第四章 strassen算法

```
#include <iostream>

using namespace std;

//STRASSEN矩阵乘法算法

const int N = 8; //常量N用来定义矩阵的大小


/*

* 矩阵的加法运算

*/

void Add(int** matrixA, int** matrixB, int** matrixResult, int length)

{

  for (int i = 0; i < length; i++) {

    for (int j = 0; j < length; j++) {

      matrixResult[i][j] = matrixA[i][j] + matrixB[i][j];

    }

  }

}


/*

* 矩阵的减法运算

*/

void Sub(int** matrixA, int** matrixB, int** matrixResult, int length)

{

  for (int i = 0; i < length; i++) {

    for (int j = 0; j < length; j++) {

      matrixResult[i][j] = matrixA[i][j] - matrixB[i][j];

    }

  }

}


/*

* 矩阵乘法

*/

void Mul(int** matrixA, int** matrixB, int** matrixResult){

  for (int i = 0; i < 2; ++i) {

    for (int j = 0; j < 2; ++j) {

      matrixResult[i][j] = 0;

      for (int k = 0; k < 2; ++k) {
           matrixResult[i][j] += matrixA[i][k] * matrixB[k][j];
      }
    }
  }
}
void Strassen(int** matrixA, int** matrixB, int** matrixResult, int length)
{
  int halfLength = length / 2;
  int** a11 = new int*[halfLength];
  int** a12 = new int*[halfLength];
  int** a21 = new int*[halfLength];
  int** a22 = new int*[halfLength];

  int** b11 = new int*[halfLength];
  int** b12 = new int*[halfLength];
  int** b21 = new int*[halfLength];
  int** b22 = new int*[halfLength];

  int** s1 = new int*[halfLength];
  int** s2 = new int*[halfLength];
  int** s3 = new int*[halfLength];
  int** s4 = new int*[halfLength];
  int** s5 = new int*[halfLength];
  int** s6 = new int*[halfLength];
  int** s7 = new int*[halfLength];

  int** matrixResult11 = new int*[halfLength];
  int** matrixResult12 = new int*[halfLength];
  int** matrixResult21 = new int*[halfLength];
  int** matrixResult22 = new int*[halfLength];

  int** temp = new int*[halfLength];
  int** temp1 = new int*[halfLength];
  if (halfLength == 1){
    Mul(matrixA, matrixB, matrixResult);
  }
  else{
    //首先将矩阵A，B 分为4块
    for (int i = 0; i < halfLength; i++) {
      a11[i] = new int[halfLength];
```

```
    a12[i] = new int[halfLength];
    a21[i] = new int[halfLength];
    a22[i] = new int[halfLength];

    b11[i] = new int[halfLength];
    b12[i] = new int[halfLength];
    b21[i] = new int[halfLength];
    b22[i] = new int[halfLength];

    s1[i] = new int[halfLength];
    s2[i] = new int[halfLength];
    s3[i] = new int[halfLength];
    s4[i] = new int[halfLength];
    s5[i] = new int[halfLength];
    s6[i] = new int[halfLength];
    s7[i] = new int[halfLength];

    matrixResult11[i] = new int[halfLength];
    matrixResult12[i] = new int[halfLength];
    matrixResult21[i] = new int[halfLength];
    matrixResult22[i] = new int[halfLength];

    temp[i] = new int[halfLength];
    temp1[i] = new int[halfLength];
    for (int j = 0; j < halfLength; j++) {
        a11[i][j] = matrixA[i][j];
        a12[i][j] = matrixA[i][j + halfLength];
        a21[i][j] = matrixA[i + halfLength][j];
        a22[i][j] = matrixA[i + halfLength][j + halfLength];
        b11[i][j] = matrixB[i][j];
        b12[i][j] = matrixB[i][j + halfLength];
        b21[i][j] = matrixB[i + halfLength][j];
        b22[i][j] = matrixB[i + halfLength][j + halfLength];
    }
}

//计算s1
Sub(b12, b22, temp, halfLength);
Strassen(a11, temp, s1, halfLength);
//计算s2
```

```
Add(a11, a12, temp, halfLength);
Strassen(temp, b22, s2, halfLength);
//计算s3
Add(a21, a22, temp, halfLength);
Strassen(temp, b11, s3, halfLength);
//计算s4
Sub(b21, b11, temp, halfLength);
Strassen(a22, temp, s4, halfLength);
//计算s5
Add(a11, a22, temp1, halfLength);
Add(b11, b22, temp, halfLength);
Strassen(temp1, temp, s5, halfLength);
//计算s6
Sub(a12, a22, temp1, halfLength);
Add(b21, b22, temp, halfLength);
Strassen(temp1, temp, s6, halfLength);
//计算s7
Sub(a11, a21, temp1, halfLength);
Add(b11, b12, temp, halfLength);
Strassen(temp1, temp, s7, halfLength);

//计算matrixResult11
Add(s5, s4, temp1, halfLength);
Sub(temp1, s2, temp, halfLength);
Add(temp, s6, matrixResult11, halfLength);
//计算matrixResult12
Add(s1, s2, matrixResult12, halfLength);
//计算matrixResult21
Add(s3, s4, matrixResult21, halfLength);
//计算matrixResult22
Add(s5, s1, temp1, halfLength);
Sub(temp1, s3, temp, halfLength);
Sub(temp, s7, matrixResult22, halfLength);

//结果送回matrixResult中
for (int i = 0; i < halfLength; i++) {
  for (int j = 0; j < halfLength; j++) {
    matrixResult[i][j] = matrixResult11[i][j];
    matrixResult[i][j + halfLength] = matrixResult12[i][j];
    matrixResult[i + halfLength][j] = matrixResult21[i][j];
```

```
        matrixResult[i + halfLength][j + halfLength] = matrixResult22[i][j];
      }
      delete(a11[i]);
      delete(a12[i]);
      delete(a21[i]);
      delete(a22[i]);

      delete(b11[i]);
      delete(b12[i]);
      delete(b21[i]);
      delete(b22[i]);

      delete(s1[i]);
      delete(s2[i]);
      delete(s3[i]);
      delete(s4[i]);
      delete(s5[i]);
      delete(s6[i]);
      delete(s7[i]);

      delete(matrixResult11[i]);
      delete(matrixResult12[i]);
      delete(matrixResult21[i]);
      delete(matrixResult22[i]);

      delete(temp[i]);
      delete(temp1[i]);
    }
    delete(a11);
    delete(a12);
    delete(a21);
    delete(a22);

    delete(b11);
    delete(b12);
    delete(b21);
    delete(b22);

    delete(s1);
    delete(s2);
    delete(s3);
    delete(s4);
    delete(s5);
    delete(s6);
    delete(s7);

    delete(matrixResult11);
    delete(matrixResult12);
    delete(matrixResult21);
    delete(matrixResult22);

    delete(temp);
    delete(temp1);
  }
}
int main()
{
  int n;
  cout << "输入矩阵行列数(2的幂指数倍)：" << endl;
  cin >> n;
  int **a = new int*[n];
  int **b = new int*[n];
  int **c = new int*[n];
  for (int i = 0; i < n; i++)
  {
    a[i] = new int[n];
    b[i] = new int[n];
    c[i] = new int[n];
  }
  cout << "输入第一个矩阵的数字：" << endl;
  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
      cin >> a[i][j];
    }
  }
  cout << "输入第二个矩阵的数字：" << endl;
  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
      cin >> b[i][j];
    }
    }
    Strassen(a, b, c, n);
    cout << "结果矩阵为：" << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            cout << c[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```