

CPSC 535 - ADVANCED ALGORITHMS PROJECT 2

Using NIC addresses to Cluster Nodes of a Sensor Network
REPORT

Team Members

Shivangi Shakya

Nicholas Bidler

Saoni Mustafi

Prof. Doina Bein

California State University Fullerton

800 N State College Blvd, Fullerton, CA 92831

Summary

This report contains information about the work done to solve the “Using NIC addresses to Cluster Nodes of a Sensor Network” problem. The inputs are text inputs or a text file with some number of 6-digit hexadecimal numbers, representing a number of NIC addresses in a sensor network. When the hash table for the input values at a given digit location is given an input such that there is a collision of one or more items, ‘chaining’ is used to handle the collision. The objective here is to find an algorithm that minimizes the difference in distance between these chains.

Essentially, the objective is to find an algorithm that functions similarly to a Bucket Sort, such that when inputs are evenly distributed in a hash table with chaining, any non-insertion operations have an average $O(n)$ time class.

The first thing the program does is to initialize one ItemCollection object for each network: each ItemCollection will hold all the HashTables and member functions for its assigned network.

Demonstration:

Example 1: demonstrating that the program output for hashed inputs and outputs is the same as the sample outputs:

Shown in the screenshots below, the example calls a function that returns the decimal value of the hex digit at the appropriate location, i.e., hashfct4() returns the decimal value of the hex digit at position 4.

```
Enter number of items to find what the six hash function returns: 2
Enter the 6-digit NIC number - 123456
Enter the 6-digit NIC number - 6789AB
hash function 1 on item 123456 returns 1
hash function 2 on item 123456 returns 2
hash function 3 on item 123456 returns 3
hash function 4 on item 123456 returns 4
hash function 5 on item 123456 returns 5
hash function 6 on item 123456 returns 6
hash function 1 on item 6789AB returns 6
hash function 2 on item 6789AB returns 7
hash function 3 on item 6789AB returns 8
hash function 4 on item 6789AB returns 9
hash function 5 on item 6789AB returns 10
hash function 6 on item 6789AB returns 11
```

```

g++ -std=c++17 -Wall SensorCluster.cpp main.cpp -o sensor_test
./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

```

Example 2: demonstrating that the program evaluation to find the most even distribution is the same as the sample cases:

After this, the program allows the user to manually enter a network: the number of NICs to be entered, followed by the 6-digit NICs.

```

Network 1
Enter number of items to add to network 1 : 2
Enter the full name of the sensor - 123456
Enter the full name of the sensor - 6789AB
Size is 2 after adding 123456 and 6789AB
BestHashing() for Network 1 ['123456', '6789AB'] returns 1

```

This creates a "network" and runs the bestHashing function, which finds the smallest and largest entry on each hash table, takes the difference of each, and returns the hash table with the smallest difference.

```

Network 2
Successfully opened file in1.txt
New network. Size is 30 after reading in1.txt.
BestHashing() for in1.txt returns 2

Network 3
Successfully opened file in2.txt
New network. Size is 37 after reading in2.txt.
BestHashing() for in2.txt returns 2

```

```
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
```

Example 3: showing the network can be interacted with in a way that changes the Best Hashing calculation.

```
Network 4
Enter number of items to remove from network 3 and create a new network: 2
Enter the 6-digit NIC number to remove - 110987
Enter the 6-digit NIC number to remove - 210FED
Successfully opened file in2.txt
New network then read in2.txt
Then remove NICs : 110987 and 210FED. Size becomes 35
BestHashing() after removing 110987 and 210FED returns 3
```

Before deletion (result of 2):

```
|123456 | 234567 | 345678 | 456789 | 56789A | 6789AB | 789ABC
|89ABCD | 9ABCDE | ABCDEF | BCDEF0 | CDEF01 | DEF012 | EF0123
|F01234 | 543210 | 43210F | 3210FE | 210FED | 10FEDC | FEDCBA
|EDCBA9 | DCBA98 | CBA987 | BA9876 | A98765 | 987654 | 876543
|765432 | 654321 | 776543 | 887654 | 998765 | 110987 | 221098
|332109 | 443210
```

