Outline

- Static hashing
  - Division
  - Mid square
  - Folding
  - Digit analysis
- Overflow handling
  - Open addressing
  - Chaining

Overflow Handling

- An overflow occurs when bucket is full
- We may handle overflows by
  1. Open addressing
     - Search the hash table in systematic fashion to exploit available buckets (and slots)
  2. Chaining
     - Eliminate overflows by permitting each bucket to keep all pairs in a linked list

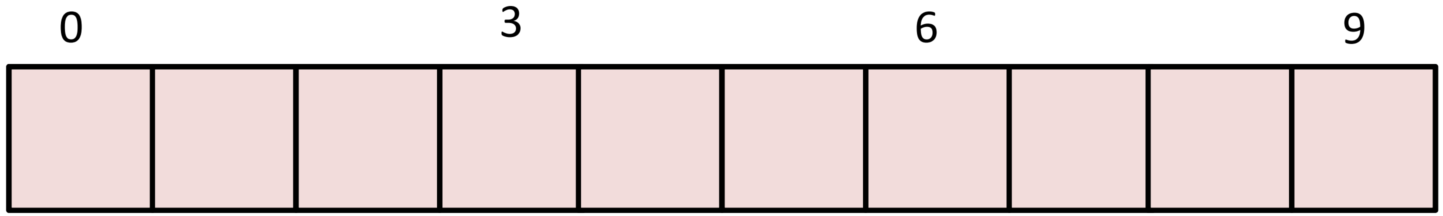
Linear Probing

- Find available bucket by examining *ht[(h(k)+j)%b]* for *j=0, 1, 2, ..., b-1*

- Hash table operations
  - Insert: find empty bucket
  - Search: find matching key; If empty, key is not in the table
  - Delete: delete matching key

Linear Probing

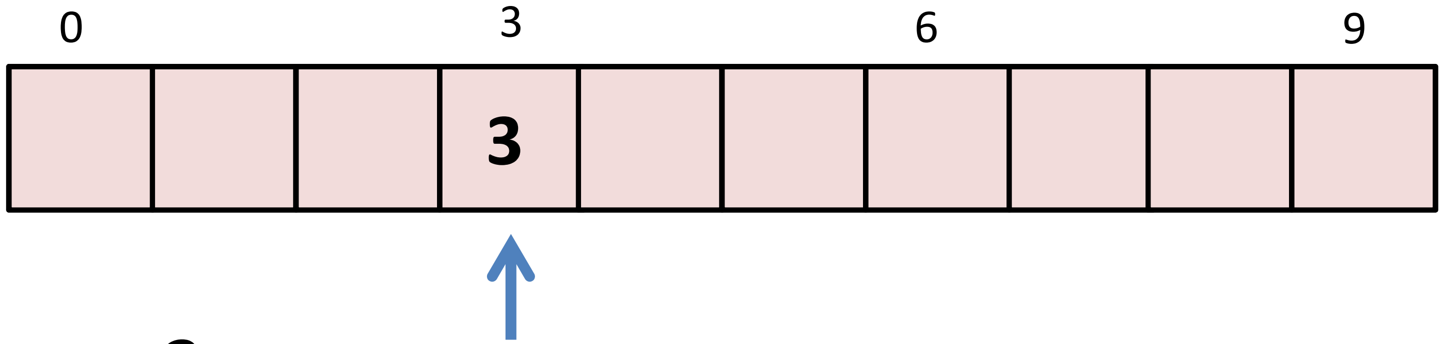
- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1



ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Linear Probing

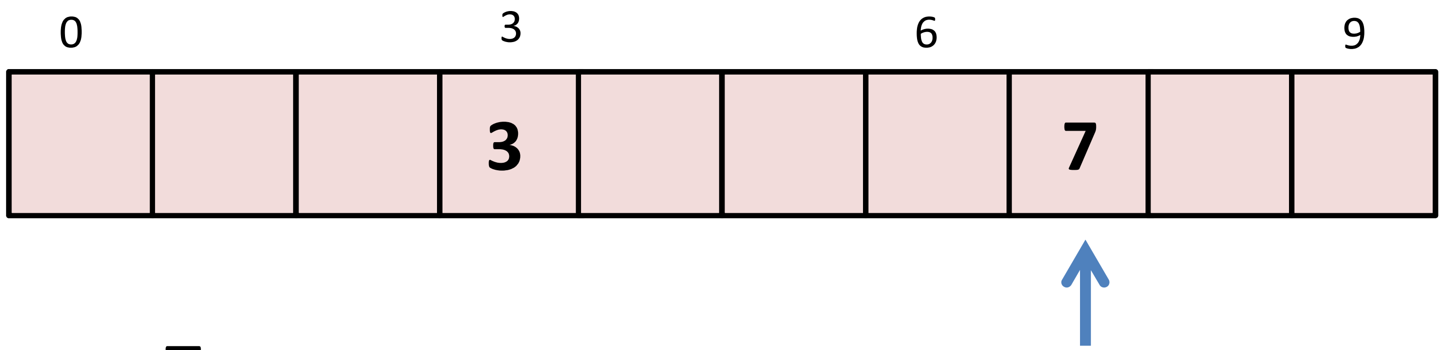
- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1



- Insert 3
  – 3%10=3

Linear Probing

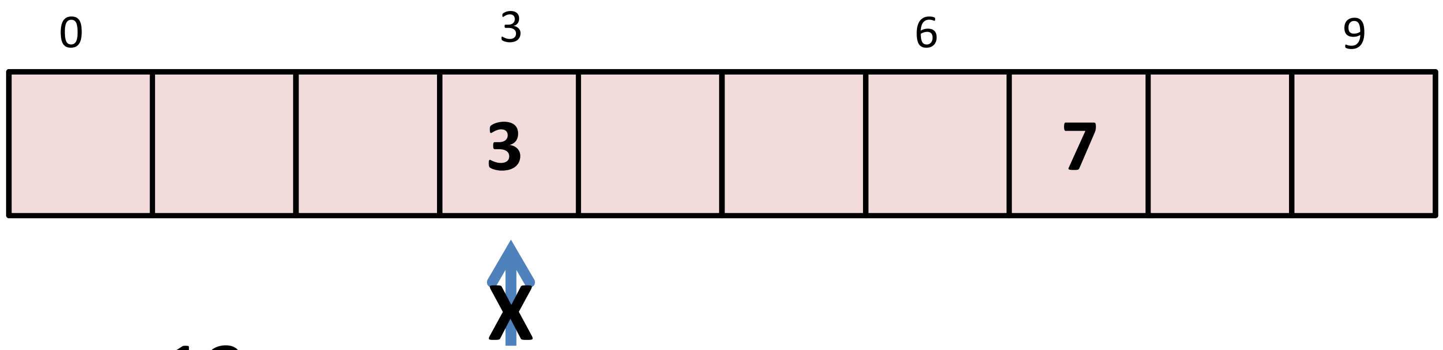
- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1



- Insert 7
  - 7%10=7

Linear Probing

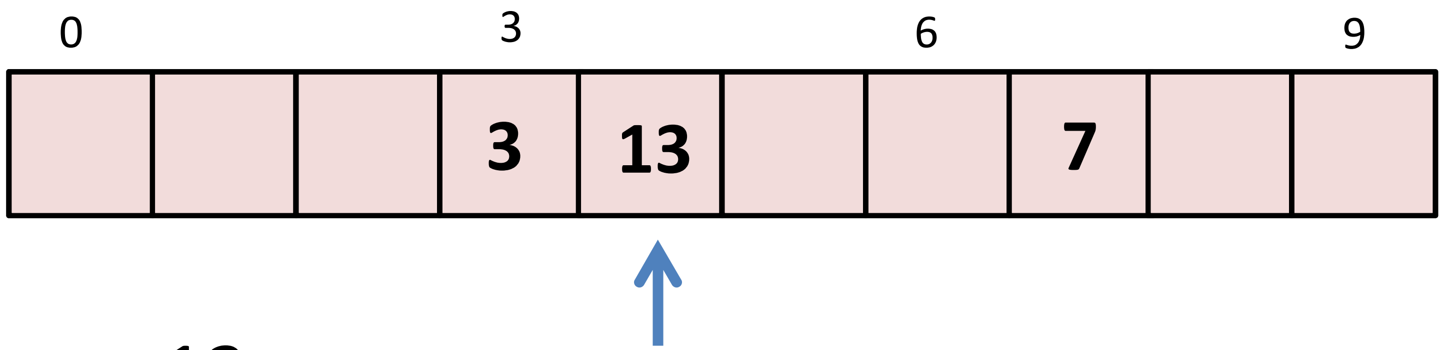
- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1



