# 基于 flink 的出租车流式数据模式挖掘

李易非　　熊苗　　林新迪

2019 年 4 月 1 日

# 目录

# 1 简介

# 2 数据集介绍

项目所使用的数据集由伍赛老师提供，包含新加坡的 1001 辆出租车在 2008 年 8 月每天不同时间的位置信息和状态信息等。

## 2.1 1Klogsdb

项目所使用数据所在文件夹，1.47G，包含 1001 个 txt 文件，每个 txt 文件以出租车的牌照号码命名，包含了该辆出租车自 2008 年 8 月 1 日至 2008 年 8 月 31 日的活动信息。

## 2.2 SH7319T.txt

此为包含牌照为 SH7319T 的出租车信息的 txt 文件案例。一个 txt 文件包含该出租车不同时刻的位置信息，状态信息等，每行代表一次采样信息，不同类型信息以逗号隔开。txt 文件信息呈现半有序状态，同一天的的采样按采样时间顺序排列，但所有采样并未按照先后顺序排列。

表 1: txt 文件数据信息样例

| 时间戳 | 出租车牌照 | 所在经度 | 所在纬度 | 某状态值 | 状态 |
|---|---|---|---|---|---|
| 24/08/2008 02:30:34 | SH1033G | 103.8927 | 1.3767 | 27 | FREE |

# 3 服务器 Flink 环境搭建

## 3.1 环境配置

- 将官方下载的 Flink 压缩包上传到云服务器，并解压
- 设置 Hadoop 的环境路径。因为官方的 Flink 版本中附带 Hadoop 最高 2.7，所以我们下载了 Hadoop Free 的版本，通过自己设置 HADOOP CLASSPATH 来关联 FIink 和 Hadoop。同时为了方便运行 Flink，我们将 FlinkBin 添加进环境变量

```
vi ~/.bashrc
export HADOOP_CLASSPATH=`hadoop classpath`
export FLINK_BIN=/root/flink/flink-1.7.2/bin
```

- 进入 conf/flink-conf.yaml 文件配置 Flink，设置相应参数只需要将 jobmanager.rpc.address 改成 host0 即可，因为 Flink 环境需要共享到另外 4 个 taskmanager 处，如果默认 localhost 可能会出错。至于其他参数，可以随着我们的运行需求进行更改。

```
jobmanager.rpc.address: host0

# The RPC port where the JobManager is reachable.

jobmanager.rpc.port: 6123

# The heap size for the JobManager JVM

jobmanager.heap.size: 1024m

# The heap size for the TaskManager JVM

taskmanager.heap.size: 1024m

# The number of task slots that each TaskManager offers. Each
    slot runs one parallel pipeline.

taskmanager.numberOfTaskSlots: 1
```

- 配置 master 和 slaves 文件 Masters:

```
host0:8081
```

Slaves: 设置 5 个 worker 的对应 IP 地址或花名，每个 worker 结点都会运行一个 TaskManager

```
host0
slave1
```

```
slave2
slave3
slave4
```

每个 IP 地址的花名通过进入 etc/hosts 文件并添加如下花名实现。此时即可通过 ssh slaveX 进入相应节点，并通过 exit 退出。

```
sudo vi /etc/hosts

10.173.32.4   host0
10.173.32.7   slave1
10.173.32.8   slave2
10.173.32.9   slave3
10.173.32.10  slave4
```

实现以上步骤之后，把配置文件 hosts + Flink 系统通过 scp 发送到其他四个节点

```
scp -r ~/flink.. slave1:~
```

## 3.2 环境测试

- 通过 `bash <(curl https://flink.apache.org/q/sbt-quickstart.sh)` 构建一个 scala 文件，这个过程可以在当前目录新建一个 flink-project（如果不改文件名，回车即可，也可以自己设定名字，遇到 version 直接回车即可）
- 进入`/root/flink/flink-1.7.2/flink-project/src/main/scala/dataBig/`里面找到.scala 文件，用自己写的代码替换
- 回到 flink-project 目录 (一定要回到这个 project 的根目录)，输入 `sbt package`
- 然后回到 flink-2.7 的那个总目录下运行`./bin/start-cluster.sh./bin/jobmanager.sh` 或者是 `./bin/standalone.sh`
- 运行代码 `./bin/flink run *.jar` --输入对应需要的参数
- 查看 dashboard `http://59.111.99.122:8081//`

# 4 数据层方法

## 4.1 数据预处理

此阶段的数据预处理在利用 Flink 产生模拟数据之前进行，在预处理之后，再将数据发送给进行车辆总数统计和频繁模式的算法进行计算。

### 4.1.1 清洗清洗

针对原始数据，我们清洗本项目不需要的数据列（某状态只、车辆状态），去除有缺少有效信息以及格式不符合要求的数据记录。而最终每条有效记录包含如下信息。

```
/**
*simpleTaxiID
*       由车辆牌照简化的车辆编码（去除前缀）
*grid
*       由车辆经纬度计算出的车辆所在方格
*/
public DateTime timeInfo;
public String taxiID;
public int simpleTaxiID;
public float longitude;
public float latitude;
public int grid;
```

### 4.1.2 经纬度转 grid

根据新加坡所在的经纬度范围，本项目将新加坡的经度跨度和纬度跨度分别 20 等分，划分为 400 个方格并按照顺序编号，根据每条记录中车辆所在的经纬度信息计算车辆所在的方格序号。

```
public static float bottomLatitude = (float)1.199148;
public static float topLatitude = (float)1.480820;
public static float leftLongtitude = (float)103.542195;
public static float rightLongtitude = (float)104.128511;

/public int getGrid(float longitude, float latitude) {
```

```
int y = Math.round(19 * (latitude - bottomLatitude) / (topLatitude -
    bottomLatitude));
int x = Math.round(19 * (longitude - leftLongtitude) /
    (rightLongtitude - leftLongtitude));

return x + 1 + y * 20;

}
```

