****

**人工智能上机实验报告**

题目： 计算最佳策略

**学生姓名 姜铁**

**学生学号 3015218088**

**学院名称 软件学院**

**专 业 软件工程**

**时 间 2017.10.15**

目 录

第一章、问题描述…………………………………………………×

第二章、程序设计…………………………………………………×

第三章、调试分析…………………………………………………×

第四章、测试分析…………………………………………………×

第五章、未来展望与思考…………………………………………×

### 实验报告编写提纲

1. 问题描述

实验问题：在存在障碍物(用’#’表示)的情况下，从起点（左上角）到格子内其他点的最短距离。并且给出到指定点（用’E’表示）的一条最短路径。

建立模型：

在本问题中，状态集合**S**等价于**4\*4=16每个格子**，行为集合**a**等价于**上下左右四个方向**，**对未来价值的折扣因子**=**1**，**当前奖励**被定义为**-1**，即每走一步奖励-1，这样取较大者即表示了最小步数，并且我们假定每个状态根据行为集合a中定义的行为转移到的合法状态的概率都是相同的。**value function**的含义就是**起点到该状态的最短距离**。这样，本问题的**贝尔曼方程**就等价于：对某个状态S，**V[S]=max{-1+V[下个状态]}**，然后我们统计下本次所有状态的V值和上次的V值之间的差异，并认为差值小于某一限额时，策略应该已经保持不变，就可以直接跳出循环，并根据当前的所有状态的value function值找出最优的状态转移。

Value Iteration:

为了求出策略（最短路径），采用值迭代的方法，问题转换为求出最优的value function。

不断地使用贝尔曼方程更新value function的值，直到value function收敛。然后用最优的value function的值来反推策略。

Policy Iteration:

在有一个初始策略的前提下，不断地根据当前的value function改进策略，同时也反过来不断地改进value function的值。直到当前策略无法再做改进，即得到了一个最优的策略和最优策略下的value function.

1. 程序设计

**policyIteration:**

#include <bits/stdc++.h>

#include <windows.h>

using namespace std;

const int MAX\_N = 4;

const float INF = -1000000000.0f;

float V[MAX\_N][MAX\_N]; //值矩阵

float OLDV[MAX\_N][MAX\_N]; //保存旧的val值

char S[MAX\_N][MAX\_N]; //状态矩阵，直观存储运动方向

char OLDS[MAX\_N][MAX\_N]; //旧的状态矩阵

char MAP[MAX\_N][MAX\_N]; //the map for the grid world

char A[4] = { 'n','e','s','w' }; //Action ,'↑','→','↓','←'

int dx[] = { -1,0,1,0 }, dy[]= { 0,1,0,-1 }; //四向移动

int r = -1; //奖励值

float disc = 1.0f; //折扣值

float en = 0.00000001f; //sub小于这个数时收敛

// 获取输出流的句柄

HANDLE hOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

//初始化

void AddEndPoint(int x,int y){MAP[x][y]=OLDS[x][y]=S[x][y]='E';}//将(x,y)位置设置为终点

void AddObstcle(int x,int y){MAP[x][y]=OLDS[x][y]=S[x][y]='#';}//将(x,y)位置设置为障碍

void init() {

for (int i = 0; i < MAX\_N; i++)

for (int j = 0; j < MAX\_N; j++) {

S[i][j] = 'n'; //初始化为向右走

OLDS[i][j] = 'n';

}

AddEndPoint(3,3);

AddObstcle(1,1);

AddObstcle(1,2);AddObstcle(1,3);

AddObstcle(3,2);

for(int i=0;i<MAX\_N;i++)for(int j=0;j<MAX\_N;j++)if(i+j>0){

V[i][j]=OLDV[i][j]=INF;

}

}

int getIndex(char ch) {

switch (ch) {

case 'n':

return 0;

case 'e':

return 1;

case 's':

return 2;

case 'w':

return 3;

default:

return -1;

