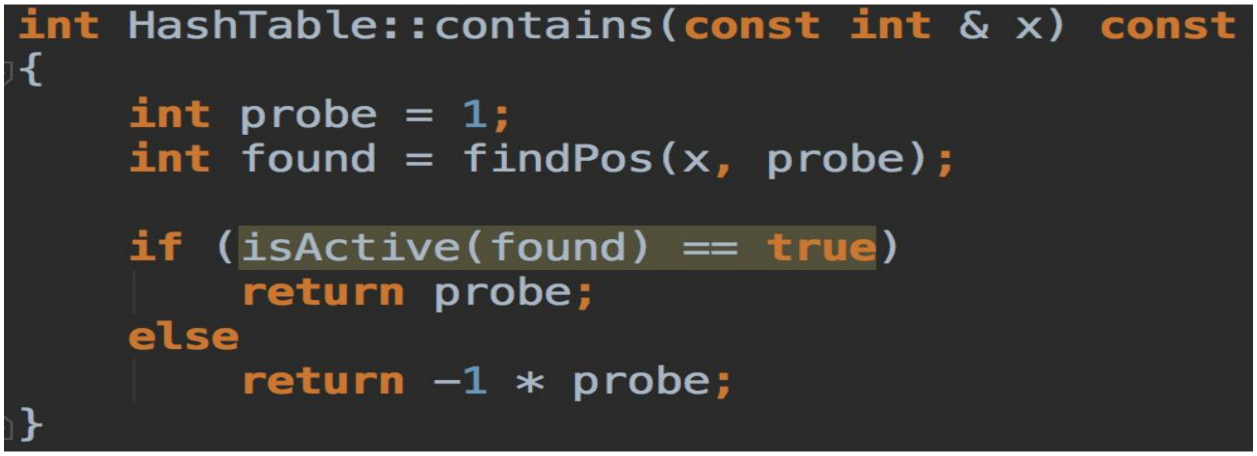
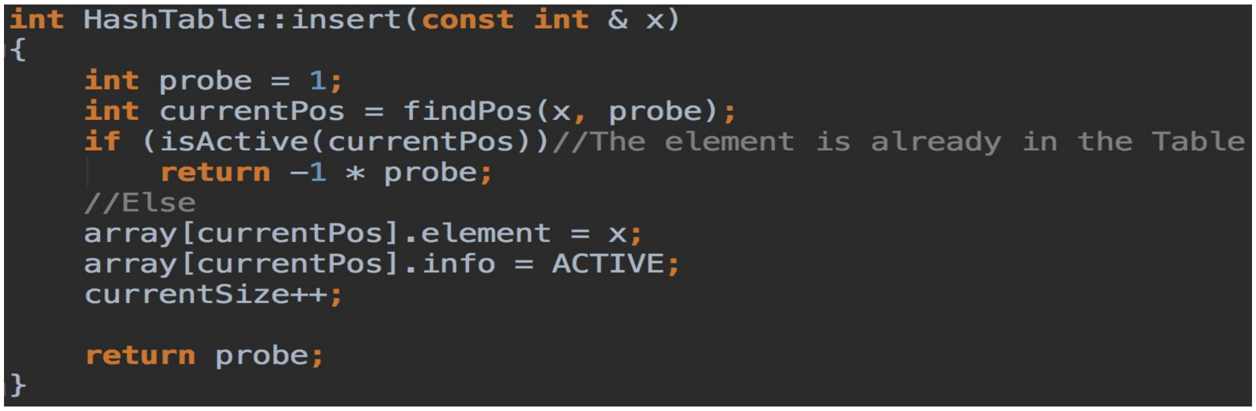
CS300 HW3 REPORT

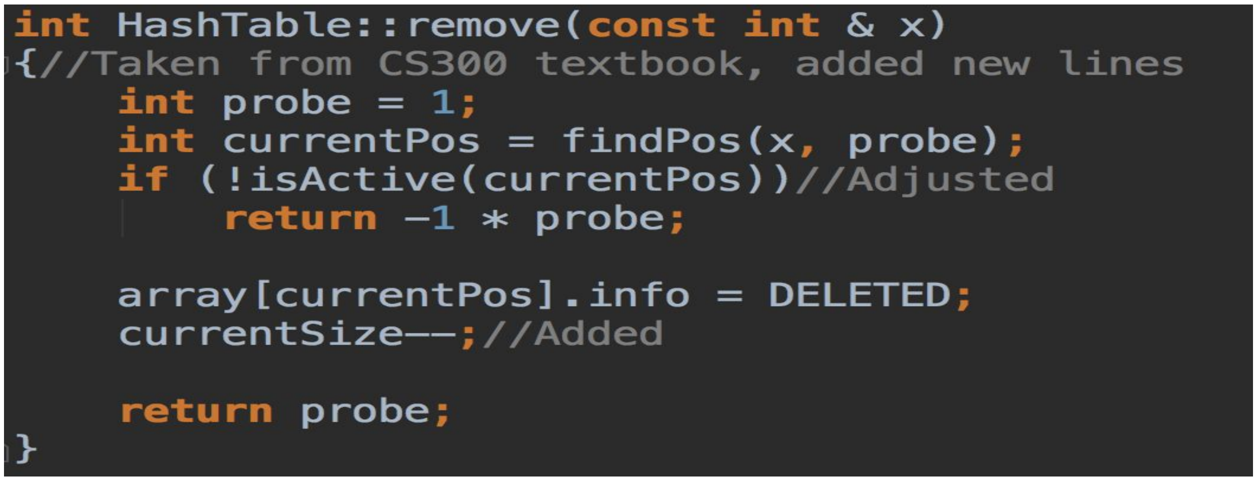
i) Changes that I've done in the code for linear probing



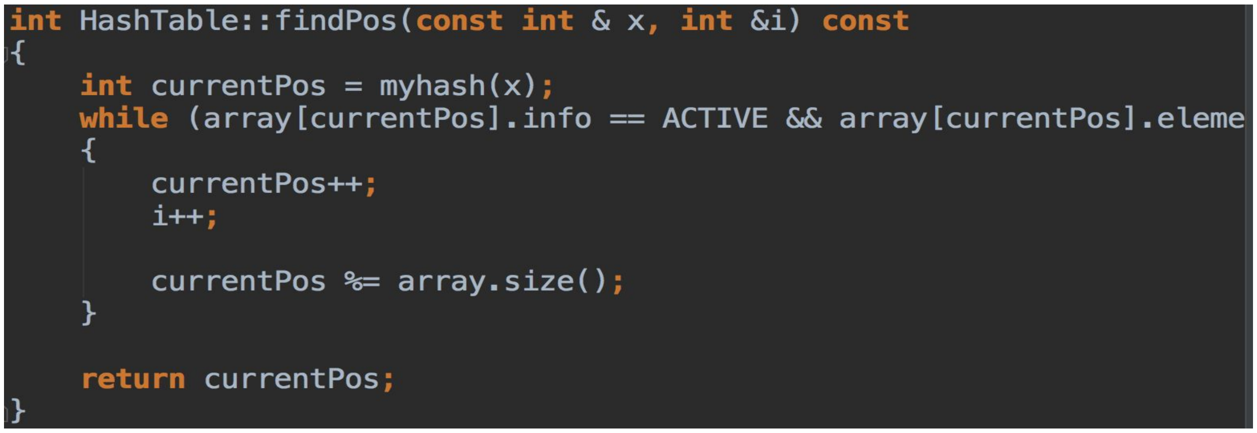
```
int HashTable::contains(const int & x) const
{
    int probe = 1;
    int found = findPos(x, probe);

    if (isActive(found) == true)
        return probe;
    else
        return -1 * probe;
}
```



