September 12, 2015

Hash Table Analysis

Assume that we have a hash table structured as a vector of lists, resolving collisions by sequential search of buckets, as in the generic hash table template `HashTable<K,D,H>` distributed in the file `hashtbl.h`. Let n denote the number of items in the table [table size] and b denote the number of buckets (or slots) in the table implementation. At this point we make no assumptions about the hash function.

Proposition 1. Denote by $e(k)$ the expected number of buckets of size k and by $p(k)$ the probability a given bucket has exactly k items. Then

$$e(k) = bp(k).$$

Proof. $e(k)$ = expected value = (frequency) $\times$ (probability) = $b \times p(k)$. $\square$

Proposition 2. Let $p(k)$ denote the probability a given bucket has exactly k items. Then the expected bucket size is

$$\sum_{k=0}^{n} kp(k) = \frac{n}{b}.$$

Proof. The expected value of bucket size is the sum of (the expected value of size of buckets of size k) $\times$ (probability a bucket has size k), where

$$\begin{aligned} \text{expected value of size of buckets of size } k &= k & &\text{duh} \\ \text{probability a bucket has size } k &= p(k) & &\text{definition of } p(k) \end{aligned}$$

Substituting, we get the formula on the left for expected value of bucket size. On the other hand, there are n items in b buckets, so the average bucket size is n/b, the formula on the right. $\square$

Proposition 3. Let $b(k)$ denote the actual number of buckets of size k. Then the size of the table is

$$n = \sum_{k \geq 0} kb(k).$$

Proof. The number of items in buckets of size k is $k \times b(k)$. Since each item is in at most one bucket, the sum on the right does not overcount the number of items. Since each item is in at least one bucket, the sum on the right does not undercount either. So the sum must be the number n of items. $\square$

Remark. The smallest safe upper index for the sum in the formula above is the maximum bucket size, because there are no items in buckets with size greater than the maximum.

We could also say that n is the smallest safe upper index for the sum, if we want a data independent answer. n is attained in the most dismal case where all items end up in the same bucket, which would make $b(i) = 0$ for $i < n$ and $b(n) = 1$, reducing the sum to $n = n \times b(n)$.

Assume now that we have assigned keys to buckets randomly, so that each key has a uniformly likely probability of being assigned to a bucket. (This is called "simple uniform hashing" in the textbook - see page 259.)

Proposition 4. The probability that a given bucket is empty is

$$p(0) = \left(\frac{b-1}{b}\right)^n .$$

Proof. The probability that one particular item is not in a given bucket is the probability it is in some other bucket. There are b buckets in total, so there are $b-1$ other buckets. Thus the probability that one particular item is not in a given bucket is $q = (b-1)/b$. The probability that two particular items are both not in that bucket is the product $q \times q = q^2$. Reasoning by induction, it follows that the probability that k particular items are not in a given bucket is $q^{k-1} \times q = q^k$. Finally, then, the probability that none of the n items is in the given bucket is q^n. □

Proposition 5. Let $0 \le k \le n$. The probability that a given bucket has size k is

$$p(k) = \binom{n}{k}\left(\frac{1}{b}\right)^k \left(\frac{b-1}{b}\right)^{n-k} .$$

Proof. This is a refinement of the argument above. Note that the probability that an item is in a given bucket is $p = 1/b$ and the probability it is not in that bucket is $q = 1 - p = (b-1)/b$. If we have a particular choice of k items, the probability that exactly these items are in a given bucket is $p^k q^{n-k}$, because those k items are each in the bucket, accounting for the p^k factor, and the remaining $n-k$ items are *not* in the bucket, accounting for the q^{n-k} factor. There are $\binom{n}{k}$ independent choices of exactly k items to put in the bucket, so the probability that the given bucket has size k is the sum of $p^k q^{n-k}$ over all these independent ways of selecting k items:

$$\sum_{i=1}^{\binom{n}{k}} p^k q^{n-k} = \binom{n}{k} p^k q^{n-k}$$

□

PROPOSITION 6. The bucket size distribution $\{e(k)\}$ satisfies the iterative calculation:

$$\begin{aligned}
e(0) &= b\left(\frac{b-1}{b}\right)^n && \text{initial condition}\\
e(k) &= \left(\frac{n-k+1}{k}\right)\left(\frac{1}{b-1}\right)e(k-1) && \text{for } k = 1\ldots n.\\
e(k) &= 0 && \text{for } k > n.
\end{aligned}$$

Proof. We use the notation established above: $p = 1/b$, $q = (b-1)/b$. Note that $p + q = 1$ and $p/q = 1/(b-1)$.

