

# Hashing

**Lecture No. 8**

# Contents

1

What is Hashing?

2

Hash Function

3

Collisions Reduction



# **What is Hashing?**

# Introduction

- ❖ **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**:  $O(1)$

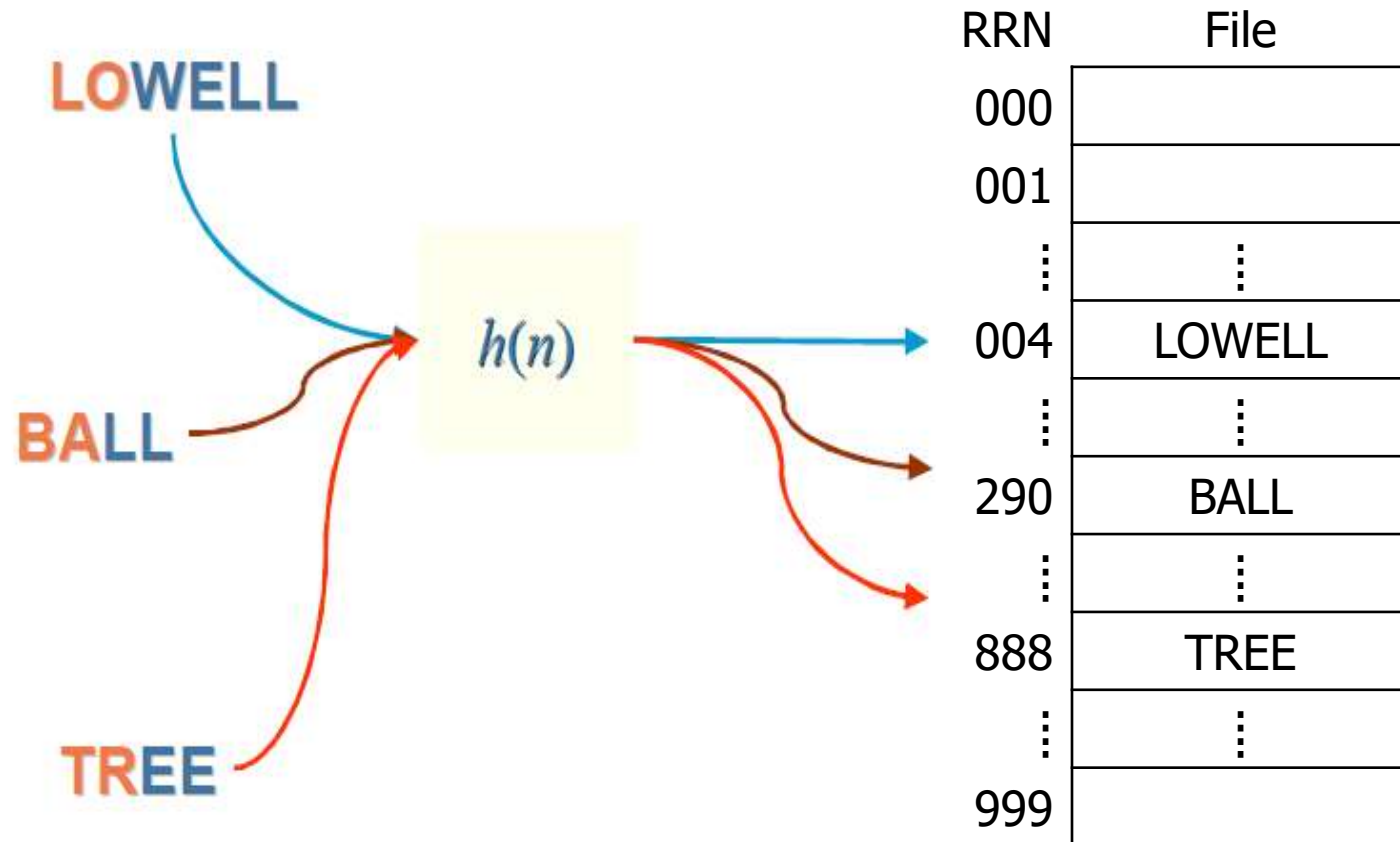
# What is Hashing?