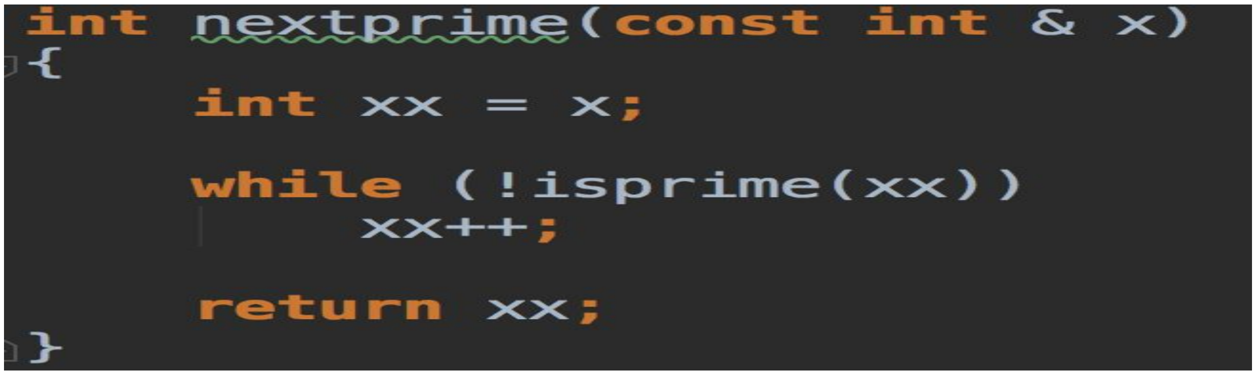
```
int HashTable::insert(const int & x)
{
    int probe = 1;
    int currentPos = findPos(x, probe);
    if (isActive(currentPos))//The element is already in the Table
        return -1 * probe;
    //Else
    array[currentPos].element = x;
    array[currentPos].info = ACTIVE;
    currentSize++;

    return probe;
}
```



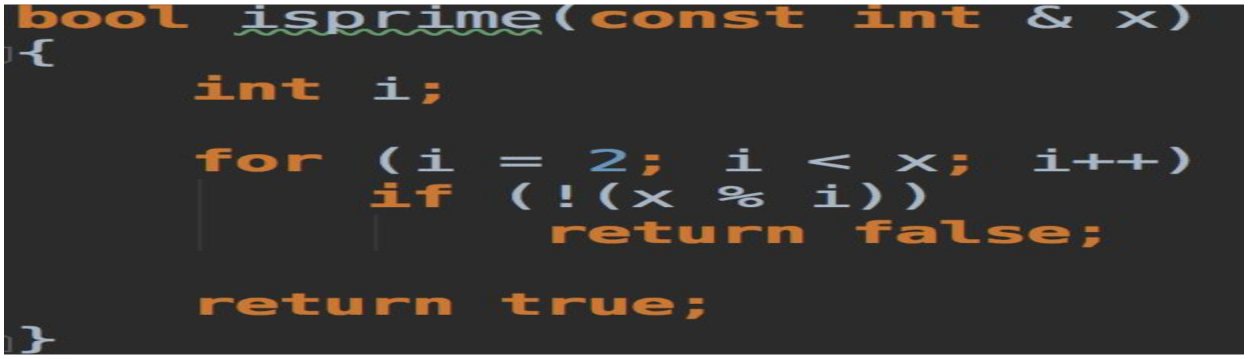
```
int HashTable::remove(const int & x)
{//Taken from CS300 textbook, added new lines
    int probe = 1;
    int currentPos = findPos(x, probe);
    if (!isActive(currentPos))//Adjusted
        return -1 * probe;

    array[currentPos].info = DELETED;
    currentSize--;//Added

    return probe;
}
```



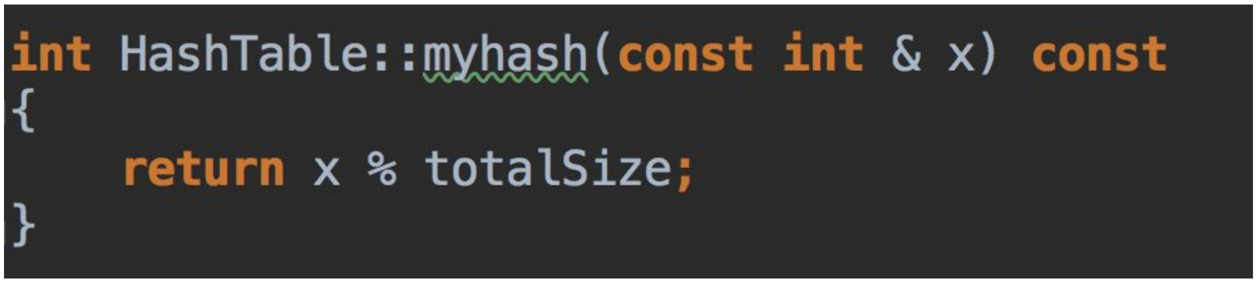
```
int HashTable::findPos(const int & x, int &i) const
{
    int currentPos = myhash(x);
    while (array[currentPos].info == ACTIVE && array[currentPos].eleme
    {
        currentPos++;
        i++;

        currentPos %= array.size();
    }

    return currentPos;
}
```



```
int nextprime(const int & x)
{
    int xx = x;

    while (!isprime(xx))
        xx++;

    return xx;
}
```



```
bool isprime(const int & x)
{
    int i;

    for (i = 2; i < x; i++)
        if (!(x % i))
            return false;

    return true;
}
```

