附录

运行环境：

Python 环境：Latest Python 3 Release - Python 3.9.5 ( anaconda notebook)

C++ 版本：TDM-GCC 4.9.2 64-bit Release

操作系统: Microsoft Windows 10 家庭中文版 Version 10.0 (Build 17134)

Java 版本: Java 1.7.0\_60-b19 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode

附录目录：

1. 数据生成(保证对).cpp
2. 第一题.cpp
3. 右图转左图 10x10 实验.html
4. 图3的测试代码（试题数据）.cpp
5. 图3的测试代码（自动生成的数据）.cpp
6. 图4扫描改良改(右图测试)
7. 520校赛 右图转左图 输出100乘100.html
8. 第二题K.cpp
9. 第二题扫描改良改+K.cpp
10. Ktest.cpp

附录一 数据生成(保证对).cpp

说明：数据生成(保证对)中可改变障碍密度和测试数据量，生成的数据在test4中。

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF=0x3f3f3f3f;

const long long mod=1000000007;

const double e=2.718281828459045;

const double pi=3.1415926535;

default\_random\_engine E;

const double P=0.3; //障碍比例

int N;

int land[100][100]; //地块状态

int T[100][100]; //时间

const int NANT=100\*100+5;

int ans;

const int A=10,B=10;

uniform\_real\_distribution<double>xx(0,1);

int ram(){

if(xx(E)<P)return 1;

else return 0;

}

int dx[8]={1,1,0,-1,-1,-1,0,1};

int dy[8]={0,1,1,1,0,-1,-1,-1};

struct node{

int x,y,t;

node(int x=0,int y=0,int t=0):x(x),y(y),t(t){};

friend bool operator<(const node &a,const node &b ){

return a.t<b.t;

}

};

int main(){

N=1000; //总数

freopen("test4.txt","w",stdout);

int cnt=0;

printf("%d\n",N);

while(cnt<N){

ans=NANT;

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++){

T[i][j]=NANT;

land[i][j]=1;

}

}

for(int i=1;i<=10;i++)for(int j=1;j<=10;j++)land[i][j]=ram();

land[A][1]=3;

land[1][B]=2;

priority\_queue < node , vector<node> >q;

node str(A,1,0);

T[A][1]=0;

q.push(str);

while( !q.empty() ){

node tmp=q.top();

q.pop();

int rx,ry,rt=tmp.t+1;

for(int r=0;r<8;r++){

rx=tmp.x+dx[r];

ry=tmp.y+dy[r];

if( land[rx][ry]==2 ){

if(ans>rt)ans=rt;

}else{

//printf("[%d][%d]: %d %d \n",rx,ry,land[rx][ry],T[rx][ry]);

if( land[rx][ry]==0 && T[rx][ry]>rt ){

T[rx][ry]=rt;

node N(rx,ry,rt);

q.push(N);

}

}

}

}

T[1][B]=ans;

if(T[1][B]<NANT){

cnt++;

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++)printf("%d\t",land[i][j]);

printf("\n");

}

printf("\n");

}

}

return 0;

}

附录2 第一题.cpp

说明：图1的测试代码（自动生成的数据）

第一题.cpp 会读取 在 test4 中 的数据

输出到 test4\_answer(godsight)

图1的测试代码（试题数据）

将第一题.cpp 中的数据源改为t1p1\_in即可

输出到 test4\_answer(godsight)

图2的测试代码（试题数据）

用 右图转左图 10x10 实验.html 中的代码处理图片

导出到指定excel后 将数据放入t1p1\_in 的数据区 即可

（实际上图1也可以这么解）

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF=0x3f3f3f3f;

const long long mod=1000000007;

const double e=2.718281828459045;

const double pi=3.1415926535;

#define CK cout<<"OK\n";

int land[100][100]; //地块状态

int T[100][100]; //时间

int A,B;

const int NANT=100\*100+5;

int ans;

int dx[8]={1,1,0,-1,-1,-1,0,1};

int dy[8]={0,1,1,1,0,-1,-1,-1};

struct node{

int x,y,t;

node(int x=0,int y=0,int t=0):x(x),y(y),t(t){};

friend bool operator<(const node &a,const node &b ){

return a.t<b.t;

}

};

void init(){

ans=NANT;

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++){

T[i][j]=NANT;

land[i][j]=1;

}

}

for(int i=1;i<=10;i++)for(int j=1;j<=10;j++)scanf("%d",&land[i][j]);

}

void checkmap(){

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++)printf("%d ",land[i][j]);

printf("\n");

}

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++)printf("%d ",T[i][j]);

printf("\n");

}

}

int main(){

freopen("t1p2\_in.txt","r",stdin);

// freopen("test4\_answer(godsight).txt","w",stdout);

int CNT;

A=10;

B=10;

scanf("%d",&CNT);

while(CNT--){

init();

//checkmap();

priority\_queue < node , vector<node> >q;

node str(A,1,0);

T[A][1]=0;

q.push(str);

while( !q.empty() ){

node tmp=q.top();

q.pop();

int rx,ry,rt=tmp.t+1;

for(int r=0;r<8;r++){

rx=tmp.x+dx[r];

ry=tmp.y+dy[r];

if( land[rx][ry]==2 ){

if(ans>rt)ans=rt;

}else{

//printf("[%d][%d]: %d %d \n",rx,ry,land[rx][ry],T[rx][ry]);

if( land[rx][ry]==0 && T[rx][ry]>rt ){

T[rx][ry]=rt;

node N(rx,ry,rt);

q.push(N);

}

}

}

}

T[1][B]=ans;

printf("%d\n",ans);

if(ans==10005){

printf("++++++\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++)printf("%d\t",land[i][j]);

printf("\n");

}

printf("++++++\n");

}

/\*

for(int i=1;i<=A;i++){

for(int j=1;j<=B;j++)printf("%d\t",T[i][j]);

printf("\n");

}

printf("\n");

\*/

/\*

1 初始化：读入每个格子的状态（0空地，1障碍，2终点，3起点），T表示每个格子的最早到达时间（初始化为正无穷)

2 将起点推入优先队列，关键字为T；

3 取出队头，观察对头所在点周围8点（边界更少）的情况；如果该点可达且经由对头达到该点用的时间 T 比当前的 T 更小，则更新并入队；否则不处理

4 重复 4 直到优先队列为空

5 从T中读出结果

\*/

}

return 0;

}

附录三 右图转左图 10x10 实验.html

说明：python编译

**from** PIL **import** Image

**import** numpy **as** np

img **=** Image**.**open('t2p23.jpg') *# 使用 PIL 打开图片*

data1 **=** img**.**getdata() *# 获取图片的数据信息 class <'ImagingCore'>*

data1 **=** np**.**array(data1) *# 把这个数据通过 numpy 转换成多维度的张量*

data1

data**=**np**.**zeros(100)**.**reshape(10,10)

**for** i **in** range(10):

**for** j **in** range(10):

cnt**=**0

**for** ii **in** range(10):

**for** jj **in** range(10):

a1**=**data1[ 1000**\***i**+**10**\***j**+**100**\***ii**+**jj,0 ]

a2**=**data1[ 1000**\***i**+**10**\***j**+**100**\***ii**+**jj,1 ]

a3**=**data1[ 1000**\***i**+**10**\***j**+**100**\***ii**+**jj,2 ]

**if**( abs(a1**-**a2)**>**10 **or** abs(a1**-**a3)**>**10 **or** abs(a2**-**a3)**>**10 ):

cnt**+=**1

**if**(cnt**>**50):

data[i,j]**=**1

**else**:

data[i,j]**=**0

**import** pandas **as** pd

writer **=** pd**.**ExcelWriter('transform2.xlsx')

data\_df **=** pd**.**DataFrame(data) *#关键1，将ndarray格式转换为DataFrame*

*# 更改表的索引*

data\_df**.**columns **=** ['1','2','3','4','5','6','7','8','9','10'] *#将第一行的0,1,2,...,9变成A,B,C,...,J*

data\_df**.**index **=** ['1','2','3','4','5','6','7','8','9','10']

data\_df**.**to\_excel(writer,'page\_1',float\_format**=**'%.5f') *#关键3，float\_format 控制精度，将data\_df写到hhh表格的第一页中。若多个文件，可以在page\_2中写入*

writer**.**save() *#关键4*

附录四 图3的测试代码（试题数据）.cpp

说明：对应数据 test3

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF=0x3f3f3f3f;

const long long mod=1000000007;

const double e=2.718281828459045;

const double pi=3.1415926535;

