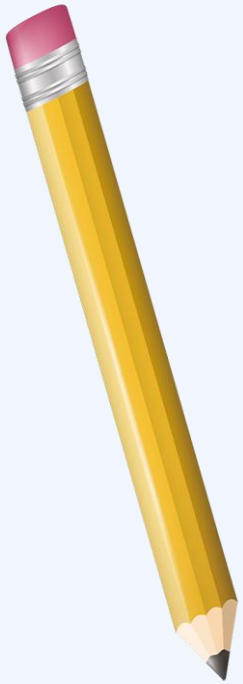
اللهم صل وسلم وبارك على سيدنا محمد وعلى
آله وصحبة وسلم تسليماً كثيراًطيباً مباركاً فيه

File Organization



Dr \ Mohammed Ahmed Mahfouz

Doctor of Information Systems
Thebes Higher Institute for Management and Information Technology

Hashing

Lecture No. 8

Contents

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

Example

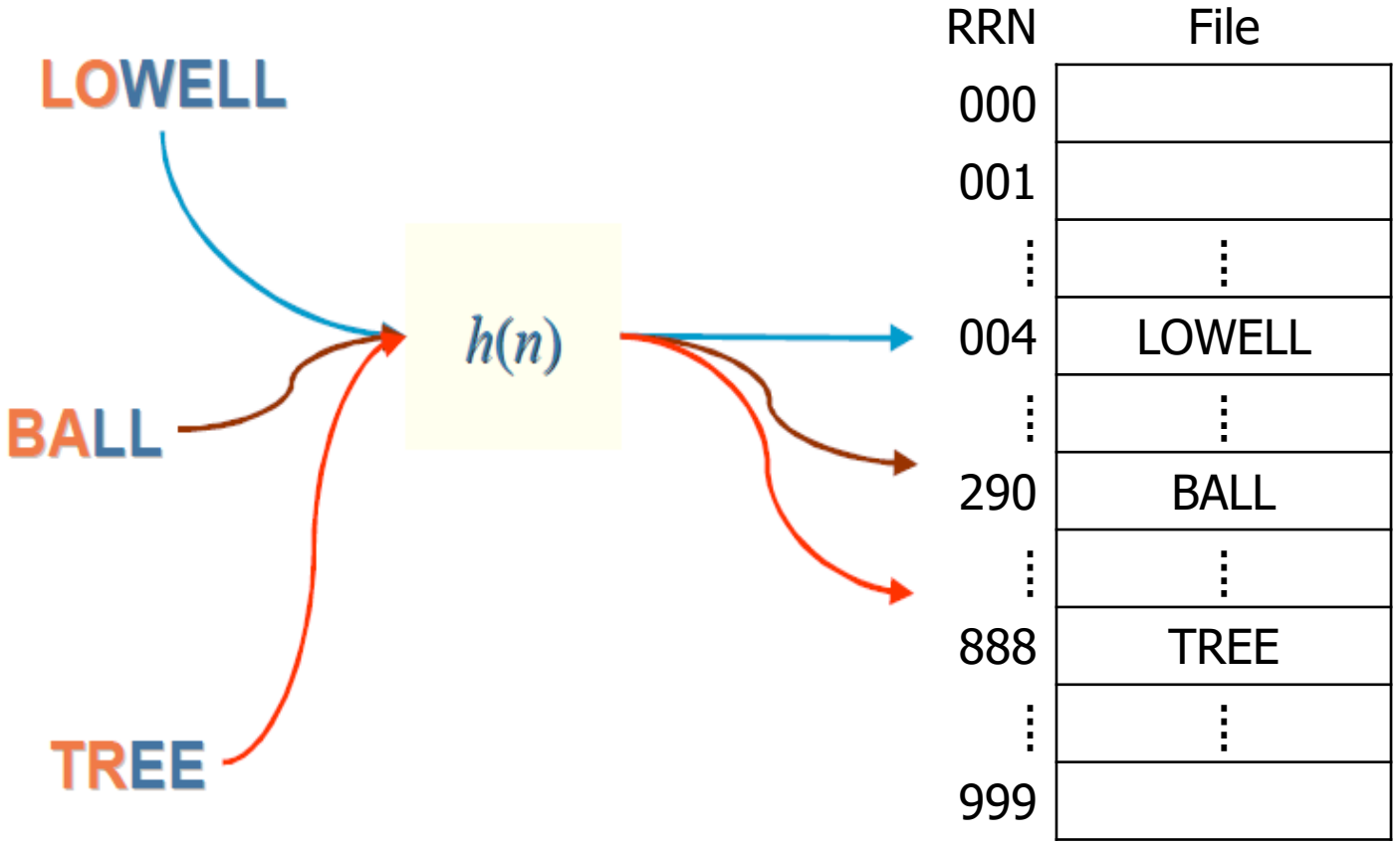
NAME	ASCII code for first two letters	PRODUCT	HOME ADDRESS
BALL	66 65	66×65=4290	290
LOWELL	76 79	76×79=6004	004
TREE	84 82	4×82=6888	888

