公共交通调度系统概要设计

学院：国际学院

班级： 2021215120

姓名： 韩志浩

北京邮电大学

目录

[一、公交车调度系统问题描述 3](#_Toc104155511)

[二、需求分析 3](#_Toc104155512)

[2.1 公交车轨道配置 3](#_Toc104155513)

[2.2 用户请求配置 4](#_Toc104155514)

[2.3 公交车行驶规则与要求 5](#_Toc104155515)

[**2.3.2 处理上下车请求的方法(怎么处理当前站点的请求)** 5](#_Toc104155516)

[2.4 公交车调度策略配置 5](#_Toc104155517)

[三、概要设计 8](#_Toc104155518)

[3.1 输入输出设计 8](#_Toc104155519)

[3.2 自动机模型 9](#_Toc104155520)

[3.3 高层数据结构设计 11](#_Toc104155521)

[3.4 系统模块划分 12](#_Toc104155522)

[3.5 高层算法设计 17](#_Toc104155523)

## 一、公交车调度系统问题描述

有一个闭环的公交车轨道，只有一辆公交车在轨道上行驶，沿途共有N个公交车站，车站间的距离相等。公交车有顺时针和逆时针两种方向可供调度。调度策略有先来先服务（FCFS）、寻找最短时间优先算法（SSTF）、顺便服务策略三种。

图片包含 图示

描述已自动生成

图1：公交车行驶路线图

现在，要求我们用程序模拟一个城市自动驾驶公交车调度系统：程序初始，默认一辆公交车从轨道的左上角发车，根据调度策略确定行驶方向，按设定车速S行驶（定时刷新显示公交车的当前位置和“时间”显示，时间以秒为单位），途经有乘车请求的车站时停车P秒，完成本站的所有请求，刷新显示“已完成请求数”和“最大等待时间”指标。

由程序接收动态发出（读文件或键盘、鼠标操作）的“停运”指令，收到停运指令后，程序不再接收新的乘车请求（准备收车状态），但需要将已有乘车请求处理完，然后更新“已完成请求数”和“最大等待时间”指标，程序退出。程序运行期间还需要计算公交车运行时所有请求完成时间的最大值。

## 二、需求分析

### 2.1 公交车轨道配置

#### 2.1.1 道路基本类型

本程序模拟的公交车运行路线为双向单条车道。在此道路中，车辆可以在同一条道路行朝顺时针和逆时针两个方向行驶，并且忽略调头时间。

图示

描述已自动生成并且默认未到达站点时无法进行进行掉头。

图2：道路行驶方向图（实际上只有一条车道，这里为了方便，表示为两条车道）

#### 2.1.2 车站基本类型

本程序要求道路上有N个车站，每个车站之间的距离相等，即车站在道路上均匀分布。每个车站都可以对公交车发送上车请求，并且请求中包含顾客期望的方向信息。

### 2.2 用户请求配置

如果要把用户精准的送到目的地，那么我们就需要三种数据信息，首先是用户的出发点，即当前车站，然后是用户的乘车方向，最后是用户的终点站，即图表

描述已自动生成目标车站。

图3：车站状态图

#### 2.2.1 用户上车请求基本类型

用户上车请求包含用户的当前站的序号以及目标站与当前站之间的方向信息。

#### 2.2.2 用户下车请求基本类型

用户下车请求包含用户终点站的序号。

#### 2.2.3 请求与请求之间的关系

上车请求指明了用户所在站点序号和用户目标站点的方向信息，而下车请求指明了用户的目标站序号，两种请求互相独立，但是互相补充。

### 2.3 公交车行驶规则与要求

#### 2.3.1 公交车的表示

在程序实现的过程中（即代码部分），我们采用结构体的方式来表示公交手机屏幕截图

低可信度描述已自动生成车。而在可视化界面中，我们使用绿点表示。

图4：公交车以及行驶路线表示图

2.3.2 处理上下车请求的方法**(**怎么处理当前站点的请求)

