基于视觉仿真扫描和蚁群算法的机器人寻路问题

摘要

机器人自动避障是机器人行动路径规划中的基本问题。本文根据题中给出的地图情况，针对不同的障碍物情况，设计出了两套扫描方法，并从扫描获得的信息量出发，建立数学模型，并利用蚁群算法对机器人的行动进行动态决策，指导机器人沿最优路线行进。

针对全图障碍信息已知的情况，本文采用A^*算法求最优路径。对于地图 1 中障碍已经栅格化的情况，用优先队列维护A^*算法，结果显示须走 12 步。对于地图 2，首先将矢量图栅格化为位图，再利用 Python 中的 PIL.image 函数库提取图片中的像素，统计每一格中属于障碍物的像素，将位图进一步转话为 10×10 格的地图数据，之后的操作同地图 1 一致，结果显示须走 11 步。

针对全图障碍信息未知的情况，本文把机器人的命令系统分为扫描、决策、行动三部，执行动态局部贪心算法。在扫描命令系统中，对于地图 3 中障碍已经栅格化的情况，本文采用模拟人类视觉扫描法获取可见域：通过比较目标方格和当前视野遮挡物相对于机器人所在方格的极角，来确定目标方格的可见性。对于地图 4 中障碍未栅格化的情况，本文采用辐射-信息量扫描法获取可见域：检测目标方格和机器人所在方格连线上的障碍情况以获得信息的损失率，以此确定目标方格的可见性。本文利用蚁群算法决策，在可见域的边缘寻找“食物”和“信息素”，再利用洪泛法找出前往“食物源”或“信息素”浓度最高的地点的路径，并以此路径为基础，决定机器人当前的行动。反复执行扫描，决策，行动三步，直到机器人到达终点或发现永远也到达不了终点为止。结果显示，对于地图 3 和地图 4，机器人分别须行进 10 步和 11 步。

为了检验方法的普适性，本文用大量地图对程序进行了测试，并以此对信息素的决定式进行了调整。统计结果显示，在障碍密度为 0.4，公式$T_{ij} = x + k \times y$中的估价因子 k 取 1.8 的情况下，相对于全图障碍信息已知时机器人的行动步数，本算法所需的步数平均多出 12.63%。通过改变障碍密度，对 k 进行灵敏度分析，可发现通常情况下 k 取 1.2-1.8 可使机器人寻路效率的到最高。

关键词： A^*算法 蚁群算法 动态局部最优 模拟人类视觉扫描法

一、问题重述

1.1 问题背景

随着技术的发展，机器人替代了很多原本需要人力操作的事情，这大大节约了人力成本。由于机器人的迅速发展，路径规划问题也变成如今的热门问题。进行路径规划是一项在一定的限制下，使机器人合理地规避空间中出现的障碍遮挡物，并且顺利从起点到达终点的工作。

对机器人所处的空间栅格化，机器人每次只能够行进到相邻的栅格，允许对角线间的行走。在栅格空间中置放了很多障碍物，机器人需要规避开障碍物行进到规定的终点。

1.2 问题提出

问题一：给出栅格空间中明确的出发点(S)和终点(D)，并在栅格中放置黑色的障碍物，这些障碍物的具体位置机器人在出发之前已经知晓，即机器人在出发前知道栅格地图的全貌信息。**问题一**中分两种情况，第一种情况是障碍物是规则的，即障碍物占据栅格的面积为 0%或 100%，机器人需要经过合理的路径规划，避开障碍物以最优的行进路线从起点到终点；第二种情况是障碍物是不规则的，即障碍物随机分布在栅格地图中，可能呈椭圆，也可能呈三角形，甚至可能是完全不规则的图形，即障碍物占据栅格的面积是 0%至 100%中的任意数。当某个栅格中障碍物所占据的面积大于 50%时，机器人无法行进至改栅格，反之，机器人可以行进至该栅格。机器人需要通过合理的算法，实现从起点到终点的最优行进路线的规划。

问题二：跟**问题一**类似，同样给出了栅格空间内的出发点(S)和终点(D)，还有黑色的障碍物，但这些障碍物的具体位置机器人在出发前并不知晓，即机器人在出发前不知晓栅格地图的全貌，和问题一的区别在于机器人知道的信息是有限的，它并不了解障碍遮挡物背后的情况。**问题二**也分两种情况，其一是规则的障碍物，机器人需要通过合理的算法获取目前的视野区域，建立数学模型，找到机器人最优的行进路线；其二是不规则的障碍物，当障碍物占据某一栅格的面积不到 50%，则机器人可以行进到该栅格，反之不行。题目要求我们需要建立合适的数学模型，找到这种情况下最好的行进路线。

二、问题分析

2.1 问题一的分析

问题一需要设计一个机器人的最优行进路线算法。在问题一的背景下，机器人在出发前已知晓栅格地图的全貌条件，我们先考虑问题一中第一种规则障碍物的情况。在随机生成的规则障碍物栅格地图下，我们通过机器人从起点到终点所需的步数来衡量算法的好坏，即机器人从起始点到终点所需的步数越少，算法越好。因此我们建立了最优行进路线模型。考虑到机器人在相邻栅格间移动和对角栅格间移动所需要的步数相同，因此我们认为它每移动一个栅格所需要的时间相同。我们的算法记录了机器人从起点出发到每一个栅格所需要的最短时间，并得到最优的行进路线。问题一的第二种不规则障碍物的情况，我们考虑到机器人在出发前知晓栅格地图，因此我们用 Python 获取地图的每一个像素点，然后判断每一个栅格中障碍物所占像素点的百分比，即障碍物的面积百分比，将此数值与 50%判断后，获得此栅格是否允许机器人通过的信息，对这些计算好比例的栅格进行归一化处理，将不规则障碍物的栅格图转化为规则障碍物的栅格图，并将这幅图的信息告知机器人，机器人便可以直接运用第一种情况的最优行进路线模型算法对每个栅格的最短时间进行计算，并找出从起点到终点的最优行进路线。

2.2 问题二的分析

问题二考虑的是移动机器人在出发前并不知晓全部栅格地图信息的背景条件下，如

何通过建立数学模型来计算得到移动机器人从起点到终点的最优行进路线。在问题二中对于行进路线的最优解是机器人在成功规避障碍物的同时能够尽量使用较少的步数行进到终点。因为问题二中移动机器人在出发之前并不知道整个栅格地图中障碍物的分布情况。因此，此模型需要先通过眼前的障碍物，获取机器人此时的视野区域，再运用科学合理的算法，得出机器人此时的最优路径。问题二中也存在两种情况，规则与不规则障碍物唯一存在区别的地方是在于视野区域范围的扫描模型的建立。解决问题的核心包括两个方面，一是找到合适的扫描方法，确定机器人视野；二是找出一套最高效的寻路机制。

三、模型假设

1. 假设机器人行进一格所需要的时间是一样（可认为机器人是跳跃行进，即机器人行进至相邻栅格的代价相同）。
2. 假设机器人只能行走在大小有限的栅格地图中。
3. 假设机器人的视线在无障碍物阻拦下可看至无穷远。

