

电子科技大学
计算机科学与工程学院

标准实验报告

(实验) 课程名称 人工智能

电子科技大学教务处制表

电子科技大学

实 验 报 告

学生姓名：刘文晨 学 号：2018080901006 指导教师：钟秀琴

实验地点：主楼 A2-412 实验时间：2020-10-18 周日 5-8 节 9-节

一、实验室名称： 计算机学院实验中心

二、实验项目名称：知识表示方法—渡河问题

三、实验学时：4 学时

四、实验原理：

本次试验选择传教士过河问题，以状态空间法实现。解答步骤如下：

1) 设置状态变量并确定值域

M 为传教士人数，C 为野人人数，B 为船数，要求 $M \geq C$ 且 $M, C \leq 3$ ，L 表示左岸，R 表示右岸。

初始状态

	L	R
M	3	0
C	3	0
B	1	0

目标状态

	L	R
M	0	3
C	0	3
B	0	1

2) 确定状态组，分别列出初始状态集和目标状态集

用三元组来表示 S_f ：(ML, CL, BL) (均为左岸状态)

其中 $0 \leq ML \leq 3, 0 \leq CL \leq 3, BL \in \{0, 1\}$

S_0 ：(3, 3, 1) \longrightarrow S_g ：(0, 0, 0)

初始状态表示全部成员在河的左岸；

目标状态表示全部成员从河的左岸全部渡河完毕。

3) 定义并确定规则集合

仍然以河的左岸为基点来考虑，把船从左岸划向右岸定义为 P_{ij} 操作。其中，第一下标 i 表示船载的传教士数，第二下标 j 表示船载的食人者数；同理，从右

岸将船划回左岸称之为 Q_{ij} 操作，下标的定义同前。共有 10 种操作，操作集为

$F = \{P_{01}, P_{10}, P_{11}, P_{02}, P_{20}, Q_{01}, Q_{10}, Q_{11}, Q_{02}, Q_{20}\}$
 P_{10} if (ML , CL , BL=1) then (ML - 1 , CL , BL - 1)
 P_{01} if (ML , CL , BL=1) then (ML , CL - 1 , BL - 1)
 P_{11} if (ML , CL , BL=1) then (ML - 1 , CL - 1 , BL - 1)
 P_{20} if (ML , CL , BL=1) then (ML - 2 , CL , BL - 1)
 P_{02} if (ML , CL , BL=1) then (ML , CL - 2 , BL - 1)
 Q_{10} if (ML , CL , BL=0) then (ML+1 , CL , BL+1)
 Q_{01} if (ML , CL , BL=0) then (ML , CL+1 , BL +1)
 Q_{11} if (ML , CL , BL=0) then (ML+1 , CL +1, BL +1)
 Q_{20} if (ML , CL , BL=0) then (ML+2 , CL +2, BL +1)
 Q_{02} if (ML , CL , BL=0) then (ML , CL +2, BL +1)

4) 当状态数量不是很大时，画出合理的状态空间图

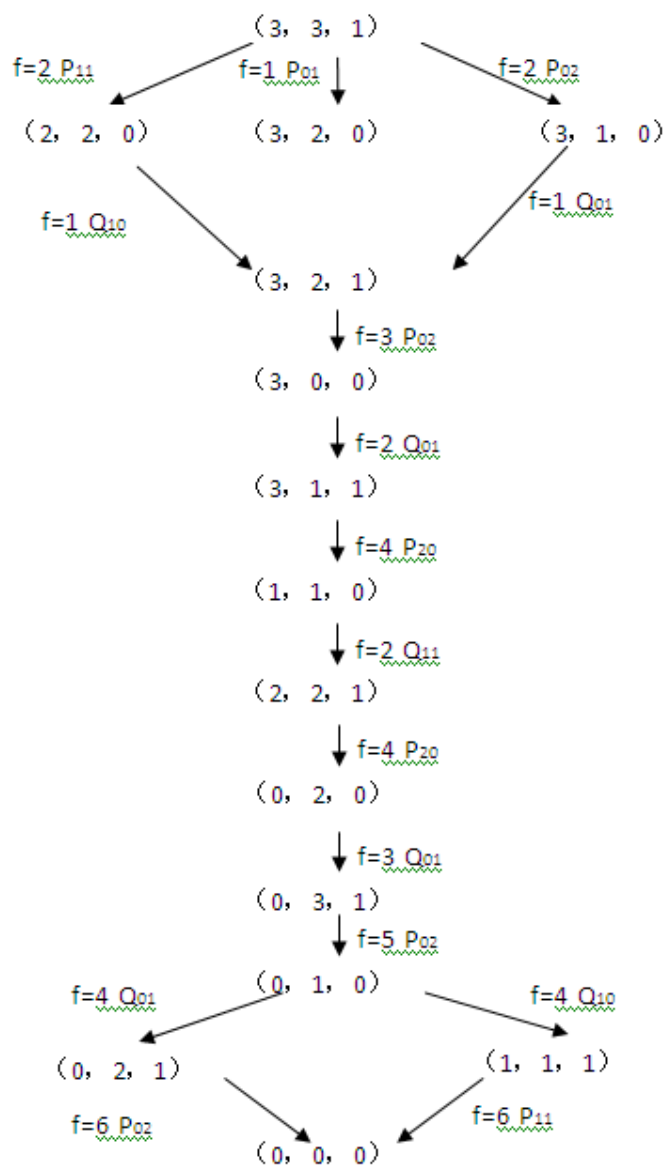


图 1 状态空间图

箭头旁边所标的数字表示了 P 或 Q 操作的下标，即分别表示船载的传教士数和食人者数。

接下来进行树的遍历，根据规则由根（初始状态）扩展出整颗树，检测每个结点的“可扩展标记”，为“-1”的即目标结点。由目标结点上溯出路径。

五、实验目的：

- (1) 了解知识表示相关技术；
- (2) 掌握状态空间法的分析方法。

六、实验内容：

状态空间法实验。从前有一条河，河的左岸有 $m(=3)$ 个传教士、 $c(=3)$ 个野人和一艘最多可乘 $n(=2)$ 人的小船。约定左岸，右岸和船上或者没有传教士，或者野人数量不超过传教士，否则野人会把传教士吃掉。搜索一条可使所有的野人和传教士安全渡到右岸的方案。

要求：

- (1) 提交源代码及可执行文件。
- (2) 提交实验报告，内容包括：对代码的简单说明、运行结果截图及说明等。
- (3) 加分点：可手动输入 m, c, n ，然后输出安全渡河的过程；如无安全渡河的方案，也给出提示

七、实验器材（设备、元器件）：

PC 微机一台

八、实验步骤：

程序执行流程：

- 1) 首先，包含状态（首次为初始状态）的结构体结点（已存入结构体数组）传入处理函数，然后判断该传入结点状态是否为目标状态。

是则遍历打印结构体数组，打印完成之后，返回递归调用处，顺序执行之后代码（此步骤关系到是否能找到所有过河路径）；

否则继续判断是否该传入结点已存在于结构体数组当中，如存在，不再往下执行，返回递归调用处，顺序执行之后代码；

若不存在，则继续判断该传入状态的人数是否合理（是否出现人物数量小于 0 的情况等），若不合理，返回递归调用处，顺序执行之后代码；

若合理，则继续判断传教士和野人人数限制条件，即在传教士人数不为 0 的情况下，野人人数是否大于传教士人数，若大于则出现吃人的情况，也就是说该传入状态也不合理，则返回递归调用处，顺序执行之后代码；

若不满足大于条件，则说明该状态是路径转态，也就是合理的，那么进行五种渡河方案的依次变换，首先为第一种渡河方案，两个传教士过河（注意：此处的 5 中渡河方案没有固定顺序，也可以是其他渡河方案），那么对该传入状态的左岸和右岸的传教士人数和野人人数进行增减。

- 2) 增减完成并改变船的状态（使用正负一表示，正一为左岸，负一为右岸）以后就产生了一个新的状态，将该状态存入结构体数组，之后此处又递归调用处理函数，将新产生的转态结点传入，再次进行上述条件限制判断。

若在该判断途中被返回至递归调用处，说明该状态不合理，则此时将已经存入结构体数组的状态结点移出结构体数组，然后程序顺序执行，进行下一个渡河方案的处理，也就是说，此时的处理是对上一个传入结点的操作（因为刚传入的已经移出了）；

若在判断途中未被返回至递归调用处，也就是说，传入的结点合理了，那么又开始从第一种渡河方案开始对该传入状态进行操作。

- 3) 按照上述过程循环执行，直到出现目标状态，回到本段开头，遍历结构体数组，打印渡河路径结点信息。
- 4) 完成以后，返回递归调用处，顺序执行之后代码，此后的操作是在寻找其他渡河路径。

程序说明：

1. 结构体 MyLeftBank 定义传教士、野蛮人和船的状态

```
1 struct MyLeftBank
2 {
3     int missionaries; //传教士
4     int barbarians;   //野蛮人
5     int boat_state;   //1表示船在左岸，0表示船在右岸
6 };
```

