3 Styles of Hashing

There are 3 variations on the idea of hashing that are in common use:

A. *Linear Open Adressing* is the style we have been discussing. We use the key's hash value to find a starting index in the table, and from there we do a linear search to find either the key we are looking for or an empty entry in the table. If h is the hash value and C is the table size this looks at entries with indices h, (h+1)%C, (h+2)%C, (h+3)%C, etc
B. *Quadratic Open Addressing* is identical to linear addressing, except that it looks at indices h, $(h+1^2)\%C$, $(h+2^2)\%C$, $(h+3^2)\%C$. etc.
C. *Chained Hashing* puts a linked list at every entry of the table. All of the entries with the same hash value are store in that value's linked list.

The *load factor* λ of a hash table is the fraction of the table that is full: the number of keys we have in the table divided by the size of the table. An empty table has load factor 0; a full table has load factor 1.

Linear Open Addressing

A proof is beyond the mathematical level of this course, but for linear open adressing it can be shown that if the hash function spreads the keys out, so that every index is equally likely to be the hash value of a key, then the expected number of comparisons we have to make before we get to an open square of the table is

$$\frac{1 + \frac{1}{(1-\lambda)^2}}{2}$$

For example, if λ is 0.5 this comes to 2.5

Think about what this means.

If we manage our table so that the load factor is never more than 0.5, then regardless of how much data we have an insertion or lookup in the table takes on average 2.5 comparisons. This is a constant; insertions and lookups are independent of how much data we have. They run in time O(1)!!

Of course there is a price to pay for this: we have to start with a table large enough to keep our load factor small. We are trading space for time.   On the next slide you can see what happens if the loadfactor becomes too large.

Here is a table of the expected number of probes needed to find an open spot in a hash table with load factor λ, assuming linear probing:

λ	Number of Probes
0.1	1.11
0.2	1.28
0.3	1.52
0.4	1.89
0.5	2.50
0.6	3.62
0.7	6.06
0.8	13.00
0.9	50.5

As you can see, the performance degrades quickly once the load factor rises above 0.5. The only alternative is to use a larger table, but remember that the hash values depend on the table size. This means that if we change the table size we have to re-insert every entry of the table. That is painful. With hashing there is a big incentive to choose a sufficiently large table at the start

Quadratic Addressing

Quadratic open addressing has the same problems as linear open addressing, plus one more. With linear addressing, we are sure that the indexes h, (h+1)%C, (h+2)%C will eventually reach every square of the table, This is not true for quadratic addressing in general, but you can show this:

If the table size is prime and if the table is no more than half full, then quadratic probing will find an empty location for any insertion.

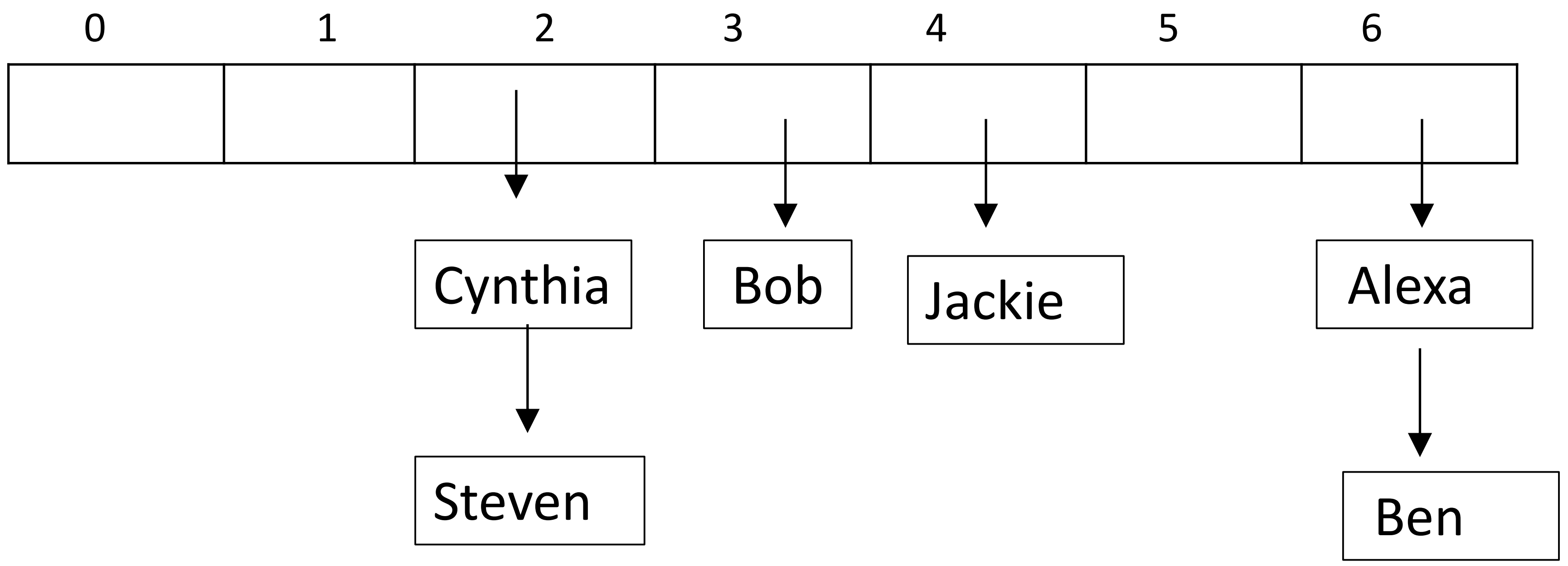
So why does anyone bother with quadratic addressing? In practice it tends to work better.   Most hash functions are not perfect, but leave the keys in clusters in a hash table. These clusters are made worse by the way linear addressing handles collisions. Quadratic addressing takes larger and larger jumps to get out of the current clusters. There is no complete mathematical analysis of the expected number of probes needed for quadratic addressing, but in both practice and simulations it runs faster than linear addressing. It is a trivial matter to switch between thee two open addressing styles, so if you ever implement your own hash tables you can easily see which one works better for your data.

Chained Hashing

Let’s revisit an example where we add some people to a hash table of size 7.

Name	hashValue
Ben	6
Bob	3
Steven	2
Cynthia	2
Alexa	6
Jackie	4

Here is a picture of a chained hash table for this data.   Each entry of the table is a linked list.   To add an item to the table we add it to the end of its hash value's list. To find an item we do a linear search through its hash value's list.



Note that the average size of a list is the sum of the sizes divided by the number of lists, which is the same as the number of keys divide by the size of the table, which is just the load factor. For chained hash tables the load factor can be larger than one, since each entry of the table is a list of keys and values. For unsuccessful searches the average number of comparisons is just the load factor. For successful searches it is half of the average length of a list, or half of the load factor.

Chained hash tables are not as sensitive to the load factor as tables with either style of open addressing. On the other hand, chained tables are very sensitive to the quality of the hash function. A function that hashes many keys to the same index will give us bad performance regardless of the load factor.

People have different preferences, but one reasonable approach to hashing is to use either style of open addressing and being careful to start with a fairly large table so the load factor remains small.