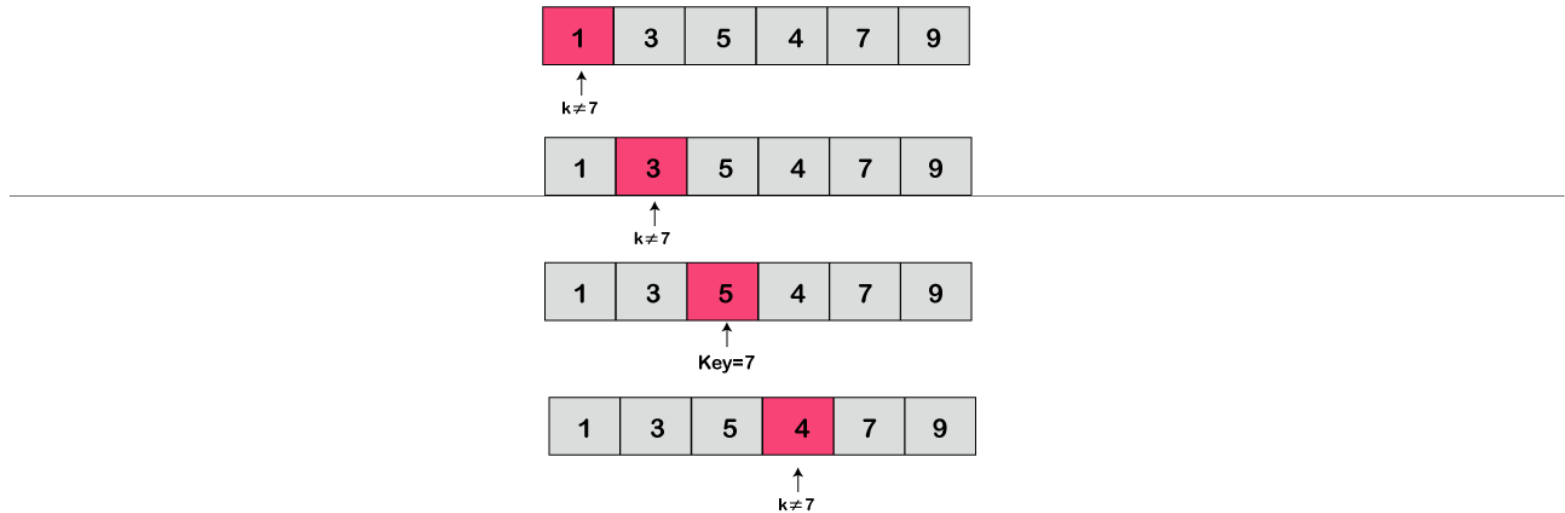


01418231

Data Structures

LECTURE-5-SEARCHING AND HASHING



Powered by Dr. Jirawan Charoensuk

Agenda

- Linear Searching
 - Unordered data
 - Sequential Search
 - Ordered data
 - Indexed Sequential Search
- Binary Search
 - Array or Linked list
- Hash Searching

Sequential Searching

- Used for not ordered list
- Used this technique for small list
- Or not search often
- Concept
 - Start search target at the beginning of list
 - Until find target or ending of list
 - Result
 - successful / unsuccessful

Algorithm Sequential Searching

```
1. Sequential_search(int a[], int n,target)
2. {
3.     int i;
4.     for (i = 0; i < n; i++)
5.     {
6.         if (target == a[i])
7.             return(i);
8.     }
9. }
```

3	27	12	52	10	48	54	78
0	1	2	3	4	5	6	7









Sequential search

target = 54

k	3	27	12	52	10	48	54	78
	0	1	2	3	4	5	6	7
	↑	↑	↑	↑	↑	↑	↑	

Sequential search

target = 99

k	3	27	12	52	10	48	54	78
	0	1	2	3	4	5	6	7
								

Concept Sequential search

Appropriate for unordered list

- Time
 - Best time = $O(1)$, target
 - Worst time = $O(n)$, target
 - Average time = $O(n/2)$

3	27	12	52	10	48	54	78
0	1	2	3	4	5	6	7

Concept Sequential search

For ordered list

- Disadvantage in unsuccessful
 - If we search until value $>$ target, we will search until ending list
 - Target

3	17	22	32	40	48	54	78
0	1	2	3	4	5	6	7

Algorithm Ordered Sequential Searching

```
1. Sequential_search(int a[], int n,target)
2. {
3.     int i;
4.     for (i = 0; i < n; i++)
5.     {
6.         if (target == a[i]) return(i);
7.         else if (target < a[i])
8.             {printf("not found"); break;}
9.     }
10. }
```

3	17	22	32	40	48	54	78
0	1	2	3	4	5	6	7

Ordered Sequential search

target = 30

k	3	17	22	32	40	48	54	78
	0	1	2	3	4	5	6	7

Agenda

Linear Searching

- Unordered data
 - Sequential Search
- **Ordered data**
 - Indexed Sequential Search

Binary Search

- Array or Linked list

Hash Searching

Indexed Sequential Searching

Appropriate for ordered list

Used this technique for large data

Adapt Sequential Searching

Store data -> array or link list

- 1) Index table
- 2) Real data

using index to search only a part of list

Indexed sequential search

sorted

Indexed sequential file:

