 SHANGHAI UNIVERSITY

**<数据结构>研讨报告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **总分100** | | | | **得分** |
| 报告 | 逻辑结构（40分） | | |  |
| 语言表达（30分） | | |  |
| 格式规范（20分） | | |  |
| 个人体会（10分） | | |  |
| **合 计** | | | |  |
| **学 院** | **计算机工程与科学学院** | | |
| **组 名** | **陈宇鑫小组** | | |
| **题 目** | **区域填充** | | |
| **日 期** | **2023-12-27** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **学号** | **姓名** | **专业** | **分工** |
| 22121402 | 陈宇鑫 | 人工智能 | 实验报告撰写，程序验收和改进 |
| 22121497 | 郑凯风 | 智能科学与技术 | PPT制作，程序验收和改进 |
| 22120051 | 党昊天 | 智能科学与技术 | MFC界面化功能实现，程序健壮性测试 |
| 22121496 | 古彬玮 | 人工智能 | 四连通代码编写 |

1. **目的与要求**

1. 掌握栈和队列的逻辑结构定义和各种存储结构的实现。

2. 熟练运用栈和队列的各种存储结构以及各种基本操作。

3. 根据实际问题的需要，选择栈和队列适合的存储结构解决问题。

**二、实验环境**

Windows操作系统

Visual Studio 2022（包含MFC库）

**三、实验内容**

**[问题描述]**区域填充是计算机图形学中的一个基本问题，即将屏幕上指定区域内的像素以某种颜色对其进行着色。种子填充算法是区域填充的一个基本方法，该方法以待填充区域中的任一点为起点（种子点），按照上、下、左、右的顺序对区域中的像素点进行着色。如下图1所示，以标记为1的点为起点，则区域内象素点的着色的顺序按照数字标记由小到大的顺序进行。

