成　绩	
评卷人	

研究生	冯朗
学　号	2015363065

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. 10. 17
课程名称	人工智能
专　　业	电子信息
年　　级	2020

武汉纺织大学研究生处制

注：（1）正文宋体小四 1.5 倍行距；（2）不要列出源程序；（3）页数不少于 8 页；

一、 引言

本文主要研究两种基本的搜索技术，宽度优先搜索（Breadth-First Search）与深度优先搜索（Depth-First Search），讨论两种算法的优缺点及可行的优化或拓展。

BFS 和 DFS 最早的提出和一个著名的问题有关，柯尼斯堡七桥问题（18 世纪柯尼斯堡有一条河，中心有两个小岛，两个小岛和两岸总共有七座桥连接，每座桥只能走一次，如何把所有桥都走一遍）如图 1 所示。该问题 1936 年被数学家欧拉解决，由此展开了一个著名的学科——图论。图论是以“图”为研究对象的一个数学分支，是组合数学和离散数学的重要组成部分。图是用来对对象之间的成对关系建模的数学结构，由“顶点”（又称“节点”或“点”）以及连接这些顶点的“边”（又称“弧”或“线”）组成。

在计算机领域，图是一种灵活的数据结构，图可以分为有向图和无向图，一般用 G=(V,E) 来表示图（一般 V 为顶点，E 为边）。经常用邻接矩阵或者邻接表来描述一副图。

在图的基本算法中，最初需要接触的就是图的遍历算法，根据访问节点的顺序，可分为广度优先搜索（BFS）和深度优先搜索（DFS）。

BFS 简单来讲就是先访问离自己最近的节点，再访问后继节点，这种算法的工作方式是从根节点开始，从内向外的逐步遍历所有的节点。由此可知，深度较浅（距离根节点近）的节点总是先被访问到。

DFS 简单来说就是顺着一条路径一直走下去，直到无路可走，然后回溯选择其他未访问过的节点。显然，这种算法他具有递归与回溯的性质。

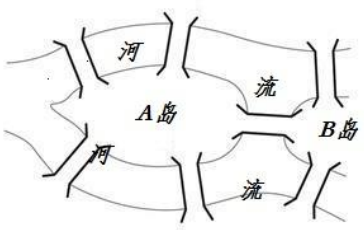


图 1

二、 相关工作

2.1 BFS 的优缺点及优化

BFS 算法的优点是具有完备性，即只要有解，一定能找到解。在有限的深度下，BFS 肯定能够找到目标节点。

但是 BFS 的缺点同样明显，如果深度太大，BFS 的时间复杂度与空间复杂度让人难以接受。以搜索树为例，假设目标节点的深度为 d,每个节点有 b 个子节点，那么第 0 层有 1 个根节点，第 1 层有 b 个子节点，第 2 层有 b^2 个子节点,…,第 d 层是 b^d 个子节点。

若访问节点时进行目标检测，已访问节点的总数为：

$$1 + b + b^2 + \cdots + b^d$$

则算法时间复杂度为 $O(b^d)$。BFS 每次访问需要保存生成的节点，以便查找最短路径，则空间复杂度为 $O(b^d)$。指数级别的复杂度是很大的，一但深度过大不管是内存需求还是时间需求都让人无法接受。

2.2 一致代价搜索（Uniform Cost Search）

BFS 算法没有考虑每一步的代价，即缺省的将节点的深度作为了代价。如果每一步的代价不同，那么 BFS 就无法找到最短路径，因为它总是从根节点最近开始访问，而不是代价最低。

实际上，在很多问题中，步数不是关键，代价才是关键。例如，从 A 点到 B 点有两种方式，坐公交直达或者坐地铁转乘，然而公交直达的时间是 2 个小时，地铁换乘只需要 1 个小时，在时间代价上，即使地铁花费的步数（换乘）更多，但在时间上还是占优势。

所以，我们可以显式的为每一步引入代价函数或者一直代价（Uniform Cost）,并对基本 BFS 进行修改，使其不再优先访问深度最浅的节点，而是优先访问代价最低的节点。这种算法被称为一致代价（Uniform Cost Search）算法。

2.3 DFS 优缺点及优化

DFS 的优点在于空间复杂度。DFS 算法只需要存储一条从根节点到叶节点

的路径以及路径上所有未被访问过的兄弟，其余的叶节点不在此次搜索中则不用花费存储空间，在最坏的情况下，路径的最大深度为 m，每个子节点有 b 个后继节点，则 DFS 只需要保存 O(bm)个节点。

