# A\_Star

## 一 程序代码

整个程序建立了三个文件：mapping.h，mapping.hpp，main.cpp．整个Ａ＊算法封装在一个类中class mapping1，类的声明mapping.h中，类的实现在mapping.hpp．main.cpp中只是用mapping1定义对象，调用建图成员函数和Ａ＊算法函数，找到最短路径，并用opencv打印出来路径图．在main.cpp中写了一个鼠标事件回调函数，这个函数可以实现，左键按下鼠标对应位置作为新的终止位置，右键按下鼠标对应位置作为新的起始点位置，滚轮按下会恢复到最原始的起始点和终止点．

### 1.1 mapping.h 头文件

#ifndef MAPPING\_H

#define MAPPING\_H

#include <string>

#include <opencv2/opencv.hpp>

#include <iostream>

#include<stdio.h>

#include<malloc.h>

using namespace cv;

using namespace std;

/\* 一些预定义 \*/

#define MaxVerNum 729 /\*定义最大节点数\*/

int flag[729];

Mat img(580, 980, CV\_8UC3);//读入图像

/\* 建图需要的结构体 \*/

struct node

{

int adjvex ;//节点号

struct node \*next ; //指向下一个邻接节点域

};

struct vnode//链表头节点

{

int vertex; //放flag

node \*firstedge ; //边表头指针

};

struct AGraph

{

vnode adjList[729] ; //邻接表

int n; //顶点数

int e=0; //边数

};

struct a\_node1

{

int num;

int set;//0 open 1 close 2 unkown

int g;

int h;

int f;

int parent;

};

/\* ．．．．．．． 地图显示 ．．．．．．．．\*/

class mapping1

{

private:

AGraph \*GG ;

a\_node1 aa\_node1[729];

Point root\_points[729][4];

Point root\_point[4];

vector<int> g\_path;

public:

mapping1()

{

start\_num=702;

stop\_num=26;

g\_path.clear();

}

int start\_num,stop\_num;

void mapping1\_init();//无参构造函数

void map\_show();

void CreateALGraph();

void a\_star1();

};

#endif // MAPPING\_H

### 1.2 mapping.hpp

#include <string>

#include <opencv2/opencv.hpp>

#include <iostream>

#include<stdio.h>

#include<malloc.h>

#include <mapping.h>

#include <vector>

using namespace cv;

using namespace std;

mapping1 ma;

/\* ．．．．．．． 地图显示 ．．．．．．．．\*/

void mapping1::mapping1\_init()//无参构造函数

{

//start\_num=702;

//flag[]={0};

for(int i=0;i<729;i++)

{

flag[i]=0;

}

for(int i=0;i<10;i++)

{

flag[10+i\*27]=1;

}

for(int i=0;i<3;i++)

{

flag[254+i]=1;

}

for(int i=0;i<5;i++)

{

flag[257+i-54]=1;

flag[257+i-27]=1;

flag[257+i]=1;

flag[257+i+27]=1;

}

for(int i=0;i<4;i++)

{

flag[73+i]=1;

flag[73+i+27]=1;

flag[73+i+54]=1;

}

for(int i=0;i<6;i++)

{

flag[185+i\*27]=1;

}

for(int i=0;i<15;i++)

{

flag[363+i]=1;

}

for(int i=0;i<7;i++)

{

flag[328+i]=1;

}

for(int i=0;i<5;i++)

{

flag[334+(i+1)\*27]=1;

}

for(int i=0;i<7;i++)

{

flag[493+i]=1;

flag[493+i+27]=1;

flag[493+i+54]=1;

}

for(int i=0;i<8;i++)

{

flag[621+i]=1;

}

for(int i=0;i<4;i++)

{

flag[606+i]=1;

flag[606+i+27]=1;

flag[606+i+54]=1;

flag[606+i+81]=1;

flag[606+i+108]=1;

}

flag[stop\_num]=3;//目标点,

flag[start\_num]=4;//起始点

}

void mapping1::map\_show()//地图显示

{

for(int i=0;i<729;i++)

{

root\_points[i][0] = Point((i%27+1)\*20, (i/27+1)\*20);

root\_points[i][1] = Point((i%27+1)\*20+20, (i/27+1)\*20);

root\_points[i][2] = Point((i%27+1)\*20+20, (i/27+1)\*20+20);

root\_points[i][3] = Point((i%27+1)\*20, (i/27+1)\*20+20);

const Point\* ppt[1] = { root\_points[i] };

int npt[] = { 4 };

if(1==flag[i])

{

fillPoly(img, ppt, npt, 1, Scalar(0,0,0));

}

else if(0==flag[i])

{

fillPoly(img, ppt, npt, 1, Scalar(255,255,255));

}

else if(3==flag[i])

{

fillPoly(img, ppt, npt, 1, Scalar(0,0,255));

}

else if(4==flag[i])

