**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 | **系统结构班** | 专业（方向） | **计算机科学与技术** |
| 学号 | **21307358** | 姓名 | **曾慧蕾** |

# 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

报告要求：报告中①（两个）的原理解释，②需要两种策略的实验需要有对实现的策略效果（四个方面的算法性能）的对比和分析，③以及自己对不同策略优缺点，适用场景的理解和认识 。

而在本次实验的两种类型中，我选择的是①DFS，②一致代价搜索（UCS）

# 实验内容

1. 算法原理

**Dfs（深度优先搜索）：**一种用于遍历或搜索树或图的算法。 沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点v的所在边都己被探寻过或者在搜寻时结点不满足条件，搜索将回溯到发现节点v的那条边的起始节点。若找到了所需结点，则将遍历过的结点置入答案集后依次返回。然而这种算法并不能找到最短的路径，只能找到是否有路径。

**Ucs（一致代价搜索）：**边界中，按路径的成本升序排列，总是扩展成本最低的那条路径，当每种动作的成本是一样的时候，和宽度优先是一样的，所有成本较低的路径都会在成本高的路径之前被扩展，成本小于最优路径的路径数量是有限的。

当最优解的路径长度为d时，宽度优先搜索的时间和空间复杂度都是O（b^（d+1）），对于一致代价搜索，当最优解的成本为C∗ ，则时间和空间复杂

度为O（b^（C\*/s+1））

1. 伪代码

Dfs：

定义答案集ans，遍历集record，读取文件得到地图map

x=startx，y=starty

dfs（x，y）{

若map[x][y]已到达终点：

Return true

否则：

对该格上下左右进行判断，若能走：

fx=next（map[x][y]）[0]

fy=next（map[x][y]）[1]

record[x+fx][y+fy]=1

判断能走的格是否访问过，若否：

dfs（x+fx，y+fy）

若递归返回值为真，将当前结点置入答案集

Dfs算法得到的答案是可行的其中一条路径，不会有多余分支，可以直接输出

Ucs：

与bfs类似，但是改进了原先盲目拓展的算法，改为对启发值最小的结点拓展

定义答案集ans，遍历集record，读取文件得到地图map

建立队列q，

根据起点初始化队列q

Ucs（x，y）{

ox=oy=0 #记录上一个结点，方便后续剪枝

当未到达终点且队列不为空时：

答案集置入x、y、ox、oy #当前结点，当前结点的上一个结点

record[x][y]=-1 #已访问

若当前格不是终点：

Update\_q（x，y） #向四个方向扩展

更新ox，oy为当前节点值

更新x，y为队列第一数据元素的值

q.pop（）

答案集置入x、y、ox、oy #此时已跳出循环，记录终点和终点的上一个结点

}

Update\_q（x，y）{

分别获取四个方向上的偏移量，记为fx fy

若map[x+fx][y+fy]已到达终点：

清空q

将x+fx，y+fy，x，y置入答案集，返回

否则

根据偏移量判断结点是否能扩展，并将相关答案置入答案集中,q的第一个值是当前已走过的路径

对q以第一个节点为标准进行升序排序

}

ucs算法得到的答案是根据给的初始偏移量得到的最短路径，会有多余分支，需要相关的剪枝算法

cut tree（x，y，ans，map）{ #x，y是S的横坐标和纵坐标

令i=ans[-1][2] j=ans[-1][3] #答案节点的上一个结点

While i！=x or j ！=y #当寻上一个结点未寻到起点

P=find（i，j） #在答案集中找到以I j作为起始结点的一项

Maze[i][j]=”\*” #将选择的路径改成星号

I=ans[p][2]

J=ans[p][3]

Return maze

}

1. 关键代码展示（带注释）

根据给定的方向顺序的不同，算法跑出来的路径也会不同，这里统一采用的方向数组是：

forward=[[0,1],[1,0],[-1,0],[0,-1]]

这个顺序可以使实验效果最明显

**Dfs：**

读入地图并对地图进行分割处理：

def read(filename): #读入文件

    content=[]

    with open(filename,'r') as file\_object:

        for line in file\_object.readlines():

            content.append(line)

        return content

def cut(maze): #将迷宫切分成一个个小方块，方便后续路径的替换 尾部的换行符要删掉

    for i in range(0,len(maze)):

        maze[i]=list(maze[i])

        maze[i].pop()

    return maze

功能性函数 得到答案的横坐标和纵坐标

def get\_front(s): return int(s[:s.find(",")])

def get\_behind(s):return int(s[s.find(",")+1:])

