《数据结构》

综合项目

题 目： 迷宫策略

学 院： 计算机与信息安全学院

专 业： 信息安全专业

组 长：

组 员：

2021年12月3日

小组分工

蔡响：代码实现，文档编写，PPT修改，可视化

陆腾辉：代码实现，文档编写

钟与旺：代码实现，文档编写

廖梓华：代码实现，文档编写

陆元秋：PPT修改，可视化

冯楚菲：文档编写，可视化

成绩评定标准及成绩

1. 能够按照格式要求、排版要求和字数要求等，有需求分析，系统设计与实现，系统测试和参考文献。（15分）
2. 报告内容行文通畅，有条理性，无错别字，无抄袭等。（15分）
3. 在验收过程中，能合理的回答问题（20分）
4. 软件运行正常，实现所提出的功能（40分）
5. 代码规范、具有自己的创新或特色（10分）

总成绩：

### 摘要

走迷宫是一种简单易玩的游戏，许多人都玩过走迷宫。每个人都会采取不同的策略走迷宫，但是怎样的策略才能最快的找到出口呢？过去我们并没有将走迷宫的过程整理成一套自己的体系，而是采用随缘的方式寻找出口。

随着学识的增长，我们学习了数据结构与算法，了解了不同的搜索策略。为了将学习与实际结合，顺便探索小时的游戏策略，故此本次实验将基于栈、队列、广度优先和深度优先算法，红黑树的A\*算法完成的迷宫搜寻策略，主要实现是在固定方格内寻找路径、最短路径以及生成随机迷宫图的过程。

在本次实验中，为了探索了不同算法在不同复杂程度迷宫下的情况，采用Prim生成，深度生成，递归生成三种生成方式来模拟生成不同特点不同大小的迷宫进行实验，根据计算不同迷宫算法在不同迷宫中的时间复杂度和搜索次数，最终得出BFS搜索次数最多，A\*算法寻找效率高的对比结论。

本次实验中，我们设计了相关的可视化界面，界面设计思路源于童年游戏回忆，界面主题采用植物大战僵尸和马里奥两款游戏进行融合参考，整合为可视化结果，对迷宫搜索进行可视化展示。

关键字：数据结构，C，C++，python，可视化

目录

[第一章 简介 1](#_Toc19669)

[1.1. 迷宫算法 1](#_Toc14776)

[1.2. 地图界面 1](#_Toc16183)

[1.3. 实验结果对比 1](#_Toc31860)

[第二章 迷宫路线搜索算法 2](#_Toc1275)

[2.1. 栈与队列 2](#_Toc28255)

[2.1.1. 顺序栈 2](#_Toc7739)

[2.1.2. 链栈 2](#_Toc11202)

[2.1.3. 顺序队列 3](#_Toc18504)

[2.1.4. 链队列 3](#_Toc3604)

[2.2 图的遍历和图的存储结构 3](#_Toc27267)

[2.2.1邻接矩阵表示法 4](#_Toc3056)

[2.2.2.邻接表 5](#_Toc750)

[2.3 迷宫算法-深度优先搜索策略 6](#_Toc19925)

[2.3.1. 简介 6](#_Toc22633)

[2.3.2. 思想 7](#_Toc27070)

[2.3.3. 特点 8](#_Toc32657)

[2.4 迷宫算法-广度优先搜索策略 8](#_Toc2129)

[2.4.1. 简介 8](#_Toc16797)

[2.4.2. 思想 9](#_Toc26014)

[2.4.3. 特点 9](#_Toc1951)

[2.4.4. 广度优先搜索与深度优先搜索的对比 9](#_Toc15998)

[2.5 迷宫算法-A\*算法 10](#_Toc11887)

[2.5.1. 简介 10](#_Toc8037)

[2.5.2. A\*算法的实现思路 10](#_Toc20181)

[2.6 迷宫算法-Dijkstra算法 10](#_Toc11887)

[2.6.1. 简介 10](#_Toc8037)

[2.6.2. 思想 1](#_Toc20181)1

[第三章 随机迷宫生成算法 11](#_Toc11131)

[3.1 Prime 迷宫生成算法 11](#_Toc16091)

[3.2 递归分割算法 11](#_Toc2612)

[3.3 深度优先算法 12](#_Toc21353)

[3.4 总结对比 13](#_Toc766)

[第四章 实验结果 13](#_Toc27602)

[4.1. 算法搜索时间对比 13](#_Toc16142)

[4.1.1. 总体柱状图对比 13](#_Toc32480)

[4.1.2. 算法对比结果说明 14](#_Toc10954)

[4.2. 算法搜索次数对比 14](#_Toc23834)

[第五章 项目代码介绍 15](#_Toc31599)

[5.1 项目总体结构 15](#_Toc14936)

[5.2 总体情况 16](#_Toc17109)

[总结 16](#_Toc24805)

[参考文献 1](#_Toc2700)6

### 第一章 简介

### 迷宫算法

本次实验中我们使⽤了使用了深度优先搜索算法、⼴度优先搜索算法、A\*算法、Dijkstra算法种算法实现了迷宫的路径搜索。

1. 深度优先搜索算法（DFS）。该算法的主要思路为沿着树的深度遍历树的节点，尽可能深的搜索树的分⽀。当节点v的所有边都被探寻过，搜索将回溯到发现节点v的那条边的起始节点。这⼀过程⼀直进⾏到从源节点可以发现可达节点为⽌。如果还存在未被发现的节点，则选择其中⼀个作为源节点并重复以上过程，整个进程反复进⾏直到到达终点或者所有节点都被访问为⽌，属于盲目搜索。
2. 宽度优先搜索算法（BFS）是最简便的搜索算法之⼀，这⼀算法也是很多重要的图的算法的原型。Dijstra单源最短路径算法和Prim最⼩⽣成树算法都采⽤了和宽度优先搜索类似的思想，其别名⼜叫做BFS，属于⼀种盲⽬搜寻法。它并不考虑结果的可能位置，彻底地搜索整张图，直到找到结果或者所有结点都被遍历。
3. A\*算法是⼀种静态⽹络中求解最短路径最有效的直接搜索⽅法，也是许多其他问题的常⽤启发式算法。
4. Dijkstra算法采用的是一种贪心的策略，单源点的最短路径问题：给定带权有向图G和源点v，求从v到G中其余各顶点的最短路径。

### 地图界面

地图界面的设计想法来源于小时候十分热门的两款游戏，基于植物大战僵尸和马里奥的游戏对界面进行了设计。草地可以说是植物大战僵尸里最普遍的场景，马里奥的人物形象到现在也是非常受欢迎的。选用这两个方面的元素，一方面是对童年的回忆，小游戏的陪伴是小时候不可或缺的乐趣；另一方面，尝试将其进行整合，是觉得彼此的元素都符合我们整个迷宫算法图界面的卡通、美化等需求。起始的界面为草地，能走的通的路显示为地板，移动人物为马里奥，迷宫的起点为一个方向牌，出口处放的是一个杨桃。迷宫走法的思想其实跟马里奥和植物大战僵尸一样，通过重重困难，当到达终点的时候，会看到成功的希望。

