数据结构与算法

实验二　扩展线性表

实验目的

- 掌握如下内容：
  - 进一步掌握指针、模板类、异常处理的使用
  - 掌握栈的操作的实现方法
  - 掌握队列的操作的实现方法
  - 学习使用栈解决实际问题的能力
  - 学习使用队列解决实际问题的能力
  - 学习使用多维数组解决实际问题的能力。

题目一：栈和队列

- 根据栈和队列的抽象数据类型的定义，按要求实现一个栈或一个队列的基本功能（四选一）。

要求：

1、实现一个共享栈

2、实现一个链栈

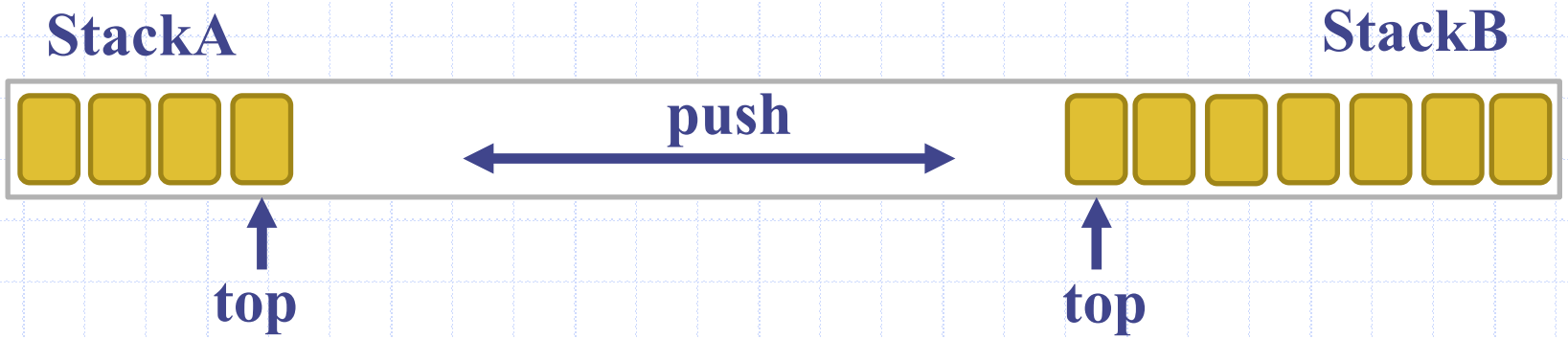
3、实现一个循环队列

4、实现一个链队列

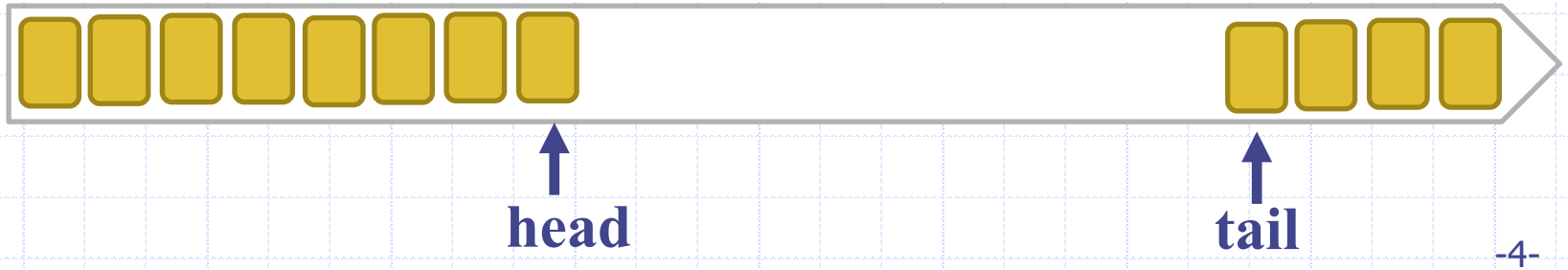
编写测试main()函数测试栈或队列的正确性

题目一：栈和队列

- 共享栈



- 循环队列



题目二：稀疏矩阵

- 根据三元组的抽象数据类型的定义，使用三元组表实现一个稀疏矩阵。

三元组的基本功能：

- 三元组的建立
- 三元组转置
- 三元组相乘
- 其他：自定义操作

编写测试main()函数测试三元组的正确性

题目二：稀疏矩阵

- 链表节点

$$(AB)_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{ip} b_{pj}$$

```
template <class T>
struct Tuple
{
    int row;
    int col;
    T data;
    Tuple<T>  *next;
};
```

- 需要注意在进行一些操作之后，处理一些值为0的节点

题目二：稀疏矩阵

测试数据：

$$\begin{pmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 13 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