#define CK cout<<"OK\n";

int land[100][100]; //地块状态 0空地，1障碍，2终点，3起点

int T[100][100]; //时间

int A,B;

int sta[100][100]; //格子的状态 -1 看不到 0 看到了

int minf[15][15]; //理想代价

int N; //测试数

const int NANT=100\*100+5;

int ans;

int dx[8]={1,1,0,-1,-1,-1,0,1};

int dy[8]={0,1,1,1,0,-1,-1,-1};

// 来源矩阵也有关

//int from[100][100]; //来源矩阵

int X,Y; //当前位置

void init(){

ans=NANT;

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++){

T[i][j]=NANT;

land[i][j]=1;

}

}

for(int i=1;i<=10;i++)for(int j=1;j<=10;j++)scanf("%d",&land[i][j]);

for(int i=1;i<=A;i++)for(int j=1;j<=B;j++){

sta[i][j]=-1;

}

X=A;

Y=1;

sta[X][Y]=0;

}

void checkmap(){

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++)printf("%d ",land[i][j]);

printf("\n");

}

}

//扫描可见域

void SCAN(){

sta[X][Y]=0;

double d,h;

//横向测试

//本格最上的h和最下的h，用于避免直接令h=1带来的第一列的bug

int Hup=1,Hdown=1;

while( land[X-Hup][Y]!=1 )Hup++;

while( land[X+Hdown][Y]!=1 )Hdown++;

int Hright=4,Hleft=1; // 纵向测试器的h

//横向扫描：右上

//检疫完成

d=0;

h=Hup;

for(int py=Y+1;py<=B;py++){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=py-Y; //h1为当前高度差，d1为当前横向距离

while( land[X-h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X-ph>0 ){

if(ph<h-1)sta[X-ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X-ph][py]!=1)sta[X-ph][py]=0;

else{

sta[X-ph][py]=0;

if( land[X-ph][py-1]!=1 && land[X-ph-1][py-1]==1 )sta[X-ph-1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1){

Hright=py-Y;

break;

}

}

//横向扫描：右下

//检疫完成

d=0;

h=Hdown;

for(int py=Y+1;py<=B;py++){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=py-Y; //h1为当前高度差，d1为当前横向距离

while( land[X+h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X+ph<=A ){

if(ph<h-1)sta[X+ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X+ph][py]!=1)sta[X+ph][py]=0;

else{

sta[X+ph][py]=0;

if( land[X+ph][py-1]!=1 && land[X+ph+1][py-1]==1 )sta[X+ph+1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1)break;

}

//横向扫描：左上

//检疫完成

d=0;

h=Hup;

for(int py=Y-1;py>=0;py--){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=Y-py; //h1为当前高度差，d1为当前横向距离

while( land[X-h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X-ph>0 ){

if(ph<h-1)sta[X-ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X-ph][py]!=1)sta[X-ph][py]=0;

else{

sta[X-ph][py]=0;

if( land[X-ph][py+1]!=1 && land[X-ph-1][py+1]==1 )sta[X-ph-1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1){

Hleft=Y-py;

break;

}

}

//横向扫描：左下

//检疫完成

d=0;

h=Hdown;

for(int py=Y-1;py>=1;py--){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=Y-py; //h1为当前高度差，d1为当前横向距离

while( land[X+h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X+ph<=A ){

if(ph<h-1)sta[X+ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X+ph][py]!=1)sta[X+ph][py]=0;

else{

sta[X+ph][py]=0;

if( land[X+ph][py+1]!=1 && land[X+ph+1][py+1]==1 )sta[X+ph+1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1)break;

}

//纵向检测

//纵向扫描：右上

d=0;

h=Hright;

for(int px=X-1;px>=1;px--){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=X-px; //h1为当前高度差，d1为当前横向距离

while( land[px][Y+h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y+ph<=B ){

if(ph<h-1)sta[px][Y+ph]=0;

else{

if( land[px][Y+ph]!=1 )sta[px][Y+ph]=0;

else{

sta[px][Y+ph]=0;

if( land[px+1][Y+ph]!=1 && land[px+1][Y+ph+1]==1 )sta[px][Y+ph+1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

//纵向扫描：右下

d=0;

h=Hright;

for(int px=X+1;px<=A;px++){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=px-X; //h1为当前高度差，d1为当前横向距离

while( land[px][Y+h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y+ph<=B ){

if(ph<h-1)sta[px][Y+ph]=0;

else{

if( land[px][Y+ph]!=1 )sta[px][Y+ph]=0;

else{

sta[px][Y+ph]=0;

if( land[px-1][Y+ph]!=1 && land[px-1][Y+ph+1]==1 )sta[px][Y+ph+1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

//纵向扫描：左上

d=0;

h=Hleft;

for(int px=X-1;px>=1;px--){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=X-px; //h1为当前高度差，d1为当前横向距离

while( land[px][Y-h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y-ph>=1 ){

if(ph<h-1)sta[px][Y-ph]=0;

else{

if( land[px][Y-ph]!=1 )sta[px][Y-ph]=0;

else{

sta[px][Y-ph]=0;

if( land[px+1][Y-ph]!=1 && land[px+1][Y-ph-1]==1 )sta[px][Y-ph-1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

//纵向扫描：左下

d=0;

h=Hleft;

for(int px=X+1;px<=A;px++){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=px-X; //h1为当前高度差，d1为当前横向距离

while( land[px][Y-h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y-ph>=1 ){

if(ph<h-1)sta[px][Y-ph]=0;

else{

if( land[px][Y-ph]!=1 )sta[px][Y-ph]=0;

else{

sta[px][Y-ph]=0;

if( land[px-1][Y-ph]!=1 && land[px-1][Y-ph-1]==1 )sta[px][Y-ph-1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

}

void check\_eyesight(){

printf("此时的可见域：\n");

for(int i=1;i<=A;i++){

for(int j=1;j<=B;j++)printf("%d\t",sta[i][j]);

printf("\n");

}

}

struct node{

int x;

int y;

int t;

node(int x,int y,int t):x(x),y(y),t(t){}

friend bool operator<(const node &WW,const node &VV ){

return WW.t<VV.t;

}

};

bool check\_edge(int x,int y){

for(int turn=0;turn<8;turn++){

int tx=x+dx[turn];

int ty=y+dy[turn];

if( (sta[tx][ty]==-1 && land[tx][ty]==0) || land[tx][ty]==2 ){

return 1;

}

}

return 0;

}

node step(){

int movement[15][15];

int reach[15][15];

bool edge[15][15];

memset(edge,0,sizeof(edge));

memset(reach,10005,sizeof(reach));

memset(movement,-1,sizeof(movement));

vector<node> v; //边缘向量组

v.clear();

priority\_queue <node> q; //工作队列

while(!q.empty())q.pop();

reach[X][Y]=0;

movement[X][Y]=-1;

edge[X][Y]=0;

q.push(node(X,Y,0));

while(!q.empty()){

node tmp=q.top();

q.pop();

if(land[tmp.x][tmp.y]==2)continue; //是终点

for(int turn=0;turn<8;turn++){

int tx=tmp.x+dx[turn];

int ty=tmp.y+dy[turn];

// if(sta[tx][ty]==-1)continue; //看不见不管

if( land[tx][ty]==1 )continue;

int tt=tmp.t+1;

if(reach[tx][ty]>tt){

reach[tx][ty]=tt;

q.push(node(tx,ty,tt));

movement[tx][ty]=turn;

