第 3 章　用栈实现迷宫探路

3.1　项 目 概 述

栈是计算机术语中比较重要的概念，在计算机中有广泛的运用。程序员无时无刻不在应用栈，函数的调用是间接使用栈的最好例子，可以说栈的一个最重要的应用就是函数的调用。栈典型的应用还有判断平衡符号、实现表达式的求值（中缀表达式转后缀表达式的问题以及后缀表达式求值问题）、在路径探索中实现路径的保存。本章中的迷宫探路问题就是用栈保存搜索的路径的实例。

本章将重点介绍栈的顺序存储结构和链式存储结构，以及栈在不同存储结构中基本操作的实现，并应用本章实现的栈及 Java 中栈的实现类 Stack 类和 LinkedList 类来解决迷宫探路问题。

3.2　项 目 目 标

本章项目学习目标如表 3-1 所示。

表 3-1　项目学习目标

序　　号	学 习 目 标	知 识 要 点
1	理解栈的逻辑结构	栈的基本概念、基本操作、抽象数据类型
2	理解栈的顺序与链式存储结构及其算法实现	栈的顺序存储：顺序栈 栈的链式存储：链式栈
3	熟悉 Java 中栈的实现类	栈的顺序存储实现类：Stack＜E＞ 栈的链式存储实现类：LinkedList＜E＞
4	能在实际问题中找出栈并用栈解决实际问题	使用栈解决迷宫探路问题

3.3　项 目 情 境

编程实现迷宫探路


视频讲解

1. 情境描述

迷宫（maze）是一个矩形区域，它有一个入口和一个出口，入口位于迷宫的左上角，出口位于迷宫的右下角，在迷宫的内部包含不能穿越的墙或障碍物，如图 3-1 所示。迷宫探路问

题是寻找一条从入口到出口的路径,该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,如图 3-2 所示。

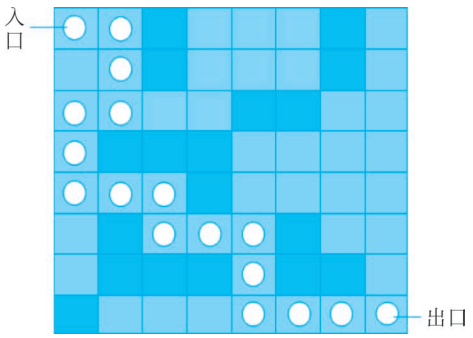

图 3-1　迷宫图

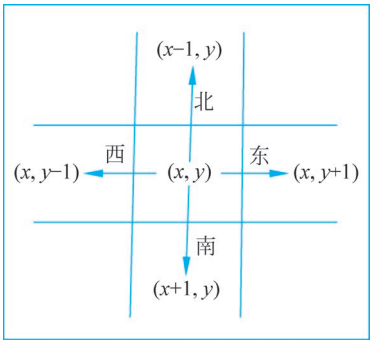

图 3-2　迷宫任意位置的 4 种移动方向

迷宫探路的基本思路是从迷宫的入口出发,沿正东方向顺时针对当前位置相邻的东、南、西、北 4 个位置依次进行判断,搜索可通行的位置。如果有,移动到这个新的相邻位置上,如果新位置是迷宫出口,那么已经找到了一条路径,搜索工作结束;否则从这个新位置开始继续搜索通往出口的路径。若当前位置四周均无通路,则将当前位置从路径中删除,顺着当前位置的上一个位置的下一方向继续走,直到到达出口(找到一条通路)或退回到入口(迷宫没有出路)时结束。

2. 基本要求

编写程序搜索迷宫通路,如果找到通路,显示“找到可到达路径”,并显示路径的位置信息;如路径没有找到,提示“没有可到达路径”。

3.4　项 目 实 施

3.4.1　分析栈的逻辑结构


视频讲解

【学习目标】

(1) 熟悉栈的逻辑结构。

(2) 熟悉栈的基本操作。

(3) 熟悉栈的抽象数据类型。

【任务描述】

为完成迷宫探路编程任务,首先对问题抽象,建立问题的抽象数据类型。一是确定数据对象的逻辑结构,找出构成数据对象的数据元素之间的关系。二是确定为求解问题需要对数据对象进行的操作或运算。最后将数据的逻辑结构及其在该结构上的运算进行封装得到抽象数据类型。

【任务实施】

步骤一:分析栈的逻辑结构

在迷宫探路问题中有两个数据对象,一个是迷宫,另一个是通路。

迷宫是一个矩形区域,用二维数组 maze[m][n]表示,maze[i][j]=0 或 1,其中,0 表

示不通，1 表示可通。当从某点向下试探时，中间点有 4 个方向（东、南、西、北）可以试探，而 4 个角点有 2 个方向，其他边缘点有 3 个方向。为使问题简单化，用 maze[$m+2$][$n+2$]来表示迷宫，设迷宫的四周的值全部为 0。这样可以使每个点的试探方向全部为 4，不用再判断当前点的试探方向有几个，这与迷宫周围是墙壁这一实际问题一致，如图 3-3 所示。图 3-4 给出了图 3-3 对应的迷宫矩阵。

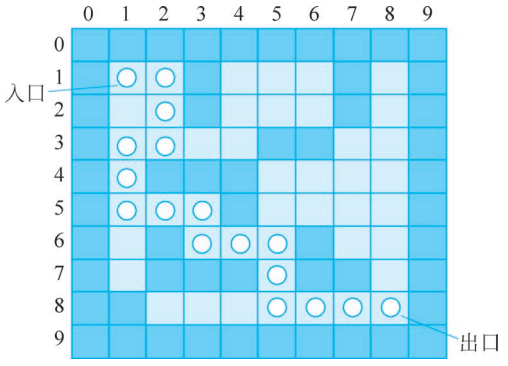

图 3-3　四周为墙壁的迷宫

$$\begin{bmatrix}
0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&1&1&1&0&1&0\\
0&1&1&0&1&1&1&0&1&0\\
0&1&1&1&1&0&0&1&1&0\\
0&1&0&0&0&1&1&1&1&0\\
0&1&1&1&0&1&1&1&1&0\\
0&1&0&1&1&1&0&1&1&0\\
0&1&0&0&0&1&0&0&1&0\\
0&0&1&1&1&1&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

