

# 基于贪心算法结合时空 A\* 算法的多智能体规划问题

## 摘要

在本文中,针对不同地图尺度和机器人数量,建立了机器人路径规划模型,并在多任务场景下进行规划,以最小时间和最优路径为目标,具有物流、无人机规划等领域的应用价值。

**对于问题一**,机器人同时发车且只有一个任务目标并且在完成任务后消失的条件之下,本文以最小时间为规划目标,考虑**顶点约束、边约束、边界约束、障碍物约束、时间约束**,建立基于 0-1 整数规划的多智能体规划模型。考虑相同时间步长之下机器人不能碰撞,利用三维**时空 A\* 算法**去进行模型的求解。可求得 8\*8 地图最小的时间步长为 9, 16\*16 地图当中最小的时间步长为 21, 64\*64 地图时间步长为 119。对于算法复杂度而言,本文通过分析算法当中具体的如何搜索状态空间得出时间复杂度为  $O(m * n^4)$

**对于问题二**,机器人的所有状态都与问题一当中类似,只添加了机器人在完成任务过后并不会消失而是会成为一个障碍物。因此本文在问题一的约束基础之上加入障碍物更新约束,要求所有机器人到达终点后,该点将始终判断该机器人停留在原地直至遍历结束。基于此本文重构了基于 0-1 整数规划的多智能体规划模型。在求解过程当中,本文重新考虑了时空 A\* 算法对于障碍的判断,更加注重全局的车辆变成障碍物的约束,构建起**全局时空 A\* 算法**,求解得到 8\*8 地图最小的时间步长为 9, 16\*16 地图当中最小的时间步长为 21, 64\*64 地图时间步长为 119。对于算法复杂度而言依旧是全局时空 A\* 算法的时间复杂度  $O(m * n^4)$

**对于问题三**,由于机器人数目与任务总数不相同,在问题一的基础之上考虑加入任务迭代约束、任务完成情况约束、任务完成时间约束、障碍物更新约束,重构基于 0-1 整数规划的多智能体规划模型。在模型求解层面来说,考虑到机器人完成的任务数量和时间呈单调关系,我们通过利用**二分枚举时间的贪心算法**去考虑多机器人的任务分配问题,贪心策略即每个机器人去完成通过时空 A\* 算法搜索所有任务里开始位置距离当前机器人位置最近且未被访问过的任务。可求得 8\*8 地图最小的时间步长为 68, 16\*16 地图当中最小的时间步长为 237, 64\*64 地图时间步长为 596。对于算法复杂度而言因为加入了二分枚举时间,因此时间复杂度为时空 A\* 算法和二分枚举时间得到的时间复杂度  $O(m * n^4 \log_2 M)$

**对于问题四**,在问题三的基础之上加入了每个机器人一定要去完成的任务,模型依旧是问题三当中加入任务迭代约束、任务完成情况约束、任务完成时间约束、障碍物更新约束,重构了的 0-1 整数规划的多智能体规划模型。本文在进行贪心考虑多机器人的任务分配问题时要考虑任务的合法性,即任务被分配给指定机器人。基于此本文利用贪心求解得到 8\*8 地图最小的时间步长为 42, 16\*16 地图当中最小的时间步长为 120, 64\*64 地图时间步长为 616。时间复杂度依旧为二分枚举时间的复杂度  $O(m * n^4 \log_2 M)$

**关键字:** 0-1 整数规划 多智能体规划模型 时空 A\* 算法 二分枚举时间贪心算法

# 一、问题重述

## 1.1 问题背景

随着业务模式的多样化与市场的发展，传统的仓储系统已难以满足快速物流的需求，因此需要更高效的物流解决方案，寻求一种高响应速度，快速货物寻找运输的自动化仓储系统。

考虑有如下的场景：在一个工厂中，我们将这个工厂按照二维网格划分。针对每一个网格，其可分为障碍物（不可通行）和可通行两种属性。在这些可通行的部分网格中，存在着部分的机器人，对于每个机器人，其具有两个位置属性，分别为其起点位置与终点位置。我们的任务是在这个场景下以最小化运输总时间完成给定的运输任务。

对于运输时机器人的规则设定，有关于时间的设定，在一个时间步内，每个机器人都可以在其当前顶点等待，或者选择从一个时间步到下一个时间步移动到相邻网格。我们设定相邻的网格仅为前后左右四个网格，对角线上的相邻网格不计入本题中的相邻网格；以及关于碰撞设定，为了避免机器人发生碰撞，我们要求为每个机器人找到一条安全的路径。首先，要求在任何时间步中不能有两个或多个机器人位于同一顶点。其次，要求在任何时间步之间的移动中两个或多个机器人不能经过同一条边。

## 1.2 问题要求

**问题 1** 假设每个机器人只有一个任务目标并在完成任务后消失。如果每个单位时间内机器人只能在停在原地和移动中选择一项，在不发生冲突的前提下建立机器人运输总时间最小化的数学模型并求解，结果记入附件。

**问题 2** 假设每个机器人只有一个任务目标并在完成任务后停留在目的地，其余规则与问题一保持一致，在不发生冲突的前提下建立机器人运输总时间最小化的数学模型并求解，结果记入附件。

**问题 3** 假设机器人数目与任务总数不对等，其余规则与问题二保持一致，在不发生冲突的前提下建立机器人运输总时间最小化的数学模型并求解，结果记入附件。

**问题 4** 假设部分任务已经被指定的机器人来完成，任务完成的先后顺序不做固定，其余规则与问题三保持一致，在不发生冲突的前提下建立机器人运输总时间最小化的数学模型并求解，结果记入附件。

## 二、问题分析

### 2.1 问题一分析

对于问题一，在此场景中，所有机器人同时从初始位置出发，并且每个机器人只有一个任务目标。任务完成后，机器人将从场景中消失，不再参与后续的路径规划。问题的核心是在约束条件下，找到所有机器人从起点到任务目标的最短路径，并确保各机器人之间在相同的时间步长内不会发生碰撞。

### 2.2 问题二分析

对于问题二，与问题一类似，但在此场景中，机器人完成任务后不会消失，其位置将被视为一个新的障碍物，持续影响后续的路径规划，模型在原有的 0-1 整数规划基础上，增加了障碍物更新的动态处理，并采用全局时空 A\* 算法来处理新的全局障碍物影响，确保在全局环境中找到最优路径。

### 2.3 问题三分析

对于问题三，在此场景中，机器人数量与任务总数不相等，要求对任务进行合理分配，使得所有任务在最短时间内被完成。在问题一的基础上，利用二分枚举时间的贪心算法进行任务分配。算法通过时空 A\* 搜索所有可能任务中最接近当前机器人的任务位置，确保在最短时间内完成所有任务。

### 2.4 问题四分析

对于问题四，要求每个机器人必须完成特定的任务，而不仅仅是完成任意任务。因此，在任务分配中加入了任务指定性，即某些任务只能由特定的机器人来完成。在保留问题三中的约束和算法基础上，增加任务分配的合法性检查。依旧采用二分枚举时间的贪心算法进行求解，确保指定任务的强制完成并最小化整体时间步长。

## 三、模型假设

为简化问题，本文做出以下假设：

- 假设 1 机器人的停止、启动、转向等均无时间步的消耗。
- 假设 2 机器人的能耗和运行时的机械磨损不计。

## 四、符号说明

符号	说明
$T_i$	第 $i$ 个机器人完成任务所需的时间
$T_{ik}$	第 $i$ 个机器人完成任务 $k$ 所需的时间
$\tau_{ik}$	第 $i$ 个机器人前往任务 $k$ 所需的时间
$C_{xyti}$	第 $i$ 个机器人在 $t$ 时刻是否出现在位置 $(x,y)$ 上
$Z_{ik}$	第 $i$ 个机器人是否完成任务 $k$
$d_{xyti}$	第 $i$ 个机器人在时间 $t$ 在位置 $(x,y)$ 的方向状态
$G_{xy}$	在位置 $(x,y)$ 上是否有障碍物
$(x_i, y_i)$	第 $i$ 个机器人完成任务的位置 $(x,y)$
$(Sx_{P_i}, Sy_{P_i})$	第 $i$ 个机器人完成的任务 $k$ 的初始位置 $(x,y)$
$(Ex_{P_i}, Ey_{P_i})$	第 $i$ 个机器人完成的任务 $k$ 的终止位置 $(x,y)$
...	...

## 五、问题一的模型的建立和求解

### 5.1 模型建立

首先我们能够知道该模型的优化目标为最小化所有机器人完成任务的时间步长。由于所有机器人为同时发车，所以其中最小的时间步长便是最晚完成任务的机器人的时间步长。因此我们能够构建出来目标函数为：

$$\min(\max T_i)$$

对应的对于题目中给出的约束条件，我们可以构造出以下的约束：

-顶点约束：

由于在问题当中一个顶点在任意时间步长之下必定不可能有两个机器人在同一个位置之上，因此我们可以构建出以下的约束：

$$C_{xyti} \neq C_{xytj}$$

其中  $C_{xyti}$  表示一个二元变量，只有在  $t$  时刻  $(x,y)$  这个坐标上面的有机器人  $i$  的存在该二元变量才为 1，否则则为 0。该约束对于同一时刻同一坐标进行了约束，使得在该坐标之上只有一个机器人的存在，不可能有两个机器人。

-边约束：

该约束旨在约束两个机器人在相邻的时候不能碰撞，即在相邻的时候将其理解成无向图当中的相邻的两个节点，在这个边上面不能够同时通过两个机器人。

首先我们需要定义机器人的移动方向状态变量  $d_{xyti}$ ，表示的是第  $i$  辆在第  $t$  时刻在  $(x, y)$  的方向状态。我们利用  $d_{xyti}(1, 0)$ 、 $d_{xyti}(-1, 0)$ 、 $d_{xyti}(0, 1)$ 、 $d_{xyti}(0, -1)$ 、 $d_{xyti}(0, 0)$  表示机器人的五种动作状态，分别将上下左右移动状态赋值为 2、-2、1、-1。因此该约束可以写成：

$$\begin{cases} d_{xyti} - d_{(x+1)yti} = 4 \\ d_{xyti} - d_{x(y+1)ti} = 2 \\ d_{xyti} - d_{(x-1)yti} = -4 \\ d_{xyti} - d_{x(y-1)ti} = -2 \end{cases}$$

#### -边界约束

该约束旨在要求机器人在移动的过程当中不可以超出地图边界，我们假设地图边界长度为  $E$ 。该约束可以写成如下的形式：

$$1 < x < E, 1 < y < E$$

#### -障碍物约束

该约束要求所有机器人在移动的过程当中不能够与障碍物产生碰撞。设二元变量  $G_{xy}$ ，在  $(x, y)$  坐标上面有障碍物为 1，没有障碍物为 0，因此该约束可以写成：

$$C_{xyT_i i} \neq G_{xy}$$

#### -时间约束

在机器人移动的过程当中我们的目标函数为找到最晚完成任务的那个机器人作为最短的时间，因此我们迭代的所有机器人时间  $t$  必定比目标函数中的  $T_i$  要来的小，并且要保证最晚的机器人到达目的地。设对于第  $i$  辆车，其任务的终点位置可以表示为  $\square x_i, y_i \square$ ，因此该约束可以写成：

$$C_{x_i y_i T_i i} - C_{x_i y_i (T_i - 1) i} = 1, 0 \leq t < T_i < M$$

假设其中机器人的数目为  $n$ ，综上所述我们可以将模型写成：

$$\min(\max T_i)$$

$$\begin{cases} C_{xyti} \neq C_{xytj}, 0 < i < n \\ \begin{cases} d_{xyti} - d_{(x+1)yti} = 4, 0 < i < n \\ d_{xyti} - d_{x(y+1)ti} = 2, 0 < i < n \\ d_{xyti} - d_{(x-1)yti} = -4, 0 < i < n \\ d_{xyti} - d_{x(y-1)ti} = -2, 0 < i < n \end{cases} \\ 1 < x < E, 1 < y < E \\ C_{xyT_i i} \neq G_{xy}, 0 < i < n \\ C_{x_i y_i T_i i} - C_{x_i y_i (T_i - 1) i} = 1, 0 \leq t < T_i < M \end{cases}$$

## 5.2 模型求解

在模型求解方面我们利用带时空维度的 A\* 算法去进行问题求解，在基础 A\* 算法的基础之上加入时间维度去防止机器人的碰撞。并且加入机器人可以停止等待的情况，使得机器人的移动方式更多元化从而有可能更快、更优的找到最优解。在时空 A\* 算法当中每一个机器人的状态由一个三元组  $(x, y, t)$  组成，其中  $x$  和  $y$  是空间坐标， $t$  是时间步。每个状态不仅描述了位置，还描述了路径在该位置所处的时间。