}

if(edge[tx][ty]==0 && land[tx][ty]!=1 && sta[tx][ty]==0 ){

if( land[tx][ty]==2 || check\_edge(tx,ty) ){ //进入 边缘向量组，之后要检测

//movement[tx][ty]=turn;

v.push\_back(node(tx,ty,tt));

edge[tx][ty]=1;

}

//加入edge

}

//完成周围点检测

}

}

//基本检测完成

/\*

printf("size: %d\n",v.size());

printf("edge\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(edge[i][j])printf("1\t");

else printf("0\t");

}

printf("\n");

}

/\*

printf("movement:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",movement[i][j]);

}

printf("\n");

}

\*/

if(sta[1][B]!=-1){ //能看见终点，则必往终点跑

v.push\_back(node(1,B,0));

edge[1][B]=1;

}

/\*

printf("reach:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(sta[i][j]==-1){

printf("?\t");

continue;

}

if(edge[i][j])printf("\*");

if(reach[i][j]>NANT)printf("%d\t",-1);

else printf("%d\t",reach[i][j]);

}

printf("\n");

}

\*/

//找到代价+成本最小的

int judge[15][15];

memset(judge,0,sizeof(judge));

int minx,miny,minjudge=10005;

for(int i=0;i<v.size();i++){

int tx=v[i].x;

int ty=v[i].y;

judge[tx][ty]=reach[tx][ty]+minf[tx][ty];

if(judge[tx][ty]<minjudge){

minjudge=judge[tx][ty];

minx=tx;

miny=ty;

}

}

/\*

printf("JUDGE:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",judge[i][j]);

}

printf("\n");

}

cout<<minx<<" "<<miny<<endl;

\*/

//minx,miny 是当前的目标点

//回溯目标点

int targetx=minx,targety=miny,lastx,lasty;

int backx,backy;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

while(lastx!=X || lasty!=Y){

targetx=lastx;

targety=lasty;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

}

// cout<<targetx<<" "<<targety<<endl;

//此时 target[x y]就是下一步的目标

return node(targetx,targety,0);

}

int main(){

A=10;

B=10;

freopen("test3.txt","r",stdin);

scanf("%d",&N);

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

minf[i][j]=max( i-1 , B-j );

// printf("%d\t",minf[i][j]);

}

// printf("\n");

}

/\*

//单点测试区

init();

X=4,Y=6;

SCAN();

check\_eyesight();

step();

\*/

//主程序程序

int cnt=0;

for(int I=0;I<N;I++){

init();

cnt=0;

printf("time: %d \t\t [%d][%d]\n",cnt++,A,1);

while( X!=1 || Y!=B ){ //这里假设图一定可以走通

//扫描可见域

SCAN();

//check\_eyesight();

node NEXT=step();

X=NEXT.x;

Y=NEXT.y;

printf("time: %d \t\t [%d][%d]\n",cnt++,X,Y);

}

//printf("time: %d\t [%d][%d]\n",cnt++,1,B);

}

return 0;

}

附录五 图3的测试代码（自动生成的数据）.cpp

说明：对应数据 test2（可用数据生成器生成）

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF=0x3f3f3f3f;

const long long mod=1000000007;

const double e=2.718281828459045;

const double pi=3.1415926535;

#define CK cout<<"OK\n";

int land[100][100]; //地块状态 0空地，1障碍，2终点，3起点

int T[100][100]; //时间

int A,B;

int sta[100][100]; //格子的状态 -1 看不到 0 看到了

int minf[15][15]; //理想代价

int N; //测试数

const int NANT=100\*100+5;

int ans;

int dx[8]={1,1,0,-1,-1,-1,0,1};

int dy[8]={0,1,1,1,0,-1,-1,-1};

// 来源矩阵也有关

//int from[100][100]; //来源矩阵

int X,Y; //当前位置

void init(){

ans=NANT;

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++){

T[i][j]=NANT;

land[i][j]=1;

}

}

for(int i=1;i<=10;i++)for(int j=1;j<=10;j++)scanf("%d",&land[i][j]);

for(int i=1;i<=A;i++)for(int j=1;j<=B;j++){

sta[i][j]=-1;

}

X=A;

Y=1;

sta[X][Y]=0;

}

void checkmap(){

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++)printf("%d ",land[i][j]);

printf("\n");

}

}

//扫描可见域

void SCAN(){

sta[X][Y]=0;

double d,h;

//横向测试

//本格最上的h和最下的h，用于避免直接令h=1带来的第一列的bug

int Hup=1,Hdown=1;

while( land[X-Hup][Y]!=1 )Hup++;

while( land[X+Hdown][Y]!=1 )Hdown++;

int Hright=4,Hleft=1; // 纵向测试器的h

//横向扫描：右上

//检疫完成

d=0;

h=Hup;

for(int py=Y+1;py<=B;py++){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=py-Y; //h1为当前高度差，d1为当前横向距离

while( land[X-h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X-ph>0 ){

if(ph<h-1)sta[X-ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X-ph][py]!=1)sta[X-ph][py]=0;

else{

sta[X-ph][py]=0;

if( land[X-ph][py-1]!=1 && land[X-ph-1][py-1]==1 )sta[X-ph-1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1){

Hright=py-Y;

break;

}

}

//横向扫描：右下

//检疫完成

d=0;

h=Hdown;

for(int py=Y+1;py<=B;py++){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=py-Y; //h1为当前高度差，d1为当前横向距离

while( land[X+h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X+ph<=A ){

if(ph<h-1)sta[X+ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X+ph][py]!=1)sta[X+ph][py]=0;

else{

sta[X+ph][py]=0;

if( land[X+ph][py-1]!=1 && land[X+ph+1][py-1]==1 )sta[X+ph+1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1)break;

}

//横向扫描：左上

//检疫完成

d=0;

h=Hup;

for(int py=Y-1;py>=0;py--){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=Y-py; //h1为当前高度差，d1为当前横向距离

while( land[X-h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X-ph>0 ){

if(ph<h-1)sta[X-ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X-ph][py]!=1)sta[X-ph][py]=0;

else{

sta[X-ph][py]=0;

if( land[X-ph][py+1]!=1 && land[X-ph-1][py+1]==1 )sta[X-ph-1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1){

Hleft=Y-py;

break;

}

}

//横向扫描：左下

//检疫完成

d=0;

h=Hdown;

for(int py=Y-1;py>=1;py--){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=Y-py; //h1为当前高度差，d1为当前横向距离

while( land[X+h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X+ph<=A ){

if(ph<h-1)sta[X+ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X+ph][py]!=1)sta[X+ph][py]=0;

else{

sta[X+ph][py]=0;

if( land[X+ph][py+1]!=1 && land[X+ph+1][py+1]==1 )sta[X+ph+1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1)break;

}

//纵向检测

//纵向扫描：右上

d=0;

h=Hright;

for(int px=X-1;px>=1;px--){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=X-px; //h1为当前高度差，d1为当前横向距离

while( land[px][Y+h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y+ph<=B ){

if(ph<h-1)sta[px][Y+ph]=0;

else{

if( land[px][Y+ph]!=1 )sta[px][Y+ph]=0;

else{

sta[px][Y+ph]=0;

if( land[px+1][Y+ph]!=1 && land[px+1][Y+ph+1]==1 )sta[px][Y+ph+1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

//纵向扫描：右下

d=0;

h=Hright;

for(int px=X+1;px<=A;px++){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=px-X; //h1为当前高度差，d1为当前横向距离

while( land[px][Y+h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y+ph<=B ){

if(ph<h-1)sta[px][Y+ph]=0;

else{

if( land[px][Y+ph]!=1 )sta[px][Y+ph]=0;

else{

sta[px][Y+ph]=0;

if( land[px-1][Y+ph]!=1 && land[px-1][Y+ph+1]==1 )sta[px][Y+ph+1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

