Assignment_7

April 24, 2024

1 Assignment 7

Peder Ørmen Bukaasen, Bård Tollef Pedersen and Eivind Lid Trøen

1.1 Exercise 1

a. f(x) = 1 (Returns "1" for all input) 0.5 point

This would mean we only get accsess to one location in the hash table. Would work, but wery inefficient.

b. f(x) = rand() (Returns a random number every time)

This would mean that we would get a random location in the hash table every time we insert or try to get a element. Wouldnt not be able to get the element back since we would not know where it is.

c. f(x) = next_empty_slot()

This would mean that we would get the next empty slot in the hash table. This works well when inserting elements, but not when we are trying to get the element back.

d. f(x) = len(x)

This would mean that we would get the length of the input string. This would work well.

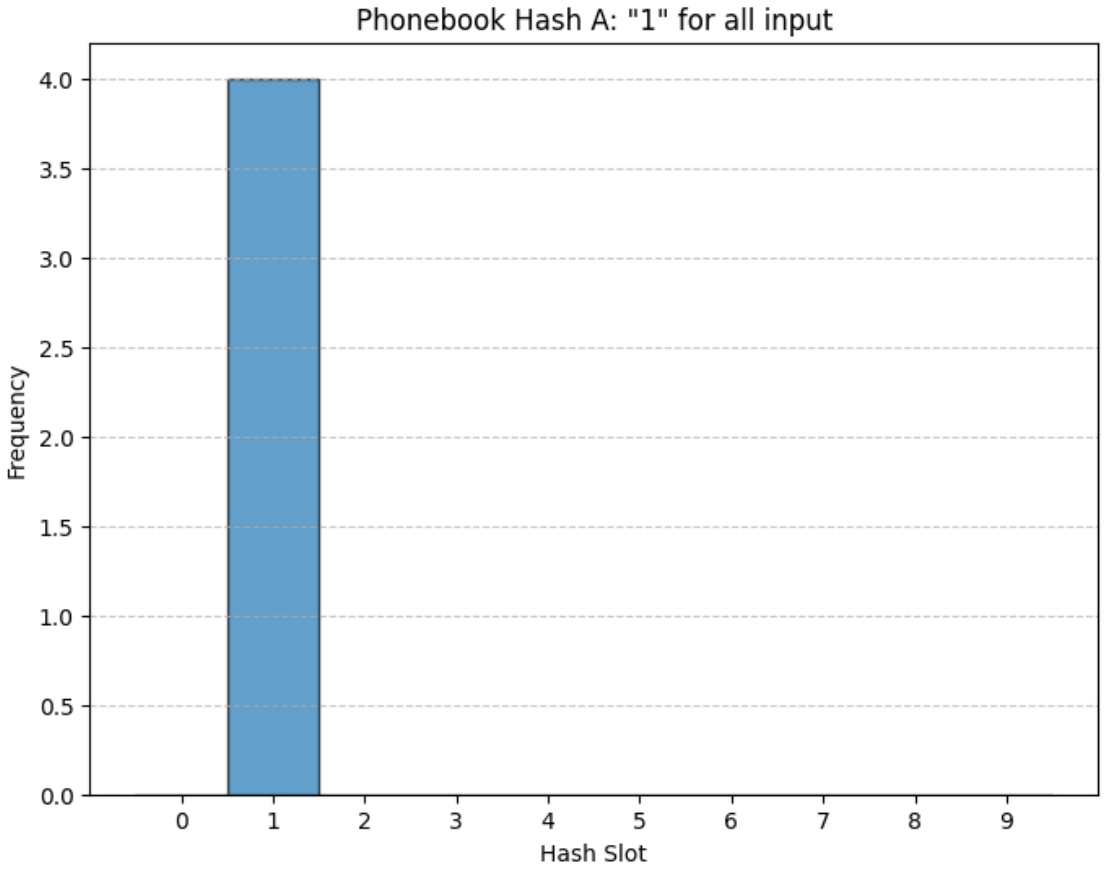
a and d are the ones that exhibit consistency.