}

}

void printVal() {//输出图上点到左上角的最短距离

puts("--------------distance----------------");

for (int i = 0; i < MAX\_N; i++) {

for (int j = 0; j < MAX\_N; j++) {

if(V[i][j]>=-16)printf(" %.0f ", -1\*V[i][j]+1e-10);

else printf("oo ");//不可达则输出无穷

}

puts("\n");

}

}

int IV[MAX\_N][MAX\_N];

void printPath(int x,int y) {//输出左上角到x行y列一条最短路径

{

char outs[MAX\_N][MAX\_N];

memcpy(outs,S,sizeof(S));//拷贝S数组而不是直接对S进行更改

for(int i=0;i<MAX\_N;i++)for(int j=0;j<MAX\_N;j++)

if(S[i][j]!='#'&&!(x==i&&y==j))outs[i][j]='.';

while(!(x==0&&y==0)){

int nx,ny;

int k=4;

while (k--) {

nx = x + dx[k], ny = y + dy[k];

if (nx >= 0 && nx < MAX\_N && ny >= 0 && ny < MAX\_N && IV[nx][ny]==IV[x][y]+1)

{outs[nx][ny]=A[k];x=nx,y=ny;break;}

}

}

for (int i = 0; i < MAX\_N; i++) {

for (int j = 0; j < MAX\_N; j++){

char ch;

switch (outs[i][j]) {

case 's':

printf(" ↑ ");

break;

case 'w':

printf("→ ");

break;

case 'n':

printf(" ↓ ");

break;

case 'e':

printf("← ");

break;

default:

printf(" %c ", outs[i][j]);

break;

}

}

puts("\n");

}puts("\n");

}

}

void PolicyEvaluation(){

float sub;

int cnt = 0;

do{

sub = 0.0f;

for (int i = 0; i < MAX\_N; i++) {

for (int j = 0; j < MAX\_N; j++)if(i+j>0){

if (S[i][j] != '#') { //不是障碍物及终点

float val = V[i][j];

int k = getIndex(S[i][j]); //根据状态得出移动下标

int x = i + dx[k], y = j + dy[k];

if (x >= 0 && x < MAX\_N && y >= 0 && y < MAX\_N && S[x][y] != '#') {

// V[i][j] =max(V[i][j], r + disc \* OLDV[x][y]);

V[i][j] =r + disc \* OLDV[x][y];

}

sub = max(sub, fabs(val - V[i][j]));

}

}

}

//把新的val值拷贝到旧的数组中

memcpy(OLDV,V,sizeof(V));

printf("cnt : %d\n", ++cnt);

printVal();

} while (sub > en);

}

void PolicyImprovement() {

while (true) {

bool stable = true;

for (int i = 0; i < MAX\_N; i++) {

for (int j = 0; j < MAX\_N; j++) {

if (OLDS[i][j] != '#' && i+j>0) {

char oldact = OLDS[i][j];

int k = 4;

float ma = INF;

while (k--) {

int x = i + dx[k], y = j + dy[k];

if (x >= 0 && x < MAX\_N && y >= 0 && y < MAX\_N && OLDS[x][y] != '#') {

float round = r + disc \* OLDV[x][y];

if (round - ma > en) {

ma = round; //改变Val值

S[i][j] = A[k]; //改变当前状态的最佳策略

}

}

}

if (oldact != S[i][j])

stable = false;//策略发生了变化，说明当前策略还有改进的余地

}

}

}

//将所有新状态拷贝至旧状态

for (int i = 0; i < MAX\_N; i++) {

for (int j = 0; j < MAX\_N; j++) {

OLDS[i][j] = S[i][j];

}

}

if (stable) {

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED);

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

printVal();

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 0xA); //亮绿

for(int i=0;i<MAX\_N;i++)for(int j=0;j<MAX\_N;j++)IV[i][j]=round(V[i][j]);

for(int i=0;i<MAX\_N;i++)for(int j=0;j<MAX\_N;j++)if(MAP[i][j]=='E'&&IV[i][j]>=-16)

printPath(i,j);

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 0x0f); //变回原状

break;

}

else {

PolicyEvaluation(); //继续策略迭代

}

}

}

int main() {

init();

PolicyEvaluation();