{

fillPoly(img, ppt, npt, 1, Scalar(255,0,0));

}

else if(7==flag[i])

{

string tmpstring=to\_string(aa\_node1[i].g);

fillPoly(img, ppt, npt, 1, Scalar(100,205,255));

putText(img,tmpstring,Point((i%27+1)\*20+3, (i/27+1)\*20+16),FONT\_HERSHEY\_SIMPLEX,0.4,Scalar(255,23,0),1,8);

}

else

{

string tmpstring=to\_string(aa\_node1[i].g);

fillPoly(img, ppt, npt, 1, Scalar(0,255,255));

putText(img,tmpstring,Point((i%27+1)\*20+3, (i/27+1)\*20+16),FONT\_HERSHEY\_SIMPLEX,0.4,Scalar(255,23,0),1,8);

}

}

for(int i=0;i<28;i++)

{ line(img,Point((i+1)\*20,20),Point((i+1)\*20,560),Scalar(0,0,255),1,CV\_AA);

line(img,Point(20,(i+1)\*20),Point(560,(i+1)\*20),Scalar(0,0,255),1,CV\_AA);

}

root\_point[0] = Point(600, 20);

root\_point[1] = Point(960,20);

root\_point[2] = Point(960,560);

root\_point[3] = Point(600,560);

const Point\* ppt[1] = { root\_point };

int npt[] = { 4 };

fillPoly(img, ppt, npt, 1, Scalar(255,255,255));

string tmpstring="A\_Star Algorithm";

putText(img,tmpstring,Point(620,50),FONT\_HERSHEY\_SIMPLEX,1,Scalar(0,0,255),2,8);

if(aa\_node1[stop\_num].set==1)

{

tmpstring="Minimum number of steps:"+to\_string(aa\_node1[stop\_num].g);

putText(img,tmpstring,Point(620,70),FONT\_HERSHEY\_SIMPLEX,0.5,Scalar(0,0,255),1,8);

tmpstring="The shortest path:";

putText(img,tmpstring,Point(620,100),FONT\_HERSHEY\_SIMPLEX,0.5,Scalar(0,0,255),1,8);

int m\_size=g\_path.size();

int m\_h;

if(0==m\_size%5)

{

m\_h=m\_size/5;

}

else

{

m\_h=m\_size/5+1;

}

for(int i=0;i<m\_h;i++)

{

tmpstring.clear();

for(int j=0;j<5;j++)

{

if(g\_path.back()>=100)

tmpstring=tmpstring+to\_string(g\_path.back());

else if(g\_path.back()>=10)

tmpstring=tmpstring+to\_string(0)+to\_string(g\_path.back());

else

tmpstring=tmpstring+to\_string(0)+to\_string(0)+to\_string(g\_path.back());

g\_path.pop\_back();

if(g\_path.empty())

break;

else

tmpstring=tmpstring+"->";

}

putText(img,tmpstring,Point(620,120+20\*i),FONT\_HERSHEY\_SIMPLEX,0.5,Scalar(0,0,255),1,8);

}

}

}

void mapping1::CreateALGraph()//建图

{

this->GG = (AGraph\*)malloc(sizeof(AGraph)) ;

int i ;

node \*s ;

this->GG->n=729;

for( i = 0 ; i <this->GG->n ; i++)

{

this->GG->adjList[i].vertex=flag[i];//flag

this->GG->adjList[i].firstedge = NULL ; //将顶点的边表头指针设置为空

/\* 判断边的信息 \*/

if((flag[i]!=1)&&(i%27!=0)&&(flag[i-1]!=1))//右边的边

{

this->GG->e++;

s = (node\*)malloc(sizeof(node)) ;

s->adjvex = i-1 ;

s->next = this->GG->adjList[i].firstedge ;

this->GG->adjList[i].firstedge = s ;

}

if((flag[i]!=1)&&((i+1)%27!=0)&&(flag[i+1]!=1))//右边的边

{

this->GG->e++;

s = (node\*)malloc(sizeof(node)) ;

//边上的第一个节点

s->adjvex = i+1 ;

s->next = this->GG->adjList[i].firstedge ;

this->GG->adjList[i].firstedge = s ;

}

if((flag[i]!=1)&&(i-27>0)&&(flag[i-27]!=1))//下面的边

{

this->GG->e++;

s = (node\*)malloc(sizeof(node)) ;

//边上的第一个节点

s->adjvex = i;

s->next = this->GG->adjList[i-27].firstedge ;

this->GG->adjList[i-27].firstedge = s ;

//边上的第二个节点

s = (node\*)malloc(sizeof(node)) ;

s->adjvex = i-27;

s->next = this->GG->adjList[i].firstedge ;

this->GG->adjList[i].firstedge = s ;