DFS 的缺点在于时间复杂度。DFS 的时间复杂度取决于访问的节点个数，也就是说，如果目标节点在后面的子节点上，目标节点深度为 d,树的最大深度为 m，可能比 d 要大的多，但是 DFS 访问时依旧会访问到 m 层，在最坏的情况下，时间复杂度为 O（b^m）。而且 m 可能是无限的，那 DFS 直接就无法访问到目标节点，此时没有完备性。

2.4 回溯搜索（Backtracking Search）

回溯搜索是 DFS 算法的变形，所需的内存空间更小。BS 算法每次只产生一个后继子节点，不会一次生成所有的后继子节点，但每个被部分访问的节点要记住下一个待访问的后继子节点。算法思想：按优先条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择。回溯算法一个经典例子就是解决八皇后问题。

2.5 深度受限搜索(Depth-Limited Search)

DLS 是 DFS 的一种优化，它在搜索到一定深度 l 时就进行回溯或者停止，可以解决 DFS 在无限状态空间中的不完备性问题。

显然，深度 l 的选择是个问题，若目标节点深度为 d ，在 l<d 的时,DLS 就不是完备的。而当 l>d 时，DLS 虽然是完备的，但又不是最优的。DLS 的时间复杂度是 O(b^l），空间复杂度是 O(bl)。

2.6 迭代加深的 DFS(Iterative Deep Depth-First Search)

IDDFS 是 DFS 的拓展，是一种迭代使用 DFS 的策略，可以用来确定目标节点的最浅深度 d。执行过程：从深度 0 开始，执行一次 DFS 算法（相当于执行 l = 0 的 DLS),接下来从深度为 1 开始,执行 DFS(相当于 l = 1 的 DLS)，依次执行，当 l = d 时就能找到目标节点。

IDDFS 同时具备 BFS 和 DFS 的优点，它的空间复杂度与 DFS 类似为 O(bd)。

当分支因子优先时，该算法是完备的。当路径代价非递减函数（路径深度越来越浅）时，该算法是最优的。时间复杂度看似很高，似乎每一次都要重复的执行 DFS 算法，但是实际上，深度较大的节点生成的次数就较少，例如，深度为 d 的节点生成了 1 次，则深度为 d-1 的节点生成了 2 次，依次类推，根节点的子节点生成了 d 次，所以算法生成的总节点数为:

$$db + (d-1)b^2 + (d-2)b^3 + \cdots + b^d$$

时间复杂度和 BFS 一样还是 $O(b^d)$。

整体上看，IDDFS 将 DFS 和 BFS 结合在一起使用。

三、 算法描述（伪代码）

Input：一张图 G，起点 v 和终点 goal，存放最短路径的列表或者字典 came_from{}

Output：是否找到终点，最短路径

3.1 BFS 搜索

def BFS(G, v,goal,came_from)

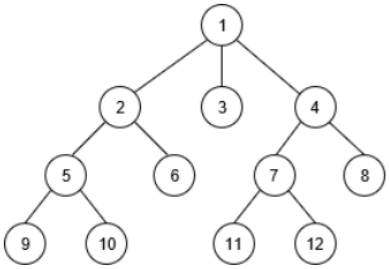
1. 创建队列，将起点 v 入队，并标记起点 v 为未访问节点；
2. 队列不为空就循环{ 将队头元素赋值给 v,队头元素出队；如果 v 未被访问，再判断 v 是否 goal，是，返回 v；不是，标记 v 已被访问，遍历 v 的子节点 w，依次入队，并保存 w 的前驱为 v}
3. 若没有找到 goal，返回空

3.2 DFS 搜索

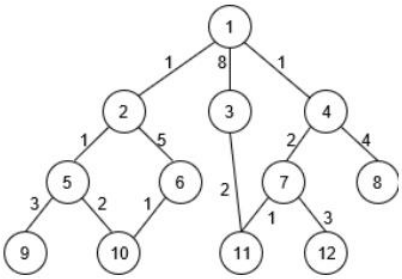
def DFS(G,v,goal,came_from):

1. 如果存放列表为空，标记起点 v 为未访问节点
2. 如果 v 是目标节点，返回 v
3. 遍历 v 的子节点 w，如果 w 未被访问过，保存 w 的前驱为 v，递归 result = DFS（G, w,goal,came_from），如果 result 为目标节点，则返回 result
4. 若没有找到 goal,返回空

四、 算法验证（结果与分析）



4-1.G1



4-2.G2

4.1 用队列实现 BFS

```
G:\人工智能\Maze_DBFS\Maze\venv\Scripts\python.exe G:/人工智能/Maze_DBFS/Maze/si
搜索过程为:
1->2->3->4->5->6->7->8->9->10->11->12->end
start: 1
goal: 12
1->4->7->12->end
```