### 4.1.3 Flink 流数据模拟

在本次项目中，我们需要建立一个能够处理真实世界流数据的系统，但是我们并未解除真正的流式数据，这就要求我们自行模拟这一过程。收到 flink 作者 training tutorial 的启发，我们采用了重载 sourceFunction 的方式来完成这个需求。

```
public class TaxiLogSource implements SourceFunction<TaxiLog> {
private final int watermarkDelayMSecs;

private final String dataFilePath;
private final int servingSpeed;

@Override
public void run(SourceContext<TaxiLog> sourceContext) throws
    Exception {}
@Override
public void cancel() {}
}
```

其中 **watermarkDelayMsecs** 指明了两个水印的时间差，**dataFilePath** 是我们预先按照时间戳排好序的文件路径，**servingSpeed** 代表流数据模拟的速度，在本次项目中，我们采用 **servingSpeed=600** 的机制，即 1s 对应数据中的 10min。在项目早期阶段，我们采用了一种非常简单的策略来实现 run 函数，按行读取文件，获取记录时间戳，通过 **servingSpeed** 与该条记录的时间戳计算出该条记录应该发出的时间，若还未到，则通过 **Thread.sleep()** 进行休眠，直到该条记录发出时间到达。在主函数中，我们设置 **TimeCharacteristi**c 为 processing time, 通过 6s 的 TimeWindow

来模拟题目中要求的 1h 窗口。

```
while((line = reader.readLine()) != null) {
taxiLog = TaxiLog.fromSimpleString(line);
if(taxiLog.grid > 400 | taxiLog.grid <= 0)
continue;
long now = Calendar.getInstance().getTimeInMillis();
long servingTime = (taxiLog.getEventTime() - dataStartTime) /
    this.servingSpeed + servingStartTime;
long waitTime = servingTime - now;
Thread.sleep((waitTime > 0) ? waitTime : 0);
sourceContext.collect(taxiLog);
}
```

但在后期，我们发现这样发出的数据还是在局部短时间内有错位，Flink 并不是完全按照发出的顺序接受数据，并且这样的实现方法也不符合该项目需要我们处理 Event Time 的初衷。为了处理 Event Time，我们的 sourceFunction 也必须按照一定的时间发出 watermark 水印，才能让 Flink 正常处理 Event Time 的流数据。最终我们仿照上文提到的教程，通过堆来维护读入的出租车记录与水印，并且在读进一定数量的记录后再发出，这样类 Buffer 的机制也提高了程序的效率。

```
PriorityQueue<Tuple2<Long, Object>> emitSchedule = new
    PriorityQueue<>();
while(emitSchedule.size() > 0 || reader.ready()) {
long curNextEventTime = !emitSchedule.isEmpty() ?
    emitSchedule.peek().f0 : -1;
long logEventTime = taxiLog != null? taxiLog.getEventTime() : -1;
while(taxiLog != null && ( // while there is a ride AND
emitSchedule.isEmpty() || // and no ride in schedule OR
logEventTime < curNextEventTime) // not enough rides in schedule
) {
emitSchedule.add(new Tuple2<Long, Object>(logEventTime, taxiLog));
read next log
}

// emit schedule is updated, emit next element in schedule
Tuple2<Long, Object> head = emitSchedule.poll();
long eventTime = head.f0;
```

```
long now = Calendar.getInstance().getTimeInMillis();
long servingTime = toServingTime(eventTime, dataStartTime,
    servingStartTime);
long waitTime = servingTime - now;
Thread.sleep( (waitTime > 0) ? waitTime : 0);
if head.f1 is taxiLog
emit taxiLog
else
emit waterMark
add next waterMark
}
```

通过上述的 sourceFunction , 我们只需在主程序中将 TimeCharacteristic 设置为 Event Time，timeWindow 设置为正常的 1h，就可以实现题目中的输入需求。

#### 4.1.4 数据传输

对于经过清洗和排序等数据预处理操作的数据，按照记录时间排序。sliding window 大小是 1 个小时，每次滑动为 30min。因此，针对频繁路线挖掘问题，数据流系统将每半小时内的所有路径信息路径写入一个 txt 文件，并将此 txt 文件的地址发送路给算法层进行频繁路线。挖掘我们使用 Flink

# 5 算法层方法

## 5.1 车辆统计算法

在经过与老师和助教的讨论后，我们总结第一问的要求如下：

- 读入数据，并在 10 分钟内进行数据去重（111222333 -> 123）
- 第一小问需要计算的是窗口滑动的前一刻（也是下个窗口开始的一刻），每个 grid 上现存的车辆数
- 第二小问需要计算的是一个窗口内每个 grid 上平均 10 分钟的车辆数（去重后的）

考虑到在上面的要求中，我们既需要计算每辆车的去重路径，也要计算每个 grid 上的车辆数，单纯在 grid 或 taxiLog 上 KeyBy 都不能满足要求，所以我们利用了 timeWindowAll 并通过实现 processAllWindowFunction 的方式来完成上述要求；在 processFunction 中，我们首先需要计算去重后的出租车路径，在此基础上，再计算 grid 上的要求：

```
var carTenMinutesGapArray : Map[String, Array[ArrayBuffer[(Int,
    Long)]]] = Map()
```

carTenMinutesGapArray 是一个 Map 结构，key 为出租车 ID，value 是一个长度为 6 的 Array，每一个元素分别对应一个 10 分钟时间段内的车辆去重路径（通过 ArrayBuffer 进行存储），Map 的构建过程也很简单，读入一个窗口内的 Log, 通过不同的 taxiID 与时间戳加入不同的 ArrayBuffer 中即可；

```
for(log : Log <- elements if log != null) {
val taxiID = log.taxiID
val eventTime: Long = log.eventTime
val grid : Int = log.grid
val timeSlotIdx : Long = (eventTime - startEventTime) / tenMinutesGap
carTenMinutesGapArray(taxiID)(timeSlotIdx.toInt).append((grid,
    eventTime))
}
```

去重之前，每一个 ArrayBuffer 中的轨迹都要通过时间戳进行排序，所以我们只需要去除每个 10 分钟内的连续重复 grid 即可

```
var lastElement : Int = trackBuffer(0)
while(j < trackBuffer.length) {
val curElement: Int = trackBuffer(j)
if(curElement == lastElement) {
lastElement = curElement
trackBuffer.remove(j)
}
else {
lastElement = curElement
j += 1
}
```

```
}
```

获得每辆出租车的去重轨迹后，获得每个 grid 上的车辆数目只需遍历即可.

