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

电 子 科 技 大 学

实 验 报 告

学生姓名：张镕麒 **学 号：**2019081301026 **指导教师：**康昭

实验地点：主楼 A2-413-2 **实验时间：**2021/11/28

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：5 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$h(n) \le h^*(n)$，$h^*(n)$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．设计搜索状态。

首先将八数码的 3*3 矩阵转化成长度为 9 的数列。大致有三种设计哈希函数思路。

（1）使用经典哈希函数。如线性哈希函数 f[i]=(f[i-1]*base+a[i])%mod，再结合 STL 中的 map，将原数列 vector 类型映射到 int 类型，同时记录反映射，即 int 类型到 vector 类型。从而实现双射，使用对应的哈希函数即可获取原状态。优点：思路简单、模型简单。缺点：时空复杂度较高。

（2）使用康托展开。由于本问题状态集大小为 9 的阶乘，状态与每一种排列一一对应，则可通过康托展开计算出唯一的哈希值。优点：拓展性较高。缺点：代码实现较复杂。

（3）使用状态压缩。八数码总共九位，每一位为 0~8 的数码，可以压缩成长度为 9*4 长度的比特串，使用 long long 类型存储。通过位运算进行编码解码。优点：编码解码简单，时空复杂度优秀。缺点：拓展性较差。

本实验中代码实现采用第三种方式，从低位到高位的每 4 位表示一个数码。

下图分别为初态 st 的读入代码。

```
//read
for(int i=1;i<=9;i++)
{
    scanf("%s",s+1);
    st=st|((1ll*s[1]-'0')<<(4*(i-1)));
}
```

2．设计估价函数。

由于 A*算法要求估价函数严格小于等于实际代价。在本实验中，单位代价表示为两个相邻数码（其中一个为 0）的交换。为保证上述要求，选择当前状态与目标状态相同位置**不同数码的个数减 1** 为估价函数（0 会被视作普通数码）。下图为计算估价函数函数 geth()，输入状态 s，输出估价函数值。

```
int geth(long long s)
{
    int ret=-1;
    for(int i=1;i<=9;i++)
        ret+=(s>>(4*(i-1))&15)!=(ed>>(4*(i-1))&15);//count different digits
    return max(ret,0);//special judge
}
```

注：当前状态与目标状态之差一步时，两个状态相同位置不同数码的个数为 2，此时不满足估价函数小于等于实际代价的约束。

简要的证明为：最优情况下，每一次移动最多使一个数码的位置归位，仅当最后一步会使两个数码的位置归位，则最少代价为当前状态与目标状态相同位置不同数码的个数减 1。非最优情况下，即某次移动不能使任何数码归位或使归位的数码移位，此时估价函数显然小于实际代价。综上上述估价函数符合要求。

3．特判无解情况。

根据排列的逆序对奇偶性可以将排列分为奇排列和偶排列。由于八数码的移动可以分为相邻两项的交换（左右移动）和左右相隔两项的两项的交换（上下移动），**在忽略 0 贡献逆序对的情况下**，分析可知八数码的移动操作不会改变排列的奇偶性，且某一个奇偶性的状态可达所有该奇偶性的状态，故可以通过简单判断两个状态的逆序对奇偶性从而判断两个状态时候互相可达。下图为逆序对计数函数。

```
int check(long long s)//O(n^2) count inverted pair
{
    int ret=0;
    for(int i=1;i<=9;i++)
    {
        int x=(s>>(4*(9-i)))&15;
        for(int j=i+1;j<=9;j++)
        {
            int y=(s>>(4*(9-j)))&15;
            if(x>y&&y!=0)ret++;
        }
    }
    return ret;
}
```

4．open 表和 close 表的定义。

由 A*算法流程可知，搜索过程中任意节点一定属于三种状态之一。未访问、访问未搜索（open 表）、已搜索（close 表）。使用 STL 中的映射可以存储每一个搜索状态的状态。声明代码如下。

```
map<long long,bool>vis[2];
```

当 vis[0][u]=0,vis[1][u]=0 时，表示 u 状态未访问。

当 vis[0][u]=1,vis[1][u]=0 时，表示 u 状态访问未搜索。

当 vis[0][u]=0,vis[1][u]=1 时，表示 u 状态已搜索。

且 open 表需要实现插入表项、f 值最大值查找、最大值删除功能，可以使用 STL 中的优先队列实现上述功能。声明代码如下。

```
priority_queue<pair<int,long long>,vector<pair<int,long long>>,greater<pair<int,long long>>>q;
```