题目三：八皇后问题

- 八皇后问题19世纪著名的数学家高斯于1850年提出的。他的问题是：在8*8的棋盘上放置8个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列、同一斜线上。
- 请设计算法打印所有可能的摆放方法。

提示：

- ➢可以使用递归或非递归两种方法实现
- ➢实现一个关键算法：判断任意两个皇后是否在同一行、同一列和同一斜线上

题目三：八皇后问题

- 递归想法：

给定已放置的皇后位置，逐行判断皇后可能摆放的位置：

（1）如果已经摆放完成，打印并返回（行--）

（2）若某位置可以放置皇后，探索下一行（递归调用）

（3）若均不能放置，返回（行--）

题目三：八皇后问题

- **非递归想法：**

使用数字存储各行皇后位置

初始化下标k和数组a

对第k行：

循环判断可放皇后的位置，记录到数组

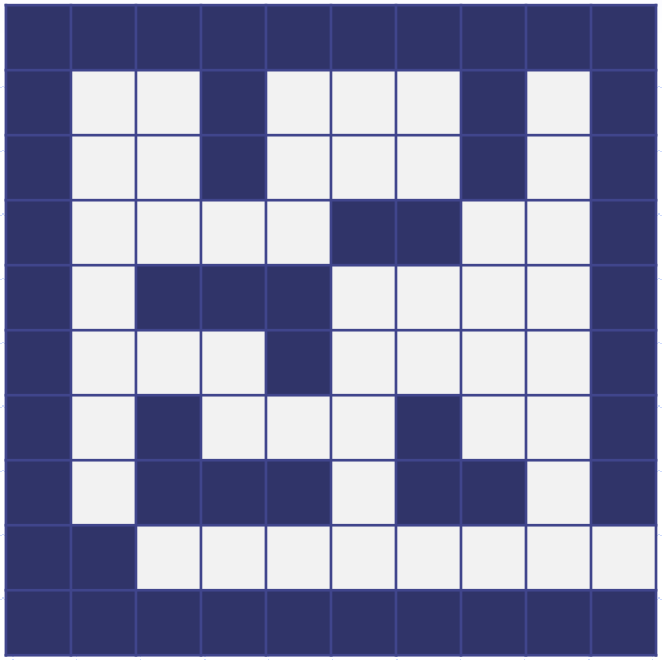
k == N-1?

k != N-1，找到可放位置?

找不到可放位置?

题目四：迷宫求解问题

- 心理学家把一只老鼠从一个无顶盖的大盒子的入口赶进迷宫，迷宫中设置很多隔壁，对前进方向形成了多处障碍，心理学家在迷宫的唯一出口放置了一块奶酪，吸引老鼠在迷宫中寻找通路以到达出口，测试算法的迷宫如图所示。



