AAI2007 Introduction to Algorithms

Week 6: Hash Table

Instructor: Jinyoung Han (jinyounghan@skku.edu)

Schedule

Tentative Schedule

수업일	내용
9/4	Course Introduction, Algorithm Basic, Level Test
9/11	Order of Complexity, List
9/18	Stack, Queue
9/25	건학 기념일
10/2	Tree, Binary Search Tree (BST)
10/9	Priority Queue, Heap, Heap Sort 한글날
10/16	Hash Table, Searching Revisited
10/23	Graph Basic
10/30	Midterm Exam
11/6	Graph Algorithms
11/13	Sorting
11/20	Dynamic Programming (1)
11/27	Dynamic Programming (2)
12/4	Greedy Algorithms
12/11	Reserved
12/18	Final Exam

In This Lecture

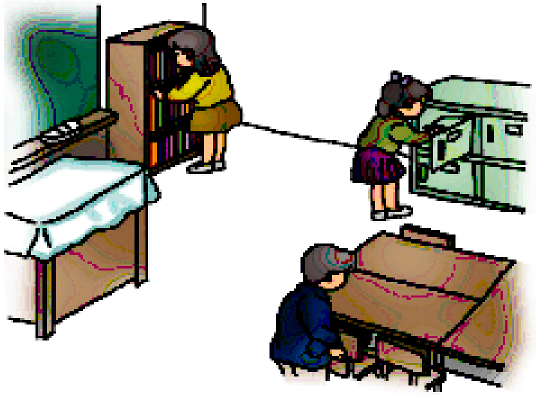
Outline

1. Hashing
2. Collision

01. Hashing

Hashing

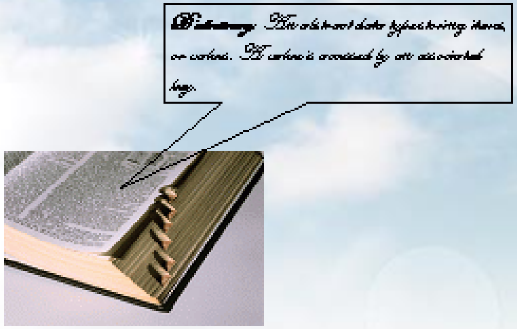
- Hashing
  - Hashing is the process of mapping a search key to a limited range of array indices
  - Cf) most of search algorithms compare keys to access
- Hash table
  - The keys are stored in an array called a **hash table,** and a **hash function** is associated with the table
    - The function converts or maps the search keys to specific entries in the table
- Hashing is similar to organizing and finding stuffs



01. Hashing

Dictionary

- Dictionary
  - Map or table
  - Two fields
    - Search key, such as English word or name
    - Value associating with search key such as word definition or phone number



Dictionary: An abstract data type storing items, or entries. An entry is accessed by an associated key.

Dictionary ADT

·Object: a set of (key, value)

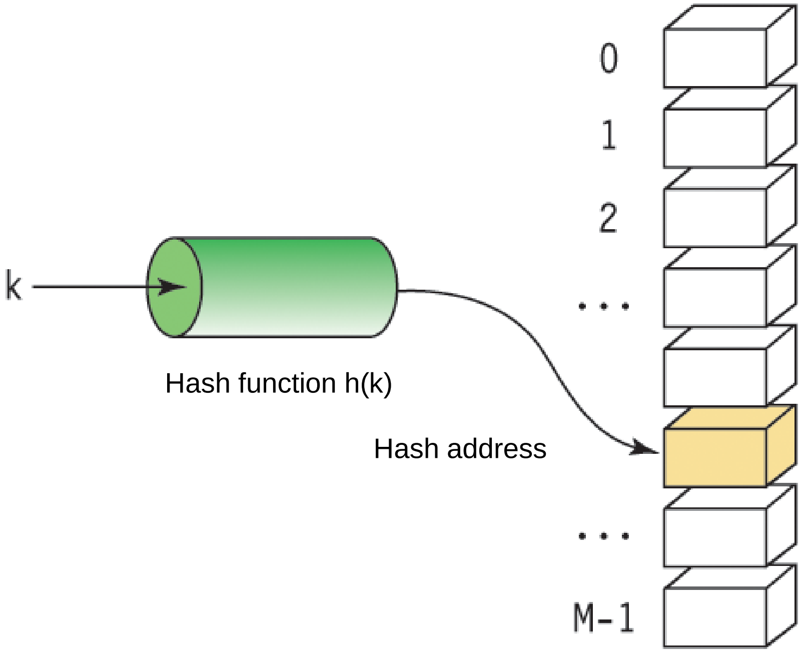
·Operations:

- add(key, value) ::= add a (key, value) pair
- delete(key) ::= delete the (key, value) associating with key, and return the value
- search(key) ::= return value associating with key

01. Hashing

Hash Function

- Hash function
  - Input: search key
  - Hash address: index in a table



Hash function h(k)

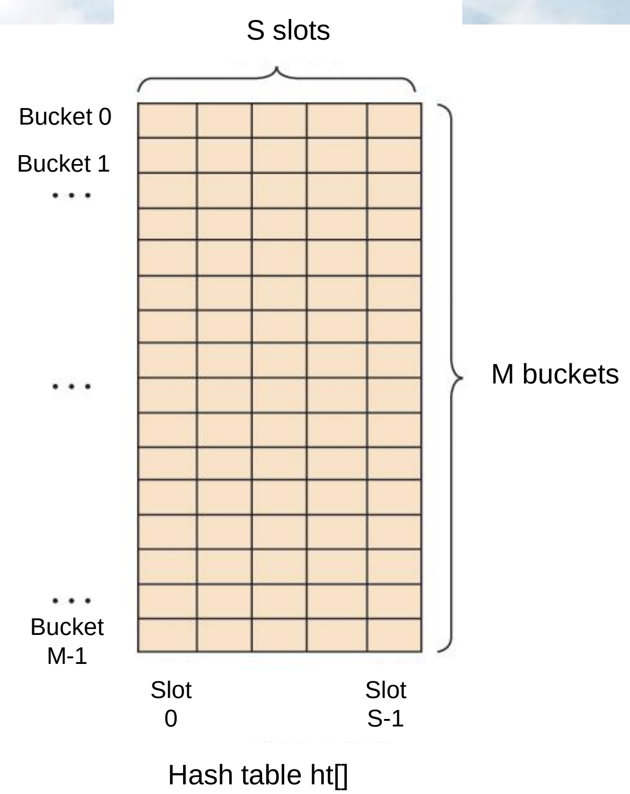
Hash address

Hash table ht[]

01. Hashing

Hash Table

- Hash table HT[]
  - Given a search key, a hash address exists
  - Hash address: index of hash table
- Collision
  - In case of h(k1) = h(k2) for different search keys k1 and k2
- Overflow
  - # collisions > # slots in given bucket
  - Should be addressed

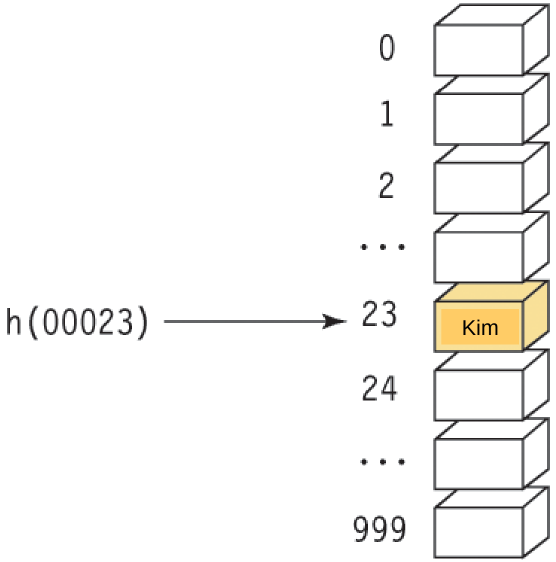


Hash table ht[]

01. Hashing

Hashing Example

- Finding a student's information with hashing
  - The last three digits represent the student id number
    - If 00023 is the student id number, information of the student stores in ht[23]
    - If the hash table has 1000 space, search time takes O(1)!

