# TP2 C++

B3111 : Edern HAUMONT et Nicolas SIX

Mercredi 11 novembre 2015

# Table des matières

# 1 Introduction

## 1.1 Choice of the data structure

Our application must support 20 million events from 1500 sensors. Consequently, we chose not to store event themselves as they arrive. However, at the launch of our program, several static arrays of entire values are created. Their indexes correspond to the information used to anwer to the user querries, for instance the sensor number, the hour,etc. Their last dimension corresponds to a sensor color. So when an event is added, we just increment one cell in each array.

# 2 Classes

## 2.1 DataHandler

All data additions and request are managed by the class DataHandler which contain itself the 4 data arrays.

### 2.1.1 Constants

This class use differents constants, first there are one error code used when the trafic char is not one of the four expected : 'V','J','R','N'

```
const unsigned int ERROR_INVALID_TRAFIC_UCHAR = 201;
```

DataHandler also use constants to define the size of the arrays used to store all the datas.

```
const int NUMBER_OF_COLORS = 4;
const int NUMBER_OF_SENSORS = 1500;
const int NUMBER_OF_MINUTES = 1440;
const int NUMBER_OF_HOURS = 24;
const int NUMBER_OF_DAYS = 7;
```

### 2.1.2 Atributs

To make our code more readable we decided to use the following type def in the DataHandler Class.

```
typedef unsigned int uint;
```

The atributs used in this class are the following :

```
IdHash idHash;
uint sensors[NUMBER_OF_SENSORS][NUMBER_OF_COLORS];
uint days[NUMBER_OF_DAYS][NUMBER_OF_COLORS];
uint daysAndHours[NUMBER_OF_DAYS][NUMBER_OF_HOURS][NUMBER_OF_COLORS];
unsigned char *daysAndMin[NUMBER_OF_DAYS][NUMBER_OF_MINUTES][NUMBER_OF_COLORS];
```

idHash is an instance of our IdHash class, which is detailled latter. It's an hash tab that we use to link the sensors id and the position in our static tabs. sensors, days, daysAndHours and daysAndMin are four static array used to store statistics which will be used in our stats methods. The fourth array is a four dimantion one with the last one, coreponding to the sensors id, is alocated using a new in the construtor.

### 2.1.3 Public Methods

#### 2.1.3.1 addData

```
int addData(const char &trafic ,const uint &min,const uint &hours ,\
        const uint &id ,const uint &day7);
```

**description :** Method used to update member arrays in corresponding cells. Complexity : O(1).

**Contract :** The values given must be ritghtly build : min shoult be between 0 and 59, hours between 0 and 23 and day7 between 0, for monday, and 6, for sunday. To work properly trafic must be set to one of : 'V','J','R','N'.

#### 2.1.3.2 sensorStats

```
int sensorStats(uint id) const;
```

**description :** Prints the sensor statistics for an id. Complexity : O(1).

**Contract :** The id given in parameter must corespond to a sensors with an already added id else the stats will use the first added sensor. Print null stats if 0 corresponding event

#### 2.1.3.3 jamStats

```
int jamStats(uchar day7);
```

**description :** prints jam statistics for a day. Complexity : O(1).

**Contract :** day7 must be between 0 and 6 ($NUMBER\_OF\_DAYS - 1$). Print null stats if 0 corresponding event.

#### 2.1.3.4 dayStats

```
int dayStats(uchar day7);
```

**description :** prints a week day statistics for a given day. Complexity : O(1).

**Contract :** day7 must be between 0 and 6 ($NUMBER\_OF\_DAYS - 1$). Print null stats if 0 corresponding event

#### 2.1.3.5 optimum

```
int optimum(uchar day7, uint begginHours, uint endHours,\
                uint idTab[], uint tabSize);
```

**description :** optimum look for the best departure time in a given interval. Complexity : $O(tabsize * (endHours - beginHours))$.

**Contract :** day7 must be between 0 and 6 ($NUMBER\_OF\_DAYS - 1$). begginHours and endHours must be between 0 and 23 with begginHours smaller than endHours. idTab must be an array of size tabSize filled with already added sensors' id (if an id is not added the first added sensors will be used).