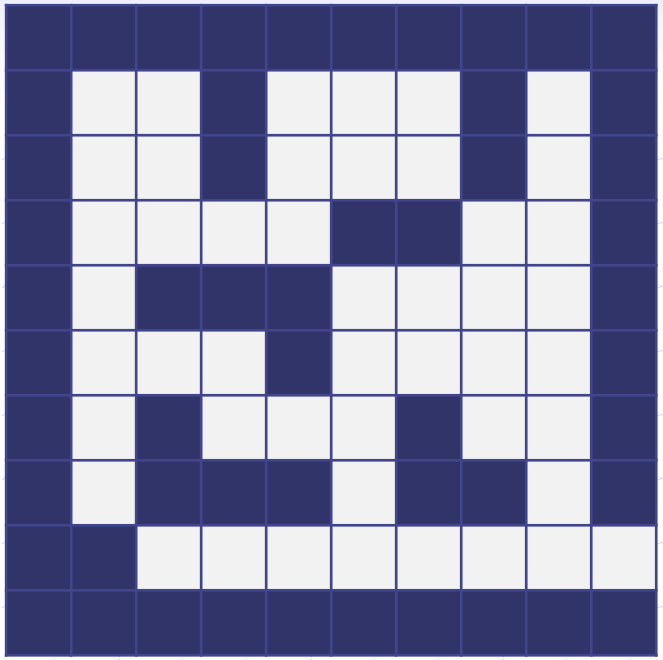
- 算法需找到任意一条没有重复的路径并打印，函数结果返回是否能够找到出口

题目四：迷宫求解问题

提示：

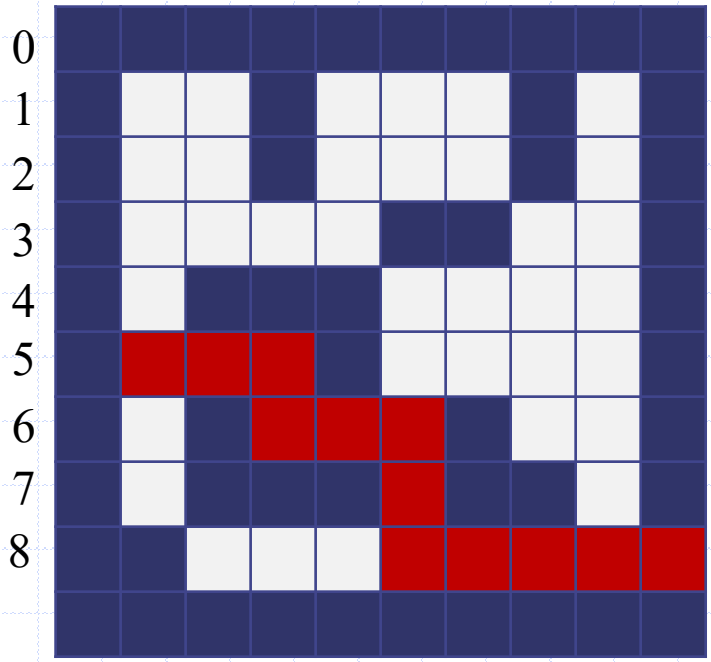
- **可以使用递归或非递归两种方法实现**
- **老鼠能够记住已经走过的路，不会反复走重复的路径**
- **可以自己任意设置迷宫的大小和障碍**
- **使用“穷举求解”的方法**
- **不需要“最短”**

题目四：迷宫求解问题



- 广度遍历(用队列实现)

```
Bfs(当前位置){
        当前位置入队;
        while（队不为空）{
                位置 i = 队首;
                if （可走&&!位置i的邻接节点走过）
                        该邻接节点入队;
        }
}
```


0
1
2
3
4
5
6
7
8

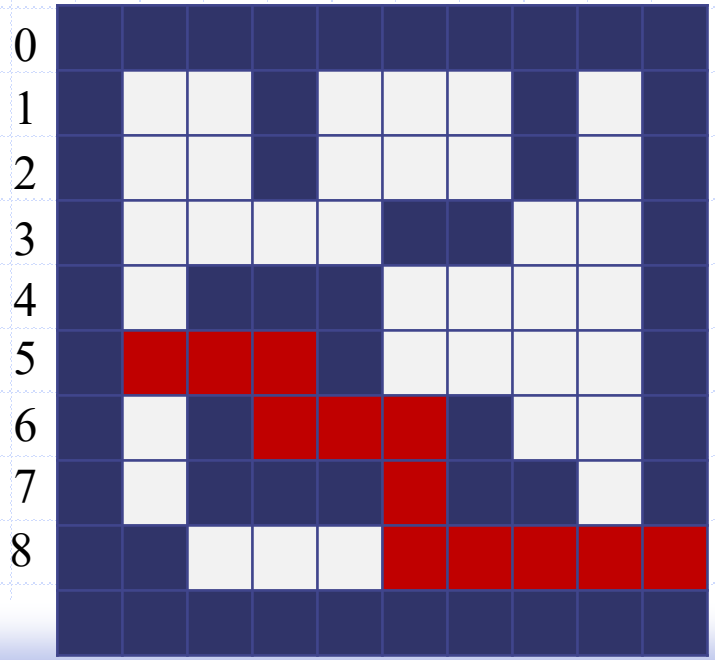
题目四：迷宫求解问题

- 深度遍历（用栈实现）

```
Dfs (当前位置){
        if (某个方向走得通 && 没走过){
                Dfs(当前位置+方向);
        }
        else
                return;
}
```

方向：可以设为[下 右 左 上]

方向的选择影响路径长度



题目五：表达式求值

- 表达式求值是程序设计语言编译中最基本的问题，它要求把一个表达式翻译成能够直接求值的序列。
- 例如输入字符串“14+((13-2)*2-11*5)*2”，程序可以自动计算得到最终的结果。在这里，我们将问题简化，假定算数表达式的值均为**非负整数常数**，不包含变量、小数和字符常量。
- 试设计一个算术四则运算表达式求值的简单计算器。