index pointer

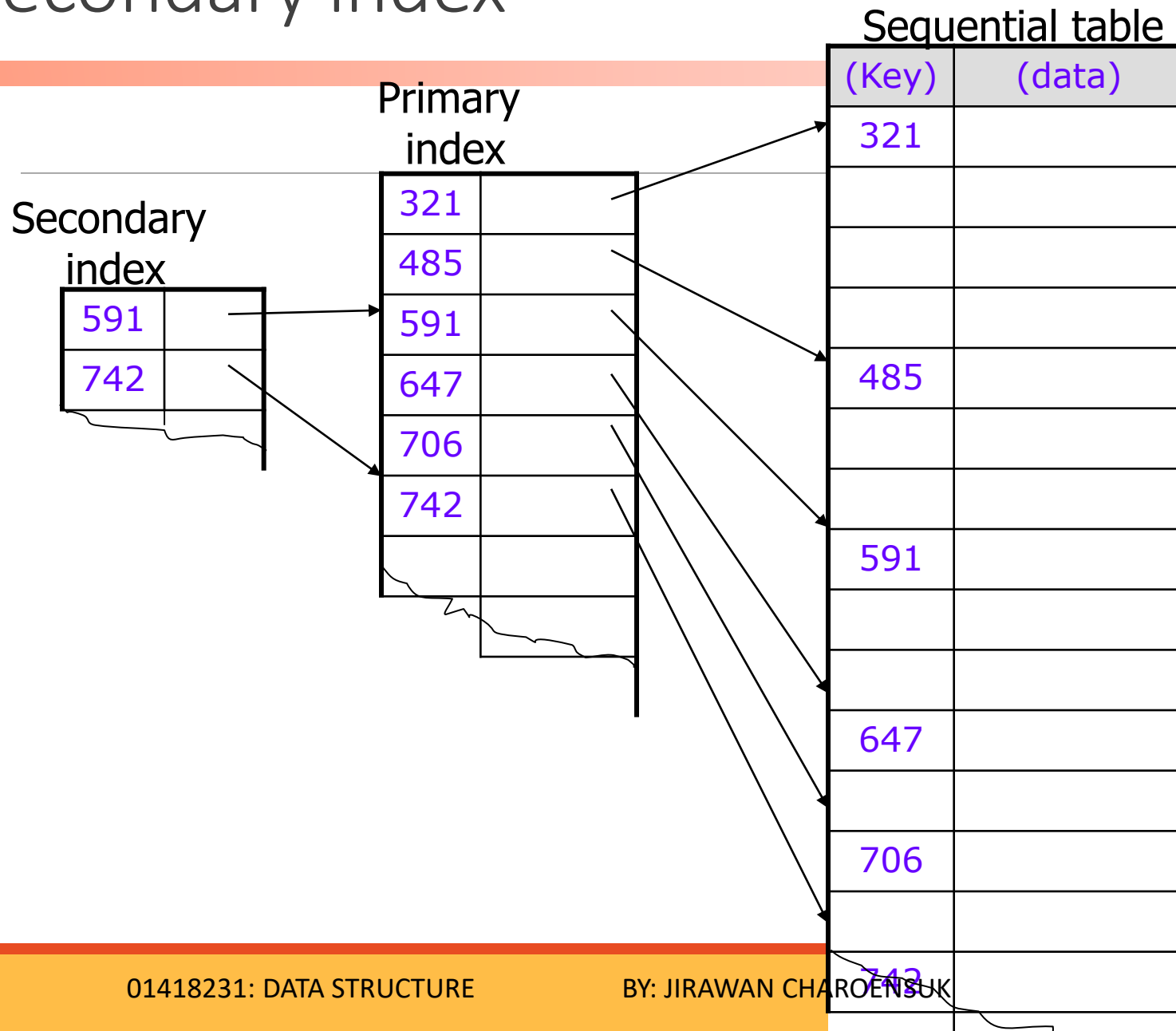
321	
592	
876	

Key = 651

Key Data

8	A
73	B
132	C
231	D
321	E
480	F
589	G
592	H
650	I
651	J
732	K
789	L
833	M
876	n

Secondary index



Agenda

Linear Searching

- Unordered data
 - Sequential Search
- Ordered data
 - Indexed Sequential Search

Binary Search

- Array or Linked list

Hash Searching

Binary search

The list must be sorted.

The list starts to become large.

- Contains more than 16 elements.

mid = (lower + upper) / 2

- Check data[mid]
 - If (k[mid] > target) check k[0]-k[mid-1]
 - Else check k[mid+1]-k[n]

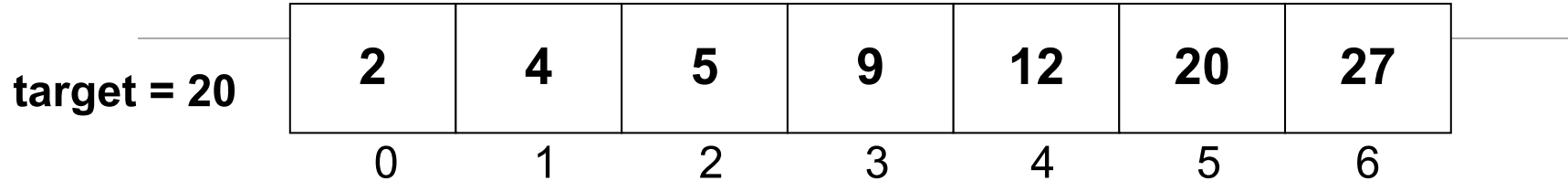
3	17	22	32	40	48	54
0	1	2	3	4	5	6

Binary Search

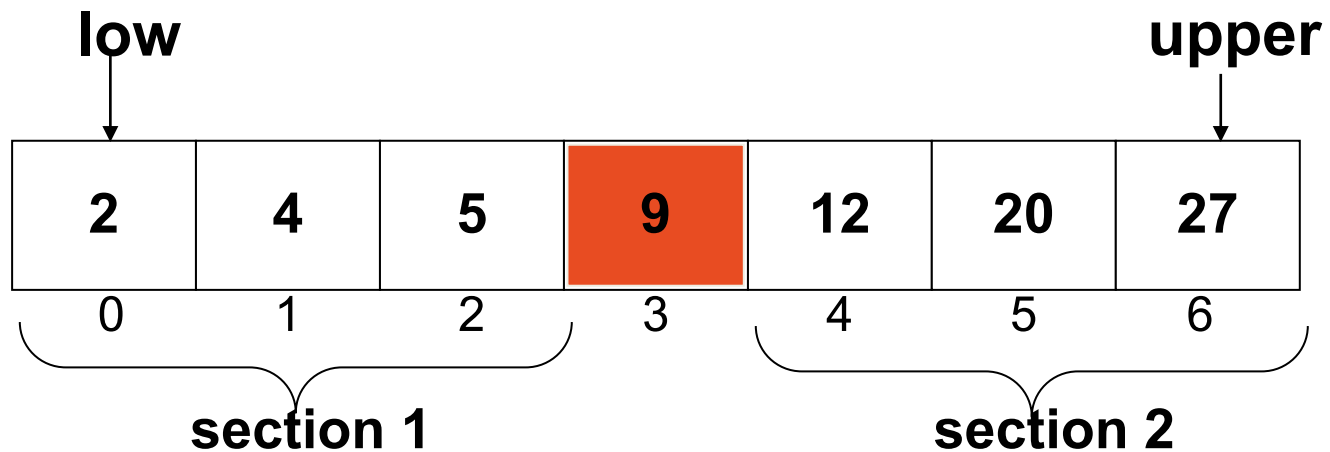
Divide and conquer methodology

- Divide the items into two parts
- Determine which part the search key belongs to and concentrate on that part
 - Keep the items sorted
 - Use the indices to delimit the part searched.

