# 算法整体思路

## 1.1算法描述

**输入**：Workflow-net表示的业务过程模型(Business Process Mode)，XML格式

**输出**：过程日志

**要求**：生成输入的业务过程模型包含的所有可能的任务执行序列，其中循环任务执行不超过1次。提交算法源代码，实现接口getLogOfModel(String modelFile, String logFile)

## 1.2算法思路

语言：Java 运行环境：Java JDK 1.8

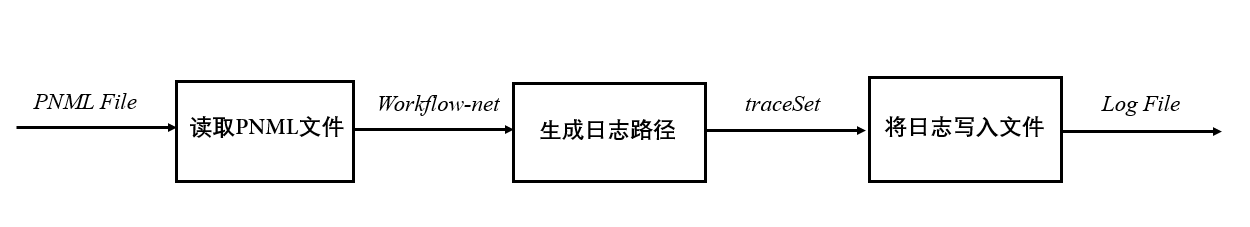


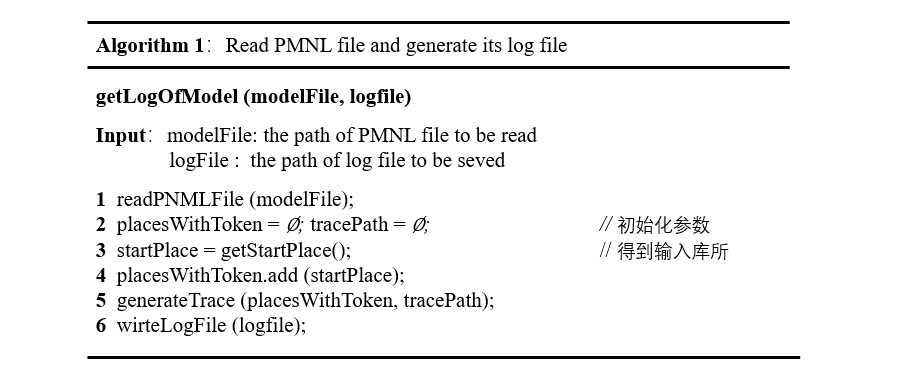
图1 算法主要步骤

1. 解析PNML文件，提取Petri网中的所有变迁、库所和弧的信息，并保存到相关类中；

2. 遍历Petri图，将每次一次运行中所有执行的变迁按照执行顺序记录下来。生成一条日志，记录所有的日志路径；

3. 将所有的日志结果写入到日志文件中。

## 1.3伪代码



算法1

# 算法各个模块思路

## 解析PNML文件

PNML是一种基于 XML的 Petri 网文件交换格式，记录了一个Petri网中的所有信息。因此模块使用DOM API读取XML结点信息，利用DocumentBuilder-Factory可以解析出库所、变迁和弧信息。

### 数据结构设计：

**Petri网类：**

Petri网的信息存储到类（wfNet class）中。类的成员变量包括：

1. 库所、变迁列表，存储网中的库所和变迁信息；
2. 变迁字典，key：变迁ID，value：变迁名称，用来根据变迁ID查询变迁的名称，输出日志路径；
3. 关联矩阵，输入输出矩阵，变迁保存每条弧的信息；
4. 计数器，用于累计变迁或库所的访问次数，判断环是否只执行了一次。

