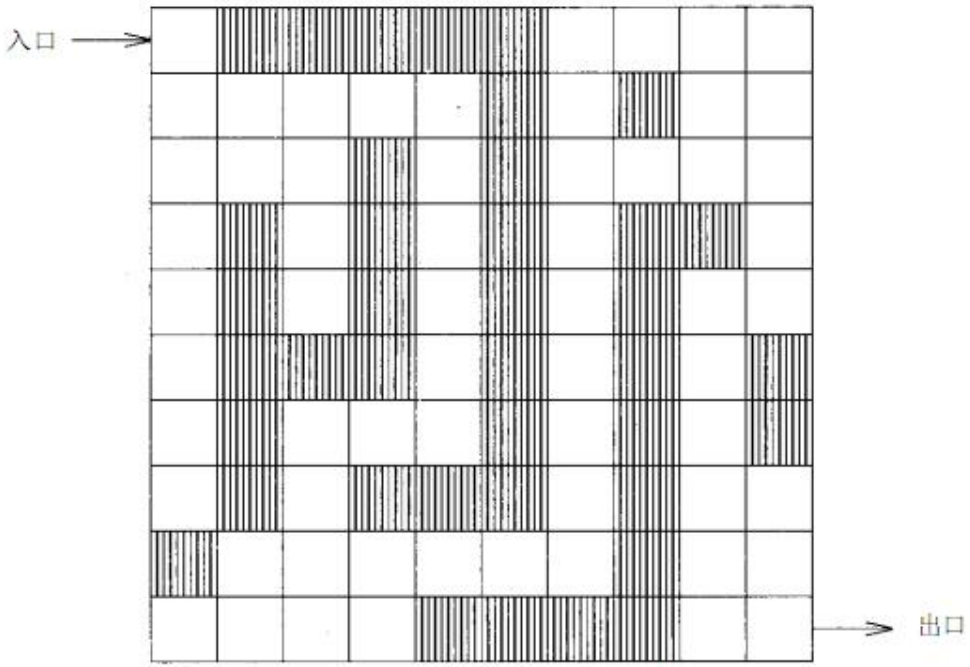
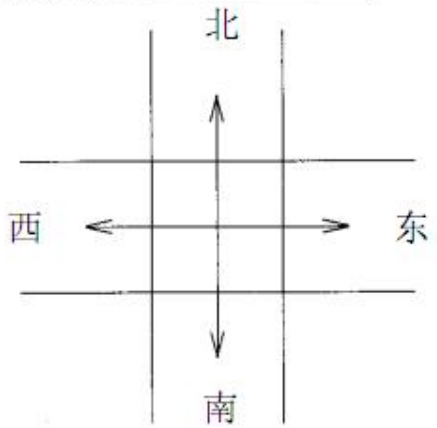
迷宫问题



0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0 0 0
0 1 0 1 0 1 0 1 1 0
0 1 0 1 0 1 0 1 0 0
0 1 1 1 0 1 0 1 0 1
0 1 0 0 0 1 0 1 0 1
0 1 0 1 1 1 0 1 0 0
1 0 0 0 0 0 0 1 0 0
0 0 0 0 1 1 1 1 0 0



- 寻找一条从入口到出口的路径。
- 【路径】是由一组位置构成的，每个位置上都没有障碍，且每个位置（第一个除外）都是前一个位置的东、南、西或北的邻居。

求解思想-穷举求解

计算机解迷宫时，通常用的是“穷举求解”的方法：

- 从入口出发，顺某一方向向前探索，若能走通，则继续往前走；
- 否则沿原路退回，换一个方向再继续探索，直至所有可能的通路都探索到为止；
- 如果所有可能的通路都试探过，还是不能走到终点，那就说明该迷宫不存在从起点到终点的通道。

求解迷宫路径算法的基本思想

(1) 若当前位置“可通”，则纳入路径，继续前进；

(2) 若当前位置“不可通”，则后退，换向探索；

(3) 若四周均“不可通”，则从路径中删除。

当前位置始终在栈顶。

FindPath的初始版

```
bool FindPath( )
{
        在迷宫中寻找一条通往出口的路径;
        if（找到一条路径）
                return true;
        else
                return false;
}
```

```
bool FindPath()
{// 寻找从位置(1,1)到出口(m,m)的路径
 增加一圈障碍物;

// 对跟踪当前位置的变量进行初始化
Position here;
here.row = 1;
here.col = 1;

maze [1] [1] = 1; // 阻止返回入口

//寻找通往出口的路径
while (不是出口) do {
 选择一个相邻位置;
 if (存在这样一个相邻位置) {
   把当前位置 here 放入堆栈 path;
   // 移动到相邻位置，并在当前位置放上障碍物
   here = neighbor;
   maze[here.row] [here.col] = 1;}
 else {
   // 不能继续移动，需回溯
   if (堆栈path为空) return false;
   回溯到path栈顶中的位置 here; }
}
return true;
```

FindPath的细化版

```
设定当前位置的初值为入口位置;
    do{
       若当前位置可通,
       则{
          将当前位置插入栈顶;                    // 纳入路径
          若该位置是出口位置, 则算法结束;
            // 此时栈中存放的是一条从入口位置到出口位置的路径
          否则切换当前位置的东邻方块为新的当前位置;
        }
       否则
       {
        若栈不空且栈顶位置尚有其他方向未被探索,
        则设定新的当前位置为: 沿顺时针方向旋转找到的栈顶位置的下一相邻
块;
        若栈不空但栈顶位置的四周均不可通,
        则{ 删去栈顶位置;                       // 从路径中删去该通道块
            若栈不空, 则重新测试新的栈顶位置,
            直至找到一个可通的相邻块或出栈至栈空;
          }
      }
    } while (栈不空);
```

偏移量（offset）表

移动	方向	offset[move].row	offset[move].col
0	向右	0	1
1	向下	1	0
2	向左	0	-1
3	向上	-1	0

```
int **maze,m;

Stack<Position> *Path;
```