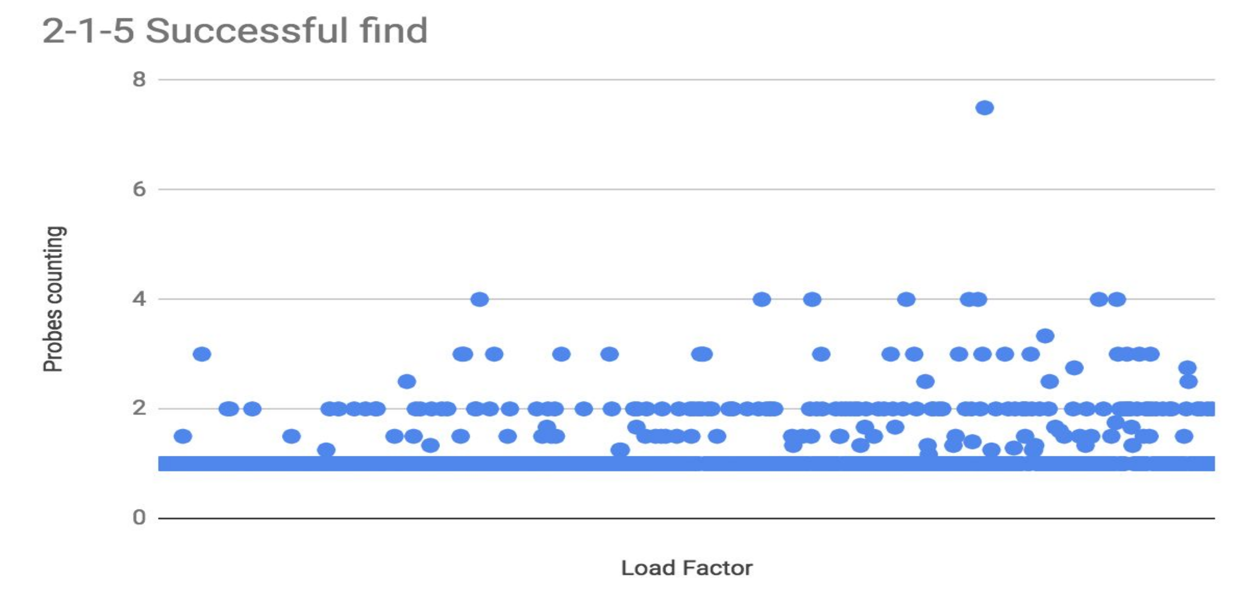


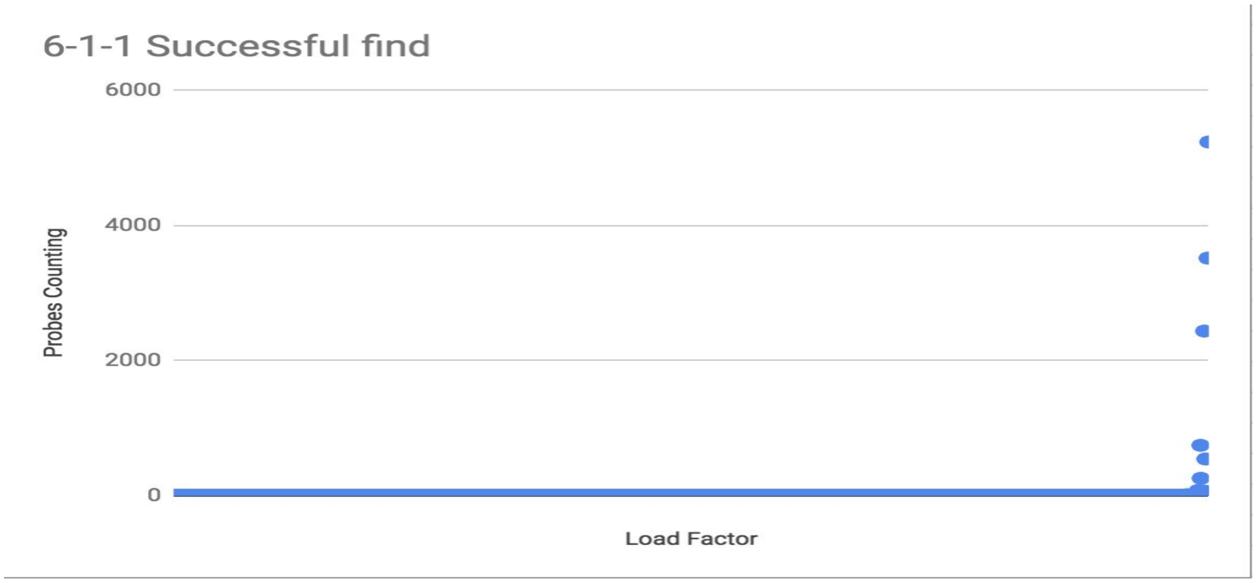
```
int HashTable::myhash(const int & x) const
{
    return x % totalSize;
}
```

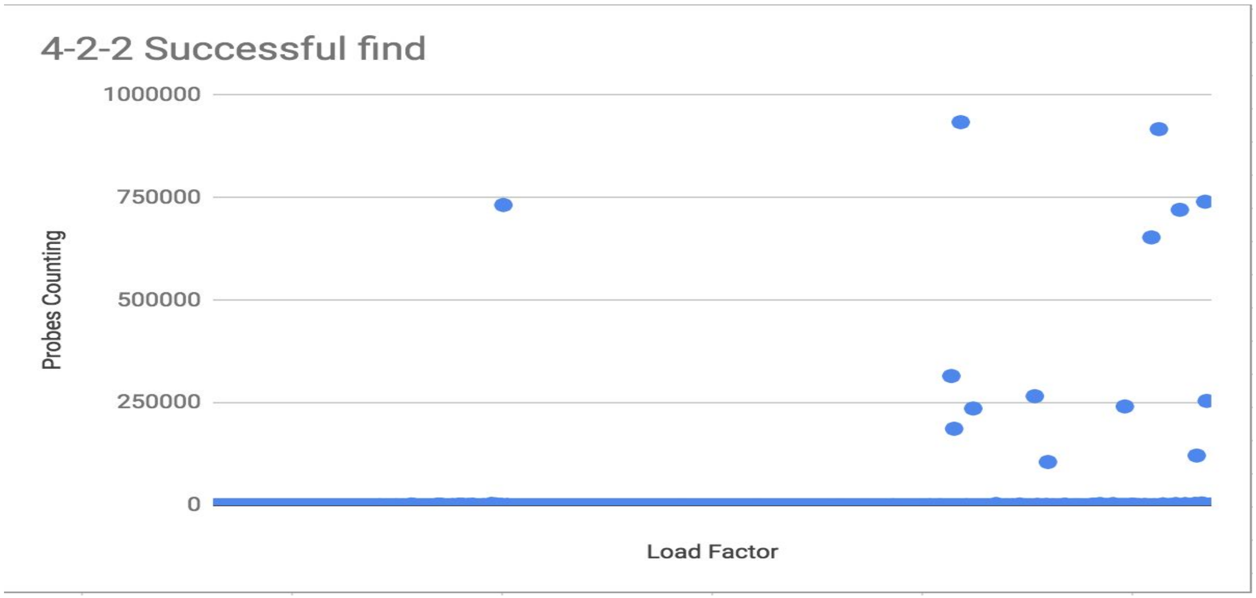
- I’ve changed insert, remove and contains functions.(Return integer instead of boolean value as implemented in the textbook)
- Used 6 2D vector to hold the statistics of the data that obtained from the combinations.
-

