# Project 3: Clustering a Sensor network using NIC addresses (35 points)

CPSC 335 - Algorithm Engineering
Fall 2021
Instructor: Doina Bein (dbein@fullerton.edu)

## Abstract

In this project you will design and implement one algorithm related to hash tables that can aid various distributed algorithms to group sensors. Your project is about reading a fairly small number of distinct Network Interface Controllers (NICs), each being a 6-digit number in hexadecimal) and deciding which digit among the six gives the best balanced distribution of the NICs. You will implement it using C++, compile, test it, and submit the files. No algorithm analysis is needed for this project and no PDF report is to be submitted. The next step is to do research on the Internet about the usefulness of having a .gitignore file and create the file .gitignore in your directory. Then upload everything on GitHub.

## The Hypothesis

This experiment will test the following hypotheses:

1. Different hash functions produce hash chains of various lengths when applied to the same dataset.
2. Hash tables offer a relatively fast way of storing and retrieving data.

## The Problem

Communication is paramount on sensor networks and it relies on grouping sensors into clusters, and providing code to be executed distributively at the sensors to maximize coverage and extend lifetime of the network, thus minimizing energy consumption for transmission. There are several types of sensors used to collect data from the environment; a simpiified classification is available at https://www.variohm.com/news-media/technical-blog-archive/types-of-sensors. A much detailed list is available here https://www.thomasnet.com/articles/instruments-controls/sensors/.

We are given a sensor network with a fairly small number of MAC addresses, one for each sensor, and we would like to group the sensors to form several clusters of sensors based on their MAC address. A MAC address (Media Access Control address) is a unique identifier assigned to a network interface controller (NIC) for on a network. A MAC Address is often referred to as a hardware address. The MAC addresses are unique and they are used for communication between components or computers. MAC addresses take the form of alphanumeric numbers in hexadecimal format arranged in groups of 2 (usually) separated by a colon. Renaming MAC addresses to minimize bits used for communication

(https://dl.acm.org/doi/abs/10.1145/501449.501463 ) is one technique used to work around MAC addresses that are unique and drawn from an extremely large number of possible values.


For example, in Fig.1 00::26:C7:CF:94:98 is the MAC address for a component called Wireless-N 1000 which is a wireless adapter (https://ark.intel.com/content/www/us/en/ark/products/59480/intel-centrino-wireless-n-1000-single-band.html).

*Fig.1 MAC-adreso - Vikipedio, Creator: Raimond Spekking, from https://eo.wikipedia.org/wiki/MAC-adreso This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.*

For our problem we consider a MAC address on 48 bits, or 6 hexadecimals, called MAC-48 and which it now refers to as EUI-48 identifiers (https://en.wikipedia.org/wiki/MAC_address).

The first 6-digits (say 00:26:C7 or 0026C7) of the MAC Address identifies the manufacturer, called as Organizational Unique Identifier (OUI); for the complement in Fig. 1 it would be Intel. The next 6 digits of the MAC address are Network Interface Controller (NIC) specific and are unique to each product (https://www.geeksforgeeks.org/introduction-of-mac-address-in-computer-network/).

For our problem, the following scenario motivates solving it. We are given a sensor network with a large number of MAC addresses, one for each sensor, and we would like to group the sensors to form a cluster of sensors based on their MAC address. We assume that the sensors are from the same manufacturer, so we consider only the NIC numbers are of importance for this problem. **We assume that the MAC addresses are 6-digit numbers instead of 6-alphanumeric characters from the set {0..9, A..F}.**

For example, let's assume that we have the following NICs:
123456, 234567, 345678, 456789, 567890, 678901, 789012, 890123, 901234, 543210, 654321, 765432, 876543, 987654, 109876, 210987, 321098, 432109
If we choose storing the NIC numbers based on the first digit that we have:
Cluster 0: no items
Cluster 1: 123456, 109876
Cluster 2: 234567, 210987
Cluster 3: 345678, 321098
Cluster 4: 456789, 432109
Cluster 5: 567890
Cluster 6: 678901
Cluster 7: 789012
Cluster 8: 890123
Cluster 9: 901234, 987654

The difference in the clusters' load is 2, for example between the load of Cluster 1 and Cluster 0.

Cluster 0: 901234, 109876
Cluster 1: 210987
Cluster 2: 123456, 321098
Cluster 3: 234567, 432109
Cluster 4: 345678, 543210
Cluster 5: 456789, 654321
Cluster 6: 567890, 765432
Cluster 7: 678901, 876543
Cluster 8: 789012, 987654
Cluster 9: 890123

The difference in the clusters' load is 1 so clearly this is a more balanced organization of the NICs than the one based on the first digit.