判断给定的方格能否拓展

def judge(a,b,maze): #判断某个方向能否前进

    if maze[a][b]==1: return False

    else: return True

dfs算法

def dfs(x,y,maze,record):

    if maze[x][y]=='E':

        return True

    else:

        for i in range(0,4):

            fx=forward[i][0]

            fy=forward[i][1]

            if judge(x+fx,y+fy,maze)==True and (record[x+fx][y+fy]=='0' or record[x+fx][y+fy]=='E'):

                record[x+fx][y+fy]='1'

                if dfs(x+fx,y+fy,maze,record)==True:

                    his=str(x+fx)+","+str(y+fy)

                    ans.append(his)

                    return True

        return False

替换地图中给出的答案路径：

def replace\_maze(maze,history):

    for i in range(0,len(history)):

        maze[get\_front(history[i])][get\_behind(history[i])]="\*"

    return maze

打印函数：

for i in range(0,row):

    for j in range(0,col-1):

        if maze[i][j]=='\*':

            sum+=1

            print("\033[7;37;46m"+'0'+"\033[0m",end='')

        else:

            print(maze[i][j],end='')

    print('1')

print("路径总长度为：",sum)

**Ucs：与dfs算法用到的函数大致相同，不过更新了拓展规则**

读入地图并对地图进行分割处理，将地图中的墙的数字更改为‘-1’：

def read(filename): #读入文件

    content=[]

    with open(filename,'r') as file\_object:

        for line in file\_object.readlines():

            content.append(line)

        return content

def cut(maze): #将迷宫切分成一个个小方块，方便后续路径的替换 尾部的换行符要删掉

    for i in range(0,len(maze)):

        maze[i]=list(maze[i])

        maze[i].pop()

return maze

def trans(maze,row,col): #转换 因为把原图中的0换成了距离，因此可能会有与墙相同的数字 所以把墙换成-1

    for i in range(0,row):

        for j in range(0,col):

            if maze[i][j]=='1':

                maze[i][j]='-1'

    return maze

功能性函数：对队列进行升序排序，查找答案式子

def sort(): #排序 每次选择成本最小的路

    for i in range(0,len(que)-1):

        for j in range(i+1,len(que)):

            if que[j-1][0] > que[j][0]:

                que[j-1],que[j]=que[j],que[j-1]

def find(x,y):

    for i in range(0,len(ans)):

        if ans[i][0]==x and ans[i][1]==y: return i

判断给定的方格能否拓展：

def judge(a,b,maze): #判断某个方向能否前进

    if maze[a][b]==1: return False

    else: return True

每次弹出队列第一个元素后进行更新队列：

def add\_q(x,y,maze,record,sum):

    flag=0

    for i in range(0,4):

        fx=forward[i][0]

        fy=forward[i][1]

        if record[x+fx][y+fy]=='E':

            for j in range(0,len(que)):

                que.pop()

            new\_add=["ans",x+fx,y+fy,x,y]

            que.append(new\_add) #只剩唯一可拓展的

            flag=1

        elif judge(x+fx,y+fy,maze)==True and (record[x+fx][y+fy]=='0'):

            if maze[x][y]=='S':

                # new\_add=[int(maze[x+fx][y+fy]),x+fx,y+fy,x,y] #最后两个x y是扩展点来源

                new\_add=[sum,x+fx,y+fy,x,y] #最后两个x y是扩展点来源

            else:

                # new\_add=[int(maze[x+fx][y+fy])+int(maze[x][y]),x+fx,y+fy,x,y]

                new\_add=[sum,x+fx,y+fy,x,y]

            que.append(new\_add)

    if flag==0: sort()

其中，绿色备注采用的h（n）是曼哈顿函数，不过最后我还是选择了当前走过的路径长作为h（n），这样效果更好。

Ucs算法：

def ucs(x,y,maze,record,):

    ox=oy=0

    sum=0

    while maze[x][y]!='E' and len(que)>0:

        sum+=1

        ans.append([x,y,ox,oy])

        record[x][y]='-1'

        if maze[x][y]!='S':

            add\_q(x,y,maze,record,sum)

        ox=que[0][3]

        oy=que[0][4]

        x=que[0][1]

        y=que[0][2]

        del que[0]

    ans.append([x,y,ox,oy])

替换答案路径的函数，算得上是一种剪枝

def replace\_map(maze,ans,x,y):

    i=ans[-1][2]

    j=ans[-1][3]

    while i!=x or j!=y:

        p=find(i,j)

        maze[i][j]="\*"

        i=ans[p][2]

        j=ans[p][3]

    return maze

打印函数：

for i in range(0,row):

    for j in range(0,col):

        if map[i][j]=='\*':

            sum+=1

            print("\033[7;37;46m"+'0'+"\033[0m",end='')

        else:

            print(map[i][j],end='')

    print('1')

print("路径总长度为：",sum)

