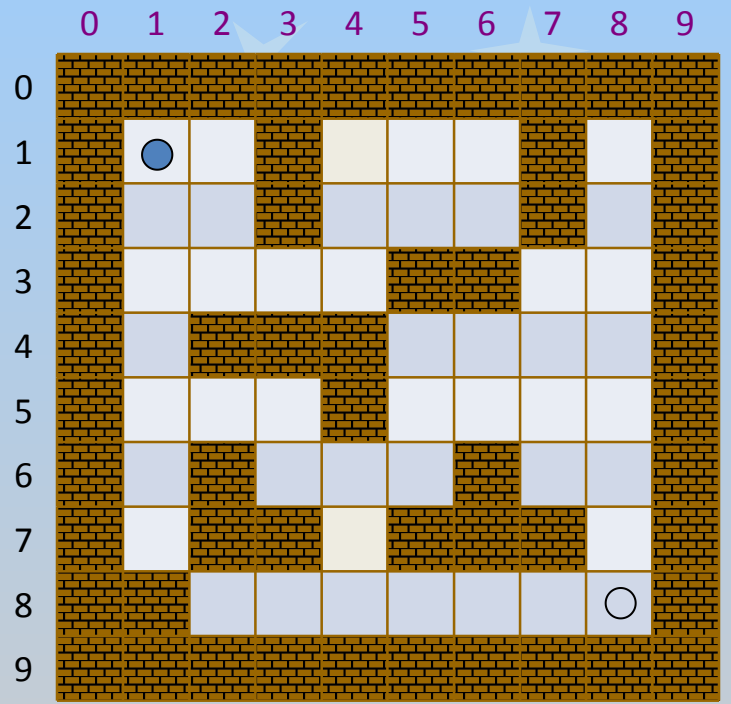
本节主题：

栈的应用2 - 迷宫问题

问题

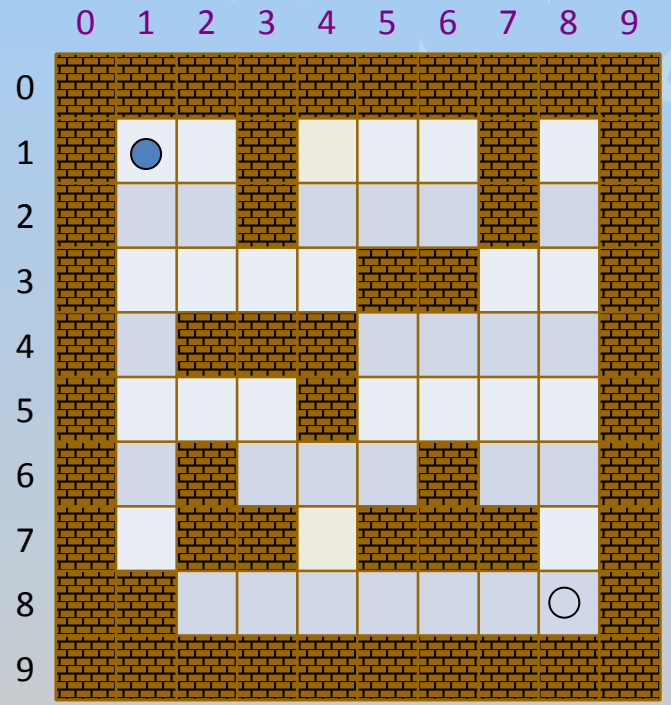
- 迷宫问题：求出从入口到出口的路径。
- 求解策略
  - 从入口出发，顺某一方向向前试探
  - 若能走通，则继续往前走；
  - 否则沿原路退回，换一个方向再继续试探，直至所有可能的通路都试探完为止。
- 关键
  - 为了保证在任何位置上都能沿原路退回（称为回溯）
  - 需要用一个后进先出的栈来保存从入口到当前位置的路径。