我们在时空 A\* 算法中定义  $g(x, y, t)$ ，其表示累积路径成本，包括移动到该位置所花费的时间和空间的代价。起点的初始值为  $g(x_{start}, y_{start}, 0) = 0$ 。定义  $h(x, y, t)$ ，作为启发式估计函数，估计时空状态  $(x, y, t)$  转移到目标状态  $(x_{goal}, y_{goal}, t')$  的估计成本。同时定义一个评估函数：

$$f(x, y, t) = g(x, y, t) + h(x, y, t)$$

其中得分用于评估路径的优先级，用于引导算法优先搜索可能达到目标的路径。同时在得到规划路径过后我们进行全局的搜索，判断是否有冲突的产生。下面我们给出时空 A\* 算法的伪代码

---

**Algorithm 1** 时空 A\* 算法

---

```
1: 输入: grid, start, goal, other_paths
2: 初始化: neighbors, close_set, came_from, gscore, fscore, open_set
3: while open_set  $\neq \emptyset$  do
4:   current  $\leftarrow$  取出 f 值最小的点
5:   if current = goal then
6:     return 重建路径和时间
7:   end if
8:   将 current 加入 close_set
9:   for 邻居  $(i, j)$  do
10:    neighbor  $\leftarrow (current_x + i, current_y + j, t + 1)$ 
11:    if neighbor 合法且无冲突 then
12:      计算并更新 gscore, fscore
13:      将 neighbor 加入 open_set
14:    end if
15:  end for
16: end while
17: return (False,  $\infty$ )
```

---

通过时空 A\* 算法我们能够得到考虑构建起来的顶点约束、边约束、边界约束、障

障碍物约束、时间约束的时候，在当中不同地图尺度下不同机器人数目的最优路径，保存在附件文件当中

### 5.3 求解结果

通过时空 A\* 算法我们得到 8\*8 地图当中最小的时间步长为 9，16\*16 地图当中最小的时间步长为 21，64\*64 地图时间步长为 119。

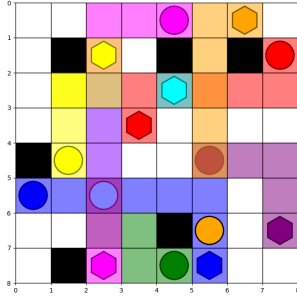
对于时空 A\* 算法的时间复杂度，我们通过分析算法当中具体的如何搜索状态空间得出。首先我们能够知道地图的大小为  $n$ ，那么每个机器人去搜索可能走的路径的复杂度为  $n^2$ ，同时机器人需要去判断所走的路径是否为可能的障碍物的复杂度为  $n^2$ 。由于有  $m$  辆车子，那么我们能够得到其时间复杂度应该为  $O(m * n^4)$ 。

下面我们给出部分具体的规划路径表格：

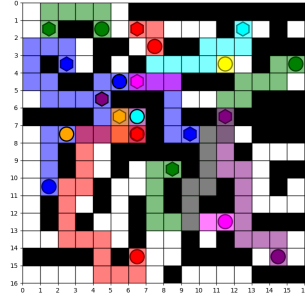
**表 1 8\*8 地图不同机器人路径与时间消耗**

机器人编号	路径	时间消耗（步）
机器人 1	[(1, 7), (2, 7), (2, 6), (2, 5), (2, 4), (2, 3), (3, 3)]	6
机器人 2	[(5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (6, 5), (7, 5)]	7
机器人 3	[(7, 4), (7, 3), (6, 3), (6, 2), (5, 2)]	4
机器人 4	[(4, 5), (4, 6), (4, 7), (5, 7), (6, 7)]	4
机器人 5	[(6, 5), (5, 5), (4, 5), (3, 5), (2, 5), (1, 5), (0, 5), (0, 6)]	7
机器人 6	[(5, 2), (4, 2), (3, 2), (2, 2), (2, 3), (2, 4)]	5
机器人 7	[(0, 4), (0, 3), (0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (6, 2), (7, 2)]	9
机器人 8	[(4, 1), (3, 1), (2, 1), (2, 1), (2, 1), (2, 2), (1, 2)]	6

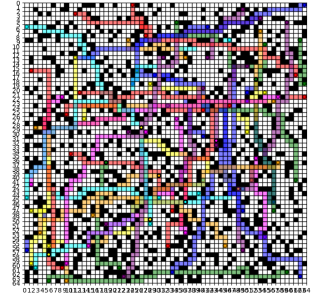
其余的 16\*16 地图和 64\*64 地图的具体路径在附件中给出。下面我们给出不同地图下面的机器人路径可视化



(a) 8\*8 地图机器人路径图片



(b) 16\*16 地图机器人路径图片



(c) 64\*64 地图机器人路径图片

图 1 总体的说明

## 六、问题二的模型的建立和求解

### 6.1 模型建立

在问题一的基础上，现在每个机器人完成任务后都停留在目的地不消失，即需要根据机器人完成任务的时间更新障碍物，因此我们可以更新构造出约束：

-障碍物更新约束：

该约束要求当所有机器人到达终点后，该点将始终判断该机器人停留在原地直至遍历结束，因此该约束可以写成：

$$C_{x_i y_i t_i} = C_{x_i y_i (t+1) i}, t \geq T_i$$

由此可以将问题一的模型更新为：

■ 针对情况一

$$\begin{aligned}
 s.t. = \quad & Z = \sum_{t,i,k} \left( \text{Price}_{ik} \cdot Z_{tik} - \sum_j \text{Cost}_{ijk} \cdot X_{tijk} \right) \\
 & \sum_k X_{tijk} \leq S_j \quad \forall t, i, j \\
 & Z_{tik} \leq \sum_j \text{Produce}_{ijk} \cdot X_{tijk} \quad \forall t, i, k \\
 & Z_{tik} \leq \text{Request}_{ik} \quad \forall t, i, k \\
 & X_{t,0,j,k} + X_{t+1,0,j,k} \leq S_j \quad \forall t, j, k \in \{1, 2, \dots, 15\} \\
 & Y_{t0jk} \cdot Y_{t1jk} = 0 \quad \forall t, k \in \{17, 18, \dots, 37\}, T_j = 6 \\
 & Y_{t0jk} \cdot Y_{(t-1)1jk} = 0 \quad \forall t, k \in \{17, 18, \dots, 37\}, T_j = 6 \\
 & Y_{t1jk} \cdot Y_{(t+1)jk} = 0 \quad \forall t, k \in \{17, 18, \dots, 37\}, T_j = 6
 \end{aligned}$$



$$\begin{aligned}
& \sum_{t=t_0}^{t_0+2} \sum_i \sum_k I_k \cdot X_{tijk} \geq S_j \quad \forall j \\
& Y_{tijk} = 1 \quad \forall t, i, k \in \{1, 2, \dots, 15\}, T_j \in \{1, 2, 3\} \\
& Y_{tijk} = 1 \quad \forall t, i, k = 16, T_j = 4 \\
& Y_{t016k} \cdot Y_{t0jk} = 0 \quad \forall t, k \in \{17, \dots, 37\}, T_j = 4 \\
& Y_{t0jk} = 0 \quad \forall t, k \notin \{1, 2, \dots, 15\}, T_j \in \{1, 2, 3\} \\
& Y_{t0jk} = 0 \quad \forall t, i \neq 16, T_j = 4 \\
& Y_{t0jk} = 0 \quad \forall t, i \notin \{17, 18, \dots, 34\}, T_j = 4 \\
& Y_{t1jk} = 0 \quad \forall t, i \notin \{35, 36, 37\}, T_j = 4 \\
& Y_{t0jk} = 0 \quad \forall t, k \notin \{17, 18, \dots, 34\}, T_j = 5 \\
& Y_{t1jk} = 0 \quad \forall t, k \notin \{38, 39, \dots, 41\}, T_j = 5 \\
& Y_{tijk} = 0 \quad \forall t, i, k \notin \{17, 18, \dots, 34\}, T_j = 6 \\
& \sum_k Y_{tijk} \leq p \quad \forall t, i, j \\
& \sum_j Y_{tijk} \leq q \quad \forall t, i, k
\end{aligned}$$

## 6.2 模型求解

对于问题二当中要求的机器人到达终点过后的成为障碍物，并且其余机器人不能够碰撞到该机器人约束，我们使用全局时空 A\* 算法去进行求解。在全局时空 A\* 算法当中我们更加注重到对于机器人在到达终点的时间步长之前是否有碰到其余已经成为障碍物的机器人，对于全局进行统筹考虑。下面我们给出全局 A\* 算法的伪代码：

## 6.3 求解结果

通过全局 A\* 算法统筹考虑机器人到达终点变成障碍物约束，我们能够正确的求解得到问题二的模型。

由于在问题二当中并没有重构整个时间 A\* 算法模型所以每个机器人去搜索可能走的路径的复杂度依旧为  $n^2$ ，同时机器人需要去判断所走的路径是否为可能的障碍物的复杂度为  $n^2$ 。由于我们假设有  $m$  辆车，那么我们能够得到其时间复杂度应该为  $O(m * n^4)$ 。

下面给出部分具体的规划路径表格

---

**Algorithm 2** 全局时空 A\* 算法

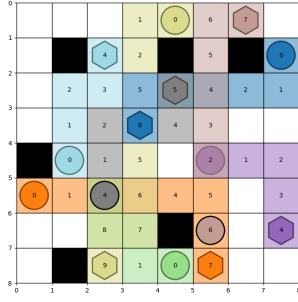
---

```
1: 输入:  $grid, start, goal, other\_paths$ 
2: 初始化:  $neighbors, close\_set, came\_from, gscore, fscore, open\_set$ 
3: while  $open\_set \neq \emptyset$  do
4:    $current \leftarrow$  取出最小  $f$  值的节点
5:   if  $current = goal$  then
6:     return 重建路径和时间
7:   end if
8:   将  $current$  加入  $close\_set$ 
9:   for 邻居  $(i, j)$  do
10:     $neighbor \leftarrow (current_x + i, current_y + j, current_t + 1)$ 
11:    if  $neighbor$  合法且无冲突 then
12:      更新  $gscore, fscore$  并将  $neighbor$  加入  $open\_set$ 
13:    end if
14:  end for
15: end while
```

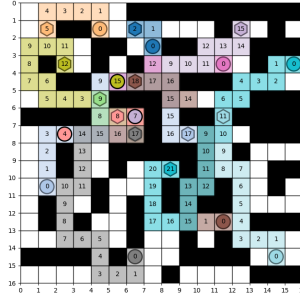
---

**表 2** 不同机器人路径与时间消耗

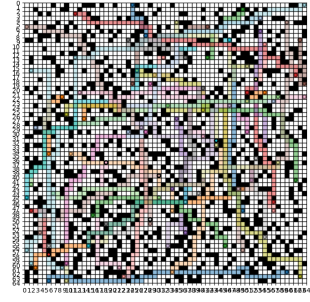
机器人编号	路径	时间消耗 (步)
机器人 1	[(1, 7), (2, 7), (2, 6), (2, 5), (2, 4), (2, 3), (3, 3)]	6
机器人 2	[(5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (6, 5), (7, 5)]	7
机器人 3	[(7, 4), (7, 3), (6, 3), (6, 2), (5, 2)]	4
机器人 4	[(4, 5), (4, 6), (4, 7), (5, 7), (6, 7)]	4
机器人 5	[(6, 5), (5, 5), (4, 5), (3, 5), (2, 5), (1, 5), (0, 5), (0, 6)]	7
机器人 6	[(5, 2), (4, 2), (3, 2), (3, 3), (3, 4), (2, 4)]	5
机器人 7	[(0, 4), (0, 3), (1, 3), (2, 3), (3, 3), (4, 3), (5, 3)]	6



(a) 8\*8 地图机器人路径图片



(b) 16\*16 地图机器人路径图片



(c) 64\*64 地图机器人路径图片

图 2 总体的说明

## 七、问题三的模型的建立和求解

### 7.1 模型建立

问题三考虑任务数多于机器人数量，因此我们需要引入一个新的“01”变量来判断任务集的完成情况，首先给出构建出来的目标函数：

$$\min(\max Z_{ik}(T_{ik} + \tau_{ik}))$$

其中  $Z_{ik}$  表示第  $i$  个机器人对任务  $k$  的完成情况， $Z_{ik} = 1$  表示完成， $Z_{ik} = 0$  表示未完成。 $T_{ik}$  为机器人  $i$  完成第  $k$  个任务所需的时间， $\tau_{ik}$  为机器人  $i$  前往第  $k$  个任务所需的路程消耗的时间。