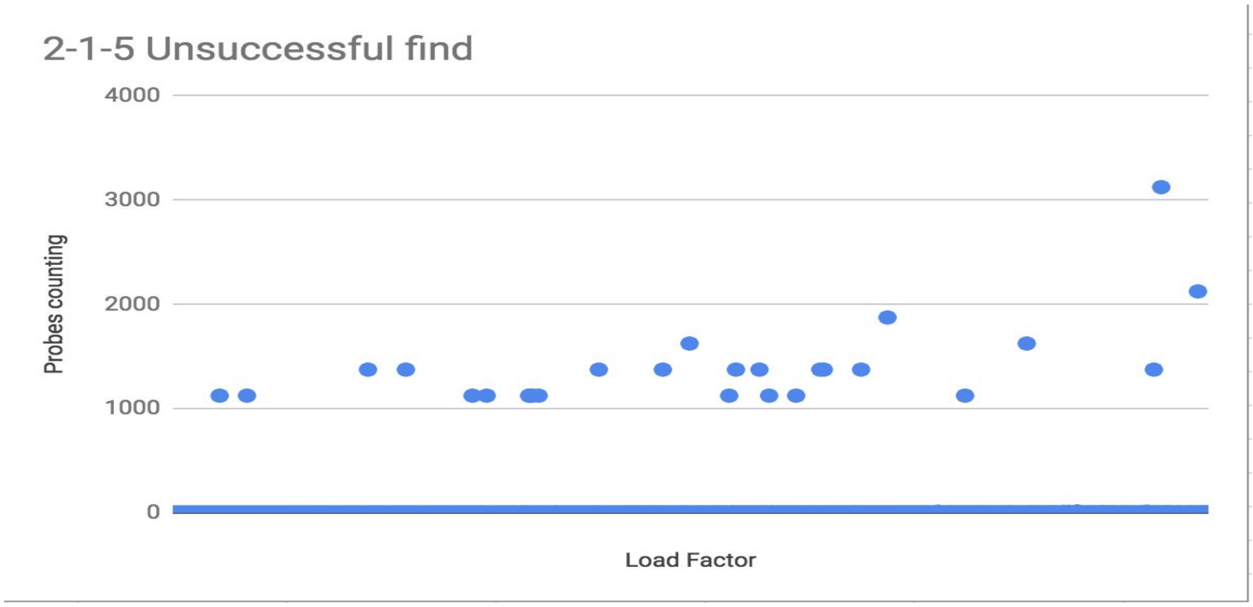
ii) Graphs

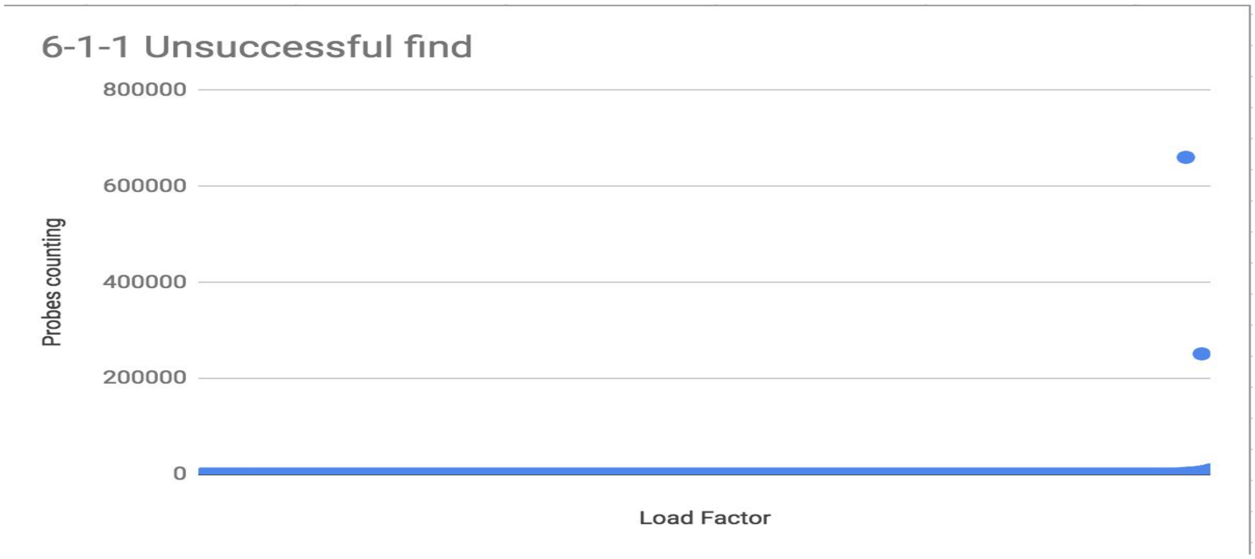
For the y-axis in each graph(average number of probes)