在调度策略的指导下，如果公交车停下的站点有乘客请求，那么公交车需要花费一秒的时间进行上客/下客，并且假定一秒可以完成所有的请求。

#### 2.3.3 公交车车速设计要求

公交车以恒定的速度沿着轨道行驶，在程序中以像素/秒为单位。

#### 2.3.4 公交车辆的数量

公交车的数量规定为1。

### 2.4 公交车调度策略配置

在该模块中，将展示公交车到达站点时的决策流程。

#### 2.4.1 FCFS（先来先服务）策略

对于先来先服务策略，车一次停站只完成一个请求，即使在这个站点上即有乘车请求，车内也有到该站的请求也只能按算法完成其中的一个。但是如果下一个请求恰好在同一站点，则可以一次停站完成2个或2个以上的请求。也就是说只停1秒完成多个请求。

伪代码：

int FCFS() {

if 任务链表为空

then 返回静止状态 end

if 当前车站等于任务链表头车站

then 处理该车站所有请求 end

寻找最近到达链表头车站的方向

将该方向设置为公交车方向（方向相同则为顺时针）

}

#### 2.4.2 SSTF（最短寻找时间优先）策略

对于最短寻找时间优先策略，一次服务的请求（目标）一旦确定，即使中途产生更优的请求也不可以更改。但如果新的请求恰好可以顺便服务（同方向的站台请求或车内请求），可以为新的请求停站。当车到达目标地点时，该站点恰好有两个相反的请求，可以停一次车完成这2个服务。

int SSTF(){

if 任务链表为空 then 返回静止状态 end

if 当前有合理的上车请求

then 执行顺便服务 end

if 当前有目标车站

then 返回当前方向 end

else 寻找最近车站并设置为目标车站，并且将最近方向设为公交车方向（方向相同则为顺时针） end

#### 2.4.3 SCAN（顺便服务）策略

对于顺便服务策略，第一次行驶方向由第一个请求出现后1秒钟内全部请求中时间最短的那个请求决定。在行使过程中，如果所有的请求按照当前的行驶方向找出的最短完成时间都超过跑完轨道一半距离时间时，应该切换行驶方向。这是唯一的一个切换方向的规则。车辆行驶过程中如果经过的站点有服务请求，则不管这个请求的类型一律停站，并认为此请求完成。这意味着一次停车可能完成3个服务。

int SCAN(){

minTime=走完总路程的时间

tempStation=0

if 任务链表为空

then 返回静止状态 end

else if 当前公交车静止

then 寻找最近车站并确定对应方向 end

else{

while(当前任务 != NULL) {

if (在任务链表中有关于本站点的上车请求)

返回上客行动;

if (Bus目标站为本站)

返回下客行动;

if minTime大于完成该任务最短时间

then minTime=该任务最短时间 并更新tempStation

当前任务=下一个任务

}

}

if ( minTime>走完一半路程所用时间)

改变公交车当前方向并且再次执行SCAN();

}

## 三、概要设计

### 3.1 输入输出设计

#### 3.1.1 核心版本

输入部分由Input.c模块完成，其功能为从配置文件之中读取公交车运行的初始化配置，存入全局变量int totalStation、int Distance、int Strategy之中。然后再继续读取公交车的运行状态，并返回到Main.c主要运行文件之中并创建Bus和Mission链表进行储存。

输出部分由Output.c模块完成，其功能为从Mission链表和Bus链表之中读取公交车的任务和当前状态，并输出到output.txt文件中。每经过一个clock都会对此模块中的WriteState()函数进行调用并进行一次输出。

输入模块伪代码为：

读取每行第一个字符：

若为#：读掉整行

若为D：独取DISTANCE

若为S: 读取STRATEGY

若为T: 读取TOTAL\_STATION

若为其他：

为c：判断为clock，返回0

counterclockwise、clockwise，读取目标车站存入mission链表，返回1

为t：读取目标车站存入mission链表，返回1

为e:返回-1，mission链表结束

输出模块伪代码为：

void WriteState() {

while (当前任务存在) {

if (目标车站)

把Targrt数组中的0改为1；

if (是顺时针方向请求)

把Clockwise数组中的0改为1;

else if (是逆时针方向请求)

把Countclockwise数组中的0改为1;

进入下一个任务链表节点;

}

输出TIME；

TIME++；

输出BUS;

