人工智能第二次实验 实验报告

实验语言：python

实验环境：pycharm

1. 广度优先求解八数码算法

*# 实现字符串任两个位置的字符交换***def** switchchar(str, p1, p2):  
 left = p1  
 right = p2  
 **if** (p1 > p2):  
 left = p2  
 right = p1  
 **elif** (p1 == p2):  
 **return** str  
 lstr = **""** rstr = **""** mstr = **""  
 if** (left == 0):  
 lstr = **""  
 else**:  
 lstr = str[:left]  
 **if** (right == len(str) - 1):  
 rstr = **""  
 else**:  
 rstr = str[right + 1:]  
 **if** ((right - left) == 1):  
 mstr = **""  
 else**:  
 mstr = str[left + 1: right]  
 **return** lstr + str[right] + mstr + str[left] + rstr  
  
  
*# 用对象表示状态点***class** StateNode:  
 **def** \_\_init\_\_(self, value, dstvalue, step):  
 self.value = value  
 self.index = value.find(**"\*"**)  
 self.assess = 0  
 self.dstvalue = dstvalue  
 **if** (self.index < 0):  
 print(self.tostring() + **" error"**)  
 **raise** RuntimeError  
 **for** i **in** range(len(value)):  
 **if** (value[i] == dstvalue[i]):  
 self.assess += 1  
 self.step = step  
  
 **def** tostring(self):  
 **return "Node:{0} assess={1} step={4}\n {2}\n {3}\n"**.format(self.value[:3], self.assess, self.value[3:6],  
 self.value[6:], self.step)  
  
 **def** left(self):  
 **if** (self.index % 3 != 0):  
 **return** StateNode(switchchar(self.value, self.index - 1, self.index), self.dstvalue, self.step + 1)  
  
 **def** right(self):  
 **if** (self.index % 3 != 2):  
 **return** StateNode(switchchar(self.value, self.index + 1, self.index), self.dstvalue, self.step + 1)  
  
 **def** up(self):  
 **if** (self.index > 2):  
 **return** StateNode(switchchar(self.value, self.index - 3, self.index), self.dstvalue, self.step + 1)  
  
 **def** down(self):  
 **if** (self.index < 6):  
 **return** StateNode(switchchar(self.value, self.index + 3, self.index), self.dstvalue, self.step + 1)  
  
  
**def** calc(startval, endval):  
 startNode = StateNode(startval, endval, 0)  
 openlist = [startNode]  
 closelist = []  
 **while True**:  
 *# 评估，使用assess值* openlist = sorted(openlist, key=**lambda** node: node.assess)  
 curNode = openlist.pop()  
 **if** (curNode == **None**):  
 print(**"End"**)  
 **return** print(curNode.tostring())  
  
 **if** (curNode.assess == 9):  
 print(**"Found,end!"**)  
 **return** leftNode = curNode.left()  
 rightNode = curNode.right()  
 upNode = curNode.up()  
 downNode = curNode.down()  
 **if** (leftNode != **None**):  
 exist = **False  
 for** node **in** openlist:  
 **if** (leftNode.value == node.value):  
 exist = **True  
 break  
 for** node **in** closelist:  
 **if** (leftNode.value == node.value):  
 exist = **True  
 break  
 if** (**not** exist):  
 openlist.append(leftNode)  
  
 **if** (rightNode != **None**):  
 exist = **False  
 for** node **in** openlist:  
 **if** (rightNode.value == node.value):  
 exist = **True  
 break  
 for** node **in** closelist:  
 **if** (rightNode.value == node.value):  
 exist = **True  
 break  
 if** (**not** exist):  
 openlist.append(rightNode)  
  
 **if** (upNode != **None**):  
 exist = **False  
 for** node **in** openlist:  
 **if** (upNode.value == node.value):  
 exist = **True  
 break  
 for** node **in** closelist:  
 **if** (upNode.value == node.value):  
 exist = **True  
 break  
 if** (**not** exist):  
 openlist.append(upNode)  
  
 **if** (downNode != **None**):  
 exist = **False  
 for** node **in** openlist:  
 **if** (downNode.value == node.value):  
 exist = **True  
 break  
 for** node **in** closelist:  
 **if** (downNode.value == node.value):  
 exist = **True  
 break  
 if** (**not** exist):  
 openlist.append(downNode)  
 closelist.append(curNode)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 calc(**"12345678\*"**, **"\*12345678"**)  
*#广度优先 八数码*

