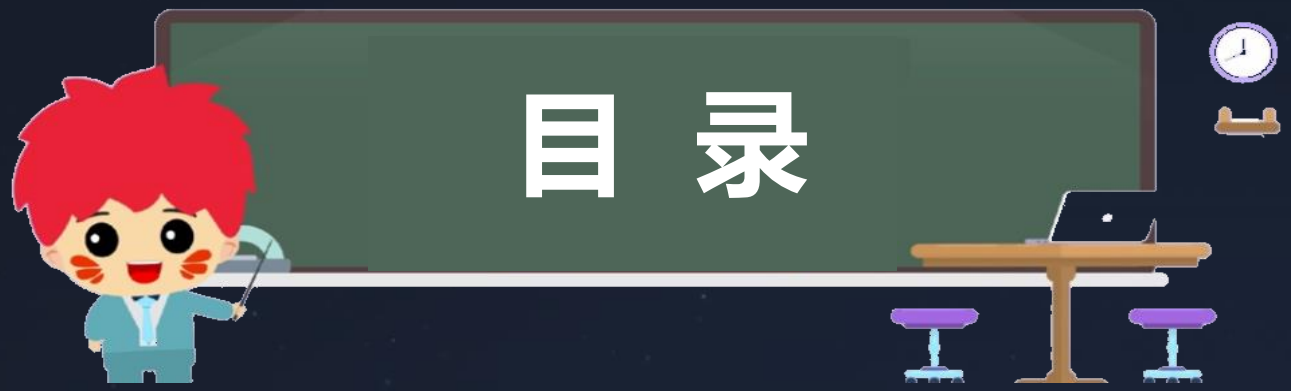
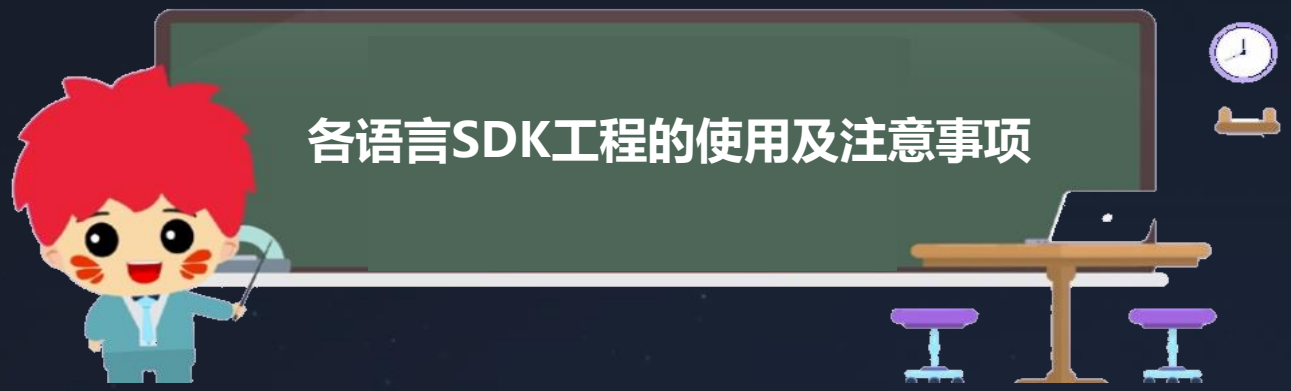
2019年华为软件精英挑战赛



目 录

1. 各语言SDK工程的使用及注意事项
2. 任务运行的常见问题
3. 赛题规则解读
4. 判题器的设计与实现

各语言SDK工程的使用及注意事项



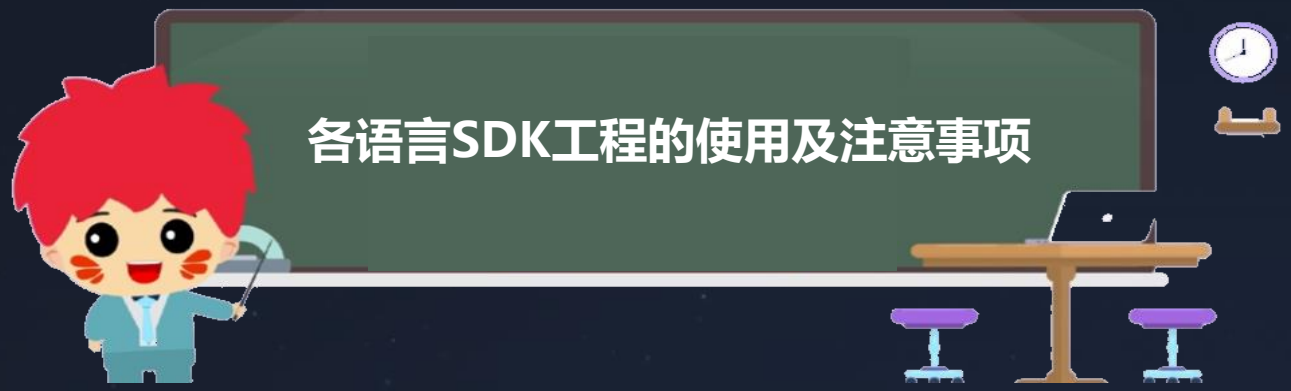
● 平台运行说明

大赛网站系统在进行系统运行时：

1. 对参赛选手的程序进行编译（python除外）
2. 对编译后的程序进行运行（python直接运行）生成答案文件（地图信息由系统自动加载，参赛选手无需考虑）

   （该步骤运行参赛选手的程序产生所有车辆路径的时间为程序运行时间，单位ms）
3. 使用地图与参赛选手输出的答案文件作为判题系统的输入，进行调度时间的计算。

注意：参赛选手的程序一次性输出车辆路径，不存在与判题系统的多次交互

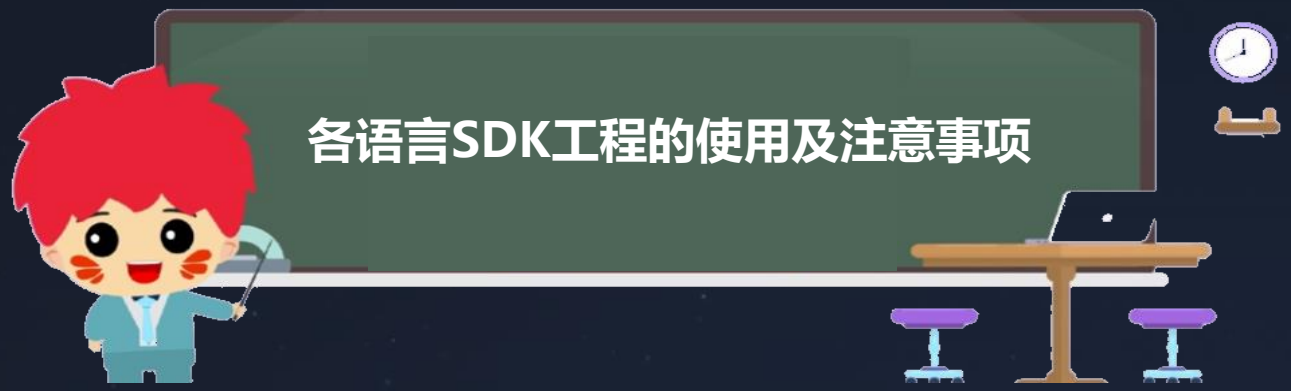


输入与输出接口

本次大赛给定的各语言SDK工程中已经明确定义了程序的输入与输出接口，三个地图文件和一个答案的输出文件全部依靠程序的输入与输出参数来进行传递。

请各位参赛选手在程序中直接使用程序的输出与输出参数，**直接操作程序输入的地图文件进行读取，写入答案至输出的答案文件即可**。不需要在程序中自行定义地图的存取位置与答案的输出位置。

注意：给定的SDK不要修改主函数所有文件名



- 平台运行说明

各语言的输入均为文件路径（含文件名），输出也为文件路径（含文件名）

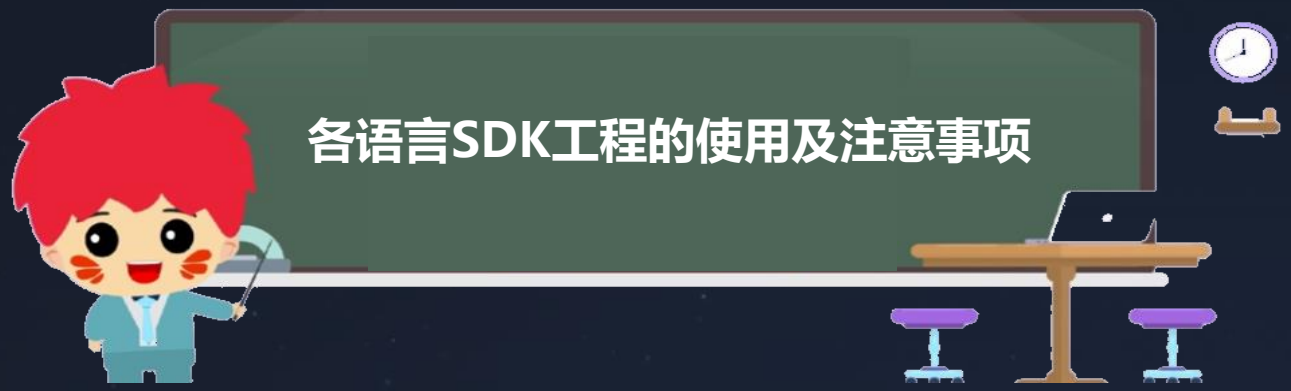
JAVA启动： com.huawei.Main $(path)/car.txt $(path)/road.txt $(path)/cross.txt $(path)/answer.txt

Python启动： python src/CodeCraft-2019.py $(path)/car.txt $(path)/road.txt $(path)/cross.txt $(path)/answer.txt

C/C++启动： CodeCraft-2019 $(path)/car.txt $(path)/road.txt $(path)/cross.txt $(path)/answer.txt