输出position;

输出Target数组;

输出STATION;

输出clockwise数组;

输出Countclockwise数组;

}

#### 3.1.2 进阶版本

# 基于单服务台负指数分布M/M/1排队系统的策略优化——以公交车调度为模型

在本公交车调度的事例中，假设该公交车满足排队论M/M/1的规则，即：

1、输入过程：顾客源是无限随机的，逐个到来，服从泊松分布；

2、服务机构：单服务台，服务时间的长短随机，服从泊松分布。

我们的工作是融合FCFS，SSCF，SCAN三大算法，并且结合实际客源情形、公交车拥挤程度，提出新的算法PONW（Policy Optimization of Nonlinear Weighting）。通过人们对各项指标的重视程度的不同，创新性地提出了这种算法，本算法是通过分别计算两种方向行驶的评分，选择更有优势的方向作为行驶方向。

补充：我们遵循了顺便服务规则和提前一秒规则，并且进行灵活目标原则，即每到一个车站进行一次决策判断，这样能有效应对突发事件。

在运行之前，需要输入五个超参数（P向量的四个参数和Beta），作为决策的权重。

1到4分别表示等待时间、请求方向、人流密度、聚合程度四个指标的参数，Beta代表改变方向的代价常量。Beta的使用参考了SCAN策略。

其中，

N=[totalStation/2] [ ]为向下取整。k(dot)代表决策前的行驶方向，若决策前静止,Beta为0

n为沿k方向第几个车站。且最大为N。由于距离的远近对调度结果的影响可以类比推荐系统的Top列表，于是这里参考了推荐系统的NDCG评估指标，当距离逐渐增大时，车站状态对于调度影响较小。此处参考了SSTF策略。

其中，

T为该请求目前的总等待时间。使用二次方是为了使等待时间更长的请求更容易被注意，二分之N代表的是公交车完成随机出现的服务的期望时间。这里对t没有采用SoftMax，是为了确保不会因为t的参数过小导致某请求长期被忽略。此处参考了FCFS策略。

kj是第j个车站的请求方向（1顺时针，-1逆时针，0下车），k是当前评估的方向，数值表示同上文。特别指出，m是当前公交车上的人数，M是总座位数，由于车上的乘客舒适度随占座率的上升而递减（并且梯度从大至小），我们使用了sigmoid激活函数来表示。

p是当前车站历史累计的请求数。由于在实际应用中，顾客请求不会是随机分布的，而是根据地区人口密度、经济水平等影响。因此我们使用p记录请求数，这样可以有效反映出当前车站的客流量，从而使公交车前往此车站的意向更大。

h是当前车站的总请求数。受到该作业三种策略的共性：目标车站的请求全处理的启发，同时因为启动车和刹车过程对油耗较大，本指标会使公交车尽可能倾向于聚合程度高的请求车站。

讨论：在后续工作中，我们会提出更为贴合实际的数据集（比如截取北京某公交路线，从而确保车站距离非相同的；每个车站请求提出的概率会服从人口密度的分布），从而对上述策略验证其有效性。并且，我们会对输入的超参数进行灵敏度检验，验证该模型的稳定性。

### 3.2 自动机模型

#### A.公交车状态图

图示

描述已自动生成

图5

图示

描述已自动生成

图6

公交车具备三个状态，分别是静止、行驶和接放乘客。如图六所示。在静止状态下，若接收到任务请求，则进入行驶状态，并且决策行驶方向是顺时针还是逆时针;在行驶状态下，若当前车站有符合规则的接送请求，则进入接放乘客状态;在接放乘客状态之后进行行驶方向决策，再次进入行驶状态。本系统中不考虑汽车熄火的问题，所以从接送乘客状态不会迁移到静止状态。 对于行驶状态，我们可以进一步细化为顺时针行驶状态和逆时针行驶状态。对于接放乘客状态，我们可以进一步细化为接乘客状态和放乘客状态。图 5给出了接乘客状态下的状态图。在放乘客状态下接收到各命令时的状态转换类似，为保持图的简洁，此处就不再给出。