图 3-4　迷宫的矩阵描述

通路是探路的过程中形成的由若干可通位置组成的路径，该位置用坐标(x，y，d)表示，其中，x 表示迷宫矩阵 maze 上的行，y 表示迷宫矩阵 maze 上的列，d 表示前进的方向，东、南、西、北用 0、1、2、3 表示。

迷宫探路是一个动态的过程，在探索的过程中，每个点有 4 个方向去试探。假设从当前位置向前试探的方向为从正东沿顺时针方向进行，当最新位置的四周均无通路时就需要删除最新加入的位置。在图 3-3 中，最初的探索路径为(1,1,0)→(1,2,1)→(2,2,1)→(3,2,0)→(3,3,0)→(3,4,3)→(2,4,0)→(2,5,0)→(2,6,3)→(1,6,2)→(1,5,2)，但走到(1,4)时，无法再探索下去，就需要探索该路径末端位置的其他方向。如果有一个方向通，修改该位置的方向；如果没有通的方向，就从路径中删除该位置，这样依次删除 (1,5,2)、(1,6,2)、(2,6,3)、(2,5,0)、(2,4,0)、(3,4,3)、(3,3,0)，直到 (3,2)点，才可以找到一个往西通行的点(3,1)，修改 (3,2,0)为(3,2,2)。按照这样的思路，找到了一条可通行的路：(1,1,0)→(1,2,1)→(2,2,1)→(3,2,2)→(3,1,1)→(4,1,1)→(5,1,0)→(5,2,0)→(5,3,1)→(6,3,0)→(6,4,0)→(6,5,1)→(7,5,1)→(8,5,0)→(8,6,0)→(8,7,0)→(8,8,−1)。到达终点时，前进的方向值为−1。

可以看到，可通的路径中除第一个加入的位置和最后一个加入的位置外，其余位置都只有一个前驱和一个后继，因此该路径是一个线性表，每个位置都是线性表中的一个结点。在没有找到出口前，探索通路末端结点的相邻位置，可通则加入通路变成新的末端结点，若四周均无通路，则删除该结点。因此通路所形成的线性表只在一端进行插入和删除，具有这种特点的线性表称为栈。

1. 栈的定义

栈(stack)是一种特殊的线性表，是一种只允许在表的一端进行插入或删除操作的线性表。表中允许进行插入和删除操作的一端称为栈顶，最下面的那一端称为栈底。栈顶是动态的，它由一个称为栈顶指针的位置指示器指示。当栈中没有数据元素时，称为空栈。栈的

插入操作称为进栈或入栈,栈的删除操作称为出栈或退栈。

若给定一个栈 $s=(a_0,a_1,a_2,\cdots,a_{n-1})$,如图 3-5 所示。在图中,$a_0$ 为栈底元素,a_{n-1} 为栈顶元素,元素 a_i 位于元素 a_{i-1} 之上。栈中元素按 $a_0,a_1,a_2,\cdots,a_{n-1}$ 的次序进栈,如果从这个栈中取出所有的元素,则出栈次序为 $a_{n-1},a_{n-2},\cdots,a_0$。

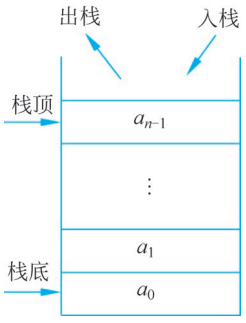

图 3-5　栈结构示意图

例如,一个数列(23,45,3,7,3,945),先对其进行入栈操作,则入栈顺序为(23,45,3,7,3,945);再对其进行出栈操作,则出栈顺序为(945,3,7,3,45,23)。入栈出栈就像只有一个口的长筒,先把数据一个个放入筒内,而拿出的时候只有先拿走上边的,才能拿走下边的。

2. 栈的特征

栈的主要特点是“后进先出”,即后进栈的元素先处理。因此栈又称为后进先出(Last In First Out, LIFO)表。

在图 3-5 中,栈中元素按 $a_0,a_1,a_2,\cdots,a_{n-1}$ 的次序进栈,而出栈次序为 $a_{n-1},a_{n-2},\cdots,a_0$。平常生活中洗碗也是一个“后进先出”的栈的例子,可以把洗净的一摞碗看作一个栈。在通常情况下,最先洗净的碗总是放在最底下,后洗净的碗总是摞在最顶上。在使用时,却是从顶上拿取,也就是说,后洗的先取用(后摞上的先取用)。如果把洗净的碗“摞上”称为进栈,把“取碗”称为出栈,那么上例的特点是后进栈的先出栈。然而,摞起来的碗实际上是一个线性表,只不过“进栈”和“出栈”,或者说,元素的插入和删除是在线性表的一端进行而已。

步骤二:分析栈的基本运算

迷宫探路问题中的若干可通位置形成的通路构成了栈,对栈的运算主要有以下几种。

(1) 初始化栈:也就是产生一个新的空栈。

(2) 入栈:在栈顶添加一个数据元素。入栈示意图如图 3-6 所示。

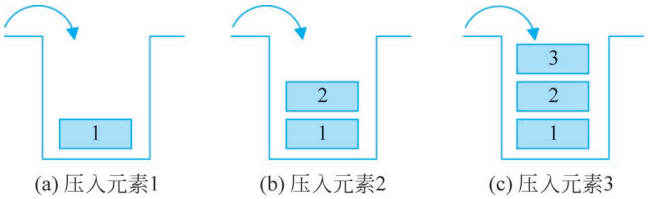

图 3-6　入栈示意图

(3) 出栈:删除栈顶数据元素。出栈示意图如图 3-7 所示。

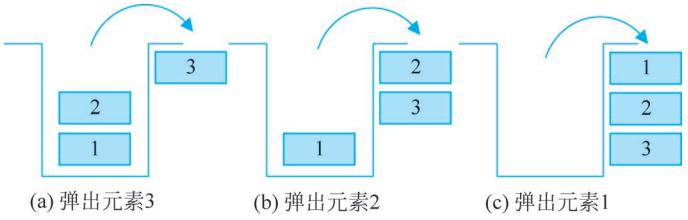

图 3-7　出栈示意图

(4) 读栈顶元素:获取栈中当前栈顶的数据元素,栈中数据元素不变。

(5) 求栈长度:获取栈中的数据元素个数。

(6) 判断栈空：判断栈中是否有数据元素。

步骤三：定义栈的抽象数据类型

根据对栈的逻辑结构及基本运算的认识，得到栈的抽象数据类型。

ADT 栈(stack)