- ❖ 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  $\{0, 1, \dots, 999\}$ , that is  $h : U \rightarrow \{0, 1, \dots, 999\}$

# Example

NAME	ASCII code for first two letters	PRODUCT	HOME ADDRESS
<b>BALL</b>	<b>66 65</b>	<b>66×65=4290</b>	<b>290</b>
<b>LOWELL</b>	<b>76 79</b>	<b>76×79=6004</b>	<b>004</b>
<b>TREE</b>	<b>84 82</b>	<b>4×82=6888</b>	<b>888</b>

# 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?

# What is Hashing?

- ❖ LOWELL, LOCK, OLIVER, and any word with first two letters L and O will be mapped to the same address  $h(\text{LOWELL})=h(\text{LOCK})=h(\text{OLIVER})=004$
- ❖ 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, 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**

# Hash Function

# A simple Hash Function

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

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

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

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

7679	8769	7676	3232	3232	3232
------	------	------	------	------	------

$7679 + 8769 + 7676 + 3232 + 3232 + 3232$

$33,820 \bmod 19937 = 13,883$

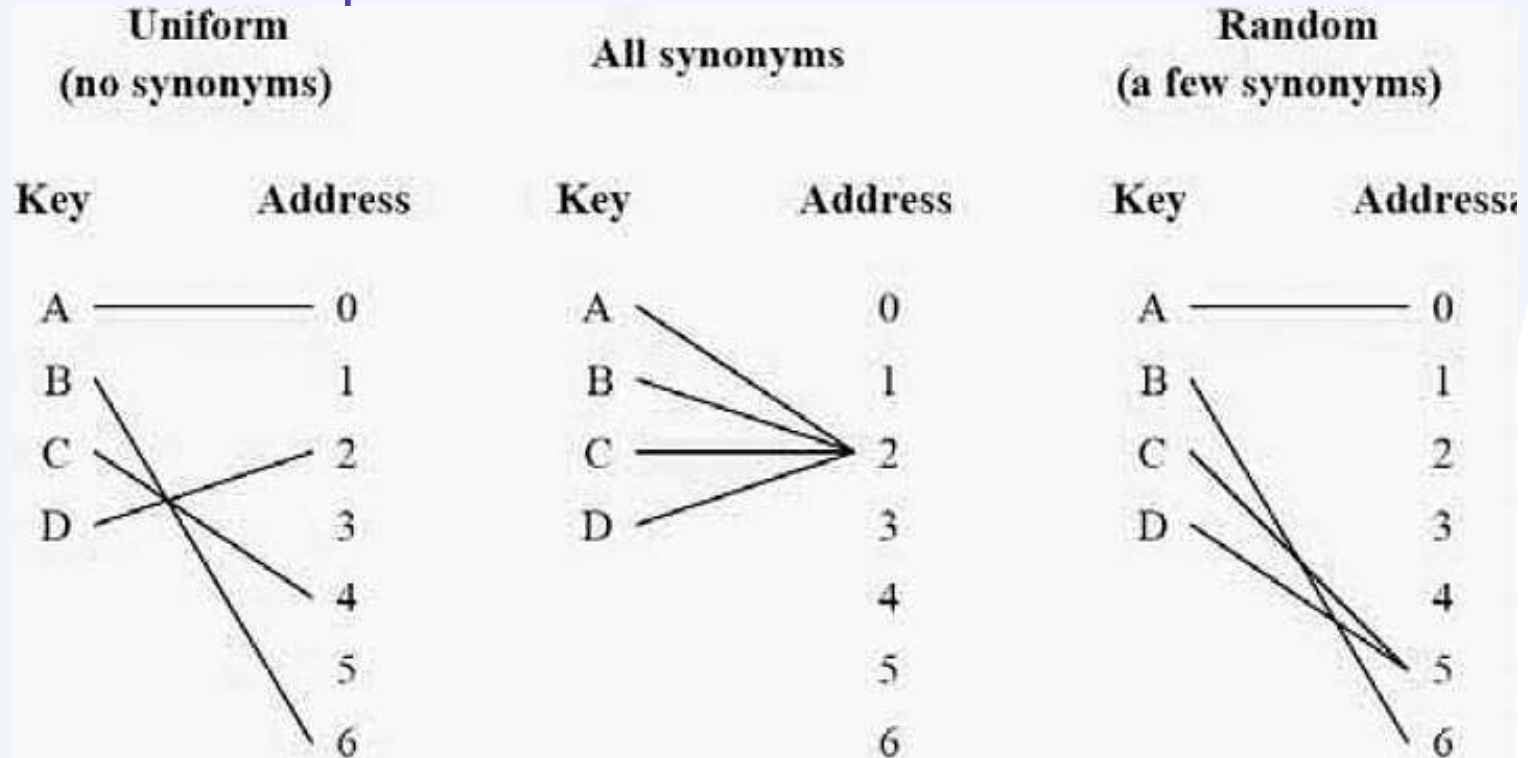
# A simple Hash Function

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

$$13,883 \bmod 101 = 46$$

# 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  
$$f(\text{year}) = (\text{year} - 1970) \bmod (2004 - 1970 + 1)$$
$$f(1970) = 0, f(1971) = 1, \dots, f(2004) = 34$$