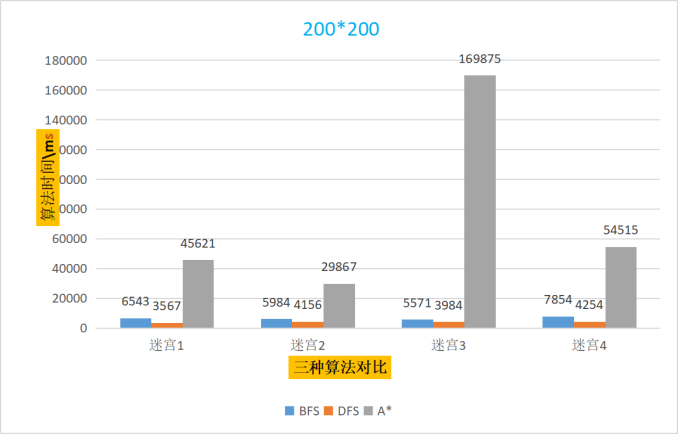
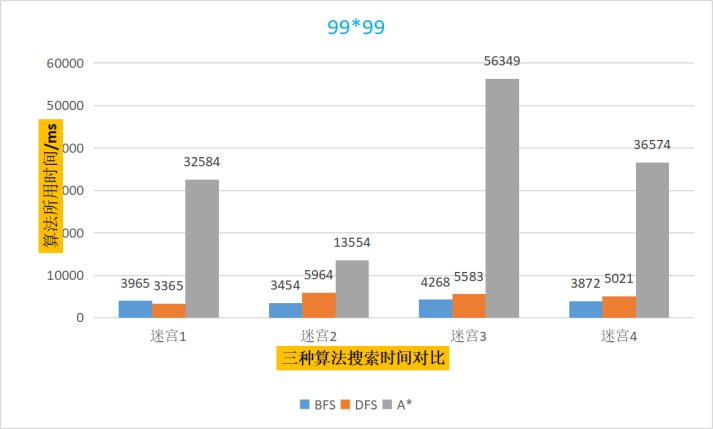
在界面中添加了到达终点时的起点方向标，杨桃，胜利图标，刷新地图等demo:



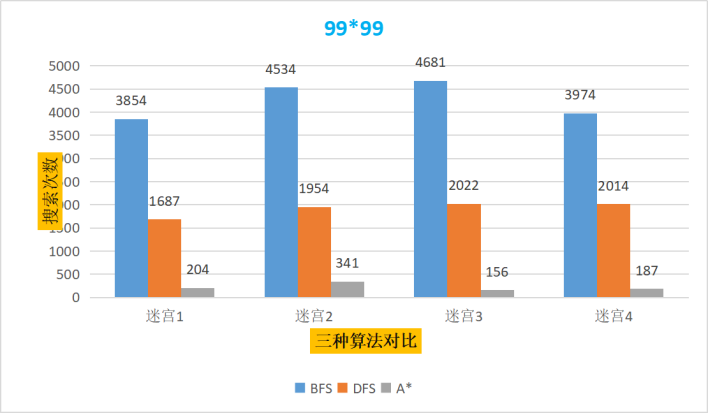
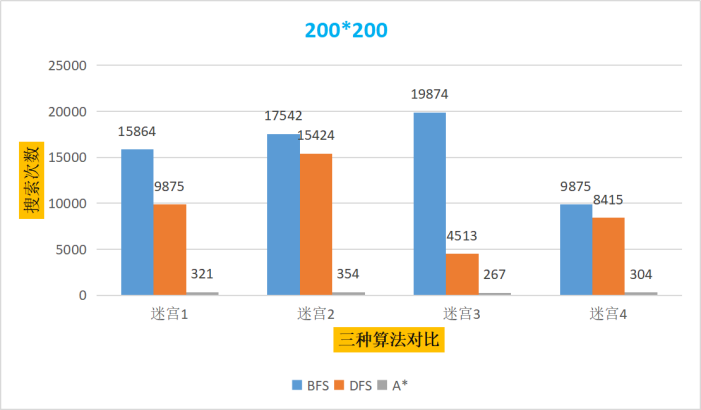
**图1-1 随机迷宫运行界面**

* 1. **实验结果对比**

在完成了算法和可视化之后，我们先使用Prim算法⽣成了多个不同规模的迷宫。然后主要对⽐了深度搜索、⼴度搜索和A\*算法的时间复杂度和搜索次数。并将结果绘制成柱形图进⾏说明。 下图为对⽐图总览，在第四章中有对表格的详细对⽐说明。



**图1-2实验结果对比 图1-3实验结果对比**



**图1-4实验结果对比 图1-5实验结果对比**

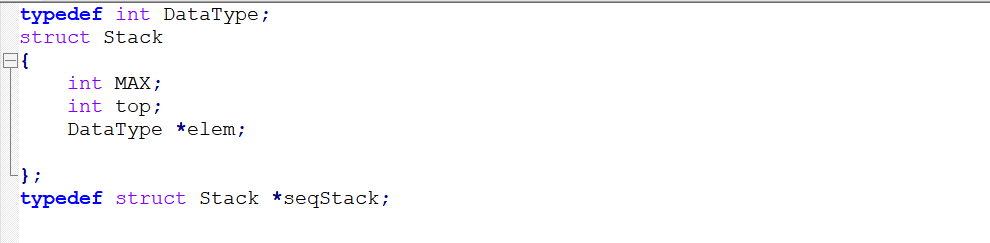
### 迷宫路线搜索算法

**2.1. 栈与队列**

**2.1.1. 顺序栈**

采用顺序存储的栈即为顺序栈，使用一组连续的存储单元来存放数据元素。栈的操作限定在栈顶进行，因此设定一个top变量指示栈顶位置，设置初始值为-1。因此，数据入栈时需要先将top指针加一，然后进栈。

给出顺序栈的类型定义：

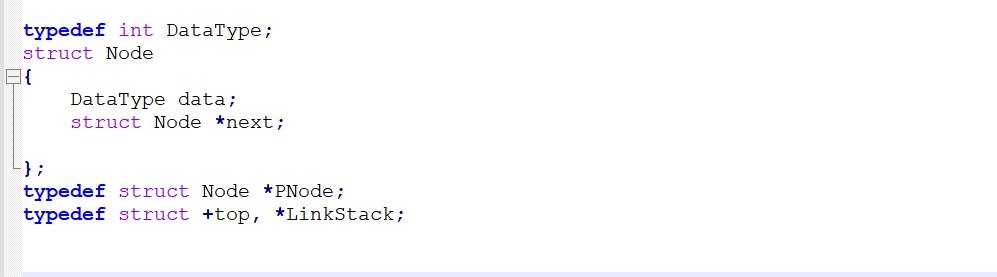


**图2-1顺序栈**

**2.1.2. 链栈**

采用链式存储的栈称为链栈，top指示栈顶元素，栈底在链表尾部。

给出链栈的定义：

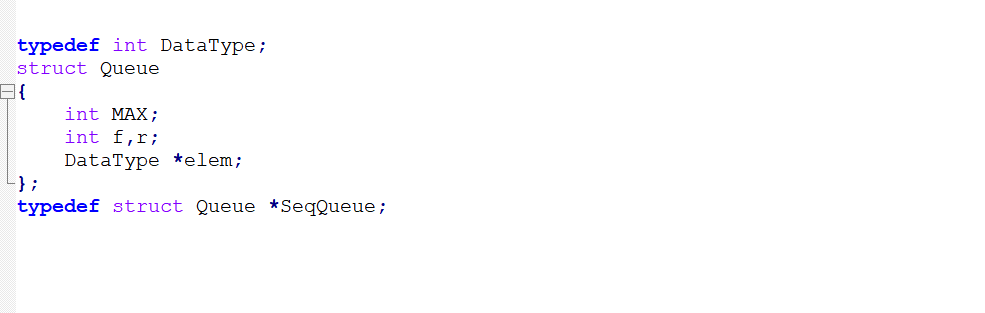


**图2-2链栈**

**2.1.3. 顺序队列**

顺序队列通常采用一维数组进行存储。其中，连续的存储单元依次存放队列中的元素。同时，使用两个指针分别表示数组中存放的第一个元素和最后一个元素的位置。其中，指向第一个元素的指针被称为队头指针front，指向最后一个元素的位置的指针被称为队尾指针rear。

