中山大学数据科学与计算机学院
计算机科学与技术专业-人工智能
本科生实验报告

（2018-2019 学年秋季学期）

学　　号：16337113

姓　　名：劳马东

教学班级：教务 2 班

专　　业：超算

一、实验题目

1、 从给定类型一和类型二中分别选择一个策略解决迷宫问题。

类型一：BFS 、DFS

类型二：一致代价、迭加深双向搜索

2、 代码要求使用 python 或者 C++；

3、 需有对实现的策略（两个）原理解释；

4、 需有两种策略的实验效果（四个方面算法性能）对比和分析。

二、实验内容

（一）算法原理

1、 迷宫问题的形式化

（1） 问题定义

- 初始状态：处在迷宫起点格 S
- 行动：向上下左右四个方向移动一格
- 状态空间：模型的所有格子
- 目标测试：处在迷宫终点格 E
- 路径耗散：从起点 S 到某点 P 经过的格子数（含 P）

（2） 问题的解：从起点 S 到终点 E 的移动方向的序列

2、 树搜索基本框架

- $Frontier$：边界，是想探索而未探索的状态的集合，在最初时包含所有的初始状态；
- $Successors(x)$：一个函数，表示节点 x 的后继节点；
- $Goal?$：目标测试函数

算法 1 树搜索

function $TreeSearch(Frontier, Successors, Goal?)$
    if $Frontier$ is empty **then**
        return failure
    end if
    $Curr \leftarrow$ select state from Frontier
    if $Goal?(Curr)$ **then**
        return $Curr$
    end if
    $Frontier' \leftarrow (Frontier - \{Curr\}) \bigcup Successors(Curr)$
    return $TreeSearch(Frontier', Successors, Goal?)$
end function

3、 深度优先搜索（DFS）

每次探索时选择栈顶的节点进行扩展，因此，该算法倾向于搜索更深的节点，直到找到解或到达临界节点才回溯。在该算法中，$Frontier$ 是栈中所有节点，一个节点的 $Successors$ 就是它上下左右四个方向（如果有）的邻居。算法大致的过程如下：

(a) 起始点 S 入栈；

(b) 如果栈空，无解退出；

(c) 从栈顶选择一个节点 P；

(d) 检测 P 是否终点 E，是就找到解，否则扩展 P 的所有邻居节点入栈，重复步骤 (b)。

算法 2 深度优先的树搜索

function $DFS(StackFrontier, Successors, Goal?)$
    if $StackFrontier$ is empty **then**
        return failure
    end if
    $Curr \leftarrow$ extract_top($StackFrontier$)
    if $Goal?(Curr)$ **then**
        return $Curr$
    end if
    push($StackFrontier, Successors(Curr)$)
    return $TreeSearch(StackFrontier, Successors, Goal?)$
end function

4、 深度受限搜索（DLS）

DFS 的缺点是它会一直向下搜索到最深才会回溯，当解在浅层时，算法的效率就极差。深度受限的做法是限制探索的最大深度，超过这个深度不再往下扩展。显然，最大深度为 L 的深度受限搜索等价于最长路径为 m 的 DFS，因此它不具备完备性和最优性，时间复杂度为 $O(b^L)$，空间复杂度为 $O(bL)$。

5、 迭代加深搜索（IDS）

DFS 和 BFS 各有优缺点，DFS 时间复杂度是指数级，但空间复杂度是线性的；BFS 时间和空间复杂度都是指数级。IDS 提出了一个折中的策略，综合两者的有点。顾名思义，它通过一个循环，逐渐增大最大深度，每次都是用 DLS 去求解直到找到解。

算法 3 迭代加深搜索

function $IDS(Successors, Goal?)$
    for depth := 0 **to** max_depth **do**
        $result \leftarrow$ DLS(empty stack,$Successors, Goal?, depth$)
        if $result$ is not failure **then**
            return $result$
        end if
    end for
    return failure
end function

