**《程序设计课程设计》实验报告**

**实验名称 《JobShop管理游戏系统程序设计》概要设计-命令行版本 <v1.3>**

**班 级 11**

**组 号 36**

**姓 名 宋伊雯、张志博、赵佳祺**

# 1 高层数据结构设计

(包括：重要的数据常量定义、数据变量定义，即各模块要共享的数据类型和参数设计，相当于头文件内容，加文字描述)

**1.1全局常量/变量定义**

**1.1.1 全局变量：**

**int** jobNum, machineNum, makeSpan; //产品数目, 机器数目, 最大完工时间

OVERHAULPTR overhaul; //检修机器信息

**1.1.2 结构：**

**typedef struct** job { //产品，用于输入与处理  
 **int** machine; //该工序指定机器号  
 **int** time; //该工序所花时间  
 **struct** job \*nextMachine; //下一机器  
} \*JOBPTR;  
**typedef struct** machine { //机器，用于输出  
 **int** job; //当前产品  
 **int** time; //当前产品工序所花时间  
 **int** timeline; //当前机器时间线  
 **struct** machine \*nextJob; //下一产品  
} \*MACHINEPTR;

**typedef struct** overhaul { //检修  
 **int** machine; //检修机器  
 **int** time; //检修时间  
 **int** timeline; //检修时长  
 **struct** overhaul \*nextOverhaul; //下一检修  
} \*OVERHAULPTR;

**1.2 \*\*模块常量与变量定义**

**1.2.1 main模块变量：**

**int** inputMode; //输入模式，1为键盘输入，2为文件输入，3为图形界面输入

JOBPTR \*job; //产品链表指针，用于输入与处理  
MACHINEPTR \*machine; //机器链表指针，用于输出

**1.2.2 其他模块变量：（暂无重要公共变量）**

# 2 系统模块划分

## 2.1 系统模块结构图

模块划分思路说明。

main.c

//graph.c

io.c

schedule.c

**1. 模块名称 main.c**

模块功能简要描述：获取输入模式，控制主体流程。

**2. 模块名称 io.c**

模块功能简要描述：完成对文件或键盘输入的订单数据获取，保存到公共变量和全局变量中，输出结果。

**3. 模块名称 schedule.c**

模块功能简要描述：安排在每台机器上工件的加工顺序，使得总的完工时间(Makespan)最小，将加工顺序保存到公共变量中，将完工时间保存到全局变量中。

**\*4. 模块名称 graph.c**

模块功能简要描述：通过图形界面完成订单数据和检修信息的获取，保存到公共变量和全局变量中，通过图形界面输出结果。

## 2.2各模块函数说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | JOBPTR \*getJobByKeyboard(); | 输入模式是键盘输入时被main调用。  完成对文件输入的订单数据获取，保存到公共变量中。 | void | JOBPTR \* |
| 2 | **void** getOverhaulByKeyboard(); | 输入模式是键盘输入时被main调用。  完成对文件输入的检修信息获取，保存到全局变量中。 | void | void |
| 3 | JOBPTR \*getByFile(); | 输入模式是文件输入时被main调用。  完成对文件输入的订单数据和检修信息获取，保存到公共变量和全局变量中。 | void | JOBPTR \* |
| \*4 | JOBPTR \*getJobByGraph(); | 输入模式是图形输入时被main调用。  完成对图形界面输入的订单数据和检修信息获取，保存到公共变量和全局变量中 | void | JOBPTR \* |
| \*5 | **void** getOverhaulByGraph(); | 输入模式是图形输入时被main调用。  完成对图形界面输入的检修信息获取，保存到全局变量中 | void | void |
| 6 | MACHINEPTR \*schedule(JOBPTR \*job); | 安排在每台机器上工件的加工顺序，使得总的完工时间最小，将加工顺序保存到公共变量中，将完工时间保存到全局变量中。 | JOBPTR \* | MACHINEPTR \* |
| 7 | MACHINEPTR \*reSchedule(MACHINEPTR \*machine, JOBPTR \*job); | 重新安排检修通知后在每台机器上工件的加工顺序，使得总的完工时间最小，将加工顺序保存到公共变量中，将完工时间保存到全局变量中。 | MACHINEPTR \*machine, JOBPTR \*job | MACHINEPTR \* |
| 8 | **void** outputOnScreen(MACHINEPTR \*machine, JOBPTR \*job); | 输入模式是键盘输入或文件输入时被main调用。  在命令行界面输出结果。 | MACHINEPTR \*machine, JOBPTR \*job | void |
| 9 | **void** outputByFile(MACHINEPTR \*machine, JOBPTR \*job); | 被main调用。  在文件中输出结果。 | MACHINEPTR \*machine, JOBPTR \*job | void |
| \*10 | **void** outputByGraph(MACHINEPTR \*machine, JOBPTR \*job); | 输入模式是图形输入时被main调用。  在图形窗口中用绘制的甘特图来表示操作在机器上的安排。 | MACHINEPTR \*machine, JOBPTR \*job | void |
| 11 | **void** freeAll(JOBPTR \*job, MACHINEPTR \*machine, OVERHAULPTR overhaul); | Free申请的内存。 | JOBPTR \*job, MACHINEPTR \*machine, OVERHAULPTR overhaul | void |

