中山大学计算机学院人工智能本科生实验报告

2022学年春季学期

课程名称：Artificial Intelligence

教学班级	人工智能（陈川）	专业（方向）	计算机科学与技术人工智能与大数据
学号	20337025	姓名	崔璨明

一、实验题目

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、迭代加深、双向搜索

实验题目给出的迷宫（其中S为起点，E为终点，1表示墙，0是可通行）：

```
11111111111111111111111111111111111
10000000000000000000000000000000S1
10111111111111111111111101111111101
10110001000100000011111100011000001
10110101010101111011111111011011111
10110101010100000000000011011000001
10110101010101011110011100011111101
10100101010001000011011111110000001
10110101011111111011000000011011111
10110100011000000011111111011000001
10000111111011111110000011011111101
11111100000010000000111101101000001
10000001111110111110100001101101111
10111111000000100000001111101100001
10000000011111101111111111011001101
11111111110000000000000000011111101
1E000000000111111111111111000000001
11111111111111111111111111111111111
```

二、实验内容

1、选择实现的策略

在本次实验中，我选择了**DFS**（深度优先搜索）和**迭代加深搜索**这两个策略来解决迷宫问题。

2、算法原理

- DFS(深度优先搜索)：深搜的算法原理很简单，算法从某一个状态开始，使用合适的方式，不断地转移状态直到无法转移，然后回退到前一步的状态，继续转移到其他状态，如此不断重复，直到找到最终的解。
- 迭代加深搜索：先设定一个较小的数dm作为有界深度搜索的限制，若在此深度限制内找到了问题的解，则算法结束；若没有找到解则增大深度限制dm，继续搜索，当最后深度限制为m时，会生成深度为m的树。

3、算法伪代码

针对本次实验的迷宫问题，两种策略的算法伪代码如下：

DFS:

```
算法输入：迷宫map,起点坐标sx,sy
算法输出：起点到终点一条路径的坐标的集合
next=[[0,1],[-1,0],[0,-1],[1,0]] //表示移动方向的数组
ans=[] //答案数组
book[][]=0  //记录某个坐标是否被访问过

DFS(sx,sy,map,book):
        if(map[sx][sy]=='E'): return True
        for i in range(0,4){
                nx=sx+next[i][0]
                ny=sy+next[i][1]
                if 坐标(nx,ny)可以到达 and book[nx][ny]==0 :
                        book[nx][ny]=1
                        if DFS(nx,ny,map,book)==True:
                                ans.put((sx,sy))
                                return True
return False


DFS(sx,sy,mao,book)
return ans
```

迭代加深搜索:

```
算法输入：迷宫map,起点坐标sx,sy
算法输出：起点到终点一条路径的坐标的集合
next=[[0,1],[-1,0],[0,-1],[1,0]] //表示移动方向的数组
ans=[] //答案数组
```

```
book[][]=0  //记录某个坐标是否被访问过

lim_DFS(sx,sy,map,book,lim,curr):
        if(curr>lim): return flase
        if(map[sx][sy]=='E'): return True
        for i in range(0,4){
                nx=sx+next[i][0]
                ny=sy+next[i][1]
                if 坐标(nx,ny)可以到达 and book[nx][ny]==0 :
                        book[nx][ny]=1
                        if DFS(nx,ny,map,book,lim,curr+1)==True:
                                ans.put((sx,sy))
                                return True
return False


def iteration_dfs:
        for lim in range(min,max):
                flag=lim_DFS(sx,sy,map,book,lim,0)
                if flag==true: return ans
                else ans=[]   //重置答案数组
```

4、关键代码展示

首先读取文本文件中的迷宫信息，将其存在一个二维数组中，方便我们对迷宫进行搜索和修改，在这方面，两个策略的处理方式是一样的：

```
f = open('temp.txt','r')
map=[]
for line in open('temp.txt'):
    line = f.readline()
    map.append(line)
#print(map)
row=len(map)

for i in range(0,row):
    map[i]=list(map[i])
    map[i].pop(len(map[i])-1)

col=len(map[1])
#print(row,col)
col-=1
```