注意：

answer_path需要以写入的方式打开，自动创建文件（系统保证文件上级目录是存在的，不需要在代码中创建目录）



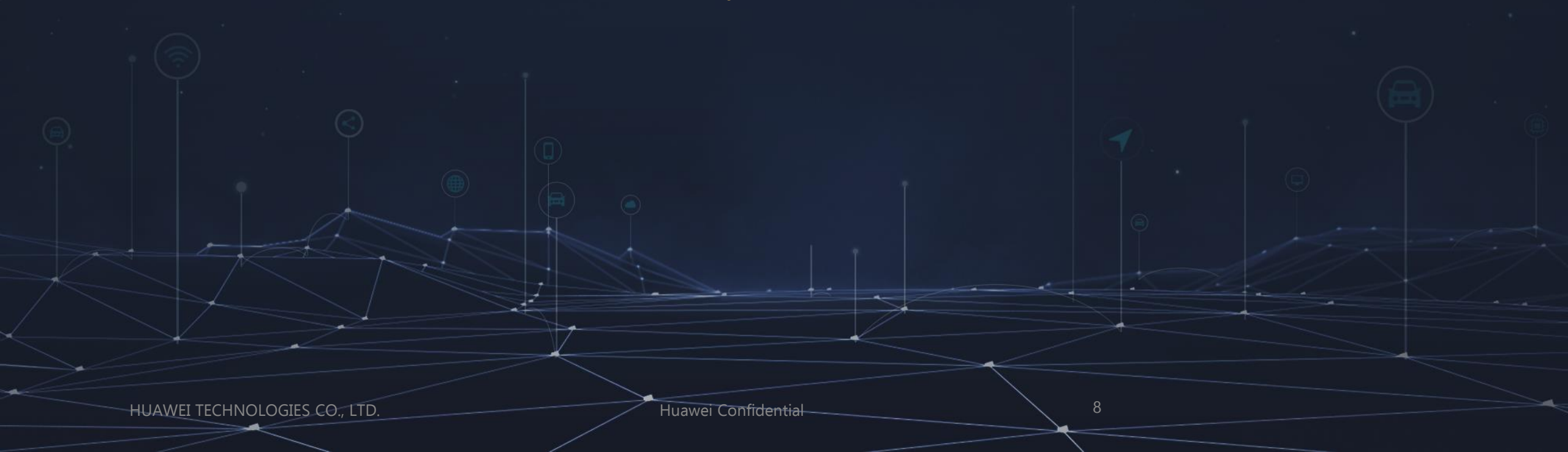
- 各语言开发需要做的事情

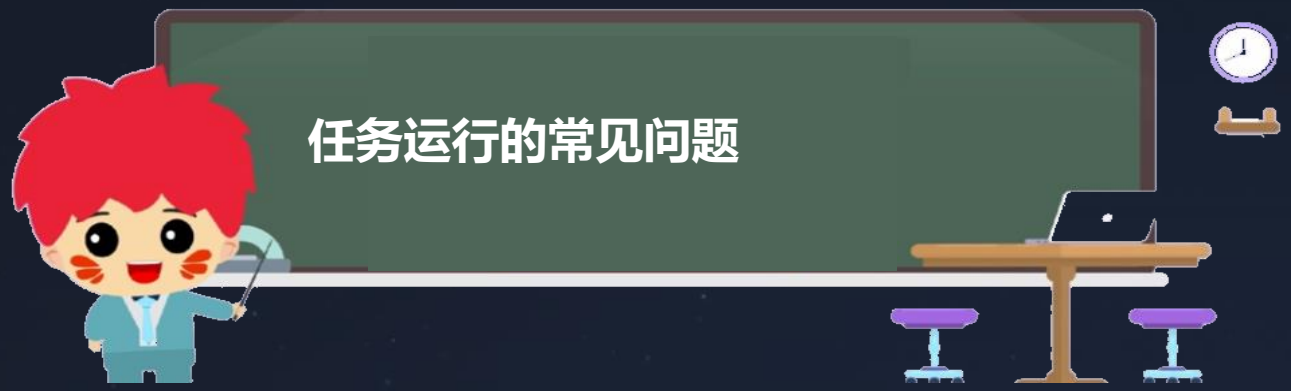
- Java：需要将添加的java文件按SDK示例添加makelist.txt中
- c/c++：需要将添加的头文件路径添加的CMakeLists.txt中,cpp文件保存在CodeCraft-2019目录。“include_directories(${PROJECT_SOURCE_DIR}/CodeCraft-2019)”
- Python：需要使用CodeCraft-2019同级目录的logs目录和日志文件，上传时请不要删除；或者将代码中日志相关代码全部删除

注意：

Java代码的包名是com.huawei，不要再增加其他的，比如main.java.com.huawei

任务运行常见问题

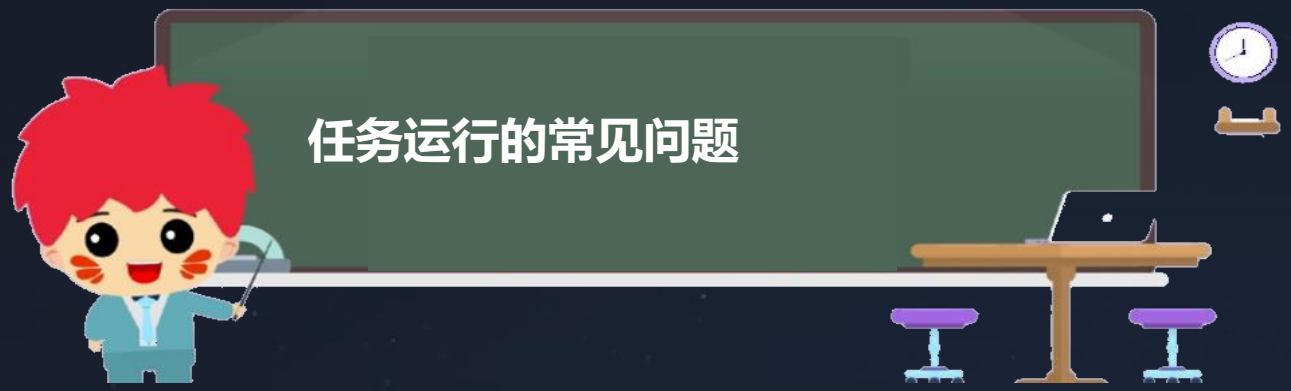




- 程序包上传失败

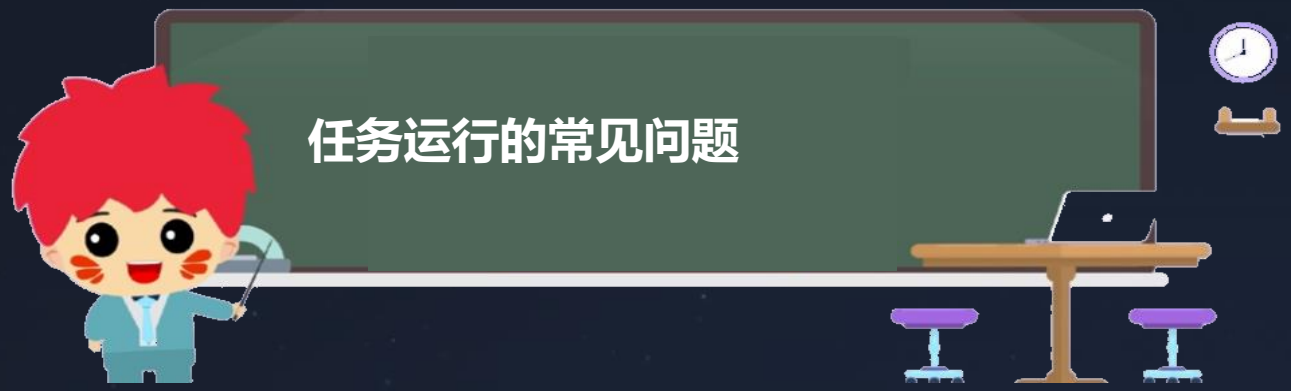
请严格按照上传的要求来进行压缩包内容的检查，尤其请注意隐藏文件、隐藏目录；
压缩包打包不需要打包程序的编译输出结果、中间过程文件、地图目录。

注意：
打包上传时使用脚本打包好的包就可以了，不需要再外层再次进行打包，否则会运行失败。
Java语言需要注意删除内部类生成的含”$”的中间过程文件
目录中存在含”~”的临时文件



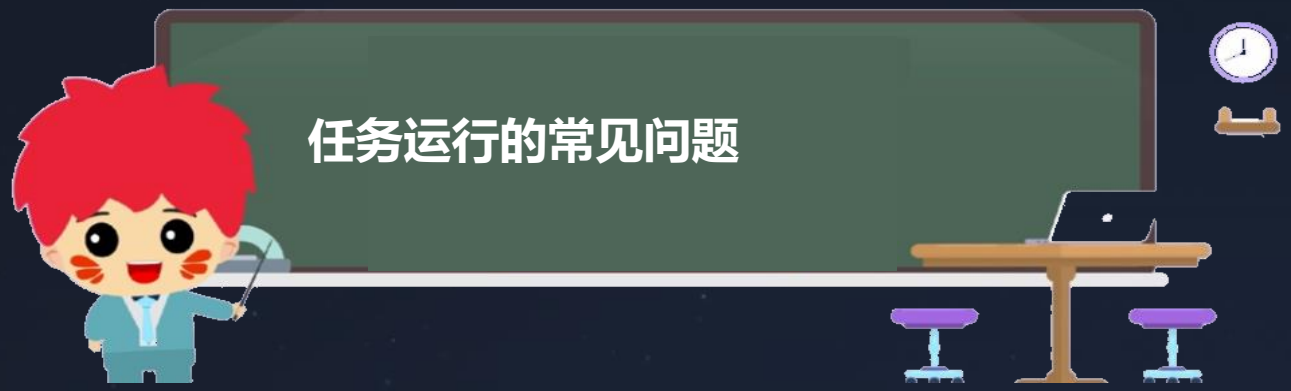
- 线上任务运行时间太久运行

线上运行时间太久是因为选手输出的车辆实际出发时间太晚太晚，比如几千万时间片以后才发车，几百万时间片后才发车的情况，判题系统需要调度很长时间才能计算完成，请耐心等待。



- 任务运行结果无效值的说明

团队任务运行结果100000000000为系统定义的无效值，当任务出现异常时，结果记此无效值。
赛事直播为发布的地图对应的成绩求和，比如两张地图对应的无效的成绩为2000000000000。



