Lesson:

Map One Shot

PW SKILLS

Pre-Requisites

- Basics of C++
- Knowledge of arrays and vectors
- Basic mathematical functions in C++
- Knowledge of C++ STL (Standard Template Library)

List of Concepts Involved

- Why Hashing?
- What is Hashing?
- What are Hash Functions?
- Collisions
- Load Factor
- Problem

Why Hashing?

Say we have 5 students in a class. Some of them have the same names, but we need to uniquely identify them. We can do this by giving unique roll no. to each of them.

1 - Sharron 2 - Larry 3 - John 4 - Cam 5 - Larry

Here, corresponding to each roll no. we have the name of the student. In a similar way we can store to the data of thousands of students. And we can do this by creating a hash table.
Hash table is nothing but a row and column structure that has a unique index and a value corresponding to that index.

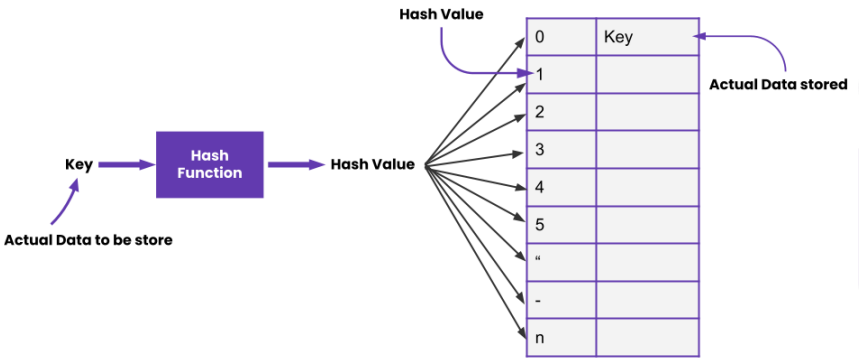
The hash table for these 5 students would look something like this:

Index (Roll No.)	Value (Name of student)
1	Sharron
2	Larry
3	John
4	Cam
5	Larry

But , the things aren't that simple. As our data increases to thousands, it is quite difficult to store all the data in an organized manner. Also there will be operations to be performed on that data like updation and searching. Hence, it is required to store that data in an efficient manner and that is done through Hashing.
Now, let us understand, what hashing actually is?

What is Hashing?

- Hashing is a technique of mapping keys, and values into the hash table by using a hash function.
- Hashing is designed to solve the problems of efficiently finding and storing elements.
- A real-life example of hashing is giving a name or a nickname to a person. It helps in uniquely identifying that person every time.
- It is the process of designing a Hash function that takes an input and gives a consistent output.
- Hash table is a data structure used to store key-value pairs.
- It helps in faster access to elements.
- The efficiency of mapping depends upon the efficiency of the hash function used.



- When we talk about key-value pairs inside the hash table, the key represents the hash value and the value represents the actual data.
- HashMaps are a default implementation of HashTables which takes amortized O(1) time to access the keys and their corresponding values.

What are Hash Functions?

- These are mathematical formulas that create a mapping between key and value.
- A hash function that maps every item into its own unique slot is known as a perfect hash function.
- A good hash function is efficiently computable, uniformly distributes the keys and minimizes collisions.
- We can create our own hash function as well.
- Based on these qualities, we generally use these 4 types of hash functions:
  1. **Division Method:** divide the value k by M and then use the remainder obtained.

     h(K) = k mod M

Example: k = 1276, M = 11
h(1276) = 1276 mod 11 = 0

2. Mid Square Method: Square the value of the key k and extract the middle r digits as the hash value.

h(K) = h(k x k)

k = 60
k x k = 60 x 60 = 3600
h(60) = 60

3. Digit Folding Method: Divide the key-value k into a number of parts i.e. k1, k2, k3,....,kn, where each part has the same number of digits except for the last part that can have lesser digits than the other parts. Add the individual parts. The hash value is obtained by ignoring the last carry if any.

k = k1, k2, k3, k4, ..., kn, s = k1+ k2 + k3 + k4 +....+ kn, h(K)= s

k = 12345
k1 = 12, k2 = 34, k3 = 5
s = k1 + k2 + k3 = 12 + 34 + 5 = 51
h(K) = 51

4. Multiplication Method: Choose a constant value A such that 0 < A < 1. Multiply the key value with A. Extract the fractional part of kA. Multiply the result of the above step by the size of the hash table i.e. M. The resulting hash value is obtained by taking the floor of the result obtained

h(K) = floor (M (kA mod 1))

k = 12345
A = 0.357840
M = 100
h(12345) = floor[100 (12345*0.357840 mod 1)]
= floor[100 (4417.5348 mod 1)]
= floor[100 (0.5348)]
= floor[53.48]
= 53

Collisions

- Sometimes, the same Hash function can result in the same hash value for different keys which is also known as a collision.
- For example, consider the hash function: h(k) = k % M where M = 2. This will give the same hash value for every odd number = 1 and every even number = 0: h(5) = 1, h(7) = 1 and h(6) = 0, h(8) = 0.
- The hashing process generates a small number for a big key, so there is a possibility that two keys could produce the same value.
- This is known as collision and it creates problems in searching, insertion, deletion, and updating of value.
- The situation where the newly inserted key maps to an already occupied hash value must be handled using some collision handling technology.
- There are mainly two methods to handle collision:
  Separate Chaining (Open Hashing): It makes each cell of the hash table point to a linked list of records with the same hash function value. Chaining is simple but requires additional memory outside the table.
  Open Addressing (Closed Hashing): All elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we examine the table slots one by one until the desired element is found or it is clear that the element is not in the table.

It further is done in three ways:

a. **Linear Probing:** Searching the hash table sequentially that starts from the original location of the hash until an empty location is found.
b. **Quadratic Probing:** Taking the original hash index and adding successive values of an arbitrary quadratic polynomial until an open slot is found.
c. **Double Hashing:** It makes use of two hash functions. The first hash function is h1(k) which takes the key and gives out a location on the hash table. But if the new location is not occupied or empty then we can easily place our key.

But in case the location is occupied (collision) we will use the secondary hash-function h2(k) in combination with the first hash-function h1(k) to find the new location on the hash table.

$$h(k, i) = (h1(k) + i * h2(k)) \% n$$

Here, i represents the collision number.

Load Factor

- The load factor of the hash table can be defined as the number of items the hash table contains divided by the size of the hash table.
- Load factor is the decisive parameter that is used when we want to rehash(hashing again) the previous hash function or want to add more elements to the existing hash table.
- It helps us in determining the efficiency of the hash function i.e. it tells whether the hash function which we are using is distributing the keys uniformly or not in the hash table.

Problem: Implement hash table with closed addressing

Approach:

- In this implementation, the HashTable class uses an array of std::list to store the key-value pairs. The hashFunction method calculates the hash index for a given key. The insert method inserts a new pair into the list at the calculated index or updates the value if the key already exists. The remove method searches for the key in the list and removes the corresponding pair if found. The get method retrieves the value associated with a given key from the list.
- In the example main function, we demonstrate the usage of the hash table by inserting some key-value pairs, retrieving values by keys, and removing a key-value pair.

Code: https://pastebin.com/Le9vzje7

Upcoming Class Teasers:

- Map one shot 2