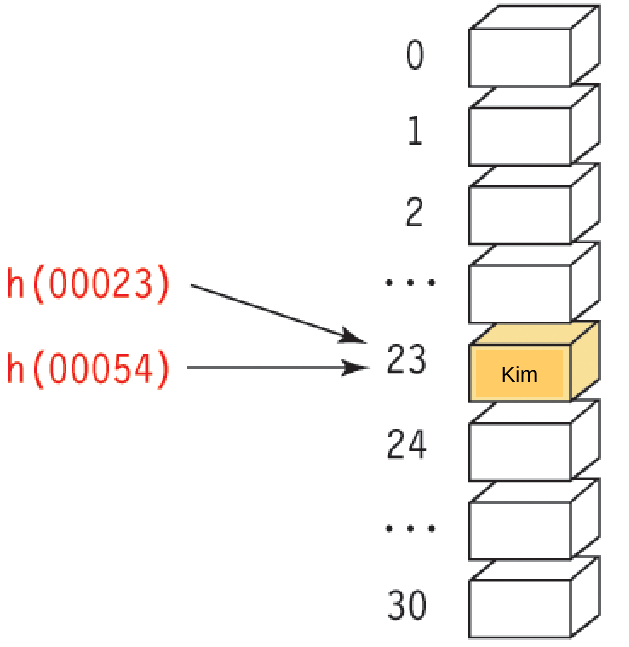


Hash table ht[]

01. Hashing

Hashing Example

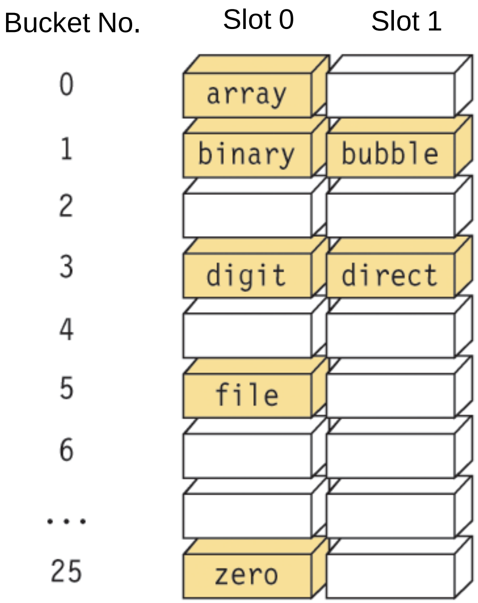
- In practice, the size of hash table is limited, e.g., 30
  - Difficult to assign spaces for all the possible keys
- A well-known hash function, ***h(k)= k mod M***, usually makes collisions and follow-up overflows



Hash table ht[]

Hashing Example

- Another example
  - hash function = the order of the first letter of a string
    - h("array")=1
    - h("binary")=2
  - Input data: array, binary, bubble, file, digit, direct, zero, bucket



Bucket No.	Slot 0	Slot 1
0	array	
1	binary	bubble
2		
3	digit	direct
4		
5	file	
6		
...		
25	zero	

