Design a simple hash function

Take the characters in a string and convert them to their corresponding ASCII values.

Sum the ASCII values of the previous characters and multiply by 17, which is a prime number, to even out the spread of the hash values - then add the ASCII value of the latest character.

Take the modulus of the above and a prime number close to the size of the hash table. This is keeping in mind the memory constraints.

A collision is where two different strings produce the same hash value which causes problems as they cannot be placed in the same index. To deal with it for this simple hash function, I would use linear probing which is where the next available index is checked if a collision occurs.

Walkthrough without a collision:

String = ‘no’

hash = 0
‘n’ = 110
‘o’ = 111

1. hash = (0 * 17 + 110) % 7 = 5
2. hash = (5 * 17 + 111) % 7 = 0

Store ‘no’ at index 0

Walkthrough with a collision

hash = 0
‘p’ = 112
‘i’ = 105

1. hash = (0 * 17 + 112) % 7 = 0
2. hash = (0 * 17 + 105) % 7 = 0

Store ‘pi’ at index 0. This isn’t possible and causes a collision!
The next available index, 1, is available. Therefore ‘pi’ is stored there to resolve the collision.