四、符号说明

符号	符号意义
T	表示移动机器人沿任意路径从起点到终点所需要的时间
$A_{i,j}$	表示坐标为(i,j)的栅格的成本
$B_{i,j}$	表示栅格所接受到的所有成本$A_{i,j}$的最小值
S	表示此栅格的总像素点个数
s_i	表示某个栅格内障碍物所包含的像素点个数
$C_{i,j}$	表示每个栅格的状态
$f(n)$	表示从起点（S）到终点（D）的估计代价
$g(n)$	表示从起点到状态 n 的成本
$h(n)$	表示从状态 n 到终点（D）的估计代价
k	表示蚁群算法中决定信息素含量的因子
$S_{参考}$	表示障碍物栅格的默认通过率
$S_{i,j}$	表示栅格的通过率
$S_{非障碍物}$	表示栅格中非障碍物所占面积
$S_{总}$	表示栅格的总面积
T_{ij}	表示从移动机器人当前所处的栅格到终点的任意路径的成本
θ_n	表示任意两条射线之间的夹角
$L\ \&\ W$	表示整个地图的长与宽

注：未列出符号及重复的符号以出现处为准

五、模型的建立与求解

5.1 问题一的模型建立与求解

5.1.1 规则障碍物情况

问题一要求我们给出一般情况下机器人的行进路线模型与相应的求解算法。一般情况是指建立的行进路线模型可以适用于 10×10 的栅格内规则障碍物任意摆放的情况，即建立的模型不是仅仅满足于个别几种情况下的规则障碍物模型，而是能适应所有情况的泛化模型。因此模型的建立不能基于任何的特殊值，不能依赖任何的特殊参照物。同时，对应的模型求解算法也不能建立在某些特殊情况下，应该要确保在随机算法任意生成规则障碍物下，都能给出行进路线最优化的解。

5.1.1.1 最优行进路线模型的工作原理

在问题一背景下，机器人事先知晓栅格地图的所有信息，因此移动机器人在行动之前已经可以通过模型的算法获取最优路径。

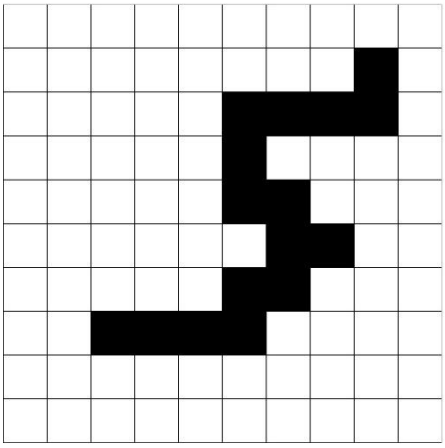


图 1 题中添加障碍物的栅格地图

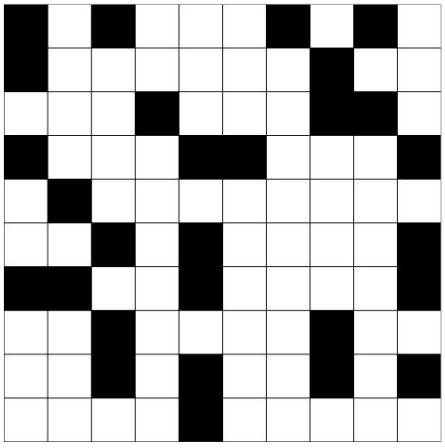


图 2 随机添加障碍物的栅格地图

机器人在栅格地图中移动，对应的会产生当前所在的栅格相对于起点(S)和终点(D)的成本与代价，机器人当前所处的位置相对于起点（S）的成本就是机器人从起点（S）到达目前位置所走过的步数，由某一个栅格到达目标格的步数称之为代价。最优行进路线模型的根本工作原理就是利用广度优先搜索算法找出终点的成本最小值。

5.1.1.2 最优行进路线的模型

1)目标函数的确定

对于移动机器人最优行进线路而言，最优的路径一定是所需步数最少的路径模型[1]。在移动步数与时间关系的假设下，我们可以知道最优路径是机器人在规则障碍物的干扰下，从起点到终点所需要的最短时间。因此我们建立目标函数：

$$\min T \#(1)$$

T 表示移动机器人沿任意路径从起点到终点所需要的时间。

2)方案设定

Step1. 首先为了考虑移动机器人行进路径，将整个栅格地图看作一个直角坐标系，其中起点坐标为 $(1,1)$，终点坐标为$(10,10)$。我们设$A_{i,j}$值是坐标为(i,j)的栅格的成本

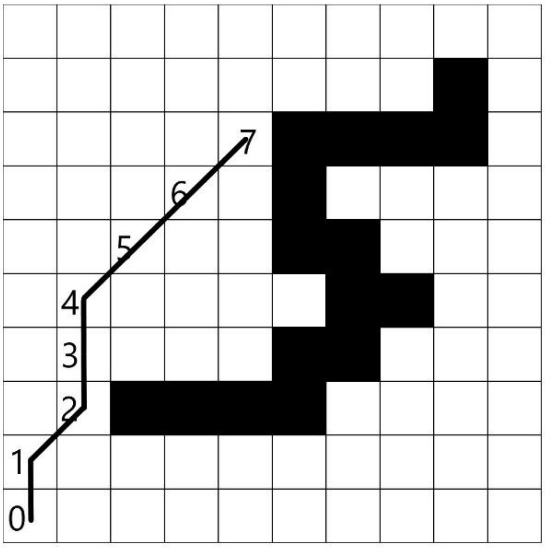


图 3 栅格代价演示图（以（5, 8）为例）

图 3 以（5,8）为例，标明了路线上每一格的代价。

*Step2.*在规划线路时，运用$\boldsymbol{A}^*$算法[2]，通过某一个栅格的成本来获取这一栅格周围栅格的成本，用公式表示为

$$\begin{cases} A_{i-1,j-1} \le A_{i,j} + 1 \\ A_{i-1,j} \le A_{i,j} + 1 \\ A_{i-1,j+1} \le A_{i,j} + 1 \\ A_{i,j-1} \le A_{i,j} + 1 \\ A_{i,j+1} \le A_{i,j} + 1 \\ A_{i+1,j-1} \le A_{i,j} + 1 \\ A_{i+1,j} \le A_{i,j} + 1 \\ A_{i+1,j+1} \le A_{i,j} + 1 \end{cases} \tag{2}$$

即$(i-1, j-1)$栅格、$(i-1, j)$栅格、$(i-1, j+1)$栅格、$(i, j-1)$栅格、$(i-1, j+1)$栅格、$(i+1, j-1)$栅格、$(i+1, j)$栅格、$(i+1, j+1)$栅格的成本至多为$A_{i,\ j}+1$，然后再通过周围一圈栅格获取更大一圈栅格的成本。

