**实验一 搜索算法的应用**

**一、实验目的**

通过传教士与野人问题（M-C问题）的求解，加强对搜索算法的理解与应用。

**二、问题描述**

有N个传教士和N个野人来到河边渡河，河岸有一条船，每次至多可供K人乘渡。任何时刻岸上、船上的传教士数量不得小于野人数量，否则传教士会被野人吃掉。所有传教士和野人都要从河的左岸渡到河的右岸。

求传教士为了安全起见，应规划的乘渡方案。

**三、问题分析**

设M、C为某一时刻传教士人数和野人人数，k为川能承载的最大人数。要求M≥C且M+C≤k。即M、C = N，boat = k，要求M>=C且M+C <= k。

又设L表示船在左岸，R表示船在右岸。其中，B=1代表在左岸，B=0代表不在左岸。

由于传教士和野人的总数是一定的，所以我们只考虑左岸或右岸即可，这样可以减少状态变量的数量。

(1) 定义状态空间。

用三元组（ML , CL , BL）来表示传教士、野人和船是否在左岸的状态，ML表示传教士在左岸的实际人数，CL表示野人在左岸的实际人数，BL用来指示船是否在左岸。

当传教士和野人都为3时，0<=ML , CL <= 3 , BL ∈{ 0 , 1}，该问题的状态空间如下：

初始状态 目标状态

L R L R

M 3 0 M 0 3

C 3 0 C 0 3

B 1 0 B 0 1

于是，问题简化为：(3 , 3 , 1) (0 , 0 , 0)

N=3的M-C问题，状态空间的总状态数为4×4×2=32，根据约束条件的要求，可以看出只有20个合法的状态。进一步分析又能排除掉4种不可能达到的状态，所以状态空间的合法状态数为16。分析结果如下：

(ML ,CL , BL)

( 0， 0， 1 ) 不可达

( 0， 1， 1 )

( 0， 2， 1 )

( 0， 3， 1 )

( 1， 0， 1 ) 不合法

( 1， 1， 1 )

( 1， 2， 1 ) 不合法

( 1， 3， 1 ) 不合法

( 2， 0， 1 ) 不合法

( 2， 1， 1 ) 不合法

( 2， 2， 1 )

( 2， 3， 1 ) 不合法

( 3， 0， 1 ) 不可达

( 3， 1， 1 )

( 3， 2， 1 )

( 3， 3， 1 )

( 0， 0， 0 )

( 0， 1， 0 )

( 0， 2， 0 )

( 0， 3， 0 ) 不可达

( 1， 0， 0 ) 不合法

( 1， 1， 0 )

( 1， 2， 0 ) 不合法

( 1， 3， 0 ) 不合法

( 2， 0， 0 ) 不合法

( 2， 1， 0 ) 不合法

( 2， 2， 0 )

( 2， 3， 0 ) 不合法

( 3， 0， 0 )

( 3， 1， 0 )

( 3， 2， 0 )

( 3， 3， 0 ) 不可达

(2) 定义操作规则集合。

此问题的规则集合由摆渡操作组成，主要包括了Pmc操作（从左岸划向右岸）和Qmc操作（从右岸划向左岸）。

每次摆渡操作，船上人有5种组合，因而组成10条规则的集合：

P10: if ( ML ,CL , BL=1 ) then ( ML–1 , CL , BL –1 )

P01: if ( ML ,CL , BL=1 ) then ( ML , CL–1 , BL –1 )

P11: if ( ML ,CL , BL=1 ) then ( ML–1 , CL–1 , BL –1 )

P20: if ( ML ,CL , BL=1 ) then ( ML–2 , CL , BL –1 )

P02: if ( ML ,CL , BL=1 ) then ( ML , CL–2 , BL –1 )

Q10: if ( ML ,CL , BL=0 ) then ( ML+1 , CL , BL+1 )

Q01: if ( ML ,CL , BL=0 ) then ( ML , CL+1 , BL +1 )

Q11: if ( ML ,CL , BL=0 ) then ( ML+1 , CL +1, BL +1 )

Q20: if ( ML ,CL , BL=0 ) then ( ML+2 , CL +2, BL +1 )

Q02: if ( ML ,CL , BL=0 ) then ( ML , CL +2, BL +1 )