该代码声明了一个以 pair<int,long long>为元素的大根堆，排序方式为 pair 默认排序规则。其中 pair 第一维存储状态的 f 值，第二维存储状态。

同时由于 close 表不需要查找最值，则不需要额外的数据结构进行维护，使用 vis 即可。

5．A*算法运行过程。

首先需要对 st 状态初始化，即计算 h、g、f 的值，插入 open 表。

```
h[st]=geth(st);
g[st]=0;
f[st]=h[st]+g[st];
vis[0][st]=1;
q.push(make_pair(f[st],st));
```

然后执行关于优先队列 q 的队列拓展过程。取队首元素、更新状态、计算后继状态、尝试将后继状态插入优先队列中（松弛操作）。下图为计算后继状态的代码，首先需要找到 0 在八数码中的位置，记录在 pos 中，然后根据 0 的位置尝试向四个方向更新，每次需要做的是交换 0 与相邻元素的位置。程序中，s 表示当前状态，ss 表示后继状态，使用基础位运算实现元素的交换。（由于实现关系，pos 存储逻辑位置 1~9，实际位运算是 4*（0~8），故 pos 在运算过程中需要减一）

```
int pos=0;
for(int i=1;i<=9;i++)if((s>>(4*(i-1))&15)==0)pos=i;
assert(pos!=0);
if(pos%3!=0)//right
{
    long long d=((s>>4*(pos-1+1))&15);
    ss=s&(~(15ll<<(4*(pos-1+1))));
    ss&=(~(15ll<<(4*(pos-1))));
    ss|=((d<<(4*(pos-1))));
    update(s,ss);
}
if(pos%3!=1)//left
{
    long long d=((s>>4*(pos-1-1))&15);
    ss=s&(~(15ll<<(4*(pos-1))));
    ss&=(~(15ll<<(4*(pos-1-1))));
    ss|=((d<<(4*(pos-1))));
    update(s,ss);
}
if(pos>=4)//down
{
    long long d=((s>>4*(pos-1-3))&15);
    ss=s&(~(15ll<<(4*(pos-1))));
    ss&=(~(15ll<<(4*(pos-1-3))));
    ss|=((d<<(4*(pos-1))));
    update(s,ss);
}
if(pos<=6)//up
{
    long long d=((s>>4*(pos-1+3))&15);
    ss=s&(~(15ll<<(4*(pos-1))));
    ss&=(~(15ll<<(4*(pos-1+3))));
    ss|=((d<<(4*(pos-1))));
    update(s,ss);
}
```

松弛操作被封装在函数 update 中。

```
void update(long long s,long long ss)
{
    if(vis[1][ss])
    {
        if(f[ss]>f[s]+1)
        {
            fa[ss]=s;
            g[ss]=g[s]+1;
            f[ss]=g[ss]+h[ss];
            vis[0][ss]=1;
            vis[1][ss]=0;
            q.push(make_pair(f[ss],ss));
            //close->open
        }
    }
    else if(vis[0][ss])
    {
        if(f[ss]>f[s]+1)
        {
            fa[ss]=s;
            g[ss]=g[s]+1;
            f[ss]=g[ss]+h[ss];
            q.push(make_pair(f[ss],ss));
            //open->open(update priority_queue)
        }
    }
    else
    {
        vis[0][ss]=1;
        fa[ss]=s;
        h[ss]=geth(ss);
        g[ss]=g[s]+1;
        f[ss]=g[ss]+h[ss];
        q.push(make_pair(f[ss],ss));
        //unvis->open
    }
}
```

update 函数主要分为三个分支，分别对应后继状态在 close 表中、open 表中和未访问的状态。每个状态的更新都需要维护所有相关变量。值得注意的是，第二个分支 open 表到 open 表的转移过程，仍然需要向优先队列中插入新元素。原因是 ss 状态的 f 值已经被更新，但是由于堆无法实现内部元素动态更新，所以需要通过增加冗余信息的方式实现每次从堆中获取的元素都是 f 值最小的元素。上述方法是本人在 ACM 竞赛中自创的方法，优点是实现简单。缺点是时空复杂度在冗余信息过多时无法保证严格 O（log）。下图是在取优先队列最值元素过程中为保证元素非冗余信息的判断语句。

```
if(f[u.second]!=u.first)continue;//redundancy
```