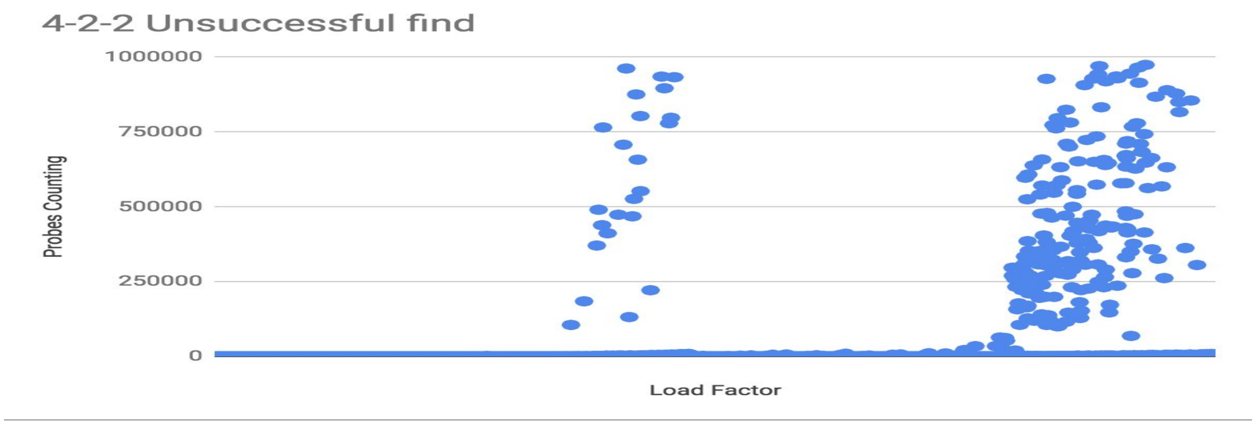


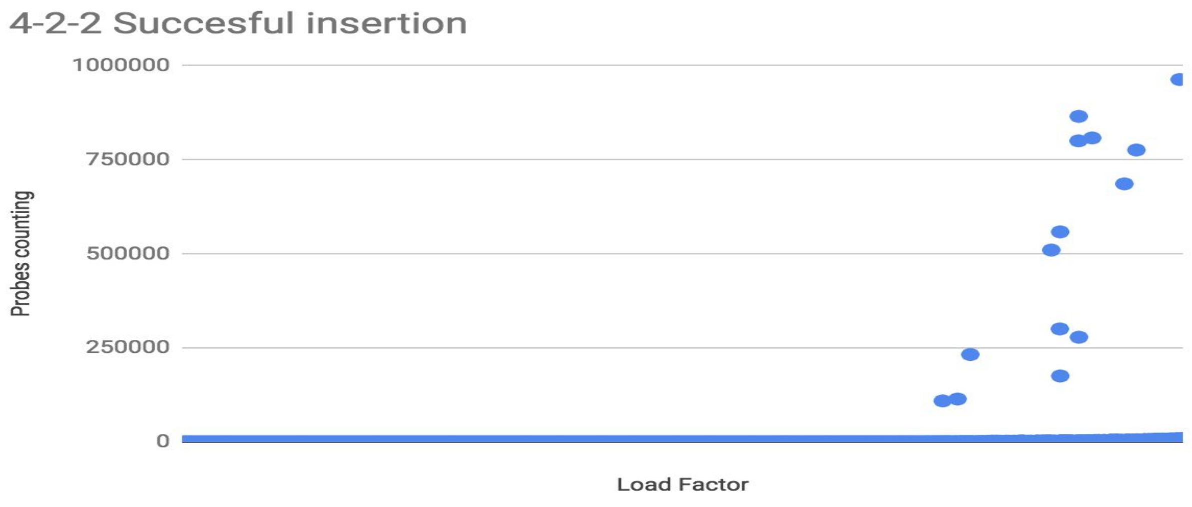

6-1-1 Successful find
6000
4000
2000
0
Probes Counting
Load Factor

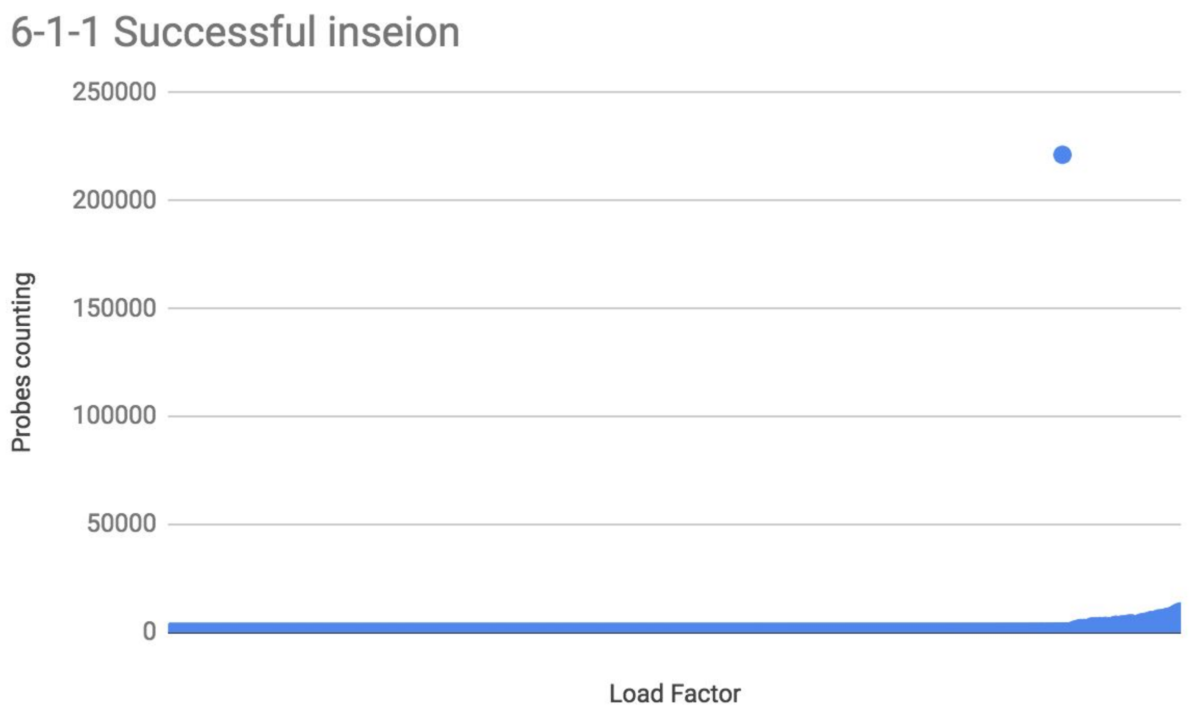

4-2-2 Successful find
1000000
750000
500000
250000
0
Probes Counting
Load Factor

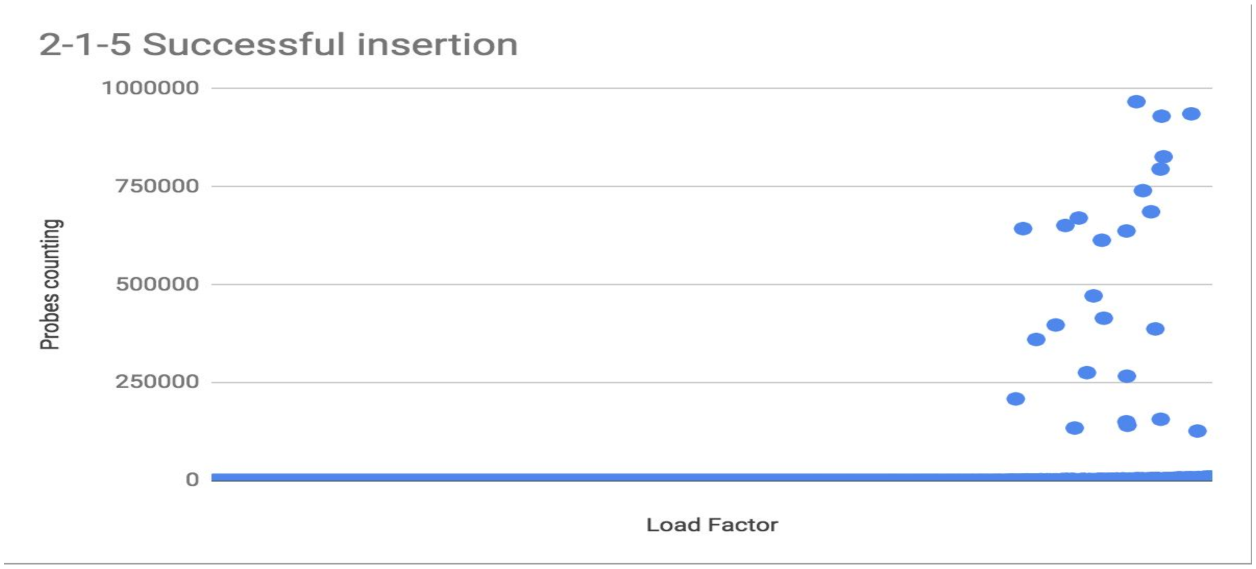

2-1-5 Unsuccessful find
4000
3000
2000
1000
0
Probes counting
Load Factor

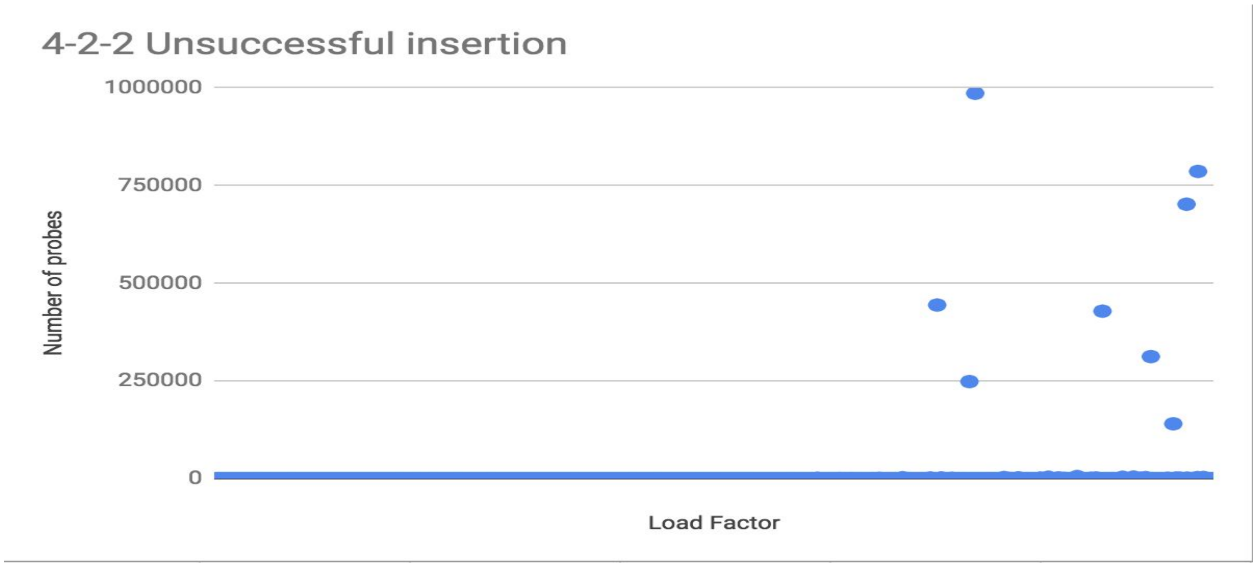

6-1-1 Unsuccessful find
800000
600000
400000
200000
0
Probes counting
Load Factor

