



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

(深圳)

# 实验报告

开课学期: 大三上学期  
课程名称: 人工智能  
实验名称: 知识表示  
实验性质: 设计  
实验时间: 2019.10 地点: T2109 实验台号  
学生专业: 计算机  
学生学号: SZ170110320  
学生姓名: 韩雪婷  
评阅教师:  
报告成绩:

实验与创新实践教育中心印制

2019 年 10 月

实验报告至少包括：

一. 简介/问题描述

1.1 待解决问题的解释

1.2 问题的形式化描述

1.3 解决方案介绍（原理）

二. 算法介绍

2.1 所用方法的一般介绍

2.2 算法伪代码

三. 算法实现

3.1 实验环境与问题规模

3.2 数据结构

3.3 实验结果

3.4 系统中间及最终输出结果（要求有屏幕显示）

四. 总结及讨论（对该实验的总结以及任何该实验的启发），

参考文献

附录一源代码及其注释（纸质版不需要打印）

小组成员贡献量：各 25%

# 1 问题描述

## 1.1 待解决问题的解释

猴子摘香蕉问题：

一个房间里，天花板上挂有一串香蕉，有一只猴子可在房间里任意活动（到处走动，推移箱子，攀登箱子等）。设房间里还有一只可被猴子移动的箱子，且猴子登上箱子时才能摘到香蕉，问猴子在某一状态下（设猴子位置为 A，箱子位置为 B，香蕉位置在 C），如何行动可摘取到香蕉。问题初始状态如图 1-1-1 所示。

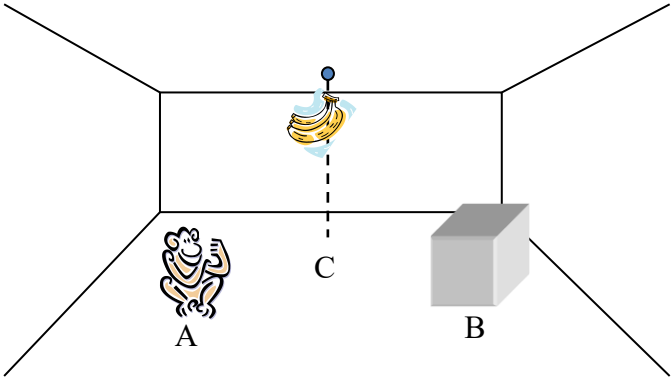


图 1-1-1

## 1.2 问题的形式化描述

[1]. 综合数据库

综合数据库由一个四元组(monkey, box, on, banana)组成，用于表示各种状态，如表 1-2-1 所示。

表 1-2-1

符号	表示	取值范围
monkey	猴子的位置	{A, B, C}
box	箱子的位置	{A, B, C}
on	是否站在箱子上	{0, 1}
banana	是否得到香蕉	{0, 1}

其中，on = 1 表示站在箱子上；banana = 1 表示得到香蕉。

[2]. 初始状态

(A, B, 0, 0)

[3]. 目标状态

(C, C, 1, 1)

[4]. 规则集

R1: IF(x, y, 0, 0)THEN(w, y, 0, 0)

Move 猴子移动

前件要求: on = 0 猴子没有站在箱子上

R2: IF(x, x, 0, 0)THEN(z, z, 0, 0)

Push 猴子推箱子

前件要求:

state.monkey.equals(state.box)&&state.on==0&&state.getbanana==0

猴子站的位置与箱子相同; 猴子不站在箱子上, 并且猴子手上没拿着东西。

R3: IF(x, x, 0, 0)THEN(x, x, 1, 0)

Climb 猴子爬箱子

前件要求:

state.monkey.equals(state.box)&&state.on==0

猴子站的位置与箱子相同; 猴子不站在箱子上。

R4: IF(x, x, 1, 0)&&(x = C)THEN(x, x, 1, 1)

Get 猴子摘香蕉

前件要求:

state.box=='C'&&state.monkey=='C'&&state.on==1&&this.flag==false

猴子与箱子都在香蕉在的地方; 猴子站在箱子上。

其中, x, y, z, w 为变量; C 为常量, 表示位置 C。

### 1.3 解决方案介绍 (原理)

采用产生式知识表示的方式来解决此问题, 下面简要介绍产生式表示系统。

[1]. 产生式表示方法

- 事实

对于确定性知识, 有(对象, 属性, 值) 和(关系, 对象 1, 对象 2)两种表示方法。

- 规则

规则也称为产生式, 其基本形式为: IF P THEN Q

其中 P 是产生式的前提, 也称为前件, 它给出了该产生式可否使用的先决条件。Q 是一组结论或操作, 也称为后件, 它指出当 P 满足时, 应该推出的结论或应该执行的动作。