6．输出解过程。

维护一个前继状态 fa，声明如下。记录每一个状态的最优前继状态。

```
map<long long,long long>fa;
```

需要在每次 update 过程中，当某个状态被松弛时，需要记录哪个状态更新的该状态（即 fa）。这样在输出的时候只需要从末态 ed 倒序向前根据 fa 指针跳转直到起始状态。

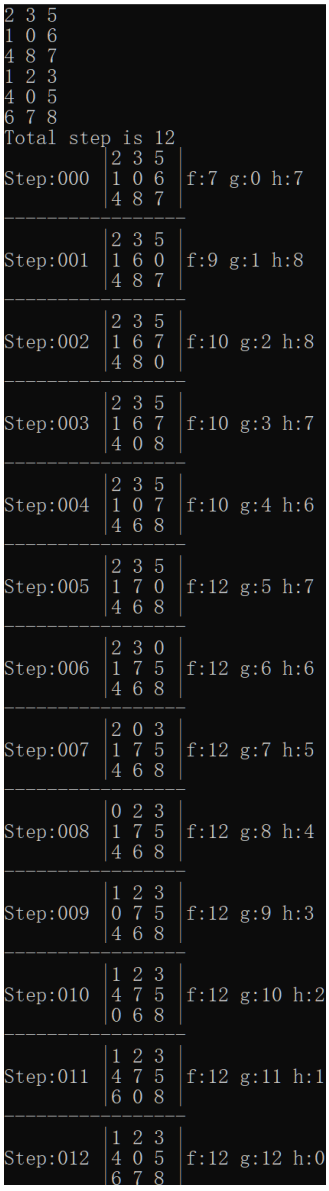
由于顺序是反向的，所以需要逆序输出 v 中的变量。（也可以通过在输入节点交换 st 和 ed 从而不需要倒序输出）

```
vector<long long>v;
while(s!=st)
{
    v.push_back(s);
    s=fa[s];
}
v.push_back(st);
```

九、实验数据及结果分析：

```
1 2 3
4 0 5
6 7 8
1 2 3
4 0 5
6 8 7
Unreachable
```

当输入状态为不可达时，程序正确判断无解。



当输入状态可达时，程序正确输出了最短解法。

十、实验结论：

A*算法正确且高效地实现了八数码的求解过程。同时计算过程符合预期，即 h 函数始终小于等于实际代价。

十一、总结及心得体会：

学习了 A*算法的计算过程，强化了编写、调试代码的能力，也进一步加深了人工智能启发式搜索过程的理解，感受到了启发式搜索的高效率。如使用普通搜索时，30 层深度的问题需访问 100000 个左右的状态，而 A*只需要访问 300 个左右的状态。

十二、对本实验过程及方法、手段的改进建议：

可以使用 splay 等常见平衡树代替堆，以获得更优的时空复杂度。在队列拓展时涉及到的操作为添加元素、动态修改队内元素、获得最优元素。其中获得最

优元素可以通过在 splay 树中维护最值与对应下标的信息，使用信息在 O（log）时间内找到最值并删除，修改元素只需使用信息找到对应位置删除后重新添加即可，时间复杂度也是 O（log）。

报告评分：

指导教师签字：

电 子 科 技 大 学

实 验 报 告

学生姓名：张镕麒　**学 号：**2019081301026　**指导教师：**康昭

实验地点：主楼 A2-413-2　**实验时间：**2021/11/28

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：5 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：

$$gain(A)=info(D)-info_A(D)$$

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. 定义结构体 node 存储输入数据，声明如下。type 存在三种取值。

```
struct node
{
    double a[maxele];
    int type;
}p[maxn],o;
```

声明结构体 vec 存储决策树顶点，声明如下。v 存储该节点下符合当前决策条件的所有个体标号。ch 存储该节点的左右儿子标号。当 isleaf 为 0 时表示该点非叶节点，k 对应属性标号，m 对应 k 的划分点。当 isleaf 为 1 是表示该点是叶节点，k 对应个体的 type 值。

```
struct vec
{
    vector<int>v;
    int k,ch[2];
    double m;//ch[0]<=m ch[1]>m
    bool isleaf;
}a[maxn];
```