1. 创新点&优化（如果有）

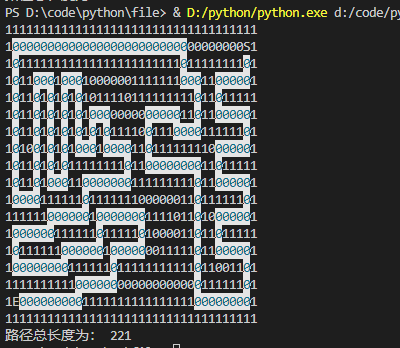
测试了24种方向排列方式最后选择了效果对比最明显的一种。。。

forward=[[0,1],[1,0],[-1,0],[0,-1]] #向左 向右 向上 向下

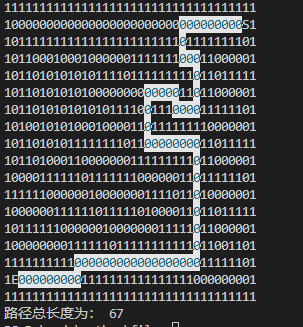
# 实验结果及分析

1.实验结果展示示例（可图可表可文字，尽量可视化）

采用dfs算法后得到的迷宫路径 ，最后总步数为221步，走法非常迷人



采用ucs算法后得到的迷宫路径，最后总步数为67步，变化巨大，可喜可贺



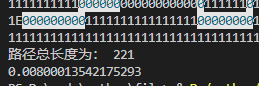
上面两种结果对比可以看出ucs算法得出的结果明显优于dfs的，而且走出来的也是最优路径。

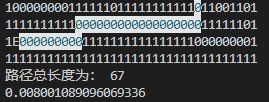
2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

两个算法四个方面的性能评价：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 算法 | 时间复杂度 | 空间复杂度 | 完备性 | 最优性 |
| Dfs | O(b^m) | O(bm) | 否 | 否 |
| Ucs | O(b^(1+C\*/s)) | O(b^(1+C\*/s)) | 是 | 是 |

运行时间：

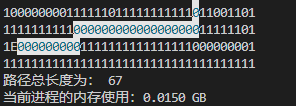
DFS：

UCS：

好像没啥区别都挺快的。。。。ucs稍微慢一点应该是因为剪枝耗时

空间复杂度：

Dfs：

Ucs：

好像也没啥区别，不过ucs内存占的确实比dfs要大，毕竟是指数级增长

最优性：

很明显dfs作为一种盲目搜索算法是不具备最优性的，而ucs由于每次扩展是根据其能扩展的h（n）最小的值而去扩展，所有其最后扩展到的路径一定是最短的

完备性：

Dfs在有无限路径时，如迷宫边缘没封时是可能存在无限搜索的情况的，这种情况下就无法搜索到路径；但ucs有，具体原因与最优性的理由类似。

**|-----------如有优化，请重复1，2，分析优化后的算法结果-----------------------|**

# 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？**

**深度优先搜索：**

**优点：**优先搜索一棵子树，然后是另一棵，所以和广搜对比，有着内存需要相对较少的优点

**缺点：**要多次遍历，搜索所有可能路径，标识做了之后还要取消。在深度很大的情况下效率不高，甚至可能无解。

**适用场景：**要搜索全部的解

**宽度优先搜索：**

**优点：**实现简单，有最优性，能查到的解一定是路径最短的

**缺点：**对空间需求量相对较大，需要维护一个队列

**适用场景：**要搜索最短路径的解

**一致代价搜索：**

**优点：**与bfs类似，是bfs的改进，同样具有最优性和完备性

**缺点：**当路径代价相同时会退化为bfs，内存需求量高

**适用场景：**搜索最短路径的解

**深度受限搜索：**

**优点：**是改进了的dfs，解决了可能存在的无限搜索的问题，当深度大于等于解的深度时可以查到解

**缺点：**深度小于解的深度时无法查找到解，深度难以确定

**适用场景：**搜索全部的解

**迭代加深搜索：**

**优点：**方便剪枝，迭代的增加深度限制可以保证找到最短的路径

**缺点：**不断地循环会造成时间浪费和空间浪费

**适用场景：**很广，在对时空要求不高的场景都可以使用

**双向搜索：**

**优点：**从两个结点开始搜索，减少了需要搜索的深度，且能保证一定能找到最优解

**缺点：**要知道起点和终点的位置，要维护两个队列且要保证搜索并行进行。

**适用场景：**在起点和终点已知的情况下求最优路径。

# 参考资料

**检查内存占用：**[Python模块之psutil详解\_python psutil\_程序员老华的博客-CSDN博客](https://blog.csdn.net/m0_72557783/article/details/128256970)

终端输出调色：[在python终端输出背景颜色字体\_python print 加粗\_IT之一小佬的博客-CSDN博客](https://blog.csdn.net/weixin_44799217/article/details/127541551)

超算习堂-课时7-课时12

六种算法的评价：[经典搜索算法总结 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/187283548?utm_id=0)