13 Hashing

Objectives

- Introduce the concept of hashing
- Examine the problem of choosing a good hashing algorithm
- Explore three approaches for reducing collisions
- Develop and use mathematical tools for analyzing performance differences resulting from the use of different hashing techniques
- Examine problems associated with file deterioration and discuss some solutions
- Examine effects of patterns of records access on performance

Content

- Introduction to Hashing
- Hash functions
- Distribution of records among addresses, synonyms and collisions
- Collision resolution by progressive overflow or linear probing

Motivation

- Hashing is a useful searching technique, which can be used for implementing indexes. The main motivation for Hashing is improving searching time.
- Below we show how the search time for Hashing compares to the one for other methods:
  - Simple Indexes (using binary search): $O(\log_2 N)$
  - B Trees and B+ trees: $O(\log_k N)$
  - Hashing: $O(1)$

What is Hashing?

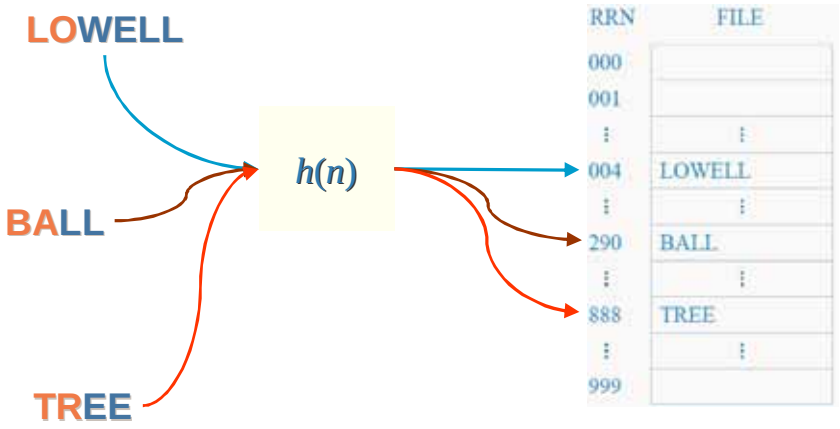
- The idea is to discover the location of a key by simply examining the key. For that we need to design a hash function.
- A **Hash Function** is a function $h(k)$ that transforms a key into an address
- An address space is chosen before hand. For example, we may decide the file will have 1,000 available addresses.
- If U is the set of all possible keys, the hash function is from U to $\{0,1,...,999\}$, that is

$$h : U \rightarrow \{0,1,...,999\}$$

Example

NAME	ASCII code for first two letters	PRODUCT	HOME ADDRESS
BALL	66 65	66×65=4290	290
LOWELL	76 79	76×79=6004	004
TREE	84 82	84×82=6888	888

What is Hashing?



What is Hashing?

- There is no obvious connection between the key and the location (randomizing)
- Two different keys may be sent to the same address generating a Collision
- Can you give an example of collision for the hash function in the previous example?

Answer

- LOWELL, LOCK, OLIVER, and any word with first two letters **L** and **O** will be mapped to the same address

  h(**LO**WELL)=h(**LO**CK)=h(**OL**IVER)=**004**

- These keys are called synonyms. The address "004" is said to be the home address of any of these keys.
- Avoiding collisions is extremely difficult
- Do you know the birthday paradox?
- So we need techniques for dealing with it.

Reducing Collisions

1. Spread out the records by choosing a good hash function
2. Use extra memory: increase the size of the address space
   - Example: reserve 5,000 available addresses rather than 1,000
3. Put more than one record at a single address: use of buckets

A Simple Hash Function

- To compute this hash function, apply 3 steps:
- **Step 1**: transform the key into a number.

Hashing 13

LOWELL

L	O	W	E	L	L						

ASCII code

76	79	87	69	76	76	32	32	32	32	32	32

A Simple Hash Function (Con't)

- **Step 2**: fold and add (chop off pieces of the number and add them together) and take the mod by a prime number

