Luke Erlewein

Dr. Denton Bobeldyk

CIS 263

7/12/2025

# Assignment 5

**Linear Probing collisions:**

Looking at Figure 1, you can see the output of the program.

```
PS D:\Git Repos\CIS 263> python -u "d:\Git Repos\CIS 263\Homework\Week 3\Assignementsix.py"
numCollide (Linear Probing): 50729
numCollide (Quadratic Probing): 17013
numCollide (Double Hash): 339
numCollide (Modified Linear Probing): 5310
PS D:\Git Repos\CIS 263>
```

Figure 1: Program output.

The initial number of collisions using standard linear probing was extremely high: 50,729 collisions during the insertion process. The reason for this is due to clustering, when multiple data points hash to the same index, they must linearly probe to find the next available slot. As more values collide and are inserted, blocks of occupied slots begin to form. These clusters continue to grow as new elements increasingly hash near them, resulting in a compounding collision problem. One major reason for this is that the initial hash function has a relatively small modulus of 300. This concentrates hash values into a narrow range, making collisions and subsequent clustering even more likely within that section of the table.

**Linear Probing Modification:**

To address this issue, I modified the linear probing strategy by increasing the step size used during collision resolution. Specifically, instead of incrementing the index by 1 when a collision occurs, I incremented it by 10. This means that instead of searching the next possible spot for a place to put the data, it would skip 10 slots instead. This would prevent the clustering of the data from happening. The data groups would become more dispersed and would increase the odds of finding an open slot sooner. From the data in Figure 1, it seems this change did significantly improve the number of collisions, as the total number of collisions using my improved method was only 5,310 compared to the 50,729 using the normal linear probing method.

```python
def linearProbing():
    hashTable = [None] * TableSize
    numCollide = 0

    for currentVar in InsertList:
        index = currentVar % HashMod

        while hashTable[index] is not None:
            numCollide += 1
            index = (index + 1) % TableSize
        hashTable[index] = currentVar

    return numCollide
```

Figure 2: Linear Probing function

```python
def modifiedLinearProbing():
    hashTable = [None] * TableSize
    numCollide = 0

    for currentVar in InsertList:
        index = currentVar % HashMod

        while hashTable[index] is not None:
            numCollide += 1
            index = (index + 10) % TableSize
        hashTable[index] = currentVar

    return numCollide
```

Figure 3: Modified Linear Probing function