Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

Hashmaps and Its Mathematical Applications in the Software Development World

For the last semi-centennial, humanity has technologically advanced faster and quicker than ever. Terms like encryption, mapping and hashing have become part of the daily life. Programming and software development had become the fastest-growing field and the technology industry is continually thriving. The need for software developers and programmers is increasing, as does the need for software solutions. One of the greatest barriers humans have encountered in the technological modern era is the storage of data. The more common hi-tech has become in our daily life, the need for more storage, availability and accessibility for storage increased. In recent decades, hi-tech products purchased by the average person and the need for more available and accessible information storage have increased significantly. In order to solve the information storage issue, resulting from the exponential growth in the information and the data science fields, software developers founded hashing - a method of storing more information in a highly available and accessible method which allows quick retrieval of storage and information, and will later develop into many additional applications such as encryption, compiler operations, and cryptocurrency.

Hashing is nowadays, one of the most commonly known data structures with which 'by the book', every programmer must be familiar. Hashing is a "dictionary in which keys are mapped to array positions by [certain] hashing functions" (National Institute of Standards and Technology). In terms of software development, "hashing is the process of converting input of

Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

any length into a fixed size [string of text] key, using a mathematical function" (Lisk). The hashing function, based on a specific algorithm, takes information as its input, and hashed it into a specific. Using an array (a bounded, fixed sized list), the function will store the information in its key's indexed reference. The keys' array is limited in its size to a relatively small array, to ensure efficient time of information retrieval from the hash table.

```
int hashing(int toHash);
int main()
{
  int * keys = new int[100]; //create out key values
  int value;
  cout << "insert a number" << endl;
  cin >>  value;
  keys[hashing(value)] = value;
  //verification:
  if(keys[hashing(value)] == value) cout << "success: " << hashing(value) << endl;
  else cout << "failure" << endl;
  return 0; //finishing program
}

int hashing(int toHash)
{
    int key = toHash % 100; // would use the last two digits of the passed value to
    return key;
}
```

The attached code[1] provides an example: the user is requested to enter a number to hash, the input. That value is then stored in the "keys" arrays, based on its key generated from the "hashing" function. The hashing function, based on its algorithm, returns as the number's key the last two digits to serve as an indexing reference, and stores it based on that indexing reference in the array at the key's position (Carroll). Here[2] is the following outcome after inserting a large number.

```
insert a number
1234567890
success: 90
```

[1] Self Programmed in the C++ shell online compiler http://cpp.sh/

[2] *Self Programmed output in the C++ shell online compiler http://cpp.sh/

Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

The user enters '1234567890' as the desired number to store. The hash function generates a key based on the algorithm (modulus 100), which results in 90 as the output and the key. The large value is then stored within the 90th references index in the keys' array, allowing quick (convenient) access to the hashed information. In other words, hashing allows the storage of large-scale information and data in the accessible small-scaled array(Arash Patrow).

There are endless algorithms for hashing, for infinite keys, which makes hashing a unique data structure for applications such as cryptocurrency, and cyber security. A common example of simple hashing algorithms is the hashing of a key using the modulus (remainder) function, dividing a large integer by 100, resulting in the last two digits of the value as its key value ad indexed reference. Another example includes the addition of all digits of the number to server as the indexed reference, algorithms based on values from the ASCII table, as well as the division or multiplication of a number by a specific factor.

In the attached code[3], the hashing algorithm creates a key by adding all the digits of the input value. Every iteration of the loop, the 'total' variable increases by the value of the last digit of

```
int hashing(string toHash)
{
    int total, length = 0;
    length = toHash.length();
    for(int i = 1; i <= length; i++)
    {
        total += (stoi(toHash) % 10);
        toHash = toHash.substr(0, toHash.length() - 1);
        cout << total << endl;
    }
    return total % 100;
}
```

[3] *Self Programmed in the C++ shell online compiler http://cpp.sh/

Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

the ‘toHash’ value, which is referred to as a string so it can be shortened every iteration by 1 character. In case the resulted key is greater than 100, the function will return the last two digits again.