Hashing 13

76	79	87	69	76	76	32	32	32	32	32	32

7679	8769	7676	3232	3232	3232

7679+8769+7676+3232+3232+3232

33,820 mod 19937 = 13,883

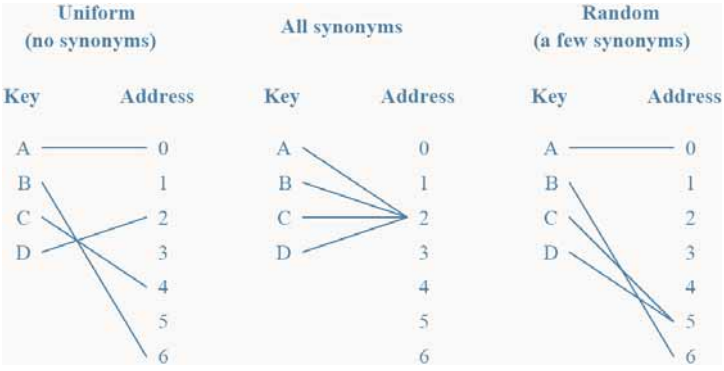
A Simple Hash Function (Con't)

- **Step 3**: divide by the size of the address space (preferably a prime number)

```
13,883 mod 101 = 46
```


Distribution of Records among Addresses

- There are 3 possibilities



- Uniform distributions are extremely rare
- Random distributions are acceptable and more easily obtainable.

Better than Random Distribution

- Examine keys for patterns
  - Example: Numerical keys that are spread out naturally such as keys are years between 1970 and 2004

    f(year)=($year$-1970) mod (2004-1970+1)

    f(1970)=0, f(1971)=1,..., f(2004)=34
- Fold parts of the key.
  - Folding means extracting digits from a key and adding the parts together as in the previous example.
  - In some cases, this process may preserve the natural separation of keys, if there is a natural separation

Better than Random Distribution (Con't)

- Use prime number when dividing the key.
  - Dividing by a number is good when there are sequences of consecutive numbers.
  - If there are many different sequences of consecutive numbers, dividing by a number that has many small factors may result in lots of collisions. A prime number is a better choice.

Randomization

- When there is no natural separation between keys, try randomization.
- You can using the following Hash functions:

1. **Square the key and take the middle**

   Example: key=453 $453^2 = 205209$

   Extract the middle = 52.

   This address is between 00 and 99.

Randomization (Con't)

2. **Radix transformation:**

   Transform the number into another base and then divide by the maximum address

   Example: Addresses from 0 to 99

   key = 453 in base 11 = 382

   hash address = 382 mod 99 = 85.

Collision Resolution: Progressive Overflow

- Progressive overflow/linear probing works as follows:

1. **Insertion of key k**:
   - Go to the home address of k: $h(k)$
   - If free, place the key there
   - If occupied, 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

Complete Table:

0	
1	
2	
⋮	⋮
19	
20	
21	
22	

Table size = 23

Progressive Overflow (Con't)

2. **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)

Example

- 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.

0	DEAN
1	EVANS
2	
⋮	⋮
19	
20	COLE
21	BATES
22	ADAMS

Advantages ✕ Disadvantages

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

Search Length

- Number of accesses required to retrieve a record.

$$\textbf{average search length} = \frac{\textbf{sum of search lengths}}{\textbf{number of records}}$$

Example

0	DEAN
1	EVANS
2	
⋮	⋮
19	
20	COLE
21	BATES
22	ADAMS

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

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

- Average search length

  (1+1+2+2+5)/5=2.2

Predicting Record Distribution

- We assume a random distribution for the hash function.
  - N = number of available addresses
  - 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 is easy to see that

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

Predicting Record Distribution (Con't)

- For N and r large enough this can be approximated by

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

Example

- N=1000, r=1000

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

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

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

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

Predicting Record Distribution (Con't)

- For N addresses, the expected number of addresses with *x* records is

$$N \cdot p(x)$$

- N=1000, r=1000

$$1000 \times p(0) = 368$$

$$1000 \times p(1) = 368$$

$$1000 \times p(2) = 184$$

$$1000 \times p(3) = 61$$

Reducing Collision by using more Addresses

- Now, we see how to reduce collisions by increasing the number of available addresses.
- Definition: **`packing density`** = r/N
- Example:

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

Questions

1. How many addresses go unused? More precisely: *What is the expected number of addresses with no key mapped to it*?

- $N \times p(0) = 1000 \times 0.607 = 607$

Questions (Con't)

2. How many addresses have no synonyms? More precisely: *What is the expected number of address with only one key mapped to it*?

- $N \times p(1) = 1000 \times 0.303 = 303$

Questions (Con't)

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*?