Example

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	32	20	Space	64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

ASCII TABLE

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(LOWELL)=h(LOCK)=h(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 mod 19937 = 13,883

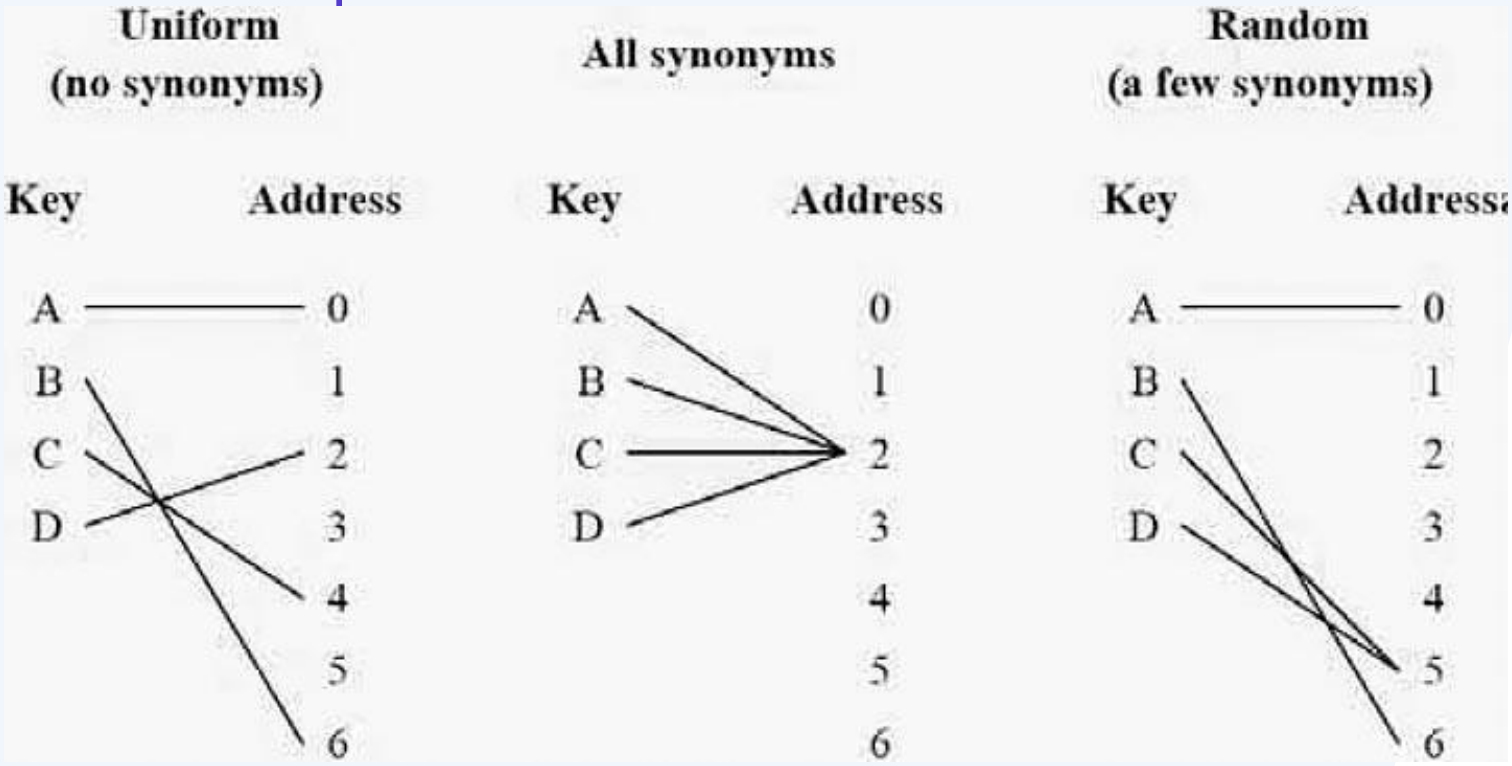
A simple Hash Function

- **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

- 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 mod 99 = 85.

Collisions Reduction

Collision Resolution: Progressive Overflow

