成　绩	
评卷人	

研究生	李雪
学　号	2015363039

武漢紡織大學

研 究 生 课 程 论 文

论文题目	基本的搜索技术
完成时间	2020 年 10 月 17 日
课程名称	人工智能
专　　业	电子信息
年　　级	2020 级

武汉纺织大学研究生处制

一、 引言

搜索引擎的发生背景在因特网发展初期，网站相对较少，新闻查找比较容易。然而随着新闻技术的飞速发展，特别是因特网应用的迅速普及，网站越来越多，并且每天全球互联网网页数目以千万级的数量增加。要在浩瀚的网络新闻中寻找所需要的材料无异于大海捞针，这时具有完备性、最优性的基本搜索技术应运而生。

搜索引擎的工作机制就是采用高效的蜘蛛程序，从指定 URL 开始顺着网页上的超链接，采用宽度优先算法或深度优先算法对整个 Internet 进行遍历，将网页信息抓取到本地数据库。然后使用索引器对数据库中的重要信息单元，如标题，关键字及摘要等或者全文进行索引，以供查询导航。最后，检索器将用户通过浏览器提交的查询请求与索引数据库中的信息以某种检索技术进行匹配，再将检索结果按某种排序方法返回给用户。

本文主要研究宽度优先搜索（BFS）和深度优先搜索（DFS）两种基本的搜索技术，二者的优缺点各不相同，基于这两种方法的实现原理编程者可以根据不同的需要改进至更优化的算法，更高效的解决搜索问题。

二、 相关工作

研究宽度优先搜索和深度优先搜索的工作流程以及相关的算法思想为算法实现做准备工作。

宽度优先搜索算法（又称广度优先搜索）是最简便的图的搜索算法之一，这一算法也是很多重要的图的算法的原型。Dijkstra 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 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，直到找到结果为止，引入优先级队列，可将 BFS 算法改进为一致代价搜索算法（UCS）。因此它的优点在于解决最短或最少问题特别有效，每个结点只访问一遍，结点总是以最短路径被访问，而且寻找深度小，但 BFS 算法内存耗费量较大。

深度优先搜索（DFS）是一种在开发爬虫早期使用较多的方法，在解决无限状态空间中搜索时存在无穷路径的问题时，可使用深度受限搜索算法（DLS）。此外，迭代使用 DFS 算法搜索策略，衍生出迭代加深的 DFS 算法

用来确定目标节点所处的最浅深度 d。DFS 算法的目的是要达到被搜索结构的叶结点，即永远沿着一条分支走到尽头，直至无路可走，再回溯到上一个访问节点，继续搜索，直至找到结果。相对于 BFS 算法而言，DFS 算法能详细找出所有解决方案，优先搜索一棵子树，然后是另一棵，所以和宽搜对比，有着内存需要相对较少的优点，但在深度很大的情况下效率不高，需要多次遍历，搜索所有可能路径，同一个节点会访问多次。

概括来说，BFS 一次访问多条路，DFS 一次访问一条路，所以 BFS 算法的运行内存较大。在数据量较大的情况下，DFS 算法栈结构容易溢出，而 BFS 通过控制队列可以很好解决“爆队列”风险。它们两者间各自的优势需要通过实际的问题来具体分析，根据它们各自的特点来应用于不同的问题中才能获得最优的性能。

三、 算法描述（伪代码）

3.1 宽度优先搜索

宽度优先搜索（BFS）采用分支限界法，节点访问是先进先出原则，故可使用队列（queue）来实现，整个过程可以看做一个倒立的树形：

①把根节点放到队列的末尾。

②每次从队列的头部取出一个元素，查看这个元素所有的下一级元素，把它们放到队列的末尾。并把这个元素记为它下一级元素的前驱。

③找到所要找的元素时结束程序。

④如果遍历整个树还没有找到，结束程序。

宽度优先搜索的伪代码如下：

```
Input:
    A graph G and a vertex v of G
    came_from is a DICT, initialized with {}
Output:
    return: GOAL or None
    came_from is changed
def BFS(G, v, came_from):
    frontier = Queue()
```

```
    frontier.enqueue(v)
    came_from[v] = None
    while not frontier.is_empty():
        v = frontier.dequeue()
        if v is not labeled as discovered:
            if v is a goal:
                return v
            else:
                label v as discovered
                for all edges from v to w in G.adjacentEdges(v):
                    if w is not labeled as discovered:
                        frontier.enqueue(w)
                        came_from[w] = v
    return None
```

