实验题目：A\*算法

实验目的：掌握A\*算法的原理，实现A\*算法

源代码;

**import** sys  
*#将地图中的点抽象化成类***class** Point:  
 **def** \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
  
 **def** \_\_eq\_\_(self, other): *#函数重载* **if**((self.x == other.x )**and** (self.y == other.y)):  
 **return** 1  
 **else**:  
 **return** 0  
  
*#通过列表实现的地图的建立 类c语言数组？***class** map\_2d:  
 **def** \_\_init\_\_(self,height,width):  
 self.height = height  
 self.width = width  
 self.data = []  
 self.data = [[0 **for** i **in** range(width)] **for** j **in** range(height)]  
 **def** map\_show(self):  
 **for** i **in** range(self.height):  
 **for** j **in** range(self.width):  
 print(self.data[i][j], end=**' '**)  
 print(**""**)  
 **def** obstacle(self,obstacle\_x,obstacle\_y):  
 self.data[obstacle\_x][obstacle\_y]=1  
 **def** end\_draw(self,point):  
 self.data[point.x][point.y] = 6  
  
*#A\*算法的实现***class** A\_star:  
 *# 设置node* **class** Node:  
 **def** \_\_init\_\_(self, point, endpoint, g):  
 self.point = point *# 自己的坐标* self.endpoint = endpoint *# 自己的坐标* self.father = **None** *# 父节点* self.g = g *# g值，g值在用到的时候会重新算* self.h = (abs(endpoint.x - point.x) + abs(endpoint.y - point.y)) \* 10 *# 计算h值* self.f = self.g + self.h  
  
 *#寻找临近点* **def** search\_near(self,ud,rl): *# up down right left* nearpoint = Point(self.point.x + rl, self.point.y + ud)  
 nearnode = A\_star.Node(nearpoint, self.endpoint, self.g + 1)  
 **return** nearnode  
  
  
 **def** \_\_init\_\_(self,start\_point,end\_point,map):*#需要传输到类中的，在此括号中写出* self.path=[]  
 self.close\_list=[] *#存放已经走过的点* self.open\_list=[] *#存放需要尽心探索的点* self.current = 0 *#现在的node* self.start\_point=start\_point  
 self.end\_point=end\_point  
 self.map = map *#所在地图* **def** select\_current(self):  
 min=10000000  
 node\_temp = 0  
 **for** ele **in** self.open\_list:  
 **if** ele.f < min:  
 min = ele.f  
 node\_temp = ele  
 self.path.append(node\_temp)  
 self.open\_list.remove(node\_temp)  
 self.close\_list.append(node\_temp)  
 **return** node\_temp  
  
 **def** isin\_openlist(self,node):  
 **for** opennode\_temp **in** self.open\_list:  
 **if** opennode\_temp.point == node.point:  
 **return** opennode\_temp  
 **return** 0  
  
 **def** isin\_closelist(self,node):  
 **for** closenode\_temp **in** self.close\_list:  
 **if** closenode\_temp.point == node.point:  
 **return** 1  
 **return** 0  
  
 **def** is\_obstacle(self,node):  
 **if** self.map.data[node.point.x][node.point.y]==1 :  
 **return** 1  
 **return** 0  
  
 **def** near\_explore(self,node):  
 ud = 1  
 rl = 0  
 node\_temp = node.search\_near(ud,rl) *#在调用另一个类的方法时（不论是子类还是在类外定义的类），都要进行实例化才能调用函数* **if** node\_temp.point == end\_point:  
 **return** 1  
 **elif** self.isin\_closelist(node\_temp):  
 **pass  
 elif** self.is\_obstacle(node\_temp):  
 **pass  
 elif** self.isin\_openlist(node\_temp) == 0:  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
 **else**:  
 **if** node\_temp.f < (self.isin\_openlist(node\_temp)).f:  
 self.open\_list.remove(self.isin\_openlist(node\_temp))  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
  
 ud = -1  
 rl = 0  
 node\_temp = node.search\_near(ud,rl) *#在调用另一个类的方法时（不论是子类还是在类外定义的类），都要进行实例化才能调用函数* **if** node\_temp.point == end\_point:  
 **return** 1  
 **elif** self.isin\_closelist(node\_temp):  
 **pass  
 elif** self.is\_obstacle(node\_temp):  
 **pass  
 elif** self.isin\_openlist(node\_temp) == 0:  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
 **else**:  
 **if** node\_temp.f < (self.isin\_openlist(node\_temp)).f:  
 self.open\_list.remove(self.isin\_openlist(node\_temp))  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
  
 ud = 0  
 rl = 1  
 node\_temp = node.search\_near(ud,rl) *#在调用另一个类的方法时（不论是子类还是在类外定义的类），都要进行实例化才能调用函数* **if** node\_temp.point == end\_point:  
 **return** 1  
 **elif** self.isin\_closelist(node\_temp):  
 **pass  
 elif** self.is\_obstacle(node\_temp):  
 **pass  
 elif** self.isin\_openlist(node\_temp) == 0:  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
 **else**:  
 **if** node\_temp.f < (self.isin\_openlist(node\_temp)).f:  
 self.open\_list.remove(self.isin\_openlist(node\_temp))  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
  