给出顺序队列的定义：

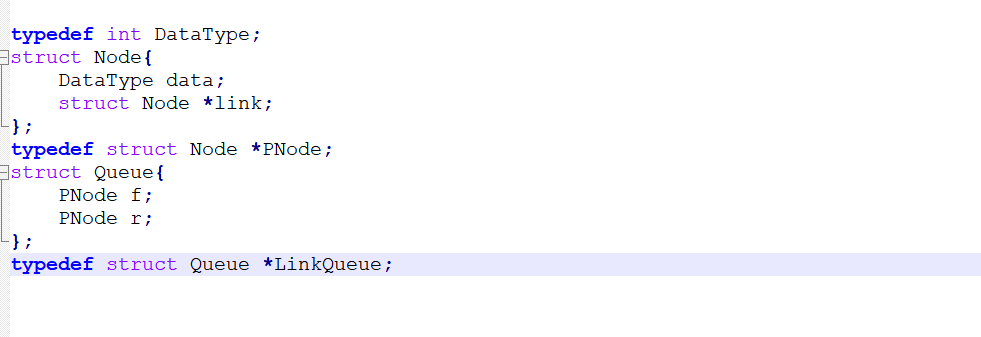


**图2-3顺序队列**

**2.1.4. 链队列**

采用链式存储结构的队列，队头指针指向链队列的第一个结点，队尾指针指向最后一个结点。

同样给出链队列的类型定义：



**图2-4链队列**

**2.2 图的遍历和图的存储结构**

图的遍历分为深度优先遍历和广度优先遍历，分别简称为DFS和BFS。图的遍历是从某一个顶点出发，访问其他顶点，但是不能重复访问（每个顶点只能访问一次）。因为图中的顶点之间可能存在多条路径，因此在访问的过程中需要根据情况结合设置不同的状态以免访问多次，直到所有的顶点都访问完毕。

图的存储结构常用的有两种，邻接矩阵表示法和邻接表表示法。

**2.2.1邻接矩阵表示法**

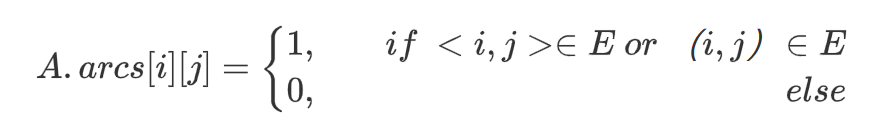
建立一个顶点表（记录各个顶点信息）和一个邻接矩阵（表示各个顶点之间关系）

设图A=(V，E)有 n 个顶点，则

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| I | 0 | 1 | 2 | …… | n-1 |
| Vexs[i] | V1 | V2 | V3 | …… | Vn |

**图2-5顶点表**

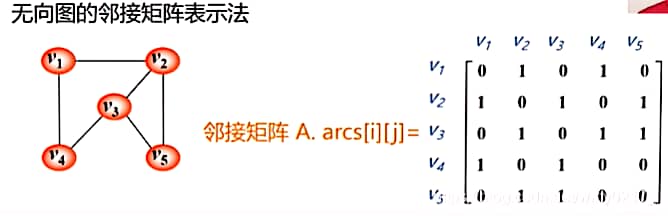
图的邻接矩阵是一个二维数组A.arcs[n][n]，定义如下:



IMG_256（1）

也就是说当对应的边是图的边时，记录二维数组为1，反之，记为0。

1. **无向图的邻接矩阵表示法**



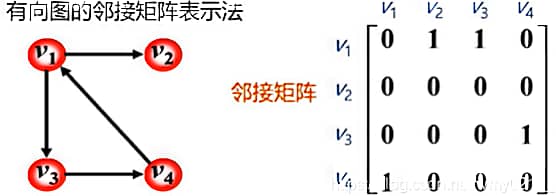
**图2-5无向图邻接矩阵**

分析1：无向图的邻接矩阵是对称的

分析2：顶点 i 的度=第 i 行(列)中 1 的个数

特点：完全图的邻接矩阵中，对角元素为0，其余为1

1. **有向图的邻接矩阵表示法**



**图2-6有向图邻接矩阵**

注：在有向图的邻接矩阵中

第i行含义：以节点vi为尾的弧（即出度边）

第i列含义：以节点vi为头的弧（即入度边）。

分析1：有向图的邻接矩阵可能是不对称的

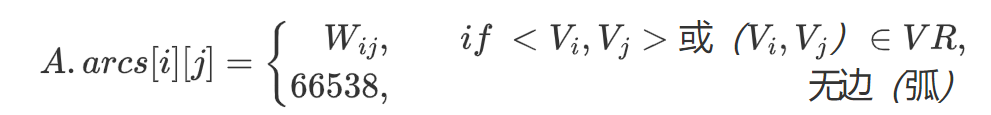
分析2：

顶点的出度=第 i 行元素之和

顶点的入度=第 i 列元素之和

顶点的度=第 i 行元素之和+第 i 列元素之和

1. **网（有权图）的邻接矩阵表示法：**

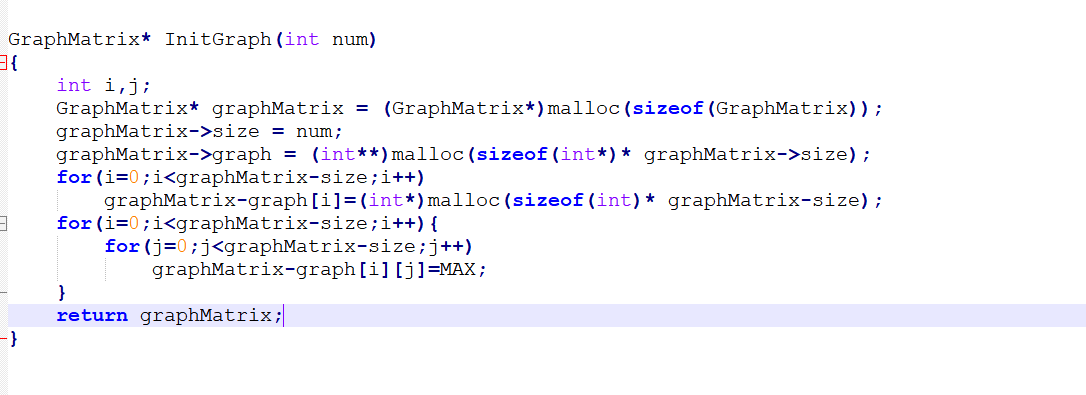
定义如下：

（2）



**图2-8带权图邻接矩阵**

1. 图的邻接矩阵初始化算法：



**图2-9邻接矩阵初始化**

**2.2.2.邻接表**

图的邻接表表示结合了顺序存储和链式存储的存储结构。把图中的n个顶点按照序号进行顺序存储，每个顶点Vi对应一个链表，这个链表代替邻接矩阵的每行，每个链表中存储和该顶点相邻的所有顶点。链表中的结点结构至少包括顶点域和指针域，而对于带权网络，在结点结构中增加权值域。

