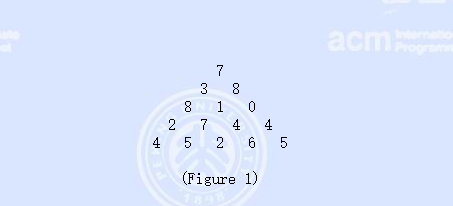
# 实验二：动态规划

**一、实验目的**

理解动态规划的基本思想，理解动态规划算法的两个基本要素最优子结构性质和子问题的重叠性质。熟练掌握典型的动态规划问题。掌握动态规划思想分析问题的一般方法，对较简单的问题能正确分析，设计出动态规划算法，并能快速编程实现。

**二、实验内容**



在上面的数字三角形中寻找一条从顶部到底边的路径，使得路径上所经过的数字之和最大。路径上的每一步都只能往左下或 右下走。只需要求出这个最大和即可，不必给出具体路径。 三角形的行数大于1小于等于100，数字为0~99

输入格式：

5 //表示三角形的行数 接下来输入三角形

7

3 8

8 1 0

2 7 4 4

4 5 2 6 5

要求输出最大和

1. **动态规划基本思想**

**（1）描述最大和路径问题的最优解结构特征**

定义一个二维数组矩阵D(r,j)来存放数字三角形,表示第r行第j个数字（r和j都从0开始算起）。

定义一个二维数组矩阵M(r,j)来存放每一个数到底边的最大和路径，其中M(r,j)表示数字D(r,j)到底边各条路径中的最大路径之和。

因此，该问题的解就变为了求M(0,0)的值。

经过总结可知，M(r,j)矩阵的最优解结构特征，由以下两种情况组成：

（1）当数字D(r,j)位于数字三角形的底边上时，M(r,j)=D(r,j)

（2）当数字D(r,j)不位于数字三角形的底边上时，则从D(i,j)出发，下一步只能走D(r+1,j)或D(r+1,j+1)。此时由M(r,j)的值为M(r+1,j)和M(r+1,j+1)中的最大值加上D(r,j)

**（2）递归地定义一个最优解的值**

根据（1）中所定义，假设D(r,j)是一个N行的矩阵，可得到如下的递归公式：

M(r,j) = D(r,j) r = N

M(r,j) = Max{ M(r+1,j), M(r+1,j+1)} + D(r,j) 其他情况

**（3）自底向上计算一个最优解的值**

从数字三角形底边开始，先初始化M(r,j) = D(r,j),再根据递归公式逐步向上递推求解，即除底边之外，每一个M(r,j)的值都由左下和右下元素中的最大值加上D(r,j)组成，即M(r,j) = Max{ M(r+1,j), M(r+1,j+1)} + D(r,j)，最后递推到M(0,0)，即为问题的解。

1. **实验过程**

**实验所用开发平台为vscode**

**实验代码描述语言为c++**

#include <iostream>

#include <algorithm>

using namespace std;

//动态规划思路

//矩阵D(r,j)存放数字三角形,D(r,j)表示第 r 行第 j 个数字

//矩阵M(r,j)存放从 D(r,j) 到 底边 各条路径的最佳路径数字之和

/\*假设D(r,j)是一个N行的矩阵，则递归公式如下:

M(r,j) = D(r,j) r = N

M(r,j) = Max{ M(r+1,j), M(r+1,j+1)} + D(r,j) 其他情况

\*/

void createDrj(int \*\*Drj, int r)

{

// 注:不能在调用函数里为二维数组指针申请内存空间,一旦函数退栈,其中的变量内存空间等都会被销毁

// 函数里所有变量空间内容等都会销毁

// Drj = (int\*\*)malloc(sizeof(int\*)\*n);

// for (int i = 0; i < n; i++)

// {

// Drj[i] = (int\*)malloc(sizeof(int)\*n);

// }

for (int i = 0; i < r; i++)

{

for (int j = 0; j <= i; j++)

{

cin >> Drj[i][j];

}

}

}

void initMrj(int \*\*Mrj, int r)

{

for (int i = 0; i < r; i++)

{

for (int j = 0; j <= i; j++)

{

Mrj[i][j] = -1;

}

}

}

void dp(int \*\*Drj, int \*\*Mrj, int n)

{

for (int i = 0; i < n; i++)

{

Mrj[n-1][i] = Drj[n-1][i];

}

for (int i = n - 2; i >= 0 ; i--)

{

for (int j = 0; j <= i; j++)

{

Mrj[i][j] = max(Mrj[i+1][j], Mrj[i+1][j+1]) + Drj[i][j];

}

}

cout << "Max sum is: " << Mrj[0][0] << endl;

}

int main()

{

int n;

int \*\*Drj;

int \*\*Mrj;

cin >> n;

Drj = (int\*\*)malloc(sizeof(int\*)\*n);

for (int i = 0; i < n; i++)

{

Drj[i] = (int\*)malloc(sizeof(int)\*n);

}

createDrj(Drj, n);

Mrj = (int \*\*)malloc(sizeof(int\*)\*n);

for (int i = 0; i < n; i++)

{

Mrj[i] = (int\*)malloc(sizeof(int)\*n);

}

initMrj(Mrj, n);

//debug

// cout <<"debug"<<endl;

// for (int i = 0; i < n; i++)

// {

// for (int j = 0; i <= i; j++)

// {

// cout << Mrj[i][j];

// }

// }

dp(Drj, Mrj, n);

system("pause");

return 0;

}

1. **实验结果**

测试样例：

输入格式：

5

7

3 8

8 1 0

2 7 4 4

4 5 2 6 5

要求输出最大和