2. 计算熵函数 getent。支持计算条件熵，下图为 getent 函数。

```
double getent(vector<int>&v,int k,double val,int dir)
{
    cnt=0;
    int p1=0,p2=0,p3=0,sz=v.size();
    assert(sz>0);
    for(int i=0;i<v.size();i++)
    {
        if(dir==0&&p[v[i]].a[k]>val){sz--;continue;}
        if(dir==1&&p[v[i]].a[k]<=val){sz--;continue;}
        cnt++;
        if(p[v[i]].type==1)p1++;
        if(p[v[i]].type==2)p2++;
        if(p[v[i]].type==3)p3++;
    }
    double ret=0;
    if(p1)ret-=1.0*p1/sz*log2(1.0*p1/sz);
    if(p2)ret-=1.0*p2/sz*log2(1.0*p2/sz);
    if(p3)ret-=1.0*p3/sz*log2(1.0*p3/sz);
//  printf("getent:%d %lf %d %lf(%d %d %d %d)\n",k,val,dir,ret,p1,p2,p3,sz);
    return ret;
}
```

v 是传入的 vector 指针，可以加速运算。v 存储了要被计算的个体标号。k 是要筛选的属性标号。val 是属性的划分点。dir 表示不等号为小于等于还是大于。当 dir 为 0 时，属性 k<=val 的个体会被筛选到，否则属性 k>val 的个体会被筛选到。被筛选到的个体会参与熵的计算过程。无任何条件的信息熵计算的调用格式为（任何元素小于等于无穷大）：

```
ent[maxele]=getent(a[u].v,1,inf,0);
```

3. 计算主体 dfs 函数。首先需要判断该点是否 type 值唯一，若唯一则停止递归。并更新该点信息、返回。否则参考 PPT 中的划分方式，需要将该节点一分为二，选择最优属性的最优划分点进行划分，下图为计算选取过程。

```
    for(int t=0;t<maxele;t++)
    {
//      puts("!");
//      if(a[u].st[t])continue;
        double mn=inf;
        vector<double>s;
        for(int i=0;i<a[u].v.size();i++)s.push_back(p[a[u].v[i]].a[t]);
        sort(s.begin(),s.end());
        s.erase(unique(s.begin(),s.end()),s.end());
        for(int i=0;i+1<s.size();i++)
        {
            double x=(s[i]+s[i+1])/2.0;
            double tmp=getent(a[u].v,t,x,0);
            ent[t]=1.0*cnt/a[u].v.size()*tmp;
            tmp=getent(a[u].v,t,x,1);
            ent[t]+=1.0*cnt/a[u].v.size()*tmp;
            if(mn>ent[t])
            {
                mn=ent[t];
                mxt[t]=x;
            }
        }
        ent[t]=ent[maxele]-mn;
//      printf("%d %lf\n",t,ent[t]);
        if(a[u].k==-1||ent[t]>ent[a[u].k])a[u].k=t;
    }
```

其中 ent 存储每种属性对应的最优信息熵，mxt 存储每种属性对应的最优划分点。值得注意的是划分点集合的计算，用到了离散化去重的操作。使用 STL 自带的 sort、erase、unique 函数分别可以实现排序、删除重复元素、将每一种重复元素除首项外保持顺序移到队尾。unique 函数需要保证传入的数组有序。则通过以上函数，将所有的属性值全部放入一个 vector 中即可得到所有不重不漏且有

序的属性值。计算后保留最优属性、划分个体到两个子树中，递归处理即可。这样决策树就构建完成了。

4. 验证决策树。由于决策树已构建好，则对于 testdata 中的每个数据，在决策树中 dfs 跑到叶节点，检查对应的 type 类型是否与自身相同，代码如下。

```
bool dfs1(int u)
{
    if(a[u].isleaf)
        return a[u].k==o.type;
    return dfs1(a[u].ch[o.a[a[u].k]>a[u].m]);
}
```

再根据匹配成功的数目除以总数目即可得到匹配成功率。

5. 输出决策树。只需要类似的 dfs 过程即可将决策树输出。当遇到空节点时需要及时返回。

```
void print(int u,int cur)
{
    if(!u)return;
    if(a[u].isleaf)
    {
        for(int i=1;i<=cur;i++)printf("\t");
        printf("'%d'\n",a[u].k);
        return;
    }
    for(int i=1;i<=cur;i++)printf("\t");
    printf("property%d:\n",a[u].k+1);
    for(int i=1;i<=cur;i++)printf("\t");
    puts("{");
    for(int i=1;i<=cur;i++)printf("\t");
    printf("'<=%.2f':\n",a[u].m);
    print(a[u].ch[0],cur+1);
    for(int i=1;i<=cur;i++)printf("\t");
    printf("'>%.2lf':\n",a[u].m);
    print(a[u].ch[1],cur+1);
    for(int i=1;i<=cur;i++)printf("\t");
    puts("}");
}
```