数据对象：

$D=\{a_i \mid 0\leqslant i\leqslant n-1, n\geqslant 0, a_i \text{ 为 } E \text{ 类型}\}$

数据关系：

$R=\{<a_i, a_{i+1}> \mid a_i, a_{i+1}\in D, i=0,\cdots,n-2\}$

数据运算：

将对栈的基本操作定义在接口 IStack 中，当存储结构确定后通过实现接口来实现这些基本操作，来确保算法定义和实现的分离。

```
public interface IStack<E> {
void init();                    //创建一个空的栈
E push(E a);                    //入栈
E pop();                        //出栈
E peek();                       //取栈顶元素
int size();                     //返回栈中元素的个数
boolean empty();                //判断栈是否为空
}
```

除初始化栈运算在实现栈的类的构造函数中实现外，栈的其他运算定义在接口 IStack 中，当存储结构确定后通过实现接口来完成这些基本运算的具体实现，确保运算的定义和实现的分离。

小贴示

栈是计算机术语中比较重要的概念，在计算机中有广泛的运用。程序员无时无刻不在应用栈，函数的调用、浏览器的“前进”和“后退”、表达式的求值都是栈的重要应用。

本项目中，用栈保存迷宫路径搜索过程中产生的路径，迷宫探路告诉我们前进的路有多条，一条不通时，不要气馁，暂且后退继续尝试下一条，只要坚持，一定会找到成功的通路。

【任务评价】

请按表 3-2 查看是否掌握了本任务所学的内容。

表 3-2 “分析栈的逻辑结构”完成情况评价表

序　号	鉴定评分点	分　值	评　分
1	能理解栈的定义和特点	25	
2	能理解栈的基本操作	25	
3	能理解栈的抽象数据类型	25	
4	能从迷宫探路问题中识别出栈	25	

3.4.2 用顺序栈实现迷宫探路

【学习目标】

(1) 掌握栈的顺序存储结构。

(2) 掌握顺序栈的实现方法。

(3) 能用顺序栈实现迷宫探路。

【任务描述】

在 3.1 节的任务中,已分析出迷宫探路问题中通路的逻辑结构为栈,并定义了栈的抽象数据类型。接下来考虑将逻辑结构为栈的通路存储到计算机中去,进行存储结构的设计。存储结构有顺序存储结构和链式存储结构两种,本任务将栈的逻辑结构映射成顺序存储结构,并基于顺序存储结构实现栈的基本运算,最后,将实现的顺序栈应用在迷宫探路问题中,编写程序实现迷宫探路。

【任务实施】

步骤一:将栈的逻辑结构映射成顺序存储结构

1. 顺序栈的定义及存储结构

用一片连续的存储空间来存储栈中的数据元素,这样的栈称为顺序栈(sequence stack)。

类似于顺序表,用一维数组来存放顺序栈中的数据元素。栈顶指示器 top 用来指示数组下标,top 随着插入和删除而变化,当栈为空时,top=−1。顺序栈的栈顶指示器 top 与栈中数据元素的关系如图 3-8 所示。

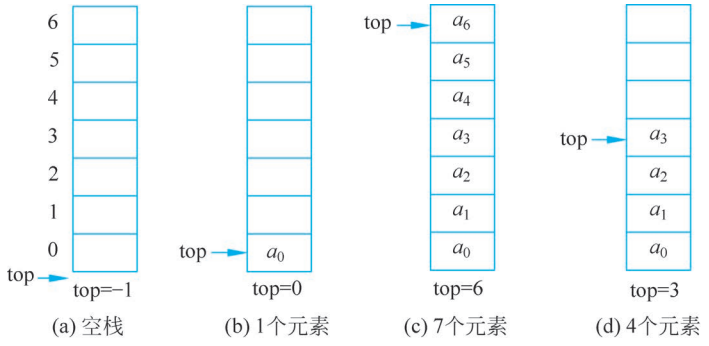

图 3-8 栈的动态示意图

当有数据元素入栈时,栈顶指示器 top 加 1;当有数据元素出栈时,栈顶指示器 top 减 1;当栈为空时,栈顶指示器 top 为−1。栈中元素的个数可由 top+1 求得。

2. 基于顺序存储结构创建顺序栈类

(1) 创建一个顺序栈类 SeqStack<E>,该类实现接口 IStack<E>,接口中定义的基本运算的算法实现在步骤二完成。

(2) 定义三个类变量 data、maxsize、top。其中,data 为一维数组,用来存储栈中的数据元素,元素类型为泛型,以实现不同数据类型的顺序栈间代码的重用。因为用数组存储顺序栈,需预先为顺序栈分配最大存储空间,maxsize 为表示顺序栈的最大容量。由于栈顶元素经常变动,设置一个变量 top 表示栈顶,top 的范围是 0～maxsize−1,如果顺序栈为空,top=−1。

```
public class SeqStack<E> implements IStack<E> {
    private E[] data;                    //数组用于存储顺序栈中的数据元素
    private int maxsize;                 //顺序栈的最大容量
```

```
    private int top;                    //顺序栈的栈顶指示器
    //初始化栈
      …
    //IStack 接口中定义的运算方法

}
```

步骤二：完成顺序栈的基本运算

1. 初始化顺序栈

初始化顺序栈就是创建一个空栈，即调用 SeqStack < E >的构造函数，申请顺序存储空间，具体执行下面的步骤。

（1）初始化 maxsize 为实际值。

（2）为数组申请可以存储 maxsize 个数据元素的存储空间，数据元素的类型由实际应用而定。

（3）初始化 top 的值为－1。

```
//初始化栈
public SeqStack(Class < E > type, int size) {
    data = (E[]) Array.newInstance(type, size);
    maxsize = size;
    top = -1;
}
```

2. 求栈的长度 size()

因栈顶指示器的值为栈中最后一个元素的下标，而下标是从 0 开始的，因此求栈长度可用 top＋1 计算得出。

```
//求栈的长度
    public int size() {
        return top + 1;
    }
```