## 5.2 频繁路线挖掘 PrefixSpan

### 5.2.1 算法简介

频繁模式算法的基础是频繁项集，支持度等概念。项目组首先了解了相关概念。称 $I = \{i_1, I-2, ..., i_m\}$ 为项 (item) 的集合，$D = /T_1, T_2, ...T_n/$ 为事物数据集（Transaction Data Itemsets ），而事务 $T_i$ 由 $I$ 中的若干项组成。设 $S$ 为由项组成的一个集合，$S = \{i|i \in I\}$，简称为项集。$T$ 为某一条事务，如果 $S \subseteq t$，则成事务 $t$ 包含项集 $S$。S 的支持度：$support(S) = (\frac{TransWithSNum}{TransNum})$ 在大部分算法中，参数包含了支持度的阈值，如果某一个项集的支持度大于等于该阈值，则成 S 为频繁项集。而在此次的项目中，每一个 grid 即为项（item），一辆车的整体路线即为一个事务，而所需要挖掘的频繁路线即为频繁项集。Apriori 和 FP-Tree 是挖掘频繁项集的经典算法。但是此题中的频繁路线实质上不是频繁项集，而是频繁序列（集合不考虑元素的顺序，而序列考虑）。因此需要采用 GSP 算法，PrefixSpan 算法等挖掘频繁序列的算法。此项目采用的是 PrefixSpan 算法，因为其所做的 projection 运算较少并且序列集合收缩较快，因此有更好的运算表现。

PrefixSpan 算法的全称是 Prefix-Projected Pattern Growth，即前缀投影的模式挖掘。在 PrefixSpan 算法中的前缀 prefix 通俗意义讲就是序列数据前面部分的子序列。给定某序列和前缀，在该序列中去掉前缀剩下的部分即为后缀，在 PrefixSpan 算法中，某一个前缀对应的所有后缀的结合我们称为前缀对应的投影数据库（即为包含此前缀的所有序列减去该前缀剩余的后缀的集合）。PrefixSpan 算法从长度为 1 的前缀开始挖掘序列模式，搜索对应的投影数据库得到长度为 1 的前缀对应的频繁序列，然后递归的挖掘长度为 2 的前缀所对应的频繁序列，以此类推，一直递归到不能挖掘到更长的前缀，即挖掘到某一前缀的投影数据库为空集为止。

在本项目中的参数设置如下：SUPPORT 即为频繁路径的支持度阈值，LENGTH 为频繁路径的长度阈值。

```
public static final int carNum = 1001;
private static final int SUPPORT = 5;
public static final int LENGTH = 3;
```

```
public static final int Infinity = 1000000;
```

PrefixSpan 主要函数与方法:

```
public static void main(String[] args) {

//读入数据层半个小时内的路径数据存入二维数组
Integer[][] matrix = Reader.readAsMatrix(
filepath, "\t", "UTF-8", carNum);
// 生成长度为1的前缀投影数据集并进行筛选
Map<Integer, List<Integer[]>> header = getHeader(matrix);

List<ArrayList<Integer>> res = new ArrayList<>();

for(Map.Entry<Integer, List<Integer[]>> entry : header.entrySet() )
{
//对前缀进行递归挖掘，更新投影数据集与前缀集合
List<ArrayList<Integer>> list = calRes(entry.getKey(),
    entry.getValue(), matrix, 1);
if(list != null) res.addAll(list);
}
//删除重复记录的频繁路径
deleteDuplicate(res);
//输入频繁路径
printList(res);
}
```

### 5.2.2 数据读入

针对数据层传入的包含半个小时的所有车辆路线的 txt 文件，本项目声明并实现了两种不同的读取出租车位置数据的方式，第一种方式就是普通的 JAVA IO 操作。

```
public static String readFile(String fileName, String encoding)
```

但是此操作不利于后续的算法计算，因此我们实际上采取了第二种操作，将所有的车的路径信息保存成一个二维数组方便读取车辆的 grid 信息。首先声明一个固定列数的二维矩阵，读取 txt 中每行的数据，分配空间，并将数

据类型转化为整数（toIntArray）类型。

```
/**
* 以非整齐的二维表的形式读取文件
* 要求严格按照车子的顺序排列路径
* @param fileName
*          文件名
* @param regex
*          文件行内的分隔符
* @param encoding
*          编码方式
* @return matrix 二维表
*/
public static Integer[][] readAsMatrix(String fileName, String regex,
    String encoding, int carNum)

public static void toIntArray(String[] strs, Integer[] res)
```

### 5.2.3 重复路径统计：1-前缀投影数据库

此步骤找出所有长度为 1 的前缀和对应的投影数据库。统计出每一个方格（grid）在矩阵中的出现位置。所有的路径都存储在了一个二维数组（matrix）中，遍历这个数组，生成 <gridID,[a,b]> 元素对，其中 gridID 为键（key），[a,b] 为该 gridID 的方格出现在二维数据（matrix）中的索引。这样生成的 Map 即为长度为 1 的前缀的投影数据库。

```
public static Map<Integer, List<Integer[]>> getHeader(Integer[][]
    matrix)
{
Map<Integer, List<Integer[]>> countMap = new HashMap<Integer,
    List<Integer[]>>();

// 遍历数组得到map
for (int i=0; i< matrix.length; i++) {
int length = matrix[i].length;
for (int j=0; j < length; j++) {
Integer[] temp = {i, j};
if (countMap.containsKey(matrix[i][j])) {
```

```
countMap.get( matrix[i][j] ).add(temp);
} else {
List<Integer[]> value = new ArrayList<Integer[]>();
value.add(temp);
countMap.put(matrix[i][j], value);
}
}
}
```