Binary Search

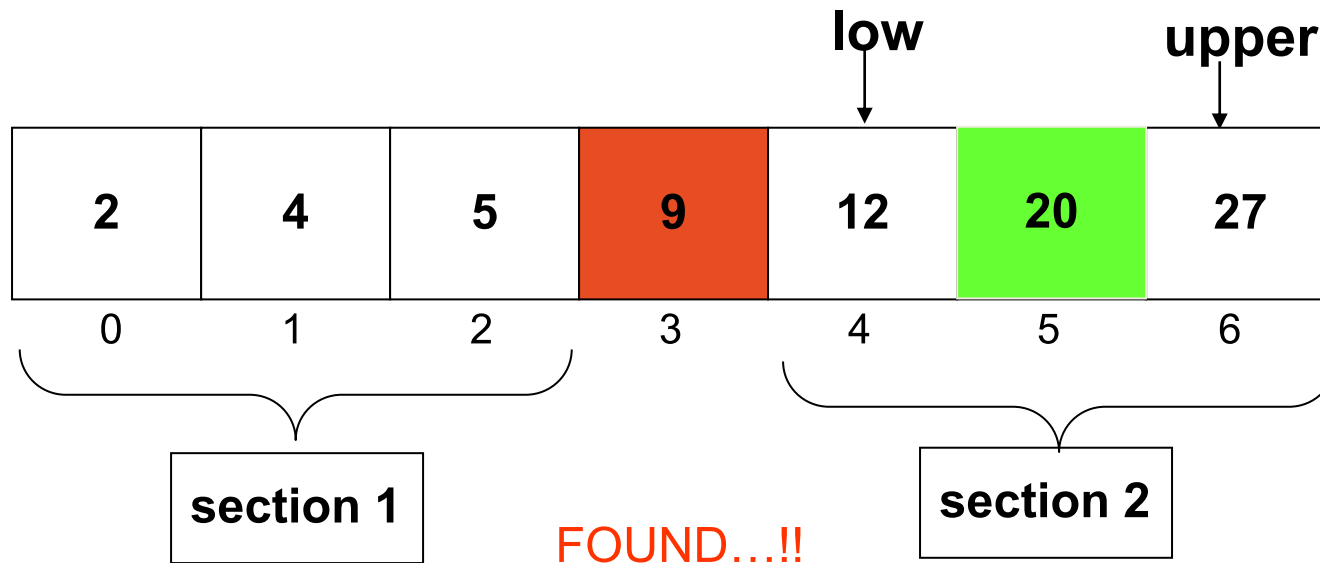


#1 -> divide into 2 section



Binary Search

#2 -> select section using target = 20 and $k[mid]$
-> divide into 2 section

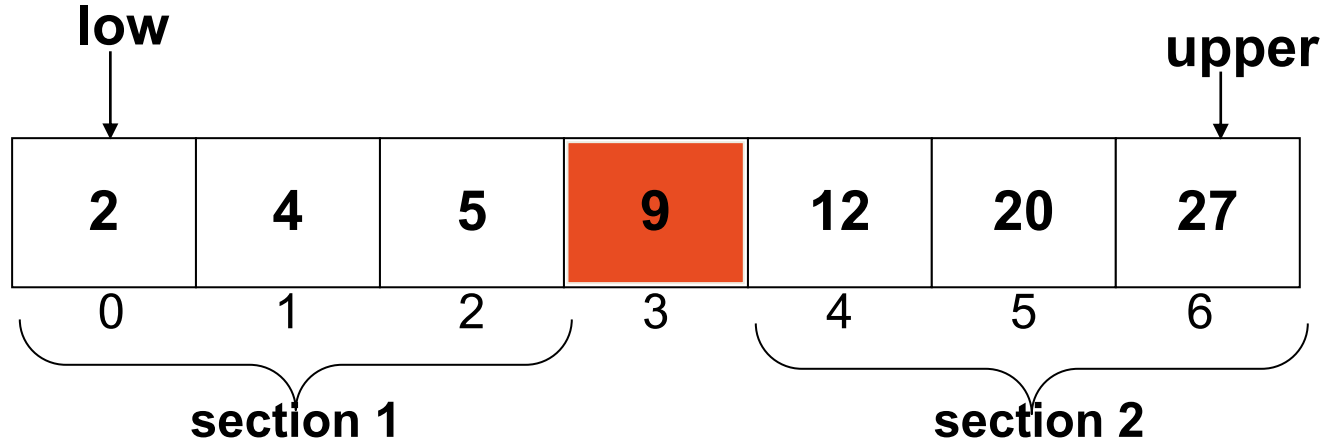


Binary Search

Key = 7

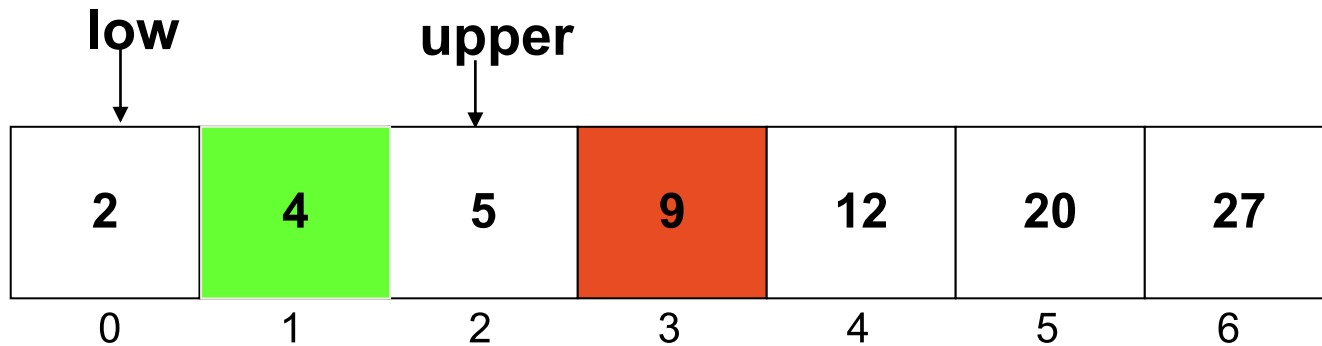
2	4	5	9	12	20	27
0	1	2	3	4	5	6

#1 -> divide into 2 section



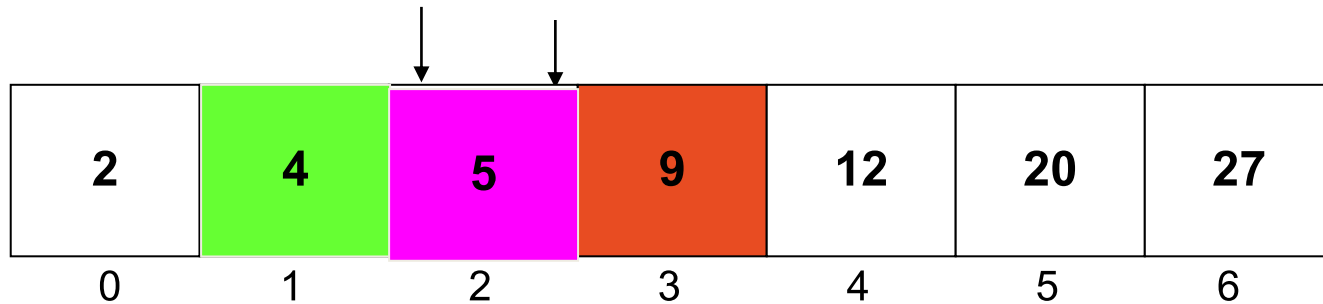
Binary Search

#2 -> select section using target = 7 and k[mid]
-> divide into 2 section



Binary Search

#3 -> compare value and target



Binary Search

กระบวนการทำงานแบบ Binary Search จะพบว่าเมื่อมีการเปรียบเทียบแต่ละครั้งจะมีการตัดข้อมูลในตารางออกไปได้ทีละครึ่งหนึ่งเสมอ ดังนั้นถ้าเริ่มต้นมีจำนวนข้อมูล n ตัว

- จำนวนข้อมูลที่นำมาเพื่อค้นหาค่าที่ต้องการหลังการเปรียบเทียบแต่ละครั้งจะเป็นดังนี้

$$\text{ครั้งที่ } 2 = n/2 \quad (\text{หรือ } n/2^1)$$

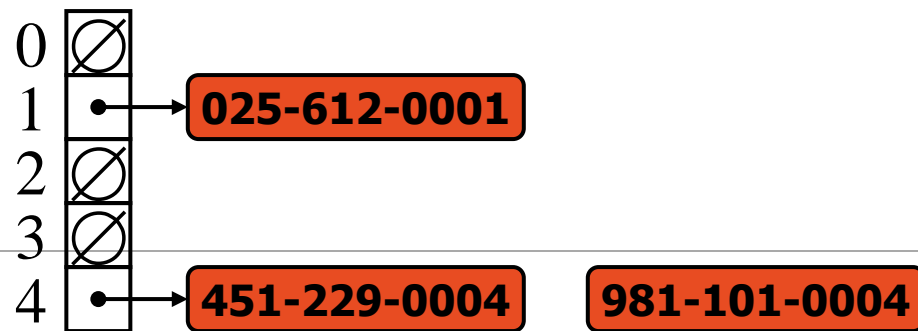
$$\text{ครั้งที่ } 3 = n/4 \quad (\text{หรือ } n/2^2)$$

$$\text{ครั้งที่ } 4 = n/8 \quad (\text{หรือ } n/2^3)$$

Binary Search Algorithm

```
1.  Binary_search(int k[], int n,key)
2.  {
3.      found = false;
4.      low = 0; upper = n-1 ;
5.      while (low <= upper ) and (not found)
6.      {
7.          mid = (low+ upper ) / 2;
8.          if (key == k[mid])  found = true; return(k[mid]); break;
9.          if key < k[mid])  upper = mid - 1
10.         else low = mid + 1
11.     }
12.     if (key != k[mid]) print ("not found");
13. }
```


HASHING SEARCHING



Agenda

Hash table

Hash function

1. The division method
2. The multiplication method
3. Folding method

Collision handling

1. Open Hashing or Separate Chain
2. Closed Hashing or Open Addressing

Summary

Hashing

Another important and widely useful technique for implementing **dictionaries**

Constant time per operation (**on the average**)

Worst case time proportional to the size of the set for each operation (just like array and chain implementation)

Dictionary

Dynamic-set data structure for **storing items indexed using keys**

Maintain a set of items with distinct keys with:

- *find* (*k*): find item with key *k*
- *insert* (*x*): insert item *x* into the dictionary
- *remove* (*k*): delete item with key *k*

Where do we use them:

- Customer records (access by id or name)
- Games (positions, configurations)
- Spell checkers (dictionary)

Basic Idea of Hashing

Use *hash function* to map *keys* into positions in a *hash table*

Ideally

If element e has key k and h is hash function, then e is stored in position $h(k)$ of table

To search for e , compute $h(k)$ to locate position.