此算法的时间复杂度为 $O(1)$。

3. 判断顺序栈是否为空 empty()

当栈顶指示器 top 的值为－1 时，顺序栈为空。

```
//判断栈是否为空
    public boolean empty() {
        return top == -1;
    }
```

此算法的时间复杂度为 $O(1)$。

4. 判断顺序栈是否为满 isFull()

maxsize 表示顺序栈的最大长度容量，top 的值为顺序栈顶的下标，当 top 的值为 maxsize－1 时，顺序栈中元素个数为 maxsize，达到了最大容量。

```
//判断栈是否为满
public boolean isFull() {
    return top == maxsize - 1;
}
```

此算法的时间复杂度为 $O(1)$。

5. 入栈操作 push(E a)

假设顺序栈顶端元素的索引保存在变量 top 中,栈中已有 top+1(0≤(top+1)≤maxsize−1)个数据元素,push 操作是将一个给定的数据元素保存在栈的最顶端,如图 3-9 所示为元素 a 即将入栈,需要执行以下步骤。


视频讲解

(1) 判断栈是否是满的,如果是,返回 false;否则执行下面的步骤。

(2) 设置 top 的值为 top+1,top 指向要入栈元素的位置,如图 3-10 所示。

(3) 设置 top 所指向的位置的值为入栈元素的值,如图 3-11 所示。

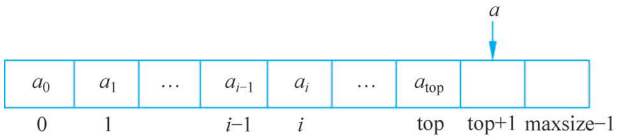

图 3-9 元素 a 即将入栈

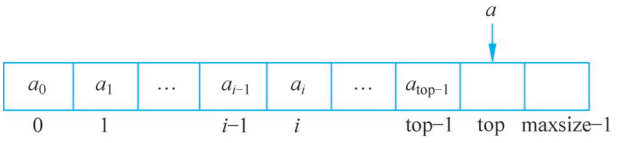

图 3-10 top 的值加 1

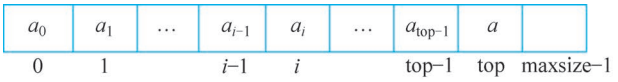

图 3-11 设置下标为 top 的数组元素的值为 a

```
//入栈操作
public E push(E a) {
    if (isFull())
        return null;
    data[++top] = a;
    return a;
}
```


视频讲解

此算法的时间复杂度为 $O(1)$。

6. 出栈操作 pop()

出栈操作是从栈的顶部取出数据,在图 3-12 中,取出 a_{top} 元素,并从栈中删除该元素,执行以下的步骤。

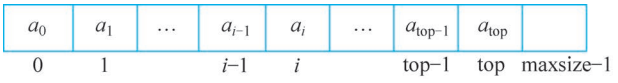

图 3-12 拟取出栈的顶部元素 a_{top}

(1) 检查栈中是否含有元素,如果无,返回 null;否则执行下面的步骤。

(2) 获取栈顶指针 top 指向的元素。

(3) 将栈顶指针 top 的值减 1,如图 3-13 所示,无法再通过栈顶指针访问出栈的元素。

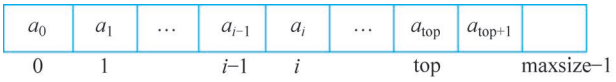

图 3-13　将栈顶指针 top 的值减 1

```
//出栈操作
    public E pop() {
        E a = null;
        if (!empty()) {
         a = data[top--];
        }
        return a;
    }
```

此算法的时间复杂度为 $O(1)$。

7. 取栈顶元素 peek()

取栈顶元素操作与出栈操作相似，只是取栈顶元素操作不改变原有栈，不删除取出的元素。

(1) 检查栈中是否含有元素，如果无，返回 null；否则执行下面的步骤。

(2) 获取栈顶指针 top 指向的元素。

```
//获取栈顶数据元素
    public E peek() {
        E a = null;
        if (!empty()) {
            a = data[top];
        }
        return a;
    }
```

此算法的时间复杂度为 $O(1)$。

步骤三：对顺序栈实现的基本运算算法进行测试

```
public class TestStack {
    public static void main(String[] args) {
        int[] data = {23,45,3,7,6,945};
    IStack<Integer> stack = new SeqStack<Integer>(Integer.class,data.length);
        //入栈操作
        System.out.println("******* 入栈操作 *******");
         for(int i = 0; i<data.length;i++){
              stack.push(data[i]);
           System.out.println(data[i]+" 入栈");
           }
        int size = stack.size();
       //出栈操作
        System.out.println("******* 出栈操作 *******");
        for(int i = 0; i<size;i++){
            System.out.println(stack.pop()+" 出栈 ");
        }
     }
}
```

步骤四：用顺序栈实现迷宫探路

1. 创建栈中数据元素类 Point

定义一个内部类 Point，用来表示栈中的数据元素位置点，有 x、y、d 三个成员变量，依次表示该点所在行坐标、列坐标及前进的方向。

```
public class Point{
    public int x,y,d;                    //行坐标、列坐标及前进的方向
    public Point(int x ,int y,int d){
        this.x = x;
        this.y = y;
        this.d = d;
    }
}
```

2. 创建迷宫类 Migong

(1) 创建一个迷宫类 Migong，定义了四个类变量 maze、row、col、sta。其中，maze 为二维数组，用来存放迷宫；row 表示迷宫矩阵的行数，col 表示迷宫矩阵的列数，sta 为存放路径的栈。

(2) 在 Migong 类的构造函数中对类变量进行初始化。根据构造函数传入二维数组，构造行数和列数加 2 的新的迷宫矩阵，并创建可容纳迷宫矩阵中数据元素的栈。

```
public class Migong {
    int[][] maze;                    //迷宫矩阵
    int row,col;                     //迷宫矩阵的行和列
    IStack<Point> sta;               //存放路径的栈
    public Migong(int[][] map) {
        row = map.length + 2;
        col = map[0].length + 2;
        maze = new int[row][col];
        for (int i = 1; i < row - 1; i++)
            for (int j = 1; j < col - 1; j++)
                maze[i][j] = map[i - 1][j - 1];
        sta = new SeqStack<Point>(Point.class,row * col);
    }
}
```