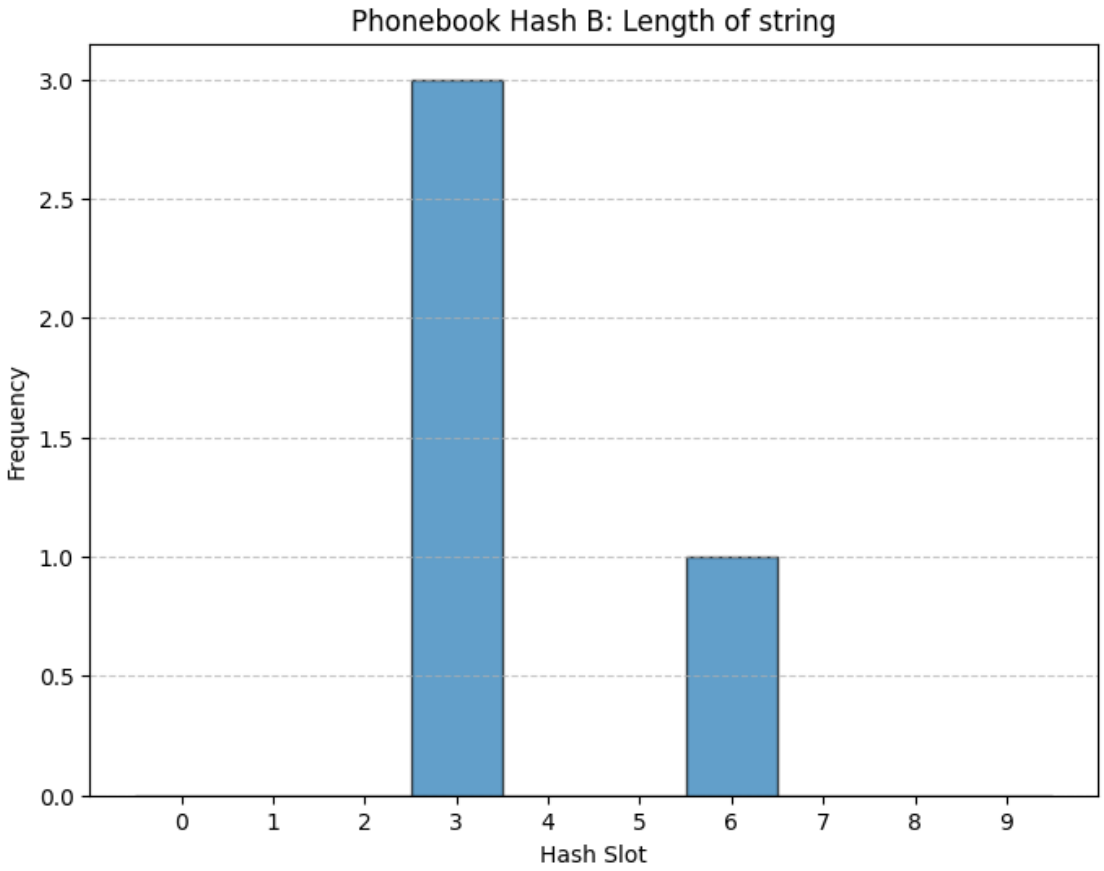
1.2 Exercise 2

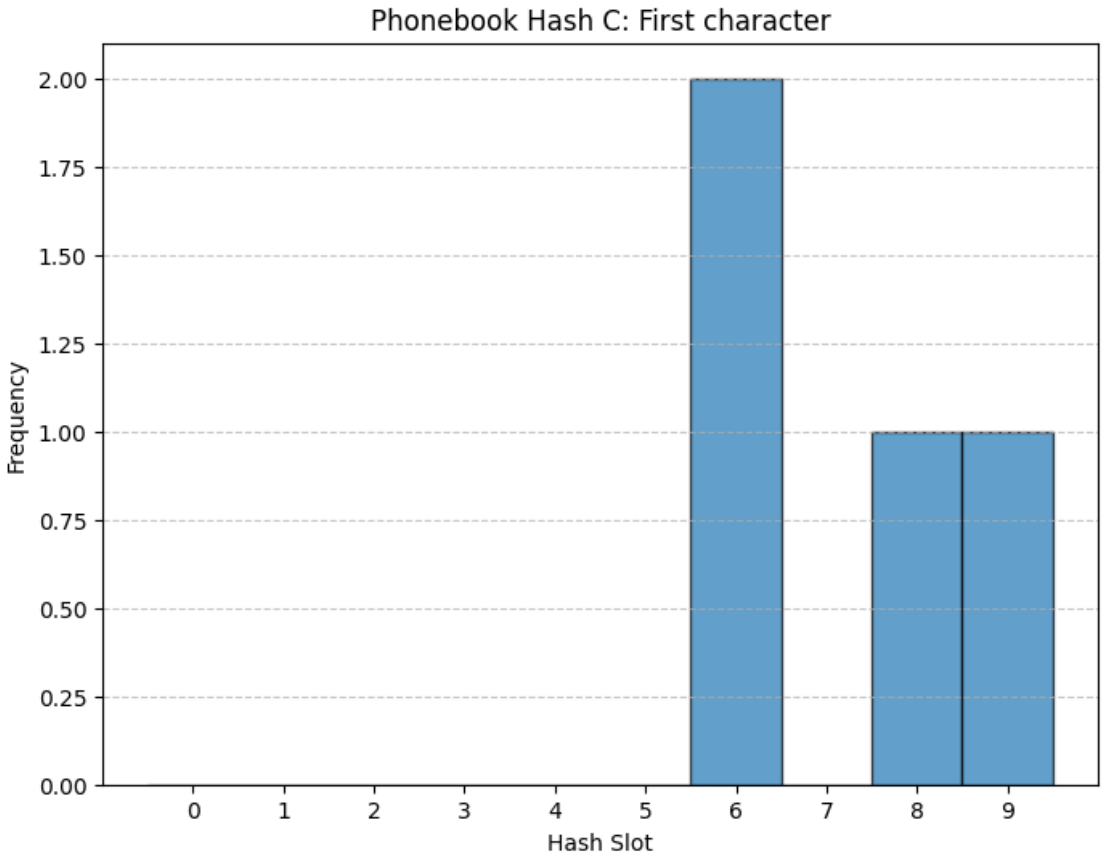
[ ]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Hash function A: Returns "1" for all input
def hash_A():
    return 1