//纵向扫描：左上

d=0;

h=Hleft;

for(int px=X-1;px>=1;px--){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=X-px; //h1为当前高度差，d1为当前横向距离

while( land[px][Y-h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y-ph>=1 ){

if(ph<h-1)sta[px][Y-ph]=0;

else{

if( land[px][Y-ph]!=1 )sta[px][Y-ph]=0;

else{

sta[px][Y-ph]=0;

if( land[px+1][Y-ph]!=1 && land[px+1][Y-ph-1]==1 )sta[px][Y-ph-1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

//纵向扫描：左下

d=0;

h=Hleft;

for(int px=X+1;px<=A;px++){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=px-X; //h1为当前高度差，d1为当前横向距离

while( land[px][Y-h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y-ph>=1 ){

if(ph<h-1)sta[px][Y-ph]=0;

else{

if( land[px][Y-ph]!=1 )sta[px][Y-ph]=0;

else{

sta[px][Y-ph]=0;

if( land[px-1][Y-ph]!=1 && land[px-1][Y-ph-1]==1 )sta[px][Y-ph-1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

}

void check\_eyesight(){

printf("此时的可见域：\n");

for(int i=1;i<=A;i++){

for(int j=1;j<=B;j++)printf("%d\t",sta[i][j]);

printf("\n");

}

}

struct node{

int x;

int y;

int t;

node(int x,int y,int t):x(x),y(y),t(t){}

friend bool operator<(const node &WW,const node &VV ){

return WW.t<VV.t;

}

};

bool check\_edge(int x,int y){

for(int turn=0;turn<8;turn++){

int tx=x+dx[turn];

int ty=y+dy[turn];

if( (sta[tx][ty]==-1 && land[tx][ty]==0) || land[tx][ty]==2 ){

return 1;

}

}

return 0;

}

node step(){

int movement[15][15];

int reach[15][15];

bool edge[15][15];

memset(edge,0,sizeof(edge));

memset(reach,10005,sizeof(reach));

memset(movement,-1,sizeof(movement));

vector<node> v; //边缘向量组

v.clear();

priority\_queue <node> q; //工作队列

while(!q.empty())q.pop();

reach[X][Y]=0;

movement[X][Y]=-1;

edge[X][Y]=0;

q.push(node(X,Y,0));

while(!q.empty()){

node tmp=q.top();

q.pop();

if(land[tmp.x][tmp.y]==2)continue; //是终点

for(int turn=0;turn<8;turn++){

int tx=tmp.x+dx[turn];

int ty=tmp.y+dy[turn];

// if(sta[tx][ty]==-1)continue; //看不见不管

if( land[tx][ty]==1 )continue;

int tt=tmp.t+1;

if(reach[tx][ty]>tt){

reach[tx][ty]=tt;

q.push(node(tx,ty,tt));

movement[tx][ty]=turn;

}

if(edge[tx][ty]==0 && land[tx][ty]!=1 && sta[tx][ty]==0 ){

if( land[tx][ty]==2 || check\_edge(tx,ty) ){ //进入 边缘向量组，之后要检测

//movement[tx][ty]=turn;

v.push\_back(node(tx,ty,tt));

edge[tx][ty]=1;

}

//加入edge

}

//完成周围点检测

}

}

//基本检测完成

/\*

printf("size: %d\n",v.size());

printf("edge\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(edge[i][j])printf("1\t");

else printf("0\t");

}

printf("\n");

}

/\*

printf("movement:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",movement[i][j]);

}

printf("\n");

}

\*/

if(sta[1][B]!=-1){ //能看见终点，则必往终点跑

v.push\_back(node(1,B,0));

edge[1][B]=1;

}

/\*

printf("reach:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(sta[i][j]==-1){

printf("?\t");

continue;

}

if(edge[i][j])printf("\*");

if(reach[i][j]>NANT)printf("%d\t",-1);

else printf("%d\t",reach[i][j]);

}

printf("\n");

}

\*/

//找到代价+成本最小的

int judge[15][15];

memset(judge,0,sizeof(judge));

int minx,miny,minjudge=10005;

for(int i=0;i<v.size();i++){

int tx=v[i].x;

int ty=v[i].y;

judge[tx][ty]=reach[tx][ty]+minf[tx][ty];

if(judge[tx][ty]<minjudge){

minjudge=judge[tx][ty];

minx=tx;

miny=ty;

}

}

/\*

printf("JUDGE:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",judge[i][j]);

}

printf("\n");

}

cout<<minx<<" "<<miny<<endl;

\*/

//minx,miny 是当前的目标点

//回溯目标点

int targetx=minx,targety=miny,lastx,lasty;

int backx,backy;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

while(lastx!=X || lasty!=Y){

targetx=lastx;

targety=lasty;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

}

// cout<<targetx<<" "<<targety<<endl;

//此时 target[x y]就是下一步的目标

return node(targetx,targety,0);

}

int main(){

A=10;

B=10;

freopen("test2.txt","r",stdin);

scanf("%d",&N);

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

minf[i][j]=max( i-1 , B-j );

// printf("%d\t",minf[i][j]);

}

// printf("\n");

}

/\*

//单点测试区

init();

X=4,Y=6;

SCAN();

check\_eyesight();

step();

\*/

//主程序程序

int cnt=0;

for(int I=0;I<N;I++){

init();

cnt=0;

printf("time: %d \t\t [%d][%d]\n",cnt++,A,1);

while( X!=1 || Y!=B ){ //这里假设图一定可以走通

//扫描可见域

SCAN();

//check\_eyesight();

node NEXT=step();

X=NEXT.x;

Y=NEXT.y;

printf("time: %d \t\t [%d][%d]\n",cnt++,X,Y);

}

//printf("time: %d\t [%d][%d]\n",cnt++,1,B);

}

return 0;

}

附录六 图4扫描改良改(右图测试)

说明：用 图4扫描改良改(右图测试).cpp 测试

对应数据 newscan\_in

先用 520校赛 右图转左图 输出100乘100.html 中的python代码处理图片

导出到指定excel后 将数据放入newscan\_in 即可

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF=0x3f3f3f3f;

const long long mod=1000000007;

const double e=2.718281828459045;

const double pi=3.1415926535;

#define CK cout<<"OK\n";

// 注释的详细说明见《纸说明1》

int T[100][100]; //时间

int A,B;

int sta[100][100]; //格子的状态 -1 看不到 0 看到了

int land[100][100]; //格子地形 -1 看不到 0 能走 1 不能走

int mas[105][105]; //原始信息 0 白 1 黑

int truemas[15][15]; //真实地图信息 0 白 1 黑

int mas1[105][105]; //hash信息 0 白 1 黑

int massrate[15][15]; //目标方格被遮挡水平

int N; //测试数

int Srate; //衍射水平：S% 为不可识别界限

//massrate 大于这个值就不能识别了

const int NANT=100\*100+5;

int ans;

int dx[8]={1,1,0,-1,-1,-1,0,1};

int dy[8]={0,1,1,1,0,-1,-1,-1};

int minf[15][15]; //理想代价

// 来源矩阵也有关

//int from[100][100]; //来源矩阵

int X,Y; //当前位置

void init(){

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++){

T[i][j]=NANT;

land[i][j]=1;

mas[i][j]=1;

truemas[i][j]=1;

}

}

ans=NANT;

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++)

if(i==0 || i==A+1 || j==0 || j==B+1){

massrate[i][j]=0;

sta[i][j]=0;

mas[i][j]=1;

truemas[i][j]=1;

}else{

massrate[i][j]=100;

sta[i][j]=-1;

}

}

for(int i=0;i<100;i++)for(int j=0;j<100;j++)scanf("%d",&mas[i][j]);

memset(mas1,0,sizeof(mas1));

//哈希

for(int i=0;i<100;i++){

int sum=0;

for(int j=0;j<10;j++)sum+=mas[i][j];

for(int j=0;j<=90;j++){

mas1[i][j]=sum;

sum=sum-mas[i][j]+mas[i][j+10];