方便起见，针对任务集  $Q$ ，我们给出两个角度进行诠释，对于所有任务， $Q = \{1, \dots, k, \dots, m\}$ ；对于每一个机器人完成的任务，我们设第  $i$  个机器人完成  $P_i$  个任务， $P_i = \{1, 2, \dots, m_i\}$ ，其中  $m_i$  为第  $i$  个机器人完成的最后一个任务。

#### -任务迭代约束

由此我们对模型进行修改，保留顶点约束、边约束、边界约束和障碍物约束不变，我们考虑任务迭代约束，因此我们可以设第  $i$  个机器人完成的  $P_i$  任务的起始位置的横纵坐标分别为  $Sx_{P_i}$  和  $Sy_{P_i}$ ，终止位置的横纵坐标分别为  $Ex_{P_i}$  和  $Ey_{P_i}$ 。由于我们在模型求解所需的任务距离为曼哈顿距离，因此我们可以给出任务迭代的约束为：

$$\begin{cases} (Sx_{P_i} - Ex_{P_i})^2 + (Sy_{P_i} - Ey_{P_i})^2 = distance \\ (Sx_{P_{i+1}} - Ex_{P_i})^2 - (Sy_{P_{i+1}} - Ey_{P_i})^2 \geq 0 \end{cases}$$

#### -任务完成情况约束

然后，我们对任务完成情况进行约束，即每个任务必须由且仅由一个机器人完成，该约束可以写成：

$$\begin{cases} Z_{ik} = Z_{jk}, Z_{ik} = 0 \\ Z_{ik} \neq Z_{jk}, Z_{ik} = 1 \end{cases}$$

-任务完成时间约束

对于任务完成的时间，同样地，在机器人移动的过程当中我们的目标函数为找到最晚完成任务的那个机器人作为最短的时间，因此我们迭代的所有机器人时间  $t$  必定比目标函数中的  $T_i$  要来的小，由此我们可以记录第  $i$  个机器人完成任务  $k$  的时间  $T_{ik}$  和第  $i$  个机器人前往任务  $k$  的时间  $\tau_{ik}$ 。因此我们可以给出约束：

$$\begin{cases} C_{Ex_{P_i}T_{ik}i} - C_{Ex_{P_i}(T_{ik}-1)i} = 1 \\ C_{Sx_{P_i}\tau_{ik}i} - C_{Sx_{P_i}(\tau_{ik}-1)i} = 1 \end{cases}$$

-障碍物更新约束

最后，考虑每个机器人完成任务后将留在原地，需要更新障碍物。因此我们可以给出约束：

$$C_{Ex_{m_i}Ey_{m_i}ti} = C_{Ex_{m_i}Ey_{m_i}(t+1)i}, t \leq T_{ik}$$

因此，假设所有的机器人的数目为  $n$ ，所有任务的数目为  $m$ ，综上所述我们可以将模型写成：

$$\begin{aligned} & \min(\max Z_{ik}(T_{ik} + \tau_{ik})) \\ & \begin{cases} C_{xyti} \neq C_{xytj}, 0 < i < n \\ \begin{cases} d_{xyti} - d_{(x+1)yti} = 4, 0 < i < n \\ d_{xyti} - d_{x(y+1)ti} = 2, 0 < i < n \\ d_{xyti} - d_{(x-1)yti} = -4, 0 < i < n \\ d_{xyti} - d_{x(y-1)ti} = -2, 0 < i < n \end{cases} \\ 1 < x < E, 1 < y < E \\ C_{xyT_i i} \neq G_{xy}, 0 < i < n \\ C_{x_i y_i T_i i} - C_{x_i y_i (T_i - 1) i} = 1, 0 \leq t < T_i < M \\ \begin{cases} (Sx_{P_i} - Ex_{P_i})^2 + (Sy_{P_i} - Ey_{P_i})^2 = distance \\ (Sx_{P_i+1} - Ex_{P_i})^2 - (Sy_{P_i+1} - Ey_{P_i})^2 \geq 0 \end{cases} \\ \begin{cases} Z_{ik} = Z_{jk}, Z_{ik} = 0 \\ Z_{ik} \neq Z_{jk}, Z_{ik} = 1 \end{cases} \\ \begin{cases} C_{Ex_{P_i}T_{ik}i} - C_{Ex_{P_i}(T_{ik}-1)i} = 1 \\ C_{Sx_{P_i}\tau_{ik}i} - C_{Sx_{P_i}(\tau_{ik}-1)i} = 1 \end{cases} \\ C_{Ex_{m_i}Ey_{m_i}ti} = C_{Ex_{m_i}Ey_{m_i}(t+1)i}, t \leq T_{ik} \end{cases} \end{aligned}$$

## 7.2 模型求解

在模型求解方面，我们依旧想利用带时空维度的  $A^*$  算法进行求解，同时为使少于任务数的机器人能尽快找到相对较优的完成任务的路径，我们引入贪心算法：

考虑贪心解，贪心策略即每个机器人去完成通过时空  $A^*$  搜索所有任务里开始位置距离当前机器人位置最近且未被访问过的任务，同时第三问中要考虑任务的合法性，即任务被分配给指定机器人，考虑到机器人完成的任务数量和时间呈单调关系，我们通过二分时间来求解贪心解，在二分枚举的每一个时间里，为每个机器人按贪心策略分配任务直至完成任务的总时间将要大于二分枚举的时间，这样即可求得贪心解。

同时观察到时空  $A^*$  的运行在不同机器人之间是串行的，即让后枚举的机器人的路径不与已经枚举的机器人的路径碰撞，因此最终的贪心解必然与枚举机器人的顺序有关，考虑到机器人的数量较少我们通过暴力枚举获得所有机器人的枚举顺序并以上述方法求解以得到最终时间最小的贪心解。

基于此我们给出贪心算法的伪代码：

---

**Algorithm 3** 贪心算法伪代码

---

```
Check $x$  初始化 paths, paths2, vis, cnt 为 0 for ii 从 0 到 len(a)-1 do
  i1  $\leftarrow$  a[ii], agent  $\leftarrow$  agents[i1]
  初始化 start1, sumt, other_paths, minw  $\leftarrow$  大值
  for i2, task in tasks do
    if task 未访问且不在其他代理的必须任务中 then
      使用 A* 计算 start1 到 task 的路径 path, t
      if t 小于 minw then
        更新任务并记录路径
      end if
    end if
  end for
  if 找到有效任务 then
    更新 paths, vis, cnt++
    while cnt 小于 len(tasks) 且 sumt  $\leq x$  do
      重复任务分配过程
    end while
  end if
end for
cnt == len(tasks), paths, paths2
Solve
l, r  $\leftarrow$  0, 10000
while l < r do
  mid  $\leftarrow$  (l + r) // 2
  if Checkmid then
    r  $\leftarrow$  mid
  else
    l  $\leftarrow$  mid + 1
  end if
end while
输出 Checkl
```

---

### 7.3 求解结果

通过基于贪心思想的全局 A\* 算法统筹考虑机器人到达终点变成障碍物约束，我们能够正确的求解得到问题三的模式。8\*8 地图当中最小的时间步长为 68，16\*16 地图当

中最小的时间步长为 237，64\*64 地图最小时间步长为 596。

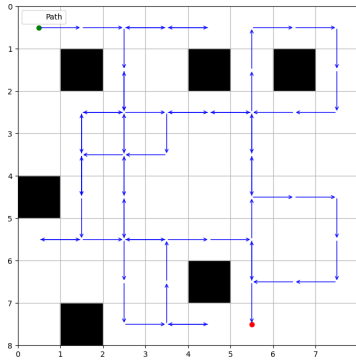
在问题三当中我们加入了二分枚举时间的贪心算法去完成通过时空 A\* 算法搜索所有任务里开始位置距离当前机器人位置最近且未被访问过的任务，因此我们的时间复杂度上面我们需要加入贪心算法的时间复杂度  $O(\log_2 M)$ ，因此整个问题三的时间复杂度可以写成  $O(m * n^4 \log_2 M)$

下面给出 8\*8 网格图的部分规划路径表格：

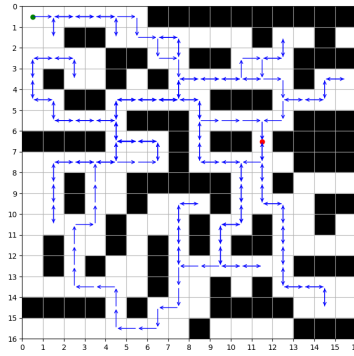
**表 3 不同机器人路径与时间消耗**

机器人编号	任务完成顺序（仅给出任务开始位置）	时间消耗（步）
机器人 1	[(0,4),(7,4),(5,2),(4,5),(6,5),(1,7),(4,1),(5,0)]	68

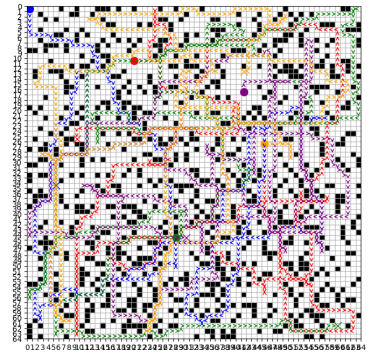
其余的 16\*16 地图和 64\*64 地图的具体路径在附件中给出。下面我们给出不同地图下面的机器人路径可视化



(a) 8\*8 地图机器人路径图片



(b) 16\*16 地图机器人路径图片



(c) 64\*64 地图机器人路径图片

**图 3 总体的说明**

## 八、问题四的模型的建立和求解

### 8.1 模型建立

问题四考虑到部分任务已经被指定的机器人来完成，因此我们只需要修改对于第  $i$  个机器人完成任务的集合为  $P_i = \{1, 2, \dots, m_i, M_i\}$ ，其中， $M_i$  表示分配给第  $i$  个机器人的任务。

因此，我们可以将模型修改为：

$$\min(\max Z_{ik}(T_{ik} + \tau_{ik}))$$

$$\left\{ \begin{array}{l} C_{xyti} \neq C_{xytj}, 0 < i < n \\ \left\{ \begin{array}{l} d_{xyti} - d_{(x+1)yti} = 4, 0 < i < n \\ d_{xyti} - d_{x(y+1)ti} = 2, 0 < i < n \\ d_{xyti} - d_{(x-1)yti} = -4, 0 < i < n \\ d_{xyti} - d_{x(y-1)ti} = -2, 0 < i < n \end{array} \right. \\ 1 < x < E, 1 < y < E \\ C_{xyT_i i} \neq G_{xy}, 0 < i < n \\ C_{x_i y_i T_i i} - C_{x_i y_i (T_i - 1) i} = 1, 0 \leq t < T_i < M \\ \left\{ \begin{array}{l} (Sx_{P_i} - Ex_{P_i})^2 + (Sy_{P_i} - Ey_{P_i})^2 = distance \\ (Sx_{P_{i+1}} - Ex_{P_i})^2 - (Sy_{P_{i+1}} - Ey_{P_i})^2 \geq 0 \end{array} \right. \\ \left\{ \begin{array}{l} Z_{ik} = Z_{jk}, Z_{ik} = 0 \\ Z_{ik} \neq Z_{jk}, Z_{ik} = 1 \end{array} \right. \\ \left\{ \begin{array}{l} C_{Ex_{P_i} T_{ik} i} - C_{Ex_{P_i} (T_{ik} - 1) i} = 1 \\ C_{Sx_{P_i} \tau_{ik} i} - C_{Sx_{P_i} (\tau_{ik} - 1) i} = 1 \end{array} \right. \\ C_{Ex_{m_i} Ey_{m_i} ti} = C_{Ex_{m_i} Ey_{m_i} (t+1) i}, t \leq T_{ik} \end{array} \right.$$

## 8.2 模型求解

对于问题四我们同样利用二分枚举时间的贪心算法去分配不同机器人之间的任务调度，其中单个机器人的循迹以及利用时空 A\* 算法去进行计算。对于问题四当中给出的要求每个机器人有固定的任务序列来说，我们去进行贪心搜索最短距离且没有访问过的过程当中还要考虑到问题的任务的合法性，即任务被分配给指定机器人。即我们需要去进行任务的判断，该任务是否有分配给其他的机器人。通过考虑问题合法性的二分枚举时间贪心算法我们能够求解问题四的模型并得出最短时间和路径。