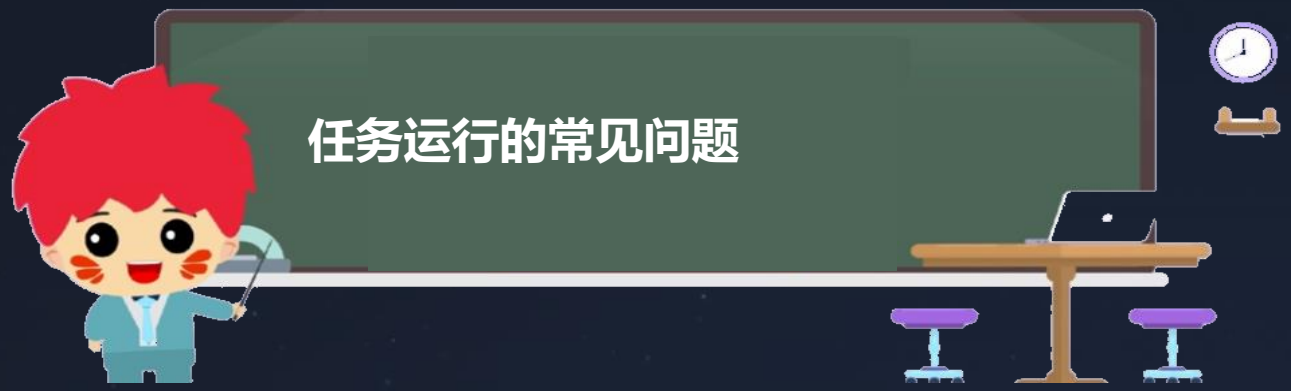
- 直播成绩

1. 目前初赛发布两张比赛地图，赛事直播页面会对两张地图的成绩求和作为总成绩进行排序。
2. 团队任务**同一代码**运行了两份地图，则直播页面会对两份在图的成绩自动求和。
3. 团队任务**仅运行一张**地图，则直播页面的总成绩默认会将一份地图的成绩按无效值处理。

注意：

请各团队针对每一份代码选择两份地图进行运行任务，直播页面总成绩就会取各个地图对应的成绩进行求和

直播成绩是取同一次上传的代码，依次先后运行两个地图对应的任务，而不是同样的代码上传两次，分别运行两个地图的任务。

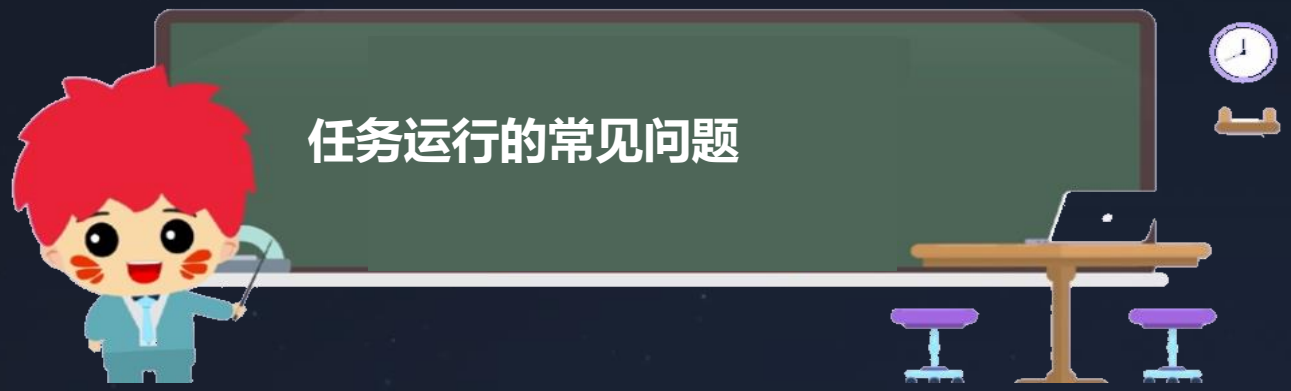


- C++编译失败或运行失败

请参赛选手在本地linux系统安装大赛指定的编译器版本进行本地测试，因**编译器版本**的不同可能会出现些不同的错误，导致在大赛网站系统的编译、运行失败。

注意：

一般情况下本地测试成功，但在大赛平台运行失败或编译错误，均为编译版本不同而没有在本地发现问题，更换同版本的编译器，本地解决告警和错误，调试正确上传后均可解决。



- 任务运行错误提示说明

➢ Programe runs too long:

1、选手的程序运行超过5分钟 2、判题器运行时间超时（远大于5分钟）

➢ Programe runs failed:

选手的程序运行失败（段错误、程序代码异常等）

➢ answer error:

1、选手的程序运行生成的答案超过系统大小限制 2、选手的程序运行结束，但是未生成answer.txt

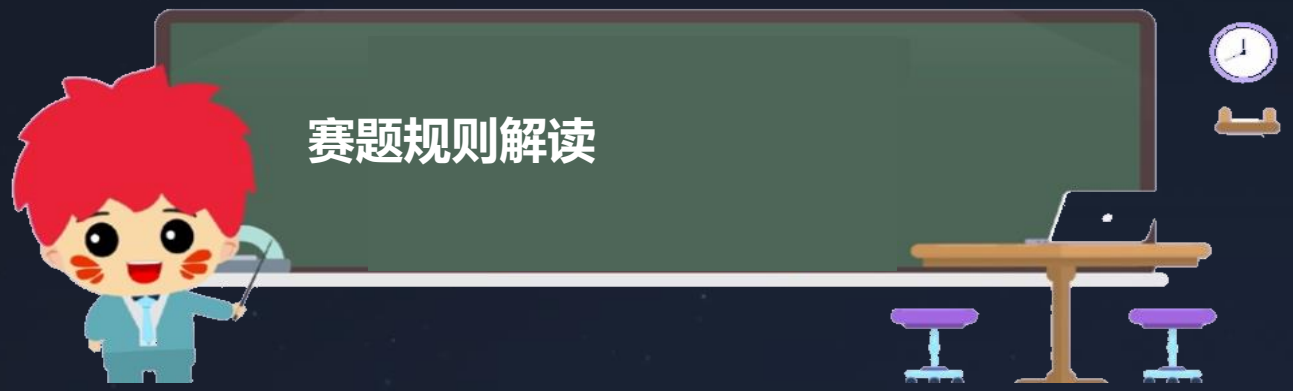
➢ system error:

因网络连接原因导致系统连接出现异常，非参赛选手的原因，请参赛选手重新运行任务即可。

➢ 其余的有具体的错误信息

阅读具体提示信息（编译错误、判题错误）

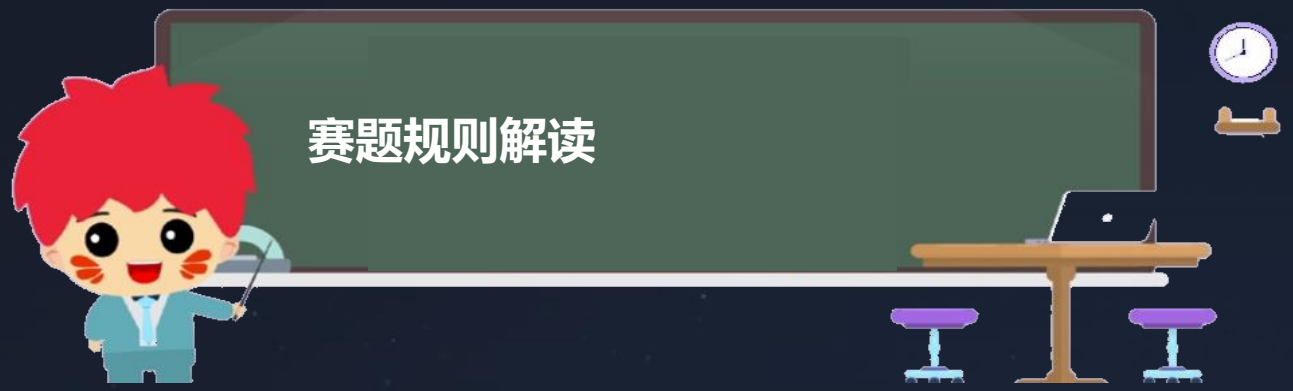
赛题规则解读



- 地图与图形化

任务书给定系统的输入道路与路口信息，与表述的图形化没有必然对应关系。根据输入的道路和路口信息，可以生成很多任意的图形界面，任务书中图形表示仅仅是以直观的形式去理解，不代表一定是该图形样式

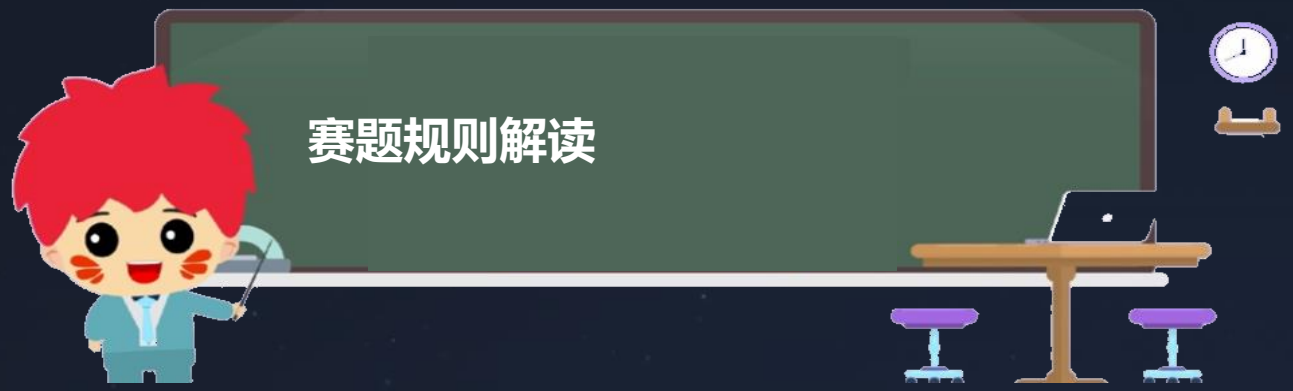
道路在图形上的形状也不做限定要求，可以理解成直的，也可以理解成弯曲的，也可以理解是螺旋上升与下降等，只需要保证道路的起始点与终止点正确连接即可