}

}

memset(mas,0,sizeof(mas));

for(int j=0;j<=90;j++){

int sum=0;

for(int i=0;i<10;i++)sum+=mas1[i][j];

for(int i=0;i<=90;i++){

mas[i][j]=sum;

sum=sum-mas1[i][j]+mas1[i+10][j];

}

}

//之后取用mas即可

//真实地图信息 0 白 1 黑

for(int i=0;i<=9;i++){

for(int j=0;j<=9;j++){

printf("%d\t",mas[i\*10][j\*10]);

}

printf("\n");

}

printf("\n");

}

//扫描可见域

void SCAN(){

for(int i=1;i<=10;i++)for(int j=1;j<=10;j++){

if( massrate[i][j]==0 )continue;

if( abs(j-Y)<=1 && abs(i-X)<=1 ){

massrate[i][j]=0;

continue;

}

int di=X-i;

int dj=Y-j;

int S=0; //最大遮挡水平

//坐标换算：10\*10 -> 100-100

int I=10\*(i-1);

int J=10\*(j-1);

int XX=10\*(X-1);

int YY=10\*(Y-1);

while( I!=XX || J!=YY ){

I+=di;

J+=dj;

if( abs(I-10\*(i-1))<10 && abs(J-10\*(j-1))<10 )continue;

if(mas[I][J]>S)S=mas[I][J];

}

massrate[i][j]=min(S,massrate[i][j]);

}

//可见域形式转化

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(massrate[i][j]<Srate)sta[i][j]=0;

}

}

}

void check\_eyesight(){

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++)printf("%d\t",massrate[i][j]);

printf("\n");

}

}

void check\_eyesight\_01(){

printf("eyesight\n");

for(int i=0;i<=11;i++){

for(int j=0;j<=11;j++)printf("%d\t",sta[i][j]);

printf("\n");

}

}

struct node{

int x;

int y;

int t;

node(int x,int y,int t):x(x),y(y),t(t){}

friend bool operator<(const node &WW,const node &VV ){

return WW.t<VV.t;

}

};

bool check\_edge(int x,int y){

for(int turn=0;turn<8;turn++){

int tx=x+dx[turn];

int ty=y+dy[turn];

if( (sta[tx][ty]==-1 && land[tx][ty]==0) || land[tx][ty]==2 ){

return 1;

}

}

return 0;

}

vector<node> v; //边缘向量组

node step(){

int movement[15][15];

int reach[15][15];

bool edge[15][15];

memset(edge,0,sizeof(edge));

memset(reach,10005,sizeof(reach));

memset(movement,-1,sizeof(movement));

v.clear();

priority\_queue <node> q; //工作队列

while(!q.empty())q.pop();

reach[X][Y]=0;

movement[X][Y]=-1;

edge[X][Y]=0;

q.push(node(X,Y,0));

while(!q.empty()){

node tmp=q.top();

q.pop();

if(land[tmp.x][tmp.y]==2)continue; //是终点

for(int turn=0;turn<8;turn++){

int tx=tmp.x+dx[turn];

int ty=tmp.y+dy[turn];

// if(sta[tx][ty]==-1)continue; //看不见不管

if( land[tx][ty]==1 )continue;

int tt=tmp.t+1;

if(reach[tx][ty]>tt){

reach[tx][ty]=tt;

q.push(node(tx,ty,tt));

movement[tx][ty]=turn;

}

if(edge[tx][ty]==0 && land[tx][ty]!=1 && sta[tx][ty]==0 ){

if( land[tx][ty]==2 || check\_edge(tx,ty) ){ //进入 边缘向量组，之后要检测

//movement[tx][ty]=turn;

v.push\_back(node(tx,ty,tt));

edge[tx][ty]=1;

}

//加入edge

}

//完成周围点检测

}

}

//基本检测完成

// printf("size: %d\n",v.size());

// check\_eyesight\_01();

/\*

printf("edge\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(edge[i][j])printf("1\t");

else printf("0\t");

}

printf("\n");

}

/\*

printf("movement:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",movement[i][j]);

}

printf("\n");

}

if(sta[1][B]!=-1){ //能看见终点，则必往终点跑

v.push\_back(node(1,B,0));

edge[1][B]=1;

}

printf("reach:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(sta[i][j]==-1){

printf("?\t");

continue;

}

if(edge[i][j])printf("\*");

if(reach[i][j]>NANT)printf("%d\t",-1);

else printf("%d\t",reach[i][j]);

}

printf("\n");

}

\*/

//找到代价+成本最小的

int judge[15][15];

memset(judge,0,sizeof(judge));

int minx,miny,minjudge=10005;

for(int i=0;i<v.size();i++){

int tx=v[i].x;

int ty=v[i].y;

// cout<<"v "<<tx<<" "<<ty<<endl;

judge[tx][ty]=reach[tx][ty]+minf[tx][ty];

if(judge[tx][ty]<minjudge){

minjudge=judge[tx][ty];

minx=tx;

miny=ty;

}

}

//看到终点：直冲终点

if(edge[1][B]==1){

minjudge=0;

minx=1;

miny=B;

}

/\*

printf("JUDGE:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",judge[i][j]);

}

printf("\n");

}

\*/

// cout<<"min judge : "<<minx<<" "<<miny<<endl;

//minx,miny 是当前的目标点

//回溯目标点

int targetx=minx,targety=miny,lastx,lasty;

int backx,backy;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

while(lastx!=X || lasty!=Y){

targetx=lastx;

targety=lasty;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

}

// cout<<targetx<<" "<<targety<<endl;

//此时 target[x y]就是下一步的目标

return node(targetx,targety,0);

}

void land\_reshape(){

for(int i=0;i<=9;i++){

for(int j=0;j<=9;j++){

if(mas[i\*10][j\*10]<Srate)land[i+1][j+1]=0;

else land[i+1][j+1]=-1;

}

}

land[A][1]=3;

land[1][B]=2;

}

int main(){

A=10;

B=10;

Srate=50;

freopen("newscan\_in.txt","r",stdin);

init();

N=1; //图的个数

land\_reshape(); //把land信息转化成10\*10

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

minf[i][j]=max( i-1 , B-j );

}

}

//land checked

/\*

printf("land\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",land[i][j]);

}

printf("\n");

}

//单点测试区

X=10,Y=1;

SCAN();

check\_eyesight(); //输出整个阵

step();

//主程序程序

\*/

X=10,Y=1;

int cnt=0;

printf("time: %d \t\t [%d][%d]\n",cnt++,A,1);

while( X!=1 || Y!=B ){ //这里假设图一定可以走通

//扫描可见域

SCAN();

//check\_eyesight();

node NEXT=step();

X=NEXT.x;

Y=NEXT.y;

printf("time: %d \t\t [%d][%d]\n",cnt++,X,Y);

}

//printf("time: %d\t [%d][%d]\n",cnt++,1,B);

return 0;

}

附录七 520校赛 右图转左图 输出100乘100.html

说明：python运行

**from** PIL **import** Image

**import** numpy **as** np

img **=** Image**.**open('t2p23.jpg') *# 使用 PIL 打开图片*

data1 **=** img**.**getdata() *# 获取图片的数据信息 class <'ImagingCore'>*

data1 **=** np**.**array(data1) *# 把这个数据通过 numpy 转换成多维度的张量*

data1

data**=**np**.**zeros(10000)**.**reshape(100,100)

**for** i **in** range(100):

**for** j **in** range(100):

a1**=**data1[ 100**\***i**+**j,0 ]

a2**=**data1[ 100**\***i**+**j,1 ]

a3**=**data1[ 100**\***i**+**j,2 ]