### 3.3 高层数据结构设计

#### 3.3.1 常量定义

//定义文件输入的状态

#define clock 0

#define target 1

#define wiseclock 2

#define counterwiseclock 3

#define end 4

//定义采取的策略

#define FCFS 1

#define SSTF 2

#define SCAN 3

//定义公交车行动

#define counterclockwise\_action -1

#define clockwise\_action 1

#define stop 0

#define get\_on 2

#define get\_off 3

**3.3.2 全局变量定义**

struct bus Bus;//定义公交车结构体

struct mission Mission;//定义任务链表

int totalTime;//运行总时间

int maxTime;//单次最大等待时间

int finishedMission;//完成任务数量

int totalStation=5;//车站总数，5是默认值

int Distance=2;//车站间距离，2是默认值

int Strategy=1;//运行策略，1,2,3分别代表策略1，2，3

Mission currentMission;//最新的任务链表头

#### 3.3.3 数据结构定义

##### （1）任务的结构

struct mission{

int station;

int direction;

struct mission\* nextMission;// Next mission

int newest;

};

任务请求的站点（station），请求的方向（direction，无方向为0），下一个任务（nextMission）,是否在上一秒内（newest）

##### （2）公交车的结构

struct bus{

int targetStation;

int currentPosition;

int currentDirection;

};

公交车的目标车站号（targetStation，若无目标，则为0），公交车当前位置（currentPosition），公交车当前方向（currentDirection，若静止，则为0）

### 3.4 系统模块划分

#### 3.4.1 软件结构图

本程序分为Input.c、Mission.c、BusControl.c、Strategy.c、几个模块。各模块功能如下：

**模块名称：Main.c**

模块功能简要描述：程序运行的主题文件。

**模块名称：Input.c**

模块功能简要描述：从文件中读取数据并存入变量和数据结构中。

**模块名称：Mission.c**

模块功能简要描述：用于储存处理任务的函数。分别是创建任务和删除已完成的任务

**模块名称：BusControl.c**

模块功能简要描述：根据策略进行的决策，更新公交车的状态以及任务链表的存储，以及更新全局变量的数值

**模块名称：Strategy.c**

模块功能简要描述：根据当前环境与配置参数，对公交车下一步的行为进行决策

**模块名称：Output.c**

模块功能简要描述：输出公交车运行的结果。

**模块名称：Global\_Variable.h**

模块功能简要描述：储存全局变量、数据结构和常用头文件。

图示

描述已自动生成

图七：文件结构关系图

#### 3.4.2 文件及函数组成

**表n 文件及函数功能描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **源文件** | **源文件说明** | **函数名** | **功能** |
| Main.c | 程序运行的主体文件 | int main() | 整合函数功能，连接各个函数，实现程序运行。 |
| Global\_Variable.h | 全局变量、函数和头文件汇总文件 | 其他包含源文件的全局函数 | 实现头文件、变量的汇总，优化在其他文件编写过程中的重复现象 |
| Input.c | 用于读取文件中的参数或指令 | int Read() | 实现指令的读取和存入任务链表并读取三个参数 |
|
| Mission.c | 用于输出下一步运行的目标车站 | void CreateMission(int station, int direction) | 创建新的任务请求，放置于任务链表 |
| void FinishMission(int station, int direction) | 完成任务请求，从任务链表移除该任务 |
| BusControl.c | 本文件中的代 码用于控制公交车的运行 | int Positioning(Bus bus, int Distance) | 获得公交车目前是否在车站或者在第几车站的信息 |
| int CheckStation(Bus bus, Mission currentMission, int station) | 检测当前车站有没有满足处理条件的请求 |
| int JudgeAction(int Strategy, Mission currentMission) | 通过所选择的策略，对公交车行动进行决策 |
| void BusPerform(Bus bus, int Distance, int totalStation, Mission currentMission,int Strategy) | 应用决策，控制公交车进行所对应的行为 |
| Strategy.c | 本文件中的代码用于判断公交车采取哪种策略 | int FCFS() | 判断采取先来先服务策略时应该执行哪种决策 |
| int SSTF() | 判断采取最短距离优先服务策略时应该执行哪种决策 |
| int SCAN() | 判断采取顺便服务策略时应该执行哪种决策 |
| Output.c | 输出当前状态到要求的文件中 | void WriteState(FILE\* fpO) | 输出当前状态 |
| void initialize(char a[]) | 初始化输出数组 |
| Draw.c | 绘制状态图 | void DrawState() | 绘制当前状态图 |