九、实验数据及结果分析：

使用 traindata.txt 中的数据，程序运算结果如下（为方便观看，程序自动重定向输出到 output.txt 中）。

```
property3:
{
'<=2.45':
        '1'
'>2.45':
        property3:
        {
        '<=4.75':
                '2'
        '>4.75':
                property4:
                {
                '<=1.75':
                        property3:
                        {
                        '<=5.05':
                                '2'
                        '>5.05':
                                property1:
                                {
                                '<=6.05':
                                        '2'
                                '>6.05':
                                        '3'
                                }
                        }
                '>1.75':
                        '3'
                }
        }
}

successful ratio:96.00%
```

程序正确构造了训练数据对应下的最优决策二叉树。

十、实验结论：

成功使用 ID3 算法构建了决策树。并通过测试集的验证发现该决策树达到了较好的预测效果。

十一、总结及心得体会：

学习了 ID3 算法构建决策树的方式，强化了代码的实现和调适能力，学习了人工智能进行预测的理论基础，进一步加深了对人工智能算法的了解。

十二、对本实验过程及方法、手段的改进建议：

可以使用图形化界面的方式输出决策树以及构建、验证过程，从而加深算法运行过程的理解。

报告评分：

指导教师签字：

电 子 科 技 大 学

实 验 报 告

学生姓名：张镕麒 **学 号：**2019081301026 **指导教师：**康昭

实验地点：主楼 A2-413-2 **实验时间：**2021/11/28

一、实验室名称： **计算机学院实验中心**

二、实验项目名称：强化学习实验

三、实验学时：6 学时

四、实验原理：

（1）Q-Learning 算法

Q-Learning 是一种记录行为值（Q value）的方法，每种行为在一定的状态都会有一个值 Q(s,a)，就是说行为 a 在 s 状态的值是 Q（s,a）。对于迷宫游戏，s 就是当前 agent 所在的地点了。每一步，智能体可以选择四种动作，所以动作 a 有四种可能性。