6、 环检测

搜索算法容易出现重复探索的问题，为了避免多次探索同一个节点，便引入了环检测。

环检测使用一个已探索节点的集合来记录走过的节点，即当扩展一个节点以将其后继存入边界时，需要检测该后继节点是否在已探索节点集合中，如果在就不扩展，即：

$$SuccessorsWithCycleCheck(x) = Successors(x) - expandedSet \tag{1}$$

算法 4 带环检测的树搜索

function $TreeSearchWithCycleCheck(Frontier, Successors, Goal?, expandedSet)$
  if $Frontier$ is empty **then**
    return failure
  end if
  $Curr \leftarrow$ select state from Frontier
  if $Goal?(Curr)$ **then**
    return $Curr$
  end if
  $expandedSet' \leftarrow expandedSet \bigcup \{Curr\}$
  $Frontier' \leftarrow (Frontier - \{Curr\}) \bigcup SuccessorsWithCycleCheck(Curr)$
  return $TreeSearch(Frontier', Successors, Goal?)$
end function

（二）关键代码

1、 DFS

```python
def depth_first_search(maze, start, end, wall):
    """
    :param maze: 迷宫，numpy.array/numpy.matrix
    :param start: 起点坐标，tuple
    :param end: 终点坐标，tuple
    :param wall: 墙字符，str
    :return: 从起点到达终点经过的所有坐标点，list
    """
    m, n = maze.shape
    path = []
    # dfs是最大深度为正无穷的DLS
    # 为了避免无限循环，将m*n视作正无穷，即当探索完迷宫的所有坐标点后退出
    # 用一个字典记录已探索节点
    depth_limited_search(maze, start, end, path, wall, m * n, dict())
    return path
```

2、 IDS

```
def iterative_deepen_search(maze, start, end, wall):
    m, n = maze.shape
    path = []
    # 为了统一，DLS中的深度表示层数，因此从1开始，到m*n-1结束
    for i in range(1, m * n):
        # DLS返回是否有解，如果找到解，就返回这条路径，不需要再继续迭代
        # 因为这已经是最优解
        if depth_limited_search(maze, start, end, path, wall, i, dict()):
            break
    return path
```

3、 DLS

深度受限搜索是 DFS 和 IDS 的基础，它的状态空间的每个元素都是 (坐标，路径耗散) 二元组。为什么要这样？

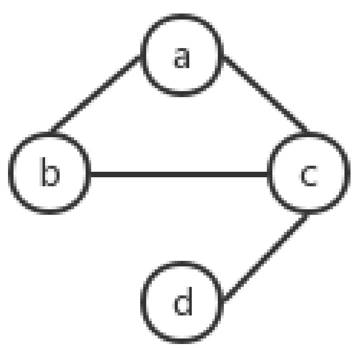
考虑图1(a)的情况，起点为 a，终点为 d，后继节点按字典序扩展，带环检测，最大深度为 2。算法的过程如下：

(a) 探索 a，深度剩余 2，扩展 b 和 c，边界为 $\{c_a, b_a\}$；

(b) 探索 b，深度剩余 1，扩展 c，边界为 $\{c_a, c_b\}$；

(c) 探索 c，深度为 0，不再扩展，回溯到 c_a；

(d) c 已经被探索过，无解退出。



(a) DLS

但实际上这个问题应该是有解的，a-c-b 的路径深度刚好为 2。问题出在哪里？因为第一次经过 c 和回溯是经过 c 可能是两种不同的状态。第一次经过 c 时，路径耗散为 2（a→b→c），而回溯时为 1（a→c）。因此，不能简单地认为探索了一个坐标点，下次就不能探索它了，还应考虑费用。

