<u>Assignment 1:</u>

Analyse the two hash functions provided:

- $h_1(k) = k \bmod 16$:
  - $h_1(0) = 0 \bmod 16 = 0$
  - $h_1(1) = 1 \bmod 16 = 1$
  - $h_1(2) = 2 \bmod 16 = 2$
  - …..
  - $h_1(16) = 16 \bmod 16 = 0$
  - …..
  - $h_1(33) = 33 \bmod 16 = 1$
  - …..

- $h_2(k) = \lfloor k / 16 \rfloor$:
  - $h_2(0) = \lfloor 0 / 16 \rfloor = 0$
  - $h_2(1) = \lfloor 1 / 16 \rfloor = 0$
  - $h_2(2) = \lfloor 2 / 16 \rfloor = 0$
  - …..
  - $h_2(17) = \lfloor 17 / 16 \rfloor = 1$
  - …..
  - $h_2(31) = \lfloor 31/ 16 \rfloor = 1$
  - $h_2(32) = \lfloor 32 / 16 \rfloor = 2$
  - …..
  - $h_2(39) = \lfloor 39 / 16 \rfloor = 2$

Evaluate the performance of each hash function by considering the given factors:

- $h_1(k) = k \bmod 16$:
  - The keys that map to the same slot are {0,16,32}, {1,17,33}, {2,18,34}, {3,19,35}, {4,20,36}, {5,21,37}, {6,22,38}, {7,23,39}, {8,24}, {9,25}, {10,26}, {11,27}, {12,28}, {13,29}, {14,30}, {15,31}
  - number of collisions: 24 (0-15 are mapped on 0-15 accordingly, 16-31 are mapped again on 0-15 accordingly, 32-39 are mapped on 0-7 accordingly)
  - average number of keys per slot: (40/16) 2.5
  - load factor: 1 (16/16)

- $h_2(k) = \lfloor k / 16 \rfloor$:

  - number of collisions: 37 (keys 0-15 are mapped to slot 0, keys 16-31 are mapped to slot 1 and keys 32-39 are mapped to slot 2)
  - average number of keys per slot: 13.33
  - load factor: 0.1875 (3/16)

Compare the two hash functions and determine which one would perform better for this problem and explain your reasoning.

- h1(k) would perform better in this case, thanks to distribution across all slots. On the other hand, h2(k) uses in this case only 3 slots and in one slot in this given example is 16 keys, which takes longer to loop through. It is important to keep in mind that the performance can change with different table sizes and input sizes.

<u>Assignment 2:</u>

The goal is to optimize the search for an element with a specific key in a linked list of length n, where each element contains a long character string key and its hash value. Hash value is a fixed-size integer that uniquely represents an input.

During my research I found various techniques that utilize hash values to improve the efficiency of searching a linked list. I picked some of them below:

1. Hash Table: A hash table is used to store the linked list elements using their hash values as keys. It uses a hash function to compute the index. This allows constant-time (O(1)) access to the elements with a specific key. The advantages are that it is fast and easy to implement. Disadvantage is that we must deal with collisions sometimes.

2. Hashing with Separate Chaining: In this technique, a hash table is used to store the linked list elements, but collisions are resolved using a linked list. Each slot in the hash table contains a linked list of elements with the same hash value. This allows for O(1) access to the elements with a specific key on average, but worst-case performance can degrade to O(n) in case of all elements end up in the same slot. The advantages are that it is fast and easy to implement, and it also resolves dealing with collisions. Disadvantages are that it uses extra space to resolve the collisions.

3. Cuckoo Hashing: In this technique, two hash tables are used to store the elements. Each table has its own hash function. If an element hashes to a slot in one table that is already occupied, it is moved to the corresponding slot in the other table. Cuckoo hashing has worst-case constant-time performance (O(1)), but it requires a large amount of memory.

Utilizing hash values improves search efficiency, however, there are some disadvantages each technique. Hash tables can require a very big memory and their performance can drop when many collisions occur. Hashing with separate chaining can also degrade in performance in the worst case, but its memory complexity is lower than that of hash tables. Cuckoo hashing requires a very large memory to store the two hash tables, but it has a constant-time performance in the worst case.

In conclusion, the utilization of hash values can greatly improve the efficiency of searching a linked list with long character string keys. The right choice of course depends on the specific problem.

References:

Jain, N. (2022, January 17). *Separate chaining*. Scaler Topics. https://www.scaler.com/topics/data-structures/separate-chaining/

Cuckoo hashing. (2022, September 18). Wikipedia, the free encyclopedia. Retrieved March 6, 2023, from https://en.wikipedia.org/wiki/Cuckoo_hashing

Cuckoo hashing - Worst case O(1) lookup! - GeeksforGeeks. (2023, January 11). GeeksforGeeks. https://www.geeksforgeeks.org/cuckoo-hashing/

Assignment 3:

- It violates the definition of big O, which describes the upper bound of the running time of an algorithm. It is the worst-case scenario in terms of time complexity. That is why the statement saying that an algorithm's runtime is at least O(n^2) doesn't make sense, because it is a lower bound. To describe the lower bound of an algorithm's runtime we use Omega notation. So, we can change the statement to "The running time of algorithm A is at least Omega(n)" for example.

Assignment 4:

- Yes

$$2^{(n+1)} = O(2^n) \quad ?$$

$$2^{(n+1)} \leq c \cdot 2^n$$

$$2 \cdot 2^n \leq c \cdot 2^n$$

$$c \leq 2 \qquad \underline{\underline{\text{TRUE}}}$$

- No

$2^{2n} = O(2^n)$

$2^{2n} \le c \cdot 2^n$

$2^n \cdot 2^n \le c \cdot 2^n$

$2^n \le c$ $\times$ not a constant $2^{2n} \neq O(2^n)$

Assignment 5:

- $O(n)$
- $O(2n) = O(n)$
- $O(n^2)$
- $O(2n^2) = O(n^2)$
- $O(n^4)$