# Hash function B: Uses the length of the string as the index
def hash_B(s, hash_size):
    return len(s) % hash_size

# Hash function C: Uses the string's first character as the index
```

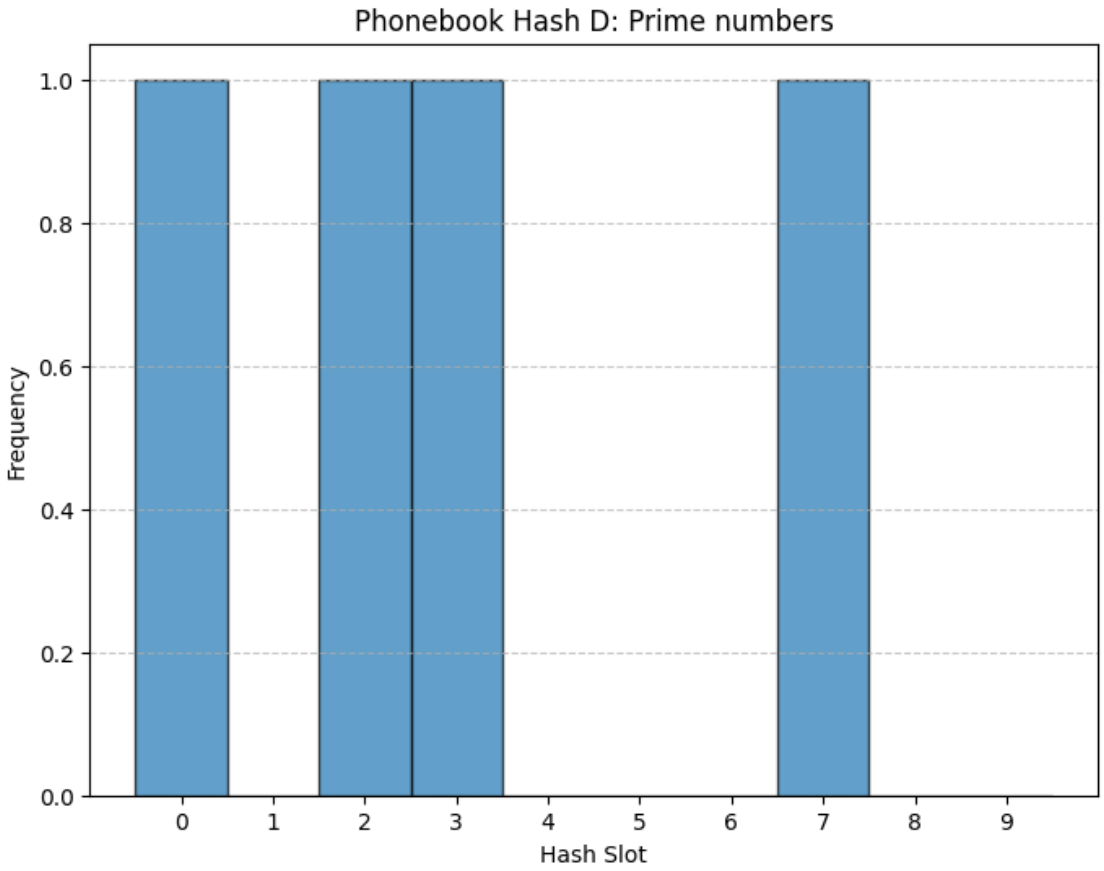
```
def hash_C(s, hash_size):
    return ord(s[0]) % hash_size

# Hash function D: Maps each letter to a prime number
def hash_D(s, hash_size):
    prime_numbers = {
        'a': 2, 'b': 3, 'c': 5, 'd': 7, 'e': 11, 'f': 13, 'g': 17, 'h': 19, 'i':
 ↪ 23,
        'j': 29, 'k': 31, 'l': 37, 'm': 41, 'n': 43, 'o': 47, 'p': 53, 'q': 59,
        'r': 61, 's': 67, 't': 71, 'u': 73, 'v': 79, 'w': 83, 'x': 89, 'y': 97,
        'z': 101
    }
    # Calculate hash value excluding spaces
    hash_value = sum(prime_numbers[char] for char in s.lower() if char != ' ')
    return hash_value % hash_size


# Create histograms
def plot_histogram(data, title, hash_size):
    plt.figure(figsize=(8, 6))
    plt.hist(data, bins=np.arange(-0.5, hash_size + 0.5, 1), edgecolor='black',␣
 ↪alpha=0.7)
    plt.xlabel('Hash Slot')
    plt.ylabel('Frequency')
    plt.title(title)
    plt.xticks(range(hash_size))
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Hash table size
hash_size = 10

# Example data
phonebook_names = ["Esther", "Ben", "Bob", "Dan"]
battery_sizes = ["A", "AA", "AAA", "AAAA"]
book_titles = ["Maus", "Fun Home", "Watchmen"]