(3) 初始和目标状态。

初始状态为 (3, 3, 1)，目标状态为 (0, 0, 0)。

建立了产生式系统描述之后，就可以通过控制策略对状态空间进行搜索，求得一个摆渡操作的序列，使其能够实现目标状态。

N=3时的空间状态图是：

f=3 Q01

f=2 P02

f=1 Q01

f=1 Q10

f=1 P01

f=2 P11

（3，3，1）

（3，2，0）

（2，2，0）

（3，1，0）

（3，2，1）

（3，0，0）

f=3 P02

（3，1，1）

f=2 Q01

（1，1，0）

f=4 P20

（2，2，1）

f=2 Q11

（1，1，0）

f=4 P20

（2，2，1）

f=2 Q11

（0，2，0）

f=4 P20

（0，3，1）

f=3 Q01

（0，1，1）

f=5 P02

（0，2，1）

f=4 Q01

（0，0，0）

f=3 Q01

（1，1，1）

f=4 Q10

从空间状态图中可以看出解序列有多个，而最短序列只有4个，均由11次摆渡操作构成。若给定其中任意两个状态分别作为初始和目标状态，就可以立即找出对应的解序列。一般情况下，求解过程就是状态空间搜索出一条解路径的过程。

**四、功能设计**

此程序包含以下功能：

1. 根据不同的N和k来判断是否可解，若可解则求解问题，以动画效果演示。
2. 演示过程中可支持随时暂停或继续。
3. 查看程序说明的帮助文档。
4. 有合法输入的任何时刻可以重新开始演示。
5. 任意时刻正常退出程序。
6. 调节动画演示的速率快慢。

**五、变量设计**

全局变量（Display类里，主要表示(ML,CL,BL)）：

int missi; // 传教士人数

int canni; // 野人人数

int boat; // 小船数量

动画过程需要的重要变量（Pictures类里的一些重要变量）：

int mnumber; //传教士数

int cnumber; //野人数

int bnumber; //船

int mnumber2; //对岸传教士数

int cnumber2; //对岸野人数

int bnumber2; //对岸船

int mbno; //传教士数

int cbno; //野人数

int mcnumber=0; //总人数

int number=0; //节点数

核心算法的重要变量：

final int N = 65535; // 系统容忍最大数值

int kofmc; // 船上可容纳人数

int nofmc; // 野人和传教士人数

int number; // 当前已经产生的结点序号

Status tree[] = new Status[N]; // 存放状态节点

Display dp[] = new Display[N]; // 存放路径

int found = 0; // 找到最优解的标识

int num; // 路径节点数

**六、算法设计**

主要使用A\*搜索算法，代价函数为f(n) = g(n)+h(n) = d(n)+B+(M+C)/k。

d(n)表示的是当前状态结点n在状态图中的深度，搜索树的深度d(n)就是初始节点S0到节点n的实际代价，而节点n所在的路径不一定是最优路径，故有d(n)≥g\*(n)，当节点所在路径是最优路径时，有d(n)=g\*(n)，所以设计g(n)=d(n)是合适的。

代价函数第二部分的h(x)中，(M+C)/k为主要启发信息，其含义为左岸剩余人数至少多少次才能运完，B为调节信息，即启发信息中加入船是否存在的条件。显然，从节点n到达目标节点的最小代价h\*(x)不可能小于B+(M+C)/k，即h(x)≤h\*(x)，满足A\*算法的条件限制，因此做h(x)=B+(M+C)/k的设计是合适的。

此程序应用A\*算法的具体步骤：

1. 将初始节点S0放入OPEN表，并计算f(S0)=g(S0)+h(S0)。
2. 如果OPEN表为空，则搜索失败，问题无解退出。
3. 将OPEN表中的第一个节点从表中移出，放入CLOSED表，并记该节点为n。
4. 考察节点n是否为目标节点，如果是，则搜索成功；否则退出。
5. 如果n不可扩展，则转到步骤2
6. 扩展n节点，生成n的子结点ni (i=1,2,3, …)，计算每一个子节点的估价值f(ni) (i=1,2,3, …)，并设置每个子节点ni的父节点为n，然后将这些子节点ni放入OPEN表。
7. 根据各节点的估价函数f(n)，对OPEN表中的全部节点按从小到大的顺序重新进行排序。
8. 跳转到步骤2

其中，OPEN表存放待扩展节点，CLOSED表存放已扩展节点。

A\*算法相关的两个核心函数：估价函数，它是指引A\*算法搜索成功的关键；条件约束函数，在问题求解过程中，要时刻检测每一个状态的合法性，预防算法生成不合法的状态，预防算法生成不合法的状态而导致结果错误，同时也可以减少中间状态的数量。