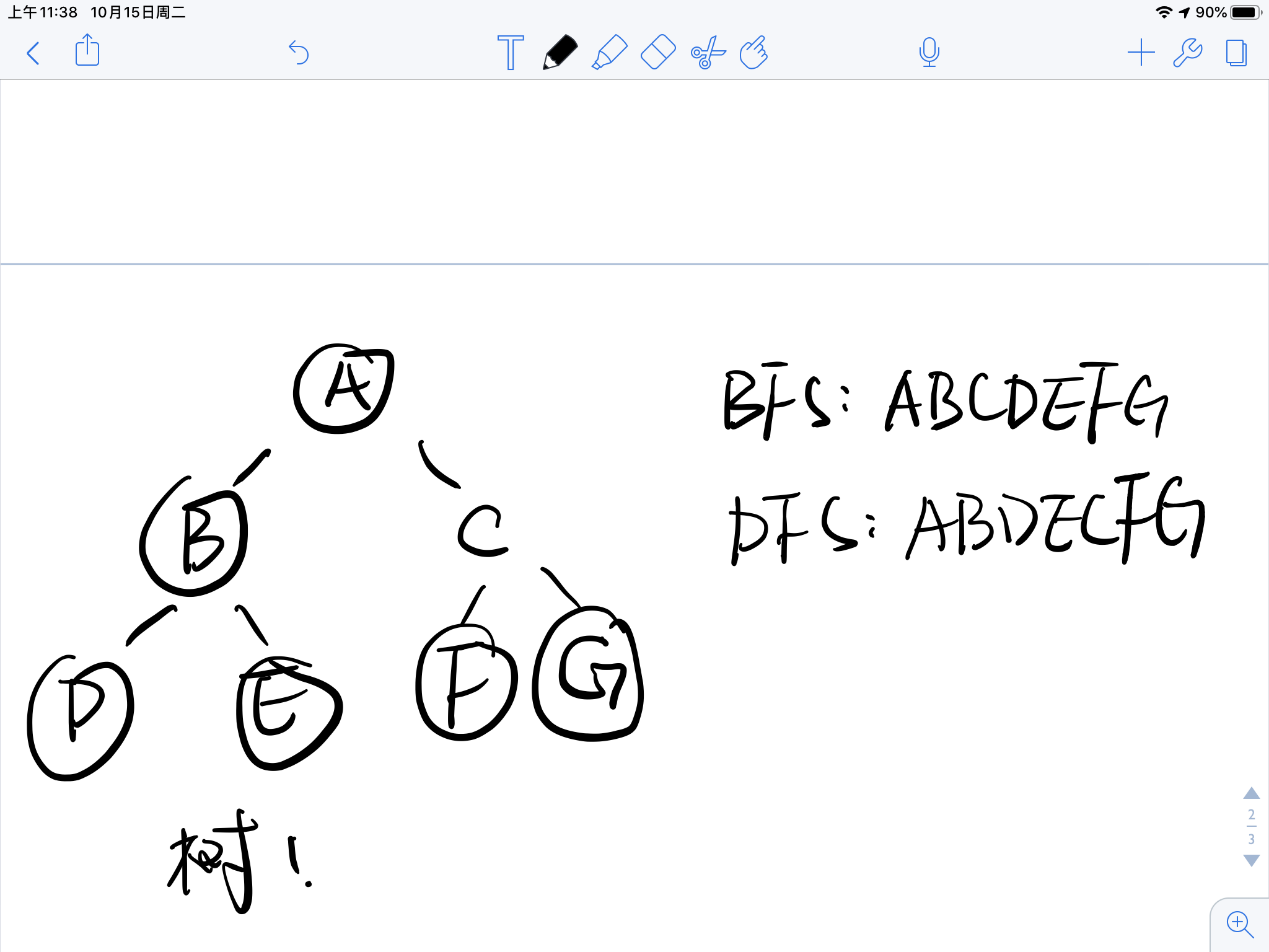
2.

