麦当劳点餐系统概要设计书

2023211311-2023211198-杜昊阳

# 一．麦当劳点餐系统问题描述

## 1.题面：

2023年5月，麦当劳在北邮开业。大量的学生去那里订餐。正因为如此，麦当劳的在线点餐系统经常关闭以避免拥挤，尤其是在午餐和晚餐时间。该系统的关闭时间不确定。北邮的学生认为这非常麻烦。

然而，北邮学生无所畏惧。北京邮电大学最优秀的学生之一(也是北邮ICPC团队的一员)Zhai Xie (ThomasX)在飞书上开发了一个实时监控机器人，它告诉我们麦当劳在线点餐系统的实时状态。有了这个机器人，北邮学生可以更方便地点餐。

在这个问题中，需要你像他一样完成这个任务，开发一个系统来模拟麦当劳的在线点餐系统。

北邮的麦当劳和它的点餐系统在07:00:00开始工作，在22:00:01关闭。麦当劳一共有N种食物和M种套餐类型，每种套餐中包含多种食物，具体配置信息将在菜单文件 (dict.dic) 中提供。对于制作和存储每种食物，规定第i种食物在t\_i秒内完成，其最大存储容量为cap\_i ，表示该种食物最多可以存储cap\_i个。麦当劳系统每天开放前，所有食物存储容量都为0，在任何时间点如果某种食物的存储量小于cap\_i，则会立即制作该食物，直到达到cap\_i。其中，不同种类食物可以同时制作，同种类食物只能依次制作。

从07:00:00到22:00:00(含)，学生可以在系统中点餐(如果系统未关闭)。每一天按照顺序有n个订单，第i个订单发生在时间a\_i:b\_i:c\_i，其要求一份type\_i类型(type\_i∈M\_combo∪N\_food，其中M\_combo和N\_food分别表示全体的套餐和食物的集合)的套餐或食物。如果点餐时系统关闭，会导致点餐失败。22:00以后如果还有之前的订单未完成，则麦当劳会继续加班，且保证23:59:59(含)前一定能完成所有订单。

对于订单处理存在如下规则：

在每一秒的开始，如果有新的食物完成，则首先存储食物，然后接受订单(如果存在)。

订单按照“先来先到，异步处理”原则进行处理。

先来先到：指的是对于有存量的食物，总会被分配给时间最早的订单(套餐或单点)。

异步处理：指的是当一个订单(套餐或单点)因为请求的食物没有被全部满足时，不必等待该订单完成，可以直接处理下一个订单。

食物一旦被分配给订单，就不能撤销。食物被分配给订单后，即便该订单尚未完成，该食物也不再占用对应类型的容量。

当订单(套餐或单点)中要求的所有食物，均已被分配给该订单，则该订单会立刻完成。

如果在某个时刻t\_0，有人下了一个订单，并且该订单无法立刻完成，导致未完成订单的数量大于W\_1，则系统立即自动关闭(不再接受订单)，但该订单仍然算作成功下单。

如果在某个时刻t\_1，未完成订单的数量小于W\_2，则系统将在1秒后重新打开。即系统可以接受t\_1+1时刻的订单，而不能接受t\_1时刻的订单。

你的系统需要输出：每一个订单是否下单成功，以及完成的时间。

## 2.菜单文件

本题为大家提供麦当劳的菜单文件(dict.dic)，按如下格式给出：

第一行给出和，其中表示食物的种类数()，表示套餐的种类数()。

第二行包含个字符串，每个字符串 表示第种食物的名称(不超过50个字符)。

第三行包含个整数()，其中表示第种食物的制作时长。

第四行包含个整数()，其中表示第种食物的最大存储容量。

第五行包含两个整数()。

接下来行，其中的第行包含多个字符串，第一个字符串表示第个套餐的名称(不超过50个字符)，后续的第个字符串表示第个套餐中包含的第种食物的名称。一个套餐包含的食物种类不超过5，每种食物只有1个。

