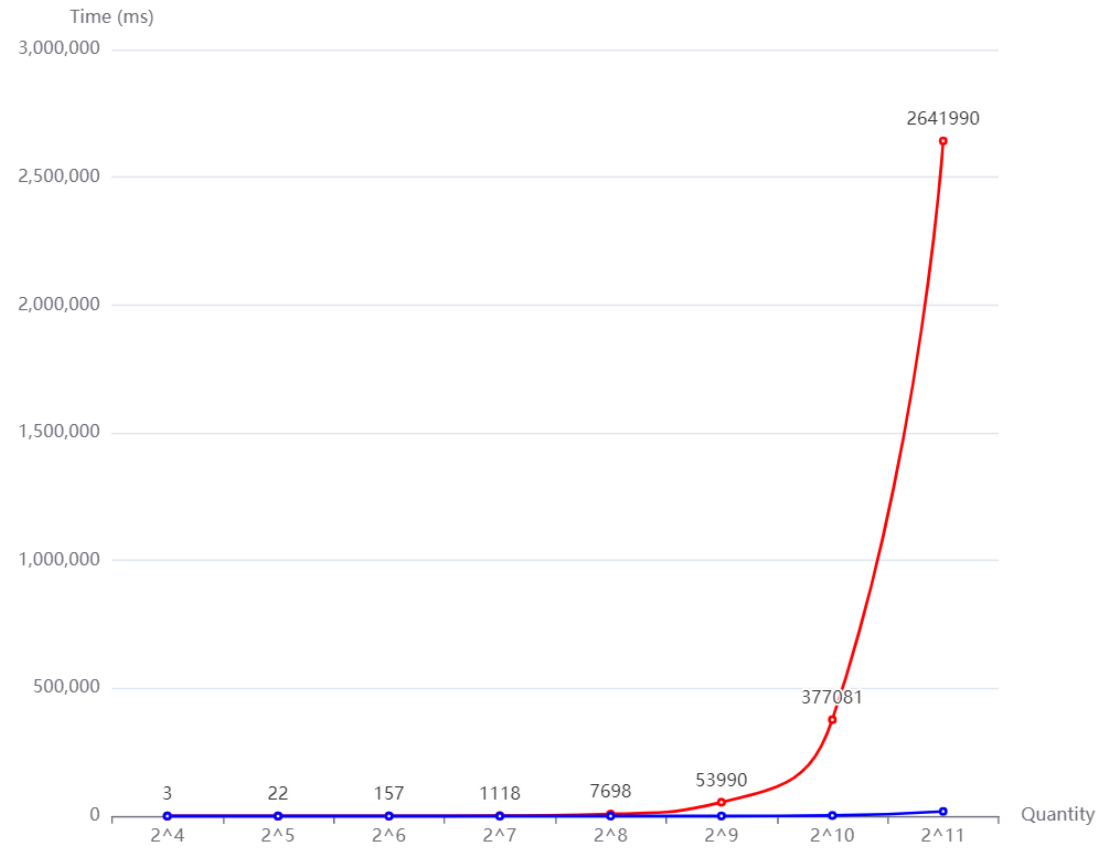
```
            Matrix m2e=matExtend(&m2);
            mr=strassen(&m1e,&m2e);
            for(int i=0;i<n;i++){
                for(int j=0;j<n-1;j++){
                    cout<<mr.m[i][j]<<" ";
                }
                cout<<mr.m[i][n-1]<<endl;
            }
            T--;
        }
    }
    return 0;
}
```

代码将会随压缩包一并提交

五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。

通过运行我上述的程序，得到了数据。为了便于作图，我还测量了 N=2^5,2^6,2^7 规模下的运行时间。整合数据后将其输入 echart 折线图，我得到了下图所示的折线图(已平滑处理了)



其中红线为 strassen 算法，蓝线为朴素矩阵乘法。

通过分析具体的数据，我发现虽然对于数据规模的增长，运行时间增长的比例和理论分析近似；但是由于 strassen 算法复杂度前较高的复杂度常数，即使面对数量较大的数据，时间复杂度也是惊人的。虽然理论分析地非常好，但是在实际的运行过程中，我们需要结合实际、不断优化，这样才可能较好地利用计算机资源。

该实验是艰巨的，面对庞大代码量的代码，调试是十分艰难的。由于自己对面向对象还不是那么的熟悉，因此本实验中我采用了自己较为熟悉的结构体来包装矩阵。同时矩阵加法、减法和数值加减法是不一样的，但是遵循的是类似的步骤，因此我采用了重载运算符使代码更加精简有效。最后，正确高效地释放内存在本实验中是必不可少的。初期自己的析构函数写错了，引起了内存泄漏，小规模的数据就占用了几个 GB 的内存空间，这显然是无法接受的。

看着自己写过的代码，回顾这次的调试流程，我感到满满的成就感，也希望在未来的学习中更好地将理论和实际结合起来。