One needs to compute six hash tables and select the first one which is the most balanced. For example, if hash table 1, 3 and 5 are the most balanced, then hash table 1 will be returned.

The first hash table organizes NICs based on the first digit, the second hash table has the hash function that organizes the NICs based on the second digit, etc. tha last hash table has the hash function that organizes the NICs based on the sixth (the last) digit.

# Implementation

You are provided with the following files.

1. `README.md` contains a brief description of the project,and a place to write the names and CSUF email addresses of the group members. You need to modify this file to identify your group members. This information will be used so that we can enter your grades into Canvas. To be completed.
2. `LICENSE` contains a description of the MIT license.
3. `main.cpp` contains tests to check on the correctness of the function members. This is provided for you to use to test your software as you are writing it. You may change this file to add helpful functions for your own testing. I will test your project with a different but similar file.
4. `SensorCluster.cpp` and `SensorCluster.hpp` contain the skeleton of the classes and function member
5. `in1.txt` contains 18 types of NICs numbers
6. `in2.txt` contains 37 types of NIC numbers

The code to decide which digit leads to the most balanced hashtable is to be implemented in class `ItemCollection`. Since you will be comparing six hashtables (which differ only in their hash function), class `ItemCollection` only has six hashtables as its member variables. The other member functions are:

1. `addItem()`: Given information about NIC number, create an Item object and insert into each of the six hash tables. Note that each hash table has the product number as the key and an Item object as the value. To be completed.
2. `removeItem()`: Given product number, remove the corresponding NIC from each of the six hash tables. To be completed.
3. `bestHashing()`: The logic to calculate the balance for each of the six hash tables, and then identifying the hash table with the best balance should go into this method. Here, balance is defined as the difference between the sizes of the largest bucket and smallest bucket. Only check the first 10 buckets! (If the lowest balance factor is shared by more than one hash table, then return the first hash table with that lowest balance factor. For example, if both hT3 and hT7 have the lowest balance factor, then return the number 3. If hT2, hT4, and ht6 all share the lowest balance factor, then return the number 2.) Some hints on how to get the number of items in each bucket are included. To be completed.
4. `readTextfile()`: The list of NIC numbers are in a text file. This method calls addItem() for each line. The code to read from the text file is already given.

The six hash tables will differ in only the hash function that they will use. You are to provide code for these hash functions. Each hash function will take a 6-digit number and return either the first, second, etc., sixth (last) digit.

- hashfct1(number): return the first digit of the NIC number. To be completed.
- hashfct2(number): return the second digit of NIC number. To be completed.
- hashfct3(number): return the third digit of the NIC number. To be completed.
- hashfct4(number): return the fourth digit of the NIC number. To be completed.
- hashfct5(number): return the fifth digit of the NIC number. To be completed.
- hashfct6(number): return the fourth digit of the NIC number. To be completed.

# Obtaining and Submitting Code

This document explains how to obtain and submit your work:

GitHub Education Instructions

Here are the invitation links for the projects:

https://classroom.github.com/a/gC1OrNGr

# What to Do

First, add your group member names to `README.md`. Implement all the skeleton functions in the provided header file. Use the test program to check whether your code works. Then create the file .gitignore in your directory. When you go to commit files in git, make sure you are not checking any generated executable/object files into git. Use the *.gitignore* file to accomplish this.

# Grading Rubric

Your grade will consist of two parts: *Form* and *Function*.

*Function* refers to whether your code works properly as defined by the test program. We will use the score reported by the test program as your Function grade.

*Form* refers to the design, organization, and presentation of your code. A grader will read your code and evaluate these aspects of your submission.

The grading rubric is below.

1. Function - 19 points: passes all the tests
2. Form - 16 points
    a. README.md complete: 3 points
    b. Style (whitespace, variable names, comments, etc.): 3 points
    c. Design (where appropriate, uses encapsulation, helper functions, data structures, etc.): 4 points
    d. Craftsmanship (no memory leaks, gross inefficiency, taboo coding practices, etc.): 4 points
    e. File `.gitignore` created and populate it correctly = 2 points

# Deadline

The project deadline is Friday, November 19, 11:59 pm.

You will be graded based on what you have pushed to GitHub as of the deadline. Commits made after the deadline will not be considered. Late submissions will not be accepted.