**关联矩阵：**

一个Petri网∑ (S,T,F) 可以表示为个n行m列的关联矩阵：

其中变迁个数n，库所个数为m。∈T，∈S

**变迁输入矩阵**： = n×m 当存在(ti,sj)∈F时, = 1,否则为0

**变迁输出矩阵**： = n×m 当存在(sj,ti)∈F时, = 1,否则为0

### 算法设计：

**算法：读取PNML文件并生成保存Petri网信息的类**

**输入**：Petri网的PNML文件

**输出**：WFNet类

**步骤1**：初试化WFNet类；

**步骤2：**读取PNML文件，解析place标签的id属性值，将库所id加入库所列表；

**步骤3：**解析PNML文件的transition标签的id属性值和二级结点name的值，将变迁的id加入变迁列表，将变迁id对应的name值形成映射加入变迁字典；

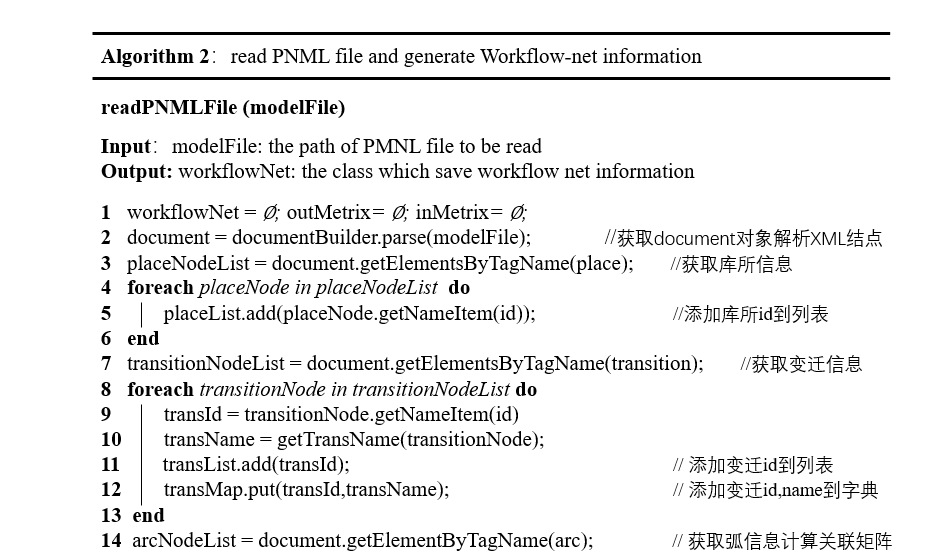
**步骤4：**得到变迁个数n，库所个数m；初始化两个n行m列全0的矩阵；

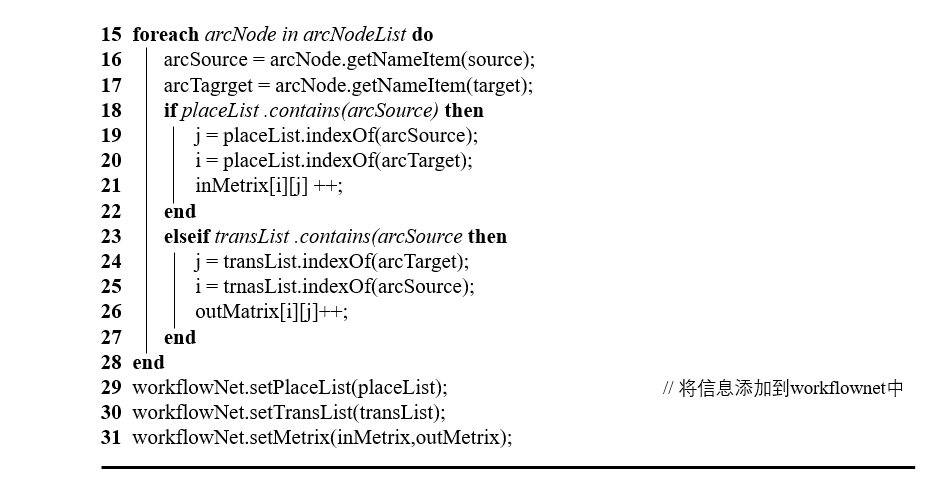
**步骤5：**解析PNML文件arc标签的sorce和target属性值；

**步骤6：**计算输出矩阵：取出变迁匹配sorce项，并将其对应target项中存储的库所读出，将输入矩阵的i行j列置为1；

**步骤7：**计算输入矩阵：取出库所匹配target项，并将其对应的target项中存储的变迁读出，将输出矩阵的i行j列置为1。

### 伪代码：





算法2

## 生成日志路径

Workflow-net的过程模型每一次运行的初始状态是输入库所中填入一个令牌，结束状态是在输出库所中填入一个令牌。所有输入库所中都填有令牌时，变迁达到可执行状态；当变迁执行时消耗每一个输入库所中的1个令牌，并在所有输出库所中填入1个新的令牌。此算法将过程模型从初始状态到结束状态中所有可执行的变迁都执行一次，按照变迁执行的顺序形成一个变迁执行序列，即一条过程日志。最后将所有的过程日志中加入到集合中。