## 8.3 求解结果

8\*8 地图当中最小的时间步长为 42，16\*16 地图当中最小的时间步长为 120，64\*64 地图最小时间步长为 616。

在问题四当中的时间复杂度与问题三当中的时间复杂度是相同的都可以写成  $O(m * n^4 \log_2 M)$

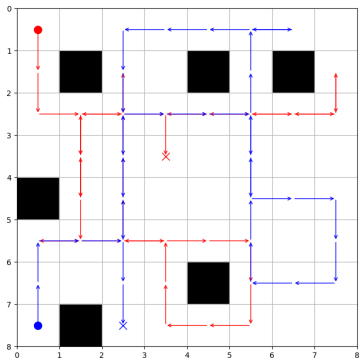
下面给出 8\*8 网格图的部分规划路径表格：



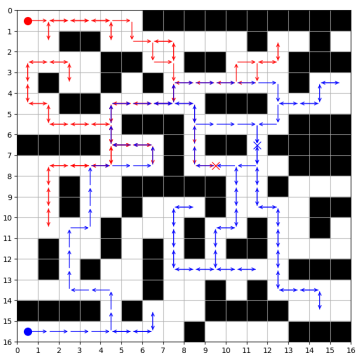
表 4 不同机器人路径与时间消耗

机器人编号	任务完成顺序	时间消耗（步）
机器人 1	[(0,0),(4,1),(1,2),(5,0),(7,5),(7,4),(5,2),(1,7)]	42
机器人 2	[(7,0),(5,2),(2,4),(4,5),(6,7),(6,5),(0,6),(0,4)]	36

其余的 16\*16 地图和 64\*64 地图的具体路径在附件中给出。下面我们给出不同地图下面的机器人路径可视化



(a) 8\*8 地图机器人路径图片



(b) 16\*16 地图机器人路径图片



(c) 64\*64 地图机器人路径图片

图 4 总体的说明

## 九、模型的评价

### 9.1 模型的优点

- 优点 1 模型采用时空 A\* 算法进行求解，在传统 A\* 算法的基础上可以较好地进行碰撞检测，同时具有较低的时间和空间复杂度。
- 优点 2 模型求解时采用了二分时间优化，进一步降低了算法的时间复杂度。
- 优点 3 算法求解时优化了时空 A\* 无法枚举顺序的缺点。

### 9.2 模型的缺点

- 缺点 1 由于时空 A\* 算法涉及多种参数的调整，这些参数的设置对算法的效果有较大影响。
- 缺点 2 模型求解结果为贪心解，容易陷入局部最优情况。

## 参考文献

[1] 司守奎, 孙玺菁. 数学建模算法与应用[M]. 北京: 国防工业出版社, 2011.

[2] 卓金武. MATLAB 在数学建模中的应用[M]. 北京: 北京航空航天大学出版社, 2011.

## 附录 A 文件列表

文件名	功能描述
Problem1.py	问题一程序代码
Problem2.py	问题二程序代码
Problem3.py	问题三程序代码
Problem4.py	问题四程序代码

## 附录 B 代码

Problem1.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from queue import PriorityQueue
4 from matplotlib.patches import Circle, RegularPolygon
5 import pandas as pd
6 import os
7
8 # 解析地图
9 def parse_map(map_data):
10     lines = map_data.strip().split("\n")
11     grid = []
12     for line in lines:
13         grid.append(line.split())
14     return np.array(grid)
15
16
17 # 解析任务
18 def parse_tasks(tasks_data):
19     lines = tasks_data.strip().split("\n")[1:] # 忽略任务数量
20     tasks = []
21     for line in lines:
22         start_x, start_y, end_x, end_y = map(int, line.split())
23     )
```

```

23         tasks.append(((start_x, start_y), (end_x, end_y)))
24     return tasks
25
26
27 # 启发式函数（曼哈顿距离）
28 def heuristic(a, b):
29     return abs(a[0] - b[0]) + abs(a[1] - b[1])
30
31
32 # 扩展A*算法以处理时间维度
33 def time_a_star_search(grid, start, goal, other_paths):
34     neighbors = [(0, 1), (1, 0), (0, -1), (-1, 0), (0, 0)]
35     close_set = set()
36     came_from = {}
37     gscore = {(start[0], start[1], 0): 0}
38     fscore = {(start[0], start[1], 0): heuristic(start, goal)}
39     open_set = PriorityQueue()
40     open_set.put((fscore[(start[0], start[1], 0)], (start[0],
start[1], 0)))
41
42     while not open_set.empty():
43         _, current = open_set.get()
44         x, y, t = current
45
46         if (x, y) == goal:
47             path = []
48             while current in came_from:
49                 path.append((current[0], current[1]))
50                 current = came_from[current]
51             path.append((start[0], start[1]))
52             return path[::-1], t # 返回路径和时间消耗
53
54         close_set.add(current)
55         for i, j in neighbors: # 添加等待的可能性
56             neighbor = x + i, y + j, t + 1

```

```

57         if 0 <= neighbor[0] < grid.shape[0] and 0 <=
neighbor[1] < grid.shape[1]:
58             if grid[neighbor[0]][neighbor[1]] == '@':
59                 continue
60
61             # 检查是否与其他机器人路径冲突（点冲突）
62             conflict = False
63             for path in other_paths:
64                 if len(path) > t + 1 and path[t + 1] == (
neighbor[0], neighbor[1]):
65                     conflict = True
66                     break
67             # 检查边冲突
68             if len(path) > t and len(path) > t + 1 and
path[t] == (neighbor[0], neighbor[1]) and path[t + 1] == (x
, y):
69                 conflict = True
70                 break
71             if conflict:
72                 continue
73
74             tentative_g_score = gscore[current] + 1
75
76             if neighbor in close_set and tentative_g_score
>= gscore.get(neighbor, 0):
77                 continue
78
79             if tentative_g_score < gscore.get(neighbor, 0)
or neighbor not in [i[1] for i in open_set.queue]:
80                 came_from[neighbor] = current
81                 gscore[neighbor] = tentative_g_score
82                 fscore[neighbor] = tentative_g_score +
heuristic((neighbor[0], neighbor[1]), goal)
83                 open_set.put((fscore[neighbor], neighbor))
84

```

```

85     return False, float('inf')
86
87
88 # 处理数据
89 map_data = """
90 . . . . . @ . . @ . . . @ @ @ . @ . . @ . . . @ . . @ . .
91   . . @ . . . . . . . . . . . . @ . . . . . . . . . @
92   @ . .
93 . . @ . . . . . @ @ . . @ . . . . . . . . . . . . @
94   . . . . . . . . . @ . . . @ . . . @ . @ . @ . @ . . . . .
95   . . .
96 . . . @ . @ . . . . . . . . @ . . . . . . . . @ @ . . @ .
97   . . . . @ @ . . . . . . . . . @ . . . @ . . . . . . .
98   . . .
99 . . . . . . . . . @ . . . @ @ . . . . . . . . @ . . . @ . .
    . @ . . . . @ . . . . . . . . @ . . . . . . . . @ . . .
    . . .

```

[illegible]

[illegible]



	. . @ @ . @ . . . @ . . @ . . . . @ . . @ . . . . . . . .
	. . .
124	. . . . . . . . . . . . . @ . . @ @ . . . @ . . @ . . . @ @ @
	@ . . . . . @ . . @ . . . . . . . @ . . . @ . . . . @ @ . .
	. . @
125	@ . @ . . . . . . . @ . . . @ . . @ . . . . . . . . @ .
	. @ . . . . . . . . . @ . . . . @ . . @ . . @ . @ . . . . .
	. . .
126	. @ @ @ . . @ . . @ . . . . . . . . . . . . @ . . . . .
	. @ . . . . . . . . . @ . . . @ . . . @ . @ . . @ . . @ @
	. . @
127	. @ . . . . @ . . . . . . . . . . . . @ . . . . . @ . .
	@ . . @ @ . . . . . . @ @ . . . . . . . . . . @ @
	. . .
128	@ . . . . . . @ . @ @ . @ . . @ @ . @ @ @ . . . @ . . . . .
	. @ . . . . . @ . . . . . . . . @ . . . . @ . . . . @ .
	. . .
129	. . . . . . . . @ @ @ . . . . . . . @ @ . . . . . . . . .
	. . @ . . . . . @ . . . . @ . @ . @ . . . . @ . @ . . . @ .
	. . .
130	. . . . . . . . . . . . . @ . @ . . @ @ . . . . . @ . .
	. @ . @ . @ . . @ . . . . . . . @ . . . @ . . . . . . .
	. . @
131	. . . @ . . @ . . @ . . . . . . . . @ @ . . . . . @ . . . .
	@ . . . . . . . . @ . . @ . . . @ @ @ . . . . . . . . .
	. @ .
132	. . @ . . . . . @ . . . . @ . . . . . . . . . @ . . @ . .
	. . . . @ . . . . @ . . . . . . @ . . . . . @ @ @ . . . . @
	. @ .
133	. . . . . . . . @ . . . . . @ @ . . . . . @ . . . . @ . @ . .
	. . . @ . . . . . @ . @ . . @ . . . . . . . . . . . @
	. . @
134	. . . @ . . . . . . . . @ . . . . . @ . . . . . . . . . .
	. . . . @ . @ @ . . . . . . . . @ . . . . . @ . . . . @
	. . .

135 . . @ @ @ . . . . . @ @ . . . . . @ @ . . . . .  
. . . . . @ . . @ . . . @ @ . . @ . . . . .  
@ . .

136 @ . . @ . . . . . @ . . @ . @ . . . @ . . . @ . . . .  
. . . @ . . . . . @ . . . @ . . . . . @ . . . @ . . . .  
. . .

137 . . . . . @ . @ . . . . . @ . @ @ . . . . . @ . . . . .  
@ . . @ . . . @ . . . . . @ @ @ . . . . . @ . . . . .  
@ @ .

138 . . @ . . . . @ . . . @ @ . . . . @ . @ . @ @ @ . . . . .  
. @ . . . @ . @ . . . . . @ . . . . . @ . . . . . @ . . .  
. . .

139 . . . . . . . . . . @ @ . . . . . @ @ . . . @ . . . . .  
. . @ @ . @ . . . . @ . @ . . . . @ @ . @ @ . . . . .  
@ . .

140 . . @ . . . . . @ . . . . . @ . . . . . @ . . . . .  
. . . . . @ @ . @ . . . . . @ . @ @ . . @ . . @ .  
. . .

141 . . . . . @ . . . . @ @ . . @ . . . @ . . . @ . . . . @  
. . . . . @ . . . . @ . . @ @ . . . . @ . . . @ . @ . . .  
. . .

142 . . @ . . . . . @ . . @ . . . . . . . . @ . @ @  
. . . . . @ @ . @ . . . @ . . . . . @ . . . . .  
. . @

143 @ @ . . . @ . . . . @ . @ . . @ . @ . . @ . . @ . @ . @  
. . . @ @ @ . . . @ . @ . . . . . @ . . @ . . . @ . . .  
. @ .

144 . . . . . @ . . . . . @ . . . . . @ . . @ . . . . .  
. . . . @ . @ . . . . . . . . @ . . . . @ . @ . @  
. . .

145 . . . @ . . . . @ . . . . . @ . @ . . . @ . . @ . . .  
. . @ @ @ . @ . . . @ . . . . . @ . . @ . @ . . . @ .  
. . .

146 . . . . . @ . . . . . @ . . . @ . . . @ @ . . . @ . . . @ @ .  
. @ . . . @ . . . . . . . . . . @ . . . @ . . . @ .

