一上来可能想到排序加贪心算法，可是贪心算法容易陷入局部最优，看下面例子:

100	0
60	1
59	58

100 * 60 + 59 * 1 + 58 * 0 < 100 * 59 + 60 * 58 + 1 * 0

为了防止陷入局部最优，we choose next step smartly,
我们采用dp辅助贪心的思想.

dp[i][j][k]可以转移到三种状态:dp[i + 1][j + 1][k],
dp[i][j + 1][k + 1], dp[i + 1][j][k + 1].

i -2	i - 1	i	i + 1 ▲	i + 2	i +3
j - 4	j - 3	j - 2	j - 1	j	j + 1 ▲
k - 1	k ▲	k + 1	k + 2	k + 3	k + 4

i -2	i - 1	i	i + 1 ▲	i + 2	i +3
j - 4	j - 3	j - 2	j - 1	j ▲	j + 1
k - 1	k	k + 1 ▲	k + 2	k + 3	k + 4

i -2	i - 1	i ▲	i + 1	i + 2	i +3
j - 4	j - 3	j - 2	j - 1	j	j + 1 ▲
k - 1	k	k + 1 ▲	k + 2	k + 3	k + 4

而对于同一个状态可以由不同的
状态转移而来，比如上面这个:
dp[i + 1][j + 1][k] = dp[i][j][k] +
a[0][i + 1] * a[1][j + 1]
下面这个:
dp[i'][j' + 1][k'] = dp[i'][j'][k'] +
a[1][j' + 1] * a[2][k' + 1]
可以看出这个式子等价于
dp[i + 1][j + 1][k] = dp[i + 1][j][k - 1] +
a[1][j + 1] * a[2][k]

于是为了全局最优，dp[i + 1][j + 1][k]
选取较大者,剩下的就是通过坐标替换
处理边界问题.

i'表示新一轮的循环降临到原来的i + 1上，
j' 和 k' 同理.

i' - 3 (i -2)	i' - 2 (i - 1)	i' - 1 (i)	i' ▲ (i + 1)	i' + 1 (i + 2)	i' + 2 (i +3)
j' - 4 (j - 4)	j' - 3 (j - 3)	j' - 2 (j - 2)	j' - 1 (j - 1)	j' (j)	j' + 1 ▲ (j + 1)
k' (k - 1)	k' + 1 ▲ (k)	k' + 2 (k + 1)	k' + 3 (k+ 2)	k' + 4 (k + 3)	k' + 5 (k + 4)