#### 3.4.3 函数说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | int main() | 运行程序 | void | 0 |
| 2 | int read() | 实现指令的读取和存入任务链表并读取三个参数 | FILE\* | clock返回0  end返回-1  三个指令返回1 |
| 3 | void CreateMission(int station, int direction ) | 创建新的任务请求，放置于任务链表 | 新的请求的对应车站和方向，若是下车请求方向为0 | void |
| 4 | void FinishMission(int station, int direction) | 完成任务请求，从任务链表移除该任务 | 完成的请求的对应车站和方向，若是下车请求方向为0 | void |
| 5 | void OldenMission() | 将任务链表中任务的newest值改为0 | void | void |
| 6 | int Positioning() | 获得公交车目前是否在车站或者在第几车站的信息 | void | 公交车如果不在车站，返回0，否则返回当前车站号 |
| 7 | int CheckStation(int station) | 检测当前车站有没有满足处理条件的请求 | 公交车所在车站station | 如有合规的上车请求，返回1，下车请求返回-1 |
| 8 | int JudgeDirection(int currentStation, int targetStation) | 通过传入当前车站和请求车站，返回之后恰当的行驶方向 | 公交车当前车站currentStation  目标车站targetStation | 顺时针为1，逆时针为-1，理论上不存在第三种情形 |
| 9 | int JudgeAction(currentMission) | 通过所选择的策略，对公交车行动进行决策 | void | 返回公交车的行动决策，-1逆时针行驶，1顺时针行驶，0停止，2处理上车请求，3处理下车请求 |
| 10 | void BusPerform() | 应用决策，控制公交车进行所对应的行为 | void | void |
| 11 | int FCFS() | 判断采取先来先服务策略时，应该执行哪种决策 | void | 返回采取FCFS  时公交车的行动决策，具体值如10所示 |
| 12 | int SSTF() | 判断采取最短时间策略时，应该执行哪种决策 | void | 返回采取SSTF  时公交车的行动决策，具体值如10所示 |
| 13 | int SCAN() | 判断采取顺便服务策略时，应该执行哪种决策 | void | 返回采取SCAN时公交车的行动决策，具体值如10所示 |
| 14 | int findNearest() | 寻找当前位置最近的请求 | void | void |
| 15 | void WriteState() | 输出当前状态到文件中 | FILE\* | void |
| 16 | void DrawState() | 绘制当前状态图 | void | void |

### 3.5 高层算法设计

计算机控制公交车运行的线程:（每读取一个clock，进行一次）

void BusPerform(Mission currentMission)

{

if (任务链表为空){

不进行决策

}

if(公交车没在车站){

令公交车沿原来方向行驶

}

行动 = 根据当前环境与设定数据进行决策

switch (行动)

{

case 逆时针行驶://目标车站在逆时针

将公交车方向改为逆时针

公交车位置减少一

break;

case 静止:

if(判断当前车站有无接放请求){

处理所有请求}

break;

case 顺时针行驶://目标车站在顺时针

将公交车方向改为顺时针

公交车位置增加一

break;

case 接一个人:

从任务链表删除该任务

if(该任务的车站和当前车站相同){

将公交车的方向和目标车站归为初始值}

if(所选策略满足同时多次接人放人的条件){

处理接放人请求}

break;

case 放下一个人:

从任务链表删除该任务

if(该任务的车站和当前车站相同){

将公交车的方向和目标车站归为初始值}

if(所选策略满足同时多次接人放人的条件){

处理接放人请求}

处理任务总数加一

break;

default:

break;

}

}

计算机读取文件的线程:（每一次主循环，进行一次）

读入一个字符

while（判断不为“#”）{

读一整行

读入一个字符

}

文件指针回退一个字符

读字符串到“=”

判断为“DISTANCE”“STRATEGY”“TOTAL\_STATION”分别填入相应参数