华为软件精英挑战赛技术方案

# 基本实现路径

#初始化对象

创建所有的car对象,保存为一个列表

创建所有的road对象，保存为一个列表

创建所有的cross对象，保存为一个列表

创建地图，保存为一个列表

根据car对象的出发地点，初始化cross的未出发车库；

#进入主体调度逻辑

For 每一个时间片：

While(道路上还有等待态的车辆)：

For 每一个路口：

#处理当前路口终止状态的车辆

#标记，是根据下一时刻的位置是否确定来判断的

For 当前路口的每一条道路：

If 当前道路是单向且道路终点是当前路口:

选择道路中存储开向当前路口的队列

进入道路上车辆的终止状态处理函数（roadID）

elif 当前道路是双向道路：

选择道路中存储开向当前路口的队列

进入道路上车辆的终止状态处理函数（roadID）

else:

转向下一条道路

#处理当前路口的等待状态车辆：

While(当前路口还有车辆可以通过路口)：

For 当前路口的每一条道路：

If 当前道路是单向且道路终点是当前路口:

进入道路上车辆的等待状态处理函数(crossID, roadID)

elif 当前道路是双向道路：

选择道路中存储开向当前路口的队列

进入道路上车辆的等待状态处理函数(crossID, roadID)

else:

转向下一条道路

#处理当前路口还未出发的车辆

进入当前路口未出发车辆的处理函数（crossID）

#处理完所以路口以后，检查车辆是否都已经到达目的地

进入调度检查函数，返回是否所有车辆都已到达目的地

#调度结束，输出调度时间、每辆车的路径

进入结果输出函数

# 基本类的定义和接口定义

## 调度系统的类实现：schedual\_system\_class

class chedual\_system:

#创建

def \_\_init\_\_(self,carlist,roadlist,crosslist,trans\_map):

self.carlist = carlist #车辆列表，存储着所有的车辆对象

self.roadlist = roadlist #道路列表，存储着所有的道路对象

self.crosslist = crosslist #路口列表，存储着所有的路口对象

self.trans\_map = trans\_map #存储着地图对象

self.time = 0 #系统时间

#主调度函数

def dispatch\_system(self):

pass

#道路上等待状态车辆的处理函数，更新等待态车辆状态，道路状态

def wait\_state\_function(self, crossID,roadID):

#获取当前道路对象

road = self.roadlist[roadID]

#获取当前道路的终点为当前路口的车辆队列

road\_quene = road.get\_quene(crossID) #根据路口ID选择

#处理每一辆

for car in road\_quene.yield\_car():

if car.car\_cur\_state == ‘wait\_state’ :

if 当前车辆能走完当前道路：

if 当前车辆的目的地是当前路口：

修改车辆状态，将车入库

elif 目标道路允许车辆进入：

if 没有车辆冲突：

修改车辆状态，出队，入队

else:

break #退出当前的等待态处理函数

else: #目标道路不允许进入

更新车辆状态

break #转向下一条道路

elif 没有前车阻挡：

更新车辆状态

elif 有前车阻挡,且前车为等待态：

更新车辆为等待态

else: #有前车阻挡，且前车为终止态：

更新车辆状态，为终止态

else:

break

#道路上终止状态车辆的处理函数,更新终止态车辆状态

def end\_state\_function(self,crossID,roadID):

#获取当前道路对象

road = self.roadlist[roadID]

#获取当前道路的终点为当前路口的车辆队列

road\_quene = road.get\_quene(crossID) #根据路口ID选择

#处理每一辆车

for car in road\_quene.yield\_car():

if car.car\_cur\_state == ‘waite\_state’:

continue

else:

if 可以走完当前道路的话：

可以走完当前道路，标记为等待态

if 前面有车阻挡，且前车是等待态：

标记为等待态

if 前面有车阻挡，且前车为终止态：

更新车辆状态（位置，时间），并标记为终止态

if 前面没有车阻挡：

更新车的状态（位置，时间），并标记为终止态

## 二）车辆相关类的定义

### 单个车辆的类：car\_class

#### 抽象数据结构

class car\_class:

#创建

def \_\_init\_\_(self,car):

self.car\_info = car #车辆的基本信息

self.car\_start\_time = None #车辆的实际出发时间

self.car\_cur\_time = None #车辆的当前时间

self.car\_final\_time = None #车辆的到达时间

self.car\_realspeed = None #车辆的实际速度

self.car\_planpath = [] #为车辆规划的路径

self.car\_realpath = []#车俩实际走过的路径，进入道路时，添加道路ID

self.car\_direction = #车辆在路口的行进方向，straight、left、right

self.car\_end\_distacne = #车辆距离当前道路的出口距离

#self.car\_begin\_distance = #车辆距离当前道路的入口距离

self.car\_cur\_state = #车辆的状态，wait\_state、end\_state

#设置车辆出发时间

def set\_car\_start\_time(self,start\_time):

if self.car\_start\_time is None:

self.car\_start\_time = start\_time

#设置车辆当前时间

def set\_car\_start\_time(self,cur\_time):

self.car\_start\_time = cur\_time

#设置车辆到达

def set\_car\_final\_time(self,final\_time):

if self.car\_start\_time is None:

self.car\_final\_time = final\_time

#车辆实际速度

def set\_car\_realspeed(self,speed):

self.car\_realspeed = speed

#规划车辆路径

def plan\_car\_route(self,地图,调度系统信息)

pass

#添加道路到真实路径

def add\_roadID\_realpath(self,roadID):

self.realpath.append(roadID)

#设置车辆的行进方向，根据当前道路，目标道路，路口信息

def set\_car\_direction(self,cur\_road,target\_road,cross):

pass

#设置车辆的状态

def set\_car\_state(self,state):

self.car\_state = state

#获取车辆的目标道路,返回目标道路ID:roadID

def get\_target\_road(self):

pass

#################################################

#为车辆的状态更新函数服务的方法

#################################################

#检查车辆等否走完当前道路

def can\_pass\_road(self):

#通过比较当前速度和距离道路出口距离

#检查当前路口是否是车辆的目的地

def is\_end\_cross(self,crossID):

#比较车辆的目的路口ID和当前路口ID

#车辆的出发的函数

#终止状态车辆的状态更改函数

#等待状态车辆的状态更改函数

#车辆的方向设置函数

#### 重要函数的实现分析：

1. #车辆的出发的函数

出发车辆需要设置修改的参数：车辆的出发时间，计划路径，实际路径，车辆所在的道路，车辆距道路出口的长度，车辆状态，真实速度，方向

处理逻辑：其实质也就是修改车辆的属性；

*#车辆出发函数  
#需要配置这些参数(车辆的出发时间，计划路径，实际路径，车辆所在的道路，车辆距道路出口的长度，车辆状态，真实速度，方向）***def** start\_car(self,cur\_time,\*,roadlist,map\_info,schedual\_system\_info,distance\_tofrontcar):  
 *#赋值初始时间* self.car\_start\_time = cur\_time  
 *#规划车辆的路径* start\_cross = self.car\_baseinfo[1]  
 end\_cross = self.car\_baseinfo[2]  
 self.car\_planpath = self.plan\_car\_route(start\_cross,end\_cross,map\_info,schedual\_system\_info)  
 *#获取道路信息，因此car\_planpath中存储的是道路ID,因此可以通过道路的存储字典来获取道路信息  
 #road = [id,length,speed,channel,from,to,isDuplex]* road = roadlist[self.car\_planpath[0]]  
 *#赋值车辆进入的道路* self.car\_cur\_road = self.car\_planpath[0]  
 *# 赋值真实路径* self.car\_realpath.append(self.car\_cur\_road)  
 *#赋值车辆的速度* self.car\_realspeed = min(self.car\_baseinfo[3],road[2],distance\_tofrontcar)  
 *#更改车辆的状态，因为可以出发，其出发了一定是终止态* self.car\_cur\_state = 1 *#置为终止态  
 ## 更改车距离道路出口的长度* self.car\_end\_distacne = road[1] - self.car\_realspeed

1. #终止状态车辆的状态更改函数

终止状态车辆状态更改函数,改变有两种可能性:一种改为等待态；二：仍是终止态；

车辆需要设置修改的参数：

这需要分为两种情况：①改为等待态：cur\_state; ②保持终止态：car\_end\_distance；

1. #等待状态车辆的状态更改函数

等待状态车辆需要设置修改的参数：

这需要分为三种情况：①到达车库：final\_time, end\_distance；②通过路口，改为终止态：planpath（可能），cur\_road，realpath，realspeed，end\_distance，cur\_state，方向(方向是需要在路口确定目标道路后才能确定)；③不通过路口，改为终止态：end\_distance,cur\_state；