- 道路、车辆等输入的最大规模

道路不存在立交的情况、每个路口最大连接4条道路

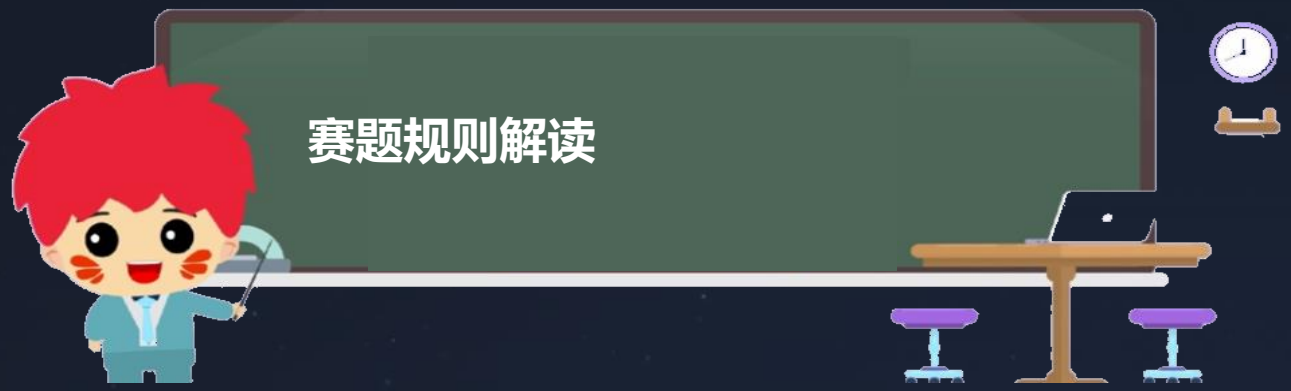
道路与车辆的规模与整个系统的难度相关，也是此系统的难点之一，恕不便透露，请谅解



- 输出文件要求

  - 系统运行在linux操作系统之上，建议参赛选手输出的答案文件格式统一为UNIX格式
  - 车辆的实际出发时间由参赛选手自行决定，但是**不得早于车辆的计划出发时间**
  - 每条车辆路径数据以“(”起始，以“)”终止，路径中**各道路（非路口）**以“,”分隔，中间只可以出现0个或多个空格“ ”，只接受英文",“，不允许出现其他非法字符。
  - **每条车辆路独占一行**。

(1005, 1, 501, 514, 504, 517, 507, 508, 509, 524)
(1006, 1, 513, 517, 521, 510, 511, 512)
(1007, 1, 513, 504, 518, 507, 521, 510, 511, 512)

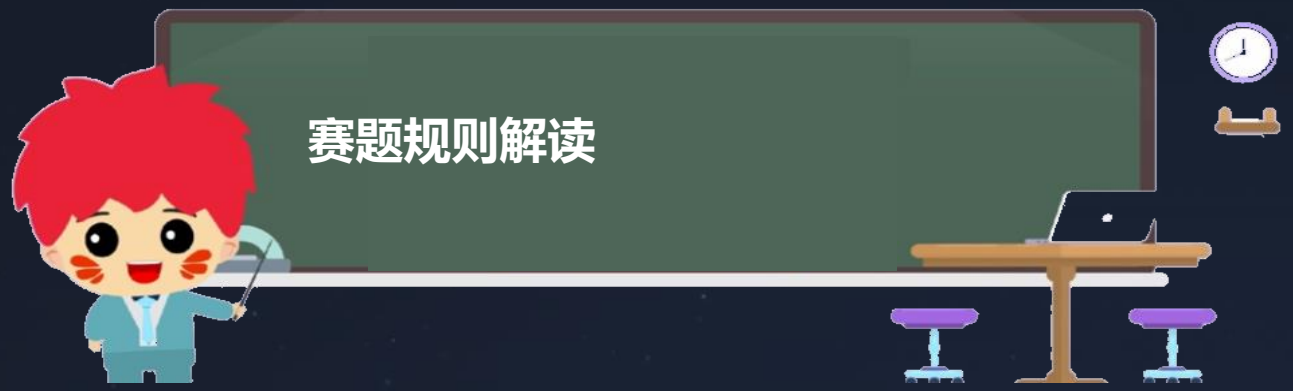


- 几点约束

为了简化系统设计，**车辆速度设定均小于等于所有道路的最小长度**，不考虑一个时间调度一个车辆可以越过一条道路的情况。

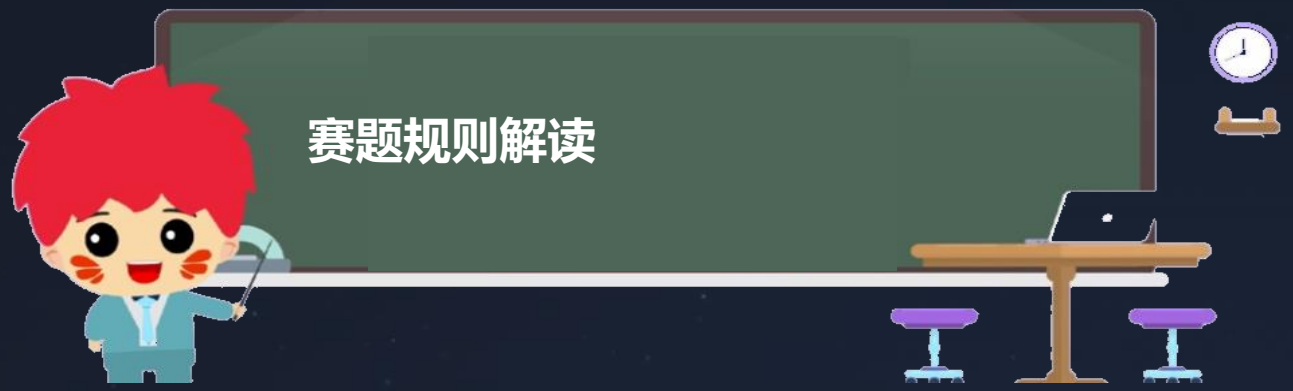
路口的向量表示，只规定**按顺时针顺序**约定该路口的各条道路，不约定第一条道路与终最后一条道路具体是哪条，只要保证顺时针顺序就可以

