FIT2085: COMPUTER SCIENCE

ASSIGNMENT-3: THE POTION SELLER

Due Date: 29 May 2022

HASH TABLE ANALYSIS

DONE BY A01G02:

THOMAS CAMERON

RISHI BIDANI

NETH BOTHEJU

KISARA BATUGEDARA

The Good Hash Method

```
@classmethod
def good_hash(cls, potion_name: str, tablesize: int) -> int:
    """
    Method that hashes a position for a given string and tablesize. This uses a universal hash method to prevent
    collisions and conflicts
    """
    value = 0
    a = largest_prime(52459)
    b = largest_prime(27183)
    for char in potion_name:
        value = (ord(char) + a * value) % tablesize
        a = a * b % (tablesize - 1)
    return value
```

Fig. 1: The Good Hash Method

The approach we decided to take to implement the good_hash method is similar to the universal hash method that was demonstrated in the Hash Table lecture in Week 9 (slide number 31) ,where we use two 5-digit prime numbers that get stored in the corresponding variables **a** and **b**. The variable **a** acts as the base which changes pseudo-randomly according to the value of **b** and the size of the table for every iteration of the provided key (in this case, it is the potion name).

These prime numbers are generated by the **largest_prime method** inside **primes.py**,that takes in an integer k as input and returns the largest prime number that is less than k. The largest_prime method uses the **Sieve of Eratosthenes** algorithm to generate a valid prime number.

The hashed position is stored in the variable **value** that is initialized as 0 at first but changes during iteration of the key according to the ASCII value of the characters in the key, the variable **a** multiplied by the variable value and as well as the size of the table. Once the iteration of the key is complete, it returns the variable **value** which eventually becomes the position the provided key gets hashed into.

We predicted that by using two 5-digit prime numbers, the keys will get hashed into more spread-out values thus minimizing the chance of running into collisions. Therefore, the statistics method should return the tuple containing the values of conflict count, total probe length and max probe as (0, 0, 0) for every key and table size value.

The Bad Hash Method

```
@classmethod
def bad_hash(cls, potion_name: str, tablesize: int) -> int:
    """
    Method that hashes a position for a given string and tablesize. This uses a bad hash method to hash a key where
    lots of collisions and conflicts can occur.
    """
    return ord(potion_name[0]) % tablesize
```

Fig. 2: The Bad Hash Method

The Approach we decided to take to implement the bad_hash method was fairly simple. All it does is take the ASCII value of the first character in the key and mod it by the table size. This will lead to a lot of collisions as multiple keys can be hashed to the same position. Therefore, the statistics method will return tuples of different values for conflict count, total probe length and max probe depending on the value of table size and the first character of the key.

Validating our predictions

To validate our predictions, we are going to use graphs to analyze how the good_hash method behaves vs the bad_hash method.

To do this, we had to come up with some fake data in the form of different table sizes and different string keys.

The following were the different table sizes tested:

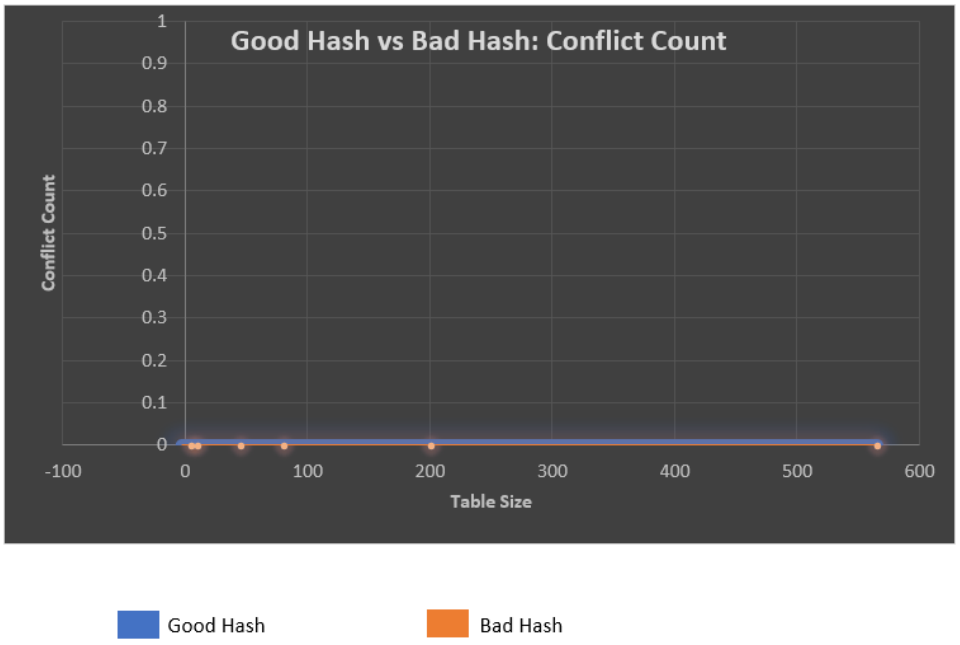
1. 200
2. 10
3. 45
4. 80
5. 5
6. 565

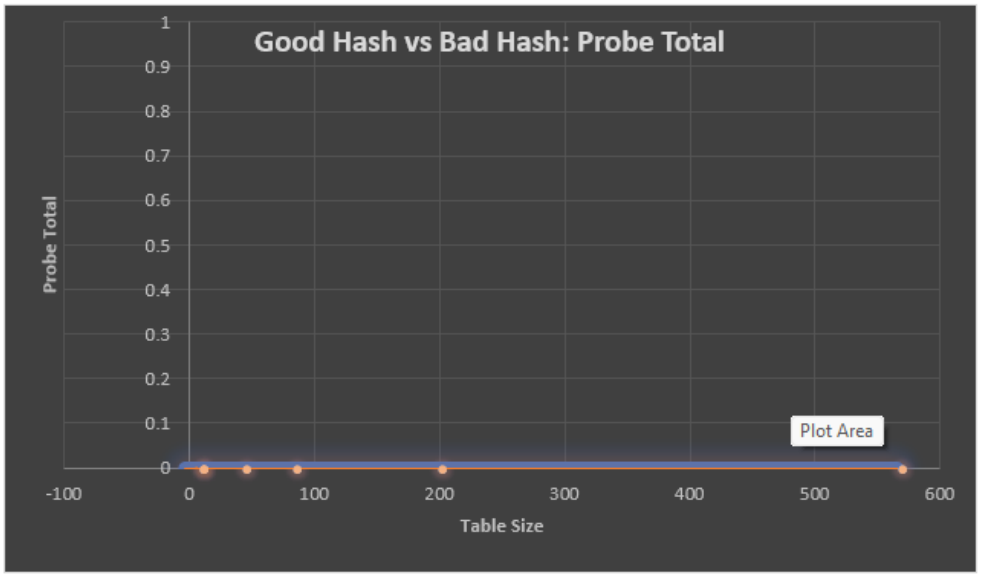
The following were the different string keys tested:

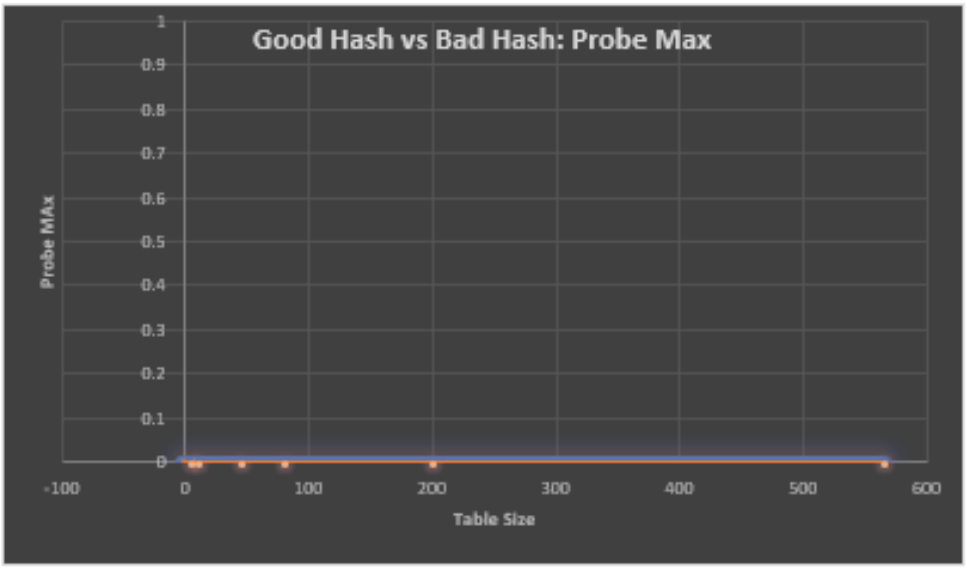
1. Potion of Health Regeneration
2. Potion of Instant Health
3. Potion of Increased Stamina

