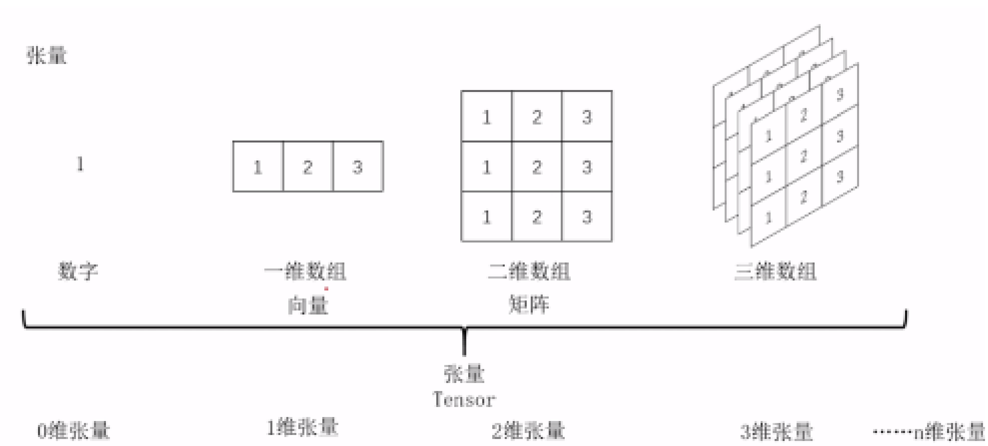
OJ10:张量相乘的最小开销问题

Yanxu Chen,January 6th,2024

[Description]

张量（tensor）乘法和广播（broadcasting）是一种在张量之间进行运算的方法，它可以用来表示一些复杂的数学和物理问题，例如神经网络，图像处理，信号处理等。为了理解张量乘法和广播，我们首先需要了解什么是张量，以及它的形状和维度。

张量（tensor）是一种可以表示多维数组的数据结构，它可以有任意的维度和形状。维度（dimension）是张量的层次，表示张量有多少个方向或轴（axis）。形状（shape）是一个表示每个维度大小的整数元组，表示张量在每个方向上有多少个元素。例如，一个标量（scalar）是一个零维张量，它只有一个数值，没有方向，也没有形状；一个向量（vector）是一个一维张量，它有一个方向，也就是一个轴，它的形状是一个单元素的元组，表示它在这个方向上有多少个元素；一个矩阵（matrix）是一个二维张量，它有两个方向，也就是两个轴，它的形状是一个双元素的元组，表示它在这两个方向上分别有多少个元素；一个立方体（cube）是一个三维张量，它有三个方向，也就是三个轴，它的形状是一个三元素的元组，表示它在这三个方向上分别有多少个元素，以此类推。我们可以用以下的图示来表示不同维度的张量：



其中0维张量可用一个可表示为标量，1维张量可表示为向量，2维张量可表示为矩阵，更高维的张量可视为由低维张量作为元素构成的向量、矩阵等：

如3维张量可表示为
$$\begin{bmatrix} \begin{bmatrix} b_{111} & b_{112} & b_{113} \\ b_{121} & b_{122} & b_{123} \end{bmatrix} \\ \begin{bmatrix} b_{211} & b_{212} & b_{213} \\ b_{221} & b_{222} & b_{223} \end{bmatrix} \end{bmatrix}$$
。

在张量之间进行运算时，我们需要考虑它们的形状是否匹配，以及是否需要进行广播（broadcasting）。广播是一种在支持张量的框架中，如Numpy和Pytorch，为了应对形状不同的张量进行运算所执行的操作。广播的目的是将两个不同形状的张量变成两个形状相同的张量，即先对小的张量添加轴（使其维度与较大的张量相同），再把较小的张量沿着新轴重复（使其形状与较大的相同）。例如，如果我们想要对一个形状为（2，3）的矩阵和一个形状为（3）的向量进行加法，我们可以先给向量添加一个轴，使其形状变为（1，3），然后再沿着新轴复制两份，使其形状变为（2，3），最后再与矩阵逐元素相加，得到一个形状为（2，3）的矩阵。我们可以用以下的过程来表示（注意，并非任意两个张量都能够进行广播，需要形状满足特定条件，后两段具体说明）：