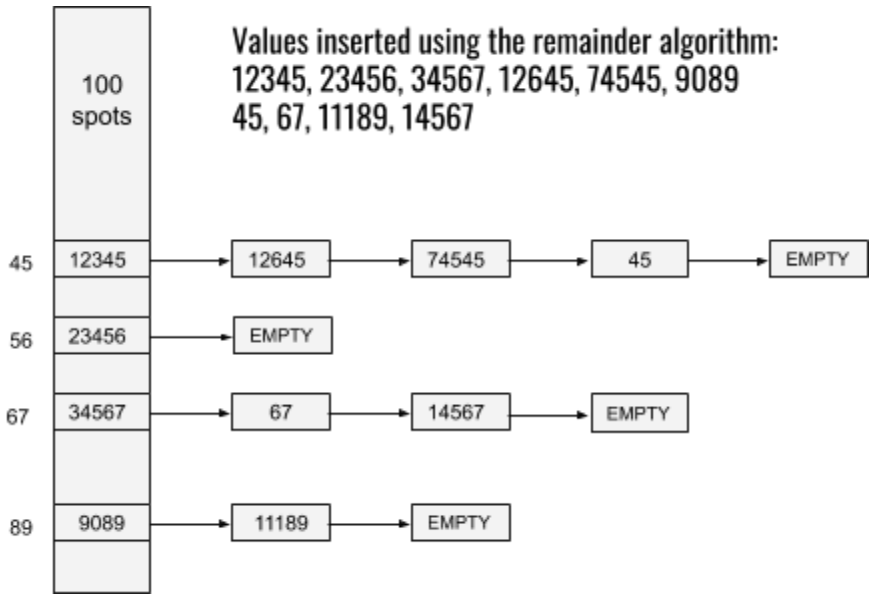
In this approach to hashing, one major problem arises: a situation where two values have the same key could result in the accidental erase and removal of memory. For instance, in the remainder algorithm, there potentially could be two (or more) values with the same last two digits, which would be assigned to the same indexed reference in the array. For example, the values 12345 and 98745 would both be assigned the 45th spot in the array, resulting in a situation where the first value to be hashed to be reassigned. To overcome this issue, hashing has an additional feature which allows the quick accessibility and availability of information, called chaining (Laakmaan McDowell). In chaining, “every time a collision occurs, [where two or more values are assigned to the spot in the bounded array of keys], just store them in a [linked] list” (Laakmaan McDowell). Every entry in the bounded array has a chain started for every array entry with two or more hashed values . The first item added to a specific spot would create that chain, with an additional “node” in the chain to an empty spot, in such a way that the next time the spot in the bounded array of keys will be assigned, that empty spot will be filled with an additional node that holds the hashed information, and is creating the next empty “node” in the chain for a future hashed value to be assigned in the same indexed reference of the bounded array of keys.

Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

As implied in the figure[4], time a collision value is inserted, a new node is created in a chain. Retrieval of information become more convenient as well as the user simply needs the referring index of the position, and then linearly traverse through the chain the find the desired value.



Hashing is therefore considered a discrete data structure. However, the questions remain: how can we design an effective and useful hashing algorithm, which hashing algorithms are considered useful? In order to understand this question, we must first explore some of the applications of hash functions.

One of the most common applications of hashing is encryption. Hash maps, as derived from its name, is the transformation of large-scale data into a small-scaled data represented by a reference key in a bounded structure (array). The key and algorithm are the ones to encrypt the information. Encryption, by its formal definition, is the conversion of information and data into code, objectively to prevent unauthorized access. By hashing information with a key with limited access, only those who hold the key value would be able to decrypt the information into its

[4] Created by the paper's author in Google Drawings

Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

original form. Decryption, the reverse action of encryption, is the retrieval of hashed information based on a given key. (Attach a hashmap of encryption, explain why hashing in encryption in that figureIV)

Decryption is the reverse action of encryption, therefore it is necessary to design an efficient encryption algorithm, a function that will allow retrieval of the information and conversion from its encrypted hashed form into the original data and information. Software developers often find it a mathematical challenge to design an efficient hashing algorithm that will allow decryption of information. In order to design such an algorithm or rather a function, that will be able to encrypt, that function must have an inverse, as decryption is the inverse of the encryption process. By definition, a function only has an inverse function of it is a bijection; by definition of a bijection (Morris) relation, a function is considered to be a bijection only if it is a one-to-one and onto function.