**if**( abs(a1**-**a2)**>**5 **or** abs(a1**-**a3)**>**5 **or** abs(a2**-**a3)**>**5 ):

data[i,j]**=**1

**else**:

data[i,j]**=**0

**import** pandas **as** pd

writer **=** pd**.**ExcelWriter('transform\_for\_cpp.xlsx')

data\_df **=** pd**.**DataFrame(data) *#关键1，将ndarray格式转换为DataFrame*

*# 更改表的索引*

data\_df**.**columns **=** ['0','1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','35','36','37','38','39','40','41','42','43','44','45','46','47','48','49','50','51','52','53','54','55','56','57','58','59','60','61','62','63','64','65','66','67','68','69','70','71','72','73','74','75','76','77','78','79','80','81','82','83','84','85','86','87','88','89','90','91','92','93','94','95','96','97','98','99'] *#将第一行的0,1,2,...,9变成A,B,C,...,J*

data\_df**.**index **=** ['0','1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','35','36','37','38','39','40','41','42','43','44','45','46','47','48','49','50','51','52','53','54','55','56','57','58','59','60','61','62','63','64','65','66','67','68','69','70','71','72','73','74','75','76','77','78','79','80','81','82','83','84','85','86','87','88','89','90','91','92','93','94','95','96','97','98','99']

data\_df**.**to\_excel(writer,'page\_1',float\_format**=**'%.5f') *#关键3，float\_format 控制精度，将data\_df写到hhh表格的第一页中。若多个文件，可以在page\_2中写入*

writer**.**save() *#关键4*

附录八 第二题K.cpp

说明：蚁群算法解决图3使用第二题K.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF=0x3f3f3f3f;

const long long mod=1000000007;

const double e=2.718281828459045;

const double pi=3.1415926535;

#define CK cout<<"OK\n";

int land[100][100]; //地块状态 0空地，1障碍，2终点，3起点

int T[100][100]; //时间

int A,B;

int sta[100][100]; //格子的状态 -1 看不到 0 看到了

double minf[15][15]; //理想代价

int N; //测试数

const int NANT=100\*100+5;

int ans;

int dx[8]={1,1,0,-1,-1,-1,0,1};

int dy[8]={0,1,1,1,0,-1,-1,-1};

double K;

// 来源矩阵也有关

//int from[100][100]; //来源矩阵

int X,Y; //当前位置

void init(){

ans=NANT;

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++){

T[i][j]=NANT;

land[i][j]=1;

sta[i][j]=0;

}

}

for(int i=1;i<=10;i++)for(int j=1;j<=10;j++)scanf("%d",&land[i][j]);

for(int i=1;i<=A;i++)for(int j=1;j<=B;j++){

sta[i][j]=-1;

}

X=A;

Y=1;

sta[X][Y]=0;

}

void checkmap(){

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++)printf("%d ",land[i][j]);

printf("\n");

}

}

//扫描可见域

void SCAN(){

sta[X][Y]=0;

double d,h;

//横向测试

//本格最上的h和最下的h，用于避免直接令h=1带来的第一列的bug

int Hup=1,Hdown=1;

while( land[X-Hup][Y]!=1 )Hup++;

while( land[X+Hdown][Y]!=1 )Hdown++;

int Hright=4,Hleft=1; // 纵向测试器的h

//横向扫描：右上

//检疫完成

d=0;

h=Hup;

for(int py=Y+1;py<=B;py++){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=py-Y; //h1为当前高度差，d1为当前横向距离

while( land[X-h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X-ph>0 ){

if(ph<h-1)sta[X-ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X-ph][py]!=1)sta[X-ph][py]=0;

else{

sta[X-ph][py]=0;

if( land[X-ph][py-1]!=1 && land[X-ph-1][py-1]==1 )sta[X-ph-1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1){

Hright=py-Y;

break;

}

}

//横向扫描：右下

//检疫完成

d=0;

h=Hdown;

for(int py=Y+1;py<=B;py++){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=py-Y; //h1为当前高度差，d1为当前横向距离

while( land[X+h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X+ph<=A ){

if(ph<h-1)sta[X+ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X+ph][py]!=1)sta[X+ph][py]=0;

else{

sta[X+ph][py]=0;

if( land[X+ph][py-1]!=1 && land[X+ph+1][py-1]==1 )sta[X+ph+1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1)break;

}

//横向扫描：左上

//检疫完成

d=0;

h=Hup;

for(int py=Y-1;py>=0;py--){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=Y-py; //h1为当前高度差，d1为当前横向距离

while( land[X-h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X-ph>0 ){

if(ph<h-1)sta[X-ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X-ph][py]!=1)sta[X-ph][py]=0;

else{

sta[X-ph][py]=0;

if( land[X-ph][py+1]!=1 && land[X-ph-1][py+1]==1 )sta[X-ph-1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1){

Hleft=Y-py;

break;

}

}

//横向扫描：左下

//检疫完成

d=0;

h=Hdown;

for(int py=Y-1;py>=1;py--){

sta[X][py]=0; //标记本格为可知

int h1=0,d1=Y-py; //h1为当前高度差，d1为当前横向距离

while( land[X+h1][py]!=1 ){ //更新下一列的阻碍情况

h1++; //直到遇到方格，算出本列的障碍高度

}

int ph=1;

while( h\*d1-ph\*d>0 && X+ph<=A ){

if(ph<h-1)sta[X+ph][py]=0; //不及前面的阻碍高，不管有没有障碍，都不妨碍

else{

if(land[X+ph][py]!=1)sta[X+ph][py]=0;

else{

sta[X+ph][py]=0;

if( land[X+ph][py+1]!=1 && land[X+ph+1][py+1]==1 )sta[X+ph+1][py]=0; //特殊情况

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if(land[X][py]==1)break;

}

//纵向检测

//纵向扫描：右上

d=0;

h=Hright;

for(int px=X-1;px>=1;px--){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=X-px; //h1为当前高度差，d1为当前横向距离

while( land[px][Y+h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y+ph<=B ){

if(ph<h-1)sta[px][Y+ph]=0;

else{

if( land[px][Y+ph]!=1 )sta[px][Y+ph]=0;

else{

sta[px][Y+ph]=0;

if( land[px+1][Y+ph]!=1 && land[px+1][Y+ph+1]==1 )sta[px][Y+ph+1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

//纵向扫描：右下

d=0;

h=Hright;

for(int px=X+1;px<=A;px++){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=px-X; //h1为当前高度差，d1为当前横向距离

while( land[px][Y+h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y+ph<=B ){

if(ph<h-1)sta[px][Y+ph]=0;

else{

if( land[px][Y+ph]!=1 )sta[px][Y+ph]=0;

else{

sta[px][Y+ph]=0;

if( land[px-1][Y+ph]!=1 && land[px-1][Y+ph+1]==1 )sta[px][Y+ph+1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

//纵向扫描：左上

d=0;

h=Hleft;

for(int px=X-1;px>=1;px--){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=X-px; //h1为当前高度差，d1为当前横向距离

while( land[px][Y-h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y-ph>=1 ){

if(ph<h-1)sta[px][Y-ph]=0;

else{

if( land[px][Y-ph]!=1 )sta[px][Y-ph]=0;

else{

sta[px][Y-ph]=0;

if( land[px+1][Y-ph]!=1 && land[px+1][Y-ph-1]==1 )sta[px][Y-ph-1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

//纵向扫描：左下

d=0;

h=Hleft;

for(int px=X+1;px<=A;px++){

sta[px][Y]=0; //标记本格为可知

int h1=0,d1=px-X; //h1为当前高度差，d1为当前横向距离

while( land[px][Y-h1]!=1 )h1++; //更新下一列的阻碍情况

// cout<<h1<<endl;

int ph=1;

while( h\*d1-ph\*d>0 && Y-ph>=1 ){

if(ph<h-1)sta[px][Y-ph]=0;

else{

if( land[px][Y-ph]!=1 )sta[px][Y-ph]=0;

else{

sta[px][Y-ph]=0;

if( land[px-1][Y-ph]!=1 && land[px-1][Y-ph-1]==1 )sta[px][Y-ph-1]=0;

break;

}

}

ph++;

}

if( h\*d1-d\*h1>0 ){

h=h1;

d=d1;

}

if( land[px][Y]==1 )break;