数据表示

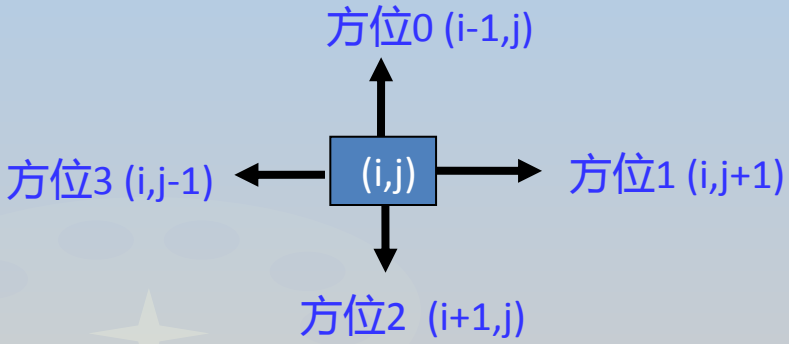
- 设置一个数组mg表示迷宫，方块为0表示对应方块是通道，为1时表示对应方块为墙：

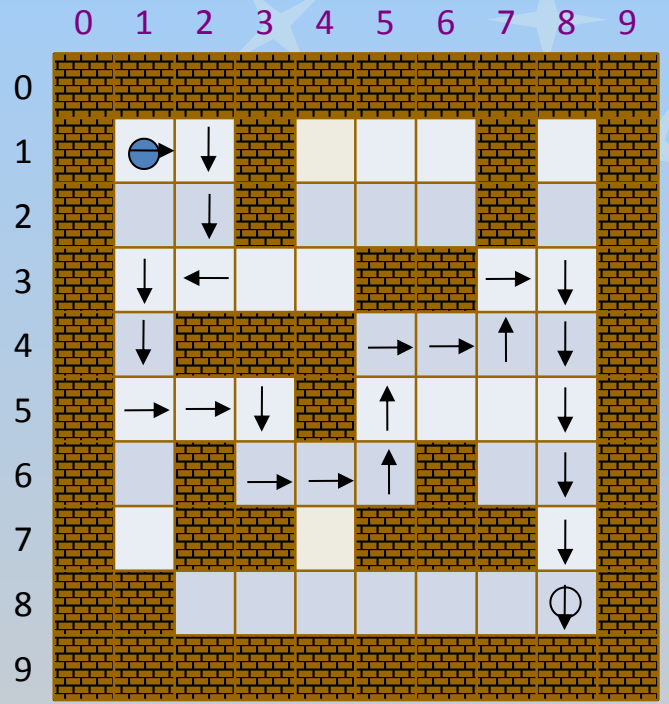
```
int mg[M+2][N+2]={ //M=8,N=8
{1,1,1,1,1,1,1,1,1,1},
{1,0,0,1,0,0,0,1,0,1},
{1,0,0,1,0,0,0,1,0,1},
{1,0,0,0,0,1,1,0,0,1},
{1,0,1,1,1,0,0,0,0,1},
{1,0,0,0,1,0,0,0,0,1},
{1,0,1,0,0,0,1,0,0,1},
{1,0,1,1,1,0,1,1,0,1},
{1,1,0,0,0,0,0,0,0,1},
{1,1,1,1,1,1,1,1,1,1}};
```