车路路径的重新规划与否需要判断道路情况，或者是否死锁；在改变等待态车辆状态之前就应该规划好路径了，而不是在修改等待状态是更新路径。

1. #车辆的方向设置函数

根据当前道路、目标道路，在路口的关系来确定方向

#### car\_class小结

attr:

*#car = [id,from,to,speed,planTime]*self.car\_baseinfo = car *# 车辆的基本信息*self.car\_start\_time = **None** *# 车辆的实际出发时间  
#self.car\_cur\_time = None # 车辆的当前时间*self.car\_final\_time = **None** *# 车辆的到达时间*self.car\_realspeed = **None** *# 车辆的实际速度*self.car\_planpath = [] *# 为车辆规划的路径，路径是道路ID  
#self.car\_plan\_crosspath = [] # 为车辆规划的路径,路径是路口节点*self.car\_realpath = [] *# 车俩实际走过的路径，进入道路时，添加道路ID*self.car\_direction = -1 *# 车辆在路口的行进方向，straight:-1;left:0;right:1;*self.car\_cur\_road = **None** *# 车当前所在的道路*self.car\_end\_distacne = **None** *# 车辆距离当前道路的出口距离  
# self.car\_begin\_distance = #车辆距离当前道路的入口距离*

*# 车辆的状态，wait\_state：-1 ；end\_state：1；在车库状态：0；*self.car\_cur\_state =0

attr:

*##关键函数*

start\_car(self,\*\*kwargs) *#车辆出发函数*

change\_end\_state\_car(self,target\_state) *#终止状态车辆修改函数*

change\_wait\_state\_car(self,\*\*kwargs) *#等待状态车辆修改函数*

plan\_car\_route(self,start\_cross,end\_cross,map\_info,schedual\_system\_info)

*#规划路径*

*##辅助函数*

find\_target\_roadID(self,has\_replanpath)#*找下一条道路*

set\_car\_direction(self,cross\_baseinfo,target\_roadID)*#设置车辆的转向*

method:

## 三）道路相关类的定义

### 1、单个道路的类：road\_class

#### 抽象数据结构

Class road\_class:

#创建

def \_\_init\_\_(self,road)

self.road\_info = road #道路基本信息

self.road\_cur\_time = None #道路当前时间

self.road\_quenes = [] #道路的车辆队列

#设置道路当前时间

def set\_road\_cur\_time(self,cur\_time):

self.road\_cur\_time = cur\_time

#向道路入队一辆车,distance\_end:车辆到道路出口的距离

def enquene(self,car\_ID,distance\_end ):

pass

#道路弹出一辆车，返回carID，distance\_end

def dequene(self, ):

pass

#访问道路第一优先级的车辆，返回carID，distance\_end

def first\_car(self):

pass

#访问道路最低优先级的车辆，返回carID，distance\_begin

Def last\_car(self): #distance\_begin是车辆到道路入口的距离

pass

#按照道路上车辆的优先级，迭代遍历每一辆车，为车辆遍历服务

def yield\_car(self): #返回carID、distance\_end:到道路出口的距离

pass

#### 重要的方法

#将车辆加入道路

#将车辆弹出道路

#访问道路第一优先级的车辆

#访问道路最低优先级的车辆

#按照优先级顺序遍历访问道路的每一辆车

#### cross\_class小结

self.road\_baseinfo = road  
self.from\_to\_quene = []  
self.to\_from\_quene = []

attr:

*##关键函数*

*#遍历道路上的每一辆车*yield\_car(self, crossID)

*#弹出道路上往当前路口行驶的最高优先级的车辆：*dequene(self,crossID):

*#将车辆加入道路的某个车道***d**enqune\_car(self, carID, distance\_end, crossID, channelID):

*#访问道路上往当前路口行驶的最高优先级的车辆：道路上有车辆就返回：队列元素；道路上没有车辆就返回：-1*first\_car(self,crossID):

*#访问道路上往当前路口行驶的,每个车道最后的一辆车：返回：车辆、车道号; 或者-1;*last\_car(self,crossID):

method:

### 2、道路上队列的类：road\_quenes

道路上队列的要求：道路的每一个方向都有一个队列，这个队列要管理这个方向的所有车道的车辆，这个队列中车辆排序的方法是：先按照‘距离道路出口的距离’排序，再按照车道ID排序。

这样才能保证道路整个方向上的车辆优先级顺序。

## 四）路口相关类的定义

### 1、单个路口的类：cross\_class

#### 抽象数据类型

class cross\_class:

#创建

def \_\_init\_\_(self,cross):

self.cross\_info = cross #路口的基本信息

self.cross\_cur\_time = None #路口当前的时间

self.cross\_begin\_carport = #未出发车辆的车库

self.cross\_end\_carport = #到达目的地车辆的车库

#设置路口的当前时间

def set\_ cross\_cur\_time(self,cur\_time):

self. cross\_cur\_time = cur\_time

#将未出发的车辆加入路口的未出发车库

def enquene\_begin\_caport(self,elem):

pass

#访问未出发车库的第一优先级车辆，返回未出发车辆的carID

def peek\_begin\_carport(self):

pass

#弹出未出发车库的第一优先级车辆,返回未出发车辆的carID

def dequene\_begin\_carport(self):

pass

#将到达目的地的车辆加入’到达目的地车库’

def enquene\_end\_carport(self,carID):

pass

#### cross\_class小结

self.cross\_baseinfo = crosslist *#基本信息*self.cross\_cur\_time = 0  
*# 存未出发的车辆，这个结构中存储的元素是先按照plantime排序从大到小，再按照carID排序从大到小,*self.cross\_begin\_carport = [] *#每一个元素是[plantime,carID]*self.cross\_end\_carport = [] *#存到达目的地的车辆 ,直接用列表存储*

attr:

*##关键函数*

enquene\_begin\_carport(self,elem) #*将车辆加入路口的‘未出发车库’*

dequene\_begin\_carport(self) *#弹出未出发的车辆*

peek\_begin\_carport(self) *#访问未出发车库第一优先级元素*

enquene\_end\_carport(self,carID) *将到达目的的车辆加入‘到达目的地车库’*

method:

### 2、路口车库的类：cross\_quene

未出发车库要求：把车库的车先按照出发时间从小到大排序，再按照车辆ID编号排序。

## 五）地图相关的类

### 地图：map\_class

#### 抽象数据结构

class map\_class:

#创建地图

def \_\_init\_\_(self,road\_list,cross\_list,direct\_num = 4,unconn = 1000):

cross\_num = len(cross\_list) #获取路口数  
self.map\_cross\_num = cross\_num #地图节点数  
self.map\_unconn = unconn #非连接节点默认权值  
self.map\_adjlist = None #地图，邻接表结构  
if cross\_num > 0:  
 map\_ =[None]\*cross\_num  
 self.\_adjlist= Map.build\_adjlist(map\_,road\_list,cross\_list)

self.map\_adjmatrix = map\_class.build\_adjmatrix() #地图邻接矩阵

#建立地图,输入道路列表、路口列表，返回一个地图的邻接表

@staticmethod

def build\_adjlist(head,road\_list,cross\_list):

pass

#检查地图是否为空

def is\_empty(self):

return self.map\_head is None

#获取一条边,返回道路ID

**def** get\_edge(self,cross\_i,cross\_j):

pass

#构建邻接矩阵

def build\_adjmatrix(self):

pass

#利用Floyd算法计算所有路口之间的最短路径，通过邻接矩阵计算

def all\_shortest\_path(self):

#返回最短路径长度矩阵、最短路径矩阵

pass

#根据最短路径矩阵，计算返回任意两个节点间的最短路径

def get\_path(self,start\_crossID,end\_crossID):

#返回一个从起点到终点的节点列表

Pass

# 三、重要函数实现逻辑

## 一）道路上终止状态车辆处理函数

#道路上等待状态车辆的处理函数，更新等待态车辆状态，道路状态

## 道路上等待状态车辆处理函数

While（有死锁）：

对每一个路口：

While（还有车可以通过路口）：

对每一条道路：

If 不是开向当前路口的道路：

Continue

Else:

For 当前道路的每一辆车：

## 未出发车辆出库的处理函数