*Step3.*在计算每个栅格的成本时会有多个$A_{i,j}$值的出现，因为每一个栅格存在多条路径从起点（S）到达终点，由不同的路径所推导得到的某一个栅格的$A_{i,j}$值又是不一样的，因此我们在对$A_{i,j}$取值时需要对其做处理。由于题目要求我们找出最优的行进路径，因此我们假设$B_{i,j}$。

$$B_{i,\ j} = min\left(A_{i,j}\right) \tag{3}$$

其中$B_{i,j}$是栅格所接受到的所有成本$A_{i,j}$的最小值。

*Step4.*从起点出发对地图中的所有栅格进行最小成本的计算，终点的最小成本所经过的路径即为本题的解，就是所要求的最优行进路径。

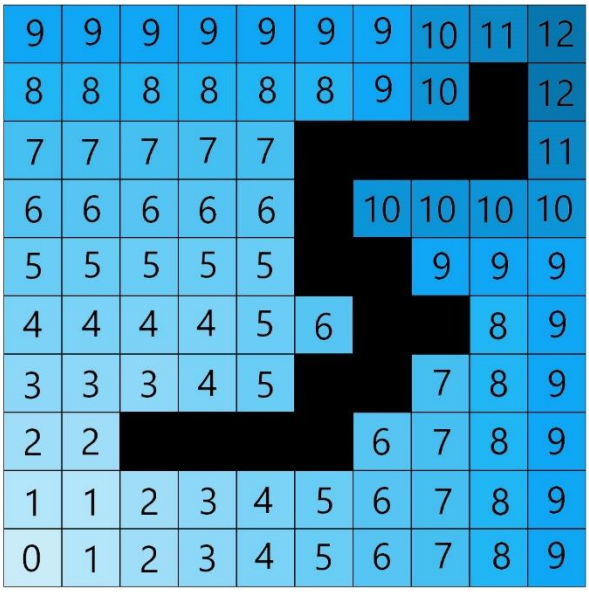


图 4 辐射模型栅格地图

5.1.1.3 最优行进路线模型的算法求解

求解模型的前提是需要给定可靠的数据，通过电脑生成规则的障碍物栅格地图，然后在此数学模型的基础上建立合理的算法[2][3]。

Step1. 定义栅格地图中所有栅格的成本为$A_{i,j}$，对它们进行赋初值操作，为方便后续的比较和求解$A_{i,j}$值，将初值定为无穷大。

Step2. 通过洪泛算法[2]，求解从起点开始的每个栅格的成本$A_{i,j}$。当由某条路径获得的栅格成本小于原来的$A_{i,j}$*时*，更新此栅格的$A_{i,j}$值。不断进行路径模拟与赋值，可以得到整幅地图所有栅格的最小成本$B_{i,j}$。

Step3. 利用 *Step2* 中求解得到的栅格图的最小现实代价$B_{i,j}$，找到从终点开始逐一递减的路径，该路径即为最优路径。

5.1.2 规则障碍物模型的结果

通过计算机随机产生两种不同规则障碍物摆放的栅格地图，将这些地图的具体参数分别带入模型之中，编程求解(详见代码文件 第一题.cpp)，得到的结果栅格示意图如下所示：

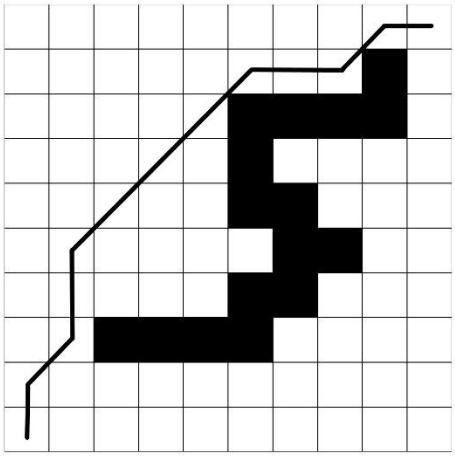


图 5 最优路径结果示意 1

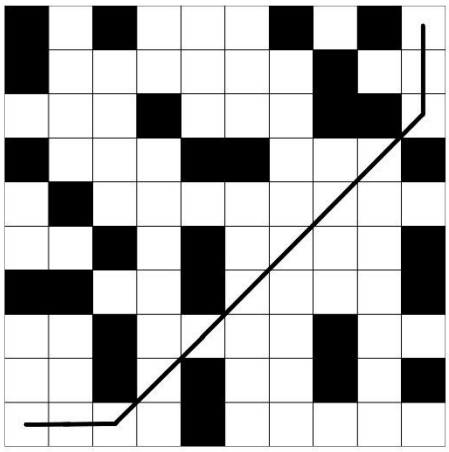


图 6 最优路径结果示意 2

5.1.3 不规则障碍物情况

问题一的第二种情况是关于不规则障碍物栅格图的最优路径求解问题，本情况的背景信息与第一种规则障碍物的一样，其不同之处在于不规则障碍物并不会占据完整的栅格，当某一栅格中的不规则障碍物所占据的面积大于 50%，则默认该栅格移动机器人不能通过，反之，可以通过。

5.1.3.1 不规则障碍物情况下最优路线模型的工作原理

尽管不规则障碍物与规则障碍物存在差异，但其关于最佳路径求解的本质工作原理是一样的，唯一有所不同的是一个是规则的，另一个是不规则的。由于本问机器人在出发之前已知晓全栅格地图信息，因此我们考虑通过合理的数学模型将不规则的图形转化为规则的障碍物或允许通行的栅格[3]，再利用第一种规则障碍物的求解方法，对这一种情况进行求解。

5.1.3.2 不规则情况下行进路线的模型与算法求解

Step1. 建立转换模型。我们采用像素格来微分所有的不规则障碍物，将其量化为一个个像素点，设某个栅格内障碍物所包含的像素点个数为s_i。设此栅格的总像素点个数为 S，当