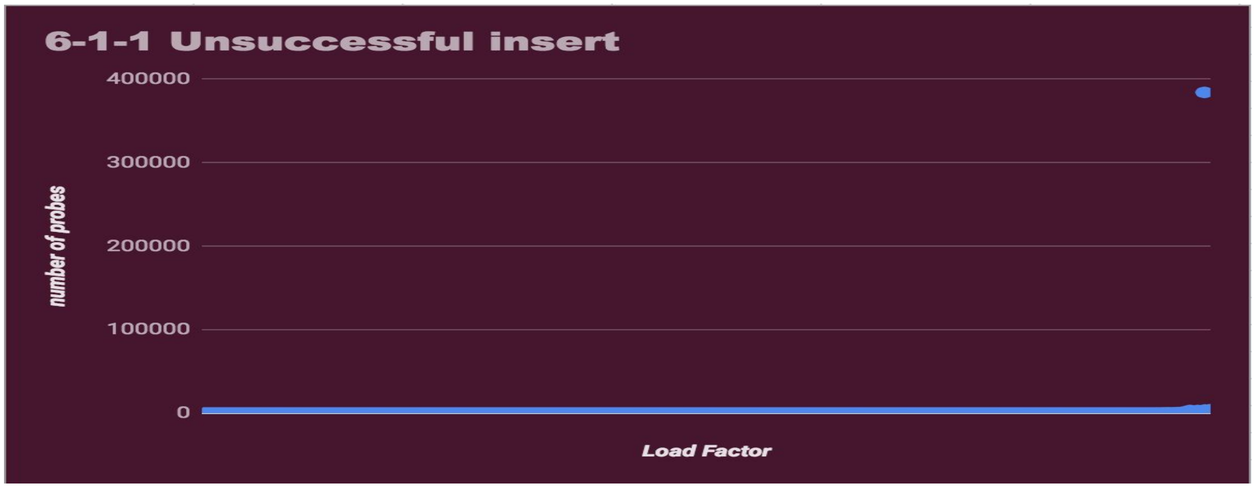

4-2-2 Unsuccessful find
1000000
750000
500000
250000
0
Probes Counting
Load Factor

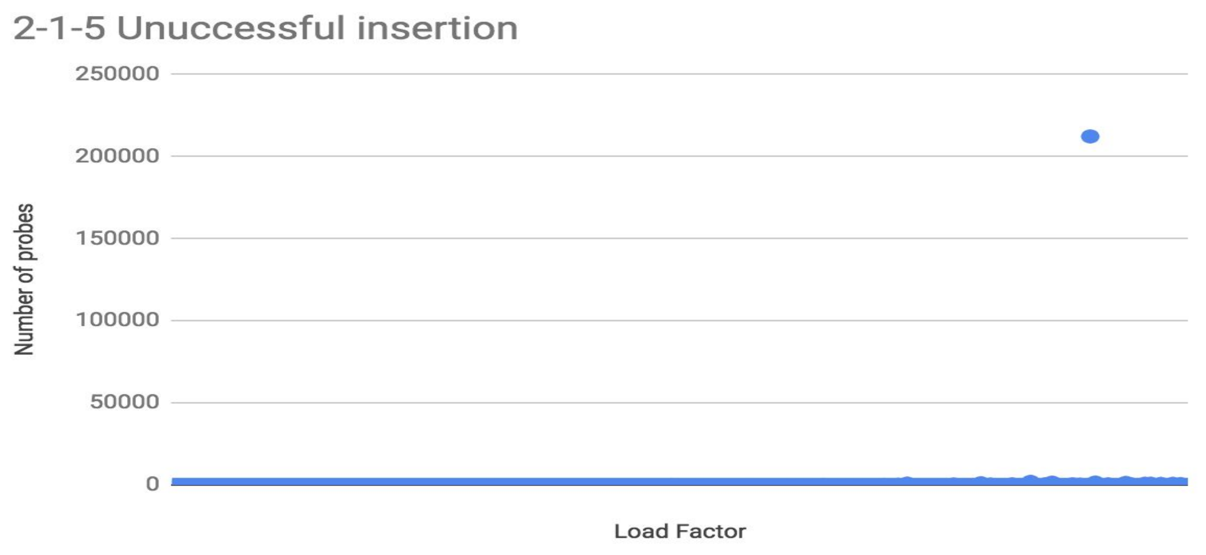

4-2-2 Succesful insertion
1000000
750000
500000
250000
0
Probes counting
Load Factor

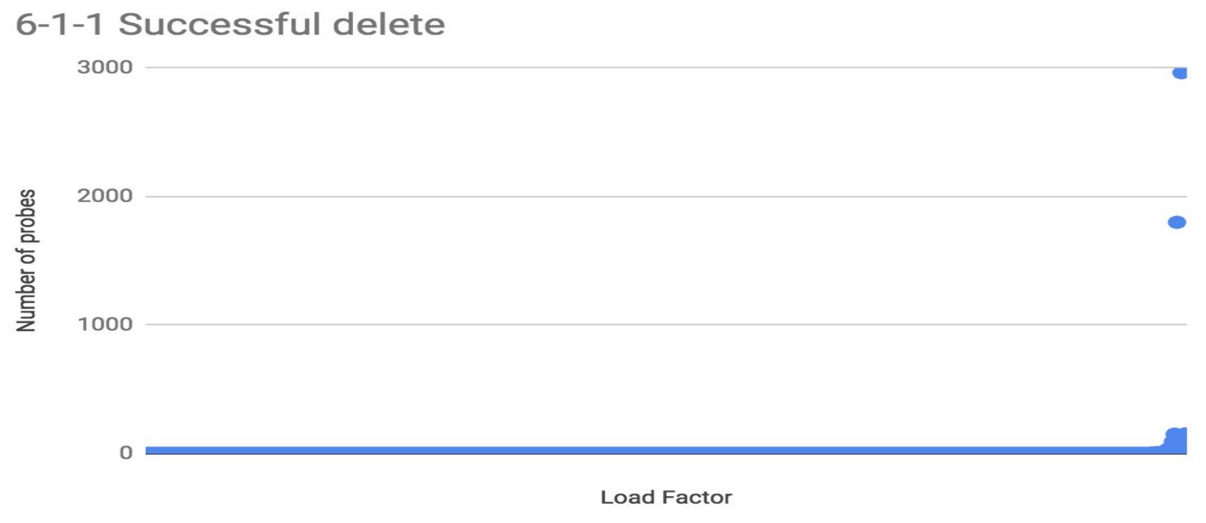

6-1-1 Successful inseion
250000
200000
150000
100000
50000
0
Probes counting
Load Factor