## 2.3 函数调用图示及说明

freeAll

main

outputOnScreen

getJobByKeyboard

outputByFile

getOverhaulByKeyboard

//outputByGraph

getByFile

schedule

reSchedule

//getJobByGraph

//getOverhaulByGraph

解释说明：

如图中所示，main先通过对getJobByKeyboard、getByFile、getJobByGraph的调用，完成对文件、键盘、图形界面输入的订单数据获取；

Main再通过对getOverhaulByKeyboard、getByFile、getOverhaulByGraph的调用，完成对文件、键盘、图形界面输入的检修信息获取；

main再通过对schedule、reSchedule的调用，完成对每台机器上工件的加工顺序和总的完工时间(Makespan)最小输出的安排、获取；

main再通过对outputOnScreen或outputByGraph的调用，完成对结果的命令行或图形化界面输出；outputOnScreen或outputByGraph通过对ouputByFile的调用，完成对结果的文件输出。

main最后通过对freeAll的调用，解放之前申请的内存。

# 3 高层算法设计

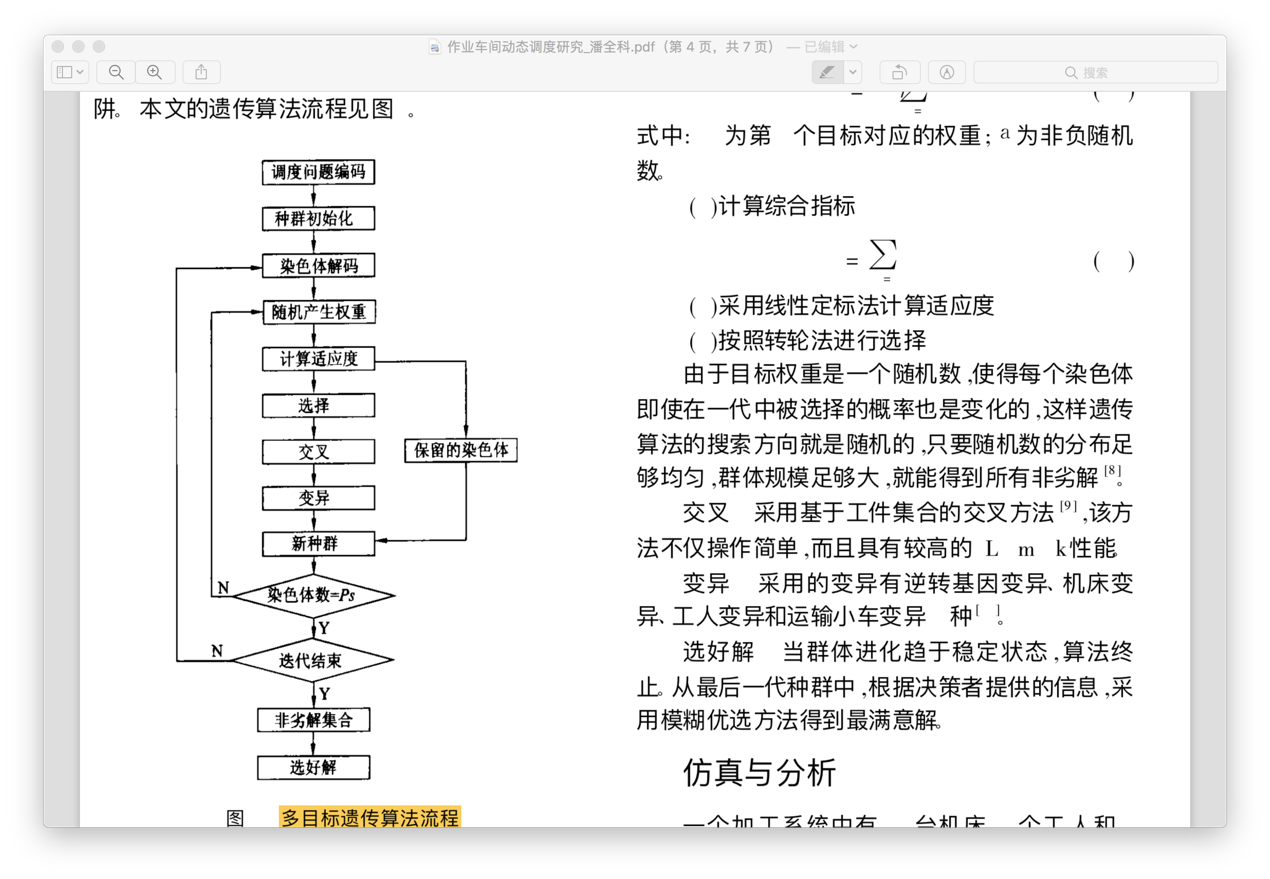
**（用伪代码、NS图或者自然语言描述清楚核心算法的程序设计思路）**

//说明重要的控制策略算法思路；

一、schedule

1.算法简介：