$$\frac{S}{2} < s_i \#(4)$$

则将该栅格定义为障碍物，反之，则定义该栅格可以通行。

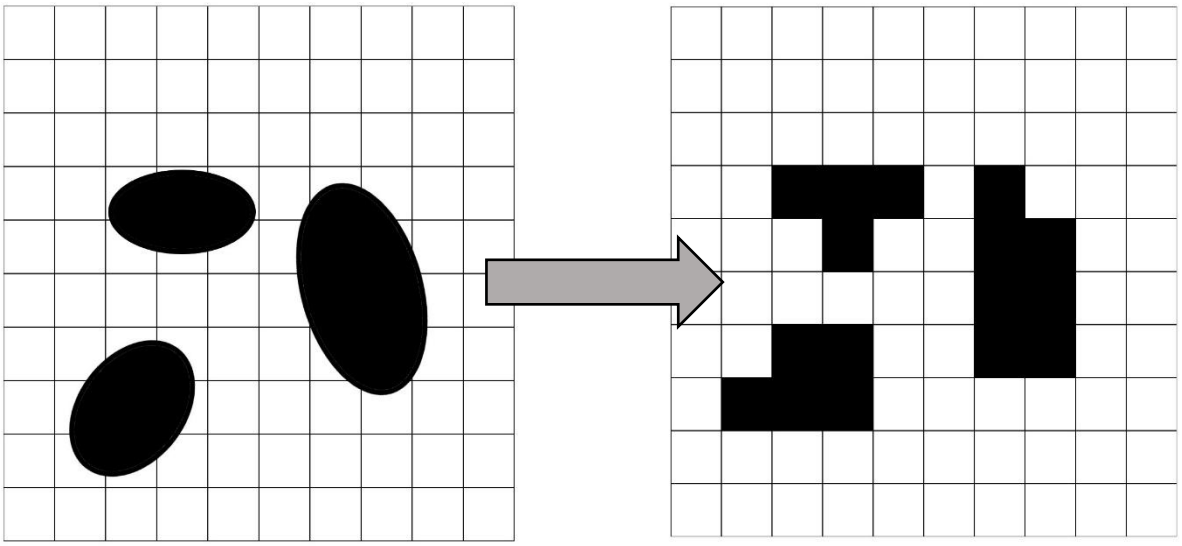


图 7 不规则障碍物转化图

Step2. 算法求解。在进行路径规划之前，将不规则障碍物的颜色渲染成与栅格图中网格线不一样的颜色，并通过 *RGB* 颜色的差异，对这些不规则的障碍物进行识别，得到每个栅格中障碍物所包含的像素点个数，将此不规则障碍物转化为规则障碍物，再利用规则障碍物情况下的算法进行求解。

5.1.4 不规则障碍物模型的结果

通过计算机随机生成的不规则障碍物栅格图，使用 Python 编程(详见代码文件 右图转左图 10x10 实验.html)提取不规则障碍物栅格图信息，并将其转化为规则障碍物的信息，将这些具体参数信息带入第一种情况的代码中求解，得到的结果栅格示意图如

下：

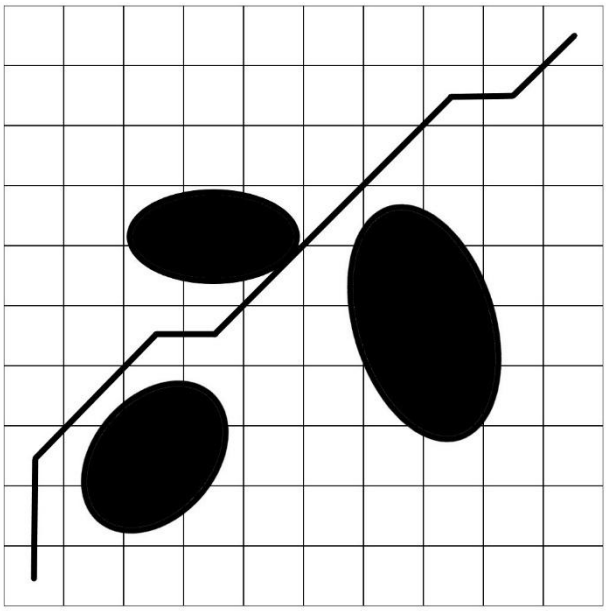


图 8 不规则障碍物结果示意图

5.1.5 最优路径模型的实用性

从结果上来看，模型针对于规则障碍物和不规则障碍物情况下所设计的路径线路均为最优解，且具有普适性。栅格地图中显示的关于成本的具体值没有错误。这也进一步验证了该路径线路模型的合理性与普适性。

5.2 问题二的模型建立与求解

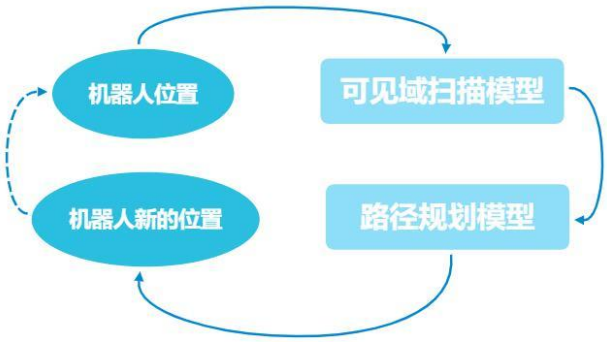


图 9 机器人运动示意图

5.2.1 规则障碍物的情况

问题二要求我们给出机器人在随机生成障碍物的栅格地图中从起点抵达终点的最优路径线路。由图 9 所示，我们将问题二的机器人行进路线问题简化为了视野区域扫描问题与具体局部动态路径规划问题。因此需要建立两个数学模型来得到机器人的最优行进路线。

5.2.1.1 规则障碍物路径系统的工作原理

规则障碍物的最优路径系统主要由两部分组成，即为视野区域扫描模型和局部动态路径规划模型。机器人每次的行进路线都需要经过视野区域扫描模型和局部动态路径规划模型的联动来实现。因此首先需要明确这两个模型的工作原理和定量化描述分析。

1)视野区域扫描模型的工作原理

在整个移动机器人行进过程中，视野区域扫描需要持续进行，即每当移动机器人移动一步，视野区域扫描模型都需要启动，对目前机器人所处位置的周围环境进行扫描。因为移动机器人在栅格地图中是需要移动的，并且该栅格地图中存在着障碍物，当移动机器人位于不同的栅格时，其所能看见的地图信息是不一样的，视野区域扫描模型的情况也是不同的，通过视野区域扫描实时更新出的线路情况同样不一样，而这些都会影响局部动态路径规划模型做出最优路径选择的判断，因此实时的视野区域扫描是必要的[4]。

在整个视野区域扫描的过程中需要对不同的栅格进行判断，有不可通行状态和可通行状态，顾名思义，这些状态信息对应于任意一个可扫描到的栅格。

设每个栅格的状态是C_{ij}，并对每个可以通过当前移动机器人位置扫描到的栅格的状态进行赋值[5]：

$$C_{i,j}=\begin{cases}-1, & \textit{不可通行状态}\\ 1, & \textit{可通行状态}\end{cases}\#(5)$$

其中，当栅格处于-1 状态时，表示该栅格存在障碍物，不可通行；当栅格处于 1 状态时，表示该栅格不存在障碍物，可通行。

2)局部动态路径规划模型的工作原理

在机器人行进过程中，局部动态路线规划需要建立在视野区域扫描的基础上进行。视野区域扫描将移动机器人所处栅格位置周围所能看见的点确定以后，局部动态路径规划需要确定从移动机器人所处的当前栅格出发，到达终点可能的最优解。为达到这个目的，需要计算视野区域扫描范围内的任意一个栅格位置所需要的成本以及从视野区域的边缘到达终点可能存在的情况，再通过利用这两个信息设计算法获得可能存在的最优路径。

