**《人工智能》**

**实 验 报 告**

**班级：**

**学号：**

**姓名：**

**提交日期：2024.6.26**

**目 录**

**实验一 过河问题**

**实验二 九宫重排**

**实验三 专家系统**

**实验一 过河问题**

◎问题描述

有三个牧师和三个野人过河，只有一条能装下两个人的船，在河的任何一方或者船上，如果野人的人数大于牧师的认输，那么牧师就会有危险.找出一种按的渡河方法。

将该问题转变为：假如有n个牧师和n个野人准备渡河，但只有一条能容纳c个人的小船，为了防止野人侵犯牧师，要求无论在何处，牧师的人数不得少于野人的人数（除非牧师人数为0），且假定两种人都会划船，试设计一个算法，确定它们能否渡过河去，若能，则给出一只小船来回次数最少的最佳方案。

◎基本要求

输入：牧师人数（即野人人数）n，小船一次至多载客人数c。

输出：若问题无解，则显示“渡河失败”信息，否则，输出一组最佳方案，用三组（X1，X2，X3）表示渡河过程中的状态。并用箭头输出这些状态之间的迁移：目的状态<- <-中间状态 <- <-初始状态。

例：当n=2,c=2时，输出 000<-021<-211<-110<-221

上述(X1,X1,X2)表示渡河过程中各个状态。

其中：X1表示始岸上牧师人数，X2表示始岸上野人人数，

X3表示小船位置，（0-在目的岸，1-在起始岸 ）

◎算法描述

（1）算法基本思想的文字描述；

从初始状态S(n,n,1)出发，形成的有合法且未达状态S11、S12、……、Sli。再分别从S11、S12、……、Sli出发形成所有合法而未达状态S111、S112、…… 、Sli1、Sli2、Sli ……最终达到目标（0，0，0）（有解），或者找不到合法而未达状态（无解）。若有解，则从目标返回找前趋状态，前趋状态的前趋状态……直到初始状态。

（2）判别（X1，X2，X3）为合法状态条件：X1=0或X1=n或X1=X2。

（3）数据结构：

1 栈STACK，记下“已达”状态及踪迹，并兼作队列。

1. 已达
2. 未达

2 STATE[X1][X2]=

（4）算法基本思想的具体实现：

1 初始化：置STATE[N+1][N+1][2]中的有状态为“未达”

置队列STACK空，cond为当前是否已达到目标：

1. 未达目标
2. 已达到目标

cond=

cond置初值

2 以S（n,n,1）为始点，置STATE为“已达”。S入队列STACK

3 while（队列STACK空且未达到目标时）

A{ 取出队头元素地址=>p1，队头元素出队列

B while（未达到目标，且P1有可达、合法、且未到达过的相邻顶点Q）

if (Q=(000) 则{cond=1,Q入队列}

否则 {置QW为“已达”，Q入队列}

/\* B可用函数COMBINE实现 \*/

1. if (cond=1)则按队列中前趋指针指示的次序依次输出序列，否则输出“渡河失败”。
2. COMBINE函数的功能等价于从数量不等的物品，分别选出1件、2件、……C件物品的所有组合，同时对每一种组合确定其合法性。

COMBINE（ ）

{ 1 栈SP初始化（SP存放已放入物品序号），NUM为已取出物品个数，NUM=0，i为准备取出物品序号，i<=1。

2 do {

while (未达到目标，且所有物品还未取尽，且NUM<C)

{若该种物品已取尽，则取下一种,i++;

取出第i种物品中一件来，该物来序号（即i）进栈，NUM++;

判断该状态合法否？！ /\* 用函数dicision实现 \*/

}

if (未达到目标，且栈SP不空)

{则读栈SP=>i，将第种物品放回一件：NUM--:退栈；i++;}

}while(未达到目标，且并非所有情况均已列举完)

}

dicision ( )

{ if (当前状态（x1,x2,x3）合法且未达)

则(x1,x2,x3)及前趋指针入队列STACK；

if ((x1,x2,x3)==0,0,0)) 则 cond=1;

}

◎程序如下：

#include <iostream>

#include <vector>

#include <map>

#include <stack>

#include <queue>

#include <set>

using namespace std;

struct state{

    int x,y;

    bool f;

};

bool operator<(const state& s1,const state& s2){

    if(s1.x!=s2.x)return s1.x<s2.x;

    if(s1.y!=s2.y)return s1.y<s2.y;

    if(s1.f!=s2.f)return s1.f<s2.f;

    return false;