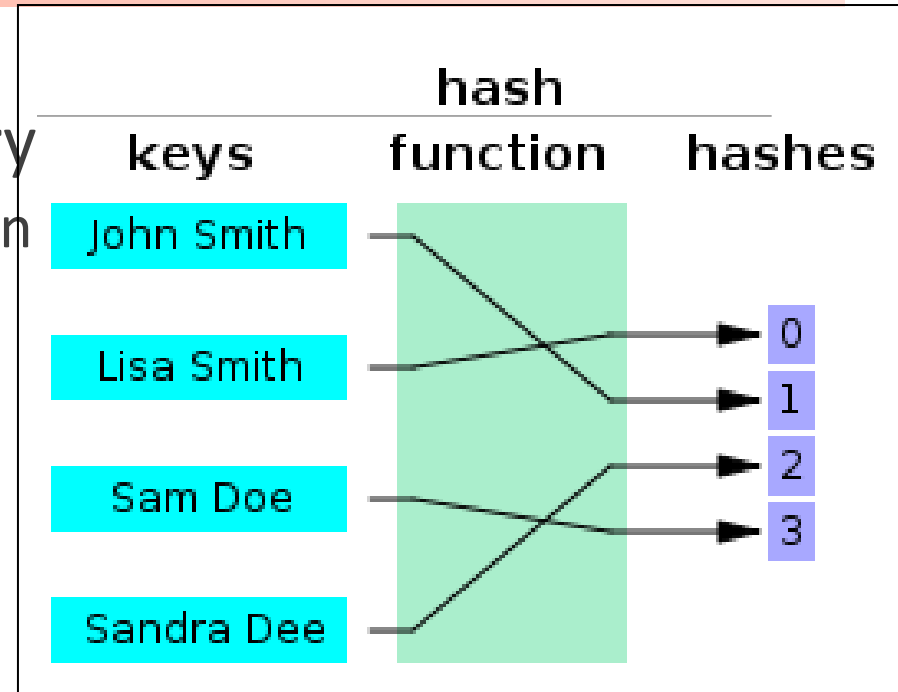
- If no element, dictionary does not contain e

Hash Tables

Hashing is function that maps each key to a location in memory

- A key's location does not depend on other elements
- and does not change after insertion.
 - unlike a sorted list

A good hash function should be easy to compute



3	10	12	52
---	----	----	----

15 New node

Hash Tables

Notation:

- U – Universe of all possible keys.
- K – Set of keys actually stored in the dictionary.

When U is very large,

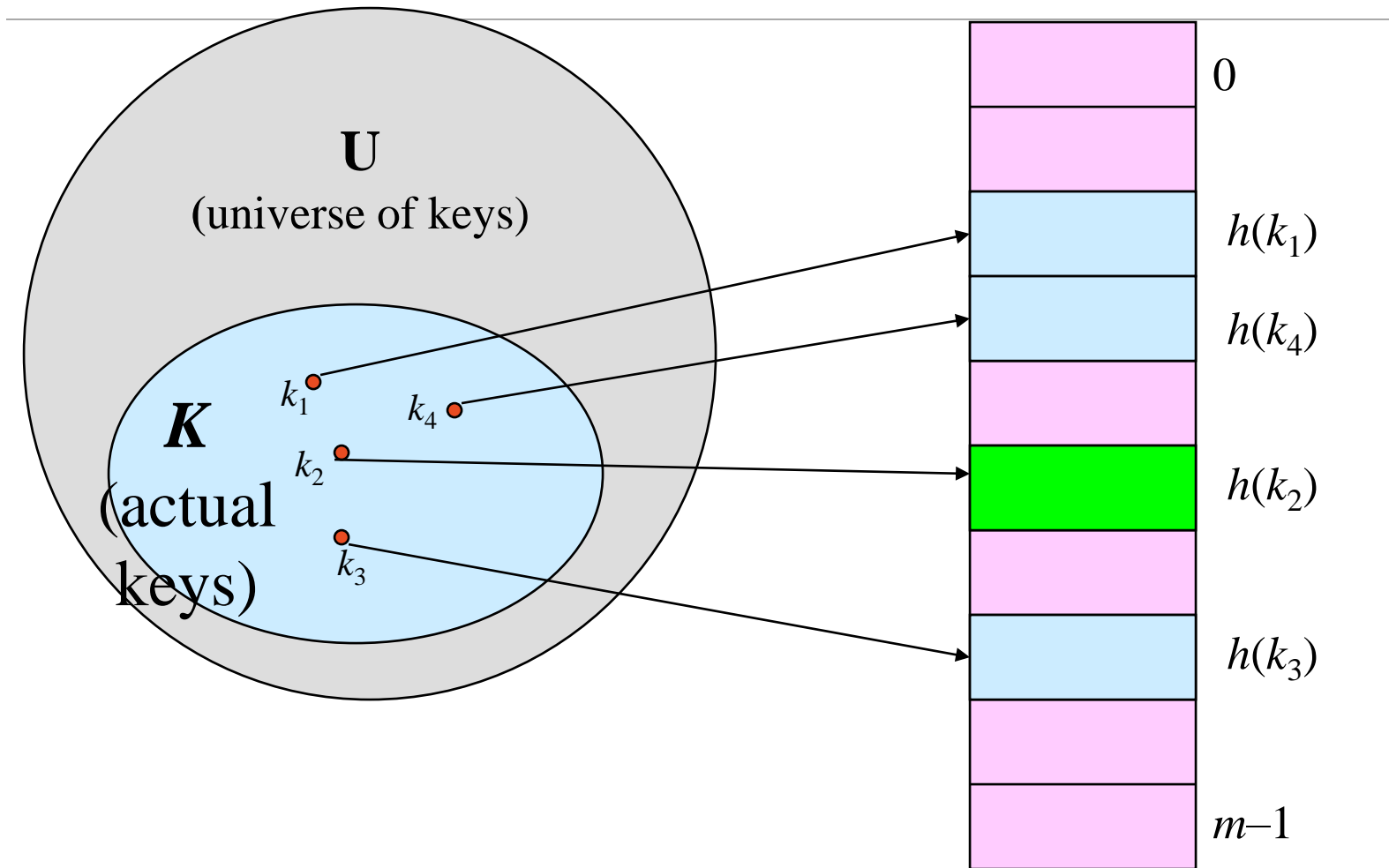
- Arrays are not practical.

Use a table of size proportional to $|K|$ – The hash tables.

- However, we lose the direct-addressing ability.
- Define functions that map keys to slots of the hash table.

