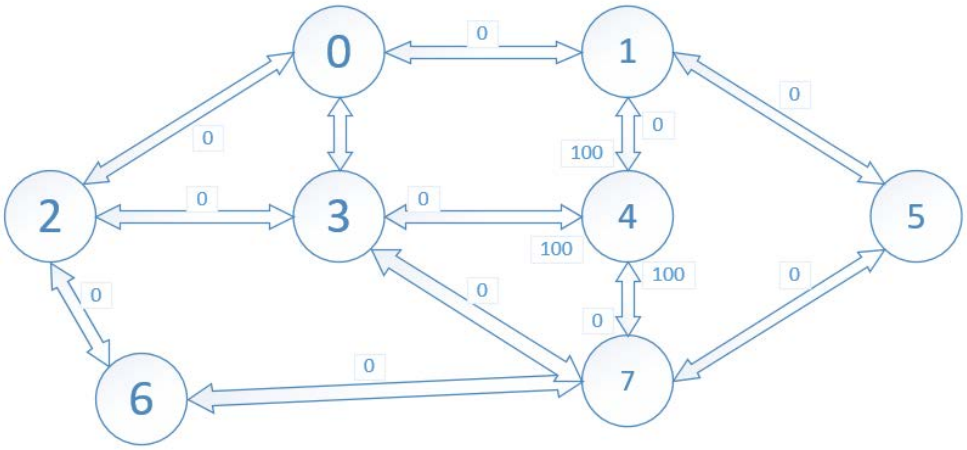
实验二　Q learning 解决迷宫问题

学号：2017526019　　班级：2015211307　　姓名：刘禾子

一、 问题描述

假设楼层内共有 8 个房间，房间之间通过一道门相连，房间编号为房间 0 到房间 7，一个智能体通过非监督学习的方法对未知的环境进行学习，用图来表示上述的房间拓扑，将每一个房间看作是一个节点，每一道们看作一条边，在任意一间房间中放置一个智能体，并期望该智能体能够从该房间开始到达一个目标房间（房间 4），为了设置这间房间 4 作为目标：

- 为每一道门（节点之间的边）赋予一个奖励值；
- 能够直接通到目标房间的门赋予一及时奖励值 100；
- 而其他的未与目标房间直接相连的门赋予奖励值 0；
- 因为每一道门都有两个方向，因此，每一道门在图中将描述为两个箭头。如图：



二、 算法思想

- 在 Q-学习中，学习的目标是达到具有最高奖励值的状态，因此，如果智能体到底了目标位置，它将永远的留在那儿。这种类型的目标被称为“吸收目标 ”absorbing state;
- Q-学习中的术语包括状态（state）和动作（action）；
- 将每一个房间称为一个“状态”，智能体从一个房间到另一个房间的移动过程称为“动作”。状态被描述为节点，动作被描述成箭头;
- 引入矩阵来表示图:将状态图和及时奖励值填入到下面的奖励表中，即矩阵 R:
  - 行代表智能体当前的状态，列代表到达下一个状态的可能的动作，矩阵的 i 行 j 列元素值表明在状态 i 下采取动作 j 的即时直接回报;
  - 表中的-1 代表空值，表示节点之间无边相连,如下图:

```
private static int[][] R=
                {{-1,0,0,0,-1,-1,-1,-1},
                 {0,-1,-1,-1,100,0,-1,-1},
                 {0,-1,-1,0,-1,-1,0,-1},
                 {0,-1,0,-1,100,-1,-1,0},
                 {-1,0,-1,0,100,-1,-1,0},
                 {-1,0,-1,-1,-1,-1,-1,0},
                 {-1,-1,0,-1,-1,-1,-1,0},
                 {-1,-1,-1,0,100,0,0,-1}};
```

- 再定义一个类似的矩阵 Q,代表智能体从经验中所学到的知识:
  - 行列的意义同 R;
  - 矩阵的 i 行 j 列元素值表明在状态 i 下采取动作 j 的总体评价。
- R 是已知的,目的是为了学习 Q;
- 智能体在最初对环境一无所知,所以矩阵 Q 被初始化为 0;
- 假设状态的数量是已知的(设为 8),若状态数未知,矩阵 Q 在最初被设为只有一个元素,若新的状态一旦被发现,为 Q 增加行和列;
- Q-学习的规则:

  Q(state, action)=R(state, action) + Gamma * Max(Q[next state, all actions]);智能体将从经验中学习,不需要教师指导信号(这被称为非监督学习).智能体将从一个状态到另一个状态进行探索，直到它到达目标状态。我们将每一次探索作为一次 episode，每一次 episode 包括智能体从初始状态到达目标状态的过程。每次智能体到达了目标状态，程序将会转向下一次探索。

