

**人工智能实验报告**

题 目 知识表示

专 业 计算机与电子通信

学　　 号

姓 名

同 组 人 员

1. **背景简介/问题描述**

猴子摘香蕉问题是人工智能领域经典的知识表示与规划问题，旨在考察智能体在受限环境下如何通过一系列动作规划获得目标状态。问题描述如下：房间中有一串挂在天花板上的香蕉，香蕉位置设为C；地面有一只可移动的箱子，初始位置为B；猴子可在房间内任意移动、推动箱子并攀登箱子。只有当猴子攀登到箱子上时才能够够到香蕉。现给定初始状态（猴子在位置B，箱子在位置B，香蕉在位置C），请设计并实现知识表示与搜索算法，推导出猴子摘取香蕉的动作序列。

1. **算法介绍**

**2.1 知识表示——一阶谓词逻辑**

**2.1.1.谓词定义：**

SITE(x, y)：对象x位于位置y。本实验中，x∈{Monkey, Box}，y∈{a, b, c}。

HANG(w, y)：物体w悬挂在位置y。本实验中，w=Banana，y∈{a, b, c}。

ON(z)：表示猴子z正在箱子上。z=Monkey。

HOLDS(z)：表示猴子z手中持有香蕉。z=Monkey。

**2.1.2.状态表示：**

系统状态由上述谓词的集合构成。例如，{SITE(Monkey,b), SITE(Box,b), HANG(Banana,c)}表示猴子和箱子在b点，香蕉c点悬挂；ON(Monkey)和HOLDS(Monkey)则分别描述攀爬和拿到香蕉两个关键状态。

状态集使用Python的set存储，便于增删和判等操作。

**2.2 动作定义——前提条件与效应**

将环境中可执行的动作形式化为经典的“前提-删除-添加”三元组：

|  |  |  |  |
| --- | --- | --- | --- |
| 动作 | 前提条件 | 删除列表 | 添加列表 |
| Goto(u, v) | SITE(Monkey,u)∈state,且ON(Monkey)∉state | SITE(Monkey,u) | SITE(Monkey,v) |
| Pushbox(v, w) | SITE(Monkey,v)∈state∧SITE(Box, v)∈state,且ON(Monkey) ∉ state | SITE(Monkey, v), SITE(Box, v) | SITE(Monkey, w), SITE(Box, w) |
| Climbbox | 存在x使SITE(Monkey, x) ∈ state ∧ SITE(Box, x) ∈ state, 且 ON(Monkey) ∉ state | (无) | ON(Monkey) |
| Grasp | ON(Monkey) ∈ state ∧ 存在y使 SITE(Monkey, y) ∈ state ∧ HANG(Banana, y) ∈ state | HANG(Banana, y) | HOLDS(Monkey) |

前提条件：确保动作逻辑正确，如只有猴子与箱子同处一地且未在箱子上时才能Pushbox。

删除列表：删除先前状态中的谓词，模拟物理变化。

添加列表：将新谓词加入，更新新状态。

**2.3 搜索算法——广度优先搜索**

状态生成：函数successors(state)枚举所有可行动作，对每个满足前提的动作调用对应函数（Goto、Pushbox、Climbbox、Grasp），获取新状态。

去重与队列：使用frozenset(state)作为状态哈希标识，存入visited集合避免重复访问。维护一个FIFO队列，每个元素为 (当前状态, 执行动作序列)。

最短路径保证：BFS按层级展开状态空间，一旦发现满足目标 HOLDS(Monkey) 的状态，即返回当前动作序列，保证最少步数解。

1. **算法实现**

**3.1 环境与依赖**

开发语言：Python 3.12.7

第三方库：collections（deque 用于队列实现）、copy（用于状态深拷贝）

运行平台：不限，推荐在支持标准库的任何操作系统下运行。

**3.2 代码结构**

|  |
| --- |
| monkey\_banana/ # 项目根目录  ├── predicates.py # 定义 SITE, HANG, ON, HOLDS 四类谓词  ├── actions.py # 实现 Goto, Pushbox, Climbbox, Grasp 四个动作函数  ├── search.py # successors, is\_goal, bfs 搜索算法主逻辑  └── main.py # 程序入口，初始化状态并调用 bfs |

predicates.py：每个谓词定义为一个 Python 类，实现 \_\_hash\_\_, \_\_eq\_\_, \_\_repr\_\_，以支持集合存储和可读性输出。

actions.py：每个动作函数接受当前状态集合 state，根据前提判断生成新状态或返回 None。

search.py：successors(state)调用四个动作函数，收集所有非空后继状态；is\_goal(state)判断 HOLDS(Monkey) 是否在 state 中；bfs(initial\_state)标准广度优先搜索实现，返回动作描述列表。

main.py：组装上述模块，打印初始状态和求解结果。

**3.3 关键函数与实现细节**

**3.3.1 谓词类定义（predicates.py）**

|  |
| --- |
| class SITE:  def \_\_init\_\_(self, x, y):  self.x, self.y = x, y  def \_\_hash\_\_(self): return hash(("SITE", self.x, self.y))  def \_\_eq\_\_(self, other): return isinstance(other, SITE) and ...  def \_\_repr\_\_(self): return f"SITE({self.x}, {self.y})"  # HANG, ON, HOLDS 定义类似 |

通过哈希与相等比较支持在 set 中增删。

**3.3.2 动作函数实现（actions.py）**

|  |
| --- |
| def Goto(state, u, v):  if SITE("Monkey", u) in state and ON("Monkey") not in state:  new = state.copy()  new.remove(SITE("Monkey", u))  new.add(SITE("Monkey", v))  return new  return None |

Pushbox, Climbbox, Grasp 按表格中的前提-删除-添加逻辑实现。

**3.3.3 搜索逻辑（search.py）**

|  |
| --- |
| from collections import deque  def bfs(initial):  queue = deque([(initial, [])])  visited = set()  while queue:  state, plan = queue.popleft()  sid = frozenset(state)  if sid in visited: continue  visited.add(sid)  if is\_goal(state): return plan  for desc, nxt in successors(state):  queue.append((nxt, plan + [desc]))  return None |

去重：visited 存储 frozenset(state)，防止重复遍历。

返回值：最先找到目标即停止搜索，并返回对应动作列表。

**3.4 实验结果**

运行主函数后，初始状态为 {SITE(Monkey, a), SITE(Box, b), HANG(Banana, c)}，程序输出动作序列如下：

|  |
| --- |
| Found plan:  Goto(a -> b)  Pushbox(b -> c)  Climbbox  Grasp |

其中，操作含义依次为：

猴子从 a 点移动到 b 点；

猴子推动箱子b点到c点；

猴子登上箱子；

猴子摘取香蕉。

实验结果验证了算法的正确性与可行性。

1. **讨论及结论**

**4.1 讨论**

状态空间规模：随着位置与对象数量增加，状态空间呈指数增长，可考虑启发式搜索（如A\*）或规划语言（如PDDL）来提升效率。

知识表示扩展：可引入框架系统或语义网络对对象属性和关系进行更丰富的表示，支持更复杂场景。

可再现性与模块化：建议将谓词和动作抽象为通用组件，便于在其他规则推理任务中复用。

**4.2 结论**

通过一阶谓词逻辑结合BFS算法，能够清晰地刻画状态和动作，成功解决猴子摘香蕉问题。该方法在教学与小规模规划任务中效果显著，但对于大规模或动态环境仍需引入启发式和更灵活的知识表示方法。