PolicyImprovement();

return 0;

}

**valueIteration:**

#include <bits/stdc++.h>

#include <windows.h>

using namespace std;

const int MAX\_N = 4;

const float INF = -9999999.0f;

float V[MAX\_N][MAX\_N]; //值矩阵

float OLDV[MAX\_N][MAX\_N]; //保存旧的val值

char S[MAX\_N][MAX\_N]; //状态矩阵，直观存储运动方向

char A[4] = { 'n','e','s','w' }; //Action ,'↑','→','↓','←'

int dx[] = { -1,0,1,0 }, dy[] = { 0,1,0,-1 }; //四向移动

int r = -1; //奖励值

float disc = 1.0f; //折扣值

float en = 0.0000001f; //sub小于这个数时收敛

void AddEndPoint(int x,int y){S[x][y]='E';}//将(x,y)位置设置为终点

void AddObstcle(int x,int y){S[x][y]='#';}//将(x,y)位置设置为障碍

//初始化

void init() {///S is the map,# is the 障碍,E is the end point

AddObstcle(0,1);AddObstcle(0,2);AddObstcle(0,3);

AddObstcle(1,1);AddObstcle(1,3);

AddEndPoint(1,2);

// S[0][1]=S[0][2]=S[0][3]='#';

// S[2][1]=S[2][2]=S[2][3]='#';

// S[1][1]=S[1][3]='#';

// S[1][2]='E';

// S[3][1] = 'E'; //E目标点

// S[2][2] = '#';

// S[1][2] = 'E';

for(int i=0;i<MAX\_N;i++)for(int j=0;j<MAX\_N;j++){

if(i==0&&j==0)OLDV[i][j]=0;//初始化起点的value function值为0

else OLDV[i][j]=INF;

}

}

void printVal() {//输出中间的value值

puts("----------------value-----------------");

for (int i = 0; i < MAX\_N; i++) {

for (int j = 0; j < MAX\_N; j++) {

printf("%.4f ", V[i][j]);

}

puts("\n");

}

}

int IV[MAX\_N][MAX\_N];

void output\_path(int x,int y){//输出起点到s[x][y]的最短路径

char outs[MAX\_N][MAX\_N];

memcpy(outs,S,sizeof(S));

for(int i=0;i<MAX\_N;i++)for(int j=0;j<MAX\_N;j++)

if(S[i][j]!='#'&&!(x==i&&y==j))outs[i][j]='.';

while(!(x==0&&y==0)){

int nx,ny;

int k=4;

while (k--) {

nx = x + dx[k], ny = y + dy[k];

if (nx >= 0 && nx < MAX\_N && ny >= 0 && ny < MAX\_N && IV[nx][ny]==IV[x][y]+1)

{outs[nx][ny]=A[k];x=nx,y=ny;break;}

}

}

for (int i = 0; i < MAX\_N; i++) {

for (int j = 0; j < MAX\_N; j++){

char ch;

switch (outs[i][j]) {

case 's':

printf(" ↑ ");

break;

case 'w':

printf("→ ");

break;

case 'n':

printf(" ↓ ");

break;

case 'e':

printf("← ");

break;

default:

printf(" %c ", outs[i][j]);

break;

}

}

puts("\n");

}puts("\n");

}

void ValueIteration() {

float sub = 0;

int cnt = 0;

do {

sub = 0;

for (int i = 0; i < MAX\_N; i++) {

for (int j = 0; j < MAX\_N; j++) if(!(i==0&&j==0)){

if (S[i][j] != '#') { //不是障碍物及终点

float val = V[i][j];

int k = 4;

float ma = INF;

while (k--) {

int x = i + dx[k], y = j + dy[k];

if (x >= 0 && x < MAX\_N && y >= 0 && y < MAX\_N && S[x][y] != '#') {

float round = r + disc \* OLDV[x][y];

if (ma < round) {

ma = round; //改变Val值

}

}

}

V[i][j] = ma; //V[i][j]取得周围环境的最大值

sub = max(sub, fabs(val - V[i][j]));

}else V[i][j]=INF;