}

bool operator==(const state& s1,const state& s2){

    return s1.x==s2.x && s1.y==s2.y && s1.f==s2.f;

}

const bool L = 1, R = 0;

struct op{

    int dx,dy;

};

state s,e;

vector<op>ops;

int n,c;

int m=0;

map<state,int>st\_to\_i;

map<int,state>i\_to\_st;

set<pair<int,int>>cnt\_edge;

vector<int>g[(size\_t)1e7];

bool check(int x,int y){

    return x>=0 && x<=n && y>=0 && y<=n;

}

void build(int u){

    auto[x,y,f]=i\_to\_st[u];

    for(auto[dx,dy]:ops){

        int p=f?-1:1;

        int nx=x+p\*dx,ny=y+p\*dy;

        if(!check(nx,ny)) continue;

        state temp={nx,ny,f^1};

        if( (nx>=ny || nx==0) && (n-nx >= n-ny || n-nx == 0)){

            if(st\_to\_i.count(temp)==0){

                int v=++m;

                st\_to\_i.insert({temp,v});

                i\_to\_st.insert({v,temp});

                cnt\_edge.insert(make\_pair(u,v));

                cnt\_edge.insert(make\_pair(v,u));

                g[u].push\_back(v);

                g[v].push\_back(u);

                build(v);

            }else{

                int v=st\_to\_i[temp];

                if(cnt\_edge.count(make\_pair(u,v))!=0)

                    continue;

                cnt\_edge.insert(make\_pair(u,v));

                cnt\_edge.insert(make\_pair(v,u));

                g[u].push\_back(v);

                g[v].push\_back(u);

            }

        }

    }

}

vector<int>pref;

vector<int>dis;

typedef pair<int,int> PII;

void dijkstra(){

    priority\_queue<PII,vector<PII>,greater<PII>>pq;

    dis[0]=0;

    pq.emplace(dis[0],0);

    while(!pq.empty()){

        auto[d,u]=pq.top();pq.pop();

        if(dis[u]<d)continue;

        for(int i=0;i<g[u].size();i++){

            int v=g[u][i],w=1;

            if(dis[v]>dis[u]+w){

                pref[v]=u;

                dis[v]=dis[u]+w;

                pq.emplace(dis[v],v);

            }

        }

    }

}

void printPath(){

    int t=st\_to\_i[e];

    stack<int>stc;

    while(t!=0){

        stc.push(t);

        t=pref[t];

    }

    stc.push(0);

    vector<state>ls;

    while(!stc.empty()){

        ls.push\_back(i\_to\_st[stc.top()]);

        stc.pop();

    }

    string s[2]={"回来","去对岸"};

    for(int i=1;i<ls.size();i++){

        if(i&1){

            cout<<ls[i-1].x<<"传教士 "<<ls[i-1].y<<"野人--->";

            cout<<ls[i-1].x-ls[i].x<<"个传教士和"<<ls[i-1].y-ls[i].y<<"个野人"<<s[i&1]<<"--->";

            cout<<ls[i].x<<"传教士 "<<ls[i].y<<"野人\n";

        }else{

            cout<<ls[i-1].x<<"传教士 "<<ls[i-1].y<<"野人--->";

            cout<<ls[i].x-ls[i-1].x<<"个传教士和"<<ls[i].y-ls[i-1].y<<"个野人"<<s[i&1]<<"--->";

            cout<<ls[i].x<<"传教士 "<<ls[i].y<<"野人\n";

        }

    }

}

int main(){

#ifndef \_\_linux\_\_

    system("chcp 65001");

#endif

    cin>>n>>c;

    if(c>=2\*n){

        cout<<"共需要渡河"<<1<<"次\n";

        return 0;

    }else if(c>=n && c<2\*n){

        cout<<"共需要渡河"<<5<<"次\n";

        return 0;

    }

    s={n,n,L};

    e={0,0,R};

    st\_to\_i.insert({s,0}),i\_to\_st.insert({0,s});

    for(int i=0;i<=c;i++){

        for(int j=0;j<=c;j++){

            if(i==0 && j==0) continue;

            if(i+j>c) continue;

            if(i>=j || i==0){

                ops.push\_back({i,j});

                ops.push\_back({i,j});

            }

        }

    }

    build(0);

    if(st\_to\_i.count(e)==0){

        cout<<-1<<'\n';

        return 0;

    }

    pref.resize(m+1);

    dis.resize(m+1,1e9);

    dijkstra();

    printPath();

    cout<<"共需要渡河"<<dis[st\_to\_i[e]]<<"次\n";

    return 0;