3.2 深度优先搜索

深度优先搜索（DFS）采用回溯法，节点访问是后进先出原则，故采用栈（stack）来实现，整个过程也可以想象成一个倒立的树形：

①把根节点压入栈中。

②每次从栈中弹出一个元素，搜索所有在它下一级的元素，把这些元素压入栈中。并把这个元素记为它下一级元素的前驱。

③找到所要找的元素时结束程序。

④如果遍历整个树还没有找到，结束程序。

深度优先搜索的伪代码如下：

```
Input:
    A graph G and a vertex v of G
    came_from is a DICT, initialized with {}
Output:
    return: GOAL or None
    came_from is changed
def DFS(G, v, came_from):
```

```
frontier = Stack()
frontier.push(v)
came_from[v] = None
while not frontier.is_empty():
    v = frontier.pop()
    if v is not labeled as discovered:
        if v is a goal:
            return v
        else:
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v):
                frontier.push(w)
                came_from[w] = v
return None
```

四、 算法验证（结果与分析）

4.1 宽度优先搜索

宽度优先搜索（迷宫案例）的一个测试结果如图 4.1 所示：

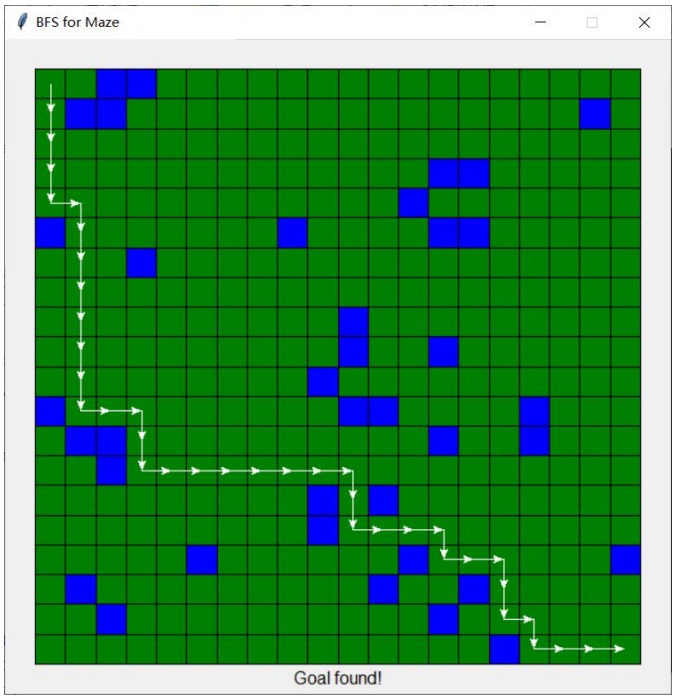

图 4.1 宽度优先搜索（迷宫）的一个结果

分析：依据宽度优先搜索原则，左上角为迷宫入口，右下角为迷宫出口。从当前位置出发搜索所有可达的状态，宽度优先搜索总是先搜索离初始状态近的状态。也就是说，它是按照开始状态--->只需 1 次转移就可以到达的所有状态--->只需 2 次转移就可以到达的所有状态--->.....，以这样的顺序开始搜索，对于同一个状态，宽度优先搜索只经过一次，因此它找出的是最短路径。这种方式是从根节点出发，一层一层的往外遍历，当发现某一层上有终点节点时，遍历结束，此时找到的也一定是最短路径，它和二叉树的层序遍历有异曲同工之妙。

