**《人工智能原理与方法》**

**实验报告**

**实验一：二阶&三阶汉诺塔问题的求解**

**（深度&广度优先搜索）**

**专业班级：12级计算机二班**

**学号：1240069 姓名：李伟康**

**实验地点： 理工楼913**

**实验时间：2014年5月8日**

实验目的：

1．加深对深度&广度优先算法的理解，并掌握其在具体问题中的应用。

2．复习《数据结构》课程的相关知识，学会根据具体问题设计可解的数据结构。

3. 复习C语言的相关知识，提升自己的编程能力以及解决具体问题的能力。

实验语言环境：

1. VC++6.0（英文版）
2. 通过C语言实现

汉诺塔问题的求解：

由于深度优先搜索和广度优先搜索只是扩展节点进入OPEN表的顺序不一样（深度优先搜索OPEN表采用栈，扩展的节点放在表头；广度优先搜索OPEN表采用队列，扩展的节点放在表尾），为了使得叙述不太冗余，我将重点以深度优先算法为例进行说明。

1. **深度优先算法说明**

(1) 初始结点S放入OPEN中；

(2) 若OPEN为空，则搜索失败，问题无解；

(3) OPEN表中最顶端结点放到CLOSED表中，并给出顺序编号n；

(4) 若n为目标结点D，则搜索成功，问题有解；

(5) 若n无子结点，转(2)；

(6) 扩展n结点，将其所有子结点配上返回n的指针，并按次入OPEN首部，转(2) 。

1. **数据结构设计与分析**

**(1) 算法实现所需数据结构分析**

由于深度优先算法中每次都要取OPEN表的顶端节点，结合《数据结构》的相关知识，此处利用栈来实现OPEN表的构建。CLOSED表仅用于记录已扩展节点的顺序（包含将要扩展的节点），其数据结构设计利用队列或栈均可，此处采用栈这一数据结构来实现CLOSED表的构建。

1. **数据结构具体设计**

#define maxsize 1000//栈长度

#define Maxsize 3//汉诺塔问题的阶数

//中间结构体定义

typedef struct{

int data\_s[Maxsize];

int data\_f[Maxsize];

}datatype;

//顺序栈定义

typedef struct{

datatype data[maxsize+1];//栈数组

int top; //栈顶指针

}Seqstack;

OPEN表和CLOSED表均用此栈进行构建。下面以OPEN表为例说明此顺序栈的具体含义：

顺序栈中的data是一个一维数组，每一个值表示OPEN表的一个节点。data数组的数据类型为datatype，该结构体用来存储每一个结点中的子节点和父节点。子节点或父节点均为一个一维数组，数组的下标表示了盘的优先级（即小、中、大盘），数组的值则记录了当前状态的具体数值。为了便于理解此处设计的栈，现举具体例子进行说明：假设OPEN表第二个元素为

|  |  |
| --- | --- |
| 11（父节点） | 12（子节点） |

具体表示为：

Open\_head->data[2].data\_f[0]=1;Open\_head->data[2].data\_f[1]=1;

Open\_head->data[2].data\_s[0]=1;Open\_head->data[2].data\_s[1]=2;

(对于CLOSED表而言，data数组的下标用来表示扩展节点的先后顺序，1表示最早扩展)

