山西大学计算机与信息技术学院

**实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓 名 | 李京 | 学 号 | 202202501110 | 专业班级 | 计算机科学与技术2202 | |
| 课程名称 | 人工智能 | | | | 实验日期 | 2024.9.14 |
| 成 绩 |  | 指导教师 | 李琳 | | 批改日期 |  |
| 实验名称 | | 实验一 经典知识表示方法的实现 | | | | |
| 一、实验目的：  •了解已有的经典知识表示方法的基本原理和实现步骤  •掌握状态空间法、问题归约法、谓词逻辑法等知识表示方法   1. 实验内容：   •以猴子摘香蕉问题为例，实现状态空间法  •以梵塔问题为例，实现问题归约法  三、实验平台：  Ubuntu24.04  Microsoft Visual Studio Code  四、实验步骤：  猴子摘香蕉问题：  MonkeyBananaProblem 类:   初始化方法 \_\_init\_\_ 设置了猴子、香蕉和箱子的位置,以及一些状态变量。  四个动作方法:   Monkey\_go\_box(): 猴子移动到箱子位置   Monkey\_move\_box(): 猴子移动箱子到香蕉位置   Monkey\_on\_box(): 猴子爬上箱子   Monkey\_get\_banana(): 猴子获得香蕉  solve() 方法:  按顺序执行必要的动作来解决问题:   如果猴子不在箱子位置,就移动到箱子那里   如果箱子不在香蕉下方,就移动箱子   如果猴子还没有爬上箱子,就爬上去   如果还没有拿到香蕉,就拿到香蕉  主程序:   从输入获取猴子、香蕉和箱子的初始位置   创建 MonkeyBananaProblem 实例   调用 solve() 方法解决问题  class MonkeyBananaProblem:      def \_\_init\_\_(*self*, *monkey*, *banana*, *box*):  *self*.monkey = monkey  *self*.banana = banana  *self*.box = box  *self*.monkey\_on\_box = False  *self*.has\_banana = False  *self*.step = 0      def Monkey\_go\_box(*self*):  *self*.step += 1          print(f"step: {*self*.step} Monkey goes to {*self*.box} from {*self*.monkey}")  *self*.monkey = *self*.box      def Monkey\_move\_box(*self*):  *self*.step += 1          print(f"step: {*self*.step} Monkey catches the box from {*self*.box} to {*self*.banana}")  *self*.box = *self*.banana  *self*.monkey = *self*.banana      def Monkey\_on\_box(*self*):  *self*.step += 1          print(f"step: {*self*.step} Monkey climbs the box.")  *self*.monkey\_on\_box = True      def Monkey\_get\_banana(*self*):  *self*.step += 1          print(f"step: {*self*.step} Monkey gets the banana!")  *self*.has\_banana = True      def solve(*self*):          if *self*.monkey != *self*.box:  *self*.Monkey\_go\_box()          if *self*.box != *self*.banana:  *self*.Monkey\_move\_box()          if not *self*.monkey\_on\_box:  *self*.Monkey\_on\_box()          if not *self*.has\_banana:  *self*.Monkey\_get\_banana()  monkey, banana, box = map(*int*, input().split())  problem = MonkeyBananaProblem(monkey, banana, box)  problem.solve()  8数码问题：  使用deque来实现广度优先搜索（BFS）。solve\_eight\_puzzle函数接受初始状态作为参数。  设置目标状态，可能的移动方向，处理输入，初始化队列和已访问状态集。使用BFS遍历所有可能的状态，直到找到目标状态或遍历完所有可能。找到空格('x')的位置，尝试所有可能的移动，确保移动有效。从用户输入获取初始状态，调用解决函数，并打印结果。  from collections import deque  def solve\_eight\_puzzle(*initial\_state*):      goal\_state = "12345678x"      moves = [1, -1, 3, -3]      initial\_state = ''.join(initial\_state.split())      queue = deque([(initial\_state, 0)])      visited = *set*([initial\_state])      while queue:          state, steps = queue.popleft()          if state == goal\_state:              return steps          x\_pos = state.index('x')          for move in moves:              new\_pos = x\_pos + move              if 0 <= new\_pos < 9 and abs(x\_pos % 3 - new\_pos % 3) <= 1:                  new\_state = *list*(state)                  new\_state[x\_pos], new\_state[new\_pos] = new\_state[new\_pos], new\_state[x\_pos]                  new\_state = ''.join(new\_state)                  if new\_state not in visited:                      queue.append((new\_state, steps + 1))                      visited.add(new\_state)      return -1  initial\_state = input()  steps = solve\_eight\_puzzle(initial\_state)  print(steps)  梵塔问题：  定义了一个名为hanoi的函数,它接受四个参数:   n: 要移动的盘子数量   source: 源柱子   auxiliary: 辅助柱子   target: 目标柱子  函数的实现使用了递归算法:   基本情况: 如果只有一个盘子(n == 1),直接将其从源柱移动到目标柱。   递归情况: 对于n个盘子,步骤如下:  a. 将n-1个盘子从源柱移到辅助柱  b. 将最后一个盘子从源柱移到目标柱  c. 将n-1个盘子从辅助柱移到目标柱   每次移动一个盘子时,程序会打印出移动的过程(如 "A > C")。  主程序部分:   通过input()函数获取用户输入的盘子数量   调用hanoi函数,将盘子从柱子A移动到柱子C,使用B作为辅助柱  def hanoi(*n*, *source*, *auxiliary*, *target*):      if n == 1:          print(f"{source} > {target}")          return      hanoi(n-1, source, target, auxiliary)      print(f"{source} > {target}")      hanoi(n-1, auxiliary, source, target)  n = *int*(input())  hanoi(n, 'A', 'B', 'C')  五、实验结果：        六、实验体会：  在猴子摘香蕉问题中，使用类来表示状态是一个很好的方法。实现过程中，广度优先搜索的实现虽然遇到了一些困难，但最终成功解决了问题。  汉诺塔问题则通过递归来解决，这让我更好地理解了递归的原理和应用。相比猴子摘香蕉，汉诺塔的代码更简洁，但逻辑理解起来稍微复杂一些。 | | | | | | |
| 教 师  评 语 |  | | | | | |