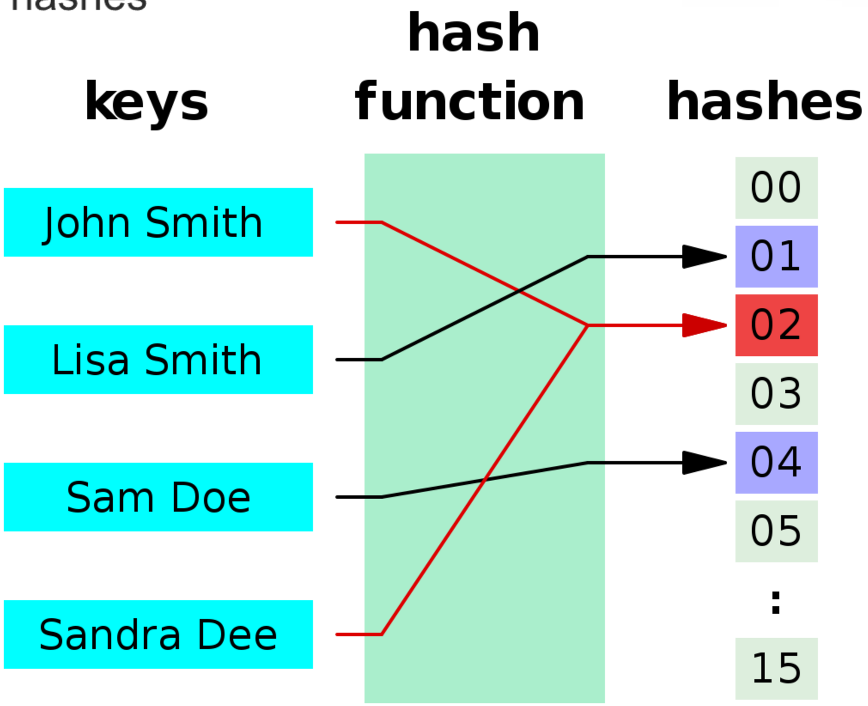
Hash table ht[]

“bucket” cannot be stored due to overflow!

01. Hashing

Hash Function

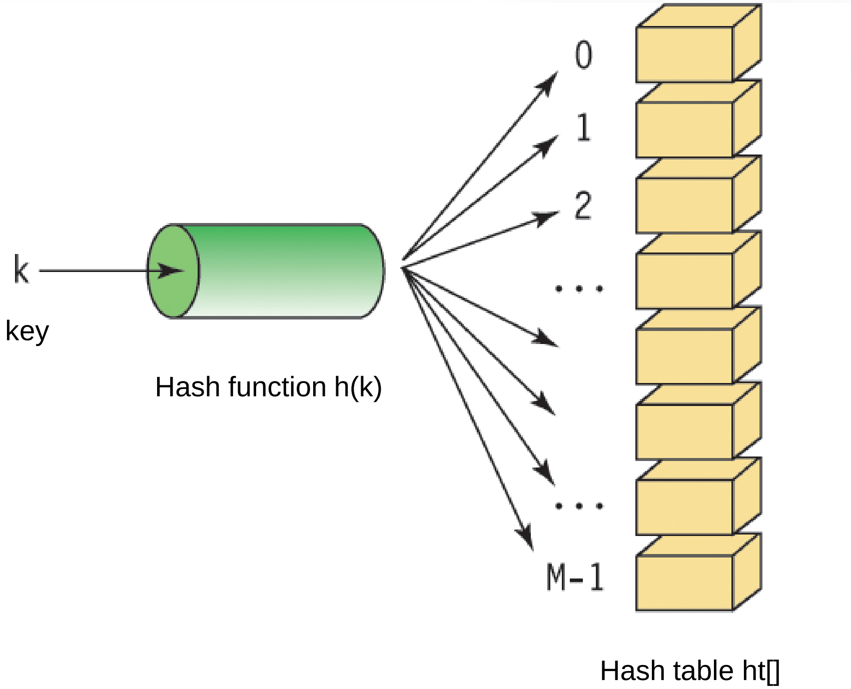
- Hash function
  - A hash function is any function that can be used to map data of arbitrary size to fixed-size values
  - The values returned by a hash function are called hash values, hash codes, digests, or simply hashes



