Bloom Filters

Bloom filters are a scheme to improve the run time of access methods that ordinarily would search through a page of user records to determine whether the desired record is present. The scheme is useful in situations typified by *linear hashing*, sequence nodes in *B+trees*, and *interpolation-based index maintenance* in which there are large pages to be searched.

A *filter* consists of a sequence of m bits. A suitable hash function

$$h(x) = \{ 0, 1, 2, \ldots, m-1 \}$$

is assumed to be available. During an `insertion`, when key k is to be placed on page p, the bit in position $h(k)$ of the filter of page p is set. During an `access` operation for key k, as page q is considered the bit position $h(k)$ of the signature of page q is examined. If it is set, we must examine the contents of the page; otherwise we can immediately proceed to next page.

Bloom Filters Analysis

Assume we are looking for key k within the data structure. We have just read from disk page p. Suppose that filter bit set by key k is set within page p. Now we can ask what is the probability that key k is really on page p? Or put in a negative way, what is the probability that the proper signature bit is set while the desired key k is not on the page. We refer to the situation, where the proper signature bit is indeed set while the desired key in not on the page, as a *false drop*. The purpose of this analysis is to determine an upper bound on the probability of a false drop for a given filter size and page size.

We assume that a signature contains m bits. We also assume a page always constains a signature as well as n keys. The probability that a particular bit is set when a key is inserted into a page is $\frac{1}{m}$. Then the probability that a particular bit is not set after n keys are inserted is $\left(\frac{m-1}{m}\right)^n$. Finally the probability that a particular bit is set after n keys are inserted is

$$1 - \left(\frac{m-1}{m}\right)^n.$$

This expression is an upper bound on the false drops associated with a particular bit. Since it determines the probability of the bit being set, it measures the presence of both the false drops and the desired key. We can give a useful approximation to this expression

$$1 - e^{-n/m} \approx \frac{n}{m}$$

The final expression $\frac{n}{m}$ is extremely accurate approximation when the value is very small.

As an example of the use of this estimate, suppose we desire to have the false drops occur no more than 10% of the time on the average. Then if $\frac{n}{m} = 0.1$, we can have no more than an average of 10% false drops. To properly select m, we first determine a useful page size n. If n is five, then m must be at least 50. We continue our example in the implementation section. A page must contain both the data and the filter.

Implementation

Here are macros implementing the *set* and *check* filter operations. We assume the number of bits m is a multiple of 32, that is $32 \times$ `W` on a 32 bits per unsigned integer machine. The filter is an array of `W` unsigned integers.

```
#define check(p,k)    ( p.signature [ k % W ]  &  ( 1 << ( k & 037 ) ) )
#define set(p,k)      ( p.signature [ k % W ]  |= ( 1 << ( k & 037 ) ) )
```

where `k` is the hash value of the key and `p` is the page. Continuing with our previous example, we would allocate `W` $= 2$ unsigned integer words for our filter.

Reference

Bloom, R.H., Space/time trade-offs in hash coding with allowable errors, *Communications of the Association for Computing Machinery* volume 13, (1970), 422-426.