2. InitSailWay 函数负责初始化传教士、野蛮人的数量和路线编号

```
1 int InitSailWay(int sailWay[2][100], int& sailWayMax, const int boatNumber)
2 {
3     int miss_number = 0, barb_number = 0;           //初始化传教士和野蛮人的数量
4     sailWayMax = 0;                                   //初始化路线编号
5     //设置路线
6     while(miss_number != boatNumber+1 || barb_number != 0)
7     {
8         barb_number++;
9         if(miss_number + barb_number <= boatNumber)
10        {
11            sailWay[0][sailWayMax] = miss_number;
12            sailWay[1][sailWayMax] = barb_number;
13            sailWayMax++;
14        }
15        else
16        {
17            miss_number++;
18            barb_number = 0;
19            sailWay[0][sailWayMax] = miss_number;
20            sailWay[1][sailWayMax] = barb_number;
21            sailWayMax++;
22        }
23    }
24    sailWayMax--; //减去多余的状态
25    sailWay[0][sailWayMax] = 0;
26    sailWay[1][sailWayMax] = 0;
27
28    return 1;
29 }
```

3. BoatGoOppositeBank 函数负责船完成一次航行后，传教士、野蛮人和船的状态转换，出错返回-1

```

1 int BoatGoOppositeBank(MyLeftBank& this_,int& sail_way,const int
  input_miss,const int input_barb)
2 {
3     //如果船在左岸
4     if(LEFT_BANK == this_.boat_state)
5     {
6         //如果船能沿着这条路线航行
7         if(this_.missionaries >= g_sail_way[0][sail_way] && this_.barbarians
          >= g_sail_way[1][sail_way])
8         {
9             //改变传教士、野蛮人和船的状态
10            this_.missionaries -= g_sail_way[0][sail_way];
11            this_.barbarians -= g_sail_way[1][sail_way];
12            this_.boat_state = (g_step_number + 1) % 2;
13            step[g_step_number] = sail_way;
14            return 1;
15        }
16        sail_way++;
17        return 0;
18    }
19    //如果船在右岸
20    else if(RIGHT_BANK == this_.boat_state)
21    {
22        int left_miss = this_.missionaries + g_sail_way[0][sail_way];
23        int left_barb = this_.barbarians + g_sail_way[1][sail_way];
24        if(left_miss <= input_miss && left_barb <= input_barb)
25        {
26            //改变传教士、野蛮人和船的状态
27            this_.missionaries = left_miss;
28            this_.barbarians = left_barb;
29            this_.boat_state = (g_step_number + 1) % 2;
30            return 1;
31        }
32        sail_way++;
33        return 0;
34    }
35    else //船的状态出错
36    {
37        cout << "\nBoat state goes wrong.\n";
38        system("pause");
39        return -1;
40    }
41 }

```

4. JudgeBothBankState 函数负责检查两岸的状况是否合法

```

1 int JudgeBothBankState(const MyLeftBank this_,int& sail_way ,const int
  input_miss, const int input_barb)
2 {
3     for(int i=0;i<g_step_number;i++)//如果船的状态与前一个状态相同，则使用下一个航
      行方式航行
4     {
5         if(g_crossing_history[0][i] == this_.missionaries &&
          g_crossing_history[1][i] == this_.barbarians && g_crossing_history[2][i] ==
          this_.boat_state)
6         {
7             return 0;
8         }
9     }
10    //排除特殊情况：野蛮人人数超过0，传教士人数为0
11    //barb_number>miss_number,但状态合法
12    if((this_.missionaries == 0 && (this_.barbarians <= input_barb)) ||
      (this_.missionaries == input_miss && (this_.barbarians <= input_barb)) ||
      (this_.missionaries == 0 && this_.barbarians == 0))
13    {
14        g_crossing_history[0][g_step_number] = this_.missionaries;
15        g_crossing_history[1][g_step_number] = this_.barbarians;
16        g_crossing_history[2][g_step_number] = this_.boat_state;
17        step[g_step_number] = sail_way;
18
19        sail_way = 0;
20        g_step_number++;
21
22        return 1;
23    }
24    //检查两岸的状况
25    int right_miss = input_miss - this_.missionaries;
26    int right_barb = input_barb - this_.barbarians;
27    if((this_.missionaries >= this_.barbarians) && (right_miss >= right_barb))
28    {
29        g_crossing_history[0][g_step_number] = this_.missionaries;
30        g_crossing_history[1][g_step_number] = this_.barbarians;
31        g_crossing_history[2][g_step_number] = this_.boat_state;
32        step[g_step_number] = sail_way;
33
34        sail_way = 0;
35        g_step_number++;
36
37        return 1;
38    }
39    else
40    {
41        return 0;
42    }
43 }

```

5. ReturnAvailableState 函数负责当到达目标状态时，返回可用的路线


```

1 int ReturnAvailableState(MyLeftBank& this_, int& sail_way,const int
  sail_way_max)
2 {
3     //当到达目的地时, 返回可用的路线
4     if((0 == this_.boat_state) && (0 == this_.missionaries) && (0 ==
      this_.barbarians))
5     {
6         g_step_number -= 2;
7         this_.missionaries = g_crossing_history[0][g_step_number - 1];
8         this_.barbarians = g_crossing_history[1][g_step_number - 1];
9         this_.boat_state = (g_step_number) % 2;
10
11         sail_way = step[g_step_number];
12         sail_way++;
13         return 1;
14     }
15     //恢复传教士、野蛮人和船的状态
16     if(++sail_way<sail_way_max)
17     {
18         this_.missionaries = g_crossing_history[0][g_step_number-1];
19         this_.barbarians = g_crossing_history[1][g_step_number-1];
20         this_.boat_state = (g_step_number) % 2;
21     }
22     //返回到最后可用的路线状态
23     while(sail_way>=sail_way_max)
24     {
25         if(g_step_number-- > 1)
26         {
27             this_.missionaries = g_crossing_history[0][g_step_number-1];
28             this_.barbarians = g_crossing_history[1][g_step_number - 1];
29             this_.boat_state = (g_step_number) % 2;
30             sail_way = step[g_step_number];
31             sail_way++;
32         }
33         else return 0;
34     }
35     return 1;
36 }

```

6. PrintingSailWay 函数负责输出安全渡河的方案编号和过程

```
1 void PrintingSailWay(const int successWay)
2 {
3     cout << "\n第" << successWay << "种方案: " << endl;
4     for(int i=0;i<g_step_number;i++)
5     {
6         cout << '(' << g_crossing_history[0][i] << "," <<
7             g_crossing_history[1][i] << "," << g_crossing_history[2][i] << ')' << endl;
8     }
9 }
```

7. CrossRiver 函数负责统计安全渡河的方案数，如无安全渡河的方案，也给出“很遗憾，我们找不到方案”的提示

```

1 void CrossRiver(MyLeftBank & leftbank, const int inputMissionaries, const int
  inputBarbarians, const int inputBoatnumber)
2 {
3     int sailWay = 0;
4     int success_way = 0;
5     InitSailWay(g_sail_way, g_sail_way_max, inputBoatnumber);
6     while(!(g_step_number == 0 && sailWay >= g_sail_way_max))
7     {
8         while(!((0 == leftbank.boat_state) && (0 == leftbank.missionaries) &&
          (0 == leftbank.barbarians)))
9         {
10             if(1 == BoatGoOppositeBank(leftbank, sailWay, inputMissionaries,
              inputBarbarians))
11             {
12                 if(0 == JudgeBothBankState(leftbank, sailWay,
                  inputMissionaries, inputBarbarians))
13                 if(0 == ReturnAvailableState(leftbank, sailWay,
                  g_sail_way_max))
14                     break;
15             }
16             if(sailWay >= g_sail_way_max)
17             {
18                 ReturnAvailableState(leftbank, sailWay, g_sail_way_max);
19             }
20         }
21         if((0 == leftbank.boat_state) && (0 == leftbank.missionaries) && (0 ==
          leftbank.barbarians))
22         {
23             success_way++;
24             PrintingSailWay(success_way);
25             ReturnAvailableState(leftbank, sailWay, g_sail_way_max);
26         }
27     }
28     if(success_way == 0)
29     {
30         cout << "\n很遗憾，我们找不到方案\n\n";
31     }
32     else cout << "\n我们总共找到了" << success_way << "种方案" << endl;
33 }