5.2.1.2 视野区域扫描模型

1)目标函数的确定

对于视野区域扫描模型而言，其所能得到的目标函数就是在移动机器人当前所处位置所能观察到的每个栅格的信息$C_{i,j}$，它表示了每个栅格各自的状态信息。

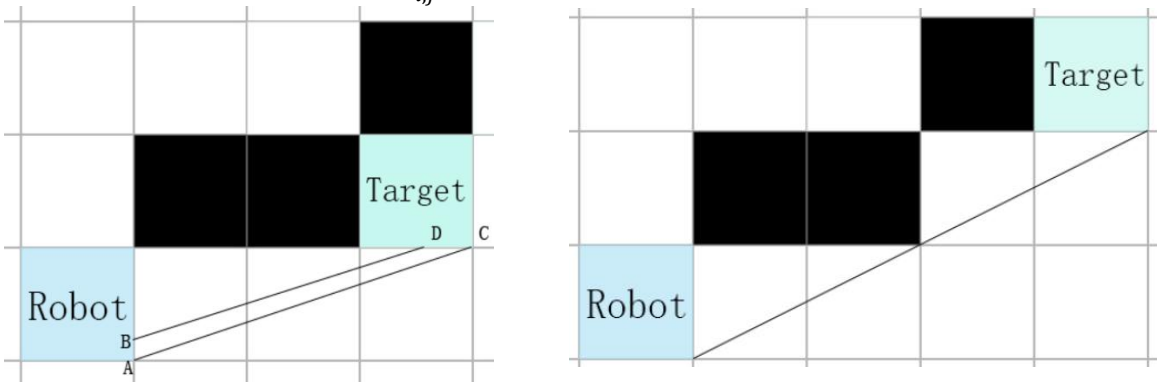


图 10 视野区域扫描区间示意图（左图看得见目标而右图看不见）

2)初始方案的策划

Step1. 定义视野区域扫描的区间范围。在机器人所处位置的栅格与目标栅格的各自任意一条边上，分别存在两点 A,B 和 C,D,点 A 和点 C 相连，点 B 和点 D 相连，当两条线段 AB 和 CD 互不相交，则认为移动机器人在所处的栅格处可以看到目标栅格；反之则认为移动机器人的视野区域范围内不包含目标栅格。

Step2. 定义特殊情况下的视野区域扫描范围。根据题目要求，移动机器人可以在对

角线之间通行，因此，存在着移动机器人对于障碍物夹着的对角线之间的视野区域扫描范围定义。如图 11 所示，黑色方块是存在障碍物的栅格，白色是可以行进的栅格，当移动机器人位于图示位置时，按照题目要求，机器人可以行进到目标栅格，但这种移动机器人行进情况的视野区域扫描并不满足 *Step1* 中的规定。因此我们特别地，在这种特殊情况下认为处于该情况下的机器人能够扫描到目标栅格的信息，但也仅限于目标栅格。

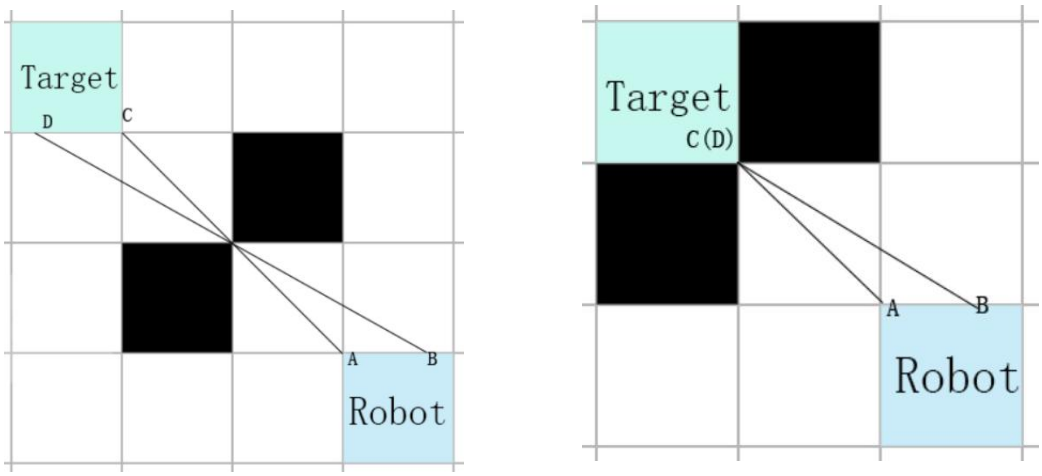


图 11 特殊情况视野区域示意图（左图看不见目标但右图看见）

Step3. 视野区域扫描的结果分析。我们充分考虑了移动机器人的智能化[4][5]，将机器人的视野类比于人类，人类拥有双眼，并且需要两只眼睛的呈像才能看到较为广泛的面积，因此我们在对视野区域扫描模型建立时，采取的是双线段模式。在 *Step1* 和 *Step2* 的视野区域扫描区域的定义下，可以直观地在栅格地图上表达出移动机器人在所处任意一个栅格时能扫描观测到的区域，并且利于视野区域扫描结果应用于局部动态路径规划的基本模型中。通过视野区域扫描可以直接定义所有移动机器人可见栅格与不可见栅格的信息，可以便于后期计算机对于这些结果的识别。

5.2.1.3 局部动态路径规划的基本模型

1)目标函数的确定

针对于局部动态路径规划模型，移动机器人需要在视野区域扫描结果的基础上成功避开障碍物并寻找到通往终点的最短路径。此处我们采用改进的蚁群算法，此算法把 Dijkstra 算法（靠近初始点的结点）和 BFS（Breadth First Search）算法（靠近目标点的结点）的优点结合起来。其估价函数为[6]

$$f(n) = g(n) + h(n) \#(6)$$

式中：$f(n)$为从起点（S）到终点（D）的估计代价；$g(n)$为从起点到状态 n 的成本；$h(n)$为从状态 n 到终点（D）的估计代价。想要获得最优路径，即使得$h(n)$最小。

2)初始方案的策划

Step1. 构建基本模型的框架，确定估价函数中各个表达式的含义。构造出衡量最短路径的函数[2]：

$$T_{ij} = x_{ij} + k \times y_{ij} \#(7)$$

式中，其中将估价函数的$g(n)$定义为视野范围内从起点（S）到视野区域边缘某一栅格的成本，用变量x表示；将不可见区域部分$h(n)$构造为$k \times y_{ij}$，其中，y_{ij}为在没有障碍物的情况下从视野区域边缘某一栅格至终点（D）的最小代价。y_{ij}的计算过程如下

$$y_{ij} = \max(|i-1| + |j-W|) \tag{8}$$

式中 W 为地图宽度。$(1, W)$即为终点坐标。

k 为估价因子。将$\frac{1}{T_{ij}}$定义为信息素浓度，此浓度只在视野区域的边缘栅格有意义，且此浓度越大，说明经过该点前往终点所需时间最短。值得注意的是，如果终点出现在可见域内，那么终点的信息素浓度一定是最大的，因为终点在蚁群算法中相当于食物源。

Step2. 估价因子 k 值的确定。因为机器人从视野区域边缘的栅格移动到终点的这块区域，对于当前位置的移动机器人而言是一个未知的、不确定的路径。我们需要对估价因子 k 的值进行估计，从而预估出此时所选择的的这条路径的成本。我们在大数据技术的帮助下，绘制出估价因子 k 值的变化曲线，以此求解出估价因子 k 的取值。