题目五：表达式求值

要求：

- 操作数均为非负整数常数
- 操作符仅为+、-、*、/、(、)
- 编写main函数进行测试

题目五：表达式求值

- 优先级

括号 > 乘号或除号 > 加减号

- **操作符栈与操作数栈**

（1）遇到数字则进行入栈（操作数栈）

（2）遇到操作符：

判断是否是括号

比较当前操作符优先级与操作符栈顶的优先级

（3）计算：从栈中弹出两个操作数和操作符，进行计算后压入操作数栈

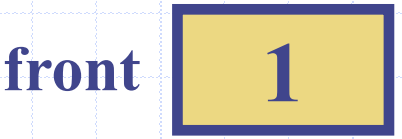
题目六：车厢重排问题

- 用**队列结构**实现车厢重排问题。
- 车厢重排问题如下：一列货车共有n节车厢，每个车厢都有自己的编号，编号范围从1~n。给定任意次序的车厢，通过转轨站将车厢编号按顺序重新排成1~n。
- 转轨站的**缓冲轨数目不固定**，缓冲轨位于入轨和出轨之间。开始时，车厢从入轨进入缓冲轨，经过缓冲轨的重排后，按1~n的顺序进入出轨。缓冲轨按照**先进先出方式**，编写一个算法，将任意次序的车厢进行重排，输出每个缓冲轨中的车厢编号。

题目六：车厢重排问题

提示：

一列火车的每个车厢按顺序从入轨进入不同缓冲轨，缓冲轨重排后出轨，重新编排成一列货车。比如：编号为3的车厢进入缓冲轨1，则下一个编号小于3的车厢则必须进入下一个缓冲轨2，而编号大于3的车厢则进入缓冲轨1，排在3号车厢的后面，这样，出轨的时候才可以按照从小到大的顺序重新编排。

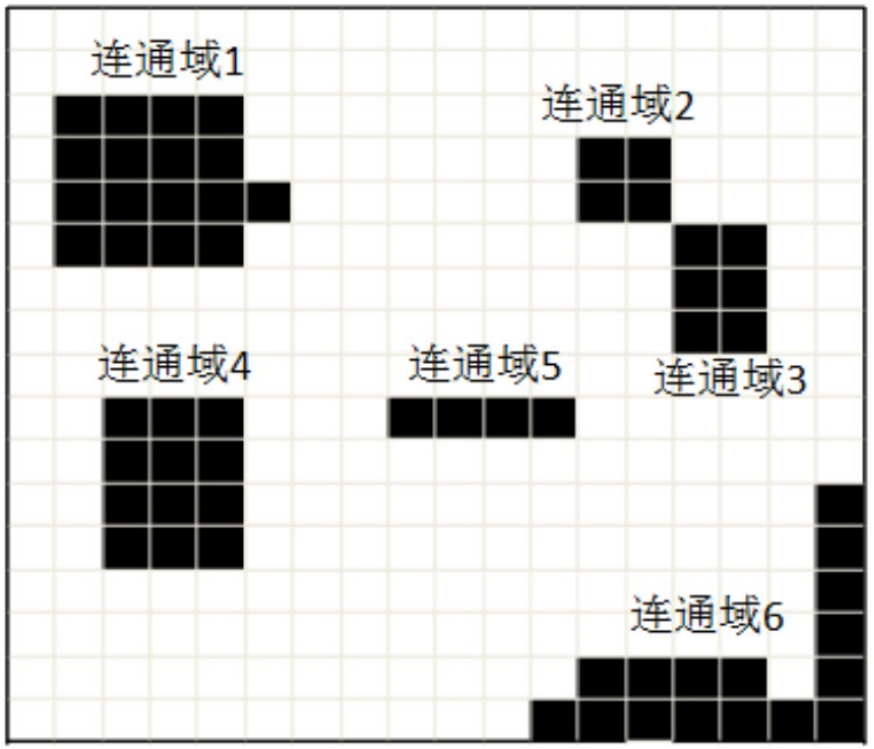


题目七：连通域问题

- 在仅有黑色像素和白色像素的图像中，将**相邻的黑色像素构成的点集**称为一个连通域。
- 连通域标记算法把连通区域所有像素设定同一个标记，常见的标记算法有四邻域标记算法和八邻域标记算法。

