C2-B题题解

题目描述

给定两个 n 阶整数方阵 A,B,试求 A×B。

输入

本题测试点包含多组数据。

第一行，一个正整数 T（1≤T≤10），表示数据组数。

对于每组数据：

第一行，一个正整数 n（1≤n≤200），表示方阵阶数。

接下来 n行，每行 n个整数 A_{ij}（-10^6≤ A_{ij}≤10^6），表示整数方阵 A。

接下来 n行，每行 n 个整数 B^{ij}(-10^6≤Bij≤10^6），表示整数方阵 B。

输出

对于每组数据：

输出 n行，每行 n个整数 $(AB)^{ij}$，表示方阵 A×B。

输入样例

```
1
3
1 0 0
0 1 0
0 0 1
0 1 0
0 0 1
1 0 0
```

输出样例

```
0 1 0
0 0 1
1 0 0
```

解题思路

显然，本题是一道标准的矩阵（方阵）乘法模版题。

让我们先回忆一下举证乘法公式：

$$c[i][j] = \sum_{k=1}^{n} a[i][k] * b[k][j], 1 \leq i \leq n, 1 \leq j \leq n$$

注意到本题的n<=200，因此我们可以采用最简单的方法——三重循环实现方阵的乘法。

注意事项

本题数据范围是 -10^6~10^6 ，当两数相乘时，可能超过 int 范围，故需要使用 long long 类型。

核心代码

三重循环实现方阵的乘法

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            c[i][j] += a[i][k] * b[k][j];
```

代码优化

我们不妨先试着将方阵乘法的循环次序改变一下，得到以下两种方案：

- ijk方案——原方案：

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            c[i][j] += a[i][k] * b[k][j];
```

- ikj方案：

```
for (int i = 0; i < n; i++)
        for (int k = 0; k < n; k++)
            for (int j = 0; j < n; j++)
                c[i][j] += a[i][k] * b[k][j];
```

接着我们对这两种方案进行测试，通过在不同的维度下测试两个函数的大致运行时间（数据来自网络）：

- ijk方法下的测量结果(单位：ms)

n	1	2	3	4
100	6	3	7	4
300	104	90	95	92
500	474	474	484	469
800	2001	2100	2100	2042
1200	10615	9821	9689	9703
2000	63677	62797	62698	62551

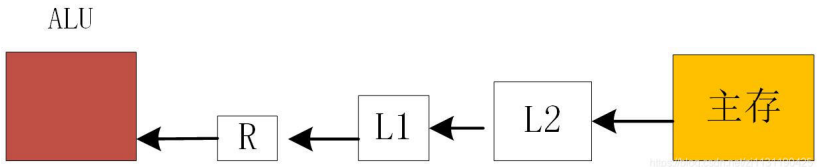
- ikj方法下的测量结果(单位：ms)

n	1	2	3	4
100	5	3	3	3
30077	79	79	81	78
500	328	351	341	359
800	1435	1413	1423	1453
1200	4901	4778	4849	4694
2000	21664	21501	21968	21947

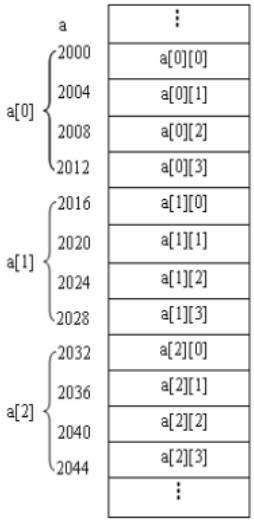
对比可知，在ikj的方案下，矩阵乘法的运算速度较快，而且在矩阵阶数n越大的时候，这种差别越是明显。在计算矩阵乘法的过程中，三层的循环嵌套共有六种排列方式，虽然在每种嵌套方式下，都要执行同样数量的操作，但是花费的时间是不同的。

那么原因呢?

因为在不同的嵌套方式下，改变了数据的访问模式，进而改变了缓存未命中的数量。最终影响了运行时间。
在程序开始运行时，数据都位于主存中，需要将参与运算的数据从主存移到寄存器再进行运算。如果需要的数据没有在一级缓存，而是在二级缓存，而需要将数据存二级缓存移动到一级缓存，这称为一级缓存未命中，当需要的数据没有在二级缓存中时，此时为二级缓存未命中，则需要将数据从主存移动到二级缓存，再移动到一级缓存。**所以可以通过减少缓存未命中的数量，提高程序的运行效率。**



或者可以简单理解为，ijk方案的**列号在外层循环**，每一次内层循环中行号先改变，也就是说每求完一个乘积都会重新读取一行。而按列存储的矩阵的每一行在内存中都是不连续的，所以行访问会很慢，这样反复多次进行行访问就会大大拖慢运行速度。
而ikj方案的**行号在外层循环**，每求完一次乘积会重读一列，但之前读出的一行是不变的。这时需要的那一行可能还在Cache里，所以可以不需要再从那段不连续的内存中读取了，速度自然更快。



完整代码

```c
#include <stdio.h>
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int n;
        long long a[210][210] = {0}, b[210][210] = {0}, c[210][210] = {0};
        scanf("%d", &n);
        // 输入A、B矩阵
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                scanf("%lld", &a[i][j]);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                scanf("%lld", &b[i][j]);
        // 三重循环实现矩阵相乘
        for (int i = 0; i < n; i++)
            for (int k = 0; k < n; k++)
                for (int j = 0; j < n; j++)
                    c[i][j] += a[i][k] * b[k][j];
        // 输出结果
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                printf("%lld ", c[i][j]);
            printf("\n");
        }
    }
}
```