进行搜索时，我们需要判断下一步的状态是否合法，即能否到达下一个坐标，判断函数`can_walk()`如下：

```
def can_walk(a,b,map):
    if a>=0 and a<len(map) and b>=0 and b<len(map[0]) and (map[a][b]=='0' or map[a][b]=='E'):
        return True
    else:
        return False
```

采用DFS（深度优先搜索）策略的主要代码如下，next数组为状态转移的工具，book数组用于判断某个坐标是否被访问，ans数组用于存储最后得到的路径走过的坐标：

```
book=[]
next=[[0,1],[-1,0],[0,-1],[1,0]]#上下左右
ans=[]
```

```
def dfs(sx,sy,map,book):
    if(map[sx][sy]=='E'):
        return True
    for i in range(len(next)):
        if can_walk(sx+next[i][0],sy+next[i][1],map) and book[sx+next[i][0]][sy+next[i][1]]==0 :
            book[sx+next[i][0]][sy+next[i][1]]=1
            if dfs(sx+next[i][0],sy+next[i][1],map,book)==True:
                ans.append([sx,sy])
                return True
    return False
```

在实现迭代加深搜索策略时，首先要实现有界深度搜索算法，只需在深度优先搜索的基础上进行修改即可：

```
#深度受限搜索
def limit_dfs(sx,sy,map,book,lim,curr_lim):
    if curr_lim>lim:#curr_lim为当前深度，lim为限制深度，但超过限制时返回False
        return False
    if(map[sx][sy]=='E'):
        return True
    for i in range(len(next)):
        if can_walk(sx+next[i][0],sy+next[i][1],map) and book[sx+next[i][0]][sy+next[i][1]]==0 :
            book[sx+next[i][0]][sy+next[i][1]]=1
            if limit_dfs(sx+next[i][0],sy+next[i][1],map,book,lim,curr_lim+1)==True:
                ans.append([sx,sy])
                return True
    return False
```

迭代加深搜索的具体代码实现如下，在这次实验中，将深度的上限设置为 $low * col$，若超过该范围还未找到终点说明无解，最后的返回值为迭代的深度：

```python
#迭代加深搜索
def iteration_dfs(startx,starty,map):
    global ans
    dep=0   #dep为每次深搜的最大限制深度
    while 1:    #迭代加深直至得到答案
        ans=[]
        book=[]
        for k1 in range(0,row):
            book.append([])
            for k2 in range(0,col):
                book[k1].append(0)
    #print(startx,starty)
        tmp=limit_dfs(startx,starty, map,book,dep,0)
        if tmp==True:
            break
        else:
            dep+=1
        #理论上，搜索的深度在数值上不超过面积，所以单深度超过面积时说明没有答案
        if dep>col*row:
            return -1
    return dep
```

三、实验结果及对比分析

实验结果展示示例

使用普通的深度优先搜索策略得出的迷宫路径如下，该路径的长度为130，即经过了130步移动：

```
PS E:\VSCODE\py> python -u "e:\VSCODE\py\DFS.py"
路径长度为:  130
1111111111111111111111111111111111111
1000000000000000000000000000000000S1
1011111111111111111111111011111111101
1011000100010000001111111000110000001
1011010101010111101111111111011011111
1011010101010000000000000011011000001
1011010101010101111100111000011111101
1010010101000100001101111111110000001
1011010101111111110110000000011011111
1011010001100000001111111110110000001
1000011111110111111100000011011111101
1111110000001000000011111011010000001
1000000111111011111010000110110111111
1011111100000010000000111111011000001
1000000001111110111111111110110011101
1111111111000000000000000000011111101
1E0000000001111111111111111000000001
1111111111111111111111111111111111111
------------------
PS E:\VSCODE\py>
```

使用迭代加深搜索策略得出的迷宫路径如下，最后在限制深度为70，即经过70步移动时找到了终点：

```
PS E:\VSCODE\py> python -u "e:\VSCODE\py\iteration_dfs.py"
最后限制深度为:  70
1111111111111111111111111111111111
1000000000000000000000000000000S1
1011111111111111111111110111111101
1011000100010000011111110001000001
1011010101010111101111111101011111
1011010101010000000000001101000001
1011010101010101111001110000111101
1010010101000100001101111111000001
1011010101111111101100000000111111
1011010001100000001111111101000001
1000011111101111110000001101111101
1111110000001000000011110110100001
1000000111111011111010000110111111
1011111100000010000000111110100001
1000000001111110111111111110100110
1111111111000000000000000000111101
1E000000000111111111111111100000001
1111111111111111111111111111111111
PS E:\VSCODE\py>
```

