Analysis of Hashing

The *load factor* λ of a hash table is the fraction of the table that is full. An empty table has load factor 0; a full one load factor 1. The previous result says that if the load factor of a table using quadratic probing is no more than 0.5 then quadratic probing is guaranteed to find a slot for any inserted item.

If the load factor grows to more than 0.5 we need to expand the table. Note that the hash function depends on the table size; expanding the table means that we need to rehash all of the data in it.

In probability theory the *expected value* of an event is the sum of all of its possible values, each multiplied times the probability it occurs. For example, if you bet someone \$5 on an event that has probabilty p of occurring, your bet's expected value is
$5*p + (-5)*(1-p) = 10p-5$. If p is 0.1, the expected value is -4, meaning that on average you will lose \$4 each time you make such a bet. That is how casinos make their money.

If you assume that the data in a hash table is randomly distributed, then the probability that any particular cell is occupied is λ and the probability it is unoccupied is $1-\lambda$. The probability that the first location a linear probe tests is unoccupied is $1-\lambda$. The probability that the first is occupied and the second is free is $\lambda*(1-\lambda)$. The probability that the first two are occupied and the third is free is $\lambda*\lambda*(1-\lambda)$. Altogether the expected number of probes we need under the assumption of complete randomness is

$$ENP = 1*(1-\lambda) + 2*\lambda*(1-\lambda) + 3*\lambda*\lambda*(1-\lambda) + \ldots.$$

$ENP = 1*(1-\lambda) + 2*\lambda*(1-\lambda) + 3*\lambda*\lambda*(1-\lambda) + ...$

$= (1-\lambda) *[1 + 2*\lambda + 3*\lambda^2 + 4*\lambda^3 + ...]$

Let S be the portion of this in square brackets:

$S = 1 + 2*\lambda + 3*\lambda^2 + 4*\lambda^3 + ...$

Then $\lambda*S = \lambda + 2*\lambda^2 + 3*\lambda^3 + 4*\lambda^4 + ...$

If we subtract these we get

$S - \lambda*S = 1 + \lambda + \lambda^2 + \lambda^3 + ...$

This is a geometric series; it sums to $1/(1-\lambda)$

So $S - \lambda*S = 1/(1-\lambda)$

$S(1-\lambda) = 1/(1-\lambda)$

$S = 1/(1-\lambda)^2$

$ENP = (1-\lambda) *S = 1/(1-\lambda)$

Unfortunately, our assumption of complete randomness is at odds with either linear or quadratic probing. If cell n is occupied, the probability that cell n+1 is also occupied is the sum of the probability that some object in the table has hashed to n+1, plus the probability that a second object has hashed to n. Since the occupancy probabilities are higher than we estimated, the expected number of probes to find an open spot is also higher. A complete analysis is beyond what we can do here, but it has been shown that the expected number of probes on an insertion with linear probing is

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

The expected number of probes is

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

For example, if λ is 0.5 this comes to 2.5 If λ is 0.7 it is just over 6. If λ increases to 0.9 the expected number of probes increases to 50.5 On the other hand, if λ decreases from 0.5 to 0.3, the expected number of probes decreases by only 1.

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

Clicker Q: Think about that table. What is a good load factor to aim for?

A. Between 0.1 and 0.2
B. Between 0.4 and 0.5
C. Between 0.6 and 0.7
D. Over 0.8

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

Note that there is only slight increase in the expected number of probes as λ increases from 0.1 to 0.4. There is little reward for keeping the hash table mostly empty. However, as the load factor approaches 1 the number of probes rises very rapidly. There is a significant penalty for letting the load factor rise much above 0.5.

The moral of this is that while there is little payoff for having a table that is nearly empty, the cost of having one that is nearly full is high. If the load factor becomes much more than 0.5 we want to increase the size of the table, probably doubling it.

Note that changing the table size requires a change in the hash function, so it forces us to rehash the entire table.

No one has yet developed a similar analysis of quadratic probing (Honors Project, anyone??) . Simulations show that quadratic probing reduces clustering and generally involves fewer steps than linear probing. Since it requires very little extra work to achieve this savings, most people prefer quadratic probing over linear probing.

We have been discussing the open addressing methods for collision resolution. There is another method called *chaining*.

With chaining we make the actual entries of the hash table linked lists. When an item hashes to index n, we add it to the linked list stored at index n. To do a lookup for an item, we go to the index to which it hashes and do a linear search on the corresponding linked list.

The load factor of a hash table with chaining is still the number of entries in the table divided by the size of the array. Note that this can be greater than 1. If every index of the table holds a list of two elements, the load factor is 2.0.

Suppose we have a chained table of size 100 and it contains 150 elements. Each of these elements will be in one of the 100 lists. If we sum the sizes of the lists we get all 150 items. The *average* size of the lists is 150/100, or 1.5. This is exactly the load factor. In other words, the average length of the lists with a chained hash table is just λ.

To search for an element that happens to be in the table, we go to its list (whose index is the hash value of the element) and do a linear search. If the list has n elements, this search might require 1, or 2 or 3 or ... or n probes; all of these cases are equally likely. So the average number of probes is (1+2+3+...+n)/n = (n)(n+1)/2n = (n+1)/2. Here the average size of the list n is our load factor λ. **So the average number of probes for a successful search in a chained hash table is (λ+1)/2.**

In an unsuccessful search we have to look through the full list, so we do λ probes.

Note that if $\lambda < 1$ then $(\lambda+1)/2$ is larger than λ, so if a chained table has load factor less than 1 then unsuccessful searches are faster on average than successful ones.

One advantage of chained hash tables is that they are not nearly as sensitive to the load factor as open addressing tables are. With quadratic open addressing, insertions may fail if the load factor becomes even slightly larger than 0.5. This does not happen with linear open addressing unless the table is completely full, but even there searches can become very expensive if the load factor becomes much more than 0.7. With chained tables the difference between searching a table with λ = 2.0 and one with λ=3.0 is just one extra probe.