注：系统每次运行时所读取的菜单文件内容可能不一样。

## 3.输入

第一行包含一个整数()表示订单个数。

对于接下来的行，用格式类似于的方式，给出第个订单的时间。然后输入一个字符串，表示套餐或食物的名称(参见dict.dic)。所有订单时间一定在内，同一个时间点不可能出现多个订单，第个订单一定早于第个()，且保证23:59:59(含)前一定能完成所有订单。

具体参见input.txt

## 4.输出

输出包括n行，按照订单顺序输出订单完成时间。对于第i行，如果第i个订单不成功，则输出Fail；否则，输出这个订单完成的时间，时间格式与输入格式(11:11:11)一致。

具体参见output.txt

# 二．需求分析

## 1.总述

开发的第一步是进行需求分析。需求分析需要从系统的数据、功能 和行为三方面进行分析。

数据方面，应当建立餐品数据，套餐数据，订单数据，任务处理数据。

功能方面，包含菜单读取，当日任务读取，订单完成情况汇报。

行为方面，包括菜单数据处理，当日任务数据处理，执行当日任务，订单完成情况输出。

## 2.数据配置

### （1）餐品数据：

struct foodkind {//食物种类

char name[30];//名称

int number;//当前数量

int max;//最大数量

int currenttime;//当前已制作时间

int needtime;//制作所需时间

};

### （2）套餐数据：

struct combinationlink {//套餐对应链表

struct foodkind \*link;//对应餐品种类

struct combinationlink \*next;//下一餐品

};

struct combination {//套餐

char name[30];//名称

int kindnumber;//餐品数量

struct combinationlink \*kind;//餐品名称

};

### （3）订单数据：

struct orderlink {//订单对应链表

struct foodkind \*link;//对应餐品种类

int provided;//是否已提供餐品

struct orderlink \*next;//下一餐品

};

struct order {//订单

int begintime;//开始时间

int finishtime;//结束时间

int remainfoodnumber;//未完成餐品数量

int state;//订单状态：0未开始，1进行中，2成功，3失败

struct orderlink \*kind;//餐品详情

};

### （4）任务处理数据：

//建立任务处理数据

int time=0;//时间戳

int hour1=0;//hh:mm:ss时间-第1位

int hour2=0;//hh:mm:ss时间-第2位

int minute1=0;//hh:mm:ss时间-第3位

int minute2=0;//hh:mm:ss时间-第4位

int second1=0;//hh:mm:ss时间-第5位

int second2=0;//hh:mm:ss时间-第6位

int foodnumber=0;//餐品种类数量

int combnumber=0;//套餐种类数量

int ordenumber=0;//订单数量

int allowmax=0;//系统关闭订单量w1

int allowmin=0;//系统恢复订单量w2

int remainorder=0;//进行中订单量

int systemstate=1;//系统开启情况，0关闭，1开启。

//建立临时变量

int i=0;

int j=0;

int k=0;

char ch[30]={0};

char c=0;

struct combinationlink \*combinationlinknewp;

struct combinationlink \*combinationlinkcurp;

struct orderlink \*orderlinknewp;

struct orderlink \*orderlinkcurp;

## 3.功能配置：

### （1）菜单读取功能

菜单文件读取：

第一行读取foodnumber和combnumber，其中foodnumber表示餐品的种类数，combnumber表示套餐的种类数。

第二行包含foodnumber个字符串，每个字符串表示第i种食物的名称。

第三行包含foodnumber个整数，其中第i个数字表示第i种食物的制作时长。

第四行包含foodnumber个整数，其中第i个数字表示第i种食物的最大存储容量。

接下来combnumber行，其中的第i行包含多个字符串，第一个字符串表示第i个套餐的名称，后续的第j个字符串表示第i个套餐中包含的第j种食物的名称。

输入读取：

第三行包含foodnumber个整数，其中t\_i表示第i种食物的制作时长。

第四行包含foodnumber个整数，其中cap\_i表示第i种食物的最大存储容量。

### （2）当日任务读取

输入读取：

第一行包含一个整数n(1≤n≤54001)表示订单个数。

对于接下来的n行，用格式类似于11:11:11的方式，给出第i个订单的时间。然后输入一个字符串type\_i，表示套餐或食物的名称(参见dict.dic)。所有订单时间一定在[07:00:00,22:00:00]内，同一个时间点不可能出现多个订单，第i-1个订单一定早于第i个(2≤i≤n)，且保证23:59:59(含)前一定能完成所有订单。