}

}

void check\_eyesight(){

printf("此时的可见域：\n");

for(int i=1;i<=A;i++){

for(int j=1;j<=B;j++)printf("%d\t",sta[i][j]);

printf("\n");

}

}

struct node{

int x;

int y;

int t;

node(int x,int y,int t):x(x),y(y),t(t){}

friend bool operator<(const node &WW,const node &VV ){

return WW.t<VV.t;

}

};

bool check\_edge(int x,int y){

for(int turn=0;turn<8;turn++){

int tx=x+dx[turn];

int ty=y+dy[turn];

if( (sta[tx][ty]==-1 && land[tx][ty]==0) || land[tx][ty]==2 ){

return 1;

}

}

return 0;

}

node step(){

int movement[15][15];

int reach[15][15];

bool edge[15][15];

memset(edge,0,sizeof(edge));

memset(reach,10005,sizeof(reach));

memset(movement,-1,sizeof(movement));

vector<node> v; //边缘向量组

v.clear();

priority\_queue <node> q; //工作队列

while(!q.empty())q.pop();

reach[X][Y]=0;

movement[X][Y]=-1;

edge[X][Y]=0;

q.push(node(X,Y,0));

while(!q.empty()){

node tmp=q.top();

q.pop();

if(land[tmp.x][tmp.y]==2)continue; //是终点

for(int turn=0;turn<8;turn++){

int tx=tmp.x+dx[turn];

int ty=tmp.y+dy[turn];

// if(sta[tx][ty]==-1)continue; //看不见不管

if( land[tx][ty]==1 )continue;

int tt=tmp.t+1;

if(reach[tx][ty]>tt){

reach[tx][ty]=tt;

q.push(node(tx,ty,tt));

movement[tx][ty]=turn;

}

if(edge[tx][ty]==0 && land[tx][ty]!=1 && sta[tx][ty]==0 ){

if( land[tx][ty]==2 || check\_edge(tx,ty) ){ //进入 边缘向量组，之后要检测

//movement[tx][ty]=turn;

v.push\_back(node(tx,ty,tt));

edge[tx][ty]=1;

}

//加入edge

}

//完成周围点检测

}

}

//基本检测完成

/\*

printf("size: %d\n",v.size());

printf("edge\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(edge[i][j])printf("1\t");

else printf("0\t");

}

printf("\n");

}

/\*

printf("movement:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",movement[i][j]);

}

printf("\n");

}

\*/

if(sta[1][B]!=-1){ //能看见终点，则必往终点跑

v.push\_back(node(1,B,0));

edge[1][B]=1;

}

/\*

printf("reach:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(sta[i][j]==-1){

printf("?\t");

continue;

}

if(edge[i][j])printf("\*");

if(reach[i][j]>NANT)printf("%d\t",-1);

else printf("%d\t",reach[i][j]);

}

printf("\n");

}

\*/

//找到代价+成本最小的

double judge[15][15];

memset(judge,0,sizeof(judge));

int minx,miny;

double minjudge=10005;

for(int i=0;i<v.size();i++){

int tx=v[i].x;

int ty=v[i].y;

judge[tx][ty]=reach[tx][ty]+K\*minf[tx][ty];

if(judge[tx][ty]<minjudge){

minjudge=judge[tx][ty];

minx=tx;

miny=ty;

}

}

if(sta[1][B]!=-1){

minx=1;

miny=B;

}

/\*

printf("JUDGE:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%.4lf\t",judge[i][j]);

}

printf("\n");

}

cout<<minx<<" "<<miny<<endl;

\*/

//minx,miny 是当前的目标点

//回溯目标点

int targetx=minx,targety=miny,lastx,lasty;

int backx,backy;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

while(lastx!=X || lasty!=Y){

targetx=lastx;

targety=lasty;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

}

// cout<<targetx<<" "<<targety<<endl;

//此时 target[x y]就是下一步的目标

return node(targetx,targety,0);

}

int main(){

A=10;

B=10;

freopen("test3.txt","r",stdin);

// freopen("test4\_20k\_answer.txt","w",stdout);

scanf("%d",&N);

double Kseq[25]={10,5,3,2.5,2, 1.8,1.6,1.4,1.2,1.1 ,1,0.9,0.8,0.7,0.6 ,0.5,0.4,0.3,0.2,0.1};

//信息素参数

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

minf[i][j]=max( i-1 , B-j );

// printf("%d\t",minf[i][j]);

}

// printf("\n");

}

/\*

//单点测试区

init();

X=4,Y=6;

SCAN();

check\_eyesight();

step();

\*/

//主程序程序

K=1.2;

int cnt=0;

for(int I=0;I<N;I++){

init();

cnt=0;

printf("time: %d \t\t [%d][%d]\n",cnt++,A,1);

// cnt++;

while( X!=1 || Y!=B ){ //这里假设图一定可以走通

//扫描可见域

SCAN();

// check\_eyesight();

node NEXT=step();

X=NEXT.x;

Y=NEXT.y;

printf("time: %d \t\t [%d][%d]\n",cnt++,X,Y);

// cnt++;

}

// printf("time: %d\t [%d][%d]\n",cnt++,1,B);

// printf("%d ",cnt);

}

printf("\n");

return 0;

}

附录九 第二题扫描改良改+K.cpp

说明：蚁群算法解决图4使用第二题扫描改良改+K.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF=0x3f3f3f3f;

const long long mod=1000000007;

const double e=2.718281828459045;

const double pi=3.1415926535;

#define CK cout<<"OK\n";

// 注释的详细说明见《纸说明1》

int T[100][100]; //时间

int A,B;

int sta[100][100]; //格子的状态 -1 看不到 0 看到了

int land[100][100]; //格子地形 -1 看不到 0 能走 1 不能走

int mas[105][105]; //原始信息 0 白 1 黑

int truemas[15][15]; //真实地图信息 0 白 1 黑

int mas1[105][105]; //hash信息 0 白 1 黑

int massrate[15][15]; //目标方格被遮挡水平

int N; //测试数

int Srate; //衍射水平：S% 为不可识别界限

//massrate 大于这个值就不能识别了

const int NANT=100\*100+5;

int ans;

int dx[8]={1,1,0,-1,-1,-1,0,1};

int dy[8]={0,1,1,1,0,-1,-1,-1};

int minf[15][15]; //理想代价

// 来源矩阵也有关

//int from[100][100]; //来源矩阵

double K;

// 来源矩阵也有关

//int from[100][100]; //来源矩阵

int X,Y; //当前位置

void init(){

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++){

T[i][j]=NANT;

land[i][j]=1;

mas[i][j]=1;

truemas[i][j]=1;

}

}

ans=NANT;

for(int i=0;i<=A+1;i++){

for(int j=0;j<=B+1;j++)

if(i==0 || i==A+1 || j==0 || j==B+1){

massrate[i][j]=0;

sta[i][j]=0;

mas[i][j]=1;

truemas[i][j]=1;

}else{

massrate[i][j]=100;

sta[i][j]=-1;

}

}

for(int i=0;i<100;i++)for(int j=0;j<100;j++)scanf("%d",&mas[i][j]);

memset(mas1,0,sizeof(mas1));

//哈希

for(int i=0;i<100;i++){

int sum=0;

for(int j=0;j<10;j++)sum+=mas[i][j];

for(int j=0;j<=90;j++){

mas1[i][j]=sum;

sum=sum-mas[i][j]+mas[i][j+10];

}

}

memset(mas,0,sizeof(mas));

for(int j=0;j<=90;j++){

int sum=0;

for(int i=0;i<10;i++)sum+=mas1[i][j];

for(int i=0;i<=90;i++){

mas[i][j]=sum;

sum=sum-mas1[i][j]+mas1[i+10][j];

}

}

//之后取用mas即可

//真实地图信息 0 白 1 黑

for(int i=0;i<=9;i++){

for(int j=0;j<=9;j++){

printf("%d\t",mas[i\*10][j\*10]);

