**北京科技大学实验报告**

学院： 计通学院 专业：计算机科学与技术 班级： 计1503

姓名： 唐誉源 学号：41503302 实验日期： 2017 年 11 月 16 日

#### **实验名称：**

算法设计与分析实验4走迷宫问题

#### **实验目的：**

1. 利用动态规划思想应用于走迷宫问题
2. 求解出最大可获得的奖励
3. 设计合适的数据结构存储每一步的中间结果
4. 注意设置合理的边界条件保证不超出迷宫

#### **实验设备（环境）及要求：**

操作系统: macOS High Sierra 10.12.1;

IDE: Xcode.

#### **实验原理：**

1. 为了保证全局最优解，将一个小人从(1, 1)点走到(N, M)点，再从(N, M)点走回(1, 1)点的过程，近似成两个小人同时从(1, 1)点走到(N, M)点的过程；
2. 由于在一个小人走的情况下，第一次走遍的格子在第二次走的时候没有奖励值，将这种情况等效在两个小人的情况下为，当两个小人走到同一个格子时，只奖励一次；
3. 由于每次从(1, 1)点走到(N, M)点只准向下和向右，所以总体的步数是一定的，为N+M-2步，所以只需要知道两个小人分别坐标的一个值，就可以计算出另一个值；
4. 利用一个三维数组r[n+m-1][n-1][m-1]记录两个小人还剩余的步数和两个小人分别向下走的步数；
5. 在每个位置状态下，两个小人的下一步可以有四种情况，分别是两个都向右走，一个向右走一个向下走，一个向下走一个向右走，两个都向下走；
6. 根据动态规划，在每一步的每个位置状态下，取两个小人到达这两点所获的的最大奖励，记录下来，即当走下一步做四种状态的遍历时，都分别跟到达位置之前所记录的最大奖励值做比较，如果大于之前的值，则进行更新，简要表示为:

* r[i][j][k] = max(r[i][j][k], r[i+1][j][k]+a[j][n+m-2-i-j]+(j==k?0:a[k][n+m-2-i-k]))  
  r[i][j][k+1] = max(r[i][j][k+1], r[i+1][j][k]+a[j][n+m-2-i-j]+(j==k+1?0:a[k+1][n+m-3-i-k]))  
  r[i][j+1][k] = max(r[i][j+1][k], r[i+1][j][k]+a[j+1][n+m-3-i-j]+(j+1==k?0:a[k][n+m-2-i-k]))  
  r[i][j+1][k+1] = max(r[i][j+1][k+1], r[i+1][j][k]+a[j+1][n+m-3-i-j]+(j+1==k+1?0:a[k+1][n+m-3-i-k]))

1. 最后记录奖励的矩阵的最后一个值，即r[0][n-1][m-1]表示的两个小人都站在(N, M)所获的奖励，即为一个小人走回原点所获的的最大的奖励；
2. 增加一个新的三维数组t[n+m-1][n-1][m-1]，记录了下更新时的动作，即若某一步进行了更新，就知道这一步是由上一步的哪个位置走过来的；
3. 记录动作的矩阵可以知道两个小人的每种位置最优的情况下是从上一步哪个位置走过来的，于是进行倒推便知道总体的路径。

#### **实验内容与步骤：**

##### *（1）实验内容*

有一个的迷宫，马小跳在第一行第一列的起点上，迷宫的终点在(N, M)。在这个迷宫中每个格子都有一个奖励值，当马小跳走到这个格子上便可以获得格子上的奖励值。但走过一遍的格子边没有了奖励值。所以马小跳为了获得最大的奖励值，便不希望走到同一个格子上。首先，马小跳会从起点走到终点，此时马小跳只能向下或向右运动到相邻的格子上。到了终点之后，马小跳又从终点开始，向上或者向左运动到相邻的格子，一直到起点。不能走到迷宫的外边，现在马小跳想知道，自己寻回一遍迷宫之后，能获得的最大的奖励值有多少？