基本 BFS 算法，使用图 4-1.G1,表现为顺序遍历，用广度优先遍历了所有节点，并将 came_from 中对应节点的 value 存放在栈中（came_from 中 key 为节点，value 为 key 的前驱节点），最后依次出栈。

4.2 用 OPEN-CLOSED 实现 BFS

```
搜索过程为:
1->2->3->4->5->6->7->8->9->10->11->12->end
start: 1
goal: 12
1->4->7->12->end
```

与基本的 BFS 算法没有太大区别，添加了 open 表和 closed 表分别存放已访问节点和未访问节点。

4.3 UCS 算法实现

```
搜索过程为:
1->2->4->5->7->10->11->6->8->9->12->end
start: 1
goal: 12
1->4->7->12->end
```

UCS 算法，使用图 4-2.G2 作为遍历的图，图中添加了每一步的代价，代价不同，访问的优先级不同，所以 UCS 的搜索过程不再是 BFS 的顺序遍历，而是根据每个节点的代价优先级来访问后继节点。从搜索过程可以看到节点 3 甚至没有被访问，说明 1->3 的代价还要大于 1->12 的代价，节省了步数。

4.4 DFS 算法递归实现

```
搜索过程为:
1->2->5->9->10->6->3->4->7->11->12->end
start: 1
goal: 12
1->4->7->12->end
Process finished with exit code 0
```

DFS 算法递归实现，使用图 4-1.G1 作为遍历图，用递归的方法先搜索某一条路径，直到没有后继节点，才回溯到有新路径的节点开始搜索。

4.5 DFS 算法非递归实现

```
搜索过程为:1->4->8->7->12->end
start: 1
goal: 12
1->4->7->12->end
Process finished with exit code 0
```

DFS 算法非递归实现，使用图 4-1.G1 作为遍历图，非递归使用 stack 的方式来实现 DFS 算法，由于栈是 LIFO，所以第一条遍历的路径不在是 1-2-5-9，而是 1-4-8。所以搜索过程发生了很大的变化。

4.6 DFS 算法的 OPEN-CLOSED 实现

```
搜索过程为:
1->4->8->7->12->end
start: 1
goal: 12
1->4->7->12->end
Process finished with exit code 0
```

DFS 的 OPEN-CLOSED 实现和非递归实现很像，添加了 open 栈和 closed 栈分别存放已访问节点和未访问节点。

4.7 IDDFS 算法实现

```
搜索过程为:
1->1->2->3->4->1->2->5->6->3->11->4->7->8->1->2->5->9->10->6->10->3->11->4->7->12->end
start: 1
goal: 12
1->4->7->12->end
Process finished with exit code 0
```

IDDFS 算法，使用图 4-2.G2 作为遍历图，迭代使用 DFS，所以出现了重复遍历同一个节点的现象，有一些节点被访问了不只一次，每次遍历的深度都增加，在这种简单图上感觉更加复杂了，没有体现出该算法的优点。