### 5.2.4 1-前缀筛选

对长度为 1 的前缀进行计数，将支持度低于阈值 的前缀对应的项从数据集 S 删除，同时得到所有的频繁 1 项序列。再用阈值（5）对 Map 中的元素对进行筛选，投影数据库的元素个数小于阈值的方格被删除，留下来的即为频繁的方格。

```
Map<Integer, List<Integer[]>> frequentMap = new HashMap<Integer,
    List<Integer[]>>();
// 过滤掉不满足支持度的项
for (Map.Entry<Integer, List<Integer[]> > entry :
    countMap.entrySet()) {
if (entry.getValue().size() >= SUPPORT)
frequentMap.put(entry.getKey(), entry.getValue());
}
```

### 5.2.5 前缀递归挖掘

对于每个长度为 length 满足支持度要求的前缀进行递归挖掘：

a) 找出前缀所对应的投影数据库。如果投影数据库为空，则递归返回。

b) 统计对应投影数据库中各项的支持度计数。如果所有项的支持度计数都低于阈值 ，则递归返回。

c) 将满足支持度计数的各个单项和当前的前缀进行合并，得到若干新的前缀。

d) 令 length=length+1，前缀为合并单项后的各个前缀，分别递归执行第 3 步。

更新前缀投影数据库：

```
private static void getMap(Integer[][] matrix, List<Integer[]> value,
    Map<Integer, List<Integer[]>> frequentMap){

Map<Integer, List<Integer[]>> countMap = new HashMap<Integer,
    List<Integer[]>>();
int i, j;
for(Integer[] item: value)
{
i = item[0]; j = item[1]+1;
// 即item记录的位置的下一个位置（即路径的下一步）
if(j < matrix[i].length)
{
Integer[] temp = {i, j};
if (countMap.containsKey(matrix[i][j])) {
countMap.get( matrix[i][j] ).add(temp);
countMap.put(matrix[i][j], countMap.get( matrix[i][j] ));
} else {
List<Integer[]> tempvalue = new ArrayList<Integer[]>();
tempvalue.add(temp);
countMap.put(matrix[i][j], tempvalue);
}
}
}

// 过滤掉不满足支持度的项
for (Map.Entry<Integer, List<Integer[]> > entry :
    countMap.entrySet()) {
if (entry.getValue().size() >= SUPPORT)
frequentMap.put(entry.getKey(), entry.getValue());
}
}
```

递归计算频繁路径的后缀，并递归挖掘的频繁路径:

```
private static List<ArrayList<Integer>> calRes(Integer key,
    List<Integer[]> value, Integer[][] matrix, int length)
{
List<ArrayList<Integer>> list = new ArrayList<ArrayList<Integer>>();
```

```
Map<Integer, List<Integer[]>> frequentMap = new HashMap<Integer,
    List<Integer[]>>();
getMap(matrix, value, frequentMap);

if(frequentMap.isEmpty())
{
if(length >= LENGTH)
{
ArrayList<Integer> temp = new ArrayList<Integer>() ;
temp.add(value.size());
temp.add(key);
list.add(temp);
return list;
}else return null;
}else
{
// 添加中间路径，如12345 12345 123457 ，最后可能返回12345: 2 和1234: 3
if(length >= LENGTH && frequentMap.size() < value.size())
{
ArrayList<Integer> temp = new ArrayList<Integer>() ;
temp.add(value.size());
temp.add(key);
list.add(temp);
}

for(Map.Entry<Integer, List<Integer[]> > entry :
    frequentMap.entrySet())
{
List<ArrayList<Integer>> tempList = calRes(entry.getKey(),
    entry.getValue(), matrix, length + 1);
if(tempList != null)
{
for(ArrayList<Integer> item : tempList)
{
// 把templist里面的每个元素添加上当前的key，添加到本轮的list里，返回
item.add(key);
list.add(item);
}
}
```

```
}
return list;
}
}
```

### 5.2.6 PrefixSpan 算法小结

PrefixSpan 算法由于不用产生候选序列，且投影数据库缩小的很快，内存消耗比较稳定，作频繁序列模式挖掘的时候效果很高。比起其他的序列挖掘算法比如 GSP,FreeSpan 有较大优势，因此是在生产环境常用的算法。

PrefixSpan 运行时最大的消耗在递归的构造投影数据库。如果序列数据集较大，项数种类较多时，算法运行速度会有明显下降。此项目的出租车频繁路径的挖掘的的项数种类较少，只有 400 个方格，但序列集合较大，包含了许多可能的路径，可能会影响算法表现。

## 6 实验结果

对于车辆统计结果，可以看到每个 grid 上的车辆数无论是平均还是窗口滑动时刻都是较为稀疏的数组，并且有数据的 grid 呈现局部连续的特征，在路径频繁挖掘的结果中我们也可以看到这一点。大量的 grid 是出租车容量很少，而少量的 grid 拥有大量的出租车。就路径挖掘结果来看，3 个 grid 的频繁路径比较常见，且大多都是局部连续的状态，这跟尝试也比较相符，通常来说出租车都在小范围内移动，每次载客车程都不会太长。由于结果输出内容较多，部分结果将于附录中给出。

## 7 团队分工

李易非：本地 Flink 环境搭建，数据预处理与 Flink 流数据模拟
熊苗：服务器 Flink 环境搭建，频繁路线挖掘算法设计与实现
林新迪：车辆统计算法设计与实现, 项目内容整合与报告撰写