- **Progressive overflow/linear probing** works as follows:

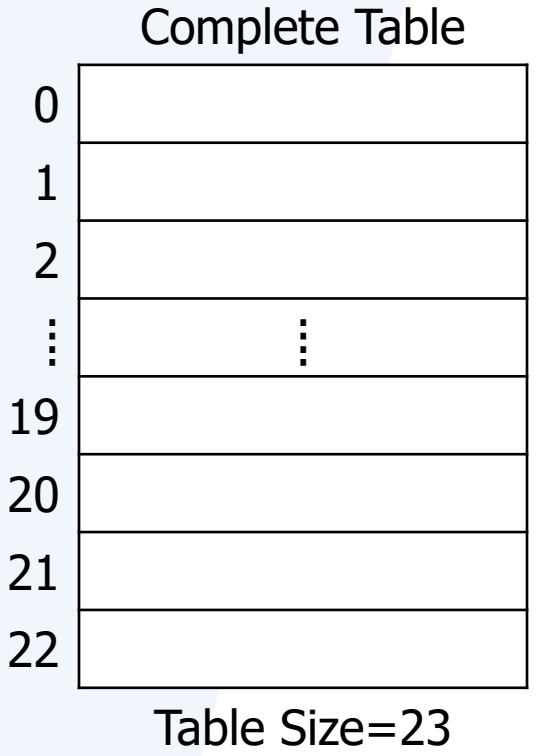
1. Insertion of key k:
   - Go to the home address of k: h(k)
   - If free, place the key there
   - If occupied, try the next position until an empty position is found

   (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



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

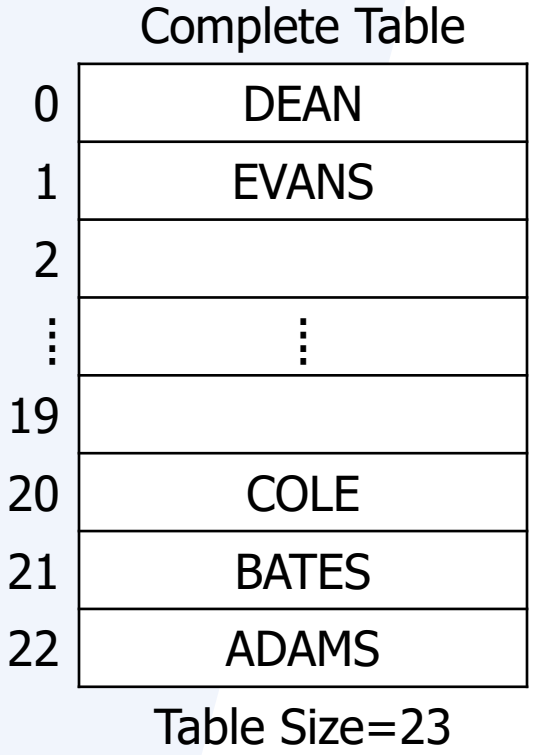
Table Size=23

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(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.