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + [10, 20, 30] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 10 & 20 & 30 \\ 10 & 20 & 30 \end{bmatrix} = \begin{bmatrix} 11 & 22 & 33 \\ 14 & 25 & 36 \end{bmatrix}$$

更具体而言，广播（broadcasting）是一种在支持张量的框架中，如Numpy[1] 和Pytorch[2]，为了应对形状不同但满足某些特定条件（下一段具体说明）的张量进行运算所执行的操作。广播的目的是将两个不同形状的张量变成两个形状相同的张量，即先对小的张量添加轴（使其维度与较大的张量相同），再把较小的张量沿着新轴重复（使其形状与较大的相同）。广播兼容的张量可以进行加法，乘法等运算，运算结果的形状是两个张量形状的较大者。

广播的原则是：如果两个张量的后缘维度（trailing dimension，即从末尾开始算起的维度）的轴长度相符，或其中的一方的长度为1，则认为它们是广播兼容的。广播会在缺失和（或）长度为1的维度上进行。例如，一个形状为（3，2，2，2）的4维张量A和一个形状为（1，1，2，2）的4维张量B是广播兼容的，它们相乘的过程如下所示，先将B在第一维方向上复制3份，第二维方向上复制2份，这样它的形状和A相同，之后进行逐元素乘。

$$\begin{bmatrix} \begin{bmatrix} a_{1111} & a_{1112} \\ a_{1121} & a_{1122} \end{bmatrix} & \begin{bmatrix} a_{1211} & a_{1212} \\ a_{1221} & a_{1222} \end{bmatrix} \\ \begin{bmatrix} a_{2111} & a_{2112} \\ a_{2121} & a_{2122} \end{bmatrix} & \begin{bmatrix} a_{2211} & a_{2212} \\ a_{2221} & a_{2222} \end{bmatrix} \\ \begin{bmatrix} a_{3111} & a_{3112} \\ a_{3121} & a_{3122} \end{bmatrix} & \begin{bmatrix} a_{3211} & a_{3212} \\ a_{3221} & a_{3222} \end{bmatrix} \end{bmatrix} \times \begin{bmatrix} \begin{bmatrix} b_{1111} & b_{1112} \\ b_{1121} & b_{1122} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} a_{1111} & a_{1112} \\ a_{1121} & a_{1122} \end{bmatrix} & \begin{bmatrix} a_{1211} & a_{1212} \\ a_{1221} & a_{1222} \end{bmatrix} \\ \begin{bmatrix} a_{2111} & a_{2112} \\ a_{2121} & a_{2122} \end{bmatrix} & \begin{bmatrix} a_{2211} & a_{2212} \\ a_{2221} & a_{2222} \end{bmatrix} \\ \begin{bmatrix} a_{3111} & a_{3112} \\ a_{3121} & a_{3122} \end{bmatrix} & \begin{bmatrix} a_{3211} & a_{3212} \\ a_{3221} & a_{3222} \end{bmatrix} \end{bmatrix} \times \begin{bmatrix} \begin{bmatrix} b_{1111} & b_{1112} \\ b_{1121} & b_{1122} \end{bmatrix} & \begin{bmatrix} b_{1111} & b_{1112} \\ b_{1121} & b_{1122} \end{bmatrix} \\ \begin{bmatrix} b_{1111} & b_{1112} \\ b_{1121} & b_{1122} \end{bmatrix} & \begin{bmatrix} b_{1111} & b_{1112} \\ b_{1121} & b_{1122} \end{bmatrix} \\ \begin{bmatrix} b_{1111} & b_{1112} \\ b_{1121} & b_{1122} \end{bmatrix} & \begin{bmatrix} b_{1111} & b_{1112} \\ b_{1121} & b_{1122} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} a_{1111}b_{1111} & a_{1112}b_{1112} \\ a_{1121}b_{1121} & a_{1122}b_{1122} \end{bmatrix} & \begin{bmatrix} a_{1211}b_{1111} & a_{1212}b_{1112} \\ a_{1221}b_{1121} & a_{1222}b_{1122} \end{bmatrix} \\ \begin{bmatrix} a_{2111}b_{1111} & a_{2112}b_{1112} \\ a_{2121}b_{1121} & a_{2122}b_{1122} \end{bmatrix} & \begin{bmatrix} a_{2211}b_{1111} & a_{2212}b_{1112} \\ a_{2221}b_{1121} & a_{2222}b_{1122} \end{bmatrix} \\ \begin{bmatrix} a_{3111}b_{1111} & a_{3112}b_{1112} \\ a_{3121}b_{1121} & a_{3122}b_{1122} \end{bmatrix} & \begin{bmatrix} a_{3211}b_{1111} & a_{3212}b_{1112} \\ a_{3221}b_{1121} & a_{3222}b_{1122} \end{bmatrix} \end{bmatrix}$$