给出邻接表存储的结构类型定义：

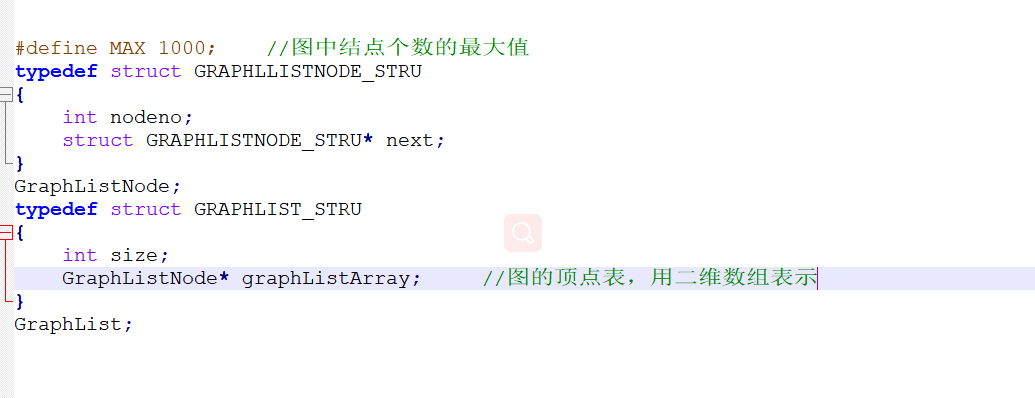
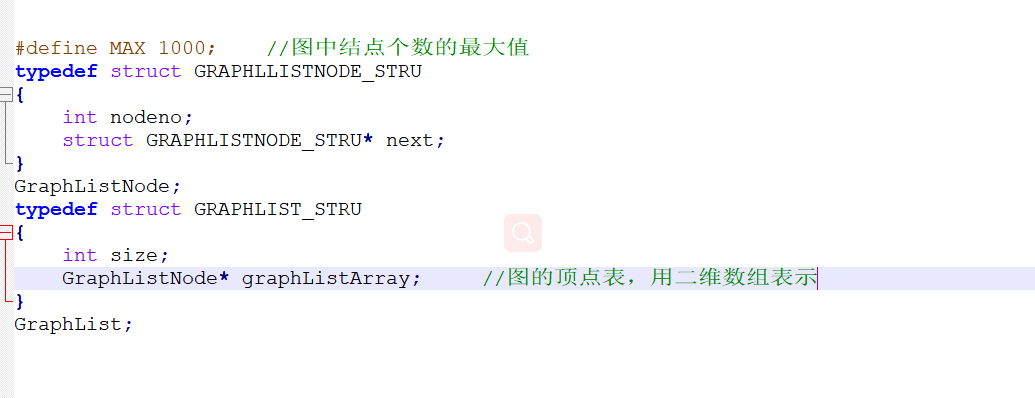


图2-10邻接表

图邻接表初始化算法：



**图2-11邻接表初始化**

**2.3 迷宫算法-****深度优先搜索策略**

**2.3.1. 简介**

DFS简介——DFS即深度优先搜索算法，属于图的遍历算法中的一种，英文缩写为DFS即Depth First Search.其搜索过程简要来说是对每一个可能的分支路径深入到不能再深入为止，而且每个节点都只会搜索一次。

它的具体思想就如同在家里找钥匙，无论从哪一间房间开始都可以，比如主卧室，然后从房间的一个角开始，将房间内的墙角、床头柜、床上、床下、衣柜里衣柜上、前面的电视柜等挨个寻找，做到不放过任何一个死角，所有的抽屉、储藏柜中全部都找遍，形象比喻就是翻个底朝天，然后再寻找下一间，直到找到为止。对于DFS的理解是：不撞南墙不回头（如果撞了，那么可能回头，可能不回头；如果回头，那么回头的过程叫做回溯）。

在具体的问题中，某些时候，当面对一些不知道该怎么去做的问题的时候（或者说找不到对应的数学模型去求解问题的时候），常常会使用深度优先搜索的方式去求解问题（实际上是一个不断试探，不断穷举的过程），深度优先搜索很适合解决这类问题，比如求解迷宫的路径（注意：这里说的是路径，不一定是指最短路径）问题，就是一个很经典的试探过程。

深度优先搜索的递归算法：

深度优先遍历其实就是一个递归的过程，如果再敏感一些，会发现其实如果将下图2-12转换成图2-13后，就是一棵树的前序遍历。

它从图中某个顶点v出发，访问此顶点，然后从v的未被访问的邻接点出发深度优先遍历图，直至图中所有和v有路径相通的顶点都被访问到。事实上，我们这里讲到的是连通图，对于非连通图，只需要对它的连通分量分别进行深度优先遍历，即在先前一个顶点进行一次深度优先遍历后，若图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访到为止。

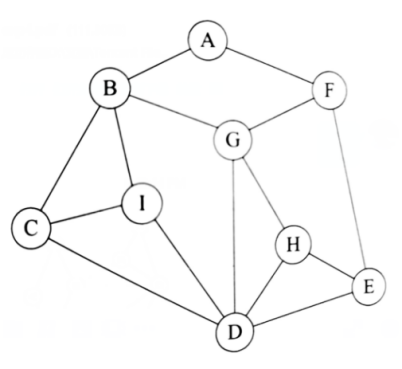
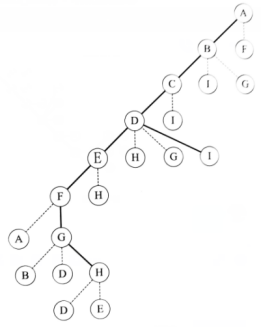
 

图2-12 图 图2-13 转化图

深度优先搜索是一种在开发爬虫早期使用较多的方法。它的目的是要达到被搜索结构的叶结点(即那些不包含任何[超链](https://baike.baidu.com/item/%E8%B6%85%E9%93%BE/3947966" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)的[HTML文件](https://baike.baidu.com/item/HTML%E6%96%87%E4%BB%B6" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)) 。在一个HTML文件中，当一个[超链](https://baike.baidu.com/item/%E8%B6%85%E9%93%BE/3947966" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)被选择后，被链接的HTML文件将执行深度优先搜索，即在搜索其余的超链结果之前必须先完整地搜索单独的一条链。深度优先搜索沿着HTML文件上的[超链](https://baike.baidu.com/item/%E8%B6%85%E9%93%BE/3947966" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)走到不能再深入为止，然后返回到某一个HTML文件，再继续选择该HTML文件中的其他超链。当不再有其他[超链](https://baike.baidu.com/item/%E8%B6%85%E9%93%BE/3947966" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)可选择时，说明搜索已经结束。