- Insert 13
  - 13%10=3 → collision & overflow!

Linear Probing

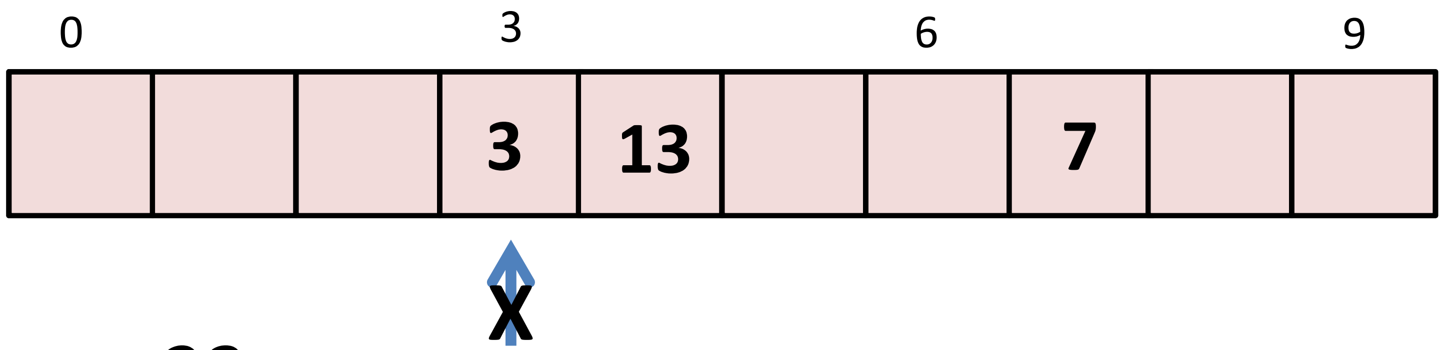
- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1



- Insert 13
  - (13+1)%10=4

Linear Probing

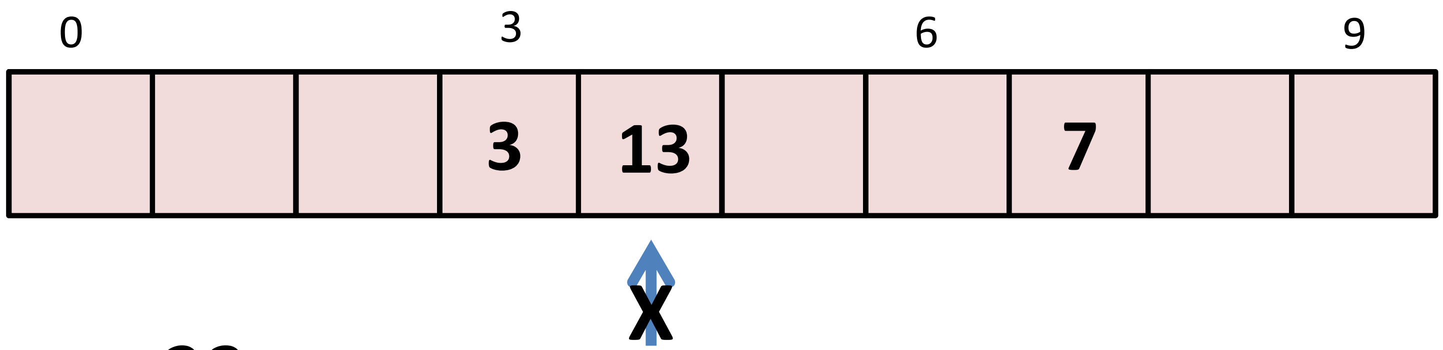
- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1



- Insert 23
  - 23%10=3 → collision & overflow!