### （3）订单完成情况汇报

输出包括n行，按照订单顺序输出订单完成时间。对于第i行，如果第i个订单不成功，则输出Fail；否则，输出这个订单完成的时间，时间格式与输入格式(11:11:11)一致。

## 4.行为配置

### （1）数据处理

创建餐品，写入餐品名称，最大数量，制作所需时间，同时将当前数量和当前制作时间初始化为0。

创建套餐，写入套餐名称，餐品数量，将对应餐品所需数量设置为1，同时将各餐品情况的指针指向餐品。

创建订单，写入对应套餐数据，记录订单剩余未完成餐品数，订单开始时间，订单状态，订单内具体餐品完成情况。

### （2）执行当日任务

即开始营业，从07：00：00到23：59：59，每秒钟进行食物制作和订单处理。

记录每个订单完成状态（未开始，进行中，成功，失败），若成功，记录完成时间。

### （3）订单完成情况输出

每日任务完成后，按订单顺序输出完成情况/完成时间。

# 三．概要设计

## 1.数据联系图

## 2.主程序设计

主程序{

读取菜单

读取输入

当日营业

订单情况汇报

}

## 3.子程序设计

### （1）读取菜单

FILE \*fp;

fp=fopen("dict.dic","r");

fscanf(fp,"%d%d",&foodnumber,&combnumber);//读取餐品数和套餐数

combnumber+=foodnumber;//单点商品也算作套餐

struct foodkind food[foodnumber];

struct combination comb[combnumber];

for(i=0;i<foodnumber;i++) {

fscanf(fp,"%s",ch);//读取餐品名称

strcpy(food[i].name,ch);

food[i].number=0;

food[i].max=0;

food[i].currenttime=-1;

food[i].needtime=0;

strcpy(comb[i].name,ch);//将餐品也定义为套餐

comb[i].kindnumber=1;

combinationlinknewp=(struct combinationlink\*)malloc(sizeof(struct combinationlink));

comb[i].kind=combinationlinknewp;

comb[i].kind->link=&food[i];

comb[i].kind->next=NULL;

}

for(i=0;i<foodnumber;i++) {

fscanf(fp,"%d",&food[i].needtime);

}//读取各餐品制作所需时间

for(i=0;i<foodnumber;i++) {

fscanf(fp,"%d",&food[i].max);

}//读取各餐品最大容量

fscanf(fp,"%d%d",&allowmax,&allowmin);//读取系统关闭订单量w1，系统恢复订单量w2

for(;i<combnumber;i++) {//读取多餐品套餐

fscanf(fp,"%s",ch);

strcpy(comb[i].name,ch);

comb[i].kindnumber=0;

comb[i].kind=NULL;

for(c=fgetc(fp);c!='\n';c=fgetc(fp)) {

fscanf(fp,"%s",ch);

combinationlinknewp=(struct combinationlink\*)malloc(sizeof(struct combinationlink));

combinationlinknewp->link=&food[correctfood(ch,food,foodnumber)];

comb[i].kindnumber++;

if(comb[i].kind==NULL) {

comb[i].kind=combinationlinknewp;

combinationlinkcurp=combinationlinknewp;

}

else {

combinationlinkcurp->next=combinationlinknewp;

combinationlinkcurp=combinationlinknewp;

}

}

combinationlinkcurp->next=NULL;

}

fclose(fp);

### （2）读取输入

scanf("%d",&ordenumber);//读取订单数

struct order orde[ordenumber];

for(i=0;i<ordenumber;i++) {

orde[i].begintime=timeread();//读取各订单开始时间并转化为时间戳

orde[i].finishtime=0;

orde[i].kind=NULL;

orde[i].state=0;

scanf("%s",ch);

j=correctcomb(ch,comb,combnumber);//找出订单对应套餐编号

orde[i].remainfoodnumber=comb[j].kindnumber;//将套餐信息复制给订单

combinationlinknewp=comb[j].kind;

for(k=0;k<comb[j].kindnumber;k++) {

orderlinknewp=(struct orderlink\*)malloc(sizeof(struct orderlink));

orderlinknewp->link=combinationlinknewp->link;

orderlinknewp->provided=0;

combinationlinknewp=combinationlinknewp->next;

if(orde[i].kind==NULL) {

orde[i].kind=orderlinknewp;

orderlinkcurp=orderlinknewp;