[2]. 产生式系统的基本结构

- 综合数据库

用于存放推理过程的各种当前信息；作为推理过程选择可用规则的依据。

- 规则库

用于存放推理所需要的所有规则，是整个产生式系统的知识集。是产生式系统能够进行推理的根本。

- 控制系统

亦称推理机，用于控制整个产生式系统的运行，决定问题求解过程的推理线路。

### [3]. 产生式系统的运行

如图 1-3-1 所示。

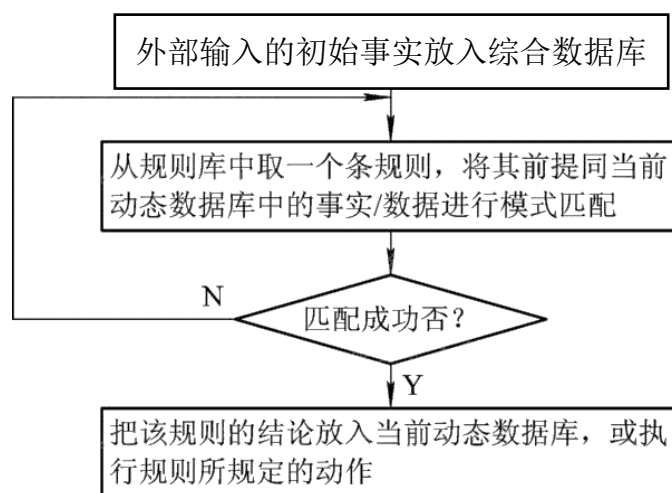


图 1-3-1

## 2 算法介绍

### 2.1 所用方法的一般介绍

Step1 将初始状态压入队列。

Step2 从队列中弹出一个状态。

Step3 将该状态可到达的状态压入队列中，记录是通过什么操作到达该状态。

Step4 重复步骤 2 和 3，直到步骤 2 弹出的状态是目标状态为止。

Step5 打印操作路线。

### 2.2 算法伪代码

#将初始状态压入队列

EntryQueue(state[0])

#从队列中弹出一个状态，检查弹出的状态是否是目标状态

```

While((state[i] = DeleteQueue) != GoalState)
    Nstates = PossibleStates(state[i])    #将该状态可到达的状态压入队列中
    Move[i] = Operation(state[i],Nstates)    #记录是通过什么操作到达该状态
Print(Move)    #打印操作路线

```

### 3 算法实现

#### 3.1 实验环境与问题规模

硬件环境：PC 机

软件环境：java

问题规模：总共可能的状态共 36 种。

#### 3.2 数据结构

栈：java.util.Stack 打印操作路线时

队列：java.util.Queue 寻找解决方案时

#### 3.3 实验结果

如图 3-3-1 所示。

```

start
当前状态:State{monkey=A, box=B, on=0, getbanana=0}
Move from A to B
当前状态:State{monkey=B, box=B, on=0, getbanana=0}
pushBox fromBto C
当前状态:State{monkey=C, box=C, on=0, getbanana=0}
Climb
当前状态:State{monkey=C, box=C, on=1, getbanana=0}
Get
当前状态:State{monkey=C, box=C, on=1, getbanana=1}
成功构建 (总时间: 0 秒)