01. Hashing

Hash Function

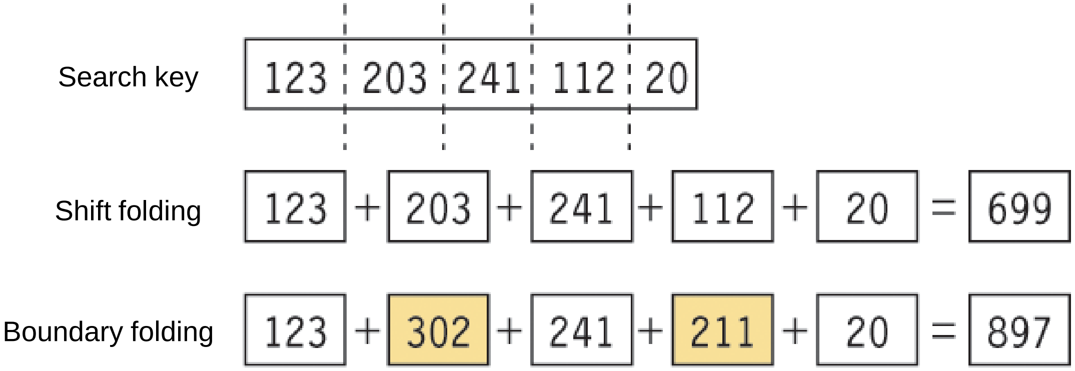
- Good hash function
  - Low collisions
  - Hash values are uniformly distributed across the address area in hash table
  - Low computing time



01. Hashing

Hash Function

- Examples
  - Division function
    - h(k)=k mod M
    - Size of hash table M is usually a prime number
  - Folding function
    - shift folding, boundary folding



Search key 123 203 241 112 20

Shift folding 123 + 203 + 241 + 112 + 20 = 699

Boundary folding 123 + 302 + 241 + 211 + 20 = 897

01. Hashing

Hash Function

- Other examples
  - Mid-square function
    - (key)^2, and get a few bits from it to a generate hash address
  - Bit extraction function
    - Assume the key is a binary number, and select k bits in arbitrary location to generate hash address
  - Digit analysis method
    - Use digits (in a key) that are uniformly distributed to generate hash address
  - Etc.

Collision

02. Collision

Collision

- Collision
  - Two different keys are associated with same hash address
  - If a collision happens, storing in hash table is not possible
  - Handling collision effectively is necessary

Hash table

- Possible solutions
  - Linear probing
    - Stores item (in collision) in the different place in hash table
  - Chaining
    - In each bucket, a linked list is used

02. Collision

Linear Probing

- If a collision happens at ht[k],
  - Check whether ht[k+1] is available
  - If not, check ht[k+2], and so on
  - Check until find the available slot
  - If the check reaches to the bottom of the table, start from the top of the table
  - If the check reaches to the original slot, ht[k], then the table is full
  - Checking positions: h(k), h(k)+1, h(k)+2,…
- Problems: ***Clustering*** and ***Coalescing***

02. Collision

Linear Probing

- Example: h(k) = k mod 7

Step 1 (8) : h(8) = 8 mod 7 = 1 (store)
Step 2 (1) : h(1) = 1 mod 7 = 1 (collision)
(h(1)+1) mod 7 = 2 (store)
Step 3 (9) : h(9) = 9 mod 7 = 2 (collision)
(h(9)+1) mod 7 = 3 (store)
Step 4 (6) : h(6) = 6 mod 7 = 6 (store)
Step 5 (13) : h(13) = 13 mod 7 = 6 (collision)
(h(13)+1) mod 7 = 0(store)

	Step1	Step2	Step3	Step4	Step5
[0]					13
[1]	8	8	8	8	8
[2]		1	1	1	1
[3]			9	9	9
[4]					
[5]					
[6]				6	6

02. Collision

Quadratic Probing

- Similar to linear probing, but the next slot for checking is calculated as follows
  - (h(k)+inc*inc) mod M