# Plot histograms
plot_histogram([hash_A() for name in phonebook_names], 'Phonebook Hash A: "1"␣
 ↪for all input', hash_size)
plot_histogram([hash_B(name, hash_size) for name in phonebook_names],␣
 ↪'Phonebook Hash B: Length of string', hash_size)
plot_histogram([hash_C(name, hash_size) for name in phonebook_names],␣
 ↪'Phonebook Hash C: First character', hash_size)
plot_histogram([hash_D(name, hash_size) for name in phonebook_names],␣
 ↪'Phonebook Hash D: Prime numbers', hash_size)
```

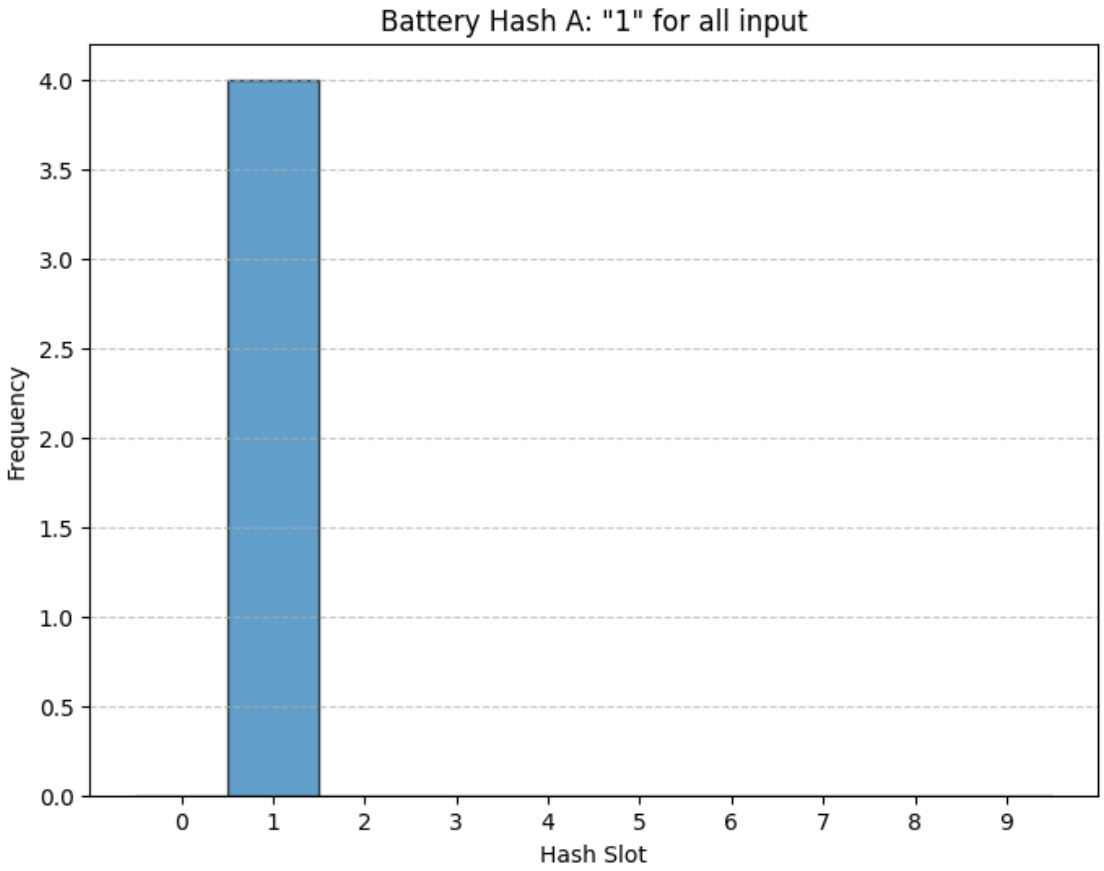
```
plot_histogram([hash_A() for size in battery_sizes], 'Battery Hash A: "1" for␣
 ↪all input', hash_size)
plot_histogram([hash_B(size, hash_size) for size in battery_sizes], 'Battery␣
 ↪Hash B: Length of string', hash_size)
plot_histogram([hash_C(size, hash_size) for size in battery_sizes], 'Battery␣
 ↪Hash C: First character', hash_size)
plot_histogram([hash_D(size, hash_size) for size in battery_sizes], 'Battery␣
 ↪Hash D: Prime numbers', hash_size)