```

    @ @ .
147 . . . . . @ . . . . @ . @ @ . . . @ @ . @ @ . . . . .
    . @ . . . . @ . @ . @ @ @ . . @ . . . . @ . . . . @ @ . . @
    . . .
148 . . . . . @ . . . . @ @ . . . . @ @ . @ . @ . . . . @ . @ .
    @ . . @ . @ . . . . . @ . . . @ . . . . @ @ . . . . .
    . . .
149 . . . . . @ @ . . @ . . . . . . . . . @ . . . . @ . .
    . . @ . . . . . @ . . @ . . @ . . . @ . @ . @ @ @ . . . .
    . . @
150 . @ . . . . . . . . @ . . . . @ . . @ . . . . @ . . . . .
    . . . . @ @ . . @ . . . . . @ . . . . @ . . . @ . . @ . .
    . . .
151 @ . @ . . . @ . . . @ . . . @ . . . @ @ @ @ @ @ . @ . . @ @ . .
    . . . . . . . . . . . . . . . @ . @ @ . . . @ . . . .
    @ . .
152 @ . . . . . . . . . @ @ . . . . @ . . @ . . . @ . . . . .
    . . . @ @ . @ . @ . . . . @ . . @ @ . . . . . . . . .
    . . .
153 @ . @ . . . . @ . . . . . . . . . . . @ . . . . . . .
    . @ . . @ . @ @ @ . . . . . . . . . @ @ . @ . . . .
    . . .
154 ""
155
156 tasks_data = ""
157 50
158 7 27 0 24
159 14 47 47 49
160 63 5 61 59
161 3 53 29 23
162 28 2 22 56
163 42 49 60 2
164 21 8 34 5
165 26 13 20 54
166 47 33 34 10

```

167	22	50	61	33
168	23	32	40	61
169	1	49	12	41
170	55	45	48	15
171	43	6	44	30
172	41	1	21	57
173	57	5	47	1
174	16	61	2	11
175	20	50	45	56
176	9	54	47	56
177	12	31	62	22
178	40	51	6	53
179	40	40	42	33
180	52	36	26	34
181	42	48	28	26
182	55	32	23	46
183	5	41	18	24
184	8	62	18	63
185	56	49	4	59
186	39	30	52	3
187	5	0	50	28
188	29	27	59	9
189	29	50	18	28
190	21	63	20	12
191	16	29	30	40
192	36	18	60	21
193	50	18	19	40
194	12	36	8	23
195	45	36	43	39
196	35	52	53	54
197	49	28	55	5
198	24	40	25	9
199	15	25	62	62
200	17	52	4	34
201	32	43	14	50

```

202 63 9 36 57
203 8 46 44 0
204 31 43 55 14
205 59 1 12 11
206 15 1 60 8
207 56 0 0 63
208 """
209
210 # 计算每个机器人的路径
211 grid = parse_map(map_data)
212 tasks = parse_tasks(tasks_data)
213
214 paths = []
215 total_time = 0
216 total_steps = 0 # 用于计算复杂度
217 branching_factor = 5 # 在每个节点处，最大可扩展5个方向（上、
    下、左、右、等待）
218
219 for i, task in enumerate(tasks):
220     start, goal = task
221     other_paths = [p[0] for p in paths] # 其他机器人路径
222     path, time = time_a_star_search(grid, start, goal,
    other_paths)
223     paths.append((path, time))
224     total_time = max(total_time, time)
225     total_steps += len(path)
226
227 # 计算算法复杂度
228 grid_size = grid.shape[0] * grid.shape[1]
229 estimated_complexity = branching_factor ** (total_steps / len(
    tasks))
230
231 # 绘图
232 fig, ax = plt.subplots(figsize=(8, 8))
233 colors = plt.cm.get_cmap('tab20', len(tasks)).colors # 使用更

```

多颜色

```
234 step_map = np.full((grid.shape[0], grid.shape[1]), -1)
235 color_map_idx = np.full((grid.shape[0], grid.shape[1]), -1)
236
237 # 绘制障碍物
238 for i in range(grid.shape[0]):
239     for j in range(grid.shape[1]):
240         if grid[i, j] == '@':
241             ax.add_patch(plt.Rectangle((j, i), 1, 1, color='
gray'))
242
243 # 绘制路径
244 for idx, (path, _) in enumerate(paths):
245     color = colors[idx % len(colors)]
246     if path:
247         # 起点
248         start_x, start_y = path[0]
249         ax.add_patch(Circle((start_y + 0.5, start_x + 0.5),
0.4, edgecolor='black', facecolor=color, lw=2))
250
251         # 终点
252         end_x, end_y = path[-1]
253         ax.add_patch(
254             RegularPolygon((end_y + 0.5, end_x + 0.5),
numVertices=6, radius=0.4, edgecolor='black', facecolor=
color,
255                             lw=2))
256
257 # 路径及时间标记
258 for t, (x, y) in enumerate(path):
259     if step_map[x, y] < t + 1:
260         step_map[x, y] = t + 1
261         color_map_idx[x, y] = idx
262         ax.add_patch(plt.Rectangle((y, x), 1, 1, color
=color, alpha=0.5))
```

```

263
264 # 在方块上标记时间
265 for i in range(step_map.shape[0]):
266     for j in range(step_map.shape[1]):
267         if step_map[i, j] > 0:
268             ax.text(j + 0.5, i + 0.5, str(step_map[i, j]-1),
269                     ha='center', va='center', color='black')
270
271 ax.set_xticks(np.arange(0, grid.shape[1] + 1, 1))
272 ax.set_yticks(np.arange(0, grid.shape[0] + 1, 1))
273 ax.grid(which='both', color='black', linestyle='-', linewidth
274         =1)
275 plt.xlim(0, grid.shape[1])
276 plt.ylim(0, grid.shape[0])
277 plt.gca().invert_yaxis() # 上下翻转y轴以匹配要求
278 plt.show()
279
280 # 输出结果
281 print(f"总运输时间: {total_time}")
282 print(f"估计的算法复杂度: O({estimated_complexity:.2e})")
283
284 for i, (path, time) in enumerate(paths, 1):
285     print(f"机器人{i}: 路径 {path}, 时间消耗 {time}步")
286
287 if os.path.exists("results_第一问 64.xlsx"):
288     # 清空文件内容
289     with open("results_第一问 64.xlsx", 'w') as file:
290         pass # 使用 pass 清空文件内容
291
292 data = {
293     "机器人编号": [],
294     "位置列表": [],
295     "时间开销": []
296 }
297
298 for i, (path, time) in enumerate(paths, 1):

```

```

296     data["机器人编号"].append(i)
297     data["位置列表"].append(path)
298     data["时间开销"].append(time)
299
300 df = pd.DataFrame(data)
301
302 # 导出到Excel文件
303 df.to_excel('results_第一问 64.xlsx', index=False)

```

Problem2.py

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  from queue import PriorityQueue
4  from matplotlib.patches import Circle, RegularPolygon
5  import pandas as pd
6  import os
7
8  # 解析地图
9  def parse_map(map_data):
10     lines = map_data.strip().split("\n")
11     grid = []
12     for line in lines:
13         grid.append(line.split())
14     return np.array(grid)
15
16 # 解析任务
17 def parse_tasks(tasks_data):
18     lines = tasks_data.strip().split("\n")[1:] # 忽略任务数量
19     # 行
20     tasks = []
21     for line in lines:
22         start_x, start_y, end_x, end_y = map(int, line.split()
23         )
24         tasks.append(((start_x, start_y), (end_x, end_y)))
25     return tasks

```



```

24
25
26 # 启发式函数 (曼哈顿距离)
27 def heuristic(a, b):
28     return abs(a[0] - b[0]) + abs(a[1] - b[1])
29
30 vis = [[0 for _ in range(100)] for _ in range(100)]
31
32 def time_a_star_search(grid, start, goal, other_paths):
33     neighbors = [(0, 1), (1, 0), (0, -1), (-1, 0), (0, 0)]
34     close_set = set()
35     came_from = {}
36     gscore = {(start[0], start[1], 0): 0}
37     fscore = {(start[0], start[1], 0): heuristic(start, goal)}
38     open_set = PriorityQueue()
39     open_set.put((fscore[(start[0], start[1], 0)], (start[0],
40 start[1], 0)))
41
42     while not open_set.empty():
43         _, current = open_set.get()
44         x, y, t = current
45
46         if (x, y) == goal:
47             path = []
48             while current in came_from:
49                 path.append((current[0], current[1]))
50                 current = came_from[current]
51             path.append((start[0], start[1]))
52             return path[::-1], t # 返回路径和时间消耗
53
54         close_set.add(current)
55         for i, j in neighbors: # 添加等待的可能性
56             neighbor = x + i, y + j, t + 1
57             if 0 <= neighbor[0] < grid.shape[0] and 0 <=
neighbor[1] < grid.shape[1]:

```

```

57         if grid[neighbor[0]][neighbor[1]] == '@':
58             continue
59
60         # 检查是否与其他机器人路径冲突（点冲突）
61         conflict = False
62         for path in other_paths:
63             if (neighbor[0], neighbor[1]) == goal:
64                 for i in range(0, len(path)):
65                     if(path[i] == goal and i >= t + 1)
:
66                     conflict = True
67                     break
68             if conflict:
69                 break
70             if len(path) > t + 1 and path[t + 1] == (
neighbor[0], neighbor[1]):
71                 conflict = True
72                 break
73             if len(path) <= t + 1 and path[-1] == (
neighbor[0], neighbor[1]):
74                 conflict = True
75                 break
76             # 检查边冲突
77             if len(path) > t and len(path) > t + 1 and
path[t] == (neighbor[0], neighbor[1]) and path[t + 1] == (x
, y):
78                 conflict = True
79                 break
80             if conflict:
81                 continue
82
83             tentative_g_score = gscore[current] + 1
84
85             if neighbor in close_set and tentative_g_score
>= gscore.get(neighbor, 0):

```

```

86         continue
87
88         if tentative_g_score < gscore.get(neighbor, 0)
or neighbor not in [i[1] for i in open_set.queue]:
89             came_from[neighbor] = current
90             gscore[neighbor] = tentative_g_score
91             fscore[neighbor] = tentative_g_score +
heuristic((neighbor[0], neighbor[1]), goal)
92             open_set.put((fscore[neighbor], neighbor))
93
94     return False, float('inf')
95
96
97 # 处理数据
98 map_data = """
99 . . . . . @ . . @ . . . @ @ @ . @ . . @ . . . . @ . . @ . .
100 . . @ . . . . . @ @ . . @ . . . . . . . . . . . . . . @
    . . . . . . . . . . . . . . . . @ . . . . . . . . . @
    @ . .
101 . . . @ . @ . . . . . . . . @ . . . . . . . . . @ @ . . @ .
    . . . . @ @ . . . . . . @ . . . @ . . . @ . . . . . . .
    . . .
102 . . . . . . . . . . . . . . @ @ . . . . @ . @ . . . @ . . .
    . . . . . . . . . @ . . . @ . . . . . @ . . . . . @ @ @ . .
    . . @
103 . . @ . . . . @ . . @ . . . . . . . . . . . . . @ . . .
    . . . . @ @ @ . . . . . . @ . . . . . . @ . . . . . .
    @ @ @
104 . . . . . @ @ . . @ . @ . . . . @ . . . . . . . . . @
    . @ . . . @ . . . . . @ . . . . . @ @ . @ . . . . . .
    . . .
105 . . @ . . . . . @ . . . . @ . . @ . @ . @ . . . @ @ . . @
    . . . . @ . . . . . . . . . @ @ @ . . @ . . @ . . @ . .

```

	. . @
106	. . . @ @ @ . @ . . . . . @ . @ . . . . . @ . . . @ @ . . . @ . . . . . . . . . . . . @ . @ . . . . . @ . . . . @ . . . .
	. . @
107	@ @ . @ . . . @ . . . . . @ . . . . . @ . . . . . @ . . . . . . . . . . . . . @ @ @ . . @ . . . . @ . . . . . @ . . . . .
	. . .
108	. . . . . . . . . @ . . . @ @ . . . . . . . @ . . . @ . . . . @ . . . . @ . . . . . . . . @ . . . . . . . . @ . . . .
	. . .
109	. . . . @ . . . . . . . @ . . . @ . . . . . . . . . . . @ . @ . . . @
	@ . .
110	. . . . . . . . @ . @ @ . . @ . . . . . . . . . . . @ @ . @ @ @ . . . @ . . . . @ . . @ . . @ @ @ . @ . . @ @ . @ . . @ . @
111	. @ . . @ @ @ . . . . . . . . . . . @ . . . . . . . . . . . . . @ . . . @ . . . . . . . . . . . @ @ @ @ . . . . . @ . . @ . .
112	. @ . . . . @ . . . @ . . . . . @ @ . . . . . @ . . . . @ . . . @ . . . . @ . . . . . . . . . @ . . @ @ @ . . . . . @ @ . . . @
113	. . . . . . @ . . . . . . . . . . . @ . . . @ . . . . . . . . . . . @ @ .
	. . .
114	. . . . . . @ @ . . . . . @ . . . . . @ . . @ . . @ . . @ @ . . . . . @ . @ . . . . . . . @ . . . . . @ . . @ @ @ @ . . . . . @ .
115	. . . . . . @ . . . . . . . . . . . @ . @ . . @ . @ . . . . . . . . @ . . . . . . . . . . . @ . . . . . . . . @ . . . @ . . . . .
116	. . . @ . . . . . . . . . . . @ . . . . . . . . . . . . . . . @ . . . . @ . @ . . . . . . @ . . . @ . . . . . . . . . . . . . @
117	. . . . . . . . . . . . . @ @ . . @ @ . . . . . . . @ . . @ .