```

九、实验数据及结果分析：

程序截图：

```
请输入传教士人数:3
请输入野蛮人人数:3
请输入船的最大载人数:2
```

```
第1种方案:
```

```
(3, 3, 1)
(3, 1, 0)
(3, 2, 1)
(3, 0, 0)
(3, 1, 1)
(1, 1, 0)
(2, 2, 1)
(0, 2, 0)
(0, 3, 1)
(0, 1, 0)
(0, 2, 1)
(0, 0, 0)
```

```
第2种方案:
```

```
(3, 3, 1)
(3, 1, 0)
(3, 2, 1)
(3, 0, 0)
(3, 1, 1)
(1, 1, 0)
(2, 2, 1)
(0, 2, 0)
(0, 3, 1)
(0, 1, 0)
(1, 1, 1)
(0, 0, 0)
```

```
第3种方案:
```

```
(3, 3, 1)
(2, 2, 0)
(3, 2, 1)
(3, 0, 0)
(3, 1, 1)
(1, 1, 0)
(2, 2, 1)
(0, 2, 0)
(0, 3, 1)
(0, 1, 0)
(0, 2, 1)
(0, 0, 0)
```

```
第4种方案:
```

```
(3, 3, 1)
(2, 2, 0)
(3, 2, 1)
(3, 0, 0)
(3, 1, 1)
(1, 1, 0)
(2, 2, 1)
(0, 2, 0)
(0, 3, 1)
(0, 1, 0)
(1, 1, 1)
(0, 0, 0)
```

```
我们总共找到了4种方案
```

```
-----
Process exited after 1.606 seconds with return value 0
```

分析可知，程序结果符合前面给出的状态空间图的过程。

当 m, c, n 增大时，程序花费的时间大幅度增加：

```
请输入传教士人数:5  
请输入野蛮人人数:5  
请输入船的最大载人数:3
```

```
我们总共找到了361种方案
```

```
-----  
Process exited after 7.826 seconds with return value 0  
请按任意键继续. . .
```

十、实验结论：

该程序实现了状态空间法求解传教士过河问题，可手动输入 m, c, n ，然后输出安全渡河的过程；如无安全渡河的方案，也给出提示。

十一、总结及心得体会：

通过本次试验，我掌握了实现了状态空间法求解传教士过河问题和深度优先搜索算法。

十二、对本实验过程及方法、手段的改进建议：

此程序仅仅实现计算效果，寻找出所有可执行方案，并未开多线程进行优化。因此当计算数据过大时，计算需要很多时间，可能导致界面进程堵塞，会造成程序停止响应。

报告评分：

指导教师签字：

电子科技大学

实验报告

学生姓名：刘文晨 学号：2018080901006 指导教师：钟秀琴

实验地点：主楼 A2-412 实验时间：2020-10-18 周日 5-8 节 9-节

一、实验室名称： 计算机学院实验中心

二、实验项目名称：A*算法实验

三、实验学时：4 学时

四、实验原理：

A*算法是一种启发式图搜索算法，其特点在于对估价函数的定义上。对于一般的启发式图搜索，总是选择估价函数 f 值最小的节点作为扩展节点。因此， f 是根据需要找到一条最小代价路径的观点来估算节点的，所以，可考虑每个节点 n 的估价函数值为两个分量：从起始节点到节点 n 的实际代价 $g(n)$ 以及从节点 n 到达目标节点的估价代价 $h(n)$ ，且 $h(n) \leq h^*(n)$ ， $h^*(n)$ 为 n 节点到目的结点的最优路径的代价。

八数码问题是在 3×3 的九宫格棋盘上，摆有8个刻有1~8数码的将牌。棋盘中有有一个空格，允许紧邻空格的某一将牌可以移到空格中，这样通过平移将牌可以将某一将牌布局变换为另一布局。针对给定的一种初始布局或结构（目标状态），问如何移动将牌，实现从初始状态到目标状态的转变。如下图表示了一个具体的八数码问题求解。



图2 八数码问题的求解

五、实验目的：

熟悉和掌握启发式搜索的定义、估价函数和算法过程，并利用 A*算法求解 N 数码难题，理解求解流程和搜索顺序。

六、实验内容：

1. 以 8 数码问题为例实现 A*算法的求解程序（编程语言不限），设计估价函数。

注：需在实验报告中说明估价函数，并附对应的代码。

2. 设置初始状态和目标状态，针对估价函数，求得问题的解，并输出移动过程。

要求：

（1）提交源代码及可执行文件。

（2）提交实验报告，内容包括：对代码的简单说明、运行结果截图及说明等。

七、实验器材（设备、元器件）：

PC 微机一台

八、实验步骤：

1) 估价函数 $f(n) = g(n) + h(n)$

其中 $f(n)$ 是节点 n 从初始点到目标点的估价函数， $g(n)$ 是在状态空间中从初始节点到 n 节点的实际代价， $h(n)$ 是从 n 到目标节点最佳路径的估计代价。

设 $g(n)$ 为已经移动的步数。在 move 函数中，每次移动加 1 即可。

设 $h(n)$ 为此状态与目标状态中相异数字的个数。由 a_start_h 函数求得：

```

//求h(n)
int a_start_h(Node *node)
{
    int old_x,old_y,End_x,End_y;
    int h=0;
    for(int k=1;k<9;k++)
    {
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                if(node->a[i][j] == k) //相应开始点的下标
                {
                    old_x=i;
                    old_y=j;
                }
                if(End.a[i][j] == k) //相应目标的结点下标
                {
                    End_x=i;
                    End_y=j;
                }
            }
        }
        //计算h(n)
        h+=abs(old_x-End_x)+abs(old_y-End_y);
    }
    return h;
}

```

2) open 表与 close 表的维护

open 表：可以简单认为是一个未搜索节点的表

close 表：可以简单认为是一个已完成搜索的节点的表（即已经将下一个状态放入 open 表内）

规则一：对于新添加的节点 S （open 表和 close 表中均没有这个状态）， S 直接添加到 open 表中。

规则二：对于已经添加的节点 S （open 表或者 close 表中已经有这个状态），若在 open 表中，与原来的状态 S_0 的 $f(n)$ 比较，取最小的一个。若在 close 表中，那就分成两种情况：第一种，close 表中的该状态 S_0 的 $f(n)$ 大于 S 的，不做修改；第二种 S_0 的 $f(n)$ 小于 S 的，那就要需要将 close 表中 S_0 的 $f(n)$ 更新，同时将该状态移入到 open 表中。

规则三：下一个搜索节点的选择问题，选取 open 表中 $f(n)$ 的值最小的状态作为下一个待搜索节点

规则四：每次需要将带搜索的节点下一个所有的状态按照规则一、二更新 open 表，close 表，搜索完该节点后，移到 close 表中。

3) 无解情况

将九宫格变成线性后，计算初始状态和目标状态的奇偶性是否一致，一

致有解，否则无解。

```
//判断是否有解：逆序数之和奇偶性相同，有解
bool isable()
{
    int s[9],e[9];
    int tf=0,ef=0,k=0;
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            s[k]=start.a[i][j];
            e[k]=End.a[i][j];
            k++;
        }
    }
    for(int i=0;i<9;i++)
    {
        for(int j=0;j<i;j++)
        {
            if (s[i]>s[j] && s[j]!=0) tf+=1;
            if (e[i]>e[j] && e[j]!=0) ef+=1;
        }
    }
    if((tf%2 == 1 && ef%2 == 1) || (tf%2 == 0 && ef%2 == 0)) return true;
    else return false;
}
```

九、实验数据及结果分析：

输入起始图和目标图，若有解，则输出移动过程和步数；若无解，则输出无解。

```
请输入起始图：
2 0 3
1 8 4
7 6 5

请输入目标图：
1 2 3
8 0 4
7 6 5

移动过程：
-----
0 2 3
1 8 4
7 6 5
-----

1 2 3
0 8 4
7 6 5
-----

1 2 3
8 0 4
7 6 5
-----

该问题有解，一共用了3步
-----
Process exited after 20.56 seconds with return value 0
请按任意键继续. . .
```

十、实验结论：

A*算法能找到的解是局部最优解，但是独特的启发式函数可以使得解为全局最优解，八数码问题就是一个能通过 A*算法求得最优解的问题。

十一、总结及心得体会：

掌握了 A*算法求解八数码问题。

十二、对本实验过程及方法、手段的改进建议：

在找最小值的时候，我们可以用二分查找，可以从 $o(n)$ 优化到 $o(\log n)$ ，这就要求我们再插入时顺序插入。因为查询次数是要大于添加 open/close 表项的，所以这个方法可以优化执行效率。

报告评分：

指导教师签字：

电子科技大学

实验报告

学生姓名：刘文晨 学 号：2018080901006 指导教师：钟秀琴

实验地点：主楼 A2-412 实验时间：2020-11-1 周日 9-节

一、实验室名称：计算机学院实验中心

二、实验项目名称：BP 神经网络实验

三、实验学时：4 学时

四、实验原理：

误差逆传播(back propagation, BP)算法是一种计算单个权值变化引起网络性能变化的较为简单的方法。由于 BP 算法过程包含从输出节点开始，反向地向第一隐含层(即最接近输入层的隐含层)传播由总误差引起的权值修正，所以称为“反向传播”。BP 神经网络是有教师指导训练方式的多层前馈网络，其基本思想是：从网络输入节点输入的样本信号向前传播，经隐含层节点和输出层节点处的非线性函数作用后，从输出节点获得输出。若在输出节点得不到样本的期望输出，则建立样本的网络输出与其期望输出的误差信号，并将此误差信号沿原连接路径逆向传播，去逐层修改网络的权值和节点处阈值，这种信号正向传播与误差信号逆向传播修改权值和阈值的过程反复进行，直训练样本集的网络输出误差满足一定精度要求为止。