输入：N和M的值，以及的数值矩阵；

输出：最大的奖励值，以及从起点到终点在返回起点的移动路径。

##### *（2）主要步骤*

将记录路径的过程加入到动态规划路径得到奖励的过程，扩充实验原理中利用动态规划的思想得到的公式编写得到路径和奖励的函数：

r[n+m-2][0][0] = a[0][0];  
**for** (i=n+m-3; i>=0; i--) {  
 **for** (j=(0>=n-2-i?0:n-2-i); j<=n+m-2-i&&j<n; j++) {  
 **for** (k=(0>=n-2-i?0:n-2-i); k<=n+m-2-i&&k<n; k++) {  
 *// both go right*  
 **if** (j>=n-1-i && k>=n-1-i &&  
 r[i][j][k] <= r[i+1][j][k]+a[j][n+m-2-i-j]+(j==k?0:a[k][n+m-2-i-k])) {  
 r[i][j][k] = r[i+1][j][k]+a[j][n+m-2-i-j]+(j==k?0:a[k][n+m-2-i-k]);  
 t[i][j][k] = 1;*// record 'both go right' as 1*  
 }  
 *// one go right, one go down*  
 **if** (j>=n-1-i && k+1<n &&  
 r[i][j][k+1] <= r[i+1][j][k]+a[j][n+m-2-i-j]+(j==k+1?0:a[k+1][n+m-3-i-k])) {  
 r[i][j][k+1] = r[i+1][j][k]+a[j][n+m-2-i-j]+(j==k+1?0:a[k+1][n+m-3-i-k]);  
 t[i][j][k+1] = 2;*// record 'one go right, one go down' as 2*  
 }  
 *// one go down, one go right*  
 **if** (k>=n-1-i && j+1<n &&  
 r[i][j+1][k] <= r[i+1][j][k]+a[j+1][n+m-3-i-j]+(j+1==k?0:a[k][n+m-2-i-k])) {  
 r[i][j+1][k] = r[i+1][j][k]+a[j+1][n+m-3-i-j]+(j+1==k?0:a[k][n+m-2-i-k]);  
 t[i][j+1][k] = 3;*// record 'one go down, one go right' as 3*  
 }  
 *// both go down*  
 **if** (j+1<=n && k+1<n &&  
 r[i][j+1][k+1] <= r[i+1][j][k]+a[j+1][n+m-3-i-j]+(j+1==k+1?0:a[k+1][n+m-3-i-k])) {  
 r[i][j+1][k+1] = r[i+1][j][k]+a[j+1][n+m-3-i-j]+(j+1==k+1?0:a[k+1][n+m-3-i-k]);  
 t[i][j+1][k+1] = 4;*// record 'both go down' as 4*  
 }  
 }  
 }  
}

倒推路径并将路径记录在r1[n+m-1]和 r2[n+m-1]两个数组中：

*// backward route*  
int r1[n+m-1], r2[n+m-1];  
j = n-1;  
k = n-1;  
**for** (i = 0; i < n+m-2; i++) {  
 r1[i] = j;  
 r2[i] = k;  
 **switch** (t[i][j][k]) {  
 **case** 1:  
 **break**;  
 **case** 2: {  
 k--;  
 **break**;  
 }  
 **case** 3:  
 j--;  
 **break**;  
 **case** 4: {  
 j--;  
 k--;  
 **break**;  
 }  
 **default**: {  
 printf("wrong.");  
 **break**;  
 }  
 }  
}

最后得到的最大奖励是r[0][n-1][m-1];

在设计main函数时，也要注意输入数据的范围，规定矩阵的大小小于：

int main() {  
 int n, m;  
 int a[100][100]={0};  
 printf("please input testing data(0<n<100, 0<m<100):\n");  
 scanf("%d %d", &n, &m);  
 **if** (n>=100||m>=100||n<=0||m<=0) {  
 printf("Wrong number. Please retry.\n");  
 **return** 1;  
 }  
 **for** (int i=0; i<n; i++) {  
 **for** (int j=0; j<m; j++) {  
 scanf("%d", &a[i][j]);  
 }  
 }  
 int reward = route(n, m, a);  
 printf("whole reward is: %d\n", reward);  
 **return** 0;  
}