Hashing

$T [0..m-1]$
(hash table)



Hashing

Hash function (h): Mapping from U to the slots of a hash table T [0..m-1]

$$h: U \rightarrow \{0, 1, \dots, m-1\}$$

With arrays, key k maps to slot T[0..m-1]

With hash tables, key k maps or “**hashes**” to slot T [**h**[k]]

h[k] is the *hash value* of key k

Hashing

Hash table

1
2
3
8
9
13
14

Ann

Student Records

Bell

Cathy

Dave

Natty

Ken

Tod

Example of Hashing

<http://research.cs.vt.edu/AVresearch/hashing/introduction.php>

Hash Table

0	
1	
2	
3	

Enter the value to be inserted

Input must be between 0 and 3

Hash Table

0	
1	1
2	
3	

Enter the value to be inserted

Input must be between 0 and 3

Placing the element in its slot

$h(1) = 1$

Enter another value

Hash Table

0

1

2

3

1
3

Enter the value to be inserted

Input must be between 0 and 3

Placing the element in its slot

$$h(1) = 1$$

Enter another value

Placing the element in its slot

$$h(3) = 3$$

Enter another value

Input = 3

Enter another value

Value 3 is already in table

No duplicates are allowed

Hashing : the basic idea

Map key values to hash table addresses

keys -> hash table address

This applies to find, insert, and remove

Usually: *integers* -> $\{0, 1, 2, \dots, H_{\text{size}-1}\}$

Non-numeric keys converted to numbers

- *For example, strings converted to numbers as*
 - Sum of ASCII values
 - First three characters

Method for Creating Hash Function

have 4 methods

- Simple Mod Function
- The multiplication method
- Folding method
- Hash Functions for Strings

Method for Creating Hash Function

have 4 methods

- Simple Mod Function
- The multiplication method
- Folding method
- Hash Functions for Strings

1. Simple Mod Function

Map a key k into one of m slots by taking the remainder of k divided by m

$$h(k) = k \bmod m_{\text{size}}$$

Example:

- If table size $m = 12$, key $k = 100$
 - $h(100) = 100 \bmod 12 = 4$

1. Simple Mod Function

hashing table

Key	$H(k) = k \bmod 100$		
60102	60102 mod 100	00	46200
46200	46200 mod 100	01	10201
37505	37505 mod 100	02	60102
10275	10275 mod 100	03	
60199	60199 mod 100	04	
21676	21676 mod 100	05	37505
10201	10201 mod 100	...	
		75	10275
		76	21676
		...	
		98	
		99	60199

Method for Creating Hash Function

have 4 methods

- Simple Mod Function
- **The multiplication method**
- Folding method
- Hash Functions for Strings

2. The multiplication method

Concat, Square and Remove the Middle!

Example:

- Table size [0..99]
- A..Z ---> 1,2, ...26
- 0..9 ----> 27,...36
- Identifier: CS1 ---> 3+19+28 (concat) = 31,928
- $(31,928)^2 = 1,019,397,184$ - 10 digits
- extract middle 2 digits (5th and 6th) -> 1,019,**397**,184
- $H(\text{CS1}) = 39$

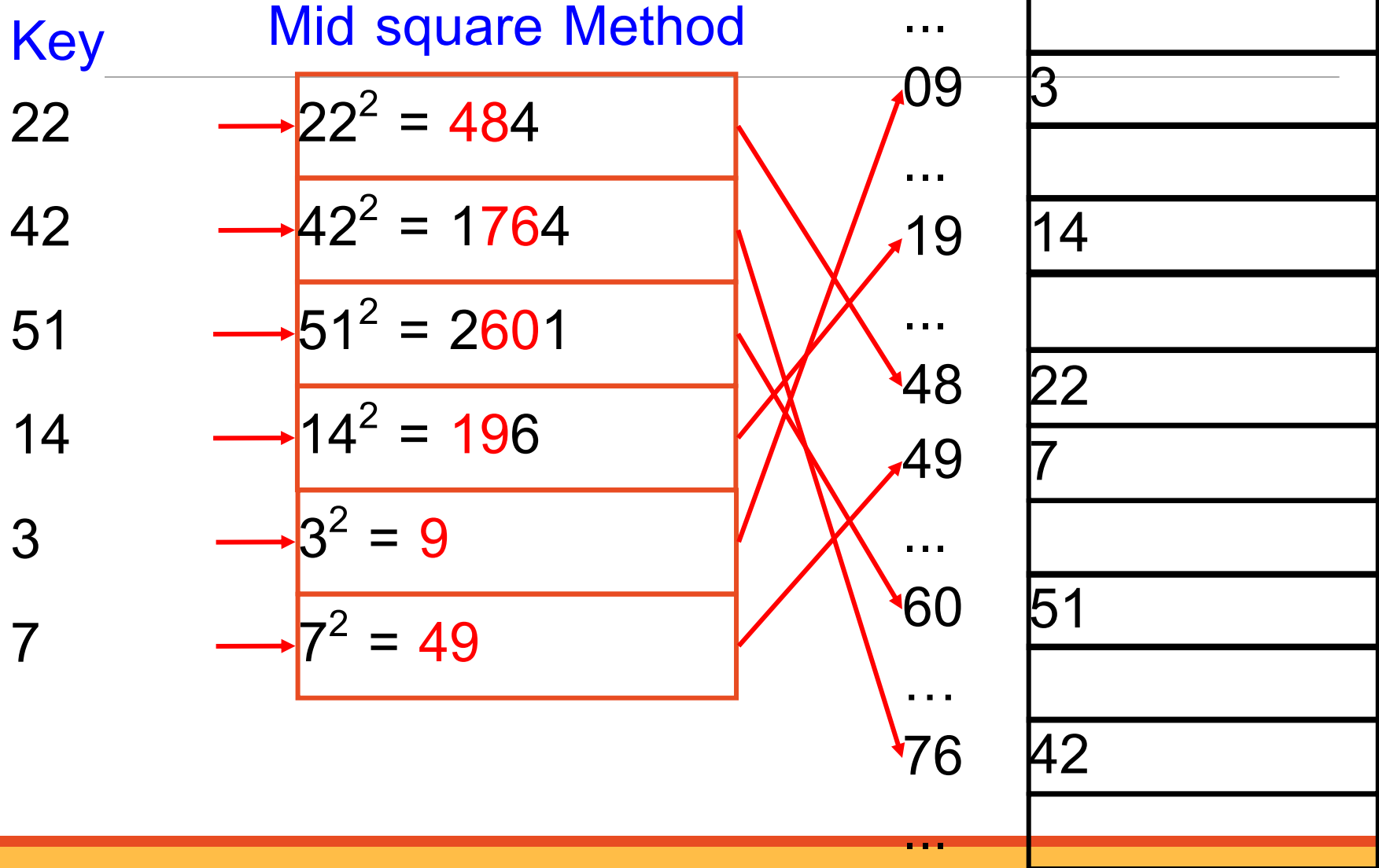
2. The multiplication method

Key -> 22 42 51 14 3 7

Size(hash table) = 100 = (00 – 99)

key	key ²	index
22	484	48
42	1764	76
51	2601	60
14	196	19
3	9	09
7	49	49

2. The multiplication method hashing table



Method for Creating Hash Function

have 4 methods

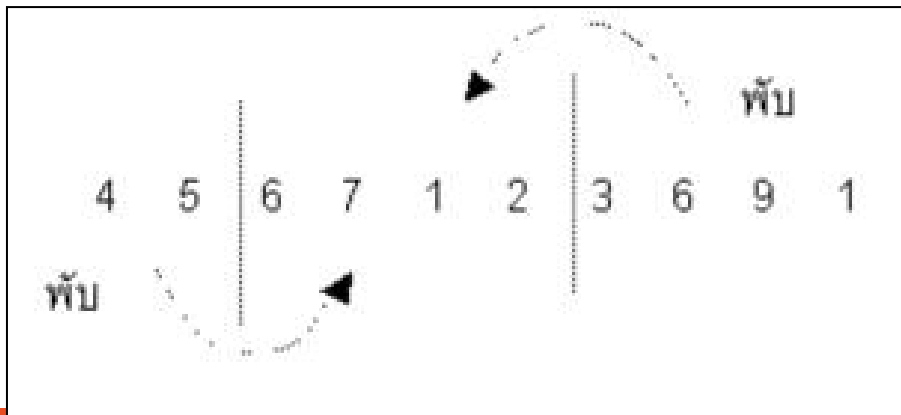
- Simple Mod Function
- The multiplication method
- **Folding method**
- Hash Functions for Strings

3. Folding method

Algorithm

1. break key up into **segments** (depend on the number of digits)
2. **XOR** these together
3. Calculate the numeric integer equivalent

Example : **4567123691** convert to 4 digits



6	7	1	2	
1	9	6	3	+
5	4			
<hr/>				
1	4	0	7	5
<hr/>				

47

3.Folding method

Example: convert to 2 digits

- $H(3205) = 32 + 05 = 37$,
- $H(7148) = 71 + 48 = (1)19$
- $H(2345) = 23 + 45 = 68$

Method for Creating Hash Function

have 4 methods

- Simple Mod Function
- The multiplication method
- Folding method
- Hash Functions for Strings