}

if((flag[i]!=1)&&(i+27<729)&&(flag[i+27]!=1))//下面的边

{

this->GG->e++;

s = (node\*)malloc(sizeof(node)) ;

s->adjvex = i;

s->next = this->GG->adjList[i+27].firstedge ;

this->GG->adjList[i+27].firstedge = s ;

}

}

}

void mapping1::a\_star1()//A\*算法

{

int tmp\_num,tmp\_f;

for(int i=0;i<729;i++)

{

aa\_node1[i].num=i;

aa\_node1[i].set=2;

aa\_node1[i].g=999;

if(i==start\_num)

{

aa\_node1[i].set=0;//把起始点放在开集里

aa\_node1[i].g=0;

}

aa\_node1[i].h=abs(i/27-stop\_num/27)+abs(i%27-stop\_num%27);

aa\_node1[i].f=aa\_node1[i].g+aa\_node1[i].h;

aa\_node1[i].parent=-1;

}

while(aa\_node1[stop\_num].set!=1)

{

/\* 2.1-----3 \*/

tmp\_num=-1;

tmp\_f=1599;

for(int i=0;i<729;i++)//在开集中找最小的ｆ

{

if(0==aa\_node1[i].set)

{

if(tmp\_f>=aa\_node1[i].f)

{

if(tmp\_f==aa\_node1[i].f)

{

if(aa\_node1[i].h<aa\_node1[tmp\_num].h)

{

tmp\_num=i;

tmp\_f=aa\_node1[i].f;

continue;

}

else

{

continue;

}

}

tmp\_num=i;

tmp\_f=aa\_node1[i].f;

}

}

}

aa\_node1[tmp\_num].set=1;//把最小的ｆ的点放在闭集里面

if((tmp\_num!=start\_num)&&tmp\_num!=stop\_num)

flag[tmp\_num]=7;

this->map\_show();

imshow("image", img);

waitKey(3);

node \*stmp;

stmp = (node\*)malloc(sizeof(node)) ;

stmp= GG->adjList[aa\_node1[tmp\_num].num].firstedge ;

int tmp;

while(stmp!=NULL)

{

tmp=stmp->adjvex;

if((aa\_node1[tmp\_num].g+1)<aa\_node1[tmp].g)

{

aa\_node1[tmp].g=aa\_node1[tmp\_num].g+1;

aa\_node1[tmp].parent=tmp\_num;

}

aa\_node1[tmp].f=aa\_node1[tmp].g+aa\_node1[tmp].h;//更新ｆ

if(2==aa\_node1[tmp].set)

aa\_node1[tmp].set=0;

stmp=stmp->next;

}

//cout<<tmp\_num<<" ";

}

int tmp\_n=aa\_node1[stop\_num].parent;

g\_path.push\_back(stop\_num);

//path

while(tmp\_n!=start\_num)

{

g\_path.push\_back(tmp\_n);

flag[tmp\_n]=2;

tmp\_n=aa\_node1[tmp\_n].parent;

}

g\_path.push\_back(start\_num);

}

### 1.3 main.cpp

#include <string>

#include <opencv2/opencv.hpp>

#include <iostream>

#include<stdio.h>

#include<malloc.h>

#include <mapping.hpp>

using namespace cv;

using namespace std;

extern mapping1 ma;

void on\_mouse(int event,int x,int y,int flags,void \*ustc)//event鼠标事件代号，x,y鼠标坐标，flags拖拽和键盘操作的代号

{

if (event == CV\_EVENT\_LBUTTONDOWN)//左键按下，读取初始坐标，并在图像上该点处划圆

{

if(x>20&&x<560&&y>20&&y<560)

{

int tmp=(y/20-1)\*27+x/20-1;

if(flag[tmp]!=1)

{

ma.stop\_num=tmp;

ma.mapping1\_init();

ma.a\_star1();

ma.map\_show();

}

}

}

if (event == CV\_EVENT\_RBUTTONDOWN)//左键按下，读取初始坐标，并在图像上该点处划圆

{

if(x>20&&x<560&&y>20&&y<560)

{

int tmp=(y/20-1)\*27+x/20-1;

if(flag[tmp]!=1)

{

ma.start\_num=tmp;

ma.mapping1\_init();

ma.a\_star1();

ma.map\_show();

}

}

}

if (event == CV\_EVENT\_MBUTTONDOWN)//左键按下，读取初始坐标，并在图像上该点处划圆

{

if(x>20&&x<560&&y>20&&y<560)

{

ma.start\_num=702;

ma.stop\_num=26;

ma.mapping1\_init();

ma.a\_star1();

ma.map\_show();

}

}

imshow("image", img);

}

int main()

{

ma.mapping1\_init();

ma.CreateALGraph() ;

namedWindow("image",1);

ma.a\_star1();

ma.map\_show();

setMouseCallback("image",on\_mouse,0);//调用回调函数

imshow("image", img);

waitKey(0);

return 0;

}

二 运行效果