三、 设计过程

1、设置参数 Gamma，以及矩阵 R 中的环境奖励值;
2、初始化 Q 矩阵为 0;
3、对每一次 episode:
随机选择一个状态;
Do while 目标状态未到达
对当前状态的所有可能的动作中，选择一个可能的动作;（事实上应该根据 Q 选择最好的一个动作）
使用这个可能的动作，到达下一个状态;
对下一个状态，基于其所有可能的动作，获得最大的 Q 值;
计算: Q(state, action)=R(state, action) + Gamma * Max(Q[next state, all actions])
设置下一个状态作为当前状态;
End For

智能体利用上述的算法从经验中学习，每一次 episode 等价于一次训练。在每一次训练中，智能体对环境进行探索（用矩阵 R 表示），并且其一旦到达目标状态，就得到奖励值。训练的目的是增强智能体的大脑，用矩阵 Q 表示。越多的训练结果将导致更优的矩阵 Q。在这种情况下，如果矩阵 Q 已经被增强，那么智能体就不会四处盲目的探索，而是会找到最快的路线到达目标状态。

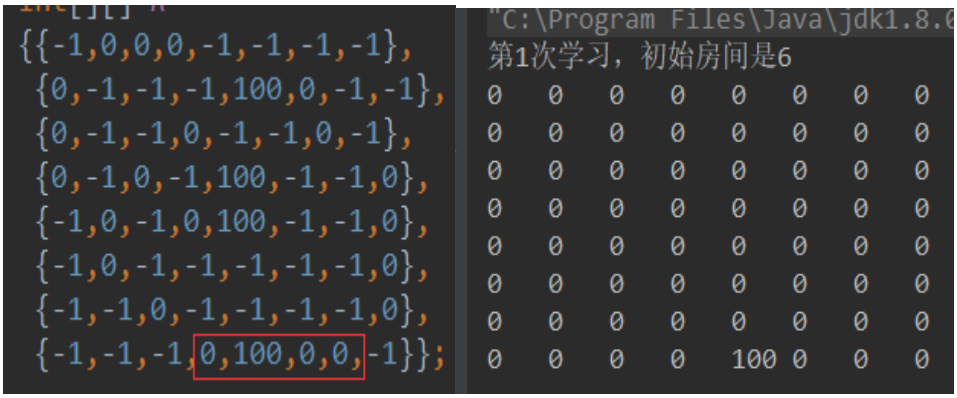
Q 一旦获得，则可以利用矩阵 Q 指导将来的动作，算法如下：
1、设置当前状态=初始状态；
2、从当前状态开始，寻找具有最高 Q 值的动作；
3、设置当前状态=下一个状态；
4、重复步骤 2 和 3，直到当前状态=目标状态。

四、 实验结果

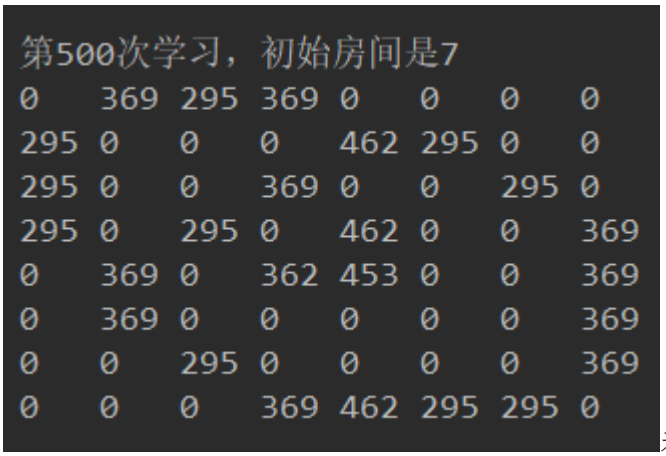
学习率 Gamma 等于 0.8,初始状态是房间 6,初始的矩阵 Q 作为一个零矩阵,R 矩阵的第 7 行(状态 6),对状态 6 来说,存在两个可能的动作:到达状态 2 或者到达状态 7,假设随机选择到达状态 7,智能体到达了状态 7 之后,观察 R 矩阵第 8 行,有 4 个可能的动作,到达状态 3,4,5 或者 6,

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

Q(7, 4) = R(7, 4) + 0.8 * Max[Q(4, 3), Q(4, 4), Q(4, 5),Q(4,6)] = 100 + 0.8 * 0 = 100



像这样一直进行下去直到第 500 次学习,Q 矩阵逐渐收敛如下图所示:



若初始状态为 0,智能体在矩阵 Q 的指导下进行移动:
在状态 0 时,在矩阵 Q 中最大的值可知下一个动作是房间 1 或 3,假设选择动作是到达 1;
在状态 1 时,矩阵 Q 建议到达状态 4;
因此,智能体的移动序列是 0-1-4.