4.2 深度优先搜索

深度优先搜索（迷宫）的一个测试结果如图 4.2 所示：

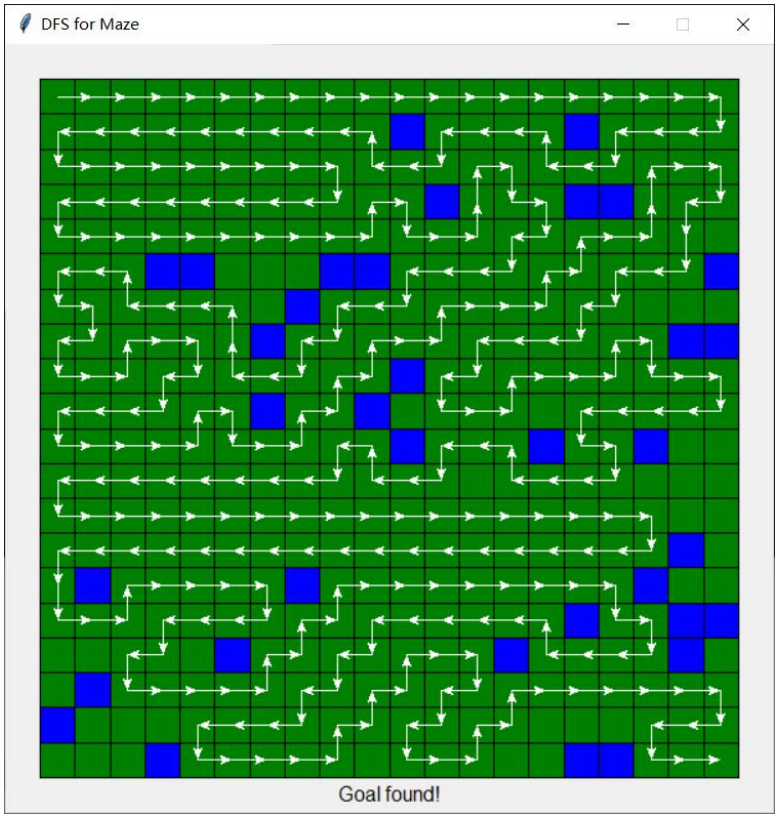

图 4.2 深度优先搜索（迷宫）的一个结果

分析：依据深度优先算法原则，以当前位置为起点，沿着一条路向前走。当遇到一个岔路口时，就选择一个岔路前进，如果这个岔路是死胡同，就退回上一个岔路口；如果这个岔路又有新的岔路口，就依据之前的做法，再选择一条岔路口继续前进。这种不撞南墙不回头的策略，形象地从深度上彻底挖掘了这个迷宫，因此这种搜索路径的方法被称作深度优先搜索。

五、 算法应用

5.1 宽度优先搜索应用举例

宽度优先搜索应用在解决图的最短路径问题和计算网络跳数方面。因为图结构在解决许多网络相关的问题时直到了重要的作用，比如，用来确定在互联网中一个网络到其他网络的网关的最佳路径。一种建模方法是采用无向图，其中顶点表示网络结点，边代表结点之间的联接。使用这种模型，可以采用宽度优先搜索来帮助确定结点间的最小跳数。此外，BFS 加上评估函数可以变为 A*算法，提高搜索效率。

5.2 深度优先搜索的应用举例

深度优先搜索应用在解决连通性问题和拓扑排序方面。有时候，我们必须根据各种事物间的依赖关系来确定一种可接受的执行顺序。比如，在大学里必须满足一些先决条件才能选的课程，或者一个复杂的项目，其中某个特定的阶段必须在其他阶段开始之前完成。要为这一类问题建模，可以采用优先级图，其采用的是有向图的思路。在优先级图中，顶点代表任务，而边代表任务之间的依赖关系。以必须先完成的任务为起点，以依赖于此任务的其他任务为终点，画一条边即可。此外 DFS 加上评估函数可以变为 IDA*算法。

六、 结论与展望

基本的搜索技术已在计算机网络跳数、拓扑排序上有很强的应用性，采用不同的数据结构可以实现宽度搜索和广度搜索。宽度搜索的遍历节点为先进先出原则故采用队列结构，深度搜索遍历节点为后进先出原则故采用栈存储结构，若采用 OPEN-CLOSED 表的实现方式，BFS 算法与 DFS 算法的唯一差别在于子节点添加到 OPEN 表的位置不同，BFS 将子节点添加到 OPEN 表的末尾，而 DFS 将子节点添加到 OPEN 表的开头。正是由于这个差异，才使得子节点被访问的顺序不同，从而衍生出两种不同的基本搜索技术。

随着互联网时代，web 信息网站的飞速发展，搜索技术在检索，查找等方面起着至关重要的作用。在信息大爆炸时代，高效准确的获取查找结果是当前算法永远追求的目标。宽度优先搜索和深度优先搜索均为盲目搜索算法，基于此算法，我们可以进行不断改进，提高算法的搜索效率。

七、 参考文献

[1] 严蔚敏,吴伟民 著.数据结构(C 语言版）:清华大学出版社,2012-05-01

[2] Thomas H.Cormen、Charles E.Leiserson. 算法导论：机械工业出版社出版，2013

[3] 杜小勤。《人工智能》课程系列，Part I: Python 程序设计基础，2018/06/13。

[4] 杜小勤。《人工智能》课程系列,Part II: Python 算法基础,2018/07/31。

[5] 杜小勤。《人工智能》课程系列,Chapter 3:基本的搜索技术,2018/08/20。

[6] 杜小勤。《人工智能》课程系列，Maze 实验平台的设计与实现，2018-09/25。