}

}

//把新的val值拷贝到旧的数组中

memcpy(OLDV,V,sizeof(V));

printf("cnt : %d\n", ++cnt);

printVal();

} while (sub >= en);

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 0xA); //亮绿

puts("---------------distance---------------");

for (int i = 0; i < MAX\_N; i++) {

for (int j = 0; j < MAX\_N; j++) {

if(V[i][j]==INF)printf("oo ");

else printf(" %.0f ",-1\* V[i][j]+1e-10);

}

puts("\n");

}

for(int i=0;i<MAX\_N;i++)for(int j=0;j<MAX\_N;j++)IV[i][j]=round(V[i][j]);

//打印策略

for(int i=0;i<MAX\_N;i++)for(int j=0;j<MAX\_N;j++)if(S[i][j]=='E'&&V[i][j]>=-16){//当前的E必须相对起点可达

output\_path(i,j);

}

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 0x0f); //变回原状

}

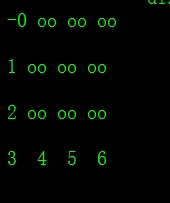
int main() {

init();

ValueIteration();

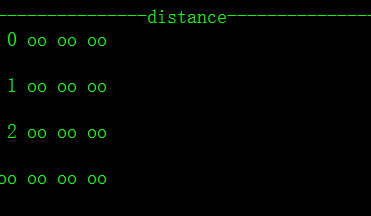
return 0;

}

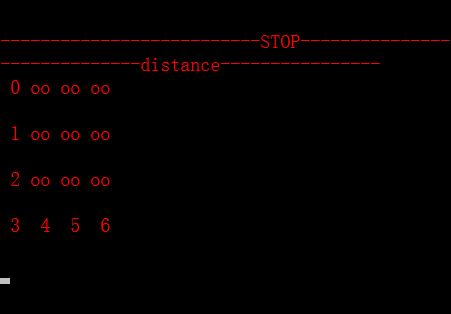
1. 调试分析（可选）

问题：输出结果时直接对V数组乘-1，出现了-0！；

解决：联想到计算机组成原理关于浮点数的内容，让每个答案\*-1之后再加上1e-10这样一个小量。

问题：程序没有跑到我们希望的收敛时就跳出了！

解决：应该令每次OLDV与V数组元素之间的差值取绝对值加到sub上,而不是直接加

问题：当初始格子布局为：

O###

O#E#

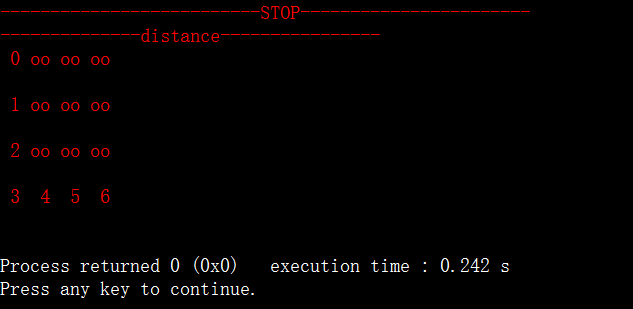
O###

OOOO

程序陷入了死循环！（‘#’代表障碍，‘E’代表终点，‘O’代表一般的格子）

解决：

原因在于从左上角根本无法到达E点，所以从E点反向搜索时根本找不到恰好value function的值等于它+1的点，这样E就无法找到起点从而跳出循环。

只要对于每个’E‘在输出路径前判断可达性即可。

问题：一直没有对给定的’E’输出路径。

解决：S数组用于保存策略，会覆盖原来的’E’，专门另开一个字符数组存图即可。

Bug总结：程序出现了诸多bug，一方面是我对模型的建立和问题的解决理解还不到位，另一方面(更多的)是理论上口头说说总是头头是道，但是一用代码实现就有诸多细节需要我们处理，需要我们在事先有足够的考虑和度量，才能保证程序能够应对所有的合法输入，都能给出正确合理的反馈。