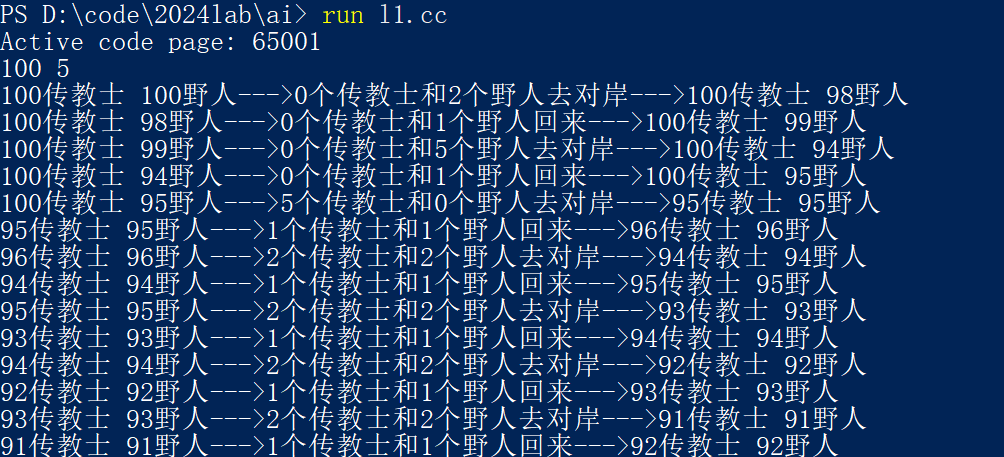
}

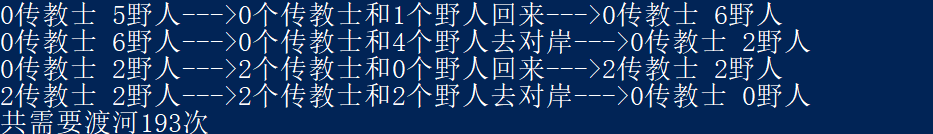
◎程序运行结果：

运行第一组测试样例n=100,c=5,其中n代表传教士和野人的数量，c代表船上最多容纳人数。

程序首先枚举每一步可行的状态建图，在可转移的状态之间建立一条有向边，就形成了一个有向图。在有向图上使用dijkstra算法搜索出从起始状态(n,n,Left)到终止状态(0,0,Right)的最短路径，并记录路径。存入栈中，最后从栈中取出最短路径经过的节点打印每一个经过的状态节点。

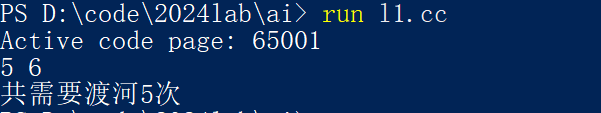
n=100,c=5运行结果为193



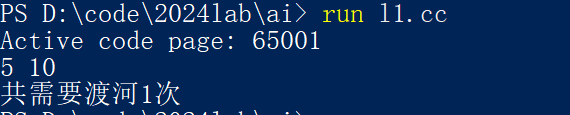


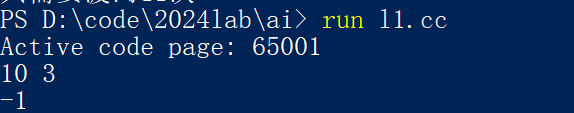
注意到当时结果恒为5且策略一致。即：

野人全部渡河->一个野人回来->全部传教士渡河->一个传教士回来->一个传教士和一个野人渡河



注意到当时所有传教士和野人可以直接渡河，结果为1。





◎实验心得：

在完成传教士与野人过河问题的人工智能实验中，我学到了以下几点：

1. **实验方法**：通过枚举每一步的可行状态建立有向图，并使用 Dijkstra 算法搜索最短路径，从起始状态(n, n, Left)到终止状态(0, 0, Right)。
2. **特殊情况处理**：
   * 当时时，结果恒为5且策略一致。
   * 当 时，所有传教士和野人可以直接渡河，结果为1。
3. **算法优化**：合理剪枝和优化状态空间，提高了算法效率，使用栈记录路径简化了存储和输出过程。