在这个实验中，边的权值都是相同的，因此可能用剩余深度来表示路径耗散，剩余深度越小，路径耗散越大。这时，判断一个节点是否需要可以扩展需要考虑：

(a) 其坐标点是否被探索过；

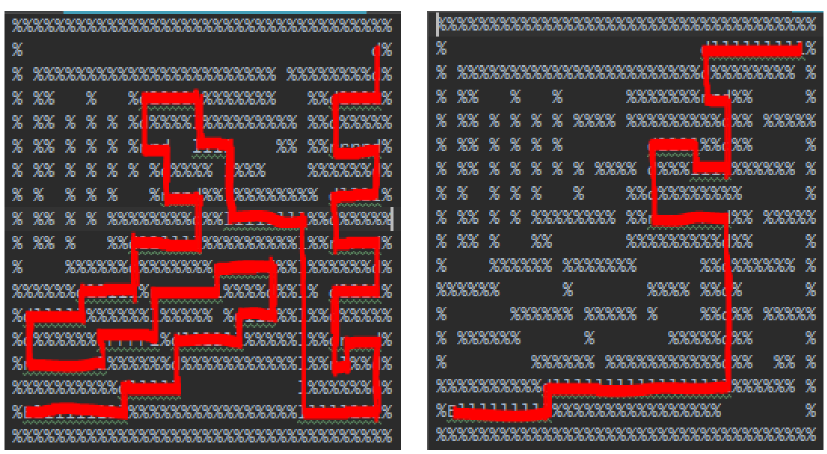
(b) 剩余深度是否比上一次探索时剩余深度大。

```python
def depth_limited_search(maze, cur, end, path, wall, depth, visited):
    # 以(坐标, 路径耗散)作为状态
    visited[cur] = depth - 1
    path.append(cur)
    if cur == end:
        return True
    else:
        if depth != 0:
            for n in successors(maze, cur, wall):
                # 环检测中, 可以多次经过一个坐标点, 只要路径耗散比上次探索的小
                if n not in visited or depth - 1 > visited[n]:
                    if depth_limited_search(maze, n, end, path, wall, depth - 1, visited):
                        return True
    # 没有找到解, 回溯时要从路径中去掉探索过的节点
    path.pop(-1)
    return False
```

三、实验结果及分析

（一）实验结果展示

在寻找四个方向的后继节点时，按照上、下、左、右的顺序遍历。如图1(b)，DFS 算法从起点往下开始探索，绕了很长的路最终到达终点，经过了 165 个坐标点（不包括终点）；图1(c)的 IDS 算法找到了一条最短的路径，经过 69 个坐标点。



(b) DFS　　(c) IDS

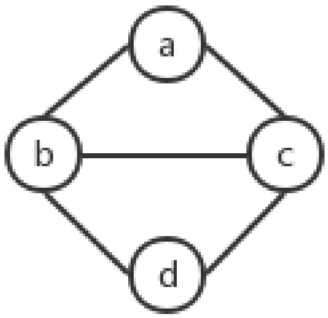
（二）评测指标展示及分析

搜索策略	完备性	最优性	时间复杂度	空间复杂度	运行时间	探索节点数
DFS	加环检测	无	$O(b^m)$	$O(bm)$	0.02	165
IDS	有	边权相等	$O(b^d)$	$O(bd)$	0.23	69

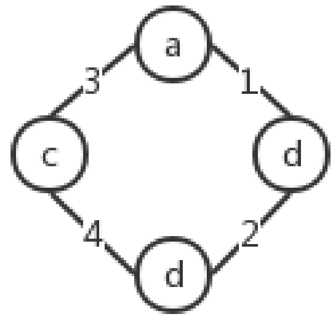
与 DFS 相比，IDS 有完备性和最优性，而 DFS 无论如何也无法保证最优；IDS 的时间复杂度和空间复杂度在 $m > d$ 时比 DFS 小，而 $m > d$ 在搜索问题中是极常见的；在实验中，IDS 虽然找到了最优解，但是用了较长的时间，因为它用了 69 次迭代才找到解。

