California State University Fullerton

Professor: Doina Bein

Project 2 Report: Using NIC addresses to Cluster Nodes of a Sensor Network
CPSC 535 -01 (14069)

Submission by:

netclust

Name	Email	Position
NIDHI SHAH	nidhishah989@csu.fullerton.edu	Team member
MERIN JOSEPH	merin.joseph@csu.fullerton.edu	Team member
APEKSHA SHAH	apeksha@csu.fullerton.edu	Team member

Table of Contents:

1. Summary of Project -Using NIC addresses to Cluster Nodes of a Sensor Network

1.1 Problem Statement

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 the problem. Our 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. Based on the lowest cluster load difference, the most balanced hash table is selected. The MAC addresses are considered as a 6-alphanumeric string from the set {0,1,2….9,A,B,C,D,E,F}. Any other alphanumeric character or upper-case characters are treated as an invalid output.

1.2 Input and Output Summary

Input: A text file of a sensor network with the name of sensors along with its respective distinct Network Interface Controllers (NICs), each being a 6-digit hexadecimal number.

Output: The best-balanced distribution of the NICs among the six hash tables (balance is defined as the difference between the sizes of the largest bucket and smallest bucket).

2. Algorithm Description

2.1 Design Overview

Read the input file with the sensor names and NICs. After reading the input file, the NIC values are assigned to the appropriate cluster in each hash table, based on the hash function associated with the NIC number or character. We have six distinct hash tables. The first hash table organizes the NICs based on the first digit, the second hash table organizes them based on the second digit, and so on. Since each NIC is of length six, we have six hash tables. After all the NICs are assigned to all six hash tables, then the difference in the cluster's load is calculated for each hashtable. The cluster load values of each hash table computed are stored in a min-priority queue. The top() value of the min-priority queue has the hashtable with the least cluster load difference and it is printed along with the corresponding hash table since a hash table with the minimum cluster load value is the most balanced distribution or organization of NICs.

Brief description of the functions in the file SensorCluster.cpp:

- **hashfct1, hashfct2, hashfct3(), hashfct4(), hashfct5(), hashfct6():** The six hash functions assign each NIC to the respective cluster in the hash table according to its corresponding hash function. If the value in the NIC is a digit value then the cluster number is chosen accordingly (cluster value =

digit%10). But if the value is a character, then the corresponding hex value is assigned as cluster value if the character is from the set {A,B,C,D,E,F}. The occurrence of any other character will be outputted as “Invalid character”.

- **addItem():** Given information about a NIC, create an Item object and insert into each of the six hash tables with consideration of hash function related to NIC.
- **removeItem():** Given a NIC, it removes the Item corresponding to the NIC from each of the six hash tables.
- **bestHashing():** The balance or cluster load difference for each of the six hash tables is calculated and stored in a min-priority queue, and then the hash table with the best balance (lowest load difference) is given as the output. This would be the ‘top’ value in the min-priority queue. Only the first 16 buckets or clusters are checked.

  If the lowest balance factor is shared by more than one hash table, then the function returns the first hash table with that lowest balance factor.
- **readTextfile():** The list of NICs and the product names are in a text file. This method calls addItem() for each line.

2.2 Programming Language

This Algorithm is written with the consideration that the programming language is c++ for project code.

2.3 Algorithm Complexity

The time complexity is O(n) as insertion, deletion, and creation in the hash table is O(1).