Step3. 通过估价因子 k 的值和目标函数，计算出机器人当前视野域范围边缘栅格的信息素浓度。

Step4. 比较其视野区域范围内不同栅格的信息素浓度，选取其中浓度最大者，找出前往该格的最短路径。

Step5. 沿求得的最短路径前进一步。

Step6. 重复 *Step3*，*Step4* 和 *Step5* 直至抵达终点。

5.2.1.4 最优路径的算法求解

最优路径的算法求解分为两个算法，其中一个是视野区域扫描模型的算法，另一个是局部动态路径规划模型的算法。

1)视野区域扫描模型的求解

对于该问题，我们提出极角排序的算法思想。考虑到不同障碍对于机器人所在栅格的不同顶点的极角不同，我们将扫描分为**横向扫描**和**纵向扫描**，而横向扫描和纵向扫描都要**对四个象限进行扫描**。以机器人当前所在栅格位置为原点，建立直角坐标系，将整个栅格地图平面划分成四个象限。

对于四个区域的求解算法都是相似的，我们以第一象限为例。从远点位置沿 x 轴做一条射线，射线逐渐绕着原点位置逆时针转动，当射线与障碍物的边或顶点相切时，记录此时的角度θ_1，但由于在算法程序中我们很难直接计算出角度θ_1，因此我们固定此角度θ_1，从射线向 x 轴做一条垂线，从垂线的长度记为h_1，垂线与 x 轴交点到原点的距离记为d_1,我们用h_1和d_1的比例关系来间接表示此时的夹角θ_1。这样我们可以在第一象限求出许多个θ_n，用这些角度来定位所有相切的射线，两射线之间的覆盖的面积就是位于原点处的移动机器人所能看到的面积。当视野区域覆盖任意一个栅格的一部分时，则知晓此栅格的通行状态$C_{i,j}$，即该栅格可见。

用该算法思想求解四个象限，即可求出视野区域扫描的栅格范围。

以下是该算法下的求可见域的案例：

如图 12，左图中蓝色区域是机器人当前可以看见的区域，α即为横向扫描时第一象限的可见极角。因此 Q 点便是可见的，而 P 点不可见。

将可见极角扫描到的区域栅格化后，便得到了右图。值得注意的是四个深色栅格是满足了特殊情况，因此被认为是可以看到的。

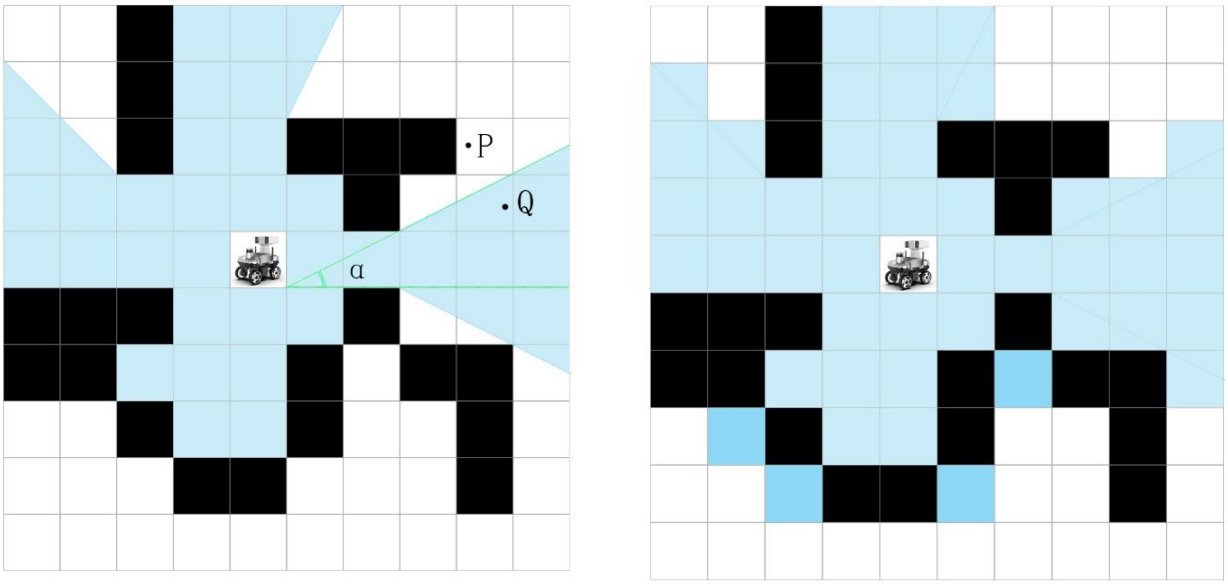


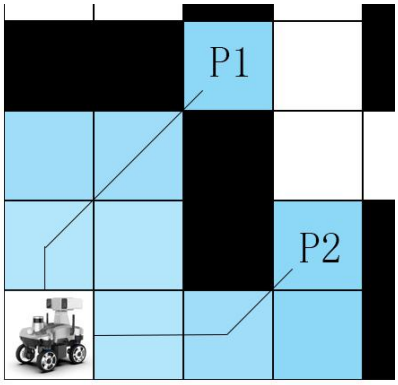
图 12 模拟人类视觉扫描法的示例

如图 12，左图中蓝色区域是机器人当前可以看见的区域，α即为横向扫描时第一象限的可见极角。因此 Q 点便是可见的，而 P 点不可见。将可见极角扫描到的区域栅格化后，便得到了右图。值得注意的是四个深色栅格是满足了特殊情况，因此被认为是可以看到的。

2)局部动态路径规划模型的求解

局部动态路径规划算法我们采用信息素浓度概念，信息素浓度与最短路径函数T_{ij}成反比。机器人会在所处栅格位置进行视野区域扫描，对扫描出的所有边界栅格求它们的信息素浓度。然后机器人向信息素浓度高的方向移动。

下面将给出一个具体的操作示例。



P1 (7,3)

P2 (9,4)

机器人位于(10,1)

$x_{7,3}=3$　　$x_{9,4}=3$

$y_{7,3}=7$　　$x_{9,4}=8$

$y_{7,3}=7$　　$x_{9,4}=8$

$T_{7,3}=3+7k$　　$T_{9,4}=3+8k$

$T_{7,3} < T_{9,4}$

图 13 局部动态路径规划模型的求解示例

如图，机器人位于坐标为(10,1)的栅格内，蓝色为其可见域，P1 和 P2 为可见域边缘的两点，前往 P1 和 P2 所需的代价x相同，但 P1 和 P2 前往终点的最小代价y不同，因此机器人将朝 P1 的方向前进一步，来到(9,1)栅格内。如此便完成了第一步。随后的操作与之类似。

5.2.2 规则障碍物路径规划的结果

利用算法求解在无法知晓栅格地图全貌信息下的最优路径规划模型(详见代码文件第二题 K.cpp)。我们通过随机生成障碍物的方式产生栅格地图模型，带入我们的算法求解的结果如下。