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
64	40	100	@	@	96	60	140	`	`
65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f
71	47	107	G	G	103	67	147	g	g
72	48	110	H	H	104	68	150	h	h
73	49	111	I	I	105	69	151	i	i
74	4A	112	J	J	106	6A	152	j	j
75	4B	113	K	K	107	6B	153	k	k
76	4C	114	L	L	108	6C	154	l	l
77	4D	115	M	M	109	6D	155	m	m
78	4E	116	N	N	110	6E	156	n	n
79	4F	117	O	O	111	6F	157	o	o
80	50	120	P	P	112	70	160	p	p
81	51	121	Q	Q	113	71	161	q	q
82	52	122	R	R	114	72	162	r	r
83	53	123	S	S	115	73	163	s	s
84	54	124	T	T	116	74	164	t	t
85	55	125	U	U	117	75	165	u	u
86	56	126	V	V	118	76	166	v	v
87	57	127	W	W	119	77	167	w	w
88	58	130	X	X	120	78	170	x	x
89	59	131	Y	Y	121	79	171	y	y
90	5A	132	Z	Z	122	7A	172	z	z
91	5B	133	[[123	7B	173	{	{
92	5C	134	\	\	124	7C	174	|	
93	5D	135]]	125	7D	175	}	}
94	5E	136	^	^	126	7E	176	~	~
95	5F	137	_	_	127	7F	177		DEL

4. Hash Functions for Strings

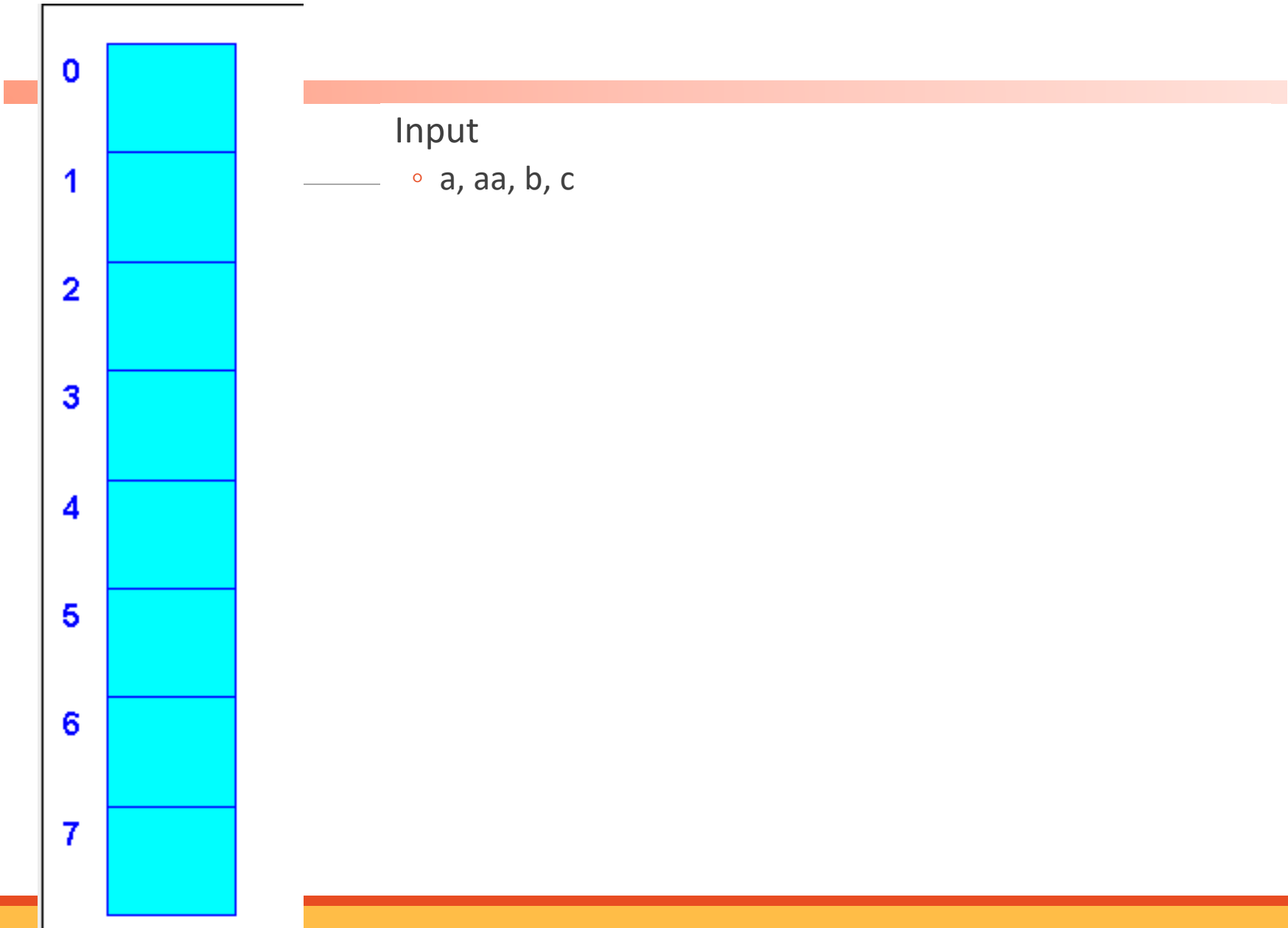
1. Sum value of Ascii code
2. Use function mod