plot_histogram([hash_A() for title in book_titles], 'Titles Hash A: "1" for all␣
 ↪input', hash_size)
plot_histogram([hash_B(title, hash_size) for title in book_titles], 'Titles␣
 ↪Hash B: Length of string', hash_size)
plot_histogram([hash_C(title, hash_size) for title in book_titles], 'Titles␣
 ↪Hash C: First character', hash_size)
plot_histogram([hash_D(title, hash_size) for title in book_titles], 'Titles␣
 ↪Hash D: Prime numbers', hash_size)
```

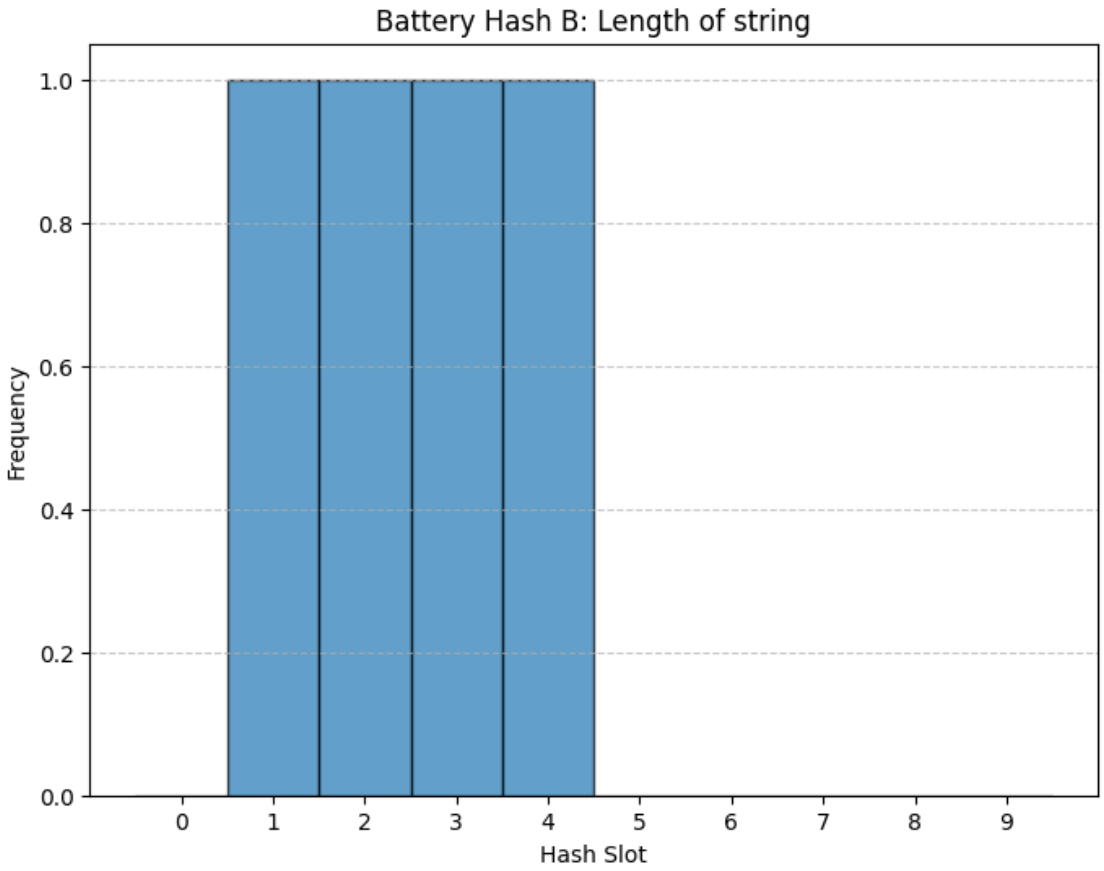


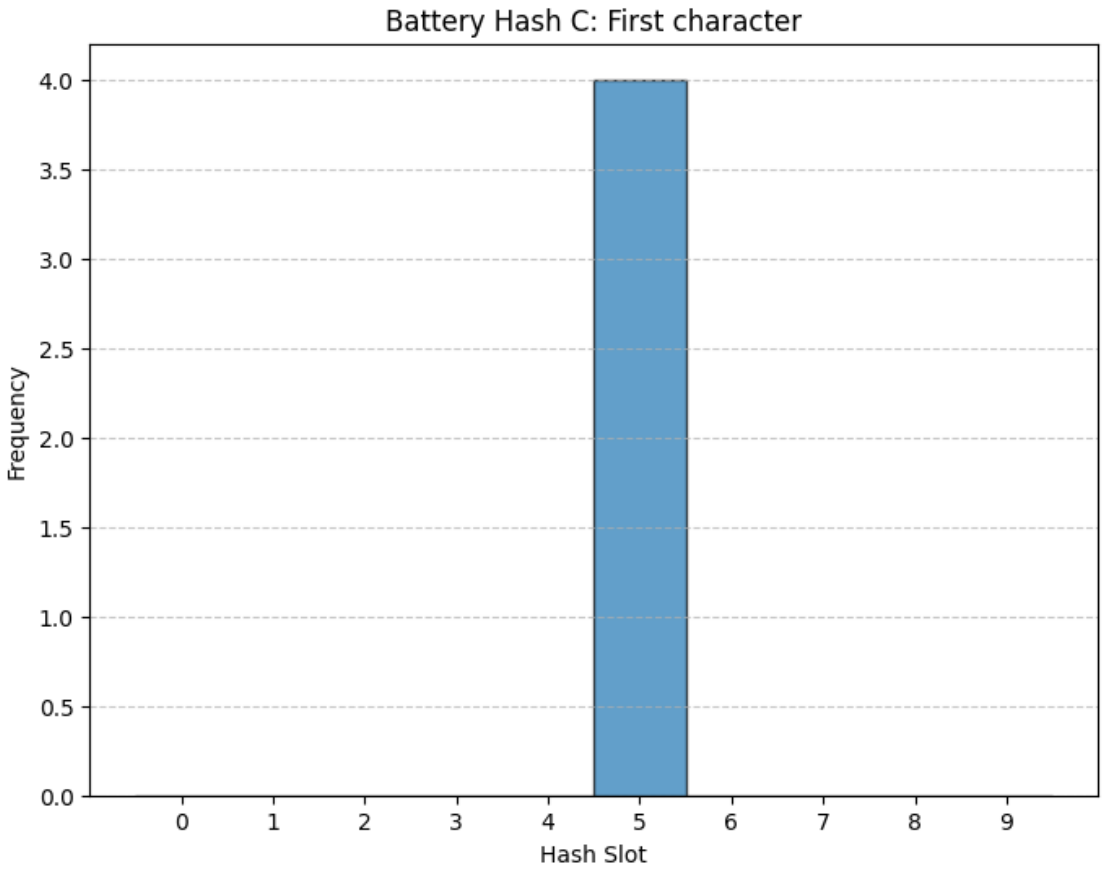

Phonebook Hash B: Length of string
Frequency
Hash Slot
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7
8
9

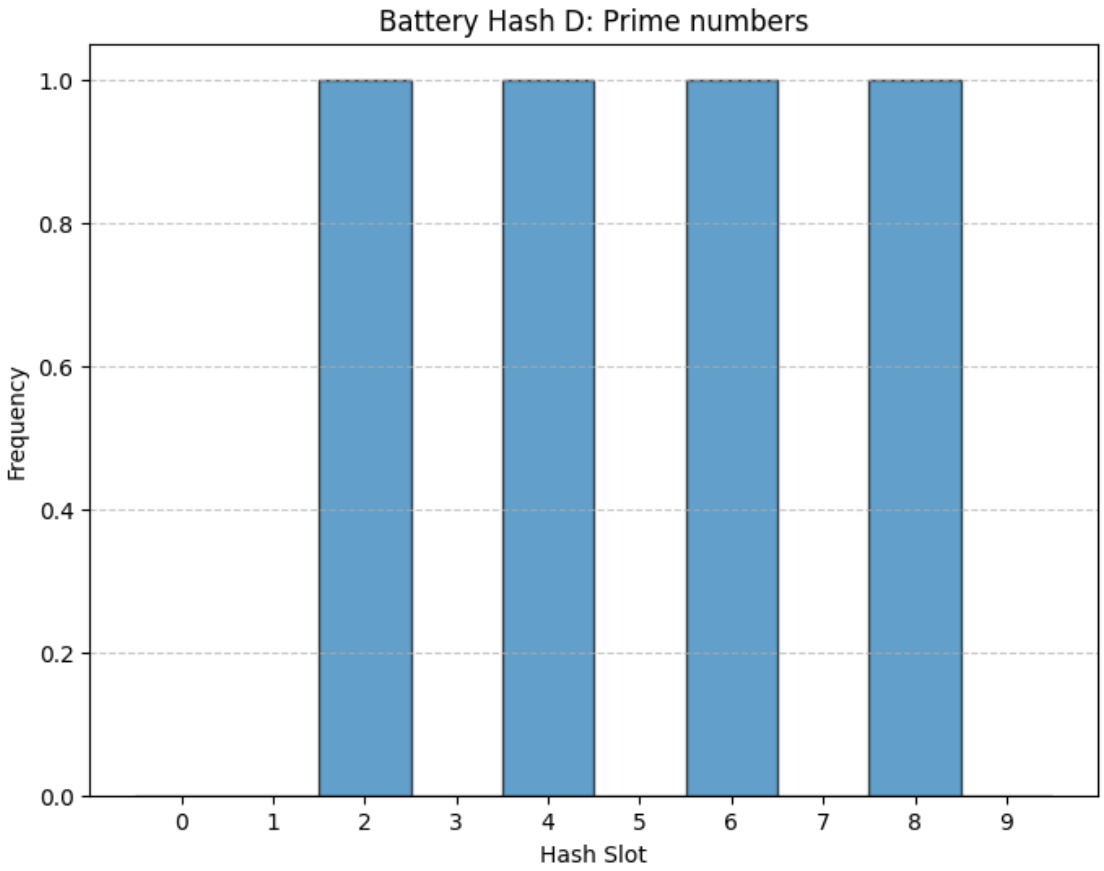

Phonebook Hash C: First character
Frequency
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1