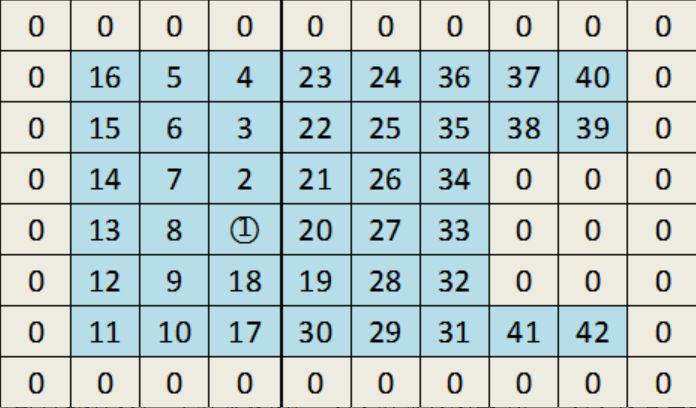


图1 种子填充示意图

**[基本要求]** 用一个20\*20的矩阵模拟屏幕，每个矩阵元素代表一个屏幕像素，0表示屏幕背景区域，-1表示待着色区域。使用栈的结构（非递归）完成上述区域填充算法。

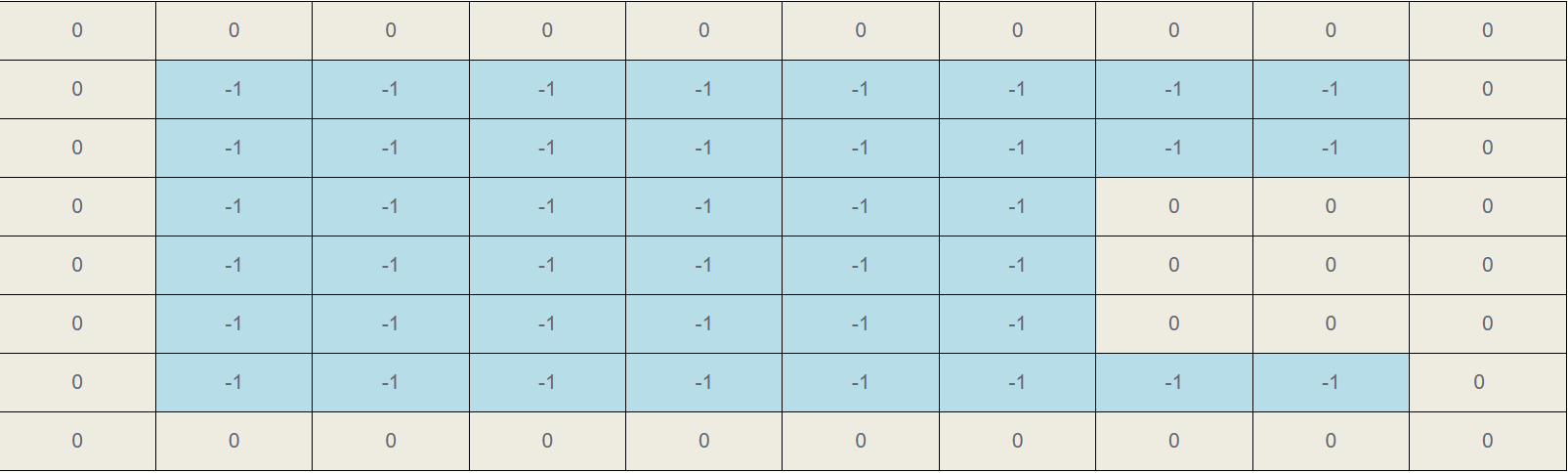


图2 模拟屏幕示意图

**[输入数据]** 1) 大小为20\*20的矩阵，其中0代表屏幕背景像素，-1代表需要着色的区域像素；（要考虑不同形状的待填充区域）2) 待着色区域中的种子点位置。

**[输出数据]** 输出屏幕矩阵，其中表示背景区域的元素仍为0，表示着色区域的元素值为按照着色先后顺序从1开始递增的整数，如上图1所示。

**[挑战性内容]** 1) 实现另一个更高效的基于扫描线的区域填充算法（具体算法描述请自行查阅资料）。2) 使用图形化窗口界面实现区域填充算法，通过鼠标交互选择种子点。

**四、实验内容的设计与实现**

在本次基于栈的区域填充实验中，我们按照要求使用种子填充算法实现区域填充。所谓种子填充，是指以待填充区域内一点为基础向四周进行扩展，对满足条件的像素点进行颜色绘制的一种思想。从像素点是否需要绘制的判断方式出发，又分为边界表示和内点表示两种方法；从待填充像素点向四周扩展的方式出发，又分为四连通和八连通两种方式。

所谓**边界表示**，就是指规定边界颜色，若当前像素点颜色是边界颜色，那么我们认为到达图形边界，对该像素点我们不进行绘制，并且也不会从这个点出发进行像素点的扩展；反之，若当前像素点颜色不是边界颜色，那么无论它是什么颜色的，我们都认为这是属于待填充图形的内点，需要进行绘制并且向外扩展。

所谓**内点表示**，就是规定待填充图形内部像素点的颜色特征，若当前像素点颜色是内点颜色，那么我们认为该点在图形内部，进行颜色绘制并且向四周扩展；反之，若当前颜色不是内点颜色，那么无论它是什么颜色，我们都认为当前已经到达了图形边界，那么对该像素点不行绘制处理并且也不能向外扩展。

**四连通扩展方式**：假设当前判断像素点(xpos, ypos)应该绘制，那么我们在绘制这个点的颜色后，应当对这个点四周的点是否也应当进行绘制，四连通方式就是对这个点的左、上、右、下四的像素点逐一进行类似操作（是否绘制以及扩展操作），进而递归式的扩展到整个区域。

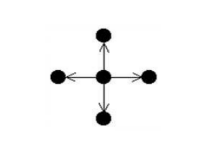


图1 四连通扩展方式

**八连通扩展方式**：假设当前判断像素点(xpos, ypos)应该绘制，那么我们在绘制这个点的颜色后，应当对这个点四周的点是否也应当进行绘制，八连通方式就是对这个点的左、左上、上、右上、右、右下、下、左下八的像素点逐一进行类似操作（是否绘制以及扩展操作），进而递归式的扩展到整个区域。