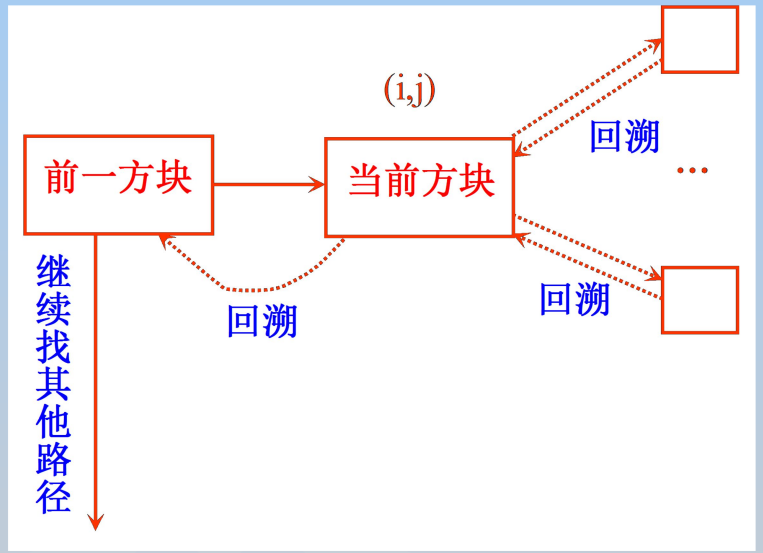
算法设计

- 对于迷宫中的每个方块，有上下左右四个方块相邻，如下图所示，第i行第j列的当前方块的位置为(i,j)，规定上方方块为方位0，顺时针方向递增编号。
- 在试探过程中，假设从方位0到方位3的方向查找下一个可走的方块。



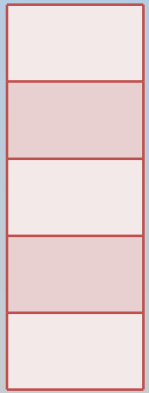


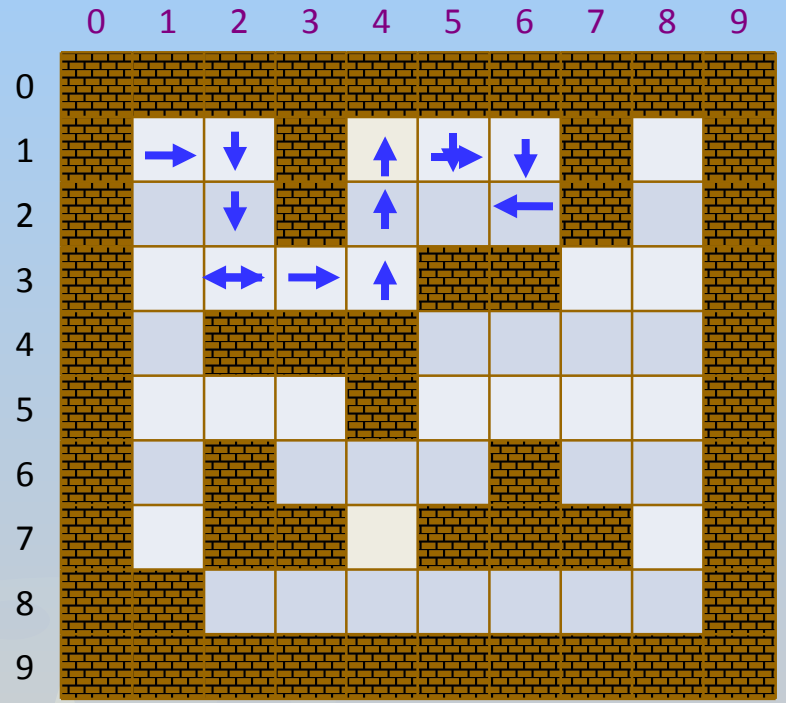
回溯策略及需要的数据结构




```
typedef struct
{
    int i;  //当前方块的行号
    int j;  //当前方块的列号
    int di; //di是下一可走相邻方位的方位号
} Box; //定义方块类型
typedef struct
{
    Box data[MaxSize];
    int top; //栈顶指针
} StType;    //顺序栈类型
```

试探轨迹栈





2	5	-1
2	6	3
2	6	-2
1	5	2
1	4	1
2	4	0
3	4	0
3	3	1
3	2	3
2	2	2
1	2	2
1	1	1

算法实现

```
bool mgpath(int xi,int yi,int xe,int ye)
{
   int i,j,k,di,find;
   StType st;
   st.top=-1; //初始化栈顶指针
   st.top++;  //初始方块进栈
   st.data[st.top].i=xi;
   st.data[st.top].j=yi;
   st.data[st.top].di=-1;
   mg[xi][yi]=-1;
   //开始走，栈不空时循环
   return false;
}
```

```
while (st.top>-1)
{
   i=st.data[st.top].i;
   j=st.data[st.top].j;
   di=st.data[st.top].di;

   //找到了出口,输出路径

   //找下一个可走方块
```