系统保证各道路ID、车辆ID、路口ID系统唯一存在（int范围内可表示），不保证连续、排序

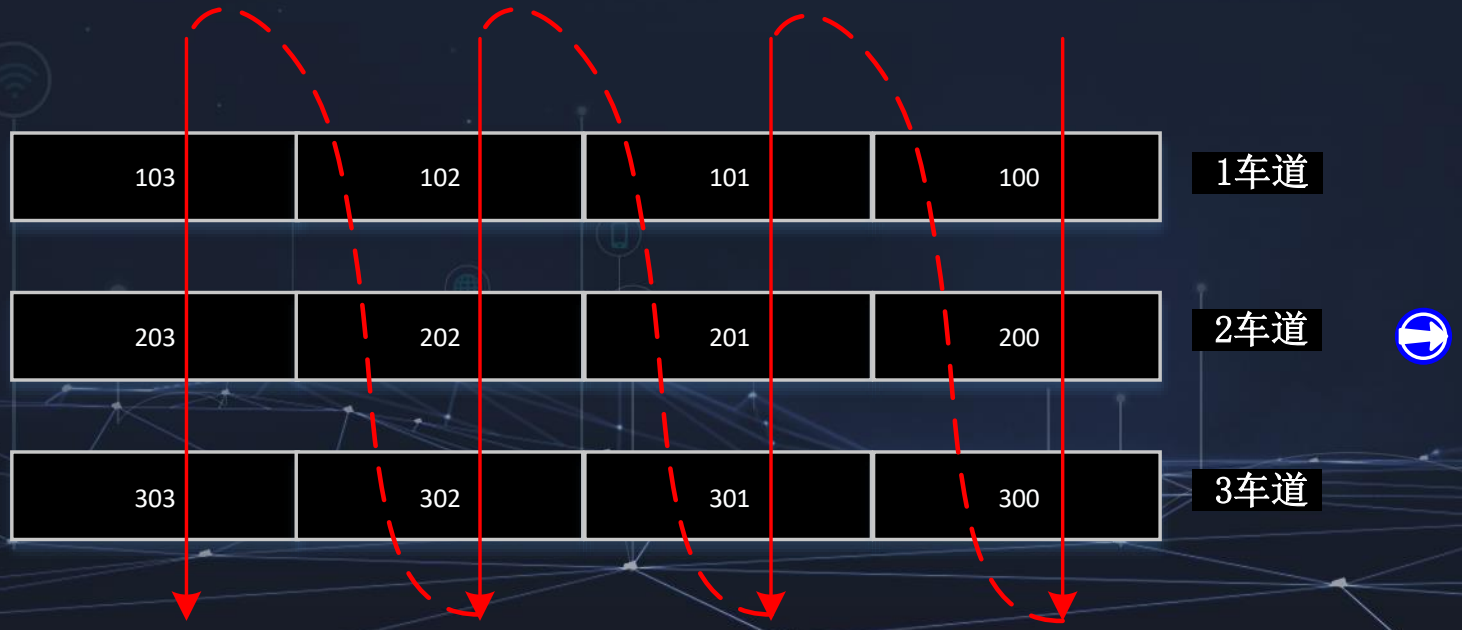


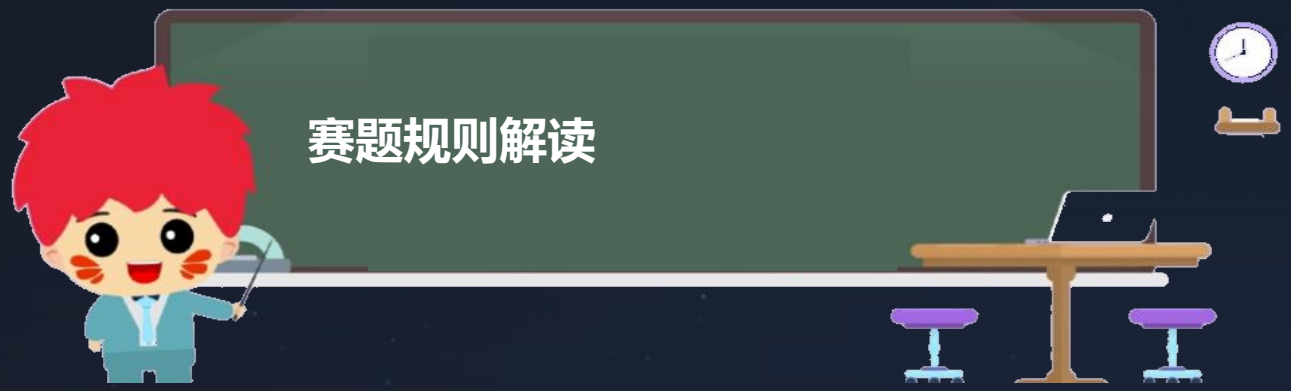
- 系统编译与运行时间要求

整个系统限制参赛选手程序编译时间最大为60s，程序运行生成answer.txt时间为300s。


赛题规则解读

- 车辆驶出道路的优先级


103
102
101
100
1车道
203
202
201
200
2车道
303
302
301
300
3车道



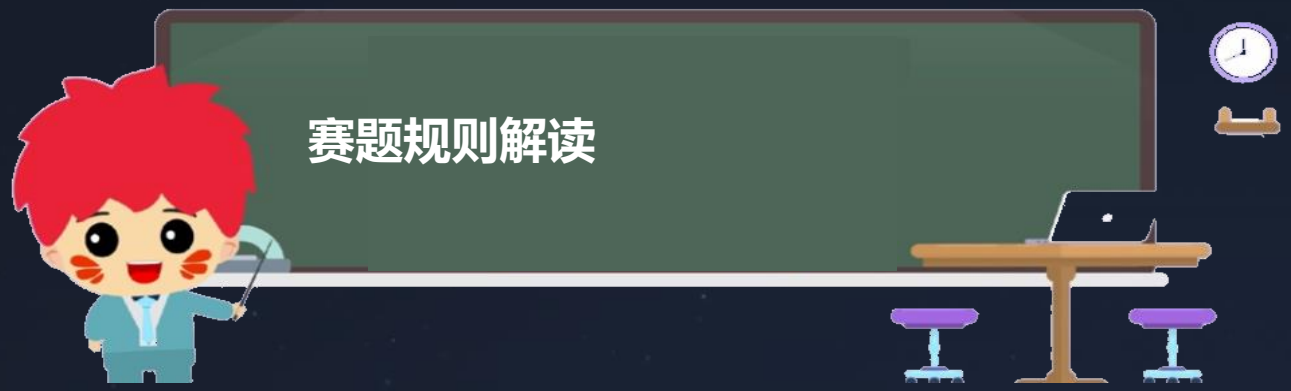
- 车辆驶入道路的优先级

① 直行进入该道路的车辆优先于其他道路转弯进入该道路的车辆；

② 左转进入该道路的车辆优先于其他道路右转进入该道路的车辆；

③ 必须等待其他路口直行或左转进入该道路的车辆行进完毕后，右转进入该道路的车辆才可以进入。

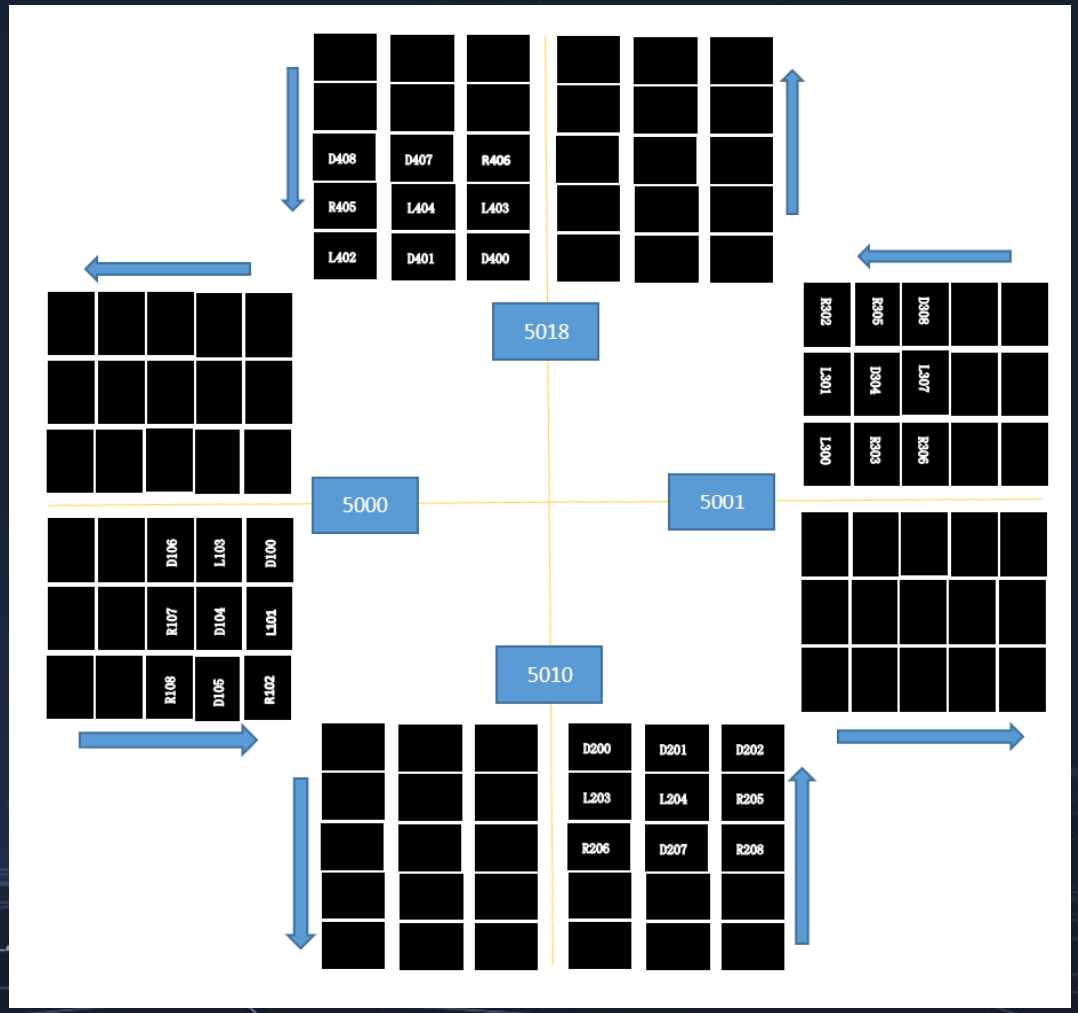
④ 可以进入该道路的直行车辆、左转车辆、右转车辆的优先级只受直行、左转、右转优先级影响，**不受车辆所在位置前后的影响**。

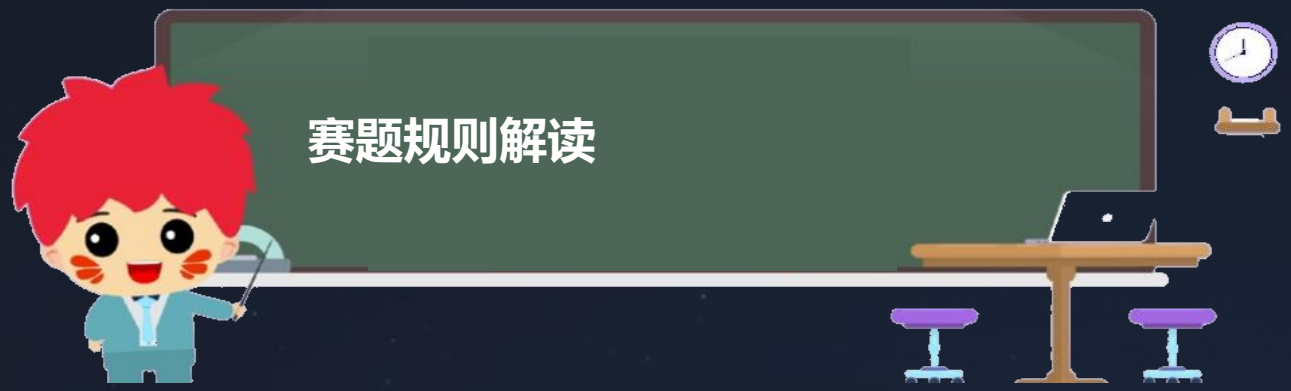


赛题规则解读

车辆调度顺序如下：

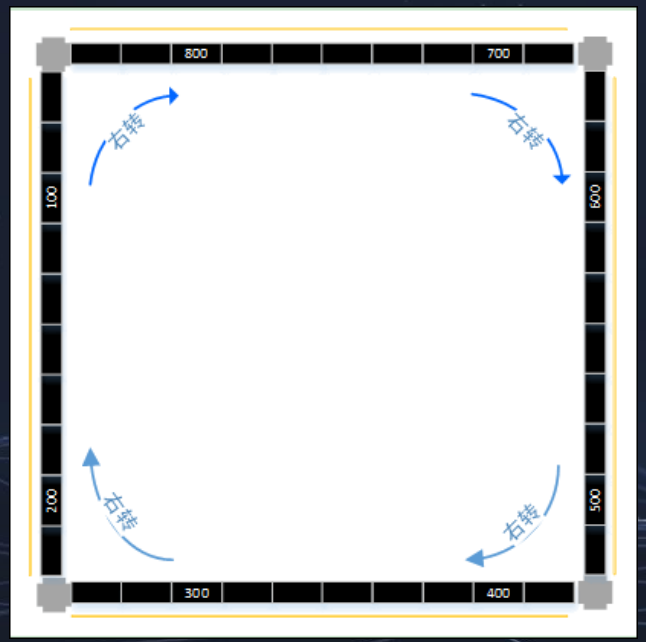
D100、D200、D201、D202、L203、L204、R205、R206、D207、R208、D400、D401、L402、L403、L404、R405、R406、D407、D408、L101、L300、L301、R302、R303、D304、R305、R306、L307、D308、R102、L103、D104、D105、D106、R107、R108