### 算法设计：

整体思路是类似图的深度优先遍历，通过递归模拟变迁在不同标识下的发生情况。

**算法：遍历Workflow-net 过程模型生成过程日志**

**输入：**填入令牌的库所列表 placesWithTokent、变迁执行序列 transPath

**输出：**过程日志

**步骤1**：找到WF-net中的输入变迁，填入1个令牌，初始化变迁执行序列为空；

**步骤2**：将库所列表，变迁执行序列当做参数调用此算法开始执行；

**步骤3:** 循环填入令牌的库所列表，判断是否含有输出库所。若含有输出库所，则代表当前执行状态处于结束状态，把执行路径加入集合中，返回集合，结束算法。否则执行步骤4；

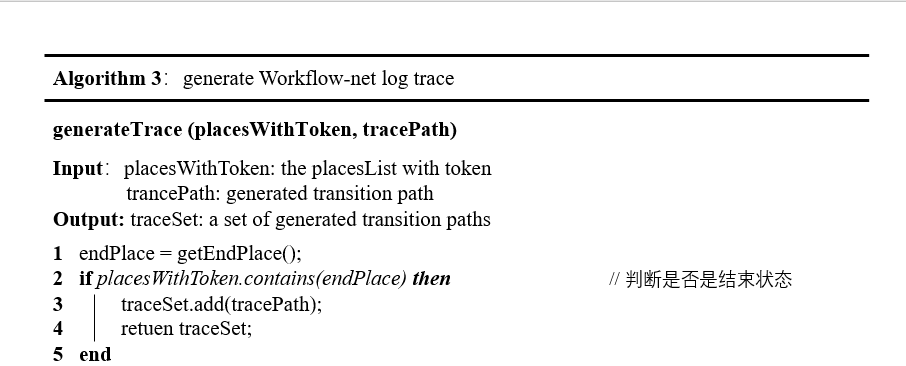
**步骤4：**遍历填入令牌的库所列表，得到库所s，取库所s可达的变迁列表，并判断当前库所s是否存在环；

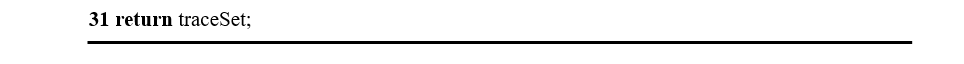
**步骤5：**遍历库所s可达的变迁，得到变迁可达的库所列表。

1. 计数加1（初试状态为0）代表第一次访问；
2. 如果存在环且环执行次数超过一次，计数器减1，重新开始步骤5；
3. 如果当前变迁是并行结构的终点，且该变迁的输入库所都在places-WithToke中，即到达该变迁的库所都填入令牌，则代表当前变迁是可执行状态，执行此变迁。消耗变迁t的所有输入库所的令牌，将其输出库所填入令牌（将placesWithToken重置为该变迁的输出库所），并将此变迁加入到执行路径transPath中，执行步骤2；
4. 如果当前变迁是并行结构的终点，且该变迁的输入库所不都在places-WithToken中，即此变迁的输入库所都未填入令牌，则代表当前变迁不具备执行状态，则返回步骤4；
5. 如果是顺序结构或者冲突结构，则在placesWithToken去掉当前库所s，加入该变迁到执行路径transPath中，执行步骤2；
6. 计数减1.