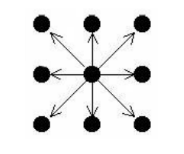


图2 八连通扩展方式

本次实验中我们的种子填充算法采用的是边界表示+四连通扩展方式。

下面谈谈**栈对于种子填充算法的作用**。

在种子填充算法中，栈的作用是存储待填充区域的边界像素点的坐标。当一个像素点被着色后，该点的邻接像素点（上、下、左、右）如果也属于待填充区域并且还未被着色，就将它们的坐标入栈，等待下一次着色。通过不断将待填充区域的边界像素点入栈和出栈的操作，可以实现对整个区域的填充。栈的先进后出的特性使得每个像素点的填充顺序是按照广度优先遍历的方式进行的。

具体操作的介绍如下：

1. 首先将种子点（指定区域内的一个起点）入栈，然后进入循环，判断栈是否为空。
2. 出栈一个点，命名为Temp\_Point，判断Temp\_Point是否在指定区域内（即屏幕的有效范围内）。
3. 如果Temp\_Point在区域内，获取Temp\_Point的颜色值，并判断颜色值是否需要填充，如果需要填充，则将Temp\_Point着色为指定的颜色。
4. 根据Temp\_Point的上、下、左、右四个相邻点的位置，计算出这四个点的坐标。
5. 对于每个相邻点，判断它是否在区域内，并且不是已经填充过的颜色，如果满足条件，则将该相邻点入栈。
6. 这样循环遍历整个区域，直到栈为空，完成区域填充。

下面是实现区域填充的关键代码段：

|  |
| --- |
| if (color != ShapeColor&&color != LineColor)  {  // 填充当前点  dc.SetPixel(Temp\_Point.x, Temp\_Point.y, ShapeColor);  //dc.SetPixel(point.x, point.y, ShapeColor);  leftPoint = Temp\_Point;  rightPoint = Temp\_Point;  upPoint = Temp\_Point;  downPoint = Temp\_Point;  leftPoint.Offset(-1, 0);  rightPoint.Offset(1, 0);  upPoint.Offset(0, -1);  downPoint.Offset(0, 1);  if (rightPoint.x >= 0 && rightPoint.x < width && rightPoint.y >= 0 && rightPoint.y < height && dc.GetPixel(rightPoint) != ShapeColor && dc.GetPixel(rightPoint) != LineColor)  {  stack.Push(rightPoint);  }  if...(继续对leftPoint、upPoint和downPoint执行相同的操作) |

首先判断当前点的颜色是否需要进行填充，即是否不等于指定的填充颜色和线条颜色。如果需要进行填充，则将当前点填充为指定颜色。然后计算了当前点的上、下、左、右四个相邻点的坐标。接着通过一系列判断，将满足条件的相邻点入栈，以便下一次迭代时进行填充。这样就实现了对当前点的填充，并扩展了待填充区域。从而实现了种子填充算法的区域填充功能。

下面介绍我们的图形化窗口界面实现区域填充算法过程。

我们通过学习MFC应用程序相关知识，实现了图形化窗口界面，以及可以通过鼠标交互选择种子点。在实现类Cseed20Dlg的过程中，有几个值得注意的函数：

在函数OnInitDialog中，我们进行了**对话框的初始化操作**。首先，我们调用基类函数CDialogEx::OnInitDialog进行默认的对话框初始化。包括RGB函数，设置了变量LineColor和 ShapeColor的颜色值。RGB函数用于创建一个RGB颜色值。我们设置了变量LineWidth的默认值为 5，表示绘制线条时的线宽。然后，我们向下拉框控件（m\_fill）添加了两个选项，分别是 "有填充" 和 "无填充"，并将默认选择设置为 "无填充"。再向下拉框控件（m\_Mode）添加了五个选项，分别是 "点"、"线"、"椭圆"、"自由" 和 "填充模式"，并将默认选择设置为 "点"。最后，我们使用UpdateData函数来更新与变量相关的数据绑定。这里将FALSE作为参数传递，表示将对话框中的数据更新到变量中。最后，我们将Mode设置为PointMode，表示默认绘图模式为点。最后，我们返回TRUE来指示对话框初始化成功。