**2.3.2. 思想**

1.深度优先遍历图的方法是，从图中某顶点v出发：

（1）访问顶点v；

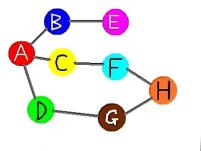
（2）依次从v的未被访问的邻接点出发，对图进行[深度优先遍历](https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E9%81%8D%E5%8E%86" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)；直至图中和v有路径 相通的顶点都被访问；

（3）若此时图中尚有顶点未被访问，则从一个未被访问的顶点出发，重新进行[深度优先遍历](https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E9%81%8D%E5%8E%86" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)，直到图中所有顶点均被访问过为止。

涉及到的一个概念：深度优先搜索树

深度优先搜索树时经过的边和所有顶点构成深度优先搜索树，如果一个图是非连通图，经过DFS组成一个深林，深林中树的个数就是图中连通分支的个数。

2.下图图2.3.2.1是一个[无向图](https://baike.baidu.com/item/%E6%97%A0%E5%90%91%E5%9B%BE" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)，如果我们从A点发起深度优先搜索（以下的访问次序并不是唯一的，第二个点既可以是B也可以是C,D），则我们可能得到如下的一个访问过程：A->B->E（没有路了！[回溯](https://baike.baidu.com/item/%E5%9B%9E%E6%BA%AF" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)到A)->C->F->H->G->D（没有路，最终[回溯](https://baike.baidu.com/item/%E5%9B%9E%E6%BA%AF" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)到A,A也没有未访问的相邻[节点](https://baike.baidu.com/item/%E8%8A%82%E7%82%B9" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)，本次搜索结束）。



**图2-13 无向图**

每次深度优先搜索的结果必然是图的一个连通分量.深度优先搜索可以从多点发起.如果将每个节点在深度优先搜索过程中的"结束时间"排序（具体做法是创建一个list，然后在每个节点的相邻节点都已被访问的情况下，将该节点加入list结尾，然后逆转整个链表)，则我们可以得到所谓的"[拓扑排序](https://baike.baidu.com/item/%E6%8B%93%E6%89%91%E6%8E%92%E5%BA%8F" \t "https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/_blank)"。

**2.3.3. 特点**

1)无论问题的内容和性质以及求解要求如何不同，它们的程序结构都是相同的，不相同的仅仅是存储结点数据结构和产生规则以及输出要求。

2)深度优先搜索法有递归以及非递归两种设计方法。一般的，当搜索深度较小、问题递归方式比较明显时，用递归方法设计好，它可以使得程序结构更简捷易懂。当搜索深度较大时，当数据量较大时，由于系统堆栈容量的限制，递归容易产生溢出，用非递归方法设计比较好。

3)深度优先搜索方法有广义和狭义两种理解。广义的理解是，只要最新产生的结点（即深度最大的结点）先进行扩展的方法，就称为深度优先搜索方法。在这种理解情况下，深度优先搜索算法有全部保留和不全部保留产生的结点的两种情况。而狭义的理解是，仅仅只保留全部产生结点的算法。不保留全部结点的算法属于一般的回溯算法范畴。保留全部结点的算法，实际上是在数据库中产生一个结点之间的搜索树，因此也属于图搜索算法的范畴。

4)不保留全部结点的深度优先搜索法，由于把扩展望的结点从数据库中弹出删除，这样，一般在数据库中存储的结点数就是深度值，因此它占用的空间较少，所以，当搜索树的结点较多，用其他方法易产生内存溢出时，深度优先搜索不失为一种有效的算法。

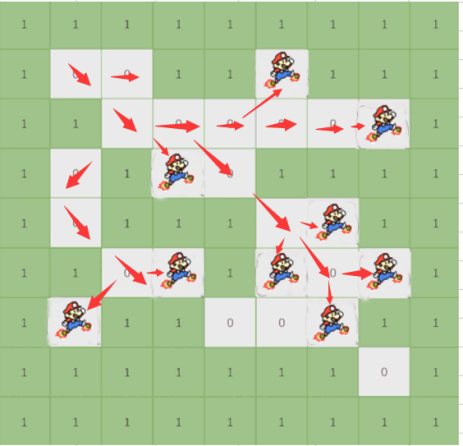
5)从输出结果可看出，深度优先搜索找到的第一个解并不一定是最优解。

如果要求出最优解的话，一种是采用动态规划法，另一种方法是修改原算法：把原输出过程的地方改为记录过程，即记录达到当前目标的路径和相应的路程值，并与前面已记录的值进行比较，保留其中最优的，等全部搜索完成后，才把保留的最优解输出。

**2.4 迷宫算法-广度优先搜索策略**

**2.4.1. 简介**

BFS，其英文全称是Breadth First Search。 BFS并不使用经验法则算法。从算法的观点，所有因为展开节点而得到的子节点都会被加进一个先进先出的[队列](https://baike.baidu.com/item/%E9%98%9F%E5%88%97" \t "https://baike.baidu.com/item/_blank)中。一般的实验里，其邻居节点尚未被检验过的节点会被放置在一个被称为 open 的容器中（例如[队列](https://baike.baidu.com/item/%E9%98%9F%E5%88%97" \t "https://baike.baidu.com/item/_blank)或是[链表](https://baike.baidu.com/item/%E9%93%BE%E8%A1%A8" \t "https://baike.baidu.com/item/_blank)），而被检验过的节点则被放置在被称为 closed 的容器中。



**图 2-14 广度优先搜索策略**

**2.4.2. 思想**

已知图G=(V,E)和一个源顶点s，宽度优先搜索以一种系统的方式探寻G的边，从而“发现”s所能到达的所有顶点，并计算s到所有这些顶点的距离(最少边数)，该算法同时能生成一棵根为s且包括所有可达顶点的宽度优先树。对从s可达的任意顶点v，宽度优先树中从s到v的路径对应于图G中从s到v的[最短路径](https://baike.baidu.com/item/%E6%9C%80%E7%9F%AD%E8%B7%AF%E5%BE%84" \t "https://baike.baidu.com/item/_blank)，即包含最小边数的路径。之所以称之为宽度优先算法，是因为算法自始至终一直通过已找到和未找到顶点之间的边界向外扩展，就是说，算法首先搜索和s距离为k的所有顶点，然后再去搜索和S距离为k+l的其他顶点。

**2.4.3. 特点**

1)在产生新的子结点时，深度越小的结点越先得到扩展，即先产生它的子结点。为使算法便于实现，存放结点的数据库一般用队列的结构。

2)无论问题性质如何不同，利用广度优先搜索法解题的基本算法是相同的，但数据库中每一结点内容，产生式规则，根据不同的问题，有不同的内容和结构，就是同一问题也可以有不同的表示方法。

