[Back to Resources]

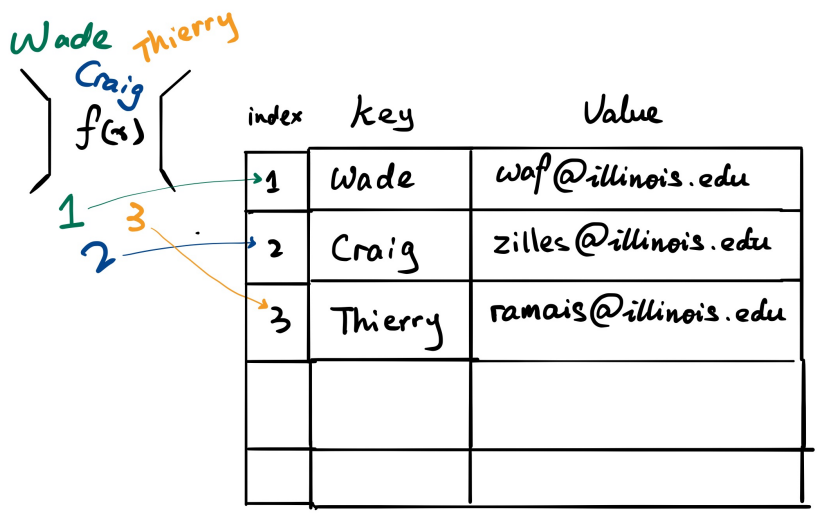
# Hash Table

by Siping Meng

## Introduction

Hashing has many uses. In 225, we talk about how hashing is used to store data. To perform hashing, we need two things

- *Hash function*: Converts a key into a smaller number and uses that number as an index in the table.
- *Hash table*: An array that stores values according to the index.



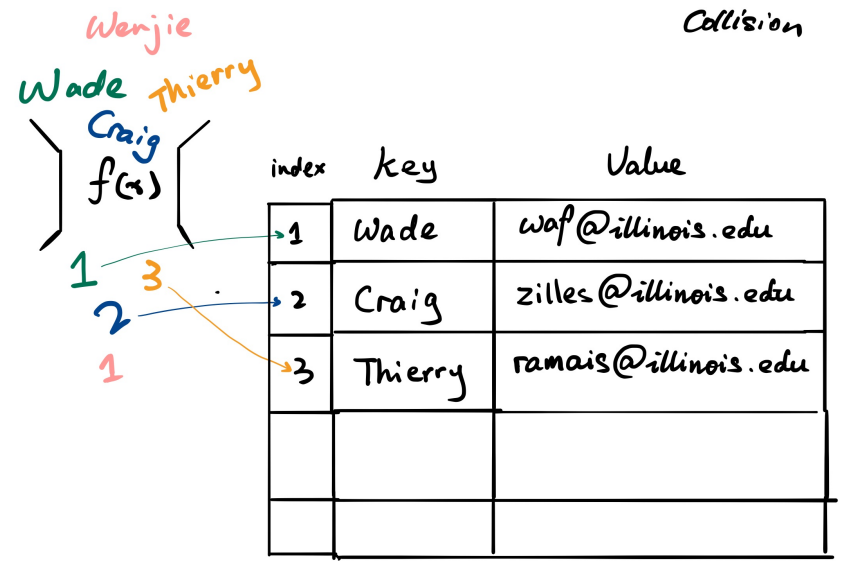

A hashing of CS225 professors' name with their email address

## Hash Collisions

Assume the previous $\mathrm{f}(\mathrm{x})$ works like the following:

- if the input start with *w*, *return 1*
- if the input start with *c*, *return 2*
- if the input start with *t*, *return 3*

Now we want to insert a TA to our contact table, say *key* : **Wenjie** and *value* : **wzhu26@illinois.edu**. The hash function will return *1*.

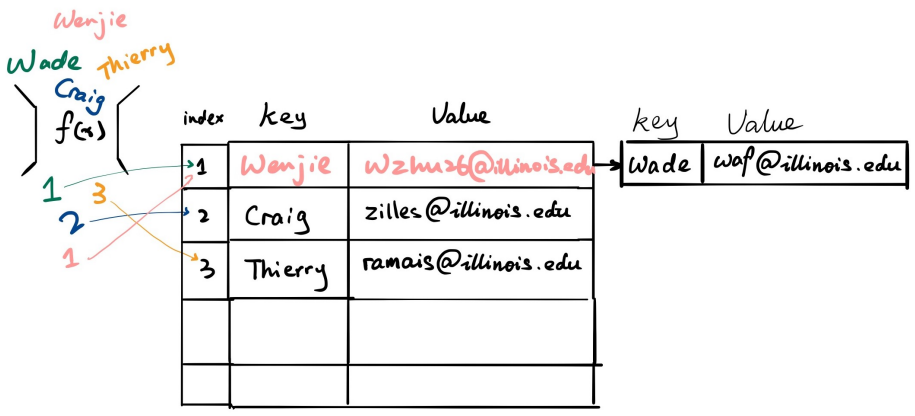



The hashes of Wade and Wenjie collide

## Two types of hash storing

### Separate Chaining

What we can do is to let the cells of our hash table point to several linked lists of values which have the same output from hash function.



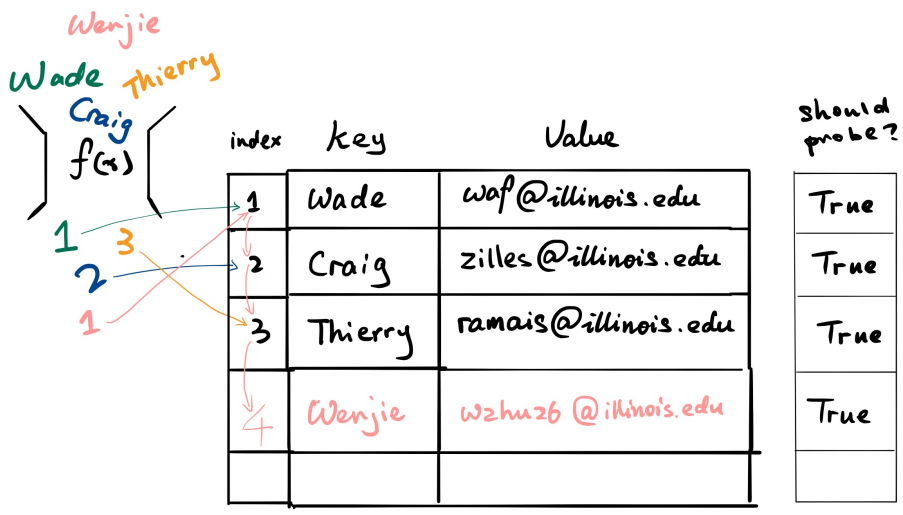

Separate chaining hashing of CS225 staff's names with their email addresses

## Linear Probing

Something else we can do is just fill the table, without creating any additional memory.

When we see a collision, we will try to linearly probe for next space. In our case, we should start to check slot *1*, but oh *waf* is there, so let's probe to slot *2*. However, professor *Craig* has already taken that space so we need to probe again to slot *3* and found lovely *Thierry* there. Luckily, slot *#4* is empty so we can put our TA's info there.

When searching for an element, we examine table slots one by one until the desired element is found , or it is clear that the element does not exist in the table.




Linear probing hashing of CS225 staff's names with their email addresses