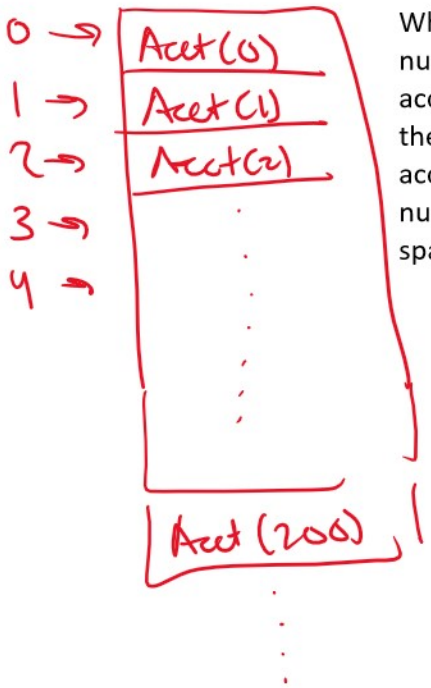
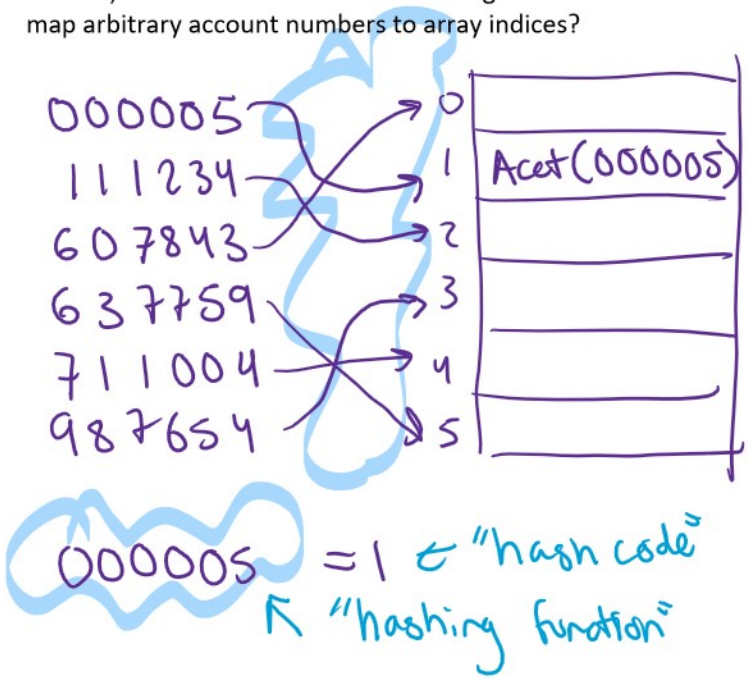
Constant time for HashMaps

Wednesday, October 12, 2022 2:09 PM



When we had a sequential block of account numbers, we were able to get constant-time access in our hashtable by going to exactly the array index that corresponded to that account number, but this idea of account numbers had some downsides (lots of empty space if accounts are closed, for one).

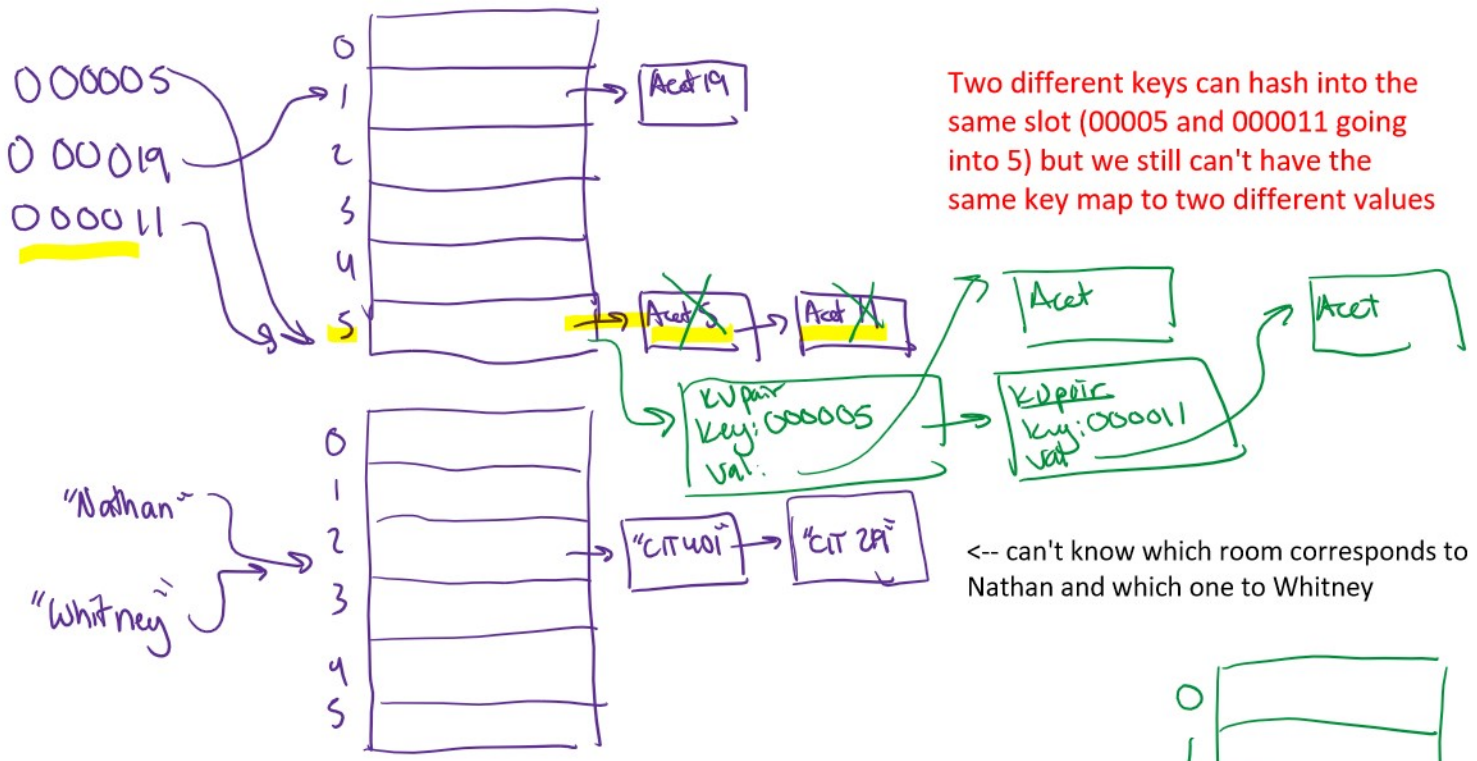
Instead, what if we had some sort of magic function that let us map arbitrary account numbers to array indices?