```

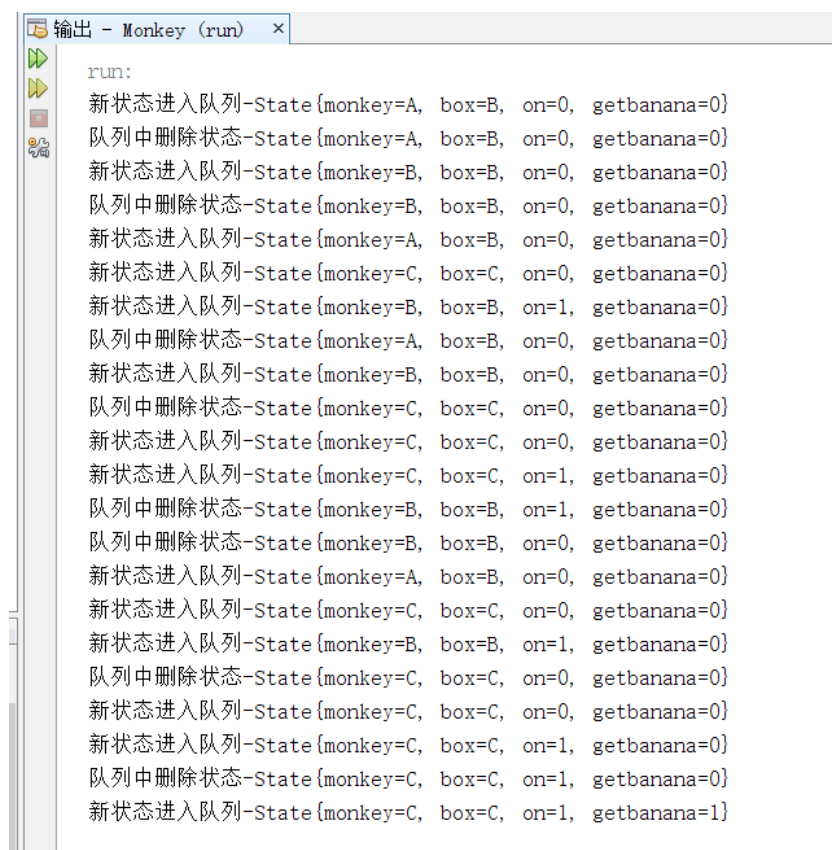
图 3-3-1

即状态转换为：

(A, B ,0 ,0)→ Move from A to B 猴子从 A 到 B  
 (B, B, 0, 0)→ PushBox from B to C 猴子推箱子从 B 到 C  
 (C, C, 0, 0)→ Climb 猴子爬箱子  
 (C, C, 1, 0)→ Get 猴子拿香蕉  
 (C, C, 1, 1)

### 3.4 系统中间及最终输出结果

中间输出如图 3-4-1.



```
run:
新状态进入队列-State {monkey=A, box=B, on=0, getbanana=0}
队列中删除状态-State {monkey=A, box=B, on=0, getbanana=0}
新状态进入队列-State {monkey=B, box=B, on=0, getbanana=0}
队列中删除状态-State {monkey=B, box=B, on=0, getbanana=0}
新状态进入队列-State {monkey=A, box=B, on=0, getbanana=0}
新状态进入队列-State {monkey=C, box=C, on=0, getbanana=0}
新状态进入队列-State {monkey=B, box=B, on=1, getbanana=0}
队列中删除状态-State {monkey=A, box=B, on=0, getbanana=0}
新状态进入队列-State {monkey=B, box=B, on=0, getbanana=0}
队列中删除状态-State {monkey=C, box=C, on=0, getbanana=0}
新状态进入队列-State {monkey=C, box=C, on=0, getbanana=0}
新状态进入队列-State {monkey=C, box=C, on=1, getbanana=0}
队列中删除状态-State {monkey=B, box=B, on=1, getbanana=0}
队列中删除状态-State {monkey=B, box=B, on=0, getbanana=0}
新状态进入队列-State {monkey=A, box=B, on=0, getbanana=0}
新状态进入队列-State {monkey=C, box=C, on=0, getbanana=0}
新状态进入队列-State {monkey=B, box=B, on=1, getbanana=0}
队列中删除状态-State {monkey=C, box=C, on=0, getbanana=0}
新状态进入队列-State {monkey=C, box=C, on=0, getbanana=0}
新状态进入队列-State {monkey=C, box=C, on=1, getbanana=0}
队列中删除状态-State {monkey=C, box=C, on=1, getbanana=0}
新状态进入队列-State {monkey=C, box=C, on=1, getbanana=1}
```

图 3-4-1

最终输入结果如图 3-3-1 所示。

## 4 总结及讨论

在课堂上学习知识表示时，往往是一眼看出解决方法，然后想着解决方法去写数据库和知识库等。在电脑上跑程序的时候，因为打印了中间的操作队列，可以看到电脑运行程序的时候看上去很傻，但是它可以按照一定规律遍历各种可能的操作序列，这里使用的是一种宽度优先遍历的思想，当然可以改进为其它搜索策略，让电脑更快地找到解决方法。猴子摘香蕉问题的规模较小，我们可以在更大规模的问题上研究这一部分。