## ❖ 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

## ❖ 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

## 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 =  $382 \bmod 99 = 85$ .



## **Collisions Reduction**

# 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)

# Collision Resolution: Progressive Overflow

## ❖ 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

# Collision Resolution: Progressive Overflow

## ❖ Example:

Key K	Home Address – $h(k)$
COLE	20
BATES	21
ADAMS	21
DEAN	22
EVANS	20

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

Table Size=23

# Collision Resolution: Progressive Overflow

## 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)

# Collision Resolution: Progressive Overflow

## ❖ Example:

- A search for 'EVANS' probes places: 20,21,22,0,1, finding the record at position 1.
- Search for 'MOURA', if  $h(\text{MOURA})=22$ , probes places 22,0,1,2 where it concludes 'MOURA' in not in the table.
- Search for 'SMITH', if  $h(\text{SMITH})=19$ , probes 19, and concludes 'SMITH' in not in the table.

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

Table Size=23

# Collision Resolution: Progressive Overflow

## ❖ Advantages:

- Simplicity

## ❖ Disadvantage:

- If there are lots of collisions of records, as in the previous example

# Collision Resolution: Progressive Overflow

## ❖ Search length:

- It is the number of accesses required to retrieve a record.

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

# Collision Resolution: Progressive Overflow

## ❖ Example:

Key K	Home Address – h(k)
COLE	20
BATES	21
ADAMS	21
DEAN	22
EVANS	20

Key K	Search length
COLE	1
BATES	1
ADAMS	2
DEAN	2
EVANS	5

Complete Table

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

Table Size=23

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

# 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

# Hashing with Buckets

## ❖ Example:

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

Key K	Home Address – $h(k)$
LOYD	34
KING	33
LAND	33
MARX	33
MUTT	33
PLUM	34
REES	34

Complete Table	
0	
⋮	⋮
33	KING
	LAND
	MARX
34	LOYD

# Hashing with Buckets

## ❖ Example:

Key K	Home Address – $h(k)$
LOYD	34
KING	33
LAND	33
MARX	33
MUTT	33
PLUM	34
REES	34

Complete Table	
0	REES
⋮	⋮
33	KING
	LAND
	MARX
34	LOYD
	MUTT
	PLUM

# Hashing with Buckets: 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
---	-------	----------------	-----------

# Hashing with Buckets: 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	//////////////////..//	//////////////////..//	//////////////////..//
---	------------------------	------------------------	------------------------

# Hashing with Buckets: Implementation issues

## 3. Loading a hash file:

- When inserting a key, remember to:
  - Be careful with infinite loops when hash file is full
  - Create a file of empty buckets before storing records.

# Making Deletions

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

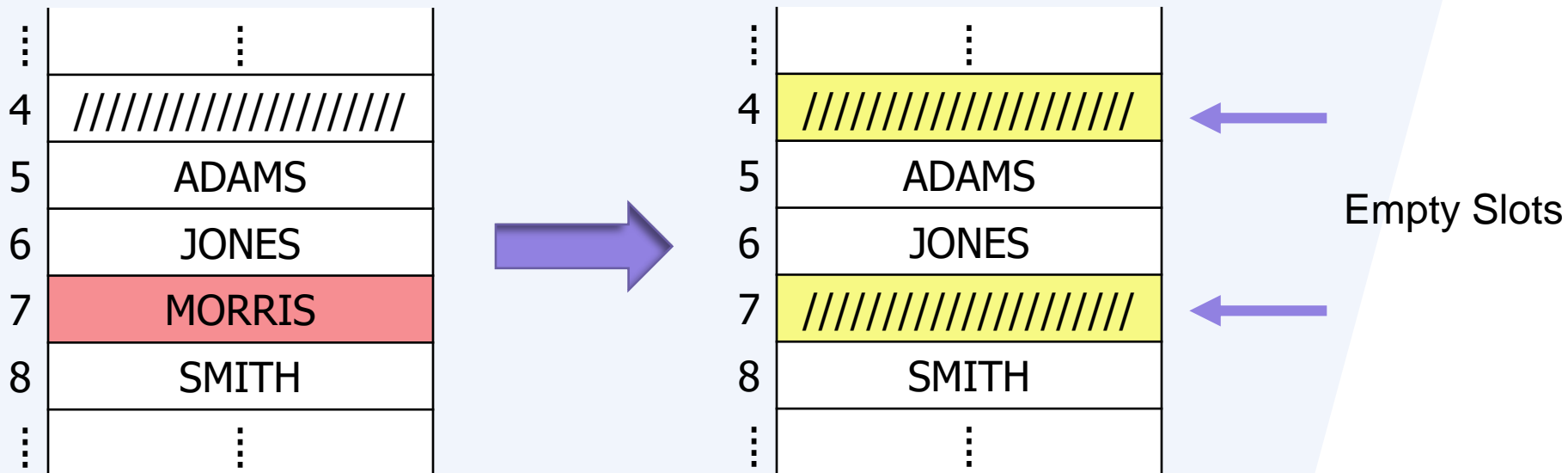
Key K	Home Address – $h(k)$
ADAMS	5
JONES	6
MORRIS	6
SMITH	5

