Computational Problem Solving CSCI-603
Chained Hash Tables Lab 7

07/26/2021

1 Introduction

In this lab, you will be revising the hash table implementation of a map to use chainning for handling collisions. You will also measure hashing performance with different hash functions.

1.1 Introduction to Chained Hashing

Chaining is an alternative to open addressing for handling collisions. Chaining handles collisions by using a hash table array that contains a list at each location. Each unique key that hashes to the same location is simply added to that list. See Figure 1 for an example.

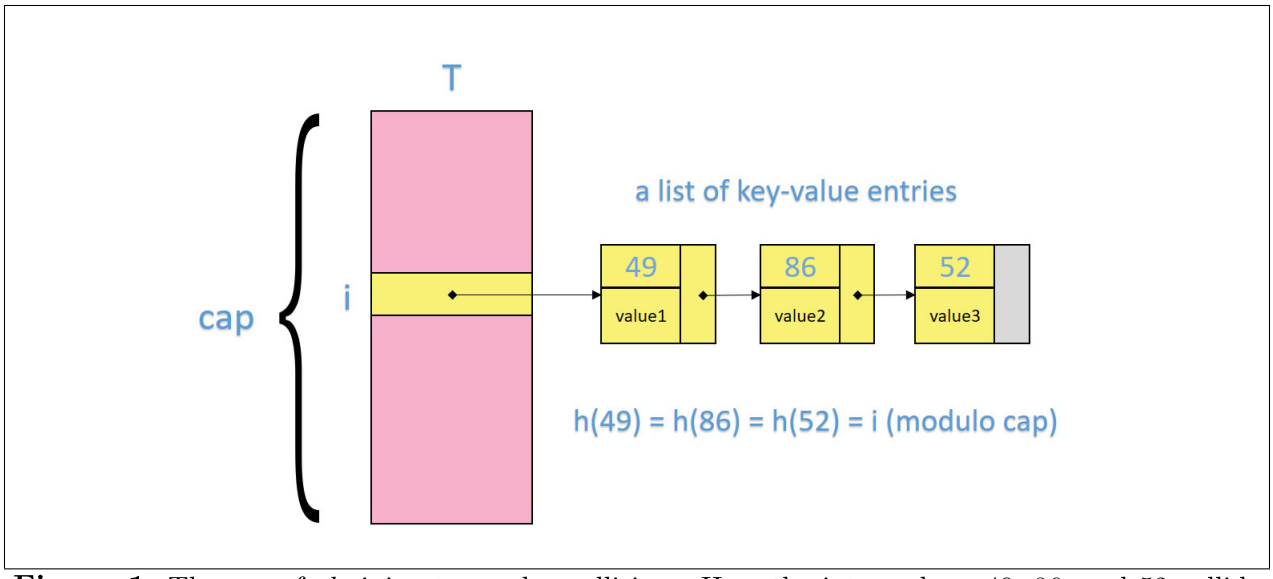

Figure 1: The use of *chaining* to resolve collisions. Here the integer keys 49, 86, and 52 collide because they all map to the same index i (location i in hash table T). To locate the record associated with key 52, a linear search will be needed after the hash function is used to find entry i. Note: the values associated with the keys are not displayed. *(This figure was adapted from slides by Erik Demaine and Charles Leiserson.)*

1.2 Measuring Fitness

In both open addressing and chaining, collisions degrade performance. Many keys mapping to the same location in a hash table result in a linear search. Clearly, good hash functions should minimize the number of collisions. How might the "goodness" or "badness" of a given hash function be measured by looking at the hash table after it has loaded its entries?

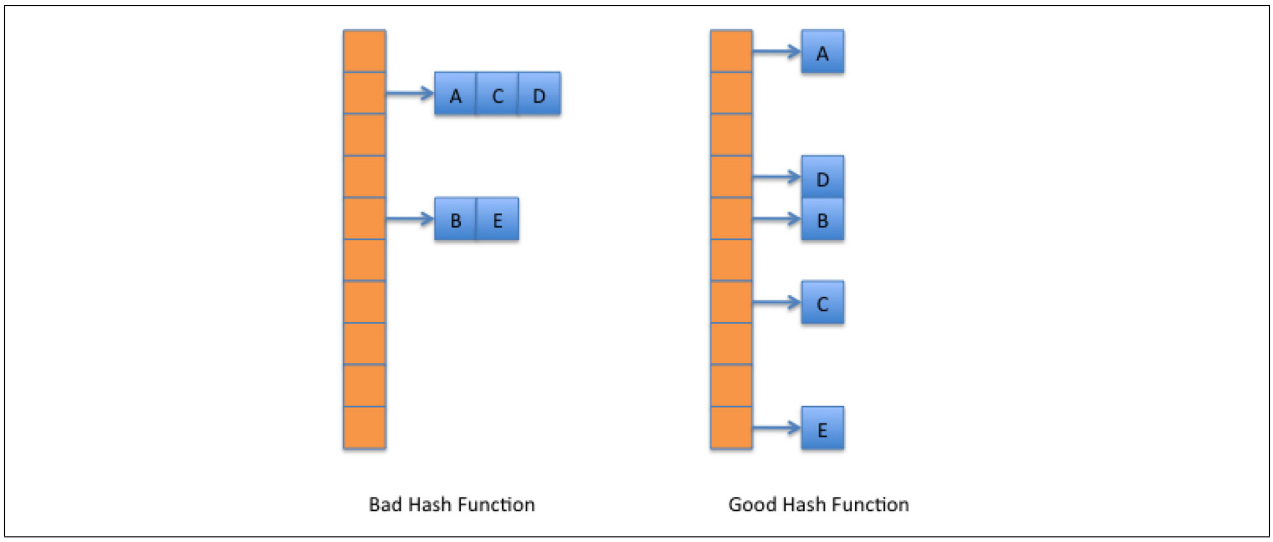

Figure 2: Two hypothetical hash functions have been applied to the same hash table array and the same sequence of keys put in the table.

The example in Figure 2 shows a program which entered 5 keys – A, B, C, D, and E – into a table of length 10 using two different hash functions. The second function provides better distribution, but how can you quantify that based on what you can measure in the tables?

2 Problem-Solving Session

1. Assume a chaining hash table of size 12 and an integer-to-string conversion function that simply adds their ordinal letter values together. Draw what your hash table would look like after putting the following `(key, value)` pairs into it. As an example of the encoding, here is how the first key converts to a number.

   'l','a','d' $\rightarrow 11 + 0 + 3 = 14$

   (a) ("lad", "English")
   (b) ("but", "English")
   (c) ("is", "Latin")
   (d) ("chin", "Dutch")
   (e) ("be", "Greek")
   (f) ("fun", "English")
   (g) ("blab", "German")
   (h) ("zoo", "Greek")

2. Show the order the entries `(key, value)` would be displayed if following the chains from top to bottom and left to right.

3. Write code that implements a hash function that sums up the ASCII values of the characters obtained using `ord` scaled by 31 to the power of the index at which that character occurs in the string, e.g.:

   'l','a','d' $\rightarrow$ `ord('l') + ord('a')*31 + ord('d')*31*31`.

4. Figure 1 illustrates the data structure design you will follow for implementing the chained hash table. Following this design, write pseudo-code for the `add(key,value)` and `remove(key)` operations.