- $N\times(p(2)+p(3)+...)= N\times(1-p(0)-p(1))= 1000 \times 0.09 = 90$

Questions (Con't)

4. Assuming that only one record can be assigned to an address. How many overflow records are expected?

$$1\times N\times p(2) + 2\times N\times p(3) + 3\times N\times p(4)+... =$$

$$N\times(2\times p(2)+3\times p(4)+...) \approx 107$$

- 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 $N\cdot p(i)$

A Simpler Formula

- Expected # of overflow records =
  #records – Expected # of non-overflow records
  $= r - (N \cdot p(1) + N \cdot p(2) + N \cdot p(3) + \cdots)$
  $= r - (1 - p(0))$
  $= N \cdot p(0) - (N - r)$

Questions (Con't)

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.
- The percentage of overflow records is

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

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

Packing Density-Overflow Records

Packing Density %	Overflow Records %
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%

Hashing with Buckets

- This is a variation of hashed files in which more than one record/key is stored per hash address.
- Bucket = block of records corresponding to one address in the hash table
- The hash function gives the **Bucket Address**
- Example:

Example

- For a bucket holding 3 records, insert the following keys

key	Home Address
LOYD	34
KING	33
LAND	33
MARX	33
NUTT	33
PLUM	34
REES	34

0	
⋮	⋮
33	KING
	LAND
	MARX
34	LOYD

Effects of Buckets on Performance

- We should slightly change some formulas

$$packing\ density = \frac{r}{b \cdot N}$$

We will compare the following two alternatives

1. Storing 750 data records into a hashed file with 1000 addresses, each holding 1 record.
2. Storing 750 data records into a hashed file with 500 bucket addresses, each bucket holding 2 records

- In both cases the packing density is 0.75 or 75%.
- In the first case r/N=0.75.
- In the second case r/N=1.50

Effects of Buckets on Performance

▶ Estimating the probabilities as defined before:

	p(0)	p(1)	p(2)	p(3)	p(4)
1) r/N=0.75 (b=1)	0.472	0.354	0.133	0.033	0.006
2) r/N=1.50 (b=2)	0.223	0.335	0.251	0.126	0.047

Effects of Buckets on Performance

Calculating the number of overflow records in each case

1. **b=1** (r/N=0.75):

$$\textit{Number of overflow records} =$$
$$= N \cdot \left[1 \cdot p(2) + 2 \cdot p(3) + 3 \cdot p(4) + \cdots\right]$$
$$= r - N \cdot \left(1 - p(0)\right)$$
$$= 750 - 1000 \cdot \left(1 - 0.472\right) = 750 - 528 = 222$$

This is about 29.6% overflow

Effects of Buckets on Performance

2. **b=2** (r/N=1.5):

$$\begin{aligned}
&\textit{Number of overflow records} = \\
&= N \cdot \left[1 \cdot p(3) + 2 \cdot p(4) + 3 \cdot p(5) + \cdots\right] \\
&= r - N \cdot p(1) - 2 \cdot N \cdot \left[p(2) + p(3) + \cdots\right] \\
&= r - N \cdot \left[p(1) + 2 \cdot \left(1 - p(0) - p(1)\right)\right] \\
&= r - N \cdot \left[2 - 2 \cdot p(0) - p(1)\right] \\
&= 750 - 500 \cdot \left(2 - 2 \cdot (0.223) - 0.335\right) = 140.5 \cong 140
\end{aligned}$$

This is about 18.7% overflow

Percentage of Collisions for Different Bucket Sizes

	Bucket Size				
Packing Density %	1	2	5	10	100
75%	29.6%	18.7%	8.6%	4.0%	0.0%

Implementation Issues

1. Bucket Structure

- A Bucket should contain a counter that keeps track of the number of records stored in it.
- Empty slots in a bucket may be marked '//.../'
- Example: Bucket of size 3 holding 2 records

2	JONES	//////////..//	ARNSWORTH

Implementation Issues

2. Initializing a file for hashing

- Decide on the Logical Size (number of available addresses) and on the number of buckets per address.
- Create a file of empty buckets before storing records. An empty bucket will look like

0	//////////..//	//////////..//	//////////..//

Implementation Issues

3. Loading a hash file

- When inserting a key, remember to:
- Be careful with infinite loops when hash file is full

Making Deletions

- Deletions in a hashed file have to be made with care

Record	ADAMS	JONES	MORRIS	SMITH
Home Address	5	6	6	5

Hashed File using Progressive Overflow

⋮	⋮
4	//////////
5	ADAMS
6	JONES
7	MORRIS
8	SMITH
⋮	⋮

Making Deletions: Delete ‘MORRIS’

- If ‘MORRIS’ is simply erased, a search for ‘SMITH’ would be unsuccessful

⋮	⋮
4	////////// ← Empty Slot
5	ADAMS
6	JONES
7	////////// ← Empty Slot
8	SMITH
⋮	⋮

Problem: you cannot find ‘SMITH’

- Search for ‘SMITH’ would go to home address (position 5) and when reached 7 it would conclude ‘SMITH’ is not in the file!