2
3
4
5
6
7
8
9
Hash Slot

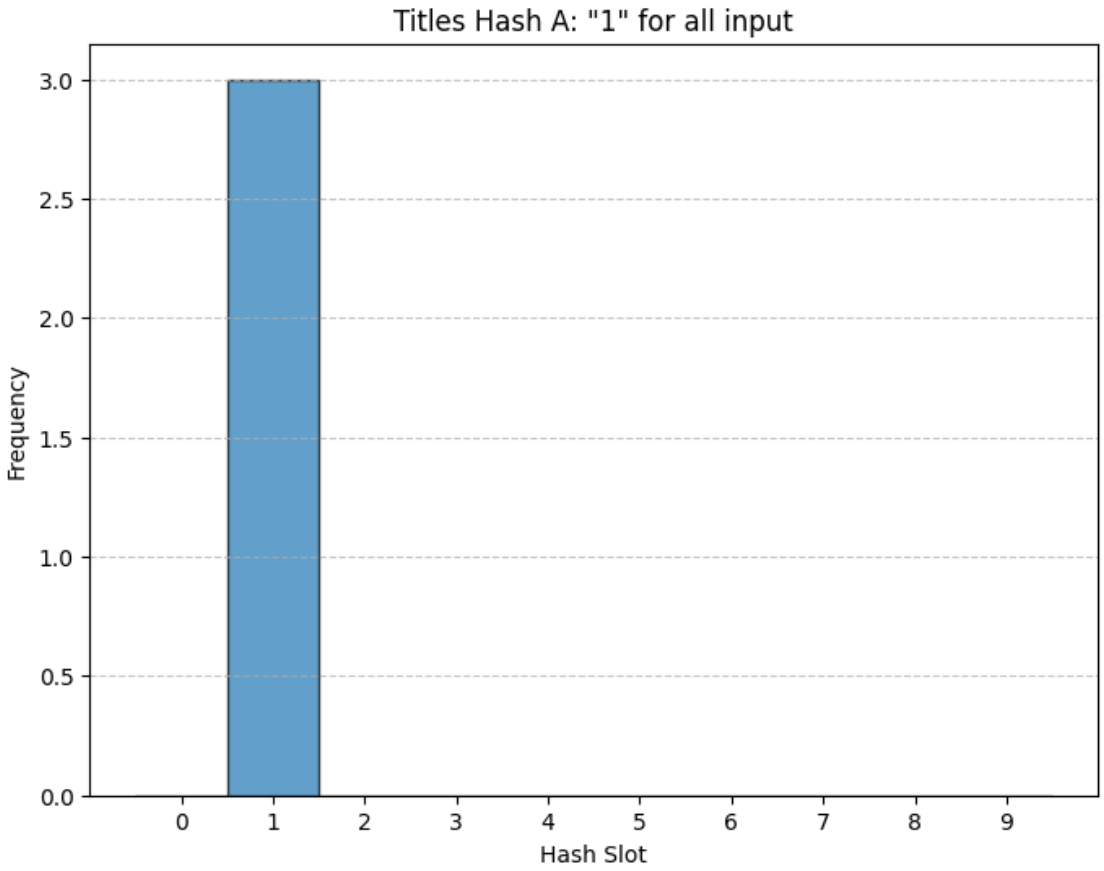

Phonebook Hash D: Prime numbers
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9
Hash Slot

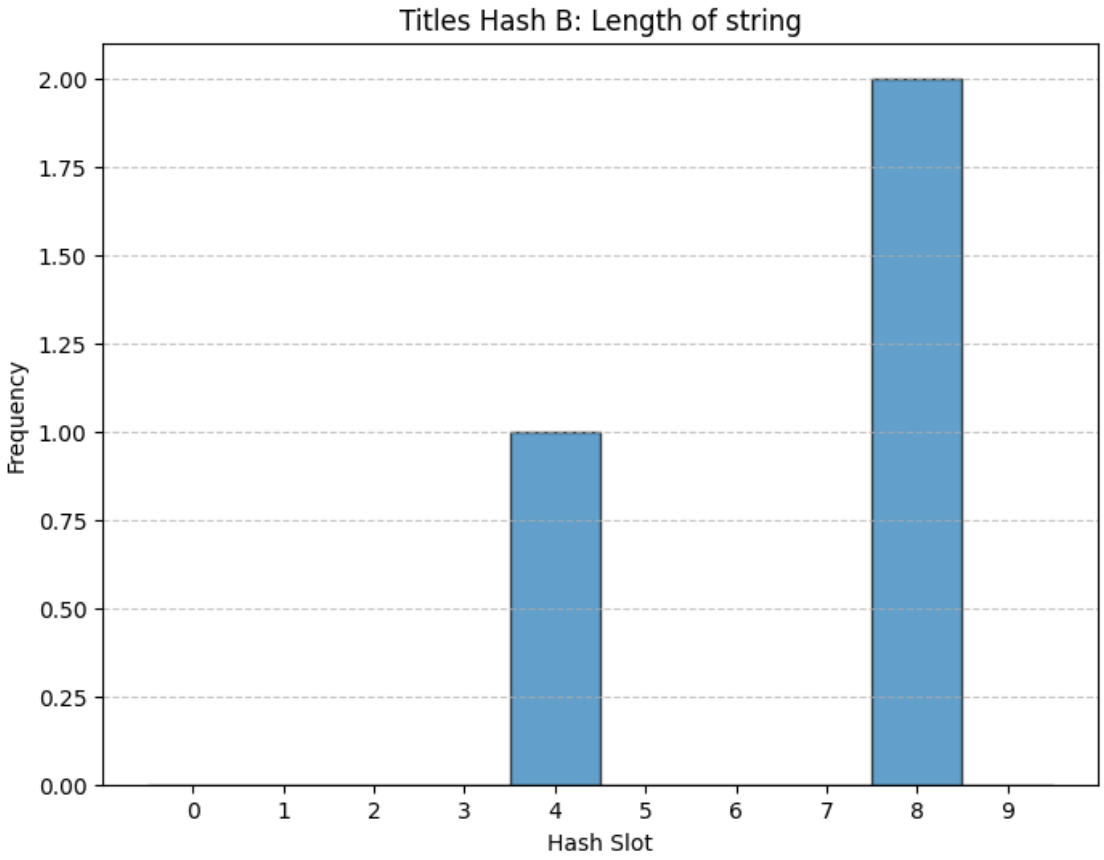

Battery Hash A: "1" for all input
Frequency
Hash Slot
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
1
2
3
4
5
6
7
8
9

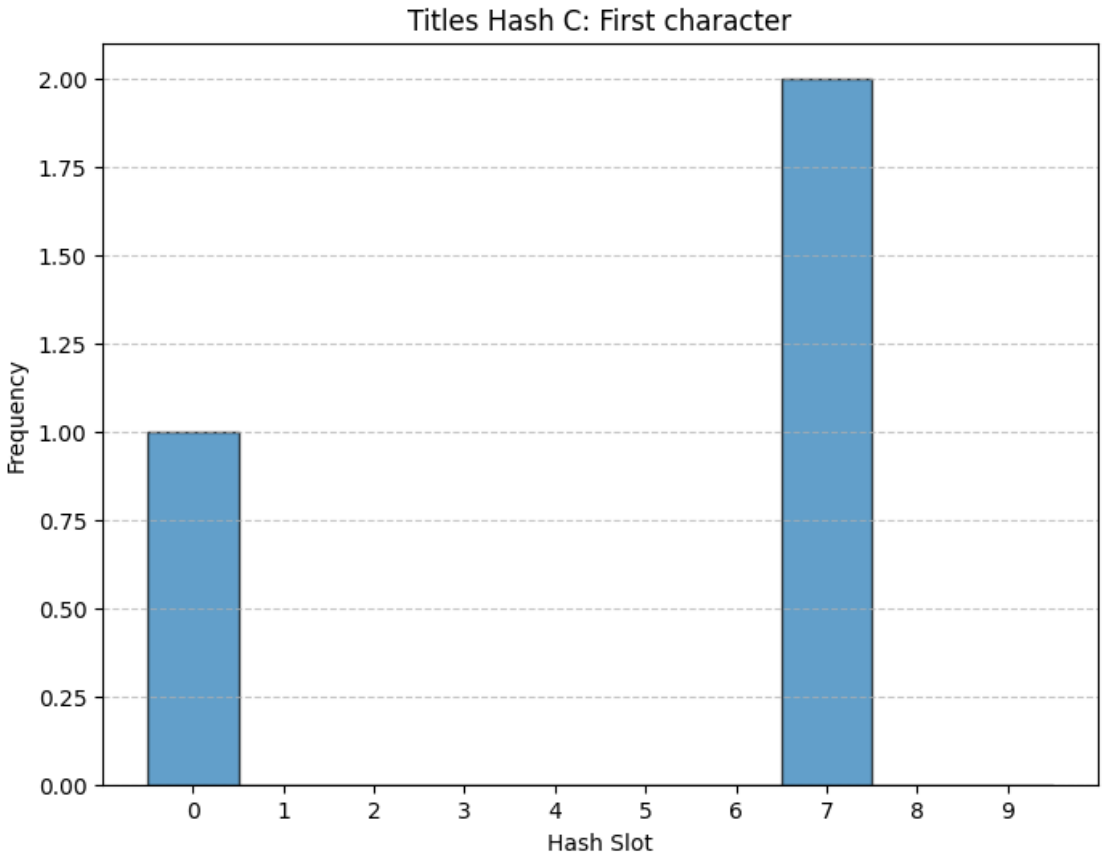

Battery Hash B: Length of string
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9
Hash Slot

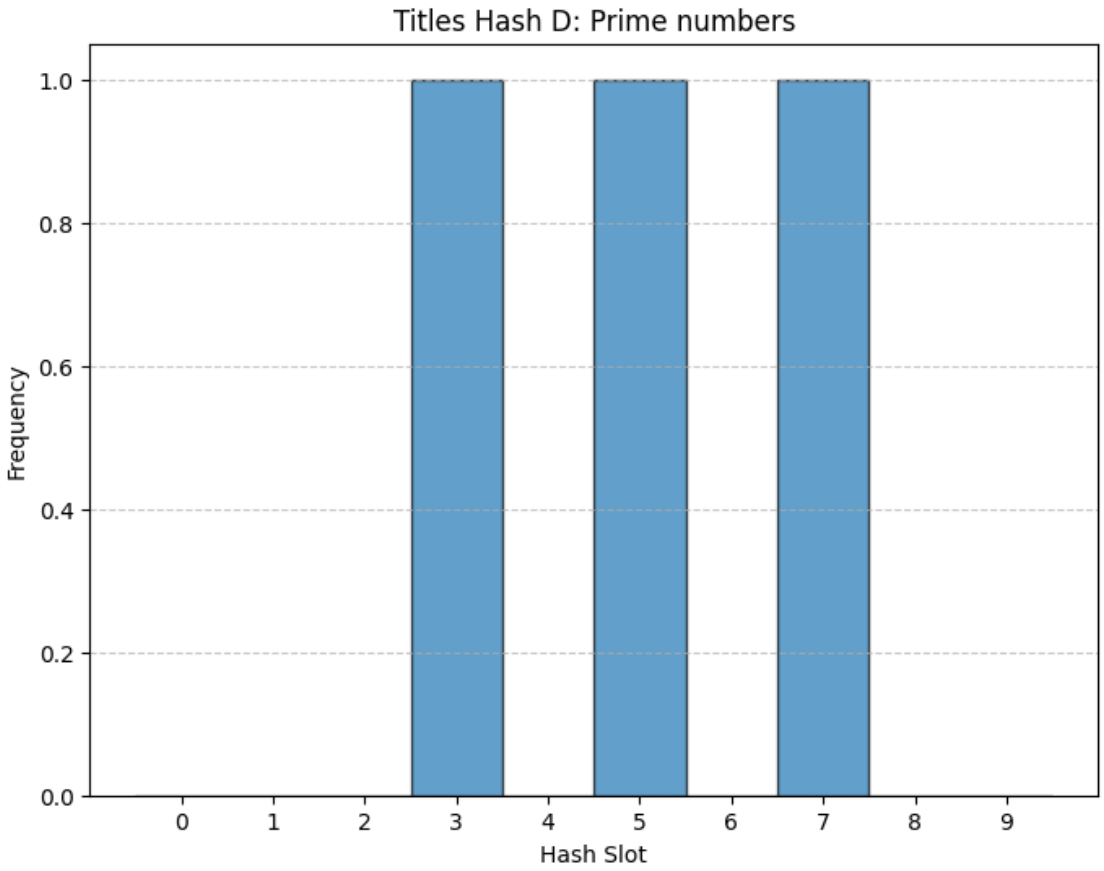

Battery Hash C: First character
Frequency
Hash Slot
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
1
2
3
4
5
6
7
8
9


Battery Hash D: Prime numbers
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9
Hash Slot