实验结果表明，这种方法能高效解决传教士与野人过河问题，并找到最优解，清晰展示每一步状态变化。通过本次实验，我深入理解了图算法、最短路径搜索及状态空间表示。

**实验二 九宫重排**

* 问题描述

利用A\*算法实现八数码难题(九宫重排)的搜索。

* 算法描述

procedure heuristic\_search

open ：=[start] ； closed：=[ ] ；f(s) ：= g(s) + h(s) ；

while open != [ ] do

begin

从open 表中删除第一个状态，称为n ；

if n = 目的状态 then return (success) ；

生成n 的所有子状态；

if n 没有任何子状态 then continue ；

for n 的每个子状态 do

case 子状态 is not already on open 表 or closed 表 ：

begin

计算该子状态的估价函数值；

将该子状态加到open 表中；

end ；

case 子状态 is already on open 表 ：

if 该子状态是沿着一条比在open 表已有的更短路径而到达

then 记录更短路径走向及其估价函数值；

case 子状态 is already on closed 表：

if 该子状态是沿着一条比在closed 表已有的更短路径而到达

begin

将该子状态从closed 表移到 open 表中；

记录更短路径走向及其估价函数值；

end ；

case end ；

将n 放入closed 表中；

根据估价函数值，从小到大重新派列open 表；

end ；

return （failure）；

end

应当注意：定义h\*(n)为状态n到目的状态的最优路径的代价。

则，估价函数f(n)中的h(n)应满足：h(n)≤h\*(n)

* 程序如下

#include <iostream>

#include <vector>

#include <functional>

#include <map>

#include <queue>

using namespace std;

int init\_st[3][3]={

    {1,2,3},

    {4,0,6},

    {7,5,8}

};

int fin\_st[3][3]={

    {1,2,3},

    {4,5,6},

    {7,8,0}

};

class state{

public:

    state(){}

    state(int data[3][3]){

        for(int i=0;i<3;i++)

            for(int j=0;j<3;j++)

                this->data[i][j]=data[i][j];

    }

    state(const state&st){

        for(int i=0;i<3;i++)

            for(int j=0;j<3;j++)

                this->at(i,j)=st.at(i,j);

    }

    vector<state> nextStates()const{

        vector<state>res;

        auto vaild=[](int x,int y){

            return x>=0 && x<3 && y>=0 && y<3;

        };

        auto[nx,ny]=this->findCeil(0);

        int dx[4]={0,1,0,-1},dy[4]={1,0,-1,0};

        for(int i=0;i<4;i++){

            if(vaild(nx+dx[i],ny+dy[i])){

                state st(\*this);

                swap(st.at(nx,ny),st.at(nx+dx[i],ny+dy[i]));

                res.push\_back(st);

            }

        }

        return res;

    }

    vector<state> optNextStates()const{

        auto res=nextStates();

        sort(res.begin(),res.end(),[](const state&s1,const state&s2){

            return s1.heurFunc()<s2.heurFunc();

        });

        return res;

    }

    int& at(int x,int y){

        return this->data[x][y];

    }

    int at(int x,int y)const{

        return this->data[x][y];

    }

    state& operator=(const state&st){

        for(int i=0;i<3;i++)

            for(int j=0;j<3;j++)

                this->at(i,j)=st.at(i,j);

        return \*this;

    }

    bool operator<(const state& st)const{

        return this->getId()<st.getId();

    }

    bool operator==(const state& st)const{

        return this->getId()==st.getId();

    }

    pair<int,int> findCeil(int x)const{

        if(x<0 || x>8)

            return {-1,-1};

        for(int i=0;i<3;i++)

            for(int j=0;j<3;j++)

                if(this->at(i,j)==x)

                    return {i,j};

        return {-1,-1};

    }

    void showState()const{

        printf("%d|%d|%d\n",this->at(0,0),this->at(0,1),this->at(0,2));

        printf("%d|%d|%d\n",this->at(1,0),this->at(1,1),this->at(1,2));

        printf("%d|%d|%d\n",this->at(2,0),this->at(2,1),this->at(2,2));

    }

    int heurFunc()const{

        int res=0;

        for(int i=0;i<3;i++){

            for(int j=0;j<3;j++){

                if(fin\_st[i][j]==0)

                    continue;

                auto[ni,nj]=this->findCeil(fin\_st[i][j]);

                res+=abs(ni-i)+abs(nj-j);

            }

        }

        for(int i=0;i<3;i++){

            for(int j=0;j<3;j++){

                if(init\_st[i][j]==0)

                    continue;

                auto[ni,nj]=this->findCeil(init\_st[i][j]);

                res+=abs(ni-i)+abs(nj-j);

            }

        }

        return res;