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

Hashed File using  
Progressive Overflow

# Making Deletions: Delete 'MORRIS'

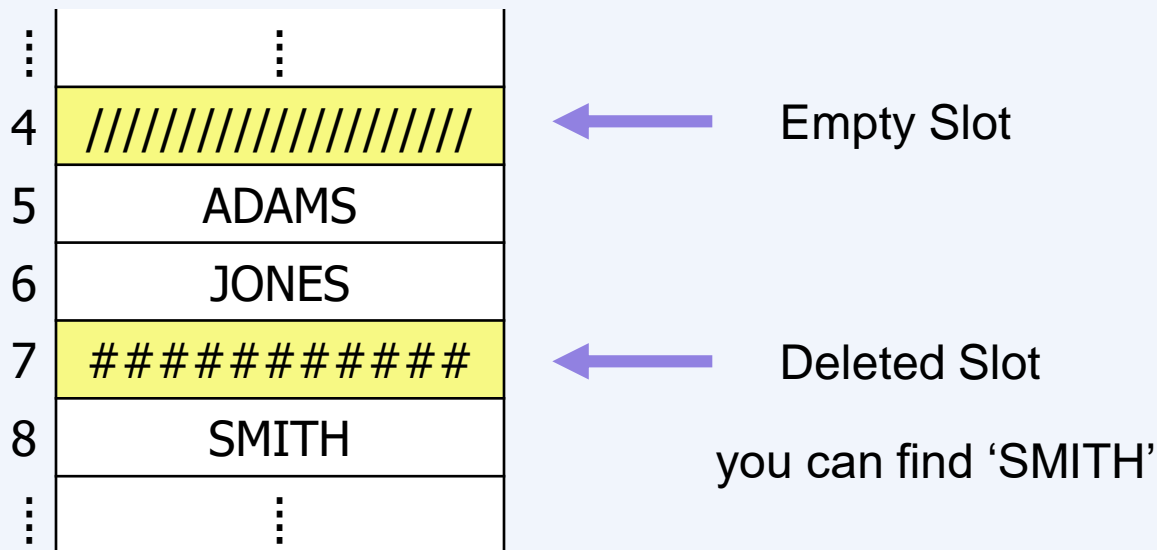
- ❖ If 'MORRIS' is simply erased, a search for 'SMITH' would be unsuccessful



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

# Making Deletions: Delete 'MORRIS'

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



- ❖ 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 Rerecord

- ❖ 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 address: if occupied, proceed by adding  $c$  to the overflow address, until an empty spot is found.

# Other Collision Resolution Techniques

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

Hashed file using double  
hashing

0	
1	
2	
3	
4	
5	ADAMS
6	JONES
7	
8	SMITH
9	
10	MORRIS

# Other Collision Resolution Techniques

- ❖ 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	XXXXXXX
1	XXXXXXX
2	XXXXXXX
3	XXXXXXX
4	XXXXXXX
5	XXXXXXX
6	XXXXXXX
7	XXXXXXX
8	XXXXXXX
9	XXXXXXX
10	XXXXXXX

# Other Collision Resolution Techniques

## 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.

# Other Collision Resolution Techniques

## ❖ Example:

Key K	home address	Progressive Overflow	Chained Progressive Overflow
ADAMS	20	1	1
BATES	21	1	1
COLES	20	3	2
DEAN	21	3	2
EVANS	24	1	1
FLI NT	20	6	3
<b>Average Search Length</b>		<b>2.5</b>	<b>1.7</b>

# Other Collision Resolution Techniques

## ❖ Example:

### Progressive Overflow

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

### Chained Progressive Overflow

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

# Other Collision Resolution Techniques

## 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

# Other Collision Resolution Techniques

## ❖ Example:

### Primary Data Area

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

### Overflow Area

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



# Thank You !