Hashing: Lecture II

Predicting Record Distribution

Throughout this section we assume a random distribution for the hash funtion.

Let
N = number of available addresses, and
r = number of records to be stored.

Let $p(x)$ be the probability that a given address will have x records assigned to it.

It can be shown that

$$p(x) = \frac{r!}{(r-x)!x!} \left[1 - \frac{1}{N}\right]^{r-x} \left[\frac{1}{N}\right]^{x}$$

and for N and r large this can be approximated by :

$$p(x) \sim \frac{(r/N)^x \, e^{-(r/N)}}{x!}$$

Example : $N = 1{,}000$ $r = 1{,}000$

$$p(0) \sim \frac{1^0 \, e^{-1}}{0!} = 0.368$$

$$p(1) \sim \frac{1^1 \, e^{-1}}{1!} = 0.368$$

$$p(2) \sim \frac{1^2 \, e^{-1}}{2!} = 0.184$$

$$p(3) \sim \frac{1^3 \, e^{-1}}{3!} = 0.061$$

For N addresses the expected number of addresses with x records is $N \cdot p(x)$.

So in the example above about:

368	addresses have no records assigned to it
368	addresses have 1 records assigned to it
184	addresses have 2 records assigned to it
61	addresses have 3 records assigned to it

Reducing Collision by Increasing the Number of Available Addresses

`packing density` $= r/N$

500 records to be spread over 1000 addresses result in `packing density` $= 500/1000 = 0.5 = 50\%$.

Some questions :

1. How many addresses go unused ? *More precisely: What is the* **expected** *number of addresses with no key mapped to it?*
   $N \cdot p(0) = 1000 \cdot 0.607 = 607$

2. How many addresses have no synonyms ? *More precisely: What is the expected number of address with only one key mapped to it?*
   $N \cdot p(1) = 1000 \cdot 0.303 = 303$

3. How many addresses contain 2 or more synonyms ? *More precisely: What is the expected number of addresses with two or more keys mapped to it ?*
   $N \cdot (p(2) + p(3) + ...) = N \cdot (1 - (p(0) + p(1)) = 1000 \cdot 0.09 = 90$

4. Assuming that only one record can be assigned to an address, how many overflow records are expected ?
   $1 \cdot N \cdot p(2) + 2 \cdot N \cdot p(3) + 3 \cdot N \cdot p(4) + ... = N \cdot [p(2) + 2 \cdot p(3) + 3 \cdot p(4) + ...] \sim 107$.
   The justification for the above formula is that there is going to be $(i-1)$ overflow records for all the table positions that have i records mapped to it, which are expected to be as many as $N \cdot p(i)$.

Now, there is a simpler formula derived by students of Section B of the course (the solution below is due to Tanya Scheffler and Pat Wisking):

```
expected # of overflow records =
= (total # of records) - (expected # of nonoverflow records)
```

$= r - (N \cdot p(1) + N \cdot p(2) + N \cdot p(3) + \ldots)$
$= r - N \cdot (1 - p(0))$ (since probabilities add up to 1)
$= N \cdot p(0) - (N - r)$

```
= (expected # of empty positions for random hash funtion)
  - (# of empty positions for perfect hash function)
```

Using this formula we get the same result as before:
$N \cdot p(0) - (N - r) = 607 - 500 = 107$

5. What is the expected percentage of overflow records ?
$107/500 = 0.214 = 21.4\%$

Note that using either formula, the percentage of overflow records depend only on the packing density ($PD = r/N$), and not on the individual values of N or r.

Indeed, using the formulas derived in 4., we get that the percentage of overflow records is:

$$\frac{r - N \cdot (1 - p(0))}{r} = 1 - \frac{1}{PD} \cdot (1 - p(0))$$

and the Poisson function that approximate $p(0)$ is a function of r/N which is equal to PD (for hashing without buckets).

So, hashing with packing density $PD = 50\%$ always yield 21% of records stored outside their home addresses.

For this reason, we can compute the expected percentage of overflow records, given the packing density. This is shown in the following table:

packing density %	**number of records away from home %**
10%	4.8%
20%	9.4%
30%	13.6%
40%	17.6%
50%	21.4%
60%	24.8%
70%	28.1%
80%	31.2%
90%	34.1%
100%	36.8%

Collision Resolution by Progressive Overflow/Linear Probing

Progressive overflow/linear probing works as follows :

Insertion of key k:
- Go to the home address of k : h(k)
- If free, place the key there
- If busy, try the next position until an empty position is found
(the 'next' position for the last position is position 0, i.e. wrap around)

Example :

key - k	**Home address - h(k)**
COLE	20
BATES	21
ADAMS	21
DEAN	22
EVANS	20

Table size = 23.

After inserting previous keys :

0	DEAN
1	EVANS
⋮	⋮
19	
20	COLE
21	BATES
22	ADAMS

Searching for key k:
- Go to the home address of k : h(k)
- If k is in home address, we are done.
- Otherwise try the next position until: key is found or empty space is found or home address is reached (in the last 2 cases, the key is not found)

Ex :
A search for 'EVANS' probes places : 20, 21, 22, 0, 1, finding the record at position 1.

Search for 'MOURA', if h(MOURA)=22, probes places 22, 0, 1, 2 where it concludes 'MOURA' in not in the table.

Search for 'SMITH', if h(SMITH)=19, probes 19, and concludes 'SMITH' in not in the table.

Advantage : Simplicity
Disadvantage : If there are lots of collisions, clusters of records can form, as in the previous example.

Search length
- Number of accesses required to retrieve a record.

`average search length = (sum of search lengths)/(numb.of records)`

In the previous example :

key	Search Length
COLE	1
BATES	1
ADAMS	2
DEAN	2
EVANS	5

Average search length = (1+1+2+2+5)/5 = 2.2.

Refer to figure 11.7 in page 489. It shows that a packing density up to 60% gives an average search length of 2 probes, but higher packing densities make search length to increase rapidly.