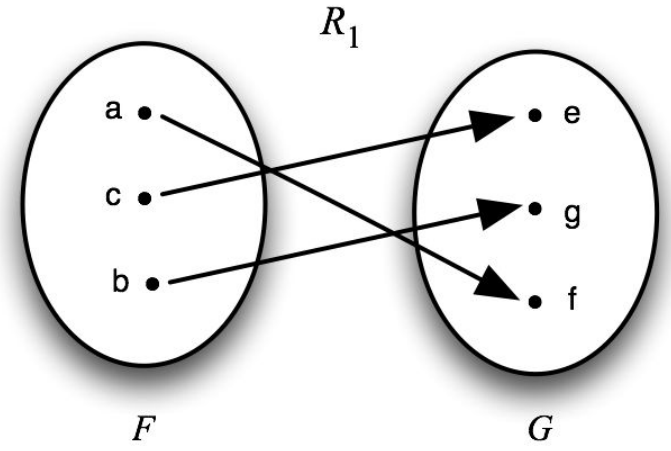
For a function to be considered 'one-to-one' (also known as an injection), every value in the domain (any input to the function) must have a corresponding unique value in the codomain (the output of the function). By its discrete mathematical definition: a function in 'one-to-one' if $\forall a, b \in A \; If \; f(a) = f(b), \; then \; a = b$, which in literal is presented as: for any two elements $a \; and \; b$ in the domain A, if the values in codomain (the output) of $a \; and \; b$ (those are $f(a) \; and \; f(b)$, respectively) will equal to each other, then they must be the same value in the domain (Lecture 18: 'One to One and Onto functions'). In other words, every element in the

Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

domain would map only into one unique value in the codomain. In the figure[5], it is seen that there do not exist two elements in the codomain that lead into the same value in the domain (image of the function), hence the function is 1:1 (one-to-one).



Comparatively, for any relation or function to be considered an 'onto', its domain (the bounded structure of hashed information) must have every single element mapped by an element from the domain. In other words, as long as no element in the codomain is left out and is mapped to by some element from the domain, the function is considered a surjective function ('onto'). Mathematically discretely we define $\forall\ b\ \in\ B,\ \exists\ a\ \in\ A\ such\ that\ f\ (a)\ =\ b$. Literally comparing, the statement could be written as: "for every value in the codomain, there exists a value in the domain such that the image [map] of the domain value equals to the value in the codomain" (Lecture 18: 'One to One and Onto functions'). In other words, for every value in the codomain, there is a value in the domain that is mapping to it.

As explained, an effective hash function is one that includes both the ability to encrypt as well as decrypt, what is a bijection function. Theoretically, a hash function cannot be a bijection, since it has an infinite domain mapped into a bounded limited domain. However, in some

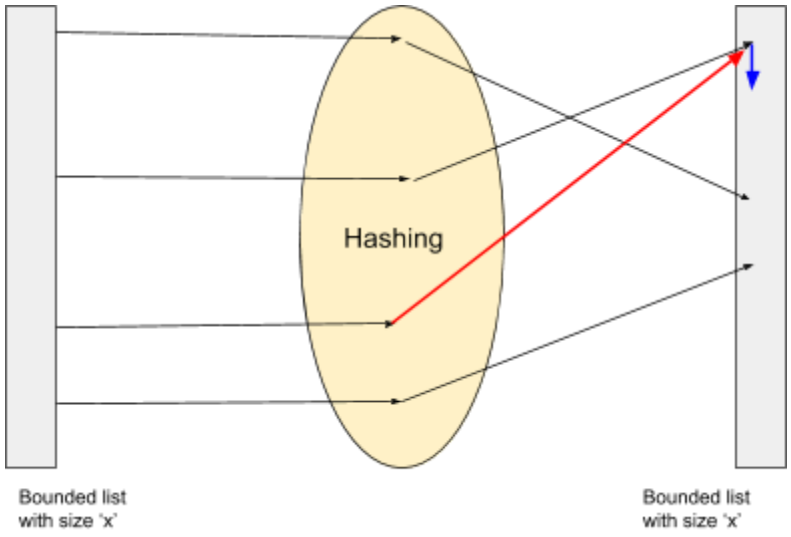
[5] Created by the paper's author in Google Drawings

Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

situations, where the domain is finite and the codomain (output) of the function does not include any collisions of hashed information (as examined previously in the paper), the function could be considered a bijective encryption function. To get around this arising issue, developers have created an encryption key which will populate a large bounded array, equivalent in size to the list structure holding the values to be hashed, but in a case of a collision, instead of creating an additional node in an endless chain, the next item (by the referenced index) that is not populated in the codomain (output array) will be assigned with the hashed value. As referred from the figure[6], both structures of encrypted and original information hold the same amount of values. As opposed to the 'endless chain' hashing, information retrieval is not nearly as quick on the large scale, but mathematically, it allows to create an inverse function for decryption purposes. In the figure, the first value is assigned to its specific indexed reference position, based on the generated hashing key. The second item, as well, generated the same key, but since the array of encrypted values already holds value at that position, the