After deletion:

```
|123456 | 234567 | 345678 | 456789 | 56789A | 6789AB | 789ABC
|89ABCD | 9ABCDE | ABCDEF | BCDEF0 | CDEF01 | DEF012 | EF0123
|F01234 | 543210 | 43210F | 3210FE |          | 10FEDC | FEDCBA
|EDCBA9 | DCBA98 | CBA987 | BA9876 | A98765 | 987654 | 876543
|765432 | 654321 | 776543 | 887654 | 998765 |          | 221098
|332109 | 443210
```

Digit count at positions:

Count of:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Pos. 1	0	2	2	3	4	2	2	3	3	3	2	2	2	2	2	2
Pos. 2	2	0	3	3	3	2	2	3	3	3	2	2	2	2	2	1
Pos. 3	1	3	2	3	2	2	3	2	3	2	2	2	2	2	1	2
Pos. 4	2	3	3	1	2	3	3	3	2	2	2	2	2	1	2	1
Pos. 5	3	3	2	2	1	3	3	2	2	3	2	2	1	2	1	2
Pos. 6	2	2	2	2	3	2	2	2	3	3	2	1	2	1	2	2

As shown by the above table, position 1 and 2 have a maximum difference of 3, while 3 through 6 have a difference of 2 - as tie-breaker, the first position encountered is the one that is used for the hash table. In this case, the result is position 3, which matches the program's output.

Pseudocode:

create an *Item* class

initialize the class with *item* dictionary

define a class function *itemKeys* and pass the *data* as an argument:

identify the *last six digits* as the *key* and the rest part of the data as the *value*

insert into the *item* dictionary

initialize a *key* list and ensure there are no items in this list

insert the *key* in this list

return the *key* of the *item* dictionary

create an *ItemCollection* class

initialize all 6 HashTables as list of lists required for the four networks

define a function *addItem* and pass the *data* as the argument:

create an object of the *Item* class

key = call *itemKeys()* function to identify the key and value for this data

k1 = call *hashfct1()* and pass the *key* of the argument

append this key and value in the appropriate position *k1* of the HashTable1

k2 = call *hashfct2()* and pass the *key* of the argument

append this key and value in the appropriate position *k2* of the HashTable2

k3 = call *hashfct3()* and pass the *key* of the argument

append this key and value in the appropriate position *k3* of the HashTable3

k4 = call *hashfct4()* and pass the *key* of the argument

append this key and value in the appropriate position *k4* of the HashTable4

k5 = call *hashfct5()* and pass the *key* of the argument

append this key and value in the appropriate position *k5* of the HashTable5

k6 = call *hashfct6()* and pass the *key* of the argument

append this key and value in the appropriate position *k6* of the HashTable6

define a *hashfct1* and pass an argument *nic*:

k = first digit of the hexadecimal *nic* and convert to int datatype

return *k*

define a *hashfct2* and pass an argument *nic*:

k = second digit of the hexadecimal *nic* and convert to int datatype

return *k*

define a *hashfct3* and pass an argument *nic*:

k = third digit of the hexadecimal *nic* and convert to int datatype

return *k*

define a *hashfct4* and pass an argument *nic*:

k = fourth digit of the hexadecimal *nic* and convert to int datatype

return *k*

define a *hashfct5* and pass an argument *nic*:

k = fifth digit of the hexadecimal *nic* and convert to int datatype
return k

define a *hashfct6* and pass an argument *nic*:

k = sixth digit of the hexadecimal *nic* and convert to int datatype
return k

define a *removeItem()* that takes a list of items to remove *remlist[]* as argument:

for every *nic* in *remlist*:

$key1$ = call *hashfct1(nic)* to compute the hashvalue

for every item in the list *HashTable1[key1]*:

if *nic* matches with the key to be deleted:

del the key and value and break from the loop

remove the empty dictionary formed by the delete operation

$key2$ = call *hashfct2(nic)* to compute the hashvalue

for every item in the list *HashTable2[key2]*:

if *nic* matches with the key to be deleted:

del the key and value and break from the loop

remove the empty dictionary formed by the delete operation

$key3$ = call *hashfct3(nic)* to compute the hashvalue

for every item in the list *HashTable3[key3]*:

if *nic* matches with the key to be deleted:

del the key and value and break from the loop

remove the empty dictionary formed by the delete operation

$key4$ = call *hashfct4(nic)* to compute the hashvalue

for every item in the list *HashTable4[key4]*:

if *nic* matches with the key to be deleted:

del the key and value and break from the loop

remove the empty dictionary formed by the delete operation

$key5$ = call *hashfct5(nic)* to compute the hashvalue

for every item in the list *HashTable5[key5]*:

if *nic* matches with the key to be deleted:

del the key and value and break from the loop

remove the empty dictionary formed by the delete operation

$key6$ = call *hashfct6(nic)* to compute the hashvalue

for every item in the list *HashTable6[key6]*:

if *nic* matches with the key to be deleted:

del the key and value and break from the loop

remove the empty dictionary formed by the delete operation

define function *bestHashing()*:

```
    initialize an empty dict{} and set j = 1
    Run a for loop for all tables in HashTable1, HashTable2, HashTable3,
    HashTable4, HashTable5, HashTable6:
        set maxcount = 0 and mincount = sys.maxsize
        for items in the list of the table:
            count = length of that list
            if the count is less than the mincount, update mincount
            if count is greater than the maxcount, update maxcount.
            set diff as difference between maxcount & mincount
            insert the diff as value and j as key in the dict{}
            increment j by 1
    determine the key with minimum value in the dict{}
    return the key
```

define function *readText* and pass the *filename* as an argument:

```
    initialize list = [] and list1 = []
    if the filename is "in1.txt":
        clear list
        fl = open the file in readmode
        print "Successfully opened in1.txt"
        datainput = read the content of the file
        datainput = update the list of data read delimited as new line
        close the file
        for every value in the datainput:
            append it to the list[]
        for every l in list:
            call addItem(l) to add 'l' to form Network 2
        size = 0
        for i in the range of length of HashTable1
            size = 0 + length of HashTable1[i]
        print size of network and the file name

    if the filename is "in2.txt":
        clear list1
        fl1 = open the file in read mode
        print "Successfully opened in2.txt"
        datainput1 = read the content of the file
        datainput1 = update the list of data read delimited as new line
        close the file
        for every value in the datainput1:
            append it to the list1[]
        for every l in list1:
            call addItem(l) to add items to the form for Network 3
        size = 0
        for i in the range of length of HashTable1
            size = 0 + length of HashTable1[i]
        print size of Network 3 and the file name
    return filename and size
```

inside the *main()* function

create four objects of *ItemCollection* class for the four networks – *item1* for Network 1, *item2* for Network 2, *item3* for Network 3 and *item4* for Network 4

initialize *itemsfind* = []

num1 = The number of items to find what the six *hashfct()* methods return

enter '*num1*' NICs to find the returned hash value and append these to *itemsfind*

for every *nic* in *itemsfind*:

k1 = call *hashfct1()*

print the hash value returned

k2 = call *hashfct2()*

print the hash value returned

k3 = call *hashfct3()*

print the hash value returned

k4 = call *hashfct4()*

print the hash value returned

k5 = call *hashfct5()*

print the hash value returned

k6 = call *hashfct6()*

print the hash value returned

print 'Network 1'

initialize *itemsadd* = []

set *num2* = The number of records to add to form Network 1

Enter *num2* names of sensors to add and append to *itemsadd*[]

for every *item* in *itemsadd*[]:

Call the *addItem()* to add the items individually to the network

print the number of elements in the network

Call the *bestHashing()* method on Network 1

Print the table which is the most balanced cluster for Network 1

print 'Network 2'

call the *readText()* through *item2* object to read the sensors in file "*in1.txt*" and form