#### **实验数据：**

样例输入：

3 3  
1 3 1  
1 1 1  
1 2 1

样例输出：

(0, 0)->(1, 0)->(2, 0)->(2, 1)->(2, 2)->(1, 2)->(1, 1)->(0, 1)->(0, 0).  
whole reward is: 11

样例输入：

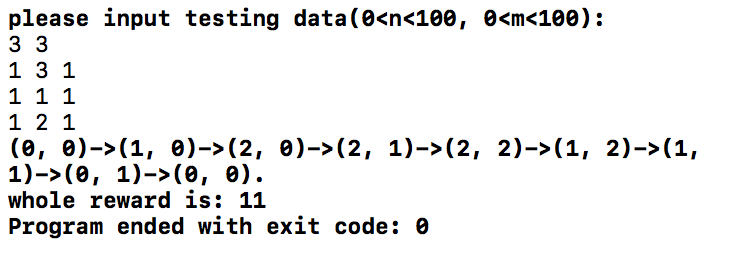
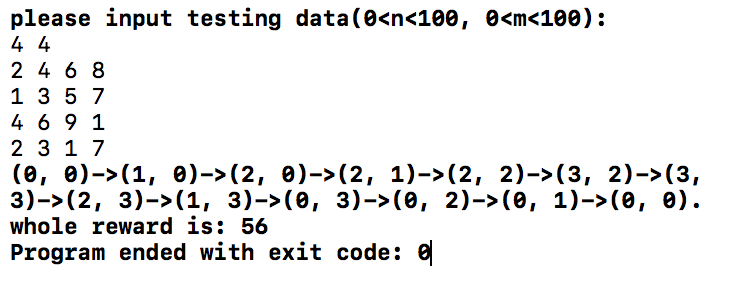
4 4  
2 4 6 8  
1 3 5 7  
4 6 9 1  
2 3 1 7

样例输出：

(0, 0)->(1, 0)->(2, 0)->(2, 1)->(2, 2)->(3, 2)->(3, 3)->(2, 3)->(1, 3)->(0, 3)->(0, 2)->(0, 1)->(0, 0).  
whole reward is: 56

#### **实验结果与分析：**

运行结果截图展示：

可以看出运行结果完整地呈现了马小跳走完一周的路径以及最后得到的奖励值，且由于程序动态规划的设计，得出了全局最优解。

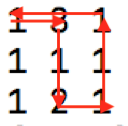
#### **实验结论：**

##### *（1）实验结论*

利用动态规划思想应用于走迷宫问题，设计了合适的数据结构进行存储，得出了全局最优的路径和最大的奖励，更加深刻理解了动态规划的设计思想和其所用的数据结构的应用，对于矩阵循环边界条件的设置也有了更加深刻的理解。

##### *（2）讨论*

1. 在最开始设计程序的时候，先设计了一个方法：将从起点到终点和从终点到起点的过程分开，进行两次动态规划（**程序也已附上**）。这种分步的动态规划，可能会得到两次（从起点到终点和从终点到起点）的局部最优解，但不一定能得到全局最优解，例如在本实验的第一个例子中，由这种方法所得到的结果为

* 
* 这样的到的最大奖励值为10，而不是本实验中得出的11；

2. 在本实验中的边界条件设置极为繁琐，由于两个小人每一步的位置都在一条斜角线上，需要分别控制每个小人的横纵坐标四个变量不超过矩阵的范围。相信本次实验也会给后续将来与矩阵斜角线相关的变量范围控制带来一定启发；

3. 在程序中只对输入的矩阵大小做了约束，而对矩阵中的奖励值没有设置范围检查，如果完善的话可以讲变量范围设置详细，以增强健壮性；