    }

    int getId()const{

        int res=0;

        int cnt=1;

        for(int i=0;i<3;i++){

            for(int j=0;j<3;j++){

                res+=this->at(i,j)\*cnt;

                cnt\*=10;

            }

        }

        return res;

    }

    string get\_op(const state& nextState){

        auto[cur\_x,cur\_y]=this->findCeil(0);

        auto[next\_x,next\_y]=nextState.findCeil(0);

        if(cur\_x==next\_x){

            if(cur\_y<next\_y)

                return "left";

            else

                return "right";

        }else{

            if(cur\_x<next\_x)

                return "up";

            else

                return "down";

        }

    }

    bool is\_odd(){

        int arr[9];

        for(int i=0;i<3;i++)

            for(int j=0;j<3;j++)

                arr[i\*3+j]=this->at(i,j);

        int cnt=0;

        for(int i=0;i<9;i++){

            for(int j=i-1;j>=0;j--){

                if(arr[i]==0||arr[j]==0)

                    continue;

                cnt+=arr[j]>arr[i];

            }

        }

        return cnt&1;

    }

private:

    int data[3][3];

};

bool operator==(const pair<state,pair<int,string>>& s1,const pair<state,pair<int,string>>& s2){

    return s1.first==s2.first;

}

int main(){

#ifndef \_\_linux\_\_

    system("chcp 65001");

#endif

    printf("输入3\*3个数字:\n");

    for(int i=0;i<9;i++) cin>>init\_st[i/3][i%3];

    state firstState(init\_st);

    state finalState(fin\_st);

    if(firstState.is\_odd()^finalState.is\_odd()){

        printf("No Answer\n");

        return 0;

    }

    vector<pair<state,pair<int,string>>>open;

    map<state,int>close;

    open.push\_back({firstState,{0,""}});

    while(!open.empty()){

        auto min\_iter=min\_element(open.begin(),open.end(),[](const pair<state,pair<int,string>>& s1,const pair<state,pair<int,string>>& s2){

            return s1.first.heurFunc()<s2.first.heurFunc();

        });

        auto[curState,info]=\*min\_iter;

        auto[step,curOp]=info;

        open.erase(min\_iter);

        printf("------------------\nstep:%d\noperator:%s\n",step,curOp.c\_str());

        curState.showState();

        printf("------------------\n");

        if(curState==finalState){

            printf("%d step to get answer\n",step);

            break;

        }

        for(const auto &nextState:curState.nextStates()){

            auto iter\_close=close.find(nextState);

            auto iter\_open=find(open.begin(),open.end(),make\_pair(curState,make\_pair(step,"")));

            if(iter\_close==close.end() && iter\_open==open.end()){

                open.push\_back({nextState,{step+1,curState.get\_op(nextState)}});

            }else if(iter\_open!=open.end() && iter\_open->second.first > step+1){

                iter\_open->second.first=step+1;

                iter\_open->second.second=curState.get\_op(nextState);

            }else if(iter\_close!=close.end() && iter\_close->second > step+1){

                open.push\_back({nextState,{step+1,curState.get\_op(nextState)}});

                close.erase(iter\_close);

            }

        }

        close.insert({curState,step});

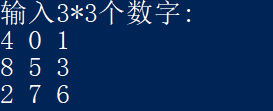
    }

    return 0;

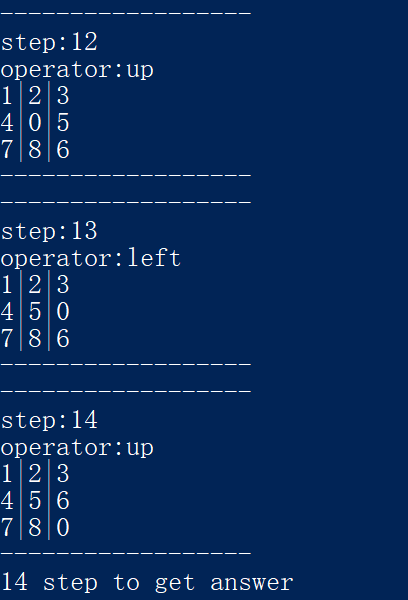
}

* 程序运行结果如下

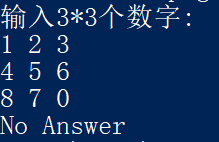
输入3\*3个数字表示起始状态，终止状态为排列后的九宫格。



程序会基于A\*算法的启发函数值，选择收益最大的一条路径进行搜索，同时打印每一步的状态。以上测试用例输出结果为14。



为了解决结果不存在而产生的程序无法正常终止的情况，我们在搜索前根据数字序列{}的逆序数的奇偶性进行判断，如果起始状态和终止状态数字序列逆序数同为奇数或同为偶数则说明存在解，否则不存在解。



