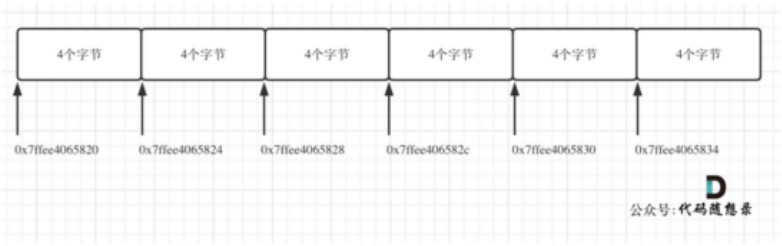
C++和C中的数组在内存中存储方式

C++和C语言中数组的内存存储方式是**连续的**，按照**行优先**的方式进行。



JAVA的二维数组在内存中存储方式

JAVA的二维数组里面存放的是一维数组的指针。所以二维数组在内存存储方式是离散的。

数组中的问题有：

区间DP：

就是dp数组的常用二维数组，第一个维度表示左边界，第二个维度表示右边界

下标及其含义为:

dp[i][j]表示[i,j]区间内满足某种条件。

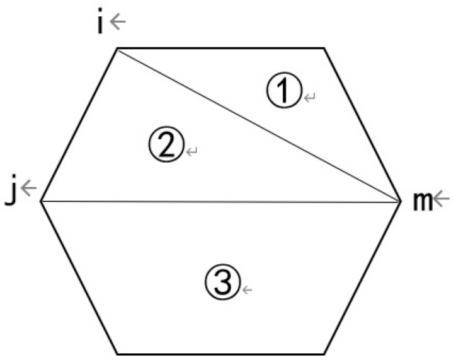
> 1039. 多边形三角剖分的最低得分
>
> 难度 中等 86
>
> 给定 `N`，想象一个凸 `N` 边多边形，其顶点按顺时针顺序依次标记为 `A[0], A[i], ..., A[N-1]`。
>
> 假设您将多边形剖分为 `N-2` 个三角形。对于每个三角形，该三角形的值是顶点标记的**乘积**，三角剖分的分数是进行三角剖分后所有 `N-2` 个三角形的值之和。
>
> 返回多边形进行三角剖分后可以得到的最低分。

dp数组及其下标含义:

> dp[i][j]表示从i到j序列的且以ii为底边最低分。我们思考的角度就是**把ij作为底边**进行**思考问题**。

递推公式

> dp[i][j]=d[i][m]+A[i]*A[j]*A[m]+dp[m][j]



遍历顺序由上方的递推公式可知

> dp[i][j]的结果来自于dp[i][m],dp[m][j]其中m<j&&m>i;
> 由于在**区间dp中一定满足i<=j的条件那么就是上三角形遍历，**
> 且本题的递推公式可知应该是**自底向上，自左向右**进行遍历，且返回dp[0][nums.size()-1].

初始化

> 对角线初始化0，且相邻元素为0

```
class Solution {
public:
    int minScoreTriangulation(vector<int>& values) {
         vector<vector<int>> dp(values.size(),vector<int>
(values.size(),0));
        //不用特意初始化了因为在dp定义时就已经将dp中的所有元素都初始化为0;
        for(int i=values.size()-3;i>=0;i--){
            for(int j=i+2;j<values.size();j++){//要加2因为相邻元素为0即[i,i
+1]为0
                //还要遍历m
                dp[i][j]=dp[i]
[i+1]+values[i]*values[i+1]*values[j]+dp[i+1][j];//计算当m为i+1
                 //为什么要单独拿出来的原因是此时是求min值但是每个单元初始化为
0
                 for(int m=i+2;m<j;m++){
                     dp[i][j]=min(dp[i][j],dp[i][m]+values[i]*values[m]*v
alues[j]+dp[m][j]);
                 }
             }
         }
        return dp[0][values.size()-1];
    }
};
```

312. 戳气球

难度 困难 719

有 `n` 个气球，编号为 `0` 到 `n - 1`，每个气球上都标有一个数字，这些数字存在数组 `nums` 中。

现在要求你戳破所有的气球。戳破第 `i` 个气球，你可以获得 `nums[i - 1] * nums[i] * nums[i + 1]` 枚硬币。 这里的 `i - 1` 和 `i + 1` 代表和 `i` 相邻的两个气球的序号。如果 `i - 1` 或 `i + 1` 超出了数组的边界，那么就当它是一个数字为 `1` 的气球。

求所能获得硬币的最大数量。

区间dp数组及其下标含义:

> dp[i][j]:数组区间为[i,j]内戳破所有气球获得最大硬币数量

递推公式

> dp[i][j]=max(dp[i][j],dp[i][m]+nums[i]*nums[m]*nums[j]+dp[m][j]);

初始化

> 对角线初始化dp[i][i]=nums[i];相邻元素为dp[i][i+1]=nums[i]*nums[i+1]+max(nums[i],nums[i+1]),其余初始化为0

递推公式:

> **从底向上,自左向右。**

```
class Solution {
public:
    int maxCoins(vector<int>& nums) {

    }
};
```

区间dp核心思想就是随着操作的进行，判断的范围越来越小，所以从小长度开始遍历，逐步化解更大范围的问题。在二层遍历中还有一个遍历就是在区间内进行找到分割点。

总结区间dp

dp数组定义都是：以[i,j]区间范围内满足某种条件

遍历顺序:先初始化对角线，且遍历的范围为上三角且**一般为**自底向上自左向右。