结合上述张量乘法和广播机制，以及标准的线性代数中的矩阵乘法，我们考虑如下的运算：计算X1 * X2 * ... * Xn，每个Xi代表一个K维张量。有以下说明：

1.它们的维度数K一样，且大于等于2。例如都是三维张量或都是四维张量。

2.将每一个K维张量看成由矩阵（2维张量）作为元素构成的K-2维张量。前K-2维按照上述的张量乘法和广播进行运算，最后2维按照标准矩阵乘法进行运算。例如

$$\begin{bmatrix} \begin{bmatrix} a_{1111} & a_{1112} \\ a_{1121} & a_{1122} \end{bmatrix} & \begin{bmatrix} a_{1211} & a_{1212} \\ a_{1221} & a_{1222} \end{bmatrix} \\ \begin{bmatrix} a_{2111} & a_{2112} \\ a_{2121} & a_{2122} \end{bmatrix} & \begin{bmatrix} a_{2211} & a_{2212} \\ a_{2221} & a_{2222} \end{bmatrix} \\ \begin{bmatrix} a_{3111} & a_{3112} \\ a_{3121} & a_{3122} \end{bmatrix} & \begin{bmatrix} a_{3211} & a_{3212} \\ a_{3221} & a_{3222} \end{bmatrix} \end{bmatrix} * \begin{bmatrix} \begin{bmatrix} b_{1111} & b_{1112} \\ b_{1121} & b_{1122} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \mathrm{A}_{11} & \mathrm{A}_{12} \\ \mathrm{A}_{21} & \mathrm{A}_{22} \\ \mathrm{A}_{31} & \mathrm{A}_{32} \end{bmatrix} \times [B_{11}] = \begin{bmatrix} \mathrm{A}_{11}B_{11} & \mathrm{A}_{12}B_{11} \\ \mathrm{A}_{21}B_{11} & \mathrm{A}_{22}B_{11} \\ \mathrm{A}_{31}B_{11} & \mathrm{A}_{32}B_{11} \end{bmatrix}$$

其中, $\mathrm{A}_{ij} = \begin{bmatrix} a_{ij11} & a_{ij12} \\ a_{ij21} & a_{ij22} \end{bmatrix}, B_{ij} = \begin{bmatrix} b_{ij11} & b_{ij12} \\ b_{ij21} & b_{ij22} \end{bmatrix}, \mathrm{A}_{ij}B_{ij} = \begin{bmatrix} a_{ij11}b_{ij11} + a_{ij12}b_{ij21} & a_{ij11}b_{ij12} + a_{ij12}b_{ij22} \\ a_{ij21}b_{ij11} + a_{ij22}b_{ij21} & a_{ij21}b_{ij12} + a_{ij22}b_{ij22} \end{bmatrix}$

3.为满足最后两维按照标准矩阵乘法，相邻两个张量的后两维必须满足矩阵乘法的要求，即X(i)的最后一维大小必须等于X(i+1)的倒数第二维大小。

4.为满足张量乘法和广播机制的要求，对前K-2维中任意第k维，任何张量Xi在该维度的大小只能是两个取值中的一个：1或Dk。Dk为一大于1的正整数，对于不同维度k，k'，对应的Dk，Dk'可不同。前K-2维中按照可广播的逐元素乘。即相邻两个张量Xi,X(i+1)相乘时，对于任意维度k: 1≤k≤K-2，如果Xi第k维大小等于X(i+1)第k维大小, 则在该维度上逐元素相乘；如果Xi第k维大小不等于X(i+1)第k维大小，即其中一个等于1，另一个等于Dk，则进行广播并逐元素乘（将该维度等于1的张量在该维度上复制Dk份后，与另一张量在该维度上逐元素乘）。