## 8 附录：实验结果

```
01 Aug 2008 02:00:00 => 01 Aug 2008 03:00:00
Current Cars on Grid:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 5 19 6 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 9 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 7 0 2 2 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 1 0
    7 4 13 0 0 0 0 0 0 0 0 0 0 0 0 6 21 4 3 13 19 5 9 5 0 0 0 0 0
    0 0 0 0 0 0 1 8 0 1 43 21 0 19 4 2 0 0 0 0 0 0 0 0 0 0 15 13 11 3
    2 25 10 0 29 3 0 0 0 0 0 0 0 0 0 0 8 5 7 3 5 19 12 17 10 5 0 0 0
    0 0 0 0 0 1 0 1 2 10 4 2 26 49 14 8 0 0 0 0 0 0 0 0 0 0 0 0 0 4
    9 10 36 2 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 22 30 3 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 1 10 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Average Cars on Grid:
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.16666667 0.8333333
    0.16666667 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 1.5 5.6666665 6.3333335 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 6.0 11.166667
    8.333333 4.1666665 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 2.8333333 2.8333333 8.0 4.8333335 0.0
    0.16666667 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 7.0 0.33333334 8.5 1.5 2.1666667 1.6666666 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.5 7.3333335 1.8333334
    8.333333 11.0 3.1666667 9.666667 0.0 1.0 0.33333334 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 5.0 13.333333 4.6666665 3.6666667
    19.666666 15.666667 7.5 7.3333335 4.6666665 0.8333333 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.3333334 2.5 7.5 4.0 2.5 39.333332
    18.333334 2.1666667 13.833333 8.166667 1.6666666 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 8.666667 13.666667 18.0 14.166667 3.5 39.0
    15.833333 1.8333334 24.5 7.8333335 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 1.0 5.8333335 7.6666665 13.833333 9.833333 14.0 46.5
    28.666666 24.0 16.5 6.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.8333333 1.0 1.0 2.6666667 18.666666 9.0 10.833333 58.666668
    66.166664 21.333334 9.333333 1.8333334 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.16666667 10.0 15.333333 16.833334
```

```
    71.333336 14.5 11.333333 2.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 2.3333333 12.166667 33.666668 59.166668
    2.3333333 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 2.5 14.166667 20.166666 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    0.0 0.0 0.0 0.0 0.0 0.0
frequent pattern:
130 ->131 ->151 : 11
130 ->131 ->151 ->171 : 9
130 ->131 ->151 ->171 ->191 : 7
130 ->131 ->151 ->171 ->191 ->211 : 7
130 ->131 ->151 ->171 ->191 ->211 ->231 : 5
130 ->110 ->89 : 5
130 ->110 ->89 ->69 : 5
130 ->110 ->90 : 12
130 ->110 ->90 ->70 : 6
131 ->130 ->110 : 13
131 ->130 ->110 ->90 : 6
131 ->131 ->151 : 5
131 ->151 ->171 : 16
131 ->151 ->171 ->191 : 12
131 ->151 ->171 ->191 ->211 : 12
131 ->151 ->171 ->191 ->211 ->231 : 8
133 ->133 ->153 : 5
133 ->153 ->152 : 5
133 ->153 ->154 : 6
133 ->153 ->154 ->174 : 5
133 ->153 ->154 ->174 ->175 : 5
269 ->249 ->248 : 9
269 ->249 ->248 ->228 : 8
269 ->249 ->269 : 5
269 ->269 ->269 : 6
269 ->270 ->290 : 9
269 ->270 ->270 : 9
```

```
269 ->270 ->270 ->270 : 5
270 ->290 ->290 : 5
270 ->290 ->291 : 6
270 ->290 ->270 : 14
270 ->290 ->270 ->269 : 5
270 ->290 ->270 ->270 : 6
270 ->269 ->249 : 9
270 ->270 ->250 : 6
270 ->270 ->270 : 44
270 ->270 ->270 ->270 : 25
270 ->270 ->270 ->270 ->270 : 13
270 ->270 ->270 ->270 ->270 ->270 : 5
270 ->270 ->270 ->271 : 5
270 ->270 ->271 : 15
270 ->270 ->271 ->271 : 8
270 ->271 ->251 : 8
270 ->271 ->251 ->231 : 5
270 ->271 ->271 : 18
270 ->271 ->271 ->271 : 5
271 ->291 ->290 : 10
271 ->291 ->291 : 13
271 ->291 ->291 ->271 : 9
271 ->291 ->271 : 22
271 ->291 ->271 ->251 : 7
271 ->291 ->271 ->271 : 5
271 ->231 ->211 : 8
271 ->231 ->211 ->191 : 6
271 ->251 ->231 : 35
271 ->251 ->231 ->211 : 20
271 ->251 ->231 ->211 ->191 : 18