◎实验心得

在使用A\*算法解决九宫格重排问题的实验中，我学到了以下几点：

无解判断：在搜索前，通过计算起始状态和终止状态数字序列的逆序数。如果两者的逆序数同为奇数或同为偶数，则说明存在解，否则不存在解，从而避免程序无法正常终止的问题。

启发式搜索：使用A\*算法结合启发式函数（从初始节点到当前节点的代价加上从当前节点到目标节点的曼哈顿距离）引导搜索方向，提升了搜索效率。

状态记录与重复检测：记录已访问的状态，避免重复搜索，进一步优化了算法性能。

实验结果表明，A算法在九宫格重排问题中高效且准确。此次实验让我深刻理解了A算法及其在实际问题中的应用，并通过逆序数判断无解情况，提高了算法的健壮性。

**实验三 专家系统**

* 问题描述

仿照书中例题，设计一个简单的电脑故障维修专家系统或者实现简单的动物识别专家系统。

例如，动物识别系统（王士同书上的写法）如下：

规则I1到I4这一组规则可用于把哺乳动物和鸟类动物区分开：

* 规则I1 如果 该动物有毛发，那么 它是哺乳动物
* 规则I2 如果 该动物能产乳，那么 它是哺乳动物
* 规则I3 如果 该动物有羽毛，那么 它是鸟类动物
* 规则I4 如果 该动物能飞行，它能生蛋，那么 它是鸟类动物

规则I5到I8把哺乳动物又分为更细的类食肉动物和有蹄动物：

* 规则I5 如果 该动物是哺乳动物，它吃肉，那么 它是食肉动物
* 规则I6 如果 该动物是哺乳动物，它长有爪子，它长有利齿，它眼睛前视，那么 它是食肉动物
* 规则I7 如果 该动物是哺乳动物，它长有蹄，那么 它是有蹄动物
* 规则I8 如果 该动物是哺乳动物，它反刍， 那么 它是有蹄动物。（并且是偶蹄动物）

以下两个规则对食肉动物进行细分：

规则I9 如果 该动物是食肉动物，它的颜色是黄褐色，它有深色的斑点 那么 它是猎豹

规则I10 如果 该动物是食肉动物，它的颜色是黄褐色，它有黑色条纹 那么 它是老虎

以下两个规则对有蹄动物进行细分：

规则I11 如果 该动物是有蹄动物，它有长腿，它有长颈，它的颜色是黄褐色，它有深色的斑点， 那么 它是长颈鹿

规则I12 如果 该动物是有蹄动物，它的颜色是白的，它有黑色条纹， 那么 它是斑马

以下对鸟类进行分类的规则：

规则I13 如果 该动物是鸟类，它不会飞，它有长腿，它有长颈，它的颜色是黑、白色相杂，那么 它是鸵鸟

*（规则I13的IF部分的条件“它有长腿”和“它有长颈”，也出现在规则I11的IF部分。I11是有蹄动物的，而I13是鸟的分类，无混淆）*

规则I14 如果 该动物是鸟类，它不能飞行，它能游水，它的颜色是黑色和白色，那么 它是企鹅

规则I15 如果 该动物是鸟类，它善于飞行，那么 它是信天翁（海燕）



识别长颈鹿的过程（con’d)：

* 假设给出的数据库中存放以下事实：

动物有暗斑，有长脖子，有长腿，有奶，有蹄。

* 求解目标：该动物是什么动物?
* 开始，观察到：（动物的颜色是黄褐色,深色斑点）规则I11还是规则I9？再看到该动物给它的幼兽喂奶，并能反刍，于是事实库内容增为：（动物的颜色是黄褐色,深色斑点,能产乳,反刍）
* 现用规则集与事实库进行匹配，I2首先可用，并更新事实库为：（哺乳动物,黄褐色,深色斑点,能产乳,反刍）
* 进而I8又能用，更新事实库为：