估价函数：

private float estimate(int M,int C,int B,int depth) {

float retu = depth+B+((float)M+C)/kofmc;

return retu;

}

条件约束函数：

private boolean res(int M,int C) {

return ((M==C)&&(M<=nofmc)&&(C<=nofmc)&&(M>=0)&&(C>=0))||((M==0)&&

(C<=nofmc)&&(C>=0))||((M==nofmc)&&(C>=0)&&(C<=nofmc));

}

A\*算法的核心代码：

public void start() {

if(large) {

found = -1;

return;

}

Start=1;

found=0;

number=0;

// 初始化tree

tree[0].M=nofmc;

tree[0].C=nofmc;

tree[0].B=1;

tree[0].depth=0;

tree[0].oorc=0;

tree[0].point = -1;

int target=0;

int point;

point = target;

while(point != -1) {

int M,C,B,depth,mt,ct;

M = tree[target].M;

C = tree[target].C;

B = tree[target].B;

depth = tree[target].depth;

extend(M,C,B,depth,target);

change(target);

target=excellent();

if(target == -1) {

System.out.println("no solution");

break;

}

if(number >= N) {

found = -1;

System.out.println("too large number course exit!");

break;

}

point = target;

mt=tree[target].M;

ct=tree[target].C;

if((mt==0)&&(ct==0)) {

int ps;

ps = target;

int numtemp=0;

while(ps != -1) {

numtemp++;

ps = tree[ps].point;

}

num = numtemp;

ps = target;

while(ps != -1) {

numtemp--;

dp[numtemp].missi =tree[ps].M;

dp[numtemp].canni =tree[ps].C;

dp[numtemp].boat=tree[ps].B;

ps = tree[ps].point;

}

found=1;

break;