五、实验目的：

编程实现 BP 神经网络算法；理解算法原理。

六、实验内容：

将 Iris（鸢尾花）数据集分为训练集（Iris-train.txt）和测试集（Iris-test.txt），分别含 75 个样本，每个集合中每种花各有 25 个样本。为了方便训练，将 3 类花分别编号为 1，2，3。使用这些数据训练一个 4 输入（分别对应 4 个特征）、隐含层（10 个神经元）、3 输出（分别对应该样本属于某一品种的可能性大小）的神经网络（4*10*3）。

使用训练集对网络进行训练，再预测测试集中每个样本的标签，并输出预测准确率（独立运行 10 次，列出 10 次的准确率，并输出平均准确率和标准差）。

要求：

（1）提交源代码及可执行文件。

（2）提交实验报告，内容包括：对代码的简单说明、运行结果的截图及说明等。

注意：实验报告中，需指明梯度下降过程中的学习率，并与相应的源码对应。

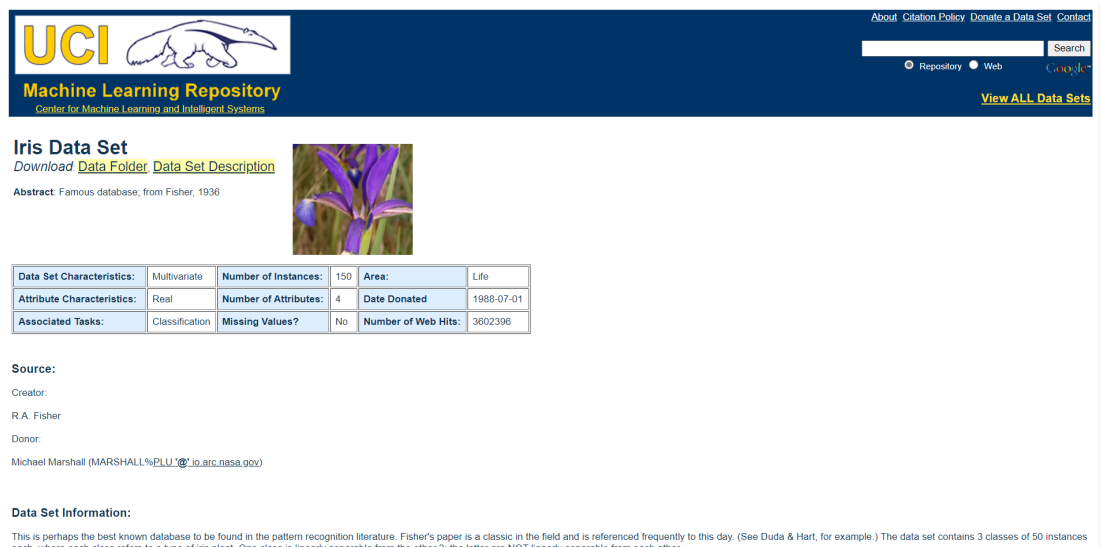
七、实验器材（设备、元器件）：

PC 微机一台

八、实验步骤：

1) 下载 Iris 数据集并导入


进入[Iris Data Set](https://archive.ics.uci.edu/ml/dataset/iris)下载鸢尾花数据集iris.data:



UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Iris Data Set
Download [Data Folder](#) [Data Set Description](#)

Abstract: Famous database, from Fisher, 1936



Data Set Characteristics:	Multivariate	Number of Instances:	150	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	4	Date Donated	1988-07-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	3602396

Source:
Creator:
R.A. Fisher
Donor:
Michael Marshall (MARSHALL%PLU%@io.arc.nasa.gov)

Data Set Information:
This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. (One class is linearly separable from the other 2; this latter can NOT be linearly separated from each other.)

导入语句为：

```
f=fopen('C:\Users\81228\Desktop\资料\人工智能\实验\实验 3\iris.data');
```

括号里是 iris.data 的地址。

2) 数据归一化

%数据归一化处理

```
x_max=max(data);
```

```
x_min=min(data);
```

```
data=(data-ones(m,1)*x_min)./(ones(m,1)*(x_max-x_min));
```

3) 划分训练样本和测试样本

将 Iris（鸢尾花）数据集分为训练集（Iris-train.txt）和测试集（Iris-test.txt），分别含 75 个样本。

%划分训练样本和测试样本 大约 1/2 用作训练，1/2 用作测试

```
num=round(m/2/3);
for i=1:3
    temp=data(1+50*(i-1):50*i,:);
    sel=randperm(50,num);
    test(1+num*(i-1):num*i,:)=temp(sel,1:4);
    test_label(1+num*(i-1):num*i,:)=temp(sel,5:7);
    temp(sel,:)=[];
    train(1+(50-num)*(i-1):(50-num)*i,:)=temp(:,1:4);
    train_label(1+(50-num)*(i-1):(50-num)*i,:)=temp(:,5:7);
end
```

4) BP 神经网络设置

```
alpha=4;%输入神经元数目
beta=10;%隐层神经元数目
lamda=3;%输出神经元数目
rng('shuffle')
W1=rand(alpha,beta);%输入层和隐层之间的权值矩阵
W2=rand(beta,lamda);%隐层和输出层间的权值矩阵
B1=rand(1,beta);%隐层阈值矩阵
B2=rand(1,lamda);%输出层阈值矩阵
B22=B2;
W11=W1;
Eta=1;%学习率
iter_max=10000;%最大迭代次数
iter=1;
```

5) 测试

```
%测试
[result,accuracy]=BP_test(test,test_label,W1,W2,B1,B2);
disp('结果为: ')
result'
disp(strcat('准确率为',num2str(accuracy*100),'%'))
function [result,accuracy]=BP_test(test,test_label,W1,W2,B1,B2)
%返回分类的结果
%accuracy 准确率
Hidden_in=test*W1;%隐层输入
Hidden_out=sigmoid(Hidden_in-B1);%隐层输出
Output_in=Hidden_out*W2;%输入层输入
```

```

Output_out=sigmoid(Output_in-B2);
[~,result]=max(Output_out,[],2);
[~,index2]=max(test_label,[],2);
right=sum(result==index2);%统计分类正确的个数
total=size(test,1);%总个数
accuracy=right/total;
end
%sigmoid函数在matlab里是logsig函数
function [Y]=sigmoid(X)
Y=1./(1+exp(-1).^X);
end

```

九、实验数据及结果分析：

单次运行结果为：

结果为：

```

ans =

    列 1 至 19
    1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

    列 20 至 38
    1     1     1     1     1     1     2     2     2     2     2     2     2     2     2     2     2     2

    列 39 至 57
    2     2     2     2     2     2     2     2     2     2     2     2     2     2     3     3     3     3

    列 58 至 75
    3     3     3     2     3     2     3     3     3     3     3     3     2     3     3     3     3     3

```

准确率为93.3333%

独立运行 10 次，预测准确率分别为：

96%	93.3333%	97.3333%	93.3333%	98.6667%
89.3333%	94.6667%	96%	96%	97.3333%

平均准确率为 95%，标准差为 0.025438487。

十、实验结论：

BP 神经网络算法能够很好地解决鸢尾花种类识别问题。

十一、总结及心得体会：

掌握了 BP 神经网络算法解决鸢尾花种类识别问题，加深了对机器学习的理解。

十二、对本实验过程及方法、手段的改进建议：

利用 PSO 算法对参数进行优化。

报告评分：

指导教师签字：

电子科技大学

实验报告

学生姓名：刘文晨 学号：2018080901006 指导教师：钟秀琴

实验地点：主楼 A2-412 实验时间：2020-10-25 周日 9-节

一、实验室名称：计算机学院实验中心

二、实验项目名称：决策树实验

三、实验学时：4 学时

四、实验原理：

(1) ID3 算法

ID3 算法的核心思想就是以信息增益度量属性选择，选择分裂后信息增益最大的属性进行分裂。下面先定义几个要用到的概念。设 D 为用类别对训练元组进行的划分，则 D 的熵（entropy）表示为：

$$\inf o(D) = - \sum_{i=1}^m p_i \log_2(p_i)$$

其中 p_i 表示第 i 个类别在整个训练元组中出现的概率，可以用属于此类别元素的数量除以训练元组元素总数量作为估计。熵的实际意义表示是 D 中元组的类标号所需要的平均信息量。现在我们假设将训练元组 D 按属性 A 进行划分，则 A 对 D 划分的期望信息为：

$$\inf o_A(D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \inf o(D_j)$$

而信息增益即为两者的差值：

$$gain(A) = \inf o(D) - \inf o_A(D)$$

ID3 算法就是在每次需要分裂时，计算每个属性的增益率，然后选择增益率最大的属性进行分裂。

对于特征属性为连续值，可以如此使用 ID3 算法：先将 D 中元素按照特征属性排序，则每两个相邻元素的中间点可以看做潜在分裂点，从第一个潜在分裂点开始，分裂 D 并计算两个集合的期望信息，具有最小期望信息的点称为这个属性的最佳分裂点，其信息期望作为此属性的信息期望。