Network 2

call the *bestHashing()* method on Network 2

print the table which is the most balanced cluster for Network 2

print 'Network 3'

call the *readText()* through *item3* object to read the sensors in file "*in2.txt*" and form

Network 3

call the *bestHashing()* method on Network 3

print the table which is the most balanced cluster for Network 3

print 'Network 4'

call *readText()* through *item4* to read "*in2.txt*"

initialize *itemsremove* = []

set *num4* = The number of NICs user wants to remove from Network 3

enter '*num4*' number of NICs to remove and append them to *itemsremove*[]

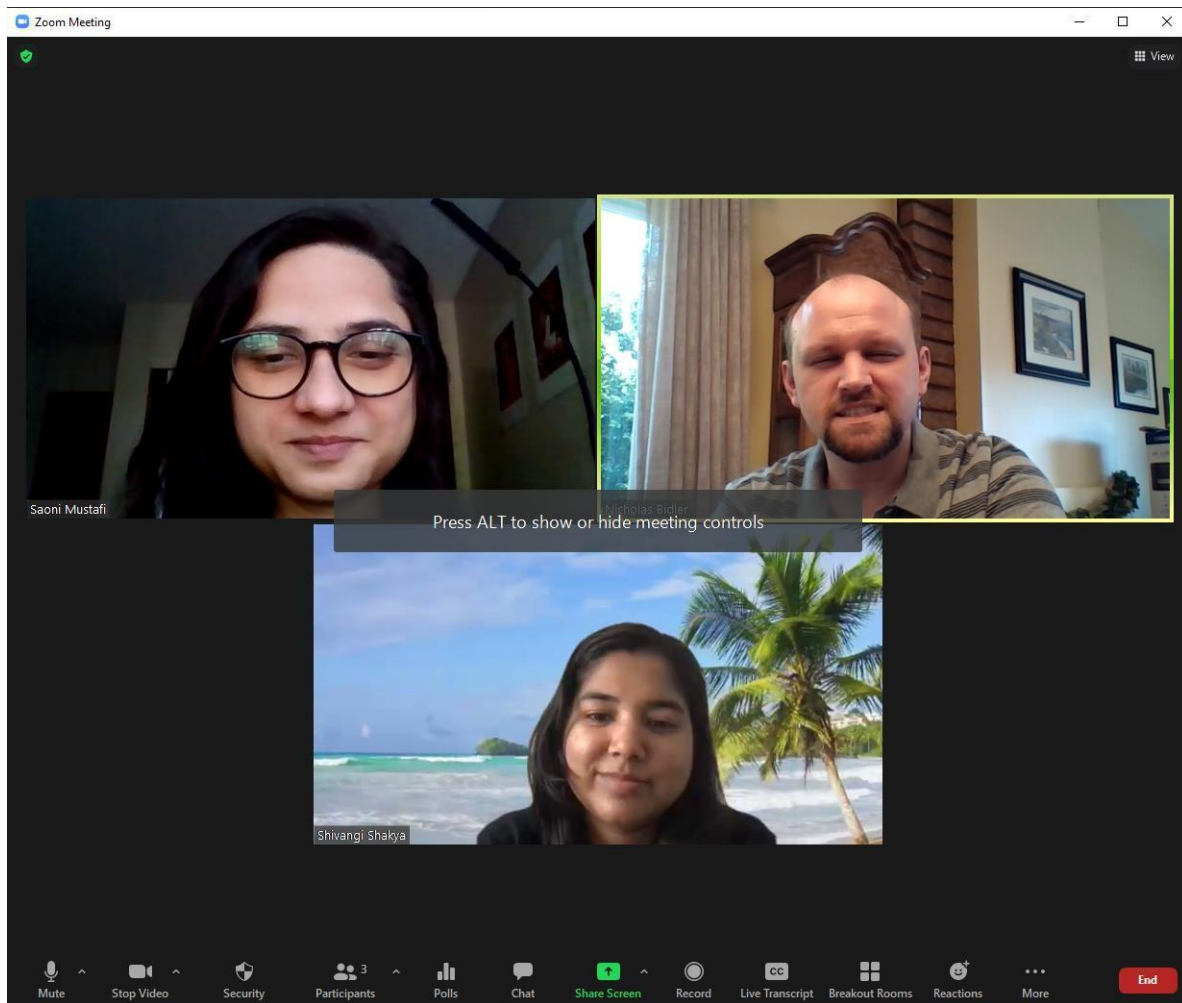
call the *removeItem()* through *item4* and pass *itemsremove[]* in the argument
call *bestHashing()* method on Network 4 and print the most balanced table