1. **算法实现设计与分析**
2. **深度优先算法实现分析**

解决方案的设计目的是模拟人脑使用算法的具体过程。因此，除了定义“静态”数据结构外，还需要定义恰当的“动态”函数。

1. **解决方案所用函数一览**

根据函数在解决方案中的重要性，将其划分为附属函数、基本函数、主要函数，每种函数种类及详细说明如下：

**A.附属函数构建**

a.void Show\_table(datatype value)//输出栈内元素

将当前扩展节点以适当格式输出。

b.bool select()//小菜单函数构建

程序执行结束后给用户的选择功能。用户输入Y，程序继续执行；用户输入N，程序终止执行。

**B.基本函数构建**

1. Seqstack \*InitStack()//初始化栈

据此函数构建OPEN表和CLOSED表，构建后的OPEN表和CLOSED表为空。

b.void Stackempty(Seqstack \*S)//判断栈是否为空

此函数主要用来检测OPEN表为空，即搜索失败的停止。若为空（不含初始化），则表示搜索失败，退出执行程序。

c.void Push(Seqstack \*S,datatype x)//入栈

此函数除了具有入栈（《数据结构》基础知识，此处不再赘述）的基本功能之外，还在此基础上增加了入栈前的判断功能（不是栈满的判断）。

**设计目的：**根据深度优先搜索算法中向OPEN表添加元素的规则知，进入OEPN表的元素为OPEN表之前和当前均不存在的。

**实现策略：**由于之前OEPN表中的元素已出栈（被当前元素覆盖），此方案中借助CLOSED表实现对之前OPEN表中的元素的遍历。此方案中的遍历采用双层for循环实现。

d.void Pop(Seqstack \*S,datatype &x)//出栈

此函数主要用来弹出当前检测节点，其实现和基本出栈操作类似。

e.void Check\_input(int value)//检查输入是否正确

根据汉诺塔问题的实际情况，节点状态的输入值只能从1-3中进行选择。若用户输入错误，程序终止执行。

f.void Init\_Tnt()//汉诺塔问题的初值输入

此函数是为了让用户输入汉诺塔问题求解前的基本要求，包括开始节点和终止节点的输入及显示。

**C.主要函数构建**

1. void Get\_way(datatype data)//找到结果后向上回溯至根节点

**设计目的：**根据找到的最终节点（搜索成功）“向上规约”找到该解的具体路径，并将其输出。

**实现策略：**从终止节点向上回滚（通过比较当前节点的父节点是否和上一节点的子节点相等实现），直至找到问题的初始节点，并将此路径以一种恰当的方式展示给用户。

b.void Show\_inithut()//汉诺塔问题的初始化描述

此函数主要是为了以一种恰当的方式向用户展现汉诺塔求解前的具体要求（开始节点、终止节点）。

c.bool Test()//检测搜索是否成功

**设计目的：**判断当前正在扩展的节点是否为该问题的终止节点。

**实现策略：**每次扩展前需弹出OPEN表的顶端节点，将顶端节点与问题的终止节点进行比较。若相等，则搜索成功，输出解的路径，返回值为true;否则，返回值为false。

d.void Caculate\_Hnt()//汉诺塔问题的具体求解

**设计目的:**计算出由当前节点可扩展的所有节点，并将其依次压入OPEN表中。

**实现策略（举例分析法）：**由于扩展的节点必须根据当前的节点进行，且盘子移动的位置相对有限。故利用举例分析法进行，具体列举时的规则如下：

小盘移动。小盘可放在大盘上，移动起来相对自由。若当前所在柱子为1号柱，其可移动至2或3号柱。依次类推其它的情况。

非小盘移动。若其上有比其小的盘，则该盘不能移动；若其上无盘，移动时需要判断其所放柱子上是否比其小的盘。

注意各盘在扩展时从开始扩展的当前节点进行。

1. **解决方案整体概述**

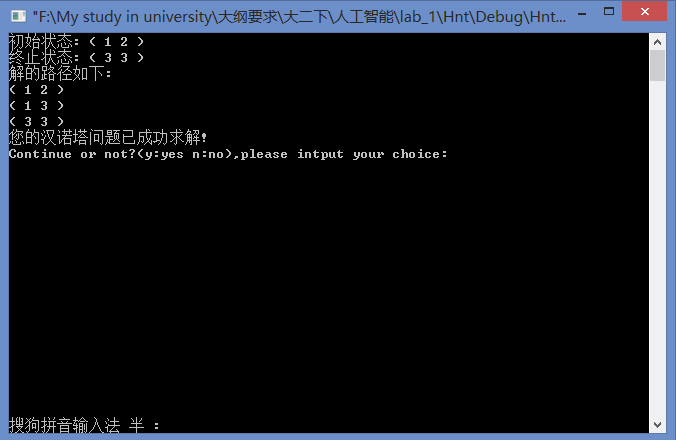
用户输入->用户数据的初始化处理->汉诺塔问题的初始化->汉诺塔问题的具体求解->最终结果的显示

Init\_Tnt()->Show\_inithut()->Caculate\_Hnt();