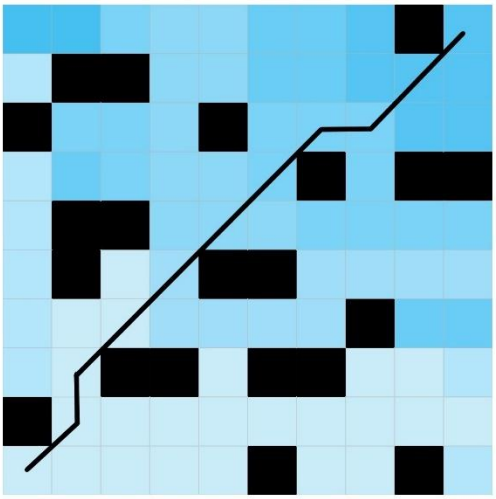


图 14 不可见规则障碍物结果 1

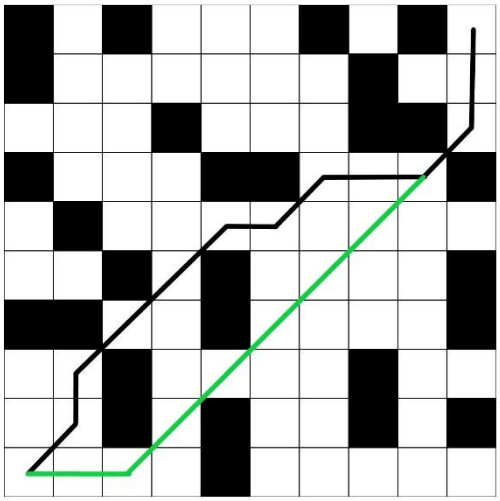


图 15 不可见规则障碍物结果 2

如图 14 中所示，蓝色方块表示可以被看见的栅格，蓝色越深则表示被看见的时间越迟。在图 14 中，机器人幸运的找到了一条事实上的最短路。图 15 中的情况则不一样。在图 15 中机器人按照动态局部贪心法则[2]的寻路，选择了偏上方的黑色路线，花了 12 步走到终点。但事实上的最优路径是途中绿色的路径（路径末段与黑线重合，因而被黑线覆盖）这也说明了动态局部贪心法则是存在缺陷的。但在信息不足的情况下，确实难以一次性找到最优路径。

通过对大数据集的测试，我们发现，相比较于问题一不规则障碍物情况下的实验结果，问题二在不规则障碍物情况下会比问题一的实验结果平均多 12.63%。

5.2.3 不规则障碍物情况

对于不规则障碍物而言，其最优路径规划系统大部分的工作原理与规则障碍物的工作原理相一致，所以部分参数、变量的定义我们将一直延续规则障碍物中的定义。但其在视野区域扫描模型上有区别，因此我们需要重新建立新的视野区域扫描模型，满足移动机器人在出发前不知晓不规则障碍物栅格地图信息的情况下的移动。

5.2.3.1 不规则障碍物最优路径规划的工作原理

1)视野区域扫描模型的工作原理

由于这是不规则障碍物的情况，我们没办法类比规则障碍物中的可见域扫描模型，因为在规则的障碍物情况中，每一个栅格只能是存在两种情况，即存在障碍物或不存在障碍物，因此我们可以通过人类视线的物理模型来建立线段不相交的视野区域扫描模型，利用观察到某个栅格的某一条边的通行状态来判断此栅格是否存在障碍物。但是在不规则障碍物中此做法并不可行。由于障碍物可能不会完全占有一个完整的栅格，只是占据了其中的部分面积，因此我们没有办法利用判断出的一条边的通行状态来推出整个栅格的通行状态$C_{i,j}$。所以提出了通过率的模型，选定栅格地图内的某一目标栅格，从机器人所在栅格位置向目标栅格做切线，由于在目标栅格与移动机器人之间可能存在一些不规

则障碍物，它们会影响移动机器人的视野，需要将它们的边界线与机器人视野的射线做相切，并利用这些切线算出当前栅格至目标栅格间通过率的大小，其中通过率定义为某一个栅格必定不存在障碍物部分的面积占栅格的面积。由此判定机器人能否经过此栅格。

2)局部动态路径规划模型的工作原理

不规则情况下的局部动态路径规划跟规则情况下的相同，此处不再做赘述。

5.2.3.2 视野区域扫描模型

1)目标函数的确立

目标函数的确立与规则障碍物的情况一样，即在保证成功规避障碍物的同时，寻找移动机器人从栅格任意位置到终点的最短路径，也就是成本目标函数

$$minT_{ij} \tag{9}$$

其中T_{ij}表示从移动机器人当前所处的栅格到终点的任意路径的成本。

2)初始方案的策划

Step1. 构建基本模型的框架。通过参考文献资料[6]，我们从光线散射的物理现象得到启发，建立了辐射-信息量扫描法的模型。从移动机器人所在的任意栅格位置，向目标栅格位置做两条平行线。若起始栅格和目标栅格位于同一个 X 轴或者同一个 Y 轴，则这两条平行线必须同时过这两个栅格的两条边；若起始栅格与目标栅格不在同一个 X 轴或者同一个 Y 轴上，则这两条平行线必须过这两个栅格的四个顶点。

将目标栅格朝着机器人所处位置的栅格在平行线间移动，获取栅格在不同位置时的通过率S_{ij}。

Step2. 确定障碍物栅格的默认通过率$S_{参考}$与某一栅格的通过率$S_{i,j}$。

障碍物栅格的默认通过率是通过机器学习的方法对大样本进行处理，使得在此$S_{参考}$下，机器人对栅格地图的判断最为准确。而移动机器人的通过率$S_{i,j}$是非障碍物面积占目标栅格总面积的比值，即

$$S_{i,j} = \frac{S_{非障碍物}}{S_{总}} \#(10)$$

通过此信息，判断出视野区域可以通行与不可通行的栅格信息。

5.2.3.3 局部动态路径规划模型

不规则障碍物最优路径规划模型与规则障碍物最优路径规划模型一样，也是存在了两个子模型，即视野区域扫描模型和局部动态路径规划模型。因为两者局部动态路径规划子模型的应用完全一样，差异在于视野区域扫描模型[4]。由图 10 可知，在进行局部动态路径规划模型处理之前，需要经过视野区域扫描模型进行处理，而不规则障碍物情况下的视野区域扫描模型最终会将结果转化为规则障碍物视野区域扫描结果的格式。由于不规则障碍物和规则障碍物两者的视野区域扫描结果的输出结果格式是相同的，因此在这里可以直接使用规则障碍物情况下的局部动态路径规划模型，所以在此将不再详细介绍路径规划模型了。

5.2.3.4 最优路径的算法求解

问题二中的不规则障碍物情况仍然有两个模型，即视野区域扫描模型和局部动态路径规划模型。由于我们的算法思想在于利用视野区域扫描模型将不规则障碍物规则化，得到问题二中第一种情况的栅格信息。因为我们的局部动态路径规划算法输入的值类型是一样的，所以我们沿用规则情况下的局部动态规划路径算法即可[3]。在此我们重点阐

