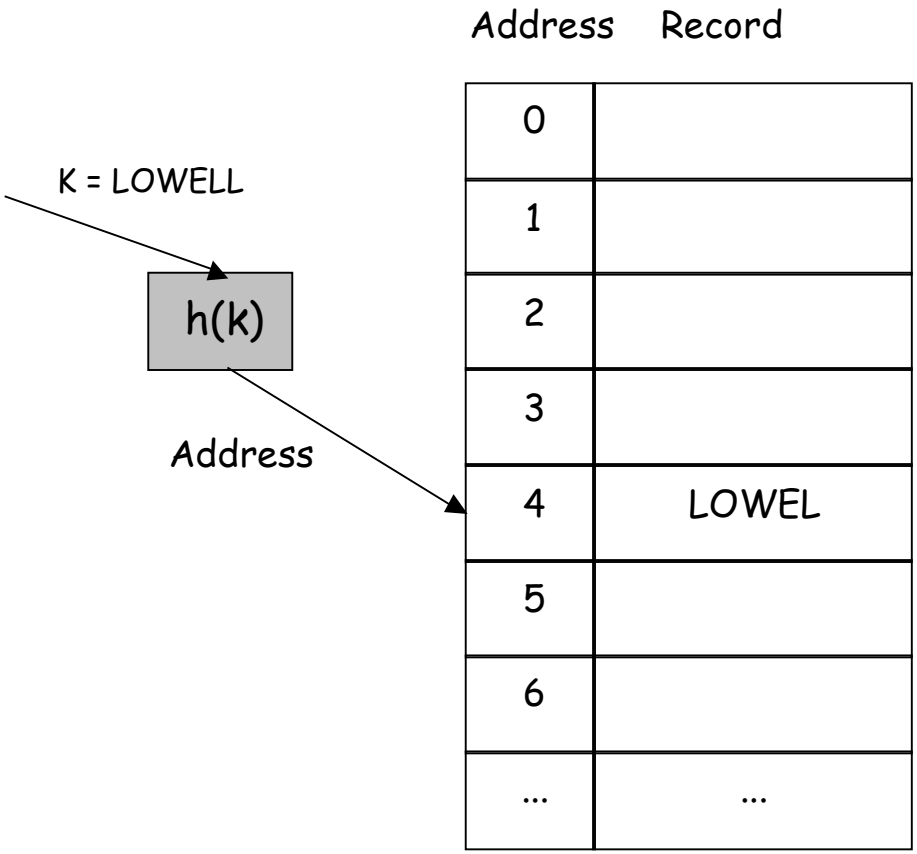
Hashing—The quest for O(1) access

- hash function produces the address of the record given the key and
- it does it in the same amount of time no matter what the key and
- each time it produces the same address

The addresses generated appear to be random, that is there is no obvious connection between the key and the address

Two different keys may hash to the same address this is called a *collision*

Example: Multiply the ascii values of the first two characters of the name



There are 1000 addresses in the file so we the hash function is:

Ord(L) * Ord(O) mod 1000

= 76 * 79 mod 1000

= 6004 mod 1000

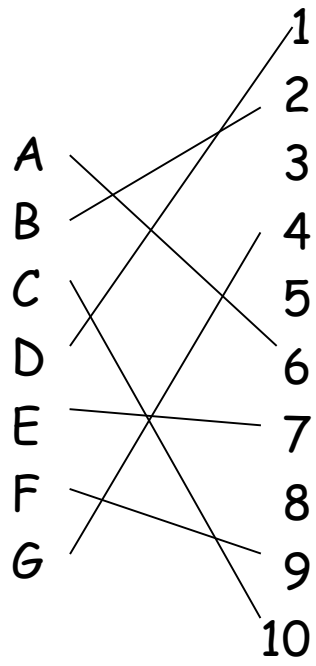
= 4

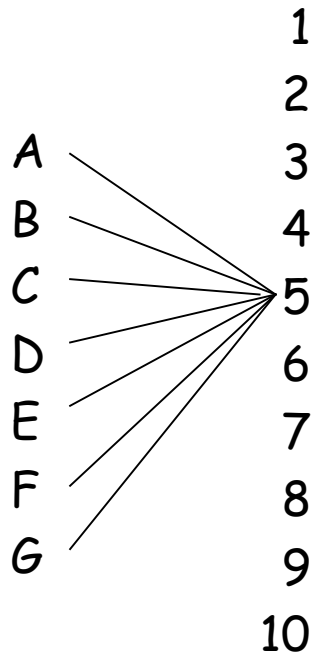
This is nice but what happens to Oliver and Olivia and Oliphant?

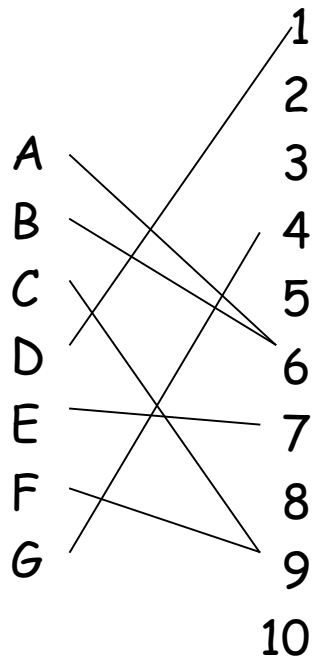
Collisions: Oliver and Olivia and Oliphant all hash to 4 as does Lorelle.

- When this happens we call these *synonyms* and we say that we have a *collision*
- So let's find a hashing algorithm that doesn't produce collisions
- But research shows that of all the possible hashing algorithms only one in $10^{120,000}$ avoid collisions altogether when storing 4000 records among 5000 addresses.
- So we need another solution because it isn't reasonable to assume that we can determine the algorithm

Best Case		Worst Case		Acceptable	
Record	Address	Record	Address	Record	Address







The goal is to develop hash function that gives us at least an acceptable distribution if we can't find one that will give us the best distribution.

Strategies for reducing the number of Collisions

- *Spread out the records:* Find an algorithm that distributes the records randomly among the available addresses
- *Use extra memory:* The larger the address space, the easier it is to find a hashing algorithm that avoids collisions
- *Put more than one record in a single address:* If we can put more than on record at an address we can reduce the cost of collisions. Multi-record addresses are sometimes called *buckets.*

A simple hashing algorithm

1. Represent the key in numerical form: OLIVIA followed by 6 blanks

   O = 79 L = 76 I = 73 V = 86 I = 73 A = 65 blank = 32

2. Folding and adding means divide the string of ASCII values into groups of two

   7976 + 7386 + 7365 + 3232 + 3232 + 3232 = 32423

   If this number is too large to fit as an integer in our system we can choose a suitable large odd prime larger than the largest expected addend and smaller than the largest allowed integer value. The author suggests 19,937

   7976 + 7386 mod 19,937 = 15,362

   15362 + 7365 mod 19,937 = 2,790

   2790 + 3232 mod 19,937 = 6,022

$$6022 + 3232 \bmod 19{,}937 = 9{,}254$$

$$9254 + 3232 \bmod 19{,}937 = 12{,}484$$

3. Divide by the Size of the Address Space

   If there are N possible addresses in the file and S is the sum from step 2, the address from the hash function is

$$A = S \bmod N$$

   So for our example

$$A = 12484 \bmod 1000$$

   give 484 as the address if we want addresses from 0 through 999

Making this better:

- Examine Keys for a pattern: If this pattern naturally spreads the values use it. Can you think of something that would do this?
- Fold parts of the key: extract digits from the key and add them together in this way you destroy the original pattern but you may get a better distribution
- Divide the key by a number: Use a prime number or a number that has large factors (i.e. > 19)
- Square the key and take the middle: This works nicely but you have to be able to do arithmetic with large numbers.
- Radix transformation: Convert to another base then use that number modulo the number of addresses. This works better than square and extract the middle