五、实验目的：

编程实现决策树算法 ID3；理解算法原理。

六、实验内容：

利用 `traindata.txt` 的数据（75*5，第 5 列为标签）进行训练，构造决策树；利用构造好的决策树对 `testdata.txt` 的数据进行分类，并输出分类准确率。

要求：

（1）提交源代码及可执行文件。

（2）提交实验报告，内容包括：对代码的简单说明、运行结果的截图及说明等。

（3）需画出决策树，指明每个分支所对应的特征/属性，以及分裂值。

注：如用到了剪枝、限定深度等技巧（加分项），请加以说明。

七、实验器材（设备、元器件）：

PC 微机一台

八、实验步骤：

1. 计算信息熵


```

1 # 计算信息熵
2 def calEnt(index, label, step):
3     kind = [0, 0, 0]
4     EntD = 0
5     pk = [0, 0, 0]
6     for n in range(step):
7         if label[index[n]] == 0:
8             kind[0] += 1
9         elif label[index[n]] == 1:
10            kind[1] += 1
11        else:
12            kind[2] += 1
13    for n in range(3):
14        pk[n] = kind[n] / len(label)
15        if pk[n] == 0:
16            EntD += 0
17        else:
18            EntD -= pk[n] * log(pk[n], 2)
19    return EntD

```

2. 划分数据集

```

1 # 根据求得的最大增益，属性，划分值，划分数据集
2 def splitDataSet(dataSet, value, line):
3     smallDataSet = np.empty(shape=(0, 5))
4     largeDataSet = np.empty(shape=(0, 5))
5     for n in range(len(dataSet[:, -1])):
6         if dataSet[n, line] < value:
7             smallDataSet = np.append(smallDataSet, [dataSet[n, :]], axis=0)
8         else:
9             largeDataSet = np.append(largeDataSet, [dataSet[n, :]], axis=0)
10    return smallDataSet, largeDataSet

```

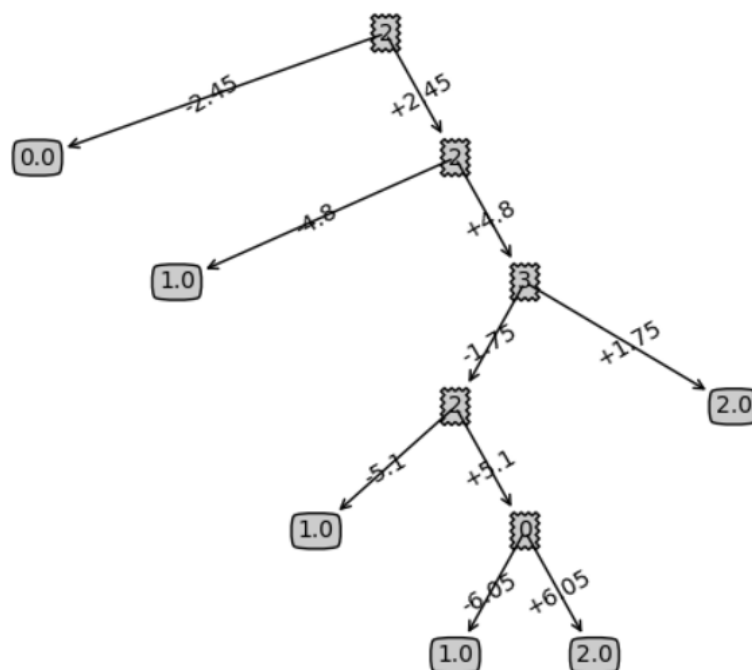
3. 计算最大增益

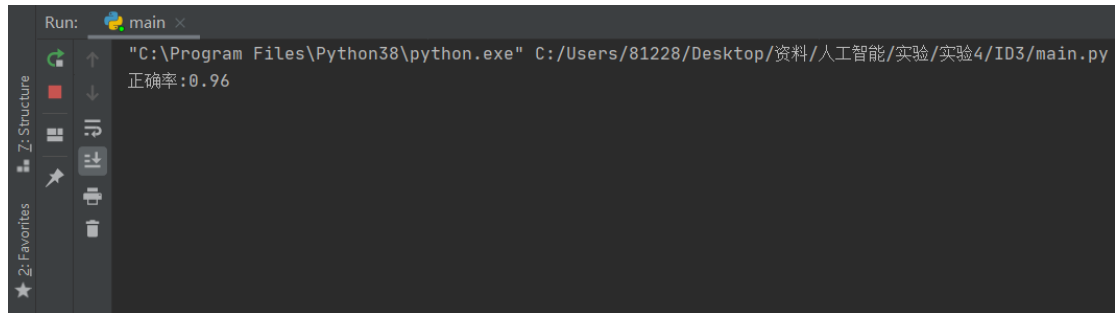
```

1 # dataSet(array类型)返回最大信息增益及其所在的属性和划分值
2 def bestFeature(dataSet):
3     index = []
4     for n in range(4):
5         index.append(np.argsort(dataSet[:, n], axis=0))
6     EntD = calEnt(range(len(dataSet[:, 0])), dataSet[:, -1], len(dataSet[:,
7     0]))
8     T = [[], [], [], []] # 4个连续属性的候选划分点
9     Gain = [[], [], [], []] # 4个连续属性的候选增益
10    for n in range(len(dataSet[:, -1]) - 1):
11        for i in range(4):
12            T[i].append((dataSet[index[i][n], i] + dataSet[index[i][n + 1],
13            i]) / 2)
14            EntDv = [0, 0]
15            for j in range(len(dataSet[:, 0]) - 1):
16                EntDv[0] = calEnt(index[i][:j + 1], dataSet[:, -1], j + 1)
17                EntDv[1] = calEnt(index[i][j + 1:], dataSet[:, -1],
18                len(index[i][j + 1:]))
19            Gain[i].append(EntD - ((j + 1) / (len(dataSet[:, 0]) - 1)) *
20            EntDv[0] - (
21                ((len(dataSet[:, 0]) - 1) - (j + 1)) / (len(dataSet[:,
22            0]) - 1)) * EntDv[1])
23    MAX_Gain = [0, 0, 0, 0]
24    MAX_index = [0, 0, 0, 0]
25    for n in range(4):
26        MAX_Gain[n] = max(Gain[n])
27        MAX_index[n] = T[n][Gain[n].index(max(Gain[n]))]
28    # 最大增益 划分值 所在的属性列
29    return max(MAX_Gain), MAX_index[MAX_Gain.index(max(MAX_Gain))],
30    MAX_Gain.index(max(MAX_Gain))

```

九、实验数据及结果分析：





```
Run: main ×  
"C:\Program Files\Python38\python.exe" C:/Users/81228/Desktop/资料/人工智能/实验/实验4/ID3/main.py  
正确率:0.96
```

十、实验结论：

本实验利用 traindata.txt 的数据（75*5，第 5 列为标签）进行训练，构造决策树，并利用构造好的决策树对 testdata.txt 的数据进行分类，分类准确率为 96%，符合预期。

十一、总结及心得体会：

掌握了编程实现决策树算法 ID3，理解了 ID3 算法原理。

十二、对本实验过程及方法、手段的改进建议：

调用 sklearn 库能够更加简单地完成实验，且分类准确率更高。

报告评分：

指导教师签字：

附件 1：实验 1 代码 CrossingtheRiver.cpp

```
#include<bits/stdc++.h>
using namespace std;

#define MISSIONARIES_MAX 3
#define BARBARIAN_MAX 3
#define SAILWAY_MAX 5
#define LEFT_BANK 1
#define RIGHT_BANK 0
#define STEP_MAX 100

extern int step[STEP_MAX];
extern int g_step_number;
extern int g_sail_way_max;
extern int g_crossing_history[3][STEP_MAX];

struct MyLeftBank
{
    int missionaries;    //传教士
    int barbarians;      //野蛮人
    int boat_state; //1 表示船在左岸，0 表示船在右岸
};

int step[STEP_MAX];        //记录每条路线
int g_step_number = 0;
int g_sail_way_max = 0;
int g_crossing_history[3][STEP_MAX];
int g_sail_way[2][STEP_MAX];

int InitSailWay(int sailWay[2][100], int& sailWayMax, const int boatNumber)
{
    int miss_number = 0, barb_number = 0;        //初始化传教士和野蛮人的数量
    sailWayMax = 0;                                //初始化路线编号
    //设置路线
    while(miss_number != boatNumber+1 || barb_number != 0)
    {
        barb_number++;
        if(miss_number + barb_number <= boatNumber)
        {
            sailWay[0][sailWayMax] = miss_number;
            sailWay[1][sailWayMax] = barb_number;
            sailWayMax++;
        }
    }
    else
```

```

        {
            miss_number++;
            barb_number = 0;
            sailWay[0][sailWayMax] = miss_number;
            sailWay[1][sailWayMax] = barb_number;
            sailWayMax++;
        }
    }
    sailWayMax--; //减去多余的状态
    sailWay[0][sailWayMax] = 0;
    sailWay[1][sailWayMax] = 0;

    return 1;
}
//start g_step_number = 1;

int BoatGoOppositeBank(MyLeftBank& this_,int& sail_way,const int input_miss,const int
input_barb)
{
    //如果船在左岸
    if(LEFT_BANK == this_.boat_state)
    {
        //如果船能沿着这条路线航行
        if(this_.missionaries >= g_sail_way[0][sail_way] && this_.barbarians >=
g_sail_way[1][sail_way])
        {
            //改变传教士、野蛮人和船的状态
            this_.missionaries -= g_sail_way[0][sail_way];
            this_.barbarians -= g_sail_way[1][sail_way];
            this_.boat_state = (g_step_number + 1) % 2;
            step[g_step_number] = sail_way;
            return 1;
        }
        sail_way++;
        return 0;
    }
    //如果船在右岸
    else if(RIGHT_BANK == this_.boat_state)
    {
        int left_miss = this_.missionaries + g_sail_way[0][sail_way];
        int left_barb = this_.barbarians + g_sail_way[1][sail_way];
        if(left_miss <= input_miss && left_barb <= input_barb)
        {
            //改变传教士、野蛮人和船的状态