五、 源代码

反馈矩阵类(R):

```
class FeedbackMatrix {
```

```
        int get(int x, int y){
            return R[x][y];
        }
        int[] getRow(int x){
            return R[x];
        }
        private static int[][] R=
                        {{-1,0,0,0,-1,-1,-1,-1},
                         {0,-1,-1,-1,100,0,-1,-1},
                         {0,-1,-1,0,-1,-1,0,-1},
                         {0,-1,0,-1,100,-1,-1,0},
                         {-1,0,-1,0,100,-1,-1,0},
                         {-1,0,-1,-1,-1,-1,-1,0},
                         {-1,-1,0,-1,-1,-1,-1,0},
                         {-1,-1,-1,0,100,0,0,-1}};
}
```

经验矩阵类(Q):

```
    class ExperienceMatrix {
        int get(int x, int y){
            return Q[x][y];
        }
        int[] getRow(int x){
            return Q[x];
        }
        void set(int x, int y, int value){
            Q[x][y]=value;
        }
        void print(){
            for (int i=0;i<8;i++){
                for (int j=0;j<8;j++){
                    String s=Q[i][j]+" ";
                    if (Q[i][j]<10)
                        s=s+"  ";
                    else if (Q[i][j]<100)
                        s=s+" ";
                    System.out.print(s);
                }
                System.out.println();
            }
        }
        private static int[][] Q=new int[8][8];
}
```

主函数(q_learning):

```
    import java.util.ArrayList;
    import java.util.Collections;
    import java.util.List;
    import java.util.Random;

    public class q_learning {
        private FeedbackMatrix R=new FeedbackMatrix();
```

```
private ExperienceMatrix Q=new ExperienceMatrix();
public static void main(String args[]) {
    q_learning ql=new q_learning();
    for (int i=0;i<500;i++){
        Random random=new Random();
        int x=random.nextInt(100)%8;//随机产生初始房间号
        System.out.println("第"+(i+1)+"次学习, 初始房间是"+x);
        ql.learn(x);
        System.out.println();
    }
}

private void learn(int x) {
    do {
        //随机选择一个联通的房间进入
        int y=chooseRandomRY(x);
        //获取以进入的房间为起点的历史最佳得分
        int qy=getMaxQY(y);
        //计算此次移动的得分
        int value=calculateNewQ(x,y,qy);
        Q.set(x,y,value);
        x=y;//y 房间作为下一次探索的起点
    }
    //走出房间则学习结束
    while (4!=x);
    Q.print();
}

private int chooseRandomRY(int x) {
    int[] qRow=R.getRow(x);
    List<Integer> yValues= new ArrayList<>();
    for (int i=0;i<qRow.length;i++){
        if (qRow[i]>=0){
            yValues.add(i);
        }
    }
    Random random=new Random();
    int i=random.nextInt(yValues.size())%yValues.size();
    return yValues.get(i);
}

private int getMaxQY(int x) {
    int[] qRow=Q.getRow(x);
    int length=qRow.length;
    List<YAndValue> yValues= new ArrayList<>();
    for (int i=0;i<length;i++){
        YAndValue yv=new YAndValue(i,qRow[i]);
        yValues.add(yv);
    }
    Collections.sort(yValues);
    int num=1;
```

```
            int value=yValues.get(0).getValue();
            //取得优先队列中价值最大的一个点
            for (int i=1;i<length;i++){
                if (yValues.get(i).getValue()==value)
                    num=i+1;
                //若有多个价值最大的点，则随机选择一个
                else
                    break;
            }
            Random random=new Random();
            int i=random.nextInt(num)%num;
            return yValues.get(i).getY();
        }
        //Q(x,y)=R(x,y)+0.8*max(Q(y,i))
        private int calculateNewQ(int x, int y, int qy) {
            return (int) (R.get(x,y)+0.8*Q.get(y,qy));

        }
        //优先队列，对每个房间到达各个房间的价值作降序排序
        public static class YAndValue implements Comparable<YAndValue>{
            int y,value;
            int getY() {
                return y;
            }
            /*public void setY(int y) {
                this.y = y;
            }*/
            int getValue() {
                return value;
            }
            YAndValue(int y, int value){
                this.y=y;
                this.value=value;
            }
            @Override
            public int compareTo(YAndValue o) {
                return o.getValue()-this.value;
            }
        }
}
```