5.定义计算开销为需要进行的标量乘法的次数。求给定n个张量依次相乘的计算开销最小的“完全括号”方案（结合律顺序）的开销。

求：计算开销最小的“完全括号”方案（结合律顺序）的开销。

参考资料：[1] https://numpy.org/

[2] https://pytorch.org/

更多参考资料：https://zhuanlan.zhihu.com/p/499189580

https://pytorch.org/docs/stable/generated/torch.bmm.html

例子

三个三维张量X1 * X2 * X3，维度大小为：X1=[1,1,2], X2=[1,2,3], X3=[10,3,4], 共有两种方案：

方案1： (X1 * X2) * X3, 计算复杂度=1 * (1 * 2 * 3)+10 * (1 * 3 * 4)=126

方案2：X1 * (X2 * X3), =10 * (2 * 3 * 4)+10 * (1 * 2 * 4)=320

方案1优于方案2，应输出126

[Input]

第一行输入两个整数n，K，代表共有n个张量相乘，每个张量都是K维。接下来n行中，每行代表一个张量，有K个由空格分隔的整数，第k个整数代表该张量第k维的大小。

[Output]

计算输入的n个张量依次相乘的计算开销最小的“完全括号”方案（结合律顺序）的开销，输出这一整数值。

[Example]

Input:

3 3
1 1 2
1 2 3
10 3 4

Output:

126

[Restrictions]

Time: 3000ms

Memory: 80000KB

[Hints]

数据范围n<2048，K<32，每个维度的大小<1000。

本题限制主要在于时间复杂度。

[Ideas]

1. 考虑使用动态规划
2. 张量的第一维是最外层的，越往后的维数对应越里层
3. 两个K维张量相乘时，设有$t_1 = (m_1, m_2, \ldots, m_{K-2}, m_{K-1}, m_K)$与$t_2 = (n_1, n_2, \ldots, n_{K-2}, n_{K-1}, n_K)$，首先考虑前K-2维，若相同位置两个张量的元素相同，则乘法次数即为这个元素；若该位置一个为1另一个为Dk，那么也就是前一个复制Dk后再乘，最后结果相同，都是乘法次数为较大的那个数。

   对于后两维，也就是普通的矩阵运算，其中$m_K = n_{K-1}$，乘法次数为$m_{K-1} * n_K$。

   两个张量相乘之后的新张量形状，对于前K-2维，每一位都是原先两个张量在该维度的较大值，后两维为矩阵相乘之后的形状。
4. 使用一个二维向量存储n个K维张量：`std::vector<std::vector<int>> tensor`，第i个张量为`tensor[i],0<=i<n`。
5. 动态规划的方法是：

   创建一个n*n的二维向量`std::vector<std::vector> dp`。`dp[i][j],i<=j`代表从第i个张量连乘到第j个张量时最小的乘法次数。那么最终问题的解就是`dp[0][n-1]`。

   初始条件：`i==j`的时候两个相同张量不需要相乘，即乘法次数为0，`dp[i][i] = 0`。

   状态转移方程：`tensor[i]`连乘到`tensor[j]`相乘的最后一步，一定是左右两部分张量各自乘法运算之后的两个新张量相乘的结果，我们将这里的分割点记为k，也就是最后的最小次数结果，一定是在某个k下，`tensor[i]`至`tensor[k]`运算后的新张量和`tensor[k+1]`至`tensor[j]`运算后的新张量，这两个新张量做乘法的次数加上已经有的乘法次数是最少的。因此我们可以写出：

`dp[i][j] = dp[i][k] + dp[k+1][j] + minTimes(i,k,j), for i<=k<j` 。这里 `minTimes(i,k,j)` 即为两个新张量做乘法的次数。

6. 不难发现，上面动态规划的方法最后构成一个n*n上三角矩阵，我们递推的顺序是从第0列开始到第n-1列，每一列从对角线上一个元素开始向上递推，因此最后想要得到 `dp[0][n-1]` 就不可避免地要求出上三角矩阵的每一个元素，时间复杂度至少为 $O(n^2)$ 。