Potion Names were used as the keys to stay consistent with the context of this assignment.

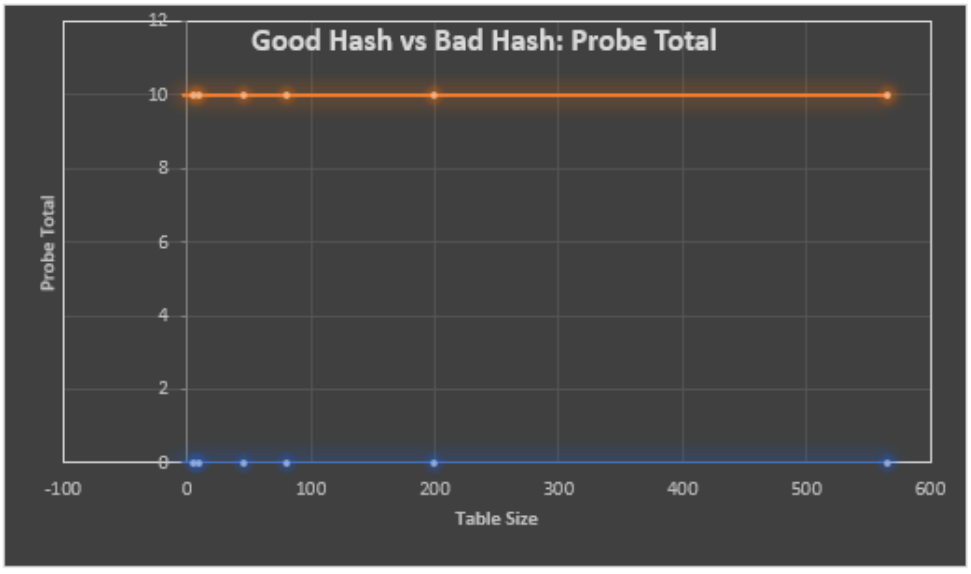
Key Used: Potion of Health Regeneration

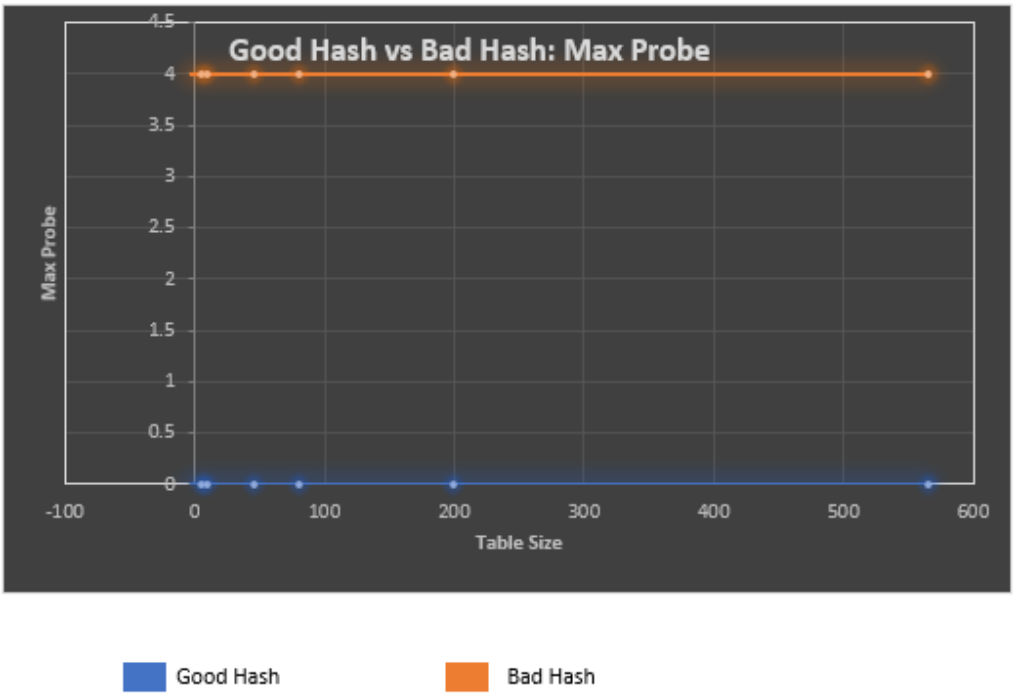



Good Hash vs Bad Hash: Probe Total
Probe Total
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-100