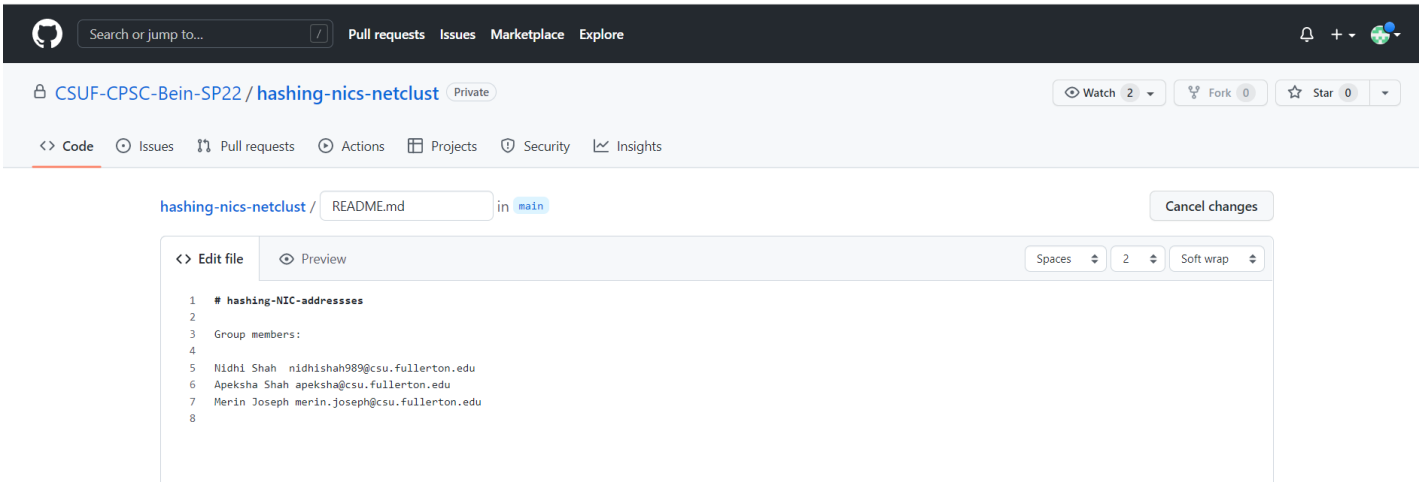
3. Description of how to run the code

The project has two .cpp files - SensorCluster.cpp and main.cpp. Run the following command in the terminal “g++ -std=c++17 -Wall SensorCluster.cpp main.cpp -o sensor_test” and then “./sensor_test”. The program is then tested using the test file and criterions provided in the skeleton code.

Our program successfully passed all the test cases mentioned in the program requirements. We also passed an invalid character for the NIC and it threw the exception with the error message “Invalid character” because the characters in the NIC string had to be from the set {0,1,2,….,9,A,B,…,F}.

4. Screenshots

4.1 Group member screenshot



4.2 Example Screenshot

(a). The output screenshot

ON Given main,cpp skeleton code. (Rubric test). Test done on in1.txt and in2.txt files.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\nidhi\OneDrive\Desktop\CPSC535-ALGORITHM\latest_new\new> g++ -std=c++17 -Wall SensorCluster.cpp main.cpp -o sensor_test
PS C:\Users\nidhi\OneDrive\Desktop\CPSC535-ALGORITHM\latest_new\new> ./sensor_test
Successfully opened file in1.txt
Successfully opened file in2.txt
Successfully opened file in2.txt
hash function 1 on item 123456 returns 1: passed, score 1/1
hash function 2 on item 123456 returns 2: passed, score 1/1
hash function 3 on item 123456 returns 3: passed, score 1/1
hash function 4 on item 123456 returns 4: passed, score 1/1
hash function 5 on item 123456 returns 5: passed, score 1/1
hash function 6 on item 123456 returns 6: passed, score 1/1
hash function 1 on item 6789AB returns 6: passed, score 1/1
hash function 2 on item 6789AB returns 7: passed, score 1/1
hash function 3 on item 6789AB returns 8: passed, score 1/1
hash function 4 on item 6789AB returns 9: passed, score 1/1
hash function 5 on item 6789AB returns 10: passed, score 1/1
hash function 6 on item 6789AB returns 11: passed, score 1/1
New network. Size is 2 after adding two NICs: Velocity sensor 123456 and Particle sensor 234567: passed, score 1/1
New network. Size is 30 after reading in1.txt: passed, score 1/1
BestHashing() for in1.txt returns 2: passed, score 1/1
New network. Size is 37 after reading in2.txt: passed, score 1/1
BestHashing() for in2.txt returns 2: passed, score 1/1
New network then read in2.txt. Then remove two NICs: 110987 and 210FED. Size becomes 35: passed, score 1/1
hash function 1: passed, score 1/1
TOTAL SCORE = 19 / 19

PS C:\Users\nidhi\OneDrive\Desktop\CPSC535-ALGORITHM\latest_new\new>
```

(b). The output for the wrong character gives the “invalid input character” error

```
22    // test hash functions
23    rubric.criterion("hash function 1 on item 123456 returns 1", 1,
24            [&]() {
25              TEST_EQUAL( "hashfct1(G23456)", 1, hashfct1("G23456") );
26            });
27
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\nidhi\OneDrive\Desktop\CPSC535-ALGORITHM\latest_new\new> g++ -std=c++17 -Wall SensorCluster.cpp main.cpp -o sensor_test
PS C:\Users\nidhi\OneDrive\Desktop\CPSC535-ALGORITHM\latest_new\new> ./sensor_test
Successfully opened file in1.txt
Successfully opened file in2.txt
Successfully opened file in2.txt
hash function 1 on item 123456 returns 1: Invalid character
terminate called after throwing an instance of 'std::length_error'
  what():  Invalid character
PS C:\Users\nidhi\OneDrive\Desktop\CPSC535-ALGORITHM\latest_new\new>
```