在本实验里，已经提供了强化学习基本的训练接口，只需要实现 Q 表格的强化学习方法即可。算法框架如下：

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until s is terminal

图 1 强化学习算法框架

可以看到，算法的核心部分是更新 Q 表格。训练目标是在评价阶段，智能体的平均奖励达到最高。

（2）迷宫游戏说明

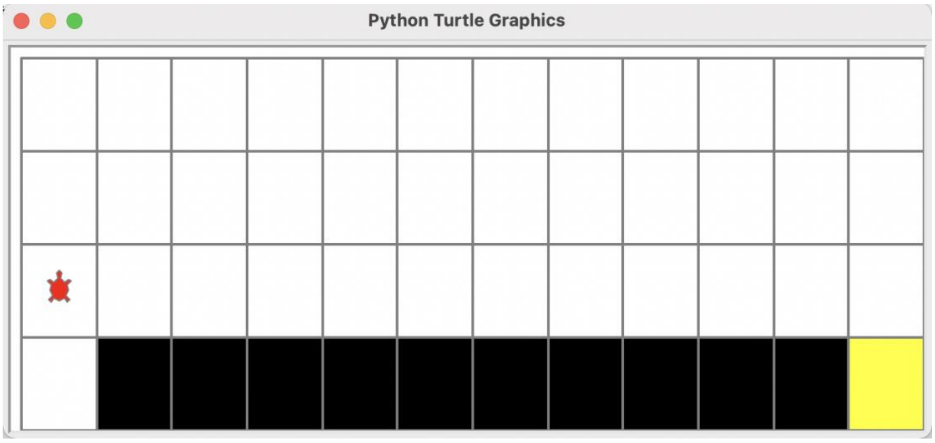


图 2 迷宫游戏示例

如上图所示，本实验所研究的迷宫由一个二维表格构成。其中白色区域是可行走区域，黑色区域是陷阱区域，黄色区域是终点。规则如下：

1） 智能体从左下角出发，到达黄色区域即游戏成功。

2） 智能体每次可选择上、下、左、右四种移动动作，每次动作得到-1 奖励。

3） 智能体不能移动出网络，如果下一步的动作命令会让智能体移动出边界，那么这一步将不会执行，即智能体原地不动，得到-1 奖励。

4） 智能体移动到黑色区域，得到-100 奖励。

5） 智能体移动到黄色区域，该回合结束。

由图可知，最优的路线需要 13 步，所以最后智能体获得的奖励指在-13 左右为最佳结果。

五、实验目的：

使用 Q 表格方法解决迷宫寻路问题，理解强化学习算法原理。

六、实验内容：

结合给定的程序代码框架，补全代码，并达到实验目的。鼓励探索新的迷宫布局和任务。

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。

七、实验器材（设备、元器件）：

PC 微机一台

八、实验步骤：

1. 选择行为函数。首先需要给出一定机会使得 agent 可以自由选择所有情况，尝试其他的可能性。剩下的大概率人工智能会根据 Q 表格进行行为决定。首先记录当前状态下 Q 表格的最大值。并通过 where 语句找到最大值对应的下表位置。若存在多个下标将返回任意一个下标。

```
def choose_action(self, state):
    if random.random() < 0.01:
        return np.random.randint(4)
    else:
        action_max = np.max(self.Q_table[state, :])
        choice_action = np.where(action_max == self.Q_table[state, :])[0]
        return np.random.choice(choice_action)
```

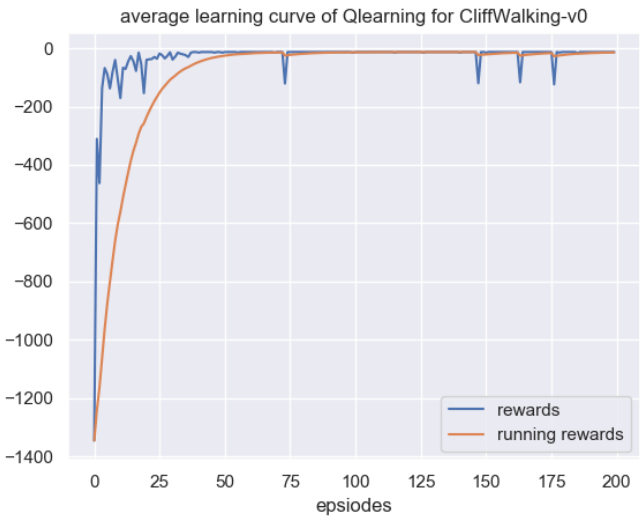
2. Q 表格更新函数。首先记录下当前的 Q 表格值，当后继状态达到终点时将该状态的 Q 值设置成正无穷。否则依据公式计算对应的 Q 值，然后按照学习速度更新当前状态的 Q 值。

```
def update(self, state, action, reward, next_state, done):
    Q_ = self.Q_table[state][action]
    if done:
        target_Q = 10000
    else:
        target_Q = reward + self.gamma * np.max(self.Q_table[next_state, :])
    self.Q_table[state][action] += self.lr * (target_Q - Q_)
```

九、实验数据及结果分析：

```
Episode:1/30, reward:-13.0
Episode:2/30, reward:-13.0
Episode:3/30, reward:-13.0
Episode:4/30, reward:-13.0
Episode:5/30, reward:-13.0
Episode:6/30, reward:-13.0
Episode:7/30, reward:-13.0
Episode:8/30, reward:-13.0
Episode:9/30, reward:-13.0
Episode:10/30, reward:-13.0
Episode:11/30, reward:-13.0
Episode:12/30, reward:-13.0
Episode:13/30, reward:-13.0
Episode:14/30, reward:-13.0
Episode:15/30, reward:-13.0
```

测试过程成功稳定到达了-13。



可以看出，agent 在初次几次迭代中迅速找到了最优解，并且后续过程稳定在最优解。

十、实验结论：

成功实现了强化学习的 Q 表格法。

十一、总结及心得体会：

学习了人工智能强化学习的 Q 表格法。锻炼了 python 的代码编写能力、可视化输出及调试能力。加深了人工智能强化学习的过程的理解。

十二、对本实验过程及方法、手段的改进建议：

可以考虑将 Q 表格方法应用到更复杂问题的求解过程中，感受其算法的运行过程。同时也可以考虑使用其他强化学习方法求解类似问题，如 Sarsa 方法，在对比中深入了解其中的算法内涵。

报告评分：

指导教师签字：