知识点回顾

在迷宫探路问题中，使用了二维数组存放迷宫。

1) 二维数组的概念

二维数组其实也是个一维数组，只不过该一维数组的每个元素又都是一个一维数组。

2) 二维数组的声明和初始化

(1) 定义二维数组。

数据类型[][] 数组名=new 数据类型[二维数组的长度][一维数组的长度];

例如：int[][] array=new int[1][2]; //数组有默认值，int 类型默认值为 0

(2) 定义并初始化二维数组。

元素的数据类型[][] 二维数组名=new 元素的数据类型[][]{

```
                {元素 1,元素 2,元素 3,…},
                {第二行的值列表},
                …
                {第 n 行的值列表}
        };
```

可简化为

```
元素的数据类型[][] 二维数组的名称={
                {元素 1,元素 2,元素 3,…},
                {第二行的值列表},
                …
                {第 n 行的值列表}
        };
```

例如：

```
int[][] array=new int[][]{{1,2,3},{4,5,6},{7,8,9}};
//或者省略 new int[][]
int[][] array={{1,2,3},{4,5,6},{7,8,9}};
```

3）二维数组的本质

array 数组有 3 个数据元素 array[0]、array[1]、array[2]，如图 3-14(a)所示。而这 3 个元素又都存着一个一维数组，array[0]存放着地址 0x001，array[1]存放着地址 0x002，array[2]存放着地址 0x003，如图 3-14(b)所示。因此 array[0][1]、array[0][2]、array[0][3]就是读取 0x001 地址所在的一组数据的三个元数的值 1、2、3。

二维数组也可以看成一个二维表，行×列组成的二维表，array.length 是二维数组中的主数组的长度，可以表示数组的行数；array[i].length 是其中分数组的长度，即第 i 行的长度，也是第 i 行的列数。

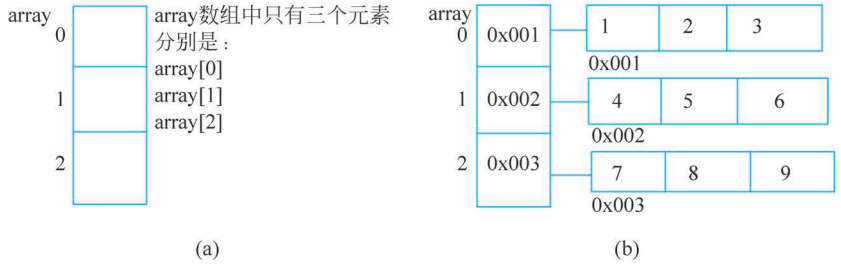

图 3-14　二维数组的存储空间示意图

3. 实现迷宫探路运算

(1) 探索通路：public boolean findPath()。

在迷宫类 Migong 中，定义 findpath()方法，该方法通过使用栈存储结构及其基本运算探找从迷宫的入口到出口的通路。栈中数据元素用类 Point 表示，该类拥有 x、y、d 三个成员变量，依次表示该点所在行坐标、列坐标及前进的方向。

```
//探测通路
    public boolean findPath() {
        Point point = null;                    //表示正在试探的位置点
        int x, y, d;                           //当前位置的坐标及前进的方向
        int x1 = 0, y1 = 0;                    //下一个位置的坐标
        point = new Point(1, 1, -1);           //入口位置点
        path.push(point);                      //入口位置点入栈
        while (!path.empty()) {
            point = path.pop();
            x = point.x;
            y = point.y;
            d = point.d + 1;                   //当前位置的下一个方向
            //根据当前位置前进的方向计算下一个位置的坐标
            while (d < 4) {
                switch (d) {
                case 0://向东前进
                    x1 = x ;
                    y1 = y+1;
                    break;
                case 1:                        //向南前进
                    x1 = x+1;
                    y1 = y ;
                    break;
                case 2:                        //向西前进
                    x1 = x;
                    y1 = y-1;
                    break;
                case 3://向北前进
                    x1 = x-1;
                    y1 = y;
                    break;
                }
                //如果下一个位置是可通的,当前位置入栈
                //探测下一个位置是否是可通的
                if (maze[x1][y1] == 1) {
                    point = new Point(x, y, d);
                    path.push(point);
                    //变换最新可通位置为当前位置
                    x = x1;
                    y = y1;
                    //表示该位置已被访问过
                    maze[x][y] = -1;
                    //如果该位置是出口,找到通路;否则,将探测方向设置为东
                    if (x == row - 2 && y == col - 2) {
                        point = new Point(x, y, -1);
                        path.push(point);
                        return true; /* 迷宫有通路 */
                    } else
                        d = 0;
                } else
                    d++;
            }
        }
        return false;                          /* 迷宫无通路 */
}
```

(2) 获取通路：public IStack < Point > getPath()。

在迷宫类 Migong 中，定义 getPath()方法，用于获取迷宫通路，该方法返回的是栈的类型，如果栈不为空表示有通路，如果为空，表示无通路。

```
//获取通路
public Point[] getpath() {
    Point[] points = new Point[path.size()];
    for (int i = points.length - 1; i >= 0; i--) {
        points[i] = path.pop();
    }
    return points;
}
```

4. 编写主类测试迷宫

创建一个测试类 TestMigong，对迷宫探路的运算算法进行测试。

```
public class TestMigong {
public static void main(String[] args) {
        int[][] map = { { 1, 1, 0, 1, 1, 1, 0, 1 },
                        { 1, 1, 0, 1, 1, 1, 0, 1 },
                        { 1, 1, 1, 1, 0, 0, 1, 1 },
                        { 1, 0, 0, 0, 1, 1, 1, 1 },
                        { 1, 1, 1, 0, 1, 1, 1, 1 },
                        { 1, 0, 1, 1, 1, 0, 1, 1 },
                        { 1, 0, 0, 0, 1, 0, 0, 1 },
                        { 0, 1, 1, 1, 1, 1, 1, 1 } };
        int row = map.length, col = map[0].length;
        System.out.println("迷宫矩阵:");
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                System.out.print(map[i][j] + " ");
            }
            System.out.println();
        }
        Migong migong = new Migong(map);
        if (migong.findPath()) {
            Point[] points = migong.getpath();
            System.out.println("可到达路径:");
            for (int i = 0; i < points.length; i++) {
                System.out.print("(" + points[i].x + ","
                                   + points[i].y + ") ");
            }
        }
        else {
            System.out.println("没有可到达路径!");
        }
    }
}
```