- Next slots for checking could be..
  - h(k), h(k)+1, h(k)+4,…

- Can address the clustering and coalescing problems

02. Collision

Double Hashing

- Double hashing
  - If an overflow happens, use another hash function
    - a.k.a. rehashing
  - step=C-(k mod C)
  - h(k), h(k)+step, h(k)+2*step, …

- Example, hash table with size 7
  - 1st hash function: k mod 7
  - An overflow happens
    - step = 5-(k mod 5)

02. Collision

Double Hashing

Step 1 (8) : h(8) = 8 mod 7 = 1 (store)
Step 2 (1) : h(1) = 1 mod 7 = 1 (collision)
(h(1)+h‘(1)) mod 7 = (1+5-(1 mod 5)) mod 7 = 5 (store)
Step 3 (9) : h(9) = 9 mod 7 = 2 (store)
Step 4 (6) : h(6) = 6 mod 7 = 6 (store)
Step 5 (13) : h(13) = 13 mod 7 = 6 (collision)
(h(13)+h‘(13)) mod 7 = (6+5-(13 mod 5)) mod 7= 1 (collision)
(h(13)+2*h‘(13)) mod 7 = (6+2*2) mod 7= 3 (store)

	Step1	Step2	Step3	Step4	Step5
[0]					
[1]	8	8	8	8	8
[2]			9	9	9
[3]					13
[4]					
[5]		1	1	1	1
[6]				6	6

02. Collision

Double Hashing

	Step1	Step2	Step3	Step4	Step5
[0]					
[1]	8	8	8	8	8
[2]			9	9	9
[3]					13
[4]					
[5]		1	1	1	1
[6]				6	6

[double hashing]

vs.

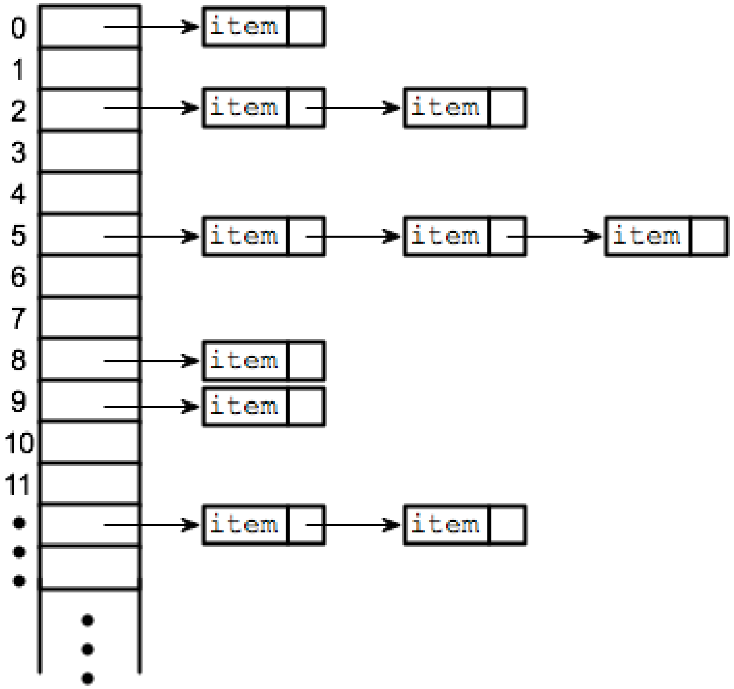
	Step1	Step2	Step3	Step4	Step5
[0]					13
[1]	8	8	8	8	8
[2]		1	1	1	1
[3]			9	9	9
[4]					
[5]					
[6]				6	6

[linear probing]

02. Collision

Chaining

- Addressing the overflow problem using a linked list
  - In each bucket, # slots is not fixed
    - Use a linked list for easy insertion and deletion
  - In a bucket, search a inked list sequentially