0
100
200
300
400
500
600
Table Size
Plot Area
Good Hash
Bad Hash

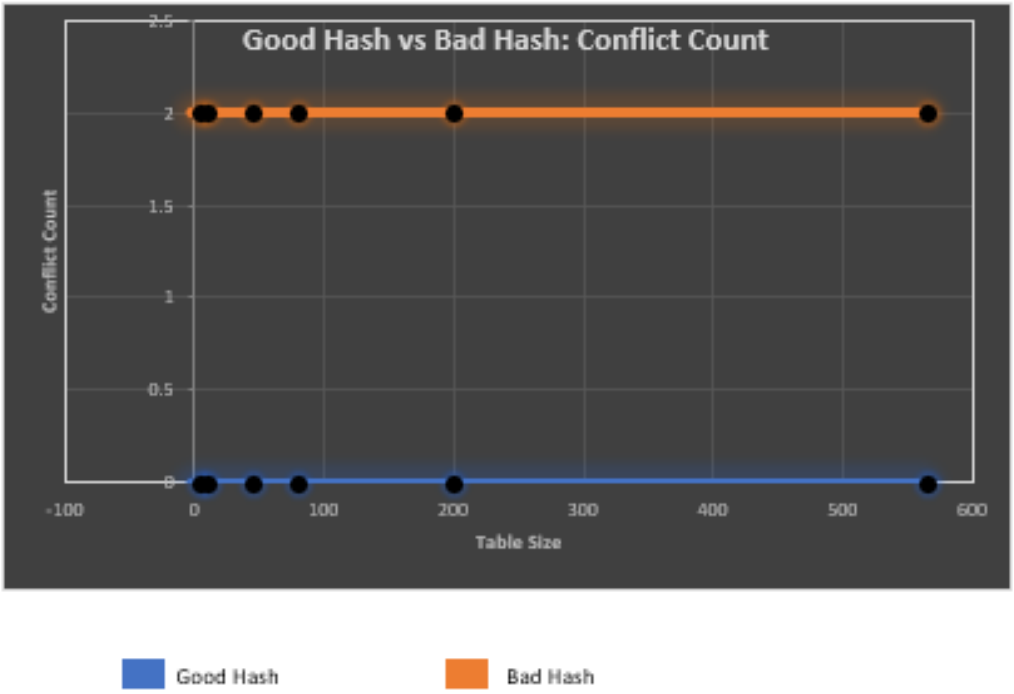

Good Hash vs Bad Hash: Probe Max
Probe MAx
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-100
0
100
200
300
400
500
600
Table Size
Good Hash
Bad Hash