Linear Probing

- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1



- Insert 23
  - (23+1)%10=4 → collision & overflow!

Linear Probing

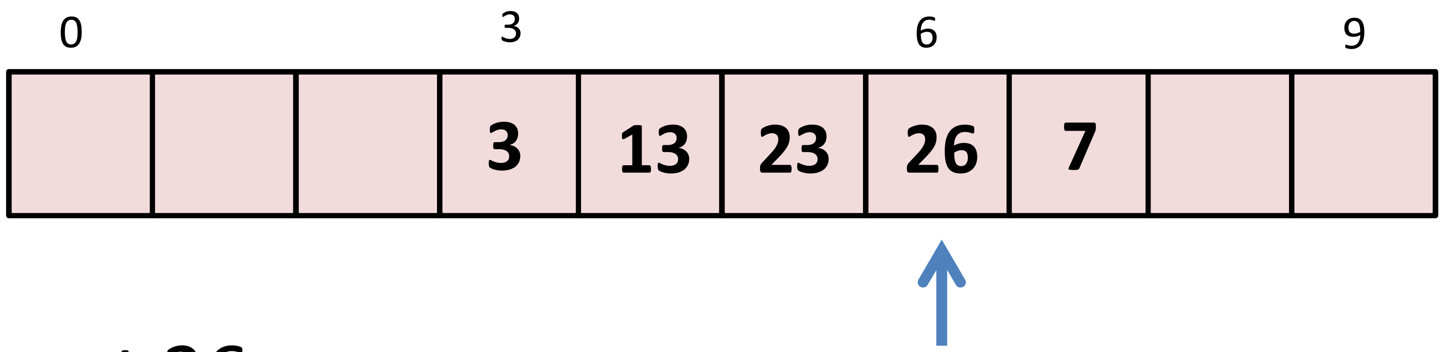
- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1

0			3			6			9
			3	**13**	**23**		**7**		

- Insert 23
  - (23+2)%10=5

Linear Probing

- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1



- Insert 26
  – 26%10=6

Linear Probing

- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1

0			3			6			9
			3	**13**	**23**	**26**	**7**	**36**	

- Insert 36
  - 36%10=6 → collision & overflow!
  - Next available bucket: 8

Linear Probing

- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1

0			3			6			9
			3	**13**	**23**	**26**	**7**	**36**	

- Same color : same hash value group

Linear Probing

- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1

0			3			6			9
			3	**13**	**23**	**26**	**7**	**36**	

- Delete 23
  - –Then, search the right-side cluster if there is a key to shift to left

Linear Probing

- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1

0			3			6			9
			3	**13**		**26**	**7**	**36**	

- Delete 23
  - h(26) = 6, h(7) = 7, h(36) = 8 (due to collision at 6) : no shifting required

Linear Probing

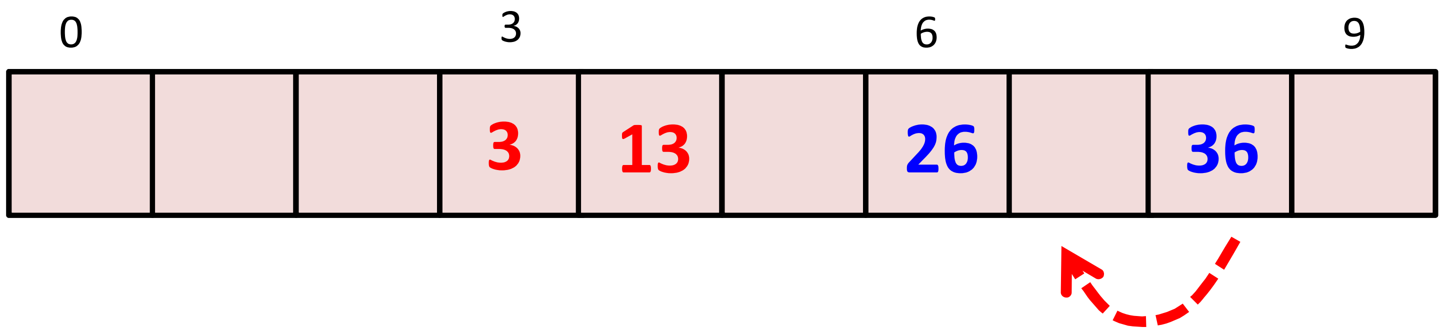
- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1