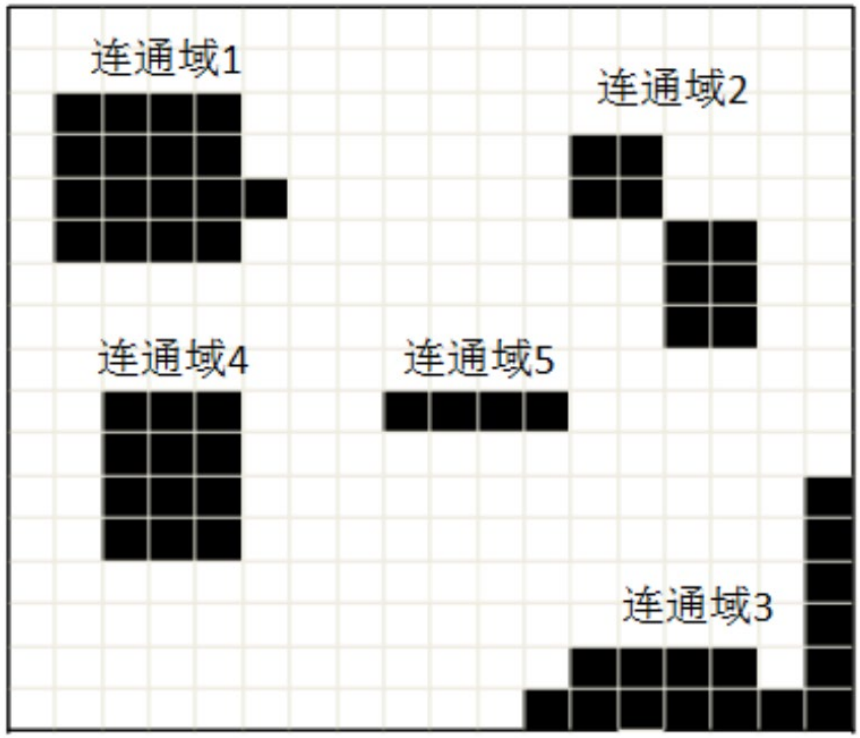
题目七：连通域问题

- 四邻域标记算法中，当前黑点与上、下、左、右任意相邻黑点属于同一连通域。
- 如图，给出了四邻域连通域示意图。



题目七：连通域问题

- 八邻域标记算法中，当前黑点与上、下、左、右及左上、左下、右上、右下任意相邻黑点属于同一连通域。
- 如图，给出了八邻域连通域示意图。



题目七：连通域问题

- 采用三元组来编写二值图像四邻域连通域标记算法，设图像采用01矩阵表示。

要求：

- ➢算法尽可能优化
- ➢输出每个像素点所属的连通域标记
- ➢编写测试main()函数测试三元组的正确性

题目七：连通域问题

- 对于一个黑色像素点，考虑如下几种情况：

该像素点的连通域编号与周围节点的关系

周围有连通域节点：设置连通域标记

周围没有连通域节点：标记新的连通域

边界情况?

搜索顺序?

是否存在连通域的合并？是否存在冗余过程?

题目七：连通域问题

测试数据：

1010101
1010111
1011001
1001111
1110001
0010001
1111111

题目八：识别BMP文件

- 实现一个识别BMP文件的图像类，能够进行以下图像处理

要求：

- ➢能够将24位真彩色Bmp文件读入内存
- ➢能够将24位真彩色Bmp文件重新写入文件
- ➢能够将24位真彩色Bmp文件进行24位灰度处理
- ➢能够将24位灰度Bmp文件进行8位灰度处理
- ➢能够将8位灰度Bmp文件转化成黑白图像
- ➢能够将图像进行平滑处理
- ➢其他自定义操作：如翻转、亮度调节、对比度调节、24位真彩色转256色等

题目八：识别BMP文件

提示：

➢参考教材《数据结构与STL》第四章4.4小节

➢灰度处理的转换公式：

Grey=0.3*Red+0.59*Blue+0.11*Green

➢平滑处理采用邻域平均法进行，分成4邻域和8邻域平滑，基本原理就是将每一个像素点的值设置为其周围各点像素值得平均值

➢ 24位真彩色转256色，需要手动添加颜色表在BMP头结构中，可以使用位截断法、流行色算法、中位切分算法、八叉树算法等方法实现

题目八：识别BMP文件

➢亮度调节公式，a为亮度调节参数，0<a<1,越接近0，变化越大：

R=pow(R,a)*pow(255,1-a)

G=pow(G,a)*pow(255,1-a)

B=pow(B,a)*pow(255,1-a)

题目八：识别BMP文件

➢对比度调节公式，a为对比度调节参数，-1<a<1，中间值一般为128：

R=中间值+(R-中间值)*(1+a)

G=中间值+(G-中间值)*(1+a)

B=中间值+(B-中间值)*(1+a)

➢注意：调整对比度的时候容易发生越界，需要进行边界处理

程序要求

1.注意内存的动态申请和释放，是否存在内存泄漏；
2.优化程序的时间性能；
3.递归程序注意调用的过程，防止栈溢出；
4.保持良好的编程风格：

- ➢代码要简洁；
- ➢代码段与段之间要有空行和缩近；
- ➢标识符名称应该与其代表的意义一致；
- ➢函数名之前应该添加注释说明该函数的功能；
- ➢关键代码应说明其功能。