	. . . . . @ . . . @ . . . @ . .
118	@ . @ . . . @ . . @ . . @ . @ . . @ . @ @ . . . . . . . . . . @ @ . . . . . @
	@ . .
119	. @ . . . . @ . . . . . @ . @ @ . . . . . . @ . . @ @ . . . . @ . @ . . . . @ . . . . @ . . . .
	. . .
120	. . . @ . . @ . . . @ . . @ . . . @ . . @ . . . @ . . . . . . . . . @ . @ . @ . . . @ . . @ . . . @ . . .
	. . .
121	. . . . . @ @ . . . . . @ . . . . @ . . . . . . . . @ @ .
	. . .
122	. . @ . . . @ . . . . . . . . . @ . . . . . . . . . . . . . . . @ . . . . @ . . . . @ @ . . . . . . . . . .
	. . .
123	. . . . @ . @ . . . . . @ . . @ . @ . . . . . . . . . . @ . @ . . . . .
	@ @ .
124	. @ . . . . @ @ . . . . . . . . . @ @ . @ . @ . @ . @ . @ @ . . . @ . . @ . @ . . . . . @ @ . . . . . . . . . @
	. . .
125	. . . @ . . . @ . @ @ . . . . @ . . . . . . . . @ @ . . . @ . . . . . @ . . . . . . . . . . . . . . . . @ . . @ @
	. . .
126	@ . @ @ . @ . . . . @ . . . . . . . @ . . . . . . . . . . . @ . . @ . . . . @ . . . . . . . . . . . . @ . . . . .
	. . .
127	. . . . . @ . . . . . . . . . . . . @ @ . . . . @ . @ . . . . . @ @ . . . . . @ . . @ . . . . @ . . . . .
	. @ .
128	. . @ @ . . . . . @ . @ . @ . . . . @ . . . @ @ . . @ . @ . . . . . @ . . . . . . . . . . . . . . . . @ . . . . . @

[illegible]

141 . @ .  
 . . @ . . . . @ . . . @ . . . . . . . . . . . . . . @ . . @ . .  
 . . . . @ . . . . @ . . . . . . . @ . . . . . @ @ @ . . . . @  
 . @ .  
 142 . . . . . . . . @ . . . . . @ @ . . . . @ . . . . . @ . @ . .  
 . . . @ . . . . . @ . @ . . @ . . . . . . . . . . . . . . @  
 . . @  
 143 . . . @ . . . . . . . . . @ . . . . @ . . . . . . . . . . .  
 . . . . @ . @ @ . . . . . . . . . @ . . . . . @ . . . . @  
 . . .  
 144 . . @ @ @ . . . . . @ @ . . . . . . . . . @ @ . . . . . . .  
 . . . . . . . . . . @ . . @ . . . @ @ . @ . . . . . . . .  
 @ . .  
 145 @ . . @ . . . . . . . . @ . . @ . @ . . . @ . . . . @ . . . .  
 . . . @ . . . . . @ . . . @ . . . . . . . . . @ . . . .  
 . . .  
 146 . . . . . @ . @ . . . . . . @ . @ @ . . . . . @ . . . . .  
 @ . . @ . . . @ . . . . . @ @ @ . . . . . @ . . . . . . .  
 @ @ .  
 147 . . @ . . . . @ . . . @ @ . . . . @ . @ . @ @ @ . . . . . .  
 . @ . . . @ . @ . . . . . . @ . . . . . @ . . . . . @ . .  
 . . .  
 148 . . . . . . . . . . . @ @ . . . . . @ @ . . . @ . . . . .  
 . . @ @ . @ . . . . @ . @ . . . . @ @ . @ @ . . . . . . .  
 @ . .  
 149 . . @ . . . . . . @ . . . . . . @ . . . . . . @ . . . . .  
 . . . . . @ @ . @ . . . . . . . . . @ . @ @ . . @ . . @ .  
 . . .  
 150 . . . . . @ . . . . @ @ . . @ . . . @ . . . @ . . . . . @  
 . . . . . @ . . . . @ . . @ @ . . . . @ . . . @ . @ . . .  
 . . .  
 151 . . @ . . . . . . @ . . @ . . . . . . . . . . @ . @ @  
 . . . . . @ @ . @ . . . @ . . . . . . . . @ . . . . . .  
 . . @  
 152 @ @ . . . @ . . . . @ . @ . . @ . @ . . @ . . @ . . @ . @

```

    . . . @ @ @ . . . @ . @ . . . . . @ . . @ . . . . @ . . .
    . @ .
153 . . . . . @ . . . . . . . @ . . . . . . . @ . . @ . . . . .
    . . . . @ . @ . . . . . . . . . . . @ . . . . . @ . @ . @
    . . .
154 . . . @ . . . . @ . . . . . . @ . @ . . . @ . . @ . . . @ . . .
    . . @ @ @ . @ . . . @ . . . . . @ . . @ . @ . . . @ . .
    . . .
155 . . . . . @ . . . . . @ . . . @ . . . @ @ . . . @ . . . @ @ .
    . @ . . . @ . . . . . . . . . . . @ . . . @ . . . @ .
    @ @ .
156 . . . . . . @ . . . . @ . @ @ . . . @ @ . @ @ . . . . . .
    . @ . . . . @ . @ . @ @ @ . . @ . . . . @ . . . . @ @ . . @
    . . .
157 . . . . . @ . . . . @ @ . . . . @ @ . @ . @ . . . . @ . @ .
    @ . . @ . @ . . . . . @ . . . @ . . . . @ @ . . . . .
    . . .
158 . . . . . . @ @ . . @ . . . . . . . . . @ . . . . @ . .
    . . @ . . . . . @ . . . @ . . . @ . @ . @ @ @ . . . .
    . . @
159 . @ . . . . . . . . @ . . . . @ . . @ . . . . @ . . . .
    . . . . @ @ . . @ . . . . . @ . . . . @ . . . @ . .
    . . .
160 @ . @ . . . @ . . . @ . . . @ . . . @ @ @ @ @ . @ . . @ @ . .
    . . . . . . . . . . . . . . . @ . @ @ . . . . @ . . .
    @ . .
161 @ . . . . . . . . @ @ . . . . @ . . @ . . . @ . . . .
    . . . @ @ . @ . @ . . . . @ . . @ @ . . . . . . . .
    . . .
162 @ . @ . . . . @ . . . . . . . . . . @ . . . . . .
    . @ . . @ . @ @ @ . . . . . . . . @ @ . @ . . .
    . . .
163 ""
164
165 tasks_data = ""

```



166	50
167	7 27 0 24
168	14 47 47 49
169	63 5 61 59
170	3 53 29 23
171	28 2 22 56
172	42 49 60 2
173	21 8 34 5
174	26 13 20 54
175	47 33 34 10
176	22 50 61 33
177	23 32 40 61
178	1 49 12 41
179	55 45 48 15
180	43 6 44 30
181	41 1 21 57
182	57 5 47 1
183	16 61 2 11
184	20 50 45 56
185	9 54 47 56
186	12 31 62 22
187	40 51 6 53
188	40 40 42 33
189	52 36 26 34
190	42 48 28 26
191	55 32 23 46
192	5 41 18 24
193	8 62 18 63
194	56 49 4 59
195	39 30 52 3
196	5 0 50 28
197	29 27 59 9
198	29 50 18 28
199	21 63 20 12
200	16 29 30 40

```

201 36 18 60 21
202 50 18 19 40
203 12 36 8 23
204 45 36 43 39
205 35 52 53 54
206 49 28 55 5
207 24 40 25 9
208 15 25 62 62
209 17 52 4 34
210 32 43 14 50
211 63 9 36 57
212 8 46 44 0
213 31 43 55 14
214 59 1 12 11
215 15 1 60 8
216 56 0 0 63
217 """
218
219 # 计算每个机器人的路径
220 grid = parse_map(map_data)
221 tasks = parse_tasks(tasks_data)
222
223 paths = []
224 total_time = 0
225 total_steps = 0 # 用于计算复杂度
226 branching_factor = 5 # 在每个节点处，最大可扩展5个方向（上、
    下、左、右、等待）
227
228 for i, task in enumerate(tasks):
229     start, goal = task
230     other_paths = [p[0] for p in paths] # 其他机器人路径
231     path, time = time_a_star_search(grid, start, goal,
    other_paths)
232     paths.append((path, time))
233     total_time = max(total_time, time)

```

```

234     total_steps += len(path)
235
236 # 计算算法复杂度
237 grid_size = grid.shape[0] * grid.shape[1]
238 estimated_complexity = branching_factor ** (total_steps / len(
    tasks))
239
240 # 绘图
241 fig, ax = plt.subplots(figsize=(8, 8))
242 colors = plt.cm.get_cmap('tab20', len(tasks)).colors # 使用更
    多颜色
243 step_map = np.full((grid.shape[0], grid.shape[1]), -1)
244 color_map_idx = np.full((grid.shape[0], grid.shape[1]), -1)
245
246 # 绘制障碍物
247 for i in range(grid.shape[0]):
248     for j in range(grid.shape[1]):
249         if grid[i, j] == '@':
250             ax.add_patch(plt.Rectangle((j, i), 1, 1, color='
    gray'))
251
252 # 绘制路径
253 for idx, (path, _) in enumerate(paths):
254     color = colors[idx % len(colors)]
255     if path:
256         # 起点
257         start_x, start_y = path[0]
258         ax.add_patch(Circle((start_y + 0.5, start_x + 0.5),
    0.4, edgecolor='black', facecolor=color, lw=2))
259
260         # 终点
261         end_x, end_y = path[-1]
262         ax.add_patch(
263             RegularPolygon((end_y + 0.5, end_x + 0.5),
    numVertices=6, radius=0.4, edgecolor='black', facecolor=

```

```

        color,
264         lw=2))
265
266     # 路径及时间标记
267     for t, (x, y) in enumerate(path):
268         if step_map[x, y] < t + 1:
269             step_map[x, y] = t + 1
270             color_map_idx[x, y] = idx
271             ax.add_patch(plt.Rectangle((y, x), 1, 1, color
=color, alpha=0.5))
272
273 # 在方块上标记时间
274 for i in range(step_map.shape[0]):
275     for j in range(step_map.shape[1]):
276         if step_map[i, j] > 0:
277             ax.text(j + 0.5, i + 0.5, str(step_map[i, j]-1),
ha='center', va='center', color='black')
278
279 ax.set_xticks(np.arange(0, grid.shape[1] + 1, 1))
280 ax.set_yticks(np.arange(0, grid.shape[0] + 1, 1))
281 ax.grid(which='both', color='black', linestyle='-', linewidth
=1)
282 plt.xlim(0, grid.shape[1])
283 plt.ylim(0, grid.shape[0])
284 plt.gca().invert_yaxis() # 上下翻转y轴以匹配要求
285 plt.show()
286
287 # 输出结果
288 print(f"总运输时间: {total_time}")
289 print(f"估计的算法复杂度: O({estimated_complexity:.2e})")
290 for i, (path, time) in enumerate(paths, 1):
291     print(f"机器人{i}: 路径 {path}, 时间消耗 {time}步")
292
293 file_path="results_第二问 64.xlsx"
294 if os.path.exists(file_path):

```

```

295     # 清空文件内容
296     with open(file_path, 'w') as file:
297         pass # 使用 pass 清空文件内容
298
299 data = {
300     "机器人编号": [],
301     "位置列表": [],
302     "时间开销": []
303 }
304
305 for i, (path, time) in enumerate(paths, 1):
306     data["机器人编号"].append(i)
307     data["位置列表"].append(path)
308     data["时间开销"].append(time)
309
310 df = pd.DataFrame(data)
311
312 # 导出到Excel文件
313 df.to_excel(file_path, index=False)

```

Problem3.py

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  from queue import PriorityQueue
4  from matplotlib.patches import Circle, RegularPolygon
5  import itertools
6  import os
7  import pandas as pd
8  def parse_map(map_data):
9      lines = map_data.strip().split("\n")
10     grid = []
11     for line in lines:
12         grid.append(line.split())
13     return np.array(grid)
14 def parse_tasks(tasks_data):

```