$$h(k) = k \bmod m_{\text{size}}$$

Example size hash table = 8

a = 97, $97\%8 = 1$

aa = $97+97 = 194\%8 = 2$



0

Input a

o

1

← a

Input aa

2

← aa

o

Input b

3

← b

o

4

c

o

Input c

5

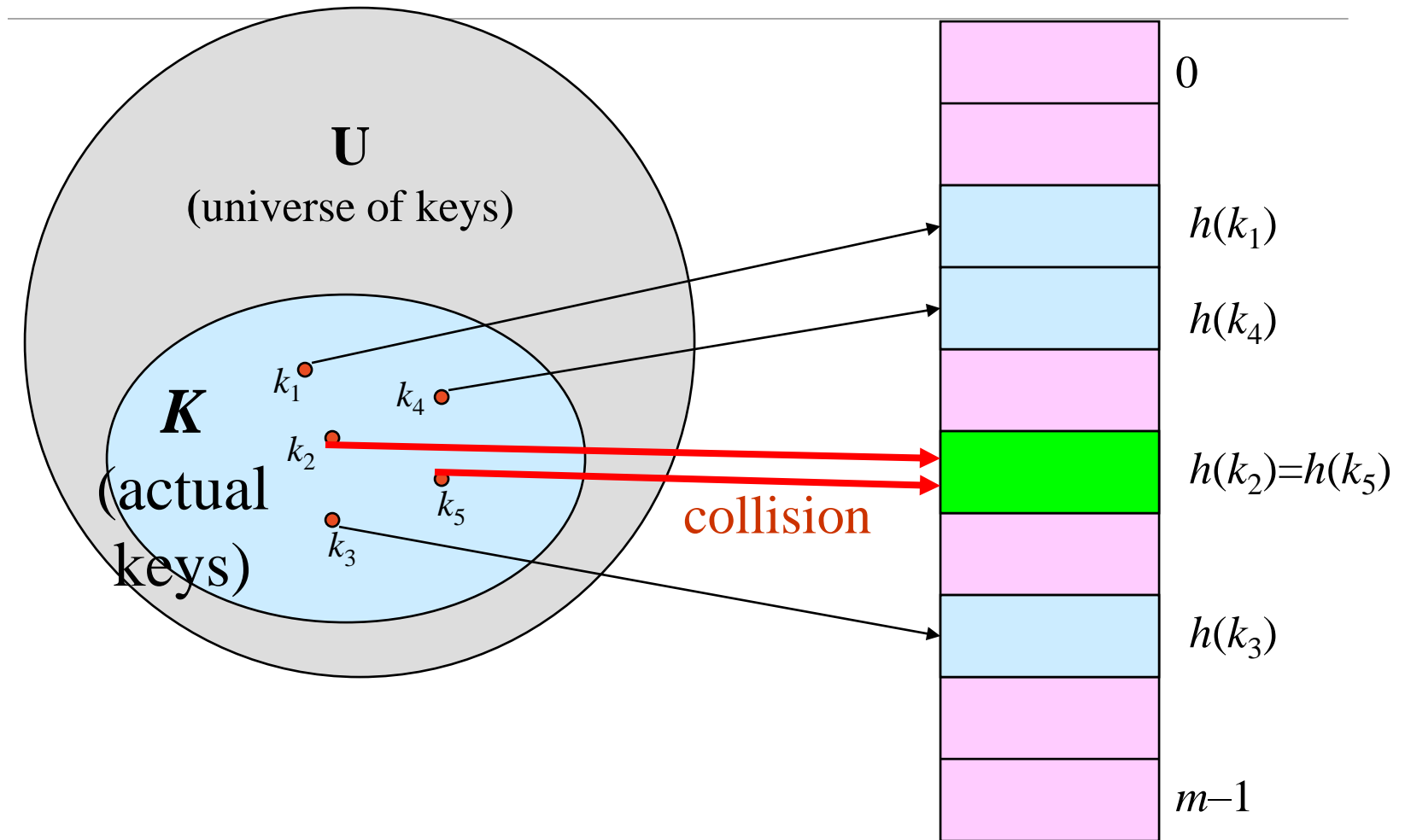
o

6

7

Collision Hashing

$T [0..m-1]$
(hash table)



Collision handling

Solve collision hashing with 2 main methods

1. **Open Hashing or Separate Chain**
2. **Closed Hashing or Open Addressing**
 - Linear Probing
 - Quadratic Probing
 - Double Hashing
 - Rehashing

Collision handling

1. **Open Hashing or Separate Chain**
2. Closed Hashing or Open Addressing
 - Linear Probing
 - Quadratic Probing
 - Double Hashing
 - Rehashing

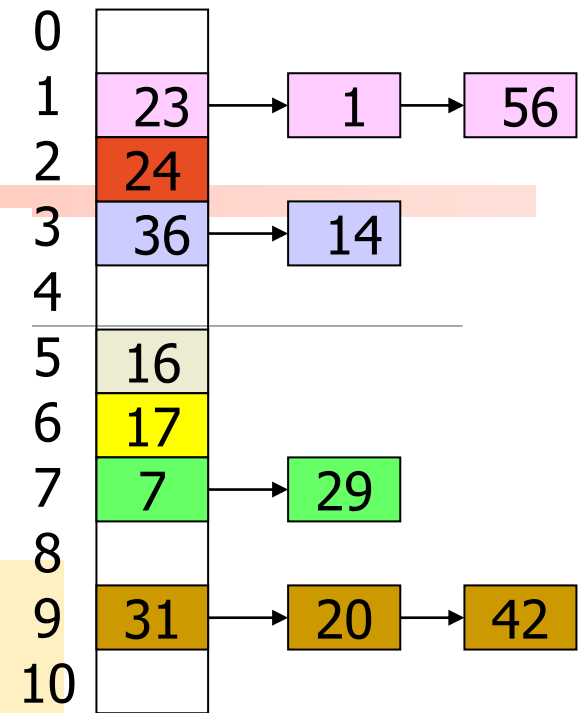
Separate chaining

Maintain a list of all elements that hash to the same value

Search -- using the hash function to determine which list to traverse

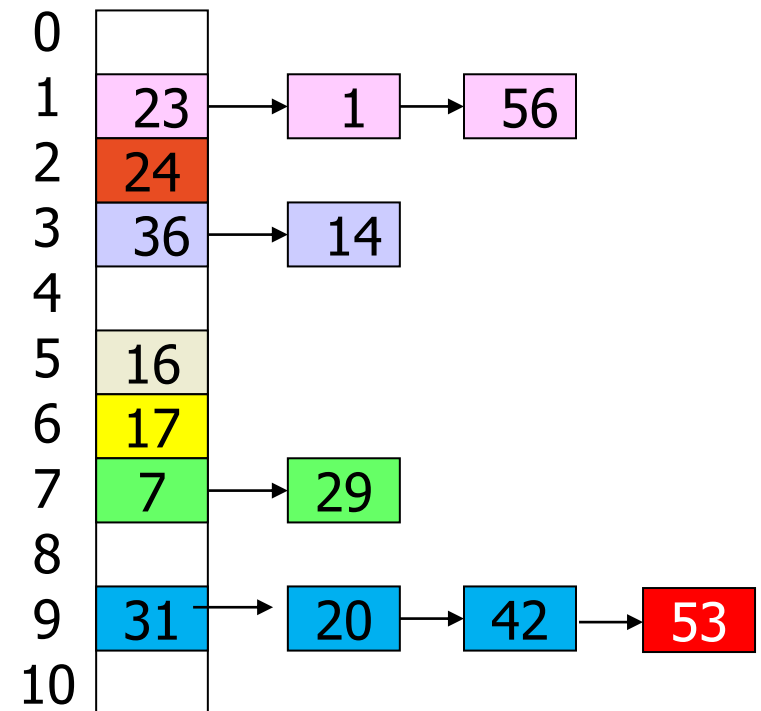
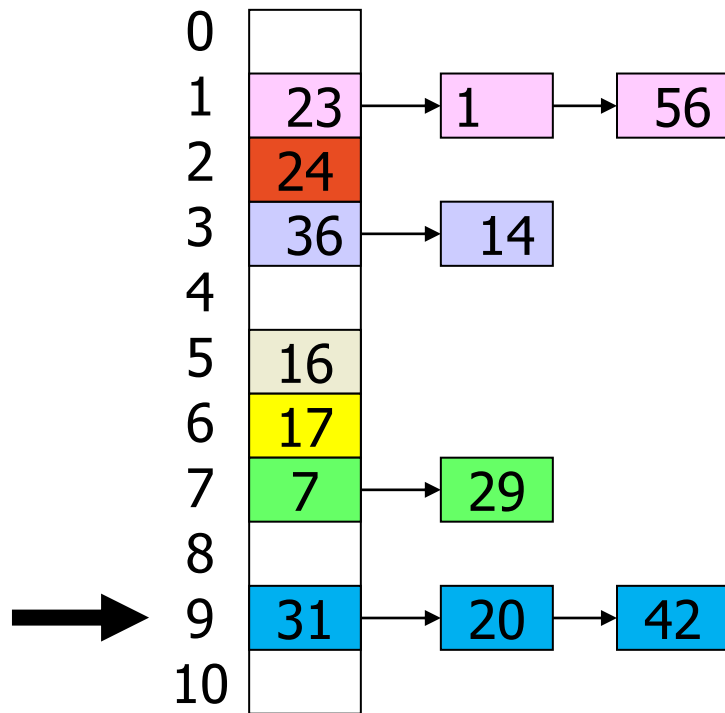
```
find(k,e)
    HashVal = Hash(k,Hsize);
    if (TheList[HashVal].Search(k,e))
        then return true;
    else return false;
```

Insert/deletion—once the “bucket” is found through *Hash*, insert and delete are list operations