（有蹄动物,偶蹄动物,哺乳动物,黄褐色,深色斑点,能产乳,反刍）

* 还无法识别，而事实库也不能和其它规则的前提相匹配，需再观察，进一步发现该动物腿和颈都很长，即得到事实库：

（动物有长腿,有长颈,有蹄动物,偶蹄动物,哺乳动物,黄褐色,深色斑点,能产乳,反刍）

* 规则I11可使用，推理出该动物为长颈鹿
* 程序如下

#include <map>

#include <vector>

#include <iostream>

#include <functional>

#include <sstream>

using namespace std;

map<int,string>feature{

    {0,"毛发"},{1,"产乳"},{2,"羽毛"},{3,"飞行"},{4,"生蛋"},{5,"吃肉"},{6,"爪子"},{7,"利齿"},{8,"前视"},{9,"有蹄"},{10,"反刍"},{11,"黄褐色"},{12,"深色斑点"},{13,"黑色条纹"},{14,"长腿"},{15,"长颈"},{16,"白色"},{17,"黑白相杂"},{18,"游水"},{19,"黑色"}

};

map<int,string>animal{

    {-1,"无法推断"},{0,"动物"},{1,"哺乳动物"},{2,"鸟类动物"},{3,"食肉动物"},{4,"有蹄动物"},{5,"猎豹"},{6,"老虎"},{7,"长颈鹿"},{8,"斑马"},{9,"鸵鸟"},{10,"企鹅"},{11,"海燕"}

};

vector<function<bool(const vector<bool>&)>>checks{

    [](const vector<bool>& v){return true;},//动物

    [](const vector<bool>& v){return v[0] || v[1];},//哺乳动物

    [](const vector<bool>& v){return v[2] || (v[3] && v[4]);},//鸟类动物

    [](const vector<bool>& v){return v[5] || (v[6] && v[7] && v[8]);},//食肉动物

    [](const vector<bool>& v){return v[9] || v[10];},//有蹄动物 4

    [](const vector<bool>& v){return v[11] && v[12];},//猎豹

    [](const vector<bool>& v){return v[11] && v[13];},//老虎

    [](const vector<bool>& v){return v[14] && v[15] && v[11] && v[12];},//长颈鹿

    [](const vector<bool>& v){return v[13] && v[16];},//斑马 8

    [](const vector<bool>& v){return !v[3] && v[14] && v[15] && v[17];},//鸵鸟

    [](const vector<bool>& v){return !v[3] && v[18] && v[16] && v[19];},//企鹅

    [](const vector<bool>& v){return v[3];},//信天翁

};

int solve(const vector<vector<int>>&tree,const vector<bool>&v){

    bool ok=0;

    int ans=-1;

    function<void(int)>dfs=[&](int a){

        if(ok) return;

        printf("这是%s->",animal[a].c\_str());

        if(tree[a].size()==0){

            ans=a,ok=1;

            return;

        }

        vector<int>nxt;

        for(auto b:tree[a])

            if(checks[b](v)) nxt.push\_back(b);

        if(nxt.size()!=1){

            printf("我不知道\n");

            ok=1;

            return;

        }

        dfs(nxt[0]);

    };

    dfs(0);

    return ans;

}

void printPage(){

#ifndef \_\_linux\_\_

    system("chcp 65001");

#endif

    int i=0;

    for(auto[k,v]:feature){

        printf("[编号:%d 特征:%s]\t",k,v.c\_str());

        if((++i)%5==0) printf("\n");

    }

    printf("\n");

    printf("--一行内输入动物具有的特征编号，以空格分割--\n");

}

vector<bool> load(){

    vector<bool>v(20);

    string s;getline(cin,s);

    stringstream sst(s);

    int x;

    while(sst>>x)

        v[x]=1;

    return v;

}

int main(){

    printPage();

    vector<bool>v=load();

    vector<vector<int>>tree(12);

    tree[0]={1,2},tree[1]={3,4},tree[2]={9,10,11},tree[3]={5,6},tree[4]={7,8};

    int res=solve(tree,v);

    printf("推理结果:%s",animal[res].c\_str());

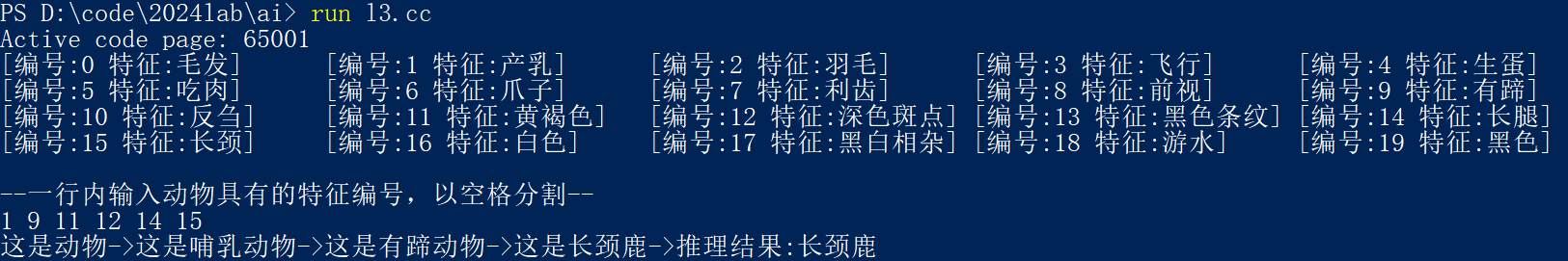
    return 0;