```

15     lines = tasks_data.strip().split("\n")
16     tasks = []
17     tot = 0
18     for line in lines:
19         start_x, start_y, end_x, end_y = map(int, line.split()
20 )
21         tot = tot + 1
22         tasks.append(((start_x, start_y), (end_x, end_y)))
23     return tasks
24 def parse_agents(agents_data):
25     lines = agents_data.strip().split("\n")
26     agent = []
27     tot = 0
28     for line in lines:
29         x, y = map(int, line.split())
30         tot = tot + 1
31         agent.append((x, y))
32     return agent
33 map_data = """
34 . . . . . @ . . @ . . . @ @ @ . @ . . @ . . . @ . . @ . .
35 . . @ . . . . . . . . . . . . @ . . . . . . . . . @
36 @ . .
37 . . @ . . . . . @ @ . . @ . . . . . . . . . . . . . . @
38 . . . . . . . . . @ . . . @ . . . @ . @ . @ . @ . . . .
39 . . . . @ @ . . . . . @ . . . @ . . . @ . . . . . . . .
40 . . .
41 . . . @ . @ . . . . . @ . . . . . . . . . @ @ . . @ .
42 . . . . @ @ . . . . . @ . . . @ . . . @ . . . . . . . .
43 . . .
44 . . . . . . . . . . . . @ @ . . . . @ . @ . . . @ . . .
45 . . . . . . . . . @ . . . . @ . . . . . @ . . . . . @ @ @ . .
46 . . @
47 . . @ . . . . @ . . @ . . . . . . . . . . . . . . @ . . .
48 . . . . @ @ @ . . . . . @ . . . . . . . . @ . . . . . . .
49 @ @ @

```

39 . . . . @ @ . . @ . @ . . . . @ . . . . . . . . . . @  
. @ . . . @ . . . . . @ . . . . . @ @ . @ . . . . . . . . .  
. . .

40 . . @ . . . . . @ . . . . @ . . @ . @ . @ . . . @ @ . . . @  
. . . . @ . . . . . . . . . . . @ @ @ . . @ . . @ . . @ . .  
. . @

41 . . . @ @ @ . @ . . . . . @ . @ . . . . @ . . . @ @ . . . @ .  
. . . . . . . . . . . . . @ . @ . . . . @ . . . . @ . . . .  
. . @

42 @ @ . @ . . . @ . . . . . @ . . . . . @ . . . . . @ . . . . .  
. . . . . . . @ @ @ . . @ . . . @ . . . . . @ . . . . .  
. . .

43 . . . . . . . . . @ . . . @ @ . . . . . . . @ . . . @ . .  
. @ . . . . @ . . . . . . . . @ . . . . . . . @ . . . .  
. . .

44 . . . . @ . . . . . . @ . . @ . . . . . . . . . . . @  
. @ . . . @  
@ . .

45 . . . . . . . @ . @ @ . . @ . . . . . . . . . . @ @ . @  
@ @ . . . @ . . . . @ . . @ . . @ . @ @ . @ . . @ @ . @ . .  
@ . @

46 . @ . . @ @ @ . . . . . . . . . . @ . . . . . . . . . . .  
. . @ . . . @ . . . . . . . . . @ @ @ @ . . . . @ . .  
@ . .

47 . @ . . . . @ . . . . @ . . . . . @ @ . . . . . @ . . . . @ . .  
. @ . . . . @ . . . . . . . . @ . . @ @ @ . . . . @ @ .  
. . @

48 . . . . . . . @ . . . . . . . . . @ . . . @ . . . . . .  
. . . . @ @ .  
. . .

49 . . . . . . . @ @ . . . . @ . . . . @ . . @ . . @ . . @ @ . .  
. . @ . @ . . . . . . @ . . . . . @ . . @ @ @ @ . . . .  
. @ .

50 . . . . . . . @ . . . . . . . . @ . @ . . @ . @ . . . . . .  
. . @ . . . . . . . . . @ . . . . . @ . . . @ .

[illegible]



. . . . . @ @ . . . . . @ . . @ . . . . @ . . . .  
 . @ .  
 63 . . @ @ . . . . . @ . @ . @ . . . . . @ . . . @ @ . . @ . @  
 . . . . . @ . . . . . . . . . . . . . . . @ .  
 . . @  
 64 @ @ . . . . @ . . . @ . . . . . . . . . . . @ . . . .  
 . . . . . . . . . @ . . . . . @ . . . @ . . . @ . . . @  
 . . @  
 65 . . . . . . . . . . . . . . . @ . @ . . . . . . . . .  
 @ @ . . . . . @ . . . . . @ . . . . . @ . . . . . @ . . . . .  
 @ . . .  
 66 @ . @ @ . . . . . @ . @ . . . @ . @ . . . @ . . @ . . . . . . . . .  
 . @ . . . . . . @ . @ . . . @ . . . @ . . . @ . . . . . . . . .  
 . . . .  
 67 @ . . . . . @ . . . . . . . . . . . . . . . @ . @ . @ . . . .  
 . . @ @ . @ . . . @ . . . @ . . . . . @ . . @ . . . . . . . . .  
 . . . .  
 68 . . . . . . . . . . . @ . . @ @ . . . @ . . @ . . . @ @ @  
 @ . . . . . @ . . @ . . . . . . . . @ . . . @ . . . . @ @ . .  
 . . @  
 69 @ . @ . . . . . . . @ . . . @ . . @ . . . . . . . . @ .  
 . @ . . . . . . . . . @ . . . . @ . . . @ . . @ . . . . . . . .  
 . . . .  
 70 . @ @ @ . . @ . . @ . . . . . . . . . . . @ . . . . . .  
 . @ . . . . . . . . . @ . . . @ . . . @ . @ . . @ . . @ @  
 . . @  
 71 . @ . . . . @ . . . . . . . . . . . . . . @ . . . . . @ . .  
 @ . . @ @ . . . . . . @ @ . . . . . . . . . . . @ @  
 . . . .  
 72 @ . . . . . . @ . @ @ . @ . . @ @ . @ @ @ . . . @ . . . . . .  
 . @ . . . . . . @ . . . . . . . . @ . . . . . @ . . . . @ .  
 . . . .  
 73 . . . . . . . . @ @ @ . . . . . . . @ @ . . . . . . . . .  
 . . @ . . . . . @ . . . . @ . @ . @ . . . . @ . @ . . . @ .  
 . . . .

74 . . . . . @ . @ . . @ @ . . . . @ . .  
. @ . @ . @ . . @ . . . . . @ . . . @ . . . . .  
. . @

75 . . . @ . . @ . . @ . . . . . @ @ . . . . @ . . . .  
@ . . . . . @ . . @ . . @ @ @ . . . . . . . . .  
. @ .

76 . . @ . . . . @ . . . @ . . . . . . . . @ . . @ . .  
. . . . @ . . . . @ . . . . . @ . . . . @ @ @ . . . . @  
. @ .

77 . . . . . @ . . . . @ @ . . . . @ . . . . @ . @ . .  
. . . @ . . . . @ . @ . . @ . . . . . . . . . . @  
. . @

78 . . . @ . . . . . @ . . . . @ . . . . . . . . . .  
. . . . @ . @ @ . . . . . . . . @ . . . . @ . . . @  
. . .

79 . . @ @ @ . . . . . @ @ . . . . . . . . @ @ . . . . .  
. . . . . . . . @ . . @ . . . @ @ . . @ . . . . .  
@ . .

80 @ . . @ . . . . . . @ . . @ . @ . . . @ . . . . @ . . . .  
. . . @ . . . . . @ . . . @ . . . . . . . . @ . . . .  
. . .

81 . . . . . @ . @ . . . . . @ . @ @ . . . . . @ . . . . .  
@ . . @ . . . @ . . . . . @ @ @ . . . . . @ . . . . .  
@ @ .

82 . . @ . . . . @ . . . @ @ . . . . @ . @ . @ @ @ . . . . .  
. @ . . . @ . @ . . . . . @ . . . . . @ . . . . @ . .  
. . .

83 . . . . . . . . . @ @ . . . . . @ @ . . . @ . . . . .  
. . @ @ . @ . . . . @ . @ . . . @ @ . @ @ . . . . .  
@ . .

84 . . @ . . . . . @ . . . . . @ . . . . . @ . . . . .  
. . . . . @ @ . @ . . . . . . . @ . @ @ . . @ . . @ .  
. . .

85 . . . . . @ . . . . @ @ . . @ . . . @ . . . @ . . . . @  
. . . . . @ . . . . @ . . @ @ . . . . @ . . . @ . @ . . .

[illegible]

```

    . @ . . @ . @ @ @ . . . . . . . . . . @ @ . @ . . . .
    . . .
98  """
99  tasks_data = """
100 7 27 0 24
101 14 47 47 49
102 63 5 61 59
103 3 53 29 23
104 28 2 22 56
105 42 49 60 2
106 21 8 34 5
107 26 13 20 54
108 47 33 34 10
109 22 50 61 33
110 23 32 40 61
111 1 49 12 41
112 55 45 48 15
113 43 6 44 30
114 41 1 21 57
115 57 5 47 1
116 16 61 2 11
117 20 50 45 56
118 9 54 47 56
119 12 31 62 22
120 40 51 6 53
121 40 40 42 33
122 52 36 26 34
123 42 48 28 26
124 55 32 23 46
125 5 41 18 24
126 8 62 18 63
127 56 49 4 59
128 39 30 52 3
129 5 0 50 28
130 29 27 59 9

```

```

131 29 50 18 28
132 21 63 20 12
133 16 29 30 40
134 36 18 60 21
135 50 18 19 40
136 12 36 8 23
137 45 36 43 39
138 35 52 53 54
139 49 28 55 5
140 24 40 25 9
141 15 25 62 62
142 17 52 4 34
143 32 43 14 50
144 63 9 36 57
145 8 46 44 0
146 31 43 55 14
147 59 1 12 11
148 15 1 60 8
149 56 0 0 63
150 """
151 agents_data = """
152 10 20
153 0 0
154 44 28
155 16 41
156 26 45
157 """
158
159 grid = parse_map(map_data)
160 tasks = parse_tasks(tasks_data)
161 agents = parse_agents(agents_data)
162 #a = [0]
163 #a = [0]
164 a = [1, 0, 2, 3, 4]
165

```

```

166
167 def heuristic(a, b):
168     return abs(a[0] - b[0]) + abs(a[1] - b[1])
169 def time_a_star_search(grid, start, goal, other_paths, tt = 0)
    :
170     neighbors = [(0, 1), (1, 0), (0, -1), (-1, 0), (0, 0)]
171     close_set = set()
172     came_from = {}
173     gscore = {(start[0], start[1], tt): 0}
174     fscore = {(start[0], start[1], tt): heuristic(start, goal)
    }
175     open_set = PriorityQueue()
176     open_set.put((fscore[(start[0], start[1], tt)], (start[0],
start[1], tt)))
177     while not open_set.empty():
178         _, current = open_set.get()
179         x, y, t = current
180
181         if (x, y) == goal:
182             path = []
183             while current in came_from:
184                 path.append((current[0], current[1]))
185                 current = came_from[current]
186             path.append((start[0], start[1]))
187             return path[::-1], t # 返回路径和时间消耗
188
189         close_set.add(current)
190         for i, j in neighbors:
191             neighbor = x + i, y + j, t + 1
192             if 0 <= neighbor[0] < grid.shape[0] and 0 <=
neighbor[1] < grid.shape[1]:
193                 if grid[neighbor[0]][neighbor[1]] == '@':
194                     continue
195
196                 conflict = False

```

```

197         for path in other_paths:
198             """
199             if(len(path) == 0):break
200             if (neighbor[0], neighbor[1]) == goal:
201                 for i in range(0, len(path)):
202                     if(path[i] == goal and i >= t + 1)
:
203                 conflict = True
204                 break
205             if conflict:
206                 break
207             """
208             if len(path) > t + 1 and path[t + 1] == (
neighbor[0], neighbor[1]):
209                 conflict = True
210                 break
211             """
212             if len(path) <= t + 1 and path[-1] == (
neighbor[0], neighbor[1]):
213                 conflict = True
214                 break
215             """
216             if len(path) > t and len(path) > t + 1 and
path[t] == (neighbor[0], neighbor[1]) and path[t + 1] == (x
, y):
217                 conflict = True
218                 break
219             if conflict:
220                 continue
221
222             tentative_g_score = gscore[current] + 1
223
224             if neighbor in close_set and tentative_g_score
>= gscore.get(neighbor, 0):
225                 continue

```

