哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

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 无

学生专业：计算机类

学生学号：SZ170110301

学生姓名：卢茉莉

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

一、 实验概述

1. 问题描述

本实验考察课程讲授的知识表示方法。利用知识表示方法，将所给的猴子摘香蕉的问题转化为更适用于逻辑推理的形式，通过设计状态和规则并进而将其转换为计算机能理解并自动进行处理的可执行程序，来得出待求解问题的最优路径（即达到目标状态的解决方案）

2. 问题的形式化描述

（1） 状态定义：

使用五元组$(monkey, banana, box, on, getbanana)$来定义问题的状态。

$monkey$：猴子的位置，$monkey \in \{A, B, C\}$

$banana$：香蕉的位置，$banana \in \{C\}$

box：箱子的位置，$box \in \{A, B, C\}$

on：猴子是否站在箱子上，$on \in \{0,1\}$

$getbanana$：猴子是否拿到香蕉，$getbanana \in \{0,1\}$

（2） 产生式规则：

$Move: if(x, y, z, 0, 0) then(w, y, z, 0, 0)$

$Push: if(x, y, x, 0, 0) then(z, y, z, 0, 0)$

$C\lim b: if(x, y, x, 0, 0) then(x, y, x, 1, 0)$

$down: if(x, y, x, 1, 0) then(x, y, x, 0, 0)$

$Get: if(x, x, x, 1, 0) then(x, x, x, 1, 1)$

（3） 初始状态：$(A,C,B,0,0)$

（4） 目标状态：$(C,C,C,1,1)$

3. 实验原理

本次实验利用了产生式知识表示方法，通过将题目的具体内容转化成为简化的符号表示，并且根据要求和相关的条件限制设计了产生式规则，实现了从自然语言到可以设计成为具体机器程序的铺垫工作，最后这个问题就转化为从开始状态到目标状态的路径搜索问题。

在本实验中，我们选择了广度优先算法来进行路径的搜索。作为完备的搜索策略，广度优先算法一定能返回一条从开始节点到目标节点的最短路径，而这一结果就是解决题目问题的操作序列。

二、 算法介绍

1. 方法介绍

在本实验的具体实现中，由于香蕉的位置从始至终都是保持不变的，因此实际上在程序中我们使用的是状态的四元组而非五元组。按照前述的实验原理，我们首先对于状态表示的数据结构和产生式规则（具体动作）进行定义，并且设定好起始状态和目标状态的表示用于进行目标结果的判断。在应用广度优先算法进行路径的搜索时，队列中的每个节点记

录猴子的一个状态，判断状态是否符合条件，从而通过产生式规则来得到猴子下一个正确且可能的状态，成为原节点的后继节点压入队列中，重复上述操作直到找到目标状态为止，根据广度优先算法的特性，从根节点到目标节点的路径必然是最短路径。

2. 伪代码

（1）根据问题分析定义状态表示和产生式规则对应的方法
（2）初始状态入队列
（3）当队列不空时：
（4）{
（5） 弹出一个节点
（6） 对四个规则进行尝试，符合条件的则将下一个状态 next 入队列，并在 record 中记录 next 的先驱节点是当前节点 pre
（7） 如果到达目标节点：
（8） Break
（9）}
（10）If 未找到路径：
（11） 打印错误信息
（12）将当前节点编号入栈
（13）While 当前节点不是起始节点：
（14）{
（15） 通过 record 表找到当前节点的前驱节点
（16） 将前驱节点编号入栈
（17）}
（18）While 栈不空时：
（19）{
（20） 将节点编号出栈
（21） 打印节点（状态信息）
（22）}