0			3			6			9
			3	**13**		**26**	**7**	**36**	

- Delete 7
  - Then, search the right-side cluster (which is 36)

Linear Probing

- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1



- Delete 7
  - h(36) = 7 (since 6 is collision and 7 is empty)

Linear Probing

- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1

0			3			6			9
			3	13		26	36		

- Delete 7
  - –Shift 36 to left

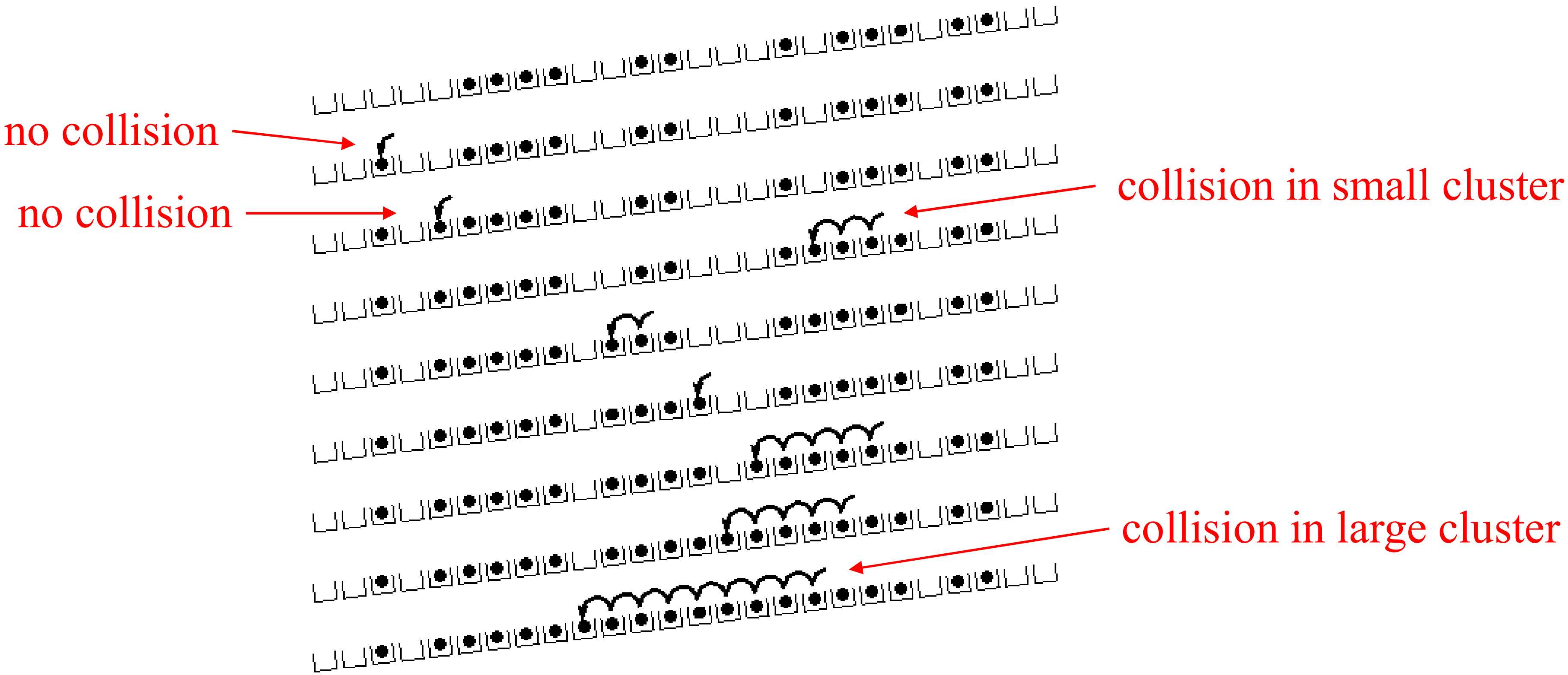
Linear Probing

- Divisor = # of buckets = 10
- h(k) = k % 10
- # of slots = 1

0			3			6			9
			3	**13**		**26**	**D**	**36**	

- Delete without shifting
  - Mark as *deleted*, and a new key can be inserted to that location later (retain cluster)