3)当结点到跟结点的费用（有的书称为耗散值）和结点的深度成正比时，特别是当每一结点到根结点的费用等于深度时，用广度优先法得到的解是最优解，但如果不成正比，则得到的解不一定是最优解。这一类问题要求出最优解，一种方法是使用后面要介绍的其他方法求解，另外一种方法是改进前面深度（或广度）优先搜索算法：找到一个目标后，不是立即退出，而是记录下目标结点的路径和费用，如果有多个目标结点，就加以比较，留下较优的结点。把所有可能的路径都搜索完后，才输出记录的最优路径。

4)广度优先搜索算法，一般需要存储产生的所有结点，占的存储空间要比深度优先大得多，因此程序设计中，必须考虑溢出和节省内存空间得问题。

5)比较深度优先和广度优先两种搜索法，广度优先搜索法一般无回溯操作，即入栈和出栈的操作，所以运行速度比深度优先搜索算法法要快些。

总之，一般情况下，深度优先搜索法占内存少但速度较慢，广度优先搜索算法占内存多但速度较快，在距离和深度成正比的情况下能较快地求出最优解。因此在选择用哪种算法时，要综合考虑。决定取舍。

**2.4.4. 广度优先搜索与深度优先搜索的对比**

DFS：

1)能找出所有解决方案

2)优先搜索一棵子树，然后是另一棵，所以和广搜对比，有着内存需要相对较少的优点

3)内存消耗小

BFS：

1)遍历次数少，搜索所有可能路径。

2)解决最短或最少问题特别有效

3)消耗内存大

·实现方式：DFS优先遍历用栈，BFS优先遍历用队列

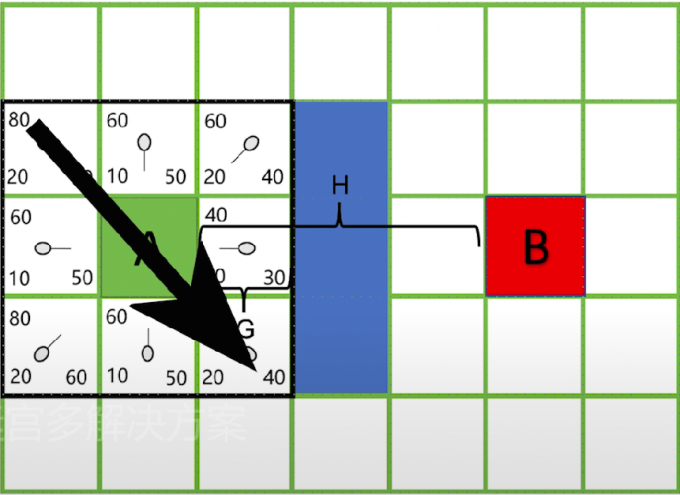
·与树的遍历关系：深度优先遍历对应树的先序、中序和后序三种遍历， 广度优先遍历 对就树的层次遍历。

· 共同点：在深度大的情况下效率较低。

**2.5 迷宫算法-****A\*算法**

**2.5.1. 简介**

A\*算法，A\*（A-Star)算法是一种静态路网中求解[最短路径](https://baike.baidu.com/item/%E6%9C%80%E7%9F%AD%E8%B7%AF%E5%BE%84/6334920" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)最有效的直接搜索方法，也是解决许多搜索问题的[有效](https://baike.baidu.com/item/%E6%9C%89%E6%95%88" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)算法。算法中的[距离](https://baike.baidu.com/item/%E8%B7%9D%E7%A6%BB/16370594" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)估算值与实际值越接近，最终[搜索](https://baike.baidu.com/item/%E6%90%9C%E7%B4%A2/1806" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)速度越快。



**图 2-15 A\*算法**

**2.5.2. A\*算法的实现思路**

A\*（A-Star)算法是一种静态路网中求解最短路径最有效的直接[搜索](https://baike.baidu.com/item/%E6%90%9C%E7%B4%A2/5395421" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)方法，也是许多其他[问题](https://baike.baidu.com/item/%E9%97%AE%E9%A2%98/13018756" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)的常用启发式算法。注意——是最有效的直接搜索算法，之后涌现了很多预处理算法（如ALT，CH，HL等等），在线查询效率是A\*算法的数千甚至上万倍，公式表示为：

（3）

其中， f(n) 是从初始[状态](https://baike.baidu.com/item/%E7%8A%B6%E6%80%81/5809342" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)经由状态n到目标状态的最小代价估计，g(n) 是在[状态空间](https://baike.baidu.com/item/%E7%8A%B6%E6%80%81%E7%A9%BA%E9%97%B4" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)中从初始状态到[状态](https://baike.baidu.com/item/%E7%8A%B6%E6%80%81/5809342" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)n的最小代价，h(n) 是从状态n到目标状态的路径的最小估计代价。

对于路径搜索问题，状态就是图中的节点，代价就是距离。

真实h(n)的选取：保证找到[最短路径](https://baike.baidu.com/item/%E6%9C%80%E7%9F%AD%E8%B7%AF%E5%BE%84" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)（最优解的）条件，关键在于估价函数f(n)的选取（或者说h(n)的选取）。

以h(n)表达状态n到目标状态估计的距离，那么h(n)的选取大致有如下三种情况：

1. 如果h(n)< h(n)，这种[情况](https://baike.baidu.com/item/%E6%83%85%E5%86%B5" \t "https://baike.baidu.com/item/A%2A%E7%AE%97%E6%B3%95/_blank)下，搜索的点数多，搜索范围大，效率低。但能得到最优解。
2. 如果h(n)=h(n)，此时的搜索效率是最高的。
3. 如果 h(n)>h(n)，搜索的点数少，搜索范围小，效率高，但不能保证得到最优解。

**2.6. 迷宫算法-Dijkstra算法**

2.6.1.简介

Dijkstra算法是从一个顶点到其余各顶点的最短路径算法，解决的是有权图中最短路径问题。Dijkstra算法主要特点是从起始点开始，采用贪心算法的策略，每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止。

采用Dijkstra完成了迷宫变种。

## 2.6.2. 思想

Dijkstra算法采用的是一种贪心的策略，声明一个数组dis来保存源点到各个顶点的最短距离和一个保存已经找到了最短路径的顶点的集合：T，初始时，原点 s 的路径权重被赋为 0 （dis[s] = 0）。若对于顶点 s 存在能直接到达的边（s,m），则把dis[m]设为w（s, m）,同时把所有其他（s不能直接到达的）顶点的路径长度设为无穷大。初始时，集合T只有顶点s。然后，从dis数组选择最小值，则该值就是源点s到该值对应的顶点的最短路径，并且把该点加入到T中，OK，此时完成一个顶点，然后，我们需要看看新加入的顶点是否可以到达其他顶点并且看看通过该顶点到达其他点的路径长度是否比源点直接到达短，如果是，那么就替换这些顶点在dis中的值。然后，又从dis中找出最小值，重复上述动作，直到T中包含了图的所有顶点。