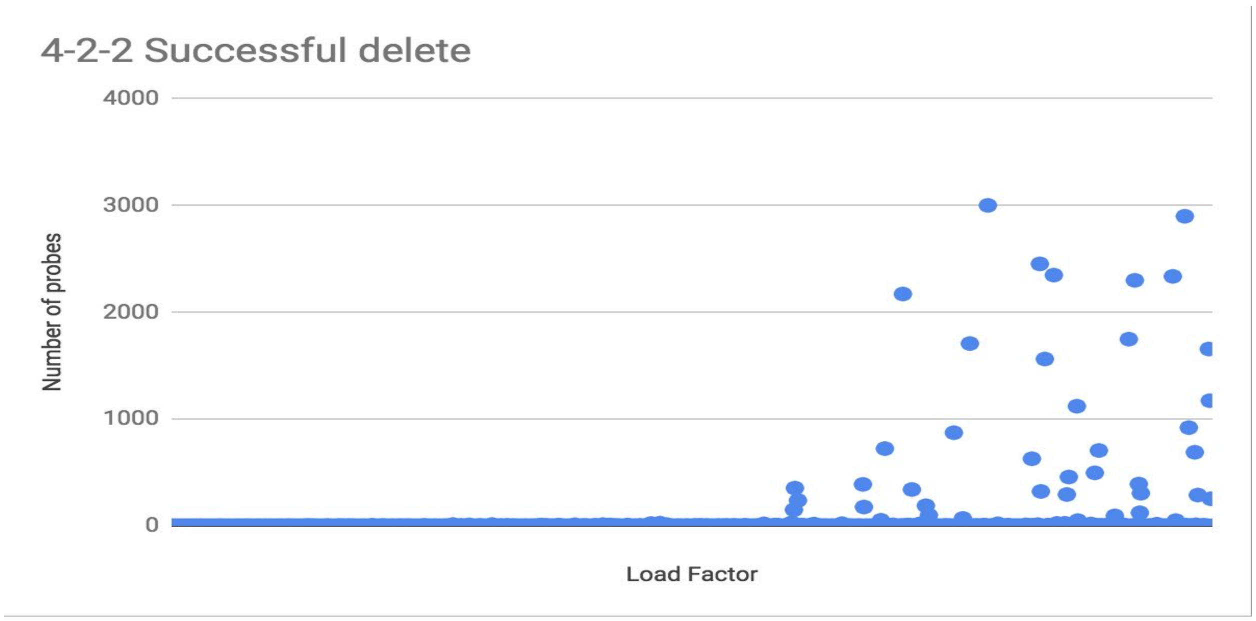

2-1-5 Successful insertion
1000000
750000
500000
250000
0
Probes counting
Load Factor

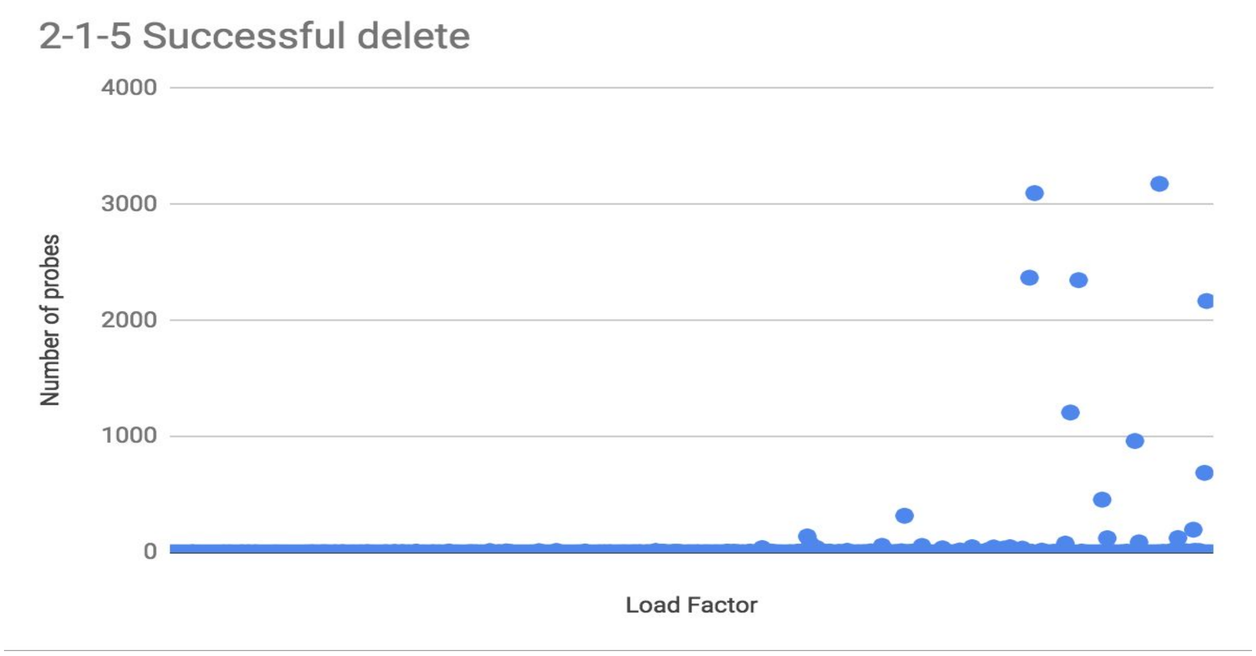

4-2-2 Unsuccessful insertion
1000000
750000
500000
250000
0
Number of probes
Load Factor

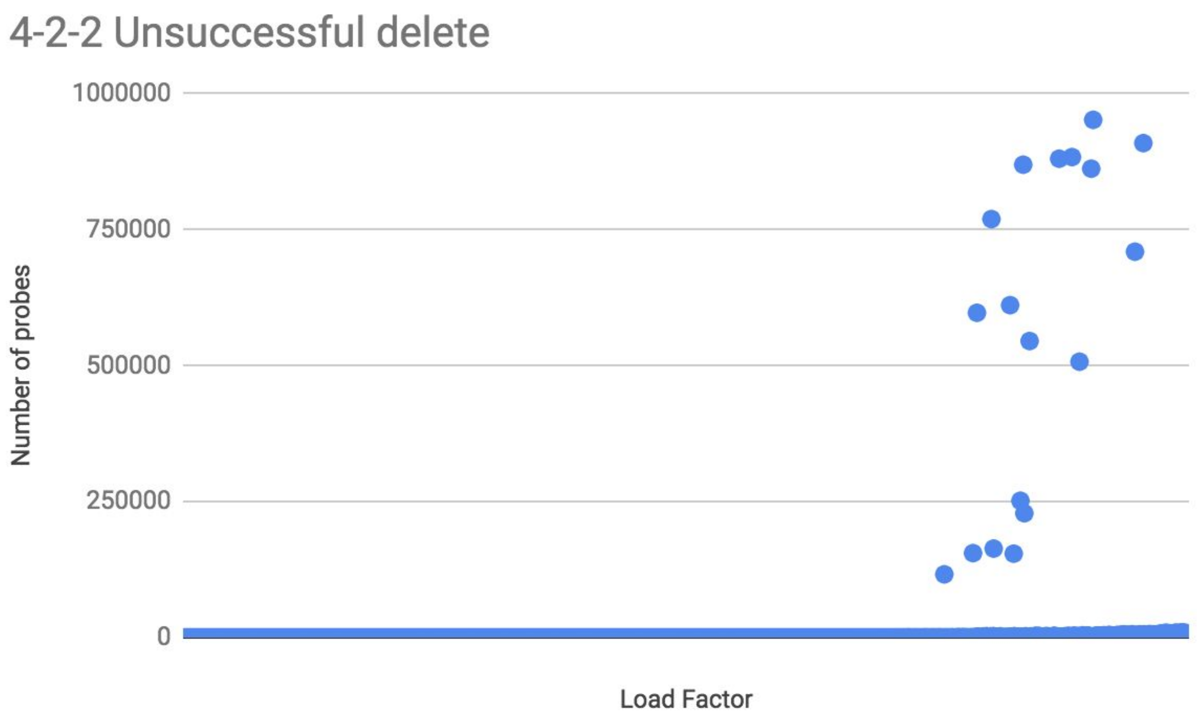

6-1-1 Unsuccessful insert
400000
300000
200000
100000
0
number of probes
Load Factor