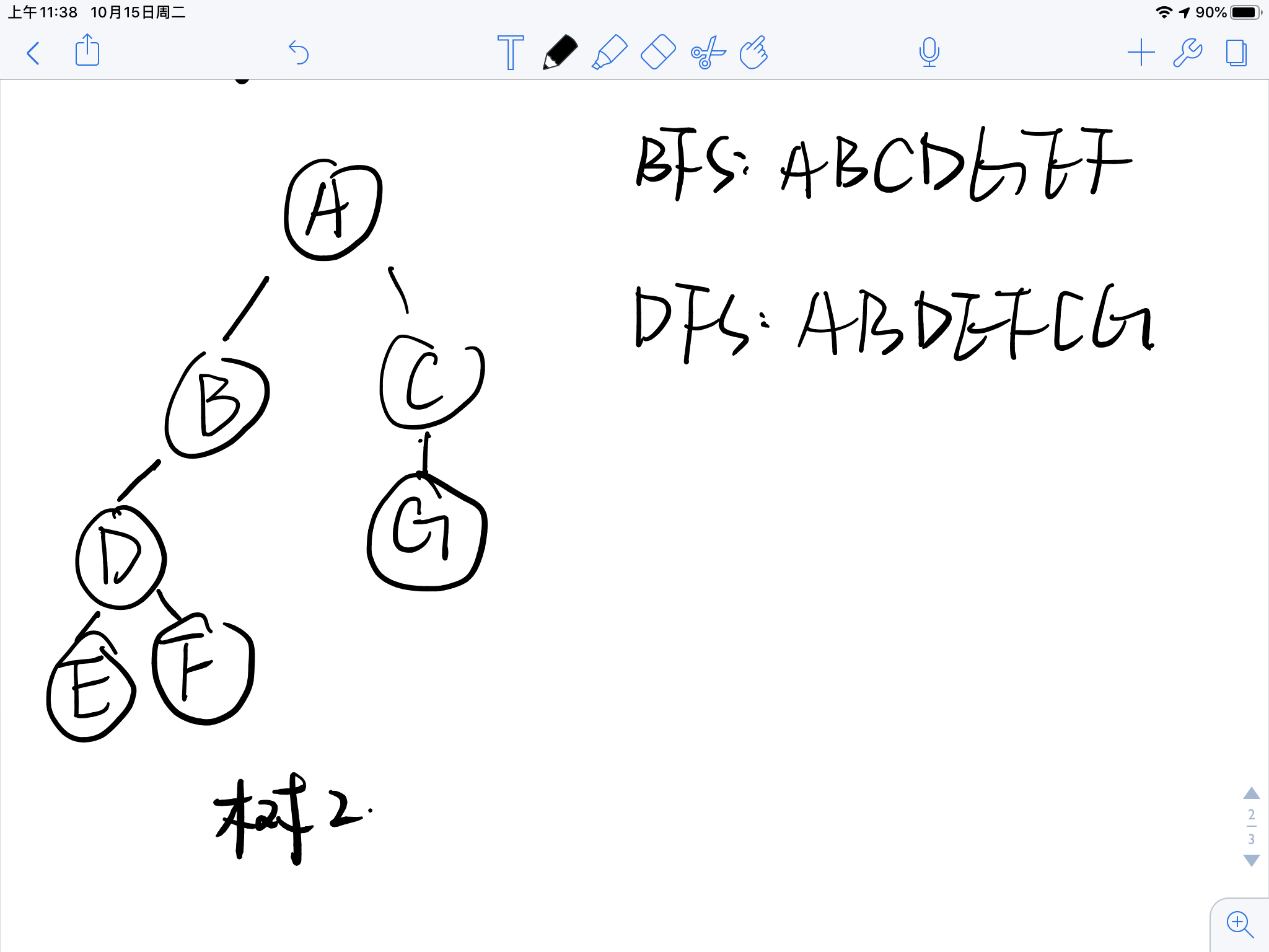
深度优先搜索DFS

广度优先搜索BFS

树搜索：

两次不同的实验：

*# 定义节点类***class** Node(object):  
 **def** \_\_init\_\_(self, val, left=**None**, right=**None**):  
 self.val = val  
 self.left = left  
 self.right = right  
  
  
*# 创建树模型  
#node = Node("A", Node("B", Node("D"), Node("E")), Node("C", Node("F"), Node("G")))*

node = Node(**"A"**, Node(**"B"**, Node(**"D"**,Node(**"E"**), Node(**"F"**))), Node(**"C"**, Node(**"G"**)))  
**def** BFS(root):  
*# 使用列表作为队列* queue = []  
 *# 将首个根节点添加到队列中* queue.append(root)  
 *# 当队列不为空时进行遍历* **while** queue:  
 *# 从队列头部取出一个节点并判断其是否有左右节点  
 # 若有子节点则把对应子节点添加到队列中，且优先判断左节点* temp = queue.pop(0)  
 left = temp.left  
 right = temp.right  
 **if** left:  
 queue.append(left)  
 **if** right:  
 queue.append(right)  
 print(temp.val,end=**" "**)  
  
**def** DFS(root):  
*# 使用列表作为栈* stack = []  
 *# 将首个根节点添加到栈中* stack.append(root)  
 *# 当栈不为空时进行遍历* **while** stack:  
 *# 从栈的末尾弹出一个节点并判断其是否有左右节点  
 # 若有子节点则把对应子节点压入栈中，且优先判断右节点* temp = stack.pop()  
 left = temp.left  
 right = temp.right  
 **if** right:  
 stack.append(right)  
 **if** left:  
 stack.append(left)  
 print(temp.val,end=**" "**)  
  
print(**"BFS"**,end=**" "**)  
BFS(node)  
print(**""**)  
print(**"DFS"**,end=**" "**)  
DFS(node)