Let's say our hashing function is
acctNum % (arraySize) <- gives us back a number between 0 and arraySize - 1 (inclusive)
Ex.
000005 % 6 = 5
000019 % 6 = 1
000011 % 6 = 5

Resolving collisions using LinkedLists

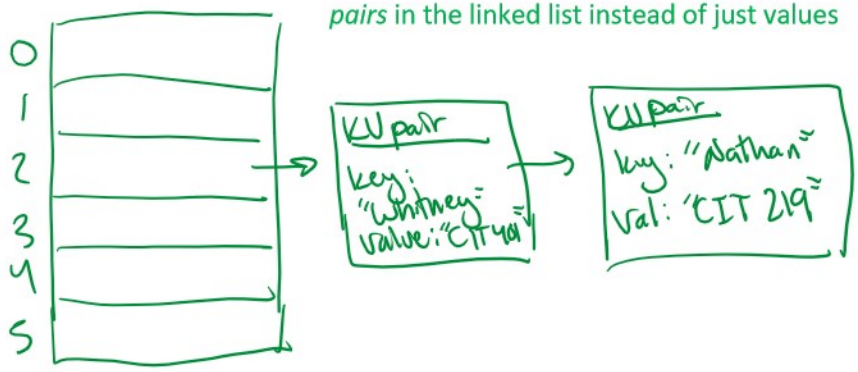
Wednesday, October 12, 2022 3:29 PM



Two different keys can hash into the same slot (00005 and 000011 going into 5) but we still can't have the same key map to two different values

<-- can't know which room corresponds to Nathan and which one to Whitney

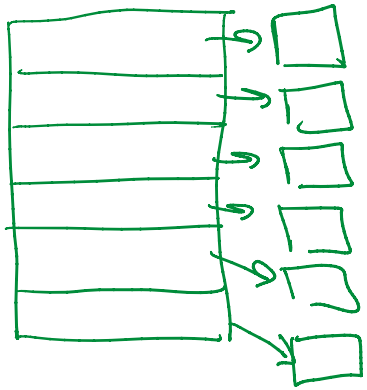
Deal with this problem by storing key-value *pairs* in the linked list instead of just values



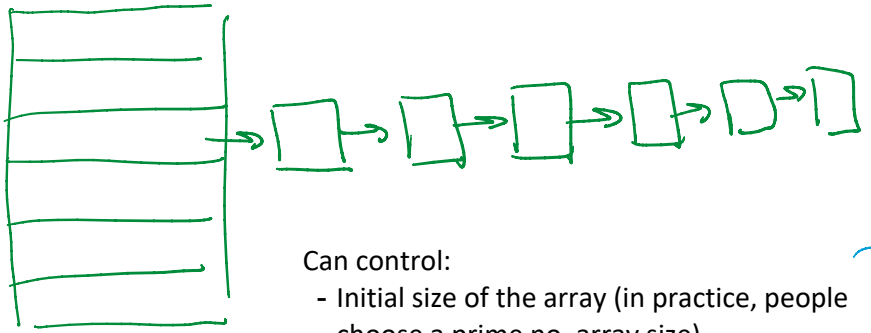
Runtime

Wednesday, October 12, 2022 3:32 PM

Optimistic:



pessimistic:



Can control:

- Initial size of the array (in practice, people choose a prime no. array size)
- If/when you resize the array (resize the array when it's about 75% full)
- Hashing function (lots of math)

Making the right choices here leads to the probability of constant lookup/insertion time on average (the Java HashMap documentation talks about why its default values result in this!)

"n" → 66 * 3

"a" → 52 * 5

"f" → * 7

"h" → * 11

"a" → * 13

"n" → * 17

+

(large number) % arraysize

Bonus: hashing strings