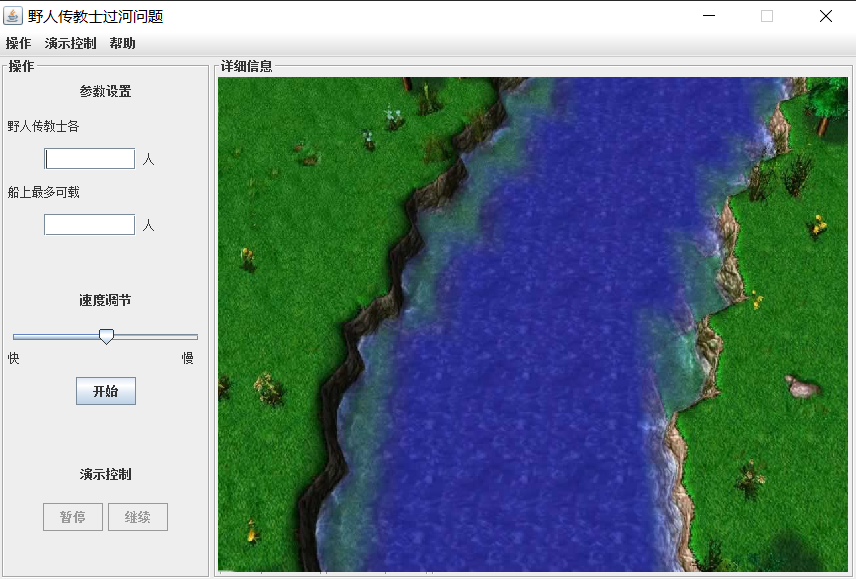
}

}

}

**七、用户界面设计**

采用Java自带的Swing技术实现用户图形界面，界面的样式布局如下：

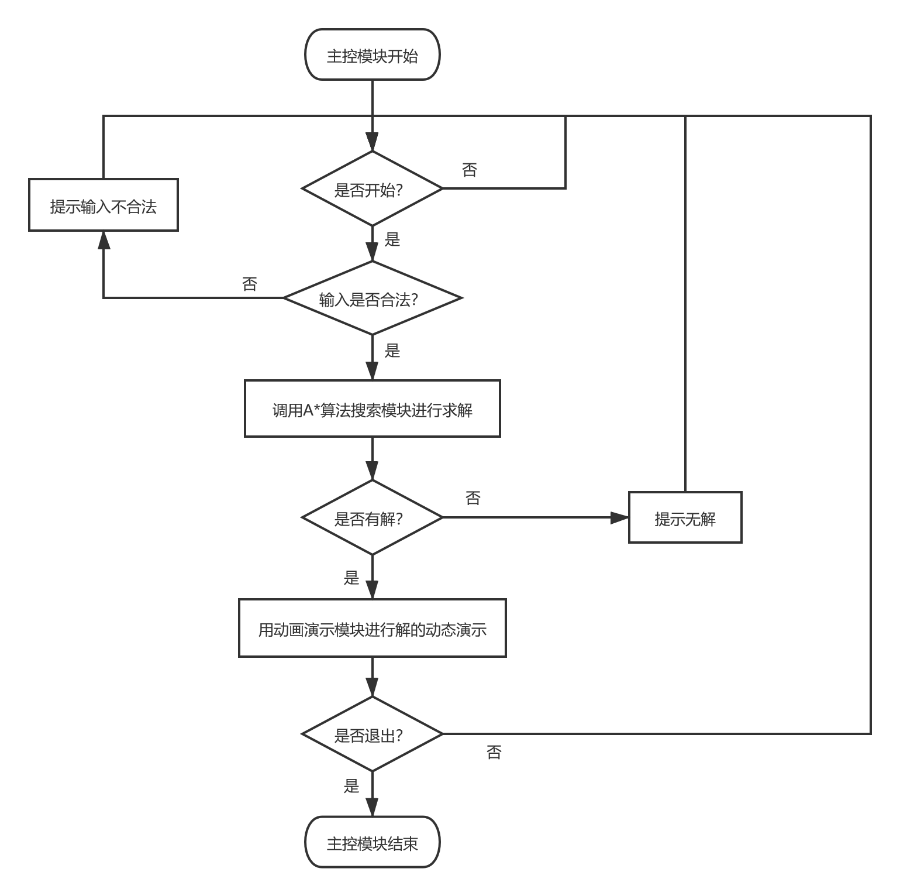


与图形界面编程伴随的是五张重要的图片，分别是河流背景板（上图右侧）、野人图标、传教士图标、左行船图标、右行船图标。

**八、总体设计与实现过程**

此程序使用Java语言编程实现，采用面向对象的思想进行程序设计，共包含八个类：Astar（A\*算法的实现类）、CrossRiver（控制系统工作流程的核心类）、Display（存储最优路径上各个节点状态信息）、DrawCanvas（动画控制类）、HelpDialog（程序帮助说明窗口类）、Main（唯一含有main函数的程序入口）、Pictures（动画线程演示类）、Status（状态空间中的节点抽象得到的状态类）。

整个系统的工作流程图如下：



考虑到暂停、继续、控制速度等可能发生的动作，系统的动画实现使用到了多线程编程技术，为此设计了一个控制线程类用来控制动画的演示速度和一个演示线程类。

演示控制分为开始、暂停、继续、报错四种情形。系统主模块调用A\*算法搜索得到问题的解存储在dp[]数组中，演示控制模块使用获取到的dp[]数组。动画暂停的时候需要记录三条重要信息：动画状态i、dp[]的当前索引、河对岸的状态，记录这些信息就可以恢复状态。

动画演示模块的线程描述动画演示过程中每一时刻的状态，以基本时间间隔为基准，动态刷新当前动画状态，实现动画演示。动画演示的状态可以由三条重要信息记录：动画状态i、dp[]的当前索引、河对岸的状态。

我们可以将动画演示线程抽象成14个演示状态，用i记录状态：

i = -1，初始状态，人在左岸，船在右岸，人未上船，两岸状态不改变

i = 0，船在左岸，人下船（初始时无该动作），左岸状态发生改变

i = 1，部分人在左岸上船，左岸状态发生改变

i = 2~6，部分人在船上，船从左岸向右岸行驶

i = 7，船在右岸，人下船，右岸状态发生改变

i = 8，部分人在右岸上船，右岸状态发生改变

i = 9~13，部分人在船上，船从右岸向左岸行驶

i = 14，取i%=14，转向i=0状态

动画演示的过程就是这14个状态来回转换的过程，在i=0和i=1时从dp[]中读取下一个状态，i=7时判断左岸状态是否为目标状态。当dp[]中节点数为空或者i=7且左岸状态为目标状态时，一轮演示结束。

Java线程实现的状态转换代码如下所示：

public void run() {

while(drawflag) {

try {

sleep(sleepTime\*15+10);

} catch(InterruptedException ie) {

System.err.println("线程中断！");

}

i = (i+1)%14;

drawcanvas.repaint();

if(i == 0 || i == 1) {

num++;

if(num >= number) {

drawflag = false;

}

}

}

}

**九、运行效果**

运行效果如下：