Separate chaining : insert 53

$$53 \bmod 11 = 9$$



Separate chaining

0

1

2

3

4

5

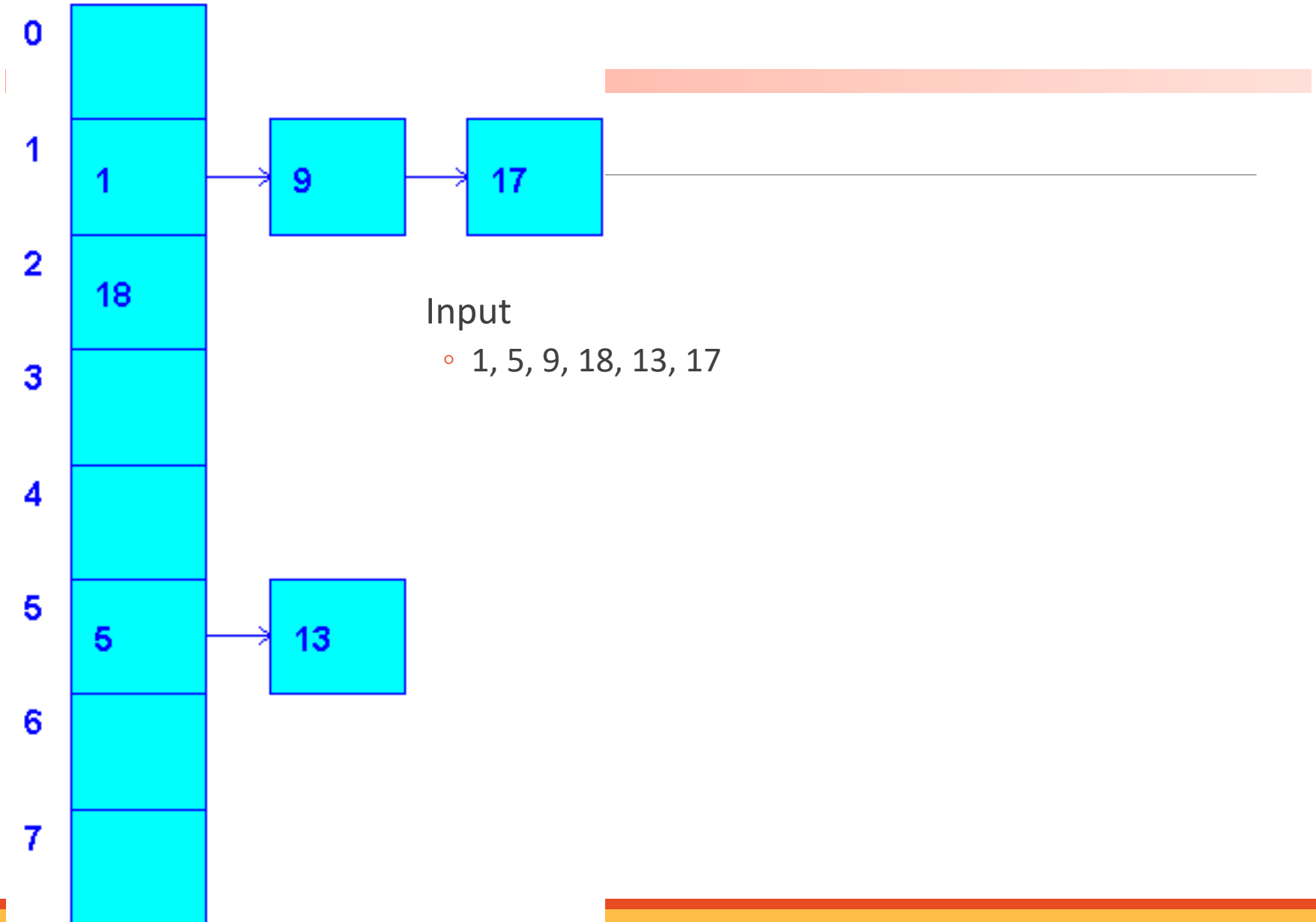
6

7

Input

○ 1, 5, 9, 18, 13, 17

Op



Collision handling

1. Open Hashing or Separate Chain
2. **Closed Hashing or Open Addressing**
 - **Linear Probing**
 - **Quadratic Probing**
 - **Double Hashing**
 - **Rehashing**

2.Open Addressing

If collision happens, alternative cells are tried until an empty cell is found.

Linear probing :
Try next available position

0	42
1	1
2	24
3	14
4	
5	16
6	28
7	7
8	
9	31
10	9

Linear Probing : insert 12

$$12 \bmod 11 = 1$$

0	42
1	1
2	24
3	14
4	
5	16
6	28
7	7
8	
9	31
10	9

0	42
1	1
2	24
3	14
4	12
5	16
6	28
7	7
8	
9	31
10	9

Linear probing : Search 15

$$15 \bmod 11 = 4$$

0	42
1	1
2	24
3	14
4	12
5	16
6	28
7	7
8	
9	31
10	9

Linear probing: Delete 9

$$9 \bmod 11 = 9$$

0	42
1	1
2	24
3	14
4	12
5	16
6	28
7	7
8	
9	31
10	9

0	42
1	1
2	24
3	14
4	12
5	16
6	28
7	7
8	
9	31
10	D

Quadratic Probing

Solves the clustering problem in Linear Probing

- Check $H(x)$
- If collision occurs check $(H(x) + 1) \% m_{\text{size}}$
- If collision occurs check $(H(x) + 4) \% m_{\text{size}}$
- If collision occurs check $(H(x) + 9) \% m_{\text{size}}$
- If collision occurs check $(H(x) + 16) \% m_{\text{size}}$
- ...
- $H(x) + i^2$

Quadratic Probing (insert 12)

$$12 \bmod 11 = 1$$

0	42
1	1
2	24
3	14
4	
5	16
6	28
7	7
8	
9	31
10	9

0	42
1	1
2	24
3	14
4	12
5	16
6	28
7	7
8	
9	31
10	9

Double Hashing

When collision occurs use a second hash function

- $\text{Hash}_2(x) = R - (x \bmod R)$
- R: greatest prime number smaller than table-size

Inserting 12

$$H_2(x) = 7 - (x \bmod 7) = 7 - (12 \bmod 7) = 2$$

- Check $H(x)$
- If collision occurs check $H(x) + 2$
- If collision occurs check $H(x) + 4$
- If collision occurs check $H(x) + 6$
- If collision occurs check $H(x) + 8$
- $H(x) + i * H_2(x)$

Double Hashing (insert 12)

$$12 \bmod 11 = 1$$
$$7 - (12 \bmod 7) = 7 - 5 = 2$$

0	42
1	1
2	24
3	14
4	
5	16
6	28
7	7
8	
9	31
10	9

0	42
1	1
2	24
3	14
4	12
5	16
6	28
7	7
8	
9	31
10	9

Rehashing

If table gets too full, operations will take too long.

Build another table, twice as big (and prime).

- Next prime number after 11×2 is 23

Insert every element again to this table

Rehash after a percentage of the table becomes full (70% for example)

Rehashing

0	6
1	15
2	
3	24
4	
5	
6	13

Figure 1

0	6
1	15
2	23
3	24
4	
5	
6	13

Figure 2

70% of table size
=5

Figure 3

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

Good and Bad Hashing Functions

Hash using the wrong key

- Age of a student

Hash using limited information

- First letter of last names (a lot of A's, few Z's)

Hash functions choices :

- keys evenly distributed in the hash table

Even distribution guaranteed by “randomness”

- No expectation of outcomes
- Cannot design input patterns to defeat randomness

Summary

Summary

Linear Searching

- Unordered data
 - Sequential Search
- Ordered data
 - Indexed Sequential Search

Binary Search

- Array or Linked list

Hash Searching

Hash Searching

METHOD FOR CREATING HASH FUNCTION

have 4 methods

- Simple Mod Function
- The multiplication method
- Folding method
- Hash Functions for Strings

COLLISION HANDLING

Open Hashing or Separate Chain

Closed Hashing or Open Addressing

- Linear Probing
- Quadratic Probing
- Double Hashing
- Rehashing

Collision handling

1. Open Hashing or Separate Chain
2. Closed Hashing or Open Addressing
 - Linear Probing
 - Quadratic Probing
 - Double Hashing
 - Rehashing

Question



[This Photo](#) by Unknown Author is licensed under [CC BY-NC](#)