```

```

        this_.missionaries = left_miss;
        this_.barbarians = left_barb;
        this_.boat_state = (g_step_number + 1) % 2;
        return 1;
    }
    sail_way++;
    return 0;
}
else //船的状态出错
{
    cout << "\nBoat state goes wrong.\n";
    system("pause");
    return -1;
}
}

```

```

int JudgeBothBankState(const MyLeftBank this_,int& sail_way ,const int input_miss, const int
input_barb)
{
    for(int i=0;i<g_step_number;i++)//如果船的状态与前一个状态相同,则使用下一个航行方
    式航行
    {
        if(g_crossing_history[0][i] == this_.missionaries && g_crossing_history[1][i] ==
        this_.barbarians && g_crossing_history[2][i] == this_.boat_state)
        {
            return 0;
        }
    }
    //排除特殊情况: 野蛮人人数超过 0, 传教士人数为 0
    //barb_number>miss_number,但状态合法
    if((this_.missionaries == 0 && (this_.barbarians <= input_barb)) || (this_.missionaries ==
    input_miss && (this_.barbarians <= input_barb)) || (this_.missionaries == 0 && this_.barbarians
    == 0))
    {
        g_crossing_history[0][g_step_number] = this_.missionaries;
        g_crossing_history[1][g_step_number] = this_.barbarians;
        g_crossing_history[2][g_step_number] = this_.boat_state;
        step[g_step_number] = sail_way;

        sail_way = 0;
        g_step_number++;

        return 1;
    }
}

```

```

//检查两岸的状况
int right_miss = input_miss - this_.missionaries;
int right_barb = input_barb - this_.barbarians;
if((this_.missionaries >= this_.barbarians) && (right_miss >= right_barb))
{
    g_crossing_history[0][g_step_number] = this_.missionaries;
    g_crossing_history[1][g_step_number] = this_.barbarians;
    g_crossing_history[2][g_step_number] = this_.boat_state;
    step[g_step_number] = sail_way;

    sail_way = 0;
    g_step_number++;

    return 1;
}
else
{
    return 0;
}
}

int ReturnAvailableState(MyLeftBank& this_, int& sail_way,const int sail_way_max)
{
    //当到达目的地时，返回可用的路线
    if((0 == this_.boat_state) && (0 == this_.missionaries) && (0 == this_.barbarians))
    {
        g_step_number -= 2;
        this_.missionaries = g_crossing_history[0][g_step_number - 1];
        this_.barbarians = g_crossing_history[1][g_step_number - 1];
        this_.boat_state = (g_step_number) % 2;

        sail_way = step[g_step_number];
        sail_way++;
        return 1;
    }
    //恢复传教士、野蛮人和船的状态
    if(++sail_way<sail_way_max)
    {
        this_.missionaries = g_crossing_history[0][g_step_number-1];
        this_.barbarians = g_crossing_history[1][g_step_number-1];
        this_.boat_state = (g_step_number) % 2;
    }
    //返回到最后可用的路线状态
    while(sail_way>=sail_way_max)

```

```

{
    if(g_step_number-- > 1)
    {
        this_.missionaries = g_crossing_history[0][g_step_number-1];
        this_.barbarians = g_crossing_history[1][g_step_number - 1];
        this_.boat_state = (g_step_number) % 2;
        sail_way = step[g_step_number];
        sail_way++;
    }
    else return 0;
}
return 1;
}

```

```

void PrintingSailWay(const int successWay)

```

```

{
    cout << "\n 第" << successWay << "种方案: " << endl;
    for(int i=0;i<g_step_number;i++)
    {
        cout << '(' << g_crossing_history[0][i] << "," << g_crossing_history[1][i] << "," <<
g_crossing_history[2][i] << ')' << endl;
    }
}

```

```

void CrossRiver(MyLeftBank & leftbank, const int inputMissionaries, const int inputBarbarians,
const int inputBoatnumber)

```

```

{
    int sailWay = 0;
    int success_way = 0;
    InitSailWay(g_sail_way, g_sail_way_max, inputBoatnumber);
    while(!(g_step_number == 0 && sailWay >= g_sail_way_max))
    {
        while(!((0 == leftbank.boat_state) && (0 == leftbank.missionaries) && (0 ==
leftbank.barbarians)))
        {
            if(1 == BoatGoOppositeBank(leftbank, sailWay, inputMissionaries,
inputBarbarians))
            {
                if(0 == JudgeBothBankState(leftbank, sailWay, inputMissionaries,
inputBarbarians))
                if(0 == ReturnAvailableState(leftbank, sailWay, g_sail_way_max))
                    break;
            }
        }
    }
}

```



```

    }
    if(sailWay >= g_sail_way_max)
    {
        ReturnAvailableState(leftbank, sailWay, g_sail_way_max);
    }
}
if((0 == leftbank.boat_state) && (0 == leftbank.missionaries) && (0 ==
leftbank.barbarians))
{
    success_way++;
    PrintingSailWay(success_way);
    ReturnAvailableState(leftbank, sailWay, g_sail_way_max);
}
}
if(success_way == 0)
{
    cout << "\n 很遗憾，我们找不到方案\n\n";
}
else cout << "\n 我们总共找到了" << success_way << "种方案" << endl;
}

```

```

int main()
{
    int input_missionaries; //传教士人数
    int input_barbarians; //野蛮人人数
    int input_boat_number; //船的最大载人数

    //输入传教士人数、野蛮人人数和船的最大载人数
    cout << "请输入传教士人数:";
    cin >> input_missionaries;

    cout << "请输入野蛮人人数:";
    cin >> input_barbarians;

    cout << "请输入船的最大载人数:";
    cin >> input_boat_number;

    MyLeftBank left_bank;
    left_bank.missionaries = input_missionaries;
    left_bank.barbarians = input_barbarians;
    left_bank.boat_state = LEFT_BANK;

    memset(step, -1, STEP_MAX * sizeof(int)); //初始化 step[STEP_MAX] 为 -1(初始态)
}

```

```

//记录两岸的状态
g_crossing_history[0][g_step_number] = input_missionaries;
g_crossing_history[1][g_step_number] = input_barbarians;
g_crossing_history[2][g_step_number] = LEFT_BANK;
step[g_step_number] = 0;

g_step_number++;
CrossRiver(left_bank, input_missionaries, input_barbarians, input_boat_number);
}

```

附件 2：实验 2 代码 Astar8nums.cpp

```

#include<bits/stdc++.h>
using namespace std;

int open_cnt=0;
int close_cnt=0;
int noresult=0;
int open_node_cnt; //open 表节点个数

struct Node
{
    int a[3][3];
    int x,y;
    int f,g,h;
    int flag; //上一次移动方向
    Node *father;
}start,End;

struct Open_Close
{
    int f;
    Node *np;
}open[10000],close[10000];

//判断是否有解：逆序数之和奇偶性相同，有解
bool isable()
{
    int s[9],e[9];
    int tf=0,ef=0,k=0;
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            s[k]=start.a[i][j];

```

```

        e[k]=End.a[i][j];
        k++;
    }
}
for(int i=0;i<9;i++)
{
    for(int j=0;j<i;j++)
    {
        if (s[i]>s[j] && s[j]!=0) tf+=1;
        if (e[i]>e[j] && e[j]!=0) ef+=1;
    }
}
if((tf%2 == 1 && ef%2 == 1) || (tf%2 == 0 && ef%2 == 0)) return true;
else return false;
}