}

else {

orderlinkcurp->next=orderlinknewp;

orderlinkcurp=orderlinknewp;

}

}

orderlinkcurp->next=NULL;

}

### （3）当日营业

①整体：

for(time=25200;time<86400;time++) {//按秒循环

制作食物；

点单处理；

}

②制作食物：

for(i=0;i<foodnumber;i++) {

if(food[i].number<food[i].max) {

food[i].currenttime++;

if(food[i].currenttime==food[i].needtime) {

food[i].number++;

food[i].currenttime=0;

}

}

}

③点单处理：

//系统状态判断

if(remainorder>allowmax)

systemstate=0;

if(remainorder<allowmin)

systemstate=1;

//订单处理

for(i=0;i<ordenumber;i++) {

if(orde[i].begintime>time)//未开始

break;

else if(orde[i].begintime==time) {//开始时

if(systemstate==1) {//系统开放点单

orde[i].state=1;//订单进行中

remainorder++;

//配餐

for(orderlinkcurp=orde[i].kind;orderlinkcurp!=NULL;orderlinkcurp=orderlinkcurp->next) {

if(orderlinkcurp->provided==0) {//当前餐品未配餐

if(orderlinkcurp->link->number!=0) {//当前餐品有存量

orderlinkcurp->link->number--;

orderlinkcurp->provided=1;

orde[i].remainfoodnumber--;

}

}

}

//判断订单是否完成

if(orde[i].remainfoodnumber==0) {

orde[i].state=2;

orde[i].finishtime=time;

remainorder--;

}

}

else {//系统暂停点餐

orde[i].state=3;//点单失败

}

}

else {

if(orde[i].state==2||orde[i].state==3)//订单已结束（完成/失败）

continue;

else {//订单进行中

//配餐

for(orderlinkcurp=orde[i].kind;orderlinkcurp!=NULL;orderlinkcurp=orderlinkcurp->next) {

if(orderlinkcurp->provided==0) {//当前餐品未配餐

if(orderlinkcurp->link->number!=0) {//当前餐品有存量

orderlinkcurp->link->number--;

orderlinkcurp->provided=1;

orde[i].remainfoodnumber--;

}

}

}

//判断订单是否完成

if(orde[i].remainfoodnumber==0) {

orde[i].state=2;

orde[i].finishtime=time;

remainorder--;

}

}

}

}

### （4）订单完成情况汇报

for(i=0;i<ordenumber;i++) {

if(orde[i].state==3)//orde[i]失败

printf("Fail\n");

else {//orde[i]成功

//将时间戳转化为hh:mm:ss时间

second1=orde[i].finishtime%60;

second2=second1%10;

second1=second1/10;

minute1=(orde[i].finishtime%3600)/60;

minute2=minute1%10;

minute1=minute1/10;

hour1=orde[i].finishtime/3600;

hour2=hour1%10;

hour1=hour1/10;

printf("%d%d:%d%d:%d%d\n",hour1,hour2,minute1,minute2,second1,second2);

}

}

return 0;

}

## 4.辅助程序设计

int correctfood(char ch[],struct foodkind food[],int foodnumber) {//返回餐品名称ch对应序号

int n;

for(n=0;n<foodnumber;n++) {

if(strcmp(ch,food[n].name)==0)

return n;

}

}

int correctcomb(char ch[],struct combination comb[],int combnumber) {//返回套餐名称ch对应序号

int n;

for(n=0;n<combnumber;n++) {

if(strcmp(ch,comb[n].name)==0)

return n;

}

}

int timeread() {//将hh:mm:ss时间转化为时间戳

int hour=0,minute=0,second=0,time=0;

scanf("%d:%d:%d",&hour,&minute,&second);

time=second+60\*minute+3600\*hour;

return time;

}