Titles Hash A: "1" for all input
Frequency
Hash Slot
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7
8
9


Titles Hash B: Length of string
Frequency
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1
2
3
4
5
6
7
8
9
Hash Slot


Titles Hash C: First character
Frequency
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1
2
3
4
5
6
7
8
9
Hash Slot



From plots we can se that:
For phonebook, hash function D is the best.
For battery size, hash function B and D is the best.
For book titles, hash function D is the best.

A is always the worst. This makes all the elements go to the same location, and worst case scenario is O(n).

For phonebook, names are more likely to start with the same letter or have the same length. This makes hash function D the best.
For battery size, the firts letter is the same, and the length vary. This makes hash function B and D the best.
For book titles, the same as for phonebook applies, but here the length is more unlikely to be the same. This makes hash function B and D the best.

1.3 Exercise 3

[ ]:
```
# Input string
input_string = "hello"

# Calculate sum of ASCII values of each character
```

```
ascii_sum = sum(ord(char) for char in input_string)

# Compute hash value using modulo 13
hash_value = ascii_sum % 13

# Print hash value
print("The hash value for the input string 'hello' is:", hash_value)
```

```
The hash value for the input string 'hello' is: 12
```

Hello in ASCII:
h = 104
e = 101
l = 108
l = 108
o = 111

sum is 532.

532 % 13 = 12

The hash value for the input string “hello” is 12.