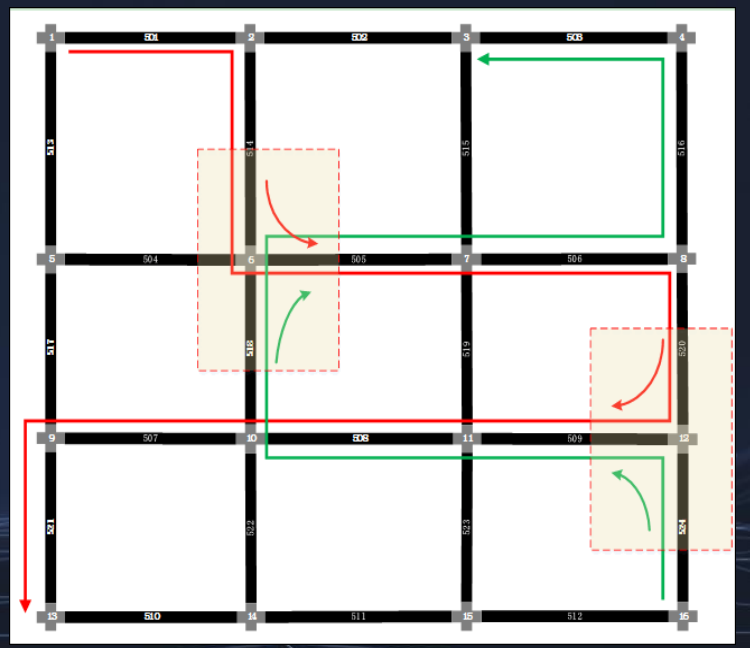




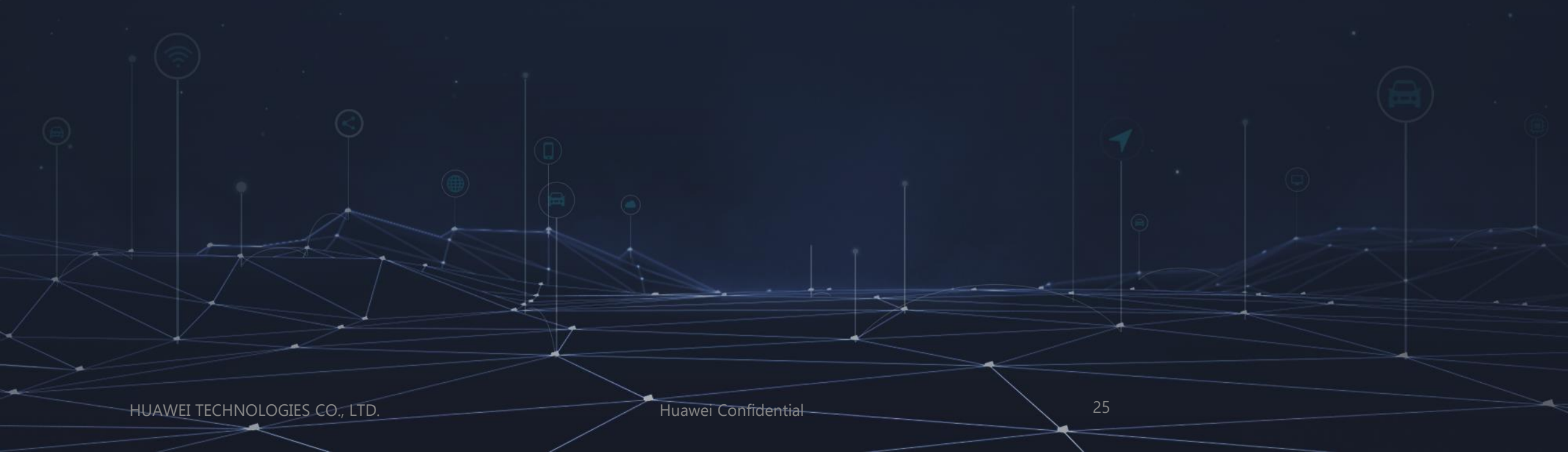
赛题规则解读

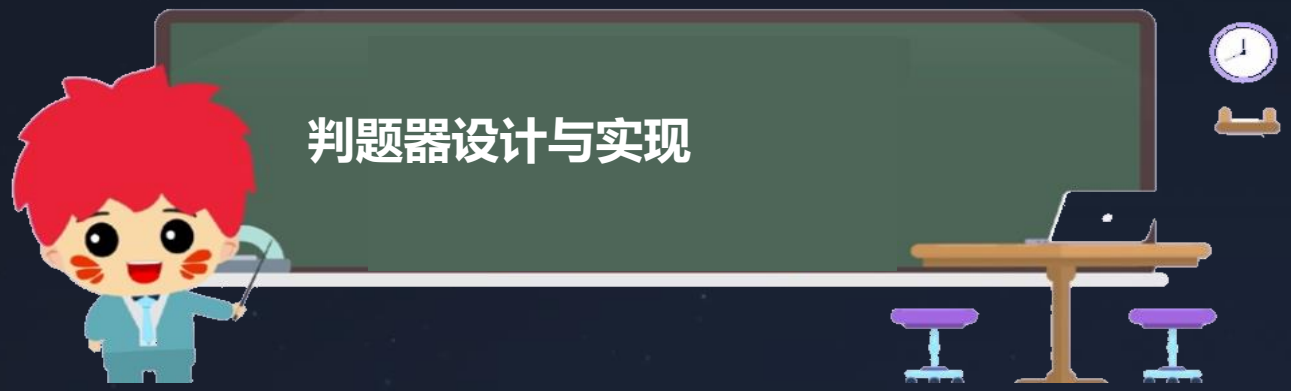
- 死锁





判题器设计与实现

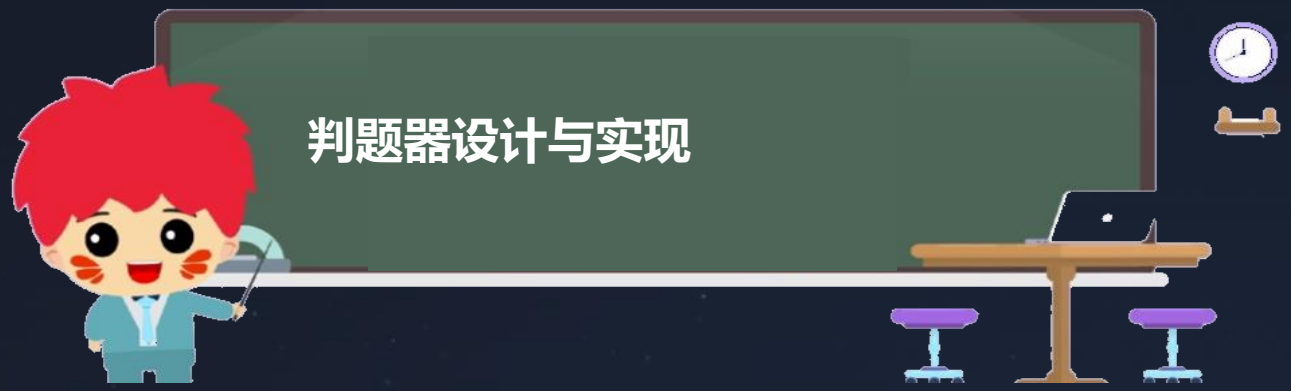




- 系统精度说明

  - 为简化实现，整个系统实现不考虑小数，全部只考虑整数的情况
  - 车辆行驶距离最小行驶距离为1，不考虑小数
  - 系统调度最小单位为1个单位，也就是一个时间片，不考虑小数。（不允许挪车）

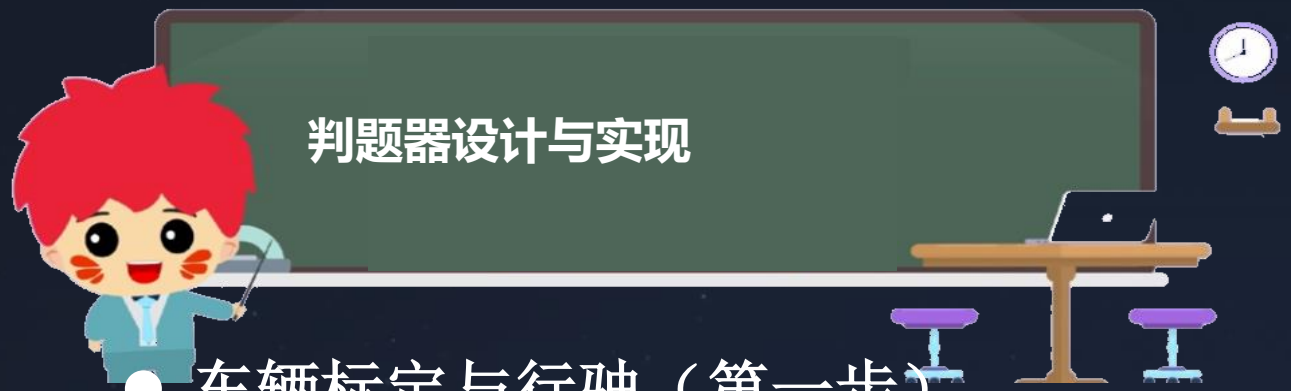
不会出现1/2时间单位，1/3时间单位，1/4时间单位等小于一个时间单位的情况。也就是说车辆要么走一个时间单位，要么不走，不会出现车辆先走1/2时间单位，再走1/2时间单位。比如一车辆的可以行驶的速度为3，则一次调度一个时间单位行驶距离3，不能以1/3时间单位、1/3时间单位、1/3时间单位调度且相应行驶距离1、距离1、距离1的情况。



- 调度优先准则

  - 整个系统按路口ID升序顺序循环调度
  - 同路口多条道路按各条道路的ID升序进行循环调度车辆行驶
  - 一个时间片，优先调度道路上行驶的车辆，再调度等待上路行驶的车辆
  - 系统对于等待上路车辆，按实际出发时间作为主序，车辆ID作为次序的原则调度上路行驶