【任务评价】

请按表 3-3 查看是否掌握了本任务所学的内容。

表 3-3 “用顺序栈实现迷宫探路”完成情况评价表

序　号	鉴定评分点	分　值	评　分
1	能理解顺序栈的定义和存储特点	20	
2	能够进行顺序栈的算法设计	20	

续表

序　号	鉴定评分点	分　值	评　分
3	能编程实现顺序栈	20	
4	能对顺序栈进行测试	20	
5	能应用顺序栈实现迷宫探路	20	

3.4.3　用链栈实现迷宫探路

【学习目标】

(1) 理解栈的链式存储结构。

(2) 掌握链栈基本运算的实现方法。

(3) 能用链栈实现迷宫探路。

【任务描述】

前面用顺序栈实现了迷宫探路,但在用顺序栈存储通路时,是有容量限制的,最大容量为 maxsize,如果超过了这个容量,新探到的位置就无法加入通路中。为了解决这个问题,可以采用链式存储结构存储迷宫通路。本任务将栈的逻辑结构映射成链式存储结构,基于链式存储结构实现栈的基本运算,并将链栈应用在迷宫探路问题中。

【任务实施】

步骤一:将栈的逻辑结构映射成链式存储结构


视频讲解

1. 理解栈的链式存储结构

用链式存储结构存储的栈称为链栈。使用链栈的优点在于它能够克服用数组实现的顺序栈空间利用率不高的特点,但是需要为每个栈元素分配额外的指针空间用来存放指针域。

链栈通常用单链表来表示,如图 3-15 所示,它的实现是单链表的简化。所以,链栈结点的结构与单链表结点的结构一样,由数据域 data 和引用域 next 两部分组成,如图 3-16 所示。由于链栈的操作只是在一端进行,为了操作方便,把栈顶设在链表的头部,并且不需要头结点。

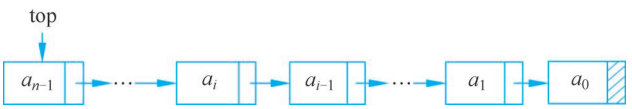

图 3-15　链栈的结构示意图

图 3-16　链栈的结点结构

2. 基于链式存储结构创建链栈类

(1) 创建一个链栈类 LinkStack < E >,该类实现接口 IStack < E >,接口中定义的基本运算的算法实现在步骤二完成。

(2) 创建一个内部类 Node,表示链栈的结点类型,包含两个成员变量:data 是结点数据域,类型为泛型;next 是下一个结点的引用域,类型为 Node。

(3) 定义一个类变量 top,为栈顶指示器,指向单链表的头结点。

```
public class LinkStack<E> implements IStack<E> {
    private class Node {
        E data;
        Node next;
```

```
        public Node(E data) {
            this.data = data;
        }
    }
    private Node top; //栈顶指示器
    //初始化链栈
    …
    //IStack 接口中定义的运算方法
    …
}
```

步骤二：完成链栈类中栈的基本运算算法

1. 初始化链栈

初始化链栈就是创建一个空栈，即调用 LinkStack<E>类的构造函数，初始化类变量设置栈顶指示器 top 为 null。

```
//初始化链栈
public LinkStack() {
    top = null;
}
```

此算法的时间复杂度为 $O(1)$。

2. 求链栈长度：size()

求链栈长度就是统计栈中数据元素的个数。

```
//求栈的长度
    public int size() {
        int size = 0;
        Node p = top;
        while (p != null) {
            size++;
            p = p.next;
        }
        return size;
    }
```

此算法的时间复杂度为 $O(n)$。

3. 判断为空：empty()

判断为空是判断栈顶指示器 top 的值是否为 null。

```
//判断链栈是否为空
    public boolean empty() {
         return top == null;
}
```

此算法的时间复杂度为 $O(1)$。

视频讲解

4. 入栈操作：push(E a)

入栈操作是将一个给定的数据元素保存在栈的最顶端，在链栈中，就是在单链表的起始处插入一个新结点。需要执行以下的步骤。

(1) 创建一个新结点，为新结点分配内存和值，如图 3-17 所示。

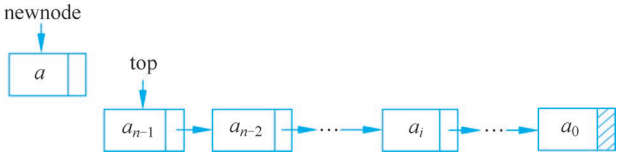

图 3-17　创建一个新结点

(2) 如果栈不为空,将新结点的 next 指向栈顶指示器 top 所指向的结点,如图 3-18 所示。

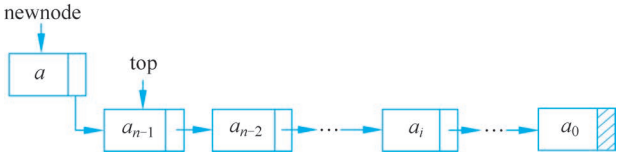

图 3-18　新结点的 next 指向 top 所指向的结点

(3) 将栈顶指示器 top 指向新结点,如图 3-19 所示。

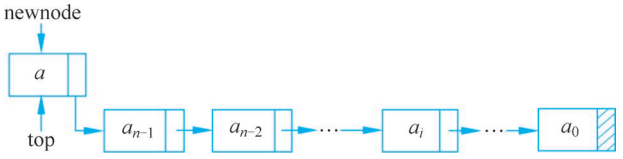

图 3-19　将 top 指向新结点

```
//入栈操作
    public E push(E a) {
        Node newnode = new Node(a);
        if (!empty())
            newnode.next = top;
        top = newnode;
        return a;
}
```


视频讲解

此算法的时间复杂度为 $O(1)$。

5. 出栈操作:pop()