Linear Probing – Clustering



[R. Sedgewick]

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Performance of Linear Probing

- Load factor $\alpha = \frac{n}{sb}$ is important for performance
  - If α is small, fewer collisions occur
  - If α is large, hash table is filling up, clusters get fewer and larger, and more collisions occur
    - Collision resolution is more costly

- Worst-case search/insert/delete time
  - O(n), when?

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Search Performance

- Expected # of probed for large tables
  - –Successful search: $\frac{1}{2}\left[1+\frac{1}{1-\alpha}\right]$
  - –Unsucessful search: $\frac{1}{2}\left[1+\frac{1}{(1-\alpha)^2}\right]$

α	S_n	U_n
0.50	1.5	2.5
0.75	2.5	8.5
0.90	5.5	50.5

Performance quickly degrades for $\alpha > 0.5$

Discussion about Linear Probing

- Pros
  - Simple to compute
- Cons
  - Clustering: increase the average time to locate keys
  - Worst case O(n) for hash table operations
  - Delete can be more expensive due to shifting
- More random redistribution is desired

Quadratic Probing

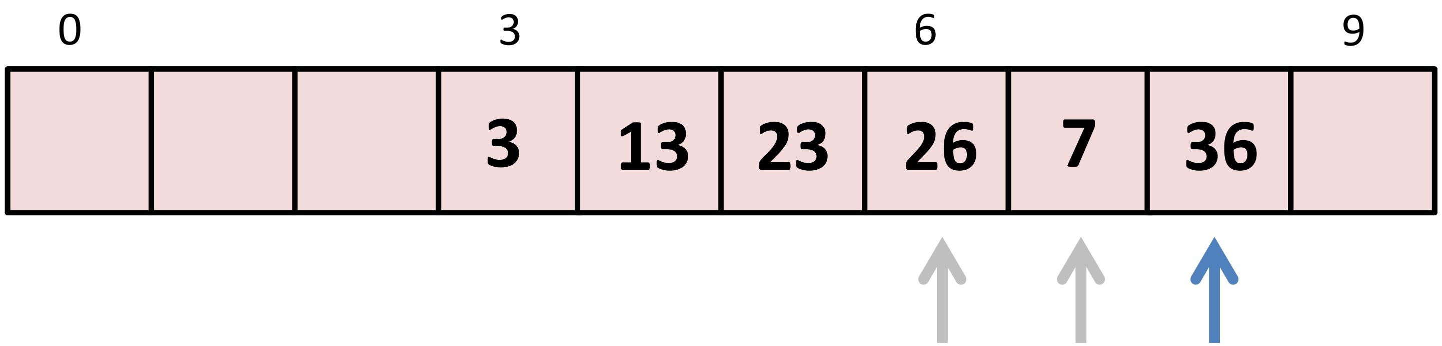
- $ht[(h(k)+j^2)\%b]$ for $j=0, 1, 2, ..., b-1$
- Search the next available bucket from the original address by the distance 1, 4, 9, 16, ...
- Pros
  - Simple calculation, reduce clustering
- Cons
  - Not all buckets can be examined

Other Open Addressing Methods

- Rehashing
  - Use a series of different hash functions $h_1, h_2, \ldots, h_m$
  - $ht[h_j(k)\%b]$ for $j=1, 2, \ldots, m$
  - Minimize clustering
- Random probing
  - $ht[(h(k)+s(i))\%b]$ for $i= 1, 2, \ldots, b-1$
  - s(i) : pseudo random number between 1 to b-1
    - Each number is generated only once

Chaining: Motivation

- Problem of open addressing
  - Unnecessarily compare keys that have different hash values → unnecessarily increase costs