在本次的实验结果显示上，迭代优先搜索显然优于普通的深度优先搜索，经过验证，迭代优先搜索得到的路径是最优路径，而DFS得到的显然不是最优路径。

对比和分析

DFS(深度优先算法)和迭代加深算法的4个标准评价对比如下：

算法	时间复杂度	空间复杂度	最优性	完备性
深度优先	$O(b^m)$	$O(bm)$	否	否
迭代加深	$O(b^d)$	$O(bd)$	是	是

（其中b为分支系数，即有多少个状态可以进行转移，d为解的深度，m为搜索树的最大深度）

运行时间对比分析：

由上表的理论对比可知，两种算法的时间复杂度接近，在本次实验中，我采用了python的time库中的计时工具来测量算法运行的时间，得出的结果如下：

DFS算法从开始运行到得出路径的时间：

```
PS E:\VSCODE\py> python -u "e:\VSCODE\py\test2.py"
0.0009989738464355469
路径长度为:  130
```

迭代加深算法从运行到得出路径的时间：

```
PS E:\VSCODE\py> python -u "e:\VSCODE\py\test2.py"
用时：  0.024000406265258 79
最后限制深度为：  70
```

可见，用于需要更改限制深度、判断是否越界等语句的存在，还有迭代加深算法的搜索树的宽度较DFS更大等原因，迭代加深算法的运行时间要比DFS的时间长。

运行时内存对比分析：

为对比两种策略所使用的空间的大小，我调用了python中`psutil`库和`os`库使用如下语句来输出两个程序运行时使用的内存：

```
print(u'当前进程的内存使用：%.4f GB'
% (psutil.Process(os.getpid()).memory_info().rss / 1024 / 1024 / 1024) )`
```

两种策略对比如下，第一个为DFS策略，第二个为迭代加深搜索策略：

```
------------------
当前进程的内存使用：0.0138 GB
PS E:\VSCODE\py>
```

```
------------------
当前进程的内存使用：0.0137 GB
PS E:\VSCODE\py>
```

可见，迭代加深策略所使用的内存空间比DFS较小。

最优性&完备性：

普通的DFS有在存在无限路径时无限运行下去的问题，且不具有最优性（如实验结果所示）和完备性，而迭代加深算法则具有最优性和完备性，在某些方面胜过DFS。它们的共同特点是搜索方向都依据了某一评价指标且搜索方向和搜索对象本身的属性无关。

四、思考题

这些策略的优缺点是什么？它们分别适用于怎样的场景？

答：

策略	优点	缺点	适用场景
宽度优先搜索	实现简单，具有最优性和完备性，因为BFS搜索过程中遇到的解一定是离根最近的，所以遇到一个解，一定就是最优解。	和同样实现简单DFS相比所用空间较大，因为在搜索过程中需要保存搜索过的状态，而且一般情况需要一个队列来记录。	用来搜索最短径路的解是比较合适的，比如求最少步数的解，最少交换次数的解。
一致代价搜索	是对BFS的改进，边界中，按路径的成本升序排列，每次选取成本最少的进行扩展，因此能确保到搜索第一次到某一个点是沿着最优的路径搜索到的，同样具有最优性和完备性。	和BFS一样要保存指数级的节点数量，需要消耗很多的空间，且在每种动作的成本是一样的时会退化为BFS。	搜索最短径路的解是比较合适，因为离终点深度小的情况下比较快。同样适合解决最短或最少问题。
深度优先搜索	其递归特性，使得算法代码简洁，需要保存的信息少，在解的深度较深时有优势。	不具有完备性和最优性，可能运行时间非常长，甚至在存在无限路径时无限运行下去的问题。	适合搜索全部的解，因为搜索时不能确保得到最优解，在需要得到全部解时更加适合。
深度受限搜索	是预先限制了搜索的深度L的DFS，在限制深度大于解的深度时具有完备性，解决了无限长度的路径导致深度优先搜索无法停止的问题	限制深度的选取要适合，且面对问题难以确定，该策略同样不具有最优性。	仍然具有DFS的适用性，适合搜索全部的解，因为搜索时不能确保得到最优解，在需要得到全部解时更加适合。
迭代加深搜索	是具有BFS思想的DFS，开始设置深度限制为L = 0，我们迭代地增加深度限制，对于每个深度限制都进行深度受限搜索，因此具有最优性和完备性，结合了DFS和BFS的优点，并利于剪枝。	比较浪费资源，且运行时间较长。	从实际应用来看，迭代加深搜索的效果比较好，并不比广度优先搜索慢很多，但是空间复杂度却与深度优先搜索相同，比广度优先搜索小很多，在一些层次遍历的题目中，迭代加深不失为一种好方法。