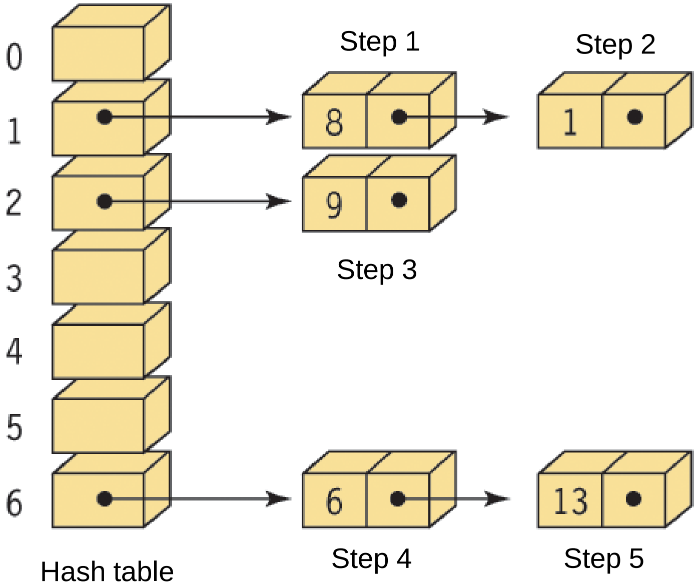


02. Collision

Chaining

- Example
  - Hash table with size 7, h(k)=k mod 7, input (8, 1, 9, 6, 13)

Step 1 (8) : h(8) = 8 mod 7 = 1 (store)
Step 2 (1) : h(1) = 1 mod 7 = 1 (collision->new node)
Step 3 (9) : h(9) = 9 mod 7 = 2 (store)
Step 4 (6) : h(6) = 6 mod 7 = 6 (store)
Step 5 (13) : h(13) = 13 mod 7 = 6 (collision->new node)

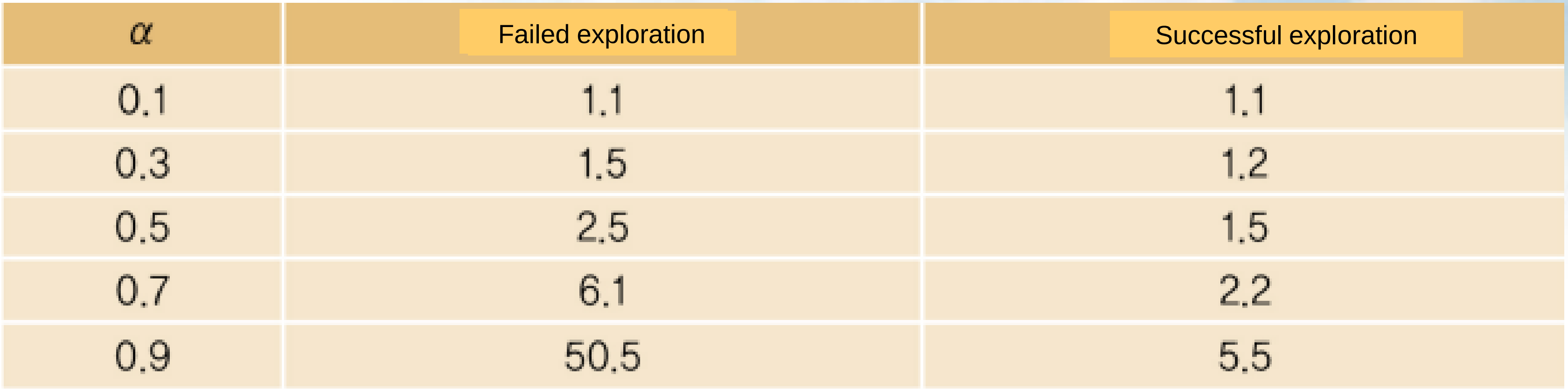


02. Collision

Hashing Performance

- Linear Probing

$$\alpha \text{ (loading density)} = \frac{\#\, stored\ items}{\#\, buckets} = \frac{n}{M}$$