271 ->251 ->231 ->211 ->191 ->171 : 11
271 ->251 ->231 ->211 ->191 ->171 ->151 : 6
271 ->251 ->231 ->211 ->191 ->191 : 5
271 ->251 ->250 : 5
271 ->251 ->251 : 21
271 ->251 ->251 ->231 : 9
271 ->251 ->252 : 5
271 ->251 ->252 ->232 : 5
271 ->251 ->271 : 7
```

```
271 ->270 ->290 : 10
271 ->270 ->290 ->270 : 9
271 ->270 ->290 ->270 ->270 : 5
271 ->270 ->270 : 6
271 ->271 ->291 : 16
271 ->271 ->291 ->271 : 6
271 ->271 ->251 : 37
271 ->271 ->251 ->231 : 20
271 ->271 ->251 ->231 ->211 : 11
271 ->271 ->251 ->231 ->211 ->191 : 11
271 ->271 ->251 ->231 ->211 ->191 ->171 : 7
271 ->271 ->251 ->251 : 10
271 ->271 ->251 ->251 ->231 : 5
271 ->271 ->270 : 11
271 ->271 ->271 : 46
271 ->271 ->271 ->251 : 14
271 ->271 ->271 ->251 ->231 : 10
271 ->271 ->271 ->251 ->231 ->211 : 6
271 ->271 ->271 ->251 ->231 ->211 ->191 : 6
271 ->271 ->271 ->271 : 16
271 ->271 ->271 ->271 ->271 : 6
147 ->147 ->147 : 8
147 ->147 ->147 ->147 : 6
148 ->128 ->148 : 7
148 ->148 ->168 : 5
148 ->168 ->188 : 10
148 ->168 ->188 ->189 : 7
148 ->168 ->188 ->189 ->209 : 7
149 ->169 ->189 : 6
150 ->150 ->150 : 5
151 ->131 ->130 : 9
151 ->131 ->130 ->110 : 6
151 ->151 ->151 : 9
151 ->151 ->151 ->151 : 6
151 ->151 ->171 : 5
151 ->151 ->171 ->191 : 5
151 ->152 ->152 : 6
151 ->152 ->172 : 5
151 ->171 ->171 : 10
```

```
151 ->171 ->171 ->191 : 6
151 ->171 ->171 ->191 ->211 : 5
151 ->171 ->191 : 27
151 ->171 ->191 ->211 : 20
151 ->171 ->191 ->211 ->231 : 15
151 ->171 ->191 ->211 ->231 ->251 : 8
152 ->151 ->171 : 8
152 ->152 ->152 : 10
152 ->152 ->152 ->152 : 7
152 ->172 ->192 : 6
152 ->172 ->152 : 7
153 ->152 ->153 : 5
154 ->174 ->175 : 8
155 ->155 ->155 : 7
290 ->290 ->290 : 5
290 ->290 ->270 : 5
290 ->291 ->271 : 6
290 ->270 ->269 : 11
290 ->270 ->269 ->249 : 8
290 ->270 ->269 ->249 ->248 : 7
290 ->270 ->269 ->249 ->248 ->228 : 6
290 ->270 ->270 : 10
290 ->270 ->270 ->270 : 5
291 ->290 ->270 : 5
291 ->291 ->291 : 18
291 ->291 ->291 ->291 : 11
291 ->291 ->291 ->291 ->291 : 6
291 ->291 ->271 : 15
291 ->291 ->271 ->251 : 5
291 ->271 ->291 : 10
291 ->271 ->291 ->271 : 5
291 ->271 ->251 : 13
291 ->271 ->251 ->231 : 5
291 ->271 ->271 : 15
291 ->271 ->271 ->251 : 7
168 ->188 ->188 : 5
168 ->188 ->189 : 11
168 ->188 ->189 ->209 : 10
169 ->189 ->210 : 6
```

```
171 ->151 ->151 : 9
171 ->151 ->152 : 11
171 ->151 ->152 ->152 : 5
171 ->151 ->171 : 6
171 ->171 ->151 : 5
171 ->171 ->171 : 30
171 ->171 ->171 ->171 : 15
171 ->171 ->171 ->171 ->171 : 9
171 ->171 ->171 ->191 : 8
171 ->171 ->191 : 18
171 ->171 ->191 ->211 : 8
171 ->171 ->191 ->191 : 6
171 ->191 ->211 : 34
171 ->191 ->211 ->212 : 5
171 ->191 ->211 ->212 ->232 : 5
171 ->191 ->211 ->231 : 21
171 ->191 ->211 ->231 ->231 : 5
171 ->191 ->211 ->231 ->251 : 12
171 ->191 ->211 ->232 : 5
171 ->191 ->191 : 10
171 ->191 ->191 ->211 : 5
172 ->192 ->192 : 6
172 ->192 ->212 : 9
172 ->192 ->212 ->232 : 5
172 ->152 ->151 : 5
172 ->152 ->151 ->171 : 5
172 ->172 ->192 : 5
172 ->172 ->172 : 11
172 ->172 ->172 ->172 : 6
174 ->194 ->194 : 6
174 ->154 ->153 : 9
174 ->154 ->153 ->133 : 6
174 ->174 ->174 : 7
175 ->174 ->154 : 6
49 ->48 ->49 : 5
49 ->69 ->69 : 5
50 ->50 ->50 : 8
50 ->50 ->50 ->50 : 6
186 ->187 ->207 : 7
```

```
186 ->206 ->186 : 6
187 ->187 ->187 : 13
187 ->187 ->187 ->187 : 7
188 ->208 ->228 : 5
188 ->208 ->188 : 9
188 ->208 ->188 ->208 : 6
188 ->188 ->208 : 5
188 ->188 ->188 : 10
188 ->188 ->188 ->188 : 5
188 ->189 ->209 : 11
189 ->209 ->208 : 8
189 ->209 ->208 ->188 : 5
189 ->210 ->211 : 6
189 ->210 ->230 : 8
189 ->169 ->149 : 5
189 ->189 ->210 : 5
189 ->189 ->189 : 6
191 ->211 ->211 : 9
191 ->211 ->212 : 6
191 ->211 ->231 : 40
191 ->211 ->231 ->231 : 9
191 ->211 ->231 ->251 : 25
191 ->211 ->231 ->251 ->251 : 5
191 ->211 ->231 ->251 ->271 : 11
191 ->211 ->231 ->251 ->271 ->271 : 5
191 ->211 ->232 : 6
191 ->171 ->131 : 6
191 ->171 ->131 ->130 : 5
191 ->171 ->151 : 30
191 ->171 ->151 ->131 : 11
191 ->171 ->151 ->131 ->130 : 7
191 ->171 ->151 ->152 : 9
191 ->171 ->171 : 15
191 ->171 ->191 : 10