遗传算法（GA）是一种高度并行、随机和自适应化的算法，他将问题的解决用染色体描述，再通过选择、交换和变异等操作对种群中的个体作用，是种群进化来进行全局优化搜索，特别适用于搜索较大的解空间。其中，基础调度方案基于遗传算法框架如下：



2.主要概念介绍：

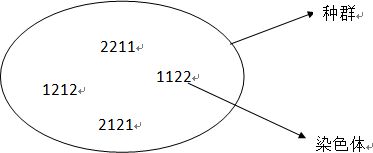
（1）染色体编码

染色体的编码是将车间调度问题转化为遗传算法可处理形态的关键步骤，所谓染色体即是将各加工订单的编号按先后顺序排列后形成的整数串。

（2）种群的创建

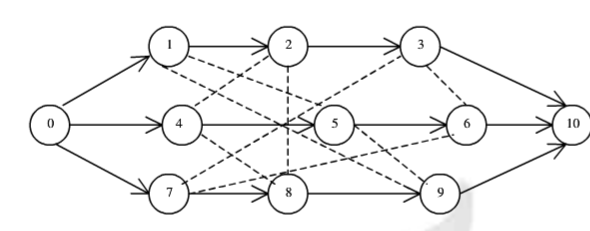
在编码之后随机生成的所有工件号的排列的集合称之为种群。

例如，假设当前有两个工件需要加工，分别用1和2表示，每个工件有两个加工步骤（工件编号出现次数代表该工件的加工步骤数），则下图就表示了该情形下所有染色体排列所形成的种群。



（3）析取图

析取图是描述jobshop的常用工具。对于n个工件、m台机器的问题，对应的析取图G=(V,A,E)。如图，其中V代表所有操作构成的顶点集，包括0和N+1两个虚拟操作（分别表示开始和结束）；A为n\*(m+1)条子边（实线）构成的边集，子边表示某工件按约束条件再所有机器上从开始到结束的加工路径；E为n\*m条子弧（虚线）构成的弧集，子弧表示在同一机器上加工的各操作的连接。



若以最大完成时间为指标，则对jobshop的求解就归结为找到各子弧围成的环（环代表机器）上作为优先决策的各操作的一组顺序，其最大完成时间即为关键路径长度。

（4）染色体的交叉

交叉是遗传算法中的一个重要操作，他的目的是从两条染色体中各自取出一部分来组合成一条新的染色体，交叉操作是遗传算法中决定全局搜索能力的主要因素。交叉操作的思想是保留并充分利用已知的优良模式，使得交叉操作向进化的方向前进。