当然这个实验的重点在于知识表示，将文字型的题目转化为综合数据库，让电脑可以理解问题并解决问题，除了这里使用的产生式表示法，还有许多其它方法，也可以实现这个目标。

## 附录

### 代码

```
package monkey;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

/**
 *
 * @author admin
 */
public class Monkey {

    class State{
        Character monkey; //猴子的位置
        Character box;    //盒子的位置
        int on; //是否在盒子上
        int getbanana; //是否摘到香蕉
        int no;      //当前状态的序号
        String op; //操作

        public State(char monkey, char box, int onbox, int banana, int no, String
op) {

            this.monkey = monkey;
            this.box = box;
            this.on = onbox;
            this.getbanana = banana;
            this.no = no;
            this.op = op;          //进行了什么操作
        }
        public State(){
```



```

    }

    @Override
    public String toString() {
        return "State{" + "monkey=" + monkey + ", box=" + box + ", on="
+ on + ", getbanana=" + getbanana+"}";
    }

}

boolean flag = false;    //手中无香蕉
private int num = 0;      //用来给一个状态编序号
int[] record  = new int[100]; //用来记录某个状态的上一个状态，方便回
溯找到到这一步的路径
Queue<State> queue = new LinkedList();
ArrayList<State> StateList = new ArrayList<>();//用来保存所有尝试过的状
态

/**
 * 进行记录
 */
public void Record(State next,int pre){
    this.StateList.add(this.num, next);
    this.record[this.num] = pre; //记录路径， state.no 为上一个状态
    this.queue.add(next);       //将该状态加入队列
    System.out.println("新状态进入队列-"+next.toString());
}

/**
 * 猴子向某一处移动
 * @param state
 */

public void Move(State state){
    State next  = null;

```

```

        if(state.on == 0){
            if(state.monkey == 'A'){
                next = new State('B',state.box,0,0,++num,"Move from A to B");
//新状态猴子从 A 走向 B

            }
            else if(state.monkey == 'B'){
                next = new State('A',state.box,0,0,++num,"Move from B to A");
//新状态猴子从 B 走向 A
            }
        }
        if(next!=null){
            Record(next,state.no);
        }
    }
    /**
     * 推箱子
     * @param state
     */
    public void push(State state){
        //推箱子的条件：猴子与盒子在同一个地方,手上没有香蕉，猴子没
        站在箱子上
        State next = null;

        if(state.monkey.equals(state.box)&&state.on==0&&state.getbanana==0){
            //只要能推盒子在任意地方都推向 C
            next = new State('C','C',0,0,++num,"pushBox
            from"+state.box.toString()+"to C");
        }
        if (next!=null){
            Record(next, state.no);
        }
    }
    /**
     * 是否能爬上箱子

```

```

    * @param state
    */
    public void Climb(State state){
        State next = null;
        //猴子和箱子在同一个地方，并且猴子不站在箱子上
        if(state.monkey.equals(state.box)&&state.on==0){
            next = new State(state.monkey,state.box,1,0,++num,"Climb");
        }
        if (next!=null){
            Record(next, state.no);
        }
    }
    /**
     * 猴子得到香蕉
     * @param state
     */
    public void Get(State state){
        State next = null;
        //猴子在 C，箱子在 C，猴子站在箱子上，猴子还没摘到香蕉

        if(state.box=='C'&&state.monkey=='C'&&state.on==1&&this.flag==false){
            next = new State('C','C',1,1,++num,"Get");
            this.flag = true;
        }
        if (next!=null){
            Record(next, state.no);
        }
    }
    /**
     * 广度优先搜索，寻找解决方案
     */
    public void findSolution(){
        State start = new State('A','B',0,0,num,"start");
        this.StateList.add(0,start);
        this.queue.add(start);
    }

```

```

System.out.println("新状态进入队列-"+start.toString());
while(!queue.isEmpty()){
    State now = queue.poll();
    System.out.println("队列中删除状态-"+now.toString());
    Move(now);
    push(now);
    Climb(now);
    Get(now);
    if(this.flag){
        break;
    }
}
if(this.flag==false){
    System.out.println("impossible");
    return;
}

```

```

Stack<Integer> stack = new Stack();
int now = this.num;        //结束状态的序号
stack.push(now);
while(now!=0){
    int pre = record[now];
    stack.push(pre);
    now = pre;
}

```

```

System.out.println("\n\n");
while(!stack.isEmpty()){
    now = stack.pop();
    State nowState = this.StateList.get(now);
    System.out.println(nowState.op);
    System.out.println("当前状态:"+nowState.toString());
}

```

```
}  
    public static void main(String[] args) {  
        Monkey test = new Monkey();  
        test.findSolution();  
    }  
}
```