1、 DFS

（1） 完备性：DFS 不具备完备性。考虑图1(d)，起点为 a，终点为 d，后继节点按字典序扩展。那么，每次迭代的 $Frontier$ 为 $\{a\}$、$\{c, b\}$、$\{c, d, c, a\}$、$\{c, d, c, c, b\}$...，永远找不到解。



(d) Completeness



(e) Optimality

（2） 最优性：DFS 无最优性. 考虑图1(e)，起点为 a，终点为 d，后继节点按从大到小扩展，最终会找到 a-c-d 路径，但是很明显 a-b-d 路径费用更低。

（3） 时间复杂度：由于算法探索到最深处才回溯，因此时间复杂度与最长路径的长度相关，假设为 m。在最坏情况下，解是状态空间中最后一个被探索的，即在找到解之前，其他所有的节点都已经被探索。因此完全 b 叉树搜索的时间为 $\sum_{i=0}^{m} b^i$，则时间复杂度为 $O(b^m)$。

（4） 空间复杂度：DFS 的回溯点是下一个未探索的兄弟节点，栈中需存储所有后继。在探索最长的一条路径（长度为 m，有 m+1 个节点）时，每个经过的节点都存储了它的后继，最多为 b 个，因此栈中最多存储 bm 个节点，即空间负复杂度为 $O(bm)$，这是线性的复杂度。

2、 IDS

（1） 完备性：当加上环检测时，IDS 具有完备性；

（2） 最优性：加上环检测，当边的权值相等时具有最优性。因为如果算法找到解，就肯定是最浅层的解，同时也是路径费用最小的解；

（3） 时间复杂度：假设解的深度为 d，那么，深度为 0 的节点被探索 d+1 次，深度为 1 的节点被探索 d 次，以此类推，时间为 $\sum_{i=0}^{d}(d+1-i)b^i$。因此时间复杂度为 $O(b^d)$，这 BFS 的时间复杂度相当；

（4） 空间复杂度：若解的深度为 d，IDS 探索的最长路径的长度也为 d，由于使用的是 DFS，故空间复杂度为 $O(bd)$。

四、思考题

4.1. 优缺点

1、 BFS：优点是时间复杂度低，为 $O(b^d)$，在不加环检测时有完备性，在边权相同时有最优性；缺点是空间复杂度太大，为 $O(b)$，当状态空间很大时，可能会由于计算机内存不够而找不到解；

2、 DFS/深度受限搜索：优点是空间复杂度低，为 $O(bm)$，而且当解较多时容易迅速找到解；缺点没有最优性，不加环检测时没有完备性，时间复杂度太大，为 $O(b^m)$，在 m 比 d 大很多时效率极差；

3、 一致代价：优点是只要边权为正数就具有完备性和最优性，这是由其选择节点的策略决定的；缺点是空间复杂度太大，为指数级；

4、 迭代加深：优点是结合了 DFS 和 BFS 的优点，时间复杂度为 $O(b^d)$，空间复杂度为 $O(bd)$，加了环检测之后有完备性，边权一致时有最优性；缺点是在小规模问题上可能不如 DFS 或 BFS，因为需要迭代多次重复搜索；

5、 双向搜索：优点是时间复杂度和空间复杂度相对于 DFS 降低很多，为 $O(bd/2)$；缺点是需要两侧都是边权相同的 BFS 才有最优性，否则甚至可能找不到解；

4.2. 适用场景

1、 BFS：相邻节点之间的路径消耗是相同的

2、 DFS/深度受限搜索：无需找到最优解，只要能找到解；问题的解较多，尤其是多分布在树搜索状态空间的左侧；

3、 一致代价：权值都是正数，且对最优性和时间复杂度要求较高；

4、 迭代加深：对时间复杂度和空间复杂度要求都较高；

5、 双向搜索：BFS 能使用的地方双向搜索一般都能使用，而且时间和空间复杂度有很大降低。