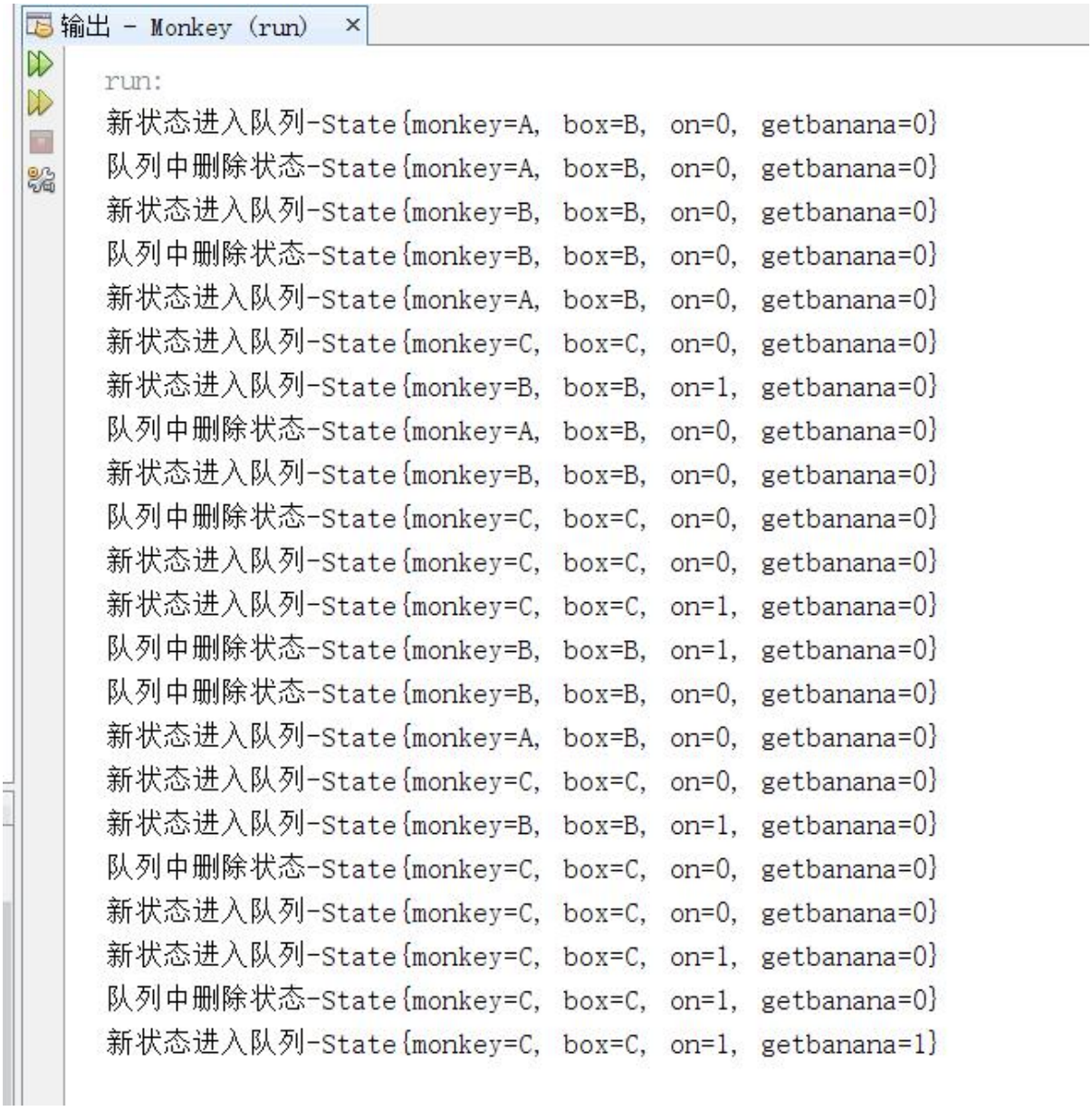
三、 算法实现

1. 实验环境：Java

2. 问题规模：本问题的解空间为以起始节点为根节点、通过产生式规则来产生后继节点的搜索树，假设所有合法的状态数为n，则问题规模大约为$O(n^2)$

3. 数据结构：本实验中对于解决方案的搜索使用的是广度优先算法，因此主要涉及的数据结构是队列，另外还采用了数组来存储节点的先驱节点的信息。

4. 实验结果：以下结果截图中，图一表示的是在路径搜索过程中队列节点进出的动态过程展示，结合本实验中设计的具体规则可以看出，每一轮进入队列的状态确实是由产生式规则得到的合法的状态；图二表示的是经过广度优先算法得到的最佳实现路径，包括路径上的每一个状态和状态转换需要的具体动作（产生式规则）。




```
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)
```

四、 总结及讨论

问题的知识表示是利用计算机辅助我们处理问题的一个新思路，目的在于为这些自然语言描述的语言找到形式化的表述，从而找到一系列相似问题的通用解法。在本实验中涉及的算法也可以适用于其他有特定模式、操作方法固定并且可以对问题进行形式化描述的问题中。在搜索路径使用的策略上同样可以灵活变通，根据具体的对时间、空间的要求来决定采用的搜索策略，如深度优先、广度优先、A*算法等等。而在本问题中，需要找到的是最简便、步骤最少的路径，于是选择了适用于不考虑路径代价的搜索树的广度优先算法。经过验证可以知道，这段程序找到的操作序列与实际分析得到的结果相同，是可以用于解决此问题的通用算法。

五、 组员贡献度

韩雪婷：25%　陈抒语：25%　廖思瑀：25%　卢茉莉：25%

附（源代码截图）：

```
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;
        }
        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();
    }
}
```