述视野区域扫描算法。

视野区域扫描算法[4]的核心在于计算机器人的通过率。我们需要选定视野区域范围内的目标栅格，在移动机器人所在栅格位置向目标栅格做切线，由于在目标栅格与机器人之间可能存在一些不规则障碍物，它们会影响移动机器人的视野，需要将它们的边界线与机器人视野的射线做相切，在从移动机器人位置到目标栅格位置之间，存在一段视野宽窄变化的线条带，先求解出这条线条带上的连续变化的宽度值，求得它的最小值，当最小值超过默认的通过率$S_{参考}$，则移动机器人可以移动到视野区域范围内的目标栅格位置，反之则不行。利用这样的方法，通过循环的程序计算得到视野区域范围内所有边界栅格的是否可以到达，若可以则该栅格为可通行，若不行则该栅格为障碍物，需要计算新的边界栅格能否通行，以此通过算法不断反复，最后求出一个视野区域的最终范围。

5.2.4 不规则障碍物最优路径规划的结果

求解在无法知晓栅格地图全貌信息时的不规则障碍物最优路径规划模型(详见代码文件 第二题扫描改良改+K.cpp)。我们通过随机生成障碍物的方式产生栅格地图模型，带入我们的算法求解的结果如下。

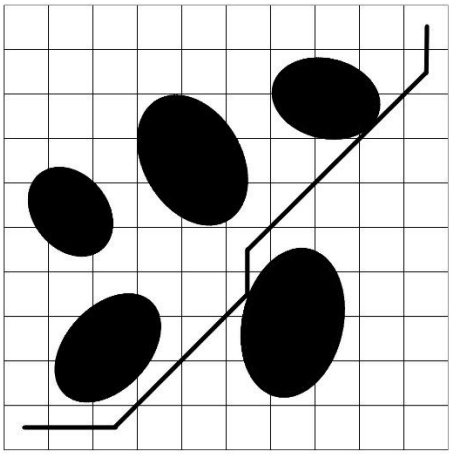


图 16 不可见不规则障碍物结果 1

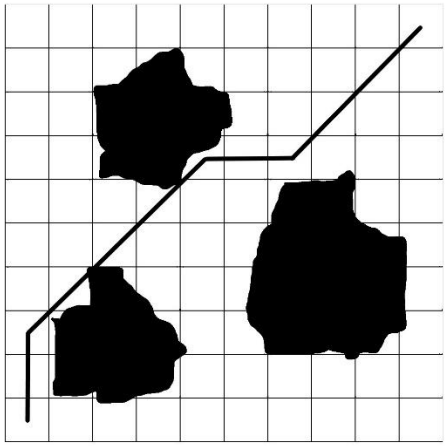


图 17 不可见不规则障碍物结果 2

可以发现，对于这两张示例图，我们的求解路径和全图信息已知的情况下得出的最优路径是一致的。但在大数据集上测试的结果显示，我们的寻路算法要比已知情况下的多花约 13%的时间。

5.2.5 不规则障碍物最优路径规划的实用性

从结果上来看，模型得出的结果均符合实际，没有出现异常的情况，即碰撞障碍物，并且也是尽量达到最优路径的选择。因为移动机器人不知道后面的障碍物分布，因此只能通过已知量来推，但是也仍然会存在比移动机器人知晓栅格地图全信息下的路径步数平均多一些。这也是模型不可避免的误差，与实际情况相符合，进一步证明了模型的准确率和实用性。

六、模型的灵敏度分析

对题目而言问题一不需要进行灵敏度分析。因为在问题一中，移动机器人在出发前已经知晓栅格地图所有的障碍物信息，在问题一中的两种情况，规则障碍物和不规则障碍物中均没有任何变量，所有的栅格地图参数都是已知量。我们针对问题二中局部动态

路径规划模型里面的比例系数做灵敏度分析。令比例系数 k 从 0.1 到 10 浮动，绘制比例系数参数与平均开销比率的关系。问题二情况下的模型灵敏度分析图如下所示(详见代码文件 Ktest.cpp)

图 18 比例系数 k 的灵敏度分析图

如图所示，对于比例系数 k 而言的结果呈现图。在考虑实际应用时，需要谨慎考虑比例系数 k 值，尽可能得取到平均开销最低值所对应的 k 值，以保证模型的效率。

七、模型的综合评价和推广

7.1 模型的综合评价

7.1.1 模型的优点

本文在考虑问题二的第一种情况，规则障碍物栅格地图的情况下，从多方面分析，确立了实用性能极强的可见域扫描模型。本文充分考量了规则障碍物的特点，即当任意一个栅格有一部分存在障碍物时，则此栅格是障碍物栅格。我们的可见域扫描模型没有进行复杂的全局扫描，但得到了理想的结果

本文在考虑问题二的第二种情况，不规则障碍物栅格地图的情况，也建立了实用性强的可见域扫描模型，简化了模型和算法的计算量，但并没有减弱此模型的功能，效率和稳定性。

本文在所有的题目中都考虑了随机生成障碍物的情况，并通过对多种情况的分析建立出满足所有随机障碍物生成状况的模型。这也使得本文对模型的考虑更加实际情况，更加精细。

7.1.2 模型的缺点分析

本文所有数据测试均使用原图进行测试，因为算法追求拟真性而导致时间复杂度较高，放在更大的地图中，可能效率较低。

采取辐射-信息率扫描时，仍然使用离散化的思路。如果改用连续性的数值计算信息素，可能会有更好的效果。

7.2 模型的推广

本文主要利用可见域扫描模型和局部动态路径规划模型对最优路径规划制定了高效的策略。其中运用的最多的就是可见域扫描模型，这个不仅可以运用在这种背景情况下的移动机器人走栅格地图中，还可以应用到更多的模型中去。对于问题本身，存在了移动机器人提前知晓和不知晓栅格地图信息的情况，这两种情况在现实生活中的应用都

很强。一些环境下，就是需要在不知晓全信息的情况下，尽可能地减少走的“弯路”，降低算法和模型的复杂度；在其它一些情况，就需要让智能体提前知晓地图信息，并规划出最短的路径，提高效率。在模型中我们也存在着一些变量，这些变量会随着模型基本参数的变化而影响模型的效率，因此我们需要及时对这些变量进行调整，以此确保得到高效的实际应用方案和模型。

八、参考文献

[1]刘永建,曾国辉,黄勃,等.改进蚁群算法的机器人路径规划研究[J].电子科技,2020(1):13-18.
[2]任红格,胡鸿长,史涛.基于改进蚁群算法的移动机器人全局路径规划[J].华北理工大学学报(自然科学版),2021,43(02):102-109.
[3]杨凌耀,张爱华,张洁,宋季强.栅格地图环境下机器人速度势实时路径规划[J/OL].计算机工程与应用:1-8[2021-05- 24].
http://kns.cnki.net/kcms/detail/11.2127.tp.20210514.1007.008.html.
[4]蒋林,方东君,周和文,黄惠保.基于射线模型的改进全局路径规划算法[J/OL].电子学报:1-10[2021-05-24].
http://kns.cnki.net/kcms/detail/11.2087.TN.20210429.1538.002.html.
[5]彭湘,向凤红,毛剑琳.一种未知环境下的移动机器人路径规划方法[J].小型微型计算机系统,2021,42(05):961-966.
[6]杨光辉.基于人工智能优化算法的大型舰船紧急疏散路径规划[J].舰船科学技术,2021,43(08):52-54.

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：

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;
}
```