出栈操作是从栈的顶部取出数据,即从链栈的起始处删除一个结点。需要执行以下的步骤。

(1) 如果栈不为空,获取栈顶指示器 top 所指向结点的值。

(2) 将栈顶指示器 top 指向单链表中下一个结点,如图 3-20 所示。

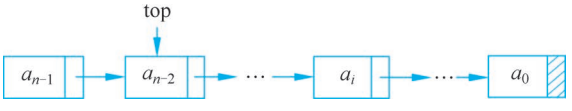

图 3-20　top 指向下一个结点

(3) 返回获取的栈顶结点的值,栈为空时返回 null。

```
//出栈操作
public E pop() {
```

```
        E a = null;
        if (!empty())
        {
            a = top.data;
            top = top.next;
        }
        return a;
    }
```

此算法的时间复杂度为 $O(1)$。

6. 取栈顶元素：peek()

取栈顶元素操作与出栈操作相似，只是取栈顶元素操作不改变原有栈，不删除取出的元素。

(1) 如果栈不为空，获取栈顶指示器 top 所指向结点的值。

(2) 返回获取的栈顶结点的值，栈为空时返回 null。

```
//获取栈顶数据元素
    public E peek() {
        E a = null;
        if (!empty())
            {
            a = top.data;
        }
    return a;
    }
```

此算法的时间复杂度为 $O(1)$。

步骤三：对链栈实现的基本运算的算法进行测试

将测试顺序栈中的代码：

```
IStack < Integer > stack = new
SeqStack < Integer >(Integer.class,data.length);
```

替换为

```
IStack < Integer > stack = new LinkStack < Integer >();
```

即可进行链栈的测试了。

步骤四：用链栈实现迷宫探路

在迷宫类 Migong 中用于存放路径的栈属性 sta 为接口类型 IStack，可用任何实现了 IStack 接口的类实例化，LinkStack 类实现该接口，因此属性 sta 可引用 LinkStack 类的实例对象。下面是迷宫 Migong 类中用顺序栈实现迷宫路径搜索问题的求解时构造函数的代码。

```
public Migong(int[][] map) {
        row = map.length + 2;
        col = map[0].length + 2;
        path = new SeqStack < Point >(Point.class,row * col);
        maze = new int[row][col];
        for (int i = 1; i < row - 1; i++)
            for (int j = 1; j < col - 1; j++)
                maze[i][j] = map[i - 1][j - 1];
    }
```

将上面代码中矩形框中的代码换为

```
path = new LinkStack < Point >();
```

就完成了用链式栈实现迷宫路径搜索问题的求解服务。

【任务评价】

请按表 3-4 查看是否掌握了本任务所学的内容。

表 3-4 “用链栈实现迷宫探路”完成情况评价表

序　　号	鉴定评分点	分　　值	评　　分
1	能理解链栈的定义和存储特点	20	
2	能进行链栈的算法设计	20	
3	能编程实现链栈	20	
4	能对链栈进行测试	20	
5	能够编写程序用链栈实现迷宫探路	20	

3.4.4 用 Java 类库实现迷宫探路

【学习目标】

(1) 熟悉 Java 中的 java.util.Stack 类。

(2) 熟悉 Java 中的 java.util.LinkedList 类实现栈操作的方法。

【任务描述】

在前面的任务中,先后使用顺序栈、链栈实现了栈的接口 IStack。实际上,Java 类库自身也提供了栈相关的类的 java.util.Stack 和 java.util.LinkedList 类,本任务使用 Java 类库中的 Stack 类和 LinkedList 类实现迷宫探路。

【任务实施】

步骤一:熟悉 Stack 类和 LinkedList 类

在 J2SDK 1.4.2 中,位于 java.util 包中的 Stack 类实现了顺序栈的功能,LinkedList 类提供了在列表开始和结尾添加与删除和显示数据元素的方法,使用这些方法把一个 LinkedList 当作链式栈使用。

1. Stack 类的方法

Stack 是 Vector 的一个子类,它实现标准的后进先出堆栈。Stack 只定义了创建空堆栈的默认构造方法。Stack 类里面主要实现的有以下几个方法。

(1) Object push(Object element):把元素压入栈。

(2) Object pop():移除堆栈顶部的对象,并返回该对象。

(3) Object peek():返回栈顶端的元素,但不从堆栈中移除它。

(4) boolean empty():判断堆栈是否为空。有时可能会看到使用 isEmpty()判断堆栈是否为空,它与 empty()从结果上来看并无区别,前者继承自 Vector,后者为 Stack 类自己定义的方法。

2. LinkedList 栈操作方法

(1) public E push(E e):把数据元素压入栈顶部,并返回该数据元素。

(2) public E pop():移除栈顶部的数据元素,并返回该数据元素。

(3) public E peek():查看栈顶部的数据元素,但不从栈中移除它。

(4) public int search(Object o)或 int indexOf(Object o)：返回指定数据元素在栈中的位置。

(5) public boolean empty()或 public boolean isEmpty()：测试栈是否为空。

步骤二：使用 Stack 类实现迷宫探路

(1) 在文件的开头导入包。

```
import java.util.Stack;
```

(2) 修改栈的类型。将 Migong 类中属性 path 声明为 Stack 类型并实例化为 Stack。

(3) 编译运行程序，观察运行结果。

步骤三：使用 LinkedList 类实现迷宫探路

(1) 在文件的开头导入包。

```
import java.util.LinkedList;
```

(2) 修改栈的类型。将 Migong 类中属性 path 声明为 LinkedList 类型并实例化为 LinkedList 对象。

(3) 编译运行程序，观察运行结果。

【任务评价】

请按表 3-5 查看是否掌握了本任务所学的内容。

表 3-5 “用 Java 类库实现迷宫探路”完成情况评价表

序　号	鉴定评分点	分　值	评　分
1	熟悉 Java 类库中 Stack 类有关栈运算相关方法	20	
2	会使用 Java 类库中 Stack 类表示迷宫路径	20	
3	熟悉 Java 类库中 LinkedList 类有关栈运算相关方法	20	
4	会使用 Java 类库中 Stack 类表示迷宫路径	20	
5	能够用 Stack 类和 LinkedList 类实现迷宫探路	20	