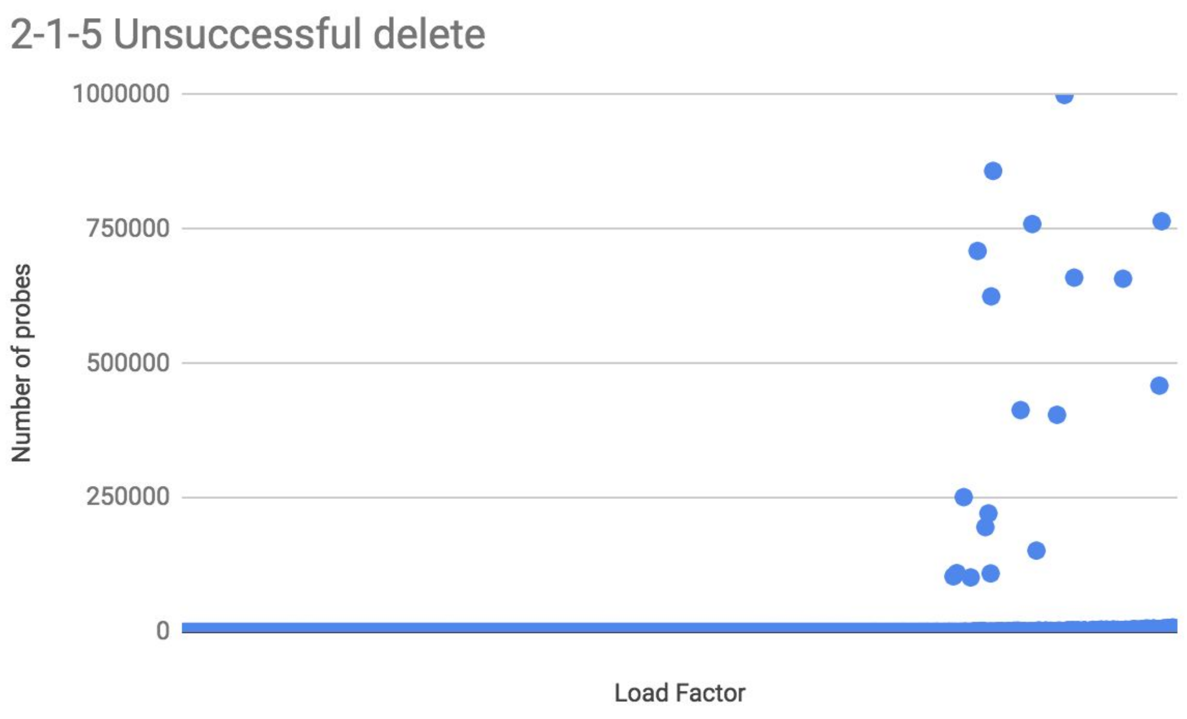

2-1-5 Unuccessful insertion
250000
200000
150000
100000
50000
0
Number of probes
Load Factor

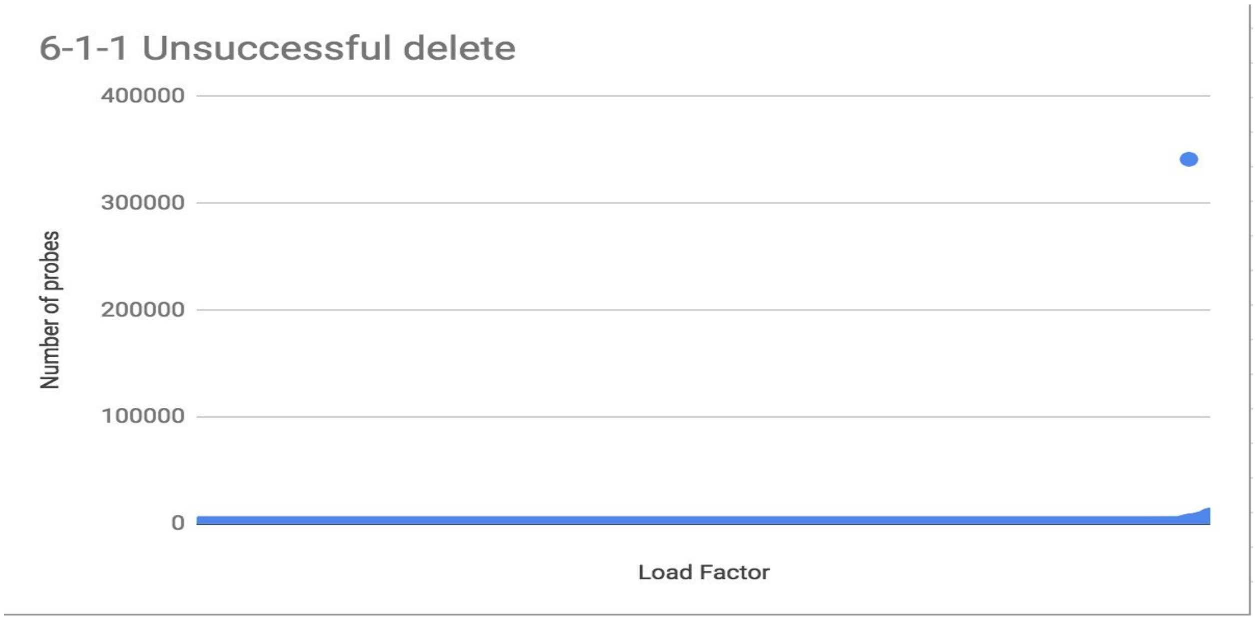

6-1-1 Successful delete
3000
2000
1000
0
Number of probes
Load Factor


4-2-2 Successful delete
4000
3000
2000
1000
0
Number of probes
Load Factor


2-1-5 Successful delete
4000
3000
2000
1000
0
Number of probes
Load Factor


4-2-2 Unsuccessful delete
1000000
750000
500000
250000
0
Number of probes
Load Factor


2-1-5 Unsuccessful delete
1000000
750000
500000
250000
0
Number of probes
Load Factor



iii) Conclusions

-First conclusion is, when the load factor increases average number of probes increase in the same proportion.

-Second conclusion: when the load factor get close to 1, we check many indexes.(to look that element)

-From the second conclusion we can obtain that; we need to increase capacity or rehash the table as the load factor get closer to 1.

-Last one: unsuccessful delete or find takes many number of probes with comparison to unsuccessful insertion.

Semih Balki-19010