    对于等待上路的车辆，会出现实际出发时间到了而道路上车辆太多无法上路，导致车辆上路时间延迟



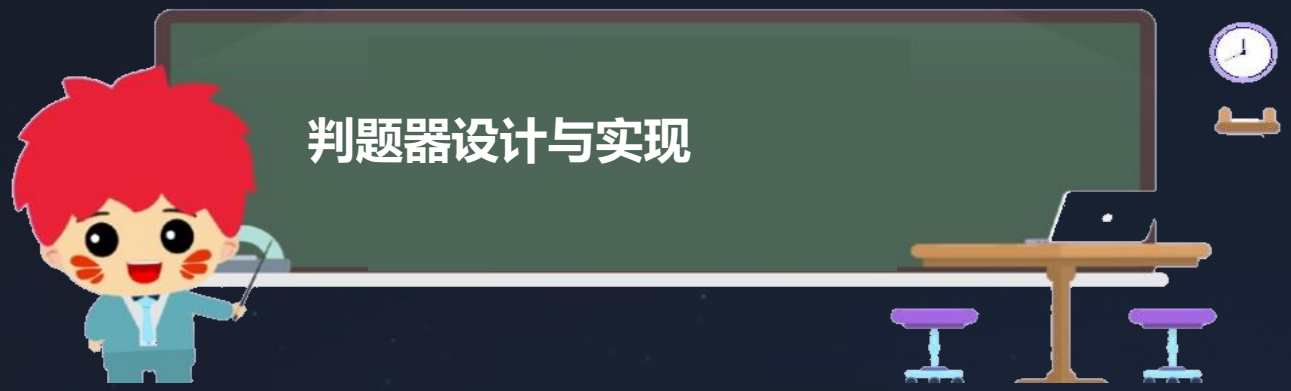
判题器设计与实现

车辆标定与行驶（第一步）

该步骤处理所有道路的车辆不影响其他道路上车辆的顺序，因此先调度哪条道路无关紧要。

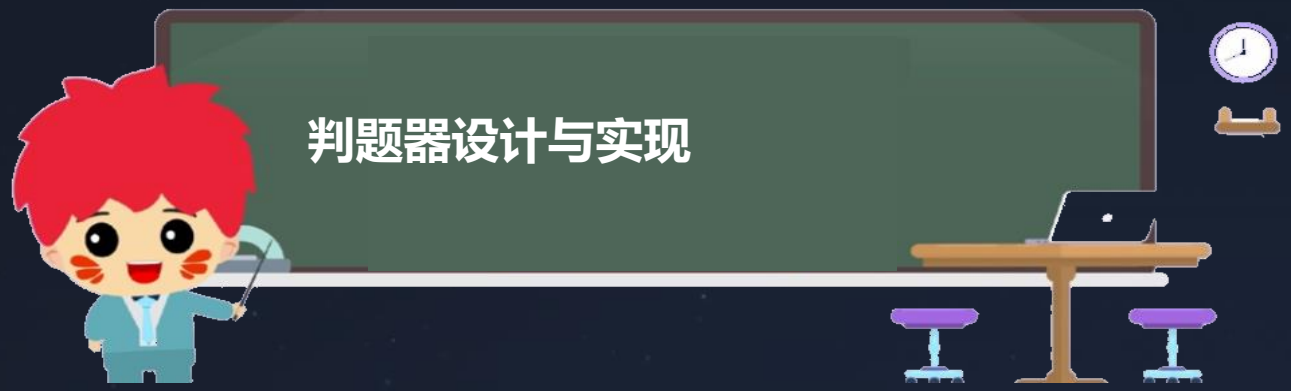
- 先处理每条道路上的车辆，将这些车辆进行遍历扫描，如果车在经过行驶速度（前方没有车辆阻挡）**可以出路口**，将这些车辆标记为**等待行驶车辆**。
- 车辆如果行驶过程中，前方没有阻挡并且也**不会出路口**（v=min(最大车速，道路限速)），则该车辆**行驶可行驶的最大车速**（v=min(最大车速，道路限速)），此时该车辆在本次调度**确定了该时刻的终止位置**。该车辆标记为**终止状态**。
- 车辆如果行驶过程中，发现前方有车辆阻挡，且**阻挡的车辆为等待车辆**，则该辆车也被标记为**等待行驶**车辆。（与阻挡车辆的距离s ＜ v*t)） 其中：v=min(最大车速，道路限速),t=1
- 车辆如果行驶过程中，发现前方有车辆阻挡，且**阻挡的车辆为终止车辆**，则该辆车也被标记为**终止车辆**。（与前方阻挡的车辆的距离记为s）则该车辆最大行驶速度为v = min(最高车速，道路限速，s/t）其中t=1，该车辆最大可行驶距离为s。

遍历道路上车辆由第一排向最后一排进行遍历，确定每辆车的行驶状态。（出道路处为道路第一排，入道路处为是后一排）



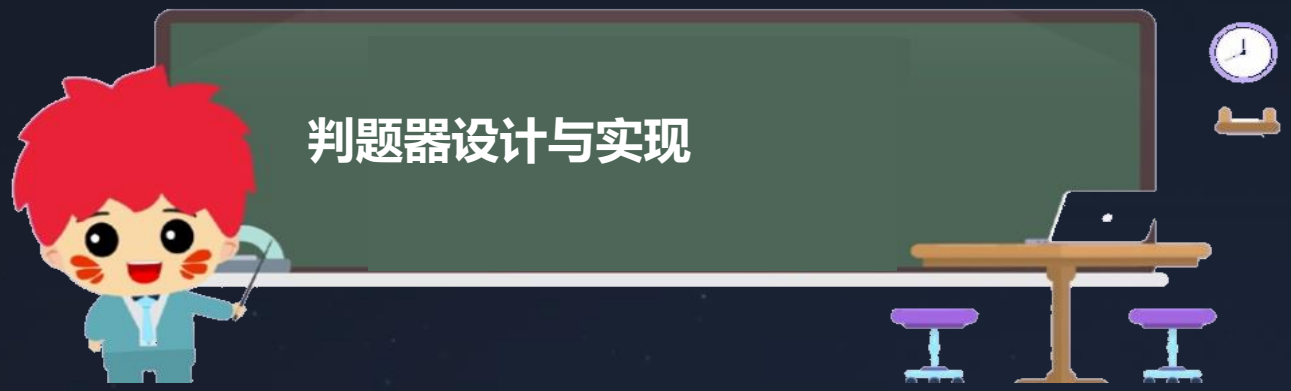
- 车辆等待状态与终止状态的标定

  - 车辆等待状态与终止状态的标定，只与车辆所在的当前道路和当前车辆有关，不与其他任何因素有关系。
  - 也就是不用考虑车辆是否马上到达终点、也不用考虑该车辆在下一条道路上是否可行驶、也不考虑在下一条道路上的车辆状态。



- 车辆到达终点

  - 车辆在到达终点前最后一道路上时，出路口按**直行**处理。
  - 该车辆依然会需要行驶**出当前道路**；到达最后一条道路最后一个位置，不能算作到达终点；
  - 同时出最后一条道路依然参与出路口车辆的**优先级排序**。



- 等待车辆的调度（第二步）

① 整个系统调度按路口ID升序进行调度各个路口，路口内各道路按道路ID升序进行调度

每个路口遍历道路时，只调度该道路出路口的方向。

如图所示：

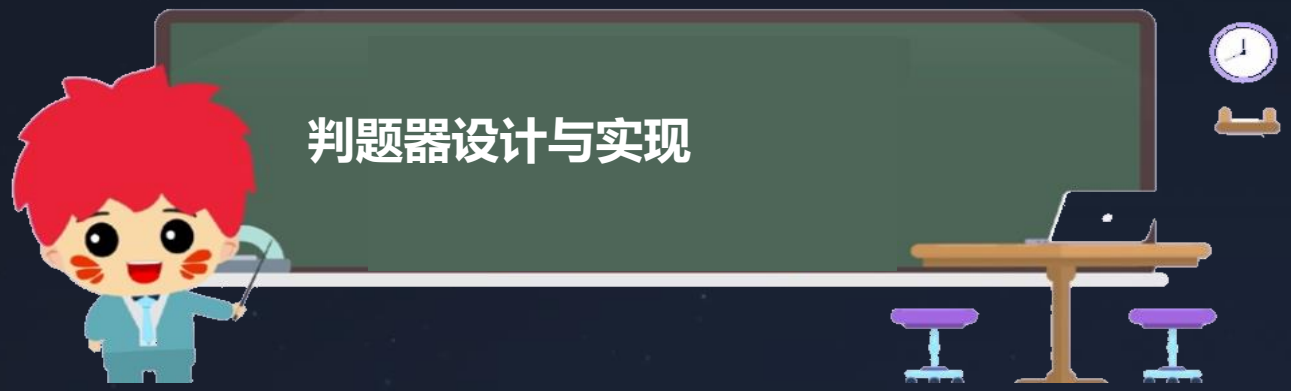
调度路口5时，只调度道路500从路口6到路口5的方向；

调度路口6时，只调度道路500从路口5到路口6的方向。

如：路口5 <-----500------ 路口6

路口5 ------500-----> 路口6

注：道路500的起始点为路口5，终止点为路口6

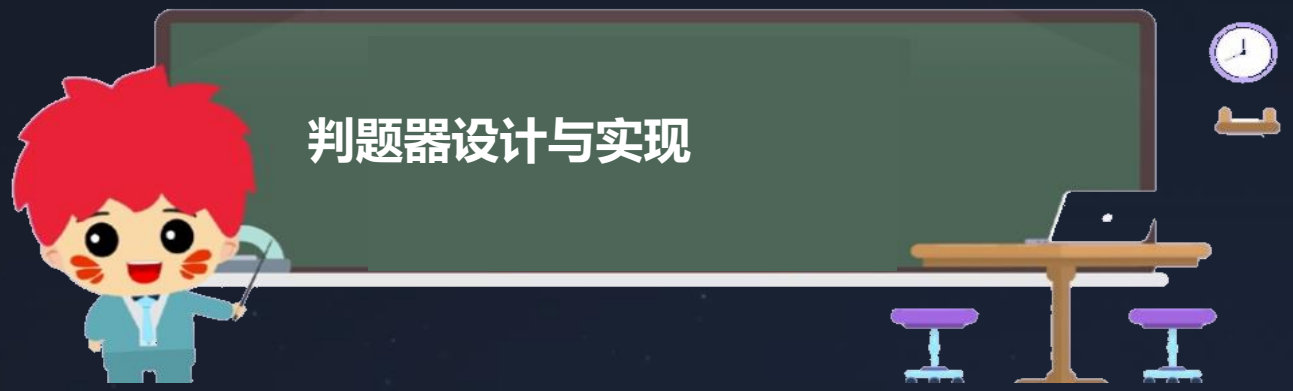


- 等待车辆的调度（第二步）

② 道路内部车辆调度按任务书给定的优先顺序进行调度

- 在每次调度中，调度到的车辆要么行驶其**可行驶的最大车速**，要么就会因等待其他车辆行驶而处于等待行驶状态，待所等待行驶的车辆行驶后，再使该车辆行驶其可行驶的最大车速。
- **是否发生冲突，只与相关道路的第一优先级车辆的行驶方向进行比较，看是否发生冲突**
- 每条道路如果当前道路第一优先级车辆不能行驶，则当前道路后面的的车辆都不能行驶，**只有第一优先级的车辆行驶了，后面第二优先级才可以确定是否可以行驶**。
- 每个道路（道路R）一旦有一辆等待车辆（记为车A，所在车道记为C）通过路口而**成为终止状态**，则会该道路R的车道C上所有车辆进行一次调度，如第一步所示，**仅仅处理该道路该车道上能在该车道内行驶后成为终止状态的车辆**（对于调度后依然是等待状态的车辆不进行调度，且依然标记为等待状态）