3.5 项 目 拓 展

1. 问题描述

汉诺塔问题来自一个古老的传说：在世界刚被创建时有一座钻石宝塔 A，其上有 64 个金碟，如图 3-21 所示。所有碟子按从大到小的次序从塔底堆放至塔顶。紧挨着这座塔有另外两个钻石宝塔 B 和 C。从世界创始之日起，婆罗门的牧师们就一直在试图把 A 塔上的碟子移动到 B 塔上去，其间借助于 C 塔的帮助。由于碟子非常重，因此，每次只能移动一个碟子。另外，任何时候都不能把一个碟子放在比它小的碟子上面。按照这个传说，当牧师们完成他们的任务之后，世界末日也就到了。

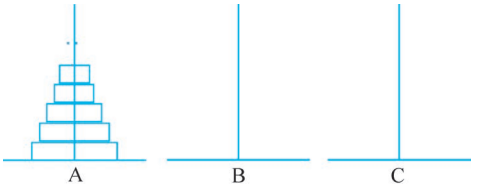

图 3-21　汉诺塔

在汉诺塔问题中，已知 n 个碟子和三座塔。初始时所有的碟子按从大到小次序从 A 塔的底部堆放至顶部，现在需要把碟子都移动到 B 塔，每次移动一个碟子，而且任何时候都不能把大碟子放到小碟子的上面。

2. 基本要求

(1) 编写一算法实现将 A 塔上的碟子移到 B 塔上,大碟在下,小碟在上。

(2) 将移动的过程显示出来。

3.6 项 目 小 结

本章在介绍栈的基本概念和抽象数据类型的基础上,重点介绍了栈及其操作在计算机中的两种表示和实现方法,并用栈解决了迷宫探路问题。

(1) 栈(Stack)是一种特殊的线性表,是一种只允许在表的一端进行插入或删除操作的线性表。允许插入和删除的一端称为栈顶(top),不允许插入和删除的一端称为栈底(bottom)。栈的主要特点是“后进先出”。

(2) 栈上可进行的主要操作有入栈、出栈和读栈顶元素。入栈是在栈顶添加一个数据元素;出栈是删除栈顶数据元素;读栈顶元素是获取栈中当前栈顶的数据元素,栈中数据元素不变。这些操作的时间复杂度都是 $O(1)$。

(3) 栈可以用顺序和链式两种存储方式,为此有顺序栈与链栈之分。其中,顺序栈用数组实现,链栈用单链表实现。

(4) 基于栈的思想还可以设计出其他一些变种的栈。例如,双端栈,是指一个线性表的两端当作栈底,分别进行入栈和出栈操作,主要利用了栈的栈底不变、栈顶变化的特征。

(5) 本项目以迷宫探路问题为主线,分析了迷宫探路问题中数据对象迷宫的通路就是由一系列的坐标点组成,而坐标点的数据关系为线性表;对线性表的插入和删除操作只能在一端进行,因此具有栈的特点;然后先后用顺序栈、链式栈及 Java 语言中的 Stack 类和 LinkedList 类解决了迷宫探路问题。

3.7 项 目 测 验

一、选择题

1. 栈中元素的进出原则是(　　)。

A. 先进先出　　B. 后进先出　　C. 栈空则进　　D. 栈满则出

2. 若已知一个栈的入栈序列是 1,2,3,…,n,其输出序列为 $p_1,p_2,p_3,\cdots,p_n$,若 $p_1=n$,则 p_i 为(　　)。

A. i　　B. $n=i$　　C. $n-i+1$　　D. 不确定

3. 若依次输入数据元素序列{a,b,c,d,e,f,g}进栈,出栈操作可以和入栈操作间隔进行,则下列哪个元素序列可以由出栈序列得到?(　　)

A. {d,e,c,f,b,g,a}　　B. {f,e,g,d,a,c,b}

C. {e,f,d,g,b,c,a}　　D. {c,d,b,e,g,a,f}

4. 一个栈的入栈序列是 1,2,3,4,5,则下列序列中不可能的出栈序列是(　　)。

A. 2,3,4,1,5　　B. 5,4,1,3,2

C. 2,3,1,4,5　　D. 1,5,4,3,2

5. 栈的插入与删除是在(　　)进行。

A. 栈顶　　B. 栈底　　C. 任意位置　　D. 指定位置

6. 设在栈中，由顶向下已存放元素 c，b，a，在第 4 个元素 d 入栈前，栈中元素可以出栈。d 入栈后，不可能的出栈序列是(　　)。

A. dcba　　B. cbda　　C. cadb　　D. cdba

7. 设栈的容量为 4，现有 ABCDEF 共 6 个元素顺序进栈，下列序列(　　)是不可能的出栈序列。

A. ADECBF　　B. AFEDCB　　C. CBEDAF　　D. CDBFEA

8. 以下哪一个不是栈的基本运算？(　　)

A. 新元素入栈　　B. 删除栈顶元素

C. 判断栈是否为空　　D. 删除栈底元素

9. 以顺序表作为栈的存储结构，假设顺序表的最大容量为 m 个元素，栈顶指针用栈顶元素所在位置的下标表示，判断栈为满的条件是(　　)。

A. 栈顶指针不等于 0　　B. 栈顶指针等于 0

C. 栈顶指针不等于 m　　D. 栈顶指针等于 $m-1$

10. 设链式栈中结点的结构为(data，next)，且 top 是指向栈顶的指针。若想摘除链栈的栈顶结点，并将被摘除结点的值保存到 x 中，则应执行的操作是(　　)。

A. x＝top. data；top＝top. next；

B. top＝top. next；x＝top. data；

C. x＝top；top＝top. next；

D. x＝top. data；

二、判断题

1. 同一个栈内各元素的类型可以不一致。(　　)

2. 栈是实现过程和函数等子程序所必需的数据结构。(　　)

3. 在执行顺序栈进栈操作时，必须判断栈是否已满。(　　)

4. 在链栈上执行进栈操作时，不需判断栈满。(　　)

5. 当问题具有先进先出特点时，就需要用到栈。(　　)

6. 一个栈的输入序列是 12345，则栈的输出序列不可能为 12345。(　　)

7. 栈和链表是两种不同的数据结构。(　　)

8. 若输入序列为 1，2，3，4，5，6，则通过一个栈可以输出序列 1，5，4，6，2，3。(　　)

9. 即使对不含相同元素的同一输入序列进行两组不同的合法的入栈和出栈组合操作，所得的输出序列也一定相同。(　　)

10. 栈是一种对所有插入、删除操作限于在表的一端进行的线性表，是一种后进先出型结构。(　　)