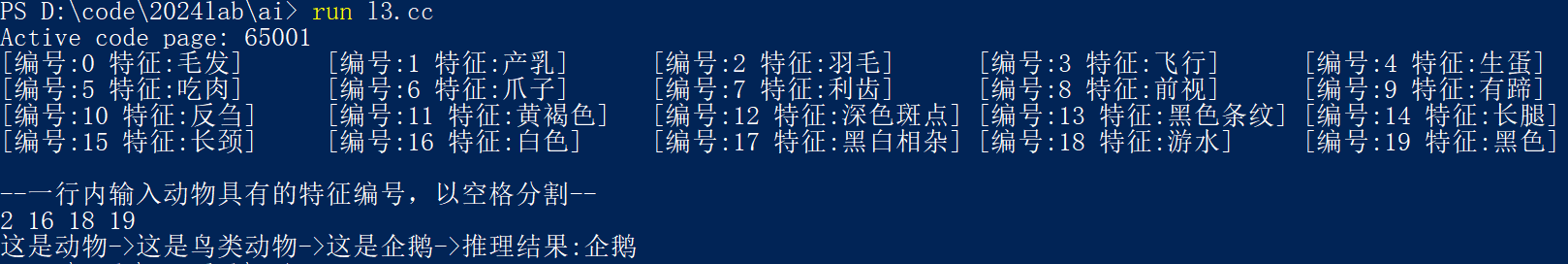
}

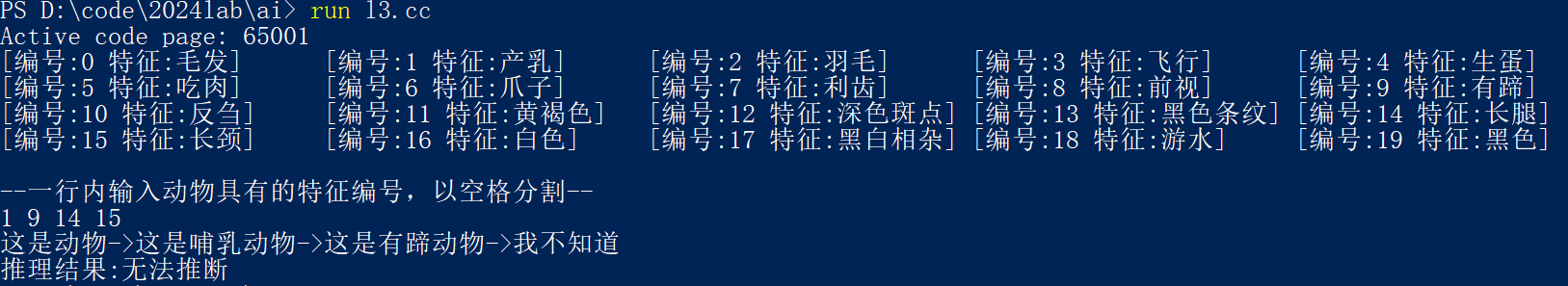
* 程序运行结果如下

根据上述对动物的描述，我们提取出20种特征，并使用0-19数字进行编码。并构建一个特征向量。

然后构建出用于识别动物的决策树，并为树上的每一个节点赋予一个检测特征向量的函数。在执行时只需输入动物拥有的特征，构建特征向量后对决策树进行遍历，当遍历到叶子结点时说明找到了一个动物，否则就无法判断。运行结果如下。







◎实验心得：

在设计识别动物的专家系统过程中，我们提取了20种特征并进行了编码，构建了特征向量 。通过决策树实现识别，每个节点对应一个检测特征的函数。用户输入动物特征后，系统通过遍历决策树进行识别，最终在叶子节点确定动物种类。该方法简洁有效，提升了识别准确性，增强了我对专家系统和人工智能应用的理解。