Instructions to run the Code:

The code has been implemented in Python. The name of the file is '*HashingNics.py*'. The two input files are located in the same directory as "*in1.txt*" and "*in2.txt*".

1. Save the HashingNics.py, in1.txt, and in2.txt in your desired location.
2. In the terminal, change the directory to the folder where you have saved these files using the command *cd <pathname>*.
3. Type '*python3 HashingNics.py*'.
4. Enter the number of items you want to test the hash function on for correctness.
 - a. Input a corresponding number of 6-digit hexadecimal numbers
 - b. The output should display each digit's value at the corresponding location in decimal
5. Enter the number of items to add to a test network
 - a. Input a corresponding number of 6-digit hexadecimal numbers
 - b. The output should display which digit produces the smallest variance in hash table chain length
6. The program will at this point output its evaluations for the .txt files, which should both be that the second digit is preferred
7. Enter the number of items to remove from the input set from in2.txt
 - a. Be sure it is a valid value before submitting, but similar to 4a and 5a, input the appropriate amount of 6-digit hex numbers

Team Screenshot



Output Screenshot (Part 1):

A screenshot of a Visual Studio Code editor window with the file "HashingNics.py" open. The terminal window shows the output of running the script. The prompt is "PS C:\Users\dell\OneDrive\Documents\CSUF_Courses\Captures\535_Advance_Algorithm_Doina_Bein\Projects\Project_2\hashing-nics-jokename>". The command entered is "python3 HashingNics.py". The output shows the results of a hash function test and the creation of three networks.

```
PS C:\Users\dell\OneDrive\Documents\CSUF_Courses\Captures\535_Advance_Algorithm_Doina_Bein\Projects\Project_2\hashing-nics-jokename> python3 HashingNics.py
Enter number of items to find what the six hash function returns: 2
Enter the 6-digit NIC number - 123456
Enter the 6-digit NIC number - 6789AB
hash function 1 on item 123456 returns 1
hash function 2 on item 123456 returns 2
hash function 3 on item 123456 returns 3
hash function 4 on item 123456 returns 4
hash function 5 on item 123456 returns 5
hash function 6 on item 123456 returns 6