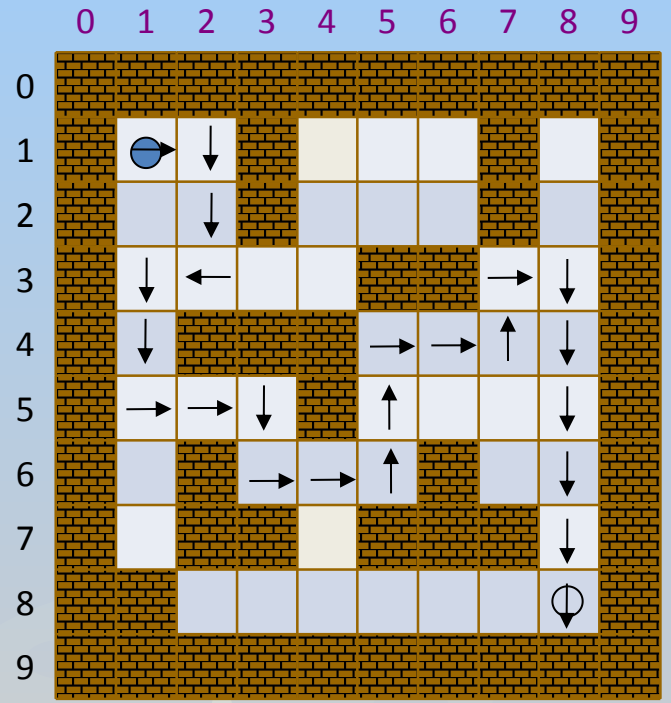
```
   //找到了下一个可走方块
}
```

```
if (find==1)
{
   st.data[st.top].di=di;
   st.top++;
   st.data[st.top].i=i;
   st.data[st.top].j=j;
   st.data[st.top].di=-1;
   mg[i][j]=-1;
}
else
{
   mg[st.data[st.top].i][st.data[st.top].j]=0;
   st.top--;
}
```

```
if (i==xe && j==ye)
{
   printf("迷宫路径如下:\n");
   for (k=0; k<=st.top; k++)
   {
      printf("\t(%d,%d)",st.data[k].i,st.data[k].j);
      if ((k+1)%5==0)
         printf("\n");
   }
   printf("\n");
   return true;
}
```

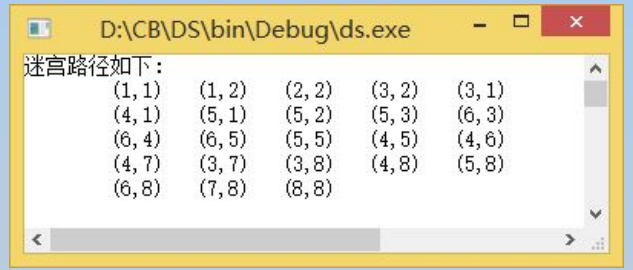
```
find=0;
while (di<4 && find==0)
{
   di++;
   switch(di)
   {
   case 0:
      i=st.data[st.top].i-1;
      j=st.data[st.top].j;
      break;
   case 1:
      i=st.data[st.top].i;
      j=st.data[st.top].j+1;
      break;
   case 2:
      i=st.data[st.top].i+1;
      j=st.data[st.top].j;
      break;
   case 3:
      i=st.data[s.top].i,j=st.data[st.top].j-1;
      break;
   }
   if (mg[i][j]==0)
      find=1; //找到下一个可走相邻方块
}
```

最后结果




```
int mg[M+2][N+2]={ //M=8,N=8
{1,1,1,1,1,1,1,1,1,1},
{1,0,0,1,0,0,0,1,0,1},
{1,0,0,1,0,0,0,1,0,1},
{1,0,0,0,0,1,1,0,0,1},
{1,0,1,1,1,0,0,0,0,1},
{1,0,0,0,1,0,0,0,0,1},
{1,0,1,0,0,0,1,0,0,1},
{1,0,1,1,1,0,1,1,0,1},
{1,1,0,0,0,0,0,0,0,1},
{1,1,1,1,1,1,1,1,1,1}};

int main()
{
   if (!mgpath(1,1,M,N))
      printf("该迷宫问题没有解!");
   return 0;
}
```



思考题

- 用栈求解迷宫问题有什么特点？