4.8 迷宫算法 BFS 和 DFS 的实现

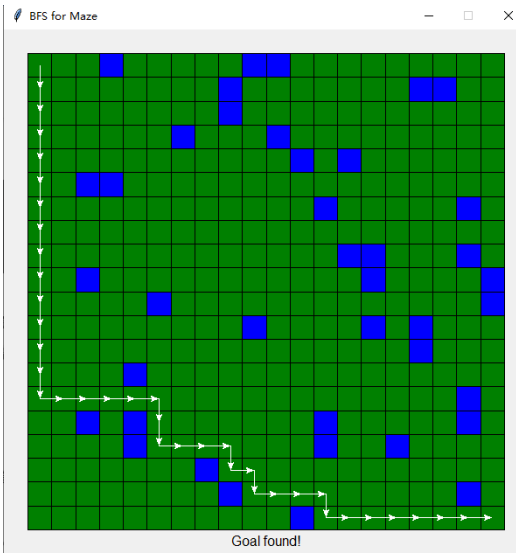


图 4-3 BFS 搜索

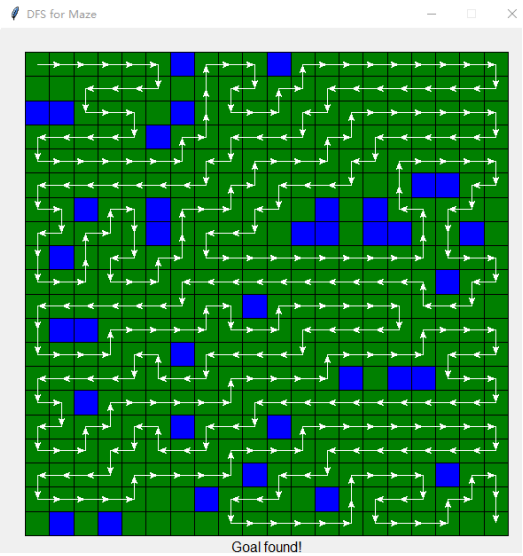


图 4-4 DFS 搜索

由图 4-4 和图 4-5 可一看出在解决迷宫问题上 BFS 和 DFS 显著区别。可以看出 BFS 搜索过程经过的路径明显要少很多，直接就是冲着迷宫的出口进行，而 DFS 则几乎遍历了整个迷宫中所有的路径。当然这并不能说明 BFS 算法优于 DFS 算法，只能说两个算法的遍历方式的区别，在迷宫问题中的表现形式不一样，DFS 表现为走到到无路可走再选择其他可行路径，BFS 则先搜索身边的路径。

五、 算法应用

BFS 和 DFS 算法作为图论中的基本算法，应用非常的广泛。例如，紧急出警、车载导航、物流配送和管线规划与设计等地理信息系统(GIS)相关领域，而在手机上常看到"连连看"游戏中,需要一种特殊的单源路径搜索方案,通过记录候选节点的转角数来筛选满足要求的路径。依据这一算法的两种实现策略就是宽度优先搜索(BFS)和深度优先搜索(DFS)。

宽度优先搜索还在在模型检查、模式数据库计算以及确定问题状态空间半径等领域中有着重要的应用。宽度优先搜索作为一种系统的图搜索算法,它通常比深度优先搜索有效得多,后者无法探测出表示同一状态的重复节点并且需要在产生所有路径后才能确定出最优解。但是,宽度优先搜索的适用规模因其空间需求大的特点受到极大限制。利用磁盘作为二级缓存来克服 BFS 内存限制的相关技术被提出。然而,单台计算机的存储能力总是有限的。因此引入分布式并行宽度优先搜索,结合多台机器的计算能力和存储资源来完成大规模的宽度优先搜索。

BFS 和 DFS 还被应用于各种各种系统的故障检测，智能坚强电网是我国电力系统发展的愿景，对电力系统的全局可观测性是实现电网智能化的前提条件。但是电力布局范围越大，故障的排除困难就会增加，运用 BFS 和 DFS 确定了发生故障区域，排除故障类型。

DFS 和 BFS 得优化或者拓展还广泛应用于连通分量计算、拓扑排序、社区检测等。随着大数据时代的来临,数据规模不断增大,数据的拓扑结构也越来越复杂,基于内存的 DFS 算法无法适用于大规模图数据,无法满足日益增长的数据规模和查询传输有效率的需求。因此 DFS 和 BFS 算法的拓展和优化算法，例如 UCS、IDDFS 和更多一些的算法应用非常广泛。

六、 结论与展望

通过对两种无信息的搜索算法 BFS 和 DFS 的研究，深刻的学习两种算法的基本思想和代码实现，对比分析其优缺点，并对其进行优化和拓展，学习了一些变体及优化的算法，不能直观的说两种算法孰优孰劣，他们在不同的情况下表现也不同，一般对各种实际问题，要根据问题选择算法或者使用多种算法融合，对算法进行变形来更好的达到需求。由此，关于 BFS 的 DFS 的研究值得深入。

随着中国互联网发展在数字经济、技术创新、网络惠民等方面不断取得重大突破，有力推动网络强国建设迈上新台阶，互联网不仅连接触手可及，使用也更加便捷。随着“互联网+”加速与产业融合，数字经济已成为中国发展的新引擎。报告显示，我国数字经济规模已达 31.3 万亿元，位居世界前列，占 GDP 的比重达 34.8%。新技术的加速应用，催生了新的产业形态，有力提升经济发展质量、推动产业结构优化升级。疫情防控中，在线教育、在线问诊、网络视频、网络购物、网络音乐等应用的快速增长，充分展现了技术发展带来的新机遇。

新基建驱动下，新技术、新产品、新业态更新迭代加速，亟待相关行业抓住机遇，迎来新一轮快速发展。在这快速的发展中会产生许多的问题与需求，而 DFS 和 BFS 作为基本的搜索技术，刚好能够对这些问题提供解决思路。

七、 参考文献

1.Wikipedia: Graph theory.. Accessed on Oct.14,2020

2. [github]. https://duxiaoqin.github.io/ .Accessed on Oct.15,2020

3.冷劲夫. 胡燮.胡颖.单路径算法的实现与比较[P].地理空间信息,2010,08(3):154-156

4.周波.配电网 PMU 优化配置及故障定位算法研究[D].长春：长春工业大学，2017

5. 徐瑛蔚. 基于分布式存储的大规模图的深度优先搜索算法研究[D]. 辽宁: 辽宁大学，2018