191 ->191 ->211 : 10
191 ->191 ->211 ->231 : 6
191 ->191 ->211 ->231 ->251 : 5
191 ->191 ->171 : 8
191 ->191 ->191 : 8
```

```
192 ->192 ->192 : 8
192 ->212 ->212 : 5
192 ->212 ->232 : 10
192 ->212 ->232 ->232 : 6
192 ->172 ->152 : 6
192 ->172 ->172 : 8
194 ->194 ->194 : 34
194 ->194 ->194 ->194 : 22
194 ->194 ->194 ->194 ->194 : 13
194 ->194 ->194 ->194 ->194 ->194 : 6
194 ->194 ->214 : 11
194 ->194 ->214 ->213 : 6
194 ->194 ->214 ->213 ->233 : 5
194 ->194 ->214 ->213 ->233 ->232 : 5
194 ->194 ->174 : 5
194 ->214 ->213 : 14
194 ->214 ->213 ->233 : 9
194 ->214 ->213 ->233 ->232 : 8
194 ->174 ->194 : 5
195 ->195 ->195 : 7
195 ->195 ->175 : 6
195 ->175 ->175 : 5
68 ->68 ->68 : 12
68 ->68 ->68 ->68 : 7
68 ->88 ->108 : 5
69 ->69 ->69 : 10
69 ->69 ->69 ->69 : 6
70 ->70 ->70 : 10
70 ->70 ->70 ->70 : 7
206 ->186 ->187 : 5
206 ->206 ->206 : 6
207 ->208 ->228 : 8
207 ->208 ->228 ->228 : 5
208 ->209 ->189 : 5
208 ->228 ->228 : 10
208 ->188 ->208 : 11
208 ->188 ->187 : 8
209 ->210 ->230 : 5
209 ->189 ->188 : 5
```

```
210 ->211 ->232 : 9
210 ->230 ->250 : 5
210 ->189 ->209 : 6
210 ->189 ->209 ->208 : 6
210 ->189 ->169 : 8
210 ->189 ->169 ->148 : 6
211 ->210 ->189 : 10
211 ->210 ->189 ->209 : 5
211 ->210 ->189 ->209 ->208 : 5
211 ->211 ->211 : 14
211 ->211 ->211 ->211 : 7
211 ->211 ->212 : 6
211 ->211 ->212 ->232 : 6
211 ->211 ->231 : 7
211 ->211 ->191 : 6
211 ->212 ->232 : 18
211 ->212 ->232 ->232 : 10
211 ->231 ->211 : 13
211 ->231 ->211 ->231 : 8
211 ->231 ->211 ->231 ->211 : 5
211 ->231 ->231 : 19
211 ->231 ->231 ->211 : 5
211 ->231 ->231 ->251 : 9
211 ->231 ->251 : 33
211 ->231 ->251 ->231 : 5
211 ->231 ->251 ->251 : 6
211 ->231 ->251 ->271 : 14
211 ->231 ->251 ->271 ->271 : 6
211 ->232 ->213 : 6
211 ->232 ->232 : 6
211 ->191 ->211 : 13
211 ->191 ->211 ->211 : 5
211 ->191 ->171 : 44
211 ->191 ->171 ->151 : 23
211 ->191 ->171 ->151 ->131 : 9
211 ->191 ->171 ->151 ->131 ->130 : 5
211 ->191 ->171 ->171 : 7
211 ->191 ->171 ->191 : 5
211 ->191 ->191 : 10
```

```
212 ->192 ->172 : 15
212 ->192 ->172 ->172 : 7
212 ->192 ->191 : 7
212 ->211 ->210 : 8
212 ->211 ->210 ->189 : 6
212 ->211 ->191 : 6
212 ->212 ->192 : 5
212 ->212 ->212 : 22
212 ->212 ->212 ->212 : 14
212 ->212 ->212 ->212 ->212 : 7
212 ->212 ->232 : 7
212 ->213 ->213 : 6
212 ->213 ->213 ->213 : 5
212 ->232 ->231 : 11
212 ->232 ->231 ->251 : 7
212 ->232 ->231 ->251 ->251 : 6
212 ->232 ->232 : 25
212 ->232 ->232 ->232 : 7
212 ->232 ->252 : 5
213 ->212 ->192 : 6
213 ->213 ->212 : 5
213 ->213 ->213 : 19
213 ->213 ->213 ->213 : 11
213 ->213 ->213 ->213 ->213 : 6
213 ->213 ->214 : 5
213 ->213 ->233 : 8
213 ->213 ->233 ->232 : 5
213 ->214 ->194 : 11
213 ->233 ->232 : 17
213 ->233 ->232 ->232 : 7
214 ->194 ->174 : 6
214 ->213 ->233 : 12
214 ->213 ->233 ->232 : 10
214 ->214 ->194 : 5
214 ->214 ->214 : 13
214 ->214 ->214 ->214 : 7
215 ->215 ->215 : 14
215 ->215 ->215 ->215 : 10
215 ->215 ->215 ->215 ->215 : 6
```

```
90 ->70 ->50 : 5
90 ->110 ->130 : 12
90 ->110 ->130 ->131 : 10
90 ->110 ->130 ->131 ->151 : 6
228 ->208 ->188 : 6
228 ->228 ->228 : 11
228 ->228 ->228 ->228 : 7
228 ->228 ->229 : 8
228 ->228 ->248 : 7
228 ->228 ->248 ->249 : 5
228 ->229 ->249 : 6
228 ->248 ->249 : 10
228 ->248 ->249 ->269 : 7
228 ->248 ->249 ->269 ->270 : 6
228 ->248 ->249 ->269 ->270 ->290 : 6
228 ->248 ->268 : 5
228 ->249 ->269 : 5
229 ->249 ->229 : 12
229 ->249 ->229 ->229 : 5
229 ->249 ->229 ->249 : 5
230 ->210 ->209 : 6
230 ->210 ->211 : 6
230 ->250 ->251 : 8
230 ->250 ->251 ->251 : 5
231 ->211 ->211 : 6
231 ->211 ->212 : 5
231 ->211 ->231 : 17
231 ->211 ->231 ->211 : 9
231 ->211 ->231 ->231 : 5
231 ->211 ->191 : 52
231 ->211 ->191 ->211 : 8
231 ->211 ->191 ->171 : 32
231 ->211 ->191 ->171 ->151 : 16
231 ->211 ->191 ->171 ->151 ->131 : 7
231 ->211 ->191 ->171 ->151 ->152 : 7
231 ->212 ->192 : 6
231 ->212 ->232 : 8
231 ->230 ->210 : 7
231 ->231 ->211 : 14
```

```
231 ->231 ->211 ->191 : 6
231 ->231 ->231 : 23
231 ->231 ->231 ->231 : 11
231 ->231 ->231 ->231 ->231 : 6