图3 对话框初始化

在函数OnLButtonDown中，我们**处理鼠标左键按下的消息**。首先，我们**将当前鼠标坐标保存到变量DownPoint中，作为种子点**。接下来，我们根据当前的绘图模式Mode进行操作。如果绘图模式是PointMode，表示选择了点模式，我们创建一个CClientDC对象dc，用于在对话框上进行绘图操作。然后，我们使用SetPixel函数绘制一个像素点，将其位置设置为鼠标按下的点坐标，并使用 LineColor设置点的颜色。如果绘图模式是OwnerDrawingMode，表示选择了自由绘制模式，我们将变量 startdrawing设置为true，表示开始进行自由绘制操作，并将变量 lastpoint设置为当前鼠标坐标。最后，我们调用基类函数CDialogEx::OnLButtonDown来处理其他的默认操作。

程序运行示例如下：

设置填充颜色为红色，画笔粗细为默认值，无填充模式，对一个椭圆内部进行填充。



图4 填充示例1

然后选择填充模式，并点击椭圆内部任意一点，开始填充。



图5 填充示例2

填充完毕。

**五、收获与体会**

陈宇鑫：

在本次实验中我学习了栈的数据结构和算法。通过实现种子填充算法，我深入了解栈的工作原理和应用场景。在与小组成员一起合作的过程中，大家各司其职。在协调和沟通中我也锻炼了团队合作能力，并最终合作完成整个项目。通过使用MFC库实现图形化窗口，我熟悉了MFC框架的基本概念和常用的类和函数。这将对我在开发Windows应用程序时非常有帮助。通过实现种子填充算法和探究图形化窗口，我也了解图形绘制的原理和技术。以上种种都对我未来的学习和工作有很大的帮助。

郑凯风：

通过这次研讨，我学习了栈这一数据结构在实际应用中的作用。并拓展学习了程序可视化的知识：通过对MFC库中各函数和类的学习，我了解了MFC框架的基本功能，我想这对我增进前端开发的知识很有帮助。在制作展示材料的过程中，我深刻认识到理工类项目展示要求逻辑清晰、表达准确；我也通过这次展示材料的制作积累了更多项目展示的经验。虽然本次研讨的时间紧迫，主题综合性很强，但我们小组通过成员之间的团结合作，最终成功完成了项目；这让我意识到团队合作的重要性，也提高了我团队合作的能力，我想在以后的学习过程中我会更加积极参与团队合作，力争取得质量更高的成果。

古彬玮：

通过编写种子填充算法，我深刻理解了栈的应用。该算法以用户输入的种子点为起点，按上下左右的顺序填充未着色区域，每次填充一步后暂停等待用户输入。这一交互式的实现方式使得填充过程更直观，并有效展示了栈在图形学中的重要作用，使得用户可以逐步观察填充的过程，更好地理解算法的执行流程。

党昊天：

在本次研讨中，我进行了MFS的学习和图形化页面的制作，将填充算法实际运用到程序中去，而不仅限于在命令提示框中对算法进行模拟，这让我获益匪浅。首先，我大致掌握了MFS这一GUI界面的使用方法，为我今后开发类似项目的图形界面奠定基础，对图形化界面制作的学习也让我掌握了这类程序的开发思路，对QT等工具也有了一定程度的了解。其次，我深刻意识到一个算法效率，其本身的时间以及空间复杂度对程序的本身影响，种子填充算法在填充算法中效率一般，具有较大的优化空间，具体在程序表现为填充速度慢，频繁且大量的开空间对程序的健壮性产生较大的影响。最后，我切身体会到栈在程序中的应用，种子算法利用栈的数据结构与DFS算法产生了相似的作用，这易于开发人员理解，但却在一定程度上加重了机器运行的负担。

**参考文献：**

# csdn文章 VS2022 的 MFC 安装之 Hello World ---- IT笔记之1

# csdn文章【计算机图形学课程】一.MFC基本绘图函数使用方法

# B站视频《带你做一个最常见的MFC绘图系统，拯救期末！(开发全程录屏)》