Solution

- Replace deleted records with a marker indicating that a record once lived there

⋮	⋮
4	//////////
5	ADAMS
6	JONES
7	###### ← Deleted Slot
8	SMITH
⋮	⋮

you can find ‘SMITH’

- A search must continue when it finds a tombstone, but can stop whenever an empty slot is found

Be careful in Deleting and Adding a Record

- Only insert a tombstone when the next record is occupied or is a tombstone
- Insertions should be modified to work with tombstones: if either an empty slot or a tombstone is reached, place the new record there.

Effects of Deletions and Additions on Performance

- The presence of too many tombstones increases search length.
- Solutions to the problem of deteriorating average search lengths:

1. Deletion algorithm may try to move records that follow a tombstone backwards towards its home address
2. Complete reorganization: re-hashing
3. Use a different type of collision resolution technique

Other Collision Resolution Techniques

1. Double Hashing

- The first hash function determines the home address
- If the home address is occupied, apply a second hash function to get a number c (c relatively prime to N)
- c is added to the home address to produce an overflow addresses: if occupied, proceed by adding c to the overflow address, until an empty spot is found.

Example

k (key)	ADAMS	JONES	MORRIS	SMITH
$h_1(k)$ (home address)	5	6	6	5
$h_2(k) = c$	2	3	4	3

Hashed file using double hashing

2	
3	
4	
5	ADAMS
6	JONES
7	
8	SMITH
9	
10	MORRIS

A Question

- Suppose the above table is full, and that a key k has $h_1(k)=6$ and $h_2(k)=3$.
- What would be the order in which the addresses would be probed when trying to insert k?

Answer: 6, 9, 1, 4, 7, 10, 2, 5, 8, 0, 3

0	XXXXX
1	XXXXX
2	XXXXX
3	XXXXX
4	XXXXX
5	XXXXX
6	XXXXX
7	XXXXX
8	XXXXX
9	XXXXX
10	XXXXX

Other Collision Resolution Techniques (Con't)

2. Chained Progressive Overflow

- Similar to progressive overflow, except that synonyms are linked together with pointers.
- The objective is to reduce the search length for records within clusters.

Example

Key	Home	Progressive Overflow	Chained Progr. Overflow
ADAMS	20	1	1
BATES	21	1	1
COLES	20	3	2
DEAN	21	3	2
EVANS	24	1	1
FLINT	20	6	3
Average Search Length :		2.5	1.7

Example (Con't)

Progressive Overflow

	data
⋮	⋮
20	ADAMS
21	BATES
22	COLES
23	DEAN
24	EVANS
25	FLINT
⋮	⋮

Chained Progressive Overflow

	data	next
⋮	⋮	⋮
20	ADAMS	22
21	BATES	23
22	COLES	25
23	DEAN	-1
24	EVANS	-1
25	FLINT	-1
⋮	⋮	⋮

Other Collision Resolution Techniques (Con't)

3. Chained with a Separate Overflow Area

- Move overflow records to a Separate Overflow Area
- A linked list of synonyms start at their home address in the Primary data area, continuing in the separate overflow area
- When the packing density is higher than 1 an overflow area is required

Example

Primary Data Area

20	ADAMS	0
21	BATES	1
22		
23		
24	EVANS	-1
25		

Overflow Area

0	COLES	2
1	DEAN	-1
2	FLINT	-1
3		
	⋮	⋮

Other Collision Resolution Techniques (Con't)

4. Scatter Tables: Indexing Revisited

- Similar to chaining with separate overflow, but the hashed file contains no records, but only pointers to data records.

index (hashed)

	⋮
20	0
21	1
22	
23	
24	4
	⋮

datafile

	data	next
0	ADAMS	2
1	BATES	3
2	COLES	5
3	DEAN	-1
4	EVANS	-1
5	FLINT	-1