231 ->231 ->231 ->251 : 5
231 ->231 ->251 : 21
231 ->231 ->251 ->231 : 7
231 ->231 ->251 ->251 : 6
231 ->231 ->251 ->271 : 7
231 ->232 ->232 : 5
231 ->251 ->231 : 20
231 ->251 ->231 ->211 : 6
231 ->251 ->231 ->211 ->191 : 5
231 ->251 ->231 ->231 : 6
231 ->251 ->231 ->251 : 5
231 ->251 ->251 : 31
231 ->251 ->251 ->231 : 9
231 ->251 ->251 ->251 : 12
231 ->251 ->251 ->251 ->251 : 5
231 ->251 ->251 ->271 : 6
231 ->251 ->271 : 26
231 ->251 ->271 ->270 : 7
231 ->251 ->271 ->271 : 8
231 ->191 ->171 : 7
232 ->211 ->191 : 5
232 ->212 ->192 : 5
232 ->212 ->211 : 16
232 ->212 ->211 ->210 : 7
232 ->212 ->211 ->210 ->189 : 5
232 ->212 ->211 ->191 : 5
232 ->212 ->231 : 5
232 ->213 ->214 : 8
232 ->213 ->214 ->194 : 7
232 ->231 ->231 : 7
232 ->231 ->251 : 11
232 ->231 ->251 ->251 : 8
232 ->232 ->211 : 7
232 ->232 ->212 : 14
232 ->232 ->212 ->211 : 7
```

```
232 ->232 ->213 : 6
232 ->232 ->231 : 6
232 ->232 ->232 : 76
232 ->232 ->232 ->232 : 41
232 ->232 ->232 ->232 ->232 : 17
232 ->232 ->232 ->232 ->232 ->232 : 6
232 ->232 ->232 ->233 : 6
232 ->232 ->232 ->252 : 6
232 ->232 ->233 : 12
232 ->232 ->251 : 5
232 ->232 ->252 : 20
232 ->232 ->252 ->232 : 9
232 ->232 ->252 ->232 ->232 : 5
232 ->232 ->252 ->251 : 11
232 ->232 ->252 ->251 ->231 : 5
232 ->233 ->213 : 8
232 ->233 ->233 : 5
232 ->233 ->234 : 5
232 ->251 ->271 : 5
232 ->252 ->232 : 13
232 ->252 ->232 ->232 : 7
232 ->252 ->251 : 18
232 ->252 ->251 ->271 : 9
232 ->252 ->252 : 7
233 ->213 ->213 : 5
233 ->232 ->232 : 15
233 ->232 ->232 ->232 : 5
233 ->233 ->232 : 7
233 ->233 ->233 : 15
233 ->233 ->233 ->233 : 9
233 ->233 ->233 ->233 ->233 : 5
233 ->234 ->214 : 7
233 ->253 ->233 : 7
233 ->253 ->253 : 8
234 ->234 ->234 : 5
234 ->235 ->215 : 5
234 ->254 ->253 : 7
234 ->254 ->253 ->252 : 6
234 ->254 ->253 ->252 ->272 : 5
```

```
108 ->108 ->108 : 10
108 ->108 ->108 ->108 : 6
110 ->130 ->131 : 15
110 ->130 ->131 ->151 : 8
110 ->130 ->131 ->151 ->171 : 6
110 ->89 ->69 : 6
110 ->90 ->70 : 7
248 ->228 ->208 : 5
248 ->248 ->248 : 10
248 ->248 ->248 ->248 : 6
248 ->249 ->269 : 9
248 ->249 ->269 ->270 : 7
248 ->249 ->269 ->270 ->290 : 7
248 ->268 ->248 : 5
249 ->229 ->249 : 9
249 ->229 ->249 ->229 : 7
249 ->248 ->228 : 10
249 ->249 ->229 : 8
249 ->269 ->269 : 6
249 ->269 ->270 : 12
249 ->269 ->270 ->290 : 8
250 ->231 ->211 : 5
250 ->250 ->250 : 14
250 ->250 ->250 ->250 : 8
250 ->250 ->250 ->250 ->250 : 5
250 ->250 ->251 : 5
250 ->251 ->251 : 13
250 ->251 ->251 ->231 : 5
250 ->251 ->271 : 7
250 ->270 ->271 : 6
251 ->231 ->211 : 41
251 ->231 ->211 ->191 : 33
251 ->231 ->211 ->191 ->171 : 22
251 ->231 ->211 ->191 ->171 ->151 : 11
251 ->231 ->211 ->191 ->171 ->151 ->131 : 6
251 ->231 ->211 ->191 ->191 : 6
251 ->231 ->212 : 10
251 ->231 ->212 ->232 : 6
251 ->231 ->230 : 13
```

```
251 ->231 ->230 ->210 : 6
251 ->231 ->231 : 15
251 ->231 ->231 ->231 : 5
251 ->231 ->251 : 18
251 ->231 ->251 ->251 : 7
251 ->231 ->191 : 6
251 ->231 ->191 ->171 : 5
251 ->250 ->251 : 8
251 ->250 ->251 ->251 : 5
251 ->251 ->231 : 34
251 ->251 ->231 ->211 : 6
251 ->251 ->231 ->212 : 7
251 ->251 ->231 ->230 : 7
251 ->251 ->231 ->251 : 5
251 ->251 ->250 : 5
251 ->251 ->251 : 52
251 ->251 ->251 ->231 : 6
251 ->251 ->251 ->251 : 25
251 ->251 ->251 ->251 ->251 : 13
251 ->251 ->251 ->271 : 11
251 ->251 ->251 ->271 ->291 : 6
251 ->251 ->271 : 25
251 ->251 ->271 ->251 : 6
251 ->251 ->271 ->271 : 9
251 ->252 ->232 : 13
251 ->252 ->232 ->232 : 7
251 ->271 ->291 : 15
251 ->271 ->291 ->291 : 6
251 ->271 ->251 : 16
251 ->271 ->251 ->251 : 5
251 ->271 ->270 : 9
251 ->271 ->270 ->290 : 5
251 ->271 ->270 ->290 ->270 : 5
251 ->271 ->271 : 33
251 ->271 ->271 ->291 : 9
251 ->271 ->271 ->251 : 7
251 ->271 ->271 ->271 : 10
252 ->232 ->232 : 19
252 ->232 ->232 ->232 : 5
```

```
252 ->232 ->232 ->232 ->232 : 5
252 ->232 ->252 : 5
252 ->251 ->251 : 5
252 ->251 ->271 : 12
252 ->251 ->271 ->271 : 6
253 ->233 ->232 : 6
253 ->233 ->253 : 5
253 ->252 ->272 : 6
253 ->253 ->233 : 6
253 ->253 ->253 : 6
```