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

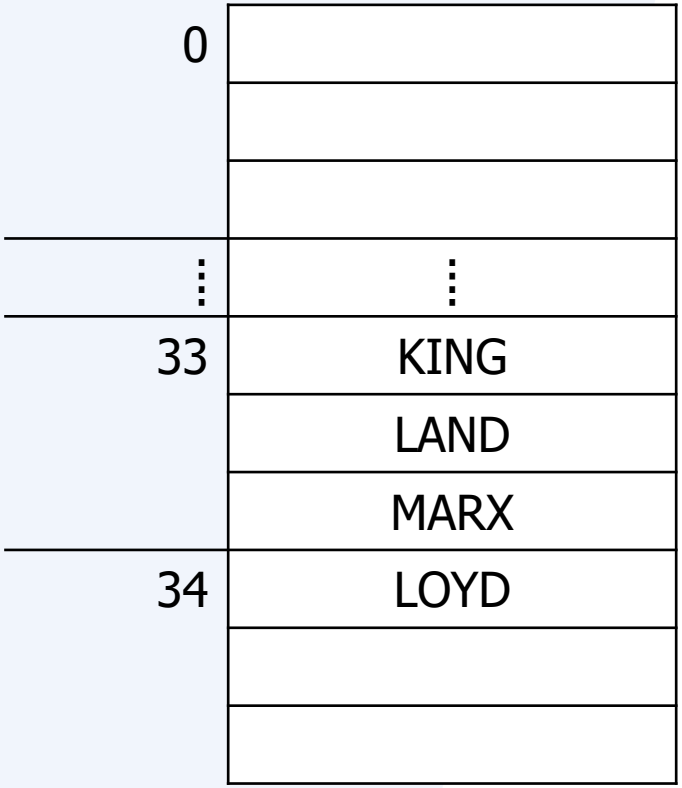
- This is a variation of hashed files in which more than one record/key is stored per hash address.
- Bucket = block of records corresponding to one address in the hash table
- The hash function gives the Bucket Address

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



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

Hashing with Buckets: Implementation issues

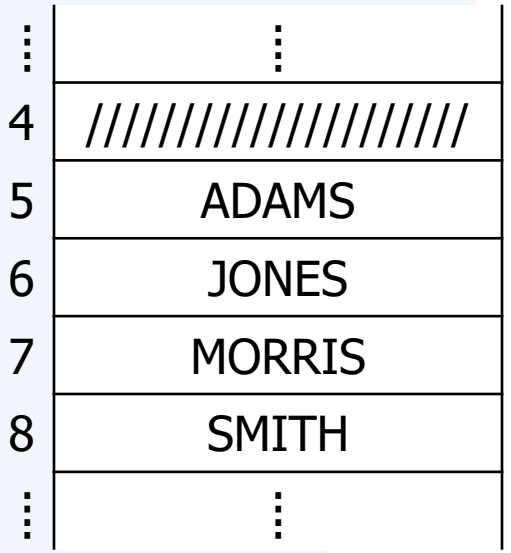
3. Loading a hash file:

- 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:

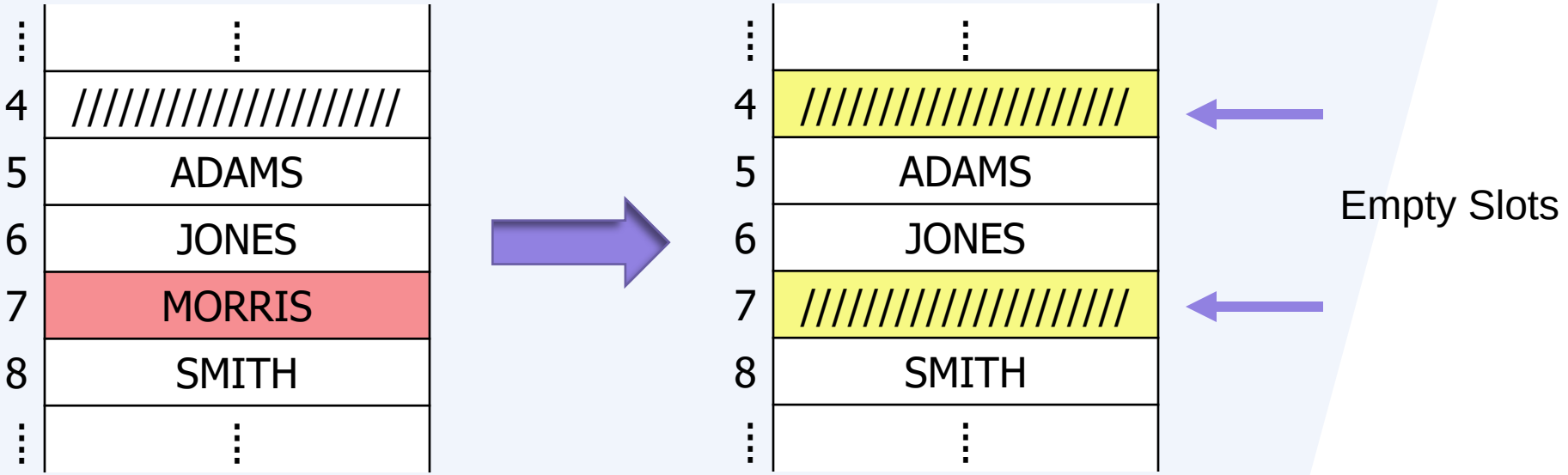
Key K	Home Address – h(k)
ADAMS	5
JONES	6
MORRIS	6
SMITH	5



Hashed File using Progressive Overflow

Making Deletions: Delete 'MORRIS'

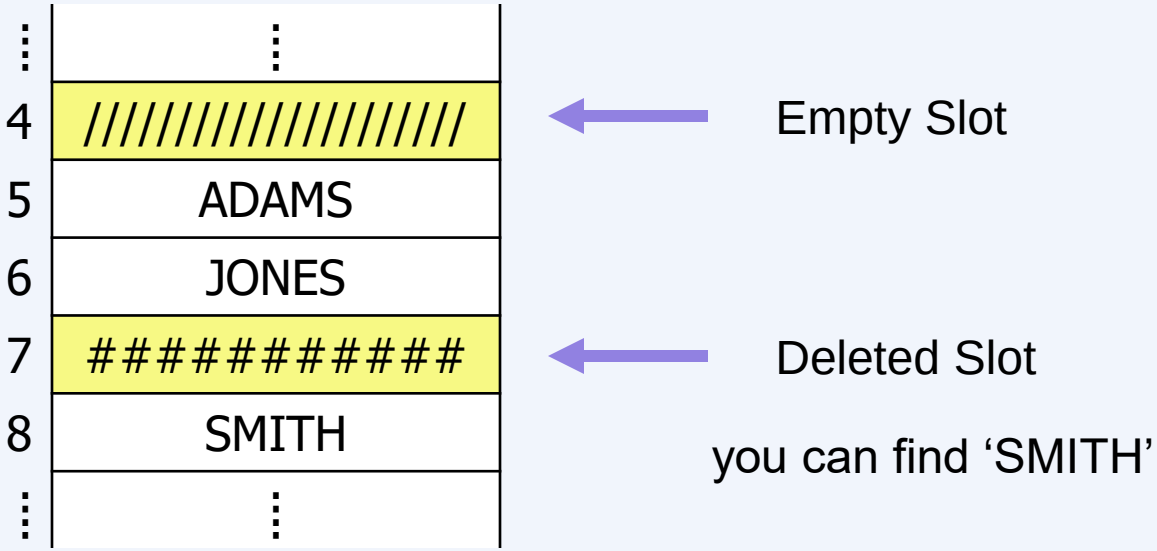
- If 'MORRIS' is simply erased, a search for 'SMITH' would be unsuccessful



- 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 Addling 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

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.

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

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

Hashed file using double hashing

Thank You !