First the initializer:

$$\begin{aligned}
e(0) &= bp(0) && \text{Proposition 2}\\
&= b\left(\frac{b-1}{b}\right)^n && \text{Proposition 4}
\end{aligned}$$

Then the recurrence:

$$\begin{aligned}
\frac{e(k)}{e(k-1)} &= \frac{bp(k)}{bp(k-1)} && \text{Proposition 2}\\
&= \frac{p(k)}{p(k-1)} && \text{cancel } b\\
&= \frac{\binom{n}{k}p^k q^{n-k}}{\binom{n}{k-1}p^{k-1}q^{n-k+1}} && \text{Proposition 5}\\
&= \frac{(n-k+1)p}{kq} && \text{applying definitions and simplifying}\\
&= \left(\frac{n-k+1}{k}\right)\left(\frac{1}{b-1}\right) && \text{substituting } p = 1/b \text{ and } q = 1-p
\end{aligned}$$

Multiplying by $e(k-1)$ completes the proof for the iterative formula. Finally, note that no bucket can have more than n items, so $e(k) = 0$ when $k > n$. □

Hash Table Analysis Algorithms. Assumptions: n = table size, b = number of buckets

Analysis of a specific hash table, with a specified hash function, number of buckets, and loaded data set, can be an important feature of hash table implementations, simply due to the variability of performance of hashing, particularly on non-random data where a bias in the hashing specified may exist. The results above provide background needed to devise algorithms required for such an analysis.

```
Algorithm 1: Maximum Bucket Size
Runtime: Θ(b)
unsigned long int MaxBucketSize()
{
  unsigned long int max
  for (size_t i = 0; i < b; ++i)
  {
    s = size of bucket i
    if (max < s)
      max = s
  }
  return max;
}
```

```
Algorithm 2: Actual Bucket Size Distribution
Runtime: Θ(b)
void ActualBucketSizeDistribution (Vector<unsigned long int>& sizeCount)
{
  // calculate and set size of distribution
  m = MaxBucketSize();
  sizeCount.SetSize(1+m,0);

  // calculate distribution
  for (size_t i = 0; i < b; ++i)
    ++sizeCount[size of bucket i];
}
```

```
Algorithm 3: Number of Non-Empty Buckets
Runtime: constant
Number of non-empty buckets = b - sizeCount[0];
```

```
Algorithm 4: Simple Uniform Hashing Bucket Size Distribution
Runtime: Θ(n)
void ExpectedBucketSizeDistribution (Vector<float>& expdCount)
{
  expdCount.SetSize(1+n,0.0);  // set size of distribution
  expdCount[0] = (b-1)/b;
  expdCount[0] = pow(expdCount[0],n-1);
  expdCount[0] *= (b-1);
  for (size_t k = 1; k < 1+n; ++k)
    expdCount[k] = ((n-k+1)/((b-1)*k)) * expdCount[k-1];
}
```

Remark. The distribution calculated in Algorithm 4 is a *binomial distribution*, so named because of the relationship with binomial choice and the expansion of the binomial $(A+B)^n$ [with $A=b$ and $B=(b-1)/b$]:

$$(A+B)^n = A^n + A^{n-1}B + ... + \binom{n}{k} A^k B^{n-k} + ... + B^n$$

In this context, because the $\frac{1}{b}$ factors are much smaller than the $\frac{b-1}{b}$ factors, it is very closely approximated by the Poisson distribution.[1] Algorithm 4 calculates many more terms than would normally be useful in practice - most of them are so small they equate to zero in float representation. It is therefore usually a waste of space and time to allocate the entire vector of size $1+n$ and calculate all of its elements. In the case here, it is more prudent to allocate space of size $1+m$ and, if a few more values are needed, calculate them on the fly (continuing the same iterative process, starting at the nth value expdCount[b]) without storing them.

[1]See entries in Wikipedia and NIST Handbook.