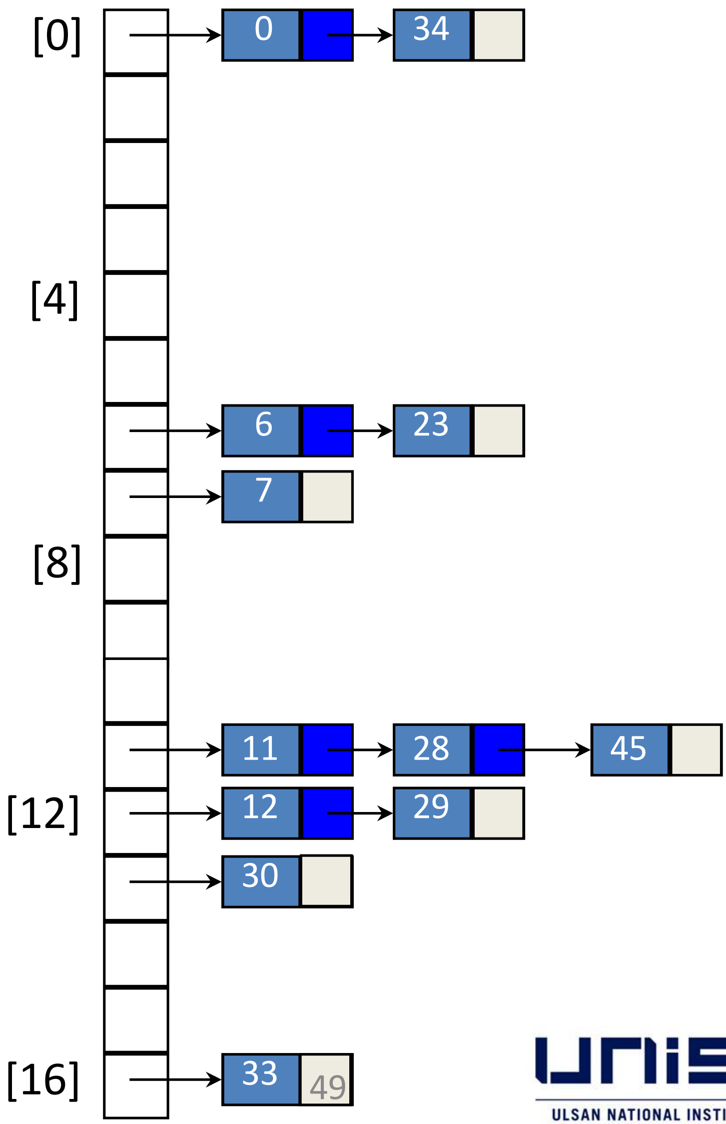
- To find 36, comparing to 26 and 7 is required
  - h(7) != h(36), so this comparison is unnecessary

Chaining

- Linear list per each hash address
  - Chain (singly linked list) is often used
  - Sorted or unsorted
- ht[0:b-1] : has table with b buckets
- ht[i] : point to the first node of the chain for bucket i

Example: Sorted Chain

- Insert 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
- h[k] = k%17



Expected Performance

- Chaining

$$U_n \approx \alpha$$

$$S_n \approx 1 + \frac{\alpha}{2}$$

$$\alpha = \frac{n}{b}: \text{ loading density}$$

α	S_n	U_n
0.50	1.25	0.5
0.75	1.375	0.75
0.90	1.45	0.9

- Less calculation than open addressing
- Extra dynamic memory usage (pointers)

Hash Table Design

- Maximum permissible loading density for given performance requirements
- e.g., linear probing
  - Max comparison for successful search : 10
    - $S_n \sim ½(1 + 1/(1 - \alpha))$
    - $\alpha <= 18/19$
  - Max comparison for unsuccessful search : 13
    - $U_n \sim ½(1 + 1/(1 - \alpha)^2)$
    - $\alpha <= 4/5$
  - Therefore, $\alpha <= \min\{18/19, 4/5\} = 4/5 = 0.8$

Hash Table Design

- Dynamic resizing of table
  - When loading density exceeds threshold
  - Array doubling
  - Rehash old table into new large table (slow)

Questions?