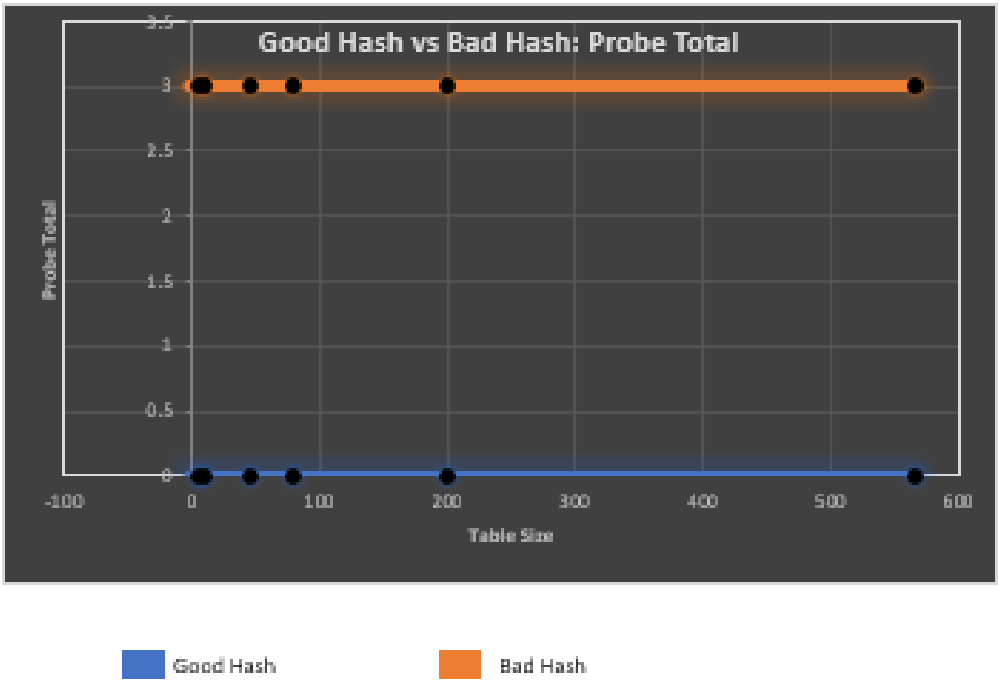
Key Used: Potion of Instant Health

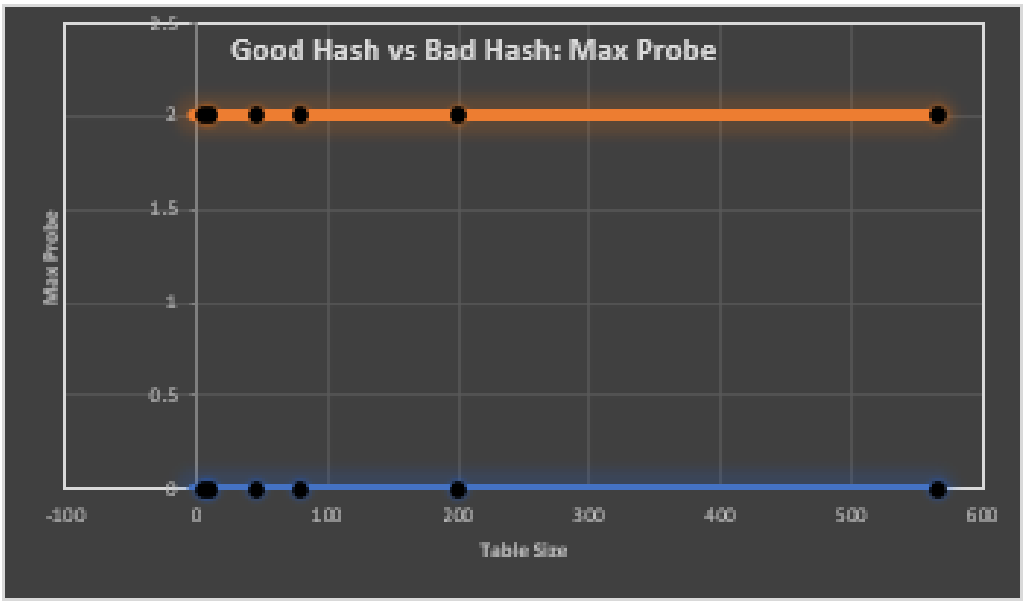



Good Hash vs Bad Hash: Max Probe
Max Probe
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-100
0
100
200
300
400
500
600
Table Size
Good Hash
Bad Hash

Key Used: Potion of Increased Stamina






Good Hash vs Bad Hash: Max Probe
Max Probe
2.5
2
1.5
1
0.5
0
-100
0
100
200
300
400
500
600
Table Size
Good Hash
Bad Hash