 ud = 0  
 rl = -1  
 node\_temp = node.search\_near(ud,rl) *#在调用另一个类的方法时（不论是子类还是在类外定义的类），都要进行实例化才能调用函数* **if** node\_temp.point == end\_point:  
 **return** 1  
 **elif** self.isin\_closelist(node\_temp):  
 **pass  
 elif** self.is\_obstacle(node\_temp):  
 **pass  
 elif** self.isin\_openlist(node\_temp) == 0:  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
 **else**:  
 **if** node\_temp.f < (self.isin\_openlist(node\_temp)).f:  
 self.open\_list.remove(self.isin\_openlist(node\_temp))  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
  
 ud = 1  
 rl = 1  
 node\_temp = node.search\_near(ud,rl) *#在调用另一个类的方法时（不论是子类还是在类外定义的类），都要进行实例化才能调用函数* **if** node\_temp.point == end\_point:  
 **return** 1  
 **elif** self.isin\_closelist(node\_temp):  
 **pass  
 elif** self.is\_obstacle(node\_temp):  
 **pass  
 elif** self.isin\_openlist(node\_temp) == 0:  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
 **else**:  
 **if** node\_temp.f < (self.isin\_openlist(node\_temp)).f:  
 self.open\_list.remove(self.isin\_openlist(node\_temp))  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
  
 ud = 1  
 rl = -1  
 node\_temp = node.search\_near(ud,rl) *#在调用另一个类的方法时（不论是子类还是在类外定义的类），都要进行实例化才能调用函数* **if** node\_temp.point == end\_point:  
 **return** 1  
 **elif** self.isin\_closelist(node\_temp):  
 **pass  
 elif** self.is\_obstacle(node\_temp):  
 **pass  
 elif** self.isin\_openlist(node\_temp) == 0:  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
 **else**:  
 **if** node\_temp.f < (self.isin\_openlist(node\_temp)).f:  
 self.open\_list.remove(self.isin\_openlist(node\_temp))  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
  
 ud = -1  
 rl = 1  
 node\_temp = node.search\_near(ud,rl) *#在调用另一个类的方法时（不论是子类还是在类外定义的类），都要进行实例化才能调用函数* **if** node\_temp.point == end\_point:  
 **return** 1  
 **elif** self.isin\_closelist(node\_temp):  
 **pass  
 elif** self.is\_obstacle(node\_temp):  
 **pass  
 elif** self.isin\_openlist(node\_temp) == 0:  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
 **else**:  
 **if** node\_temp.f < (self.isin\_openlist(node\_temp)).f:  
 self.open\_list.remove(self.isin\_openlist(node\_temp))  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
  
 ud = -1  
 rl = -1  
 node\_temp = node.search\_near(ud,rl) *#在调用另一个类的方法时（不论是子类还是在类外定义的类），都要进行实例化才能调用函数* **if** node\_temp.point == end\_point:  
 **return** 1  
 **elif** self.isin\_closelist(node\_temp):  
 **pass  
 elif** self.is\_obstacle(node\_temp):  
 **pass  
 elif** self.isin\_openlist(node\_temp) == 0:  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
 **else**:  
 **if** node\_temp.f < (self.isin\_openlist(node\_temp)).f:  
 self.open\_list.remove(self.isin\_openlist(node\_temp))  
 node\_temp.father = node  
 self.open\_list.append(node\_temp)  
  
 **return** 0  
  
*##建图并设立障碍*ss=map\_2d(10,20)  
**for** i **in** range(10):  
 ss.obstacle(4,i)  
**for** i **in** range(19):  
 ss.obstacle(0,i+1)  
**for** i **in** range(9):  
 ss.obstacle(i+1,0)  
**for** i **in** range(9):  
 ss.obstacle(i+1,19)  
**for** i **in** range(19):  
 ss.obstacle(9,i)  
ss.obstacle(8,6)  
ss.obstacle(6,8)  
ss.obstacle(6,15)  
ss.obstacle(9,10)  
start\_point = Point(1,2)  
end\_point = Point(9,19)  
ss.end\_draw(end\_point)  
ss.end\_draw(start\_point)  
  
*#初始化设置A\**a\_star = A\_star(start\_point,end\_point,ss)  
start\_node = a\_star.Node(start\_point,end\_point,0)  
a\_star.open\_list.append(start\_node)  
  
flag=0 *#到达终点的标志位*m=0 *#步数统计  
  
#进入循环***while** flag != 1 :  
 a\_star.current = a\_star.select\_current()*#从openlist中选取一个node* flag=a\_star.near\_explore(a\_star.current)*#对选中的node进行周边探索* m=m+1  
 print(m)  
  
*#画出地图路径***for** node\_path **in** a\_star.path:  
 ss.end\_draw(node\_path.point)  
ss.map\_show()