```

226
227         if tentative_g_score < gscore.get(neighbor, 0)
or neighbor not in [i[1] for i in open_set.queue]:
228             came_from[neighbor] = current
229             gscore[neighbor] = tentative_g_score
230             fscore[neighbor] = tentative_g_score +
heuristic((neighbor[0], neighbor[1]), goal)
231             open_set.put((fscore[neighbor], neighbor))
232
233     return False, float('inf')
234
235 def check(x):
236
237     paths = [[] for _ in range(len(agents))]
238     paths2 = [[] for _ in range(len(agents))]
239     vis = [0 for _ in range(len(tasks))]
240     cnt = 0
241     for ii in range(0, len(a)):
242         i1 = a[ii]
243         agent = agents[i1]
244         if (cnt == len(tasks)): break
245         start1 = agent
246         start2 = -1
247         goal2 = -1
248         id = -1
249         minw = 100000
250         sumt = 0
251         other_paths = []
252         if(ii != 0):
253             for i in range(0, ii - 1):
254                 other_paths.append(paths[a[i]])
255
256         for i2, task in enumerate(tasks):
257             start, goal = task
258             if (vis[i2]): continue

```



```

259         path, t = time_a_star_search(grid, start1, start,
other_paths, sumt)
260         if (minw > t):
261             id = i2
262             start2 = start
263             goal2 = goal
264             minw = t
265
266         path1, t1 = time_a_star_search(grid, start1, start2,
other_paths, sumt)
267         path2, t2 = time_a_star_search(grid, start2, goal2,
other_paths, t1)
268         if (t2 > x): continue
269
270         for p in path1:
271             paths[i1].append((p))
272         for p in path2[1:]:
273             paths[i1].append((p))
274         paths2[i1].append(path1)
275         paths2[i1].append(path2)
276         vis[id] = 1
277         cnt = cnt + 1
278         sumt = t2
279         start1 = goal2
280         while (cnt != len(tasks) and sumt <= x):
281             start2 = -1
282             goal2 = -1
283             id = -1
284             minw = 100000
285             other_paths = []
286             if(ii != 0):
287                 for i in range(0, ii - 1):
288                     other_paths.append(paths[a[i]])
289
290             for i2, task in enumerate(tasks):

```

```

291         start, goal = task
292         if (vis[i2]): continue
293         path, t = time_a_star_search(grid, start1,
start, other_paths, sumt)
294         if (minw > t):
295             id = i2
296             start2 = start
297             goal2 = goal
298             minw = t
299
300         path1, t1 = time_a_star_search(grid, start1,
start2, other_paths, sumt)
301         path2, t2 = time_a_star_search(grid, start2, goal2
, other_paths, t1)
302         if (t2 > x): break
303         for p in path1[1:]:
304             paths[i1].append((p))
305         for p in path2[1:]:
306             paths[i1].append((p))
307         paths2[i1].append(path1)
308         paths2[i1].append(path2)
309         vis[id] = 1
310         cnt = cnt + 1
311         sumt = t2
312         start1 = goal2
313     flag = False
314     if cnt == len(tasks):
315         flag = True
316
317     return flag, paths, paths2
318 def solve():
319     l = 0
320     r = 10000
321     while(l < r):
322         mid = int((l + r) / 2)

```

```

323     print(mid)
324     flag, paths, paths2 = check(mid)
325     if(flag):
326         r = mid
327     else:
328         l = mid + 1
329     flag, paths, paths2 = check(l)
330     total_time = 1
331     total_steps = 0
332     for p in paths:
333         total_steps += len(p)
334
335     print(l)
336     for i, p in enumerate(paths):
337         print(len(p) - 1, i + 1, p)
338     for p in paths2:
339         print(len(p))
340         for pp in p:
341             print(pp)
342
343     file_path = "results_第三问 64.xlsx"
344     if os.path.exists(file_path):
345         with open(file_path, 'w') as file:
346             pass
347
348     data = {
349         "机器人编号": [],
350         "位置列表": [],
351         "任务列表": [],
352         "时间开销": []
353     }
354
355     for i, p in enumerate(paths, 1):
356         data["机器人编号"].append(i)
357         data["位置列表"].append(p)

```

```

358         data["时间开销"].append(len(p)-1)
359     for p in paths2:
360         data["任务列表"].append(p)
361
362     df = pd.DataFrame(data)
363     df.to_excel(file_path, index=False)
364
365 solve()

```

Problem4.py

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  from queue import PriorityQueue
4  from matplotlib.patches import Circle, RegularPolygon
5  import itertools
6  import pandas as pd
7  import os
8  def parse_map(map_data):
9      lines = map_data.strip().split("\n")
10     grid = []
11     for line in lines:
12         grid.append(line.split())
13     return np.array(grid)
14 def parse_tasks(tasks_data):
15     lines = tasks_data.strip().split("\n")
16     tasks = []
17     tot = 0
18     for line in lines:
19         start_x, start_y, end_x, end_y = map(int, line.split()
20 )
21         tot = tot + 1
22         tasks.append(((start_x, start_y), (end_x, end_y)))
23     return tasks
24 def parse_agents(agents_data):
25     lines = agents_data.strip().split("\n")

```

```

25     agent = []
26     tot = 0
27     for line in lines:
28         x, y = map(int, line.split())
29         tot = tot + 1
30         agent.append((x, y))
31     return agent
32
33 map_data = """
34 . . . . .
35 . @ . . @ . @ .
36 . . . . .
37 . . . . .
38 @ . . . . .
39 . . . . .
40 . . . . @ . . .
41 . @ . . . . .
42 """
43 tasks_data = """
44 1 7 3 3
45 5 0 7 5
46 7 4 5 2
47 4 5 6 7
48 6 5 0 6
49 5 2 2 4
50 0 4 7 2
51 4 1 1 2
52 """
53 agents_data = """
54 0 0
55 7 0
56 """
57 grid = parse_map(map_data)
58 tasks = parse_tasks(tasks_data)
59 agents = parse_agents(agents_data)

```

```

60
61 must = [[0,1,2],[]]
62 #must = [[0,1,5],[]]
63 #must = [[],[0,6,7,10,17,23,32,40],[],[1,5,8,20],[]]
64
65 a = [1, 0]
66 #a = [0, 1]
67 #a = [0, 3, 2, 4, 1]
68
69 def heuristic(a, b):
70     return abs(a[0] - b[0]) + abs(a[1] - b[1])
71 def time_a_star_search(grid, start, goal, other_paths, tt = 0)
72     :
73     neighbors = [(0, 1), (1, 0), (0, -1), (-1, 0), (0, 0)]
74     close_set = set()
75     came_from = {}
76     gscore = {(start[0], start[1], tt): 0}
77     fscore = {(start[0], start[1], tt): heuristic(start, goal)
78 }
79     open_set = PriorityQueue()
80     open_set.put((fscore[(start[0], start[1], tt)], (start[0],
81 start[1], tt)))
82     while not open_set.empty():
83         _, current = open_set.get()
84         x, y, t = current
85
86         if (x, y) == goal:
87             path = []
88             while current in came_from:
89                 path.append((current[0], current[1]))
90                 current = came_from[current]
91             path.append((start[0], start[1]))
92             return path[::-1], t # 返回路径和时间消耗
93
94     close_set.add(current)

```

```

92         for i, j in neighbors:
93             neighbor = x + i, y + j, t + 1
94             if 0 <= neighbor[0] < grid.shape[0] and 0 <=
neighbor[1] < grid.shape[1]:
95                 if grid[neighbor[0]][neighbor[1]] == '@':
continue
96
97                 conflict = False
98                 for path in other_paths:
99                     """
100                     if(len(path) == 0):break
101                     if (neighbor[0], neighbor[1]) == goal:
102                         for i in range(0, len(path)):
103                             if(path[i] == goal and i >= t + 1)
:
104                             conflict = True
105                             break
106                     if conflict:
107                         break
108                     """
109                     if len(path) > t + 1 and path[t + 1] == (
neighbor[0], neighbor[1]):
110                         conflict = True
111                         break
112                     """
113                     if len(path) <= t + 1 and path[-1] == (
neighbor[0], neighbor[1]):
114                         conflict = True
115                         break
116                     """
117                     if len(path) > t and len(path) > t + 1 and
path[t] == (neighbor[0], neighbor[1]) and path[t + 1] == (x
, y):
118                         conflict = True
119                         break

```

```

120         if conflict: continue
121
122         tentative_g_score = gscore[current] + 1
123
124         if neighbor in close_set and tentative_g_score
125         >= gscore.get(neighbor, 0): continue
126
127         if tentative_g_score < gscore.get(neighbor, 0)
128         or neighbor not in [i[1] for i in open_set.queue]:
129             came_from[neighbor] = current
130             gscore[neighbor] = tentative_g_score
131             fscore[neighbor] = tentative_g_score +
132             heuristic((neighbor[0], neighbor[1]), goal)
133             open_set.put((fscore[neighbor], neighbor))
134
135     return False, float('inf')
136
137 def check(x):
138     paths = [[] for _ in range(len(agents))]
139     paths2 = [[] for _ in range(len(agents))]
140     vis = [0 for _ in range(len(tasks))]
141     cnt = 0
142     for ii in range(0, len(a)):
143         i1 = a[ii]
144         agent = agents[i1]
145         if cnt == len(tasks): break
146         start1 = agent
147         start2 = -1
148         goal2 = -1
149         id = -1
150         minw = 100000
151         sumt = 0
152         other_paths = []
153         if ii != 0:
154             for i in range(0, ii - 1):

```



```

152         other_paths.append(paths[a[i]])
153
154     for i2, task in enumerate(tasks):
155         start, goal = task
156         if vis[i2]: continue
157
158         flag = False
159         for i3 in range(0, len(a)):
160             if flag: break
161             if i3 == i1: continue
162             for i4 in must[i3]:
163                 if i4 == i2:
164                     flag = True
165                     break
166             if flag: continue
167         path, t = time_a_star_search(grid, start1, start,
other_paths, sumt)
168         if minw > t:
169             id = i2
170             start2 = start
171             goal2 = goal
172             minw = t
173         if goal2 == -1: continue
174
175         path1, t1 = time_a_star_search(grid, start1, start2,
other_paths, sumt)
176         path2, t2 = time_a_star_search(grid, start2, goal2,
other_paths, t1)
177
178         if t2 > x: continue
179
180         for p in path1: paths[i1].append((p))
181         for p in path2[1:]: paths[i1].append((p))
182         paths2[i1].append(path1)
183         paths2[i1].append(path2)

```

```

184     vis[id] = 1
185     cnt = cnt + 1
186     sumt = t2
187     start1 = goal2
188     while cnt != len(tasks) and sumt <= x:
189         start2 = -1
190         goal2 = -1
191         id = -1
192         minw = 100000
193
194         for i2, task in enumerate(tasks):
195             start, goal = task
196             if (vis[i2]): continue
197
198             flag = False
199             for i3 in range(0, len(a)):
200                 if flag: break
201                 if i3 == i1: continue
202                 for i4 in must[i3]:
203                     if i4 == i2:
204                         flag = True
205                         break
206             if flag: continue
207             path, t = time_a_star_search(grid, start1,
start, other_paths, sumt)
208             if minw > t:
209                 id = i2
210                 start2 = start
211                 goal2 = goal
212                 minw = t
213             if goal2 == -1: break
214
215             path1, t1 = time_a_star_search(grid, start1,
start2, other_paths, sumt)
216             path2, t2 = time_a_star_search(grid, start2, goal2

```

```

, other_paths, t1)
217         if t2 > x: break
218         for p in path1[1:]: paths[i1].append((p))
219         for p in path2[1:]: paths[i1].append((p))
220         paths2[i1].append(path1)
221         paths2[i1].append(path2)
222         vis[id] = 1
223         cnt = cnt + 1
224         sumt = t2
225         start1 = goal2
226
227     flag = False
228     if cnt == len(tasks): flag = True
229
230     return flag, paths, paths2
231 def solve():
232     l = 0
233     r = 1250
234     while(l < r):
235         mid = (l + r) // 2
236         print(mid)
237         flag, paths, paths2 = check(mid)
238         if flag: r = mid
239         else: l = mid + 1
240     flag, paths, paths2 = check(l)
241
242     print(l)
243     for i, p in enumerate(paths): print(len(p) - 1, i + 1, p)
244     for p in paths2:
245         print(len(p))
246         for pp in p: print(pp)
247
248     file_path = "results_第四问 8.xlsx"
249     if os.path.exists(file_path):
250         with open(file_path, 'w') as file:

```

```
251         pass
252
253     data = {
254         "机器人编号": [],
255         "位置列表": [],
256         "任务列表": [],
257         "时间开销": []
258     }
259
260     for i, p in enumerate(paths, 1):
261         data["机器人编号"].append(i)
262         data["位置列表"].append(p)
263         data["时间开销"].append(len(p)-1)
264     for p in paths2:
265         data["任务列表"].append(p)
266
267     df = pd.DataFrame(data)
268     df.to_excel(file_path, index=False)
269
270 solve()
```