```

//求 h(n)

```

int a_start_h(Node *node)
{
    int old_x,old_y,End_x,End_y;
    int h=0;
    for(int k=1;k<9;k++)
    {
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                if(node->a[i][j] == k) //相应开始点的下标
                {
                    old_x=i;
                    old_y=j;
                }
                if(End.a[i][j] == k) //相应目标的结点下标
                {
                    End_x=i;
                    End_y=j;
                }
            }
        }
        //计算 h(n)
        h+=abs(old_x-End_x)+abs(old_y-End_y);
    }
    return h;
}

```

```

//输入
void input()
{
    cout<<"请输入起始图: "<<endl;
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            cin>>start.a[i][j];
            if(start.a[i][j] == 0)
            {
                start.x=i;
                start.y=j;
            }
        }
    }
    cout<<endl;
    cout<<"请输入目标图: "<<endl;
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            cin>>End.a[i][j];
            if(End.a[i][j] == 0)
            {
                End.x=i;
                End.y=j;
            }
        }
    }
    cout<<endl;

    start.g=0;
    start.h=a_start_h(&start);
    start.f=start.g+start.h;
}

```

```

//打印过程
int print(Node *node)
{
    Node *p=node;
    if(p == &start) return 1;
    else print(p->father);
}

```

```

        cout<<"-----\n";

        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                cout<<p->a[i][j]<<" ";
            }
            printf("\n");
        }
        cout<<"-----\n\n";
    }

//判断是否为目标节点
bool isend(Node *node)
{
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            if(node->a[i][j]!=End.a[i][j]) return false;
        }
    }
    return true;
}

//open 表排序
void sort(Open_Close *open)
{
    int min=99999,min_flag=0;
    Open_Close temp;
    for(int i=0;i<=open_cnt;i++)
    {
        if(min>open[i].f && open[i].f>0)
        {
            min=open[i].f;
            min_flag=i;
        }
    }
    temp=open[min_flag];
    open[min_flag]=open[0];
    open[0]=temp;
}

```

//向四个方向扩展

```
void move(int flag,Node *node)
```

```
{
```

```
    int temp;
```

```
    //左移
```

```
    if(flag == 1 && node->x>0)
```

```
    {
```

```
        Node *n=new Node();
```

```
        for(int i=0;i<3;i++)
```

```
        {
```

```
            for(int j=0;j<3;j++)
```

```
            {
```

```
                n->a[i][j]=node->a[i][j];
```

```
            }
```

```
        }
```

```
        n->a[node->x][node->y]=node->a[node->x-1][node->y];
```

```
        n->a[node->x-1][node->y]=0;
```

```
        n->x=node->x-1;
```

```
        n->y=node->y;
```

```
        n->flag=3;
```

```
        n->father=node;
```

```
        //求 g(n)
```

```
        n->g=node->g+1;
```

```
        n->h=a_start_h(n);
```

```
        //求 f(n)
```

```
        n->f=n->g+n->h;
```

```
        open_cnt++;
```

```
        open_node_cnt++;
```

```
        //添加到 open 表
```

```
        open[open_cnt].np=n;
```

```
        open[open_cnt].f=n->f;
```

```
    }
```

```
    //上移
```

```
    else if(flag == 2 && node->y<2)
```

```
    {
```

```
        Node *n=new Node();
```

```
        for(int i=0;i<3;i++)
```

```
        {
```

```
            for(int j=0;j<3;j++)
```

```
            {
```

```
                n->a[i][j]=node->a[i][j];
```

```
            }
```

```
        }
```

```

    }
    n->a[node->x][node->y]=node->a[node->x][node->y+1];
    n->a[node->x][node->y+1]=0;
    n->x=node->x;
    n->y=node->y + 1;
    n->flag=4;
    n->father=node;
    //求 g(n)
    n->g=node->g+1;

    n->h=a_start_h(n);
    //求 f(n)
    n->f=n->g+n->h;

    open_cnt++;
    open_node_cnt++;
    //添加到 open 表
    open[open_cnt].np=n;
    open[open_cnt].f=n->f;
}
//右移
else if(flag == 3 && node->x<2)
{
    Node *n=new Node();
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            n->a[i][j]=node->a[i][j];
        }
    }
    n->a[node->x][node->y]=node->a[node->x+1][node->y];
    n->a[node->x+1][node->y]=0;
    n->x=node->x + 1;
    n->y=node->y;
    n->flag=1;
    n->father=node;
    //求 g(n)
    n->g=node->g + 1;

    n->h=a_start_h(n);
    //求 f(n)
    n->f=n->g+n->h;

```

```

        open_cnt++;
        open_node_cnt++;
        //添加到 open 表
        open[open_cnt].np=n;
        open[open_cnt].f=n->f;
    }
    //下移
    else if(flag == 4 && node->y>0)
    {
        Node *n=new Node();
        for(int i=0;i<3;i++)
        {
            for(int j=0; j<3; j++)
            {
                n->a[i][j]=node->a[i][j];
            }
        }
        n->a[node->x][node->y]=node->a[node->x][node->y-1];
        n->a[node->x][node->y-1]=0;
        n->x=node->x;
        n->y=node->y-1;
        n->flag=2;
        n->father=node;
        //求 g(n)
        n->g=node->g+1;

        n->h=a_start_h(n);
        //求 f(n)
        n->f=n->g+n->h;

        open_cnt++;
        open_node_cnt++;
        //添加到 open 表
        open[open_cnt].np=n;
        open[open_cnt].f=n->f;
    }
}

```

//节点扩展

```

void expand(Node *node)
{
    for(int i=1;i<5;i++)
    {
        if(i!=node->flag) move(i,node);
    }
}

```



```

    }
}

int main()
{
    input();
    //start 放入 open 表
    open[0].np=&start;
    open_node_cnt=1;
    if(isable())
    {
        while(true)
        {
            //open 表不为空
            if(isend(open[0].np))
            {
                cout<<"\n 移动过程: \n";
                //打印过程
                print(open[0].np);
                //输出移动步数
                cout<<"该问题有解，一共用了"<<open[0].np->g<<"步"<<endl;
                break;
            }
            //扩展最优节点的子节点
            expand(open[0].np);
            open[0].np=NULL;
            open[0].f=-1;
            open_node_cnt--;
            //open 表排序
            sort(open);
        }
    }
    else cout<<"该问题无解";

    return(0);
}

```

附件 3：实验 3 代码 IrisBP.m

```

clear
clc
f=fopen('C:\Users\81228\Desktop\资料\人工智能\实验\实验 3\iris.data');
Iris=textscan(f,'%f%f%f%f%f%s','delimiter',' ');
fclose(f);

```

```

[~,n]=size(Iris);
for i=1:n-1
    data(:,i)=Iris{1,i};
end
[m,n]=size(data);
for i=1:m
    if strcmp(Iris{1,5}{i,1}, 'Iris-setosa')
        data(i,5:7)=[1 0 0];
    elseif strcmp(Iris{1,5}{i,1}, 'Iris-versicolor')
        data(i,5:7)=[0 1 0];
    else
        data(i,5:7)=[0 0 1];
    end
end

%数据归一化处理
x_max=max(data);
x_min=min(data);
data=(data-ones(m,1)*x_min)./(ones(m,1)*(x_max-x_min));

%划分训练样本和测试样本 大约 1/2 用作训练, 1/2 用作测试
num=round(m/2/3);
for i=1:3
    temp=data(1+50*(i-1):50*i,:);
    sel=randperm(50,num);
    test(1+num*(i-1):num*i,:)=temp(sel,1:4);
    test_label(1+num*(i-1):num*i,:)=temp(sel,5:7);
    temp(sel,:)=[];
    train(1+(50-num)*(i-1):(50-num)*i,:)=temp(:,1:4);
    train_label(1+(50-num)*(i-1):(50-num)*i,:)=temp(:,5:7);
end

[m,n]=size(train);
alpha=4;%输入神经元数目
beta=10;%隐层神经元数目
lamda=3;%输出神经元数目
rng('shuffle')
W1=rand(alpha,beta);%输入层和隐层之间的权值矩阵
W2=rand(beta,lamda);%隐层和输出层间的权值矩阵
B1=rand(1,beta);%隐层阈值矩阵
B2=rand(1,lamda);%输出层阈值矩阵
B22=B2;
W11=W1;
Eta=1;%学习率

```

```

iter_max=10000;%最大迭代次数
iter=1;
%BP 神经网络，每次仅针对一个训练样例更新连接权和阈值
while iter<=iter_max
    for i=1:m
        hidden_in=train(i,:)*W1;%隐层输入
        hidden_out=sigmod(hidden_in-B1);%隐层输出
        output_in=hidden_out*W2;%输出层输入
        output_out=sigmod(output_in-B2);%输出层输出
        %计算误差
        E(i)=sum((output_out-train_label(i,:)).^2);%求平方和可用 sumsqr 函数

        %更新参数,BP 神经网络中最核心的部分
        g=output_out.*(1-output_out).*(train_label(i,:)-output_out);%计算输出层神经元的梯度项
        e=hidden_out.*(1-hidden_out).*(g*W2');%计算隐层神经元的梯度项
        Deta_W2=Eta*hidden_out'*g;
        Deta_B2=-Eta*g;
        Deta_W1=Eta*train(i,:)'*e;
        Deta_B1=-Eta*e;
        W1=W1+Deta_W1;
        W2=W2+Deta_W2;
        B1=B1+Deta_B1;
        B2=B2+Deta_B2;
    end
    %计算训练集的累积误差
    E=mean(E);
    if E<1e-4 %目标误差
        iter;
        break
    end
    if mod(iter,1000)==0
        iter;
    end
    iter=iter+1;%更新迭代次数
end

%测试
[result,accuracy]=BP_test(test,test_label,W1,W2,B1,B2);
disp('结果为: ')
result'
disp(strcat('准确率为',num2str(accuracy*100),'%'))
function [result,accuracy]=BP_test(test,test_label,W1,W2,B1,B2)
%返回分类的结果
%accuracy 准确率