策略	优点	缺点	适用场景
双向搜索	从起点和终点同时开始搜索，且是运用BFS交替运行直到相遇，其优点是提高单个bfs的搜索效率，因为搜索深度变成了d/2层，而搜索深度和层数的关系是指数级的关系。且同样具有最优性和完备性。	缺点是需要知道起点和解的位置才可以使用，还需要同时维护两个搜索队列，空间复杂度较高。	适用于解决同时知道起点和终点，求最优路径的问题。

五、创新点&优化

在实验过程中，我发现DFS算法最后得到的路径虽然是盲目搜索的，但它的首选前进方向会受到`next`数组中的顺序的影响（因为循环从0下标开始），因此只要一开始判断起点于终点的大致方位，然后根据其相对方位调整`next`数组中的顺序，便可以缩短深搜得到的路径长度，因此我对程序进行优化，加入了以下代码。

判断起点和终点相对位置，并根据相对位置调整`next`数组元素顺序：

```
index1=endx-startx
index2=endy-starty

next=[[0,-1],[-1,0],[0,1],[1,0]]

if index1>0 and index2>0:
    next=[[0,1],[1,0],[0,-1],[-1,0]]
elif index1>0 and index2<0:
    next=[[1,0],[0,-1],[0,1],[-1,0]]
elif index1<0 and index2<0:
    next=[[0,-1],[-1,0],[0,1],[1,0]]
else:
    next=[[0,1],[-1,0],[0,-1],[1,0]]
```

修改前实验结果：

```
PS E:\VSCODE\py> python -u "e:\VSCODE\py\test2.py"
路径长度为：  174
1111111111111111111111111111111111
1000000000000000000000000000000S1
1011111111111111111111101111111101
1011000100010000001111111000110000001
1011010101010111101111111110110111111
1011010101010000000000000110110000001
1011010101010111100111000011111101
1010010101000100001101111111100000001
1011010101111111101100000000110111111
1011010001100000001111111110110000001
1000011111101111111000000110111111101
1111110000001000000011110110100000001
1000000111111011111100000110111111101
1000000111111011111100000110111111101
1011111100000010000000111110110000001
1000000001111110111111111110110011101
1111111111000000000000000000011111101
1E000000000111111111111111100000000001
1111111111111111111111111111111111111
```

修改后实验结果：

```
PS E:\VSCODE\py> python -u "e:\VSCODE\py\test2.py"
路径长度为：  76
1111111111111111111111111111111111
1000000000000000000000000000000S1
1011111111111111111111101111111101
1011000100010000001111111000110000001
1011010101010111101111111110110111111
1011010101010000000000000110110000001
1011010101010111100111000011111101
1010010101000100001101111111100000001
1011010101111111101100000000110111111
1011010001100000001111111110110000001
1000011111101111111000000110111111101
1111110000001000000011110110100000001
1000000111111011111100000110111111101
1011111100000010000000111110110000001
1000000001111110111111111110110011101
1111111111000000000000000000011111101
1E000000000111111111111111100000000001
1111111111111111111111111111111111111
------------------
PS E:\VSCODE\py>
```

可以看到DFS得到的路径缩短了。

六、实验总结和感想

通过这次实验，我对盲目搜索策略的各种实现方式有了更深的理解，并且能将这些策略进行对比，得出每个策略的特性和适合的应用场景，在面对具体问题时能采取更适合的策略。除此之外，我也看到了盲目搜索的局限性和其优点，这使我受益匪浅。

七、参考资料

- 人工智能第六讲%20正式版.pdf