hash function 1 on item 6789AB returns 6
hash function 2 on item 6789AB returns 7
hash function 3 on item 6789AB returns 8
hash function 4 on item 6789AB returns 9
hash function 5 on item 6789AB returns 10
hash function 6 on item 6789AB returns 11

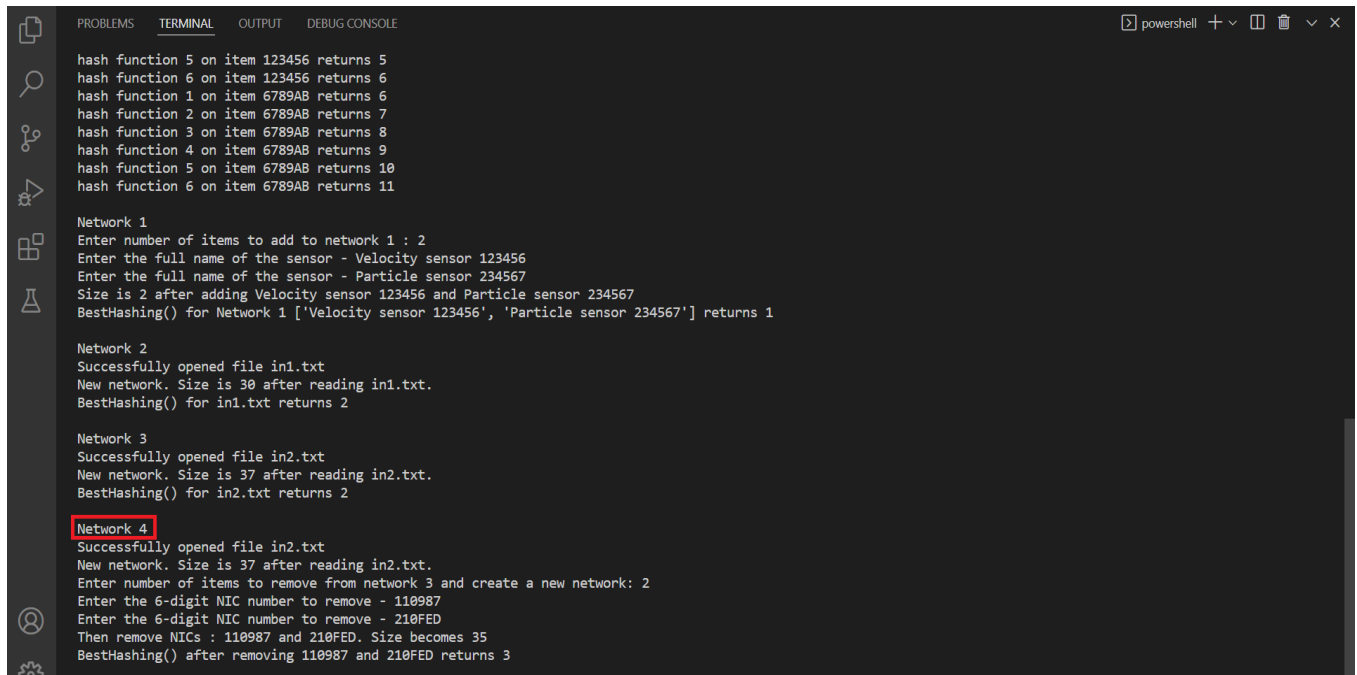
Network 1
Enter number of items to add to network 1 : 2
Enter the full name of the sensor - Velocity sensor 123456
Enter the full name of the sensor - Particle sensor 234567
Size is 2 after adding Velocity sensor 123456 and Particle sensor 234567
BestHashing() for Network 1 ['Velocity sensor 123456', 'Particle sensor 234567'] returns 1

Network 2
Successfully opened file in1.txt
New network. Size is 30 after reading in1.txt.
BestHashing() for in1.txt returns 2

Network 3
Successfully opened file in2.txt
New network. Size is 37 after reading in2.txt.
BestHashing() for in2.txt returns 2

Network 4
Successfully opened file in2.txt
```

Output Screenshot (Part 2):



```
hash function 5 on item 123456 returns 5
hash function 6 on item 123456 returns 6
hash function 1 on item 6789AB returns 6
hash function 2 on item 6789AB returns 7
hash function 3 on item 6789AB returns 8
hash function 4 on item 6789AB returns 9
hash function 5 on item 6789AB returns 10
hash function 6 on item 6789AB returns 11

Network 1
Enter number of items to add to network 1 : 2
Enter the full name of the sensor - Velocity sensor 123456
Enter the full name of the sensor - Particle sensor 234567
Size is 2 after adding Velocity sensor 123456 and Particle sensor 234567
BestHashing() for Network 1 ['Velocity sensor 123456', 'Particle sensor 234567'] returns 1

Network 2
Successfully opened file in1.txt
New network. Size is 30 after reading in1.txt.
BestHashing() for in1.txt returns 2

Network 3
Successfully opened file in2.txt
New network. Size is 37 after reading in2.txt.
BestHashing() for in2.txt returns 2

Network 4
Successfully opened file in2.txt
New network. Size is 37 after reading in2.txt.
Enter number of items to remove from network 3 and create a new network: 2
Enter the 6-digit NIC number to remove - 110987
Enter the 6-digit NIC number to remove - 210FED
Then remove NICs : 110987 and 210FED. Size becomes 35
BestHashing() after removing 110987 and 210FED returns 3
```