  ----尽可能多、尽可能快地将车辆确定为终止状态



● 等待车辆的调度（第二步）

假定道路有如下车辆，车AB车速为3，CD均为车速为1,按步骤一后ABCD均为等待状态。

A空空B空空CD　（路口）

在处理待状态车辆D后，假定D前进到其他道路后，此刻道路状态变化为A空空B空空C空（路口）

接下来因D车辆的前进后，需要对该条道路该车道的所有车辆进行一次调度，只调度经过一次调度会依然在该车道内车辆且状态为终止状态（不出路口）

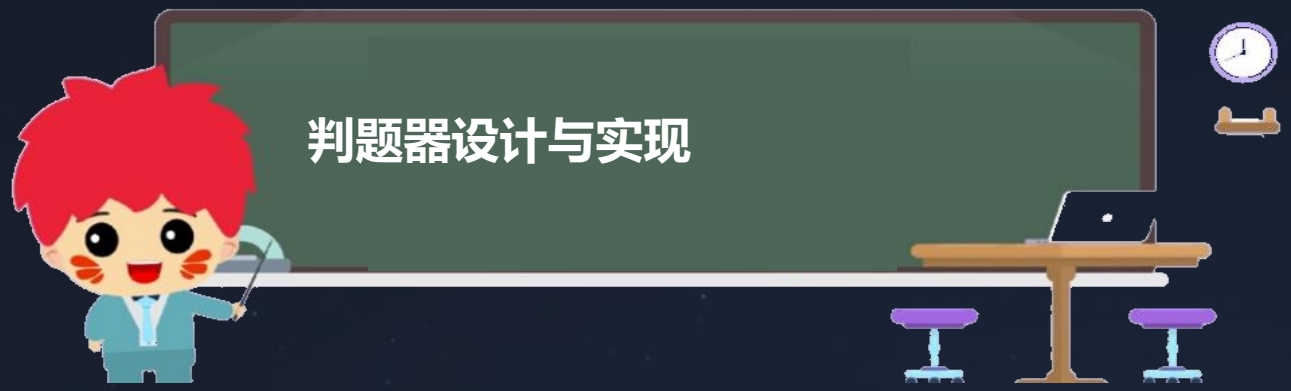
因此：

C的车速为1，则C车前进1个距离，且为终止状态。A空空B空空空C （路口）

B的车速为3，则B车前进3个距离，且为终止状态。A空空空空空BC （路口）

A的车速为3，则A车前进3个距离，且为终止状态。空空空A空空BC （路口）

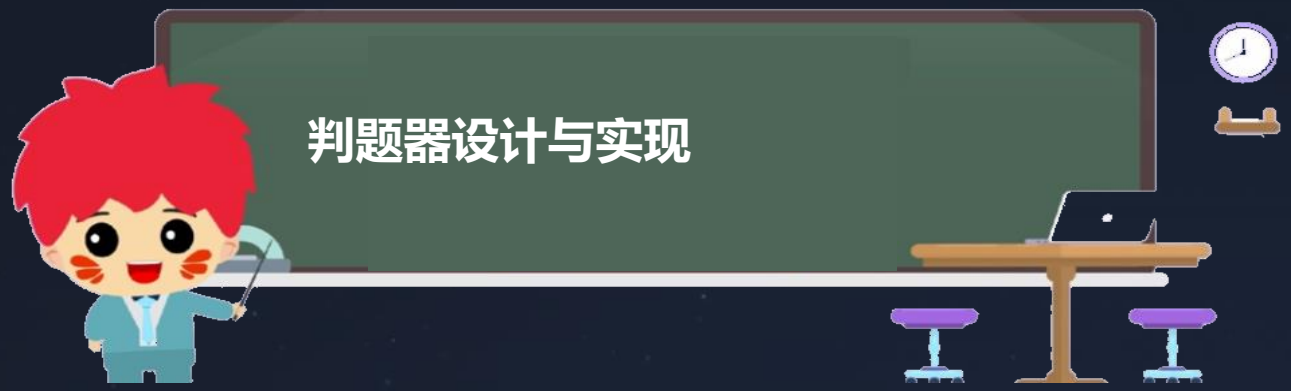
调度后道路车辆分布为：空空空A空空BC （路口），且ABC均为终止状态



- 等待车辆的调度（第二步）

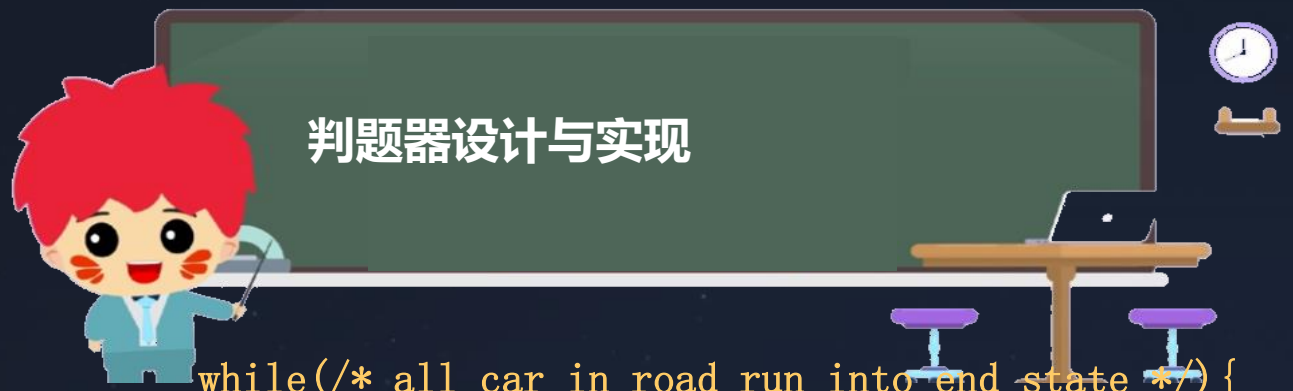
③ 重复调度时①与②，直到所有车辆达到终止状态或死锁状态

- 一个时间单位内，一次调度时①与②的调度会出现多次重复调度，因每次调度有可能因为等待其他道路车辆的行驶而导致当前车辆无法行驶，因此**会循环调度**使所有车辆行驶一个时间单位
- 每次调度到一条道路，直到该道路无车辆可调度，或该条道路上车辆处于冲突状态。也就是说**尽可能多地**让该道路上的车辆行驶，直到没有车辆或者车辆与其他车辆发生冲突不可行驶。
- 如果本次循环因冲突失去调度权限，则需要等下一次循环，此道路才会获得调度权限。当一个条道路获得调度权限时，**尽可能多地**让此道路的车辆进行行驶。
- 依次按道路ID升序调度该路口所连接的所有道路，再次循环该路口下所有道路，直到所有道路车辆**全部处于终态或死锁状态**



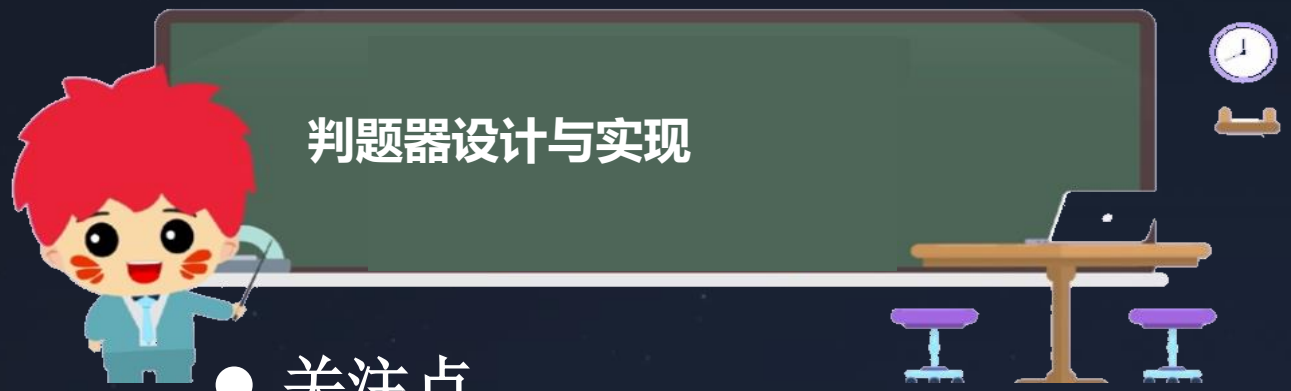
- 伪码

```
for(/* 按时间片处理 */) {
    foreach(roads) {
            driveAllCarJustOnRoadToEndState(allChannel);
    }
    …(未完待续)
```



判题器设计与实现

```
while(/* all car in road run into end state */){
    foreach(crosses){
        foreach(roads){
          while(/* wait car on the road */){
              Direction dir = getDirection();
              Car car = getCarFromRoad(road, dir);
              if (conflict){
                  break;
              }
              channel = car.getChannel();
              if(!car.moveToNextRoad()) { break;}
              driveAllCarJustOnRoadToEndState(channel);
          }
        }
    }
}
driveCarInGarage();
}
```



判题器设计与实现

- 关注点

① **性能：**大量车与路口，而且嵌套多层循环，性能需要重点考虑

② **死锁检测：**一个时间片后所有等待车辆的状态未发生改变/所有路口的遍历超过经过一定量级（最差情况下不小于车辆数目）/等待车辆的数量未发生变化

③ 车辆上路行驶：车辆初始上路即可行驶其可行驶的最大速度

④ 车辆到达终点：终点为路口，非道路的最前一格位置，车辆需要行驶出当前路口

⑤ 车辆受下一道路的影响：车辆也只有行驶出路口才可以判定其是否受下一道路限速的影响停留在本道路的最前面一格

⑥ **冲突检测：**只与相冲突道路的第一优先级车辆对比/只检测进入相同车道的道路情况

⑦ **优先级排序：**只有出路口的车辆才参与优先级排序

祝大家取得优异的成绩！


Run the World
Code Craft
THANKS!