To compute the optimum departure time this methode consider that the time taken to go throuht a sensors is the most probable one at this time of the day acording to the added data. In case of equality on trafic information, the quiker will be used (optimistic).

## 2.2 IdHash

This class makes the link between a sensor id of the user and indexes in data tables It is a hashTable and its structure is an array. It is not designed for more than 25000 ids.

### 2.2.1 Constants

This Class use differents constants, most of them are related to the hash function.

```
const int SIZE_OF_HASHTABLE = 5000;
//hash function parameters
const int PRIME_NUMBER = 7001;
const int A = 1;
const int B = 0;
```

### 2.2.2 Atributs

The private atributs used in this class are the following :

```
unsigned * hashTable[SIZE_OF_HASHTABLE][2];
unsigned sizeOfHashList[SIZE_OF_HASHTABLE][2];
unsigned lastIdInTab;
```

hashTable is an array of two dimentions plus one last dynamic one. The first one is the hash array itself and correspond to the position given by the hash function. The segond one is there to have both the sensors id and the position on the program array. The last dimention is a way to handel case where the hash function give the same position to tow differents id.

sizeOfHashList is an two dimentions array created to now the size and the used size of each dynamic part of hashTable.

lastIdInTab keep the value of the last added position in order to just have to increment it for the next add.

### 2.2.3 Public Methods

#### 2.2.3.1 addId

```
unsigned addId(const unsigned & id);
```

**description :** add the user id to the hash table if not already in and returns an array position used by the program.

**Contract :** id in int range.

#### 2.2.3.2 getTabId

```
unsigned getTabId(const unsigned & id) const;
```

**description :** returns the array position used by the program corresponding to a sensor id called by the the user.

**Contract :** id in unsigned int range if id is not in the hash tab, return 0.

#### 2.2.4 Hash function

In order to have a good repartition of our datas on the hash tab we use the clasical :

$$hash(id) = (A * id + B)\%PRIME\_NUMBER)\%SIZE\_OF\_HASHTABLE \qquad (1)$$

With $A$, $B$, $PRIME\_NUMBER$ and $SIZE\_OF\_HASHTABLE$ the constants defined in the constents section.

This function can be improved with random $A$ and $B$ generated at the begining of the execution in order to forgive a data set to always be the worst one.

## 3 tests

**Introduction** We used a few manually generated test and we generated some bigger input files with a generation code. This script also ofently generates an expected output file.

To test or application, we made it read th input file instead of the keyboard input. We also redirect the application output to another file which is afterwards compared to the expected output file.

### 3.1 manually generated tests

#### 3.1.1 TestAdd

The first test just check that the ADD query runs fine. We add several event and check each time the state of the table with STATS_C.

#### 3.1.2 TestAddStatC and TestStatC

These tests check the comportment of the STATS_C query, especially with no values, one value, and after several ADD.

#### 3.1.3 TestStatD7

This test checks the comportment of the STATS_D7 query, especially with no values, one value, and after several ADD.

#### 3.1.4 TestJamDH

This test checks the comportment of the JAM_DH query, especially with no values, full jam values and multiple values.

#### 3.1.5 TestOpt

This test checks the comportment of the OPT query. We made this request with 0 sensors, sensors with no value, 1 and 2 sensors.

### 3.2 machine generated tests

#### 3.2.1 Test2

The code generates entries (all are green) for 1500 sensors and a restricted time interval. Calls JAM_DH query to test input values. Used to check that the program runs.

#### 3.2.2 Test3

The code generates entries for 50 sensors but in bigger time intervals and multiple traffics. Calls the 3 different queries with several parameters. Used o check that the program computes datas properly.

#### 3.2.3 Test4

The code generates some entries to test basically if the last query works with some missing values. Fast to check.

#### 3.2.4 Test5

Our biggest test. 20M events on 1500 sensors. To compare with outputs from other groups. Sensor numbers between 0 and 1G