   整体递推思路是：

```
for(int j = 1; j < n; j++){
    for(int i = j - 1; i--; i>=0){
        for(int k = i; k < j; k++){
            //求出这种分割下的乘法次数
        }
        //取最小乘法次数赋值给dp[i][j]
        dp[i][j]=...;
    }
}
```

   然后对于每一列进行操作时，我们需要知道 `tensor[i]` 到 `tensor[j]` 的前K-2维每一维度上的最大值，因为这个维度不是这个这个值就是1，一定会广播复制到这个值，也等于这一维度的乘法次数。对于每一个j的循环，i是从j-1开始逐渐减小的，因此可以在第一个循环内创建一个临时张量 `std::vector<int> temp(K)` ，初始时将 `tensor[j]` 复制给他，然后在i的循环中，i每向前一步，更新 `temp` 中的值，使得它的各个维度的大小始终是当前 `tensor[i]` 到 `tensor[j]` 的最大值。然后 `temp` 中前K-2维元素连乘，在乘上 `tensor[i][K - 2] * tensor[k][K - 1] * tensor[j][K - 1]` ，就是我们所说的 `minTimes(i,k,j)` 。对k进行循环，找到最小值即可。

   这种方法避免了i移动过程中的重复计算，时间效率较高，最后的时间复杂度应为 $O(K * n^3)$ 。

[Code]

C++:

```cpp
#include <cstdio>
#include <vector>
#include <algorithm>
// #include <cstdlib>
// #include <cmath>

int main(int argc, const char *argv[])
{
    // n个张量，每个都是K维
    // n<2048,K<32
    int n, K;
    scanf("%d%d", &n, &K);
    std::vector<std::vector<int>> tensor(n, std::vector<int>(K));
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < K; j++)
        {
            scanf("%d", &tensor[i][j]);
        }
    }

    // dp[i][j]表示从第i个张量到第j个张量运算的最小乘法次数，最后问题的解就是dp[0][n-1]
```

```cpp
    // i<=j，这是一个上三角矩阵
    // 边界条件dp[i][i]=0;
    // 状态转移方程dp[i][j]=min(dp[i][k]+dp[k+1][j]+cost(ikj)),i<=k<j
    // cost(ikj)为被第k个张量分割的前后两部分相乘的乘法次数
    std::vector<std::vector<int>> dp(n, std::vector<int>(n));
    for (int i = 0; i < n; i++)
    {
        dp[i][i] = 0;
    }
    for (int j = 1; j < n; j++)
    {
        // 对于第j列，从下向上求解
        // 预先计算一部分前K-2维相乘所需要的乘法次数
        std::vector<int> temp = tensor[j];
        for (int i = j - 1; i >= 0; i--)
        {
            // 求解dp[i][j]
            for (int x = 0; x < K - 2; x++)
            {
                // i每向前移一位，更新张量各个维度的最大值
                temp[x] = std::max(temp[x], tensor[i][x]);
            }

            int result = tensor[i][K - 2] * tensor[j][K - 1];
            for (int x = 0; x < K - 2; x++)
            {
                result *= temp[x];
            }

            int min_times = 0x7fffffff;
            for (int k = i; k < j; k++)
            {
                min_times = std::min(min_times, dp[i][k] + dp[k + 1][j] + result * tensor[k][K - 1]);
            }
            dp[i][j] = min_times;
        }
    }

    printf("%d", dp[0][n - 1]);

    return 0;
}

```

最后附上作者的通过照片:

提交详情（张量相乘的最小开销问题）

提交者: 2322022010597　　创建时间: 2024-01-24 21:46:24

运行结果　　　　分数 100.00

#	状态	时间	内存
1	Accepted	0 ms	912 KB
2	Accepted	0 ms	916 KB
3	Accepted	40 ms	1996 KB
4	Accepted	0 ms	912 KB
5	Accepted	40 ms	1992 KB
6	Accepted	0 ms	920 KB
7	Accepted	40 ms	1992 KB
8	Accepted	44 ms	2012 KB
9	Accepted	2156 ms	13740 KB
10	Accepted	2164 ms	13744 KB

[Summary]

至此，2023秋季学期数据与算法10次oj完结撒花啦~笔者也终于实现了10次oj全满分的小成就，小小夸奖一下自己！当然，这也离不开同学和朋友们之间的互相讨论和思考、大佬的指点，衷心的感谢大家的帮助！！！