

# 中山大学数据科学与计算机学院

## 计算机科学与技术 人工智能

### 本科生实验报告

(2018-2019) 学年秋季学期

课程名称: Artificial Intelligence

|      |          |         |          |
|------|----------|---------|----------|
| 教学班级 | 16级计科二班  | 专业 (方向) | 计算机科学与技术 |
| 学号   | 16337334 | 姓名      | 周启恒      |

## 无信息搜索

### (1) 原理

#### 类型一: BFS

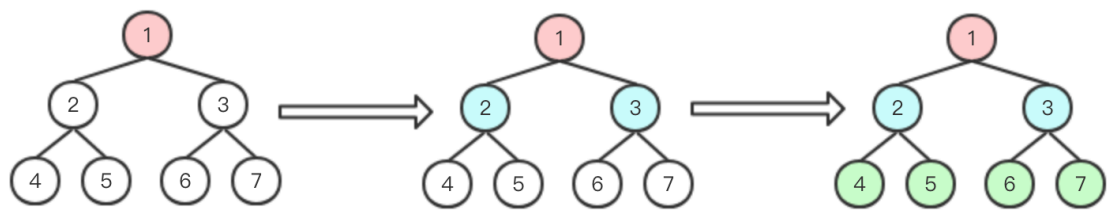
BFS是一种盲目搜索法, 按照层结构, 系统地展开并检查图中的所有节点。

搜索步骤:

1. 从根节点开始探索, 将它的叶节点放入队列中。
2. 从队列中依次拿出节点, 对他们分别进行步骤1。

完备性: 因为搜寻了整个图/树, 一定能找到解。

最优性: 宽度优先, 每次扩展一层, 找到的一定是一条从起点到终点的最短的路径。



#### 类型二: Iterative deepening search

本质: 持续执行有深度限制的深度优先搜索, 不断增加深度, 直至找到目标。

在单次搜索中, 按照深度优先搜索的方式, 但在整个迭代加深的过程中不断扩展深度, 类似宽度优先。这样保证了时间复杂性相同的情况下, 减少了空间复杂度。

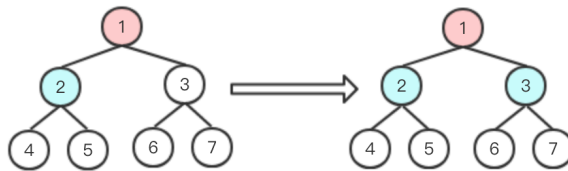
搜索步骤：

1. 设置depth，从1到 $+\infty$
2. 按照depth进行深度优先搜索，限制搜索的深度
3. 若找到目标，则停止搜索，返回深度

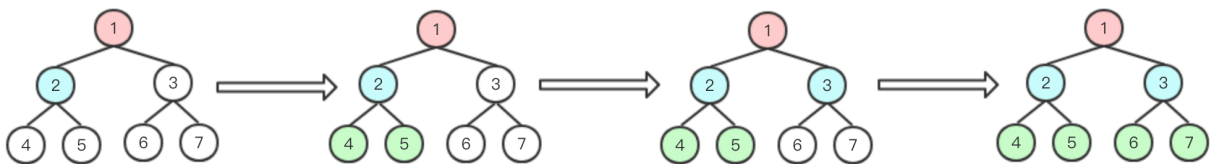
完备性：搜寻了整个图/树，一定能找到解。

最优性：全局上类似宽度优先，每次增加深度，找到的一定是一条从起点到终点的最短的路径。

- depth = 2



- depth = 3



## (2) 关键代码（带注释）

### BFS

```
def BFS(start, end, maze, visit):  
    ...  
    start:起点  
    end:终点  
    maze:地图  
    visit:标记访问过的节点的矩阵  
    ...  
    q = Queue()  
    q.put(start)  
  
    parent_dict = {}  
    path = np.zeros( (len(maze), len(maze[0])) )  
    while not q.empty():  
        head = q.get()  
        #找到终点  
        if head == end:  
            print("exit found")  
            BFS_print_path(parent_dict, path, start, end)  
            return  
        #向四个邻居扩展,若邻居可扩展,即加入队列中
```

```

if illegal(head[0] - 1, head[1], maze, visit):
    q.put( (head[0] - 1, head[1]) )
    parent_dict[(head[0] - 1, head[1])] = head
    visit[head[0] - 1][head[1]] = 1

if illegal(head[0] + 1, head[1], maze, visit):
    q.put( (head[0] + 1, head[1]) )
    parent_dict[(head[0] + 1, head[1])] = head
    visit[head[0] + 1][head[1]] = 1

if illegal(head[0], head[1] - 1, maze, visit):
    q.put( (head[0], head[1] - 1) )
    parent_dict[(head[0], head[1] - 1)] = head
    visit[head[0]][head[1] - 1] = 1

if illegal(head[0], head[1] + 1, maze, visit):
    q.put( (head[0], head[1] + 1) )
    parent_dict[(head[0], head[1] + 1)] = head
    visit[head[0]][head[1] + 1] = 1

```

## iterative deepening search

```

def iterative_deepening_search(x, y, end, depth, maze, visit):
    #指定depth, 限制深度优先搜索的深度。
    if (x, y) == end:
        print('exit found')
        print_path(visit)
        return True
    else:
        if depth < 0:
            return False
        #向四个邻居扩展
        if illegal(x + 1, y, maze, visit):
            #标记扩展的邻居
            visit[x+1][y] = 1
            if iterative_deepening_search(x+1, y, end, depth-1, maze,
visit):
                return True
            #回溯
            visit[x+1][y] = 0
        if illegal(x - 1, y, maze, visit):
            visit[x-1][y] = 1
            if iterative_deepening_search(x-1, y, end, depth-1, maze,
visit):
                return True
            visit[x-1][y] = 0
        if illegal(x, y + 1, maze, visit):
            visit[x][y + 1] = 1

```

```

        if iterative_deepening_search(x, y + 1, end, depth-1, maze,
visit):
            return True
        visit[x][y + 1] = 0
    if illegal(x, y - 1, maze, visit):
        visit[x][y - 1] = 1
        if iterative_deepening_search(x, y - 1, end, depth-1, maze,
visit):
            return True
        visit[x][y - 1] = 0
    return False

```

## 判断节点是否可以扩展

```

def illegal(x, y, maze, visit):
    '''
    x,y表示当前节点的坐标
    '''
    #判断节点是否在迷宫范围内，且可以访问（即迷宫中为0且未访问过）
    if x < 0 or x >= len(maze) or y < 0 or y >= len(maze[0]):
        return False
    elif maze[x][y] != 1 and visit[x][y] == 0:
        return True
    return False

```

## 调用迭代加深搜索

```

'''
从1开始遍历深度，直至找到合适的深度。
'''
depth = 1
while not iterative_deepening_search(start[0], start[1], end, depth, maze,
visit):
    depth += 1
iterative_deepening_search(start[0], start[1], end, depth, maze, visit)

```

## 实验结果展示

算法比较：

- 均具有正确性、易读性、健壮性。
- 时间、空间
  - 时间比较：