### 随机迷宫生成算法

随机迷宫生成有三个算法：深度优先算法、prim算法和递归分割算法，下面逐个进行介绍。

**3.1 Prime 迷宫生成算法**

Prime迷宫生成算法的原理：

1）初始地图所有位置均设为墙

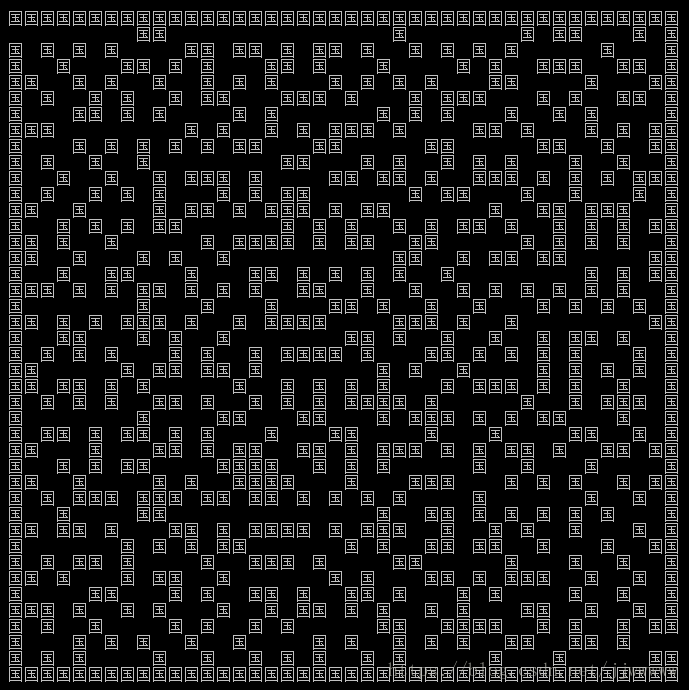
2）任意插入一个墙体进墙队列

3）判断此时墙体是否可以设置为路（判断依据在于上下左右四个位置是否只有一个位置是路）

4）若设置为路，则将该位置周围（上下左右）的所有墙插入队列，接着执行5），若无法设置为路，直接执行5）

5）从墙队列中删去当前位置所在节点

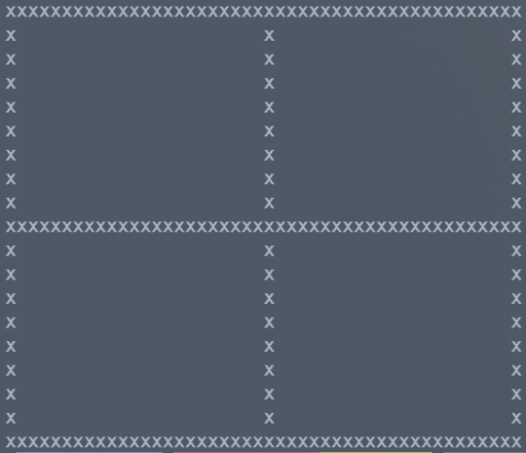
5)若墙队列不为空，则从队列中随机选取一面墙重新执行3），直到墙队列为空。



# 图3-1 Prime算法迷宫生成

# 3.2 递归分割算法

原理很简单，首先假设迷宫全是路，在里面画四面墙，把迷宫分割成四个新区域，如下：



**图3-2 递归算法迷宫生成**

随机选择三面墙打通，这时原本隔开的四个区域又被打通了。以此类推，在四个新区域内继续设墙分割区域，然后把墙打通，直到不能继续分割才结束。



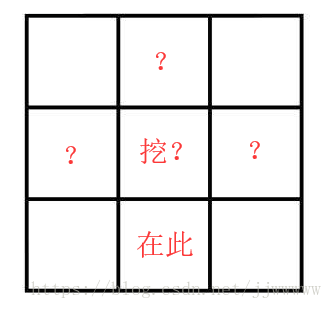
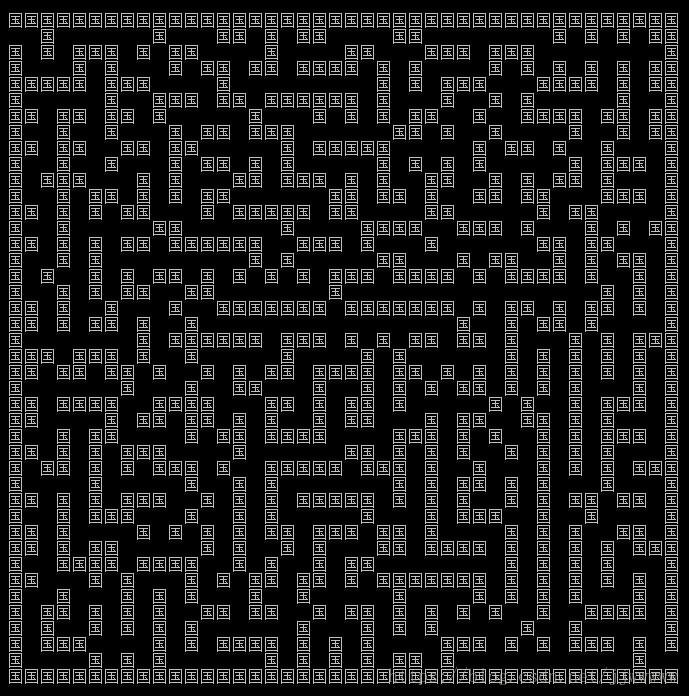
**图3-3 递归算法迷宫生成**

3.3 **深度优先算法**

这个时候请把自己想象成一只地鼠，要在这个区域不停的挖，直到任何一块区域再挖就会挖穿了为止。

我们挖的道路就像树结构，树上有很多的分支，分支也有子分支，每个子分支都不能相交，相交了就说明墙被挖穿了，那么此时的迷宫就可能存在多条正确道路，这不是我想看到的。

那么基于唯一道路的原则，我们向某个方向挖一块新的区域时，要先判断新区域是否有挖穿的可能，如果可能挖穿要立即停止并换个方向再挖。

**图3-4 深度优先算法 图3-5深度优先算法**

**3.4 总结对比**

不同的生成策略会导致生成迷宫形状的不同。

递归算法由于基于四墙通三面的核心思想，故生成的迷宫中多会众多连续的迷宫墙。

对于深度优先来说，是朝向一个区域挖掘，当可能出现错误时，才会改变改变方向。而Prime算法则是通过判断某个点的四周是否可以行走来判断是否挖通。这里的深度优先与Prime算法就类似于DFS，BFS。一个是执着于一路深挖，故会生成小区域的墙。另一个是四面尝试，所以产生的墙体较小，看上去复杂。

### 实验结果

**4.1. 算法搜索时间对比**

**4.1.1. 总体柱状图对比**

1.随机生成99\*99的，迷宫

**图4-6算法搜索时间对比**

2.随机生成200\*200的迷宫

**图4-7算法搜索时间对比**

**4.1.2. 算法对比结果说明**

**1.** BFS能够求得最短路径，因为BFS每进行一次相当于当前的路径长度。对于一个N\*N矩阵，BFS最多运行n\*n次。BFS使用一个队列来保存已经被标记但是它的邻接表还没有被访问的顶点。先把起点加入队列，然后重复以下操作直到队列为空