α	Failed exploration	Successful exploration
0.1	1.1	1.1
0.3	1.5	1.2
0.5	2.5	1.5
0.7	6.1	2.2
0.9	50.5	5.5

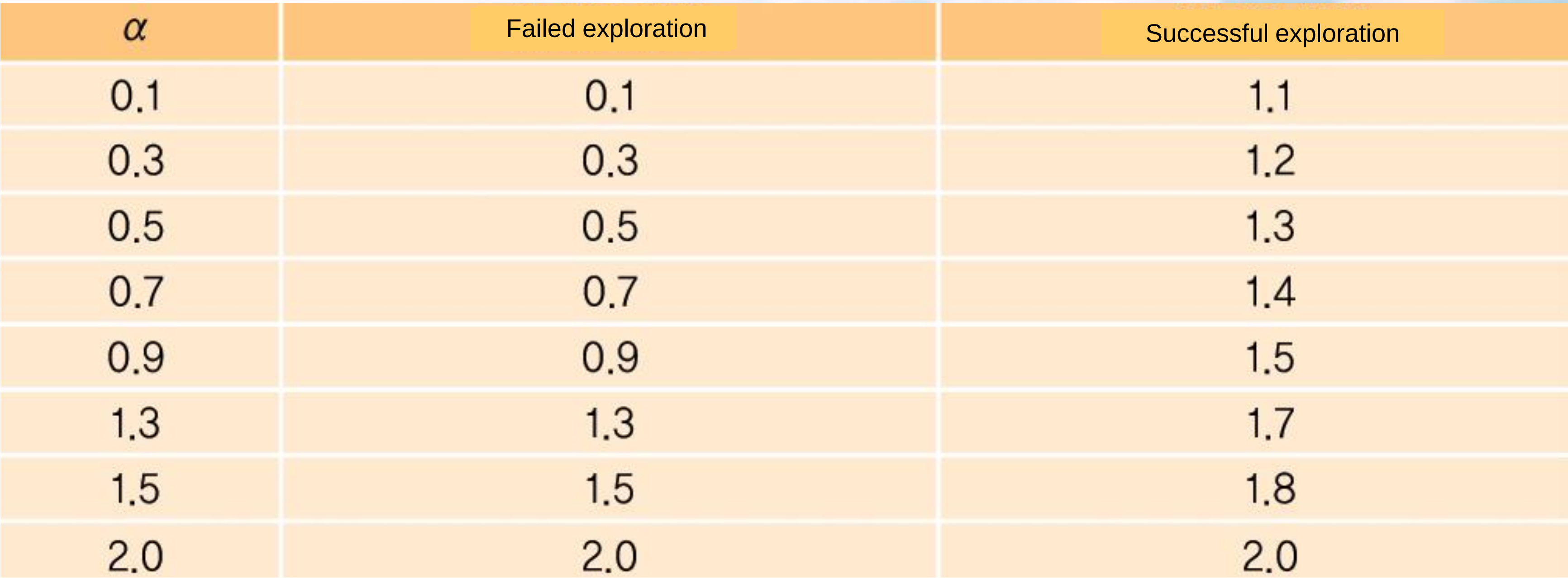
[# comparisons]

02. Collision

Hashing Performance

- Chaining

α (loading density)= $\frac{\#\ stored\ items}{\#\ buckets} = \frac{n}{M}$



α	Failed exploration	Successful exploration
0.1	0.1	1.1
0.3	0.3	1.2
0.5	0.5	1.3
0.7	0.7	1.4
0.9	0.9	1.5
1.3	1.3	1.7
1.5	1.5	1.8
2.0	2.0	2.0

[# comparisons]

What You Need to Know

Summary

- Hashing
  - Hash table, hash function, collision, overflow
- Addressing collision
  - Linear probing
  - Chaining

Thanks

Week 6: Hash Table
Instructor: Jinyoung Han (jinyounghan@skku.edu)