**步骤6：**返回执行路径集合。

### 伪代码：



算法3

## 日志结果写入文件

### 算法设计：

**算法**：**将变迁执行路径集合写入文件中**

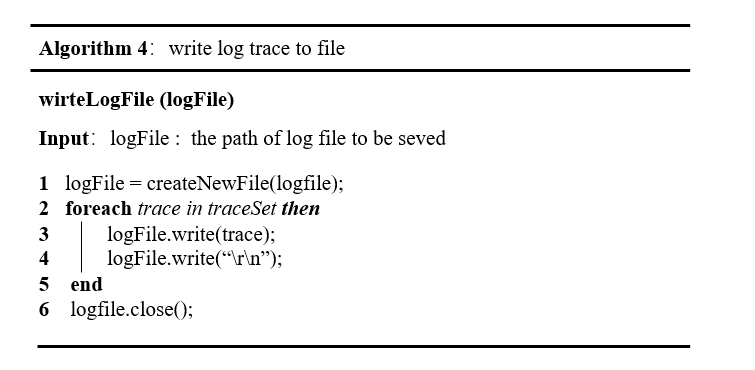
**输入**：日志文件路径

**步骤1**：新建日志文件

**步骤2：**遍历变迁路径集合，取出根据每一条路径中的变迁id得到变迁名称，作为写入到文件中的一条过程日志。

**步骤3：**将路径写入文件中

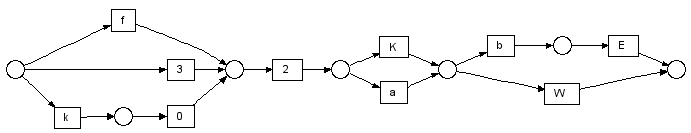
### 伪代码：



算法4

# 测试用例及结果

**Model1：**

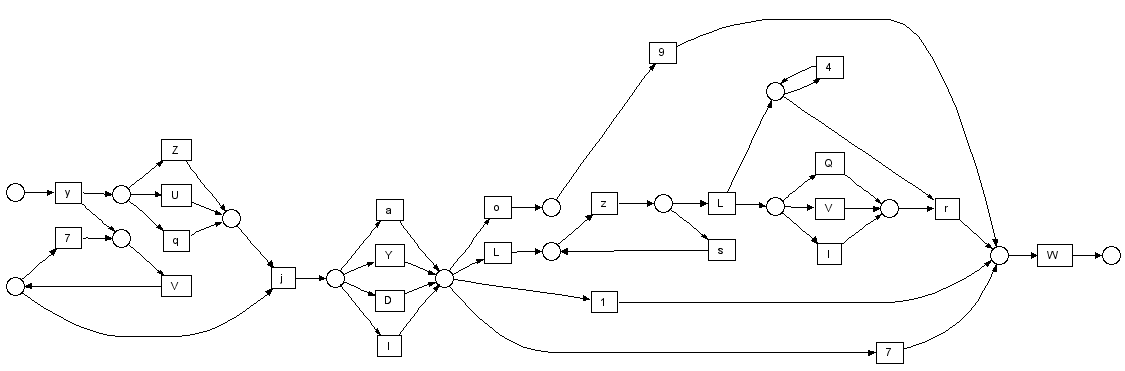


**运行结果：**

共12条过程日志，生成的日志文件内容如下图：



**Model2：**

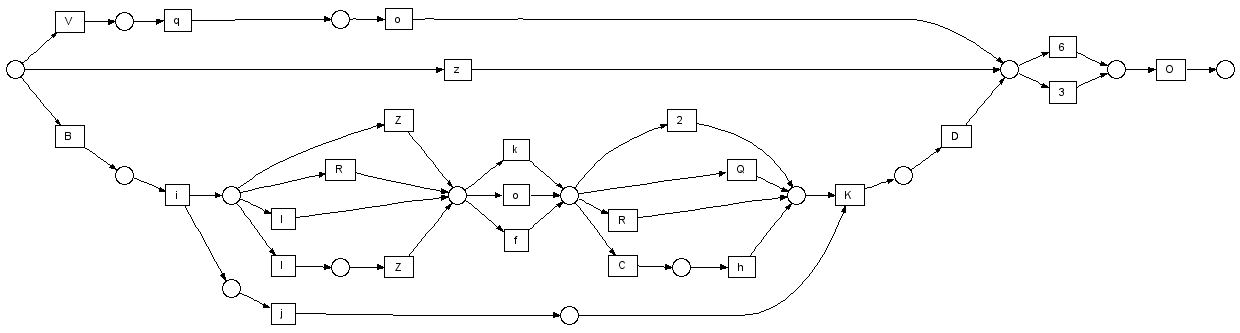


**运行结果**：

共1512条过程日志，生成的日志文件部分如下图



**Model3：**



**运行结果：**

共436条过程日志，生成的日志文件部分如下图