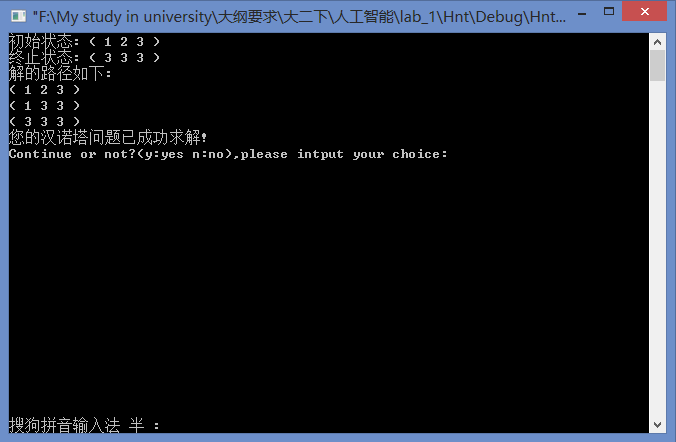
1. **程序运行结果图示说明**
2. **用户选择界面**



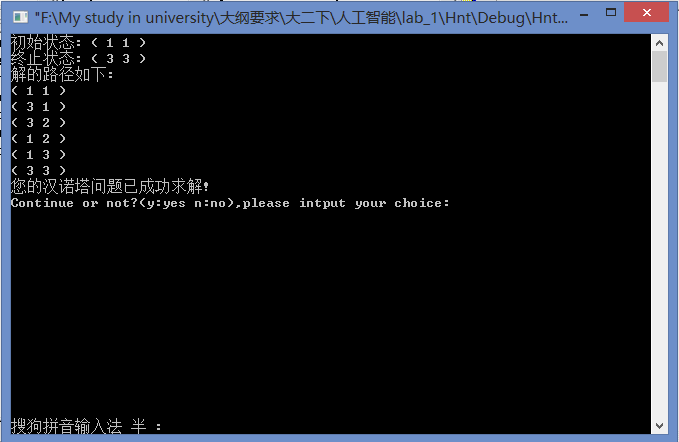
1. **广度优先搜索**
2. **二阶汉诺塔问题**



1. **三阶汉诺塔问题**



1. **深度优先搜索**
2. **二阶汉诺塔问题**



1. **三阶汉诺塔问题**



分析总结：

1. **主要的工作流程**
2. 用户数据的获取与初始化
3. 汉诺塔问题的初始化
4. 汉诺塔问题的具体求解
5. 最终结果的显示
6. **出现的主要问题与解决过程**
7. **数据结构的设计**

数据结构的设计对于整个问题的求解至关重要。

**主要问题：**开始设计时，由于过度借鉴之前写过的有关栈的程序，结果导致设计的数据结构不能很好的表示和解决相关问题。

**解决过程：**

1. 利用多维数组设计。（未采用，地址的询问和转移太过复杂，且不能准确表示出相关状态节点的信息）
2. 通过结构体设计，实现数据的“复合化”。（采用，方法表示灵活，层间数据访问方便，非常准确地表示了相关状态节点的信息）
3. **扩展节点的生成**

**主要问题：**开始设计时对程序设计方法了解不够，未能有效利用深度优先算法简化相关设计。由于自己的操作“跑题”，犯了许多算法上的低级错误。例如，扩展的节点不全面、新扩展的节点并未完全进入OPEN表中、直接对生成的节点进行判断。

**解决过程：**在意识到应根据算法准确拟合后，自己便准确地完成了节点的生成。

1. **OPEN表的入栈操作**

**主要问题：**开始时思维偏离具体算法，设计的入栈函数未能考虑进入OPEN表的节点的限制。

**解决过程：**紧扣算法，利用双层for循环将即将进入OPEN表的节点和当前OPEN表及CLOSED表（存放着之前OPEN表中的节点）的节点进行比较，从而判断新扩展的节点能否进入OPEN表。

1. **解的路径的生成**

**主要问题：**起初进行设计时只是简单的将CLOSED表已扩展的节点顺次打印出来，并未从终止节点进行归约，回溯到起始节点。

**解决过程：**新建一个栈表，装入从终止节点到初始节点的归约的各个节点，并倒序打印出来。

1. **不足**

该项目目前可解决所有的二阶汉诺塔问题的深度优先搜索和广度优先搜索，但对一些三阶汉诺塔问题的广度优先搜索却无法得到正确的结果（可能是扩展的节点不太全面）。

1. **反思**
2. 以后对具体问题进行求解时应该采用从整体到个体、一般到特殊的思想进行分析，避免犯错，少走弯路。(虽然自己通过边做边想的方法完成了此次实验，但花费时间较多，并且犯了许多错误。)
3. 对具体算法描述时一定要紧扣算法本身，从算法出发，切忌主观臆想。
4. 明确算法和程序的关系，编程只是为了描述相关算法。构建合适易行的算法才是解决问题的根本。