}

printf("\n");

}

printf("\n");

}

//扫描可见域

void SCAN(){

for(int i=1;i<=10;i++)for(int j=1;j<=10;j++){

if( massrate[i][j]==0 )continue;

if( abs(j-Y)<=1 && abs(i-X)<=1 ){

massrate[i][j]=0;

continue;

}

int di=X-i;

int dj=Y-j;

int S=0; //最大遮挡水平

//坐标换算：10\*10 -> 100-100

int I=10\*(i-1);

int J=10\*(j-1);

int XX=10\*(X-1);

int YY=10\*(Y-1);

while( I!=XX || J!=YY ){

I+=di;

J+=dj;

if( abs(I-10\*(i-1))<10 && abs(J-10\*(j-1))<10 )continue;

if(mas[I][J]>S)S=mas[I][J];

}

massrate[i][j]=min(S,massrate[i][j]);

}

//可见域形式转化

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(massrate[i][j]<Srate)sta[i][j]=0;

}

}

}

void check\_eyesight(){

printf("eyesight\_real\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++)printf("%d\t",massrate[i][j]);

printf("\n");

}

}

void check\_eyesight\_01(){

printf("eyesight\_turned\n");

for(int i=0;i<=11;i++){

for(int j=0;j<=11;j++)printf("%d\t",sta[i][j]);

printf("\n");

}

}

struct node{

int x;

int y;

int t;

node(int x,int y,int t):x(x),y(y),t(t){}

friend bool operator<(const node &WW,const node &VV ){

return WW.t<VV.t;

}

};

bool check\_edge(int x,int y){

for(int turn=0;turn<8;turn++){

int tx=x+dx[turn];

int ty=y+dy[turn];

if( (sta[tx][ty]==-1 && land[tx][ty]==0) || land[tx][ty]==2 ){

return 1;

}

}

return 0;

}

vector<node> v; //边缘向量组

node step(){

int movement[15][15];

int reach[15][15];

bool edge[15][15];

memset(edge,0,sizeof(edge));

memset(reach,10005,sizeof(reach));

memset(movement,-1,sizeof(movement));

v.clear();

priority\_queue <node> q; //工作队列

while(!q.empty())q.pop();

reach[X][Y]=0;

movement[X][Y]=-1;

edge[X][Y]=0;

q.push(node(X,Y,0));

while(!q.empty()){

node tmp=q.top();

q.pop();

if(land[tmp.x][tmp.y]==2)continue; //是终点

for(int turn=0;turn<8;turn++){

int tx=tmp.x+dx[turn];

int ty=tmp.y+dy[turn];

// if(sta[tx][ty]==-1)continue; //看不见不管

if( land[tx][ty]==1 )continue;

int tt=tmp.t+1;

if(reach[tx][ty]>tt){

reach[tx][ty]=tt;

q.push(node(tx,ty,tt));

movement[tx][ty]=turn;

}

if(edge[tx][ty]==0 && land[tx][ty]!=1 && sta[tx][ty]==0 ){

if( land[tx][ty]==2 || check\_edge(tx,ty) ){ //进入 边缘向量组，之后要检测

//movement[tx][ty]=turn;

v.push\_back(node(tx,ty,tt));

edge[tx][ty]=1;

}

//加入edge

}

//完成周围点检测

}

}

//基本检测完成

// printf("size: %d\n",v.size());

// check\_eyesight\_01();

/\*

printf("edge\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(edge[i][j])printf("1\t");

else printf("0\t");

}

printf("\n");

}

/\*

printf("movement:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",movement[i][j]);

}

printf("\n");

}

if(sta[1][B]!=-1){ //能看见终点，则必往终点跑

v.push\_back(node(1,B,0));

edge[1][B]=1;

}

printf("reach:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

if(sta[i][j]==-1){

printf("?\t");

continue;

}

if(edge[i][j])printf("\*");

if(reach[i][j]>NANT)printf("%d\t",-1);

else printf("%d\t",reach[i][j]);

}

printf("\n");

}

\*/

//找到代价+成本最小的

//找到代价+成本最小的

double judge[15][15];

memset(judge,0,sizeof(judge));

int minx,miny;

double minjudge=10005;

for(int i=0;i<v.size();i++){

int tx=v[i].x;

int ty=v[i].y;

judge[tx][ty]=reach[tx][ty]+K\*minf[tx][ty];

if(judge[tx][ty]<minjudge){

minjudge=judge[tx][ty];

minx=tx;

miny=ty;

}

}

//看到终点：直冲终点

if(edge[1][B]==1){

minjudge=0;

minx=1;

miny=B;

}

/\*

printf("JUDGE:\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",judge[i][j]);

}

printf("\n");

}

\*/

// cout<<"min judge : "<<minx<<" "<<miny<<endl;

//minx,miny 是当前的目标点

//回溯目标点

int targetx=minx,targety=miny,lastx,lasty;

int backx,backy;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

while(lastx!=X || lasty!=Y){

targetx=lastx;

targety=lasty;

backx=dx[ movement[targetx][targety] ];

backy=dy[ movement[targetx][targety] ];

lastx=targetx-backx;

lasty=targety-backy;

}

// cout<<targetx<<" "<<targety<<endl;

//此时 target[x y]就是下一步的目标

return node(targetx,targety,0);

}

void land\_reshape(){

for(int i=0;i<=9;i++){

for(int j=0;j<=9;j++){

if(mas[i\*10][j\*10]<Srate)land[i+1][j+1]=0;

else land[i+1][j+1]=-1;

}

}

land[A][1]=3;

land[1][B]=2;

}

int main(){

A=10;

B=10;

Srate=50;

freopen("newscan\_in.txt","r",stdin);

init();

N=1; //图的个数

land\_reshape(); //把land信息转化成10\*10

double Kseq[25]={10,5,3,2.5,2, 1.8,1.6,1.4,1.2,1.1 ,1,0.9,0.8,0.7,0.6 ,0.5,0.4,0.3,0.2,0.1};

//信息素参数

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

minf[i][j]=max( i-1 , B-j );

}

}

//land checked

/\*

printf("land\n");

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

printf("%d\t",land[i][j]);

}

printf("\n");

}

//单点测试区

X=10,Y=1;

SCAN();

check\_eyesight(); //输出整个阵

step();

//主程序程序

\*/

K=1.2;

X=10,Y=1;

int cnt=0;

printf("time: %d \t\t [%d][%d]\n",cnt++,A,1);

while( X!=1 || Y!=B ){ //这里假设图一定可以走通

//扫描可见域

SCAN();

check\_eyesight();

printf("\n");

check\_eyesight\_01();

node NEXT=step();

X=NEXT.x;

Y=NEXT.y;

printf("time: %d \t\t [%d][%d]\n",cnt++,X,Y);

printf("\n");

}

//printf("time: %d\t [%d][%d]\n",cnt++,1,B);

return 0;

}

附录十 Ktest.cpp

说明：Ktest 是用于计算最优K取值的程序

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF=0x3f3f3f3f;

const long long mod=1000000007;

const double e=2.718281828459045;

const double pi=3.1415926535;

#define CK cout<<"OK\n";

double a[1050];

int N;

double k[25];

int nk;

double kans[25][1050];

int main(){

freopen("ktest.txt","r",stdin);

freopen("ktest\_answer.txt","w",stdout);

N=1000;

nk=20;

double minsum=10;

int mink=0;

for(int i=0;i<N;i++){

scanf("%lf",&a[i]);

}

for(int j=0;j<20;j++){

double sum=0;

scanf("%lf",&k[j]);

for(int i=0;i<N;i++){

scanf("%lf",&kans[j][i]);

sum+=kans[j][i]/a[i];

}

sum/=1000;

if(sum<minsum){

minsum=sum;

mink=j;

}

printf("K= %.2lf\t\tave=%.4lf\n",k[j],sum);

}

printf("\nmin: k=%.2lf\t\tave=%.4lf\n",k[mink],minsum);

return 0;

}