**2.**深度优先搜索相当于一个人在走迷宫，广搜相当于是无穷人沿着不同方向走（因为每条路都同时有人走）。DFS相当于是一个下压栈。是先进后出的原则（如果找不到，往回走，直到回到有路的地方）（DFS隐式使用了栈）

**3.**A\*算法的魅力之处在于它不仅能找到地图中从A到B的一条路径，还能保证找到的是一条最短路径，相对稳定，逐一有序的进行，但是耗时长。

1）在⼩规模迷宫下，深度优先搜索的搜索速度要慢于⼴度优先搜索算法。

2）在⼤规模迷宫下，深度优先搜索的搜索速度要快于⼴度优先搜索，因为⼤规模⽹络下⼴度优先搜索需要进⾏更多次的搜索。

3）A\*算法在某些迷宫下表现优良，但是在某些迷宫下花费时间会⾮常⻓，这是由于A\*算法如果遇到需要”回头“的情况，需要花费较⻓的时间去处理。

**4.2. 算法搜索次数对比**

**4.2.1. 总体柱状图对比**

1.随机99\*99的迷宫

**图4-8算法搜索次数对比**

2.随机200\*200的迷宫

**图4-9算法搜索次数对比**

**4.2.2. 算法对比结果说明**

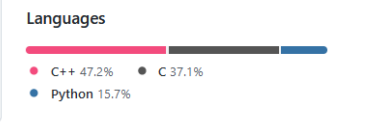
1.用BFS算法走迷宫 搜索的次数一般都比 DFS算法的次数大，要重复多次的访问个方向的结点。

2. A\*算法由于没次搜索的最优路径，虽然耗时长，但是寻找的次数效率高，因此A\*算法搜索的次数一般都比BFS和DFS的次数少**。**

### 项目代码介绍

**5.1 项目总体结构**

本次项目使用C，C++和python三种语言实现。由于C与C++在底层上实现快，运行效率高，而python开发效率高，因此这两者可以相辅相成，不管是在Python项目中嵌入C++代码，或是在C++项目中用Python实现外围功能。



**图5-1项目总体结构**

本项目采用C与C++实现走迷宫，随机迷宫的生成以及迷宫测试。可视化过程通过python中的pygame实现。最后python的走迷宫采用C++生成的so文件实现。

The file tree structure show as follows：



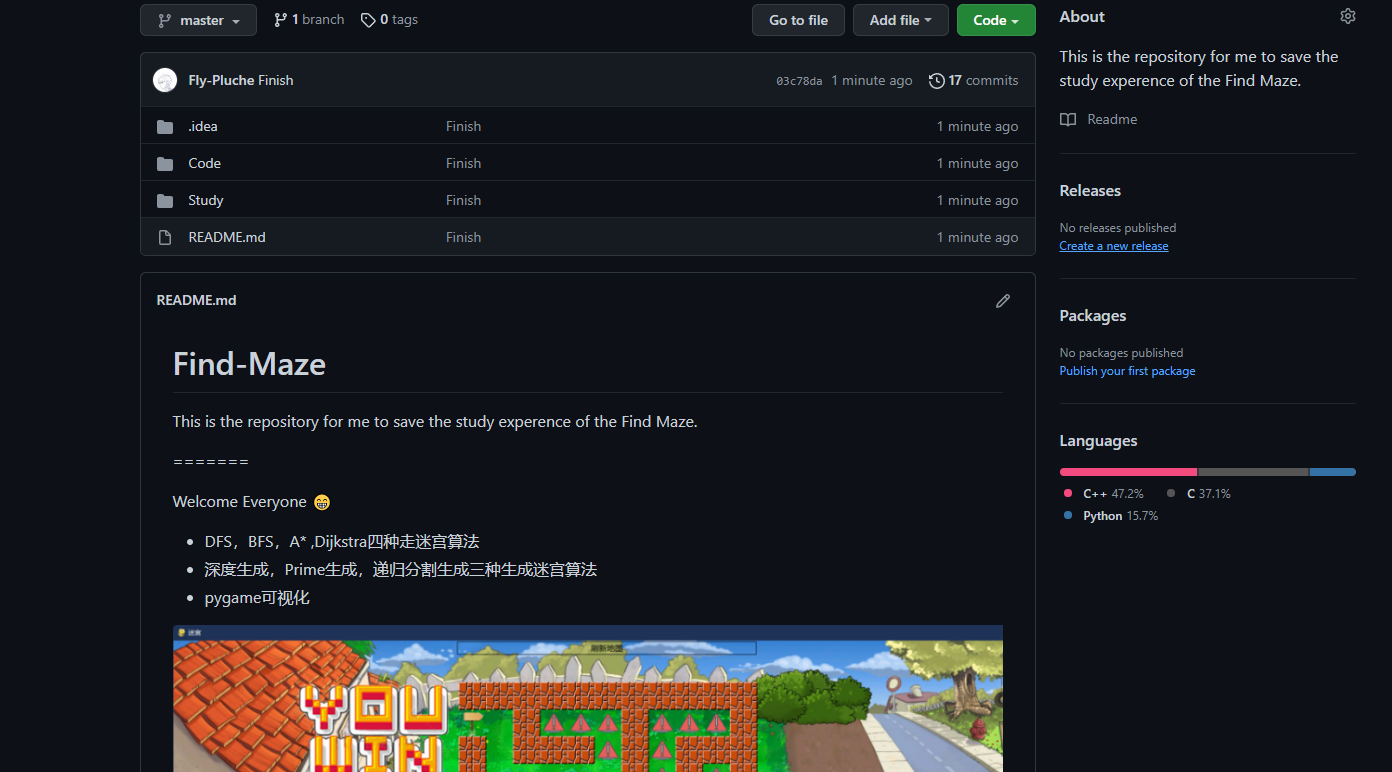
**图5-2项目总体结构**

**5.2 总体情况**

代码量较大，不便在实验报告中全部贴出展示。整个项目过程通过GitHub实现管理，用于存放相关的学习资料以及实现代码。

Since the repository was created，has pushed 17 commits to master and 17 commits to all branches. On master, 51 files have changed and there have been 2,249

additions and 617 deletions.



**图5.2.1总体情况**

### 总结

本次实验，我们通过随机生成迷宫，探索了基于栈、队列、广度优先和深度优先算法，红黑树的A\*算法完成的迷宫搜寻策略，最终得出了最终得出BFS搜索次数最多，A\*算法寻找效率高的对比结论，在以后走迷宫的过程可以根据迷宫的样式采用不用的策略。

### 参考文献

[1]张瑞霞,张敬伟.数据结构与算法[M].北京:清华大学出版社,2018.

[2]邻接矩阵. https:/blog.csdn.net/wmy0217\_/article/details/104336767#1\_7,2020-02-15/2021-1 2-01

[3]程杰.大话数据结构[M].北京:清华大学出版社,2011.

[4]广度搜索策略.https://www.cnblogs.com/cs-whut/p/11147348.html，2019-07-07/2021-12-01

[5]BFS和DFS的差别.https:/blog.csdn.net/qq\_43680117/article/details/84033882,2018-11-13 /2021-12-01

[6]DFS(深度优先搜索）和BFS(广度优先搜索）求迷官路径问题的总结.https:/blog.cs dn.net(qg\_36737934/ar ticleldetails/88918855,2019-03-31/2021-12-01

[7]深度优先搜索和广度优先搜索及典例分析. https: blog.csdn.netqq41681241/article/details/8 1432634,201 9-06-01/2021-12-01

[8]A\*算法. https:/blog.csdn.net/cail3160674275/article/details/52972609，2019-06-01/2021-12- 01