[6] Created by the paper's author in Google Drawings

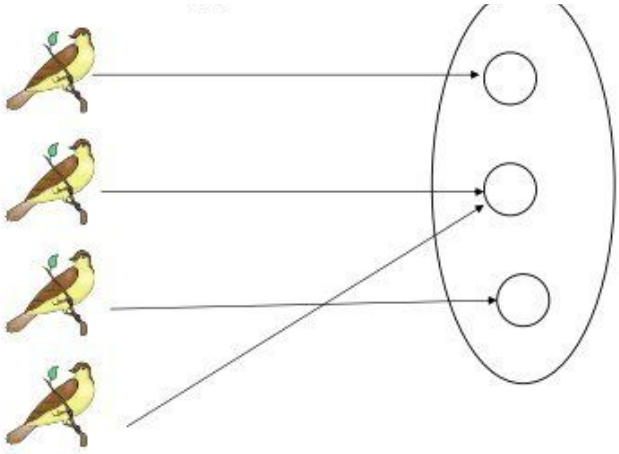
Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

currently hashed item (the second item according to the figure), will be inserted into the indexed position after the one generated as its key position (Caroll). In a situation where the hashing function sends elements from an array (list) of items of a bigger size than then array of the codomain, mathematically, by the "pigeonhole principle", the function would not be 1:1.

According to the pigeonhole principle: suppose F is a function from X to Y, where x and y are finite sets. If $|X| > |Y|$, then the function F is not 1:1. Since F is said to be a function, every element from its codomain is sent out to its domain. Since the size of the codomain is smaller than the size of the domain, some values will be overwritten when sent to values that already preoccupied a position in the array of the codomain (Morris). The Pigeonhole principle is best explained by the analogy of pigeons: if more pigeons were to be sent to a box where there are not enough pigeonholes, some pigeons[7] will "leave" and will not be tracked, lost from records. Similarly, in hash functions, the information will be overridden and data will be lost. Mathematically, then, hash function cannot be 1:1. In practice, however, software developers do not see the collision as the violation or dissatisfaction



[7] Created by the paper's author in Google Drawings

of the one to one condition since no overriding of information will occur. Hence hash maps in practice of programming could be designed to be bijective, and be built to have decryption maps, which will allow a group of collided keys in the bounded structure to be mapped into their possible outcomes in the domain.

All in all, there is no doubt in saying hashmaps are a revolutionary tool in the industry of software development, specifically data science, information storage, cryptocurrency, and even cybersecurity; hashing is a tool used widely in the software development world. As examined, a common use of the hash function is encryption of information, allowing data accessibility to authorized users only. However, in order for encryption to be available for users, there must be a decryption function, reversing the encryption process, or in other words, an inverse function of the hashing relation. Mathematically, in order for a function to have an inverse, it must be considered a bijection, or by the definition of a bijective function, it must be considered injective (one-to-one) and surjective (onto function). As examined, although in mathematical theory hash functions could not be considered bijective and therefore not have an inverse (a decryption possibility), in the practice of software development although collisions of information occur, the injective property of a bijective function is not violated, hence it is possible to design a bijective encryption hash function. It is hence concluded and seen that hashing has significantly revolutionized the software engineering world, creating unique applications that could solve issues as the world of technology continues to exponentially grow.

Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

Works Cited

"Basics of Hash Tables" *Hacker Earth.* https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/

Carroll, Hyrum. "Hashing: Efficiency" *YouTube,* Hyrum Carroll, 21 Jan, 2015. https://www.youtube.com/watch?v=Xigaybhsg6U

"Frege's Theorem and Foundations for Arithmetic". Stanford Encyclopedia of Philosophy, 26 Jun, 2018. https://plato.stanford.edu/entries/frege-theorem/

"General Purpose Hash Function Algorithm."Arash Patrow. https://www.partow.net/programming/hashfunctions/

"Hashing Algorithms" *JScrambler*, JScrambler. 25 Oct, 2018. https://blog.jscrambler.com/hashing-algorithms/

"Hash Table", National Institutes of Standards and Technology. https://xlinux.nist.gov/dads/HTML/hashtab.html

Laakmaan McDowell, Gayle. "Data Structures: Hash Tables." *YouTube*, HackerRank, 27 Sep, 2016. https://www.youtube.com/watch?v=shs0KM3wKv8

"Lecture 18: 'One to One and Onto Function'", University of Colorado Boulder, Department of Computer Science. https://www.cs.colorado.edu/~srirams/courses/csci2824-spr14/functionTypes-18.html

Morris, Jason. *Discrete Math Course - Course Notes*, CS17, Las Positas College. 2019.

Shahaf Dan

Math 10 / Computer Science 17 Honors Project

Professor Morris

"What is Hashing? Hash Functions Explained Simply" *YouTube,* Lisk, 08 Aug, 2018. https://www.youtube.com/watch?v=2BldESGZKB8&t=3s