```

```

Hidden_in=test*W1;%隐层输入
Hidden_out=sigmoid(Hidden_in-B1);%隐层输出
Output_in=Hidden_out*W2;%输入层输入
Output_out=sigmoid(Output_in-B2);
[~,result]=max(Output_out,[],2);
[~,index2]=max(test_label,[],2);
right=sum(result==index2);%统计分类正确的个数
total=size(test,1);%总个数
accuracy=right/total;
end
%sigmoid 函数在 matlab 里是 logsig 函数
function [Y]=sigmoid(X)
Y=1./(1+exp(-1).^X);
end

```

附件 4：实验 4 代码 main.py

```

import numpy as np
from math import log
import operator
import matplotlib.pyplot as plt

# 定义文本框和箭头格式
decisionNode = dict(boxstyle="sawtooth", fc="0.8")
leafNode = dict(boxstyle="round4", fc="0.8")
arrow_args = dict(arrowstyle="<-")
test = "Iris-train.txt"
train = "Iris-test.txt"

# 读取数据函数，np.array 类型的测试数据集和训练数据集
# 已知数据集为 75*5 的大小
testData = np.loadtxt(test)
trainData = np.loadtxt(train)

# 计算信息熵
def calEnt(index, label, step):
    kind = [0, 0, 0]
    EntD = 0
    pk = [0, 0, 0]
    for n in range(step):
        if label[index[n]] == 0:
            kind[0] += 1
        elif label[index[n]] == 1:
            kind[1] += 1
        else:
            kind[2] += 1

```

```

for n in range(3):
    pk[n] = kind[n] / len(label)
    if pk[n] == 0:
        EntD += 0
    else:
        EntD -= pk[n] * log(pk[n], 2)
return EntD

# dataSet(array 类型)返回最大信息增益及其所在的属性和划分值
def bestFeature(dataSet):
    index = []
    for n in range(4):
        index.append(np.argsort(dataSet[:, n], axis=0))
    EntD = calEnt(range(len(dataSet[:, 0])), dataSet[:, -1], len(dataSet[:, 0]))
    T = [[], [], [], []] # 4 个连续属性的候选划分点
    Gain = [[], [], [], []] # 4 个连续属性的候选增益
    for n in range(len(dataSet[:, -1]) - 1):
        for i in range(4):
            T[i].append((dataSet[index[i][n], i] + dataSet[index[i][n + 1], i]) / 2)
            EntDv = [0, 0]
            for j in range(len(dataSet[:, 0]) - 1):
                EntDv[0] = calEnt(index[i][j + 1], dataSet[:, -1], j + 1)
                EntDv[1] = calEnt(index[i][j + 1:], dataSet[:, -1], len(index[i][j + 1:]))
                Gain[i].append(EntD - ((j + 1) / (len(dataSet[:, 0]) - 1)) * EntDv[0] - (
                    ((len(dataSet[:, 0]) - 1) - (j + 1)) / (len(dataSet[:, 0]) - 1)) * EntDv[1])
            MAX_Gain = [0, 0, 0, 0]
            MAX_index = [0, 0, 0, 0]
        for n in range(4):
            MAX_Gain[n] = max(Gain[n])
            MAX_index[n] = T[n][Gain[n].index(max(Gain[n]))]
    # 最大增益 划分值 所在的属性列
    return max(MAX_Gain), MAX_index[MAX_Gain.index(max(MAX_Gain))],
    MAX_Gain.index(max(MAX_Gain))

# 根据求得的最大增益, 属性, 划分值, 划分数据集
def splitDataSet(dataSet, value, line):
    smallDataSet = np.empty(shape=(0, 5))
    largeDataSet = np.empty(shape=(0, 5))
    for n in range(len(dataSet[:, -1])):
        if dataSet[n, line] < value:
            smallDataSet = np.append(smallDataSet, [dataSet[n, :]], axis=0)
        else:
            largeDataSet = np.append(largeDataSet, [dataSet[n, :]], axis=0)
    return smallDataSet, largeDataSet

```

```

def majorityCnt(classList):
    classCount = {}
    for vote in classList:
        if vote not in classCount.keys(): classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount

```

递归

```

def createTree(dataSet):
    classList = dataSet[:, -1].tolist()
    # 以下返回类别完全相同的
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    # 遍历完所有特征时返回出现次数最多的
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)
    MaxGain, value, line = bestFeature(dataSet)
    myTree = {line: {}}
    small, large = splitDataSet(dataSet, value, line)
    myTree[line]["-" + str(value)] = createTree(small)
    myTree[line]["+" + str(value)] = createTree(large)
    return myTree

```

获取树的叶子数

```

def getNumLeafs(myTree):
    numLeafs = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        # 测试节点的数据类型是否为字典
        if type(secondDict[key]).__name__ == 'dict':
            numLeafs += getNumLeafs(secondDict[key])
        else:
            numLeafs += 1
    return numLeafs

```

获取树的深度

```

def getTreeDepth(myTree):
    maxDepth = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():

```

```

        if type(secondDict[key]).__name__ == 'dict':
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:
            thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth
    return maxDepth

def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    createPlot.ax1.annotate(nodeTxt, xy=parentPt, xycoords='axes fraction',
                             xytext=centerPt, textcoords='axes fraction',
                             va="center", ha="center", bbox=nodeType,
    arrowprops=arrow_args)

# 在父子节点中填充文本信息
def plotMidText(cnrPt, parentPt, txtString):
    xMid = (parentPt[0] - cnrPt[0]) / 2.0 + cnrPt[0]
    yMid = (parentPt[1] - cnrPt[1]) / 2.0 + cnrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center", rotation=30)

def plotTree(myTree, parentPt, nodeTxt):
    # 计算宽和高
    numLeafs = getNumLeafs(myTree)
    depth = getTreeDepth(myTree)
    firstStr = list(myTree.keys())[0]
    cnrPt = (plotTree.xOff + (1.0 + float(numLeafs)) / 2.0 / plotTree.totalW, plotTree.yOff)
    # 标记子节点属性值
    plotMidText(cnrPt, parentPt, nodeTxt)
    plotNode(firstStr, cnrPt, parentPt, decisionNode)
    secondDict = myTree[firstStr]
    # 减小 y 偏移
    plotTree.yOff = plotTree.yOff - 1.0 / plotTree.totalD
    for key in secondDict.keys():
        if type(secondDict[key]).__name__ == 'dict':
            plotTree(secondDict[key], cnrPt, str(key))
        else:
            plotTree.xOff = plotTree.xOff + 1.0 / plotTree.totalW
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cnrPt, leafNode)
            plotMidText((plotTree.xOff, plotTree.yOff), cnrPt, str(key))
    plotTree.yOff = plotTree.yOff + 1.0 / plotTree.totalD

def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])

```

```

createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)
plotTree.totalW = float(getNumLeafs(inTree))
plotTree.totalD = float(getTreeDepth(inTree))
plotTree.xOff = -0.5 / plotTree.totalW;
plotTree.yOff = 1.0;
plotTree(inTree, (0.5, 1.0), "")
plt.show()

def retrieveTree(i):
    listOfTrees = [{ 'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}},
                    { 'no surfacing': {0: 'no', 1: {'flippers': {0: {'head': {0: 'no', 1: 'yes'}}, 1:
'no'}}}}
                    ]
    return listOfTrees[i]

def classify(myTree, testData):
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    testVec = testData
    # 将标签字符串转换为索引
    featIndex = [0, 1, 2, 3].index(firstStr)
    for key in list(secondDict.keys()):
        if float(key) < 0:
            if testVec[featIndex] <= -float(key):
                if type(secondDict[key]).__name__ == 'dict':
                    classLabel = classify(secondDict[key], testVec)
                else:
                    classLabel = secondDict[key]
            else:
                if testVec[featIndex] >= float(key):
                    if type(secondDict[key]).__name__ == 'dict':
                        classLabel = classify(secondDict[key], testVec)
                    else:
                        classLabel = secondDict[key]
    return classLabel

percent = 0
myTree = createTree(trainData)
for i in range(len(testData[:, -1])):
    if classify(myTree, testData[i, :]) == testData[i, -1]:
        percent += 1
print("正确率:" + str(percent / len(testData[:, -1])))

createPlot(myTree)

```