1. 结果分析

* 参数分析、代码运行结果及分析

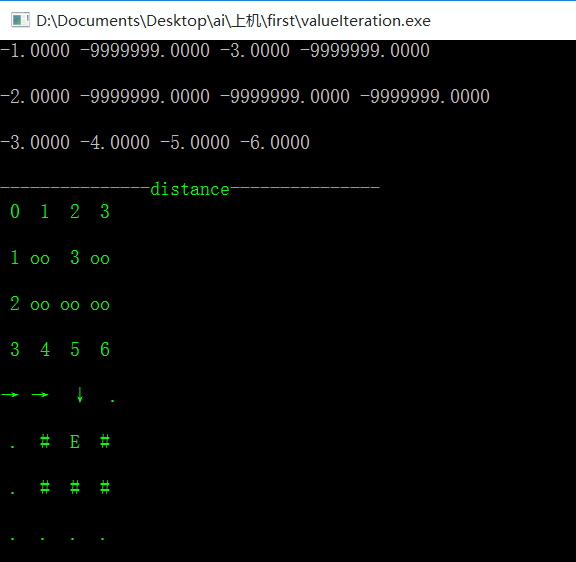
Value iteration:

图：OOOO

O#O#

O###

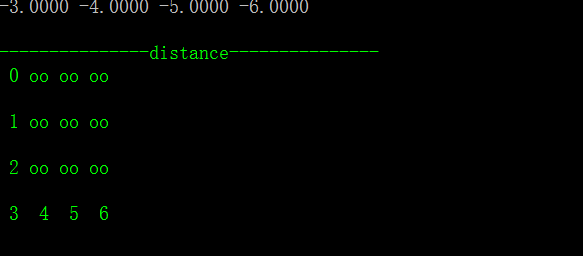
OOOO

O###

O#E#

O###

OOOO



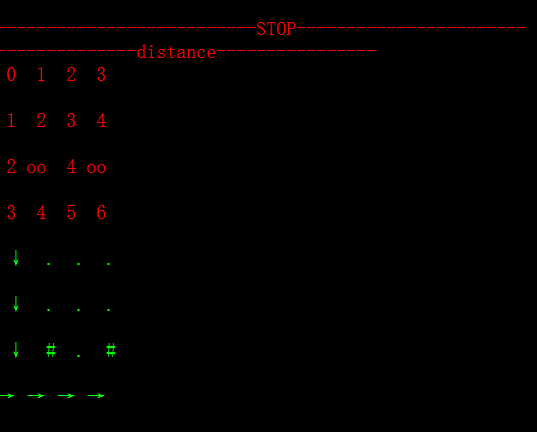
Policy iteration:

OOOO

OOOO

O#O#

OOOE

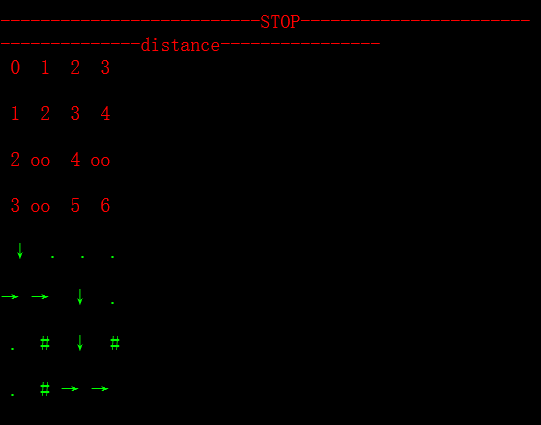


OOOO

OOOO

O#O#

O#OE



五、未来展望与思考

在课堂上我对策略迭代比较理解了，但是对于值迭代认识不是很清楚。不过通过这次的实验，我对于值迭代已经足够的理解了。同时，我发现了自己建模的能力比较差，虽然我理解了策略迭代，但是涉及到具体问题还是不知道如何下手。原因一方面主要是我最初理解策略迭代时，没有认识到value function的重要性，我以为value function自始至终是一个固定的，没想到value function也是随着最优策略的找寻不断地改造，使之接近最终的value function。正所谓”读万卷书，不如行万里路“，理论上的理解认识还是需要实践来加深和检验的。