（5）染色体的变异

变异操作通常发生在交叉操作之后，他的操作对象是交叉得到的新染色体。

例如：变异前的染色体：ABCABCABC

随机选取两个位置：ABCABCABC

变异后的染色体：AACABCBBC

（6）染色体解码

解码操作是利用染色体中的信息形成与其唯一对应的有向无环图，而此情况下的最优解则是从起点到终点中所有路径中总完成时间最长的路径对应的完成时间。

3.主要函数说明：

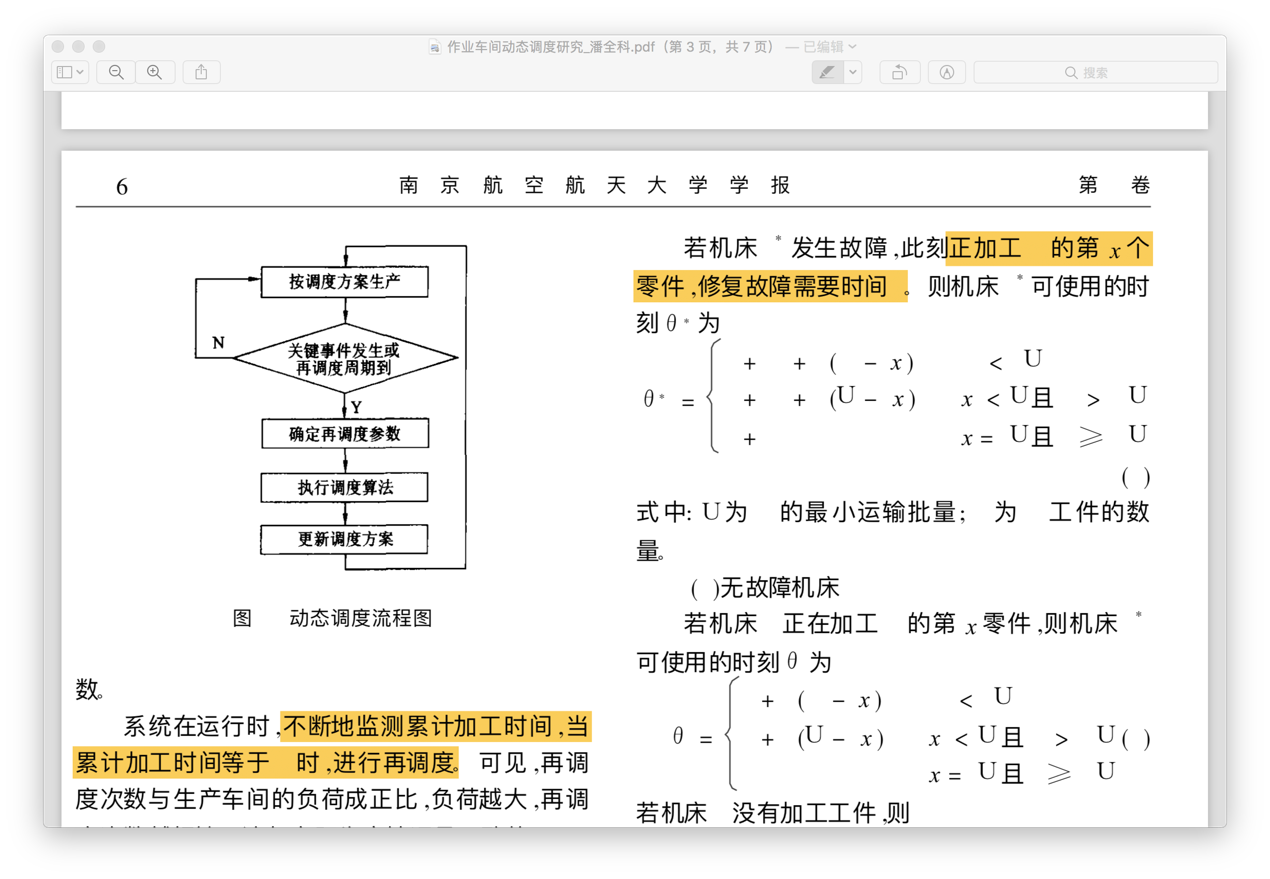
|  |  |  |
| --- | --- | --- |
| 函数名 | 函数功能 | 变量传入与返回值 |
| int\*\* initPopulation(\*JOBPTR) | 生成初始种群的编码 | \*JOBPTR（传入存有所有工件信息的数组）；  返回随机生成的染色体组 |
| void crossover(\*\*int,int,int) | 选取两条染色体进行交叉 | 传入染色体数组，以及需要进行交叉操作的两条染色体编号 |
| void mutation(\*\*int,int) | 挑选一条染色体进行变异 | 传入染色体数组，以及需要进行变异操作的一条染色体编号 |
| MACHINEPTR \*computeDAG(\*\*int,int) | 进行染色体的解码，构造析取图 | 传入染色体数组；返回记录节点信息和节点的指向及相对位置的信息 |
| int computeTime(\*\*int,int) | 根据析取图计算该染色体编码下的对应时间 | 传入染色体数组和特定的染色体编码，调用computeDAG函数计算时间；返回所需时间 |

二、reSchedule

主思路：基于遗传算法的动态调度

1.算法简介：

基于事件调度对加工过程进行再调度，以贴合工程中突发事件的实际性。

对无检修情况下的遗传算法上加以改进，将检修时间段等效为零产出的工件，新建立一个结构数组存储。但由于检修的起始时间与终止时间受限，故重点在于染色体编码、交叉、变异三个步骤的实现。动态调度方案基于遗传算法框架如下：

2. 基于遗传算法的再调度

（1）染色体编码

编码时加入代表检修的零产出工件们

（2）种群的再创建

筛选出零产出工件位置合适的染色体们

（3）染色体的交叉

根据检修约束，只对正常工件染色体部分进行操作

（4）染色体的变异

根据检修约束，只对正常工件染色体部分进行操作

（5）染色体解码

**教师评语：**